SAVITRIBAI PHULE PUNE UNIVERSITY

**A PROJECT REPORT ON**

**Algorithmic Trading**

# BY

Kashyap Mavani **(72219119H)** Pranjal Mulay (**72219128G)** Vaishnavi Jadhav **(72219053M)**

Under the Guidance of **Prof. R.M. Samant**

In partial fulfillment of **S.T.E.S’s**

### NBN SINHGAD TECHNICAL INSTITUTES CAMPUS, PUNE-41

**B.E (INFORMATION TECHNOLOGY)**

**Savitribai Phule Pune University**

# May-June 2024-2025



**NBN SINHGAD TECHNICAL INSTITUTES CAMPUS, PUNE-41 DEPARTMENT OF INFORMATION TECHNOLOGY**

1

**CERTIFICATE**

This is to certify that the Project entitled **“Algorithmic Trading”**

Submitted by

**Kashyap Mavani (72219119H) Pranjal Mulay (72219128G) Vaishnavi Jadhav (72219053M)**

Is record of bonafide work carried out by him/her, under my guidance, in partial fulfillment of the requirement for the award of the Degree of Bachelor of Engineering (Information Technology) of Savitribai Phule Pune University

**Date:**

**Place: NBN Sinhgad School Of Engineering, Pune-41**

## Prof. R. M. Samant Prof. R.M. Samant

**(Internal Guide) Head, Dept. of Information Technology**

### Prof. Dr. Shivprasad. P. Patil

**External Examiner PRINCIPAL**

# ACKNOWLEDGEMENT

The entire session of partial project completion phase so far was a great experience, providing us with great insight and innovation into learning various software engineering concepts and benefits of any software project, people are the most important assets; our project would not materialize without the cooperation of many of the people involved

We are very much thankful to our respected project guide **Prof. R.M Samant** for his personal involvement and constructive suggestion through the project schedule.He took deep interest in correcting the minor problems and guided as through our journey so far.

We are sincerely thankful to our **(Principle) Dr. Shivprasad. P. Patil** Sir and **(H.O.D) Prof. R.M. Samant Sir** for their valuable recommendation and suggestions

We express our deep gratitude to all staff members of our department and special thanks to **Prof. R.M Samant** for providing us needed help.

**Kashyap Mavani (72219119H) Pranjal Mulay (72219128G) Vaishnavi Jadhav (72219053M)**

# ABSTRACT

The stock market is a fast-moving, data-driven environment where traders must make quick and informed decisions to maximize profits and minimize risks. This project focuses on developing a system that predicts stock market movements by applying algorithmic techniques and moving averages, leveraging live data from the yfinance module. Unlike automated trading systems, this tool provides valuable market analysis to assist traders in making their own decisions. By offering predictions for both long-term investments and intraday trading, the system accommodates a variety of trading strategies.

The predictive models use a combination of technical analysis indicators, such as moving averages, to identify trends and enhance the accuracy of predictions. The system’s ability to process real-time stock data ensures that users have access to the most current market information. In addition to the core prediction functionality, the project integrates a Telegram bot that sends live notifications based on the system’s analysis, allowing traders to act swiftly in response to market changes. This notification feature reduces the need for constant market monitoring, giving traders the flexibility to manage their time effectively while staying updated.

The project is designed to be user-friendly and caters to both novice and experienced investors. By providing accurate and timely predictions, this system empowers traders to make better-informed decisions in the highly volatile stock market. Ultimately, this project aims to bridge the gap between traditional market analysis and real-time data processing, delivering a comprehensive tool for enhanced trading performance.

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO.** | **TABLE OF CONTENTS**  **TITLE** | **PAGE NO** |
|  | **ABSTRACT** | **7** |
| **1** | **INTRODUCTION** | **8** |
|  | 1.1 Background / Context | 8 |
|  | 1.2 Purpose of the project | 8 |
|  | 1.3 Motivation | 9 |
|  |  |  |
| **2** | **LITERATURE REVIEW** | **10** |
| **3** | **METHODOLOGY** | **11** |
|  | 3.1 Introduction to Methodology | 11 |
|  | 3.2 Data Collection | 11 |
|  | 3.3 Data Structuring | 11 |
|  | 3.4 System Architecture | 12 |
|  | 3.5 Algorithms and Techniques | 12 |
|  | 3.6 Prediction and Indicator Calculation  3.7 Notification System  3.8 System Testing and Performance  3.9 Limitations | 12  13  13  13 |
|  | 3.10 Libraries Used | 14 |
|  | 3.11 Architecture | 15 |
|  | 3.12 Flow Diagram | 16 |
|  |  |  |

1. [CONCLUSION 17](#_TOC_250002)
2. [REFERENCES 18](#_TOC_250001)
3. [OUTPUT SCREENSHOTS 20](#_TOC_250000)
4. APPENDIX 21
5. PUBLICATION & PLAGIARISM

REPORT 35

# ABSTRACT

# 

The stock market is a fast-moving, data-driven environment where traders must make quick and informed decisions to maximize profits and minimize risks. This project focuses on developing a system that predicts stock market movements by applying algorithmic techniques and moving averages, leveraging live data from the yfinance module. Unlike automated trading systems, this tool provides valuable market analysis to assist traders in making their own decisions. By offering predictions for both long-term investments and intraday trading, the system accommodates a variety of trading strategies.

The predictive models use a combination of technical analysis indicators, such as moving averages, to identify trends and enhance the accuracy of predictions. The system’s ability to process real-time stock data ensures that users have access to the most current market information. In addition to the core prediction functionality, the project integrates a Telegram bot that sends live notifications based on the system’s analysis, allowing traders to act swiftly in response to market changes. This notification feature reduces the need for constant market monitoring, giving traders the flexibility to manage their time effectively while staying updated.

The project is designed to be user-friendly and caters to both novice and experienced investors. By providing accurate and timely predictions, this system empowers traders to make better-informed decisions in the highly volatile stock market. Ultimately, this project aims to bridge the gap between traditional market analysis and real-time data processing, delivering a comprehensive tool for enhanced trading performance.

# INTRODUCTION

# 

In today's fast-moving stock market, traders face the challenge of processing vast amounts of data to make timely and informed decisions. Algorithmic trading, which uses algorithms to analyze market trends and predict future movements, has gained popularity but often automates trades, limiting trader control. This project aims to bridge that gap by developing a system that provides predictive insights for long-term and intraday trading without automating trades. Using live stock data from the yfinance module and technical analysis techniques like moving averages, the system predicts market movements and sends real-time notifications via a Telegram bot, empowering traders to make better-informed decisions.

# BACKGROUND / CONTEXT

The financial markets are driven by complex and rapidly changing factors, from economic indicators to geopolitical events. Traders often struggle to keep up with these changes while trying to analyze large amounts of data to make profitable decisions. Algorithmic trading has emerged as a solution that leverages algorithms to analyze market data and predict trends, enabling traders to navigate this complexity more effectively.

# In this context, this project focuses on creating a predictive tool to assist traders in making decisions based on data analysis, filling the need for a system that supports informed decision-making without taking control away from the trader.

# PURPOSE OF THE PROJECT

# This project seeks to create a stock market prediction tool that uses algorithmic techniques to help traders predict both long-term investments and short-term trading opportunities. The purpose is to bridge the gap between real-time data analysis and human decision-making.

# By offering predictive insights and real-time notifications, the system aims to help traders make timely, data-driven decisions in a highly dynamic market.

# MOTIVATION

# In today’s dynamic stock market, traders face the challenge of rapidly processing and interpreting massive amounts of data to make effective trading decisions. Traditional trading methods can be time-consuming and susceptible to emotional bias, while automated systems can limit a trader’s control over their strategies. This project is motivated by the need for a tool that bridges these gaps—empowering traders with predictive insights without automating their decisions. By leveraging real-time data and algorithmic analysis, this system allows traders to make informed choices promptly, improving accuracy and responsiveness. The integration of real-time notifications further motivates the system’s development, as it enables traders to stay updated without constant monitoring, creating a user-friendly yet powerful tool for both novice and experienced traders.

# LITERATURE REVIEW

The application of technical indicators in algorithmic trading has become increasingly popular due to their ability to identify market trends, predict price movements, and provide actionable insights. Research on this topic has demonstrated the effectiveness of indicators like Simple Moving Average (SMA), Exponential Moving Average (EMA), and Relative Strength Index (RSI) in various trading strategies.

In *Algorithmic Trading using Technical Indicators* (IEEE, 2021), the authors explored a range of technical indicators, including SMA, EMA, and RSI, to analyze and predict trends in financial markets. The study presented the indicators’ effectiveness in both long- and short-term forecasting, emphasizing their role in smoothing price fluctuations and identifying entry and exit points. By leveraging these indicators within algorithmic frameworks, traders can improve the precision of their trades and minimize risks. The study underscored the importance of EMA in capturing recent price trends due to its weighting of recent prices, making it more responsive than SMA in highly volatile markets. RSI was highlighted for its utility in identifying overbought or oversold conditions, which can serve as signals for trend reversals.

Further analysis of these moving averages was provided in *Understanding Stock Market Trends Using SMA and EMA* (IEEE, 2023), which delved deeper into the mechanics and applications of SMA and EMA. The authors argued that SMA offers a reliable approach for smoothing out short-term fluctuations, providing traders with a clearer view of the underlying trend. However, they noted that SMA lags behind price changes due to equal weight distribution across periods, which can lead to delayed signals. Conversely, EMA was shown to reduce this lag, offering more timely insights by placing a greater emphasis on recent data points. The paper concluded that while SMA is effective in stable market conditions, EMA may be more suitable for markets prone to sudden movements, making it especially relevant for intraday or swing trading.

A broader application of moving averages in stock prediction is presented in *Stock Price Prediction using Moving Average Time Series* (IJRPR, 2023). This paper demonstrated how moving averages, particularly SMA, can be applied to historical price data to forecast future trends, offering traders a simple yet powerful tool for making data-driven decisions. The authors applied SMA across various time frames, concluding that longer periods yield more reliable signals for long-term investments, while shorter periods better capture short-term trends. They also discussed potential limitations, such as SMA’s inability to account for market sentiment or external factors, suggesting that combining SMA with other indicators like EMA or RSI could enhance predictive accuracy.

Lastly, *Evaluating the Impact of Technical Indicators on Stock Forecasting* (ResearchGate, 2021) examined how combining multiple technical indicators could improve the robustness and reliability of stock price predictions. The study assessed several indicators, including RSI, EMA, and SMA, on their own and in combination. The authors observed that while each indicator individually offers valuable insights, combining them can provide a more comprehensive view of market conditions, leading to better-informed decisions. For instance, combining EMA with RSI allows for the detection of trends and momentum while also identifying potential overbought or oversold conditions, making the trading signals more robust. The research further noted that this multi-indicator approach reduces false signals, especially in volatile markets, by cross-verifying the information provided by each indicator.

These studies validates the effectiveness of SMA, EMA, and RSI in both short- and long-term analysis. Together, these indicators allow for a nuanced approach to algorithmic trading, where each serves a complementary role. SMA and EMA help identify trends, with EMA offering greater responsiveness, while RSI adds an extra layer by detecting overbought and oversold conditions. This literature establishes the relevance of these indicators in modern trading algorithms, suggesting that combining them in this project could enhance its predictive capabilities and reliability. The insights derived from these papers will be instrumental in implementing and fine-tuning the trading algorithms in the project.

# METHODOLOGY

**3.1 Introduction to Methodology**

* **Overview**: This methodology details the steps for real-time data collection, transfer, and processing to predict stock market trends. By leveraging data-driven technical indicators, the project aims to support informed trading decisions without automating trades.
* **Objective**: To systematically collect and analyze live stock market data, focusing on key technical indicators (EMA, SMA, and RSI) to generate actionable predictions for both short-term and long-term trading.

# DATA COLLECTION

* **Data Sources**: Stock data is sourced in real time using the yfinance Python module with specified symbols, including indices such as "^NSEI" and cryptocurrencies like "BTC-USD."
* **Implementation**: A Python script retrieves stock data using yfinance at a specified interval, downloading specified days of data for each update.
* **Data Transmission**: The data is then converted into JSON format and transmitted to a Node environment via a WebSocket connection for real-time analysis and further processing.

# DATA STRUCTURING

* **Array Formation**: Once the data is cleaned, the relevant fields (OPEN, CLOSE, HIGH, LOW) are extracted from the JSON format and stored in separate arrays. This structuring is crucial for efficient calculation and analysis.

**3.4 SYSTEM ARCHITECTURE**

* **Overview of Components**: The architecture consists of a data collection module (Python), a data transfer module (WebSocket), and an analysis module (Node environment). Data collected from yfinance is sent in JSON format to the Node environment, where further calculations and indicators are derived.
* **Process Flow**: Data from yfinance is processed in a loop to maintain real-time data flow, converted into JSON, and transferred to Node.js. Here, OPEN, CLOSE, HIGH, and LOW values are separated into arrays, allowing for efficient indicator calculations.

**3.5 ALGORITHMS AND TECHNIQUES**

* **Technical Indiactors Used**: The CLOSE values are used to calculate the Exponential Moving Average (EMA), Simple Moving Average (SMA), and Relative Strength Index (RSI). These indicators help assess price trends, momentum, and potential reversal points.
* **Implementation of Calculations**: EMA, SMA, and RSI are calculated in the Node environment using the CLOSE array. Each indicator is computed to assess short- and long-term trends, aiding in the generation of buy or sell signals.

**3.6 PREDICTION AND INDICATOR CALCULATION**

* **EMA and SMA Calculation**: Using the CLOSE values, the system calculates EMA and SMA to smooth price data and identify trends. These moving averages help in determining entry and exit points.
* **RSI Calculation**: The RSI calculation assesses momentum, identifying overbought or oversold market conditions. This complements the moving averages by signaling potential trend reversals.

**3.7 NOTIFICATION SYSTEM**

* **Real-Time Alerts**: A Telegram bot sends notifications based on calculated signals from EMA, SMA, and RSI. When significant market conditions (e.g., overbought or oversold) are detected, the bot alerts users, enabling timely decision-making.
* **Alert Conditions**: Notifications are triggered based on predefined thresholds for the indicators, such as EMA/SMA crossovers and RSI levels.

**3.8 SYSTEM TESTING AND PERFORMANCE**

* **Testing Data Flow and Accuracy:** The WebSocket connection is stress-tested for data continuity, and the accuracy of the EMA, SMA, and RSI calculations is validated against historical data.
* **Performance Metrics:** Performance metrics are tracked, focusing on the reliability of predictions, latency of data updates, and promptness of notifications to ensure effective real-time functionality.

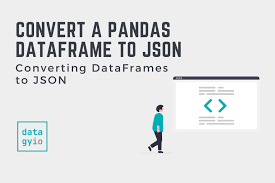
**3.9 LIMITATIONS**

* **Data Dependency**: The project’s predictive accuracy depends on the reliability of yfinance data and WebSocket connections. Unpredictable market events may also affect the accuracy of the predictions.
* **Scope of Analysis**: The project focuses on technical analysis through moving averages and RSI and does not account for other factors like fundamental analysis or external economic conditions.

# 3.10 LIBRARIES USED

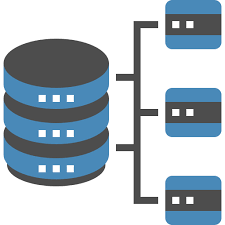
1. **Python Libraries**:
   * **yfinance**: Used to download real-time and historical stock market data. This library provides an easy interface to fetch data for various financial instruments, including stocks, indices, and cryptocurrencies.
   * **Pandas**: Employed for data manipulation and analysis. Pandas is essential for handling the data frames, cleaning the data, and structuring it for further processing.
   * **NumPy**: Utilized for numerical operations on arrays. NumPy supports efficient array manipulations required for calculating technical indicators like EMA, SMA, and RSI.
   * **Asyncio**: Used to handle asynchronous programming, enabling the WebSocket server to manage real-time data streams without blocking operations.
   * **Websockets**: Facilitates real-time communication between the Python environment and the Node.js server. This library is crucial for transmitting JSON data over WebSocket connections.
2. **Node.js Libraries**:
   * **WebSocket**: Utilized for setting up the WebSocket server to receive data from the Python script. This library is integral for real-time data transfer and communication.
   * **Technicalindicators**: A Node.js library used for computing various technical indicators. This library supports the calculation of EMA, SMA, RSI, and other indicators required for market analysis.
   * **Axios**: Employed for making HTTP requests if needed for additional data fetching or API interactions within the Node.js environment.
   * **Express**: Used to set up a basic server if required for handling API requests or serving client-side applications.
3. **Telegram Bot API**:
   * **Python-Telegram-Bot**: A Python wrapper for the Telegram Bot API, used to send real-time notifications to users. This library enables seamless integration with Telegram to deliver alerts based on the system’s predictions.

# ARCHITECTURE DIAGRAM



**2. Converting the Pandas Dataframe to JSON**

**1. Fetching Live Data from Yahoo finance**



**3. Passing the Data to different Algorithms for Prediction**

**4. Live Prediction made is notified to user on Telegram**

* 1. **FLOW DIAGRAM**

YFinance API Call

If passed data satisfies the Algorithm or not

Pass Data through Algorithms

Data Conversion to JSON & sending it via WebSocket

Receive data in Node.js Server

Data Extraction & Calculate Technical Indicator

Send notification to Telegram with Prediction

NO

**YES**

# CONCLUSION

The algorithmic trading system developed in this project effectively leverages real-time stock market data and advanced technical analysis to provide valuable insights and predictions. By integrating the yfinance module for data collection, websockets for real-time communication, and technical indicators for market analysis, the system offers a robust solution for both novice and experienced investors. The use of the Exponential Moving Average (EMA), Simple Moving Average (SMA), and Relative Strength Index (RSI) enhances the accuracy of market trend predictions.

The inclusion of a Telegram bot for real-time notifications ensures that users are promptly informed of significant market movements, enabling timely and informed decision-making. Although this system does not automate trades, it serves as a powerful tool for market analysis and strategy development.

Overall, this project demonstrates the potential of combining financial data with modern programming techniques to create a sophisticated trading aid. Future enhancements could include expanding the range of technical indicators, incorporating machine learning models for more advanced predictions, and developing a user-friendly dashboard for better visualization of market data and trends. This project lays a solid foundation for further advancements in algorithmic trading and financial analysis.

# REFERENCES

1. **Data Source**:
   1. Yahoo Finance. (n.d.). Retrieved from <https://finance.yahoo.com>
      1. Source for historical and real-time stock market data used in the project.
2. **Libraries and Tools**:
   1. yfinance Documentation. (n.d.). Retrieved from <https://pypi.org/project/yfinance/>
      1. Provides detailed information on how to use the yfinance library for fetching financial data.
   2. Pandas Documentation. (n.d.). Retrieved from https://pandas.pydata.org/docs/
      1. Comprehensive guide on using Pandas for data manipulation and analysis.
   3. NumPy Documentation. (n.d.). Retrieved from https://numpy.org/doc/
      1. Documentation on NumPy for numerical operations on arrays.
   4. asyncio Documentation. (n.d.). Retrieved from <https://docs.python.org/3/library/asyncio.html>
      1. Official Python documentation for handling asynchronous programming with asyncio.
   5. websockets Documentation. (n.d.). Retrieved from <https://websockets.readthedocs.io/>
      1. Guide for using the websockets library for real-time communication in Python.
      2. Technicalindicators Documentation. (n.d.). Retrieved from <https://www.npmjs.com/package/technicalindicators>
   6. Node.js library documentation for calculating various technical indicators.
   7. Python-Telegram-Bot Documentation. (n.d.). Retrieved from <https://python-telegram-bot.readthedocs.io/>
      1. Guide on using the Python-Telegram-Bot library for creating and managing Telegram bots.
3. **Articles**:

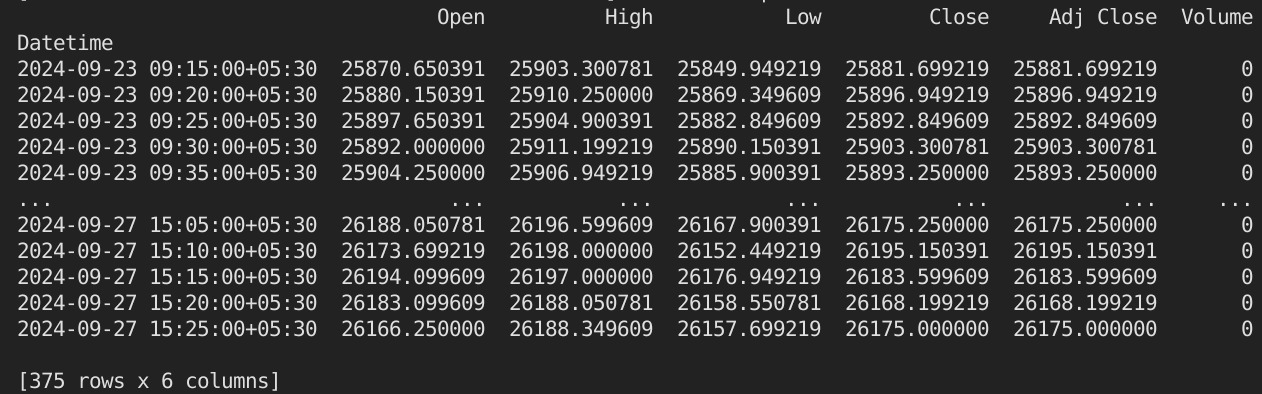
* Investopedia. (n.d.). *Exponential Moving Average (EMA)*. Retrieved from https://www.investopedia.com/terms/e/ema.asp
  + Detailed explanation of the Exponential Moving Average (EMA) and its applications in trading.
* Investopedia. (n.d.). *Simple Moving Average (SMA)*. Retrieved from

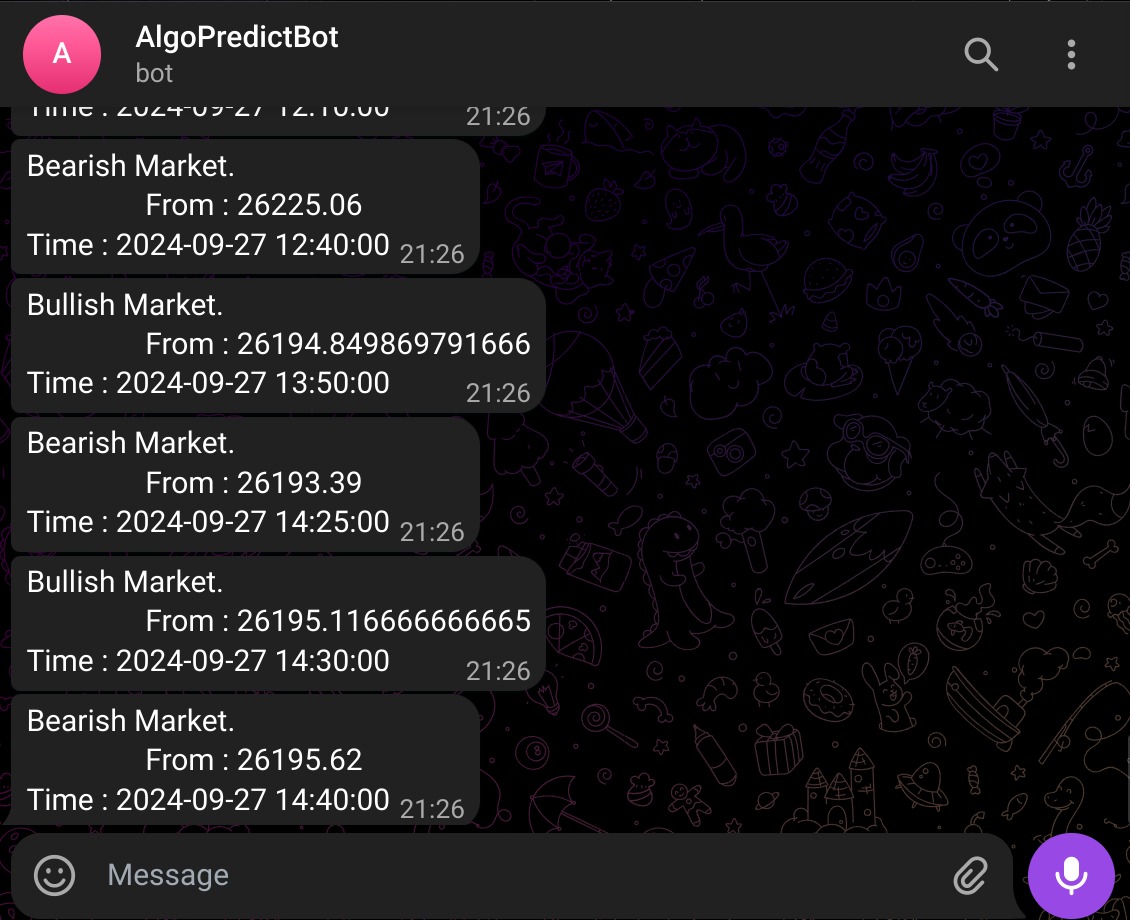
https://www.investopedia.com/terms/s/sma.asp

* + Overview of the Simple Moving Average (SMA) and how it is used in technical analysis.
* Investopedia. (n.d.). *Relative Strength Index (RSI)*. Retrieved from https://www.investopedia.com/terms/r/rsi.asp
  + Explanation of the Relative Strength Index (RSI) and its role in trading strategies.

1. **Research papers**:

* Algorithmic Trading using Technical Indicators. Retrieved from <https://ieeexplore.ieee.org/document/9510135>
* Understanding Stock Market Trends Using Simple Moving Average (SMA) and Exponential Moving Average (EMA) Indicators. Retrieved from <https://ieeexplore.ieee.org/document/10398001>
* Stock Price Prediction using Moving Average Time Series. Retrieved from <https://ijrpr.com/uploads/V4ISSUE6/IJRPR14104.pdf>
* Evaluating the impact of technical indicators on stock forecasting. Retrieved from <https://www.researchgate.net/publication/313802735_Evaluating_the_impact_of_technical_indicators_on_stock_forecasting>
  1. **OUTPUT SCREENSHOTS**





# APPENDIX

* + 1. **Server Initiation & WebSocket Connection**

**server.py**

This script sets up a WebSocket server that streams live financial data from Yahoo Finance. It fetches stock data for a specified symbol, converts the data to JSON format, and sends it to connected clients at regular intervals.

# import asyncio

# import websockets

# import pandas as pd

# import yfinance as yf

# import time

# async def handler(websocket):

# async for message in websocket:

# print(f"Received message from client: {message}")

# SYMBOL = "^NSEI" # any symbol from yahoo finance "^NSEI","^NSEBANK","BTC-USD"

# while True:

# data = yf.download(SYMBOL,period ="2d",interval = "5m")

# print(data)

# data = data.to\_json()

# 

# await websocket.send(data)

# min = 5 # 1440 minutes <- 1d ,

# time.sleep(60 \* min)

# start\_server = websockets.serve(handler, "localhost", 3000)

# asyncio.get\_event\_loop().run\_until\_complete(start\_server)

# asyncio.get\_event\_loop().run\_forever()

# client.js

# This script establishes a WebSocket connection to a server running at ws://localhost:3000/. Once connected, it sends a greeting message to the server. The script listens for incoming messages, which contain stock data in JSON format. Upon receiving data, it parses the JSON and stores it in a variable named Data. It then calls the processDataAndRunAlgo function from an external module (./utilities/ProcessData) to process the received data and execute trading algorithms. This setup enables real-time data processing and algorithm execution.

const WebSocket = require('ws');

const { processDataAndRunAlgo } = require('./utilities/ProcessData')

let Data;

const ws = new WebSocket("ws://localhost:3000/"); // Replace with your server URL and port

ws.onopen = function() {

console.log("WebSocket connection opened!");

// You can send data to the server now

ws.send("Hello from JavaScript!");

};

ws.onmessage = async function(event) {

let message = await event.data;

message = await JSON.parse(message)

Data = message

// console.log(Data);

await processDataAndRunAlgo(Data)

};

* + 1. **Data Extraction & Calling Algorithms**

**ProcessData.js**

This script processes financial data received via WebSocket and runs various technical analysis algorithms. The data includes Open, Close, High, Low prices, and timestamps. The script performs the following:

**Data Extraction**: Extracts and stores Open, Close, High, Low values, and timestamps from the received data.

**Technical Indicator Calculation**: Calculates Simple Moving Average (SMA), Exponential Moving Average (EMA), and Relative Strength Index (RSI) for the Close values.

**Algorithm Execution**: Calls different trading algorithms (SMA\_EMA\_Algo, LIVE\_SMA\_EMA\_Algo, EMA\_Algo, RSI\_Algo) with the calculated indicators and timestamps to analyze market trends and make predictions.

const { calculateSMA } = require('./SMA');

const { calculateEMA } = require('./EMA');

const { TimeStampCoverter } = require('./TimeStampConverter');

const { SMA\_EMA\_Algo } = require('../Algorithims/SMA\_EMA\_Algo');

const { LIVE\_SMA\_EMA\_Algo } = require('../Algorithims/LIVE\_SMA\_EMA\_ALGO');

const { EMA\_Algo } = require('../Algorithims/EMA(5)\_Algo');

const { calculateRSI } = require('./RSI');

const { RSI\_Algo } = require('../Algorithims/RSI\_Algo');

async function processDataAndRunAlgo(Data) {

let OpenValues = [];

let CloseValues = [];

let HighValues = [];

let LowValues = [];

let TimeStamp =[];

setTimeout(() => {

const { Open, Close, High, Low } = Data; // destructuring Data object

//------- storing TimeStamp values in array----------------------------------------------

for (const key in Open) {

const value = TimeStampCoverter(key);

TimeStamp.push(value);

}

//------- storing Open values in array----------------------------------------------

for (const key in Open) {

const value = Open[key];

OpenValues.push(value);

}

//------- storing Close values in array----------------------------------------------

for (const key in Close) {

const value = Close[key];

CloseValues.push(value);

}

//------- storing High values in array----------------------------------------------

for (const key in High) {

const value = High[key];

HighValues.push(value);

}

//------- storing Low values in array----------------------------------------------

for (const key in Low) {

const value = Low[key];

LowValues.push(value);

}

const ndaysSMA = 15;

const ndaysEMA = 5;

const period = 14;

const smaSeries = calculateSMA(CloseValues, ndaysSMA);

const emaSeries = calculateEMA(CloseValues, ndaysEMA);

const RSI\_Values = calculateRSI(CloseValues,period);

// const RSI\_Values\_Tailed = RSI\_Values.slice(-75); // -75 values for 5 min 2 day , -300 values for 5 min 5 day

// const Close\_Values\_Tailed = CloseValues.slice(-75); // -75 values for 5 min 2 day , -300 values for 5 min 5 day

// const TimeStamp\_Tailed = TimeStamp.slice(-75); // -75 values for 5 min 2 day , -300 values for 5 min 5 day

// console.log(OpenValues); // total 75 values for 5 min 1 day

// console.log(CloseValues); // total 75 values for 5 min 1 day

// console.log(HighValues); // total 75 values for 5 min 1 day

// console.log(LowValues); // total 75 values for 5 min 1 day

// console.log(smaSeries); // total 71 values for 5 min 1 day

// console.log(emaSeries) // total 71 values for 5 min 1 day

// console.log(RSI\_Values\_Tailed); // total 61 values for 5 min 1 day

// console.log(Close\_Values\_Tailed);

// console.log(TimeStamp\_Tailed);

// ------------------CALLING ALGORITHMS---------------------------------//

SMA\_EMA\_Algo(smaSeries, emaSeries, TimeStamp);

// LIVE\_SMA\_EMA\_Algo(smaSeries, emaSeries, TimeStamp)

// EMA\_Algo(CloseValues,OpenValues,HighValues, LowValues, emaSeries, TimeStamp)

// RSI\_Algo(RSI\_Values\_Tailed, Close\_Values\_Tailed, TimeStamp);

}, 3000);

}

module.exports = {processDataAndRunAlgo};

* + 1. **Calculating Values of Moving Averages (EMA, SMA) & RSI**

**EMA.js**

This script calculates the Exponential Moving Average (EMA) for a given set of financial data. It uses the exponential-moving-average library to perform the calculation. The function calculateEMA takes in an array of price data and the number of days for the EMA calculation (ndaysEMA). It processes the data to produce an array of EMA values, which are then returned. This allows for the smoothing of price data to help identify trends and make more informed trading decisions.

const ema = require('exponential-moving-average');

function calculateEMA(data, ndaysEMA) {

let emaValues = [];

let EMAValues = ema(data,ndaysEMA);

for(let value in EMAValues){

emaValues.push(Number(EMAValues[value]))

}

return emaValues;

}

module.exports ={calculateEMA}

**SMA.js**

This script calculates the Simple Moving Average (SMA) for a given set of financial data. The calculateSMA function takes an array of price data and the number of days (ndays) for the moving average calculation. It computes the average of the price data within a sliding window of the specified length and returns an array of SMA values. This function helps in smoothing out price data to identify trends and make informed trading decisions.

function calculateSMA(data, ndays) {

const sma = [];

for (let i = ndays - 1; i < data.length; i++) {

const window = data.slice(i - ndays + 1, i + 1);

const average = window.reduce((sum, price) => sum + price, 0) / ndays;

// const average = data.reduce((sum, price) => sum + price, 0) / ndays;

sma.push(average);

}

return sma;

}

module.exports = {calculateSMA};

**RSI.js**

This script calculates the Relative Strength Index (RSI) for a given array of closing prices. The calculateRSI function takes an array of closing prices and a specified period (default is 14). It computes the daily price changes, separates gains and losses, and calculates the average gain and loss. The RSI is then calculated using a smoothing technique for both the initial and subsequent periods, resulting in an array of RSI values. This indicator helps traders assess market momentum and potential reversal points.

function calculateRSI(closingPrices, period = 14) {

if (closingPrices.length < period) {

throw new Error("The list of closing prices must be longer than the period.");

}

// Calculate daily price changes

const priceChanges = closingPrices.slice(1).map((price, index) => price - closingPrices[index]);

// Separate gains and losses

const gains = priceChanges.map(change => (change > 0 ? change : 0));

const losses = priceChanges.map(change => (change < 0 ? -change : 0));

// Calculate the average gain and loss for the initial period

let avgGain = gains.slice(0, period).reduce((acc, val) => acc + val, 0) / period;

let avgLoss = losses.slice(0, period).reduce((acc, val) => acc + val, 0) / period;

// Initialize an array to hold RSI values

const rsiValues = [];

// Calculate RSI for the first period

let rs = avgGain / avgLoss || 0;

let rsi = 100 - 100 / (1 + rs);

rsiValues.push(rsi);

// Calculate RSI for the remaining periods

for (let i = period; i < closingPrices.length - 1; i++) {

const currentGain = gains[i];

const currentLoss = losses[i];

// Update the average gain and loss using the smoothing technique

avgGain = ((avgGain \* (period - 1)) + currentGain) / period;

avgLoss = ((avgLoss \* (period - 1)) + currentLoss) / period;

// Calculate RSI

rs = avgGain / avgLoss || 0;

rsi = 100 - 100 / (1 + rs);

rsiValues.push(rsi);

}

return rsiValues;

}

module.exports = { calculateRSI };

**TimeStampCoverter.js**

This script provides a function called TimeStampCoverter that converts a timestamp in milliseconds into a human-readable date and time format. It creates a JavaScript Date object from the provided timestamp, extracts the year, month, day, hours, minutes, and seconds, and formats these components into a string. The result is a date string formatted as YYYY-MM-DD HH:MM:SS, making it suitable for displaying timestamps in a more user-friendly way.

function TimeStampCoverter(time) {

timestamp = Number(time)

const utcDate = new Date(timestamp);

const istDate = new Date(utcDate.getTime());

// Get the year, month (0-indexed), day, hours, minutes, and seconds

const year = istDate.getFullYear();

const month = String(istDate.getMonth() + 1).padStart(2, '0'); // Add padding for single-digit months

const day = String(istDate.getDate()).padStart(2, '0');

const hours = String(istDate.getHours()).padStart(2, '0');

const minutes = String(istDate.getMinutes()).padStart(2, '0');

const seconds = String(istDate.getSeconds()).padStart(2, '0');

// Format the date and time in a human-readable way

const formattedTime = `${year}-${month}-${day} ${hours}:${minutes}:${seconds}`;

return formattedTime;

}

module.exports = {TimeStampCoverter};

* + 1. **Telegram Bot**

**bot.js**

This script sets up a Telegram bot using the node-telegram-bot-api library. It initializes the bot with a token and provides functionality to send messages to a specific chat ID. The bot can be extended to handle incoming messages, although this functionality is commented out. The main function, sendMessage, sends a given message to the specified chat ID and logs the success or error of the operation. This setup facilitates real-time notifications or alerts via Telegram.

require('dotenv').config();

const TelegramBot = require('node-telegram-bot-api');

const TOKEN = '6387199944:AAGhfpZ8pnt5kUW74ANNlyYJ8rjg9Ri4fv0';

const bot = new TelegramBot(TOKEN,{polling: true});

function sendMessage(message) {

const chatId = 494298168;

bot.sendMessage(chatId, message)

.then(() => {

console.log('Message sent successfully!');

})

.catch((error) => {

console.error('Error sending message:', error);

});

}

module.exports = {sendMessage};

* + 1. **Algorithms**

**SMA\_EMA\_Algo.js**

This script implements a trading algorithm that analyzes Simple Moving Average (SMA) and Exponential Moving Average (EMA) series to identify bullish and bearish market conditions. It iterates through the EMA series, checking conditions to determine market trends. When a bullish condition is detected, it sends a notification message via a Telegram bot and logs the event. Similarly, if a bearish condition is identified, it also sends a notification and logs the information. The algorithm includes a sleep function to pause between message sends, enhancing the readability of logs and preventing message spamming.

const { sendMessage } = require('../Tele-Bot/bot')

require('dotenv').config();

function sleep(ms) {

return new Promise(resolve => setTimeout(resolve, ms));

}

async function SMA\_EMA\_Algo(smaSeries, emaSeries, TimeStamp){

for(let i = 0; i < emaSeries.length-1; i++){

// For Bullish

if (smaSeries[i] > emaSeries[i+10] & emaSeries[i+1+10] > smaSeries[i+1]){

sendMessage(`Bullish Market.

From : `+ String(smaSeries[i+1])+ '\nTime : ' + String(TimeStamp[i+1+10+4])

)

console.log(`Bullish Market.

From : `+ String(smaSeries[i+1])+ '\nTime : ' + String(TimeStamp[i+1+10+4]));

console.log();

await sleep(1000);

}

// For Bearish

if(emaSeries[i+10] > smaSeries[i] & smaSeries[i+1] > emaSeries[i+1+10]){

sendMessage(`Bearish Market.

From : `+ String(emaSeries[i+1+10]) +'\nTime : '+ String(TimeStamp[i+1+10+4])

)

console.log(`Bearish Market.

From : `+ String(emaSeries[i+1+10]) +'\nTime : '+ String(TimeStamp[i+1+10+4]));

console.log();

await sleep(1000);

}

}

}

# PUBLICATION AND PLAGIARISM REPORT