Applied Machine Learning: Predictive Analysis of Financial Time Series

Carlo Cetrone

October 10, 2025

Abstract

The short-term prediction of value fluctuations in financial instruments has been a significant research area for decades, both for the academic world and for market operators. This study evaluates the predictive effectiveness of machine learning algorithms applied to historical datasets related to different financial instruments, with the aim of algorithmically identifying correlations, trends, and recurring patterns. The investigation aims to determine the impact of different data analysis techniques, including input selection, processing, and normalization, noise reduction, and output optimization, on the predictive performance of artificial neural networks. Through an experimental approach based on historical data, various methodological configurations are tested to quantify the contribution of each factor to improving prediction accuracy.

1 Introduction

The realization of this study involves implementing a series of procedures using code written in Python. Part of this code uses mathematical and logical tools natively implemented in the development environment; in other cases, however, due to the complexity of the operations to be performed or the need to access external databases, libraries and modules developed by third parties are used, whose documentation is easily found through links to their respective web pages. The code used in each phase of the study is fully available and, for simplicity, all mentioned functions and classes have been collected in a single module named `project_functions`.

1.1 Data Retrieval

The first procedure necessary for carrying out the study consists of retrieving the historical series on which to perform the analyses. For this purpose, the Python library `yfinance` is used, an open-source library downloadable directly from the terminal, which allows importing historical time series data as lists. All related documentation is available on the library's official page[1]. To ensure the availability of the maximum possible amount of data for analysis, the time window for the data to be downloaded is set to the entire quotation period provided by `yfinance` for the considered instrument, while the sampling interval is varied from time to time to verify its impact on the model's predictive capabilities.

1.2 Data Analysis

The operations of data selection, formatting, noise reduction, and normalization, necessary for producing effective datasets, are performed mainly using tools natively implemented in Python; in some cases, however, to speed up processes and improve code readability, commands from external modules like `math` and `numpy` are used. The objective of these operations is the production of homogeneous, normalized input and output sets, organized into two ordered lists of the same cardinality, through which the predictive model is trained. Herein lies the very essence of the study, which aims, through the definition of an input selection and output association technique, to optimize the model's predictive performance.

[1]`https://ranaroussi.github.io/yfinance/`

1.3 Machine Learning

The tools necessary to correlate the input and output sets and, subsequently, generate predictions, are algorithms of varying complexity belonging to the branch of artificial intelligence known as machine learning, whose detailed discussion is beyond the scope of this study. In this specific case, algorithms known as neural networks are used. A complete and effective presentation of how these algorithms work is available on the official IBM website[2] . These algorithms are implemented using the Python library called `PyTorch`, an open-source library whose documentation can be consulted on its dedicated webpage[3] . The techniques and methods for implementing the neural network model are illustrated in the dedicated section.

1.4 Reliability Verification

In order to effectively evaluate the predictive performance of the trained model, it is essential to distinguish between the data used during the learning process and those used to verify the reliability of the predictions. To this end, the input and output sets are divided into two distinct sections, to be used at their respective times.

The verification of the model's reliability consists of comparing the predictions generated for each element of the input set with the corresponding element of the output set.

To quantify the model's effectiveness, mathematical tools implemented directly in Python are used, along with additional libraries such as the aforementioned `math` and `numpy`, and a very useful library for graphically visualizing data, especially if numerous, called `matplotlib`. Specifically, scatter plots will be used to correlate each model prediction with its corresponding output, bin plots (interval graphs) to highlight the average value represented by the scatter plot, and backtest graphs, useful for simulating a real operational situation in the markets.

[2]`https://www.ibm.com/it-it/topics/neural-networks`

[3]`https://docs.pytorch.org/docs/stable/index.html`

2 Implementation

This section illustrates the techniques and methods by which each phase of the study, previously described, is concretely implemented. The objective is to provide a complete and detailed view of the operational steps, so as to make not only the methodological choices adopted more understandable, but also their impact on the study's outcomes.

2.1 Data Selection

The first selection performed on the data provided by the `yfinance` library concerns the category of the financial instrument. In particular, time series relating to the values of stocks, currencies, cryptocurrencies, and commodities are analyzed. For each of these categories, three specific instruments are chosen to ensure a representative sample for subsequent analyses.

Table 1: Instruments Grouped by Category
The table lists the specific instruments selected for each category - stocks, currencies, cryptocurrencies, and commodities - to ensure representative datasets for the subsequent analyses.

Category	Selected Instruments
Stocks	Apple, Intel, Amd
Currencies	Euro, Pound, Yen
Cryptocurrencies	Bitcoin, Ethereum, Solana
Commodities	Gold, Silver, Oil

using the `download_price_history` function, defined within the `project_functions` module, we retrieve the time series for the value of the instrument we want to examine; this function receives 3 strings as arguments that allow us to select the instrument, the period (the length of the time series), and the sampling interval (time elapsed between one value in the series and its subsequent one). To provide an idea of how to use this function, the call `download_price_history('AAPL','20y','1d')` returns a string containing the daily sampled values of Apple stock over the last 20 years. If we were to graphically represent the time series just obtained using the `matplotlib` library, we would get the following chart.

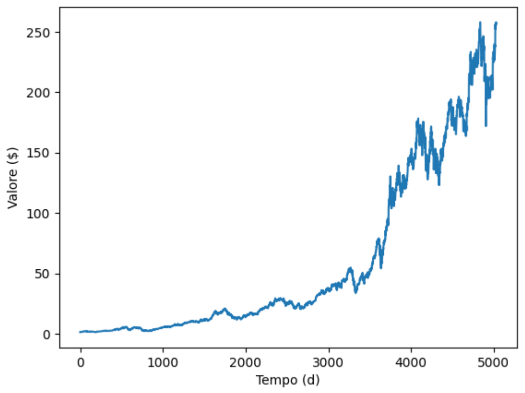

Figure 1: Chart of Apple stock value over the last 20 years

In order to present a complete and in-depth analysis, the predictive performance obtained using time series characterized by sampling intervals of one minute, one hour, and one day will be compared.

2.2 Data Processing

From the chart in Figure 1, it is noted that the historical series has an exponential growth trend. To make the data more manageable, a natural logarithm transformation is applied to each value in the series, thus obtaining a new series (Figure 2) that is more suitable for analysis and has some characteristics that will be useful as the study progresses.

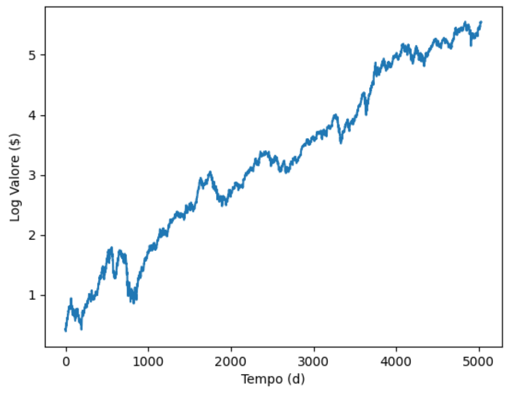

Figure 2: Chart of the natural logarithm of Apple stock value over the last 20 years

Once this operation is complete, it is necessary to transform the obtained time series into an input and output dataset, suitable for training and validating the predictive models. For this purpose, two parameters are introduced:

`observation_window`: indicates the length of the input sequences

`prediction_window`: indicates the length of the output sequences

The input set contains all consecutive sequences of length equal to `observation_window`. The output set contains, for each input sequence, the immediately following sequence of length equal to `prediction_window`. For convenience, the last element of the input sequence is repeated as the first element of the output sequence. In this way, the two sets (input and output) have the same size, and the model can be trained to recognize correlations between the two. The values of the `observation_window` and `prediction_window` variables can be selected during the course of the study to evaluate their impact on the trained model's performance.

Two examples of input (blue) and output (orange) sequence plots for `observation_window=64` and `prediction_window=4` are shown in Figure 3.

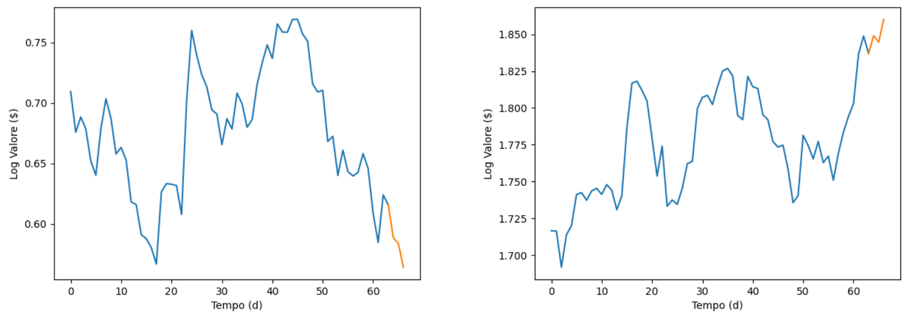

Figure 3: Plot of two general input sequences and their corresponding outputs

As observed in the two plots in Figure 3, sequences derived from different sections of the historical time series have values belonging to very different ranges; this represents a problem for training a predictive model, which, to be effective, must be trained on similar inputs and outputs. For this purpose, we perform a normalization of the inputs and outputs.

Regarding the input sequences, there are several normalization techniques that can be adopted. The first, known as min-max normalization, consists of scaling every value in the sequence - which, for convenience, we denote as S - to the interval $[0, 1]$ by defining a new numerical series $S_{norm} = \{\frac{x-min(S)}{max(S)-min(S)} \forall x \in S\}$.

In this way, all sequences have the same minimum (0) and the same maximum (1). However, information about the real width of the original interval is lost.

The sequences seen in Figure 3, normalized using this method, are shown in Figure 4.

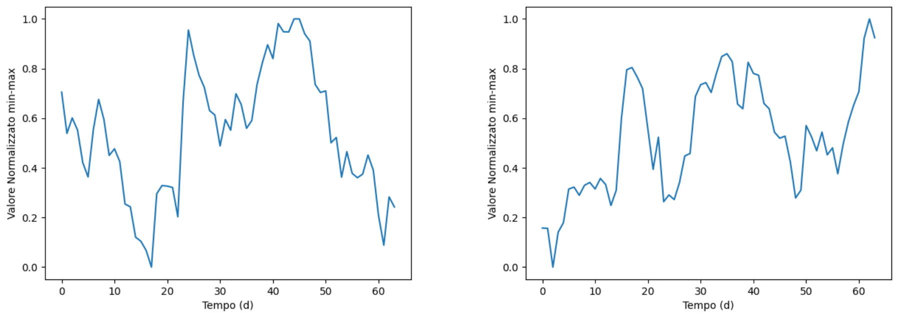

Figure 4: Plot of the sequences in Figure 3 normalized (min-max)

An alternative to the proposed normalization consists of defining $S_{norm} = \{x - a \ \forall x \in S\}$ where a is the last element of S. Thanks to the previous application of the logarithm, this technique allows preserving information about volatility, but does not guarantee that the sequences will have the same maximums and minimums. It does, however, guarantee that at least one value in the sequence (the last one) is always equal to 0.

The sequences seen in Figure 3, normalized using this method, are shown in Figure 5.

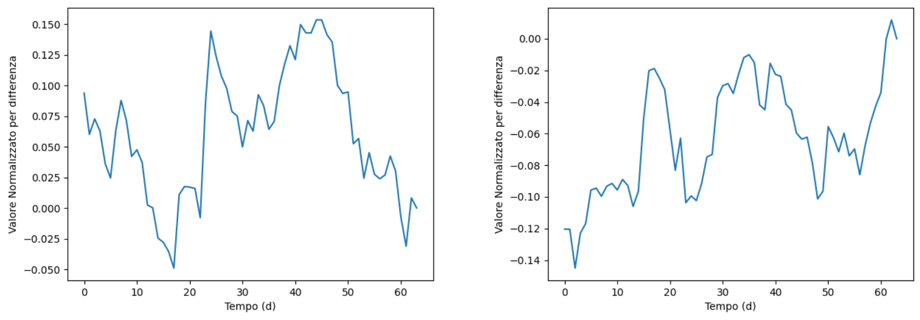

Figure 5: Plot of the sequences in Figure 3 normalized (logarithmic difference)

Regarding the normalization of output sequences, it would be possible to try to predict every single value of the sequence. However, this choice would significantly complicate the evaluation of the model's accuracy. To simplify the process, each output sequence is reduced to a single value, calculated as the difference between the first and last element of the sequence itself. In this way, each sequence in the input set is associated with the difference between the logarithms of the values that the instrument assumes at the end and at the beginning of the prediction window, respectively.

Since, as is known, the difference between logarithms preserves information about percentage change, the outputs can be considered normalized.

2.3 Model Selection

As mentioned earlier, a detailed explanation of how the machine learning algorithms offered by the PyTorch library work is outside the scope of this study. Here, we limit ourselves to using them for their mathematical properties, without delving into the theoretical mechanisms behind them.

As a starting point, feedforward models will be used, chosen for their simplicity in both concept and implementation. These models, in fact, require a relatively low computational load and allow for an immediate verification of the effectiveness of the data preparation and normalization procedures described in the previous sections. The model just described is implemented within the `project_functions` module through the class named `FeedforwardNeuralNetwork`. The specific technical choices made during the implementation of this model aim to ensure sufficient complexity for processing the data we will submit to the model and a training time adequate for the study we are conducting.

Subsequently, the analysis can be extended to more complex and advanced models, such as the Transformer. This model is particularly suited to the project due to its architecture, based on attention mechanisms, which allows it to capture long-term relationships within time series, going beyond the limitations of feedforward models; this analysis, although interesting, would require significantly higher computational resources and is therefore omitted from this study.

2.4 Model Training

The model training phase is the one richest in technicalities and algorithms, the explanation of which, as already mentioned, is beyond the scope of this study. For now, it is sufficient to know that the number of epochs indicates the number of times the entire training process is repeated. Increasing the number of epochs generally allows the model to refine its predictive capabilities on the data it has been exposed to previously; however, this does not guarantee an improvement in predictive performance on datasets external to the training process. On the contrary, an excessive number of epochs can lead to a phenomenon known as overfitting, where the model achieves very high accuracy on the training data but loses its ability to generalize, showing poor performance in making predictions on new data. For this reason, during training, it is essential to find a sufficient balance between learning and generalization. To this end, we initialize two variables `epochs` and `train_ratio`; The first allows us to vary the number of epochs during the study, while the second is used to divide the generated dataset into two parts, the first to be used during the training phase, the second during the validation and backtest phase.

2.5 Model Evaluation

For the purpose of evaluating the model's performance, we implement several quantitative analysis tools in Python. First, we use a scatter plot that allows us to represent the observed outputs as a function of the model's corresponding prediction values, providing an immediate qualitative measure of the degree of correlation between the two. Since the dispersion of the data can make a direct visual assessment difficult, we then apply a binning procedure to the predictions, dividing them into intervals and calculating, for each interval, the average of the observed outputs. This graphical representation helps to highlight the average relationship between prediction and observed output, reducing the noise due to the high variability of the sample. Finally, to evaluate the model's real performance during the evaluation period, we implement an operational simulation (backtest): the cumulative return of the strategy derived from the model's indications is compared with that of a "buy and hold" benchmark strategy and with the inverse strategy. This approach allows us to analyze not only the predictive ability in the strict sense but also the robustness and usefulness of the predictions in a real decision-making context.

To carry out this simulation, the average value of the predictions made by the model on the training dataset is considered. Subsequently, the cumulative return obtained from a strategy that involves investing when the model's predictions on the validation dataset exceed this average value and divesting when, on the contrary, they are lower is calculated. The return obtained is compared with that of the opposite strategy and with the "buy and hold" benchmark (overall increase during the validation period).

3 Data Collection and Interpretation

Summarizing what has been said in the previous section, the objective of this study is to verify the possibility of finding a significant correlation between the input and output sequences we have defined through the implemented machine learning algorithms and, if successful, to quantitatively evaluate the impact that each initialized parameter has on the predictive capabilities of these algorithms. The initialized parameters that can be adjusted are as follows.

Table 2: Optional Parameters for Model Initialization
The table lists the parameters that can be adjusted during the experimental phase and their respective tested value ranges, which influence the predictive performance of the neural networks.

Parameter	Values to Test
Instrument	Stocks, Currencies, Cryptocurrencies, Commodities
Interval	1 Minute, 1 Hour, 1 Day
Observation Window	32, 64, 128
Prediction Window	2, 4, 16
Normalization Technique	min-max, logarithmic difference
Epochs	64, 256, 512, 1024, 4096

It is recalled that, since the first value of the output sequence coincides with the last of the input sequence and the output is normalized by calculating the difference between the last and the first value of the sequence, for `prediction_window` values of 2, 4, and 16, the relative change observed corresponds to 1, 3, and 15 values, respectively, after the end of the input sequence.

As is evident, it will not be possible to exhaustively test every combination of values that can be assigned to the listed parameters, as this would be redundant and not functional for the research aims. Consequently, targeted choices will be made during the study, justifying the adoption of specific values over others. These decisions will be aimed at ensuring, on one hand, the completeness of the analysis, by including a sufficiently representative set of configurations, and on the other hand, the conciseness of the study, by avoiding redundancies and focusing attention on the aspects that are truly significant for evaluating the predictive performance of the trained models.

3.1 Comparison of Normalization Techniques

In the first phase of data collection, we want to determine which normalization technique is most effective for our purposes, since we can assume that a better normalization will have better performance regardless of the values selected for the other parameters. For this purpose, we set an intermediate value for the observation window (64) and for the sampling interval (1h), the minimum for the prediction window (2), as this facilitates operations during the model backtesting phase, and an intermediate value for the epochs (1024); clearly, these parameters will be modified later based on the outcomes of the subsequent training sessions.

We proceed with training the model by initializing the respective variables based on the values just listed, which are reported, for convenience, in the following table.

Table 3: Initialized variable values in the first training phase
The listed parameters define the configuration adopted for the initial comparison between normalization techniques, providing a baseline for evaluating their relative effectiveness.

Parameter	Assigned Value
Interval	1 Hour
Observation Window	64
Prediction Window	2
Epochs	1024

We begin by using stocks as the instrument category and logarithmic difference normalization; for the first selected stock (Apple), we observe the plots representing the model's performance on the dataset used for training.

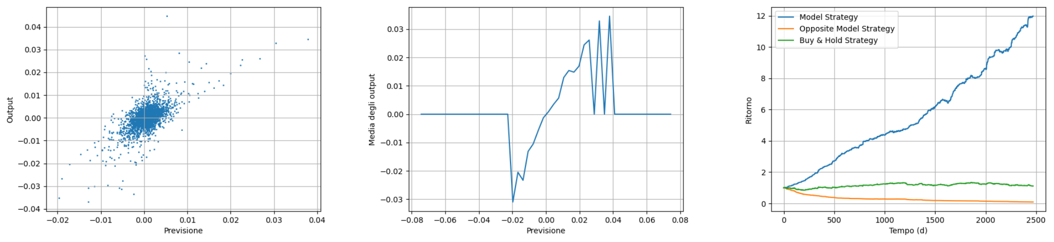

Figure 6: Plots of model performance on known dataset

The meaning of each individual plot is explained in section 2.5 Model Evaluation. As expected, since the model is tested on known data, the performance is excellent. From the first plot, we deduce quite clearly that as the predictions increase, the real outputs tend to increase. This is confirmed by the second plot, which highlights the average output as a function of the different value ranges that the predictions assume. Finally, the plot representing an operational simulation shows us that the cumulative return of the strategy derived from the model's indications (blue) is far greater than that of the benchmark "buy and hold" strategy (green) and the strategy opposite to the model's (orange), despite having operated only 1255 hours out of the 2472 total.

Let's observe how the performance changes when the model is presented with a dataset different from the one used for training.

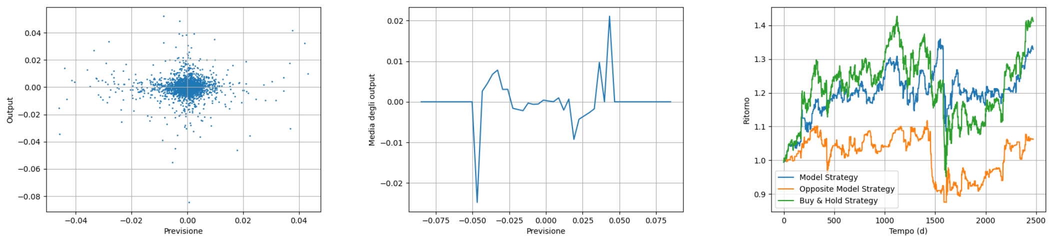

Figure 7: Plots of model performance on unknown dataset

As expected, the model's performance on an unknown dataset is lower than that observed previously. Firstly, the scatter plot proves to be uninformative regarding the model's real predictive capabilities, as the distribution of points does not allow for significant conclusions. The bin plot, however, shows a positive correlation: as the interval of predicted values increases, the average observed output also increases. However, this result, while encouraging, must be interpreted with caution, as most predictions assume values close to zero, and the extremes of the plot are statistically unrepresentative.

Finally, the backtesting plot provides a more concrete indication of the model's effectiveness: the strategy based on the predictions (blue) recorded an increase of 32.8%, operating on 1,434 of the 2,472 total hours in the test period. In the same time frame, the benchmark buy and hold strategy achieved an increase of 41.2%, while the strategy opposite to the model's recorded an increase of 6.3%.

The statistics emerging from the backtesting plot are, therefore, the most significant for performance evaluation, which is why they will be adopted as the main reference in subsequent analyses.

The model training is repeated multiple times to verify that the results obtained are not random, and the same procedure is applied to all instruments listed in Table 1. The collected data are reported in the following table.

The entire procedure is repeated using the min-max normalization technique. Again, the collected data are reported in the following table.

Observing the data reported in the two tables, some considerations can be drawn. Firstly, it is noted that the cumulative increment values of the different strategies during the operational simulation

Table 4: Backtest performance - logarithmic difference normalization
The table reports the cumulative returns (percentage increase obtained during the validation period) achieved by the model, the opposite strategy, and the "buy and hold" benchmark, highlighting the impact of logarithmic difference normalization. Each comparison is repeated three times to increase the statistical relevance of the conclusions drawn from the observed data.

Instrument	Model Strategy (%)	Opposite Strategy (%)	Benchmark Strategy (%)
Apple	9.3, 43.7, 23.4	-1.0, -24.7, -12.4	8.1, 8.1, 8.1
Intel	78.1, 11.7, 27.4	-26.9, 16.5, 2.1	30.1, 30.1, 30.1
Amd	15.1, 18.3, 13.9	-15.3, -17.6, -14.4	-2.4, -2.4, -2.4
Euro	1.5, -0.7, -3.1	3.5, 5.7, 8.1	5.0, 5.0, 5.0
Pound	0.0, 3.0, 0.9	0.8, -2.2, -0.2	0.8, 0.8, 0.8
Yen	5.6, 1.8, 2.4	-7.7, -4.3, -4.8	-2.6, -2.6, -2.6
Bitcoin	62.7, 28.3, 45.0	12.6, 42.7, 26.3	83.1, 83.1, 83.1
Ethereum	38.3, 110.6, 64.5	22.5, -19.6, 3.0	69.4, 69.4, 69.4
Solana	150.7, 108.3, 28.6	-34.6, -21.3, 27.4	63.9, 63.9, 63.9
Gold	39.8, 26.5, 27.6	-0.7, 9.7, 8.7	38.7, 38.7, 38.7
Silver	11.1, 0.67, 30.2	21.5, 26.6, 3.7	35.0, 35.0, 35.0
Oil	4.4, -8.2, -1.8	-13.8, -1.9, -8.3	-10.0, -10.0, -10.0

Table 5: Backtest performance - min-max normalization
The table reports the cumulative returns (percentage increase obtained during the validation period) achieved by the model, the opposite strategy, and the "buy and hold" benchmark, highlighting the impact of min-max normalization. Each comparison is repeated three times to increase the statistical relevance of the conclusions drawn from the observed data.

Instrument	Model Strategy (%)	Opposite Strategy (%)	Benchmark Strategy (%)
Apple	-6.0, 41.6, -12.3	15.1, -23.5, 23.4	8.2, 8.2, 8.2
Intel	4.4, 71.3, 54.1	26.4, -22.9, -14.4	32.0, 32.0, 32.0
Amd	-0.4, 23.6, 24.6	-1.6, -20.8, -21.4	-2.0, -2.0, -2.0
Euro	10.2, 9.4, -4.9	-4.7, -4.0, 10.4	5.0, 5.0, 5.0
Pound	3.0, 8.6, 8.3	-2.1, -7.3, -7.0	0.7, 0.7, 0.7
Yen	1.5, -0.3, -8.9	-4.0, -2.1, 7.0	-2.5, -2.5, -2.5
Bitcoin	34.3, 27.6, 77.5	35.4, 42.5, 2.5	81.9, 81.9, 81.9
Ethereum	32.1, 74.5, 12.6	27.0, -3.7, 49.2	68.0, 68.0, 68.0
Solana	-13.0, 6.1, 42.2	84.9, 51.6, 13.2	60.1, 60.1, 60.1
Gold	10.8, 33.3, 19.8	25.1, 3.9, 15.7	38.6, 38.6, 38.6
Silver	-2.1, 5.4, 27.0	37.9, 28.1, 6.3	35.0, 35.0, 35.0
Oil	1.4, -8.2, -28.4	-11.2, -2.0, 25.6	-10.0, -10.0, -10.0

phase show high variance; it is therefore not very significant to calculate their average. On the contrary, it is interesting to analyze how frequently the strategy derived from the trained model's predictions obtains a higher increment than that of the opposite strategy (success). In such a case, in fact, we can state that the model has allowed achieving an increment greater than half of that obtained by the benchmark strategy, while operating, approximately, half the time. In stating this, it is assumed that the average of the predictions made on the training dataset coincides, approximately, with the median of the predictions made on the validation dataset. This hypothesis is legitimate because, during the data collection phase, it was constantly observed that the number of predictions above the threshold is very similar to the number of predictions below it, indicating a sufficiently symmetric distribution of the model's predictions around their average value.

We proceed by defining $p = \frac{favorable\ cases}{total\ cases}$ (p represents the Success Rate of the models trained during the specific data collection session) for each table, comparing the values obtained as the different parameters vary. To ensure the statistical relevance of the results, the p values are supplemented with 95% confidence intervals, obtained through a normal approximation of the sample mean of independent events (corresponding to the outcomes of the different training sessions).

Continuing with the observation of the data reported in the two tables, it is noted that the strategy based on the predictions of the models trained using logarithmic difference normalization obtains, during the operational simulation phase, results superior to the opposite strategy in 27 out of 36 cases, corresponding to

$$p = 0.75 \pm 0.14.$$

In the case of min-max normalization, however, the strategy based on the predictions is better in 21 out of 36 cases, i.e.,

$$p = 0.58 \pm 0.16.$$

From these results, it emerges that logarithmic difference normalization is considerably more effective for the purposes of the predictions considered in the study. Therefore, this method will be adopted for training all models tested from this point forward.

Table 6: Success Rate for normalization techniques
The table shows the frequency with which each normalization method produced models with performance superior to the opposite strategy, providing quantitative evidence of the greater stability of the logarithmic method.

Normalization	Success Rate
log-difference	0.75 ± 0.14
min-max scale	0.58 ± 0.16

Wanting to continue with the analysis of the collected data, we can observe that by reducing the pool of instrument categories considered to only stocks and cryptocurrencies, the statistic just observed, regarding the logarithmic difference normalization method, rises to 16 out of 18 cases, i.e.,

$$p = 0.89 \pm 0.14$$

This might suggest that these instrument categories are more suited to the type of analysis we are performing. However, we leave this type of conclusion for later stages of the study, during which further data will allow us to confirm or deny them with greater certainty.

After this first phase of data collection, we can also conclude that the selected number of epochs, although not necessarily optimal, proves to be adequate for the type of training we are performing, as the performance obtained by the predictive models on unknown datasets is comparable to that obtained on known datasets.

3.2 Comparison of Sampling Intervals

We proceed to verify whether changing the data sampling interval among minutes, hours, and days significantly affects the models' performance. At the same time, we reduce the number of epochs to 512 to evaluate if this adjustment improves the results during the backtest phase, for example by reducing overfitting or speeding up the training phase.

Table 7: Backtest performance - 1 minute interval
The table reports the cumulative returns (percentage increase obtained during the validation period) achieved by the model, the opposite strategy, and the "buy and hold" benchmark, to highlight the accuracy of predictions against high-frequency data. Each comparison is repeated three times to increase the statistical relevance of the conclusions drawn from the observed data.

Instrument	Model Strategy (%)	Opposite Strategy (%)	Benchmark Strategy (%)
Apple	-0.14, -0.32, -0.36	0.33, 0.52, 0.56	0.19, 0.19, 0.19
Intel	-0.66, -3.90, -2.07	2.34, 5.77, 3.82	1.67, 1.67, 1.67
Amd	0.37, 1.39, 1.67	-1.34, -2.33, -2.60	-0.98, -0.98, -0.98
Euro	0.08, 1.19, 0.72	-0.06, -1.13, -0.68	0.01, 0.01, 0.01
Pound	-0.44, -0.89, -1.02	0.23, 0.68, 0.80	-0.21, -0.21, -0.21
Yen	0.27, -0.67, 0.78	-0.67, 0.27, -1.17	-0.40, -0.40, -0.40
Bitcoin	0.13, 0.17, 1.59	-2.27, -2.31, -3.68	-2.14, -2.14, -2.14
Ethereum	-2.37, -0.36, 2.58	-4.80, -6.72, -9.40	-7.06, -7.06, -7.06
Solana	9.38, 12.29, 10.63	-18.74, -20.84, -19.66	-11.11, -11.11, -11.11
Gold	0.03, 0.45, 0.43	0.47, 0.05, 0.07	0.50, 0.50, 0.50
Silver	0.20, 1.11, 0.91	0.36, -0.97, -0.74	0.15, 0.15, 0.15
Oil	1.60, 2.61, 1.85	2.08, 1.07, 1.82	3.70, 3.70, 3.70

Table 8: Backtest performance - 1 hour interval
The table reports the cumulative returns (percentage increase obtained during the validation period) achieved by the model, the opposite strategy, and the "buy and hold" benchmark, to highlight the accuracy of predictions against medium-frequency data. Each comparison is repeated three times to increase the statistical relevance of the conclusions drawn from the observed data.

Instrument	Model Strategy (%)	Opposite Strategy (%)	Benchmark Strategy (%)
Apple	19.9, 17.2, 9.8	-7.0, -4.9, 1.5	11.5, 11.5, 11.5
Intel	135.4, 101.1, 56.0	-42.2, -32.4, -12.8	36.0, 36.0, 36.0
Amd	-3.6, 0.7, 25.5	-1.6, -5.8, -24.5	-5.1, -5.1, -5.1
Euro	3.8, -2.5, 0.0	1.34, 7.9, 5.2	5.2, 5.2, 5.2
Pound	-2.2, -0.9, 2.3	2.8, 1.5, -1.8	0.5, 0.5, 0.5
Yen	-1.2, -0.2, 1.4	-2.6, -2.5, -5.0	-3.7, -3.7, -3.7
Bitcoin	45.7, 57.0, 37.2	19.1, 10.6, 26.7	73.7, 73.7, 73.7
Ethereum	64.3, 68.7, 72.4	-4.2, -6.8, -8.7	57.0, 57.0, 57.0
Solana	93.8, 36.3, 249.7	-29.7 , 0.01, -0.61	36.3, 36.3, 36.3
Gold	27.6, 29.5, 16.3	11.1, 9.5, 21.8	41.7, 41.7, 41.7
Silver	4.9, 30.3, -2.3	32.9, 7.0, 42.3	39.4, 39.4, 39.4
Oil	-9.8, -6.7, -16.2	6.1, 2.6, 14.3	-4.2, -4.2, -4.2

Table 9: Backtest performance - 1 Day interval
The table reports the cumulative returns (percentage increase obtained during the validation period) achieved by the model, the opposite strategy, and the "buy and hold" benchmark, to highlight the accuracy of predictions against low-frequency data. Each comparison is repeated three times to increase the statistical relevance of the conclusions drawn from the observed data.

Instrument	Model Strategy (%)	Opposite Strategy (%)	Benchmark Strategy (%)
Apple	520, 826, 155	70, 14, 314	958, 958, 958
Intel	121, 38, 191	-53, -24, -64	4.7, 4.7, 4.7
Amd	2061, 664, 693	166, 654, 626	5668, 5668, 5668
Euro	3.1, 8.7, 7.0	5.1, -0.3, 1.3	8.3, 8.3, 8.3
Pound	14.9, 3.3, 11.2	-21.5, -12.7, -18.9	-9.8, -9.8, -9.8
Yen	-10.8, -3.1, -14.8	-9.5, -16.8, -5.2	-19.3, -19.3, -19.3
Bitcoin	1520, 660, 929	-21, -69, -25	1182, 1182, 1182
Ethereum	-9.2, 62.8, 38.7	18.0, 34.2, 22.8	7.1, 7.1, 7.1
Solana	309, 583, 636	161, 56, 45	969, 969, 969
Gold	98.0, 63.1, 85.6	75.7, 113.4, 87.6	248.0, 248.0, 248.0
Silver	41.3, 180.0, 47.6	116.7,9.5, 107.5	206.2, 206.2, 206.2
Oil	57.9, 36.5, 42.3	9.7, 30.0, 21.8	73.3, 73.3, 73.3

Observing the data reported in the three tables, we can draw some conclusions. Firstly, it is noted that, by sampling the data minute by minute, the models achieve results better than the opposite strategy in 23 out of 36 cases, corresponding to $p = 0.64 \pm 0.16$, sampling the data hour by hour in 25 out of 36 cases, i.e., $p = 0.69 \pm 0.15$, and sampling the data day by day in 28 out of 36 cases, corresponding to $p = 0.78 \pm 0.14$. From these results, it emerges that increasing the data sampling interval allows for obtaining models with superior predictive capabilities. At the same time, it is necessary to consider that reducing this parameter allows for operating with greater frequency and, consequently, potentially achieving better performance despite the lower precision.

Table 10: Success Rate as a function of sampling interval
The table compares the predictive success frequencies for the different sampling intervals, highlighting the improvement obtained with wider time windows.

Sampling Interval	Success Rate
1 minute	0.64 ± 0.16
1 hour	0.69 ± 0.15
1 day	0.78 ± 0.14

Analyzing the data independently of the sampling interval, but considering the breakdown by instrument category, it is noted that, for stocks, the models obtain results better than the opposite strategy in 20 out of 27 cases, $p = 0.74 \pm 0.17$, for currencies in 16 out of 27 cases, $p = 0.59 \pm 0.19$, for cryptocurrencies in 26 out of 27 cases, $p = 0.96 \pm 0.07$, and for commodities in 14 out of 27 cases, $p = 0.52 \pm 0.19$.

Table 11: Success Rate as a function of instrument category
The table compares the predictive success frequencies for the different instrument categories considered, highlighting the greater predictability of cryptocurrencies and stocks.

Instrument	Success Rate
Stocks	0.74 ± 0.17
Currencies	0.59 ± 0.19
Cryptocurrencies	0.96 ± 0.07
Commodities	0.52 ± 0.19

This data supports the hypothesis formulated earlier, according to which stocks and cryptocurrencies are more suited to the type of analysis conducted and are more predictable.

3.3 Comparison of Observation Windows

We proceed to verify if increasing and decreasing the observation window parameter leads to significant variations in the performance of the trained models.

Table 12: Backtest performance - 32-hour observation window
The table reports the cumulative returns (percentage increase obtained during the validation period) achieved by the model, the opposite strategy, and the "buy and hold" benchmark, to highlight the accuracy of predictions for short time windows. Each comparison is repeated three times to increase the statistical relevance of the conclusions drawn from the observed data.

Instrument	Model Strategy (%)	Opposite Strategy (%)	Benchmark Strategy (%)
Apple	16.0, 2.8, 0.0	-2.6, 9.9, 12.9	12.9, 12.9, 12.9
Intel	48.3, 108.5, 71.0	7.9, -23.3, -6.4	60.0, 60.0, 60.0
Amd	4.1, 14.8, 25.2	-5.9, -14.6, -21.7	-2.0, -2.0, -2.0
Euro	2.9, 2.3, 8.1	2.9, 4.6, -1.1	6.9, 6.9, 6.9
Pound	2.6,5.4, -0.3	0.3, -2.4, 3.3	2.9, 2.9, 2.9
Yen	5.4, 2.8, 3.5	-4.9, -2.5, -3.1	0.2, 0.2, 0.2
Bitcoin	78.3, 71.2, 71.3	10.3, 14.9, 14.9	96.8, 96.8, 96.8
Ethereum	66.7, 167.3, 117.2	11.7, -30.4, -14.3	86.1, 86.1, 86.1
Solana	150.0, 120.4, 93.8	-36.8, -29.2, -18.4	58.1, 58.1, 58.1
Gold	23.6, 22.2, 28.9	18.8, 20.0, 12.8	46.7, 46.7, 46.7
Silver	48.6, 55.9, 51.8	2.6, -2.3, 0.3	52.4, 52.4, 52.4
Oil	-0.4, -19.7, -20.1	-21.6, -2.1, -1.5	-21.3, -21.3, -21.3

Table 13: Backtest performance - 64-hour observation window
The table reports the cumulative returns (percentage increase obtained during the validation period) achieved by the model, the opposite strategy, and the "buy and hold" benchmark, to highlight the accuracy of predictions for medium-length time windows. Each comparison is repeated three times to increase the statistical relevance of the conclusions drawn from the observed data.

Instrument	Model Strategy (%)	Opposite Strategy (%)	Benchmark Strategy (%)
Apple	8.1, 27.9, 18.6	3.1, -12.8, -5.9	11.6, 11.6, 11.6
Intel	56.3, 93.4, 26.2	6.9, -13.7, 32.1	70.0, 70.0, 70.0
Amd	44.4, 8.8, -12.3	-26.9, -3.0, 20.3	5.5, 5.5, 5.5
Euro	8.1, 7.7, 5.5	-1.0, -0.6, 1.5	7.0, 7.0, 7.0
Pound	-1.7, 2.2, 6.8	4.8, 0.7, -3.6	3.0, 3.0, 3.0
Yen	4.3, -1.3, 7.0	-3.3, 2.2, -5.8	0.8, 0.8, 0.8
Bitcoin	75.4, 84.2, 86.0	12.4, 7.0, 6.0	97.2, 97.2, 97.2
Ethereum	73.2, 42.7, 155.1	7.9, 31.0, -26.7	87.0, 87.0, 87.0
Solana	175.1, 227.1, 126.4	-41.1, -50.5, -28.5	61.8, 61.8, 61.8
Gold	13.6, 28.7, 15.8	31.0, 15.4, 28.6	48.8, 48.8, 48.8
Silver	27.7, 52.0, 12.6	22.6, 3.0, 38.9	56.5, 56.5, 56.5
Oil	4.6, -13.6, -14.5	-21.4, -4.7, -3.7	17.7, 17.7, 17.7

Table 14: Backtest performance - 128-hour observation window
The table reports the cumulative returns (percentage increase obtained during the validation period) achieved by the model, the opposite strategy, and the ”buy and hold” benchmark, to highlight the accuracy of predictions for long time windows. Each comparison is repeated three times to increase the statistical relevance of the conclusions drawn from the observed data.

Instrument	Model Strategy (%)	Opposite Strategy (%)	Benchmark Strategy (%)
Apple	17.9, 13.9, 12.5	-9.9, -6.7, -5.6	6.2, 6.2, 6.2
Intel	-9.8, 10.2, 5.2	7.8, -11.7, -7.6	-2.7, -2.7, -2.7
Amd	16.8, 23.1, 32.1	-5.3, -10.2, -16.4	10.5, 10.5, 10.5
Euro	7.4, 7.9, 7.6	1.8, 1.4, 1.7	9.4, 9.4, 9.4
Pound	5.3, 3.8, 6.1	2.0, 3.6, 1.2	7.5, 7.5, 7.5
Yen	4.5, 2.4, 9.5	0.0, 2.0, -4.6	4.5, 4.5, 4.5
Bitcoin	56.7, 38.9, 82.2	9.1, 23.1, -6.1	71.0, 71.0, 71.0
Ethereum	62.7, 79.3, 64.9	-4.3, -13.1, -5.5	55.7, 55.7, 55.7
Solana	70.4, 45.1, 122.1	-18.0, -3.6, -37.0	39.8, 39.8, 39.8
Gold	34.3, 51.3, 13.4	16.7, 3.8, 38.5	57.0, 57.0, 57.0
Silver	46.6, 34.2, 9.4	3.0, 12.6, 38.1	51.1, 51.1, 51.1
Oil	-7.1, -16.1, -17.3	-14.0, -4.8, -3.4	-20.1, -20.1, -20.1

Observing the data reported in the three tables, we can draw some conclusions. Firstly, it is noted that, by initializing the observation window parameter to 32, the models achieve results better than the opposite strategy in 29 out of 36 cases, $p = 0.81 \pm 0.13$, initializing it to 64 in 28 out of 36 cases, $p = 0.78 \pm 0.14$, and initializing it to 128 in 31 out of 36 cases, $p = 0.86 \pm 0.11$. From these results, it emerges that increasing or decreasing the observation window parameter has little effect on the trained model's performance, suggesting that most of the information useful for prediction is contained in the concluding part of the input sequence.

Table 15: Success Rate as a function of observation window
The table compares the predictive success frequencies as a function of the observation window length, highlighting a weak correlation between the two values.

Observation Window	Success Rate
32 hours	0.81 ± 0.13
64 hours	0.78 ± 0.14
128 hours	0.86 ± 0.11

Analyzing the data independently of the observation window parameter, but considering the breakdown by instrument class, it is noted that, for stocks, the models obtain results better than the opposite strategy in 23 out of 27 cases, $p = 0.85 \pm 0.13$, for currencies in 22 out of 27 cases, $p = 0.81 \pm 0.15$, for cryptocurrencies in 27 out of 27 cases, $p = 1.00 \pm 0.00$, and for commodities in 16 out of 27 cases, $p = 0.59 \pm 0.19$.

Table 16: Success Rate as a function of instrument category
The table compares the predictive success frequencies for the different instrument categories considered, highlighting the greater predictability of cryptocurrencies and stocks.

Instrument	Success Rate
Stocks	0.85 ± 0.13
Currencies	0.81 ± 0.15
Cryptocurrencies	1.00 ± 0.00
Commodities	0.59 ± 0.19

Again, the data supports the hypothesis formulated earlier: stocks and cryptocurrencies seem to have a more predictable trend than the other instrument categories analyzed.

4 Conclusions

The objective of this study was to evaluate the effectiveness of machine learning algorithms, particularly feedforward neural networks, in processing historical financial data for short-term price change prediction. The analysis highlighted how the quality of data preparation is a determining factor for the predictive performance of the models. Among the normalization techniques examined, the one based on the logarithmic difference allowed for the production of models characterized by greater stability and better generalization ability compared to min-max normalization, thus proving to be the most suitable for the type of prediction being made.

Regarding the sampling frequency, it emerged that wider time intervals (daily data) produce more robust models with better predictive capabilities compared to those trained on high-frequency data (hourly or by the minute). This suggests that the considered neural networks are more effective at capturing medium-term patterns rather than short-term micro-variations, which are often dominated by noise.

Variations in the observation window parameter did not produce significant effects on performance, suggesting that most of the relevant information for prediction is contained in the last values of the input sequence.

Overall, the results show that machine learning applied to financial data series can provide consistent and potentially useful operational indications, especially in the case of assets characterized by high volatility such as cryptocurrencies and stocks. However, it is important to emphasize that the observed performance is relative to an ideal context and does not take into account operational costs, data retrieval delays, and all the real market conditions that make short-term price change prediction a challenge even for the best analysts.

In future prospects, the integration of more advanced models, such as recurrent networks or Transformer-LSTM hybrids, and the expansion of datasets with macroeconomic or technical indicators could allow for further improvement in the predictive ability and robustness of the strategies derived from the learning models.

—The entire data collection phase carried out in this study can be replicated using the Python code available in the dedicated repository[4].—

[4]`https://github.com/CarloCetrone/Predicting-Financial-Time-Series-Using-Python`