In a Hybrid ARIMA-LSTM Modeling For a Random Walk Process, a Good Linear Prediction Is All You Need (Capstone)

Emmanuel Ameyaw[1,2]

Contributing authors: emml.ameyaw@gmail.com;

Abstract

Cryptocurrencies are dynamic financial instruments characterized by high volatility and high non-stationarity with many intricate patterns for different time periods. This makes prediction a challenging task. In recent years, machine learning approaches, including Support Vector Machines, Random Forests, and Artificial Neural Networks, and others have shown promise in predicting cryptocurrency prices. In this paper, we propose an ARIMA-LSTM framework for better price prediction and trading strategy development given an asset price that exhibits a random walk process. Thus, combining the strengths of both the ARIMA model and the LSTM model. We also consider two different filters—the moving average filter and the HP filter—in the disaggregating the data into low and high volatility components. We found that using the HP filter significantly performs better than using a moving average filter. The performance of the model, however, is not due to the low volatility series (modeled using LSTM) but the high volatility series (modeled using the ARIMA). Thus, the contribution of the non-linearities in the data and the subsequent modeling of these non-linearities to the performance of the model is very limited. Precisely, in a hybrid ARIMA-LSTM modeling for a random walk process, we found that a good linear prediction is all you need.

Keywords: Cryptocurrency , Hybrid ARIMA-LSTM, Price Prediction

1 Introduction

Cryptocurrencies are currently expanding as a new financial instrument, and their popularity has lately surged, as a result of their enormous growth and incredibly high market capitalizations. Despite its early stage compared to other financial assets, the

cryptocurrency exchange market has become a worldwide phenomenon, particularly noted for its volatility and diversity, capturing the interest of many new and experienced investors (Hileman & Rauchs, 2017). The advent and rise of cryptocurrencies have transformed the landscape of global financial markets with cryptocurrencies highly influenced by market sentiments and technological advancements.

Forecasting financial time series is a highly challenging task since these series are characterized by non-stationarity, heteroscedasticity, discontinuities, outliers, and high-frequency multi-polynomial components, making market movement prediction highly complex (Hadavandi et al., 2010). The intricate properties of financial time series, especially cryptocurrencies, as well as the massive amounts of data that must be analyzed in order to correctly forecast financial time series, have encouraged the adoption of more sophisticated approaches, models, and simulation techniques (Borges & Neves, 2020). Recently, machine learning or data mining techniques, which are frequently used in financial market forecasting, have shown better outcomes than simple technical or fundamental research methodologies. Machine learning approaches are capable of identifying patterns and forecasting future trends in order to discover the optimal entry and exit points in a financial time series in order to get the maximum returns with the lowest risk (Nadkarni & Ferreira Neves, 2018). Machine Learning techniques such as Support Vector Machine (Peng et al., 2018), Random Forest (Sun Yin et al., 2019) and Artificial Neural Networks (Kristjanpoller & Minutolo, 2018) have also been used to forecast Bitcoin prices due to their ability to capture complicated nonlinear patterns or relationships among many variables. Due to their great representational capacity, Deep Learning methods have demonstrated good performance for financial time-series forecasting (Park & Yang, 2022). Specifically, LSTM by using recurrent and gate mechanisms have outperformed other machine learning models due to its ability to capture long range dependencies(Patel et al., 2020; Zhang et al., 2021).

1.1 Problem Statement

The Development of trading and investment strategy for cryptocurrency such as Litecoin, just like any financial asset is highly reliant on accurate prediction of the price movement. To this end, several methods have been developed by researchers to address this challenge. In this paper, we aim to develop a trading strategy based on price prediction using ARIMA-LSTM framework that leverages on the capturing of long-term trends using an ARIMA model and capturing non-linear short-term dependencies and patterns using an LSTM model. We combine the strengths of the ARIMA model and the LSTM model to generate trading signals of Buy/Sell/Hold—using the combined forecast from these two models.

1.2 Related literature

Several models have been employed for cryptocurrency price prediction over the years. These can be classified into statistical and machine learning and Deep Learning models. Examples are the Autoregressive and integrated moving averages (ARIMA)

employed by (Wirawan et al., 2019) to predict bitcoin price. Peng et al. (2018) used Generalized Autoregressive Conditional Heteroskedasticity (GARCH) in conjunction with support vector machine (SVM) for the prediction of prices of Ethereum, Bitcoin and Dash and recorded improved accuracy. Hitam et al. (2019) utilized SVM in conjunction with particle Swarm optimization (PSO) for the prediction of prices of Bitcoin, Ethereum, Litecoin, Ripple, Nem and Stellar. Rathan et al. (2019) employed Decision Tree and Linear Regression to predict Bitcoins' price from historical price data. Wu et al. (2018) employed Long Short-Term Memory (LSTM) to forecast the prices of Bitcoin. In forecasting the prices of Litecoin and Monero, Patel et al. (2020) utilized a hybrid LSTM-GRU which produced a high accuracy compared to classical methods. Serrano (2022) utilized deep learning in conjunction with reinforcement learning to predict upward and downward movement of prices of Bitcoin. Also, Borges & Neves (2020) employed Ensemble of different machine learning models to generate technical indicators using trading indicators as inputs. Ortu et al. (2022) combined trading indicator and social media indicator to predict the direction of the market for cryptocurrency. Park & Yang, (2023) developed a trading strategy by employing Adaboost-LSTM for the prediction of Bitcoin's price and also employed Markov Regime-Switching to capture structural volatility in the market.

As can be seen from the various studies mentioned above, hybrid models are indeed increasingly becoming a commonplace in asset price prediction. However, in many cases, there is not a proper assessment of the source of the good performance of the hybrid model. For example, in a hybrid ARIMA-LSTM model, which model (ARIMA or LSTM) dominates the other in terms of the overall performance of the model? Put differently, what is the contribution of the LSTM part of the hybrid ARIMA-LSTM model, for example. And what is the contribution of the ARIMA part? LSTM models, indeed, are generally known to be better at capturing non-linearities in the data, but do they significantly contribute to the overall performance of the model? Besides, in terms of filtering the data into the various components to be used in the hybrid model, which filtering algorithm should one use? In this paper, we specifically compare the effectiveness of the moving average filtering algorithm to the HP filtering algorithm. We found that using an HP filter to disaggregate the data into low volatility (cycle) and high volatility (trend) components performs better than using a moving average filter.

Within a class of HP filters with different values for the smoothing parameter (λ), we observed that the distribution of the low volatility series from the model with $\lambda = 100$ exhibits a sharp and higher peak around the mean than the other two models with $\lambda = 1000$ and $\lambda = 10000$. The contribution of this distributional property of the low volatility component of the data in the overall performance of the model, however, seems to be less significant. Overall, we found that the ARIMA part of the hybrid model (which uses the high volatility (or trend) component of the disaggregated or filtered data as input) significantly outweigh the LSTM part of the hybrid model (which uses the low volatility (or cycle) component of the disaggregated data as input). In other words, in the context of our data which exhibits a random walk, it

seems a good linear prediction using an ARIMA model is all you need. The contribution of the LSTM part to the overall performance of the model is very limited.

The remainder of this paper is as follows: Section 2 provides a theoretical framework of our hybrid ARIMA-LSTM model. Section 3 presents the methodology and how the analysis was carried our. In Section 4, we discuss our results, and Section 5 concludes the paper.

2 Theoretical Framework/model

Cryptocurrency data are typically time series data, hence the use of traditional time series models of ARIMA and LSTM that captures the linear and nonlinear features in a dataset.

2.1 Autoregressive Integrated Moving Average model (ARIMA)

The ARIMA model is one of the most popular traditional models for time series forecasting that relies on Box-Jekins (Bartholomew, 1971) methodology of model identification, estimation and diagnostic. The model can be written in compact form as

$$\left(1-\sum_{i=1}^{p}\varphi_i L^i\right)(1-L)^d X_t = \left(1+\sum_{i=1}^{q}\theta_i L^i\right)\varepsilon_t \tag{1}$$

where φ_i represents the parameters of the autoregressive part of the model and θ_i represents the parameters of the moving averages part. L represent a lag operator and X_t represent the value of the time series data at time t. Lastly, ε_t represent the error term. Thus, the ARIMA (p, d, q) model is a combination of an AR and an MA model, where p denotes the order of the autoregressive part, d denotes the order of integration of the data, and q denotes the order of moving average part of the model. In finance, $d = 1$ is the most common and simplest case, as the raw price data is typically not stationary. By taking the first difference, however, many price data become stationary, at least mean stationary. In the literature, it is sometime argued whether one should model the first difference of X_t using ARIMA $(p, 0, q)$ = ARMA (p, q) or one should instead model X_t using ARIMA $(p, 1, q)$. As mentioned in Chan (2017), the later is advantageous, more flexible and gives better results.

2.2 The Long Short Term Memory (LSTM)

Long short-term memory (LSTM) is an efficient and scalable variant of recurrent neural networks (RNN) for learning sequential data. Because of its capacity to manage optimization challenges such as exploding and vanishing gradients, which often hamper vanilla RNNs, LSTM networks have been utilized in time series prediction due to its ability to capture nonlinear trends in data. The core LSTM architecture is made up of a memory cell that preserves its state throughout time and nonlinear gating units that control the flow of information into and out of the cell.An LSTM block's design

contains three gating mechanisms (input, forget, and output), a constant error carousel (a single cell), a block input (candidate state), and an output activation function. The block's output is transmitted back to all gates and the candidate state on a sequential basis. The structure of LSTM is shown below (Fan et al., 2021):

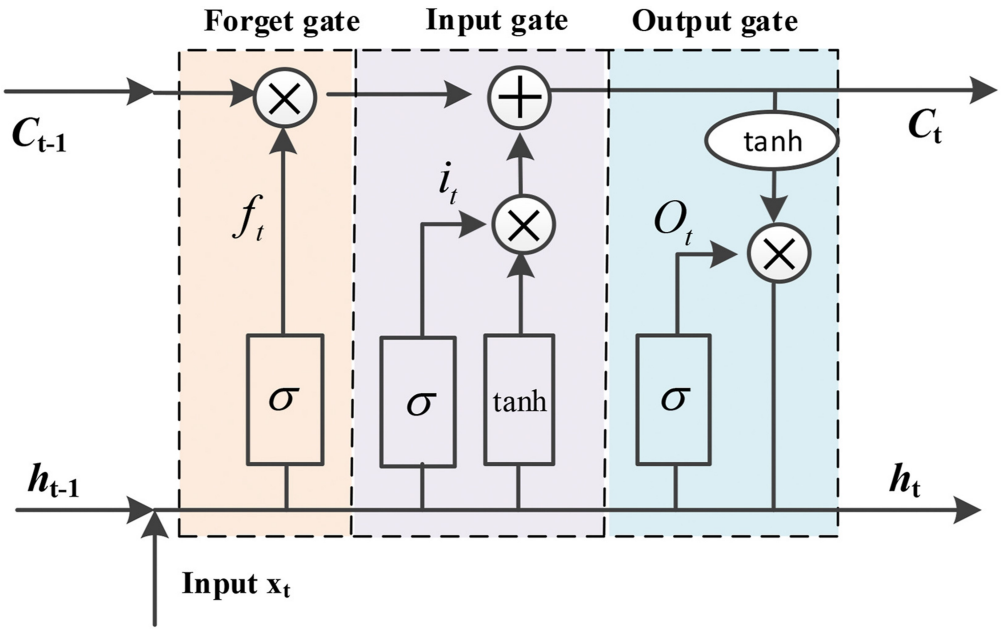

Fig. 1 The structure of LSTM. Source: Fan, et al. (2021).

The mathematical equations that represents the LSTM block's forward pass is as follows:

$$f_t = g\left(W_f \times X_f + U_f \times h_{t-1} + b_f\right) \tag{2}$$

$$\widehat{it} = \sigma\left(W_i \times X_t + U_i \times ht - 1 + b_i\right) \tag{3}$$

$$C_t = \sigma\left(W_c \times X_t + U_c \times h_{t-1} + b_c\right) \tag{4}$$

$$c_t = C_t \cdot \widehat{it} + ct - 1 \cdot f_t \tag{5}$$

$$O_t = \left(W_o \times X_t + U_o \times h_{t-1} + b_o\right) \tag{6}$$

$$h_t = O_t \cdot \tan h\left(C_t\right) \tag{7}$$

At each time step t, the LSTM block performs a series of computations to update its cell state and output. These computations involve a set of input weights

(W_f, W_i, W_c, W_o) that are associated with the forget gate, input gate, cell gate, and output gate, respectively. Additionally, recurrent weights (U_f, U_i, U_c, U_o) are introduced to account for the memory component by considering the impact of the previous time step's hidden state (h_{t-1}). To account for any biases introduced during these computations, bias weights (b_f, b_i, b_c, b_o) are considered for the forget gate, input gate, cell gate, and output gate, respectively. These biases contribute to the overall behavior of the LSTM block by modulating its responses. The input vector at time t, denoted as X_t, is an integral part of the computations. This vector contributes to determining the LSTM block's behavior at the current time step. Within this framework, various activation functions come into play. Firstly, an activation function g (commonly a sigmoid function) is employed for the forget gate to manage the degree of information to be retained from the previous time step. Moreover, the sigmoid activation function (σ) finds utility in computing the input modulation gate $(\widehat{i}_t)$ that contributes to regulating the information to be added to the cell state (c_t).

The hyperbolic tangent activation function (tanh) plays a crucial role in computing both the cell gate (C_t), which encapsulates the updated candidate cell state, and the cell modulation gate $(\widetilde{C}_t)$. The new cell state (c_t) is determined through the integration of the computed cell gate (C_t) and the input modulation gate $(\widehat{i}_t)$, while also accounting for the cell state from the previous time step (c_{t-1}) and the cell modulation gate $(\widetilde{C}_t)$. Furthermore, the output gate (O_t) is established through the use of the sigmoid activation function (σ). It governs the extent to which the cell state (c_t) is reflected as the output of the LSTM block. Ultimately, the output of the LSTM block at time t (h_t) is ascertained by multiplying the output gate (O_t) with the hyperbolic tangent of the cell state (c_t). These computations characterize the intricate interplay of the various elements within the LSTM model.

2.3 A hybrid ARIMA-LSTM model)

In our proposed ARIMA-LSTM hybrid model, we first use a filter to disaggregate our data into high-volatility (or trend) and low-volatility (or cycle) components. We then use an ARIMA model to model and forecast the high-volatility (or trend) component of our data, while an LSTM model is employed to model and forecast the low-volatility (or cycle) component of the data. We then add the two forecasts to make a final prediction. It should be noted that the hybrid model used in this paper is just one example of how to combine ARIMA and LSTM models to predict time series data. And indeed, there are others as the hybrid model certainly depends on the input data used in each of the two models—i.e., the ARIMA and LSTM models. One could. for example, decompose the data into trend, seasonal, and error components, and then use the LSTM and/or ARIMA model to model and forecast the different components of the data, and combine the forecast of the various components into one big forecast.

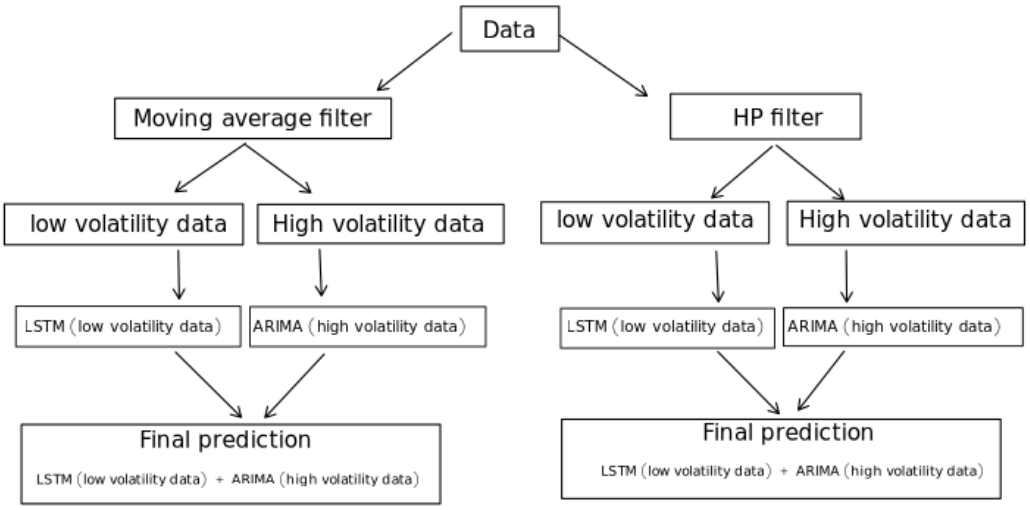

Fig. 2 Schematic Diagram of the proposed model

2.4 Evaluation Metrics

2.4.1 Performance Evaluation Measures for Regression

The performance of our model is mainly assessed using the cumulative returns (cumulative returns $= \prod_{t=1}^{n}(1 + \mathrm{r}_t) - 1$) on the test set. Precisely, we first calculate the difference between the predicted price at time $t+1$ and the actual price at time t, i.e., Δp. If $\Delta p > 0$, we buy the asset and take a long position. If $\Delta p < 0$, we sell the asset and take a short position. If $\Delta p = 0$, we take no action and hold our position. We compute the returns at each step and finally compute our cumulative returns over the test data. The model that yields the highest cumulative returns is then chosen as the best model.

3 Methodology

3.1 Data

We use Litecoin (LTC) historical data in this project, specifically 'LTC-USD'. The data is downloaded from yahoo finance using the yfinance package in Python, spanning about 5 years. A visual inspection of the data seem to suggest that it is a random walk.

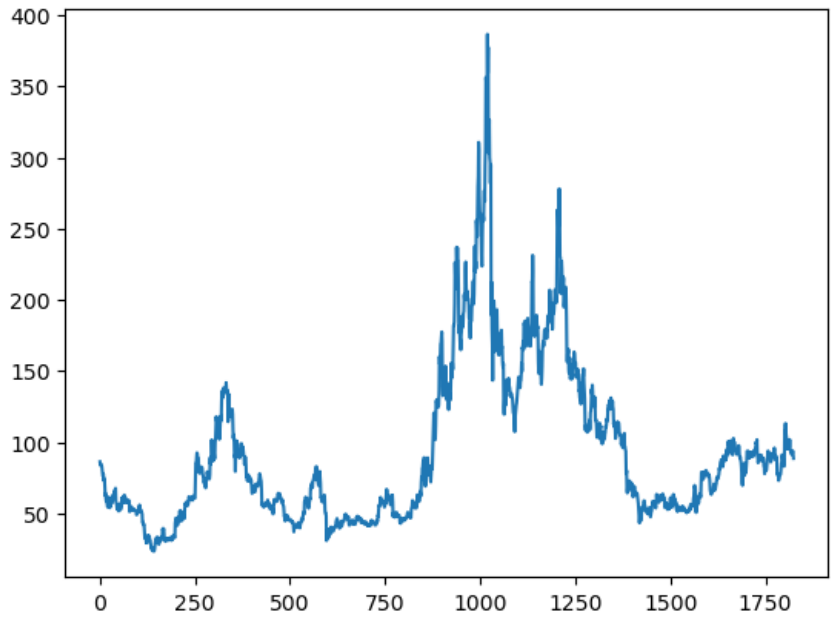

Fig. 3 Data

In the literature, a typical test for assessing whether a given series follows a random walk is the Augmented Dickey-Fuller (ADF) test, which checks for the presence of a unit root in a given time series. The null hypothesis in the ADF test posits that the time series under scrutiny does indeed possess a unit root (or in other words, follows a random walk). And by statistically computing the p-value of the test, we can reject or fail to reject this null hypothesis. Conventionally, in significance testing, the choice of a significance level, $\alpha = 0.05$, is typically employed as a threshold. A computed p-value below this chosen threshold offers substantial grounds to reject the null hypothesis, thereby prompting the conclusion that the time series does not conform to a random walk pattern. For our data, our computed p-value (0.2634) is greater than 0.05, indicating that we do not have enough evidence to reject the null hypothesis that our time series has a unit root. Thus, our time series follows a random walk. Theoretically, a random walk process is not variance-stationary as shown below, where we plot the residual of the AR(1) process of our price data.

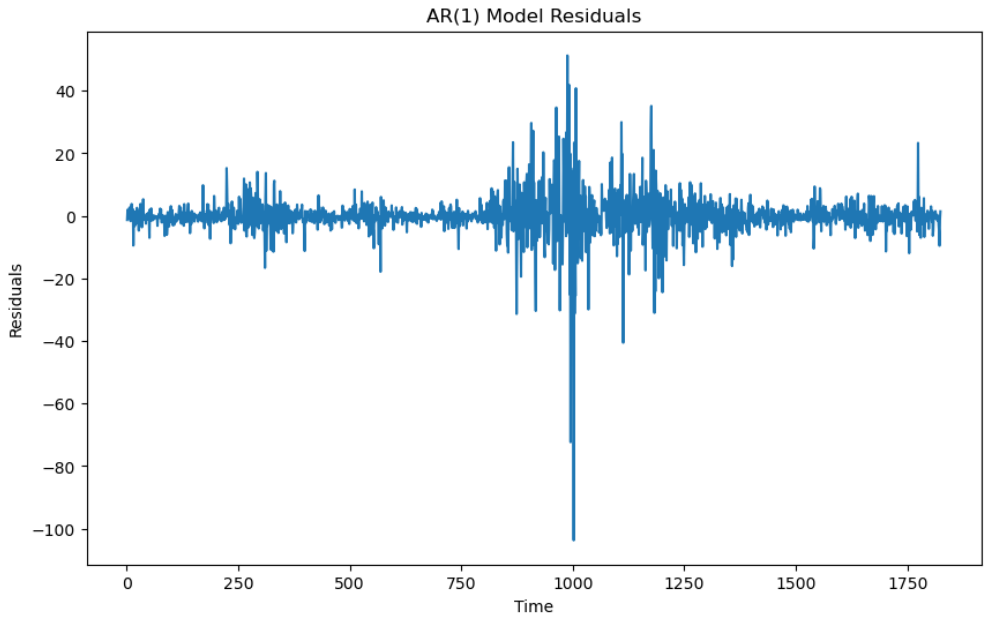

Fig. 4 AR(1) residuals of raw price data

Figure 4 above suggest our data, in its raw form, is not variance stationary. To stabilize the variance in our data, we apply log-transform the data. As shown below, where we plot the residual of the AR(1) process of the logged transformed price data, we can see that the variance of the logged transformed price data is relatively more stable than the variance of the original data.

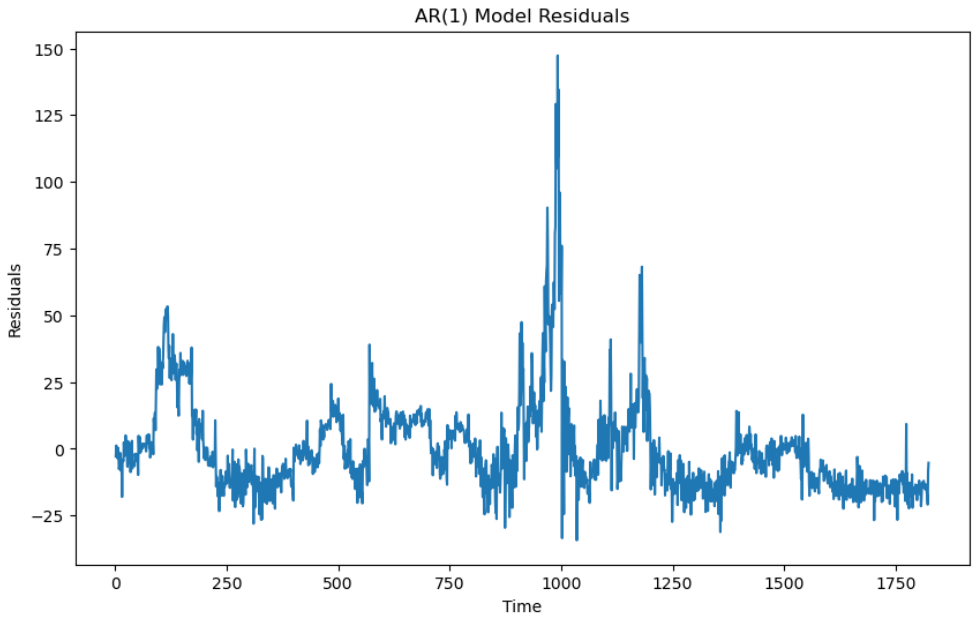

Fig. 5 AR(1) residuals of logged transformed price data

Next, before estimating our model, we divide our transformed data into training data (about 70% of the dataset) and the test data (about 30% of the dataset). We use the training data to train the model and the test data to evaluate its performance

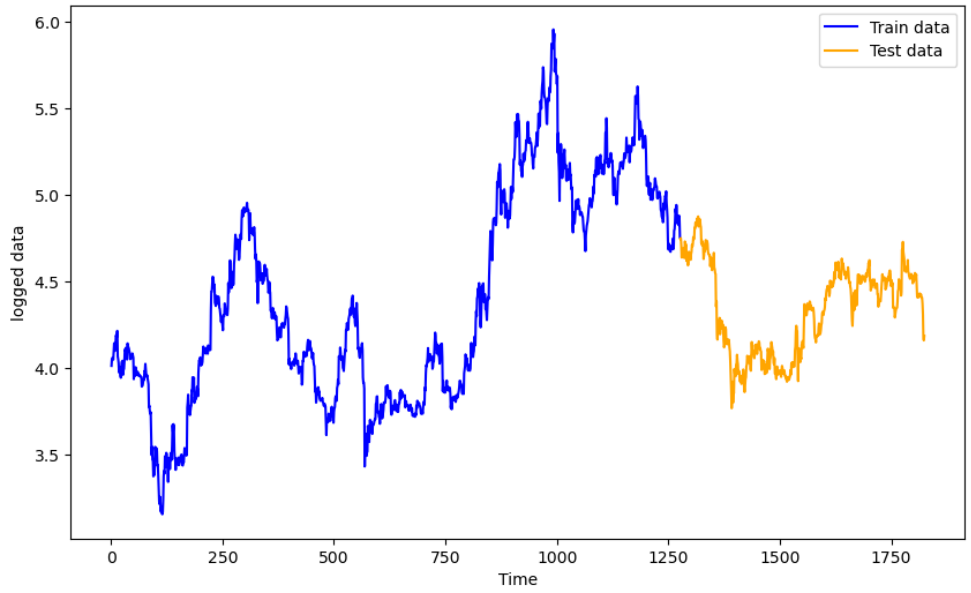

Fig. 6 Train and test data

3.2 Hybrid ARIMA-LSTM

In our hybrid ARIMA-LSTM model, we use an ARIMA model to model and forecast the high-volatility (or trend) component of our data, while an LSTM model is employed to model and predict the low-volatility (or cycle) component of the data. Hence, we commence our analysis by decomposing or filtering our data into two distinct components: high-volatility (or trend) and low-volatility (or cycle) components. We utilize two different filtering techniques. In our first model (Model 1), we filter our data using a moving average filtering algorithm, while in our second model (Model 2), we filter our data using the HP filter algorithm. We subsequently assess the comparative performance of these two filtering algorithms to determine their effectiveness. While the authors in Shui-Ling & Li (2017) employ a moving average filtering algorithm, we investigate whether this method outperforms alternative filtering algorithms, specifically, the HP filter algorithm.

4 Results and Discussion

4.0.1 Model 1: Using a moving average filter

In our first hybrid ARIMA-LSTM model, the decomposition of our data is done using a moving average filter. In other words, we use a moving average filter to separate our logged-transformed data into trend (high volatility) and cycle (low volatility) components. We first calculate a simple moving average (SMA) using a window size of 188. Following Shui-Ling & Li (2017), the optimized moving average period (188)

was obtained such that the resulting output of our moving average algorithm follows a normal distribution with a kurtosis of about 3, allowing for a 5% tolerance. We call our computed moving average series a high volatility series (or a trend). To compute the low volatility series (which we also refer to as cycles), we subtract the high volatility series from the original logged-transformed data. The two series are shown below in figure 7.

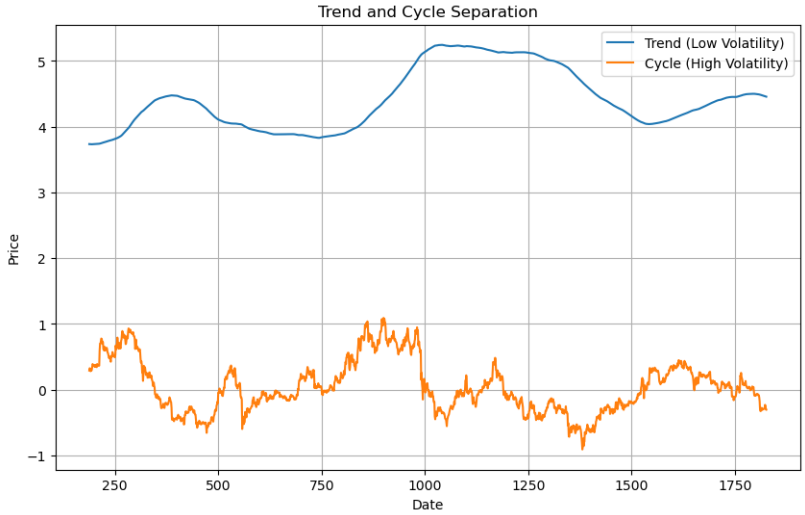

Fig. 7 Trend (high volatility) and cycle (low volatility) separation

Next, an Autoregressive Integrated Moving Average (ARIMA) model is employed to forecast the high volatility series, while a Long Short-Term Memory (LSTM) model is used to forecast the low volatility series or component of the data. Finally, these two individual forecasts or predictions are combined or summed to yield a unified forecast—i.e., Combined Forecast = ARIMA Forecast (using High Volatility series) + LSTM Forecast (using Low Volatility series).

In our ARIMA modeling, we make use of the pmdarima Python package to identify the most suitable ARIMA model, utilizing the Akaike Information Criterion (AIC). The goal here is to select the model with the lowest AIC, indicating a good balance between model fit and complexity. Our best ARIMA model is found to be ARIMA(1,2,1), which we then use to compute our predicted series. The predicted series is computed via the following process. First, we initialize an empty list called, prediction, to store the predicted values. Then, a loop iterates through the elements of the test data, which contains our high-volatility time series for the test period. Within each iteration, a new ARIMA model is created and fitted to the updated historical train data. Thus, mimicking a real-time forecasting scenario. The ARIMA model produces a one-step-ahead forecast or prediction for the next value in the time series. This forecast is appended to the prediction list, and the actual value from

the test set is added to the training data. This cycle continues until predictions are generated for all elements in the test data. This process allows for sequential forecasting, reflecting how predictions would be made in a real-world application. Lastly, we compute evaluation metrics, including Mean Squared Error (MSE) and Root Mean Squared Error (RMSE), to assess the quality of the generated predictions. The predictions and the actual test data are shown below. It is worth noting here that an AR(p) model yields a similar result as the ARIMA model employed here.

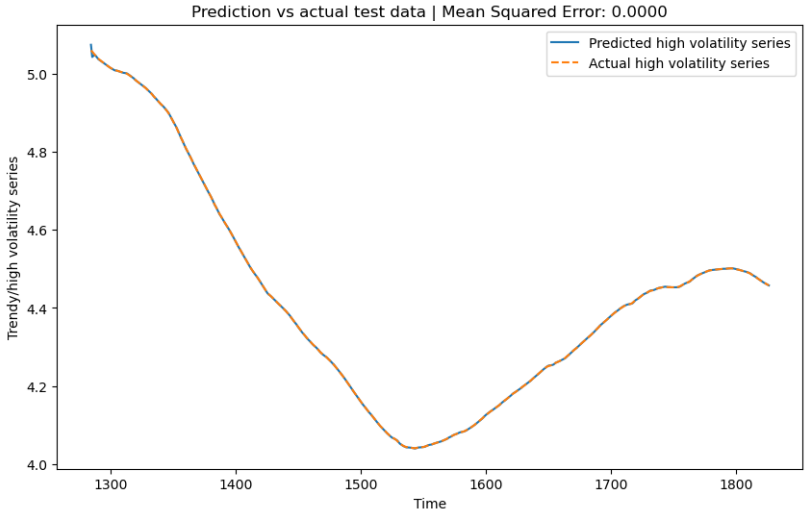

Fig. 8 Prediction vs actual test data (High volatility series)

Next, we utilize an LSTM model to forecast the low volatility or cycle component of our data. We search for the optimal combination of hyperparameters that minimizes the mean squared error (MSE) on the test dataset. The hyperparameters we consider are the number of LSTM units in our model, the activation function (either 'relu' or 'tanh'), and the learning rate used in the model. Our LSTM model is a sequential model with an LSTM layer followed by a Dense layer. We initiate a tuner (as RandomSearch), configured to perform a random search with a maximum of 10 trials. During each trial, the tuner explores different combinations of hyperparameters in the hyperparameters space. The model is trained for 10 epochs with a validation split of 20%. Once the search is complete, the model with the best architecture and hyperparameters are obtained. Finally, the best model is trained on the full training dataset, which is then used for sequential forecasting. Again, this reflects how predictions would be made in a real-world application. The predictions and the actual test data are shown below. It is important to note here that the predictors in our LSTM model are the past values of our target variable, that is, our price data. Thus, assume we want to predict the price of our asset (Litcoin) at time t, utilizing our LSTM model, we use the past values of our target variable up to time $t-5$—i.e.,, $t-1$, $t-2$, $t-3$, $t-4$, and $t-5$. This is the same data that one would use, for example, in an AR(5) model.

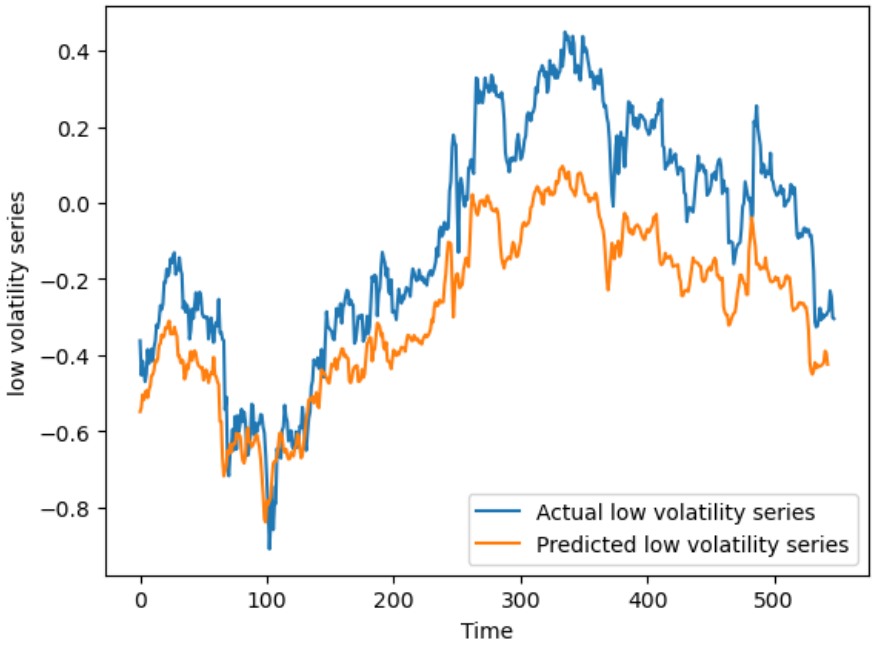

Fig. 9 Prediction vs actual test data (Low volatility series)

Finally, we simply sum the two predictions from the two models—the ARIMA model and the LSTM model—to form a unified prediction, shown below.

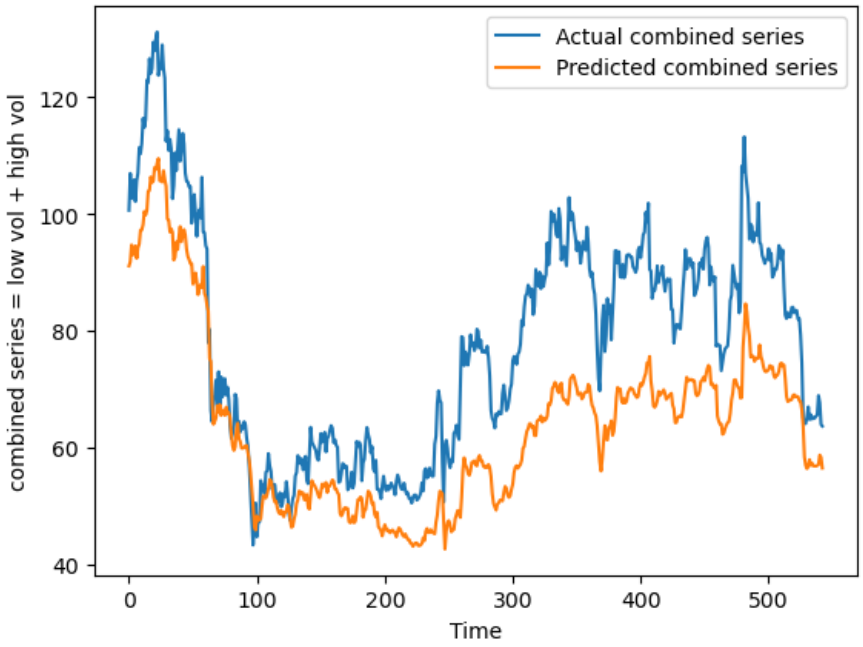

Fig. 10 Prediction (combined series) vs actual test data

Using our predictions (for time t+1), made with data up to time t (i.e.,a one-step ahead forecast), we generate the following simple trading signal: buy when the predicted price is above the current price, and sell when it is below the current price. The initial capital is set at \$1. We then backtest this strategy. Our backtest loop iterates through each time step of the test data and evaluates the strategy's performance based on the predicted/forecasted prices (for time $t+1$) and the actual prices (at time t). To be more precise, our backtest strategy/algorithm does the following. If the prediction for the next time step (i.e., the forecasted price) is higher than the current actual price, we buy the asset (and take a long position), calculates the capital change, and records the transaction. If the prediction is lower, we sell the asset (and take a short position) and record the transaction. If the prediction for the next time step (i.e., the forecasted price) is the same as the current actual price, we hold our current position and neither buy nor sell. Our backtest algorithm calculates the return and capital change at each step and eventually computes the final capital, profit, and cumulative returns. Starting from an initial capital of \$1, our final capital at the end of the test period is \$0.69. In other words, our strategy based on model 1 (where the data is filtered using a moving average filter) generated no profit, but a loss of \$-0.31. The returns and cumulative returns over the test period are shown below.

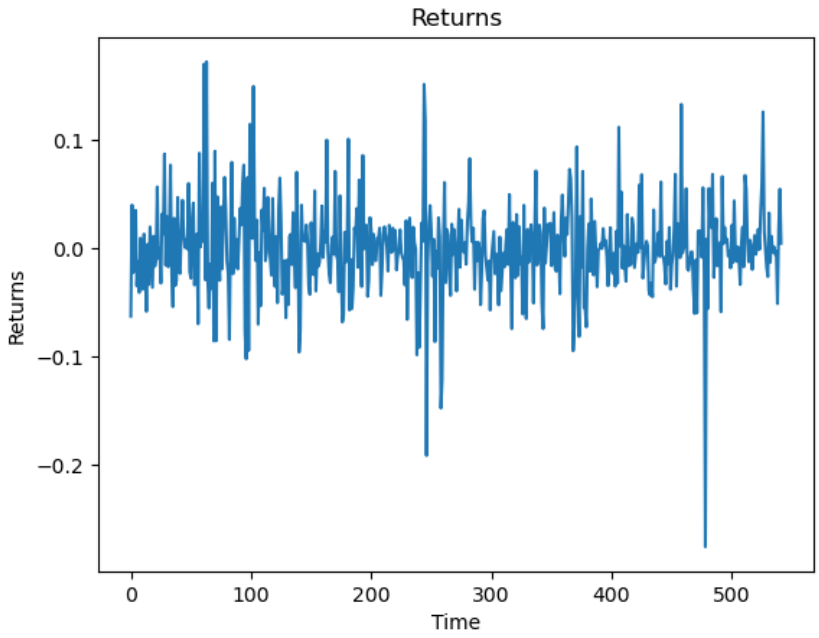

Fig. 11 Returns

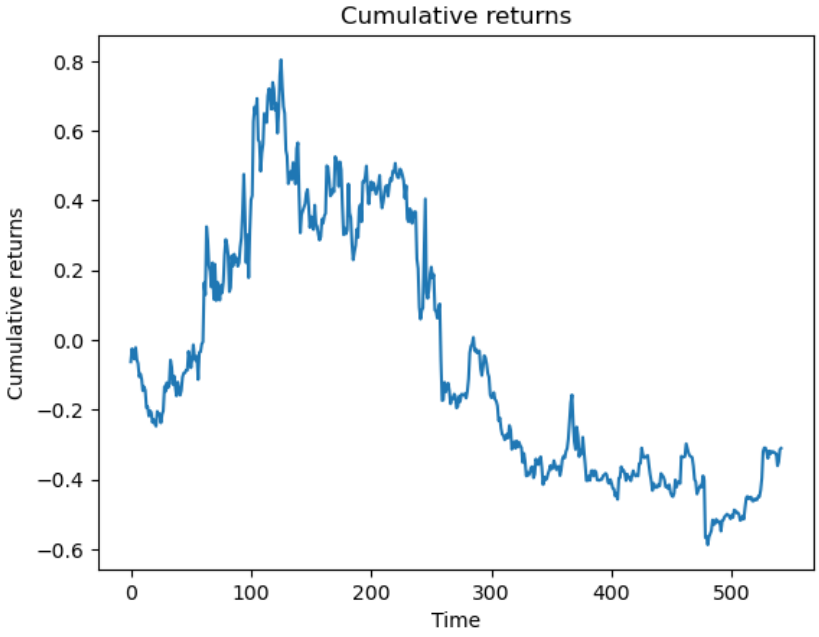

Fig. 12 Cumulative Returns. The strategy yields a cumulative return of -0.31 percent

In the above results, as already mentioned, we used an optimal window size of 188 such that the resulting output of our moving average algorithm follows a normal distribution with a kurtosis of about 3 (allowing for a 5% tolerance). However, would other window sizes give a better result? In view of this, we consider other window sizes, specifically, window sizes of 94 (188/2) and 376 (188 $\times$ 2). The comparison between these models in terms of cumulative returns are shown below.

Table 1 Comparison of cumulative returns across models

	Window size = 94	Window size = 188	Window size = 376
Cumulative return	0.19 percent	-0.31 percent	-0.40 percent

From the above results, the model with the lowest window size of 94 generates the largest cumulative returns relative to the other two models with window sizes of 188 and 376. These results somehow cast some doubt on the ability of the Talib package to find an optimal window size that guarantee that the resulting output of the moving average algorithm follows a normal distribution. The code we used to generate the optimal window size of 188, borrowed from Christopher Niamo's Github's page, is found here. The code, however, does not seem to do what it claims to do. A histogram of the disaggregated data—i.e., the high volatility or trend series and the low volatility or cycle series—are shown below for each model. The low volatility or cycle series from the model with a window size of 94 is relatively more normal than the other two models of window sizes 94 and 376. From these results, it seems that the more normal the low volatility or cyclical component of the data, the higher the cumulative returns. Note that we are not claiming causality here, but just a mere observation.

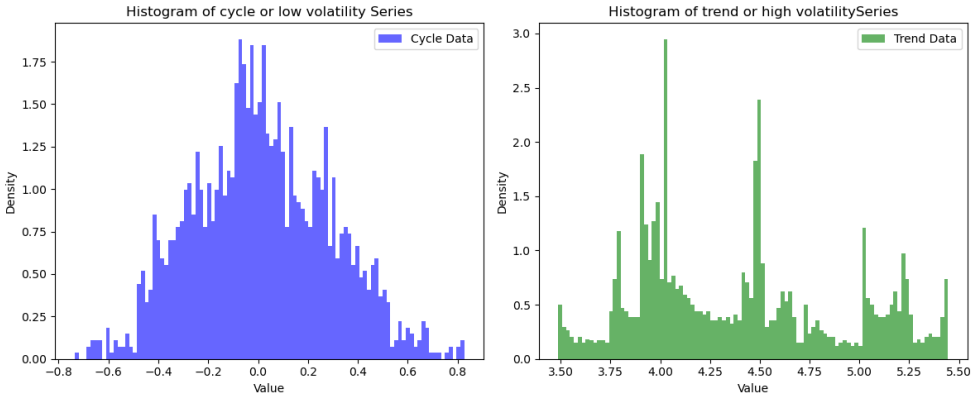

Fig. 13 Model with a window size of 94: Histogram for Trend (high volatility) and cycle (low volatility) series

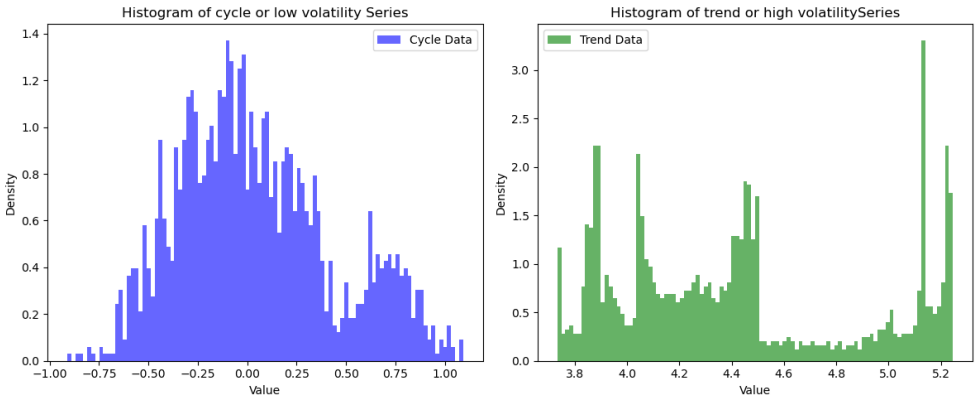

Fig. 14 Model with a window size of 188: Histogram for Trend (high volatility) and cycle (low volatility) series

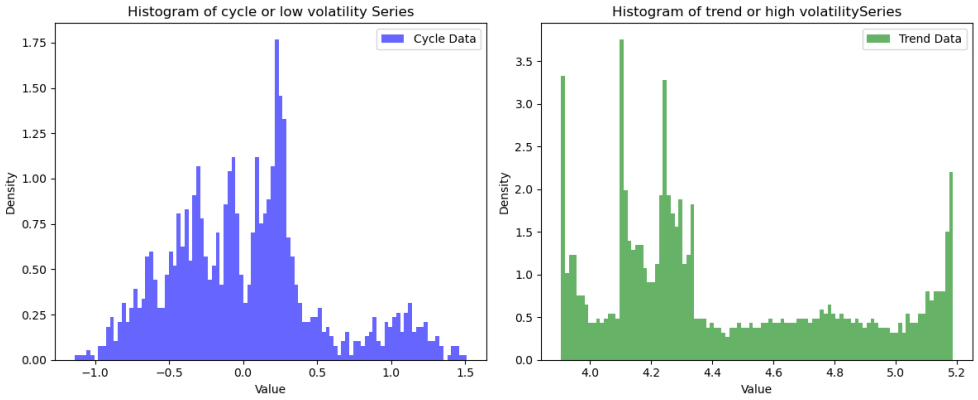

Fig. 15 Model with a window size of 376: Histogram for Trend (high volatility) and cycle (low volatility) series

4.0.2 Model 2: Using an HP filter

In our second hybrid ARIMA-LSTM model, the decomposition of our data is done using the Hodrick-Prescott (HP) filtering algorithm. Everything else from the previous subsection remains the same. In our baseline model, we use a smoothing parameter (λ) of 1000. The disaggregated data are shown below.

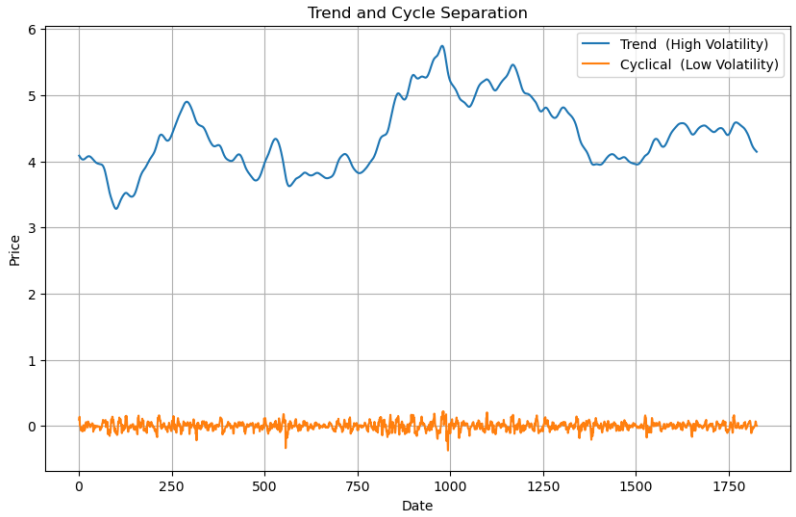

Fig. 16 Trend (high volatility) and cycle (low volatility) separation

Next, as before, the Autoregressive Integrated Moving Average (ARIMA) model is employed to forecast the high volatility series, while the Long Short-Term Memory (LSTM) model is used to forecast the low volatility series. Finally, these two individual forecasts are combined to yield a unified forecast. The predictions from the ARIMA model and the actual test data are shown below. Again, it is worth noting here that an AR(p) model yields a similar result as the ARIMA model used here. The best ARIMA model was found to be ARIMA (3,1,0).

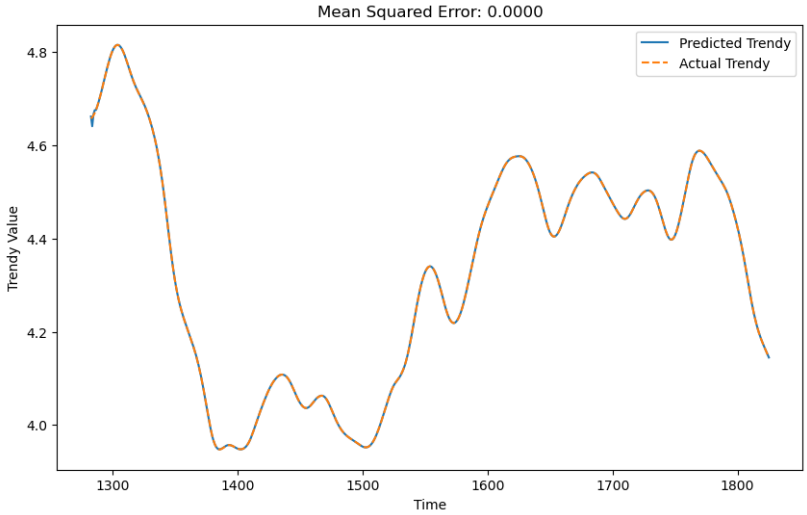

Fig. 17 Prediction vs actual test data (High volatility series)

Like in the previous subsection, we utilize an LSTM model to forecast the low volatility or cycle component of our data. We follow the same procedure as before. The predictions and the actual test data are shown below.

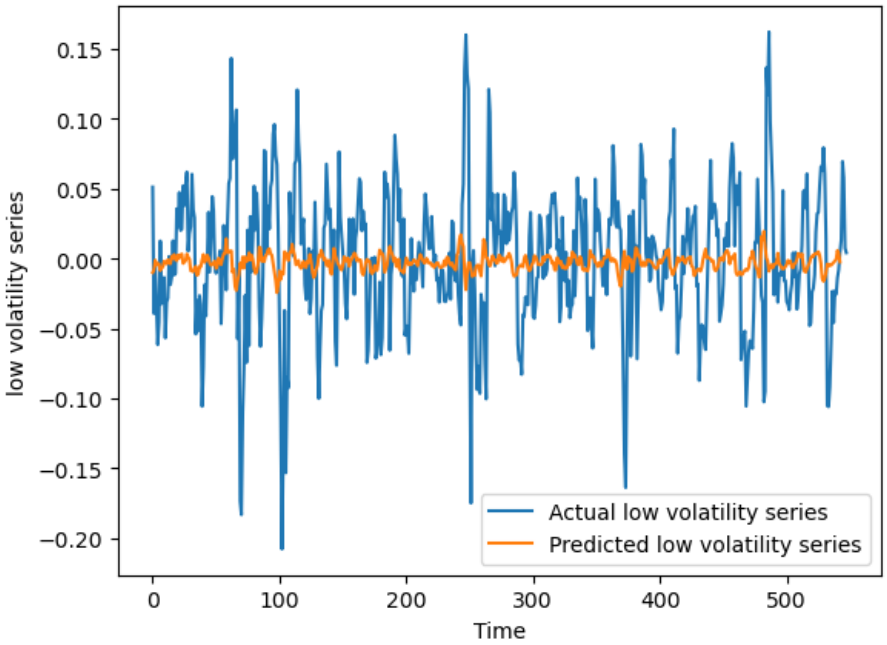

Fig. 18 Prediction vs actual test data (Low volatility series)

Finally, we simply sum the two predictions from the two models—the ARIMA model and the LSTM model—to form a unified prediction, shown below.

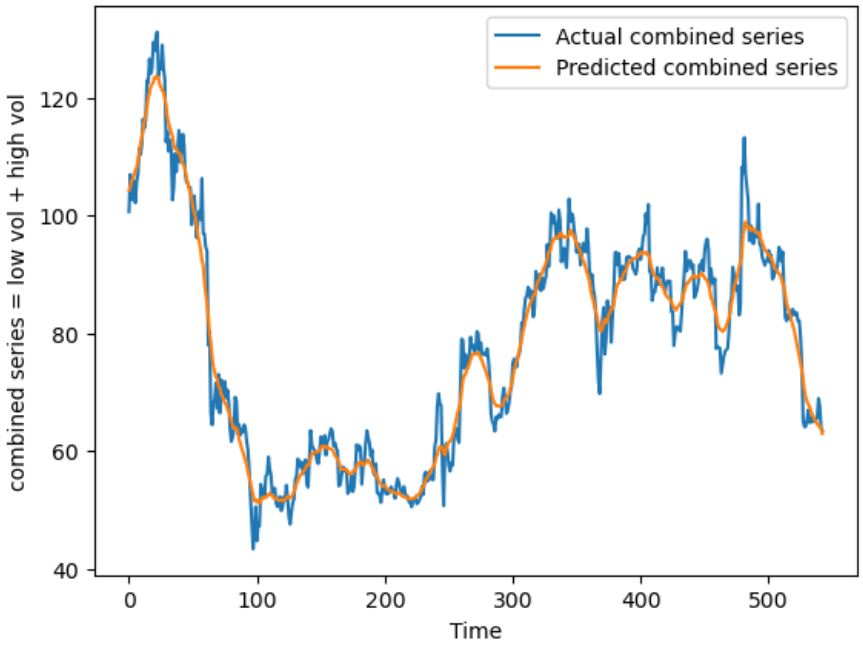

Fig. 19 Prediction (combined series) vs actual test data

Using the same backtest procedures as before, we calculate the return and capital change at each step and eventually computes the final capital, profit, and cumulative returns of our trading strategy (i.e., buy when the predicted price is above the current price and sell when it is below the current price.). Starting from an initial capital of $1, our final capital at the end of the test period is $954.15. In other words, our strategy based on model 2 (where the data is filtered using an HP filter) generated a profit of $954.15 The returns and cumulative returns over the test period are shown below. We can see that model 2 (using the HP filter algorithm) significantly outperforms model 1 (using a moving average filter algorithm).

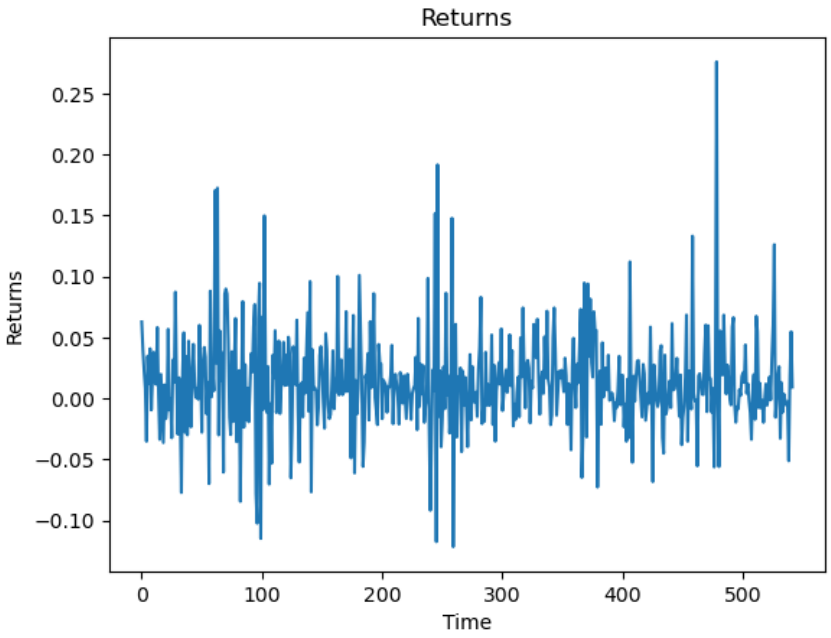

Fig. 20 Returns

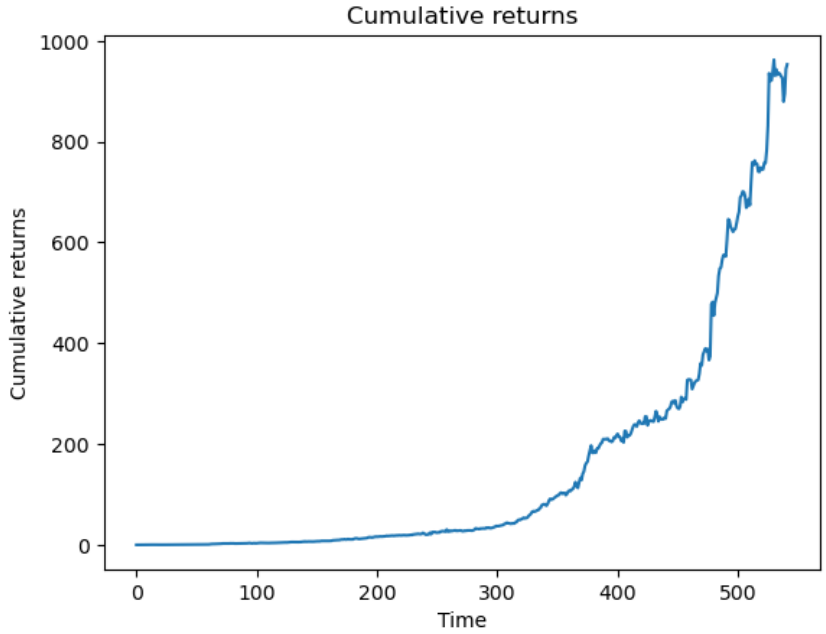

Fig. 21 Cumulative Returns. The strategy yields a cumulative return of 954.15 percent

In the above results, we have used a smoothing parameter (λ) of 1000. However, would other values of λ give better results? In the table below, we consider other values of λ, specifically, $\lambda = 100$ and $\lambda = 10000$. The comparison between these models in terms of cumulative returns are shown in the table below.

Table 2 Comparison of cumulative returns across models

	$\lambda = 100$	$\lambda = 1000$	$\lambda = 10000$
Cumulative return	15959.53 percent	954.15 percent	154.47 percent

From the above results, the model with the smallest $\lambda = 100$ generates the largest cumulative returns relative to the other two models with $\lambda = 1000$ and $\lambda = 10000$. $\lambda = 100$ is significantly lower than the value suggested by Ravn and Uhlig (2002) for daily data, thus, $\lambda = 1600 \times \left(\frac{365}{4}\right)^4$. In other words, one should exercise caution when adhering to generic rules about the appropriate value of λ. As shown above, sometimes, a lower value of λ may be appropriate even for daily data.

Similarly, as in the previous subsection, we show below a histogram of the disaggregated data—i.e., the high volatility or trend series and the low volatility or cycle series—for each of the three models. We observe that the distribution of the low volatility or cycle series from the model with $\lambda = 100$ exhibits a sharp and higher peak around the mean, thus, suggesting less spread and less variability. In other words, the distribution of the low volatility or cycle series from the model with $\lambda = 100$ seems to exhibit thinner tails than the other two models. This relatively lower uncertainty in the low volatility series, perhaps, contributes to the good performance of the model with $\lambda = 100$ relative to the other two models. Later, we examine the significance of the low volatility or cycle series in the performance of the model relative to the high volatility or trend series.

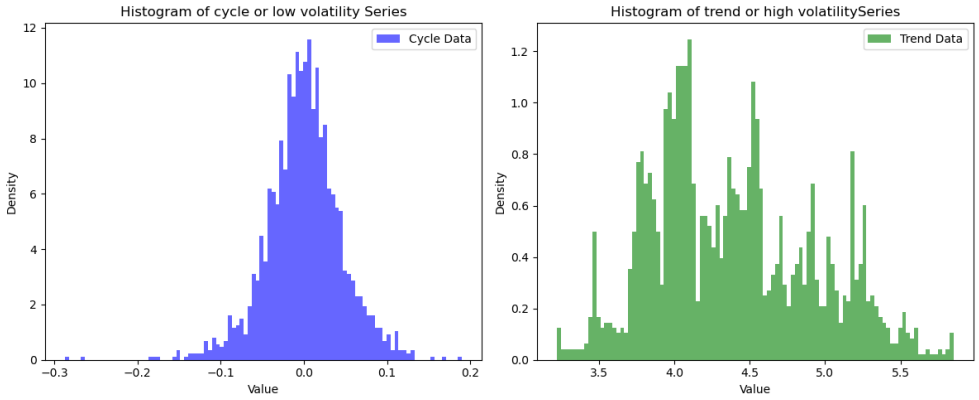

Fig. 22 Model with $\lambda = 100$: Histogram for Trend (high volatility) and cycle (low volatility) series

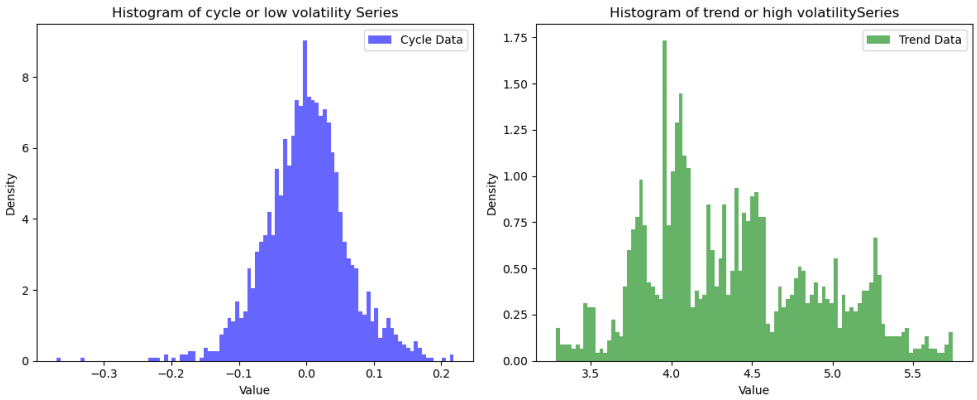

Fig. 23 Model with $\lambda = 1000$:: Histogram for Trend (high volatility) and cycle (low volatility) series

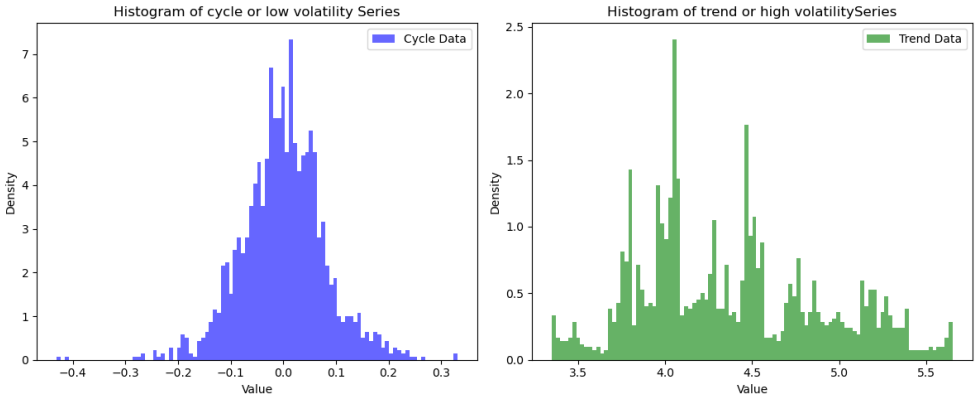

Fig. 24 Model with $\lambda = 10000$:: Histogram for Trend (high volatility) and cycle (low volatility) series

To summarize, our findings so far, suggest that using the HP filter to decompose the data into high and low volatility components is better than using the moving average filter. Whether these results are dependent on the data used in this paper—i.e., a purely random walk process—has not been explored in this paper. We leave this analysis for future research. Future research should also consider other filters besides the moving average filter and the HP filter.

4.0.3 Model performance: Low volatility (cycle) or high volatility (trend)?

In our analysis so far, we have focused on the distributional properties of the low volatility (or cycle) component of the data as the possible driving force behind the performance of the model. Thus, we have so far been silent on the contribution of the high volatility (or trend) component of the data in the performance of the model.

The model using an HP filter with $\lambda = 100$, for example, yields the best result, and it is associated with a low volatility series with a strong central tendency around the mean. An interesting question, however, is, what role does the high volatility series—modeled using the ARIMA model—play in the performance of the model? In other words, is the good performance of the model coming primarily from the distributional properties of the low volatility component of the disaggregated data or is it coming primarily from the high volatility component of the disaggregated data? Put differently, does the high volatility component of the data—modeled using the ARIMA model—play a less or a more significant role in the performance of the model?

To gain some insights into the questions above, we performed the following counterfactual analysis. First, we combine (1) the computed low volatility (or cycle) series from the model with $\lambda = 100$ and (2) the computed high volatility (or trend) series from the model with $\lambda = 10000$. The data are shown below.

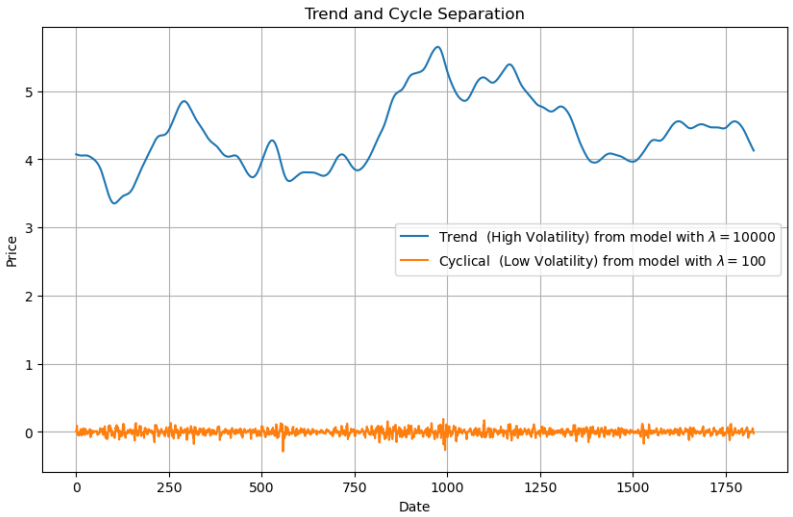

Fig. 25 Predictions of low volatility series (Model with $\lambda = 100$)

In model I below, both the low and high volatility series are from the model with $\lambda = 100$. In model II, however, the low volatility series is from the model with $\lambda = 100$ while the high volatility series is from the model with $\lambda = 10000$. In other words, the sole distinction between the two models lies in the high volatility (or trend) series employed in each model. We found that the cumulative returns declined from about 15959.53 percent (in model I) to about 138.96 percent in model II.

Table 3 Comparison of cumulative returns

	(Model I) Both the low and high volatility series are from the model with $\lambda = 100$	(Model II) Low volatility series is from the model with $\lambda = 100$ and the high volatility series is from the model with $\lambda = 10000$
Cumulative return	15959.53 percent	138.96 percent

These findings seem to indicate that the role of the high volatility (or trend) series in the performance of the model significantly outweigh that of the low volatility (or cycle) series. In other words, the performance of the model seems to be primarily associated with the high volatility (or trend) component of the data rather than the low volatility (or cycle) component. Filters that generate good high volatility (or trend) component of the data should be preferred.

5 Conclusion

This paper applies a hybrid ARIMA-LSTM model to forecast asset prices that exhibits a random walk. Specifically, we disaggregate the data into low volatility (cycle) and high volatility (trend) components, and then model each component separately. We use the ARIMA model to forecast the high volatility (or trend) component of the data and the LSTM model to forecast the low volatility (or cycle) component of the data. We compare the performance of two filtering techniques—the moving average filter and the HP filter—to examine which filtering technique performs better given an asset price that exhibits a random walk. We found that using an HP filter to disaggregate the data into low volatility (cycle) and high volatility (trend) components performs better than using a moving average filter. Within a class of HP filters with different values for the smoothing parameter, λ, we observed that the distribution of the low volatility series from the model with $\lambda = 100$ exhibits a sharp and higher peak around the mean than the other two models with $\lambda = 1000$ and $\lambda = 10000$. The contribution of this distributional property of the low volatility component of the data in the performance of the model, however, seems to be less significant. Overall, we found that the role of the high volatility (or trend) series in the performance of the model significantly outweigh that of the low volatility (or cycle) series. In other words, in the context of our data which exhibits a random walk, it seems a good linear prediction using an ARIMA model is all you need. Future studies should consider other filters such as the Hamilton filter and also other stochastic processes besides a random walk.

References

Bartholomew, D. J. (1971). Review of Time Series Analysis Forecasting and Control. [Review of Review of Time Series Analysis Forecasting and Control., by G. E. P. Box G. M. Jenkins]. Operational Research Quarterly (1970-1977), 22(2), 199–201. https://doi.org/10.2307/3008255

Borges, T. A., Neves, R. F. (2020). Ensemble of machine learning algorithms for cryptocurrency investment with different data resampling methods. Applied Soft Computing, 90, 106187. https://doi.org/10.1016/j.asoc.2020.106187

Chan, E. P. (2017). Machine trading: Deploying computer algorithms to conquer the markets. John Wiley Sons.

Fan, D., Sun, H., Yao, J., Zhang, K., Yan, X., Sun, Z. (2021). Well production forecasting based on ARIMA-LSTM model considering manual operations. Energy, 220, 119708. https://doi.org/10.1016/j.energy.2020.119708

Hadavandi, E., Shavandi, H., Ghanbari, A. (2010). Integration of genetic fuzzy systems and artificial neural networks for stock price forecasting. Knowledge-Based Systems, 23(8), 800–808. https://doi.org/10.1016/j.knosys.2010.05.004

Hileman, G., Rauchs, M. (2017). 2017 Global Cryptocurrency Benchmarking Study (SSRN Scholarly Paper 2965436). https://doi.org/10.2139/ssrn.2965436

Hitam, N. A., Ismail, A. R., Saeed, F. (2019). An Optimized Support Vector Machine (SVM) based on Particle Swarm Optimization (PSO) for Cryptocurrency Forecasting. Procedia Computer Science, 163, 427–433. https://doi.org/10.1016/j.procs.2019.12.125

Kristjanpoller, W., Minutolo, M. C. (2018). A hybrid volatility forecasting framework integrating GARCH, artificial neural network, technical analysis and principal components analysis. Expert Systems with Applications, 109, 1–11. https://doi.org/10.1016/j.eswa.2018.05.011

Nadkarni, J., Ferreira Neves, R. (2018). Combining NeuroEvolution and Principal Component Analysis to trade in the financial markets. Expert Systems with Applications, 103, 184–195. https://doi.org/10.1016/j.eswa.2018.03.012

Ortu, M., Uras, N., Conversano, C., Bartolucci, S., Destefanis, G. (2022). On technical trading and social media indicators for cryptocurrency price classification through deep learning. Expert Systems with Applications, 198, 116804. https://doi.org/10.1016/j.eswa.2022.116804

Park, S., Yang, J.-S. (2022). Interpretable deep learning LSTM model for intelligent economic decision-making. Knowledge-Based Systems, 248, 108907.

https://doi.org/10.1016/j.knosys.2022.108907

Park, S., Yang, J.-S. (2023). Intelligent cryptocurrency trading system using integrated AdaBoost-LSTM with market turbulence knowledge. Applied Soft Computing, 145, 110568. https://doi.org/10.1016/j.asoc.2023.110568

Patel, M. M., Tanwar, S., Gupta, R., Kumar, N. (2020). A Deep Learning-based Cryptocurrency Price Prediction Scheme for Financial Institutions. Journal of Information Security and Applications, 55, 102583. https://doi.org/10.1016/j.jisa.2020.102583

Peng, Y., Albuquerque, P. H. M., Camboim de Sá, J. M., Padula, A. J. A., Montenegro, M. R. (2018). The best of two worlds: Forecasting high frequency volatility for cryptocurrencies and traditional currencies with Support Vector Regression. Expert Systems with Applications, 97, 177–192. https://doi.org/10.1016/j.eswa.2017.12.004

Rathan, K., Sai, S. V., Manikanta, T. S. (2019). Crypto-Currency price prediction using Decision Tree and Regression techniques. 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), 190–194. https://doi.org/10.1109/ICOEI.2019.8862585

Ravn, M. O., Uhlig, H. (2002). On adjusting the Hodrick-Prescott filter for the frequency of observations. Review of economics and statistics, 84(2), 371-376.

Serrano, W. (2022). Deep Reinforcement Learning with the Random Neural Network. Engineering Applications of Artificial Intelligence, 110, 104751. https://doi.org/10.1016/j.engappai.2022.104751

Shui-Ling, Y. U., Li, Z. (2017). Stock price prediction based on ARIMA-RNN combined model. In 4th International Conference on Social Science (ICSS 2017) (pp. 1-6).

Sun Yin, H. H., Langenheldt, K., Harlev, M., Mukkamala, R. R., Vatrapu, R. (2019). Regulating Cryptocurrencies: A Supervised Machine Learning Approach to De-Anonymizing the Bitcoin Blockchain. Journal of Management Information Systems, 36(1), 37–73. https://doi.org/10.1080/07421222.2018.1550550

Wirawan, I. M., Widiyaningtyas, T., Hasan, M. M. (2019). Short Term Prediction on Bitcoin Price Using ARIMA Method. 2019 International Seminar on Application for Technology of Information and Communication (ISemantic), 260–265. https://doi.org/10.1109/ISEMANTIC.2019.8884257

Wu, C.-H., Lu, C.-C., Ma, Y.-F., Lu, R.-S. (2018). A New Forecasting Framework for Bitcoin Price with LSTM. 2018 IEEE International Conference on Data Mining Workshops (ICDMW), 168–175. https://doi.org/10.1109/ICDMW.2018.00032

Zhang, Z., Dai, H.-N., Zhou, J., Mondal, S. K., García, M. M., Wang, H. (2021).

Forecasting cryptocurrency price using convolutional neural networks with weighted and attentive memory channels. Expert Systems with Applications, 183, 115378. https://doi.org/10.1016/j.eswa.2021.115378