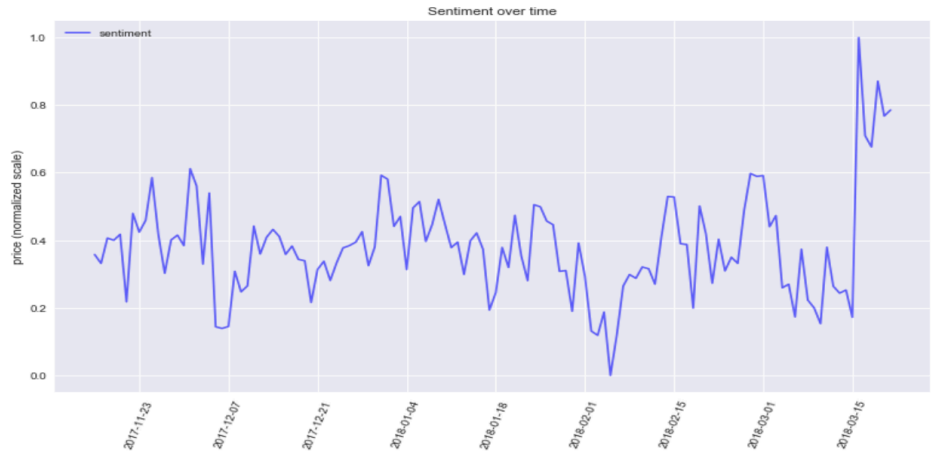
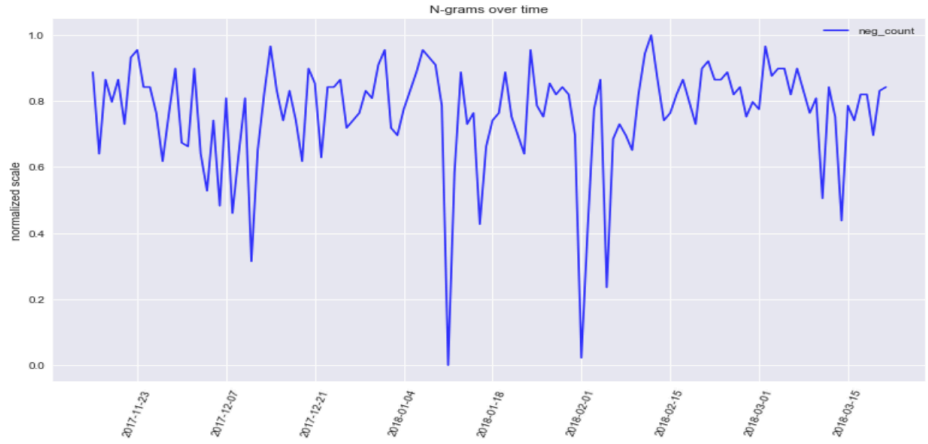
Investigating new features in Ethereum price prediction:

What effects the price for ethereum as opposed to bitcoin? Bitcoin is the first crypto currency made, the main coin. Arguably it is the coin that people get into when they first buy into the hype of crypto. Ethereum is said to be preferred by the techies because of its acclaimed superior blockchain algorithm and support for smart contracts. That being said, the differences in the nature of each coin's technology are still debated as pros vs cons.

As far as prediction of price goes, the coin's technology currently has no real "direct" implication on its price thus far, at least not in the way that some identifiable external stochastic variables seem to. It's so speculative as to what actually influences the rise and fall. (or is it? Stay tuned!) For this, predicting price is another animal. In regards to predicting ethereum price, out of all the features I looked at, I expected bitcoin price to score the highest in feature importance when predicting ethereum price. Instead I saw that the dow jones index scored the highest in feature importance by a landslide when training the predictive model for ethereum. This is now when I thought to try predicting bitcoin using the other remaining features (including Ethereum price) and Dow jones index scored very low in feature importance when predicting bitcoin, and it was google search frequency that scored high. The interesting takeback from this is that the dow jones index seems to be highly predictive of ethereum price but not bitcoin price. We used our model to our model to predict various ethereum values. This is not a prediction, since we had to train our data on 80% of randomly shuffled dates and respective values, and test accuracy on the remaining 20% of the shuffled data. The idea is that we trained a model to in fact predict Etheruem values when given a particular values for the Dow jones index.

Scraping Data with the Reddit API:
Scraping four months of reddit posts of about 12 different subreddits, took a very long time. I figured to make as dense of a dataset as possible in order to accurately sample the space. (reddit at its best embodies the following: what is being spoken about on the internet on a certain topic… reddit is a perfect place for this because it is essentially where people funnel existing informational web sources to one place and then have debate and discussion.) but given how much noise exists in internet posts, we needed a dense amount of information to accurately sample the space.





The exploration of new features:

I've always held a strong belief that public sentiment and the reaction against certain current events in the media, had some influence on the market price.

I sat down to think what could it be that effects ethereum price? And where can I find it? Aside from conspiracy theories of massive one-party trade orders to manipulate the market, I felt for sure there is something in the wind of internet text data. (Yes, that analogy was inspired by

weather prediction since both the weather and stock market are non-deterministic.) I had some fun Deciding which features to use and test and building a model with this.

1) Current Events in Media related to the backing and banning of bitcoin
   (for example the bitcoin crash in September when china banned bitcoin or the ripple price spike when banks said they would “back it”)
2) General sentiment: I again used text blob to try to analyze the daily sentiment
3) Bitcoin price
4) Dow jones index
5) Google search frequency

Obtaining the features:

features I was trying to examine were sentiment, google search frequency, and bitcoin price, and my personal favorite feature: the occurrence of specific “n-grams” in a post, specifically these n-grams of interest can be described as the following.

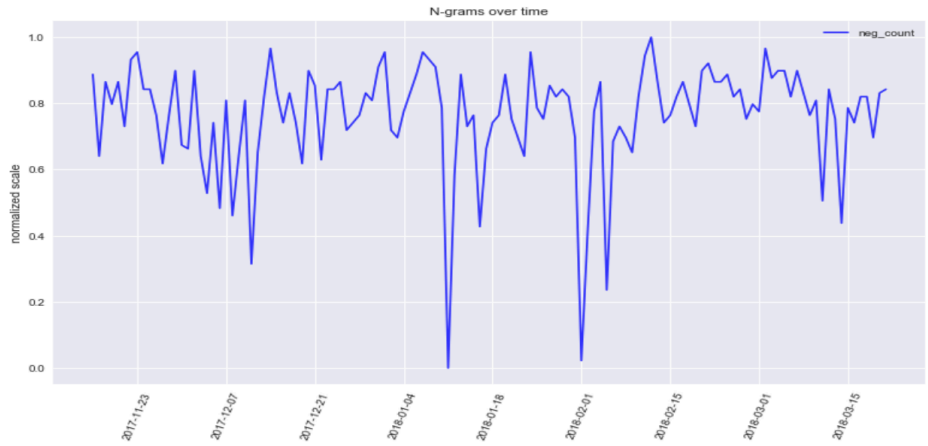
that signaled if a current event regarding the “backing” or “banning” of crypto currency, was being discussed. t takes into account historical price fluctuations to predict future ones. The issue is, this type of modeling doesn’t predict well with large rises and falls in our data which is what we had within the range of prices we were aiming to test.

Constructing the data for the sentiment and “n-gram” features used in the model.
I first had fun getting to learn to use Textblob, a very robust yet straightforward library for text analysis. I converted all the reddit post strings to textblob objects then implemented embedding and parts of speech tagging. There is a built function that allows one to get the sentiment polarity score of a string of text, so I made a new column in my dataframe titled “Sentiment” and

Counting “back’s” or “ban’s” N-grams:

I wrote a chunk of code that sifted through a given reddit post (now textblob object) and implemented embedding and parts of speech tagging. I now wanted to test if particular 3-gram exists. I would count the occurrence of any 3-gram if the following conditions are simultaneously satisfied. If the first word in that 3-gram is a proper noun 'PPN' , the second word is a word like "back's" or "endorse's" or "support's" (any words from a complete list I call "back's_list") and if the third word would be a word like "bitcoin" or "crypto" (again any words from a list "crypto_list"). If all three of these requirements are met, then a count would be incremented and saved as a "pos_count". The variable "pos_count" a count that tallies up all the times proper nouns have backed or endorsed cryptocurrency. The same would be done to keep track of the times that a proper noun has "banned" bitcoin or crypto and the count would be stored under "neg_count."

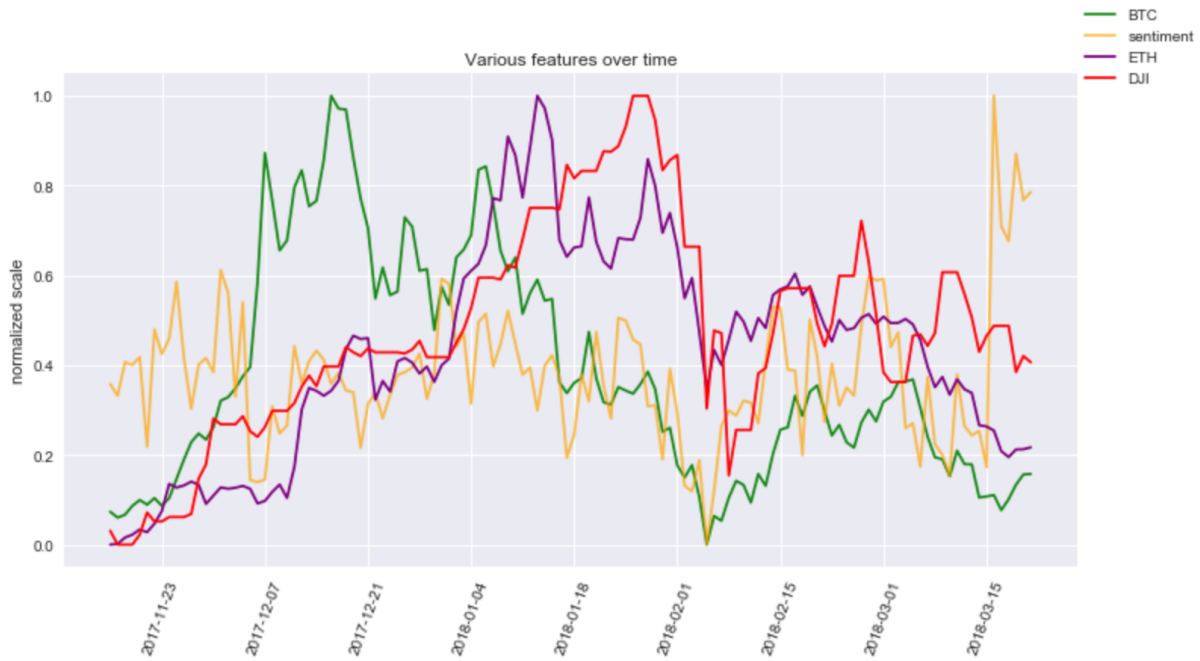
To illustrate with some examples, if the 3-gram "JPMorgan back's ripple" exists, this would increment the "pos_count" variable and if the 3-gram "China bans bitcoin" exists then the "neg_count" would increment. After iterating through an entire reddit post, These counts are saved in another column in the dataframe, and then the pandas.groupby method can take the sum of these counts from all posts for every day in the index. We would then be left with data sorted by day with a corresponding sum of counts. Below I included a graph of n-grams over time.



This was very time-expensive and it needed to be narrowed down to only being performed on a few subreddits, r/bitcoin. When the data was collected for the feature, it was unfortunately fairly sparse and possibly inconclusive. I would like to revisit constructing this with a GPU although the GPU would need support for the textblob library, so perhaps an extremley fast CPU could make enough of a difference upon revisiting constructing this feature.

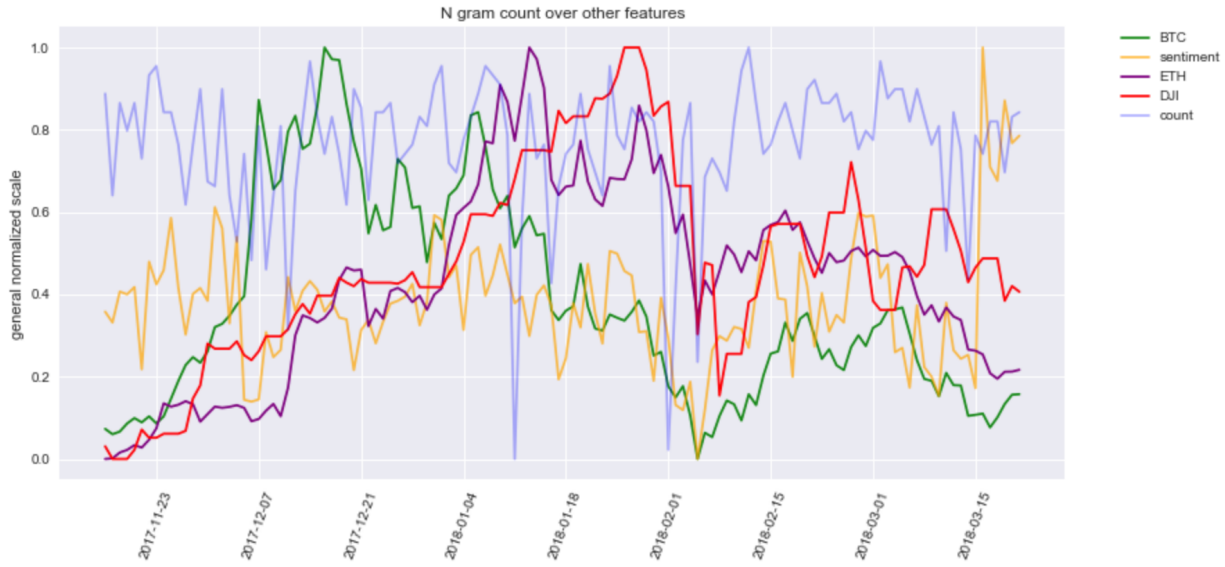
Comparing Possible features through examining EDA:

Once we constructed and gathered all the data to create our features of interest, we graphed all the data on (a normalized scale) over time as seen below.



We can see the purple is the ETH price and our target variable. From this we can also see the red data series (Dow Jones Index) lines up well with the purple data series (ethereum price) and this could indicate that it may be of strong feature importance for our target variable which is in fact Ethereum price. The next data series that seems to be trending similarly is the green data series or Bitcoin price. We can see thought that the orange data series is not trending very similarly with the Ethereum data but we can't rule out that it may still have some feature importance and may add to the accuracy of the model.

In the graph below we take the same graph as seen above but overlay it with the n-gram count data series (blue) and here we see ngrams count do not follow in the trend as the target, we will later see that the feature importance score on this data series goes to zero.



Modeling:

Random Forest Regression:

Then the random forest model was what worked best in the end. It was very simple to use I used a standard scaler for my data, then I selected subsets from my data frame to be converted to np arrays since that is the required format that the function takes in. My feature and label a test train split, and to perform an optimization of parameters using a grid cross validation search. We defined our regr = RandomForestRegressor(max_depth=2, random_state=0) first picking max_depth=2, and random_state=0 to later perform a grid cross validation search amongst a few parameters.

We fit the regressor with regr.fit(X_train, Y_train) and then looked at the feature importance with regr.feature_importances_.

```
In [13]: print(regr.feature_importances_) #strength of importance of feature
         [0.01590281 0.93784831 0.03763274 0.00459188 0.00402426 0.
          0.        ]
```

(This can be seen at output 13 in the file RandomForesetRegressor.ipynb under the MODELING folder in the github repository for this project.)

The above numbers signify the following features respectively. 1) search frequency 2) Dow Jones Index 3) Bitcoin Price 4) sentiment 5) count of positive n-gram 6) count of negative n

gram (effective net by first multiplying the vector by a negative 1 then substracting its minimum) 7) the addition of the last two.

This is where we noticed that unfortunately the new features I wanted to test out did not exactly help improve the model so much. Sentiment and positive n-grams were essentially non-zero and the last two features were indeed zero contribution to the predictive model. Although this was a bit of a disappointment, it doesn't mean we didn't notice some interesting results. The Dow jones index was the highest importance out of all the features.

```
regr.score(X_test, Y_test)
```

```
Out[15]: 0.8038670793562294
```

```
In [20]: grid_cv_regr.score(X_test, Y_test)
```

```
Out[20]: 0.9245428491244888
```

I then proceeded to drop the features that were zero or deemed essentially zero. After doing so, we are predicting ethereum price based off of the features of google search frequency, Dow Jones, Bitcoin price. In the end it was Dow jones index which was calculated to have the highest feature importance when I was expecting it to be bitcoin price.

```
In [24]: print(regr.feature_importances_)
```

```
[0.01590281 0.94405113 0.04004606]
```

```
regr.score(X_test, Y_test)
```

```
Out[26]: 0.8017445780922451
```

```
In [33]: grid_cv_regr.score(X_test, Y_test)
```

```
Out[33]: 0.9439053866082423
```

I then proceeded to drop the features that were zero or deemed essentially zero. After doing so, we are predicting Ethereum price based off of the features of google search frequency, Dow Jones, Bitcoin price. In the end it was Dow jones index which was calculated to have the highest feature importance when I was expecting it to be bitcoin price. The feature importance was very high of about 0.92 for the R value and then after the grid cv search we were able to optimize it to about 0.94.

I then wanted to check is the Dow Jones index correlated to bitcoin in a similar way, since I held the common belief that bitcoin was the main coin influenced by external variables and that other coins followed the rise and fall of bitcoin? When I tested this (as can be seen in my Random forest modeling python notebooks for bitcoin on this projects github repository) the coefficient for dow jones was very low (about 0.08!) And google search trends was of highest feature importance (about 0.70). But then again, we reflect on the fact that google search trends is most likely a causation from price rising/dropping and probably not the other way around, (although we can't exactly rule that out either) but, trying to predict the search trends count (possibly with a time series analysis) and using that as a feature in the model to predict the bitcoin price, just seems a little bit backwards and most likely not very accurate.

When removing google search trends, Dow Jones Index was as expected the highest importance, but not very high in feature importance, which means it does not improve the model very well. We went through and with the prediction which only gave us an R squared value of about 0.4 and then after performing cross validation grid search we were able to bump the R value to about 0.6 (as can be seen in the notebook _____ in folder _____ of the github repository)

Random Forest Classifier:

I decided to create a random forest classifier to see if the price would rise or fall for the coin for Ethereum. I used the price of Ethereum to create a new list that says if the price of today is greater compared to the day before then we assign a "1" to that date and if it fell in price compared to the day before we would assign a "0', resulting in a new dataframe column, a vector of 0's and 1's. I then assigned the 0 and 1 vector to be the "labels" or "target variable" keeping the old feautures from before for this new model. After the prediction results were very poor, I realized I made a mistake in keeping the old features as is (i.e I know the price of bitcoin, ethereum, Dow jones, as their actual values) the issue with this is that if you were to take an entire row and exampine it, the values in the feautures coulums give no information on the cell that came before, and that is exactly the target we are trying to predict. According to our model the prices of today tell nothing about what the prices of yesterday were.

To remedy this issue, I turned all the feautures columns into columns of 0's and 1's, where each entry for each date would be a 0 or 1 corresponding to weather the prise rose or fall from the day before. When I used these binary rise/fall columns as feauture's, the classification model improved quite a bit. In the previous scenario the results were 0.46 and .60 with cross validation improvement, but now with our features being binary indication a rise or fall, we got the classification model score up to 0.68, and with no improvement after cv hyperparametriation.

So we got a really good model performance with the price prediction. Since this data set was only over 4 months, I wanted to see my results with a longer data set using only bitcoin price and DJI considering the sentiment data was not very indicative and deeming the google search frequency resultant from price. Over the span of 3 years of historical data, the feature

importances were about 0.713 for Dowjones index and 0.286 for bitcoin price. The regr.score was about 0.931 and the grid_search_cv score was about 0.975!

These are great results! but keep in mind, this is not forecasting, this is just training the model, the idea is after a model is trained, that it could possibly be used to predict prices when given a value for Dow Jones index or for bitcoin price. I attempted Time series analysis on Ethereum directly but it was deemed unsuccessful mostly due to the fact that Ethereum does not have enough historical data. Since it was possibly time series analysis can be performed on these two features to get

ARIMA:

Something that I first investigated but later realized to be a slight detour, The ARIMA model (Autoregressive Integrated Moving Average) was originally of great interest to me upon starting this project. I knew that I could build an ARIMA model based off of the historical and then apply a few exogeneous variables using the ARIMAX model. (ARIMAX is essentially the ARIMA model with the extra input of exogeneous variables)

The exogeneous variables that I was interested input into the model would be the ones I described in the above section,

Why did in the end ARIMA and ARIMAX not end up not being the best choice for a model? This is 1) because of the nature of the data (one large rise and one large crash in the history of its existence) and because of the fact that I scraped text data which covers the span of 4 months, and since it was the information extracted from text that I mostly wanted to test out as a feature, I could only train an ARIMAX model on those 4 months.

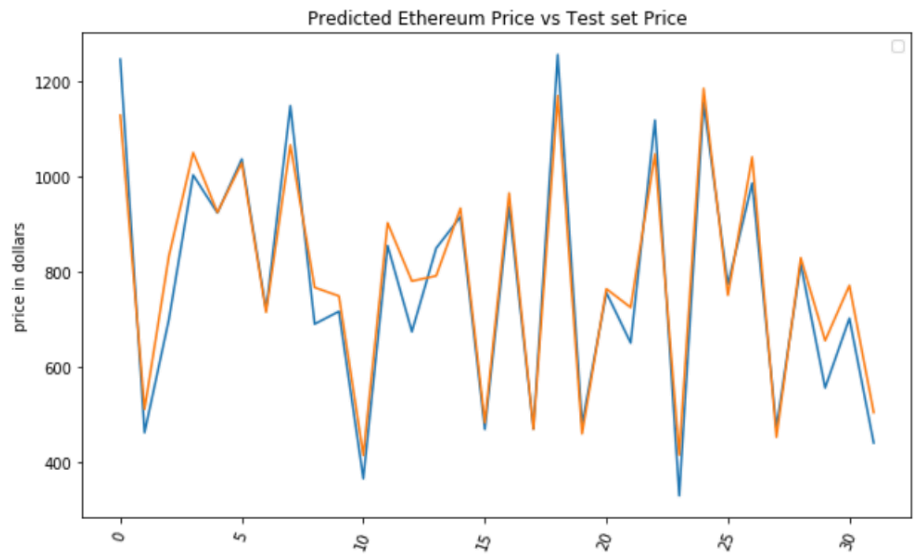
In other words we did not have a long enough range of scraped data of the exogeneous features that I had in mind in order to make this model work. And that within this range, the steep large rise and crash (a fairly recent development) in our data is not something that Time series analysis can model or forecast very accurately.

I implemented many time-series analysis techniques such as the augmented dickey fuller test, the augmented dickey fuller test, differencing was implemented in order to stationarize data, parameters were chosen by examining the ACF and PACF of our time series data, and the Granger Causality Test was performed between features. A few notebooks where these methods were applied to my data can be found on the github to this project. Unfortunaley the model was not predicting very accurately.

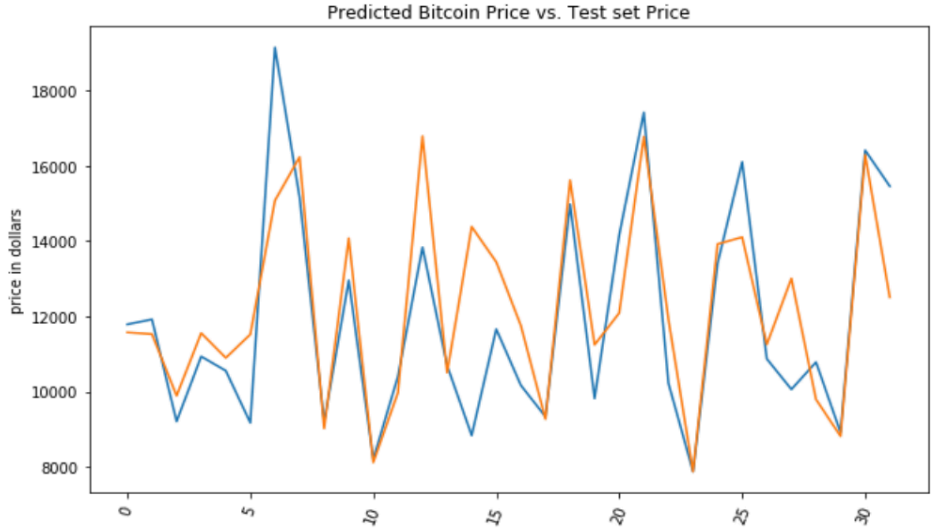
RESULTS:

(DJI even more so with ethereum than DJI with bitcoin? What does that imply? That in general, investors are turning towards ethereum?

So the general results are that ethereum can be strongly predicted given the DJI index.



And below we have Bitcoin.



The issue for the latter is that google search frequency was the strongest feature in the model. But if one wanted to predict values for a few google search values you can't really predict google search frequency because it is understood that the search frequency is a RESPONSE to the bitcoin fluctuation, aka bitcoin price is causal to the google searches so to predict google searches you would need to predict bitcoin price.

but for the latter, hypothetically, you can predict Dow jones with time series analysis of its historical data, and if you have decent accuracy with the time series prediction, you could use the predicted Dow Jones index values to then feed those predicted values into a regression model.

CONCLUSION:

This project has been quite a journey, I did quite a bit of scraping and text analysis for two features that in the end did not really add to the performance of my model, but I am still stubborn not to let this one go! Ideally my future plans for this project to scrape more data from different crypto forums and articles, (also maybe hacker news and twitter), so that I have an even denser data set, and learn how to deploy this heavy computation of constructing sentiment or the "n-gram" feature, to a High powered computing cluster. I still hold the belief that the sentiment and current events have influence on price. That being said, to sample the entire data set of where the measure of general sentiment and where an event occurrence exists (AKA the whole internets crypto news output and discussion forums) is highly dimensional! This means an extremely large dataset would be necessary due to accurately sample your space! Ah! Here we are met with the curse of dimensionality!

Future Plans with this project:

Ideally I would like to build an automated model where it predicts for about 4 days into the future but as every day passes, it takes the actual data collected from that day, stores to a database, and adds to its training set, shifting the predictive set up by one day.
(We also looked into, How has the volume increased with bitcoin as opposed to ethereum?)