CHAPTER 1
INTRODUCTION

1.1 INTRODUCTION TO THE PROJECT

The subject of stock price prediction has interested investors, researchers, and data scientists. Effective stock price forecasting can provide an investment strategy with a competitive advantage, thus making the decision-making process more effective and risk management easier. However, the complexity in predicting stock prices arises due to the dynamic, non-linear, and volatile nature of financial markets. This unpredictability is further fueled by external factors such as economic events, market sentiment, and global trends. Stock price forecasting is challenging and intellectually stimulating. Time series forecasting is important in the study of the temporal patterns of stock price movements. Traditional statistical models like ARIMA have been widely used for this purpose, using historical price data to predict future trends. But most of these Models do not catch the non-linearity or complicated market dynamics. The advanced machine learning techniques and deep learning have surfaced with far more complex tools that would potentially reveal intricate patterns and dependency in financial time series. In this project, we are going through a full spectrum of prediction techniques which will be followed for forecasting stock prices in terms of both classical and contemporary ones. We start with ARIMA for modeling linear patterns and then move to machine learning models such as Random Forest and XG-Boost when dealing with non-linear relations. Finally, we shift to deep learning methods in the form of CNNs, RNNs, and LSTMs to capture sequential dependencies. This project compares these methods based on various metrics, such as accuracy and robustness. It provides insights into how applicable and effective these methods are in stock price forecasting. Our work not only highlights the strengths and limitations of each approach but also serves as a valuable resource for analysts and practitioners who seek to adopt data driven strategies in the financial domain.

1.2 STATEMENT OF THE PROBLEM

In this project, we aim to predict stock prices by using a variety of time series forecasting techniques. Stock price prediction is a challenging task due to the dynamic and volatile nature of financial markets. Our aim is to explore and compare the effectiveness of several models, ranging from classical statistical approaches to advanced machine learning and deep learning techniques. We begin by applying AR, MA, ARIMA (Auto-Regressive Integrated Moving Average), a traditional time series method known for its ability to capture linear patterns. Next, we employ Regression and ensemble models like XG-Boost and Random Forest, which offer flexibility in handling both linear and nonlinear relationships in the data. To push the boundaries further, we integrate deep learning techniques such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Long Short-Term Memory (LSTM) networks, known for their ability to capture sequential dependencies in time series data. Through this multi-technique approach, we evaluate the models based on their predictive accuracy, robustness to market fluctuations, and overall performance in stock price forecasting. Our findings highlight the advantages and limitations of each method, offering valuable insights for financial analysts and traders looking to make data-driven investment decisions.

1.3 SYSTEM SPECIFICATIONS

This section outlines the hardware and software requirements used to develop the stock price prediction system. These specifications ensure the system is designed to handle data preprocessing, model training, evaluation, and deployment efficiently.

1.3.1 Hardware Requirements

The system requires robust hardware to handle large datasets, train Deep learning models, and perform computational tasks.

The following specifications are recommended:

- Processor: Intel Core i5 (10th gen or later) or AMD Ryzen 5 (3000 series or later) (6 cores/12 threads or more recommended)
- RAM: 16 GB (minimum), 32 GB (recommended for large datasets)
- Storage: 512 GB SSD (minimum), 1 TB SSD (preferred)
- GPU: NVIDIA GeForce RTX 3060 (6GB VRAM or more) or equivalent AMD Radeon GPU.
- Operating System: Windows 10 (20H2 or later), Windows 11, macOS (latest stable version), or Linux (Ubuntu 20.04 LTS or later).

1.3.2 Software Requirements

The system leverages various software tools and libraries for data preprocessing, feature engineering, and predictive modelling:

- Programming Language: Python 3.9 or later.
- IDE: Visual Studio Code, Google Colab or PyCharm
- Key Libraries:
  - Data Manipulation and Analysis: Pandas, NumPy.
  - Data Visualization: Matplotlib, Seaborn.
  - Statistics Models: statsmodels.
  - Machine Learning: scikit-learn.
  - Time Series Prediction Models: XG-Boost, Random-Forest, TensorFlow.
- Deployment Platform: Stream-lit for interactive dashboards.

CHAPTER 2
LITERATURE SURVEY

1. “Profit Prediction Using ARIMA, SARIMA and LSTM Models in Time Series Forecasting: A Comparison” by Uppala Meena Sirisha, Manjula C Belavagi & Girija Attigeri.

This paper compares ARIMA, SARIMA, and LSTM models for profit prediction. The study emphasizes the strengths and weaknesses of these models in time series forecasting, particularly in terms of predictive accuracy and computational efficiency. LSTM was found to outperform ARIMA and SARIMA in handling non-linear patterns due to its ability to learn from sequential data and capture dependencies across time steps.

ARIMA is a statistical approach for time series forecasting that models the relationship between an observation and its lagged values, including error terms from previous predictions. It is ideal for non-seasonal data and focuses on making non-stationary data stationary before applying the model.
Key Components:

- AR (Auto-Regressive) Order (p): Represents the number of lag observations included in the model.
- I (Integrated) Order (d): Specifies the number of differencing required to make the series stationary.
- MA (Moving Average) Order (q): Indicates the number of lagged forecast errors in the model.

SARIMA enhances ARIMA by incorporating seasonal patterns, making it suitable for data with periodic fluctuations. It introduces additional parameters to model seasonality.
Additional Parameters:

- P (Seasonal AR Order): Number of seasonal autoregressive terms.
- D (Seasonal Differencing Order): Number of seasonal differences required.
- Q (Seasonal MA Order): Number of seasonal moving average terms.
- S (Seasonality Period): Defines the periodicity of the data.

LSTM is a deep learning approach designed to capture long-term dependencies in sequential data. It excels in time series forecasting due to its ability to model complex patterns, trends, and seasonality dynamically.
Architecture:

- LSTM Layer: Stores and updates memory over time steps. It uses gates (input, forget, and output) to manage the flow of information.
- Dense Layer: Outputs the final prediction.

This paper examines the performance of ARIMA, SARIMA, and LSTM models in profit prediction for time series data.

- ARIMA & SARIMA Models: These statistical models are shown to perform well for data with linear trends and seasonality. However, they struggle with non-linear patterns and long-term dependencies.
- LSTM Model: As a neural network-based model, LSTM can capture complex, non-linear relationships and dependencies over time, making it ideal for time series forecasting.
- Comparison: The paper concludes that LSTM outperforms ARIMA and SARIMA in accuracy for datasets with non-linear trends, though it requires more computational power and tuning expertise.
- Applications: Demonstrates the usability of LSTM in financial forecasting tasks where traditional statistical models fail to generalize well.

2. "On the Autoregressive Time Series Model Using Real and Complex Analysis", by Torsten Ullrich

This paper delves into the global structure of autoregressive models using real and complex analysis. It highlights that autoregressive models can be understood as combinations of sinusoidal, exponential, and polynomial functions, aiding in the model selection process. These insights help avoid unsuccessful modeling attempts by ensuring compatibility with the data's global properties.

The Autoregressive (AR) model is a fundamental time series analysis method that predicts a data point based on its past values and a stochastic error term. It's suitable for modeling short-term relationships within a stationary time series.
Key Features

- Stationarity Requirement: The series should have constant statistical properties over time.
- Short-Term Dependency: Captures relationships between recent data points.

This research explores autoregressive (AR) models using both real and complex mathematical frameworks.

- Global and Local Structure: The authors analyse how AR models derive their short-term predictions from previous values but often lack explicit global structural insights.
- Complex Analysis: By combining polynomial, sinusoidal, and exponential functions, the paper highlights how AR models' global behaviour can be better understood.
- Applications: This approach aids in model selection, ensuring that the chosen model aligns with the data's inherent properties.
- Key Contribution: Offers theoretical insights into how complex analysis can refine AR models, reducing the risk of poor model selection and providing a mathematical basis for understanding temporal patterns.

3. “XGBoost: A Scalable Tree Boosting System” by Tianqi Chen & Charlos Guestrin

XGBoost is presented as a highly efficient gradient-boosting system optimized for both speed and accuracy. The paper introduces sparsity-aware algorithms for handling missing data and discusses system-level optimizations like cache utilization and parallel processing, making XGBoost scalable for large datasets.

XGBoost, or eXtreme Gradient Boosting, is a scalable and powerful tree boosting system widely used in machine learning.

1. Overview: XGBoost is an implementation of gradient tree boosting, a technique that iteratively builds decision trees to optimize a specified loss function. It combines predictions from multiple trees to improve accuracy, making it effective for both classification and regression tasks.
2. Key Features:
   - o Regularized Learning Objective: XGBoost introduces a regularization term to control model complexity, preventing overfitting and ensuring smoother weight adjustments across trees.
   - o Sparsity Awareness: It handles datasets with missing values or sparse inputs efficiently, leveraging default directions for handling such entries.
   - o Weighted Quantile Sketch: This method enables accurate split proposals for datasets with weighted instances, enhancing its ability to handle large-scale data.

This paper introduces XGBoost, a highly scalable gradient boosting framework for machine learning.

Key Innovations:

- Sparsity-aware Algorithm: Handles missing data efficiently.
- Weighted Quantile Sketch: Enables approximate tree learning for large datasets.
- System-Level Optimizations: Enhancements in cache usage, parallel computing, and distributed processing enable the handling of billions of data points.

Performance: XGBoost achieves state-of-the-art results across a wide range of machine learning challenges, including Kaggle competitions.

4. “Stock Price Prediction Using CNN and LSTM-Based Deep Learning Models” by Sidra Mehtab & Jaydip Sen

This study employs CNN and LSTM models for predicting stock prices of the NIFTY 50 index. It highlights that LSTM-based models excel at capturing temporal dependencies, while CNN is faster in execution. The encoder-decoder convolutional LSTM model was the most accurate, while a simple CNN model was the quickest.

The CNN is a type of deep learning algorithm designed to extract features from structured data like images or tensors.
1. Purpose: CNNs are effective in identifying spatial patterns and features in data. Originally designed for image recognition, they can be adapted for other domains, such as stock market prediction, by processing tensors derived from 1D or 2D data.
2. Architecture: A CNN is composed of multiple layers:

- Convolutional Layers: These apply filters to input data to capture patterns. Each filter focuses on a specific aspect, such as edges or textures in image data or trends in tensors for time-series data.
- Pooling Layers: These reduce dimensionality and help retain essential features while reducing computational complexity.
- Flatten Layer: Converts multidimensional data into a 1D vector for input into subsequent layers, such as dense or fully connected layers.

This study develops deep learning models for predicting stock prices using historical data from the NIFTY 50 index.
CNN-Based Models:

- Extract spatial features and capture short-term patterns.
- Faster to execute but less effective for long-term dependencies.

LSTM-Based Models:

- Handle temporal dependencies and non-linear relationships effectively.
- Achieve the highest prediction accuracy.

Hybrid Models: The encoder-decoder convolutional LSTM combines the strengths of CNN and LSTM, emerging as the most accurate model for weekly forecasting.
Key Findings: While CNN models are computationally efficient, LSTM models offer superior accuracy, especially in dynamic financial environments.

5. “Understanding Random Forests: From Theory to Practice” by Gilles Louppe

This paper provides an in-depth theoretical and practical understanding of random forests. It discusses feature importance metrics, ensemble learning's robustness, and out-of-bag error as a cross-validation technique. The study emphasizes random forests' ability to handle large, high-dimensional datasets effectively.

Random Forest is an ensemble learning algorithm that builds a "forest" of decision trees to solve classification or regression tasks. It aggregates the predictions of all trees to make a final decision, enhancing accuracy and reducing overfitting.

This paper provides a comprehensive overview of random forests, a widely used ensemble learning technique.

- Theory: Discusses the theoretical foundations of decision tree ensembles, including bagging (bootstrap aggregating) and feature randomization.
- Feature Importance: Explains techniques to measure feature contributions, making random forests interpretable.
- Applications: Highlights use cases in classification, regression, and feature selection for high-dimensional data.
- Performance: Demonstrates robustness to overfitting, making random forests suitable for datasets with noise and missing values.

6. "NSE Stock Market Prediction Using Deep Learning Models" by Hiransha M, GopalKrishna E.A, Vijay Krishna Menon, Soman K.P

This research compares deep learning architectures—MLP, RNN, LSTM, and CNN—for stock market prediction using NSE and NYSE data. CNN outperformed others in accuracy and generalization, showcasing its ability to learn from shared dynamics between distinct markets.

This study applies deep learning architectures to stock price prediction, comparing models trained on NSE (India) data.

- Models Used:
    - Multilayer Perceptron (MLP)
    - Recurrent Neural Networks (RNN)
    - Long Short-Term Memory (LSTM)
    - Convolutional Neural Networks (CNN)
- Findings: CNN models outperform others in both prediction accuracy and generalization ability. They also successfully predict prices in NYSE stocks, demonstrating shared dynamics between markets.

7. "Stock Market Prediction Using Artificial Intelligence: A Systematic Review of Reviews."

This review aggregates findings from multiple studies on AI techniques like neural networks, reinforcement learning, and ensemble methods in stock market prediction. It identifies trends, challenges, and future directions, emphasizing AI's potential to predict market trends with high accuracy through hybrid and ensemble approaches.

This paper synthesizes findings from multiple reviews on AI applications in stock market prediction.

- Techniques Reviewed:
    - Neural networks (e.g., CNN, LSTM)
    - Reinforcement learning
    - Hybrid and ensemble models
- Challenges Identified:
    - Overfitting in small datasets
    - Difficulty in integrating external factors like news and sentiment analysis
- Future Directions: Emphasizes combining AI with advanced data sources (e.g., sentiment analysis, macroeconomic indicators) to improve predictive performance.
- Key Takeaway: AI-driven models, particularly hybrid ones, show promise in improving prediction accuracy and mitigating market volatility.

CHAPTER 3
SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

The existing methods for time series analysis and forecasting primarily focus on creating models based on one of the following three categories:

- Statistical Models: These models typically rely on traditional statistical methods to capture the underlying patterns in time series data. Common statistical models include ARIMA (Autoregressive Integrated Moving Average), Exponential Smoothing State Space Models (ETS), and SARIMA (Seasonal ARIMA). These models are widely used for tasks like trend and seasonality detection, and for forecasting future values based on historical data.
- Machine Learning Models: Machine learning methods aim to learn patterns and relationships within time series data using algorithms that do not require explicitly defined equations like statistical models. Examples of machine learning models include Random Forests, Support Vector Machines (SVM), XGBoost, and Gradient Boosting Machines. These models can handle more complex data and relationships, often offering superior accuracy over traditional statistical methods, especially when dealing with large datasets or non-linear relationships.
- Deep Learning Models: Deep learning methods, particularly Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, are designed to work with sequences of data and capture temporal dependencies more effectively. These models are capable of learning more complex patterns, and they excel in cases with large amounts of time series data or when the data is non-stationary, meaning its statistical properties change over time.

3.2 LIMITATIONS OF THE EXISTING SYSTEM

While the existing systems utilizing statistical, machine learning, and deep learning models have been effective, they exhibit certain limitations when applied to specific use cases, such as stock price prediction for a single stock:

- Lack of Integration: In the current systems, different models are generally applied independently to time series data. However, these systems do not combine or evaluate multiple models within the context of a single stock. The absence of such integrated analysis could potentially lead to missed opportunities in selecting the best-suited model for prediction, especially in volatile markets like stock prices.
- Model Selection Issues: The existing systems may not provide a clear framework for comparing the effectiveness of different models on the same dataset, leading to a lack of understanding of which approach would give the most accurate results. Users may need to manually test several models, which can be time-consuming and may not always yield the best solution.

3.3 PROPOSED SYSTEM

To overcome the limitations mentioned above, the proposed system aims to provide a comprehensive solution where a single stock is analyzed using all state-of-the-art models across three domains: Statistical Models, Machine Learning Models, and Deep Learning Models. The system would compare the results of all these models using a common performance metric, namely RMSE (Root Mean Square Error), to allow for easy comparison of their prediction accuracy.

The core features of the proposed system include:

- Comprehensive Model Application: The system applies a range of models from statistical, machine learning and deep learning techniques to a single stocks historical data. This ensures that the most effective model can be identified for the given stock.
- Comparison via RMSE: By comparing the models using RMSE, the system can objectively evaluate which model performs best. A lower RMSE indicating better accuracy.
- User-Centric: Users can choose the model based on their requirements, such as the level of complexity they desire or the speed of the model.

3.4 ADVANTAGES OF THE PROPOSED SYSTEM

The proposed system offers several advantages over existing solutions:

- Model Comparison: By utilizing multiple models and comparing them using a standard evaluation metric (RMSE), users can more easily identify which model works best for their specific use case, providing a more informed decision-making process.
- Flexibility: Users have the flexibility to select the model that best fits their needs. For example, if the user requires a quick prediction with a simpler model, they might prefer a statistical or machine learning model. On the other hand, if the user is dealing with a complex, high-volume dataset, they might choose a deep learning model for better accuracy.
- Comprehensive Forecasting: The system integrates statistical, machine learning, and deep learning techniques, allowing for more robust forecasting. Each model type has its strengths and weaknesses, and the proposed system ensures that the user can explore all of them to achieve the most accurate prediction.
- Improved Prediction Accuracy: By analysing a stock using different modelling techniques, the proposed system enhances the chances of obtaining more accurate predictions, which is crucial in stock price forecasting, where volatility and sudden market shifts are common.

CHAPTER 4
SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

The system architecture is designed to process historical stock market data, train predictive models, and provide future stock price predictions. The architecture consists of the following components:

1. Data Ingestion:
   - Sources: Yahoo Finance API.
2. Feature Engineering:
   - Creating time-based features (e.g., day, month, year).
   - Incorporating technical indicators (e.g., moving averages, RSI).
3. Model Training:
   - Algorithms: AR, MA, ARIMA, Random Forest, XGBoost, RNN and LSTM for time series analysis.
   - Evaluation Metrics: RMSE.
4. Prediction and Visualization:
   - Predicting closing stock prices based on trained models.
   - Visualizing historical data and predictions using interactive charts.
5. Deployment:
   - Hosting on a local server.
   - User-friendly interface for stakeholders to view predictions and charts

4.2 ARCHITECTURE DIAGRAM

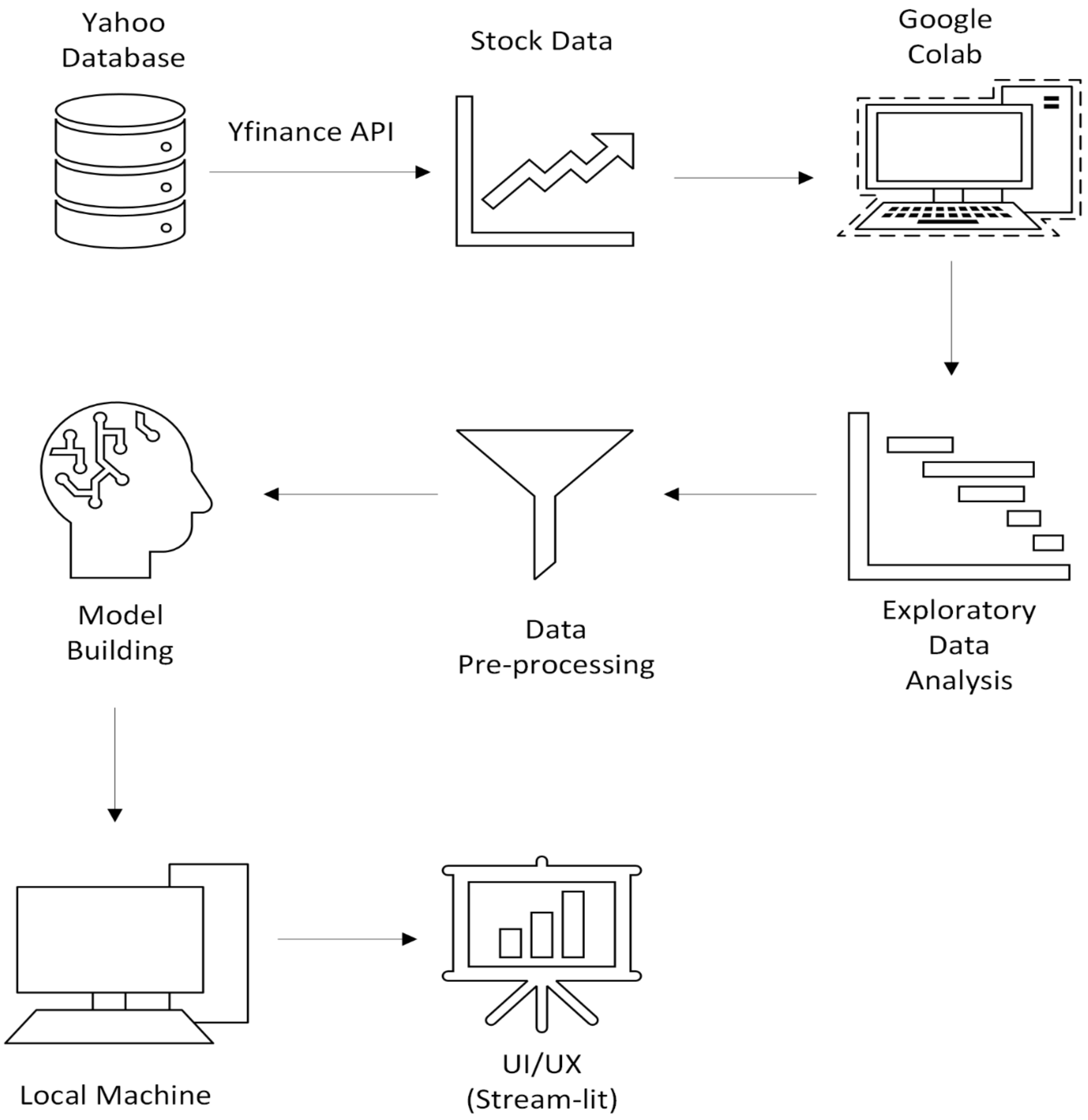

Fig 1: Architecture Diagram

Figure 1 depicts the Architecture of our System for stock price prediction. The process begins with fetching stock data from the Yahoo Database using the YFinance API. The data is processed and analyzed in Google Colab, involving steps such as exploratory data analysis and data preprocessing. The processed data is then used for model building, and the resulting model is deployed locally for user interaction through a Streamlit-based UI/UX.

CHAPTER 5
DATA COLLECTION AND PREPRATION

5.1 DATA COLLECTION

Table 1: Head of the Data

Date	*Adj Close*	*Close*	*High*	*Low*	*Open*	*Volume*
1996-01-01	3.8028	7.3454	7.3583	7.2707	7.3191	104121369
1996-01-02	3.7732	7.2884	7.3637	7.2354	7.3280	168743308
1996-01-03	3.8019	7.3441	7.7457	7.3280	7.4081	209323879
1996-01-04	3.7668	7.2762	7.2977	7.1780	7.2747	216900264
1996-01-05	3.7409	7.2262	7.2477	7.1638	7.2477	166708467

Table 2: Tail of the Data

Date	*Adj Close*	*Close*	*High*	*Low*	*Open*	*Volume*
2024-12-17	1245.3000	1245.3000	1263.9000	1242.8000	1261.0500	17462791
2024-12-18	1253.2500	1253.2500	1259.9499	1240.6500	1240.6500	12670179
2024-12-19	1230.4499	1230.4499	1244.9000	1229.0000	1239.0000	14244653
2024-12-20	1205.3000	1205.3000	1239.5000	1201.5000	1224.0000	20312896
2024-12-23	1222.3000	1222.3000	1227.1999	1213.1999	1215.0000	10052824

Table 1 and Table 2 represent the first 5 and last 5 rows of Reliance stock data, respectively, stored in the data frame. These tables provide a snapshot of the initial and final stock records in the dataset for quick reference.

5.2 DATA PLOT

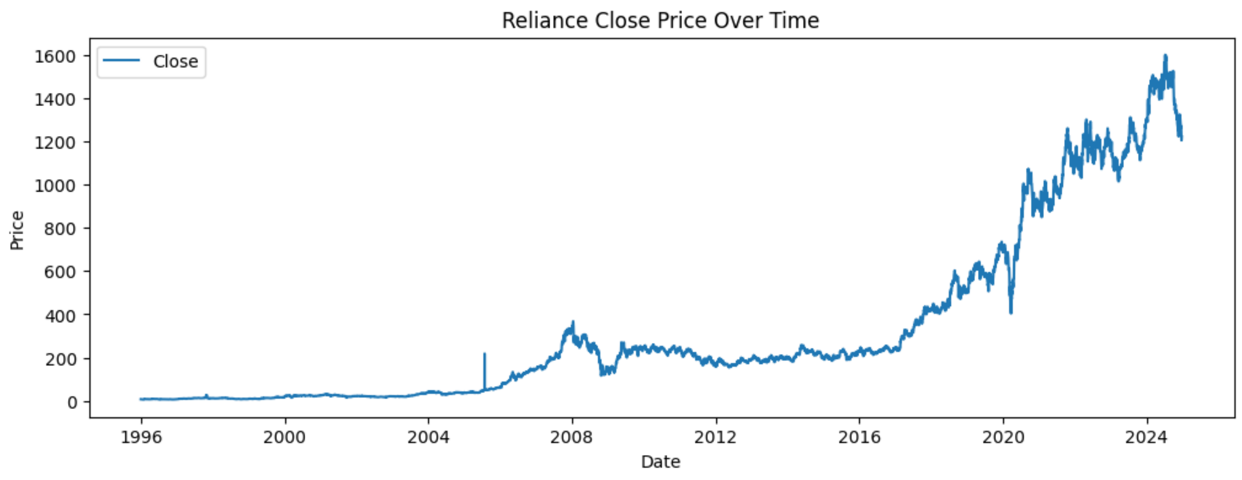

Fig 2: Reliance Close Price Over Time

Figure 2 presents the daily closing price of Reliance stock from 1996 to 2024.

5.3 DATA PREPRATION

Data Preprocessing: Handling Null Values

In data analysis, Null values (or missing values) are a common issue, as they can arise due to various reasons such as incomplete data collection, errors in data entry, or problems during data extraction. If these missing values are not handled properly, they can lead to inaccurate analysis or predictions. Therefore, addressing Null values is an essential step in the data preprocessing phase.

The process involves the following steps:

1. Identifying Null Values
   - First, the dataset is inspected for missing or Null values. This step is done to identify any gaps in the data, such as cells that are empty or contain Nan (Not a Number) values, which often represent missing data.
   - Common tools like Pandas in Python offer methods like isnull () or isna () to easily check for these Null values in each column of the dataset.

2. Replacing Null Values with the Mean of Nearby Data
   - Once the Null values are identified, they need to be replaced (also known as imputation) with reasonable estimates to avoid removing rows or columns with missing data, which could lead to loss of information.
   - One common technique for handling missing data is replacing Null values with the mean of nearby data points. This means filling in the missing value with the average

(mean) of the values in the surrounding data points. The term "nearby" usually refers to adjacent or neighboring values within the time series or dataset.

- For example, if a particular data point is missing in a time series, the missing value can be replaced by the average of the previous and next values. In some cases, the mean of a defined range of values (like the previous few data points) is used to ensure more contextual consistency.

4. Potential Limitations

- Not Always Accurate: This approach works well when the data is relatively stable and has small variations. However, in cases where the data fluctuates greatly or contains significant outliers, replacing missing values with the mean may introduce errors.
- Doesn't Capture Complex Relationships: This method does not account for complex relationships or patterns that might exist between different variables. For example, if the missing value is influenced by multiple other variables, a more sophisticated imputation method (like using machine learning models or interpolation) might be needed.

CHAPTER 6
EXPLORATORY DATA ANALYSIS

6.1 DATA EXPLORATION

In time series analysis, the goal is to break down the data into its core components: Trend, Seasonality, and Residuals. These components help to understand the underlying patterns in the data, which can be useful for forecasting and model building. By visualizing these components through decomposition, we can better interpret how each part contributes to the overall time series.

Exploration of Data for Trends, Seasonality, and Residuals:

In time series analysis, the goal is to decompose data into three components:

1. Trend: The long-term movement or direction in the data (e.g., upward or downward).
2. Seasonality: The regular, repeating patterns over fixed intervals (e.g., daily, weekly, yearly).
3. Residuals: The random fluctuations or noise remaining after removing trend and seasonality.

Decomposition breaks down the time series into these components for better analysis and forecasting. By plotting the decomposed components:

- Trend plot shows the overall direction of the data.
- Seasonality plot reveals recurring cycles or patterns.
- Residual plot helps identify randomness or unexplained variations.

Decomposing the data helps:

- Identify and forecast long-term trends.
- Account for periodic patterns (seasonality).
- Evaluate residuals to check for unexplained noise.

Methods:

- Classical Decomposition (Additive or Multiplicative)
- STL Decomposition (robust method using LOESS)

In essence, this process aids in better understanding the time series and improving forecasting accuracy.

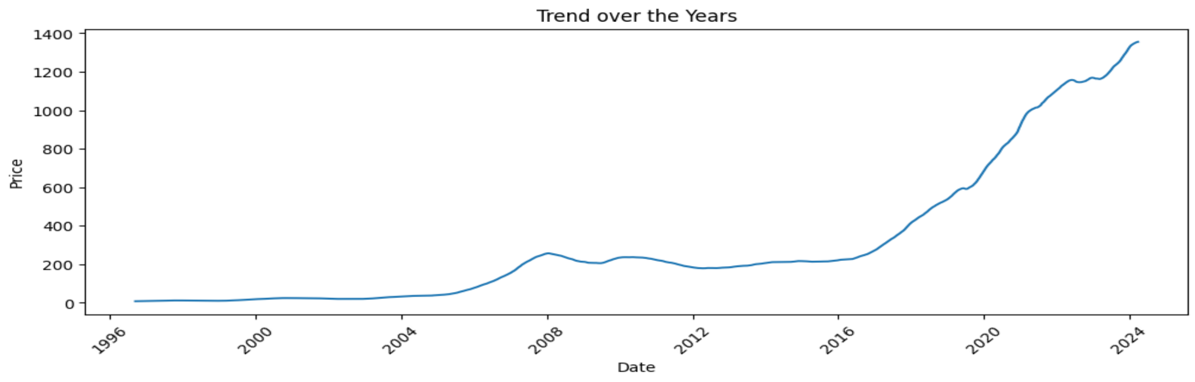

Fig 3: Trend Component over the years

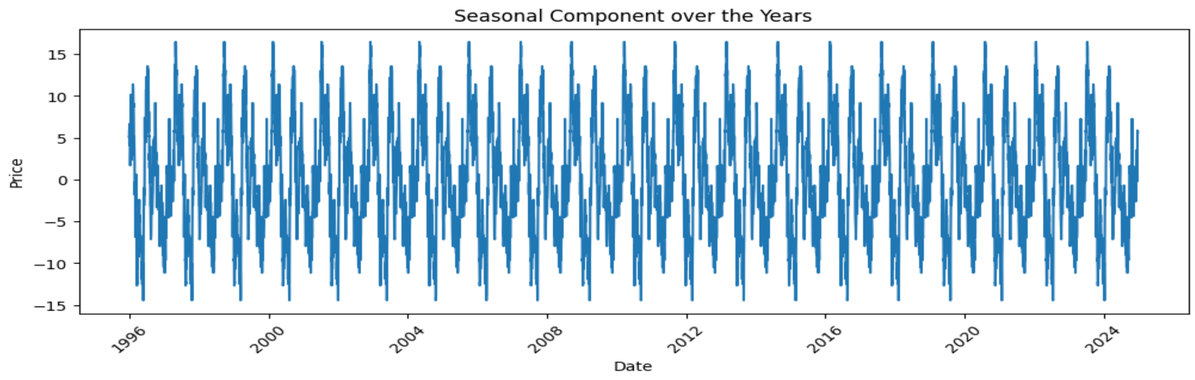

Fig 4: Seasonal Component over the years

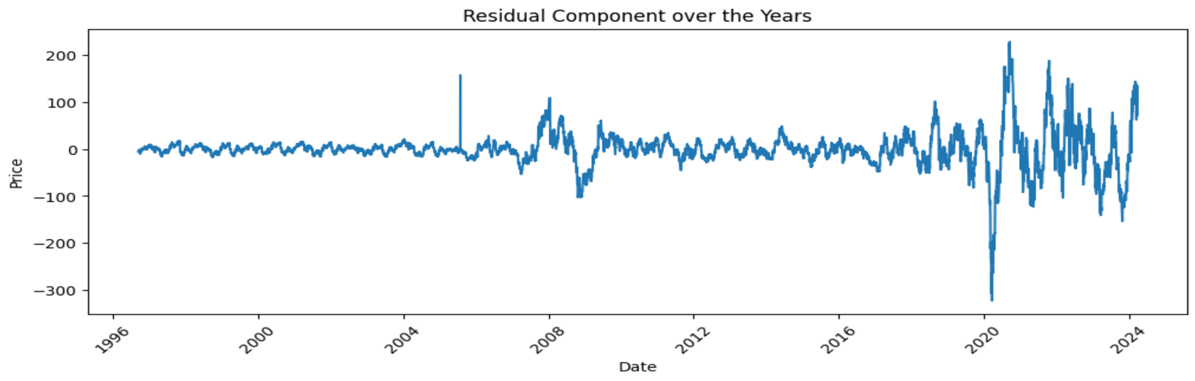

Fig 5: Residual Component over the years

Figure 3, Figure 4, and Fig 5 present the trend, seasonal, and residual components of the Close price after Classical decomposition respectively.

CHAPTER 7
METHODOLOGY

7.1 PROJECT WORKFLOW

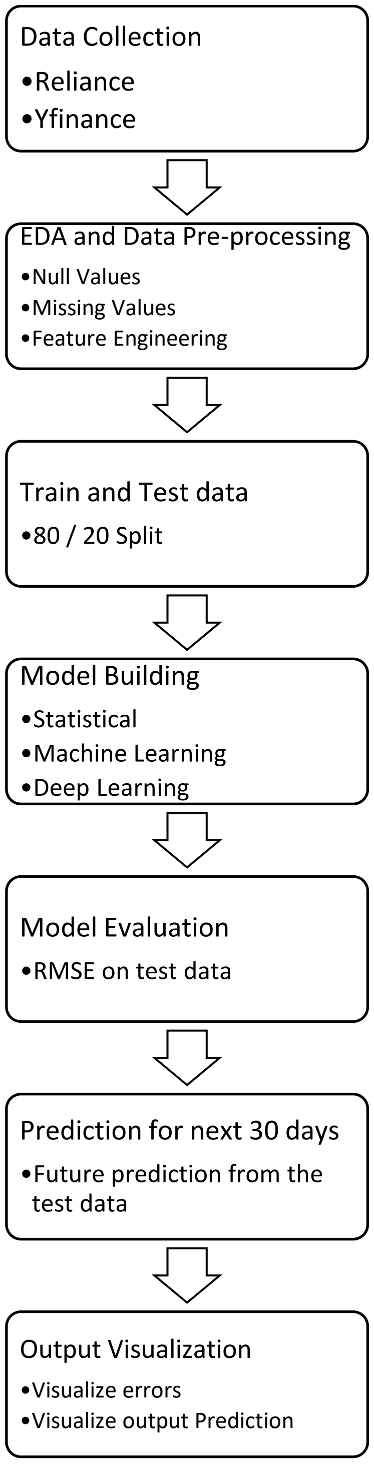

Fig 6: Project Workflow

Figure 6 depicts the workflow of the project.

7.2 DATA MODELS

AR MODEL

The Autoregressive (AR) model is a fundamental time series analysis method that predicts a data point based on its past values and a stochastic error term. It’s suitable for modeling short-term relationships within a stationary time series.

Key Features

- Stationarity Requirement: The series should have constant statistical properties over time.
- Short-Term Dependency: Captures relationships between recent data points.

Model Properties

The AR model assumes that the time series can be represented as a combination of exponential, sinusoidal, and polynomial trends, depending on the roots of its characteristic polynomial.

- Real Roots: Indicate exponential trends.
- Complex Roots: Represent oscillatory behavior (e.g., sinusoidal components).

Parameter Estimation

Parameters (ϕ\phi) are estimated using techniques like:

- Yule-Walker Equations: Based on autocorrelations.
- Ordinary Least Squares (OLS): Regression-based estimation.

MA MODEL

The Moving Average (MA) model is another fundamental time series analysis method that expresses the current value of a time series as a linear combination of past white noise terms (random shocks). It is particularly suitable for modelling short-term dependencies and smoothing noise in a stationary time series.

Key Features

- Stationarity Requirement: The series must have constant statistical properties over time.
- Dependency on Error Terms: Captures the influence of past random shocks on the current data point.

Model Properties

The MA model assumes that the time series behaviour arises from the interaction of past stochastic error terms, which can lead to patterns that mimic trends or periodicity, depending on the weights applied to past errors.

- Fixed Lag Structure: Only considers a finite number of past error terms.
- Uncorrelated Noise: Relies on the assumption that the white noise terms are uncorrelated and have a mean of zero.

ARIMA MODEL

ARIMA is a statistical approach for time series forecasting that models the relationship between an observation and its lagged values, including error terms from previous predictions. It is ideal for non-seasonal data and focuses on making non-stationary data stationary before applying the model.

Key Components:

- AR (Auto-Regressive) Order (p): Represents the number of lag observations included in the model.
- I (Integrated) Order (d): Specifies the number of differences required to make the series stationery.
- MA (Moving Average) Order (q): Indicates the number of lagged forecast errors in the model.

Methodology:

1. Stationarity Check:

Time series data is tested for stationarity using statistical methods like the Augmented Dickey-Fuller (ADF) test and KPSS test. Non-stationary series are transformed using techniques like differencing or logarithmic scaling.

2. ACF and PACF Plots:

These plots help determine the values of p and q by analyzing correlations at different lags.

3. Model Fitting:

The ARIMA model is built using statistical methods like maximum likelihood estimation to minimize errors. After fitting, the data is scaled back to its original form for interpretation.

Strengths:

- Suitable for datasets with trend components but no strong seasonality.
- Provides interpretable parameters for model tuning.

Limitations:

- Requires extensive preprocessing for non-stationary data.
- Performance degrades with highly seasonal or complex datasets.

SARIMA MODEL

SARIMA enhances ARIMA by incorporating seasonal patterns, making it suitable for data with periodic fluctuations. It introduces additional parameters to model seasonality.

Additional Parameters:

- P (Seasonal AR Order): Number of seasonal autoregressive terms.
- D (Seasonal Differencing Order): Number of seasonal differences required.
- Q (Seasonal MA Order): Number of seasonal moving average terms.
- s (Seasonality Period): Defines the periodicity of the data.

Methodology:

1. Order Selection:

Seasonal patterns are identified using seasonal ACF and PACF plots. P, D, Q, and s values are determined accordingly.

2. Model Fitting:

The SARIMA model is trained similarly to ARIMA but accommodates both trend and seasonality.

3. Residual Diagnostics:

The model’s validity is checked through residual analysis to ensure unbiased predictions.

Strengths:

- Effective for time series data with both trend and seasonality.
- Provides interpretable seasonal components.

Limitations:

- Computationally intensive due to additional parameters.
- Seasonal components must be explicitly defined.

VARIMA MODEL

The Vector Autoregressive Integrated Moving Average (VARIMA) model is an extension of ARIMA designed to handle multivariate time series data, where multiple interdependent variables are modeled together. It combines the features of ARIMA with vectorized (multivariate) modeling, making it suitable for systems where variables influence one another over time.

Key Features

- Multivariate Dependency: Models interrelationships between multiple time series.
- Stationarity Adjustment: Handles non-stationary data using differencing, similar to ARIMA.
- Lagged Relationships: Incorporates past values of all variables and their error terms.

XG-BOOST

XGBoost, or Extreme Gradient Boosting, is a scalable and powerful tree boosting system widely used in machine learning.

1. Overview: XGBoost is an implementation of gradient tree boosting, a technique that iteratively builds decision trees to optimize a specified loss function. It combines predictions from multiple trees to improve accuracy, making it effective for both classification and regression tasks.

2. Key Features:

- Regularized Learning Objective: XGBoost introduces a regularization term to control model complexity, preventing overfitting and ensuring smoother weight adjustments across trees.
- Sparsity Awareness: It handles datasets with missing values or sparse inputs efficiently, leveraging default directions for handling such entries.
- Weighted Quantile Sketch: This method enables accurate split proposals for datasets with weighted instances, enhancing its ability to handle large-scale data.

3. Optimization Techniques:

o Parallel and Distributed Computing: XGBoost supports multi-threading and distributed processing to speed up model training on large datasets.

o Cache-Aware Access: By optimizing memory access patterns, it reduces delays caused by non-contiguous memory reads.

o Out-of-Core Computation: It processes datasets that do not fit into memory by dividing data into manageable blocks and using techniques like compression and sharding.

4. Practical Impact: XGBoost has been instrumental in achieving state-of-the-art results in various machine learning challenges, such as Kaggle competitions and industry applications like ad click prediction and fraud detection.

5. Flexibility and Integration: It supports various objectives (classification, ranking, custom objectives) and integrates seamlessly with languages like Python, R, and Julia. It is highly regarded for its speed and efficiency compared to other tree-boosting systems.

RANDOM FOREST

The Random Forest is an ensemble machine learning model that uses multiple decision trees to improve prediction accuracy and robustness. It is versatile and suitable for both classification and regression tasks.

Key Features

- Ensemble Method: Combines multiple decision trees to reduce overfitting and variance.
- Bootstrap Aggregation (Bagging): Builds trees on random subsets of data with replacement.

Applications

- Classification: Predicting customer churn, spam detection.
- Regression: House price prediction, stock price forecasting.
- Anomaly Detection: Identifying outliers in datasets.

ANN ARCHITECTURE

An Artificial Neural Network (ANN) is a machine learning model inspired by the structure and functioning of the human brain. It learns complex relationships in data through multiple interconnected layers of neurons.

Key Features

- Non-linearity: Captures non-linear patterns in data.
- Layered Structure: Includes input, hidden, and output layers.
- Learning Mechanism: Adjusts weights using backpropagation and gradient descent.

Applications

- Image and Speech Recognition: Computer vision, voice assistants.
- Predictive Modelling: Time series prediction, medical diagnosis.
- Classification and Regression Tasks.

CNN ARCHITECTURE

The CNN is a type of deep learning algorithm designed to extract features from structured data like images or tensors.

CNN Basics:

1. Purpose: CNNs are effective in identifying spatial patterns and features in data. Originally designed for image recognition, they can be adapted for other domains, such as stock market prediction, by processing tensors derived from 1D or 2D data.

2. Architecture:

o A CNN is composed of multiple layers:

▪ Convolutional Layers: These apply filters to input data to capture patterns. Each filter focuses on a specific aspect, such as edges or textures in image data or trends in tensors for time-series data.

▪ Pooling Layers: These reduce dimensionality and help retain essential features while reducing computational complexity.

▪ Flatten Layer: Converts multidimensional data into a 1D vector for input into subsequent layers, such as dense or fully connected layers.

3. Key Components:

o Kernel Size: Determines the size of the filter used in convolution. For example, a kernel size of 3 means the filter examines a 3x3 grid of data at a time.

o Activation Functions: Functions like ReLU (Rectified Linear Unit) introduce non-linearity, allowing the network to model complex relationships.

o Feature Extraction: By stacking multiple convolutional and pooling layers, the CNN captures hierarchical features from data.

RNN ARCHITECTURE

A Recurrent Neural Network (RNN) is a specialized type of ANN designed to handle sequential data. It retains information from previous steps in the sequence through feedback connections.

Key Features

- Sequential Data Processing: Suitable for time series, natural language processing, and other sequence-based tasks.
- Memory Mechanism: Maintains a hidden state to capture temporal dependencies.
- Weight Sharing: Uses the same weights across different time steps.

Applications

- Time Series Analysis: Stock price prediction, weather forecasting.
- Natural Language Processing (NLP): Text generation, sentiment analysis, machine translation.

LSTM ARCHITECTURE

LSTM is a deep learning approach designed to capture long-term dependencies in sequential data. It excels in time series forecasting due to its ability to model complex patterns, trends, and seasonality dynamically.

Architecture:

- LSTM Layer: Stores and updates memory over time steps. It uses gates (input, forget, and output) to manage the flow of information.
- Dense Layer: Outputs the final prediction.

Methodology:

1. Data Preprocessing:

   o Normalize features using MinMaxScaler to scale data into the range [0, 1].

   o Reshape the dataset into sequences of input-output pairs (e.g., using a rolling window approach).

2. Model Definition:

   o Define the LSTM layer with neurons and activation functions (e.g., sigmoid or ReLU).

   o Use mean squared error as the loss function and Adam optimizer for training.

3. Training and Validation:

   o Train the model on a portion of the dataset and evaluate it using validation data.

   o the model learns sequential dependencies and adjusts weights dynamically.

Strengths:

- o Handles both linear and non-linear dependencies.
- o Adapts to varying complexities in the data, making it robust for real-world applications.

Limitations:

- o High computational cost and training time.
- o Requires careful tuning of hyperparameters like batch size, epochs, and learning rate.

7.3 RESULTS

In this section, we will examine the performance of the models on the test data.

AR Model

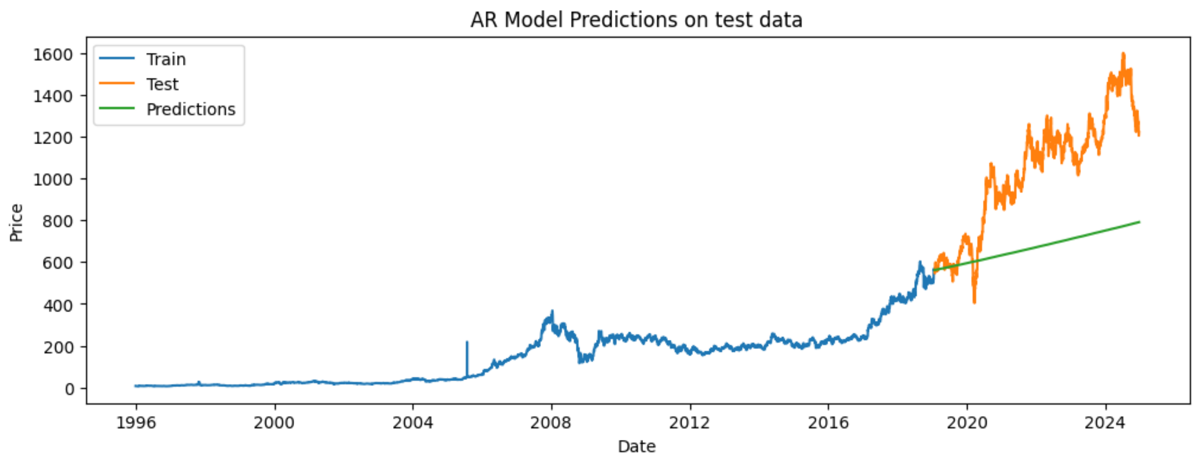

Fig 7: AR Model Predictions on the test data

MA Model

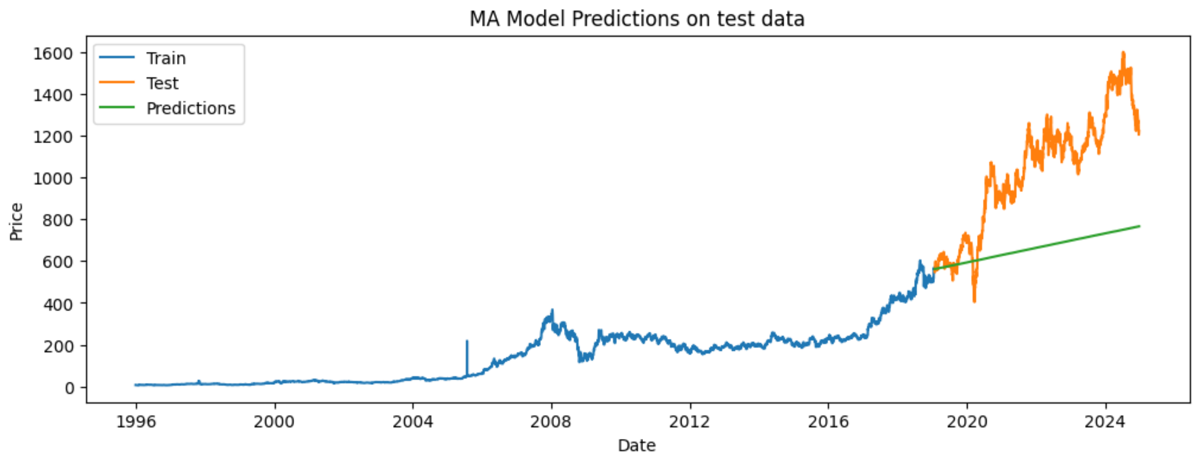

Fig 8: MA Model Predictions on the test data

Figures 7 and 8 present the performance of the AR and MA models on the test data respectively.

ARIMA Model

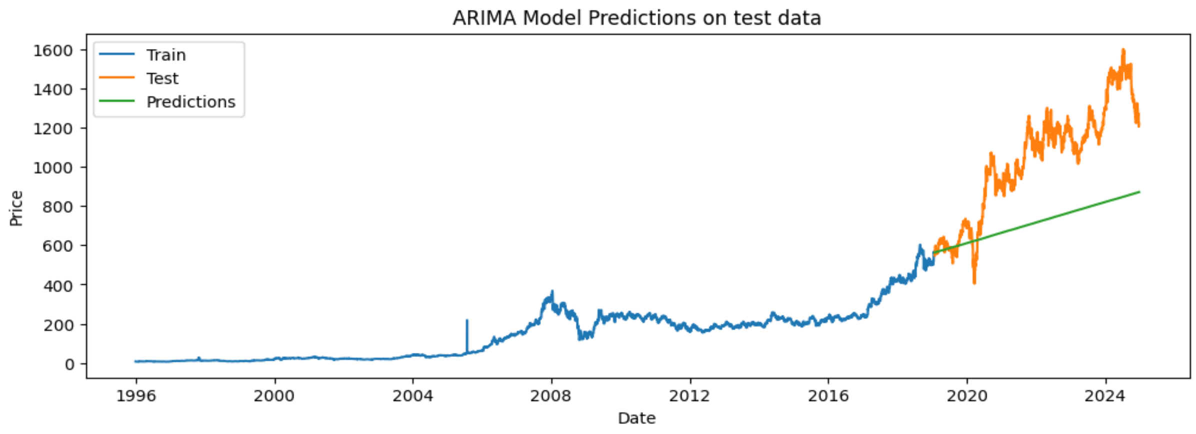

Fig 9: ARIMA Model Predictions on the test data

SARIMA Model

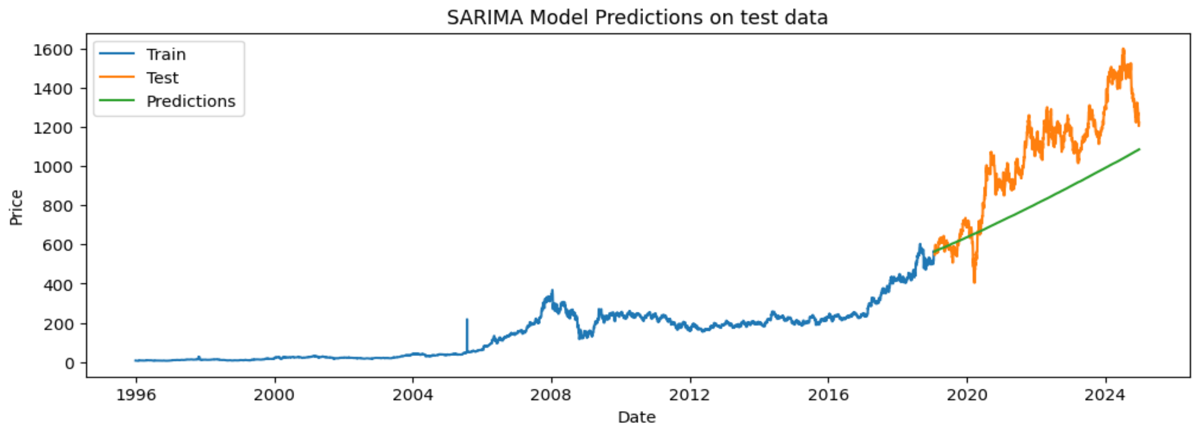

Fig 10: SARIMA Model Predictions on the test data

VARIMA Model

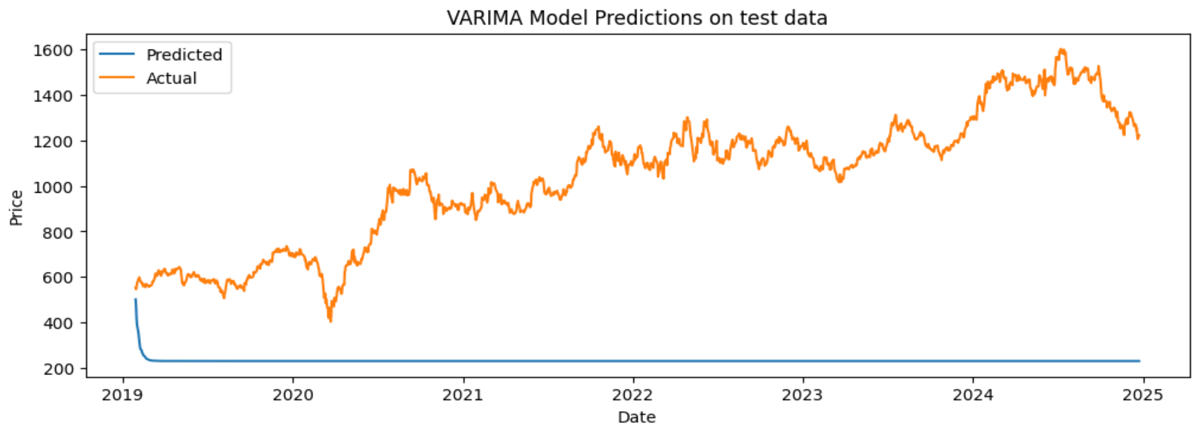

Fig 11: VARIMA Model Predictions on the test data

Figures 9,10 and 11 present the performance of the ARIMA, SARIMA and VARIMA models on the test data respectively.

XGBoost Model

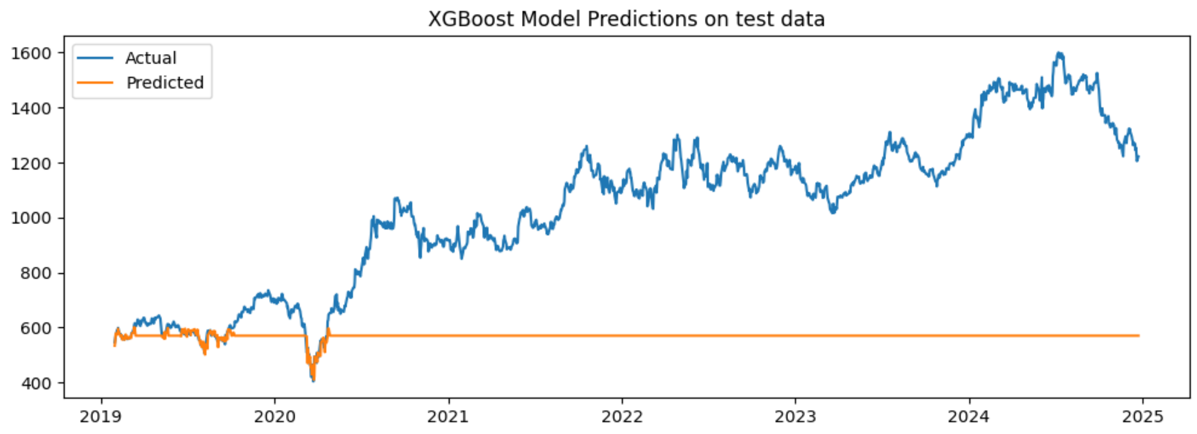

Fig 12: XGBoost Model Predictions on the test data

Random Forest

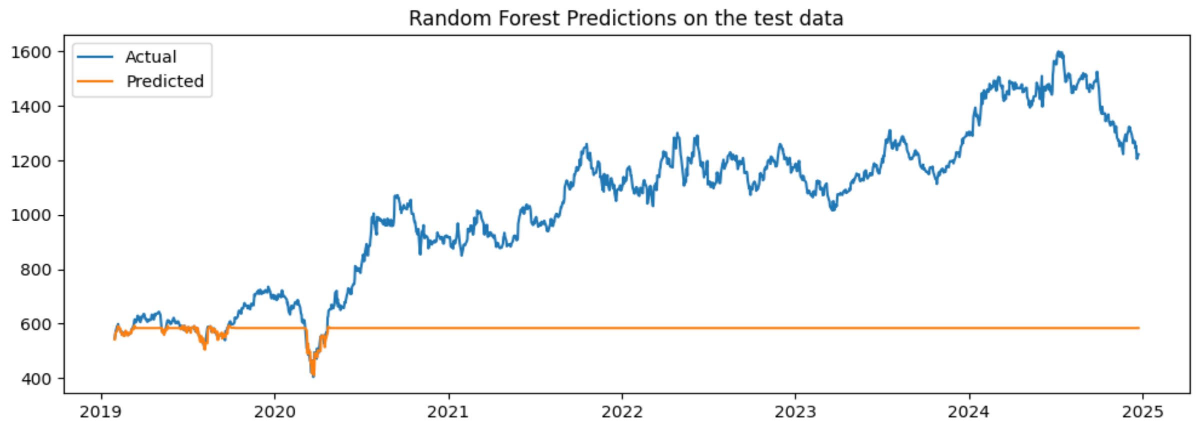

Fig 13: Random Forest Model Predictions on the test data

ANN Model

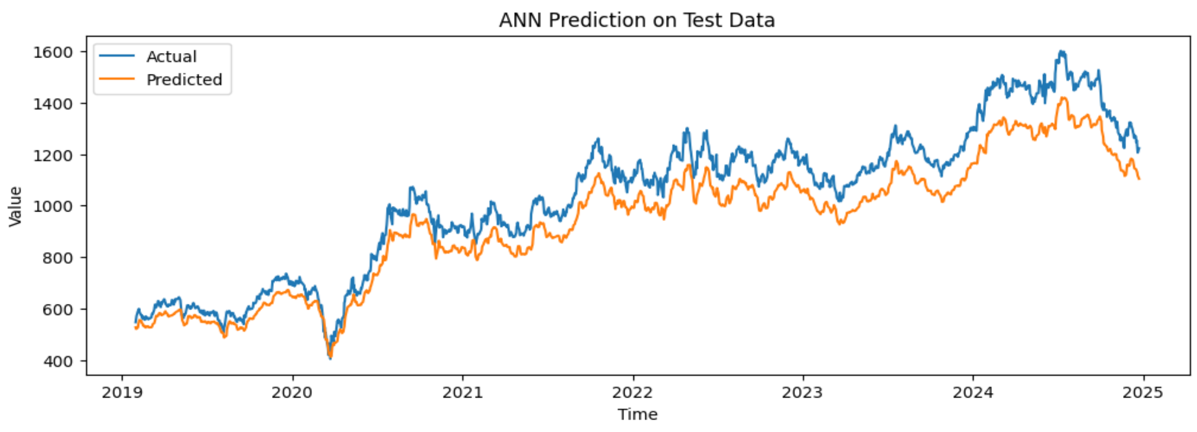

Fig 14: ANN Model Predictions on the test data

Figures 12,13 and 14 presents the performance of the XGBoost, Random Forest and ANN models on the test data respectively.

CNN Model

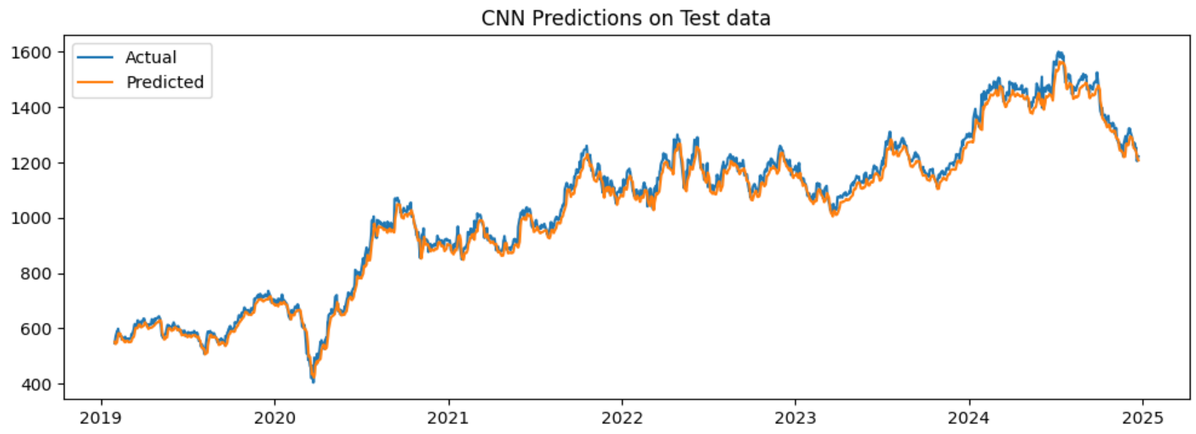

Fig 15: CNN Model Predictions on the test data

RNN Model

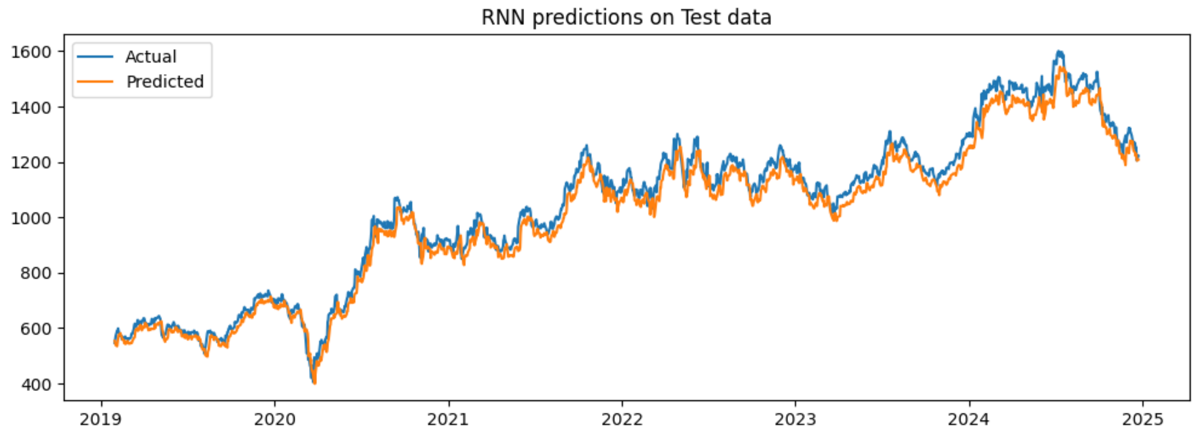

Fig 16: RNN Model Predictions on the test data

LSTM Model

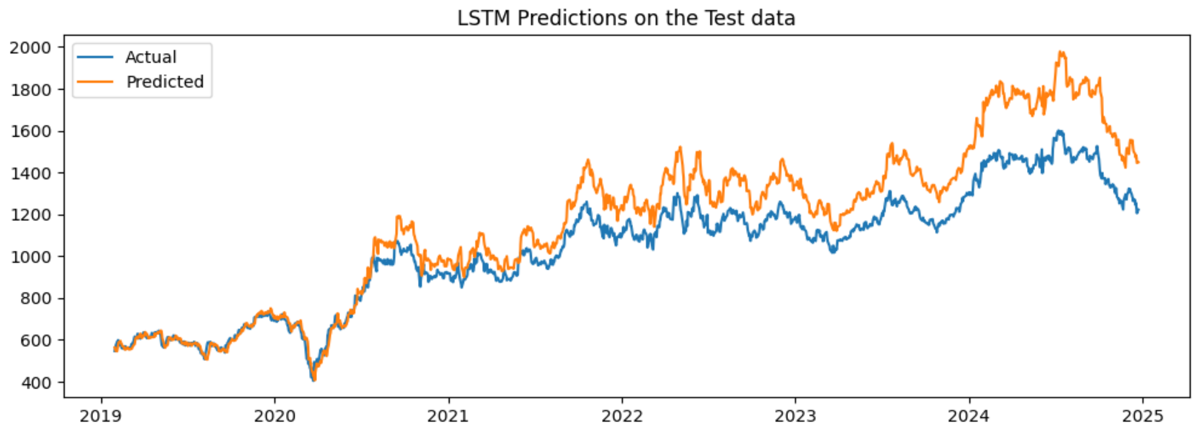

Fig 17: LSTM Model Predictions on the test data

Figure 15,16,17 presents the performance of the CNN, RNN and LSTM models on the test data respectively.

RMSE

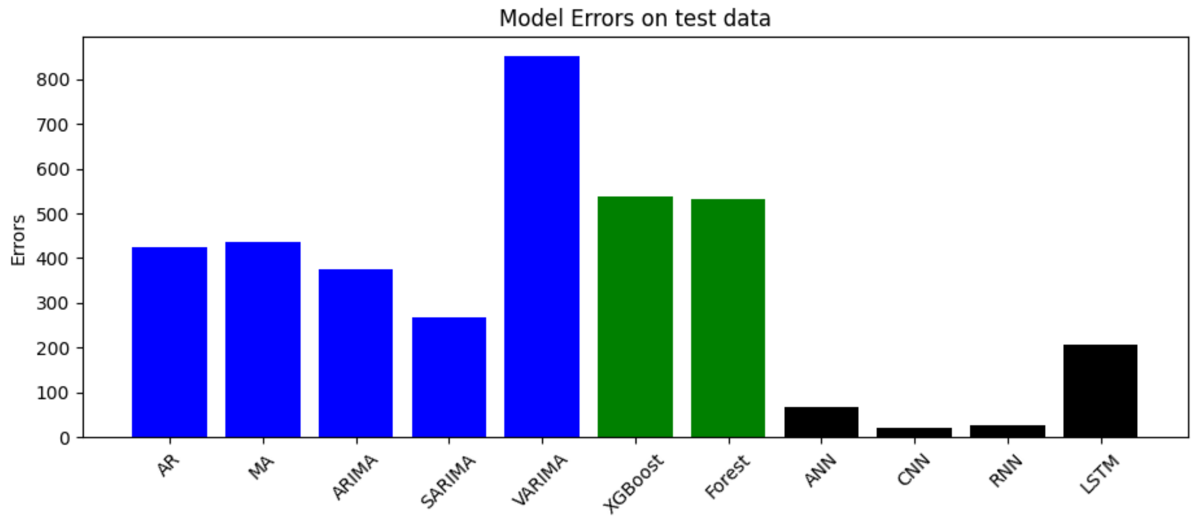

Fig 18: Model Errors the test data

Figure 18 represents the root mean square error of all the models for comparison.

STREAMLIT INTERFACE

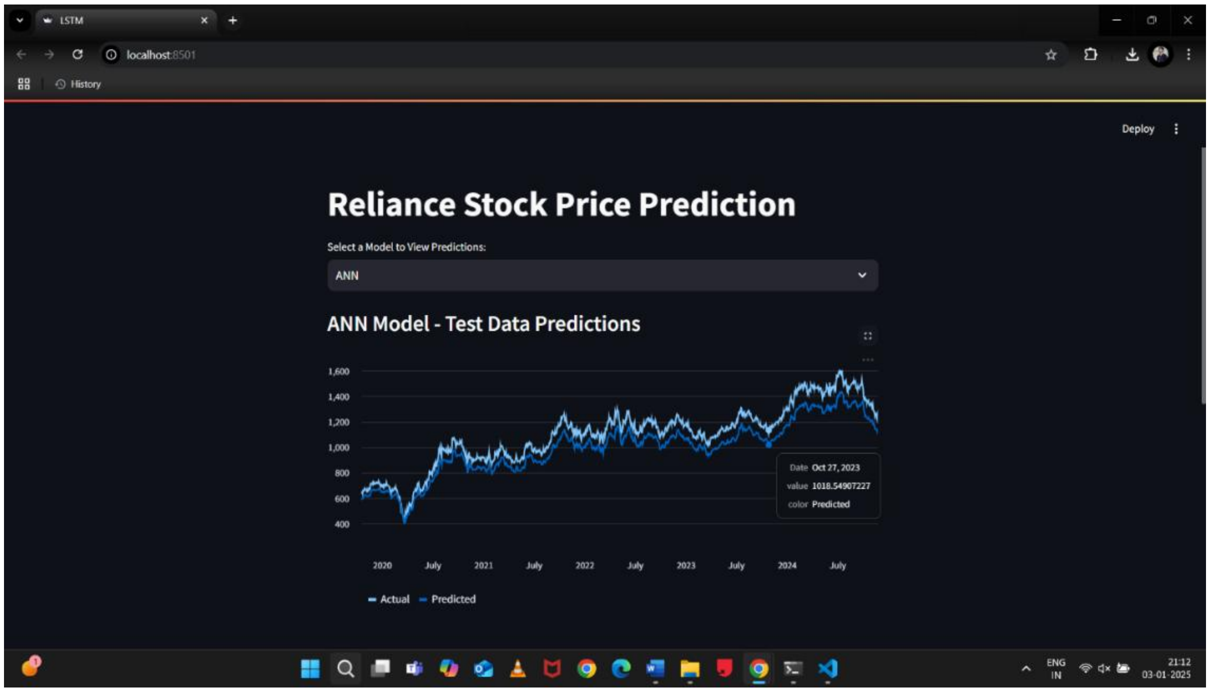

Fig 19: WebApp Displaying ANN Model

Figure 19 presents the Web interface developed using Stream-lit, which depicts the predictions of the ANN Model

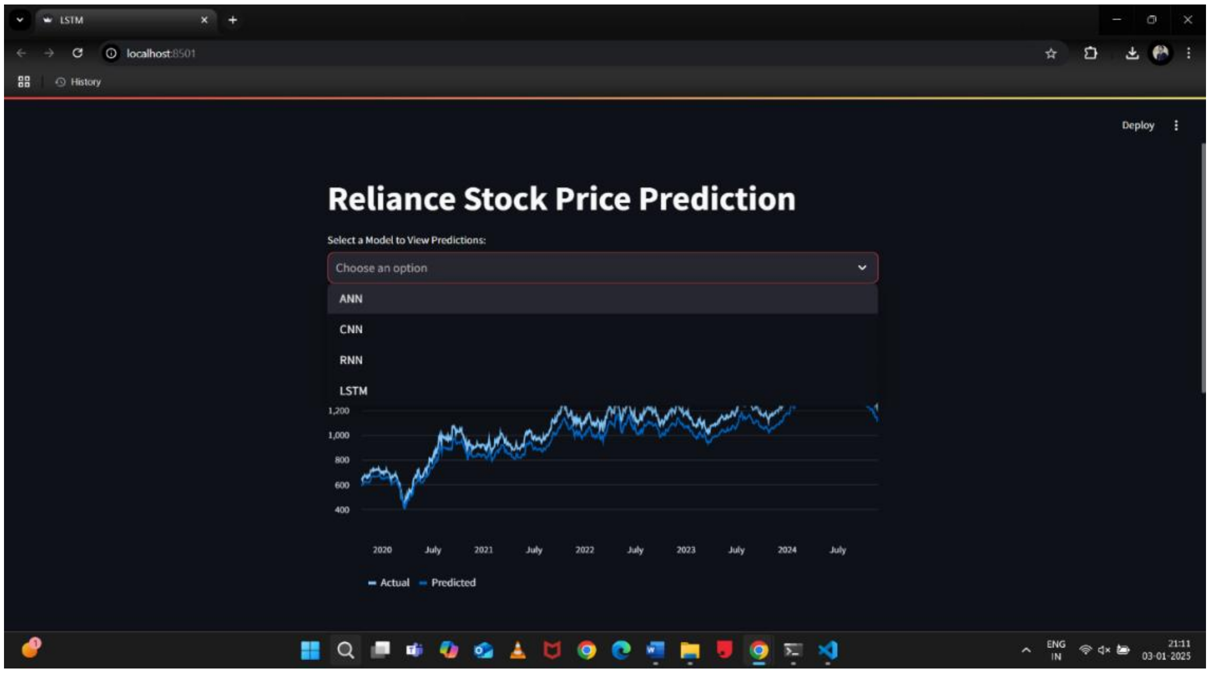
Reliance Stock Price Prediction
Select a Model to View Predictions:
Choose an option
ANN
CNN
RNN
LSTM
Actual
Predicted
Deploy

Fig 20: WebApp Displaying Select option.

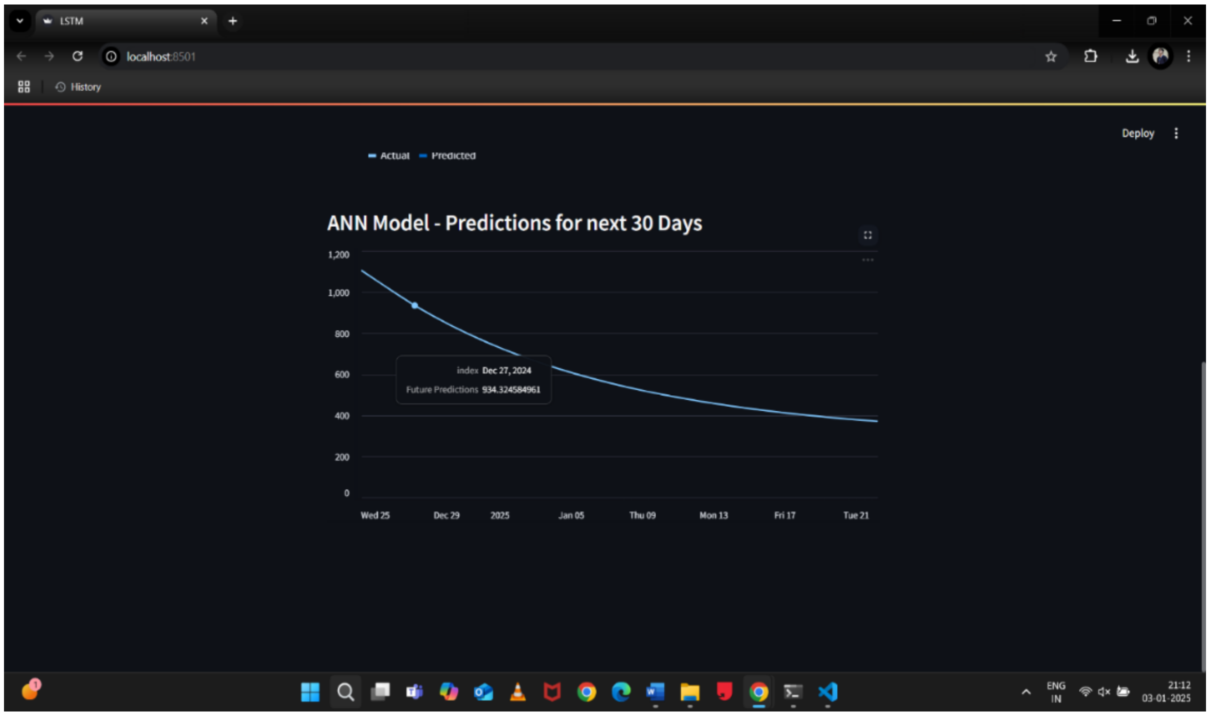
ANN Model - Predictions for next 30 Days
index Dec 27, 2024
Future Predictions 934.324584961
Actual
Predicted
Deploy

Fig 21: WebApp Displaying ANN model Prediction for next 30 days.

Figure 20 presents the web interface showing the options to select the Model among ANN, CNN, RNN and LSTM.
Figure 21 presents the ANN model's future predictions for the next 30 days.

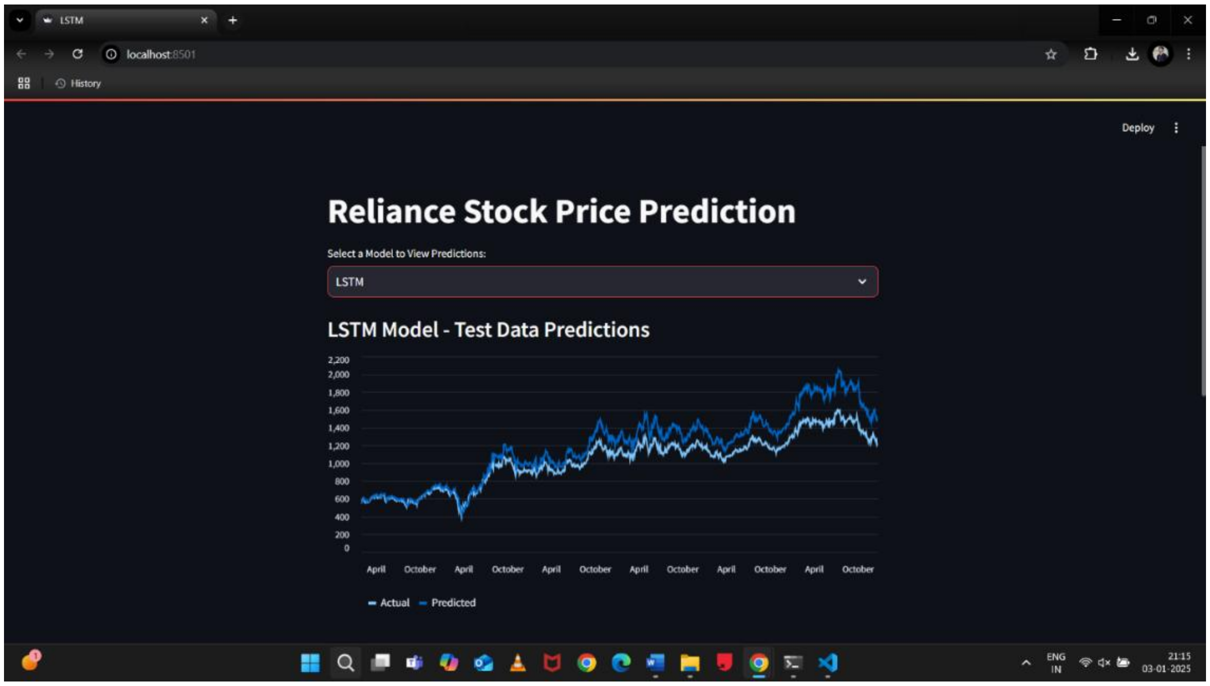

Fig 22: WebApp Displaying LSTM model test data prediction.

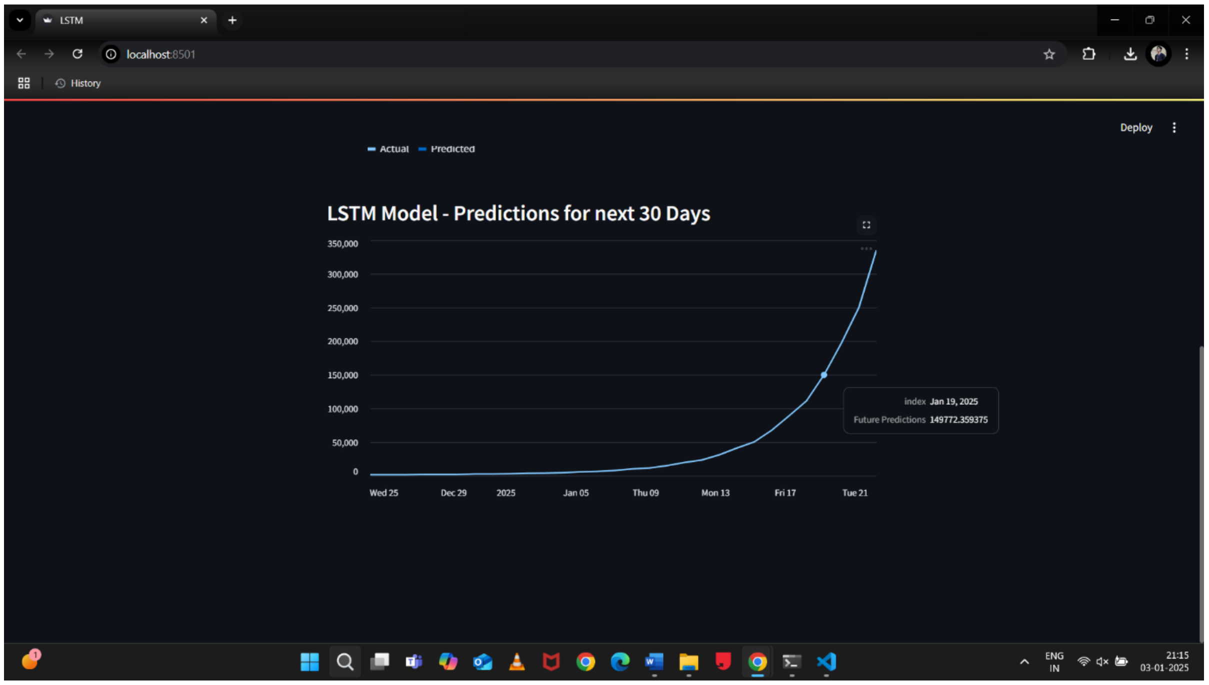

Fig 23: WebApp Displaying LSTM model prediction for next 30 days.

Figure 22 presents LSTM model's predictions on the test data
Figure 23 presents the LSTM model's future predictions for the next 30 days.

CHAPTER 8
TESTING

AR

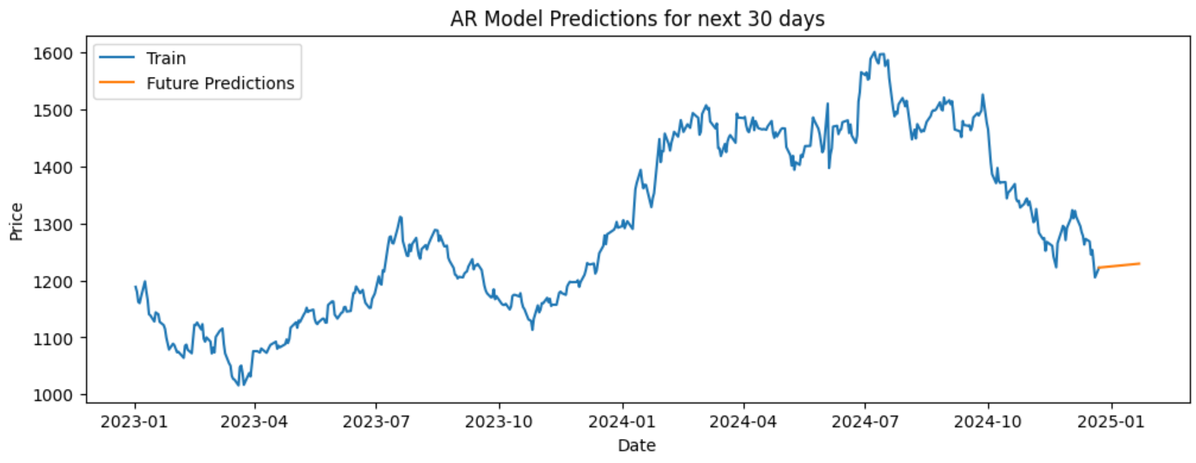

Fig 24: AR model prediction for next 30 days

MA

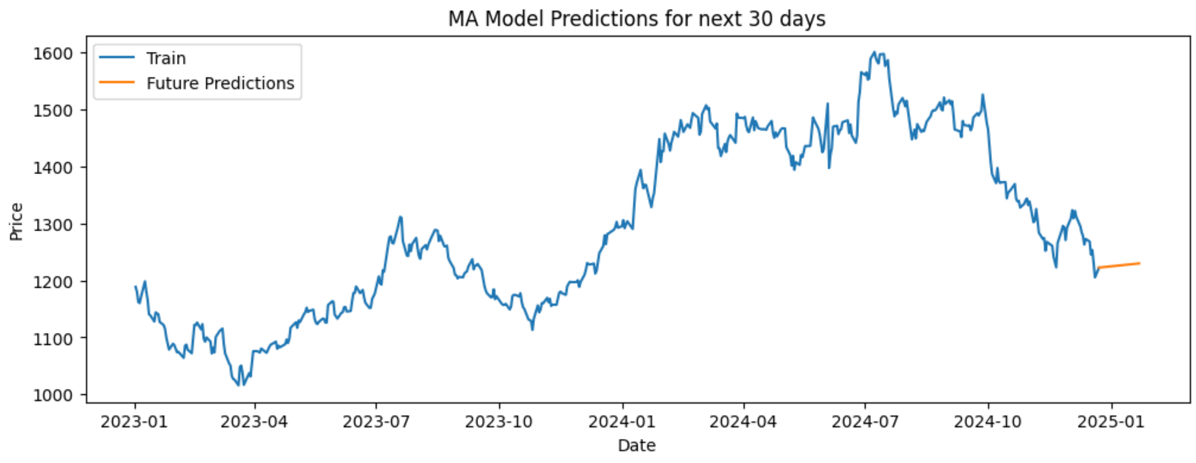

Fig 25: MA model prediction for next 30 days

Figures 24 and 25 presents the predictions by the AR and MA model for the future 30 days.

ARIMA

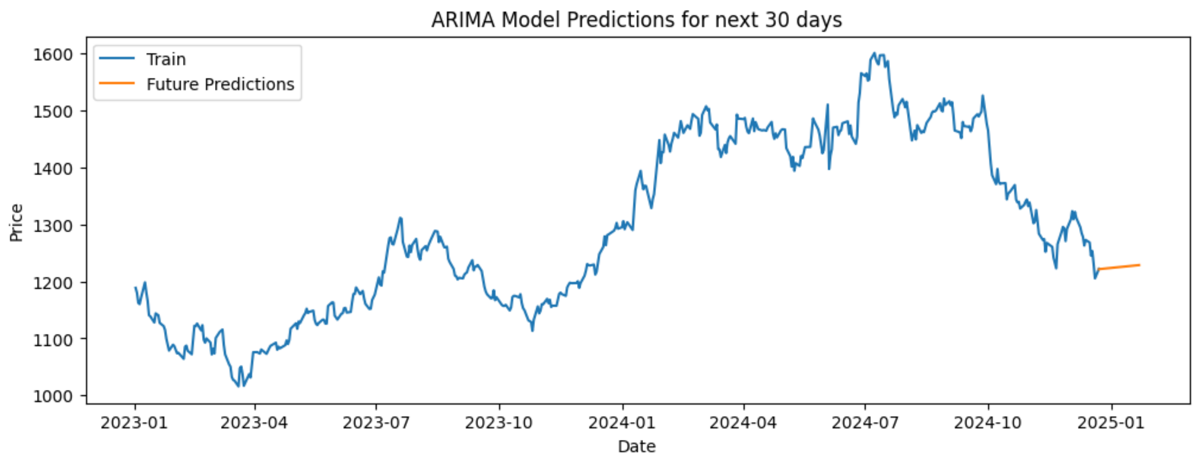

Fig 26: ARIMA model prediction for next 30 days

SARIMA

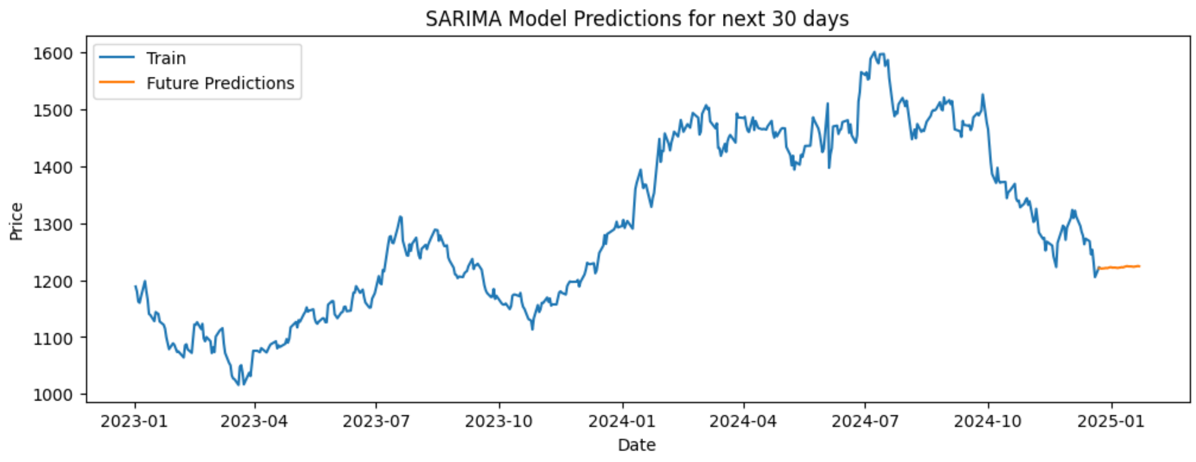

Fig 27: SARIMA model prediction for next 30 days

Figures 26 and 27 present the predictions by the ARIMA and SARIMA model for the future 30 days.

VARIMA

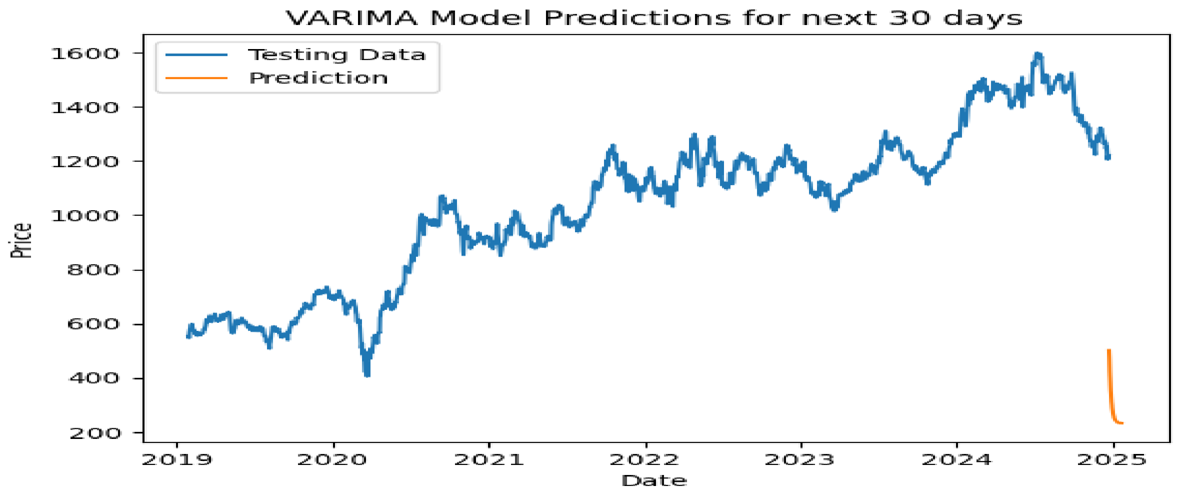

Fig 28: VARIMA model prediction for next 30 days

XGBOOST

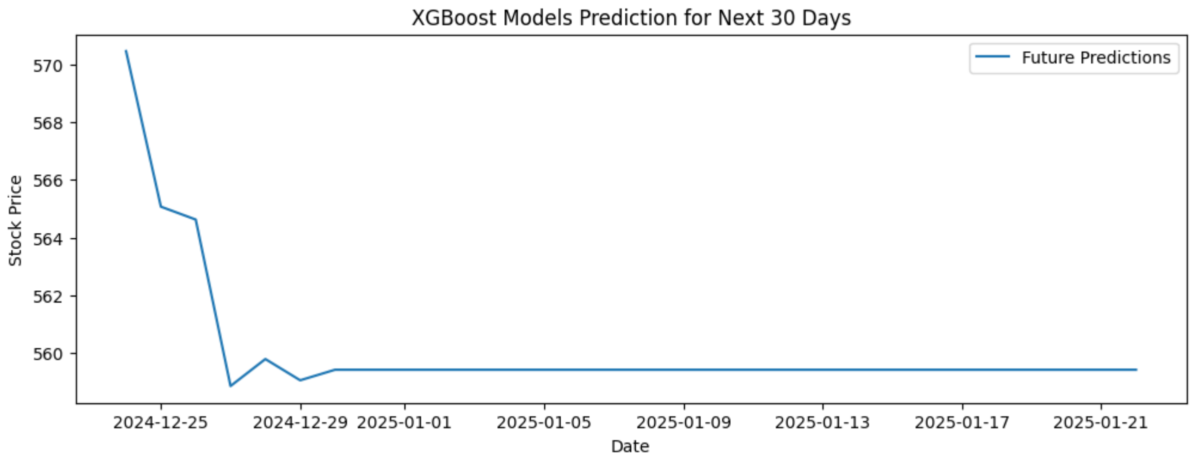

Fig 29: XGBoost model prediction for next 30 days

Figures 28 and 29 present the predictions by the VARIMA and XGBoost model for the future 30 days.

RANDOM FOREST

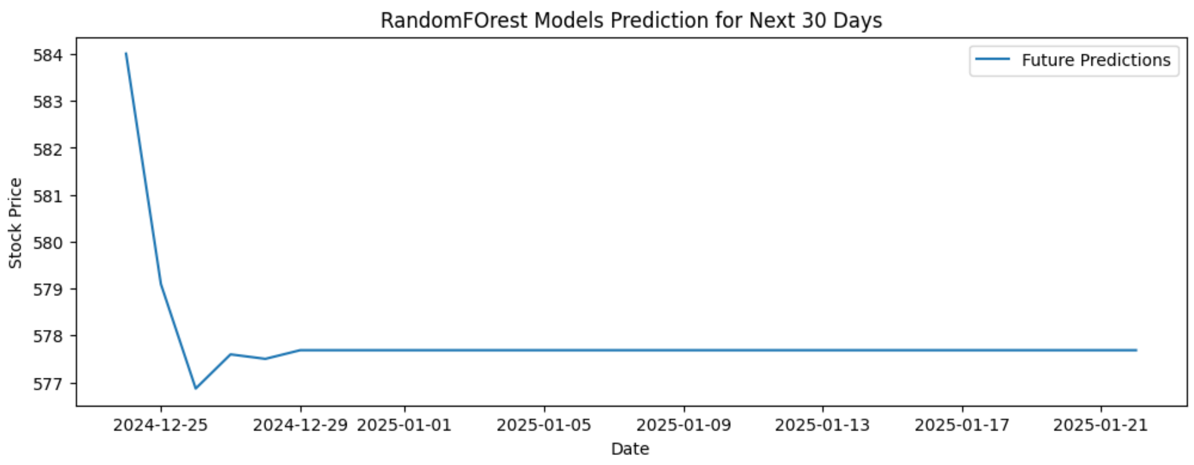

Fig 30: Random Forest model prediction for next 30 days

ANN

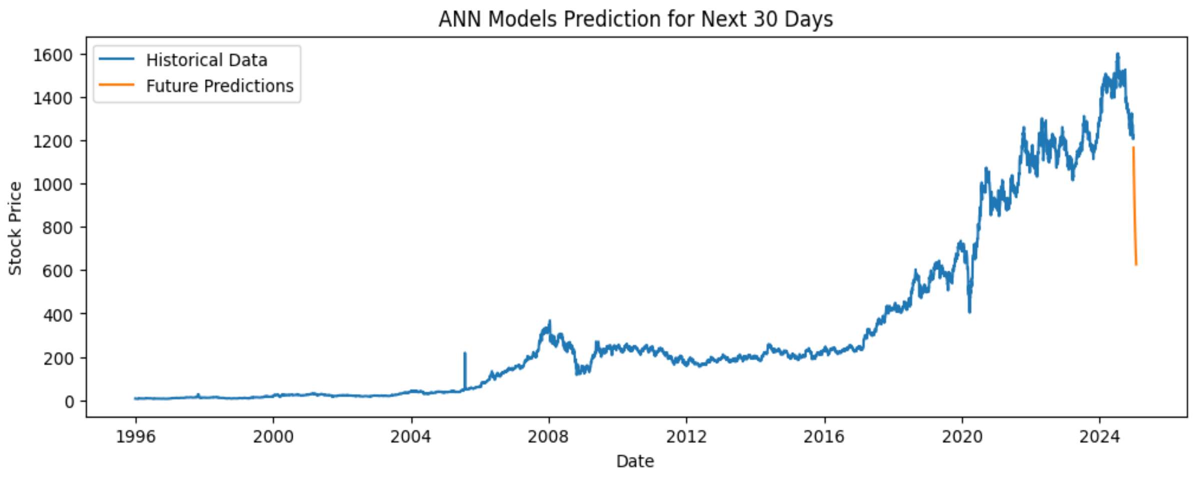

Fig 31: ANN model prediction for next 30 days

Figures 30 and 31 present the predictions by the Random Forest and ANN model for the future 30 days.

CNN

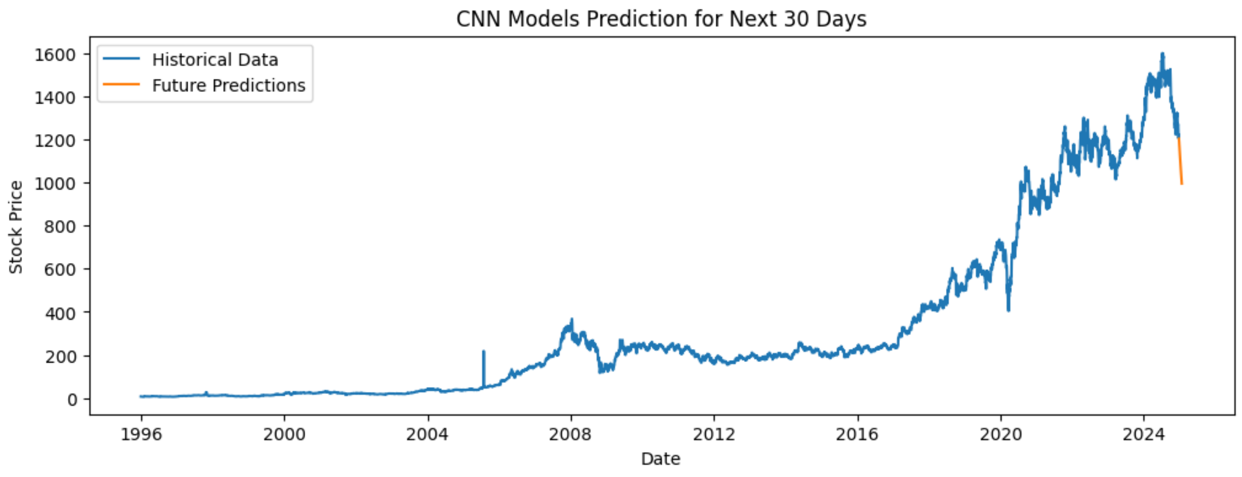

Fig 32: CNN model prediction for next 30 days

RNN

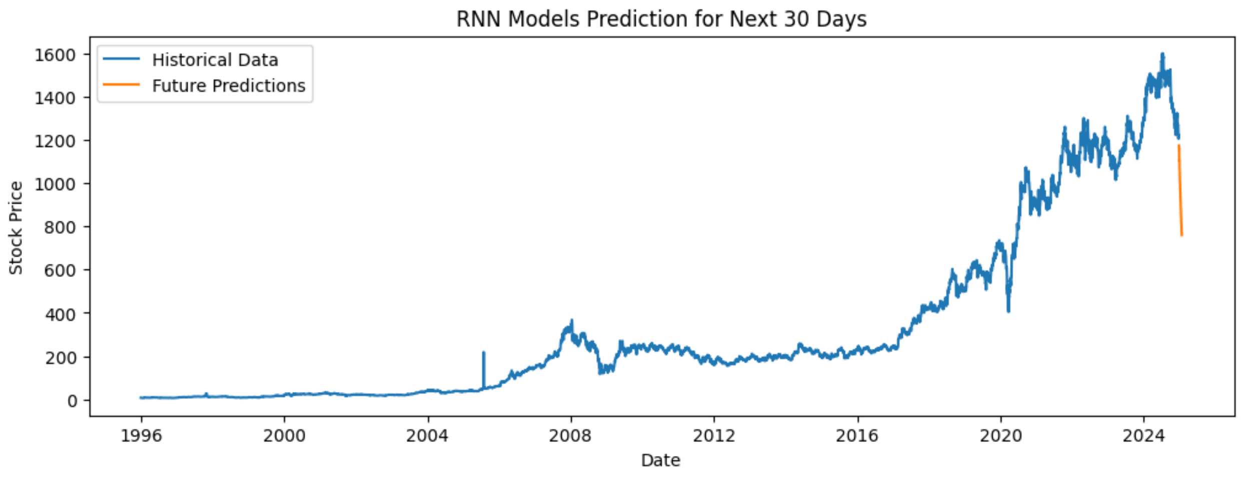

Fig 33: RNN model prediction for next 30 days

Figures 32 and 33 presents the predictions by the CNN and RNN model for the future 30 days.

LSTM

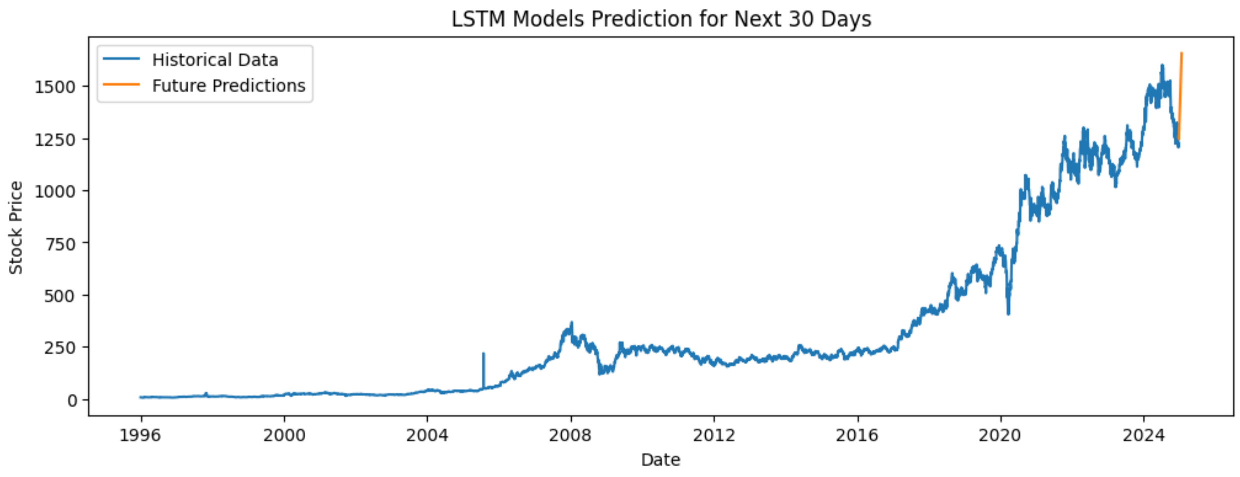

Fig 34: LSTM model prediction for next 30 days

Figures 34 presents the predictions by the LSTM model for the future 30 days.

CHAPTER 9
CONCLUSION

This project offers a comprehensive and robust approach to stock price prediction, drawing on a diverse range of modeling techniques from statistical, machine learning, and deep learning paradigms. By leveraging these three methodologies, we were able to assess and compare the strengths and weaknesses of different models in the context of time series forecasting. The evaluation of 11 models—5 statistical models (AR, MA, ARIMA, SARIMA, VARIMA), 2 machine learning models (XGBoost, Random Forest), and 4 deep learning models (ANN, CNN, RNN, LSTM)—enabled us to comprehensively explore various forecasting techniques and their applicability in predicting stock prices.

Throughout the course of the project, a variety of factors were considered to assess model performance, such as handling seasonality, trend extraction, data complexity, and computational efficiency. The **SARIMA** model was found to be particularly effective in capturing seasonality and trends, making it a strong candidate for time series data with clear periodic patterns. **Random Forest**, a robust machine learning technique, excelled in handling high-dimensional feature sets and interactions, providing reliable predictions in the presence of noisy data. Meanwhile, the **CNN**, with its ability to capture spatial patterns and local dependencies, showed great promise in recognizing subtle but important features within the historical stock price data, making it particularly useful in scenarios with non-linear relationships.

The integration of these models into a single predictive framework provides a well-rounded solution that can accommodate different types of data and forecasting requirements. The insights gained from this analysis have significant implications for stock market forecasting, where stakeholders can benefit from tailored recommendations on the most effective models depending on the specific characteristics of the stock data being analyzed. Whether looking for simple trend-based predictions or complex non-linear relationships, the project highlights the flexibility and power of combining traditional statistical techniques with more advanced machine learning and deep learning models.

A key strength of this project is the **Streamlit-based frontend** that was developed, enabling stakeholders to easily visualize the predicted stock prices for the next 30 days. This interactive platform empowers users to make informed, data-driven decisions, as they can easily interpret predictions and trends, track future market movements, and assess potential risks. The visualizations provided offer valuable insights that enhance decision-making processes and add practical value to the overall system.

FUTURE ENHANCEMENTS

Real-Time Analysis and Forecasting Using Big Data Technologies

The current system can be upgraded to perform real-time analysis and forecasting by integrating big data technologies. By harnessing tools such as Apache Kafka, Spark Streaming, and Apache Flink, the system can continuously ingest and process live stock market data. This would enable the prediction models to update in real-time, providing stakeholders with the most up-to-date insights. Additionally, leveraging big data platforms like Hadoop or Apache Hive can help manage and process vast amounts of financial data, enabling more accurate and timely predictions. The integration of real-time data feeds from financial markets could also allow for dynamic adaptation of models to account for rapidly changing market conditions, further improving the predictive power of the system.

Hybrid Models for Enhanced Accuracy

Another avenue for improvement lies in combining different modeling techniques to form hybrid models, which can leverage the strengths of various approaches. For instance, combining the time series forecasting capabilities of SARIMA with the predictive power of Random Forest or XGBoost could produce a more robust and accurate model. Hybrid models may take the form of ensemble learning, where the outputs of several models are combined (e.g., through stacking or boosting), or neural-ensemble models, where deep learning models such as CNN or LSTM are fused with traditional statistical or machine learning methods. This approach can reduce individual model biases and improve the overall prediction accuracy by incorporating complementary strengths.

Improved Feature Engineering

The predictive power of the models can also be increased through more advanced feature engineering. By incorporating complex features such as volatility indices, moving averages, momentum indicators, and market liquidity measures, the models can account for deeper market dynamics. Additionally, dimensionality reduction techniques like PCA (Principal Component Analysis) or t-SNE could be used to extract the most important features from high-dimensional data, improving model training efficiency and reducing overfitting.

BIBLIOGRAPHY

Books:

1. Time Series Forecasting in Python -Marco Peixeiro
2. Python Machine Learning by Example -Yuxi (Hayden) Liu
3. Deep Learning with Python -Francois Chollet

Research Papers:

[1] "Profit Prediction Using ARIMA, SARIMA and LSTM Models in Time Series Forecasting: A Comparison" by Uppala Meena Sirisha, Manjula C Belavagi & Girija Attigeri.
[2] "On the Autoregressive Time Series Model Using Real and Complex Analysis", by Torsten Ullrich
[3] "XGBoost: A Scalable Tree Boosting System" by Tianqi Chen & Charlos Guestrin
[4] "Stock Price Prediction Using CNN and LSTM-Based Deep Learning Models" by Sidra Mehtab & Jaydip Sen
[5] "Understanding Random Forests: From Theory to Practice" by Gilles Louppe

Online Resources:

https://companiesmarketcap.com/inr/india/most-profitable-indian-companies/
https://seeve.medium.com/time-series-modeling-13f076d6d6ca
https://medium.com/@batuhansenerr/ai-powered-financial-analysis-multi-agent-systems-transform-data-into-insights-d94e4867d75d
https://towardsdatascience.com/exploring-the-lstm-neural-network-model-for-time-series-8b7685aa8cf
https://pub.towardsai.net/can-machine-learning-outperform-statistical-models-for-time-series-forecasting-73dc2045a9c5

APPENDIX

```
import yfinance as yf
import numpy as np
import pandas as pd
import matplotlib. pyplot as plt
from statsmodels.tsa. seasonal import seasonal_decompose
from statsmodels.tsa. arima. model import ARIMA
from statsmodels.tsa. statespace. sarimax import SARIMAX
from statsmodels.tsa. stattools import adfuller
from statsmodels. graphics. tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.ar_model import AutoReg
from statsmodels.tsa.api import VARMAX
from sklearn. metrics import mean_squared_error

data = yf. download ('RELIANCE.NS’, end = '2024-12-24')

plt. figure (figsize= (12,4))
plt. plot(data['Close'], label = 'Close')
plt. title ('Reliance Close Price Over Time')
plt. xlabel('Date')
plt. ylabel('Price')
plt. legend ()
plt. show ()

decomposition = seasonal_decompose(data['Close'], model = 'additive', period = 365)

plt. figure (figsize= (12,4))
plt. plot (decomposition. trend)
plt. title ('Trend over the Years')
plt. xlabel('Date')
plt. ylabel('Price')
plt. xticks (rotation = 45)
plt. show ()

plt. figure (figsize = (12,4))
plt. plot (decomposition. seasonal)
plt. title ('Seasonal Component over the Years')
plt. xlabel('Date')
plt. ylabel('Price')
plt. xticks (rotation = 45)
plt. show ()

result = adfuller(data["Close"])
print (f"ADF Statistic: {result [0]}")
print (f"p-value: {result [1]}")

# ACF Plot
plt. figure (figsize= (12, 6))
plt. subplot (1, 2, 1)
plot_acf(data["Close"], lags=40, ax=plt.gca ())
plt. title ('Autocorrelation Function (ACF)')

# PACF Plot
plt. subplot (1, 2, 2)
plot_pacf(data["Close"], lags=40, ax=plt.gca ())
plt. title ('Partial Autocorrelation Function (PACF)')

plt. tight_layout ()
plt. show ()

train = data['Close’] [:int (0.8*(len(data)))]. dropna ()
test = data['Close’] [int (0.8*(len(data))):]. dropna ()

model = AutoReg (train, lags=1)
model_fit = model.fit ()

AR_predictions=model_fit. predict(start=len(train), end=len(train)+len(test)-1, dynamic=False)
AR_predictions. index = test. index
plt. figure (figsize= (12,4))
plt. plot (train, label='Train')
plt. plot (test, label='Test')
plt. plot (AR_predictions, label='Predictions')
plt. title ('AR Model Predictions on test data')
plt. xlabel('Date')
plt. ylabel('Price')
plt. legend ()
plt. show ()

model = AutoReg (new_train, lags=1)
model_fit = model.fit ()
AR_future_prediction = model_fit. predict(start=len(new_train), end=len(new_train) + 30,
dynamic=False)

model = ARIMA (train, order= (0, 2, 1)) # (p, d, q) where p=2, d=2, q=1
model_fit = model.fit ()

MA_predictions = model_fit. predict (start = len(train), end = len(train)+len(test)-1,
dynamic=False)

model = ARIMA (new_train, order= (0, 2, 1)) # (p, d, q) where p=2, d=2, q=1
model_fit = model.fit ()
MA_future_prediction = model_fit. predict(start=len(new_train), end=len(new_train) + 30,
dynamic=False)

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models  import Sequential
from tensorflow.keras.layers import Dense, LSTM, SimpleRNN
from tensorflow.keras.layers import Conv1D, Dense, Flatten, Dropout, MaxPooling1D
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import mean_squared_error

# Create lag features
def create_lagged_features(series, lag=120):
    df = pd.DataFrame()
    for i in range(lag, 0, -1):
        df[f'lag_{i}'] = series.shift(i)
    df['target'] = series
    return df.dropna()

lagged_data = create_lagged_features(series, lag=3)
X = lagged_data.drop('target', axis=1)
```

```
y = lagged_data['target']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,shuffle = False)

# Scale the data
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = Sequential([
    Dense(64, activation='relu', input_dim=X_train.shape[1]),
    Dense(32, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1)  # Output layer
])

model.compile(optimizer='adam', loss='mse', metrics=['mae'])

%%capture
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2)

predictions = model.predict(X_test)
```

Streamlit code:

```
import pandas as pd
import numpy as np
import yfinance as yf
from tensorflow. keras. models import load_model
from sklearn. preprocessing import MinMaxScaler
from sklearn. model_selection import train_test_split
import streamlit as st

# Load data
data = yf. download ('RELIANCE.NS', end='2024-12-24')
series = data['Close']

# Create lagged features
def create_lagged_features (series, lag=120):
    df = pd. DataFrame ()
    for i in range (lag, 0, -1):
        df[f'lag_{i}'] = series. shift(i)
    df['target'] = series
    return df. dropna ()

# Prepare data
lagged_data = create_lagged_features (series, lag=3)
X = lagged_data. drop ('target', axis=1)
y = lagged_data['target']

# Split the data
X_train, X_test, y_train, y_test = train_test_split (X, y, test_size=0.2, shuffle=False)

# Scale the data
scaler = MinMaxScaler ()
X_train = scaler.fit_transform(X_train)
X_test = scaler. transform(X_test)

# Load trained models
models = {
    'ANN': load_model ('Reliance Models/ann.keras'),
    'CNN': load_model ('Reliance Models/cnn.keras'),
    'RNN': load_model ('Reliance Models/rnn.keras'),
    'LSTM': load_model ('Reliance Models/LSTM.keras')
}

# Predict on test data
predictions = {name: model. predict(X_test). flatten () for name, model in models. items ()}

# Future predictions
def predict future (model, last_data, steps=30):
    future_preds = []
    for _ in range(steps):
        scaled_input = scaler. transform (last_data. reshape (1, -1))
        next_pred = model. predict(scaled_input)
        future_preds. append (next_pred [0, 0])
        last_data = np. append (last_data [1:], next_pred)
    return np. array(future_preds)

last_lagged_data = X. iloc [-1]. values
future_predictions = {name: predict_future (model, last_lagged_data) for name, model in models.
items ()}

# Streamlit UI
st. title ("Reliance Stock Price Prediction")

# Model selection
model_choice = st. selectbox ("Select a Model to View Predictions:", list (models. keys ()))

st. subheader(f"{model_choice} Model - Test Data Predictions")
st. line_chart (pd. DataFrame ({
    'Actual': y_test. values,
    'Predicted': predictions[model_choice]
}, index=y_test. index). style.set_properties (
    **{'color': '#FF5733'} # Change this to the desired color code
))

st. subheader(f"{model_choice} Model - Predictions for next 30 Days")
future_dates = pd. date_range (series. index [-1] + pd. Timedelta(days=1), periods=30)
```

```
st. line_chart (pd. DataFrame ({
   'Future Predictions': future_predictions[model_choice]
}, index=future_dates))
```