Bitcoin Time Series Analysis using ARIMA in Python

Bhavya Dharamsey[1], Jaydeep Ashtamkar[2], Soham Halbandge[3], Dr. Asha Jindal[4]

Researcher (1) (2) (3), Mentor and Head of the Department, Statistics (4), Kishinchand Chellaram College, Churchgate, Mumbai, India

Email: bhavyadharamsey786@gmail.com[1], ashtamkar.jaydeep@gmail.com[2], logzi123@gmail.com[3], asha.jindal@kccollege.edu.in[4]

Abstract

The main objective of our research work is to identify a model to forecast Bitcoin prices. The data in our study is Bitcoin Historical Data from December 2011 to March 2021. We performed data analysis using Autoregressive Integrated Moving Average (ARIMA) modeling techniques in Python. We performed stepwise search to minimize Akaike Information Criterion (AIC). We selected a suitable forecasting model by ARIMA in Python. The model itself generates the optimal p, d, and q values which would be suitable for the data set to provide better forecasting. ARIMA model (1,0,2)(0,0,0)[0] model was the best suitable model that fit our data. Then, we tested data using this model and calculated mean absolute percentage error (MAPE) to be 24.56737% indicating that the model is very accurate.

Keywords: **Time Series Analysis, ARIMA, Bitcoin.**

Introduction

Bitcoin was invented in 2009 by a pseudonymous engineer named Satoshi Nakamoto. His creation solved a problem in computer science which goes as Byzantine Generals Problem and created a trillion-dollar industry [1]. Bitcoin emerged out of the 2008 global economic crisis when big banks were caught misusing borrowers' money, manipulating the system, and charging exorbitant fees. To address such issues, Bitcoin creators wanted to put the owners of bitcoins in charge of the transactions, eliminate the middleman, cut high-interest rates and transaction fees, and make transactions transparent. They created a distributed network system, where people could transparently control their funds. That's more valuable than any of the tech unicorns founded in the last two decades. Bitcoin, as a dominant cryptocurrency, has started to gain momentum in financial institutions around the world as a major sovereign and the private organisation started acknowledging its impact. It remains to be seen how it affects mass consumer utility in the near future but to gauge the rise of Bitcoin, its history offers crucial insights.

Several statistical methods like time series analysis, regression analysis are available for forecasting. Using these methods, we select a suitable forecasting model. The selected forecasting model should have higher accuracy, which means minimum error.

Methodology

Our main objective is to identify a best suitable model for predicting the bitcoin price, using time series modeling techniques. We have used Jupyter Notebook, Python Kernel and various Python libraries for data preprocessing and data analysis.

About Data

The given data is in CSV format includes historical bitcoin market data at 1-min intervals for select bitcoin exchanges where trading takes place for the time period of to December 2020, with updates of OHLC (Open, High, Low, Close), Volume in BTC and indicated currency, and weighted bitcoin price [2]. We used the data set about the hourly price of bitcoin over a period from December 2011 to March 2021. Dataset has 4857377 observations under 8 variables namely

1. Timestamp
2. Open
3. High
4. Low
5. Close
6. Volume_(BTC)
7. Volume_(Currency)
8. Weighted_Price

Data Preprocessing

Among given variables, Timestamp variable is as an integer datatype rather than as dates. The method fromtimestamp() of the date class computes the date corresponding to a given timestamp and returns it. Hence, we have used the fromtimestamp() function which converts the arguments to dates. Before proceeding to analysis, there is a need to check whether missing values are present in data. If some of the observations are missing, then we must impute the missing observations.

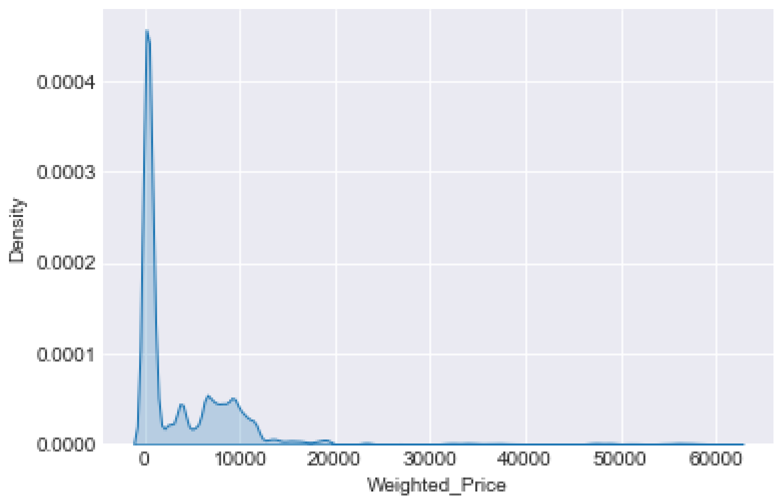
Except for the Timestamp variable, we have 1243608 missing observations which constitutes 25.60246% observations of all the dataset.

Handling Missing Values

Time series data has a lot of variations against time. Hence, imputing using backfill and forward fill isn't the best possible solution to address the missing value problem. A more appropriate alternative would be to use interpolation methods, where the values are filled with incrementing or decrementing values. **Linear interpolation** is an imputation technique that assumes a linear relationship between data points and utilises non-missing values from adjacent data points to compute a value for a missing data point [3].

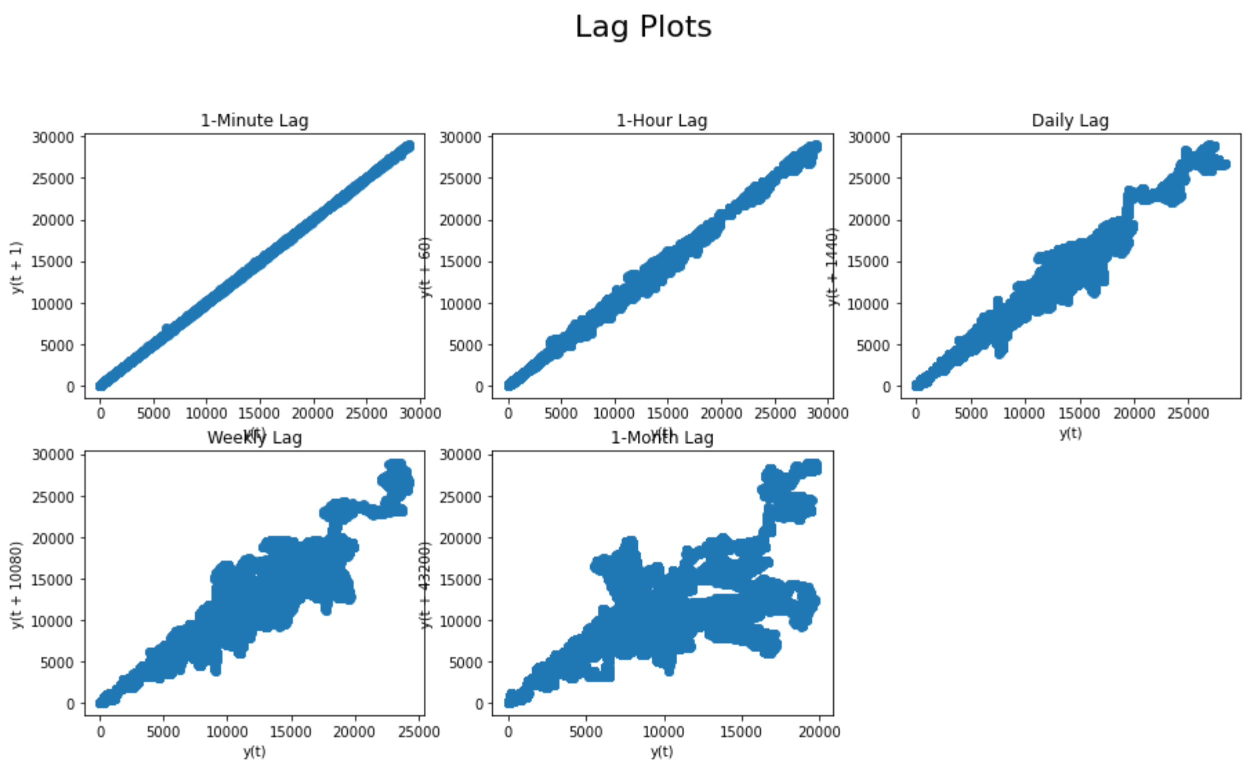
Data Exploration

Summarizing the data with Density plots to see where the mass of the data is located.



[Type here]

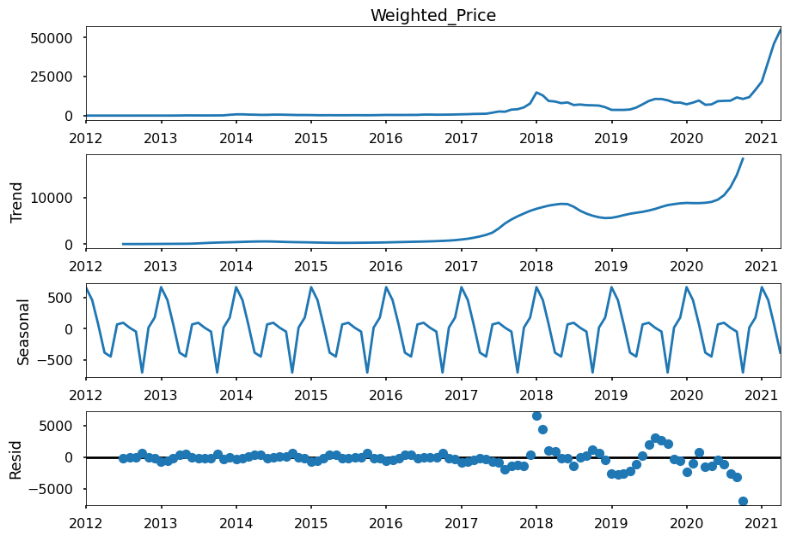
Lag plots are used to observe the autocorrelation. These are crucial when we try to correct the trend and stationarity and we have to use smoothing functions. Lag plot helped us to understand the data better.

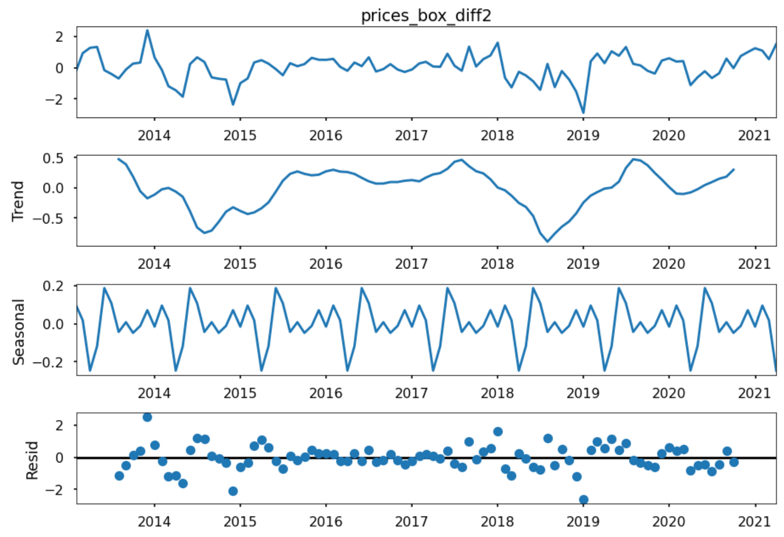


We can see that there is a positive correlation for minute, hour and daily lag plots. We observe absolutely no correlation for month lag plots. It makes sense to re-sample our data at most at the Daily level, thereby preserving the autocorrelation as well. For that, we use a process called **time resampling** to aggregate data into a defined time period. The pandas library has a resample() function which resamples such time series data. The resample function in pandas is similar to its groupby method as it is essentially grouping according to a certain time span. data.resample() is used to resample the stock data. The '24H' stands for daily frequency, and denotes the offset values by which we want to resample the data.

Time Series Decomposition & Statistical Tests

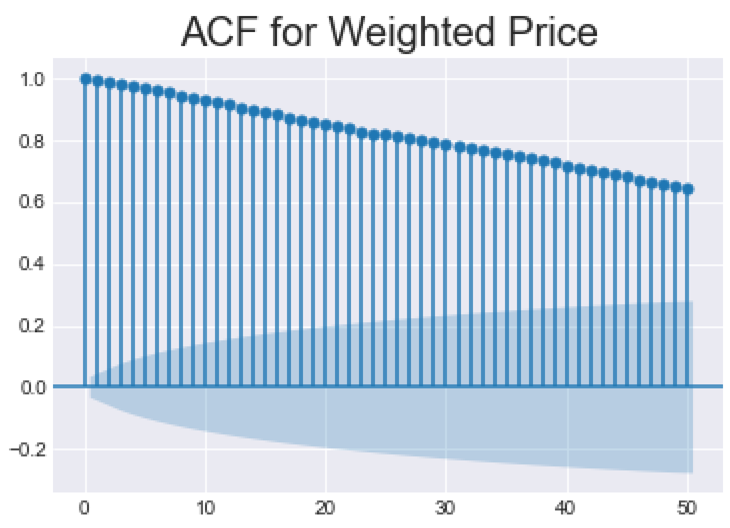
We decomposed the time series data into trend, seasonal and remainder components. The series has been decomposed as an additive or multiplicative combination of the base level, trend, seasonal index and the residual. The seasonal_decompose in statsmodels is used to implement the decomposition. Post time series decomposition we don't observe any seasonality. Also, there is no constant mean, variance and covariance, hence we conclude that the series is Non Stationary [4].



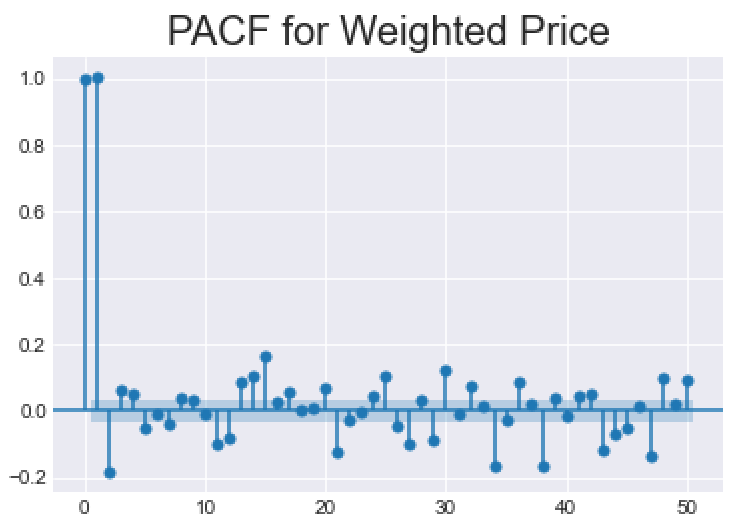


We plot Autocorrelation Plot (ACF Plot) to visualise autocorrelation in data that changes over time).

[Type here]



The above graph shows that effects rarely deteriorate over time, so past values affect the present ones. The more lags we include, the better our model will fit the dataset, now the risk is coefficients might predict the dataset too well, causing an overfitting. In our model, we always try to include only those lags which have a direct effect on our present value.



Coefficient values for lag > 5 are statistically not significant and their impact on the model is minimal [5].

KPSS Test

[Type here]

The KPSS test, short for Kwiatkowski-Phillips-Schmidt-Shin (KPSS), is a type of Unit root test that tests for the stationarity of a given series around a deterministic trend. Here, the null hypothesis is that the series is stationary. That is, if p-value is < 0.05, then the series is non-stationary and vice versa [6].

The p-value reported by the test is the probability score based on which we can decide whether to reject the null hypothesis or not. If the p-value is less than a predefined alpha level (typically 0.05), we reject the null hypothesis.

The KPSS statistic is the actual test statistic that is computed while performing the test. The number of lags reported is the number of lags of the series that was actually used by the model equation of the kpss test.

In order to reject the null hypothesis, the test statistic should be greater than the provided critical values. If it is in fact higher than the target critical value, then that should automatically reflect in a low p-value. That is, if the p-value is less than 0.05, the kpss statistic will be greater than the 5% critical value.

Results of KPSS Test:

```
Test Statistics : 0.7963392979047433

p-value : 0.01

Critical Values : {'10%': 0.119, '5%': 0.146, '2.5%': 0.176, '1%': 0.216}
```

Hence, the KPSS test concludes that the series is **not stationary**.

ADF Test

The null hypothesis of the test is the presence of unit root, that is, the series is non-stationary. The alternate hypothesis is that the series is stationary or trend-stationary. If p-value of less than 5% means we can reject the null hypothesis that there is a unit root [7].

Results of ADF Test:

[Type here]

```
Test Statistic 6.302121

p-value 1.000000

#Lags Used 29.000000

Number of Observations Used 3349.000000

Critical Value (1%) -3.432304

Critical Value (5%) -2.862403

Critical Value (10%) -2.567230

dtype: float64
```

Hence, the ADF test concludes that the series is **stationary**.

Conclusion

KPSS concludes that the series is not stationary and ADF says the series is stationary. This precisely indicates that the series is stationary, hence we have used differencing to make the series stationary [8].

Rolling Windows

The data is noisy due to high fluctuations in the market. As a result, it becomes difficult to gauge a trend or pattern in the data. As we're looking at daily data, there's quite a bit of noise present. It would be efficient if we could average it on a weekly basis, which is where a rolling mean comes in [9].

A rolling mean, or moving average, is a transformation method which helps average out noise from data. It works by simply splitting and aggregating the data into windows according to function, such as mean(), median(), count(), etc. For this dataset, we have used a rolling mean for 3, 7 and 30 days.

Cross Validation

To measure the performance of our forecasting model, We usually want to split the time series into a training period and a validation period which is called **fixed partitioning**. If the time series has some seasonality, we generally want to ensure that each period contains a whole number of seasons [10].

We'll train our model on the training period and we'll evaluate it on the validation period. Here's where we experimented to find the right architecture for training. And work on it and its hyper parameters, until we get the desired performance, measured using the validation set. Once we've done that, we can re-train using both the training and validation data and then test on the test (or forecast) period to see if our model will perform just as well.

And if it does, then we could take the unusual step of retraining again, using also the test data because the test data is the closest data we have to the current point in time. And as such it's often the strongest signal in determining future values. If our model is not trained using that data, too, then it may not be optimal.

Here, we opted for a hold-out based validation. Hold-out is used very frequently with time-series data. In this case, we will select all the data for 2020 as a hold-out and train our model on all the data from 2012 to 2019.

ARIMA Model

ARIMA is a class of model that captures a suite of different standard temporal structures in time series data. The ARIMA stands for,

AR: **Autoregression** A model that uses the dependent relationship between an observation and some number of lagged observations.

I: **Integrated** The use of differencing of raw observations in order to make the time series stationary.

MA: **Moving Average** A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

A standard notation is used of **ARIMA(p, d, q)** where the parameters are substituted with integer values to quickly indicate the specific ARIMA model being used.

p: The number of lag observations included in the model, also called the lag order.

d: The number of times that the raw observations are different, also called the degree of differencing.

q: The size of the moving average window, also called the order of moving average.

Let, y_t denote the dth difference of Y_t. Then, mathematically, ARIMA(p, d, q) model is given by,

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + e_t + \theta_1 e_{t-1} + \theta_2 e_{t-2} + ... + \theta_q e_{t-q}$$

where, $y_t = \left(Y_t - Y_{t-1}\right) - \left(Y_{t-1} - Y_{t-2}\right) - \dots - \left(Y_{t-d+1} - Y_{t-d}\right)$, p is the number of autoregressive terms, q is the number of lagged forecast errors and d is the number of non-seasonal differences needed for stationarity [11].

If p ≠ 0, d = 0 and q = 0, then, it is an AR(p) model. AR indicates an autoregressive model. If p = 0, d = 0 and q ≠ 0, then, it is the MA(q) model. MA indicates a moving average model [12].

We select the best model from the set which provides the minimum AIC(Akaike's Information Criterion)

ARIMA(2,0,2)(0,0,0)[0] : AIC=39275.523,

ARIMA(0,0,0)(0,0,0)[0] : AIC=40301.927,

ARIMA(1,0,0)(0,0,0)[0] : AIC=inf,

ARIMA(0,0,1)(0,0,0)[0] : AIC=39560.737,

ARIMA(0,0,0)(0,0,0)[0] : AIC=40317.090,

ARIMA(1,0,2)(0,0,0)[0] : AIC=39254.975,

ARIMA(0,0,2)(0,0,0)[0] : AIC=39285.483,

ARIMA(1,0,1)(0,0,0)[0] : AIC=39378.606,

ARIMA(1,0,3)(0,0,0)[0] : AIC=39275.258,

ARIMA(0,0,3)(0,0,0)[0] : AIC=39280.527,

ARIMA(2,0,1)(0,0,0)[0] : AIC=39379.497,

ARIMA(2,0,3)(0,0,0)[0] : AIC=39257.181,

ARIMA(1,0,2)(0,0,0)[0] : AIC=39252.737,

ARIMA(0,0,2)(0,0,0)[0] : AIC=39287.673,

ARIMA(1,0,1)(0,0,0)[0] : AIC=39377.232,

ARIMA(2,0,2)(0,0,0)[0] : AIC=39311.956,

ARIMA(1,0,3)(0,0,0)[0] : AIC=39280.090,

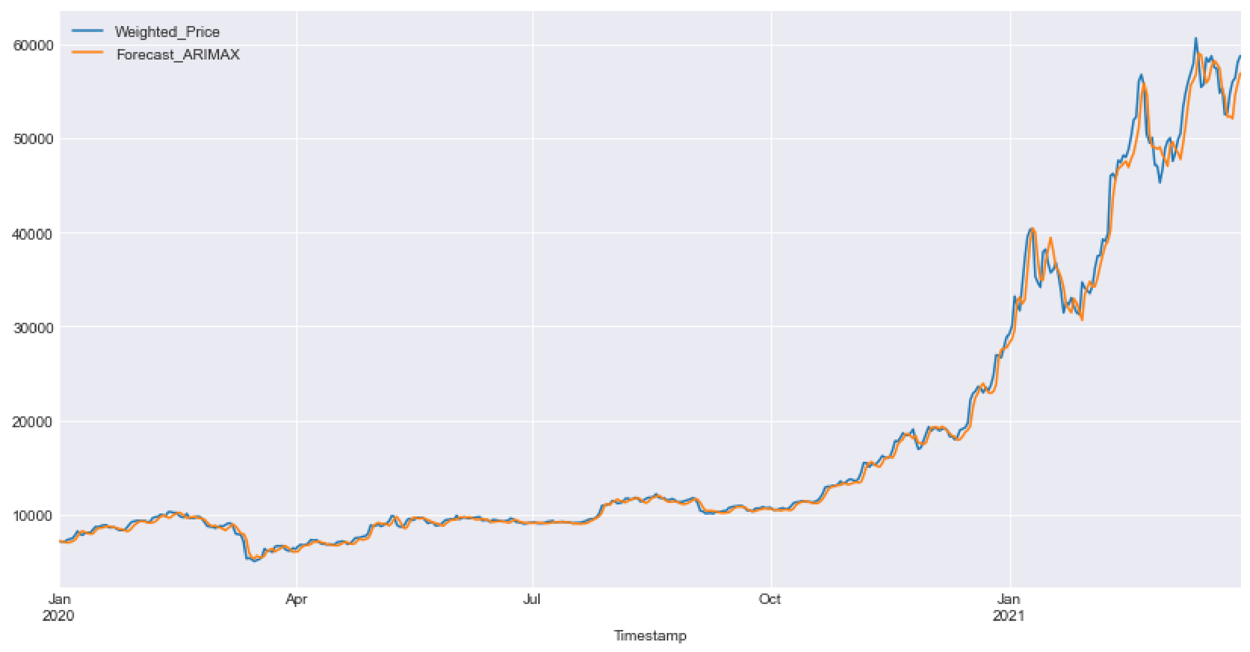
ARIMA(0,0,1)(0,0,0)[0] : AIC=39558.115,

ARIMA(0,0,3)(0,0,0)[0] : AIC=39276.410,

ARIMA(2,0,1)(0,0,0)[0] : AIC=39378.084,

ARIMA(2,0,3)(0,0,0)[0] : AIC=39284.957,

Out of the available models AIC is minimum for (1,0,2)(0,0,0)[0] which becomes our best model. We consecutively plot our Forecast ARIMA model of Weighted price against Timestamp.



We also checked for Root Mean Square Error (RMSE) [13] which is a standard way to measure the error of a model in predicting quantitative data and Mean Absolute Error (MAE) [14] measures the average magnitude of the errors in a set of predictions, without considering their direction. It's the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight.

RMSE=1072.5007477777647

MAE=594.8043834852598

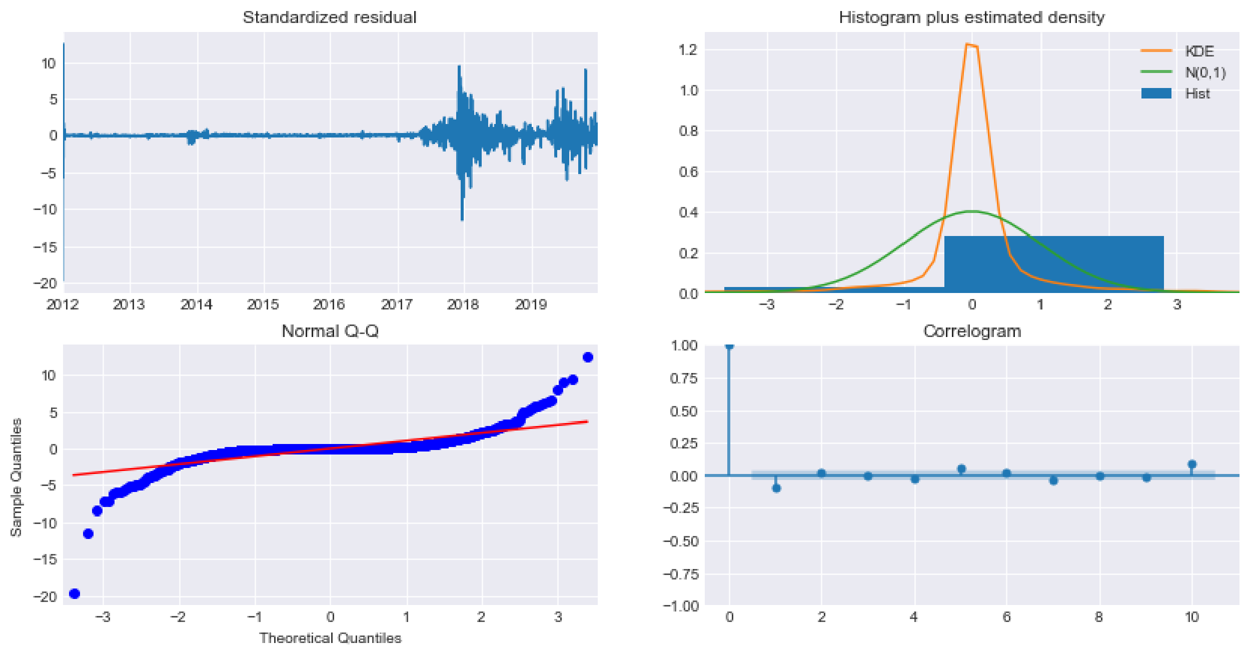
Both MAE and RMSE give average model prediction error in units of the variable under study. MAE and RMSE both can range from 0 to ∞ and are indifferent to the direction of errors. They are negatively-oriented scores, which means lower values are better.

Conclusion

We further analyse model diagnostics to ensure that none of the assumptions made by the model have been violated. The plot_diagnostics object allows us to quickly generate model diagnostics and investigate for any unusual behavior [15].

[Type here]

Our primary concern is to ensure that the residuals of our model are uncorrelated and normally distributed with zero-mean. If the seasonal ARIMA model does not satisfy these properties, it is a good indication that it can be further improved.



These diagnostics lead us to conclude that our model produces a satisfactory fit that helps us to understand our time series data and forecast future values. Hence, the Auto ARIMA model (1,0,2)(0,0,0)[0] could forecast bitcoin prices effectively.

References

[1] Bitcoin nearing a trillion dollar total value https://www.forbes.com/sites/billybambrough/2021/02/18/as-bitcoin-total-value-nears-1-trillion-these-crypto-prices-are-leaving-it-in-the-dust/?sh=34048a024689

[2] Data https://www.kaggle.com/mczielinski/bitcoin-historical-data

[3] Linear Interpolation https://www.analyticsvidhya.com/blog/2021/06/power-of-interpolation-in-python-to-fill-missing-values/

[4] Time Series Decomposition https://otexts.com/fpp2/decomposition.html

[Type here]

[5] ACF/PACF https://machinelearningmastery.com/gentle-introduction-autocorrelation-partial-autocorrelation/

[6] KPSS https://www.statisticshowto.com/kpss-test/

[7] ADF https://www.statisticshowto.com/adf-augmented-dickey-fuller-test/

[8] Stationarity and Differencing https://otexts.com/fpp2/stationarity.html

[9] Rolling Window https://medium.com/making-sense-of-data/time-series-next-value-prediction-using-regression-over-a-rolling-window-228f0acae363

[10] Cross Validation https://www.analyticsvidhya.com/blog/2018/05/improve-model-performance-cross-validation-in-python-r/

[11] Statistical forecasting Notes https://people.duke.edu/~rnau/411home.htm

[12] Auto ARIMA https://towardsdatascience.com/time-series-forecasting-using-auto-arima-in-python-bb83e49210cd

[13] RMSE https://www.statisticshowto.com/probability-and-statistics/regression-analysis/rmse-root-mean-square-error/

[14] MAE https://www.statisticshowto.com/absolute-error/

[15] Plot Diagnostics https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima.model.ARIMAResults.plot_diagnostics.html