Ethereum: Price Prediction Analysis Final Report

## By: Nicholas Roller

# Problem Statement

You work for an in-house trading desk that makes investments in cryptocurrencies. One of the larger portfolios is in Ethereum, the second largest crypto by market cap. The firm is wondering if it should increase or decrease its position size with a one month outlook in mind. You are tasked with determining which direction Ethereum will be one year from now, and to what magnitude.

# Dataset – Cleaning & Wrangling

<https://www.kaggle.com/varpit94/ethereum-data>

Our dataset consisted of 2244 rows and 7 columns for the daily Ethereum price ranging from 2015 to 2021. We found only 4 rows consisting of missing data and handled these using a simple forward fill imputation. We dropped the adjusted close column since this is only applicable for traditional securities that trade during specific hours, whereas cryptocurrencies trade around the clock and 7 days a week. We decided to create a singular “Price” feature to use in our models which was the average mean of the open and close prices for each day.

# Exploratory Data Analysis

We first wanted to look at any seasonal trends or outliers, so we plotted boxplots aggregated by month for our entire dataset.

Chart, box and whisker chart

Description automatically generated

We saw an interesting trend where during the summer months, June and July, price fluctuations were confined to a much smaller range, inter-quartile range, but did tend to have outliers. Where as the closer it was to the end of the year, prices could see a lot more volatility. This indicates that we could expect to see a fairly consistent seasonality throughout the year.

We then wanted to get a wholistic view of our price data, so we plotted our timeseries.

Chart, line chart

Description automatically generated

Immediately we noticed that since prices varied on such a massive scale, it may be better to view this plot on a logarithmic scale.

Chart, line chart

Description automatically generated

To get an idea of the smoother trends within our timeseries, we plotted the log price on top of a weekly and monthly resample from Jan 2020 to present:

Chart

Description automatically generated

The weekly resampled plot looked ideal as it still exhibited more detailed trends without as much of the noise. Our goal is to predict a price forecast one year in the future, after all, so we won’t be too worried about the more fine-grained movements.

We wanted to see how volume played in, so we plotted the weekly price aggregates in parallel with volume:

Chart, line chart

Description automatically generated

We noticed a fairly exponential increased in volume as well as price.

Finally, we wanted to see how exponential smoothing affected our trend, visually, before finally moving onto preprocessing:

Chart, line chart

Description automatically generated

Looking at the lag plots for Price and LogPrice it is apparent that both have a strong correlation with their t-1 lags.

Chart, scatter chart

Description automatically generated

LogPrice Lag plot:Chart, line chart

Description automatically generated

# Preprocessing

The preprocessing stage was fairly straightforward as we were analyzing a univariate time series. We reduced our data frame down into our key components, the date and price column. We converted our date column to a datetime index and set it as the index to our data frame. We create our final dependent variable data frame by taking the weekly aggregate mean of our Price column through the df.resample() method. Finally, established our train and test splits of our dataset using the following code:

split=int(len(ETH['Price']) \* 0.8)

y\_train, y\_test = ETH['Price'][0:split], ETH['Price'][split:len(ETH['Price'])]

# Final Model & Recommendations

The first key step before we actually model our time series is to establish stationarity. We can do this through a number of methods, such as taking the difference of the time series or taking it’s log, or a combination. We wrote a function to take an input timeseries and plot the timeseries and it’s rolling mean and standard deviations, so we can visually see if it is stationary. In addition, it calculates and outputs the results of the Dickey-Fuller test. Below is the output for the timeseries without any modifications:

Chart, histogram

Description automatically generated

Our p-value is above 0.05 and our test statistic is not smaller than our critical value, therefore we cannot reject the null-hypothesis of non-stationarity.

We then take the difference of y and test again:

Graphical user interface, chart, histogram

Description automatically generated

Again, the same, so we continue modifying our timeseries until we achieve stationarity. This time we simply take the log of y and test again:

Chart, histogram

Description automatically generated

Still not quite stationary yet. Finally, we try to combine these two methods taking the log transform of y and its difference:

Graphical user interface

Description automatically generated

We have finally achieved a p-value below 0.05 and a test statistic that is more negative than our critical value. We can also see visually that our timeseries does not have a trend and that the rolling mean and standard deviations remain constant from our plot.

To get an idea of the three primary components, trend, seasonality and residual, of our time series we plotted the decomposition of our y\_log series:

A picture containing histogram

Description automatically generated

We then tested the stationarity of our residuals to determine the Q coefficient for our ARIMA model and determined Q to be 1:

Graphical user interface, chart, histogram

Description automatically generated

To find our P and R coefficients we plotted the ACF and PACF plots for our y\_log\_diff series:

Chart, line chart

Description automatically generated

From this determined 4 and 3 were likely going to be our best P and R values, respectively. Plugging these into our ARIMA model and adjusting them slightly we found the lowest RSS (residual sum of squares) to be 5.6154 with a ARIMA(4,1,3) model.

Chart

Description automatically generated

With our fitted model we used the plot\_predict() function to forecast 52 steps into the future, or 1 year.

Chart, line chart

Description automatically generated

Analyzing our in-sample forecast metrics we found this model to have a mean square error (MSE) of 0.663 and mean absolute error (MAE) of 0.601. The final price predicted one year into the future was $9,070.15, or a 195% increase from the current price.

Our second model was the FB Prophet model. For this we simply restructured our data frame to have a ‘ds’ column containing our price data and a ‘y’ column with our dates. We fit the model and make a future model, for our forecast, with 52 (weekly) periods to forecast 1 year out-of-sample. Calling the plot\_components() on our forecast plots our trend and yearly seasonality:

Chart, line chart

Description automatically generated

We then plot our forecast to see both the in-sample and out-of-sample forecasts:

Chart, line chart

Description automatically generated

Our final FB Prophet model gives us an MSE of 0.149 and an MAE of 0.296 with a 1 year forecasted price of $27,856, or 806.1% increase from the current Ethereum price.

# Future Thoughts & Questions

Provided both of our models predicted an increase in the price of Ethereum one year from now, our suggestion would be that our trading firm increases their position in ETH with a one-year forecast.

As far as our model goes, we would suggest using the FB Prophet model for future forecasts. Although perhaps less interpretable than our ARIMA model, our initial testing shows it provides predictions with lower error metrics. That being said, both models could be improved given time and resources for further hyperparameter tuning. Additionally, Ethereum has historically been fairly correlated with the price of Bitcoin so a multivariate model could have the potential to produce a far stronger, albeit more complex, model.