Trend Analysis of Bitcoin Value

By Mark Parayil

Version 1.0 – December 2017

Table of Contents

Introduction:

Cryptocurrencies have gone up in popularity over the past several years due to the rise in blockchain technology and computing power across processors. They have become synonymous with DApps (Distributed Applications). The premise is that these currencies allow transactions to occur between 3rd parties without any middle man such as bank or governmental institution. For example, bitcoin takes banks and financial institutions out of the picture by allowing transactions to occur solely between two private parties with transactions being recorded on an electronic ledger called the blockchain. The same applies for Ethereum which allows smart contracts to be enforced without any institution or middle man enforcing the smart contract. There endless applications for cryptocurrencies like Ethereum such Voting system designs, managing healthcare data, and facilitates peer-to-peer contracts and applications via its own currency vehicle.

Purpose of Technical Report

The purpose of this report to walkthrough the data science process flow from a Time Series Modeling perspective using cryptocurrency price datasets.

- The Problem to solve
- Acquisition of Dataset
- Data Preprocessing & Feature Extraction
- Model Building
- Model Evaluation
- Understanding Time Series Modeling Process
- Next Steps to further tune process

Problem Overview & Approach:

The goal of this project is to use the historical pricing data of Bitcoin and other currencies to predict future prices using Time Series Modeling techniques. The data spans back from April 2013 – November 2017 for bitcoin. Ethereum was also looked at and it dated back to 2015 to 2017.

The purpose is to forecast bitcoin prices several months to a year out. I will begin by visualizing and analyzing the trends of the dataset by looking at the auto correlation between lags using two different types of correlation plots. Through these plots, the order of AR and MA values will be determined. Once those hyperparameters are identified, I will forecast the model several periods out in days to see how accurate the forecast is and if it matches up with the already existing data on a plot.

The focus of the data will be bitcoin “Close” price over daily periods. “Close” price under the bitcoin is determined to be the final price at the end of the 24-hour day for that period. For example, the close price for bitcoin on 12/08/2017 would be the price at exactly 11:59 pm UTC on that given day (or any day). There were other features that could be considered such as Open, High, Low, Volume, and Market Cap. For the purposes of this iteration in the process, I focused on the “Close” price from 04/28/2013 – 11/03/2017.

Data Acquisition & Loading:

The dataset was pulled from Kaggle.com that contained in CSV tables for pricing data on a variety of cryptocurrencies consisting of Bitcoin, Ethereum, Litecoin, Monero, and those are only a few to begin with. For the purpose of this project, I focused on Bitcoin price table with only the Date and Close columns for the analysis. The features consisted of the following:

```
In [116]:
1 btc_price.tail()
2 # 2013-04-28
3 # 2017-11-03
```

Out[116]:

Date	Open	High	Low	Close	Volume	Market Cap
2017-11-03	7087.53	7461.29	7002.94	7207.76	3369860000	118,084,000,000
2017-11-04	7164.48	7492.86	7031.28	7379.95	2483800000	119,376,000,000
2017-11-05	7404.52	7617.48	7333.19	7407.41	2380410000	123,388,000,000
2017-11-06	7403.22	7445.77	7007.31	7022.76	3111900000	123,379,000,000
2017-11-07	7023.10	7253.32	7023.10	7144.38	2326340000	117,056,000,000

```
In [117]:
1 btc_price.head()
```

Out[117]:

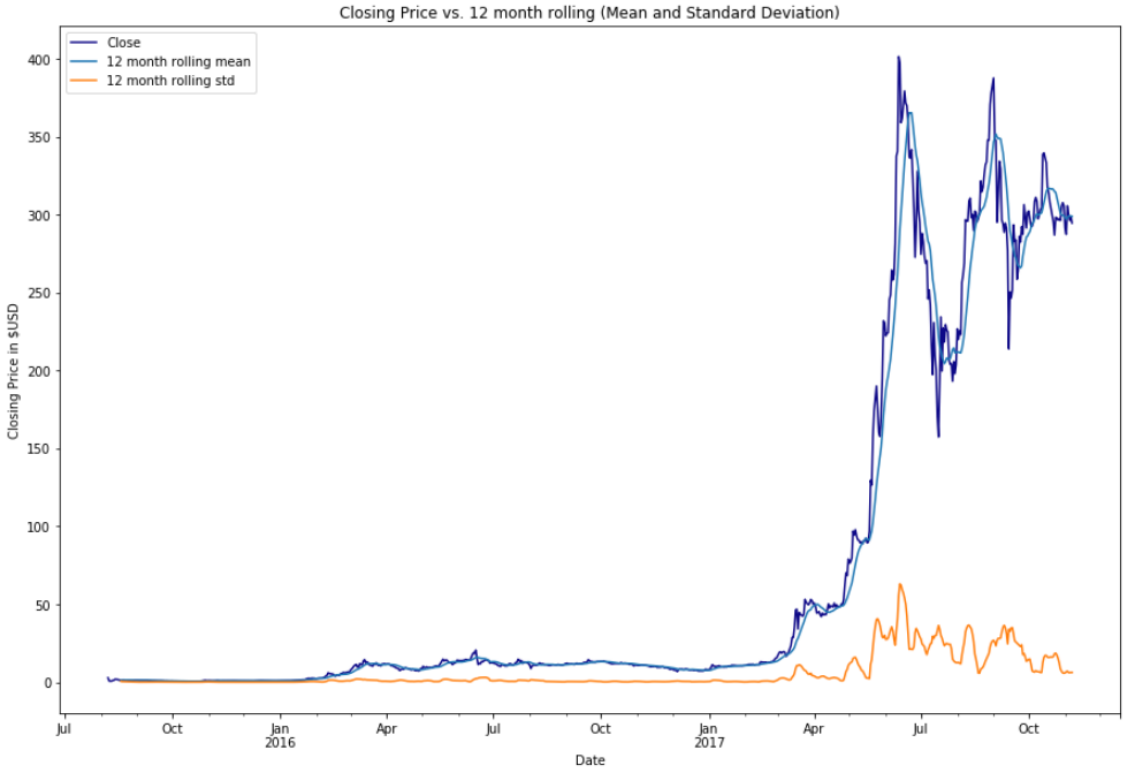
Date	Open	High	Low	Close	Volume	Market Cap
2013-04-28	135.30	135.98	132.10	134.21	-	1,500,520,000
2013-04-29	134.44	147.49	134.00	144.54	-	1,491,160,000
2013-04-30	144.00	146.93	134.05	139.00	-	1,597,780,000
2013-05-01	139.00	139.89	107.72	116.99	-	1,542,820,000
2013-05-02	116.38	125.60	92.28	105.21	-	1,292,190,000

Data Preprocessing & EDA:

1. Ethereum Analysis & EDA:

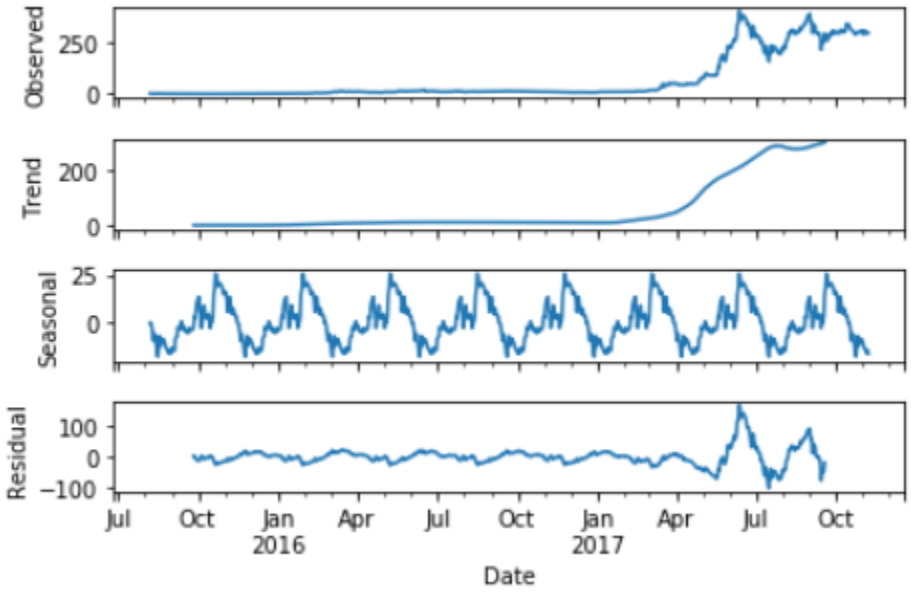
To start EDA, I wanted to see what Ethereum's Closing price looked like over time to see if there was good stationarity or trends to build a model. To start analyzing the data, I began with formatting to be ready to for visualization and modeling. First my converted the date column to datetime object through pandas, then sorted the values of the Ethereum closing price by earliest date to most recent. Next, I set the index of the of the DataFrame to be the datetime object so that index can be sorted by the date. Lastly, I removed null values from the dataset to ensure that it was ready for modeling purposes.

Let's visualize at exactly what the distribution of the close prices looks over time compared to the 12-month rolling and 12 month rolling standard deviation.



We can see that standard deviation tends to increase dramatically after the March of 2017 so this gives an idea that the close price of Ether tends to become volatile with sharp spikes in June and July of 2017.

To check for seasonality, we use seasonal_decompose and plot to see trends, seasonality, and residuals.

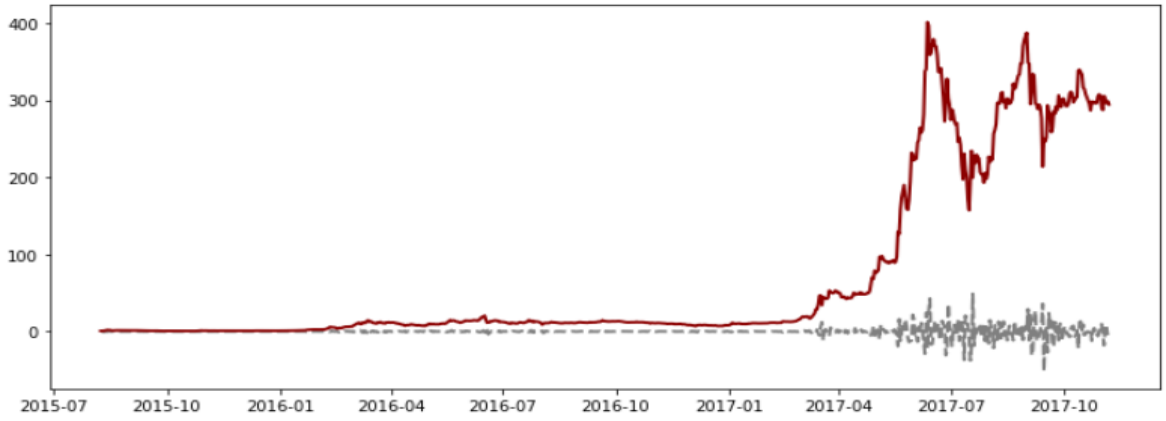


We can see there is some seasonality but to be sure let's use the Dicky-fuller test that will return the p-value. After running the function, we see a p-value of 0.68 which means the data is non-stationary. To create stationarity, we need to do a transformation on the closing price called differencing which helps remove trend or cycles. The idea is that if original data series does not have constant properties over time, then the change from one period to another might. It's calculated by subtracting one period's values from the previous period's values. After differencing once, we run the Dicky-Fuller test and return a p-value of 0.0002.

Once the data is made stationary we want to run a PACF and ACF correlation plot and determine to do an order of [2,1,5] for the p,d,q hyper parameters. We chose 2 for AR terms and 0 for MA because there is a sharp drop off on the first lag for the PACF and ACF plots.

Basic Modeling for Ethereum:

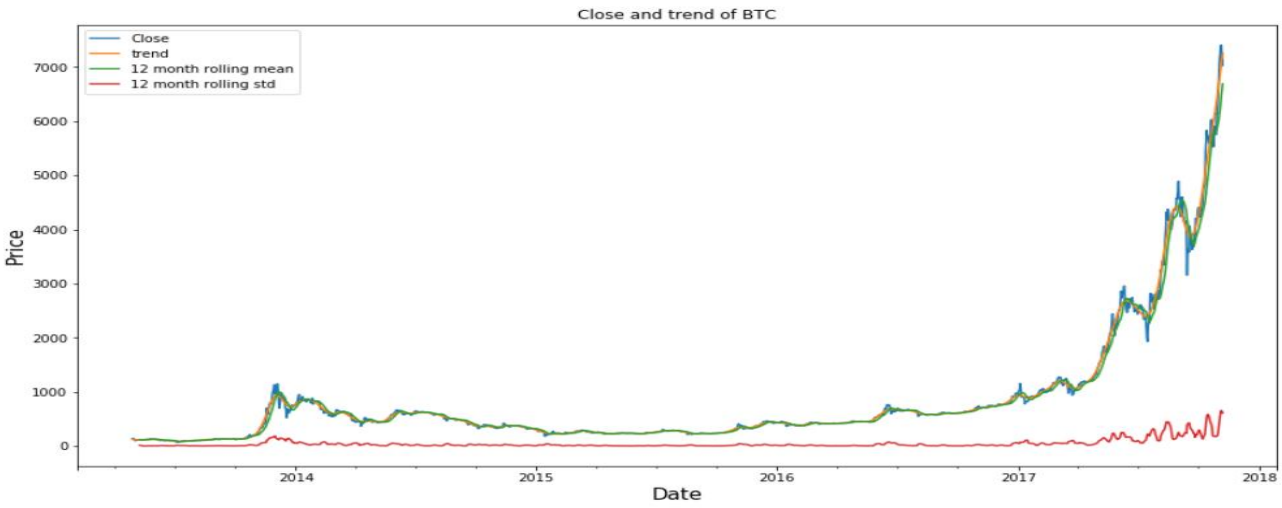
I ran an ARIMA model on the first difference values with an order of (2,1,0) after plotting the ARIMA fitted values and the first difference close prices of ether it looked like that difference values matched up pretty nicely. However, when I tried to predict the model the ARIMA model didn't fit well with the existing data:



My R2 score is -0.66 and that was the best score I could came with just from tuning the order of (p,d,q). If I were to dig deeper and have more time I would dig deeper with the other features in the Ethereum price dataset as well as try other time series models with these 'Close' prices in the future.

Bitcoin Close pricing cleaning & EDA:

To start off with the BTC close price data, I first converted the date column to the index as a datetime object and sorted the values by oldest to most recent. After doing a quick check and

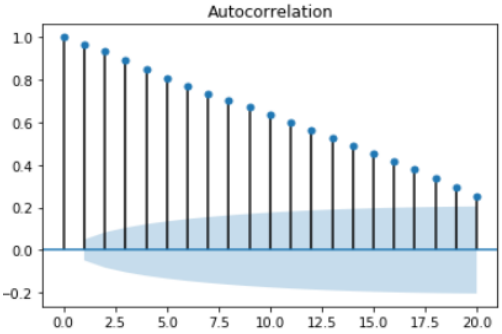


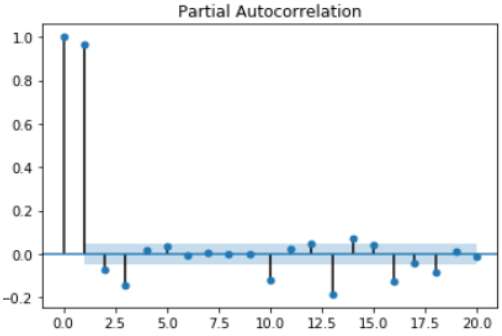
inspection there are no null values, I plotted the trend, 12 month rolling mean, and 12 month rolling standard deviation values across a plot to see how the BTC Close prices were distributed over the past several years.

Viewing the seasonal decomposition, it looked like the “Close” data had seasonality but after running the Dicky-Fuller test, I needed to difference the data once. I also decided to difference the data for a monthly period with a shift of 30 periods to create monthly seasonality. After testing the stationary, I removed the null values from the differenced series.

For correlation plotting, I plotted the close price itself, the first difference, and the seasonal difference to determine the order for p,d,q. There is a sharp drop-ff in the PACF plot of differenced series so I will need to consider adding an AR term to the model to explain the auto-correlation rather than Moving Average terms

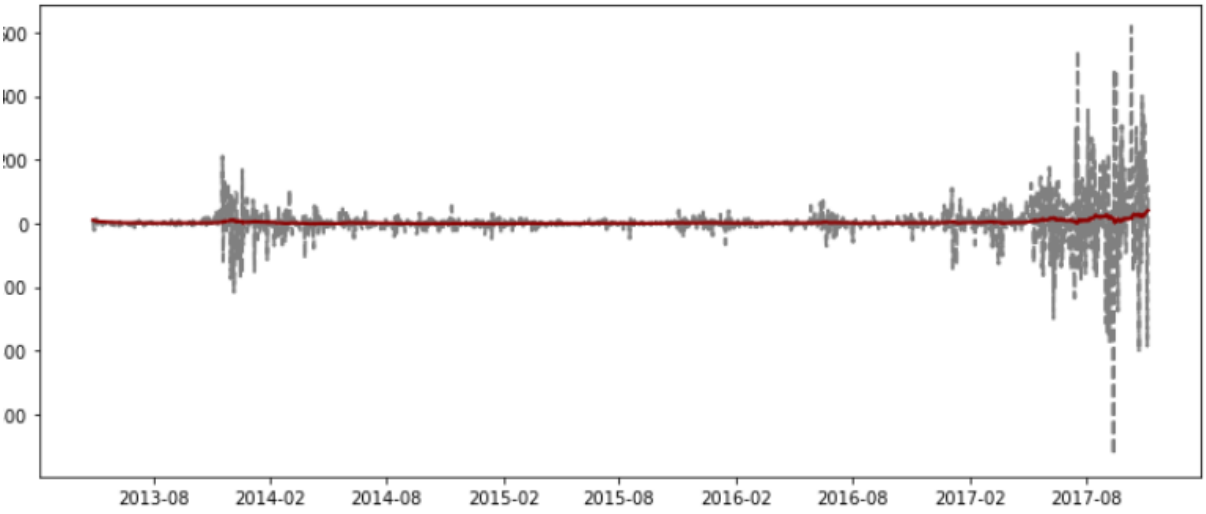
```
# Plotting for Seasonal Difference
plot_acf(btc_price_close['Seasonal Diff'].dropna(), lags=20);
plot_pacf(btc_price_close['Seasonal Diff'].dropna(), lags=20);
# since there is a sharp drop off after the first lag so we will be using an Autoregressive K number model.
# If there was a gradual decline with our partial autocorrelation plot then we would use a moving average model.
```





Model Development & Evaluation:

Initially I tried an ARMA model with a differencing order of (2,1,0). 2 being number of lags for AR features, 1 being the difference order, and 0 being the number of MA terms. Considering there is a sharp drop off in the PACF plot for the first difference values. When plotting the ar1 fitted values against the close price for the first differenced series we see the following plot as a well as a low r2 score:



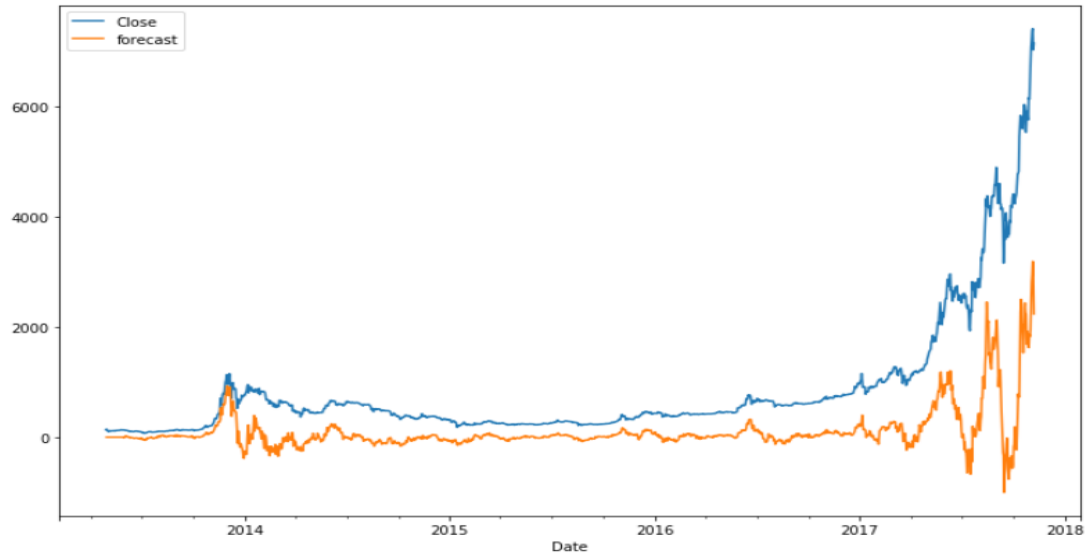

```
from sklearn.metrics import r2_score

r2_score(btc_price_close['First Diff'].dropna(), ar1.fittedvalues)
# really bad
```

Next create the out of sample predictions from the ar1 fitted model and after plotting the model we see that the predictions are not accurate and show 0 to correlation to the initial values. Next for the sake of testing, I will try a Moving Average model, however based on the PACF and ACF plots from earlier, the MA model performs worse the AR model with a R2 score 0.00006.

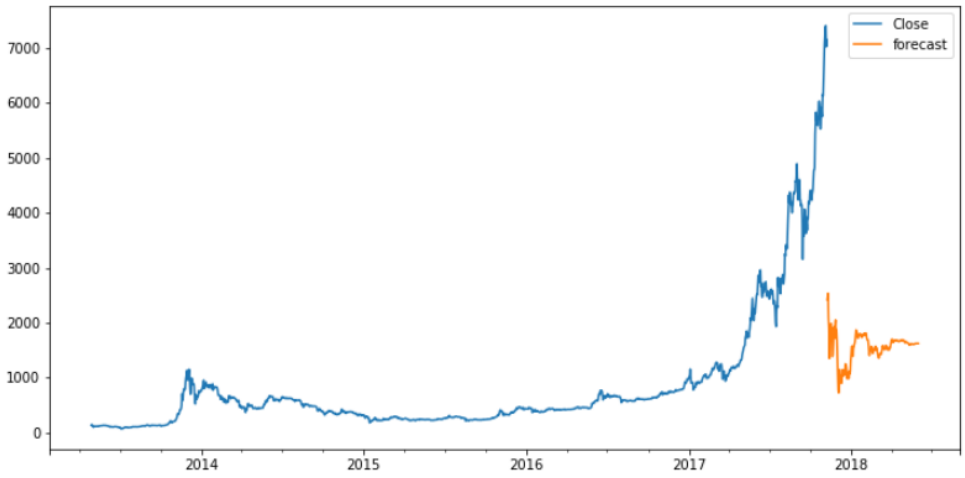
Next, I will try an ARIMA model with an order of (3,1,1) of the first difference values, however even after fitting that I was still getting a very negative correlation of -0.99. I decided to add consider seasonality to my model and use a SARIMAX model with seasonal difference series with an order of (2,1,1). This model performs much better and actually follows the original "close" price series.

```
btc_price_close['forecast'] = smodel.predict(start=1, end=1655)
btc_price_close[['Close', 'forecast']].plot(figsize=(12,8))
# started at 1, testing at 1495 to acccomodate for recent volatility
# now testing at 1344 for beginning of January 2017
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20e80691438>
```



Now I want to make predictions so I add 360 empty observations to my pandas DataFrame and concat it to my original DataFrame with the bitcoin closing price values. Then I create a forecast column with forecasted values of the SARIMAX model and plot it.

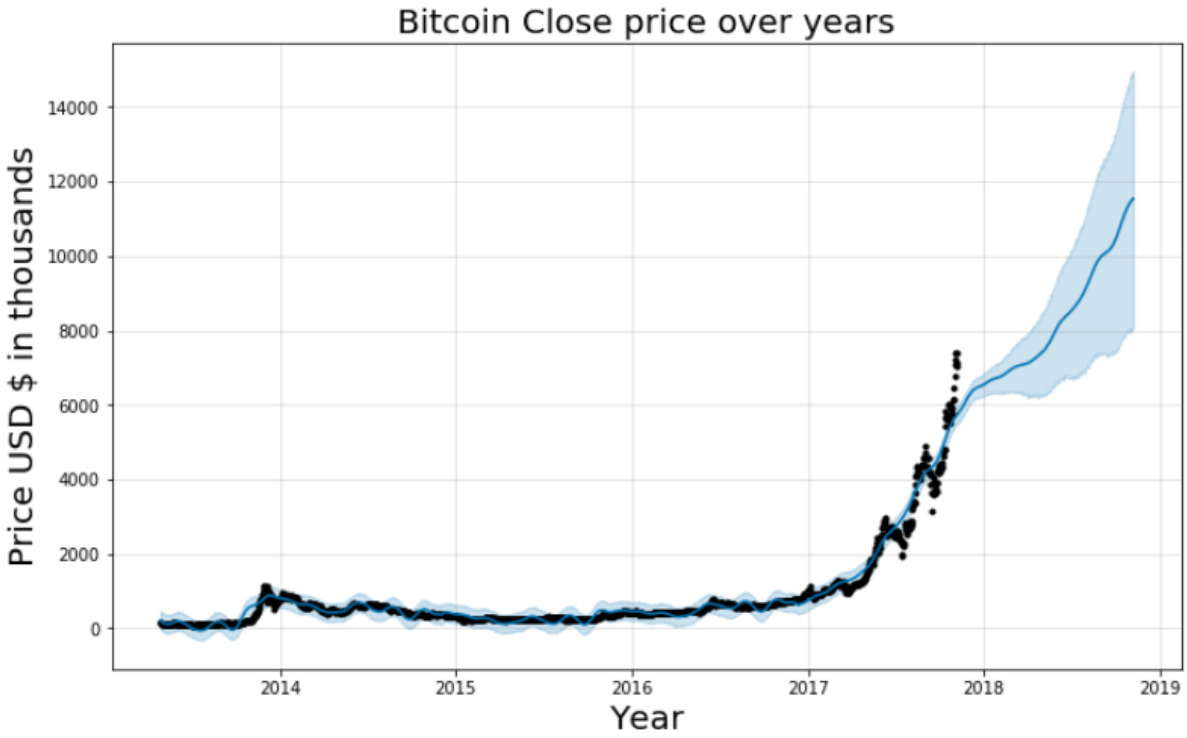


Unfortunately, the predictions don't even come in range with the true values. It looks like the model was overfitted as the R2 score resulted in a 0.97.

From there I decided to try another type of library for forecasting called Prophet which was created by Facebook for python. Prophet allows for quicker modeling, at it's core it's an additive regression model with these main components:

- Piecewise linear/logistic curve trend. It automatically detects changes in trends by selecting changepoint from the data
- Yearly seasonal components are modeled using Fourier series (function as sum of simple sine waves)
- Allows for weekly seasonal components using dummy variables
- Custom user-provided list of important holidays.

After creating an model instance with periods forecasted out as 365 days and on daily period, below is the plot at first untuned model within Prophet:

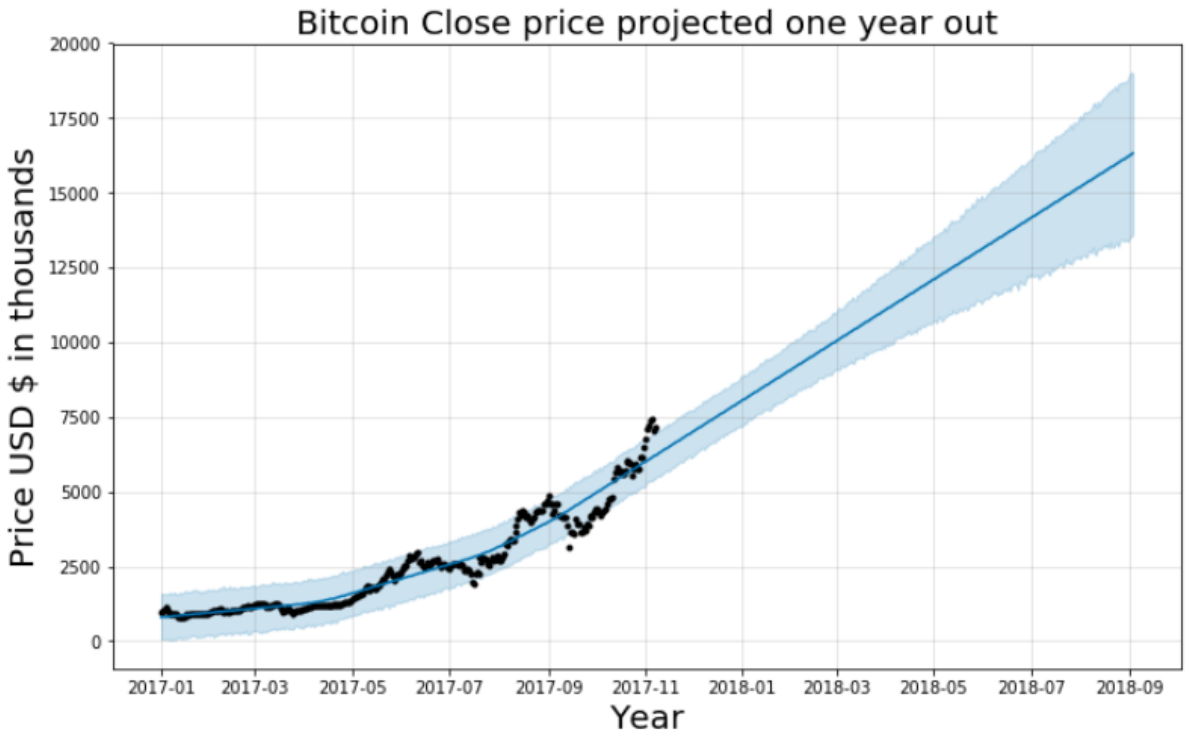


If you look closely at all values greater than 2018, the projections look promising but slowly start to taper off. Also, the uncertainty levels are smaller than they should be considering the volatility of the data. Looking at the seasonal components we can get a better understanding of how the different forecast components look. The price of bitcoin doesn't move up much until the beginning of January and from there it starts trending up.

At this point we have a vanilla model up and running that shows a solid forecast for 365 days out. Realistically the forecast is off considering the current BTC prices of where they're at but this is a good start given the data we have up to this point.

To further finetune the model and make more accurate predictions, I increased the uncertainty interval width to include a wider range of values to due to the volatile nature of this data as well as start the model to take data from '2017-01-01' as that's where most of the growth in pricing occurs moving forward.

After adjusting for these, below you will find an updated forecast to look somewhat reflective of what prices should be like today:



As you can see, the price is forecasted to reach above $15K around July 2018 and forward.

Next Steps:

- Consider more research into the other types of models that can be used to better forecast volatile data such as bitcoin prices such as GARCH, GAS, Gaussian, and Bayesian Models.
- Scrape most recent bitcoin pricing that's not included in the dataset to add more values to the forecast.
- Overlap other variables and feature to create a multi-variate time series model, possibly layering over other cryptocurrency prices with bitcoin to obtain a more accurate forecast

- To try aggregating models over each other. There's a function in Pyflux that can do this and it would be helpful to see how that helps with forecasting results.
- Overall, I would like to try other time series models and see what's more appropriate for such high volatile and variation like data as cryptocurrencies. We only covered a 2-3 in class and there over 15 or so models in time series that can be used. Understanding their intricacies and how they can be used is challenge and require more time and research.

Github:

- https://github.com/altoid51/DSI-Capstone/blob/master/Capstone-Bitcoin_Price_Trends.ipynb

References:

- https://research.fb.com/prophet-forecasting-at-scale/
- https://www.datascience.com/blog/introduction-to-forecasting-with-arima-in-r-learn-data-science-tutorials
- http://www.pyflux.com/
- https://machinelearningmastery.com/decompose-time-series-data-trend-seasonality/
- https://www.percona.com/blog/2017/03/20/prophet-forecasting-our-metrics-or-predicting-the-future/
- http://pbpython.com/prophet-overview.html
- https://arnesund.com/2017/02/26/using-facebook-prophet-forecasting-library-to-predict-the-weather/
- https://www.youtube.com/watch?v=0unf-C-pBYE
- https://www.youtube.com/watch?v=tJ-O3hk1vRw&t=2552s
- https://www.analyticsvidhya.com/blog/2015/12/complete-tutorial-time-series-modeling/
- https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/
- https://www.otexts.org/fpp/6/4
- https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/
- http://www.seanabu.com/2016/03/22/time-series-seasonal-ARIMA-model-in-python/
- https://people.duke.edu/~rnau/411arim3.htm
- https://www.udemy.com/python-for-finance-and-trading-algorithms/