Forecasting Stock Market Direction with Random Forest, XGBoost, and LSTM

Abstract

Predicting stock market trends has long been a subject of interest for researchers due to the market's complex and highly dynamic nature. The inherent volatility in stock prices makes accurate forecasting particularly challenging. Traditional forecasting and diffusion models, while useful, are often not sufficient to address the wide range of issues involved, especially in short-term predictions. Since forecasting errors are closely linked to market risk, reducing these errors is crucial to making more informed and less risky investment decisions.

In this study, we approach stock market prediction as a classification problem, using machine learning algorithms to predict the direction of price movements rather than their exact values. We explore and compare the performance of three different models: Random Forest, XGBoost, and Long Short-Term Memory (LSTM) networks. We incorporate several technical indicators, such as the Relative Strength Index (RSI) and the Moving Average Convergence Divergence (MACD), as input features to enhance model learning. Our aim is to evaluate how effectively these models can capture trends and assist in making more reliable market predictions.

1.Introduction

The stock market plays a critical role in the global economy, offering opportunities for investment and wealth creation. However, predicting stock market trends remains one of the most complex and challenging tasks due to the many uncertainties and factors that influence stock prices. Market behaviour is driven by a variety of variables, including economic conditions, investor sentiment, political developments, and company-specific news. These factors often interact in unpredictable ways, causing sudden fluctuations in stock prices and making the market highly volatile.

Stock prices are inherently dynamic, non-linear, and often chaotic in nature. They tend to exhibit random movements, particularly over short time periods, which makes accurate prediction even more difficult. Despite this randomness, long-term trends can sometimes emerge, where certain stocks may display more stable or linear patterns over extended periods. Nonetheless, the volatile and noisy character of stock prices poses significant risks for investors and traders. In such a high-risk environment, having the ability to predict future price movements becomes essential for making informed investment decisions. Traders, for instance, are more inclined to buy stocks they expect to rise in value while avoiding or selling stocks they anticipate will decrease in value. Therefore, accurate prediction of stock price trends is crucial for maximizing profits and minimizing potential losses.

Over the years, researchers have explored various methodologies to forecast stock prices. Among these, some of the prominent approaches include technical analysis, time series forecasting, machine learning and data mining techniques, and mathematical modelling of stock price volatility using differential equations. Each of these methods offers distinct advantages and limitations. This dissertation focuses primarily on machine learning and data mining methods, as they are well-suited for handling the vast and complex datasets associated with stock market prediction. Traditional techniques often struggle to process such large volumes of data effectively, whereas machine learning models are designed to uncover hidden patterns and relationships within big data, enabling more accurate and reliable predictions.

Early models used in stock forecasting primarily involved statistical methods such as time series models and multivariate analysis, as studied by Gencay (1999), Timmermann and Granger (2004), and Bao and Yang (2008). Gencay (1999), for example, investigated the predictability of spot foreign exchange rate returns using past buy-sell signals generated from simple technical trading rules, applying nearest neighbours and feedforward neural network regressions. In these approaches, stock price movement was treated as a function of time series data and addressed as a regression problem.

However, predicting the exact values of stock prices is highly difficult due to the chaotic nature of the market and its significant volatility. Stock prediction is often more effective when it is formulated as a classification problem rather than a regression problem. The goal is to design an intelligent model that can learn from market data using machine learning techniques and forecast future trends in stock price movements. The predictive outputs from these models can be used to support decision-making for individuals who invest in the stock market.

Researchers have employed various algorithms in this domain, including Support Vector Machines (SVM), Neural Networks, and Naive Bayesian Classifiers. The works of other authors related to this problem will be discussed in the following section.

2. Background and Related Work

The use of prediction algorithms to forecast stock market trends often stands in contrast to the Efficient Market Hypothesis (EMH) proposed by Fama and Malkiel (1970). According to EMH, stock prices already incorporate all relevant information, meaning that it should be impossible to consistently gain an advantage by analyzing historical stock data. If such an advantage did exist, the market would quickly correct itself as the information becomes widely known. Although this hypothesis is widely recognized, it remains a subject of debate. Some researchers have challenged EMH by developing algorithms capable of modelling the more complex dynamics of financial systems (Malkiel, 2003).

Over the years, a variety of algorithms have been applied to the stock prediction problem, including Support Vector Machines (SVM), Neural Networks, Linear Discriminant Analysis, Linear Regression, K-Nearest Neighbours (KNN), and Naive Bayesian Classifiers. Literature surveys indicate that SVM is one of the most frequently used algorithms in this domain.

For instance, Li, Li, and Yang (2014) studied how stock prices respond to external conditions, incorporating daily commodity prices such as gold, crude oil, natural gas, corn, and cotton, along with currency data in euros and yen. They compiled a comprehensive dataset containing daily trading data for 2,666 U.S. stocks listed on the NYSE and NASDAQ between January 1, 2000, and November 10, 2014. The features were derived from historical stock data combined with these external factors. Their study found that logistic regression achieved the best performance, with a success rate of 55.65%.

Similarly, Dai and Zhang (2013) utilized stock data from the 3M company spanning from September 2008 to November 2013, covering 1,471 data points. They applied multiple algorithms, including Logistic Regression, Quadratic Discriminant Analysis, and SVM, to both next-day and long-term prediction models. The next-day model achieved accuracies ranging from 44.52% to 58.2%, while the long-term model performed better, with the highest accuracy of 79.3% when the prediction window was set to 44 days. Dai and Zhang argued that their results reflect the semi-strong form of market efficiency, where neither technical nor fundamental analysis guarantees superior returns.

In another study, Xinjie (2014) analysed stock data for AAPL, MSFT, and AMZN from January 2010 to December 2014. They incorporated various technical indicators such as Relative Strength Index (RSI), On Balance Volume, and Williams %R to construct 84 features. An Extremely Randomized Tree algorithm (Geurts and Louppe, 2011) was used for feature selection, and the selected features were then used to train an RBF-kernelized SVM.

Devi, Bhaskaran, and Kumar (2015) proposed a hybrid model combining Cuckoo Search optimization with SVM using a Gaussian kernel. The Cuckoo Search technique was applied to optimize SVM parameters. Their model utilized technical indicators like RSI, Money Flow Index, Exponential Moving

Average (EMA), Stochastic Oscillator, and MACD. The data consisted of daily closing prices of the BSE-Sensex and CNX-Nifty, sourced from Yahoo Finance between January 2013 and July 2014.

An important contribution to this domain comes from Khaidem, Saha, and Dey (2016), who proposed using Random Forest classifiers for predicting the direction of stock prices. Their model applied exponential smoothing on time-series data and extracted features including RSI, Stochastic Oscillator, MACD, and On-Balance Volume. The model was evaluated across datasets from Apple Inc., GE, and Samsung Electronics, with impressive results. For instance, using the Apple dataset, the Random Forest model achieved 88.26% accuracy for 30-day (1-month) predictions, 93.06% accuracy for 60-day (2-month) predictions, and 94.53% accuracy for 90-day (3-month) predictions. Similarly, the accuracy for the Samsung and GE datasets also ranged from 85% to 94% across different time horizons. The study demonstrated that Random Forest, an ensemble method, effectively captures non-linear patterns in stock data, outperforming many existing algorithms.

The literature survey provides a clear understanding of the potential of using machine learning algorithms to predict the direction of stock prices, as well as how these methods have performed in previous studies. Building on this foundation, this study aims to apply Random Forest, an ensemble learning method, XGBoost, a gradient boosting technique known for its high predictive accuracy, and LSTM, a type of recurrent neural network (RNN) designed to handle sequential data and capture long-term dependencies. The performance of each of these models will be evaluated and compared to determine their effectiveness in stock market prediction.

Below, you can find a brief and intuitive explanation of the three models implemented in this study - Random Forest, Extreme Gradient Boosting (XGBoost), and Long Short-Term Memory (LSTM). These overviews focus on the conceptual understanding of each model and their relevance to stock market prediction, without going into detailed mathematical formulations.

2.1 Random Forest

Decision trees are widely used in machine learning due to their interpretability and flexibility. They can model complex, non-linear relationships by recursively partitioning the data based on feature values. However, when a decision tree is grown too deep in an attempt to capture highly irregular patterns, it often results in overfitting, a situation where the model performs well on training data but poorly on unseen data. This occurs because individual decision trees generally exhibit low bias but high variance, making them highly sensitive to noise and small fluctuations in the dataset.

Random Forests, an ensemble learning method, address this issue by building multiple decision trees using different subsets of the feature space and training data. By introducing randomness into both data and feature selection, Random Forests aggregate the predictions from diverse trees, thereby stabilizing the model and reducing overfitting. While this comes at the cost of slightly increased bias, the overall reduction in variance often leads to improved generalization performance.

In a decision tree, the dataset is recursively split at each node. These splits are guided by a splitting criterion, a mathematical function that quantifies the impurity (or disorder) in the data at a node. Two commonly used impurity measures are Gini Impurity and Shannon Entropy.

Gini Impurity measures the probability of incorrectly classifying a randomly chosen element from the dataset if it were labelled according to the class distribution at that node. It is defined as:

$$G(N) = \sum_{i \neq j} P(\omega_i) \times P(\omega_j)$$

Where $P(\omega_i)$ is the proportion of samples in node N that belong to class ω_i. The Gini impurity is lowest (i.e., 0) when the node is pure (all samples belong to one class).

Shannon Entropy, originally introduced in information theory, quantifies the amount of uncertainty or disorder in a system. In the context of decision trees, it measures how mixed the classes are within a node:

$$H(N) = -\sum_{i=1}^{d} P(\omega_i) \times log^2(P(\omega_i))$$

where d is the number of classes and $P(\omega_i)$ is the proportion of samples in node N with label ω_i. Entropy is maximized when all classes are equally represented in the node and minimized when the node contains samples from only one class.

To decide the optimal feature and value for splitting a node, decision trees use a heuristic called information gain, which represents the reduction in impurity achieved by the split. It is calculated as:

$$\Delta I(N) = I(N) - P_L \times I(N_L) - P_R \times I(N_R)$$

Here, I(N) is the impurity (Gini or Entropy) of the current node N, N_L and N_R are the left and right child nodes resulting from the split, P_L and P_R are the proportions of samples in N that go to the left and right nodes respectively. A split with the highest reduction in impurity (i.e., maximum information gain) is selected as the optimal choice at that node.

At the core of Random Forest lies the technique of bootstrap aggregating, commonly known as bagging. Bagging involves training each decision tree on a randomly sampled subset of the training data, drawn with replacement. The randomness ensures that each tree in the forest learns different patterns. During prediction, the outputs of individual trees are aggregated usually via majority voting in classification tasks to produce the final decision.

This ensemble approach improves the stability and accuracy of the model and is particularly effective at mitigating the high variance characteristic of single decision trees. By averaging the predictions of multiple weak learners (shallow or deep trees), bagging enhances the overall generalization ability of the model.

2.2 XGBoost Classifier

Similar to Random Forest, XGBoost (Extreme Gradient Boosting) is an ensemble learning method. However, unlike Random Forest, which uses bagging, XGBoost is based on boosting. In boosting, decision trees are built sequentially, with each new tree focusing on correcting the errors (residuals) made by the previous trees. This process gradually improves the overall model performance by reducing the remaining errors at each step.

In contrast to bagging techniques, which often use fully grown trees, boosting typically uses shallow trees (weak learners) with limited splits. The number of trees, learning rate, maximum depth of each tree, and other hyperparameters can be fine-tuned to achieve optimal accuracy.

The general steps involved in boosting can be summarized as follows:

Initially, a simple model F_0 is trained to predict the target variable y, resulting in an initial set of errors or residuals (y - F_0). A new model h_1 is then trained to predict these residuals. The predictions of h_1 are combined with F_0 to form an updated model F_1, which reduces the overall error compared to F_0:

$$F_1(x) = F_0(x) + h_1(x)$$

This process is repeated iteratively, with each new model h_m being trained on the residual errors of the previous combined model F_{m-1}:

$$F_m(x) = F_{m-1}(x) + h_m(x)$$

After several iterations, the final boosted model F_m is obtained, which achieves significantly lower prediction error.

In this project, XGBoost Classifier applies the same principle but is designed for classification tasks, where the residuals are computed based on classification errors, and the model outputs probabilities that are later converted into class predictions.

2.3 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) designed specifically to handle sequential and time-series data. While traditional machine learning algorithms like Random Forest and XGBoost treat each data point independently, LSTM is capable of learning patterns over time by remembering past information in the sequence. This makes it particularly useful for financial forecasting, where the temporal relationships between stock prices are crucial.

The key feature of LSTM is its ability to retain information over long sequences using memory cells and gates. These gates called input, output, and forget gates allow the network to decide what information to keep, what to discard, and what to output at each step in the sequence. This structure helps LSTMs overcome the limitations of standard RNNs, such as the vanishing gradient problem, which hampers learning from long-term dependencies.

Forget gate determines what portion of the previous cell state c_{t-1} should be retained.

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

Input gate controls how much new information is added to the cell state.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

The cell state c_t is updated by combining the previous memory (modulated by the forget gate) and the new candidate memory (modulated by the input gate).

$$c_t = f_t \odot c_{t-1} + i_t \odot \widetilde{c_t}$$

Output gate, This gate determines how much of the memory cell is exposed to the output (hidden state).

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

Here, x_t: input vector at time t (e.g., stock price features), h_{t-1}: hidden state from the previous time step, σ: sigmoid activation function, $tanh$: hyperbolic tangent function, $\odot$: element-wise multiplication, W_f, W_i, W_c, W_o: weight matrices, b_f, b_i, b_c, b_o: bias terms.

In the context of this study, LSTM models are trained on sequences of historical stock market data to predict the direction of future price movements. By feeding in a sliding window of past values and associated technical indicators, the model can capture both short-term fluctuations and long-term trends in the S&P 500 index.

Unlike tree-based models, LSTMs learn the temporal dynamics of the data without relying on handcrafted rules for feature importance or data splits. Instead, they learn directly from raw sequences through backpropagation, adjusting internal weights to minimize prediction errors.

While LSTMs are powerful in capturing time-based relationships, they are computationally more intensive and require careful tuning of parameters such as sequence length, number of layers, learning rate, and batch size. However, when trained effectively, they can uncover deeper patterns in stock price behaviour that static models might miss.

3. Methodology and Analysis

The learning algorithms used in this study are Random Forest, XGBoost, and LSTM. The process begins with the collection of time-series stock market data, which is then smoothed using exponential smoothing to reduce noise and fluctuations. After smoothing, a set of technical indicators is extracted, which are key variables that help provide insights into the potential future behaviour of stock prices. These extracted features are then used to train the selected models. The following sections will discuss each step of the process in detail.

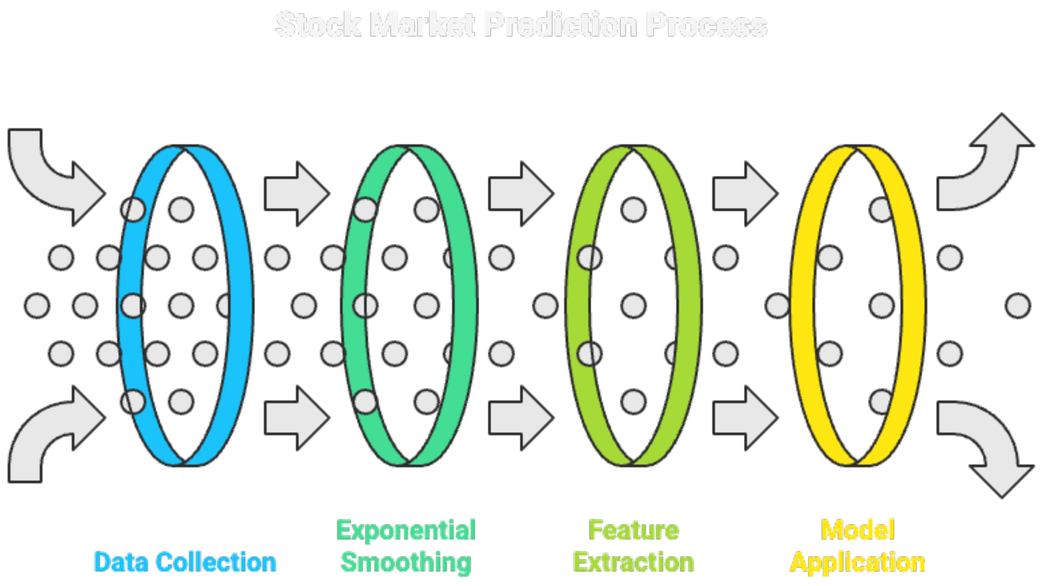

Fig 1: Proposed methodology

3.1 Data Preprocessing

For this study, the data used across all models Random Forest, XGBoost, and LSTM is based on the S&P 500 index, obtained from Yahoo Finance. The dataset covers the period from January 1990 to July 2025, providing a comprehensive historical time series of the index's prices. The objective is to predict the future direction of the S&P 500 index, rather than the exact price values, by leveraging this historical data.

Before deriving features or feeding the data into the models, the time series data is first smoothed using exponential smoothing. Exponential smoothing is a technique that assigns greater weight to recent observations while giving exponentially decreasing weight to past data points. This helps reduce random noise in the data and allows the models to better capture long-term price trends. The smoothed statistic of a time series Y is computed recursively as:

$$S_0 = Y_0$$

$$S_t = \alpha \times Y_t + (1 - \alpha) \times S_{t-1} \quad \text{for } t > 0$$

Here, α is the smoothing factor, with $0 < \alpha < 1$. A higher α results in less smoothing, giving more emphasis to recent data points. When $\alpha = 1$, the smoothed value is simply equal to the latest observation.

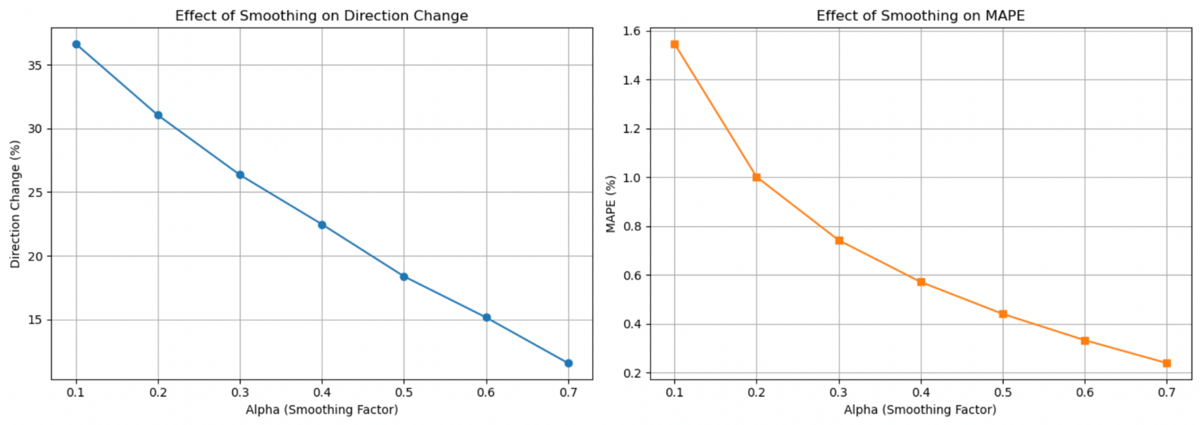

Fig 2: Effect of Smoothing Factor (α) on Direction Change and MAPE

To determine the most appropriate value for the smoothing parameter α in Single Exponential Smoothing (SES), I evaluated two key metrics across varying alpha values (α = 0.1 to 0.7). Rather than relying solely on visual inspection of smoothed curves, the goal was to quantitatively assess both the similarity of the smoothed series to the original data and the extent to which smoothing alters the direction of price movement.

The first metric used was the Mean Absolute Percentage Error (MAPE), which measures how close the smoothed values are to the actual values. The formula is given by:

$$MAPE = (1/n) \times \Sigma_i =^{1}\, {}^{n}|(x_i - \hat{x_i})/x_i| \times 100$$

where x_i is the actual value, $\hat{x}_i$ is the smoothed value, and n is the total number of data points.

The second metric was Mismatch Percentage, which captures how often the direction of price movement (i.e., whether the price goes up or down) changes after smoothing. This is computed using the following formula:

$$Mismatch\% = (Number\ of\ mismatches\ in\ direction/Total\ number\ of\ data\ points) \times 100$$

Both metrics were plotted across various α values in Figure 2. The analysis revealed two key observations: as α increases, the percentage of direction changes (Mismatch %) decreases. Additionally, MAPE also decreases, but the change becomes smaller after α=0.3.

To complement this analysis, I also evaluated the model performance for different α values. Among the tested values, α = 0.3 consistently produced the highest prediction accuracy for both next-day and multi-day forecasts. This suggests that α = 0.3 offers a strong balance. It smooths out short-term noise without overly distorting the signal or trend in the data.

Based on these quantitative metrics, plotted trends, and model accuracy, α = 0.3 emerges as the most suitable smoothing factor for this study. It maintains the underlying structure of the data while improving signal clarity, making it a robust choice for further modeling and analysis.

Once the data is smoothed, technical indicators are calculated from the processed time series. These indicators serve as derived features, capturing various aspects of stock price behavior and market trends. The resulting features are then organized into a feature matrix, which is used to train the models.

The target variable for each data point is defined based on the change in closing prices after a specified number of days, denoted as d. Specifically, the target for the i$^{\text{th}}$ day is calculated as:

$$\text{target}_i = \text{Sign}(close_{i+d} - close_i)$$

A target value of +1 indicates a positive price shift after d days, while -1 indicates a negative shift. These target values are assigned as labels corresponding to each row in the feature matrix, enabling the models to learn and predict the future direction of stock prices.

3.2 Feature Extraction

Technical indicators are important parameters calculated from time-series stock data that aim to assist in forecasting the direction of financial markets. These indicators are widely used by investors to identify potential bullish (upward) or bearish (downward) signals in the market. In this study, different sets of features, including various technical indicators, were used for each of the models employed. The details of the features used for each model are outlined in the following sections.

3.2.1 Features Used for Random Forest (RF)

For the Random Forest model, a set of features was derived from the exponentially smoothed S&P 500 time series data. These features were designed to capture both price ratios and trend information over different time horizons, which can help the model understand both short-term and long-term market behaviours.

I attempted to replicate the approach proposed by Khaidem, Saha, and Dey (2016), who utilized Random Forest classifiers to predict the direction of stock prices. They employed a set of technical indicators - Relative Strength Index (RSI), Stochastic Oscillator, Williams %R, Moving Average Convergence Divergence (MACD), Price Rate of Change (ROC), and On-Balance Volume (OBV) as input features. In their study, they achieved a prediction accuracy of 86.83% and precision of 88.18% for a 30-day ahead forecast on Samsung stock.

In contrast, applying the same set of features and methodology to the S&P 500 index, achieved a lower accuracy of 53.35% and precision of 66.95% for the same 30-day prediction window. This might be due to the fact that broad market indices like the S&P 500, which aggregate the performance of hundreds of companies, tend to exhibit smoother and less volatile price movements compared to individual stocks. However, this aggregation can make them less responsive to technical indicators, which are generally more effective at capturing short-term patterns and fluctuations in individual stock behaviour than in composite index trends. Additionally, the original study may have applied different feature engineering techniques, time windows, or tuning strategies that were not fully disclosed, making exact replication difficult and potentially contributing to the variation in predictive performance.

To identify the most effective features for Random Forest in this study, I adopted an experimental approach. Various combinations of technical indicators and features derived from the closing price were tested by training multiple Random Forest models. The feature sets that resulted in higher accuracy for next-day price direction prediction were selected for the final model.

The features selected for the Random Forest model include Close_Ratio_2, Trend_2, Close_Ratio_5, Trend_5, Close_Ratio_60, Trend_60, Close_Ratio_250, Trend_250, Close_Ratio_1000, and Trend_1000. Each of these features was computed based on specific rolling windows (horizons): 2, 5, 60, 250, and 1000 days.

The Close Ratio for each horizon measures the ratio between the current closing price and the average closing price over the given horizon. This helps capture whether the current price is relatively high or low compared to its historical average over that period.

$$\text{Close\_Ratio\_horizon} = \frac{\text{Current Close Price}}{\text{Rolling Average Close Price over horizon}}$$

The trend for each horizon counts the number of days the price increased within the specified window, providing a sense of how consistently the market has been moving upwards.

$$\text{Trend\_horizon} = \text{Sum of positive price changes over the past horizon days}$$

By combining the relative price position (Close Ratio) with the historical trend of price increases (Trend) across various time windows, the Random Forest model is provided with a comprehensive view of the stock's short-term and long-term behavior, aiding in the prediction of future price direction.

3.2.2 Features Used for XGBoost

For the XGBoost model, a diverse set of features was derived from the exponentially smoothed S&P 500 index data, aiming to capture essential aspects of price trends, volatility, momentum, and trading activity. To determine the most effective set of inputs, a series of experiments were conducted using different combinations of technical indicators and engineered variables. The final selection was guided not only by improvements in predictive accuracy but also through an analysis of feature importance using XGBoost's built-in feature_importances_ attribute. This allowed for a more data-driven refinement of the feature set, ensuring that only the most informative predictors were retained. This iterative and interpretive process helped optimize the model's ability to generalize across different time horizons.

The features used for XGBoost are: Trend_2, Close_Ratio_5, Trend_60, Boll_Width, Garman_Klass, Rolling_Mean_250, Close_Ratio_2, Close_Ratio_60, Dollar_Volume, and Rolling_Mean_5.

These features were computed as follows:

Trend_2, Trend_60: These features count the number of days within the past 2-day and 60-day windows, respectively, where the price increased. This helps the model understand recent upward trends.

Close_Ratio_2, Close_Ratio_5, Close_Ratio_60: These features represent the ratio of the current smoothed closing price to the rolling average closing price over 2, 5, and 60 days, indicating how the current price compares to its historical average in different time frames.

Rolling_Mean_5, Rolling_Mean_250: These are the simple moving averages of the smoothed closing price calculated over 5-day and 250-day windows, reflecting short-term and long-term trends.

Boll_Width: This represents the Bollinger Band Width, calculated as the normalized difference between the upper and lower Bollinger Bands over a 20-day window. It captures the volatility of the stock by measuring the spread of price movements.

$$\text{Boll\_Width} = \frac{\text{Upper Band} - \text{Lower Band}}{\text{Rolling Mean}}$$

Where:

$$\text{Upper Band} = \text{Rolling Mean} + 2 \times \text{Rolling Standard Deviation}$$
$$\text{Lower Band} = \text{Rolling Mean} - 2 \times \text{Rolling Standard Deviation}$$

Garman_Klass: The Garman-Klass volatility estimator provides a measure of price volatility using high, low, open, and close prices. It is considered more efficient than traditional volatility measures.

$$\text{Garman\_Klass} = 0.5 \times \left(\ln \frac{\text{High}}{\text{Low}}\right)^2 - (2\ln 2 - 1) \times \left(\ln \frac{\text{SES\_Close}}{\text{Open}}\right)^2$$

Dollar_Volume: This feature is computed by multiplying the smoothed closing price with the trading volume, providing an estimate of the total dollar value traded, which reflects the liquidity and market activity.

$$\text{Dollar\_Volume} = \text{SES\_Close} \times \text{Volume}$$

This carefully engineered feature set enables the XGBoost model to capture both price dynamics and underlying market behaviours effectively.

3.2.3 Features Used for LSTM

For the LSTM model, a comprehensive set of features was constructed to capture a wide range of market dynamics, including price trends, momentum, volatility, and historical price dependencies. The selection of features was guided by an experimental approach, where multiple combinations of technical indicators and time-dependent variables were tested to assess their predictive utility. Through iterative experimentation and evaluation, the most effective set of features was identified based on the model's accuracy and consistency across different forecasting horizons. This data-driven feature engineering process ensured that the LSTM model was provided with inputs that could effectively capture temporal patterns and underlying market behavior.

The features selected for training the LSTM model are:

SES_Close, Close_Ratio_2, Trend_2, Close_Ratio_5, Trend_5, Close_Ratio_60, Trend_60, Close_Ratio_250, Trend_250, MACD, RSI, Stoch_d, OBV, Lag_1, Lag_2, Lag_3, Lag_4, Lag_5, Lag_10, SMA_5, SMA_10, SMA_20, EMA_5, EMA_10, EMA_20.

In addition to the features described previously, the following technical indicators and lag features were specifically calculated for the LSTM model:

RSI (Relative Strength Index): A momentum oscillator that measures the speed and change of price movements, typically calculated over a 14-period window.

$$\text{RSI} = 100 - \frac{100}{1 + RS}$$

Where:

$$RS = \frac{\text{Average Gain over 14 periods}}{\text{Average Loss over 14 periods}}$$

MACD (Moving Average Convergence Divergence): A trend-following momentum indicator that shows the relationship between two exponential moving averages (EMAs) of closing prices.

$$\text{MACD} = EMA_{12} - EMA_{26}$$

Stochastic Oscillator (Stoch_d): A momentum indicator comparing a particular closing price to a range of prices over a certain period, helping to identify overbought or oversold conditions.

$$\text{Stoch\_d} = \text{3-period SMA of Stochastic\_k}$$

Where,

$$\text{Stochastic\_k} = \frac{\text{SES\_Close} - \text{Lowest Low (n)}}{\text{Highest High (n)} - \text{Lowest Low (n)}} \times 100$$

Where, SES_Close is the smoothed close price

OBV (On-Balance Volume): A volume-based indicator that relates volume flow to price changes, providing insights into the strength of price trends.

Lag Features: Past values of the SES_Close are included as lag features, specifically the closing prices lagged by 1, 2, 3, 4, 5, and 10 days to help the model recognize patterns over recent days.

Simple Moving Averages (SMA): The average of the closing prices over fixed window periods of 5, 10, and 20 days.

$$SMA_n = \frac{\sum_{i=0}^{n-1} \text{SES\_Close}_{t-i}}{n}$$

Exponential Moving Averages (EMA): A type of moving average that gives more weight to recent prices, calculated over 5, 10, and 20 days.

$$EMA_t = \alpha \times \text{SES\_Close}_t + (1 - \alpha) \times EMA_{t-1}$$

Where:

$$\alpha = \frac{2}{n+1}$$

3.3 Test for Linear Separability

Before training the models, we tested whether the two classes in our dataset could be separated using a straight line. This concept is called linear separability. If two classes are linearly separable, it means there’s a clear boundary (like a straight line in 2D space) that can divide one class from the other without any overlap.

To check this, we used a method involving convex hulls. Think of wrapping a rubber band around each group of data points, the shape formed is the convex hull. If these hulls don’t overlap, the classes are linearly separable. But if they intersect, they are not.

Since our data has many features, we applied Principal Component Analysis (PCA) to reduce it to two dimensions for easier visualization. As shown in the Figure 3, the convex hulls of the two classes overlap significantly, meaning the classes are not linearly separable.

To further validate this observation, I applied Principal Component Analysis (PCA) with three components and visualized the data in 3D (Figure 3). This additional perspective again reveals significant overlap between the two classes, reinforcing the conclusion that the dataset is not linearly separable. Even in three dimensions, there is no evident hyperplane that can cleanly divide the classes without misclassification.

This result justifies the use of non-linear models like Random Forest, XGBoost, and LSTM, which can learn complex boundaries and interactions in the data. These models are more suitable than simpler linear models like Linear Discriminant Analysis (LDA), which assume the classes can be separated by a straight line.

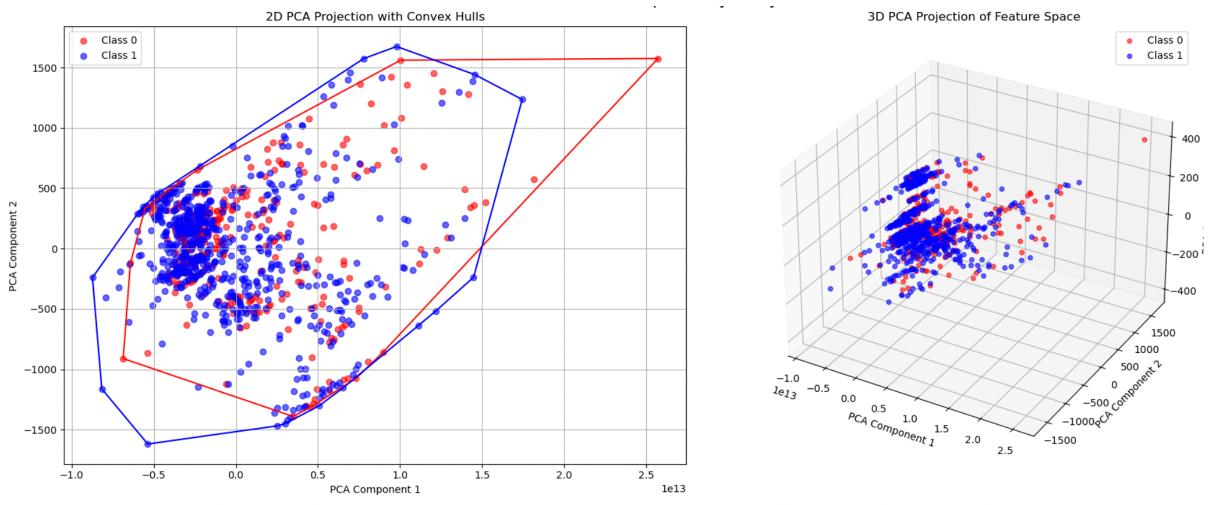

Fig 3: PCA based class separability analysis

3.4 Implementation

3.4.1 Random Forest

The Random Forest classifier was used to predict the directional movement of the S&P 500 index one day ahead. After data preprocessing, which included exponential smoothing and the extraction of technical indicators and engineered features, all predictors were shifted appropriately to prevent data leakage. The target variable was defined as a binary label indicating whether the index price increased the following day. The same framework was later extended to predict market direction for longer horizons, including 5-day, 10-day, 20-day, 30-day, and 40-day ahead forecasts.

For model training and evaluation, the dataset was split in an 80-20 chronological manner, where the first 80% of the data was used for training and the remaining 20% was reserved for testing. This time-aware split preserves the sequential nature of financial time series data, ensuring that the model only learns from past information and is evaluated on future, unseen data. Such an approach prevents data leakage and avoids introducing lookahead bias, which could occur if the data were randomly shuffled. This setup more closely reflects real-world forecasting conditions where models are expected to make predictions based solely on historical trends.

In addition to this final test set, back testing was implemented to evaluate how the model would have performed across different historical market conditions. The back testing approach followed a walk-forward validation method. Initially, the first 2500 data points equivalent to approximately 10 years of data were used to train the model, which was then tested on the data from the 11th year. This process was repeated iteratively; each time, the training window moved forward by one year to predict the subsequent year. This rolling-window approach allowed us to observe how well the model generalized over different market regimes and avoided overfitting to a particular period.

To further improve model performance, a manual hyperparameter tuning process was conducted. Several combinations of n_estimators, min_samples_split, and max_depth were tested. For next-day prediction, the best performance was observed when the Random Forest was configured with 100 decision trees (n_estimators=100), a minimum sample split of 100 (min_samples_split=100), and a tree depth capped at 5 (max_depth=5). These settings struck a good balance between model complexity and generalization, helping reduce overfitting while capturing relevant patterns in the data.

A thresholding technique was also applied to improve classification reliability. Instead of using the default probability threshold of 0.5 for predicting class labels, we used the predict_proba method of the classifier and introduced a custom threshold of 0.6. This means the model would only predict a positive movement (i.e., price increase) when it was at least 60% confident, allowing for more conservative but reliable predictions.

This systematic approach combining thoughtful feature engineering, time-aware validation, hyperparameter tuning, and calibrated classification formed the core of the Random Forest implementation. Once the model was optimised for next-day predictions, the same training pipeline was extended to generate forecasts for longer-term horizons, including 5-day, 10-day, 20-day, 30-day, and 40-day ahead predictions. This was achieved by modifying the target variable to reflect the respective forecast horizon while keeping the rest of the methodology consistent. Additionally, for each prediction horizon, separate hyperparameter tuning was performed to ensure that the model was appropriately adapted to the different temporal dynamics and patterns associated with each forecasting window.

3.4.2 XGBoost

The Extreme Gradient Boosting model, widely known as XGBoost, was implemented using the same historical S&P 500 index dataset spanning from January 1990 to July 2025. As part of the preprocessing pipeline, exponential smoothing was first applied to the raw index data. This helped suppress short-term fluctuations and highlight underlying long-term patterns that are more relevant for trend prediction. From this smoothed data, a diverse range of technical indicators and engineered features were extracted, including momentum indicators, moving averages, volume-based metrics, and lagged values. These features were carefully shifted to ensure no overlap between current input data and future outcomes, thereby eliminating the risk of data leakage, a critical step in time series modelling.

The dataset was divided using an 80-20 time-based split strategy, where the first 80% of the data served as the training set, and the remaining 20% was used exclusively for testing. This chronological split preserves the temporal nature of the data and more accurately simulates real-world forecasting scenarios where models are deployed to predict future trends based on past information. No shuffling was applied during the split, further maintaining the order of events as they unfolded historically.

XGBoost, being a powerful ensemble learning algorithm based on gradient-boosted decision trees, was selected for its ability to model complex non-linear relationships and interactions between features. To enhance interpretability and control over the model's confidence in its predictions, the predict_proba method was used during inference. A classification threshold of 0.6 was applied, meaning that the model only predicted a positive class (price increase) if it assigned a probability greater than 60% to that outcome. This thresholding approach provides a more conservative and potentially more robust signal, reducing false positives in uncertain situations.

To identify the most effective combination of hyperparameters, I used a manual tuning approach. This method allowed me to incorporate back testing alongside model training, enabling a more comprehensive evaluation of performance across different time periods. By adjusting hyperparameters and observing results over various historical scenarios, I ensured that the model maintained consistent and reliable performance under changing market conditions while preserving the temporal integrity of the time series data.

Initially, the model was trained to forecast the direction of the market for the very next trading day. Once the optimal pipeline for next-day prediction was established, it was extended to accommodate longer-term forecasts. Specifically, the target variable was redefined to capture the market movement after 5, 10, 20, 30, and 40 trading days. For each of these extended horizons, the same training framework was applied including feature extraction, time-aware splitting, and hyperparameter tuning ensuring that the model was consistently evaluated and optimized for various future intervals. This step-wise expansion allows for comparative performance analysis across different temporal scales and reveals how well the model adapts to short-term versus long-term market dynamics.

3.4.3 LSTM

LSTM is a type of recurrent neural network (RNN) particularly suited for time series data, as it is capable of learning long-term dependencies and patterns by retaining information over time. This makes it a powerful model for sequential financial data where current trends often depend on historical context.

The same preprocessing steps such as exponential smoothing and the extraction of technical indicators were applied to the dataset, following the process outlined in the feature engineering section. For the LSTM model, the data was then reformatted into a structure suitable for sequence-based learning. A sequence generation approach was used, where a sliding window of past observations (look-back period) was created for each point in the dataset. Specifically, for each time step, the previous 30 days of feature data were grouped into a sequence, and the target label corresponding to the day immediately following the window was assigned as the output. This structure allows the model to learn from short-term temporal patterns present in the financial data. A look-back window of 30 was selected after trying multiple values, as it consistently provided better predictive performance while keeping the model complexity manageable.

Once the sequences were generated, the dataset was divided into training, validation, and testing sets in a time-aware manner to avoid data leakage. 70% of the sequences were allocated to training, 15% to validation, and the remaining 15% to testing. This sequential split respects the chronological order of the data and ensures that no future information leaks into the past.

To optimize the architecture and performance of the LSTM model, a manual hyperparameter tuning approach was adopted. This method involved systematically testing different configurations through controlled experiments, with the goal of identifying the most effective combination of parameters for learning temporal dependencies in stock price data. Manual tuning allowed for greater interpretability and hands-on control over how the model responded to various changes in its structure and learning process, which was especially valuable given the complexity and noise inherent in financial time series.

Several key hyperparameters were selected for tuning, each playing a critical role in the model's learning dynamics and generalization capacity. The number of LSTM units was varied to assess how well the model could capture sequential patterns and retain important information across time steps. Adjusting this value allowed the model to balance between underfitting and overfitting based on the complexity of the data. The dropout rate was another essential parameter, included as a regularization mechanism to reduce overfitting by randomly deactivating a subset of neurons during training. This helped prevent the model from becoming overly dependent on specific pathways in the network.

The learning rate was also tuned carefully, as it directly influences the speed and stability of convergence during training. A rate that is too high can lead to erratic updates and failure to converge, while a rate that is too low can cause the model to get stuck in suboptimal solutions. Lastly, L2 regularization strength was explored to penalize large weights in the network, promoting smoother learning and better generalization to unseen data. By adjusting these parameters manually and monitoring the model's performance across various prediction horizons, an effective configuration was identified and used consistently in the final model evaluations.

The final LSTM architecture consisted of a single LSTM layer, which processes sequences of input features and captures time-dependent patterns, followed by a Dropout layer to regularize the network and reduce the risk of overfitting. This was then connected to a fully connected (Dense) output layer with a sigmoid activation function, suitable for binary classification tasks such as predicting whether the stock price will rise or not.

The model was compiled using the binary cross-entropy loss function, which is standard for binary classification problems, and the Adam optimizer, known for its adaptive learning rate and robust performance across a wide range of tasks. To further enhance training stability and prevent the model from overfitting, an early stopping mechanism was introduced. This technique monitors the validation loss during training and halts the process if the model fails to improve for 5 consecutive epochs, restoring the best weights encountered during the training process.

Following training, the model's performance was evaluated on the test set using accuracy and precision metrics. Predictions were generated using the predict_proba method (interpreted as the model's confidence), and a threshold of 0.6 was applied to classify an observation as class 1 (indicating the market would go up). That is, if the model assigned a probability greater than 60% to class 1, the prediction was taken as positive.

Overall, the LSTM model demonstrated strong performance in capturing complex, non-linear temporal patterns in the financial time series, complementing the tree-based models discussed earlier.

3.5 Model Performance Metrics

Using the prediction results produced by the model, we can make trading decisions—whether to buy or sell a stock. If the prediction is +1, it indicates that the price is expected to rise after *n* days, and the suggested decision is to buy. Conversely, if the prediction is –1, it suggests the price will fall after *n* days, and the recommended action is to sell.

However, incorrect predictions can lead to significant financial losses. Therefore, it is essential to evaluate the robustness of the model using standard performance metrics for binary classification, namely: Accuracy, Precision, Recall (Sensitivity), Specificity and f1 score.

Accuracy measures the proportion of total correct predictions (both positive and negative) among all predictions.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision measures the proportion of correctly predicted positive instances among all instances predicted as positive.

$$Precision = \frac{TP}{TP + FP}$$

Recall (Sensitivity) indicates the model's ability to correctly identify actual positive cases.

$$Recall = \frac{TP}{TP + FN}$$

Specificity assesses the model's ability to correctly identify actual negative cases.

$$Specificity = \frac{TN}{TN + FP}$$

F1 Score measures the balance between precision and recall. It is particularly useful in cases of class imbalance, as it takes both false positives and false negatives into account. The F1 Score is the harmonic mean of precision and recall.

$$F1\ score = (2 \times \text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})$$

The Receiver Operating Characteristic (ROC) curve is a widely used graphical tool for evaluating the performance of a binary classification model. It visualizes the diagnostic ability of the classifier by plotting the True Positive Rate (TPR), also known as recall or sensitivity against the False Positive Rate (FPR), which is calculated as *(1 – specificity)*, across a range of decision thresholds.

Each point on the ROC curve represents a different threshold used to convert predicted probabilities into class labels. As the threshold varies, the trade-off between sensitivity (correctly identifying positive cases) and specificity (correctly identifying negative cases) becomes visible. This trade-off is crucial when the costs of false positives and false negatives differ.

A model with high discriminatory power will have an ROC curve that bows sharply towards the top-left corner of the plot. This indicates that the classifier achieves a high TPR while keeping the FPR low, suggesting strong performance. Conversely, a curve that lies close to the 45-degree diagonal represents a model with no discriminatory ability, essentially performing no better than random guessing.

The Area Under the ROC Curve (AUC) provides a single scalar value to summarize the overall performance of the model. AUC values closer to 1.0 indicate excellent classification ability, whereas values around 0.5 suggest poor performance.

ROC curves are particularly valuable when comparing multiple models. They allow one to visually assess which model achieves a better balance between sensitivity and specificity, and to discard models that are clearly suboptimal.

4. Results

This study evaluates the predictive performance of three models - LSTM, Random Forest, and XGBoost across varying prediction horizons, ranging from 1 to 40 days. The accuracy scores at each forecast interval were used to assess the effectiveness of each model in predicting the directional movement of stock prices. These results are visualized in Figure 4.

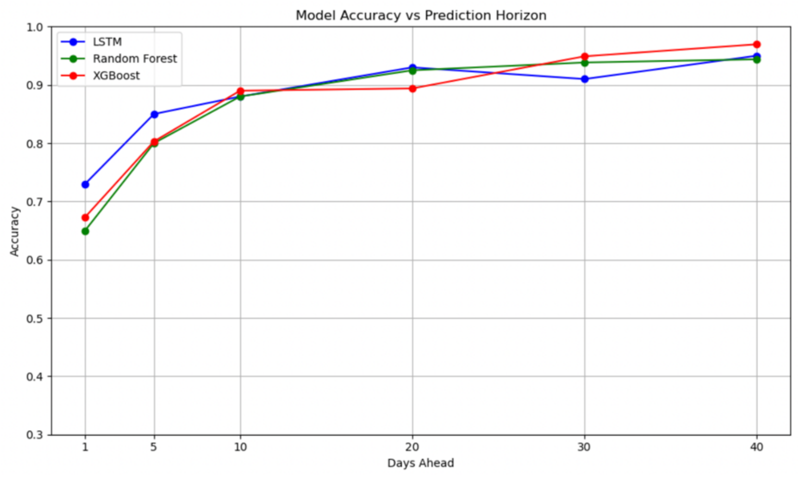

Fig 4: Model Accuracy Across Different Prediction Horizons (1 to 40 Days)

Across all prediction horizons, Random Forest and XGBoost consistently outperformed LSTM. At the shortest horizon (1-day ahead), LSTM achieved an accuracy of 63%, while Random Forest and XGBoost performed slightly better, reaching 65% and 67.3% respectively. As the prediction window extended, this performance gap became more apparent. At the 10-day mark, LSTM reached 85% accuracy, while Random Forest and XGBoost recorded higher values of 88% and 89%. This trend continued over longer timeframes, with XGBoost in particular exhibiting a strong and steady increase in performance, culminating in a peak accuracy of 96.96% at 40 days ahead the highest among all models.

Random Forest also demonstrated robust results throughout the range, ending with an accuracy of 94.39% at 40 days. While LSTM showed steady improvement with increased prediction horizons, it consistently trailed behind the ensemble-based models. The superior performance of Random Forest and XGBoost, especially over medium and long-term forecasts, suggests their effectiveness in capturing more complex patterns and interactions in the dataset.

An additional observation from the accuracy curves is the rate of model improvement across different horizons. All models show a significant rise in accuracy during the early phase, particularly between 1 to 10 days ahead, where the slope of the curve is noticeably steep. However, after the 10-day mark, the rate of improvement begins to taper off, as evidenced by the flattening of the curves in Figure 4. This suggests that while predictive accuracy increases with longer horizons, the marginal gains become smaller, indicating diminishing returns beyond the short-to-medium term window.

These findings highlight not only the superiority of ensemble learning methods like Random Forest and XGBoost over LSTM, but also the presence of a performance plateau in accuracy gains as the forecast horizon extends. While LSTM remains a viable option for sequence modelling, it appears less competitive than the ensemble-based models within this study's framework, particularly for medium- to long-term predictions.

Days Ahead	LSTM	Random Forest	XGBoost
1	0.63	0.65	0.673
5	0.73	0.8	0.803
10	0.85	0.88	0.89
20	0.88	0.925	0.8939
30	0.93	0.9384	0.949
40	0.91	0.9439	0.9696

Table 1: Model Accuracy Across Different Prediction Horizons

4.1 Random Forest

For the next-day prediction task, the Random Forest classifier achieved an accuracy score of 0.65, indicating a solid baseline performance in classifying short-term stock price movements. To ensure the model's robustness over time, a rolling back test was conducted, beginning from the 10th year of historical data. In this setup, the model was retrained annually using all prior data and tested on the following year, simulating real-world deployment and avoiding lookahead bias.

This rolling evaluation revealed consistent results, with yearly prediction accuracy ranging between 0.608 and 0.712. Such stability indicates that the model generalizes reasonably well across varying market conditions, maintaining its performance despite the inherent volatility of financial data.

In terms of classification performance, the model achieved a precision score of 0.746, meaning that when the model predicted an upward price movement, it was correct roughly 74.6% of the time. The recall score of 0.6177 reflects its ability to correctly identify 61.77% of all actual upward movements, while the specificity of 0.6636 indicates it correctly identified 66.36% of the actual downward movements. This suggests that the model is slightly better at predicting negative movements (trends going down) than capturing all positive movements. The F1 score of 0.6760, which balances precision and recall, highlights that the model maintains a relatively even trade-off between accurately detecting upward price changes and minimizing false positives.

To further assess the model's discriminative power, a Receiver Operating Characteristic (ROC) curve was plotted (see Figure 5). The ROC curve illustrates the trade-off between the true positive rate (recall) and the false positive rate across different classification thresholds. The resulting Area Under the Curve (AUC) of 0.69 indicates a moderate ability to distinguish between positive and negative classes. While not exceptionally high, an AUC of 0.69 still represents meaningful predictive value, especially in financial time series contexts, where market noise, regime shifts, and non-stationarity make high discrimination particularly challenging.

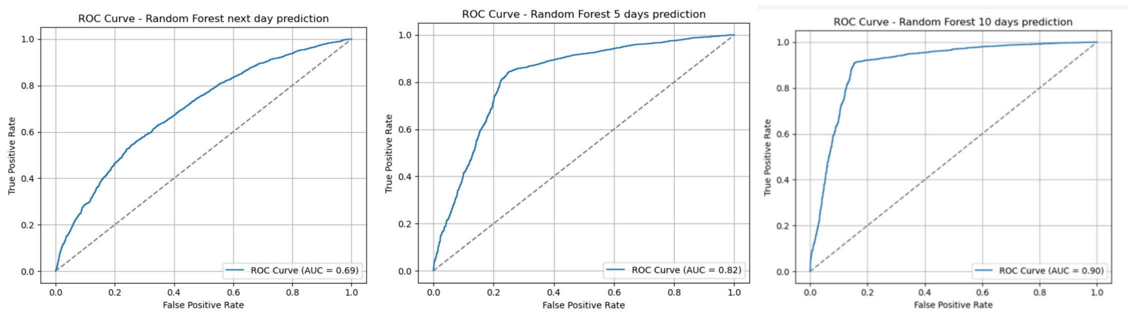

Fig 5: ROC Curves – Random Forest (1-day, 5-day, and 10-day Forecasts)

For the 5-day ahead forecast, the Random Forest model achieved an accuracy of 0.80, demonstrating strong predictive ability over a slightly extended horizon. Back-testing confirmed this consistency, with yearly accuracies ranging from 0.76 to 0.88, suggesting the model remains robust across different market conditions. In terms of classification performance, the model delivered a precision of 0.8476, meaning that when it predicted an upward movement, it was correct approximately 84.76% of the time. The recall of 0.8389 indicates that it successfully identified 83.89% of all actual upward trends, while the specificity of 0.7525 shows it correctly classified 75.25% of all downward trends. This balance suggests that while the model is slightly stronger in detecting upward price movements, it remains reliable in flagging downward ones as well. The F1 score of 0.8432 further confirms that the model maintains a healthy trade-off between precision and recall. The corresponding ROC curve (Figure 5) produced an AUC of 0.82, indicating a strong ability to discriminate between rising and falling price movements for medium-term predictions.

For the 10-day ahead prediction, the Random Forest model achieved a high accuracy of 0.88. Back-testing across multiple years confirmed this robustness, with yearly accuracies ranging from 0.86 to 0.92, underscoring its consistent performance across diverse market conditions. The classification metrics reinforce this strength: a precision of 0.9132 means upward trend predictions were correct over 91% of the time, while a recall of 0.9080 shows the model identified more than 90% of all actual upward trends. The specificity of 0.8460 indicates that 84.60% of downward trends were correctly predicted, reflecting a model that is nearly equally strong in identifying both directions. The F1 score of 0.9106 demonstrates that the balance between precision and recall is exceptionally well-maintained. The ROC curve for this forecast horizon (Figure 5) yielded an AUC of 0.90, signifying excellent discriminative capability between positive and negative classes.

4.1.1 Hyper parameter tuning

The Random Forest model was tuned manually using nested for-loops to systematically explore different combinations of key hyperparameters. This manual tuning approach was deliberately chosen to integrate rolling back testing alongside parameter optimization, enabling a more realistic and robust assessment of model performance over time. Rather than relying solely on automated methods such as grid search or random search, the manual process allowed for greater control and deeper insight into how each parameter influenced the model’s behaviour across different historical periods.

The parameter grid used for this tuning process included variations in the number of estimators (50, 100, 200), the minimum number of samples required to split an internal node (25, 50, 100), and the maximum depth of the trees (5, 10). Each of these parameters plays a significant role in shaping the complexity, flexibility, and generalisability of the Random Forest model.

The number of estimators determines how many individual decision trees are built and averaged in the ensemble. A higher number generally improves model stability and accuracy but also increases computational cost. Maximum depth controls the longest path from the root to a leaf in each tree. Limiting tree depth helps prevent overfitting by constraining the model's ability to memorize training data. A depth that is too shallow, however, may lead to underfitting. Minimum samples split sets the threshold for how many samples are required to allow an internal node to split. Increasing this value can reduce model variance by enforcing more conservative splitting, thus improving generalisation.

Through this iterative tuning process, the best-performing configuration for the next-day prediction task was identified as 100 estimators, a minimum samples split of 100, and a maximum depth of 5. Before tuning, the model achieved an accuracy of 0.61. After applying these optimized parameters, the accuracy improved to 0.65. This enhancement underscores the importance of careful hyperparameter selection in improving predictive performance, particularly in the context of financial time series, where overfitting to short-term noise is a common risk.

For the remaining prediction horizons, 5, 10, 20, 30, and 40 days , the optimal hyperparameters obtained through tuning for the Random Forest model are summarised in Table 2.

Prediction Horizon (Days)	n_estimators	min_samples_split	max_depth	Accuracy
5	200	25	5	0.8
10	50	25	5	0.88
20	50	50	5	0.925
30	50	25	5	0.9384
40	100	50	5	0.9439

Table 2: Tuned Hyperparameters for Random Forest at Different Prediction Horizons

4.2 XGBoost Classifier

For the next-day prediction task, the XGBoost classifier outperformed the other models, achieving an accuracy of 0.673, making it the top performer among the approaches tested for short-term forecasting. To assess its robustness, a rolling back test was conducted in which the model was trained on data up to the 10th year and then tested year by year in a forward-looking manner. This back test revealed yearly accuracies ranging from 0.56 to 0.68, demonstrating that the model generalises well across different historical periods and market conditions.

From a classification perspective, the model achieved a precision of 0.7172, meaning that when it predicted an upward price movement, it was correct approximately 71.72% of the time. Its recall of 0.6784 indicates that it successfully detected 67.84% of all actual upward trends, while the specificity of 0.5251 shows that it correctly identified 52.51% of downward trends. This comparatively lower specificity suggests that the model is more prone to false positives when predicting declines. The F1 score of 0.6973 reflects a balanced trade-off between precision and recall, indicating that while the model is more effective at capturing upward movements, there is still room to improve its ability to correctly identify downward ones.

The ROC curve for this task (Figure 6) yielded an AUC of 0.64, indicating a fair level of discriminative ability better than random guessing, though with room for enhancement. Even so, the combination of consistent back testing performance and balanced classification metrics makes XGBoost a dependable choice for short-term market movement forecasting.

For the 5-day ahead prediction, the XGBoost model achieved an accuracy of 0.803, with back testing showing yearly accuracies ranging from 0.61 to 0.82, indicating moderately stable performance across different market environments. The classification metrics highlight a precision of 0.8504, meaning upward trend predictions were correct 85.04% of the time, and a recall of 0.6329, showing that 63.29% of all actual upward trends were detected. The specificity of 0.7902 reveals that 79.02% of downward trends were correctly classified, indicating better performance in detecting declines than in capturing all upward movements. The F1 score of 0.7257 suggests that the model's predictive strength leans more toward precision than recall, prioritising correctness of positive predictions over capturing every true positive. The ROC curve for this forecast horizon (Figure 6) reported an AUC of 0.78, indicating a solid level of class separability for this medium-short prediction window.

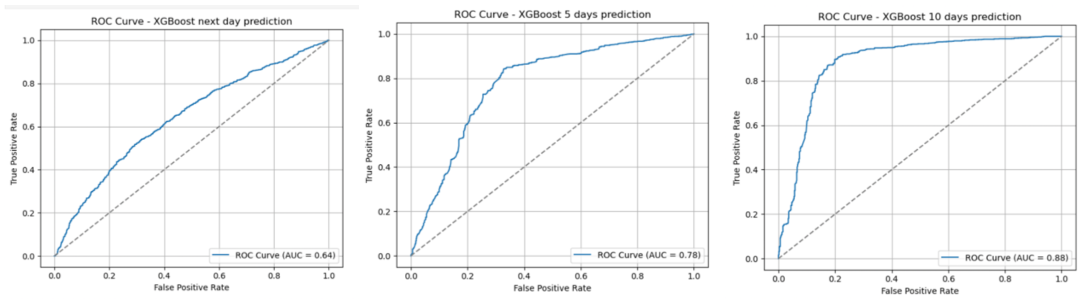

Fig 6: ROC Curves – XGBoost (1-day, 5-day, and 10-day Forecasts)

For the 10-day ahead forecast, the XGBoost model achieved a strong accuracy of 0.89, with back testing showing yearly accuracies ranging from 0.70 to 0.80. This consistency demonstrates stable generalisation across different market conditions over the medium term. The classification metrics further highlight its robustness: a precision of 0.9161 means that when the model predicted an upward price movement, it was correct 91.61% of the time. The recall of 0.8514 indicates it successfully detected 85.14% of all actual upward trends, while the specificity of 0.8328 shows it correctly identified 83.28% of downward trends. These values suggest that the model is highly reliable in capturing both rising and falling movements, with only a small gap between its performance on positive and negative classes. The F1 score of 0.8825 reflects an excellent balance between precision and recall, ensuring that high prediction accuracy does not come at the cost of missing a significant number of true cases. The ROC curve for this horizon (Figure 6) reported an AUC of 0.88, confirming the model's strong discriminative ability between positive and negative classes in medium-term forecasting.

4.2.1 Hyper Parameter tuning

For the XGBoost model, manual hyperparameter tuning was employed in order to identify the optimal parameter combinations while simultaneously incorporating back testing into the evaluation process. This approach ensured that the selected parameters not only yielded strong performance on a single test set but also demonstrated consistency across multiple time periods. The parameters were tuned over a predefined grid, which included variations in the number of estimators (100, 250, 350), maximum tree depth (4, 5, 8), learning rate (0.01, 0.1, 0.3), subsample ratio (0.6, 0.8), column sampling by tree (0.6, 0.8, 1.0), and gamma value (5).

Each of these hyperparameters plays a critical role in shaping the model's performance. The number of estimators determines how many boosting rounds are run, while the maximum depth controls the complexity of each individual tree. The learning rate adjusts how much each new tree contributes to the overall model, effectively slowing down learning to prevent overfitting. The subsample parameter defines the fraction of training samples used for each boosting round, adding an element of randomness that helps with generalisation. Similarly, colsample_bytree specifies the proportion of features considered when building each tree. Lastly, the gamma value sets a minimum loss reduction threshold required for further partitioning, helping to regulate tree growth and avoid unnecessary splits.

For the next-day prediction task, the best-performing combination was found to be 100 estimators, a maximum depth of 4, a learning rate of 0.01, a subsample ratio of 0.6, full feature sampling (colsample_bytree of 1.0), and a gamma value of 5. This configuration led to a noticeable improvement in performance, increasing the model's accuracy from 0.61 before tuning to 0.673 after tuning.

For the remaining prediction horizons, 5, 10, 20, 30, and 40 days , the optimal hyperparameters obtained through tuning for the XGBoost model are summarised in Table 3

Prediction Horizon	n_estimators	max_depth	learning_rate	subsample	colsample_bytree	gamma
5 Days	100	4	0.01	0.6	0.8	5
10 Days	250	4	0.01	0.6	0.8	5
20 Days	350	5	0.01	0.8	1	5
30 Days	250	8	0.01	0.8	0.8	5
40 Days	100	5	0.01	0.6	1	5

Table 3: Tuned Hyperparameters for XGBoost at Different Prediction Horizons

4.3 LSTM

The LSTM model was developed using manually tuned hyperparameters, selected through multiple rounds of experimentation aimed at improving performance across different prediction horizons. This approach allowed for greater flexibility in identifying optimal configurations that align with the sequential nature of financial time series data.

For the next-day prediction task, the LSTM model achieved an accuracy of 0.6122, reflecting moderate predictive capability in short-horizon forecasting. The precision of 0.6878 means that when the model predicted an upward price movement, it was correct 68.78% of the time, while the recall of 0.6840 indicates it successfully identified 68.40% of all actual upward movements. The specificity of 0.4956 shows that it correctly detected 49.56% of downward movements, revealing a tendency toward false positives when predicting declines. The F1 score of 0.6859, which balances precision and recall, suggests the model maintained a reasonable trade-off between detecting upward trends and avoiding incorrect predictions. The ROC curve for this task (Figure 7) produced an AUC of 0.60, representing moderate discriminative power better than random guessing, but with clear room for improvement.

At the 5-day ahead prediction horizon, the model's performance improved noticeably, achieving an accuracy of 0.7206. The precision of 0.7992 means 79.92% of predicted upward trends were correct, while the recall of 0.7503 shows that 75.03% of actual upward movements were captured. The specificity of 0.6682 indicates better reliability in identifying downward movements compared to the one-day horizon. The F1 score of 0.7740 reflects a stronger balance between sensitivity and precision, with the model performing more evenly across both classes. The ROC curve for this horizon (Figure 7) yielded an AUC of 0.74, pointing to a moderate-to-good level of class separation and a clear improvement in discriminative ability over the one-day forecast.

At the 10-day prediction horizon, the LSTM model delivered its strongest results. The accuracy rose to 0.8476, supported by a precision of 0.8996, meaning nearly 90% of predicted upward movements were correct, and a recall of 0.8675, indicating that over 86% of all actual upward trends were detected. The specificity of 0.8081 further shows that more than 80% of downward movements were correctly identified, demonstrating balanced predictive strength across both classes. The F1 score of 0.8833 confirms an excellent trade-off between precision and recall. The ROC curve for this forecast horizon (Figure 7) reported an AUC of 0.88, highlighting a high level of discriminative capability and marking a significant improvement compared to shorter horizons.

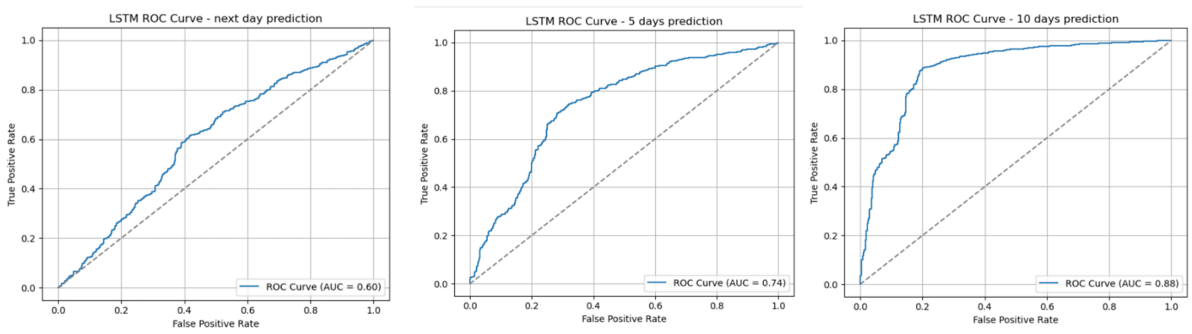

Fig 7: ROC Curves – LSTM (1-day, 5-day, and 10-day Forecasts)

4.3.1 Hyper parameter tuning

Hyperparameter tuning for the LSTM model was conducted manually through a series of controlled experiments, where different combinations of network configurations and training settings were evaluated based on their predictive performance. This process aimed to identify a model architecture that was not only effective in capturing sequential dependencies but also resilient to overfitting and generalised well across different prediction horizons.

After extensive experimentation, the final model architecture that consistently delivered the best performance across all forecasting windows was selected. The LSTM model consisted of a single Long Short-Term Memory (LSTM) layer with 32 units, followed by a Dropout layer with a dropout rate of 0.3, and a final Dense output layer with a sigmoid activation function. Both the LSTM and Dense layers incorporated L2 regularisation with a penalty coefficient of 0.001 to discourage overly complex weight configurations and promote generalisation. The model was trained using the Adam optimizer with a learning rate of 0.001, a value that provided a stable balance between convergence speed and training stability.

Each component of this configuration was carefully chosen for its role in enhancing model performance. The LSTM layer is central to the architecture, as it is designed specifically to handle time-dependent data by maintaining memory across sequences—making it particularly suitable for financial forecasting, where temporal context is essential. The 32 units were found to be sufficient to learn meaningful patterns in the data without introducing excessive computational complexity.

The Dropout layer serves as a regularisation technique, randomly deactivating a fraction of neurons during each training step. This helps prevent the model from becoming overly reliant on specific pathways and reduces the risk of overfitting, especially important in financial data where noise and variance are high. A dropout rate of 0.3 was effective in maintaining a good trade-off between regularisation and learning capacity.

The Dense layer with a sigmoid activation function was used as the output layer to produce probability estimates for binary classification. Applying L2 regularisation to both the LSTM and Dense layers helped penalise large weight magnitudes, further supporting model robustness by smoothing decision boundaries.

Finally, the Adam optimizer was chosen for its adaptive learning capabilities and computational efficiency, making it particularly well-suited for training deep learning models on noisy and high-dimensional datasets. A learning rate of 0.001 ensured gradual convergence, avoiding the instability that can arise from overly aggressive updates.

This configuration consistently yielded strong performance across the 1-day, 5-day, and 10-day prediction horizons. Its effectiveness was validated by the classification metrics and ROC-AUC scores reported in the results section, highlighting its capacity to generalise well across varying forecast windows. The careful manual selection of these hyperparameters played a crucial role in the model's ability to learn complex patterns while maintaining resilience against overfitting—a common challenge in financial time series prediction.

5. Discussion and Conclusion

This study set out to investigate the performance of machine learning models specifically Random Forest, XGBoost, and Long Short-Term Memory (LSTM) in predicting the directional movement of the S&P 500 index over multiple forecast horizons. The modelling approach involved constructing a robust pipeline for data preparation, feature engineering, manual hyperparameter tuning, model training, and evaluation using both performance metrics and ROC curves.

One of the central findings of this research is that Random Forest and XGBoost models consistently outperformed the LSTM model across most forecast horizons. While LSTM demonstrated reasonably good accuracy, especially for 10-day forecasts (with accuracy reaching 0.85), it generally lagged behind the tree-based models in overall predictive performance and class separability. The likely explanation lies in the nature of the dataset and the way sequential dependencies are encoded. Although LSTM is architecturally suited for sequential data, the engineered features used in this study may have been more amenable to ensemble-based models that can better capture non-linear patterns without requiring time-sequenced input.

Random Forest, in particular, benefited from a manual hyperparameter tuning approach, which enabled simultaneous evaluation across different historical periods through rolling back testing. This allowed the model's robustness to be verified under varying market conditions. The model demonstrated stable performance, with back tested accuracy ranging from 0.608 to 0.712 for next-day predictions, and increasing further as the forecast horizon extended to 10 days. For the 10-day prediction task, the model achieved its best balance between accuracy (0.88), precision (0.91), recall (0.90), and specificity (0.85), underscoring its practical applicability in short- to medium-term trading strategies.

The XGBoost classifier, while similarly tuned manually, exhibited the highest predictive performance overall, particularly at the 40-day mark where it achieved an accuracy of 0.9696. Its ability to handle imbalanced features and learn complex interactions gave it an edge in capturing subtle patterns within the dataset. Interestingly, the performance of both Random Forest and XGBoost showed a trend of increasing accuracy up to approximately 10 days, after which the marginal improvement tapered off. This observation aligns with the notion that short-term market movements are often more predictable due to momentum effects and recent trends, whereas longer-term predictions encounter increased noise and external variability.

Feature engineering also played a critical role in model performance. A wide range of technical indicators such as RSI, MACD, OBV, and others were experimented with and selected based on empirical performance through multiple trials. This iterative and data-driven process ensured that only the most relevant predictors were included in the final models, enhancing interpretability and predictive power.

Another important contribution of this study is the rigorous evaluation framework employed. Rather than relying solely on accuracy, the study incorporated precision, recall, specificity, and F1 scores to provide a multi-dimensional understanding of model behavior. Furthermore, the use of ROC curves and AUC values helped visualize the models' discriminative capabilities at various threshold levels. For instance, the Random Forest model's AUC of 0.90 for 10-day predictions and the XGBoost model's AUC of 0.93 for 40-day predictions underline their reliability in classifying market direction with a high degree of confidence.

While the LSTM model did not outperform the tree-based models, its results were still promising, particularly as the forecast horizon increased. The performance of LSTM improved significantly from 0.6122 (next-day prediction) to 0.8476 (10-day prediction), suggesting that its ability to learn temporal dependencies may require a longer input-output sequence to be fully effective. The model architecture, tuned manually with 32 LSTM units, a dropout rate of 0.3, and L2 regularization, prioritized generalization and robustness. Future work could explore more advanced LSTM architectures or hybrid models that integrate tree-based predictors with deep learning for enhanced performance.

In conclusion, this study demonstrates that ensemble learning models such as Random Forest and XGBoost are highly effective for short- to medium-term stock index direction prediction, especially when supported by thoughtful feature engineering and thorough tuning. The findings also highlight the importance of tailoring evaluation methods to the characteristics of time series data, as well as the value of back testing to gauge real-world applicability. While no model guarantees perfect prediction particularly in volatile and non-stationary financial markets, the strategies developed here provide a strong foundation for further exploration in both academic and practical investment contexts.

6. Future Work

While this study demonstrates promising results in predicting S&P 500 stock index movements using machine learning models such as Random Forest, XGBoost, and LSTM, several avenues remain open for future research and model enhancement.

One key area of focus could be the further optimization of the LSTM model. Although LSTM showed competitive performance at certain prediction horizons, there is significant room for improvement through more rigorous hyperparameter tuning, feature engineering, and architectural experimentation. Exploring deeper LSTM networks, integrating bidirectional or attention-based LSTM variants, or introducing more regularization techniques could potentially improve temporal pattern recognition and generalization.

Another promising direction is to implement model stacking or ensemble methods, combining the strengths of different models such as Random Forest, XGBoost, and LSTM into a meta-learner that could capture diverse patterns in the data more effectively. Such hybrid approaches might enhance both predictive performance and robustness across market regimes.

In addition, incorporating alternative feature sets beyond technical indicators and historical price data may further enrich model learning. One particularly relevant domain is sentiment analysis using financial news, earnings reports, or even social media feeds. Natural Language Processing (NLP) techniques could be used to extract sentiment signals and merge them with quantitative indicators, potentially improving forecasting power in volatile or event-driven markets.

Advancements in deep learning also offer exciting opportunities. Future studies could explore state-of-the-art architectures like Transformers or Temporal Fusion Transformers (TFT), which have recently shown promise in time-series forecasting. These models can better handle long-range dependencies, attention mechanisms, and multi-modal inputs, making them suitable candidates for financial time series prediction.

Finally, expanding the study to other indices or individual stocks, or performing cross-market validation, would help assess the generalizability of the findings. Likewise, experimenting with multi-step forecasting frameworks or probabilistic prediction models (such as Bayesian deep learning) can provide not just point predictions but also uncertainty estimates, which are particularly valuable in financial decision-making contexts.

In summary, future work could enhance both the technical sophistication and real-world applicability of the models explored, contributing to a deeper understanding of stock market dynamics and advancing the state of predictive financial modeling.

7. References

1. Gençay, R. (1999). Linear, non-linear and essential foreign exchange rate prediction with simple technical trading rules. *Journal of International Economics*, 47(1), 91–107.
2. Timmermann, A., & Granger, C. W. J. (2004). Efficient market hypothesis and forecasting. *International Journal of Forecasting*, 20(1), 15–27.
3. Bao, D., & Yang, Z. (2008). Intelligent stock trading system by turning point confirming and probabilistic reasoning. *Expert Systems with Applications*, 34(1), 620–627.
4. Malkiel, B. G., & Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*, 25(2), 383–417.
5. Malkiel, B. G. (2003). The efficient market hypothesis and its critics. *The Journal of Economic Perspectives*, 17(1), 59–82.
6. Li, H., Yang, Z., & Li, T. (2014). Algorithmic trading strategy based on massive data mining. *Stanford University*.
7. Dai, Y., & Zhang, Y. (2013). Machine learning in stock price trend forecasting. *Stanford University*.
8. Xinjie. (2014). Stock trend prediction with technical indicators using SVM. *Stanford University*.
9. Geurts, P., & Louppe, G. (2011). Learning to rank with extremely randomized trees. *JMLR: Workshop and Conference Proceedings*, 14, 49–61.
10. Devi, K. N., Bhaskaran, V. M., & Kumar, G. P. (2015). Cuckoo optimized SVM for stock market prediction. In *IEEE Sponsored 2nd International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*.
11. Khaidem, L., Saha, S., & Dey, S. R. (n.d.). *Predicting the direction of stock market prices using random forest*.
12. Varadharajan, V., Smith, N., Kalla, D., Samaah, F., Polimetla, K., & Kumar, G. R. (2024). *Stock Closing Price and Trend Prediction with LSTM-RNN*. Journal of Artificial Intelligence and Big Data.
13. Zhenglin, L., Xie, Z., & Zhang, R. (2023). *Stock Market Analysis and Prediction Using LSTM: A Case Study on Technology Stocks*. Innovations in Applied Engineering and Technology, vol. 2, no. 1.
14. Sang, S. & Li, L. (2024). *Attention Mechanism Variant LSTM for Stock Price Prediction*. Mathematics, vol. 12, no. 7.
15. Shi, Z., Hu, Y., Mo, G., & Wu, J. (2022). *Attention-based CNN–LSTM and XGBoost Hybrid Model for Stock Prediction*.
16. Mehtab, S., Sen, J., & Dutta, A. (2020). *Stock Price Prediction Using Machine Learning and LSTM-Based Deep Learning Models*.
17. Huang, M. (2024). *Effectiveness Validation of LSTM for Stock Prices Prediction on Four Stocks*.