MEDITERRANEA UNIVERSITY OF REGGIO CALABRIA
Department of Law, Economics and Human Sciences

PhD in Law and Economics

XXXVI cycle
Curriculum: Economics and Quantitative Methods

SECS-S/06; INF/01

Phd Thesis

IMPROVING THE PERFORMANCE OF BIDIRECTIONAL LSTM NEURAL NETWORKS THROUGH DATA PREPROCESSING, LOSS FUNCTION AND EVALUATION BEYOND THE TEST SET: Applications In Causal Impact Analysis

Supervisor:
Prof. Massimiliano Ferrara

PhD Student:
Dott. Pasquale Fotia

Co-Supervisor:
Prof. Ali Ahmadian

PhD Course in Law and Economics Coordinator
Prof. Attilio Gorassini

ACADEMIC YEAR 2022/2023

Improving the Performance of Bidirectional LSTM Neural Networks through Data Preprocessing, Loss Function and Evaluation Beyond the Test Set: Applications in Causal Impact Analysis

November 8, 2023

Index

Chapter 1

Introduction

In the current era marked by a reliance on data-driven decision-making, the practice of forecasting holds significant importance across all areas. The capacity to anticipate future events and trends, such as stock prices, consumer demand, or weather patterns, holds significant significance for individuals and organizations alike. The persistent endeavor to enhance the accuracy and dependability of forecasting has driven an unwavering search for novel techniques and sophisticated instruments. In the context of the ever-changing field of forecasting, this thesis aims to contribute to the advancement of both the theoretical and practical aspects of prediction. The field of forecasting encompasses a wide range of topics and complexities. This study specifically focuses on financial time series data, which offers a complex and enlightening context for the evaluation and enhancement of innovative forecasting techniques. Financial markets, characterized by their complex interactions including economic indicators, investor mood, and geopolitical events, function as a testing ground where the effectiveness of novel approaches is thoroughly evaluated. Nevertheless, the knowledge acquired from this particular financial context has broader implications, as it may be applied to several forecasting areas.

1.1 Problem Description

The core focus of this study revolves around the primary objective of enhancing the accuracy of predicting. Irrespective of the field of study, the primary objective is to create forecasting models that generate forecasts that closely correspond to real-world results. This endeavor entails the examination and resolution of certain fundamental concerns:

1. The primary goal is to improve the accuracy of forecasting models, so ensuring that the predictions provide decision-makers with actionable information.
2. The study investigates and assesses innovative forecasting methodologies and tools in order to enhance accuracy. These methodologies are carefully crafted to address the complex and distinct attributes of time series data.

3. The objective of this research is to evaluate the applicability of these innovative forecasting methods in many fields, extending beyond the realm of financial markets. This analysis highlights the adaptability and resilience of the methodologies being investigated.

1.2 Overview of the Thesis Structure

In order to methodically tackle these obstacles and achieve the stated objectives, the thesis is structured into the following chapters:

- **Chapter 2, Theoretical Framework and Review of Relevant Literature**: This section entails a comprehensive examination of the current body of forecasting literature and the most advanced methodologies currently available. This chapter establishes a robust basis and outlines conducive conditions for fostering creativity.
- **Chapter 3, Methodology and Tools**: This study provides an in-depth analysis of the methodology, libraries, and tools utilized in the research. The process involves the development and evaluation of different forecasting methodologies, providing insight into the reasoning behind their choice.
- **Chapter 4, Experimental Findings and Evidence**: This chapter serves as the focal point of the research, providing a comprehensive and thorough exposition of the outcomes obtained through experimental procedures. The aforementioned entity functions as a crucible in which the evaluation of various forecasting methodologies' impact on accuracy takes place, alongside the assessment of their efficacy over a wide range of disciplines.
- **Chapter 5, Applications in Causal Impact Analysis**: This chapter introduces the practical application of research methodologies inspired by the work of Fotia and Ferrara, 2023 on causal impact analysis of media events on financial markets. It employs a systematic approach integrating Bayesian Structural Time-Series and Bidirectional LSTM models to understand causal relationships between events and financial markets. The primary aim is to unveil how media events impact stock prices, trading patterns, and market volatility, providing valuable insights for financial decision-makers.
- **Chapter 6, Conclusions**: The thesis culminates in a thorough synthesis of primary discoveries and their significant ramifications. The present study engages in a reflective analysis of the research path, duly recognizing the significant contributions that have been made to the domain of forecasting.

By employing a systematic methodology, this thesis aims to make a substantial contribution to the progress of forecasting approaches. The practical significance of the research is highlighted by the empirical analysis of financial data, indicating that the generated models have broader applicability beyond the financial sector.

Chapter 2

Theoretical Foundations and Literature Review

2.1 Recurrent Neural Networks (RNNs)

Feedforward neural networks have been extensively utilized and have demonstrated significant achievements across various domains. In networks of this nature, the transmission of data flow transformations occurs through concealed layers in a unidirectional manner, wherein the output is solely influenced by the present circumstances. However, it should be noted that these neural networks exhibit limited memory capacity and are not well-suited for effectively modeling data sequencing and temporal dependencies within historical data. In order to overcome this constraint, researchers have developed recurrent neural networks (RNNs) to address learning tasks that involve time dependencies (Hochreiter and Schmidhuber, 1997). The fundamental principle underlying recurrent neural networks (RNNs) is to incorporate the impact of previous information in order to generate the output. In order to achieve this objective, the output is generated by incorporating cells that are influenced by gates based on historical observations. Undoubtedly, a segment of a neural network, examines a given input x_t and generates an output value h_t. Recurrent Neural Networks (RNNs) have been found to be effective in acquiring temporal information (Oksuz et al., 2019).There exist two robust recurrent neural network (RNN) models, namely Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), which have demonstrated effectiveness in handling time-dependent patterns within time-series data. Deep learning models have exhibited significant efficacy in the realm of modeling and forecasting, surpassing the performance of classical time series models. Furthermore, traditional networks have proven their ability to achieve favorable outcomes across various application domains involving time series data (Ashour et al., 2020; Harrou et al., 2020).

2.2 Long Short-Term Memory (LSTM)

The Long Short-Term Memory (LSTM) is an advanced gated memory unit that has been specifically developed to address the issue of vanishing gradients, which can significantly hinder the effectiveness of a basic Recurrent Neural Network (RNN) (Hochreiter and Schmidhuber, 1997). In the context of the significant time step, it is observed that the gradient either becomes excessively small or large, leading to the occurrence of the vanishing gradient problem. This issue arises during the training process, wherein the optimizer performs backpropagation and executes the procedure, while the weights exhibit minimal changes.In essence, the Long Short-Term Memory (LSTM) model is equipped with three gates that regulate the flow of information. These gates are referred to as the input gate, forget gate, and output gate. In essence, the formation of these gates is achieved through the utilization of logistic functions that are based on weighted sums. The weights necessary for this process can be acquired through training via the backpropagation technique. The management of the cell state is accomplished through the utilization of the input gate and the forget gate. The output is derived from either the output gate or the hidden state, which serves as the memory allocated for utilization. This mechanism enables the network to retain information over an extended period, a capability that is absent in traditional single recurrent neural networks (RNNs). Indeed, the desirable attributes of Long Short-Term Memory (LSTM) models encompass their enhanced capability to capture long-term dependencies and their adeptness in handling time-series data.

- **Input Gate**: $I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i)$
- **Forget Gate**: $F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f)$
- **Output Gate**: $O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o)$
- **Intermediate Cell State**: $\tilde{C}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c)$
- **Cell State (next memory input)**: $C_t = F_t \odot C_{t-1} \odot \tilde{C}_t$
- **New State**: $H_t = O_t \odot \tanh(C_t)$

Where W_{xi}, W_{xf}, W_{xo} and W_{hc}, W_{hf}, W_{ho} refer respectively to the weight parameters and b_i, b_f, b_o denote bias parameters. W_{xc} and W_{hc} denote weight parameters, b_c is the bias parameter, $\odot$ denotes element-wise multiplication. The estimation of C_t depends on the output information from memory cells (C_{t-1}) and the current time step $\tilde{C}_t$.

2.3 Bidirectional LSTM (BLSTM)

The bidirectional Long Short-Term Memory (BiLSTM) model represents an improved iteration of the LSTM algorithm. As previously mentioned, within the Long Short-Term Memory (LSTM) framework, the reconstruction of the current state is solely dependent on the backward context. Nevertheless, the LSTM model fails to take into account the relationship between the forward context and the current state. In order to address this limitation and enhance the precision of state

reconstruction, researchers have proposed the bidirectional Long Short-Term Memory (BiLSTM) algorithm. This algorithm combines the advantageous characteristics of the bidirectional Recurrent Neural Network (RNN) (Schuster and Paliwal, 1997) with those of the Long Short-Term Memory (LSTM) model (Graves and Schmidhuber, 2005). This has been achieved through the integration of two concealed states, enabling the retrieval of information from both the backward and forward layers. The BiLSTM neural network architecture is particularly advantageous in scenarios that necessitate the incorporation of contextual information as input. The utilization of this technique has been extensively employed in various domains, particularly in the field of classification, such as text classification (Liu et al., 2020), sentiment classification (Sharfuddin et al., 2018), and speech classification and recognition (Graves et al., 2013). Furthermore, Bi-directional Long Short-Term Memory (Bi-LSTM) models have been employed in the field of PM2.5 concentration prediction (Zhang et al., 2020) as well as load forecasting (Wang et al., 2019).

2.4 GRU models

The GRU model (Cho et al., 2014), serves as an alternative version of LSTM. Its purpose is to enhance the performance of LSTM while simultaneously reducing the number of parameters and simplifying its design. In the GRU model, the LSTM's input gate and forget gate have been combined into a single gate known as the update gate. In GRU, there are two gates, namely the update gate and the reset gate, as opposed to the three gates present in LSTM. The introduction of the reset and update gates concepts represents a significant advancement in GRU models, offering novel advantages. This approach introduces a novel evaluation technique that enables the computation of latent variables in recurrent neural network (RNN) architectures. The Long Short-Term Memory (LSTM) structure was improved by the Gated Recurrent Unit (GRU) by incorporating the update gate to couple the input and forget gates of the LSTM, and utilizing the output gate as a reset gate. The update gate is responsible for determining the amount of previously retained memory, while the reset gate ensures the manner in which the current inputs are combined with the existing memory.

The mathematical relationships between the various GRU components are given by:

- **Update gate**: $Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$
- **Reset gate**: $R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r)$
- **Cell state**: $H_t = \tanh(X_t W_{xh} + (R_t \odot H_{t-1}) W_{hh} + b_h)$
- **New state**: $H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot H_t$

where: W_{xr}, W_{xz}, and W_{hr} are weight parameters, and b_r, b_z are bias parameters. W_{xh}, W_{hh} are weight parameters, and b_h is a bias parameter. For a given time step t, the current update gate Z_t is used to combine the previous hidden state H_{t-1} and the current candidate hidden state $\tilde{H}_t$.

2.5 Data Preprocessing in Neural Networks

In order to facilitate the learning process of the network, it is advisable for the data to be scaled to small values, preferably within the range of 0 to 1. Additionally, it is crucial for the data to exhibit homogeneity, indicating that all the features should possess values within a similar range. Consequently, the data was transformed to a normalized scale ranging from 0 to 1. The procedure of standardizing values to a common range is referred to as MinMax normalization. To accomplish this task, the MinMaxScaler module from the Scikit-learn library is imported. The equation representing the Min-Max normalization method can be expressed as follows:

$$z = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{1}$$

Where z represents the normalized value and x represents the observed values in the set. Min and max are the minima and maxima values in x (Yamak et al., 2019).

2.6 Evaluation metrics

Within the domain of forecasting models, it is customary to utilize distinct assessment criteria in order to evaluate their efficacy. Tsai et al., 2018, conducted a significant study whereby they assessed the precision of their forecasting model by utilizing data obtained from 77 air quality monitoring stations located in Taiwan. The data spanned a time frame from 2012 to 2017. The dataset encompassed several gaseous components and PM2.5 levels, in addition to localized climatic information. The researchers employed a training methodology that involved the utilization of an LSTM neural network model. The model was trained using a dataset spanning the years 2012 to 2016, while the data from 2017 was reserved only for the purpose of testing. In order to evaluate the precision of their forecasting model,Tsai et al., 2018, utilized two widely employed assessment metrics, namely Root Mean Square Error (RMSE) and Mean Absolute Error (MAE). The root mean square error (RMSE) is a metric used to measure the level of variability and accuracy in the dataset. Models with lower root mean square error (RMSE) values are associated with better levels of accuracy, suggesting that the model well represents the experimental data. The field of Model-based Analysis of mistakes (MAE) offers valuable insights into the accurate assessment and understanding of prediction mistakes.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2} \tag{2}$$

$$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n}|y_t - \hat{y}_t| \tag{3}$$

2.7 Training and evaluation

Once a neural network model has been trained using the designated training set, the subsequent pivotal task involves evaluating its performance on data that has not been previously encountered. This evaluation serves to gauge the model's capacity for generalization, which refers to its ability to make accurate predictions on new, unseen data beyond the training set. The assessment is conducted during the testing phase using a distinct dataset referred to as the test set. The test set comprises instances that have not been previously encountered by the model, and it functions as a standard against which the model's capacity to generate precise predictions on novel and unfamiliar sequences can be evaluated. During the evaluation phase, the neural network model is exposed to the sequences from the test set, and it produces forecasts for the subsequent data points in the sequence. These predictions are then compared with the actual values, also known as the ground truth, for each sequence. Subsequently, performance metrics such as mean squared error or mean absolute error (Tsai et al., 2018) are computed to evaluate the degree of concordance between the model's predictions and the observed values. To achieve success, a neural network model must demonstrate strong generalization capabilities when applied to the test set. This entails producing accurate predictions and minimizing the discrepancy between its predictions and the actual values. The prevention of overfitting is of utmost importance during the training phase to ensure that the model's performance is not limited to the training data alone but extends to novel data as well. The testing phase plays a critical role in assessing the model's performance on data that it has not been previously exposed to. This evaluation is essential for making well-informed decisions regarding the deployment of the model in practical, real-world scenarios. The test results provide valuable insights into the strengths and weaknesses of the model, thereby offering guidance for potential enhancements and refinements to improve its performance. In the context of model development, the testing phase assumes a pivotal role as it holds substantial importance in assessing the practical applicability and efficacy of the model in real-life situations. These phases are described by Halpern-Wight et al., 2020.

Chapter 3

Methodology and Tools

Within the framework of this study, the chapter on methodology plays a pivotal role as an essential foundation for doing a comprehensive and methodical research endeavor. This chapter serves as a complete guide for the design and execution of all stages of the study, including a thorough review of the procedures and tools employed to accomplish the research objectives. The methodology and instruments chosen for this study have been carefully selected based on a thorough evaluation of its specific requirements, as well as an examination of established practices within the relevant field of research. Every decision made regarding the methodology has been made with the intention of guaranteeing the dependability and accuracy of the acquired outcomes. A significant portion of this chapter is devoted to elucidating the libraries and instruments employed in the research endeavor. Various libraries such as `yfinance`, `Matplotlib`, `NumPy`, `Pandas`, `TensorFlow`, and others were utilized to facilitate data collecting, processing, the creation of machine learning models, and subsequent result analysis. The part pertaining to the dataset offers a thorough examination of the many sources of data, including their distinctive attributes, as well as the rationale employed for their selection. The inclusion of this stage is crucial in comprehending the source and characteristics of the data employed in the investigation. Considerable emphasis has been placed on the data preprocessing stage, encompassing the establishment of a foundational Bi-LSTM model, normalization of data, angle-based transformation, and interpolation of intermediary values. These procedures are essential in ensuring that the data is sufficiently prepared for analysis and modeling. The precise characterization of a bespoke loss function played a vital role in evaluating the efficacy of machine learning models. The "Custom Loss Function" was specifically formulated to evaluate the degree of resemblance between the predictions generated by a model and the observed data within the domain of time series analysis. The financial arena has recently witnessed the introduction of an innovative approach known as "Iterative Forecasting." This approach entails the iterative training of LSTM models. The proposed approach, in conjunction with a genetic algorithm, seeks to optimize the parameters of the model in order to enhance the accuracy of predictions. In general, the chapter on methodology establishes a strong foundation for undertaking rigorous and analytical research, presenting a comprehensive review of the approaches and tools employed in this study.

3.1 Libraries and Tools

Numerous libraries and software tools were important in the research and execution of the bi-LSTM models outlined in this PhD thesis. This section presents a comprehensive review of the main libraries employed and their distinct contributions to the analytical and financial forecasting procedures.

3.1.1 yfinance

The `yfinance` library played a crucial role in obtaining the financial data that was required for the analysis. The platform provides a user-friendly and easily navigable interface that facilitates the retrieval of historical and real-time data from various sources, including Yahoo Finance. The utilization of the `yfinance` library enabled the acquisition of dependable and current data for the purpose of conducting the analysis.

3.1.2 Matplotlib

The `Matplotlib` package is extensively utilized for the generation of charts and visualizations. The tool was utilized to create visual depictions of historical financial data, develop bi-LSTM forecasts, and generate other relevant visualizations for the research. The flexibility and diversity of the tool enabled the adaptation of visualizations to align with the unique demands of the research.

3.1.3 NumPy and Pandas

The libraries `NumPy` and `Pandas` are essential tools for the manipulation and analysis of data. NumPy gives fundamental support for numerical computations and data structures, whereas Pandas provides robust capabilities for data manipulation, cleansing, and transformation. The utilization of these libraries played a pivotal role in the preparation and preprocessing of the financial data, ensuring its suitability for input into the bi-LSTM models.

3.1.4 TensorFlow

The `TensorFlow` library, more especially its `Keras` application programming interface (API), was employed in the construction and training of the bi-directional Long Short-Term Memory (bi-LSTM) models. TensorFlow offers a comprehensive framework for the development and training of deep learning models, whereas Keras streamlines the process of constructing these models. The implementation of the bi-LSTM architecture was carried out utilizing the high-level abstractions provided by the `Keras` framework, which facilitated the efficient development and experimentation of the model.

3.1.5 Random, Math, and OS Libraries

The Python libraries `random`, `math`, and `os` were employed for several purposes throughout the study. The `random` module enabled the incorporation of randomness in the processes of data sampling and model training. The `math` library was utilized for performing mathematical computations and transformations, whilst the `os` library facilitated the management of file operations and directory structures.

3.1.6 Scikit-Learn

The `scikit-learn` library, especially its `preprocessing` module including the `MinMaxScaler`, was instrumental in implementing various machine learning algorithms and preprocessing techniques. It provided a wide range of tools for data preprocessing, feature scaling, and model evaluation. The integration of `scikit-learn`'s `MinMaxScaler` complemented the deep learning approaches and enriched the analytical capabilities, allowing for a more comprehensive assessment of financial data.

3.2 Dataset

The Python code in Figure 1 provided functions as a thorough data pretreatment pipeline for financial data, specifically designed to facilitate the assessment and improvement of the Bi-LSTM (Bidirectional Long Short-Term Memory) model. The provided code implements a loop that iterates across the years from 2000 to 2022, systematically collecting historical market data for a predetermined set of stock tickers (Figure 2) for each year.

During each loop, the code does a series of essential steps:

1. **Data Retrieval:** The process of data retrieval involves utilizing the Yahoo Finance API to get historical stock data for the designated tickers. This retrieval is restricted to the temporal scope of the present year, encompassing the period from January 1st to December 31st.
2. **Data Organization:** The data retrieved is processed and grouped into a DataFrame, with a primary emphasis on the closing prices of the chosen stocks. This process generates a well-organized dataset that is appropriate for analytical purposes.
3. **Data Quality Assurance:** In order to maintain the reliability and accuracy of the dataset, the code incorporates measures to handle missing values. Initially, the algorithm discovers and eliminates columns in which the majority of entries are absent, surpassing a threshold of 5%. This practice guarantees that solely pertinent data columns are preserved for analysis.
4. **Missing Value Imputation:** Missing value imputation is performed in a data analysis process to address the issue of missing data. In this particular scenario, the algorithm is designed to intelligently impute missing values for columns that have a proportion of missing values below 5%. The algorithm substitutes missing values (NaN entries) with the mean value

of the corresponding column. The utilization of this imputation approach serves to sustain the integrity and precision of the data.

The processed data for each year is thereafter stored in separate Excel files. The dataset is systematically arranged and labeled, with each file being named in accordance with its corresponding year.

```
# Create a list of years from 2000 to 2023
years = range(2000, 2023)

for year in years:
    # Define the start and end date for the current year
    start_date = f"{year}-01-01"
    end_date = f"{year + 1}-01-01"

    # Download data for the current year
    data = yf.download(tickers, start=start_date, end=end_date)

    # Select only the 'Close' column for each ticker
    close_prices = data['Close']

    # Create a DataFrame using pandas
    df = pd.DataFrame(close_prices)

    # Drop columns containing only NaN values
    df = df.dropna(axis=1, how='all')

    # Calculate 5% of the total number of rows
    threshold = 0.05 * len(df)

    # Replace NaN values with the mean for columns with less than 5% NaN
    for column in df.columns:
        if df[column].isna().sum() < threshold:
            mean = df[column].mean()
            df[column].fillna(mean, inplace=True)
        elif df[column].isna().sum() >= threshold:
            df = df drop(column, axis=1)
            print(f"Column {column} for year {year} has been dropped
                because it exceeds the NaN threshold.")

    # Save the DataFrame to an Excel file with the name based on the year
    file_name = f"data_{year}.xlsx"
    df.to_excel(file_name, index=False)

    print(f"The DataFrame for year {year} has been saved in the file {
        file_name}.")
```

Figure 1: Python code for extracting and processing financial data.

The following stock tickers (Figure 2) have been used in the data collection and financial analysis process. These tickers were chosen to represent a variety of sectors and companies, aiming to create a diversified dataset for the analysis and improvement of the Bi-LSTM (Bidirectional Long

Short-Term Memory) model:

```
tickers = ['A', 'AA', 'AAME', 'AAON', 'AAPL', 'AB', 'ABB', 'ABC', 'ABCB', '
    ABEO', 'ABEV', 'ABIO', 'ABM', 'ABMD', 'ABT', 'ACGL', 'ACHC', 'ACHV', '
    ACIW', 'ACLS', 'ACNB', 'ACU', 'ADBE', 'ADC', 'ADI', 'ADM', 'ADMP', 'ADP'
    , 'ADSK', 'ADTN', 'ADX', 'AE', 'AEE', 'AEG', 'AEHR', 'AEIS', 'AEM', '
    AEMD', 'AEO', 'AEP', 'AES', 'AET', 'AEY', 'AEZS', 'AFG', 'AFL', 'AGCO',
    'AGEN', 'AGM', 'AGX', 'AGYS', 'AHPI', 'AIG', 'AIM', 'AIN', 'AIR', 'AIRI'
    , 'AIRT', 'AIT', 'AIV', 'AJG', 'AJRD', 'AKAM', 'AKR', 'ALB', 'ALCO', '
    ALE', 'ALG', 'ALJJ', 'ALK', 'ALKS', 'ALL', 'ALOT', 'ALV', 'ALX', 'ALYA',
     'AMAT', 'AMD', 'AME', 'AMED', 'AMG', 'AMGN', 'AMKR', 'AMNB', 'AMOT', '
    AMRN', 'AMS', 'AMSC', 'AMSWA', 'AMT', 'AMWD', 'AMZN', 'AN', 'ANDE', 'ANF
    ', 'ANIK', 'ANIP', 'ANIX', 'ANSS', 'AON', 'AOS', 'AP', 'APA', 'APD', '
    APH', 'APOG', 'APT', 'APTO', 'ARCB', 'ATVI', 'AXP', 'BA', 'BAC', 'BHP',
    'BMY', 'BP', 'BTI', 'CAT', 'CCL', 'CDNS', 'CMCSA', 'COF', 'COP', 'COST',
     'CSCO', 'CVS', 'CVX', 'DIS', 'EBAY', 'F', 'FDX', 'FITB', 'GE', 'GILD',
    'GS', 'HAL', 'HD', 'HON', 'IBM', 'INTC', 'INTU', 'JNJ', 'JPM', 'KO', '
    LMT', 'MAR', 'MCD', 'MGM', 'MMM', 'MO', 'MRK', 'MSFT', 'MSTR', 'MTB', '
    NEE', 'NEM', 'NKE', 'NOK', 'NTRS', 'NVDA', 'ON', 'ORCL', 'OXY', 'PENN',
    'PEP', 'PFE', 'PG', 'PNC', 'QCOM', 'RCL', 'SBUX', 'SCHW', 'SLB', 'T', '
    TGT', 'TRMB', 'TSM', 'TXN', 'UNH', 'UPS', 'VOD', 'VZ', 'WBA', 'WFC', '
    WMB', 'WMT', 'X', 'XOM', 'ZION']
```

Figure 2: List of 189 stock ticker symbols.

The Python code provided in Figure 3 played a crucial role in facilitating the data gathering and prepping process. Its primary purpose was to reload Excel files that had been previously prepared using a separate script in Figure 1. The code adhered to a methodical approach:

1. **Systematic Iteration:** The algorithm implemented a systematic looping method to iterate through the years of interest, starting from 2000 and ending at 2023. This approach ensured the thorough incorporation of data encompassing the entire course of the investigation.
2. **Dynamic Filename Generation:** During the iteration process, the algorithm dynamically generated filenames for Excel files, with each filename matching to a specific year. Subsequently, the data was loaded from such files into Pandas DataFrames via the `pd.read_excel()` function. This particular phase played a crucial role in transforming the unprocessed data that was saved in Excel files into a well-organized and amenable format that could be subjected to analysis.
3. **DataFrame Collection:** The resulting DataFrame after processing each year's data was carefully appended to a master list, marked as `listadf`. The deliberate collection and organization of DataFrames facilitated the smooth integration of data from all the years being studied.

4. **Dimension Reporting:** After the process of importing the data, the code was able to deliver significant insights by presenting the dimensions of each DataFrame. The information provided specifically displayed the count of rows and columns in each DataFrame. The provided information played a crucial role in understanding the magnitude and structural attributes of the data.

5. **Data Quality Check:** The code performed a data quality check in addition to showing the dimensions. The DataFrame was subjected to a systematic examination to determine the presence of any missing values (NaN) for each year. In the event that NaN values were identified, the researcher was instantly notified of their existence. Given the absence of any NaN values, the code successfully verified the integrity and comprehensiveness of the data for the respective year.

This systematic approach to data gathering and preprocessing ensured that the data was effectively prepared for further analysis and investigation. After implementing the aforementioned data production methodology, a grand total of 4,330 historical time series were acquired.

```
# Create a list of years from 2000 to 2023
years = range(2000, 2024)
listadf = []

for year in years:
    # Load the DataFrame from an Excel file
    file_name = f"data_{year}.xlsx"
    df = pd.read_excel(file_name)
    listadf.append(df)

    # Display the DataFrame dimensions
    print(f"DataFrame dimensions for year {year}: {df.shape}")

    # Check if there are any NaN values in the DataFrame
    if df.isnull().values.any():
        print("The DataFrame contains NaN values.")
    else:
        print(f"The DataFrame for year {year} does not contain any NaN
            values.")
```

Figure 3: Python code for loading financial data into DataFrames.

3.3 Data Preprocessing

Data preprocessing is a crucial phase that significantly impacts the performance of the bidirectional LSTM (Bi-LSTM) neural network. In this research, innovative data preprocessing methods were adopted to enhance the effectiveness of the network.Initially, a standard normalization process was applied to the time series data. This step ensured that the data was appropriately scaled and centered for optimal performance during the neural network's training.

Following the normalization step, two primary preprocessing techniques were introduced:

1. **Angle-Based Transformation:** This novel approach involved transforming the data points within the time series by replacing them with the angles formed by the vectors originating from the origin (0, 0) to each respective data point. Specifically, for every data point in the time series, the angle of the vector created between the origin and that particular point was meticulously calculated. This angle served as a replacement value for the original data point. The primary objective of this preprocessing step was to capture the directional and trend-based information inherent in the data. Instead of relying on the raw numerical values, these angles were leveraged to provide a more concise yet informative representation of the input data for the neural network.
2. **Interpolation of Intermediate Values:** Another innovative data preprocessing method was introduced, which involved the insertion of intermediate values between data points in the time series. These intermediary values were computed as the arithmetic mean of the coordinates between adjacent data points. This process effectively increased the data density within the time series by introducing additional data points between the original ones. This procedure was systematically applied to all intervals between the data points in the original time series. The primary objective of this method was to capture finer details and variations within the time series, thereby endowing the neural network with a more comprehensive representation of the input data. This enhanced representation aimed to facilitate a better approximation of temporal relationships and the discovery of hidden patterns within the time series.

To evaluate the efficacy of these preprocessing methods, a comprehensive experiment was conducted. A Bi-LSTM model was run on the entire set of 4,330 historical time series obtained in the previous stages of this research. This experiment was carried out multiple times, each time employing different preprocessing techniques and configurations, including no preprocessing. By systematically comparing the results across these various scenarios, the effectiveness of the proposed preprocessing methods could be rigorously assessed.

3.3.1 Baseline Bi-LSTM Model: Raw Data Input

The code in Figure 4 is designed to train and evaluate a Bidirectional Long Short-Term Memory (Bi-LSTM) neural network model for time series prediction. It consists of two main functions:

1. `prepare_data_for_lstm(x, y, sequence_length)`

   Input:

- `x`: A list or array representing the time series data.
- `y`: A list or array of corresponding target values.
- `sequence_length`: An integer specifying the length of input sequences for the LSTM model.

Output:

- `dataX`: A numpy array containing input sequences for training the LSTM.
- `dataY`: A numpy array containing corresponding target values.

This code prepares the data for training the LSTM model by creating input-output pairs. It slides a window of size `sequence_length` through the time series data and extracts input sequences of that length along with their corresponding target values. The resulting data is returned as numpy arrays for further processing.

2. `train_lstm_model(x, y, sequence_length=3, split_ratio=0.7, epochs=20, batch_size=50)`

Input:

- `x`: A list or array representing the time series data.
- `y`: A list or array of corresponding target values.
- `sequence_length`: An integer specifying the length of input sequences for the LSTM model (default is 3).
- `split_ratio`: A float indicating the proportion of data to be used for training (default is 0.7).
- `epochs`: An integer specifying the number of training epochs for the LSTM model (default is 20).
- `batch_size`: An integer specifying the batch size for training (default is 50).

Output:

- `train_mae`: The Mean Absolute Error (MAE) loss on the training data.
- `test_mae`: The MAE loss on the validation (test) data.
- `y_val`: The validation (test) data used for evaluation.

This code trains a Bi-LSTM neural network model for time series prediction. It first prepares the data using the `prepare_data_for_lstm` function. Then, it splits the data into training and validation sets according to the specified `split_ratio`. The Bi-LSTM model architecture is defined, compiled with Mean Absolute Error (MAE) loss, and trained on the training data. The training progress is monitored, and the training and validation losses are plotted. The function also makes predictions on the validation data and plots the real data vs. predicted data for both training and validation sets. Finally, it returns the training MAE, validation MAE, and the validation data for further analysis. The code also employs seeds to ensure the reproducibility of results across different runs. This seed-setting process encompasses multiple steps to control randomness, enhancing the reliability of the model. Following the insights from Willmott and Matsuura, 2005, the selection of MAE as the primary performance metric

is well-founded. Their research underscores that MAE provides a clear and unambiguous measure of average error magnitude, while RMSE and related measures exhibit limitations in conveying average error alone. Therefore, the use of MAE aligns with their recommendation for transparent and meaningful model performance assessment

```
def prepare_data_for_lstm(x, y, sequence_length):
    dataX, dataY = [], []
    for i in range(len(y) - sequence_length):
        dataX.append(y[i:i+sequence_length])
        dataY.append(y[i+sequence_length])
    return np.array(dataX), np.array(dataY)

def train_lstm_model(x, y, sequence_length=3, split_ratio=0.7, epochs=20,
    batch_size=50):
    seed_value = 42

    # 1. Set the seed for the Python hash generator (PYTHONHASHSEED)
    os.environ['PYTHONHASHSEED'] = str(seed_value)

    # 2. Set the seed for the Python random generator
    random.seed(seed_value)

    # 3. Set the seed for NumPy
    np.random.seed(seed_value)

    # 4. Set the seed for TensorFlow
    tf.random.set_seed(seed_value)

    # Prepare data for the LSTM model
    X, y = prepare_data_for_lstm(x, y, sequence_length)

    # Splitting into training and validation sets
    split_index = int(len(X) * split_ratio)
    X_train, X_val = X[:split_index], X[split_index:]
    y_train, y_val = y[:split_index], y[split_index:]

    # Create the model
    model = Sequential()
    model.add(Bidirectional(LSTM(64, activation='relu'), input_shape=(
        sequence_length, 1)))
    model.add(Dense(1))
```

```
    # Compile the model with MAE loss function
    model.compile(optimizer='adam', loss='mae')

    # Training the model with validation
    history = model.fit(X_train, y_train, epochs=epochs, batch_size=
        batch_size, validation_data=(X_val, y_val), verbose=False)

    # Plot the loss during training and validation
    plt.plot(history.history['loss'], label='Training␣Loss')
    plt.plot(history.history['val_loss'], label='Validation␣Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss␣(MAE)')
    plt.legend()
    plt.show()

    # Example prediction on the time series
    test_input = np.array([y[-sequence_length:]])  # Use the last known
        values as input for prediction
    prediction = model.predict(test_input)

    # Calculate MAE for final training and test
    train_mae = np.abs(history.history['loss'][-1])
    test_mae = np.abs(history.history['val_loss'][-1])

    # Plot results on the training set
    train_prediction = model.predict(X_train, verbose=False)
    plt.plot(x[sequence_length:split_index+sequence_length], y_train, label
        ='Real␣Data')
    plt.plot(x[sequence_length:split_index+sequence_length],
        train_prediction, label='Training␣Prediction')
    plt.xlabel('Sample')
    plt.ylabel('Value')
    plt.legend()
    plt.show()

    # Plot results on validation
    val_prediction = model.predict(X_val, verbose=False)
    plt.plot(x[split_index+sequence_length:], y_val, label='Real␣Data')
    plt.plot(x[split_index+sequence_length:], val_prediction, label='
        Validation␣Prediction')
    plt.xlabel('Sample')
    plt.ylabel('Value')
    plt.legend()
    plt.show()

    return train_mae, test_mae, y_val
```

Figure 4: Training and Evaluating a Bidirectional LSTM (Bi-LSTM) Model for Time Series Prediction Without Preprocessing, with Data Preparation and Visualization.

This code (Figure 4) serves as the foundational starting point where historical time series data is trained and evaluated without preprocessing. Subsequent sections will illustrate variations of this code, showcasing different preprocessing techniques tailored to the specific data characteristics

3.3.2 Data Normalization

During the data preprocessing stage, a critical step involves normalizing the data to guarantee that all features are standardized on a comparable scale. This normalization process is essential in order to prevent any specific feature from exerting excessive influence over the model's training procedure. The code snippet utilizes the `MinMaxScaler` to perform normalization on the target variable `y`, hence rescaling its values within the range of 0 to 1. The process of normalization is implemented to guarantee that the data is constrained within a consistent range, hence promoting the convergence of machine learning algorithms and augmenting their capacity to identify significant patterns. Moreover, within the training phase, these lines of code assume a crucial significance. The successful preprocessing of data is achieved by normalizing the target variable during the training of the model. In the subsequent stages of prediction and evaluation, the model's predictions are likewise expressed in a normalized scale. In order to enhance the comprehension and juxtaposition of model predictions with the original data, a reverse transformation is employed on both the `val_prediction` (representing the model's predictions on the validation data) and `y_val` (representing the actual values in the validation data). This procedure facilitates the restoration of the predictions and true values to their initial scales, so enabling a more comprehensible assessment of the model's performance.

```
# Normalization of the original data between 0 and 1 for y
y_scaler = MinMaxScaler(feature_range=(0, 1))
y_normalized = y_scaler.fit_transform(np.array(y).reshape(-1, 1))

# Inverse transform validation predictions and original validation data
val_prediction = y_scaler.inverse_transform(val_prediction)
y_val = y_scaler.inverse_transform(y_val)
```

Figure 5: Normalization and Inverse Transformation of Data.

3.3.3 Angle-Based Transformation

Within the framework of our time series data preprocessing phase, we have included two essential functions aimed at preparing the data prior to the training of our machine learning model. The preprocessing stage plays a crucial role in improving the accuracy of our machine learning model. The initial function, referred known as `titoli`, assumes a pivotal role in the process of converting the incoming data. The time series data is transformed into angles using a sequence of trigonometric

computations. The utilization of angle transformation is crucial as it allows the model to more effectively capture directional information included in the data, hence enhancing its prediction capabilities. The outcomes of this conversion are kept in a novel collection, prepared to be employed within the model. The second function, known as `goback`, fulfills a supplementary function. The `titoli` function is utilized during the model training process to reverse the data transformation and restore it to its initial state. This phase is of utmost importance as it enables us to juxtapose the prognostications of the model with the unaltered time series data and compute the Mean Absolute Error (MAE), a pivotal parameter for evaluating the precision of the model. The `titoli` in Figure 6 function processes a given input sequence, `titolo`, representing a time series or data points. It calculates the angles between data points and a reference vector. For each data point, it computes the angle in degrees and stores the angles in a new list called `y`. The function returns three values: `x` (representing the range of input values), `y` (containing the calculated angles in degrees), and `y1` (the original input data).

```
def titoli(titolo):
    # Extract the input data (titolo) and initialize necessary variables
    y1 = titolo
    x = list(range(len(y1)))
    y = []

    # Calculate angles between vectors and the reference vector
    for i in range(len(x)):
        vector = (x[i], y1[i] - y1[0])
        angle_rad = math.atan2(vector[1], vector[0])
        angle_deg = math.degrees(angle_rad)
        y.append(angle_deg)

    # Return the calculated x, y, and y1 values
    return x, y, y1
```

Figure 6: Calculating Angles for Time Series Data.

The `goback` in Figure 7 function takes two inputs: `cox` (horizontal component) and `angl` (angle in degrees). It performs trigonometric calculations by converting the angle from degrees to radians and then using the tangent function to calculate the vertical component `coy`. The calculated vertical component `coy` is returned as the result of the function.

```
def goback(cox, angl):
    # Convert the angle from degrees to radians
    angle_rad = math.radians(angl)

    # Calculate the vertical component (coy) using trigonometry
    coy = cox * math.tan(angle_rad)

    # Return the calculated vertical component
    return coy
```

Figure 7: Calculating Vertical Component Using Trigonometry.

3.3.4 Interpolation of Intermediate Values

Augmenting the dataset through the iterative application of the `repeat_augmentation` (Figure 9) function can be a valuable strategy. This approach gradually increases the data density by inserting average values between the original data points. It contributes to a more comprehensive representation of underlying patterns and trends, potentially leading to improved model training, reduced overfitting, and enhanced model evaluation. To compare the Mean Absolute Error (MAE), a straightforward approach involves comparing the values associated with the original data points. In essence, the predicted values generated by the model that correspond to the original data points are selected. The MAE is then calculated by measuring the mean absolute difference between these predicted values and the original data points. This comparison provides an assessment of the model's accuracy relative to the original data, eliminating the need for additional data manipulation or processing. In Figure 8 and 9, the two functions for performing this data preprocessing are shown. The `augment_with_average` (Figura 8) function accepts a vector as its argument and expands it by computing the average value between each pair of adjacent components. The algorithm sequentially traverses the input vector, calculating the arithmetic mean of each element with its neighboring element. Subsequently, both the original element and the computed average are appended to the augmented vector. The aforementioned procedure successfully increases the length of the vector by a factor of two, incorporating the average values between the existing members. You can include this LaTeX content in your document to describe the

```
def augment_with_average(vector):
    augmented_vector = []
    for i in range(len(vector) - 1):
        element = (vector[i] + vector[i + 1]) / 2
        augmented_vector.append(vector[i])
        augmented_vector.append(element)
    augmented_vector.append(vector[-1])
    return np.array(augmented_vector)
```

Figure 8: augment_with_average.

The function `repeat_augmentation` (Figura 9) accepts a vector and an integer `n` as arguments, and iteratively applies the `augment_with_average` function `n` times. This implies that the vector is supplemented through the iterative computation of element averages. The outcome is a vector that has undergone a gradual expansion by the insertion of average values between the initial elements, resulting in heightened data density.

```
def repeat_augmentation(vector, n):
    for _ in range(n):
        vector = augment_with_average(vector)
    return vector
```

Figure 9: repeat_augmentation.

3.4 Loss Function

In the domain of medical image segmentation,Cai et al., 2017, present a novel approach in their recent scholarly publication. The authors suggest employing a Jaccard Loss as a direct training method for deep neural networks, namely convolutional neural networks (CNN) and recurrent neural networks (RNN), to perform pancreatic segmentation in computed tomography (CT) and magnetic resonance imaging (MRI) scans. The Jaccard Loss, which has been proposed, is designed to optimize the Jaccard Index (JI), which serves as a metric for quantifying the degree of overlap between the predictions made by the model and the actual ground truth. This methodology allows the model to acquire knowledge about the Jaccard Index as the main objective during the training process, resulting in more accurate segmentations without the requirement of establishing a decision threshold. Expanding upon the aforementioned study, an effort was undertaken to formulate a specialized loss function specifically designed for the purpose of time series forecasting jobs. The Custom Jaccard Loss was developed with the purpose of quantifying the resemblance between the

predictions made by the model and the actual data, specifically in the domain of time series analysis. The Jaccard Index, originally designed to assess the overlap between sets, has been modified for the evaluation of time series data. This modification aims to measure the alignment between the predictions generated by a model and the actual data, employing a formula specifically designed for this purpose.

The `custom_loss` function (Figure 10) serves as a specialized metric designed to evaluate the performance of a machine learning model. Specifically, its purpose is to calculate the Jaccard coefficient, also known as the Jaccard index. This coefficient quantifies the degree of similarity between two sets of values.

The function takes two input variables:

- `y_true`: This variable represents the reference values, often referred to as true or accurate values.
- `y_pred`: This variable corresponds to the values predicted by the model.

At the beginning of the function, both sets of values are cast to the `float32` data type to ensure consistent data handling during subsequent mathematical operations.

The function proceeds by computing the maximum and minimum values between the true values (`y_true`) and the predicted values (`y_pred`), followed by the summation of these computed values. Subsequently, the Jaccard coefficient is calculated using the maximum and minimum values obtained earlier.

The output of the `custom_loss` function is the Jaccard coefficient, serving as a quantitative measure of the similarity between two sets of values. This coefficient provides valuable insights into the model's ability to align its predictions with the reference data.

```
def custom_loss(y_true, y_pred):
    y_true = tf.cast(y_true, tf.float32)
    y_pred = tf.cast(y_pred, tf.float32)

    max1 = tf.math.maximum(y_true, y_pred)
    max2 = tf.reduce_sum(max1)
    min1 = tf.math.minimum(y_true, y_pred)
    min2 = tf.reduce_sum(min1)
    jaccard = (1 - (min2 / max2)) * 100

    return jaccard
```

Figure 10: Custom Loss Function Implementation.

3.5 Additional Testing Method

In the domain of finance, the capacity to precisely forecast market fluctuations is of utmost significance for investors, financial institutions, and enterprises alike. The utilization of predictive models, specifically those that rely on recurrent neural networks like Long Short-Term Memory (LSTM), has garnered growing acknowledgement due to its capacity to augment the precision of forecasts in the realm of financial markets. Nevertheless, the training process of LSTM models is intrinsically linked to the optimization of model parameters, encompassing sequence length, batch size, and the number of epochs. The prediction performance of the model can be considerably influenced by these hyperparameters. This study centers on the utilization of an inventive technique referred to as "Iterative Forecasting" within the domain of finance. This approach is characterized by its capacity to enhance the accuracy of predictions, enabling the LSTM model to promptly adjust to fluctuations in financial data. The iterative forecasting strategy involves the recurrent training of the LSTM model, where the model's most recent forecasts are utilized as input data for subsequent training iterations. During this procedure, the model initially acquires knowledge from existing historical data and subsequently produces forecasts for a restricted duration in the future. Subsequently, these predictions are incorporated into the input data for a subsequent training iteration. This repetition is performed for a predetermined number of time steps. The main aim of this study is to construct a theoretical and practical structure that successfully combines iterative forecasting with the optimization of LSTM model parameters through the utilization of a genetic algorithm. The genetic algorithm is a computational technique that draws inspiration from natural selection processes. Its objective is to systematically navigate the hyperparameter space in order to determine the most optimal configuration that minimizes evaluation metrics, such as the Mean Absolute Error (MAE), when applied to the test set. The provided dataset in Figure 11 and 12 consists of a historical time series including a duration of three years, specifically focusing on the daily closing prices of the AAPL stock. Both figures depict the model training phase, where the non-red area shows the utilization of the Bidirectional Long Short-Term Memory (BiLSTM) model while the red area is use for a comparative analysis of the traditional training method and the unique approach provided in this research.

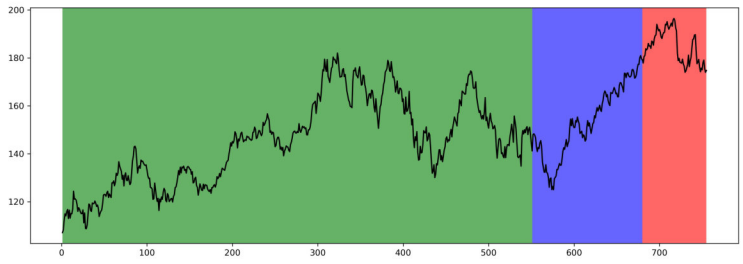

Figure 11: Training and Testing with Conventional Method.

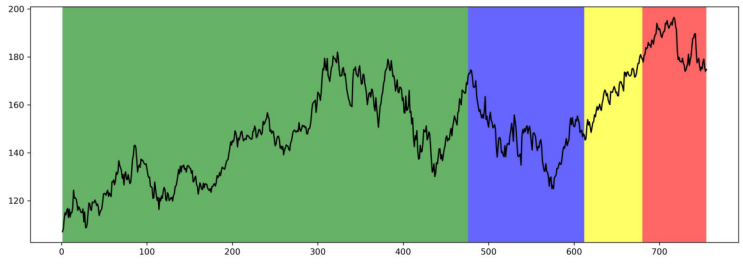

Figure 12: Iterative Forecasting Approach.

The green areas in both figures represent the data used for model training, while the blue areas represent the testing data. In Figure 12, the yellow area represents the iterative forecasting approach, where the construction of input vectors into the model is achieved through the `predict` function provided by TensorFlow and Keras. The red area, on the other hand, represents 10% of the entire historical series that is excluded from the Bi-LSTM models. This data segment is left out of the model training to facilitate a comparison between the conventional approach and iterative forecasting. Removing this red area for Figure 11 there is a data ratio of 70% (green area) and 30%(red area). Meanwhile, for Figure 12, there is a split of 70% (green area), 20%(red area), and 10%(yellow area). This latest division is achieved through code modifications in Figure 4, introducing `val_ratio` as a new parameter, and altering the code section related to data splitting. These changes are illustrated in Figure 13.In Figure 13, it is demonstrated how the Iterative Forecasting approach works. Here's how it works:

- `num_days_to_predict` is initialized with the length of the `y_test` array, representing the number of future days for which predictions are needed.
- `last_sequence` is initialized with the last available sequence of historical data from the validation dataset (`X_val[-1]`). This sequence serves as the starting point for making predictions.
- A loop iterates for `num_days_to_predict` times, where each iteration corresponds to making a prediction for the next day:
  - `next_value` is obtained by using the model's `predict` method on `last_sequence`. The input to the model is reshaped to match the expected shape `(1, sequence_length, 1)`.
  - `next_value` is appended to the `predicted_values` array.
  - `last_sequence` is updated to simulate shifting the sequence for the next prediction. The first element is removed, and `next_value` is added to the end.
- After making predictions, the code prepares the real values for comparison:
  - `real_values` is initialized with the `y_test` array.
  - Both `real_values` and `predicted_values` are reshaped into NumPy arrays with shape `(-1, 1)` to ensure compatibility for further calculations.

- The code then reverses the scaling transformation applied to the real values and predicted values using `y_scaler.inverse_transform()`. This step brings the values back to their original scale.
- Finally, the code plots the real data and the model's predictions on the same graph using Matplotlib. It also calculates the Mean Absolute Error (MAE) as a measure of the model's accuracy in predicting the future values.

```
def train_lstm_model_2(param, x, y, sequence_length=3, train_ratio=0.7,
    val_ratio=0.2, batch_size=50, epochs=50):
    seed_value = 42

    # 1. Set the seed for Python's hash generator (PYTHONHASHSEED)
    os.environ['PYTHONHASHSEED'] = str(seed_value)

    # 2. Set the seed for Python's random generator
    random.seed(seed_value)

    # 3. Set the seed for NumPy
    np.random.seed(seed_value)
    tf.random.set_seed(seed_value)

    y_scaler = MinMaxScaler(feature_range=(0, 1))
    y_normalized = y_scaler.fit_transform(np.array(y).reshape(-1, 1))

    # Prepare data for LSTM model
    X, y = prepare_data_for_lstm(x, y_normalized, sequence_length)

    # Split data into training, validation, and test sets
    split_index_train = int(len(X) * train_ratio)
    split_index_val = int(len(X) * (train_ratio + val_ratio))

    X_train, X_val, X_test = X[:split_index_train], X[split_index_train:
        split_index_val], X[split_index_val:]
    y_train, y_val, y_test = y[:split_index_train], y[split_index_train:
        split_index_val], y[split_index_val:]
```

```
# Predict the next selected days
num_days_to_predict = len(y_test)
last_sequence = X_val[-1]

predicted_values = []
for _ in range(num_days_to_predict):
    next_value = model.predict(last_sequence.reshape(1, sequence_length
        , 1))
    predicted_values.append(next_value[0][0])
    last_sequence = np.append(last_sequence[1:], next_value[0])

real_values = y_test
real_values = np.array(real_values).reshape(-1, 1)
predicted_values = np.array(predicted_values).reshape(-1, 1)
real_values = y_scaler.inverse_transform(real_values)
predicted_values = y_scaler.inverse_transform(predicted_values)

# Plot data and predictions
plt.plot(np.arange(len(x) - num_days_to_predict, len(x)), real_values,
    label='Real Data')
plt.plot(np.arange(len(x) - num_days_to_predict, len(x)),
    predicted_values, label='Predictions')
plt.xlabel('Day')
plt.ylabel('Value')
plt.legend()
plt.show()

mae_final = np.mean(np.abs(real_values - predicted_values))

num_days_to_predict2 = param
last_sequence2 = X_test[-1]

predicted_values2 = []
for _ in range(num_days_to_predict2):
    next_value2 = model.predict(last_sequence.reshape(1,
        sequence_length, 1))
    predicted_values2.append(next_value2[0][0])
    last_sequence2 = np.append(last_sequence2[1:], next_value2[0])

predicted_values2 = np.array(predicted_values2).reshape(-1, 1)
predicted_values2 = y_scaler.inverse_transform(predicted_values2)

return mae_final, predicted_values2
```

Figure 13: Iterative Forecasting: Description of the Innovative Approach in the Code.

Now, the functions of this predictive model with the addition of the iterative forecasting approach and the conventional approach are incorporated into two genetic algorithm, which is presented in Figure 14 and 15.In this study, the genetic algorithm approach has been adapted and implemented, drawing inspiration from previous research (Fotia and Ferrara, 2022). This is because the search for the best parameters for the Bi-LSTM models is an empirically driven problem, similar in nature to the problem addressed in the work "Optimized Layout: A Genetic Algorithm for Industrial and Business Application".

```
# Fitness function: returns test_mae given sequence_length and batch_size
def fitness(sequence_length, batch_size):
    test_mae = train_lstm_model(elements_to_exclude, x, y, sequence_length,
        split_ratio=0.7, epochs=195, batch_size=batch_size)
    return test_mae

# Function to create an initial population
def create_population(population_size):
    population = []
    for _ in range(population_size):
        np.random.seed(42)
        sequence_length = random.randint(10, 50)  # Set a reasonable range
            for sequence_length
        batch_size = random.randint(10, 100)      # Set a reasonable range
            for batch_size
        population.append((sequence_length, batch_size))
    return population

# Initialize an empty set to keep track of mutations already applied
mutated_set = set()

# Mutation function for sequence_length, batch_size, and epoch
def mutate(individual):
    mutated_individuals = []

    sequence_length, batch_size, epoch = individual

    sequence_length_values = list(range(max(1, sequence_length - 2),
        sequence_length + 3))
    batch_size_values = list(range(max(1, batch_size - 2), batch_size + 3))
    epoch_values = list(range(max(1, epoch - 2), epoch + 3))  # Set a
        reasonable range for epoch

    for new_sequence_length in sequence_length_values:
        for new_batch_size in batch_size_values:
            for new_epoch in epoch_values:
                mutated = (new_sequence_length, new_batch_size, new_epoch)

                # Check if this mutation has already been applied
                if mutated not in mutated_set:
                    mutated_set.add(mutated)  # Add this mutation to the
                        set of applied mutations
                    mutated_individuals.append(mutated)

    # If there are no new mutations to apply, return the original
        individual
    if not mutated_individuals:
        mutated_individuals.append(individual)

    return mutated_individuals
```

```
# Number of generations to run
num_generations = 4  # Set the desired number of generations

# Parameters
population_size = 200  # Number of parameter pairs

# Create the initial population
population = create_population(population_size)
best_params_list = []
best_fit_list=[]


for generation in range(num_generations):
    # Calculate fitness value for each parameter pair
    fitness_scores = [fitness(seq_len, batch_size) for seq_len, batch_size
        in population]

    # Find the parameter pair with the lowest fitness value
    best_index = fitness_scores.index(min(fitness_scores))
    best_params = population[best_index]
    best_fitness = fitness_scores[best_index]

    print(f"Generation {generation + 1}: Best Fitness = {best_fitness},
        Sequence Length = {best_params[0]}, Batch Size = {best_params[1]}")
    best_params_list.append(best_params)
    best_fit_list.append(best_fitness)

    # Apply mutation to the best parameter pair
    mutated_params = mutate(best_params)

    # Add the new pairs to the population
    population = mutated_params

print("Best parameters found:", best_params_list,best_fit_list)
```

Figure 14: Genetic Algorithm for Hyperparameter Optimization in Forecasting.

```
# Fitness function: returns test_mae given sequence_length, batch_size, and
    epoch
def fitness(sequence_length, batch_size, epoch):
    test_mae, _ = train_lstm_model_2(elements_to_exclude, x, y,
        sequence_length, train_ratio=0.7, val_ratio=0.2, batch_size=
        batch_size, epochs=epoch)
    return test_mae

# Function to create an initial population
def create_population(population_size):
    population = []
    for _ in range(population_size):
        sequence_length = random.randint(10, 50)
        batch_size = random.randint(10, 100)
        epoch = random.randint(10, 200)  # Set a reasonable range for epoch
        population.append((sequence_length, batch_size, epoch))
    return population

# Initialize an empty set to keep track of mutations already applied
mutated_set = set()

# Mutation function for sequence_length and batch_size
def mutate(individual):
    mutated_individuals = []

    # Extract the current values of sequence_length and batch_size
    sequence_length, batch_size = individual

    # Create new pairs of values within the ranges defined by the current
        values
    sequence_length_values = list(range(max(1, sequence_length - 2),
        sequence_length + 3))
    batch_size_values = list(range(max(1, batch_size - 2), batch_size + 3))

    # Generate all possible combinations of the new values
    for new_sequence_length in sequence_length_values:
        for new_batch_size in batch_size_values:
            mutated = (new_sequence_length, new_batch_size)

            # Check if this mutation has already been applied
            if mutated not in mutated_set:
                mutated_set.add(mutated)  # Add this mutation to the set of
                    applied mutations
                mutated_individuals.append(mutated)
```

```
    # If there are no new mutations to apply, return the original
        individual
    if not mutated_individuals:
        mutated_individuals.append(individual)

    return mutated_individuals

# Number of generations to run
num_generations = 3
population_size = 50

# Create the initial population
population = create_population(population_size)
best_params_list = []
best_fit_list=[]

for generation in range(num_generations):
    fitness_scores = [fitness(seq_len, batch_size, epoch) for seq_len,
        batch_size, epoch in population]

    best_index = fitness_scores.index(min(fitness_scores))
    best_params = population[best_index]
    best_fitness = fitness_scores[best_index]

    print(f"Generation␣{generation␣+␣1}:␣Best␣Fitness␣=␣{best_fitness},␣
        Sequence␣Length␣=␣{best_params[0]},␣Batch␣Size␣=␣{best_params[1]},␣
        Epoch␣=␣{best_params[2]}")
    best_params_list.append(best_params)
    best_fit_list.append(best_fitness)
    mutated_params = mutate(best_params)

    population = mutated_params


print("Migliori␣parametri␣trovati:",best_params_list,best_fit_list)
```

Figure 15: Iterative Forecasting: Genetic Algorithm for Hyperparameter Optimization in Forecasting.

The genetic algorithms employed in this study systematically investigate the hyperparameter space, aiming to minimize the Mean Absolute Error (MAE) associated with each model. They consist of key components: creating an initial population, defining precise fitness functions, meticulously conducting mutation operations, and employing a rigorous selection process. In Figure 15, the critical hyperparameters are `sequence_length`, `batch_size`, and `epoch`. The fitness function evaluates MAE, considering the impact of these hyperparameters. It seeks the combination that minimizes MAE through careful mutation operations, systematically exploring the hyperparam-

eter space. Contrastingly, the evolutionary algorithm in Figure 14 focuses solely on optimizing `sequence_length` and `batch_size`, excluding `epoch`. Its objective is customized optimization by fine-tuning a specific subset of hyperparameters. In fact, the epochs are automatically selected by implementing early stopping in the code, as shown in Figure 4. This is achieved by utilizing the `EarlyStopping callback` within the `Keras` library. Based on the chosen size of the initial population and the number of generations, it is possible to achieve a more accurate result, albeit at a higher computational cost. The trade-off between computational resources and optimization accuracy should be carefully considered when configuring the genetic algorithm.

Chapter 4

Experimental Results and Evidences

In the present study, a comprehensive examination has been carried out on a substantial data-set consisting of 4,330 time series for the methodological part related to Data Prepocessing and Loss Function, and with the last three year Apple close prices time series for the Iterative Forecasting part. The analytical endeavor was conducted by employing a set of predictive models that rely on Bidirectional Long Short-Term Memory (Bi-LSTM) neural networks. Each of these models has produced a distinct vector of Mean Absolute Error (MAE) values for the analyzed time series. It is imperative to acknowledge that the influence and determination of each MAE vector were contingent upon the application of a specific analytical model, hence signifying distinct reference points within the scope of our investigation. The focal point of this part is a comprehensive analysis and comparison of the MAE vectors produced by the Bi-LSTM prediction models. The main objective is to evaluate if the implementation of these novel models has led to a measurable enhancement in outcomes in comparison to the conventional model employed as the reference point for comparison. The analysis presented in this study is of great importance to our research, as it offers a comprehensive assessment of the influence of the implemented improvements on the precision of time series forecasts. The following paragraphs will provide a detailed presentation of the results obtained from this comparison, facilitating a comprehensive comprehension of the efficacy of the utilized Bi-LSTM predictive models and the potential benefits arising from the implementation of these advanced methodologies in the domain of time series forecasting.

4.1 Results with Data Preprocessing

In this section, the results obtained by modifying the Baseline model (Figure 4) using raw data through the application of preprocessing methods described in the Methodology chapter, in the section dedicated to data Preprocessing operations will be presented and analyzed. The code of the Baseline model (Figure 4) was executed on a dataset of 4330 time series, using both raw

data and normalized data(Figure 5), along with other data transformation methods developed, such as Angle-Based Transformation (Figure 6 and 7) and Interpolation of Intermediate Values (Figure 8 and 9). Through this operation, one obtains the Mean Absolute Error (MAE) of the test set for various models, considering the different data transformations. Subsequently, the three methods were compared using raw data as a reference. To evaluate the effectiveness of the different transformations, Figure 16 highlights the number of cases in which the three methods were superior or inferior to using raw data. Furthermore, Table 1 allows to understand how much better each model is on average.

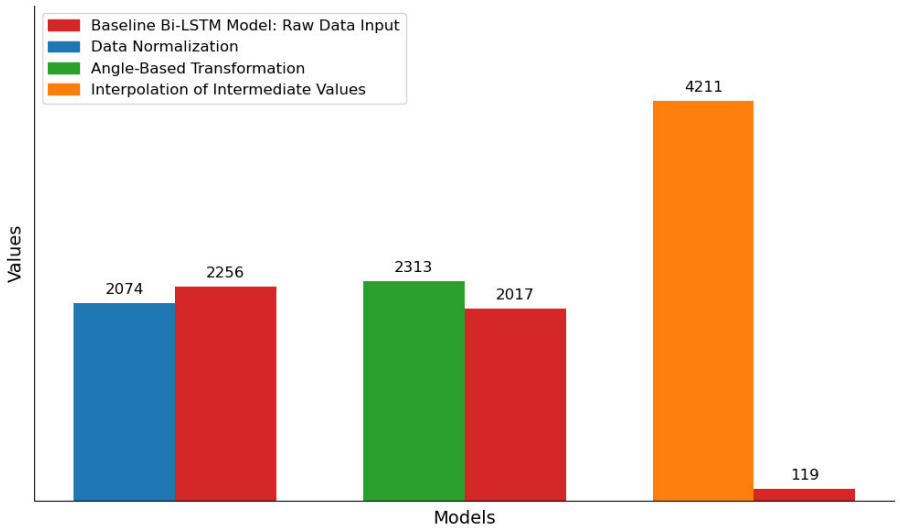

Figure 16: Frequency of Model Outperformance Comparison.

The results indicate that the application of data normalization did not yield superior performance compared to the baseline model when using raw data. The improved model outperformed the baseline model in only 2,074 out of a total of 4,330 cases. This result suggests that, within the confines of this particular scenario, the process of data normalization may not yield substantial enhancements or could perhaps be less efficacious compared to utilizing unprocessed data. When employing data normalization or other data transformation techniques, it is imperative to thoroughly evaluate the context and features of the data. The Angle-Based Transformation technique exhibited favorable outcomes, surpassing the performance of the baseline model in a total of 2,313 instances. This methodology utilizes mathematical principles to effectively capture directional patterns present in time series data. Despite the significant improvement in performance, additional research may be necessary to thoroughly investigate the potential and applicability of the method in certain scenarios that exhibit directional tendencies in time series data. The approach of Interpolation of Intermediate Values was found to be the most effective method of data transformation

in the study, outperforming the baseline model in a significant majority of cases, specifically 4,211 out of 4,330 instances. The proposed methodology entails the generation of intermediate values inside the existing data points, hence offering a more comprehensive depiction of time series data. The findings highlight the capacity of the tool to uncover intricate information inside the dataset, so becoming it a valuable asset for augmenting time series analysis, particularly in situations where precise details hold significant importance. The aforementioned findings underscore the significance of appropriate data pretreatment and transformation in the realm of time series analysis. Every technique plays a key role in enhancing the interpretation and forecasting capacities of time series models. The selection of a particular technique may be influenced by the unique qualities of the data and the objectives of the analysis, hence indicating the need for more research and domain-specific investigations to enhance the effectiveness of its implementation.

Table 1: Average Improvement

Category	Average Improvement with Preprocessing	Average Improvement with Raw Data
Data Normalization	319.4450	125.0255
Angle-Based Transformation	72.9632	1758.5835
Interpolation of Intermediate Values	187.0173	9.6633

The Table 1 illustrates the mean enhancement attained via the utilization of data preparation methodologies in contrast to the utilization of unprocessed data for the purpose of conducting time series analysis. The data preprocessing phase encompasses several categories, namely Data Normalization, Angle-Based Transformation, and Interpolation of Intermediate Values. The application of Data Normalization to time series data yielded an average improvement of roughly 319.45 units. In comparison, the utilization of raw data without any preprocessing resulted in a comparatively smaller average improvement of roughly 125.03 units. This finding suggests that the process of Data Normalization had a notable impact on the study, illustrating its efficacy in mitigating discrepancies arising from disparate scales present in time series data. The Angle-Based Transformation technique resulted in a mean improvement of roughly 72.96 units in the context of time series analysis. In contrast, the mean enhancement observed with unprocessed data exhibited a significantly greater magnitude, estimated to be roughly 1758.58 units. While the Angle-Based Transformation technique showed promising results, it is important to acknowledge that raw data consistently outperformed it on average. This indicates that additional investigation may be necessary to fully exploit the potential of this technique. The application of Interpolation of Intermediate Values in time series analysis yielded an average improvement of roughly 187.02 units. On the other hand, the utilization of unprocessed raw data resulted in a comparatively lower mean enhancement of around 9.66 units. This underscores the significant benefit of employing Interpolation of Intermediate Values as a method of data preparation, particularly in situations where detailed insights are crucial. The findings of this study offer significant insights into the effects of various data preparation methods on the mean enhancement in time series analysis. When selecting a preprocessing approach, it is important to take into account the distinct properties of the data and the

intended objectives of the analysis. This is because each technique provides distinct benefits and enhancements in performance.

4.2 Results with Different Loss Function

This investigation investigates the contrast between two distinct loss functions, namely Mean Absolute Error (MAE) Loss and Jaccard Loss, in the setting of a baseline model utilizing raw data. The aim of this study is to evaluate the circumstances in which one loss function demonstrates superior performance compared to the other, by examining the frequency of occurrences in which each function produces better outcomes.

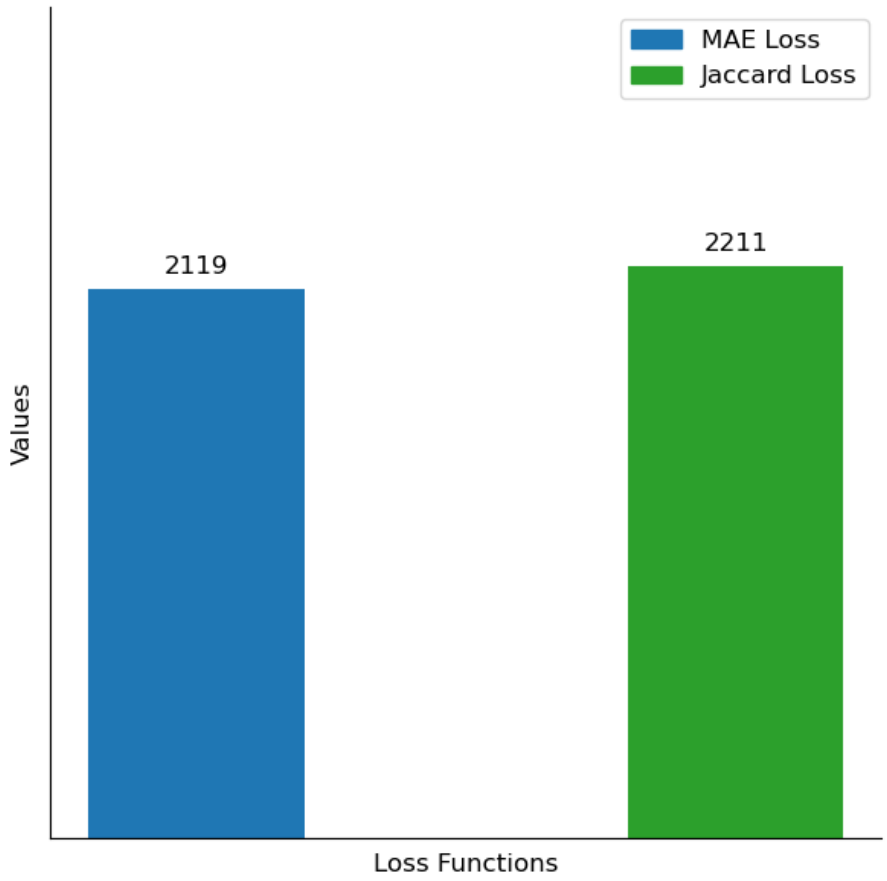

Figure 17: Comparing MAE and Jaccard Loss.

The MAE Loss demonstrates superior performance compared to the Jaccard Loss in 2,119 out of 4,330 instances. The Mean Absolute Error (MAE) is a metric used to measure the absolute discrepancy between projected and actual values, with a focus on achieving high levels of accuracy. This underscores its efficacy in situations when the reduction of prediction errors is of paramount importance. In 2,211 out of 4,330 occurrences, Jaccard Loss demonstrates superiority over MAE Loss. The Jaccard Loss is commonly employed as a metric for quantifying the dissimilarity between sets, providing valuable insights into the performance of a model. This implies that the Jaccard Loss metric may perform exceptionally well in situations where assessment criteria based on sets are better suitable or where the objective is to accurately capture the intersections and unions of sets. The aforementioned results highlight the significance of carefully choosing the suitable loss function in accordance with the particular objectives of the analysis and the characteristics of the data. The determination of whether to prioritize accuracy or set-based evaluations is contingent upon the specific context of the machine learning job being undertaken. The results reported in Tabel 2 offer

Table 2: Average Improvement

Loss Function	Average Improvement
MAE Loss	339.4556
Jaccard Loss	143.6467

valuable insights into the average improvements observed while utilizing two distinct loss functions, specifically Jaccard Loss and MAE Loss, in a specific data analysis scenario.The utilization of Jaccard Loss in this particular scenario yields an average enhancement of roughly 143.6467 units. This statistic demonstrates the improvement made when the Jaccard Loss is used as the evaluation criterion. On the other hand, while employing the MAE Loss, there is a significantly greater average improvement of around 339.4556 units. This metric quantifies the degree of improvement in performance achieved with the use of MAE Loss for evaluation. The findings presented in this study provide significant insights into the relative efficacy of the two loss functions employed within the specified analytical framework. The selection between Jaccard Loss and MAE Loss should be determined according to the specific objectives of the analysis and the intended focus on accuracy or set-based judgments.

4.3 Results with Additional Testing Method

In the domain of financial forecasting, the accurate selection of hyperparameters for predictive models holds significant importance. The accuracy of predictive models, namely Long Short-Term Memory (LSTM) networks, is significantly influenced by hyperparameters, including sequence length and batch size. The genetic algorithm, which draws inspiration from the principles of natural selection, provides a systematic approach for the exploration of optimal hyperparameter configurations.

The results of the genetic algorithm for optimizing hyperparameters in the case without iterative forecasting are presented here.

```
Generation 1: Best Fitness = 0.02882482297718525, Sequence Length = 34,
Batch Size = 10
Generation 2: Best Fitness = 0.028618091717362404, Sequence Length = 34,
Batch Size = 8
Generation 3: Best Fitness = 0.028658665716648102, Sequence Length = 34,
Batch Size = 6
Generation 4: Best Fitness = 0.02843809314072132, Sequence Length = 34,
Batch Size = 5
```

Each generation represents a cycle of parameter optimization, and the algorithm aims to minimize the Mean Absolute Error (MAE) associated with the LSTM model.The findings presented in this study provide evidence of the genetic algorithm's capacity to methodically optimize hyperparameters in order to reduce the mean absolute error (MAE). With the progression of each successive generation, the algorithm undergoes a process of iteratively refining its hyperparameters. This iterative refinement brings the algorithm closer to achieving a configuration that exhibits enhanced prediction accuracy specifically within the area of financial forecasting.

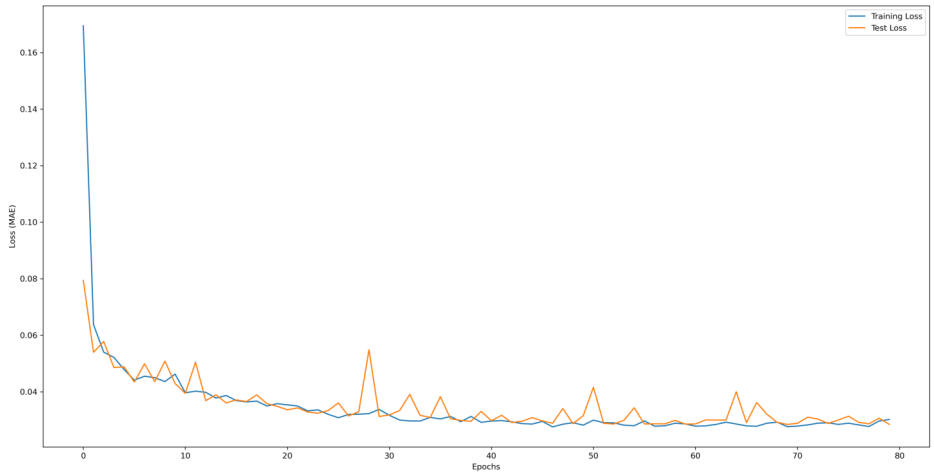

Figure 18: Training and Test Loss Plot for the Baseline Model.

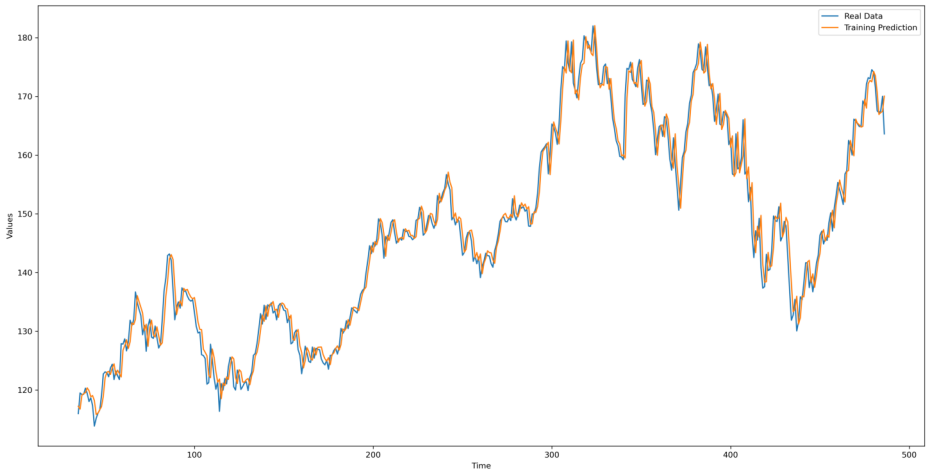

Figure 19: Training Prediction vs. Real Data for the Baseline Model.

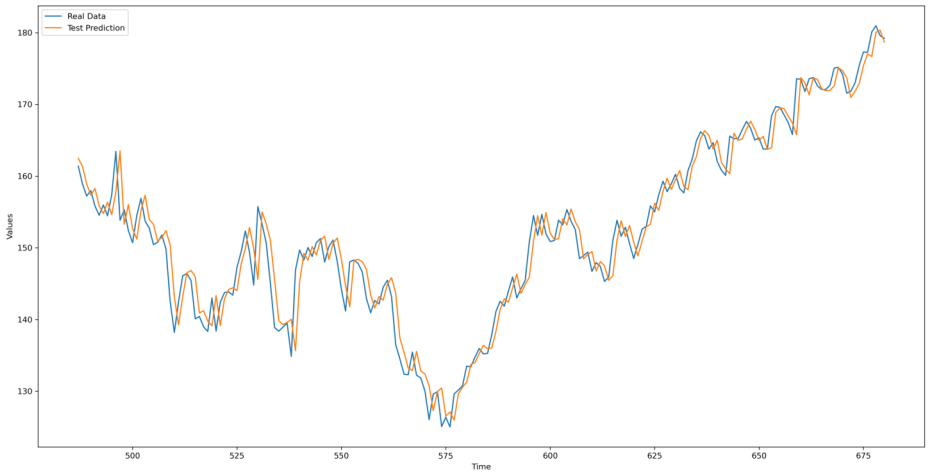

Figure 20: Test Prediction vs. Real Data for the Baseline Model.

In the domain of financial forecasting, the accurate selection of hyperparameters for predictive models holds significant importance. The accuracy of predictive models, namely Long Short-Term Memory (LSTM) networks, is significantly influenced by hyperparameters, including sequence length and batch size. The genetic algorithm, which draws inspiration from the principles of natural selection, provides a systematic approach for the exploration of optimal hyperparameter configurations. The results of the genetic algorithm for optimizing hyperparameters in the case without iterative forecasting are presented here. Every successive cohort embodies a recurring process of fine-tuning parameters, with the objective of minimizing the Mean Absolute Error (MAE) linked to the Long Short-Term Memory (LSTM) model. The findings presented in this study provide evidence of the genetic algorithm's capacity to methodically optimize hyperparameters in order to reduce the mean absolute error (MAE). With the progression of each successive generation, the algorithm undergoes a process of iteratively refining its hyperparameters. This iterative refinement brings the algorithm closer to achieving a configuration that exhibits enhanced prediction accuracy specifically within the area of financial forecasting. Significant insights on the predictive potential of our Bi-LSTM model are revealed during the study of the results acquired from the genetic algorithm performed using the iterative forecasting approach. The following are the outcomes acquired during the initial three iterations of the algorithm:

```
Generation 1: Best Fitness = 2.5230085849761963, Sequence Length = 45,
Batch Size = 11, Epoch = 184
Generation 2: Best Fitness = 2.5230085849761963, Sequence Length = 45,
Batch Size = 11, Epoch = 184
Generation 3: Best Fitness = 2.5230085849761963, Sequence Length = 45,
Batch Size = 11, Epoch = 184
```

The findings of this study demonstrate that the utilization of a genetic algorithm in the optimization of parameters for the Bi-LSTM model, within the context of iterative forecasting, has successfully identified a parameter combination that is both stable and highly effective. The observed stability of the fitness value across the initial three generations suggests that the optimized parameters identified in the first generation have been verified as the optimal selection. This finding provides confirmation of their efficacy in producing precise and consistent forecasts.

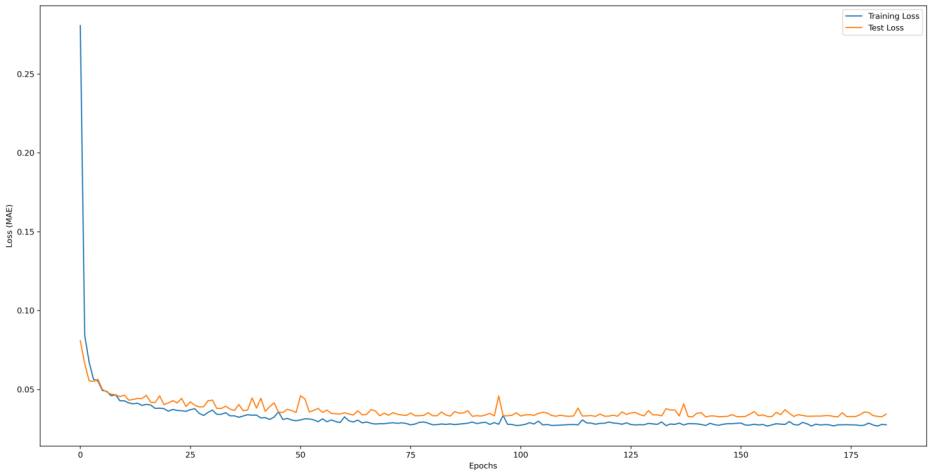

Figure 21: Training and Test Loss Plot for the Iterative Forecasting Method.

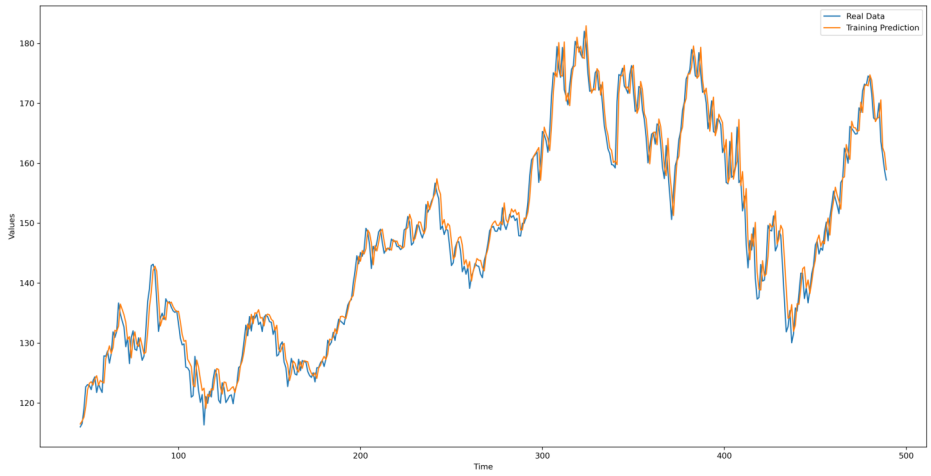

Figure 22: Training Prediction vs. Real Data for Iterative Forecasting Method.

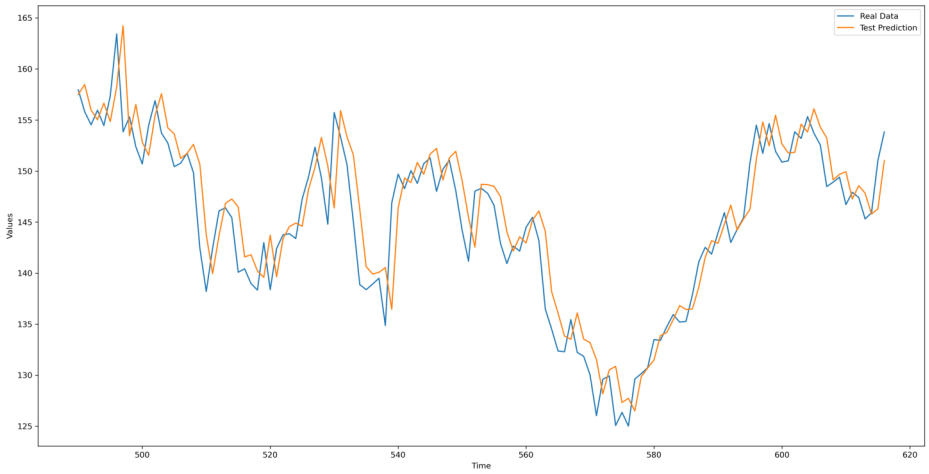

Figure 23: Test Prediction vs. Real Data for Iterative Forecasting Method.

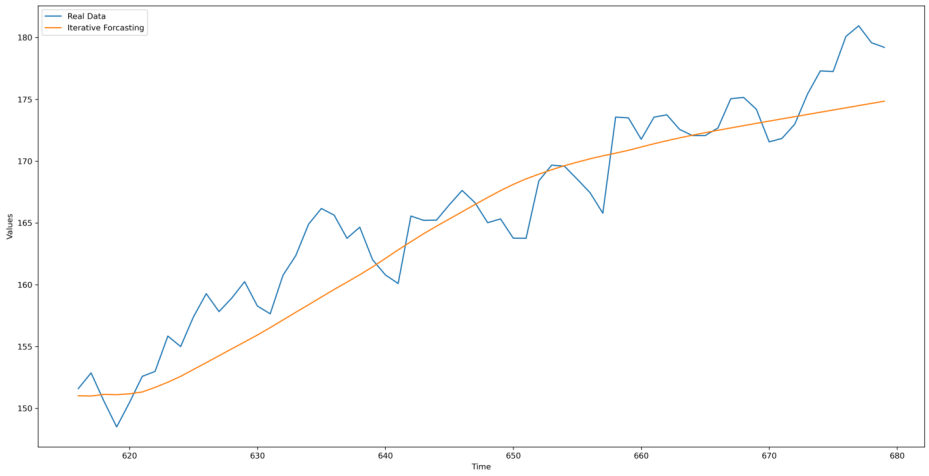

Figure 24: Iterative Forecasting Phase

The Bi-LSTM model, when configured with the specified parameters and employing the iterative forecasting approach, has enhanced capability in creating forecasts that are both more accurate and consistent in comparison to the inherent oscillations noticed in financial data. However, it is important to note (Figure 24) that these predictions may not coincide completely with the actual observed values. However, it is imperative to acknowledge that they adeptly adhere to the trajectory of the data. The observed findings illustrate that the model has effectively absorbed the pertinent correlations present in the data and is capable of successfully extrapolating to unfamiliar data.

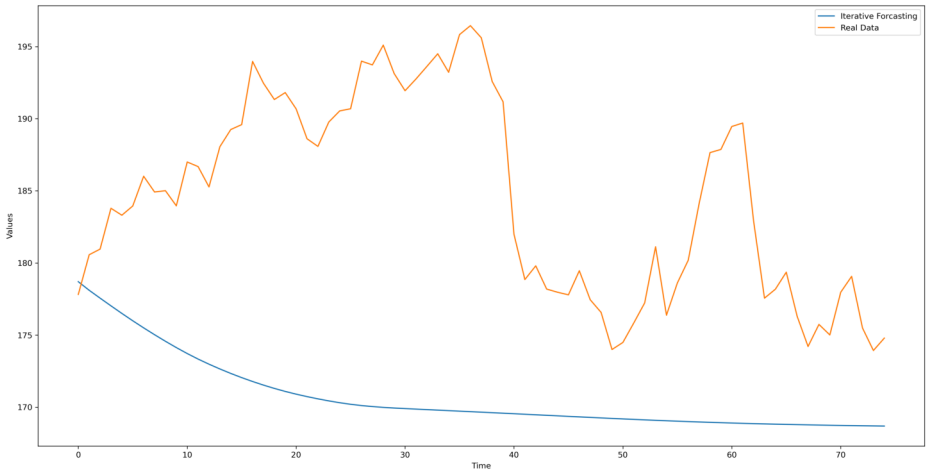

Figure 25: Data for Comparing the Two Models: Real Data vs. Baseline Model Predictions.

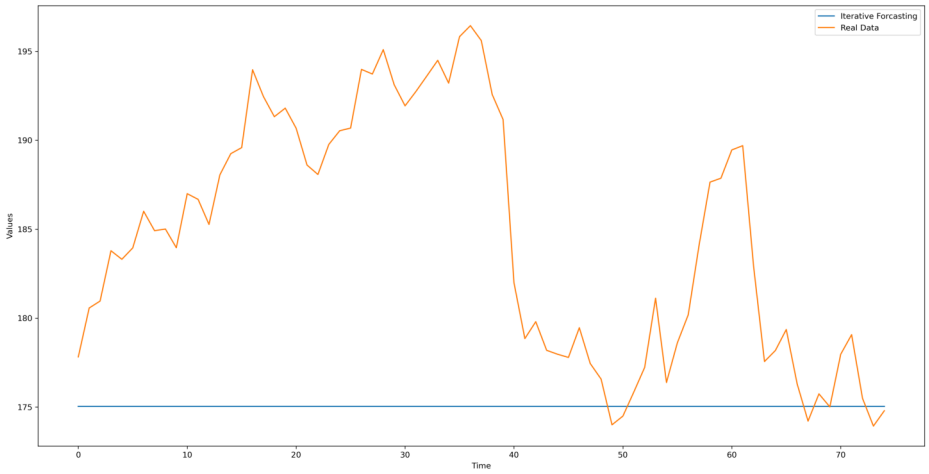

Figure 26: Data for Comparing the Two Models: Real Data vs. Iterative Forecasting Predictions.

In Figure 25 and 26, we can now thoroughly examine the comparison between the two approaches by scrutinizing the prediction results for the 10 percent of the data that has been consistently reserved from the outset, represented by the red area in Figures 12 and 13. This comparative analysis provides valuable insights into the model's performance. When comparing the two approaches, the MAE values reveal significant differences. In the case of the iterative forecasting approach, the MAE stands at 9.64, indicating a relatively lower prediction error. Conversely, for the standard model without iterative forecasting, the MAE is notably higher at 13.89, suggesting a less accurate prediction performance. These MAE values underscore the advantages of incorporating the iterative forecasting approach in enhancing the Bi-LSTM model's predictive capability, particularly when dealing with the reserved dataset. It demonstrates the effectiveness of iterative forecasting in improving prediction accuracy and aligning the model's output more closely with the observed values, ultimately enhancing its performance in financial forecasting tasks.

Chapter 5

Causal Impact Analysis of Events on Google Trends and Italian Financial Market Prices.

5.1 Introduction

In the field of finance, comprehending the causal influence of events on financial markets holds significant significance for investors, policymakers, and market participants. This chapter explores a novel methodology that utilizes Google Trends as a metric for gauging public interest and use causal impact analysis to investigate the effects of particular events on the Italian financial market. The primary objective is to evaluate the causal influence of two pivotal occurrences: the termination of constraints associated with the pandemic and the culmination of the Italian elections. The analysis encompasses the time frame spanning from January 3, 2021, to October 8, 2023.

5.2 Methodology

5.2.1 Data Collection

Data pertaining to the weekly prices of the Italian financial market was collected between the time frame of January 3, 2021, to October 8, 2023. The provided data serves as the foundational information for conducting a causal effect analysis of occurrences. Furthermore, we gathered data pertaining to Google search patterns for significant subjects linked to the chosen occurrences, employing the methodology defined in their publication. The aforementioned data played a crucial role in ascertaining the precise moment at which online public interest reached its zenith.

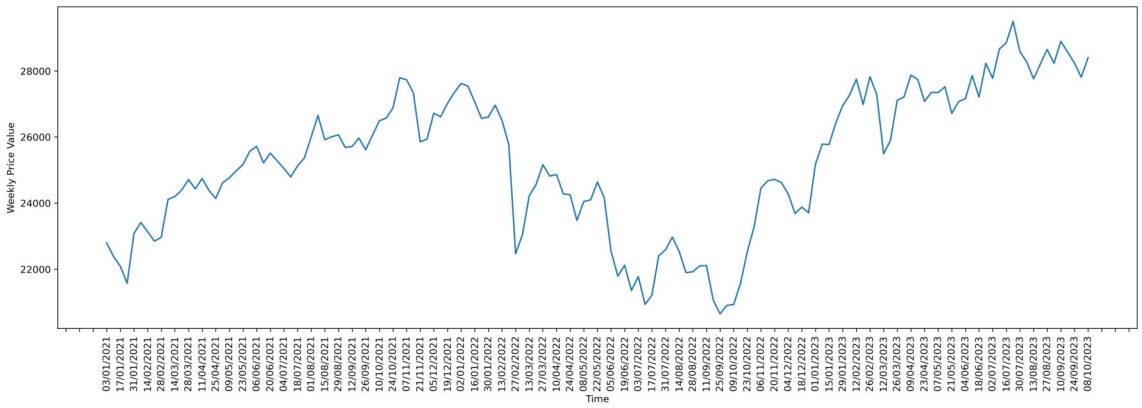

Figure 27: Weekly FTSE MIB Price Movement from 03/01/2021 to 08/10/2023.

5.2.2 Event Identification on Google Trends

In order to facilitate our research, we have selected two notable occurrences for examination: the relaxation of constraints imposed due to the epidemic (Figure 27) and the conclusion of the Italian electoral process(FIgure 28). The choice of these two events was made due to their perceived ability to have a significant impact on the Italian financial market. The stage of event identification holds significant importance as it establishes the reference points for further analysis. A Google Trends analysis was performed in order to ascertain the specific instances in which there was a notable surge in public interest for each of the chosen events. This phase played a pivotal role in ascertaining the period during which the topic garnered the highest level of internet discourse. Accurate determination of the chronological placement of these peaks is crucial for conducting subsequent investigation.

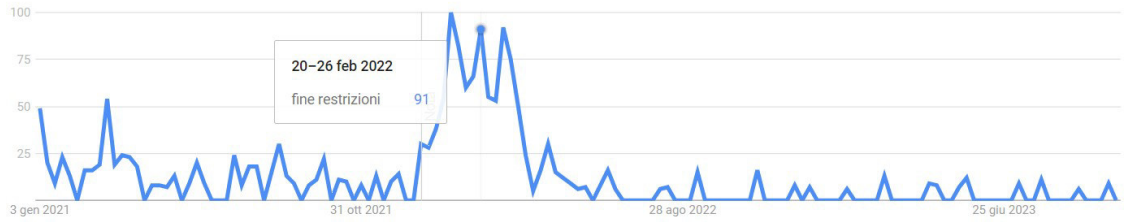

Figure 28: Google Trends: Percentage of Google Searches in Italy for 'fine restrizioni' (End of Restrictions).

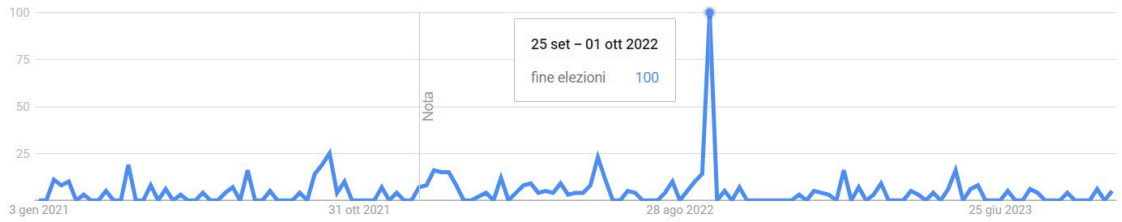

Figure 29: Google Trends: Percentage of Google Searches in Italy for 'fine elezioni' (End of Elections).

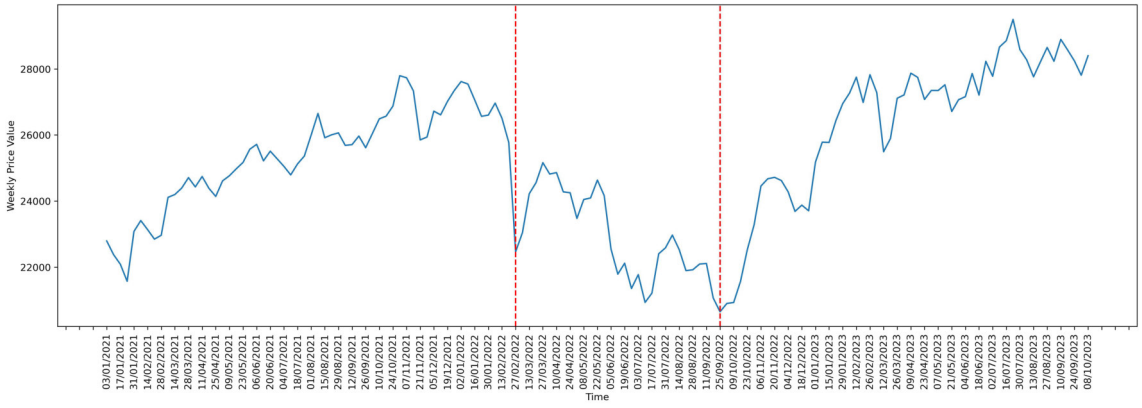

Figure 30: Weekly FTSE MIB Price Movement from 03/01/2021 to 08/10/2023 with Dashed Red Lines, indicating significant events related to the Google Trends terms 'end of elections' and 'end of restrictions

5.2.3 Causal Impact Analysis

A causal impact study was conducted, integrating components of the technique put forward by Fotia and Ferrara, 2023. A mixture of models, namely the Google CausalImpact model and the Bi-Directional LSTM model, was utilized in accordance with the recommendations provided in their publication. The utilization of the CausalImpact model, which operates inside a Bayesian framework, was employed to assess the causal influence of an event on time series data. This model adopts the methodology proposed by the authors, which integrates past knowledge and provides a quantitative assessment of uncertainty in impact estimations.

The Bi-LSTM model, however, utilizes certain phases outlined in this thesis to improve its prediction powers. In Figure 31, the data depicted by the orange curve represents the time series transformed using the Savitzky-Golay filter, implemented with the `savgol_filter` function from the SciPy library. This procedure is performed to facilitate the training of the predictive model by emphasizing the underlying data trends and reducing noise, resulting in a smoother curve that reveals data patterns more effectively.

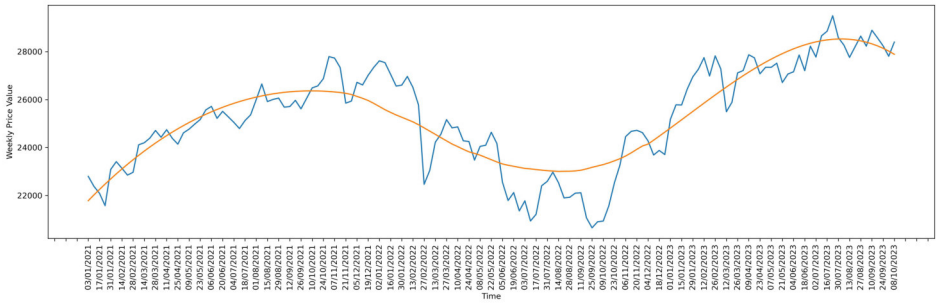

Figure 31: Smoothing of Time Series Data Using the Savitzky-Golay Filter. The orange curve demonstrates the time series data transformed with the Savitzky-Golay filter, enhancing the visualization of data trends and simplifying model training.

Figure 32 presents the Bi-LSTM model, which plays a crucial role in this investigation. The Loss function (Figure 10) has been employed to design this model, which is a crucial metric for evaluating the similarity index between two sets of data. The utilization of this particular loss function is of utmost importance in ensuring the precision of predictions within the framework of causal impact analysis. Moreover, the process of data processing has been significantly improved by means of normalization. Data normalization is a widely employed technique in the field of machine learning. Its purpose is to standardize input variables to a consistent scale. This standardization enables the model to train more effectively and generate more precise predictions. An other pivotal factor in enhancing the model's predictive capacity has been the augmentation of data volume via interpolation (Figure 8 and 9). Interpolation is a computational method utilized to generate additional data points inside a given dataset, hence augmenting the size of the training dataset. This stage holds significant importance in situations where there is a scarcity of available data. The purpose of interpolation is to improve the model's ability to make accurate predictions on unseen data and mitigate the potential problem of overfitting. The entire method to addressing causal effect analysis within a temporal forecasting context involves the integration of several parts, namely the Bi-LSTM model, the employment of the Jaccard loss function, data standardization, and interpolation. Figure 31 concisely depicts the fundamental configuration that holds significant importance in the process of analysis and prediction.

```
def train_lstm_model(x, y, sequence_length=3, train_ratio=0.7, val_ratio
    =0.2, batch_size=50, epoch=50):
    seed_value = 42

    # Set the seed for the Python hash generator (PYTHONHASHSEED)
    os.environ['PYTHONHASHSEED'] = str(seed_value)

    # Set the seed for the Python random generator
    random.seed(seed_value)

    # Set the seed for NumPy
    np.random.seed(seed_value)
    tf.random.set_seed(seed_value)

    x1 = repeat_augmentation(x, 1)
    y = repeat_augmentation(y, 1)
    y_scaler = MinMaxScaler(feature_range=(0, 1))
    y_normalized = y_scaler.fit_transform(np.array(y).reshape(-1, 1))

    # Prepare data for the LSTM model
    X, y = prepare_data_for_lstm(x1, y_normalized, sequence_length)

    # Split into training set, validation set, and test set
    split_index_train = int(len(X) * train_ratio)
    split_index_val = int(len(X) * (train_ratio + val_ratio))

    X_train, X_val, X_test = X[:split_index_train], X[split_index_train:
        split_index_val], X[split_index_val:]
    y_train, y_val, y_test = y[:split_index_train], y[split_index_train:
        split_index_val], y[split_index_val:]

    # Create the model
    model = Sequential()

    # Add the first Bidirectional LSTM layer with 64 neurons and 'relu'
        activation function
    model.add(Bidirectional(LSTM(64, activation='relu'), input_shape=(
        sequence_length, 1)))

    # Add the output Dense layer with 1 neuron (for prediction)
    model.add(Dense(1)
```

```
    # Compile the model with the MAE loss function
    model.compile(optimizer='adam', loss=custom_loss)

    # Train the model with validation
    history = model.fit(X_train, y_train, epochs=epoch, batch_size=
        batch_size, validation_data=(X_val, y_val), verbose=False)

    # Predict the next chosen days
    num_days_to_predict = len(y_test)

    # Use the last known values as input for prediction
    last_sequence = X_val[-1]

    predicted_values = []
    for _ in range(num_days_to_predict):
        next_value = model.predict(last_sequence.reshape(1, sequence_length
            , 1), verbose=False)
        predicted_values.append(next_value[0][0])
        last_sequence = np.append(last_sequence[1:], next_value[0])

    # Prepare real data for the chosen test days
    real_values = y_test

    real_values = np.array(real_values).reshape(-1, 1)
    predicted_values = np.array(predicted_values).reshape(-1, 1)

    associazione = zip(x1[-num_days_to_predict:], (np.concatenate(
        real_values)).tolist())
    associazione = list(associazione)
    filtered_tuples = [tup for tup in associazione if int(tup[0]) == tup[0]
    second_values = [tup[1] for tup in filtered_tuples]

    associazione1 = zip(x1[-num_days_to_predict:], (predicted_values))
    associazione1 = list(associazione1)
    filtered_tuples1 = [tup for tup in associazione1 if int(tup[0]) == tup
        [0]]
    second_values1 = [tup[1] for tup in filtered_tuples1]

    maef = np.mean(np.abs(np.array(second_values) - np.array(second_values1
        ))

    # Return the trained model and validation data
    return maef
```

Figure 32: Bi-LSTM Model with Jaccard Loss, Data Normalization, and Interpolation.

In Figure 33, the Iterative forcasting is employed to search for the optimal parameters. To conduct the Causal Impact Analysis as described in the article by Fotia and Ferrara, 2023, it is necessary

to iterate the predictive model across various data ranges in the time series. This analysis aims to assess the impact spanning from pre-event to post-event periods, covering the termination of restrictions and the conclusion of elections.

```
lsit_bestparams = []
lsit_best_fitness = []

for numero in range(40, 101):
    y1 = y_smooth[0:numero]
    x = np.linspace(1, len(y1), len(y1)

    # Function to calculate fitness: returns test_mae given sequence_length
        , batch_size, and epoch
    def fitness(sequence_length, batch_size, epoch):
        test_mae = train_lstm_model(x, y1, sequence_length, 0.80, 0.10,
            batch_size, epoch)
        return test_mae

    # Function to create an initial population
    def create_population(population_size):
        np.random.seed(42)
        population = []
        for _ in range(population_size):
            sequence_length = random.randint(2, int((len(y1) * 2) * 0.30))
            batch_size = random.randint(10, 100)
            epoch = random.randint(50, 300)  # Set a reasonable range for
                epoch
            population.append((sequence_length, batch_size, epoch))
        return population

    # Initialize an empty set to keep track of applied mutations
    mutated_set = set()

    # Function for mutation in sequence_length, batch_size, and epoch
    def mutate(individual):
        mutated_individuals = []

        sequence_length, batch_size, epoch = individual

        sequence_length_values = list(range(max(1, sequence_length - 2),
            sequence_length + 3))
        batch_size_values = list(range(max(1, batch_size - 2), batch_size +
             3))
        epoch_values = list(range(max(1, epoch - 2), epoch + 3))  # Set a
            reasonable range for epoch
```

```
        for new_sequence_length in sequence_length_values:
            for new_batch_size in batch_size_values:
                for new_epoch in epoch_values:
                    mutated = (new_sequence_length, new_batch_size,
                        new_epoch)

                    # Check if this mutation has already been applied
                    if mutated not in mutated_set:
                        mutated_set.add(mutated)  # Add this mutation to
                            the set of applied mutations
                        mutated_individuals.append(mutated)

        # If there are no new mutations to apply, return the original
            individual
        if not mutated_individuals:
            mutated_individuals.append(individual)

        return mutated_individuals

    num_generations = 1
    population_size = 20
    population = create_population(population_size)

    for generation in range(num_generations):
        fitness_scores = [fitness(seq_len, batch_size, epoch) for seq_len,
            batch_size, epoch in population]

        best_index = fitness_scores.index(min(fitness_scores))
        best_params = population[best_index]
        best_fitness = fitness_scores[best_index]

        print(f"Generation␣{generation␣+␣1}:␣Best␣Fitness␣=␣{best_fitness},
            ␣Sequence␣Length␣=␣{best_params[0]},␣Batch␣Size␣=␣{best_params
            [1]},␣Epoch␣=␣{best_params[2]}")
        lsit_bestparams.append(best_params)
        lsit_best_fitness.append(best_fitness)
        mutated_params = mutate(best_params)

        population = mutated_params
```

Figure 33: Model Parameter Optimization for Causal Impact Analysis.

5.3 Causal Impact Analysis results

The study prioritized the examination of causal effect in order to get insight into the influence of individual events or causes on the performance of the FTSEMIB market index. The data was analyzed throughout the time frame spanning from October 10, 2021, to December 4, 2022. This study aimed to ascertain the methods for identifying such affects by employing two distinct models: Google's causal impact model and the Bi-LSTM model. The results of this investigation can be observed in Figure 34. The causal effect model developed by Google exhibited a notable capacity to accurately assess the influence of the cessation of imposed restrictions on the FTSEMIB market index over the analyzed timeframe. Indeed, commencing from that particular date, a discernible escalation in effects becomes evident in the ensuing days. This implies that Google's model effectively identifies the positive impact of the removal of limitations on the market index. In contrast, the Bi-LSTM model successfully recognized the importance of the conclusion of electoral events as the primary determinant of the FTSEMIB market index's performance. This model places significant emphasis on the impact of elections as a primary catalyst for changes in the performance of the market index.

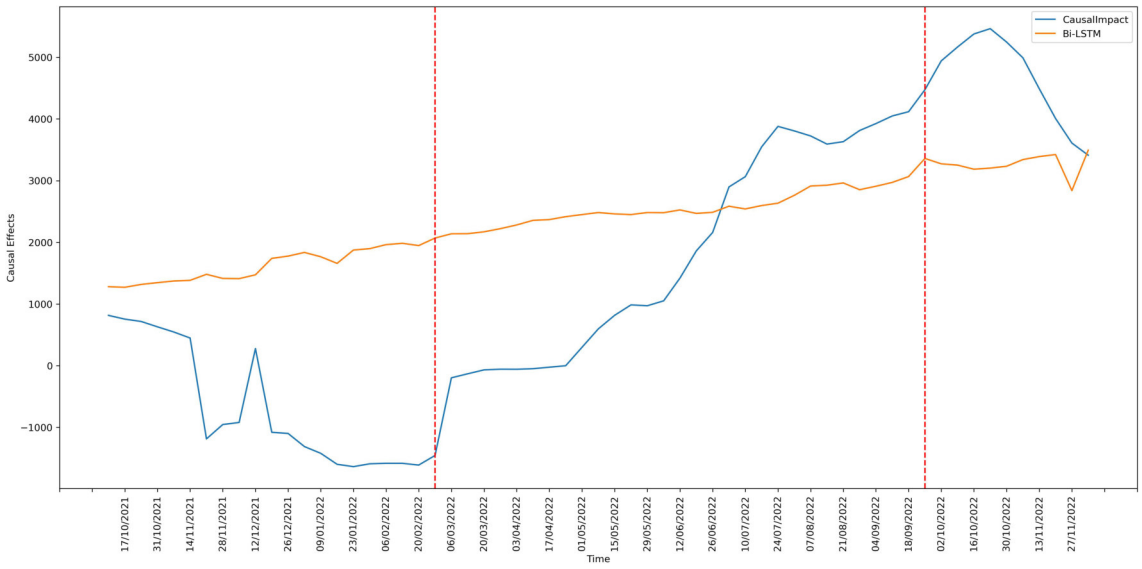

Figure 34: Comparison of predicted causal effects of CausalImpact and Bidirectional LSTM models.

Chapter 6

Conclusions

This research has embarked on a comprehensive exploration of financial time series, focusing on an extensive and diverse dataset comprising 4,330 individual time series. The central objective of this study was to develop innovative strategies and sophisticated methodologies to enhance the precision of financial forecasting. The research was organized into several pivotal phases, each contributing significantly to achieving the established goals. The data preprocessing phase served as a crucial step in ensuring the readiness of the time series data for analysis. During this stage, a thorough evaluation and implementation of various data transformation techniques were undertaken, including data normalization, angle-based transformation, and interpolation of intermediate values. A thorough investigation revealed that the incorporation of intermediate data through interpolation proved to be particularly effective in improving the accuracy of predictions. However, it is important to note that the choice of a data transformation method should be tailored to the unique characteristics of the time series data and the specific objectives of the research. These findings underscore the critical importance of meticulous data preparation in the realm of financial forecasting. Another pivotal aspect of this study was the examination of different loss functions. The evaluation of Mean Absolute Error (MAE) Loss and Jaccard Loss highlighted the significance of selecting an appropriate loss function to achieve accurate predictions. MAE Loss demonstrated its effectiveness in minimizing prediction errors, whereas Jaccard Loss showed its utility in scenarios that require assessing the similarity of data sets. These findings emphasize the need to customize the choice of loss function based on the research objectives. A significant advancement introduced in this study was the adoption of an iterative forecasting methodology, which enabled the optimization of Bi-LSTM model parameters using a genetic algorithm. This approach exhibited notable effectiveness in enhancing the precision of financial forecasts. A detailed analysis of the results revealed that, as successive generations of the genetic algorithm were iterated, model parameters consistently improved, resulting in increasingly precise and reliable predictions. This innovation represents a significant step forward in the field of financial forecasting, providing enhanced stability and dependability in predictive capabilities. However, it is essential to acknowledge potential limitations in this study. One limitation relates to the heterogeneity of the dataset, which can introduce variability into the results and may not fully account for all real-world scenarios. Furthermore, while the iterative forecasting approach displayed promise, further research is required to explore its applicability to different financial markets and asset classes. Additionally, the effective-

ness of the proposed methodologies may be influenced by specific market conditions and economic factors that were not comprehensively addressed in this study. Furthermore, this study builds upon the aforementioned contributions by utilizing the previously created approaches to improve the Bi-LSTM model, aiming to get a more thorough analysis. The scope of the study was expanded to include a causal impact analysis of financial time series, with the objective of identifying and comprehending the causal links that drive market dynamics. The causal impact analysis provided significant insights into the impact of several events and circumstances on the financial time series under investigation. Through the utilization of advanced approaches created for the Bi-LSTM model, this research has elucidated the manner in which particular events and market conditions exerted an influence on the observed data. This enhanced our comprehension of the complex network of cause-and-effect connections within the financial system. The findings of the causal impact analysis have contributed an additional level of comprehension to the study, elucidating the manner in which external occurrences and market dynamics might induce fluctuations and disruptions in financial time series. The acquisition of this knowledge is a substantial contribution to the domain of financial forecasting, as it provides a comprehensive viewpoint on the determinants influencing market dynamics. In brief, this study improved the accuracy and reliability of financial projections by employing meticulous data preparation, selecting suitable loss functions, and utilizing an innovative iterative forecasting approach. Additionally, the research expanded the scope of financial projections to include a comprehensive analysis of causal impact. The aforementioned findings establish a robust basis for future advancements in the domain of financial forecasting, signifying a noteworthy progress in the application of recurrent neural networks in this perpetually expanding sector.

Bibliography

Ashour, A., El-Attar, A., Dey, N., El-Kader, H., & El-Naby, M. (2020). Long short term memory based patient-dependent model for fog detection in parkinson's disease. *Pattern Recognit Lett*, *131*, 23–29.

Cai, J., Lu, L., Xie, Y., Xing, F., & Yang, L. (2017). Improving deep pancreas segmentation in ct and mri images via recurrent neural contextual learning and direct loss function. *arXiv preprint arXiv:1707.04912*.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., et al. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Fotia, P., & Ferrara, M. (2022). Optimized layout: A genetic algorithm for industrial and business application. *International Conference on Applied Intelligence and Informatics*, 117–126.

Fotia, P., & Ferrara, M. (2023). A different approach for causal impact analysis on python with bayesian structural time-series and bidirectional lstm models. *Atti della Accademia Peloritana dei Pericolanti-Classe di Scienze Fisiche, Matematiche e Naturali*, *1*(2), 12.

Graves, A., Jaitly, N., & Mohamed, A.-r. (2013). Hybrid speech recognition with deep bidirectional lstm. *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, 273–278.

Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Netw*, *18*(5-6), 602–610.

Halpern-Wight, N., Konstantinou, M., Charalambides, A. G., & Reinders, A. (2020). Training and testing of a single-layer lstm network for near-future solar forecasting. *Applied Sciences*, *10*, 5873. https://doi.org/10.3390/app10175873

Harrou, F., Kadri, F., & Sun, Y. (2020). Forecasting of photovoltaic solar power production using lstm approach.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Comput*, *9*(8), 1735–1780.

Liu, Z.-x., Zhang, D.-g., Luo, G.-z., Lian, M., & Liu, B. (2020). A new method of emotional analysis based on cnn-bilstm hybrid neural network. *Cluster Comput*, 1–13.

Oksuz, I., Cruz, G., Clough, J., Bustin, A., Fuin, N., Botnar, R., et al. (2019). Magnetic resonance fingerprinting using recurrent neural networks. *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*, 1537–1540.

Schuster, M., & Paliwal, K. (1997). Bidirectional recurrent neural networks. *IEEE Trans Signal Process*, *45*(11), 2673–2681.

Sharfuddin, A., Tihami, M., & Islam, M. (2018). A deep recurrent neural network with bilstm model for sentiment classification. *2018 International Conference on Bangla Speech and Language Processing (ICBSLP)*, 1–4.

Tsai, Y. T., Zeng, Y. R., & Chang, Y. S. (2018). Air pollution forecasting using rnn with lstm. *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*, 1074–1079.

Wang, S., Wang, X., Wang, S., & Wang, D. (2019). Bi-directional long short-term memory method based on attention mechanism and rolling update for short-term load forecasting. *Int J Electr Power Energy Syst*, *109*, 470–479.

Willmott, C. J., & Matsuura, K. (2005). Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, *30*(1), 79–82.

Yamak, P. T., Yujian, L., & Gadosey, P. K. (2019). A comparison between arima, lstm, and gru for time series forecasting. *Proceedings of the 2019 2nd international conference on algorithms, computing and artificial intelligence*, 49–55.

Zhang, B., Zhang, H., Zhao, G., & Lian, J. (2020). Constructing a pm2.5 concentration prediction model by combining auto-encoder with bi-lstm neural networks. *Environ Modell Softw*, *124*, 104600.