Project work 2: Time Series Analysis of BTC/USD Exchange Rate

Alessia Tani, Iiro Vendelin, Sara Zambetti

Sampling Frequency and Continuity of the Time Series

Although the BTC/USD dataset is nominally sampled at one-minute intervals, the raw observations are not perfectly continuous. After converting timestamps to `datetime` format and ordering the data chronologically, the time gaps between consecutive points were inspected. The median step is one minute, but several timestamps are missing and the spacing between observations is not always constant, meaning the series cannot be regarded as strictly sampled at a fixed rate.

To obtain a time series suitable for autoregressive and recurrent models, the sampling frequency was regularised by constructing a uniformly spaced time grid based on the median observed step and resampling the data with MATLAB's `retime` function. This procedure aligns observations to the new grid, fills temporal gaps, and aggregates multiple points falling within the same interval using the mean, resulting in a uniformly sampled series with one observation per timestamp.

Temporal Synchrony and Time Series Alignment

The BTC/USD dataset used in this project is univariate, as only the *Close* price is analysed. As a result, issues of synchrony across variables do not arise: each observation is a single value associated with a timestamp, and the series is trivially synchronous.

In a multivariate setting, however, different variables (such as price, volume, volatility indicators, or external signals) are often recorded at different frequencies or with misaligned timestamps. In such cases, temporal synchronisation becomes essential to ensure that all variables refer to the same point in time. A typical strategy is to resample each series to a common resolution (for example, one minute or one hour) using suitable aggregation rules, and then define a shared time grid on which all variables are expressed. Any gaps created by this alignment can be treated through interpolation or filling schemes, while functions such as `synchronize` and `retime` in MATLAB automate much of this process by aligning multiple timetables on common timestamps with user-specified aggregation or interpolation methods.

Although these procedures were not required for the present univariate series, they are standard tools for enforcing synchrony in multivariate time series and would become necessary if additional variables were incorporated into the modelling pipeline.

Missing Values and Strategies for Handling Incomplete Data

As part of the initial preprocessing, the dataset was inspected for missing observations in the *Close* price column. Although financial time series rarely contain explicit `NaN` values, the dataset may still exhibit implicit missingness caused by gaps in the timestamps. To address this form of missingness, rows containing undefined numerical values were removed using MATLAB's `rmmissing` function, ensuring that no observation with an invalid price entered the subsequent analysis.

The amount and structure of the missing values were then quantified on the regularised one-minute series. In total, 1160 missing observations were detected in the *Close* price, corresponding to approximately 0.0159% of the complete dataset. The analysis of consecutive `NaN` segments showed the presence of exactly one missing block, with a maximum block length of

1160 consecutive minutes (about nineteen hours of data). Thus, even though the proportion of missing data is very small in relative terms, it is concentrated in a single contiguous gap.
In this project, following the initial preprocessing and temporal regularisation steps, the BTC/USD series at minute-level granularity contained a single consecutive block of missing values. These missing points were filled by means of linear interpolation on the price process. Denoting by P_{t-k} and P_{t+m} the nearest observed prices before and after the gap, and by k and m their respective distances from an intermediate time index t within the missing block, the interpolated price P_t was computed as

$$P_t = P_{t-k} + \big(P_{t+m} - P_{t-k}\big) \cdot \frac{t-(t-k)}{(t+m)-(t-k)}. \tag{1}$$

This construction assumes a smooth evolution of prices over the missing period and amounts to connecting the endpoints of the gap with a straight line in the time–price plane. The same linear interpolation scheme was applied consistently both to the one-minute series (after regularisation) and to the hourly series used for decomposition and autocorrelation analysis. Through the combination of row-level cleaning, temporal regularisation and interpolation, the final BTC/USD series used in this project is complete, continuous, and suitable for subsequent modelling tasks.

Outlier Detection Through STL Decomposition

An STL decomposition was applied to the last twelve months of hourly BTC/USD data to detect anomalous observations. STL separates the series Y_t into trend, seasonal, and remainder components:

$$Y_t = T_t + S_t + R_t, \tag{2}$$

where T_t captures long-term movements, S_t models the weekly seasonal cycle ($P = 168$ hours), and R_t contains the unexplained residual fluctuations. Since R_t isolates deviations not accounted for by trend or seasonality, it provides a natural basis for identifying potential outliers. The residuals were standardised via

$$z_t = \frac{R_t - \mu_R}{\sigma_R}, \tag{3}$$

with μ_R and σ_R denoting their sample mean and standard deviation. Values with $|z_t| > 3$ were marked as outliers, following a conventional significance threshold ($p < 0.003$) under approximate Gaussianity.
The analysis revealed a cluster of abnormal values early in the year (147), indicating a localised deviation that was inconsistent with the underlying behaviour of the series. These points were considered outliers because they could not be justified by normal fluctuations once trend and seasonality were removed.
Once identified, outliers may be either removed or replaced. In this analysis, a cleaned version of the series was reconstructed by substituting the flagged points with the sum of the trend and seasonal components estimated by STL, that is

$$Y_{\text{clean},t} = T_t + S_t \quad \text{for flagged indices } t. \tag{4}$$

This procedure preserves the natural temporal structure of the signal while eliminating distortions that could negatively affect forecasting models.
Although the code relies on the cleaned series for model estimation, financial time series modelling requires careful treatment of extreme events. From a statistical perspective, removing

outliers and working with a smoothed series can lead to more stable parameter estimates and clearer diagnostics. However, from a financial perspective, large price jumps represent genuine market dynamics that are crucial for risk management and volatility assessment. For this reason, the cleaned series is used primarily for model development and stability checks, while the original series, with outliers retained, is employed in the final evaluation phase to ensure that model performance is assessed under realistic market conditions. Ignoring tail events would otherwise produce models that appear well behaved statistically, but are insufficiently exposed to the risks present in real-world financial data.

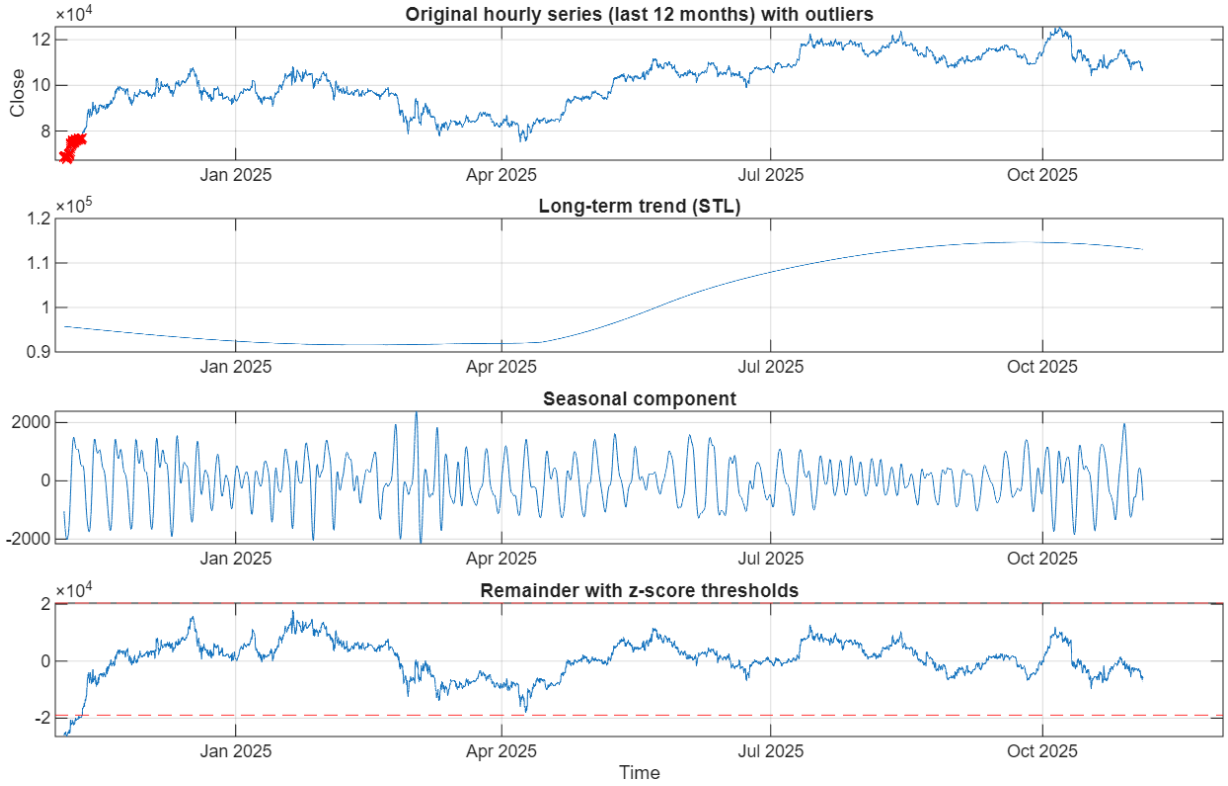

Figure 1: STL decomposition and detected outliers in the remainder component.

In the top panel, the original series is displayed together with red markers highlighting observations flagged as outliers from the STL remainder; these form a noticeable cluster at the beginning of 2025.
The second panel reports the long-term trend, which captures the smooth downward movement until spring 2025, followed by a recovery in summer and a stabilisation towards the end of the year.
The third panel shows a strong seasonal component with regular daily and weekly cycles, confirming the presence of pronounced seasonality in the data.
The bottom panel displays the remainder together with $\pm 3,\sigma$ thresholds used for outlier detection: points outside this band correspond to anomalous deviations that are not explained by trend or seasonality.

Baseline Autoregressive Model

To provide a reference for evaluating more advanced recurrent models, a baseline autoregressive model was constructed. The hourly *Close* price was used as the target variable, and a set of lagged observations was employed as predictors. In particular, an autoregressive structure of

order 24 (AR(24)) was chosen, corresponding to one full day of hourly lags, a common choice for financial or seasonal hourly time-series.
A regressor matrix was built by stacking the previous 24 values of the series for each time step, while the corresponding target vector contained the subsequent value. A standard linear regression model was then fitted on the training portion of the dataset, and one-step-ahead predictions were evaluated on both the training and the test sets.
The model achieved a training MSE of approximately 139208.21 and a training MAE of 248.34. On the test set, the MSE was 117039.27 and the MAE was 226.14. These results show that, despite its simplicity, the AR(24) model is capable of capturing a substantial part of the short-term dynamics of the hourly BTC/USD series. The test–set plot confirms the close alignment between the predicted and actual trajectories, with discrepancies appearing mainly during sharper movements, which is expected for a purely linear autoregressive predictor.

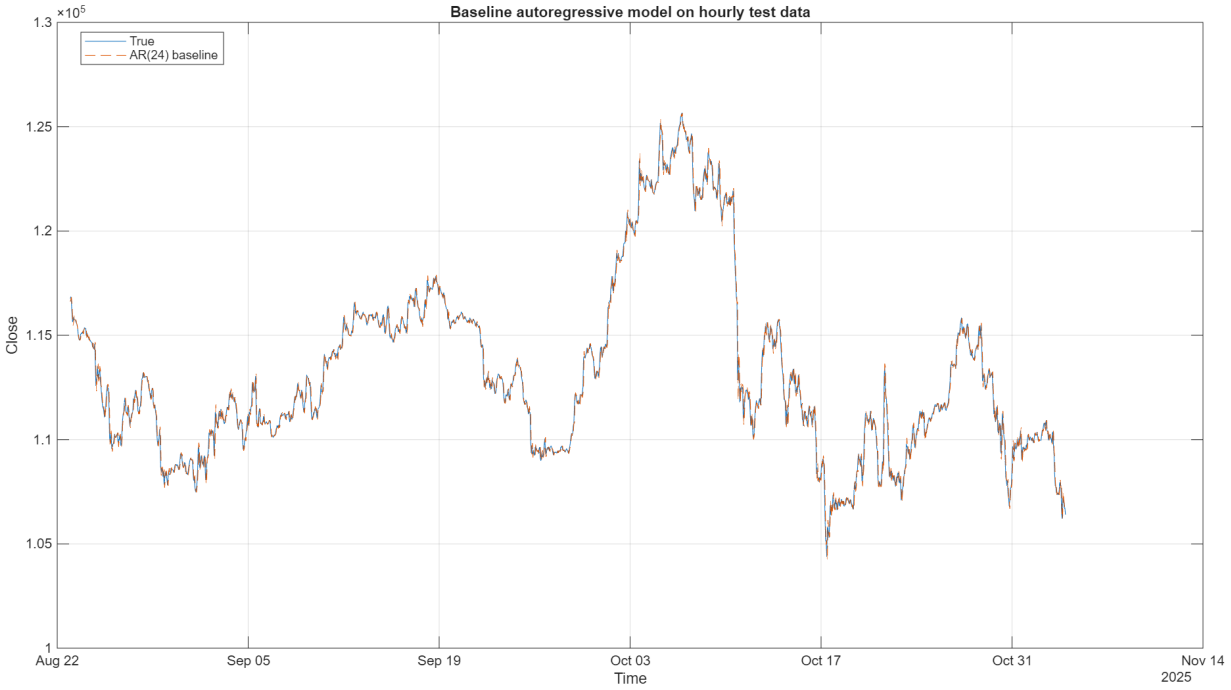

Figure 2: Baseline autoregressive model on hourly test data.

Literature Review for LSTM

Dividing long time-series into sub-sequences is a standard strategy for preparing data for machine learning models, particularly recurrent neural networks such as LSTMs. In sliding window or block partitioning methods, the time-series is split into overlapping or non-overlapping windows to generate input-output pairs for training [1]. The length of these windows should consider underlying seasonality, as failing to include a full seasonal pattern within a sequence can reduce model accuracy [2]. Proper window design helps ensure that models capture both short- and long-term dependencies across cycles present in financial or industrial time-series.
Seasonality and trend can be addressed directly within LSTM and other deep learning models by providing the network with decomposed components (such as deseasonalized and detrended series, or by explicitly adding time indices as features) [2]. However, literature also suggests that modern LSTM architectures with sufficient historical context can, in many cases, learn periodic patterns implicitly [4]. Preprocessing with seasonal-trend decomposition and considering these components in feature engineering can still be beneficial, especially for data with strong calendar effects.

Standardization of input data is critical for stable and efficient training of LSTMs. Common practice involves z-score normalization (subtracting the mean and dividing by the standard deviation) or min-max scaling to $[0, 1]$ or $[-1, 1]$ intervals [3]. For non-stationary series, rolling-window normalization is sometimes preferred to account for local shifts in mean and variance [2]. These preprocessing steps align variable scales and help neural networks converge more rapidly, preventing features with large absolute values from dominating the learning process.

References

[1] J. Brownlee, "How to Prepare Sequence Data for Deep Learning," *Machine Learning Mastery*, 2018.

[2] K. Benidis et al., "Neural Forecasting: Introduction and Literature Overview," *International Journal of Forecasting*, vol. 36, no. 3, pp. 718-746, 2020.

[3] K. Bandara, C. Bergmeir, and S. Smyl, "Forecasting Across Time Series Databases Using Neural Networks on Groups of Similar Series: A Review," *Expert Systems with Applications*, vol. 140, 2020.

[4] N. Laptev, J. Yosinski, L. Li, and S. Smyl, "Time-series Extreme Event Forecasting with Neural Networks at Uber," *International Conference on Machine Learning (ICML) Time Series Workshop*, 2017.