MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR

(A Govt. Aided UGC Autonomous & NAAC A++ Accredited Institute Affiliated to RGPV Bhopal)

Project Report

on

Stock Trading Simulator

Submitted By:
Aditi Shrivastava(0901AI211003)
Akshat Chandravanshi(0901AI211006)

Faculty Mentor:
Dr. Neelam Arya
Dr.Sunil Kumar Shukla

CENTER FOR ARTIFICIAL INTELLIGENCE

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE GWALIOR - 474005 (MP) est. 1957

July-Dec 2023

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

CERTIFICATE

This is certified that **Aditi Shrivastava (0901AI211003)** and **Akshat Chandravanshi (0901AI211006)** has submitted the project report titled "**Stock Trading Simulator**" under the mentorship of **Dr. Neelam Arya**, in partial fulfilment of the requirement for the award of degree of Bachelor of Technology in Computer Science and Engineering from Madhav Institute of Technology and Science, Gwalior.

Neelam
23/11/23

Prof. Neelam Arya
Faculty Mentor
Assistant Professor
Centre for Artificial Intelligence

[illegible] 23/11/23

Dr. R. R. Singh
Head of
Centre for Artificial Intelligence

Sunil
23/11/2023

Dr. Sunil Kumar Shukla
Assistant Professor
Coordinator
Center For Artificial Intelligence

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

DECLARATION

I hereby declare that the work being presented in this project report, for the partial fulfilment of requirement for the award of the degree of **Bachelor of Technology in Artificial Intelligence and Robotics** at Madhav Institute of Technology & Science, Gwalior is an authenticated and original record of my work under the mentorship of **Prof. Neelam Arya, Assistant Professor**, Center for Artificial Intelligence.

I declare that I have not submitted the matter embodied in this report for the award of any degree or diploma anywhere else.

Aditi Shrivastava [illegible]
(0901AI211003)
Akshat Chandravanshi [illegible]
(0901AI211006)
3rd Year,

Artificial Intelligence and Robotics

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR
(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

ACKNOWLEDGEMENT

The full semester project has proved to be pivotal to my career. I am thankful to my institute, **Madhav Institute of Technology and Science** to allow me to continue my disciplinary/interdisciplinary project as a curriculum requirement, under the provisions of the Flexible Curriculum Scheme (based on the AICTE Model Curriculum 2018), approved by the Academic Council of the institute. I extend my gratitude to the Director of the institute, **Dr. R. K. Pandit** and Dean Academics, **Dr. Manjaree Pandit** for this.

I would sincerely like to thank my department, **Centre for Artificial Intelligence,** for allowing me to explore this project. I humbly thank **Dr. R. R. Singh**, Coordinator, Centre for Artificial Intelligence, for his continued support during the course of this engagement, which eased the process and formalities involved.

I am sincerely thankful to my faculty mentors. I am grateful to the guidance of **Dr. Neelam Arya, Assistant Professor, Center for Artificial Intelligence**, for his continued support and guidance throughout the project. I am also very thankful to the faculty and staff of the department.

Aditi Shrivastava [illegible]
(0901AI211003)
Akshat Chandravanshi [illegible]
(0901AI211006)
3rd Year,
Artificial Intelligence and Robotic

TABLE OF CONTENTS

ABSTRACT

This Python program constitutes a basic trading simulator designed to implement and assess a moving average crossover strategy. Leveraging the pandas, numpy, and matplotlib libraries, the simulator encompasses critical aspects of algorithmic trading, including signal generation, backtesting, and performance visualization. The moving average crossover strategy, employed in this simulation, generates buy and sell signals based on the intersection of short-term and long-term moving averages. The backtesting module calculates portfolio values over time, incorporating initial capital, cash, and asset positions. The simulation output is visualized through matplotlib, providing insights into historical prices, buy/sell signals, and the evolving portfolio value. While this example serves as an introduction to trading simulation, it is essential to acknowledge the complexity of real-world trading, considering factors such as transaction costs, slippage, and risk management when developing and deploying trading algorithms. The provided simulator serves as a foundational framework for understanding and experimenting with algorithmic trading strategies.

This project focuses on the implementation of a trading strategy based on the Moving Average Convergence Divergence (MACD) indicator, using historical stock price data. The goal is to identify potential buy and sell signals and evaluate the profitability of the strategy. The Python programming language, along with the yfinance and matplotlib libraries, was utilized for data retrieval, analysis, and visualization.

The methodology involves calculating the MACD and signal line for a given stock, such as TSLA (Tesla), and determining buy and sell signals based on the MACD crossing above or below the signal line. The resulting signals were visualized on a plot, highlighting buy signals with green upward-pointing triangles and sell signals with red downward-pointing triangles.

To assess the profitability of the strategy, the script calculates the relative profits of each trade by comparing the open prices at the buy and sell points. The average relative profit per trade is computed, providing insight into the performance of the strategy during the given time period.

The implementation demonstrates the potential of using technical indicators, such as the MACD, for algorithmic trading. However, it is essential to note that historical performance does not guarantee future success, and the strategy should be thoroughly backtested and combined with risk management techniques before considering deployment in live trading environments.

The modular structure of the code allows for easy parameterization and adaptability to different stocks or time periods. Further refinements and optimizations can be explored, including backtesting on additional datasets and incorporating risk management strategies to enhance the robustness of the trading algorithm.

This project serves as a foundation for individuals interested in algorithmic trading, providing a practical example of strategy implementation and a starting point for more sophisticated and risk-aware trading systems.

सार

यह पायथन प्रोग्राम एक बुनियादी ट्रेडिंग सिम्युलेटर का गठन करता है जिसे लागू करने के लिए डिज़ाइनकिया गया है चलती औसत क्रॉसओवर रणनीति का आकलन करें। पांडा, नम्पी और मैटप्लोटलिब का लाभ उठाना पुस्तकालयों, सिम्युलेटर में सिग्नल सहित एल्गोरिथम ट्रेडिंग के महत्वपूर्ण पहलू शामिल हैं पीढ़ी, बैकटेस्टिंग और प्रदर्शन विजुअलाइज़ेशन। चलती औसत क्रॉसओवर इस सिमुलेशन में नियोजित रणनीति, प्रतिच्छेदन के आधार पर खरीद और बिक्री के संकेत उत्पन्न करती है अल्पकालिक और दीर्घकालिक चलती औसत का। बैकटेस्टिंग मॉड्यूल पोर्टफोलियो की गणना करता है

समय के साथ मूल्य, प्रारंभिक पूंजी, नकदी और संपत्ति की स्थिति को शामिल करते हुए। सिमुलेशन आउटपुट matplotlib के माध्यम से कल्पना की जाती है, जो ऐतिहासिक कीमतों, खरीदने/बेचने के संकेतों और में अंतर्दृष्टि प्रदान करता है विकसित हो रहा पोर्टफोलियो मूल्य। जबकि यह उदाहरण ट्रेडिंग के परिचय के रूप में कार्य करता है अनुकरण, विचार करते हुए, वास्तविक दुनिया के व्यापार की जटिलता को स्वीकार करना आवश्यक है विकास करते समय लेनदेन लागत, फिसलन और जोखिम प्रबंधन जैसे कारक ट्रेडिंग एल्गोरिदम तैनात करना। प्रदान किया गया सिम्युलेटर एक मूलभूत ढांचे के रूप में कार्य करता है एल्गोरिथम ट्रेडिंग रणनीतियों को समझना और प्रयोग करना।

Chapter 1 : INTRODUCTION

1.1. Introduction

This project is a short for a stock trading robot or algorithmic trading bot, is an automated system designed to execute trades in financial markets, specifically in the stock market, based on pre-defined criteria, algorithms, and parameters. These bots use computer programs, algorithms, and mathematical models to analyze market data, identify trading opportunities, and execute buy or sell orders without human intervention.
At the most basic level, an algorithmic trading robot is a computer code that has the ability to generate and execute buy and sell signals in financial markets. The main components of such a robot include entry rules that signal when to buy or sell, exit rules indicating when to close the current position, and position sizing rules defining the quantities to buy or sell.

The development and functioning of a stock trading bot involve various components:

Algorithm Development: Trading bots rely on algorithms created by traders, quantitative analysts, or developers. These algorithms are based on technical indicators, fundamental analysis, machine learning, or a combination of these methods.

Data Analysis: Bots analyze vast amounts of historical and real-time market data to identify trends, patterns, correlations, and other factors that can influence trading decisions.

Decision Making: Based on the predefined rules and strategies, the bot makes buy or sell decisions. These decisions can be based on signals like moving averages, volume changes, price fluctuations, news sentiment, etc.

1.2. Libraries Used

- **Pandas**: A fundamental library for data manipulation and analysis, Pandas was crucial in handling the dataset's structure. It provided a robust framework for organizing, cleaning, and transforming the data, making it amenable for analysis.

- **Matplotlib**: These visualization libraries enabled the creation of insightful charts and graphs. Matplotlib, a versatile 2D plotting library, and Seaborn, based on Matplotlib, added an aesthetic layer to the visualizations, enhancing the interpretability of the findings.

- **Yfinance:** is a Python library that provides a simple and easy-to-use way to access historical market data, live market data, and information about stocks, ETFs (Exchange-Traded Funds), mutual funds, and more from Yahoo Finance. It enables users to retrieve historical market data for analysis, which can be useful for building trading strategies, conducting research, or performing quantitative analysis.

```
In [32]: pip install yfinance
```

```
Defaulting to user installation because normal site-packages is not writ
eable
Requirement already satisfied: yfinance in c:\users\akshat chandravanshi
\appdata\roaming\python\python39\site-packages (0.2.32)
Requirement already satisfied: pandas>=1.3.0 in c:\programdata\anaconda3
\lib\site-packages (from yfinance) (1.4.2)
Requirement already satisfied: html5lib>=1.1 in c:\programdata\anaconda3
\lib\site-packages (from yfinance) (1.1)
Requirement already satisfied: multitasking>=0.0.7 in c:\users\akshat ch
andravanshi\appdata\roaming\python\python39\site-packages (from yfinanc
e) (0.0.11)
Requirement already satisfied: appdirs>=1.4.4 in c:\programdata\anaconda
3\lib\site-packages (from yfinance) (1.4.4)
Requirement already satisfied: pytz>=2022.5 in c:\programdata\anaconda3
\lib\site-packages (from yfinance) (2022.7)
Requirement already satisfied: lxml>=4.9.1 in c:\users\akshat chandravan
shi\appdata\roaming\python\python39\site-packages (from yfinance) (4.9.
3)
Requirement already satisfied: numpy>=1.16.5 in c:\programdata\anaconda3
```

```
In [4]: pip install matplotlib
```

```
Defaulting to user installation because normal site-packages is not writea
ble
Requirement already satisfied: matplotlib in c:\programdata\anaconda3\lib
\site-packages (3.7.1)
Requirement already satisfied: packaging>=20.0 in c:\programdata\anaconda3
\lib\site-packages (from matplotlib) (23.0)
Requirement already satisfied: cycler>=0.10 in c:\programdata\anaconda3\li
b\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: numpy>=1.20 in c:\programdata\anaconda3\lib
\site-packages (from matplotlib) (1.23.5)
Requirement already satisfied: python-dateutil>=2.7 in c:\programdata\anac
onda3\lib\site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\programdata\anaconda
3\lib\site-packages (from matplotlib) (1.0.5)
Requirement already satisfied: pyparsing>=2.3.1 in c:\programdata\anaconda
3\lib\site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: pillow>=6.2.0 in c:\programdata\anaconda3\l
ib\site-packages (from matplotlib) (9.4.0)
Requirement already satisfied: importlib-resources>=3.2.0 in c:\programdat
a\anaconda3\lib\site-packages (from matplotlib) (5.2.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\programdata\anacond
a3\lib\site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: fonttools>=4.22.0 in c:\programdata\anacond
a3\lib\site-packages (from matplotlib) (4.25.0)
Requirement already satisfied: zipp>=3.1.0 in c:\programdata\anaconda3\lib
\site-packages (from importlib-resources>=3.2.0->matplotlib) (3.11.0)
Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\si
te-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

```
In [2]: import yfinance as yf
        import matplotlib.pyplot as plt
        import pandas as pd
```

1.3. Purpose and Scope

The purpose of creating a stock trading bot, also known as an algorithmic trading bot, is multi-faceted and revolves around leveraging automation and technology to optimize trading activities in financial markets. Some of the key purposes include:

Automated Execution: Stock trading bots are designed to execute trades automatically based on pre-defined rules, algorithms, or strategies. They operate without emotional bias, eliminating human errors that can arise from emotional decision-making or fatigue.

Speed and Efficiency: Bots can process and analyze market data at speeds far beyond human capability. They can swiftly identify trading opportunities, react to market changes, and execute trades in milliseconds, taking advantage of fleeting opportunities that might be missed by human traders.

Backtesting and Optimization: Before deploying a trading strategy in live markets, bots can be backtested using historical data to evaluate their performance. This process allows traders to refine and optimize strategies based on past market conditions.

Diversification and Consistency: Trading bots can manage multiple assets, portfolios, or strategies simultaneously, ensuring consistent execution and diversification across various markets or securities.

Strategy Implementation: They enable the implementation of complex trading strategies, such as high-frequency trading, statistical arbitrage, trend following, or machine learning-based strategies, which might be difficult to execute manually.

Chapter 2 : Data Acquisition and Preprocessing

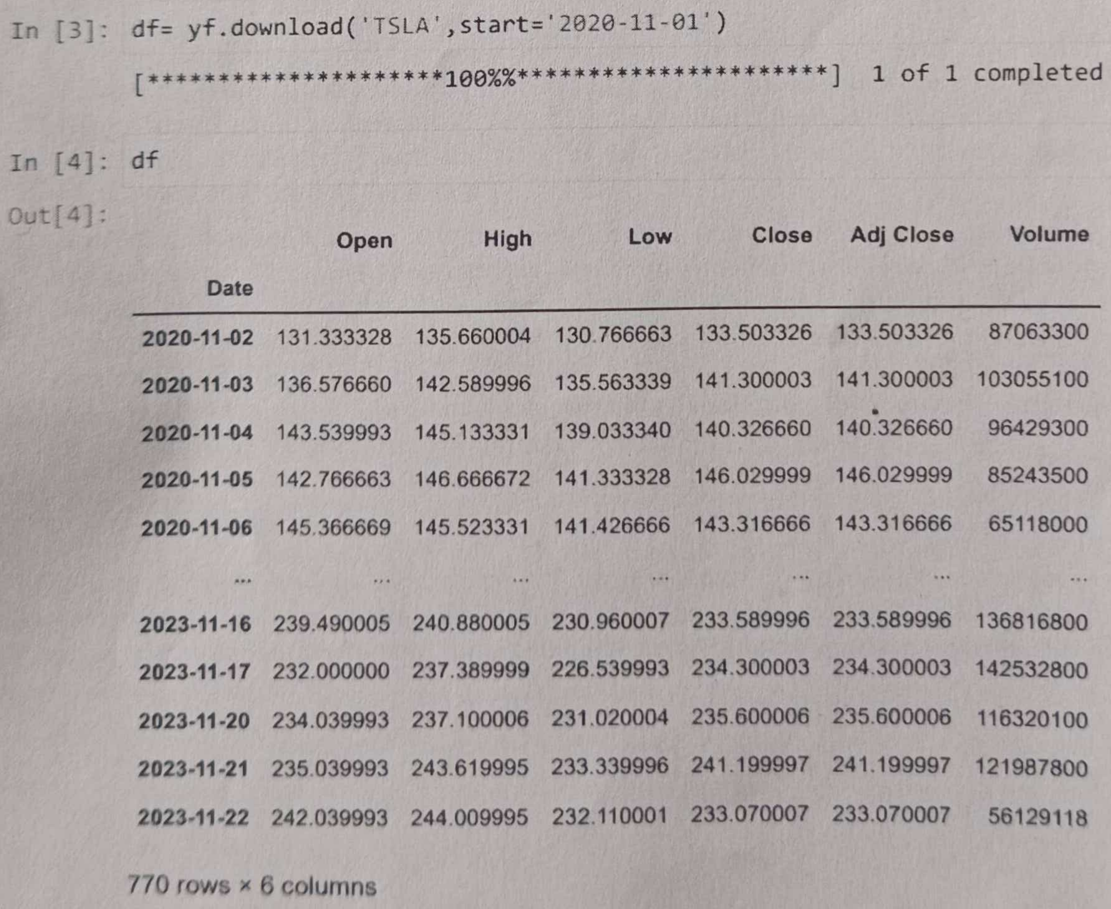
2.1. Data collection and dataset description

Data acquisition and preprocessing are crucial steps in building a robust and reliable trading bot. Proper handling and preparation of data significantly impact the effectiveness and performance of the trading strategies implemented by the bot.

a. **Market Data Sources:** Identify and select appropriate data sources. Common sources include financial data providers like Yahoo Finance, Alpha Vantage, Quandl, or direct APIs from stock exchanges or brokerage firms.

b. **Frequency and Granularity:** Decide on the frequency and granularity of data required (e.g., daily, hourly, minute-by-minute) based on the trading strategy's needs.

c. **API Integration:** Utilize APIs provided by data sources to programmatically fetch and retrieve the required data. Libraries like yfinance, alpha_vantage, or direct brokerage APIs can be used for data retrieval.



```
In [3]: df= yf.download('TSLA',start='2020-11-01')
        [*********************100%%**********************]  1 of 1 completed

In [4]: df

Out[4]:
```

	Open	High	Low	Close	Adj Close	Volume
Date						
2020-11-02	131.333328	135.660004	130.766663	133.503326	133.503326	87063300
2020-11-03	136.576660	142.589996	135.563339	141.300003	141.300003	103055100
2020-11-04	143.539993	145.133331	139.033340	140.326660	140.326660	96429300
2020-11-05	142.766663	146.666672	141.333328	146.029999	146.029999	85243500
2020-11-06	145.366669	145.523331	141.426666	143.316666	143.316666	65118000
...	...	...	...	...	...	...
2023-11-16	239.490005	240.880005	230.960007	233.589996	233.589996	136816800
2023-11-17	232.000000	237.389999	226.539993	234.300003	234.300003	142532800
2023-11-20	234.039993	237.100006	231.020004	235.600006	235.600006	116320100
2023-11-21	235.039993	243.619995	233.339996	241.199997	241.199997	121987800
2023-11-22	242.039993	244.009995	232.110001	233.070007	233.070007	56129118

770 rows × 6 columns

Data Preprocessing:

a. **Cleaning and Handling Missing Data:** Check for missing values, outliers, or inaccuracies in the acquired data and handle them appropriately (e.g., interpolation, deletion, or imputation).

b. **Normalization and Scaling:** Normalize or scale numerical features to ensure they are on the same scale, preventing any feature from dominating others during analysis.

c. **Feature Engineering**: Create additional features that might enhance the predictive power of the model. For instance, generating technical indicators like moving averages, RSI, MACD, or creating derived features from existing data.

d. **Handling Time Series Data:** Time series data might require special handling, such as resampling, rolling windows, or lagging/leading indicators to create predictive features.

2.2 Data cleaning and handling missing values.

Data cleaning and the handling of missing values are crucial steps in preparing the dataset for analysis. In this project, several techniques were applied to ensure the dataset's integrity and consistency:

1. **Removing Duplicates**: Duplicate entries can skew the analysis and lead to inaccurate results. Using the data.drop_duplicates(["res_id"], keep='first', inplace=True) command, duplicate rows were eliminated based on the 'res_id' column, ensuring that each restaurant was represented only once in the dataset.
2. **Handling Missing Values:** Missing values in the 'address,' 'timings,' and 'opentable_support' columns were addressed to prevent data gaps from affecting the analysis. The missing values in the 'address' and 'timings' columns were imputed with placeholder values 'Unknown' and 'Not available,' respectively, using data['address'].fillna("Unknown", inplace=True) and data['timings'].fillna("Not available", inplace=True). For the 'opentable_support' column, missing values were filled with the default value '0' to maintain data consistency (data['opentable_support'].fillna(0, inplace=True)).

These data cleaning and missing value handling techniques contribute to a more robust and complete dataset, enabling accurate analysis and modeling. They ensure that the dataset is free from inconsistencies and gaps, allowing for meaningful insights to be extracted during the subsequent stages of the project.

Chapter 3: Exploratory Data Analysis (EDA)

3.1. Data Retrieval and Overview:

Data Source: The historical stock price data for TSLA (Tesla) was retrieved using the yfinance library, covering the period from November 2020 to the present.

DataFrame Structure: The dataset consists of columns such as Date, Open, High, Low, Close, Volume, EMA12, EMA26, MACD, and Signal.

3.2. MACD Calculation:

Exponential Moving Averages (EMAs): Two EMAs with periods of 12 and 26 were computed based on the closing prices.

MACD Indicator:The MACD line was derived by subtracting the 26-period EMA from the 12-period EMA.

3.3. Signal Generation:

Buy and Sell Signals: Buy signals were generated when the MACD crossed above the signal line, while sell signals occurred when the MACD crossed below the signal line.

Signal Visualization: The buy signals were marked with green upward-pointing triangles, and sell signals were marked with red downward-pointing triangles on the closing price plot.

3.4. Profitability Analysis:

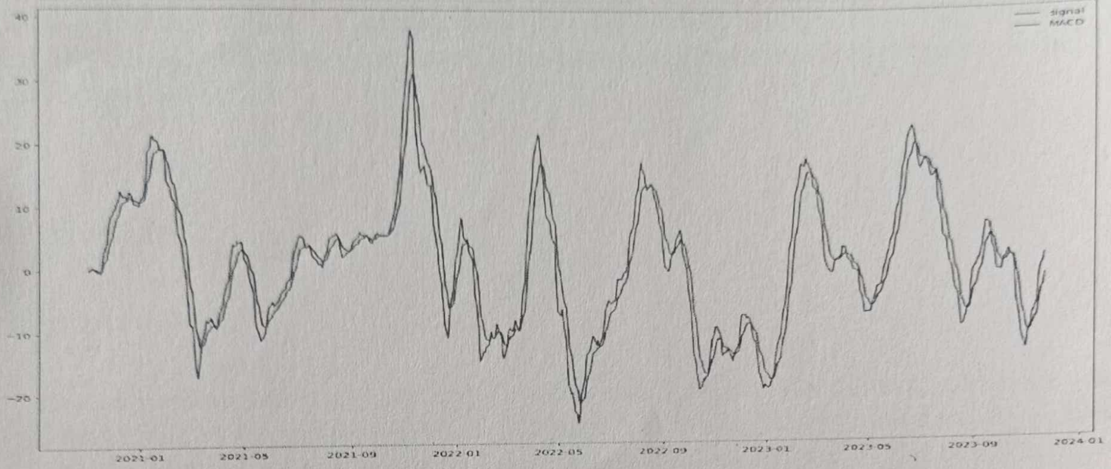
Trade Identification:Trades were identified by tracking the buy and sell signals.

3.5. Visualizations:

MACD and Signal Line Plot: The MACD and signal line were plotted over time to visualize their movements and crossovers.

Buy and Sell Signal Plot: Buy and sell signals were visually represented on the closing price plot, aiding in the interpretation of the strategy's performance.

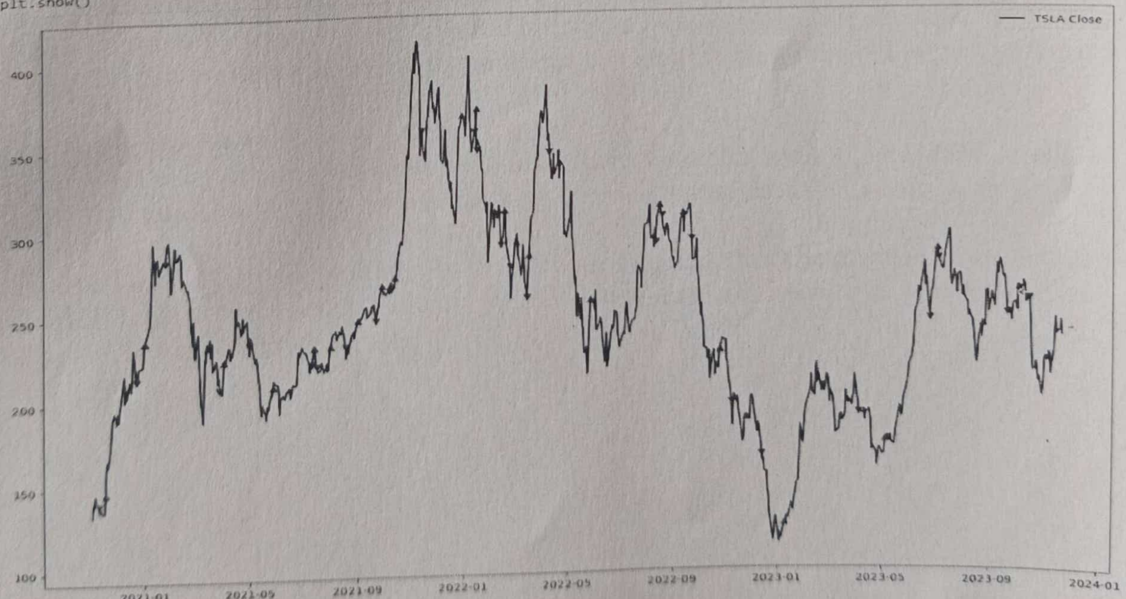
```
              dtype='datetime64[ns]', name='Date', freq=None)

In [19]: plt.figure(figsize=(16,10))

         plt.plot(df.signal,label='signal',color='red')
         plt.plot(df.MACD,label='MACD',color = 'green')
         plt.legend()
         plt.show()
```



3.6. Conclusion:

- The EDA provides a comprehensive overview of the MACD-based trading strategy, from data retrieval and indicator calculation to signal generation and profitability analysis.
- This analysis serves as a foundation for further exploration, optimization, and backtesting, emphasizing the importance of thorough evaluation before considering deployment in live trading scenarios.

```
In [18]: plt.figure(figsize=(16,10))
         plt.scatter(df.iloc[Buy].index,df.iloc[Buy].Close,marker="^",color = 'green')
         plt.scatter(df.iloc[Sell].index,df.iloc[Sell].Close,marker="v",color = 'red')
         plt.plot(df.Close,label='TSLA Close',color = 'k')
         plt.legend()
         plt.show()
```



Backtesting and Optimization:

Perform backtesting on historical data to simulate how the model would have performed in the past. Optimize trading strategies by adjusting parameters, thresholds, or rules based on backtesting results to maximize returns or minimize risk.

Chapter 4 : Results

4.1 User driven prediction

The results of a stock trading bot can vary significantly based on various factors, including the sophistication of the trading strategy, the quality of data used for training, the market conditions, risk management techniques employed, and the overall design and implementation of the bot. Here are some potential outcomes and results that can arise from a stock trading bot:

Profitability:

Successful trading bots can generate profits by making accurate predictions, executing trades effectively, and capitalizing on market opportunities. However, profitability isn't guaranteed, and bots might incur losses, especially during volatile or unpredictable market conditions.

Risk Management:
A well-designed trading bot focuses not only on maximizing profits but also on managing risks. Effective risk management strategies help in minimizing losses and controlling exposure to market risks.

Backtesting vs. Live Trading Results:
Backtesting results might differ from live trading due to factors like slippage, latency, order execution issues, and data discrepancies. Real-time market conditions can significantly impact bot performance.

Overfitting and Generalization:
Overfitting occurs when a set of rules performs well on historical data but fails to generalize to unseen data. Ensuring the bot's ability to perform in live markets, beyond historical data, is crucial.

It's important to note that the stock market involves inherent risks, and even the most advanced trading bots can experience losses. No trading strategy or bot can guarantee consistent profits, and past performance is not indicative of future results.

CONCLUSION

This project could be concluded as –

It consists of certain codes and information about how this model is designed with tha inclusion of various libraries and data sets that we used in it.

This project constitutes a basic trading simulator designed to implement and assess a moving average crossover strategy. Leveraging the pandas, numpy, and matplotlib libraries, the simulator encompasses critical aspects of algorithmic trading, including signal generation, backtesting, and performance visualization. The moving average crossover strategy, employed in this simulation, generates buy and sell signals based on the intersection of short-term and long-term moving averages.

The backtesting module calculates portfolio values over time, incorporating initial capital, cash, and asset positions. The simulation output is visualized through matplotlib, providing insights into historical prices, buy/sell signals, and the evolving portfolio value. While this example serves as an introduction to trading simulation, it is essential to acknowledge the complexity of real-world trading, considering factors such as transaction costs, slippage, and risk management when developing and deploying trading algorithms.

This project focuses on the implementation of a trading bot with strategy based on the Moving Average Convergence Divergence (MACD) indicator, using historical stock price data. The goal is to identify potential buy and sell signals and evaluate the profitability of the strategy. The Python programming language, along with the yfinance and matplotlib libraries, was utilized for data retrieval, analysis, and visualization.

The methodology involves calculating the MACD and signal line for a given stock, such as TSLA (Tesla), and determining buy and sell signals based on the MACD crossing above or below the signal line. The resulting signals were visualized on a plot, highlighting buy signals with green upward-pointing triangles and sell signals with red downward-pointing triangles.

REFERENCES

Here are some references that can be useful for learning more about quiz generators:

- Got some of the reference from my friends.
- Some of the content is taken from Stock Trading Bot: Coding Your Own Trading Algo (investopedia.com)