STOCK MARKET PRICE PREDICTION USING MACHINE LEARNING ALGORITHMS

A summer internship report submitted in partial fulfillment for the requirements to achieve an award of degree in **"BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING".**

Submitted by
PRANAV REDDY G **Reg No. : 221910308044**

Under the guidance of
MR.DEEPAK N.BIRADAR

Department of Computer Science and Engineering
GITAM School of Technology
GITAM Deemed to be University
Hyderabad Campus - 502329
August - 2022

GANDHI INSTITUTE OF TECHNOLOGY AND MANAGEMENT (GITAM)
(Declared as Deemed-to-be-University u/s 3 of UGC Act 1956)
HYDERABAD CAMPUS

DECLARATION

I hereby declare that the summer internship report entitled **"Stock Market Price Prediction Using Machine Learning Algorithm"** is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, submitted in partial fulfillment for the requirements to achieve an award of the degree in **"Bachelor of Technology In Computer Science And Engineering".** The work had not been submitted to any other college or university for the award of any degree or diploma.

Campus: Hyderabad
Date: 19-08-2022

Pranav Reddy G
(221910308044)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GITAM SCHOOL OF TECHNOLOGY
GITAM
(DEEMED TO BE UNIVERSITY)
HYDERABAD CAMPUS

CERTIFICATE

This is to certify that the Internship report entitled **"Stock Market Price Prediction Using Machine Learning Algorithm"** is a bonafide record of work carried out by **Pranav Reddy G (221910308044)** submitted in partial fulfillment for the requirement to achieve the award of the degree in **"Bachelors of Technology in Computer Science and Engineering."**

Mr.Deepak N.Biradar
Assistant Professor
Dept. of CSE

Dr.S.Phani Kumar
Professor & HOD
Dept. of CSE

CERTIFICATE OF COMPLETION

To whomsoever it may concern

This is to certify that **Mr. Pranav Reddy G**, a student of **GITAM University** has completed internship from May 21, 2022 to Jul 4, 2022. During this period he worked on a ML based project **"Stock Market Price Prediction"** and his performance was good.

Mr. Deepak N.Biradar,
Professor
Dept. of CSE

ACKNOWLEDGEMENT

I express my thanks to **Mr. Deepak N.Biradar**, Assistant Professor from the Department of Computer Science for taking part in useful decisions & giving necessary advice and guidance. I choose this moment to acknowledge his contribution gratefully. I would also like to thank **Mrs.Vani Prasanna** Ma'am and the CSE department for giving me such a wonderful opportunity to work on this internship project.

Sincerely,
Pranav Reddy G
221910308044

ABSTRACT

A stock is a financial instrument that denotes a portion of ownership in a firm, by buying a stock, one acquires a share. Investors purchase stocks or shares with the hope that the company's value will rise, increasing the value of their initial investment. Machine learning algorithms can be applied to the previously accessible data to train a model that can forecast future stock market patterns based on previous trends.

In this research, the closing price was predicted using stock indicators such as ‘Moving Averages’ and ‘Relative Strength Index’ as training data and by applying machine learning algorithms namely ‘XGBoost Regressor’ and ‘Linear Regression’. The accuracy rates achieved were 93.8% and 99.8%, respectively.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

1.0 Introduction :

A stock is a security that represents fractional ownership in a company, by purchasing a stock one owns a small piece of the company called share. Investors buy shares or stocks expecting the company's value to go up which in turn increases the value of their investment.

Machine Learning is a study where computers are trained to learn without being explicitly programmed. It is a subset of Artificial Intelligence where computational learning of algorithms takes place, to learn from and make predictions on data.

Supervised learning algorithms like 'XGBoost Regressor' that attempts to predict the target variable by combining estimates of simpler, weaker models and 'Linear Regression' which predicts the target variable based on independent variables can be used to predict closing price of stocks.

1.1 Stock Market And Data Analysis :

A stock market is the gathering of buyers and sellers of stocks, which represent ownership claims on businesses. These securities may include stock that is only traded privately, such as shares of private companies that are offered to investors through equity crowdfunding platforms, as well as stock that is listed on a public stock exchange. An investing strategy is typically present when making an investment. The marketplace where stockbrokers and traders can buy and sell bonds, equity stock, and other assets is a stock exchange. Data related to the current stock market can be collected through 'yfinance' a yahoo finance based API.

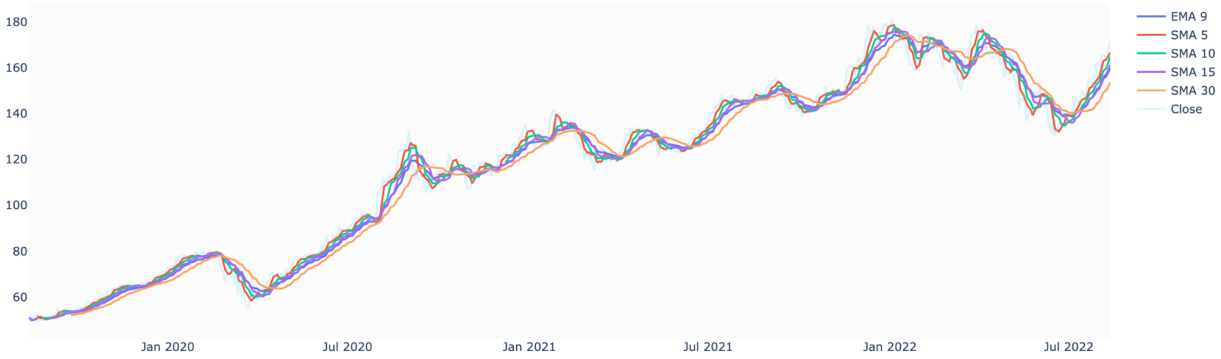
The acquired data can be analyzed by OHLC chart and a couple of technical indicators like 'Relative Strength Index' (RSI) , 'Moving Averages' and Moving average convergence divergence (MACD) signal.

OHLC which typically stands for Open-High-Low-Close is a type of chart which is used to illustrate movements in the price of a financial instrument over time.

Fig. 1.0 OHLC Chart

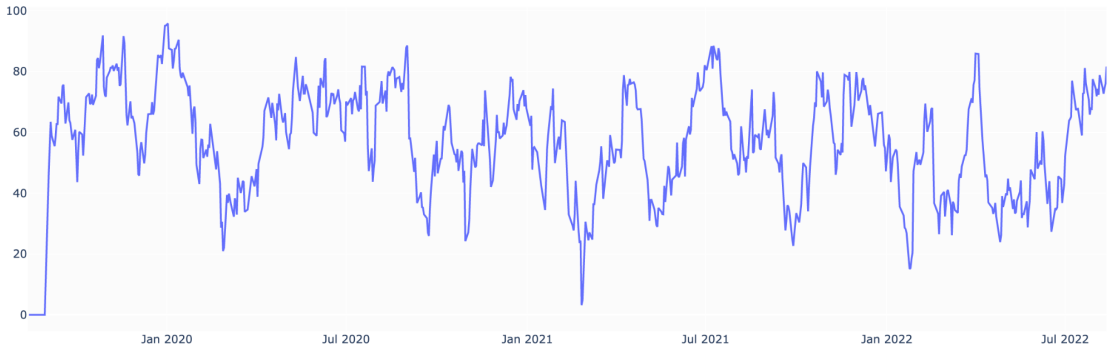
In finance, a moving average (MA) is a stock indicator commonly used in technical analysis. The reason for calculating the moving average of a stock is to help smooth out the price data by creating a constantly updated average price.

Fig. 2.0 Moving Averages Graph



Relative Strength Index (RSI) is an indicator which indicates whether the stock is overbought/oversold.

Fig. 3.0 RSI Graph



'Moving Average Convergence/divergence', or MACD, is a trading indicator used in technical stock price research. It is intended to show alterations in a trend's strength, direction, momentum, and duration in the price of a stock.

Fig. 4.0 MACD Graph

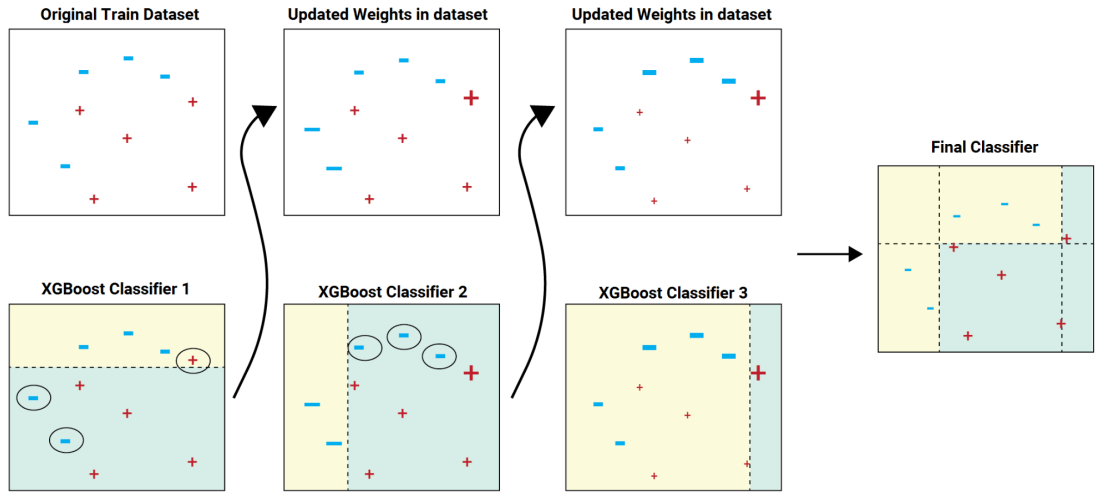
1.2 Machine Learning Algorithms Applied :

Supervised Learning algorithms are used in this project to determine the prediction. A supervised learning algorithm takes a known set of input data (the learning set) and known responses to the data (the output), and forms a model to generate reasonable predictions for the response to the new input data.

1.2.1 XGBoost :

[1] A class of ensemble machine learning methods known as gradient boosting can be applied to classification or regression predictive modeling issues. Decision tree models are used to build ensembles. In order to repair the prediction mistakes caused by earlier models, trees are added one at a time to the ensemble and fitted. Boosting is a term used to describe this kind of ensemble machine learning model. Any arbitrary differentiable loss function and the gradient descent optimization procedure are used to fit the models. Gradient boosting gets its name from this because as the model is fitted, it minimizes the loss gradient, much like a neural network. An effective open-source implementation of the gradient boosting technique is called Extreme Gradient Boosting, or XGBoost.

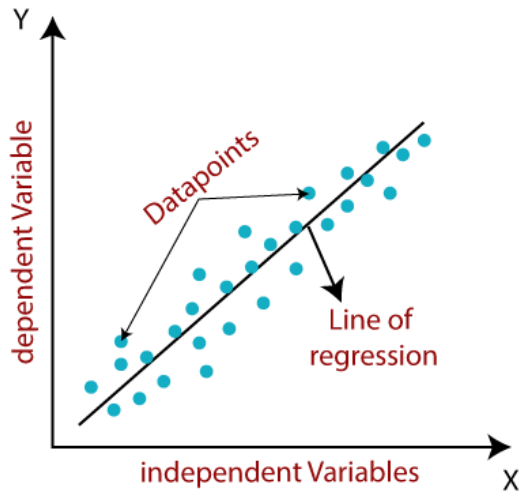
Fig. 5.0 XGBoost Representation



1.2.2 Linear Regression :

It is a statistical technique for performing predictive analysis. For real, numerical, or continuous data, linear regression makes predictions. The term "linear regression" refers to a procedure that displays a linear relationship between a dependent (y) and one or more independent (y) variables. Given that linear regression demonstrates a linear relationship, it may be used to determine how the dependent variable's value changes as a function of the independent variable's value.

Fig. 6.0 Linear Regression Representation



1.3 Technologies Implemented :

1.3.1 Python:

Python is a high-level, interpreted, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.

1.3.2 Google Colab:

Google Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning and data analysis.

1.3.3 Sklearn Library:

Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms.

1.4 Project Goals:

1. Data Collection And Analysis: The current stock data can be obtained from yfinance.The Yahoo Finance API provides access to the information about stocks historical prices,stock actions (including splits and dividends).The data obtained can be analyzed for further preparing the model.

```
Enter Stock name:AAPL
```

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
0	2019-08-13	49.167755	51.887609	49.035673	51.112255	188874000	0.0	0.0
1	2019-08-14	49.691176	50.493435	49.551757	49.590893	146189600	0.0	0.0
2	2019-08-15	49.764558	50.175470	48.837555	49.343861	108909600	0.0	0.0
3	2019-08-16	49.965128	50.669553	49.857508	50.508121	110481600	0.0	0.0
4	2019-08-19	51.515824	52.031912	51.371516	51.449787	97654400	0.0	0.0
...	...	...	...	...	...	...	...	...
752	2022-08-08	166.369995	167.809998	164.199997	164.869995	60276900	0.0	0.0
753	2022-08-09	164.020004	165.820007	163.250000	164.919998	63135500	0.0	0.0
754	2022-08-10	167.679993	169.339996	166.899994	169.240005	70170500	0.0	0.0
755	2022-08-11	170.059998	170.990005	168.190002	168.490005	57149200	0.0	0.0
756	2022-08-12	169.820007	172.169998	169.399994	172.100006	67946400	0.0	0.0

757 rows × 8 columns

2. Cleaning The Data: Cleaning the data is important because it may contain unnecessary columns and missing data cells which need to be replaced or removed before training the model.

```
[ ]  drop_cols = ['Date', 'Volume', 'Open', 'Low', 'High']

     train_df = train_df.drop(drop_cols, 1)
     valid_df = valid_df.drop(drop_cols, 1)
     test_df  = test_df.drop(drop_cols, 1)
```

3. Creating The Model: Here a model is created with a suitable machine learning algorithm that best fits the data and gives a valid/precise output.

```
[ ]  from sklearn.linear_model import LinearRegression
     lin=LinearRegression().fit(X_train,y_train)
     lin.predict(X_test)
```

```
array([159.49949959, 158.42583401, 156.06958503, 152.71199495,
       153.70201799, 156.881109  , 160.58193908, 164.35585284,
       166.43638534, 168.35192252, 170.03819231, 172.75426765,
       173.67128193, 174.58132143, 176.93601358, 177.4819454 ,
       174.79549345, 173.76308382, 175.98553115, 175.38752061,
       173.7904204 , 173.33824219, 171.45396519, 167.52245258,
       167.81988841, 169.43891271, 166.66833319, 165.29628042,
       165.90966837, 165.51354255, 164.84209843, 161.91268224,
       160.85961603, 156.71923239, 155.10981959, 159.49194305,
       158.29183195, 158.08673213, 159.38172493, 163.89005705,
       159.83450434, 158.84809232, 154.91251014, 154.42063802,
       149.05152882, 145.3500127 , 145.66330802, 145.82897224,
       148.13765552, 144.93972752, 140.96301535, 138.07435901,
       141.1667974 , 140.98602215, 142.27113362, 143.39961601,
       147.43013434, 148.13627928, 149.34327546, 150.31335192,
       146.70327743, 145.97643146, 147.48782916, 147.91700352,
       144.49779982, 140.51444812, 134.80396743, 133.37831731,
       134.99649852, 133.33286907, 133.55708136, 136.38058971,
       136.13799084, 137.82480444, 140.94382812, 141.11207615,
       138.49349556, 138.40969352, 135.91548774, 135.95353048,
       138.63167928, 141.49317249, 145.1906812 , 147.83245702,
       146.87901605, 147.42169206, 147.49072463, 148.92475585,
       150.93457727, 149.72064383, 150.98505682, 152.3550907 ,
       154.46957257, 154.32124524, 153.6823577 , 151.750416  ,
       154.25258919, 156.01108299, 160.41179702, 161.35162744,
       160.89055108, 163.83684913, 165.13322347, 165.04581283,
       165.30357602, 165.20866574, 167.16291418, 168.15766055])
```

1.5 Motivation:

This model is created with the intention of predicting future trends of the stock market, so that investors can make rational decisions before investing their valuable money into stocks or companies that are certain to go down in value.

1.6 Objective:

Training an appropriate model which predicts the price of the stock accurately based on the technical indicators.

1.7,1.8 Problem Definition, Approach:

Stock market price prediction based on the data collected from yfinance .The approach to this problem is by training a machine learning model that uses supervised learning to predict the output for example XGBoost Regressor and Linear Regression.

1.9 Dataset Description:

The dataset used in this project is stock market data that is readily available, collected from the application programming interface yfinance. yfinance is a popular open source library developed by Ran Aroussi as a means to access the financial data available on Yahoo Finance.

1.10 Outputs From The Project:

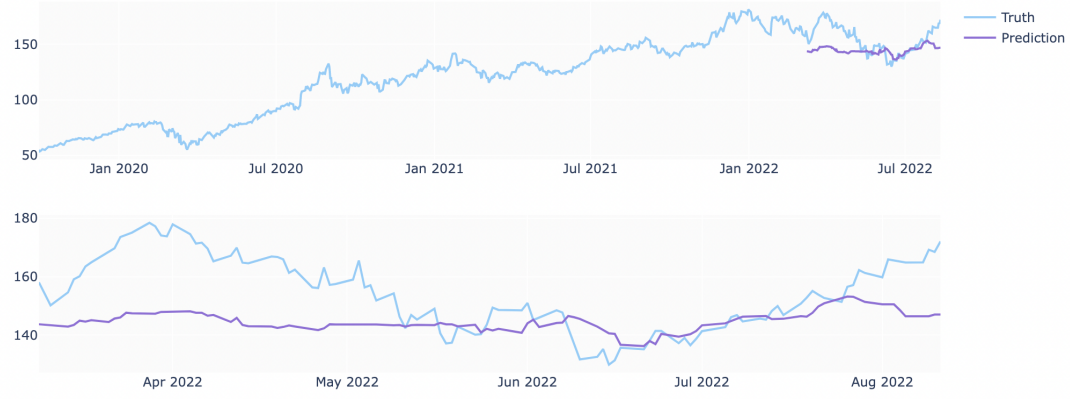
XGBoost:

```
model = xgb.XGBRegressor()
model.fit(X_train, y_train, eval_set=[(X_train, y_train), (X_valid, y_valid)], verbose=False)
```

```
y_pred = model.predict(X_test)
print(f'y_true = {np.array(y_test)[:5]}')
print(f'y_pred = {y_pred[:5]}')
```

```
y_true = [158.06788635 154.28868103 150.19039917 154.64764404 159.13482666]
y_pred = [143.71854 143.549   143.39397 142.9181  143.47704]
```

Fig. 7.0 XGBoost Prediction



Linear Regression:

```
from sklearn.linear_model import LinearRegression
lin=LinearRegression().fit(X_train,y_train)
```

```
y_pred = lin.predict(X_test)
print(f'y_true = {np.array(y_test)[:5]}')
print(f'y_pred = {y_pred[:5]}')
print(f'mean_squared_error = {mean_squared_error(y_test, y_pred)}')
```

```
y_true = [158.06788635 154.28868103 150.19039917 154.64764404 159.13482666]
y_pred = [159.49949959 158.42583401 156.06958503 152.71199495 153.70201799]
mean_squared_error = 13.036370238989551
```

Fig. 8.0 Linear Regression Prediction

CHAPTER 2

2.0 Literature Review:

2.0.1 Limitations Of Stocks:

1. Risk : Predictions with high error in accuracy can lead to huge losses in investments.
2. Common stockholders paid last : Preferred stockholders and bondholders or creditors get paid first if a company goes broke.
3. Time : Stocks take a lot of time to mature and add value to the initial investment.
4. Professional competition: Institutional investors and professional traders have more time and knowledge to invest.

2.0.2 Limitations Of XGBoost:

One of the problems causing the prediction to be way lower than the actual price is that tree-based algorithms (like XGBRegressor used here) cannot extrapolate data. If the validation or test set reaches higher than the training set maximum value, tree-based classes cannot predict them correctly.

2.0.3 Solution :

The solution to the above flaw in the model can be fixed by using Linear Regression, a supervised learning based model which is good at predicting trends where data points are linearly spread.

CHAPTER 3

3.0 Modules Imported:

3.0.1 OS : The OS module in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules.

3.0.2 NumPy : NumPy is a Python library used for working with arrays. It also has functions for working in the domain of linear algebra, fourier transform, and matrices.

3.0.3 Pandas : Pandas is an open source library in Python. It provides ready to use high-performance data structures and data analysis tools. Pandas module runs on top of NumPy and it is popularly used for data science and data analytics.

3.0.4 XGBoost : XGBoost (Extreme Gradient Boosting) belongs to a family of boosting algorithms and uses the gradient boosting (GBM) framework at its core. It is an optimized distributed gradient boosting library.

3.0.5 Matplotlib : Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

3.0.6 Plotly : The plotly Python library is an interactive, open-source plotting library that supports over 40 unique chart types covering a wide range of statistical, financial, geographic, scientific, and 3-dimensional use-cases.

3.0.7 yfinance : Yfinance is a python package that enables us to fetch historical market data from Yahoo Finance API in a Pythonic way

3.0.8 Seaborn : Seaborn is a library that uses Matplotlib underneath to plot graphs. It will be used to visualize random distributions.

3.0.9 stldecompose : A Python implementation of seasonal trend with Loess (STL) time series decomposition.

Code :

stockpredictionINTERN.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share P

+ Code + Text

RAM Disk Editing

```
import os
import numpy as np
import pandas as pd
import xgboost as xgb
import matplotlib.pyplot as plt
from xgboost import plot_importance, plot_tree
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split, GridSearchCV

# Time series decomposition
!pip install stldecompose
from stldecompose import decompose

# Chart drawing
import plotly as py
import plotly.io as pio
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot

# Mute sklearn warnings
from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)
simplefilter(action='ignore', category=DeprecationWarning)

# Show charts when running kernel
init_notebook_mode(connected=True)

# Change default background color for all visualizations
layout=go.Layout(paper_bgcolor='rgba(0,0,0,0)', plot_bgcolor='rgba(250,250,250,0.8)')
fig = go.Figure(layout=layout)
templated_fig = pio.to_templated(fig)
pio.templates['my_template'] = templated_fig.layout.template
pio.templates.default = 'my_template'
```

```
pip install yfinance
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting yfinance
  Downloading yfinance-0.1.74-py2.py3-none-any.whl (27 kB)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.7/dist-packages (from yfinance) (0.0.11)
Collecting requests>=2.26
  Downloading requests-2.28.1-py3-none-any.whl (62 kB)
     |████████████████████████████████| 62 kB 1.2 MB/s
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.7/dist-packages (from yfinance) (1.3.5)
Requirement already satisfied: lxml>=4.5.1 in /usr/local/lib/python3.7/dist-packages (from yfinance) (4.9.1)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.7/dist-packages (from yfinance) (1.21.6)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.24.0->yfinance) (2022.2)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.24.0->yfinance) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas>=0.24.0->yfinance) (1.15.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->yfinance) (1.24.3)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->yfinance) (2.10)
Requirement already satisfied: charset-normalizer<3,>=2 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->yfinance) (2.1.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->yfinance) (2022.6.15)
Installing collected packages: requests, yfinance
  Attempting uninstall: requests
    Found existing installation: requests 2.23.0
    Uninstalling requests-2.23.0:
      Successfully uninstalled requests-2.23.0
Successfully installed requests-2.28.1 yfinance-0.1.74
```

```
[4] import seaborn as sns
    sns.set_style('whitegrid')
    plt.style.use("fivethirtyeight")
    %matplotlib inline
    from pandas_datareader.data import DataReader
    import yfinance as yf
    from datetime import datetime
```

```
n=input("Enter Stock name:") #AAPL, GOOG, MSFT, AMZN
df = yf.Ticker(n).history(period='3y').reset_index()
df
```

```
Enter Stock name:AAPL
```

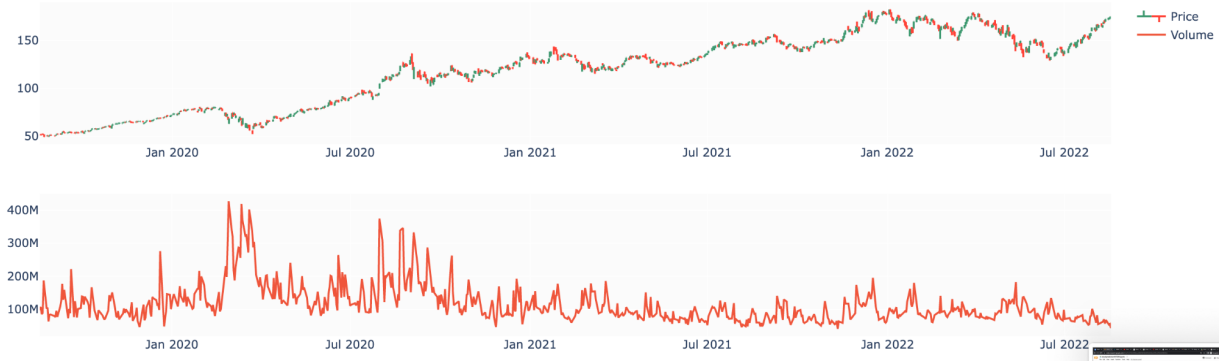
	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
0	2019-08-19	51.515828	52.031916	51.371520	51.449791	97654400	0.0	0.0
1	2019-08-20	51.579432	52.183573	51.442462	51.452244	107537200	0.0	0.0
2	2019-08-21	52.095515	52.256943	51.755533	52.009907	86141600	0.0	0.0
3	2019-08-22	52.144432	52.450171	51.547628	51.965881	89014800	0.0	0.0
4	2019-08-23	51.224757	51.865589	49.162854	49.563984	187272000	0.0	0.0
...	...	...	...	...	...	...	...	...
751	2022-08-11	170.059998	170.990005	168.190002	168.490005	57149200	0.0	0.0
752	2022-08-12	169.820007	172.169998	169.399994	172.100006	67946400	0.0	0.0
753	2022-08-15	171.520004	173.389999	171.350006	173.190002	54091700	0.0	0.0
754	2022-08-16	172.779999	173.710007	171.660004	173.029999	56180100	0.0	0.0
755	2022-08-17	172.770004	174.779999	172.570007	174.350006	43584150	0.0	0.0

756 rows × 8 columns

OHLC Chart

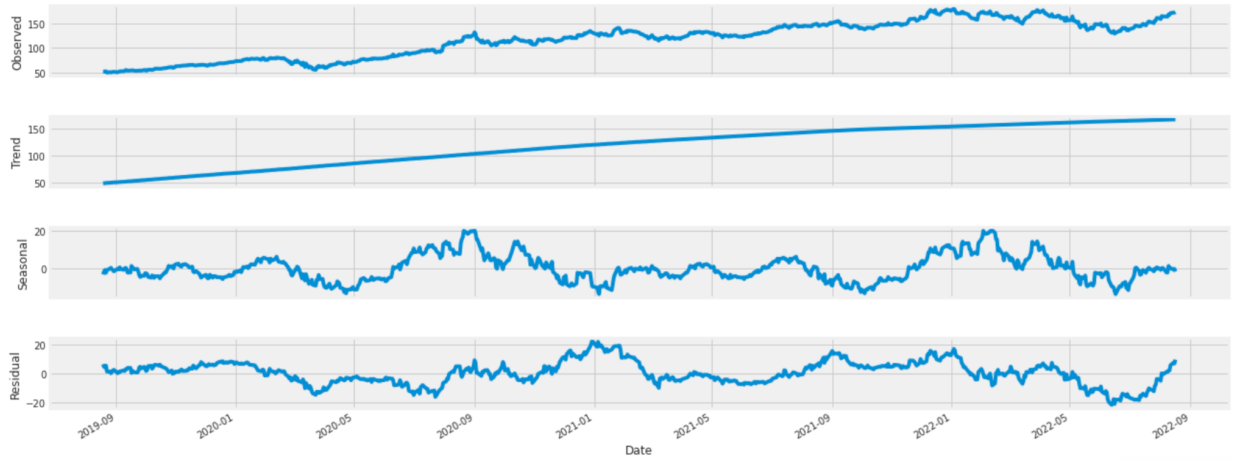
```
import plotly.io as pio
pio.renderers.default = "colab"
```

```
fig = make_subplots(rows=2, cols=1)

fig.add_trace(go.Ohlc(x=df["Date"],
                      open=df["Open"],
                      high=df["High"],
                      low=df["Low"],
                      close=df["Close"],
                      name='Price'), row=1, col=1)

fig.add_trace(go.Scatter(x=df.Date, y=df.Volume, name='Volume'), row=2, col=1)

fig.update(layout_xaxis_rangeslider_visible=False)
fig.show()
```



Decomposition:

```
df_close = df[['Date', 'Close']].copy()
df_close = df_close.set_index('Date')
df_close.head()

decomp = decompose(df_close, period=365)
fig = decomp.plot()
fig.set_size_inches(20, 8)
```



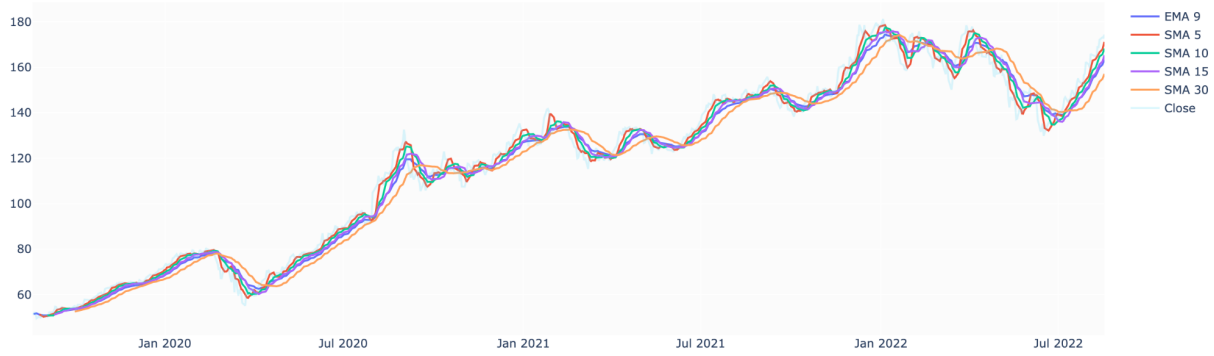
Technical indicators

Moving Averages

Calculating few moving averages to be used as features: SMA5 , SMA10, SMA15, SMA30 and EMA9

+ Code + Text

```
[9] df['EMA_9'] = df['Close'].ewm(9).mean().shift()
    df['SMA_5'] = df['Close'].rolling(5).mean().shift()
    df['SMA_10'] = df['Close'].rolling(10).mean().shift()
    df['SMA_15'] = df['Close'].rolling(15).mean().shift()
    df['SMA_30'] = df['Close'].rolling(30).mean().shift()

    fig = go.Figure()
    fig.add_trace(go.Scatter(x=df.Date, y=df.EMA_9, name='EMA 9'))
    fig.add_trace(go.Scatter(x=df.Date, y=df.SMA_5, name='SMA 5'))
    fig.add_trace(go.Scatter(x=df.Date, y=df.SMA_10, name='SMA 10'))
    fig.add_trace(go.Scatter(x=df.Date, y=df.SMA_15, name='SMA 15'))
    fig.add_trace(go.Scatter(x=df.Date, y=df.SMA_30, name='SMA 30'))
    fig.add_trace(go.Scatter(x=df.Date, y=df.Close, name='Close', opacity=0.2))
    fig.show()
```

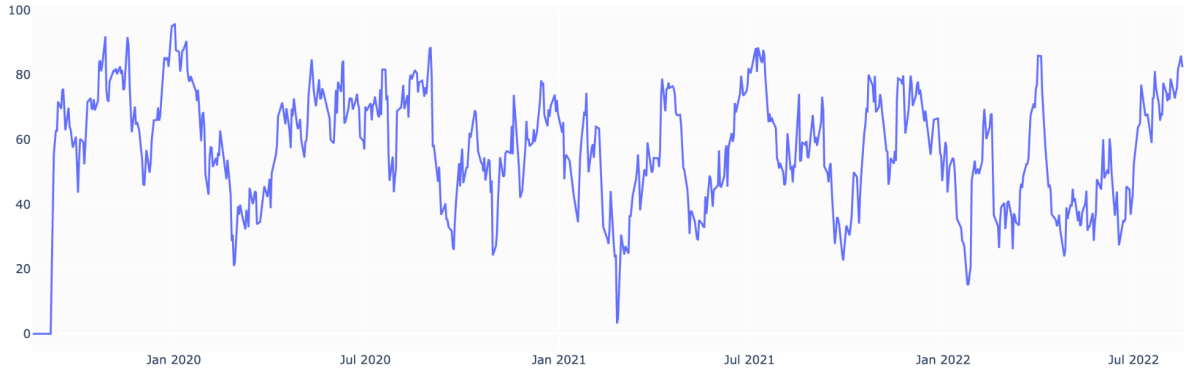


Relative Strength Index

Adding RSI indicator to predict whether a stock is overbought/oversold.

0s

```
def relative_strength_idx(df, n=14):
    close = df['Close']
    delta = close.diff()
    delta = delta[1:]
    pricesUp = delta.copy()
    pricesDown = delta.copy()
    pricesUp[pricesUp < 0] = 0
    pricesDown[pricesDown > 0] = 0
    rollUp = pricesUp.rolling(n).mean()
    rollDown = pricesDown.abs().rolling(n).mean()
    rs = rollUp / rollDown
    rsi = 100.0 - (100.0 / (1.0 + rs))
    return rsi

df['RSI'] = relative_strength_idx(df).fillna(0)
fig = go.Figure(go.Scatter(x=df.Date, y=df.RSI, name='RSI'))
fig.show()
```



MACD

```
EMA_12 = pd.Series(df['Close'].ewm(span=12, min_periods=12).mean())
EMA_26 = pd.Series(df['Close'].ewm(span=26, min_periods=26).mean())
df['MACD'] = pd.Series(EMA_12 - EMA_26)
df['MACD_signal'] = pd.Series(df.MACD.ewm(span=9, min_periods=9).mean())

fig = make_subplots(rows=2, cols=1)
fig.add_trace(go.Scatter(x=df.Date, y=df.Close, name='Close'), row=1, col=1)
fig.add_trace(go.Scatter(x=df.Date, y=EMA_12, name='EMA 12'), row=1, col=1)
fig.add_trace(go.Scatter(x=df.Date, y=EMA_26, name='EMA 26'), row=1, col=1)
fig.add_trace(go.Scatter(x=df.Date, y=df['MACD'], name='MACD'), row=2, col=1)
fig.add_trace(go.Scatter(x=df.Date, y=df['MACD_signal'], name='Signal line'), row=2, col=1)
fig.show()
```

```
[12] df['Close'] = df['Close'].shift(-1)
```

Clean data by removing invalid samples

```
df1=df
```

```
[14] df = df.iloc[33:] # Because of moving averages and MACD line
     df = df[:-1]      # Because of shifting close price

     df.index = range(len(df))
```

Splitting the stock data into:

70% Training

15% Validation

15% Test

```
test_size  = 0.15
valid_size = 0.15

test_split_idx  = int(df.shape[0] * (1-test_size))
valid_split_idx = int(df.shape[0] * (1-(valid_size+test_size)))

train_df  = df.loc[:valid_split_idx].copy()
valid_df  = df.loc[valid_split_idx+1:test_split_idx].copy()
test_df   = df.loc[test_split_idx+1:].copy()
```

```
fig = go.Figure()
fig.add_trace(go.Scatter(x=train_df.Date, y=train_df.Close, name='Training'))
fig.add_trace(go.Scatter(x=valid_df.Date, y=valid_df.Close, name='Validation'))
fig.add_trace(go.Scatter(x=test_df.Date,  y=test_df.Close,  name='Test'))
fig.show()
```

Drop unnecessary columns

```
drop_cols = ['Date', 'Volume', 'Open', 'Low', 'High']

train_df = train_df.drop(drop_cols, 1)
valid_df = valid_df.drop(drop_cols, 1)
test_df  = test_df.drop(drop_cols, 1)
```

Split into features and labels

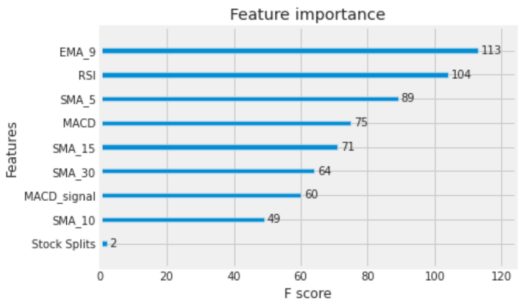
```
[17] y_train = train_df['Close'].copy()
     X_train = train_df.drop(['Close'], 1)

     y_valid = valid_df['Close'].copy()
     X_valid = valid_df.drop(['Close'], 1)

     y_test  = test_df['Close'].copy()
     X_test  = test_df.drop(['Close'], 1)

     X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Dividends      506 non-null    float64
 1   Stock Splits   506 non-null    float64
 2   EMA_9          506 non-null    float64
 3   SMA_5          506 non-null    float64
 4   SMA_10         506 non-null    float64
 5   SMA_15         506 non-null    float64
 6   SMA_30         506 non-null    float64
 7   RSI            506 non-null    float64
 8   MACD           506 non-null    float64
 9   MACD_signal    506 non-null    float64
dtypes: float64(10)
memory usage: 39.7 KB
```

```
[18] model = xgb.XGBRegressor()
     model.fit(X_train, y_train, eval_set=[(X_train, y_train), (X_valid, y_valid)], verbose=False)
```

```
[17:47:41] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
XGBRegressor()
```

```
plot_importance(model);
```

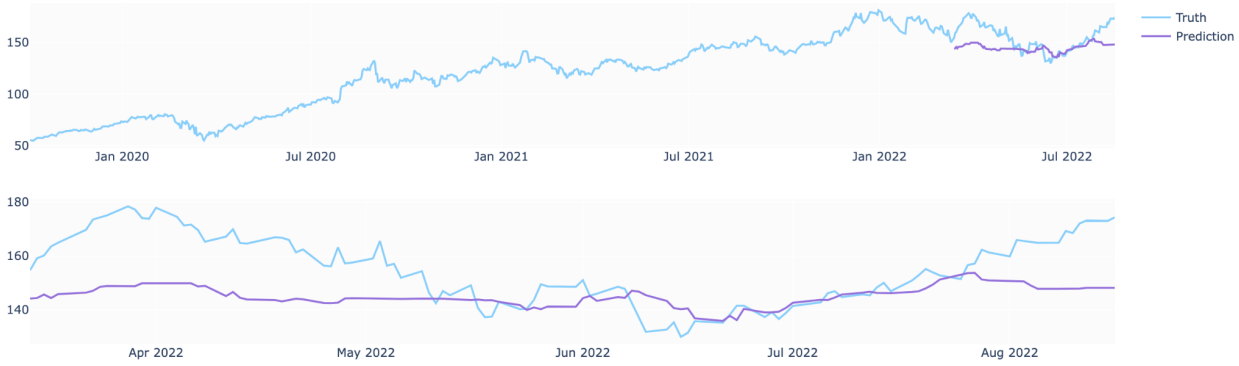


Calculate and visualize predictions

```
y_pred = model.predict(X_test)
print(f'y_true = {np.array(y_test)[:5]}')
print(f'y_pred = {y_pred[:5]}')
print(f'mean_squared_error = {mean_squared_error(y_test, y_pred)}')
accuracy=((y_pred.mean()/y_test.mean()))*100
print("Accuracy:",accuracy)
```

```
y_true = [154.64764404 159.13482666 160.16188049 163.51229858 164.90830994]
y_pred = [144.09009 144.31175 145.61047 144.31175 145.76445]
mean_squared_error = 214.0204712006423
Accuracy: 93.71416003567921
```

```
predicted_prices = df.loc[test_split_idx+1:].copy()
predicted_prices['Close'] = y_pred

fig = make_subplots(rows=2, cols=1)
fig.add_trace(go.Scatter(x=df.Date, y=df.Close,
                         name='Truth',
                         marker_color='LightSkyBlue'), row=1, col=1)

fig.add_trace(go.Scatter(x=predicted_prices.Date,
                         y=predicted_prices.Close,
                         name='Prediction',
                         marker_color='MediumPurple'), row=1, col=1)

fig.add_trace(go.Scatter(x=predicted_prices.Date,
                         y=y_test,
                         name='Truth',
                         marker_color='LightSkyBlue',
                         showlegend=False), row=2, col=1)

fig.add_trace(go.Scatter(x=predicted_prices.Date,
                         y=y_pred,
                         name='Prediction',
                         marker_color='MediumPurple',
                         showlegend=False), row=2, col=1)

fig.show()
```




```
[32] from sklearn.linear_model import LinearRegression
     lin=LinearRegression().fit(X_train,y_train)
```

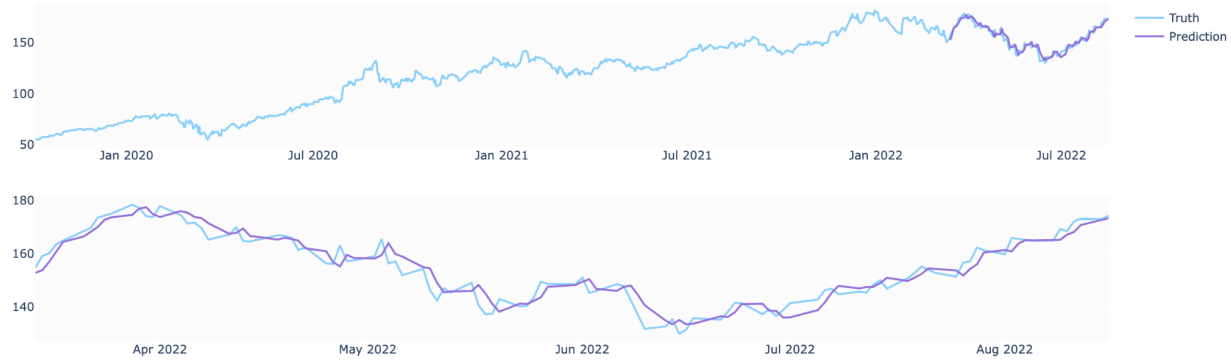
```
[33] y_pred = lin.predict(X_test)
     print(f'y_true = {np.array(y_test)[:5]}')
     print(f'y_pred = {y_pred[:5]}')
     print(f'mean_squared_error = {mean_squared_error(y_test, y_pred)}')
```

```
y_true = [154.64764404 159.13482666 160.16188049 163.51229858 164.90830994]
y_pred = [152.80318531 153.78024103 156.937333   160.64013037 164.41058733]
mean_squared_error = 12.623888910092088
```

```
if(y_pred.mean()>y_test.mean()):
  d=y_pred.mean()-y_test.mean()
  accuracy=((y_test.mean()-d)/y_test.mean())*100
else:
  accuracy=((y_pred.mean()/y_test.mean()))*100
print("Accuracy:",accuracy)
```

```
Accuracy: 99.92706890498474
```

```
predicted_prices = df.loc[test_split_idx+1:].copy()
predicted_prices['Close'] = y_pred

fig = make_subplots(rows=2, cols=1)
fig.add_trace(go.Scatter(x=df.Date, y=df.Close,
                         name='Truth',
                         marker_color='LightSkyBlue'), row=1, col=1)

fig.add_trace(go.Scatter(x=predicted_prices.Date,
                         y=predicted_prices.Close,
                         name='Prediction',
                         marker_color='MediumPurple'), row=1, col=1)

fig.add_trace(go.Scatter(x=predicted_prices.Date,
                         y=y_test,
                         name='Truth',
                         marker_color='LightSkyBlue',
                         showlegend=False), row=2, col=1)

fig.add_trace(go.Scatter(x=predicted_prices.Date,
                         y=y_pred,
                         name='Prediction',
                         marker_color='MediumPurple',
                         showlegend=False), row=2, col=1)

fig.show()
```




```
[28] df2 = pd.DataFrame(columns = ['Dividends', 'Stock Splits', 'EMA_9', 'SMA_5', 'SMA_10', 'SMA_15', 'SMA_30', 'RSI', 'MACD', 'MACD_signal'])
     df2.loc[0] = [0.0,  0.0 ,71,  23, 34, 16, 161 ,20,  -13,-133]
     model.predict(df2)
```

```
array([73.93058], dtype=float32)
```

```
[29] df1['SMA_5'] = df1['Close'].rolling(5).mean().shift()
     df1['SMA_10'] = df1['Close'].rolling(10).mean().shift()
```

Alert System:

```
a=df1['SMA_5'].get(len(df1['SMA_5'])-1)
b=df1['Open'].get(len(df1['Open'])-1)
c=b-a
d=(df1['SMA_10'].get(len(df1['SMA_10'])-1) - df1['Open'].get(len(df1['Open'])-1))
if(c>=3):
  print(c,"is the difference between Moving Average for 5 days and Open, You should Sell")
elif(c<=-3):
   print(c,"is the difference between Moving Average for 5 days and Open, You should Buy")
else:
  print("The difference between the Moving Average for 5 days and Open is",c)
if(d>=3):
  print(d,"is the difference between Moving Average for 10 days and Open, You should Sell")
elif(d<=-3):
   print(d,"is the difference between Moving Average for 10 days and Open, You should Buy")
else:
  print("The difference between the Moving Average for 10 days and Open is",d)
```

```
The difference between the Moving Average for 5 days and Open is 0.5380004882812557
-3.6580017089843864 is the difference between Moving Average for 10 days and Open, You should Buy
```

Make Your Prediction Here:

Enter in the format:
'Dividends', 'Stock Splits', 'EMA_9', 'SMA_5', 'SMA_10', 'SMA_15', 'SMA_30', 'RSI', 'MACD', 'MACD_signal'

```
df2 = pd.DataFrame(columns = ['Dividends', 'Stock Splits', 'EMA_9', 'SMA_5', 'SMA_10', 'SMA_15', 'SMA_30', 'RSI', 'MACD', 'MACD_signal'])
df2.loc[0] = [0.0,  0.0,  154.077849, 161.277979, 157.610074, 154.987383, 148.173846, 72.355610,  5.294613, 4.060149]
lin.predict(df2)
```

```
array([165.10910238])
```

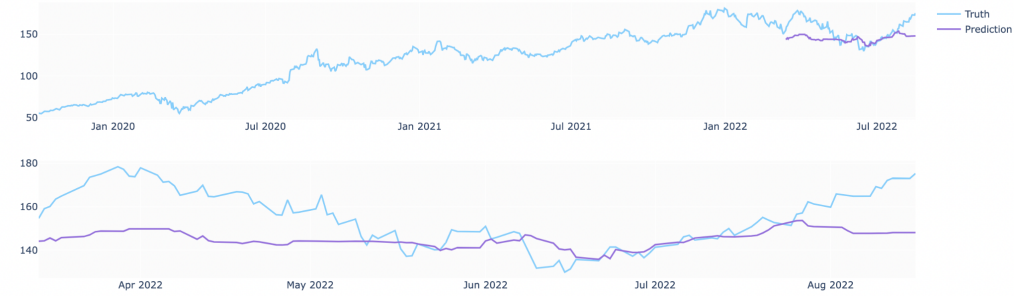
CHAPTER 4

4.0 System Testing Results :

Results From XGBoost Model:

```
y_pred = model.predict(X_test)
print(f'y_true = {np.array(y_test)[:5]}')
print(f'y_pred = {y_pred[:5]}')
print(f'mean_squared_error = {mean_squared_error(y_test, y_pred)}')
accuracy=((y_pred.mean()/y_test.mean()))*100
print("Accuracy:",accuracy)
```

```
y_true = [154.64764404 159.1348114  160.16188049 163.51228333 164.90830994]
y_pred = [144.09009 144.31175 145.61049 144.31175 145.76447]
mean_squared_error = 214.5006540973964
Accuracy: 93.70872677042027
```



Results From Linear Regression Model:

```
[57] y_pred = lin.predict(X_test)
     print(f'y_true = {np.array(y_test)[:5]}')
     print(f'y_pred = {y_pred[:5]}')
     print(f'mean_squared_error = {mean_squared_error(y_test, y_pred)}')
```

```
y_true = [154.64764404 159.1348114  160.16188049 163.51228333 164.90830994]
y_pred = [152.80318503 153.78024105 156.93732475 160.64012536 164.4105743 ]
mean_squared_error = 12.650712578960258
```

CHAPTER 5

5.0 Conclusion :

The accuracy achieved from the XGBoost model is 93.7% and a mean squared error of 214.5 which is rectified by the Linear Regression model which achieved an accuracy of 99.9% and a mean squared error of 12.6. The Linear Regression model has a higher accuracy due to the fact that XGBoost cannot extrapolate data properly and also because the data points from stock market dataset are linearly correlated.

CHAPTER 6

6.0 References :

[1] https://www.kaggle.com/

[2] https://finance.yahoo.com/

[3] https://machinelearningmastery.com/xgboost-for-regression/

[4] https://www.javatpoint.com/linear-regression-in-machine-learning

[5] https://www.geeksforgeeks.org/