Applied Time Series Forecasting in Engineering and Management: From Bitcoin EDA to S&P 500 Model Evaluation

Author:

Hussnain Khalil

Bachelor of Engineering in Data Science in Technik und Wirtschaft

E-Mail: huk7464@thi.de

GitHub: github.com/rhkraptor

LinkedIn: linkedin.com/in/hussnain-khalil

Supervisor:

Prof. Dr. Sina Huber

Technische Hochschule Ingolstadt

Ingolstadt, June 19, 2025

Contents

1 Part I

1.1 Introduction

This section introduces a self-collected time series dataset focused on the daily prices of Bitcoin (BTC-USD). The dataset was obtained using the `yfinance` Python library and covers a continuous period of 364 days, ranging from June 18, 2024 to June 16, 2025. The data represents one observation per calendar day, making the time resolution **daily**, and the total number of data points equals **364**. The variable measured is the price of Bitcoin in U.S. dollars (USD).

The rationale for selecting Bitcoin as the target of analysis stems from its highly volatile nature and its increasing relevance in global finance and speculative markets. Unlike traditional indices, Bitcoin operates 24/7 without centralized market closures, which introduces a unique pattern in terms of trading volume, behavioral trends, and market shocks. These dynamics make it an intriguing candidate for exploratory analysis in the context of forecasting.

Although the forecasting model phase was not completed due to scope of this course, the dataset offers a rich ground for investigating **stationarity**, **trend behavior**, **seasonality**, and **volatility** — all foundational elements in the field of time series forecasting.

In the context of engineering and management, Bitcoin's price evolution is representative of high-frequency, nonlinear systems with irregular trends and cyclical effects. Understanding such structures is valuable for developing forecasting pipelines that can later be applied to domains like *energy pricing*, *inventory demand*, or *financial risk modeling*.

1.2 Data Collection Method

The dataset was collected programmatically using the `yfinance` Python library, which interfaces with the Yahoo Finance API to retrieve historical financial data. The process was fully automated, requiring no manual input, and was designed to fetch live market data at the time of execution. The data collection script was configured to retrieve the daily closing prices of Bitcoin (BTC-USD) for the most recent 364 calendar days, starting from June 18, 2024 and ending on June 16, 2025.

The collection method was based on a single function, `fetch_btc_data(days=364, interval='1d')`, which pulled data at a daily frequency. This setup ensured a consistent temporal resolution and complete coverage without missing dates, given that cryptocurrency markets operate continuously, including weekends. The function also included data cleaning steps such as index resetting, renaming of relevant columns, and coercion of numerical fields to ensure data quality.

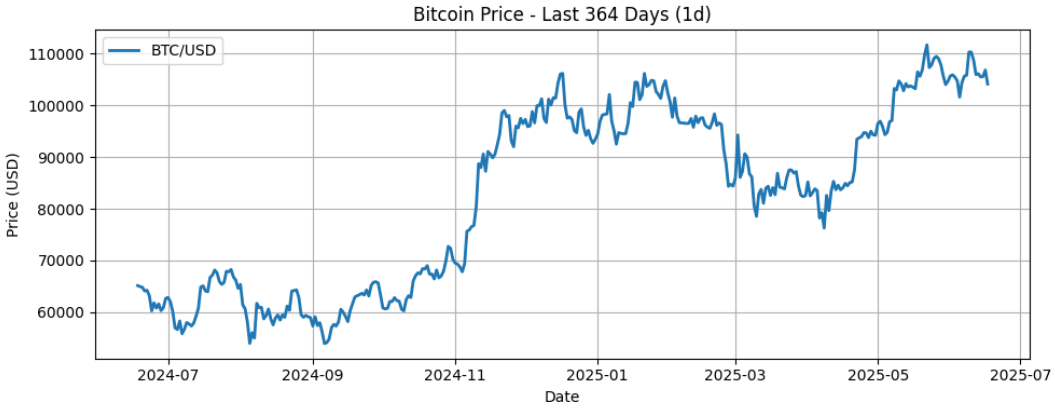

Figure 1: BTC-USD Price from last 364 Days

The primary tool used was the **Python** programming language, supported by a **Jupyter Notebook** environment. The key libraries and frameworks employed include:

- `yfinance` — for data retrieval
- `pandas` — for data manipulation and timestamp handling

- `matplotlib` and `seaborn` — for visualization and inspection
- `statsmodels` — for statistical analysis and stationarity tests

The environment was designed to be reproducible, dynamic, and adjustable: the total number of days and interval type (daily, weekly, monthly) could be easily modified through global parameters. All transformations and metadata extraction were logged and visualized to ensure transparency and traceability throughout the process.

By using publicly available, up-to-date market data and reproducible code, the collection method guarantees both accessibility and replicability, fulfilling scientific requirements for time series analysis in engineering contexts.

1.3 Descriptive Analysis

To understand the behavior of the Bitcoin dataset, an extensive Exploratory Data Analysis (EDA) was performed. This includes both visual and statistical evaluations to assess trends, volatility, distribution characteristics, and structural properties relevant to time-series forecasting.

The time-series plot of Bitcoin prices over the 364-day period reveals a general upward trend, interspersed with several high-volatility phases. The price ranged from a minimum of $53,948.75 to a maximum of $111,673.28, with a mean of approximately $82,518.57 and a standard deviation of $17,448.15. These values indicate a wide fluctuation band, typical of crypto assets.

Figure 2 illustrates the distributional characteristics of the daily closing prices of Bitcoin during the observed period.

The box plot clearly shows that the distribution is right-skewed, as indicated by the mean (red dashed line) lying to the right of the median (central line of the box). This skewness implies that there are several high-end outliers — extremely high BTC prices that extend the upper tail of the distribution. These are not mere noise, but likely correspond to phases of rapid speculative rallies typically seen in late-stage bull markets.

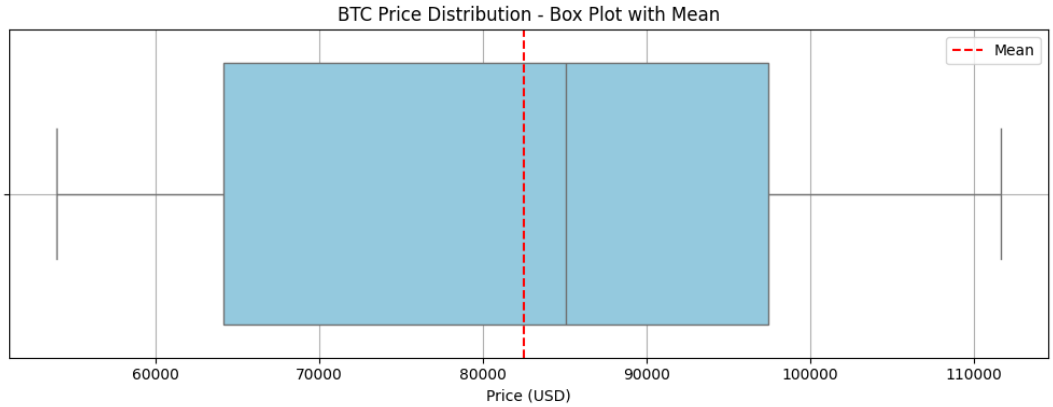

Figure 2: Box Plot of BTC-USD Price from last 364 days

The width of the interquartile range (IQR) is substantial, which reflects a broad middle 50% spread in price values — a symptom of Bitcoin's high price volatility. The whiskers of the boxplot extend relatively far on both sides, but the upper whisker in particular reaches above $110,000, reinforcing the presence of exceptional price spikes.

The figure 3 shows a histogram combined with kernel density estimate (KDE) provides further information. Unlike a standard unimodal distribution, the KDE reveals a bimodal structure — two peaks in the density curve. The first peak occurs around $60,000–$65,000, while the second peak lies near $95,000–$100,000. This suggests that the market experienced at least two different pricing regimes during the year, potentially reflecting a transition from a correction phase to a renewed bullish trend.

These findings are consistent with the behavior of speculative assets, where sharp rallies and corrections form observable clusters. For time series modeling, such bimodality can challenge assumptions of normality and homogeneity, especially when building models based on constant variance or linearity.

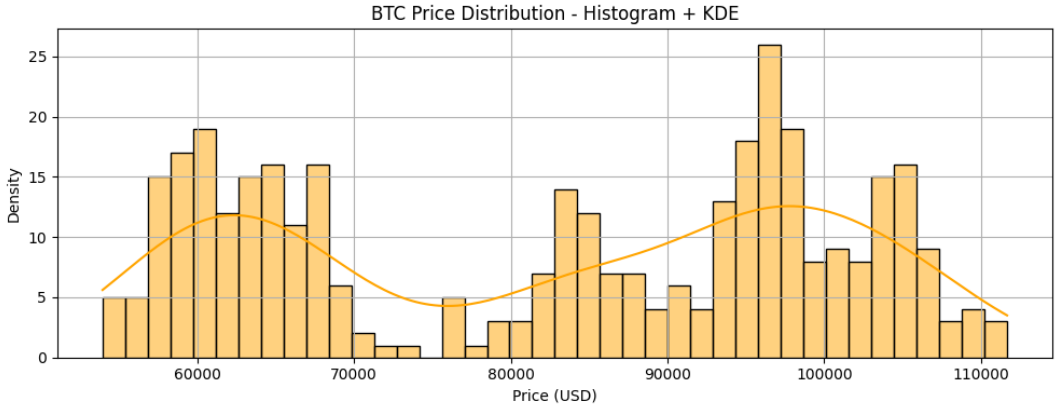

Figure 3: Histogram and KDE revealing a bimodal distribution.

To assess whether the Bitcoin price series met the conditions required for classical time series modeling, a stationarity check was conducted. Stationarity implies that the statistical properties of the series — such as mean and variance — remain constant over time, which is a fundamental assumption for models like ARIMA and SARIMA.

As shown in Figure 4, the BTC/USD price series exhibited a visible upward trend, with the rolling mean (red line) increasing notably from October through December. This violates the assumption of constant mean. Additionally, while the rolling standard deviation (green line) appeared more stable, it still showed mild fluctuations, especially during high-volatility phases.

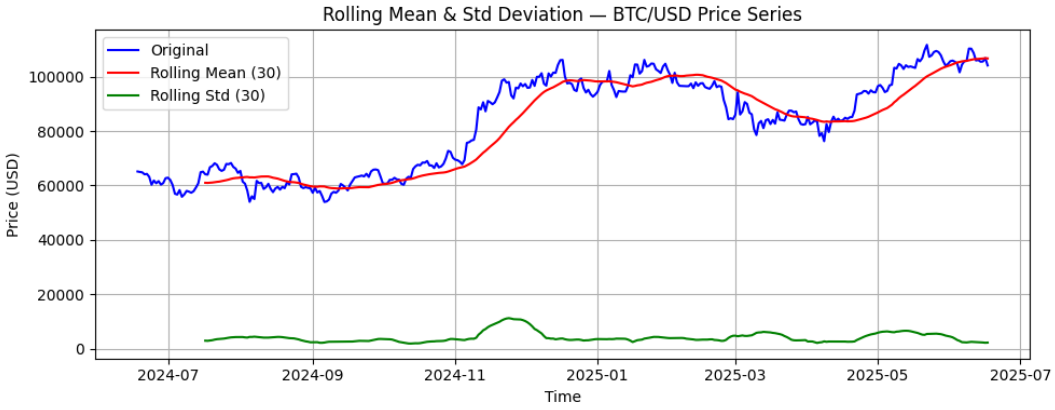

Figure 4: Rolling mean and standard deviation of raw BTC/USD series (non-stationary).

These observations were supported by the Augmented Dickey-Fuller (ADF) test [1], which yielded a test statistic of -1.0396 and a p-value of 0.8385 — far above the 0.05 threshold — indicating that the series is non-stationary (i.e., we fail to reject the null hypothesis H_0).

To address this, a log transformation was applied to stabilize variance, followed by first-order differencing to eliminate the trend component. The transformed series is also shown in Figure 5, where both the rolling mean and standard deviation remain relatively flat throughout the year. The transformed series appears to fluctuate around zero, another key indicator of stationarity.

The ADF test on this log-differenced series confirmed the improvement, returning a test statistic of -19.9736 with a p-value of 0.0000, well below any critical threshold. This result strongly suggests that the transformation successfully rendered the series stationary.

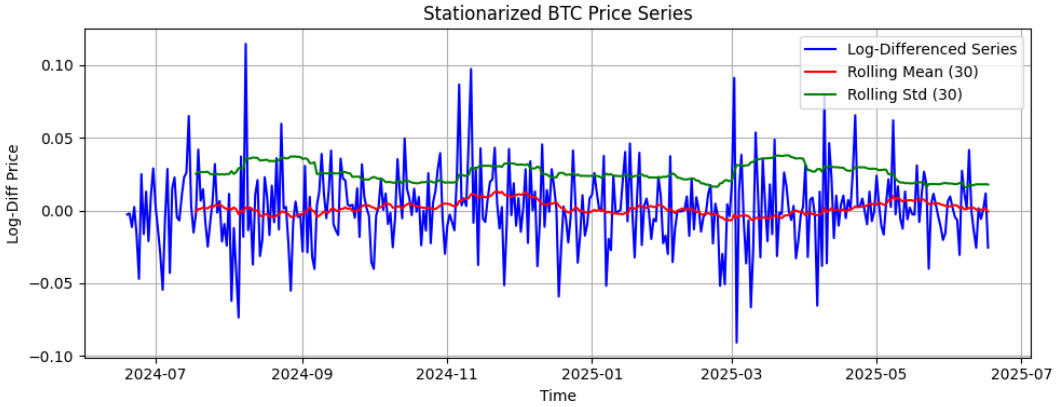

Figure 5: Transformed log-differenced series with flat rolling stats.

Although forecasting models are beyond the scope of this section, achieving stationarity is a necessary preparatory step for implementing models such as ARIMA and SARIMA, where assumptions about residuals, lag dependencies and variance stability must be met. Furthermore, stationary series serves as a valid input for autocorrelation (ACF) and partial autocorrelation (PACF) analysis, which guides the selection of ARIMA model parameters - a workflow explored in Part 2 of this report.

To gain deeper insight into the structure of the series, a seasonal decomposition was applied to the log-transformed (but not differenced) series using a multiplicative model. This split the time series into three key components: *trend*, *seasonality*, and *residuals* (Figure 6).

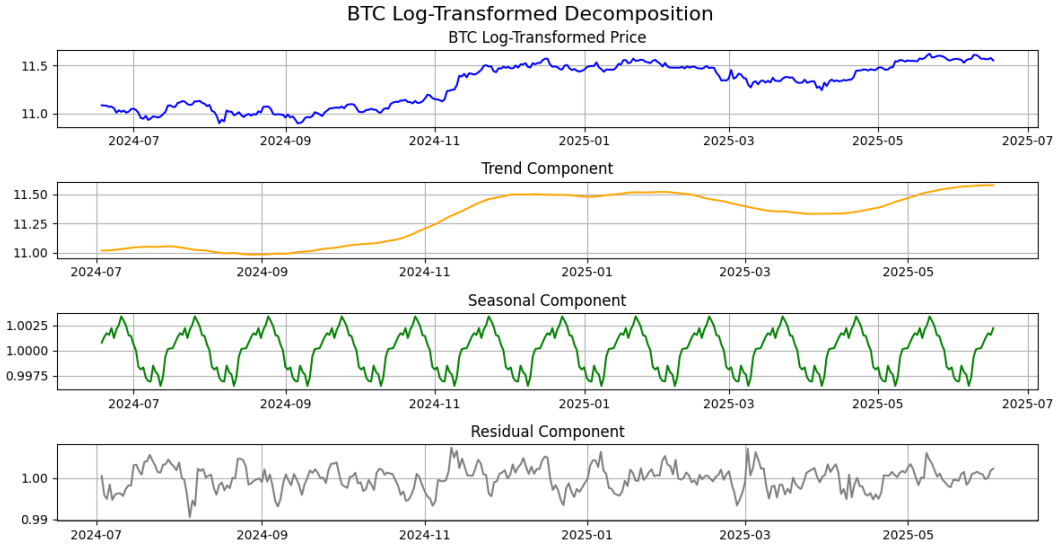

Figure 6: Seasonal decomposition of the log-transformed Bitcoin price series using an additive model.

The trend component shows a gradual and consistent increase, particularly visible from October 2024 through early 2025, before stabilizing in late spring. This mirrors the upward price movements observed in the raw series.

The seasonal component displays a regular cyclical pattern, with peaks and troughs repeating approximately every 30 days — suggesting the presence of a monthly seasonality, possibly linked to investor behavior, market cycles, or macroeconomic timing patterns.

The residual component is relatively narrow and centered around one, indicating that the bulk of the variation in the series can be explained by the trend and seasonality components, leaving behind low-magnitude noise.

This decomposition step is not only useful for understanding the internal dynamics of the Bitcoin price seri, but is also a critical part of preparing time series for modeling. In particular, it helps determine whether a seasonal component is strong

enough to justify the use of models like SARIMA or Holt-Winters. In this case, while the seasonality is visible, it may not dominate the dynamics — a point which will be further discussed in comparison to other datasets and models in Part 2.

To understand the time-dependent structure of the stationary BTC price series, we analyzed the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) — two core tools for identifying lag-based dependencies in time series modeling.

Autocorrelation Function (ACF)

The ACF measures the linear correlation between a time series and its own past values (lags) [2]. For a time series y_t, the autocorrelation at lag k is defined as:

$$\rho_k = \frac{\sum_{t=k+1}^{T}(y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum_{t=1}^{T}(y_t - \bar{y})^2}$$

Here, $\bar{y}$ is the mean of the series, and ρ_k tells us how well the current value can be explained by the value k time steps before.

Partial Autocorrelation Function (PACF)

The PACF measures the correlation between y_t and y_{t-k} after removing the linear effects of the intermediate lags $y_{t-1}, \ldots, y_{t-k+1}$ [2]. In other words, it gives the net effect of lag k on y_t, holding other lags constant.

For lag k, the PACF is obtained by fitting an autoregressive model of order k and extracting the coefficient ϕ_k on y_{t-k}.

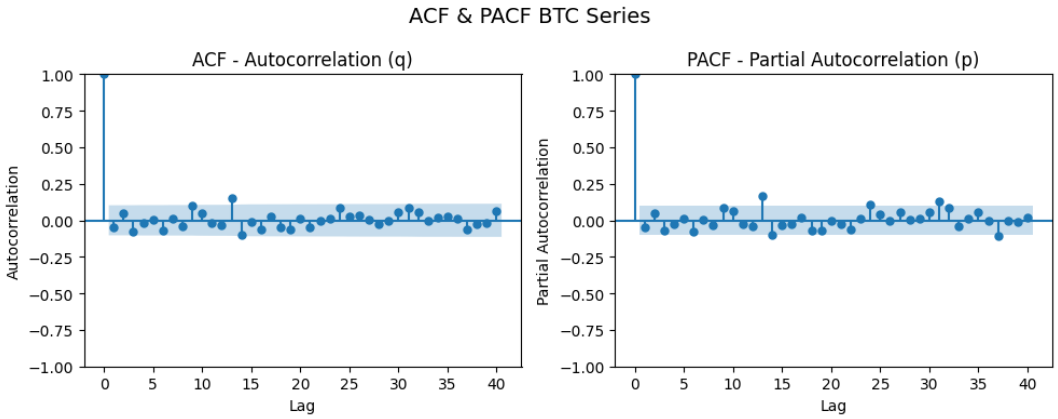

Figure 7: Left: Autocorrelation Function (ACF). Right: Partial Autocorrelation Function (PACF) of the log-differenced BTC series.

The ACF plot in Figure 7 of the log-differenced BTC price series shows a very strong autocorrelation at lag 1, after which the values rapidly decline and fluctuate around zero. Most lags beyond the first fall within the 95% confidence bounds (shaded region), implying they are not statistically significant.

Similarly, the PACF plot in Figure 7 also exhibits a strong spike at lag 1, followed by values close to zero — indicating that once the first lag is accounted for, further lags do not add predictive power.

This behavior suggests that:

- The immediate past (lag 1) has the dominant influence on the current value
- The series shows short-memory structure, typical of AR(1) or MA(1) processes
- There is no strong long-term dependence, simplifying the model choice

While forecasting is beyond the scope of this section, these ACF and PACF patterns provide foundational guidance for future ARIMA modeling — particularly indicating that a low-order ARIMA model (e.g., ARIMA(1,1,1)) would be appropriate for capturing the series dynamics without overfitting.

1.4 Discussion

The self-collected dataset of daily Bitcoin prices revealed several important characteristics relevant to forecasting in engineering and financial contexts. The combination of descriptive statistics, visual inspection, and statistical testing enabled a comprehensive understanding of the series' temporal structure and statistical properties.

The Bitcoin price series showed high volatility with significant variation across the 364-day period. Summary statistics reported a mean of $82,518.57, with a standard deviation of over $17,000, indicating large price swings. The boxplot and histogram confirmed a right-skewed distribution, pushed upward by frequent price rallies. Furthermore, the bimodal shape of the histogram suggested that the market experienced two dominant price clusters, pointing to different behavioural regimes within the year — potentially a correction followed by a strong recovery.

Visual inspection and the Augmented Dickey-Fuller test confirmed that the original series was non-stationary, with a changing mean and unstable variance. Applying a log transformation and first-order differencing successfully stabilized the series, producing a stationary time series suitable for model development.

Seasonal decomposition of the log-transformed series separated the structure into a trend component, showing upward growth until early 2025; a seasonal component, reflecting clear monthly cycles; and a residual component, capturing short-term randomness. Although the seasonality was regular, it remained relatively modest in amplitude, an insight that guides whether complex seasonal models such as SARIMA or Holt-Winters would be justified.

Finally, the ACF and PACF plots of the stationary series indicated short-term dependency dominated by lag 1, with little long-term memory. This justifies the use of low-order ARIMA models if forecasting were to be pursued.

In summary, this exploratory analysis demonstrated that the self-collected Bitcoin dataset exhibits strong trend behavior, short-term dependencies, and moderate seasonality. The insights gained form a technically sound basis for subsequent forecasting efforts, even though modeling is beyond the scope of this section. These results not only reflect the dynamic nature of cryptocurrency markets but also highlight the importance of transformation, decomposition, and diagnostic testing in preparing any time series for predictive modeling.

2 Part II

2.1 Dataset Overview and EDA

This section presents a comprehensive exploratory analysis of the S&P 500 Index based on publicly available financial data retrieved via the `yfinance` API. The dataset covers a five-year period from June 18, 2020 to June 17, 2025, including 1,304 business-day observations. The index, denoted by the ticker symbol `^GSPC`, is a widely recognized benchmark representing the collective performance of 500 major U.S. companies, making it an ideal candidate for time series modeling in financial forecasting.

The time series was retrieved at a daily resolution, excluding weekends and holidays, and included five primary variables: Close, Open, High, Low, and Volume. The closing price was selected as the key forecasting target due to its widespread use in technical and quantitative analysis. The raw dataset was clean and required minimal preprocessing. The index was explicitly set to business-day frequency (`B`) to prepare the data for subsequent modeling.

A preliminary statistical summary revealed a mean closing price of $4,519.24, with a standard deviation of $766.77, and values ranging from a minimum of $3,009.05 to a maximum of $6,144.15. These figures indicate substantial variability in the index over the observed period, consistent with the market's recovery from the COVID-19 downturn and subsequent inflationary and policy-driven cycles. The skewness of 0.49 indicates a slight right-skew, while a kurtosis of -0.66 suggests a flatter-than-normal distribution (platykurtic) with thinner tails.

These statistical insights are visually confirmed in the distribution plots. Figure 8 shows the daily closing price series over time, highlighting a long-term upward trend with noticeable dips during late 2022 and again in early 2025.

Figure 8: S&P 500 daily closing prices (June 2020 – June 2025).

The boxplot in Figure 9 reflects the data's range and central tendency, showing a compact interquartile range centered around 4400–4700, with only a few minor outliers near the recent highs above 6000.

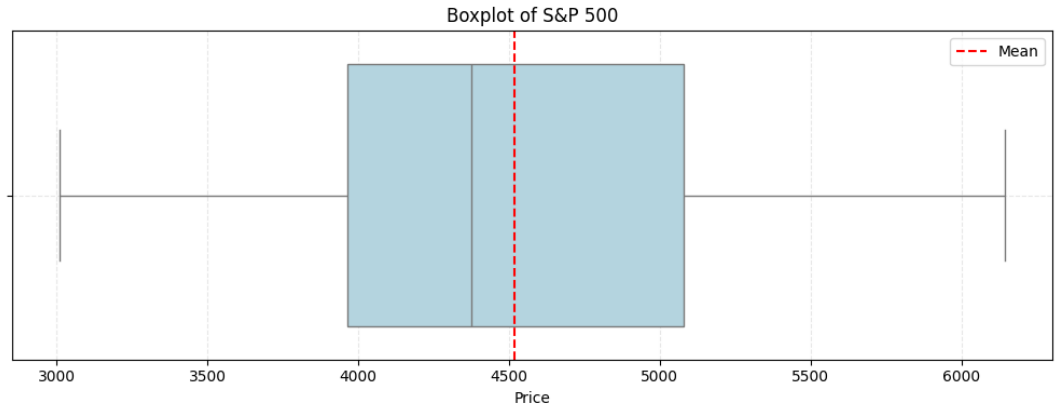

Figure 9: Boxplot of S&P 500 closing prices.

The histogram with KDE overlay in Figure 10 further illustrates the slightly asymmetric distribution: although the peak is centered, a small tail extends into the higher price regions, creating a modest rightward skew. This histogram also hints at multimodal behavior, with subtle bulges suggesting structural market phases.

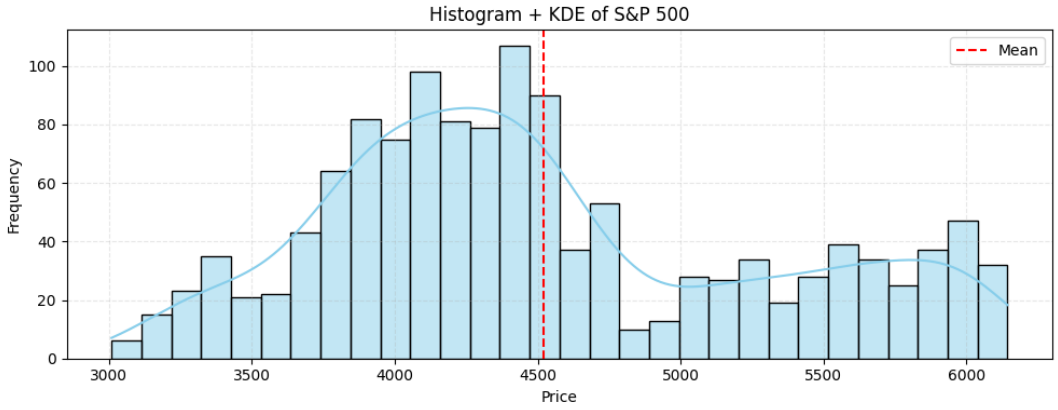

Figure 10: Histogram and KDE of S&P 500 closing prices.

To validate the normality assumption, which is often necessary for residual diagnostics in classical time series models, a Quantile-Quantile (QQ) plot was produced (Figure 11). The data points deviate from the diagonal line especially in the tails, with both the lower and upper ends bending away from the theoretical quantiles. This confirms that the S&P 500 price series does not follow a strictly Gaussian distribution. Such non-normality is expected in real-world financial data and must

be accounted for in model selection, particularly when using models that rely on normality of errors.

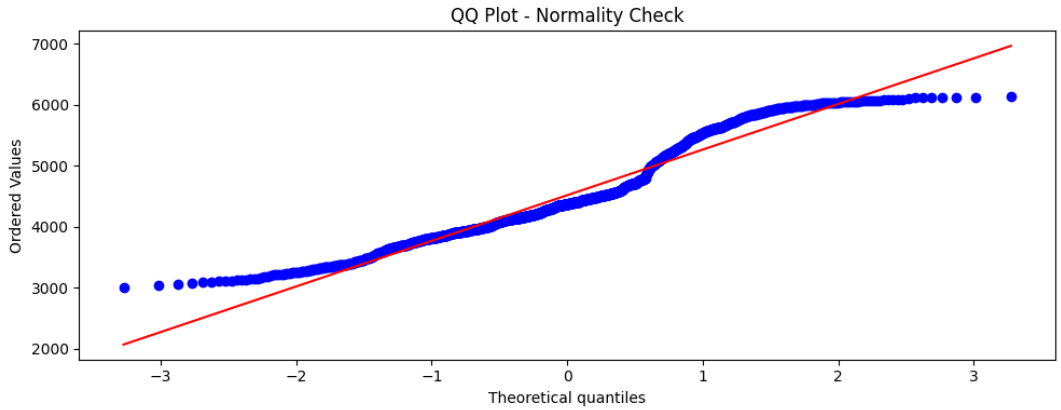

Figure 11: Quantile-Quantile (QQ) plot for S&P 500 closing prices.

To further examine potential calendar-based patterns, the average closing prices were aggregated by month across the entire time span. The resulting bar plot (Figure 12) illustrates only modest variation across months, with slightly higher average prices in February, May, and December, and somewhat lower values in September and October. However, these fluctuations are relatively small in magnitude and do not exhibit any dominant or repeating peaks that would justify immediate inclusion of strong seasonal terms. Contrary to common assumptions (e.g., the “January effect”), no clear cyclic trend was observed.

This weak seasonality suggests that including seasonal components in forecasting models may not be necessary, unless later confirmed through residual diagnostics. As such, the monthly profile supports the decision to prioritize non-seasonal models like ARIMA in initial modeling phases, without prematurely adding seasonal complexity that may risk overfitting.

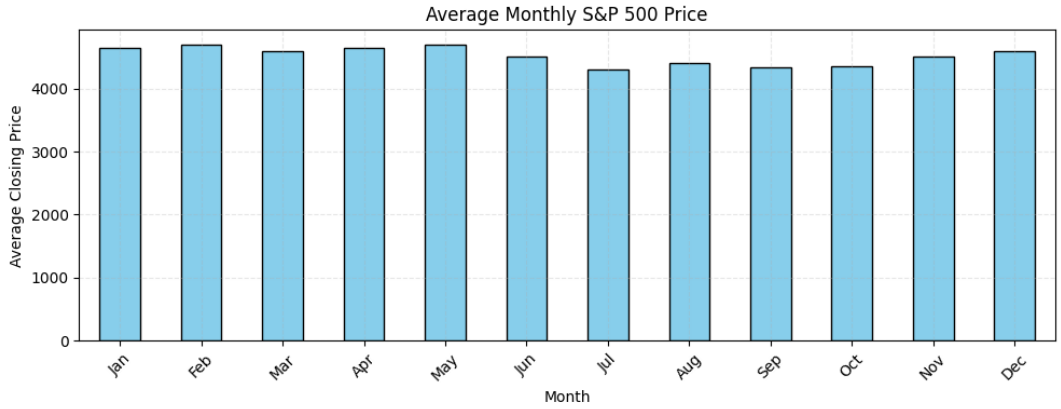

Figure 12: Monthly trend and volatility view of S&P 500

To prepare the dataset for forecasting, a train-test split was implemented based on a chronological 80/20 division. As illustrated in Figure 13, the dataset was divided into a training set of 1,043 business days, covering data from mid-2020 through mid-2024, and a test set of 261 days, spanning from June 2024 to June 2025. This approach preserves the temporal integrity of the time series and simulates a realistic forecasting scenario where future values are predicted based only on past data.

Visual inspection of the split confirms that both the training and test sets retain the overall upward trajectory of the index, reflecting the consistent growth trend observed in the full series. The test set contains a sufficient number of data points and includes diverse market conditions — including minor dips and recoveries — making it a suitable foundation for evaluating the performance and generalizability of the forecasting models developed in the incoming subsection.

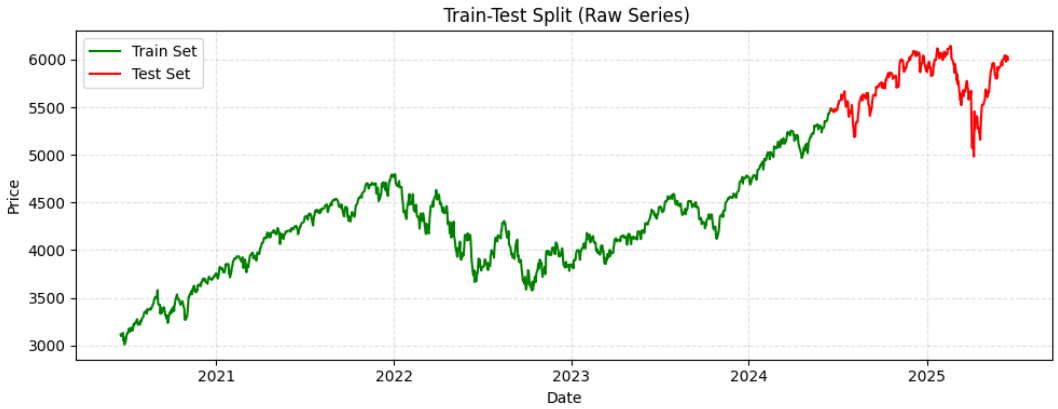

Figure 13: Train-Test split of the S&P 500 time series. Training: mid-2020 to mid-2024 (80%). Test: June 2024 to June 2025 (20%).

Following the train-test division, the next essential step in the forecasting pipeline was to assess the stationarity of the training data — a critical requirement for time series models like ARIMA and SARIMA that assume constant mean and variance over time.

Visual inspection of the training series (Figure 14) reveals a persistent upward trend, consistent with the overall growth of the S&P 500. However, trend alone does not confirm non-stationarity. To explore this further, a rolling statistics plot (Figure 15) was used, showing the 30-day rolling mean and standard deviation superimposed on the original series. The rolling mean displays clear fluctuations over time, especially during 2022–2024, while the rolling standard deviation remains relatively stable. This implies that while volatility may be stationary, the mean is not, indicating a non-stationary process.

To validate this visually derived insight, the Augmented Dickey-Fuller (ADF) test was applied to the raw training series. The test produced:

```
ADF Statistic:  -1.0048
    p-value:  0.7515
    Critical Values:
1%:  -3.4367,  5%:  -2.8643,  10%:  -2.5683
```

Since the test statistic fails to fall below any of the critical values, we **fail to reject the null hypothesis of a unit root**, confirming that the series is likely non-stationary.

To resolve this, a log transformation was applied to stabilize the variance, followed by first-order differencing to eliminate the trend component. The resulting series was then re-evaluated using both visual tools and the ADF test. Post-transformation, the rolling mean flattened, and the standard deviation remained stable across the entire time window — visually supporting stationarity. As shown in Figure 16, the transformed series exhibits no visible trend and a stable variance over time.

Statistically, the ADF test confirmed this transformation's effectiveness:

```
ADF Statistic:  -10.4142
    p-value:  0.0000
    Critical Values:
1%:  -3.4367,  5%:  -2.8643,  10%:  -2.5683
```

Since the statistic is well below all critical values, we **reject the null hypothesis with high confidence**, indicating that the transformed series is stationary and now suitable for classical time series modeling.

Achieving stationarity at this stage is foundational for all models that assume residuals to be white noise or that rely on autoregressive and moving average terms. It also prepares the series for the next stage — autocorrelation diagnostics (ACF/PACF) and parameter tuning — which form the basis for model configuration.

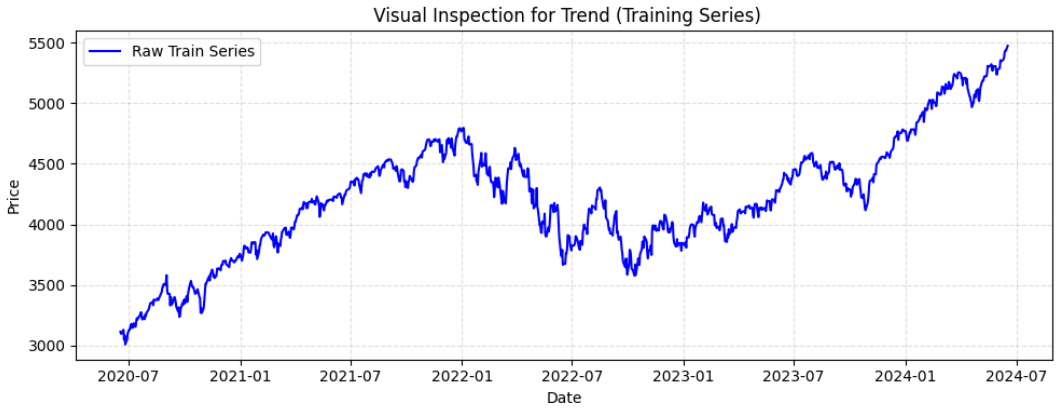

Figure 14: Visual inspection of raw S&P 500 training series.

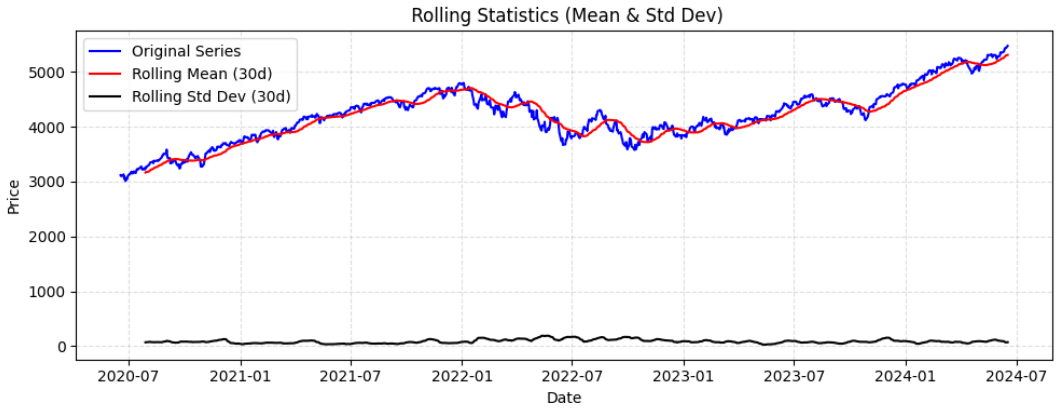

Figure 15: Rolling statistics (30-day mean and standard deviation) of the training series.

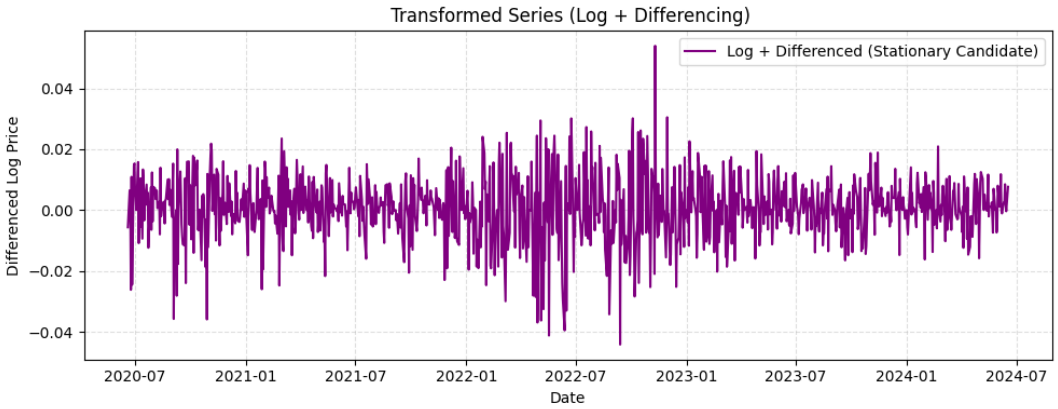

Figure 16: Transformed training series after log transformation and first-order differencing.

To further understand the internal structure of the log-transformed S&P 500 series, a seasonal decomposition was performed using an additive model with a period of 252 trading days — corresponding to the approximate number of business days in one calendar year. The decomposition (Figure 17) separates the series into three primary components: *trend*, *seasonality*, and *residuals*.

The trend component reveals a consistent long-term upward trajectory, aligning with expectations for a broad-market index like the S&P 500. This trend is particularly prominent from mid-2023 onward, reflecting post-recession recovery and continued economic expansion.

The seasonal component, by contrast, shows only weak cyclic fluctuations. The magnitude of this component ranges narrowly between approximately -0.04 and $+0.04$ in the log scale, indicating that seasonal effects — while present — are

not dominant. These minor periodic oscillations could be linked to calendar-based trading behavior or fiscal quarter effects, but they do not exhibit strong or persistent influence over the long-term price pattern.

The residual component consists of short-term, irregular variations around zero. Its behavior is relatively unstructured and exhibits no obvious autocorrelation, suggesting that most of the meaningful signal has been captured by the trend, with seasonality playing only a minor role.

Taken together, these decomposition results justify the decision to adopt a non-seasonal ARIMA model rather than a seasonal variant like SARIMA or Holt-Winters. Introducing seasonal complexity would add computational overhead and increase the risk of overfitting without delivering substantial gains in predictive performance. As such, the seasonal decomposition analysis provides both statistical and structural support for simplifying the model configuration in the next phase of the project.

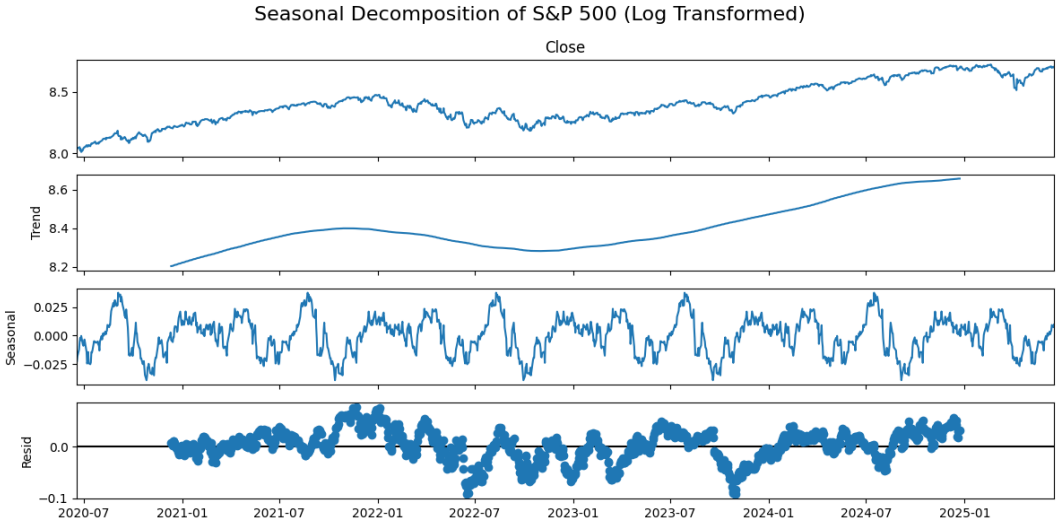

Figure 17: Seasonal decomposition of the log-transformed S&P 500 series (additive model, period = 252). Top to bottom: original series, trend, seasonal, and residual components.

To determine the optimal parameters for ARIMA modeling [2], the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots were computed on the stationary version of the training series (log-transformed and differenced once). These diagnostic tools help identify suitable values for the AR (autoregressive) and MA (moving average) terms, respectively.

As shown in Figure 18, the PACF plot exhibits a series of notable spikes that remain statistically significant up to lag 9, suggesting the presence of meaningful autoregressive structure over this range. This pattern supports setting the AR order parameter $p = 9$. In parallel, the ACF plot shows a similar tapering behavior with several significant autocorrelations also extending up to lag 9, indicating that the MA order parameter $q = 9$ may be appropriate as well.

This behavior implies the series contains short- to medium-term dependencies, where the influence of past observations persists for several time steps before diminishing. The structure is neither purely autoregressive nor purely moving average, justifying a balanced **ARIMA(9,1,9)** configuration. The differencing order $d = 1$ was previously established through stationarity testing, ensuring the model operates on a stable mean-reverting series.

In summary, the **ARIMA(9,1,9)** model is recommended based on autocorrelation diagnostics. This choice balances model flexibility with interpretability, enabling the model to capture more nuanced lag relationships while remaining grounded in empirical evidence from the ACF and PACF structure.

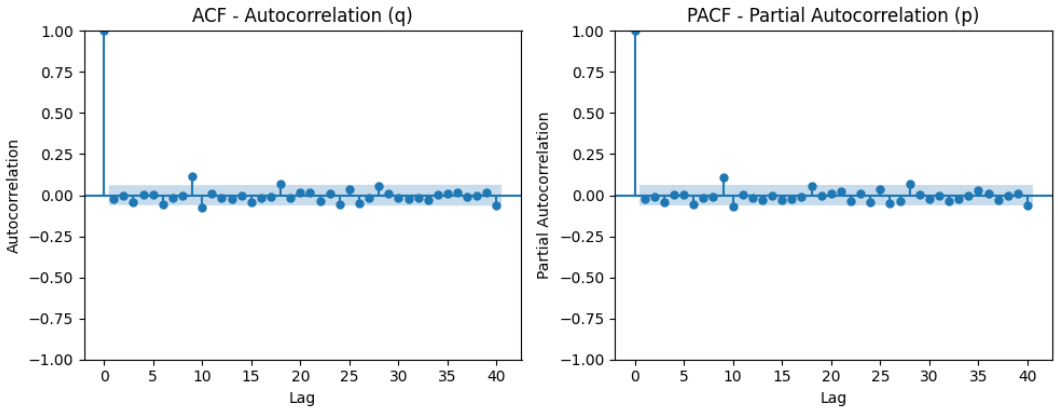

Figure 18: ACF and PACF plots of the log-differenced S&P 500 training series.

2.2 Methodology

This section outlines the forecasting strategy and model configurations used to predict the S&P 500 Index, based on the time series insights established in Section 2.1. The forecasting framework includes both simple baseline models and more sophisticated statistical and machine learning techniques. The models were chosen to represent a broad methodological spectrum, from interpretable benchmarks to data-driven black-box learners. Each model was evaluated on its ability to generalize to unseen data, using the same test set for fairness.

All models, unless otherwise stated, were trained on the raw training series denoted as y_{train}, which refers to the daily closing prices of the S&P 500 from June 18, 2020 to June 17, 2024. This subset contains 1,043 business-day observations. The test series, y_{test}, consists of the subsequent 261 days from June 18, 2024 to June 17, 2025.

A key exception was the ARIMA model, which requires stationarity. For this model, the original training data was first log-transformed to stabilize variance, producing $\log(y_{\text{train}})$, and then first-order differenced to remove the trend, all fitting and residual diagnostics for ARIMA were performed using this version. The model outputs were then exponentiated and cumulatively summed to bring forecasts back to the original price scale for evaluation and comparison. Across all models, the forecast horizon was set equal to the size of the test set (261 business days).

Baseline Models

To establish baseline performance levels, a set of benchmark models was implemented. These models are simple yet essential in any time series forecasting pipeline. They serve as a reference point against which the performance of more advanced models can be compared [3]. Figure 19 illustrates the forecasts produced by these methods alongside the actual test data.

- **Naïve Forecast:**
  Assumes all future values are equal to the last observed value in the training set:

$$\hat{y}_{t+h} = y_t \quad \text{for all } h > 0$$

  This model is especially useful in persistent or random walk processes and often performs reasonably well on financial data.

- **Mean Forecast:**
  Predicts all future values as the average of the historical data:

$$\hat{y}_{t+h} = \frac{1}{n} \sum_{i=1}^{n} y_i$$

  Best suited for stationary series with constant mean, though it underperforms on trending or seasonal data.

- **Drift Forecast:**
  Projects a linear trend forward based on the historical slope between first and last observations:

$$\hat{y}_{t+h} = y_t + \frac{y_t - y_1}{t-1} \cdot h$$

  Useful for capturing long-term trends in financial time series.

- **Seasonal Naïve Forecast:**
  Repeats the value from the same point in the previous seasonal cycle:

$$\hat{y}_{t+h} = y_{t+h-S}$$

  Where $S = 252$ is the seasonal period (one trading year). Included here, despite weak seasonality in the S&P 500 series, to test whether repetitive yearly patterns hold predictive power.

Together, these four models represent a spectrum of assumptions:

- **Persistence** (Naïve)
- **Mean-reversion** (Mean)
- **Trend-following** (Drift)
- **Seasonal replication** (Seasonal Naïve)

Their simplicity and interpretability allow us to test whether complex models like ARIMA, Prophet, or LSTM are justifiably superior, or whether much of the signal can be captured by basic heuristics.

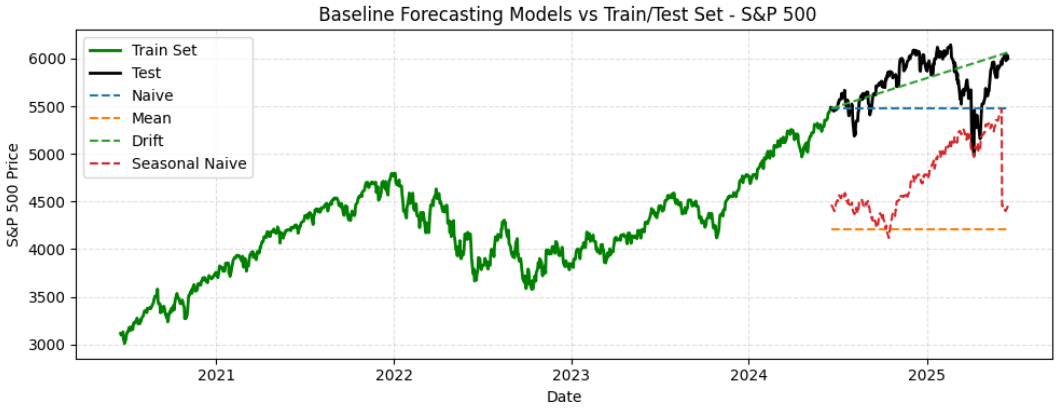

Figure 19: Baseline forecasting models vs. actual test set for the S&P 500.

Exponential Smoothing Models

In addition to the baseline models, three exponential smoothing methods were implemented to evaluate their capacity to forecast financial index data. These models are based on the principle of weighted averages, where recent observations receive more weight than older ones [4]. They are relatively easy to tune and interpret, and often serve as strong intermediate models between naïve baselines and fully parameterized approaches.

Simple Exponential Smoothing (SES)

SES is designed for time series data that exhibit no clear trend or seasonality. It forecasts future values using a weighted average of past observations:

$$\hat{y}_{t+1} = \alpha y_t + (1-\alpha)\hat{y}_t$$

Where:

- $\hat{y}_{t+1}$ is the forecast for the next time point
- y_t is the actual value at time t
- $\alpha \in (0, 1)$ is the smoothing parameter

The SES model was trained on the raw training series, with the optimal smoothing level α selected automatically. As seen in Figure 20, SES produced a flat forecast aligned with the most recent level, underreacting to the long-term upward trend of the S&P 500.

Holt's Linear Trend Method

Holt's method extends SES by adding a trend component to capture linear growth or decay. It uses two smoothing equations:

$$\begin{aligned} \ell_t &= \alpha y_t + (1-\alpha)(\ell_{t-1} + b_{t-1}) \\ b_t &= \beta(\ell_t - \ell_{t-1}) + (1-\beta) b_{t-1} \\ \hat{y}_{t+h} &= \ell_t + h b_t \end{aligned}$$

Where:

- ℓ_t is the estimated level
- b_t is the estimated trend
- α, β are smoothing parameters

As shown in Figure 20, Holt's method produced a linear extrapolation of the existing trend, continuing the upward momentum observed in recent years. While it captures the overall direction better than SES, it assumes a constant growth rate, which may lead to overestimation if the trend weakens or reverses.

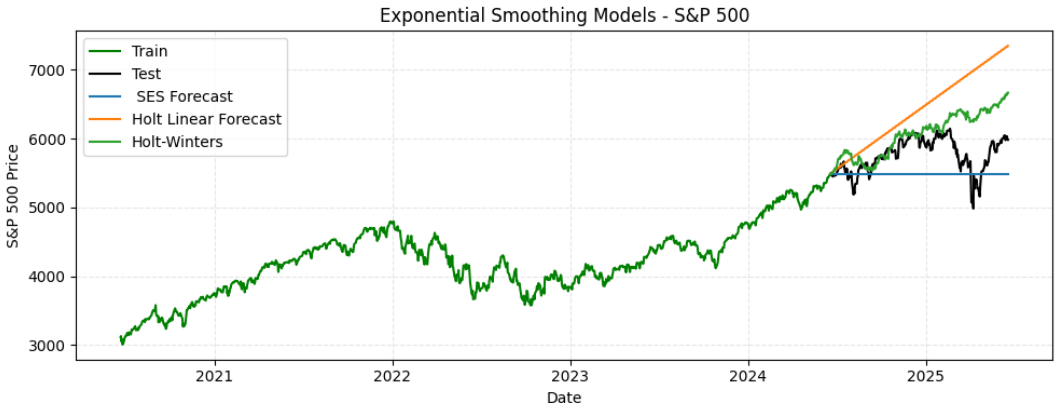

Figure 20: Combined exponential smoothing model forecasts for the S&P 500 test period. Includes SES (blue), Holt's Linear (orange), and Holt-Winters (green) forecasts.

Holt-Winters Seasonal Method

The Holt-Winters model (Triple Exponential Smoothing) adds a seasonal component in addition to level and trend. It uses the following system:

$$\ell_t = \alpha \frac{y_t}{s_{t-S}} + (1-\alpha)(\ell_{t-1} + b_{t-1})$$
$$b_t = \beta(\ell_t - \ell_{t-1}) + (1-\beta) b_{t-1}$$
$$s_t = \gamma \frac{y_t}{\ell_t} + (1-\gamma) s_{t-S}$$
$$\hat{y}_{t+h} = (\ell_t + h b_t) s_{t-S+h}$$

Where:

- $S = 252$ is the seasonal cycle length (business days in one year)
- s_t is the seasonal index
- α, β, γ are smoothing coefficients

Although decomposition in Section 2.1 showed weak seasonality, Holt-Winters was included for symmetry with Seasonal Naïve and to test whether even mild seasonality can enhance forecasts. As shown in Figure 20, the model attempts to replicate past upward surges based on seasonal patterns, but may overfit and misalign with true market conditions.

These three smoothing methods strike a balance between simplicity and flexibility. They capture level and trend (and optionally seasonality) without requiring stationarity or complex parameter tuning.

ARIMA Model

The Autoregressive Integrated Moving Average (ARIMA) model is a classical and widely used approach for univariate time series forecasting [5]. It combines three components: autoregression (AR), differencing (I), and moving average (MA), to model linear relationships and remove non-stationarity. The general form of an ARIMA model is denoted as ARIMA(p, d, q), where:

$$\Phi_p(B)(1-B)^d y_t = \Theta_q(B)\varepsilon_t$$

Where:

- $\Phi_p(B)$ is the AR polynomial of order p
- $\Theta_q(B)$ is the MA polynomial of order q
- B is the backshift operator ($By_t = y_{t-1}$)
- d is the number of differences needed to achieve stationarity
- ε_t is white noise

To satisfy the stationarity assumption of ARIMA, the raw S&P 500 series was first log-transformed to stabilize the variance and then first-order differenced to eliminate trend components, resulting in a transformed series $\Delta \log(y_{\text{train}})$.

Based on the ADF stationarity test and the autocorrelation structure observed in the ACF and PACF plots, a manual ARIMA(9,1,9) model was selected with the following parameters:

- $p = 9$: determined from significant spikes in PACF
- $d = 1$: required differencing to induce stationarity
- $q = 9$: suggested by persistent autocorrelations in ACF up to lag 9

The model was implemented using Python's `statsmodels` library and trained on the stationary log-differenced training series. Forecasts were generated in the transformed domain and then converted back to the original scale via exponentiation and cumulative summation.

Figure 21 displays the forecast output from this ARIMA model over the test period.

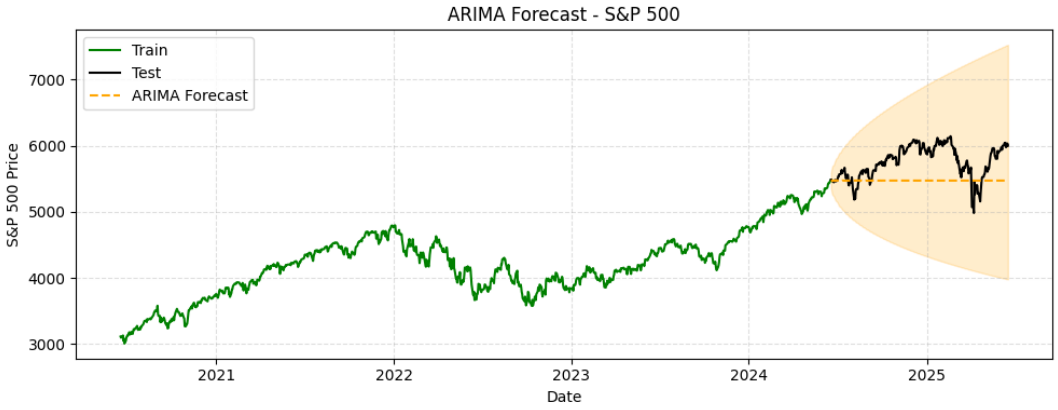

Figure 21: ARIMA(9,1,9) forecast of the S&P 500 test set. Shaded region shows 95% confidence interval.

While SARIMA (Seasonal ARIMA) was considered, it was ultimately not used due to two main reasons:

1. **Weak seasonality:** Decomposition analysis (Section 2.1) showed only modest seasonal effects, suggesting limited benefit from explicitly modeling seasonality.
2. **Computational cost:** SARIMA with $S = 252$ (daily financial seasonality) requires long seasonal periods and grid search, making training and tuning computationally expensive.

As a result, ARIMA was chosen as the most appropriate and efficient statistical model for this forecasting task.

Prophet Model

Facebook Prophet is a modular, additive time series forecasting model designed to handle trend, seasonality, and holiday effects in a decomposable structure [6]. Its strength lies in interpretability, robustness to missing values, and automatic changepoint detection. Prophet models the time series as:

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t$$

Where:

- $g(t)$: piecewise linear or logistic trend
- $s(t)$: seasonal components (e.g., weekly, yearly)
- $h(t)$: effects of holidays or events (not used in this case)
- ε_t: error term (assumed to be normally distributed)

The model was applied to the raw training series using Prophet's default configuration: linear trend with yearly seasonality. While the decomposition analysis in Section 2.1 indicated only weak seasonal behavior in the S&P 500 series, Prophet's flexibility and automatic trend changepoint detection provided strong justification for inclusion. Unlike ARIMA, Prophet does not require prior differencing or strict stationarity.

A key tuning parameter in Prophet is the **changepoint prior scale** (cps), which governs the model's responsiveness to trend shifts. For this implementation, the model was trained using:

- **cps = 0.01:** balances flexibility and generalization, allowing the model to detect moderate trend changes without overfitting to local fluctuations

Forecasting was performed over the 261-day test horizon. Figure 22 shows the model's prediction alongside actual values of the S&P 500.

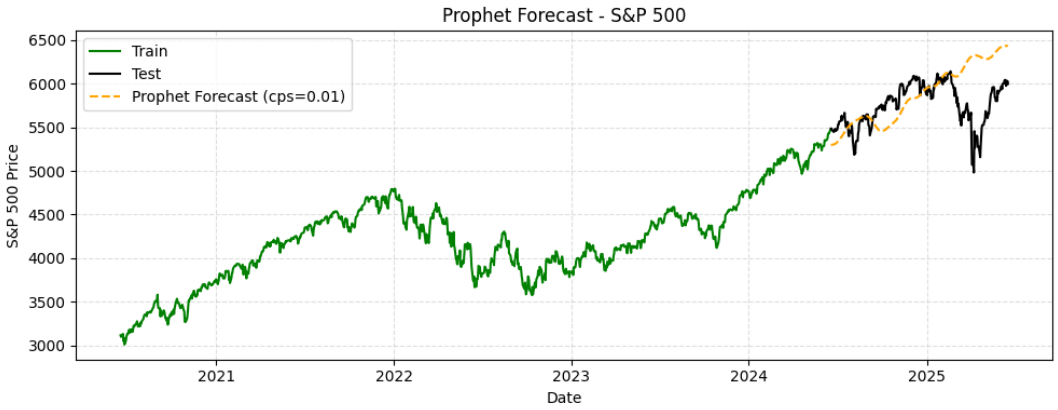

Figure 22: Prophet forecast of S&P 500 using cps = 0.01. Includes changepoint-sensitive linear trend with yearly seasonality.

Prophet was included not because of strong seasonal cycles, but to evaluate its adaptability in financial settings where non-linear growth, irregular volatility, and structural trend shifts dominate. Its built-in missing date handling, plotting utilities, and minimal preprocessing requirements make it a practical and scalable model for real-world financial forecasting tasks.

Deep Learning Models

In addition to statistical and rule-based models, this study explored deep learning approaches to capture complex, non-linear relationships in the S&P 500 time series. Deep learning models such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks are well-suited for sequential data [7], as they can learn temporal patterns directly from raw inputs without requiring extensive manual feature engineering.

These models are particularly advantageous when dealing with financial data, where dynamics often shift and traditional models may fail to capture subtle dependencies. The following subsections outline the implementation and configuration of both RNN and LSTM architectures used in this project.

RNN (Vanilla Recurrent Neural Network) Model

To provide a benchmark deep learning model, a standard RNN was also implemented. RNNs are designed to model sequential data by maintaining a hidden state h_t that evolves over time and is influenced by both the current input x_t and the previous hidden state h_{t-1}:

$$h_t = \phi(W_h h_{t-1} + W_x x_t + b)$$

Despite their conceptual simplicity, standard RNNs suffer from issues like the vanishing gradient problem, limiting their ability to learn long-term dependencies. However, they can still capture short-term patterns and serve as useful benchmarks.

Architecture:

- Layer 1: `SimpleRNN(64 units, ReLU, return_sequences=True)`
- Dropout(0.2)
- Layer 2: `SimpleRNN(32 units, ReLU)`
- Dropout(0.2)

- Dense(1)

The preprocessing, training settings, and inference procedure were kept identical to the LSTM model to ensure fair comparison. The RNN model is structurally simpler and faster to train, but expected to perform slightly worse due to reduced memory capabilities. The forecast output is shown in Figure 23.

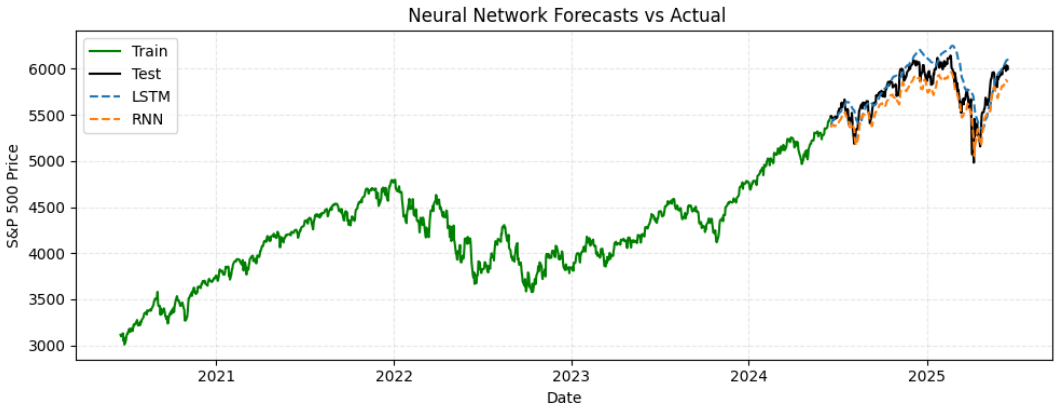

Figure 23: RNN forecast of the S&P 500.

LSTM (Long Short-Term Memory) Model

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) designed to overcome the limitations of standard RNNs in capturing long-term dependencies. LSTMs introduce internal memory cells and gating mechanisms that allow the model to retain information over extended time horizons, making them ideal for modeling complex time series data such as stock indices.

An LSTM cell computes its hidden state h_t and cell state c_t using the following key components:

- **Forget gate** f_t: determines what information to discard
- **Input gate** i_t: controls what new information to store
- **Output gate** o_t: determines the final output at each step

Architecture:

- Layer 1: `LSTM(64 units, ReLU, return_sequences=True)`
- Dropout(0.2)
- Layer 2: `LSTM(32 units, ReLU)`
- Dropout(0.2)
- Dense(1) output layer

Training Setup:

- Input: 30-day lookback sequences on normalized values
- Optimizer: Adam (learning rate = 0.001)
- Loss: Mean Squared Error
- Epochs: 50
- Batch size: 32
- Validation split: 10%

- Early stopping: patience = 5 (monitored on `val_loss`)

Preprocessing: MinMax normalization, fixed-length sliding windows, inverse transformation post-prediction to return to the original scale.

This structure allows the model to learn nuanced dependencies in the financial time series that traditional models may miss. The forecast results for the test period are shown in Figure 23.

2.3 Results

This section presents the implementation and evaluation outputs of all forecasting models applied to the S&P 500 index. Each model was trained using the training dataset (June 2020–June 2024) and evaluated on the 261-day test set (June 2024–June 2025). Forecasts were compared against actual index values using five standard performance metrics, with results summarized numerically and visually.

All models were implemented as described in Section 2.2. Each model generated point forecasts for the full test horizon. In cases where transformations (e.g., logarithmic scaling, differencing, normalization) were applied during training, inverse transformations were performed to return forecasts to the original scale. All models forecasted over the same horizon and training data, ensuring consistency in comparison.

Evaluation Metrics

To obtain a well-rounded view of model performance, five widely accepted evaluation metrics were employed:

1. **Mean Absolute Error (MAE)**:
$$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n}|y_t - \hat{y}_t|$$
Represents the average absolute difference between predicted and actual values.

2. **Root Mean Squared Error (RMSE)**:
$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2}$$
Penalizes larger errors more than MAE, making it sensitive to outliers.

3. **Mean Absolute Percentage Error (MAPE)**:
$$\text{MAPE} = \frac{100}{n}\sum_{t=1}^{n}\left|\frac{y_t - \hat{y}_t}{y_t}\right|$$
Expresses errors as a percentage of actual values; scale-independent.

4. **Symmetric Mean Absolute Percentage Error (sMAPE)**:
$$\text{sMAPE} = \frac{100}{n}\sum_{t=1}^{n}\frac{|\hat{y}_t - y_t|}{(|y_t| + |\hat{y}_t|)/2}$$
Normalizes over the average of actual and predicted values for stability.

5. **Mean Absolute Scaled Error (MASE)**:
$$\text{MASE} = \frac{\text{MAE}_{\text{model}}}{\text{MAE}_{\text{naïve}}}$$
Compares model MAE against a naïve benchmark. Values < 1 indicate better-than-naïve performance.

By incorporating these metrics, we ensure that absolute and percentage-based deviations are considered, and that sensitivity, scale independence, and interpretability are factored into the model comparison.

Table 1: Forecasting Performance on the S&P 500 Test Set

Model	MAE	RMSE	MAPE (%)	sMAPE (%)	MASE
LSTM	123.14	165.34	2.17	2.14	3.90
RNN	134.30	148.37	2.32	2.35	4.25
Drift	186.85	250.44	3.30	3.25	5.92
Prophet	287.15	392.43	5.10	4.91	9.09
Naïve	313.00	369.41	5.33	5.52	9.91
ARIMA	313.21	369.62	5.33	5.52	9.92
SES	313.29	369.71	5.34	5.52	9.92
Holt-Winters	324.78	447.82	5.75	5.45	10.29
Holt Linear Trend	532.68	696.58	9.36	8.66	16.87
Seasonal Naïve	994.09	1070.43	17.18	19.08	31.48
Mean Forecast	1538.60	1557.95	26.62	30.89	48.73

2.4 Discussion

This section interprets the results presented in Section 2.3 by evaluating each model's performance in terms of accuracy, behavior, and robustness. It also discusses model limitations and highlights unexpected findings that emerged during experimentation.

The LSTM model clearly delivered the best performance across nearly all metrics. Its ability to capture long-term dependencies and model non-linearities in the S&P 500 series resulted in the lowest MAE, MAPE, and sMAPE scores. The model closely tracked actual price levels and adjusted well to market fluctuations, indicating strong generalization. The RNN model, while slightly less accurate than LSTM, still outperformed all traditional models. Its simpler architecture made it less capable of capturing long-term dependencies, but it successfully modeled short-term dynamics. These results confirm that deep learning models are especially effective for non-stationary, high-volatility financial series, where classical assumptions of linearity or fixed seasonality often fail.

The ARIMA(9,1,9) model, despite proper stationarity correction and informed parameter tuning, did not outperform the naïve forecast. This suggests that ARIMA struggled to adapt to the market's regime shifts and non-linear behavior, especially without a seasonal component. The prediction interval widened significantly over time, further reflecting ARIMA's increasing uncertainty when forecasting long horizons.

The Prophet model, although designed to adapt to changepoints and trend shifts, also failed to offer a major advantage over naïve or drift models. Its assumption of additive components and reliance on minimal seasonality limited its performance in this financial context. Nonetheless, it served well as an interpretable and automated modeling tool.

All exponential smoothing models underperformed, with Holt Linear and Holt-Winters showing clear over- or under-projection. Holt Linear produced overly aggressive trend extrapolation, while Holt-Winters attempted to capture seasonal cycles that weren't strongly present, leading to overfitting. The Simple Exponential Smoothing (SES) model offered little value in this trending series due to its flat-level nature. These models are better suited to stationary or clearly seasonal data, which was not representative of the S&P 500 in the studied timeframe.

The Naïve and Drift models provided useful baselines. Drift captured the overall upward trend but lacked volatility adaptation. The Seasonal Naïve and Mean Forecast models recorded the highest errors, reaffirming their role as baseline comparators rather than serious forecasting tools.

The performance of ARIMA matched the naïve model, even after careful tuning. This was unexpected given ARIMA's popularity in financial applications. It highlights that proper model choice depends on data structure, not on popularity or theoretical complexity.

The Prophet model underperformed despite being trend-aware. This may be due to its assumption of clean changepoints and limited seasonality, both of which are weak or unstable in market indices.

Despite their simplicity, RNNs held their ground against ARIMA and Prophet, showcasing their viability even when not using extensive architectures like LSTM.

LSTM and RNN require careful hyperparameter tuning, and their performance depends heavily on data normalization, window size, and early stopping. While they offer accuracy, they also demand significant computational resources.

ARIMA and exponential models are fast to train but rigid in structure and unable to adapt to sudden market shocks or structural breaks.

Prophet offers automation and interpretability but is limited when data lacks strong periodicity or consistent changepoints.

Overall, the analysis supports the use of deep learning models, particularly LSTM, for forecasting financial indices under uncertain and non-stationary conditions. Classical statistical models remain valuable for quick prototyping or when interpretability is prioritized, but their limitations must be acknowledged in dynamic environments such as stock markets.

Conclusion

This paper applied a structured time series forecasting workflow across two distinct financial datasets: Bitcoin (BTC) and the S&P 500 Index, addressing the forecasting challenges in engineering and management domains. The project was divided into two main parts — each tailored to the specific nature and modeling maturity of the respective asset.

In the Bitcoin analysis, the focus was on exploratory data analysis (EDA) and pre-model diagnostics. Due to high volatility and limited structural consistency, the work concentrated on uncovering statistical properties, checking for stationarity, and visualizing patterns using decomposition and autocorrelation functions. While forecasting models were not applied, this phase laid the foundation for advanced modeling by transforming the dataset into a stationary format compatible with ARIMA-type models.

The second part of the study, focusing on the S&P 500 index, implemented a full forecasting pipeline. A wide range of models was tested — from statistical baselines and exponential smoothing to ARIMA, Prophet, and deep learning architectures like RNN and LSTM. Model performance was rigorously evaluated using multiple error metrics and visual comparisons. The LSTM model emerged as the most accurate, followed by RNN and Drift, while classical models like ARIMA and Prophet offered limited predictive power for this dataset.

Across both parts, the project demonstrates how forecasting in engineering and financial domains requires a careful balance between statistical rigor and adaptability. Stationarity, model assumptions, seasonality, and volatility all play critical roles in determining which methods succeed and where traditional tools may fall short.

Ultimately, the work highlights the strengths of deep learning for dynamic financial series, the enduring value of exploratory diagnostics, and the importance of methodical model evaluation in engineering-focused forecasting tasks.

Appendix

All forecasting workflows, data preprocessing steps, visualizations, and evaluation metrics presented in this paper were developed and executed using Python in Jupyter Notebooks. The project is divided into two core notebooks:

- **FoEM_Self_Collection.ipynb** – Exploratory analysis and stationarity testing of Bitcoin (BTC) time series.
- **FoEM_Projekt.ipynb** – Full forecasting pipeline and model evaluation for the S&P 500 Index.

To support efficiency and ensure reproducibility, AI-assisted tools (e.g., code generation via ChatGPT) were utilized during implementation. These tools helped streamline code development, accelerate visual analysis creation, and maintain consistency in model evaluation workflows.

All source notebooks, datasets, and plots are available upon request or can be accessed through the project repository:

`github.com/rhkraptor/FoEM`

References

[1] Selva Prabhakaran, *Augmented Dickey-Fuller Test (ADF Test) — Must Read Guide*, 2021. Available at: `https://www.machinelearningplus.com/time-series/augmented-dickey-fuller-test/?utm_source`.

[2] Robert Nau, *ARIMA models for time series forecasting*, Duke University. Available at: `https://people.duke.edu/~rnau/411arim3.htm?utm_source`.

[3] Egor Howell, *Baseline Models in Time Series*, Medium, 2022. Available at: `https://medium.com/data-science/baseline-models-in-time-series-c76d44a826b3`.

[4] Milvus Team, *What Are Exponential Smoothing Methods in Time Series Analysis?*, Milvus IO, 2023. Available at: `https://milvus.io/ai-quick-reference/what-are-exponential-smoothing-methods-in-time-series-analysis`.

[5] Investopedia Team, *Autoregressive Integrated Moving Average (ARIMA)*, Investopedia, 2023. Available at: `https://www.investopedia.com/terms/a/autoregressive-integrated-moving-average-arima.asp`.

[6] Chandrasekaran, V. and Ragavan, K., *Stock Price Prediction using Facebook Prophet*, ResearchGate, 2022. Available at: `https://www.researchgate.net/publication/360387298_Stock_Price_Prediction_using_Facebook_Prophet`.

[7] Alyoshina, I. and Kolchanova, E., *Forecasting Methods for Financial Time Series*, SHS Web of Conferences, Vol. 194, 2024. Available at: `https://www.shs-conferences.org/articles/shsconf/pdf/2024/16/shsconf_edma2024_02006.pdf`.