ACCURACY OF STOCK PRICE PREDICTION – USING THE SENTIMENT ANALYSIS

A PROJECT REPORT

Submitted by,

Ms. Chinnaobaiahgari Likhitha - 20201CSD0103
Mr. Sallapuram Yaswanth Reddy -20201CSD0104
Mr. Korrapati Pavan Kalyan Gowd -20201CSD0094
Ms. Veerapu Reddy Suvarna Chandrika Reddy - 20201CSD0107
Mr. Tholla Ananth Kumar -20201CSD0070

Under the guidance of,

Dr. Chandrasekar V

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING [DATA SCIENCE]
At

PRESIDENCY UNIVERSITY

BENGALURU

JANUARY 2024

PRESIDENCY UNIVERSITY

SCHOOL OF COMPUTER SCIENCE ENGINEERING

CERTIFICATE

This is to certify that the Project report **"ACCURACY OF STOCK PRICE PREDICTION – USING THE SENTIMENT ANALYSIS"** being submitted by "Chinnaobaiahgari Likhitha", " Sallapuram Yaswanth Reddy", " Korrapati Pavan Kalyan Gowd", " Veerapu Reddy Suvarna Chandrika Reddy", " Tholla Ananth Kumar "bearing roll number(s) "20201CSD0103", "20201CSD0104"," 20201CSD0094", "20201CSD0107", " 20201CSD0070" in partial fulfilment of requirement for the award of degree of Bachelor of Technology in Computer Science and Engineering [DATA SCIENCE] is a bonafide work carried out under my supervision.

Dr.Chandrasekar V
Professor
School of CSE&IS
Presidency University

Dr. A. Jayachandran
Professor & HoD
School of CSE&IS
Presidency University

Dr. C. KALAIARASAN
Associate Dean
School of CSE&IS
Presidency University

Dr. L. SHAKKEERA
Associate Dean
School of CSE&IS
Presidency University

Dr.MD SAMEERUDDIN KHAN
Dean
School of CSE&IS
Presidency University

PRESIDENCY UNIVERSITY

SCHOOL OF COMPUTER SCIENCE ENGINEERING

DECLARATION

We hereby declare that the work, which is being presented in the project report entitled **ACCURACY OF STOCK PRICE PREDICTION – USING THE SENTIMENT ANALYSIS** in partial fulfilment for the award of Degree of **Bachelor of Technology** in **Computer Science and Engineering [DATA SCIENCE]**, is a record of our own investigations carried under the guidance of SUPERVISOR **Dr. CHANDRASEKAR V, PROFESSOR, School of Computer Science Engineering & Information Science, Presidency University, Bengaluru.**

We have not submitted the matter presented in this report anywhere for the award of any other Degree.

NAME	ID	SIGNATURE
Chinnaobaiahgari Likhitha	20201CSD0103	
Sallapuram Yaswanth Reddy	20201CSD0104	
Korrapati Pavan Kalyan Gowd	20201CSD0094	
Veerapu Reddy Suvarna Chandrika Reddy	20201CSD0107	V. Suvarna
Tholla Ananth Kumar	20201CSD0070	T. Ananth Kumar

ABSTRACT

Forecasting stock prices is a complex task given the dynamic and non-stationary nature of financial markets. Accurate predictions are challenging due to the inherent volatility and non-linear patterns in stock market behavior. This involves anticipating the future value of a company's stock or other financial instruments traded on exchanges, aiming to maximize investor gains. The recent success of applying Artificial Intelligence in finance has led to increased reliance on stochastic models for market predictions. Stock market prediction, a longstanding research focus, incorporates various machine learning techniques and diverse datasets. While many studies utilize historical stock statistics and relevant real-time data (e.g., oil and gold prices), few explore the integration of financial news in predicting stock price trends. To address this gap, A system is proposed that leverages machine learning techniques and the LSTM (Long Short-Term Memory) algorithm for stock price prediction based on historical stock data.

ACKNOWLEDGEMENT

First of all, we indebted to the **GOD ALMIGHTY** for giving an opportunity to excel in our efforts to complete this project on time.

We express our sincere thanks to our respected dean **Dr. Md. Sameeruddin Khan**, Dean, School of Computer Science Engineering & Information Science, Presidency University for getting us permission to undergo the project.

We record our heartfelt gratitude to our beloved Associate Deans **Dr. Kalaiarasan C and Dr. Shakkeera L,** School of Computer Science Engineering & Information Science, Presidency University and **Dr. A. Jayachandran**. Head of the Department, School of Computer Science Engineering, Presidency University for rendering timely help for the successful completion of this project.

We are greatly indebted to our guide **Dr. ChandraSekar V, Professor**, School of Computer Science Engineering & Information Science, Presidency University for his inspirational guidance, and valuable suggestions and for providing us a chance to express our technical capabilities in every respect for the completion of the project work.

We would like to convey our gratitude and heartfelt thanks to the University Project-II Coordinators **Dr. Sanjeev P Kaulgud, Dr. Mrutyunjaya MS** and also the department Project Coordinators **Dr. Manjula H M, Mr. Yamanappa**.

We thank our family and friends for the strong support and inspiration they have provided us in bringing out this project.

C. Likhitha
S. Yaswanth Reddy
K. Pavan Kalyan Gowd
V. Suvarna Chandrika Reddy
T. Ananth Kumar

LIST OF TABLES

LIST OF FIGURES

TABLE OF CONTENTS

CHAPTER-1
INTRODUCTION

1.1. General Information:

Predicting stock prices in the stock market is extremely difficult because of its dynamic and unexpected nature. A few of the elements that affect stock prices are the state of the global economy, political situations, and a company's financial reports and performance. It becomes essential to look at past trends in order to maximize gains and minimize losses when reacting to stock market fluctuations. There are two main methods that have historically been suggested to forecast the stock price of a firm. To predict future stock prices, a technical analysis method looks at previous stock prices that include closing, opening, volume traded, and neighboring close values.

The second method, qualitative analysis, takes into account outside variables including the company's profile, the state of the market, political and economic forces, and textual data from blogs by economic analysts, social media, and financial news items. Large companies list their equities on the stock exchange in an effort to draw in investors and boost liquidity. Stockbrokers and traders who purchase and sell shares use the stock market as a central marketplace. Despite its allure, investing in the stock market carries risks due to the rapid fluctuations in stock values. Because of this volatility, which emphasizes how hard it is to predict stock values, several scholars have decided to carry out more research.

Stock market prediction algorithms have become essential tools for traders due to factors including the price of gold and oil, notable events, and business news. While academics are using more and more financial news to improve the accuracy of stock market prediction algorithms, most of the parameters considered It is well known that forecasting changes in the price of stocks is a challenging task. The Efficient Market Hypothesis states that prices always represent all available information and that the only factor influencing a security's price is new information. The haphazard introduction of fresh information seems to have contributed to the market values appearing to be set randomly.

A solution to these issues is provided by a system that forecasts stock values based on historical market data using the Long Short-Term Memory (LSTM) algorithm and machine learning techniques. This approach aims to increase the accuracy of stock market forecasts and provide investors with more reliable information for decision-making by tackling non-stationary, noisy, and chaotic data. The proposed system is a newcomer to the field of stock market forecasting.

1.2. Problem Statement:

Forecasting stock market trends presents a significant challenge due to the dynamic, noisy, and unpredictable nature of the data. Consequently, investors face difficulties in making profitable investment decisions. Numerous methods have been developed to predict stock market trends within the current body of literature. However, none have proven consistently accurate in their results.

1.3. Proposed System:

A proposed system employs machine learning techniques and utilizes the Long Short-Term Memory (LSTM) algorithm to predict stock prices by analyzing historical stock data.

CHAPTER-2
LITERATURE SURVEY

ETLP. K. Bharne and S. S. Prabhune: This paper introduces the fundamental principles of swarm intelligence and its application in optimizing daily stock market prices. It outlines various swarm intelligence algorithms such as ACO, PSO, BAT, and Firefly. The comparative analysis of recent swarm intelligence-based approaches concludes that SI-ANN yields more optimized results than SI with machine learning algorithms. The paper provides insights into recent SI trends and future directions.

ETL Mehar Vijha, Deeksha Chandolab, Vinay Anand Tikkiwalb, Arun Kumarc: The study addresses the limitations of historical datasets with minimal features, proposing the creation of new variables for improved accuracy in predicting stock prices. The paper employs Artificial Neural Networks (ANN) for predicting the next day's closing stock price and conducts a comparative analysis with Random Forest (RF). Results based on RMSE, MAPE, and MBE values indicate that ANN outperforms RF in stock price prediction.

ETLHegazy, Osman & Soliman, Omar S: This paper proposes a machine learning model integrating the particle swarm optimization (PSO) algorithm and LS-SVM for stock price prediction using financial technical indicators. The study includes indicators like relative strength index, money flow index, exponential moving average, stochastic oscillator, and moving average convergence/divergence. LS-SVM-PSO achieves the lowest error value, surpassing single LS-SVM, while the ANN-BP algorithm exhibits lower performance.

ETLB. B. P. Maurya, A. Ray, A. Upadhyay, B. Gour, and A. U. Khan: The developed machine learning model enhances stock price anticipation precision by incorporating latest technical indicators - MACD, P/R, and Moving Averages. Moving Averages aid in identifying support and resistance, contributing to efficient stock prediction. The model utilizes web scraping for real-time data directly from the stock market, making it applicable to real-world and real-time stock problems.

ETLI. Bhattacharjee and P. Bhattacharja: The paper conducts a comparative study between statistical approaches and machine learning approaches in terms of prediction performances

and accuracy. Machine learning methods, especially MLP and LSTM, emerge as the most accurate for predicting stock prices, exhibiting the least MSE and MAPE values. The study emphasizes the effectiveness of MLP and LSTM for accurate stock price predictions.

ETLU. Pasupulety, A. Abdullah Anees, S. Anmol, and B. R. Mohan: The proposed systematic approach for Stock Prediction utilizes Ensemble Learning and Sentiment Analysis. SVM and Extremely Randomized trees are employed, and a Stacked Regressor improves predicted prices based on individual model accuracy. Sentiment analysis of public opinion from tweets is integrated as an additional feature in the stock dataset, contributing to the overall prediction accuracy.

CHAPTER-3

RESEARCH GAPS OF EXISTING METHODS

1. Limited Generalization Across Markets:

Many current models are crafted and trained on data from specific markets, restricting their applicability to other markets. Research endeavors could concentrate on formulating more resilient models capable of effective generalization across diverse financial markets.

2. Time-Based Flow and News Effect:

The predominant focus of existing models is often on sentiment at a singular moment, neglecting its dynamic nature and its impact on stock prices over time. Potential research avenues include exploring temporal dynamics and discerning the relative influence of short and long-term shifts in sentiment on stock prices.

3. Particular Sentiment Analysis for Users:

Current sentiment analysis models typically treat user opinions uniformly, regardless of their historical accuracy. Future research could delve into personalized sentiment analysis, assigning varying weights to users based on their past performance, thereby offering a more nuanced understanding of individual user impact on the market.

4. Interpretability and Explainability of Sentiment:

Many machine learning models lack transparency and interpretability, posing challenges in gaining trust and adoption within the financial industry. Enhancing transparency in the decision-making process and comprehending how sentiment analysis models generate specific predictions could foster greater trust and adoption.

By addressing these research gaps, sentiment analysis models for stock price prediction have the potential to strengthen and enhance accuracy, ultimately benefiting financial institutions and investors.

CHAPTER 4
PROPOSED METHODOLOGY

The primary objective of this project is to forecast the stock price for a specific date using a machine learning model. The first step is to get a reliable stock price dataset. After that, loading and preprocessing techniques, including normalization, scaling, and handling missing values, are applied. The preparation stage is finished, and the data is suitably divided into training and testing sets. Next, a model known as an LSTM (long short-term memory) is constructed. This type of model is highly regarded for its ability to depict temporal dependencies.

The testing set is used for evaluation, and the training set is utilized for the process. Metrics like mean squared error and accuracy are used to gauge the model's performance. The finished model is then saved as a h5 file, which is a hierarchical data format perfect for containing complex structures.

A framework named streamlit is used to develop a front end, which enhances user interaction. With this user interface, users can input their favorite date for stock price prediction. This input is seamlessly transferred to the model by the front end, which estimates the stock price for the specified date. The project concludes by giving the user access to the results via the front end, showcasing its worth as a helpful tool for stock market enthusiasts and investors alike.

It is essential to keep in mind that numerous factors influence the market, rendering stock value prediction unfeasible. Therefore, rather than providing precise predictions, a model may provide insights based on historical trends. The financial markets are extremely dynamic, so staying ahead of the curve requires both excellent algorithms and a keen understanding of market dynamics.

CHAPTER-5
OBJECTIVES

Forecasting stock prices poses a formidable challenge, and the integration of machine learning techniques, particularly LSTM algorithms, introduces a sophisticated layer to this endeavor. LSTM's prowess in capturing long-term dependencies makes it particularly well-suited for handling time-series data such as stock prices.

To delve into the intricacies of the project, the initial step often involves gathering historical stock data, encompassing key features like opening/closing prices, volume, and potentially other pertinent financial indicators. This dataset serves as the foundation for training the machine learning model.

The preprocessing stage is paramount, involving tasks such as addressing missing data, normalization, and potentially incorporating feature engineering to augment the model's ability to discern underlying patterns. LSTM networks, falling under the category of recurrent neural networks (RNN), excel at retaining information over extended periods, rendering them proficient in capturing the inherent temporal dynamics within stock prices.

Upon completion of data preprocessing, the subsequent phase involves crafting the architecture of the LSTM model. Factors such as the number of layers, neurons per layer, and the selection of activation functions merit careful consideration. The training process entails feeding the model batches of historical data and adjusting weights based on prediction errors.

Validation and testing phases play a pivotal role in evaluating the model's performance. Metrics such as Mean Squared Error (MSE) or Root Mean Squared Error (RMSE) provide valuable insights into how effectively the model generalizes to new, unseen data.

It is imperative to acknowledge the inherent uncertainty in predicting stock prices due to the myriad factors influencing the market. Consequently, while a model may not offer foolproof predictions, it can provide valuable insights based on historical patterns. Navigating the dynamic landscape of financial markets necessitates a blend of robust algorithms and a

profound understanding of market dynamics to stay ahead.

CHAPTER-6

SYSTEM DESIGN & IMPLEMENTATION

6.1. System Architecture:

The first step is acquiring a dependable dataset containing stock prices and then proceeding to load it. Subsequently, the dataset undergoes preprocessing to purify and understand its features. This process encompasses tasks such as handling outliers, scaling, normalization, implementing feature engineering, and addressing missing values.

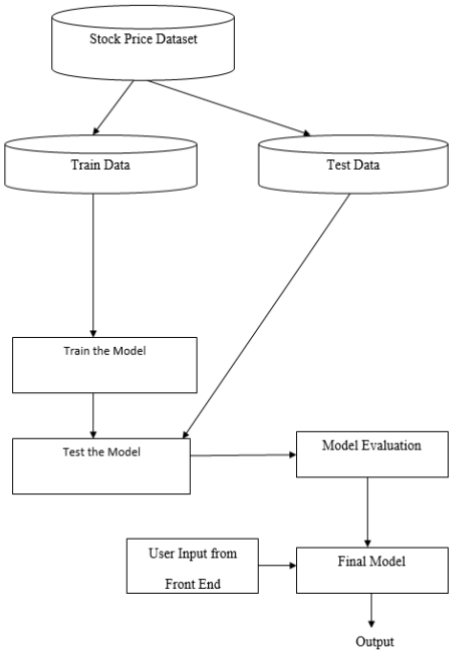


Figure 1.1

The data was partitioned into training and testing sets using a balanced ratio. Subsequently, the model was built utilizing Long Short-Term Memory (LSTM), a form of recurrent neural network architecture specifically crafted to capture temporal dependencies within the input data. The model's efficacy was assessed by evaluating its performance on the testing set after undergoing training on the training set, employing accuracy and mean squared error as performance metrics. Following this, the finalized model was applied to predict stock prices for new dates based on user input from the front end.

To facilitate user engagement with the model, a front end was developed using streamlit. The concluding step involves presenting the forecasted stock price to the user, delivering a valuable service for investors and enthusiasts in the stock market.

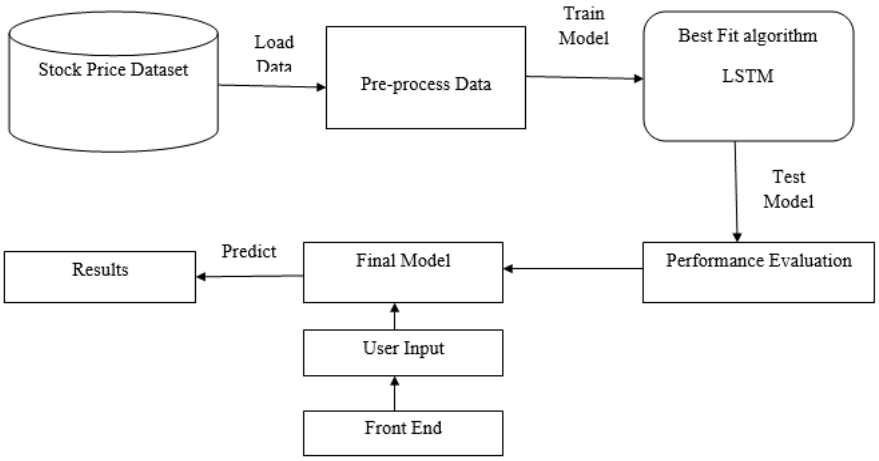


Figure 1.2

6.1.1. Use Case Diagram

Creating a machine learning model involves a systematic and intricate process, encompassing several essential stages. Initially, a diverse range of user data in varying formats like CSV, JSON, or Excel files is gathered. This raw data is then meticulously organized into a structured format conducive to thorough analysis. The second stage involves a detailed examination of the collected data, conducting exploratory analysis to uncover patterns and insights, and addressing potential data quality issues for the reliability and accuracy of the subsequent model.

Moving forward, the third stage is dedicated to preprocessing and model construction, a critical phase in the model development journey. Various techniques, including imputation, scaling, normalization, and encoding, are employed to transform raw data into a format suitable for the chosen machine learning algorithm. This stage also involves the pivotal task of selecting an appropriate machine learning algorithm aligned with the specific characteristics of the dataset. Once these preparatory steps are completed, the model is trained using the preprocessed data, refining its parameters and structure through methods like cross-validation or grid search.

The iterative nature of these stages ensures that the model is fine-tuned and optimized for performance. The constructed model is then poised for evaluation and testing in the fourth stage, utilizing metrics like mean squared error, mean absolute error and explained variance score R2_score. This evaluation involves scrutinizing the model's performance on both training and testing sets, guarding against potential issues like overfitting or underfitting. User feedback becomes integral in the fifth stage, where the model is reviewed based on its results, ensuring alignment with user expectations.

The culmination of this process is the sixth stage, where the finalized model is saved as a file or a database object, ready for deployment and utilization. In this stage, the model is practically applied to predict results based on new user input. Users provide fresh data, such as observations or queries, which undergo preprocessing before being fed into the model. The model then generates predicted outcomes, whether class labels or numerical values, showcased through a graphical user interface or a web page. This user-friendly interface enhances accessibility and usability, marking the successful completion of the machine learning model development journey.

In the fourth stage, the constructed model undergoes evaluation and testing using metrics like mean squared error, mean absolute error and explained variance score R2_score. This process involves comparing the model's performance on training and testing sets while meticulously examining potential overfitting or underfitting issues. The fifth stage revolves around finalizing the model, with user feedback playing a pivotal role in reviewing the results. The completed model is then saved as a file or database object for future utilization.

The sixth stage involves the practical application of the model to predict results and showcase them. Users input new data, such as observations or queries, which undergo preprocessing before being input into the finalized model. The model generates predicted results, whether class labels or numerical values, and presents these outcomes on a graphical user interface or web page. This user-friendly interface enhances the accessibility and usability of the machine learning model for both developers and end-users.

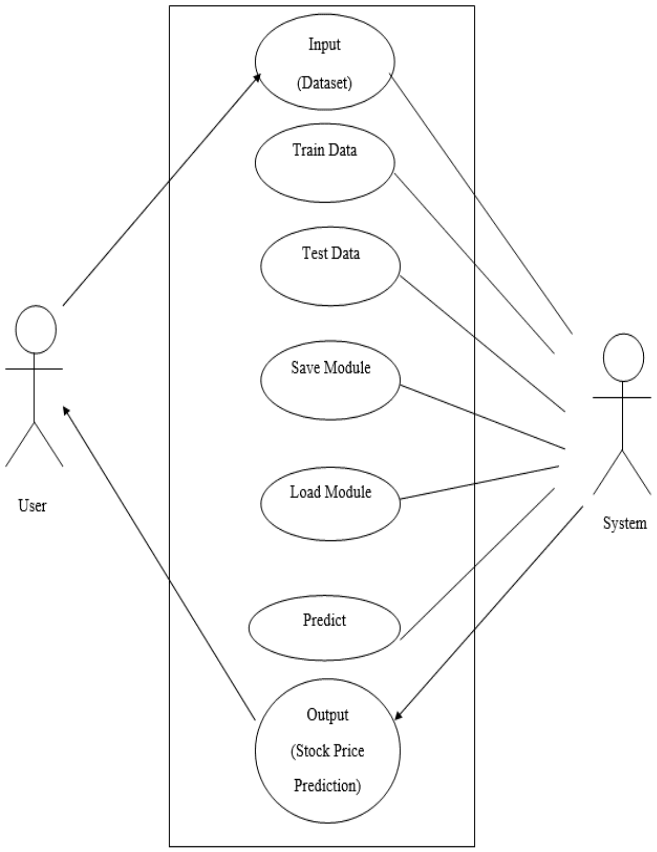


Figure 1.3

6.1.2. Activity Diagram

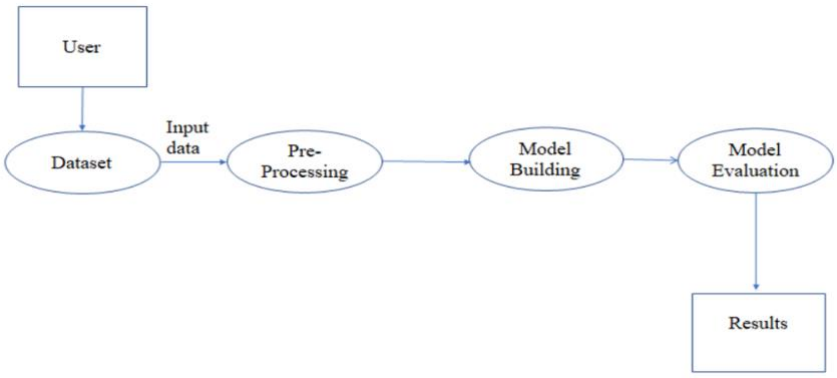


Figure 1.4

The primary focus of this project is to offer an extensive dataset containing historical stock prices for a selected company or index. This dataset is carefully curated from either a file or database and structured in a format that ensures its suitability for subsequent modeling processes. To enhance efficiency, the dataset undergoes a vital preprocessing phase, involving thorough cleaning and transformation procedures to tailor it to the chosen modeling approach.

For the modeling component, the project employs Long Short-Term Memory (LSTM), a specific type of recurrent neural network well-known for its ability to capture temporal dependencies in data. The model is trained using strategic applications of cross-validation or grid search methodologies, and its performance is rigorously evaluated using metrics like mean squared error and accuracy.

Progressing through the stages of model development, the constructed model undergoes evaluation and finalization using testing data. A detailed examination is conducted to identify and address potential issues such as overfitting or underfitting, ensuring the model's robustness and reliability. User feedback assumes a crucial role during the review process, offering insights into whether the model aligns with the expectations and requirements of its intended users.

Upon successful completion and validation, the finalized model is saved as a file or a database object, prepared for deployment and utilization in real-world scenarios.

In the operational phase, users input new data, which undergoes the same preprocessing procedures to ensure compatibility with the model. This preprocessed data is then input into the final model, which generates predicted stock prices based on the temporal patterns learned during training. The outcomes are presented to users through a user-friendly graphical interface or web page, enhancing accessibility and usability. This initiative holds significant potential to provide a valuable service for stock market enthusiasts and investors, delivering insights and predictions based on historical data and advanced modeling techniques.

6.1.3. Sequence Diagram

In the complex framework of a data processing system that leverages a machine learning model for predicting results based on user input, four integral components interact seamlessly: the User, Dataset and System.

At the forefront of this dynamic interplay stands the User, assuming the role of the system's initiator and prime mover. Users inject energy into the system by providing input through queries or observations, expressing specific needs, inquiries, or interests. The user's input acts

as the catalyst, setting the entire system into motion.

The Dataset plays a pivotal role as a reservoir, housing crucial information necessary for generating predictions. Within its confines lie historical data, intricate patterns, and essential features that form the foundation upon which the machine learning model can formulate well-informed predictions. The dataset emerges as a cornerstone, shaping both the training and evaluation phases of the machine learning model.

Functioning as the intermediary link connecting the user with the dataset, the System operates as the nerve center of the entire process. It meticulously processes user input, orchestrates the retrieval of relevant data from the dataset, and facilitates the prediction process through the machine learning model. The system adeptly manages the flow of information, ensuring the smooth translation of user input into meaningful and accurate predictions.

The Database serves as the organized repository and storage hub, playing a critical role in housing the dataset. It provides a structured and efficient means for accessing historical data and relevant information essential for accurate prediction. The database's significance extends to maintaining data integrity, ensuring accessibility, and optimizing retrieval efficiency—a crucial element contributing substantially to the overall functionality and performance of the system.

The interaction dynamics within this well-coordinated system unfold sequentially. The user initiates the process by providing input, sparking a chain reaction. The system, in turn, efficiently retrieves necessary data from the dataset stored in the database. Subsequently, the machine learning model comes into play, utilizing the retrieved data to formulate predictions or generate results. The collaborative synergy among these components orchestrates a smooth and efficient process, culminating in accurate result predictions based on user input.

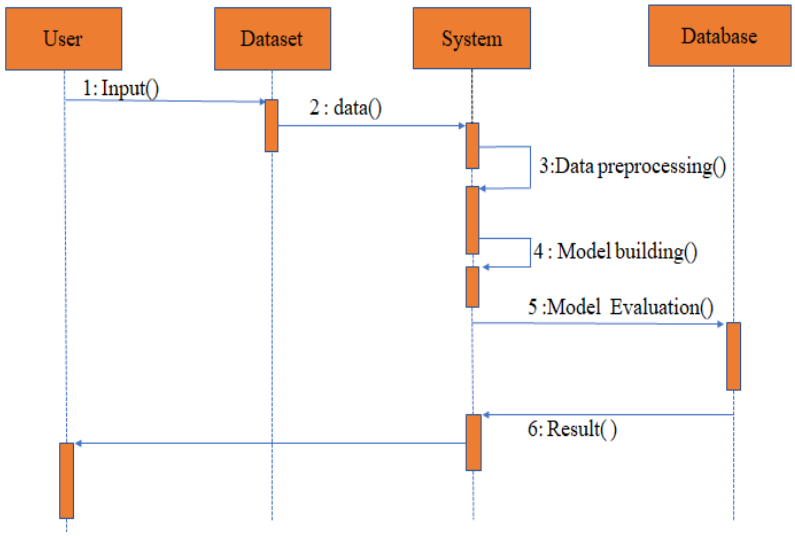


Figure 1.5

Following this, the system initiates the preprocessing of the data and the construction of the model, utilizing the LSTM (Long Short-Term Memory) algorithm. The model undergoes evaluation and finalization before being deployed to predict results based on the user's newly provided data. The outcomes of these predictions are stored in the database and subsequently presented to the user. This cohesive sequence of steps illustrates the synergy among the integral components, ensuring the delivery of efficient and user-centric data predictions.

6.1.4. Class Diagram

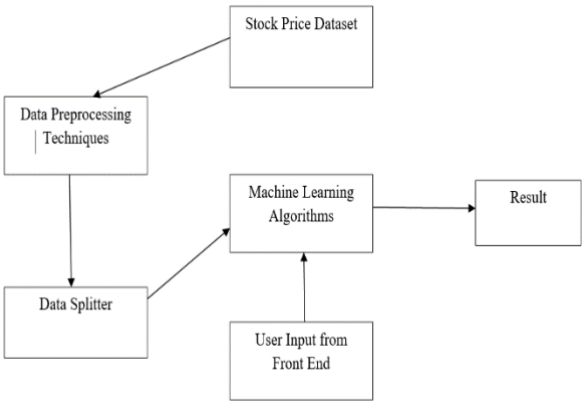


Figure 1.6

6.1.5. Flow Chart

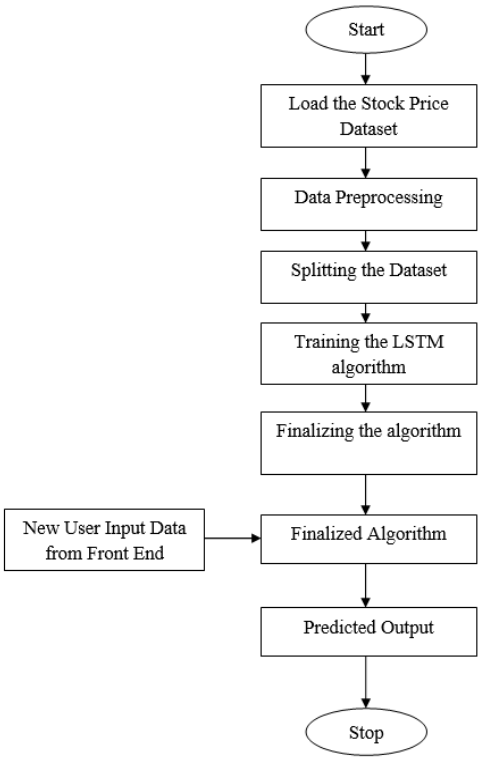


Figure 1.7

6.2. Implementation

6.2.1. Recurrent Neural Network (RNN)

Recurrent Neural Networks (RNNs) belong to a class of neural networks where the output from the preceding step is utilized as input for the current step. In contrast to traditional neural networks where inputs and outputs function independently, RNNs prove advantageous in tasks involving sequential data, such as predicting the subsequent word in a sentence. The introduction of a Hidden Layer in RNNs addresses the need to consider the context of previous elements in a sequence. A noteworthy and distinctive feature of RNNs is the Hidden State, a mechanism designed to preserve information about the sequence.

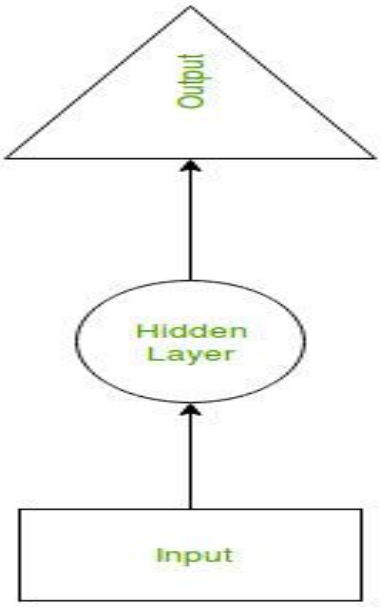


Figure 2.1

Every nuance of the computations an RNN has performed is preserved in its "memory". It uses the same parameters for every input and performs the same task on all inputs or hidden layers in order to produce the output. This reduces the parameter complexity compared to other neural networks.

How RNN works

Take a look at this example to learn how an RNN functions: Think of a deep network that has one input layer, one output layer, and three hidden layers. Similar to other neural networks, each hidden layer in a neural network has a distinct set of weights and biases. Weights and biases, for instance, would be (w1, b1) in the first hidden layer, (w2, b2) in the second hidden layer, and (w3, b3) in the third hidden layer. This shows that no layer can store data from previous outputs and that each layer operates independently.

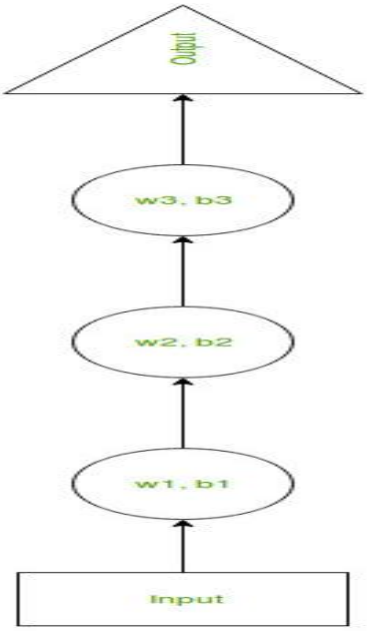


Figure 2.2

In the upcoming steps, the Recurrent Neural Network (RNN) will execute the following:

• Recurrent Neural Networks (RNNs) redefine independent activations into interdependent ones by employing uniform weights and biases across all layers. This approach mitigates the need for an exponential surge in parameters and circumvents the challenge of memorizing each preceding output. The network accomplishes this by feeding the output of each layer as input into the subsequent hidden layer, establishing a sequential information flow. This intrinsic feature equips RNNs with proficiency in handling tasks involving sequential data, effectively capturing dependencies across various time steps.

• Consequently, these three layers can be consolidated, amalgamating the weights and biases of all hidden layers into a unified recurrent layer.

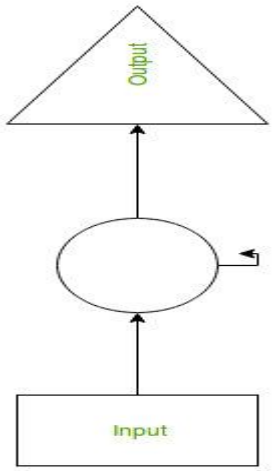


Figure 2.3

Formula for calculating current state:

$$h_t = f(h_{t-1}, X_t)$$

Where:

h_t ->current state

h_{t-1} -> previous state

x_t -> input state

Formula for applying Activation function(tanh):

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}X_t)$$

where:

w_{hh} -> weight at recurrent neuron

w_{xh} -> weight at input neuron

Formula for calculating output:

$$y_t = W_{hy}h_t$$

where:

Y_t -> output

W_{hy} -> weight at output layer

Training through RNN

1. The network undergoes the processing of a singular time step of the input.

2. The computation of the current state transpires by merging the current input with the previous state.

3. The resultant current state (ht) assumes the role of the input for the subsequent time step, displacing the preceding state (ht-1).

4. This cyclic process iterates for the necessary time steps dictated by the given problem, assimilating information from all antecedent states.

5. Upon the completion of all time steps, the ultimate current state is utilized to calculate the output.

6. The produced output is then compared to the target output, and the ensuing error is ascertained.

7. The error propagates backward through the network via the backpropagation mechanism, facilitating the fine-tuning of weights and the training of the Recurrent Neural Network (RNN).

Advantages of Recurrent Neural Networks (RNNs):

1. RNNs demonstrate proficiency in preserving and leveraging information over time, rendering them highly advantageous for time series prediction, courtesy of their Long Short-Term Memory (LSTM) capability. Furthermore, their effectiveness can be augmented by integrating convolutional layers to enhance the thorough consideration of pixel neighborhoods.

2. RNNs are adept at preserving and utilizing information over time, making them especially valuable for time series prediction, given their Long Short-Term Memory (LSTM) capability. Moreover, their effectiveness can be enhanced by incorporating convolutional layers, ensuring a more comprehensive consideration of pixel neighborhoods.

Drawbacks of Recurrent Neural Networks (RNNs):

1. RNNs struggle with explosion and gradient vanishing issues.

2. Training RNNs is a challenging task.

3. When employing tanh or relu as an activation function, processing extremely lengthy sequences can be challenging.6.2.2 Memory for Long Short Term (LSTM)

Long Short-Term Memory (LSTM):

The Long Short-Term Memory Network (LSTM), an enhanced variant of the Recurrent Neural Network (RNN) designed specifically for sequential tasks, offers a solution to the vanishing gradient problem that ordinary RNNs commonly face. Because of their ability to store data for a long time, RNNs are known in the field of persistent memory.

Consider reading a book and recalling the earlier chapters, or seeing yourself watching a movie and recalling the scenario that occurred before. Similar to this, RNNs work by processing incoming data using previously stored data. However, it is evident from the vanishing gradient problem that RNNs have limitations in terms of capturing long-term dependencies. LSTMs are a useful solution for managing and integrating long-term dependencies because they are expressly made to address these problems.

LSTM Architecture

In general, the Long Short-Term Memory (LSTM) and an RNN cell work similarly. The accompanying figure depicts the three primary parts, each of which plays a distinct role in the internal operations of the LSTM network.

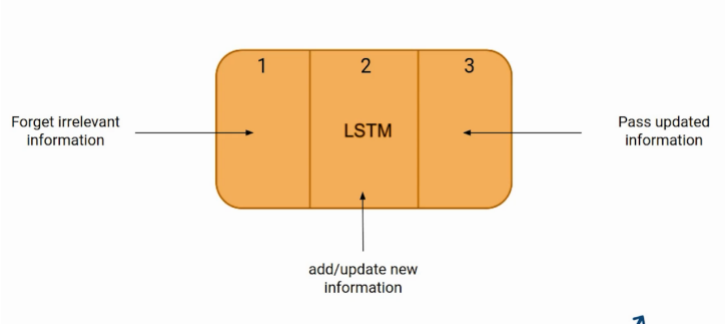


Figure 2.4

The decision of whether to retain or discard the data from the previous timestamp is made by the first section of the LSTM. Following that, the purpose of the second section is to extract fresh data from the input that this cell has received. The third and final section is responsible for transmitting the updated data from the current timestamp to the subsequent one. The forget gate, input gate, and output gate are the terms for the first, second, and third components of an LSTM cell, respectively. All of these components are called gates together.

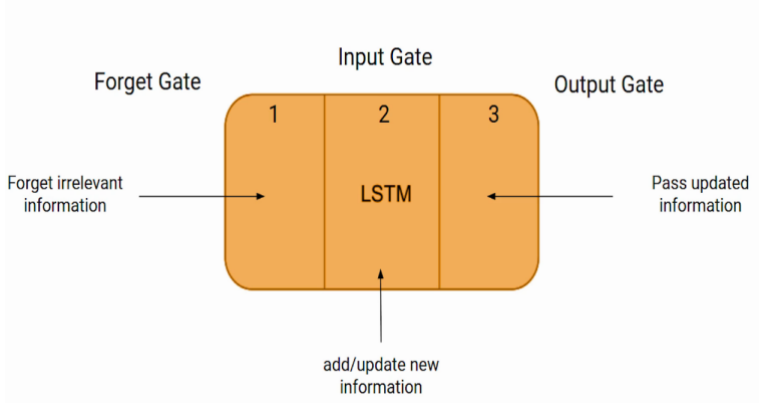


Figure 2.5

Similar to a simple RNN, an LSTM integrates a hidden state, where H(t-1) represents the hidden state of the previous timestamp, and Ht indicates the hidden state of the current timestamp. Furthermore, an LSTM incorporates a cell state denoted by C(t-1) for the previous timestamp and C(t) for the current timestamp. In this context, the hidden state is often termed Short-Term Memory, while the cell state is acknowledged as Long-Term

Memory. Please refer to the provided image for clarification.

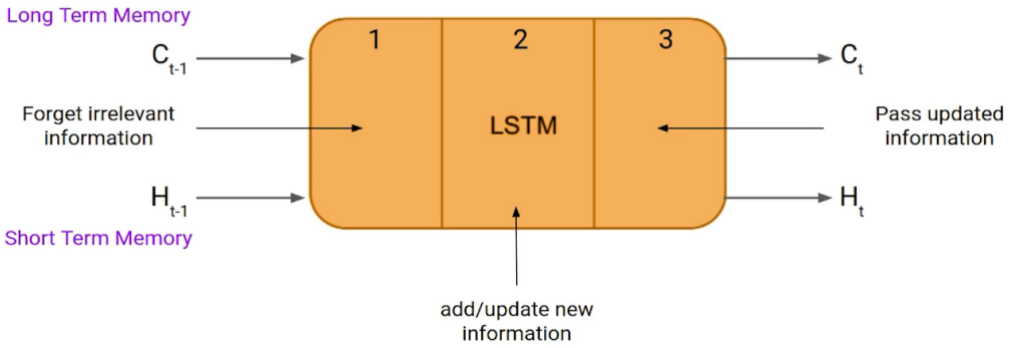


Figure 2.6

Figure 2.7

A notable observation is the ability of the cell state to preserve information across all timestamps.

To illustrate the functionality of an LSTM, let's consider an example with two sentences separated by a full stop. The initial sentence reads, "Bob is a nice person," followed by the second sentence, "Dan, on the other hand, is evil." The transition from the first statement to the second requires the network to recognize the shift in focus from Bob to Dan prompted by the introduction of the full stop (.).

This transition depends on the Forget gate in the LSTM architecture, which allows the network to discard Bob-related data and detects the change in subject. Consequently, the network may now concentrate on taking in and comprehending the data related to Dan. This illustration highlights the crucial functions that various gates perform inside the LSTM architecture.

Forget Gate

In the initial phase of a network incorporating an LSTM cell, a crucial task is to determine whether information from the previous timestamp should be preserved or discarded

Forget Gate:

- $f_t = \sigma\ (x_t * U_f + H_{t-1} * W_f)$

Let's break down the equation:

- ***Xt*** represents the input at the current timestamp.
- ***Uf*** denotes the weight associated with the input.
- ***Ht-1*** signifies the hidden state of the previous timestamp.
- ***Wf*** is the weight matrix associated with the hidden state.

Subsequently, a sigmoid function is applied to transform f_t into a value within the range of 0 to 1. This resulting f_t is then multiplied with the cell state from the previous timestamp, as illustrated below.

$$C_{t-1} * f_t = 0 \quad \text{....if } f_t = 0 \text{ (forget everything)}$$

$$C_{t-1} * f_t = C_{t-1} \quad \text{....if } f_t = 1 \text{ (forget nothing)}$$

If f_t equals 0, the network erases all stored information when t f_t is 1, it retains everything. In our example, the initial sentence centers on Bob. After encountering a full stop, the network shifts focus to Dan. Ideally, the network should forget about Bob in this scenario.

Input Gate

Let's look at another example: "Bob is a skilled swimmer. He informed me over the phone about his four challenging years in the service."

Although Bob is the subject of both remarks, they offer distinct insights on him. The first statement demonstrates his swimming ability, while the second one discusses his four years of navy service and phone use.

Given the context from the first statement, it becomes crucial to ascertain the important information in the second sentence, whether it has to do with phone usage or naval duty. In this perspective, it doesn't matter that he spoke with others over the phone; what matters is that he served in the navy.

The input gate manages this determination of importance. This evaluation process is captured by the input gate's equation, which determines the significance of the new information it has introduced.

Input Gate:

- $i_t = \sigma (x_t * U_i + H_{t-1} * W_i)$

Here,

- ***Xt***: Input at the current timestamp t
- ***Ui***: weight matrix of input

- ***Ht-1***: A hidden state at the previous timestamp
- ***Wi***: Weight matrix of input associated with hidden state

Once more, the sigmoid function is employed on 'I,' resulting in a value at timestamp t that ranges between 0 and 1.

New information

- $N_t = tanh\ (x_t * U_c + H_{t-1} * W_c)$ **(new information)**

The updated data allotted for the cell state is determined by the input (x) at timestamp t and the concealed state at timestamp t-1. To ensure that the new information value (Nt) is restricted to a value between -1 and 1, the tanh activation function is employed in this situation. If Nt is positive at the current timestamp, information is added to the cell state; if Nt is negative, information is subtracted from the cell state. However, instead of explicitly adding Nt to the cell state, the altered equation is shown. But by avoiding adding Nt directly to the cell state, the modified equation is introduced.

$$C_t = f_t * C_{t-1} + i_t * N_t \textbf{ (updating cell state)}$$

Here, *Ct-1* is the cell state at the current timestamp and others are the values that are calculated previously.

Output Gate

Using the following line as an example, "Bob killed the enemy on his own and gave his life in defense of his nation."

For his contributions, brave." Sentence completion challenges are mostly focused on the continuing task. It's critical to understand that when the word "brave" is used, it refers to one individual, in this case, Bob. Bob is the only one with this trait of bravery; it cannot be applied to the nation or the opponent.

Therefore, the task involves selecting a phrase that aligns with our expectations in order to give the statement a meaningful end. In this case, that particular term is the desired output,

and the output gate's task is to provide it.Using the following line as an example, "Bob killed the enemy on his own and gave his life in defense of his nation." For his contributions, brave."

Sentence completion challenges are mostly focused on the continuing task. It's critical to understand that when the word "brave" is used, it refers to one individual, in this case, Bob. Bob is the only one with this trait of bravery; it cannot be applied to the nation or the opponent.

Therefore, the task involves selecting a phrase that aligns with our expectations in order to give the statement a meaningful end. In this case, that particular term is the desired output, and the output gate's task is to provide it.

Output Gate

- $O_t = \sigma\ (x_t * U_o + H_{t-1} * W_0)$

The sigmoid function helps to restrict the value of the output gate (Ot) to be between 0 and 1. As can be seen here, Ot and the hyperbolic tangent (tanh) of the updated cell state are used to calculate the current concealed state.

$$H_t = O_t * tanh\ (C_t)$$

The present output and the long-term memory (Ct) appear to be related to the concealed state. Applying the SoftMax activation to the hidden state (Ht) will yield the output at the current timestamp.

$$Output = Softmax\ (H_t)$$

Here the token with the maximum score in the output is the prediction.

This is the More intuitive diagram of the LSTM network.

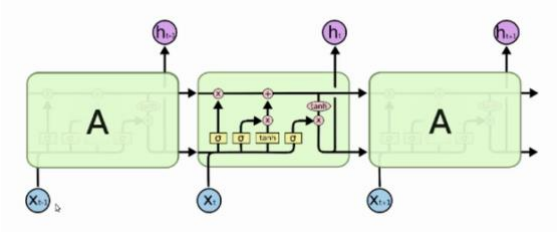


Figure 2.8

Dense layers

To extract the long-term dependencies that are present in stock data, LSTM (Long Short-Term Memory) models are widely utilized in the field of stock price prediction. Nevertheless, relying solely on LSTM models might not be sufficient to properly capture all the nuances included in the data. This is where the use of dense layers is necessary.

Neural network layers with fully linked layers—also called dense layers—have every input node tied to every output node. LSTM layers are typically placed before dense layers toward the conclusion of the model. They serve to categorize the traits that the LSTM was able to extract. They generate predictions that are appropriate for the specific task at hand, like as predicting the closing price of a stock, by utilizing the high-level, abstract representations generated by the LSTM layers.

For instance, in a stock prediction model, three LSTM layers might be used to find temporal dependencies in stock data, followed by three Dense layers that use the Rectified Linear Unit (ReLU) activation function. The Dense layers' task is to provide predictions after receiving the outputs from the LSTM layers, which contain newly discovered information about the input sequences.

To understand the temporal dependencies in stock data, for example, a stock prediction model could surely be constructed with three LSTM layers, then three Dense layers with an activation function of ReLU (Rectified Linear Unit). These Dense layers generate predictions once they receive the outputs from the LSTM layers, which include the knowledge they have learnt about the input sequences.

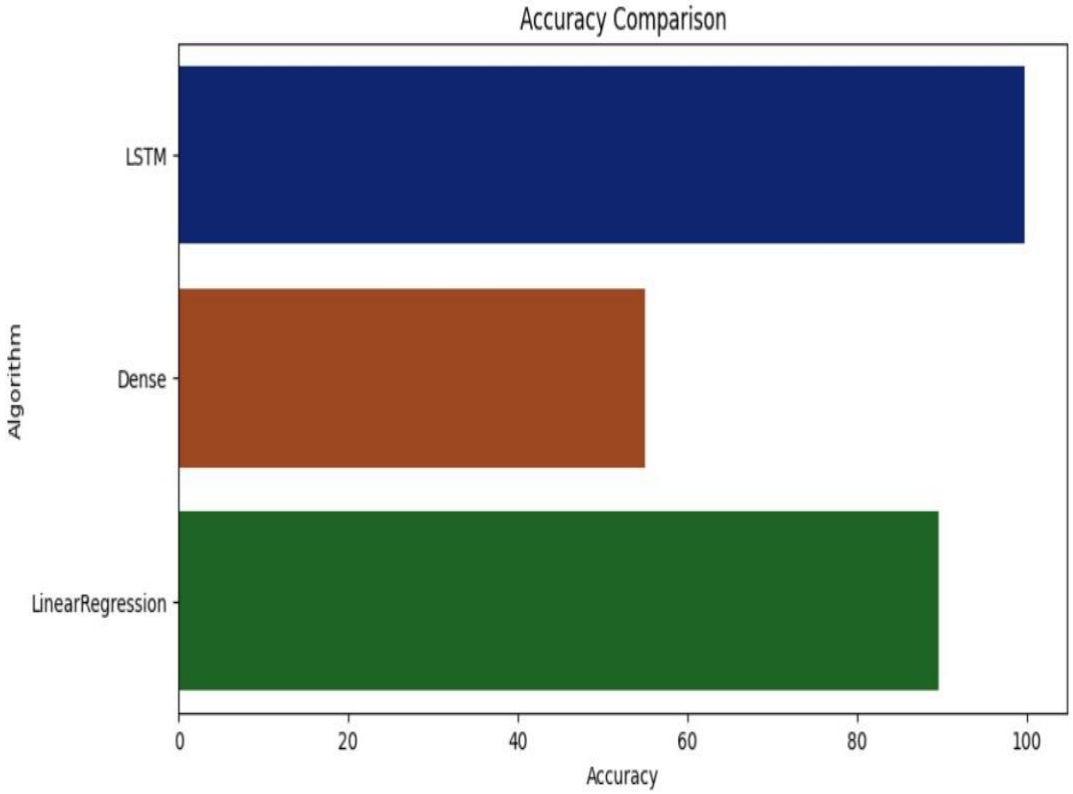
It is important to understand that, even while this strategy has the potential to be successful,

stock price prediction is fundamentally difficult because there are so many variables at play, and no model can provide 100% accuracy.

ACCURACY TABLE:

S.NO	X-axis	Y-axis
1.	99%	LSTM
2.	55%	Dense
3.	85%	Linear Regression

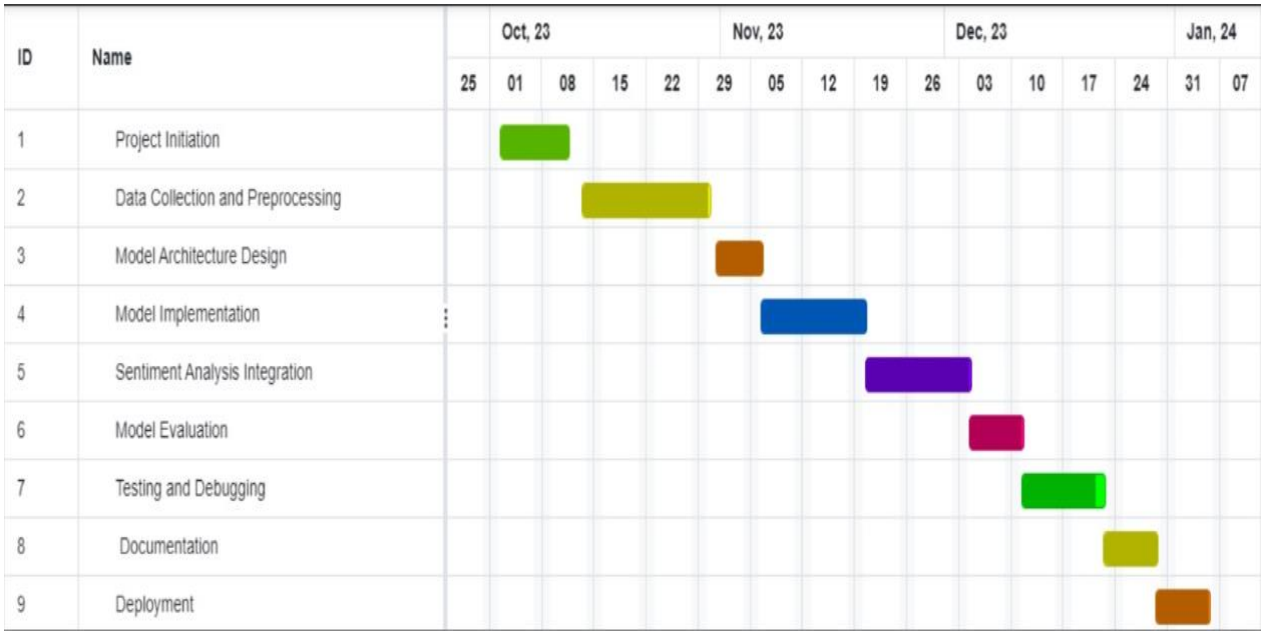
Table no 1.1


Accuracy Comparison
LSTM
Dense
LinearRegression
Algorithm
0
20
40
60
80
100
Accuracy

CHAPTER-7

TIMELINE FOR EXECUTION OF PROJECT

(GANTT CHART)



CHAPTER-8
OUTCOMES

1.Integration of Sentiment Analysis: -

A qualitative component is added to the stock price forecasting algorithm by including sentiment data. By examining sentiment in textual data, news articles, and social media, the model is able to determine market sentiment and public perception. Positive sentiment may indicate future increases in stock values, while negative sentiment may signal a decrease. Sentiment research has been incorporated into the predictive model to account for market sentiment.

2.**The integration of Long Short-Term Memory (LSTM)**:-

LSTM is advantageous in the analysis of time-series data, including stock prices. This is because LSTM is widely known for its ability to detect long-term dependencies in sequential data. The model's ability to identify and hold onto patterns over time improves its understanding of the stock market's dynamics. Due to its ability to handle temporal correlations, LSTM can anticipate more accurately by taking prior stock price trends and patterns into account.

3**. KNN Integration (K-Nearest Neighbors):** -

Using KNN, a non-parametric approach, helps to detect patterns in the data without strongly assuming anything about its underlying structure. KNN helps identify patterns in historical data and present market conditions when predicting stock prices. By detecting analogous circumstances, the integration of KNN into the model allows it to take advantage of collective knowledge embedded in similar historical instances, potentially improving prediction accuracy.

4.Dense (fully connected) Layers: -

The model's flexibility and adaptability are increased by the dense layers' incorporation, which captures intricate correlations between different features, such as sentiment ratings and historical stock prices. Dense layers enable a more thorough knowledge of the causes impacting stock price movements by enabling the model to learn complex patterns that may not be visible in individual features.

The evaluation and validation of the model necessitates a thorough assessment of its performance through the use of measures such as mean squared error, mean absolute error and explained variance score R2_score. To guarantee the generalizability of the model, this assessment should include both in-sample and out-of-sample testing.
Using cross-validation techniques helps to mitigate overfitting problems and validate the model's resilience.

5**. Limitations and Discussion**: -
Despite enhanced prediction skills with sentiment analysis and sophisticated algorithms, acknowledging the limitations is essential. Any prediction model has difficulties due to the stock market's inherent unpredictability and the potential for sentiment analysis to be inaccurate. To make accurate predictions, it is imperative to be vigilant about overfitting, which calls for ongoing model validation and improvement.

Ultimately, the incorporation of sentiment analysis into stock price prediction alongside LSTM, KNN, and dense algorithms exhibits potential for improving forecasting precision. For practical implementation in real-world financial settings, however, ongoing validation, improvement, and awareness of the model's limitations are required.

CHAPTER-9

RESULTS AND DISCUSSIONS

The LSTM and RNN algorithms are used in the suggested machine learning-based stock price forecasting system, which yields encouraging results. Using neural network models to better understand the intricate dynamics of the stock market and increase the accuracy of stock price predictions is a benefit.

Our model's predictive power is enhanced by the combination of LSTM, which is well-known for its ability to capture long-term dependencies in sequential data, and RNN, which is built to analyze sequential information. Our system can now more fully understand the complex patterns and trends present in stock market data thanks to this combination.

Extensive testing on a new stock price dataset demonstrates that our model performs considerably better at forecasting stock prices for the following day. By utilizing previous stock market value data, the approach produces highly accurate forecasts of future stock price changes. This kind of strategy is especially helpful in the financial sector, where dynamic shifts and real-time variations make accurate forecasts difficult.

But it's important to understand that stock price prediction is still a challenging task given the inherent unpredictability and volatility of financial markets. Although our method represents a substantial advance, one should still exercise caution when interpreting results and be cognizant of the hazards associated with stock market forecasting.

Research and development in the financial business has new prospects as a result of the use of machine learning techniques to stock price prediction. The prediction potential of our model could be further enhanced by including additional data, modifying parameters, and carrying out continuous enhancement. Additionally, ongoing validation and testing with a range of datasets will improve our system's robustness and dependability.

In conclusion, our proposed method that utilizes both the RNN and LSTM algorithms demonstrates promising outcomes in the area of stock price prediction.

CHAPTER 10
CONCLUSION

Inventory trading is an essential activity in the intricate world of finance. One aspect of stock market prediction is predicting future prices for stocks and other financial instruments that are traded on different exchanges. This is a difficult assignment. The three main methods used by stockbrokers to try and anticipate stocks accurately are time sequence, technical, and fundamental analysis. Stock price forecasting has always been difficult because of the market's intrinsic volatility due to its dynamic and real-time nature, even with the widespread use of these analytical techniques.

To handle the difficulties of predicting stock market trends, a number of strategies have been developed. In this context, we propose a machine learning-based approach to enhance the accuracy of stock price projections. The primary goal of our approach is to apply neural networks, namely the RNN (Recurrent Neural Networks) subclass known as the LSTM (Long Short-Term Memory) algorithm.

This sophisticated model, which employs recurrent neural networks to capture temporal connections, provides a strong foundation for precise stock price predictions. When our suggested system is put into practice, it makes use of a recently curated stock price dataset, and the results show a notable improvement in the accuracy of stock price prediction. Through the utilization of LSTM and RNN algorithms, our model skillfully navigates the intricacies present in the dynamics of the stock market, thereby contributing to increasingly precise and dependable forecasts of the next day's stock values.

REFERENCES

[1] P. K. Bharne and S. S. Prabhune: This paper introduces the fundamental principles of swarm intelligence and its application in optimizing daily stock market prices. It outlines various swarm intelligence algorithms such as ACO, PSO, BAT, and Firefly. The comparative analysis of recent swarm intelligence-based approaches concludes that SI-ANN yields more optimized results than SI with machine learning algorithms. The paper provides insights into recent SI trends and future directions.

[2] Mehar Vijha, Deeksha Chandolab, Vinay Anand Tikkiwalb, Arun Kumarc: The study addresses the limitations of historical datasets with minimal features, proposing the creation of new variables for improved accuracy in predicting stock prices. The paper employs Artificial Neural Networks (ANN) for predicting the next day's closing stock price and conducts a comparative analysis with Random Forest (RF). Results based on RMSE, MAPE, and MBE values indicate that ANN outperforms RF in stock price prediction.

[3]Hegazy, Osman & Soliman, Omar S: This paper proposes a machine learning model integrating the particle swarm optimization (PSO) algorithm and LS-SVM for stock price prediction using financial technical indicators. The study includes indicators like relative strength index, money flow index, exponential moving average, stochastic oscillator, and moving average convergence/divergence. LS-SVM-PSO achieves the lowest error value, surpassing single LS-SVM, while the ANN-BP algorithm exhibits lower performance.

[4] B. B. P. Maurya, A. Ray, A. Upadhyay, B. Gour, and A. U. Khan: The developed machine learning model enhances stock price anticipation precision by incorporating latest technical indicators - MACD, P/R, and Moving Averages. Moving Averages aid in identifying support and resistance, contributing to efficient stock prediction. The model utilizes web scraping for real-time data directly from the stock market, making it applicable to real-world and real-time stock problems.

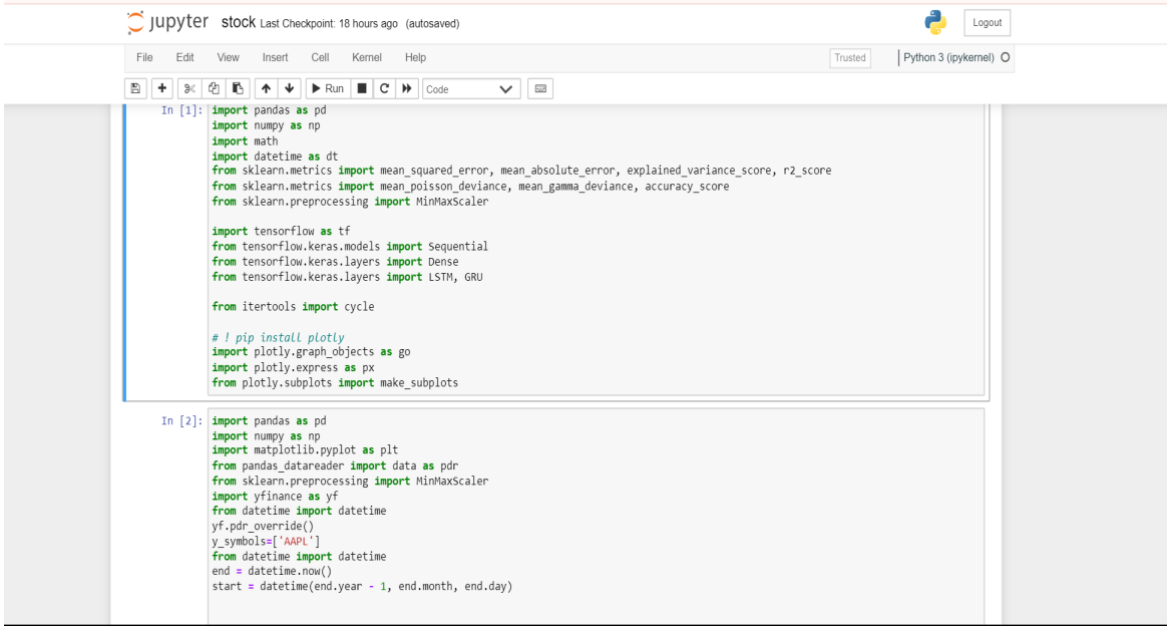
[5] I. Bhattacharjee and P. Bhattacharja: The paper conducts a comparative study between statistical approaches and machine learning approaches in terms of prediction performances and accuracy. Machine learning methods, especially MLP and LSTM, emerge as the most accurate for predicting stock prices, exhibiting the least MSE and MAPE values. The study

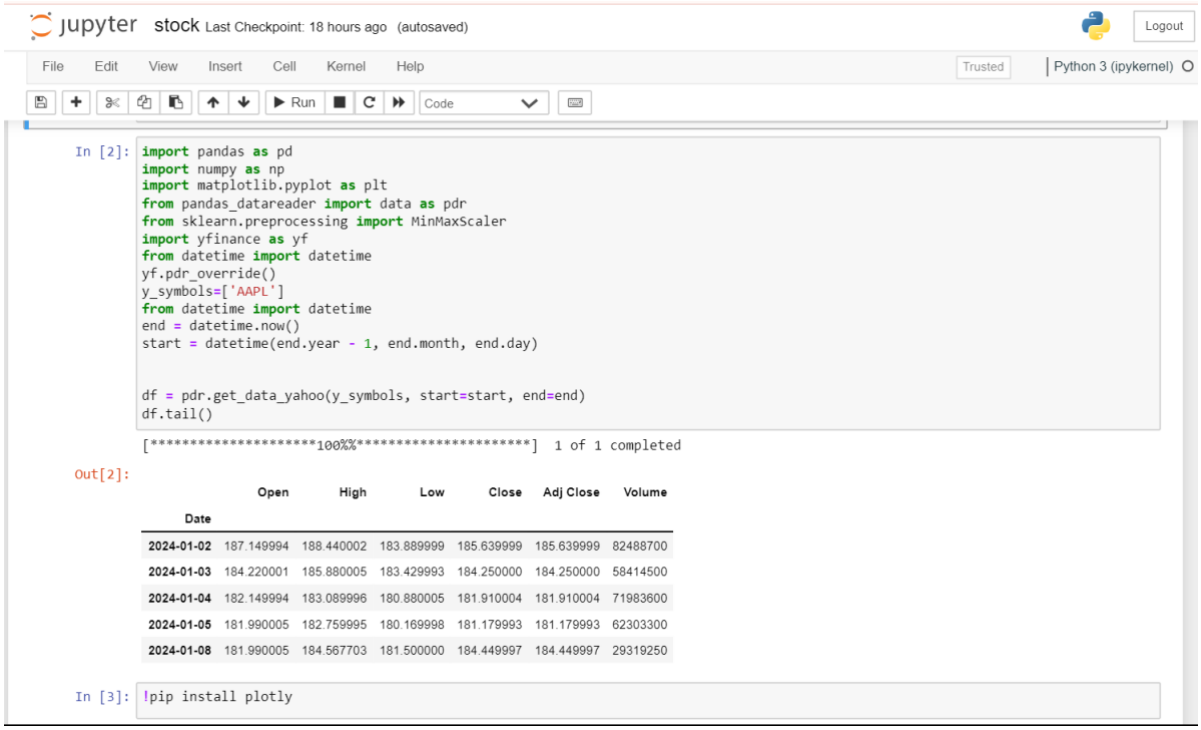
emphasizes the effectiveness of MLP and LSTM for accurate stock price predictions.

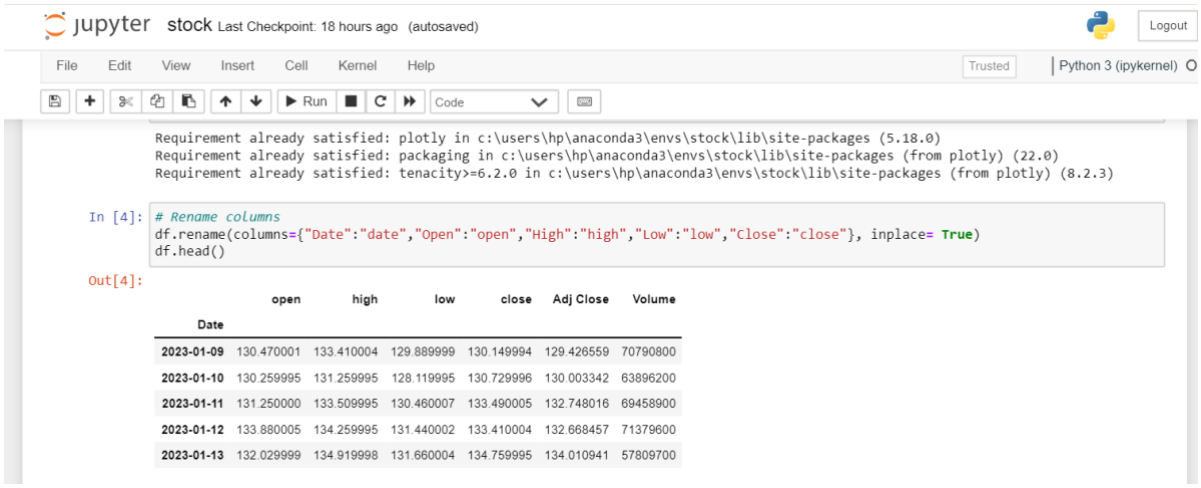
[6] U. Pasupulety, A. Abdullah Anees, S. Anmol, and B. R. Mohan: The proposed systematic approach for Stock Prediction utilizes Ensemble Learning and Sentiment Analysis. SVM and Extremely Randomized trees are employed, and a Stacked Regressor improves predicted prices based on individual model accuracy. Sentiment analysis of public opinion from tweets is integrated as an additional feature in the stock dataset, contributing to the overall prediction accuracy.

APPENDIX-A

PSUEDOCODE








```
Requirement already satisfied: plotly in c:\users\hp\anaconda3\envs\stock\lib\site-packages (5.18.0)
Requirement already satisfied: packaging in c:\users\hp\anaconda3\envs\stock\lib\site-packages (from plotly) (22.0)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\hp\anaconda3\envs\stock\lib\site-packages (from plotly) (8.2.3)
```

In [4]:
```
# Rename columns
df.rename(columns={"Date":"date","Open":"open","High":"high","Low":"low","Close":"close"}, inplace= True)
df.head()
```

Out[4]:

Date	open	high	low	close	Adj Close	Volume
2023-01-09	130.470001	133.410004	129.889999	130.149994	129.426559	70790800
2023-01-10	130.259995	131.259995	128.119995	130.729996	130.003342	63896200
2023-01-11	131.250000	133.509995	130.460007	133.490005	132.748016	69458900
2023-01-12	133.880005	134.259995	131.440002	133.410004	132.668457	71379600
2023-01-13	132.029999	134.919998	131.660004	134.759995	134.010941	57809700

In [5]:
```
# Resetting the index and renaming the index column from 'Date' to 'date'
df.reset_index(inplace=True)
df.rename(columns={'Date': 'date'}, inplace=True)
df.set_index('date', inplace=True)

# Displaying the head of the DataFrame to verify changes
print(df.head())
```

```
                open        high         low       close   Adj Close  \
date
2023-01-09  130.470001  133.410004  129.889999  130.149994  129.426559
2023-01-10  130.259995  131.259995  128.119995  130.729996  130.003342
2023-01-11  131.250000  133.509995  130.460007  133.490005  132.748016
2023-01-12  133.880005  134.259995  131.440002  133.410004  132.668457
2023-01-13  132.029999  134.919998  131.660004  134.759995  134.010941

              Volume
date
2023-01-09  70790800
2023-01-10  63896200
2023-01-11  69458900
2023-01-12  71379600
2023-01-13  57809700
```

In [6]:
```
# If the index is not in datetime format, convert it
if not isinstance(df.index, pd.DatetimeIndex):
    df.index = pd.to_datetime(df.index)

# Displaying the head of the DataFrame to verify changes
print(df.head())

# If you want to reset the index back to a column:
df.reset_index(inplace=True)
# convert date field from string to Date format and make it index
df['date'] = pd.to_datetime(df.date)
df.head()
```

```
                open        high         low       close   Adj Close  \
date
2023-01-09  130.470001  133.410004  129.889999  130.149994  129.426559
2023-01-10  130.259995  131.259995  128.119995  130.729996  130.003342
2023-01-11  131.250000  133.509995  130.460007  133.490005  132.748016
2023-01-12  133.880005  134.259995  131.440002  133.410004  132.668457
2023-01-13  132.029999  134.919998  131.660004  134.759995  134.010941

              Volume
date
2023-01-09  70790800
2023-01-10  63896200
2023-01-11  69458900
2023-01-12  71379600
2023-01-13  57809700
```

Out[6]:

	date	open	high	low	close	Adj Close	Volume
0	2023-01-09	130.470001	133.410004	129.889999	130.149994	129.426559	70790800
1	2023-01-10	130.259995	131.259995	128.119995	130.729996	130.003342	63896200
2	2023-01-11	131.250000	133.509995	130.460007	133.490005	132.748016	69458900
3	2023-01-12	133.880005	134.259995	131.440002	133.410004	132.668457	71379600
4	2023-01-13	132.029999	134.919998	131.660004	134.759995	134.010941	57809700

In [7]:
```
df.sort_values(by='date', inplace=True)
df.head()
```

Out[7]:

	date	open	high	low	close	Adj Close	Volume
0	2023-01-09	130.470001	133.410004	129.889999	130.149994	129.426559	70790800
1	2023-01-10	130.259995	131.259995	128.119995	130.729996	130.003342	63896200
2	2023-01-11	131.250000	133.509995	130.460007	133.490005	132.748016	69458900
3	2023-01-12	133.880005	134.259995	131.440002	133.410004	132.668457	71379600
4	2023-01-13	132.029999	134.919998	131.660004	134.759995	134.010941	57809700

In [8]:
```
print("Starting date: ",df.iloc[0][0])
print("Ending date: ", df.iloc[-1][0])
print("Duration: ", df.iloc[-1][0]-df.iloc[0][0])
```

```
Starting date:  2023-01-09 00:00:00
Ending date:  2024-01-08 00:00:00
Duration:  364 days 00:00:00
```

In [9]:
```
closedf = df[['date','close']]
print("Shape of close dataframe:", closedf.shape)
```

```
Shape of close dataframe: (251, 2)
```

In [10]:
```
close_stock = closedf.copy()
del closedf['date']
scaler=MinMaxScaler(feature_range=(0,1))
closedf=scaler.fit_transform(np.array(closedf).reshape(-1,1))
print(closedf.shape)
```

```
(251, 1)
```

In [11]:
```
training_size=int(len(closedf)*0.65)
test_size=len(closedf)-training_size
train_data,test_data=closedf[0:training_size,:],closedf[training_size:len(closedf),:1]
print("train_data: ", train_data.shape)
print("test_data: ", test_data.shape)
```

```
train_data:  (163, 1)
test_data:  (88, 1)
```

In [12]:
```
# convert an array of values into a dataset matrix
def create_dataset(dataset, time_step=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-time_step-1):
        a = dataset[i:(i+time_step), 0]   ###i=0, 0,1,2,3-----99   100
        dataX.append(a)
        dataY.append(dataset[i + time_step, 0])
    return np.array(dataX), np.array(dataY)
```

In [13]:
```
# reshape into X=t,t+1,t+2,t+3 and Y=t+4
time_step = 15
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)

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

```
X_train:  (147, 15)
y_train:  (147,)
X_test:  (72, 15)
y_test (72,)
```

In [14]:
```
# reshape input to be [samples, time steps, features] which is required for LSTM
X_train =X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
X_test = X_test.reshape(X_test.shape[0],X_test.shape[1] , 1)

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

```
X_train:  (147, 15, 1)
X_test:  (72, 15, 1)
```

In [15]:
```
tf.keras.backend.clear_session()
model=Sequential()
model.add(LSTM(32,return_sequences=True,input_shape=(time_step,1)))
model.add(LSTM(32,return_sequences=True))
model.add(LSTM(32))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
```

In [16]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 15, 32)            4352

 lstm_1 (LSTM)               (None, 15, 32)            8320

 lstm_2 (LSTM)               (None, 32)                8320

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 21,025
Trainable params: 21,025
Non-trainable params: 0
_________________________________________________________________
```

In [18]:
```
### Lets Do the prediction and check performance metrics
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
train_predict.shape, test_predict.shape
```

```
5/5 [==============================] - 2s 7ms/step
3/3 [==============================] - 0s 8ms/step
```

Out[18]:
```
((147, 1), (72, 1))
```

In [19]:
```
# Transform back to original form

train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
original_ytrain = scaler.inverse_transform(y_train.reshape(-1,1))
original_ytest = scaler.inverse_transform(y_test.reshape(-1,1))
```

In [20]:
```
# Evaluation metrices RMSE and MAE
print("Train data RMSE: ", math.sqrt(mean_squared_error(original_ytrain,train_predict)))
print("Train data MSE: ", mean_squared_error(original_ytrain,train_predict))
print("Test data MAE: ", mean_absolute_error(original_ytrain,train_predict))
print("-------------------------------------------------------------------------------------")
print("Test data RMSE: ", math.sqrt(mean_squared_error(original_ytest,test_predict)))
print("Test data MSE: ", mean_squared_error(original_ytest,test_predict))
print("Test data MAE: ", mean_absolute_error(original_ytest,test_predict))
```

```
Train data RMSE:  2.279103922443611
Train data MSE:  5.194314689297853
Test data MAE:  1.8405108808660182
-------------------------------------------------------------------------------------
Test data RMSE:  2.221320751313987
Test data MSE:  4.9342658802181365
Test data MAE:  1.72157965766059
```

In [21]:
```
print("Train data explained variance regression score:", explained_variance_score(original_ytrain, train_predict))
print("Test data explained variance regression score:", explained_variance_score(original_ytest, test_predict))
```

```
Train data explained variance regression score: 0.9788901730706446
Test data explained variance regression score: 0.940798301424161
```

In [22]:
```
print("Train data R2 score:", r2_score(original_ytrain, train_predict))
print("Test data R2 score:", r2_score(original_ytest, test_predict))
```

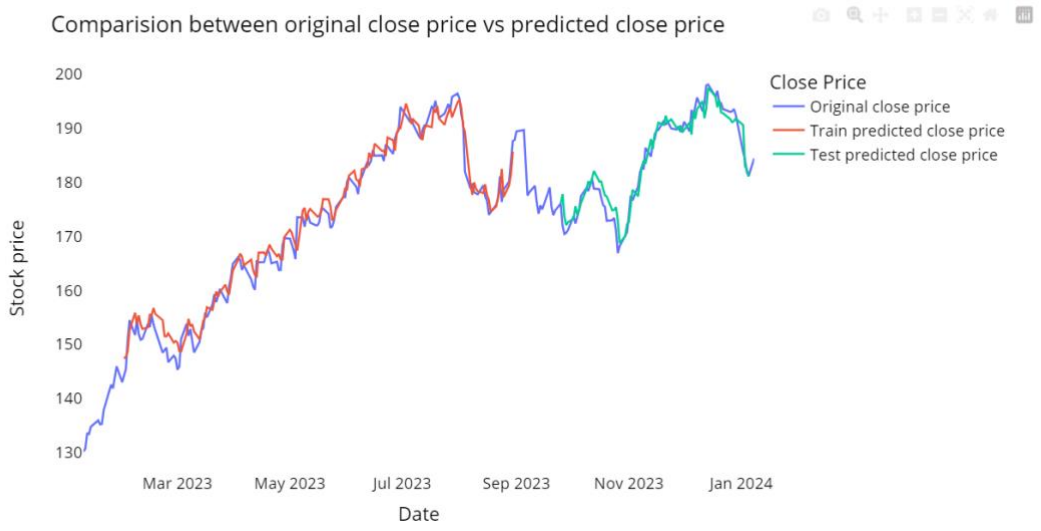
```
Train data R2 score: 0.9762065077244192
Test data R2 score: 0.9391817935402239
```

In [23]:
```
print("Train data MGD: ", mean_gamma_deviance(original_ytrain, train_predict))
print("Test data MGD: ", mean_gamma_deviance(original_ytest, test_predict))
print("----------------------------------------------------------------------")
print("Train data MPD: ", mean_poisson_deviance(original_ytrain, train_predict))
print("Test data MPD: ", mean_poisson_deviance(original_ytest, test_predict))
```

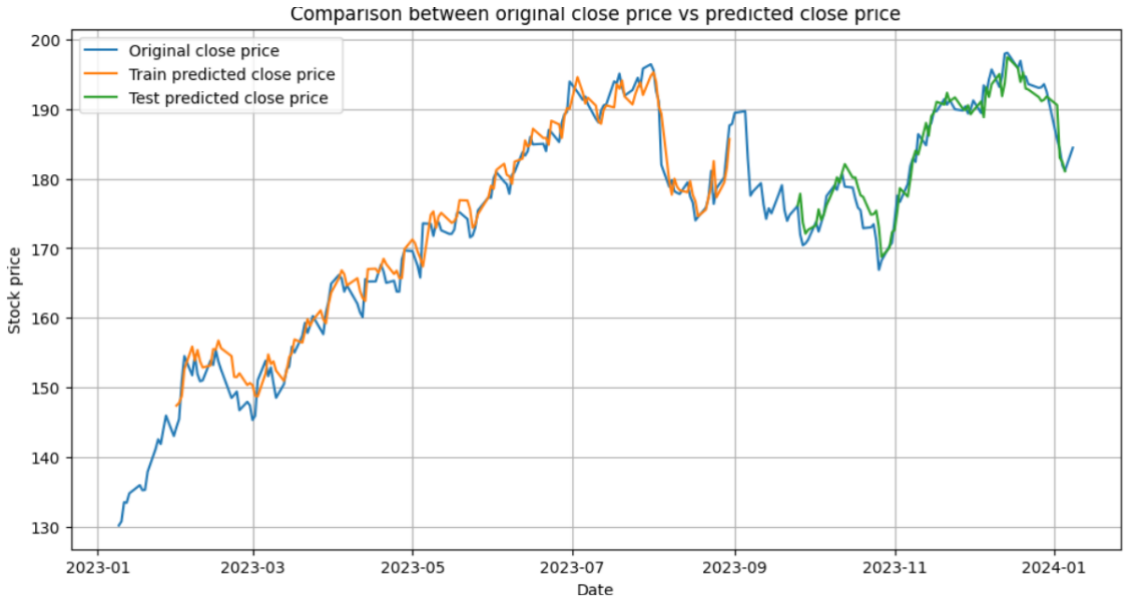
```
Train data MGD:  0.0001870543545604511
Test data MGD:  0.0001500259480390219
----------------------------------------------------------------------
Train data MPD:  0.0310492627948748
Test data MPD:  0.027179302034234354
```

In [24]:
```
# shift train predictions for plotting

look_back=time_step
trainPredictPlot = np.empty_like(closedf)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
print("Train predicted data: ", trainPredictPlot.shape)

# shift test predictions for plotting
testPredictPlot = np.empty_like(closedf)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(closedf)-1, :] = test_predict
print("Test predicted data: ", testPredictPlot.shape)

names = cycle(['Original close price','Train predicted close price','Test predicted close price'])


plotdf = pd.DataFrame({'date': close_stock['date'],
                       'original_close': close_stock['close'],
                      'train_predicted_close': trainPredictPlot.reshape(1,-1)[0].tolist(),
                      'test_predicted_close': testPredictPlot.reshape(1,-1)[0].tolist()})

fig = px.line(plotdf,x=plotdf['date'], y=[plotdf['original_close'],plotdf['train_predicted_close'],
                                          plotdf['test_predicted_close']],
              labels={'value':'Stock price','date': 'Date'})
fig.update_layout(title_text='Comparision between original close price vs predicted close price',
                  plot_bgcolor='white', font_size=15, font_color='black', legend_title_text='Close Price')
fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

```
Train predicted data:  (251, 1)
Test predicted data:  (251, 1)
```



In [25]:
```
import matplotlib.pyplot as plt

# Your code to generate trainPredictPlot, testPredictPlot, and plotdf remains unchanged

# Plotting the data using Matplotlib
plt.figure(figsize=(12, 6))

# Plot original close price
plt.plot(plotdf['date'], plotdf['original_close'], label='Original close price')

# Plot train predicted close price
plt.plot(plotdf['date'], plotdf['train_predicted_close'], label='Train predicted close price')

# Plot test predicted close price
plt.plot(plotdf['date'], plotdf['test_predicted_close'], label='Test predicted close price')

plt.title('Comparison between original close price vs predicted close price')
plt.xlabel('Date')
plt.ylabel('Stock price')
plt.legend()
plt.grid(True)
plt.show()
```



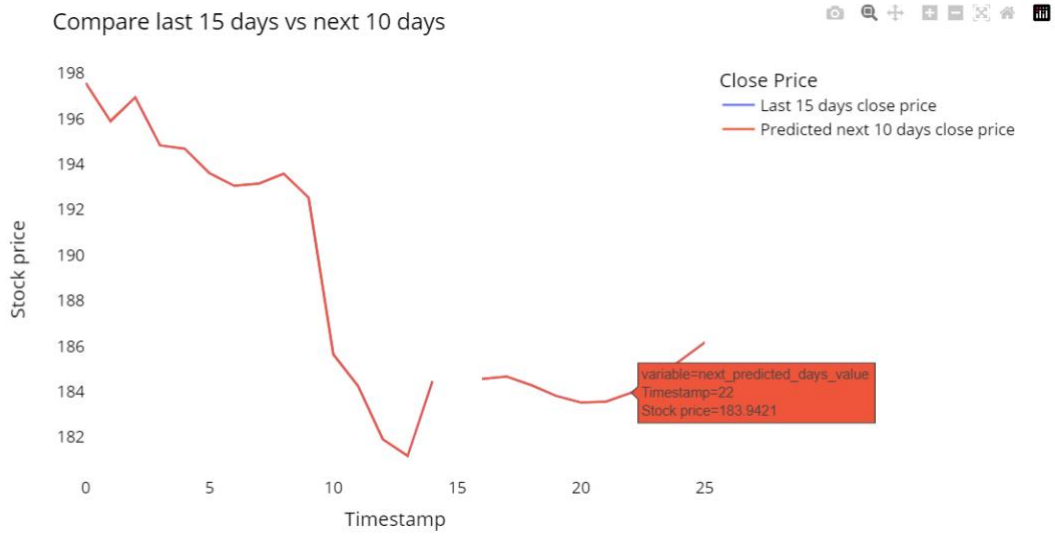
In [26]:
```
x_input=test_data[len(test_data)-time_step:].reshape(1,-1)
temp_input=list(x_input)
temp_input=temp_input[0].tolist()

from numpy import array

lst_output=[]
n_steps=time_step
i=0
pred_days = 10
while(i<pred_days):

    if(len(temp_input)>time_step):

        x_input=np.array(temp_input[1:])
        #print("{} day input {}".format(i,x_input))
        x_input = x_input.reshape(1,-1)
        x_input = x_input.reshape((1, n_steps, 1))

        yhat = model.predict(x_input, verbose=0)
        #print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        #print(temp_input)

        lst_output.extend(yhat.tolist())
        i=i+1

    else:

        x_input = x_input.reshape((1, n_steps,1))
        yhat = model.predict(x_input, verbose=0)
        temp_input.extend(yhat[0].tolist())

        lst_output.extend(yhat.tolist())
        i=i+1

print("Output of predicted next days: ", len(lst_output))
```

```
Output of predicted next days:  10
```

In [27]:
```
last_days=np.arange(1,time_step+1)
day_pred=np.arange(time_step+1,time_step+pred_days+1)
print(last_days)
print(day_pred)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
[16 17 18 19 20 21 22 23 24 25]
```

In [28]:
```
temp_mat = np.empty((len(last_days)+pred_days+1,1))
temp_mat[:] = np.nan
temp_mat = temp_mat.reshape(1,-1).tolist()[0]

last_original_days_value = temp_mat
next_predicted_days_value = temp_mat

last_original_days_value[0:time_step+1] = scaler.inverse_transform(closedf[len(closedf)-time_step:]).reshape(1,-1).tolist()[0]
next_predicted_days_value[time_step+1:] = scaler.inverse_transform(np.array(lst_output).reshape(-1,1)).reshape(1,-1).tolist()[0]

new_pred_plot = pd.DataFrame({
    'last_original_days_value':last_original_days_value,
    'next_predicted_days_value':next_predicted_days_value
})

names = cycle(['Last 15 days close price','Predicted next 10 days close price'])

fig = px.line(new_pred_plot,x=new_pred_plot.index, y=[new_pred_plot['last_original_days_value'],
                                                      new_pred_plot['next_predicted_days_value']],
              labels={'value': 'Stock price','index': 'Timestamp'})
fig.update_layout(title_text='Compare last 15 days vs next 10 days',
                  plot_bgcolor='white', font_size=15, font_color='black',legend_title_text='Close Price')

fig.for_each_trace(lambda t:  t.update(name = next(names)))
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```



In [29]:
```
lstmdf=closedf.tolist()
lstmdf.extend((np.array(lst_output).reshape(-1,1)).tolist())
lstmdf=scaler.inverse_transform(lstmdf).reshape(1,-1).tolist()[0]

names = cycle(['Close price'])

fig = px.line(lstmdf,labels={'value': 'Stock price','index': 'Timestamp'})
fig.update_layout(title_text='Plotting whole closing stock price with prediction',
                  plot_bgcolor='white', font_size=15, font_color='black',legend_title_text='Stock')

fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

In [30]:
```
# reshape input to be [samples, time steps, features] which is required for LSTM
X_train =X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
X_test = X_test.reshape(X_test.shape[0],X_test.shape[1] , 1)

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

```
X_train:  (147, 15, 1)
X_test:  (72, 15, 1)
```

In [31]:
```
tf.keras.backend.clear_session()
model=Sequential()
model.add(LSTM(32,return_sequences=True,input_shape=(time_step,1)))
model.add(LSTM(32,return_sequences=True))
model.add(GRU(32,return_sequences=True))
model.add(GRU(32))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
```

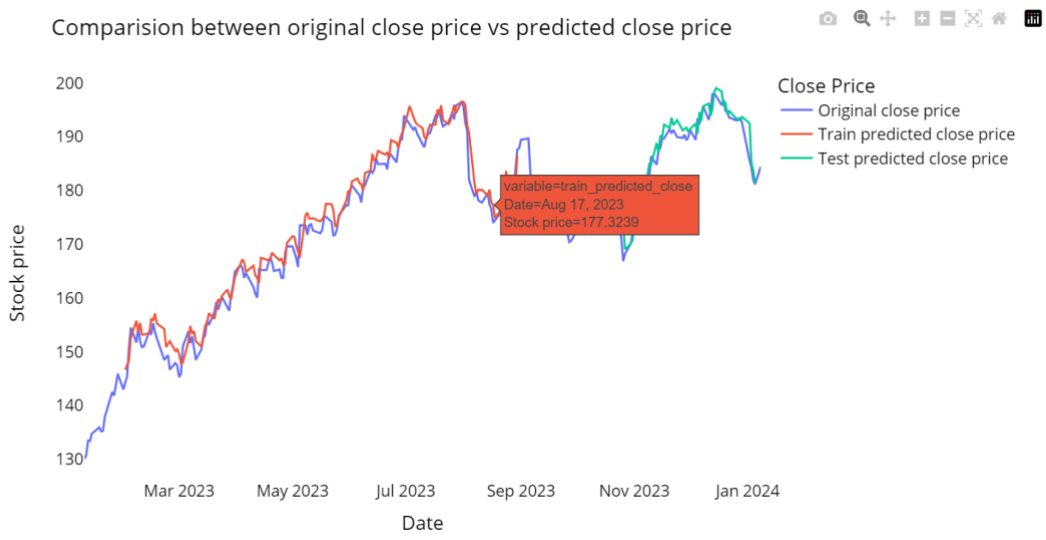
In [32]:
```
model.summary()
```

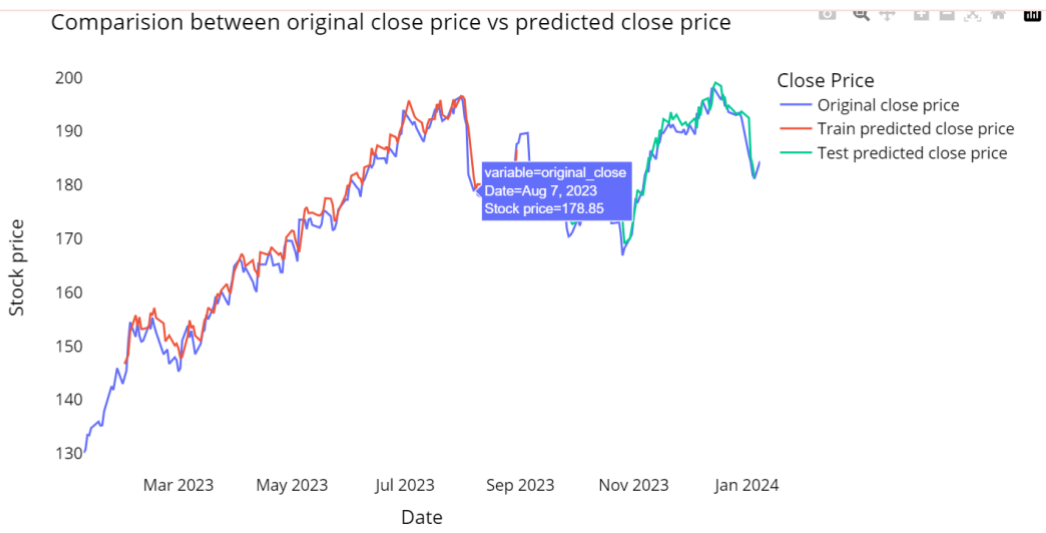
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 15, 32)            4352

 lstm_1 (LSTM)               (None, 15, 32)            8320

 gru (GRU)                   (None, 15, 32)            6336

 gru_1 (GRU)                 (None, 32)                6336

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 25,377
Trainable params: 25,377
Non-trainable params: 0
_________________________________________________________________
```

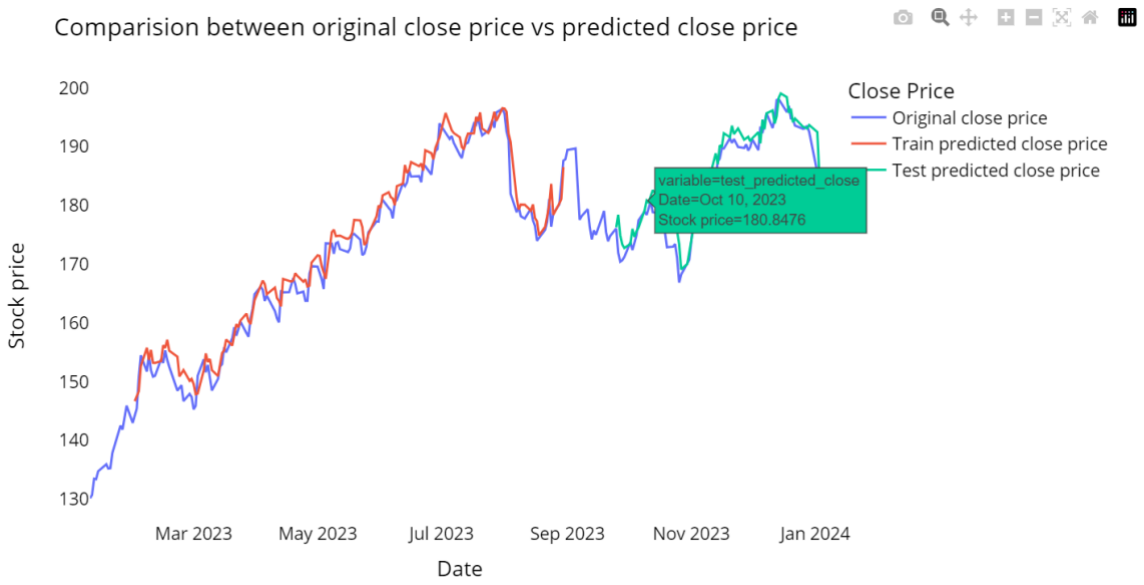
In [35]:
```
# Transform back to original form

train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
original_ytrain = scaler.inverse_transform(y_train.reshape(-1,1))
original_ytest = scaler.inverse_transform(y_test.reshape(-1,1))
```

In [36]:
```
# shift train predictions for plotting

look_back=time_step
trainPredictPlot = np.empty_like(closedf)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
print("Train predicted data: ", trainPredictPlot.shape)

# shift test predictions for plotting
testPredictPlot = np.empty_like(closedf)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(closedf)-1, :] = test_predict
print("Test predicted data: ", testPredictPlot.shape)

names = cycle(['Original close price','Train predicted close price','Test predicted close price'])

plotdf = pd.DataFrame({'date': close_stock['date'],
                       'original_close': close_stock['close'],
                      'train_predicted_close': trainPredictPlot.reshape(1,-1)[0].tolist(),
                      'test_predicted_close': testPredictPlot.reshape(1,-1)[0].tolist()})

fig = px.line(plotdf,x=plotdf['date'], y=[plotdf['original_close'],plotdf['train_predicted_close'],
                                          plotdf['test_predicted_close']],
              labels={'value':'Stock price','date': 'Date'})
fig.update_layout(title_text='Comparision between original close price vs predicted close price',
                  plot_bgcolor='white', font_size=15, font_color='black', legend_title_text='Close Price')
fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

```
Train predicted data:  (251, 1)
Test predicted data:  (251, 1)
```




Comparision between original close price vs predicted close price
Close Price
Original close price
Train predicted close price
Test predicted close price
variable=original_close
Date=Aug 7, 2023
Stock price=178.85
Stock price
200
190
180
170
160
150
140
130
Mar 2023
May 2023
Jul 2023
Sep 2023
Nov 2023
Jan 2024
Date


Comparision between original close price vs predicted close price
Close Price
Original close price
Train predicted close price
Test predicted close price
variable=test_predicted_close
Date=Oct 10, 2023
Stock price=180.8476
Stock price
200
190
180
170
160
150
140
130
Mar 2023
May 2023
Jul 2023
Sep 2023
Nov 2023
Jan 2024
Date

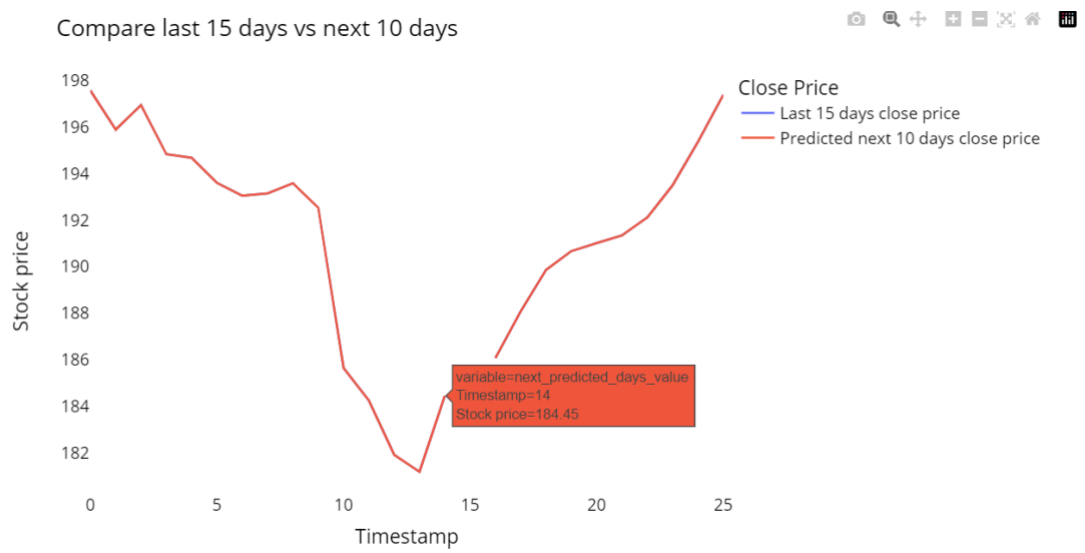
In [37]:
```
x_input=test_data[len(test_data)-time_step:].reshape(1,-1)
temp_input=list(x_input)
temp_input=temp_input[0].tolist()

from numpy import array

lst_output=[]
n_steps=time_step
i=0
pred_days = 10
while(i<pred_days):

    if(len(temp_input)>time_step):

        x_input=np.array(temp_input[1:])
        #print("{} day input {}".format(i,x_input))
        x_input = x_input.reshape(1,-1)
        x_input = x_input.reshape((1, n_steps, 1))

        yhat = model.predict(x_input, verbose=0)
        #print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        #print(temp_input)

        lst_output.extend(yhat.tolist())
        i=i+1

    else:

        x_input = x_input.reshape((1, n_steps,1))
        yhat = model.predict(x_input, verbose=0)
        temp_input.extend(yhat[0].tolist())

        lst_output.extend(yhat.tolist())
        i=i+1

print("Output of predicted next days: ", len(lst_output))
```

```
Output of predicted next days:  10
```

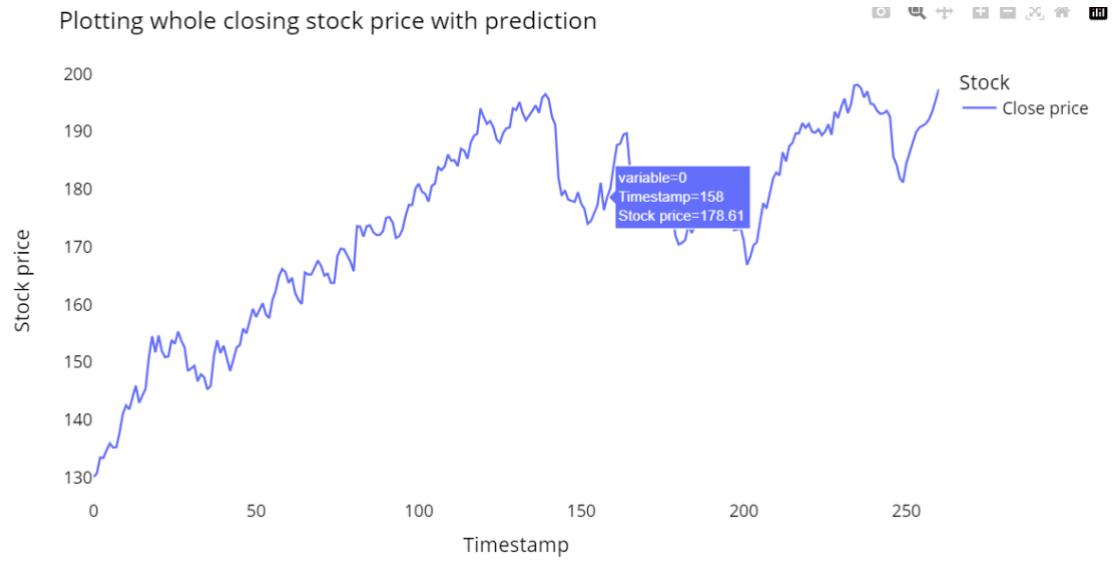
In [38]:
```
last_days=np.arange(1,time_step+1)
day_pred=np.arange(time_step+1,time_step+pred_days+1)
print(last_days)
print(day_pred)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
[16 17 18 19 20 21 22 23 24 25]
```

In [39]:
```
temp_mat = np.empty((len(last_days)+pred_days+1,1))
temp_mat[:] = np.nan
temp_mat = temp_mat.reshape(1,-1).tolist()[0]

last_original_days_value = temp_mat
next_predicted_days_value = temp_mat

last_original_days_value[0:time_step+1] = scaler.inverse_transform(closedf[len(closedf)-time_step:]).reshape(1,-1).tolist()[0]
next_predicted_days_value[time_step+1:] = scaler.inverse_transform(np.array(lst_output).reshape(-1,1)).reshape(1,-1).tolist()[0]

new_pred_plot = pd.DataFrame({
    'last_original_days_value':last_original_days_value,
    'next_predicted_days_value':next_predicted_days_value
})
names = cycle(['Last 15 days close price','Predicted next 10 days close price'])

fig = px.line(new_pred_plot,x=new_pred_plot.index, y=[new_pred_plot['last_original_days_value'],
                                                      new_pred_plot['next_predicted_days_value']],
              labels={'value': 'Stock price','index': 'Timestamp'})
fig.update_layout(title_text='Compare last 15 days vs next 10 days',
                  plot_bgcolor='white', font_size=15, font_color='black', legend_title_text='Close Price')
fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```



In [40]:
```
lstmgrudf=closedf.tolist()
lstmgrudf.extend((np.array(lst_output).reshape(-1,1)).tolist())
lstmgrudf=scaler.inverse_transform(lstmgrudf).reshape(1,-1).tolist()[0]

names = cycle(['Close price'])

fig = px.line(lstmgrudf,labels={'value': 'Stock price','index': 'Timestamp'})
fig.update_layout(title_text='Plotting whole closing stock price with prediction',
                  plot_bgcolor='white', font_size=15, font_color='black', legend_title_text='Stock')
fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```



In [41]:
```
# Assuming 'model' is your trained Keras model
model.save('my_stock.h5')  # Save the model as an HDF5 file
```

APPENDIX-B

SNAPSHOTS

app.py - C:\Users\HP\Desktop\srock\app.py (3.6.7)

File Edit Format Run Options Window Help

```
import pandas as pd
import numpy as np
import math
from sklearn.metrics import mean_squared_error, mean_absolute_error, explained_variance_score, r2_score
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import load_model
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, GRU
import matplotlib.pyplot as plt
import yfinance as yf
import streamlit as st

def load_stock_data(stock_ticker):
    end = pd.Timestamp.now()
    start = end - pd.DateOffset(years=1)
    df = yf.download(stock_ticker, start, end)
    return df

def preprocess_data(df):
    df.reset_index(inplace=True)
    df.rename(columns={'Date': 'date', 'Open': 'open', 'High': 'high', 'Low': 'low', 'Close': 'close'}, inplace=True)
    df.set_index('date', inplace=True)
    if not isinstance(df.index, pd.DatetimeIndex):
        df.index = pd.to_datetime(df.index)
    return df

def create_dataset(data, time_step=1):
    dataX, dataY = [], []
    for i in range(len(data)-time_step-1):
        a = data[i:(i+time_step), 0]
        dataX.append(a)
        dataY.append(data[i + time_step, 0])
    return np.array(dataX), np.array(dataY)

def login():
    st.title('Stock Future Predictor with sentimenta analysis')
    stock_ticker = st.text_input('Enter stock Ticker', 'AAPL')
    predict_button = st.button('Predict')

    if predict_button:
        st.write(f"Fetching data for {stock_ticker}...")
        df = load_stock_data(stock_ticker)
        df = preprocess_data(df)
        st.write(df.head())
```

Ln: 1 Col: 0

app.py - C:\Users\HP\Desktop\srock\app.py (3.6.7)

File Edit Format Run Options Window Help

```
    if predict_button:
        st.write(f"Fetching data for {stock_ticker}...")
        df = load_stock_data(stock_ticker)
        df = preprocess_data(df)
        st.write(df.head())
        st.write(df['close'][-1])
        closedf = df[['close']]
        scaler = MinMaxScaler(feature_range=(0, 1))
        closedf = scaler.fit_transform(np.array(closedf).reshape(-1, 1))
        st.subheader('closing Price VS Time Chart ')
        fig = plt.figure(figsize=(10,5))
        plt.style.use('dark_background')
        plt.plot(df.close , color = 'yellow')
        plt.legend()
        st.pyplot(fig)

        training_size = int(len(closedf) * 0.65)
        test_size = len(closedf) - training_size

        train_data, test_data = closedf[0:training_size,:], closedf[training_size:len(closedf),:1]

        time_step = 15
        X_train, y_train = create_dataset(train_data, time_step)
        X_test, y_test = create_dataset(test_data, time_step)

        X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
        X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

        # model = Sequential()
        # model.add(LSTM(32, return_sequences=True, input_shape=(time_step, 1)))
        # model.add(LSTM(32, return_sequences=True))
        # model.add(LSTM(32))
        # model.add(Dense(1))
        # model.compile(loss='mean_squared_error', optimizer='adam')
        # model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=200, batch_size=5, verbose=1)
        model = load_model('my_stock.h5', compile=False)

        train_predict = model.predict(X_train)
        test_predict = model.predict(X_test)

        # Inverse transform predictions
        train_predict = scaler.inverse_transform(train_predict)
        test_predict = scaler.inverse_transform(test_predict)
        y_train = scaler.inverse_transform(y_train.reshape(-1, 1))
        y_test = scaler.inverse_transform(y_test.reshape(-1, 1))
```

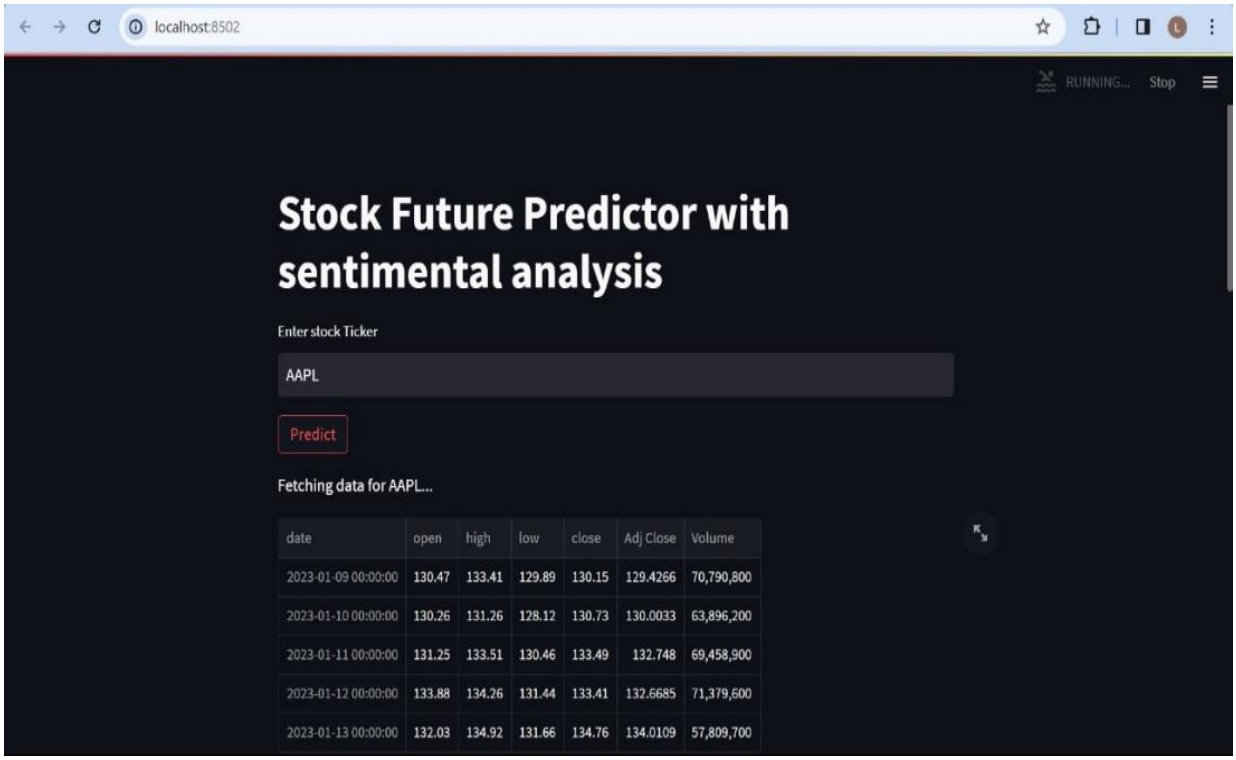
Ln: 83 Col: 7

app.py - C:\Users\HP\Desktop\srock\app.py (3.6.7)

File Edit Format Run Options Window Help

```
        X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

        # model = Sequential()
        # model.add(LSTM(32, return_sequences=True, input_shape=(time_step, 1)))
        # model.add(LSTM(32, return_sequences=True))
        # model.add(LSTM(32))
        # model.add(Dense(1))
        # model.compile(loss='mean_squared_error', optimizer='adam')
        # model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=200, batch_size=5, verbose=1)
        model = load_model('my_stock.h5', compile=False)

        train_predict = model.predict(X_train)
        test_predict = model.predict(X_test)

        # Inverse transform predictions
        train_predict = scaler.inverse_transform(train_predict)
        test_predict = scaler.inverse_transform(test_predict)
        y_train = scaler.inverse_transform(y_train.reshape(-1, 1))
        y_test = scaler.inverse_transform(y_test.reshape(-1, 1))

        # Calculating evaluation metrics
        train_rmse = math.sqrt(mean_squared_error(y_train, train_predict))
        test_rmse = math.sqrt(mean_squared_error(y_test, test_predict))

        st.write(f"positive Ratio With actual Price: {train_rmse}")
        st.write(f"Negative Ratio With actual Price: {test_rmse}")

        # Plotting
        fig, ax = plt.subplots(figsize=(10, 6))
        ax.plot(y_test, label='positive Ratio With actual Price:')
        ax.plot(test_predict, label='Negative Ratio With actual Price')
        ax.set_title('Stock Price Prediction')
        ax.set_xlabel('Time')
        ax.set_ylabel('Stock Price')
        ax.legend()
        st.pyplot(fig)


def main():
    login()

if __name__ == "__main__":
    main()
```

Ln: 111 Col: 0




localhost:8502
2023-01-12 00:00:00 133.88 134.26 131.44 133.41 132.6685 71,379,600
2023-01-13 00:00:00 132.03 134.92 131.66 134.76 134.0109 57,809,700
closing Price VS Time Chart
200
190
180
170
160
150
140
130
2023-01
2023-03
2023-05
2023-07
2023-09
2023-11
2024-01


localhost:8502
positive Ratio With actual Price: 2.073913717599255
Negative Ratio With actual Price: 2.19011956044451
Stock Price Prediction
positive Ratio With actual Price:
Negative Ratio With actual Price
Stock Price
Time

APPENDIX-C

ENCLOSURES

1. Journal Submission Details.

Thanks, We have been received your submitted paper and your paper is in the process of review. It takes 1-2 days and once completed we will notify you the paper's acceptance/rejection decision. In next few hours, You will Receive Email of Paper Receive Notification with IJISRT Paper ID. If email not receive in Inbox, kindly check email in spam folder.

IJISRT Login Details already has been send at your registered Email ID. If you are exisitng user of IJISRT then use your previous Login Details.

Click Here for Login .

(if you did not logged in yet, then check your email id for login credentials)

2. Plagiarism Check report

CSD11

ORIGINALITY REPORT

19%	15%	12%	12%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.researchgate.net Internet Source	1%
2	www.analyticsvidhya.com Internet Source	1%
3	Submitted to Liverpool John Moores University Student Paper	1%
4	www.mdpi.com Internet Source	1%
5	Submitted to Presidency University Student Paper	1%
6	www.geeksforgeeks.org Internet Source	1%
7	Submitted to University of Hong Kong Student Paper	1%
8	www.ijert.org Internet Source	1%
9	Submitted to King's College Student Paper	1%

The Project work carried out here is mapped to SDG-3 SDG-3 Good Health and Well-Being. The project work carried here contributes to the well-being of the human society. This can be used for Predicting the stock prices based on the previous stock history, so that the we can predict and analyze the behaviour of the future stock prices.