A Statistical Analysis of Bitcoin Daily Returns And A Price Forecast With Monte-Carlo Simulation And ARIMA *

Baoye Chen

May 12, 2021

Abstract

In this project, I analyze the daily return pattern of Bitcoin and perform a price forecast using (non-parametric) Monte-Carlo Simulation. Then, I also use the auto-regressive integrated moving average (ARIMA) model to fit the time series and use this model to perform a rolling forecast of the Bitcoin price.
Old ideas: MLE, Time Series, Auto-correlation, Hypothesis Testing, Monte-Carlo
New ideas: Augmented Dickey-Fuller, Shapiro-Wilk, Kolmogorov-Smirnov, Kernel Density Estimator, Akaike Information Criterion, Bayesian Information Criterion, ARIMA

1 Introduction

During the past few months, Bitcoin has drawn unprecedentedly high attentions from investors from all over the world. The topic "Bitcoin" has been trending a lot in social medias such as Twitter, Reddit, and Weibo, which are filled with retail investors from all over the world. Other than the retail investors, Bitcoin has also attracted investments from tech companies such as Tesla as well as Bulge Brackets financial institutions such as Morgan Stanley, who have confirmed that they are offering its wealth management clients exposure to bitcoin by way of pair of external crypto funds [1].

In this project, I decide to perform some statistical analysis of this popular financial instrument. The rest of the work is as follows. In Section 2, I introduce the source of the data and describe some processings I did on the data. In Section 3, I analyze the stationarity of the price and daily return. In section 4, I perform statistical analysis on the distribution of daily return. In section 5, I use Monte-Carlo Simulation to make a prediction of Bitcoin's future prices. In section 6, I fit the time series with an autoregressive integrated moving average (ARIMA) model and perform a rolling forecast of Bitcoin price. Section 7 concludes the project, admits some drawbacks of the study, and discusses possible future topics of research.

2 Data Description

The data used in this study comes from Quandl [8]. It contains the daily closing price of Bitcoin from 2010-08-16 to 2021-05-05. Below is the description statistics and the plot [1] of the closing price data.

*Codes, Figures, and (most) simulation outputs can be found in this repository: https://github.com/baoye-chen/Bitcoin_Monte-Carlo_ARIMA.git

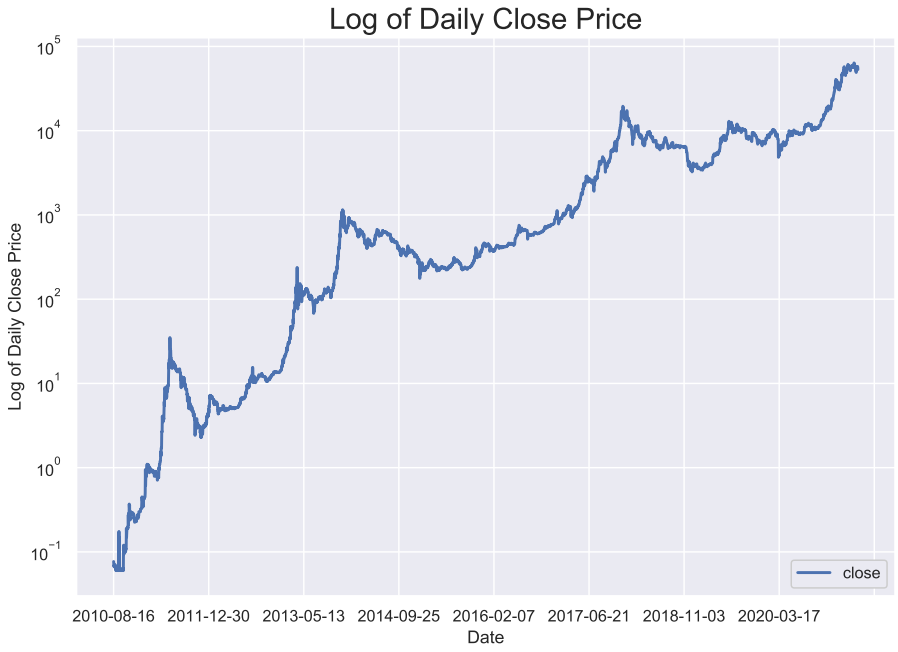

Figure 1: Log of Daily Closing Price

	close
count	3915.000000
mean	4486.874664
std	9240.883542
min	0.060000
25%	93.488915
50%	581.070000
75%	6607.448333
max	63554.440000

From the plot of this time series, we can clearly see the closing price of Bitcoin is trending up and not stationary. Therefore, to better perform the analysis, I decide to add a column of daily return. Usually, the return is defined as

$$return = \frac{P_t}{P_{t-1}} - 1$$

In this project, I decided to express the return with simply

$$return = \frac{P_t}{P_{t-1}}$$

In this way, it is easier for calculation and manipulation of the data when doing the Monte-Carlo Simulation and price forecast. Figure [2] is a plot of the daily return. From the plot, we can see there are some abnormally high returns in the year of 2010 and 2011. Therefore, to obtain a better result, I decide to drop the data from those years and only use results starting from 2012-01-01. Figure [3] is the resulting daily return plot after dropping the data from 2010 and 2011. From this plot, we can see return of Bitcoin seems to be stationary. The rigorous analysis of stationarity will be reserved for section 3.

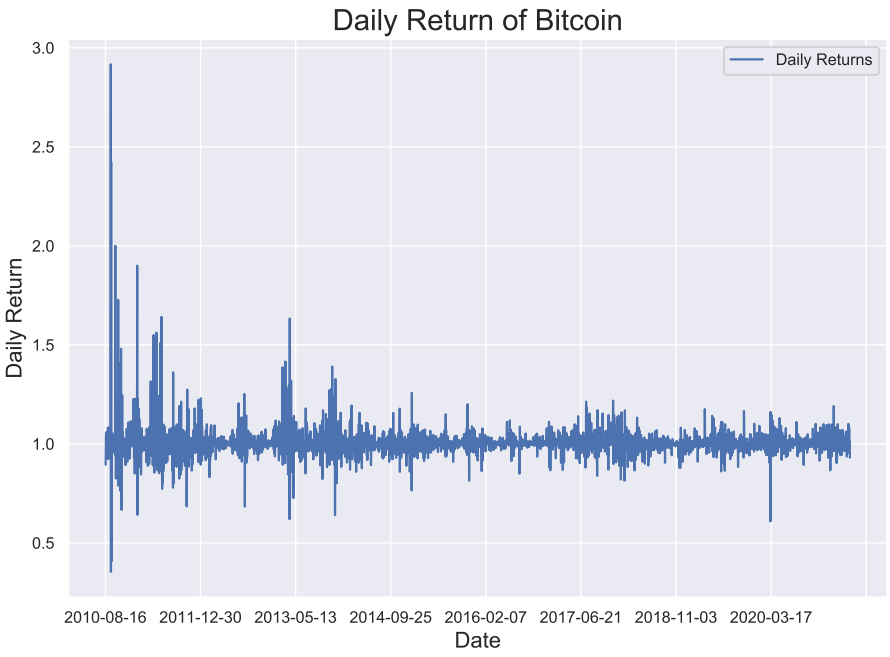

Figure 2: Daily Return

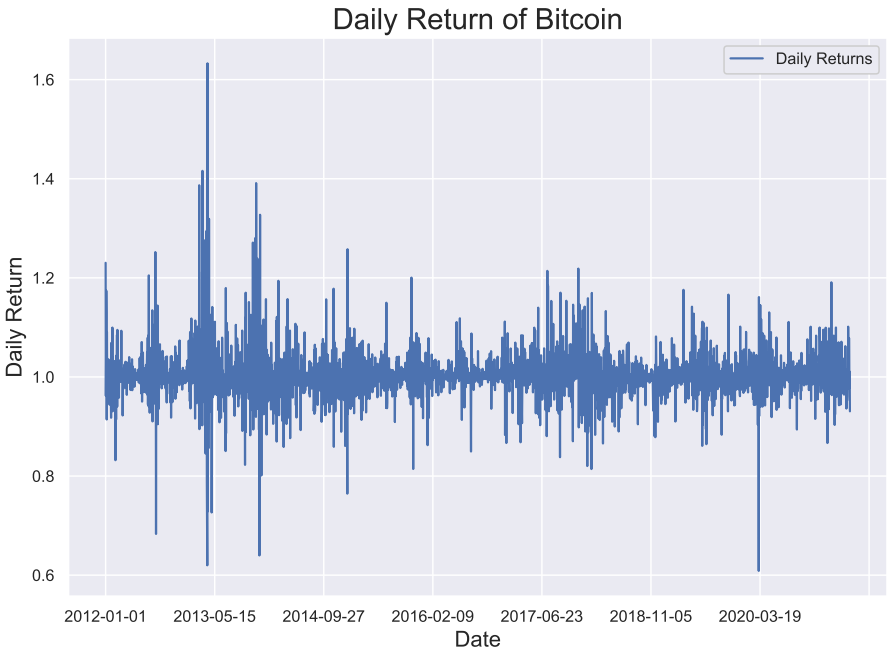

Figure 3: Processed Daily Return Data

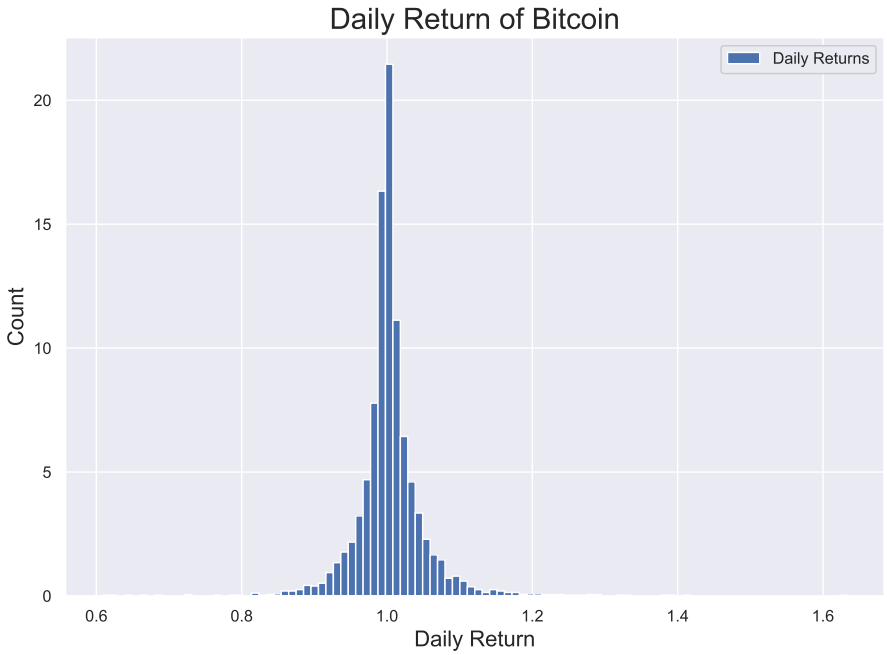

Figure 4: Distribution of Daily Return

Finally, figure [4] is a plot of the distribution of the daily return. We can see at the first glance that the distribution of the return looks normal. Details about the analysis of this distribution will be presented in section 4.

3 Stationarity Analysis

In this section, I perform the stationarity analysis of the closing price and the daily return. Generally, a time series is said to be stationary if a shift in time doesn't cause a change in the shape of the distribution [13]. In other words, if a time series is (weakly) stationary, its mean $\mu(t)$ and auto-correlation $\gamma(\tau)$ (where τ is the lag) does not depend on time t.

By observing the plot of the time series, one can have some intuitions about the statinoarity. That is, from the plot of closing price [1] and the plot of daily return [3], we can see that the closing price is non-stationary since it is trending upward, while the daily return is stationary, since the values seem to be fluctuating around a mean. To formally analyze stationarity of the time series, we need to perform the Augmented Dickey-Fuller Test.

3.1 Augmented Dickey-Fuller Test

The Augmented Dickey-Fuller Test (ADF) is an extension of the Dickey-Fuller Test [3]. It is a test to decide whether a series is stationary or non-stationary. The null hypothesis of the test is:

$$H_0\text{: A unit root is present in the time series.}$$

A unit root is a characteristic of a time series that makes it non-stationary [7]. In a more technical sense, a unit root exists in a time series when $\gamma = 0$ in the following model [2]:

$$\Delta Y_t = \alpha + \beta t + \gamma Y_{t-1} + \delta_1 \Delta_{y_{t-1}} + \delta_2 \Delta_{y_{t-2}} + ...$$

Here, Y_t is the value at time t and Y_{t-1} is the value at time $t-1$. If the value $\gamma = 0$, the time series is non-stationary. So equivalently, we can say the null hypothesis of the ADF test is $\gamma = 0$ in the model above. The test statistics is

$$DF = \frac{\hat{\gamma}}{SE(\hat{\gamma})}$$

In this part, I perform ADF Test for three series: daily closing price, differenced daily closing price, and return. Differencing is a straightforward process, which is just to compute the differences between consecutive observations. It is useful because even if the time series of the closing price is non-stationary, the differences between each value might be stationary. That is why differencing is a widely used way to make a non-stationary process stationary. The following is the result of ADF Test:

Tested Data	**Test Statistics**	**p-value**
Closing Price	3.387267	$1.000000e+00$
Differenced Closing Price	-9.216125	$1.831870e-15$
Daily Return	-9.733915	$8.853900e-17$

From the result of the test, we can see that we have a strong evidence that the closing price is non-stationary because the p-value is essentially 1. Moreover, we have strong evidence that both the differenced closing price and the daily return are stationary, because both of their p-values are essentially 0. Thus, the ADF test confirms the previous observation that the closing price is non-stationary, while the differenced closing price and the daily return are stationary.

4 Distribution of Daily Return

In this part, I try to figure out the distribution of the daily return. Since from the graph [4], we see that the data looks normal, I start by fitting the data to a normal distribution. By "fitting" the data to a distribution, I calculate the MLE of the given model and plot the fitted graph [10]. The result suggests that the MLE estimators of normal distribution for this data is

$$\mu = 1.00398478$$
$$\sigma = 0.0500097$$

And the figure [5] is a plot of the fitted line along with the histogram of the real return data. From the plot, we can observe that compared to the fitted normal distribution, the real data has a very pointy middle part and a long tail, which is not abnormal for a financial data. Therefore, to formally test if the data is normal, I decide to perform some tests of normality.

4.1 Test For Normality

In this part, I perform three types of tests to determine if the data is indeed well-fitted to a normal distribution: Kolmogorov–Smirnov test (K–S test) and Shapiro-Wilk Test. The following are some brief introductions to these three tests:

- K-S Test [11]: This test can be used to determine if a set of data comes from any given distribution.
  H_0: the data comes from the specified distribution.
  The test statistics is
  $$D = \sup_x |F_0(x) - F_{data}(x)|$$
  where $F_0(x)$ is the cdf of the hypothesized distribution, and the $F_{data}(x)$ is the empirical distribution of the data.

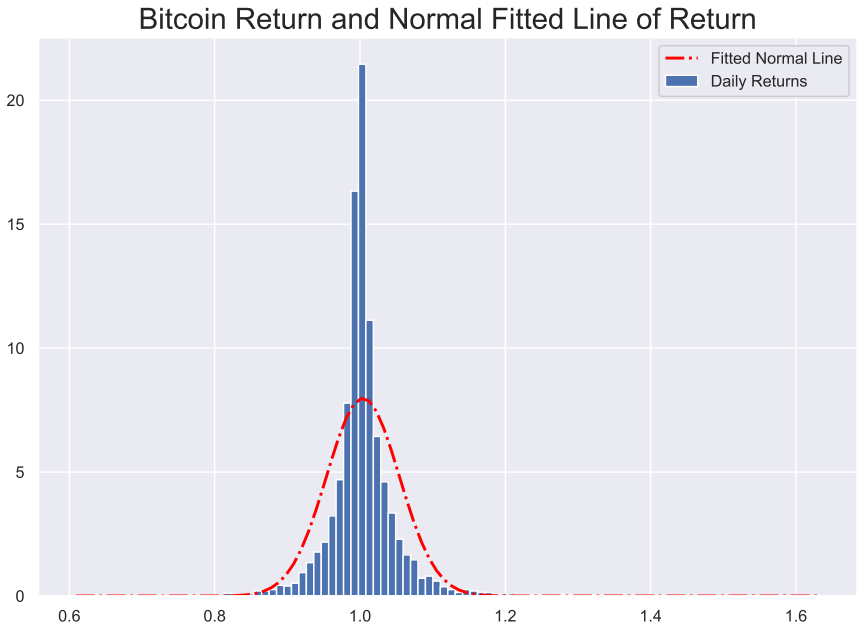

Figure 5: Fit to Normal Distribution

- Shapiro-Wilk Test [12]:
  H_0: the data comes from a normal distribution.
  The test statistics is
  $$W = \frac{(\sum_{i=1}^{n} a_i x_{(i)})^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$
  where x_i are ordered random sample values and a_i are constants generated from the covariances, variances, and means of the sample from normally distributed sample.

The following is the result of the tests:

Test Used	**Test Statistics**	**p-value**
K-S Test	0.1423673019620226	$1.6418475338732905e-60$
Shapiro-Wilk Test	0.807490	0

So actually, although from the graph, it looks like the data does follow a normal distribution, the result of both tests indicate that we have strong evidence to reject the null hypothesis that the data comes from a normal distribution. The reason behind it should be the pointy middle part and the long tail.

4.2 Fitting Other Distributions

In order to find the best-fitted distribution for the return data, I decide to loop over all the distributions that Python's *Scipy* library has, calculate the MLE, and perform the K-S Test to decide the goodness of fit. The following are the results for a few best fitted distributions[1]:

Distribution	**Test Statistics**	**p-value**
cauchy	0.0313406	0.002451
foldcauchy	0.028748	0.007097
johnsonsu	0.021563	0.083673

[1]the full result of all 87 distributions can be found on the Github repository

Among the three best-fitted distributions, Cauchy might be the only distribution that we are familiar with. The best-fitted distribution, and actually the only one that passes the K-S Test, is the Johnson S_U Distribution, which according to Wikipedia, is a model that has been used successfully to model asset returns for portfolio management [4], which is indeed what we are modeling right now. However, since this is not the focus of the project, I will not make any further investigation or discussion about the distribution of the return or the Johnson S_U distribution. I will end this section with a graph [6] of the fitted Johnson S_U distribution.

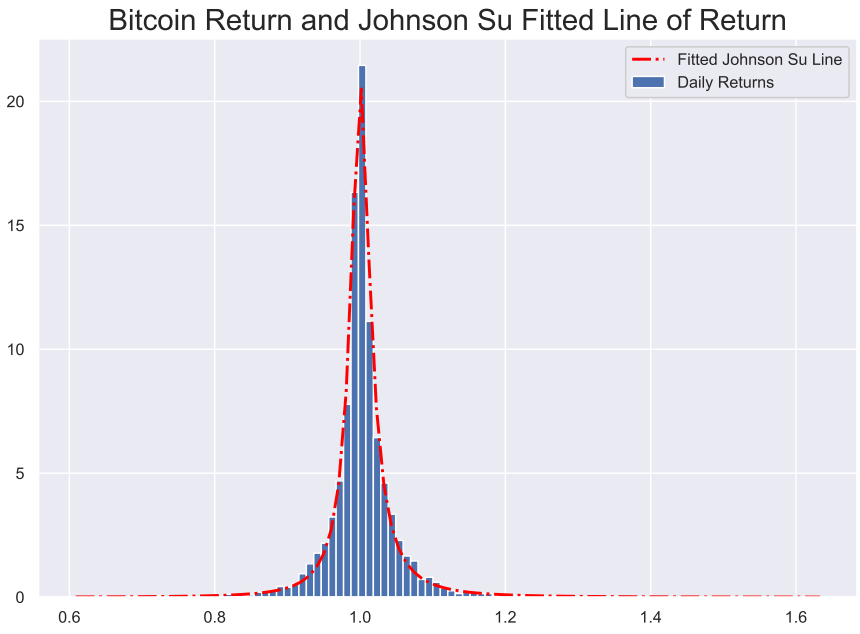

Figure 6: Fit to Johnson S_U Distribution

5 Monte-Carlo Simulation of Future Prices

In this section, I attempt to perform a Monte-Carlo Simulation to forecast the Bitcoin Price using the historical prices. In particular, I am using non-parametric Monte-Carlo Simulation. For every day that I want to forecast, I randomly draw a historical return and use it as the predicted return of that day. This process will be repeated N times, so there will be N random walks generated. The following is the algorithm:

Algorithm 1: Vanilla Monte-Carlo Price Forecast Using Historical Return

```
for i = 1 : N do
    for every day I want to estimate do
        randomly draw a return from the historical data;
    end
    store the random walk of returns as one simulation in a dataframe column;
end
```

This algorithm will return N random walks of returns. By calculating the cumulative product of each column (to get the cumulative return at each date) and multiplying each element in the column by the latest historical price, we will get a random walk of predicted future prices. Here, I will do $N = 50,000$ simulations for each experiment.

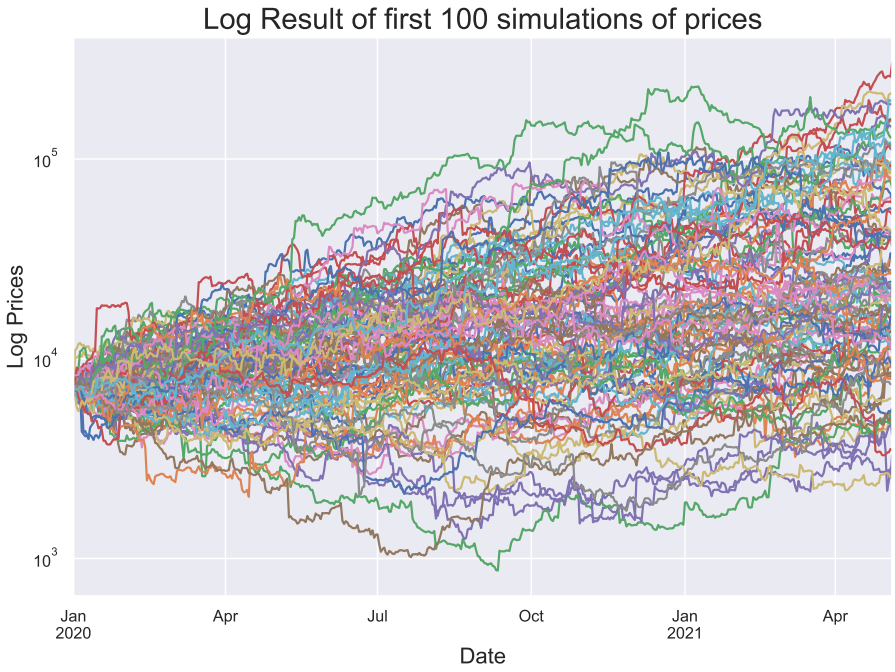

Figure 7: The Result of Monte-Carlo Simulation From 2020-01-01 to 2021-05-05

5.1 Simulate From 2020-01-01 to 2021-05-05

First, I decide to simulate the price from 2020-01-01 to 2021-05-05 and compare the result to the real price in reality. Figure [7] is a visualization of the first 100 random walks. Then, I focus on the predicted final price at 2021-05-05 and examine the distribution of the prices predicted on that day. Since the real data is discrete, it will be easier to manipulate it if it could be made continuous. Therefore, I decide to employ a common non-parametric way of estimating the pdf of a random variable: kernel density estimation (KDE). Essentially, KDE is a data smoothing problem where inferences about the population are made based on a finite data sample [5]. The kernel density estimator is defined as follows:

Definition 1. *[14] Given a kernel K and a positive number h, called the* ***bandwidth****, the* ***kernel density estimator*** *is defined to be*

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}K\left(\frac{x - X_i}{h}\right)$$

Moreover, a **kernel** is defined to be any smooth function K s.t. $K(x) \geq 0$, $\int K(x)dx = 1$, $\int xK(x)dx = 0$ and $\sigma_k^2 \equiv \int x^2K(x)dx > 0$ [14]. In this project, I am using the Gaussian kernel, which is defined as follows:

Definition 2. *The N-dimensional Gaussian kernel is*

$$G_{ND}(\vec{x};\sigma) = \frac{1}{\left(\sqrt{2\pi}\sigma\right)^N}e^{-\frac{|\vec{x}|^2}{2\sigma^2}}$$

I use the *Scipy.stats* [9] library's **Gaussian KDE** method to do the estimation. By default, the method uses the "Scott" method to estimate the bandwidth. As described in Scipy's documentation, the Scott's Rule is $bandwidth = n^{-1/(d+4)}$, where n is the number of data points and d is the number

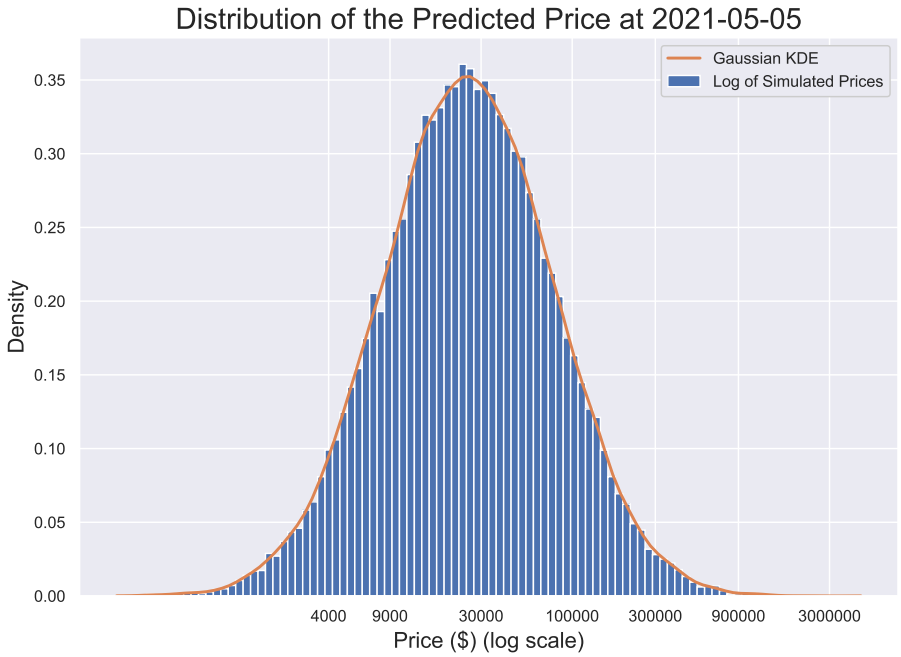

Figure 8: The Distribution of Final Prices And the KDE

of dimensions. Figure [8] shows the resulting histogram of the simulated final prices and the fitted Gaussian KDE.

With this Gaussian KDE, it is easy to get the possible price with the highest possibility, the 2.5% quantile and the 97.5% quantile. Figure [9] with the most likely predicted price, the 2.5% quantile, the 97.5% quantile, and the real price at 2021-05-05 labeled. From the graph, we can see the simulation slightly underestimates the final price, which might be because of the unusual surge in Bitcoin price at the beginning of this year. This is an example of unusual events that are hard for any model to capture.

After this, I decide to take a step forward and use the following algorithm to further analyze the most likely path.

Algorithm 2: Finding the Most Likely Path

for *every day I estimated* **do**
  fit a Gaussian KDE to the $50,000$ possible prices I estimated;
  record the most likely price, 2.5% Quantile, and the 97.5% Quantile;
end
return the 3 paths: most likely, the 2.5% quantile, and the 97.5% quantile;

With this algorithm, I can visualize the most likely path I simulated and compare it to the real path. Figure [10] is a visual demonstration of the simulated most likely path versus the real path. Again, we can see that the forecast is actually quite accurate until the end of 2020 or the beginning of 2021. However, with the two sudden surge during earlier this year, the prediction started to deviate.

Another thing to notice is that the trend in reality after the two surges has synced up again with the simulation, but just at different price level. So we can see our estimation does a decent job predicting the price movement in usual situations, but it would not be able to predict the abnormal price movements like the ones at the beginning of this year. These abnormal events in the market are extremely difficult to model and requires much more sophisticated methods.

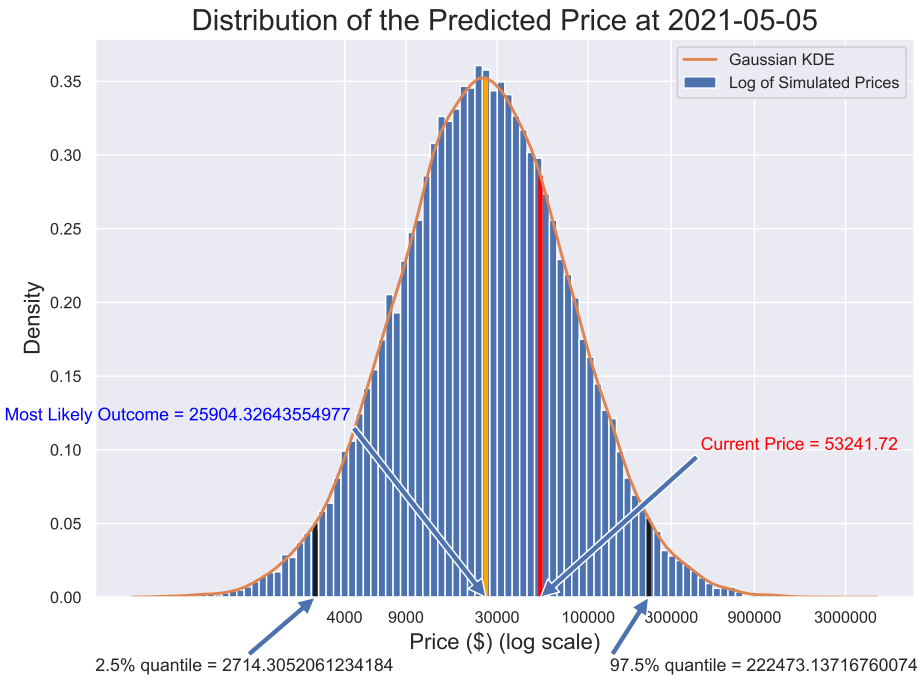

Figure 9: The Distribution of Final Prices And the KDE with labels

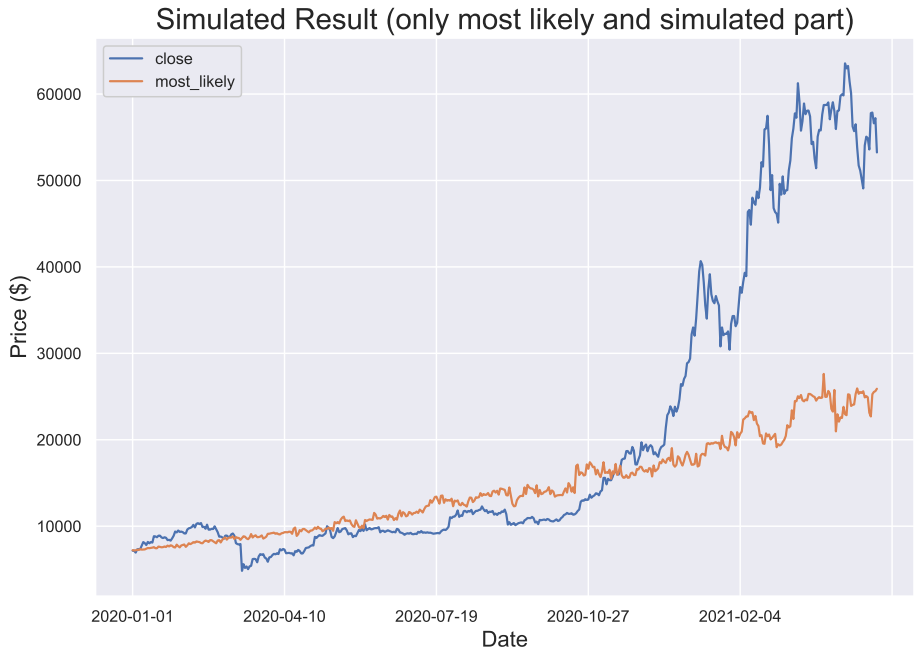

Figure 10: The Most Likely Path And the Real Path

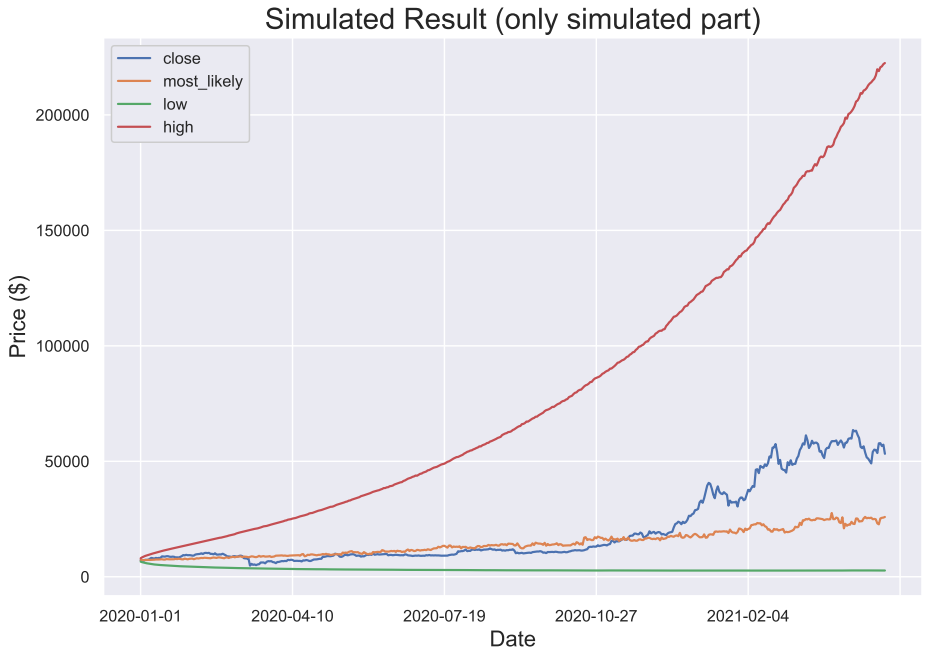

Figure 11: The Most Likely Path, the 2.5% Quantile, and the 97.5% Quantile

Finally, figure [11] is a plot including the 2.5% quantile and the 97.5% quantile. We can see that these two quantiles create quite a large interval. Therefore, the variance of our Monte-Carlo Simulation is relatively big and it will be more helpful for us to focus on the most likely path generated.

5.2 Predict Price from 2021-05-05 to 2021-12-31

Since from section section 5.1, we can conclude that this Monte-Carlo Simulation can give a decent result except in the situations of abnormal market movements such as the one we see earlier this year, it will be interesting to simulate the prices from 2021-05-05 to 2021-12-31. In that way, we can see how the model think the price at the end of this year will be. Therefore, I use all the historical data I have and perform a Monte-Carlo Simulation in the same way. Figure [12] shows the first 500 simulated random walks.

Figure [13] shows the distribution and the fitted Gaussian KDE of the possible final prices at 2021-12-31. From the result, we can see that the model predicts the most likely outcome at the end of this year will be 107198.58, which is almost two times of the current price. Therefore, the models seems to predict a continuing bullish market for Bitcoin in the rest of the year. With this KDE, it is easy to calculate that under the prediction of our simulation, the probability that the Bitcoin price at the end of the year is higher than the current price is as high as 0.80266.

However, it is important to notice again that our simulation does not perform as good in case of abnormal market movements. Moreover, there are many other factors affecting the price of cryptocurrencies. Here, I am only using a relatively simple model and only relying on the historical returns for the prediction. Therefore, this model cannot be an investor's only reference. To make an informed investment decision, investors should employ more data and build other models, as well.

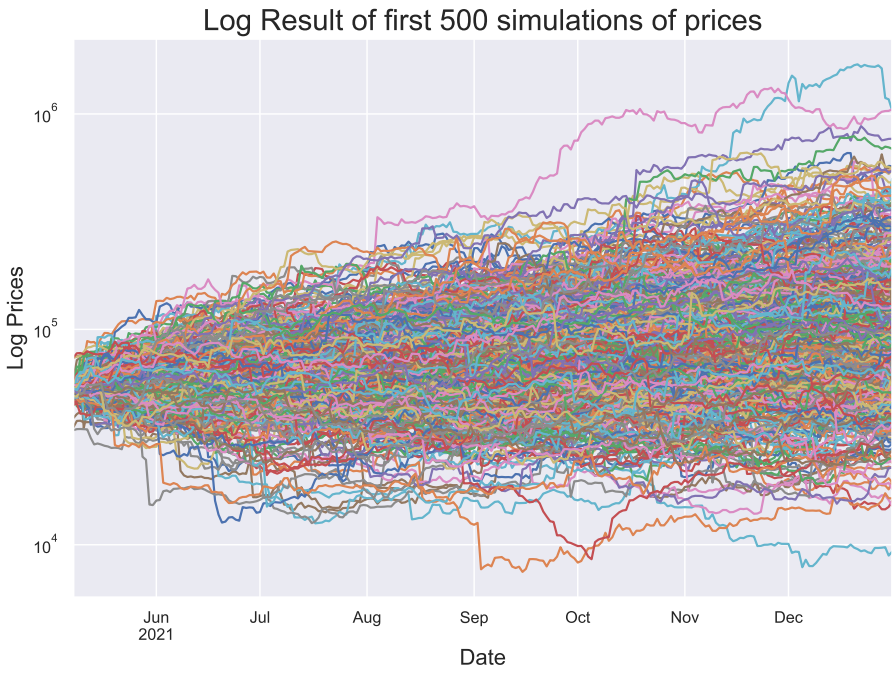

Figure 12: The Result of Monte-Carlo Simulation From 2021-05-05 to 2021-12-31

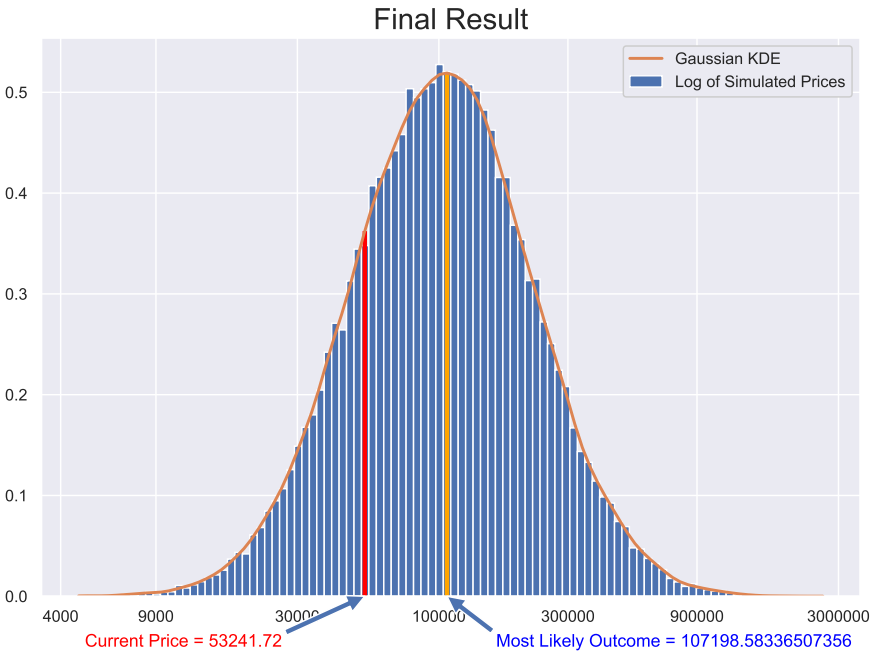

Figure 13: The Distribution of Final Prices At 2021-12-31 And the KDE with labels

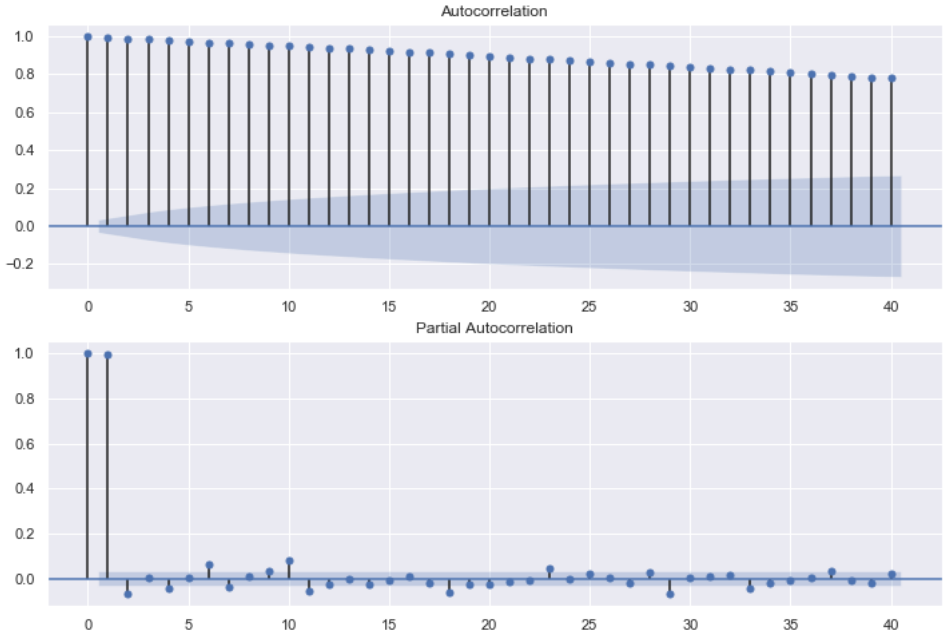

Figure 14: ACF and PACF of Daily Closing Price

6 ARIMA Forecasting

In the last section, I will fit the data to an auto-regressive integrated moving average (ARIMA) model to perform another forecast. The ARIMA model [6] is actually a combination of 3 parts: the auto-regressive part AR, the moving average part MA, and the integrated part I. So there are three parameters corresponding to the three parts: p is the order of the AR term, which is the number of lags to be used as predictors; q is the order of the MA term, which is the number of lagged forecast errors to be used; d is the order of differencing (I) term, which indicates how many times you need to difference the series to make it stationary. In Section 3, we have seen that the first order differenced closing price is stationary, which is an indication that $d = 1$ is probably a proper choice.

6.1 ACF and PACF Analysis

In this section, I will follow some proper procedures to select the three parameters. Before the procedures, it is useful to look at the auto-regressive correlation function and the partial auto-regressive correlation function of the daily closing price [14]. As we have shown in Section 3, a first order differencing is needed to make the series stationary. Therefore, we also need to look at the ACF and PACF of the differenced closing price [15].

To better understand the AR and MA part, it is useful to look at a pure AR model and a pure MA model, respectively. As mentioned earlier, an AR model describes the correlation between a time series and its own lags. Mathematically, it is the following model [6]

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + ... + \beta_p Y_{t-p} + \epsilon \tag{1}$$

where Y_{t-1} is lag 1 of the series, β_1 is the coefficient of lag 1 of the model, and α is the interception. Similarly, an MA model describe the time series correlation with the lagged forecast errors.

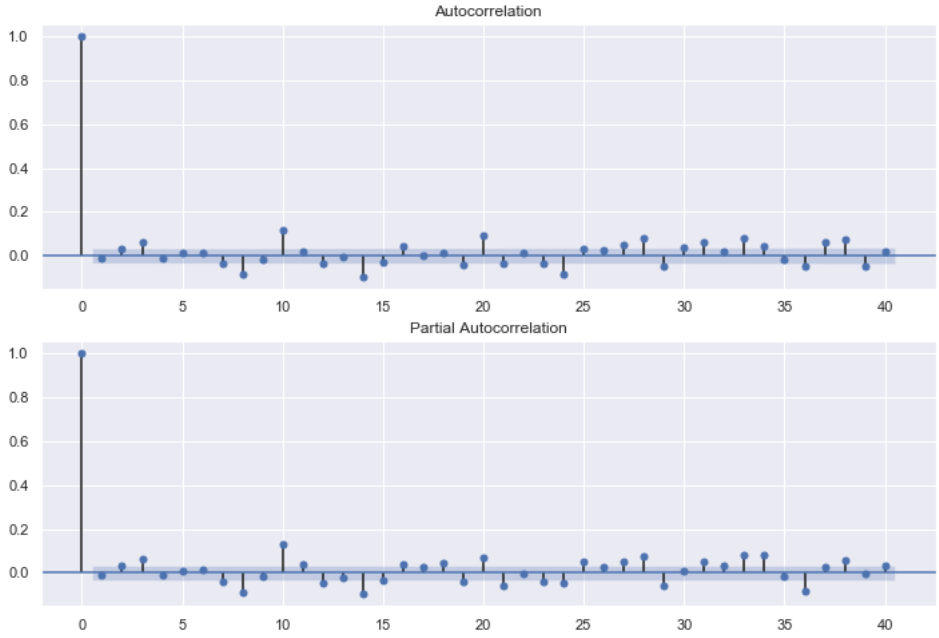

Figure 15: ACF and PACF of Differenced Daily Closing Price

Mathematically, it is the following model [6]

$$Y_t = \alpha + \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + ... + \phi_q \epsilon_{t-q} \tag{2}$$

where the error terms are the errors from the following equations:

$$Y_t = \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + ... + \beta_0 Y_0 + \epsilon_t$$
$$Y_{t-1} = \beta_1 Y_{t-2} + \beta_2 Y_{t-3} + ... + \beta_0 Y_0 + \epsilon_{t-1}$$

We will get the ARIMA model by combining the AR and the MA part, and use the appropriate order for I.

Therefore, we need to tentatively find an order for AR and MA. To find a proper order of AR, we can inspect the PACF plot. Mathematically, the formula for PACF is [6]

$$Y_t = \alpha_0 + \alpha_1 Y_{t-1} + \alpha_2 Y_{t-2} + ... \tag{3}$$

The partial auto-correlation of lag k is just the coefficient α_k in the above formula. The PACF graph simply plots the value of the coefficient at each lag. An important point to know is that any auto-correlation in a stationarized series can be rectified by adding enough AR terms. So, we can tentatively take the AR term to be equal to the number of lags that crosses the significance limit (indicated by the blue shades) in the PACF plot [6]. Therefore, by looking at figure [15], we can tentatively set the order of the AR term as $p = 1$.

To determine the MA term, we need to look at the ACF plot, which tells how many MA terms are required to remove any auto-correlation in the stationarized series [6]. Again, we look for the first number of lags that crosses the significance limit, but this time in the ACF plot. We can see that the first lag that cross the significance limit in [15] is also 1. Therefore, based on our analysis, we can temporarily decide that the proper model to use is $ARIMA(1, 1, 1)$.

6.2 Model Selection and Fitting

In this section, I will use Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) to perform a more rigorous model selection. Both of them are methods for scoring and model selection. AIC evaluates the models' fit on the data, with a penalty term for the complexity of the model [15]. The formula for AIC is

$$AIC = -2\ln(L) + 2k \tag{4}$$

where L is the likelihood of the model and k is the number of parameters in the model. The lower the AIC score is, the better the model is.

Similarly, BIC is defined as

$$BIC = -2\ln(L) + \ln(N) * k \tag{5}$$

where L is the likelihood, N is the number of examples in the training dataset, and k is the number of parameters. Similarly, the lower the BIC score is, the better the model is.

To figure out the best ARIMA model, I enumerate all the possible combinations of $p, d, q \leq 2$, fit an ARIMA to the data with these parameters, and calculate both AIC and BIC. Then, I will choose the model with the lowest AIC and BIC. Similar to the previous part, I take the data before 2020-01-01 as the training set. Here is part of the result with a few best models [2]:

Model	AIC	BIC
(2, 1, 1)	−9221.417977	−9191.519571
(2, 1, 2)	−9219.465671	−9183.587584
(1, 1, 0)	−9213.368619	−9195.429575
(2, 1, 0)	−9212.944966	−9189.026241
(1, 1, 1)	−9212.870525	−9188.951800

Therefore, although the difference is not that large, the best model is actually $ARIMA(2, 1, 1)$. Although the BIC for $ARIMA(1, 1, 0)$ is slightly higher than the BIC for $ARIMA(2, 1, 1)$, the difference between their AIC is larger. Therefore, I will use $ARIMA(2, 1, 1)$. The following is the summary of the fitted result.

ARIMA Model Results

Dep. Variable:	D.close	No. Observations:	2921
Model:	ARIMA(2, 1, 1)	Log Likelihood	4615.709
Method:	css-mle	S.D. of innovations	0.050
Date:	Sun, 09 May 2021	AIC	-9221.418
Time:	00:14:47	BIC	-9191.520
Sample:	01-02-2012 - 12-31-2019	HQIC	-9210.649

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0025	0.001	2.039	0.041	9.59e-05	0.005
ar.L1.D.close	0.8050	0.029	28.097	0.000	0.749	0.861
ar.L2.D.close	0.1588	0.018	8.676	0.000	0.123	0.195
ma.L1.D.close	-0.9524	0.023	-41.743	0.000	-0.997	-0.908

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	1.0321	+0.0000j	1.0321	0.0000
AR.2	-6.1010	+0.0000j	6.1010	0.5000
MA.1	1.0500	+0.0000j	1.0500	0.0000

Figure 16: The Summary of ARIMA(2, 1, 1)

[2]The full result is stored in a CSV on the Github repo

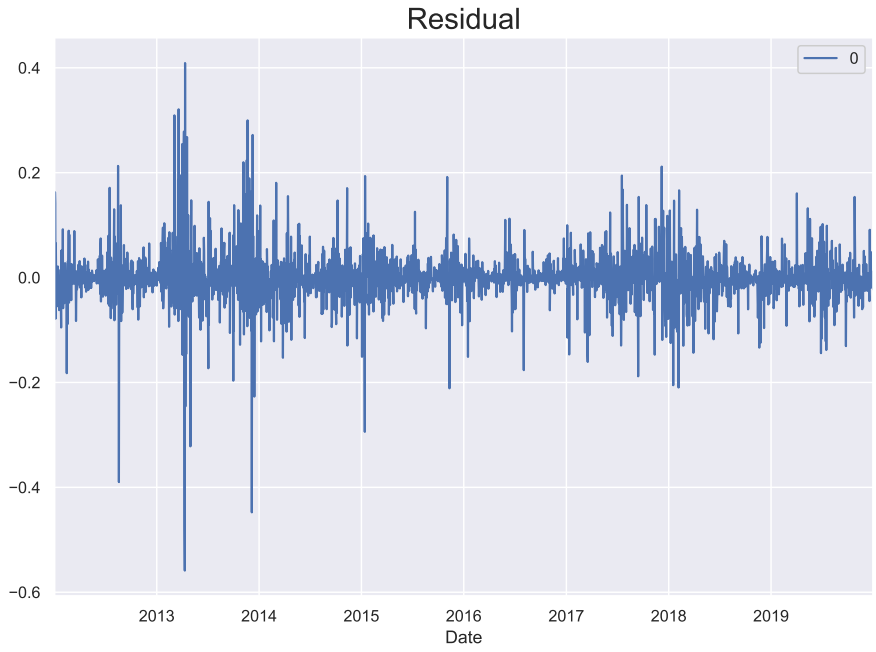

Figure 17: The Residual of ARIMA(2, 1, 1)

Moreover, figure [17] is the residual of the model. From the summary, we can see the all of the parameters are statistically significance. Moreover, from the residual plot, we can see the residuals are roughly symmetrically distributed about the x-axis. Both of these are evidence that the model is good and it is safe to perform a prediction using this model. Finally, based on the result, the expression of the fitted $ARIMA(2, 1, 1)$ is given by

$$Y_t = 0.0025 + 0.8050Y_{t-1} + 0.1588Y_{t-2} - 0.9524\epsilon_{t-1} + \epsilon_t \tag{6}$$

6.3 Forecasting With the Model

In this part, I attempt simulate the price movement from 2020-01-01 to 2021-05-05, using the ARIMA model. Figure [18] is the forecasted path of the price movement. Similar to the vanilla Monte-Carlo Simulation in section 5, our ARIMA model does quite well in forecasting the price before the abnormal surge started at the end of 2020, but it is unable to predict the sudden surges in price. Figure [19] shows the prediction up to 2020-12-31, which is approximately the beginning of the price surge. We can already see the gap is closer compared to the difference we see at 2021-05-05, which is after the two surges in price earlier this year.

6.4 A Rolling Forecast

In this last part, I decide to perform a simulation of a rolling forecast of the price from 2020-01-01 to 2021-05-05. That is, for everyday between 2020-01-01 and 2021-05-05, I use the historical data up until the previous date and fit an ARIMA model, and use this model to predict the price at the day. Below is the algorithm:

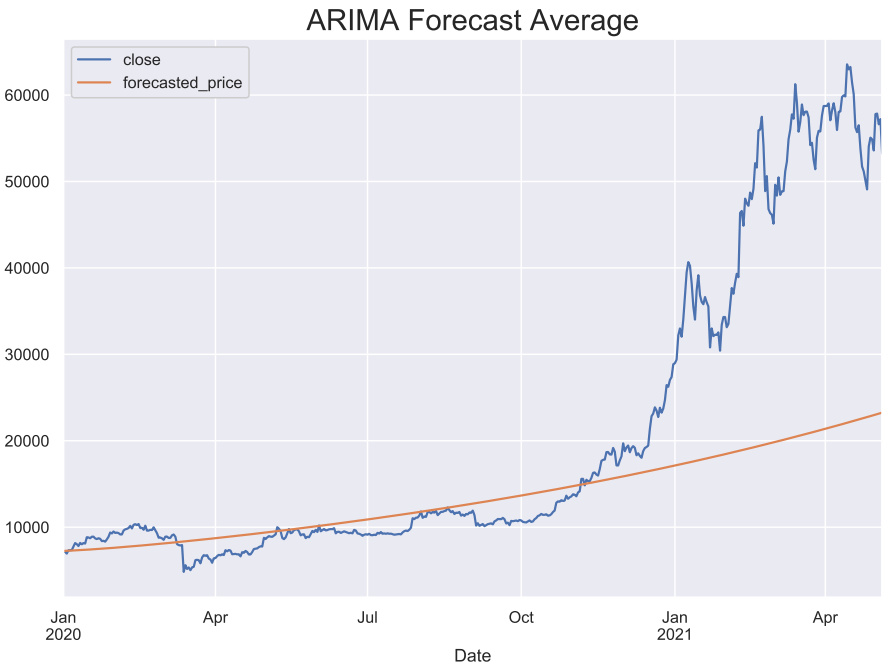

Figure 18: ARIMA Forecast from 2020-01-01 to 2021-05-05

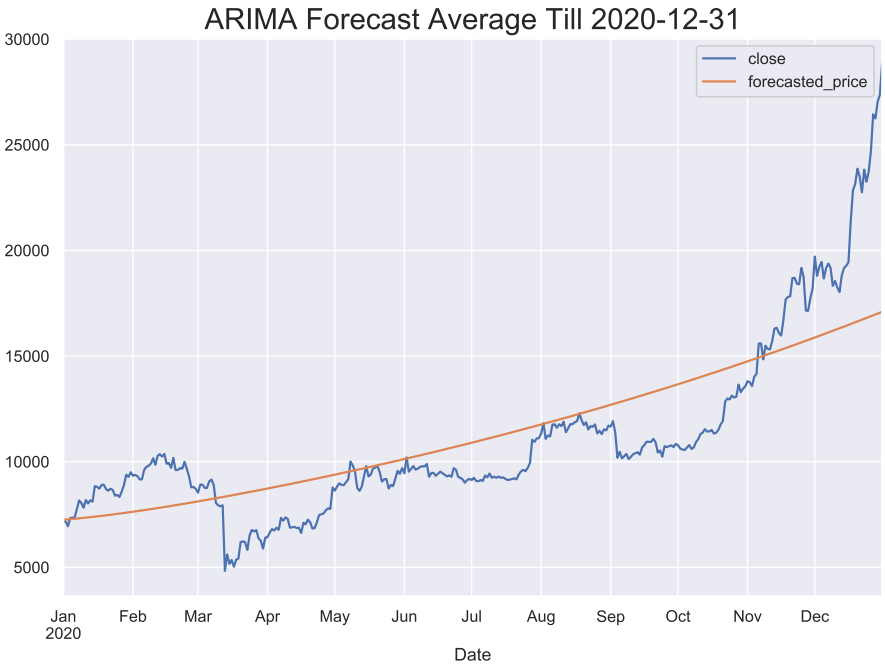

Figure 19: ARIMA Forecast from 2020-01-01 to 2021-05-05

Figure 20: A Rolling Forecast From 2020-01-01 to 2021-05-05

Algorithm 3: A Rolling Forecast With ARIMA

for *every day from 2020-01-01 to 2021-05-05* **do**
    training set = all the data till the previous day;
    fit an $ARIMA(2, 1, 1)$ to the data;
    use the model to predict the price for today;
end
return the predicted path of the price;

So essentially, the difference is, in our previous simulation, we use only the data until 2019-12-31 to predict the prices for all the days until 2021-05-05, but now, we are constantly appending the real price to our training set, fitting a new in every iteration, and predicting the price with the new model fitted in that iteration.

Figure [20] plots the predicted price movement with the rolling forecast algorithm along with the real price path. We can see this simulation produces a very accurate result. This makes sense because we are constantly updating our model with a new value from the real historical price. Therefore, for each iteration, we are only using the fitted model to predict the price in one period ahead.

However, as good as the result looks, it is actually not pragmatic in practice. The reason is that if we are predicting the price for a period of time in the future, there will be no new data available for you to make a new model in every iteration. For instance, on 2021-05-05, if we are using all the data we have to forecast the price until 2021-12-31, that is all the data we have. We are able to have an accurate prediction for 2021-05-06, but starting from the prediction for 2021-05-07, we will not have the real price data to make a new model anymore. That is, for the price prediction for 2021-05-07, we will not have the real price data on 2021-05-06. Therefore, it is not a practical algorithm to make a price forecast in reality.

7 Conclusion

In this project, I showed the return and the first order differenced closing price of Bitcoin is stationary. Moreover, I showed the daily return follows a Johnson S_U distribution. Then, I performed a Monte-Carlo Simulation as well as an ARIMA forecasting to predict the Bitcoin daily closing price. Both of these forecasting method performs well in usual situation, but unable to predict accurately during abnormal price surges. The rolling forecast with ARIMA produces an incredible accuracy, but it is unfortunately not practical in reality.

In the future, a possible direction for research is developing potentially complimentary models that could partially make up the shortcoming of the inability to predict abnormal price movements. A possible approach is analyzing the social media sentiments on Bitcoin or cryptocurrencies as a whole, because as a category of financial instruments with a large amount of retail investors, sentiment could play a large role in the price.

References

[1] Nate DiCamillo. *Morgan Stanley Confirms Wealth Management Clients Have Access to 2 Crypto Funds*. Apr. 2021. URL: https://www.coindesk.com/morgan-stanley-confirms-wealth-management-clients-have-access-to-2-crypto-funds.

[2] M. D. Scheuerell E. E. Holmes. *Applied Time Series Analysis for Fisheries and Environmental Sciences*. Mar. 2021. URL: https://nwfsc-timeseries.github.io/atsa-labs/sec-boxjenkins-aug-dickey-fuller.html.

[3] Fjd et al. *FJD*. URL: https://www.real-statistics.com/time-series-analysis/stochastic-processes/dickey-fuller-test/.

[4] *Johnson's SU-distribution*. Mar. 2021. URL: https://en.wikipedia.org/wiki/Johnson's_SU-distribution.

[5] *Kernel density estimation*. Mar. 2021. URL: https://en.wikipedia.org/wiki/Kernel_density_estimation.

[6] Selva Prabhakaran. *ARIMA Model - Complete Guide to Time Series Forecasting in Python*. Mar. 2021. URL: https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/.

[7] Selva Prabhakaran. *Augmented Dickey-Fuller (ADF) Test - Must Read Guide*. Mar. 2021. URL: https://www.machinelearningplus.com/time-series/augmented-dickey-fuller-test/.

[8] *Quandl*. URL: https://www.quandl.com/data/BCHAIN/MKPRU-Bitcoin-Market-Price-USD.

[9] *scipy.stats.gaussian_kde*. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.gaussian_kde.html#id5.

[10] *scipy.stats.rv_continuous.fit¶*. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.rv_continuous.fit.html.

[11] Stephanie. *Kolmogorov-Smirnov Goodness of Fit Test*. Dec. 2020. URL: https://www.statisticshowto.com/kolmogorov-smirnov-test/.

[12] Stephanie. *Shapiro-Wilk Test: What it is and How to Run it*. Jan. 2021. URL: https://www.statisticshowto.com/shapiro-wilk-test/.

[13] Stephanie. *Stationarity Differencing: Definition, Examples, Types*. Apr. 2021. URL: https://www.statisticshowto.com/stationarity/.

[14] Larry Wasserman. *All of statistics a concise course in statistical inference*. Springer, 2004.

[15] Alexandre Zajic. *Introduction to AIC-Akaike Information Criterion*. Dec. 2019. URL: https://towardsdatascience.com/introduction-to-aic-akaike-information-criterion-9c9ba1c96ced.