ALCX_Twitter

April 12, 2021

0.1 Alchemix Analytics Question: Bounty #13

Author Discord Username: dtradke7

Date: April 11, 2021

0.1.1 Introduction

This notebook is authored for the intention of answering question 13 for the Alchemix Analytics Questions, found at the following link (https://www.notion.so/Analytics-Bounties-for-Alchemix-ffd6c25bdef3486c9b8dfa9476ac6a39). The substance of the question involves finding relationships between Twitter activity and the Alchemix cryptocurrency (ALCX). The question was purposely vague to allow answers to be widespread. I used the Tweepy Python library to collect Tweets from Twitter, and Numpy and Pandas Python libraries for data analysis. I performed a SQL query on the Flipside Crypto Ethereum Price database to collect the price of ALCX hourly for the previous week. It is important to understand that the notebook works with real-time data, however you are required to have a personal Twitter developer account for API tokens. For this reason, I have added additional functionality to save and read CSV files containing the same data as real-time price and Twitter data. This addition makes the notebook functional to the everyday user with the provided offline data.

The substance of my insights can be organized in four points:

1. Impact of the main @AlchemixFi twitter account activity on coin price.
2. How the quantity of tweets containing the string "ALCX" impacts the coin price.
3. How the sentiment of tweets containing the string "ALCX" impacts the coin price.
4. How the reach, or number of followers exposed to each tweet containing "ALCX" impacts the coin price.

Throughout this notebook, I will work to analyze how these factors may influence the price of the Alchemix cryptocurrency. As a direct impact of this study, I expect investors to follow certain trends and highly influential coinholders to be cognisant of their activities on social media regarding ALCX.

The first part of my code is necessary to import the libraries that I will use throughout the notebook. These libraries must be installed on your computer through the command line interface, typically using Pip (i.e. pip3 install numpy).

[1]:
```
# Import libraries
import re, tweepy, datetime, time, csv
```

```
from tweepy import OAuthHandler
from textblob import TextBlob
import matplotlib.pyplot as plt
import pandas as pd
import urllib.request as rq
import numpy as np
import json
import os
```

The below function loads the ALCX data from the Flipside crypto dataset. The data is either loaded in real-time, or a CSV of data corresponding to the historical data when the tweets were collected is loaded using the JSON library. Once loaded, I parse the data into arrays and dictionaries corresponding with separate days of the month, and the data is then sorted accordingly to reflect the actual price of ALCX. This function returns a dictionary with information about each day, such as the price of ALCX (hour resolution), year, and month.

[2]:
```
# function to load price data for ALCX from Flipside dataset

def getPrices(path):

    if os.path.exists(path):
        with open(path, 'r') as fp:
            dataset = json.load(fp)
    else:
        url = 'https://api.flipsidecrypto.com/api/v2/queries/
↪f75196fc-c9a3-4b97-beb4-5437d5fd1f90/data/latest'
        try:
            dataset = rq.urlopen(url)
            dataset = dataset.read()
            dataset = json.loads(dataset)
        except Exception as e:
            print('Unable to get data from flipsidecrypto API. Check the URL␣
↪below: \n{}'.format(url))

        with open(path, 'w') as fp:
            json.dump(dataset, fp)

    price, time = [], []
    year, month, day, hour = [], [], [], []
    for i, val in enumerate(dataset):
        year.append(int(val["HOUR"][:4]))
        price.append(float(val["PRICE"]))
        time.append(val["HOUR"])
        hour.append(int(val["HOUR"][11:13]))
        day.append(int(val["HOUR"][8:10]))
        month.append(val["HOUR"][5:7])
```

```
    year = np.array(year)
    month = np.array(month)
    hour = np.array(hour)
    day = np.array(day)
    price = np.array(price)

    day_dict = {}
    for i in range(np.amin(day), np.amax(day)):
        today_year = year[day==i]
        today_month = month[day==i]
        today_hours = hour[day == i]
        today_day = day[day==i]
        today_prices = price[day == i]
        p = today_hours.argsort()
        sorted_hours = today_hours[p]
        sorted_prices = today_prices[p]
        day_dict[i] = {'year': today_year[0], 'month':today_month[0], 'hours':
↪sorted_hours, 'prices':sorted_prices, 'tweets':[]}

    return day_dict
```

This function helps clean the data and line up the Twitter attributes with the days and hours corresponding to the ALCX price data from Flipside Crypto dataset. The function takes in a dictionary of price data, a Pandas DataFrame containing tweets and their features, and a boolean flag corresponding with sentiment analysis. The function returns four dictionaries with data to be further analyzed with reference to the ALCX price data.

[3]:
```
# Function to clean and line-up data
def matchTweetsPrices(day_dict, tweets_df, get_sentiment=False):
    tweet_amt = {}
    tweet_reach = {}
    tweet_sentiment_dict = {}
    for i in day_dict.keys():
        day = str(i).zfill(2)
        # print(day)
        tweet_timestamp = str(day_dict[i]['year'])+str(day_dict[i]['month'])+day
        tweet_timestamp = int(tweet_timestamp)

        tweet_dates = np.array(tweets_df['date'])
        tweet_amt[i] = np.count_nonzero(tweet_dates == tweet_timestamp)

        followers = np.array(tweets_df['followers'])
        tweet_reach[i] = np.sum(followers[tweet_dates == tweet_timestamp])

        if get_sentiment:
            tweet_sentiments = np.array(tweets_df['sentiment'])
            try:
```

```
                tweet_sentiment_dict[i] = round(np.
↪mean(tweet_sentiments[(tweet_dates == tweet_timestamp) & (tweet_sentiments !
↪= 0)]),2)
            except:
                tweet_sentiment_dict[i] = 0

    return day_dict, tweet_amt, tweet_sentiment_dict, tweet_reach
```

The below code is used to plot the collected data and help provide further insights into the correlations of Twitter data and the price of ALCX. Shown later in the notebook, the plots created using this function show a line graph of the ALCX price with daily background coloring corresponding with the variables we are interested in. Along the top of each chart, we explicitely list the numerical value of each variable. Note that we also normalize the shading of these variables so that the lowest value in the chart is 0 (white), and the highest value is 1 (dark blue).

[4]:
```
# function to plot our data. This will be used for multiple aspect of price and␣
↪twitter data
def plotTweetAmts(day_dict, tweet_char, username, sentiment=False, reach=False):
    fig=plt.figure(figsize=(10, 6))
    prices = []
    for i in day_dict.keys():
        prices.append(day_dict[i]["prices"])

    prices = np.concatenate(prices, axis=0)

    # y axis
    plt.yticks(fontsize=14)
    plt.ylabel("Price (USD)", fontsize=16)


    # tweet amount normalized
    tweet_char = np.array(list(tweet_char.values()))
    tweet_char[np.isnan(tweet_char)] = 0
    tweet_char_normalized = (tweet_char - np.amin(tweet_char)) / (np.
↪amax(tweet_char) - np.amin(tweet_char))


    for i in range(1, len(list(day_dict.keys()))+1):
        left = prices.size-(24*i)
        right = prices.size-(24*(i-1))
        if left < 0: left = 0
        plt.axvspan(prices.size-(24*i), prices.size-(24*(i-1)), facecolor='b',␣
↪alpha=tweet_char_normalized[-1*i])

    # x axis
    label_idx, label_str = [], []
    for i in range(prices.size,0,-1):
```

```
        if (prices.size - i)%24 == 0:
            label_idx.append(i)
            label_str.append(str(-1*int((prices.size - i)//24)))
            # print("here: ", -1*int((prices.size - i)//24))
            if sentiment:
                shift = 20
            elif reach:
                shift = 21
            else:
                shift = 15
            if reach:
                col_text = str(round(tweet_char[-1*(1+int((prices.size - i)//
↪24))] / 1000,1))+"k"
            else:
                col_text = str(tweet_char[-1*(1+int((prices.size - i)//24))])

            plt.text(i-shift, np.amax(prices)-5, col_text, fontsize=15,␣
↪bbox=dict(facecolor='w', edgecolor='k', pad=3.0))

    plt.xticks(label_idx, label_str, fontsize=14)
    plt.xlabel("Previous Days", fontsize=16)

    plt.plot(np.arange(prices.shape[0]), prices, c='r', label='ALCX Price')
    plt.legend(loc='lower left', fontsize=18)
    if sentiment: plt.title(username+" Tweets Sentiment Effect on ALCX Price",␣
↪fontsize=18)
    elif reach: plt.title("Reach (Followers) of ALCX Tweets Effect on ALCX␣
↪Price", fontsize=18)
    else: plt.title(username+" Tweet Amount Effect on ALCX Price", fontsize=18)
    plt.show()
```

The code block below contains the TwitterClient class. This class has functions which are used to access the Twitter API and scrape tweets from particular users or containing specific phrases or strings. The various functions in this class also clean the tweets, and uses TextBlob to perform sentiment analysis. If you would like to use this code for real-time data, you need to sign up for a Twitter developer account and insert your own keys and tokens below for the variables: consumer_key, consumer_secret, access_token, and access_token_secret.

[5]:
```
# Create twitter client class
class TwitterClient(object):

    def __init__(self):
#         need to add your own keys here for the Twitter API
        consumer_key = ""
        consumer_secret = ""
        access_token = ""
        access_token_secret = ""
```

```
        try:
            self.auth = OAuthHandler(consumer_key, consumer_secret)
            self.auth.set_access_token(access_token, access_token_secret)
            self.api = tweepy.API(self.auth)

        except:
            print("Error: Authentication Failed")

    def clean_tweet(self, tweet):
        '''
        Cleans the tweet text
        '''
        return ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/
↪\S+)", " ", tweet).split())

    def get_tweet_sentiment(self, tweet):
        '''
        Uses TextBlob to collect sentiment of tweet, assigns real number label:
        negative = -1,
        neutral = 0,
        positive = 1
        '''
        # create TextBlob object of passed tweet text
        analysis = TextBlob(self.clean_tweet(tweet))
        # set sentiment
        if analysis.sentiment.polarity > 0:
            return 1
        elif analysis.sentiment.polarity == 0:
            return 0
        else:
            return -1

    def getUserTweets(self, username, count=10):
        try:
            # Creation of query method using parameters
            tweets = tweepy.Cursor(self.api.user_timeline,id=username).
↪items(count)

            # Pulling information from tweets iterable object
            tweets_list = []
            for tweet in tweets:
                timestamp = str(tweet.created_at).split(" ")
                sep = ""
                tweet_date = int(sep.join(timestamp[0].split("-")))
                time_clock = timestamp[1]
```

```
                tweets_list.append([tweet_date, time_clock, tweet.id, tweet.
↪user.followers_count, tweet.text])
            # tweets_list = [[tweet.created_at, tweet.id, tweet.text] for tweet␣
↪in tweets]

            # Creation of dataframe from tweets list
            # Add or remove columns as you remove tweet information
            tweets_df = pd.DataFrame(tweets_list)
        except BaseException as e:
            print('failed on_status,',str(e))
            time.sleep(3)

        return tweets_df

    def getTextSearchTweets(self, text_query, count=1000):
        today = datetime.datetime.now()
        today = today.replace(hour=23, minute=59, second=59,␣
↪microsecond=999999) # set from the beggining of the day
        tweets_df = pd.DataFrame(columns = ['date','time', 'id', 'followers',␣
↪'tweet', 'sentiment'])

        week = np.arange(8)
        for d in week:
            time_to_the_past = int(d)
            yesterday = today - datetime.timedelta(time_to_the_past)
            # next_day = yesterday + datetime.timedelta(time_to_the_past) #␣
↪equivalent to today

            try:
                # Creation of query method using parameters
                tweets = tweepy.Cursor(self.api.search,q=text_query,until =␣
↪yesterday.date()).items(count)

                # Pulling information from tweets iterable object
                tweets_list = []
                for tweet in tweets:
                    timestamp = str(tweet.created_at).split(" ")
                    sep = ""
                    tweet_date = int(sep.join(timestamp[0].split("-")))
                    time_clock = timestamp[1]
                    tweets_df.loc[len(tweets_df)] = [tweet_date, time_clock,␣
↪tweet.id, tweet.user.followers_count, tweet.text, self.
↪get_tweet_sentiment(tweet.text)]


            except BaseException as e:
```

```
                print('failed on_status,',str(e))
                time.sleep(3)

        return tweets_df
```

The below function is used to load the previous 100 tweets by any Twitter account, and save the date, time, ID, number of followers of the account, and the tweet for each tweet. In this notebook, I only collect the tweets from the @AlchemixFi Twitter account, however this function is general and could be used for any account. I also save the data to a CSV for future offline use without a Twitter developer account.

[6]:
```
# function to load @AlchemixFi tweets
def loadDFAlchemix(api, path, username):
    if os.path.exists(path):
        tweets_df = pd.read_csv(path, index_col=0)
    else:
        tweets_df = api.getUserTweets(username, count=100)
        tweets_df.columns = ['date','time', 'id', 'followers', 'tweet']
        tweets_df.to_csv("data/AlchemixFi_tweets.csv")
    return tweets_df
```

The below function is used to load the previous tweets containing a certain string. I use this function to collect "count" amount of tweets each day which contain "ALCX" to gain an understanding of how many accounts are conversing about Alchemix. Additionally, we record the calculated sentiment of each tweet using the TextBlob library.

[7]:
```
# function to load tweets containing 'ALCX'
def loadDFALCX(api, path, count):
    if os.path.exists(path):
        tweets_df = pd.read_csv(path, index_col=0)
    else:
        tweets_df = api.getTextSearchTweets(text_query='ALCX', count=count)
        tweets_df.columns = ['date','time', 'id', 'followers', 'tweet',␣
 ↪'sentiment']
        tweets_df.to_csv("data/ALCX_tweets"+str(count)+".csv")
    return tweets_df
```

Now that we have gone through all of the functions that clean, format, and calculate our data, we are ready to show how the outputs of these functions are used to provide important insights. For the remainder of the notebook, we will perform queries to the above functions and show visual and analytical insights into our aggregated Twitter and ALCX price data. First, we will create a Twitter client object called "api".

[8]:
```
# create twitter client object
api = TwitterClient()
```

Next, we must load the Flipside Crypto dataset (or the saved JSON) using the getPrices function

above. I pass the path to the saved JSON file, and if it does not exist, the online most recent 7 days of data is loaded. Loading new data should only be done if you have a Twitter developer account so the tweets and price correspond with the same timeframe.

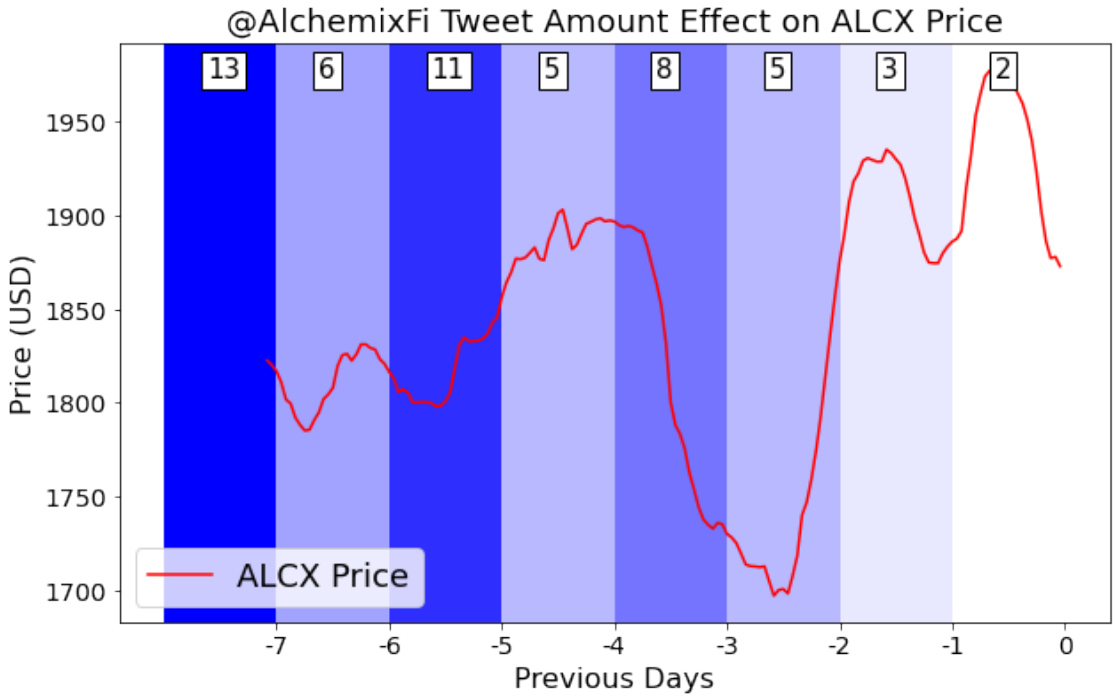
[9]:
```
# load prices from Flipside dataset
# NOTE: in many cases, I am loading a .csv to account for twitter requiring a␣
 ↪developer account to load new tweets
# This means if YOU have a developer account, you can add your 'keys' and load␣
 ↪current data
path = 'data/ALCX_prices.json'
day_dict = getPrices(path)
```

0.1.2 Experiment 1: Affect of @AlchemixFi Activity

Now that I have loaded the ALCX price data, I conduct the first experiment focused on how the activity of the @AlchemixFi Twitter account affects the price of ALCX. The first step is to load the previous tweets by @AlchemixFi, done with the function calls below. We pass the path to a CSV file which corresponds to the previous 7 days before this notebook was created for people who do not have a Twitter developer account.

Results The plot below shows the results from this experiment. The y-axis corresponds with the price of ALCX in USD, and the x-axis corresponds with time, the previous 7 days. I am limited to the previous 7 days of Twitter data as per the Twitter API limitations. The red line corresponds with the actual price of ALCX (in USD), and the blue shading corresponds with the amount of tweets by @AlchemixFi which is normalized so that the most tweets in a day (13) is the darkest blue, and the lowest amount (2) is white. Viewing the graph, it is not particularly obvious if there is a correlation between the tweet activity of @AlchemixFi and the price of ALCX. One trend that is particularly noticeable is that the amount of tweets by @AlchemixFi decreases with time. This corresponds with the price of ALCX fluctuating by 13%, or around $250 USD. While this trend would need more data to verify, it appears that the volume of outgoing tweets by @AlchemixFi may influence the stability of the price of ALCX.

[10]:
```
path = 'data/AlchemixFi_tweets.csv'
tweets_df = loadDFAlchemix(api, path,username="AlchemixFi")
day_dict, tweet_amt,_, tweet_reach = matchTweetsPrices(day_dict, tweets_df)
plotTweetAmts(day_dict, tweet_amt, username='@AlchemixFi')
```



0.1.3 Experiment 2: Tweet Volume Effect

Moving beyond the @AlchemixFi twitter account, I now source all of the tweets within the previous 7 days containing the string "ALCX" to analyze if there is a correlation between overall tweet volume and price. I set the number of previous tweets to be 50,000 to more than cover the previous 7 days, while staying within the Twitter data limit. Executing this code with the attached CSV will grab offline historical tweets, whereas if you have a Twitter Developer account, you can run this code with online data.

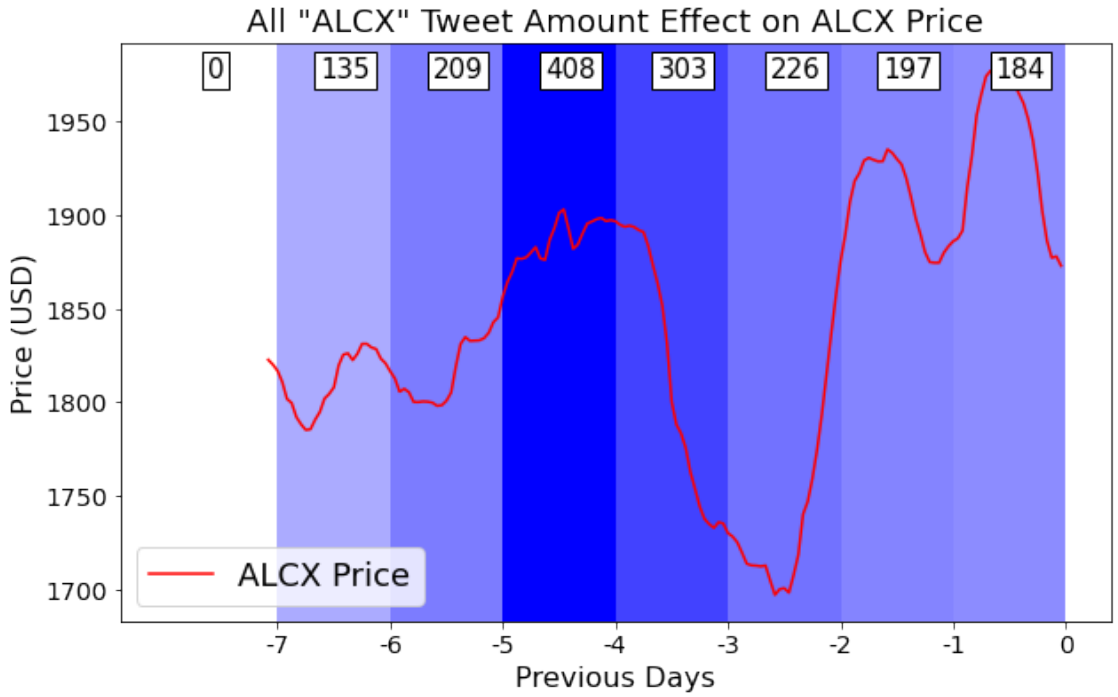
Results The graph below has the same attributes as above, however the shading and numerical values correspond with the volume of tweets each day contianing "ALCX". It is obvious that the most tweets about the currency come just before the moment when the price dropped $200 USD. There also tend to be less tweets about the currency when it is most vaiable, seen in the beginning and end of this 7 day period. This is an interesting result, as one would expect people to be discussing the currency while the price changes.

One potential hypothesis for this comes from the Theory of Mind in Artificial Intelligence (AI) and Cognitive Science, which revolves around a person (or agent) modeling the potential strategical behavior of other humans or agents. With "opponent modeling", currency holders may be tweeting less to observe what the community believes instead of revealing their own beliefs about the current state of the price. A result of this observation is that one may expect large price variance depending on lower volumes of twitter activity, and adjus their position accordingly.

[11]:
```
# now we want to load ALL user's tweets about $ALCX
count = 50000 # number of previous tweets
```

```
path = 'data/ALCX_tweets'+str(count)+'.csv'
tweets_df = loadDFALCX(api, path, count)
```

[12]:
```
# The number of tweets each day containing the string "ALCX"
day_dict, tweet_amt, tweet_sentiment, tweet_reach = matchTweetsPrices(day_dict,␣
↪tweets_df)
plotTweetAmts(day_dict, tweet_amt, username='All "ALCX"')
```



0.1.4 Experiment 3: Effect of Tweet Sentiment on Price of ALCX

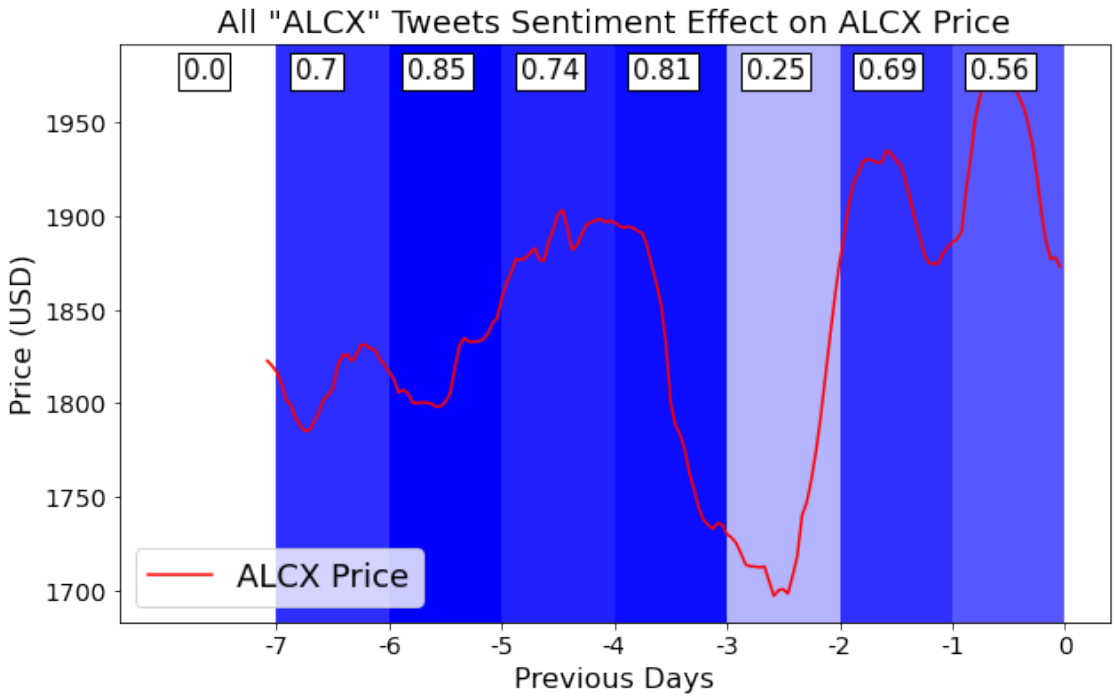
Sentiment analysis is a populat method in the AI field of Natual Language Processing (NLP). Twitter data provides a large corpus of textual data for NLP tasks in both supervised and unsupervised learning. While more recent models such as Transformer deep learning models and GPT-3 from OpenAI perform staggering NLP tasks, sentiment analysis can still be done using other AI methods such as N-Grams which are more traditional. I use the TextBlob library for sentiment analysis which uses N-Grams and other light weight NLP methods for sentiment analysis to avoid large and expensive deep learning models. This could be left for future work, as models such as BERT and other Transformers could achieve higher classification accuracy.

Results The x and y axes are the same as the previous two graphs, however the shading now corresponds with the mean sentiment of tweets throughout each day, where 1 is "positive", "-1" is negative, and the shades of blue are once again normalized. The first staggering observation is how positive the sentiment is about ALCX tweets, a 7-day average of 0.66 (ignoring the first day due to lack of data), showing the optimism about the currency in the community discussing it on Twitter.

It is interesting to note that the least positive tweets about ALCX come just after a drop of almost $200 USD in price, which is intuitive. Perhaps this shows the mindset of the community to act negatively to any downturn in price, despite the opportunity to "buy the dip" and achieve more future gains from the lower cost. Surprisingly, the mean sentiment of tweets does not return to the level before the dip despite the price being higher on average. Perhaps this shows how the fluctuation in the previous 2 days makes the community less optimistic, potentially leading to the fluctuations themselves in the first place. While this sentiment analysis is interesting, more advanced AI models could lead to a more accurate assessment of the communities response to fluctuation.

[13]:
```
# How the sentiment of tweets affects the price
day_dict, tweet_amt, tweet_sentiment, tweet_reach = matchTweetsPrices(day_dict,␣
 ↪tweets_df, get_sentiment=True)
plotTweetAmts(day_dict, tweet_sentiment, username='All "ALCX"', sentiment=True)
```

```
/Users/dtradke/.pyenv/versions/3.8.6/lib/python3.8/site-
packages/numpy/core/fromnumeric.py:3372: RuntimeWarning: Mean of empty slice.
  return _methods._mean(a, axis=axis, dtype=dtype,
/Users/dtradke/.pyenv/versions/3.8.6/lib/python3.8/site-
packages/numpy/core/_methods.py:170: RuntimeWarning: invalid value encountered
in double_scalars
  ret = ret.dtype.type(ret / rcount)
```

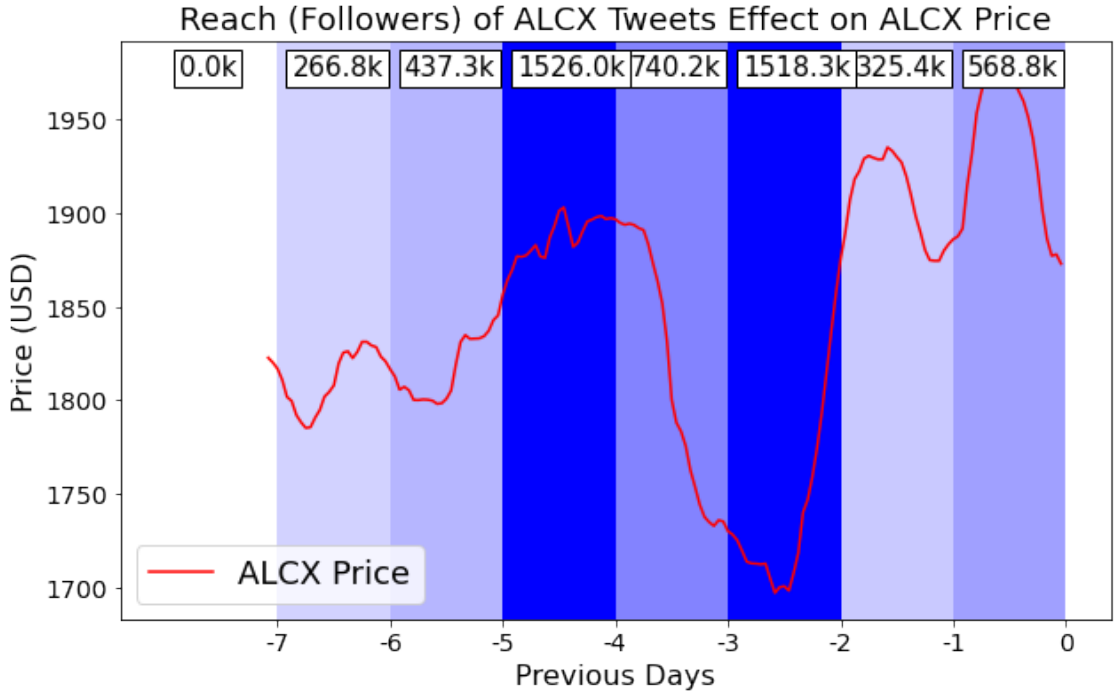


0.1.5 Experiment 4: Tweet Reach Effect on Price

In this last experiment, we analyze the number of followers each account tweeting about ALCX has, also known as the "reach" of each tweet. This represents the visibility of each tweet, giving insight into how many people are exposed to the information about ALCX.

Results Similar to the previous graphs, the x and y axes correspond to time and the price of ALCX. The shading and numbers across the top of the figure correspond to the total reach of tweets per-day, rounded to the nearest hundred. Interestingly enough, the figure below shows that the two days with almost the same and most reach (1526k and 1518.3k) see large inclines in the price of ALCX. This result supports the hypothesis that highly influential Twitter accounds could have a large impact on the price of ALCX. Interestingly enough, the largest increase in price comes when low average sentiment tweets are broadcasted to a large audience. Furthermore, the three days with the highest reach correspond with a 13% change in price. In conclusion, there is serious reason to believe that Twitter reach has a direct impact on the price of ALCX variability and overall price.

[14]:
```
# How the reach (followers) of tweets affected the price
plotTweetAmts(day_dict, tweet_reach, username='All "ALCX"', reach=True)
```



0.1.6 Conclusion

This notebook offers an in-depth analysis of how Twitter profiles, tweet volume, sentiment, and reach affect the price of the ALCX cryptocurrency using Twitter and Flipside Crypto datasets. I have used direct cross-correlations to analyze the affect of @AlchemixFi Twitter account and tweet volume. Furthermore, while I have used various AI methods for sentiment analysis of tweets at

day resolution, future work could include more advanced methods of intelligence to have further confidence in sentiment classification with higher-resolution scales. The highest influence on the price of ALCX I found had to do with the reach, or total followers of tweets containing "ALCX". I found that the price had the highest fluctuation on days where the reach of tweets was the most. Tese results suggest that the Twitter social media platform may have serious impacts on the price and perception of ALCX.

[ ]:

[ ]: