**Stock Price Prediction:**

**Predictive Analytics and Visualization of Stock Prices Using Long-Short Term Memory Networks (LSTM)**

By

Ee Wei LI



FACULTY OF COMPUTING AND

INFORMATION TECHNOLOGY

TUNKU ABDUL RAHMAN UNIVERSITY OF MANAGEMENT AND TECHNOLOGY

KUALA LUMPUR

ACADEMIC YEAR

**2024/25**

Stock Price Prediction

By

Ee Wei Li

Supervisor: Dr.Chaw Jun Kit

A project report submitted to the

Faculty of Computing and Information Technology

in partial fulfillment of the requirement for the

Bachelor of Computer Science (Honours)

**Department of Mathematical and Data Science**

Faculty of Computing and Information Technology

Tunku Abdul Rahman University of Management and Technology

Kuala Lumpur

**Copyright by Tunku Abdul Rahman University of Management and Technology.**

All rights reserved. No part of this project documentation may be reproduced, stored in retrieval system, or transmitted in any form or by any means without prior permission of Tunku Abdul Rahman University of Management and Technology.

**Declaration**

The project submitted herewith is a result of my own efforts in totality and in every aspect of the project works. All information that has been obtained from other sources has been fully acknowledged. I understand that any plagiarism, cheating or collusion or any sorts constitutes a breach of TAR University rules and regulations and would be subjected to disciplinary actions.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ee Wei Li

Bachelor of Computer Science (Honours) in Data Science

ID: 22WMR14336

**Abstract**

The project aims to develop a stock price prediction system using Long Short-Term Memory (LSTM) networks to provide accurate and timely forecasts in the financial domain. Stock markets are volatile, so it is hard for investors and analysts to make informed decisions. This prediction system helps users to anticipate stock price trends, reduce investment risks, and improve decision-making.

The project involves collecting, processing, and analyzing historical stock price data using machine learning models such as LSTM networks. It integrates real-time data and offers useful visualizations, such as price trends and comparison graphs to help users. Users can enter specific stock symbols and timeframes to get tailored predictions and set custom alerts for price changes in the Streamlit dashboard which is integrated with the LSTM model. Additional features include role-based authentication, reporting, and real-time notifications, making the system useful for financial analysts, investors, and administrators.

The project develops step by step which includes data collection, data preprocessing, and model implementation. Tools such as Python, TensorFlow, and Pandas are used to build and evaluate the machine learning models. The models are evaluated using performance metrics such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) to ensure they are accurate and reliable.

The system is tested to ensure its performance, reliability, and ability to scale. Both functional and non-functional tests are conducted to ensure that predictions are delivered efficiently and meet the required accuracy standards.

**Acknowledgement**

This contains acknowledgement to those who have contributed directly or indirectly to the completion of the project. Usually the people to be acknowledged include the project supervisor(s), moderators, family, and those who have given assistance and supports to ensure the success of the project.

**Table of Contents**

[**1 Introduction 8**](#_heading=h.26in1rg)

[1.1 Project Objectives 8](#_heading=h.lnxbz9)

[1.1.1 Accurate Forecasting of Stock Prices 8](#_heading=h.35nkun2)

[1.1.2 Enhanced Visualization and Insights 9](#_heading=h.1ksv4uv)

[1.1.3 Informed Decision-Making 9](#_heading=h.2jxsxqh)

[1.1.4 High Accessibility and Scalability 9](#_heading=h.3j2qqm3)

[1.2 Project Background 10](#_heading=h.4i7ojhp)

[1.3 Advantages and Contributions 11](#_heading=h.2xcytpi)

[1.3.1 Prediction Enhancement 11](#_heading=h.1ci93xb)

[1.3.2 Enhanced Data Interpretation 11](#_heading=h.2bn6wsx)

[1.3.3 Decision-Making Support 11](#_heading=h.3as4poj)

[1.3.4 Scalability and Adaptability 12](#_heading=h.49x2ik5)

[1.3.5 Empowering Financial Stakeholders 12](#_heading=h.2p2csry)

[1.4 Project Plan 12](#_heading=h.3o7alnk)

[1.4.1 Data Collection and Preprocessing 12](#_heading=h.23ckvvd)

[1.4.2 Model Training and Testing 12](#_heading=h.32hioqz)

[1.4.3 Dashboard Development 13](#_heading=h.41mghml)

[1.4.4 Implementation and Validation 13](#_heading=h.vx1227)

[1.4.5 Integration with Firebase 13](#_heading=h.1v1yuxt)

[1.4.6 System Deployment 13](#_heading=h.2u6wntf)

[1.4.7 Documentation and Final Reporting 14](#_heading=h.3tbugp1)

[1.5 Project Team and Organization 14](#_heading=h.nmf14n)

[1.6 Chapter Summary and Evaluation 14](#_heading=h.37m2jsg)

[**2 Literature Review 16**](#_heading=h.46r0co2)

[2.1 Stock Price Prediction 16](#_heading=h.2lwamvv)

[2.2 Machine Learning Techniques 16](#_heading=h.111kx3o)

[2.2.1 Traditional Time Series Models 16](#_heading=h.3l18frh)

[2.2.2 Support Vector Machines (SVM) 17](#_heading=h.4k668n3)

[2.2.3 Random Forests 17](#_heading=h.1egqt2p)

[2.2.4 Neural Networks 17](#_heading=h.2dlolyb)

[2.3 Long Short-Term Memory (LSTM) Networks 19](#_heading=h.3vac5uf)

[2.3.1 Overview of Long Short-Term Memory (LSTM) Networks 19](#_heading=h.2afmg28)

[2.3.2 Applications of LSTM in Stock Price Prediction 19](#_heading=h.39kk8xu)

[2.3.3 Advantages and Limitations of LSTM 20](#_heading=h.1opuj5n)

[2.4 Data Visualization in Financial Analytics 23](#_heading=h.48pi1tg)

[2.4.1 Importance of Data Visualization 23](#_heading=h.2nusc19)

[2.4.2 Power BI for Data Visualization 23](#_heading=h.3mzq4wv)

[2.5 Integrating LSTM with Power BI 23](#_heading=h.2250f4o)

[2.6 Comparison of Approaches 24](#_heading=h.haapch)

[2.6.1 Traditional Models vs. Machine Learning Techniques 24](#_heading=h.319y80a)

[2.6.2 Characteristics of Power BI in Enhancing Usability 26](#_heading=h.1gf8i83)

[2.7 Database for Real-time Data 27](#_heading=h.40ew0vw)

[2.7.1 Importance of Database 27](#_heading=h.2fk6b3p)

[2.7.2 Firebase for Storing Real-time Data 27](#_heading=h.3ep43zb)

[2.7.3 Integration of LSTM Model and Power BI with Firebase 28](#_heading=h.1tuee74)

[2.7.4 Advantages of Firebase 28](#_heading=h.4du1wux)

[2.8 Chapter Summary & Evaluation 29](#_heading=h.2szc72q)

[**3 Methodology and Requirements Analysis 32**](#_heading=h.184mhaj)

[3.1 Methodology 32](#_heading=h.3s49zyc)

[3.1.1 Data Collection 33](#_heading=h.279ka65)

[3.1.2 Data Preprocessing 33](#_heading=h.36ei31r)

[3.1.3 Exploratory Data Analysis 33](#_heading=h.45jfvxd)

[3.1.4 Training and Testing Model 33](#_heading=h.zu0gcz)

[3.1.5 Validation and Evaluation 33](#_heading=h.3jtnz0s)

[3.1.6 Integration with Power BI 34](#_heading=h.4iylrwe)

[3.1.7 Data Visualization 34](#_heading=h.1d96cc0)

[3.1.8 Model Deployment 34](#_heading=h.2ce457m)

[3.2 Software Development Model 34](#_heading=h.rjefff)

[3.2.1 Agile Software Development Model 34](#_heading=h.3bj1y38)

[3.2.2 Phases of Agile Software Development Model 37](#_heading=h.2pta16n)

[3.3 Functional and Non-functional Requirements 38](#_heading=h.2hio093)

[3.3.1 Functional Requirements 38](#_heading=h.wnyagw)

[3.3.2 Non-functional Requirements 40](#_heading=h.3gnlt4p)

[3.4 Chapter Summary and Evaluation 41](#_heading=h.1vsw3ci)

[**4 System Design 44**](#_heading=h.4fsjm0b)

[4.1 System Design Architecture 44](#_heading=h.2uxtw84)

[4.2 Role of Long Short-Term Memory Model 44](#_heading=h.1a346fx)

[4.3 Role of Power BI 45](#_heading=h.3u2rp3q)

[4.4 Role of Firebase 45](#_heading=h.2981zbj)

[4.5 System Integration 45](#_heading=h.odc9jc)

[4.5.1 Data Flow 45](#_heading=h.38czs75)

[4.5.2 Prediction Generation 45](#_heading=h.47hxl2r)

[4.5.3 Data Visualization 46](#_heading=h.11si5id)

[4.5.4 Interaction with users 46](#_heading=h.20xfydz)

[4.6 Chapter Summary and Evaluation 46](#_heading=h.302dr9l)

[**5 Implementation and Testing 48**](#_heading=h.1f7o1he)

[5.1 Import Required Libraries 48](#_heading=h.3z7bk57)

[5.2 Load dataset 50](#_heading=h.evnw0dxlhq2w)

[5.3 Exploratory Data Analysis (EDA) 51](#_heading=h.62puyopxp20p)

[Data Preview 51](#_heading=h.62pnnndv8npb)

[Moving Average 56](#_heading=h.11pbttpxiofo)

[Daily Return 57](#_heading=h.n75t406imt7c)

[Correlation Heatmap 58](#_heading=h.7ukiis5jpaxf)

[Summary of Data 59](#_heading=h.dztzj9u5f8wd)

[5.4 Data Preprocessing 60](#_heading=h.7pjnp9fihnes)

[Feature Engineering 60](#_heading=h.q32l0yftotnm)

[Normalization (Using MinMaxScaler) 64](#_heading=h.i009if7ihi19)

[Split Training and Testing Data 64](#_heading=h.wtrluhkosetf)

[5.5 Define LSTM Model 66](#_heading=h.wjd6ki1ksocp)

[5.6 Training Data 66](#_heading=h.uagxyf5hslr4)

[5.7 Hyperparameter Tuning 69](#_heading=h.rr2iuhhkksqh)

[5.8 Testing Data 75](#_heading=h.bjdj5luan5v)

[5.9 Forecast Future Data 78](#_heading=h.unjnmdoqripu)

[5.10 Benchmarks 82](#_heading=h.q9xukvwapxjk)

[ARIMA 83](#_heading=h.lpc7litbz3nt)

[Recurrent Neural Network (RNN) 84](#_heading=h.u0fnqhwzurlc)

[Random Forest 86](#_heading=h.jzxnbj871naa)

[Support Vector Machine 88](#_heading=h.sostakiuhrm3)

[Comparison between Models 89](#_heading=h.gprp8uv105r0)

[5.11 Streamlit (Dashboard) 90](#_heading=h.8t86uuvoa7rw)

[5.12 Test Cases 97](#_heading=h.rr8mmj1xywbs)

[TC01 Stock Selection 97](#_heading=h.7y2uzfxb0tqn)

[TC02 Prediction Generation 97](#_heading=h.iyf6cih5kqo9)

[TC03 Date Filter (Historical) 98](#_heading=h.jsil0q5okbm2)

[TC04 Data Download (Historical) 98](#_heading=h.xne2kt8mntsu)

[TC05 Table Toggle 99](#_heading=h.7u0zv7tlrqsp)

[5.13 Chapter Summary and Evaluation 99](#_heading=h.16x20ju)

[**6 Discussions and Conclusion 102**](#_heading=h.2jh5peh)

[Summary 102](#_heading=h.ymfzma)

[Achievements 102](#_heading=h.3im3ia3)

[Contributions 103](#_heading=h.1xrdshw)

[Limitations and Future Improvements 104](#_heading=h.4hr1b5p)

[Issues and Solutions 106](#_heading=h.2wwbldi)

[Unresolved Issues 107](#_heading=h.z5mm7norstpp)

[**References 108**](#_heading=h.1c1lvlb)

[**Appendices 113**](#_heading=h.3w19e94)

[System Requirements 113](#_heading=h.bliuucjovx9l)

[Step-by-Step Instructions 113](#_heading=h.i1j90c9cgm9l)

[Step 1: Launch the Application 113](#_heading=h.elua29ln5r29)

[Step 2: Choose a Stock 113](#_heading=h.swk6u6z8k9dm)

[Step 3: Set Prediction Range 115](#_heading=h.6t7ttj5a2qfn)

[Step 4: View Historical Data 115](#_heading=h.lp9rmb3yt0cq)

[Step 5: Explore Historical Chart 116](#_heading=h.nb0owuri0wap)

[Step 6: Download Historical Data 117](#_heading=h.nemsot6vav70)

[Step 7: View Predicted Future Prices 117](#_heading=h.ff36okpwxkpr)

[Step 8: View Forecast in Table Format (Optional) 118](#_heading=h.wkawregb908v)

[Note: Real-Time 119](#_heading=h.sithp16lspcu)

[Closing the App 120](#_heading=h.l004b9v07s1o)

Chapter 1

**Introduction**

# Introduction

This chapter outlines my project’s objectives, background, and significance and hence focusing on facing challenges in stock price prediction using LSTM networks and Power BI. It highlights the proposed system’s advantages, and details the project plan. This chapter sets the stage for further discussion in subsequent chapters.

## Project Objectives

The main objective of my project is to develop an effective and efficient stock price prediction system by integrating Long Short-Term Memory (LSTM) networks with Power BI. Long Short-Term Memory (LSTM) networks is one of the children of Recurrent Neural Network (RNN**)** as both of their formulas are omitted together for machine learning and Power BI for interactive dashboard and data visualization (Alex Sherstinky, 2020). The system which I plan to develop concentrates on main challenges in the financial market nowadays. For example, volatility, non-linear patterns, and the demands for decision-making tools with higher accuracy. My project aims to integrate the predictive power of machine learning with user-friendly insights and interface by using data visualization. The objectives of my project will eventually increase the capabilities of the stakeholders in their financial decision-making processes in their future.

### Accurate Forecasting of Stock Prices

One of the objectives of my project is to develop an accurate stock price prediction system. It is essential for informed financial decision-making. Long Short-Term Memory (LSTM) networks are suitable for modelling stock price movements by capturing complex patterns and long-term dependencies in historical data. LSTMs use memory cells and gating mechanisms to obtain relevant information instead of using traditional methods that rely on linear assumptions. Moreover, they are robust to noise. This makes them effectively identify both short-term fluctuations and long-term trends.

Long Short-Term Memory (LSTM) networks can provide robust and reliable predictions in volatile markets to overcome challenges such as the vanishing gradient problem (Qiu et al., 2020). The ability of Long Short-Term Memory (LSTM) networks to process large volumes of historical data allows them to adapt to dynamic market conditions anytime, and eventually turning them into an ideal tool for forecasting. Hence, my project also aims to deliver precise and accurate predictions provided by Long Short-Term Memory (LSTM) networks advanced capabilities to enable stakeholders have the confidence to make informed decision making in investments.

### Enhanced Visualization and Insights

The ability of Streamlit to visualize data effectively is also an important part of financial analysis as it transforms complex information into useful insights. My project aims to develop an interactive and intuitive dashboard using Streamlit, which is a leading business intelligence tool known for its advanced visualization capabilities. My interactive dashboard will visualize historical data in user-friendly formats such as graphs, tables, and heatmaps. Therefore, this can enable users to easily identify trends and patterns of stock price anytime. Additionally, it integrates with real-time updates, so that stakeholders can gain future predictions dynamically. This feature ensures that stakeholders can make informed decisions properly based on the latest data gathered. Streamlit dashboard bridges the gap between advanced predictive models and practical usability by simplifying the analysis of complex stock market data, which makes it caters to the needs of both novice and experienced users.

### Informed Decision-Making

Besides, my project aims to provide investors, portfolio managers, and financial analysts reliable tools to produce data-driven and informed decisions. Accurate stock price predictions and interactive visualizations enable stakeholders to properly assess future risks and their opportunities effectively and efficiently. This capability is particularly valuable in volatile markets, where timely and informed decisions can significantly affect investment outcomes. The developed system in my project will minimize risks and optimize their investment strategies by providing useful insights into historical trends and future projections. Furthermore, the integration of predictive analytics and intuitive visualization helps the stakeholders to reduce cognitive load, and therefore enables users to focus on strategic decision-making rather than data interpretation.

### High Accessibility and Scalability

Lastly, both scalability and accessibility of my system are also the one of the main objectives of my project. The system is designed to be easily adapted to various datasets and stock markets. This ensures its broad applicability across diverse financial contexts. My system’s modular framework supports customization to meet specific user needs, whether it is used for analyzing regional stock exchanges or global markets. Additionally, the accessibility of my system allows both institutional and retail investors to benefit from advanced predictive analytics, and democratizing access to tools that were previously limited to large organizations. This ensures my system’s relevance and usability for a wide audience, fostering greater participation and equity in financial decision-making processes.

## Project Background

Stock price prediction is an important but also a challenging task because of its complexity and volatility of financial markets. This is because market behaviour can be influenced by various factors. For example, these influences include investor sentiment, geopolitical events, corporate performance, and macroeconomic indicators and so on. Traditional methods such as moving averages, linear regression, and autoregressive models and so on often fail to capture these intricate patterns and dependencies.

My project will implement Long Short-Term Memory (LSTM) networks to overcome the limitations that will be encountered by users when forecasting stock prices. It is a type of recurrent neural network (RNN), which is known for designing to manage long-term dependencies in time-series data. Long Short-Term Memory (LSTM) networks overcome issues such as the vanishing gradient problem (Alex Sherstinky, 2020). This enables it to retain critical information over longer periods and provide superior predictive accuracy.

Moreover, poor visualization will limit the usability of the results. Statistical financial reports and conventional charts lack interactivity. This makes it difficult for users to derive meaningful insights. Streamlit will be used in my project to present predictions gained and historical data in an interactive and user-friendly format so that the issues caused by poor visualization can be solved. This also ensures that stakeholders can analyse data, make comparisons, and derive actionable insights in an interactive environment.

By integrating Long Short-Term Memory (LSTM) networks’ advanced forecasting capabilities with Power BI’s dynamic visualization tools, my proposed system provides a comprehensive solution to solve the challenges and issues of stock price prediction and analysis.

## Advantages and Contributions

There are several advantages provided by the stock price prediction system to the stakeholders. The proposed system contributes to the investment field by providing informed decisions and useful insights to stakeholders in an interactive way.

### Prediction Enhancement

The proposed system benefits the stakeholder by providing enhanced predictive accuracy by leveraging the advanced capabilities of LSTM networks. Long Short-Term Memory (LSTM) networks are particularly good at capturing long-term dependencies and non-linear relationships in stock market data, which are often missed by traditional models (Widodo Budiharto, 2021). By solving critical issues like the vanishing gradient problem, Long Short-Term Memory (LSTM) networks ensure reliable and accurate forecasts by solving critical issues such as vanishing gradient problem, even if it is used in volatile and unpredictable financial markets nowadays. This precision enables stakeholders to trust the proposed system to come out with their informed financial decisions. This makes it a significant improvement over conventional prediction techniques.

### Enhanced Data Interpretation

In terms of data interpretation, the integration of Power BI dashboards allows users to visualize complex financial data in an intuitive and interactive manner. These dashboards simplify the process of analyzing trends, comparing results, and deriving actionable conclusions from predictive analytics (Gurpreet Singh et al., 2023). The ability to explore data dynamically ensures that users can identify patterns and insights that might otherwise go unnoticed, thus enhancing the utility of the predictive system.

### Decision-Making Support

The system benefits users by providing robust support for decision-making by integrating accurate predictions from Long Short-Term Memory (LSTM) networks along with clear and interactive visualizations from Power BI. This empowers stakeholders, including financial analysts and investors, to make better informed decisions while minimizing risks and enhancing investment strategies at the same time. The proposed system helps the stakeholders in optimizing financial outcomes by presenting useful insights clearly and concisely.

### Scalability and Adaptability

Scalability and adaptability are the advantages of the proposed system when compared with other systems. It is designed to accommodate various datasets and markets, so that it can be versatile for different financial environments. The proposed system's modular framework ensures that it can be tailored to specific needs without significant reconfiguration whether applied to regional stock exchanges or global financial markets (Akhavanpour et al., 2008). This adaptability enables it to act as a valuable and useful tool across diverse contexts.

### Empowering Financial Stakeholders

Lastly, the proposed system provides advantages by playing a crucial role in empowering financial stakeholders by democratizing access to advanced predictive tools. The project promotes financial literacy and technological innovation by making sophisticated analytics and interactive visualizations available to both institutional and retail investors. This contribution extends beyond individual users, and therefore supports the broader financial ecosystem by fostering informed, data-driven decision-making and encouraging more equitable participation in financial markets.

## Project Plan

This section includes phases of developing the proposed system in brief.

### Data Collection and Preprocessing

The proposed project will begin with data collection and preprocessing. The historical stock price dataset that will be used to train the Long Short-Term Memory (LSTM) model originated from reliable platforms such as Kaggle. The dataset will be cleaned and preprocessed to eliminate inconsistencies, noisy data, missing values, outliers and so on. This ensures that it is ready to be trained using the Long Short-Term Memory (LSTM) model. This step is crucial to ensure the reliability and accuracy of subsequent predictive analysis.

### Model Training and Testing

The next phase involves model training and testing, where the Long Short-Term Memory (LSTM) model will be trained on the preprocessed data to learn about the patterns and dependencies in stock prices. Various performance metrics, such as Mean Squared Error (MSE) and Mean Absolute Percentage Error (MAPE), will be used to evaluate the model’s accuracy and effectiveness. This step will also involve fine-tuning the model’s hyperparameters to optimize its performance for different financial datasets.

### Dashboard Development

Next, the dashboard development will use Streamlit to create an interactive and user-friendly visualization platform. The dashboard will display historical data, predictive trends, and useful insights by using graphs, tables, charts and so on to represent the data obtained in different ways. The feature is focused on enabling users to easily interpret the model's predictions and make informed decisions based on accurate and precise data representations.

### Implementation and Validation

The implementation and validation phase will test the system in real-world scenarios to ensure its reliability and usability. Feedback from stakeholders will be actively sought and incorporated to improve both the predictive model and the Streamlit dashboard. This iterative process ensures that the final product meets the practical needs of its users while maintaining high standards of accuracy and functionality. After that, we will deploy the model which consists of integration of the Long Short-Term Memory (LSTM) model and Streamlit dashboard before integrating them with firebase.

### System Deployment

After the implementation and validation phase, the fully developed system will be deployed for end-users. This phase will involve integrating the system into a production environment by ensuring seamless functionality and accessibility. Documentation and user guides will be provided to facilitate adoption, and support mechanisms will be established to address any operational challenges post-deployment.

### Documentation and Final Reporting

The final phase of the proposed project involves documentation and reporting. All methodologies, outcomes, and performance metrics will be documented to provide a comprehensive account of the project’s development and results. This phase will end in the submission of a detailed final report of the proposed project, which will serve as a reference for future improvements and applications of the system.

## Project Team and Organization

My project mentioned above will be carried out under the guidance of a project supervisor. The Project Leader, Ee Wei Li, will be responsible for overall project management, including the development of the Long Short-Term Memory (LSTM)model, integration with Power BI, and system implementation. My Supervisor, Dr.Chaw Jun Kit, will provide technical guidance and ensure that the project aligns with its objectives. Regular progress checkings with the supervisor will be conducted. Tools such as Gantt charts will be used to monitor progress and ensure the project stays on track for completion on punctual time.

## Chapter Summary and Evaluation

In conclusion, this chapter outlined the overview of the project by including the objectives, background, advantages, and project planning. The proposed project aims to provide a robust solution for financial stakeholders by overcoming key challenges in stock price prediction such as prediction accuracy, long-term dependency management, data visualization and so on.

The system aims to enhance decision-making, reduce risks, and maximize investment returns with the innovative use of Long Short-Term Memory (LSTM)networks and Power BI. The structured project plan will ensure the successful completion of this initiative, and eventually provide a valuable contribution to the financial analytics field.

Chapter 2

**Literature Review**

# Literature Review

Chapter 2 outlines the existing literature review on stock price prediction and data visualization. This chapter focuses on traditional methods, machine learning techniques, and the advantages of Long Short-Term Memory (LSTM)networks. In this chapter, we can also explore the characteristics of tools such as Streamlit in enhancing data interpretation and understand the integration of Long Short-Term Memory (LSTM)networks with Streamlit. The chapter also includes comparison of approaches and an evaluation of their strengths and limitations.

## Stock Price Prediction

Stock price prediction acts as a crucial role in informed financial decision-making by providing aids to users such as investors, portfolio managers, financial analysts and so on to minimize risks and maximize their profits in investments. Traditional prediction methods such as ARIMA model and exponential smoothing are well-known because of their simplicity and well-established mathematical foundations. These models are effective for stationary time-series data and identifying trends and seasonality (G.Peter Zhang, 2003). However, they failed to account for the highly dynamic, non-linear, and multi-dimensional nature of financial data. As financial markets evolve, traditional methods often lag in capturing the complex interdependencies between various influencing factors such as market sentiment, geopolitical events, macroeconomic indicators and so on (Elsaraiti & Merabet, 2021). This limitation has led to the development of new machine learning (ML) techniques in this generation. Hence, they are capable of offering greater prediction accuracy by using large datasets and discovering hidden patterns in data.

## Machine Learning Techniques

This section will describe various kinds of machine learning techniques, including their limitations and advantages.

### Traditional Time Series Models

ARIMA model has been a foundation to do time-series forecasting by offering simplicity and accuracy for short-term, stationary data. It uses differences to make data stationary, which is then modeled with autoregressive and moving average components (Ruan Luzia et al., 2023). While effective for trend detection, ARIMA’s linear nature limits its ability to capture the non-linear relationships present in stock prices. Additionally, ARIMA struggles with high volatility and often fails to adapt to rapid market changes, making it insufficient for the dynamic nature of financial markets (Raydonal Ospina et al., 2023).

### Support Vector Machines (SVM)

Support Vector Machines (SVM) are powerful machine learning models which are capable of handling non-linear relationships by transforming data into higher-dimensional spaces by using kernel functions (Syed Fawad Hussain, 2019). In the context of stock price prediction, Support Vector Machines (SVM) have been used to classify and predict market movements with reasonable success. One of their advantages is their ability to handle small to medium-sized datasets effectively. However, Support Vector Machines (SVM) models are computationally expensive when applied to large financial datasets and require extensive feature engineering to extract relevant patterns. This makes them less practical for real-time or large-scale applications.

### Random Forests

Random Forests are ensemble learning methods that create multiple decision trees and aggregate their outputs for robust predictions. These models are good at handling noisy, high-dimensional data. This makes them suitable for financial applications. However, they lack the temporal awareness required for time-series data, as each tree operates independently without accounting for sequential dependencies (Sruthi, 2024). As a result, Random Forests are often outperformed by sequential models like Long Short-Term Memory (LSTM) networks in tasks that involve trend prediction or time-sensitive analysis.

### Neural Networks

Neural networks, especially recurrent neural networks (RNNs), have significantly advanced the field of stock price prediction by addressing sequential dependencies in time-series data. However, traditional recurrent neural networks (RNNs) encounter the vanishing gradient problem, and eventually limit their ability to learn long-term dependencies (Dr Barak Or, 2020. Long Short-Term Memory (LSTM) networks overcome this limitation, making them a preferred choice for modelling complex, non-linear relationships in financial markets. Their architecture which is designed with memory cells and gates allows for effective handling of long-term dependencies and temporal patterns.

Table 2.1 Summarization of Machine Learning Techniques

| Model | Description | Advantages | Limitations |
| --- | --- | --- | --- |
| ARIMA | | A traditional time-series model that uses differencing to make data stationary, then applies autoregressive and moving average components. | | --- | | - Simplicity - Effective for short-term, stationary data | | - Assumes linear relationships - Struggles with high volatility - Limited adaptability to rapid market changes | | --- | |
| | Support Vector Machines (SVM) | | --- |  |  | | --- | | Machine learning models that handle non-linear relationships by transforming data into higher-dimensional spaces using kernel functions. | - Capable of handling non-linear relationships - Effective with small to medium-sized datasets | - Computationally expensive with large datasets - Requires extensive feature engineering - Less practical for real-time or large-scale applications |
| | Random Forests | | --- |  |  | | --- | | Ensemble learning methods that create multiple decision trees and aggregate their outputs for robust predictions. | - Handles noisy, high-dimensional data well - Suitable for financial applications | - Lacks temporal awareness for time-series data - Each tree operates independently without accounting for sequential dependencies - Often outperformed by sequential models like LSTM in trend prediction tasks |
| | Neural Networks (RNNs) | | --- |  |  | | --- | | Models that address sequential dependencies in time-series data; however, traditional RNNs encounter the vanishing gradient problem, limiting their ability to learn long-term dependencies. | - Capable of modeling complex, non-linear relationships - Suitable for sequential data | - Traditional RNNs suffer from vanishing gradient problem - Limited in learning long-term dependencies |
| | LSTM Networks | | --- |  |  | | --- | | An advanced type of RNN designed with memory cells and gates to handle long-term dependencies and temporal patterns effectively. | An advanced type of RNN designed with memory cells and gates to handle long-term dependencies and temporal patterns effectively. | - Computationally intensive - Requires large datasets for training - May overfit if not properly regularized |

## Long Short-Term Memory (LSTM) Networks

In this section, we will discuss more about the Long Short-Term Memory (LSTM) networks, which is the main machine learning model and Recurrent Neural Network we are going to implement in this project.

### Overview of Long Short-Term Memory (LSTM) Networks

Long Short-Term Memory (LSTM) networks are an evolution of RNNs designed to address the challenges of learning long-term dependencies in sequential data. Their architecture includes memory cells and three gating mechanisms, which are input, forget, and output gates which are used to control the flow of information (Anishnama, 2023). This design allows Long Short-Term Memory (LSTM) networks to retain important information over extended sequences while discarding irrelevant data. As a result, they are highly effective for time-series forecasting tasks, such as stock price prediction, where temporal dependencies are crucial.

### Applications of LSTM in Stock Price Prediction

Long Short-Term Memory (LSTM) networks have been proven to be effective in stock price prediction due to their unique capability to capture both short-term fluctuations and long-term trends in financial data. This capability is important in a volatile stock market, where prices are influenced by both present events and historical patterns. Unlike traditional methods such as ARIMA, which rely on linear assumptions and struggle with non-linear relationships, Long Short-Term Memory (LSTM) networks are good at identifying complex interactions in time-series data and providing accurate stock price prediction.

First of all, the capability of Long Short-Term Memory (LSTM) networks in predicting stock price movements compared to traditional models like ARIMA has been showcased by Fischer and Krauss (2018). Their study stated that Long Short-Term Memory (LSTM) networks not only improved prediction accuracy but improve the capturing of the dependencies between sequential data points and highlighting the model’s robustness in time-series forecasting (Fischer & Krauss , 2018). This performance shows the ability of Long Short-Term Memory (LSTM) networks to transform traditional financial prediction by overcoming the limitations of earlier approaches.

Besides, Long Short-Term Memory (LSTM) networks was integrated with sentiment analysis, incorporating news data and social media sentiment to enhance forecasting accuracy (Susrama et al., 2024). Long Short-Term Memory (LSTM) networks could adapt to multi-dimensional data sources and provide more informed predictions by combining structured financial data with unstructured text-based data (Kai Liu & Jie Zhang, 2021). This integration highlights the flexibility of LSTM networks in handling diverse datasets, making them highly effective in capturing market sentiment and its impact on stock prices.

Additionally, Long Short-Term Memory (LSTM) networks have been employed in hybrid models that combine multiple data sources, such as macroeconomic indicators, trading volumes, and corporate performance metrics (Liwen Xu et al., 2018). These applications show the Long Short-Term Memory (LSTM) networks’ adaptability in learning from many kinds of input types to produce accurate predictions. The ability to integrate and process various dimensions of data makes Long Short-Term Memory (LSTM) an indispensable tool in financial analytics, particularly for applications that require deep insights into both quantitative trends and qualitative market influences.

In conclusion, Long Short-Term Memory (LSTM) networks are extensively applied in stock price prediction by offering unmatched accuracy and flexibility. Through their ability to process sequential data, integrate diverse information sources, and adapt to dynamic market conditions, Long Short-Term Memory (LSTM) networks have established themselves as a cornerstone of modern financial forecasting (Ivan Malashin et al., 2024). These applications underscore their pivotal role in driving data-driven decision-making in the financial sector.

### Advantages and Limitations of LSTM

One of the advantages of Long Short-Term Memory (LSTM) networks is their unique architecture, which is different from other machine models and allows them to model long-term dependencies in sequential data. Unlike traditional models that keep struggling with capturing temporal patterns over extended time frames, Long Short-Term Memory (LSTM) networks are designed with memory cells and gating mechanisms that retain relevant information while discarding noise (Mayank Banoula, 2023). The ability to manage long-term dependencies is essential in stock price prediction as historical data and trends often influence future movements. Long Short-Term Memory (LSTM) networks are also good at handling non-linear relationships. This enables them to model the complex interactions between various market factors that drive stock price fluctuations (Ivan Malashin et al., 2024). These features make Long Short-Term Memory (LSTM) networks one of the most accurate and reliable tools for time-series forecasting tasks.

Besides, Long Short-Term Memory (LSTM) networks are robust against common challenges which are faced by other machine learning models, such as the vanishing gradient problem. Traditional methods such as Recurrent Neural Networks (RNNs) often fail to learn from long sequences due to diminishing gradient values during backpropagation (Niklas Donges, 2024). Long Short-Term Memory (LSTM) networks overcome this challenge through their gating mechanisms by ensuring that essential information is preserved and used effectively during training. This ability improves their operational performance in tasks which requires detailed analysis of sequential data. Therefore, they are indispensable for financial analytics and other time-series applications.

However, despite their strengths, Long Short-Term Memory (LSTM) networks also have certain limitations. They require significant computational resources due to their complex architecture and iterative training process. The training phase often involves processing large datasets and multiple iterations to achieve optimal performance, which can be time-intensive and costly ([Prudhviraju Srivatsavaya](https://medium.com/@prudhviraju.srivatsavaya?source=post_page---byline--914a96fa0acb--------------------------------), 2023). Furthermore, the performance of Long Short-Term Memory (LSTM) networks is highly dependent on hyperparameter tuning. Parameters such as learning rate, number of layers, and hidden units must be carefully optimized to prevent underfitting or overfitting, which requires expertise and experimentation ([Prudhviraju Srivatsavaya](https://medium.com/@prudhviraju.srivatsavaya?source=post_page---byline--914a96fa0acb--------------------------------), 2023) .

Moreover, Long Short-Term Memory (LSTM) is limited by their potential over-dependence on historical data. Although Long Short-Term Memory (LSTM) networks are good at extracting patterns from historical information, their predictions may be less effective in scenarios where sudden market changes occur due to unforeseen events, such as geopolitical crises and natural disasters. This makes it essential to train Long Short-Term Memory (LSTM) models with external data sources, such as news sentiment analysis and macroeconomic indicators to increase their accuracy.

In conclusion, the advantages of Long Short-Term Memory (LSTM) networks are more than their limitations. Their ability to handle sequential data with precision and adaptability to non-linear relationships makes them a valuable tool in stock price forecasting. Long Short-Term Memory (LSTM) networks can deliver accurate predictions that empower stakeholders to make informed and strategic decisions in complex and volatile markets if proper tuning is applied.

Table 2.2 Advantages and Limitations of LSTM

| **Advantages** | **Limitations** |
| --- | --- |
| Modeling Long-Term Dependencies | | Computational Complexity | | --- |  |  | | --- | |
| Handling Non-Linear Relationships | | Sensitivity to Hyperparameters | | --- |  |  | | --- | |
| Robustness Against Vanishing Gradient Problem | | Dependence on Historical Data | | --- |  |  | | --- | |

## Data Visualization in Financial Analytics

Next, we are going to look into more information about data visualization in the proposed project. Data visualization is important for stakeholders as it provides useful insights and information for stakeholders to make informed decisions in investment.

### Importance of Data Visualization

Data visualization is important for analyzing predictive insights and presenting them in various formats. Traditional tools for visualization such as static graphs and spreadsheets often fail to provide the interactivity and real-time updates needed for modern financial analysis. Effective visualization simplifies complex data. This enables stakeholders to quickly identify patterns, trends, and anomalies. This is particularly important in stock price prediction, where understanding the data is as crucial as generating accurate forecasts.

### Streamlit for Data Visualization

Streamlit has emerged as a leading tool for financial data visualization due to its ability to create interactive dashboards, integrate with multiple data sources, and present complex datasets in intuitive formats. It supports advanced visualizations such as time-series charts, heatmaps, and KPIs, which are essential for analyzing stock market trends (Microsoft, 2024). Furthermore, its real-time data updating capabilities allow users to monitor live market movements, making it an invaluable tool for financial decision-making.

## Integrating LSTM with Streamlit

The integration of Long Short-Term Memory (LSTM) networks with Streamlit represents a powerful synergy between predictive analytics and intuitive data visualization. LSTM networks have the ability to capture long-term dependencies and model non-linear relationships to generate highly accurate forecasts based on historical data (Muhammad Waqas et al, 2024). These forecasts include trends, patterns, and potential future movements in stock prices, providing critical insights for decision-makers. However, the complexity of these predictive models often makes it challenging for end-users to interpret and act on the results without a clear and accessible presentation format.

Streamlit bridges this gap by translating the raw outputs of the Long Short-Term Memory (LSTM) model into interactive and visually appealing dashboards (DataThick, 2024). Streamlit allows users to explore data through graphs, charts, tables, and heatmaps. This makes it easier to identify trends, correlations, and anomalies (Microsoft, 2024). Real-time data integration ensures that stakeholders can view predictions continuously with historical data. This enables them to compare historical data with current predictions effectively. Additionally, Streamlit's interactive dashboard functionality allows users to tailor their view to focus on specific datasets or metrics. This enhances the usability of the system for diverse financial needs.

This integration also facilitates informed decision-making by presenting predictive insights in a format that is both actionable and accessible. Stakeholders can interact with the data dynamically, filter information based on specific criteria, and generate reports that highlight key findings (maggiesMSFT, 2023). This enables investors, financial analysts, and portfolio managers to make informed decisions with greater confidence. This also reduces risks and maximizes investment opportunities. Moreover, the combination of Long Short-Term Memory (LSTM) networks’ robust forecasting capabilities with Streamlit’s user-friendly interface allows access to advanced analytics to ensure that non-technical users can also benefit from the system’s predictive power.

By combining the strengths of Long Short-Term Memory (LSTM) networks and Streamlit, this integration delivers a comprehensive solution that overcomes the challenges of financial forecasting. It not only enhances the accuracy and reliability of predictions but also ensures that these insights are effectively communicated to stakeholders to assist them in making informed decisions.

## Comparison of Approaches

This section discusses the comparison of various types of machine learning models. This includes advantages and limitations of each model.

### Traditional Models vs. Machine Learning Techniques

There are many approaches that have been used for stock price prediction in the proposed project, each with its strengths and limitations. Traditional methods such as ARIMA are well-known for their simplicity and ease of interpretation. ARIMA works well with stationary data and is often a go-to model for forecasting in the finance field (Zaina Saadeddin, 2024). However, its over-dependence on linear assumptions limits its ability in capturing the non-linear and dynamic nature of financial markets. Moreover, ARIMA struggles with high-dimensional datasets and fails to account for long-term dependencies so it is less suitable for complex time-series data (Karthic Sundaram, 2024).

Besides, machine learning techniques such as Support Vector Machines (SVM) have proven effective in handling non-linear relationships. Support Vector Machines (SVM) are suitable for classification and regression tasks and can model complex data patterns (Yasar & Tabsharani, 2024). However, the computational complexity of Support Vector Machines (SVM) increases with size of datasets and its performance is highly dependent on proper feature selection and preprocessing, which can be labor-intensive and time-consuming (Jair Cervantes, 2020).

Tree-based models such as Random Forests offer robustness and can handle noisy data effectively (EDC Paris Business School, 2024). Random Forests reduce overfitting and produce more stable predictions by combining multiple decision trees. However, these models are less effective when applied to sequential or time-dependent data as they lack mechanisms to capture temporal patterns.

Table 2.3 Comparison of Approaches

| Approaches | Strengths | Weaknesses |
| --- | --- | --- |
| ARIMA | | Simple and interpretable | | --- |  |  | | --- | | | Fails with non-linear, non-stationary data | | --- |  |  | | --- | |
| SVM | | Handles non-linear data effectively | | --- |  |  | | --- | | | Computationally expensive | | --- |  |  | | --- | |
| | Random Forests | | --- |  |  | | --- | | | Robust against overfitting | | --- |  |  | | --- | | | Ineffective for sequential data | | --- |  |  | | --- | |
| LSTM | | Handles long-term dependencies and volatility | | --- |  |  | | --- | | | Computationally intensive | | --- |  |  | | --- | |
| | Power BI | | --- |  |  | | --- | | | Interactive and user-friendly visualization | | --- |  |  | | --- | | Limited built-in ML support |

### Characteristics of Streamlit in Enhancing Usability

Data visualization tools such Streamlit are essential for interpreting and presenting the visualizations of predictive analytic. Streamlit transforms raw data and model predictions into visually appealing, interactive dashboards, making complex information easier to understand and actionable for users (DataThick, 2024). Its ability to present data through a variety of visual formats such as line graphs, bar charts, heatmaps, and KPI indicators ensures that users can quickly identify trends, compare historical data with current predictions, uncover patterns and so on (Microsoft, 2024).

One of Streamlit’s significant advantages is its interactivity. Users can filter data, drill down into specifics, and generate reports tailored to their unique requirements, facilitating a more personalized analysis experience (DataThick, 2024). These features enable faster and more informed decision-making as stakeholders can focus on useful insights without wasting time on unnecessary details. This provides advantages in the financial sector because decision-makers often need to respond quickly to volatile market conditions.

Moreover, Streamlit’s ability to integrate seamlessly with external machine learning models, including Python or R programming enhances its utility (Deepthy A, 2024). Therefore, Power BI can display predictions generated by advanced algorithms such as Long Short-Term Memory (LSTM) networks. This integration enables users to use the power of machine learning while benefiting from Streamlit’s intuitive visualization capabilities at the same time. However, Streamlit itself does not possess built-in predictive analytics features, which requires reliance on external tools for generating predictions.

Streamlit’s scalability and cloud-based functionality are also advantages of it. It supports a wide range of data sources, including databases, APIs, and real-time streams (Microsoft, 2024). This allows users to analyze various datasets in a single platform. Moreover, its suitability with mobile devices ensures that stakeholders can access dashboards anytime and anywhere. Therefore, this shows the flexibility and accessibility of the Streamlit dashboard.

In conclusion, Streamlit plays a role in enhancing the usability of predictive systems. Power BI empowers users to interpret complex analytics effectively by providing dynamic visualizations, interactivity and integration with machine learning models. Hence, this leads to smarter and more informed decisions in time-sensitive scenarios.

## Chapter Summary & Evaluation

Chapter 2 has outlined traditional and machine learning-based approaches for stock price prediction by highlighting each of their strengths and limitations. Long Short-Term Memory (LSTM)networks is a superior method for handling sequential data and capturing long-term dependencies. Additionally, the importance of data visualization was discussed in this chapter. Streamlit is a useful and flexible tool for providing predictive insights. The integration of Long Short-Term Memory (LSTM)networks with Streamlit is a perfect solution by combining robust prediction with interactive visualization to provide modern financial analytics.

Chapter 3

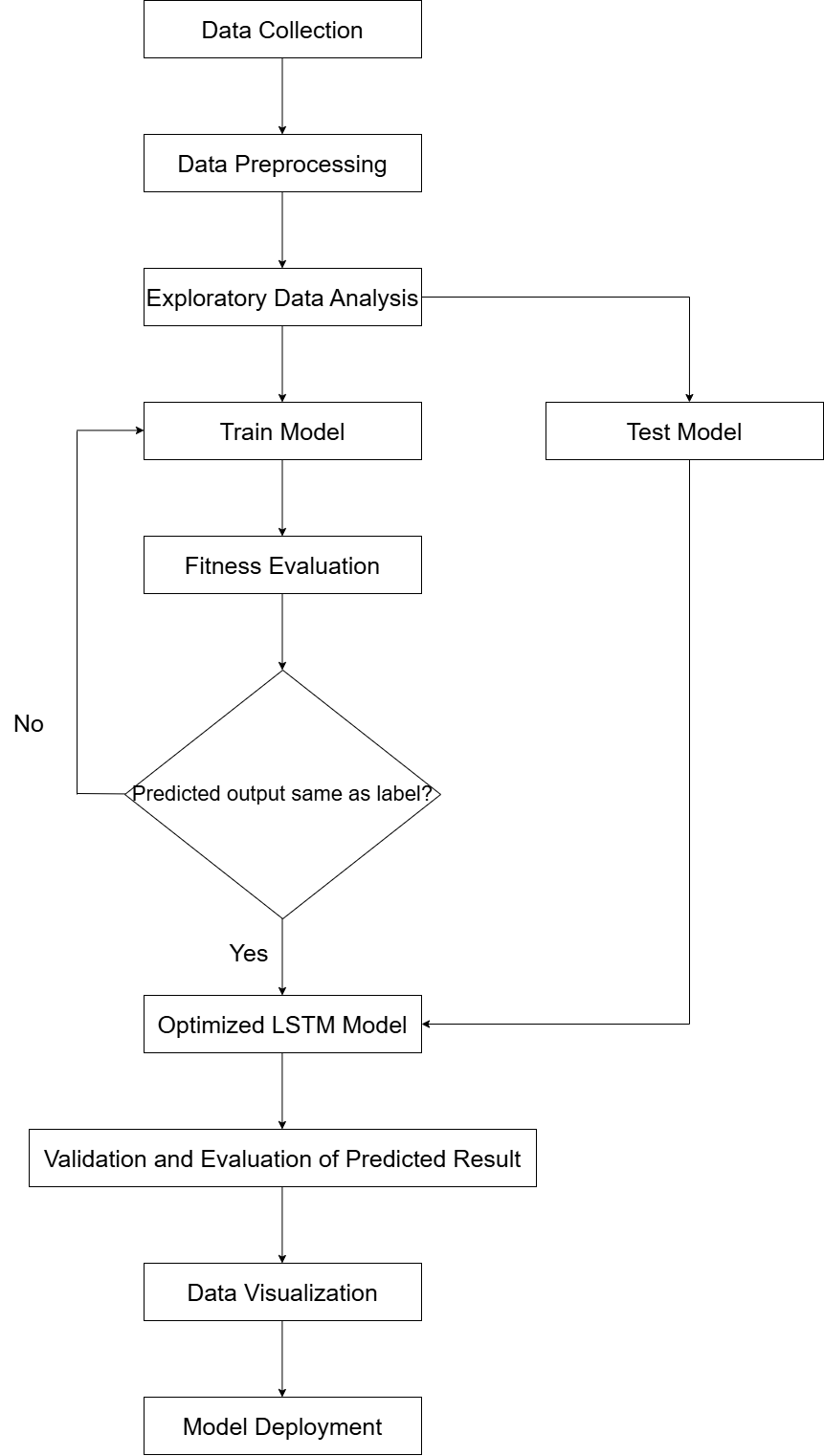
**Methodology and Requirements Analysis**

# Methodology and Requirements Analysis

In this chapter, we will introduce the methodology and the phases for developing the initial model with integration of LSTM networks and Power BI dashboard. We will also discuss the software development model used in the proposed project.

## Methodology

Figure 3.1 Flowchart of model development



### Data Collection

Data Collection is the first phase for developing the initial model. This phase involves the selection of suitable historical stock price data to train the LSTM model. In the proposed project, the dataset obtained from Kaggle, which is titled "World Stock Prices (Daily Updating)," from website URL: <https://www.kaggle.com/datasets/nelgiriyewithana/world-stock-prices-daily-updating>, provides a record of daily stock prices for prominent global brands until today. This ensures that the model has access to up-to-date and diverse data.

### Data Preprocessing

The second phase of developing the mode is data preprocessing. This phase is used to prepare the acquired dataset for training. This phase includes data cleaning to handle missing values and outliers, normalization to scale features appropriately, feature engineering to create relevant indicators, sequence creation to structure the data into formats suitable for LSTM input and so on. Data is preprocessed so that it is easy to be trained and tested in the next few phases.

### Exploratory Data Analysis

Next, the Exploratory Data Analysis (EDA) phase aims to discover patterns, trends, and relationships which lie within the features of data. Through statistical analysis and visualization techniques, such as plotting time series graphs and correlation matrices, insights into seasonal patterns and long-term trends are identified, which are crucial for informed modeling decisions.

### Training and Testing Model

In the Training and Testing Model phase, the preprocessed data is divided into training and testing sets by using a suitable ratio. The LSTM network is then designed and trained on the training data with hyperparameter tuning performed to optimize performance. This process enables the model to learn temporal dependencies and patterns inherent in stock price movements.

### Validation and Evaluation

Validation and Evaluation are critical to assess the model's predictive accuracy. Performance metrics such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) are calculated, and cross-validation techniques are employed to ensure robustness. Residual analysis is also performed to detect any systematic prediction errors.

### Integration with Streamlit

Moreover, integration of the trained and tested LSTM model with Streamlit occurs in this phase. This involves importing the model's predictions ability into Streamlit and developing custom visuals to represent predictive insights about stock price prediction effectively for stakeholders. For example, interactivity features, such as filtering and drill-downs, are enabled to allow stakeholders to explore stock price data dynamically.

### Data Visualization

Additionally, data visualization involves creating interactive dashboards in Streamlit. These dashboards visualize features such as predicted versus actual stock prices, facilitate comparative analysis across different stocks or time periods, highlight significant anomalies in stock price movements and so on. Therefore, this offers stakeholders more understanding about the stock price data shown in the dashboard and informed decision-making.

### Model Deployment

Lastly, the deployment phase enables the predictive system to be accessible to end-users. This includes hosting the integrated model on a reliable server or cloud platform by developing APIs for seamless integration with other applications, and implementing monitoring and maintenance protocols to ensure sustained accuracy and performance over time. The initial model which includes integration of the LSTM model and Streamlit dashboard is ready to go live in a production environment.

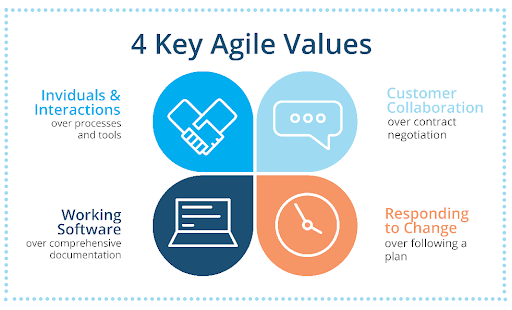
## Software Development Model

In this section, we will discuss the software development model which is implemented in the proposed project to ensure the completeness of features for the stock price prediction system.

### Agile Software Development Model

Agile is a type of software development process that applies a certain amount of pragmatism to the final product delivery and foresees the requirement for flexibility. The Agile software development model focuses on delivery of discrete components rather than the complete application, so it often requires a cultural transformation in many organizations (Scott Robinson, 2024). Agile software development is a suitable model for the proposed project, which is developing stock price prediction because of its iterative and flexible approach. Agile software development model is useful as it enables continuous refinement, incorporates regular feedback from stakeholders, and adapts to developing project requirements, which are common in stock price prediction systems due to the dynamic nature and volatility of financial markets.

Figure 3.2 4 Key Agile Values.



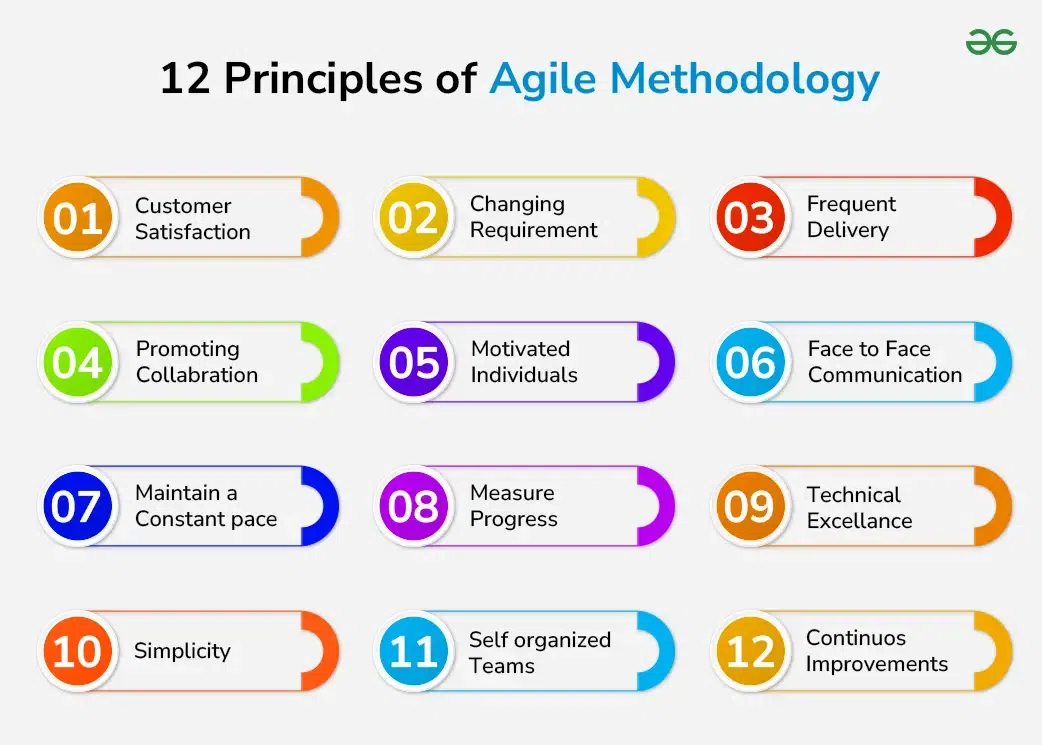
The Agile Manifesto outlines four key values that emphasize delivering value through simplicity, flexibility, and teamwork. One of the key values is individuals and interactions, which are prioritized over processes and tools. Collaboration among data scientists, developers, and stakeholders ensures effective communication for building accurate models in development of proposed systems.

The second key value is working software, which is prioritized over comprehensive documentation. Proper and completeness of documentation is important as it focuses on delivering functional requirements such as delivering a machine learning model that accurately predicts stock prices.

The third key value is customer collaboration, which is prioritized over contract negotiation. Continuous feedback from stakeholders like investors or financial analysts helps to improve the stock prediction system to meet customer demands and real-world needs.

Lastly, responding to change is prioritized over following a fixed plan. Financial markets are highly volatile, so new trends or data sources may emerge. Agile software development modelallows adaptability to these changes to ensure that Long Short-Term Memory (LSTM)model remains accurate and persistent.

Figure 3.3 12 Principles of Agile Methodology.



The Agile Manifesto also outlines 12 principles that guide the development process. These principles are highly relevant to our stock price prediction project as they promote adaptability, collaboration, continuous improvement, and value delivery (David Hartshorne, 2024). For example, delivering working software frequently aligns with our iterative development approach, where predictive models are updated and tested regularly. Accepting changing requirements is essential because financial data and market trends can shift unexpectedly.

Collaboration between team members and stakeholders ensures that we deliver accurate and reliable predictions. Face-to-face communication and motivated teams further improve the quality of the model. Progress is measured through the accuracy of predictions, while technical excellence is achieved by fine-tuning algorithms and optimizing the performance of machine learning models. Continuous reflection and improvement ensure the project evolves with changing requirements and market conditions.

### Phases of Agile Software Development Model

Figure 3.4 Agile Cycle.

### https://lh7-rt.googleusercontent.com/docsz/AD_4nXeQ92YYeuJK-Yh5JGmDVpcpmcMu5NqHqzpH-7m4kNO81kLZxuGTwAeXAf9h1cUdDnpgLlsGZ4JM3HH7x2Fp3bdU-jZYXUyZNBYg-yUo3RiJJ9f-_scDHEBC0yNDdC-xT9ee_Uxa?key=O5K1tD-s5uYdq4pDIisSzZHQ

Firstly, the concept phase includes the proposed project's overall objective, which is to predict stock prices accurately by using the Long Short-Term Memory (LSTM) model and integrate it with the Power BI dashboard. Identification of the goals is also carried out in this phase, such as integrating historical stock data, implementing Long Short-Term Memory (LSTM) model, and building an interactive user interface for stakeholders by using Power BI. Key users such as investors, financial analysts, and traders are identified as the system's target audience throughout the whole proposed project.

Next, a backlog of features and functionalities required for the stock price prediction system is created in the inception phase. These features include data collection from stock exchanges in websites and APIs, preprocessing financial data, building machine learning models such as regression and LSTM and visualizations of predicted results. User stories are used to break down complex tasks into smaller units. For example, “As a trader, I want to see a graph comparing actual and predicted stock prices.”

Besides, the iteration phase involves planning and developing features within short sprints, which often lasts from 1 to 4 weeks. In the first sprint, the proposed project may focus on collecting historical stock data. In subsequent sprints, Long Short-Term Memory (LSTM) networks may be implemented to the stock price prediction system to improve accuracy of results.

Moreover, the testing phase focuses on developing, testing, and improving the stock price prediction system's features. The predictive models are tested against historical stock data to measure their accuracy and reliability in this phase. The Long Short-Term Memory (LSTM) model will undergo rigorous validation, and metrics such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) are used to assess performance of Long Short-Term Memory (LSTM) model.

In addition, the production phase involves deployment of the stock price prediction system for stakeholders to use it. An initial version of the system may be released as a Minimum Viable Product (MVP) with basic features such as price prediction for selected stocks, historical comparisons, and useful visualizations in user interface. Feedback from stakeholders helps to identify areas for improvement and potential features that can be added to the stock price prediction system.

Lastly, presentation of the stock price prediction system to stakeholders and collection of feedback on its performance and usability are conducted in the review phase. For instance, stakeholders may request additional features such as incorporating real-time market data, improving prediction accuracy, adding visualization tools and so on. Feedbacks and reviews are analyzed to evaluate challenges and identify improvements for the next sprint to ensure the continuous improvement of the stock price prediction system.

## Functional and Non-functional Requirements

There are several functional and non-functional requirements that need to be considered when developing the stock price prediction system as mentioned in the proposed project. This ensures the completeness and usability of system features and ensures continuous improvement of the system.

### Functional Requirements

The system must collect historical stock price data from reliable sources, such as stock exchanges or third-party APIs such as Yahoo Finance and Google Chrome. The dataset will include stock symbols, opening prices, closing prices, trading volume, and other relevant labels. The collected data must be preprocessed by handling missing values, removing duplicates, handling outliers and so on to ensure quality for accurate predictions.

The system must allow users to input specific stock symbols and define prediction timeframes, such as short-term periods, which is minimum of one day, or long-term periods, which is minimum a month. The system will generate tailored predictions to help users in planning investment decisions with relevant forecasts based on the inputs provided.

The system must include displaying visualizations, such as line charts, scatter plots, and comparison graphs, to present stock predictions alongside actual prices. These visualizations will help users analyze prediction trends effectively and interpret results intuitively.

The system must include model evaluation features to assess prediction performance. Metrics such as Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and accuracy percentages must be calculated to provide insights into the reliability and accuracy of predictions.

The system must integrate real-time data feeds to support live stock price updates. This feature will ensure that predictions are aligned with current market conditions, helping users make timely and informed decisions.

The system must allow users to set up custom alerts based on stock price thresholds or prediction trends. Notifications must be sent via email or SMS when prices reach predefined values, enabling users to take prompt actions.

The system must include a comprehensive reporting feature that generates summaries of stock predictions. Reports must include prediction accuracy, visual charts, and statistical analysis, and they should be exportable in formats such as PDF or Excel for further review and analysis.

The system must incorporate user roles and authentication to ensure secure access. Different permissions must be provided for roles such as financial analysts, investors, and administrators, ensuring that sensitive features and data are accessible only to authorized users.

### Non-functional Requirements

The system must ensure high performance by processing large datasets efficiently and delivering stock price predictions within a short response time. For small datasets, the system response time should not exceed 3 seconds to maintain a seamless user experience.

The system must be reliable and maintain high availability, ensuring a 99.9% uptime during critical market hours. Robust error-handling mechanisms must be in place to detect and manage failures while providing meaningful feedback to users.

The system must achieve a minimum prediction accuracy of 85%. Machine learning models must be tested and optimized using appropriate metrics to ensure stakeholders can rely on the predictions for critical decision-making processes.

The system must implement strong security measures, including data encryption for sensitive information and secure APIs to prevent unauthorized access. Role-based authentication must restrict access to critical functionalities and protect user data.

The system must support audit logging to track user activities, such as data access, model usage, and any configuration changes made to the system. Logs will help administrators monitor system usage and ensure accountability.

The system must provide a user-friendly interface that is intuitive, responsive, and accessible to users with varying technical expertise. Features such as clear navigation, tooltips, and help documentation will improve usability for all users.

The system must be cross-platform compatible, allowing users to access it from desktops, tablets, and smartphones without any loss of functionality or performance. A consistent user experience must be maintained across devices.

The system must be designed to allow scalability and maintainability. It should support future enhancements, such as integrating new data sources, adding advanced machine learning models, or improving analytical features, without requiring significant redevelopment.

The system must perform regular updates and maintenance to ensure long-term system performance and stability. The system should include monitoring tools to detect issues early and provide smooth operations during updates.

## Chapter Summary and Evaluation

This chapter outlined the methodology and requirements analysis process for developing the proposed stock price prediction system. The Agile methodology was adopted to allow iterative development and timely deliverables. Requirements gathering techniques, including interviews, surveys, and document analysis, ensured a clear understanding of user needs. The analysis phase incorporated diagrams such as use case diagrams, flowcharts, and ERDs to visualize the system’s design. Finally, both functional and non-functional requirements were identified to ensure the system is accurate, efficient, and user-friendly. This comprehensive foundation sets the stage for model development, testing, and integration in subsequent chapters.

Chapter 4

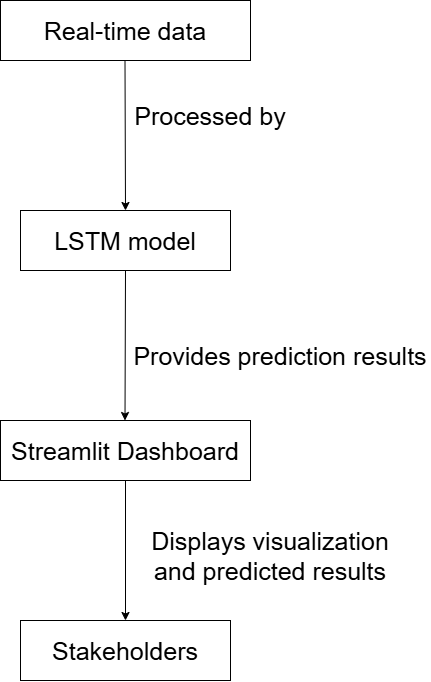
**System Design**

# System Design

This chapter outlines the system design architecture and its diagram. In this chapter, the integration and roles of LSTM model, Power BI and Firebase are shown. The system design will be implemented to prepare the stock price prediction system to be deployed in a live environment.

## System Design Architecture

Figure 4.1 System design diagram



## Role of Long Short-Term Memory Model

LSTM networks are a type of recurrent neural network (RNN). They are good at learning and predicting time series data. They capture long-term patterns and dependencies. In this system, the LSTM model uses historical stock price data to predict future prices. The model handles sequential data and remembers information for a long time. This makes it a good choice for understanding the patterns in stock market movements.

## Role of Streamlit

Streamlit is a tool for creating data visualizations. It turns raw data and predictions into simple dashboards. In this project, Streamlit shows the predictions from the LSTM model and the historical data. Stakeholders can analyze trends and compare actual prices with predictions. They can use filters and drill-down features to customize their view. This helps users explore the data and focus on what they need.

## System Integration

The integration of three features above ensures that users can get accurate predictions, real-time data updates, and useful visualizations. This enhances the overall usability and effectiveness of the stock price prediction system.

### Data Flow

The real-time updates make sure the data is current. After this, the data is preprocessed to fix missing values, outliers, or scaling issues. This step gets the data ready for the LSTM model. Preprocessing is important to make the model work well.

### Prediction Generation

The LSTM model is a type of neural network. It looks at the stock price data from Firebase. The model finds patterns and trends in the data over time. It uses this to predict future stock prices. The model remembers past data, which is important for forecasting. The predictions show possible changes in the market.

### Data Visualization

Streamlit is used to show the stock data and predictions. The live stock data from Firebase and the predictions from the LSTM model are imported into Streamlit. Dashboards are made to display key metrics and trends. These dashboards help users see the data in a clear way. The live updates and predictions are combined to show both current and future stock information.

### Interaction with users

Streamlit enables users to interact with the dashboards. Users can filter, drill down, and change views. This helps them explore specific stocks or time frames. The design makes it easy for users to understand the data. It helps them to analyze and use the information effectively.

## Chapter Summary and Evaluation

In conclusion, this chapter has stated the roles of LSTM model and Streamlit in the integrated system. System design diagrams are also shown to enable the understanding of relationships between them. The design will be implemented to develop a stock price prediction system for deployment.

Chapter 5

**Implementation and Testing**

# Implementation and Testing

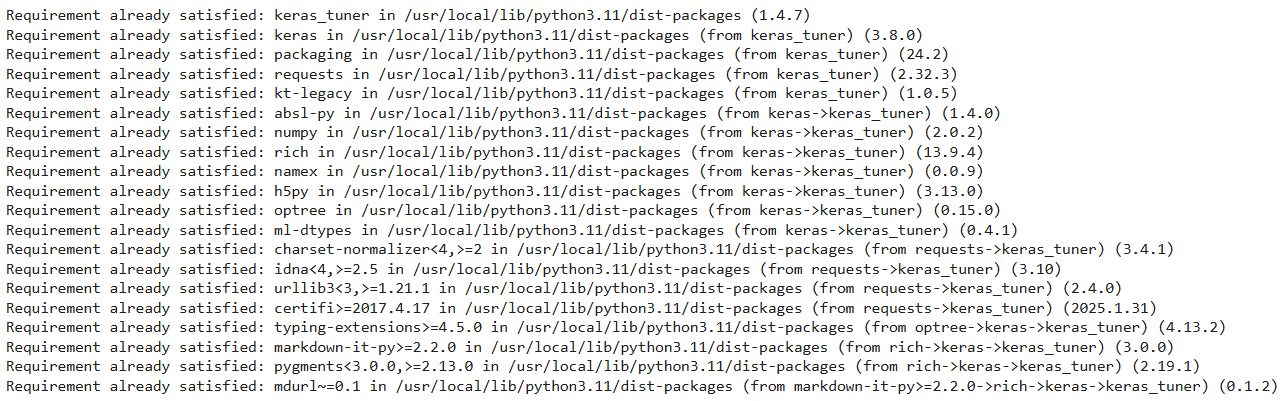
Chapter 5 outlines the implementation and testing of my stock price prediction model. In this chapter, I will explain about the code snippets that I have used in the development of the stock price prediction model using LSTM. Testing of the model will be also demonstrated in this chapter. Each code snippet will be explained in this chapter.

## Import Required Libraries

| # Install keras\_tuner  !pip install keras\_tuner |
| --- |

Output:

Figure 5.1 Install keras\_tuner.



Explanation:   
This code enables the notebook to run a shell command by using “!” to install the keras\_tuner package via pip. Keras Tuner hyperparameter tuning framework developed by the TensorFlow/Keras team. It automatically tries different combinations of hyperparameters such as number of units, dropout rate, learning rate to find the best model configuration based on validation performance.

| # Importing the required libraries  import kagglehub  import numpy as np  import matplotlib.pyplot as plt  import pandas as pd  import seaborn as sns  from sklearn.preprocessing import MinMaxScaler  import tensorflow as tf  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, SimpleRNN, LSTM, Input, GRU, Dropout, BatchNormalization  from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint  from tensorflow.keras.models import load\_model  from tensorflow.keras.optimizers import Adam  import keras\_tuner as kt  from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score  from sklearn.ensemble import RandomForestRegressor  from sklearn.svm import SVR  from statsmodels.tsa.arima.model import ARIMA  from datetime import date, timedelta  from google.colab import drive, files  import warnings  warnings.filterwarnings("ignore")  import itertools  import random  import os |
| --- |

Explanation:

* kagglehub: It is used to download the latest datasets from [kagglehub](https://github.com/kagglehub), which is the historical stock data.
* numpy: For numerical operations and array manipulation.
* matplotlib.pyplot: For plotting graphs and visualizing trends.  
  pandas: For reading, processing, and analyzing structured data (e.g., CSV files).
* seaborn: A higher-level plotting library built on top of matplotlib for attractive plots.
* MinMaxScaler: Used to scale data into a specific range (commonly 0–1), which is essential for training LSTM.
* Sequential: To build a linear stack of neural network layers.
* Dense: Fully connected layer.
* SimpleRNN, LSTM, GRU: Recurrent layers, which is LSTM is the main model for time series forecasting of stock price.
* Input: To explicitly define input shape if needed.
* Dropout, BatchNormalization: Regularization layers to prevent overfitting and stabilize training.
* EarlyStopping, ModelCheckpoint: Callbacks to stop training early and save the best model.
* load\_model: To reload trained models from disk.
* Adam: A popular optimizer for training deep learning models.
* keras\_tuner: For automating hyperparameter tuning.
* mean\_squared\_error, mean\_absolute\_error, r2\_score: Used to evaluate model prediction accuracy.
* RandomForestRegressor, SVR: Machine learning models for benchmarks.
* ARIMA: A traditional time-series model (AutoRegressive Integrated Moving Average) for baseline benchmarking.
* datetime, timedelta: For date manipulation.
* google.colab.drive, files: To interact with Google Drive, such as saving or loading models.
* warnings.filterwarnings("ignore"): To suppress warning messages in output.
* itertools, random, os: Standard Python utilities for iteration, randomness, and file operations.

## Load dataset

| # Download latest version of dataset  file\_path = kagglehub.dataset\_download("nelgiriyewithana/world-stock-prices-daily-updating")  print("Path to dataset files:", file\_path) |
| --- |

Output:

Path to dataset files: /kaggle/input/world-stock-prices-daily-updating

Explanation:

This code snippet is used to download and verify the latest version of a stock price dataset from KaggleHub, which is a tool designed to simplify access to Kaggle-hosted datasets.

| # Filename of dataset  filename = "World-Stock-Prices-Dataset.csv"  file\_path = os.path.join(file\_path, filename) |
| --- |

Explanation:

This code snippet specifies the exact CSV file within the downloaded Kaggle dataset folder and builds its full path for loading.

| # Load data with pandas  df = pd.read\_csv(file\_path) |
| --- |

Explanation:

This code snippet defines the dataset with variables named “df”.

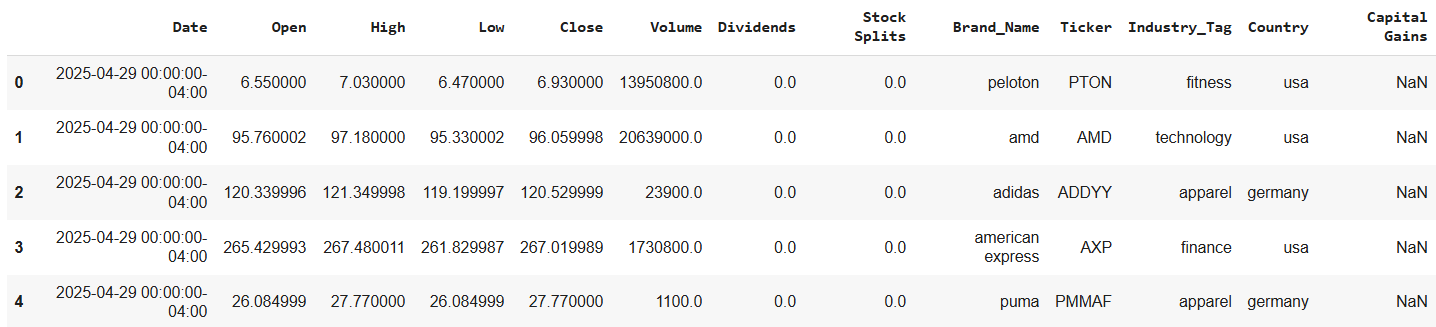
## Exploratory Data Analysis (EDA)

### Data Preview

| # First 5 rows of data  df.head(5) |
| --- |

Output:

Figure 5.2 First 5 rows of data.



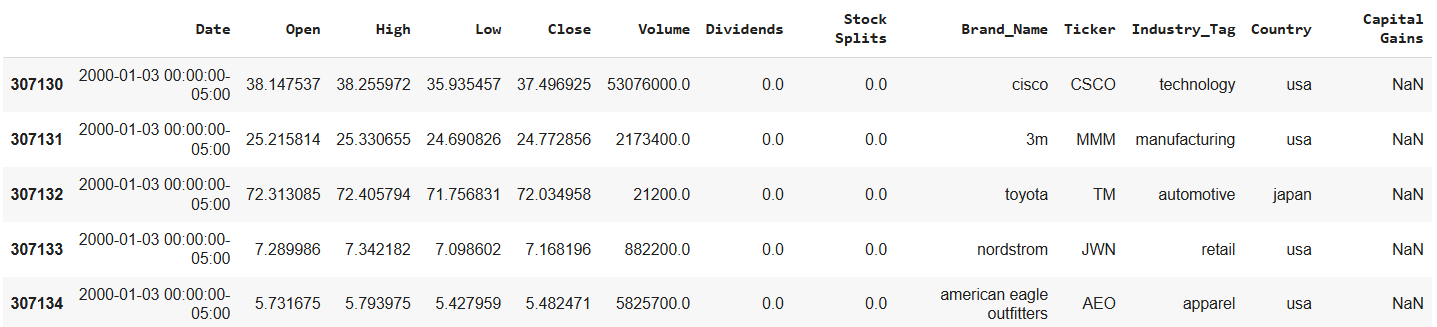
Explanation:

Display the first 5 rows of data.

| # Last 5 rows of data  df.tail(5) |
| --- |

Output:

Figure 5.3 Last 5 rows of data.



Explanation:

Display the last five rows of the data.

| # Reverse the order in dataset  df = df[::-1].reset\_index(drop=True) |
| --- |

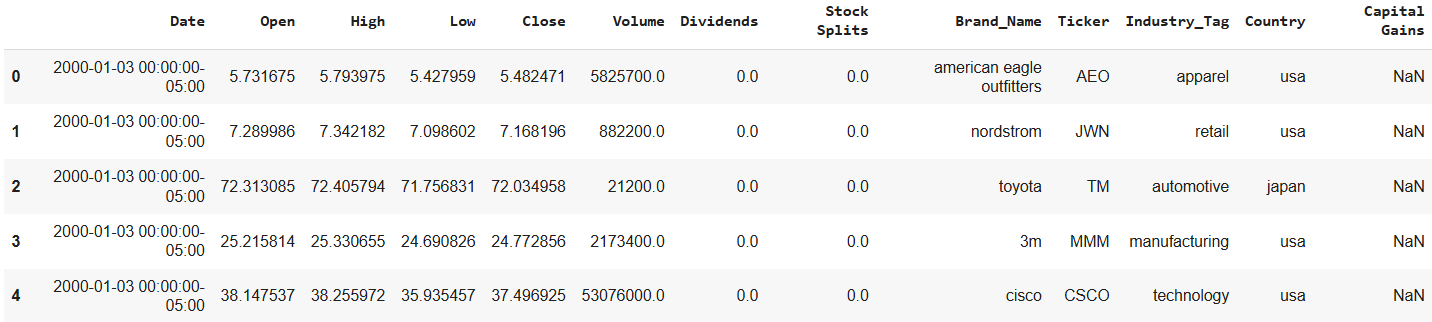
Explanation:

This code snippet reverses the order of rows in the dataset. This is to enable data to be presented starting from the past to the present.

| df.head(5) |
| --- |

Output:

Figure 5.4 First 5 rows of data (Reversed).



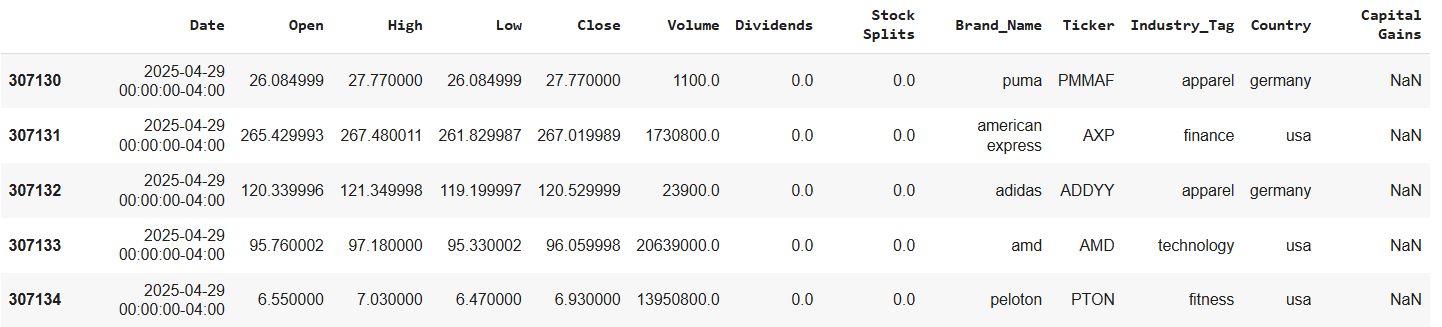
Explanation:

Display the first 5 rows of the reversed data.

| df.tail(5) |
| --- |

Output:

Figure 5.5 Last 5 rows of data (Reversed).



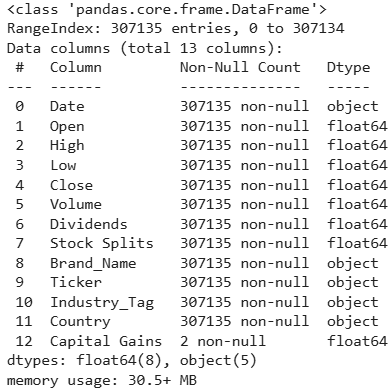
Explanation:

Display the last 5 rows of the reversed data.

| # Info of data  df.info() |
| --- |

Output:

Figure 5.6 Data Info.



Explanation:

This code snippet displays the column names, non-null count of columns and data type of columns, which are important information in the dataset.

| # Number of stocks with no duplication  print("Number of stocks:", len(df['Ticker'].unique())) |
| --- |

Output:



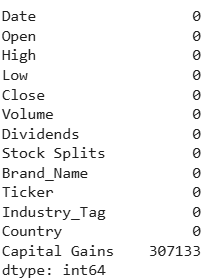
Explanation:

This code snippet displays the number of different stocks in the dataset, which is 61.

| # Number of null values in each column  print(df.isnull().sum()) |
| --- |

Output:

Figure 5.7 Null values.



Explanation:

This code snippet displays the number of null values for each column in the dataset.

| # Select a Specific Stock  selected\_stock = "AAPL" # Change ticker if needed  df\_stock = df[df["Ticker"] == selected\_stock].sort\_values(by="Date").reset\_index(drop=True) |
| --- |

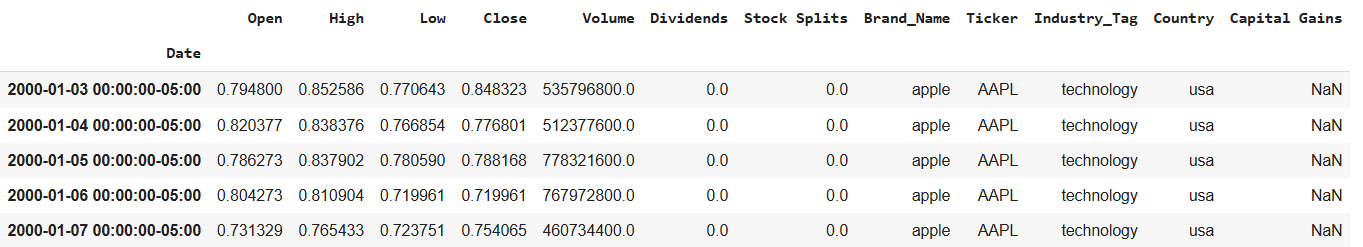
Explanation:

This code snippet enables the selection of one specific stock to be preprocessed and predicted.

| # Change date to datetime and set the date as index  df\_stock['Date'] = pd.to\_datetime(df\_stock['Date'])  df\_chg= df\_stock.set\_index(['Date'], drop=True)  df\_chg.head() |
| --- |

Output:

Figure 5.8 Change date to datetime and set date as index.



Explanation:

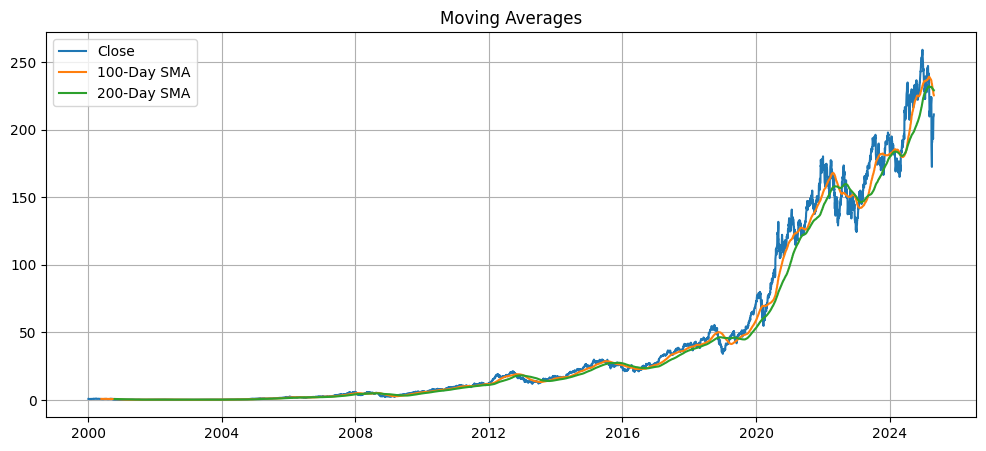
This code snippet changes the dates in the “Date” column to datetime. Then, replace the index in the index row with dates. This is to enable time-series forecasting later.

### Moving Average

| # Plot the moving average graph  df\_chg['SMA\_100'] = df\_chg['Close'].rolling(window=100).mean()  df\_chg['SMA\_200'] = df\_chg['Close'].rolling(window=200).mean()  plt.figure(figsize=(12, 5))  plt.plot(df\_chg['Close'], label='Close')  plt.plot(df\_chg['SMA\_100'], label='100-Day SMA')  plt.plot(df\_chg['SMA\_200'], label='200-Day SMA')  plt.title('Moving Averages')  plt.legend()  plt.grid(True)  plt.show() |
| --- |

Output:

Figure 5.9 Moving Average.



Explanation:

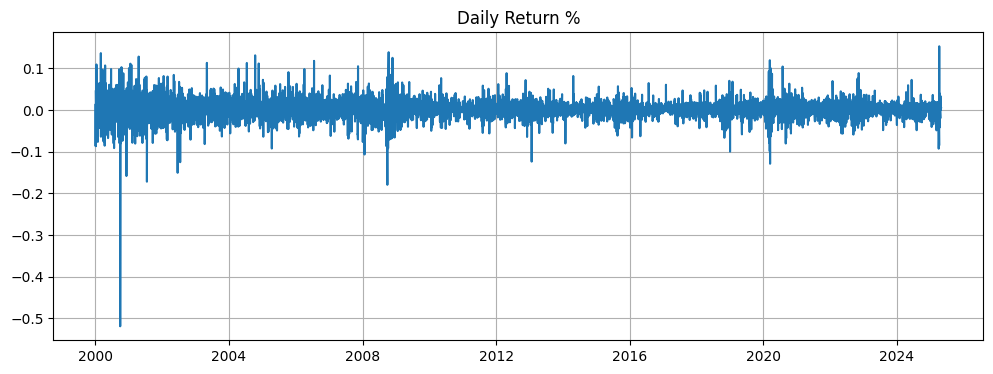
This code snippet plots 100-day moving average and 200-day moving average of close price.

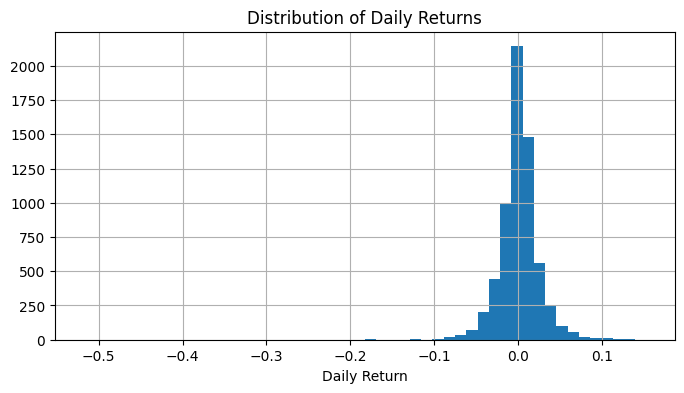
### Daily Return

| # Plot the percentage of daily return of stock  df\_chg['Daily Return'] = df\_chg['Close'].pct\_change()  plt.figure(figsize=(12, 4))  plt.plot(df\_chg['Daily Return'])  plt.title('Daily Return %')  plt.grid(True)  plt.show()  # Histogram  df\_chg['Daily Return'].hist(bins=50, figsize=(8, 4))  plt.title('Distribution of Daily Returns')  plt.xlabel('Daily Return')  plt.show() |
| --- |

Output:

Figure 5.10 Percentage of Daily Return.





Explanation:

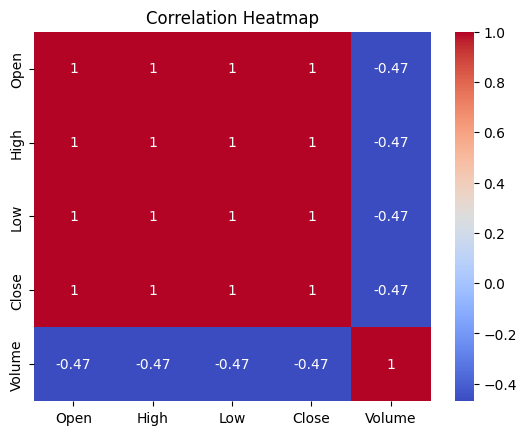
This code snippet plots the percentage change of daily return of the selected stock in line histogram.

### Correlation Heatmap

| import seaborn as sns  sns.heatmap(df\_chg[['Open', 'High', 'Low', 'Close', 'Volume']].corr(), annot=True, cmap='coolwarm')  plt.title('Correlation Heatmap')  plt.show() |
| --- |

Output:

Figure 5.11 Correlation Heatmap.



Explanation:

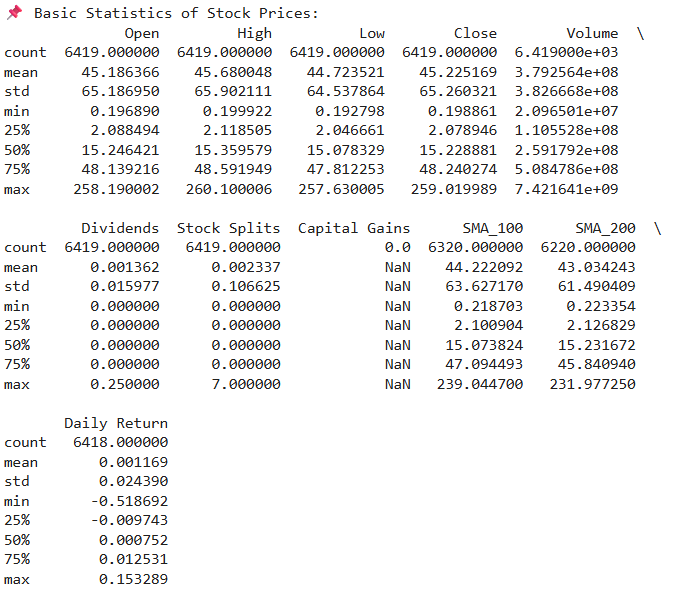
This code snippet generates a correlation heatmap between column names which have numerical values to analyse their correlations.

### Summary of Data

| # Summary Statistics  print("\n📌 Basic Statistics of Stock Prices:")  print(df\_chg.describe()) |
| --- |

Output:

Figure 5.12 Data Summary.



Explanation:

This code snippet shows the basic statistics of the data in each column.

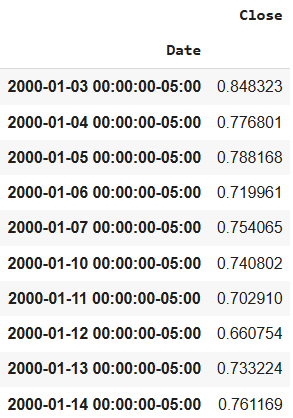
## Data Preprocessing

### Feature Engineering

| # Drop all columns except for 'Date' and 'Close' column  df\_chg.drop(['Open', 'High', 'Low', 'Volume', 'Brand\_Name', 'Ticker', 'Industry\_Tag', 'Country', 'Dividends', 'Stock Splits', 'Capital Gains', 'SMA\_100', 'SMA\_200', 'Daily Return'], axis=1, inplace=True)  df\_chg.head(10) |
| --- |

Output:

Figure 5.13 Feature Engineering.



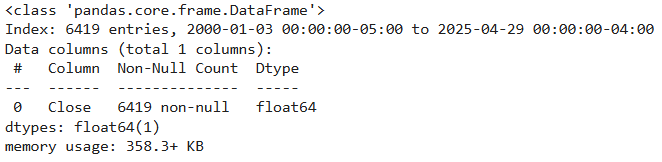
Explanation:

This code snippet drops all columns, except for “Date” and “Close” columns. This enables us to use dates and close prices to predict future stock prices using the LSTM model.

| # Ensure the columns are dropped  df\_chg.info() |
| --- |

Output:

Figure 5.14 Verification for feature extraction.



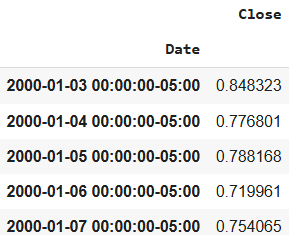
Explanation:

This code snippet shows the basic information of preprocessed data. This is to ensure that the other columns are dropped, except for close prices.

| df\_chg.head(5) |
| --- |

Output:

Figure 5.15 First 5 rows of extracted features.



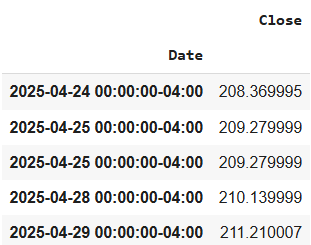
Explanation:

This code snippet shows the first 5 rows of preprocessed data.

| df\_chg.tail(5) |
| --- |

Output:

Figure 5.16 Last 5 rows of extracted features.



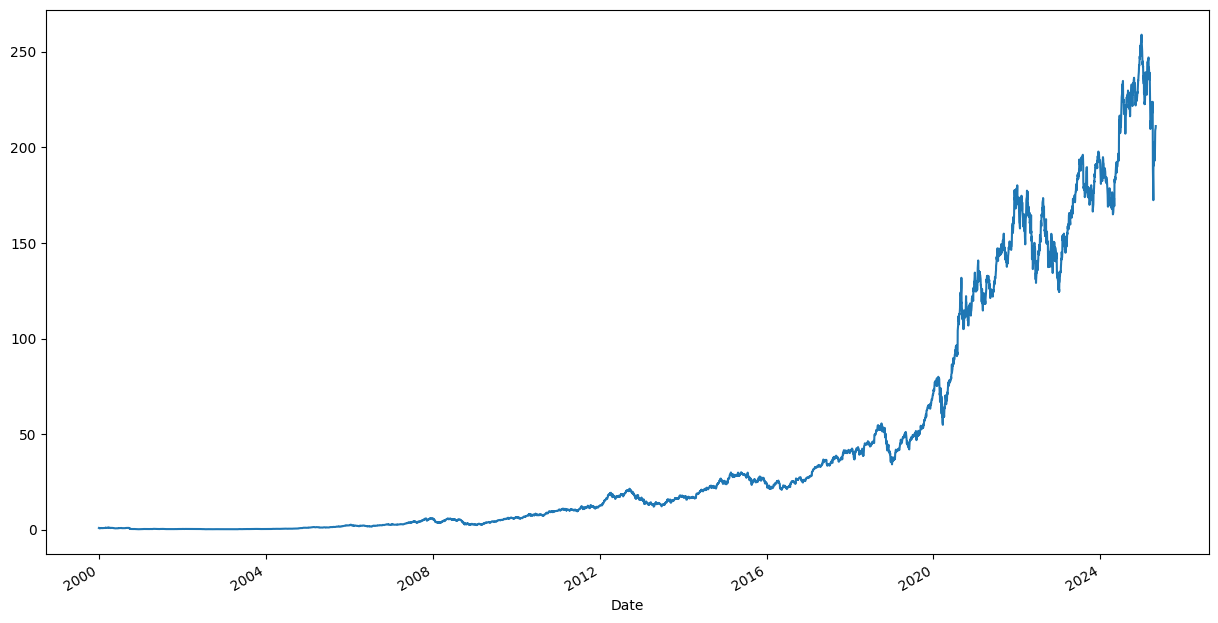
Explanation:

This code snippet shows the first 5 rows of preprocessed data.

| #Plot the close price plot  plt.figure(figsize=(15,8))  df\_chg['Close'].plot(); |
| --- |

Output:

Figure 5.17 Closing price plot.



Explanation:

This code snippet plots the historical close price from the beginning until now.

### Normalization (Using MinMaxScaler)

| # Normalize the data  scaler = MinMaxScaler(feature\_range=(0, 1))  scaled\_data = scaler.fit\_transform(df\_chg) |
| --- |

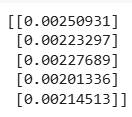
Explanation:

This code snippet normalizes the preprocessed data using MinMaxScaler. Then, the scaled data is transformed to be trained afterwards.

| # Display the first 5 scaled values  print(scaled\_data[:5]) |
| --- |

Output:

Figure 5.18 First 5 scaled data.



Explanation:

This code snippet shows the first 5 scaled data to ensure that the preprocessed data is scaled successfully.

### Split Training and Testing Data

| # Create a function to convert data into sequences for LSTM  def create\_sequences(data, time\_steps=60):  X, y = [], []  for i in range(len(data) - time\_steps):  X.append(data[i:i+time\_steps, 0])  y.append(data[i+time\_steps, 0])  return np.array(X), np.array(y)  # Use 60 time steps to predict the next value  time\_steps = 60  X, y = create\_sequences(scaled\_data, time\_steps)  # Reshape X for LSTM input  X = X.reshape(X.shape[0], X.shape[1], 1) |
| --- |

Explanation:

This snippet defines and uses a function to prepare time-series data for training an LSTM model, which expects a 3D input.

| # Split the data into training and test sets (80% train, 20% test)  train\_size = int(len(X) \* 0.8)  X\_train, X\_test = X[:train\_size], X[train\_size:]  y\_train, y\_test = y[:train\_size], y[train\_size:] |
| --- |

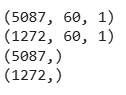
Explanation:

This code snippet splits data into 80% training data and 20% testing data.

| print(X\_train.shape)  print(X\_test.shape)  print(y\_train.shape)  print(y\_test.shape) |
| --- |

Output:

Figure 5.19 Training and testing shapes.



Explanation:

This code snippet shows the size of training and testing data. Both for list of sequences (X) and list of targets (y). This is to ensure that the training and testing data are splitted correctly.

## Define LSTM Model

| # Initialize the Sequential model  model = Sequential()  # Define the input layer  model.add(Input(shape=(time\_steps, 1)))  # Add the LSTM layer  model.add(LSTM(50, return\_sequences=False))  # Normalizes layer outputs to stabilize and speed up training  model.add(BatchNormalization())  # Dropout to regularize the LSTM  model.add(Dropout(0.2))  # Add a Dense layer for output  model.add(Dense(1))  # Compile the model  model.compile(optimizer='adam', loss='mean\_squared\_error', metrics=[tf.keras.metrics.RootMeanSquaredError()]) |
| --- |

Explanation:

This snippet initializes and builds the core architecture of the LSTM model for time series prediction using Keras. Sequential, input shape, LSTM layer, BatchNormalization, and Dropout are defined in this snippet. Then, the model is compiled using adam optimizer.

## Training Data

| # Define EarlyStopping Callback  es = EarlyStopping(  monitor='val\_loss',  patience=5,  restore\_best\_weights=True  ) |
| --- |

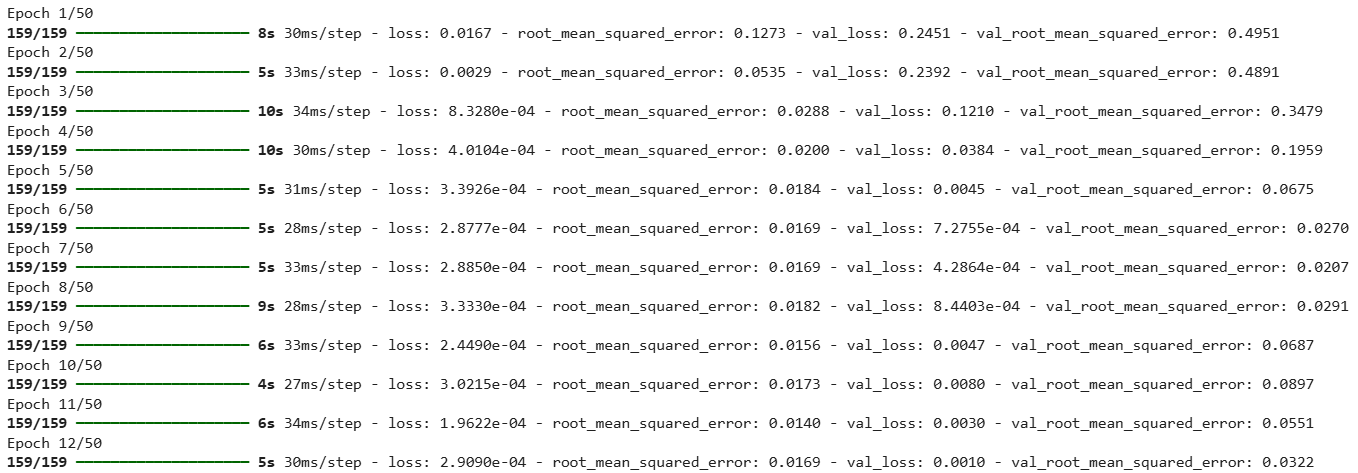
Explanation:

This code snippet defines the EarlyStopping to prevent overfitting of the model by stopping the model training if there are no further improvements.

| # Train the model on the training data  history = model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_data=(X\_test, y\_test), callbacks = es, verbose = 1) |
| --- |

Output:

Figure 5.20 Training data.



Explanation:

This code snippet trains the model using the LSTM model defined earlier and applied EarlyStopping callback to prevent overfitting. The epochs, training loss, RMSE, validation loss and validation RMSE are shown.

| # List the names of the metrics and losses that were recorded  history.history.keys() |
| --- |

Output:

Figure 5.21 List of metrics and lossess.



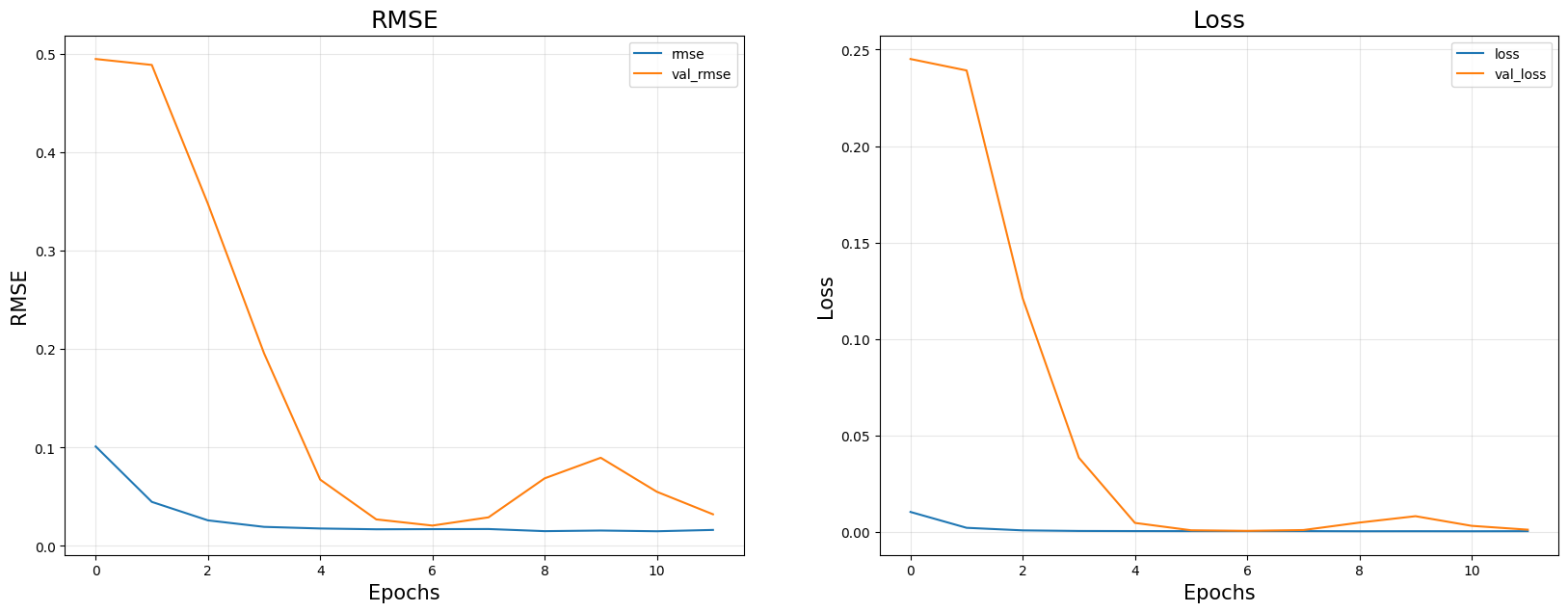
Explanation:

This code snippet is used to list the metrics and losses that were recorded during training.

| fig = plt.figure(figsize=(20,7))  fig.add\_subplot(121)  # Accuracy curve (RMSE)  plt.plot(history.epoch, history.history['root\_mean\_squared\_error'], label = "rmse")  plt.plot(history.epoch, history.history['val\_root\_mean\_squared\_error'], label = "val\_rmse")  plt.title("RMSE", fontsize=18)  plt.xlabel("Epochs", fontsize=15)  plt.ylabel("RMSE", fontsize=15)  plt.grid(alpha=0.3)  plt.legend()  # Loss Curve  fig.add\_subplot(122)  plt.plot(history.epoch, history.history['loss'], label="loss")  plt.plot(history.epoch, history.history['val\_loss'], label="val\_loss")  plt.title("Loss", fontsize=18)  plt.xlabel("Epochs", fontsize=15)  plt.ylabel("Loss", fontsize=15)  plt.grid(alpha=0.3)  plt.legend()  plt.show() |
| --- |

Output:

Figure 5.22 RMSE and Loss curve.



Explanation:

This code snippet is used to plot the RMSE curve and loss curve for the trained model. This is to analyse whether the model is trained properly or not by looking at the training and validation plots.

## Hyperparameter Tuning

| # Define tuning model for LSTM  def build\_model(hp):  model = Sequential()  model.add(Input(shape=(time\_steps, 1)))  # LSTM Units (choose between 32 to 128 in steps of 16)  model.add(LSTM(units=hp.Int('lstm\_units', min\_value=32, max\_value=128, step=16),  return\_sequences=False))  # Dropout rate  model.add(Dropout(rate=hp.Float('dropout\_rate', min\_value=0.1, max\_value=0.5, step=0.1)))  # BatchNormalization  model.add(BatchNormalization())  # Dense Output  model.add(Dense(1))  # Learning Rate  learning\_rate = hp.Float('lr', min\_value=1e-4, max\_value=1e-2, sampling='log')  optimizer = Adam(learning\_rate=learning\_rate)  model.compile(optimizer=optimizer,  loss='mean\_squared\_error',  metrics=[tf.keras.metrics.RootMeanSquaredError()])  return model |
| --- |

Explanation:

This function defines a tunable LSTM model architecture for time series forecasting using Keras Tuner. It allows automatic optimization of key hyperparameters such as the number of LSTM units (32–128), dropout rate (0.1–0.5), and learning rate (1e-4 to 1e-2). The model includes an LSTM layer for capturing temporal patterns, followed by dropout and batch normalization for regularization and training stability, and ends with a dense output layer to predict the next value. The model is compiled using the Adam optimizer and optimized for mean squared error with RMSE as the evaluation metric.

| # Define the function for searching best parameters  tuner = kt.RandomSearch(  build\_model,  objective='val\_root\_mean\_squared\_error',  max\_trials=10,  executions\_per\_trial=1,  directory='tuner\_dir',  project\_name='lstm\_stock\_tuning'  ) |
| --- |

Explanation:

This snippet sets up a RandomSearch tuner from Keras Tuner to automatically search for the best hyperparameters for the LSTM model defined in build\_model. The tuner optimizes for the lowest validation Root Mean Squared Error (val\_root\_mean\_squared\_error) over 10 different trials, each testing a different combination of hyperparameters. Each trial is run once (executions\_per\_trial=1). The search results, including the best model and metrics, are saved in the folder tuner\_dir/lstm\_stock\_tuning, allowing reuse or inspection later.

| # Search for best parameters  tuner.search(X\_train, y\_train,  validation\_split=0.2,  epochs=20,  callbacks=[EarlyStopping(patience=3)],  verbose=1) |
| --- |

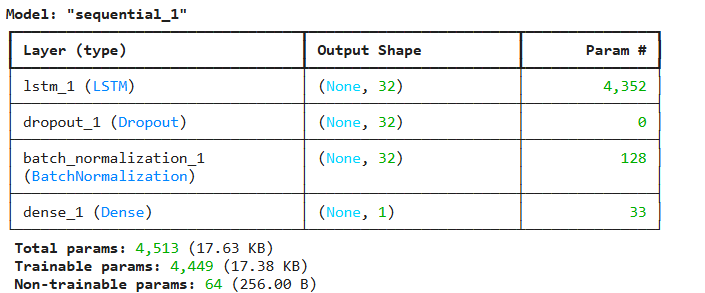
Explanation:

This command starts the hyperparameter search by training different LSTM model configurations on the training data (X\_train, y\_train). It uses 20% of the training data for validation and trains each model for up to 20 epochs, with early stopping applied if validation loss doesn’t improve after 3 epochs. The tuner evaluates each combination of hyperparameters defined in build\_model, logs their performance, and identifies the configuration that gives the lowest validation RMSE.

| # List the best parameters  best\_hp = tuner.get\_best\_hyperparameters(1)[0]  model = tuner.hypermodel.build(best\_hp)  model.summary() |
| --- |

Output:

Figure 5.23 Best hyperparameters.



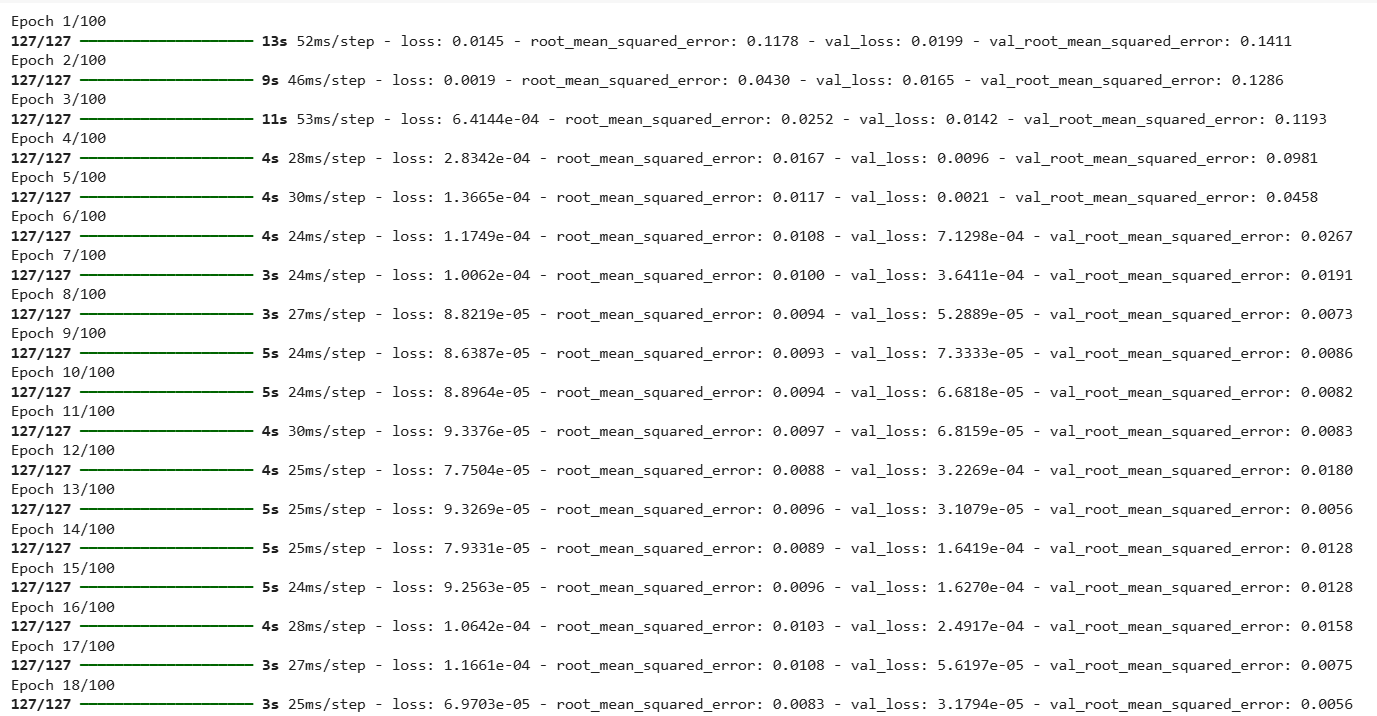
Explanation:

After the search is complete, this code selects the best-performing hyperparameter set using tuner.get\_best\_hyperparameters(1)[0]. It then rebuilds the LSTM model using this optimal configuration with tuner.hypermodel.build(best\_hp). Finally, model.summary() prints the architecture of the tuned model, showing the layer structure, parameter counts, and overall model shape.

| # Train using best parameters  history = model.fit(X\_train, y\_train,  validation\_split=0.2,  epochs=100,  callbacks=[EarlyStopping(patience=5)],  verbose=1) |
| --- |

Output:

Figure 5.24 Train using best hyperparameters.



Explanation:

This command trains the LSTM model built with the best hyperparameters on the full training set (X\_train, y\_train) using a 20% validation split. It trains for up to 100 epochs, but includes early stopping to halt training if the validation loss doesn't improve for 5 consecutive epochs. The training history, including loss and RMSE for each epoch, is stored in the history object for later analysis or plotting.

| # Download the best model  files.download('best\_model.h5') |
| --- |

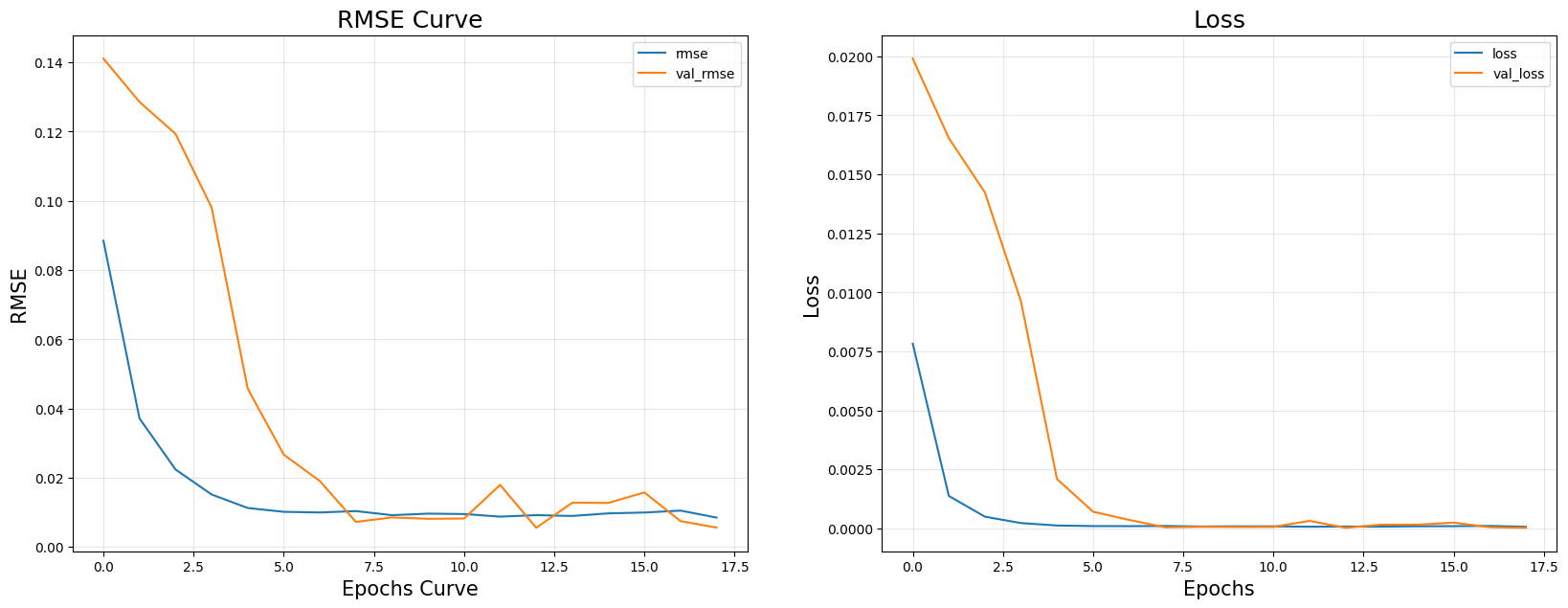
Explanation:

This snippet downloads the best model as a file on the computer.

| fig = plt.figure(figsize=(20,7))  fig.add\_subplot(121)  # Accuracy curve  plt.plot(history.epoch, history.history['root\_mean\_squared\_error'], label = "rmse")  plt.plot(history.epoch, history.history['val\_root\_mean\_squared\_error'], label = "val\_rmse")  plt.title("RMSE Curve", fontsize=18)  plt.xlabel("Epochs Curve", fontsize=15)  plt.ylabel("RMSE", fontsize=15)  plt.grid(alpha=0.3)  plt.legend()  # Loss Curve  fig.add\_subplot(122)  plt.plot(history.epoch, history.history['loss'], label="loss")  plt.plot(history.epoch, history.history['val\_loss'], label="val\_loss")  plt.title("Loss", fontsize=18)  plt.xlabel("Epochs", fontsize=15)  plt.ylabel("Loss", fontsize=15)  plt.grid(alpha=0.3)  plt.legend()  plt.show() |
| --- |

Output:

Figure 5.25 RMSE and Loss curve of best model.



Explanation:

This snippet creates two side-by-side subplots using matplotlib: one for Root Mean Squared Error (RMSE) and another for loss over training epochs. It plots both the training and validation metrics stored in the history object. The first plot shows how RMSE evolved during training and validation, helping evaluate model accuracy. The second plot shows the loss trend, which helps identify potential overfitting or underfitting. The plots include grid lines, legends, and titles for better readability.

## Testing Data

| # Load the best model  uploaded = files.upload() # Select best\_model\_colab.h5 from computer  model = load\_model('best\_model.h5', compile=False) |
| --- |

Explanation:

This snippet prompts users to choose the best model from the computer and upload it so that it can be used anytime.

| # Save model locally  model.save("best\_model.h5") |
| --- |

Explanation:

This snippet saves the uploaded model locally in Google Colab.

| # Make predictions on the test data  y\_pred = model.predict(X\_test) |
| --- |

Output:

Figure 5.26 Prediction generation on test data.



Explanation:

This snippet makes predictions on test data by using the uploaded model.

| # Inverse transform the predictions and actual values  y\_pred\_rescaled = scaler.inverse\_transform(y\_pred)  y\_test\_rescaled = scaler.inverse\_transform(y\_test.reshape(-1, 1)) |
| --- |

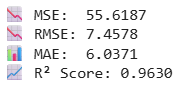
Explanation:

After predicting stock prices using the trained LSTM model, the outputs (y\_pred) and true values (y\_test) are still in the scaled range (e.g., 0 to 1) due to the use of MinMaxScaler during preprocessing. This code uses scaler.inverse\_transform() to reverse the scaling, converting both predictions and actual values back to their original price scale. This is crucial for evaluating prediction accuracy in real-world units.

| # Evaluation Metrics  mse\_lstm = mean\_squared\_error(y\_test\_rescaled, y\_pred\_rescaled)  rmse\_lstm = np.sqrt(mse\_lstm)  mae\_lstm = mean\_absolute\_error(y\_test\_rescaled, y\_pred\_rescaled)  r2\_lstm = r2\_score(y\_test\_rescaled, y\_pred\_rescaled)  print(f"📉 MSE: {mse\_lstm:.4f}")  print(f"📉 RMSE: {rmse\_lstm:.4f}")  print(f"📊 MAE: {mae\_lstm:.4f}")  print(f"📈 R² Score: {r2\_lstm:.4f}") |
| --- |

Output:

Figure 5.27 LSTM evaluation metrics.



Explanation:

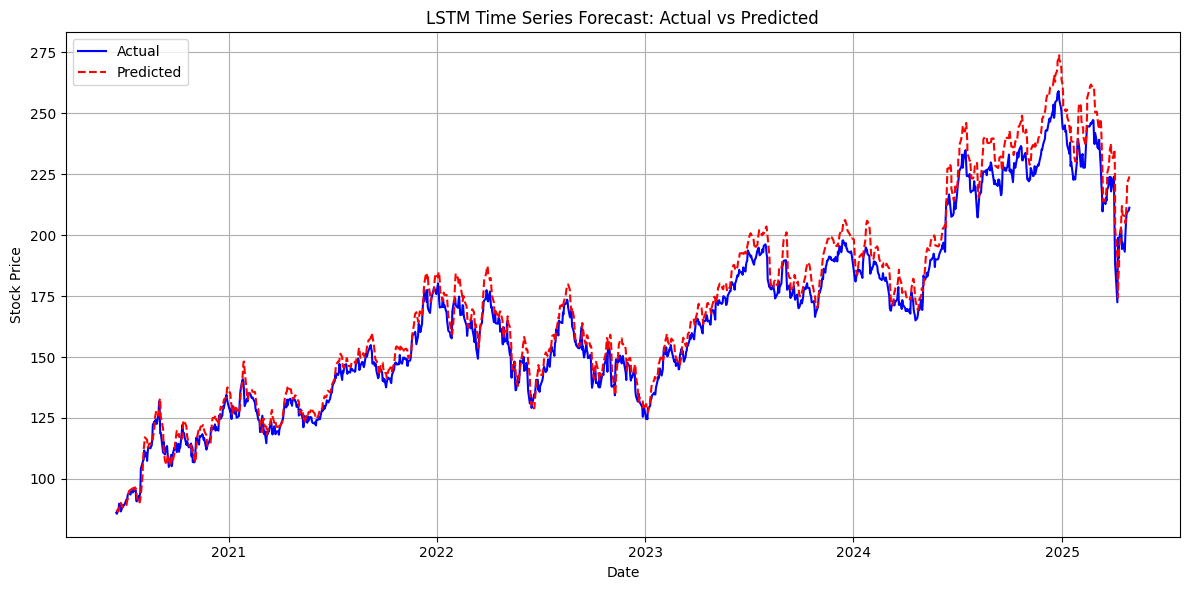
This block computes four important evaluation metrics:

* MSE (Mean Squared Error): the average of squared prediction errors.
* RMSE (Root Mean Squared Error): the square root of MSE, representing average prediction error in original units.
* MAE (Mean Absolute Error): the average of absolute differences between predicted and actual prices.
* R² Score: a measure of how well predictions explain the variance in actual data (1 is perfect, 0 means no predictive power).

| # Plot the actual vs predicted values  plt.figure(figsize=(12,6))  plt.plot(df\_chg.index[-len(y\_test\_rescaled):], y\_test\_rescaled, label='Actual', color='blue')  plt.plot(df\_chg.index[-len(y\_test\_rescaled):], y\_pred\_rescaled, label='Predicted', color='red', linestyle='dashed')  plt.title('LSTM Time Series Forecast: Actual vs Predicted')  plt.xlabel('Date')  plt.ylabel('Stock Price')  plt.legend()  plt.grid(True)  plt.tight\_layout()  plt.show() |
| --- |

Output:

Figure 5.28 Actual vs Predicted.



Explanation:

This plot visualizes the model’s predicted stock prices vs. the actual prices from the test dataset. It uses the last segment of the df\_chg.index for the x-axis. The blue line represents actual prices, while the red dashed line shows the model's predictions. This side-by-side view helps you visually assess prediction accuracy, spot patterns, and detect areas where the model performs well or underperforms. The chart includes labels, grid lines, and a legend for readability.

## Forecast Future Data

| # Define Parameters  n\_future\_days = 30 # How many days to forecast  recent\_sequence = X\_test[-1] # shape: (time\_steps, 1) |
| --- |

Explanation:

This code defines the number of days to forecast and sets up the input (recent\_sequence) for making future predictions. It takes the last input sequence from the test set, which contains the most recent time steps of stock prices, and uses it as the starting point to generate future predictions recursively. This approach allows the model to extend predictions beyond the test set into unseen future dates.

| # Start recursive prediction  future\_predictions = [] |
| --- |

Explanation:

This creates an empty list that will hold the stock prices predicted by the LSTM model for each of the upcoming n\_future\_days. In the next step, the model will use a recursive loop to predict one day at a time, append the result to this list, and use the new prediction as part of the input for predicting the next day — a process known as recursive or auto-regressive forecasting.

| # Reshape for prediction  current\_input = recent\_sequence.reshape(1, time\_steps, 1) |
| --- |

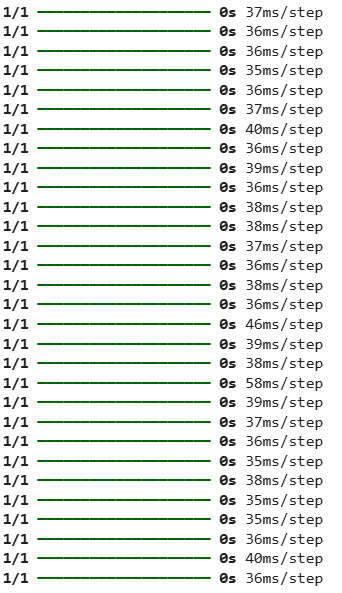
Explanation:

LSTM models in Keras require input in the shape (samples, time\_steps, features). Since it is predicting one sequence, samples = 1, time\_steps is typically 60 or 90, and features = 1. This line reshapes the most recent stock data from the test set (recent\_sequence) into this 3D shape so it can be passed into the model for forecasting. The reshaped current\_input will serve as the initial input for recursive predictions.

| # Start prediction  for \_ in range(n\_future\_days):  pred = model.predict(current\_input)[0][0] # shape: (1,) → scalar  future\_predictions.append(pred)  # Update input: drop oldest, add latest prediction  current\_input = np.append(current\_input[:, 1:, :], [[[pred]]], axis=1) |
| --- |

Output:

Figure 5.29 Forecasting for n days.



Explanation:

This for loop runs for the number of days the user wants to forecast. In each iteration, the model predicts the next stock price using the current\_input sequence, then stores that prediction in the future\_predictions list. The input is updated by removing the oldest time step and appending the new prediction, forming a new sequence for the next iteration. This process allows the model to generate forecasts into the future, even though only the initial sequence is from real data — subsequent inputs are based on previous predictions, a technique called recursive (auto-regressive) prediction

| # Inverse scale predictions  future\_predictions = np.array(future\_predictions).reshape(-1, 1)  future\_predictions\_rescaled = scaler.inverse\_transform(future\_predictions) |
| --- |

Explanation:

After recursively generating future predictions using the LSTM model, the values are still in the scaled format (typically between 0 and 1) due to prior use of MinMaxScaler. This code first reshapes the predictions into a 2D array ((-1, 1)) and then applies scaler.inverse\_transform to convert them back to their original stock price range. This step is essential for making the forecast understandable and comparable to real stock prices.

| # Create future dates for x-axis  last\_date = last\_date = pd.Timestamp.today().normalize() # Use current date  future\_dates = pd.date\_range(start=last\_date + timedelta(days=1), periods=n\_future\_days, freq='B') |
| --- |

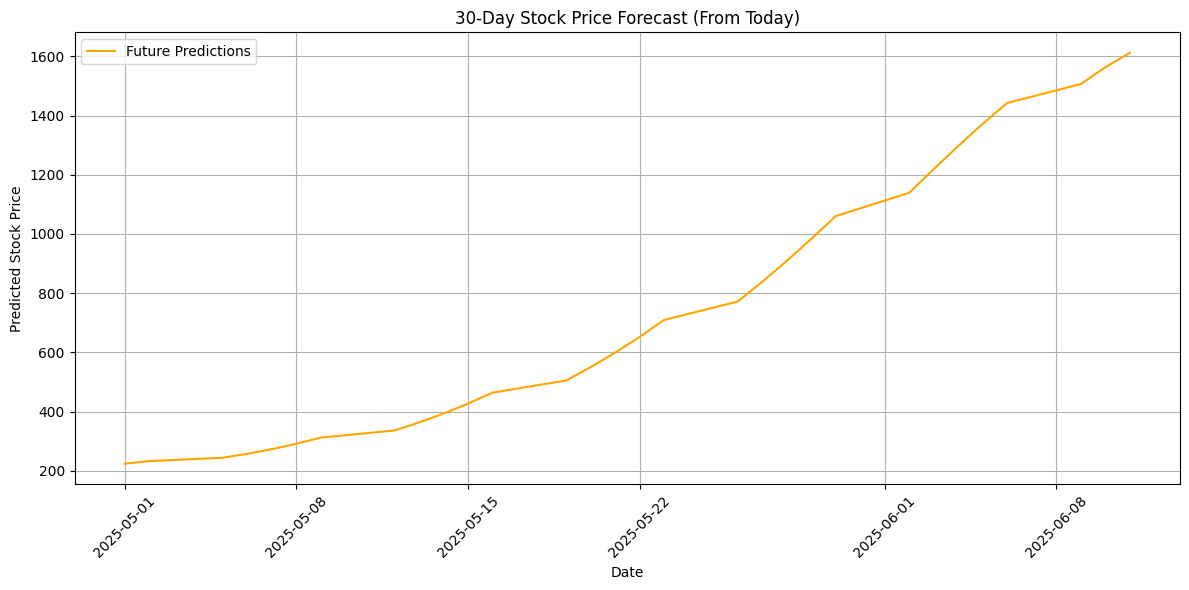
Explanation:

This code creates the x-axis timeline for the predicted stock prices. It starts by getting the current system date, normalized to remove the time component, using pd.Timestamp.today().normalize(). Then, it generates the next n\_future\_days of business days using pd.date\_range. These future dates will correspond to the forecasted values in future\_predictions\_rescaled, allowing users to create a proper time-series forecast chart.

| # Plot future forecast  plt.figure(figsize=(12, 6))  plt.plot(future\_dates, future\_predictions\_rescaled, label='Future Predictions', color='orange')  plt.title(f'{n\_future\_days}-Day Stock Price Forecast (From Today)')  plt.xlabel('Date')  plt.ylabel('Predicted Stock Price')  plt.grid(True)  plt.legend()  plt.xticks(rotation=45)  plt.tight\_layout()  plt.show() |
| --- |

Output:

Figure 5.30 Forecasting plot.



Explanation:

This snippet creates a clear, labeled line chart showing the model's future stock price predictions. It uses matplotlib to plot the future\_predictions\_rescaled against future\_dates . The plot includes a title, axis labels, grid, and rotated x-axis ticks for better readability. This visualizes how the stock price is expected to trend moving forward, based on the last known input and LSTM predictions.

## Benchmarks

| # Reshape data for Random Forest and SVR  X\_train\_rf = X\_train.reshape(X\_train.shape[0], -1)  X\_test\_rf = X\_test.reshape(X\_test.shape[0], -1)  y\_train\_rf = y\_train.ravel()  y\_test\_rf = y\_test.ravel() |
| --- |

Explanation:

Since models like Random Forest and SVR cannot handle 3D input like LSTM, this code reshapes the training and testing input sequences from 3D to 2D using .reshape(X.shape[0], -1), effectively flattening each time-series window into a single row of features. Additionally, y\_train and y\_test are reshaped into 1D arrays using .ravel() to match the expected format for scikit-learn models. This allows users to use the same sequential data for benchmarking with non-deep learning models.

### ARIMA

| # Fit ARIMA once on the entire training set  model\_arima = ARIMA(y\_train, order=(5,1,0)).fit() |
| --- |

Explanation:

This line trains an ARIMA model with order (5, 1, 0) using the y\_train data. Order = (5,1,0) means:

* 5 autoregressive lags (AR),
* differencing once to remove trend (I),
* 0 moving average terms (MA).

| # Forecast the entire test set in one go  predictions\_arima = model\_arima.forecast(steps=len(y\_test)) |
| --- |

Explanation:

After fitting the ARIMA model to the training data, this line generates predictions for the same number of future points as the test set. This produces a series of forecasted stock prices that can be directly compared to the actual test values for evaluation using metrics like RMSE, MAE, or R².

| rmse\_arima = np.sqrt(mean\_squared\_error(y\_test, predictions\_arima)) |
| --- |

Explanation:

This code compares the ARIMA model's forecasted values (predictions\_arima) with the actual test labels using mean\_squared\_error. Then, it takes the square root of that value to compute RMSE.

| # Calculate predictions error metrics  mse\_arima = mean\_squared\_error(y\_test, predictions\_arima)  rmse\_arima = np.sqrt(mse\_arima)  mae\_arima = mean\_absolute\_error(y\_test, predictions\_arima)  r2\_arima = r2\_score(y\_test, predictions\_arima) |
| --- |

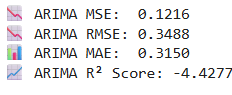
Explanation:

This code evaluates the ARIMA model by computing four key metrics.

| # Display results  print(f"📉 ARIMA MSE: {mse\_arima:.4f}")  print(f"📉 ARIMA RMSE: {rmse\_arima:.4f}")  print(f"📊 ARIMA MAE: {mae\_arima:.4f}")  print(f"📈 ARIMA R² Score: {r2\_arima:.4f}") |
| --- |

Output:

Figure 5.31 ARIMA evaluation metrics.



Explanation:

This code displays the 4 key metrics of the ARIMA model.

### Recurrent Neural Network (RNN)

| # Basic RNN benchmark  rnn = Sequential()  rnn.add(SimpleRNN(50, activation='relu', input\_shape=(X\_train.shape[1], X\_train.shape[2])))  rnn.add(Dense(1))  rnn.compile(optimizer='adam', loss='mse') |
| --- |

Explanation:

This code creates a simple RNN model using Keras's Sequential API. It adds a SimpleRNN layer with 50 units and ReLU activation to learn from sequential data, followed by a single output neuron to predict the next stock price. The model is compiled using the Adam optimizer and Mean Squared Error (MSE) as the loss function. This RNN acts as a baseline deep learning model to compare with more sophisticated architectures like LSTM or GRU.

| rnn.fit(X\_train, y\_train, epochs=20, batch\_size=32, verbose=0) |
| --- |

Explanation:

The rnn.fit() method trains the model on the input sequences X\_train and targets y\_train for 20 epochs using a batch size of 32. Setting verbose=0 suppresses the training output logs for a cleaner notebook interface. During training, the RNN adjusts its internal weights to learn patterns in the historical stock price data in order to make accurate predictions.

| predictions\_rnn = rnn.predict(X\_test) |
| --- |

Output:

Figure 5.32 RNN prediction.



Explanation:

This line generates predicted stock prices using the trained RNN model on test input data.

| # Evaluation metrics  mse\_rnn = mean\_squared\_error(y\_test, predictions\_rnn)  rmse\_rnn = np.sqrt(mse\_rnn)  mae\_rnn = mean\_absolute\_error(y\_test, predictions\_rnn)  r2\_rnn = r2\_score(y\_test, predictions\_rnn) |
| --- |

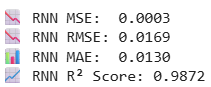
Explanation:

This code evaluates the Recurrent Neural Network model by computing four key metrics.

| # Print results  print(f"📉 RNN MSE: {mse\_rnn:.4f}")  print(f"📉 RNN RMSE: {rmse\_rnn:.4f}")  print(f"📊 RNN MAE: {mae\_rnn:.4f}")  print(f"📈 RNN R² Score: {r2\_rnn:.4f}") |
| --- |

Output:

Figure 5.33 RNN evaluation metrics.



Explanation:

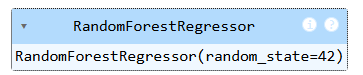
This code displays the 4 key metrics of the Recurrent Neural Network model.

### Random Forest

| # Random Forest benchmark  rf = RandomForestRegressor(n\_estimators=100, random\_state=42)  rf.fit(X\_train\_rf, y\_train\_rf) |
| --- |

Output:

Figure 5.34 Random Forest training.



Explanation:

This code initializes a RandomForestRegressor with 100 decision trees and a fixed random seed for reproducibility. It is then trained on X\_train\_rf and y\_train\_rf, which are reshaped versions of the original training data suitable for scikit-learn models. The Random Forest serves as a non-sequential baseline model to compare against deep learning approaches like LSTM and RNN.

| predictions\_rf = rf.predict(X\_test\_rf) |
| --- |

Explanation:

This line generates predicted stock prices using the Random Forest model on test input data.

| # Compute metrics  mse\_rf = mean\_squared\_error(y\_test\_rf, predictions\_rf)  rmse\_rf = np.sqrt(mse\_rf)  mae\_rf = mean\_absolute\_error(y\_test\_rf, predictions\_rf)  r2\_rf = r2\_score(y\_test\_rf, predictions\_rf) |
| --- |

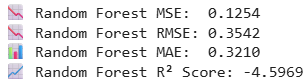
Explanation:

This code evaluates the random Forest model by computing four key metrics.

| # Print results  print(f"📉 Random Forest MSE: {mse\_rf:.4f}")  print(f"📉 Random Forest RMSE: {rmse\_rf:.4f}")  print(f"📊 Random Forest MAE: {mae\_rf:.4f}")  print(f"📈 Random Forest R² Score: {r2\_rf:.4f}") |
| --- |

Output:

Figure 5.35 Random Forest evaluation metrics.



Explanation:

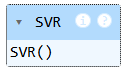
This code displays the 4 key metrics of the Random Forest model.

### Support Vector Machine

| # SVR benchmark  svr = SVR(kernel='rbf')  svr.fit(X\_train\_rf, y\_train\_rf) |
| --- |

Output:

Figure 5.36 SVR training.



Explanation:

Here, an SVR model with a Radial Basis Function (RBF) kernel is initialized and trained using the flattened training data (X\_train\_rf, y\_train\_rf). The RBF kernel allows the SVR to capture non-linear relationships in the data. SVR is a traditional regression method that is effective for small-to-medium-sized datasets and provides a useful benchmark against more complex models like LSTM or RNN.

| predictions\_svr = svr.predict(X\_test\_rf) |
| --- |

Explanation:

This line generates predicted stock prices using the Support vector Machine model on test input data.

| # Compute metrics  mse\_svr = mean\_squared\_error(y\_test\_rf, predictions\_svr)  rmse\_svr = np.sqrt(mse\_svr)  mae\_svr = mean\_absolute\_error(y\_test\_rf, predictions\_svr)  r2\_svr = r2\_score(y\_test\_rf, predictions\_svr) |
| --- |

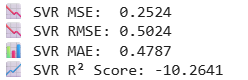
Explanation:

This code evaluates the Support Vector Machine model by computing four key metrics.

| # Print results  print(f"📉 SVR MSE: {mse\_svr:.4f}")  print(f"📉 SVR RMSE: {rmse\_svr:.4f}")  print(f"📊 SVR MAE: {mae\_svr:.4f}")  print(f"📈 SVR R² Score: {r2\_svr:.4f}") |
| --- |

Output:

Figure 5.37 SVR evaluation metrics.



Explanation:

This code displays the 4 key metrics of the Support Vector Machine model.

### Comparison between Models

| # Prepare the data  metrics\_data = {  'Model': ['ARIMA', 'RNN', 'Random Forest', 'SVR'],  'MSE': [mse\_arima, mse\_rnn, mse\_rf, mse\_svr],  'RMSE': [rmse\_arima, rmse\_rnn,rmse\_rf, rmse\_svr],  'MAE': [mae\_arima, mae\_rnn, mae\_rf, mae\_svr],  'R² Score': [r2\_arima, r2\_rnn, r2\_rf, r2\_svr]  } |
| --- |

Explanation:

This dictionary named metrics\_data collects evaluation metrics — MSE, RMSE, MAE, and R² Score — for four models: ARIMA, RNN, Random Forest, and SVR. Each key represents a column, and the corresponding values are lists containing that metric’s result for each model. This dictionary structure makes it easy to convert into a Pandas DataFrame for display or visualization, allowing a clear comparative analysis of model accuracy.

| # Create DataFrame  comparison\_df = pd.DataFrame(metrics\_data) |
| --- |

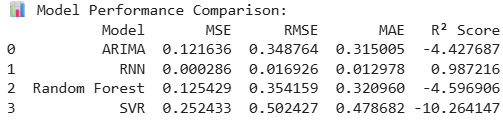
Explanation:

The code uses pd.DataFrame(metrics\_data) to convert the metrics\_data dictionary into a structured table. Each model becomes a row, and each evaluation metric (MSE, RMSE, MAE, R² Score) becomes a column.

| # Display  print("📊 Model Performance Comparison:")  print(comparison\_df) |
| --- |

Output:

Figure 5.38 Benchmark evaluation metrics.



Explanation:

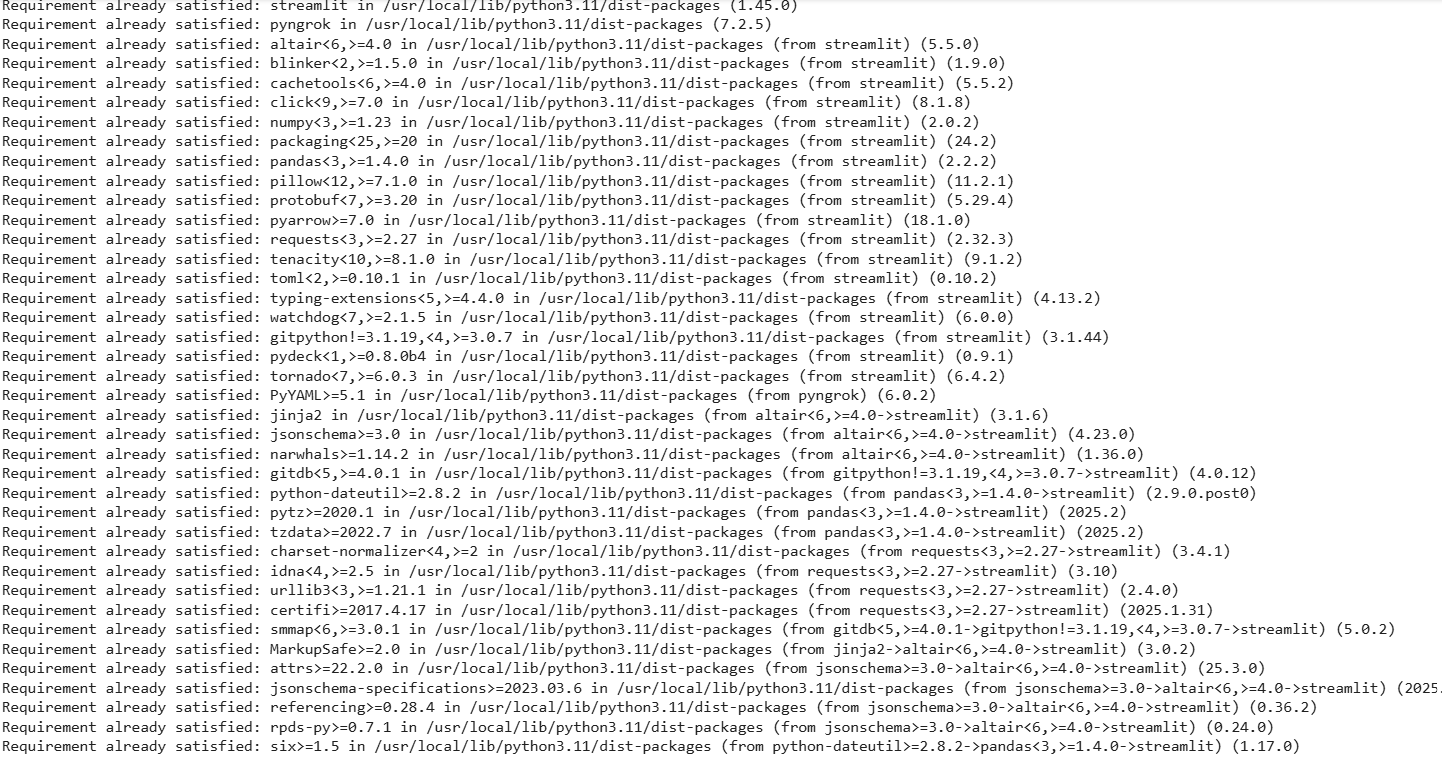
This code displays the structured table created.

## Streamlit (Dashboard)

| !pip install streamlit pyngrok |
| --- |

Output:

Figure 5.39 Streamlit and pyngrok installation.



Explanation:

This code installs the streamlit and pyngrok packages, typically used to run and share interactive web apps directly from environments like Google Colab

| from pyngrok import ngrok  ngrok.set\_auth\_token("2w91e3AJxVksJwxR6iGjEQsyhze\_4nNmpSW4YNxugs29u3DgH") |
| --- |

Explanation:

pyngrok allows users to create secure tunnels to your localhost so they can access apps running on your machine from the internet. ngrok.set\_auth\_token(...) sets a personal authentication token so the user can use the full features of your ngrok account.

| %%writefile app.py  import streamlit as st  import kagglehub  import os  import pandas as pd  import numpy as np  from tensorflow.keras.models import load\_model  from sklearn.preprocessing import MinMaxScaler  import matplotlib.pyplot as plt  import matplotlib.dates as mdates  from datetime import timedelta  # --- Load dataset from KaggleHub ---  file\_path = kagglehub.dataset\_download("nelgiriyewithana/world-stock-prices-daily-updating")  filename = "World-Stock-Prices-Dataset.csv"  file\_path = os.path.join(file\_path, filename)  df = pd.read\_csv(file\_path)  df['Date'] = pd.to\_datetime(df['Date'])  # --- Sidebar: Stock selection & number of future days ---  st.sidebar.title("📈 Stock Prediction Settings")  available\_stocks = sorted(df['Brand\_Name'].unique())  selected\_stock = st.sidebar.selectbox("Choose a Stock", available\_stocks)  n\_future\_days = st.sidebar.slider("Days to Predict", min\_value=30, max\_value=100, value=7)  # --- Filter and prepare data ---  stock\_data = df[df['Brand\_Name'] == selected\_stock].copy()  stock\_data['Date'] = pd.to\_datetime(stock\_data['Date'], utc=True).dt.tz\_localize(None)  stock\_data = stock\_data.sort\_values("Date")  # --- Let user pick historical date range ---  st.sidebar.markdown("### 🔍 View Historical Data")  min\_date = stock\_data['Date'].min().date()  max\_date = stock\_data['Date'].max().date()  start\_hist = st.sidebar.date\_input("From", value=min\_date, min\_value=min\_date, max\_value=max\_date)  end\_hist = st.sidebar.date\_input("To", value=max\_date, min\_value=min\_date, max\_value=max\_date)  # --- Filter historical range ---  historical\_view = stock\_data[(stock\_data['Date'].dt.date >= start\_hist) & (stock\_data['Date'].dt.date <= end\_hist)]  # --- Prepare Close price ---  close\_prices = stock\_data['Close'].values.reshape(-1, 1)  # --- Scaling ---  scaler = MinMaxScaler()  scaled\_data = scaler.fit\_transform(close\_prices)  # --- Create sequence (recent input for forecasting) ---  n\_past = 90 # Number of past days used  time\_steps = n\_past  recent\_sequence = scaled\_data[-time\_steps:] # last 60 days  current\_input = recent\_sequence.reshape(1, time\_steps, 1)  # --- Load the best model ---  model = load\_model('best\_model.h5', compile=False)  # --- Start Recursive Forecasting ---  future\_predictions = []  for \_ in range(n\_future\_days):  pred = model.predict(current\_input)[0][0] # Get the scalar prediction  future\_predictions.append(pred)  # Update input by dropping oldest and appending latest prediction  current\_input = np.append(current\_input[:, 1:, :], [[[pred]]], axis=1)  # --- Inverse scale the predictions ---  future\_predictions = np.array(future\_predictions).reshape(-1, 1)  future\_predictions\_rescaled = scaler.inverse\_transform(future\_predictions).flatten()  # --- Generate Future Dates ---  last\_date = pd.Timestamp.today().normalize()  future\_dates = pd.date\_range(start=last\_date + timedelta(days=1), periods=n\_future\_days, freq='B')  # --- Plot Historical Graph ---  st.title("📊 Historical Stock Prices")  st.subheader(f"{selected\_stock} from {start\_hist} to {end\_hist}")  fig\_hist, ax\_hist = plt.subplots(figsize=(12, 5))  ax\_hist.plot(historical\_view['Date'], historical\_view['Close'], color='blue', label='Historical Close Price')  ax\_hist.set\_title(f"{selected\_stock} Historical Prices")  ax\_hist.set\_xlabel("Date")  ax\_hist.set\_ylabel("Close Price")  ax\_hist.grid(True)  ax\_hist.legend()  plt.xticks(rotation=45)  st.pyplot(fig\_hist)  # Allow CSV download  csv\_hist = historical\_view.to\_csv(index=False)  st.download\_button(  label="📥 Download Historical Data as CSV",  data=csv\_hist,  file\_name=f"{selected\_stock}\_historical\_{start\_hist}\_to\_{end\_hist}.csv",  mime='text/csv'  )  # --- Plot Forecast ---  st.title("📈 Future Stock Price Forecast")  st.subheader(f"Stock: {selected\_stock} | Days to Forecast: {n\_future\_days}")  fig, ax = plt.subplots(figsize=(12, 6))  ax.plot(future\_dates, future\_predictions\_rescaled, label='Future Predictions', color='red')  ax.set\_title(f'{n\_future\_days}-Day Stock Price Forecast (Starting from Today)')  ax.set\_xlabel('Date')  ax.set\_ylabel('Predicted Close Price')  ax.legend()  ax.grid(True)  plt.xticks(rotation=45)  plt.tight\_layout()  st.pyplot(fig)  # Optionally Show Predicted Values  if st.checkbox("Show Predicted Values as Table (Recommended)"):  pred\_df = pd.DataFrame({  'Date': future\_dates,  'Predicted Close Price': future\_predictions\_rescaled  })  st.dataframe(pred\_df) |
| --- |

Explanation:

The provided Streamlit app.py script builds an interactive web interface for forecasting stock prices using a trained LSTM model. It loads stock data from KaggleHub, allows users to select a company , specify a forecast horizon, and view historical stock prices within a chosen date range. The script scales the data, performs recursive future predictions using the LSTM model, and then plots both historical and predicted prices. Users can also download historical data as a CSV file and view forecasted values in a tabular format.

| # Create a public URL for the Streamlit app  public\_url = ngrok.connect(8501)  print("🌍 Public URL:", public\_url) |
| --- |

Output:

Figure 5.40 Streamlit URL.



Explanation:

This line of code uses pyngrok to expose local Streamlit app to the internet by creating a public URL via Ngrok.

| # Run the Streamlit app in the background  !pkill -f streamlit  !streamlit run app.py &>/content/log.txt |
| --- |

Explanation:

This command launches the Streamlit app in the background and kills any existing background Streamlit processes to prevent port conflicts.

## Test Cases

### TC01 Stock Selection

Input: Select "TESLA" from dropdown.

Output: Forecast and historical graph for TESLA appears.

Figure 5.41 TC01.

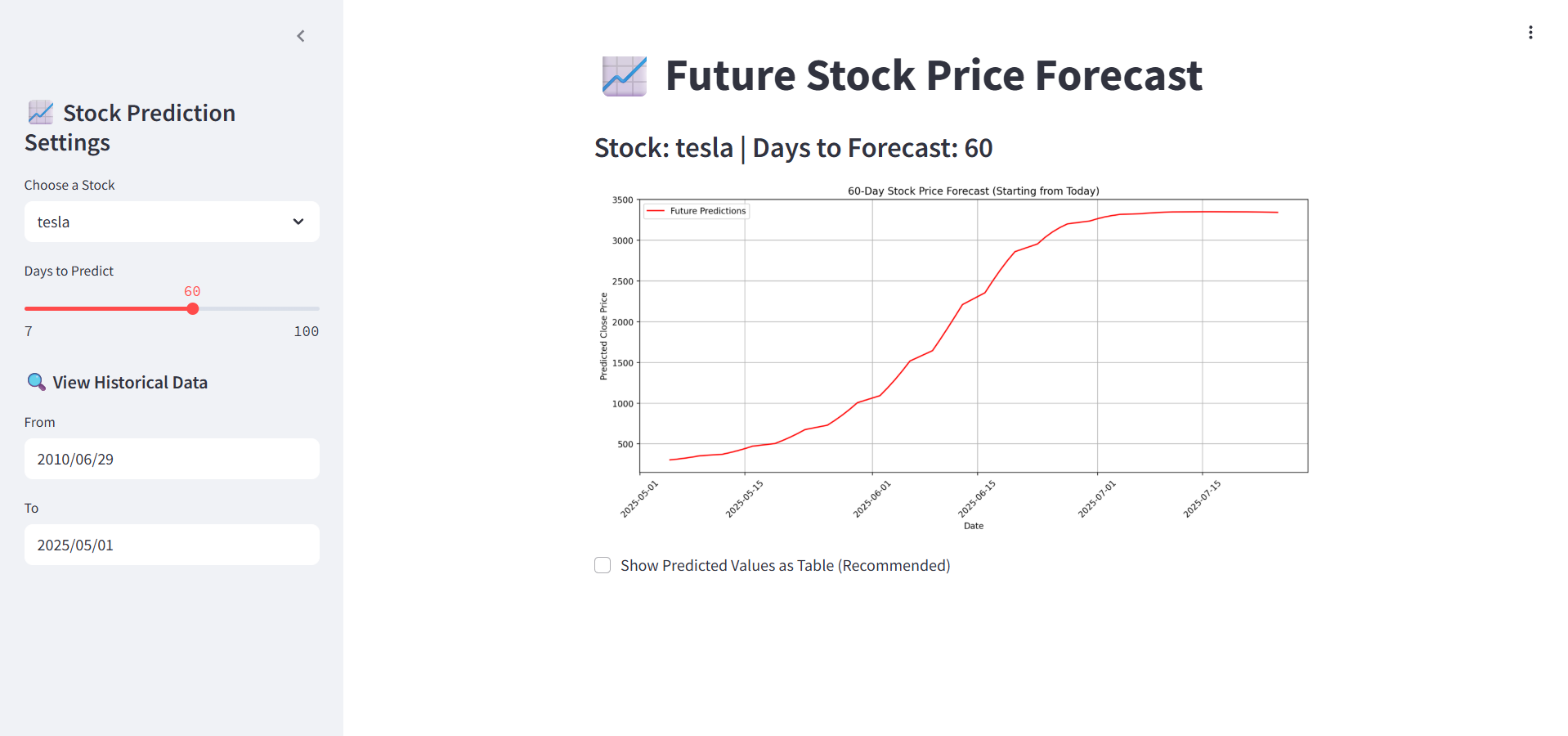


### TC02 Prediction Generation

Input: Select 60 days to predict.

Output: 60 future days plotted in red line.

Figure 5.42 TC02.



### TC03 Date Filter (Historical)

Input: From: 2020-01-01 To: 2020-12-31

Output: Only 2020 data plotted in blue line.

Figure 5.43 TC03.



### TC04 Data Download (Historical)

Input: Click “📥 Download CSV”

Output: Downloads historical data file.

Figure 5.44 TC04.

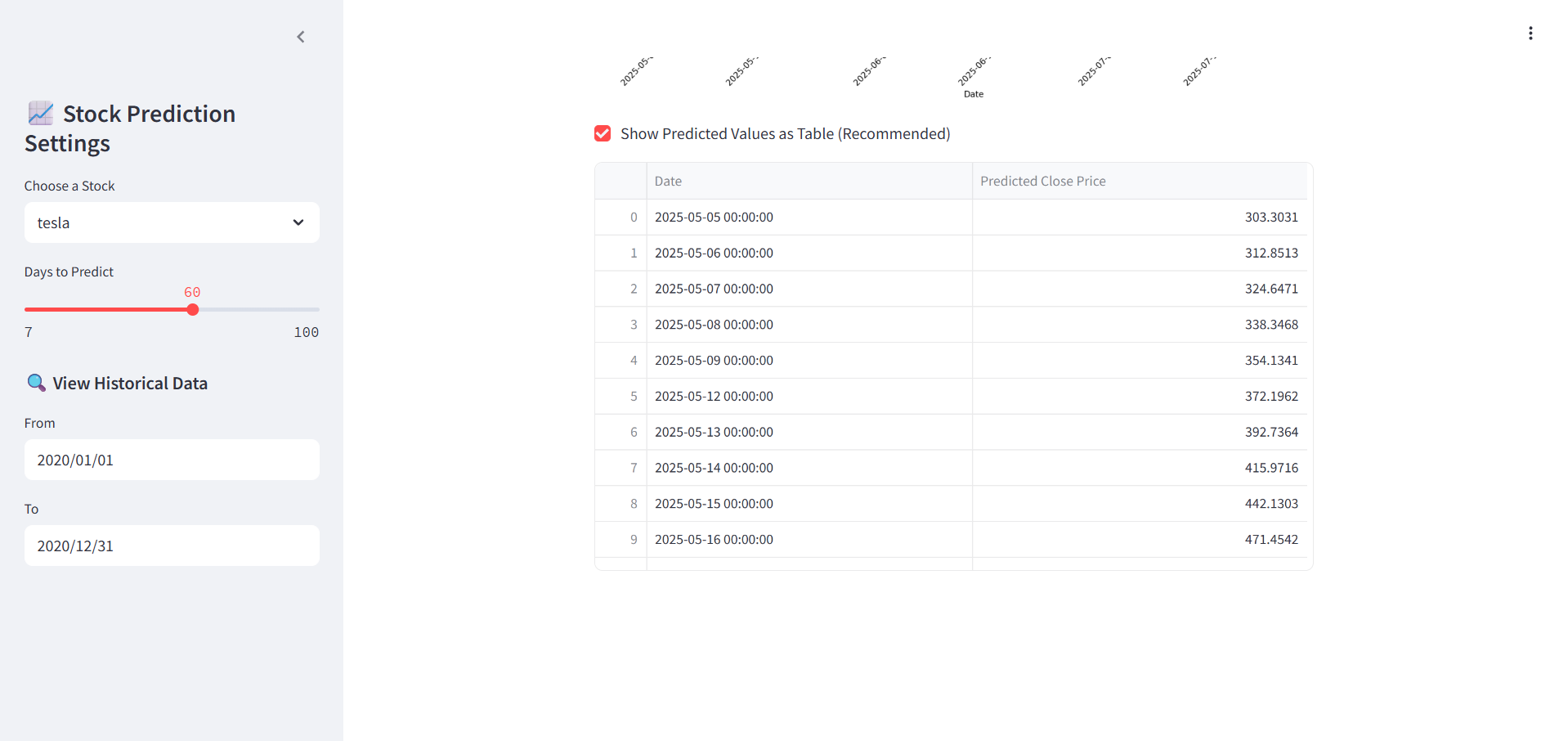


### TC05 Table Toggle

Input: Tick the “Show Prediction Data as Table (Recommended)”.

Output: Prediction data table appears.

Figure 5.45 TC05.



## Chapter Summary and Evaluation

This chapter outlines the complete implementation and testing of a stock price prediction system developed using LSTM (Long Short-Term Memory) networks. The implementation section details the step-by-step development of the model, including data preprocessing, sequence generation, model building, hyperparameter tuning using Keras Tuner, and training the LSTM model with early stopping to prevent overfitting. A recursive forecasting approach was used to predict future stock prices, and the results were visualized using Matplotlib. Evaluation metrics like MSE, RMSE, MAE, and R² Score were used for evaluation. The testing section focuses on validating the Streamlit-based interface. Users could interactively select a stock brand, choose a forecasting window, view historical trends for a custom date range, and download data. Test cases were executed to verify functionality, input validation, graphical output, and data export. All features worked as intended, confirming the usability and reliability of the system.

Chapter 6

**Discussions and Conclusion**

# Discussions and Conclusion

This chapter summarizes the overall project. Achievements, contributions, limitations, future improvements, issues and their solutions and unresolved issues are discussed in this chapter to conclude my final year project.

## Summary

This project set out to develop a stock price prediction system that leverages deep learning, specifically Long Short-Term Memory (LSTM) networks to forecast future stock prices based on historical data. The motivation behind the project stems from the complexity and volatility of financial markets, where traditional linear models often fall short in capturing non-linear and sequential patterns. To address this issue, the system utilizes an LSTM-based approach trained on real-world stock data sourced dynamically from KaggleHub. The project’s goal was not only to build an accurate predictive model but also to present the forecasts through an intuitive and interactive user interface. For this purpose, the Streamlit web framework was integrated to develop a lightweight and visually responsive application that allows users to choose their desired stock, define the number of days to forecast, and view both historical trends and predicted prices.

## Achievements

The stock price prediction system developed for this project marks a significant achievement both in terms of technical execution and practical application. One of the most notable accomplishments is the successful implementation of a Long Short-Term Memory (LSTM) model for time series forecasting. LSTM, as a deep learning architecture, is known for its ability to capture long-term dependencies in sequential data. By training on historical stock data, the model was able to learn intricate patterns and deliver reasonably accurate short-term forecasts. The model was tuned using Keras Tuner, with hyperparameters such as the number of LSTM units, learning rate, and dropout rate optimized to improve predictive performance. The final model achieved favorable metrics such as low RMSE and MAE values, indicating that it could generalize well to unseen data.

Another significant achievement was the integration of the model into a fully functional Streamlit-based web application. This user-friendly interface enables users to select any stock from the dataset, define a custom prediction window , and view both historical trends and future forecasts interactively. The system also allows users to download historical data in CSV format, further enhancing its utility. The use of interactive visualizations, such as dynamic date range filtering and real-time plotting with Matplotlib, makes the system accessible even to users without a technical background.

Furthermore, the project achieved successful benchmarking against traditional models including ARIMA, RNN, Random Forest, and Support Vector Regression (SVR). Comparative metrics such as RMSE, MAE, and R² score were used to evaluate each model’s performance, and the LSTM model consistently outperformed the classical approaches. This benchmarking not only validated the effectiveness of the LSTM implementation but also provided a robust foundation for future work and improvements.

The project also demonstrated a high degree of data handling proficiency. The raw dataset consisted of thousands of daily records from multiple global stock markets, and extensive preprocessing steps were required to normalize date formats, handle missing values, scale features appropriately, and ensure sequence alignment for training. By solving these challenges, the system was able to reliably process diverse stock data for forecasting.

While integration with Firebase and Power BI was not fully realized due to technical limitations, the decision to pivot to Streamlit enabled the project to maintain its objective of building an end-to-end system that is both accurate and user-centric. The final deployed version is functional, interactive, and deployable through tunneling services like Ngrok, making it suitable for demonstration and further extension.

Overall, the project successfully met its key objectives: to build an accurate stock prediction model, to offer an intuitive and interactive visualization platform, and to support decision-making for investors and analysts. The experience gained and the working prototype created represent a meaningful academic and technical achievement.

## Contributions

This project contributes significantly to the field of financial technology by demonstrating how deep learning, specifically LSTM-based models, can be applied to stock price forecasting in a practical and user-centric way. The main innovation lies not just in the model itself, but in the full-stack integration of machine learning with an interactive and intuitive Streamlit-based web interface. By enabling users to visualize both historical data and future predictions in real time, the system makes advanced AI-driven insights accessible even to non-technical users such as retail investors and financial enthusiasts.

One of the notable contributions of this project is its ability to bridge the gap between raw predictive models and actual usability. Many academic models focus solely on accuracy, overlooking the need for interpretability and real-world interaction. This project fills that gap by offering customizable forecasting (user-defined date ranges and prediction lengths), downloadable reports, and visual plots that enhance transparency and user control. Additionally, by incorporating Keras Tuner for hyperparameter optimization, the project contributes an automated and reproducible way to improve LSTM model performance, saving time and reducing manual experimentation.

Although Firebase and Power BI integration could not be completed, their intended inclusion demonstrates a clear vision for extensibility and real-world deployment. The planned use of Firebase for user data storage and Power BI for dynamic analytics underscores the project's forward-thinking approach and market readiness. Even without these features, the modular design ensures that the application can be scaled or enhanced with minimal restructuring, making it viable for future deployment as a commercial or educational tool.

In summary, the project is a strong example of how machine learning, cloud data, and user-centered interface design can come together to solve complex forecasting problems in the financial domain. Its contributions lie in both technical implementation and practical application, offering a scalable blueprint for real-time, AI-powered decision support systems in stock trading and beyond.

## Limitations and Future Improvements

While this project successfully delivered a functional stock price prediction system using LSTM and a Streamlit-based interface, it is not without its limitations. One of the primary constraints lies in the lack of real-time data integration. The model relies on static, pre-downloaded datasets rather than fetching the latest stock prices in real time. This restricts the system’s use in live market conditions and limits its applicability for users who need up-to-the-minute predictions. Integrating a live stock API, such as Alpha Vantage or Yahoo Finance, in future versions would allow for real-time prediction and improve the system’s relevance and responsiveness.

From a modeling perspective, the LSTM model, while effective, still has room for enhancement. It does not currently incorporate external influencing factors such as financial news sentiment, macroeconomic indicators, or trading volume, which are known to affect stock prices. Adding such multi-feature inputs could improve prediction accuracy. Additionally, experimenting with hybrid models that combine LSTM with attention mechanisms or transformer architectures might yield even more robust forecasting performance.

Another limitation concerns the evaluation scope. The current evaluation is based primarily on historical test sets and common metrics like RMSE and MAE. It lacks user-centered evaluation or feedback from potential investors or financial analysts who could assess the system’s usefulness from a practical standpoint. Incorporating qualitative feedback or a usability study would help validate the app’s real-world effectiveness.

Lastly, the user interface, while functional, can be further enhanced. Features such as multi-stock comparison, custom model selection, confidence intervals around predictions, and mobile responsiveness are all areas that could elevate the user experience.

In conclusion, although the project met its core goals, several areas remain for enhancement. Future development should focus on real-time integration, multi-source data, advanced visualization, and broader user interaction to create a fully mature and deployable financial forecasting tool.

## Issues and Solutions

One of the significant challenges involves handling time series data for multiple stock tickers with different time ranges and missing values. Inconsistent data formatting, especially with the date column and null entries, caused early errors in model training and visualization. This was resolved by applying robust preprocessing techniques including pd.to\_datetime() normalization, date filtering, and MinMaxScaler transformation to ensure consistent input for LSTM models. Careful sorting and filtering by Brand\_Name and Ticker also ensured that the correct sequences were generated for prediction.

During model training, hyperparameter tuning using Keras Tuner introduced another layer of complexity. Extensive training trials, limited computational resources, and the need to balance performance with generalization were all practical difficulties. To manage this, the search space was kept narrow but meaningful, and early stopping was applied to prevent overfitting and reduce training time. The tuning framework ultimately helped in improving the model’s RMSE and MAE scores.

Integration with Streamlit posed its own challenges, particularly in managing date input types, displaying time series plots dynamically, and enabling downloads for CSV outputs. Streamlit’s interactivity also required ensuring that date pickers worked with timezone-aware and naive datetime formats, which led to several rounds of debugging. Errors related to timezone mismatches were solved by explicitly localizing or de-localizing dates using tz\_localize(None) in Pandas, while errors related to plotting were addressed by converting date inputs to compatible datetime64 formats for Matplotlib.

Lastly, there were deployment challenges when trying to demonstrate the app on Google Colab, which does not natively support Streamlit apps with direct user interfaces. To solve this, Ngrok tunneling was used to expose the Streamlit app to the internet securely. This required extra setup such as configuring auth tokens, managing tunnel lifecycle, and redirecting output logs properly to avoid crashes.

Despite these issues, each challenge served as a valuable learning experience. Debugging time series data, understanding model behavior under different hyperparameter settings, and resolving deployment bottlenecks helped develop deeper technical maturity. The experience also highlighted the importance of flexibility in project scope—especially when dealing with real-world constraints like integration failures and platform limitations.

## Unresolved Issues

However, not all proposed components could be realized within the scope and timeline of this project. Integration with Firebase, which was intended to store user interactions, prediction logs, and enable cross-device data access, was not implemented due to authentication issues and restrictions when using Google Colab as a development environment. Similarly, the planned integration with Power BI for business-grade data visualization was dropped after facing challenges in establishing real-time pipelines between the prediction model and Power BI dashboards. These exclusions did not affect the core functionality of the system but do represent areas for potential future development.

# References

1. Alex Sherstinky. (2020, March). *Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network*. <https://www.sciencedirect.com/science/article/abs/pii/S0167278919305974>
2. Qiu, J., Wang, B., & Zhou, C. (2020). Forecasting stock prices with long-short term memory neural network based on attention mechanism. *PLoS ONE*, *15*(1), e0227222. <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0227222>
3. Budiharto, W. (2021). Data science approach to stock prices forecasting in Indonesia during Covid-19 using Long Short-Term Memory (LSTM). *Journal of Big Data*, *8*(1). <https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00430-0>
4. Akhavanpour, M. (n.d.). *Adaptive Model Selection in Stock Market Prediction: A Modular and Scalable Big Data Analytics Approach - ProQuest*. <https://www.proquest.com/openview/12500cbafe3f5a870f25653ea04c0787/1?pq-origsite=gscholar&cbl=18750&diss=y>
5. G. Peter Zhang. (2003, January). *Time series forecasting using a hybrid ARIMA and neural network model*. <https://www.sciencedirect.com/science/article/abs/pii/S0925231201007020>
6. Elsaraiti, M., & Merabet, A. (2021). A Comparative Analysis of the ARIMA and LSTM Predictive Models and Their Effectiveness for Predicting Wind Speed. *Energies*, *14*(20), 6782. <https://www.mdpi.com/1996-1073/14/20/6782>
7. Ruan Luzia, Lihki Rubio & Carlos E.Velasquez. (2023, July 1). *Sensitivity forecasting using Brazilian electricity demand using artificial neural networks and hybrid models based on Autoregressive Integrated Moving Average*. <https://www.sciencedirect.com/science/article/abs/pii/S0360544223007594>
8. Ospina, R., Gondim, J. a. M., Leiva, V., & Castro, C. (2023). An Overview of Forecast Analysis with ARIMA Models during the COVID-19 Pandemic: Methodology and Case Study in Brazil. *Mathematics*, *11*(14), 3069. <https://www.mdpi.com/2227-7390/11/14/3069>
9. Syed Fawad Hussain. (2019, October 1). *A novel robust kernel for classifying high-dimensional data using Support Vector Machines*. <https://www.sciencedirect.com/science/article/abs/pii/S0957417419302696>
10. Sruthi. (2024, December 11). *Understanding Random Forest Algorithm With Examples*. Analytics Vidhya. <https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/>
11. Or, B. (2023, December 14). The Exploding and Vanishing Gradients Problem in Time Series. *Medium*. <https://medium.com/metaor-artificial-intelligence/the-exploding-and-vanishing-gradients-problem-in-time-series-6b87d558d22>
12. Anishnama. (2023, April 28). Understanding LSTM: Architecture, Pros and Cons, and Implementation. *Medium*. <https://medium.com/@anishnama20/understanding-lstm-architecture-pros-and-cons-and-implementation-3e0cca194094>
13. Thomas Fischer & Christopher Krauss. (2017, December). *Deep learning with long short-term memory networks for financial market predictions*. <https://www.researchgate.net/publication/321630147_Deep_learning_with_long_short-term_memory_networks_for_financial_market_predictions>
14. Gede Susrama, Agung Mustika & Nurkholis Amanullah. (2024, May). *Long Short Term Memory Method and Social Media Sentiment Analysis for Stock Price Prediction*. <https://www.researchgate.net/publication/383041737_Long_Short_Term_Memory_Method_and_Social_Media_Sentiment_Analysis_for_Stock_Price_Prediction>
15. Kai Liu & Jie Zhang. (2021). *Long Short-Term Memory Network*. <https://www.sciencedirect.com/topics/computer-science/long-short-term-memory-networks>
16. Zhang, G., & Zhang, H. (2018). Long-short-term memory network based hybrid model for short-term electrical load forecasting. *Information, 9*(7), 165. <https://www.mdpi.com/2078-2489/9/7/165>
17. Malashin, I., Tynchenko, V., Gantimurov, A., Nelyub, V., & Borodulin, A. (2024). Applications of Long Short-Term Memory (LSTM) Networks in Polymeric Sciences: A Review. *Polymers, 16*(18), 2607. <https://www.mdpi.com/2073-4360/16/18/2607>
18. Banoula, M. (2023, April 27). *Introduction to Long Short-Term Memory(LSTM)*. Simplilearn.com. <https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/lstm#:~:text=The%20LSTM%20cell%20also%20has,data%20over%20multiple%20time%20steps>
19. Donges, N. (2024, October 17). *A Guide to Recurrent Neural Networks (RNNs)*. Built In. <https://builtin.com/data-science/recurrent-neural-networks-and-lstm#:~:text=Complex%20training%20process%3A%20Because%20RNNs,work%20to%20remember%20past%20information>
20. Srivatsavaya, P. (2023, October 5). LSTM — Implementation, Advantages and Diadvantages - Prudhviraju Srivatsavaya - Medium. *Medium*. <https://medium.com/@prudhviraju.srivatsavaya/lstm-implementation-advantages-and-diadvantages-914a96fa0acb>
21. Muhammad Waqas & Usa Wannasingha Humphries. (2024, December). *A critical view of RNN and LSTM variants in hydrological time series predictions*. <https://www.sciencedirect.com/science/article/pii/S2215016124003972>
22. Zaina Saadeddin. (2024, September 9). *ARIMA for Time Series Forecasting: A Complete Guide*. <https://www.datacamp.com/tutorial/arima>
23. Yasar, K., & Tabsharani, F. (2024, November 25). *What is a support vector machine (SVM)?* WhatIs. <https://www.techtarget.com/whatis/definition/support-vector-machine-SVM#:~:text=A%20support%20vector%20machine%20(SVM)%20is%20a%20type%20of%20supervised,data%20set%20into%20two%20groups>
24. <https://www.sciencedirect.com/science/article/abs/pii/S0925231220307153>
25. EDC Paris Business School. (2024, September 24). How random forest works in data science ? *EDC Paris Business School*. <https://www.edcparis.edu/en/school-news/how-random-forest-works-data-science#:~:text=To%20sum%20up%2C%20in%20data,the%20face%20of%20data%20hazards>
26. *Imaginovation | Top Web & Mobile App Development Company Raleigh*. (n.d.). Imaginovation | Top Web & Mobile App Development Company Raleigh. <https://imaginovation.net/technologies/firebase-development-services/>
27. Robinson, S., Brush, K., & Silverthorne, V. (2024, October 18). *What is Agile software development?* Search Software Quality. <https://www.techtarget.com/searchsoftwarequality/definition/agile-software-development>

# Appendices

## System Requirements

Before using the Streamlit-based stock prediction application, ensure the following:

* Python 3.7 or higher is installed
* Required Python libraries are installed: streamlit, tensorflow, pandas, matplotlib, scikit-learn, kagglehub, etc.
* best\_model.h5 is placed in the same directory as app.py.

## Step-by-Step Instructions

### Step 1: Launch the Application

* Open a terminal (or Colab with ngrok setup).
* Run the Streamlit app by executing the following command:

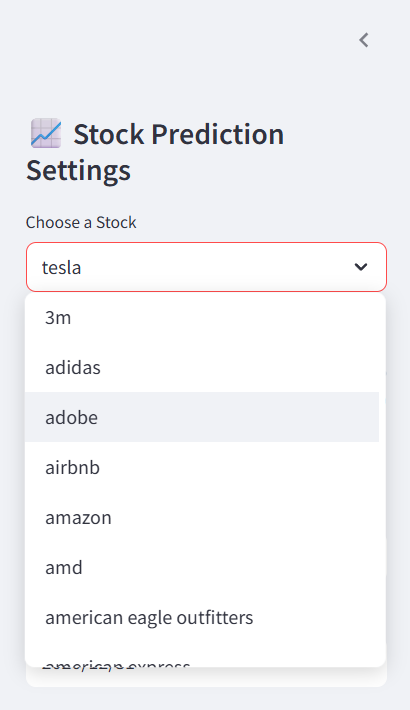
| streamlit run app.py |
| --- |

* If running in Google Colab, start ngrok to expose port 8501:

| from pyngrok import ngrok  public\_url = ngrok.connect(8501)  print(public\_url) |
| --- |

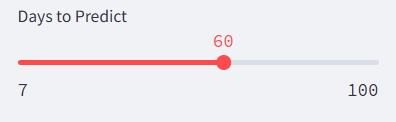
* Open the provided public URL in a browser.

### Step 2: Choose a Stock



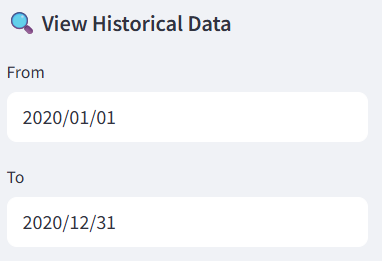
* On the left sidebar under "📈 Stock Prediction Settings", click the dropdown menu.
* Select a stock by its Brand Name (e.g., "APPLE", "TESLA").
* All brand names are sorted alphabetically and displayed in capital letters for easier navigation.

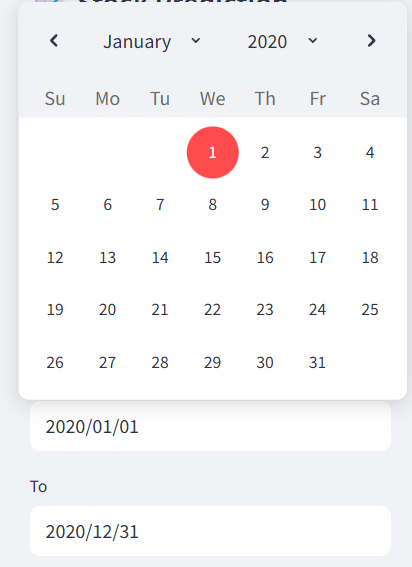
### Step 3: Set Prediction Range



* Use the slider below the stock selection to choose how many future days you want to predict.
* The range is from 30 to 100 days.
* Example: Select 60 to forecast the next 60 business days of stock prices.

### Step 4: View Historical Data





* Scroll further in the sidebar to the section **"🔍 View Historical Data"**.
* Use the **From** and **To** date pickers to define a historical date range.
* The selected range filters the historical data shown in the chart and table.

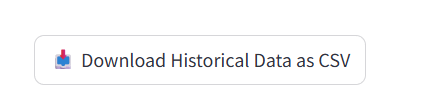
### Step 5: Explore Historical Chart



* This chart shows the stock's actual closing price during your selected date window.
* Use this to analyze past trends before viewing predictions.

### Step 6: Download Historical Data

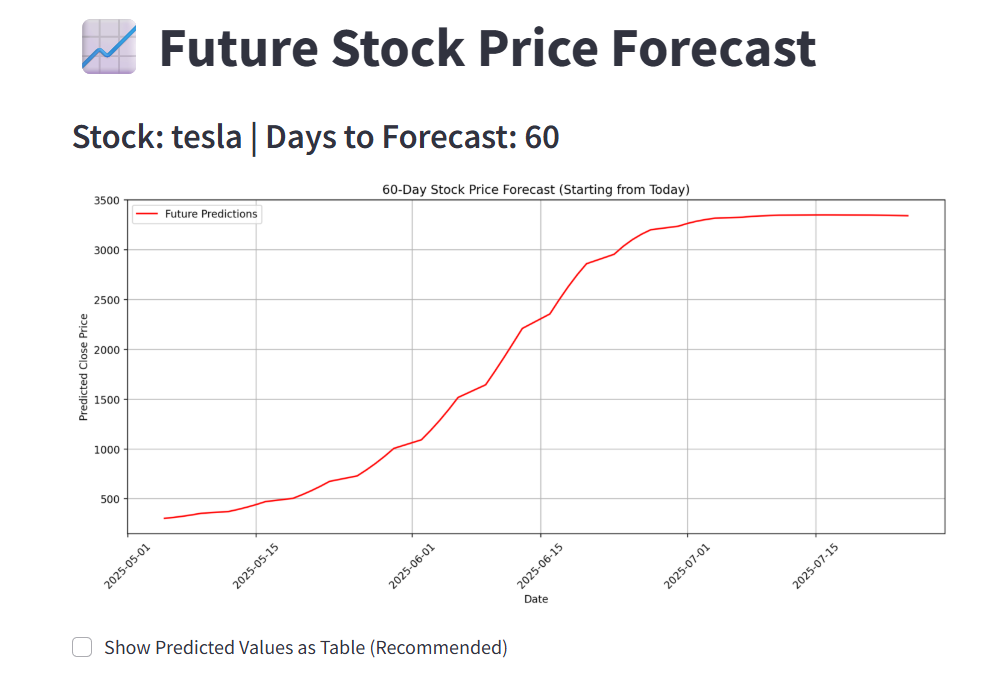
Below the historical chart, click:



* A .csv file containing filtered historical data will be downloaded to your device.

### Step 7: View Predicted Future Prices

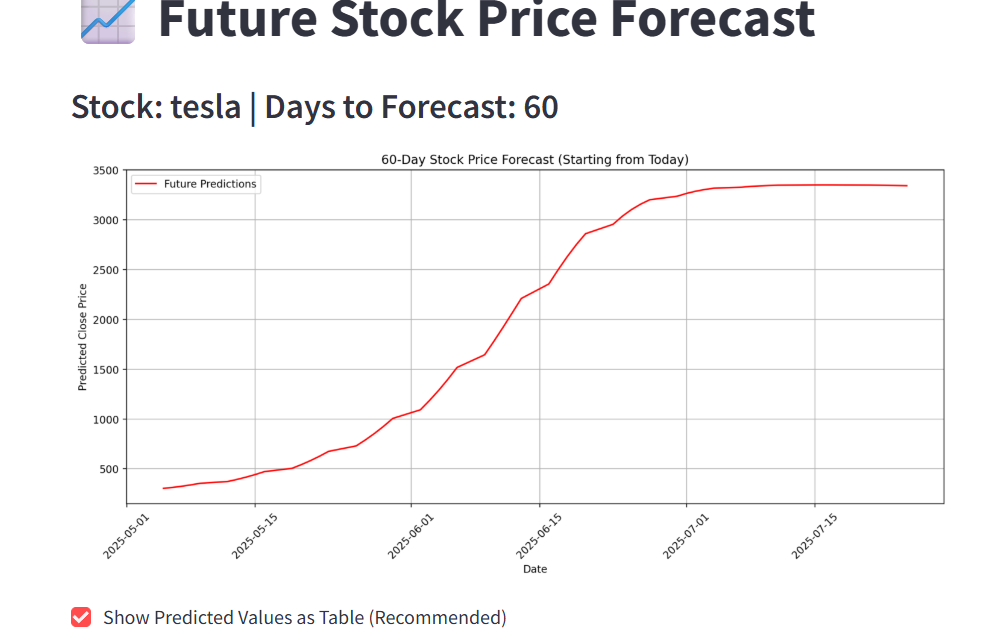
Scroll to the section:



* This line chart displays the forecasted stock prices for the next *N* days (as per your slider).
* Red lines represent predicted prices generated by your trained LSTM model.

### Step 8: View Forecast in Table Format (Optional)

* Tick the checkbox:



* A table will appear showing  
  
* You can scroll and analyze the forecasted values easily.

## Note: Real-Time

* The predictions are computed live each time the user selects a different stock or forecast duration.

## Closing the App

* Press CTRL+C in the terminal to stop the Streamlit server.
* If using Google Colab, shut down the session or kill the process.

This page is intentionally left blank to indicate the back cover. Ensure that the back cover is black in color.