Project Report on

Real-time Stock Performance Analysis and Forecasting of EATON and TVS MOTOR using Neural Network

by

Dhiraj Dharmadip Raut

Table of Contents

Introduction

A stock (also known as equity) is a security that represents the ownership of a fraction of a corporation. This entitles the owner of the stock to a proportion of the corporation's assets and profits equal to how much stock they own. Units of stock are called "shares"

An investor can buy a stock and sell it later. If the stock price increases, the investor profits, If it decreases,the investor with incur a loss. Determining the stock price is complex; it depends on the number of outstanding shares, the size of the company's future profits, revenue, losses, company news, public perception, and much more. But, An essential factor is the company's growth of profit/revenue. People trade stocks throughout the day the stock ticker is a report of the price of a certain stock, updated continuously throughout the trading session by the various stock market exchanges.

As a Data Scientist/Analyist working for an investment firm that helps customers invest their money in stocks, It is necessary part to extract essential financial data like **historical share price** and **quarterly revenue** reportings, of a company or an organisation, from a dataset and display it to visualise, so that individuals can make correct decisions based on the data.

In the project, I will extract stock price data and revenue data for Eaton Corporation and TVS Motor from various sources using Python libraries and webscraping. Both Process will be from different webpages and with diffrent coding approach. At last, will display this data in a graph, which can be also called as a simple Dashboard, to visualize and compare the price of the stock vs the revenue by moving a RangeSlider as our need, to identify patterns or trends and to see if there is a correlation between the two.

Import the Requied Modules and the Functions

Install Python libraries required for this project, which are BeautifulSoup4, lxml, html5, requests, yfinance, pandas, and plotly.

Once Installed all libraries, import all Modules and Functions requied from these libraries.

In [1]:
```
# pip install bs4
#!pip install lxml
#!pip install html5lib
#!pip install requests
#!pip install yfinance
#!pip install pandas
#!pip install plotly
```

In [2]:
```
import tensorflow as tf
import yfinance as yf
import pandas as pd
import numpy as np
import requests
from bs4 import BeautifulSoup
import matplotlib.pyplot as plt
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import warnings
warnings.filterwarnings('ignore')
```

Extract EATON Corporation Stock Data

yfinance python Library is used for extracting Stock data of Tesla. Using the `Ticker` function of the yfinance, we need to enter the ticker symbol of the stock we want to extract data on to create a ticker object.

The stock is Eaton_Corporation_PLC and its ticker symbol is `ETN` .

In [3]:
```
eaton = yf.Ticker("ETN")
```

Using this ticker object eaton and using the function `history` , share price of the stock extracted and saved it in a dataframe named `eaton_stock` .

A share is the single smallest part of a company's stock that you can buy, the prices of these shares fluctuate over time. Using the `history()` method we can get the share price of the stock over a certain period of time. Using the `period` parameter we can set how far back from the present to get data. The options for `period` are 1 day (1d), 5d, 1 month (1mo) , 3mo, 6mo, 1 year (1y), 2y, 5y, 10y, ytd, and max. The format that the data is returned in is a Pandas DataFrame.

Here, The `period` parameter has set to `max` to get information for the maximum amount of time. Displayed the complete stock data extracted.

In [4]:
```
eaton_stock = eaton.history(period="max")
display(eaton_stock)
```

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
1972-06-01	0.000000	0.366865	0.362025	0.364929	82978	0.0	0.0
1972-06-02	0.363960	0.363960	0.360088	0.361056	120318	0.0	0.0
1972-06-05	0.359121	0.359121	0.353313	0.354281	107871	0.0	0.0
1972-06-06	0.354281	0.356217	0.350409	0.352345	327763	0.0	0.0
1972-06-07	0.351377	0.351377	0.349441	0.349441	43563	0.0	0.0
...	...	...	...	...	...	...	...
2022-08-29	139.940002	142.009995	139.229996	140.710007	1223900	0.0	0.0
2022-08-30	142.380005	142.589996	138.139999	139.240005	1153700	0.0	0.0
2022-08-31	139.940002	139.940002	136.100006	136.639999	2249900	0.0	0.0
2022-09-01	135.779999	138.839996	135.169998	138.770004	1538900	0.0	0.0
2022-09-02	140.479996	140.690002	136.869995	137.300003	1080200	0.0	0.0

12675 rows × 7 columns

Now, **Reset the index** of the eaton_data DataFrame with the `reset_index` function and Set the `inplace` paramter to `True` so the change takes place to the DataFrame itself.

The last five rows of the `eaton_stock` dataframe are displayed using the `tail` function. For the first five rows, use `head` function.

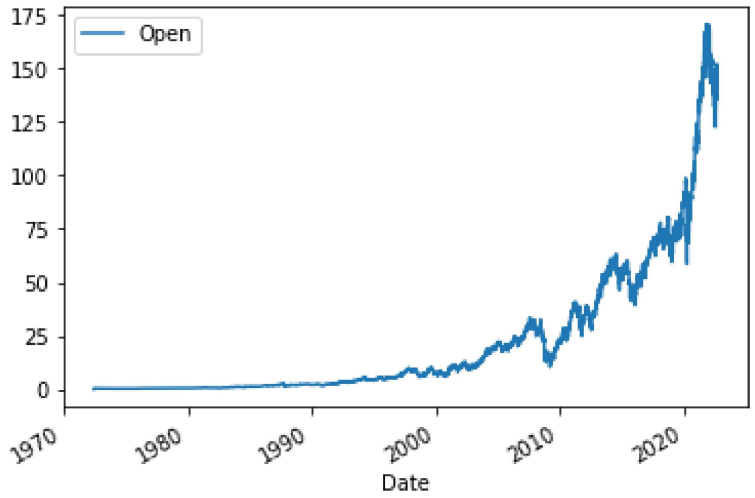
In [5]:
```
eaton_stock.reset_index(inplace=True)
display(eaton_stock.tail())
```

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
12670	2022-08-29	139.940002	142.009995	139.229996	140.710007	1223900	0.0	0.0
12671	2022-08-30	142.380005	142.589996	138.139999	139.240005	1153700	0.0	0.0
12672	2022-08-31	139.940002	139.940002	136.100006	136.639999	2249900	0.0	0.0
12673	2022-09-01	135.779999	138.839996	135.169998	138.770004	1538900	0.0	0.0
12674	2022-09-02	140.479996	140.690002	136.869995	137.300003	1080200	0.0	0.0

can plot the `Open` price against the `Date` . Similary, we can plot required data on the y-axis against another data on the x-axis.

In [6]:
```
#!pip install matplotlib
display(eaton_stock.plot(x="Date", y="Open"))
```

```
<AxesSubplot:xlabel='Date'>
```

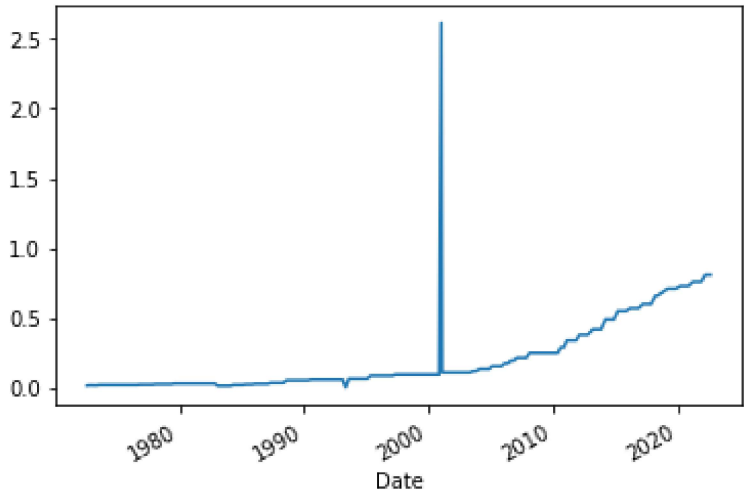


We can visualize **Divendends** as well.

Dividends are the distribution of a companys profits to shareholders. In this case they are defined as an amount of money returned per share an investor owns. Using the variable `dividends` we can get a dataframe of the data. The period of the data is given by the period defined in the 'history` function.

In [7]:
```
display(eaton.dividends.plot())
```

```
<AxesSubplot:xlabel='Date'>
```



Extract EATON Corporation Revenue Data using WebScrapping

Used the `requests` library and downloaded the webpage https://www.macrotrends.net/stocks/charts/ETN/eaton/revenue (https://www.macrotrends.net/stocks/charts/ETN/eaton/revenue).

Stored it as a string in the variable url. Saved the text of the response as a variable named `eaton_html_data` .

In [8]:
```
url = "https://www.macrotrends.net/stocks/charts/ETN/eaton/revenue"
eaton_html_data = requests.get(url).text
```

Now, Parse the eaton_html_data using `beautiful_soup` .

BeautifulSoup is a Python library for pulling data out of HTML and XML files. This is accomplished by representing the HTML as a set of objects with methods used to parse the HTML. We can navigate the HTML as a tree and filter out what we are looking for.

To parse a document, pass it into the `BeautifulSoup` constructor, the `BeautifulSoup` object, which represents the document as a nested data structure.

In [9]:
```
soup = BeautifulSoup(eaton_html_data,"html.parser")
```

First, the document is converted to Unicode, (similar to ASCII), and HTML entities are converted to Unicode characters. Beautiful Soup transforms a complex HTML document into a complex tree of Python objects.

We can use the method `prettify()` to display the HTML in the nested structure.

In [10]:
```
#print(soup.prettify())
```

Now, Used `BeautifulSoup` object to extract the table named `Eaton Quarterly Revenue` available on the above mentioned url and stored it into a dataframe named `eaton_revenue` .

The dataframe have columns `Date` and `Revenue` .

The same Process can be done using `read_html` function of pandas python library.

In [11]:
```
eaton_revenue = pd.DataFrame(columns=["Date","Revenue"])

# First isolate the body of the table which contains all the information
# Then loop through each row and find all the column values for each row
for row in soup.find_all("tbody")[1].find_all("tr"):
    col = row.find_all("td")
    if (col!=[]):
        date = col[0].text
        revenue = col[1].text
        # Finally append the data of each row to the table
        eaton_revenue = eaton_revenue.append({"Date":date, "Revenue":revenue},ignore_index=True)

display(eaton_revenue)
```

	Date	Revenue
0	2022-06-30	$5,212
1	2022-03-31	$4,843
2	2021-12-31	$4,798
3	2021-09-30	$4,923
4	2021-06-30	$5,215
5	2021-03-31	$4,692
6	2020-12-31	$4,687
7	2020-09-30	$4,526
8	2020-06-30	$3,856
9	2020-03-31	$4,789
10	2019-12-31	$5,238
11	2019-09-30	$5,314
12	2019-06-30	$5,533
13	2019-03-31	$5,305
14	2018-12-31	$5,459
15	2018-09-30	$5,412
16	2018-06-30	$5,487
17	2018-03-31	$5,251

	Date	Revenue
18	2017-12-31	$5,213
19	2017-09-30	$5,211
20	2017-06-30	$5,132
21	2017-03-31	$4,848
22	2016-12-31	$4,867
23	2016-09-30	$4,987
24	2016-06-30	$5,080
25	2016-03-31	$4,813
26	2015-12-31	$5,057
27	2015-09-30	$5,203
28	2015-06-30	$5,372
29	2015-03-31	$5,223
30	2014-12-31	$5,565
31	2014-09-30	$5,728
32	2014-06-30	$5,767
33	2014-03-31	$5,492
34	2013-12-31	$5,527
35	2013-09-30	$5,607
36	2013-06-30	$5,602
37	2013-03-31	$5,310
38	2012-12-31	$4,333
39	2012-09-30	$3,950
40	2012-06-30	$4,068
41	2012-03-31	$3,960
42	2011-12-31	$4,033
43	2011-09-30	$4,123
44	2011-06-30	$4,090
45	2011-03-31	$3,803

	Date	Revenue
46	2010-12-31	$3,663
47	2010-09-30	$3,571
48	2010-06-30	$3,378
49	2010-03-31	$3,103
50	2009-12-31	$3,131
51	2009-09-30	$3,028
52	2009-06-30	$2,901
53	2009-03-31	$2,813

Need to remove the comma and dollar sign from the `Revenue` column.

In [12]:
```
warnings.filterwarnings('ignore')
eaton_revenue["Revenue"] = eaton_revenue['Revenue'].str.replace(',|\$',"")
```

Now, Need to remove an null or empty strings in the Revenue column.

In [13]:
```
eaton_revenue.dropna(inplace=True)
eaton_revenue = eaton_revenue[eaton_revenue['Revenue'] != ""]
```

Now, Displayed the First 5 rows of the `eaton_revenue` dataframe using the `head` function. For Last 5 rows, use `tail` function.

In [14]:
```
display(eaton_revenue.head())
```

	Date	Revenue
0	2022-06-30	5212
1	2022-03-31	4843
2	2021-12-31	4798
3	2021-09-30	4923
4	2021-06-30	5215

Extract TVS MOTOR Stock Data using WebScrapping

Downloaded the webpage https://in.investing.com/equities/tvs-motor-company-historical-data?end_date=1662057000&st_date=1078857000 (https://in.investing.com/equities/tvs-motor-company-historical-data?end_date=1662057000&st_date=1078857000).

Note: Update this webpage after every day for live data

Stored it as a string in the variable `tvs_stock_url`. Saved the text of the response as a variable named `tvs_stock_html_data`.

In [15]:
```
tvs_stock_url = "https://in.investing.com/equities/tvs-motor-company-historical-data?end_date=1662057000&st_date=1
tvs_stock_html_data = requests.get(tvs_stock_url).text
```

Now, Parse the tvs_stock_html_data using `beautiful_soup`.

In [16]:
```
soup = BeautifulSoup(tvs_stock_html_data,"html.parser")
```

Beautiful Soup uses the `NavigableString` class to contain this text.

Here, Name of the complete data is obtained by extracting the string of the `Tag`

In [17]:
```
soup.title.string
```

Out[17]:
```
'TVS Motor Company (TVSM) Historical Prices - Investing.com India'
```

Now, need to find all HTML tables in the web page. In html, table is represented by the tag `<table>`.

We can see how many tables were found by checking the length of the tables list

In [18]:
```
tables = soup.find_all('table')
len(tables)
```

Out[18]:
```
5
```

After Going through all indices for tables, `table[1]` contains the required information of stock data.

Now, need to convert all the content into the pandas DataFrame and name it as `tvs_stock`.

In [19]:
```
tvs_stock = pd.DataFrame(columns=["Date", "Open", "High", "Low", "Close", "Volume"])

# First we isolate the body of the table which contains all the information
# Then we loop through each row and find all the column values for each row
for row in tables[1].tbody.find_all('tr'):
    col = row.find_all("td")
    if (col != []):
        date = col[0].text.strip()
        close = col[1].text.strip()
        Open = col[2].text.strip()
        high = col[3].text.strip()
        low = col[4].text.strip()
        volume = col[5].text.strip()
        # Finally we append the data of each row to the table
        tvs_stock = tvs_stock.append({"Date":date, "Open":Open, "High":high, "Low":low, "Close":close, "Volume":vo

display(tvs_stock)
```

	Date	Open	High	Low	Close	Volume
0	Sep 01, 2022	983.90	1,028.80	978.00	1,015.10	3.39M
1	Aug 30, 2022	964.00	989.75	962.70	985.60	1.06M
2	Aug 29, 2022	935.00	960.00	934.50	957.85	680.20K
3	Aug 26, 2022	958.80	967.25	951.55	954.45	1.28M
4	Aug 25, 2022	964.95	968.00	945.90	949.10	1.54M
...	...	...	...	...	...	...
4584	Mar 16, 2004	44.25	44.98	42.00	42.27	103.84K
4585	Mar 15, 2004	45.25	46.00	44.50	44.70	66.44K
4586	Mar 12, 2004	47.00	47.00	44.75	45.77	82.13K
4587	Mar 11, 2004	47.00	47.50	46.12	46.33	42.85K
4588	Mar 10, 2004	48.00	48.23	47.00	47.42	52.31K

4589 rows × 6 columns

Remove an null or empty strings in the Revenue column.

In [20]:
```
tvs_stock.dropna(inplace=True)
tvs_stock = tvs_stock[tvs_stock['Volume'] != ""]
```

Before Proceeding Ahead, we need to remove unnecessary string in Data columns which are `\nE` , `\nD` , `\nS` .

In [21]:
```
# tvs_stock = tvs_stock.loc[::-1]
# tvs_stock = tvs_stock.reset_index(drop=True)
tvs_stock["Date"] = tvs_stock['Date'].str.replace('\nE',"")
tvs_stock["Date"] = tvs_stock['Date'].str.replace('\nD',"")
tvs_stock["Date"] = tvs_stock['Date'].str.replace('\nS',"")

for i in ["Open","High","Low","Close","Volume"]:
    tvs_stock[i] = tvs_stock[i].str.replace(',',"")
```

Extract TVS MOTOR Revenue Data using WebScrapping

Downloaded the webpage from https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/1/Standalone (https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/1/Standalone) to https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/10/Standalone (https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/10/Standalone) serial wise from `1/standalone` to `10/standalone` .

Note: Update this webpages after every 3 months for live data

Stored it as a string in the variable url. It means we need to download `10 webpages` .

Saved the text of the response as a variable named `tvs_revenue_html_data` .

In [22]:
```
url1 = "https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/1/Standalone"
tvs_revenue_html_data1 = requests.get(url1).text
```

Now, Parse the tvs_revenue_html_data using `beautiful_soup` .

First, the document is converted to Unicode, (similar to ASCII), and HTML entities are converted to Unicode characters.
Beautiful Soup transforms a complex HTML document into a complex tree of Python objects.

In [23]:
```
soup1 = BeautifulSoup(tvs_revenue_html_data1,"html.parser")
```

Now, find all `<tables>` tag available in the HTML. Use proper Index to get required table of data.

In [24]:
```
tables1 = soup1.find_all('table')
display(len(tables1))
```

```
14
```

After Going through all indices for tables, `tables1[9]` contains the required information of revenue data.

Now, Before converting all the content into the pandas DataFrame, we need to convert data into required order.

In [25]:
```
requiredtable1 = tables1[9].find_all('tr')[0:2]
display(requiredtable1)
```

```
[<tr>
 <td class="tdh tdL">Figures in Rs crore</td>
 <td class="tdh">Jun-2022</td>
 <td class="tdh">Mar-2022</td>
 <td class="tdh">Dec-2021</td>
 <td class="tdh">Sep-2021</td>
 <td class="tdh">Jun-2021</td>
 </tr>,
 <tr>
 <td class="tdL wD200" colspan="0">Revenue</td>
 <td class="">6008.71</td>
 <td class="">5530.31</td>
 <td class="">5706.43</td>
 <td class="">5619.41</td>
 <td class="">3934.36</td>
 </tr>]
```

`Tanspose` of the content using pandas DataFrame is must, as it will arrange data in the requied order.

Name the DataFrame columns as `Date` and `Revenue` .

In [26]:
```
requiredtable1 = pd.DataFrame(requiredtable1).T

requiredtable1.columns=["Date","Revenue"]

display(requiredtable1)
```

	Date	Revenue
0	\n	\n
1	[Figures in Rs crore]	[Revenue]
2	\n	\n
3	[Jun-2022]	[6008.71]
4	\n	\n
5	[Mar-2022]	[5530.31]
6	\n	\n
7	[Dec-2021]	[5706.43]
8	\n	\n
9	[Sep-2021]	[5619.41]
10	\n	\n
11	[Jun-2021]	[3934.36]
12	\n	\n

As you can see output, `requiredtable1` dataframe contains unnecessary rows with `\n`.
Such unnecessary rows are removed.

In [27]:
```
requiredtable1.dropna(inplace=True)
requiredtable1 = requiredtable1[requiredtable1['Revenue'] != "\n"]
display(requiredtable1)
```

	Date	Revenue
1	[Figures in Rs crore]	[Revenue]
3	[Jun-2022]	[6008.71]
5	[Mar-2022]	[5530.31]
7	[Dec-2021]	[5706.43]
9	[Sep-2021]	[5619.41]
11	[Jun-2021]	[3934.36]

Here, in output, first row is not required and need to be avoided.

`Index` of the DataFrame also need to be reset starting from 0.

In [28]:
```
requiredtable1 = requiredtable1.iloc[1:6]

requiredtable1 = requiredtable1.reset_index(drop=True)

display(requiredtable1)
```

	Date	Revenue
0	[Jun-2022]	[6008.71]
1	[Mar-2022]	[5530.31]
2	[Dec-2021]	[5706.43]
3	[Sep-2021]	[5619.41]
4	[Jun-2021]	[3934.36]

In [29]:
```
display(requiredtable1.iloc[0,1])
```

```
<td class="">6008.71</td>
```

As you can see above, Content saved in the DataFrame is HTML. It is important to extract content from this HTML tag and save it at the same location of the DataFrame. Used `for loop` to perform this operation.

In [30]:
```
for i in range(5):
    for j in range(2):
        requiredtable1.iloc[i,j] = requiredtable1.iloc[i,j].text.strip()

display(requiredtable1.iloc[0,1])
```

```
'6008.71'
```

Remove an null or empty strings in the Revenue column.

In [31]:
```
requiredtable1 = requiredtable1[requiredtable1['Revenue'] != ""]
```

First, stored data of requiredtable1 DataFrame into the `tvs_revenue` DataFrame

In [32]:
```
tvs_revenue = requiredtable1
display(tvs_revenue)
```

	Date	Revenue
0	Jun-2022	6008.71
1	Mar-2022	5530.31
2	Dec-2021	5706.43
3	Sep-2021	5619.41
4	Jun-2021	3934.36

Now, We need to repeat above same steps for extracting data from Webpages2,3,.....10 and at last append all of them in the DataFrame named 'tvs_revenue'.

Create a `list` of URLs that includes url of all remaining webpages. Use `for loop` to go through all the URLs.

In [33]:
```
URLs = ["https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/2/Standalone",
        "https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/3/Standalone",
        "https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/4/Standalone",
        "https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/5/Standalone",
        "https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/6/Standalone",
        "https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/7/Standalone",
        "https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/8/Standalone",
        "https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/9/Standalone",
        "https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/10/Standalone",
       ]

for url in URLs:
    print(url)
```

```
https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/2/Standalone (https://www.busines
s-standard.com/company/tvs-motor-co-8029/financials-quaterly/2/Standalone)
https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/3/Standalone (https://www.busines
s-standard.com/company/tvs-motor-co-8029/financials-quaterly/3/Standalone)
https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/4/Standalone (https://www.busines
s-standard.com/company/tvs-motor-co-8029/financials-quaterly/4/Standalone)
https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/5/Standalone (https://www.busines
s-standard.com/company/tvs-motor-co-8029/financials-quaterly/5/Standalone)
https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/6/Standalone (https://www.busines
s-standard.com/company/tvs-motor-co-8029/financials-quaterly/6/Standalone)
https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/7/Standalone (https://www.busines
s-standard.com/company/tvs-motor-co-8029/financials-quaterly/7/Standalone)
https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/8/Standalone (https://www.busines
s-standard.com/company/tvs-motor-co-8029/financials-quaterly/8/Standalone)
https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/9/Standalone (https://www.busines
s-standard.com/company/tvs-motor-co-8029/financials-quaterly/9/Standalone)
https://www.business-standard.com/company/tvs-motor-co-8029/financials-quaterly/10/Standalone (https://www.busine
ss-standard.com/company/tvs-motor-co-8029/financials-quaterly/10/Standalone)
```

Now, **Append** all the rows of `requiredtable` DataFrame into `tvs_revenue` DataFrame and Update it.

Need to `reset_index` after append of data.

In [34]:
```
for url in URLs:
    tvs_revenue_html_data = requests.get(url).text

    soup = BeautifulSoup(tvs_revenue_html_data,"html.parser")
    tables = soup.find_all('table')
    requiredtable = tables[9].find_all('tr')[0:2]

    requiredtable = pd.DataFrame(requiredtable).T
    requiredtable.columns=["Date","Revenue"]

    requiredtable.dropna(inplace=True)
    requiredtable = requiredtable[requiredtable['Revenue'] != "\n"]

    requiredtable = requiredtable.iloc[1:6]
    requiredtable = requiredtable.reset_index(drop=True)

    for i in range(0,len(requiredtable)):
        for j in range(0,len(requiredtable.columns)):
            requiredtable.iloc[i,j] = requiredtable.iloc[i,j].text.strip()

    requiredtable = requiredtable[requiredtable['Revenue'] != ""]

    tvs_revenue = tvs_revenue.append(requiredtable)

    tvs_revenue = tvs_revenue.reset_index(drop=True)
    tvs_revenue = tvs_revenue[tvs_revenue['Revenue'] != ""]

    tvs_revenue["Date"] = tvs_revenue['Date'].str.replace('-',", ")

display(tvs_revenue)
```

	Date	Revenue
0	Jun, 2022	6008.71
1	Mar, 2022	5530.31
2	Dec, 2021	5706.43
3	Sep, 2021	5619.41
4	Jun, 2021	3934.36
5	Mar, 2021	5321.93
6	Dec, 2020	5391.39

	Date	Revenue
7	Sep, 2020	4605.49
8	Jun, 2020	1431.73
9	Mar, 2020	3481.42
10	Dec, 2019	4125.46
11	Sep, 2019	4347.84
12	Jun, 2019	4468.62
13	Mar, 2019	4384.02
14	Dec, 2018	4663.98
15	Sep, 2018	4993.47
16	Jun, 2018	4168.45
17	Mar, 2018	4007.24
18	Dec, 2017	3698.67
19	Sep, 2017	4064.72
20	Jun, 2017	3399.51
21	Mar, 2017	2844.50
22	Dec, 2016	2983.38
23	Sep, 2016	3426.49
24	Jun, 2016	2880.94
25	Mar, 2016	2797.31
26	Dec, 2015	2895.90
27	Sep, 2015	2837.36
28	Jun, 2015	2574.09
29	Mar, 2015	2443.42
30	Dec, 2014	2639.26
31	Sep, 2014	2667.44
32	Jun, 2014	2305.39
33	Mar, 2014	2159.79
34	Dec, 2013	2057.60

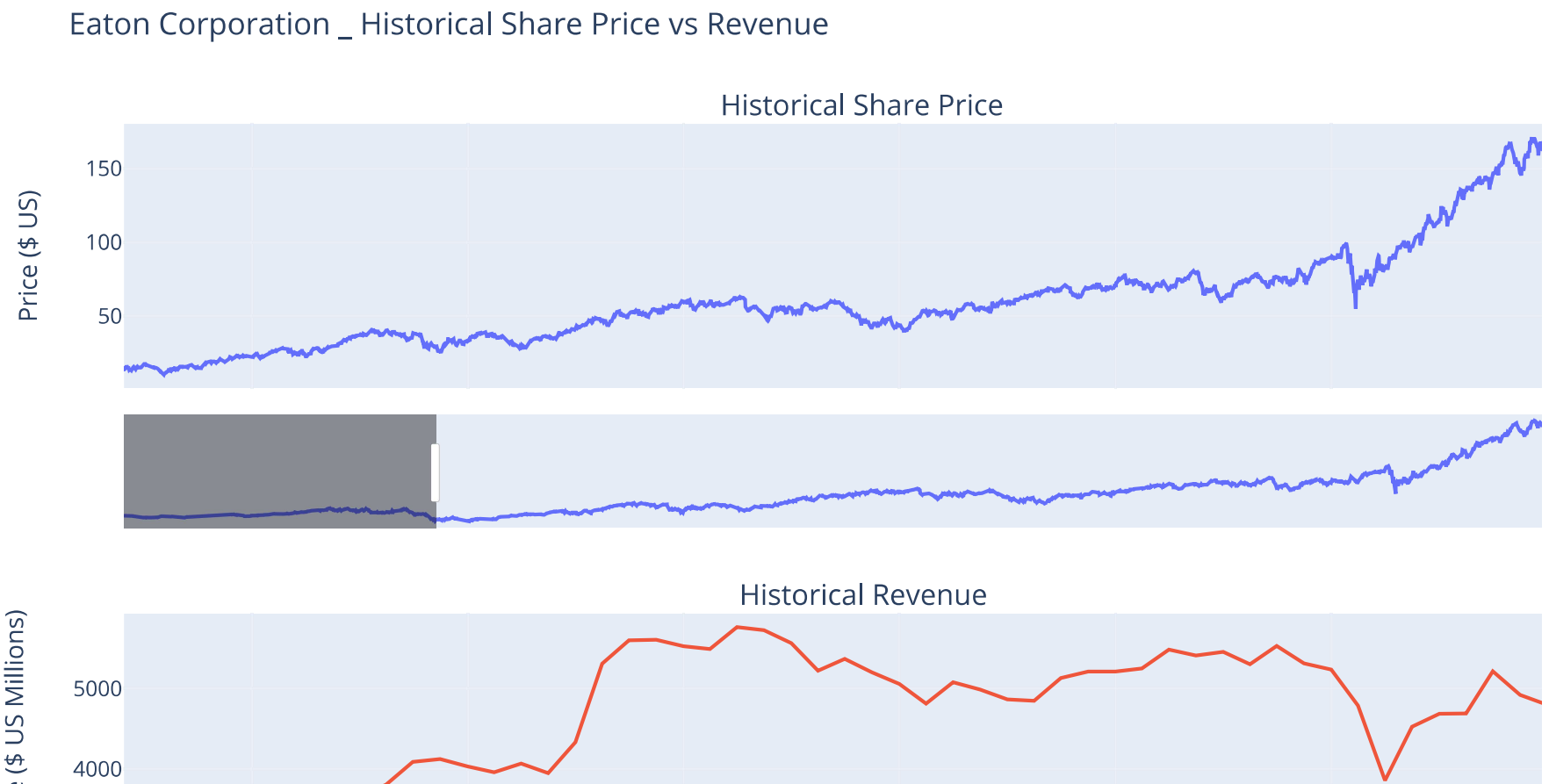
	Date	Revenue
35	Sep, 2013	1988.37
36	Jun, 2013	1760.18
37	Mar, 2013	1775.19
38	Dec, 2012	1821.59
39	Sep, 2012	1712.35
40	Jun, 2012	1849.38
41	Mar, 2012	1637.13
42	Dec, 2011	1775.45
43	Sep, 2011	1991.30
44	Jun, 2011	1746.03
45	Mar, 2011	1635.29
46	Dec, 2010	1646.65

Dashboard to Visualize EATON Corporation Stock vs Revenue

Used `make_subplots` function of `Plotly` library to visualize Eaton data.

One Column, 2 Rows are made for two graphs.
In-between, X-axis rangeslider made functional.
Can define dates for the stock and revenue data to be plotted.

In [35]:
```
fig = make_subplots(rows=2, cols=1, shared_xaxes=True, subplot_titles=("Historical Share Price", "Historical Reven

stock_data_specific = eaton_stock[eaton_stock.Date >= '2005-02-10']
revenue_data_specific = eaton_revenue[eaton_revenue.Date >= '2009-02-10']

fig.add_trace(go.Scatter(x=pd.to_datetime(stock_data_specific.Date, infer_datetime_format=True), y=stock_data_spec
fig.add_trace(go.Scatter(x=pd.to_datetime(revenue_data_specific.Date, infer_datetime_format=True), y=revenue_data_

fig.update_xaxes(title_text="Date", row=2, col=1)
fig.update_yaxes(title_text="Price ($ US)", row=1, col=1)
fig.update_yaxes(title_text="Revenue ($ US Millions)", row=2, col=1)

fig.update_layout(showlegend=False,height=600,title='Eaton Corporation _ Historical Share Price vs Revenue',xaxis_

fig.show()
```

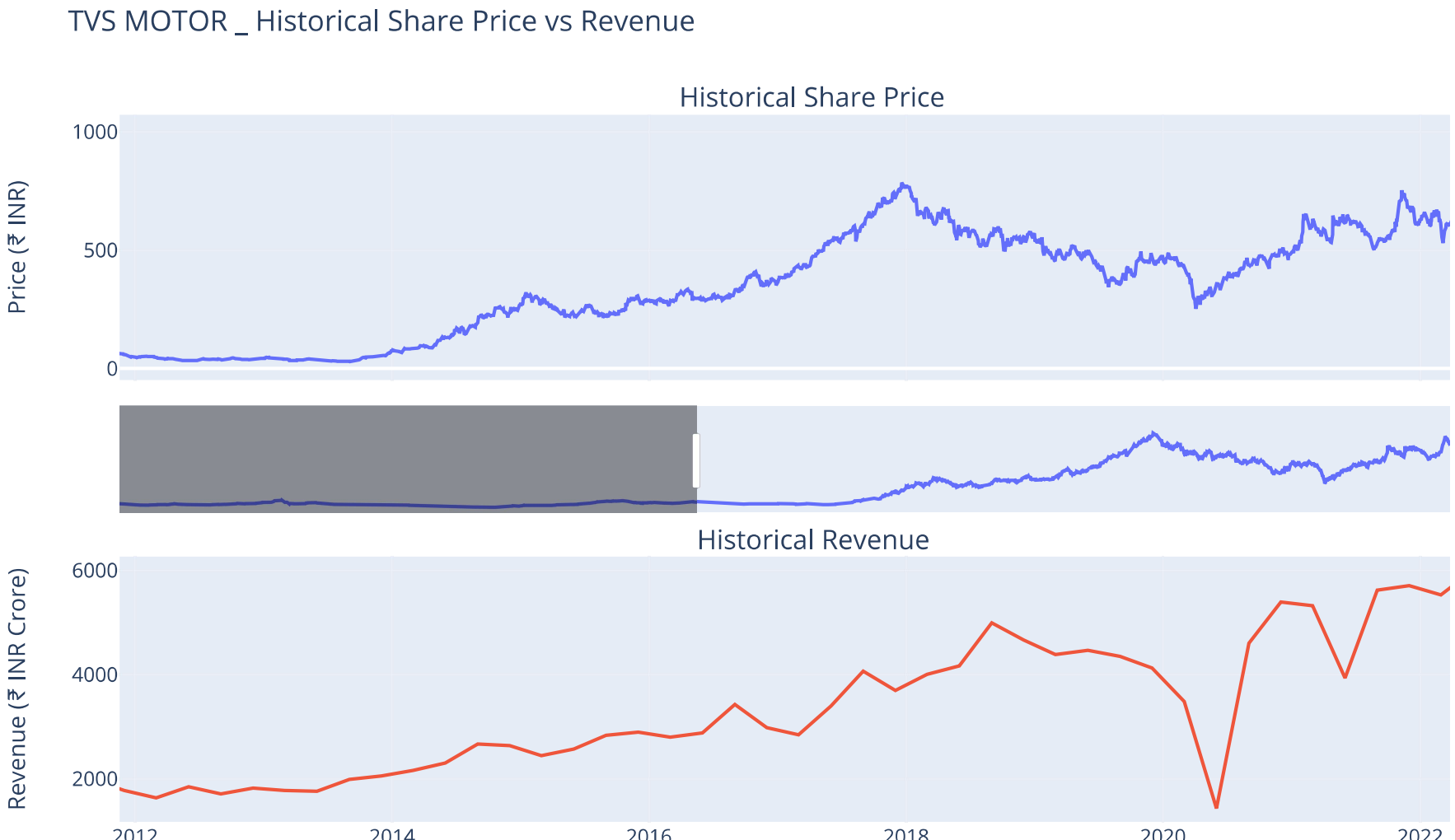


> On the DashBoard, we can move X-axis Slider_cursor to the required position to know the exact variation between Stock vs Revenue for that time period.

Dashboard to Visualize TVS MOTOR Stock vs Revenue

Used `make_subplots` function of `Plotly` library to visualize Eaton data.

In [36]:
```
fig = make_subplots(rows=2, cols=1, shared_xaxes=True, subplot_titles=("Historical Share Price", "Historical Reven

fig.add_trace(go.Scatter(x=pd.to_datetime(tvs_stock.Date, infer_datetime_format=True), y=tvs_stock.Close.astype("f
fig.add_trace(go.Scatter(x=pd.to_datetime(tvs_revenue.Date, infer_datetime_format=True), y=tvs_revenue.Revenue.ast

fig.update_xaxes(title_text="Date", row=2, col=1)
fig.update_yaxes(title_text="Price (₹ INR)", row=1, col=1)
fig.update_yaxes(title_text="Revenue (₹ INR Crore)", row=2, col=1)

fig.update_layout(showlegend=False,height=600,title='TVS MOTOR _ Historical Share Price vs Revenue',xaxis_rangesli

fig.show()
```



> On the DashBoard, we can move X-axis Slider_cursor to the required position to know the exact variation between Stock vs Revenue for that time period.

Forecasting of EATON Corporation Stock Price

Here, Forecasting is done on 30 days. For Now, daily Close Stock Price data is considered for Forecasting. To make this Forecasting better, there is a scope to include Sentiments of news and Sentiments of Peoples on those news.

Time Step considered to be Previous 100 days Close Stock Price data from the day we want to Forecast Stock Price.

Extracted Close column data from `eaton_stock` dataframe

In [37]:
```
eaton_stock1 = eaton_stock.Close.astype("float")
display(eaton_stock1)
```

```
0          0.364929
1          0.361056
2          0.354281
3          0.352345
4          0.349441
             ...
12670    140.710007
12671    139.240005
12672    136.639999
12673    138.770004
12674    137.300003
Name: Close, Length: 12675, dtype: float64
```

LSTM are sensitive to the scale of the data. So, Feature Scaling is must using MinMax scaler.

Each Close data value will get scaled between 0 to 1.

In [38]:
```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
eaton_stock1=scaler.fit_transform(np.array(eaton_stock1).reshape(-1,1))
```

In [39]:
```
print(eaton_stock1)
```

```
[[0.00120275]
 [0.00118009]
 [0.00114043]
 ...
 [0.79874684]
 [0.8112126 ]
 [0.80260948]]
```

Splitted `eaton_stock1` dataset into Train set and Test set. Training Size is taken as 80% of the whole dataset.

In [40]:
```
training_size=int(len(eaton_stock1)*0.8)
test_size=len(eaton_stock1)-training_size
eaton_train_data = eaton_stock1[0:training_size,:]
eaton_test_data = eaton_stock1[training_size:len(eaton_stock1),:1]
```

In [41]:
```
print(training_size,test_size)
```

```
10140 2535
```

In [42]:
```
display(eaton_train_data)
```

```
array([[0.00120275],
       [0.00118009],
       [0.00114043],
       ...,
       [0.1955684 ],
       [0.19596825],
       [0.19743434]])
```

Defined a function named `create_dataset` to convert an array of values into a dataset matrix from the start of the time_step to the end of it.

In [43]:
```
def create_dataset(dataset, time_step=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-time_step-1):
        a = dataset[i:(i+time_step), 0]   ### i = 0,1,2,3-----99   100
        dataX.append(a)
        dataY.append(dataset[i + time_step, 0])
    return np.array(dataX), np.array(dataY)
```

Time_Step is taken as 100. It means, Previous 100 days Close Stock Price data from the day we want to Forecast Stock Price will be taken as input to Forecast next day Stock Price.

Reshape dataset into X=t,t+1,t+2,t+3 and Y=t+4

In [44]:
```
time_step = 150
X_train_eaton, y_train_eaton = create_dataset(eaton_train_data, time_step)
X_test_eaton, y_test_eaton = create_dataset(eaton_test_data, time_step)
```

In [45]:
```
print(X_train_eaton)
```

```
[[0.00120275 0.00118009 0.00114043 ... 0.00094735 0.00093582 0.00095889]
 [0.00118009 0.00114043 0.0011291  ... 0.00093582 0.00095889 0.00094735]
 [0.00114043 0.0011291  0.00111211 ... 0.00095889 0.00094735 0.00095889]
 ...
 [0.18679686 0.18944159 0.18779414 ... 0.19253566 0.19213939 0.19121458]
 [0.18944159 0.18779414 0.19629184 ... 0.19213939 0.19121458 0.19068143]
 [0.18779414 0.19629184 0.19624839 ... 0.19121458 0.19068143 0.1955684 ]]
```

In [46]:
```
print(X_train_eaton.shape)
print(y_train_eaton.shape)
```

```
(9989, 150)
(9989,)
```

In [47]:
```
print(X_test_eaton.shape)
print(y_test_eaton.shape)
```

```
(2384, 150)
(2384,)
```

reshape input to be [samples, time steps, features] which is required for LSTM

In [48]:
```
X_train_eaton =X_train_eaton.reshape(X_train_eaton.shape[0],X_train_eaton.shape[1] , 1)
X_test_eaton = X_test_eaton.reshape(X_test_eaton.shape[0],X_test_eaton.shape[1] , 1)
```

Create the Stacked LSTM model

In [49]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
```

In [50]:
```
eaton_model=Sequential()
eaton_model.add(LSTM(50,return_sequences=True,input_shape=(X_train_eaton.shape[1],1)))
eaton_model.add(LSTM(50,return_sequences=True))
eaton_model.add(LSTM(50))
eaton_model.add(Dense(1))
eaton_model.compile(loss='mean_squared_error',optimizer='adam')
```

In [51]:
```
eaton_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 150, 50)           10400

 lstm_1 (LSTM)               (None, 150, 50)           20200

 lstm_2 (LSTM)               (None, 50)                20200

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
```

In [52]:
```
eaton_model.fit(X_train_eaton,y_train_eaton,validation_data=(X_test_eaton,y_test_eaton),epochs=100,batch_size=64,v
```

```
Epoch 1/100
157/157 [==============================] - 57s 330ms/step - loss: 2.6316e-04 - val_loss: 0.0045
Epoch 2/100
157/157 [==============================] - 48s 304ms/step - loss: 1.8081e-05 - val_loss: 0.0020
Epoch 3/100
157/157 [==============================] - 45s 288ms/step - loss: 1.6958e-05 - val_loss: 0.0033
Epoch 4/100
157/157 [==============================] - 42s 270ms/step - loss: 1.6040e-05 - val_loss: 0.0015
Epoch 5/100
157/157 [==============================] - 42s 271ms/step - loss: 1.5180e-05 - val_loss: 0.0013
Epoch 6/100
157/157 [==============================] - 43s 271ms/step - loss: 1.3488e-05 - val_loss: 0.0035
Epoch 7/100
157/157 [==============================] - 42s 270ms/step - loss: 1.2578e-05 - val_loss: 9.6697e-04
Epoch 8/100
157/157 [==============================] - 43s 272ms/step - loss: 1.1817e-05 - val_loss: 6.7729e-04
Epoch 9/100
157/157 [==============================] - 43s 275ms/step - loss: 1.0762e-05 - val_loss: 4.8824e-04
Epoch 10/100
```

Do the prediction and check performance metrics

In [53]:
```
train_predict_eaton=eaton_model.predict(X_train_eaton)
test_predict_eaton=eaton_model.predict(X_test_eaton)
```

```
313/313 [==============================] - 20s 60ms/step
75/75 [==============================] - 5s 60ms/step
```

Transformback to original form

In [54]:
```
train_predict_eaton=scaler.inverse_transform(train_predict_eaton)
test_predict_eaton=scaler.inverse_transform(test_predict_eaton)
```

Calculate RMSE performance metrics

In [55]:
```
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(y_train_eaton,train_predict_eaton))
```

Out[55]:
```
12.727818005568286
```

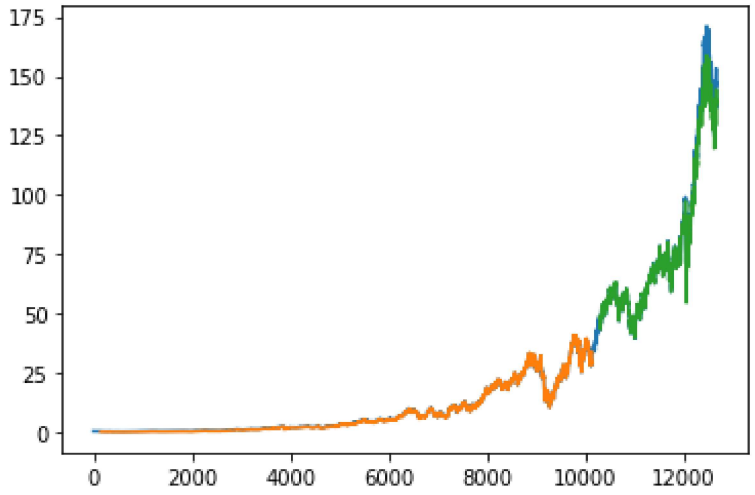
Test Data RMSE

In [56]:
```
math.sqrt(mean_squared_error(y_test_eaton,test_predict_eaton))
```

Out[56]:
```
83.22806374847316
```

Plotting :

In [57]:
```
# shift train predictions for plotting
look_back=time_step
trainPredictPlot = np.empty_like(eaton_stock1)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict_eaton)+look_back, :] = train_predict_eaton
# shift test predictions for plotting
testPredictPlot = np.empty_like(eaton_stock1)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict_eaton)+(look_back*2)+1:len(eaton_stock1)-1, :] = test_predict_eaton
# plot baseline and predictions
plt.plot(scaler.inverse_transform(eaton_stock1))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```



In [58]:
```
x_input=eaton_test_data[(len(eaton_test_data)-time_step):].reshape(1,-1)
x_input.shape
```

Out[58]:
```
(1, 150)
```

In [59]:
```
temp_input=list(x_input)
temp_input=temp_input[0].tolist()
```

In [60]:
```
temp_input
```

Out[60]:
```
[0.9109577345611037,
 0.9164833491108242,
 0.9281100365488519,
 0.9017485258139333,
 0.8681921784211526,
 0.8631846906995639,
 0.8710700952240138,
 0.9018637246433524,
 0.8900066395464865,
 0.8662352271437175,
 0.8749264876655926,
 0.8942659599867778,
 0.9028997103953614,
 0.8762502025868949,
 0.8817758171366153,
 0.8725666083894449,
 0.8584649321472686,
 0.8642206764515729,
 0.8861502471049078,
```

demonstrate prediction for next 30 days

In [61]:
```
from numpy import array

future_days_want_to_predict = 30

lst_output_eaton=[]
n_steps=time_step
i=0
while(i<future_days_want_to_predict):

    if(len(temp_input)>time_step):
        #print(temp_input)
        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))
        #print(x_input)
        yhat = eaton_model.predict(x_input, verbose=0)
        print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        #print(temp_input)
        lst_output_eaton.extend(yhat.tolist())
        i=i+1
    else:
        x_input = x_input.reshape((1, n_steps,1))
        yhat = eaton_model.predict(x_input, verbose=0)
        print(yhat[0])
        temp_input.extend(yhat[0].tolist())
        print(len(temp_input))
        lst_output_eaton.extend(yhat.tolist())
        i=i+1

print(lst_output_eaton)
```

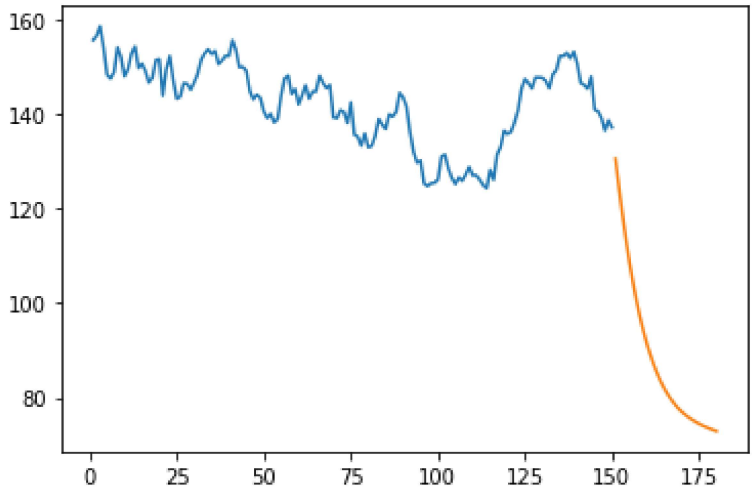
```
[0.76365167]
151
1 day input [0.91648335 0.92811004 0.90174853 0.86819218 0.86318469 0.8710701
 0.90186372 0.89000664 0.86623523 0.87492649 0.89426596 0.90289971
 0.8762502  0.88177582 0.87256661 0.85846493 0.86422068 0.88615025
 0.88712881 0.84200329 0.87492649 0.89104271 0.86117014 0.83820449
 0.84056428 0.85737135 0.85673472 0.84932708 0.85823945 0.86871432
 0.88648111 0.89429382 0.89944455 0.89377302 0.89753466 0.88121474
 0.88567088 0.89134241 0.89151601 0.91072966 0.89875014 0.87733727
 0.87762661 0.87247597 0.84753301 0.83740534 0.84325038 0.83925727
```

```
0.82241645 0.81402506 0.81952291 0.8087586  0.81309901 0.8426718
0.86356369 0.86686239 0.8442343  0.85019517 0.83115513 0.8423245
0.85488287 0.83833139 0.84689647 0.84735941 0.86674657 0.85777642
0.85146834 0.8553459  0.81495094 0.81396711 0.82415336 0.82170865
0.80802999 0.83364119 0.79371105 0.79114988 0.77985772 0.79510799
0.77805329 0.77950846 0.79231401 0.81291942 0.80581817 0.80069584
0.81874018 0.81600443 0.82188332 0.8454572  0.84004392 0.82851886
0.7966796  0.77234907 0.75878678 0.76163889 0.7331174  0.72968313
0.73329198 0.73375764 0.73765753 0.76641187 0.76885659 0.75180189
```

In [62]:
```
day_new_eaton=np.arange(1,(time_step+1))
day_pred_eaton=np.arange((time_step+1),(time_step+1+future_days_want_to_predict))
```

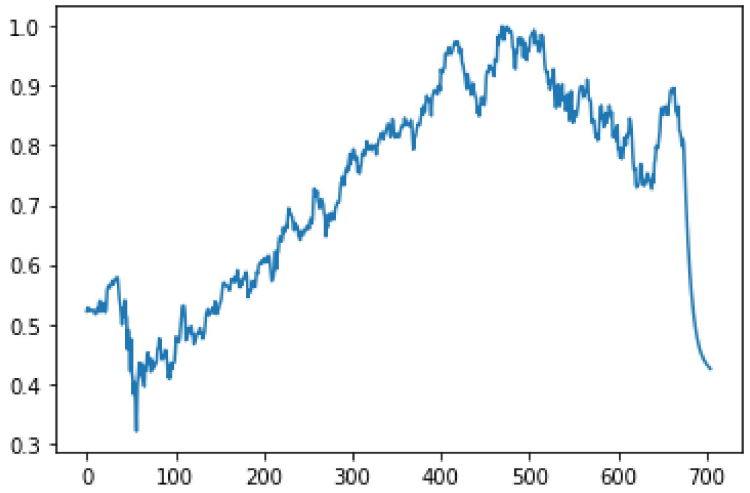
In [63]:
```
plt.plot(day_new_eaton,scaler.inverse_transform(eaton_stock1[(len(eaton_stock1)-time_step):]))
plt.plot(day_pred_eaton,scaler.inverse_transform(lst_output_eaton))
```

Out[63]:
```
[<matplotlib.lines.Line2D at 0x2fada835c00>]
```



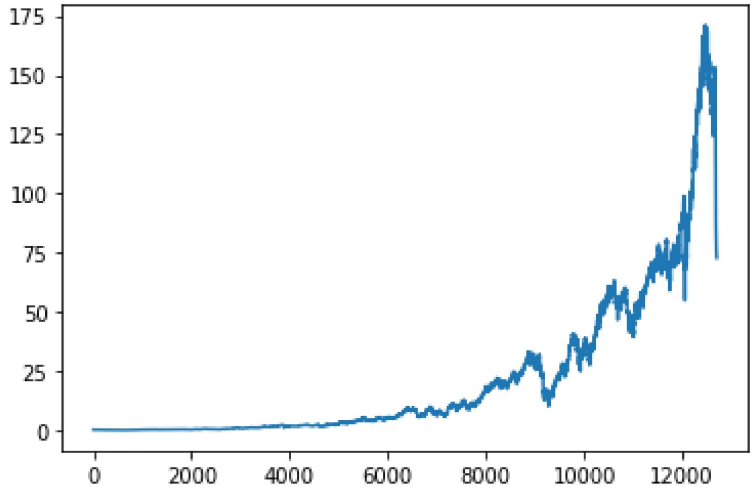
In [64]:
```
eaton_stock3=eaton_stock1.tolist()
eaton_stock3.extend(lst_output_eaton)
eaton_stock3
plt.plot(eaton_stock3[12000:])
```

Out[64]:
```
[<matplotlib.lines.Line2D at 0x2fada7a78e0>]
```



In [65]:
```
eaton_stock3=scaler.inverse_transform(eaton_stock3).tolist()
plt.plot(eaton_stock3)
```

Out[65]:
```
[<matplotlib.lines.Line2D at 0x2fada7720e0>]
```



Forecasting of TVS MOTOR Stock Price

Here, Forecasting is done on 30 days. For Now, daily Close Stock Price data is considered for Forecasting. To make this Forecasting better, there is a scope to include Sentiments of news and Sentiments of Peoples on those news.

Time Step considered to be Previous 100 days Close Stock Price data from the day we want to Forecast Stock Price.

Extracted Close column data from `tvs_stock` dataframe

In [66]:
```
tvs_stock = tvs_stock.loc[::-1]
tvs_stock = tvs_stock.reset_index(drop=True)
```

In [67]:
```
tvs_stock1 = tvs_stock.Close.astype("float")
tvs_stock1
```

Out[67]:
```
0         47.42
1         46.33
2         45.77
3         44.70
4         42.27
         ...
4584     949.10
4585     954.45
4586     957.85
4587     985.60
4588    1015.10
Name: Close, Length: 4589, dtype: float64
```

LSTM are sensitive to the scale of the data. So, Feature Scaling is must using MinMax scaler.

Each Close data value will get scaled between 0 to 1.

In [68]:
```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
tvs_stock1=scaler.fit_transform(np.array(tvs_stock1).reshape(-1,1))
```

In [69]:
```
print(tvs_stock1)
```

```
[[0.0395903 ]
 [0.03850849]
 [0.0379527 ]
 ...
 [0.94318013]
 [0.97072164]
 [1.        ]]
```

Splitted `tvs_stock1` dataset into Train set and Test set. Training Size is taken as 80% of the whole dataset.

In [70]:
```
training_size=int(len(tvs_stock1)*0.8)
test_size=len(tvs_stock1)-training_size
tvs_train_data,tvs_test_data=tvs_stock1[0:training_size,:],tvs_stock1[training_size:len(tvs_stock1),:1]
```

In [71]:
```
training_size,test_size
```

Out[71]:
```
(3671, 918)
```

In [72]:
```
tvs_train_data
```

Out[72]:
```
array([[0.0395903 ],
       [0.03850849],
       [0.0379527 ],
       ...,
       [0.55362903],
       [0.55824409],
       [0.56330578]])
```

Defined a function named `create_dataset` to convert an array of values into a dataset matrix from the start of the time_step to the end of it.

In [73]:
```
def create_dataset(dataset, time_step=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-time_step-1):
        a = dataset[i:(i+time_step), 0]   ### i = 0,1,2,3-----99   100
        dataX.append(a)
        dataY.append(dataset[i + time_step, 0])
    return np.array(dataX), np.array(dataY)
```

Time_Step is taken as 100. It means, Previous 100 days Close Stock Price data from the day we want to Forecast Stock Price will be taken as input to Forecast next day Stock Price.

Reshape dataset into X=t,t+1,t+2,t+3 and Y=t+4

In [74]:
```
time_step = 150
X_train, y_train = create_dataset(tvs_train_data, time_step)
X_test, y_test = create_dataset(tvs_test_data, time_step)
```

In [75]:
```
print(X_train)
```

```
[[0.0395903  0.03850849 0.0379527  ... 0.03500501 0.03443929 0.0343599 ]
 [0.03850849 0.0379527  0.03689074 ... 0.03443929 0.0343599  0.03423087]
 [0.0379527  0.03689074 0.03447899 ... 0.0343599  0.03423087 0.03437975]
 ...
 [0.60394811 0.61551058 0.61362486 ... 0.52995822 0.53626051 0.54067707]
 [0.61551058 0.61362486 0.60930754 ... 0.53626051 0.54067707 0.55883958]
 [0.61362486 0.60930754 0.6065782  ... 0.54067707 0.55883958 0.55362903]]
```

In [76]:
```
print(X_train.shape), print(y_train.shape)
```

```
(3520, 150)
(3520,)
```

Out[76]:
```
(None, None)
```

In [77]:
```
print(X_test.shape), print(y_test.shape)
```

```
(767, 150)
(767,)
```

Out[77]:
```
(None, None)
```

reshape input to be [samples, time steps, features] which is required for LSTM

In [78]:
```
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)
```

Create the Stacked LSTM model

In [79]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
```

In [80]:
```
model=Sequential()
model.add(LSTM(50,return_sequences=True,input_shape=(X_train.shape[1],1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
```

In [81]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_3 (LSTM)               (None, 150, 50)           10400

 lstm_4 (LSTM)               (None, 150, 50)           20200

 lstm_5 (LSTM)               (None, 50)                20200

 dense_1 (Dense)             (None, 1)                 51

=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
```

In [82]:
```
model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=100,batch_size=64,verbose=1)
```

```
Epoch 1/100
55/55 [==============================] - 19s 263ms/step - loss: 0.0032 - val_loss: 0.0014
Epoch 2/100
55/55 [==============================] - 13s 238ms/step - loss: 1.6601e-04 - val_loss: 0.0020
Epoch 3/100
55/55 [==============================] - 13s 242ms/step - loss: 1.7448e-04 - val_loss: 0.0014
Epoch 4/100
55/55 [==============================] - 13s 236ms/step - loss: 1.6501e-04 - val_loss: 0.0014
Epoch 5/100
55/55 [==============================] - 13s 238ms/step - loss: 1.4103e-04 - val_loss: 0.0013
Epoch 6/100
55/55 [==============================] - 13s 238ms/step - loss: 1.3666e-04 - val_loss: 0.0011
Epoch 7/100
55/55 [==============================] - 13s 238ms/step - loss: 1.2546e-04 - val_loss: 0.0010
Epoch 8/100
55/55 [==============================] - 13s 236ms/step - loss: 1.3685e-04 - val_loss: 0.0013
Epoch 9/100
55/55 [==============================] - 13s 235ms/step - loss: 1.1881e-04 - val_loss: 9.9465e-04
Epoch 10/100
```

Do the prediction and check performance metrics

In [83]:
```
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
```

```
110/110 [==============================] - 8s 61ms/step
24/24 [==============================] - 3s 60ms/step
```

Transformback to original form

In [84]:
```
train_predict=scaler.inverse_transform(train_predict)
test_predict=scaler.inverse_transform(test_predict)
```

Calculate RMSE performance metrics

In [85]:
```
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(y_train,train_predict))
```

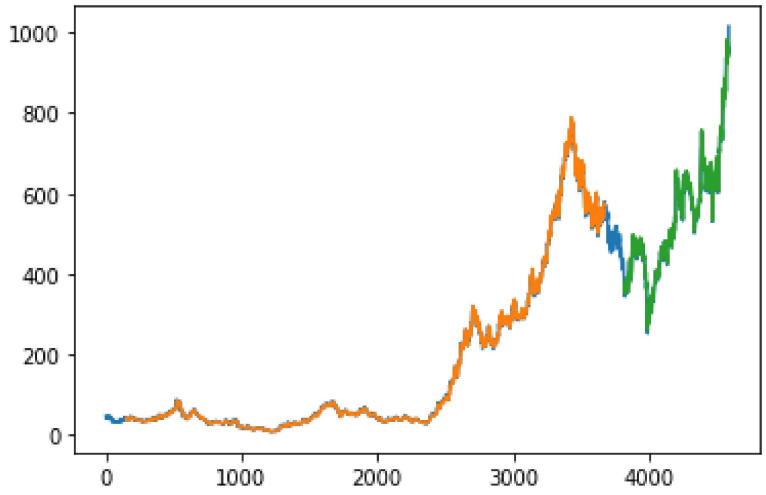
Out[85]:
```
251.8190429620704
```

In [86]:
```
### Test Data RMSE
math.sqrt(mean_squared_error(y_test,test_predict))
```

Out[86]:
```
561.5904794298759
```

Plotting :

In [87]:
```
# shift train predictions for plotting
look_back=time_step
trainPredictPlot = np.empty_like(tvs_stock1)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
# shift test predictions for plotting
testPredictPlot = np.empty_like(tvs_stock1)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(tvs_stock1)-1, :] = test_predict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(tvs_stock1))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```



In [88]:
```
x_input=tvs_test_data[(len(tvs_test_data)-time_step):].reshape(1,-1)
x_input.shape
```

Out[88]:
```
(1, 150)
```

In [89]:
```
temp_input=list(x_input)
temp_input=temp_input[0].tolist()
```

In [90]:
```
temp_input
```

Out[90]:
```
[0.6001270383199182,
 0.6085631767519875,
 0.6176444316523914,
 0.5899540478577171,
 0.5942713657611877,
 0.6140714789046914,
 0.6139722302172553,
 0.6448385720098851,
 0.6406205027938505,
 0.624691088460355,
 0.6440942068541143,
 0.647419037883224,
 0.6559544250027293,
 0.6480145300078406,
 0.6294054011135702,
 0.6508431175997696,
 0.6526295939736196,
 0.6586837639072223,
 0.6510416149746419,
```

Demonstrate prediction for next 30 days

In [91]:
```
from numpy import array

future_days_want_to_predict = 30

lst_output=[]
n_steps=time_step
i=0
while(i<future_days_want_to_predict):

    if(len(temp_input)>time_step):
        #print(temp_input)
        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))
        #print(x_input)
        yhat = model.predict(x_input, verbose=0)
        print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        #print(temp_input)
        lst_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))
        lst_output.extend(yhat.tolist())
        i=i+1

print(lst_output)
```

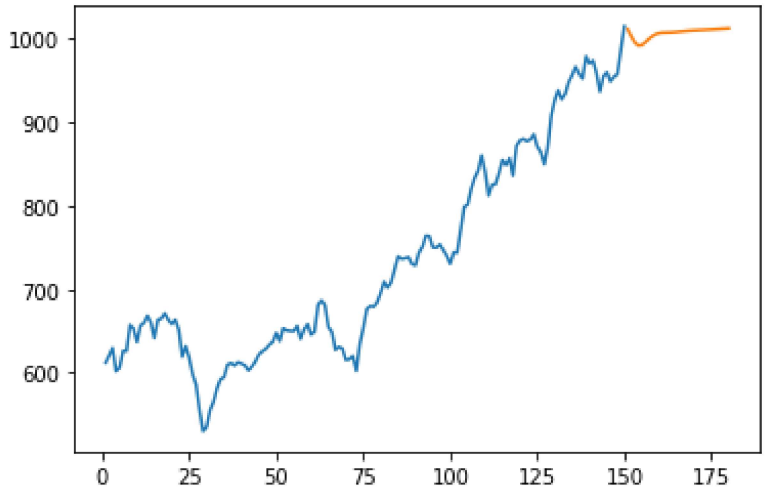
```
[0.99637556]
151
1 day input [0.60856318 0.61764443 0.58995405 0.59427137 0.61407148 0.61397223
 0.64483857 0.6406205  0.62469109 0.64409421 0.64741904 0.65595443
 0.64801453 0.6294054  0.65084312 0.65262959 0.65868376 0.65104161
 0.64597993 0.65104161 0.63908215 0.6069752  0.61972865 0.60548647
 0.5859841  0.57367726 0.54310867 0.51864387 0.52365592 0.54380341
 0.55352978 0.56975694 0.58032693 0.58285777 0.59764582 0.59963079
 0.59665333 0.60047441 0.59908493 0.59690146 0.59099616 0.5953631
 0.60136765 0.60965491 0.61347599 0.6165527  0.62136626 0.62474071
```

```
0.63555882 0.62568358 0.6410175  0.6385859  0.63789116 0.63769267
0.64359796 0.62831367 0.63933027 0.64578143 0.6332761  0.63699793
0.66890638 0.67396806 0.66875751 0.6424566  0.63670018 0.61531209
0.61883542 0.61685044 0.60335262 0.60409699 0.60801731 0.59025179
0.62240837 0.64111675 0.66359657 0.66761615 0.66652441 0.67193346
0.68319819 0.69605089 0.68940123 0.69550503 0.71078932 0.72661949
0.72354278 0.7241879  0.72572625 0.71733974 0.71575176 0.73098643
0.73793384 0.75098504 0.75043918 0.73733835 0.73679248 0.7407128
0.73401352 0.72711573 0.71768711 0.7313338  0.73083756 0.75872644
```

In [92]:
```
day_new=np.arange(1,(time_step+1))
day_pred=np.arange((time_step+1),(time_step+1+future_days_want_to_predict))
```

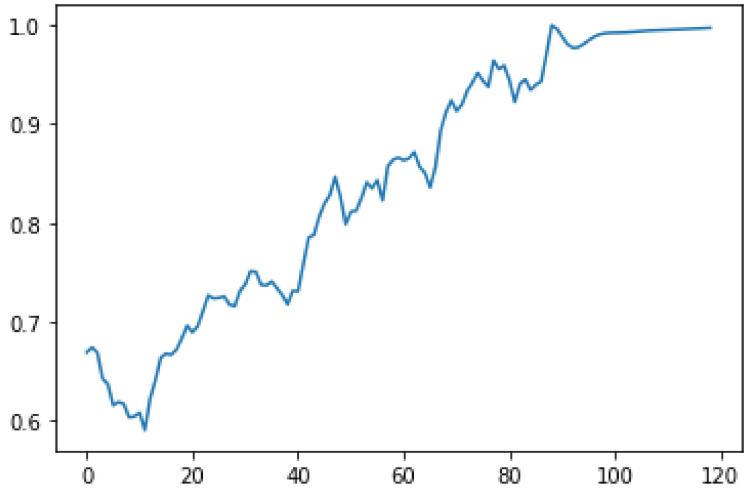
In [93]:
```
plt.plot(day_new,scaler.inverse_transform(tvs_stock1[(len(tvs_stock1)-time_step):]))
plt.plot(day_pred,scaler.inverse_transform(lst_output))
```

Out[93]:
```
[<matplotlib.lines.Line2D at 0x2faf97d5480>]
```



In [94]:
```
tvs_stock3=tvs_stock1.tolist()
tvs_stock3.extend(lst_output)
plt.plot(tvs_stock3[4500:])
```

Out[94]:
```
[<matplotlib.lines.Line2D at 0x2faf98518a0>]
```



In [95]:
```
tvs_stock3=scaler.inverse_transform(tvs_stock3).tolist()
```

In [96]:
```
plt.plot(tvs_stock3)
```

Out[96]:
```
[<matplotlib.lines.Line2D at 0x2faf98cda20>]
```