Development of a Prediction Model for NTT Stock Prices

Kalalou Ilias

March 22, 2025

Contents

1 Introduction

This report describes the project to develop a prediction model for NTT stock prices as part of an internship. The objective is to develop a solution for analyzing and forecasting time series using machine learning methods. The project was carried out in several stages:

- Data understanding and Exploratory Data Analysis (EDA)
- Data preprocessing and feature engineering
- Selection and training of prediction models
- Performance evaluation and results analysis
- Model improvement and iterative refinement

2 Exploratory Data Analysis (EDA)

2.1 Data Overview

The data used in this project comes from the NTT stock prices. Each record includes the following information:

- (Date)
- (Closing Price)
- (Opening Price)
- (Highest Price)
- (Lowest Price)
- (Volume)
- % (Percentage Change)

An initial overview was obtained by displaying the first five rows and the dimensions of the data matrix. Subsequently, several tables were generated to examine the structure of the data in detail and draw conclusions on data quality. For example, a graph of the closing price was plotted, which allowed us to observe the overall trend of the stock.

2.2 Visualizations and Descriptive Analysis

To better understand the behavior of the stock price, several graphs were created:

- **Trend in NTT Closing Price:** This graph (Figure 1) shows the closing price curve from the initial listing of NTT on the stock market. It can be observed that NTT had a very successful market debut, followed by a sharp decline before eventually stabilizing and gradually increasing.
- **Distribution of the Closing Price:** This graph (Figure 2) shows the histogram and density curve of the closing prices. It reveals that the most frequent values lie between 30 and 60 dollars, providing an estimate of the average stock price.

- **Monthly Variation of the Closing Price:** The graph shown in Figure 3 illustrates the monthly variations of the closing price. High volatility is observed from March to June and in October. This suggests that particular attention should be paid to these periods during modeling.

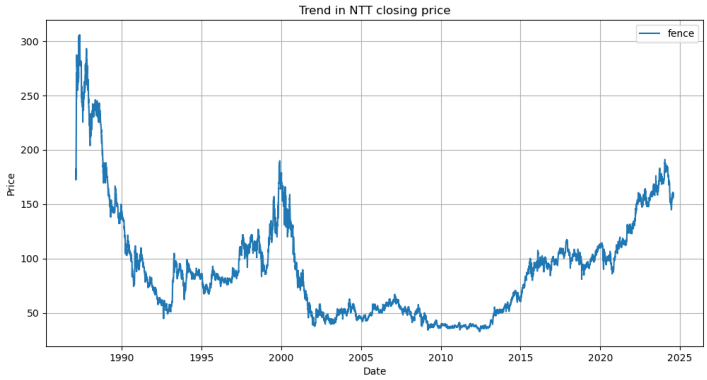

Figure 1: Trend in NTT Closing Price

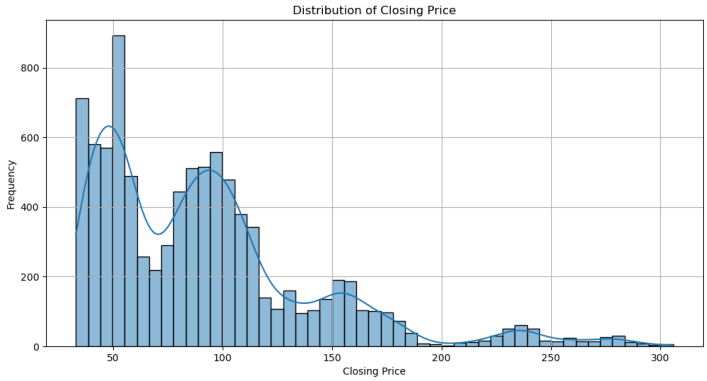

Figure 2: Distribution of the Closing Price

Figure 3: Monthly Variation of the Closing Price

These visualizations, along with descriptive statistics (mean, median, standard deviation, etc.), allowed us to identify overall trends and potential anomalies in the data, forming the basis for the choice of prediction models.

3 Data Preprocessing and Feature Engineering

3.1 Cleaning and Reindexing

Data preprocessing was carried out in several steps:

- **Date Conversion:** The column was converted to `datetime` format to facilitate temporal operations.
- **Reindexing:** The DataFrame was reindexed to include all business days (from the minimum to the maximum date). Missing values (due to weekends and holidays) were then interpolated using a linear method, resulting in a continuous time series.

3.2 Normalization

To ensure that all variables are on a comparable scale, the following numeric columns were normalized using the `MinMaxScaler`:

- (Closing Price)
- (Opening Price)
- (Highest Price)
- (Lowest Price)
- (Volume)

The column was pre-processed using a dedicated function (which converts strings like "79.15M" into absolute numbers). Additionally, to stabilize variance and facilitate modeling, a logarithmic transformation was applied to the target :

$$\texttt{log()} = \ln\Big(+ 10^{-6}\Big)$$

3.3 Creation of Time Sequences

For neural network models, the time series was divided into sequences. In our approach, sequences of length 20 were used to predict the next value of the target (i.e., `log()`). This step is crucial to capture temporal dependencies and provide the model with sufficient contextual information for accurate predictions.

4 Prediction Models and Training

4.1 ARIMA Model

The ARIMA (AutoRegressive Integrated Moving Average) model was chosen for several reasons:

- **Advantages:**
  - *Simplicity and interpretability:* ARIMA is a classical statistical model that identifies the autoregressive (AR) and moving average (MA) components and determines the need for differencing (I) to render the series stationary.
  - *One-step-ahead forecasting efficiency:* By basing forecasts on past values and previous errors, ARIMA can provide robust forecasts for the next time step.
- **Disadvantages:**
  - *Linearity assumption:* As a linear model, ARIMA may be limited in capturing non-linear dynamics present in financial data.
  - *Degradation of multi-step forecasts:* When forecasting multiple steps ahead, the rolling forecast method tends to accumulate errors, often leading forecasts to converge toward the mean.

These characteristics make ARIMA a pertinent model to establish a baseline and analyze the linear dynamics of time series, while also serving as a reference to compare with more advanced approaches.

4.1.1 Implementation of the ARIMA Model

The ARIMA model was implemented following these steps:

1. **Target Transformation:** The (closing price) column was log-transformed to stabilize variance:
   $$\texttt{\_log} = \ln\left(+ 10^{-6}\right)$$
   This helps manage large variations and makes the series more stationary.
2. **Stationarity Test:** An Augmented Dickey-Fuller (ADF) test was performed on the log-transformed series to check for stationarity. The results (e.g., 0.34) confirmed the need for differencing (d=1).
3. **Train/Test Split:** The log-transformed series was split into two sets, with approximately 95% of the data used for training and 5% for testing. This allows the model to be trained on a large volume of historical data and to evaluate its forecasting ability on new data points.
4. **Rolling Forecast:** A rolling forecast approach was adopted to simulate a real-time prediction scenario. The ARIMA model (of type ARIMA(0,1,1)) is retrained at each iteration on the available history to make a one-step-ahead forecast. At each step, the actual observed value is added to the history, enabling continuous model updates.
   $$Predictiony_{t+1} = ARIMA(y_1, y_2, \ldots, y_t)$$
   The forecasts are then exponentiated to return to the original scale.

4.1.2 Forecast Evaluation for ARIMA

To evaluate the ARIMA model's reliability, several metrics were computed:

- **RMSE (Root Mean Squared Error)** and **MAE (Mean Absolute Error)**: These metrics measure the overall discrepancy between forecasts and actual values.
- **Customized Trend and Proximity Evaluation:** An algorithm was developed to assess the model's ability to correctly predict the direction (up or down) of the price movement and the closeness of the forecast to the actual value. For each day (starting from the second), the daily changes of the actual and forecast values are compared:
  - *Trend Accuracy:* The percentage of days where the sign of the change is correctly predicted.
  - *Proximity Accuracy:* A score is assigned daily based on the absolute difference between the forecast and the actual value, with a maximum score (100%) for zero difference and a score of 0% for a difference of 1$ or more.

  The global score is defined as the average of these two accuracies.

4.1.3 Results for ARIMA

The ARIMA model produced the following results:

- **RMSE:** 1.71
- **MAE:** 1.32

Furthermore, the customized evaluation yielded:

- **Trend Accuracy:** Ranging from 54% to 63% (depending on the train/test split).

These results indicate that, although the ARIMA model performs well in forecasting the next value, its ability to predict longer horizons is limited due to the accumulation of errors.

4.1.4 Visualization of ARIMA Results

Figure 4 below illustrates the forecast curve obtained using the rolling forecast method of the ARIMA model compared to the actual values. The forecast curve generally follows the overall trend of the stock, although some deviations persist. These deviations are mainly due to the linear nature of ARIMA and the progressive accumulation of errors in multi-step forecasts.

After extensive testing, we found that the ARIMA model performed significantly better when complete data (i.e., after interpolating missing values) was used and when the data was not normalized. Normalization slightly altered the scale of the values, affecting forecast accuracy, whereas using complete data better captured the intrinsic dynamics of the stock price.

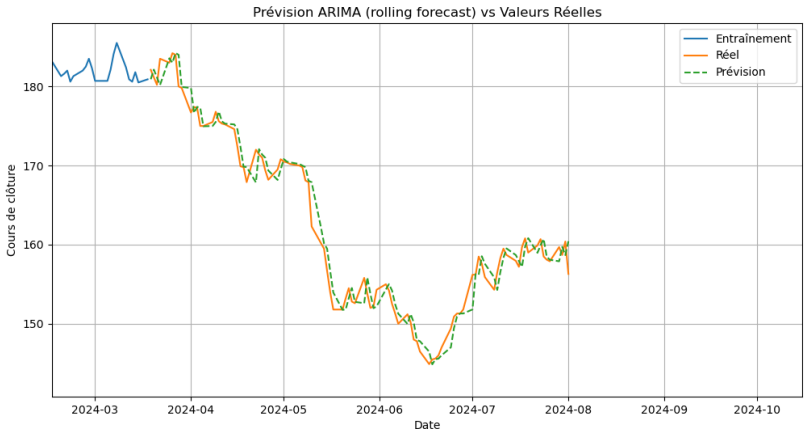

Figure 4: ARIMA Rolling Forecast vs. Actual Values

4.2 LSTM Models

A univariate LSTM model was developed to serve as a reference and to compare its performance with the ARIMA model. The choice of an LSTM (Long Short-Term Memory) is based on its ability to capture long-term dependencies in time series, overcoming issues such as exploding or vanishing gradients that affect standard recurrent neural networks.

Advantages:

- *Long-term dependency management:* The internal memory structure of LSTMs enables them to retain information over long periods, which is particularly useful for financial series with complex dynamics.
- *Adaptability to non-linearities:* Unlike linear models like ARIMA, LSTMs can model non-linear relationships between variables, offering potential improvements for highly volatile series.

Disadvantages:

- *Computational complexity:* Training LSTMs generally requires more computation time and resources, particularly if hyperparameters are not well-tuned.
- *Large data requirements:* LSTMs often need a large number of observations to achieve reliable performance, which can be challenging for smaller time series.

4.2.1 Implementation of the LSTM Model

The LSTM model was implemented through the following steps:

1. **Data Preparation:** The time series corresponding to the (closing price) column was extracted either in its raw form or normalized. Subsequently, fixed-length time sequences (e.g., 10 time steps) were created to serve as input to the network.

2. **Train/Test Split:** The series was split into training and test sets, for example, using 95% of the sequences for training and 5% for testing. This split allows the model to be trained on a large volume of historical data while evaluating its ability to predict unseen values.

3. **Model Architecture:** The LSTM model consists of two recurrent layers with 50 neurons each, followed by a fully connected layer to produce the forecast. The initial states of the LSTM are set to zero for each mini-batch during training.

4. **Training and Optimization:** The model was trained for 50 epochs using the Adam optimizer and a Mean Squared Error (MSE) loss function. A progress bar was displayed to monitor the training loss.

5. **Prediction and Evaluation:** After training, forecasts were generated on the test set. RMSE and MAE metrics were computed to quantify the overall error, and a custom evaluation algorithm measured the trend accuracy (i.e., correctly predicting the direction of change) and proximity score, yielding a global reliability score.

4.2.2 Evaluation and Results for LSTM

The evaluation of the LSTM model was performed on both raw and normalized data. For raw data, the error metrics were very high (RMSE = 159.84, MAE = 159.49), while on normalized data, the errors were extremely low (RMSE = 0.14, MAE = 0.14). This discrepancy indicates that the very low errors on normalized data are primarily due to the reduced scale rather than truly accurate predictions. Overall, the LSTM demonstrates potential for modeling non-linear relationships, but its performance in this project is limited, and further improvements (e.g., better hyperparameter tuning or more sophisticated architectures) are required. The LSTM thus serves as a baseline for comparison with more classical methods like ARIMA.

4.2.3 Visualization of LSTM Results

Figure 5 illustrates the comparison between actual values and forecasts produced by the LSTM model, both for raw and normalized datasets. Although the model sometimes follows the general trend, significant discrepancies highlight the limitations of the LSTM approach for this specific time series.

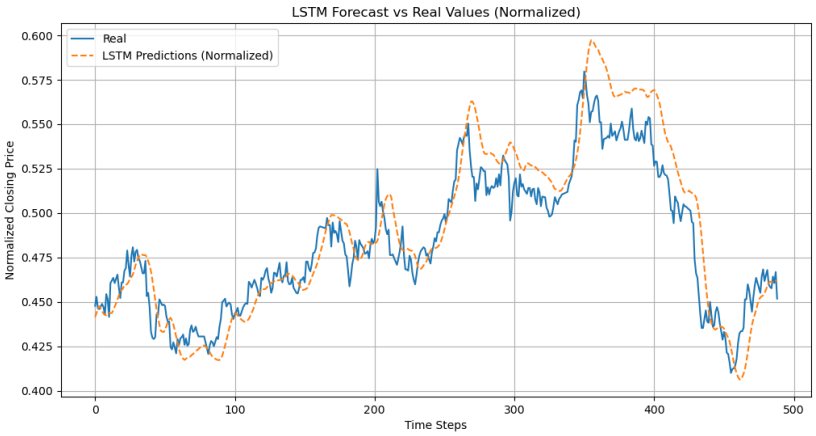

Figure 5: LSTM Forecast vs. Actual Values

4.3 GRU Multivariate

This model exploits multiple features to predict the logarithmic transformation of . Unlike a univariate model that only considers the target series, the multivariate GRU integrates various indicators (e.g., opening price, highest price, lowest price, and volume) to better capture the interactions and dynamics of financial data.

4.3.1 Advantages:

- *Utilization of multivariate information:* Using several features simultaneously allows the model to detect complex interactions and improve forecast accuracy.
- *Parameter efficiency:* GRUs generally require fewer parameters than LSTMs, making them easier to train on large datasets.
- *Training stability:* Techniques such as gradient clipping help stabilize training, which is particularly important in highly volatile financial series.

4.3.2 Disadvantages:

- *Hyperparameter sensitivity:* The model's performance is highly influenced by choices regarding sequence length, learning rate, train/test split, etc., requiring extensive experimentation.
- *Preprocessing complexity:* Preparing multivariate data demands rigorous normalization to prevent differences in scale from biasing the learning process.

4.3.3 Implementation and Training:

The multivariate GRU model was implemented as follows:

1. **Data Preparation:** The column was extracted (either raw or normalized), and additional features were integrated into the DataFrame. The series was then segmented into fixed-length time sequences (e.g., 10 time steps) using a dedicated function.

2. **Train/Test Split:** The sequences were split into training and testing sets, for instance using 95% of the data for training and 5% for testing.

3. **Model Architecture:** The GRU multivariate model is built with two GRU layers, each containing 100 hidden neurons, followed by a fully connected layer that outputs the forecast. Gradient clipping is applied to stabilize training.

4. **Training:** The model was trained for 50 epochs using an Adam optimizer with a learning rate of 0.001. At each epoch, the model minimizes the Mean Squared Error (MSE) between its predictions and the actual values.

4.3.4 Evaluation and Results:

The performance of the GRU multivariate model was evaluated using classical metrics (RMSE and MAE) as well as a custom evaluation measuring trend accuracy and proximity. For example, using the normalized series, the model yielded:

- **On non-normalized data:**
  - *RMSE:* 155.58
  - *MAE:* 155.22
  - *Trend Accuracy:* 48.77%
- **On normalized data:**
  - *RMSE:* 0.017
  - *MAE:* 0.014
  - *Trend Accuracy:* 48.15%

These results suggest that while the multivariate GRU helps reduce prediction errors compared to univariate approaches, the extremely low error values on normalized data reflect mainly the effect of scaling rather than truly accurate predictions. Furthermore, the trend accuracy remaining around 48–49% indicates that the model struggles to correctly anticipate daily changes. This points to the need for further enhancements, such as improved feature selection or more refined hyperparameter tuning.

4.3.5 Visualization of GRU Results:

Figure 6 compares actual values with the forecasts generated by the multivariate GRU model. Although the model generally follows the overall trend, there are noticeable discrepancies in daily changes.

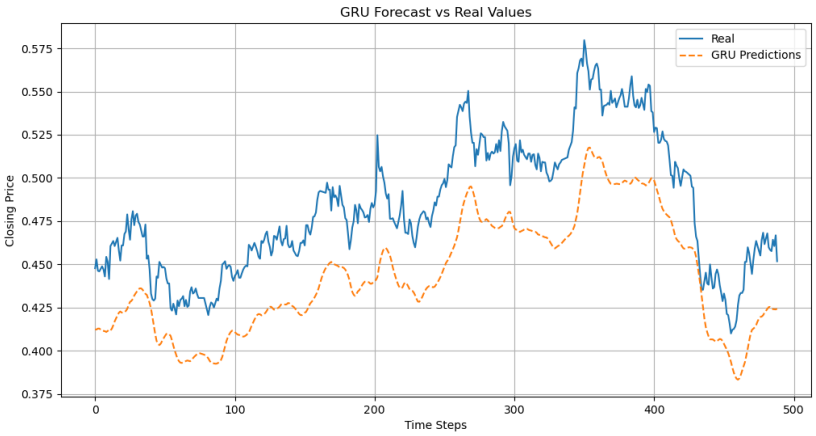

Figure 6: GRU Multivariate Forecast vs. Actual Values

4.4 TCN (Temporal Convolutional Network)

The TCN model uses dilated temporal convolution blocks to capture temporal dependencies at multiple scales. The `TCN` class stacks these blocks—each composed of two dilated convolution layers, dropout, and ReLU activation—and then applies a linear layer on the output of the last time step to generate the forecast.

4.4.1 Advantages:

- *Multi-scale capture:* Dilated convolutions allow the model to capture both short- and long-term dependencies without relying on recurrent mechanisms.
- *Parallelization:* Convolution operations can be parallelized, reducing training time compared to recurrent models.
- *Gradient stability:* The convolutional architecture helps mitigate issues related to exploding or vanishing gradients.

4.4.2 Disadvantages:

- *Hyperparameter sensitivity:* The performance of the TCN is sensitive to the kernel size, number of layers, dilation factor, and dropout rate, which require careful tuning.
- *Interpretability:* The convolution and dilation operations can make the model's inner workings less intuitive compared to classical linear models.

4.4.3 Implementation and Training:

The implementation of the TCN model involved the following steps:

1. **Data Preparation:** The time series corresponding to the column was extracted (either raw or normalized) and divided into fixed-length sequences (e.g., 10 time steps) using a dedicated function.

2. **Model Architecture:** The TCN is built by stacking several temporal blocks. Each block applies two dilated convolutions (with padding calculated to preserve sequence length), followed by ReLU activations and dropout. A residual connection is included to ease gradient propagation. The output from the last block is passed to a linear layer to obtain the final forecast.

3. **Training:** The model was trained for 50 epochs with a learning rate of 0.001 using MSE as the loss function. A progress bar was displayed to monitor training, with loss values reported every 10 epochs.

4. **Evaluation:** After training, the model was evaluated on a test set (e.g., 10% of the data). RMSE and MAE were computed to quantify the overall forecast error, and a custom evaluation was performed to assess trend accuracy and proximity, yielding a global reliability score.

4.4.4 Results for TCN:

Tests conducted on the TCN model (trained on the normalized series) yielded the following metrics:

- **On non-normalized data:**
  - *RMSE:* 8.791
  - *MAE:* 8.354
  - *Trend Accuracy:* 49.28%
- **On normalized data:**
  - *RMSE:* 0.213
  - *MAE:* 0.199
  - *Trend Accuracy:* 51.63%

These results indicate that the TCN effectively captures temporal dependencies at multiple scales and provides competitive forecasts compared to traditional recurrent models. However, the discrepancy between non-normalized and normalized metrics underscores the model's sensitivity to data scaling, suggesting that proper normalization and hyperparameter optimization are critical.

4.4.5 Visualization of TCN Results:

Figure 7 below presents a comparison between the actual values and the forecasts generated by the TCN model. Although the model generally follows the overall trend, some occasional discrepancies highlight areas for further improvement in its predictive capability.

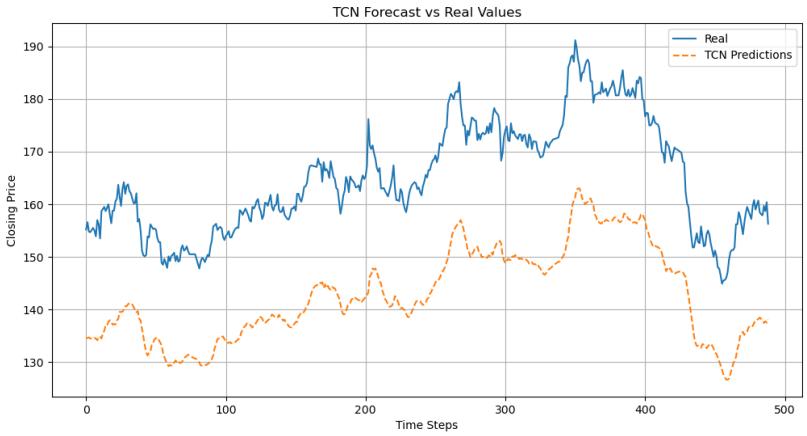

Figure 7: TCN Forecast vs. Actual Values

5 Summary of Results and Preparation of Presentation Materials

This section synthesizes all the results obtained during the training of the different prediction models (ARIMA, LSTM, Multivariate GRU, and TCN) and describes the preparation of presentation materials designed to communicate these results clearly and accessibly.

5.1 Summary of Results

The different prediction models were evaluated using classic metrics such as RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error), as well as a custom evaluation of trend accuracy (ability to correctly predict the direction of daily changes) and proximity between forecasts and actual values. The key results are as follows:

- **ARIMA:**
  - *RMSE:* 1.71
  - *MAE:* 1.32
  - *Trend Accuracy:* between 54% and 63% (depending on the train/test split)
- **LSTM (Univariate):**
  - On non-normalized data: RMSE = 159.84, MAE = 159.49, Trend Accuracy = 47.74%
  - On normalized data: RMSE = 0.14, MAE = 0.14, Trend Accuracy = 49.59%
- **Multivariate GRU:**
  - On non-normalized data: RMSE = 155.58, MAE = 155.22, Trend Accuracy = 48.77%
  - On normalized data: RMSE = 0.017, MAE = 0.014, Trend Accuracy = 48.15%
- **TCN:**
  - On non-normalized data: RMSE = 8.79, MAE = 8.35, Trend Accuracy = 49.28%
  - On normalized data: RMSE = 0.213, MAE = 0.199, Trend Accuracy = 51.63%

Overall, for our project, the ARIMA model emerged as the best solution. Its performance (with an RMSE of 1.71 and MAE of 1.32) demonstrates robust forecasting ability on the original scale of the data. Additionally, a trend accuracy between 54% and 63% indicates a relatively satisfactory capacity to anticipate the direction of daily changes. In comparison, while neural network-based models (LSTM, multivariate GRU, and TCN) theoretically offer improved capabilities for modeling non-linear relationships, in our case their performance on raw data is very low (almost negligible). These weak results seem to be linked to an excessive sensitivity to normalization steps and overfitting on the normalized scale, thus limiting their ability to provide reliable forecasts on the original scale.

5.2 Visualization of Results

The generated graphs illustrate the comparison between actual values and forecasts for each model. They show that, despite some occasional deviations, the ARIMA model provides more consistent and robust forecasts, whereas the neural network-based models struggle to yield significant results on the raw scale. These visualizations confirm the suitability of ARIMA for our project while highlighting the limitations of the other approaches in this specific case study.

5.3 Improvement Opportunities for Each Model

To further optimize performance and robustness, several improvement avenues have been identified for each approach:

- **ARIMA:**
  - *Incorporation of exogenous variables:* Transitioning to an ARIMAX model could allow the integration of external factors (such as macroeconomic indicators or sector signals) to improve forecasts.
  - *Seasonal models:* Utilizing SARIMA could better capture periodic variations present in financial data.
  - *Optimization of train/test split:* Experimenting with different ratios and error weighting techniques might reduce the accumulation of errors in multi-step forecasts.
- **LSTM (Univariate):**
  - *Deeper or bidirectional architecture:* Exploring deeper networks or bidirectional LSTMs may better capture complex temporal dependencies.
  - *Attention mechanisms:* Incorporating attention mechanisms might help the model focus on key periods, potentially improving forecasting accuracy.
  - *Hyperparameter optimization:* Fine-tuning the number of neurons, learning rate, and regularization (dropout, L2) could enhance generalization, particularly for volatile time series.
- **Multivariate GRU:**
  - *Feature selection and engineering:* Enhancing the quality of the features (e.g., by incorporating additional technical indicators) could fully exploit available information.
  - *Fine-tuning hyperparameters:* Adjusting the hidden layer size, number of layers, and gradient clipping can further stabilize training.
  - *Increased regularization:* Applying additional regularization techniques, such as dropout or L2 regularization, may help prevent overfitting.
- **TCN:**
  - *Varied architectures and kernel sizes:* Experimenting with deeper architectures and different kernel sizes could better capture dependencies at various scales.

- *Optimization of dilation factor and dropout:* Fine-tuning these parameters is essential to balance model flexibility and stability.
- *Enhanced residual connections:* Strengthening residual connections and applying further normalization techniques may ease convergence and improve long-horizon forecasts.

6 Global Conclusion

This project posed a significant challenge both technically and conceptually. The implementation and evaluation of various prediction models—from classical approaches like ARIMA to advanced neural network techniques (LSTM, multivariate GRU, TCN)—required extensive research to identify and understand appropriate AI models for time series, as well as their respective advantages and limitations.

Moreover, implementing these models involved several difficulties. The complexity of data preprocessing (including normalization, sequence creation, and train/test splitting), along with understanding and interpreting the generated graphs, were particularly demanding steps. Adjusting hyperparameters and optimizing architectures, especially for recurrent models and temporal convolutional networks, also required a considerable investment in time and expertise.

Despite these obstacles, the project enabled the development of a robust comparative approach to different forecasting techniques. Each model provided complementary insights into the dynamics of financial time series while highlighting trade-offs between complexity, interpretability, and predictive performance.

Ultimately, this work not only enhanced my skills in predictive modeling and time series analysis but also underscored the importance of clear communication of results. Preparing well-structured presentation materials proved indispensable for conveying complex technical concepts and facilitating data-driven decision-making.

In summary, this project illustrates the richness and diversity of AI approaches applied to time series forecasting, while also highlighting the inherent challenges in their implementation and interpretation.

7 Code Organization and GitHub Repository Structure

To facilitate maintenance and readability, the code is organized into several files, each dedicated to a specific step of the forecasting process. Below is an overview of the main files and their functions:

- **data_preprocessing.py**:
  - *load_and_preprocess_data*: Loads data from a CSV file, converts the date column to datetime, cleans the data (notably converting the column), and reindexes the DataFrame to include all business days.
  - *normalize_data*: Normalizes the specified columns using MinMaxScaler.
  - *create_sequences*: Splits a time series into fixed-length sequences for model input preparation.
  - *visualisation*: Generates exploratory data analysis (EDA) graphs such as trends, distributions, and monthly variations.
  - *information*: Provides descriptive statistics and identifies missing dates.
  - *convert_volume*: Converts volume values (e.g., "79.15M") into absolute numeric values.
  - *plot_forecast*: Displays forecast curves compared to actual values.
  - *adjust_monthly_volatility*: Adjusts monthly volatility by reducing abnormal peaks to smooth the series.
- **model_arima.py**:
  - Implements the ARIMA model.
  - *rolling_forecast_arima*: Performs rolling forecasts by retraining the ARIMA model at each iteration.
- **model_lstm.py**:
  - Defines the `LSTMForecast` class that implements a univariate LSTM model.
  - *train_and_evaluate*: Trains the LSTM model and returns predictions along with evaluation metrics (RMSE, MAE).
- **model_gru.py**:
  - Defines the `GRUMultivariate` class for the multivariate GRU model.
  - Also contains a *create_sequences* function (similar to that in data_preprocessing.py) and the *train_and_evaluate_gru* function for training and evaluating the model.
- **model_tcn.py**:
  - Contains the `TemporalBlock` class, which defines a dilated convolution block with padding and dropout.
  - Defines the `TCN` class that stacks multiple TemporalBlocks and applies a linear layer to generate the forecast.

  - Includes the *create_sequences* and *train_and_evaluate_tcn* functions for training and evaluating the TCN model.

- **evaluation.py**:
  - Contains the *evaluate_forecast* function which computes custom evaluation metrics such as trend accuracy and proximity.
  - Also computes classical metrics like RMSE and MAE.

- **main.py**:
  - Orchestrates the execution of all parts of the project: data loading, preprocessing, training and evaluation of each model (ARIMA, LSTM, GRU, TCN), and graph generation.
  - Coordinates the function calls from the other files to produce the final results.

- **stock_price.csv**:
  - The raw data file containing stock information (e.g., , , , , , etc.).

Each file is carefully commented to explain the role of the different functions and to facilitate code maintenance and extension.

8 Appendices

Final Remarks

- The GitHub repository will include all the files listed above, along with a detailed README explaining how to run the code and reproduce the results.
- The presentation slides will build on this report to illustrate the complete process—from data understanding to model validation.