Cryptocurrency Price Prediction Facilitated by Article Sentiment Analysis

Scott Sims

Table of Contents

1. INTRODUCTION

To all investors out there, how does almost 4,000%, 88,000%, or even 15,390,000,000% R.O.I. Return on investment sound? Well those figures are the YTD returns on Litecoin (LTC), Ethereum (ETH), and Bitcoin (BTC) respectively. Since the first decentralized cryptocurrency, Bitcoin, was created in 2009, upwards of over 1500 new coins have been incepted. This has created a realm with a market cap of over $1 Trillion. Millionaires have been created within years, months, even days. But with great gains also comes great losses.

At over 1500 coins to invest in, there are undoubtedly huge winners, and huge losers in the cryptoworld to choose from. Each coin is usually tied to some project to bring it value. Other coins are just plain and simple frauds. However, even if a coin is legitimate, there's no guarantee that the project backing it will succeed, nor are there guarantees it is a project the market is interested in. Adding to the ambiguity of crypto investment strategy is the space's decentralized nature. Since there is no governing body controlling the value or flow of these currencies, price fluctuations are at the mercy of free market forces only. Simply put, the market controls the supply as well as the demand.

An algorithm evaluating sentiment for daily news articles on a coin's trading performance and/or it's associated project, would facilitate an investor in improving their R.O.I. Conversely, it could also minimize the damage in a poor investment.

2. DATA ACQUISITION

There are two main categories of data being extracted for this project. The first type is historical data price from Investing.com. No API was needed to extract this data since the website provides an easy to use feature to download the desired data in CSV format. The data columns within the CSV are: Price, Open, High, Low, Volume, and Day to Day Change Percentage.

The ultimate goal would be to develop an API that would learn an algorithm for any coin of interest. However, Ripple (XRP) was utilized as the coin of interest for this project to establish a foundation and proof of concept. Most exchanges don't allow direct trading of cryptocurrency coins with fiat[1]. Because of this, the most common currency used to purchase cryptocurrency is other cryptocurrency. In conjunction with XRP historical data being utilized, historical data was also extracted for Bitcoin (BTC) and Ethereum (ETH). These coins were selected because they are the most predominantly used forms of currency to buy all other crypto. Also, the market as a whole tends to follow the trend of Bitcoin.

The second type of data extracted was 360 different news articles from 5 different cryptocurrency news sites. These articles ranging from May 16th, 2017 to June 4th, 2018 were scraped from Bitcoin, CNBC,

1 Fiat money is currency that a government has declared to be legal tender, but it is not backed by a physical commodity. The value of fiat money is derived from the relationship between supply and demand rather than the value of the material from which the money is made.

Coindesk, Cointelegraph, and Forbes. Although these Articles were manually searched for in each website, a custom Python web scraping class was created to aid in parsing all articles. This class can be viewed in Appendix A or in this Jupyter Notebook. The class takes advantage of the BeautifulSoup and Requests libraries of Python.

The web address for each article is contained in a text file for each site as seen here. A custom function that utilizes the web page parsing class then iterates through each article's web address, accesses the site, scrapes the article text, headline, as well as the date and time of writing. These items are then written to CSV where each row is an individual article as seen below:

	A	B	C	D	E	
1	Date	Coin	website	Headline	Text	Link
2	2018-02-09 12:03:00	Ripple	forbes	Crypto Watch: Ripple (XRP) Price Surges 21% in 10 Hours	GERMANY, BONN - JANUARY 31: Symbol photo on the topics crypto currency, digital currency, currency speculation, course gains, course losses, money laundering, crime. etc. The picture shows Ripple coins and a Bitcoin (physical). (Photo by Ulrich Baumgarten via Getty Images) Ripple solidifies #3 cryptocurrency spot with a $35.7B market cap. [Ed note: Investing in cryptocoins or tokens is highly speculative and the market is largely unregulated. Anyone considering it should be prepared to lose their entire investment.] After a general trend of consolidation in the worldwide cryptocurrency market—following drops in the Dow Jones industrial average, Facebook banning cryptocurrency advertising, news of Chinese and South Korean regulation, India pushing toward a crackdown and news of Tether and Bitfinex subpoenas—the market cap is beginning to inch back up after January's lows. One coin in particular made big leaps overnight (PST): Ripple (XRP). Disclosure: I own some Ripple. According to CoinMarketCap, Ripple was trading for $0.771540 USD at 9:19pm PST Feb. 8. By 6:44am on Feb. 9, that number was $0.931411—a jump of 21 percent in just 10 hours. Ripple 24-hour trading chart The price is currently $0.916888, 21 percent higher than it was a day ago. This performance is 2-4x higher than other major cryptocurrencies such as Bitcoin, Ethereum, Bitcoin Cash, Litecoin, and Cardano.GERMANY, BONN - JANUARY 31: Symbol photo on the topics crypto currency, digital currency, currency speculation, course gains, course losses, money laundering, crime. etc. The picture shows Ripple coins and a Bitcoin (physical). (Photo by Ulrich Baumgarten via Getty Images) Ripple solidifies #3 cryptocurrency spot with a $35.7B market cap. [Ed note: Investing in cryptocoins or tokens is highly speculative and the market is largely unregulated. Anyone considering it should be prepared to lose their entire investment.] After a general trend of consolidation in the worldwide cryptocurrency market—following drops in the Dow Jones industrial average, Facebook banning cryptocurrency advertising, news of Chinese and South Korean regulation, India pushing toward a crackdown and news of Tether and Bitfinex subpoenas—the market cap is beginning to inch back up after January's lows. One coin in particular made big leaps overnight (PST): Ripple (XRP). Disclosure: I own some Ripple. According to CoinMarketCap, Ripple was trading for $0.771540 USD at 9:19pm PST Feb. 8. By 6:44am on Feb. 9, that number was $0.931411—a jump of 21 percent in just 10 hours. Ripple 24-hour trading chart The price is currently $0.916888, 21 percent higher than it was a day ago. This performance is 2-4x higher than other major cryptocurrencies such as Bitcoin, Ethereum, Bitcoin Cash, Litecoin, and Cardano. After exhibiting unprecedented growth in December 2017, Ripple then became the worst-performing major coin of January, performing at -49.56 percent. The recent surge follows announcements of partnerships with IDT, Mercury FX, MoneyGram and Santander as well as an "open payments corridor" to China via LianLian Yintong Electronic Payment Company. Furthering its position in Asia, SBI announced the creation SBI Virtual Currencies on Jan. 30, an exchange that will help Asian markets bridge the gap from fiat to crypto. Ripple still has a long way to go to reclaim its #2 spot on the cryptocurrency market, as Ethereum is holding strong with $83.3B—more than double Ripple's current market cap. Editor's Note & Disclosure: The author invests in cryptocurrency markets. Neither the author nor Forbes endorses participation in any token sale or cryptocurrency investment, all of which have significant inherent risk. Seek advice from a financial advisor as well as do your own due diligence before considering investment. Jesse Damiani is Editor-at-Large of VRScout, Series Editor of Best American Experimental Writing, and CEO of Galatea, a screenwriting and project management tool for AR and VR stories. Jesse is an entrepreneur, advisor, journalist, and public figure in emerging technology. He is Editor-at-Large of VRScout, Series Editor of Best American Experimental Writing, and CEO of Galatea, a writing and project management tool for VR and AR stories. He regularly cover...	https://www.f
3	2018-02-07 04:45:00	Ripple	forbes	Mellon Banking Heir's New Crypto Fortune: Almost $1B In Ripple's XRP	This story appears in the February 28, 2018 issue of Forbes. Subscribe Matthew Mellon Individual Investor Crypto Net Worth: $900 million-$1 billion* Matthew Mellon This heir to one of America's great banking fortunes, and a former chair of the New York Republican Party's finance committee, has struggled with drug addiction. So when he began dabbling heavily in cryptocurrencies years ago, his friends and family tried to dissuade him, figuring it was another erratic obsession. And, indeed, he abandoned some early investments and sold his Bitcoin a few years ago. But then Mellon got turned on to XRP, spending some $2 million to acquire coins, which he liked because it's one of the few cryptocurrencies that are working within the banking system. "Crypto is scary and dark. It's anti-America," says Mellon, 54. "I am pro-America, pro-business and pro-bank. That's why I went with Ripple." Mellon's XRP is worth around $1 billion. Recently	

Figure 2:1 - Example of Forbes articles written to CSV after being parsed with custom function and web page parsing class

3. DATA CLEANING AND PREPROCESSING

The historical price data for each coin is already complete and didn't require any imputations or extensive cleaning. A desired date range was selected, and all data for that range is downloaded to CSV. The range was selected so that data was present for all 3 coins and also to create a time series that coincided with the time series of the scraped articles. Due to string values present within the data and different orders of magnitude for the volume, minor manipulation had to be performed on this time series. The bulk of data cleaning and preprocessing was performed on the scraped articles. Since the goal was to perform sentiment analysis on the text, they had to be prepared in a manner in order to improve accuracy of the sentiment score. Essentially, everything that provided little to no value to the ultimate meaning or sentiment of the text had to be removed.

3.1 Historical Price Data Preprocessing

The price data for all 3 coins was represented by a multi-index data frame. Although the bulk of the data was numerical, there were a lot of non numerical values that needed to be converted, namely “Change %” and “Vol.”.

	Coin	XRP						BTC						ETH					
	Date	Change %	High	Low	Open	Price	Vol.	Change %	High	Low	Open	Price	Vol.	Change %	High	Low	Open	Price	Vol.
Date																			
2017-08-01	2017-08-01	9.57%	0.1850	0.1585	0.1630	0.1786	26.82M	-4.37%	2901.6	2615.8	2854.3	2731.2	42.31K	12.53%	232.59	200.80	200.81	225.97	352.31K
2017-08-02	2017-08-02	-4.59%	0.1800	0.1680	0.1786	0.1704	11.47M	-1.07%	2757.8	2640.0	2733.8	2702.0	23.76K	-3.53%	229.70	215.00	225.97	218.00	159.47K
2017-08-03	2017-08-03	2.46%	0.1772	0.1688	0.1704	0.1746	4.59M	3.27%	2813.0	2698.0	2702.0	2790.3	16.86K	3.11%	228.20	217.07	218.00	224.79	96.19K
2017-08-04	2017-08-04	-0.57%	0.1780	0.1710	0.1746	0.1736	7.21M	2.50%	2874.8	2762.6	2790.3	2860.0	18.99K	-1.81%	228.30	218.66	224.79	220.73	83.46K
2017-08-05	2017-08-05	5.99%	0.1920	0.1710	0.1736	0.1840	20.38M	13.86%	3331.9	2855.0	2860.0	3256.4	50.56K	15.01%	259.90	219.23	220.73	253.87	234.87K
2017-08-06	2017-08-06	-2.01%	0.1898	0.1765	0.1840	0.1803	12.15M	-0.88%	3309.8	3156.0	3255.0	3227.9	16.95K	4.10%	271.50	251.00	253.87	264.29	209.86K
2017-08-07	2017-08-07	-0.72%	0.1830	0.1763	0.1803	0.1790	10.48M	5.23%	3440.0	3190.0	3232.5	3396.7	24.54K	1.98%	274.60	257.32	264.29	269.51	146.17K
2017-08-08	2017-08-08	9.44%	0.2000	0.1768	0.1790	0.1959	31.91M	0.54%	3482.9	3343.8	3395.0	3415.0	31.72K	9.77%	299.50	265.00	269.51	295.83	207.56K
2017-08-09	2017-08-09	-6.38%	0.1959	0.1800	0.1959	0.1834	13.38M	-2.20%	3424.4	3236.8	3415.0	3339.9	26.25K	-0.62%	316.00	277.00	295.83	293.99	400.20K
2017-08-10	2017-08-10	-1.42%	0.1850	0.1770	0.1834	0.1808	11.26M	2.04%	3444.4	3310.0	3340.2	3407.9	19.13K	0.67%	310.05	287.60	293.99	295.97	155.41K

Figure 3:1 - Historical Price Data for Ripple(XRP), Bitcoin (BTC), and Ethereum (ETH)

Ripple (XRP) was traded in the millions whereas Bitcoin (BTC) and Ethereum (ETH) ere traded in the thousands. To keep all things even, XRP volume was converted to units of thousands (K) and the three columns were converted to numerical data. Daily Change Percentage for all 3 coins was also converted to numerical values by simply dropping the % symbol.

Lastly, since XRP Price was the variable of interest, that column was shifted up by predetermined dates in order to represent future price points. Initially each XRP Price was shifted up by one day to represent “next day” price results.

3.2 Text Preprocessing

When dealing with scraped text, the first issue to combat was encoding. As the text was parsed and written to CSV files, the encoding was set as “utf-8”[2]. However when reading back from CSV, with encoding set as “utf-8”, python still manage to misrepresent some characters. Thankfully all characters encoded incorrectly were the same and contained the following string: \xao. This string is non-breaking space in Latin1 (ISO 8859-1). A simple replace function fixed all these incorrect strings.

Next, each website’s articles contained certain sentences or phrases within them that brought no added value to the text. These sentences/phrases for each text varied a little in writing and punctuation, but for the most part they were the same in each article. This made it relatively easy to remove from the text. The following table shows an example of these dispensable sentences from each medium.

2 A character in UTF8 can be from 1 to 4 bytes long. UTF-8 can represent any character in the Unicode standard. UTF-8 is backwards compatible with ASCII. UTF-8 is the preferred encoding for e-mail and web pages

Forbes	*[Ed note: Investing in cryptocoins or tokens is highly speculative and the market is largely unregulated. Anyone considering it should be prepared to lose their entire investment.]*
CNBC	*Playing Share this video... Watch Next...*
Cointelegraph	*Price Analysis The views and opinions expressed here are solely those of authors/contributors and do not necessarily reflect the views of Cointelegraph.com. Every investment and trading move involves risk, you should conduct your own research when making a decision.* *Follow us on Facebook Follow us on Facebook Hide Comments For updates and exclusive offers, enter your e-mail below. Cointelegraph covers Fintech, Blockchain and Bitcoin bringing you the latest news and analyses on the future of money. Thank you for contacting us! We will reply to you as soon as possible. Thank you for your interest in our franchise program. We are considering your request and will contact you in due course. If you have any further queries, please contact: franchise@cointelegraph.com*
Coindesk	*Disclosure: CoinDesk is a subsidiary of Digital Currency Group, which has an ownership stake in Ripple. Climbing wall image via Shutterstock The leader in blockchain news, CoinDesk is a media outlet that strives for the highest journalistic standards and abides by a strict set of editorial policies. CoinDesk is an independent operating subsidiary of Digital Currency Group, which invests in cryptocurrencies and blockchain startups.*
Bitcoin	*https://t.co/CNaJZzHtaZ pic.twitter.com/huq84amUJG Download the Bitcoin.com Wallet right to your device for easy and secure access to your bitcoins. Perfect for beginners, the Bitcoin.com Wallet makes using and holding bitcoins easy. No logins required. Facebook has recently made some changes to show you less news and more family-oriented pics and videos – even though you've previously subscribed. Here's how to make Facebook show you our content again; Facebook has recently made some changes to show you less news. Here's how to make Facebook show you our content again; a. Visit our Facebook Page https://t.co/d5QIuhCFPt b. Click the "Follow" button under our cover image. c. Select the "See first" option. pic.twitter.com/tFmrF73WQ7 — Bitcoin News (@BTCTN) February 19, 2018 1. Visit the Bitcoin.com Facebook Page. 2. Click the "Follow" button under our cover image. 3. Select the "See first" option. Get the latest price charts, statistics and our news feed on your site. Check out our widget services. We also deliver bite-sized news to your favourite messaging app; join our Telegram channel.*

There were other superfluous artifacts that were manually removed from each article, but the most reoccurring ones for each article are those listed in the table above.

3.3 *Further Preprocessing*

While performing sentiment analysis, Python does have a few sentiment analysis libraries that can be utilized. Two of these sentiment libraries were experimented with to ascertain performance capabilities on ranking our cryptocurrency articles. Vader sentiment, which comes bundled within the Natural Language Tool Kit (NLTK) library, and TextBlob, which comes bundled within the textblob library, were initially used. It wasn't enough to just remove unnecessary sentences from article texts before using these libraries. The following preprocessing had to be performed:

- Words Only – The process of removing numbers, punctuation, or any element that is not a capital or lower case letter from text.

- Stemming – Stemming algorithms work by cutting off the end or the beginning of the word, taking into account a list of common prefixes and suffixes that can be found in an inflected word. The NLTK library has several stemming algorithms: The Lancaster, Porter, and Snowball stemmers were experimented with in this project. Words were ran through each stemmer, and a list was compiled for all 3 results. From there, the result that appeared the most frequent in the list was the used stem.
- Lemmatization – Lemmatization takes into consideration the morphological analysis of words. To do so, it is necessary to have detailed dictionaries which the algorithm can look through to link the form back to its lemma. NLTK uses WordNetLemmatizer to perform this act. Please see the WordNet docs for further information
- Stopword Removal – Stop words are extremely common words that add little to no value to the true meaning behind text. E.g. [is, of, how, because, so, etc.]

A custom sentiment class was created to facilitate in performing the preprocessing and sentiment steps with these libraries. This custom class can be seen in Appendix B, or the Jupyter Notebook in which all preprocessing and sentiment was performed. To test performance of these libraries, a text was chosen at random and a sentiment object was instantiated from the custom class.

4. *SENTIMENT ANALYSIS*

The dictionary defines sentiment analysis as "*the process of computationally identifying and categorizing opinions expressed in a piece of text, especially in order to determine whether the writer's attitude towards a particular topic, product, etc., is positive, negative, or neutral.*" Performance of the "out of the box" (OTB) libraries were gaged on a randomly selected text with all different combinations of the preprocessing steps above. All results alluded to the same conclusion; the OTB libraries performed poorly ranking sentiment on cryptocurrency articles. Ultimately, a custom sentiment library had to be created to accommodate the specialized terminology of the cryptocurrency arena.

4.1 *OTB Sentiment Example Performance*

The first text randomly chosen to evaluate performance was the following:

> *"Shutterstock Ripple seizes the #4 spot with $18B market cap. Alongside major developments in Bitcoin, Litecoin, and Ethereum, other altcoins are beginning to gain traction in the broader eye. One such coin is Ripple (XRP), a cryptocurrency known for its connection with the banking world. Disclosure: I own some XRP. 24 hours ago, the price of XRP was $0.27. Earlier this morning (PST), it hit $0.51—an increase of 84 percent. At the time of this writing the price is $0.46 (CoinMarketCap).*
> *What you need to know about Ripple Alongside its cryptocurrency, Ripple operates as a payment network called RippleNet. The goal of the platform is to optimize easy transfer of funds—to almost any other currency or cryptocurrency in the world in 4 seconds. Ripple is working with banks and financial institutions to become the premier cryptocurrency of record, offering a quick, cost-effective way to transfer funds globally. For example, if you wanted to send your friend in Italy $50, you could trade for $50 worth of XRP, and they could quickly trade that out for Euros. So why has XRP gone up so quickly? The increased activity in Bitcoin following Cboe launching Bitcoin futures trading on Dec. 10, is generating more activity around the entire cryptocurrency world.*

As the #4 (and occasionally #5 when Litecoin surpassed it) cryptocurrency, of course Ripple has felt the, well, ripple effect.Continued from page 1 Shutterstock Another explanation is that as the Bitcoin craze cools off, more people are trading for Ripple—which some see as a more stable asset. "I think that markets view XRP as a very stable digital asset, so they feel safe parking funds in XRP when they exit other assets. If someone wants to get out of BTC, but doesn't want to necessarily move into fiat, he or she moves the value into XRP," said Miguel Vias, head of XRP markets at Ripple in an interview with Coindesk. Especially as network speeds lag and transaction fees soar on Bitcoin, Ripple may be an appealing trading alternative with its super-fast speeds. AMEX Partnership Unlike many cryptocurrencies, whose ethos moves away from traditional banking and financial institutions, Ripple seeks to use the new technology to optimize how money is moved. To that end, a recently announced partnership with American Express could be driving buzz around XRP. Ripple will be working with AMEX to "solve liquidity shortfalls in remittances by offering instant blockchain-based payments." "American Express has a long history of integrating new technologies...," said American Express Chief Information Officer Marc Gordon, in a statement. "This collaboration with Ripple and Santander represents the next step forward on our blockchain journey, evolving the way we move money around the world."

Any reader can see that this text is fairly positive in favor of XRP. The positivity isn't exactly through the roof, but the author is conveying a positive sentiment nonetheless. Preprocessing was performed on the text before ranking sentiment with the OTBs. The first step was removing all punctuation and numbers, leaving only the text of the article:

shutterstock ripple seizes the spot with b market cap alongside major developments in bitcoin litecoin and ethereum other altcoins are beginning to gain traction in the broader eye one such coin is ripple xrp a cryptocurrency known for its connection with the banking world disclosure i own some xrp hours ago the price of xrp was earlier this morning pst it hit an increase of percent at the time of this writing the price is coinmarketcap what you need to know about ripple alongside its cryptocurrency ripple operates as a payment network called ripplenet the goal of the platform is to optimize easy transfer of funds to almost any other currency or cryptocurrency in the world in seconds ripple is working with banks and financial institutions to become the premier cryptocurrency of record offering a quick cost effective way to transfer funds globally for example if you wanted to send your friend in italy you could trade for worth of xrp and they could quickly trade that out for euros so why has xrp gone up so quickly the increased activity in bitcoin following cboe launching bitcoin futures trading on dec is generating more activity around the entire cryptocurrency world as the and occasionally when litecoin surpassed it cryptocurrency of course ripple has felt the well ripple effect continued from page shutterstock another explanation is that as the bitcoin craze cools off more people are trading for ripple which some see as a more stable asset i think that markets view xrp as a very stable digital asset so they feel safe parking funds in xrp when they exit other assets if someone wants to get out of btc but doesn t want to necessarily move into fiat he or she moves the value into xrp said miguel vias head of xrp markets at ripple in an interview with coindesk especially as network speeds lag and transaction fees soar on bitcoin ripple may be an appealing trading alternative with its super fast speeds amex partnership unlike many cryptocurrencies whose ethos moves away from traditional banking and financial institutions ripple seeks to use the new technology to optimize how money is moved to that end a recently announced partnership with american express could be driving buzz around xrp ripple will be working with amex to solve liquidity shortfalls in remittances by offering instant blockchain based payments american express has a long history of integrating new technologies said american express chief information officer marc gordon in a statement this collaboration with ripple and santander represents the next step forward on our blockchain journey evolving the way we move money around the world

With all punctuation and numbers removed, a bag of words (B.O.W) is all that is left. Analysis be could perform on this B.O.W, but further preprocessing was performed in the form of lemmatizaiton and stemming. Both methods word tried (not together), and sentiment was ranked. The output from stemming looks like the following:

shutterstock rippl seiz the spot with b market cap alongsid major develop in bitcoin litecoin and ethereum other altcoin are begin to gain traction in the broader eye one such coin is rippl xrp a cryptocurr known for it connect with the bank world disclosur i own some xrp hour ago the price of xrp was earlier thi morn pst it hit an increas of percent at the time of thi write the price is coinmarketcap what you need to know about rippl alongsid it cryptocurr rippl oper as a payment network call ripplenet the goal of the platform is to optim easi transfer of fund to almost ani other currenc or cryptocurr in the world in second rippl is work with bank and financi institut to becom the premier cryptocurr of record offer a quick cost effect way to transfer fund global for exampl if you want to send your friend in itali you could trade for worth of xrp and they could quick trade that out for euro so whi has xrp gone up so quick the increas activ in bitcoin follow cboe launch bitcoin futur trade on dec is generat more activ around the entir cryptocurr world as the and occasion when litecoin surpass it cryptocurr of cours rippl has felt the well rippl effect continu from page shutterstock anoth explan is that as the bitcoin craze cool off more peopl are trade for rippl which some see as a more stabl asset i think that market view xrp as a veri stabl digit asset so they feel safe park fund in xrp when they exit other asset if someon want to get out of btc but doesn t want to necessarili move into fiat he or she move the valu into xrp said miguel via head of xrp market at rippl in an interview with coindesk especi as network speed lag and transact fee soar on bitcoin rippl may be an appeal trade altern with it super fast speed amex partnership unlik mani cryptocurr whose etho move away from tradit bank and financi institut rippl seek to use the new technolog to optim how money is move to that end a recent announc partnership with american express could be drive buzz around xrp rippl will be work with amex to solv liquid shortfal in remitt by offer instant blockchain base payment american express has a long histori of integr new technolog said american express chief inform offic marc gordon in a statement thi collabor with rippl and santand repres the next step forward on our blockchain journey evolv the way we move money around the worl

And the output after performing Lemmatization is:

shutterstock ripple seizes the spot with b market cap alongside major development in bitcoin litecoin and ethereum other altcoins are beginning to gain traction in the broader eye one such coin is ripple xrp a cryptocurrency known for it connection with the banking world disclosure i own some xrp hour ago the price of xrp wa earlier this morning pst it hit an increase of percent at the time of this writing the price is coinmarketcap what you need to know about ripple alongside it cryptocurrency ripple operates a a payment network called ripplenet the goal of the platform is to optimize easy transfer of fund to almost any other currency or cryptocurrency in the world in second ripple is working with bank and financial institution to become the premier cryptocurrency of record offering a quick cost effective way to transfer fund globally for example if you wanted to send your friend in italy you could trade for worth of xrp and they could quickly trade that out for euro so why ha xrp gone up so quickly the increased activity in bitcoin following cboe launching bitcoin future trading on dec is generating more activity around the entire cryptocurrency world a the and occasionally when litecoin surpassed it cryptocurrency of course ripple ha felt the well ripple effect continued from page shutterstock another explanation is that a the bitcoin craze cool off more people are trading for ripple which some see a a more stable asset i think that market view xrp a a very stable digital asset so they feel safe parking fund in xrp when they exit other asset if someone want to get out of btc but doesn t want to necessarily move into fiat he or she move the value into xrp said miguel vias head of xrp market at ripple in an interview with coindesk especially a network speed lag and transaction fee soar on bitcoin ripple may be an appealing trading alternative with it super fast speed amex partnership unlike many cryptocurrencies whose ethos move away from traditional banking and financial institution ripple seek to use the new technology to optimize how money is moved to that end a recently announced partnership with american express could be driving buzz around xrp ripple will be working with amex to solve liquidity shortfall in remittance by offering instant blockchain based payment american express ha a long history of integrating new technology said american express chief information officer marc gordon in a statement this collaboration with ripple and santander represents the next step forward on our blockchain journey evolving the way we move money around the worl

It can be seen that Lemmatization and Stemming both produce different effects. Stemming appears to just cut off the ending of the majority of words, effectively creating words that don’t exist in a dictionary. Lemmatization performs better when trying to recreate a root word. One would expect that the lemmatized text would perform adequately when ranking sentiment. But the reality, as already stated, is that both methods performed poorly using both libraries, Vader Sentiment and TextBlob Sentiment.

Vader Sentiment provides a sentiment score in the form of **{'compound': xxx, 'neg': xxx, 'neu': xxx, 'pos': xxx}**. From the Vader Sentiment documentation:

- **Compound score** is computed by summing the valence scores of each word in the lexicon, adjusted according to the rules, and then normalized between -1 (most extreme negative) and +1 (most extreme positive). This is the most useful metric if you want a single unidimensional measure of sentiment for a given sentence.
- **Positive Sentiment**: compound score >-= 0.5
- **Neutral Sentiment**: compound score -0.5 and compound score < 0.5
- **Negative Sentiment**: compound score <= -0.5

TextBlob Sentiment provides a sentiment score in the form of a namedtuple **Sentiment(polarity = xxx, subjectivity = xxx)**. From the TextBlob documentation:

- **Polarity** score is a float within the range [-1.0, 1.0]
- **Subjectivity** is a float within the range [0.0, 1.0] where 0.0 is very objective and 1.0 is very subjective

For more information on Vader Sentiment see its *documentation*. For more information on TextBlob, see its *documentation*

For the example text, each of the libraries had the following performance:

Sentiment Library	Preprocessing	Sentiment Score
Vader Library	“Words Only”, “Stemmed”	{‘compound’ : 0.9371, ‘neg’ : 0.01, ‘neu’ : 0.932, ‘pos’ : 0.058}
	“Words Only”, “Lemmatize”	{‘compound’ : 0.9895, ‘neg’ : 0.011, ‘neu’ : 0.87, ‘pos’ : 0.12}
TextBlob	“Words Only”, “Stemmed”	{polarity = 0.129785, subjectivity = 0.412247}
	“Words Only”, “Lemmatize”	{polarity = 0.157429, subjectivity = 0.40967}

The Vader compound score is very close to one. This is heavily on the “positive” side of the spectrum. Although the text is overall positive, it shouldn’t rank almost a perfect 1 on the positive scale. Conversely, the TextBlob polarity score is correctly on the positive side. However, it is a little light on the positive ranking. There were no negative sentences within the text and there are more than enough positive sentences to move it away from a near neutral score. The OTB libraries failed to capture the real strength of sentiment in the text.

4.2 *Custom Sentiment Analysis Library*

The failure to correctly characterize an article’s text as negative or positive can be attributed to what is considered positive or negative in everyday speech compared to what is considered positive or negative in regards to cryptocurrency. For example:

*“The **surging** waves came **shooting** up over the sea wall and flooded the city. People were **jumping** in tehir cars to escape the **danger**. Eventually the flood came **down** and all was normal.”*

---

*“Bitcoin was **surging** to new heights today. It has been **shooting** up in price for the past month, **jumping** up higher than everyone’s expectations. Even though it is on the rise, experts fear a bubble may burst, and the price is in **danger** of coming **down**.”*

Though the same words are utilized in both sentences, they provide a totally different meaning/sentiment due to the nature of the subject. Surging, shooting, and jumping helped convey a negative sentiment in the first sentence. But when used in the context of cryptocurrency, they provide a positive connotation. For this exact reason, a custom dictionary of positive and negative words was created to represent common terminology used in crypto world. A random sample of, both, obviously negative and obviously positive texts were extracted. From each, a list of negative and positive keywords were created. These negative and positive word libraries are text files which can be viewed in Appendix C or in my github repository here. With a custom cryptocurrency sentiment dictionary, an algorithm was needed on how to use it.

4.3 *Custom Sentiment Algorithm*

In order to determine what makes a body of text positive or negative, a few assumptions were drawn. A body of text is made up of sentences, so it is safe to assume that when there are more negative sentences than positive sentences, the text overall must be negative. The same can be said of the opposite. Conversely, a sentence is made up of words. Therefore it is safe to assume that when there are more negative words in a sentence than positive, the sentence overall must be negative. Using the cryptocurrency dictionary, a distinction is made between the number of negative words to positive words in a sentence to get its sentiment. Then a relation is made between the number of negative sentences to positive in the text to get its overall sentiment. From these assumptions, not only is the overall sentiment of the text extrapolated, but also the strength of sentiment. With a higher percentage of negative sentences in a text, one can assume a stronger negative sentiment. Again, the same can be said of the opposite.

There are few things to consider however:

- First, it isn’t enough to just say when [**# of negative words > # of positive words**], then sentence is negative. Some sentences are more negative than others. To catch the strength of sentiment, one must also capture strength of positivity or negativity of each sentence. E.g.:

  Bitcoin is surging(+1) today after suffering(-1) losses(-1) all last week. The surge(+1) is umprompted as there have been no major developments out of the Bitcoin camp. Investors don't mind the sudden upswing(+1). Smart investors however know not to get too excited(+1) as the price will more than likey experience a negative(-1) correction(-1) Some experts believe that not only will the price drop(-1), but it will plummet(-1) to yearly lows(-1) leaving today's investors in a world of hurt(-1).

Analyzing sentence by sentence:

1. **Sentence one: (+1) + (-1) + (-1) = -1**
2. **Setnence two: (+1) = 1**
3. **Sentence three: (+1) = 1**
4. **Sentence four (+1) + (-1) + (-1) = -1**
5. **Setence five: (-1) + (-1) + (-1) + (-1) = -4**

Summing all sentence scores nets a grand total of -4. One could stop here and perform the sentiment analysis like this, but this wouldn't accurately model uneven text. For example, what if the first sentence contained 10 negative keywords? Meanwhile, the rest of the sentences were positive by containing only one positive keyword in each. That would net a score of (-10 + 4) = -6. This would violate the initial assumption "if there are more positive or negative sentences in a text, then that text must be positive or negative respectively".

- Second, sentences need to be proportioned in order to reduce the effect of outliers. Outliers being those sentiences with an unusual amount of keywords. The easiest way to do this is by:

  1. Calculating the percentage of negative and positive sentences in the text
  2. Summing the score of all negative sentences and summing the score of all positive sentences
  3. Multiplying the score of all negative sentences by the percentage of negative sentences and the score of all positive sentences by the percentage of positive sentences.

  Using the five sentences above, there were (3/5) negative sentences with a total negative point score of (-6) and (2/5) positive sentences with a total positive score of (2).

  - **Strength of negative sentiment = (3/5) x (-6) = -3.6**
  - **Strength of positive sentiment = (2/5) x (2) = 0.8**

  This upon initial inspection seems reasonable. The overall sentiment of the text is strongly negative, and our sentiment score properly reflects that with a -3.6. Also, there are some light positive elements in the example text and they are represented by a light positive sentiment score. But how strong [or weak] is -3.6 compared to other texts? What is the maximum possible score on both ends of the spectrum? How will a strongly positive text with very few sentences score compare to a moderately positive text with dozens of sentences? These questions bring about the next consideration.

- Lastly, the scores must be properly normalized. When writing an article, the author decides how strong or weak they want to make the article by placing more or less of the emphasis words (keywords) in the article. For conversation sake, let's say you are the author of the article above. Your editor has placed a 12 emphasis word limit on you. So in 12 words you must get as much

positive and negative sentiment into the text that you want to convey. Therefore, in order to normalize the sentiment scores, we can divide them by the total number of emphasis words in the text

- **Strength of negative sentiment = (-3.6) / 12 = -0.3**
- **Strength of positive sentiment = (0.8) / 12 = 0.06**

When strength of positive sentiment is close to +1, a strong positive sentiment is present. Conversely, when strength of negative sentiment is close to -1, the stronger the negative sentiment. Close proximity to zero for either number represents a lesser amount of positive/negative sentiment in the text (more neutral). When the two numbers are close to equal (but opposite) then the text can also be deemed neutral overall. The algorithm constructed to perform the custom sentiment analysis can be seen in Appendix D

4.4 *Custom Sentiment Performance*

Reverting back to the randomly selected text, the performance of the custom sentiment analyzer was tested. Remember that Vader Sentiment gaged the text to be extremely strong in positive sentiment, with a score of almost a perfect 1. Although the text was positive, to rank it a near one was a gross over estimation. The TextBlob analysis failed to grasp the strength of positivity in the text with a near neutral score of 0.15. The custom sentiment library was designed to give a score in the form of a namedtuple; **Sentiment(Strength_of_Positivity = xxxx, Strength_of_Negativity = xxxx)**. When analyzing the random text, the algorithm returned a score of Sentiment(Strength_of_Positivity=0.5454, Strength_of_Negativity=0.0).

Utilizing the custom sentiment algorithm, the score is a more reasonable 0.54. Remember that the closer to 1 a score is, the stronger in positivity it is. Conversely the closer to 0 it is, the weaker in positivity it is (neutral). The same logic applies to the negativity scale, -1 to 0. This score appears to have capture the neutrality contained in the text (all the neutral sentences and background text) while providing a seemingly accurate strength of positivity score in that the score is positive, but not to the overly extreme.

4.5 *Sentiment Aggregation*

Analysis was then performed on all scraped text in order to quantify their positivity/negativity. Analysis was performed on the articles' headlines as well as their texts. However, only the text sentiment was explored initially. The headline sentiment was reserved in the event more data was needed to draw conclusions from.

Date	Date	Positive Sentiment	Negative Sentiment
2013-12-19	2013-12-19	0.140625	-0.020833
2017-05-16	2017-05-16	0.187856	-0.034156
2017-05-26	2017-05-26	0.461538	-0.004525
2017-06-07	2017-06-07	0.181159	-0.007246
2017-06-21	2017-06-21	0.175000	-0.056250
2017-07-20	2017-07-20	0.250000	-0.083333
2017-07-21	2017-07-21	0.615385	0.000000
2017-08-02	2017-08-02	0.181185	-0.074332
2017-08-04	2017-08-04	0.168038	-0.105758
2017-08-07	2017-08-07	0.071023	-0.102273
2017-08-09	2017-08-09	0.083333	-0.083333
2017-08-11	2017-08-11	0.097222	-0.069444
2017-08-13	2017-08-13	0.068108	-0.190270
2017-08-15	2017-08-15	0.700000	0.000000
2017-08-16	2017-08-16	0.129808	-0.024038
2017-08-17	2017-08-17	0.357143	0.000000
2017-08-18	2017-08-18	0.114035	-0.193860
2017-08-21	2017-08-21	0.332331	-0.006015
2017-08-22	2017-08-22	0.046154	-0.242308
2017-08-22	2017-08-22	0.444444	0.000000

Figure 4:1 - Quantifying Articles' Sentiment

The first issued that had to be resolved with this data was how to score dates that had more than one sentiment score. To have multiple scores tied to one date meant that there were several articles written and parsed for that day. For example, September 8th had 4 different articles written that day.

Date	Date	Positive Sentiment	Negative Sentiment
2017-09-08	2017-09-08	0.368286	-0.002558
2017-09-08	2017-09-08	0.411765	0.000000
2017-09-08	2017-09-08	0.104956	-0.174927
2017-09-08	2017-09-08	0.104956	-0.174927

Figure 4:2 - Multiple Sentiment Scores for a Single Day

There were a few different ways to handle the situation. Ultimately, aggregates of sentiment averaged, sentiment root mean square (RMS), and sentiment summed were calculated in order to later analyze the best metric. RMS is the square root of the mean of the values.

Secondly, in conjunction with intrinsic values of positive and negative sentiment for each article, an "overall" sentiment needed to be established. This "overall" needed to be representative of the magnitude of both intrinsic scores. In accordance with how the multiple articles were aggregated, the same methods were applied to the "overall sentiment". The Positive and Negative Score for each article was averaged, RMS, and summed with each other in order to produce an "overall sentiment" for each article. Again, all methods were applied in order to later analyze the best metric. The resulting time series data frame was refined to 189 rows and 9 columns.

Date	Positive Sentiment Averaged	Negative Sentiment Averaged	Overall Sentiment Averaged	Positive Sentiment Summed	Negative Sentiment Summed	Overall Sentiment Summed	Positive Sentiment RMS	Negative Sentiment RMS	Overall Sentiment RMS
2013-12-19	0.140625	-0.020833	0.119792	0.140625	-0.020833	0.119792	0.140625	-0.020833	0.119792
2017-05-16	0.187856	-0.034156	0.153700	0.187856	-0.034156	0.153700	0.187856	-0.034156	0.153700
2017-05-26	0.461538	-0.004525	0.457014	0.461538	-0.004525	0.457014	0.461538	-0.004525	0.457014
2017-06-07	0.181159	-0.007246	0.173913	0.181159	-0.007246	0.173913	0.181159	-0.007246	0.173913
2017-06-21	0.175000	-0.056250	0.118750	0.175000	-0.056250	0.118750	0.175000	-0.056250	0.118750
2017-07-20	0.250000	-0.083333	0.166667	0.250000	-0.083333	0.166667	0.250000	-0.083333	0.166667
2017-07-21	0.615385	0.000000	0.615385	0.615385	0.000000	0.615385	0.615385	-0.000000	0.615385
2017-08-02	0.181185	-0.074332	0.106852	0.181185	-0.074332	0.106852	0.181185	-0.074332	0.106852
2017-08-04	0.168038	-0.105758	0.062280	0.168038	-0.105758	0.062280	0.168038	-0.105758	0.062280
2017-08-07	0.071023	-0.102273	-0.031250	0.071023	-0.102273	-0.031250	0.071023	-0.102273	-0.031250
2017-08-09	0.083333	-0.083333	0.000000	0.083333	-0.083333	0.000000	0.083333	-0.083333	0.000000
2017-08-11	0.097222	-0.069444	0.027778	0.097222	-0.069444	0.027778	0.097222	-0.069444	0.027778
2017-08-13	0.068108	-0.190270	-0.122162	0.068108	-0.190270	-0.122162	0.068108	-0.190270	-0.122162
2017-08-15	0.700000	0.000000	0.700000	0.700000	0.000000	0.700000	0.700000	-0.000000	0.700000
2017-08-16	0.129808	-0.024038	0.105769	0.129808	-0.024038	0.105769	0.129808	-0.024038	0.105769

Figure 4:3 - Aggregated Sentiment

5. *DATA EXPLORATION*

After initial acquisition and wrangling of the two data types, historical price and sentiment data, two separate time series data frames was constructed. The sentiment data consisted of 360 rows representing the 360 scraped articles that span 14 months. The historical data, data frame consisted of 310 rows of data spanning 11 months. After performing aggregation, both data frames were merged by performing a full outer join. The various columns and their statistics are the following:

	Positive Sentiment Averaged	Negative Sentiment Averaged	Overall Sentiment Averaged	Positive Sentiment Summed	Negative Sentiment Summed	Overall Sentiment Summed	Positive Sentiment RMS	Negative Sentiment RMS	Overall Sentiment RMS
count	309.000000	309.000000	309.000000	309.000000	309.000000	309.000000	309.000000	309.000000	309.000000
mean	0.123342	-0.065281	0.058061	0.274749	-0.107171	0.167578	0.130561	-0.071082	0.059478
std	0.148384	0.099289	0.181044	0.496202	0.170689	0.468065	0.157067	0.103101	0.185235
min	0.000000	-0.550725	-0.521739	0.000000	-1.083815	-0.898197	0.000000	-0.550725	-0.521739
25%	0.000000	-0.102273	0.000000	0.000000	-0.180000	0.000000	0.000000	-0.116667	0.000000
50%	0.071314	-0.008929	0.000000	0.074435	-0.017798	0.000000	0.071970	-0.011224	0.000000
75%	0.209695	0.000000	0.142429	0.315789	0.000000	0.203947	0.227273	0.000000	0.144622
max	0.750000	0.000000	0.750000	3.204048	0.000000	2.873581	0.750000	0.000000	0.750000

	XRP_Change	XRP_High	XRP_Low	XRP_Open	XRP_Price	XRP_Vol	BTC_Change	BTC_High	BTC_Low	BTC_Open	BTC_Price	BTC_Vol	ETH_Change	ETH_High	ETH_Low	ETH_Open	ETH_Price
count	309.000000	309.000000	309.000000	309.000000	309.00000	309.000000	309.000000	309.000000	309.000000	309.000000	309.000000	309.000000	309.000000	309.000000	309.000000	309.000000	309.000000
mean	0.843204	0.683485	0.585181	0.637424	0.63840	18410.762006	0.470485	8702.078511	7928.982751	8350.134013	8365.755631	50.832265	0.548058	589.918770	532.729288	563.454466	564.788835
std	9.801673	0.576468	0.459148	0.521588	0.52126	21687.769818	5.539742	3809.798327	3305.489705	3609.441163	3595.704879	29.576107	6.143635	284.895277	248.454400	269.453083	268.695537
min	-30.960000	0.160800	0.145500	0.152500	0.00000	945.460000	-18.440000	2757.800000	2615.800000	2702.000000	2702.000000	11.160000	-20.180000	228.200000	198.000000	200.810000	218.000000
25%	-3.970000	0.229000	0.203000	0.214300	0.21480	5580.000000	-2.370000	5803.200000	5450.000000	5636.800000	5694.000000	29.540000	-2.500000	319.780000	299.000000	309.750000	309.820000
50%	0.000000	0.620000	0.557000	0.591050	0.59925	11860.000000	0.400000	8350.700000	7839.090000	8097.300000	8097.300000	44.800000	0.250000	531.200000	492.010000	511.280000	511.990000
75%	3.230000	0.921090	0.800000	0.872430	0.87243	20690.000000	3.610000	10540.000000	9583.900000	10144.000000	10145.000000	63.110000	3.900000	768.010000	692.280000	736.770000	736.770000
max	81.150000	3.280000	2.560000	2.770670	2.77000	143370.000000	22.720000	19891.000000	18622.000000	19210.000000	19187.000000	192.760000	23.220000	1423.200000	1264.400000	1379.900000	1380.000000

Figure 5:1 - Resulting data after wrangling and manipulation The tables above show the statistics for each column.

First thing to notice is the magnitude of each column. Sentiment is as low as 0 whereas XRP Volume is as high as 143,370. Because of this, eventually all data was scaled. Secondly, XRP was as low as $0.15 cents and reached a maximum of $3.28. A $1,000 investment at that initial price point would've netted a return of almost $22,000 at XRP's highest price point. Trading volume for XRP has also been pretty volatile swinging from 1,460,000 to as high as 143,370,000.

5.1 *XRP Correlation Analysis*

Since XRP was the dependent variable of of interest, a correlation analysis was performed on XRP price versus all other variables (columns of the data frame). Initially it was hypothesized that Bitcoin (BTC) would have the highest correlation with XRP price since the tendency of the whole market always seems to follow Bitcoin. The following correlation matrix highlights the results of that analysis.

Positive Sentiment Averaged	0.255892
Negative Sentiment Averaged	-0.168991
Overall Sentiment Averaged	0.117050
Positive Sentiment Summed	0.422747
Negative Sentiment Summed	-0.324336
Overall Sentiment Summed	0.329885
Positive Sentiment RMS	0.272192
Negative Sentiment RMS	-0.201272
Overall Sentiment RMS	0.118773
XRP_Change	0.064863
XRP_High	0.977217
XRP_Low	0.968592
XRP_Open	0.966195
XRP_Price	1.000000
XRP_Vol	0.070674
BTC_Change	-0.082697
BTC_High	0.701639
BTC_Low	0.706575
BTC_Open	0.710609
BTC_Price	0.701040
BTC_Vol	0.019197
ETH_Change	0.068052
ETH_High	0.836097
ETH_Low	0.828926
ETH_Open	0.820749
ETH_Price	0.841482
ETH_Vol	0.140664

Figure 5:2 - XRP Price Correlation Matrix. A score close to one represents high positive correlation. A score close to -1 represents a high negative correlation.

The closer to 1 a value is, a strong positive correlation is prevalent. Instantly it is revealed that, although BTC did have a high correlation factor with XRP price at 0.701, Ethereum proved to be the better metric at 0.842. Conversely, XRP’s own previous day’s High and Low values lend the strongest insight into next day price correlation at 0.977 and 0.969 respectively.

Visualizing the Bitcoin-XRP and Ethereum-XRP pair plots provided further explanation into the correlation values.

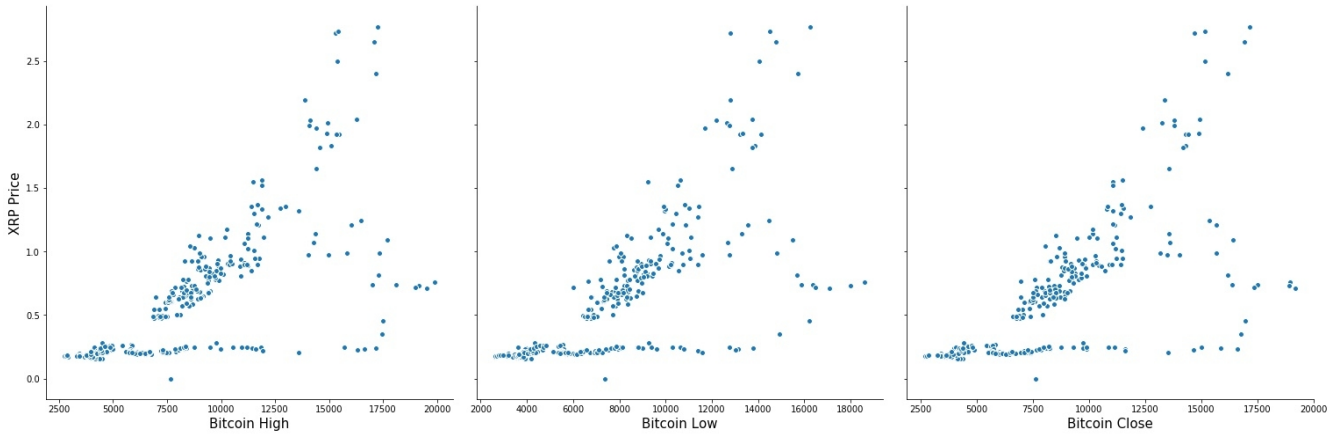


Figure 5:3 - Bitcoin vs XRP Price correlation pair plots

When plotting Bitcoin prices (High, Low, Close) vs XRP price, the positive correlation is clear. However it is now evident why BTC didn't warrant as high of a correlation. There seems to have been a stretch in which BTC greatly increased in value while XRP remained pretty stagnant. This goes against the original notion that the market follows BTC closely.

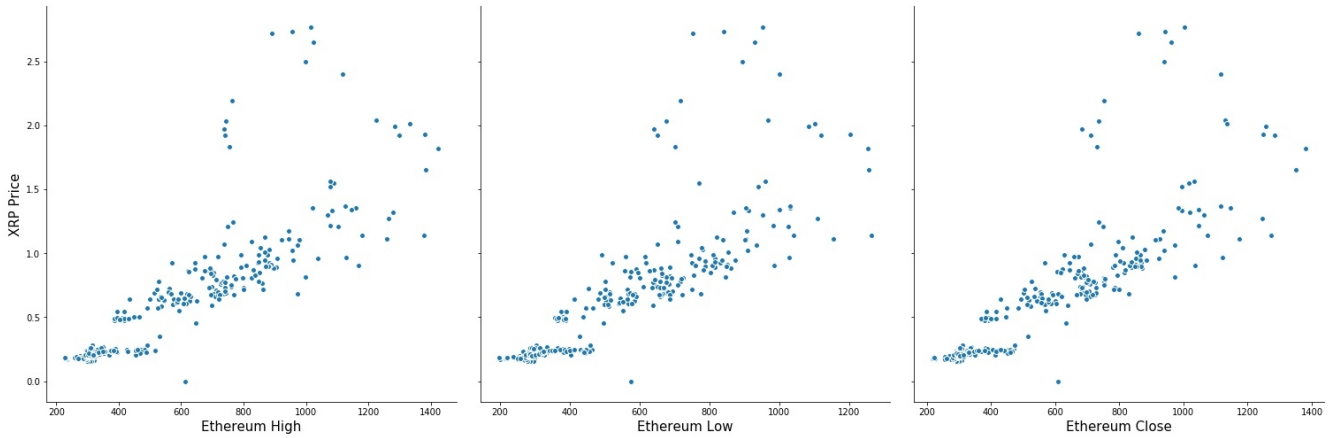


Figure 5:4 - Ethereum vs XRP Price correlation pair plots

After plotting the pair plot between ETH and XRP, it is revealed that the two almost mirrored each other 1:1. Only after reaching a certain price point did XRP spike exponentially while ETH appears to have continued a linear trend. Consequently, the notion that ETH is a better indicator of XRP price negates the possibility of BTC being the trend to watch.

5.2 *Time Series Analysis*

As previously stated, XRP is usually bought with either BTC or ETH on the majority of exchanges. It wouldn't have been considered presumptuous to assume that XRP would follow closely

in trend of either. This was confirmed in the previous correlation analysis. When plotting all 3 time series, the correlation characteristics fully reveal themselves. Before plotting all 3 prices, they were scaled and normalized to 1. This was done because Bitcoin was almost $20,000 at its high end and XRP was at $0.14 cents at the low end.

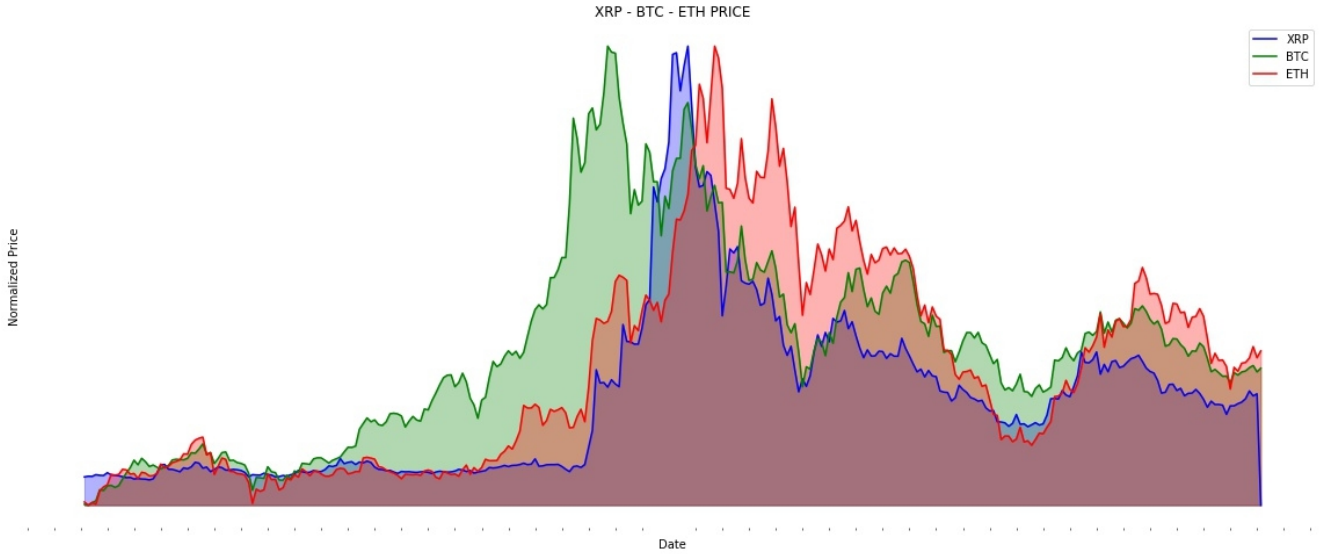


Figure 5:5: XRP, BTC, ETH time series

Apparently XRP trails price fluctuations in ETH by a day or two. However both coins can trail BTC by up to a week or more. This explains the flat line correlation trend between XRP and BTC in the pair plot. It is easily seen that BTC did indeed shoot up in price while both XRP and ETH remained pretty flat. This is interesting because not only, as stated earlier, the market as whole tends to follow BTC, but also the trading volume of XRP was much greater than both coins combined. BTC and ETH traded in the hundreds of thousands whereas XRP traded in the millions. Although this could simply be due to the price point differences. BTC average price was $8300, ETH average was approximately $550 and XRP average hovered around $0.62. Therefore a dollar has more buying power with respect to XRP. But one could speculate with volume in the hundreds of millions, demand is high therefore price should be as well.

In hopes of revealing further trend characteristics, XRP price was plotted against the full High, Low, Close (HLC) characteristics for BTC and ETH.

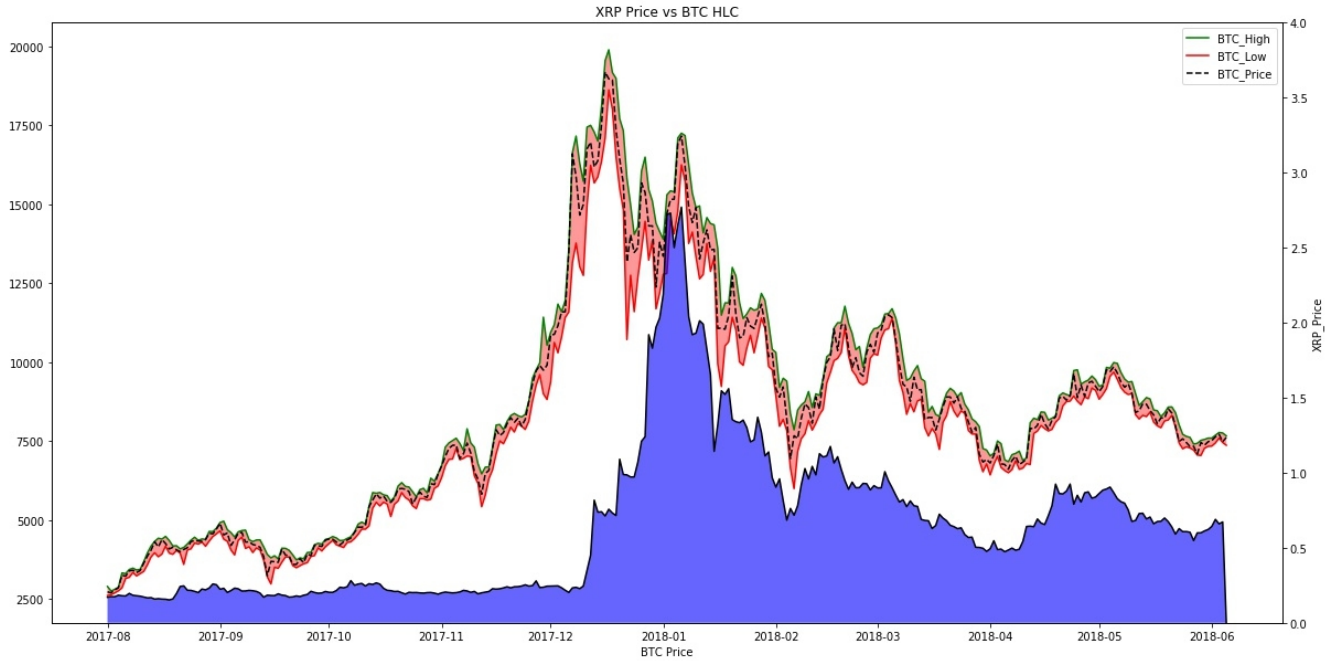


Figure 5:6 - Bitcoin High/Low/Close vs XRP

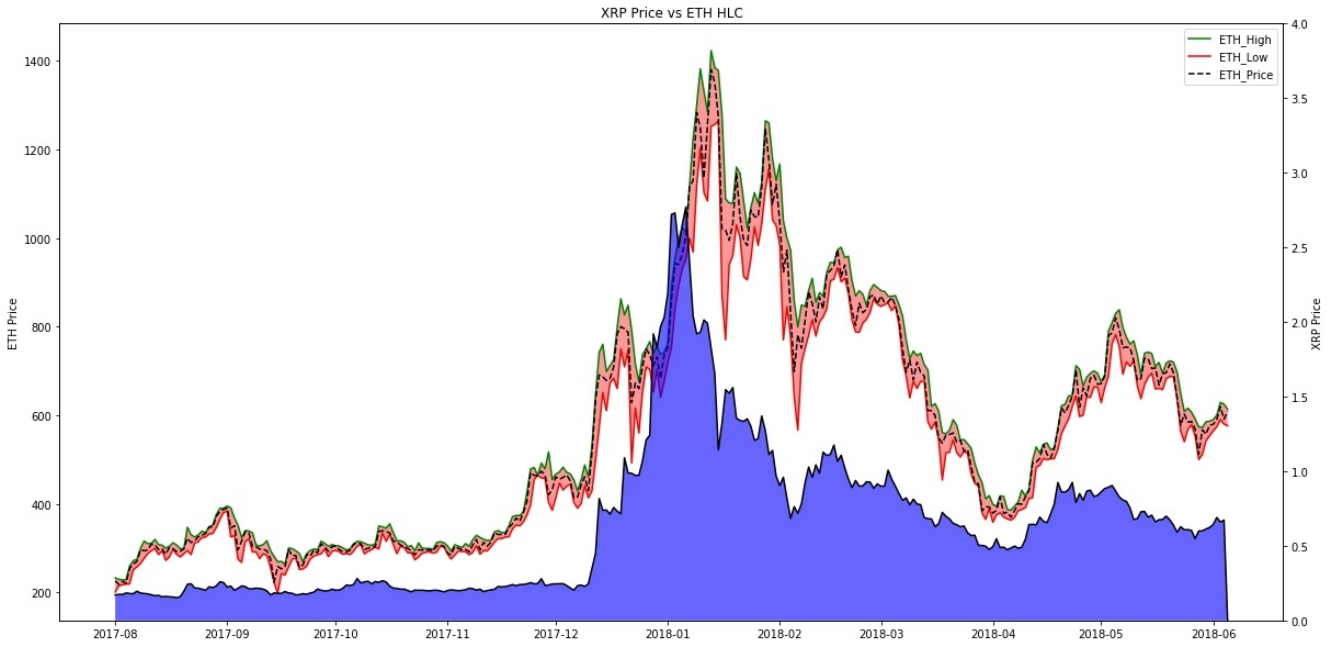


Figure 5:7 - Ethereum High/Low/Close vs XRP

Unfortunately, no further trend was revealed when breaking down BTC and ETH to their respective constituents of HLC. One thing to note however is that both ETH and XRP really exploded after BTC fell from it's highest high. This could be interpreted as investors rode the BTC surge until it began to fall from it's highs. Then investors pivoted to other coins such as XRP and ETH. That could potentially be another trend to watch for, the sudden drop in particular coins resulting in the increase of other coins.

5.3 *XRP Response to Sentiment*

After the time series analysis, sentiment was explored and how price responded to it. Remember that there were three aggregates of sentiment created {summed, averaged, RMS}. Since RMS is a form of averaging, their values were somewhat close to each other. Therefore it was a bit superfluous to analyze both at the same time. Summed and RMS were primarily focused on first. Had RMS appeared to show more insight than summed, then the averaged statistic would have been investigated further. Conflicting narratives were discovered while extracting the linear regression line for sentiment and price then plotting the bivariate distributions.

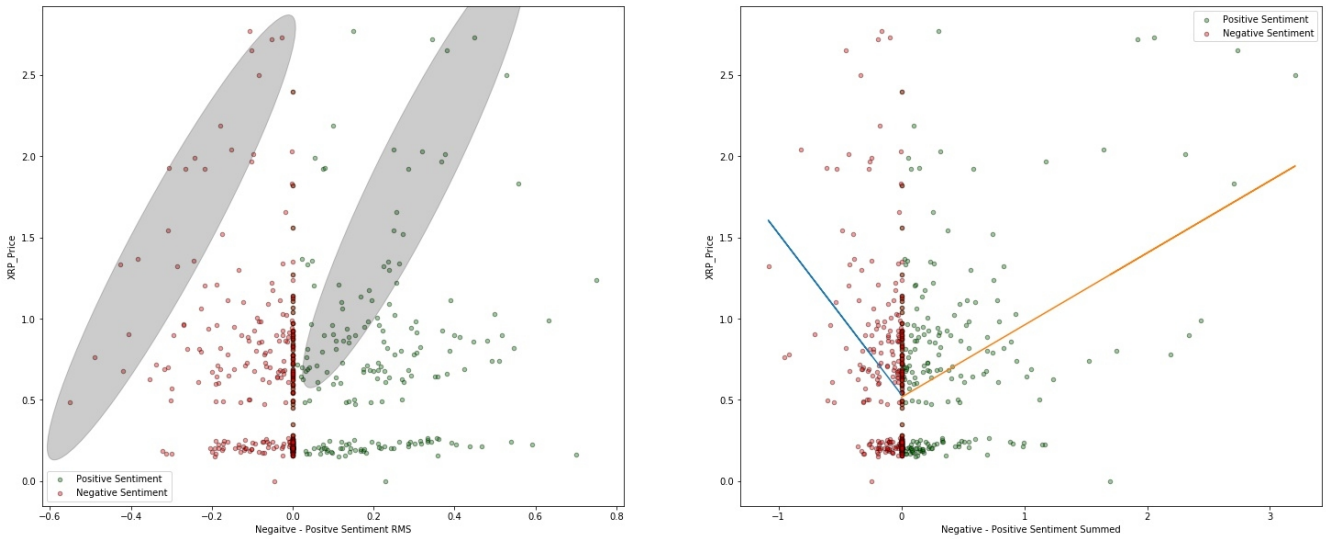


Figure 5:8 - Sentiment vs XRP Price. The figure on the left represents sentiment RMS. The figure on the right represent sentiment summed.

Interestingly enough, visually there was a trend that we would expect to see in the gray ovals for the bivariate distribution on the left. The trend can also be seen in the distribution on the right, but is only highlighted in the left. As the negative sentiment increases in the negative direction, there is somewhat of a trend of price decreasing. With increasing positive sentiment, the XRP price is increasing. However, due to the clusters of negative and positive points around the lowest XRP Price, the actual linear regression line is being affected. It's producing a trend we wouldn't expect to see as presented in the distribution on the right (with increasing negative sentiment, increasing XRP Price). The regression line is displayed only in the figure on the right. But, the same trend line exists on the distribution of price and sentiment RMS (left figure). Qualitatively examining both distributions, it is clear that the RMS statistic might be too sensitive to change and isn't very representative of how sentiment should be affecting price. Examining the figure on the left, a positive sentiment of 0.5 can result in a price below $0.50 cents or as high as $2.60. Utilizing Summed Sentiment, a sentiment of 3 should only, and does translate to a high XRP Price value. At this point it was pretty clear that Sentiment Summed was the better metric, but further analysis was performed.

Both positive and negative sentiment for each statistic was plotted as a time series versus price.

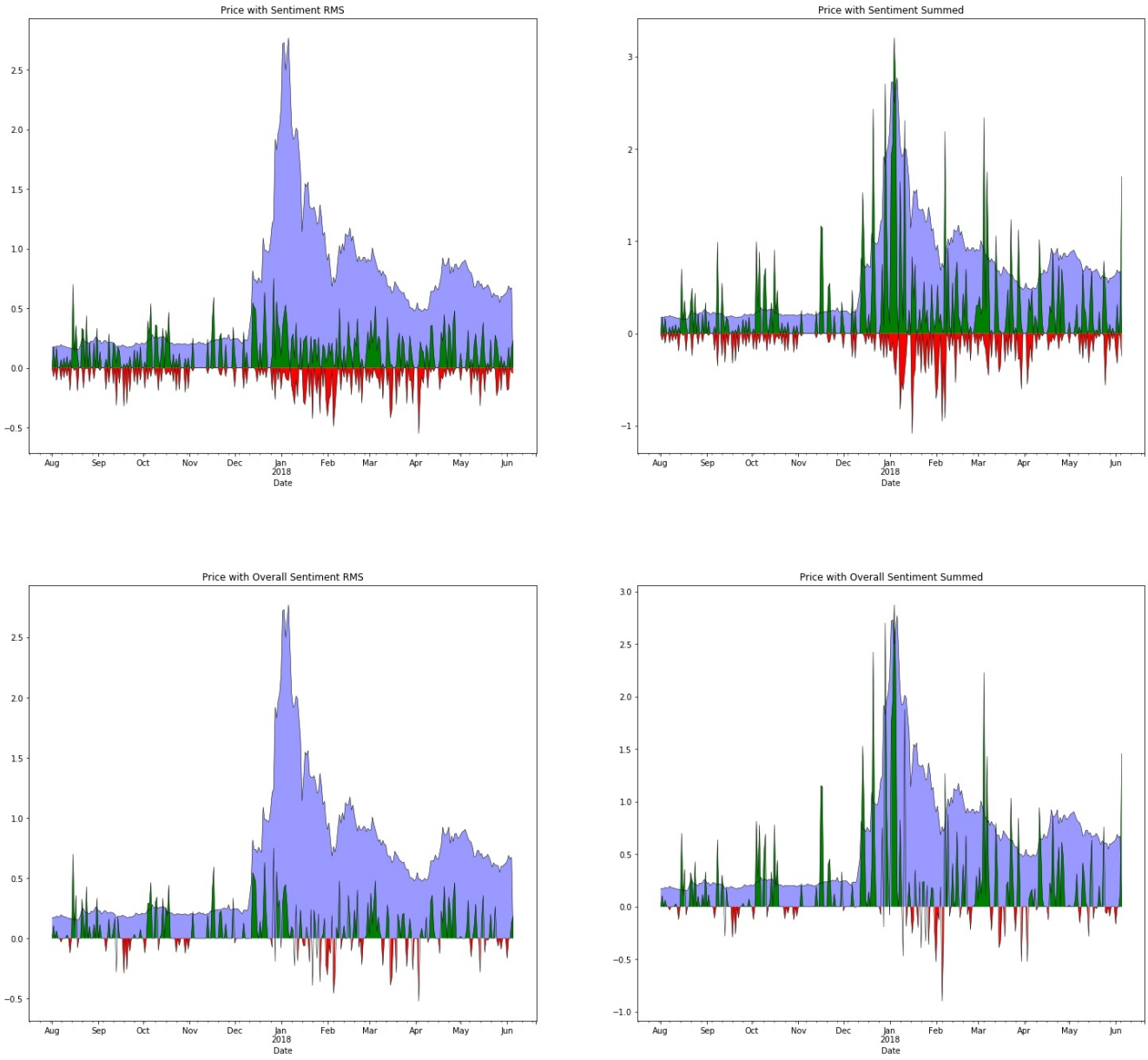


Figure 5:9 - XRP Price and Sentiment RMS time series (Top Left). XRP Price and Sentiment Summed time series (Top Right). XRP Price and Overall Sentiment RMS (Bottom Left). XRP Price and Overall Sentiment Summed (Bottom Right).

Again, it is shown that Sentiment RMS doesn't explain XRP price shifts like Sentiment Summed does. At XRP's highest Price point, Sentiment RMS has a value no higher than when XRP's price point was close to it's lowest. Sentiment Summed however, does reflect a huge leap in XRP Price at the same instance Sentiment explodes.

One thing to note is that these sentiments are only day to day sentiments. But when an investor is investing, they wouldn't only look at articles on the day they were investing. They would research days, and sometimes weeks prior to determine sentiment and performance. Therefore, a day’s sentiment should also bear weight of the previous days' sentiment as well. This will not only help smooth out the

sentiment curve, but correct all 0 values of sentiment. A value of 0 can simply mean that no article was written on that day. If There had been extremely high sentiment for the past week, but today there were no articles written on XRP, that doesn't mean sentiment on the coin has dropped. This just means that no article was written today on the coin. This was accounted for.

A 5 day rolling average for sentiment was performed. When averaging the values however, yesterday's value being averaged into today would be it's original value, not it's rolling average value from when the 5 day average was performed on it. For example, when performing a 3 day rolling average:

- *Day 20 - Original Value: 2* | Three - *Day Average: 2.7*
- Day 21 - Original Value: 1 | Three - Day Average: 2.5
- Day 23 - Original Value: 4 | Three -Day Average: 3.5

Notice from day 3, it got its 3 day average from the original values of the previous 2 days, not their 3 Day Average. The effect of performing a rolling average on sentiment can be see below.

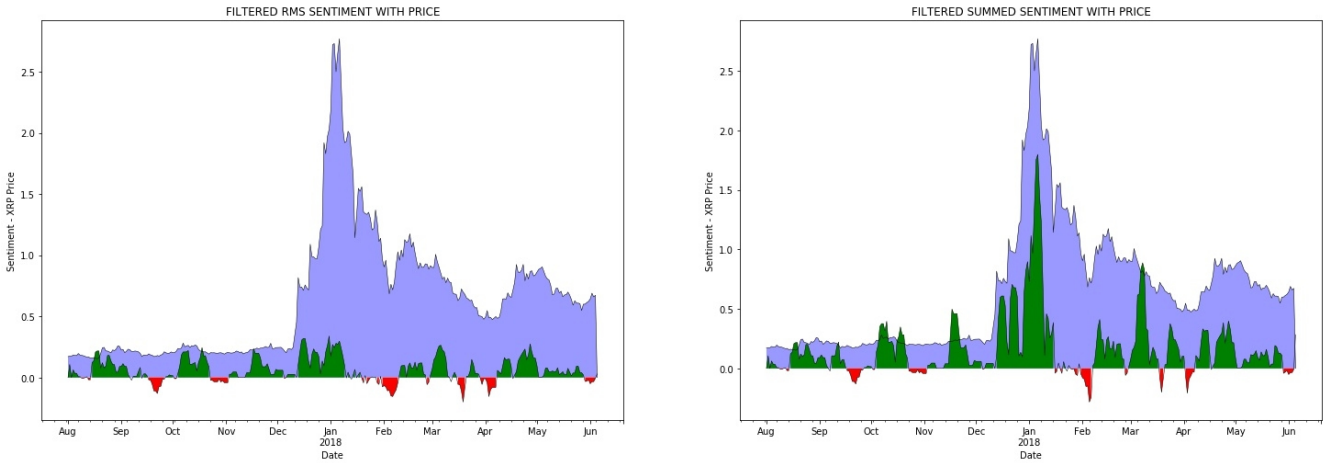


Figure 5:10 - Sentiment rolling average time series analysis

At this point, it was safe to assume that the Sentiment RMS and Sentiment averaged statistics could be ignored. They're not translating very well to the change that XRP experienced.

With the sentiment now represented as a 5 day average, the curve is smoothed while still generalizing to the XRP Price. By performing this transformation on the sentiment, the correlation coefficient increased to 0.578 from the original value of 0.36. This is a 61% improvement. Trading volume was examined next.

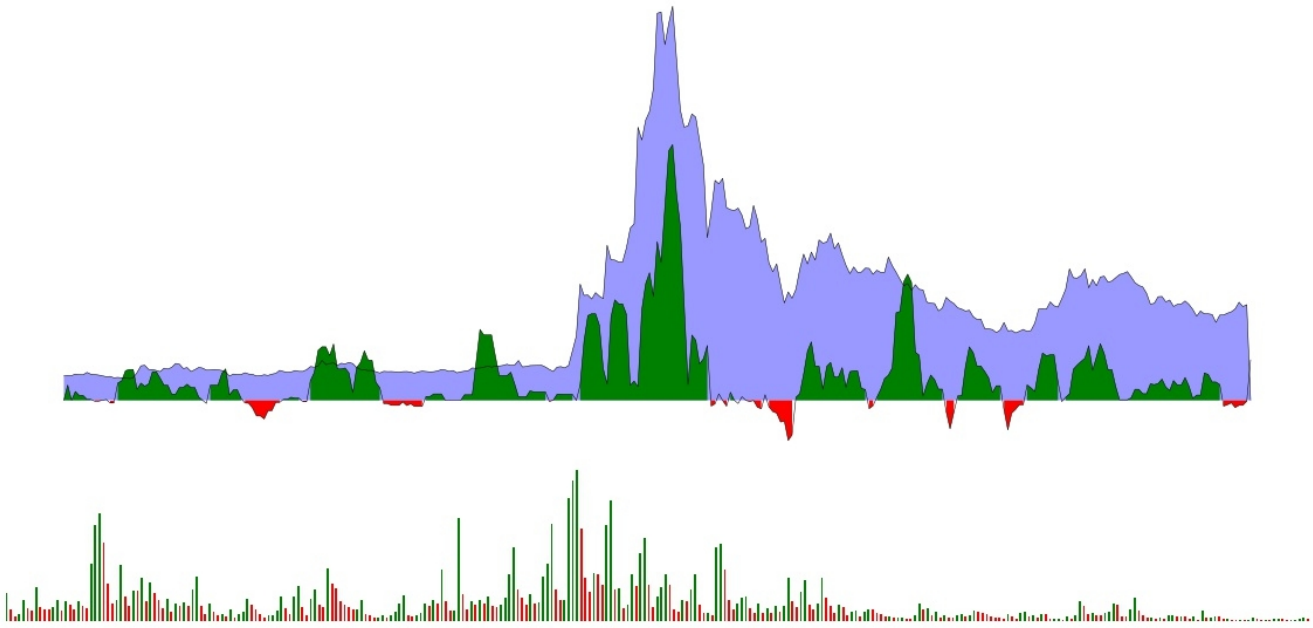


Figure 5:11 - XRP Price historical trading values with volume

It was apparent that a huge spike in volume almost always preceded a spike in sentiment. Especially when looking at XRP's first huge jump in price. Volume spiked to it's highest level the day before. The next day, Sentiment also spiked to higher values. Take note that the highest trading volume didn't happen at XRP's highest price point. In actuality, trading volume was in a declining pattern as XRP was spiking out. This is a reasonable trend as one could speculate that prices shot up, people weren't buying as much with the assumption they missed the optimum buy in point. Or they were waiting for a price correction. Therefore it appears that sentiment was the sole driver for the price spike to its highest point since volume was in a decline. Some investors saw value and continued to buy at higher price points, while the majority of all other investors were retreating

5.4 *Price Performance*

Another driver in an investor's decision making process is not day to day performance. Weekly and monthly performance might be of more interest. As sentiment was manipulated to account for past days' performance into current day statistics, price change percentage was also transformed. The data extracted online provided a day to day price change percentage. Therefore a 5, 10, 15, 20 and 30 day performance statistics was calculated from the original data.

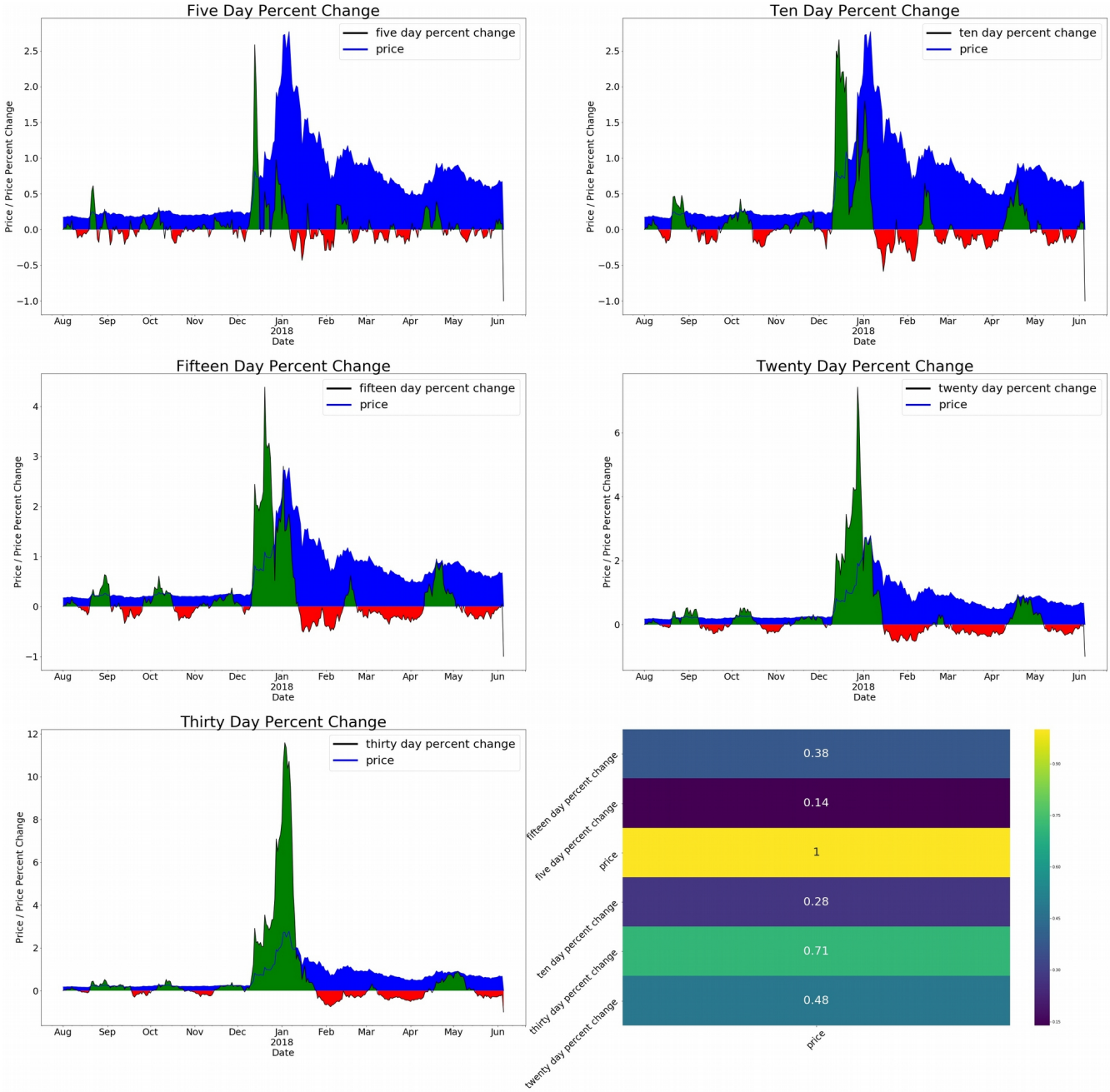


Figure 5:12 - XRP Price performance at various intervals. Each performance trend was correlated to next day XRP price with 30 day price performance achieving the highest coefficient

It appears that the 30 day performance metric correlates very well XRP Price. This makes sense in that the typical investor would get more insight from long term trends than short day price swings. With a correlation of 0.71 at the 30 day performance metric, a long term insight is always good to have. However a good investor shouldn't lose sight of the short term. For instance, someone more interested in shorting the investment, instead of being long could've potentially made a 54% return on investment on August 22nd within a 5 day span. Whereas an investor going long would've had a ROI of only 35% in a 30 day span. 35% is nothing to scoff at. But at 6x the wait of the short term investor, and a deficit of of almost 20% ROI, going long shouldn't be the only focus.

10 and 30 day XRP price performance metrics were merged with the original data and a correlation matrix was composed.

	ten day percent change	thirty day percent change
Overall Sentiment Summed	0.324272	0.459191
BTC_High	0.456819	0.537852
BTC_Low	0.470358	0.530409
BTC_Open	0.450775	0.538036
BTC_Price	0.466798	0.539930
RMS_box_filter	0.512933	0.495004
Summed_box_filter	0.419707	0.759169
Averaged_box_filter	0.512803	0.490454
ten day percent change	1.000000	0.561450
thirty day percent change	0.561450	1.000000

The correlation matrix above was filtered to reveal only those correlations greater than 0.3.
The 30 day performance statistic performed better than the 10 day performance statistic. Therefore, a comparison was drawn between XRP 30 day performance and XRP next day price, to examine a better metric to predict; percent change or an actual price.

	XRP_Price	thirty day percent change
Positive Sentiment Summed	0.422747	0.472139
Overall Sentiment Summed	0.329885	0.459191
XRP_High	0.977217	0.684631
XRP_Low	0.968592	0.635442
XRP_Open	0.966195	0.639066
XRP_Price	1.000000	0.710211
BTC_High	0.701639	0.537852
BTC_Low	0.706575	0.530409
BTC_Open	0.710609	0.538036
BTC_Price	0.701040	0.539930
ETH_High	0.836097	0.311696
ETH_Low	0.828926	0.296715
ETH_Open	0.820749	0.284864
ETH_Price	0.841482	0.318974
Summed_box_filter	0.578319	0.759169
thirty day percent change	0.710211	1.000000

XRP Price was still the stronger dependent variable to predict with the given independent variables. Again, the above correlation matrix was filtered to reveal only those variables that have a coefficient greater than 0.3. Still, trying to predict an exact price might be too difficult. A little more reasonable might be predicting a price range.

5.5 Price Range or Exact Price

Correlating exact prices might have been too specific. As the next trial method, XRP Price was cut into 50 different ranges that are approximately $0.05 cents in range. The ranges were labeled 0 – 49

```
Price Range 0:[0.15 - 0.20)
Price Range 1:[0.20 - 0.26)
Price Range 2:[0.26 - 0.31)
Price Range 3:[0.31 - 0.36)
Price Range 4:[0.36 - 0.41)
Price Range 5:[0.41 - 0.47)
Price Range 6:[0.47 - 0.52)
Price Range 7:[0.52 - 0.57)
Price Range 8:[0.57 - 0.62)
Price Range 9:[0.62 - 0.68)
Price Range 10:[0.68 - 0.73)
Price Range 11:[0.73 - 0.78)
Price Range 12:[0.78 - 0.83)
Price Range 13:[0.83 - 0.89)
Price Range 14:[0.89 - 0.94)
Price Range 15:[0.94 - 0.99)
Price Range 16:[0.99 - 1.04)
Price Range 17:[1.04 - 1.09)
Price Range 18:[1.09 - 1.15)
Price Range 19:[1.15 - 1.20)
Price Range 20:[1.20 - 1.25)
Price Range 21:[1.25 - 1.30)
Price Range 22:[1.30 - 1.36)
Price Range 23:[1.36 - 1.41)
Price Range 24:[1.41 - 1.46)
Price Range 25:[1.46 - 1.51)
Price Range 26:[1.51 - 1.57)
Price Range 27:[1.57 - 1.62)
Price Range 28:[1.62 - 1.67)
Price Range 29:[1.67 - 1.72)
Price Range 30:[1.72 - 1.78)
Price Range 31:[1.78 - 1.83)
Price Range 32:[1.83 - 1.88)
Price Range 33:[1.88 - 1.93)
Price Range 34:[1.93 - 1.98)
Price Range 35:[1.98 - 2.04)
Price Range 36:[2.04 - 2.09)
Price Range 37:[2.09 - 2.14)
Price Range 38:[2.14 - 2.19)
Price Range 39:[2.19 - 2.25)
Price Range 40:[2.25 - 2.30)
Price Range 41:[2.30 - 2.35)
Price Range 42:[2.35 - 2.40)
Price Range 43:[2.40 - 2.46)
Price Range 44:[2.46 - 2.51)
Price Range 45:[2.51 - 2.56)
Price Range 46:[2.56 - 2.61)
Price Range 47:[2.61 - 2.67)
Price Range 48:[2.67 - 2.72)
Price Range 49:[2.72 - 2.77)
```

Implementing this method did reduce price resolution, but when overlaying price ranges with actual price, momentum is still conserved

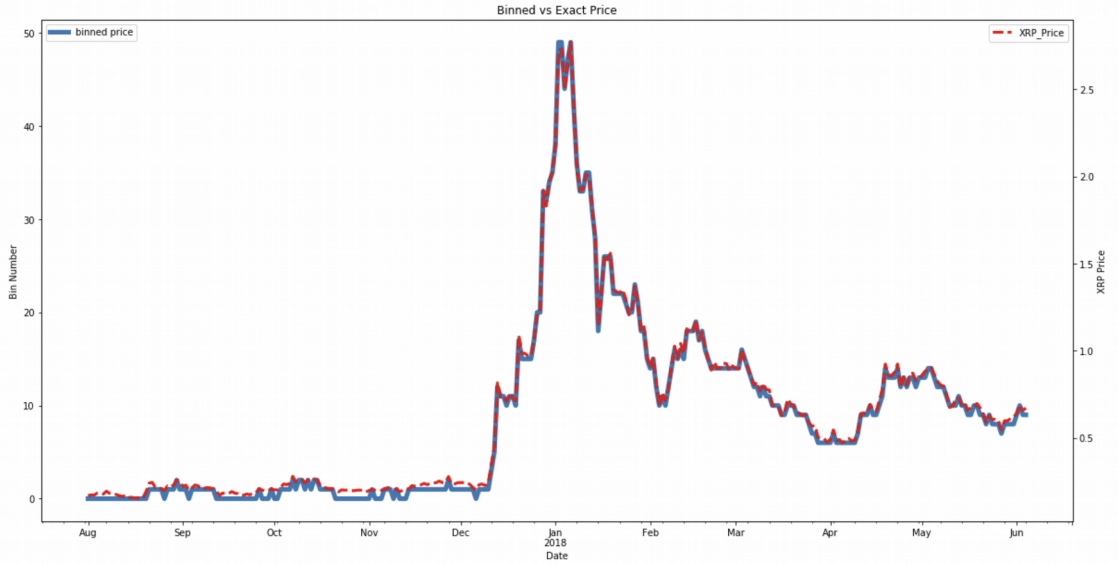


Figure 5:13 - Price ranges represented by bin numbers are illustrated on the left y-axis. Exact Prices illstrated on the right y-axis

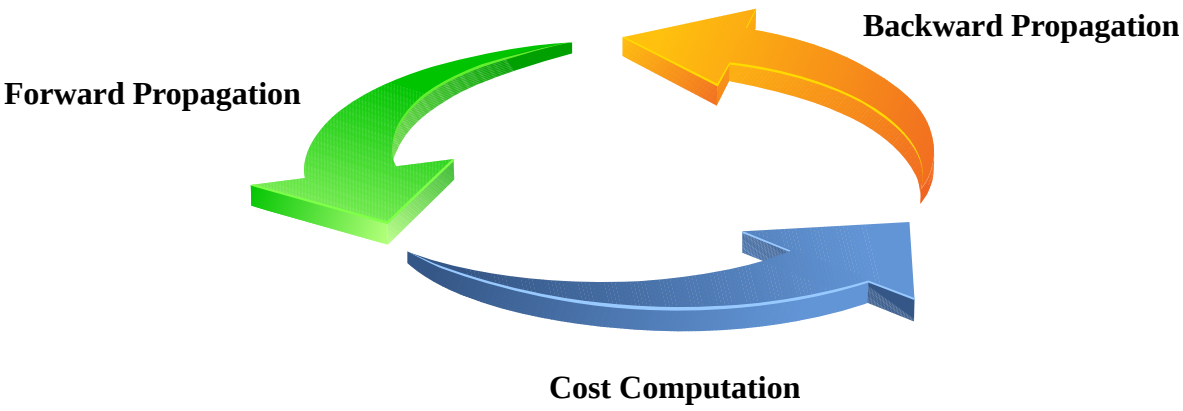
Utilizing all variables, a correlation matrix was then constructed to compare the correlation coefficients of exact price vs the $0.05 cent price ranges.

	binned price	XRP_Price
Positive Sentiment Summed	0.445597	0.422747
Overall Sentiment Summed	0.349601	0.329885
XRP_High	0.979355	0.977217
XRP_Low	0.970971	0.968592
XRP_Open	0.968363	0.966195
XRP_Price	0.999553	1.000000
BTC_High	0.700327	0.701639
BTC_Low	0.705891	0.706575
BTC_Open	0.709501	0.710609
BTC_Price	0.699898	0.701040
ETH_High	0.838770	0.836097
ETH_Low	0.832076	0.828926
ETH_Open	0.823656	0.820749
ETH_Price	0.844364	0.841482
Summed_box_filter	0.581359	0.578319
thirty day percent change	0.707909	0.710211
binned price	1.000000	0.999553

The binned prices didn't drastically improve any correlation coefficients, but they also didn’t degraded them either. Because of this, it was still deemed more feasible to utilize ranges when predicting XRP Price for accuracy sake. A Deep Neural Network was constructed to build the predictive model.

6. PREDICTIVE MODELS WITH DEEP NEURAL NETS

Due to the extreme random nature of the crypto world, a predictive model in this space was deemed too complex for a simple machine learning algorithm such as K-Neighbors or Naive Bayes. Instead, deep learning models were constructed and tuned. A complete in depth discussion on Neural Networks and Deep Learning are out of scope for this document. A high level overview will be had for minimum required knowledge transfer needed for the duration of this report. Training a neural network requires the following steps:



6.1 Forward Propagation and the Perceptron

The basic building block of a neural network is the perceptron. The perceptron is a binary classifier that takes some inputs, multiplies them by some weights, applies an activation function and responds with an output of 0 or 1 (hence binary).

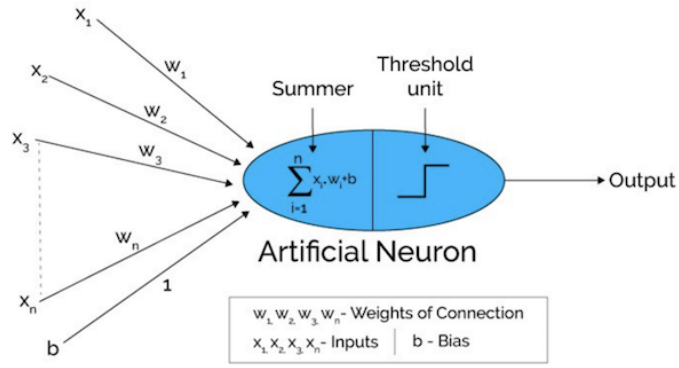


Figure 6:1 - Graphical representation of a perceptron, the building blocks of a neural network

The inputs x_i are the independent variables used to predict the dependent variable of interest y. During the training stage, weights w_i and bias term, b are randomly initialized. These terms are automatically tuned during the training process. The easiest way to relate to these terms are in the equation for a line: y = m*x + b, where m is the slope and b is the intercept term. Here, weights are equivalent to slope m. For every input variable of x, it is multiplied by its respective weight w, and all those products are summed together with the bias term b. Since the perceptron is a binary classifier, it must output a 0 or 1. Therefore this summation is a function of some activation function. There are various types of activation functions, but for this project, RELU (rectified linear units) and sigmoid functions were used.

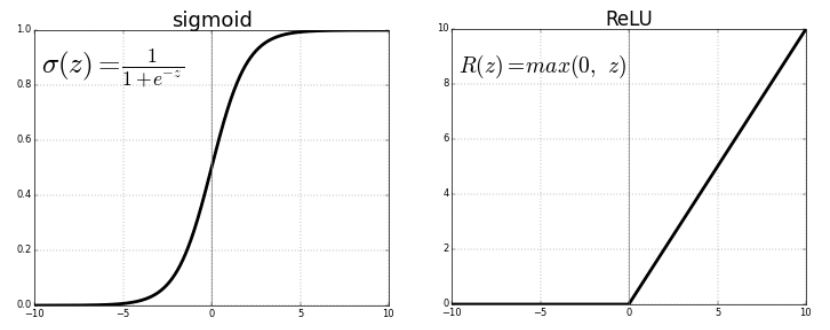


Figure 6:2 - Sigmoind function (left) has asymptotes at 0 and 1. Any input value (z) will return some value between these asymptotes. RELU (right) returns some value between 0 and infinity

Keeping within scope of this report, it is left to the reader to learn further about activation functions. The main takeaway is that these functions, when built into the neural network architecture, will return some value between 0 and 1 (the desired binary ouput). A deep neural network is just a conglomeration of these perceptrons.

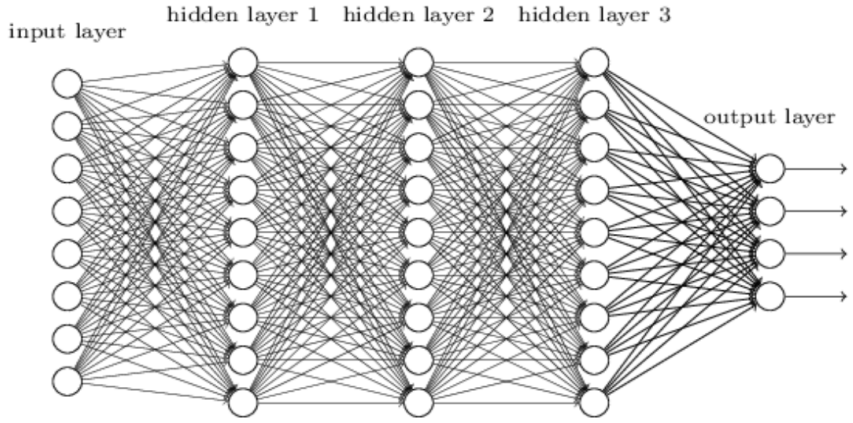


Figure 6:3 - Simple representation of a neural network. The input layer is representitative of the input features to make the prediction. Each layer between the input and output layer is called a hidden layer. A NN can have as many of these hidden layers as desired with as many nodes in each layer. The output layer will always be the size of the desired output. If predicting a single value, the output layer will also be 1

The input variables x are fed through the first [hidden] layer in the neural network where each perceptron outputs it's activation value as an input value for the next layer. The overall architecture of the neural net (number of hidden layers and number of nodes in each layer) is at the discretion of the user. With a larger and/or deeper network, more complicated trends can be learned. This "feed forward" process is continued until the final output layer is reached with the model's initial prediction. This is called forward propagation.

6.2 Cost Function

The error between the true value y and the model's prediction $\hat{y}$, also called the cost, is calculated after the first feed forward pass. This is a measure of how far off the model is making predictions. Although not derived here, this error equation is:

i. $$J(\Theta)=\frac{1}{m}\sum_{i=1}^{m} Cost\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$$

ii. $$\text{Cost}\left(\text{h}_\theta\left(\text{x}\right), \text{y}\right)=\begin{Bmatrix} -\log\left(\text{h}_\theta\left(\text{x}\right)\right), \text{if y}=1 \\ -\log\left(1-\text{h}_\theta\left(\text{x}\right)\right), \text{if y}=0 \end{Bmatrix}$$

iii. $$Cost\left(h_\theta(x), y\right)=-y log\left(h_\theta(x)\right)-(1-y)\log\left(1-h_\theta(x)\right)$$

The above equations might look daunting, but they're rather simple. $h_\theta(x)$ is nothing more than the model's predicted output $\hat{y}$. For each observation ran through the model, its true label y will be either 0 or 1. If the label is 1, then the top equation in item ii is utilized. Conversely if the true label y is 0, then the bottom equation in item ii is used. The model's prediction is inserted as $h_\theta(x)$, and the output is the cost for that observation. The cost for all observations are summed and divided by the total number of observations m. The goal during training phase is to minimize this cost to the lowest value possible.

6.3 Back Propagation and Gradient Descent

Back propagation is an extremely advanced topic and is only touched on for completeness of this brief intro to neural networks. After the cost is computed, the neural network is then traversed backwards starting from the output layer. At each node (perceptron), while traversing backwards, the partial derivative of the cost function is calculated with respect to the weights and bias terms. As a gross over simplification, this is done to quantify how much error any given node was responsible for in the total error. The weights and bias terms at each node is then updated by subtracting the error due to their respective node. This process is known as gradient descent.

After performing the update on each node's weight and bias term, the feed forward process starts all over again. Each round trip in this training process is called an epoch. The goal is to repeat these epochs iteratively until the weights and bias terms are updated to values that create the lowest error during the forward propagation process. In theory, this should result in the model making a correct prediction.

7. NN TRAINING

The Jupyter Notebook containing Neural Network creation and training process for this project can be viewed here. The main function that iterates through forward prop and backwards prop can also be seen in Appendix E. Special functions were created to facilitate the training process. These functions are modified functions from my code utilized in my Deep Learning course on Coursera with Andrew Ng. You can see those notebooks HERE on my github for in depth details on how these functions work and more insight on how Neural Networks work.

7.1 Variable Preprocessing

Recall from the data exploration analysis, it was decided that price ranges might be more pertinent than trying to predict an actual price. In order to train the NN on this strategy, the y values (XRP Price) were categorized into separate bin ranges. The price data spanned $0.15 cents on the low end to $2.77 on the high end. If we wanted to break this into 3 separate ranges, for simplicity, we would label those ranges 0-2. For clarity:

- [$0.15 - $1.025) = bin 0
- [$1.025 - $1.8975) = bin 1
- [$1.8975 - $2.77] = bin 2

Therefore, a XRP Price of $1.09 would be classified in bin 1 since it exists in between the range of $1.025 and $1.8975. The NN can only predict between 0 and 1 however (utilizing RELU activation for the L-1 layers and Sigomoid activation for layer L). Because of this, "One Hot" encoding was performed on the dependent variables, XRP Price Ranges. All this means, utilizing the 3 bin range example, is that each XRP Price is represented as an array of length "# of bins". All values of this array will be 0 except for the value at position = "bin #". For example:

- $1.025 = **bin 1** out of 0-2 bins
- **bin 1** out of 0-2 bins = [0, 1, 0]
- $1.025 = [0, 1, 0]

For concreteness:

- $0.23 =**bin 0** out of 0-2 bins
- **bin 0** out of 0-2 bins = [1, 0, 0]
- $0.23 = [1, 0, 0]

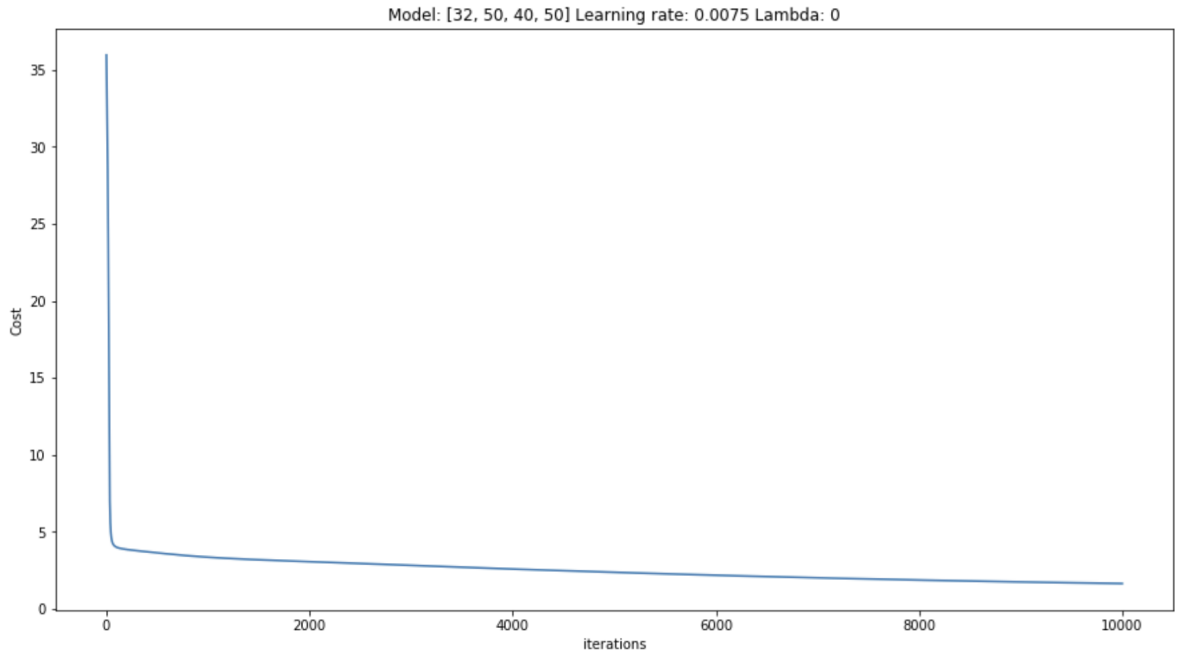
Currently all the data resided in one large dataframe with 32 columns and 309 rows. Refer to Figure 5:1 for all column variables and their statistics. The data needed to be preprocessed in a particular manner before being fed through a neural network while taking into account these price ranges. A function was created to perform the following:

1. Independent variables X are separated from the Dependent variable y
2. X is normalized to 1 to alleviate dimensionality discrepancies
3. y is one hot encoded to support the desired amount of price ranges

7.2 Next Day Price Prediction & Initial Model Performance

The first goal for predictive model creation was the ability to create a model that could accurately predict tomorrow's price. Before any hyper parameter tuning and model tweaking was performed, a baseline needed to be established. The data was split into 60% training samples and 40% test samples. The XRP Prices were categorized into 50 different price ranges. This price ranges are identical to those in Section 5.5. Model architecture was arbitrarily chosen to be (32, 50, 40, 50). This is a 4 layer model with an input layer of 32. The 32 nodes are the 32 input features (columns of the dataframe that were spearated and scaled in the preprocessing step). The first hidden layer had 50 nodes, the second 40, and the output layer was 50. The 50 output nodes are representative of the 50 price ranges created. The model outputs all zeros for the 50 nodes in the output layer except for one. The location of the node that returns a 1 is the price range bin the model predicted based on the input variables.

While training is in progress, performance is monitored real-time by the cost output for each epoch (iteration). After all epochs are complete, in this case 10,000 epochs, accuracy is computed on both the training and test sets. Those results are shown below



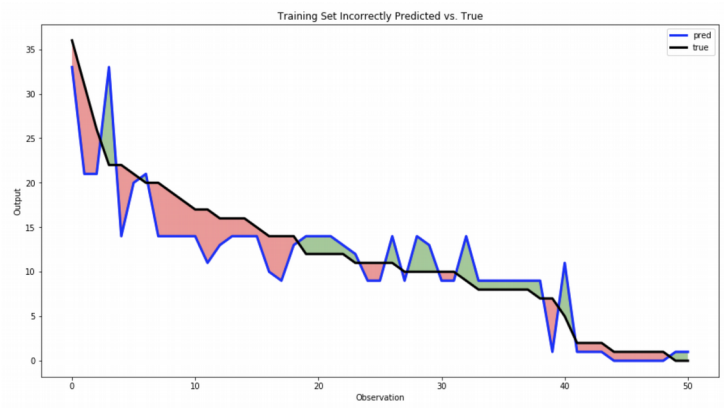

```
This Model's Train Set Accuracy is: 0.7228260869565217
This Model's Test Set Accuracy is: 0.3629032258064516
```

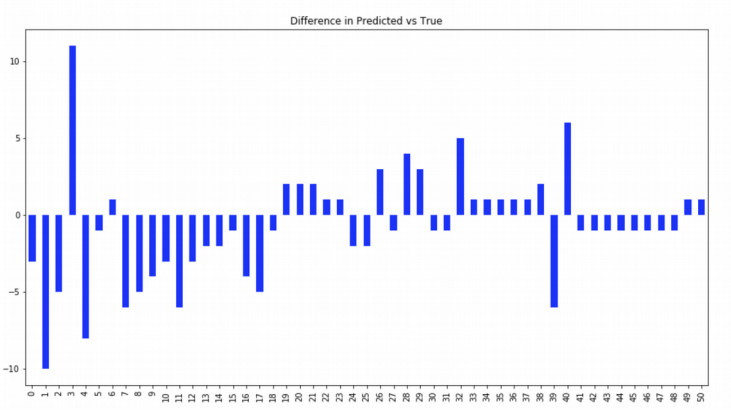
Figure 7:1 - Initial model performance. Cost after each epoch is plotted. Overall model accuracy is shown underneath cost performance

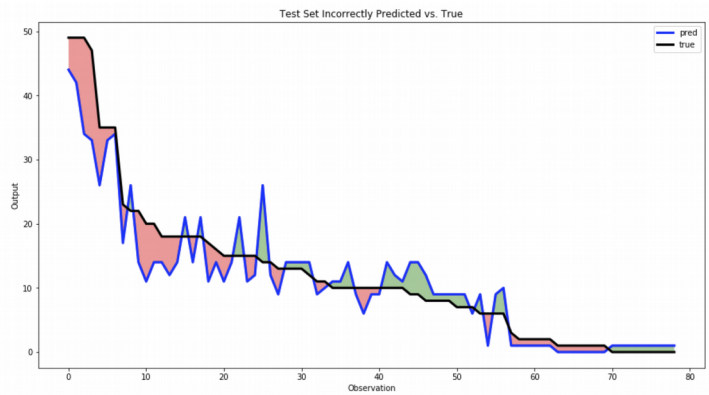
After 10,000 iterations the cost was down to approximately 1.63. A lower cost is desirable, but the current value isn't terrible. After each epoch, the cost was continuously decreasing without jumping up and down. Because of this, either the number of iterations or the learning rate, could be increased. Increasing iterations can be computationally expensive. Conversely, increasing the learning rate to too high of a value might prevent the cost from ever converging to a minimum. This is a value that can be tuned.

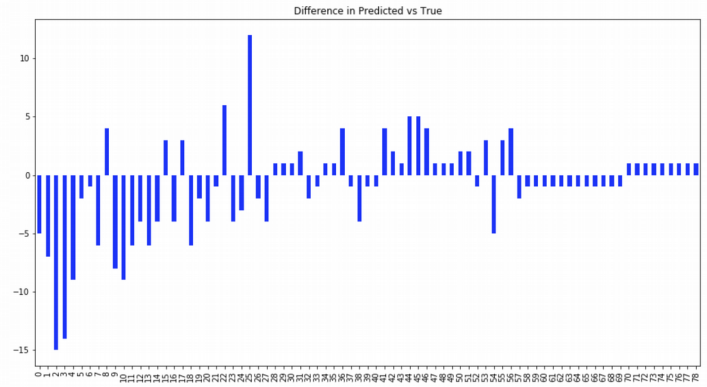
On the first dry run the model achieved 72% accuracy on the Training Set and 36% accuracy on the test set. With 28% and 64% error respectively, the model suffered from both High Bias and High Variance. It did poorly on the training set, and therefore has high bias. Its performance on the test set was worse, indicating high variance. Because of this, regularization wouldn't help very much since there is already high bias on the training set.

Further analysis was performed on the samples that were incorrectly predicted. The correctly predicted samples were filtered out, leaving only incorrect predictions. The true values were plotted against the predicted values to gauge how severely wrong the predictions were.









From the graphs above it is revealed that the model is under predicting the expected price range. Which isn't a huge problem to have in the grand scheme of things. When investing it is always better to predict on the lower end than to over estimate. However when the model does over predict the expected price range, it predicted much higher than the actual range. this is seen on both the training and testing data. The training data has a difference of as much as 12 price ranges. That would be the same as the model predicting $0.88 - $0.94 cents but the real value being $0.15 to $0.20 cents.

7.3 Model Tuning

After initial model analysis, various parameters were tuned in conjunction with several different model architectures. Each new model was trained with varying amounts of training samples and performance was gaged on the subsequent test samples. The expected trend from this method is; with increasing training sample sizes, error would increase. However, error on the test set should decrease. The best performing model, with architecture [32,50,40,50,30,50], resulted with the following metrics:

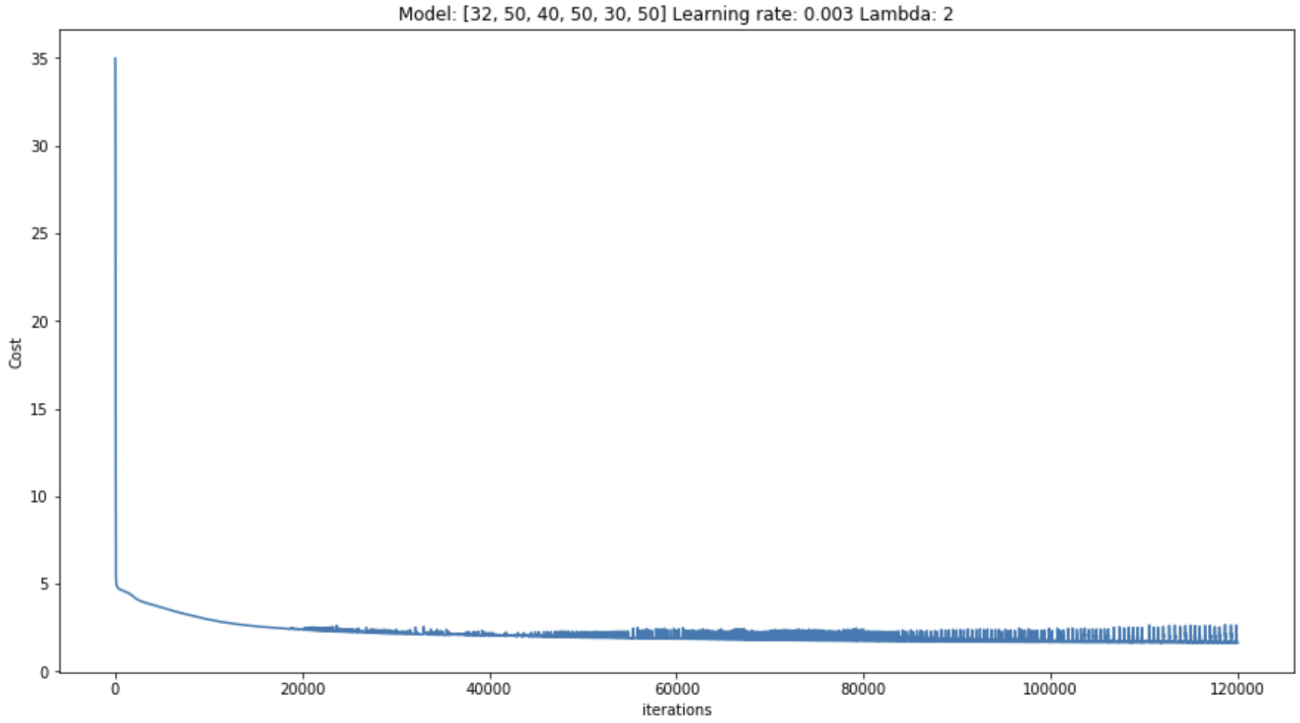


Figure 7:2 - Cost analysis from best performing model on price range predictions

Cost was decreasing with each iteration, but notice the spikes along the way. This is an indication that the learning rate is too high. Gradient descent is struggling to converge on a minima. Fortunately on the last iteration, the cost appears to have still found a minima

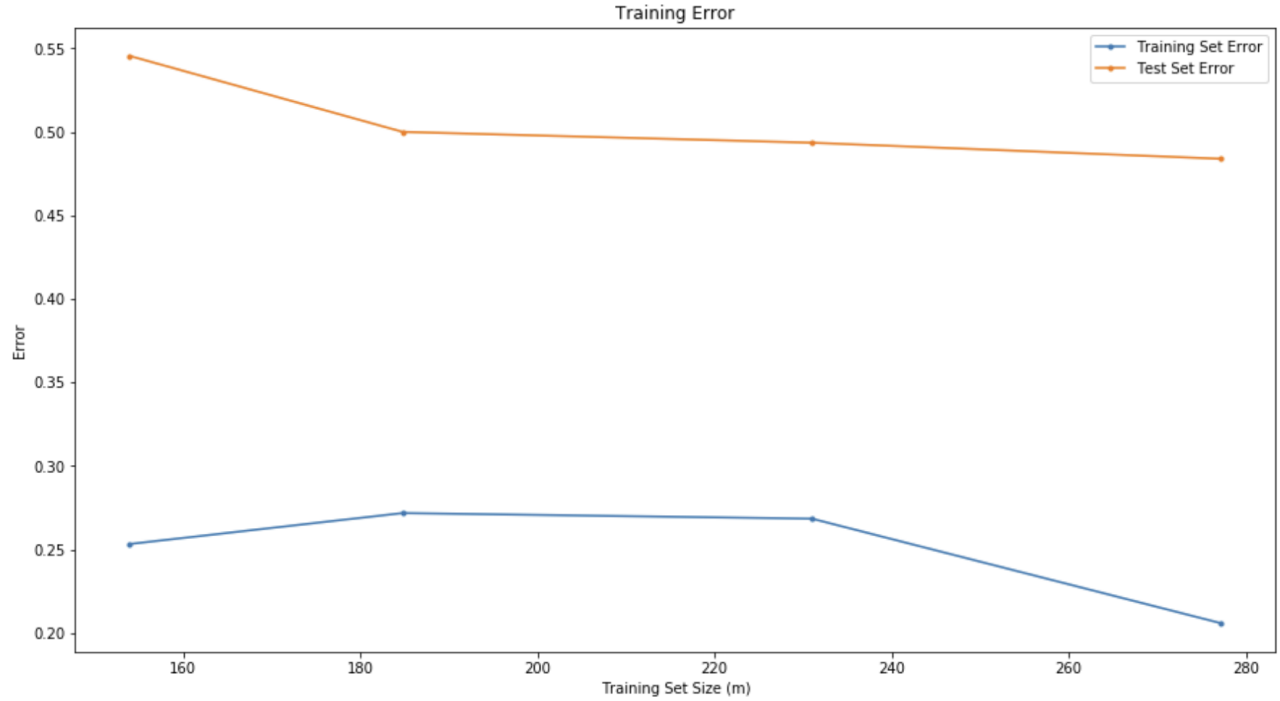


Figure 7:3 - Model's overall performance on both training and test sets after being trained by various size training samples

As this NN was learning (trained), each increase of sample sizes in the training set reduced error in the test set. This is an indication that the model is beginning to generalize well to unseen data. However as the best performing model, the test set still had about 48% error. This is better than the initial 64% error, but not good enough.

To further evaluate the model, a confusion matrix was constructed to determine the model's performance on predicting each individual bin (price range).

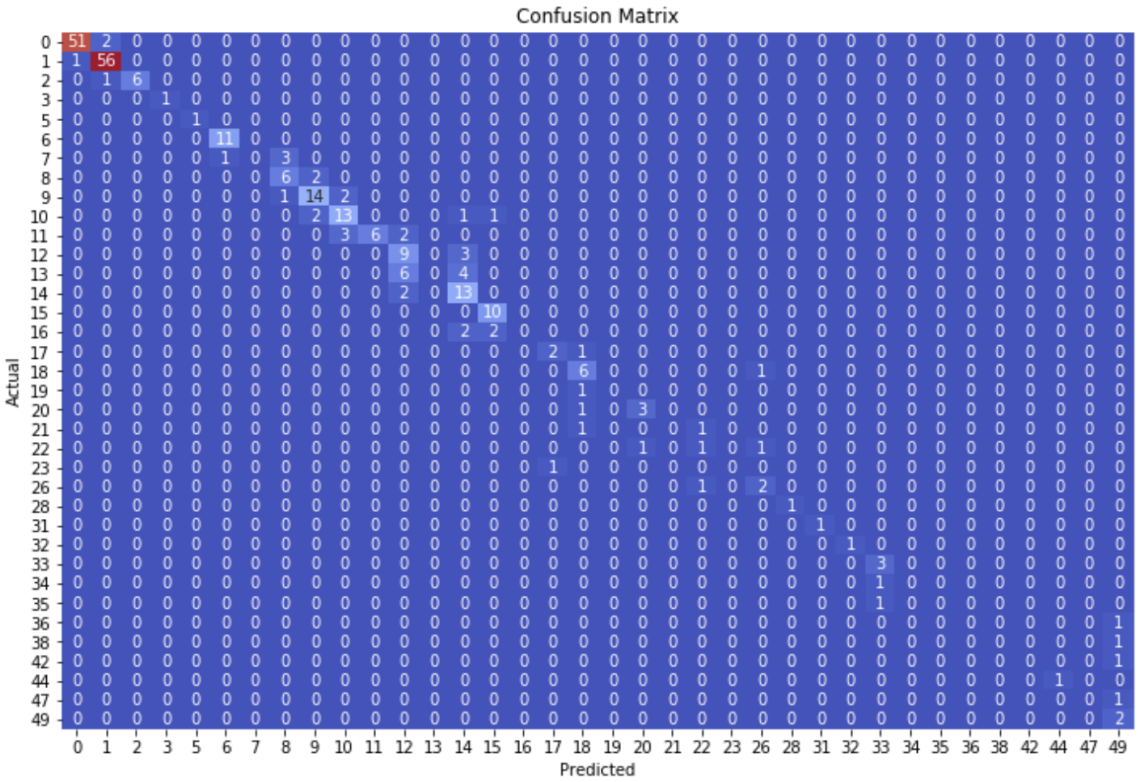


Figure 7:4 - Training Set confusion matrix. Illustrates prediction performance. The matrix shows how many times a price range was predicted vs what the actual price range was.

From this evaluation, it was pretty evident why the model's performance was incapable of reaching higher standards. The training samples consisted heavily of the lower price ranges. Of the 247 training samples, 110 consisted of bin ranges 0 and 1. The model learned these ranges almost perfectly, but struggles at at almost all other bin ranges due to never learning them adequately. This is highlighted when constructing a confusion matrix on the test sample.

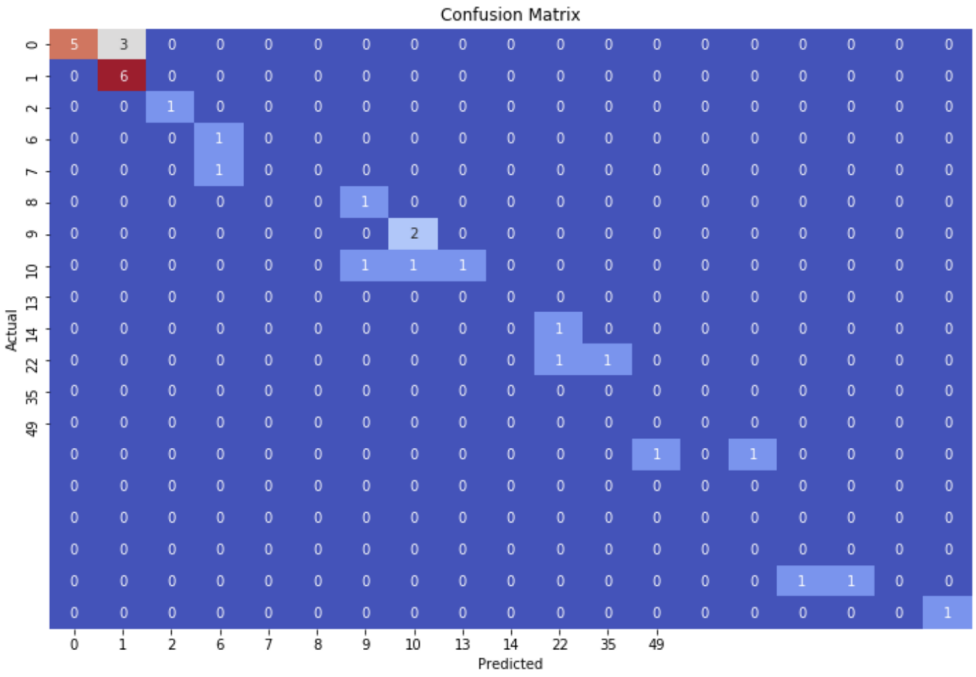


Figure 7:5 - Test Set Confusion Matrix. Illustrates model's performance on the test set

When making predictions on the test set, the model is fairly accurate on the lower price ranges. But it can be seen to struggle at the higher price ranges. It even makes predictions on price ranges that didn't even exist in the test set. This is represented by the values in the matrix that don't have a corresponding x or y axis value.

When a histogram is plotted of the price range distributions, it is clearly shown just how semi homogeneous the data is.

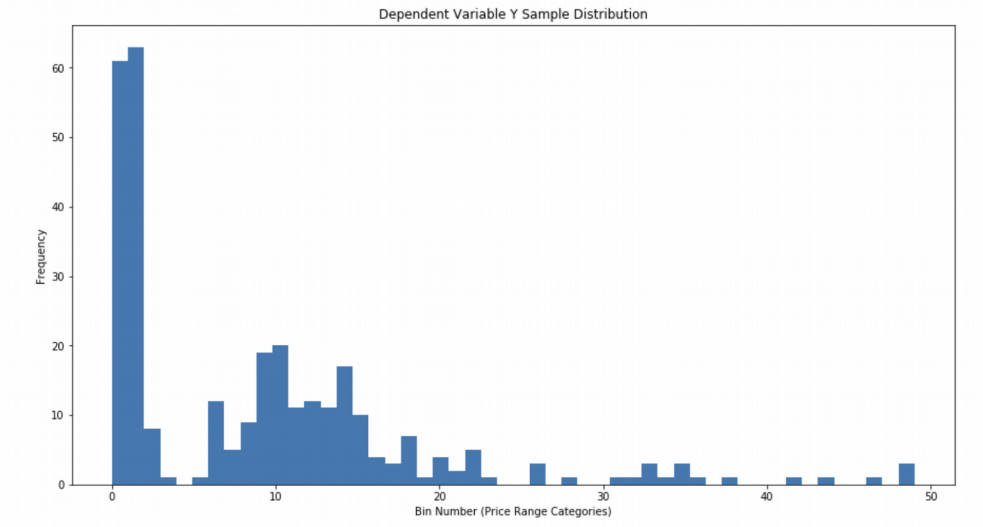


Figure 7:6 - Histogram distribution of all 50 price range bins

Bins 0 and 1 are in order of magnitudes larger than all other bins (Price Ranges). There are several bins with only 1 sample as well as several bins with 0 samples to represent them. It's obvious that 50

separate price ranges was too many, as there is not enough data in the higher price ranges to train the model with. Distributions were plotted in order to examine 30 and 20 price ranges.

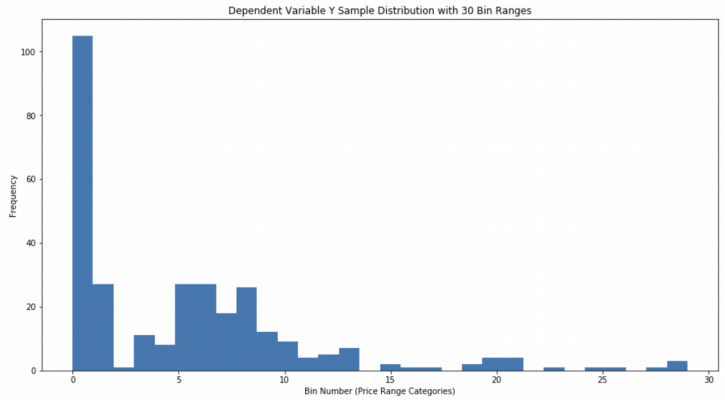


Figure 7:8 - Histogram distribution of 30 different price range bins

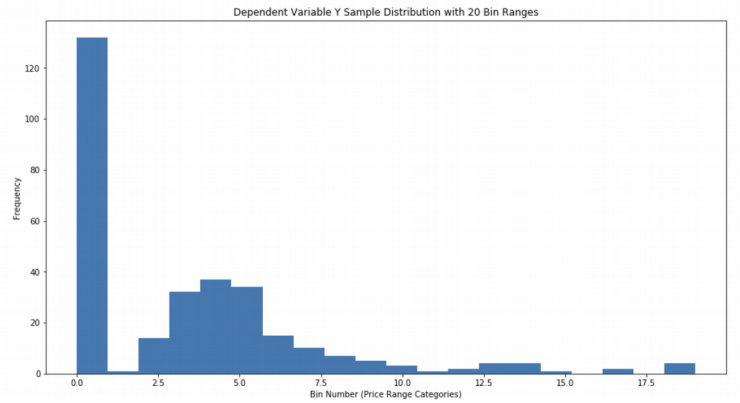


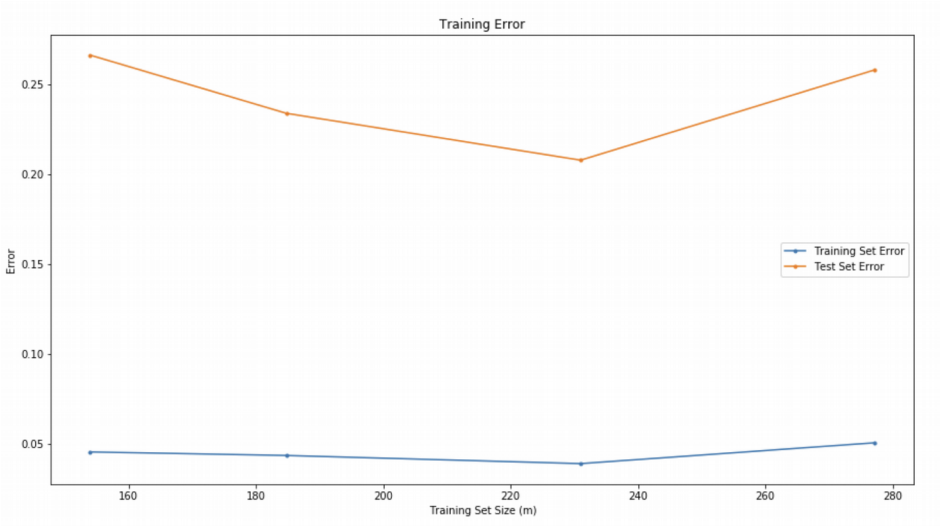
Figure 7:7 - Histogram distribution of 20 different price range bins

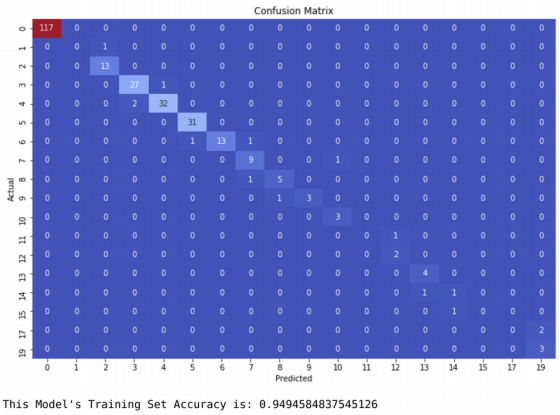
Reducing the amount of Price Ranges (bins) is creating more samples for each bin. However there are 2 drawbacks to doing this.

1.The first bin will become larger when reducing price range bins. This still doesn't alleviate the issue of the model being trained on primarily one sample type. It might not generalize very well to unseen samples.

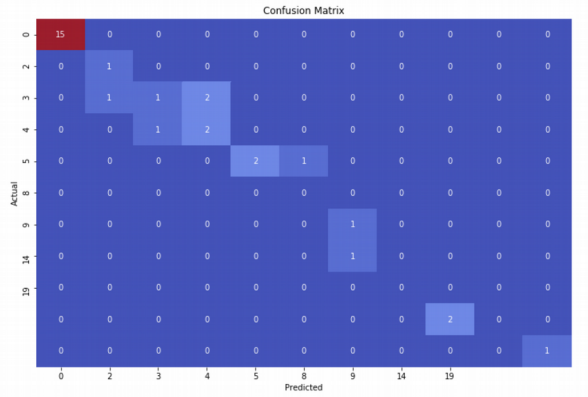
2.The price ranges become larger, making the predictions very inaccurate.

Remember that 50 price ranges correlated to price ranges of approximately $0.05 cents. Predicting a price with a maximum error of $0.05 cents is more than tolerable. Let's see how the performance looks with only 20 price range bins





This Model's Training Set Accuracy is: 0.9494584837545126



This Model's Test Set Accuracy is: 0.7419354838709677

Trying various models structures and hyper parameters, the best results achieved was 75% on the test set at 60% training examples, 72% at 75% training examples and 74% at 90% training examples. 74% isn’t a horrible score. In fact it is 20% more than the accuracy we were able to achieve with the 50 bins model. But as stated earlier, at 20 price range intervals (bins), the margin for error was already small since the ranges are so wide. It is now evident that predicting price ranges isn’t the best method

7.4 Regression Model with NN

Due to unbalanced data, a multi-class classification NN just didn't provide the desired results. So the strategy was pivoted to a regression model. Our model utilizes both RELU and sigmoid funcations which restricts output values between 0 and 1. Therefore, each individual XRP Price was normalized to 1 instead of one hot encoding it into different price ranges.

The best performing model was of architecture [32,20,20,10,5,1]. There are still the same 32 input features, but now only one output layer. This is due to the fact that the model no longer predicts one of the many price ranges. Instead it now predicts an actual price point. This price is scaled between 0 and 1 and is transformed back into a non-normalized price by reversing the transformation applied to the input data. The training set evaluation appears as such:

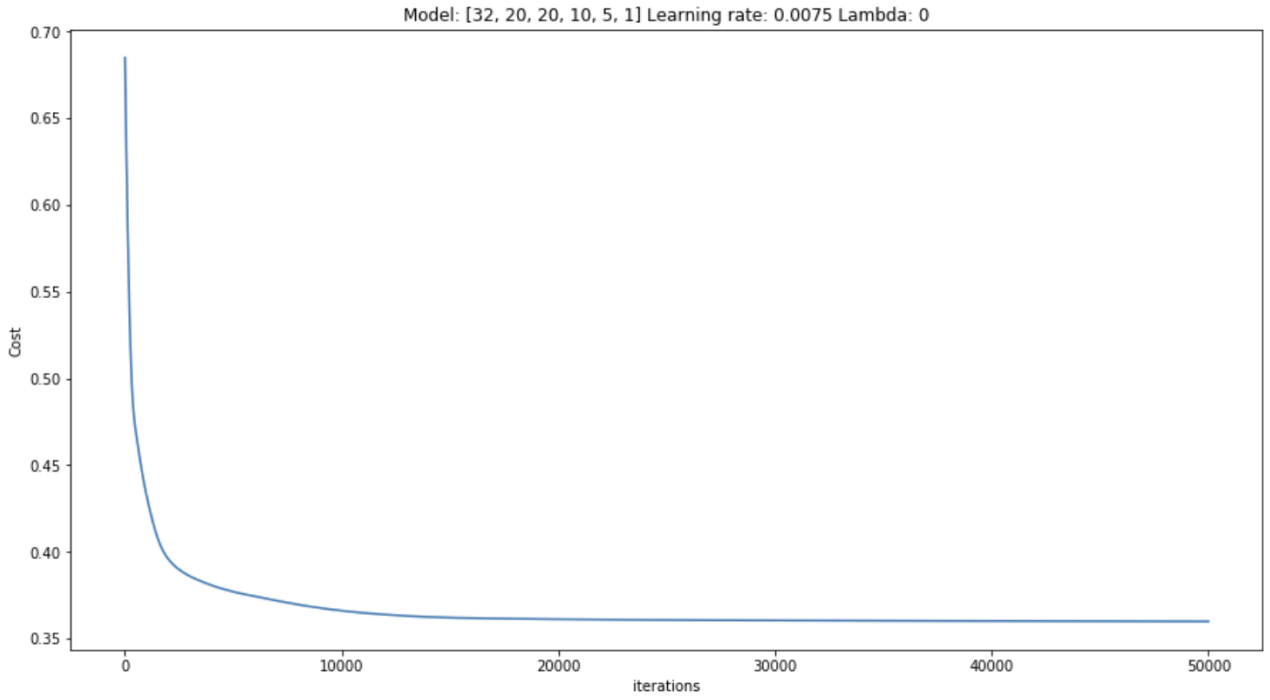


Figure 7:9 - Best Performing NN Regressional Model Cost Analysis

Cost continually decreased with each iteration. To avoid the spike issues seen while training the last model, the learning rate was set lower and the epoch iterations was set higher at 50,000 iterations.

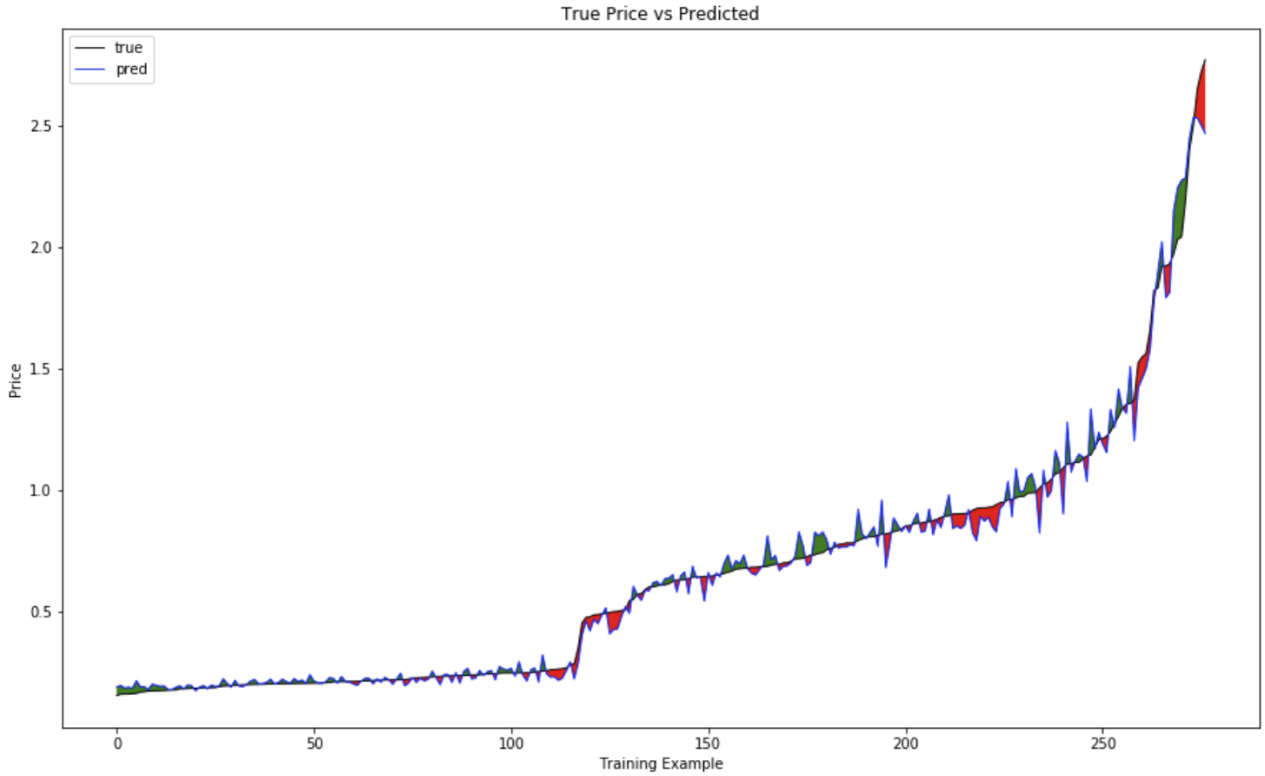
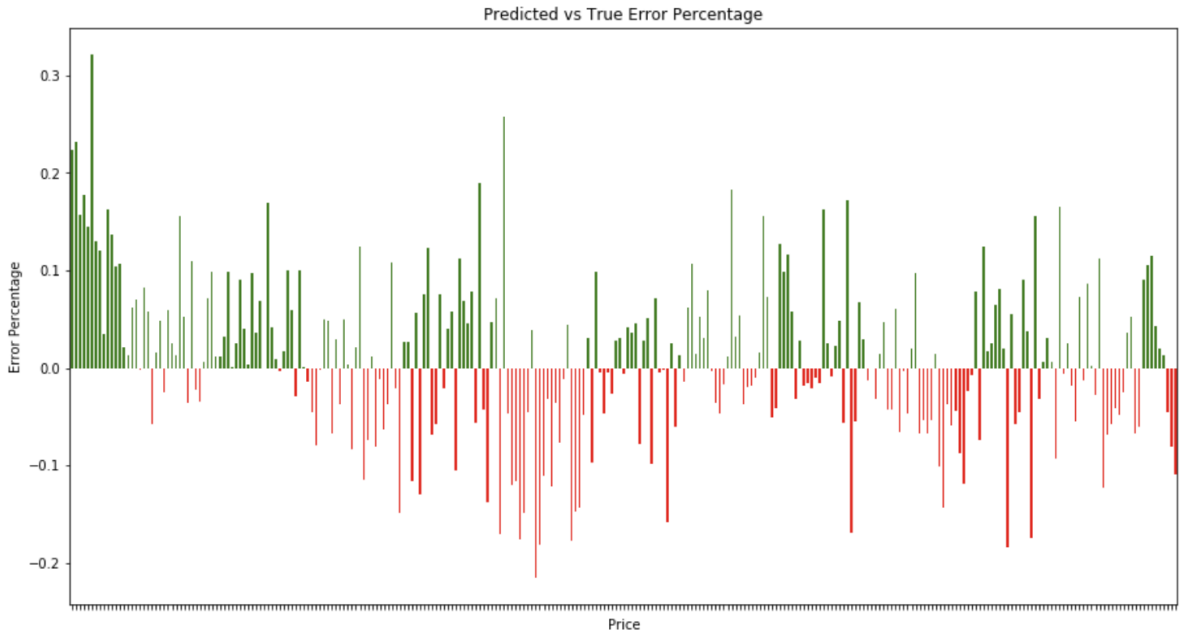


Figure 7:10 - Predicted vs Actual Price. Black is the actual Price, Blue is the prediction made by the model. The error between prediction and actual is highlighted. Green for over prediction, red for under prediction

While plotting true price versus the model's predicted prices, the model appeared to do surprisingly well at predicting not only the prices, but the trend. It wasn't perfect however. Clearly the model did struggle at certain price points. Fortunately, some of the largest error differences the model produced was at the lower price points. So a high error percentage doesn't translate to a high error in price point.



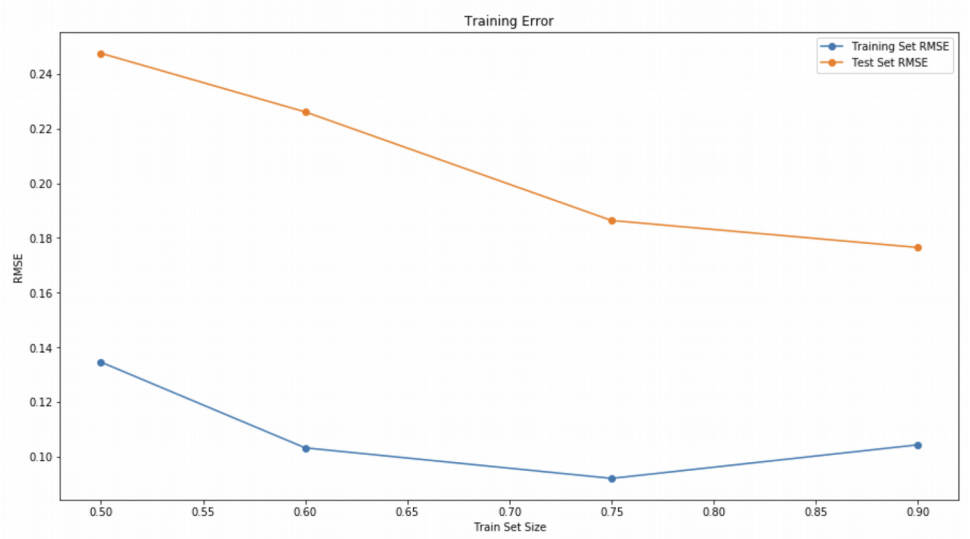

```
RMSE For These Parameters: 0.05994845429944895
```

Figure 7:11 - Error percentages for model's prediction at each observation. The highest errors were made at the lowest price points.

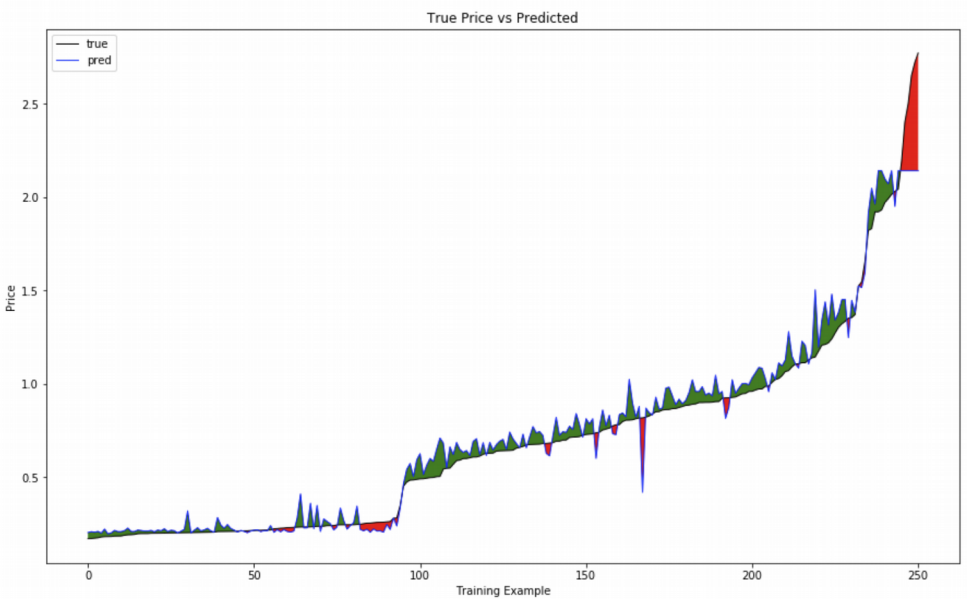
After plotting the error percentage for each prediction in the training set, it was revealed that the highest error was about 30%. This error was a prediction made at the $0.20 price zone where 30% error constitutes only $0.06 cents. Also take note that the RMSE (root mean square error) was a miniscule 0.06. This could be deemed as an acceptable model until more heterogeneous data could be collected.

7.5 30-Day Price Prediction

An investor is not only interested in day to day prices, but potential long term performance as well. A model was trained to predict XRP Price 30 days in the future. This was done by following all the same procedures as before but now XRP Price was shifted up by 30 days to coincide input data (indpendent variables X) with a price that was seen 30 days later. The best performing model produced the following results with several things to take note of:



First thing to note is that the RMSE is higher when predicting 30 day prices in comparison to next day prices. This is to be expected since there is a lot more randomness being introduced as time is increased. There's no way to completely account for all randomness in the model.



Second, even though the above prediction plot only shows the results for a model trained with 90% training samples, there's a very distinct pattern in all the training sessions with varying sample sizes. The model almost always over predicts at the lower price points. Before the first price spike, it under predicts, right at the elbow. Once the price almost flattens out again after the first spike, the model begins to over predict again with intermittent "under predictions" along the way. These under predictions are almost always at the same spot in every session. And lastly, as the model tops out at the higher price predictions, it greatly under predicts at the end. This trend is happening in every training session, and as such might be easy to compensate for.

Lastly, now that we are extending the timeline from next day to 30 days for price prediction, we can learn two things from this model.

1. Predicted 30 day future price with error accounted for
2. General Price trend over 30 days (will the price go up or down in 30 days from today)

In regards to item 1.) any price prediction the model gives will have some error that needs to be compensated for. Small price ranges are examined to determine how much error is present.

Error Statistics By Price Ranges

bins	count	mean
(2.666, 2.77]	2.0	-0.220307
(2.562, 2.666]	1.0	-0.192423
(2.458, 2.562]	1.0	-0.143968
(2.354, 2.458]	1.0	-0.108300
(0.275, 0.379]	3.0	-0.062643
(1.626, 1.73]	1.0	-0.036017
(2.146, 2.25]	1.0	-0.022795
(1.522, 1.626]	1.0	-0.020196
(1.418, 1.522]	1.0	0.000756
(1.938, 2.042]	5.0	0.035400
(1.314, 1.418]	5.0	0.037121
(0.795, 0.899]	23.0	0.044667
(0.899, 1.003]	22.0	0.049716
(0.691, 0.795]	20.0	0.055399
(1.003, 1.107]	8.0	0.061472
(0.587, 0.691]	30.0	0.062009
(0.168, 0.275]	92.0	0.063260
(1.834, 1.938]	3.0	0.081771
(1.73, 1.834]	2.0	0.085578
(0.379, 0.483]	2.0	0.086299
(1.107, 1.21]	9.0	0.089446
(1.21, 1.314]	4.0	0.096798
(0.483, 0.587]	14.0	0.181853

When aggregating the average error on the training set, it is revealed just how much the model is over or under predicting (on average) for a given price segment. E.g. if the model predicted a price of $0.90 cents, there’s about 5% error in that prediction according to the stats above.

In regards to item 2.), a trend plot is constructed with the true prices, and the predicted prices are overlaid. The goal isn’t to measure exact price accuracy. Instead the goal is to examine overall trend accuracy. The model, if not price accurate, should be able to determine, at a minimum, an increase or decrease in price 30 days from now.

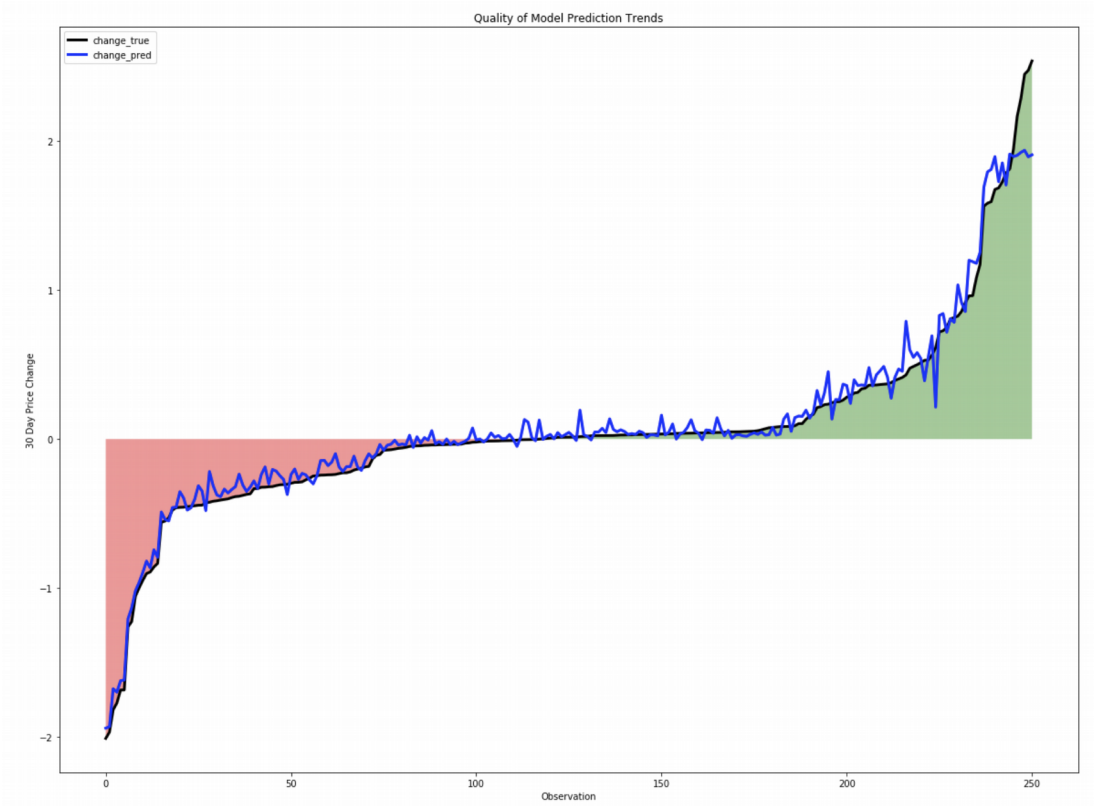


Figure 7:12 - Actual price trends (black) versus model predictions (blue)

In the above graph, it is shown that when the true price has dropped in 30 days from the original price, the model almost always also predicted a drop in price as well. Conversely, the model almost always shows a positive 30 day trend when the true price has increased in 30 days from the original price. With that being said, towards the middle, where price change is minimal in 30 days, the model appears to struggle. Where the trend line is almost flat, the model appeared to be over predicting. That particular region was further focused on and analyzed to determine how wrong the trend error is.

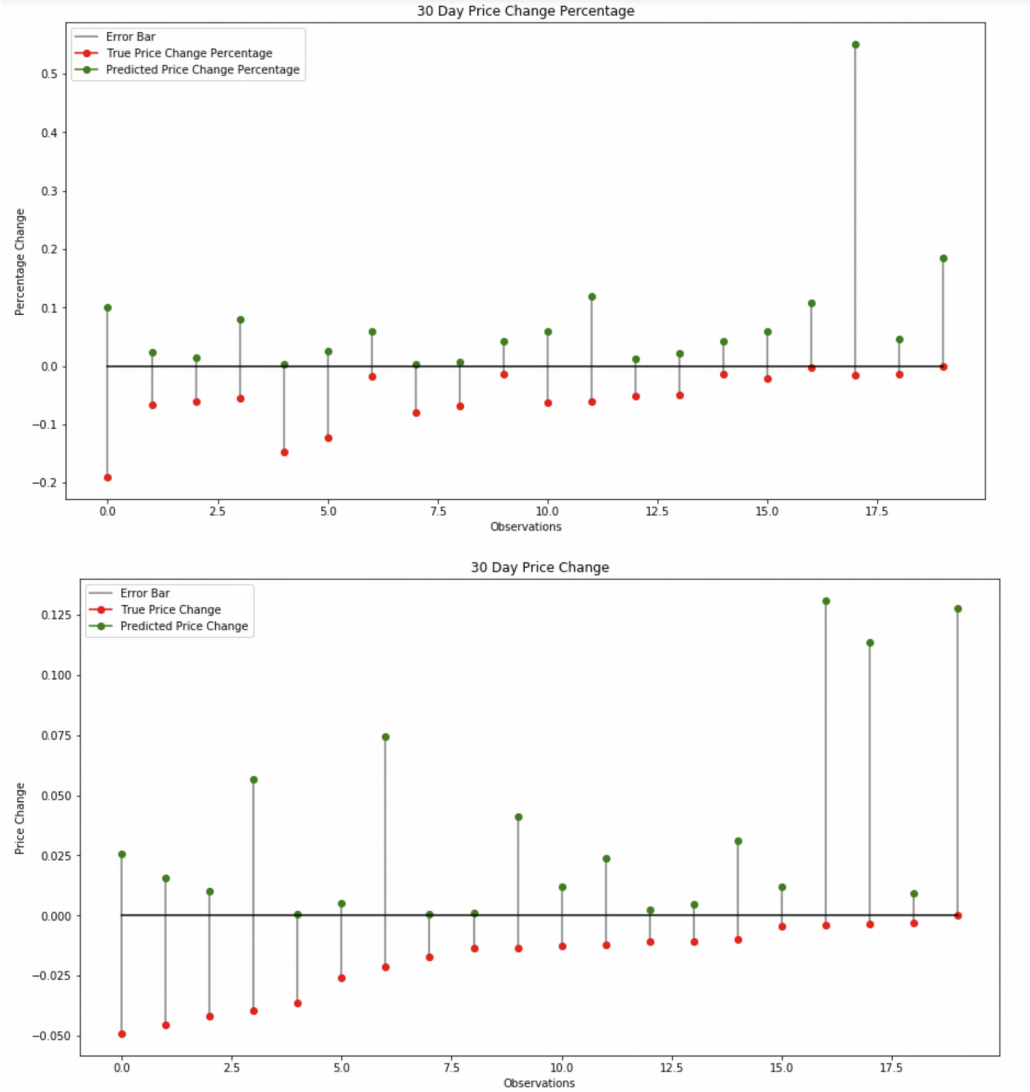


Figure 7:13 - Trend errors for each observation the model incorrectly over-predicted trend. The top graph is representative of Percent Change error. The bottom graph is representative of the actual price change error

The graphs above are interpreted as such: Each vertical bar is an individual observation. The horizontal black line represents the original price for each individual observation (the zero point for the prediction). The red dots are the true prices, 30 days from the original price and the green dots are the model's predicted price, 30 days from the original price. The biggest error the model made on an observation was predicting a 55% percent price increase in 30 days (top graph). The predicted price increase was 0.11 cents (bottom graph). The original price was approx $0.205 cents. The model predicted $0.31 cents in 30. The true 30 day price was 0.202 cents, a difference of 0.003. So although the model predicted a substantial jump in price with this particular observation, at least the loss (if real money had been invested) would be almost negligible. In fact, this is the case with all 20 observations. The biggest loss an investor would have suffered is 20% at a point in which the model only predicted a modest 10% increase. It is doubtful an investor would have made a massive investment with predictions of only 10% short-term return. Next, let's look at the model's under predictions

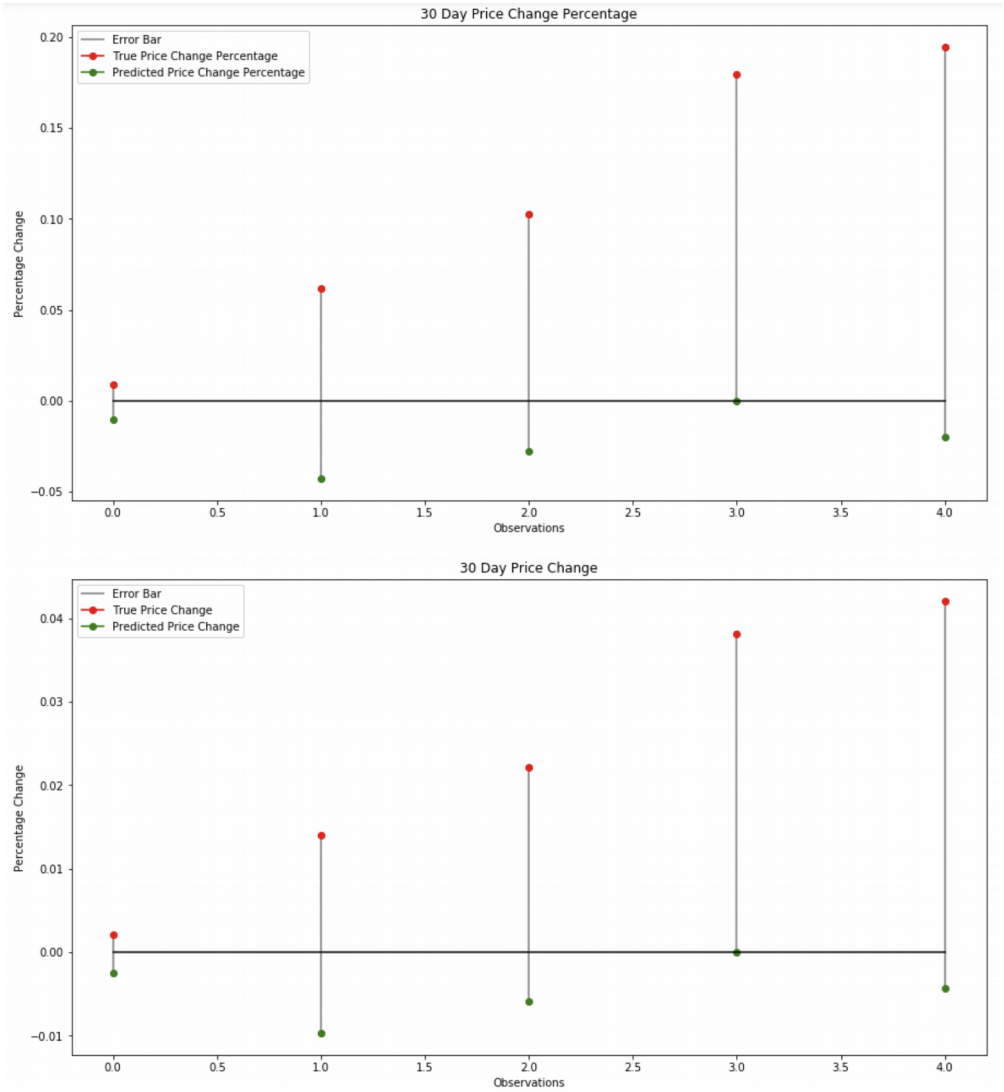


Figure 7:14 - Trend errors for each observation the model incorrectly under-predicted trend. The top graph is representative of Percent Change error. The bottom graph is representative of the actual price change error

There were only 5 instances in which the model under predicted the 30 day price trend. In these instances, the largest missed opportunity was a 20% return on investment (ROI). At that point, the model predicted approx a 2% price drop over the course of 30 days.

The overall “trend” statistics are as follows:

```
Models Overall Trend Accuracy is: 90.04%
Percentage of Over Predicted: 7.97%
Percentage of Under Predicted: 1.99%
```

The model was 90% accurate at predicting at least a correct 30 day future price trend. When it did predict incorrectly, approx 8% of the time it was over predicting, potentially losing an investor's money

(but a negligible amount). 2% of the time it was under predicting, costing an investor potential gains. This model is deemed acceptable, again until more heterogeneous data can be collected

7.6 Model in Action

On June 4th, 2018 the model was executed to predict a price 30 days in the future. The model returned:

```
Predicted Price for July 4th, 2018 is 0.656
```

APPENDIX A – Custom Web Articles Page Parsing Class

```
from bs4 import BeautifulSoup
from bs4.element import Comment
import requests
import re
from datetime import datetime


class parse_web_page:
    #constructor
    def __init__(self,url,website):
        try:
            #Make a request to the url and read in the webpage, set SSL verificaiton to False (not transmitting sensitive info)
            self.html = requests.get(url, verify = False).text
            #Make soup!
            self.soup = BeautifulSoup(self.html, 'html.parser')
            #Extract all text from the website
            self.text = self.soup.get_text()
            #Set flags depending on which website is being processed
            if website == 'forbes':
                self.flag = 1
            if website == 'cnbc':
                self.flag = 2
            if website == 'cointelegraph':
                self.flag = 3
            if website == 'bitcoin':
                self.flag = 4
            if website == 'coindesk':
                self.flag = 5
        except:
            print('Please enter a valid URL; {}'.format(url))


    def page_numbers(self):
        #ONLY FOR FORBES SITES - Forbes websites has multiple pages for their articles
        #Extract how many pages there are for an article
        try:
            if self.flag is 1:
                #Example of Match Pattern: 'Page 1 / 3'
                #Search for pattern and return last value as total pages for the article
                total_pages = re.findall('Page\s[0-9]\s/\s([0-9])',self.text)[0]
                return int(total_pages)
            else:
                return None
        except:
            return None

    def get_headline(self):
        #Extract the headline of the article
        #Headlines for Coindesk articles are under the <title> tag
        if self.flag is 5:
            headline = self.soup.title.text
        #Headlines for all other websites are under the <h1> tag
        else:
            headline = self.soup.h1.text
        return headline

    def date_and_time(self):
        #Forbes date and time
        if self.flag == 1:
            '''date = re.search('[A-Za-z]{3}\s[0-9]+,\s[0-9]{4}\s@\s[0-9]+:[0-9]{2}\s[A-Z]{2}',self.text)
            date_time_object = date.group().split('@')
            date_time = datetime.strptime(''.join(date_time_object).replace(',',''), '%b %d %Y  %I:%M %p')'''
            date = ''.join(self.soup.time.text.replace(',',''))
            date_time = datetime.strptime(date, '%b %d %Y  %I:%M %p')

            return date_time
        #CNBC date and time
        if self.flag == 2:
            date_time_object = self.soup.time.text.split()

            if len(date_time_object) != 8:
                order =  [0,1,2,4,5]
                day = [date_time_object[i] for i in order]
                try:
                    date_time = datetime.strptime(' '.join(day),'%b %d, %Y %I:%M %p')
                except:
                    date_time = datetime.strptime(' '.join(day),'%B %d, %Y %I:%M %p')
                return date_time
            else:
                order = [6,5,-1,1,2]
                date_time_list = [date_time_object[i] for i in order]
                try:
                    date_time = datetime.strptime(' '.join(date_time_list),'%b %d %Y %I:%M %p')
                except:
                    date_time = datetime.strptime(' '.join(date_time_list),'%B %d %Y %I:%M %p')
                return date_time
        #Cointelegraph date and time
        if self.flag == 3:
            tag = self.soup.find('div', class_ = 'date')
            date_time_object = tag['datetime']
            date_time = datetime.strptime(date_time_object, '%Y-%m-%d %H:%M:%S')
            return date_time

        #Coindesk date and time
        if self.flag == 5:
            date_time_object = self.soup.find(attrs={"class": "article-container-left-timestamp"}).text.split('UTC')[0].replace('\n\n','').strip().replace('at','').replace(',','')
            d = date_time_object.split()
            idx = [2,0,1,3]
            date_time_object = ' '.join([d[i] for i in idx])
            date_time = datetime.strptime(date_time_object,'%Y %b %d %H:%M')
            return date_time

        #Bitcoin date and time
        if self.flag ==  4:
            tag = self.soup.time
            date_time_object = tag['datetime'].replace('T',' ').split('+')[0]
            date_time = datetime.strptime(date_time_object,'%Y-%m-%d %H:%M:%S')
            return date_time

    def raw_text(self):
        #Extract all <p> tags from articles - actual text of each website
        p_tags = self.soup.find_all('p')
        #create a list of all found <p> tags
        text = [item.text for item in p_tags]
        #Join all <p> tags to reconstruct the paragraphs of the articles
        return(' '.join(text))
```

APPENDIX B – CUSTOM SENTIMENT CLASS

```
from textblob import TextBlob
import nltk
from nltk.corpus import stopwords
import string
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk import word_tokenize
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from nltk.stem.lancaster import LancasterStemmer
from nltk.stem.porter import PorterStemmer
from nltk.stem.snowball import SnowballStemmer
from nltk.stem.wordnet import WordNetLemmatizer

class sentiment_analysis:

    def __init__(self,text,headline = False):
        #INPUT -> Article Headline
        if headline is True:
            self.text = text.lower()
            self.blob = TextBlob(text.lower())
        #INPUT -> List of Sentences
        elif isinstance(text,list):
            self.sentences = [word.lower() for word in text]
        #INPUT -> String of Sentences
        else:
            self.text = text[:-1].lower()
            self.blob = TextBlob(text)

    def words_only(self):
        try:
            self.text = ' '.join(self.sentences)
            words_only = re.sub('[^A-Za-z]', ' ', self.text)
            return sentiment_analysis(words_only)
        except:
            words_only = re.sub('[^A-Za-z]', ' ', self.text)
            return sentiment_analysis(words_only)

    def stemmed(self):
        words = self.text.split()
        lancaster = LancasterStemmer()
        porter = PorterStemmer()
        snowball = SnowballStemmer('english')

        stemmed_text = []
        for word in words:
            stems = [lancaster.stem(word), porter.stem(word), snowball.stem(word)]
            stemmed_word = max(set(stems), key=stems.count)
            stemmed_text.append(stemmed_word)
        return sentiment_analysis(' '.join(stemmed_text))

    def lemmatize(self):
        words = self.text.split()
        wordnet = WordNetLemmatizer()
        lem = [wordnet.lemmatize(word) for word in words]
        return sentiment_analysis(' '.join(lem))


    def remove_stopwords(self, remove_punc = False):
        #INPUT -> string of sentences or headline
        #OUTPUT -> string of sentences or headline with stopwords and/or punctuation removed
        if hasattr(self, 'text'):
            #if user wants punctuation removed in conjuction with removing stop words
            if remove_punc == True:
                self.remove_punctuation = [char for char in self.text if char not in string.punctuation]
                self.no_punctuation = ''.join(self.remove_punctuation)
                self.no_stopwords = [word for word in self.no_punctuation.split() if word.lower() not in stopwords.words('english')]
                return sentiment_analysis(' '.join(self.no_stopwords))
            #if user only wants to remove stop words
            else:
                word = nltk.word_tokenize(self.text)
                self.no_stopwords = [word for word in word if word not in set(stopwords.words('english'))]
                return sentiment_analysis(' '.join(self.no_stopwords))

        #INPUT -> list of sentences
        #OUTPUT -> list of sentences with stopwords and/or punctuation removed
        else:
            self.list_no_stopwords = []
            #if user wants punctuation removed in conjuction with removing stop words
            if remove_punc == True:
                for sentence in self.sentences:
                    self.remove_punctuation = [char for char in sentence if char not in string.punctuation]
                    self.no_punctuation = ''.join(self.remove_punctuation)
                    self.no_stopwords = [word for word in self.no_punctuation.split() if word.lower() not in stopwords.words('english')]
                    self.list_no_stopwords.append(' '.join(self.no_stopwords))
                return sentiment_analysis(self.list_no_stopwords)
            #if user only wants to rmove stop words
            else:
                for sentence in self.sentences:
                    word = nltk.word_tokenize(sentence)
                    self.no_stopwords = [word for word in word if word.lower() not in stopwords.words('english')]
                    self.list_no_stopwords.append(' '.join(self.no_stopwords))
                return sentiment_analysis(self.list_no_stopwords)


    def darth_vader(self):
        #INPUT -> list of sentences
        #OUPUT -> vader sentiment analysis for each sentence in the list
        try:
            sid = SentimentIntensityAnalyzer()
            self.vader_scores = [sid.polarity_scores(sentence) for sentence in self.sentences]
            return self.vader_scores
        #INPUT -> sentence string
        #OUTPUT -> vader sentiment analysis for the string
        except:
            sid = SentimentIntensityAnalyzer()
            self.vader_score = sid.polarity_scores(self.text)
            return self.vader_score



    def blobbed(self):
        #INPUT -> list of sentences
        #OUPUT -> TextBlob sentiment analysis for each sentence in the list
        try:
            blobbed_sentences = [TextBlob(sentence) for sentence in self.sentences]
            self.blob_scores = [sentence.sentiment for sentence in blobbed_sentences]
            return self.blob_scores
        #INPUT -> sentence string
        #OUTPUT -> TextBlob sentiment analysis for the string
        except:
            blobbed_headline = TextBlob(self.text)
            self.blob_score = blobbed_headline.sentiment
            return self.blob_score


    #INPUT -> string of sentences or a single sentence
    #OUTPUT -> word cloud composed of words from the string input
    def word_cloud(self):
        wordcloud = WordCloud(background_color = 'white',
                              max_words = 200,
                              max_font_size = 40,
                             scale = 3,
                              random_state = 1).generate(self.text)
        fig = plt.figure(figsize=(15,15))
        self.ax = plt.imshow(wordcloud)
        plt.axis('off')
        plt.show()
        return self.ax
```

APPENDIX C – POSITIVE WORD LIBRARY

climbs
climbing
climbed
climb
surge
surges
surged
surging
surgin
surg
up
big
leap
leaps
jumps
jump
jum
higher
high
growth
unprecedented
unpreced
gains
gain
fortune
earned
earn
skyrocketed
millionaires
lucky
valuable
wealthiest
surpassed
richest
richahead
headlines
rich
largest
huge
skyrocket
lucky
lucki
billionair
billionaire
surpass
richest
ahead
return
rich
spiked
spike
spikes
spiking
billion
billions

useful
spiked
spike
strong
increase
rise
recovered
recover
recov
new
partnership
bull
optimistic
stong
increase
value
serious
posit
surgeasy
easier
easi
run
advantage
superior
partnership
partnering
risen
strong
climbed
climb
rising
rise
fantastic
fantast
inch
closer
close
explosive
explos
gains
gain
growth
jumping
jump
historic
histor
milestone
mileston
premier
record
quick
cost-effective
astronomical
approaching
approach

benefited
benefit
high
additional
stable
safe
appealing
appeal
soar
driving
drive
record
moon
gain
increase
increas
up
stabl
appeal
buzz
thriving
thrive
banner
leapt
leap
grew
shot
progressing
progress
astounding
astound
above
explosion
unprecedented
newcomers
popular
record
reliable
mad
comfortable
opportunity
uptick
frenzy
useful
appealing
big
jump
fascinating
fascinate
explos
unpreced
hord
newcom
ramp
up

opportun
winner
win
boom
uptick
surpass
frenzi
largest
large
rose
rise
climbed
climb
blockbuster
appreciating
appreciate
rally
rallies
rallying
increase
momentum
robust
benefit
benefitting
interested
interest
blockbust
appreci
ralli
increas
benefit
swell
strong
upward
milestones
milestone
appreciation
partnership
partnering
boost
benefits
benefit
positive
fomo
appreci
higher
raised
raise
rally
admire
admirable
attract
attracted
quick
quickly

cheap
cheaply
testing
test
driving
drive
velocity
bull
bullish
best
milestone
rich
richer
richest
soared
soar
outpacing
outpace
outpac
best-performing
overtook
overtake
increase
increas
growing
grow
cement
cemented
attract
attracted
fuel
recovery
recover
recovered
impress
impressive
optimistic
upside
implement
implementing
overtake
overtaken
overtaking
rebound
rebounds
rebounding
strong
stronger
bargain
bargains
testing
heights

APPENDIX C (CONT) – NEGATIVE WORD LIBRARY

loss
losses
lose
loser
drop
dropped
drops
ban
banned
banning
crackdown
worst
worst-performing
worthless
fell
fall
decrease
weak
weakening
weakened
less
volatile
hard
fade
fades
decline
declined
declining
down
bear
bearish
bear
bears
panic
bad
skeptical
worse
artificial
problem
problems
risk
risky
fallen
falling
fallin

losing
declines
correction
fades
fading
fade
fad
correcting
pullback
retract
shed
lower
low
lowest
retractment
uncertain
uncertainty
adverse
ban
banning
banned
fud
pull
back
low
lows
decline
pullback
fade
fading
faded
dumping
dumped
dump
corrections
below
restrict
restricting
restrictions
regulation
regulating
regulate
red
downward
crack

spooked
spook
suffer
suffering
suffered
underperformance
underperforming
underperform
overvalued
overvalue
plunge
plunging
dips
dip
dipping
depreciating
depreciate
depreciated
nervous
losing
suffered
suffer
suffering
shaken
shaking
shake
dips
dipping
dipped
shaken
shy
shaky
skittish
volatile
tumultuous
headwinds
headwind
negative
battered
batter
banned
ban
banning
collapsed
collapsing

collapse
regulating
regulate
regulated
oversold
selloff
bubble
crushed
crush
burst
bursting
pop
popping
popped
worse
worst
worsen
crashing
crashed
crash
exclude
excluding
removed
remove
removes
complain
complains
complained
failure
failures
failed
fail
resist
resistance
resisting
resisted
weak
turmoil
difficult
slump
slumped
slumping
failed
failing
fail

removing
restrict
restricting
regulating
regulate
regulations
sink
sinking
sinks
struggle
struggles
struggling
sell
invalidate
invalidated
tricky
reluctant
reluctance
barrier
barriers
excludes
exclude
fomo
fear
risk
risks
risking
vanish
vanishing
bubble
tumble
tumbles
tumbled
flee
nervous
freeze
freez
red
decline
disappoint
disappointment
disappointing
scrutiny

APPENDIX D – CUSTOM SENTIMENT ANALYSIS ALGORITHM

```
from collections import namedtuple
Sentiment = namedtuple('Sentiment', ['Strength_of_Positivity', 'Strength_of_Negativity'])

#Load the negative cryptocurrency word library
with open('negative.txt') as fh:
    file = fh.readlines()
    negatives = [line.strip() for line in file]

#load the positive cryptocurrency word library
with open('positive.txt') as fh:
    file = fh.readlines()
    positives = [line.strip() for line in file]


########### Custom Sentiment Analysis
def coin_sentiment(text, headline = False):
    #Create sentiment_analysis object with text
    text = sentiment_analysis(text)

    #Initialize dictionary to keep track of sentence statistics
    sentence_tracker = {'neg':{'score':0, 'sentences':0},
                    'pos':{'score':0, 'sentences':0},
                    'neu':{'sentences':0}}

    #Break the text into individual sentences and loop through each sentence
    for sent in text.blob.sentences:
        #Remove all punctuation and numbers then split the sentence into individual words
        words = sentiment_analysis(str(sent)).words_only().text.split()

        neg = 0
        pos = 0
        #Loop through each word in the sentence
        for word in words:
            #check if word is in the "negatives" keywords list
            if word in negatives:
                neg -= 1
            #check if word is in the "positives" keywords list
            if word in positives:
                pos += 1
        #Calculate the score for current sentence to determine if sentence overall is positive, negative, or neutral
        sentence_score = pos + neg

        #Negative sentence
        if sentence_score < 0:
            #running total of negative sentences in the text
            sentence_tracker['neg']['sentences'] += 1
            #running total of negative score of text
            sentence_tracker['neg']['score'] += sentence_score
        #Neutral sentence
        elif sentence_score == 0:
            #running total of neutral sentences in the text
            sentence_tracker['neu']['sentences'] += 1
        #Positive sentence
        else:
            #running total of positive sentences in the text
            sentence_tracker['pos']['sentences'] += 1
            #running total of positive scores of text
            sentence_tracker['pos']['score'] += sentence_score


    if headline is True:
        positive_sentiment = sentence_tracker['pos']['score']
        negative_sentiment = sentence_tracker['neg']['score']
        return (positive_sentiment, negative_sentiment)

    else:
        #Total number of sentences analyzied in the text
        number_of_sentences = sentence_tracker['neg']['sentences'] + sentence_tracker['neu']['sentences'] + sentence_tracker['pos']['sentences']

        #Total number of points in text AKA total number of positive and negative keywords in text
        total_points = sentence_tracker['pos']['score'] + abs(sentence_tracker['neg']['score'])

        #Percentage of positive sentences
        pct_pos = sentence_tracker['pos']['sentences'] / number_of_sentences

        #Percentage of negative sentences
        pct_neg = sentence_tracker['neg']['sentences'] / number_of_sentences

        #Percentage of neutral sentences
        pct_neu = sentence_tracker['neu']['sentences'] / number_of_sentences

        positive_sentiment = sentence_tracker['pos']['score'] * pct_pos / total_points
        negative_sentiment = sentence_tracker['neg']['score'] * pct_neg / total_points
        sentiment = Sentiment(positive_sentiment, negative_sentiment)
        return sentiment
```

APPENDIX E – NEURAL NETWORK MODEL TRAINING

```
def dnn_model(X, Y, layers_dims, learning_rate = 0.0075, lambd = 0.1, num_iterations = 3000, print_cost = True):
    '''
    Implements a L-layer neural network: [LINEAR->RELU]*(L-1)->LINEAR->SIGMOID.

    Arguements:
    X -- data, numpy array of shape (number of examples, number of features)
    Y -- true "label" vector of shape (1, number of examples)
    layers_dims -- list containing the input size and each layer size of length (number of layers + 1)
    learning_rate -- learning rate of the gradient descent update rule
    num_iterations -- number of iterations of the optimization loop
    print_cost -- if True, prints the cost every 100 steps

    Returns:
    parameters -- parameters learned by the model. They can then be used to predict.
    '''
    np.random.seed(1)
    costs = []

    #initialize parameters
    parameters = initialize_parameters_deep(layers_dims)

    #Loop (gradient descent)
    for i in range(0, num_iterations):

        #Forward Propagation: [LINEAR->RELU]*(L-1) -> LINEAR -> SIGMOID.
        AL, caches = L_model_forward(X, parameters, layers_dims)


        #Compute Cost
        cost = compute_cost(AL,Y,lambd,caches)

        #Backward Propagation
        grads = L_model_backward(AL, Y, lambd, caches)

        #Update Parameters
        parameters = update_parameters(parameters, grads, learning_rate)

        #Print cost every 100 training examples
        if print_cost and i % 200 == 0:
            print('Cost after iteration %i: %f' %(i, cost))
        #if print_cost and i % 10 == 0:
        costs.append(cost)
    fig = plt.figure(figsize = (15,8))
    plt.plot(np.squeeze(costs))
    plt.ylabel('Cost')
    plt.xlabel('iterations')
    plt.title('Model: '+ str(layers_dims)+' Learning rate: ' + str(learning_rate)+' Lambda: ' + str(lambd))
    plt.show()

    return parameters
```