

## PROYECTO FIN DE GRADO

**TITLE:**

AI in Finance: Transformers applied to multivariate time series forecasting.

**AUTOR/A:** Víctor Vallejo Carmona

**TITULACIÓN:** Grado en Ingeniería y Sistemas de Datos

**DIRECTOR/A:** Joaquín Ordieres Meré

**TUTOR/A:** Miguel Ortega Mier

**DEPARTAMENTO:** Ingeniería de Organización, Administración de Empresas y Estadística

**Miembros del Tribunal Calificador:**

**PRESIDENTE/A:** Luis Narvarte Fernández

**TUTOR/A:** Miguel Ortega Mier

**SECRETARIO/A:** Álvaro García Sánchez

**Fecha de lectura:**

**Calificación:**

VºBº TUTOR/A

El Secretario/La Secretaria,



*“Don’t be afraid to take risks and embrace failure that’s where the best opportunities often lie.”*

- Jim Simons





## Agradecimientos

Muchas gracias a mis padres, mis abuelos y mi hermano por su apoyo incondicional y siempre estar dispuestos a ayudarme y a empujarme a conseguir lo que me propongo.

Gracias también a mis amigos de toda la vida y a los que he hecho en la universidad porque sin ellos este camino habría sido más aburrido y seguro que menos fructífero.

Por último, agradecerle a Joaquín y a Miguel todo el tiempo que han invertido en dirigir este proyecto y en guiarme para hacerlo lo mejor posible.



## Resumen

La predicción de precios en mercados financieros es una de las aplicaciones más prometedoras y rentables de la Inteligencia Artificial (IA), con inversiones significativas en la búsqueda de sistemas de IA capaces de anticipar los movimientos del mercado. El desarrollo de un sistema de IA que pueda predecir cada vez con mejor precisión los precios del mercado, podría generar ganancias ilimitadas. De esto trata el artículo publicado por Jeff Kearns en el Fondo Monetario Internacional [1] y en el que hace referencia a líderes del sector financiero del nivel de Jamie Dimon, CEO de J.P.Morgan Chase.

En este proyecto, se explora y evalúa el uso de transformers como alternativas a la tecnología de memoria a largo plazo (LSTM) en el análisis multivariante de series temporales en el sector financiero. Las LSTM son un tipo de red neuronal recurrente (RNN) muy utilizado en las predicciones de series temporales.

El objetivo principal de este proyecto es la comparación entre las diferentes tecnologías actuales de inteligencia artificial para la predicción de series temporales en el ámbito financiero. Para ello, un sub objetivo es desarrollar un transformer con atención multi cabeza capaz de estimar cada vez con mayor precisión la evolución de los precios de cierre en un horizonte temporal predefinido y comparar con los resultados de investigaciones similares, valorando sus resultados. Este enfoque innovador se diferencia por la comparación entre el rendimiento de los transformers con atención multi cabeza y la tecnología LSTM, habitualmente utilizada en este tipo de análisis. Los datos utilizados en el entrenamiento del transformer se agrupan en tres categorías: análisis del sentimiento en Twitter/X y Bloomberg, lo que proporciona conocimiento sobre el sentimiento tanto del público aficionado como del profesional; volumen diario de negociación de la acción; y factores relacionados con el precio de la acción. Estos datos son de quince valores incluidos en el índice S&P 500, principal índice bursátil de EE.UU. durante el período comprendido entre 2015 y 2023.

La relevancia de este proyecto en el ámbito financiero radica en su capacidad para aportar perspectivas novedosas sobre las metodologías de predicción de precios y comparativas con lo existente. Al destacar la importancia del volumen bursátil y el sentimiento del mercado en la previsión financiera, se aportan ideas clave para mejorar la toma de decisiones de inversión. Este proyecto tiene el potencial de revolucionar la forma en que los inversores y los analistas financieros abordan la predicción de precios, lo que podría tener un impacto significativo en la toma de decisiones de inversión y en la gestión de riesgos.

Los resultados obtenidos en el proyecto han sido claramente satisfactorios. El estudio de los modelos se ha realizado para predicciones a 5 periodos temporales diferentes: corto plazo (1, 7 y 14 días), medio plazo (30 días) y largo plazo (90 días). Para los periodos más corto placistas, las predicciones realizadas por los modelos de LSTM han arrojado unos resultados superiores a los del transformer. En cuanto el periodo de tiempo de las predicciones se va ampliando hacia medio y largo plazo, los resultados del transformer pasan a ser superiores a los de las LSTM, llegando incluso a reducir el MSE en más de un 95% con respecto a las LSTM en el periodo de 90 días.

Además, se ha realizado una comparación entre los resultados de este PFG con los resultados de otros dos papers de objetivo similar. El primero de ellos es una comparativa con un modelo que utiliza una red neuronal convolucional (CNN) y el segundo es un modelo LSTM que con análisis de sentimiento trata de predecir los precios de cierre de ciertas acciones.

En resumen, este proyecto busca elaborar una comparación minuciosa entre los sistemas de IA

actuales para la predicción de series temporales, añadiendo un sistema innovador que pueda predecir con precisión los precios del mercado, utilizando transformers y análisis multivariante. La comparación con la tecnología LSTM y el enfoque en el sentimiento del mercado y el volumen bursátil como variables clave para la predicción de precios, hacen de este proyecto una contribución significativa en el campo de la Inteligencia Artificial aplicada a las finanzas.

## Abstract

Price prediction in financial markets is one of the most promising and profitable applications of Artificial intelligence, with significant investments in the search for AI (Artificial intelligence) systems capable of anticipating market movements. The development of an AI system that can accurately predict market prices could generate unlimited profits. This is the subject of an article published by Jeff Kearns in the International Monetary Fund [1], in which he refers to financial sector leaders such as Jamie Dimon, CEO of J.P. Morgan Chase.

In this project, the exploration and evaluation of transformers as alternatives to long short-term memory (LSTM) technology in multivariate time series analysis in the financial sector is conducted. LSTMs are a type of recurrent neural network (RNN) widely used in time series predictions.

The main objective of this project is the comparison between different current artificial intelligence technologies for time series prediction in the financial field. To this end, a sub-objective is to develop a multi-headed attention transformer capable of estimating with increasing accuracy the evolution of closing prices over a predefined time horizon and to compare with the results of similar researches, evaluating their results. This innovative approach differs by comparing the performance of the multi head attention transformers and LSTM technology, commonly used in this type of analysis. The data used in training the transformer are grouped into three categories: sentiment analysis on Twitter/X and Bloomberg, which provides insight into both amateur and professional public sentiment; daily trading volume of the stock; and factors related to the stock price. This data is for fifteen stocks included in the S&P 500 index, the leading U.S. stock index, over the period from 2015 to 2023.

The relevance of this project in the financial field lies in its ability to provide novel perspectives on price forecasting methodologies and comparisons with existing ones. By highlighting the importance of stock market volume and market sentiment in financial forecasting, key insights are provided to improve investment decision making. This project has the potential to revolutionize the way investors and financial analysts approach price forecasting, which could have a significant impact on investment decision making and risk management.

The results obtained in the project have been clearly satisfactory. The study of the models has been carried out for predictions over 5 different time periods: short term (1, 7 and 14 days), medium term (30 days) and long term (90 days). For the shorter term periods, the predictions made by the LSTM models have yielded results superior to those of the transformer. As the time period of the predictions extends towards the medium and long term, the results of the transformer become superior to those of the LSTM, even reducing the MSE by more than 95% with respect to the LSTM in the period of 90 days.

In addition, a comparison has been made between the results of this PFG with the results of two other papers with similar objectives. The first one is a comparison with a model that uses a convolutional neural network (CNN) and the second one is a LSTM model that with sentiment analysis tries to predict the closing prices of certain stocks.

In summary, this project seeks to develop a thorough comparison between current AI systems for time series prediction, adding an innovative system that can accurately predict market prices, using transformers and multivariate analysis. The comparison with LSTM technology and the focus on market sentiment and stock volume as key variables for price prediction make this project a significant contribution to the field of Artificial Intelligence applied to finance.

Esta plantilla es una ligera modificación de la “Plantilla en LaTeX acorde con la Normativa para la elaboración de informes de TFT de la ETSII (UPM)” que es un trabajo original de Javier Soto Pérez-Olivares. Javier es el verdadero responsable de la gran cantidad de trabajo que hay detrás de esta guía/plantilla.



Plantilla en LaTeX acorde con la “Guía del alumno Trabajo Fin de Grado del Grado de Ingeniería y Sistemas de Datos (actualizado octubre 2023)” by Miguel Ortega-Mier is licensed under a Creative Commons Attribution

4.0 International License.

## Table of Contents

<b>Resumen</b>	<b>vi</b>
<b>Abstract</b>	<b>viii</b>
<b>List of tables</b>	<b>xiv</b>
<b>List of figures</b>	<b>xvii</b>
<b>List of abbreviations</b>	<b>xviii</b>
<b>Codes Index</b>	<b>xx</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Technological framework</b>	<b>3</b>
2.1 Artificial intelligence . . . . .	3
2.2 Python . . . . .	5
2.3 Machine learning . . . . .	6
2.4 Deep learning . . . . .	9
2.5 Neural networks . . . . .	10
2.5.1 Structure and Functioning . . . . .	10
2.5.2 Applications and advances . . . . .	11
2.5.3 Introduction to LSTM . . . . .	11
2.6 Long-Short Term Memory . . . . .	12
2.6.1 LSTM architecture and functionality . . . . .	12
2.6.2 Advantages of LSTMs . . . . .	12
2.6.3 Applications of LSTMs . . . . .	13
2.6.4 Financial market predictions with LSTM . . . . .	13
2.7 Transformers . . . . .	13
2.7.1 Encoder . . . . .	14
2.7.2 Self-Attention mechanism . . . . .	14

2.7.3	Multi-Head attention . . . . .	15
2.7.4	Output linear layer . . . . .	15
2.7.5	Applications of transformers . . . . .	15
2.7.6	Financial applications of transformers . . . . .	15
<b>3</b>	<b>Design specifications and requirements</b>	<b>17</b>
3.1	Design specifications . . . . .	17
3.1.1	System flow diagram . . . . .	17
3.1.2	System functionality . . . . .	17
3.2	Requirements . . . . .	18
<b>4</b>	<b>Description of the proposed solution</b>	<b>21</b>
4.1	Data study . . . . .	21
4.1.1	Stocks . . . . .	22
4.1.2	Features . . . . .	22
4.2	Preparation for simulation scenarios . . . . .	23
4.3	System Operation . . . . .	24
4.3.1	Data Preprocessing . . . . .	24
4.3.2	LSTM Models . . . . .	26
4.3.3	Transformer models . . . . .	31
<b>5</b>	<b>Experiments and results</b>	<b>35</b>
5.1	Scenario 1 results . . . . .	36
5.2	Scenario 2 results . . . . .	41
5.3	Scenario 3 results . . . . .	44
5.4	Transformers parameters results comparison . . . . .	48
5.5	Comparison of results with other studies . . . . .	50
5.6	Application to a trading strategy . . . . .	51
5.7	Application to portfolio management . . . . .	52
<b>6</b>	<b>Impacts of the project</b>	<b>55</b>



<b>7</b>	<b>Planning and budget</b>	<b>57</b>
7.1	Planning . . . . .	57
7.2	Budget . . . . .	58
<b>8</b>	<b>Conclusions and future works</b>	<b>59</b>
8.1	Conclusions . . . . .	59
8.2	Future works . . . . .	60
8.3	Personal learning . . . . .	60
	<b>References</b>	<b>63</b>
	<b>Appendix</b>	<b>67</b>
	Appendix 1: Stock Class . . . . .	67
	Appendix 2: LSTM Class . . . . .	68
	Appendix 3: Transformers Classes . . . . .	71
	Appendix 4: User manual . . . . .	72



## List of tables

5.1	Comparison of the results for the univariate and multivariate transformers . . . .	49
7.1	Materials and tools costs. . . . .	58
7.2	Personnel costs. . . . .	58
7.3	Total costs. . . . .	58



**List of figures**

2.1	AI framework [8]. . . . .	3
2.2	Dog cat classification [17]. . . . .	6
2.3	Spam detection [18]. . . . .	7
2.4	Clustering for customers segmentation [19]. . . . .	7
2.5	Principal component analysis [20]. . . . .	8
2.6	Anomaly detection [21]. . . . .	8
2.7	Neural Network architecture [28]. . . . .	11
2.8	LSTM architecture [33]. . . . .	12
2.9	Transformer architecture [3]. . . . .	14
3.1	Flow diagram [self elaboration]. . . . .	17
4.1	Dataset example . . . . .	21
4.2	Correlation matrix of scaled features [self elaboration]. . . . .	23
4.3	System operation [self elaboration]. . . . .	24
4.4	Stock Class [self elaboration]. . . . .	26
4.5	LSTM Class [self elaboration]. . . . .	27
4.6	Basic LSTM [self elaboration]. . . . .	28
4.7	Stacked LSTM [self elaboration]. . . . .	29
4.8	Attention-based LSTM [self elaboration]. . . . .	30
4.9	Uni/multi dimension transformer architecture proposed [self elaboration]. . . . .	32
5.1	Scenario 1 configuration [self elaboration]. . . . .	35
5.2	Scenario 2 configuration [self elaboration]. . . . .	35
5.3	Scenario 3 configuration [self elaboration]. . . . .	36
5.4	AMZN-LSTM-MSE-70% [self elaboration]. . . . .	37
5.5	AMZN-Transformer-MSE-70% [self elaboration]. . . . .	37
5.6	AMZN-TransformerMulti-MSE-70% [self elaboration]. . . . .	38
5.7	MSE stocks comparison scenario1 [self elaboration]. . . . .	39
5.8	Train test split MSE comparison scenario 1 [self elaboration]. . . . .	40

5.9	AMZN-STCKLSTM-MSE-75% [self elaboration]. . . . .	41
5.10	AMZN-Transformer-MSE-75% [self elaboration]. . . . .	41
5.11	AMZN-Transformer-m-MSE-75% [self elaboration]. . . . .	42
5.12	MSE stocks comparison scenario2 [self elaboration]. . . . .	43
5.13	Train test split MSE comparison scenario 2 [self elaboration]. . . . .	44
5.14	AMZN-AttLSTM-MSE-80% [self elaboration]. . . . .	45
5.15	AMZN-Transformer-MSE-80% [self elaboration]. . . . .	45
5.16	AMZN-Transformer-m-MSE-80% [self elaboration]. . . . .	46
5.17	MSE stocks comparison scenario3 [self elaboration]. . . . .	47
5.18	Train test split MSE comparison scenario 3 [self elaboration]. . . . .	48
5.19	The result of MAE comparison among different methods [46]. . . . .	50
5.20	RMSE values from different models [47]. . . . .	50
5.21	Amazon lstm MAE scenario 1 [self elaboration]. . . . .	52
5.22	Amazon transformer multivariate MAE scenario 1 [self elaboration]. . . . .	52
6.1	SDGs aligned with the project [48]. . . . .	56
7.1	Project's Gantt chart [self elaboration]. . . . .	57

## List of abbreviations

<b>AI</b>	Artificial Intelligence
<b>LSTM</b>	Long-short term memory
<b>NN</b>	Neural Network
<b>NLP</b>	Natural Language Processing
<b>CNN</b>	Convolutional Neural Network
<b>RNN</b>	Recurrent Neural Network
<b>MSE</b>	Mean Squared Error
<b>MAE</b>	Mean Absolute Error
<b>GRU</b>	Gated Recurrent Unit
<b>SDG</b>	Sustainable Development Goals
<b>VAT</b>	Value-Added Tax





**List of codes**

8.1	Stock Class . . . . .	67
8.2	LSTM Class . . . . .	68
8.3	LSTM Basic function . . . . .	68
8.4	LSTM Stacked function . . . . .	69
8.5	LSTM Attention function . . . . .	70
8.6	Transformer Class . . . . .	71
8.7	Forward method . . . . .	71



# 1 Introduction

Artificial intelligence is for many the new revolution of the 21st century. Today, there are countless projects, companies and workers focused on the development of new and increasingly innovative AI tools. More specifically, in the field of finance, it is one of the sectors in which more money and resources are invested in this type of project. This is because the investment bank or hedge fund that gets the best algorithm to predict the market wins. In this project a comparison of the most commonly used AI technologies in price prediction (LSTM) [2] has been made with two variants of transformers [3], one univariate and one multivariate.

The motivation behind this project lies in the potential of AI to revolutionize financial markets. The interest in applying AI to financial markets lies in its potential to significantly improve predictions and then investment decisions. However, algorithms trained on millions of data and tens of millions of dollars in investments are frequently observed to fail and make very poor predictions. The challenge of applying AI to financial markets is that they are not just numbers; markets also reflect the psychology of citizens and what society believes will happen, making it extremely difficult to predict their behavior.

Main Objective of the project:

- Compare the performance of LSTM and transformer technologies in predicting the closing stock price at different time horizons (1 day, 7 days, 14 days, 30 days and 90 days).

Secondary Objectives:

- Develop three types of LSTM networks: classical LSTM, stacked LSTM, and attention-based LSTM.
- Develop and test a transformer architecture, both univariate and multivariate, for price prediction.
- Promote the application of multi-head attention transformers to identify their capabilities and limitations regarding prediction horizons.
- Evaluate the performance of each of the LSTM models.
- Conduct a critical analysis comparing the performance of different transformer architectures against traditional LSTM models.
- Make an analysis comparing different values for the system parameters (window size, number of heads, etc).
- Perform a cross-sectional analysis on different stocks to understand if the characteristics of the stocks can affect the conclusions.

In this project the technologies to be compared, as previously mentioned, will be mainly two: LSTM and transformers. As for the LSTM, 3 types of these neural networks have been used for this project: classical LSTM, stacked LSTM and attention-based LSTM. Three different types of LSTMs in which they differ in that the Stacked LSTM [4] is an extension of classical LSTM that uses multiple layers of LSTM units on top of each other. Each layer in a stacked LSTM takes as input the output of the previous layer, allowing the network to learn hierarchical and complex representations of the input data. And the attention-based LSTM [5] is a variant of

LSTM networks that uses attention mechanisms to weight the importance of different parts of the input at each time step. It allows the network to focus on specific parts of the input sequence that are relevant to the current task, which can improve performance on long and complex sequence modeling tasks. As for transformers, a transformer has been developed with an architecture that is explained in the 4 Description of the proposed solution section.

It should be noted that the objective of the work is not to develop a super-accurate price prediction system, since even the world's largest investment banks have not achieved this with large amounts of resources. What is sought is the comparison of the technologies available in the market with a proprietary transformer for the task of price prediction, taking into account the obvious limitations of time, computational capacity and money.

To achieve these objectives, the following methodology has been used:

1. Review the state of the art of LSTM and transformer technologies in price prediction.
2. Prepare, serialize, and normalize time series data for use in the models.
3. Develop and train three variants of LSTM networks: classical LSTM, stacked LSTM, and attention-based LSTM.
4. Design, implement, and train univariate and multivariate transformer architectures.
5. Apply multi-head attention transformers and analyze their capabilities and limitations.
6. Compare the performance of different transformer architectures with traditional LSTM models.
7. Conduct the different analysis.
8. Evaluate the results obtained and draw conclusions.

The structure of the document is as follows: first there is an introduction to the subject and the objectives of the project are defined, followed by an analysis of the technological framework to know the state of the art of the technologies to be studied in the project. In the third section, the design specifications of the system are developed as well as the requirements that have been presented. This is followed by a description and explanation of the proposed solution as well as the results obtained in the project and a comparison with two other related projects in this area. After this, the impacts of the project are analyzed from different approaches such as social, economic and technological. In addition, it explains the SDGs (Sustainable Development Goals) in which this project is framed. Then there is a budget and a planning for this project. Finally, the conclusions obtained after the study are explained and the future developments. At the end of each section, there is a small section summary to recap the contents that has been explained in it.

## 2 Technological framework

Going back to what was mentioned in the introduction, the development of AI systems applied to time series analysis is constantly growing [6]. These systems can be used for fields as varied as NLP, economic forecasting, sales forecasting, etc. Focusing on the field of study of this dissertation, financial data, the key concepts of the tools and technologies used in this project will be explained.

### 2.1 Artificial intelligence

Artificial Intelligence (AI) is a multidisciplinary field of study that seeks to develop systems capable of performing tasks that require human intelligence. Over the decades, AI has experienced significant advances, driven by the growth in computational capacity, access to large amounts of data and the development of more sophisticated algorithms. The foundations of AI date back to the 1950s, with pioneers such as Alan Turing and his famous Turing Test, which raised the question of whether a machine could exhibit intelligent behavior indistinguishable from that of a human being [7]. Around the same time, the first neural network models were developed, inspired by the workings of the human brain. During the following decades, AI experienced periods of great interest and progress, alternating with periods of stagnation known as "AI winters". However, in the last two decades, AI has experienced a renaissance driven by advances in key areas:

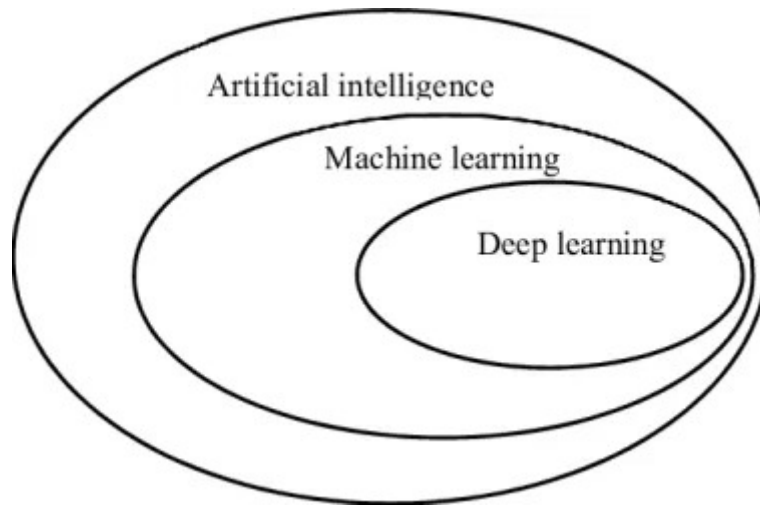


Figure 2.1: AI framework [8].

**Machine Learning:** Machine learning, a subset of artificial intelligence, encompasses various techniques for teaching computers to learn from data and make decisions without being explicitly programmed. One fundamental distinction in machine learning is between supervised and unsupervised learning paradigms. This three paradigms are better explained in the section 2.3 Machine learning, but here is a small definition of each one:

**Supervised learning** involves training a model with labeled data, where the correct answer is known. During training, the model adjusts its parameters to minimize the discrepancy between predictions and actual responses. It is applied in classification and regression problems, such as image recognition or sentiment analysis.

Unsupervised learning, on the other hand, relies on unlabeled data, where the model must discover patterns or structures by itself. It can be used for tasks like clustering or dimensionality reduction, applied in areas such as customer segmentation or social network analysis.

Reinforcement learning [9] entails the model learning through interaction with an environment, receiving feedback in the form of rewards or punishments based on its actions. The goal is to maximize the accumulated reward over time, applied in games, robotics, or resource optimization. This learning approach, inspired by behavioral psychology, has been fundamental to the success of AI systems in complex games and robot control.

Semi-supervised learning combines elements of supervised and unsupervised learning, using both labeled and unlabeled data during training. It is applied when obtaining labeled data is costly or difficult, such as in text analysis or image classification with ambiguous labels.

Federated learning enables distributed training of the model on multiple devices or servers, without sharing raw data between them. It is used in scenarios where data is sensitive or private, such as in healthcare or banking, allowing model training without compromising data privacy.

**Deep Learning:** Deep learning, a sub-discipline of machine learning, has revolutionized the way complex problems are approached. Models such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs) have set new performance standards in a wide range of applications.

**Natural Language Processing (NLP) [10]:** Natural language processing has advanced significantly in recent years, enabling machines to understand and generate text more effectively. Models such as GPT (Generative Pre-trained Transformer) have demonstrated remarkable capability for text generation tasks, machine translation and more.

**Image Recognition and Computer Vision [11]:** Computer vision systems have achieved impressive levels of accuracy, often surpassing human capability in tasks such as image classification, object detection and facial recognition.

AI applications are diverse and span a wide range of industries, including medicine, finance, manufacturing, transportation and more. However, as AI continues to advance, ethical and social challenges arise related to data privacy, algorithmic discrimination, and impact on employment.

For this project, focusing on financial data and time series analysis, the following technologies and methodologies have been particularly relevant:

- **Transformers:** The core of this project is the application of transformer models for multivariate time series analysis. A transformer is a neural network that learns context and, therefore, meaning by tracking relationships in sequential data [12].
- **Sentiment Analysis:** Utilizing sentiment analysis on Twitter/X and Bloomberg data has provided valuable insights into public and professional sentiment, which could be crucial for making accurate market predictions.
- **Supervised Learning:** This project heavily relies on supervised learning techniques to train the models using labeled financial data, allowing for precise prediction of stock prices.

While other AI techniques such as unsupervised learning, reinforcement learning, and federated learning have their own merits and potential applications in the financial sector, they have not

been directly applied in this project. Future research could explore these areas to enhance model performance and address challenges related to data privacy and complex decision-making environments.

## 2.2 Python

Python is a programming language that supports multiple programming paradigms, including structured (particularly procedural), object-oriented, and functional programming [13]. Its simplicity, readability, and extensive library support make it an ideal choice for a wide range of applications, particularly in the field of data science and engineering. The exponential growth of data usage has created a need for technologies that can efficiently handle large volumes of data, and Python has emerged as the preferred language among data professionals such as data engineers, data scientists, and AI engineers.

One of the key reasons for Python's popularity in data-related fields is its rich ecosystem of libraries and frameworks. Libraries such as NumPy and Pandas provide robust tools for data manipulation and analysis, while Matplotlib and Seaborn are widely used for data visualization. For machine learning and artificial intelligence, libraries like Scikit-learn, TensorFlow, and PyTorch offer powerful functionalities that enable rapid development and deployment of complex models.

Python's application extends across various sectors, from finance and healthcare to manufacturing and robotics. Its ability to streamline data workflows and simplify the analysis process contributes to its widespread adoption. For instance, in the finance sector, Python is used for algorithmic trading, risk management, and financial analysis. In healthcare, it facilitates the analysis of large datasets for medical research and the development of predictive models for patient outcomes. In manufacturing, Python aids in optimizing production processes and predictive maintenance through data analysis.

The language's growth trajectory in the data science community is evident from several milestones. In 2016, Python surpassed R on Kaggle, a leading platform for data science competitions, highlighting its rising preference among practitioners. By 2017, Python had also overtaken R in KD Nuggets' annual survey of data scientists, further solidifying its dominance. According to a 2018 survey, approximately 66% of data scientists reported using Python daily, making it the top language for analytics professionals [14].

In addition to its technical strengths, Python's supportive and active community plays a significant role in its success. The community continuously contributes to the development of new libraries and tools, ensures the language remains up-to-date with the latest advancements, and provides a wealth of resources for learning and problem-solving. This collaborative environment not only accelerates individual learning curves but also drives innovation within the field.

TensorFlow, developed by the Google Brain team, has become a leading framework in the deep learning landscape. Its flexibility and scalability allow developers to build and deploy machine learning models on various platforms, from mobile devices to large-scale distributed systems. TensorFlow's high-level APIs, such as Keras, simplify model creation and training, making it accessible to both beginners and experts. The framework's support for GPU acceleration and TensorFlow Extended (TFX) facilitates the development of production-level machine learning pipelines, further cementing its role in state-of-the-art AI research and industrial applications [15].

PyTorch, developed by Facebook’s AI Research lab (FAIR), has gained significant traction due to its dynamic computational graph and intuitive interface, which closely aligns with Python’s programming style. This dynamic nature allows for more flexibility and debugging ease during model development. PyTorch has become the framework of choice for many researchers in academia, leading to rapid adoption in cutting-edge research. Its ecosystem includes tools like TorchServe for model serving and PyTorch Lightning for simplifying complex model training, making PyTorch a comprehensive and versatile tool for both research and production in the field of machine learning [16].

### 2.3 Machine learning

As mentioned in 2.1 Artificial intelligence, machine learning is a subset of Artificial Intelligence. Machine learning is a field that studies algorithms and statistical techniques that allow computer systems to ”learn” and improve their performance on a given task through experience. Instead of explicitly programming instructions to perform a task, machine learning algorithms create a mathematical model based on input data, known as training data, to make predictions or decisions without being explicitly programmed to perform the task. There are three main types of machine learning:

- Supervised learning involves using a set of labeled input and output data to train a model. The goal is for the model to be able to make accurate predictions on new input data by computing on patterns learned from labeled training data. Some examples of supervised learning tasks are classification (assigning a category or label to an input data), regression (predicting a continuous numerical value), and pattern recognition. Popular supervised learning algorithms include decision trees, support vector machines (SVM), neural networks, logistic regression, and random forests.

For example, in image classification tasks, a supervised learning model is trained on a dataset of labeled images (e.g., dog and cat images labeled accordingly). The model learns to recognize patterns and characteristics that distinguish cats from dogs. Once trained, the model can classify new, unseen images such as a cat or a dog.

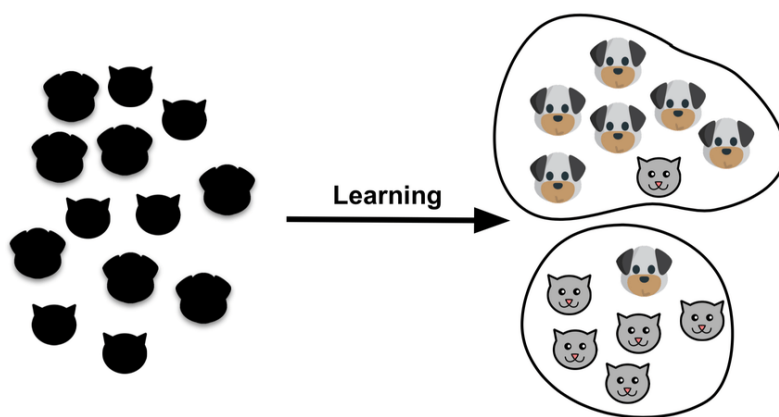


Figure 2.2: Dog cat classification [17].

Another example is spam detection in email, which can be flagged as a classification problem. The model is trained with a dataset of emails labeled as spam or non-spam. Learn patterns and characteristics in email content, subject lines, sender addresses, etc., that can distinguish spam from legitimate emails. The trained model can then classify new incoming emails as spam or not spam.



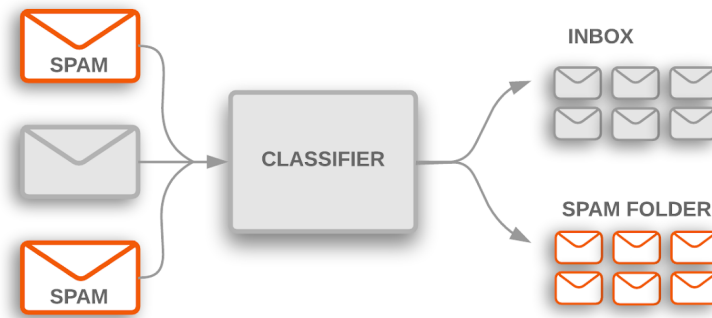


Figure 2.3: Spam detection [18].

In regression problems, such as house price prediction, the model is trained with a data set of houses with their associated characteristics (e.g., size, number of bedrooms, location) and their sales prices. The model learns the relationships between these characteristics and the sales price. Then, given the characteristics of a new home, the model can predict its likely selling price.

Supervised learning is widely used in various applications including computer vision, natural language processing, speech recognition, fraud detection, recommender systems and many more domains where labeled data is available to train predictive models.

- In unsupervised learning, the input data has no associated labels. The goal is to discover inherent patterns, structures, or relationships in the data [3]. Some examples of unsupervised learning tasks are clustering (grouping similar data into groups), dimensionality reduction (reducing the number of random variables in a data set), and anomaly detection.

Popular unsupervised learning algorithms include k-means clustering, hierarchical clustering, principal component analysis (PCA), and self-organizing maps (SOM).

In clustering, algorithms group data into clusters or groups based on inherent similarities between the data. For example, in customer data analysis, clustering could be used to segment customers into groups with similar purchasing patterns, allowing companies to offer more personalized products and services.

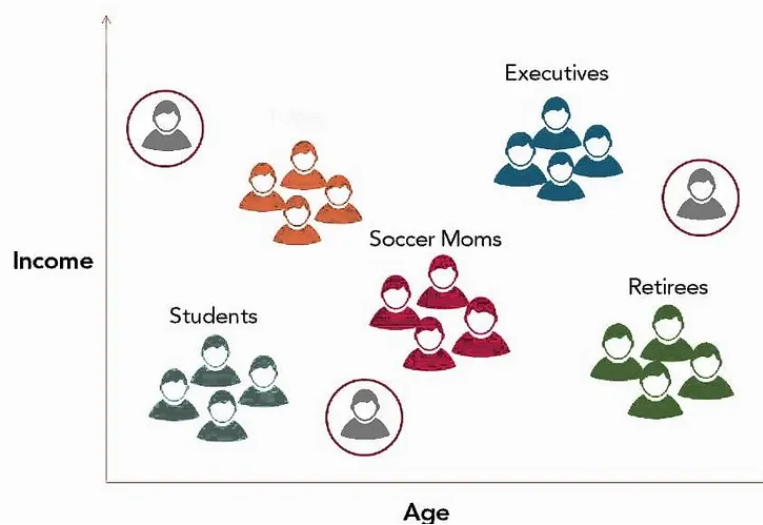


Figure 2.4: Clustering for customers segmentation [19].

Dimensionality reduction is useful when working with high-dimensional data sets with many features. Techniques such as PCA allow us to reduce the dimensionality of the data by identifying the most relevant characteristics and representing the data in a lower dimensional space. This can make data visualization and analysis easier, and improve the performance of other machine learning algorithms.

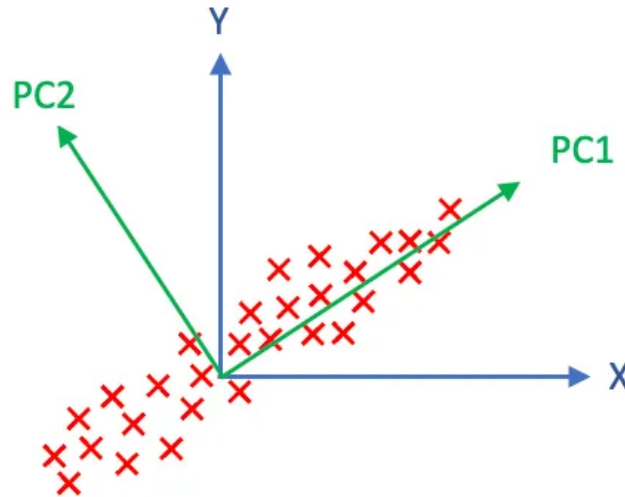


Figure 2.5: Principal component analysis [20].

Anomaly detection is used to identify unusual or deviant observations in a data set. For example, in fraud detection, anomaly detection algorithms can help identify suspicious transactions that deviate significantly from normal behavior.

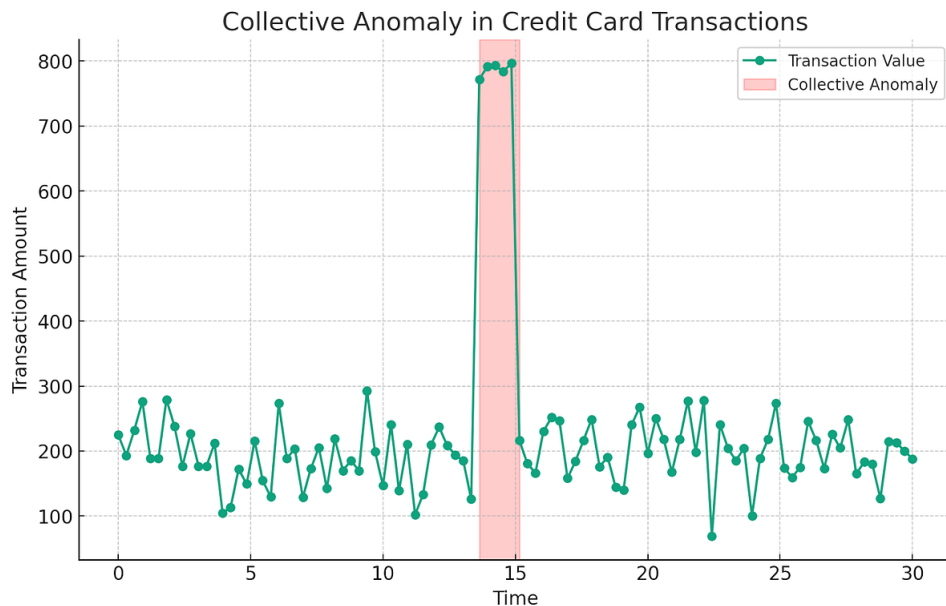


Figure 2.6: Anomaly detection [21].

Unsupervised learning stands out for its usefulness when labeled data is not available or when seeking to discover underlying patterns and structures in the data without imposing predefined labels.

- Reinforcement learning involves making sequential decisions in an environment, where an agent (the learning algorithm) learns through interaction with the environment, taking actions and receiving rewards or punishments. The goal is to maximize the total reward in the long term.

This type of learning is useful in sequential control and decision-making problems, such as playing chess or video games, controlling robots or autonomous vehicles. Popular reinforcement learning algorithms include Q-learning, SARSA, and temporal difference (TD) methods.

## 2.4 Deep learning

Deep learning, a sub-field of machine learning, has emerged as one of the most influential areas in artificial intelligence research in recent decades. Its ability to automatically model complex patterns in unstructured data has led to significant advances in a variety of fields, from speech recognition to autonomous driving.

### 1. Deep Neural Networks:

At the core of deep learning are deep neural networks, computational models that mimic the functioning of the human brain through layers of interconnected artificial neurons. These networks can have multiple hidden layers, allowing them to learn increasingly abstract representations of input data.

### 2. Convolutional Neural Network (CNN) [22]:

Convolutional neural networks have revolutionized image processing by capturing spatial patterns in two-dimensional data. They use convolutional layers to extract relevant features from images, followed by pooling layers to reduce dimensionality. CNNs have demonstrated outstanding performance in tasks such as image classification, object detection, and semantic segmentation.

### 3. Recurrent Neural Networks (RNN):

Recurrent neural networks are suitable for modeling sequential data, such as text, audio, and time series. Unlike traditional neural networks, RNNs have feedback connections that allow them to maintain and update an internal state as they process each element of the sequence. This makes them especially effective in tasks such as language modeling, machine translation, and text generation.

### 4. Transformer Architectures:

Transformer architectures have gained prominence in recent years, especially in natural language processing. Introduced by the "Attention is All You Need" [3] model, Transformer architectures eliminate sequential dependencies and enable greater parallelization in training. Models like Gemini AI [23] and GPT [24] have set new standards in NLP tasks such as language understanding and text generation.

### 5. Transfer Learning and Pre-training [25]:

The use of transfer learning and pre-training has proven crucial for success in many deep learning applications. These techniques involve training models on large generic datasets, such as ImageNet or Wikipedia, and then fine-tuning them on specific tasks with smaller datasets. This allows models to leverage prior knowledge learned on related tasks and generalize better to new domains.

### 6. Advances in Optimization and Hardware:

Advances in optimization algorithms, such as stochastic gradient descent (SGD) and its variants, have improved the efficiency and speed of training deep learning models. Additionally, the development of specialized hardware, such as graphics processing units (GPUs) and tensor processing units (TPUs), has further accelerated the training and deployment of deep learning models.

### 7. Emerging Trends and Future Directions:

- **Generative Models:** Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) are at the forefront of generating realistic synthetic data, with applications in image synthesis, style transfer, and anomaly detection.
- **Explainable AI (XAI):** As deep learning models become more complex, there is a growing need for explainability. XAI techniques aim to make the decision-making process of AI models more transparent and understandable to humans.
- **Federated Learning:** This approach enables training models across decentralized devices while keeping data localized, enhancing privacy and security.
- **Neural Architecture Search (NAS):** Automated methods for designing neural network architectures are helping discover optimal model configurations, reducing the need for manual tuning.
- **Integration with IoT and Edge Computing:** Deploying deep learning models on edge devices is becoming increasingly feasible, allowing real-time processing and decision-making in applications like autonomous vehicles and smart cities.

By continually advancing in these areas, deep learning is set to remain at the forefront of AI research and application, driving innovation and transforming industries.

## 2.5 Neural networks

Neural networks (NN) [26] have emerged as a powerful tool in the field of artificial intelligence, inspired by the functioning of the human brain. These computational models, also known as connectionist systems, have evolved from various scientific contributions throughout history. They consist of a set of units called artificial neurons, connected together to transmit signals and process input information, producing output values.

### 2.5.1 Structure and Functioning

[27] An artificial neural network is composed of layers of interconnected nodes, where each node, or artificial neuron, is connected to others and has associated link weights. These weights can increase or inhibit the activation of adjacent neurons, and an activation function modifies the output of the neuron. The input information traverses the neural network, undergoing various operations on each neuron, to produce output values.

The learning process in neural networks is autonomous, rather than explicitly programmed. It seeks to minimize a loss function that evaluates the overall performance of the network by adjusting the weights of the connections through a process called back propagation. This approach allows neural networks to adapt and continuously improve their accuracy over time. In figure 2.7 Neural Network architecture [28]. depicts the basic architecture of a neural network.

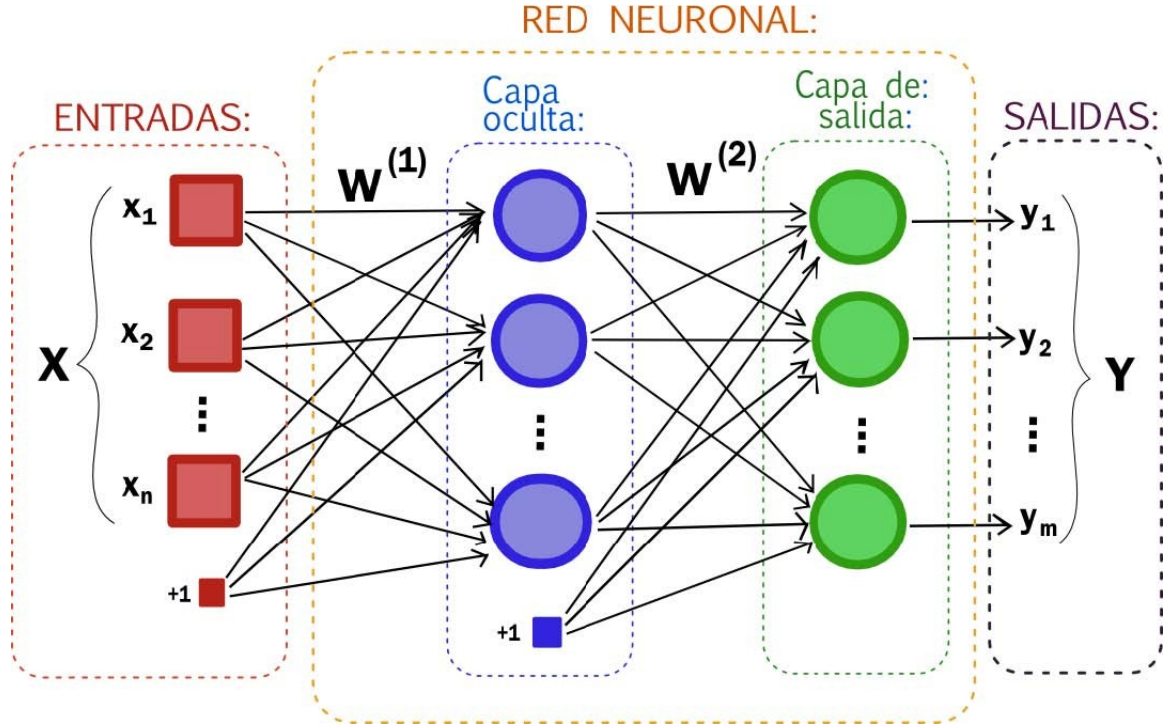


Figure 2.7: Neural Network architecture [28].

### 2.5.2 Applications and advances

[29] Neural networks find application in a wide variety of fields, including computer vision, speech recognition and natural language processing. Their ability to solve complex problems, where solution or feature detection is difficult to express through conventional programming, makes them invaluable tools.

Recent research on the human brain inspires new approaches to neural networks, such as the exploration of extended connections and link processing layers beyond adjacent neurons. These advances promise a greater understanding and ability of neural networks to tackle increasingly complex tasks.

### 2.5.3 Introduction to LSTM

Within the broad spectrum of neural networks, LSTM (Short-Term and Long-Term Memories) represent a specialized variant of recurrent neural networks (RNNs), the first appearance of the LSTM in the literature was in the *LONG SHORT-TERM MEMORY* published by Sepp Hochreiter and Jürgen Schmidhuber in 1997 [30]. Unlike traditional RNNs, LSTMs are designed to address the challenge of retaining and remembering relevant information over time in sequences of data. Their unique architecture allows them to capture long-term temporal dependencies, making them particularly well suited for tasks involving natural language processing, text generation, and time series analysis. In the next section, the exploration will delve into more detail on how LSTMs have revolutionized the field of sequential learning and significantly improved the performance of neural networks in a variety of applications.

## 2.6 Long-Short Term Memory

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) designed to handle the challenges of modeling sequential data. Unlike traditional RNNs, LSTMs are capable of learning long-term dependencies in data, making them particularly well-suited for applications involving time series analysis, natural language processing, and financial market predictions. [31]

### 2.6.1 LSTM architecture and functionality

The LSTM architecture consists of memory cells that maintain and update internal states over time. Each cell is equipped with three types of gates: input gates, forget gates, and output gates. These gates regulate the flow of information into and out of the memory cell, allowing LSTMs to learn and retain information over long periods of time. This unique architecture enables LSTMs to model complex patterns in sequential data with remarkable accuracy [32]. The 2.8 LSTM architecture [33]. compares the internal architecture of a standar NN, a standar RNN and a LSTM NN.

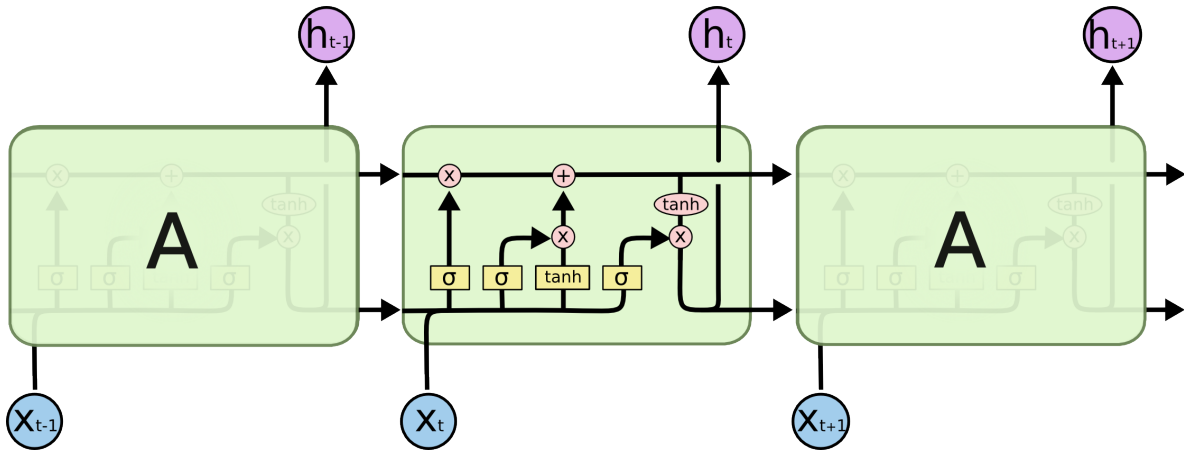


Figure 2.8: LSTM architecture [33].

### 2.6.2 Advantages of LSTMs

LSTMs offer several advantages over traditional RNNs, including:

- Ability to learn long-term dependencies: LSTMs can learn patterns and relationships in data that span hundreds or even thousands of time steps. [34] [35]
- Resistance to vanishing gradients: LSTMs are designed to mitigate the vanishing gradient problem, which can occur when training RNNs. [35] [36]
- Improved modeling of sequential data: LSTMs can model complex patterns and relationships in sequential data, making them well-suited for applications involving time series analysis and natural language processing. [36]

### 2.6.3 Applications of LSTMs

LSTMs have been successfully applied to a wide range of applications, including:

- Natural Language Processing: LSTMs have been used for language modeling, machine translation, and text summarization.
- Time Series Analysis: LSTMs have been used for forecasting, anomaly detection, and pattern recognition in time series data.
- Speech Recognition: LSTMs have been used for speech recognition and speech synthesis.

### 2.6.4 Financial market predictions with LSTM

In the financial domain, LSTMs have been used to predict stock prices, exchange rates, and other financial metrics. By modeling the temporal sequence of financial data, LSTMs can identify patterns and trends that are not apparent through traditional analysis techniques. This enables investors and analysts to make more informed decisions about investments and trading strategies. The use of LSTMs in financial market predictions has been shown to improve the accuracy of predictions and reduce the risk of losses.

## 2.7 Transformers

Transformers are a type of neural network architecture introduced in 2017 by Vaswani et al [3]. They revolutionized the field of natural language processing (NLP) by providing a novel approach to sequence-to-sequence tasks. Unlike traditional recurrent neural networks (RNNs), Transformers rely solely on self-attention mechanisms to process input sequences in parallel, making them faster and more efficient. In the figure 2.9 Transformer architecture [3]. it is shown the architecture of a classic transformer.

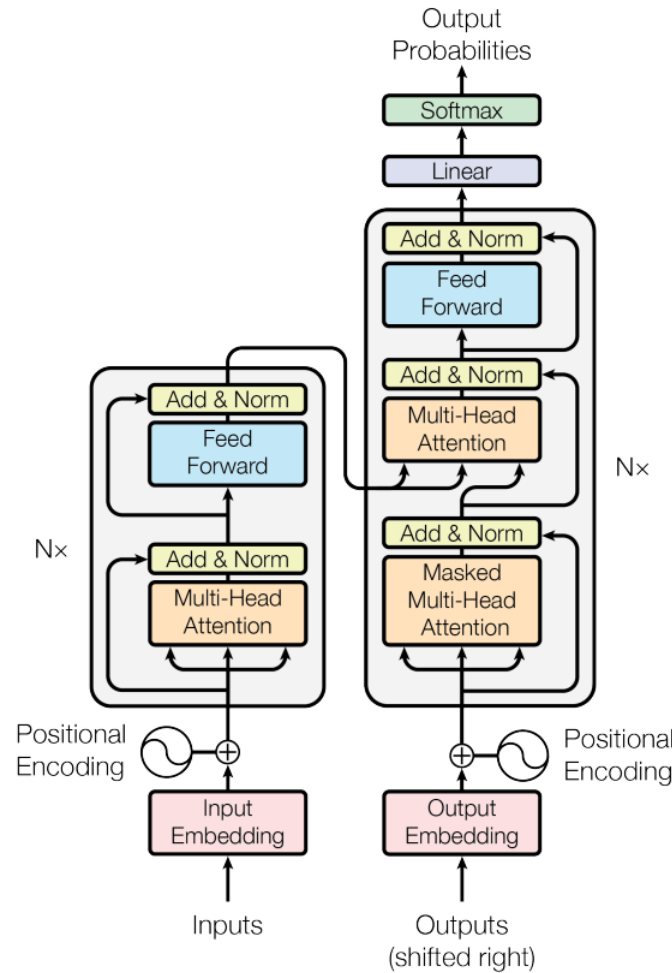


Figure 2.9: Transformer architecture [3].

### 2.7.1 Encoder

The encoder is the part of the Transformer architecture that takes in a sequence of tokens (e.g. words or characters) and outputs a continuous representation of the input sequence. The encoder consists of a stack of identical layers, each of which applies self-attention transformations to the input sequence.

### 2.7.2 Self-Attention mechanism

The self-attention mechanism is the core component of the Transformer architecture. It allows the model to attend to different parts of the input sequence simultaneously and weigh their importance. This is different from RNNs, which process the input sequence sequentially and have recurrence connections that allow them to capture long-range dependencies. [3] The self-attention mechanism consists of three main components [37]:

- **Query:** The Query represents the processed information of the current word. It's a matrix that helps in the scoring process to see how relevant other words are to the current word.



- **Key:** The key represents the processed information of all the words in the sentence, including the current word. It's used to compute a score that represents the relationship between different parts of the sentence.
- **Value:** The value represents the raw information of all words in the sentence. Once the scores between different parts of the sentence are computed, they are used to weight the Value matrix, which in turn gives an aggregated representation of the words in context.

The Query is often derived from the word for which you want to calculate the attention score. Meanwhile the Key and Value are derived from all the words in the context, including the current word itself [37]. The self-attention mechanism computes the attention weights by taking the dot product of the query and key matrices, and then applying a softmax function to the result. The attention weights are then used to compute a weighted sum of the value matrix, which produces the output of the self-attention mechanism.

### **2.7.3 Multi-Head attention**

The Transformer architecture uses multi-head attention, which allows the model to jointly attend to information from different representation subspaces at different positions. This enables the model to capture complex contextual relationships between different parts of the input sequence.

### **2.7.4 Output linear layer**

The output linear layer is a linear layer that is applied to the output to produce the final output sequence. The output linear layer is used to transform the output into a probability distribution over the possible tokens.

### **2.7.5 Applications of transformers**

Transformers have been widely adopted in NLP tasks [38], including:

- **Machine Translation:** Transformers have achieved state-of-the-art results in machine translation tasks, outperforming traditional sequence-to-sequence models.
- **Language Modeling:** Transformers have been used for language modeling, achieving better results than traditional RNN-based models.
- **Text Classification:** Transformers have been used for text classification tasks, such as sentiment analysis and spam detection.

### **2.7.6 Financial applications of transformers**

Transformers have also been applied to financial applications, including:

- **Financial Text Analysis:** Transformers have been used for financial text analysis, such as sentiment analysis of financial news articles. [39]

- Time Series Forecasting: Transformers have been used for time series forecasting, such as predicting stock prices and exchange rates. [40]

Transformers have revolutionized the field of NLP by providing a novel approach to sequence-to-sequence tasks. Their ability to process input sequences in parallel and capture complex contextual relationships makes them particularly well-suited for tasks involving sequential data. Their applications extend beyond NLP to financial applications, where they have shown promising results. Therefore, in this project, the focus has been on leveraging the incredible performance of transformers in NLP to tackle time series prediction in financial markets.

### Section summary

Section 2 Technological framework provides a deep look at the technological framework of the project, focusing on the application of AI systems to financial time series analysis. It begins with an overview of Artificial Intelligence, detailing its evolution, key concepts, and various learning paradigms such as supervised, unsupervised, and reinforcement learning. AI is followed by the machine learning section where the different types of machine learning are described in more detail. In addition, there is a case of use for each type of machine learning. The section then goes deeper into deep learning, highlighting advancements in neural network architectures, including convolutional neural networks (CNNs) and recurrent neural networks (RNNs), and their impact on fields like image recognition and natural language processing. The role of Long Short-Term Memory (LSTM) networks in handling sequential data is also discussed, particularly their advantages and applications in tasks like language processing and financial market predictions. Finally, the section explores Transformer architectures, emphasizing their revolutionary self-attention mechanisms and their success in natural language processing and financial forecasting, demonstrating the wide applicability and transformative potential of these technologies in this project's context.

### 3 Design specifications and requirements

#### 3.1 Design specifications

The primary objective of the project is to carry out a comprehensive comparison of the effectiveness of transformer models with other advanced artificial intelligence technologies in multivariate financial time series forecasting. This comparison will focus on the analysis of financial data from 15 different stocks that are part of the S&P 500 index, one of the most representative stock indexes of the U.S. and global financial market.

##### 3.1.1 System flow diagram

In figure 3.1 it is shown an example of the system's flow diagram. This example shows two different flows, one in which the whole model is executed, and another one in which the only thing you want to execute are the results. If the whole model is executed, the main will act as an orchestrator and will execute sequentially in order each of the instructions passed by the `-o` parameter in the CMD. For obvious reasons, it is recommended that the last one is always the one of the data visualization to be able to obtain the results of all the launched modules.

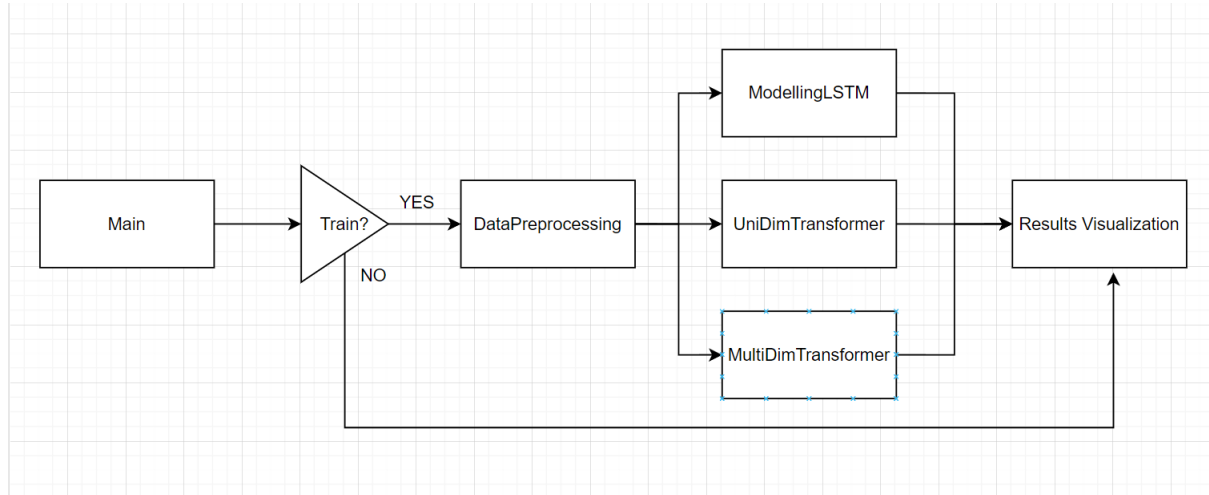


Figure 3.1: Flow diagram [self elaboration].

##### 3.1.2 System functionality

- Users can choose to either run the entire system or solely run the modules that they prefer. Initially, running the entire system is necessary to generate input data and results from the training and testing processes of the models.
- Processed data is stored in the following directory flow: `dataprocessed > input > window size > train/test ratio`. The outputs of the models are saved as a pickle files named as: *scenario\_name-model-output.pkl* for the univariate models and *scenario\_name-model-m-output.pkl* for the multivariate model. And follows this directory structure: `dataprocessed > output > window size > train/test ratio`.
- Users can customize simulation scenarios by editing the *config.yaml* file.

- For the LSTM models, users can choose from three different types: classical LSTM, stacked LSTM and attention-based LSTM to compare the results with transformers.
- In the multivariate transformer, users can select the features that wants to use for the training process from the 10 features available. It can be chosen from 1 to 10 features.
- The visualization process, presents the performance of each model for individual stocks across different time periods (1 day, 1 week, 2 weeks, 1 month and 3 months)
- The comparisons will be provided grouped by stocks, comparing models, and grouped by models, comparing stocks.

As mentioned before, users can select which module or part of the system want to run. This creates an enormous flexibility execution of the system because users don't have to wait hours for the system to finish executing all modules, they can just execute the modules they prefer. This design decision was taken in order to create a multiple, modular, fully integrated and flexible execution, thanks to a vectorized configuration.

The selection of execution of the different modules is done through the `-o` parameter when the *main.py* of the system is launched and can have the following values:

- `pre`: launches data preprocessing
- `lstm`: launch the lstm model
- `1DT`: runs the univariate transformer
- `MDT`: run the multivariate transformer
- `post`: run the graphical results

To be able to launch more than one module at a time, it must be done by separating each of the values by a semicolon ";". If users run more than one at the same time, the modules will be launches one after the other in a secuencial way.

## 3.2 Requirements

For the development of the project, the following constraints have been taken into account:

- **Requirements install:** In order to run the system, users must have install the *requirements.txt* file in their environment. This file contains all the python libraries with their correct versions to run the program.
- **Scenario structure:** one of the main constraints is the design of the structure of the simulation scenarios. This structure can be found in the *config.yaml* file of the project code and allows the user to create scenarios according to his interests.
- **Use of GPUs for performance and scalability enhancement:** Efforts have been made to optimize performance and execution times in the training process of each of the models, and for this purpose, GPUs have been chosen for the transformers. This greatly increases the scalability of the models, since the training speed of a GPU is approximately 10 times faster than that of a CPU as Ricardo Moya mentions in his article [41].

- Use of open source tools: A decision of this project has been to use open source technologies and tools. With this restriction, access to free software and collaboration between communities of programmers is sought.
- In transformers models, when using multi-head attention, the embed dimension must be divisible by the number of heads. This restriction is created because the embed dimension will be split across number of heads. Therefore each head have dimension `embed_dim // number_heads`.

## Section summary

Section 3 Design specifications and requirements details the system flow diagram and functionality, allowing users to select which system modules to run, allowing great flexibility by avoiding the need to run all modules. This design decision, achieved through a vectorized configuration, allows for multiple, modular and integrated execution. In addition, users can customize their simulations parameters via the `config.yaml` file. Processed data and model outputs are stored in a specific directory structure. The section also discusses the project's constraints, including the requirement to install dependencies listed in *requirements.txt*, the use of GPUs to enhance performance and scalability, and the decision to use open-source tools to encourage collaboration and access to free software.



## 4 Description of the proposed solution

The solution proposed in this project addresses the need to develop a price prediction system for financial markets by implementing artificial intelligence models in python, specifically transformers, in a multivariate environment. To achieve this goal, an architecture is designed that allows the comparison between transformers and other AI technologies, applied to data from 15 different stocks within the S&P 500 index.

The proposed architecture consists of several key components that are integrated to efficiently manage financial data and facilitate the price prediction process. Data is collected from diverse sources, including sentiment analysis from Twitter and Bloomberg, daily trading volume, and factors related to stock prices. This data is preprocessed and stored in pickle files. The architecture allows the user to customize simulation scenarios through a YAML configuration file, providing the flexibility to adapt to various forecasting examples and contexts. In addition, the user can choose between different types of LSTM models, such as classical LSTM, stacked LSTM and attention-based LSTM, to compare their results with the transformer models. The LSTM models development have been done with Tensorflow library, which is one of the most used python libraries for AI and the transformers models have been developed in PyTorch, the other most used library for AI in python.

The visualization of results is carried out by representing the performance of each model for each action and for different time horizons, which facilitates the comparison and evaluation of their effectiveness in different contexts to compare their results with the transformer models.

### 4.1 Data study

In order to carry out this project, the first part even prior to the development is the study of the data to be used. As mentioned in the section 1 Introduction, the data used in this PFG are financial data that collect certain characteristics of 15 stocks of the S&P 500 index from 2015 to the end of 2023. All these data have not been obtained in this project but have been provided by the Department of Ingeniería de Organización, Administración de Empresas y Estadística of the ETSIST. Partial data files example:

1	date	TWITTER_SENTIMENT_DAILY_AVG	TWITTER_PUBLICATION_COUNT	TWITTER_NEG_SENTIMENT_COUNT	TWITTER_POS_SENTIMENT_COUNT	TWITTER_NEUTRAL_SENTIMENT_CNT	NEWS_SENTIMENT_DAILY_AVG	NEWS_PUBLICATION_COUNT	NEWS_NEG_SENTIMENT_C
2	2015-01-01	-0.0938	2703.0	539.0	136.0	1604.0	-0.1934	964.0	64.0
3	2015-01-02	-0.0981	1782.0	367.0	66.0	1085.0	0.1099	979.0	23.0
4	2015-01-05	-0.0984	2386.0	354.0	102.0	1005.0	0.0559	1367.0	13.0
5	2015-01-06	-0.0374	4272.0	575.0	283.0	2743.0	0.191	1966.0	28.0
6	2015-01-07	0.0197	5719.0	571.0	602.0	3910.0	0.1275	2298.0	39.0
7	2015-01-08	0.0276	3676.0	302.0	438.0	2429.0	0.0932	2099.0	29.0
8	2015-01-09	0.0863	5718.0	354.0	1079.0	3311.0	0.2387	2521.0	42.0
9	2015-01-12	0.1252	1830.0	46.0	396.0	1029.0	0.0472	1119.0	13.0
10	2015-01-13	0.0502	3554.0	345.0	533.0	2180.0	0.1217	2002.0	43.0
11	2015-01-14	-0.0276	5239.0	603.0	562.0	3434.0	0.2851	2602.0	103.0
12	2015-01-15	-0.0363	4578.0	638.0	300.0	3060.0	0.1497	2445.0	92.0
13	2015-01-16	-0.0349	5484.0	747.0	299.0	3622.0	0.119	2248.0	88.0

Figure 4.1: Dataset example

In figure 4.1 it is shown and example of the first 13 rows of the AAPL's stock dataset with its first 8 columns. It can be detected that the scales of these first rows are completely different, so it is needed a normalization process.

### 4.1.1 Stocks

The stocks studied in this project are as follows: AAPL: Apple, ADBE: Adobe, AMZN: Amazon, AVGO: Broadcom, CMCSA: Comcast, COST: Costco, CSCO: Cisco, GOOG: Alphabet (Google), GOOGL: Alphabet (Google), META: Meta Platforms (Facebook), MSFT: Microsoft, NVDA: NVIDIA, PEP: PepsiCo, TMUS: T-Mobile, TSLA: Tesla. Among these 15 are the shares of the so-called "Magnificent 7", which are the technology megacaps stocks that, as of today, weigh about half of the NASDAQ [42], biggest technological stock index in the US.

### 4.1.2 Features

All the data files for each of the stocks contain the same 17 columns or variables, which are as follows:

- DATE: timestamp of the record
- TWITTER\_SENTIMENT\_DAILY\_AVG: average of the daily sentiment in Twitter/X
- TWITTER\_PUBLICATION\_COUNT: number of daily publications in Twitter/X about that stock
- TWITTER\_NEG\_SENTIMENT\_COUNT: count of negative sentiment tweets in Twitter/X
- TWITTER\_POS\_SENTIMENT\_COUNT: count of positive sentiment tweets in Twitter/X
- TWITTER\_NEUTRAL\_SENTIMENT\_CNT: count of neutral sentiment tweets in Twitter/X
- NEWS\_SENTIMENT\_DAILY\_AVG: average of the daily sentiment in news articles about the stock
- NEWS\_PUBLICATION\_COUNT: number of daily news articles about the stock
- NEWS\_NEG\_SENTIMENT\_COUNT: count of negative sentiment news articles about the stock
- NEWS\_POS\_SENTIMENT\_COUNT: count of positive sentiment news articles about the stock
- NEWS\_NEUTRAL\_SENTIMENT\_COUNT: count of neutral sentiment news articles about the stock
- PX\_HIGH: highest price of the stock on that day
- PX\_LOW: lowest price of the stock on that day
- PX\_OPEN: opening price of the stock on that day
- PX\_LAST: closing price of the stock on that day
- VOLUME: trading volume of the stock on that day
- RSI\_14D: 14-day Relative Strength Index (RSI) of the stock



As mention before, these are the columns of the raw data in the csv files but these are not the features used in the training process. These features are generated and explained in the data preprocessing, in the next section.

The figure 4.2 Correlation matrix of scaled features [self elaboration]. represents the correlation between the different features of the data. As it can be observed, price-related features are highly correlated with each other, as are trend features and sentiment analysis. It is important to highlight that volatility, even though it is derived from daily price changes, it has almost no correlation with any other stock. Knowing this, it can be concluded that it has sense to create a simulation scenario with one or two features of each group of correlated features in addition to volatility.

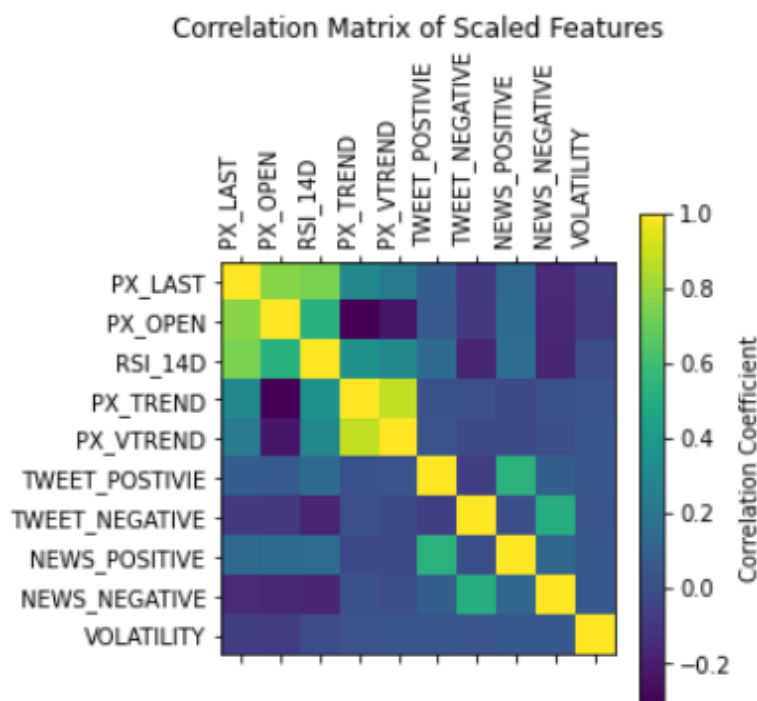


Figure 4.2: Correlation matrix of scaled features [self elaboration].

## 4.2 Preparation for simulation scenarios

In this architecture, the simulations are made using a configuration yaml file where users can insert the values they want for the presented keys. It can be change everything in the file like paths, date of the data, filename. Inside the scenarios, users can select the tickers of the stocks they want to use for their simulations, training/test split ratio, window size, number of iterations of the training process, epoch, batch size, number of hidden neurons. In the LSTM section of the scenario users can choose between the 3 LSTM models developed for this project (classic, stacked and attention-based) and in the transformer section, users can the number of head for the multi head attention, number of layer and the features selected for the multivariate date.

The following parameters are lists train test ratio, window size, list of ahead, tickers and features. This decision has been taken in order to facilitate users the creation of simulations with the values they prefer. In case users want to predict for other ahead periods, or to compare between different window size, this provide a easy way to do it.

With this structure what has been sought has been to create a file where every parameter can be change and to create a system that is as generic and automatic as possible in order to compare results for multiple different parameters.

### 4.3 System Operation

The architecture of this project is a flow of python scripts that are executed an structured by a main script which manage the execution of different the modules. In the first place users will decide which modules of the system they want to run. The use of this system's features is related to the parameters that the user inputs when user runs the main script. Users have to run the main script with the parameter `-c config_file_path` which is the path of the configuration file to run the system, and with the parameter `-o [operations]` which is a list of the modules to run. If users execute a module that has no data to work on, for example no csv files for the data preprocessing module or no processed data for the transformer modules, it will execute and error warning that user have to run some other modules before that. When users select the options, the system will start running.

As shown in the figure 4.3 and as mentioned before, if the input in not valid, the system asks users to enter a new valid input.

```
parser.add_argument("-c", "--params_file", required=True, nargs='?', action='store', help="Configuration file path")
parser.add_argument("-o", "--operations", required=True, help="Operations to run: 'pre;lstm;1DT;MDT;post'")
```

Figure 4.3: System operation [self elaboration].

#### 4.3.1 Data Preprocessing

The first step is the data preprocessing, in this script raw data in csv files is loaded to process it. Each csv file has the data for each stock so in the same file is impossible to have data from different stocks. In order to keep this structure, the decision taken was to create a stock class for each stock as shown in the 8.1. With this implementation, each Stock object will have its own data, parameters and processing process (every step of the preprocessing flow is a method of the class Stock). This step can be separated in three threads.

The first one is the transformation of the common features for the univariate and the multivariate data. The second thread is the process of the univariate data and the final thread is the process of the multivariate data.

Starting with the preprocessing, the first part is the computation of the sentiment scores. This method adapts the sentiment analysis from the raw data to the specify form. In this case, this method add four new columns to the dataset, positive and negative ratio for Twitter/X and news sentiment analysis. For example, the ratio for the positive tweets has been calculated as the count of positive publications divided by the total of publications.

The second part is the calculation of the volatility. In this case the way used to compute the volatility of the stock has been the standard deviation of the daily percentage change in the stock price. Despite the volatility was not as raw data column, it is one of the most important metrics in the markets data. Volatility is used in a wide range of formulas, systems and applications in the financial engineering and quantitative finance field [43].

Following the volatility, the next step is the trend's calculation. For this project, it is compute

not only the price trend but the volume trend too. The reason for this decision is because volume is a metric commonly used by traders to try to predict the market. The next steps are the drop of the nans values, because the percentage of nans compare to the total data is not relevant so it can be drop it, and the detection of infinite values.

Once the common features are processed, it starts the second thread with the univariate data preprocessing. This thread starts with the windowing process of the data creating the new features matrix with the windows size extracted from the config file. Then, the data is adapted to the ahead value for the different periods prediction. And finally starts the normalization process, in this case, the normalization method applied is the min-max normalization following the formula of 4.1:

$$\frac{(x - \min)}{(\max - \min)} \quad (4.1)$$

The normalization had been applied manually instead of using a python library in order to ensure that it normalizes all the data and to avoid possible registers normalization failure. The min and max values had been calculated with the matrix  $x$  centered in the mean instead of calculate them just using the  $x$ . Finally, with the data normalized, the final  $x$  DataFrame and  $y$  pandas Series are divided in train and test sets using the percentage established in the config file. As for the normalization, the split in train and test has been done manually too, using the `iloc` method with slicing to save just the pmod first registers for train and the last for testing.

Finally, when the univariate data is preprocessed, starts the multivariate data preprocessing. Conceptually, it is the same preprocessing as the univariate data with the same normalization and the same processes. But as this are bigger arrays, they have to be treated in a different way.

The main difference with the univariate data is, obviously, that features matrix is created with all the features, but in the models training process, users can select to used the features they prefer. In the univariate data the features matrix uses only the close/last price of the stock, which is the same feature that the models will predict. Another big difference is the way the data is saved, in univariate data as the shape of the data has only 2 dimension (number of rows and window size), it can be stored in pandas DataFrame. When it comes to multivariate data, as data shape is (number of rows, windows size and features), it can't be stored as DataFrame so it is stored as numpy array. In figure 4.4 it is shown the Stock class with its parameters for each stock and the methods that preprocess the data

Stock
df : DataFrame mserial_dict : dict scen_name serial_dict : dict ticker tr_tst
calculate_max(data, axis) calculate_mean(data, axis) calculate_min(data, axis) check_infinite_values(df) check_nan_values(train_x, test_x, ahead) compute_sentiment_scores() compute_trend() compute_volatility() prepare_data_for_modeling(x_norm, y_norm, multi) prepare_multivariate_data(df_x, win, n_fts, idx) process_multivariate_data(mwin, m_fts) process_stocks() process_univariate_data(win)

Figure 4.4: Stock Class [self elaboration].

### 4.3.2 LSTM Models

Once the data is fully preprocessed, stored in pickle files and ready to be used as inputs of the model, the second step in the system is the training and evaluations of the LSTM models. As mentioned in the 1 Introduction and also in the 4.2 Preparation for simulation scenarios subsection in this project three types of LSTM has been developed. In the config file of scenarios simulation users can choose which of the three types of LSTM they want to use and also the hyperparameters to build the models. The functionality of this step is to train the chosen model and to save the output of the testing process. The codes structures for each of the three models are in the appendix section 8.3. Here in 4.5LSTM Class [self elaboration]. it is shown how the LSTM training process has been developed. It is a python class that has all the parameters needed to train each model. The model creation, train and test is done in the methods. As shown in the 4.5LSTM Class [self elaboration]. each LSTM model has a method for all the process.

TrainLSTM
Y
bsize
epoch
n_fts
nhn
stock
testX
testY
trainX
trainY
vdd
win
att_lstm_fun(ahead, seed)
lstm_fun(ahead, seed)
stck_lstm_fun(ahead, seed)

Figure 4.5: LSTM Class [self elaboration].

Basic LSTM model: The first model is a basic LSTM designed to capture patterns in the financial datasets. This model it is build with a single LSTM layer followed by a dense layer. The LSTM layer has neurons with a ReLu as activation function, and the dens layer produces the final output. This approach allows the model to learn short-term dependencies in the time series. Training process is performed using the training dataset obtained in the preprocessing stage explained int the 4.3 System Operation section. In the 4.6Basic LSTM [self elaboration]. it is shown the architecture described in a graphical way. In this particular case, the parameters of the network are a window size of 5 and 64 as number of hidden layers. This parameters are also used for the graphical representation of the two other models.

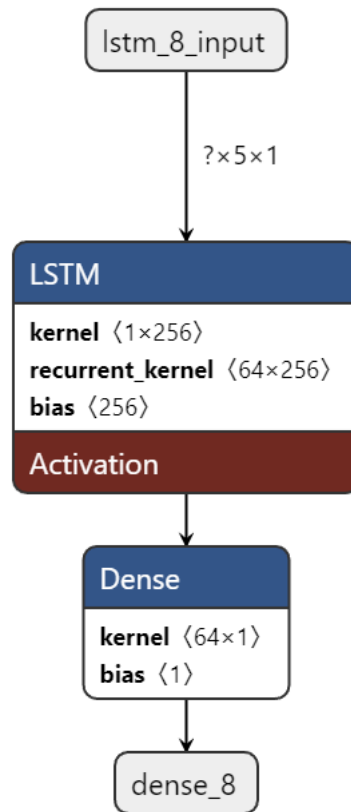


Figure 4.6: Basic LSTM [self elaboration].

**Stacked LSTM Model:** This second model is a more complex LSTM version. In this case, the architecture includes three sequential LSTM layers. Each LSTM layer is configured to return sequences except the last one which produces the output. This configurations allows the model to capture more complex and deeper patterns in the time series compare to the basic LSTM. The model's output is generated by a dense layer as in the first model. The training and evaluation processed are similar to the basic one. The architecture is shown in the 4.7Stacked LSTM [self elaboration].

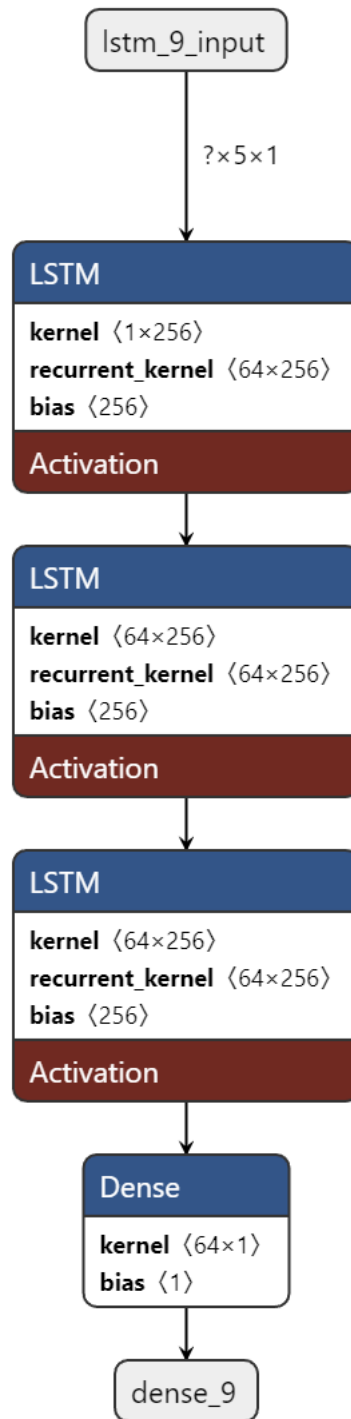


Figure 4.7: Stacked LSTM [self elaboration].

**Attention-based LSTM:** The third and more complex model which incorporates an attention mechanism along with LSTM and GRU (Gated Recurrent Unit) [44] architecture. This model includes an embedding layer to represent the input sequences, followed by the GRU layer with an automatic attention mechanism. This attention mechanism allows the model to focus on relevant parts of the input sequences, improving prediction accuracy in tasks where certain moments in time are more important than others. For example, in financial data for stock predictions, the most recent records are more likely to be more important and are correlated to the prediction than older records of the sequence. The last layer is a dense layer as the other two models.

In addition, this model also includes a regularization technique named model checkpointing to prevent overfitting and ensure the best version of the model during training process. The architecture is represented in the 4.8Attention-based LSTM [self elaboration].

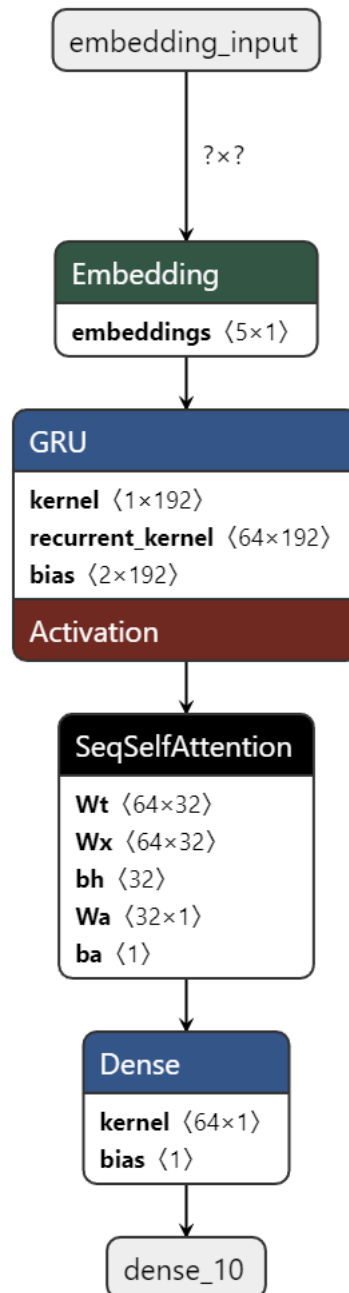


Figure 4.8: Attention-based LSTM [self elaboration].

For each model, after training, predictions are evaluated using two metrics: Mean Squared Error (MSE) and Mean Absolute Error (MAE). The results include comparing the predictions with the actual values and assessing each model's ability to generalize to unseen data. Each training iteration is validated through a series of tests to ensure the stability and robustness of the models.



### 4.3.3 Transformer models

In this section, the proposed solution for the transformer model for multivariate time series forecasting is presented. The architecture and operation of the model are detailed below, as well as the justifications for the design decisions adopted. All the transformer's parameters are taken from the *config.yaml* file so they can be changed everytime users want without opening the transformer's code.

#### Transformer model architecture

The transformer model developed for this project is based on the classic Transformers architecture introduced by Vaswani et al. (2017) [3] and adapted for time series analysis. The architecture includes the following key components:

**Embedding layer:** Description: The embedding layer is a linear layer that converts input features into a higher dimensional representation (embedding dimension). This layer is essential for mapping input features to a richer feature space, making it easier to capture complex relationships in the data.

**Encoder layers:** This Transformer includes multiple encoder layers, implemented using PyTorch's `TransformerEncoderLayer` class. These layers are essential components that allow the model to learn complex patterns and long-term dependencies in the time series. The structure, operation and justification of the encoder layers are detailed below.

Each `TransformerEncoderLayer` consists of two main sub components:

**Multi-Head attention layer:** This sub layer performs several attention operations in parallel (defined by the number of heads, number of heads). Each attention head computes a different representation of the input sequences, allowing the model to capture different aspects of the dependencies between sequence positions. The input to the sub layer is processed using a dot-product scaled attention operation, followed by a combination of the outputs of all heads using a linear layer. Multi-head attention allows the model to attend to different parts of the input sequence simultaneously, which is crucial for capturing complex patterns and long-term relationships in time series.

**Positional feed forward network:** This sub layer consists of a feed forward neural network that is applied independently and in parallel to each position of the input sequence. Typically, this network has two linear layers separated by a nonlinear activation (ReLU). Each position vector in the sequence is processed individually through the feed forward network, allowing nonlinear features and complex transformations of the inputs to be captured. The positional feed forward network provides additional nonlinear transformation capability to each position in the sequence, improving the model's ability to represent and learn complex features.

In addition to these two sub components, each `TransformerEncoderLayer` includes dropout mechanisms:

**Dropout:** A dropout is applied after the multi-head attention and feed forward network to prevent over fitting. Dropout regularizes the model by randomly turning off neurons during training, improving its generalization ability.

Design decisions related to encoder layers are justified by their ability to handle the complexities inherent to time series: Multi-Head attention: Using multiple attention heads allows the model to consider multiple representations of the input sequence simultaneously, capturing various

temporal relationships and long-term dependencies. This is particularly useful in this project for the financial time series, where dependencies can vary over different time horizons from 1 day to 90 days.

**Positional feed forward network.** The addition of feed forward networks in each layer of the encoder allows nonlinear transformations to be applied to each positional vector, enriching the model's ability to represent and learn complex patterns in the data.

**Dropout.** These mechanism helps stabilize training and prevent over fitting, ensuring that the model generalizes well to unseen data. Dropout adds a layer of regularization necessary to prevent the model from over fitting the training data.

**Output layer:** Description: A final linear layer that converts the encoder output into the desired prediction. The output layer maps the learned feature representations to the final predictions, allowing specific results for the prediction task.

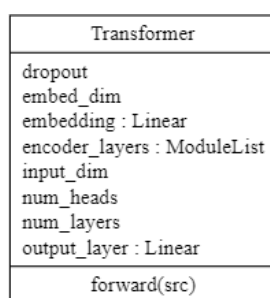


Figure 4.9: Uni/multi dimension transformer architecture proposed [self elaboration].

### Transformer Model Operation

The Transformer model training and prediction process takes place in several fundamental steps, starting with data preparation. The input and output data are converted to PyTorch tensors, ensuring that the values are suitable for processing in a deep learning environment. The inputs are restructured to have the necessary shape to be processed by the model, which includes adapting the dimensionality so that each input tensor fits the requirements of the Transformer.

At the start of training, the model is configured using the AdamW optimizer and the mean squared error loss function (MSELoss). The AdamW optimizer is chosen for its efficiency and effectiveness in fast convergence, while MSELoss is appropriate for regression tasks where the objective is to minimize the quadratic difference between the predictions and the actual values. During training iterations, which are repeated in each epoch, the model processes batches of data based on the batch size (batch size). In each batch, the model weights are adjusted through the back propagation process, allowing the model to learn patterns and relationships in the training data.

Throughout training, continuous evaluation of the model is performed on the test set after each epoch. This allows you to monitor the performance of the model and adjust the hyper parameters if over fitting problems are detected. Evaluating the model regularly is crucial to ensure that the predictions generated are generalizable and not over fitted to the training data.

Once training is completed, the model generates predictions that must be denormalized to obtain values on the original scale of the data. This step is vital to correctly interpret the predictions in the real context of the problem. The prediction error is then calculated using specific metrics

such as mean square error (MSE) and mean absolute error (MAE). These metrics provide a quantitative measure of model performance, evaluating the accuracy of the predictions and allowing comparison with other techniques or models.

This detailed approach at every stage, from data preparation to final evaluation, ensures that the Transformer model is not only accurate, but also robust and efficient in time series prediction, especially in financial contexts where accuracy is critical.

### Section summary

The section 4 Description of the proposed solution aims to develop an artificial intelligence-based price prediction system for financial markets, leveraging transformative models in a multivariate environment. The architecture designed for this solution facilitates comparison between transformers and other AI technologies, applied to data from 15 different stocks within the S&P 500 index.

The architecture integrates several key components to efficiently manage and predict financial data. Data is collected from various sources, including Twitter and Bloomberg sentiment analysis, daily trading volumes, and stock price-related factors. This data is preprocessed and stored in pickle files, while simulation scenarios can be customized via a YAML configuration file. This configuration allows flexibility to adapt to various forecasting examples and contexts.

Users can choose from different LSTM models, including classic, stacked, and attention-based LSTMs, to compare their performance with transformer models. LSTM models are developed using TensorFlow, while transformer models are built with PyTorch, reflecting the use of two predominant AI libraries in Python.

The system also includes robust visualization capabilities, presenting the performance of each model across different actions and time horizons. This visualization helps evaluate the effectiveness of each model in various contexts.

The data study involved the analysis of financial data of 15 S&P 500 index securities from 2015 to 2023, provided by the ETSIST Department of Ingeniería de Organización, Administración de Empresas y Estadística. Data features include Twitter and news sentiment, stock prices and trading volumes, among others.

Simulation scenarios are defined through a yaml file where users can specify parameters such as stock tickers, train/test split ratios, window sizes, number of training iterations, epochs, batch sizes, and the number of hidden neurons. This setup allows users to select between different LSTM models and customize transformer parameters, creating a generic, automated system for comparing multiple parameters.

The operation of the system is managed by a flow of Python scripts executed by a main script. Users can choose to run the entire system or only display the results if there are results from previous runs. The data preprocessing step involves loading raw data from CSV files, calculating sentiment scores, calculating volatility and trends, and normalizing the data for univariate and multivariate data sets. Univariate data uses close/last price features, while multivariate data includes all features, stored as numerous matrices due to their dimensionality.

For LSTM models, a basic LSTM captures short-term dependencies, a stacked LSTM captures more complex patterns, and an attention-based LSTM incorporates an attention mechanism for better prediction accuracy. The transformer model is based on the classical architecture, including embedding layers, encoder layers with multi-head attention, and positional feed forward

networks. It has a crucial difference with the classical architecture and this difference is that there is no decoder. In other applications like NLP for translation, the decoder is needed in order to know what is expected to get as output. For example, if the task is to translate from Spanish to English, if the input of the encoder is a word in Spanish, the input target of the decoder must be that word in English. But for this numerical time series, there is not a target data because there is only one value for each input that is the close price. Training involves the use of the AdamW optimizer and the MSELoss function, with continuous evaluation to monitor performance and avoid over fitting. Predictions are denormalized to original scales, and performance is measured using metrics such as MSE and MAE, ensuring the robustness and efficiency of the model in predicting financial time series.

## 5 Experiments and results

For this project, three different scenarios have been used to launch the simulations. Although some of the transformer and LSTM parameters are the same in every simulation, like number of hidden layers or embed dimension, users can select the value they want for the simulations. Of course, values can be all different to evaluate different scenarios.

Figures 5.1 , 5.2 and 5.3 shown the configuration of the scenarios used for the simulations in this project. These three scenarios has been designed after a test sub task. In this sub task it has been experimented with different values and selected some of them that can conclude with good results to compare.

```
- name: scenario_1
  tickers: ["AAPL", "ADBE", "AMZN", "AVGO", "CMCSA"]
  win: [5]
  tr_tst: [0.7, 0.8]
  lahead: [1, 7, 14, 30, 90]
  n_features: 1
  n_itr: 10
  epochs: 256
  batch_size: 32
  nhn: 64
  LSTM:
    model: "lstm" #lstm, attlstm, stcklstm
  Transformer:
    model: "transformer"
    num_layers: 4
    num_heads: 16
    features: ["PX_LAST", "RSI_14D", "PX_TREND", "PX_VTREND"]
```

Figure 5.1: Scenario 1 configuration [self elaboration].

```
- name: scenario_2
  tickers: ["AMZN", "GOOGL", "MSFT", "NVDA", "TSLA"]
  win: [20]
  tr_tst: [0.75, 0.85]
  lahead: [1, 7, 14, 30, 90]
  n_features: 1
  n_itr: 2
  epochs: 256
  batch_size: 32
  nhn: 64
  LSTM:
    model: "stcklstm" # lstm, attlstm, stcklstm
  Transformer:
    model: "transformer"
    num_layers: 2
    num_heads: 4
    features: ["TWEET_POSTIVIE", "TWEET_NEGATIVE", "NEWS_POSITIVE", "NEWS_NEGATIVE"]
```

Figure 5.2: Scenario 2 configuration [self elaboration].

```
- name: scenario_3
  tickers: ["AMZN", "PEP", "TMUS", "CSCO"]
  win: [2]
  tr_tst: [0.8, 0.875]
  lahead: [1, 7, 14, 30, 90]
  n_features: 1
  n_itr: 5
  epochs: 256
  batch_size: 32
  nhn: 64
  LSTM:
    model: "attlstm" # lstm, attlstm, stocklstm
  Transformer:
    model: "transformer"
    num_layers: 2
    num_heads: 8
    features: ["PX_LAST", "PX_TREND", "VOLATILITY"]
```

Figure 5.3: Scenario 3 configuration [self elaboration].

In order not to make the project report too long, the individual results that will be displayed will only be the AMZN stock results because is the stock repeated in the 3 scenarios and at the end it will be compared all the Amazon's results for the different scenarios. In addition, for the transformers it will be displayed the number of heads attention and the number of layers for comparison between different values of these parameters. The rest of the results are in the figures folder of this project's repository [45].

## 5.1 Scenario 1 results

Starting with the results for the first scenario. As shown in 5.1 Scenario 1 configuration [self elaboration]. the LSTM model used in it is the basic LSTM. The selected features for the multivariate transformer are four variables, three derived from the prices of the stock and one derived from the volume. The first results obtained are the performance of each model for every stock and for every ahead value. In this scenario, results are shown for the Amazon's stock. In each graph, the results are compared to a historical MSE. This metric is calculated by copying the previous day's closing price.

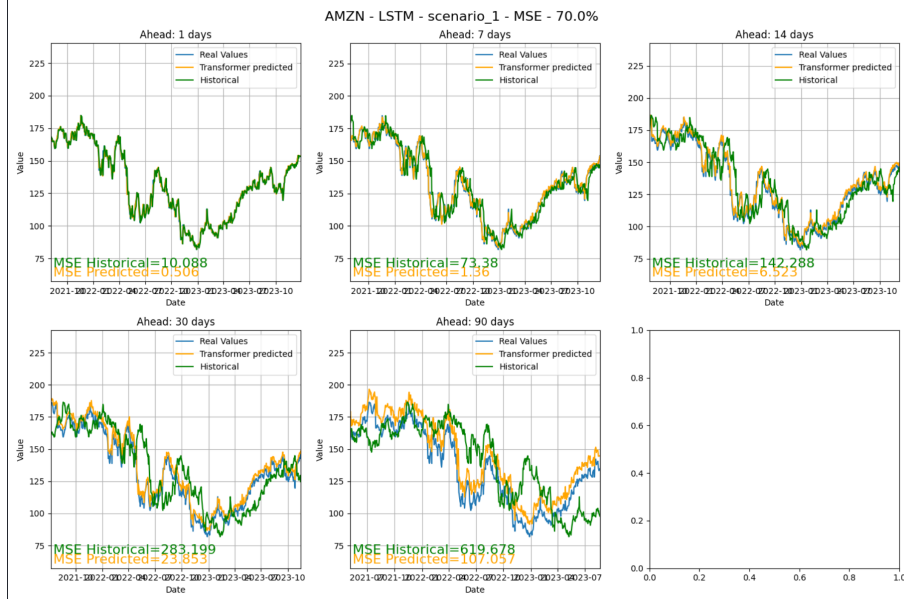


Figure 5.4: AMZN-LSTM-MSE-70% [self elaboration].

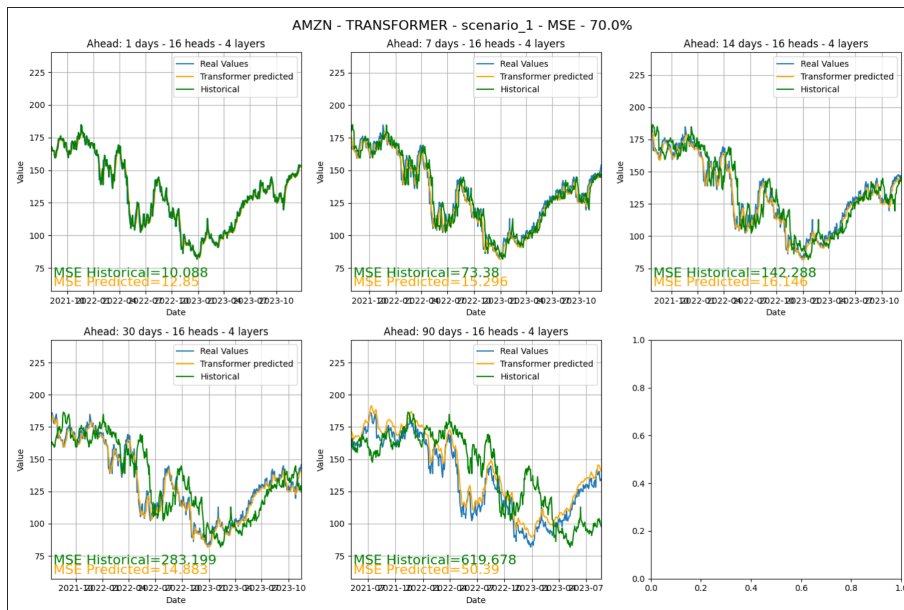


Figure 5.5: AMZN-Transformer-MSE-70% [self elaboration].

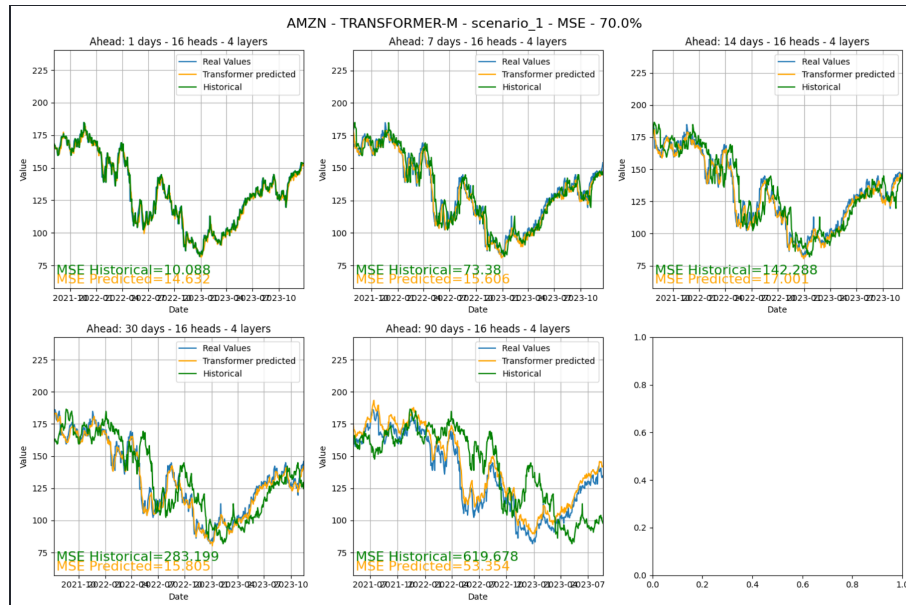


Figure 5.6: AMZN-TransformerMulti-MSE-70% [self elaboration].

Looking the three figures 5.4, 5.5 and 5.6 there are five plots for each figure, this is done because each subplot graphs the different values for the ahead forecasting. Starting from 1 in the upper left subplot. 7 in the upper mid subplot, 14 in the upper right subplot and 30 and 90 for the other two subplots in order. After examining these results, some conclusions are obtained.

- LSTM's results are highly accurate in the short term. Looking in the results, the MSE for the ahead values of 1, 7 and 14 days, the metric is so much lower than the transformers. Quantifying these words for ahead 1 day, the MSE in LSTM is almost a 96% lower than the MSE for the same ahead value in the transformer models. For the prediction of 7 days ahead, the MSE in LSTM is around a 90% lower than transformers and for the 14 days ahead forecasting, the metric is more than 60% lower. These results can be interpreted as the basic LSTM model having better performance for short-term investment prediction in the apple stock.
- However, the results of transformers are better than LSTM in the medium and long term, that is, 30 and 90 days. For the 30 days ahead prediction, transformers improve in a 40% the results of the LSTM and for the 90 days ahead it improves the results by more than a 52%.

With these results, it can be concluded that, at least for this simulation, LSTM perform so much better for short term predictions than transformers, but transformers do it better in medium and long term predictions which are usually more difficult to achieve good results.

Now, let's see a comparison between all the MSE for the list of stocks of the scenario.



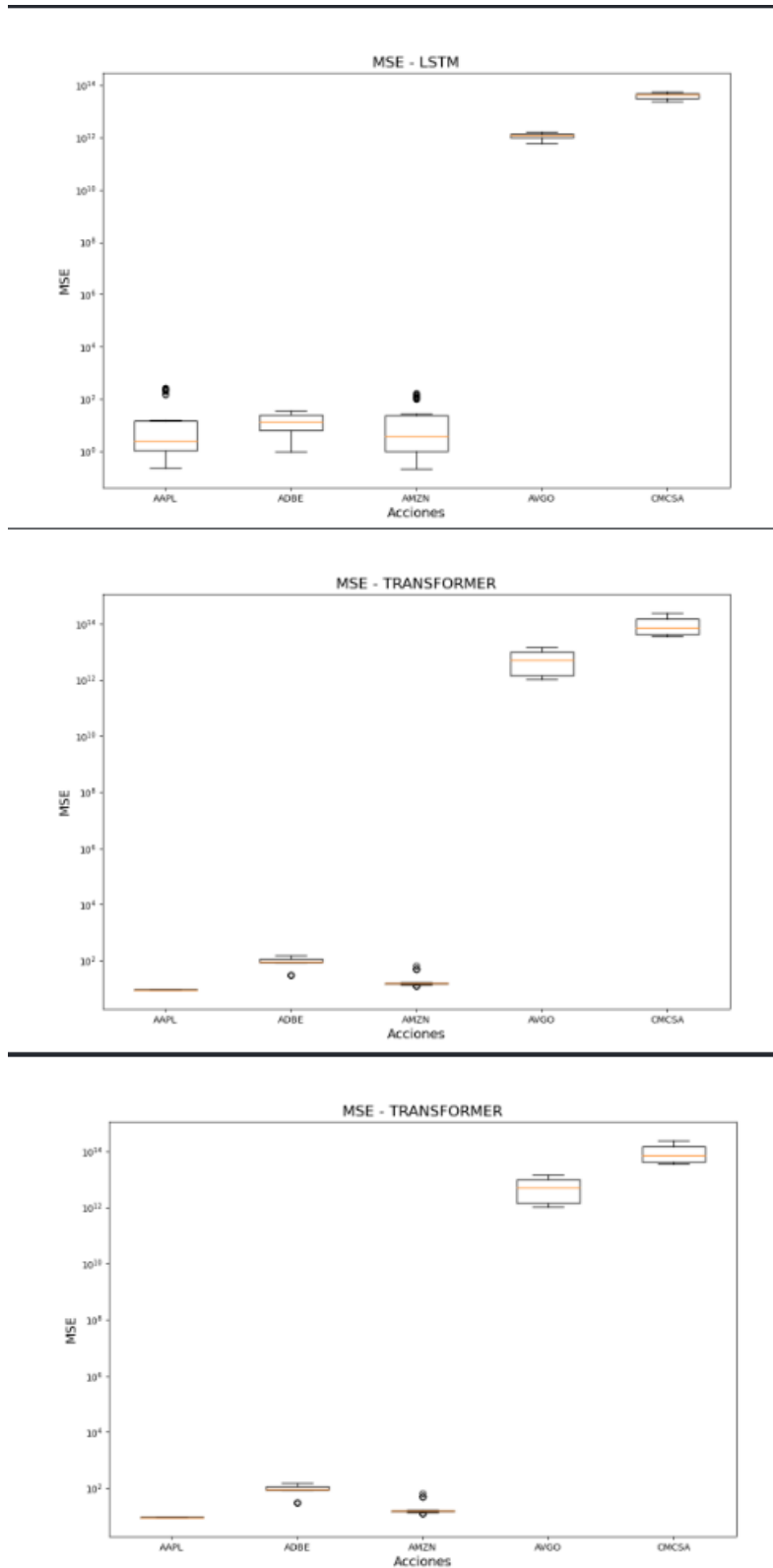


Figure 5.7: MSE stocks comparison scenario1 [self elaboration].

As shown in 5.7 MSE stocks comparison scenario1 [self elaboration]. each stock has its own MSE range, this can happen due to different price scales after the denormalization process. Focusing on each stock for the 3 models, it can be detected that for the 3 stocks that has a lower MSE

like are AAPL ADBE and AMZN, transformers get a better performance. This is shown in the box plot with a huge reduction of the box size which indicates that there is a really low variance of the MSE for the different ahead days predictions. As it can be seen in the figures 5.4, 5.5 and 5.6, the MSE in the transformer models, is more stable than the LSTM which has many more variations. It can be concluded that the Apple's stock is the stock with the best performance for the transformer models. Amazon's and Adobe's stocks has good results too but they have some outliers and are a little bit higher in the y axis than the Apple's stock.

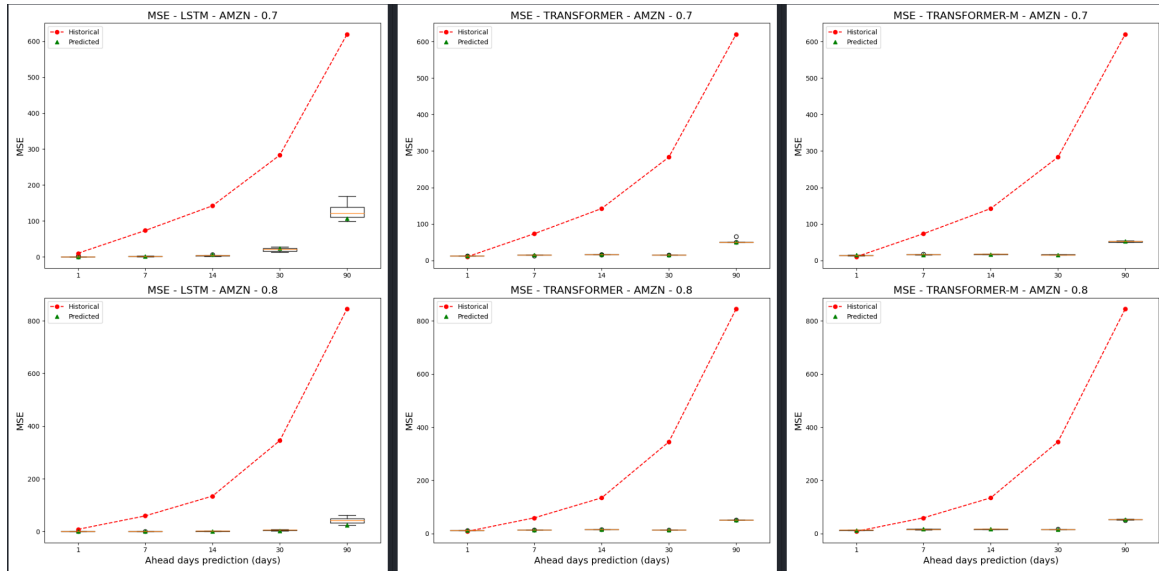


Figure 5.8: Train test split MSE comparison scenario 1 [self elaboration].

The results presented in the figure 5.8 Train test split MSE comparison scenario 1 [self elaboration]. compare the MSE for the Amazon's stock for different train test split percentages. It seems that for the three models, there is an improvement in the MSE when using a higher train test split percentage. For example, for LSTM 90 days, with the 70% boxplot has a wider range with the upper leg bigger than the bottom leg and with the mean line located in the lower part of the box. On the other hand, for the 80% the boxplot range values is considerably lower and the mean line is ubicated in the middle of the box. These results can be interpreted as that with the 80%, the model gets better predictions than with 70%. This pattern is replicated for the transformer models where the 80% gets better results than the 70% but the difference it is smaller than what happened with the LSTM.

The conclusions for this first scenario are:

- Using the 80% of the data for training is better than using the 70
- The LSTM model works better for the 1, 7 and 14 days ahead predictions.
- The transformers perform better for the 30 and 90 days ahead predictions.
- There is not a significant difference between the univariate and the multivariate transformers. This can happen because both are a multi-head attention transformer and even with a multivariate data set, the model cannot perform better for technical limitations.

## 5.2 Scenario 2 results

Following with the results for the second scenario. This scenario's configuration is shown in 5.2 Scenario 2 configuration [self elaboration]. the LSTM model used in it is the stacked LSTM, this model is a LSTM with three LSTM layers as explained in the 4.3.2 LSTM Models section. The features selected for the multivariate transformer are four variables, all four of which are related to the sentiment score; Positive and negative Twitter/X ratios and positive and negative news ratios. The stock list is made up of five tech stocks that are five of the seven magnificents. As for the scenario 1, the first results presented are the performance of each model for every stock and for every ahead value. Also, results are shown for the Amazon's stock to compare the final results of the three scenario for the same stock.

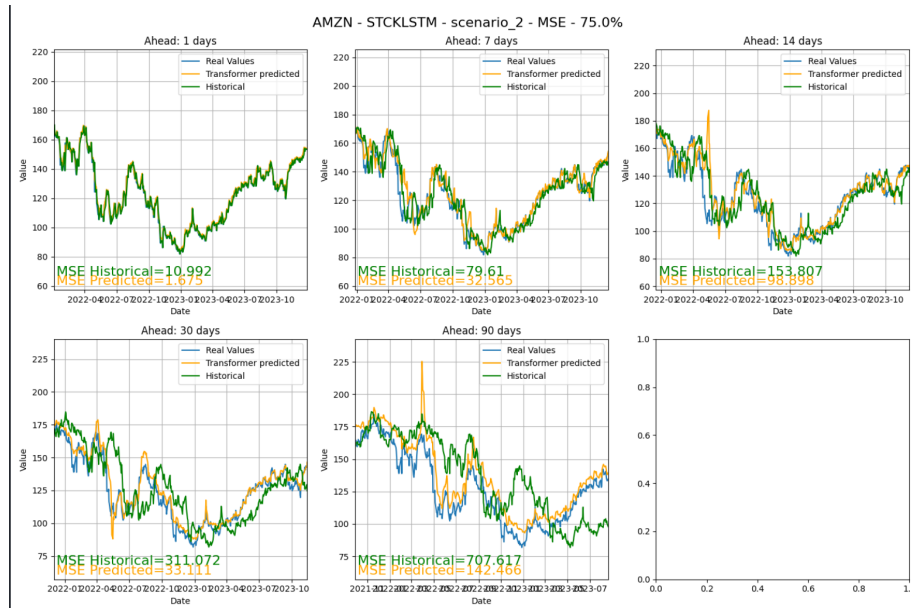


Figure 5.9: AMZN-STCKLSTM-MSE-75% [self elaboration].

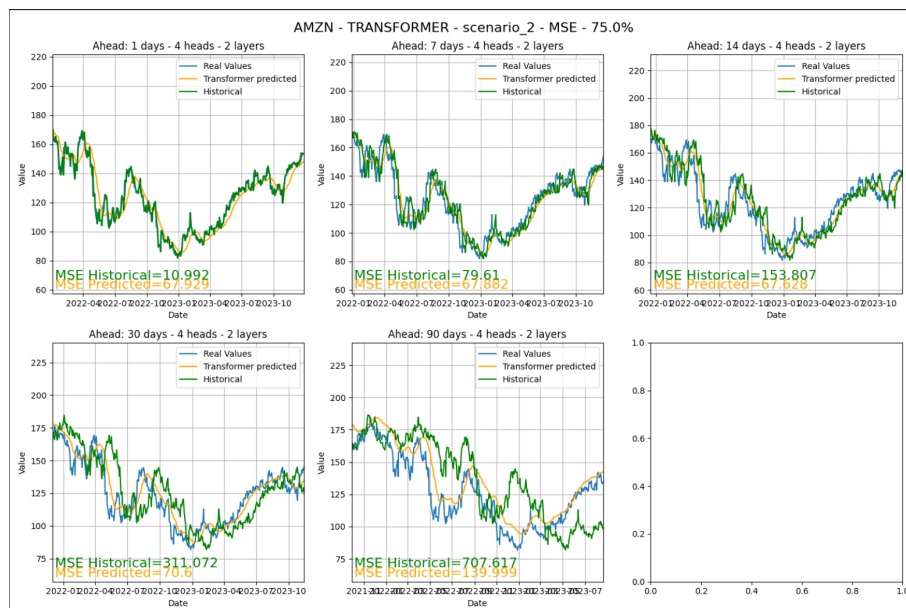


Figure 5.10: AMZN-Transformer-MSE-75% [self elaboration].

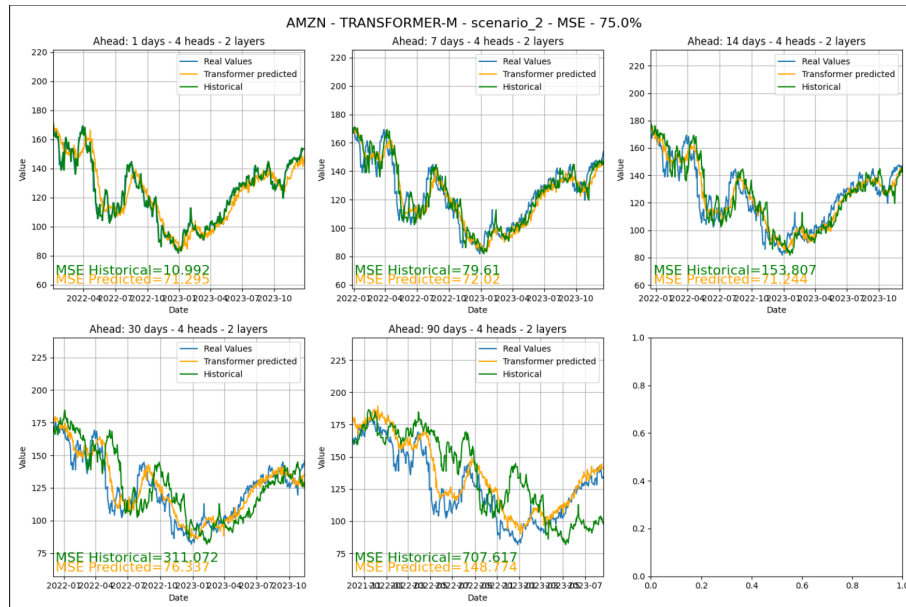


Figure 5.11: AMZN-Transformer-m-MSE-75% [self elaboration].

The above results are very similar to the results of scenario 1, where the lstm has a lower MSE for short-term prediction horizons, while the transformers perform better for medium- and long-term predictions. However, the stacked LSTM performs worse than the basic LSTM, this could be due to the difference in window size. There is another point interesting to highlight and is that the transformers MSE predicted lines are more smooth than the LSTM. The LSTM gets more atypical peaks in some ranges of the time period, while the transformers seem to not be able to learn about fast price movements too well.

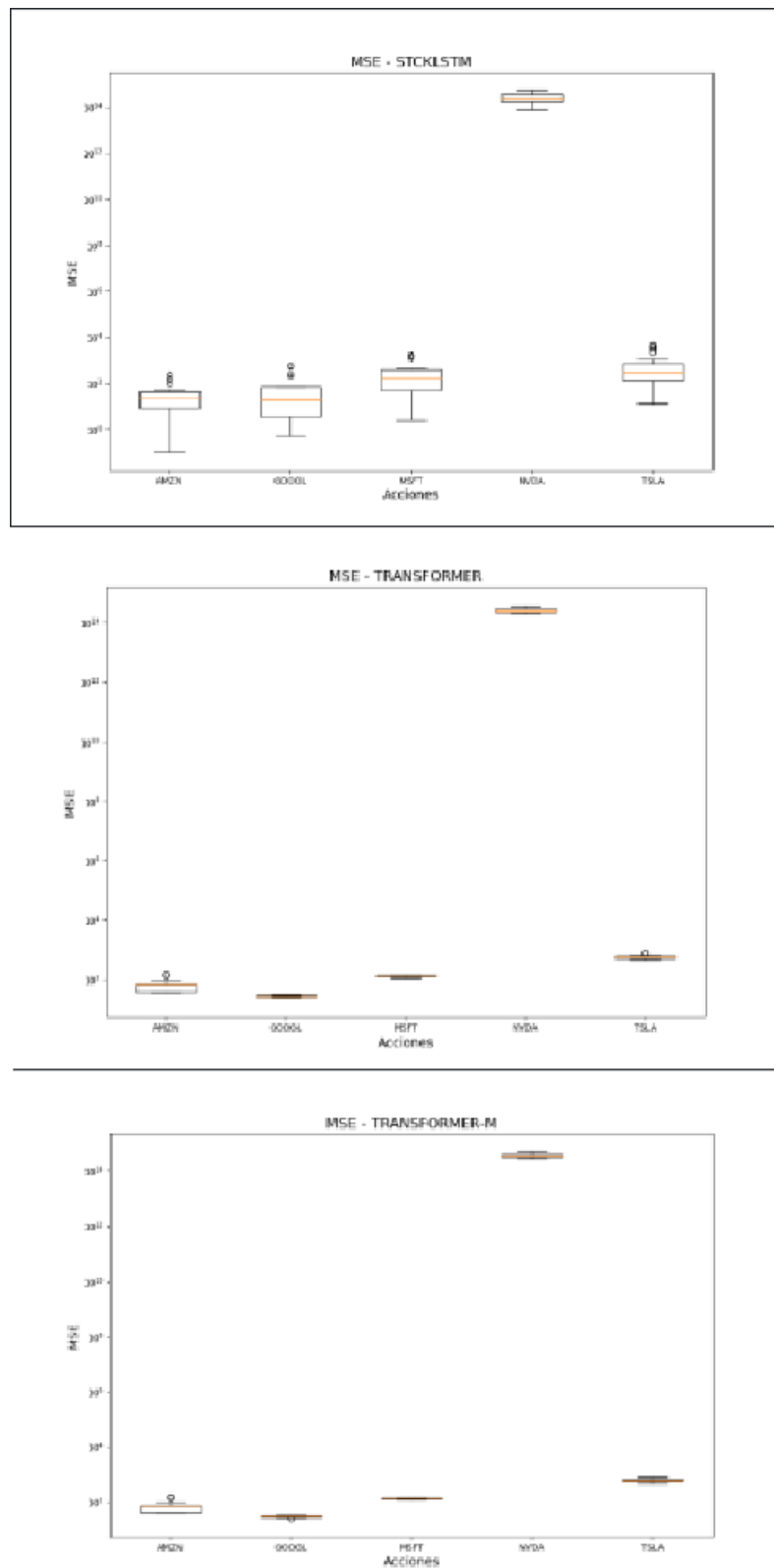


Figure 5.12: MSE stocks comparison scenario2 [self elaboration].

For this second scenario, the stocks with the best performances are Amazon, Google, Microsoft and Tesla. Nevertheless, Nvidia has a good boxplot with no outliers and a small box representing a low variance in its MSEs but it has values of MSE too big comparing to the others. This

can happen because of a huge volatility in the stock price. Looking at what stock is having this problem, Nvidia, it has sense to approve that idea. Nvidia's stock has experienced an exponential rise in recent years. On the other hand, for the multivariate transformer, Microsoft has the lowest variance in MSE making its boxplot almost like a straight line, but it has higher MSE than Google who's stock has the best performance for this scenario configuration.

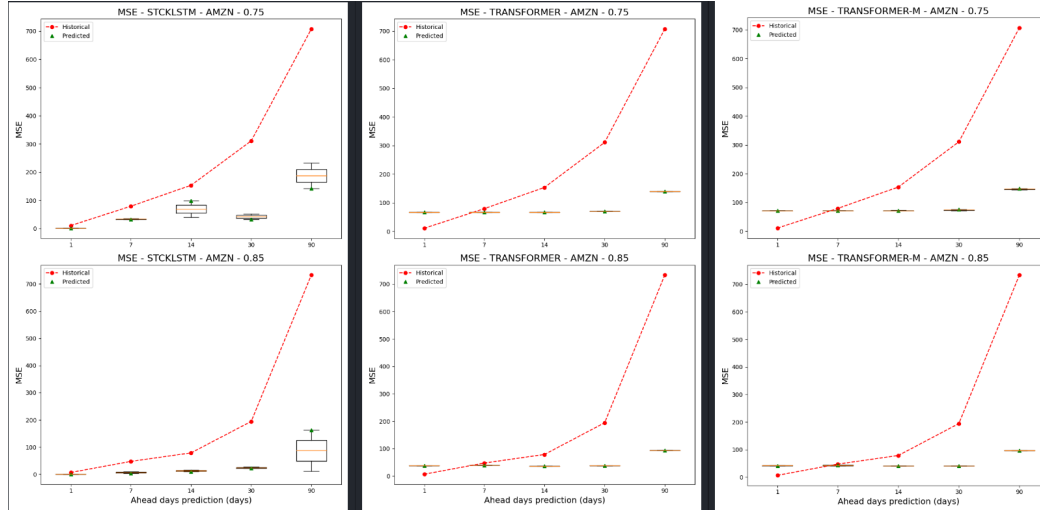


Figure 5.13: Train test split MSE comparison scenario 2 [self elaboration].

Regarding the analysis of the different percentages for the train test split represented in figure 5.13 Train test split MSE comparison scenario 2 [self elaboration]., the same pattern is repeated in this scenario as in scenario 1 and that is that, within these ranges, the higher the percentage, the better the results obtained by the model. This may be due to the fact that in order to learn well, especially over longer time horizons, a larger training set is necessary. It is worth noting that in the stacked LSTM it can be observed that, although the boxplot is placed lower on the vertical axis, an enlargement of the size of the boxplot can be seen, indicating a higher variance among the MSE values for 90 days.

It is also important to comment that the worsening of the results of scenario 2 with scenario 1 has been produced by the change of the window size. During the testing process, tests were made to understand how the performance of the predictions was as a function of the window size, and the result was the same as here, that the larger the window size, the worse the performance in terms of efficiency (the longer the model takes to train) and the worse results were obtained from the predictions.

### 5.3 Scenario 3 results

As for the other two scenarios, this scenario's configuration is shown in 5.3 Scenario 3 configuration [self elaboration]. The LSTM model used in it is the attention-based LSTM which is the LSTM model that theoretically could perform better. The features selected for the multivariate transformer are just three, close price, price trend and volatility. The stock list is made up of four stocks. As for the other scenarios, the first results presented are the performance of each model for every stock and for every ahead value. Also, results are shown for the Amazon's stock to compare the final results of the three scenario for the same stock.

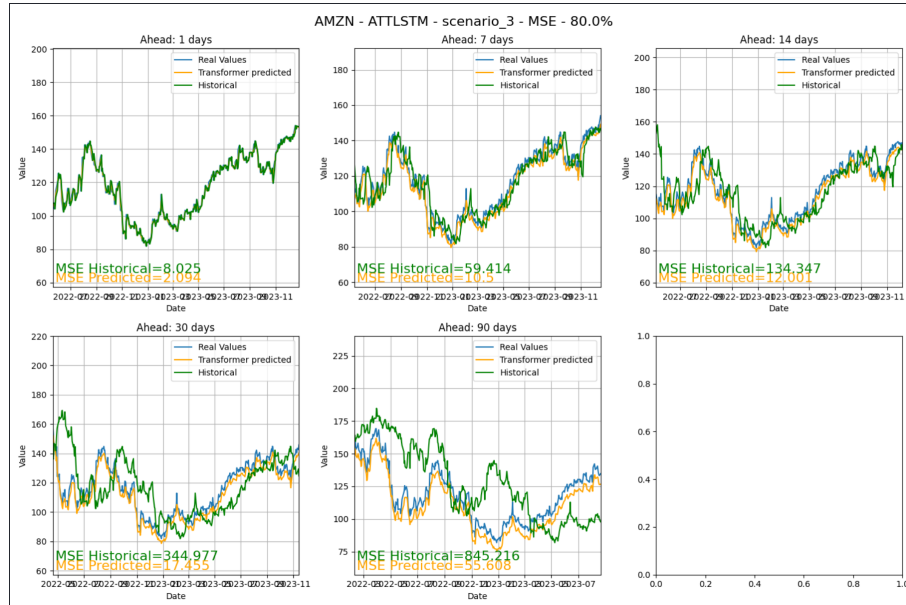


Figure 5.14: AMZN-AttLSTM-MSE-80% [self elaboration].

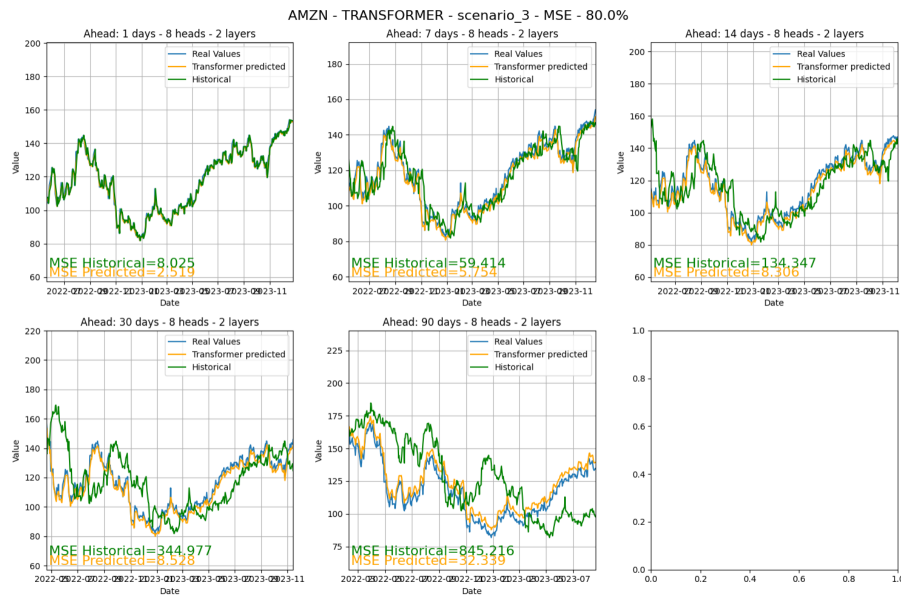


Figure 5.15: AMZN-Transformer-MSE-80% [self elaboration].

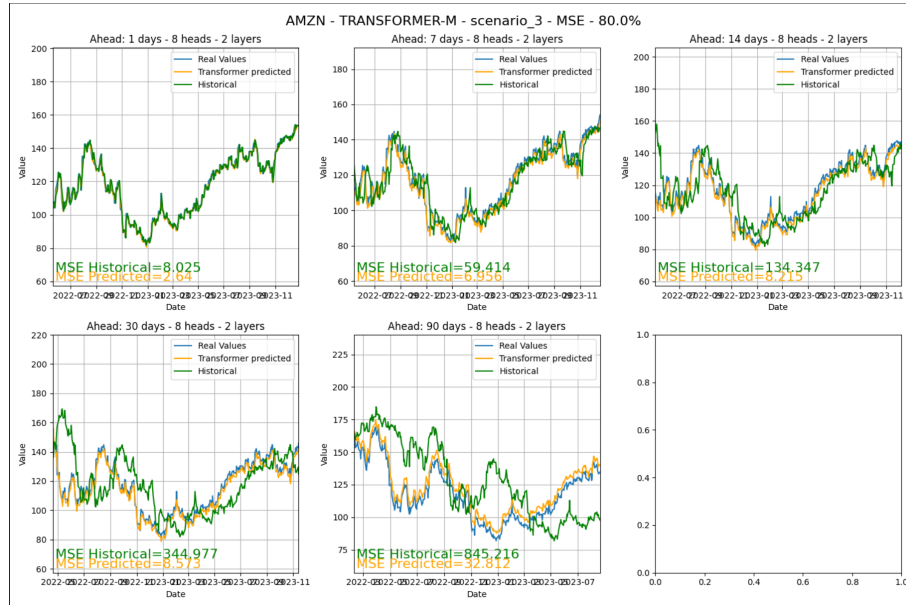


Figure 5.16: AMZN-Transformer-m-MSE-80% [self elaboration].

The results of this scenario are slightly different from the previous two scenarios. In this case, the attention-based LSTM fails to improve the transformer predictions for any time horizon other than 1 day. However, this attention-based LSTM performs better than the other two LSTM models. What can be concluded from the results of this scenario is that the parameter fit of both the window size and the transformer parameters is much better than that of the previous results, which is reflected in the comparison between different scenarios.



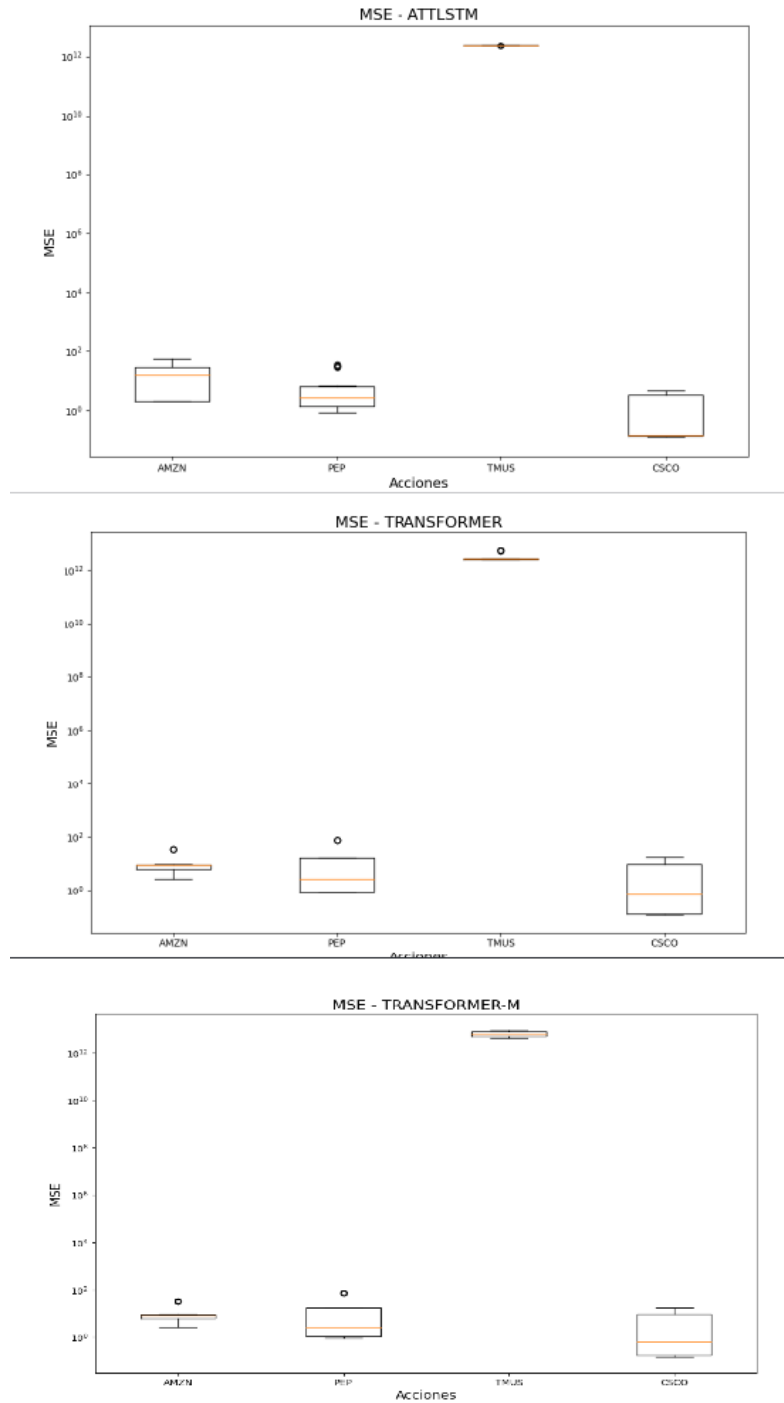


Figure 5.17: MSE stocks comparison scenario3 [self elaboration].

As was the case in previous scenarios, there are some stocks with much worse predictions than the rest and this may be due to the volatility of the stock price. As for the rest, no major improvements are observed between one model and the others, except in the Amazon stock. In the Amazon boxplot for the LSTM the box is much larger than in the Transformers, in the Transformers the box is practically a straight one although having a somewhat large lower whisker. What this indicates is that the variance of the MSE is much reduced in the transformers compared to the LSTM. In the rest of the actions, what happens is not a reduction of the size of the box but a movement of the line representing the mean, towards a more central point of

the box.

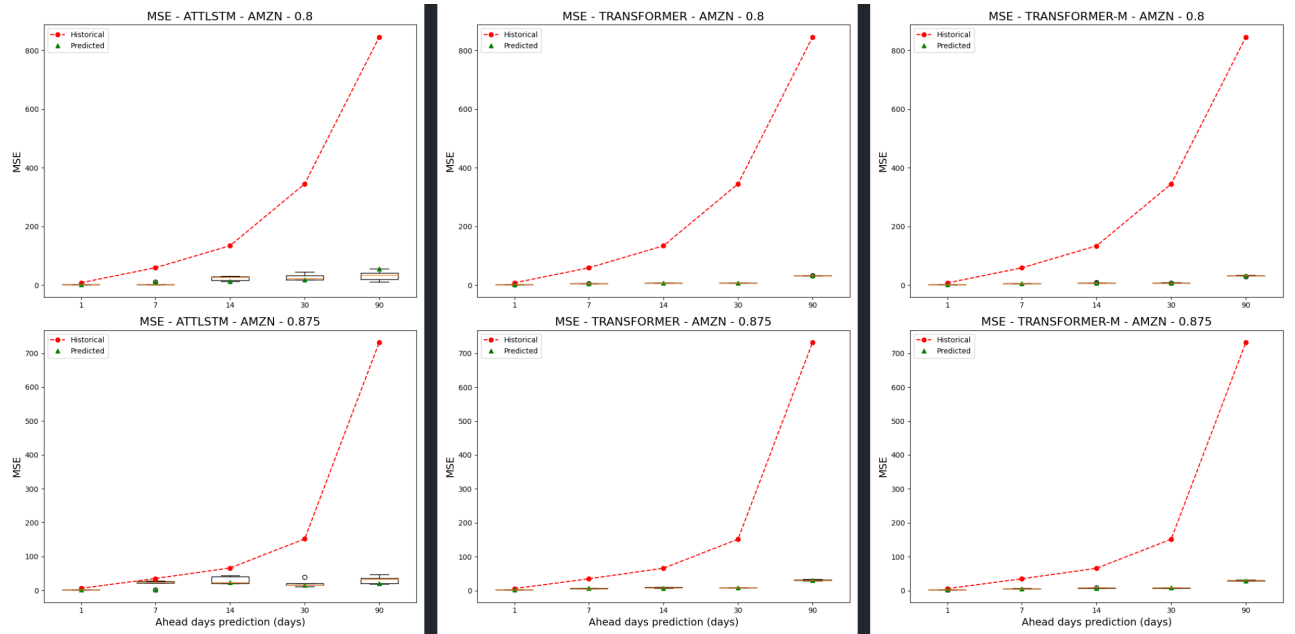


Figure 5.18: Train test split MSE comparison scenario 3 [self elaboration].

The results presented in the figure 5.18 Train test split MSE comparison scenario 3 [self elaboration]. follow the trend from the other scenarios. Here the larger the train data is the better predictions. However, it seems like 87.5% is near the limit for train data because it improves the predictions of the 80% but it is really close between. The conclusion is that the more data to train with, the better, but also that too much training data could end up worsening the results due to over-fitting.

#### 5.4 Transformers parameters results comparison

In the table 5.1 Comparison of the results for the univariate and multivariate transformers it is presented a comparison between the results of the univariate and multivariate transformer. The goal of this table is to get an idea of whether the multivariate data provides valuable information for the training of the model or not and to see if the different values for number of heads of attention and number of layers are related to better results. The results presented are for the 1 day and 90 day horizons because these are the extreme values and the best and worst results can be obtained.

Table 5.1: Comparison of the results for the univariate and multivariate transformers

Dimensionality	Train/test split	Heads/Layers	Days ahead	MSE
1 Dimension	70%	16 / 4	1	12.85
	70%	16 / 4	90	50.39
	75%	4 / 2	1	67.92
	75%	4 / 2	90	139.99
	80%	8 / 2	1	2.51
	80%	8 / 2	90	32.33
Multi Dimension	70%	16 / 4	1	14.63
	70%	16 / 4	90	53.35
	75%	4 / 2	1	71.29
	75%	4 / 2	90	148.77
	80%	8 / 2	1	2.64
	80%	8 / 2	90	32.81
1 Dimension	80%	16 / 4	1	11.55
	80%	16 / 4	90	51.06
	85%	4 / 2	1	38.25
	85%	4 / 2	90	94.0
	87.5%	8 / 2	1	2.52
	87.5%	8 / 2	90	30.57
Multi Dimension	80%	16 / 4	1	11.86
	80%	16 / 4	90	53.52
	85%	4 / 2	1	40.73
	85%	4 / 2	90	96.82
	87.5%	8 / 2	1	2.61
	87.5%	8 / 2	90	29.67

The conclusions extracted from the table 5.1 Comparison of the results for the univariate and multivariate transformers are the following:

- There is not a crucial difference between the unidimensional and the multidimensional transformer prediction. This may be because the real reason for the great performance of the transformer is the multi-head attention mechanism and adding more features no longer improves it.
- For the values of these simulations, the larger the train test split, the better the performance of the model. Comparing the same scenario pairwise (70%, 80%), (75%, 85%) and (80%, 87.5%) the higher the percentage, the lower MSE and therefore, better results.
- The results obtained in the simulations, shows that the best combination between number of multi attention heads and the number of layers is the 8 / 2. With this comparison, it can be extracted that is not necessary to use too much heads as well as is not optimal to use not enough of them. It has to be an analysis to determine the best number of heads for each solution.
- Looking to the results of the three types of LSTMs, there is a conclusion that can be extracted from the results section. The good performance in long-term predictions of a model that forecasts time series seems to be due to the attention mechanism and not to the multivariate data. The attention base LSTM and the multi head attention transformers, have obtained the best results for the bigger time horizons predictions.

### 5.5 Comparison of results with other studies

In this sub section, it will be discussed a comparison between the results of this project, with other related papers results. In this case, the main paper to compare the results is the Wenjie Lu et al. paper where it is proposed an architecture mixing a CNN with a LSTM for a short term price prediction [46]. In this paper, the forecasting goal is to predict the closing price of the stock for the next day, which is a time horizon studied in this PFG. The results obtained in the paper are as follows:

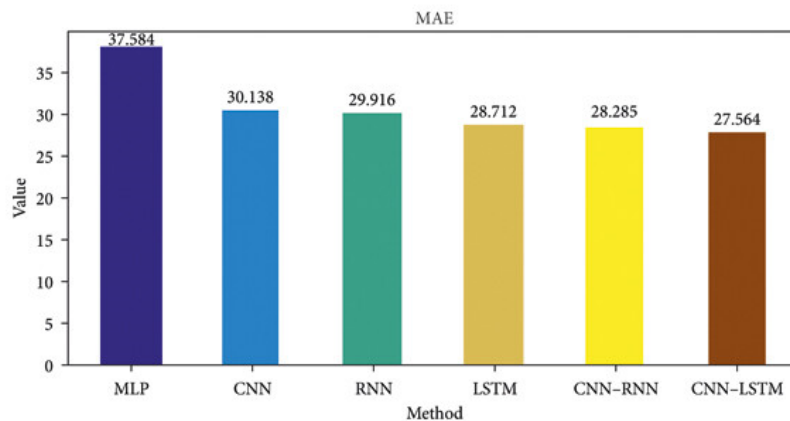


Figure 5.19: The result of MAE comparison among different methods [46].

As shown in the figure 5.19, the CNN-LSTM network improves the MAE for the 1 day ahead prediction compared to the LSTM. For the study of this PFG, the CNN-LSTM seems to perform better than the classic LSTM developed. It could be interesting to compare it with the stacked and the attention-based LSTM to see if it can improve their results too.

Other related study is the Ching-Ru Ko and Hsien-Tsung Chang LSTM-based sentiment analysis for stock price forecast [47]. This study, uses the BERT transformer for the sentiment analysis and develop a LSTM network to perform the price prediction. The main difference between Ko and Chang's project and this PFG is that in this PFG the price prediction model for the sentiment analysis data has been a multivariate transformer while for the other paper is a LSTM model that performs the price prediction. The results obtained in the Ching-Ru Ko et al. paper are as follows:

Method	Stock ID					
	1301	2317	2330	2610	2498	3406
LSTMP	0.641	1.222	3.463	0.073	0.493	6.97
LSTMN	0.616	1.181	3.355	0.067	0.467	6.874
LSTMF	0.62	1.158	3.402	0.063	0.424	6.464
LSTMNF	0.602	1.108	3.307	0.058	0.411	5.874

Figure 5.20: RMSE values from different models [47].

The methods shown in the figure 5.20 are these:

- LSTMP: Opening price, Closing price, Highest price, Lowest price and Volume.
- LSTMN: Opening price, Closing price, Highest price, Lowest price, Volume, *Ppos* of news and *Pneg* of news.
- LSTMF: Opening price, Closing price, Highest price, Lowest price, Volume, *Ppos* of PTT forum and *Pneg* of PTT forum.
- LSTMNF: Opening price, Closing price, Highest price, Lowest price, Volume, *Ppos* of PTT forum, *Pneg* of PTT forum, *P pos* of news and *P neg* of news.

It appears that when the method adds sentiment analysis to the price-related characteristics, the LSTM performance is better. The RMSE decreases when the method has sentiment analysis variables. On the other hand, the study conducted in this PFG did not obtain this improvement in results while there was no relative difference between univariate and multivariate data.

A conclusion that could be drawn from this comparison is that it is possible that the lack of accuracy of LSTMs in long-term price predictions can be improved by adding sentiment analysis, while transformers, being more robust for the long term, are unlikely to be improved by adding these variables.

## 5.6 Application to a trading strategy

With these analysis carried out, these results could be applied to a real algorithmic trading example. To do this, it is better to use the MAE (mean absolute error) metric since it gives a real representation of the error that the model has. Using the MAE, 1 MAE is equivalent to 1 dollar in the stock price. Knowing this, a small use case in algorithmic trading can be explained.

Imagining that you want to invest in Amazon in 90 days, and having the results of the system reflected in the figures 5.21 Amazon lstm MAE scenario 1 [self elaboration]. and 5.22 Amazon transformer multivariate MAE scenario 1 [self elaboration]. the multivariate transformer prediction can be used since it has a lower MAE. One of the strategies can be to use the MAE as a value for take profit. The take profit in trading is a limit order that the user places so that if the share price exceeds that limit and the user is long (buy), their position is automatically sold. Knowing this, if the user buys Amazon at 100\$ per share and wants to close the position near 135\$, user can conservatively put the take profit below the difference between the target price and the MAE. That is, if the system predicts that the price will reach 137\$, he user, knowing the MAE of the prediction, can set his take profit at  $137 - 6.43 = 124.14$  \$, in order to have a greater probability that the price touches the take profit level and close the trade with profits.

In conclusion, using the transformer multivariate instead of the LSTM, the take profit level can be more adjusted to the predicted stock price.

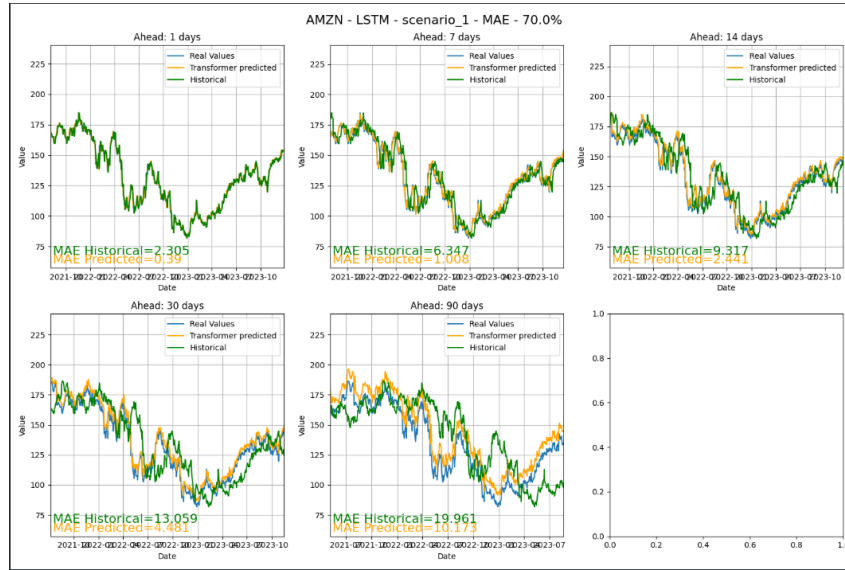


Figure 5.21: Amazon lstm MAE scenario 1 [self elaboration].

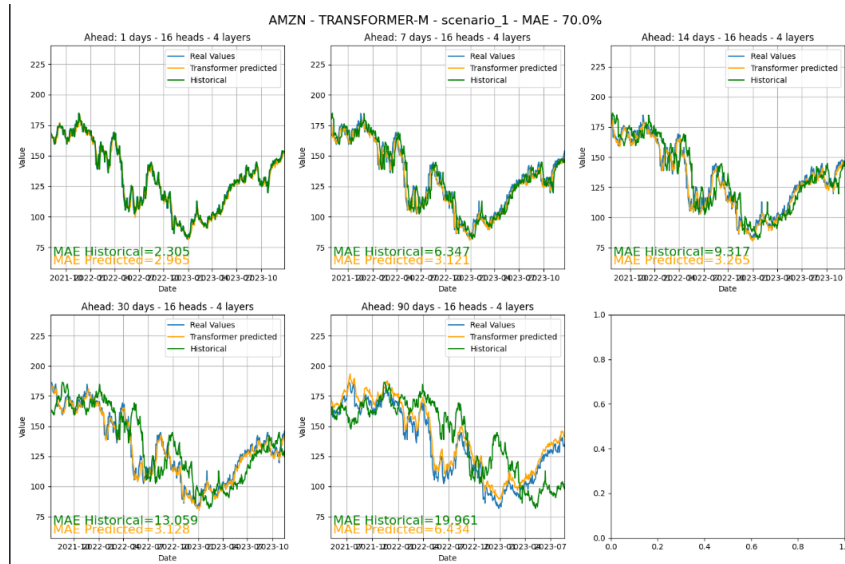


Figure 5.22: Amazon transformer multivariate MAE scenario 1 [self elaboration].

## 5.7 Application to portfolio management

Based on the analyses conducted, the results can be applied to a real-world portfolio management scenario. For this application, using the MSE (mean squared error) metric is advantageous as it penalizes larger errors more heavily, providing a robust measure of model accuracy.

Consider a portfolio manager aiming to optimize a diverse investment portfolio over a 90-day horizon. Suppose the manager is deciding between using LSTM or transformer models to predict future asset prices. Given that transformers generally yield a lower MSE for the 90 day-horizon, they offer more precise forecasts.

One practical strategy could involve adjusting the asset allocation based on the predicted returns. For instance, if the manager uses the predicted prices to determine the expected returns and

volatilities, they can use these predictions to optimize the portfolio according to the mean-variance optimization framework selecting the assets with less MSE in their predictions for a more secure investment, or mixing larger MSE with lower MSE to create a more risky strategy. Knowing that the transformer has a lower MSE, the manager can have more confidence in the predicted returns and thus make more informed allocation decisions.

In summary, leveraging the multivariate transformer model for portfolio management can lead to more accurate predictions of asset prices, thereby enhancing the ability to make data-driven investment decisions and optimize the portfolio for better returns.





## 6 Impacts of the project

This project impacts directly in the quantitative finance and financial engineering fields. It can also impact the time series researching field where this study contributes making a comparison among the technologies that are use to predict them.

### Social impact

This project was created with a specific emphasis on transparency, reproducibility, and knowledge accessible to all. It has been decided to make the project's GitHub repository public [45] in order to provide access to the source code, documentation, and resources related to the project. The ability to replicate results is a key component of both scientific study and software creation. By sharing the project's code on GitHub, it enables other researchers, students, and professionals to duplicate the results and enhance and extend the project. This project's goal is not only to address a particular issue in multivariate time series analysis but also to add to current knowledge and assist the scientific and technical community at large.

### Economic impact

Results obtain in this project can impact significantly in the investment industry because it has been detected which AI model is better for price predictions in different time periods. This can cause users to improve their results and lose less money.

### Technological impact

Thanks to the analysis of the different technologies studied in this project for the prediction of time series, the results and conclusions drawn can be used not only for the financial industry, but for any application of time series.

### Contribution to the Sustainable Development Goals (SDG)

This project makes significant contributions to several Sustainable Development Goals (SDGs) by leveraging advanced AI systems and innovative forecasting tools. By focusing on economic growth and technological advancements, the project not only aims to enhance financial performance but also to improve the working conditions of employees in the financial sector. Here are the SDGs aligned with this project:



Figure 6.1: SDGs aligned with the project [48].

SDG 8: Decent work and economic growth. This project directly supports SDG 8 by promoting economic growth and indirectly enhances the lives of workers in the financial sector. The primary objective is to compare AI systems for stock price prediction. By identifying the most effective models for short, medium, and long-term investments, the project can generate economic growth for both entities and individuals. Additionally, by streamlining investment decisions, employees will benefit from the system's assistance, improving their work experience and overall job satisfaction.

SDG 9: Industry, innovation and infrastructure. This project aims to compare the most innovative tools in the field of time series forecasting. By applying each model to real-world scenarios derived from this project, significant reductions in investment losses and even increased profits can be achieved. These savings or gains can then be reinvested in the pursuit of new innovative techniques, further advancing the financial and technological industries and saving valuable time for professionals working in these sectors.

## 7 Planning and budget

### 7.1 Planning

All tasks were performed sequentially, one after the other, with the exception of the documentation, which was done in parallel with the last tasks.

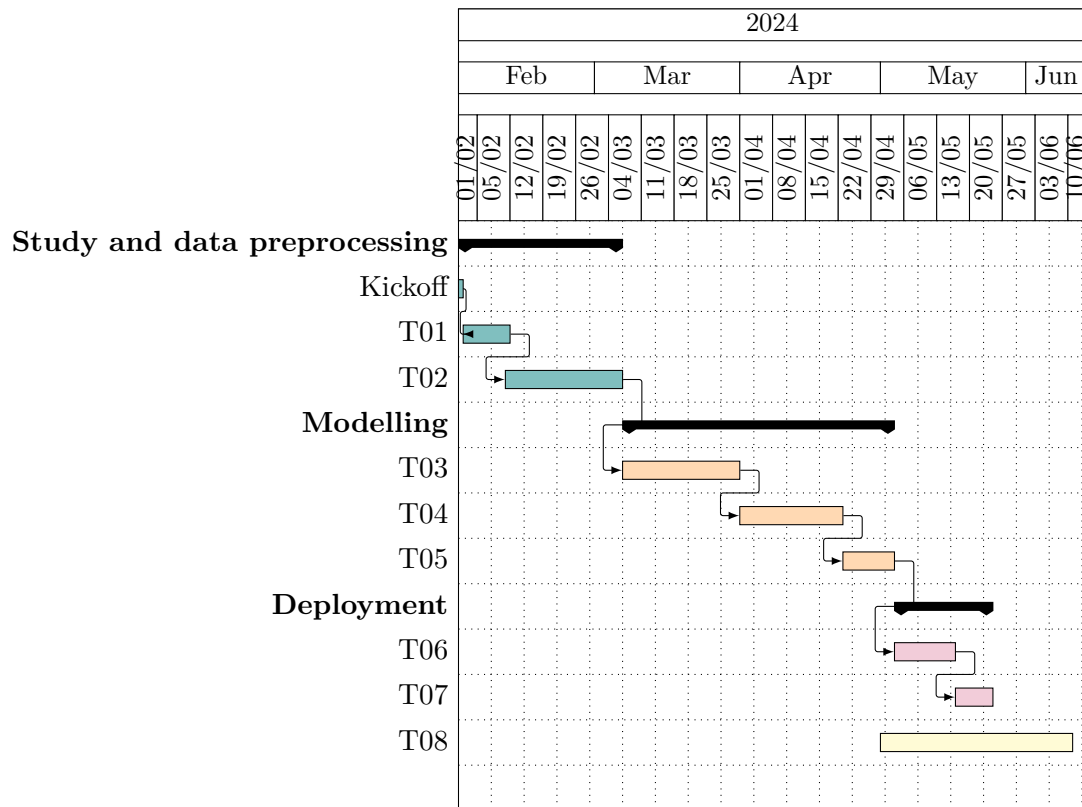


Figure 7.1: Project's Gantt chart [self elaboration].

Here is the list of tasks represented in the Project's Gantt chart [self elaboration].

1. T01.- Business understanding
2. T02.- To identify the available data and its pre-procesing
3. T03.- AI transformer creation and training
4. T04.- Transformers for different time periods and stocks
5. T05.- Validation Strategy
6. T06.- Multiheading alternatives
7. T07.- Design and create the system's execution flows
8. T08.- Documentation

## 7.2 Budget

In this section it is explained the budget of the project. The budget has been divided into materials and tools costs and the personnel budget.

Table 7.1: Materials and tools costs.

ITEM	PRICE
Windows 11 Home	145 €
IDE Visual Studio Code	0 €
Python packages listed in the <i>requirements.txt</i>	0 €
Computer AMD Ryzen 7 5800X 8-Core Processor 3.80 GHz, 32GB RAM and GPU MSI GeForce RTX 3060 (student's computer)	1.672,65 €
NVIDIA A30 de 24GB (provided by the department of Ingeniería de Organización, Administración de Empresas y Estadística of the ETSIST)	4.702,35 €
<b>TOTAL</b>	<b>6.520,00 €</b>

Table 7.2: Personnel costs.

ITEM	PRICE	HOURS	SUBTOTAL
Data engineer	23,5 €/h	355 h	8.342,5 €
<b>TOTAL</b>			<b>8.342,5 €</b>

Table 7.3: Total costs.

ITEM	PRICE
Direct costs	14.862,5 €/h €
General costs	1.084,53 €/h €
Benefits	583,97 €/h €
VAT (Value-Added Tax)	3.471,51 €
<b>TOTAL</b>	<b>20.002,51 €</b>

The total cost of the materials for the development of this project has been 6.520,00 €, this added to the personnel cost that is 8.342,5 € makes the direct costs that are 14.862,5 €. Then, the 13% of the direct cost is the general costs of the company and finally the 7% of the direct costs is the industrial benefits. To this sum it has to be added the Spain's VAT of 21%. The final budget of the project has been 20.002,51 €.

The data engineer salary has been taken from the mean data engineer salary in Spain, 37.500 €/year provided by the job portal Glassdoor [49].

## 8 Conclusions and future works

### 8.1 Conclusions

For the analysis between LSTM and transformers, it has been proven that each of the architectures has a good performance depending on what you are looking for. If you want to make price predictions over a shorter time horizon, then LSTMs are the best option. However, if what you are looking for is a medium and long term prediction (1 month and 3 months), the best option is to use transformers.

As for the comparison between the two types of transformers, univariate and multivariate, despite what was expected, adding more features to the data has not led to better results. The conclusion to be drawn is that the improvement in the results comes, rather, from the attention mechanisms. These mechanisms are responsible for the fact that in the long term, the model continues to make good predictions.

For the comparisons between LSTM models, as expected, the results were better depending on the complexity of the model. The basic LSTM obtained the worst results, while the attention-based LSTM obtained the best, also supported by the previous conclusion of the importance of attention.

A small convolutional transformer model has also been developed to test if it improved the results of the linear transformer but it did not. It has not been added to the project's system since it was a simple and not robust test. However, it may add to a line of future work the use of a robust convolutional transformer to compare the results with the rest of the system.

Moving on to the conclusions of the parameters, in spite of what was thought, the window size, the smaller the better. Tests were done for window sizes related to the most commonly used moving averages in trading 5, 10, 20, 50 and 200 periods, the larger windows not only required more training time, but it also gave worse results. It was therefore decided to simulate 3 scenarios with window sizes of 5, 20 and 2 to show that the larger the window size the worse.

Regarding the size of the training data split, it has been concluded that the higher the percentage of training data the better. This may be caused to the fact that a large amount of data is really needed for the model to learn the time series patterns, but it may also be due to another reason. This other reason could be that as the training and testing sets are ordered chronologically, i.e. if there are 10 years of data with 70% of training the first 7 years are for training and the last 3 are for testing, as these last years have experienced the covid crisis which is an extreme situation, it may be that even if the model learns the data well, it is not able to make good predictions because of the extreme covid situation.

In addition, for the number of heads and layers of the multi-head attention mechanism of the transformers, it has been concluded that the more heads and layers the better and that what needs to be done is a study to determine the optimal values for these variables.

Finally, in the analysis of the results for each stock, it has also been demonstrated that not all stocks are as predictable as the rest, and that for example in the case of Nvidia, which is a stock whose share price has a lot of volatility due to the enormous growth of the last years, it is very difficult to make good long-term predictions.

### 8.2 Future works

In this project a comparison has been made between the current technologies used for financial time series forecasting. It has been developed a uni dimensional and a multivariate multi-head attention transformer to compare them with the most used AI systems for this task, the LSTM. The results obtained showed that depending on how many days the prediction is made, one model or another works better. For this study, LSTM obtained better metrics for short term predictions while transformers work better for mid and long term predictions.

However, a few limitations were identified, such as lack of comparison with other models due to lack of development time, no updating of datasets and lack of integration with an automated trading system.

Based on the previous limitations and the results of this project, three future lines of work have been identified:

- Compare with more models: develop, train and evaluate more models to obtain a comparison within a wider range of models. For example a convolutional transformer, or other LSTM architectures like an ensemble of LSTM [50].
- Update datasets: Implement a system of periodic data updating to improve the accuracy and relevance of predictions.
- Integration with Broker API: Connect the developed system with a broker API to perform algorithmic trading in real time. This is the ultimate goal of any financial prediction AI system.

### 8.3 Personal learning

The completion of this project has provided important personal learning experiences, which focus primarily on the following three aspects:

**Overcome coding challenges:** One of the most valuable lessons learned during this project was how to effectively address coding problems. Finding bugs and errors, and sometimes spending up to two weeks searching for a solution, has been an invaluable part of the learning process. This experience has taught me perseverance, problem-solving skills, and the importance of a meticulous approach to debugging and testing code.

**Apply the knowledge of the degree:** This project offered the opportunity to apply and consolidate many concepts learned in different subjects of the degree program. From the theoretical foundations to the practical implementations, concepts learned in class have been applied to this project. This hands-on experience has deepened understanding of complex topics in data engineering and artificial intelligence. In one of the degree's deep learning course the professor talked about transformers and how they were one of the most innovative AI models but in his class he could not teach anything about them. With this project it was possible to learn how to develop one type of these transformers.

**Conducting research:** Another crucial learning outcome was the development of research skills. Engaging in extensive research, from reviewing the state of the art to analyzing and comparing different AI technologies, has led to improved ability to conduct rigorous research. This process has improved the skills of critical thinking, literature review, and systematic analysis, all of which are essential for any scientific investigation.

Overall, this project has been an important learning journey, providing practical coding skills, reinforcing academic knowledge and improving research capabilities.





## References

- [1] J. Kearns, “Las repercusiones de la inteligencia artificial en las finanzas,” *Finance & Development*, 2023.
- [2] GeeksforGeeks. “Deep learning: Introduction to long short term memory.” Accessed: 30/04/2024. (), [Online]. Available: <https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory>.
- [3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL].
- [4] J. Brownlee. “Stacked long short-term memory networks.” Accessed: 30/04/2024. (), [Online]. Available: <https://machinelearningmastery.com/stacked-long-short-term-memory-networks>.
- [5] D. Bahdanau and K. Cho, “Neural machine translation by jointly learning to align and translate,” *ICLR*, p. 15, 2015.
- [6] S. Analytics, “Ia y modelos de series temporales revolucionan la predicción de la demanda,” 2023. [Online]. Available: <https://www.sumoanalytics.ai/es/post/ia-y-modelos-de-series-temporales-revolucionan-la-prediccion-de-la-demanda>.
- [7] A. Turing, “Computing machinery and intelligence,” *Mind*, vol. 59, no. 236, pp. 433–460, 1950.
- [8] N. Nargund. “Comparing artificial intelligence, machine learning, and deep learning.” Accessed: 04/05/2024. (), [Online]. Available: <https://medium.com/@nisarg.nargund/comparing-artificial-intelligence-machine-learning-and-deep-learning-6da92404a8f6>.
- [9] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT Press, 2018.
- [10] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, *Language models are few-shot learners*, 2020. arXiv: 2005.14165 [cs.CL].
- [11] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
- [12] R. Merrit, *¿qué es un modelo transformer?* 2022. [Online]. Available: <https://la.blogs.nvidia.com/blog/que-es-un-modelo-transformer/>.
- [13] Wikipedia. “Python (programming language).” (), [Online]. Available: [https://en.wikipedia.org/wiki/Python\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Python_(programming_language)) (visited on 05/25/2024).

- [14] Nodd3r. “¿por qué se utiliza python en la ciencia de datos?” (), [Online]. Available: <https://nodd3r.com/blog/por-que-se-utiliza-python-en-la-ciencia-de-datos> (visited on 05/25/2024).
- [15] Tenworkflow. “Tensorflow.” (), [Online]. Available: <https://www.tensorflow.org> (visited on 05/25/2024).
- [16] DataDcientist.com. “Pytorch: All about facebook’s deep learning framework.” (), [Online]. Available: <https://datascientest.com/en/pytorch-all-about-this-framework#:~:text=Based%20on%20the%20former%20Torch,a%20simple%20and%20efficient%20way>. (visited on 05/25/2024).
- [17] *Learning a multiview weighted majority vote classifier: Using pac-bayesian theory and boosting*, [https://www.researchgate.net/figure/An-example-of-a-Supervised-Learning-classification-of-cats-and-dogs-and-b\\_fig1\\_328576527](https://www.researchgate.net/figure/An-example-of-a-Supervised-Learning-classification-of-cats-and-dogs-and-b_fig1_328576527), Image used from this link, last access: May 27, 2024, 2018.
- [18] *How spam detection taught us better tech support*, <https://cloud.google.com/blog/topics/developers-practitioners/how-spam-detection-taught-us-better-tech-support>, Image used from this link, last access: May 27, 2024, 2021.
- [19] *Customer segmentation using kmeans clustering*, <https://medium.com/@ashim.maity8/customer-segmentation-using-kmeans-clustering-cd8edc0a8e87>, Image used from this link, last access: May 27, 2024, 2020.
- [20] *Deep learning bible - d. un-supervised learning - eng.* <https://wikidocs.net/214112>, Image used from this link, last access: May 27, 2024, 2024.
- [21] *Guide to machine learning for anomaly detection: Introduction*, <https://medium.com/@gabrielpierobon/guide-to-machine-learning-for-anomaly-detection-introduction-e9d0f810dec5>, Image used from this link, last access: May 27, 2024, 2023.
- [22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
- [23] Google DeepMind. “Gemini - google deepmind.” Accessed: 04/05/2024. (), [Online]. Available: <https://deepmind.google/technologies/gemini/#introduction>.
- [24] OpenAI. “Gpt - openai.” Accessed: 04/05/2024. (), [Online]. Available: <https://openai.com/index/gpt-4>.
- [25] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2019. arXiv: 1810.04805 [cs.CL].
- [26] “Red neuronal artificial.” Accessed: 01/05/2024. (), [Online]. Available: [https://es.wikipedia.org/wiki/Red\\_neuronal\\_artificial](https://es.wikipedia.org/wiki/Red_neuronal_artificial).
- [27] “¿qué son las redes neuronales?” Accessed: 01/05/2024. (), [Online]. Available: <https://www.ibm.com/es-es/topics/neural-networks>.

- [28] F. Martínez Márquez. “Comparación entre las redes neuronales artificiales y las biológicas.” Accessed: 01/05/2024. (), [Online]. Available: <https://sujeto.es/comparacion-entre-las-redes-neuronales-artificiales-y-las-biologicas>.
- [29] “¿qué es una red neuronal?” Accessed: 01/05/2024. (), [Online]. Available: <https://aws.amazon.com/es/what-is/neural-network/>.
- [30] J. Schmidhuber and S. Hochreiter, *Long short-term memory*, 1997. [Online]. Available: <https://www.bioinf.jku.at/publications/older/2604.pdf>.
- [31] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: <http://www.deeplearningbook.org>.
- [33] F. Sanz. “Modelos de secuencia. todo lo que necesitas es atención.” Accessed: 01/05/2024. (), [Online]. Available: [https://www.themachinelearners.com/modelos-secuencia/#GRU\\_vs\\_LSTM](https://www.themachinelearners.com/modelos-secuencia/#GRU_vs_LSTM).
- [34] J. Brownlee, “A gentle introduction to long short-term memory networks by the experts,” *Machine Learning Mastery*, 2021. [Online]. Available: <https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/>.
- [35] N. Developer, “Long short-term memory (lstm),” *NVIDIA Developer*, [Online]. Available: <https://developer.nvidia.com/long-short-term-memory-lstm>.
- [36] A. Sugandhi, “A guide to long short term memory (lstm) networks,” *KnowledgeHut*, 2024. [Online]. Available: <https://www.knowledgehut.com/blog/data-science/lstm>.
- [37] J. Olano. “Confusion about q, k, and v matrices.” (), [Online]. Available: <https://community.deeplearning.ai/t/confusion-about-q-k-and-v-matrices/426146> (visited on 05/12/2024).
- [38] Amazon Web Services. “What is transformers in artificial intelligence?” Accessed: 01/05/2024. (Feb. 22, 2022), [Online]. Available: <https://aws.amazon.com/es/what-is/transformers-in-artificial-intelligence/>.
- [39] M. Kaya and M. Karsligil, “Stock price prediction using financial news articles,” Oct. 2010, pp. 478–482. DOI: 10.1109/ICIFE.2010.5609404.
- [40] B. Lim, S. O. Arik, N. Loeff, and T. Pfister, *Temporal fusion transformers for interpretable multi-horizon time series forecasting*, 2020. arXiv: 1912.09363 [stat.ML].
- [41] R. Moya. “Cpu vs gpu en deep learning: ¿cuánta diferencia hay en tiempo de entrenamiento?” (), [Online]. Available: <https://jarroba.com/cpu-vs-gpu-en-deep-learning-cuanta-diferencia-hay-en-tiempo-de-entrenamiento/> (visited on 05/08/2024).

- [42] M. Galgani. “Magnificent seven stocks: Market cap shake-up for nvidia, tesla continues.” (), [Online]. Available: <https://www.investors.com/research/magnificent-seven-stocks-latest-news-market-cap-weighting/> (visited on 05/08/2024).
- [43] C. F. Institute. “Volatility.” (), [Online]. Available: <https://corporatefinanceinstitute.com/resources/career-map/sell-side/capital-markets/volatility-vol/#:~:text=Volatility%20is%20a%20measure%20of,to%20predict%20their%20future%20movements.> (visited on 05/15/2024).
- [44] Wikipedia. “Gated recurrent unit.” (), [Online]. Available: [https://en.wikipedia.org/wiki/Gated\\_recurrent\\_unit](https://en.wikipedia.org/wiki/Gated_recurrent_unit) (visited on 05/18/2024).
- [45] V. Vallejo, *Ai in finance: Transformers applied to multivariate time series forecasting*, 2024. [Online]. Available: <https://github.com/jordieres/Finance-AI>.
- [46] W. Lu, J. Li, Y. Li, A. Sun, and J. Wang, “A cnn-lstm-based model to forecast stock prices,” *Complexity*, 2020. DOI: <https://doi.org/10.1155/2020/6622927>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1155/2020/6622927>. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1155/2020/6622927>.
- [47] C.-R. Ko and H.-T. Chang, “Lstm-based sentiment analysis for stock price forecast,” *PeerJ Computer Science*, 2021. DOI: <https://doi.org/10.7717/peerj-cs.408>. [Online]. Available: <https://peerj.com/articles/cs-408/>.
- [48] *Sdg poster and individual goals*, <https://www.un.org/sustainabledevelopment/news/communications-material/>, Image used from this link, last access: May 31, 2024.
- [49] Glassdoor, *Sueldos para el puesto de data engineer en españa*, 2024. [Online]. Available: [https://www.glassdoor.es/Sueldos/data-engineer-sueldo-SRCH\\_K00,13.htm#:~:text=El%20sueldo%20medio%20para%20el,2000%20%E2%82%AC%20y%204200%20%E2%82%AC..](https://www.glassdoor.es/Sueldos/data-engineer-sueldo-SRCH_K00,13.htm#:~:text=El%20sueldo%20medio%20para%20el,2000%20%E2%82%AC%20y%204200%20%E2%82%AC..)
- [50] Y. Lin, Y. Yan, J. Xu, Y. Liao, and F. Ma, “Forecasting stock index price using the ceemdan-lstm model,” *The North American Journal of Economics and Finance*, vol. 57, 2021, ISSN: 1062-9408. DOI: <https://doi.org/10.1016/j.najef.2021.101421>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1062940821000553>.

## Appendix

### Appendix 1: Stock Class

Code Stock Class used to create the object Stock. Each stock is an object with its ticker, file name and the rest of the parameters described in the scenario configuration.

```
1 class Stock:
2     def __init__(self, ticker: str, file_name: str, lahead: list, tr_tst:
      ↪ float, scen_name: str):
3         self.ticker = ticker
4         self._data = pd.DataFrame(self._read_data(file_name))
5         self._lahead = lahead
6         self.tr_tst = tr_tst
7         self.df = pd.DataFrame(self._read_data(file_name))
8         self.scen_name = scen_name
9         self.serial_dict = {}
10        self.mserial_dict = {}
11        self.serial_dict['INPUT_DATA'] = {}
12        self.mserial_dict['INPUT_DATA'] = {}
```

Code 8.1: Stock Class

## Appendix 2: LSTM Classes

```

1 class TrainLSTM:
2     def __init__(self, trainX: pd.DataFrame, trainY: pd.DataFrame,
3                 testX: pd.DataFrame, testY: pd.DataFrame,
4                 Y: pd.DataFrame, vdd: dict, epoch: int, bsize: int,
5                 nhn: int, win: int, n_ftrs: int, stock: str):
6         self.trainX = trainX
7         self.trainY = trainY
8         self.testX = testX
9         self.testY = testY
10        self.Y = Y
11        self.vdd = vdd
12        self.epoch = epoch
13        self.bsize = bsize
14        self.nhn = nhn
15        self.win = win
16        self.n_ftrs = n_ftrs
17        self.stock = stock

```

Code 8.2: LSTM Class

```

1 def lstm_fun(self, ahead: int, seed: int) -> dict:
2     nptrX = self.trainX.to_numpy().reshape(
3         self.trainX.shape[0], self.trainX.shape[1], self.n_ftrs)
4     nptrY = self.trainY.to_numpy()
5     nit = 0
6     lloss = np.nan
7     while math.isnan(lloss) and nit < 5:
8         tf.random.set_seed(seed)
9         model = Sequential()
10        model.add(LSTM(self.nhn, activation='relu', input_shape=(self.win,
11        ↪ self.n_ftrs)))
12        model.add(Dense(self.n_ftrs)) # Output of a single value
13        model.compile(loss='mean_squared_error', optimizer='adam')
14        checkpoint = ModelCheckpoint('model-basic.keras', verbose=1, \
15        ↪ monitor='val_loss', save_best_only=True, mode='auto')
16        hist = model.fit(nptrX, nptrY, epochs=self.epoch,
17        ↪ batch_size=self.bsize, verbose=0, callbacks=[checkpoint])
18        lloss = hist.history['loss'][-1]
19        nit = nit + 1
20    # Predict
21    nptstX = self.testX.to_numpy().reshape(
22        self.testX.shape[0], self.testX.shape[1], self.n_ftrs)
23    nptstY = self.testY.to_numpy()
24    res1 = eval_lstm(nptstX, nptstY, self.testX,
25        model, self.vdd, self.Y, ahead)
26    df_result = {'MSEP': res1.get("mse"), 'MSEY': res1.get("mse"),
27        'MAEP': res1.get("maep"), 'MAEY': res1.get("mae"),
28        'Stock': self.stock, 'DY': res1.get("Ys"), 'nhn': self.nhn,
29        'win': self.win, 'ndims': 1, 'lossh': lloss, 'nit': nit,
30        'model': model}
31    return(df_result)

```

Code 8.3: LSTM Basic function

```

1 def stck_lstm_fun(self, ahead: int, seed: int) -> dict:
2     nptrX = self.trainX.to_numpy().reshape(
3         self.trainX.shape[0], self.trainX.shape[1], self.n_ftrs)
4     nptrY = self.trainY.to_numpy()
5     nit = 0
6     lloss= np.nan
7     while math.isnan(lloss) and nit < 5:
8         tf.random.set_seed(seed)
9         stmodel = Sequential()
10        stmodel.add(LSTM(self.nhn, activation='relu', return_sequences=True,
11            input_shape=(self.win, self.n_ftrs)))
12        stmodel.add(LSTM(self.nhn, activation='relu',
13            ↪ return_sequences=True))
14        stmodel.add(LSTM(self.nhn, activation='relu'))
15        stmodel.add(Dense(self.n_ftrs)) # Output of a single value
16        stmodel.compile(loss='mean_squared_error', optimizer='adam')
17        checkpoint = ModelCheckpoint('model-stacked.keras', verbose=1, \
18            monitor='val_loss', save_best_only=True, mode='auto')
19        hist = stmodel.fit(nptrX, nptrY, epochs=self.epoch,
20            ↪ batch_size=self.bsize, verbose=0, callbacks=[checkpoint])
21        lloss= hist.history['loss'][-1]
22        nit = nit + 1
23    # Predict
24    nptstX = self.testX.to_numpy().reshape(
25        self.testX.shape[0], self.testX.shape[1], self.n_ftrs)
26    nptstY = self.testY.to_numpy()
27    res1 = eval_lstm(nptstX, nptstY, self.testX, stmodel, self.vdd,
28        ↪ self.Y, ahead)
29    df_result = {'MSEP': res1.get("mse"), 'MSEY': res1.get("mse"),
30        'MAEP': res1.get("maep"), 'MAEY': res1.get("maey"),
31        'Stock': self.stock, 'DY': res1.get("Ys"),
32        'ALG': 'STACK-LSTM', 'seed': seed, 'epochs': self.epoch,
33        'nhn': self.nhn, 'win': self.win, 'ndims': 1,
34        'loss': lloss, 'nit': nit, 'model': stmodel}
35    return(df_result)

```

Code 8.4: LSTM Stacked function

```

1  def att_lstm_fun(self, ahead, seed):
2      aptx = self.trainX.to_numpy().reshape(
3          self.trainX.shape[0], self.win, self.n_fts)
4      aptrY = self.trainY.to_numpy()
5      nit = 0
6      lloss = np.nan
7      while math.isnan(lloss) and nit < 5:
8          amodel = Sequential()
9          amodel.add(Embedding(input_dim=self.win, output_dim=int(self.win//3),
10                               mask_zero=True))
11          amodel.add(GRU(self.nhn, activation='relu', \
12                          return_sequences=True, input_shape=(self.win, self.n_fts)))
13          amodel.add(SeqSelfAttention(attention_activation='sigmoid'))
14          amodel.add(Dense(self.n_fts))
15          amodel.compile(loss='mean_squared_error', optimizer='adam')
16          earlyStopping = EarlyStopping(monitor='val_loss', \
17                                         patience=10, verbose=0, mode='min')
18          checkpoint = ModelCheckpoint('model-attention.h5', verbose=1, \
19                                       monitor='val_loss', save_best_only=True, mode='auto')
20          hist = amodel.fit(aptx, aptrY, epochs=self.epoch,
21                            ↪ callbacks=[checkpoint], \
22                               validation_split=0.15, batch_size=self.bsize, verbose=1)
23          lloss = hist.history['val_loss'][-1]
24          nit = nit + 1
25          # Predict
26          amodel = load_model('model-attention.h5', custom_objects={
27              'SeqSelfAttention': SeqSelfAttention})
28          aptstX = self.testX.to_numpy().reshape(
29              self.testX.shape[0], self.win, self.n_fts)
30          aptstY = self.testY.to_numpy()
31          res5b = eval_lstm(aptstX, aptstY, self.testX, amodel, self.vdd,
32                            ↪ self.Y, ahead)
33          #
34          df_result = {'MSEP': res5b.get("mse"),
35                       'MSEY': res5b.get("mse"), 'Stock': self.stock,
36                       'DY': res5b.get("Ys"), 'ALG': 'ATT_LSTM',
37                       'seed': seed, 'epochs': self.epoch, 'nhn': self.nhn,
38                       'win': self.win, 'ndims': 1, 'loss': lloss, 'nit': nit,
39                       'model': amodel}
40      return(df_result)

```

Code 8.5: LSTM Attention function



## Appendix 3: Transformers Classes

```

1 class TransformerL(nn.Module):
2     def __init__(self, input_dim: int, embed_dim: int, num_layers: int,
3         ↪ num_heads: int, dropout: float):
4         super(TransformerL, self).__init__()
5         self.input_dim = input_dim
6         self.embed_dim = embed_dim
7         self.num_layers = num_layers
8         self.num_heads = num_heads
9         self.dropout = dropout
10
11         self.embedding = nn.Linear(self.input_dim, self.embed_dim)
12         self.encoder_layers =
13             ↪ nn.ModuleList([nn.TransformerEncoderLayer(self.embed_dim,
14                 ↪ self.num_heads, dim_feedforward=4*self.embed_dim,
15                 ↪ dropout=self.dropout)
16                             for _ in range(self.num_layers)])
17         self.output_layer = nn.Linear(self.embed_dim, 1)

```

Code 8.6: Transformer Class

```

1 def forward(self, src) -> torch.Tensor:
2     src_embedded = self.embedding(src)
3     src_embedded = src_embedded.permute(1, 0, 2)
4
5     for encoder in self.encoder_layers:
6         src_embedded = encoder(src_embedded)
7
8     output = self.output_layer(src_embedded)
9     return output

```

Code 8.7: Forward method

## Appendix 4: User manual

The installation of the code to run the system and launch the simulations it is explained in the *README.md* file at the project's repository [45]. Here is the installation section:

### Installation

1. Clone the repository:

```
git clone https://github.com/jordieres/Finance-AI.git
```

```
cd Finance-AI
```

2. Install the required packages:

```
pip install -r requirement.txt
```

### Configuration

Customize the YAML configuration file to specify simulation parameters.

### Execution

Run the main script to start the simulation with the path of the configuration file and the operations to run: operations: 'pre;lstm;1DT;MDT;post'

```
python3 src/main.py -c path_to_config_file.yaml -o [operations]
```