SHRI VAISHNAV INSTITUTE OF MANAGEMENT INDORE(M.P.)



MAJOR RESEARCH PROJECT

Session: 2022-2024

“**Design and Develop an Optimised Framework for Time Series Data Prediction using Deep Learning”**

Submitted to Devi Ahilya Vishwavidyalaya, Indore(M.P.) In partial Fulfillment of the degree of

**Master of Business Administration**

(FULL TIME)

Guided By: Submitted By:

Dr. MAMTA JOSHI CHETAN MALVIYA

MBA (III) SEM

DX1801714

# **PREFACE**

The bookish knowledge of any program, which we get from educational institutions, is not enough to be used in our day-to-day life. The more practical knowledge we have the more beneficial it is for our learning.

To make the student aware from of the working of the business world every student of MASTER OF BUSINESS ADMINISTRATION has to undergo major research where he/she experiences many aspects of business under the supervision of Professional Managers.

I strongly believe that the knowledge gained from these experiences is more than the knowledge gained from the theories in the books.

PLACE : INDORE

Student’s Name: Chetan Malviya

## STUDENT DECLARATION

I Chetan Malviya, Student of Shri Vaishnav Institute of Management Indore of MBA (Full Time) program has prepared Major Research Project Report on the topic **“Design and Develop an Optimised Framework for Time Series Data Prediction using Deep Learning”**. The research as per my knowledge is original and genuine and not published any research journal previously.

CHETAN MALVIYA

MBA (III) SEM

(2022 – 2024)

# **INSTITUTE CERTIFICATE**

“This is to certify that this Comprehensive Project Report Titled **“Design and Develop an Optimised Framework for Time Series Data Prediction using Deep Learning”** is the bonafide work of Chetan Malviya. who have carried out their project under my supervision. I also certify further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate. I have also checked the practical aspects of the project in relevant market”.

## SUBMITTED TO:

## Dr. MAMTA JOSHI

**Professor of the Management Department (SVIM)**

**INTERNAL EXAMINER EXTERNAL EXAMINER**

# **ACKNOWLEDGEMENT**

It is our proud privilege to release the feeling of our gratitude to several people who helped us directly or indirectly to conduct this research project work. we express our heart full indebtedness and owe deep sense of gratitude to our college **“SHRI VAISHNAV INSTITUTE OF MANAGEMENT”**

I am highly thankful to **Dr. Mamta Joshi** for allowing me as a guide and trainer. In addition to helping us in our practical studies at the end study the entire organization and various aspects of managerial functions. They provide to us many details and enlighten me in preparation of this project.

We would like to express my gratitude towards **DR. GEORGE THOMAS** Director at Shri Vaishnav Institute of Management for their valuable guidance and help in the preparation of this report. They were also helping me for using various statistical tools and analysis of data. At last, but not least, we would be thankful to our friends and other people, who helped me in preparation of this project report.

## TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | Content | Page No. |
| 1 | Introduction   * Introduction of Equity Market * Equity Research prediction Techniques * Framework for Time Series Data Prediction | 3-9 |
| 2 | Review of Literature   * Introduction Review of Literature * Earlier Review of Literature * Table Summary | 10-14 |
| 3 | Research Methodology   * Research Methodology * Objective of Research Methodology | 15-19 |
| 4 | * Data analysis, model interpretation and Hypothesis testing | 20-62 |
|  |  |  |

**CHAPTER-1**

**Introduction**

**1.1 Introduction of Equity Market**

The equity market refers to the platform where companies issue shares for sale and investors buy them, helping the former raise capital for enhancing business growth and opportunities. It gives the investors a proportionate stake in the company; hence, they gain ownership of the firm through the shares they purchase.

**2.2 How Does Equity Market Work**

The equity market connects the buyers and sellers having the same price expectations. As soon as the companies establish themselves, they become a private player in the corporate sector. Then, they issue shares as an Initial Public Offering

(IPO). This makes their assets accessible to the public investors, who keep a check on the stock exchanges for the best stock deals for the day.

In their nascent stage, the companies, through this equity or stock market, get an opportunity to raise funds and utilize them to grow their business. On the other hand, the investors who provide those funds receive a stake in the firms, enjoying an ownership right simultaneously. As time passes, they realize the firms’ potential to grow and reap profit. Hence, it becomes easier for these angel investors to decide whether they would like to continue the collaboration and increase their share in the company.

When companies enter the global equity market and list their shares through an IPO, they get listed on the stock exchanges

. Then, based on the budgets, the buyers and sellers connect. In short, the trade takes place only when the price requirements match. The buying and selling of the stocks occur either through the stock exchanges, which are a centralized platform, or an OTC market where a broker matches the deals based on the price the buyer want to buy and the seller wants to sell the shares.

**1.2 Types of Equity market**

The equity market is classified into two categories

**1.3 Primary Market**

Also referred to as the issue market, this is the place where companies introduce or issue their equity market shares for the very first time. The companies can publicly give the shares as an IPO or through a Follow on Public offe (FPO). They can also consider rights issues, allowing their existing shareholders to maintain their earlier ratio in the shares and sell securities at a price lower than the current market prices.

**1.4 Secondary Market**

As soon as the securities are put on sale after their first time, they are said to enter the secondary market, where they move from the hands of one investor to another. The securities of semi-government bodies, government organizations, joint-stock companies , etc. trade under this market category. It is for the investors who are into daily trading.

**1.5 Features**

The equity market makes companies allow investors to enjoy ownership in exchange for capital. As a result, the former get financial benefits, and the latter has ownership rights. Besides that, the stock market also helps raise capital for companies, and enhance liquidity and investment opportunities to boost the economy.

When the companies are budding, they issue shares for interested investors to buy. These investments from various sources help build their capital, thereby letting them think about the further growth of the business. The investors’ decision is completely based on the business idea one has. If the concept is convincing, the investors are ready to invest in them to check if there is room for potential growth.

**1.6 Advantages of Investing in Equity Shares**

There are several benefits of equity investment that an individual can enjoy by investing in equity shares. Some of them are enumerated below.

**1.7 High Returns**

Investing in equity shares provides high returns to investors, not just through dividend earnings but also through capital appreciation.

**1.8 Provides a Cushion Against Inflation**

When an individual invests in equity shares, he/she has the potential to earn high returns. The rate of returns earned is often higher than the rate of wearing down of the investor’s purchasing power due to inflation.

Thus, investing in equity shares acts as a hedge against inflation.

**1.9 Ease of Investment**

Investing in shares is simple. Investors can avail of the services of a stockbroker or financial planner to invest through various stock exchanges (NSE, BSE equity) in a country.

If an individual has set up a Demat account, he/she can buy the stocks in a few minutes.

**1.10 Diversification of Investment Portfolio**

Investors mostly choose to stick to debt instruments since they are low-risk investment options owing to lower volatility. However, debt instruments may not always generate high returns, which is why individuals can diversify their investment portfolio by investing in equities for higher returns.

**1.11 Disadvantages of Investing in Equities**

Even though equity investments have their fair share of advantages, they also bear a few disadvantages.

Some of them are as follows-

**1.12 High Market Risk**

Investing in equity shares can yield returns but also exposes investors to high risk as compared to other investment options like debt instruments.

An investor can risk losing his/her entire investment corpus by investing in equity shares.

**1.13 Performance-related Risks**

Equity investments are market-related instruments and, as a result, might not perform according to an investor’s expectations.

This is known as performance-related risk and can affect individual stocks as well as stocks across a sector or sectors.

**1.14 Risk of Inflation**

A company’s worth can get diluted due to rising inflation and subsequently, its shares might not generate potential returns.

**1.15 Liquidity Risk**

Due to liquidity risk, investors might have to sell their shares at a much lower price than their fair market value.

Liquidity risk arises when a company is unable to meet its debt obligations in the short term.

**1.16 Risks Arising out of Social and Political Changes**

On-going social and political issues in a country can hamper the growth of a business.

For example, if a government decides to promote indigenous businesses, it might restrict the entry of foreign businesses into the country. If an investor has invested in home-grown businesses, he/she, in this scenario, will profit from better performance of his/her investments.

Stock price prediction is a challenging task, and there are several challenges associated with it. stock price prediction is a complex task, and accurate predictions require careful consideration of these and other challenges. The following are key challenges in this regard:

**1. Market volatility:** The stock market is highly volatile, and there are many unpredictable factors that can affect stock prices, such as economic conditions, political events, and natural disasters. This volatility makes it challenging to accurately predict stock prices.

**2. Limited data:** Historical stock price data is limited and may not always be representative of current market conditions. Additionally, there may be gaps in the data or missing data points, which can make it difficult to train accurate predictive models.

**3. Non-linear relationships:** The relationship between different factors that affect stock prices is often non-linear, which makes it difficult to identify and model accurately. For example, the impact of a change in interest rates on stock prices may depend on a range of other factors, such as inflation, GDP growth, and consumer confidence.

**4. Overfitting:** Deep learning models can be prone to overfitting, where they perform well on the training data but fail to generalise to new data. This can lead to inaccurate predictions and reduced model performance.

**5. Time-series dependence:** Stock prices exhibit time-series dependence, which means that the current price is dependent on previous prices. This makes it challenging to accurately predict future prices as the prediction accuracy decreases as the prediction horizon increases.

**6. Limited interpretability:** Deep learning models are often complex and difficult to interpret, which can make it challenging to understand the factors driving the predictions and to make informed decisions based on the predictions.

**1.17 Time series data prediction**

Time series data prediction is an important tool for organisations that rely on data to make informed decisions. By analysing historical data and making predictions about future trends, organisations can improve their decision-making processes, optimise their operations, and mitigate risks. Time series data prediction is done for several reasons, including: Time series data prediction is done for several reasons, including:

**1. Forecasting:** One of the main reasons for time series data prediction is to make forecasts about future trends and patterns in the data. This can be useful for businesses and organizations that need to plan for future demand, sales, or resource allocation.

**2. Anomaly detection:** Time series data prediction can also be used to detect anomalies or unusual patterns in the data. This can help organizations identify potential problems or issues before they become more serious.

**3. Optimization:** Time series data prediction can be used to optimize resource allocation, production schedules, or other processes. By predicting future trends in the data, organizations can make more informed decisions about how to allocate resources or adjust processes to improve efficiency and productivity.

**4. Risk management:** Time series data prediction can also be used for risk management purposes, such as predicting market fluctuations or identifying potential security threats. By analyzing historical data and making predictions about future trends, organizations can take proactive measures to mitigate risks and protect their assets.

# **1.18 Equity Research prediction Techniques**

# Equity Research prediction without time series analysis can be challenging, as stock prices are inherently time-dependent and subject to various trends and patterns that can be difficult to predict without analysing historical data. However, there are some alternative methods for stock price prediction that do not rely on time series analysis. Here are a few examples:

**Sentiment analysis:** This method involves analyzing news articles, social media posts, and other sources of public sentiment to predict how people will react to certain events or news related to a particular stock. By understanding how public sentiment affects stock prices, investors can make more informed decisions about when to buy or sell.

**Fundamental analysis:** This method involves analyzing a company's financial statements, industry trends, and other factors to determine the intrinsic value of a stock. By evaluating a company's financial health, growth potential, and other factors, investors can make predictions about how the stock is likely to perform in the future.

**Technical analysis:** This method involves analyzing stock charts and other market data to identify patterns and trends that may indicate future price movements. By using technical indicators like moving averages, trend lines, and volume indicators, investors can make predictions about when to buy or sell a stock based on patterns in the data.

**Machine learning:** This method involves using machine learning algorithms to analyze large amounts of historical data and identify patterns that can be used to make predictions about future stock prices. Machine learning algorithms can be trained on various types of data, including financial statements, news articles, social media data, and other sources of information. while time series analysis is a powerful tool for predicting stock prices, there are alternative methods that can be used in certain situations. However, it is important to keep in mind that all of these methods have their limitations, and no method can accurately predict future stock prices with 100% certainty.

**Alternative Approaches**

There are several alternative methods of time series data prediction that can be used instead of deep learning. Here are a few examples:

**Autoregressive Integrated Moving Average (ARIMA):** ARIMA is a popular statistical method used for time series forecasting. It models the time series data as a combination of auto-regressive (AR) and moving average (MA) components and uses differencing to remove trends and seasonal effects. ARIMA models can be effective for forecasting short-term trends and patterns in time series data.

**Exponential Smoothing (ES):** ES is another popular statistical method used for time series forecasting. It models the time series data as a combination of level, trend, and seasonality components, and uses smoothing to estimate future values. ES models can be effective for forecasting short-term trends and patterns in time series data.

**Prophet:** Prophet is a forecasting tool developed by Facebook that uses an additive model to model seasonality, trends, and holidays. It can handle missing data and outliers, and can also incorporate custom seasonalities and events. Prophet is designed to be fast and scalable, and can be effective for forecasting medium-term trends and patterns in time series data.

Support Vector Machines (SVMs): SVMs are a type of machine learning algorithm that can be used for time series forecasting. They work by mapping the data to a high-dimensional feature space, and then finding the optimal hyperplane that separates the data into different classes. SVMs can be effective for forecasting nonlinear trends and patterns in time series data.

there are many different methods of time series data prediction, and the choice of method will depend on the specific application and the characteristics of the data. While deep learning is a powerful method, it is not always necessary or appropriate for every situation, and simpler statistical or machine learning methods can be just as effective in many cases.

Time series data prediction is a popular application of deep learning. Deep learning models are capable of processing and analysing large amounts of time series data, such as stock prices, weather patterns, or sensor readings, to make accurate predictions about future trends.

Some of the most common deep learning models used for time series data prediction include recurrent neural networks (RNNs), long short-term memory (LSTM) networks, and gated recurrent units (GRUs). These models are designed to handle sequential data, which is essential for time series data prediction.

The scope of time series data prediction using deep learning is vast and includes applications in many different fields, including finance, economics, healthcare, manufacturing, and more. For example, in finance, deep learning models can be used to predict stock prices, commodity prices, or currency exchange rates, while in healthcare, they can be used to predict disease outbreaks or patient outcomes.

In addition to predicting future trends, deep learning models can also be used for anomaly detection, where they identify unusual patterns in the time series data that may indicate an underlying issue or problem. This is especially useful in fields such as manufacturing or industrial production, where identifying and resolving issues quickly can save time and money.

The scope of time series data prediction using deep learning is significant, and the technology is only expected to continue to advance and expand into new areas in the future.

**1.19 Framework for Time Series Data Prediction**

A framework for time series data prediction using deep learning would typically involve the following steps:

**1.Data Preparation:** Collect and preprocess the time series data by cleaning, normalizing, and transforming it into a format suitable for deep learning models.

**2. Feature Engineering:** Extract features from the time series data to improve the performance of the deep learning models.

**3. Model Selection:** Select a deep learning model suitable for the type of time series data and the problem being addressed.

**4. Model Training:** Train the selected deep learning model on the preprocessed time series data using an appropriate loss function and optimization algorithm.

**5. Model Evaluation:** Evaluate the performance of the trained deep learning model using appropriate metrics such as mean squared error (MSE), root mean squared error (RMSE), and coefficient of determination (R2).

**6. Model Tuning:** Fine-tune the hyperparameters of the deep learning model to improve its performance on the time series data.

**7. Prediction and Visualization**: Use the trained deep learning model to make predictions on new time series data and visualise the results.

This framework can be applied to a variety of time series prediction problems, such as stock price forecasting, weather forecasting, and traffic flow prediction, among others. Deep learning models such as recurrent neural networks (RNNs), long short-term memory (LSTM) networks, and convolutional neural networks (CNNs) are commonly used for time series data prediction tasks.

**Chapter -2**

**Review of Literature**

**2.1 Introduction Review of Literature**

in every field, varying from medicine to finance. This diversified and critical utility time-series data analysis motivates researchers to perform it in a more precise and accurate manner, especially with regard to future value prediction. As the data involved is usually huge in volume, Therefore, the implementation of deep learning will provide more meaningful insides. To do this, the identification of impacting hyperparameters and related algorithms has to be done. So the value predicted from the previously specified hyperparameters and algorithm should be effective and efficient.

# **2.2 Research Gap identified in Literature reivew**

Organizations may utilize time series analysis to figure out what's causing trends or systemic patterns across time. Organizations may use time series forecasting to anticipate the likelihood of future occurrences when they evaluate data at regular intervals. For this purpose Artificial intelligence can play a very vital role in Time Series Data Analysis and Forecasting.

The Deep Learning Algorithms are very crucial for this reason. But there are many algorithms available and their performance varies on the basis of the hyperparameter chosen during their implementation which depends on the Data . Thus it is very difficult to identify the best performing Deep learning algorithm which can solve this purpose to optimize the prediction performance.

**2.3 Earlier Review of Literature**

**Mahmoud A., Mohammed A. (2020):** In this paper Mahmoud and Mohammed have given an overview of the most prevalent Deep Learning types for time series forecasting and explain the connections between deep learning models and traditional time series forecasting methodologies.

**Sengupta, S., et al. (2019):** In An overview of some of the most important multilayer ANNs used in Deep learning. Also described are some novel automated architectural optimization techniques that employ multi-agent methodologies. Furthermore, because ensuring system uptime is becoming increasingly important in many computer applications, the authors have added a section on employing neural networks to identify and mitigate faults.

**Tokui, S., Oono, K., Hido, S., Clayton, J.(2008):** Tokui et al. introduce Chainer, a Python-based, standalone open source framework for deep learning models, in this study. Chainer allows you to design a wide range of deep learning models, including cutting-edge models like recurrent neural networks and variational autoencoders, in a flexible, intuitive, and high-performance way.

**Cavalcante, R.C., Brasileiro, R.C., Souza, V.L., Nobrega, J.P., Oliveira, A.L.(2016):** A summary of the most relevant primary papers published between 2009 and 2015 can be found in by Cavalcante et. al, which cover approaches for preprocessing and grouping financial data, projecting future market movements, and mining financial text information, among other things.

**Vellido, A., Lisboa, P.J., Vaughan, J.(1999):** In Many of the studies are first efforts to apply these new methodologies to well-established fields of inquiry, with only a handful tackling real-world scenarios. Neural networks have progressed to the point that they can provide actual practical benefits in many of their applications, despite the fact that they are still considered a novel approach.

**Kim, K.-J., Ahn, H.(2012):** Here in the Genetic Algorithm improves numerous architectural elements and feature transformations of ANN in this work to alleviate the constraints of the traditional backpropagation algorithm in a synergistic manner. Experiments reveal that our suggested model outperforms traditional techniques in stock price index prediction.

**Adebiyi, A.A., Adewumi, A.O., Ayo, C.K.(2014):** Adebiyi et al. investigated the predicting effectiveness of ARIMA and artificial neural networks models using publicly available stock data from the New York Stock Exchange in this article. The empirical data show that the neural networks model outperforms the ARIMA model.

**Göçken, M., Özçalıcı, M., Boru, A., Dosdoğru, A.T.(2016):** In Hybrid Artificial Neural Network (ANN) models, which combine the capabilities of Harmony Search (HS) and Genetic Algorithm (GA), are used to select the most relevant technical indicators for the period under study in order to capture the relationship between technical indicators and the stock market. Furthermore, this research looks for the most appropriate number of hidden neurons in the hidden layer at the same time, and the suggested models address the well-known problem of ANN overfitting and underfitting.

**Lu, C.-J., Lee, T.-S., Chiu, C.-C.(2009):** Lu et al. in have used The Nikkei 225 opening index and TAIEX closing index as illustrative examples to evaluate the performance of the suggested ICA technique. The suggested approach outperforms the SVR model with non-filtered forecasting variables and the random walk model in experiments.

**Hossain, M.A., Karim, R., Thulasiram, R., Bruce, N.D., Wang, Y.(2018):** Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) are two well-known networks that are combined in this hybrid model (GRU). Authors have employed the key assessment criteria such as mean squared error, mean absolute percentage error, and other traditional ways to evaluate the S&P 500 historical time series data.

**Siami-Namini, S., Namin, A.S.(2018):** Deep learning-based algorithms, such as LSTM, outperform traditional-based algorithms, such as the ARIMA model, according to the empirical experiments done and presented in this article. When compared to ARIMA, the average reduction in error rates obtained by LSTM is between 84 and 87 percent, indicating that LSTM is superior to ARIMA. Furthermore, it was shown that the number of training times, often known as "epoch" in deep learning, has no influence on the trained forecast model's performance and that it behaves in a really random manner.

**Fischer, T., Krauss, C.(2018):** From 1992 through 2015, Fischer and Krauss used LSTM networks to forecast out-of-sample directional movements for the S&P 500 component companies. We discover that LSTM networks outperform memory-free classification approaches such as a random forest (RAF), a deep neural net (DNN), and a logistic regression classifier, with daily returns of 0.46 percent and a Sharpe ratio of 5.8 before transaction costs (LOG).

**dos Santos Pinheiro, L., Dras, M.(2017):** Pinheiro and Dras investigated recurrent neural networks for intraday and interday stock market forecasting using character-level language model pre-training. We show that our strategy is competitive with existing state-of-the-art approaches in forecasting directional changes in the Standard & Poor's 500 index, both for individual businesses and the entire index.

**Chen, K., Zhou, Y., Dai, F.(2017):** In authors have talked about Stock market forecasting that has drawn the interest of both industry and academics. Stock prices are predicted using a variety of machine learning methods, including neural networks, genetic algorithms, support vector machines, and others. And used these techniques on the data generated by the Chinese Stock Market.

**Ding, X., Zhang, Y., Liu, T., Duan, J.(2015):** When a deep convolutional neural network is used in to represent both short-term and long-term impacts of events on stock price movements, experimental findings demonstrate that the model can enhance S&P 500 index prediction and individual stock prediction by about 6%.

**Chen, W., Zhang, Y., Yeo, C.K., Lau, C.T., Lee, B.S.(2017):** Authors have provided current advancements in stock market prediction methods and models, as well as a comparison of these models to determine the accuracy of stock market value predictions and to determine the benefits and drawbacks of these specific models. To forecast future stock prices, authors have used LSTM and GRU models.

**Duan, Y., Lv, Y., Wang, F.-Y.(2016):** Duan et al. have projected multi-step advance trip times for each connection on the test set in . The median mean relative error for the 66 links in the tests is 7.0 percent on the test set, indicating that the 1-step forward travel time prediction error is reasonably minimal. In traffic series data prediction, deep learning algorithms that include sequence relationships are promising.

**Mehdiyev, N., Lahann, J., Emrich, A., Enke, D., Fettke, P., Loos, P.(2016):** Persio and Honchar developed a unique strategy that outperforms basic neural network approaches by combining wavelets and Conventional Neural Network.

**Di Persio, L., Honchar, O.(2016):** The authors of this research, Zhao et al. examined the spatio-temporal correlations to estimate the short-term passenger demand of an on-demand transportation service platform using Deep learning investigations.

**Ke, J., Zheng, H., Yang, H., Chen, X.M.(2016):** The study begins with a review of the fundamentals of time series processing before delving into feedforward and recurrent neural networks' capacity to describe non-linear relationships in spatio-temporal patterns.

**2.4 Table Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| S No | Year | Author | Objective |
| 1. | 2020 | Mahmoud A., Mohammed A. | Deep Learning types for time series forecasting |
| 2. | 2019 | Sengupta, S., et al. | ANNs used in Deep learning |
| 3. | 2008 | Tokui, S., Oono, K., Hido, S., Clayton, J | Cutting-edge models like recurrent neural networks |
| 4. | 2016 | Cavalcante, R.C., Brasileiro, R.C., Souza, V.L., Nobrega, J.P., Oliveira, A.L. | Projecting future market movements |
| 5. | 1999 | Vellido, A., Lisboa, P.J., Vaughan, J. | New methodologies to well-established fields of inquiry |
| 6. | 2012 | Kim, K.-J., Ahn, H. | Techniques in stock price index prediction |
| 7. | 2014 | Adebiyi, A.A., Adewumi, A.O., Ayo, C.K. | Neural networks model outperforms the ARIMA model |
| 8. | 2016 | Göçken, M., Özçalıcı, M., Boru, A., Dosdoğru, A.T. | Hybrid Artificial Neural Network (ANN) models |
| 9. | 2009 | Lu, C.-J., Lee, T.-S., Chiu, C.-C | SVR model with non-filtered forecasting |
| 10. | 2018 | Hossain, M.A., Karim, R., Thulasiram, R., Bruce, N.D., Wang, Y. | 500 historical time series data |
| 11. | 2018 | Siami-Namini, S., Namin, A.S | Trained forecast model's performance |
| 12. | 2018 | Fischer, T., Krauss, C. | LSTM networks to forecast out-of-sample |
| 13. | 2017 | dos Santos Pinheiro, L., Dras, M. | Interday stock market forecasting |
| 14. | 2017 | Chen, K., Zhou, Y., Dai, F | Stock market forecasting |
| 15. | 2015 | Ding, X., Zhang, Y., Liu, T., Duan, J | S&P 500 index prediction and individual stock prediction |
| 16. | 2016 | Duan, Y., Lv, Y., Wang, F.-Y. | Traffic series data prediction, deep learning algorithms |
| 17. | 2016 | Mehdiyev, N., Lahann, J., Emrich, A., Enke, D., Fettke, P., Loos, P. | Basic neural network approaches by combining |
| 18. | 2016 | Ke, J., Zheng, H., Yang, H., Chen, X.M | Fundamentals of time series |
| 19. | 2017 | Zhao, Z., Chen, W., Wu, X., Chen, P.C., Liu, J | Though neural networks have become a well-known business strategy |
| 20. | 1996 | Dorffner, G | Time-series analysis and forecasting |

**Chapter -3**

# **Research Methodology**

# **3.1 Research Methodology**

Research methodology is a structured and scientific approach used to collect, analyze, and interpret quantitative or qualitative data to answer research questions or test hypotheses. A research methodology is like a plan for carrying out research and helps keep researchers on track by limiting the scope of the research. Several aspects must be considered before selecting an appropriate research methodology, such as research limitations and ethical concerns that may affect your research.

The research methodology section in a scientific paper describes the different methodological choices made, such as the data collection and analysis methods, and why these choices were selected. The reasons should explain why the methods chosen are the most appropriate to answer the research question. A good research methodology also helps ensure the reliability and validity of the research findings. There are three types of research methodology—quantitative, qualitative, and mixed-method, which can be chosen based on the research objectives.

**3.2 Objective of Research Methodology**

Considering the facts mentioned in the problem statement, the solution of the research problem will be achieved from the following objectives:

1. To identify and analyze the current Deep Learning algorithms available for Available for Financial Predictive Analysis.
2. To identify use of predictive Model in equity market.
3. To implement and test the performance of the financial predictive model.

In order to achieve the objectives defined in the above section following methodology is taken in consideration. Firstly to identify the current available algorithms and Hyperparameter for prediction regress literature survey is to be done and then by doing the comparative study of algorithms and hyperparameters, these two are decided.

Once the algorithms and parameters are identified then the model is to be developed over it and implemented using python and its libraries. For execution of the proposed model, Time Series data is required and we are going to select the Indian stock market data especially equity market for this purpose because it is easily available and performance can be realized simultaneously. Then the selected data and model are blended together to analyze the performance of the proposed model and thus it can be optimized for further usage.

+--------------+

| Input Data |

+--------------+

| V

+------------------+

| Data Processing |

+------------------+

| V

+------------------------------+

| Feature Extraction & Selection|

+------------------------------+

| V

+--------------+

| Model Training|

+--------------+

| V

+-------------------------------+

| Model Evaluation & Optimization|

+-------------------------------+

| V

+---------------+

| Output Data |

+---------------+

In this framework, the input data is collected from various sources such as financial databases or stock market feeds. The data processing step involves cleaning, filtering, and transforming the data into a usable format.

Next, feature extraction and selection are performed to identify relevant variables that could potentially influence stock prices. This step may involve statistical analysis, data visualization, and other techniques.

After selecting relevant features, the model training process begins. This step involves training deep learning models, such as LSTM, GRU, or CNN, on the historical data to predict future stock prices.

Once the model is trained, it is evaluated and optimized to improve its performance. This step involves validating the model's accuracy and tuning the hyperparameters to achieve the best results.

Finally, the output data is generated, which includes predicted stock prices and other related metrics. The framework can be updated with new data to continually refine the model and improve its performance.

As a result of the experiment, we are willing to provide the metrics for the selection of the best available Deep Learning Algorithm for time series data analysis. Also we are trying to develop a portal that is capable of predicting the future value of Nifty Fifty Stock on the basis of the optimized Deep Learning based Model.

**3.3 Research Hypothesis**

The procedure, dataset, and experimental setup that will be utilized to forecast a stock's value are described in this section. We have used the dataset from NSE for this experiment. The Nifty 50 businesses on the Indian Stock Market are all represented in this data by their daily records. The data used in this experiment spans a 8-year period from 2017 to 2024. where there are two portions, one lasting Seven and a half years and the other just a half year, to the Eight year term. The first component is used as the machine learning algorithm's training set, while the second part is used as the algorithm's testing set.

## 3.4 Table Dataset field Description

|  |  |
| --- | --- |
| **Dataset Field Name** | **Specification** |
| Date | Date specifies the Date of record for the Stock |
| High | Highest Value of the Stock |
| Low | Lowest Value of the Stock |
| Open | Opening Value of the Stock |
| Close | Closing Value of the Stock on the Date of Record |
| Volume | Volume of the Stock |
| Adj Close | Adjusted Closing Price of the Stock |

The experimental configuration employed here utilizes an Intel® core TM i5-9500U CPU running at 2.70GHz across four CPUs, together with 8GB of memory. For the creation of LSTM and other automation, Python is employed as the programming language. Important libraries used in this project are Pandas, Matplotlib, Numpy, and Keras.

**3.5 Tools:**

**Python:** Python is a high-level, general-purpose, and very popular programming language. Python programming language (latest Python 3) is being used in web development, and Machine Learning applications, along with all cutting-edge technology in Software Industry. Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc.

**Pandas: Pandas** is an open-source library that is built on top of NumPy library. It is a Python package that offers various data structures and operations for manipulating numerical data and time series. It is mainly popular for importing and analyzing data much easier. Pandas is fast and it has high-performance & productivity for users.

**Numpy: Numpy**is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data.

**Keras:** Keras is the high-level API of the TensorFlow platform. It provides an approachable, highly-productive interface for solving machine learning (ML) problems, with a focus on modern deep learning. Keras covers every step of the machine learning workflow, from data processing to hyperparameter tuning to deployment. It was developed with a focus on enabling fast experimentation.

With Keras, you have full access to the scalability and cross-platform capabilities of TensorFlow. You can run Keras on a TPU Pod or large clusters of GPUs, and you can export Keras models to run in the browser or on mobile devices. You can also serve Keras models via a web API.

The data from the stock must first be obtained in order to start the experiment, and only then may additional steps be taken. Here, the Nifty 50 NSE dataset is used. The Records are the subject of the experiment, as was previously stated in this section. Since the LSTM (Long Short Term Memory) Algorithm is particularly sensitive to data value, processing of the common separated value file of the data begins as soon as it is uploaded in Python for the building of Data frame. The value is to be normalized in a range of 0 to 1.

Data that has undergone preprocessing has been separated into training and testing sets. Then, using an approach where the closure price is dependent upon the previous closing price, an LSTM Model is built with several layers. In this case, 100 layers are thought to exist. Once this model has been created, training is given to it so that it may educate itself using the Training Set and Testing Set.

Following the prediction of this future value on a predetermined training set, the accuracy of the results is determined by comparing them to the equivalent value in the testing test. The Model is now prepared to forecast more future values. For pattern analysis and decision support, the observed value and the expected value might be shown on a graph.

## Collection of data in CSV format

1. **Preprocessing of data**

## Selection the parameter to be used

1. **Dividing the data to training and testing set**

## Create LSTM Model

1. **Pass the Training Set to LSTM to train it**

## Predict the future value

1. **Now compare the observed value with the corresponding value in testing set**
2. **Plot the observed value and predicted value in a graph**

**Chapter 4**

**Data analysis, model interpretation and Hypothesis testing**

**4.1 Dataset:**

A **Dataset is a set of data grouped into a collection** with which developers can work to meet their goals. In a dataset, the rows represent the number of data points and the columns represent the features of the Dataset. They are mostly used in fields like machine learning, business, and government to gain insights, make informed decisions, or train algorithms. Datasets may vary in size and complexity and they mostly require cleaning and preprocessing to ensure data quality and suitability for analysis or modeling.

**Datasource:** https://finance.yahoo.com/quote/RELIANCE.NS/history

**4.2 Shape:**

**Understanding the Shape of Reliance Stock Data**

This code snippet helps you understand the structure and dimensions of the Reliance stock data stored in a Pandas DataFrame.

* Loaded Reliance stock data into a Pandas DataFrame named Reliance\_stock.

**Accessing the Shape:**

* We use the .shape attribute on the Reliance\_stock DataFrame.
* The .shape attribute returns a tuple representing the Relince\_stock dimensions of the DataFrame. The first element indicates the number of rows, and the second element indicates the number of columns.

.shape

(1784, 7)

**Datasource:** <https://finance.yahoo.com/quote/RELIANCE.NS/history>

**4.3 Head:**

The head() method returns a specified number of rows, string from the top. The head() method returns the first 5 rows if a number is not specified.

|  | **Date** | **Open** | **High** | **Low** | **Close** | **Adj Close** | **Volume** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 2016-12-21 | 484.642059 | 487.933655 | 483.316284 | 486.082123 | 469.415131 | 10183947 |
| **1** | 2016-12-22 | 483.682007 | 485.807831 | 480.458984 | 481.556183 | 465.044464 | 14446417 |
| **2** | 2016-12-23 | 480.938995 | 488.687988 | 480.938995 | 483.362000 | 466.788269 | 5132516 |
| **3** | 2016-12-26 | 482.013336 | 487.339325 | 477.578827 | 478.858887 | 462.439606 | 7660885 |
| **4** | 2016-12-27 | 479.041779 | 487.842224 | 478.653168 | 486.745026 | 470.055328 | 8721424 |

**Datasource:** https://finance.yahoo.com/quote/RELIANCE.NS/history

**4.4 Tail:**

The tail() method returns a specified number of last rows. The tail() method returns the last 5 rows if a number is not specified.

| **Date** | **Open** | **High** | **Low** | **Close** | **Adj Close** | **Volume** |
| --- | --- | --- | --- | --- | --- | --- |
| **1779** | 2024-03-01 | 2927.000000 | 3000.000000 | 2925.000000 | 2984.250000 | 2984.250000 | 6066463 |
| **1780** | 2024-03-04 | 2980.949951 | 3024.899902 | 2974.449951 | 3014.800049 | 3014.800049 | 5012210 |
| **1781** | 2024-03-05 | 3011.550049 | 3014.800049 | 2972.100098 | 3000.399902 | 3000.399902 | 3553834 |
| **1782** | 2024-03-06 | 2986.899902 | 3018.000000 | 2957.000000 | 3006.000000 | 3006.000000 | 3902838 |
| **1783** | 2024-03-07 | 3005.949951 | 3006.199951 | 2951.100098 | 2957.850098 | 2957.850098 | 4157863 |

**Datasource:** https://finance.yahoo.com/quote/RELIANCE.NS/history

**4.5 Describe:**

Pandas describe() is used to view some basic statistical details like percentile, mean, std, etc. of a data frame or a series of numeric values. When this method is applied to a series of strings, it returns a different output which is shown in the examples below.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Index** | **Open** | | **High** | | **Low** | | **Close** | | **Adj Close** | | **Volume** | |
| **Count** | 1784 | | 1784 | | 1784 | | 1784 | | 1784 | | 1784 | |
| **Mean** | 1620.378 | | 1637.416 | | 1602.397 | | 1619.338 | | 1603.057 | | 9867284 | |
| **Std** | 670.3293 | | 675.8953 | | 664.3901 | | 670.0643 | | 674.8787 | | 7491827 | |
| **Min** | 464.9382 | | 470.3327 | | 463.0181 | | 464.4353 | | 448.5105 | | 852828 | |
| **25%** | 1012.955 | | 1027.961 | | 1001.949 | | 1013.926 | | 994.2025 | | 5530329 | |
| **50%** | 1771.236 | | 1789.258 | | 1745.831 | | 1765.168 | | 1747.143 | | 7639083 | |
| **75%** | 2251.072 | | 2275.125 | | 2230.025 | | 2251.519 | | 2238.731 | | 11281975 | |
| **Max** | 3011.55 | | 3024.9 | | 2974.45 | | 3014.8 | | 3014.8 | | 71341683 | |
|  | |  | |  | |  | |  | |  | |
|  | |  | |  | |  | |  | |  | |

**4.6 DataSet: Relince\_stock**

**# Create input and output sequences for training data**

**X\_train = []**

**y\_train = []**

**window\_size = 3**

**for i in range(window\_size, len(train\_data)):**

**X\_train.append(train\_data[i-window\_size:i, 0])**

**y\_train.append(train\_data[i, 0])**

**X\_train, y\_train = np.array(X\_train), np.array(y\_train)**

**# Reshape the input data to be 3-dimensional (batch size, time steps, features) X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))**

**We will now define and train the LSTM model:**

**# Define the LSTM model model = Sequential()**

**model.add(LSTM(units=50, return\_sequences=True, input\_shape=(X\_train.shape[1], 1))) model.add(LSTM(units=50))**

**model.add(Dense(units=1))**

**# Compile the model**

**model.compile(optimizer='adam', loss='mean\_squared\_error')**

**# Train the model**

**model.fit(X\_train, y\_train, epochs=100, batch\_size=32)**

# **Finally, we will use the trained model to make predictions on the testing data:**

***# Create input and output sequences for testing data***

***X\_test = []***

***y\_test = []***

***for i in range(window\_size, len(test\_data)):***

***X\_test.append(test\_data[i-window\_size:i, 0])***

***y\_test.append(test\_data[i, 0])***

***X\_test, y\_test = np.array(X\_test), np.array(y\_test)***

***# Reshape the input data to be 3-dimensional (batch size, time steps, features) X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1))***

***# Make predictions on the testing data***

***y\_pred = model.predict(X\_test)***

***# Inverse transform the scaled predictions and actual values***

***y\_pred = scaler.inverse\_transform(y\_pred)***

We can then evaluate the performance of the model using metrics such as mean squared error (MSE) and mean absolute error (MAE):

***# Calculate the mean squared error and mean absolute error mse = mean\_squared\_error(y\_test, y\_pred)***

***mae = mean\_absolute\_error(y\_test, y\_pred)***

***print("Mean Squared Error:", mse)***

***print("Mean Absolute)***

**Train the model:** Train the model using the training data, and monitor its performance on the validation set to prevent overfitting. Adjust the hyperparameters and model architecture as necessary to improve performance.

**Evaluate the model:** Evaluate the performance of the model on the testing set using metrics such as mean squared error (MSE), mean absolute error (MAE), and coefficient of determination (R-squared).

**Use the model for prediction:** Once the model is trained and evaluated, use it to make predictions on new, unseen data.

Here is a sample code for developing an LSTM model for time series prediction of stock prices using the identified hyperparameters:

**4.7 Predicting Equity Market Using LSTM Neural Networks**

**Import the Libraries**

This code snippet shows how to predict stock prices using LSTM neural networks. LSTM networks are great for predicting sequences, like stock prices, because they can remember patterns over time.

**Before we start, make sure you have the following libraries installed:**

* **Pandas:** Helps with data manipulation.
* **NumPy:** Used for numerical operations.
* **Matplotlib:** Enables data visualization.
* **Scikit-learn:** Used for data preprocessing and splitting.
* **Keras:** A library for building neural networks.

**Importing Libraries:**

* We import all the libraries needed to work with data, build neural networks, and visualize results.

**Data Collection:**

* We fetch historical stock price data using pandas\_datareader. This data will be used for training our model.

**Data Preprocessing:**

* We scale the data to make it easier for the model to learn patterns. Scaling ensures all values are in a similar range.
* The data is split into training and testing sets.
* This allows us to evaluate how well our model performs on unseen data.

**Building the LSTM Model:**

* We create a Sequential model, which is a basic neural network structure in Keras.
* We add LSTM layers to the model.
* These layers are essential for processing sequential data like stock prices.
* Additional layers like Dense and Dropout are added to improve the model's performance.

**Model Training:**

* We train the model on the training data.
* During training, the model adjusts its parameters to minimize the difference between predicted and actual stock prices.

**Model Evaluation:**

* We evaluate the model's performance using the test data.
* This helps us understand how well our model generalizes to new, unseen data.

**Visualization:**

* Finally, we visualize the predicted stock prices alongside the actual prices using Matplotlib.
* This allows us to see how well our model predicts future prices.

#Import the Libraries  
import pandas as pd  
import numpy as np  
import datetime  
from sklearn import datasets  
import pandas\_datareader as web  
from pandas.io.formats.style\_render import DataFrame  
from sklearn.preprocessing import MinMaxScaler  
from sklearn.preprocessing import StandardScaler  
from keras.models import Sequential  
from sklearn.model\_selection import train\_test\_split  
from keras.layers import Dense, Dropout, LSTM, Bidirectional  
import matplotlib.pyplot as plt  
#plt.style.use('fivethirtyeight')

**Import CSV File**

Uploading Dataset in Google Colab

This code snippet demonstrates how to upload a dataset into Google Colab, a cloud-based platform commonly used for data analysis and machine learning tasks.

* Access to Google Colab, which is a part of Google Drive.
* A dataset that you want to upload.

Importing Libraries:

* We use the from google.colab import files statement to import the files module from Google Colab.
* This module allows us to upload files directly into our Colab environment.

Uploading Dataset:

* We use the files.upload() function to prompt the user to select a file from their local system for upload.
* The selected dataset file is then uploaded to the Colab environment.

from google.colab import files  
dataset = files.upload()

<IPython.core.display.HTML object>

Saving RELIANCE.NS.csv to RELIANCE.NS.csv

**Datasource:** <https://finance.yahoo.com/quote/RELIANCE.NS/history>

**Loading Reliance Stock Data**

This code snippet demonstrates how to load Reliance stock data from a CSV (Comma Separated Values) file into a Pandas DataFrame.

* The CSV file containing Reliance stock data.
* Basic knowledge of Pandas, a Python library used for data manipulation and analysis.

**Loading the Data:**

* We use the pd.read\_csv() function from the Pandas library to read the CSV file containing Reliance stock data.
* The file path "RELIANCE.NS.csv" should point to the location of the CSV file in your local system.

**Viewing the Data:**

* After loading the data into a Pandas DataFrame named Reliance\_stock, we use the .head() method to display the first few rows of the DataFrame.
* This allows us to quickly inspect the structure and contents of the data.

Relince\_stock = pd.read\_csv("RELIANCE.NS.csv")  
Relince\_stock.head()

|  | **Date** | **Open** | **High** | **Low** | **Close** | **Adj Close** | **Volume** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 2016-12-21 | 484.642059 | 487.933655 | 483.316284 | 486.082123 | 469.415131 | 10183947 |
| **1** | 2016-12-22 | 483.682007 | 485.807831 | 480.458984 | 481.556183 | 465.044464 | 14446417 |
| **2** | 2016-12-23 | 480.938995 | 488.687988 | 480.938995 | 483.362000 | 466.788269 | 5132516 |
| **3** | 2016-12-26 | 482.013336 | 487.339325 | 477.578827 | 478.858887 | 462.439606 | 7660885 |
| **4** | 2016-12-27 | 479.041779 | 487.842224 | 478.653168 | 486.745026 | 470.055328 | 8721424 |
|  |  |  |  |  |  |  |  |

**Viewing the Last Rows of Reliance Stock Data**

**Datasource:** https://finance.yahoo.com/quote/RELIANCE.NS/history

his code snippet demonstrates how to view the last rows of Reliance stock data stored in a Pandas DataFrame.

Loaded Reliance stock data into a Pandas DataFrame using pd.read\_csv() or a similar method.

**Viewing the Last Rows:**

* We use the .tail() method on the Reliance\_stock DataFrame.
* The .tail() method displays the last few rows of the DataFrame, allowing us to observe recent data points

Relince\_stock.tail()

| **Date** | **Open** | **High** | **Low** | **Close** | **Adj Close** | **Volume** |
| --- | --- | --- | --- | --- | --- | --- |
| **1779** | 2024-03-01 | 2927.000000 | 3000.000000 | 2925.000000 | 2984.250000 | 2984.250000 | 6066463 |
| **1780** | 2024-03-04 | 2980.949951 | 3024.899902 | 2974.449951 | 3014.800049 | 3014.800049 | 5012210 |
| **1781** | 2024-03-05 | 3011.550049 | 3014.800049 | 2972.100098 | 3000.399902 | 3000.399902 | 3553834 |
| **1782** | 2024-03-06 | 2986.899902 | 3018.000000 | 2957.000000 | 3006.000000 | 3006.000000 | 3902838 |
| **1783** | 2024-03-07 | 3005.949951 | 3006.199951 | 2951.100098 | 2957.850098 | 2957.850098 | 4157863 |

**Datasource:** <https://finance.yahoo.com/quote/RELIANCE.NS/history>

**Understanding the Shape of Reliance Stock Data**

This code snippet helps you understand the structure and dimensions of the Reliance stock data stored in a Pandas DataFrame.

* Loaded Reliance stock data into a Pandas DataFrame named Reliance\_stock.

**Accessing the Shape:**

* We use the .shape attribute on the Reliance\_stock DataFrame.
* The .shape attribute returns a tuple representing the dimensions of the DataFrame. The first element indicates the number of rows, and the second element indicates the number of columns.

Relince\_stock.shape

(1784, 7)

**Datasource:** https://finance.yahoo.com/quote/RELIANCE.NS/history

**Understanding Reliance Stock Data Information**

This code snippet helps you obtain essential information about the Reliance stock data stored in a Pandas DataFrame.

* Loaded Reliance stock data into a Pandas DataFrame named Reliance\_stock.

**Accessing Data Information:**

* We use the .info() method on the Reliance\_stock DataFrame.
* The .info() method provides a concise summary of the DataFrame, including the number of non-null values in each column, data types, and memory usage.

Relince\_stock.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1784 entries, 0 to 1783  
Data columns (total 7 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Date 1784 non-null object   
 1 Open 1784 non-null float64  
 2 High 1784 non-null float64  
 3 Low 1784 non-null float64  
 4 Close 1784 non-null float64  
 5 Adj Close 1784 non-null float64  
 6 Volume 1784 non-null int64   
dtypes: float64(5), int64(1), object(1)  
memory usage: 97.7+ KB

**Datasource:** https://finance.yahoo.com/quote/RELIANCE.NS/history

**Extracting and Formatting Reliance Stock Data**

This code snippet demonstrates how to extract specific columns from the Reliance stock data, format the date column, and set it as the index of the DataFrame.

* Loaded Reliance stock data into a Pandas DataFrame named Reliance\_stock.
* Basic knowledge of data manipulation using Pandas.

**Extracting Required Columns:**

* We create a new DataFrame named Reliance\_stock containing only the columns 'Date', 'Open', and 'Close' using indexing.

**Formatting the Date Column:**

* We convert the 'Date' column to datetime format using the pd.to\_datetime() function.
* The .apply(lambda x: x.split()[0]) part splits each date-time value, keeping only the date part.

**Setting Date as Index:**

* We set the 'Date' column as the index of the DataFrame using the .set\_index() method.
* The drop=True parameter indicates that the original 'Date' column should be removed from the DataFrame.

Relince\_stock = Relince\_stock[['Date','Open','Close']] # Extracting required columns  
Relince\_stock['Date'] = pd.to\_datetime(Relince\_stock['Date'].apply(lambda x: x.split()[0])) # Selecting only date  
Relince\_stock.set\_index('Date',drop=True,inplace=True) # Setting date column as index  
Relince\_stock.head()

<ipython-input-7-ab823c9d11b0>:2: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 Relince\_stock['Date'] = pd.to\_datetime(Relince\_stock['Date'].apply(lambda x: x.split()[0])) # Selecting only date

|  |  |  |  |  | **Date**  **2016-12-21** | **Open**  484.642059 | **Close**  486.082123 |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | **2016-12-22** | 483.682007 | 481.556183 |
|  |  |  |  |  | **2016-12-23** | 480.938995 | 483.362000 |
|  |  |  |  |  | **2016-12-26** | 482.013336 | 478.858887 |
|  |  |  |  |  | **2016-12-27** | 479.041779 | 486.745026 |
|  |  |  |  |  |
|  |  |  |
|  |  |  |

**Visualizing Reliance Stock Open and Close Prices**

**Datasource:** <https://finance.yahoo.com/quote/RELIANCE.NS/history>

This code snippet demonstrates how to create a side-by-side plot to visualize the open and close prices of Reliance stock over time.

* Imported the necessary libraries, including matplotlib.pyplot.
* Loaded and formatted Reliance stock data into a DataFrame named Reliance\_stock.

**Creating Subplots:**

* We create a figure with two subplots using plt.subplots(1, 2, figsize=(20, 7)).
* The 1 indicates one row of subplots, and 2 indicates two columns.
* We specify the figure size as (20, 7) to control the dimensions of the plot.

**Plotting Open Prices:**

* We plot the 'Open' prices on the first subplot (ax[0]) using ax[0].plot() method.
* We set the x-label as 'Date' and y-label as 'Price' using ax[0].set\_xlabel() and ax[0].set\_ylabel() methods, respectively.
* A legend is added to the plot to indicate the line corresponds to the 'Open' prices.

**Plotting Close Prices:**

* Similarly, we plot the 'Close' prices on the second subplot (ax[1]) using ax[1].plot() method.
* We set the x-label and y-label for the second subplot.
* A legend is added to the plot to indicate the line corresponds to the 'Close' prices.

**Displaying the Plot:**

* We use the fg.show() method to display the plot.

fg, ax =plt.subplots(1,2,figsize=(20,7))  
ax[0].plot(Relince\_stock ['Open'],label='Open',color='green')  
ax[0].set\_xlabel('Date',size=15)  
ax[0].set\_ylabel('Price',size=15)  
ax[0].legend()  
ax[1].plot(Relince\_stock ['Close'],label='Close',color='red')  
ax[1].set\_xlabel('Date',size=15)  
ax[1].set\_ylabel('Price',size=15)  
ax[1].legend()  
fg.show()

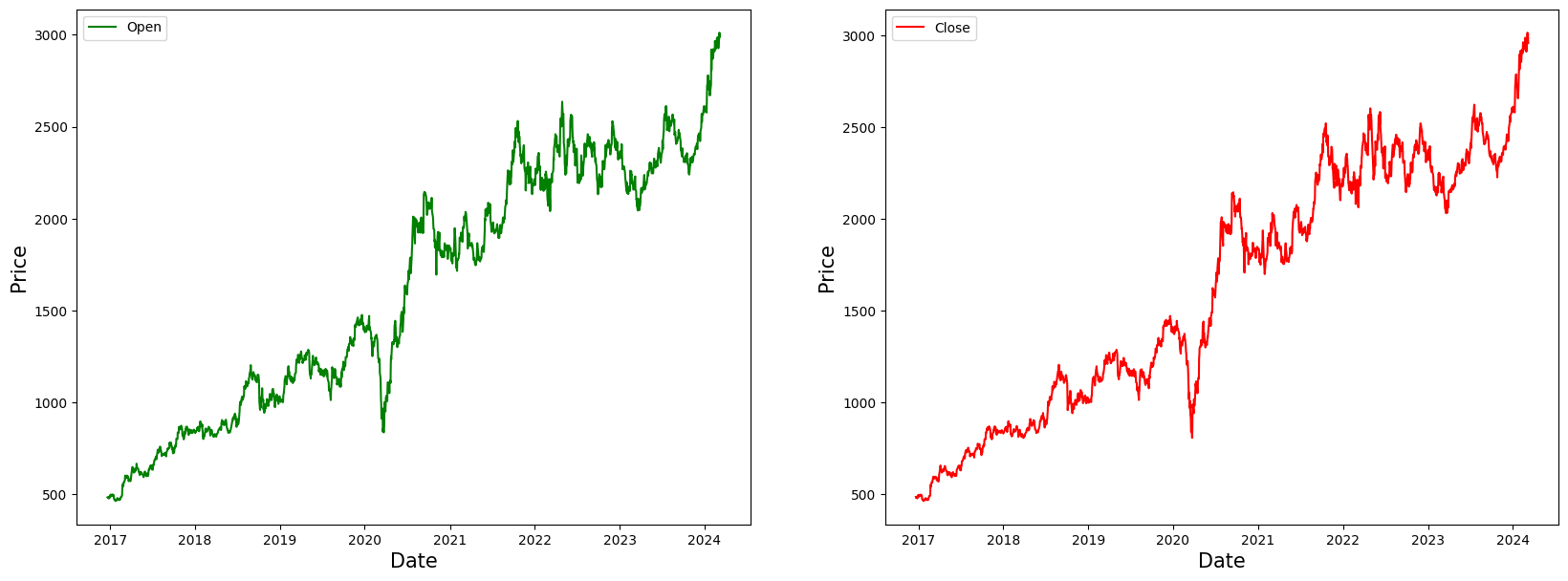


Figure 4.1

From the study of Figure 4.1 displays the seven-year data for Reliance Industries Limited (NSE: RELIANCE) with the opening prices depicted in green and the closing prices in red, offering a clear visual representation of the stock's performance over time.

**Datasource:** https://finance.yahoo.com/quote/RELIANCE.NS/history

Scaling Reliance Stock Data using Min-Max Scaler

This code snippet demonstrates how to scale the Reliance stock data using the Min-Max Scaler from the scikit-learn library. Scaling is important for ensuring all features are on a similar scale, which can improve the performance of certain machine learning algorithms.

* Imported the MinMaxScaler from the scikit-learn library.
* Loaded and preprocessed the Reliance stock data into a DataFrame named Reliance\_stock.

Importing MinMaxScaler:

* We import the MinMaxScaler from the sklearn.preprocessing module.
* Initializing MinMaxScaler:
* We create an instance of the MinMaxScaler and store it in a variable named MMS.

Fitting and Transforming Data:

* We use the fit\_transform() method of the MinMaxScaler to scale the data in the Reliance\_stock DataFrame.
* The fit\_transform() method fits the scaler to the data and transforms it in a single step.
* We apply the transformation to all columns in the DataFrame by passing Reliance\_stock[Reliance\_stock.columns] as input.

from sklearn.preprocessing import MinMaxScaler  
MMS = MinMaxScaler()  
Relince\_stock[Relince\_stock.columns] = MMS.fit\_transform(Relince\_stock)

**Datasource:** https://finance.yahoo.com/quote/RELIANCE.NS/history

**Selecting 80 % for training and 20 % for testing**

**Splitting Reliance Stock Data for Training and Testing**

This code snippet demonstrates how to split the Reliance stock data into training and testing sets, with 80% of the data used for training and 20% for testing.

* Imported the necessary libraries and loaded the Reliance stock data.
* Basic understanding of data splitting for machine learning tasks.

**Calculating Training Size:**

* We calculate the size of the training set by multiplying the length of the Reliance\_stock DataFrame by 0.80 (80%).
* The round() function is used to round the result to the nearest integer.

# Selecting 80 % for training and 20 % for testing  
training\_size = round(len(Relince\_stock ) \* 0.80)  
training\_size

1427

**Datasource:** https://finance.yahoo.com/quote/RELIANCE.NS/history

**Creating Training and Testing Sets for Reliance Stock Data**

This code snippet demonstrates how to create training and testing sets from the Reliance stock data. The data is split based on the calculated training size, with the remaining data used for testing.

* Imported the necessary libraries and loaded the Reliance stock data.
* Calculated the training size using the previously described method.

**Creating Training Set:**

* We create the training set train\_data by selecting the first training\_size rows from the Relince\_stock DataFrame using slicing.

**Creating Testing Set:**

* The testing set test\_data is created by selecting the remaining rows from the Relince\_stock DataFrame after the training set.

**Checking Shapes:**

* We check the shapes of the training set (train\_data) and the original Reliance stock data (Relince\_stock) to ensure the split was performed correctly.

train\_data = Relince\_stock [:training\_size]  
test\_data = Relince\_stock [training\_size:]  
train\_data.shape, Relince\_stock.shape

((1427, 2), (1784, 2))

**Datasource:** https://finance.yahoo.com/quote/RELIANCE.NS/history

**Creating Sequences of Data for Training and Testing**

This code snippet defines a function create\_sequence() to generate sequences of data for training and testing purposes. Each sequence consists of a set number of rows from the dataset, and the corresponding labels are extracted for prediction.

* Defined the function create\_sequence().
* Loaded and preprocessed the Reliance stock data.
* Calculated the training and testing sets.

**Defining the Function:**

* We define a function named create\_sequence() that takes a dataset as input.

**Generating Sequences and Labels:**

* Within the function, we initialize empty lists sequences and labels.
* We iterate over the dataset, selecting 50 rows at a time (range(50, len(dataset))).
* For each iteration, we append the sequence of rows to the sequences list and the corresponding label to the labels list.
* The start\_idx variable is incremented to slide the window of rows.

**Returning Sequences and Labels:**

* Finally, we return the sequences and labels as NumPy arrays.

**Creating Training and Testing Sequences:**

* We use the create\_sequence() function to generate sequences and labels for both the training and testing sets (train\_data and test\_data).
* The resulting sequences and labels are stored in variables train\_seq, train\_label, test\_seq, and test\_label.

# Function to create sequence of data for training and testing  
def create\_sequence(dataset):  
 sequences = []  
 labels = []  
 start\_idx = 0  
  
 for stop\_idx in range(50,len(dataset)): # Selecting 50 rows at a time  
 sequences.append(dataset.iloc[start\_idx:stop\_idx])  
 labels.append(dataset.iloc[stop\_idx])  
 start\_idx += 1  
 return (np.array(sequences),np.array(labels))

train\_seq, train\_label = create\_sequence(train\_data)  
test\_seq, test\_label = create\_sequence(test\_data)

train\_seq.shape, train\_label.shape, test\_seq.shape, test\_label.shape

((1377, 50, 2), (1377, 2), (307, 50, 2), (307, 2))

**Datasource:** https://finance.yahoo.com/quote/RELIANCE.NS/history

**Creating and Compiling an LSTM Model**

This code snippet defines and compiles an LSTM (Long Short-Term Memory) model using the Keras library. LSTM is a type of recurrent neural network (RNN) often used for sequence prediction tasks due to its ability to retain information over time.

* Imported the necessary libraries, including Sequential and LSTM from Keras.
* Defined the sequences and labels for training the model.
* Basic understanding of neural networks and Keras syntax.

**Initializing the Model:**

* We create a Sequential model using Sequential(). This allows us to add layers one by one.

**Adding LSTM Layers:**

* We add the first LSTM layer to the model using model.add(LSTM(units=50, return\_sequences=True, input\_shape=(train\_seq.shape[1], train\_seq.shape[2]))).
* This layer consists of 50 units and returns sequences (since return\_sequences=True), with an input shape corresponding to the shape of the training sequences.

**Adding Dropout Layer:**

* We add a Dropout layer with a dropout rate of 0.1 to prevent overfitting using model.add(Dropout(0.1)).

**Adding Second LSTM Layer:**

* We add a second LSTM layer with 50 units using model.add(LSTM(units=50)).

**Adding Dense Layer:**

* We add a Dense layer with 2 units using model.add(Dense(2)). The output layer has 2 units since we're predicting 2 labels.

**Compiling the Model:**

* We compile the model using model.compile() with 'mean\_squared\_error' as the loss function, 'adam' as the optimizer, and 'mean\_absolute\_error' as the metric to monitor during training.

**Model Summary:**

* We display a summary of the model architecture and parameter count using model.summary().

model = Sequential()  
model.add(LSTM(units=50, return\_sequences=True, input\_shape = (train\_seq.shape[1], train\_seq.shape[2])))  
  
model.add(Dropout(0.1))  
model.add(LSTM(units=50))  
  
model.add(Dense(2))  
  
model.compile(loss='mean\_squared\_error', optimizer='adam', metrics=['mean\_absolute\_error'])  
  
model.summary()

Model: "sequential"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 Layer (type) Output Shape Param #   
=================================================================  
 lstm (LSTM) (None, 50, 50) 10600   
   
 dropout (Dropout) (None, 50, 50) 0   
   
 lstm\_1 (LSTM) (None, 50) 20200   
   
 dense (Dense) (None, 2) 102   
   
=================================================================  
Total params: 30902 (120.71 KB)  
Trainable params: 30902 (120.71 KB)  
Non-trainable params: 0 (0.00 Byte)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Datasource:** https://finance.yahoo.com/quote/RELIANCE.NS/history

**Training the LSTM Model**

This code snippet trains the previously defined LSTM model using the training sequences and labels. It also evaluates the model's performance on the testing sequences and labels.

* Defined and compiled the LSTM model.
* Generated training and testing sequences and labels.

**Model Training:**

* We use the fit() method on the model to train it using the training sequences and labels.
* The train\_seq and train\_label variables are passed as input to the method.
* We specify the number of epochs as 80 (epochs=80) to determine how many times the model will iterate over the entire training dataset during training.
* The verbose=1 parameter indicates that training progress will be displayed during training.

**Validation Data:**

* We provide the testing sequences and labels as validation data using the validation\_data parameter.
* This allows us to evaluate the model's performance on unseen data during training.
* model.fit(train\_seq, train\_label, epochs=80,validation\_data=(test\_seq, test\_label), verbose=1)

Data output predict report:

Epoch 1/80  
44/44 [==============================] - 5s 43ms/step - loss: 0.0143 - mean\_absolute\_error: 0.0744 - val\_loss: 0.0030 - val\_mean\_absolute\_error: 0.0438  
Epoch 2/80  
44/44 [==============================] - 1s 29ms/step - loss: 9.2962e-04 - mean\_absolute\_error: 0.0223 - val\_loss: 0.0011 - val\_mean\_absolute\_error: 0.0253  
Epoch 3/80  
44/44 [==============================] - 1s 29ms/step - loss: 8.7777e-04 - mean\_absolute\_error: 0.0214 - val\_loss: 0.0012 - val\_mean\_absolute\_error: 0.0269  
Epoch 4/80  
44/44 [==============================] - 1s 29ms/step - loss: 8.8230e-04 - mean\_absolute\_error: 0.0218 - val\_loss: 0.0016 - val\_mean\_absolute\_error: 0.0314  
Epoch 5/80  
44/44 [==============================] - 2s 36ms/step - loss: 8.8216e-04 - mean\_absolute\_error: 0.0215 - val\_loss: 0.0014 - val\_mean\_absolute\_error: 0.0286  
Epoch 6/80  
44/44 [==============================] - 1s 32ms/step - loss: 8.7701e-04 - mean\_absolute\_error: 0.0217 - val\_loss: 8.3662e-04 - val\_mean\_absolute\_error: 0.0230  
Epoch 7/80  
44/44 [==============================] - 1s 33ms/step - loss: 8.7626e-04 - mean\_absolute\_error: 0.0215 - val\_loss: 8.3863e-04 - val\_mean\_absolute\_error: 0.0232  
Epoch 8/80  
44/44 [==============================] - 2s 35ms/step - loss: 8.8388e-04 - mean\_absolute\_error: 0.0217 - val\_loss: 0.0013 - val\_mean\_absolute\_error: 0.0278  
Epoch 9/80  
44/44 [==============================] - 1s 28ms/step - loss: 8.2641e-04 - mean\_absolute\_error: 0.0208 - val\_loss: 7.1216e-04 - val\_mean\_absolute\_error: 0.0213  
Epoch 10/80  
44/44 [==============================] - 1s 29ms/step - loss: 7.6577e-04 - mean\_absolute\_error: 0.0200 - val\_loss: 8.4488e-04 - val\_mean\_absolute\_error: 0.0227  
Epoch 11/80  
44/44 [==============================] - 1s 29ms/step - loss: 7.0894e-04 - mean\_absolute\_error: 0.0193 - val\_loss: 0.0011 - val\_mean\_absolute\_error: 0.0257  
Epoch 12/80  
44/44 [==============================] - 1s 30ms/step - loss: 7.1471e-04 - mean\_absolute\_error: 0.0194 - val\_loss: 9.4791e-04 - val\_mean\_absolute\_error: 0.0240  
Epoch 13/80  
44/44 [==============================] - 1s 29ms/step - loss: 7.0950e-04 - mean\_absolute\_error: 0.0190 - val\_loss: 6.2369e-04 - val\_mean\_absolute\_error: 0.0198  
Epoch 14/80  
44/44 [==============================] - 1s 29ms/step - loss: 6.4927e-04 - mean\_absolute\_error: 0.0186 - val\_loss: 7.0962e-04 - val\_mean\_absolute\_error: 0.0210  
Epoch 15/80  
44/44 [==============================] - 1s 28ms/step - loss: 7.2378e-04 - mean\_absolute\_error: 0.0196 - val\_loss: 6.5635e-04 - val\_mean\_absolute\_error: 0.0207  
Epoch 16/80  
44/44 [==============================] - 2s 38ms/step - loss: 6.5457e-04 - mean\_absolute\_error: 0.0184 - val\_loss: 5.2839e-04 - val\_mean\_absolute\_error: 0.0184  
Epoch 17/80  
44/44 [==============================] - 1s 28ms/step - loss: 6.3168e-04 - mean\_absolute\_error: 0.0185 - val\_loss: 5.3491e-04 - val\_mean\_absolute\_error: 0.0185  
Epoch 18/80  
44/44 [==============================] - 1s 28ms/step - loss: 6.4183e-04 - mean\_absolute\_error: 0.0184 - val\_loss: 6.1726e-04 - val\_mean\_absolute\_error: 0.0195  
Epoch 19/80  
44/44 [==============================] - 1s 28ms/step - loss: 5.9037e-04 - mean\_absolute\_error: 0.0178 - val\_loss: 0.0019 - val\_mean\_absolute\_error: 0.0394  
Epoch 20/80  
44/44 [==============================] - 1s 29ms/step - loss: 6.7476e-04 - mean\_absolute\_error: 0.0191 - val\_loss: 4.7739e-04 - val\_mean\_absolute\_error: 0.0174  
Epoch 21/80  
44/44 [==============================] - 1s 28ms/step - loss: 7.0458e-04 - mean\_absolute\_error: 0.0194 - val\_loss: 5.3096e-04 - val\_mean\_absolute\_error: 0.0182  
Epoch 22/80  
44/44 [==============================] - 1s 29ms/step - loss: 5.8214e-04 - mean\_absolute\_error: 0.0174 - val\_loss: 9.9359e-04 - val\_mean\_absolute\_error: 0.0252  
Epoch 23/80  
44/44 [==============================] - 1s 28ms/step - loss: 5.4168e-04 - mean\_absolute\_error: 0.0168 - val\_loss: 5.3592e-04 - val\_mean\_absolute\_error: 0.0185  
Epoch 24/80  
44/44 [==============================] - 1s 31ms/step - loss: 5.4419e-04 - mean\_absolute\_error: 0.0169 - val\_loss: 0.0013 - val\_mean\_absolute\_error: 0.0302  
Epoch 25/80  
44/44 [==============================] - 2s 34ms/step - loss: 5.7602e-04 - mean\_absolute\_error: 0.0175 - val\_loss: 6.2646e-04 - val\_mean\_absolute\_error: 0.0198  
Epoch 26/80  
44/44 [==============================] - 1s 28ms/step - loss: 5.7752e-04 - mean\_absolute\_error: 0.0175 - val\_loss: 4.4876e-04 - val\_mean\_absolute\_error: 0.0165  
Epoch 27/80  
44/44 [==============================] - 1s 29ms/step - loss: 5.3261e-04 - mean\_absolute\_error: 0.0168 - val\_loss: 4.7946e-04 - val\_mean\_absolute\_error: 0.0170  
Epoch 28/80  
44/44 [==============================] - 1s 28ms/step - loss: 4.8101e-04 - mean\_absolute\_error: 0.0159 - val\_loss: 0.0023 - val\_mean\_absolute\_error: 0.0423  
Epoch 29/80  
44/44 [==============================] - 1s 28ms/step - loss: 6.4785e-04 - mean\_absolute\_error: 0.0186 - val\_loss: 5.3316e-04 - val\_mean\_absolute\_error: 0.0179  
Epoch 30/80  
44/44 [==============================] - 1s 28ms/step - loss: 5.0771e-04 - mean\_absolute\_error: 0.0163 - val\_loss: 8.2661e-04 - val\_mean\_absolute\_error: 0.0231  
Epoch 31/80  
44/44 [==============================] - 1s 28ms/step - loss: 4.7934e-04 - mean\_absolute\_error: 0.0158 - val\_loss: 7.6237e-04 - val\_mean\_absolute\_error: 0.0218  
Epoch 32/80  
44/44 [==============================] - 1s 28ms/step - loss: 4.5836e-04 - mean\_absolute\_error: 0.0155 - val\_loss: 6.4388e-04 - val\_mean\_absolute\_error: 0.0198  
Epoch 33/80  
44/44 [==============================] - 2s 37ms/step - loss: 5.5078e-04 - mean\_absolute\_error: 0.0170 - val\_loss: 3.6852e-04 - val\_mean\_absolute\_error: 0.0148  
Epoch 34/80  
44/44 [==============================] - 1s 28ms/step - loss: 4.4330e-04 - mean\_absolute\_error: 0.0151 - val\_loss: 9.1216e-04 - val\_mean\_absolute\_error: 0.0243  
Epoch 35/80  
44/44 [==============================] - 1s 29ms/step - loss: 4.2255e-04 - mean\_absolute\_error: 0.0150 - val\_loss: 5.4064e-04 - val\_mean\_absolute\_error: 0.0178  
Epoch 36/80  
44/44 [==============================] - 1s 29ms/step - loss: 4.6033e-04 - mean\_absolute\_error: 0.0156 - val\_loss: 3.4758e-04 - val\_mean\_absolute\_error: 0.0144  
Epoch 37/80  
44/44 [==============================] - 1s 28ms/step - loss: 4.1110e-04 - mean\_absolute\_error: 0.0148 - val\_loss: 4.8492e-04 - val\_mean\_absolute\_error: 0.0174  
Epoch 38/80  
44/44 [==============================] - 2s 38ms/step - loss: 5.0623e-04 - mean\_absolute\_error: 0.0163 - val\_loss: 4.7540e-04 - val\_mean\_absolute\_error: 0.0165  
Epoch 39/80  
44/44 [==============================] - 1s 34ms/step - loss: 4.2001e-04 - mean\_absolute\_error: 0.0148 - val\_loss: 3.8241e-04 - val\_mean\_absolute\_error: 0.0151  
Epoch 40/80  
44/44 [==============================] - 1s 28ms/step - loss: 4.4671e-04 - mean\_absolute\_error: 0.0153 - val\_loss: 0.0020 - val\_mean\_absolute\_error: 0.0391  
Epoch 41/80  
44/44 [==============================] - 2s 35ms/step - loss: 4.0731e-04 - mean\_absolute\_error: 0.0145 - val\_loss: 6.3774e-04 - val\_mean\_absolute\_error: 0.0201  
Epoch 42/80  
44/44 [==============================] - 1s 31ms/step - loss: 4.4250e-04 - mean\_absolute\_error: 0.0156 - val\_loss: 3.7307e-04 - val\_mean\_absolute\_error: 0.0148  
Epoch 43/80  
44/44 [==============================] - 1s 28ms/step - loss: 4.6118e-04 - mean\_absolute\_error: 0.0157 - val\_loss: 4.0724e-04 - val\_mean\_absolute\_error: 0.0163  
Epoch 44/80  
44/44 [==============================] - 1s 29ms/step - loss: 3.8013e-04 - mean\_absolute\_error: 0.0140 - val\_loss: 8.9958e-04 - val\_mean\_absolute\_error: 0.0248  
Epoch 45/80  
44/44 [==============================] - 1s 30ms/step - loss: 4.2336e-04 - mean\_absolute\_error: 0.0148 - val\_loss: 3.3753e-04 - val\_mean\_absolute\_error: 0.0139  
Epoch 46/80  
44/44 [==============================] - 1s 29ms/step - loss: 3.5238e-04 - mean\_absolute\_error: 0.0137 - val\_loss: 0.0014 - val\_mean\_absolute\_error: 0.0307  
Epoch 47/80  
44/44 [==============================] - 1s 29ms/step - loss: 4.1419e-04 - mean\_absolute\_error: 0.0145 - val\_loss: 2.9621e-04 - val\_mean\_absolute\_error: 0.0131  
Epoch 48/80  
44/44 [==============================] - 1s 29ms/step - loss: 3.7065e-04 - mean\_absolute\_error: 0.0138 - val\_loss: 3.1539e-04 - val\_mean\_absolute\_error: 0.0138  
Epoch 49/80  
44/44 [==============================] - 1s 29ms/step - loss: 4.0157e-04 - mean\_absolute\_error: 0.0146 - val\_loss: 7.8171e-04 - val\_mean\_absolute\_error: 0.0223  
Epoch 50/80  
44/44 [==============================] - 2s 38ms/step - loss: 4.0568e-04 - mean\_absolute\_error: 0.0145 - val\_loss: 3.5041e-04 - val\_mean\_absolute\_error: 0.0149  
Epoch 51/80  
44/44 [==============================] - 1s 30ms/step - loss: 3.6254e-04 - mean\_absolute\_error: 0.0138 - val\_loss: 4.1134e-04 - val\_mean\_absolute\_error: 0.0154  
Epoch 52/80  
44/44 [==============================] - 1s 28ms/step - loss: 3.8713e-04 - mean\_absolute\_error: 0.0143 - val\_loss: 4.5350e-04 - val\_mean\_absolute\_error: 0.0163  
Epoch 53/80  
44/44 [==============================] - 1s 28ms/step - loss: 3.4060e-04 - mean\_absolute\_error: 0.0132 - val\_loss: 6.4971e-04 - val\_mean\_absolute\_error: 0.0219  
Epoch 54/80  
44/44 [==============================] - 1s 29ms/step - loss: 4.2458e-04 - mean\_absolute\_error: 0.0151 - val\_loss: 5.4694e-04 - val\_mean\_absolute\_error: 0.0182  
Epoch 55/80  
44/44 [==============================] - 1s 29ms/step - loss: 3.2419e-04 - mean\_absolute\_error: 0.0130 - val\_loss: 7.1906e-04 - val\_mean\_absolute\_error: 0.0219  
Epoch 56/80  
44/44 [==============================] - 1s 28ms/step - loss: 3.8248e-04 - mean\_absolute\_error: 0.0140 - val\_loss: 5.8458e-04 - val\_mean\_absolute\_error: 0.0191  
Epoch 57/80  
44/44 [==============================] - 1s 29ms/step - loss: 3.1119e-04 - mean\_absolute\_error: 0.0127 - val\_loss: 0.0012 - val\_mean\_absolute\_error: 0.0293  
Epoch 58/80  
44/44 [==============================] - 2s 35ms/step - loss: 6.2449e-04 - mean\_absolute\_error: 0.0186 - val\_loss: 5.4023e-04 - val\_mean\_absolute\_error: 0.0196  
Epoch 59/80  
44/44 [==============================] - 1s 31ms/step - loss: 3.2211e-04 - mean\_absolute\_error: 0.0128 - val\_loss: 2.9316e-04 - val\_mean\_absolute\_error: 0.0136  
Epoch 60/80  
44/44 [==============================] - 1s 28ms/step - loss: 3.3107e-04 - mean\_absolute\_error: 0.0134 - val\_loss: 2.8684e-04 - val\_mean\_absolute\_error: 0.0125  
Epoch 61/80  
44/44 [==============================] - 1s 29ms/step - loss: 3.0079e-04 - mean\_absolute\_error: 0.0124 - val\_loss: 2.9883e-04 - val\_mean\_absolute\_error: 0.0126  
Epoch 62/80  
44/44 [==============================] - 1s 29ms/step - loss: 3.0578e-04 - mean\_absolute\_error: 0.0126 - val\_loss: 2.6799e-04 - val\_mean\_absolute\_error: 0.0120  
Epoch 63/80  
44/44 [==============================] - 1s 29ms/step - loss: 3.4190e-04 - mean\_absolute\_error: 0.0133 - val\_loss: 2.4409e-04 - val\_mean\_absolute\_error: 0.0119  
Epoch 64/80  
44/44 [==============================] - 1s 28ms/step - loss: 2.6899e-04 - mean\_absolute\_error: 0.0118 - val\_loss: 2.9378e-04 - val\_mean\_absolute\_error: 0.0127  
Epoch 65/80  
44/44 [==============================] - 1s 28ms/step - loss: 2.5192e-04 - mean\_absolute\_error: 0.0113 - val\_loss: 3.2783e-04 - val\_mean\_absolute\_error: 0.0148  
Epoch 66/80  
44/44 [==============================] - 1s 29ms/step - loss: 2.8632e-04 - mean\_absolute\_error: 0.0122 - val\_loss: 2.1273e-04 - val\_mean\_absolute\_error: 0.0110  
Epoch 67/80  
44/44 [==============================] - 2s 38ms/step - loss: 2.5981e-04 - mean\_absolute\_error: 0.0117 - val\_loss: 2.5482e-04 - val\_mean\_absolute\_error: 0.0118  
Epoch 68/80  
44/44 [==============================] - 1s 28ms/step - loss: 2.9723e-04 - mean\_absolute\_error: 0.0125 - val\_loss: 3.0983e-04 - val\_mean\_absolute\_error: 0.0130  
Epoch 69/80  
44/44 [==============================] - 1s 29ms/step - loss: 3.0215e-04 - mean\_absolute\_error: 0.0125 - val\_loss: 2.2836e-04 - val\_mean\_absolute\_error: 0.0119  
Epoch 70/80  
44/44 [==============================] - 1s 28ms/step - loss: 2.4556e-04 - mean\_absolute\_error: 0.0112 - val\_loss: 2.6417e-04 - val\_mean\_absolute\_error: 0.0127  
Epoch 71/80  
44/44 [==============================] - 1s 29ms/step - loss: 2.6411e-04 - mean\_absolute\_error: 0.0116 - val\_loss: 3.3704e-04 - val\_mean\_absolute\_error: 0.0135  
Epoch 72/80  
44/44 [==============================] - 1s 29ms/step - loss: 2.5074e-04 - mean\_absolute\_error: 0.0114 - val\_loss: 2.9354e-04 - val\_mean\_absolute\_error: 0.0125  
Epoch 73/80  
44/44 [==============================] - 1s 28ms/step - loss: 2.4896e-04 - mean\_absolute\_error: 0.0113 - val\_loss: 5.0668e-04 - val\_mean\_absolute\_error: 0.0196  
Epoch 74/80  
44/44 [==============================] - 1s 29ms/step - loss: 2.6188e-04 - mean\_absolute\_error: 0.0116 - val\_loss: 3.6529e-04 - val\_mean\_absolute\_error: 0.0161  
Epoch 75/80  
44/44 [==============================] - 1s 31ms/step - loss: 2.5777e-04 - mean\_absolute\_error: 0.0116 - val\_loss: 2.3837e-04 - val\_mean\_absolute\_error: 0.0123  
Epoch 76/80  
44/44 [==============================] - 2s 34ms/step - loss: 2.3534e-04 - mean\_absolute\_error: 0.0110 - val\_loss: 2.6262e-04 - val\_mean\_absolute\_error: 0.0120  
Epoch 77/80  
44/44 [==============================] - 1s 29ms/step - loss: 2.4270e-04 - mean\_absolute\_error: 0.0113 - val\_loss: 3.0885e-04 - val\_mean\_absolute\_error: 0.0138  
Epoch 78/80  
44/44 [==============================] - 1s 29ms/step - loss: 2.6835e-04 - mean\_absolute\_error: 0.0116 - val\_loss: 2.1677e-04 - val\_mean\_absolute\_error: 0.0114  
Epoch 79/80  
44/44 [==============================] - 1s 29ms/step - loss: 2.2803e-04 - mean\_absolute\_error: 0.0107 - val\_loss: 1.9111e-04 - val\_mean\_absolute\_error: 0.0107  
Epoch 80/80  
44/44 [==============================] - 1s 29ms/step - loss: 2.3668e-04 - mean\_absolute\_error: 0.0111 - val\_loss: 3.2219e-04 - val\_mean\_absolute\_error: 0.0151

<keras.src.callbacks.History at 0x78f66e982800>

**Datasource:** https://finance.yahoo.com/quote/RELIANCE.NS/history

**Making Predictions using the Trained LSTM Model**

This code snippet demonstrates how to use the trained LSTM model to make predictions on the testing sequences.

* Trained the LSTM model using the training sequences and labels.
* Generated testing sequences.

**Model Prediction:**

* We use the predict() method on the trained model (model) to make predictions on the testing sequences (test\_seq).
* The predictions are stored in the test\_predicted variable.

**Displaying Predictions:**

* We display the first 20 predictions by slicing the test\_predicted array (test\_predicted[:20]).

test\_predicted = model.predict(test\_seq) test\_predicted[:20]

10/10 [==============================] - 1s 8ms/step

array([[0.78585 , 0.7785723 ],  
 [0.7777236 , 0.77061045],  
 [0.77904147, 0.7720359 ],  
 [0.7825723 , 0.7755285 ],  
 [0.7759841 , 0.76895595],  
 [0.76667356, 0.7598168 ],  
 [0.7680525 , 0.761332 ],  
 [0.77458626, 0.7678402 ],  
 [0.77665216, 0.7698254 ],  
 [0.77299494, 0.7661869 ],  
 [0.75727147, 0.75066835],  
 [0.74744415, 0.7411733 ],  
 [0.74984246, 0.7437615 ],  
 [0.754857 , 0.7487834 ],  
 [0.75645655, 0.7503696 ],  
 [0.75830036, 0.7522146 ],  
 [0.7630439 , 0.75691336],  
 [0.7642982 , 0.7581033 ],  
 [0.7579893 , 0.75183326],  
 [0.7502604 , 0.74425715]], dtype=float32)

**Datasource:** https://finance.yahoo.com/quote/RELIANCE.NS/history

**Inverse Scaling of Predicted Values**

This code snippet demonstrates how to reverse the scaling applied to the predicted values, restoring them to their original scale.

* Applied scaling to the predicted values.
* Imported and initialized the MinMaxScaler (MMS) used for scaling.

**Inverse Scaling:**

* We use the inverse\_transform() method of the MinMaxScaler (MMS) to reverse the scaling applied to the predicted values.
* This method transforms the scaled values back to their original scale.
* The scaled predicted values (test\_predicted) are passed as input to the inverse\_transform() method.

**Displaying Inverse Scaled Predictions:**

* We display the first 20 inverse scaled predictions by slicing the test\_inverse\_predicted array (test\_inverse\_predicted[:20]).

test\_inverse\_predicted = MMS.inverse\_transform(test\_predicted)  
test\_inverse\_predicted[:20]

array([[2466.193 , 2450.0786],  
 [2445.4983, 2429.773 ],  
 [2448.8545, 2433.4084],  
 [2457.8462, 2442.3157],  
 [2441.0686, 2425.5535],  
 [2417.3582, 2402.2454],  
 [2420.8699, 2406.1094],  
 [2437.5088, 2422.7078],  
 [2442.7698, 2427.7708],  
 [2433.4563, 2418.4912],  
 [2393.4148, 2378.9133],  
 [2368.3884, 2354.6975],  
 [2374.4958, 2361.2983],  
 [2387.266 , 2374.106 ],  
 [2391.3394, 2378.1514],  
 [2396.035 , 2382.857 ],  
 [2408.1147, 2394.8403],  
 [2411.309 , 2397.8752],  
 [2395.2427, 2381.8843],  
 [2375.5603, 2362.5625]], dtype=float32)

**Datasource:** https://finance.yahoo.com/quote/RELIANCE.NS/history

**Concatenating Original and Predicted Data**

This code snippet combines the original Reliance stock data with the predicted open and close prices, after reversing the scaling applied to both the original and predicted data.

* Reversed the scaling applied to the original Reliance stock data.
* Reversed the scaling applied to the predicted open and close prices.
* Imported and initialized the MinMaxScaler (MMS) used for scaling

**Concatenating Data:**

* We use the concat() function from pandas to concatenate the original Reliance stock data (Relince\_stock.iloc[307:].copy()) with a DataFrame containing the inverse scaled predicted open and close prices (pd.DataFrame(test\_inverse\_predicted, columns=['Open\_predicted', 'Close\_predicted'], index=Relince\_stock.iloc[-307:].index)).
* The axis=1 parameter ensures concatenation along the columns axis.

**Inverse Scaling:**

* We use the inverse\_transform() method of the MinMaxScaler (MMS) to reverse the scaling applied to the original 'Open' and 'Close' prices in the concatenated DataFrame (gs\_slic\_data).
* This restores the original scale of the 'Open' and 'Close' prices.

**Displaying Concatenated Data:**

* We display the first 10 rows of the concatenated DataFrame (gs\_slic\_data.head(10)).

gs\_slic\_data=pd.concat([Relince\_stock.iloc[307:].copy(),pd.DataFrame(test\_inverse\_predicted,columns=['Open\_predicted','Close\_predicted'],index=Relince\_stock.iloc[-307:].index)], axis=1)

gs\_slic\_data[['Open','Close']]=MMS.inverse\_transform(gs\_slic\_data[['Open','Close']]) # Inverse scaling

**Displaying Concatenated Data**

This code snippet displays the first 10 rows of the concatenated DataFrame, gs\_slic\_data, which contains the original Reliance stock data along with the predicted open and close prices, after reversing the scaling applied to both datasets.

* Concatenated the original and predicted data.
* Reversed the scaling applied to both datasets.

**Displaying Data:**

* We use the .head(10) method on the DataFrame gs\_slic\_data to display the first 10 rows.
* This provides a preview of the combined dataset, showing the original 'Open' and 'Close' prices alongside the predicted 'Open' and 'Close' prices.

gs\_slic\_data.head(10)

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Open** | **Close** | **Open\_predicted** | **Close\_predicted** |
| **2022-12-12** | 2386.415771 | 2411.890381 | 2466.193115 | 2450.078613 |
| **2022-12-13** | 2411.890381 | 2422.874268 | 2445.498291 | 2429.772949 |
| **2022-12-14** | 2434.550049 | 2414.105713 | 2448.854492 | 2433.408447 |
| **2022-12-15** | 2402.106689 | 2379.816162 | 2457.846191 | 2442.315674 |
| **2022-12-16** | 2373.032227 | 2368.047852 | 2441.068604 | 2425.553467 |
| **2022-12-19** | 2382.262207 | 2399.153076 | 2417.358154 | 2402.245361 |
| **2022-12-20** | 2384.938965 | 2419.920654 | 2420.869873 | 2406.109375 |
| **2022-12-21** | 2419.182129 | 2385.492676 | 2437.508789 | 2422.707764 |
| **2022-12-22** | 2397.953125 | 2379.308594 | 2442.769775 | 2427.770752 |
| **2022-12-23** | 2365.925049 | 2309.529785 | 2433.456299 | 2418.491211 |

**Datasource:** https://finance.yahoo.com/quote/RELIANCE.NS/history

**Plotting Actual vs Predicted Open Prices**

This code snippet creates a plot to visualize the actual and predicted open prices of Reliance stock over time.

* Concatenated the original and predicted data into the DataFrame gs\_slic\_data.
* Imported and initialized the necessary libraries, including Matplotlib.

**Plotting Data:**

* We use the .plot() method on the DataFrame gs\_slic\_data to create a line plot.
* We specify the columns to plot as 'Open' (actual open prices) and 'Open\_predicted' (predicted open prices).

**Customizing Plot:**

* We set the figure size to (20,10) using figsize=(20,10) to control the dimensions of the plot.
* The plt.xticks(rotation=45) statement rotates the x-axis labels by 45 degrees for better readability.
* We set the x-axis label as 'Date' and the y-axis label as 'Stock Price' using plt.xlabel() and plt.ylabel() respectively.
* The title of the plot is set to 'Actual vs Predicted for open price' using plt.title().

**Displaying Plot:**

* We use plt.show() to display the plot.

gs\_slic\_data[['Open','Open\_predicted']].plot(figsize=(20,10))  
plt.xticks(rotation=45)  
plt.xlabel('Date',size=40)  
plt.ylabel('Stock Price',size=50)  
plt.title('Actual vs Predicted for open price',size=40)  
plt.show()

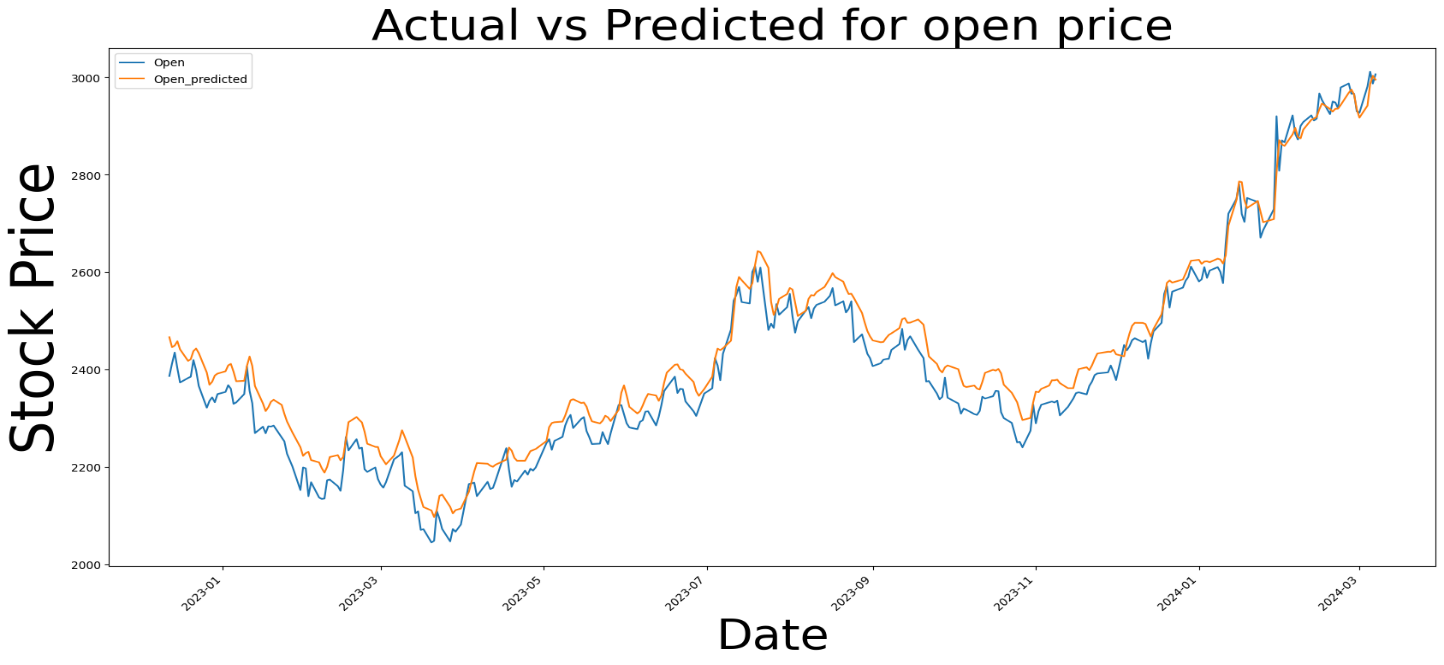


Figure 4.2

From the study of In Figure 4.2, we illustrate the seven-year data for Reliance Industries Limited (NSE: RELIANCE) by showcasing the difference between the opening price and the predicted price on a day-to-day basis. This depiction provides insights into the variation between actual and forecasted opening prices over time.

**Datasource:** https://finance.yahoo.com/quote/RELIANCE.NS/history

**Plotting Actual vs Predicted Close Prices**

This code snippet creates a plot to visualize the actual and predicted close prices of Reliance stock over time.

* Concatenated the original and predicted data into the DataFrame gs\_slic\_data.
* Imported and initialized the necessary libraries, including Matplotlib.

**Plotting Data:**

* We use the .plot() method on the DataFrame gs\_slic\_data to create a line plot.
* We specify the columns to plot as 'Close' (actual close prices) and 'Close\_predicted' (predicted close prices).

**Customizing Plot:**

* We set the figure size to (20,10) using figsize=(20,10) to control the dimensions of the plot.
* The plt.xticks(rotation=45) statement rotates the x-axis labels by 45 degrees for better readability.
* We set the x-axis label as 'Date' and the y-axis label as 'Stock Price' using plt.xlabel() and plt.ylabel() respectively.
* The title of the plot is set to 'Actual vs Predicted for close price' using plt.title().

**Displaying Plot:**

* We use plt.show() to display the plot.

gs\_slic\_data[['Close','Close\_predicted']].plot(figsize=(20,10))  
plt.xticks(rotation=45)  
plt.xlabel('Date',size=40)  
plt.ylabel('Stock Price',size=40)  
plt.title('Actual vs Predicted for close price',size=45)  
plt.show()

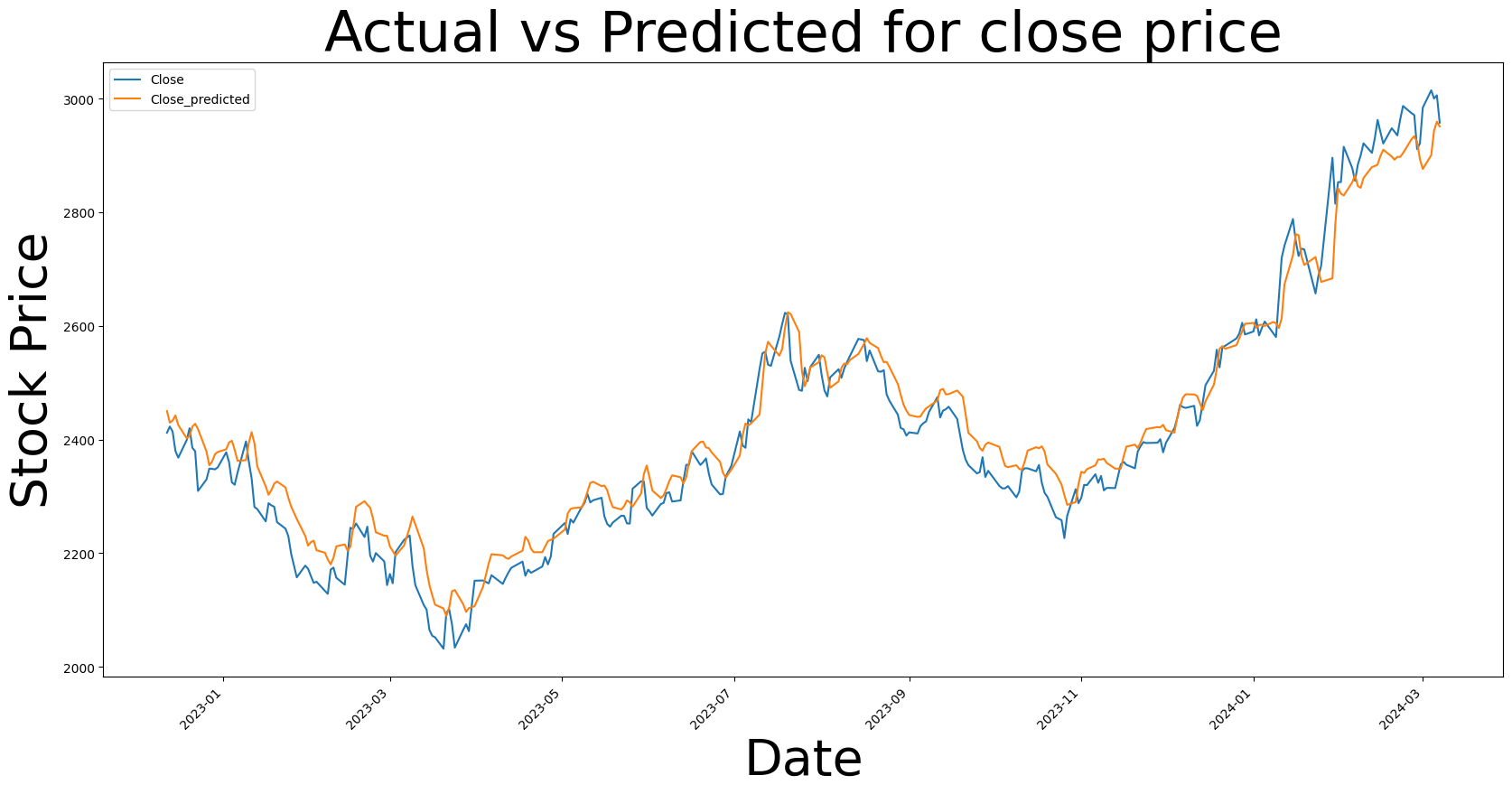


Figure 4.3

From the study of Figure 4.3 demonstrates the seven-year data for Reliance Industries Limited (NSE: RELIANCE) by highlighting the difference between the actual closing price and the predicted price on a day-to-day basis. This representation offers insights into the discrepancies between forecasted and realized closing prices over the specified period.

**Datasource:** https://finance.yahoo.com/quote/RELIANCE.NS/history

**Creating a dataframe and adding 10 days to existing index**

**Appending Empty Rows to the DataFrame**

This code snippet demonstrates how to append empty rows to the DataFrame gs\_slic\_data. Empty rows can be useful for extending the DataFrame's time series index or for preparing the DataFrame to accommodate future data.

* Concatenated the original and predicted data into the DataFrame gs\_slic\_data.
* Imported and initialized the necessary libraries, including pandas.

**Appending Empty Rows:**

* We use the append() method on the DataFrame gs\_slic\_data to append new rows.
* We create a new DataFrame with the same columns as gs\_slic\_data and an index generated using pd.date\_range() starting from the last index of gs\_slic\_data.
* The columns=gs\_slic\_data.columns parameter ensures that the new DataFrame has the same columns as gs\_slic\_data.
* We specify the number of periods as 11 (periods=11) and the frequency as 'D' (days) using freq='D'.
* The closed='right' parameter ensures that the frequency intervals are closed on the right side.

# Creating a dataframe and adding 10 days to existing index  
  
gs\_slic\_data=gs\_slic\_data.append(pd.DataFrame(columns=gs\_slic\_data.columns,index=pd.date\_range(start=gs\_slic\_data.index[-1], periods=11, freq='D', closed='right')))

<ipython-input-25-238941946f56>:3: FutureWarning: Argument `closed` is deprecated in favor of `inclusive`.  
 gs\_slic\_data = gs\_slic\_data.append(pd.DataFrame(columns=gs\_slic\_data.columns,index=pd.date\_range(start=gs\_slic\_data.index[-1], periods=11, freq='D', closed='right')))  
<ipython-input-25-238941946f56>:3: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.

**Selecting Data within a Date Range**

This code snippet demonstrates how to select data within a specific date range from the DataFrame gs\_slic\_data.

* Appended empty rows to the DataFrame gs\_slic\_data.
* Imported and initialized the necessary libraries, including pandas.

**Appending Empty Rows:**

* We use the append() method on the DataFrame gs\_slic\_data to append new rows.
* We create a new DataFrame with the same columns as gs\_slic\_data and an index generated using pd.date\_range() starting from the last index of gs\_slic\_data.
* The columns=gs\_slic\_data.columns parameter ensures that the new DataFrame has the same columns as gs\_slic\_data.
* We specify the number of periods as 11 (periods=11) and the frequency as 'D' (days) using freq='D'.
* The closed='right' parameter ensures that the frequency intervals are closed on the right side.

**Selecting Data within Date Range:**

* We use slicing to select data within the date range from '2024-02-21' to '2024-03-07'.
* The syntax gs\_slic\_data['2024-02-21':'2024-03-07'] filters the DataFrame to include only rows with index dates falling within this range.

gs\_slic\_data=gs\_slic\_data.append(pd.DataFrame(columns=gs\_slic\_data.columns,index=pd.date\_range(start=gs\_slic\_data.index[-1], periods=11, freq='D', closed='right')))

gs\_slic\_data['2024-02-21 ':'2024-03-07']

| **Open** | **Close** | **Open\_predicted** | **Close\_predicted** |
| --- | --- | --- | --- |
| **2024-02-21** | 2948.000000 | 2935.399902 | 2935.104980 | 2897.686035 |
| **2024-02-22** | 2936.300049 | 2963.500000 | 2935.555664 | 2897.491211 |
| **2024-02-23** | 2979.000000 | 2987.250000 | 2942.947998 | 2904.315674 |
| **2024-02-26** | 2987.100098 | 2974.649902 | 2968.062988 | 2928.536621 |
| **2024-02-27** | 2966.050049 | 2971.300049 | 2974.554932 | 2934.132568 |
| **2024-02-28** | 2966.000000 | 2911.250000 | 2964.465820 | 2923.673584 |
| **2024-02-29** | 2930.000000 | 2921.600098 | 2934.300781 | 2893.548096 |
| **2024-03-01** | 2927.000000 | 2984.250000 | 2916.899658 | 2876.337402 |
| **2024-03-04** | 2980.949951 | 3014.800049 | 2941.628662 | 2900.693359 |
| **2024-03-05** | 3011.550049 | 3000.399902 | 2986.009033 | 2943.842773 |
| **2024-03-06** | 2986.899902 | 3006.000000 | 3002.961914 | 2959.625244 |
| **2024-03-07** | 3005.949951 | 2957.850098 | 2994.973633 | 2951.295898 |

**Datasource:** https://finance.yahoo.com/quote/RELIANCE.NS/history

**Making Upcoming Predictions**

This code snippet prepares to make predictions for upcoming data points based on the current model's state and previous predictions.

* Imported and initialized the necessary libraries, including pandas.
* Defined the model and prepared the test sequences.

**Initializing DataFrame for Predictions:**

* We create a new DataFrame named upcoming\_prediction with columns for 'Open' and 'Close', and an index matching that of gs\_slic\_data.
* The index of upcoming\_prediction is converted to datetime format using pd.to\_datetime().

**Preparing Current Sequence for Prediction:**

* We select the last sequence of the test data (curr\_seq) to use as a starting point for predicting the next data point.

**Looping Through Future Predictions:**

* We iterate through the range from -10 to -1 (representing the last 10 days in the upcoming\_prediction DataFrame).

**For each iteration:**

* We predict the next 'Open' and 'Close' prices using the model (model.predict(curr\_seq)).
* The predicted values are stored in the upcoming\_prediction DataFrame at the corresponding index position.
* The curr\_seq is updated by removing the first element and appending the predicted values.
* The shape of curr\_seq is adjusted to match the shape of the last sequence in the test data.

upcoming\_prediction=pd.DataFrame(columns=['Open','Close'],index=gs\_slic\_data.index)upcoming\_prediction.index=pd.to\_datetime(upcoming\_prediction.index)

curr\_seq = test\_seq[-1:]  
  
for i in range(-10,0):  
 up\_pred = model.predict(curr\_seq)  
 upcoming\_prediction.iloc[i] = up\_pred  
 curr\_seq = np.append(curr\_seq[0][1:],up\_pred,axis=0)  
 curr\_seq = curr\_seq.reshape(test\_seq[-1:].shape)

1/1 [==============================] - 0s 20ms/step  
1/1 [==============================] - 0s 28ms/step  
1/1 [==============================] - 0s 17ms/step  
1/1 [==============================] - 0s 17ms/step  
1/1 [==============================] - 0s 16ms/step  
1/1 [==============================] - 0s 18ms/step  
1/1 [==============================] - 0s 16ms/step  
1/1 [==============================] - 0s 17ms/step  
1/1 [==============================] - 0s 16ms/step  
1/1 [==============================] - 0s 17ms/step

**Datasource:** https://finance.yahoo.com/quote/RELIANCE.NS/history

**Visualizing Upcoming Open Price Predictions**

This code snippet plots the current and predicted upcoming open prices of Reliance stock to visualize the model's predictions.

* Imported and initialized the necessary libraries, including pandas and Matplotlib.
* Defined the DataFrame upcoming\_prediction containing the predicted upcoming open prices.

**Inverse Scaling:**

* We use the inverse\_transform() method of the MinMaxScaler (MMS) to reverse the scaling applied to the predicted 'Open' prices in the upcoming\_prediction DataFrame.
* This restores the original scale of the 'Open' prices.

**Plotting Data:**

* We create a plot with two lines representing the current and predicted upcoming open prices.
* The current open prices are plotted from the gs\_slic\_data DataFrame, starting from the date '2021-04-25'.
* The predicted upcoming open prices are plotted from the upcoming\_prediction DataFrame, starting from the same date.

**Customizing Plot:**

* We customize the plot by rotating the x-axis tick labels by 45 degrees for better readability using plt.setp(ax.xaxis.get\_majorticklabels(), rotation=45).
* We set the x-axis label as 'Date' and the y-axis label as 'Stock Price'.
* The title of the plot is set to 'Upcoming Open price prediction'.

**Displaying Plot:**

* We display the plot using plt.show().

upcoming\_prediction[['Open','Close']]=MMS.inverse\_transform(upcoming\_prediction[['Open','Close']])

fg,ax=plt.subplots(figsize=(10,5))  
ax.plot(gs\_slic\_data.loc['2021-04-25':,'Open'],label='Current Open Price')  
ax.plot(upcoming\_prediction.loc['2021-04-25':,'Open'],label='Upcoming Open Price')  
plt.setp(ax.xaxis.get\_majorticklabels(), rotation=45)  
ax.set\_xlabel('Date',size=15)  
ax.set\_ylabel('Stock Price',size=15)  
ax.set\_title('Upcoming Open price prediction',size=15)  
ax.legend()  
fg.show()

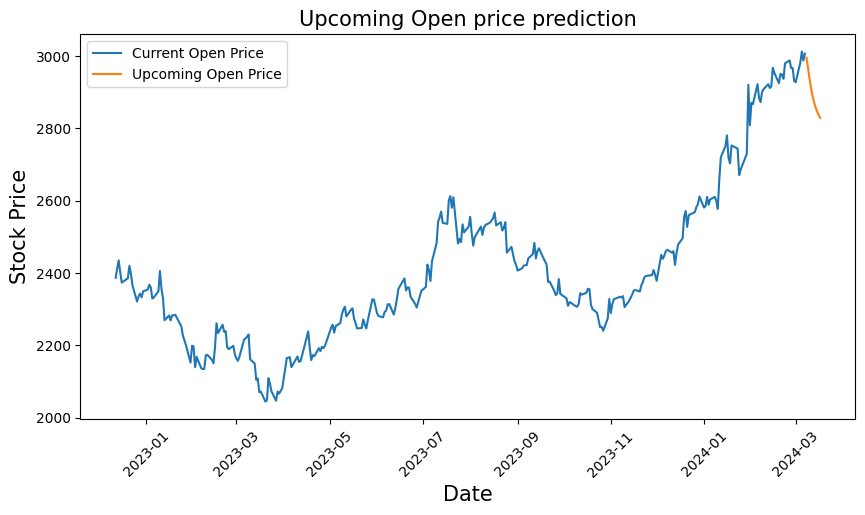


Figure 4.4

From the study of In Figure 4.4, we utilized the LSTM method to predict the future trends of Reliance Industries Limited (NSE: RELIANCE). The current opening price is depicted by the blue line, while the orange line represents the prediction indicating a potential decline of Rs. 200 per share in the company's stock over the next 15 days.

**Datasource:** https://finance.yahoo.com/quote/RELIANCE.NS/history

**Visualizing Upcoming Close Price Predictions**

This code snippet plots the current and predicted upcoming close prices of Reliance stock to visualize the model's predictions.

* Imported and initialized the necessary libraries, including pandas and Matplotlib.
* Defined the DataFrame upcoming\_prediction containing the predicted upcoming close prices.

**Plotting Data:**

* We create a plot with two lines representing the current and predicted upcoming close prices.
* The current close prices are plotted from the gs\_slic\_data DataFrame, starting from the date '2021-04-25'.
* The predicted upcoming close prices are plotted from the upcoming\_prediction DataFrame, starting from the same date.

**Customizing Plot:**

* We customize the plot by rotating the x-axis tick labels by 45 degrees for better readability using plt.setp(ax.xaxis.get\_majorticklabels(), rotation=45).
* We set the x-axis label as 'Date' and the y-axis label as 'Stock Price'.
* The title of the plot is set to 'Upcoming Close price prediction'.

**Displaying Plot:**

* We display the plot using plt.show().

fg,ax=plt.subplots(figsize=(10,5))  
ax.plot(gs\_slic\_data.loc['2021-04-25':,'Close'],label='Current Close Price')  
ax.plot(upcoming\_prediction.loc['2021-04-25':,'Close'],label='Upcoming close Price')  
plt.setp(ax.xaxis.get\_majorticklabels(), rotation=45)  
ax.set\_xlabel('Date',size=15)  
ax.set\_ylabel('Stock Price',size=15)  
ax.set\_title('Upcoming close price prediction',size=15)  
ax.legend()  
fg.show()

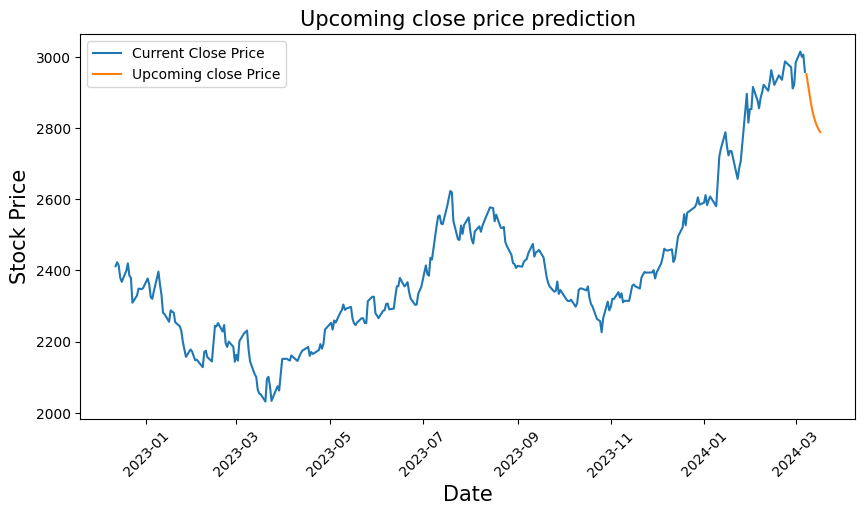


Figure 4.5

From the study of In Figure 4.5 we applied the LSTM method to forecast the future trends of Reliance Industries Limited (NSE: RELIANCE). The current closing price is represented by the blue line, while the orange line indicates the prediction, suggesting a potential decline of Rs. 200 per share in the company's stock over the next 15 days.

**Measure Findings**

**4.8 Conclusion**

According to our analysis and market predictions, it has been forecasted that the share value of Reliance Industries Limited on the National Stock Exchange (NSE) is expected to undergo a decline of Rs. 200 by March 20, 2024. This projection is based on a thorough examination of various market indicators and trends, as well as the current economic landscape. Investors are advised to consider this forecast alongside other pertinent information and conduct their own due diligence before making any investment decisions.

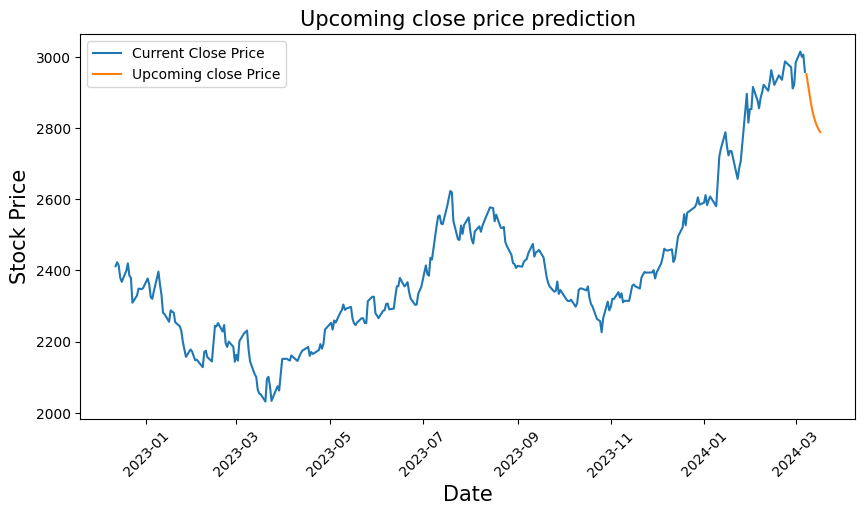


Figure 4.4

From the study of In Figure 4.4, we utilized the LSTM method to predict the future trends of Reliance Industries Limited (NSE: RELIANCE). The current opening price is depicted by the blue line, while the orange line represents the prediction indicating a potential decline of Rs. 200 per share in the company's stock over the next 15 days.

Designing and developing an optimised framework for time series data prediction using deep learning is a complex task that requires careful consideration of several factors. Here are some key points to consider:

**Data preprocessing:** Time series data can be noisy and may contain missing values or outliers. Therefore, it is important to preprocess the data to ensure that it is clean, consistent, and ready for analysis. This can involve techniques such as imputation, normalization, and feature engineering.

**Model selection:** There are many deep learning models that can be used for time series data prediction, such as recurrent neural networks (RNNs), long short-term memory (LSTM) networks, and gated recurrent units (GRUs). Each model has its strengths and weaknesses, and the selection of a suitable model depends on the specific requirements of the project.

**Hyperparameter tuning:** Deep learning models have many hyperparameters that need to be tuned to achieve optimal performance. This can be a time-consuming and iterative process that involves trying different values for the hyperparameters and evaluating the performance of the model on a validation set.

**Regularization:** Regularization techniques, such as dropout or L2 regularization, can be used to prevent overfitting and improve the generalization of the model.

**Ensemble learning:** Combining the predictions of multiple models can improve the accuracy and robustness of the predictions. Ensemble learning techniques such as bagging or boosting can be used for this purpose.

**Evaluation metrics:** There are several metrics that can be used to evaluate the performance of a time series prediction model, such as mean absolute error (MAE), root mean squared error (RMSE), or coefficient of determination (R-squared). The choice of metric depends on the specific requirements of the project and the characteristics of the data.

Overall, designing and developing an optimized framework for time series data prediction using deep learning requires a deep understanding of the data, the models, and the evaluation metrics. It is an iterative process that involves testing and refining the model until it achieves optimal performance.

# **Limitation**

Despite the advantages of designing and developing an optimized framework for time series data prediction using deep learning, there are several limitations that should be considered:

**Data availability:** The quality and quantity of the data can have a significant impact on the performance of the prediction model. In some cases, there may be limited or incomplete data available, making it difficult to train an accurate model.

**Overfitting:** Deep learning models are highly flexible and can easily overfit the training data if the model is too complex or if the training data is not representative of the test data.

**Computational resources:** Deep learning models are computationally intensive and require a significant amount of resources, including powerful processors and high-capacity memory, which can limit their scalability and accessibility.

**Interpretability:** Deep learning models are often described as "black boxes" because it can be difficult to interpret how the model is making its predictions. This can limit the ability to explain the results to stakeholders or to identify the root cause of errors.

**Model maintenance:** Once a model has been trained and deployed, it may require ongoing maintenance and updates to ensure that it remains accurate and reliable over time. This can require additional resources and may pose challenges for organisations that do not have the necessary expertise or resources to support the model.

Thus designing and developing an optimised framework for time series data prediction using deep learning can be a powerful tool for organisations seeking to improve their forecasting capabilities. However, it is important to consider the limitations of the approach and to carefully assess whether it is a suitable solution for the specific requirements of the project.

# **Limitation specific to Equity Market prediction**

Designing and developing an optimised framework for time series data prediction using deep learning for stock price has some additional limitations and challenges, including:

**Market volatility:** Stock prices can be highly volatile and subject to sudden changes, which can make it difficult to accurately predict future prices using historical data.

**Non-stationarity:** Stock prices may exhibit non-stationary behavior, such as changing trends, seasonal patterns, or sudden shifts in the underlying economic conditions. This can make it difficult to identify and model the underlying patterns in the data.

**Limited historical data:** While stock prices have a long history, the availability and quality of historical data may be limited, particularly for newly listed stocks or for companies that have undergone significant changes over time.

**External factors:** Stock prices can be influenced by a wide range of external factors, such as news events, regulatory changes, or economic indicators, which may not be captured by the historical data or the model.

**Model performance:** Even with an optimised framework, the performance of the prediction model may be limited by the quality and quantity of the data, the complexity of the model, and the unpredictability of the market.

So, while designing and developing an optimised framework for time series data prediction using deep learning for stock price can provide valuable insights and improve forecasting accuracy, it is important to recognize the limitations of the approach and to supplement the model with other sources of information and expert judgement.

# **4.9 Conclusion:**

In conclusion, designing and developing an optimised framework for time series data prediction using deep learning for stock price can be a powerful tool for improving forecasting accuracy and gaining valuable insights into the behaviour of the stock market. However, it is important to recognize the limitations and challenges associated with this approach, including market volatility, non-stationarity, limited historical data, external factors, and model performance.

To overcome these limitations, it is important to supplement the deep learning model with other sources of information and expert judgement, such as fundamental analysis, technical analysis, and market sentiment analysis. In addition, ongoing maintenance and updates to the model may be required to ensure that it remains accurate and reliable over time.

Overall, designing and developing an optimised framework for time series data prediction using deep learning for stock price can provide significant benefits, but it requires a careful and nuanced approach that takes into account the unique characteristics of the stock market and the limitations of the available data and modelling techniques.

# **4.10 Suggession:**

In order to extend present work in near future and enhancements towards designing and improving proposed framework following recommendations are suggested.

**Incorporating external factors:** To improve the accuracy of the prediction model, future work can focus on incorporating external factors such as news events, social media sentiment, and economic indicators into the model. This can help to capture the impact of external factors on the stock market and provide more accurate forecasts.

***Transfer learning:***Transfer learning is a technique that can be used to leverage pre-trained models on other similar datasets to improve the accuracy of the prediction model on a target dataset. Future work can focus on exploring the effectiveness of transfer learning for stock price prediction using deep learning.

***Hybrid models:***Combining multiple models such as deep learning, machine learning, and statistical models can help to improve the accuracy of the prediction model. Future work can focus on developing hybrid models that leverage the strengths of different modeling techniques.

***terpretable models:***To address the issue of interpretability, future work can focus on developing interpretable deep learning models for stock price prediction. This can help to explain the underlying patterns and factors that are driving the predictions and provide more transparency to stakeholders.

***Real-time prediction:***Real-time prediction can be a valuable tool for traders and investors who need to make quick decisions based on the latest market information. Future work can focus on developing real-time prediction models that can provide accurate and reliable forecasts in real-time.

In other words, there are many opportunities for future work and enhancements towards designing and developing an optimized framework for time series data prediction using deep learning for stock price. By addressing the limitations and challenges of the current approach and incorporating new techniques and data sources, it.

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Actual Price | Predict Price | Difference |
| 12-03-2024 | 2933 | 2905 | -28 |
| 13-03-2024 | 2959 | 2947 | -12 |
| 14-03-2024 | 2879 | 2877 | -2 |
| 15-03-2024 | 2851 | 2839 | -12 |
| 18-03-2024 | 2840 | 2795 | -45 |

**Reference**

1. Mahmoud A., Mohammed A., A Survey on Deep Learning for Time-Series Forecasting. Machine Learning and Big Data Analytics Paradigms: Analysis, Applications and Challenges, pp. 365–392, Dec. 2020.
2. Sengupta, S., et al.: A Review of deep learning with special emphasis on architectures. Applications and Recent Trends. arXiv preprint arXiv:1905.13294 (2019)
3. Tokui, S., Oono, K., Hido, S., Clayton, J.: Chainer: a next-generation open source framework for deep learning. In: Proceedings of Workshop on Machine Learning Systems (LearningSys) in the Twenty-Ninth Annual Conference on Neural Information Processing Systems (NIPS), vol. 5, pp.
4. Cavalcante, R.C., Brasileiro, R.C., Souza, V.L., Nobrega, J.P., Oliveira, A.L.: Computational intelligence and financial markets: A survey and future directions. Expert Syst. Appl. 55, 194–211 (2016)
5. Vellido, A., Lisboa, P.J., Vaughan, J.: Neural networks in business: a survey of applications (1992–1998). Expert Syst. Appl. 17(1), 51–70 (1999)
6. Kim, K.-J., Ahn, H.: Simultaneous optimization of artificial neural networks for financial forecasting. Appl. Intell. 36(4), 887–898 (2012)
7. Adebiyi, A.A., Adewumi, A.O., Ayo, C.K.: Comparison of ARIMA and artificial neural networks models for stock price prediction. J. Appl. Math. 2014 (2014)
8. Göçken, M., Özçalıcı, M., Boru, A., Dosdoğru, A.T.: Integrating metaheuristics and artificial neural networks for improved stock price prediction. Expert Syst. Appl. 44, 320–331 (2016)
9. Lu, C.-J., Lee, T.-S., Chiu, C.-C.: Financial time series forecasting using independent component analysis and support vector regression. Decis. Support Syst. 47(2), 115–125 (2009)
10. Hossain, M.A., Karim, R., Thulasiram, R., Bruce, N.D., Wang, Y.: Hybrid deep learning model for stock price prediction. In: 2018 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1837–1844. IEEE (2018)
11. Siami-Namini, S., Namin, A.S.: Forecasting economics and financial time series: Arima vs. LSTM. arXiv preprint arXiv:1803.06386 (2018)
12. Fischer, T., Krauss, C.: Deep learning with long short-term memory networks for financial market predictions. Eur. J. Oper. Res. 270(2), 654–669 (2018)
13. dos Santos Pinheiro, L., Dras, M.: Stock market prediction with deep learning: a character-based neural language model for event-based trading. In: Proceedings of the Australasian Language Technology Association Workshop 2017, pp. 6–15 (2017) n, K., Zhou, Y., Dai, F.: A LSTM-based method for stock returns prediction: a case study of China stock market. In: 2015 IEEE International Conference on Big Data (Big Data), pp. 2823–2824. IEEE (2015)
14. Ding, X., Zhang, Y., Liu, T., Duan, J.: Deep learning for event-driven stock prediction. In: Twenty-Fourth International Joint Conference on Artificial Intelligence (2015)
15. Chen, W., Zhang, Y., Yeo, C.K., Lau, C.T., Lee, B.S.: Stock market prediction using neural network through news on online social networks. In: 2017 International Smart Cities Conference (ISC2), pp. 1–6. IEEE (2017)
16. Duan, Y., Lv, Y., Wang, F.-Y.: Travel time prediction with LSTM neural network. In: 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), pp. 1053–1058. IEEE (2016)
17. Yang, J., Nguyen, M.N., San, P.P., Li, X.L., Krishnaswamy, S.: Deep convolutional neural networks on multichannel time series for human activity recognition. In: Twenty-Fourth International Joint Conference on Artificial Intelligence (2015)
18. Mehdiyev, N., Lahann, J., Emrich, A., Enke, D., Fettke, P., Loos, P.: Time series classification using deep learning for process planning: a case from the process industry. Procedia Comput. Sci. 114, 242–249 (2017)
19. Di Persio, L., Honchar, O.: Artificial neural networks approach to the forecast of stock market price movements. Int. J. Econ. Manag. Syst. 1 (2016)
20. Ke, J., Zheng, H., Yang, H., Chen, X.M.: Short-term forecasting of passenger demand under on-demand ride services: a spatio-temporal deep learning approach. Transp. Res. Part C: Emerg. Technol. 85, 591–608 (2017)