FORECASTING THE NEXT WINNING STOCK: A COMPARATIVE ANALYSIS OF MACHINE LEARNING MODELS

BLANCA ELVIRA FERNÁNDEZ MÉNDEZ

A THESIS SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF ARTS

GRADUATE PROGRAM IN APPLIED MATHEMATICS
YORK UNIVERSITY
TORONTO, ONTARIO

August 2025

Abstract

Stock price prediction is a common and complex problem due to the high volatility of financial markets. This master's thesis presents a new approach to stock price forecasting by reformulating the problem as a multiclass classification task. The main objective is to predict which stock will yield the highest return the next day within a given set of features. To this end, various statistical and machine learning models are analyzed, with special emphasis on the Transformer model due to its relevance and alignment with the structure of this work. The present study proposes a novel idea to address the problem. Its contributions stand out in an initial exploratory analysis of model performance, as well as in risk minimization in investments, enabling portfolio diversification thanks to the Transformer model.

Keywords: Forecasting stocks, NASDAQ, Machine Learning, LSTM, Transformers, Transformer Models

Dedication

First of all, and although it may not be the most conventional choice, I would like to dedicate this thesis to myself. This master's degree has not been an easy journey, even before it began. There have always been more problems than solutions, and the latter often took months to arrive. For this reason, I dedicate it to myself: for insisting, again and again, when everything seemed to encourage me to stop.

To my parents and my brother, for rejoicing in each of my achievements and for always being there. To my lifelong friends, for their constant support and for encouraging me from the very first day; and to those I have met along the way, for making this journey more human and lighter.

To my classmates, for sharing days and nights of study, notes, help, advice, and honest opinions. Thank you for walking by my side.

To L'Aquila, for gifting me so many wonderful people, with great hearts and unique stories. Thank you for the bonds created, the laughter shared, and for making this stage of my life warmer.

To Canada, for offering me a more personal and introspective perspective, allowing me to see the world through different eyes and grow as a person. And to all those people who, especially towards the end of my stay, became friends.

Finally, I sincerely thank the Rector of the Complutense University of Madrid, in the Faculty of Mathematical Sciences, Dr. Bru Espino. Without his kindness and support on my last official day at the Faculty, these words would probably not exist today.

Acknowledgments

I would like to express my gratitude to my supervisor, Professor Jairo Díaz Rodríguez, and to Professor Pavlos Gkasis, for their constant support and guidance throughout the development of this thesis. To Professor Jairo Díaz, for introducing me to the fascinating world of Artificial Intelligence and Machine Learning models, for his idea that inspired the development of this thesis, for always being approachable and kind, and for his remarkable ability to make complex topics easy to understand through his clear and insightful explanations, as well as for his availability and support from the very beginning, despite the tight deadlines. To Professor Pavlos Gkasis, for his valuable financial perspective, always thoughtful insights, and his continuous willingness to help.

Table of Contents

List of Tables

Chapter 1

INTRODUCTION

Stock price prediction in financial markets is a common yet critically important problem that has garnered significant attention over the years. Beyond enabling investors to maximize their profits, accurate stock forecasting plays a vital role in assessing a country's economic performance and informing its policy decisions. As such, the problem has become a focal point for both academic researchers and financial practitioners. The inherent complexity of the task—driven by the wide range of influencing factors, indices, and the high volatility of financial markets—poses considerable challenges. This master's thesis aims to propose a redefined methodology for stock price prediction by integrating both traditional statistical techniques and modern computational approaches.

1.1 Motivation

The motivation behind this work stems from the recent advances in Transformer model architectures [29]. Transformer models and their associated self-attention mechanisms have brought significant breakthroughs in natural language processing (NLP), enabling the effective handling of complex sequential data. These advancements have led to substantial improvements in a variety of tasks, including machine translation, text summarization, and question answering. The core innovation of self-attention lies in its ability to assess the contextual relevance of each token in relation to others within a sequence, thereby enhancing the model's capacity to focus on pertinent information, particularly in tasks such as next-token prediction [6]. Recent efforts to enhance the computational efficiency of self-attention have yielded empirical success not only within NLP but also across other domains such as computer vision [4] and protein design [24]. Furthermore, recent studies have investigated the application of Transformer models to financial data [22, 13], emphasizing their potential fit for addressing the challenges inherent in time-series analysis. Nonetheless, this remains an emerging area of research that demands further exploration and development.

Traditional methods for stock price prediction have predominantly focused on forecasting continuous variables—most commonly, the stock's closing price—or, alternatively, on binary classification tasks such as determining whether a stock's price will increase or decrease, or identifying broader market trends [16]. These methodologies typically aim to minimize error metrics such as the Mean Squared Error (MSE) or the Mean Absolute Percentage Error (MAPE) [30, 11]. In contrast, this study approaches the problem from a novel perspective: rather than formulating it as a regression task, we frame it as

a multiclass classification problem, a formulation that, to the best of our knowledge, has not been previously explored in the literature. The objective is to predict, from a given set of stocks, the one that will yield the highest return on the following trading day. Under this framework, each stock is treated as a 'token', and the model's task is to identify the next element in the sequence corresponding to the daily top-performing stock. This formulation is naturally aligned with the Transformer architecture and its sequence modeling capabilities.

Nonetheless, given that this constitutes a completely novel approach, it is essential to conduct a comparative analysis with other well-established models commonly employed in the literature for stock price prediction—such as ARIMA, Random Forest, and Long Short-Term Memory (LSTM) networks—prior to implementing Transformer-based architectures.

Accordingly, this Master's thesis introduces a pioneering perspective in the field of stock price prediction by presenting, for the first time, an analysis of the multiclass classification framework through the application of both statistical and Machine Learning models.

1.2 The Difficulty of the Problem

Stock price prediction has posed a persistent challenge since the inception of financial markets. Its inherent complexity has led to numerous historical episodes that underscore the uncertainty and volatility of the financial environment. A typical example occurred shortly before the 1929 stock market crash and the subsequent Great Depression in the United States, when several prominent experts confidently asserted that both the economy and the markets would continue to grow indefinitely [9]. One notable case involves the renowned business management scholar Peter Drucker, who was promoted to full professor after publishing an article in 1929 forecasting a prolonged stock market boom. Following the complete failure of this prediction, Drucker resolved never to make economic forecasts again for the remainder of his career. Another significant episode took place in 1994, when Nobel laureates Robert Merton and Myron Scholes founded the investment firm Long-Term Capital Management (LTCM), which promised exceptional returns through the use of sophisticated financial strategies. However, in 1998, the firm collapsed after incurring massive losses, driving many individual investors into bankruptcy [17]. Similarly, during the late 1990s—a period marked by robust stock market growth—some economists claimed that traditional economic cycles had been overcome and that recessions were a phenomenon of the past [26]. Nevertheless, in 2001, one of the most significant market downturns in recent history began, lasting until the end of 2003. The markets subsequently entered a sustained recovery phase that persisted until mid-2007.

These historical episodes illustrate a fundamental aspect of free markets: prices fluctuate according to the forces of supply and demand. While there is broad consensus that, over the long term, prices tend to reflect the intrinsic or fundamental value of assets, short-term deviations are frequent and can be substantial, often driven by irrational factors. The speculative and emotionally driven behavior of many market participants contributes significantly to the unpredictability of short-term price movements.

The random walk hypothesis, proposed by Fama (1995) [7], posits that stock price changes follow an essentially random pattern, thereby rendering any attempt at accurate fore-

casting theoretically unfounded. Similarly, the Efficient Market Hypothesis (EMH) [18] maintains that future stock price movements cannot be reliably inferred from available historical data. However, these assumptions have been challenged by various studies suggesting that, in certain contexts, stock prices exhibit repetitive patterns that may stem from the collective and systematic behavior of investors. Consequently, it has been argued that stock prices may be partially predictable [25, 5]. The following section offers a brief overview of several models that have been employed in the literature for the purpose of stock price prediction.

1.3 Literature Review

Stock price prediction has been studied from various methodological standpoints over the years, encompassing approaches that range from traditional statistical models to advanced *Machine Learning* algorithms and neural network architectures. Among the most widely used classical models are autoregressive techniques such as the Autoregressive Integrated Moving Average (ARIMA), which are designed to model univariate time series by accounting for trends and seasonal components [21, 34]. Conversely, classification models such as *Logistic Regression* have predominantly been employed to forecast the directional movement of stock prices—i.e., whether they will increase or decrease—rather than their continuous values [14]. Nevertheless, when combined with other methods, logistic regression has also been adapted to address regression tasks in financial contexts.In recent years, the application of machine learning algorithms has gained considerable traction. Models such as *Random Forests*, which are based on ensembles of decision trees, have demonstrated strong performance in capturing non-linear and complex relationships among predictive variables [30]. Recurrent neural network architectures, particularly *Long Short-Term Memory* (LSTM) networks, have also proven to be highly effective for modeling temporal sequences with long-term dependencies, such as historical stock price data [1]. LSTM models have frequently outperformed traditional statistical approaches in capturing the non-linear and dynamic behaviors characteristic of financial markets [27]. However, some studies have highlighted that classical models such as ARIMA may outperform LSTMs in scenarios involving a limited number of features—particularly when only a single time series variable is considered [11].

Meanwhile, *Transformer Models* [29], an architecture initially introduced for natural language processing tasks, have recently begun to gain traction in the financial domain. Owing to their attention mechanism, Transformers are capable of efficiently capturing long-range dependencies within sequential data, positioning them as a promising alternative for modeling complex time series such as stock prices. Recent research has investigated their applicability to tasks such as market trend forecasting and asset classification [22, 13]. Despite their considerable potential, the use of Transformer models in finance remains a nascent and rapidly evolving area of study. In particular, there is substantial room for advancement in extending their application beyond continuous regression to more nuanced tasks, such as multiclass classification or ordinal prediction of relative asset performance.

1.4 Summary of Contributions

This work introduces an innovative approach to the problem of stock prediction, diverging from the traditional framework that primarily focuses on forecasting continuous variables such as a stock's closing price. Drawing inspiration from the Transformer architecture, the problem is reformulated as a multiclass classification task. In this framework, the best-performing stocks on each trading day are treated as a sequence, and the objective is to predict the next element in that sequence—an approach that aligns naturally with the underlying principles of Transformer models.

The primary contribution of this thesis lies in the application of Transformer-based architectures to the task of identifying the top-performing financial stock in each time interval. The performance of these models is systematically compared with that of more established approaches, with the aim of assessing their suitability for classification tasks within financial contexts.

Chapter 2 presents the dataset used throughout the study, along with the rationale behind its selection. Chapter 3 provides a detailed description of the models employed. The methodology implemented for each model is discussed in the corresponding sections. In Chapter 4, a comparative analysis of the results obtained across all models is presented. Finally, Chapter 5 summarizes the main contributions of the work and outlines potential directions for future research. All code has been developed in Python using Jupyter Notebooks and is available at [19].

Chapter 2

Data

The quality and quantity of data used in this type of study are critical for the development of efficient models and the generation of accurate results. Consequently, selecting a consistent and reliable dataset is a key step to prevent predictive errors and erroneous assumptions. Equally important is the execution of proper data preprocessing, which involves identifying and handling missing values, detecting outliers, applying normalization techniques, and conducting thorough exploratory data analysis. These procedures are essential to ensure the robustness and reliability of the results produced by the models.

2.1 The Stocks

In this study, seven stocks with the highest market capitalization according to NASDAQ were selected for analysis.[1] Accordingly, the dataset employed in this project contains historical price information for the following seven stocks: Apple (AAPL), Amazon (AMZN), Google (Alphabet Inc. Class C, GOOG), Meta (META), Microsoft (MSFT), NVIDIA (NVDA), and Tesla (TSLA). The datasets used in this study are available at [20].

2.1.1 Why Seven Stocks?

Given that this study focuses on stock price prediction using forecasting models based on self-attention mechanisms, the selection of the companies under analysis is critical to ensuring both the validity and the relevance of the model. For this reason, a dedicated section is included to explain the rationale behind the choice of these specific companies. The seven stocks with the highest market capitalization within the NASDAQ-100 index were selected for the following reasons:

1. **Market Leadership and Representativeness:** Companies with the largest market capitalizations are typically global industry leaders. Their stock behavior tends to be more stable and less influenced by idiosyncratic factors, thereby facilitating the identification of macroeconomic and sector-specific patterns. Moreover, these

[1]The NASDAQ (National Association of Securities Dealers Automated Quotation) is the second-largest electronic stock exchange in the United States by market capitalization. It hosts numerous high-technology companies, including those in the biotechnology, telecommunications, and information technology sectors. Established in 1971 and headquartered in New York, it was the world's first electronic stock exchange. Notably, it does not have a physical trading floor; instead, all transactions are conducted via a telecommunications network, without the traditional face-to-face matching of buy and sell orders.

companies constitute a significant portion of the NASDAQ index and substantially influence its overall performance.

2. **High Liquidity and Trading Volume:** These stocks exhibit high liquidity, which reduces volatility stemming from a lack of market participants. This characteristic is particularly important in forecasting studies, as models benefit from more stable and continuous data, minimizing distortions caused by low trading frequency or illiquid movements.

3. **Avoidance of Noise from Smaller Companies:** Incorporating a large number of companies—particularly those with lower market capitalizations—can introduce noise into the dataset due to heightened volatility, vulnerability to firm-specific events (such as mergers, acquisitions, or financial distress), and decreased predictability. By focusing on the largest firms, the model is more likely to capture general market patterns rather than company-specific biases.

4. **Sufficient Diversification Without Loss of Focus:** Selecting seven companies offers a balanced compromise between achieving diversification and maintaining model tractability. Choosing only four or five stocks could limit representativeness, while including ten or more might decrease the model's precision due to increased variability and complexity in the data.

5. **Consistency in Market Capitalization:** Over recent years, the top seven companies in the NASDAQ have maintained relatively stable market capitalizations [31]. This consistency facilitates longitudinal analysis and reduces the impact of abrupt shifts in the dataset's composition, thereby enhancing the interpretability of the model's predictive results.

6. **Investment Strategy Validation:** Investment strategies often concentrate on large-cap companies, as these firms are considered more stable and less susceptible to market manipulation. A predictive model capable of accurately forecasting the performance of such companies has the potential to inform real-world financial decision-making.

7. **Computational and Training Efficiency:** Utilizing a reduced yet highly representative set of stocks enables more efficient training of Transformer-based models without compromising their generalization capabilities. Including a larger number of companies would significantly increase computational costs without necessarily yielding better predictive performance.

For these reasons, selecting the top seven NASDAQ-listed companies by market capitalization offers an optimal balance between representativeness, stability, computational efficiency, and practical applicability. This strategy allows for training a model on robust and meaningful data, mitigating unnecessary biases and enabling more reliable predictions of stock market trends.

It is important to note that Google's shares are divided into two distinct classes: Alphabet Inc. Class C Capital Stock (GOOG) and Alphabet Inc. Class A Common Stock (GOOGL). The primary distinction lies in voting rights: GOOG shares are publicly available but do not grant voting privileges, whereas GOOGL shares, typically held by the company's founders, confer decision-making authority. This dual-class share structure

was introduced as a governance strategy to enable capital raising through public markets while allowing the founders to retain long-term strategic control. The share split took place in April 2014, motivated by high investor demand and the company's interest in preserving centralized decision-making power. Both classes exhibit highly similar price behavior and statistical properties, which may lead to multicollinearity or high covariance in data-driven analyses. To avoid redundancy and ensure model stability, only GOOG shares have been included in this study, as they are more representative of standard publicly traded stock behavior.

2.2 The Time Period

The temporal scope of the dataset plays a critical role in the performance of predictive models. In particular, complex architectures such as Transformers or recurrent neural networks require extended time sequences to effectively capture long-term dependencies and extract intricate patterns from time series dynamics. Nevertheless, it is essential to recognize that an excessively broad time horizon may incorporate information that no longer reflects the current behavior of the market or the phenomenon under investigation. Shifts in macroeconomic conditions, regulatory environments, or unforeseen events—such as financial crises or global pandemics—can significantly alter system dynamics. As a result, incorporating outdated data may introduce noise or bias that undermines the model's generalization capabilities for more recent data. Striking an appropriate balance between the length of the time horizon and the relevance of the data to the current context is therefore essential for building effective predictive models.

Initially, the models were trained on data covering the period from 2021 to early 2025, with 1,016 data points per stock. This time frame was selected to avoid the extreme market volatility observed during the COVID-19 crisis in 2020, as well as the heightened fluctuations following the U.S. presidential elections on January 20, 2025, both of which could introduce instability into forecasting models. However, after attempting to apply deep learning models—specifically LSTM architectures—it became evident that the sample was insufficient to train such models effectively. Consequently, the decision was made to extend the dataset. In addition, tests were conducted using historical stock data retrieved from platforms such as Yahoo Finance [35], which offers a broader time span and a wider range of financial variables compared to other platforms. Nevertheless, the earliest available data for one of the selected stocks (META) begins in May 2012. Therefore, this date was adopted as the common starting point across all datasets to ensure consistency. Ultimately, it was decided to extract the data directly from NASDAQ [23], as it is the primary and most reliable source, publicly accessible and more accurate than third-party providers. This choice also eliminates potential inconsistencies previously observed in Yahoo Finance data.

The data utilized in this study is publicly accessible via the official NASDAQ website [23]. The dataset comprises daily trading records from Monday to Friday, between 9:30 AM and 4:00 PM Eastern Time, excluding U.S. public holidays. The earliest available data point corresponds to May 7, 2015, while the most recent corresponds to May 6, 2025, resulting in a total of 2,515 data points per stock and covering a ten-year period—the maximum historical range provided by NASDAQ. Furthermore, during the validation phase of the machine learning models presented in Section 3.1.7, an additional three-year time window (spanning 2022 to 2024 inclusive) will be introduced to evaluate whether

model performance improves under a more recent and narrower temporal scope.

2.3 Features

Features constitute the information that the model utilizes to learn and generate predictions. The quality, relevance, and appropriate representation of these variables have a direct impact on model performance. Therefore, careful feature selection enables the model to capture meaningful relationships with the target variable, mitigate overfitting, and enhance generalization capabilities.
The NASDAQ dataset includes the following features:

- **Date**: The date corresponding to the record.
- **Volume**: The number of shares traded during market hours.
- **Close/Last**: The closing price of the stock.
- **Open**: The opening price of the stock.
- **High**: The highest trading price of the stock.
- **Low**: The lowest trading price of the stock.

In addition to these, the target variable used in this study is the daily return expressed as a percentage, calculated as follows:

$$\text{Daily Return} = \left(\frac{\text{Closing Price} - \text{Opening Price}}{\text{Opening Price}} \right) \times 100 \tag{2.1}$$

Subsequently, the datasets for all selected stocks are concatenated, with an additional column named `Stock` introduced to identify the stock to which each observation corresponds.

However, after implementing the initial models—particularly the Long Short-Term Memory (LSTM) networks—it became apparent that enriching the dataset with additional features was necessary to improve model performance and maximize predictive accuracy. Accordingly, three categorical features were added to represent the day of the week, the month, and the week number within the year, as these temporal factors may influence stock price behavior. Furthermore, a set of technical indicators was incorporated, which are described in detail in the next subsection.

2.3.1 Technical Indicators: Definitions and Calculations

As mentioned earlier, the implementation of models such as LSTM revealed the need for more technical indicators to effectively train the models. This procedure has also been implemented in other research [1], [8]. As a result, the following columns were introduced into the dataset:

1. Relative Strength Index (RSI)

An oscillator measures the speed and change of price movements. It indicates how "overbought" or "oversold" an asset is, helping identify potential reversal points. RSI > 70 usually indicates overbought conditions (possible drop), while RSI < 30 suggests oversold (possible rebound). It was introduced by J. Welles Wilder in his book *New Concepts in Technical Trading Systems* [32], where a standard 14-day period is established for its calculation. The steps are:

1. Calculate daily price changes:

$$\Delta P_t = P_t - P_{t-1},$$

   where P_t represents the closing price on day t

2. Separate gains and losses:

$$G_t = \max(\Delta P_t, 0), \quad L_t = -\min(\Delta P_t, 0)$$

3. Compute the exponential moving average of gains and losses: $AvgGain$ and $AvgLoss$.

4. Compute RS:

$$RS = \frac{AvgGain}{AvgLoss}$$

5. Compute RSI:

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

2. Simple Moving Average (SMA)

Represents the simple average of closing prices over a given period. In our case, a 14-day window is used—matching that of the RSI—to minimize null values in the dataset columns. It smooths daily fluctuations and reveals the overall trend. A rising SMA indicates a bullish[2] trend, while a declining one indicates a bearish[3] trend. It is calculated as:

$$\text{SMA}_n(t) = \frac{1}{n} \sum_{i=0}^{n-1} P_{t-i}$$

where P_{t-i} is the closing price on day $t - i$, and n is the time window (in days).

3. Exponential Moving Average (EMA)

Similar to SMA but assigns more weight to recent prices, allowing faster reaction to market changes. It better detects rapid trend shifts than SMA.
It is calculated recursively as:

$$EMA_t = \alpha \times P_t + (1 - \alpha) \times EMA_{t-1}$$

[2]upward-trending market
[3]downward-trending market

where

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

P_t is the closing price and EMA_{t-1} is the EMA from the previous period. Again, a 14-day period ($n = 14$) is used in our case.

4. Moving Average Convergence Divergence (MACD)

A momentum indicator that shows the difference between two EMAs (typically 12 and 26 days):

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

Once the MACD line is calculated, the Signal Line is defined to smooth the difference. This line is simply an exponential moving average of the MACD itself, typically over 9 periods:

$$\text{Signal}_t = EMA_9(MACD_t)$$

where $EMA_9(MACD_t)$ is the exponential moving average of the MACD with smoothing parameter $\alpha = \frac{2}{10}$, corresponding to a 9-day period.
m!

From these two lines (MACD and Signal), the MACD Histogram is defined, which measures the difference between them and provides a clear visualization of the asset's momentum strength:

$$\text{Histogram}_t = MACD_t - \text{Signal}_t$$

This histogram helps identify potential trend shifts. Increasing positive values suggest bullish momentum, while increasing negative values (in magnitude) suggest bearish acceleration.

5. Bollinger Bands

Bollinger Bands display volatility levels and identify overbought or oversold conditions based on volatility. Plotted at a distance of standard deviation from the SMA (typically 20 days [2]). Calculated as:

$$UpperBand = SMA_{20} + 2 \times \sigma_{20}$$
$$LowerBand = SMA_{20} - 2 \times \sigma_{20}$$

where σ_{20} is the standard deviation of prices over the last 20 days.

6. Average True Range (ATR)

The Average True Range (ATR) is a volatility measure based on the largest difference between daily price ranges. It assesses how much a price moves on average and helps adjust stops or manage risk. Calculated as:
The *True Range* (TR) is the maximum of:

$$TR_t = \max\left(High_t - Low_t, \quad |High_t - Close_{t-1}|, \quad |Low_t - Close_{t-1}|\right)$$

Then,

$$\text{ATR}_t = \frac{1}{n} \sum_{i=t-n+1}^{t} \text{TR}_i$$

The standard period is 14 days [32].

7. Stochastic Oscillator

The Stochastic Oscillator is an indicator that compares the current price to the range of prices over a period (typically 14 days [12]). It measures momentum and detects overbought or oversold conditions. It is calculated as:

$$\%K = 100 \times \frac{C - L_n}{H_n - L_n}$$

where:

- C is the current closing price,
- H_n is the highest price over the last n days,
- L_n is the lowest price over the last n days.

After adding these technical indicators, it becomes clear that the first 19 rows of each stock will contain at least one null value, due to the calculation of the Bollinger Bands. Therefore, these rows with null values are removed from the dataset. As a result, our final dataset contains a total of 17,472 rows and 23 columns, with 2,496 data points per stock. The data starts on June 4, 2015, and ends on May 6, 2025. This is the summary of the data set features:

Table 2.1: Features of the data set

Feature	**Description**
Date	Calendar date of the recorded data point.
Close/Last	Closing price or the last recorded price of the stock for that date.
Volume	Total number of shares traded during that trading day.
Open	The price at which the stock opened trading on that day.
High	The highest price reached by the stock during the trading session.
Low	The lowest price reached by the stock during the trading session.
daily_return (%)	The percentage change in the closing price from the previous day.
Stock	Ticker symbol or name of the stock.
RSI	Relative Strength Index, a momentum oscillator that measures the speed and change of price movements.
SMA	Simple Moving Average, calculated over a given period to smooth out price data.
EMA	Exponential Moving Average, similar to SMA but giving more weight to recent prices.
EMA_12	EMA calculated over a 12-day period.
EMA_26	EMA calculated over a 26-day period.
MACD	Moving Average Convergence Divergence.
MACD_Signal	Signal line of the MACD, a 9-day EMA of the MACD itself.
MACD_Histogram	Difference between MACD and MACD_Signal, used to spot momentum.
BB_Upper	Upper band of the Bollinger Bands, representing overbought threshold.
BB_Lower	Lower band of the Bollinger Bands, representing oversold threshold.
ATR	Average True Range, a volatility indicator measuring market fluctuations.
%K	Stochastic oscillator's main line, measuring the closing price relative to recent highs/lows.
Week_day	Day of the week (e.g., Monday, Tuesday).
Month	Month of the year (e.g. January, May)
Week_number	Week number within the year (1–52).

2.4 Data Preprocessing

Prior to inputting the data into the models, it is essential to perform data cleaning and normalization to ensure consistency across features. The principal preprocessing steps include the detection of missing values, identification of outliers, and standardization of variables.

2.4.1 Missing Values

Missing or null values in the dataset represent a critical issue, as they can distort model outcomes—particularly in models that require synchronized data across multiple stocks. The absence of data points for any given stock could introduce bias into the predictive process. Fortunately, the dataset employed in this study contains no null or missing values.

2.4.2 Outliers

The detection and treatment of outliers is particularly important in financial datasets, as extreme values can distort the scale of variables and adversely affect the stability and accuracy of predictive models. In the context of data scaling, outliers may disproportionately influence normalization procedures, compressing the majority of data points into narrow ranges and thereby complicating the identification of meaningful patterns.

Several outliers were detected in the target variable, predominantly associated with Tesla and NVIDIA stocks. Notably, a strong correlation was observed between the occurrence of these extreme values and the frequency with which these stocks were classified as winners. This suggests that such outliers do not necessarily represent erroneous or anomalous data points, but rather reflect genuine market phenomena with significant predictive value. Consequently, these outliers were retained in the final dataset, as they provide valuable information to support model training. Likewise, outliers present in other variables are considered to represent meaningful variations relevant for forecasting stock behavior. Nevertheless, to mitigate their influence on the models, these values will be standardized as described in the following section.

2.4.3 Standardization

As previously discussed, standardizing the variables is crucial to ensure that all input features to the model operate on a comparable scale, thereby preventing predictive bias toward features with inherently larger or smaller value ranges. With the exception of the temporal variables introduced into the model (`Week_day`, `Month`, `Week`) as well as the `Date` and `Stock` identifiers, all other variables employed in this study are continuous and exhibit varying scales and magnitudes. Consequently, it is necessary to transform these features prior to model training. The primary objective of this transformation is to avoid dominance of variables with numerically larger values over others, while also promoting more efficient convergence during the training process.

Standardization was performed using `StandardScaler`, which rescales each variable to have zero mean and unit standard deviation. This technique is particularly advantageous when variables approximate a normal distribution and is well suited for models sensitive to feature scale, such as Long Short-Term Memory (LSTM) networks and Transformer architectures. Although some variables in the dataset do not strictly adhere to a normal distribution, standardization remains an appropriate choice. Firstly, it enhances numerical stability and accelerates the training efficiency of deep learning models that are scale-sensitive. Secondly, alternative methods such as `MinMaxScaler`—which normalizes data to the $[0, 1]$ range—would be suboptimal in this context due to the presence of substantial outliers. Given the decision to retain these outliers for their informational value, min-max normalization would disproportionately compress the majority of

the data, thereby hindering the model's ability to detect meaningful patterns. Moreover, since this is a multiclass classification problem, it is especially important that the features maintain balanced representations to prevent bias toward any particular class arising from differences in scale. Standardization using `StandardScaler` ensures that all variables contribute proportionately to gradient computations and enhances the numerical stability during model training.

Furthermore, categorical variables—such as day of the week, month of the year, or week number—must be appropriately transformed to be compatible with machine learning algorithms. A widely adopted technique is *one-hot encoding*, which generates a separate binary indicator column for each category. While this approach preserves categorical information without imposing an arbitrary ordinal relationship among categories, it also increases the dimensionality of the dataset significantly. This expansion may result in prolonged training times and greater memory requirements. Moreover, in the case of temporal variables, *one-hot encoding* does not adequately capture their cyclical nature, as it fails to represent the proximity between values such as December and January, or Monday and Friday. To address this limitation, we adopted a **cyclical encoding** approach using trigonometric functions. This technique transforms each discrete temporal variable x into two continuous features through the sine and cosine of its normalized value, defined as follows:

$$\text{sin\_cyclic}(x) = \sin\left(2\pi \cdot \frac{x}{T}\right), \qquad \text{cos\_cyclic}(x) = \cos\left(2\pi \cdot \frac{x}{T}\right)$$

where T denotes the total period of the cycle (e.g., $T = 5$ for weekdays, $T = 12$ for months, and $T = 52$ for weeks of the year). This transformation effectively maps the discrete values onto the unit circle, preserving their cyclical continuity and the natural closeness between the endpoints of each cycle. Cyclical encoding offers significant advantages in capturing the inherent continuity of cyclical variables, thereby avoiding artificial discontinuities and enabling machine learning models to exploit angular relationships and seasonal patterns. Furthermore, since the sine and cosine transformations produce values naturally bounded within the interval $[-1, 1]$, this encoding preserves scale consistency with the remainder of the dataset. This is particularly convenient given that the other variables were standardized using `StandardScaler`, which centers features around zero with unit variance, often resulting in values distributed within a comparable range.

As a consequence of this preprocessing step, the final dataset, `Tesis_data_normalized.csv`, contains three additional columns, resulting in a total of 26 columns and 17,472 rows.

2.5 Exploratory Data Analysis (EDA)

Prior to model implementation, it is essential to gain an initial understanding of the dataset and the distribution of its variables. The primary objective of the Exploratory Data Analysis (EDA) is to identify key patterns in stock behavior over time.

In this section, particular attention is drawn to the pronounced class imbalance in the target variable, *daily_return (%)*, with Tesla emerging as the predominant winner. Furthermore, 70% of the winning stocks in the test set are concentrated among Tesla, NVIDIA, and Apple. This imbalance represents a significant challenge that subsequent predictive models must address.

Additionally, the distribution of daily returns exhibits a positive mean close to zero, accompanied by a substantial standard deviation (1.93%), indicating elevated volatility among the highest daily performers. Extreme values, both positive and negative, were also observed, suggesting the presence of isolated events that substantially impact stock prices.

The visualizations generated—such as histograms, boxplots, and time series plots—reveal no discernible behavior or pattern that would facilitate the prediction of the next day's winning stock, which is the central objective of this study. This lack of obvious patterns underscores the necessity of employing machine learning models to effectively address the problem.

Conversely, heatmaps depicting the correlations among the various dataset variables, as well as between daily returns and individual stocks, reveal noteworthy insights. A particularly strong collinearity is observed between Microsoft and the daily returns of nearly all other stocks, with the notable exception of Tesla. This pronounced correlation suggests that certain models may struggle to adequately differentiate between Microsoft and other stocks such as Apple or Google. Accordingly, during the model implementation phase (Section 4), the same models will also be evaluated after excluding Microsoft to determine whether predictive performance improves. The correlation heatmap also reveals that many variables exhibit high intercorrelations. Such multicollinearity can pose significant challenges for certain models, particularly those based on multivariate linear frameworks, such as the VARMAX model (Vector Autoregressive Moving Average with eXogenous variables). Elevated collinearity among predictors may result in instability of coefficient estimates, information redundancy, overfitting, and a marked deterioration in the model's predictive accuracy. Consequently, a variable preselection was performed for the VARMAX model, retaining only those variables whose pairwise correlations are below 80%. This approach enhances model robustness and mitigates the adverse effects associated with multicollinearity. In contrast, the other models employed in this study benefit from utilizing the complete set of features. Further details are provided in Section 3.2.2. To ensure a fair comparison between models, an additional version of each model was implemented using the reduced feature set established for the VARMAX model (i.e., excluding variables with correlations exceeding 80%). This approach enables the evaluation of whether performance differences arise from the model architecture itself or from the selection of input features.

Chapter 3

Models

This section introduces different models employed to address the prediction of the winning stock for the next day. Classic methods studied in the literature, such as ARIMA and LSTM models [11, 27], combined with other models like Logistic Regression or Random Forest [3, 16], are analyzed with the objective of better understanding the nature of the problem, its complications, and having a range to compare different results. Finally, the Transformers model is detailed and implemented. Below, the models are briefly described.

3.1 Background

The models used can be differentiated between continuous models and classification models. In the first case, we employ ARIMA, VARIMA, and VARMAX. While in the second, we have implemented multinomial logistic regression, Random Forest, and LSTM models. Later, Transformers Models will be introduced.

3.1.1 ARIMA

The ARIMA (Autoregressive Integrated Moving Average) model was proposed in the 1970s by Box and Jenkins. It is a classical tool to model time series that attempts to capture the internal dynamics of an observed series over time. Its name comes from the combination of two models: AR (Autoregressive model, which uses past values of the series) and MA (Moving Average model, which uses past prediction errors). The letter I in its name indicates the level of integration of the analyzed variable. Integrated variables are variables that can become stationary through differencing, enabling the model to handle non-stationary datasets. This is very important as ARIMA models require stationary data to process time series datasets.
It is denoted as ARIMA(p, d, q) where:

- p: number of lags of the series itself.
- d: number of differences required to make the series stationary.
- q: number of lags of the prediction errors.

The daily return is calculated as follows:

- **AR (p)**: the return on day t depends on the returns of the last p days.

$$r_t = \phi_1 r_{t-1} + \phi_2 r_{t-2} + \cdots + \phi_p r_{t-p} + \varepsilon_t$$

where r_{t-i} represents the daily return on the previous i-th day and ϕ_i, for $i = 1, 2, \ldots, p$, are weights.

- **MA (q)**: today's return also depends on previous prediction errors.

$$r_t = \cdots + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

where ε_{t-i} represents the prediction errors on the previous i-th day and θ_i, for $i = 1, 2, \ldots, q$, are weights.

Then, the predicted daily return (%) is given by:

$$r_t = \phi_1 r_{t-1} + \phi_2 r_{t-2} + \cdots + \phi_p r_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

3.1.2 VARIMA

The VARIMA (*Vector Autoregressive Integrated Moving Average*) model is a multivariate extension of the ARIMA model, designed to model and predict multiple correlated time series simultaneously. Let $y_t = [r_t^{(1)}, r_t^{(2)}, \ldots, r_t^{(k)}]^\top$ be the vector of daily returns of k stocks on day t. The VARIMA model of order (p, d, q) can be written as:

$$\Phi(B)(1 - B)^d y_t = \Theta(B)\epsilon_t$$

where:

- B is the lag operator: $By_t = y_{t-1}$,
- $(1 - B)^d$ represents the differencing operator of order d (in this case $d = 0$, since returns are already stationary),
- $\Phi(B) = I - \Phi_1 B - \cdots - \Phi_p B^p$ is the autoregressive polynomial,
- $\Theta(B) = I + \Theta_1 B + \cdots + \Theta_q B^q$ is the moving average polynomial,
- ϵ_t is a vector of innovations with zero mean and constant covariance.

Before training a VARIMA(p, d, q) model, as in the ARIMA model, it is necessary to determine the optimal values of p (autoregressive order), d (number of differences, $d = 0$ in our case), and q (moving average order).

3.1.3 VARMAX

The VARMAX (*Vector Autoregressive Moving Average with eXogenous variables*) model is an extension of the VARIMA model that allows incorporating not only the temporal dependence among multiple multivariate series, but also the effect of additional exogenous variables, i.e., external variables that can influence the behavior of the target series but are not predicted by the model. This type of model is especially useful when the goal is to predict a multivariate time series considering not only its own past values and prediction errors but also other auxiliary variables that may explain its evolution.

A VARMAX model of order (p, q) can be expressed as:

$$y_t = \sum_{i=1}^{p} \Phi_i y_{t-i} + \sum_{j=1}^{q} \Theta_j \varepsilon_{t-j} + Bx_t + \varepsilon_t$$

where:

- y_t is the vector of target variables at time t (daily returns of the stocks),
- x_t is the vector of exogenous variables at time t (technical and temporal features),
- $\Phi_i \in \mathbb{R}^{k \times k}$ are the autoregressive coefficient matrices,
- $\Theta_j \in \mathbb{R}^{k \times k}$ are the moving average coefficient matrices,
- $B \in \mathbb{R}^{k \times d}$ is the weight matrix of the exogenous variables,
- $\varepsilon_t \sim \mathcal{N}(0, \Sigma)$ is a vector of errors (innovations) with zero mean and covariance matrix Σ.

As with ARIMA and VARIMA, before training a VARMAX(p, q) model, it is necessary to determine the optimal values of p (autoregressive order) and q (moving average order).

3.1.4 Multinomial Logistic Regression (LR)

Multinomial Logistic Regression is a model that estimates the probability that each class is the winner, then choosing the class with the highest probability. The following notation must be introduced to mathematically explain the model:

$$W \in \mathbb{R}^{c \times d} \quad \text{and} \quad b \in \mathbb{R}^{c}$$

where c is the number of classes of the model, d are the features of the model, and W and b are the model parameters (weights and biases per class respectively).
The probability for each class c is estimated as:

$$z_c = w_c^\top x + b_c$$

$$P(y = c \mid x) = \frac{\exp(w_c^\top x + b_c)}{\sum_{j=1}^{7} \exp(w_j^\top x + b_j)}$$

where w_c is the weight vector corresponding to class c.
The prediction is the class with the highest probability:

$$\hat{y} = \arg\max_c P(y = c \mid x)$$

The model is trained minimizing the *log-loss* function:

$$\mathcal{L} = -\frac{1}{N} \sum_{t=1}^{N} \log P(y^t \mid x^t)$$

where N is the total number of examples in the training set.

3.1.5 Random Forest Classifier (RF)

Random Forest Classifier is based on an ensemble of decision trees $\{h_m\}_{m=1}^{M}$, where each $h_m(x)$ produces an individual prediction. Each tree is trained with a random sample of the training set and uses a random selection of features at each node. The final prediction of the model is the class most voted among all trees:

$$\hat{y}^t = \text{mode}\left\{h_1(x^t), h_2(x^t), \ldots, h_M(x^t)\right\}$$

The loss function is not explicitly optimized as in linear models, since Random Forest is a non-parametric method. However, the objective is to minimize the classification error through majority voting, combining weakly correlated trees to improve generalization. The training of the model includes:

- Construction of M trees using the entire training set directly, without sampling with replacement, to avoid breaking the temporal structure of the series.
- At each node, a random subset of features is selected to find the best split.

3.1.6 LSTM

The LSTM model (*Long Short-Term Memory*) [28] is a deep learning method capable of identifying structure and pattern of data such as non-linearity and complexity in time series forecasting. It is a type of RNN (Recurrent Neural Network) introduced in 1997 by Hochreiter and Schmidhuber [10] with the capability of remembering values from earlier stages for future use. They were designed to overcome the main limitations of RNNs, especially in tasks that involve long sequences of data. Before delving into how an LSTM works, it is convenient to briefly review what an artificial neural network (ANN) is and how it evolved into RNNs.

An artificial neural network (ANN) generally consists of three types of layers: an input layer, one or more hidden layers, and an output layer. Each layer is composed of units or neurons connected to each other through synaptic weights. These weights determine the relative importance of each connection in the propagation of information. The network learns by adjusting these weights through an iterative process known as error backpropagation, with the goal of minimizing a cost function, usually associated with the difference between the predicted and the actual output. Activation functions such as sigmoid or hyperbolic tangent are applied in the hidden layers to introduce non-linearity and allow the network to learn complex representations.

Recurrent Neural Networks (RNNs) are designed to handle sequential data. Unlike traditional ANNs, RNNs incorporate recurrent connections that allow the output of a neuron at one time step to affect the input of the same or another neuron at the next time step. This architecture enables them to capture temporal dependencies in sequences, which is essential for tasks like time series forecasting or natural language processing. However, traditional RNNs suffer from vanishing and exploding gradient problems when trying to learn long-term dependencies, making effective learning difficult in long sequences.

Therefore, Long Short-Term Memory (LSTM) networks were introduced. Their internal structure includes a memory cell responsible for preserving relevant information over time and three gates that regulate the flow of information: the forget gate, the input gate, and the output gate.

- **The forget gate** decides which old information should be discarded from the memory cell.
- **The input gate** regulates which new information should be added to the memory.
- **The output gate** determines which part of the cell's content will be used to generate the current output.

Each of these gates is based on a sigmoid function, generating values between 0 and 1, smoothly controlling the passage of information. A value of 0 represents passing no information, and 1 represents passing all the information; for example, 0.3 means that 30% of the information passes through. This architecture allows LSTMs to retain relevant information over long periods, effectively solving tasks where temporal dependencies are complex or prolonged. The clever design of LSTMs has made them one of the standard architectures for problems such as time series forecasting, language modeling, or pattern recognition in biological sequences. They have also become one of the most widely used models in stock price prediction in recent years. Below, Figure 3.1 presents a schematic to visually clarify these concepts:

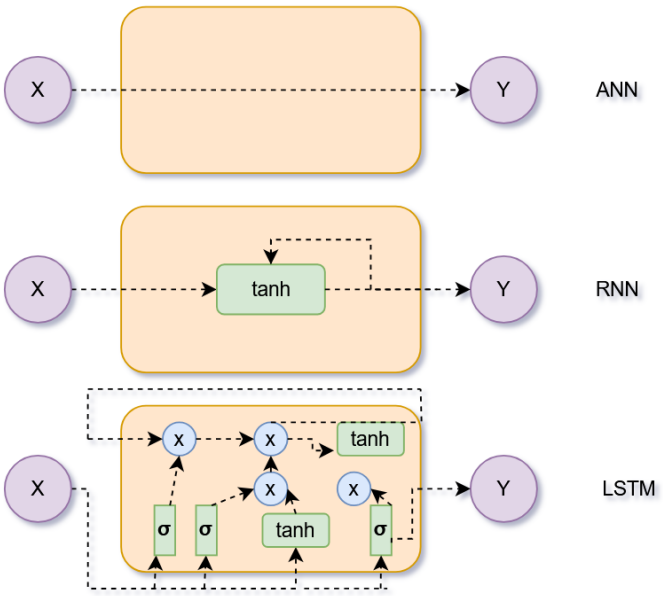

Figure 3.1: Comparative diagram of the architectures of Artificial Neural Network (ANN), Recurrent Neural Networks (RNN), and LSTM, where X is the input and Y the corresponding output. Source: Own elaboration

3.1.7 Transformer Models

Transformer models have revolutionized the field of machine learning, especially in natural language processing (NLP) and, more recently, in time series tasks, computer vision, and sequential prediction in general. Introduced by Vaswani et al. in 2017 in the paper *Attention is All You Need* [29], *Transformers* represent a paradigm shift by completely dispensing with recurrent mechanisms (such as in LSTM or GRU) and basing their architecture on an attention mechanism that allows modeling long-term dependencies more efficiently.

The fundamental component of a *Transformer* model is the *self-attention* mechanism, which allows each element in a sequence to consider all other elements when building its contextualized representation. Unlike recurrent networks, which process information sequentially and thus limit parallelization, Transformers process the entire sequence simultaneously, significantly speeding up training and facilitating modeling of long-range relationships. Given a set of input vectors $X = (x_1, x_2, \ldots, x_n)$ of dimension d, the attention mechanism transforms these vectors into three representations: *queries* (Q), *keys* (K), and *values* (V), through multiplications by trainable matrices:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

where W^Q, W^K, and W^V are learnable parameter matrices of dimension $d \times d_k$, where d_k is the dimensionality of the space in which queries and keys are compared. Then, attention is calculated as a weighted combination of the *values*, where the weights come from the dot product between the *queries* and the *keys*, normalized by *softmax*:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

This mechanism is extended with multi-head attention, where multiple attention operations are performed in parallel with different linear projections, allowing the model to capture different types of relationships simultaneously. The outputs of each head are concatenated and projected again to the original dimension.
Each Transformer encoder (or decoder) block is composed of:

- A *multi-head attention* layer followed by layer normalization and a residual connection.
- A fully connected *feed-forward* network applied individually to each position, also followed by normalization and residual connection.

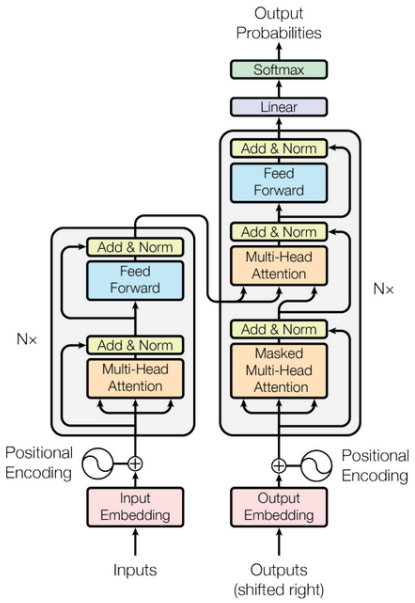

Figure 3.2: Diagram of the Transformer model. Image taken from [29].

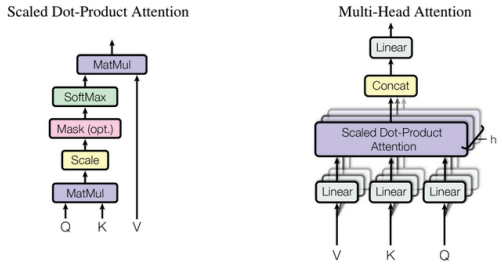

Figure 3.3: (Left) Scaled Dot-Product Attention. (Right) Multi-Head Attention. Image taken from [29].

Since the Transformer architecture lacks an explicit structure that incorporates the temporal or positional order of sequence elements (as RNNs would), a positional encoding is explicitly added. These encodings, which can be fixed (e.g., sinusoidal) or learnable, are added to the inputs before the first attention block to provide the model with information about the relative or absolute position of the sequence elements.

In summary, the Transformer combines multi-head attention, parallel processing, and positional encodings to capture complex dependencies in sequences and has proven to be an extremely powerful architecture in tasks requiring sequential modeling.

3.2 Methodology

This section outlines the methodology adopted in the case study of predicting the next day's winning stock for each of the previously described models. The objective of the models is to perform a multiclass classification among seven stocks, identifying the one with the highest non-absolute value of the target variable *daily_return (%)*, as defined in Equation 2.1. The section begins by detailing the data splitting strategy used for training and evaluation purposes. Subsequently, the specific methodology employed for each model is described. For clarity, the models are categorized into two groups: Continuous Models and Classification Models. Finally, the models based on Transformer architectures are introduced.

3.2.1 Data Split

A proper division of the dataset into training, validation, and test subsets is essential for the reliable implementation and evaluation of statistical or Machine Learning models. The splitting strategy adopted in this work is as follows:

1. **Training set:** Comprises of all available data from 2015 to the end of 2022, representing approximately 75% of the total dataset.

2. **Validation set:** Includes data from January 2023 to June 2024 inclusive, accounting for approximately 15% of the data.

3. **Test set:** Contains the remaining data, from July 2024 to early May 2025, and represents around 10% of the overall dataset.

This chronological split ensures a realistic and forward-looking evaluation framework, in line with common practice in time-series forecasting and financial applications. The training set is used to fit the models, the validation set to fine-tune hyperparameters and select the best configuration, and the test set to evaluate the final performance. Specifically, after identifying the best configuration based on validation performance, models are retrained using the combined *training+validation* set and subsequently tested on the test set. The selected dates aim to align with natural calendar boundaries (mid-year and year-end) to enhance interpretability and replicability. Furthermore, this partitioning adheres to the standard proportions typically adopted in the literature, where training sets usually constitute 70–80% of the data, validation sets 10–15%, and test sets 5–10% [1]. To assess performance during model selection, the metric used is classification accuracy on the validation set, defined as:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{3.1}$$

Accuracy is chosen as it directly reflects the objective of this study: correctly identifying the winning stock as often as possible. The next subsection presents the specific implementation details for each of the proposed models.

3.2.2 Continuous Models

In this category of models, rather than addressing a multiclass classification problem, the numerical value of the daily return for each stock is directly predicted. Subsequently, the stock with the highest predicted daily return is selected as the winner. This section focuses on models that rely exclusively on the target variable, *daily_return (%)*, employing ARIMA and VARIMA methodologies. In subsequent analyses, additional explanatory variables will be incorporated through models such as the VARMAX. The models considered in this section are described in detail below.

ARIMA

In the specific context of this study, the parameters of the ARIMA model were selected as follows:

- p: number of lags of the time series. Values between 1 and 30 were evaluated, with the best results obtained for $p = 1$.

- d: number of differencing operations required to achieve stationarity. As the daily return time series are already stationary, we set $d = 0$.

- q: number of lags of the forecast errors. Integer values from 1 to 5 were considered, with the best performance observed for $q = 1$.

The ARIMA model is applied individually to the daily return (%) time series of each of the seven stocks. The stock with the highest predicted daily return is then selected. The optimal parameters identified for each stock are presented below:

Stock	**p**	**d**	**q**
MSFT	1	0	1
TSLA	0	0	1
AAPL	0	0	1
NVDA	1	0	0
META	1	0	1
AMZN	1	0	1
GOOG	1	0	1

Table 3.1: Optimal ARIMA parameters for each stock

For the sake of consistency and to facilitate a more accurate comparison among stocks, it was decided to employ the same parameter configuration for all series. Specifically, the parameters $(1, 0, 1)$ were selected for this purpose. Subsequently, the model was also implemented using the optimal parameters identified individually for each stock. In both configurations, two distinct approaches were evaluated: one in which predictions were made on the test set without updating the model with new observations, and another in which the model was retrained after incorporating each new data point. The best overall performance was achieved using the $(1, 0, 1)$ configuration without retraining. Nevertheless, it is worth noting that the model failed to adequately capture the behavior of Amazon's time series. This modeling approach is relatively simple and does not account for the temporal interdependence among different stocks. For this reason, a multivariate extension, VARIMA, was implemented in the following section.

VARIMA

The VARIMA model allows for capturing dynamic interdependencies between the daily returns of different stocks. In this study, a VARIMA model is employed to predict which stock will exhibit the highest daily return (%) on the following day, using solely the historical daily returns of each stock as input variables. Prior to training a VARIMA(p, d, q) model, it is essential to determine appropriate values for the autoregressive order p, the order of differencing d (set to 0 in our case, given the stationarity of the series), and the moving average order q. Although an exhaustive hyperparameter search across various combinations of p and q was initially considered—as done with the univariate model ARIMA—such an approach was ultimately constrained by the high computational cost involved, which could not be addressed without access to a more powerful computing environment. Consequently, the selection of model parameters was based on the analysis of the autocorrelation function (ACF) and partial autocorrelation function (PACF).

Autocorrelation Function (ACF).
Let $y_t \in \mathbb{R}^k$ denote a weakly stationary multivariate time series. In our case, the series of daily returns are stationary, as previously verified during the implementation of the ARIMA model. The autocorrelation function of the i-th component of y_t, denoted by $y_t^{(i)}$, quantifies the correlation between the values of that component and its own lagged values. For each lag h, it is defined as:

$$\rho^{(i)}(h) = \frac{\mathrm{Cov}(y_t^{(i)}, y_{t-h}^{(i)})}{\mathrm{Var}(y_t^{(i)})} = \frac{\mathbb{E}[(y_t^{(i)} - \mu_i)(y_{t-h}^{(i)} - \mu_i)]}{\sigma_i^2}$$

where μ_i and σ_i^2 denote the mean and variance of the i-th component of the series, respectively. The ACF is instrumental in identifying temporal dependencies in the residuals and is primarily used to estimate the order q of the moving average component of the model.

Partial Autocorrelation Function (PACF).
The partial autocorrelation function measures the direct correlation between $y_t^{(i)}$ and $y_{t-h}^{(i)}$ after removing the influence of the intermediate lags $1, \ldots, h-1$. For a given lag h, the PACF is defined as the coefficient $\phi_{hh}^{(i)}$ in the following linear regression model:

$$y_t^{(i)} = \phi_{h1}^{(i)} y_{t-1}^{(i)} + \phi_{h2}^{(i)} y_{t-2}^{(i)} + \cdots + \phi_{hh}^{(i)} y_{t-h}^{(i)} + \varepsilon_t$$

Thus, the partial autocorrelation at lag h is given by:

$$\mathrm{PACF}^{(i)}(h) = \phi_{hh}^{(i)}$$

The PACF is particularly useful for determining the order p of the autoregressive component of the model.

Relationship with the VARIMA model.
Given that the VARIMA(p, d, q) model adopted in this study is expressed as:

$$\Phi(B)(1 - B)^d y_t = \Theta(B)\epsilon_t$$

where y_t denotes the vector of daily returns of the various stocks, the analysis of the ACF and PACF of each component $y_t^{(i)}$ is fundamental for estimating appropriate values for both the autoregressive and moving average orders of the model. In particular:

- A marked decline in the PACF after lag p suggests the suitability of an autoregressive component of order AR(p).
- A sharp drop in the ACF after lag q indicates the presence of a moving average component of order MA(q).

In our case, the PACF applied to the individual return series of each stock exhibited a significant decline after the first lag, indicating an autoregressive order of $p = 1$. Similarly, the ACF plots revealed strong correlations at the first lag, followed by a rapid decay, supporting the inclusion of a moving average component of order $q = 1$.

Since the return series are already stationary, no additional differencing was required, leading to the choice of $d = 0$. Consequently, the final model implemented was a VARIMA(1,0,1), which captures both autoregressive and moving average dependencies across the multiple return time series. Although this model is more computationally demanding than simpler alternatives, it offers a flexible framework to jointly model the dynamics of the different stocks. It is important to note that, in this case, the model was trained directly on the *train+val* dataset, as the hyperparameters had already been selected in advance.

In parallel, a VARIMA(p,0,0) model—excluding the moving average component—was also evaluated. A rolling window approach was adopted, varying p from 1 to 30. The best

performance was achieved with $p = 7$. Among both configurations, the VARIMA(7,0,0) model yielded the highest accuracy, reaching 20.19%. Nonetheless, the model failed to detect Google and Microsoft in its predictions, resulting in zero values for precision, recall, and F1-score in both cases. Once again, the model proves to be overly simplistic, as it does not incorporate other relevant time series that may influence the daily returns. Therefore, we proceed to explore a slightly more sophisticated model that integrates additional time series from the dataset.

VARMAX

The VARMAX model (*Vector Autoregressive Moving Average with eXogenous variables*) extends the capacity to model temporal dependencies among multiple multivariate time series by incorporating the influence of additional exogenous variables.

Let $y_t \in \mathbb{R}^k$ represent the vector of target series — in this study, the daily returns `daily_return (%)` of $k = 7$ stocks — and let $x_t \in \mathbb{R}^d$ denote the vector of associated exogenous variables. It is important to note that not all available variables were included due to the presence of high collinearity among many of them. High collinearity adversely affects model stability and interpretability, and may lead to overfitting during training. Therefore, following an initial approach that utilized all variables available in the dataset, a subset of features exhibiting low mutual correlation was selected. Based on the heatmap analysis conducted during the exploratory data analysis (Section 2.5), the following features were retained: `daily_return (%)`, `Close/Last`, `Volume`, `RSI`, `MACD`, `MACD_Signal`, `MACD_Histogram`, `%K`, `Week_day_sin`, `Week_day_cos`, `Month_sin`, `Month_cos`, `Week_sin`, and `Week_cos`.

Prior to training the VARMAX(p, q) model, it is necessary to determine the optimal values of the autoregressive order p and the moving average order q. In this study, various combinations of p and q were initially considered; however, the high computational cost associated with exhaustive hyperparameter tuning precluded a comprehensive comparison in the absence of more powerful computational resources. Consequently, analogous to the approach taken for the VARIMA model, the selection of optimal parameters was based on the analysis of the autocorrelation and partial autocorrelation functions.

Relationship with the VARMAX model.

The VARMAX(p, q) model employed in this study is defined as:

$$y_t = \sum_{i=1}^{p} \Phi_i y_{t-i} + \sum_{j=1}^{q} \Theta_j \varepsilon_{t-j} + Bx_t + \varepsilon_t$$

where analyzing the autocorrelation function (ACF) and partial autocorrelation function (PACF) of each component $y_t^{(i)}$ provides initial guidance for selecting appropriate lag orders in both the autoregressive and moving average components:

- A pronounced decay in the PACF after lag p suggests the presence of an AR(p) component.
- A marked decline in the ACF after lag q indicates an MA(q) component.

In the context of the VARMAX model, this analysis applies similarly as the ACF and PACF relate exclusively to the daily return series. Based on this assessment, a VARMAX(1,1) model was fitted, capturing both autoregressive and moving average dependencies.

For the model implementation, the exogenous variables were standardized and incorporated alongside the daily returns of the stocks. The VARMAX(1,1) model attained an accuracy of 16.04%, which represents a decline in performance relative to the simpler models previously evaluated. Furthermore, the model predicted only two stocks as winners: AAPL and NVDA.

Although the VARMAX model enables the incorporation of exogenous variables to enhance the information available for modeling the system, its predictive performance can be adversely affected when the time series exhibit a strong and dominant autoregressive temporal structure, as is frequently observed in financial data. In such cases, models like ARIMA or VARIMA, which focus exclusively on exploiting the endogenous dependencies within and between series, are often better suited to capture the relevant dynamics with reduced complexity and improved stability in parameter estimation. Moreover, VARMAX models require the simultaneous estimation of multiple interdependent equations, which may result in the propagation of modeling errors across equations, even when the exogenous variables exhibit weak correlations. This inter-equation error transmission can lead to deteriorated predictive accuracy, as errors in one variable's forecast negatively impact the predictions of others. Therefore, while the VARMAX model offers a compromise between predictive power and computational complexity, providing interpretability by explicitly modeling both inter-series dependencies and the influence of external factors, it does not achieve high accuracy in this context and proves to be inadequate for the problem at hand.

3.2.3 Classification Models

In this case study, the problem is formulated as a multiclass classification task. The objective is to identify, from a set of seven stocks, the one that will yield the highest return on the following day. The winning stock is defined as the one with the highest *non-absolute* daily return. To address this objective, the initial implementation of the models used the time series of the target variable, `daily_return (%)`, as the sole input feature. This first approach was applied to the Random Forest and Multinomial Logistic Regression models. Subsequently, these models were extended to incorporate all available variables in the dataset. This was done with the aim of enabling a direct comparison with the continuous models presented earlier—namely, ARIMA and VARIMA—which also rely exclusively on `daily_return (%)` as an input feature, as well as to assess the performance improvement of each classification model when additional features are introduced.

To implement the initial approach using only `daily_return (%)` as input, matrices were constructed where each row corresponds to a stock, and each column represents the daily return of that stock on successive days. Specifically, the first column contains the daily return on day one, the second column on day two, and so on. These matrices serve as the input to the classification models. To explore the impact of different input window sizes, the number of columns—corresponding to the number of past days considered—was varied from 1 to 30. The goal was to identify the window length that yields the highest predictive accuracy.

Additionally, the following notation is established for both models:

- $y^t \in \{0, \ldots, 6\}$: target variable for day t, indicating the index of the stock with the highest daily return on that day.
- $x^t \in \mathbb{R}^d$: feature vector for day t, where $d = 7 \times n$, representing the daily returns of all seven stocks over the previous n days.

The implementation details for each model are described below:

Multinomial Logistic Regression

Case I: Only *daily_return* (%)

In this setting, the model is trained to learn patterns from historical daily returns in order to predict which stock will be the winner on the following trading day. A hyperparameter optimization process is conducted, considering the following search space:

- **C**: {0.01, 0.1, 1, 10}
- **solver**: {lbfgs, saga}
- **max_iter**: {500, 1000}

Furthermore, four distinct implementations are evaluated to identify the most effective configuration. These variations account for the presence or absence of class balancing, as well as whether or not the model is retrained using the new test set data. The best performance is achieved by the model without retraining and without class balancing, with the following configuration: $n = 3$ (i.e., three previous days), $C = 0.1$, solver *lbfgs*, and $max\_iter = 500$. This setup yields an accuracy of 26.29% on the test set. Specifically, when $n = 3$, each input vector x has $3 \times 7 = 21$ features. Additionally, the loss values obtained for both the validation and test sets are lower than those associated with a uniform (random) prediction, indicating that the model provides meaningful predictions. Nonetheless, a significant limitation is observed: the model predominantly predicts only two stocks—Tesla and NVIDIA—regardless of the true class. This suggests a lack of generalization capacity and restricts its practical applicability.

Case II: With all features

This model extends the previous multinomial logistic regression by incorporating the full set of financial variables available in the dataset for each stock. Instead of relying solely on past returns, we construct the feature vector $x \in \mathbb{R}^d$, with $d = 7 \times 24$, encapsulating the information from all features for each stock at a given point in time. Accordingly, rather than having one row per stock and date, all the variables associated with the seven stocks are aggregated into a single row corresponding to each common date. All features, except for the temporal ones, have been previously standardized using the *StandardScaler* to enhance both the stability and efficiency of the optimization process. In this framework, various sliding windows are tested, i.e., the number of previous days the model utilizes to predict the next day's winning stock ($n = 1, 5, 10, \ldots, 45, 60, 90$). Within each window, the average of each feature is computed in order to reduce noise and better capture short-

and medium-term trends. The optimal hyperparameter configuration—same search space as in Case I—is selected using GridSearchCV.

Five configurations are explored. First, we compare balanced versus unbalanced classes, observing a clear performance advantage when no class balancing is applied. Second, under the unbalanced setting, the model is trained using only the features derived from the VARMAX model (see Subsubsection 3.2.2, VARMAX) to allow for a consistent comparison across modeling strategies. Furthermore, two retrained versions are implemented, wherein the model is updated with each newly available real value in the test set. This retraining is carried out for both the full feature set and the VARMAX-derived features.[1] The best performance is achieved by the retrained model using all features, without class balancing, attaining an accuracy of 27.83% with the optimal hyperparameters: `'C': 0.01`, `'max_iter': 500`, and `'solver': 'lbfgs'`. Nevertheless, it is noteworthy that this configuration only succeeds in predicting three classes: `AAPL`, `NVDA`, and `TSLA`.

Random Forest

Case I: Only daily_return (%)

In the case where only the daily_return (%) is used as a variable, as in the previous model, four implementations are carried out: with balanced and unbalanced classes, and retraining the model with the newly known data from the test set and without retraining. In addition, to find the best configuration, a grid search (*GridSearchCV*) is performed over the hyperparameters :

- `n_estimators` $\in \{100, 200\}$
- `max_depth` $\in \{\text{None}, 10, 20\}$
- `min_samples_split` $\in \{2, 5\}$

Better results are obtained with the model without class balancing and without retraining, consistent with the findings from the previous case. The optimal number of days is $n = 21$, meaning that each feature vector x^t comprises $7 \times 21 = 147$ elements. With the best hyperparameters—`max_depth`: 20, `min_samples_split`: 2, and `n_estimators`: 200—the model achieves an accuracy of 27.23% on the test set.

Unlike the logistic regression model, the Random Forest algorithm can capture nonlinear relationships between past returns and the winning stock, resulting in reduced bias toward certain classes. Nevertheless, in our case, the model still consistently predicts only TSLA and NVDA as winners. This indicates that, despite its greater flexibility, the approach remains insufficient to capture the full complexity of the problem.

Case II: With all features

In this model, the Random Forest classifier is extended by using all available features per stock. The dataset is structured by date, where each row represents the set of features of all stocks on a given day, following the same structure as in the Logistic Regression model with all features. The difference is that in this model, when adding time windows,

[1] Although retraining would be desirable for all models, due to the high computational cost involved, it has only been feasible in this case for the full-feature and VARMAX-feature versions.

instead of using rolling windows (i.e., the average of each feature), we keep the values of each feature and stock on the corresponding day. This procedure is implemented using the `create_lagged_features` function, which builds a new data representation from a fixed-size time window w. For each row corresponding to day t, the values of all features from the previous w days, including the current day, are gathered. These values are concatenated into a single vector, preserving the temporal sequence, allowing the model to capture short-term behavioral patterns among the different financial stocks. For example, if a window of size $w = 3$ is used, the input for day t will consist of the concatenation of the values of all features from days $t-2$, $t-1$, and t. In this way, a feature matrix is generated where each row represents a temporal block of i consecutive days, with the complete information of all available stocks. Visually, it would be:
Suppose we use 3 stocks (AAPL, AMZN, GOOG) and 2 features per stock (Close and Volume). The structure of each daily feature vector is:

$$x_t = \begin{bmatrix} \text{AAPL}_{\text{Close}} & \text{AAPL}_{\text{Volume}} & \text{AMZN}_{\text{Close}} & \text{AMZN}_{\text{Volume}} & \text{GOOG}_{\text{Close}} & \text{GOOG}_{\text{Volume}} \end{bmatrix}$$

To create the lagged feature vector for time t using a window size of 3, we concatenate the vectors of the last 3 days:

$$X_t = \begin{bmatrix} x_{t-2} & x_{t-1} & x_t \end{bmatrix} = \begin{bmatrix} \text{AAPL}_{t-2}^{\text{Close}} & \text{AAPL}_{t-2}^{\text{Volume}} & \dots & \text{GOOG}_{t}^{\text{Volume}} \end{bmatrix}$$

This results in a flattened vector of size $3 \times 3 \times 2 = 18$ features.

That is, in the case study, we define an input vector $x^t \in \mathbb{R}^d$ where $d = N \times w \times d$, with $N = 7$ the total number of stocks, w the window size, and $d = 24$ the number of features per stock, i.e., $d = 7 \times w \times 24$. This strategy is particularly suitable for models like Random Forest, which can handle large numbers of variables without the need to reduce dimensionality. Moreover, it allows the model to detect nonlinear and complex relationships between past values and the target variable. However, in the previous Logistic Regression model, this was not suitable due to the linearity of the model and its high sensitivity to multicollinearity and the high dimensionality of the data, which is why rolling windows were used.

Finally, to train the model, we follow the standard Random Forest approach. Four versions are implemented: with and without class balancing (to mitigate bias toward the most frequent classes), and using all features or only those used in 3.2.2 (VARMAX). The prediction is evaluated on the validation and test sets, seeking to maximize accuracy. To find the best configuration, a grid search (*GridSearchCV*) is performed over the same hyperparameters as in Case I. The best configuration is obtained with the model without class balancing and using the VARMAX features. The model achieves an accuracy of 26.42% on the test set, using a time window of 45 days and the best hyperparameters: `max_depth`: 20, `min_samples_split`: 5, and `n_estimators`: 100.

This model effectively leverages the combined information from all stocks, capturing multivariate relationships and global market patterns. Nevertheless, as in the previous model, some classes are still not predicted correctly, reflecting the high complexity of the problem and the need for complementary strategies to address the classification of multiple financial stocks in real-world settings. Moreover, we observe worse performance compared to Case I, where only the daily_return (%) was used as a feature.

LSTM

Mathematically speaking, and in our specific implementation, the following structure is followed: Let $S = \{s_1, \ldots, s_N\}$ be the set of N stocks and $x_t^{(s)} \in \mathbb{R}^d$ the vector of d technical features of stock $s \in S$ at time t. On each day t, there is a multi-stock feature matrix $X_t \in \mathbb{R}^{N \times d}$. The input of the model is a temporal sequence of window size w days:

$$\mathcal{X}_t = \{X_{t-w}, X_{t-w+1}, \ldots, X_{t-1}\}, \quad \mathcal{X}_t \in \mathbb{R}^{w \times N \times d}$$

Each $\mathcal{X}_t$ is transformed into a sequence of dimension $w \times (N \cdot d)$, which is fed as input into the LSTM network. The goal is to predict a categorical variable $y_t \in \{1, \ldots, N\}$ that represents the index of the stock with the highest daily return on day t, that is:

$$y_t = \arg\max_{s \in S} r_t^{(s)}, \quad \text{where } r_t^{(s)} \text{ is the daily return of } s \text{ at } t.$$

The LSTM model is composed of one LSTM recurrent layer, followed by dense layers and an output layer with `softmax` activation, which produces a probability distribution $p_t \in \mathbb{R}^N$ over the possible assets:

$$p_t = \text{softmax}(f_\theta(\mathcal{X}_t)), \quad \hat{y}_t = \arg\max_i p_t^{(i)}$$

where f_θ represents the neural network parameterized by θ. The model is trained by minimizing the cross-entropy loss between the predictions and the true labels y_t:

$$\mathcal{L} = -\sum_t \sum_{i=1}^{N} \delta_{y_t=i} \cdot \log(p_t^{(i)})$$

where $\delta_{y_t=i}$ is the Kronecker delta.

Additionally, to mitigate the class imbalance (some stocks are more frequently winners than others), class weights inversely proportional to their frequency have been applied. The model was iterated over different values to find the best set of hyperparameters. This iterative process involved testing the combinations of the following parameters:

- **Window sizes**: {5, 10, 15, 20, 25, 30, 35, 40, 45, 60, 90}. These define the number of past time steps used as input to predict the future value.
- **Number of neurons in the LSTM layer(s)**: {16, 32, 64, 128}. This controls the model's capacity to learn temporal patterns.
- **Number of training epochs**: {30, 50}. Each epoch represents a full pass through the training data.
- **Batch sizes**: {16, 32, 64}. This parameter determines the number of samples processed before the model's internal parameters are updated.
- **Number of dense layers**: {1, 2}. These fully connected layers are added after the LSTM layer(s) to transform the extracted features before the final output.

We emphasize that the 5-step increments in window sizes are due to the data being collected on a 5-day cycle, from Monday to Friday.

LSTM models can be trained with various activation functions, loss functions, and optimizers. In our specific case, the following were selected to train the model:

1. **ReLU (Rectified Linear Unit) activation function**: introduces non-linearity and is defined as
$$f(x) = \max(0, x).$$

2. **Softmax activation function**: converts the outputs into probabilities that sum to 1, assigning a probability to each class.

3. **Loss function**: `loss='categorical_crossentropy'`, used for multi-class classification with one-hot encoded labels. This function measures the difference between the predicted and the true distribution.

4. **Optimizer**:
$$\text{optimizer} = \text{Adam}(),$$
an advanced optimization algorithm that adjusts the model weights to minimize the loss. The model predicts the class of each sample based on the highest probability.

5. `shuffle=False` is used, i.e., samples are not shuffled before each epoch, which is useful for sequential data to preserve the temporal order.

It is worth noting that implementing this model required adding more data 2.3.1 and temporal variables. This was due to the fact that the first implementation, using only NASDAQ data from 2021 onwards, revealed the need to increase the dataset size and include more variables due to its high bias towards frequent winners such as Tesla and the class imbalance, as detailed in Section 2.

The model is also trained with standardized variables using *StandardScaler*, except for cyclically encoded variables to preserve their cyclical nature. As detailed in 2.4.3, *StandardScaler* was preferred over *MinMaxScaler* or other normalization techniques due to its better performance in LSTM models, by centering the data and preserving relative variance. This improves numerical stability and the ability to detect temporal patterns, especially in financial series with subtle variations across assets, and reduces the impact of outliers.

The model is implemented using all features and the features from VARMAX. The best result is obtained with all features, achieving 21.14% accuracy on the test set for $w = 90$, 128 neurons, 50 epochs, 32 batch size, and two dense layers. However, an accuracy of 74.37% was achieved on the *train+val* set, highlighting the complexity of the problem. Despite being a more complex model, LSTM is still inefficient in adequately capturing temporal patterns. In addition, after analyzing the best results filtered by those exceeding 23% accuracy, it is observed that the most influential parameters are the number of neurons (with 128 being the best) and larger batch sizes.

3.2.4 Transformers

Although originally designed for natural language processing (NLP) tasks, *Transformer* models have recently been successfully adapted to multivariate time series analysis, leading to specialized architectures such as the *Time Series Transformer* [36], the *Informer* [37], the *Autoformer* [33], and the *Temporal Fusion Transformer (TFT)* [15]. These variants incorporate advanced mechanisms such as hierarchical attention, seasonal series encoders, or integration of known covariates to effectively capture temporal patterns such as seasonality, trends, or the presence of missing values. This section explores

the implementation and evaluation of *Transformer* models introduced by [29] adapted to the multivariate stock prediction problem presented earlier, with the goal of analyzing their performance compared to classical models and recurrent neural networks.

The Transformer model developed in this thesis is based on an encoder architecture (*Transformer Encoder*) specifically designed for the multiclass classification task on multivariate financial data. In particular, the aim is to predict, from a fixed-size temporal window, which stock will have the highest daily return the following day.

Before proceeding with the methodology followed in this model, it is important to dedicate a section to the reasons why this architecture was chosen for this project.

Why use Transformer Models?

The *Transformer* architecture implemented in this work presents multiple advantages that make it highly suitable for the multiclass classification problem posed with financial data. Firstly, our problem can be interpreted as a sequence of winning stocks, where the model's objective is to predict which stock will be the next winner in the sequence. This strategy is analogous to that used by *Natural Language Processing* (NLP) models, where the goal is to predict the next word or *token* in a sentence or document. In our case, instead of words, the *tokens* are representations of financial stocks. However, the objective remains the same: to predict the next *token* (stock) from the previous context. For this reason, Transformer-based models constitute an especially appropriate architecture to address this type of problem.

Moreover, one of its main strengths lies in the model's ability to capture complex and long-range relationships between different variables over time. The *self-attention* mechanism equips the model with the capacity to assign different weights to various temporal moments and features, explicitly learning which parts of the sequence are more relevant for predicting the winning stock. This flexibility is useful in financial contexts, where relationships among variables are not necessarily local or linear, and relevant patterns may arise in variable and unpredictable temporal windows.

Additionally, Transformers do not need to process sequences strictly sequentially, unlike recurrent networks (LSTM). It should be noted, however, that although Transformers are not constrained to process data in temporal order, in this problem temporal order is critical and has been incorporated via positional encodings (in this case, sinusoidal or learnable), allowing the model to distinguish the relative position of each time step within the sequence. This provides the model with the flexibility to capture both local and long-range dependencies.

Furthermore, Transformer architectures enable more efficient parallelization during training compared to sequential models, resulting in a significant reduction in computation time and better scalability when working with large volumes of historical data from multiple stocks.

Another advantage is their structural flexibility. The architecture can be easily adapted to include new explanatory variables or financial indicators without redesigning the model structure. Moreover, Transformers do not assume fixed seasonality or predefined periodicity in the data, which is crucial in such a dynamic and volatile environment as financial markets.

Data

The data used for the Transformer model correspond to our full dataset containing all the features described in Section 3.2.1. In order to ensure a more efficient and stable model training, as in previous models, all numerical features were normalized using the *StandardScaler* method, which transforms each variable to have zero mean and unit standard deviation. This normalization process is important in architectures such as Transformers, which do not impose implicit constraints on the scale of the inputs. Without scaling, some features could dominate the attention mechanism or cause instability during gradient backpropagation. The data splitting into training, validation, and test sets is identical to that used in Section 3, with the aim of maintaining a fair comparison between models. This approach is consistent with previous works on financial time series prediction such as [22], [8].

In addition to using the full feature set, variants of the model were also evaluated using only the variables employed in the VARMAX model (see Section 3.2.2), i.e., those features that are not highly correlated with each other. This selection aims to reduce informational redundancy and mitigate the effect of multicollinearity, which could benefit the model's generalization capability. After implementing the models with both the full features and the VARMAX feature subset, better results were observed using all features. Therefore, subsequent implementations employ the full feature set.

Model Architecture

The model receives as input a time window of w days, where each day is represented by a financial feature vector of dimension d, with $d = 24$ in our case. To generate the input data, the historical prices of each stock are traversed using a sliding window of size w. For each window, its features are extracted as input X_t, and the label is selected as the stock with the highest daily return on day $t + 1$, thus converting the problem into a sequential classification based on historical context. Therefore, the input shape is `batch-first`, $[B, w, d]$, where B is the batch size[2]. Before being processed, this input first passes through a linear layer that projects the original vectors into an intermediate dimension d_{model}, which is the dimension used by the *Transformer* block. This projection allows the *Transformer* block to work in a latent space of higher capacity:

$$\mathbf{X}_{\text{proj}} = \text{Linear}(d, d_{\text{model}})(\mathbf{X}) \quad \Rightarrow \quad \mathbf{X}_{\text{proj}} \in \mathbb{R}^{B \times w \times d_{\text{model}}}$$

Next, the projected sequence is fed into a *Transformer Encoder*, composed of L layers of the type proposed in [29], each including:

- Multi-head attention with h heads, allowing the model to learn dependencies across different days.
- A *feed-forward* network with two linear layers separated by a ReLU activation.
- Layer normalization (*LayerNorm*) and residual connections.
- Dropout with rate p_{drop} applied after the attention and the feed-forward network.

[2]Recall that the batch is the number of samples processed simultaneously by the model at each training or evaluation step, determining how many data points are used to compute the gradient before updating the weights.

These layers are implemented using PyTorch's `nn.TransformerEncoderLayer` class, configured with `batch_first=True` to maintain the format $[B, w, d_{\text{model}}]$. No decoder or causal masking is included, as all past information is available. The encoder's output has shape $[B, w, d_{\text{model}}]$ and is flattened to $[B, w \cdot d_{\text{model}}]$ to be processed by a neural network acting as a classifier. This network consists of a first fully connected linear layer projecting the encoder output to a fixed hidden dimension of size 128, followed by a ReLU activation and a dropout layer with rate 0.3 to reduce overfitting. Finally, a second linear layer projects this representation to the total number of classes C (i.e., the number of different stocks in the dataset). The model's final output is a vector of C logits per sample, to which a *softmax* function is implicitly applied inside the cross-entropy loss function, generating a probability distribution over the candidate stocks. Below is a schematic of the architecture used:

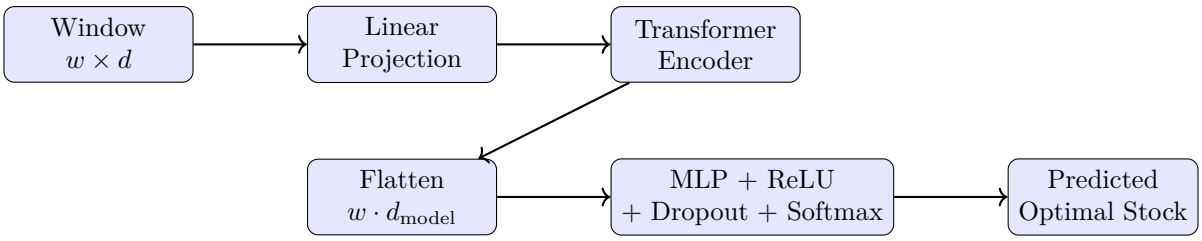

Figure 3.4: Architecture of the *Transformer* model used for multivariate time series classification. Own elaboration.

Positional Encoding

One of the main limitations of *Transformer* models is that, by design, they do not take into account the sequential order of the data. Unlike recurrent neural networks, which process inputs in an ordered manner, Transformers treat sequence elements as a set, which poses a problem in time series tasks where the chronological relationship between data points is essential. To address this limitation, explicit temporal information is incorporated through mechanisms known as *positional encoding*. In this work, two different positional encoding strategies were implemented and compared. The first, known as *sinusoidal positional encoding*, follows the original formulation by Vaswani et al. [29] and consists of adding to each input vector a deterministic signal generated from sine and cosine functions of different frequencies. This non-trainable technique allows the model to infer the relative position between events without introducing additional parameters.

The second strategy implemented corresponds to a *learnable positional encoding*, where a positional embedding matrix is defined as a set of free parameters adjusted during training. This variant allows the model to directly learn the positional representations most suitable for the specific task, adapting more flexibly to complex temporal patterns. Both approaches were integrated into the proposed architecture and evaluated. Better results were obtained for the sinusoidal encoding when comparing models using all features. Therefore, this encoding was chosen for the remaining model versions discussed in the next chapter.

Optimization and Implementation

In order to achieve the best possible performance, a *grid search* was conducted over a set of relevant model hyperparameters. Specifically, different combinations of the following

were explored:

- Window size (w): {5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 90}
- Model dimension (d_{model}), with values {32, 64}.
- Number of multi-head attention heads (n_{head}), taking values {2, 4}.
- Training parameters: learning rate (0.001), batch size ({32}), and number of epochs (100).

For each hyperparameter combination, a model was trained from scratch using the training set and its performance evaluated on the validation set. The *Adam* optimizer, known for its efficiency in deep neural networks, was used during training, and *early stopping* with a patience of 10 epochs was applied, halting training early if no improvement in validation accuracy was observed. This aims to prevent overfitting and speed up the search process.
Once all combinations were explored, the configuration yielding the highest validation accuracy was selected. With the best hyperparameters identified, the final model was retrained using the combined training and validation sets to maximize data usage. Finally, the model was evaluated on the independent test set to measure its generalization capability. Below, Table 3.2 summarizes the parameters used in the model[3].

Table 3.2: Hyperparameters used for the *Transformer* model

Hyperparameter	**Value or Range**
Input size	24
Model dimension (d_{model})	{32, 64}
Number of output classes	7
Window size	{5, 10, ..., 90}
Number of encoder layers	2
Number of multi-head attention heads (*nhead*)	{2, 4}
Hidden layer size in classifier	128
Dropout in encoder	{0.1}
Dropout in classifier	0.3
Loss function	`CrossEntropyLoss`
Optimizer	`Adam`
Learning rate	{0.001}
Batch size	{32}
Training epochs	100
Fine-tuning epochs	3

The analysis of the performance of the parameters iterated over window size, d_{model}, and n_{head} on validation accuracies concludes that the window size has a significant effect on

[3]It is worth noting that, as these are relatively new models and there is limited literature on this problem approach, parameter selection was based on expert judgment and computational capacity. However, guidance was taken from [22], [8].

accuracy improvement, highlighting time periods of 15, 20, and 25 days. However, the period that stands out most is 60 days, which also yielded the best results in the first two model versions. On the other hand, models with representation dimension $d_{\text{model}} = 32$ outperform those with $d_{\text{model}} = 64$. This better performance may be due to the limited number of samples in the dataset; a lower d_{model} is sufficient to capture relevant relationships between features without introducing unnecessary noise or redundancy. Additionally, a higher d_{model} increases the number of model parameters, raising the risk of overfitting, especially in scenarios where the data volume does not justify such high representation capacity. Models with lower dimension are also computationally more efficient, allowing for more stable optimization during training. This translates into faster convergence and more consistent performance across different runs. Meanwhile, n_{head} does not significantly influence the model's accuracies.

It is also noteworthy that most models require no more than 5 epochs to reach their best validation accuracy, with 1 epoch being sufficient in most cases. Therefore, reducing the number of epochs to 5 is a recommended improvement to decrease the computational time of the models.

This model achieves its best results for a 60-day window, $d_{\text{model}} = 32$ and $n_{\text{head}} = 4$, with an accuracy of 23.21% on the validation set.

The next section will present and compare the results of all these proposed models.

Chapter 4

Results and Comparison

This section presents a comprehensive comparison of all the models implemented for the present analysis 3: including classical continuous statistical models (ARIMA, VARIMA, VARMAX), machine learning models (Logistic Regression, Random Forest, LSTM), and the model based on the *Transformer* architecture. The best result obtained by each model is chosen based on the accuracy achieved on the test set. Additionally, the F1-score values[1] for all stocks per model and the average weighted F1-score (Avg Weight), that is, the weighted mean of the F1-scores of all stocks, are compared. Below is the comparative table of the best results obtained by each model:

4.1 All features

Table 4.1: Model comparison by variable type and method

Features	**Type**	**Model**	**F1-socre Stocks**							**Accuracy %**	**Avg Weight**
			AAPL	AMZN	GOOG	META	NVDA	TSLA	MSFT		
Only Daily Return	Classification	LR	0.0000	0.0000	0.0000	0.0000	0.3158	0.3938	0.0000	26.29	0.1702
		RF	0.0000	0.0000	0.0000	0.0000	0.3103	0.4145	0.0000	27.23	0.1745
	Continuous	ARIMA	0.0638	0.0000	0.0784	0.0984	0.2810	0.1750	0.1765	15.96	0.1455
		VARIMA	0.1333	0.1111	0.0000	0.1404	0.2830	0.3130	0.0000	20.19	0.1932
All Features	Classification	LR	0.1091	0.0000	0.0000	0.0000	0.3650	0.3804	0.0000	27.83	0.1983
		RF	0.1224	0.0000	0.0000	0.0000	0.3805	0.2828	0.0000	26.42	0.1790
		LSTM	0.3380	0.1538	0.1538	0.0000	0.3125	0.0500	0.0000	21.14	0.1701
	Continuous	VARMAX	0.2960	0.0000	0.0000	0.0000	0.0357	0.0000	0.0000	16.04	0.0641
	Transformers	Transformers	0.0794	0.0000	0.0531	0.0131	0.2765	0.3550	0.0290	23.21	0.1717

It is observed that, considering continuous models, those that use only daily return as a feature outperform VARMAX, which uses more variables, as explained in 3.2.2. Conversely, regarding classification models, there is no clear pattern favoring either a greater or smaller number of features. Despite the large class imbalance, Logistic Regression (LR) and Random Forest (RF) achieve better results in all cases when class balancing is not applied.

[1]Remember F1-score is given by

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

The model achieving the highest overall *accuracy* is *Logistic Regression* with all variables, obtaining 27.83% on the test set 4.1. This value exceeds by slightly more than 1% the *naive* model that systematically predicts Tesla (the most frequent winning stock, 26.29% of the time). However, from a risk management and portfolio construction perspective, this model is insufficient. Its predictive capacity is almost exclusively concentrated on two classes: Tesla and NVIDIA. This implies that, in a real financial decision-making scenario, the strategy derived from this model would lead to asset concentration in very few stocks, increasing exposure to specific risk and reducing diversification possibilities. This lack of variety in predictions significantly limits the model's usefulness in contexts where diversification is essential to mitigate volatility and maximize the stability of expected returns.

In contrast, the *Transformer* model, although it shows a slightly lower *accuracy* (23.21% on the test set), demonstrates a much more robust ability to predict multiple classes. Except for Amazon, it is able to identify all other stocks as winners in different situations within the test set. This also happens with the ARIMA model. However, both the accuracy and the average weight of the model are lower than those obtained with Transformers.

The ability to predict more stocks represents a crucial advantage from a financial point of view: it allows building a balanced and diversified portfolio, reducing the dependence on a single stock and better adapting to the real dynamics of the market. Moreover, correctly predicting a wider variety of classes reflects the aptitude of Transformer models to capture complex patterns and nonlinear temporal relationships among the different variables in the system, something that simpler models such as LR or RF fail to model properly. In terms of effective portfolio diversification, that is, the possibility of distributing capital among multiple stocks that may lead in different contexts, the Transformer proves to be significantly more powerful. Therefore, although its accuracy is not the highest in absolute terms, the *Transformer* model stands out as the most suitable for the proposed problem, offering a much more realistic and useful balance between accuracy and diversity of predictions. In financial environments where diversification and resilience to uncertainty are fundamental, this type of behavior is valued more than a marginal improvement in the *accuracy* metric.

Nevertheless, the results obtained are not entirely satisfactory. Considering that in the test set Tesla wins 26.29% of the time, all models except Logistic Regression for all features and Random Forest achieve accuracies lower than a basic naive model that would simply predict Tesla. And in the cases where this accuracy is surpassed, the percentage barely exceeds 1%. Furthermore, we observe that the F1-score for MSFT is 0.0000 for all models except for ARIMA and Transformers. For this reason, after analyzing the correlation of daily return among the seven stocks, which is very high in the case of Microsoft (not lower than 0.60, except with Tesla). It is decided to reimplement all the previous models, suppressing this stock in order to avoid redundant patterns and the difficulty for the models to distinguish between classes. The results obtained are presented below.

4.2 Without MSFT

Table 4.2: Model comparison by variable type and method excluding MSFT

Features	Type	Model	F1-score by Stock						Accuracy %	Avg Weight
			AAPL	AMZN	GOOG	META	NVDA	TSLA		
Only Daily Return	Classification	LR	0.0000	0.0000	0.0000	0.0000	0.1982	0.3547	22.07	0.1361
		RF	0.0000	0.0000	0.0000	0.0000	0.2500	0.3894	25.82	0.1564
	Continuous	ARIMA	0.0833	0.0000	0.1695	0.1972	0.2764	0.1687	18.31	0.1620
		VARIMA	0.2571	0.0000	0.0455	0.1481	0.3521	0.1837	22.54	0.2044
All Features	Classification	LR	0.1270	0.0000	0.0000	0.0000	0.3586	0.3791	27.83	0.2049
		RF	0.0000	0.0000	0.0000	0.0000	0.3721	0.2444	24.06	0.1442
		LSTM	0.0455	0.0645	0.2059	0.1176	0.3000	0.0816	16.34	0.1350
	Continuous	VARMAX	0.0000	0.0000	0.0000	0.0000	0.0000	0.4120	25.94	0.1069
	Transformers	Transformers	0.2254	0.0185	0.0986	0.1053	0.3002	0.3410	25.78	0.2284

The accuracy improves in continuous models and in the Transformer model. Particularly noteworthy is the improvement observed in the Transformer model, which reaches an accuracy very close to that of a naive model predicting only Tesla, while significantly enhancing the average weighted F1-score. This model allows capital to be invested across all stocks, minimizing risk for investors without sacrificing profitability.

4.3 Big Players and Small Players

However, with the aim of improving the results, an inspection of the distributions of winning stocks and the outcomes shown in Table 4.1 revealed that the models utilizing the full set of features and yielding the highest accuracies tend to predict almost exclusively among Tesla, NVIDIA, and Apple. Consequently, the stock universe was partitioned into two subsets: *Big Players* and *Small Players*. The *Big Players* group comprises the three most frequently winning stocks—TSLA, NVDA, and AAPL—whereas the *Small Players* group consists of the remaining four stocks. This partition is particularly meaningful, as each subgroup presents a relatively balanced distribution of wins among its members. This helps mitigate bias towards dominant classes and allows for a fairer evaluation of the models' predictive performance. Such analytical segmentation is a widely adopted approach in economic and financial research. By assessing the models independently on these subsets, it is possible to evaluate their robustness and capacity to generalize beyond a single prevailing pattern. This type of analysis is especially valuable when developing predictive tools intended to inform strategic decision-making or guide portfolio diversification.

From a technical standpoint, class imbalance poses a significant challenge in multiclass classification tasks, as it can lead to inflated metrics that do not reflect the model's true predictive capabilities. For example, a model that consistently predicts the historically most frequent stock may yield high overall accuracy, while offering poor performance in identifying the remaining classes, as illustrated in the upper rows of Table 4.1. The strategy adopted in this thesis—decomposing the problem into more homogeneous subsets—facilitates more precise model comparison and reduces the risk of biased results due to class dominance. The results of this analysis are presented below:

Table 4.3: Model Comparison of Big Players

Features	**Type**	**Model**	**F1-score Stocks**			**Accuracy %**	**Avg Weight**
			AAPL	NVDA	TSLA		
Only Daily Return	Classification	LR	0.3256	0.3200	0.3953	35.21	0.3472
		RF	0.1124	0.4020	0.3308	31.92	0.2722
	Continuous	ARIMA	0.3472	0.3855	0.2241	32.86	0.3177
		VARIMA	0.2903	0.3380	0.3750	33.80	0.3329
All Features	Classification	LR	0.2195	0.1695	0.4732	33.96	0.2940
		RF	0.0000	0.4755	0.0988	31.60	0.1761
		LSTM	0.3407	0.2174	0.3799	33.00	0.3152
	Continuous	VARMAX	0.0253	0.0000	0.4982	33.49	0.1738
	Transformers	Transformers	0.0000	0.0000	0.5212	35.25	0.1837

Table 4.4: Model Comparison of Small Players

Features	**Type**	**Model**	**F1-score Stocks**				**Accuracy %**	**Avg Weight**
			AMZN	GOOG	META	MSFT		
Only Daily Return	Classification	LR	0.3167	0.2600	0.3400	0.2830	30.05	0.3000
		RF	0.2885	0.3485	0.3008	0.1404	29.11	0.2766
	Continuous	ARIMA	0.0345	0.3065	0.2714	0.1923	23.00	0.2020
		VARIMA	0.0000	0.0000	0.4045	0.0000	25.35	0.1025
All Features	Classification	LR	0.2472	0.3846	0.1957	0.1639	28.30	0.2540
		RF	0.2745	0.2022	0.3529	0.0435	26.89	0.2257
		LSTM	0.1622	0.4088	0.1263	0.1212	25.48	0.2135
	Continuous	VARMAX	0.0000	0.0328	0.0357	0.3413	21.23	0.0905
	Transformers	Transformers	0.0000	0.4156	0.0000	0.000	26.23	0.1090

In the *Big Players* subset, the models exhibit a performance pattern comparable to the general case. The stock with the highest frequency of wins is AAPL, which accounts for 36.52% of the instances. No model surpasses this percentage. Notably, all models perform better when class balancing is not applied. In contrast, for the *Small Players* subset, models that rely solely on a single feature—the target variable *daily return*—achieve superior performance compared to those incorporating multiple features. In particular, three models, namely Logistic Regression using only the daily return and both Random Forest configurations, attain higher accuracies than the naive baseline model that always predicts GOOG, which is the most frequent winner within this subset, with a win rate of 27.23%. Logistic Regression with only the daily return as input exceeds this naive benchmark by nearly 3% and also records a high F1-score. This makes it a particularly suitable model for this specific classification scenario.

Regarding the *Transformer Models*, the division into *Big Players* and *Small Players* does not lead to meaningful results. In both subsets, the models consistently predict a single winning stock across all instances. This lack of predictive diversity may be attributed to several factors. Firstly, Transformer architectures are inherently designed to capture intricate relationships within multivariate sequences, benefiting most from a broad set of interacting entities. The reduction in the number of stocks—either to three or four—diminishes the variety of patterns and interdependencies available for the model to learn, thereby impairing its ability to differentiate between distinct behaviors. Moreover, the reduction in the number of classes may induce overfitting, with the model becoming biased towards the most frequent stock. For these reasons, despite their architectural

sophistication, Transformer-based models underperform in these simplified settings when compared to configurations involving a larger stock universe. This limitation highlights the importance of maintaining sufficient complexity in the input space to leverage the strengths of such models, particularly in tasks requiring balanced and generalizable classifications.

4.4 Reduced Time Period

Lastly, a shorter time window is reintroduced. Although the time window was previously extended to ten years due to the poor performance of the LSTM model, it is possible that the rest of the models might be generalizing poorly due to the presence of long-term trends that may differ significantly from current ones. Moreover, events such as the U.S. elections in January 2025 destabilized stock behavior during the early months of 2025. Therefore, all models are reimplemented using a three-year time window, from 2022 to 2024. This period was selected to exclude the COVID crisis and the 2025 U.S. elections—both events that had a significant impact on NASDAQ stock prices.

As before, the data is split into training, validation, and test sets, following the same proportions used in the previous 10-year window (3.2.1). The objective is to study the behavior of the models with smaller datasets. The results obtained are as follows:

Table 4.5: Model Comparison by variable type and method 2022-2024

Features	**Type**	**Model**	**F1-socre Stocks**							**Accuracy %**	**Avg Weight**
			AAPL	AMZN	GOOG	META	NVDA	TSLA	MSFT		
Only Daily Return	Classification	LR	0.0000	0.0000	0.0000	0.0000	0.3333	0.3571	0.0000	24.62	0.1645
		RF	0.0000	0.0000	0.0000	0.0000	0.3784	0.3729	0.0000	27.69	0.1792
	Continuous	ARIMA	0.1250	0.0000	0.0000	0.1818	0.3600	0.1667	0.0000	21.54	0.1755
		VARIMA	0.0000	0.0000	0.0000	0.0000	0.3951	0.0000	0.0000	24.62	0.0972
All Features	Classification	LR	0.1091	0.0000	0.0000	0.0000	0.3684	0.3556	0.0000	23.44	0.1697
		RF	0.1176	0.0000	0.0000	0.0000	0.2857	0.2800	0.0000	21.88	0.1602
		LSTM	0.4348	0.0000	0.0000	0.0000	0.0000	0.2353	0.0000	17.50	0.1902
	Continuous	VARMAX	0.0000	0.0000	0.0000	0.0000	0.0000	0.3797	0.0000	23.44	0.0890
	Transformers	Transformers	0.0889	0.0000	0.0000	0.0000	0.3478	0.2286	0.0000	19.55	0.1735

Once again, it is observed that, with the exception of the Random Forest model using daily return, no model outperforms the basic prediction model that always forecasts NVDA (the winning stock in 24.62% of cases in the test set). The percentages are not directly comparable to Table 4.1, as they are based on a different dataset. Additionally, it is important to note that LSTM and Transformer models cannot be properly implemented for 90-day time windows due to insufficient data in the test set, despite 90 days being the optimal window length for both models.

However, it is evident that most models struggle more to predict winning stocks beyond the usual top performers: Apple, NVIDIA, and Tesla. No model detects Microsoft, Google, or Amazon, and only ARIMA predicts Meta. This suggests that reducing the time window biases the predictions toward commonly winning classes, which in this case represent 70% of the winners in the test set as in the previous cases. Therefore, reducing the time window is not recommended for any of the models, and the 10-year window remains the most appropriate choice.

Chapter 5

Conclusion

This study has explored multiple modeling approaches to address stock price forecasting framed as a multiclass classification problem among a selected set of stocks. The results confirm that this remains a highly challenging task, primarily due to the intrinsic high volatility of financial markets and the pronounced class imbalance. Certain stocks emerge as winners far more frequently than others, complicating the learning process for models aiming to identify representative patterns across minority classes. Furthermore, the dynamic nature of the stock market environment, where influential factors evolve continually, demands models with strong capabilities for adaptation and generalization. A comprehensive comparison across all models reveals that neither classical statistical methods nor traditional machine learning algorithms adequately capture the underlying patterns in this highly imbalanced context. The LSTM model, despite its increased complexity, fails to provide substantial improvements. In contrast, Transformer architectures—even in their basic configurations—demonstrate a markedly superior ability to generalize, discriminate between classes, and adapt to the temporal dependencies present in the data. This evidence supports the appropriateness of employing Transformers for sequential financial prediction tasks and justifies their selection as the reference model in this work.

Among the models analyzed, the Transformer-based approach offers the best balance between accuracy and overall predictive capability. Although it does not achieve the highest absolute accuracy, it consistently predicts multiple stocks as winners. This characteristic is particularly valuable in financial contexts, where diversification across stocks is a strategic priority. In summary, Transformer models have proven to be a promising tool for stock prediction from a classification perspective, not only due to their enhanced generalization in complex and dynamic scenarios but also because of their adaptability, computational efficiency, and practical relevance in finance.

This work opens several avenues for future research. Notably, exploring more advanced Transformer variants, such as pretrained models adapted specifically to the financial domain, may yield further improvements. Incorporating exogenous information—such as news sentiment or macroeconomic indicators—also represents a promising direction. Moreover, applying techniques to address class imbalance, including data augmentation or customized loss functions, could enhance predictive performance. Lastly, expanding the universe of considered stocks, although introducing greater volatility, may provide a more comprehensive modeling framework.

Bibliography

[1] Hum Nath Bhandari et al. "Predicting stock market index using LSTM". In: *Machine Learning with Applications* 9 (May 2022), p. 100320. DOI: 10.1016/j.mlwa.2022.100320.

[2] John Bollinger. *Bollinger on Bollinger Bands*. New York: McGraw-Hill, 2001.

[3] Giovanni Campisi, Silvia Muzzioli, and Bernard De Baets. "A comparison of machine learning methods for predicting the direction of the US stock market on the basis of volatility indices". In: *International Journal of Forecasting* 40.3 (2024), pp. 869–880. ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2023.07.002. URL: https://www.sciencedirect.com/science/article/pii/S0169207023000729.

[4] Mark Chen et al. *Generative Pretraining From Pixels*. Ed. by Hal Daumé III and Aarti Singh. July 2020. URL: https://proceedings.mlr.press/v119/chen20s.html.

[5] Eunsuk Chong, Chulwoo Han, and Frank C. Park. "Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies". In: *Expert Systems with Applications* 83 (2017), pp. 187–205. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2017.04.030. URL: https://www.sciencedirect.com/science/article/pii/S0957417417302750.

[6] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL]. URL: https://arxiv.org/abs/1810.04805.

[7] Eugene F. Fama. "Random Walks in Stock Market Prices". In: *Financial Analysts Journal* 51.1 (1995), pp. 75–80. ISSN: 0015198X. URL: http://www.jstor.org/stable/4479810 (visited on 07/15/2025).

[8] Tizian Fischer, Marius Sterling, and Stefan Lessmann. "Fx-spot predictions with state-of-the-art transformer and time embeddings". In: *Expert Systems with Applications* 249 (2024), p. 123538. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2024.123538. URL: https://www.sciencedirect.com/science/article/pii/S0957417424004032.

[9] John Kenneth Galbraith. *The Great Crash 1929*. Boston: Houghton Mifflin, 1955.

[10] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[11] Dariusz Kobiela et al. "ARIMA vs LSTM on NASDAQ stock exchange data". In: *Procedia Computer Science* 207 (Oct. 2022), pp. 3830–3839. DOI: 10.1016/j.procs.2022.09.445.

[12] George C. Lane. “The Stochastic Indicator”. In: *Stocks & Commodities Magazine* 4.1 (1986), pp. 50–56.

[13] Shuozhe Li, Zachery B Schulwolf, and Risto Miikkulainen. “Transformer Based Time-Series Forecasting for Stock”. In: *arXiv:2502.09625* (Jan. 2025). URL: http://www.cs.utexas.edu/users/ai-lab?li:arxiv25.

[14] Syed Shahan Li, Muhammad Mubeen, and Adnan Hussain. “Prediction of stock performance by using logistic regression model: Evidence from Pakistan Stock Exchange (PSX)”. In: *Asian Journal of Empirical Research* 15 (2018), p. 212.

[15] Bryan Lim et al. *Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting*. 2020. arXiv: 1912.09363 [stat.ML]. URL: https://arxiv.org/abs/1912.09363.

[16] Yuang Liu. “Stock Prediction Analysis Based on Logistic Regression and Random Forest”. In: *DeanFrancis* (2025).

[17] Roger Lowenstein. *When Genius Failed: The Rise and Fall of Long-Term Capital Management*. New York: Random House, 2000.

[18] Burton G. Malkiel. “Efficient Market Hypothesis”. In: *Finance*. Ed. by John Eatwell, Murray Milgate, and Peter Newman. London: Palgrave Macmillan UK, 1989, pp. 127–134. ISBN: 978-1-349-20213-3. DOI: 10.1007/978-1-349-20213-3_13. URL: https://doi.org/10.1007/978-1-349-20213-3_13.

[19] Blanca Elvira Fernández Méndez. *Codes ”Forecasting the Next Winning Stock”*. https://drive.google.com/drive/folders/1O2WtadVs_2QO-i1NxM9VGWyHUu1r2jND?usp=sharing. 2025.

[20] Blanca Elvira Fernández Méndez. *Data ”Forecasting the Next Winning Stock”*. https://drive.google.com/drive/folders/12wt2GE7ADcSU-jpiUvdTlImrf23atgfJ?usp=sharing. 2025.

[21] Prapanna Mondal, Labani Shit, and Saptarsi Goswami. “Study of Effectiveness of Time Series Modeling (Arima) in Forecasting Stock Prices”. In: *International Journal of Computer Science, Engineering and Applications* 4 (Apr. 2014), pp. 13–29. DOI: 10.5121/ijcsea.2014.4202.

[22] Leila Mozaffari and Jianhua Zhang. “Predictive Modeling of Stock Prices Using Transformer Model”. In: *Proceedings of the 2024 9th International Conference on Machine Learning Technologies*. ICMLT ’24. Oslo, Norway: Association for Computing Machinery, 2024, pp. 41–48. ISBN: 9798400716379. DOI: 10.1145/3674029.3674037. URL: https://doi.org/10.1145/3674029.3674037.

[23] Nasdaq. *Market Activity - Quotes*. Accedido el 2 de julio de 2025. 2025. URL: https://www.nasdaq.com/market-activity/stocks (visited on 07/02/2025).

[24] Erik Nijkamp et al. *ProGen2: Exploring the Boundaries of Protein Language Models*. 2022. arXiv: 2206.13517 [cs.LG]. URL: https://arxiv.org/abs/2206.13517.

[25] Dev Shah, Haruna Isah, and Farhana Zulkernine. “Stock Market Analysis: A Review and Taxonomy of Prediction Techniques”. In: *International Journal of Financial Studies* 7.2 (2019). ISSN: 2227-7072. DOI: 10.3390/ijfs7020026. URL: https://www.mdpi.com/2227-7072/7/2/26.

[26] Robert J. Shiller. *Irrational Exuberance*. Princeton: Princeton University Press, 2000.

[27] Sima Siami-Namini and Akbar Siami Namin. *Forecasting Economics and Financial Time Series: ARIMA vs. LSTM*. 2018. arXiv: 1803.06386 [cs.LG]. URL: https://arxiv.org/abs/1803.06386.

[28] Sima Siami-Namini and Akbar Siami Namin. *Forecasting Economics and Financial Time Series: ARIMA vs. LSTM*. 2018. arXiv: 1803.06386 [cs.LG]. URL: https://arxiv.org/abs/1803.06386.

[29] Ashish Vaswani et al. "Attention is All You Need". In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017. URL: https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[30] Mehar Vijh et al. "Stock Closing Price Prediction using Machine Learning Techniques". In: *Procedia Computer Science* 167 (2020). International Conference on Computational Intelligence and Data Science, pp. 599–606. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2020.03.326. URL: https://www.sciencedirect.com/science/article/pii/S1877050920307924.

[31] Wikipedia contributors. *Anexo:Empresas por capitalización de mercado — Wikipedia, la enciclopedia libre*. Consultado el 3 de julio de 2025. 2024. URL: https://es.m.wikipedia.org/wiki/Anexo%3AEmpresas_por_capitalizaci%C3%B3n_de_mercado.

[32] J. Welles Jr. Wilder. *New Concepts in Technical Trading Systems*. New York: Trend Research, 1978.

[33] Haixu Wu et al. *Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting*. June 2021. DOI: 10.48550/arXiv.2106.13008.

[34] Yuchen Xiang. "Forecasting the NASDAQ Index Based on the ARIMA Model". In: *Advances in Economics, Management and Political Sciences* 141 (Dec. 2024), pp. 206–213. DOI: 10.54254/2754-1169/2024.GA18869.

[35] *Yahoo Finance*. 2025. URL: https://finance.yahoo.com.

[36] George Zerveas et al. "A Transformer-based framework for multivariate time series representation learning". In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2021).

[37] Haoyi Zhou et al. "Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.12 (2021), pp. 11106–11115. DOI: 10.1609/aaai.v35i12.17325. URL: https://ojs.aaai.org/index.php/AAAI/article/view/17325.