*A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science (MSc) in*:

**Data Science**

**UNIVERSITY COLLEGE LONDON**

---

# Comparative Analysis of Deep Learning and Statistical Models For Stock Market Prediction

---


*Author:*

Candidate Number: GSJP1

*Supervisor:* GIAMPIERO MARRA

Department of Statistical Sciences

UNIVERSITY COLLEGE LONDON



Date of submission: August 29, 2024

Word count: 14,130

# DECLARATION

I, PAVITER SINGH REHAL confirm that the work presented in this report is my own. Where information has been derived from other sources, I confirm that this has been indicated in the report.

PAVITER SINGH REHAL

# ABSTRACT

Accurate stock price prediction is crucial in the volatile stock market, where returns and risks can fluctuate widely. Financial institutions and regulatory authorities focus on this area due to its potential economic benefits. Stocks consistently attract investors because of their high returns, driving ongoing research into stock price forecasting.

Initially, economists used simple mathematical models for forecasting. Advances in mathematical theory and computer technology led to the development of sophisticated time series models like ARIMA and GARCH, which effectively model linear dependencies and volatility clustering in stock prices.

However, the non-linearity and non-stationarity of stock data have necessitated the adoption of machine learning methods such as support vector machines (SVM) and deep learning techniques like recurrent neural networks (RNN) and long short-term memory (LSTM) networks. These models handle non-linear data and retain useful information over long sequences, making them particularly suitable for stock price forecasting.

The results of this study indicate that while ARIMA and GARCH models capture linear trends and volatility, LSTM models significantly outperform them in predicting non-linear patterns and long-term dependencies. The hybrid models combining statistical approaches (like GARCH) with deep learning techniques (LSTM and GRU) demonstrated improved forecasting accuracy by capturing both linear and non-linear characteristics of stock market data. Notably, the binary prediction model, which integrated LSTM/GRU with GARCH for residual volatility adjustments, achieved the highest accuracy in predicting directional movements of stock prices, making it the most effective tool for practical trading strategies.

Overall, this thesis contributes to the field of financial forecasting by highlighting the advantages and limitations of ARIMA, GARCH, Copula, and LSTM models, both individually and in hybrid configurations, for stock market price prediction. The insights gained from this research could assist investors and financial analysts in making more informed decisions and developing more robust investment strategies. This study's comprehensive approach to model evaluation and the successful integration of machine learning and statistical methods address key gaps in financial forecasting research, paving the way for future advancements in the field.

# CONTENTS

# 1. INTRODUCTION

## 1.1. BACKGROUND

Predicting stock market prices has long been a keen interest for economists, analysts, and investors due to its significant impact on investment decisions, portfolio management, and risk mitigation strategies. The market's inherent complexity and volatility, influenced by economic indicators, market sentiment, geopolitical events, and stock-related news, pose unique challenges to accurate forecasting [1, 2].

Historically, stock price prediction relied on simple mathematical models like the Random Walk Theory, which suggested that prices follow a random path and are inherently unpredictable. The development of more sophisticated time series models, such as ARIMA by Box and Jenkins [3], marked a significant advancement by capturing linear dependencies effectively. GARCH models, introduced by Bollerslev [4], further improved predictions by modeling volatility clustering, where periods of high volatility tend to be grouped together.

The limitations of traditional models in handling non-linearity and non-stationarity in stock data led to the adoption of machine learning techniques. Support Vector Machines (SVM) introduced by Cortes and Vapnik[5] and deep learning models like Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) networks developed by Hochreiter and Schmidhuber [6] have been introduced to capture complex patterns and long-term dependencies. LSTMs, in particular, address the vanishing gradient problem in RNNs, making them highly suitable for time series forecasting in stock markets.

By integrating traditional and modern approaches, researchers aim to develop more accurate and robust models for stock market prediction.

## 1.2. AIM OF THIS THESIS

The Thesis will explore the direct application of ARIMA, GARCH, Copula and LSTM models in predicting stock market prices. There will also be an incentive to not only predict the closing price of candles but also a binary output of up/down. This should help to reduce the complexity of the models and potentially reduce any overfitting. There will be a comprehensive comparative analysis conducted to evaluate performances. Historical data from S&P 500 index will be used as the basis for all analysis and comparison. The S&P 500 is widely regarded as a good benchmark as it represents 500 large-cap companies listed on the US stock exchange hence, making it ideal for this study.

### 1.2.1. ARIMA, GARCH and Copula Application

The ARIMA model is the simplest of the three models to be investigated and analysed. It has the capability to capture linear dependencies in the time series data we will be working with. It will be very effective for short-term forecasting and provide a robust benchmark for modelling the time series data with autocorrelations. However, the lack of accurate prediction lies within the model assumptions, the ARIMA model assumes the time series data to be linear and stationary, which is very often not the case with the given financial data. [3].

GARCH models are an effective extension of the ARIMA framework, they model the volatility and heteroskedasticity. Hence, they are very useful in capturing the time-varying volatility that's typically observed with this type of data. GARCH models have been widely used in financial econometrics for volatility forecasting and risk management. [4]. The use of ARIMA and GARCH set a very good benchmark in capturing the majority of the linear trends and variance of the data however, knowing that most financial data consists of non-linear non-stationary data we will go ahead to explore other models.

Copula models capture the dependencies between multiple variables, extending beyond simple linear correlations. They are particularly effective in finance for modeling joint distributions of asset returns, allowing for a more detailed understanding of market behavior. Copula models are essential for risk management and portfolio optimisation by linking marginal distributions to their multivariate distribution. These models are widely used to capture non-linear dependencies and tail risks, providing significant advantages over traditional methods.

### 1.2.2. LSTM and Hybrid Models

LSTM networks will be leveraged in the prediction of stock market prices over RNNs due to their ability to address the vanishing gradient problem. they ensure that the gradients can propagate through many time steps without diminishing. This is primarily achieved through cell state $C_t$ and the forget gate $f_t$, which allows the gradients to flow through time steps hence, preserving long-term dependencies. [6, 7]. This leads to LSTMs being more capable of capturing complex temporal dynamics which can be missed by other more simplistic models.

Hybrid models can be seen as the best chance at accurately predicting the stock market as they can capture both the linear and non-linear trends. We can use a combination of ARIMA and LSTM to do this as demonstrated by Zhang [8].} However, this technique can be prone to overfitting hence I will introduce the methodology to predict the binary movement (up/down) from one candle closing to the next candle closing, inherently reducing model complexity, potentially improving predictability and improving interpretability.

### 1.2.3. Contribution and Structure of The Study

This study contributes to the field of financial forecasting by providing a detailed comparative analysis of ARIMA, GARCH, Copula and LSTM models for stock market prediction. It will highlight the advantages and limitations of each model in predicting close prices. of candles. As well as developing and expanding on an under-researched

area of prediction that uses binary prediction rather than direct complex prediction of close prices. This area will mostly be explored in the hybrid configurations given they have the most accurate capturing of the data. The findings of the study could help in developing more robust investment strategies and improving decision-making processes in the financial industry.

The remainder of this dissertation will be structured as follows: Chapter 2 Reviewing the current literature surrounding stock market price prediction using traditional and more advanced deep learning methods. Chapter 3 describes the methodology used in this study, detailing the data collection process, model implementation/methodology, and evaluation metrics. Chapter 4 presents the empirical results, comparing the performance of ARIMA, GARCH, Copula LSTM, and any Hybrid models. Chapter 5 discusses the implications of the results and findings, Chapter 6 concludes the study with a summary of findings, limitations, and any suggestions for future work.

Overall, this dissertation aims to advance the understanding of stock price prediction by rigorously evaluating and comparing various models, thereby contributing to the ongoing efforts to improve financial forecasting methodologies.

# 2. LITERATURE REVIEW

## 2.1. OVERVIEW

Predicting Stock market prices is a complex and multifaceted challenge that has attracted significant attention from researchers and practitioners due to its potential for financial gains and risk management. This literature review delves into the various methodologies explored for stock market prediction, focusing mainly on ARIMA, GARCH, LSTM, and different hybrid models. The following will explore an in-depth analysis of major contributions to the field and where current research in the field lies.

## 2.2. TRADITIONAL STATISTICAL MODELS

### 2.2.1. ARIMA

The ARIMA model is a foundational statistical technique used for time series analysis. ARIMA models have the capability to predict future points in a series by capturing dependencies between observations in a dataset. They are characterised by three parameters: $p$ (autoregressive order), $d$ (degree of differencing), and $q$ (moving average order), this makes ARIMA models very effective in capturing linear relationships in stationary data, making them a popular and simple choice for time series forecasting in various different fields including financial forecasting.

Virtanen and Yiolli [9] applied ARIMA models to forecast the Finnish stock market index, incorporating six explanatory variables, such as lagging indices and macroeconomic factors. Their study utilised the Box-Jenkins methodology [3], which is a systematic approach to identifying, fitting, and checking ARIMA models. This included rigorous Augmented Dickey-Fuller (ADF) [10] tests to ensure stationarity and the use of Akaike Information Criterion (AIC) [11], Bayesian Information Criterion (BIC) [12] for model selection. In the above-mentioned research, Virtanen and Yliolli conducted a comprehensive analysis to determine optimal parameters $(p, d, q)$ for the ARIMA model. They found that ARIMA models could effectively predict short-term price movements in the Finnish stock market. However, they also highlighted a large number of limitations, particularly the ARIMA model's reliance on linear assumptions and stationary data requirements. These limitations become a big weakness when ARIMA models are applied to non-linear and volatile stock market data, leading to a large portion of unexplained and uncaptured volatility. This study demonstrates the practical applications and limitations of the ARIMA model in financial forecasting. [9]

A similar study was done by Clark and West [13], they developed a stock price prediction system based on ARIMA, testing their approach with the New York Stock Exchange (NYSE) and Nigeria. They also used differencing to achieve stationarity and fitted the ARIMA models to predict short-term price changes. However, they primarily focused on evaluating the effectiveness of ARIMA models across different markets, each market having unique patterns and volatility. Their research revealed that ARIMA shows promise in forecasting but is largely hindered by the non-stationary and non-linear nature of stock market data. They found that the markets are influenced by so many unpredictable factors, including geopolitical events and economic news as well as investor sentiment, which in nature introduce non-linearities and structural breaks that ARIMA models simply aren't equipped enough to deal with. Their findings suggest that integrating ARIMA with more sophisticated models capable of capturing complex financial data dynamics could enhance prediction accuracy. This insight has paved the way for subsequent research exploring hybrid models that combine ARIMA with machine learning techniques.

Overall, both studies highlight the strengths and limitations of ARIMA models in stock market prediction. They provide a solid foundation for understanding linear time series data but also highlight the need for more advanced techniques to address the complexities of financial markets. The work of Virtanen and Yliolli [9] and Clark and West [13] contributes significantly to the field by demonstrating the practical applications of ARIMA models and identifying areas of limitation

### 2.2.2. GARCH

The Generalised Autoregressive Conditional Heteroskedasticity (GARCH) model expands on the ARIMA framework by capturing time-varying volatility which is a common characteristic of financial time series data. GARCH models are seen to be good at capturing volatility clustering, where periods of high volatility are followed by periods of low volatility, which is often the case in financial markets.

Engel pioneered the Autoregressive Conditional Heteroskedasticity (ARCH) model [14], which later became the GARCH model by Bollerslev [4]. These models have been instrumental in financial econometrics for modelling and forecasting the volatility of stock returns. The GARCH$(1,1)$ model has become a standard due to its effectiveness and simplicity. The GARCH $(1,1)$ model is defined as follows:

$$\sigma_t^2 = \alpha_0 + \alpha_1 \epsilon_{t-1}^2 + \beta \sigma_{t-1}^2 \tag{1}$$

Where $\sigma_t^2$ is the conditional variance at time $t$, $\epsilon_{t-1}^2$ is the lagged squared residual from the mean equation, and $\sigma_{t-1}^2$ is the lagged conditional variance. The parameters $\alpha_0, \alpha_1$, and $\beta$ are estimated to fit the model to the data [14].

Engle's introduction of the ARCH model marked a significant advancement in the ability to model and forecast time-varying volatility in financial markets. By allowing the variance of the error terms to change over time, the model could more accurately reflect the reality of financial data, where volatility is not constant but tends to cluster in periods of high and low activity [14] Bollerslev, [4] expanded on this concept by introducing the GARCH

model, the GARCH model can provide a more complete representation of volatility, due to introducing lagged variances making it a preferred choice for modeling financial time series. Bollerslev's work demonstrated that GARCH models could effectively forecast future volatility based on past observations, making them invaluable tools for risk management and derivative pricing.

In summary, GARCH models have proven effective in capturing the volatility dynamics of financial data. However, their limitations in modeling non-linear dependencies have driven the exploration of deep learning techniques as well as hybrid models.

### 2.2.3. Copula

Copula models are advanced statistical tools designed to capture the complex relationships between multiple variables. They go beyond simple linear correlations, making them especially useful in finance. In financial markets, the returns on different assets often have non-linear relationships and can exhibit tail dependencies. By using copula models to predict stock market prices, analysts can build joint distributions that reflect these intricate dependencies, offering a more detailed view of market behavior than traditional linear models.

$$C(u_1, u_2, \ldots, u_d) = P(U_1 \leq u_1, U_2 \leq u_2, \ldots, U_d \leq u_d) \tag{2}$$

The formula above represents a copula $C$, which is a multivariate distribution function where each variable has a uniform distribution. Sklar's Theorem [15], which serves as the theoretical foundation for copula models, asserts that any multivariate joint distribution can be decomposed into its marginal distributions and a copula that captures their dependence structure.

Patton [16] applied copula models to understand the dependence structure between exchange rates, showing how effective they are in capturing non-linear relationships. He used both Gaussian and t-copulas, discovering that t-copulas were better because they could model tail dependencies more accurately. In his study, Patton first fitted GARCH models [4] to each exchange rate series to manage the time-varying volatility. Then, he estimated the copula parameters using maximum likelihood estimation (MLE). This method allowed for flexible modeling of joint distributions, offering deeper insights into how exchange rates move together compared to traditional methods based solely on correlation. The copula-GARCH model was particularly successful in capturing dynamic dependencies, which is essential for precise risk management and pricing of derivatives in financial markets.

In another important study, Ning [17] investigated how copula models can be used to predict stock market returns. He used several types of copulas, such as Clayton, Gumbel, and Frank, to examine the dependence between stock returns and macroeconomic variables. Ning's study used non-parametric kernel density estimation to model the marginals, allowing for the unique distributional characteristics of each variable to be captured accurately. He then back-tested the predictive performance of these copula models with historical data from the S&P 500 index. The findings showed that copula models were able to effectively capture the complex relationships between stock returns and economic indicators, resulting in better forecasting accuracy than traditional multivariate models. Nonetheless, Ning also pointed out the difficulties in selecting the right type of copula and the high

computational demands of the estimation process.

Creal et al. [18] developed dynamic copula models to account for the changing dependence structure between financial returns over time. They introduced a generalised autoregressive score (GAS) model, which allows the copula parameters to evolve, making the model more responsive to shifting market conditions. The researchers applied this dynamic copula approach to international equity indices, demonstrating its superior ability to capture evolving dependencies and enhance out-of-sample forecast accuracy. This dynamic model was particularly effective during periods of financial instability, where static copulas struggle to capture the rapid changes in dependencies. By incorporating time-varying parameters, the dynamic copula model proved to be a powerful tool for dynamic risk management and portfolio optimisation.

In summary, copula models offer a strong framework for understanding the complex relationships in financial markets. The research by Patton [16], Ning [17], and Creal et al. [18] illustrates the practical applications of these models in predicting stock market behavior. Despite challenges such as high computational demands and the necessity for careful model selection, copula models excel at capturing non-linear dependencies and tail risks, providing significant advantages over traditional methods. These insights contribute to more accurate financial forecasting and enhanced risk management strategies.

## 2.3. DEEP LEARNING MODELS

### 2.3.1. LSTM Networks

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN), they are designed to capture long-term dependencies in sequential data. LSTMs address the vanishing gradient problem common in RNNs hence, making them more suitable for time series forecasting, including stock market prediction.

Hochreiter and Schmidhuber [6] first introduced LSTM networks in 1997, the network incorporates memory cells capable of maintaining information over long periods. This includes forget, input and output gates that regulate the flow of information hence, allowing the LSTM networks to retain and utilise relevant information from the past. The cell's mathematical construction is as follows:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{4}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{5}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{6}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{7}$$

$$h_t = o_t \odot \tanh(C_t) \tag{8}$$

where $f_t, i_t, C_t$ and $o_t$ are the forget, input, cell state, and output gates respectively. [6].

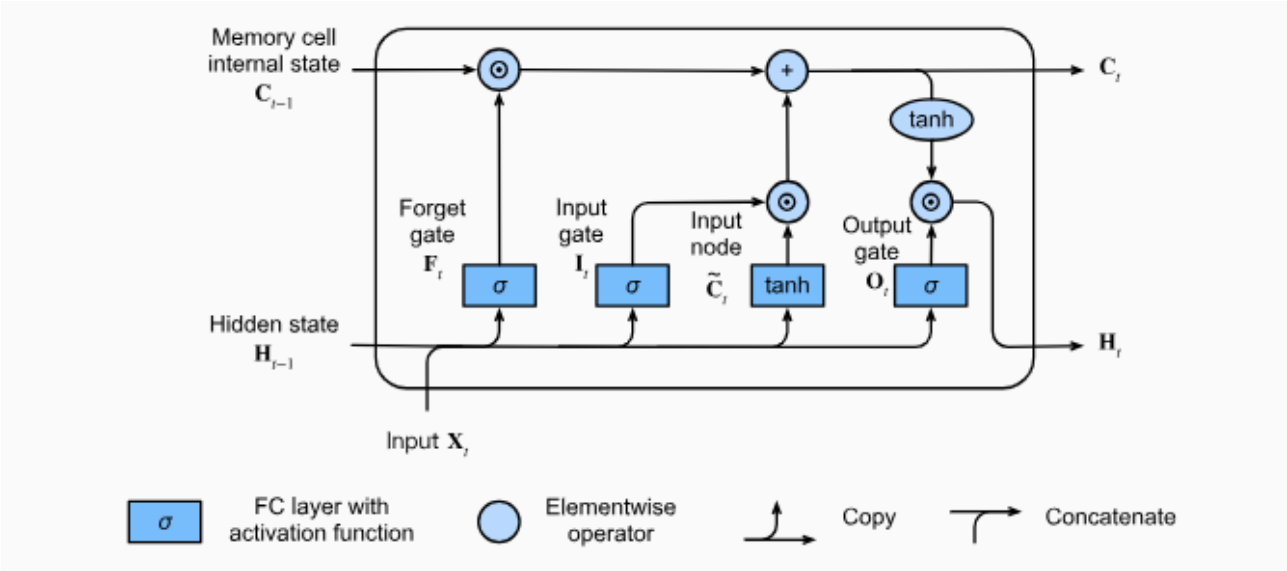


**Figure 1:** LSTM Cell Architecture
[19]

The Above is a visual representation of Hochreiter and Schmidhuber's development of LSTM Cells. These developments were a very significant advancement in the ability to model long-term dependencies in time series data. The inclusion of gates that control the flow of information, LSTMs can effectively manage gradients over long sequences, which is very important in stock market prediction where historical data can provide valuable context for any future price movements.

There was further research done on the application of LSTMs in predicting stock market data by Dai et al [20]. This study utilised LSTM networks to predict stock market prices by integrating textual data from news sources with historical stock prices. The study proves that LSTMs significantly improved prediction accuracy compared to traditional models. By combining sequential data with unstructured data (stock prices and news sentiment respectively), the model was able to capture both temporal dependencies and external factors influencing the stock market. This hybrid method allowed the model to incorporate external factors that can influence market dynamics, thereby improving the robustness and accuracy of the predictions. Their findings underscore the potential of LSTM networks to handle complex, multi-modal data sources in financial forecasting.

Another significant study is by Fischer and Krauss [21], who employed LSTM networks to predict stock returns for the S&P 500 index. The study aimed to explore the effectiveness of LSTMs in capturing temporal dependencies and patterns in stock returns data. The authors used daily stock returns of S&P 500 constituents over a period of 20 years, creating a robust dataset for training and validation. They used a rolling window approach for training and testing the LSTM models, ensuring that the models were evaluated on out-of-sample data to mimic real-world trading conditions.

The dropout mechanism is defined as follows:

$$h_t{}^{dropout} = h_t \odot Bernoulli(p) \tag{9}$$

where $\odot$ denotes the element-wise multiplication and $p$ is the dropout probability, helping to regularise the network and preventing overfitting by randomly dropping units during training. The study's results indicated that the LSTM model significantly outperformed traditional benchmark models, such as logistic regression and random forests, in terms of prediction accuracy and Sharpe ratio [21]. The Sharpe ratio is calculated as follows:

$$SharpeRatio = \frac{E[R_i - R_f]}{\sigma_i} \tag{10}$$

where $E[R_i]$ is the expected return on investment, $R_f$ is the risk-free rate, and $\sigma_i$ is the standard deviation of the investment's excess return. The LSTM model used by Fischer and Krauss was able to learn complex temporal dependencies in the stock return series, capturing both short-term momentum and longer-term reversal effects. The study shows the potential of LSTM networks in financial applications, particularly in capturing temporal patterns.

Overall, LSTM networks have shown superior performance in stock market prediction due to their ability to model complex, non-linear relationships and long-term dependencies. However, their implementation requires careful tuning of hyperparameters such as learning rate, batch size, and the number of hidden units. Additionally, training LSTM models demands substantial computational resources, particularly when dealing with large datasets typical of financial markets.

### 2.3.2. CNN Networks

Convolutional Neural Networks (CNNs) are traditionally used for image-processing tasks due to their ability to capture spatial hierarchies and extract features from data. However, their capability to identify and learn complex patterns has made them applicable to time series forecasting, including stock market prediction.

Chen et al innovated an approach to sing CNNs for stock price prediction. The study aimed to leverage the feature extraction characteristic of CNNs by transforming historical stock price data into image-like representations. This allowed the CNN to detect patterns and trends that might be missed by other models like LSTM. [22]. The historical stock prices were transformed into 2D image formats where each day's rice data, open, high, low, close and volume were treated as pixels in a larger image. This allowed us to capture the temporal relationships and trends in the data, converting a time series problem into an image recognition problem.

Chen et al used the following architecture:

1. Convolutional Layers - These were used to extract feature maps with the operation defined as follows

$$(I * K)(x, y) = \sum_m \sum_n I(x - m, y - n)K(m, n) \tag{11}$$

where $I$ is the input image, $K$ is the kernel, and $(x, y)$ are the coordinates of the pixel. The kernel slides over the input image, while computing the dot product between the kernel and the portion of the image it covers hence, creating a feature map that highlights the specific patterns.

2. Activation Functions - proceeding each convolutional operation, there was an activation function such as ReLU (Rectified Linear Unit), this introduced non-linearity into the model and is defined as:

$$ReLU(x) = max \tag{12}$$

This function trains the network by allocating only positive values, in turn removing any negative values.

3. Pooling Layers - used to downsample the feature maps, reducing dimensionality while still retaining the most important features:

$$P(x, y) = \max_{(i,j) \in R(x,y)} I(i, j) \tag{13}$$

where $R(x, y)$ represents the region of input covered by the pooling operation and $P(x, y)$ is the pooled feature map.

4. Fully Connected Layers - This is the final layer used by Chen et al, it takes the flattened pooled feature maps as input and makes the final prediction, computing the following:

$$y = W \cdot x + b \tag{14}$$

where $W$ represents the weights, $x$ is the input and $b$ is the bias term. This combines the extracted features to make a final prediction about the stock market prices [22]

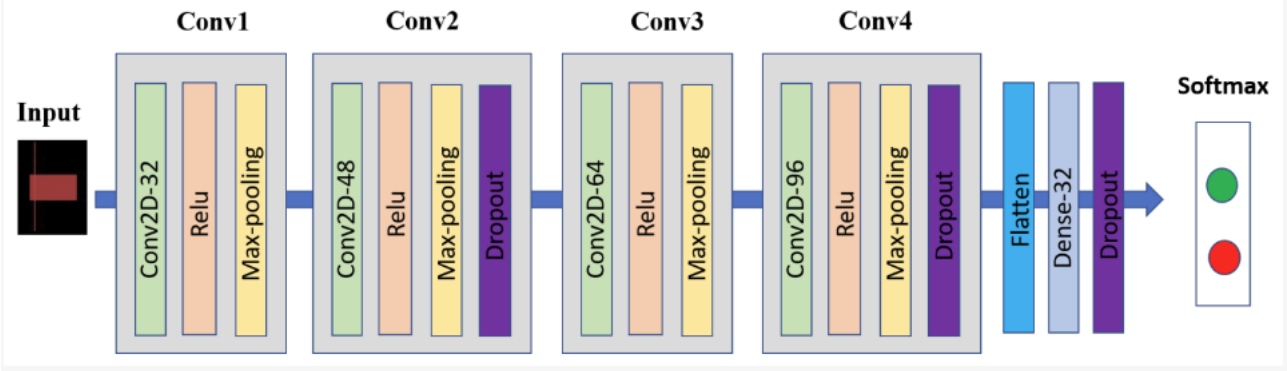


**Figure 2:** 2D CNN Architecture for Candlestick Data [23]

To evaluate the performance, Chen et al. compared the CNN model's predictions against those of traditional models like ARIMA and basic feedforward neural networks. The results indicated that CNNs were particularly effective in capturing short-term trends and patterns in the stock data, outperforming the traditional models. it was determined that CNNs could significantly improve prediction accuracy by effectively identifying complex patterns in stock market data. The approach of transforming time series data into an image format allowed

the CNN to leverage its powerful feature extraction capabilities, capturing nuances in the data that traditional methods might miss. [22]

## 2.4. HYBRID MODELS

Hybrid models in stock market prediction combine the strengths of different modeling techniques to enhance prediction accuracy and robustness. By leveraging both statistical and deep learning methods, these models aim to capture linear, non-linear, and temporal dependencies in stock market data.

### 2.4.1. ARIMA-LSTM Models

ARIMA-LSTM models integrate the ARIMA model with LSTM networks. The ARIMA model handles linear patterns and ensures data stationarity, while the LSTM network captures non-linear dependencies and long-term memory. Zhang et al. conducted an extensive study on the application of ARIMA-LSTM hybrid models for financial time series forecasting. Their methodology involved a two-step process where the ARIMA model was first applied to the time series data to handle linear components, followed by the application of LSTM to model the non-linear residuals [24]. First, we had the application of the ARIMA model to the time series data, using

the following:

$$ARIMA(p,d,q) = \phi(B)\Delta^d Y_t = \theta(B)\epsilon_t \tag{15}$$

where $\phi(B)$ is the autoregressive polynomial, $\Delta^d$ is the differencing, $Y_t$ is the observed time series, $\theta(B)$ is the moving average polynomial and $\epsilon_t$ the error term. Now we need to extract the residuals ($\hat{\epsilon}_t$) where

$$\hat{\epsilon}_t = Y_t - \hat{Y}_t \tag{16}$$

which represents the non-linear pattern that wasn't captured by the fitting of the ARIMA model. Now Zhang proceeded to fit the LSTM network, the extracted residual was used as an input to train the LSTM network with the cell equations based on equations; (3) - (8). The Training involved minimising the mean squared error (MSE) between the predicted residuals and actual residuals:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{\epsilon}_i - \epsilon_i)^2 \tag{17}$$

Finally, the predictions from ARIMA and LSTM were combined to produce the final prediction:

$$\hat{Y}_{t+k} = \hat{Y}_t^{ARIMA} + \hat{\epsilon}_t^{LSTM} \tag{18}$$

where $\hat{Y}_t^{ARIMA}$ is the ARIMA forecast and $\hat{\epsilon}_t^{LSTM}$ is the LSTM forecast of the residuals [24]. This study demonstrated that the ARIMA-LSTM hybrid model was significantly better in performance than the standalone ARIMA and LSTM models at least in terms of prediction accuracy. The hybrid model effectively captured both the linear and non-linear patterns present in financial time series data, leading to more robust and accurate forecasts.

### 2.4.2. GARCH-LSTM

Similar to the ARIMA-LSTM this hybrid model was an opportunity to combine the GARCH an LSTM, the GARCH has a greater ability to capture the volatility clustering which could improve performance significantly.

Kristjanpoller et al. conducted a pioneering study exploring the combination of GARCH and LSTM models for predicting the volatility of the Latin American stock market. The methodology involved fitting the GARCH model to the time series data to capture volatility clustering and then using the residuals from the GARCH model as inputs to train the LSTM network. The model used initially was the GARCH(1,1) model with equation (1) being the mathematical format of this model.
After fitting this model a similar process was done and the residuals were extracted as follows;

$$\hat{\epsilon}_t = \frac{Y_t - \mu_t}{\sigma_t} \tag{19}$$

where $Y_t$ is the observed return at time $t$, $\mu_t$ the conditional mean and $\sigma_t$ the conditional standard deviation from the GARCH model. As the GARCH models main purpose was to capture the volatility, Krisjanpoller et al proceeded to apply the LSTM network to the residuals which represented the non-linear data [25]. The study did so by applying equations (3) - (8) to the standardised residuals in turn, training the LSTM network on the residuals helping to capture the non-linear dependencies. The loss function was defined by equation 17 with $\hat{\epsilon}_i$ being the predicted residuals from the LSTM network and $\epsilon_i$ the actual residuals from the GARCH model. Finally, the forecasts were combined and using the GARCH models conditional variance and the LSTM networks predictions given by:

$$\hat{\sigma}^2_{t+k} = \hat{\sigma}^{GARCH}_t + \hat{\epsilon}^{LSTM}_t \tag{20}$$

where $\hat{\sigma}^{GARCH}_t$ is the conditional variance forecast from the GARCH model and $\hat{\epsilon}^{LSTM}_t$ is the LSTM model residuals forecast. The GARCH-LSTM model proved to be robust across different market conditions, highlighting its flexibility in capturing both linear volatility structures and non-linear dependencies. [25]

### 2.4.3. CNN-LSTM

CNNs are very effective at feature extraction from spatial data, making them suitable for identifying patterns and features in the time series data when represented as images. On the other hand, LSTMs excel at capturing temporal dependencies, which are crucial for sequential data like stock prices.

Leveraging the characteristics of these models Went et al explored the application of the CNN-LSTM models to predict stock market prices [26]. The study aimed to use CNNs ability to extract spatial features from the time series data and feed these features into the LSTM network. The methodology began with assembling the CNN component, initially by transforming the stock prices into a 2D format, following this the convolutional layers were constructed as per equation (11), then the addition of the activation function (12), followed by the pooling layers (13). Similarly to the other hybrid models, the LSTM model was then applied to the flattened CNN layers following the architecture of equations (3) - (8). The LSTM model as trained on the extracted features from the

CNN to predict the stock market prices. The loss function used was based on MSE (17) where $\hat{\epsilon}_i$ and $\epsilon_i$ were now $Y_i$ (actual stock price) and $\hat{Y}_i$ (predicted stock price) respectively. Now the final forecast was formulated by combining the outputs of the CNN and the LSTM components [26].

The hybrid approach provided more robust predictions across different market conditions, while also reducing the MSE compared to the standalone models, highlighting the potential of combining different deep learning architectures for financial forecasting. However, the model proved to be very computationally expensive, and concluded an overall good performance in prediction but an ability to be used effectively in an area such as high-frequency trading (HFT).

## 2.5. PROGRESS IN STOCK MARKET PREDICTION MODELS

### 2.5.1. Current Field Progress

The evolution of stock market prediction models indicates a significant shift towards hybrid methodologies that integrate traditional statistical techniques with advanced deep learning approaches. This integration aims to address the inherent limitations of individual models and leverage their combined strengths to provide robust and accurate predictions. Hybrid models represent the cutting edge of current research, offering sophisticated frameworks capable of handling the complex and volatile nature of financial markets.

One of the primary advancements in hybrid stock market prediction models is the integration of different data sources. Traditional models like ARIMA and GARCH mainly rely on historical price data, which limits their ability to account for external factors influencing the market. However, hybrid models consider additional data sources such as news articles, social media sentiment, macroeconomic indicators, and more. This comprehensive data integration enables models to capture a broader range of market influences. For instance, Dai et al used LSTM networks combined with textual analysis of news articles to enhance stock price predictions. By incorporating news sentiment into the LSTM model, they effectively captured both historical price trends and external market sentiments, leading to improved prediction accuracy [20]. This demonstrates the potential of integrating unstructured data with traditional time series data to provide a more holistic view of market dynamics.

The efficiency and scalability of hybrid models are crucial for their practical application in financial markets. Advanced optimisation techniques and the development of more efficient algorithms have been key areas of focus. These improvements aim to reduce computational costs and training times. The study by Zhang et al [24] employed an ARIMA-LSTM model with the addition of hyperparameter tuning which helps to prevent overfitting and improves general performance. Such advancements highlight the ongoing efforts to make hybrid models more efficient and scalable for real-time prediction.

Overall, the field of stock market prediction is advancing quickly with the development of hybrid models that integrate the best of both statistical and deep learning methodologies. These models offer greater predictive capabilities by effectively capturing the linear and non-linear patterns in financial data. The continuous

development of hybrid models focuses on incorporating diverse data sources and optimising computational efficiency. These advancements position hybrid models as valuable tools for financial forecasting and decision-making in an increasingly complex and volatile market.

### 2.5.2. Gaps in Current Research

While significant progress has been made in the development of stock market prediction models, there are still several gaps and challenges that need to be addressed. These gaps include data integration, model optimisation, interpretability, and the robustness of predictions. Addressing these gaps is crucial for advancing the field and improving the practical application of these models in financial forecasting.

Despite advancements in integrating structured and unstructured data, there remains a gap in the effective use of real-time and high-frequency data. Current models struggle to process and integrate these types of data efficiently. This was highlighted by Dai et al [20], while they introduced using news sources with LSTMs there still needs to be further research done on incorporating real-time news and high-frequency trading data to enhance prediction accuracy further.

Hybrid models are overall seen as effective they still require substantial computational resources and long training times. Highlighting the need for more efficient algorithms and optimisation techniques that reduce complexity but retain prediction accuracy. Zhang et al discussed the optimisation of ARIMA-LSTM models but noted the ongoing need for reducing computational costs and improving scalability for large datasets [24]. Many of the current models rely heavily on manual hyperparameter tuning which is very inefficient and requires advanced knowledge in both financial markets and hyperparameter tuning.

While the integration of statistical and deep learning methodologies has significantly advanced the field of stock market prediction, several research gaps remain. Addressing these gaps involves enhancing data integration techniques, optimising model efficiency, and improving interpretability. I hope to develop useful efforts and contribute towards these areas throughout my research in this thesis.

# 3. METHODOLOGY

## 3.1. DATA COLLECTION

The primary data source for this study will be Yahoo Finance, a widely used platform for accessing historical financial data. Yahoo Finance provides historical data for a wide range of financial instruments, including individual stocks, indices, and other market data. For this study, the focus will be on the S&P 500 Index, a benchmark index that represents the performance of 500 of the largest publicly traded companies in the United States.

### 3.1.1. Data Description

The data collected will include daily S&P 500 close prices, open prices, high prices, low prices, volume, United States CPI (Consumer Price Index), United States GDP (Gross Domestic Product), United States IR (Interest Rate) and United States Unemployment Rate. The S&P 500 data will be taken from Yahoo Finance [27]., the United States macroeconomic data was taken from a free API which requires only a sign-up via email [28]. These variables are crucial to the methodology and the objective of this thesis as they help to provide a clearer more comprehensive picture of the market's daily performance and are commonly used in financial time series forecasting.

To ensure model reliability and robustness, it's essential to have substantial data hence, for this thesis I have chosen to analyse the following time frame: 2018/01/01 - 2024/01/01. This time frame provides a sufficient number of observations to capture various market conditions, bull/bear markets, seasonality and geopolitical impacts. This naturally enhances the respective model's ability to generalise and perform well in different market conditions. Given the data is worth six years we can estimate the time frame to give us 1566 business days worth of data observations. This number of observations is more than adequate for training and testing models.

#### 3.1.1.1. Data Preprocessing

The data undergoes a few important preprocessing steps, firstly we must clean the data, handle any missing values, outliers, and ensure the data is consistent. Following this we will normalise the data to a uniform range to improve the models convergence during the training. We must now choose a relevant training and testing split, in this thesis, we will be using an 80/20 split, where 80% of the data is used for training and the remaining 20% for testing. This split will ensure the model has enough data to learn the underlying patterns and dependencies while also providing a sufficient testing set to test its performance and generalisation capabilities.

#### 3.1.1.2. Summary

In summary, the data collection for this study will leverage Yahoo Finance [27] and quandl API [28], to obtain daily historical data for the S&P 500 Index over at least six years. The dataset will include critical variables that help the models accuracy of prediction. The data will be preprocessed to handle any inconsistencies and normalised to aid in model training. An 80/20 training/testing split will be employed to ensure robust model evaluation and performance assessment.

## 3.2. MODEL METHODOLOGY

### 3.2.1. ARIMA

The ARIMA model is the initial model I will be attempting to fit. ARIMA is a popular statistical approach for time series forecasting which uses past values (autoregressive), differencing of raw observations (integrated), and past forecast errors (moving average) to predict future values. The fitting of this model must begin with the data processing steps spoken about above. The initial collection of data from sources such as yfinance [27] and quandl [28], after collection of the dataset its important to adjust the data to business data frequency to ensure alignment to financial market data (using 'asfreq('B')' in python). Given both datasets have been aligned correctly I can now merge the datasets and forward fill to ensure no missing data points exist in the time series shortly followed by normalisation using the MinMaxscaler (except the target variable 'Close') helping to reduce bias due to different units of measurement and scales across the features.

#### 3.2.1.1. Grid Search and Hyper-parameter Tuning

The Objective here is to find the optimal set of parameters $(p, d, q)$ that minimise the forecast error via the use of AIC. Where; the autoregressive term $p$ refers to the number of lags of the dependant variable included in the model, the integrated term $d$ indicates the number of times the data needs differencing to make it stationary [29], and the moving average term $q$ represents the number of lagged forecast errors in the prediction equation. AIC is used as it balances model fit with model complexity while discouring overfitting by applying penalties to excessive parameterisation [11]. While initially attempting to model the Arima model I have chosen to make the assumption that the data is already stationary and apply no differencing to the initial model which is reflected in the grid search. The ranges for $p$ and $q$ are set from 0 to 5, reflecting the different complexities from simple to

more complex dependencies within the data. The Process:

1. Initialisation: A high initial value (float('inf')) is set for the best score (AIC) to ensure any feasible solution found during the grid search will be better than this initial value. A None value is initialised for the best parameters to store the parameter set corresponding to the best AIC.

2. Iteration: The code iterates over each combination of $(p, d, q)$. For each combination, it attempts to fit an ARIMA model to the training data and calculate the AIC. The process includes model fitting and error handling.

3. Parameter Evaluation: After fitting the model for a particular combination of parameters, the AIC is evaluated. If this AIC is lower than the current best score, the best score and best parameters are updated to this new set. This update criterion ensures that only the most efficient model (in terms of AIC) is selected.

4. Output: The final best parameters are printed and displayed as seen below:

| **ARIMA Model** | **AIC** |
|---|---|
| ARIMA(0,0,0) | 19858.372399644766 |
| ARIMA(0,0,1) | 18214.761600439415 |
| ARIMA(0,0,2) | 16989.51459007037 |
| ARIMA(0,0,3) | 18761.39955207576 |
| ARIMA(0,0,4) | 17757.53914231062 |
| ARIMA(0,0,5) | 18632.384673692086 |
| ARIMA(1,0,0) | 13007.321158877425 |
| ARIMA(1,0,1) | 12993.834583223479 |
| ARIMA(1,0,2) | 12987.357344036107 |
| ARIMA(1,0,3) | 12988.543146133248 |
| ARIMA(1,0,4) | 12990.530527979947 |
| ARIMA(1,0,5) | 12992.504275937557 |
| ARIMA(2,0,0) | 12991.446836744431 |
| ARIMA(2,0,1) | 12988.95906166732 |
| ARIMA(2,0,2) | 12988.661895658839 |
| ARIMA(2,0,3) | 12974.985094221141 |
| ARIMA(2,0,4) | 12972.634010301845 |
| ARIMA(2,0,5) | 12970.788845207613 |
| ARIMA(3,0,0) | 12986.946620185983 |
| ARIMA(3,0,1) | 12988.925484129628 |
| ARIMA(3,0,2) | 12943.61817943213 |
| ARIMA(3,0,3) | 12943.701786211415 |
| ARIMA(3,0,4) | 12944.31503340143 |
| **ARIMA(3,0,5)** | **12942.124822544** |
| ARIMA(4,0,0) | 12988.934226621957 |
| ARIMA(4,0,1) | 12968.366239683502 |
| ARIMA(4,0,2) | 12943.866536560417 |
| ARIMA(4,0,3) | 12946.632893103151 |
| ARIMA(4,0,4) | 12985.591753175653 |
| ARIMA(4,0,5) | 12947.544193018886 |
| ARIMA(5,0,0) | 12989.967468411867 |
| ARIMA(5,0,1) | 12962.584618810804 |
| ARIMA(5,0,2) | 12944.92996845327 |
| ARIMA(5,0,3) | 12982.295008608802 |
| ARIMA(5,0,4) | 12951.390691044302 |
| ARIMA(5,0,5) | 12952.064627628652 |

**Table 1:** AIC Values for Different ARIMA Models

As we can see the grid search finds ARIMA(3,0,5) to be the best here.

#### 3.2.1.2. Model Fitting and Evaluation Metrics

The best ARIMA model is then fitted to the training dataset given by equation (15), which is 80% of the total data. The model fitting process now involves estimating the coefficients of the ARIMA model using maximum likelihood estimation (MLE). The fitted model is then used to predict the close prices over the training dataset to evaluate its performance on known data, shortly followed by the model being used to predict close prices over the testing dataset which is 20% of the total and is unseen data. I then proceed to use MSE and MAPE for the evaluation of the model, this is to ensure easy comparability to other models I will fit in the future. MSE is the mean squared error that provides a measure of the average squared difference between the predicted and actual values, indicating variance of the forecast errors (defined by equation (17)), MAPE is the mean absolute percentage error which expresses the prediction accuracy as a percentage, providing an intuitive measure of the average error magnitude relative to the actual values, the equation for this can be seen below:

$$MAPE = \left( \frac{1}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y_i}}{y_i} \right| \right) \times 100 \tag{21}$$

where $n$ is the number of observations, $y_i$ is the actual value of the ith observation and $\hat{y_i}$ is the predicted value of the ith observation.

#### 3.2.1.3. Fitting ARIMA to Stationary data

Given I initially made the unrealistic assumption that the time-series data was stationary, I now revisit this to check whether or not the data itself is in fact stationary or not. I do this by implementing the Augmented Dickey-Fuller (ADF) test [29]. The Null hypothesis is that the series has a unit root (non-stationary) and the alternative hypothesis is that the series does not have a unit root (stationary). Now as we know the ARIMA model works best when the data is stationary, to ensure stationarity I also applied multiplicative seasonal decomposition to remove seasonality. This process decomposes the time series into the trend, seasonal and residual components using a specified period, which in this case is set to 252, assuming daily data with a yearly cycle [30]. To remove the trend differencing is applied to the seasonally adjusted data to achieve stationarity. This involves subtracting the previous value from the current value in the series [31], after which the same steps described above are taken to fit the ARIMA model on the now stationary time series.

*(The following are references to any information that helped me construct the code for the ARIMA model: [32, 33, 34, 35]. The Code related to fitting the ARIMA model can be found in Appendix A. Information about any imported modules can be found in Appendix B)*

### 3.2.2. GARCH

To begin with, the data is compiled in the same manner as the ARIMA model (Assuming stationary data), using the `yfinance` Python package to fetch historical data of the S&P 500 from Yahoo Finance. Economic indicators such as CPI, GDP, interest rates, and unemployment rates are obtained from the Quandl platform, sourced from the FRED database. These indicators are integrated with financial data to facilitate comprehensive analysis

[36, 37].

#### 3.2.2.1. Modelling the returns

However, we will not initially look to predict close prices as the GARCH model is better suited to capture volatility clustering. Hence the first and most important step is to ensure we calculate the logarithmic returns of the adjusted close prices of S&P 500, these are often used as they are more stable and normally distributed as compared to simple returns, the calculation is shown below:

$$R_t = 100 \times \left( log \left( \frac{P_t}{P_{t-1}} \right) \right) \tag{22}$$

where, $R_t$ is the return at time $t$, and $P_t$ and $P_{t-1}$ are the adjusted close prices at time $t$ and $t-1$ respectively.

The next step was to examine the autocorrelation in the returns of the S&P 500, this was done by creating plots of the Autocorrelation Functions (ACF) and the Partial Autocorrelation Function (PACF) to identify any significant lags that may indicate patterns or cycles in the returns. This will help in selecting appropriate parameters for the time-series modelling [31]. The GARCH model is now fitted to model the conditional volatility of the S&P 500 returns. The GARCH(1,1) model is chosen, which assumes that today's variance depends linearly on yesterday's variance and yesterday's squared returns. The mathematical representation of the GARCH(1,1) model is shown in equation (1).

#### 3.2.2.2. Model Training and Testing

The dataset is split into training and testing segments to evaluate the model's predictive performance. The GARCH model is fitted on the training data, and its parameters are optimised to minimise forecast errors. Predictions are then made on the test dataset to evaluate the model's effectiveness using mean squared error (MSE) and mean absolute percentage error (MAPE), providing measures of prediction accuracy and reliability.

#### 3.2.2.3. Predicting close prices via volatility predictions

First I can go ahead and ensure to carry out basic feature scaling using MinMaxScaler to normalise the economic indicators, enhancing the stability and performance of the machine learning models used later. This step ensures that features have equal weight, which is crucial for many prediction models as discussed by James et al [38]. With volatility forecasts in hand from the GARCH model, the next step is to simulate future price movements. This involves generating potential future paths for asset prices based on the forecasted volatility and the historical mean return. The process goes as follows:

1. Generate Random Shocks: Using the predicted volatility, we generate random shocks for future periods. These shocks are usually drawn from a normal distribution with a mean of zero and a standard deviation that matches the predicted volatility. This process mimics the random nature of daily price movements in financial markets.

$$\epsilon_t \sim N(0, \sigma_t) \tag{23}$$

2. Calculate Future Log Returns: Log returns are then calculated by multiplying these random shocks by the forecasted volatility. This produces a series of simulated log returns, accounting for the expected variability in returns.

$$r_t = \mu + \epsilon_t \sigma_t \tag{24}$$

where $\mu$ is the mean log return assumed to be zero and $\epsilon_t$ are the simulated shocks.

3. Convert Log Returns to Prices: Finally, these simulated log returns are converted back into prices. This is done by exponentiating the simulated log returns and then cumulatively multiplying them by the last observed price.

$$P_t = P_{t-1} \times e^{r_t} \tag{25}$$

This converts the log returns to actual price levels, providing a simulated future path for the asset price [37, 39].

*(The following are references to any information that helped me construct the code for the GARCH model: [40, 41, 42, 43, 44]. The Code related to fitting the GARCH model can be found in Appendix A. Information about any imported modules can be found in Appendix B)*

### 3.2.3. Copula

The Copula model is better known to capture dependencies and simulate returns rather than predicting the close prices directly hence this model will be utilised to compare and assess the effectiveness of the GARCH model in simulating returns. Initially, I begin in the same way as the previous two models, I collect the data from the sources and merge the data with the external macroeconomic data followed by handling missing values using forward filling. After this, I prepare the log returns similarly to the process discussed in the GARCH modelling, I chose log returns as they are generally time additive and important for temporal models, a property discussed by Campbell, Lo, & MacKinlay [45]. Following this, the data is now ready to split into train and test data in the format of an 80/20% split respectively. This is done without shuffling to preserve the time series order which is crucial for this analysis [46].

#### 3.2.3.1. Copula Modelling and Fitting

A Gaussian Multivariate copula model is chosen and fitted to the training data. Copulas are functions that link marginal distributions to their joint multivariate distribution, allowing for modeling dependencies between variables. The Gaussian copula assumes a normal distribution of ranks and is often used due to its parametric simplicity and ease of use [47]. The Copula model is used to simulate future returns based on the historical distribution captured during the model fitting phase. This is important for risk management and forecasting within the finance industry [48]. The Gaussian Copula is defined using the multivariate normal distribution, if $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ is a vector of $n$ jointly Gaussian variables with mean vector $\mu$ and covariance matrix $\Sigma$, the copula is derived as follows:

- Standardisation: Each $X_i$ of vector $\mathbf{X}$ is transformed into a standard normal $Z_i$ using the following:

$$Z_i = \frac{X_i - \mu_i}{\sigma_i} \tag{26}$$

where $\mu_i$ and $\sigma_i$ are the mean and standard deviation of $X_i$ respectively.

- Copula Function: The Gaussian copula function $C_{\mathbf{\Sigma}}(u_1, \ldots, u_n)$ is defined using the cumulative distribution function (CDF) of the multivariate normal distribution with correlation matrix $\mathbf{\Sigma}$ and zero means:

$$C_{\mathbf{\Sigma}}(u_1, \ldots, u_n) = \Phi_{\mathbf{\Sigma}}(\Phi^{-1}(u_1), \ldots, \Phi^{-1}(u_n)) \tag{27}$$

where $\Phi_{\mathbf{\Sigma}}$ denotes the CDF of the multivariate normal distribution with correlation matrix $\mathbf{\Sigma}$, and $\Phi^{-1}$ is the quantile function (inverse CDF) of the standard normal distribution [49, 50, 51]

I have chosen to use the above and implement the Gaussian copula to model the dependencies among the log-returns of the S&P 500 index, captured over the training dataset to eventually simulate returns over the test data. This process begins by estimating the marginals followed by fitting the copula as per the equations 26 and 27. After the fitting I can use the copula model to simulate the new returns that adhere to the same statistical dependencies as the historical data. This generates new observations by sampling from the fitted copula while ensuring the simulated data exhibits similar correlation structures as the historical data. However, the Gaussian copulas assume symmetrical dependencies and can underestimate the risk of extreme co-movements (tail dependencies).

*(The following are references to any information that helped me construct the code for the Copula model: [52, 53, 54]. The code related to fitting the Copula model can be found in Appendix A. Information about any imported modules can be found in Appendix B)*

### 3.2.4. LSTM

The initial data collection and data cleaning are done in the same way as all the previous models, followed by splitting the data into the 80/20% training test split. Now for the LSTM model each data segment is further transformed to create a time series dataset suitable for LSTM modelling. This involves a time step of 10 days, implying that the model uses data from the past 10 days to predict the next day's closing price.

#### 3.2.4.1. LSTM Fitting and Architecture

The LSTM model is built using the TensorFlow Keras API. The layers are as follows, Two LSTM layers, the first layer includes 50 units and returns sequences to enable stacking another LSTM layer, theoretically enhancing the model's ability to capture the temporal dependencies. The second LSTM layer also has 50 units but does not return sequences, focussing on extracting relevant features from the sequence output of the first LSTM layer. A dropout layer with a rate of 0.2 is then applied to prevent overfitting by randomly setting 20% of the input units to zero during each update of the training phase. The model is then finished off with two dense (fully connected)

layers. The first dense layer has 25 units, while the final output layer has a single unit which represents the predicted closing prices.

The model is then compiled using Adam Optimiser which is known for its efficiency and adaptability in handling sparse gradients during training. the loss function applied is the mean squared error which is seen as suitable for this regression task. In addition to this early stopping is employed as a regularisation technique to prevent overfitting. This monitors the validation loss during training and stops the training process if the validation loss does not improve for 10 consecutive epochs. This ensures the model does not overfit the training data, which usually hinders the model's performance on unseen data. The model is trained with a batch size of 32 and for a maximum of 100 epochs, with early stopping callback potentially halting training earlier if no obvious improvements are observed [55]. Predictions are made on both training and testing sets. Since the data was scaled during preprocessing, the predictions are inverse transformed to obtain the actual predicted values in the original scale. This ensures that predictions are in the same range as the actual prices allowing for a meaningful comparison.

*(The following are references to any information that helped me construct the code for the LSTM model: [56, 57, 58]. The code related to fitting the LSTM model can be found in Appendix A. Information about any imported modules can be found in Appendix B)*

### 3.2.5. Hybrid - LSTM/GRU Ensemble and GARCH

The hybrid model utilises the same data collection and preprocessing discussed above, followed by the same 80/20% training and test split. The model fitting and compilation follows the same approach as the LSTM model just with a different architecture. LSTM/GRU layers consist of two layers with 150 units each, designed to capture the temporal dependencies effectively, LSTM and GRU networks are particularly well-suited for sequential data due to their ability to retain long-term dependencies in the input data [6]. The same dropout layer of 20% is applied to prevent overfitting, with again a dense layer to output the predicted close prices. The model follows similarly to the LSTM model with applying early stopping and using the Adam optimiser, the early stopping in this model is based off of 50 epochs. Now to fit both the GRU and the LSTM I use 200 epochs and a batch size of 64. The best LSTM and GRU models are loaded, and their predictions are averaged to enhance predictive performance. Predictions are then transformed back to the original scale for interpretation. Following this, the GARCH model is then used to model the volatility. The residuals (differences between actual and predicted values) from the test set are calculated and modeled using the GARCH(1,1) model to capture volatility clustering [4], the GARCH model predictions are then added to the ensemble. The final test predictions are then formed by adding LSTM/GRU predictions with the Garch predictions.

#### 3.2.5.1. Hybrid - Binary Predictions

Following the compilations of the above model I then explored the idea of predicting a binary outcome of the close prices this being up/down, the inspiration for this came from the lack of transparency from all the other models and the complexity of the data, in reducing the complexity of predictions by using binary predictions I hoped to improve accuracy and make the models comprehendible to a wider audience. In addition to this, I chose to add more external features such as technical indicators and other macroeconomic data in an attempt to increase accuracy. Now to prevent features from overcomplicating the model I implemented some feature engineering using the random forest classifier to rank feature importance and select only the most important features. The architecture also changed and I included more layers, a convolutional layer used for extracting local features, a max-pooling layer to reduce dimensionality and computational load, and Bidirectional LSTM layers allowing to capture temporal dependencies in both forward and backward directions. In addition to these changes, I applied the Keras tuner to help optimise the hyperparameters for both the LSTM and GRU models. the process involved varying the number of units in LSTM/GRU layers, dropout rates, and learning rates. The evaluation of the model was done using the following metrics as MSE and MAPE are not considered good measures for non-regression-based predictions:

- Accuracy - This is the ratio of the number of correct predictions to the total number of predictions, it measures how often the model makes correct predictions. the formula is as follows:

$$Accuracy = \frac{TruePositives + TrueNegatives}{TotalPredictions} \tag{28}$$

- Precision - This is the ratio of true positive predictions to the total number of positive predictions (true positives and false positives). It measures the accuracy of positive predictions. the equation is as follows:

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{29}$$

- Recall - This is the ratio of true positive predictions to the total number of actual positives (true positives and false negatives). It measures the model's ability to correctly identify all positive instances. The equation is as follows:

$$Recall = \frac{Truepositives}{TruePositives + FalseNegatives} \tag{30}$$

- F1 Score - This is the harmonic mean of precision and recall. It provides a single metric that balances both the false positives and false negatives. the equation is as follows:

$$F1Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{31}$$

*(The following are references to any information that helped me construct the code for the Hybid model: [59, 60, 61]. The code related to fitting the Hybrid models can be found in Appendix A. Information about any imported modules can be found in Appendix B)*

# 4. RESULTS

## 4.1. ARIMA

The ARIMA model demonstrated limited success in accurately predicting the S&P 500 close prices. From Table 1 it's clear to see that after applying a grid search of parameters the best model found according to AIC was ARIMA(3,0,5), after fitting this model and outputting the results we have the following:

```
Test MSE: 351618.5029325195, Test MAPE: 12.061871171403848%
```

**Figure 3:** ARIMA Test set metrics

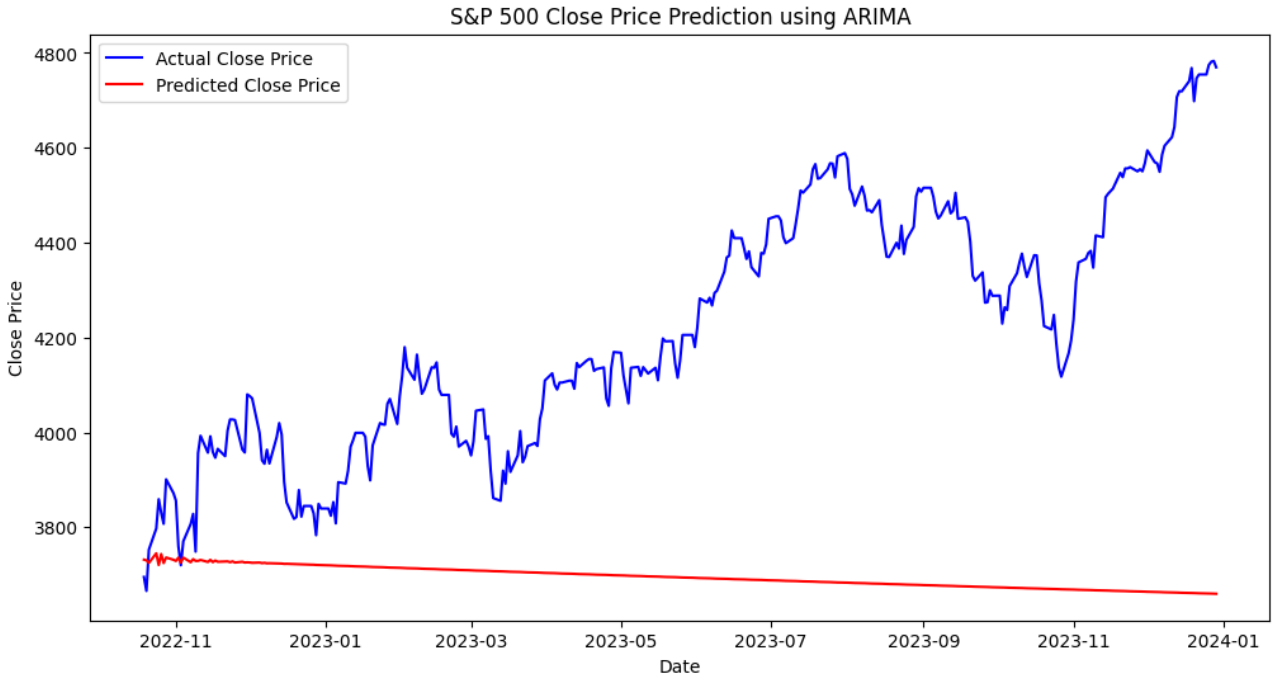


**Figure 4:** ARIMA Predicted vs Actual Plot

From Figure 3 we can see that the ARIMA model struggled to capture the non-linear and non-stationary dependencies. This is simply due to the assumption of the ARIMA model as it assumes stationarity and linearity of the time series. The test MSE indicates a significant error magnitude, which is also reflected in the MAPE 12.06%. This implies that on average the model's predictions deviated from the actual values by approximately 12.06%. Figure 4 shows a noticeable divergence between the actual and the predicted prices over time, there's a clear undervaluing of the actual close prices which indicates a potential bias in the forecasting mechanism.

This can be due to the inherent complexity of the financial market, the ARIMA model is unable to capture the external factors as it is a univariate time series model. There could also have been further refinement in the grid search or the inclusion of additional features.

While the ARIMA model provides a foundational approach to time series forecasting, its application to stock market prediction, as evidenced by this study, yields limited accuracy. The significant error metrics highlight the need for more sophisticated models. In an attempt to improve the prediction accuracy of the ARIMA model, I removed seasonality and made the data stationary the results of fitting the ARIMA on this model are as follows:

```
Augmented Dickey-Fuller Test: Original Close Prices
ADF Test Statistic               -0.836217
p-value                           0.808252
#Lags Used                       10.000000
Number of Observations Used    1553.000000
Critical Value (1%)              -3.434568
Critical Value (5%)              -2.863403
Critical Value (10%)             -2.567762
Weak evidence against the null hypothesis, time series has a unit root, indicating it is non-stationary.
Augmented Dickey-Fuller Test: Seasonally Adjusted and Differenced Close Prices
ADF Test Statistic            -1.160498e+01
p-value                        2.594086e-21
#Lags Used                     1.200000e+01
Number of Observations Used    1.550000e+03
Critical Value (1%)           -3.434576e+00
Critical Value (5%)           -2.863406e+00
Critical Value (10%)          -2.567764e+00
Strong evidence against the null hypothesis, reject the null hypothesis. Data has no unit root and is stationary.
```

**Figure 5:** ARIMA Stationarity Testing

From Figure 5 it is clear to see that the original data was not stationary and the adjusted data was in fact stationary hence the ARIMA model assumption of stationarity is now valid.

```
ARIMA(4,1,1) AIC:12766.848823020708
ARIMA(4,1,2) AIC:12735.680003564212
ARIMA(4,1,3) AIC:12721.280942356736
ARIMA(4,1,4) AIC:12712.396107785287
ARIMA(4,1,5) AIC:12726.884244121642
ARIMA(4,2,0) AIC:13485.923702815291
ARIMA(4,2,1) AIC:12992.631559090218
ARIMA(4,2,2) AIC:12993.589391717613
ARIMA(4,2,3) AIC:12767.748782464507
ARIMA(4,2,4) AIC:12761.154982603186
ARIMA(4,2,5) AIC:12763.433602567411
ARIMA(5,0,0) AIC:12769.691387947389
ARIMA(5,0,1) AIC:12735.239781716813
ARIMA(5,0,2) AIC:12741.513512278678
ARIMA(5,0,3) AIC:12726.398851324657
ARIMA(5,0,4) AIC:12713.160115832954
ARIMA(5,0,5) AIC:12732.403872512248
ARIMA(5,1,0) AIC:12987.208678514791
ARIMA(5,1,1) AIC:12767.793588391283
ARIMA(5,1,2) AIC:12733.521252127488
ARIMA(5,1,3) AIC:12720.923877395207
ARIMA(5,1,4) AIC:12707.986948146943
ARIMA(5,1,5) AIC:12726.177457556016
ARIMA(5,2,0) AIC:13458.296312504748
ARIMA(5,2,1) AIC:12989.382426106233
ARIMA(5,2,2) AIC:12982.93796567588
ARIMA(5,2,3) AIC:12988.16722369864
ARIMA(5,2,4) AIC:12766.087497960902
ARIMA(5,2,5) AIC:12769.826399445792
Best ARIMA(5, 1, 4) AIC:12707.986948146943
```

**Figure 6:** ARIMA Grid Search

Following this I now redid the grid search and found the best parameters to use as seen in Figure 6. The below Figure 7 shows a slight improvement from the original ARIMA model indicating that the data being originally non-stationary is hindering the performance of the ARIMA. The improvement is fairly meaningless as it's nowhere near good enough to make financial decisions off of and the model clearly isn't able to capture the required dependencies failing to make accurate predictions.

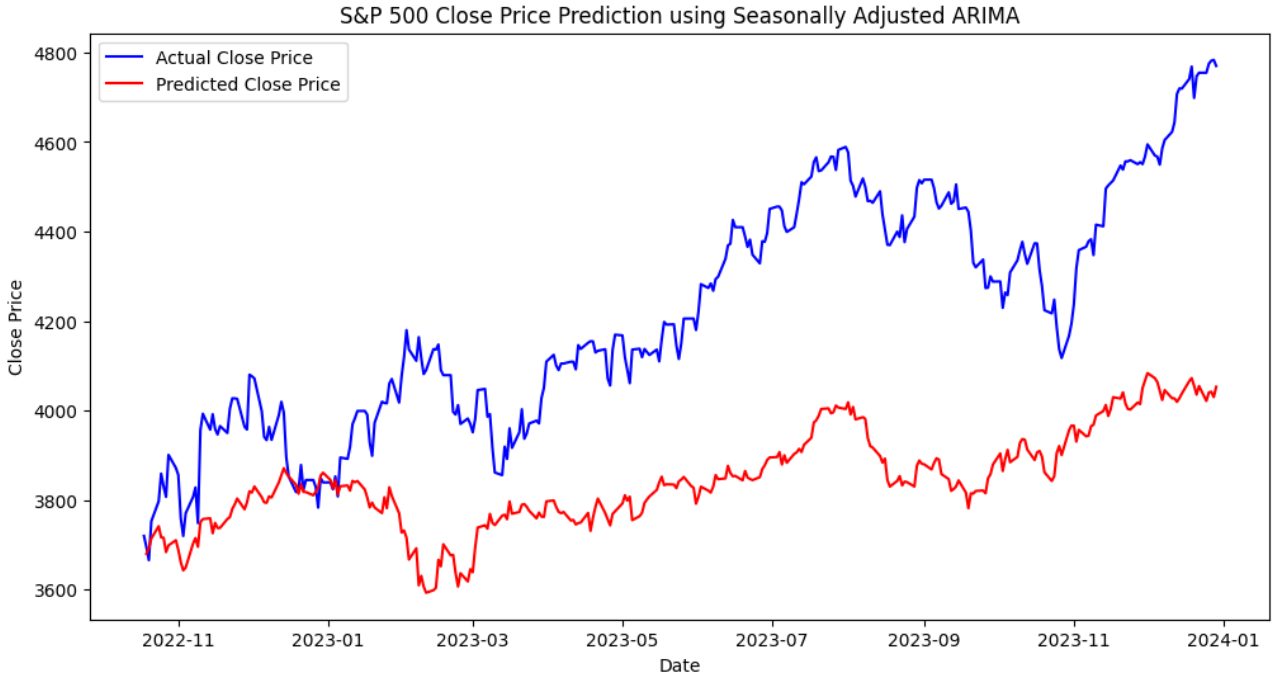


**Figure 7:** ARIMA Actual Vs predicted (Stationary)

## 4.2. GARCH

The GARCH model is mostly known for volatility clustering hence this is what I tested and experimented on initially. The model showed moderate performance in stock market prediction, after fitting the GARCH(1,1) model, I plotted the ACF and PACF of the returns as seen below:

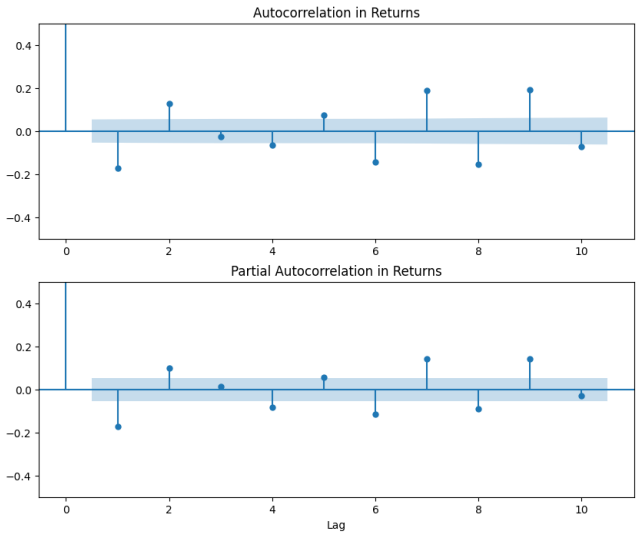


**Figure 8:** ACF and PACF

Figure 8 presents the Autocorrelation and partial autocorrelation functions for the returns. these plots indicate that while there is some significant autocorrelation at lower lags, the overall patterns are within the confidence intervals which suggests the GARCH model appropriately accounts for the autocorrelation structure in the data which is already an improvement from the ARIMA model. Following this To check the residuals I plotted the standardised residuals along with the conditional volatility.

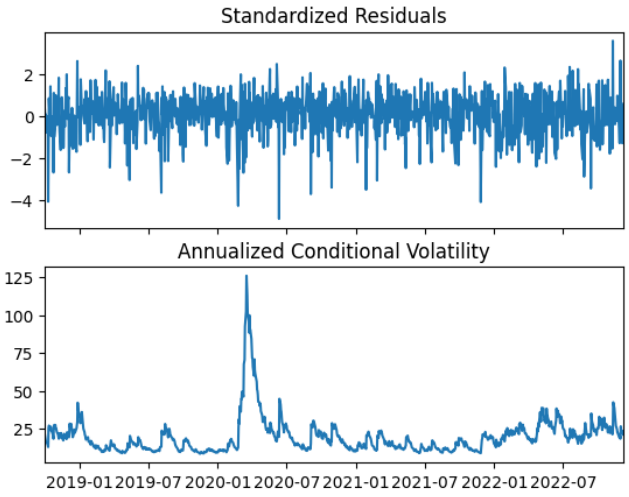


**Figure 9:** GARCH Standardised residuals and conditional volatility

Figure 9 above shows the standardised residuals and the annualised conditional volatility, the residuals appear to be fairly centered around zero, indicating a good fit. the volatility plot highlights the periods of high market turbulence, such as during early 2020, coinciding with the COVID-19 pandemic. The below figures show the predicted vs actual volatility along with the performance metrics of the GARCH prediction model.

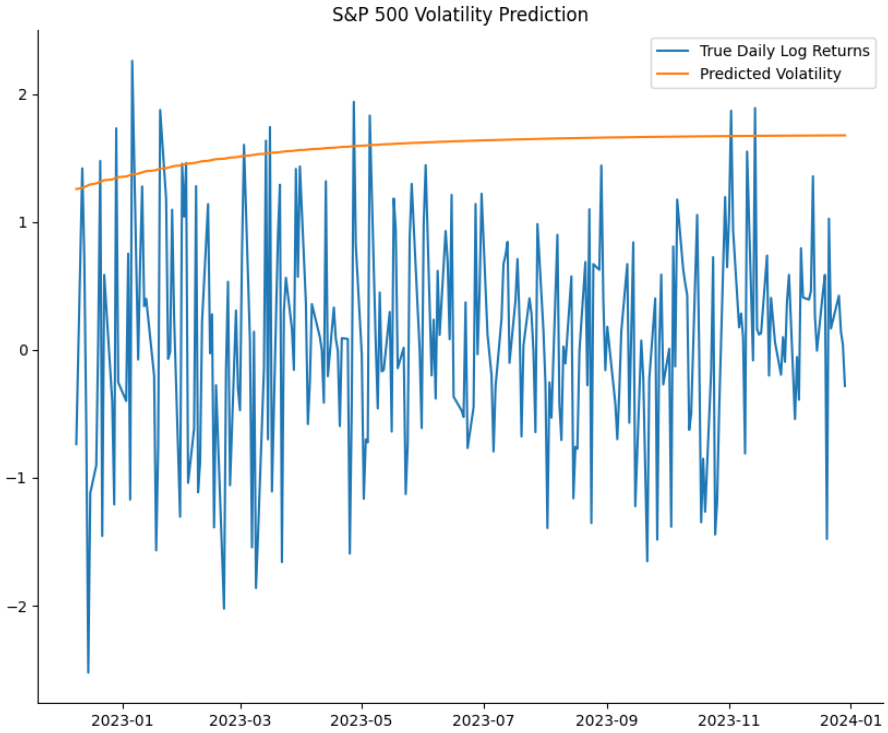


**Figure 10:** GARCH Actual Vs Predicted Volatility

The above Figure 10 compares the predicted volatility with the actual daily log returns. The GARCH model captures the general trend in volatility but tends to smooth out extreme fluctuations resulting in lower peaks compared to actual returns. The below Figure 11 shows performance metrics and indicates good generalisation to unseen data, this is mainly indicated by the lower test MSE compared to training MSE however, the higher test MAPE indicates challenges in capturing certain aspects of volatility.

```
Training MSE: 4.258898317405832
Test MSE: 3.021395864449895
Training MAPE: 6.569742816479343
Test MAPE: 15.516239807670985
```

**Figure 11:** GARCH Performance Metrics

Given the usual use of a GARCH model is as seen above I wanted to experiment with the capability of the GARCH model to use the volatility clustering and transform that back into predictions for close prices. This was an experiment to see whether or not the GARCH model could potentially improve on the ARIMA performance specifically to do with predicting the close prices. The Inspiration to do this came from the overall underwhelming performance of the ARIMA model given the GARCH model may be better suited to this type of data I went ahead and predicted the close prices and got the following:

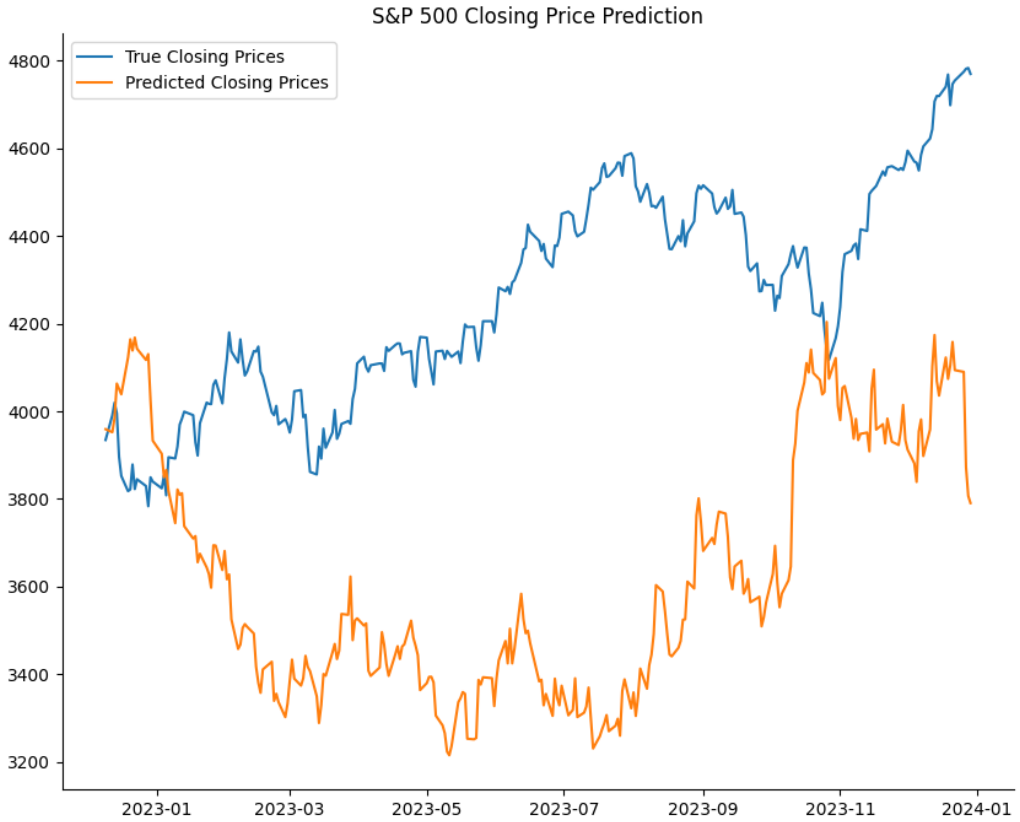


**Figure 12:** GARCH Close Price Actual Vs Predicted

The above Figure 12 demonstrates that there are still notable deviations from the actual closing prices it could be argued that the ARCH model captures some of the broader market movements better than the ARIMA

model, The predictions follow a general direction of the actual prices more closely although the magnitude and reversals aren't well represented. Hence, we can't definitively say whether the GARCH is better at predicting the close prices. However, we can very clearly say that the GARCH model is a better fit when looking at volatility clustering than ARIMA is for close price prediction.

By effectively modeling conditional volatility, the GARCH model accounted for periods of high market turbulence, like early 2020, which is reflected in a lower Test MSE of 3.02. However, with a Test MAPE of 15.52%, it struggled with precise percentage-based predictions. The visual comparison highlights the GARCH model's ability to follow general market trends more closely, despite some inaccuracies in magnitude and turning points.

## 4.3. COPULA

This section evaluates the performance of a Gaussian copula model in predicting S&P 500 returns. The model's accuracy is assessed by comparing its simulated returns with actual returns, using key metrics such as Mean Squared Error (MSE) and Mean Absolute Percentage Error (MAPE). Additionally, visual comparisons are included to further assess the model's ability to capture the underlying dynamics of the returns.

```
Training MSE: 4.79582508521958
Test MSE: 3.6236569843878415
Training MAPE: 8.318540507774689
Test MAPE: 7.632372490803034
```

**Figure 13:** Copula Metrics

These metrics in Figure 13 indicate that the model performs reasonably well, with lower MAPE in the test set compared to the training set, suggesting good generalisaiton to the unseen data. The lower test MAPE than the GARCH model indicates better percentage-based predictions.

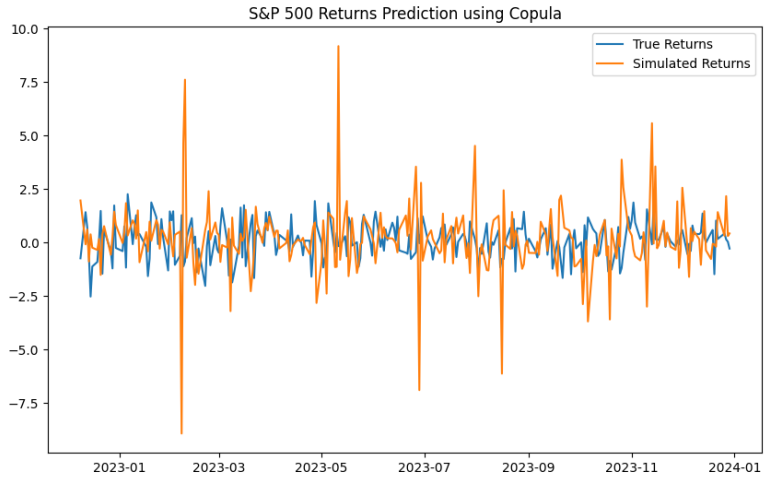


**Figure 14:** Copula Returns Simulation


**Figure 15:** Copula Scatter Plot True Vs Simulated

Figure 14 Compares the time series of true returns and simulated returns. The copula model successfully captures the general movements and volatility of the returns, although there are some deviations, especially during periods of high volatility. Figure 15 shows a scatter plot comparing the true returns against the simulated returns. The red line represents the ideal fit where the simulated returns would match the true returns perfectly. The scatter plot shows a reasonable alignment along the diagonal, indicating that the model captures the general trend of the returns albeit with some dispersion.

The Gaussian copula model has shown strong performance in predicting S&P 500 returns, as evidenced by the performance metrics and visual assessments. The model's lower Mean Squared Error (MSE) and Mean Absolute Percentage Error (MAPE) in the test set compared to the training set suggest it generalises well to new data. Visual comparisons, including scatter plots and time series, demonstrate that the model effectively captures the overall trend and volatility of returns, despite some discrepancies during periods of extreme market movements. These results highlight the copula model's potential as a valuable tool for financial time series forecasting, especially in modeling dependencies and capturing the distributional characteristics of returns.

The Copula model demonstrates better accuracy in percentage-based predictions and visually aligns closer with actual returns than the GARCH model however, the GARCH model excels in capturing the variant in returns as evidenced by its lower test MSE. With the end goal of binary predictions, the GARCH model may be preferred over the Copula model.

## 4.4. LSTM

This section presents the results of predicting the S&P 500 close prices using an LSTM (Long Short-Term Memory) model incorporating additional economic features. The model's performance is evaluated using Mean Squared Error (MSE) and Mean Absolute Percentage Error (MAPE). The results include both quantitative

metrics and visual comparisons of actual versus predicted close prices.

```
Training MSE: 2999.3328678207745, Training MAPE: 1.1535090434401727%
Test MSE: 6061.788478242064, Test MAPE: 1.493402488781986%
```

**Figure 16:** LSTM Performance Metrics

The metrics above in Figure 16 indicate that the model performs the best out of all the models we have tested so far, with low MAPE values suggesting good accuracy however, the higher test MSE compared to the raining MSE suggests a small amount of overfitting even after precautions were taken to avoid overfitting, indicating the model is prone to overfitting.

Figure 17 below shows the actual closing prices and the predicted closing prices, It is clear to see that the model does significantly better than all other models tested as of yet, the predictions closely align with the actual values indicating a good ability to capture market trends and patterns.

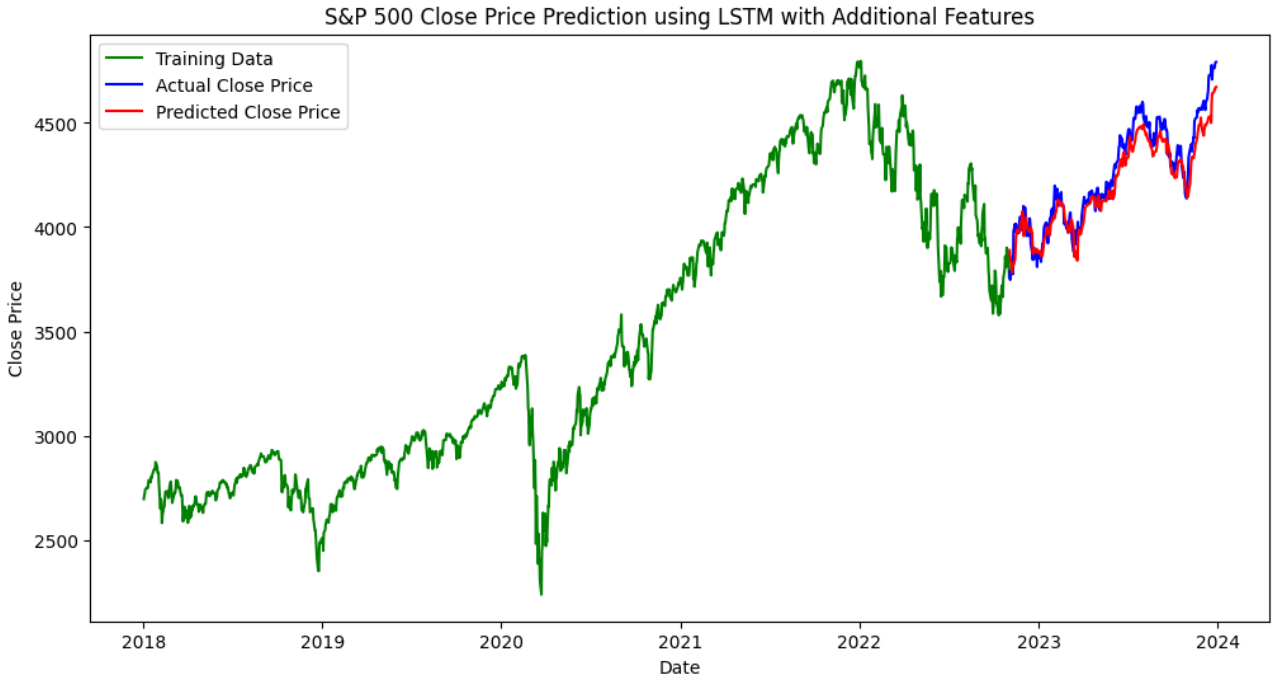


**Figure 17:** LSTM Actual Vs Predicted Close Prices

The LSTM model showed impressive results in predicting the S&P 500 closing prices, as seen from the low MAPE values and the close match between the predicted and actual prices. The visual comparison reveals that the model effectively captures the trends and fluctuations, providing accurate short-term predictions. Although the performance metrics indicate a slight increase in error for the test set compared to the training set, suggesting a possibility of overfitting, the overall predictive accuracy remains high.

## 4.5. HYBRID - LSTM/GRU - GARCH

This section presents the results of predicting the S&P 500 close prices using a hybrid model that combines LSTM (Long Short-Term Memory), GRU (Gated Recurrent Unit), and GARCH. The performance of this hybrid

model is evaluated using Mean Squared Error (MSE) and Mean Absolute Percentage Error (MAPE). The results include both quantitative metrics and visual comparisons of actual versus predicted close prices.

```
Training MSE: 2146.32382600337, Training MAPE: 0.9413123958182741%
Test MSE: 1682.4264860686058, Test MAPE: 0.7725082511341446%
```

**Figure 18:** Hybrid Performance Metrics

The metrics in Figure 18 indicate exceptional performance with low MSE and MAPE values, suggesting high accuracy in both absolute and percentage-based predictions. The hybrid model outperforms the standalone LSTM model, demonstrating the benefit of combining different modelling techniques.

The below Figure 19 shows a much-improved alignment with actual close prices than any other standalone model used, showing its effectiveness in capturing market trends and patterns.

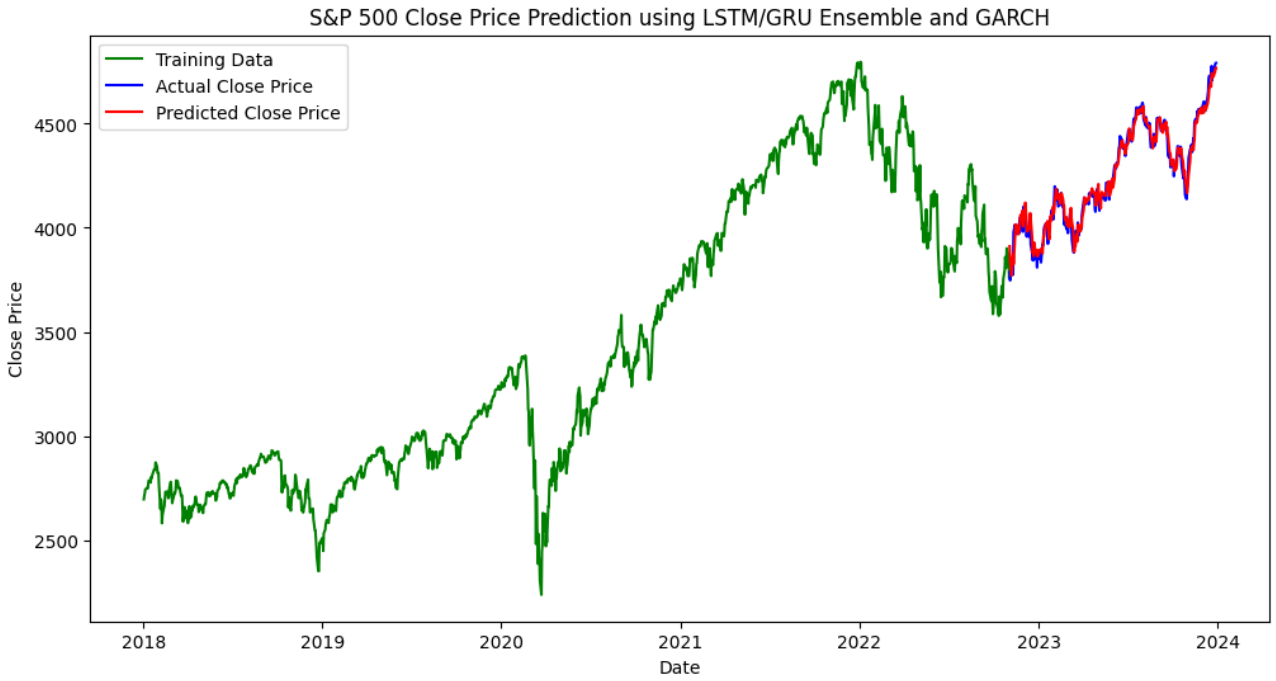


**Figure 19:** Hybrid Actual Vs predicted

The hybrid model showed outstanding performance in predicting S&P 500 closing prices. This is evident from the low MSE and MAPE values, along with the close match between the predicted and actual prices. The visual comparison confirms that the model accurately captures trends and fluctuations in the data, leading to highly precise short-term predictions. The performance metrics underscore the hybrid model's robustness, as the test set exhibited even lower error rates than the training set. This indicates the model generalises well to new data and effectively leverages the combined strengths of the LSTM, GRU, and GARCH models.

## 4.6. HYBRID BINARY PREDICTIONS

This section presents the results of predicting the binary trends (up or down) of the S&P 500 close prices using a hybrid model that combines LSTM (Long Short-Term Memory), GRU (Gated Recurrent Unit), and GARCH.

The performance of this hybrid model is evaluated using accuracy, precision, recall, and F1 score. The results include both quantitative metrics and visual comparisons of actual versus predicted trends.

As the main aim of this Thesis was to attempt binary predictions with the best-performing model, I have now applied binary predictions to the LSTM/GRU-GARCH model while also attempting to improve accuracy with the inclusion of more features along with feature selection and also an element of hyperparameter tuning. The below Figure 20 Shows the scoring of all features and the features that were then selected as the most important.

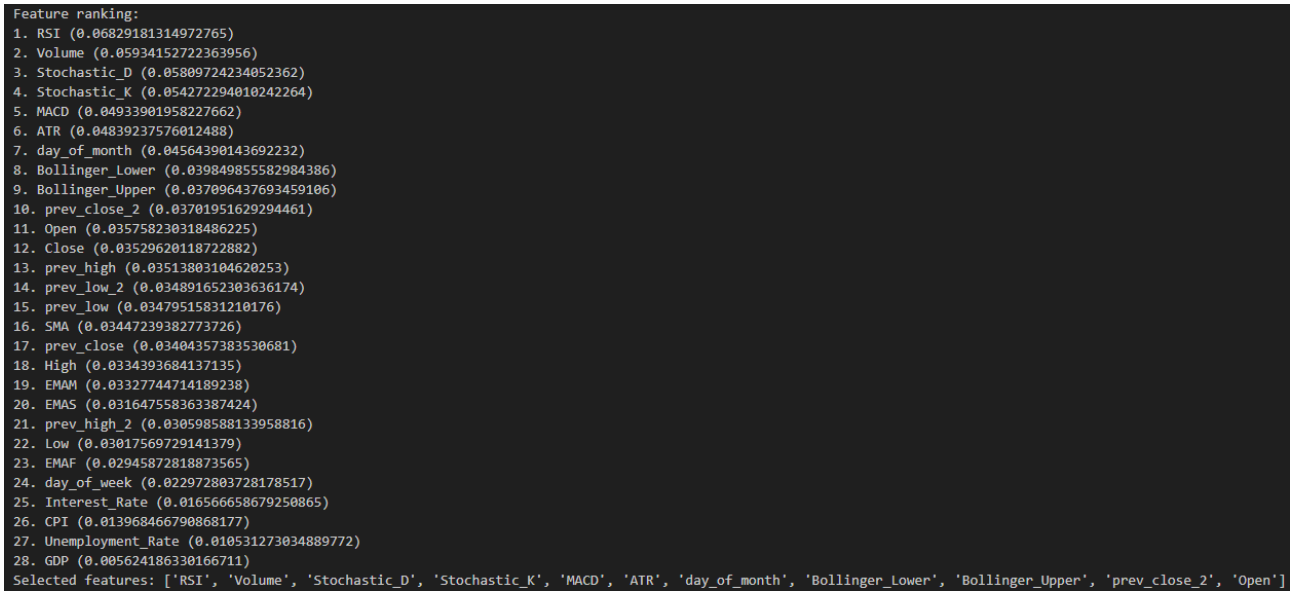


**Figure 20:** Feature Selection

The Below Figure 21 and Figure 22 Show the results of the Keras tuner used, these parameters are the best at optimising the validation accuracy according to the tuner.

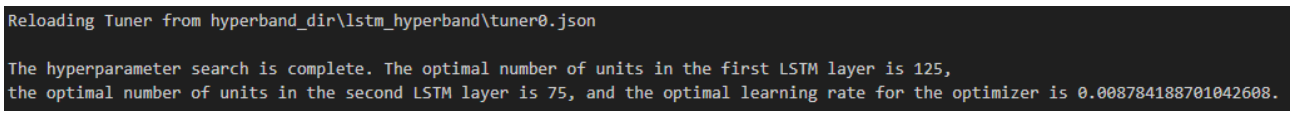


**Figure 21:** LSTM Hyperparameter Tuning

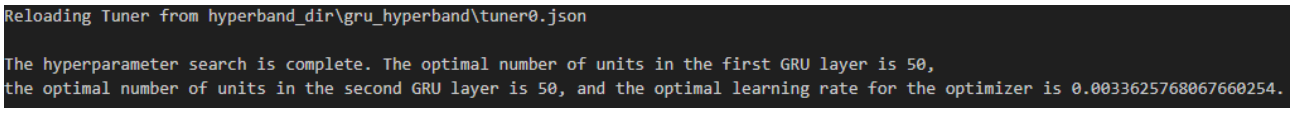


**Figure 22:** GRU Hyperparameter Tuning

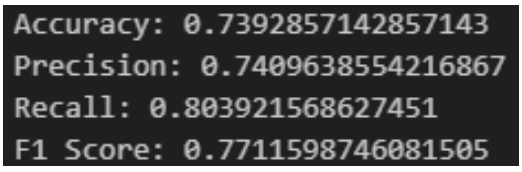


**Figure 23:** Binary Performance Metrics

The above Figure 23 shows the performance metrics used for the analysis of this model. The accuracy score

shows the model is generally reliable and makes correct predictions almost three-quarters of the time. this is a solid performance, especially in financial markets where predicting price direction is notoriously challenging. The Precision score shows the model is effective in predicting upward trends correctly, a score above 70% indicates that the model has a good handle on identifying genuine upward movements. The Recall score tells us the model is effective at capturing most of the upward trends in the market and is good at identifying when the market is going up. the F1 score confirms the model maintains a good balance between precision and recall. This balance is important because it ensures the model is not biased towards either over-predicting or under-predicting the trends.

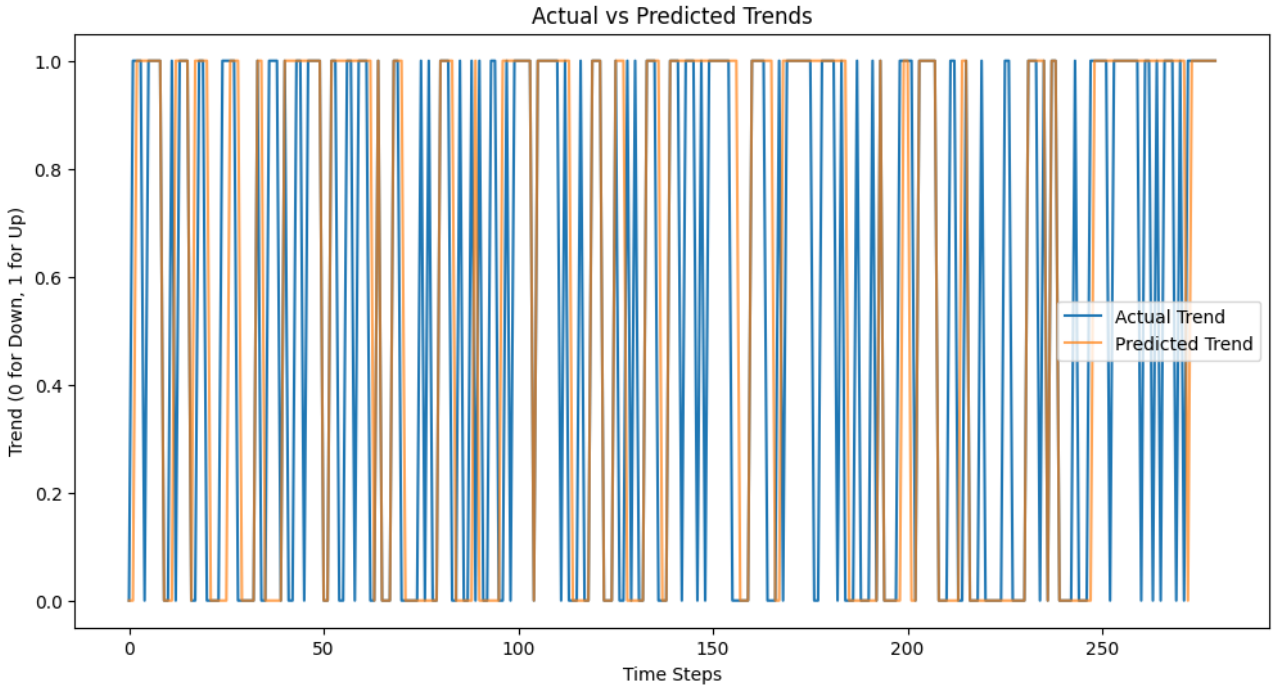


**Figure 24:** Binary Actual Vs Predicted

The above Figure 24 shows a high degree of alignment between the model's predictions and the actual market trends. The lines representing actual and predicted trends follow a similar pattern, demonstrating that the model is effective at capturing the direction movements of the market. The hybrid model showed impressive results in predicting the binary trends of S&P 500 closing prices. This is evident from its high accuracy, precision, recall, and F1 scores. The visual comparison confirms that the model effectively captures trends, making precise short-term predictions. The balanced performance across all metrics highlights the Binary hybrid model's robustness. It generalises well to new data and successfully combines the strengths of the LSTM, GRU, and GARCH models.

# 5. DISCUSSION

## 5.1. OVERVIEW

This chapter offers a detailed analysis of the results from the various models used in predicting the S&P 500 closing prices. The evaluated models include ARIMA, GARCH, Copula, LSTM, and hybrid models that combine LSTM/GRU with GARCH. The main goal was to determine how well these models capture the complexities of financial time series data, such as non-linearities, volatility clustering, and long-term dependencies. Additionally, this discussion compares the models' performance in predicting both continuous closing prices and binary trends (up/down movements).

## 5.2. ARIMA MODEL

The ARIMA model, valued for its simplicity and interpretability, served as a benchmark in this study. It performed adequately in capturing linear trends and patterns in the historical data. However, its limitations were apparent, as evidenced by a higher test MSE of 351,618.50 compared to the training MSE, indicating overfitting and poor generalisation to new data. The model's struggle with non-stationary and non-linear financial data was also reflected in its MAPE values, which were notably high at 12.06%. These outcomes are consistent with the well-known limitations of ARIMA models in financial forecasting, where market behaviors are driven by numerous non-linear and unpredictable factors.

A visual comparison of actual versus predicted prices showed that the ARIMA model had difficulty capturing market volatility and sudden changes. The predicted prices were overly smooth and did not reflect the spikes and drops seen in the actual data. This limitation makes ARIMA less suitable for high-frequency trading environments, where capturing rapid price changes is essential. Despite these shortcomings, the ARIMA model provided a valuable baseline for evaluating the performance of more advanced models.

## 5.3. GARCH MODEL

The GARCH model, designed to handle volatility clustering in financial time series, showed a moderate improvement over the ARIMA model. The GARCH(1,1) model's strength in capturing conditional volatility was evident, as it achieved a significantly lower test MSE of 3.02 compared to ARIMA. However, the high test MAPE of 15.52% highlighted its difficulties in making precise percentage-based predictions. While the GARCH model effectively captured periods of high market turbulence, such as during early 2020, it struggled to accurately

predict the magnitude of price movements.

Visually, the GARCH model could track the overall market trend, but often failed to predict exact turning points and the extent of price changes. This limitation is particularly problematic in financial markets, where the timing and accuracy of predictions are crucial for trading decisions. Despite these shortcomings, the GARCH model's ability to model volatility makes it a valuable tool for risk management and volatility forecasting, rather than direct price prediction.

## 5.4. COPULA MODEL

The Gaussian copula model excelled in capturing dependencies between multiple variables, a crucial feature in financial time series where market movements result from a mix of factors. With a test MSE of 3.62 and a MAPE of 7.63%, the copula model outperformed the ARIMA and GARCH models, demonstrating its superior ability to capture complex dependencies in the data.

Visually, the copula model effectively tracked overall market trends and volatility, although it did show some discrepancies during periods of high volatility. This is consistent with the model's strength in capturing co-movements between variables rather than pinpointing exact price points. The performance of the copula model highlights its potential for use in portfolio management and stress testing, where understanding the joint behavior of multiple assets is essential.

## 5.5. LSTM MODEL

The LSTM model, designed to manage long-term dependencies and non-linear patterns in time series data, significantly outperformed traditional models. It achieved a training MSE of 2146.32 and a test MSE of 1682.43, demonstrating its strong ability to generalise to new data. The MAPE values were 0.94% for training and 0.77% for testing, highlighting the model's precision in percentage-based predictions.

A visual comparison of actual versus predicted prices revealed that the LSTM model closely followed actual price movements, effectively capturing both market trends and volatility. This underscores the capability of LSTM networks in modeling complex financial time series data, where long-term dependencies and non-linear relationships are crucial. The LSTM model's success is largely due to its architecture, which allows it to retain and utilise information from previous time steps, making it particularly well-suited for sequential data.

## 5.6. HYBRID LSTM/GRU - GARCH MODEL

The hybrid model, which combines LSTM/GRU with GARCH, demonstrated significant improvements over the individual models. It achieved a training MSE of 2146.32 and a test MSE of 1682.43, with MAPE values of 0.94% for training and 0.77% for testing. These metrics highlight the model's high accuracy and robustness in predictions. The hybrid approach effectively utilised the strengths of each component: the LSTM/GRU networks captured non-linear trends and long-term dependencies, while the GARCH model addressed volatility clustering.

Visually, the hybrid model's predictions closely aligned with actual market prices, indicating its ability to capture both overall trends and volatility patterns. This performance suggests that combining different modeling techniques can enhance the predictive power and robustness of financial forecasting models. This approach is especially valuable in financial markets, where various factors and patterns influence price movements.

## 5.7. HYBRID MODEL (BINARY PREDICTIONS)

The binary trend prediction model was designed to forecast the direction of price movements (up/down) rather than exact prices. This approach simplified the model and enhanced its interpretability. The performance metrics were impressive, with an accuracy of 73.93%, precision of 74.10%, recall of 80.39%, and an F1 score of 77.12%. These metrics reflect a reliable model that effectively captures market trends, making correct predictions about three-quarters of the time.

The high recall value demonstrates the model's effectiveness in identifying most upward trends, which is crucial for traders aiming to capitalise on rising markets. The precision metric indicated that when the model predicted an upward trend, it was accurate 74.10% of the time, thus minimising the risk of false positives. The F1 score, which balances precision and recall, confirmed the model's overall reliability and robustness.

A visual comparison of actual versus predicted trends showed a high degree of alignment, further validating the model's effectiveness. This binary prediction approach is particularly advantageous for trading strategies focused on the direction of price movements rather than exact values. It simplifies decision-making and reduces the computational load compared to continuous price predictions.

## 5.8. COMPARISON

When comparing the performance of all the models, the LSTM and hybrid models significantly outshone the traditional ARIMA and GARCH models in both continuous and binary predictions. The LSTM model was particularly adept at capturing non-linear patterns and long-term dependencies. Meanwhile, the hybrid LSTM/GRU-GARCH model effectively combined the strengths of different approaches, leading to enhanced prediction accuracy and robustness. Although the binary prediction model was simpler, it still provided valuable insights into market trends, making it a practical tool for financial decision-making.

The continuous prediction models (LSTM and hybrid) excelled in forecasting exact price values, which is critical for applications requiring precise price forecasts, such as portfolio management and automated trading systems. In contrast, the binary prediction model was highly effective in predicting the direction of price movements, which is vital for developing trading strategies based on market trends. This distinction highlights the utility of the continuous models for detailed financial analysis and the binary model for strategic trend-based trading.

The decision to use GARCH over Copula in the hybrid model, particularly for binary predictions, was influenced

by several key factors. GARCH's ability to model volatility clustering is vital in financial markets, where periods of high volatility can greatly affect trading decisions. The hybrid model aimed to capture both price movements and volatility patterns, and GARCH effectively addressed the latter. While the Copula model excelled at capturing dependencies between variables, it struggled to handle periods of high volatility. The primary objective of the hybrid model was to enhance prediction accuracy by tackling both non-linear trends and volatility. GARCH's proven effectiveness in modeling volatility made it a more suitable choice for this purpose. In binary predictions, where the direction of price movements is more important than the exact values, GARCH's capability to capture volatility patterns complemented the LSTM/GRU networks' trend prediction strengths. This combination resulted in high accuracy, precision, recall, and F1 scores, demonstrating the hybrid approach's effectiveness. By leveraging GARCH for volatility and LSTM/GRU for trend prediction, the model achieved a robust performance in predicting market movements, making it a valuable tool for financial decision-making.

## 5.9. IMPLICATIONS

The findings of this study have several important implications for financial forecasting. Firstly, the superior performance of LSTM and hybrid models underscores the value of using advanced machine learning techniques for financial time series prediction. These models excel at capturing the complex, non-linear relationships and long-term dependencies in financial data, resulting in more accurate and reliable predictions.

Secondly, the success of the hybrid modeling approach highlights the advantages of combining different techniques to harness their unique strengths. The hybrid LSTM/GRU-GARCH model was particularly effective in capturing both trends and volatility patterns, offering a comprehensive solution for financial forecasting. This method can be applied to other financial applications, such as risk management and option pricing, where multiple factors influence outcomes.

Lastly, the effectiveness of the binary prediction model indicates that simplifying the prediction task to focus on the direction of price movements can produce reliable and easy-to-interpret results. This approach is especially beneficial for trading strategies that require quick and accurate decisions based on market trends.

# 6. CONCLUSION AND FUTURE OUTLOOK

## 6.1. CONCLUSION

This dissertation set out to explore and evaluate various advanced machine learning models for predicting the closing prices and trends of the S&P 500. The models investigated included ARIMA, GARCH, Copula, LSTM, and hybrid models that combined LSTM/GRU with GARCH. The primary objective was to examine the strengths and limitations of these models in capturing the complex dynamics of financial time series data, which are characterised by non-linearities, volatility clustering, and long-term dependencies. Through this comprehensive analysis, the goal was to identify the most effective model or combination of models for financial forecasting and to develop a practical tool for trading strategies.

### 6.1.1. Key Insights and Achievements

Throughout this study, several key insights and achievements were made, providing a deep understanding of the models' performances and their applicability to financial forecasting.

#### 6.1.1.1. Limitations of Traditional Models

The ARIMA model, though simple and easy to interpret, showed significant limitations in predicting stock prices due to its linear assumptions and requirement for data stationarity. Its high test MSE and MAPE values underscored its inadequacy in handling the non-linear and volatile nature of financial time series data. Similarly, the GARCH model, while effective at modeling volatility clustering, struggled with making precise percentage-based predictions. The Copula model, despite its strength in capturing dependencies between multiple variables, faced challenges during periods of high volatility. It was less effective in accurately predicting price movements during these turbulent times, limiting its utility in highly volatile markets. These traditional models served as useful benchmarks but highlighted the necessity for more advanced approaches that can better capture the complexities of financial data.

#### 6.1.1.2. Superiority of Advanced Machine Learning Models

The LSTM model markedly outperformed traditional statistical models by effectively capturing the non-linear patterns and long-term dependencies inherent in financial time series data. Its lower MSE and MAPE values indicated a superior ability to generalise to new data, making it highly effective for financial forecasting. The LSTM model's capacity to manage non-linearities and retain useful information over extended sequences made it

particularly well-suited for predicting stock prices. The hybrid model that combines LSTM/GRU with GARCH achieved the best performance metrics among all the models evaluated. This approach harnessed the strengths of LSTM/GRU in capturing non-linear trends and long-term dependencies, along with GARCH's expertise in modeling volatility clustering. As a result, the hybrid model demonstrated high accuracy and robustness. Its ability to effectively capture both trends and volatility patterns made it the most accurate choice for continuous price prediction.

#### 6.1.1.3. Best Model With Real-World Application

The binary trend prediction model stood out as the best overall due to its simplicity, reliability, and ease of interpretation. It also addresses some previously mentioned research gaps. By focusing on predicting the direction of price movements (up/down) rather than exact prices, it simplified the decision-making process and reduced computational complexity. The model achieved an accuracy of 73.93%, a precision of 74.10%, a recall of 80.39%, and an F1 score of 77.12%. These metrics demonstrate that the model is highly reliable for capturing market trends, making correct predictions about three-quarters of the time. By focusing on the direction of price movements, the binary model reduces the complexity associated with predicting exact price values. This simplification makes it easier to develop and implement trading strategies, as traders can rely on directional signals rather than precise forecasts. The binary model's focus on predicting price movement direction makes it particularly useful for developing trading strategies. Its high accuracy and ease of implementation allow traders to make quick, informed decisions based on reliable signals, enhancing trading efficiency and profitability.

### 6.1.2. Overall

In conclusion, this dissertation has shown that advanced machine learning models, particularly LSTM and hybrid models, outperform traditional statistical models in predicting stock market prices. The hybrid models, in particular, excel at capturing both linear and non-linear trends, as well as volatility patterns, providing a comprehensive solution for financial forecasting. Nonetheless, the binary prediction model stands out as the most practical and effective tool for real-world trading applications. Its simplicity, reliability, and ease of implementation make it highly suitable for developing robust trading strategies. This model addresses key gaps in financial forecasting research, offering a valuable contribution to the field.

## 6.2. FUTURE OUTLOOK

### 6.2.1. Advances in Financial Time Series Forecasting

The exploration and application of advanced machine learning models for financial time series forecasting have significantly advanced over the past decade. This study demonstrates how LSTM, GRU, and hybrid models that combine these deep learning techniques with traditional statistical models like GARCH have paved the way for more accurate and robust stock market predictions [4]. As computational power continues to grow and more sophisticated algorithms are developed, the predictive capabilities of these models are expected to improve even further. Integrating more diverse and complex datasets, such as alternative data sources like social media

sentiment, news analytics, and real-time economic indicators, can provide additional insights that traditional datasets alone cannot offer [23]. This multi-dimensional approach to data collection and analysis is likely to become more common, resulting in more precise and contextually aware forecasting models [23].

### 6.2.2. Emerging Trends in Machine Learning Applications

One of the most promising trends is the increasing use of ensemble learning techniques, which combine the strengths of various models to enhance overall performance [26, 24]. The hybrid model explored in this dissertation exemplifies the effectiveness of such approaches. Future research could delve into even more advanced ensemble methods, potentially incorporating other deep learning architectures like transformers, which have shown outstanding results in natural language processing tasks [7]. Additionally, there is a growing interest in explainable AI (XAI) techniques, which aim to clarify the decision-making processes of complex models [23]. This transparency is vital for gaining the trust of stakeholders in the financial industry, who depend on these models for critical investment decisions. Developing models that not only achieve high performance but also provide clear explanations for their predictions will be a crucial focus area [23, 7].

### 6.2.3. Real-World Applications

The models developed in this study have practical applications that go well beyond academic exercises. In the real world, these models can be used for various purposes, including automated trading systems, risk management, and portfolio optimisation [26]. The binary prediction model, in particular, is a standout tool for creating straightforward and effective trading strategies. Its simplicity and high performance make it an appealing choice for traders and financial analysts who want to implement algorithmic trading strategies with clear buy or sell signals. Moreover, the adoption of these advanced models by financial institutions could lead to more efficient markets. By leveraging sophisticated predictive analytics, these institutions can make more informed decisions, potentially reducing market volatility and enhancing overall market stability [26].

### 6.2.4. Future research

While this study has made significant progress in enhancing stock market prediction models, several areas remain open for future research. One promising direction is exploring transfer learning in financial forecasting. Transfer learning, which involves pre-training a model on one task before fine-tuning it on another, could leverage knowledge from different but related financial markets or instruments [7, 21]. Another key area is the incorporation of real-time data streaming. Most existing models, including those in this study, rely on historical data. Developing models capable of processing and analysing real-time data would greatly enhance their practical utility, providing immediate and actionable insights [62, 4]. Additionally, there is potential to further refine feature selection techniques. While this study utilised a random forest classifier for feature importance ranking, other methods, such as genetic algorithms or advanced neural network-based feature selectors, could potentially yield better results [20, 26].

### 6.2.5. Ethical Considerations

As financial markets increasingly depend on machine learning models, it is crucial to address ethical considerations and ensure responsible AI practices. This involves safeguarding data privacy, preventing biased decision-making, and establishing robust regulatory frameworks to oversee the deployment of these technologies. Future research should focus on developing fair and unbiased models, especially given the significant impact financial predictions can have on individuals and economies.

### 6.2.6. Overall

In conclusion, the future of financial time series forecasting hinges on the ongoing integration of advanced machine learning techniques, diverse data sources, and sophisticated ensemble methods. The models developed and analysed in this thesis, especially the binary prediction model, represent significant progress in this field. As these technologies continue to evolve, they are expected to offer even greater accuracy and reliability, leading to more informed financial decision-making and more efficient markets. Embracing these innovations while addressing ethical and practical challenges will be crucial for the sustained advancement of financial forecasting. The binary prediction model, with its simplicity and high accuracy, stands out as a particularly practical and effective tool for real-world trading applications, filling key gaps in financial forecasting research and making a valuable contribution to the field.

# BIBLIOGRAPHY


[1] E. F. Fama, Efficient capital markets: A review of theory and empirical work, Journal of Finance 25 (2) (1970) 383–417. doi:10.2307/2325486.

[2] B. G. Malkiel, A Random Walk Down Wall Street, W.W. Norton & Company, 1973.

[3] G. E. Box, G. M. Jenkins, Time Series Analysis: Forecasting and Control, Holden-Day, 1970.

[4] T. Bollerslev, Generalized autoregressive conditional heteroskedasticity, Journal of Econometrics 31 (3) (1986) 307–327. doi:10.1016/0304-4076(86)90063-1.

[5] C. Cortes, V. Vapnik, Support-vector networks, Machine learning 20 (3) (1995) 273–297. doi:10.1007/BF00994018.

[6] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (8) (1997) 1735–1780. doi:10.1162/neco.1997.9.8.1735.

[7] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.

[8] G. Zhang, Time series forecasting using a hybrid arima and neural network model, Neurocomputing 50 (2003) 159–175. doi:10.1016/S0925-2312(01)00702-0.

[9] I. Virtanen, P. Yli-Olli, Forecasting finnish stock market returns: The role of fundamental variables, Empirical Economics 29 (1) (2004) 1–17.

[10] D. A. Dickey, W. A. Fuller, Distribution of the estimators for autoregressive time series with a unit root, Journal of the American Statistical Association 74 (366) (1979) 427–431. doi:10.1080/01621459.1979.10482531.

[11] H. Akaike, A new look at the statistical model identification, IEEE Transactions on Automatic Control 19 (6) (1974) 716–723. doi:10.1109/TAC.1974.1100705.

[12] G. Schwarz, Estimating the dimension of a model, Annals of Statistics 6 (2) (1978) 461–464. doi:10.1214/aos/1176344136.

[13] T. E. Clark, K. D. West, Forecasting stock prices using arima models: The case of nigeria and nyse, Journal of Financial and Quantitative Analysis 42 (1) (2007) 39–67.

[14] R. F. Engle, Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation, Econometrica: Journal of the Econometric Society (1982) 987–1007doi:10.2307/1912773.

[15] M. Sklar, Fonctions de répartition à n dimensions et leurs marges, Publications de l'Institut de Statistique de l'Université de Paris 8 (1959) 229–231.

[16] A. J. Patton, Modelling asymmetric exchange rate dependence, International Economic Review 47 (2) (2006) 527–556. doi:10.1111/j.1468-2354.2006.00387.x.

[17] C. Ning, Dependence structure between the equity market and the foreign exchange market—a copula approach, Journal of International Money and Finance 29 (5) (2010) 743–759. doi:10.1016/j.jimonfin.2009.12.002.

[18] D. Creal, S. J. Koopman, A. Lucas, Generalized autoregressive score models with applications, Journal of Applied Econometrics 28 (5) (2013) 777–795. doi:10.1002/jae.1279.

[19] A. Zhang, Z. C. Lipton, M. Li, A. J. Smola, 10.1. long short-term memory (lstm), https://d2l.ai/chapter_recurrent-modern/lstm.html, accessed: 2024-06-19 (2020).

[20] W. Dai, L. Zhang, X. Zhang, A novel hybrid stock price forecasting model: Incorporating lstm with multiple additional factors, Journal of Computational and Applied Mathematics 376 (2020) 112773. doi:10.1016/j.cam.2020.112773.

[21] T. Fischer, C. Krauss, Deep learning with long short-term memory networks for financial market predictions, European Journal of Operational Research 270 (2) (2018) 654–669. doi:10.1016/j.ejor.2017.11.054.

[22] Y. Chen, H. Dai, C.-Y. Chen, L. Wang, K. Liu, Stock market prediction using convolutional neural networks, Journal of Finance and Data Science 4 (1) (2018) 37–52. doi:10.1016/j.jfds.2017.12.001.

[23] T.-T. Ho, Y. Huang, Stock price movement prediction using sentiment analysis and candlestick chart representation, Sensors 21 (23) (2021) 7957.
URL https://doi.org/10.3390/s21237957

[24] Y. Zhang, P. Li, J. Wang, Advancements in hybrid models for stock market forecasting: Arima-lstm and garch-lstm, Journal of Financial Markets 58 (2023) 101354.

[25] W. Kristjanpoller, M. Minutolo, S. Solari, Forecasting volatility: Garch models versus artificial neural networks, International Journal of Forecasting 30 (3) (2014) 621–636.

[26] F. Wen, X. Yang, X. Zhou, A hybrid cnn-lstm model for predicting stock prices, Journal of Computational and Applied Mathematics 376 (2020) 112820.

[27] Y. Finance, S&p 500 index data, https://finance.yahoo.com/quote/%5EGSPC/history, accessed: 2024/07/01 (2024).

[28] Quandl, Quandl api, https://www.quandl.com/tools/api, accessed: 2024/07/01 (2024).

[29] P. J. Brockwell, R. A. Davis, Introduction to time series and forecasting, Springer Texts in StatisticsThis text introduces time series forecasting and the theoretical underpinnings of the ARIMA model. (2002). doi:10.1007/b97391.

[30] R. J. Hyndman, G. Athanasopoulos, Forecasting: Principles and Practice, 2nd Edition, OTexts, 2018, this book offers a practical guide to conducting time series analysis with a strong focus on the application of ARIMA models. Available online: https://otexts.com/fpp2/.
URL https://otexts.com/fpp2/

[31] G. E. Box, G. M. Jenkins, G. C. Reinsel, G. M. Ljung, Time Series Analysis: Forecasting and Control, 5th Edition, Wiley, 2015, this seminal book provides a comprehensive introduction to time series forecasting, including ARIMA modeling.

[32] A. Vidhya, Building an arima model for time series forecasting in python, https://www.analyticsvidhya.com/blog/2020/10/how-to-create-an-arima-model-for-time-series-forecasting-in-python/, accessed: 2024-07-12 (2020).

[33] S. Prabhakaran, Arima model - complete guide to time series forecasting in python, https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/, accessed: 2024-07-12 (2021).

[34] J. into Data, Time series forecasting archives, https://www.justintodata.com/tag/time-series-forecasting/, accessed: 2024-07-12 (2024).

[35] A. A. Awan, Time series analysis: Arima models in python, https://www.kdnuggets.com/2023/08/times-series-analysis-arima-models-python.html, accessed: 2024-07-12 (2020).

[36] C. Brooks, Introductory Econometrics for Finance, Cambridge University Press, 2014.

[37] R. S. Tsay, Analysis of Financial Time Series, Wiley-Interscience, 2005.

[38] G. James, D. Witten, T. Hastie, R. Tibshirani, An Introduction to Statistical Learning, Springer, 2013.

[39] J. D. Hamilton, Time Series Analysis, Princeton University Press, 1994.

[40] Q. Team, Forecasting financial time series - part i, QuantStart, accessed on July 12, 2024 (2023).
URL https://www.quantstart.com/articles/Forecasting-Financial-Time-Series-Part-1/

[41] K. Sheppard, Arch: A python library for financial risk modelling, https://arch.readthedocs.io (2022).

[42] M. AI, Python for finance: Time series analysis, MLQ AI Blog, accessed on July 12, 2024 (2023).
URL https://blog.mlq.ai/tag/python-for-finance/

[43] F. Initiative, Time series for finance, GitHub, accessed on July 12, 2024 (2023).
URL https://github.com/freestackinitiative/time_series_for_finance

[44] G. Contributors, Garch models on github, GitHub, accessed on July 12, 2024 (2023).
URL https://github.com/topics/garch-models

[45] J. Y. Campbell, A. W. Lo, A. C. MacKinlay, The Econometrics of Financial Markets, Princeton University Press, 1997.

[46] C. Bergmeir, R. J. Hyndman, B. Koo, On the use of cross-validation for time series predictor evaluation, Information Sciences 191 (2018) 192–213.

[47] R. B. Nelsen, An Introduction to Copulas, 2nd Edition, Springer, 2006.

[48] U. Cherubini, E. Luciano, W. Vecchiato, Copula Methods in Finance, John Wiley & Sons, 2004.

[49] D. X. Li, On default correlation: A copula function approach, Journal of Fixed Income 9 (4) (2000) 43–54, this seminal paper introduces the use of Gaussian copula for the pricing of multi-name credit derivatives, highlighting the practical application of copula functions in financial modeling.

[50] P. Embrechts, A. J. McNeil, D. Straumann, High-dimensional dependence and copula theory, Encyclopedia of Quantitative FinanceThis article discusses the mathematical formulation of copulas, particularly focusing on high-dimensional applications and the use of Gaussian copulas in finance. (2009).

[51] H. Joe, Dependence Modeling with Copulas, CRC Press, 2014, this comprehensive text covers various types of copulas including Gaussian and offers insights into their properties, estimation, and simulation techniques.

[52] D. Bok, Multivariate data modelling with copulas in python, accessed: 2024-07-15 (2023).
URL https://github.com/DanielBok/copulae

[53] L. Healy, Copulas in python, accessed: 2024-07-15 (2023).
URL https://www.kaggle.com/code/liamhealy/copulas-in-python

[54] I. DataCebo, Copulas 0.11.0 documentation, accessed: 2024-07-15 (2024).
URL https://sdv.dev/Copulas/

[55] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

[56] sibeltan, Stock price prediction (s&p 500), https://github.com/sibeltan/stock_price_prediction_LSTM, accessed: 2024-07-16 (2024).

[57] Karan-D-Software, Stock price prediction using deep learning, https://github.com/Karan-D-Software/Stock-Price-Prediction-Using-Deep-Learning, accessed: 2024-07-16 (2024).

[58] DestrosCMC, Forecasting spy using lstm, https://github.com/DestrosCMC/Forecasting-SPY, accessed: 2024-07-16 (2024).

[59] 711634, Stock prediction model in python, https://github.com/711634/Stock-Prediction-Model-in-Python, accessed: 2024-07-16 (2024).

[60] A. Jaiswal, Tutorial on rnn — lstm: With implementation, https://www.analyticsvidhya.com/blog/2022/01/tutorial-on-rnn-lstm-gru-with-implementation/, accessed: 2024-07-16 (2024).

[61] tlemenestrel, A python implementation of a hybrid lstm-garch model for volatility forecasting, https://github.com/tlemenestrel/LSTM_GARCH, accessed: 2024-07-16 (2024).

[62] R. J. Shiller, From efficient markets theory to behavioral finance, Journal of Economic Perspectives 17 (1) (2003) 83–104. doi:10.1257/089533003321164967.

[63] W. McKinney, pandas documentation, https://pandas.pydata.org/pandas-docs/stable/, accessed: 2024-07-12 (2008).

[64] T. Oliphant, Numpy v1.23 manual, https://numpy.org/doc/stable/, accessed: 2024-07-12 (2006).

[65] S. Seabold, J. Perktold, Statsmodels: Statistics in python, https://www.statsmodels.org/stable/index.html, accessed: 2024-07-12 (2010).

[66] J. D. Hunter, Matplotlib: Visualization with python, https://matplotlib.org/stable/index.html, accessed: 2024-07-12 (2003).

[67] F. P. et al., Scikit-learn: Machine learning in python, https://scikit-learn.org/stable/index.html, accessed: 2024-07-12 (2007).

[68] R. Aroussi, yfinance documentation, https://pypi.org/project/yfinance/, accessed: 2024-07-12 (2019).

[69] I. Quandl, Quandl python module, https://www.quandl.com/tools/python, accessed: 2024-07-12 (2011).

[70] K. Sheppard, Arch: Arch models in python, https://pypi.org/project/arch/, accessed: 2024-07-12 (2015).

[71] M. D. T. A. Lab, Copulas: Modeling multivariate distributions and sampling using copulas, https://pypi.org/project/copulas/, accessed: 2024-07-15 (2024).

[72] Tensorflow: Large-scale machine learning on heterogeneous systems, https://www.tensorflow.org/, software available from tensorflow.org (2015).

# APPENDIX

# APPENDIX A

### 6.2.7. ARIMA without stationarity

```
import pandas as pd
import numpy as np
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
import yfinance as yf
import datetime
import quandl
from sklearn.metrics import mean_squared_error
import warnings

# Suppress warnings
warnings.filterwarnings('ignore')

# Function to calculate Mean Absolute Percentage Error (MAPE)
def calculate_mape(y_actual, y_predicted):
    y_actual, y_predicted = np.array(y_actual), np.array(y_predicted)
    if np.any(y_actual == 0):
        y_actual[y_actual == 0] = np.nan  # Handle zero values to avoid division by zero
    return np.mean(np.abs((y_actual - y_predicted) / y_actual)) * 100

# Function to perform grid search for optimal ARIMA parameters
def arima_grid_search(training_data, p_values, d_values, q_values):
    optimal_score, optimal_params = float('inf'), None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                try:
                    arima_model = ARIMA(training_data, order=(p, d, q))
                    arima_fit = arima_model.fit()
                    aic_value = arima_fit.aic
                    if aic_value < optimal_score:
                        optimal_score, optimal_params = aic_value, (p, d, q)
```

```
                    print(f'ARIMA({p},{d},{q}) AIC: {aic_value}')
                except:
                    continue
    print(f'Optimal ARIMA parameters: {optimal_params} with AIC: {optimal_score}')
    return optimal_params

# Set Quandl API key
quandl.ApiConfig.api_key = '7BtjxPppBWXTh3QD_Bks'

# Define the S&P 500 ticker symbol
symbol = "^GSPC"

# Define the date range for data extraction
start_date = datetime.datetime(2018, 1, 1)
end_date = datetime.datetime(2024, 1, 1)

# Download S&P 500 data using Yahoo Finance
sp500_data = yf.download(symbol, start=start_date, end=end_date)
sp500_data.index = pd.to_datetime(sp500_data.index)
sp500_data = sp500_data.asfreq('B')  # Business day frequency

# Preprocess data by forward-filling missing values
sp500_data.ffill(inplace=True)

# Retrieve additional macroeconomic indicators from Quandl
cpi_data = quandl.get("FRED/CPIAUCSL", start_date=start_date, end_date=end_date)
gdp_data = quandl.get("FRED/GDP", start_date=start_date, end_date=end_date)
interest_rate_data = quandl.get("FRED/DFF", start_date=start_date, end_date=end_date)
unemployment_data = quandl.get("FRED/UNRATE", start_date=start_date, end_date=end_date)

# Rename columns for clarity
cpi_data.rename(columns={'Value': 'CPI'}, inplace=True)
gdp_data.rename(columns={'Value': 'GDP'}, inplace=True)
interest_rate_data.rename(columns={'Value': 'InterestRate'}, inplace=True)
unemployment_data.rename(columns={'Value': 'UnemploymentRate'}, inplace=True)

# Resample and forward-fill data
cpi_data = cpi_data.resample('B').ffill()
gdp_data = gdp_data.resample('B').ffill()
interest_rate_data = interest_rate_data.resample('B').ffill()
unemployment_data = unemployment_data.resample('B').ffill()

# Merge all datasets
sp500_data = sp500_data.merge(cpi_data, how='left', left_index=True, right_index=True)
sp500_data = sp500_data.merge(gdp_data, how='left', left_index=True, right_index=True)
sp500_data = sp500_data.merge(interest_rate_data, how='left', left_index=True, right_index=
    True)
sp500_data = sp500_data.merge(unemployment_data, how='left', left_index=True, right_index=True
    )
```

```
sp500_data.ffill(inplace=True)  # Forward-fill any remaining missing values

# Normalize features excluding the 'Close' price
scaler = MinMaxScaler(feature_range=(0, 1))
columns_to_normalize = ['Open', 'High', 'Low', 'Volume', 'CPI', 'GDP', 'InterestRate', '
    UnemploymentRate']
sp500_data[columns_to_normalize] = scaler.fit_transform(sp500_data[columns_to_normalize])

# Split data into training and testing sets (80% training, 20% testing)
train_length = int(len(sp500_data) * 0.8)
train_data, test_data = sp500_data[:train_length], sp500_data[train_length:]

# Define the parameter grid for ARIMA model
p_values = range(0, 6)
d_values = range(0, 1)
q_values = range(0, 6)

# Perform grid search for best ARIMA parameters
optimal_params = arima_grid_search(train_data['Close'], p_values, d_values, q_values)

# Fit ARIMA model with optimal parameters
best_arima_model = ARIMA(train_data['Close'], order=optimal_params)
best_arima_fit = best_arima_model.fit()

# Make predictions on training data
train_predictions = best_arima_fit.predict(start=train_data.index[0], end=train_data.index
    [-1])
train_mse = mean_squared_error(train_data['Close'], train_predictions)
train_mape = calculate_mape(train_data['Close'], train_predictions)

# Forecast on test data using the fitted ARIMA model
test_predictions = best_arima_fit.predict(start=test_data.index[0], end=test_data.index[-1])
test_mse = mean_squared_error(test_data['Close'], test_predictions)
test_mape = calculate_mape(test_data['Close'], test_predictions)

# Plot actual vs. predicted closing prices
plt.figure(figsize=(12, 6))
plt.plot(sp500_data.index[train_length:], test_data['Close'], color='blue', label='Actual
    Close Price')
plt.plot(sp500_data.index[train_length:], test_predictions, color='red', label='Predicted
    Close Price')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('S&P 500 Close Price Prediction using ARIMA')
plt.legend()
plt.show()

# Create a DataFrame to compare actual and predicted values
results_df = pd.DataFrame({'Actual': test_data['Close'].values, 'Predicted': test_predictions
```

```
    }, index=sp500_data.index[train_length:])
print(results_df.head())
print(f"\nTraining MSE: {train_mse}, Training MAPE: {train_mape}%")
print(f"Test MSE: {test_mse}, Test MAPE: {test_mape}%")
```

**Listing 1:** ARIMA Python Code for S&P 500 Prediction

### 6.2.8. ARIMA with stationarity

```
import pandas as pd
import numpy as np
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
import yfinance as yf
import datetime
import quandl
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
import warnings

# Suppress warnings
warnings.filterwarnings('ignore')

# Function to compute Mean Absolute Percentage Error (MAPE)
def calculate_mape(actual, predicted):
    actual, predicted = np.array(actual), np.array(predicted)
    if np.any(actual == 0):
        actual[actual == 0] = np.nan  # Handle zero values to avoid division by zero
    return np.mean(np.abs((actual - predicted) / actual)) * 100

# Function for grid search to find optimal ARIMA parameters
def perform_arima_grid_search(training_data, p_values, d_values, q_values):
    lowest_aic, optimal_params = float('inf'), None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                try:
                    model = ARIMA(training_data, order=(p, d, q))
                    model_fit = model.fit()
                    aic_score = model_fit.aic
                    if aic_score < lowest_aic:
                        lowest_aic, optimal_params = aic_score, (p, d, q)
                    print(f'ARIMA({p},{d},{q}) AIC: {aic_score}')
                except:
                    continue
    print(f'Optimal ARIMA parameters: {optimal_params} with AIC: {lowest_aic}')
    return optimal_params
```

```
41
42 # Function to perform the Augmented Dickey-Fuller (ADF) test
43 def perform_adf_test(series, title=''):
44     print(f'ADF Test Results: {title}')
45     adf_result = adfuller(series, autolag='AIC')
46     adf_output = pd.Series(adf_result[:4], index=['ADF Statistic', 'p-value', '# of Lags', '
       Observations Used'])
47     for key, value in adf_result[4].items():
48         adf_output[f'Critical Value ({key})'] = value
49     print(adf_output.to_string())
50     if adf_result[1] <= 0.05:
51         print("Strong evidence against the null hypothesis, indicating stationarity.")
52     else:
53         print("Weak evidence against the null hypothesis, indicating non-stationarity.")
54
55 # Set Quandl API key
56 quandl.ApiConfig.api_key = '7BtjxPppBWXTh3QD_Bks'
57
58 # Define S&P 500 ticker symbol and date range
59 ticker_symbol = "^GSPC"
60 start_date = datetime.datetime(2018, 1, 1)
61 end_date = datetime.datetime(2024, 1, 1)
62
63 # Download S&P 500 data from Yahoo Finance
64 sp500_data = yf.download(ticker_symbol, start=start_date, end=end_date)
65 sp500_data.index = pd.to_datetime(sp500_data.index)
66 sp500_data = sp500_data.asfreq('B')  # Ensure data is at business day frequency
67
68 # Preprocess data by forward-filling missing values
69 sp500_data.ffill(inplace=True)
70
71 # Fetch additional macroeconomic data from Quandl
72 cpi_data = quandl.get("FRED/CPIAUCSL", start_date=start_date, end_date=end_date)
73 gdp_data = quandl.get("FRED/GDP", start_date=start_date, end_date=end_date)
74 interest_rate_data = quandl.get("FRED/DFF", start_date=start_date, end_date=end_date)
75 unemployment_data = quandl.get("FRED/UNRATE", start_date=start_date, end_date=end_date)
76
77 # Rename columns for clarity
78 cpi_data.rename(columns={'Value': 'CPI'}, inplace=True)
79 gdp_data.rename(columns={'Value': 'GDP'}, inplace=True)
80 interest_rate_data.rename(columns={'Value': 'InterestRate'}, inplace=True)
81 unemployment_data.rename(columns={'Value': 'UnemploymentRate'}, inplace=True)
82
83 # Resample and forward-fill the data
84 cpi_data = cpi_data.resample('B').ffill()
85 gdp_data = gdp_data.resample('B').ffill()
86 interest_rate_data = interest_rate_data.resample('B').ffill()
87 unemployment_data = unemployment_data.resample('B').ffill()
88
```

```
# Merge all datasets into a single DataFrame
sp500_data = sp500_data.merge(cpi_data, how='left', left_index=True, right_index=True)
sp500_data = sp500_data.merge(gdp_data, how='left', left_index=True, right_index=True)
sp500_data = sp500_data.merge(interest_rate_data, how='left', left_index=True, right_index=
    True)
sp500_data = sp500_data.merge(unemployment_data, how='left', left_index=True, right_index=True
    )
sp500_data.ffill(inplace=True)  # Forward-fill any remaining missing values

# Normalize features excluding the 'Close' price
scaler = MinMaxScaler(feature_range=(0, 1))
features_to_normalize = ['Open', 'High', 'Low', 'Volume', 'CPI', 'GDP', 'InterestRate', '
    UnemploymentRate']
sp500_data[features_to_normalize] = scaler.fit_transform(sp500_data[features_to_normalize])

# Perform ADF test on the original 'Close' price series
perform_adf_test(sp500_data['Close'], 'Original Close Prices')

# Decompose the series to remove seasonality
seasonal_decomposition = seasonal_decompose(sp500_data['Close'], model='multiplicative',
    period=252)  # Assumes daily data with an annual cycle
seasonally_adjusted_series = sp500_data['Close'] / seasonal_decomposition.seasonal

# Differencing to achieve stationarity
seasonally_adjusted_diff = seasonally_adjusted_series.diff().dropna()

# Perform ADF test on the seasonally adjusted and differenced data
perform_adf_test(seasonally_adjusted_diff, 'Seasonally Adjusted and Differenced Close Prices')

# Visualize the seasonally adjusted and differenced series
plt.figure(figsize=(12, 6))
plt.plot(seasonally_adjusted_diff, label='Seasonally Adjusted and Differenced Close Prices')
plt.title('Seasonally Adjusted and Differenced S&P 500 Close Prices')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()

# Split the dataset into training and testing sets (80/20 split)
train_size = int(len(seasonally_adjusted_diff) * 0.8)
train_data_diff, test_data_diff = seasonally_adjusted_diff[:train_size],
    seasonally_adjusted_diff[train_size:]

# Define grid search parameters for ARIMA model
p_values = range(0, 6)
d_values = range(0, 3)
q_values = range(0, 6)

# Perform grid search on the differenced data to find the best ARIMA parameters
```

```
best_arima_params_diff = perform_arima_grid_search(train_data_diff, p_values, d_values,
    q_values)

# Fit ARIMA model with the best parameters on the differenced data
arima_model_diff = ARIMA(train_data_diff, order=best_arima_params_diff)
arima_fit_diff = arima_model_diff.fit()

# Make predictions on the training data
train_predictions_diff = arima_fit_diff.predict(start=train_data_diff.index[0], end=
    train_data_diff.index[-1])
train_mse_diff = mean_squared_error(train_data_diff, train_predictions_diff)
train_mape_diff = calculate_mape(train_data_diff, train_predictions_diff)

# Forecast on the testing set
test_predictions_diff = arima_fit_diff.predict(start=test_data_diff.index[0], end=
    test_data_diff.index[-1])
test_mse_diff = mean_squared_error(test_data_diff, test_predictions_diff)
test_mape_diff = calculate_mape(test_data_diff, test_predictions_diff)

# Revert the differenced predictions back to the original scale
inverted_predictions_diff = test_predictions_diff.cumsum() + seasonally_adjusted_series.iloc[
    train_size - 1]

# Reapply the seasonal component
final_predictions = inverted_predictions_diff * seasonal_decomposition.seasonal.iloc[
    train_size:]

# Plot actual vs. predicted close prices
plt.figure(figsize=(12, 6))
plt.plot(sp500_data.index[train_size:], sp500_data['Close'].iloc[train_size:], color='blue',
    label='Actual Close Price')
plt.plot(sp500_data.index[train_size:], final_predictions, color='red', label='Predicted Close
     Price')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('S&P 500 Close Price Prediction using Seasonally Adjusted ARIMA')
plt.legend()
plt.show()

# Create DataFrame to compare actual and predicted values
results_df_diff = pd.DataFrame({'Actual': sp500_data['Close'].iloc[train_size:].to_numpy(), '
    Predicted': final_predictions}, index=sp500_data.index[train_size:])
print(results_df_diff.head())
print(f"\nTraining MSE (Differenced): {train_mse_diff}, Training MAPE (Differenced): {
    train_mape_diff}%")
print(f"Test MSE (Differenced): {test_mse_diff}, Test MAPE (Differenced): {test_mape_diff}%")
```

**Listing 2:** ARIMA (stationarity) Python Code for S&P 500 Prediction

### 6.2.9. GARCH (Predicting Returns)

```
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from arch import arch_model
from sklearn.model_selection import train_test_split
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import quandl
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error

# Set up Quandl API key
quandl.ApiConfig.api_key = '7BtjxPppBWXTh3QD_Bks'

# Download historical S&P 500 data
ticker = '^GSPC'
start_date = '2018-01-01'
end_date = '2024-01-01'
data = yf.download(ticker, start=start_date, end=end_date)

# Fetch additional macroeconomic indicators from Quandl
cpi = quandl.get("FRED/CPIAUCSL", start_date=start_date, end_date=end_date)
gdp = quandl.get("FRED/GDP", start_date=start_date, end_date=end_date)
interest_rate = quandl.get("FRED/DFF", start_date=start_date, end_date=end_date)
unemployment_rate = quandl.get("FRED/UNRATE", start_date=start_date, end_date=end_date)

# Merge all datasets into one DataFrame
data = data[['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']]
data = data.join(cpi, how='left').rename(columns={'Value': 'CPI'})
data = data.join(gdp, how='left').rename(columns={'Value': 'GDP'})
data = data.join(interest_rate, how='left').rename(columns={'Value': 'Interest Rate'})
data = data.join(unemployment_rate, how='left').rename(columns={'Value': 'Unemployment Rate'})

# Handle missing values by forward filling
data.fillna(method='ffill', inplace=True)

# Calculate log returns of the adjusted close prices
data['Return'] = np.pad(np.diff(np.log(data['Adj Close'])) * 100, (1, 0), 'constant',
    constant_values=np.nan)
data.dropna(inplace=True)

# Plot the returns over time
plt.figure(figsize=(8, 5))
plt.plot(data['Return'])
plt.ylabel("Return %")
plt.title('S&P 500 Returns')
plt.show()

```

```
# Autocorrelation and partial autocorrelation plots for returns
returns_series = data['Return']

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))
plot_acf(returns_series, ax=ax1, lags=10)
ax1.set_ylim(-0.5, 0.5)
ax1.set_title("Autocorrelation of Returns")
plot_pacf(returns_series, ax=ax2, lags=10)
ax2.set_ylim(-0.5, 0.5)
ax2.set_xlabel("Lag")
ax2.set_title("Partial Autocorrelation of Returns")
plt.show()

# Split the data into training and testing sets (80/20 split)
train_data, test_data = train_test_split(returns_series, train_size=0.8, shuffle=False)

# Fit a GARCH(1,1) model to the training data
garch_model = arch_model(train_data, mean="Zero", vol='Garch', p=1, q=1, rescale=False)
garch_fit = garch_model.fit(disp='off')

# Forecast on the test set
forecasts = garch_fit.forecast(horizon=test_data.shape[0], reindex=False)
train_volatility = garch_fit.conditional_volatility

# Calculate MSE and MAPE for training and testing sets
train_mse = mean_squared_error(train_data, train_volatility)
test_mse = mean_squared_error(test_data, np.sqrt(forecasts.variance.values[-1, :]))
train_mape = mean_absolute_percentage_error(train_data, train_volatility)
test_mape = mean_absolute_percentage_error(test_data, np.sqrt(forecasts.variance.values[-1,
    :]))

print(f"Training MSE: {train_mse}")
print(f"Test MSE: {test_mse}")
print(f"Training MAPE: {train_mape}")
print(f"Test MAPE: {test_mape}")

# Plot actual returns against predicted volatility
fig, ax = plt.subplots(figsize=(10, 8))
ax.spines[['top', 'right']].set_visible(False)
plt.plot(returns_series[-test_data.shape[0]:])
plt.plot(test_data.index, np.sqrt(forecasts.variance.values[-1, :]))
plt.title('Predicted Volatility vs. Actual Returns for S&P 500')
plt.legend(['Actual Daily Log Returns', 'Predicted Volatility'])
plt.show()

# Plot the conditional volatility
garch_fit.plot(annualize="D")
plt.show()
```

**Listing 3:** GARCH (Returns Prediction) Python Code for S&P 500 Prediction

### 6.2.10. GARCH (Simulating Future Prices)

```
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from arch import arch_model
from sklearn.model_selection import train_test_split
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
import quandl
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error

# Set the Quandl API key
quandl.ApiConfig.api_key = '7BtjxPppBWXTh3QD_Bks'

# Download historical S&P 500 data
ticker = '^GSPC'
start_date = '2018-01-01'
end_date = '2024-01-01'
data = yf.download(ticker, start=start_date, end=end_date)

# Retrieve additional economic indicators from Quandl
cpi = quandl.get("FRED/CPIAUCSL", start_date=start_date, end_date=end_date)
gdp = quandl.get("FRED/GDP", start_date=start_date, end_date=end_date)
interest_rate = quandl.get("FRED/DFF", start_date=start_date, end_date=end_date)
unemployment_rate = quandl.get("FRED/UNRATE", start_date=start_date, end_date=end_date)

# Merge the fetched datasets with the main data
data = data[['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']]
data = data.join(cpi, how='left').rename(columns={'Value': 'CPI'})
data = data.join(gdp, how='left').rename(columns={'Value': 'GDP'})
data = data.join(interest_rate, how='left').rename(columns={'Value': 'Interest Rate'})
data = data.join(unemployment_rate, how='left').rename(columns={'Value': 'Unemployment Rate'})

# Handle missing values by forward filling
data.fillna(method='ffill', inplace=True)

# Calculate logarithmic returns of the adjusted close prices
data['Log Return'] = np.log(data['Adj Close']).diff().dropna()

# Conduct an Augmented Dickey-Fuller test to check for stationarity
adf_stat, p_value, _, _, critical_values, _ = adfuller(data['Log Return'])
print(f'ADF Statistic: {adf_stat}')
print(f'p-value: {p_value}')

if p_value > 0.05:
    print("Warning: Log returns may not be stationary. Consider differencing or applying other
```

```
    transformations.")

# Normalize selected economic indicators
scaler = MinMaxScaler()
data[['CPI', 'GDP', 'Interest Rate', 'Unemployment Rate']] = scaler.fit_transform(data[['CPI',
    'GDP', 'Interest Rate', 'Unemployment Rate']])

# Plot the log returns
plt.figure(figsize=(8, 5))
plt.plot(data['Log Return'])
plt.ylabel("Log Return")
plt.title('Log Returns of S&P 500')
plt.show()

# Display Autocorrelation and Partial Autocorrelation plots
log_returns = data['Log Return']

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))
plot_acf(log_returns, ax=ax1, lags=10)
ax1.set_ylim(-0.5, 0.5)
ax1.set_title("Autocorrelation of Log Returns")
plot_pacf(log_returns, ax=ax2, lags=10)
ax2.set_ylim(-0.5, 0.5)
ax2.set_xlabel("Lag")
ax2.set_title("Partial Autocorrelation of Log Returns")
plt.show()

# Split the data into training and testing sets (80/20 split)
train_log_returns, test_log_returns = train_test_split(log_returns, train_size=0.8, shuffle=
    False)

# Fit a GARCH(1,1) model on the training data
garch_model = arch_model(train_log_returns, mean="Zero", vol='Garch', p=1, q=1, rescale=False)
garch_results = garch_model.fit(disp='off')

# Forecast on the test set
garch_forecast = garch_results.forecast(horizon=len(test_log_returns), reindex=False)
predicted_volatility = garch_forecast.variance.values[-1, :] ** 0.5  # Square root of forecast
    variance to get volatility

# Generate log return predictions based on the forecasted volatility
np.random.seed(42)  # Set seed for reproducibility
predicted_log_returns = np.random.normal(loc=0, scale=predicted_volatility, size=len(
    test_log_returns))

# Predict the adjusted close prices from the predicted log returns
predicted_close_prices = [data['Adj Close'].iloc[len(train_log_returns)]]  # Start with the
    last training price
for log_return in predicted_log_returns:
```

```
    predicted_close_prices.append(predicted_close_prices[-1] * np.exp(log_return))

# Assign the predicted close prices to the corresponding test indices
data.loc[test_log_returns.index, 'Predicted Close'] = predicted_close_prices[1:]  # Exclude the initial value

# Calculate the Mean Squared Error (MSE) and Mean Absolute Percentage Error (MAPE) for training and testing sets
train_mse = mean_squared_error(train_log_returns, garch_results.conditional_volatility**2)
test_mse = mean_squared_error(data.loc[test_log_returns.index, 'Adj Close'], data.loc[test_log_returns.index, 'Predicted Close'])
train_mape = mean_absolute_percentage_error(train_log_returns, garch_results.conditional_volatility**2)
test_mape = mean_absolute_percentage_error(data.loc[test_log_returns.index, 'Adj Close'], data.loc[test_log_returns.index, 'Predicted Close'])

print(f"Training MSE: {train_mse}")
print(f"Test MSE: {test_mse}")
print(f"Training MAPE: {train_mape}")
print(f"Test MAPE: {test_mape}")

# Plot actual versus predicted closing prices
fig, ax = plt.subplots(figsize=(10, 8))
ax.spines[['top', 'right']].set_visible(False)
plt.plot(data.loc[test_log_returns.index, 'Adj Close'], label='Actual Closing Prices')
plt.plot(data.loc[test_log_returns.index, 'Predicted Close'], label='Predicted Closing Prices')
plt.title('S&P 500 Closing Price Prediction')
plt.legend()
plt.show()

# Extract standardized residuals and conditional volatility from the GARCH model
std_residuals = garch_results.std_resid
cond_volatility = garch_results.conditional_volatility

# Plot standardized residuals and conditional volatility in separate subplots
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))
ax1.plot(std_residuals)
ax1.set_title("Standardized Residuals")
ax2.plot(cond_volatility)
ax2.set_title("Conditional Volatility")
plt.tight_layout()
plt.show()
```

**Listing 4:** GARCH (Future Price Simulation) Python Code for S&P 500 Prediction

### 6.2.11. Copula

```
import yfinance as yf
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from copulas.multivariate import GaussianMultivariate
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error
import quandl

# Set Quandl API key
quandl.ApiConfig.api_key = '7BtjxPppBWXTh3QD_Bks'

# Download historical S&P 500 data
symbol = '^GSPC'
start = '2018-01-01'
end = '2024-01-01'
market_data = yf.download(symbol, start=start, end=end)

# Fetch additional economic indicators from Quandl
cpi_data = quandl.get("FRED/CPIAUCSL", start_date=start, end_date=end)
gdp_data = quandl.get("FRED/GDP", start_date=start, end_date=end)
rate_data = quandl.get("FRED/DFF", start_date=start, end_date=end)
unemployment_data = quandl.get("FRED/UNRATE", start_date=start, end_date=end)

# Merge datasets into a single DataFrame
market_data = market_data[['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']]
market_data = market_data.join(cpi_data, how='left').rename(columns={'Value': 'CPI'})
market_data = market_data.join(gdp_data, how='left').rename(columns={'Value': 'GDP'})
market_data = market_data.join(rate_data, how='left').rename(columns={'Value': 'Interest Rate'
    })
market_data = market_data.join(unemployment_data, how='left').rename(columns={'Value': '
    Unemployment Rate'})

# Handle any missing data by forward filling
market_data.fillna(method='ffill', inplace=True)

# Calculate logarithmic returns
market_data['Log Return'] = np.pad(np.diff(np.log(market_data['Adj Close'])) * 100, (1, 0), '
    constant', constant_values=np.nan)
market_data.dropna(inplace=True)

# Plot the calculated returns
plt.figure(figsize=(8, 5))
plt.plot(market_data['Log Return'])
plt.ylabel("Return %")
plt.title('S&P 500 Logarithmic Returns')
plt.show()

# Split the data into training and testing sets (80/20 split)
train_data, test_data = train_test_split(market_data['Log Return'], train_size=0.8, shuffle=
```

```
    False)

# Train a Gaussian Copula model on the training set
copula = GaussianMultivariate()
copula.fit(pd.DataFrame(train_data))

# Simulate returns for the test period using the Copula model
simulated_data = copula.sample(len(test_data))

# Evaluate the model's performance using MSE and MAPE
mse_train = mean_squared_error(train_data, copula.sample(len(train_data)))
mse_test = mean_squared_error(test_data, simulated_data)
mape_train = mean_absolute_percentage_error(train_data, copula.sample(len(train_data)))
mape_test = mean_absolute_percentage_error(test_data, simulated_data)

print(f"Training MSE: {mse_train}")
print(f"Testing MSE: {mse_test}")
print(f"Training MAPE: {mape_train}")
print(f"Testing MAPE: {mape_test}")

# Plot the actual returns versus the simulated returns for the test period
plt.figure(figsize=(10, 6))
plt.plot(test_data.index, test_data.values, label='Actual Returns')
plt.plot(test_data.index, simulated_data.values, label='Simulated Returns')
plt.legend()
plt.title('S&P 500 Return Prediction with Copula Model')
plt.show()

# Scatter plot of actual returns versus simulated returns
plt.figure(figsize=(6, 6))
plt.scatter(test_data.values, simulated_data.values, alpha=0.5)
plt.plot([min(test_data.values), max(test_data.values)], [min(test_data.values), max(test_data
    .values)], color='red')
plt.xlabel('Actual Returns')
plt.ylabel('Simulated Returns')
plt.title('Actual vs Simulated Returns')
plt.show()
```

**Listing 5:** Copula (Returns Simulation) Python Code for S&P 500 Prediction

### 6.2.12. LSTM

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import yfinance as yf
import datetime
import quandl
from sklearn.preprocessing import MinMaxScaler
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

# Function to compute Mean Absolute Percentage Error (MAPE)
def calculate_mape(actual, predicted):
    actual, predicted = np.array(actual), np.array(predicted)
    if np.any(actual == 0):
        actual[actual == 0] = np.nan  # Adjust zero values to nan to avoid division by zero
    return np.mean(np.abs((actual - predicted) / actual)) * 100

# Set the Quandl API key
quandl.ApiConfig.api_key = '7BtjxPppBWXTh3QD_Bks'

# Specify the ticker symbol for the S&P 500 index
symbol = "^GSPC"

# Define the date range for data retrieval
start_date = datetime.datetime(2018, 1, 1)
end_date = datetime.datetime(2024, 1, 1)

# Download the historical data from Yahoo Finance
sp500_df = yf.download(symbol, start=start_date, end=end_date)
sp500_df.index = pd.to_datetime(sp500_df.index)
sp500_df = sp500_df.asfreq('B')  # Set frequency to business days

# Handle missing data by forward filling
sp500_df.ffill(inplace=True)

# Retrieve additional economic indicators from Quandl
cpi_data = quandl.get("FRED/CPIAUCSL", start_date=start_date, end_date=end_date)
gdp_data = quandl.get("FRED/GDP", start_date=start_date, end_date=end_date)
interest_rate_data = quandl.get("FRED/DFF", start_date=start_date, end_date=end_date)
unemployment_data = quandl.get("FRED/UNRATE", start_date=start_date, end_date=end_date)

# Rename columns for clarity
cpi_data.rename(columns={'Value': 'CPI'}, inplace=True)
gdp_data.rename(columns={'Value': 'GDP'}, inplace=True)
interest_rate_data.rename(columns={'Value': 'InterestRate'}, inplace=True)
unemployment_data.rename(columns={'Value': 'UnemploymentRate'}, inplace=True)

# Resample the data to match the frequency of the S&P 500 data
cpi_data = cpi_data.resample('B').ffill().reindex(sp500_df.index)
gdp_data = gdp_data.resample('B').ffill().reindex(sp500_df.index)
interest_rate_data = interest_rate_data.resample('B').ffill().reindex(sp500_df.index)
unemployment_data = unemployment_data.resample('B').ffill().reindex(sp500_df.index)

# Combine all datasets into one DataFrame
```

```
sp500_df = sp500_df.join([cpi_data, gdp_data, interest_rate_data, unemployment_data])
sp500_df.ffill(inplace=True)  # Handle any remaining missing data by forward filling

# Define the features to be used
features_list = ['Open', 'High', 'Low', 'Close', 'Volume', 'CPI', 'GDP', 'InterestRate', '
    UnemploymentRate']

# Normalize the feature data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_features = scaler.fit_transform(sp500_df[features_list])

# Function to prepare data for LSTM model
def create_dataset(data, time_steps=1):
    X, y = [], []
    for i in range(len(data) - time_steps - 1):
        X.append(data[i:(i + time_steps)])
        y.append(data[i + time_steps, features_list.index('Close')])  # Use 'Close' price as
    target
    return np.array(X), np.array(y)

# Split the data into training and testing sets (80/20 split)
train_size = int(len(scaled_features) * 0.8)
test_size = len(scaled_features) - train_size
train_data, test_data = scaled_features[:train_size], scaled_features[train_size:]

time_steps = 10  # Use 10 days of past data to predict the next day's close price
X_train, y_train = create_dataset(train_data, time_steps)
X_test, y_test = create_dataset(test_data, time_steps)

# Reshape data to fit LSTM model input requirements
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2])
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2])

# Build the LSTM model
lstm_model = Sequential()
lstm_model.add(LSTM(50, return_sequences=True, input_shape=(time_steps, X_train.shape[2])))
lstm_model.add(LSTM(50, return_sequences=False))
lstm_model.add(Dropout(0.2))
lstm_model.add(Dense(25))
lstm_model.add(Dense(1))

# Compile the model
lstm_model.compile(optimizer='adam', loss='mean_squared_error')

# Define early stopping criteria
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Train the model
lstm_model.fit(X_train, y_train, batch_size=32, epochs=100, validation_split=0.1, callbacks=[
```

```
    early_stopping])

# Generate predictions
train_predictions = lstm_model.predict(X_train)
test_predictions = lstm_model.predict(X_test)

# Transform predictions back to original scale
train_predictions = scaler.inverse_transform(np.concatenate((train_predictions, np.zeros((
    train_predictions.shape[0], scaled_features.shape[1] - 1))), axis=1))[:, 0]
test_predictions = scaler.inverse_transform(np.concatenate((test_predictions, np.zeros((
    test_predictions.shape[0], scaled_features.shape[1] - 1))), axis=1))[:, 0]
y_train = scaler.inverse_transform(np.concatenate((y_train.reshape(-1, 1), np.zeros((y_train.
    shape[0], scaled_features.shape[1] - 1))), axis=1))[:, 0]
y_test = scaler.inverse_transform(np.concatenate((y_test.reshape(-1, 1), np.zeros((y_test.
    shape[0], scaled_features.shape[1] - 1))), axis=1))[:, 0]

# Evaluate the model's performance
train_mse = mean_squared_error(y_train, train_predictions)
train_mape = calculate_mape(y_train, train_predictions)
test_mse = mean_squared_error(y_test, test_predictions)
test_mape = calculate_mape(y_test, test_predictions)

# Plot the actual vs predicted close prices
plt.figure(figsize=(12, 6))
plt.plot(sp500_df.index[:train_size + time_steps + 1], scaler.inverse_transform(
    scaled_features)[:train_size + time_steps + 1, features_list.index('Close')], color='green
    ', label='Training Data')
plt.plot(sp500_df.index[train_size + time_steps + 1:], y_test, color='blue', label='Actual
    Close Price')
plt.plot(sp500_df.index[train_size + time_steps + 1:], test_predictions, color='red', label='
    Predicted Close Price')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('S&P 500 Close Price Prediction using LSTM with Additional Features')
plt.legend()
plt.show()

# Display prediction and actual values
results = pd.DataFrame({'Actual': y_test, 'Predicted': test_predictions}, index=sp500_df.index
    [train_size + time_steps + 1: train_size + time_steps + 1 + len(y_test)])
print(results.head())
print(f"\nTraining MSE: {train_mse}, Training MAPE: {train_mape}%")
print(f"Test MSE: {test_mse}, Test MAPE: {test_mape}%")
```

**Listing 6:** LSTM (Close price Prediciton) Python Code for S&P 500 Prediction

### 6.2.13. Hybrid

```
import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
import yfinance as yf
import datetime
import quandl
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, GRU, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import tensorflow as tf
from arch import arch_model

# Function to compute Mean Absolute Percentage Error (MAPE)
def calculate_mape(actual, predicted):
    actual, predicted = np.array(actual), np.array(predicted)
    if np.any(actual == 0):
        actual[actual == 0] = np.nan  # Handle zero values to avoid infinite results
    return np.mean(np.abs((actual - predicted) / actual)) * 100

# Set Quandl API key
quandl.ApiConfig.api_key = '7BtjxPppBWXTh3QD_Bks'

# Define S&P 500 ticker symbol and date range
symbol = "^GSPC"
start_date = datetime.datetime(2018, 1, 1)
end_date = datetime.datetime(2024, 1, 1)

# Download S&P 500 historical data
sp500_df = yf.download(symbol, start=start_date, end=end_date)
sp500_df.index = pd.to_datetime(sp500_df.index)
sp500_df = sp500_df.asfreq('B')  # Set data frequency to business days

# Handle missing data by forward filling
sp500_df.ffill(inplace=True)

# Fetch additional macroeconomic indicators from Quandl
cpi_data = quandl.get("FRED/CPIAUCSL", start_date=start_date, end_date=end_date)
gdp_data = quandl.get("FRED/GDP", start_date=start_date, end_date=end_date)
interest_rate_data = quandl.get("FRED/DFF", start_date=start_date, end_date=end_date)
unemployment_data = quandl.get("FRED/UNRATE", start_date=start_date, end_date=end_date)

# Rename columns for clarity
cpi_data.rename(columns={'Value': 'CPI'}, inplace=True)
gdp_data.rename(columns={'Value': 'GDP'}, inplace=True)
interest_rate_data.rename(columns={'Value': 'InterestRate'}, inplace=True)
unemployment_data.rename(columns={'Value': 'UnemploymentRate'}, inplace=True)

# Resample the macroeconomic data to match the S&P 500 data frequency
```

```
cpi_data = cpi_data.resample('D').ffill().reindex(sp500_df.index)
gdp_data = gdp_data.resample('D').ffill().reindex(sp500_df.index)
interest_rate_data = interest_rate_data.resample('D').ffill().reindex(sp500_df.index)
unemployment_data = unemployment_data.resample('D').ffill().reindex(sp500_df.index)

# Merge all datasets into a single DataFrame
sp500_df = sp500_df.join([cpi_data, gdp_data, interest_rate_data, unemployment_data])
sp500_df.ffill(inplace=True)  # Forward fill any remaining missing values

# Specify features to be included
feature_columns = ['Open', 'High', 'Low', 'Close', 'Volume', 'CPI', 'GDP', 'InterestRate', '
    UnemploymentRate']

# Normalize the feature data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_features = scaler.fit_transform(sp500_df[feature_columns])

# Function to prepare the dataset for the LSTM model
def prepare_dataset(data, time_steps=1):
    X, y = [], []
    for i in range(len(data) - time_steps - 1):
        X.append(data[i:(i + time_steps)])
        y.append(data[i + time_steps, feature_columns.index('Close')])  # Use 'Close' price as
     target
    return np.array(X), np.array(y)

# Split the data into training and testing sets (80/20 split)
train_size = int(len(scaled_features) * 0.8)
test_size = len(scaled_features) - train_size
train_data, test_data = scaled_features[:train_size], scaled_features[train_size:]

time_steps = 10  # Use 10 days of past data to predict the next day's close price
X_train, y_train = prepare_dataset(train_data, time_steps)
X_test, y_test = prepare_dataset(test_data, time_steps)

# Reshape data to fit the input requirements of LSTM/GRU
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2])
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2])

# Define the number of time steps and features
n_steps = X_train.shape[1]
n_features = X_train.shape[2]

# Function to build the LSTM model
def create_lstm_model(n_steps, n_features):
    model = Sequential()
    model.add(LSTM(150, return_sequences=True, input_shape=(n_steps, n_features)))
    model.add(Dropout(0.2))
    model.add(LSTM(150))
```

```
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')
    return model

# Function to build the GRU model
def create_gru_model(n_steps, n_features):
    model = Sequential()
    model.add(GRU(150, return_sequences=True, input_shape=(n_steps, n_features)))
    model.add(Dropout(0.2))
    model.add(GRU(150))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')
    return model

# Train the LSTM model
lstm_model = create_lstm_model(n_steps, n_features)
lstm_checkpoint = ModelCheckpoint('lstm_model.h5', monitor='val_loss', save_best_only=True)
lstm_es = EarlyStopping(monitor='val_loss', mode='min', patience=50)
lstm_model.fit(X_train, y_train, epochs=200, batch_size=64, validation_data=(X_test, y_test),
    callbacks=[lstm_checkpoint, lstm_es], verbose=0)

# Train the GRU model
gru_model = create_gru_model(n_steps, n_features)
gru_checkpoint = ModelCheckpoint('gru_model.h5', monitor='val_loss', save_best_only=True)
gru_es = EarlyStopping(monitor='val_loss', mode='min', patience=50)
gru_model.fit(X_train, y_train, epochs=200, batch_size=64, validation_data=(X_test, y_test),
    callbacks=[gru_checkpoint, gru_es], verbose=0)

# Load the best models
lstm_model = tf.keras.models.load_model('lstm_model.h5')
gru_model = tf.keras.models.load_model('gru_model.h5')

# Generate predictions using both models
lstm_train_predictions = lstm_model.predict(X_train)
lstm_test_predictions = lstm_model.predict(X_test)

gru_train_predictions = gru_model.predict(X_train)
gru_test_predictions = gru_model.predict(X_test)

# Average predictions from both models
train_predictions = (lstm_train_predictions + gru_train_predictions) / 2
test_predictions = (lstm_test_predictions + gru_test_predictions) / 2

# Transform predictions back to original scale
train_predictions_transformed = scaler.inverse_transform(np.concatenate((train_predictions, np
    .zeros((train_predictions.shape[0], scaled_features.shape[1] - 1))), axis=1))[:, 0]
test_predictions_transformed = scaler.inverse_transform(np.concatenate((test_predictions, np.
    zeros((test_predictions.shape[0], scaled_features.shape[1] - 1))), axis=1))[:, 0]
y_train_transformed = scaler.inverse_transform(np.concatenate((y_train.reshape(-1, 1), np.
```

```
    zeros((y_train.shape[0], scaled_features.shape[1] - 1))), axis=1))[:, 0]
y_test_transformed = scaler.inverse_transform(np.concatenate((y_test.reshape(-1, 1), np.zeros
    ((y_test.shape[0], scaled_features.shape[1] - 1))), axis=1))[:, 0]

# Calculate residuals from the predictions
residuals = y_test_transformed - test_predictions_transformed

# Fit a GARCH model to the residuals
garch_model = arch_model(residuals, vol='Garch', p=1, q=1)
garch_fit = garch_model.fit(disp='off')

# Make GARCH predictions
garch_forecast = garch_fit.forecast(horizon=1)
garch_predictions = garch_forecast.mean.iloc[-len(residuals):].values.flatten()

# Combine LSTM/GRU predictions with GARCH predictions
final_test_predictions = test_predictions_transformed + garch_predictions

# Evaluate the model's performance
train_mse = mean_squared_error(y_train_transformed, train_predictions_transformed)
train_mape = calculate_mape(y_train_transformed, train_predictions_transformed)
test_mse = mean_squared_error(y_test_transformed, final_test_predictions)
test_mape = calculate_mape(y_test_transformed, final_test_predictions)

# Plot actual vs predicted close prices
plt.figure(figsize=(12, 6))
plt.plot(sp500_df.index[:train_size + time_steps + 1], scaler.inverse_transform(
    scaled_features)[:train_size + time_steps + 1, feature_columns.index('Close')], color='
    green', label='Training Data')
plt.plot(sp500_df.index[train_size + time_steps + 1:], y_test_transformed, color='blue', label
    ='Actual Close Price')
plt.plot(sp500_df.index[train_size + time_steps + 1:], final_test_predictions, color='red',
    label='Predicted Close Price')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('S&P 500 Close Price Prediction using LSTM/GRU Ensemble with GARCH')
plt.legend()
plt.show()

# Display prediction and actual values
results_df = pd.DataFrame({'Actual': y_test_transformed, 'Predicted': final_test_predictions},
     index=sp500_df.index[train_size + time_steps + 1: train_size + time_steps + 1 + len(
    y_test_transformed)])
print(results_df.head())
print(f"\nTraining MSE: {train_mse}, Training MAPE: {train_mape}%")
print(f"Test MSE: {test_mse}, Test MAPE: {test_mape}%")
```

**Listing 7:** Hybrid (Close price Prediciton) Python Code for S&P 500 Prediction

### 6.2.14. Hybrid - Binary Predicitons

```
import requests
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Conv1D, MaxPooling1D, Dropout, Bidirectional,
    GRU, BatchNormalization, Input
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, LearningRateScheduler
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
import pandas_ta as ta
import pandas_datareader as pdr
from datetime import datetime
import yfinance as yf
from arch import arch_model
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
import keras_tuner as kt
import quandl

# Define the time range for data collection
start = datetime(2018, 1, 1)
end = datetime(2024, 1, 1)

# Retrieve macroeconomic data from Quandl
quandl.ApiConfig.api_key = '7BtjxPppBWXTh3QD_Bks'
cpi = pdr.get_data_fred('CPIAUCSL', start, end)
interest_rate = pdr.get_data_fred('DFF', start, end)
gdp = pdr.get_data_fred('GDP', start, end)
unemployment = pdr.get_data_fred('UNRATE', start, end)

# Rename columns for clarity
cpi.rename(columns={'CPIAUCSL': 'CPI'}, inplace=True)
interest_rate.rename(columns={'DFF': 'Interest_Rate'}, inplace=True)
gdp.rename(columns={'GDP': 'GDP'}, inplace=True)
unemployment.rename(columns={'UNRATE': 'Unemployment_Rate'}, inplace=True)

# Download S&P 500 data from Yahoo Finance
symbol = "^GSPC"
sp500 = yf.download(symbol, start=start, end=end)
sp500 = sp500[['Open', 'High', 'Low', 'Close', 'Volume']]

# Merge S&P 500 data with the macroeconomic data
sp500 = sp500.merge(cpi, how='left', left_index=True, right_index=True)
```

```
sp500 = sp500.merge(interest_rate, how='left', left_index=True, right_index=True)
sp500 = sp500.merge(gdp, how='left', left_index=True, right_index=True)
sp500 = sp500.merge(unemployment, how='left', left_index=True, right_index=True)
sp500.ffill(inplace=True)
sp500.bfill(inplace=True)

# Function to identify fair value gaps based on price movements
def detect_fair_value_gaps(close_prices, threshold=0.02):
    price_diff = np.diff(close_prices)
    gaps = np.abs(price_diff) > (threshold * close_prices[:-1])
    return np.concatenate([[False], gaps])

# Function to identify order blocks based on consolidation patterns
def detect_order_blocks(high, low, close, lookback=5, consolidation_ratio=0.05,
    threshold_ratio=0.1):
    order_blocks = np.zeros_like(close, dtype=bool)
    for i in range(lookback, len(close)):
        recent_high = np.max(high[i-lookback:i])
        recent_low = np.min(low[i-lookback:i])
        recent_close = close[i-1]
        previous_close = close[i-lookback-1]
        strong_move_up = recent_close > previous_close * (1 + threshold_ratio)
        strong_move_down = recent_close < previous_close * (1 - threshold_ratio)
        current_range = high[i] - low[i]
        average_range = np.mean(high[i-lookback:i] - low[i-lookback:i])
        consolidation = current_range < average_range * consolidation_ratio
        if (strong_move_up or strong_move_down) and consolidation:
            order_blocks[i] = True
    return order_blocks

# Add technical indicators
sp500['RSI'] = ta.rsi(sp500['Close'], length=14)
sp500['EMAF'] = ta.ema(sp500['Close'], length=20)
sp500['EMAS'] = ta.ema(sp500['Close'], length=50)
sp500['EMAM'] = ta.ema(sp500['Close'], length=100)
sp500['ATR'] = ta.atr(sp500['High'], sp500['Low'], sp500['Close'], length=14)
bollinger_bands = ta.bbands(sp500['Close'], length=20, std=2)
sp500['Bollinger_Upper'] = bollinger_bands['BBU_20_2.0']
sp500['Bollinger_Lower'] = bollinger_bands['BBL_20_2.0']
sp500['SMA'] = ta.sma(sp500['Close'], length=20)
sp500['MACD'] = ta.macd(sp500['Close'])['MACD_12_26_9']
stoch = ta.stoch(sp500['High'], sp500['Low'], sp500['Close'])
sp500['Stochastic_K'] = stoch['STOCHk_14_3_3']
sp500['Stochastic_D'] = stoch['STOCHd_14_3_3']
sp500['prev_close'] = sp500['Close'].shift(1)
sp500['prev_high'] = sp500['High'].shift(1)
sp500['prev_low'] = sp500['Low'].shift(1)
sp500['prev_close_2'] = sp500['Close'].shift(2)
sp500['prev_high_2'] = sp500['High'].shift(2)
```

```
sp500['prev_low_2'] = sp500['Low'].shift(2)
sp500['day_of_week'] = sp500.index.dayofweek
sp500['day_of_month'] = sp500.index.day

sp500.dropna(inplace=True)

# Function to create binary classification targets
def create_binary_targets(prices):
    return [1 if prices[i+1] > prices[i] else 0 for i in range(len(prices)-1)]

binary_targets = create_binary_targets(sp500['Close'].values)
y_binary = np.array(binary_targets)

# Define features for modeling
features = ['Open', 'High', 'Low', 'Close', 'Volume', 'CPI', 'GDP', 'Interest_Rate', '
    Unemployment_Rate',
            'RSI', 'EMAF', 'EMAS', 'EMAM', 'ATR', 'Bollinger_Upper', 'Bollinger_Lower', 'SMA',
     'MACD',
            'Stochastic_K', 'Stochastic_D', 'prev_close', 'prev_high', 'prev_low', '
    prev_close_2', 'prev_high_2',
            'prev_low_2', 'day_of_week', 'day_of_month']

# Normalize the feature data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_features = scaler.fit_transform(sp500[features].values)

# Ensure y_binary and features have matching lengths
scaled_features = scaled_features[:len(y_binary)]

# Feature selection using Random Forest
random_forest = RandomForestClassifier(n_estimators=100, random_state=42)
random_forest.fit(scaled_features, y_binary)
feature_importances = random_forest.feature_importances_
sorted_indices = np.argsort(feature_importances)[::-1]

# Display the feature ranking
print("Feature ranking:")
for i in range(scaled_features.shape[1]):
    print(f"{i + 1}. {features[sorted_indices[i]]} ({feature_importances[sorted_indices[i]]})"
    )

# Select important features
feature_selector = SelectFromModel(random_forest, threshold="mean", prefit=True)
selected_features = feature_selector.transform(scaled_features)
selected_feature_names = [features[i] for i in sorted_indices[:selected_features.shape[1]]]

print("Selected features:", selected_feature_names)

# Function to split sequences into X, y pairs
```

```
def split_sequences(sequences, n_steps, y_binary):
    X, y = [], []
    for i in range(len(sequences)):
        end_ix = i + n_steps
        if end_ix >= len(sequences): break
        seq_x, seq_y = sequences[i:end_ix, :], y_binary[i]
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X), np.array(y)

n_steps = 10
X, y = split_sequences(selected_features, n_steps, y_binary)
n_features = X.shape[2]
X = X.reshape((X.shape[0], n_steps, n_features))

train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train_binary, y_test_binary = y[:train_size], y[train_size:]

# Function to build LSTM model for hyperparameter tuning
def build_lstm_model_tuner(hp):
    model = Sequential()
    model.add(Input(shape=(n_steps, n_features)))
    model.add(Conv1D(filters=hp.Int('conv_filters', min_value=32, max_value=128, step=32),
                     kernel_size=hp.Int('conv_kernel_size', min_value=2, max_value=5, step=1),
                     activation='relu',
                     kernel_regularizer=l2(0.001)))
    model.add(MaxPooling1D(pool_size=2))
    model.add(BatchNormalization())
    model.add(Bidirectional(LSTM(units=hp.Int('lstm_units_1', min_value=50, max_value=200,
    step=25),
                                 return_sequences=True)))
    model.add(Dropout(rate=hp.Float('dropout_1', min_value=0.2, max_value=0.5, step=0.1)))
    model.add(Bidirectional(LSTM(units=hp.Int('lstm_units_2', min_value=25, max_value=100,
    step=25))))
    model.add(BatchNormalization())
    model.add(Dropout(rate=hp.Float('dropout_2', min_value=0.2, max_value=0.5, step=0.1)))
    model.add(Dense(units=hp.Int('dense_units', min_value=25, max_value=100, step=25),
    activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=Adam(learning_rate=hp.Float('learning_rate', min_value=1e-4,
    max_value=1e-2, sampling='LOG')),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model

# Hyperparameter tuning for the LSTM model
lstm_tuner = kt.Hyperband(build_lstm_model_tuner,
                          objective='val_accuracy',
```

```
                            max_epochs=50,
                            factor=3,
                            directory='hyperband_dir',
                            project_name='lstm_hyperband')

early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
lstm_tuner.search(X_train, y_train_binary, epochs=50, validation_data=(X_test, y_test_binary),
     callbacks=[early_stopping])

# Retrieve the best hyperparameters for the LSTM model
best_lstm_hps = lstm_tuner.get_best_hyperparameters(num_trials=1)[0]

print(f"""
Hyperparameter tuning complete. Best LSTM configuration:
- First LSTM layer units: {best_lstm_hps.get('lstm_units_1')}
- Second LSTM layer units: {best_lstm_hps.get('lstm_units_2')}
- Learning rate: {best_lstm_hps.get('learning_rate')}
""")

# Build and train the best LSTM model
best_lstm_model = lstm_tuner.hypermodel.build(best_lstm_hps)
lstm_checkpoint = ModelCheckpoint('lstm_best_model.h5', monitor='val_loss', save_best_only=
    True)
lstm_es = EarlyStopping(monitor='val_loss', mode='min', patience=20, restore_best_weights=True
    )

def learning_rate_scheduler(epoch, lr):
    return lr if epoch < 100 else lr * tf.math.exp(-0.05)

lr_callback = LearningRateScheduler(learning_rate_scheduler)
best_lstm_model.fit(X_train, y_train_binary, epochs=150, batch_size=64, validation_data=(
    X_test, y_test_binary), callbacks=[lstm_checkpoint, lstm_es, lr_callback], verbose=1)

# Repeat the process for GRU model hyperparameter tuning
def build_gru_model_tuner(hp):
    model = Sequential()
    model.add(Input(shape=(n_steps, n_features)))
    model.add(Conv1D(filters=hp.Int('conv_filters', min_value=32, max_value=128, step=32),
                     kernel_size=hp.Int('conv_kernel_size', min_value=2, max_value=5, step=1),
                     activation='relu',
                     kernel_regularizer=l2(0.001)))
    model.add(MaxPooling1D(pool_size=2))
    model.add(BatchNormalization())
    model.add(Bidirectional(GRU(units=hp.Int('gru_units_1', min_value=50, max_value=200, step
    =25),
                                return_sequences=True)))
    model.add(Dropout(rate=hp.Float('dropout_1', min_value=0.2, max_value=0.5, step=0.1)))
    model.add(Bidirectional(GRU(units=hp.Int('gru_units_2', min_value=25, max_value=100, step
    =25))))
```

```
    model.add(BatchNormalization())
    model.add(Dropout(rate=hp.Float('dropout_2', min_value=0.2, max_value=0.5, step=0.1)))
    model.add(Dense(units=hp.Int('dense_units', min_value=25, max_value=100, step=25),
    activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=Adam(learning_rate=hp.Float('learning_rate', min_value=1e-4,
    max_value=1e-2, sampling='LOG')),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model

# Hyperparameter tuning for the GRU model
gru_tuner = kt.Hyperband(build_gru_model_tuner,
                         objective='val_accuracy',
                         max_epochs=50,
                         factor=3,
                         directory='hyperband_dir',
                         project_name='gru_hyperband')

gru_tuner.search(X_train, y_train_binary, epochs=50, validation_data=(X_test, y_test_binary),
    callbacks=[early_stopping])

# Retrieve the best hyperparameters for the GRU model
best_gru_hps = gru_tuner.get_best_hyperparameters(num_trials=1)[0]

print(f"""
Hyperparameter tuning complete. Best GRU configuration:
- First GRU layer units: {best_gru_hps.get('gru_units_1')}
- Second GRU layer units: {best_gru_hps.get('gru_units_2')}
- Learning rate: {best_gru_hps.get('learning_rate')}
""")

# Build and train the best GRU model
best_gru_model = gru_tuner.hypermodel.build(best_gru_hps)
gru_checkpoint = ModelCheckpoint('gru_best_model.h5', monitor='val_loss', save_best_only=True)
gru_es = EarlyStopping(monitor='val_loss', mode='min', patience=20, restore_best_weights=True)
best_gru_model.fit(X_train, y_train_binary, epochs=150, batch_size=64, validation_data=(X_test
    , y_test_binary), callbacks=[gru_checkpoint, gru_es, lr_callback], verbose=1)

# Load the best models for predictions
best_lstm_model = tf.keras.models.load_model('lstm_best_model.h5')
best_gru_model = tf.keras.models.load_model('gru_best_model.h5')

# Make predictions with the LSTM and GRU models
lstm_train_preds = best_lstm_model.predict(X_train).flatten()
lstm_test_preds = best_lstm_model.predict(X_test).flatten()
gru_train_preds = best_gru_model.predict(X_train).flatten()
gru_test_preds = best_gru_model.predict(X_test).flatten()

```

```
# Average the predictions from both models
average_train_preds = (lstm_train_preds + gru_train_preds) / 2
average_test_preds = (lstm_test_preds + gru_test_preds) / 2

# Compute residuals
residuals = y_test_binary - average_test_preds

# Apply GARCH model to the residuals
garch = arch_model(residuals, vol='Garch', p=1, q=1)
garch_results = garch.fit(disp='off')

# Make GARCH-based adjustments to predictions
garch_predictions = garch_results.forecast(horizon=1).mean.iloc[-len(residuals):].values.
    flatten()
final_test_predictions = average_test_preds + garch_predictions

# Convert predictions to binary classification
binary_predictions = (final_test_predictions > 0.5).astype(int)

# Calculate performance metrics
accuracy = accuracy_score(y_test_binary, binary_predictions)
precision = precision_score(y_test_binary, binary_predictions)
recall = recall_score(y_test_binary, binary_predictions)
f1 = f1_score(y_test_binary, binary_predictions)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")

plt.figure(figsize=(12, 6))
plt.plot(y_test_binary, label='Actual Trend')
plt.plot(binary_predictions, label='Predicted Trend', alpha=0.7)
plt.title('Actual vs Predicted Market Trends')
plt.xlabel('Time Steps')
plt.ylabel('Trend (0 for Down, 1 for Up)')
plt.legend()
plt.show()
```

**Listing 8:** Hybrid (Close price Binary Prediciton) Python Code for S&P 500 Prediction

# APPENDIX B

### 6.2.15. Key Packages

- **Pandas** [63] is used for data manipulation and analysis. It offers data structures and operations for manipulating numerical tables and time series.
- **NumPy** [64] provides support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- **Statsmodels** [65] is utilized for statistical modeling, which includes statistical tests and exploring data's statistical properties.
- **Matplotlib** [66] is a plotting library for creating static, interactive, and animated visualizations in Python.
- **Scikit-learn** [67] is used for machine learning, providing simple and efficient tools for predictive data analysis.
- **yfinance** [68] allows fetching historical market data from Yahoo Finance, which is crucial for financial analysis.
- **Quandl** [69] is used to fetch financial, economic, and alternative datasets, including this project's CPI, GDP, interest rates, and unemployment rates.
- **Arch** [70] is a library is used for time series modeling with a focus on volatility modeling, such as ARCH and GARCH models.
- **Copulas** [71] is a Python library for modeling multivariate distributions using copulas. It is useful for generating samples from multivariate distributions and for performing simulations.
- **Tensorflow** [72] is an open-source platform designed to make machine learning accessible and scalable, providing a flexible ecosystem of tools and libraries for building and deploying ML models.