POLITÉCNICA

UNIVERSIDAD POLITÉCNICA DE MADRID

S&P500 Time Series Forecasting

Data Mining & Time Series Project

Academic year 2023-2024

Authors:
Nicola Cecere
Francesco Mattioli
Luca Petracca

Professors:
Juan Pedro Caraca-Valente Hernandez
Aurora Perez Perez

Contents

1 Introduction **1**
1.1 Project Description . . . 1
1.2 Domain . . . 1
1.3 Dataset Overview . . . 1
1.4 Project Goal . . . 1

2 Data Visualization & Exploration **2**
2.1 Data Visualization . . . 2
2.2 Data Exploration . . . 3
2.3 Missing Values . . . 5
2.4 Fourier Transformation . . . 5
2.5 Autocorrelation . . . 7
2.6 Partial Autocorrelation . . . 8
2.7 Stationarity Analysis . . . 9
2.7.1 Project Application . . . 11

3 Data Preparation **12**
3.1 Data Cleaning . . . 12
3.2 Data Splitting . . . 13

4 Outlier Analysis **13**
4.1 Outlier Detection in Financial Metrics . . . 13

5 Feature Engineering **15**
5.1 Identifying Seasonality Patterns . . . 15
5.2 Encoding Seasonality . . . 16
5.3 Lag Features . . . 17
5.4 Clustering . . . 17
5.5 Moving Average . . . 17
5.5.1 Project Application . . . 18

6 Stock Price Prediction **19**
6.1 LSTM . . . 19
6.2 Data Preprocessing and Feature Engineering . . . 19
6.3 LSTM Model Architecture . . . 20
6.4 Model Architecture Details . . . 20
6.5 Training and Evaluation . . . 20
6.6 Evaluation of Model Performance . . . 21
6.7 Model Generalization Across Different Stocks . . . 22
6.8 Experimentation with a Hybrid CNN-LSTM Model . . . 23

7 Conclusion **25**

References **26**

1 Introduction

1.1 Project Description

This project focuses on extracting insights from time-series data using data mining techniques. It involves selecting an appropriate dataset, cleaning and preparing the data, and then applying various analysis methods. The key goal is to uncover patterns and relationships in the data, which can inform decision-making or predictions. The approach encompasses a range of techniques, from basic statistical analysis to complex machine learning models, emphasizing the practical application of data mining in understanding and extracting knowledge from time-series data.

1.2 Domain

In the context of this project, the chosen dataset is from the S&P 500 stock market index [3]. The S&P 500, short for Standard & Poor's 500, is a stock market index that measures the performance of 500 large publicly traded companies in the United States. It is considered a benchmark for the overall performance of the U.S. stock market. It is widely followed by investors, financial professionals, and analysts as an indicator of the health and direction of the U.S. stock market. The dataset includes:

- **Time Series Data**: Daily closing prices, trading volumes, and other financial indicators.
- **Volatility and Trends**: The data reflects the market's dynamic nature, influenced by economic, political, and global events.
- **Domain-Specific Challenges**: Handling the inherent volatility and unpredictability of financial markets, as well as the implications of missing or irregular data points.

1.3 Dataset Overview

The dataset is composed by four folders with a total of 9210 files from Frobes2000, NASDAQ, Nyse and S&P500. We've decided to focus our attention only on the S&P500 index. For what concern this index, the dataset provides 409 csv files, one for each company. Each file is characterized by the following features:

- **Date**: date in the format yyyy-mm-dd
- **Volume**: the total number of shares traded in a day
- **High**: the highest price at which a stock is traded during a day
- **Low**: the lowest price at which a stock is traded during a day
- **Open**: tells the price at which a stock started trading when the market opened during a day
- **Close**: the price traders agreed on after all the action throughout the day
- **Adjusted Close**: while the closing price simply refers to the cost of shares at the end of the day, the adjusted closing price takes dividends, stock splits, and new stock offerings into account.

1.4 Project Goal

For our project, we decided to focus our attention only on the Apple stock market data, identified by the ticker 'AAPL'. Our goal is to predict the close price of the stock for the following day, utilizing data from the preceding five days. Our objective is to develop a predictive model that effectively interprets historical data, enabling it to accurately forecast the stock's movement in the coming days. This insight is particularly valuable for investors and market analysts, aiding them in making strategic investment choices and understanding market dynamics.

2 Data Visualization & Exploration

In this chapter, we delve into the exploring and understanding of the dataset, which is a crucial first step in our project. Data visualization and exploratory analysis are the focus of this phase, as they are essential tools for identifying underlying patterns, trends, and characteristics in the data. Through the use of a variety of visualization techniques, we want to gain a thorough understanding of the structure and behavior of the data, which will help to direct our preprocessing steps going forward.

2.1 Data Visualization

In the data visualization phase of our project, we employed various functions to visually analyze the stock market data, focusing on specific tickers. This approach enabled a detailed examination of individual stocks, providing insights into their behavior and characteristics. Here's a deeper look at each type of visualization we implemented:

- **Plotting a Selected Column Over Time for a Selected Ticker**: This function is crucial for understanding the time-series nature of stock market data. By selecting a specific ticker and a particular column (such as 'Close', 'Volume', etc.), we can plot how this attribute has changed over time. This step is not just about tracing patterns and trends but is crucial for initial data quality checks. By visually inspecting these plots, we can quickly spot unusual spikes or drops, which might indicate potential errors or abnormal market activities [1]. It's an essential first step in ensuring the data's integrity before moving on to more detailed outlier analysis and further modeling.

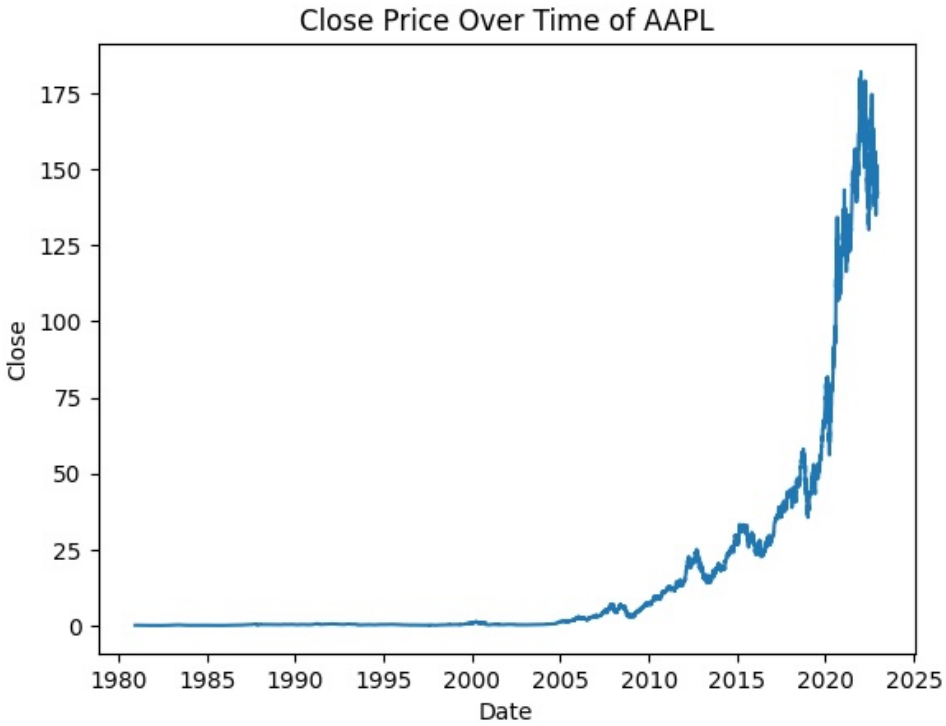


Figure 1: AAPL 'Close' trend over the time.

- **Printing Feature Relationships for a Selected Ticker**: This function allows us to explore the relationships between different features of a particular stock. For instance, we might look at how the 'Open' and 'Close' prices relate to each other, or how 'Volume' changes with respect to 'High' prices. An example of Apple is in Fig 2.

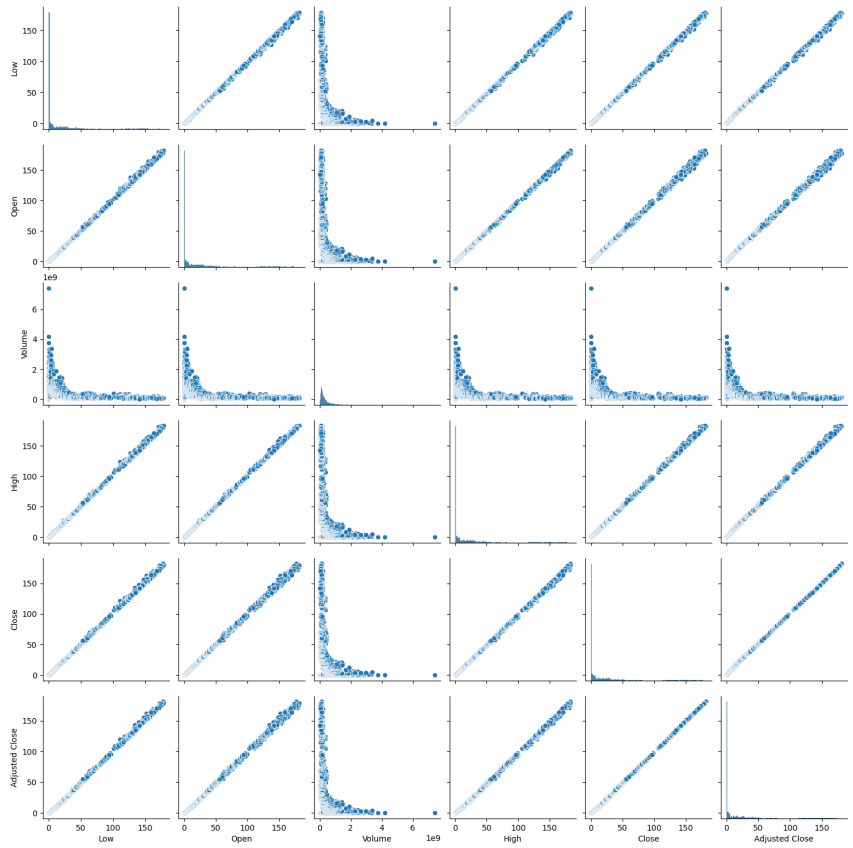


Figure 2: AAPL Features Relationship

- **Correlation Matrix for a Selected Ticker**: The correlation matrix is a powerful tool for understanding how different aspects of a stock's behavior are interrelated. By focusing on a single ticker, we can see how closely connected features like 'Close', 'High', 'Low', 'Open', and 'Volume' are. After examining the correlation matrix for the stock market dataset, we've noticed numerous instances of a correlation coefficient of 1, Fig 3, indicating a perfect linear relationship between various features. This high correlation is particularly prevalent among the 'Open', 'High', 'Low', and 'Close' prices of stocks. Such a pattern typically arises because these prices within a single trading day are inherently interrelated. For instance, the 'High' and 'Low' prices are within the range set by the 'Open' and 'Close' prices. Market dynamics, such as investor reactions to news events or changes in market sentiment, tend to impact these prices simultaneously, leading to their closely aligned movements. This phenomenon reflects the integrated nature of stock market prices and is a crucial aspect to consider in financial time series analysis.

2.2 Data Exploration

During the data exploration stage, we set out to fully comprehend each aspect of our stock market dataset. Plotting the distribution of records across several years was the first thing we did. The plot in Fig 4 showed any notable changes in data density over the years and gave insight into the distribution of data entries over time.

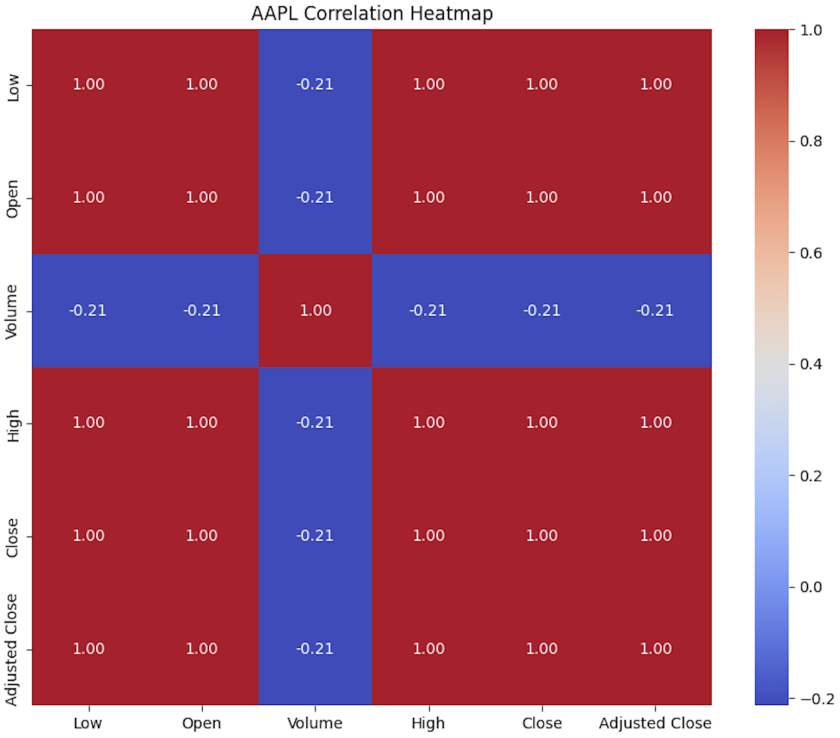


Figure 3: AAPL Correlation Heatmap

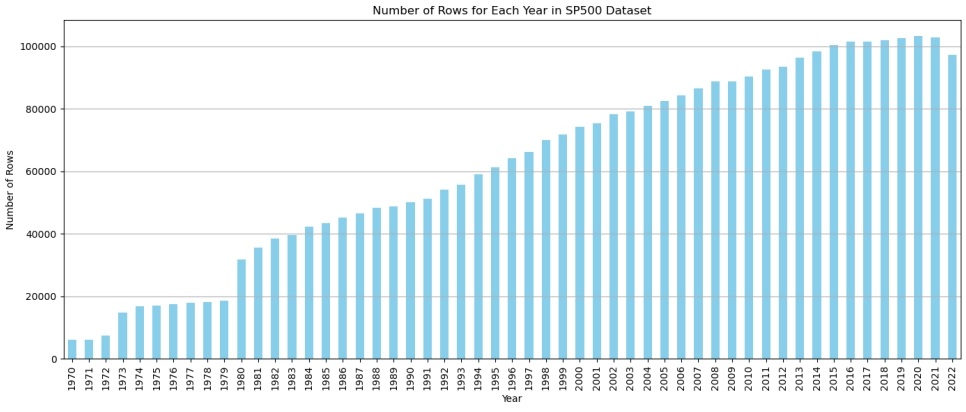


Figure 4: Number of records for each year in the dataset

Next, we delved into the range of values within the dataset by extracting and examining the maximum 'Close' value for each ticker symbol. This exploration was vital in understanding the scale and variability of stock prices among different companies. This kind of analysis prepared the way for thinking about data normalization in later stages, guaranteeing consistency and comparability among different stock tickers. We found out that the Interval is [0.5500, 539180.0000] and the range is 539179.4500. Finally, we turned our attention to the identification and analysis of missing values within the dataset. The analysis of the missing values will be deepened in the section 2.3.

2.3 Missing Values

In our data mining project, a critical step involved the meticulous analysis of missing values within our dataset. Initially, our focus was directed towards identifying tickers that contained missing data. Through this process, we discovered that 11 tickers had incomplete information, Fig 5. Subsequently, we conducted a more granular analysis for each of these tickers to determine the overall percentage of missing data they contained.

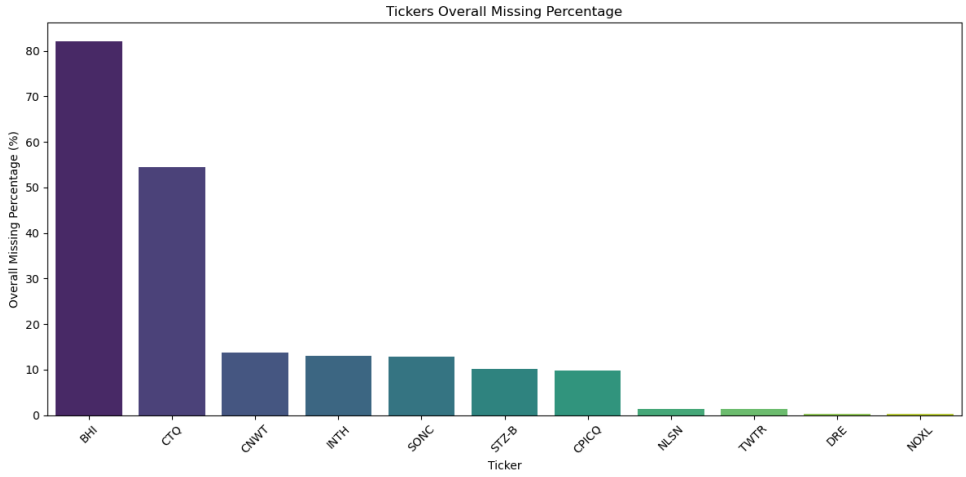


Figure 5: Tickers Overall Missing Percentage

This examination led to the revelation that two tickers exhibited exceptionally high levels of missing data, exceeding 80%. Given the substantial gaps in information, we made the strategic decision to exclude these two tickers from our dataset. This step was crucial to ensure the integrity and reliability of our subsequent analyses.

Moreover, we extended our exploration by generating a file that highlighted the percentage of missing data for each ticker on an annual basis. This analysis was instrumental in identifying specific years that were particularly affected by data gaps. The insights gained, as shown in Fig 6, proved invaluable when it came time to segment our dataset for further analysis, allowing us to make informed decisions on how to handle these missing values effectively in the data cleaning process, analyzed in section 3.1.

2.4 Fourier Transformation

In our analysis, we transformed the closing price data from the time domain into the frequency domain using the Fast Fourier Transform (FFT) method. This mathematical transformation allowed us to decompose the time series into its constituent frequencies, revealing any periodic patterns or underlying trends in the stock price movements. By plotting the frequencies against their amplitudes on a logarithmic scale, we were able to discern the most significant cycles that characterize the stock's behavior. The resulting visualization, shown in Fig 7, served as a powerful tool for identifying dominant frequencies, which correspond to regular fluctuations in the stock's closing price, potentially informing our forecasting models.

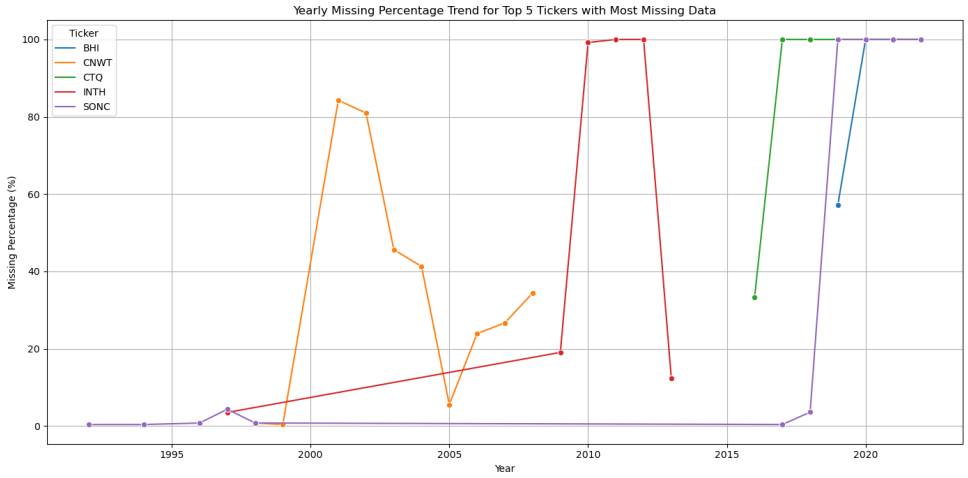


Figure 6: Yearly Missing Percentage for each Ticker

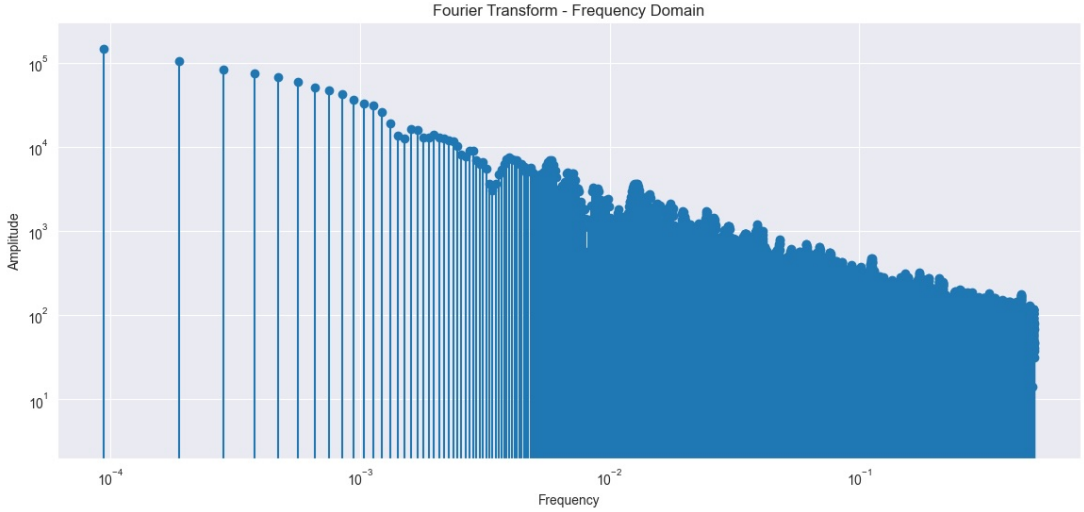


Figure 7: Fourier Transform - Frequency Domain

We filtered these frequencies to focus on those that correspond to time periods ranging from five days to approximately a trading year (252 days), as shown in Fig 8. This helped us isolate the most relevant cycles impacting stock movements. The identified significant frequencies were then inverted to time periods, giving us a clear view of the cyclic patterns that could be critical for predicting future price movements.

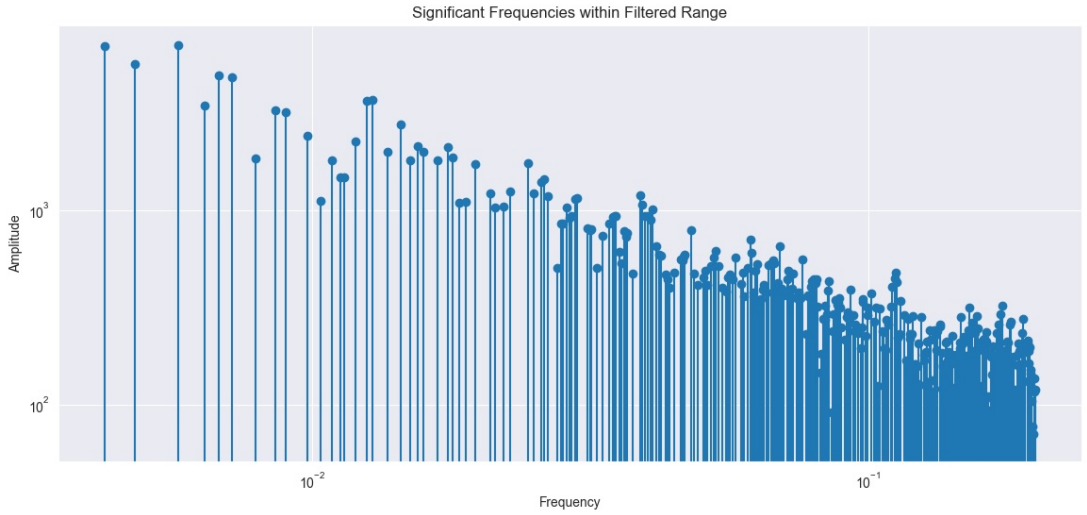


Figure 8: Significant frequencies in range

We approximated the stock's price by using a limited number of frequency components to reconstruct the time series. Fig 9 shows how the reconstructed signals with 5, 10, and 25 frequency components were compared with the actual closing prices to reveal the underlying patterns and trends in the data.

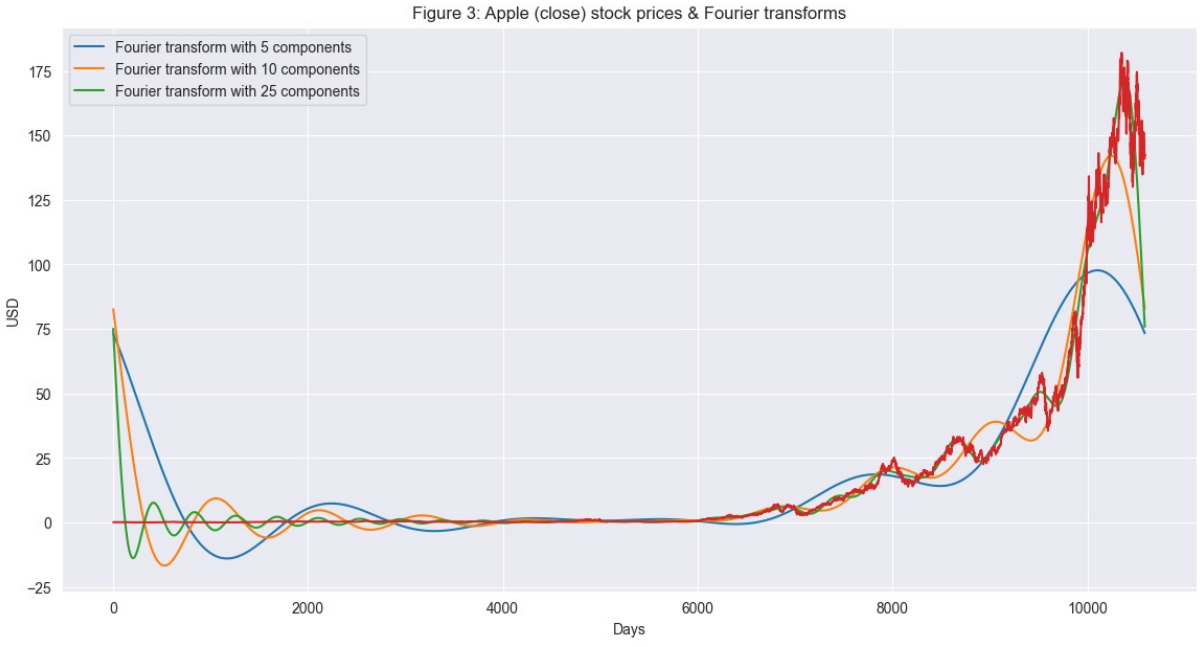


Figure 9: Fourier Transform - Apple close stock prices

2.5 Autocorrelation

Autocorrelation measures the linear relationship between lagged values of a time series. If a series is significantly autocorrelated, that means, the previous values of the series (lags) may be helpful in predicting the current value. Setting 'Date' as the index and resampling to monthly averages is essential for meaningful autocorrelation analysis, to understand whether previous monthly 'Close' values are correlated to the future ones. We generated the autocorrelation plots for the monthly average data,

showing correlations at different lags. The first plot in Fig 10 displays long-term autocorrelation (up to 90 lags), while the second focuses on short-term (up to 30 lags).

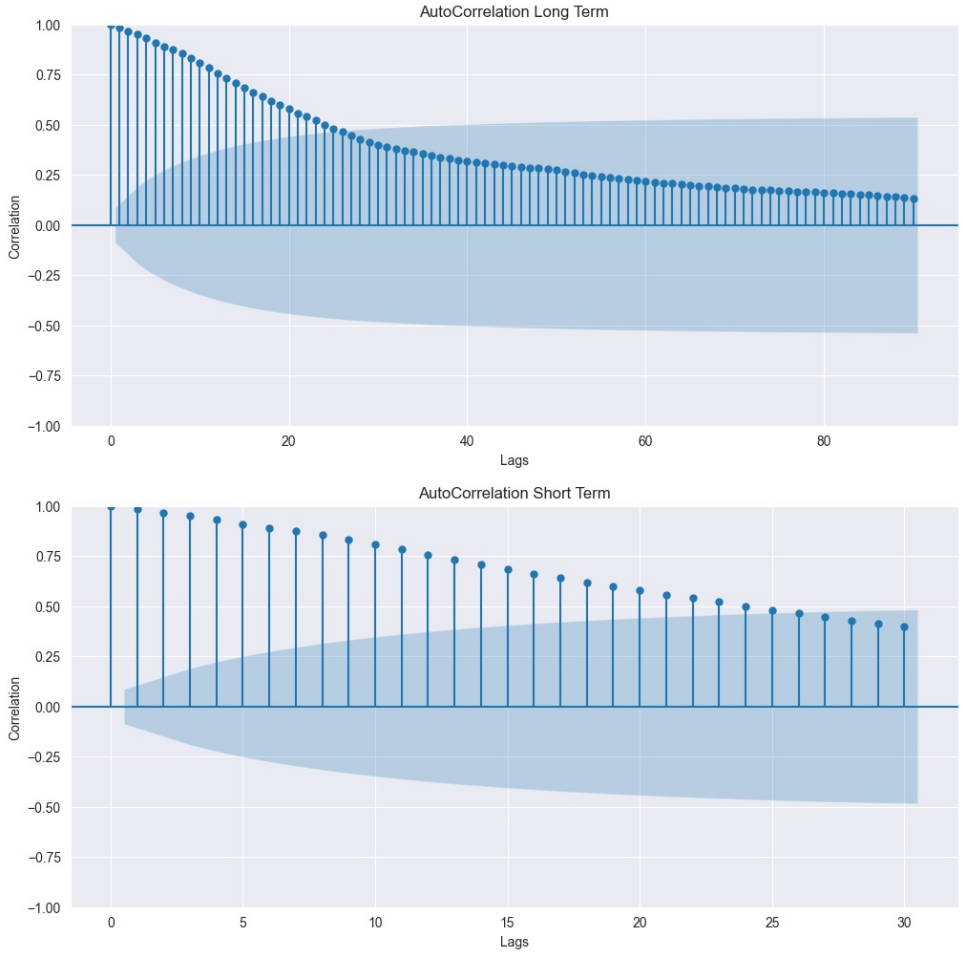


Figure 10: Autocorrelation Long Term - Short Term

The autocorrelation plots demonstrate a gradual decay in correlation as the number of lags increases, indicating a slow decline in the relationship over time. This suggests that past values have a persisting influence on future values, characteristic of a non-stationary time series. This shows that the series is not random and good for time series modeling.

2.6 Partial Autocorrelation

Partial autocorrelations measure the linear dependence of one variable after removing the effect of other variables(s) that affect both variables.

The key difference between the autocorrelation (ACF) and partial autocorrelation (PACF) plots lies in the relationships they measure. The ACF plot shows the total correlation between a time series and its lagged values, including indirect effects. In contrast, the PACF plot isolates the direct correlation at each lag, removing the influence of correlations at shorter lags.

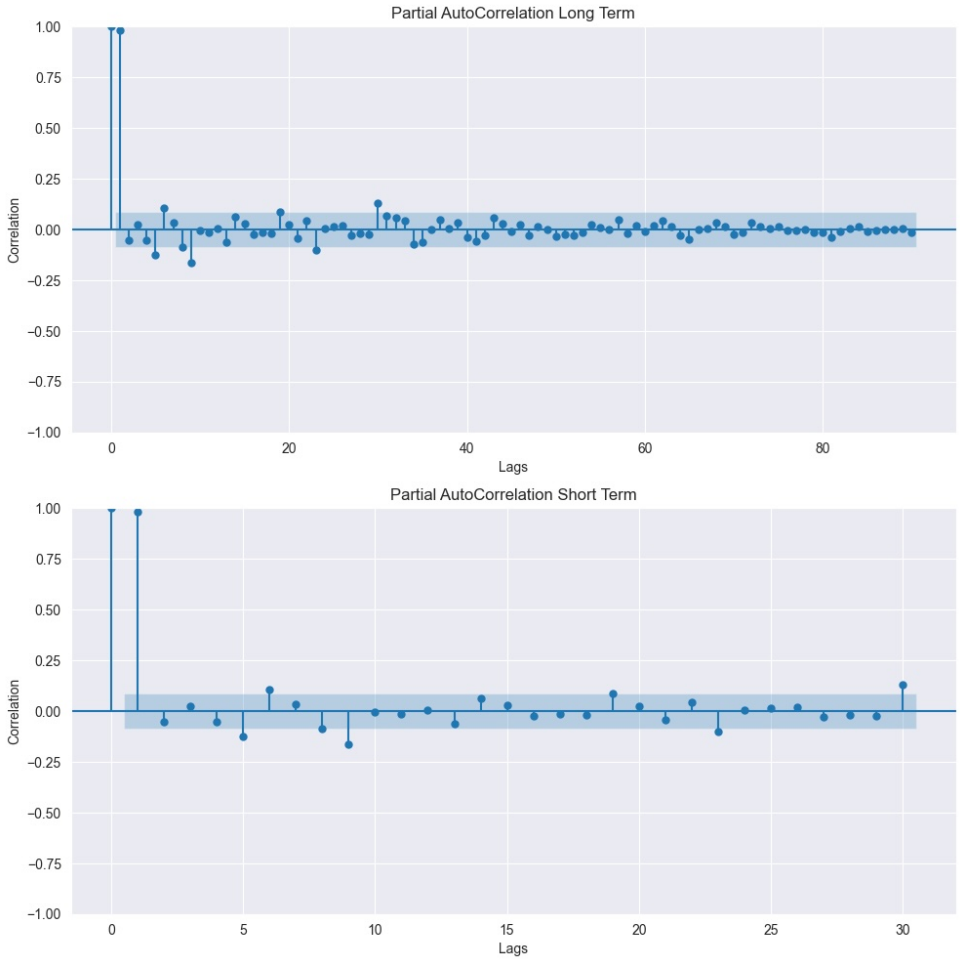


Figure 11: Partial Autocorrelation Long Term - Short Term

The partial autocorrelation plots in Fig 11 show a significant direct correlation at the first lag for both long-term and short-term views, which quickly diminishes and becomes statistically insignificant for subsequent lags, suggesting a potential AR(1) process where only the immediate past value has a direct impact on the current value.

2.7 Stationarity Analysis

Stationarity describes the concept that how a time series is changing will remain the same in the future. In mathematical terms, a time series is stationary when its statistical properties are independent of time:

- Constant mean
- Constant variance
- Covariance is independent of time

Some time series forecasting models (e.g., autoregressive models) require a stationary time series because they are easier to model due to their constant statistical properties.

You can test a time series for stationarity in two ways:

- Intuitive approach: Visual assessment
- Statistical approach: Unit root test

To visually assess the stationarity of a time series by dividing the time series in half and comparing the mean, amplitude, and cycle length from the first half to the second half of the time series, Fig. 12.

- Constant mean: The mean value of the first half of the time series should be similar to that of the second half.
- Constant variance: The amplitude of the first half of the time series should be similar to that of the second half.
- Covariance is independent of time: The cycle length in the first half of the time series should be similar to that in the second half. The cycles should be independent on time (e.g., not weekly or monthly, etc.).

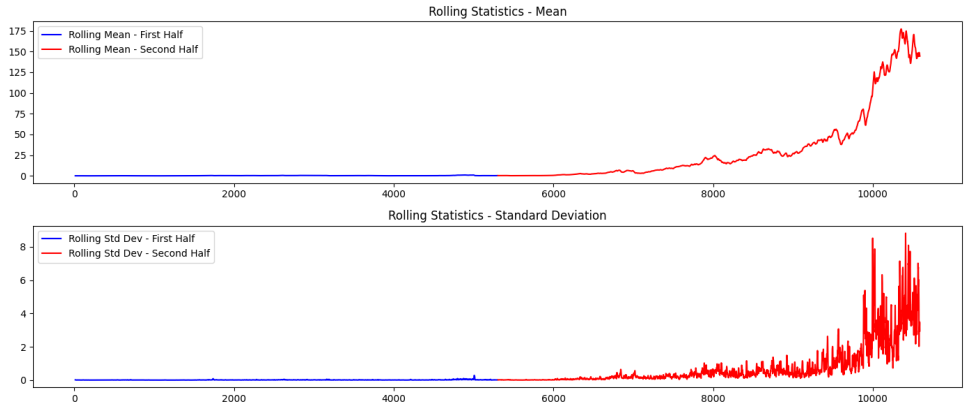


Figure 12: Visual check for stationarity

To statistically assess the stationarity we use two unit root tests.

Augmented Dickey-Fuller test

The hypotheses for the Augmented Dickey-Fuller (ADF) test are:

- *Null hypothesis (H0)*: The time series is not stationary because there is a unit root (if p-value > 0.05)
- *Alternative hypothesis (H1)*: The time series is stationary because there is no unit root (if p-value $<= 0.05$)

The time series is stationary if we can reject the null hypothesis of the ADF test.

Kwiatkowski-Phillips-Schmidt-Shin test

The hypotheses for the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test are:

- *Null hypothesis (H0)*: The time series is stationary because there is no unit root (if p-value > 0.05)
- *Alternative hypothesis (H1)*: The time series is not stationary because there is a unit root (if p-value $<= 0.05$)

The time series is stationary if we fail to reject the null hypothesis of the KPSS.

Make a time series stationary

Different transformations can be applied to a non-stationary time series to try to make it stationary:

- Differencing
- Fractional Differencing
- Detrending by model fitting
- Log transformation

Once applied one of these transformations ADF and KPSS can be applied again to check for stationarity.

2.7.1 Project Application

We started by visually assess the stationarity on the AAPL stock by plotting the rolling mean and the rolling standard deviation for the first and second half of the 'Close'. The dataset's first half appears stationary with a steady mean and low, consistent standard deviation. In the second half, however, the mean trends upwards and the standard deviation increases, indicating the data's statistical properties are changing over time. This shift in mean and variance suggests the second half of the dataset is not stationary, reflecting an underlying change in the data-generating process. Stationary time series models may not suit this data due to these changes.

We assessed the stationarity of our dataset using the Augmented Dickey-Fuller (ADF) test and the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test. The ADF test yielded a statistic of 2.301392 with a p-value of 0.998956, suggesting that we cannot reject the null hypothesis of a unit root presence; this implies non-stationarity.

Complementing this, the KPSS test produced a statistic of 7.9526701498163845, significantly beyond the critical values across all significance levels, with a p-value of 0.01. This result strongly rejects the null hypothesis of stationarity, indicating a trend or drift in the time series.

The ADF test suggested that the time series has a unit root, thus it is non-stationary. In contrast, the KPSS test indicated that the series is not trend-stationary. When both tests are considered together, they point to a non-stationary series with a stochastic trend, confirming that the time series in our project does not have a constant mean or variance over time.

Upon discovering that our time series was non-stationary, we tried to apply fractional differencing. Fractional differencing is a technique that allows us to transform our data into a stationary series while retaining the maximum amount of information from the original time series. This method is particularly advantageous as it provides a balance between differencing enough to achieve stationarity and preserving the memory of the series as shown in Fig. 13.

After applying fractional differencing to our dataset, we conducted the Augmented Dickey-Fuller (ADF) test and the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test once more to evaluate the impact of our transformation.

The results post-transformation were promising. The ADF test showed a significant decrease in the test statistic, well below the critical value thresholds, and the p-value fell, suggesting we could reject the null hypothesis of a unit root. Concurrently, the KPSS test statistic also fell within the range of stationarity, with a higher p-value indicating that we could not reject the null hypothesis of stationarity.

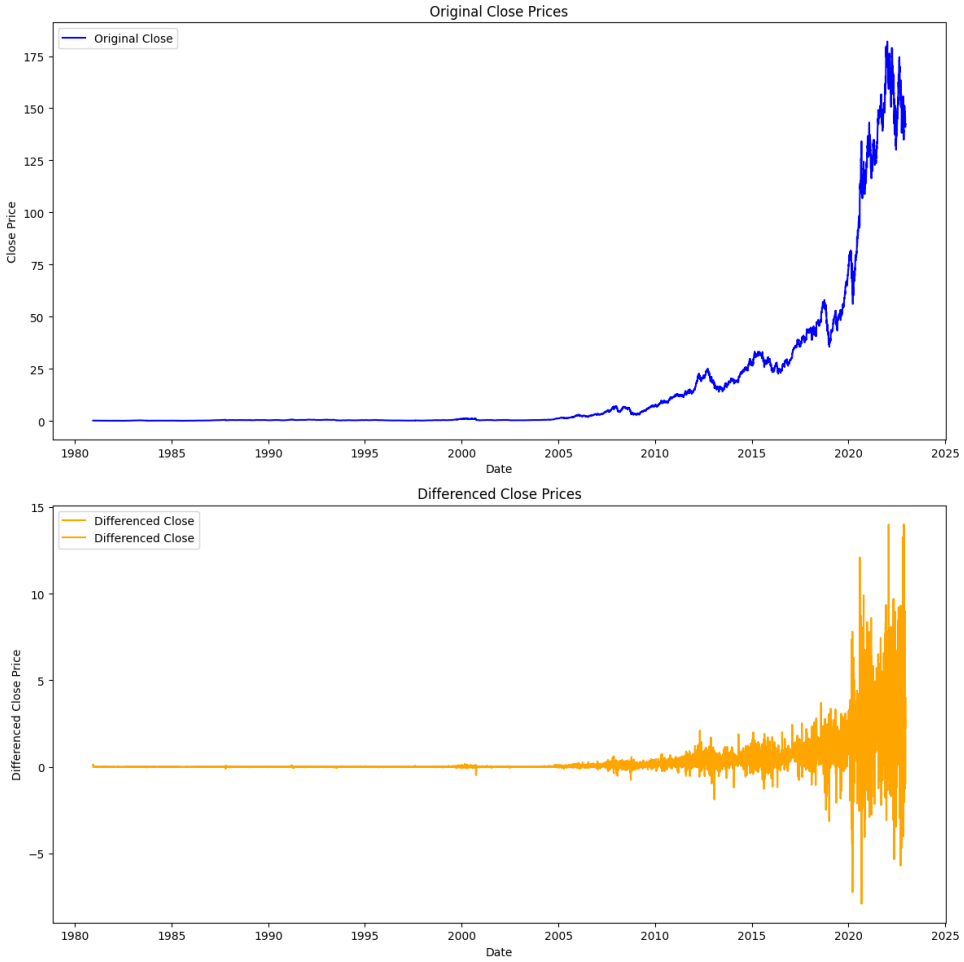


Figure 13: Non stationary and Stationary 'Close' for AAPL

We've done this to try using ARIMA models, but then we decided to not use it because of our project goals since we were using 5 days batches to predict the next one, while ARIMA model works better without data batches.

3 Data Preparation

3.1 Data Cleaning

In the data cleaning phase of our project, our initial step was to remove the 'Adjusted Close' column from our dataset. This decision was based on the fact that our analysis focused on the 'Close' column, rendering 'Adjusted Close' redundant for our purposes.

As we delved deeper into handling missing values, our strategy was tailored to the specific characteristics of our dataset. We identified four tickers that lacked data for a significant portion of the period post-2019. Since our dataset was split at the year 2019, with subsequent years forming our test set, the absence of data in these years for these tickers rendered them impractical for our analysis. Consequently, we decided to exclude these tickers from our dataset.

Our approach to addressing missing values was multifaceted. The detailed report we previously generated, outlining the yearly percentage of missing values for each ticker, presented us with a spectrum of scenarios. Some years exhibited minimal missing data, while others were characterized by substantial gaps.

For years with relatively few missing values, we contemplated various interpolation techniques. Linear interpolation was an option, offering a straightforward method to fill gaps by drawing straight lines between existing data points. Polynomial interpolation, a more complex approach, could provide a better fit for data exhibiting non-linear trends. Gaussian interpolation was another consideration, potentially offering a sophisticated method to model data with normal distribution characteristics.

However, before implementing any interpolation technique, we sought to validate the authenticity of these missing values. Our investigation extended to online sources [2], comparing our dataset against actual market history. This comparison revealed a crucial insight: the missing values and records in our dataset precisely matched real market occurrences.

Given this discovery, we opted for a more conservative approach. Rather than interpolating potentially non-existent values, we chose to simply remove all instances of missing data. This decision ensured that our dataset accurately reflected the realities of the market, maintaining the integrity and authenticity of our analysis.

3.2 Data Splitting

The process of splitting the dataset into training, validation, and test sets is a critical step to ensure the effectiveness and generalizability of the model. Given the large size of our dataset, we adopted a strategic approach to data division, aiming to balance the need for sufficient training data against the necessity for robust validation and testing.

We divided the dataset into two primary segments: 90% for the combined training and validation set, and 10% for the test set. The final split of the data is in an 80-10-10. This division was designed to allocate a substantial portion of the data for model training and testing, thereby allowing the model to learn and adapt to the complexities of the financial time series data effectively.

In order to prevent data leakage, the dataset was sorted chronologically before splitting. This prevented the use of future information to predict past values.

4 Outlier Analysis

Outlier detection holds a significant place in the analysis of time series datasets, particularly in the context of financial markets like the S&P 500. The primary reason for its importance stems from the potential impact outliers can have on the overall analysis and forecasting models. In financial markets, outliers often represent unusual events or anomalies that can range from typos, market crashes, sudden economic policy changes, to unexpected corporate events like mergers, acquisitions, or earnings surprises. These outliers can drastically distort the typical pattern observed in stock market data, leading to skewed results in predictive modeling and risk assessment.

4.1 Outlier Detection in Financial Metrics

We analyzed the key financial metrics of our dataset: 'Low', 'Open', 'Volume', 'High' and 'Close', and detected outliers using z-score from the scipy library [4], expressed in terms of standard deviations from the mean. We created an empty log to record the details of detected outliers, which was later converted into a DataFrame for easier handling and analysis. This approach, visualized in Fig 14, allowed us to understand the distribution of each financial metric across different stocks and pinpoint specific instances where the market behavior deviated significantly from the norm.

Considering outliers in the whole dataset for all numerical values did not yield meaningful insights due to the inherent comparison of these values against the mean computed over all years. This approach is problematic especially for stocks that have shown significant growth over a long period, such as AAPL (Apple) [1]. For instance, a stock like AAPL, which has grown substantially from 1980 to 2022, would be unfairly compared against a long-term mean, resulting in almost every recent entry being classified as an outlier. To address this, we decided to refine our strategy by checking for outliers in smaller, more manageable batches of 10 years. This approach allows for a more contextually relevant analysis of the

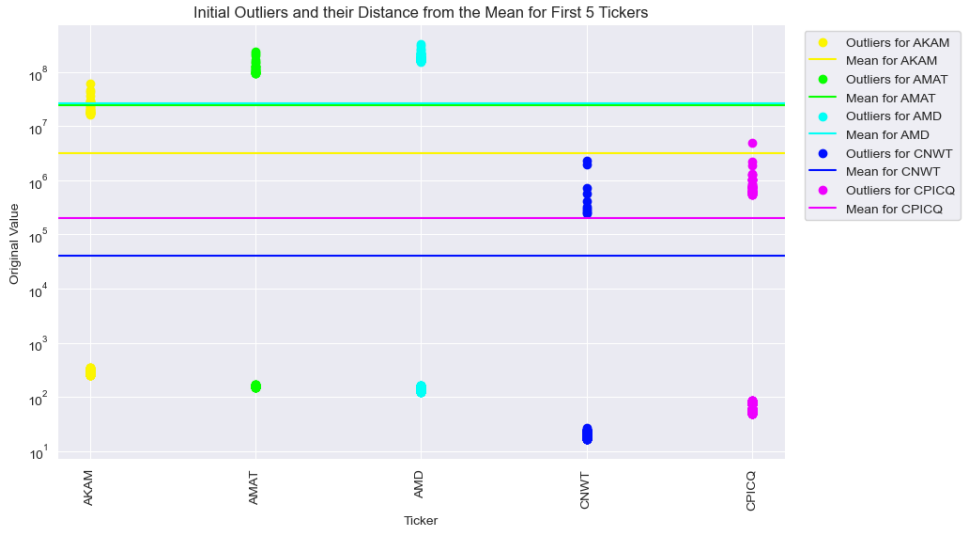


Figure 14: First Outlier Analysis: z-score with coefficient = 5

data, taking into consideration the different market phases and growth trajectories individual stocks have experienced over specific decades, as shown in Fig 15.

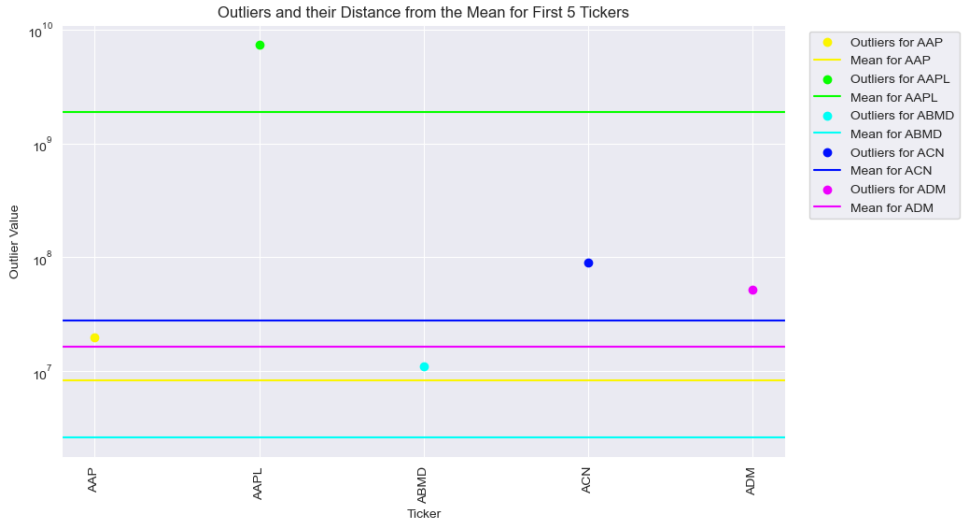


Figure 15: Second Outlier Analysis: z-score with coefficient = 5

The dataset did not have any significant outliers. The last table consisted of 284 possible outliers, which were mostly due to sudden changes in trading volume. We performed a sample check to verify if they corresponded to real stock market data. Based on our methodology, we concluded that these outliers represent actual market behavior. Hence, we chose not to exclude them from our analysis.

5 Feature Engineering

In this chapter, we explore the feature engineering process, which is crucial in predictive modeling, especially within the volatile sphere of stock market data. By creating informative and unique features, we can significantly improve the predictive power of our algorithms, providing more precise and robust forecasts.

5.1 Identifying Seasonality Patterns

Seasonal patterns in the stock market reflect the cyclical nature of the markets, where certain times of the year are more bullish, causing stocks to rise, while others are more bearish, leading to a decline in stock values. Identifying these patterns is crucial in gaining a deeper knowledge about the context and for better feature engineering. To detect these patterns, we plotted the yearly and monthly close price means over the AAPL dataset. To do so, we first extracted the 'Year', 'Month', and 'Day' features from the 'Date' column.

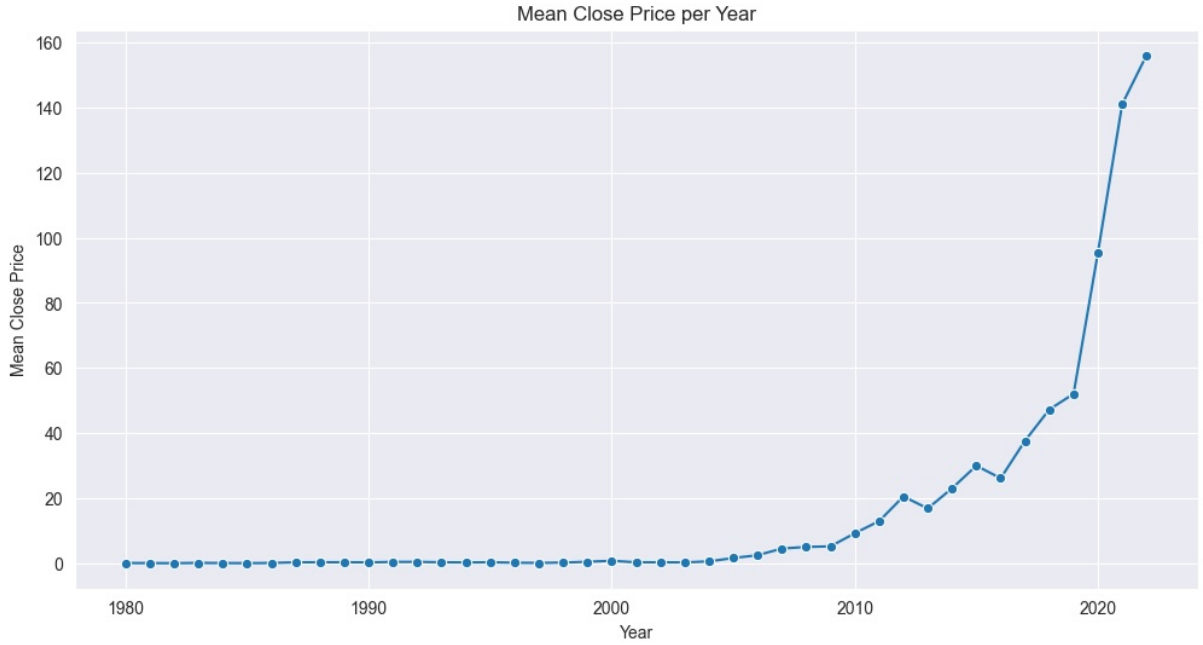


Figure 16: Mean Close Price per Year

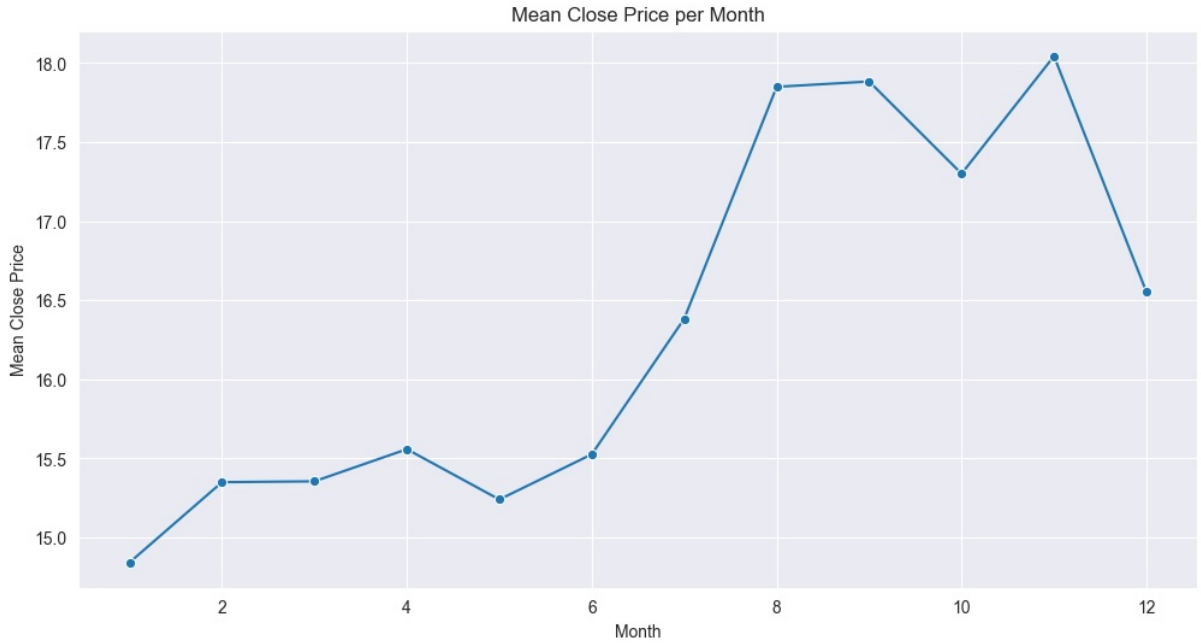


Figure 17: Mean Close Price per Month

The yearly trend graph for Apple stock in Fig 16 demonstrates a remarkable ascending trajectory, with average close prices showing exponential growth from the company's IPO in the 1980s through 2019. In terms of seasonality, our monthly analysis uncovers more nuanced fluctuations. As Fig 17 shows, we observe that average close prices tend to dip from December through January, postulating a bearish market sentiment during these months. Conversely, prices peak around August and September, but reach their highest in November, indicating bullish trends during this late summer to early autumn period.

5.2 Encoding Seasonality

In our feature engineering process, we applied a trigonometric transformation to the date information in our dataset. Specifically, we converted the day of the week into two separate features using the sine and cosine functions. This cyclical encoding captures the repetitive nature of weekly patterns, providing our model with structured temporal information that can be crucial for forecasting. For instance, stock market trends often follow weekly cycles due to regular human activities and institutional operations. By including these sinusoidal time features, our machine learning model can better understand and leverage the inherent periodicity in the stock data, potentially improving predictive performance for Apple's stock price movements.

Then, with our time series analysis of stock market data, a key step involved the encoding of months into three distinct categories, based on the results of the previous analysis: 'Bullish', 'Bearish', and 'Standard'. This categorization was based on historical market performance trends, with certain months typically exhibiting more bullish or bearish behavior than others.

- **Bullish Months:** The months where the mean close price is significantly higher than in other months. The plot shows peak prices around August and September, suggesting these months may generally exhibit bullish behavior for Apple stock.
- **Bearish Months:** The months characterized by a comparatively lower mean close price, indicating a bearish trend. The transition from September to October shows a dip in the mean close price, marking the start of a potentially bearish period.

- **Normal Months:** Months in which the mean close price does not show significant peaks or troughs and is relatively stable. From February through July, the mean close price appears stable without notable highs or lows, suggesting these months may be considered to exhibit a normal market trend for Apple stock.

For effective integration into our predictive models, we utilized one-hot encoding, a standard technique in machine learning for converting categorical data into a numerical format. We represented 'Bullish', 'Bearish' and 'Standard' explicitly adding three columns to the csv. By capturing these seasonal trends, we aimed to enhance our model's capability to recognize and adapt to recurring market patterns, an aspect crucial for accurate time series forecasting in the financial domain.

5.3 Lag Features

In our analysis of the AAPL stock market data, the introduction of lag features was crucial. Lag features provide historical context, which is particularly important in financial markets where past trends and movements can significantly influence future stock prices. Our primary focus was on the 'Close' price, for which we created three lag features. These lag features represent the 'Close' prices of the stock from the previous three days. We handled the initial missing values by filling them with the first available 'Close' value for each ticker. This approach ensured that our model had access to a complete dataset, avoiding the loss of potentially valuable information in the initial days.

5.4 Clustering

In our feature engineering process, we experimented with the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) and K-Means clustering algorithms. DBSCAN is known for its ability to form clusters of arbitrary shapes, while K-Means is effective in partitioning data into K distinct, non-overlapping subgroups.

To evaluate the performance of these clustering algorithms, we employed the Silhouette Score as our primary metric. The Silhouette Score, S, for each sample is calculated as follows:

$$S = \frac{b - a}{\max(a, b)}$$

where a is the mean distance to the points in the nearest cluster that the point is not a part of, and b is the mean distance of all the points in the same cluster. The value ranges from -1 to 1, where a high value indicates that the object is well-matched to its own cluster and poorly matched to neighboring clusters.

For both DBSCAN and K-Means, we conducted extensive hyperparameter tuning using the Optuna framework. This involved over 200 iterations of parameter optimization to find the most effective settings for each model. Through this process, K-Means emerged as the superior algorithm, achieving a higher Silhouette Score of 0.49. This indicated a more distinct and appropriate clustering for our dataset.

Following the clustering, we explored the possibility of enhancing our model by incorporating the cluster labels as a new feature. However, upon further analysis and testing, we found that this addition did not yield any significant improvement in our results. Therefore, we ultimately decided against including the cluster information in our final model.

5.5 Moving Average

In the field of financial forecasting, two predominant types of moving averages stand out: the Simple Moving Average (SMA) and the Exponentially Weighted Moving Average (EWMA). Both serve the purpose of smoothing out price data to identify trends, yet they differ fundamentally in their approach to weighting data points, which leads to distinct implications in their application in financial markets.

Simple Moving Average (SMA): SMA is the arithmetic mean of a specified number of past prices. It assigns equal weight to each data point within its window, treating older and newer data with identical importance. This equal weighting results in a smoothed line that is easy to calculate and interpret,

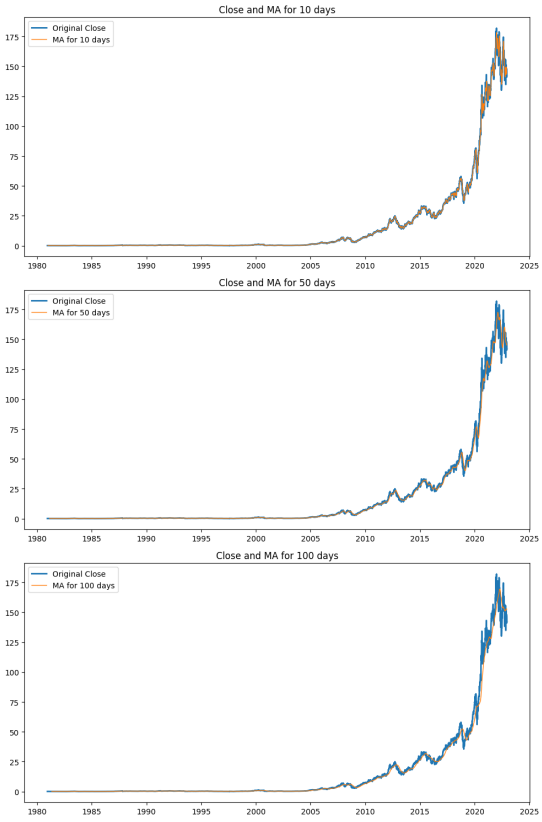
making it a popular choice for identifying long-term trends and support/resistance levels in financial markets.

Exponentially Weighted Moving Average (EWMA): In contrast, EWMA prioritizes recent data points by assigning them exponentially decreasing weights. The most recent data has the most influence on the moving average, with the impact fading for older data. This weighting approach makes EWMA more sensitive to recent market movements, allowing it to adjust more quickly to new trends and reversals.

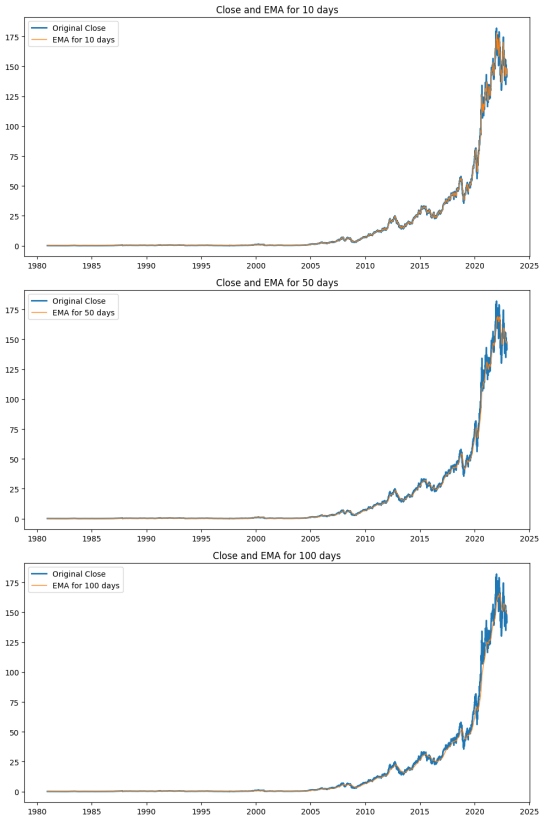
In financial forecasting, choosing between Simple Moving Average (SMA) and Exponentially Weighted Moving Average (EWMA) depends on the analysis goals. SMA, with its equal weighting across data points, is ideal for long-term trend analysis due to its stability and simplicity. It effectively filters out short-term fluctuations, providing a clear view of longer-term market trends. Conversely, EWMA is preferred for short-term strategies as it's more responsive to recent market changes. Its exponentially decreasing weights for older data make it sensitive to immediate market movements, crucial for rapid response in volatile markets.

5.5.1 Project Application

We explored the use of both Simple Moving Average (SMA) and Exponentially Weighted Moving Average (EWMA) in financial forecasting. The initial phase involved plotting moving averages over various time windows, aiming to balance between capturing market trends and minimizing fluctuations. This experimentation led to the selection of a 50-day window as the optimal balance point, Fig: 18.



(a) Different windows for MA



(b) Different windows for EWMA

Figure 18: Different moving average windows

Rate of Change (ROC) Analysis

We utilized the ROC, a momentum indicator, to analyze trends based on a 50-day moving average. The steps included:

1. **Calculating the 50-Day Moving Average:** Averaged the closing prices over the past 50 days, updating it daily by including the latest closing price.
2. **Determining the Percentage Change for ROC:** The ROC was calculated by measuring the percentage change in this moving average over a designated period (rocwindow).
3. **Categorizing Trends:** Classified the ROC into five categories from 'very high positive' to 'high negative' based on their strength and direction. This categorization provided a detailed view of market trends, indicating strong or weak upward or downward movements, or a stable trend.

The result is a percentage that tells you how much the moving average has increased or decreased over the 'rocwindow' period. A positive ROC value indicates that the moving average has increased, suggesting an uptrend, while a negative ROC value indicates a decrease, suggesting a downtrend. A value close to zero implies little to no change over the period.

Feature Generation

Ultimately, the decision to utilize EWMA over SMA was driven by the project's focus on short-term trends. EWMA's greater responsiveness to recent market movements made it more suitable for our needs. This choice was particularly beneficial given the project's emphasis on agility and adaptability in response to rapidly changing market conditions.

Finally, to integrate this feature effectively into our model, we employed categorical encoding. This process transformed the ROC categories into a format conducive to quantitative analysis, ensuring seamless incorporation into the broader forecasting framework. The encoded EWMA-based ROC feature thus became a pivotal element of our financial forecasting model.

Model Integration and Categorical Encoding

The decision to use EWMA was driven by the project's focus on short-term trends. We then encoded the ROC categories for quantitative analysis, making the EWMA-based ROC a key element in our financial forecasting model.

6 Stock Price Prediction

6.1 LSTM

This project employs a Long Short-Term Memory (LSTM) neural network for predicting the stock price of Apple Inc. LSTM networks are especially suitable for time series data, like stock prices, due to their ability to retain information over long sequences, making them ideal for capturing complex temporal patterns.

6.2 Data Preprocessing and Feature Engineering

Before feeding the data into the LSTM model, several preprocessing and feature engineering steps were undertaken to improve the model's performance:

- **Data Normalization:** The stock price data was normalized to have a mean of zero and a standard deviation of one. This step ensures that all features contribute equally to the model's learning process.
- **Trigonometric Date Encoding:** This innovative approach encodes dates as trigonometric features, helping the model to capture time-related patterns such as cyclical behavior in stock prices.

- **Seasonality Addition:** By adding seasonal trends to the data, the model can account for predictable changes that occur at specific times of the year, a common characteristic in financial markets.
- **Lag Features:** Creating lagged versions of the stock price allows the model to learn from historical data points, providing a context for predicting future stock movements.
- **Moving Average for Rate of Change:** Applying a moving average to the rate of change in stock prices helps in smoothing out short-term fluctuations and capturing longer-term trends.

6.3 LSTM Model Architecture

The LSTM model was meticulously designed with the following architecture:

- **Input Layer:** Shaped according to the 5-day time window and the number of features engineered from the data. This layer receives the preprocessed and feature-enhanced input data.
- **LSTM Layers:** Multiple LSTM layers were stacked to enhance the model's ability to learn from the data's temporal dynamics. Each LSTM layer captures different levels of temporal abstraction, making the model robust in understanding complex patterns.
- **Dense Output Layer:** The final layer is a dense layer with a single neuron and a linear activation function, designed to predict the continuous value of Apple's stock price.

6.4 Model Architecture Details

The architecture of the LSTM model, named "Simple_LSTM_regressor", is designed to be relatively simple, taking into consideration the limited number of samples available. Below is the detailed structure of the model:

Layer (type)	**Output Shape**	**Param #**
Input (InputLayer)	[(None, 5, 19)]	0
lstm_20 (LSTM)	(None, 5, 16)	2304
lstm_21 (LSTM)	(None, 16)	2112
dense_10 (Dense)	(None, 1)	17

Table 1: Detailed Structure of the Simple_LSTM_regressor Model

Total Parameters: 4433 (17.32 KB Trainable, 0.00 Byte Non-trainable)

The model begins with an input layer designed to accept sequences of length 5 with 19 features each. It then passes through two LSTM layers, the first maintaining the sequence while the second condenses the information into a 16-unit vector. The final output is produced by a dense layer with a single neuron, providing a scalar prediction of the stock price.

6.5 Training and Evaluation

The model was trained using a historical dataset of Apple's stock prices. The training process involved:

- **Loss Function:** Mean Squared Error (MSE) was used as the loss function, which is standard for regression problems.
- **Early Stopping:** An Early Stopping callback was implemented to prevent overfitting. Specifically, the training process was configured to stop when the validation loss ceased to decrease. The model's training was halted after 10 epochs without improvement in validation loss. This approach ensures that the model retains the most optimal parameters learned during training, enhancing its ability to generalize to unseen data.

- **Learning Rate Reduction:** A learning rate reduction strategy was employed using. The learning rate was reduced by a factor of 0.5 if the validation loss did not improve for a patience of 5 epochs. This approach helps in fine-tuning the model's training by making smaller updates to the weights when the model's performance plateaus, with a minimum learning rate set to 1×10^{-5}.
- **Performance Metrics:** In addition to MSE, Mean Absolute Error (MAE) was also monitored to evaluate the model's performance on both training and validation sets.

6.6 Evaluation of Model Performance

Loss During Training and Validation

The LSTM model's performance over the training epochs is visualized in Fig 19. The training and validation loss curves exhibit the typical behavior of a learning model. Initially, the training loss decreases sharply, indicating rapid learning. The validation loss also decreases but with noticeable volatility. This volatility is expected due to the stochastic nature of the mini-batch gradient descent optimization and the small validation set size.

As the epochs progress, the training loss continues to decrease and begins to plateau, signaling that the model is starting to converge. The early stopping mechanism ensures that the model ceases training before overfitting, as evidenced by the minimal gap between the training and validation losses towards the end of the training.

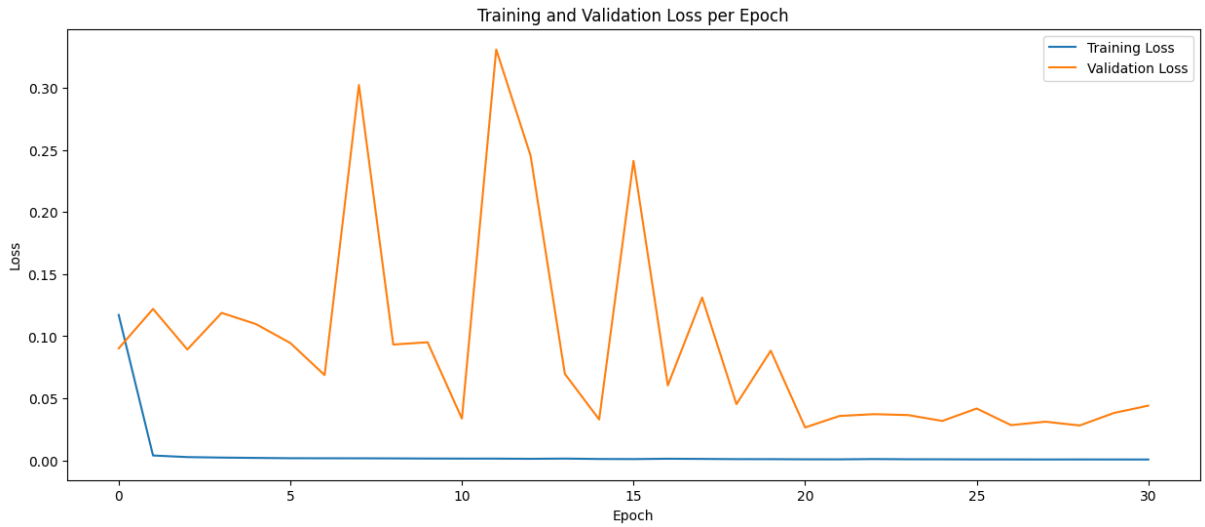


Figure 19: Training and Validation Loss per Epoch

Test Set Results

Fig 20 shows a comparison of the actual values against the predicted values by the LSTM model on the test set. The model demonstrates a strong correlation between the predicted and actual values, which suggests that it has learned the underlying pattern in the data effectively. The predictions closely track the actual price movements, capturing both the trend and the fluctuations to a reasonable extent.

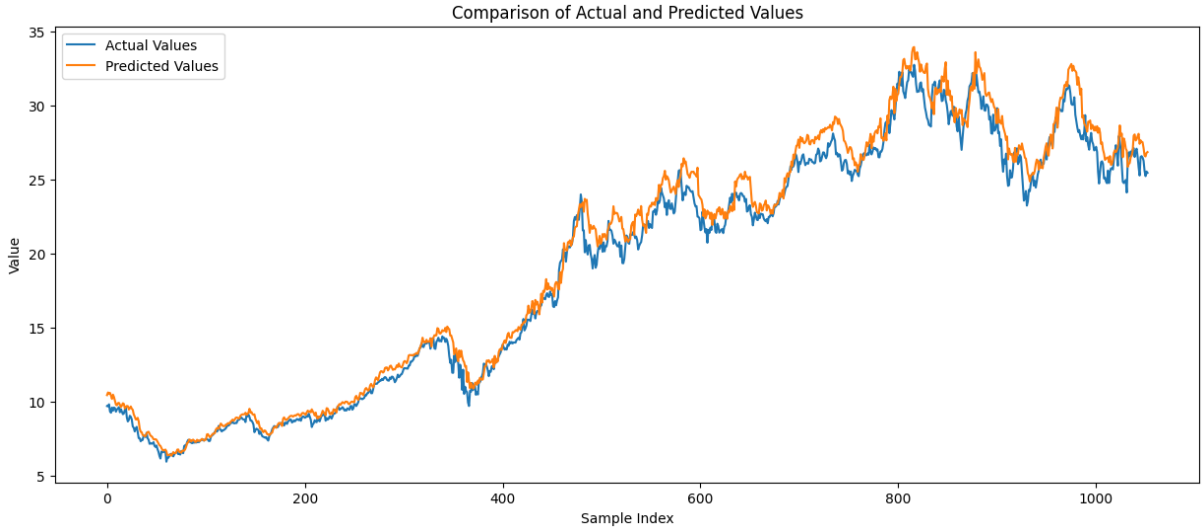


Figure 20: Comparison of Actual and Predicted Values on the Test Set

The close correspondence between the two curves in the test set indicates that the model can generalize well to new data. This is a promising result for the application of the LSTM model in predicting stock prices, showing its potential as a tool for financial analysis and decision-making.

6.7 Model Generalization Across Different Stocks

To evaluate the generalizability of the LSTM model, the same architecture was applied to predict the stock prices of two other major companies in the IT sector: Amazon and IBM. The purpose was to assess whether the model, which was built by analyzing Apple stock data, could provide accurate predictions for other stocks within the same industry.

Amazon Stock Prediction

The graph in Fig 21 presents the comparison between the actual and predicted values for Amazon stock prices. The model has captured the trends and fluctuations of Amazon's stock price with a degree of accuracy that suggests a good generalization. While there are areas where the predicted values diverge from the actual data points, the overall pattern follows closely, indicating that the model has learned relevant features that are applicable across different stocks within the IT sector.

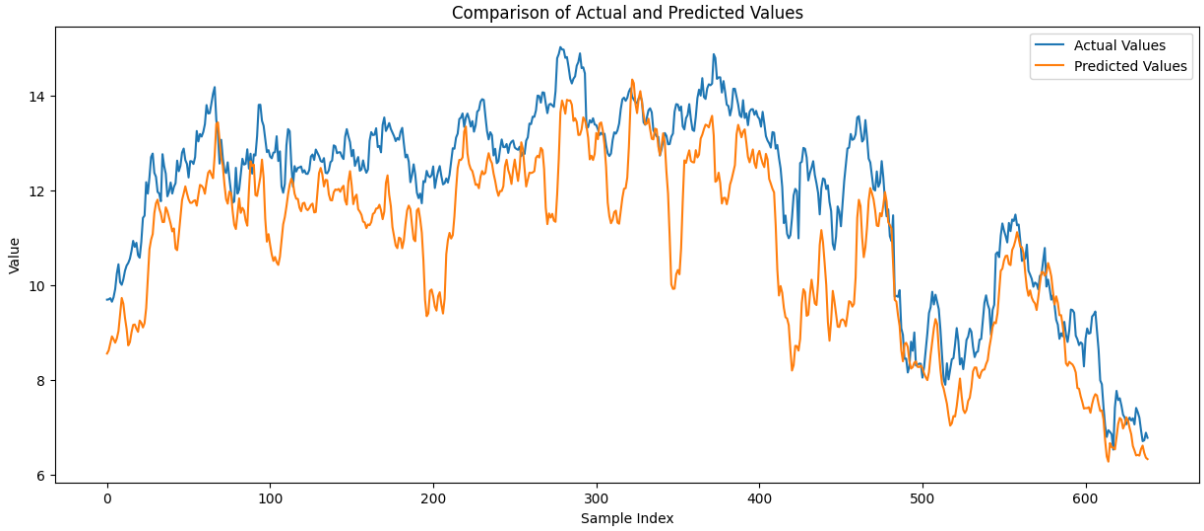


Figure 21: Comparison of Actual and Predicted Values for Amazon Stock

IBM Stock Prediction

Similarly, the model's predictions for IBM stock prices are shown in Fig 22. The predicted values are in close alignment with the actual prices, with some discrepancies which are to be expected in stock price prediction. The consistency in capturing the overall movement of the stock price highlights the model's robustness and its potential to be used as a predictive tool in diverse scenarios within the financial domain.

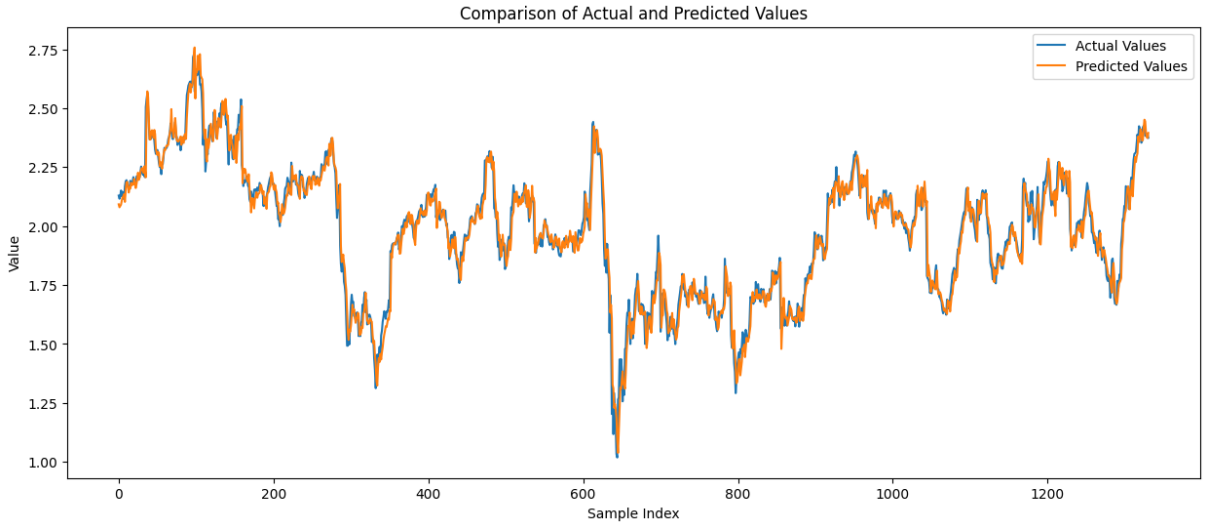


Figure 22: Comparison of Actual and Predicted Values for IBM Stock

These results on both Amazon and IBM stocks reinforce the adaptability of the LSTM model.

6.8 Experimentation with a Hybrid CNN-LSTM Model

In an effort to explore the effectiveness of combining convolutional neural networks (CNN) with Long Short-Term Memory (LSTM) units, a hybrid CNN-LSTM model was developed. The intention was

to leverage the feature extraction capabilities of CNNs along with the sequence learning properties of LSTMs.

Hybrid Model Architecture

The architecture of the hybrid model is outlined in Table 2. It starts with a one-dimensional convolutional layer designed to extract patterns from the input sequences, which is then followed by LeakyReLU activation and max pooling for dimensionality reduction and non-linearity. A second convolutional layer further processes the features, which is again followed by LeakyReLU activation and max pooling. Subsequently, the data is fed into an LSTM layer with a dropout mechanism to prevent overfitting. This is followed by a second LSTM layer for deeper temporal processing. Finally, the model uses a dense layer with LeakyReLU activation and a dropout layer before the output layer, which predicts the stock price.

Layer (type)	**Output Shape**	**Param #**
Input (InputLayer)	[(None, 5, 16)]	0
conv1d (Conv1D)	(None, 5, 64)	3136
leaky_re_lu (LeakyReLU)	(None, 5, 64)	0
max_pooling1d (MaxPooling1D)	(None, 2, 64)	0
conv1d_1 (Conv1D)	(None, 2, 16)	3088
leaky_re_lu_1 (LeakyReLU)	(None, 2, 16)	0
max_pooling1d_1 (MaxPooling1D)	(None, 1, 16)	0
lstm (LSTM)	(None, 1, 32)	6272
dropout (Dropout)	(None, 1, 32)	0
lstm_1 (LSTM)	(None, 32)	8320
dropout_1 (Dropout)	(None, 32)	0
dense (Dense)	(None, 16)	528
leaky_re_lu_2 (LeakyReLU)	(None, 16)	0
dropout_2 (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 1)	17

Table 2: Architecture of the Hybrid CNN-LSTM Model

Performance of the Hybrid Model

Despite the theoretical advantage of the hybrid approach, the performance of the CNN-LSTM model was not satisfactory. The results indicated that the complexity introduced by the convolutional layers did not translate into better predictive accuracy. One possible explanation for the underwhelming performance could be the relatively small size of the dataset, which might not have been sufficient for the model to fully learn and generalize the underlying patterns. Additionally, the hyperparameters of the model, such as the number of filters in the convolutional layers and the number of neurons in the LSTM layers, may require further tuning to suit the characteristics of the stock price data.

The experiment with the hybrid model reinforces the idea that complexity does not always yield better results, particularly in cases where data is limited. It underscores the importance of model selection and hyperparameter tuning in achieving optimal performance in time series forecasting.

7 Conclusion

In conclusion, despite the challenging nature of predicting unstable financial time series, the right combination of data analysis, preprocessing, and model selection can yield promising results. Our study demonstrated that neural networks, when complemented with thorough preprocessing, can be effective in this context. However, the precision of these models isn't yet reliable enough for critical decision-making. Given more time for fine-tuning hyperparameters and exploring ensemble models, there's potential for significantly improved accuracy and reliability in forecasting financial time series.

References

[1] Apple. *Yahoo Finance*. https://finance.yahoo.com/quote/AAPL/.

[2] Yahoo Finance. *SP500 Trend*. https://ca.finance.yahoo.com/quote/NOXL/history?period1=883612800&period2=915062400&interval=1d&filter=history&frequency=1d&includeAdjustedClose=true. 2023.

[3] Kaggle. *Stock Market Data Dataset*. https://www.kaggle.com/datasets/paultimothymooney/stock-market-data. 2023.

[4] Scipy. *Scipy Z-Score*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.zscore.html.