CS579 Project 1: Social Media Data Analysis

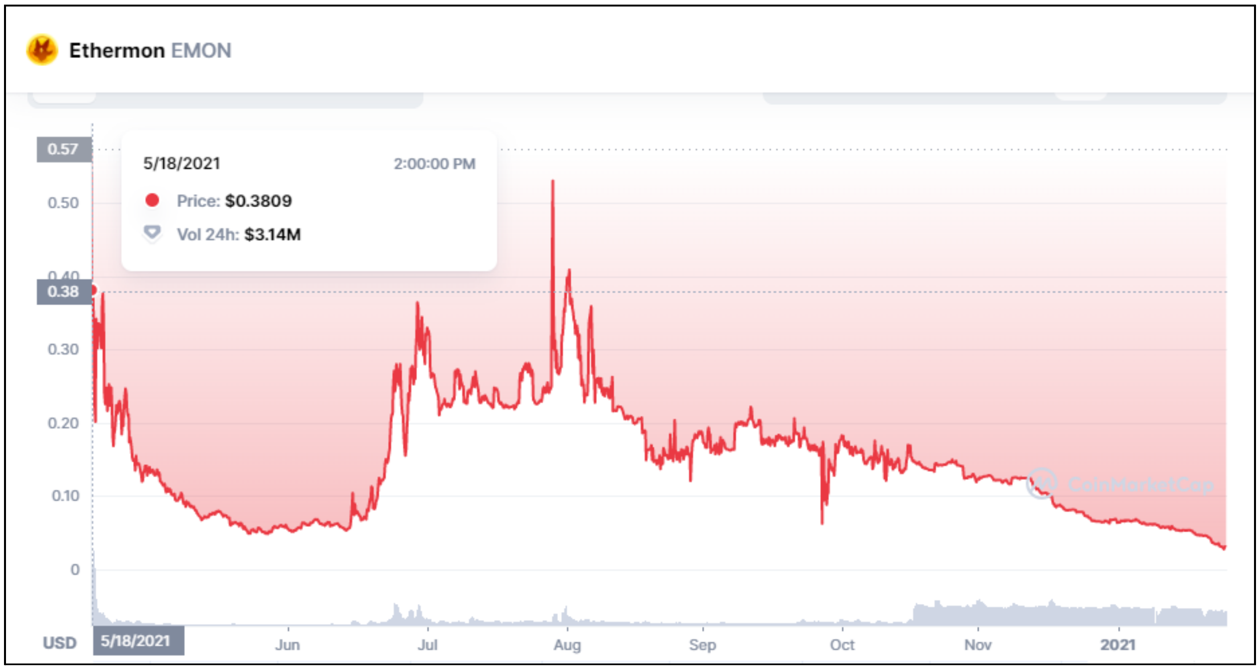
Priyam Dinesh Shah, William E Bodell III
Group - 20

I. Project Details

Title: Verifying the price hype for NFTs over the social network to understand the cause

Short Description
We believe that many NFTs are blindly hyped by creators and marketers to make money. There are a number of major influencers on Twitter who ill-market the NFT based digital arts to their followers, often being paid to do so. We're planning to correlate the price of a NFT use case ”ëthermon” and its tweets from the users to check the hype around it over a period of time during its launch and steep fall. During this analysis we are trying to uncover any trends which might suggest why this NFT projects have experienced organic growth and how prices got influenced by high-profile shilling.

Price Chart of Ethermon (https://coinmarketcap.com/currencies/ethermon/)

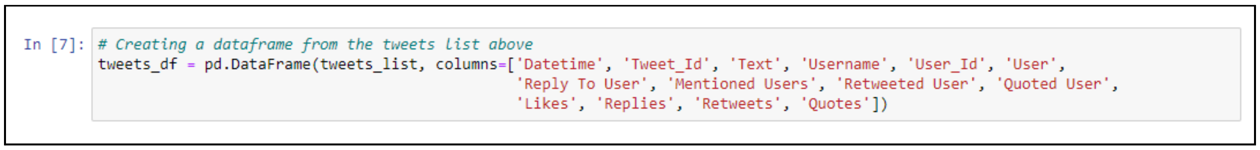


Based on the price chart we considered our analysis time period in two phases
1. Ethermon Project Launch Phase (May'21)
2. Massive Price decline phase (Jun'21)

1. Data Collection and Processing

We selected Twitter as a social media platform for our project. Initially we tried collecting data using the Twitter API, but it was not helpful for historical data collection as we required data of previous year. We later also tried applying for the academic research api access but had no luck in getting credentials for it. So finally, we used the snscrape library of python for scraping the historical data for specific time periods and a given query.

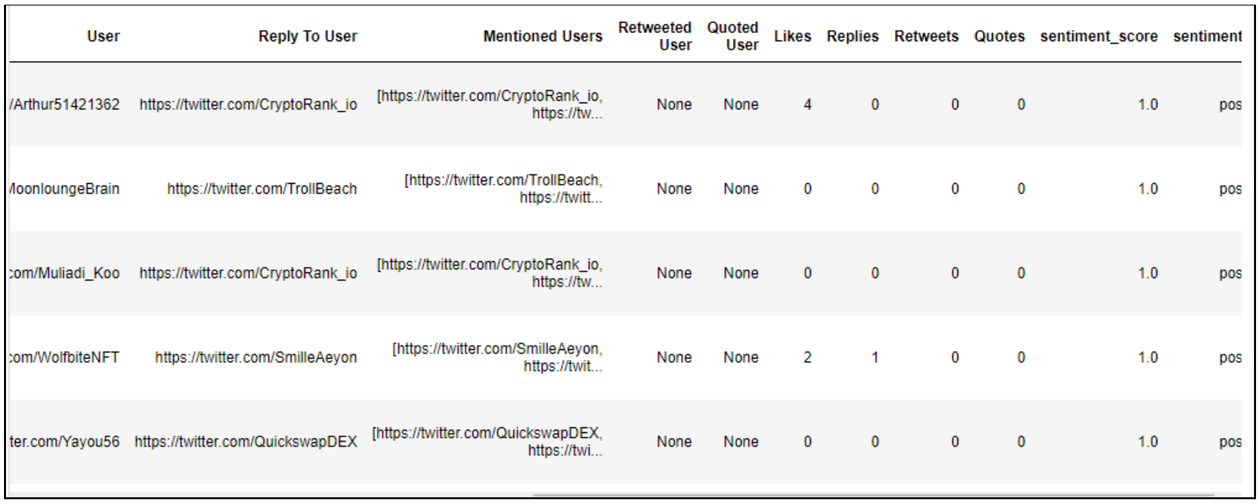
We scraped the following data using snscrape and stored them in a dataframe for both the time periods. The data collected is stored in dataframe and is used on the fly for all below calculations.



```
In [7]: # Creating a dataframe from the tweets list above
        tweets_df = pd.DataFrame(tweets_list, columns=['Datetime', 'Tweet_Id', 'Text', 'Username', 'User_Id', 'User',
                                                       'Reply To User', 'Mentioned Users', 'Retweeted User', 'Quoted User',
                                                       'Likes', 'Replies', 'Retweets', 'Quotes'])
```

Dataframe structure for tweet data we collected.

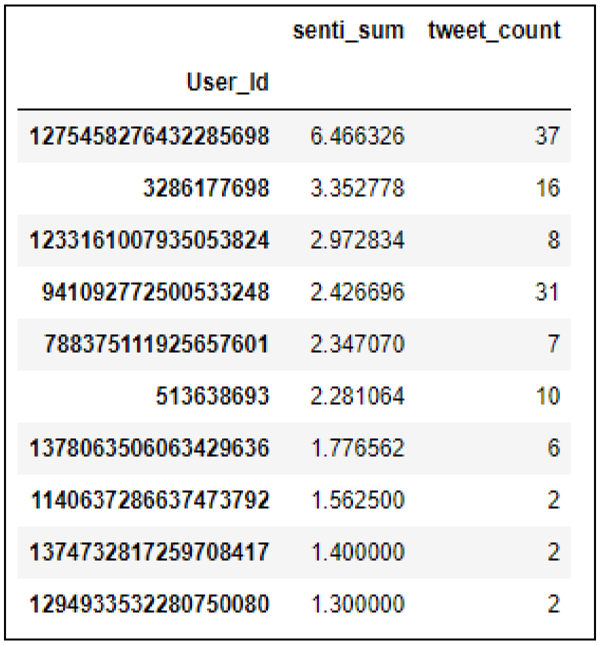
Since we required user reaction (red for positive reaction and blue for negative reaction in below results) for our graph to decide whether the sentiment of tweets for that time period is positive or negative. We used sentiment analysis on the stored data using textblob library and added an additional column called sentiment_score.



User	Reply To User	Mentioned Users	Retweeted User	Quoted User	Likes	Replies	Retweets	Quotes	sentiment_score	sentiment
/Arthur51421362	https://twitter.com/CryptoRank_io	[https://twitter.com/CryptoRank_io, https://tw...	None	None	4	0	0	0	1.0	pos
/loonloungeBrain	https://twitter.com/TrollBeach	[https://twitter.com/TrollBeach, https://twitt...	None	None	0	0	0	0	1.0	pos
:om/Muliadi_Koo	https://twitter.com/CryptoRank_io	[https://twitter.com/CryptoRank_io, https://tw...	None	None	0	0	0	0	1.0	pos
:om/WolfbiteNFT	https://twitter.com/SmilleAeyon	[https://twitter.com/SmilleAeyon, https://twit...	None	None	2	1	0	0	1.0	pos
ter.com/Yayou56	https://twitter.com/QuickswapDEX	[https://twitter.com/QuickswapDEX, https://twi...	None	None	0	0	0	0	1.0	pos

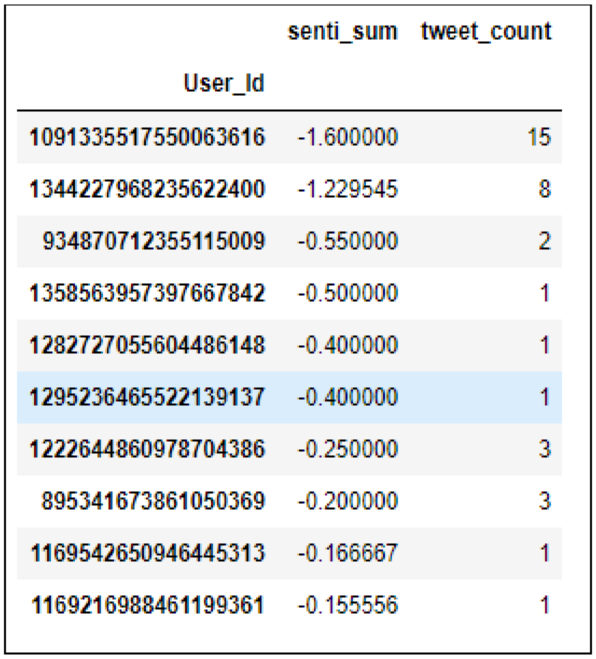
Updated results of dataframe after sentiment analysis

We further filtered the data collected as it had similar users, so we decided to aggregate them and their sentiment scores. Finally we filtered our data to provide relevancy by adding a condition of keeping higher tweet count and sentimental score values only.



User_Id	senti_sum	tweet_count
1275458276432285698	6.466326	37
3286177698	3.352778	16
1233161007935053824	2.972834	8
941092772500533248	2.426696	31
788375111925657601	2.347070	7
513638693	2.281064	10
1378063506063429636	1.776562	6
1140637286637473792	1.562500	2
1374732817259708417	1.400000	2
1294933532280750080	1.300000	2

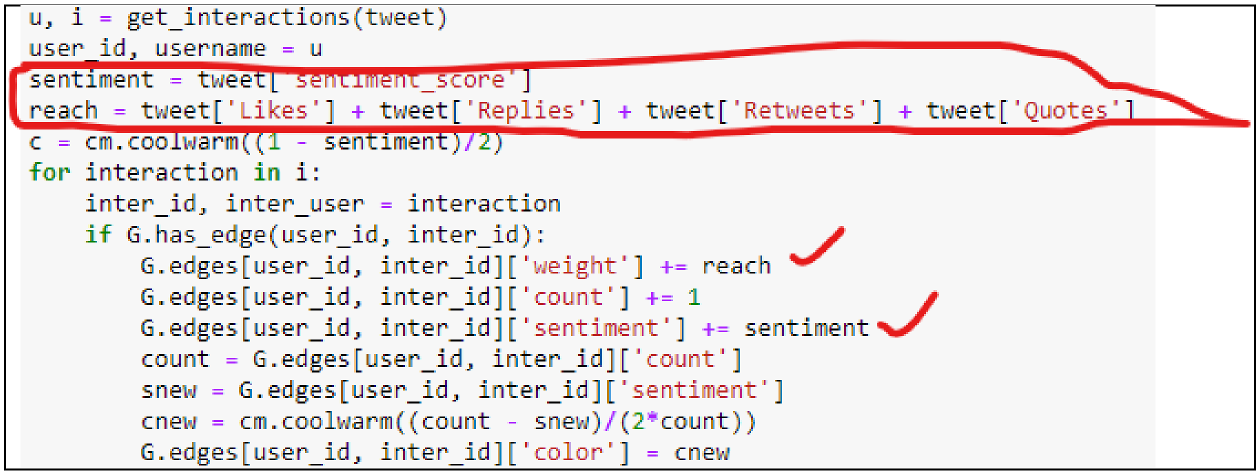
Higher Positive Sentiment Values



User_Id	senti_sum	tweet_count
1091335517550063616	-1.600000	15
1344227968235622400	-1.229545	8
934870712355115009	-0.550000	2
1358563957397667842	-0.500000	1
1282727055604486148	-0.400000	1
1295236465522139137	-0.400000	1
1222644860978704386	-0.250000	3
895341673861050369	-0.200000	3
1169542650946445313	-0.166667	1
1169216988461199361	-0.155556	1

Higher Negative Sentiment Values

For the weights on the edges of the graph we used the summation of likes count, replies count, retweets count, and quotes count. We also colored the edges according to sentiment, red for positive and blue for negative.



```
u, i = get_interactions(tweet)
user_id, username = u
sentiment = tweet['sentiment_score']
reach = tweet['Likes'] + tweet['Replies'] + tweet['Retweets'] + tweet['Quotes']
c = cm.coolwarm((1 - sentiment)/2)
for interaction in i:
    inter_id, inter_user = interaction
    if G.has_edge(user_id, inter_id):
        G.edges[user_id, inter_id]['weight'] += reach
        G.edges[user_id, inter_id]['count'] += 1
        G.edges[user_id, inter_id]['sentiment'] += sentiment
        count = G.edges[user_id, inter_id]['count']
        snew = G.edges[user_id, inter_id]['sentiment']
        cnew = cm.coolwarm((count - snew)/(2*count))
        G.edges[user_id, inter_id]['color'] = cnew
```

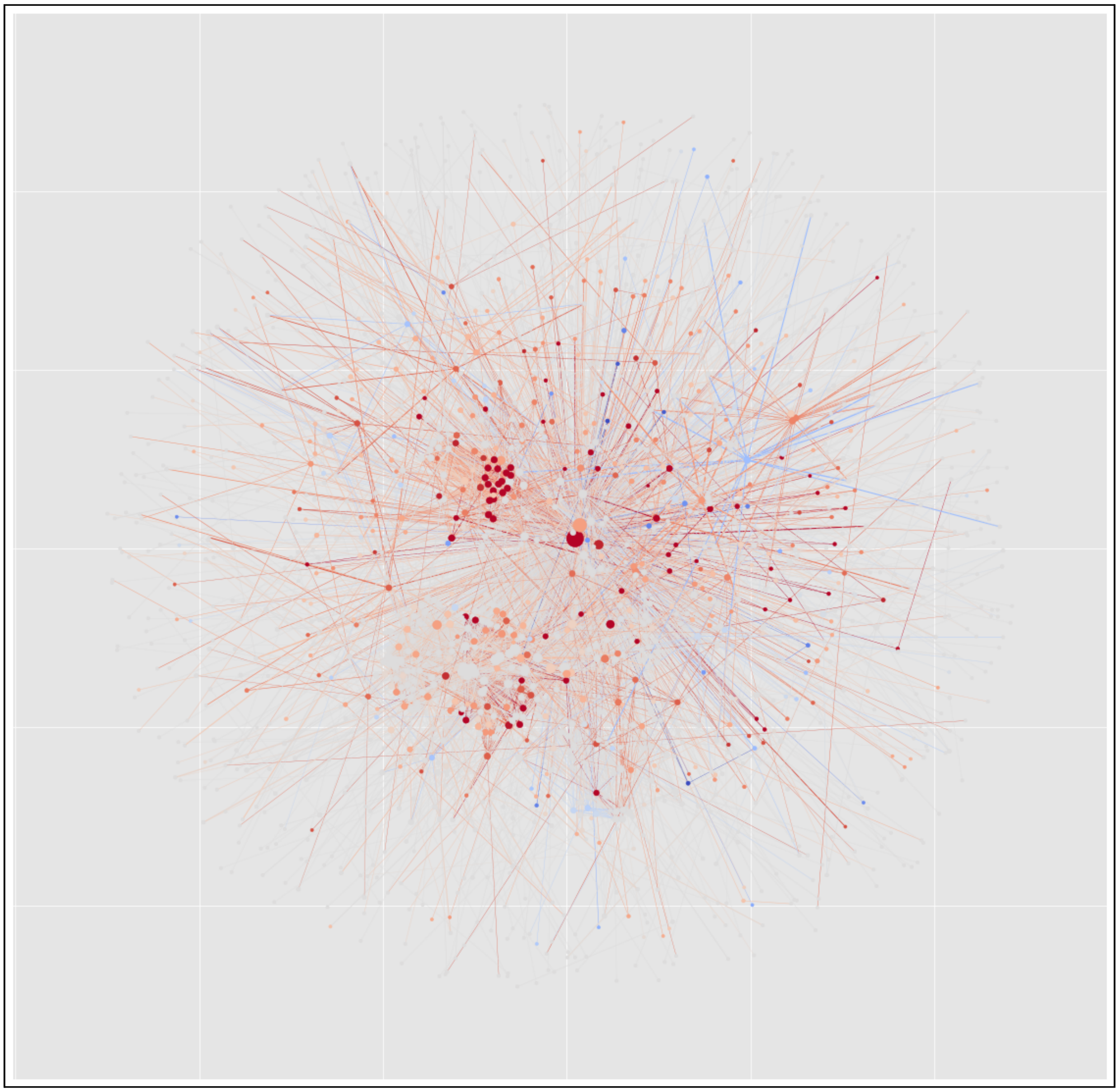
Code Snippet for weights and edge color for interaction graph

2. Data Visualization

We have used the NetworkX tool for our graph visualization. We have plotted interaction graphs for two different time periods as mentioned above.

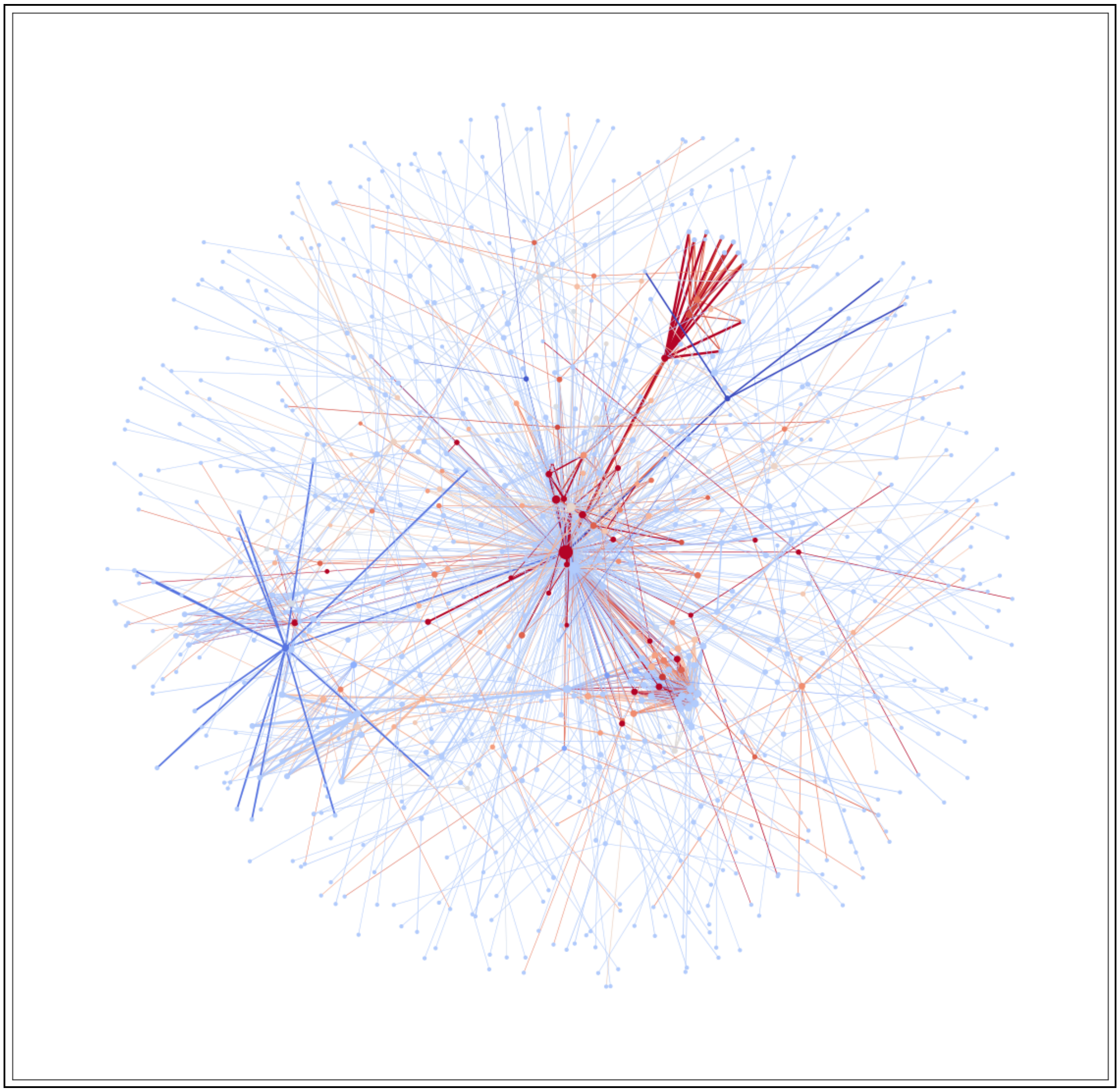
The nodes are users, and edges interactions between them. Node size represents the number of tweets by the user, and its color the overall sentiment score, with red being positive and blue *being negative. Edges are colored according to sentiment as well, and weighted according to the number of likes, replies, retweets and quotes the tweets received. If a user mentions another user in multiple tweets, we group the interactions into one edge.*

Below are the graphs and their understandings.



Graph for May'21 phase when the price went from 0.38$ -> 0.139$ drop

As we can see from the graph for May'21 during the launch phase and till slight decrement in price had very high positive reaction for ethermon project among the users. The big red nodes which shows the influencers, creators and marketers promoting heavily to raise the price values.

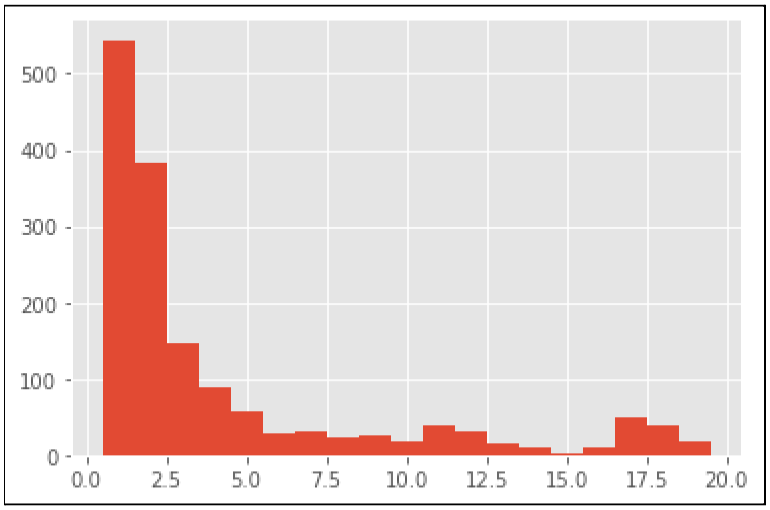


Graph for Jun'21 phase when the price was completely down from 0.38$ -> 0.05$ drop

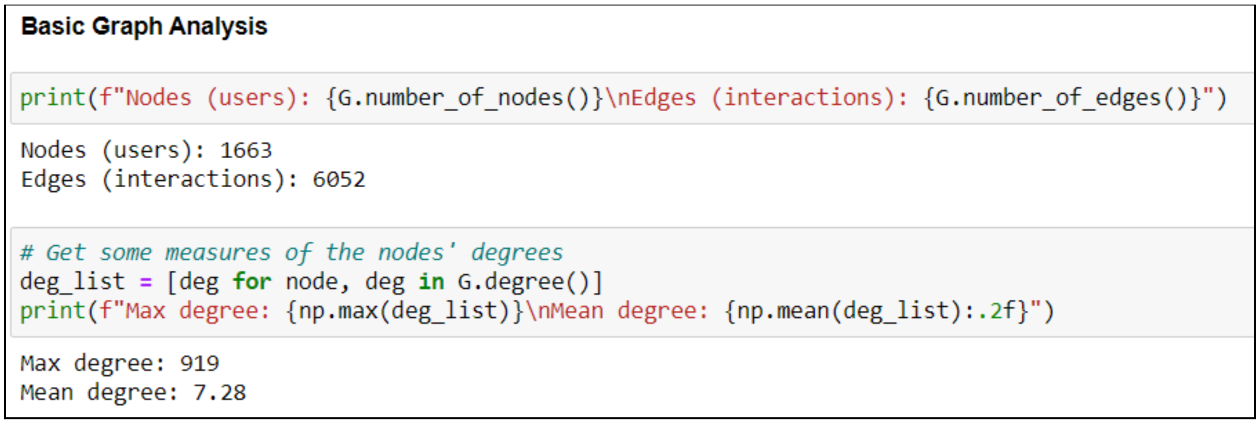
As we can see during this time period the complete tweets were of negative sentiments, with few positive ones trying to still push the price up to recover their loss. But since the creators, influencers and money makers have already made their profits till mid-May now the project price went way down. Soon, after realizing the motives of the project, people started heavily commenting harsh about it.

3. Network Measures Calculation

For the graph of May'21 the network measures are as followed



Degree Distribution as a histogram of Graph for May'21



Basic Graph Analysis

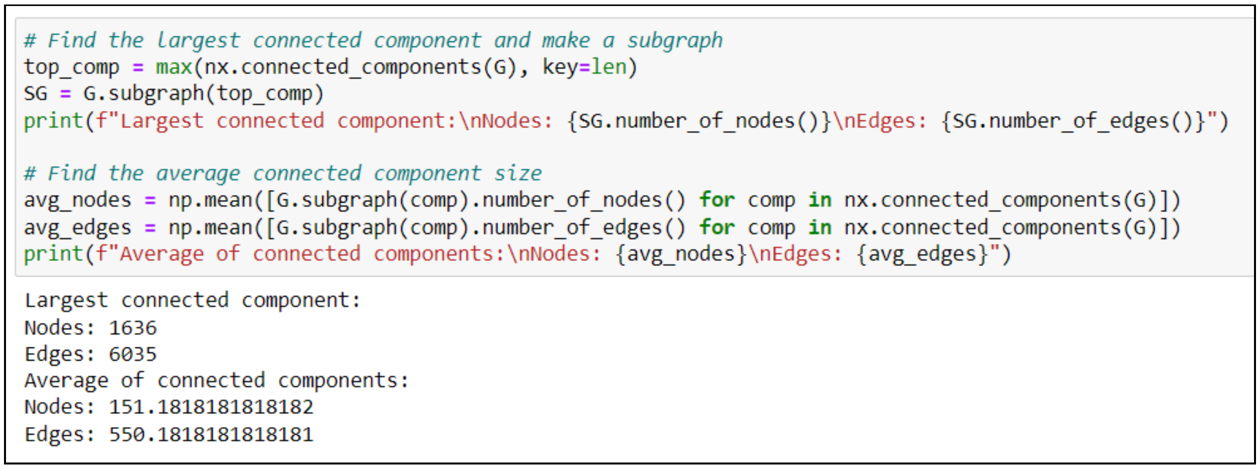
```
print(f"Nodes (users): {G.number_of_nodes()}\nEdges (interactions): {G.number_of_edges()}")
```

```
Nodes (users): 1663
Edges (interactions): 6052
```

```
# Get some measures of the nodes' degrees
deg_list = [deg for node, deg in G.degree()]
print(f"Max degree: {np.max(deg_list)}\nMean degree: {np.mean(deg_list):.2f}")
```

```
Max degree: 919
Mean degree: 7.28
```

Node and Edge Count, Max and Mean Degree of Graph for May'21

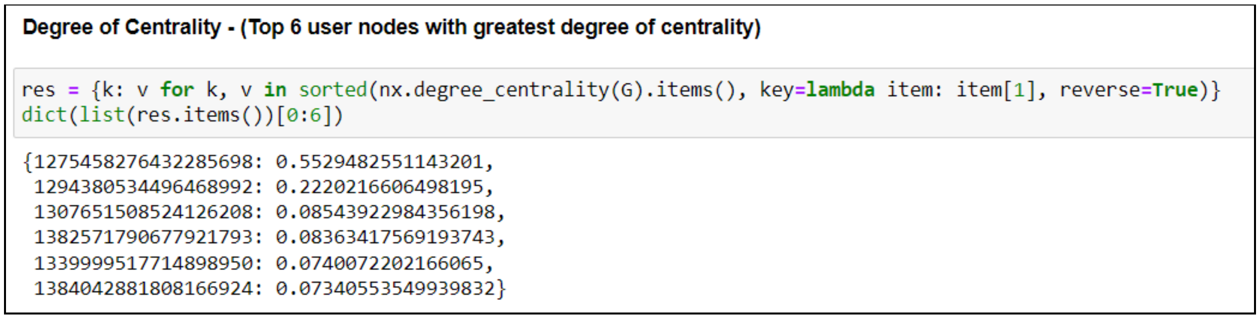


```
# Find the largest connected component and make a subgraph
top_comp = max(nx.connected_components(G), key=len)
SG = G.subgraph(top_comp)
print(f"Largest connected component:\nNodes: {SG.number_of_nodes()}\nEdges: {SG.number_of_edges()}")

# Find the average connected component size
avg_nodes = np.mean([G.subgraph(comp).number_of_nodes() for comp in nx.connected_components(G)])
avg_edges = np.mean([G.subgraph(comp).number_of_edges() for comp in nx.connected_components(G)])
print(f"Average of connected components:\nNodes: {avg_nodes}\nEdges: {avg_edges}")
```

```
Largest connected component:
Nodes: 1636
Edges: 6035
Average of connected components:
Nodes: 151.1818181818182
Edges: 550.1818181818181
```

Connected Component Measure of Graph for May'21

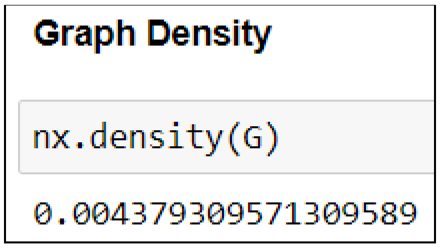


Degree of Centrality - (Top 6 user nodes with greatest degree of centrality)

```
res = {k: v for k, v in sorted(nx.degree_centrality(G).items(), key=lambda item: item[1], reverse=True)}
dict(list(res.items())[0:6])
```

```
{1275458276432285698: 0.5529482551143201,
 1294380534496468992: 0.2220216606498195,
 1307651508524126208: 0.08543922984356198,
 1382571790677921793: 0.08363417569193743,
 1339999517714898950: 0.0740072202166065,
 1384042881808166924: 0.07340553549939832}
```

Degree of Centrality for Top 6 user nodes with greatest values for May'21 time period



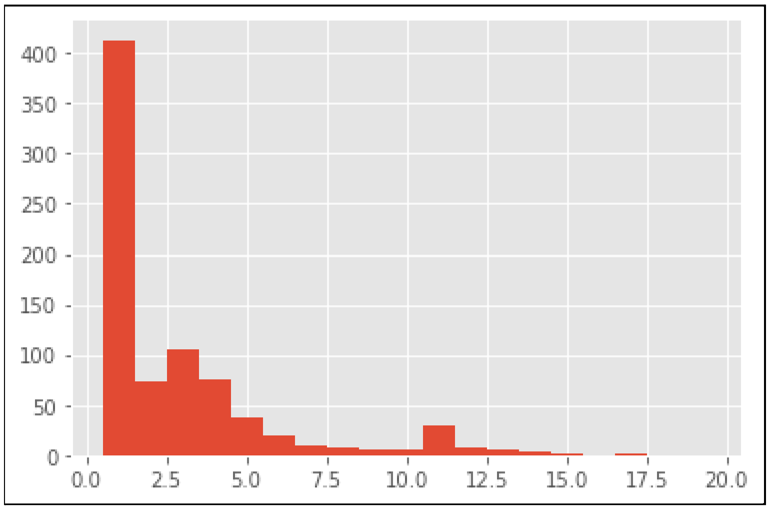
Graph Density

```
nx.density(G)
```

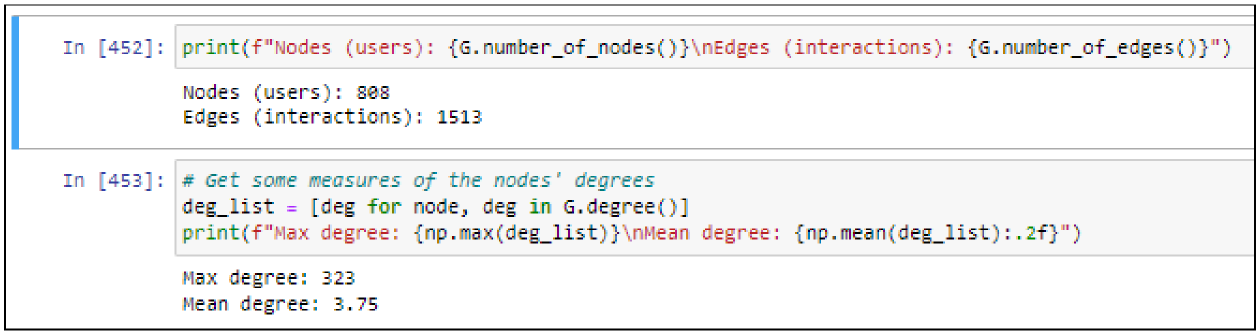
```
0.004379309571309589
```

Graph Density for May'21 time period

For the graph of June'21 the network measures are as followed



Degree Distribution as a histogram of Graph for Jun'21



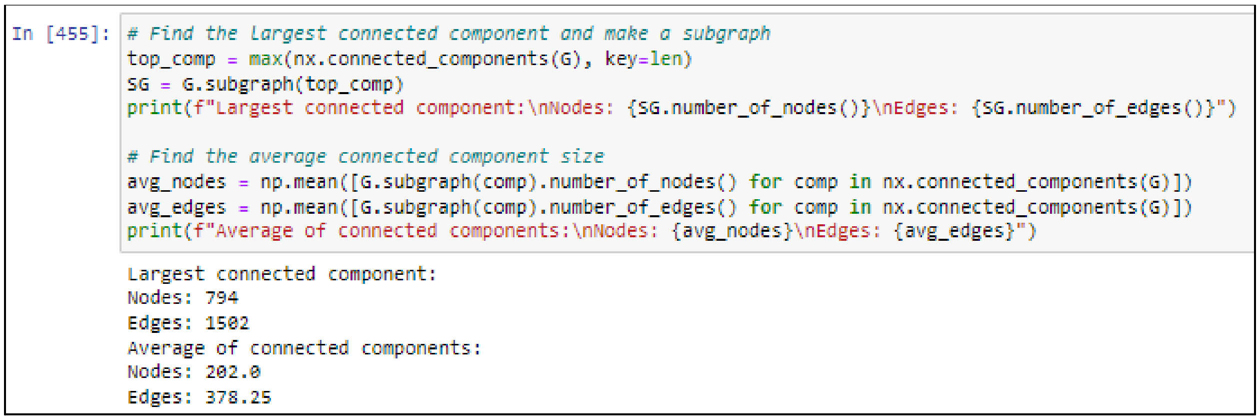
```
In [452]: print(f"Nodes (users): {G.number_of_nodes()}\nEdges (interactions): {G.number_of_edges()}")

          Nodes (users): 808
          Edges (interactions): 1513

In [453]: # Get some measures of the nodes' degrees
          deg_list = [deg for node, deg in G.degree()]
          print(f"Max degree: {np.max(deg_list)}\nMean degree: {np.mean(deg_list):.2f}")

          Max degree: 323
          Mean degree: 3.75
```

Node and Edge Count, Max and Mean Degree of Graph for Jun'21

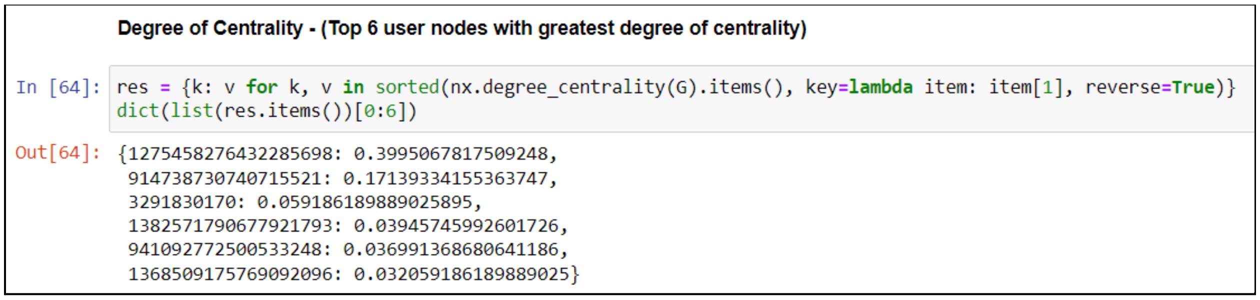


```
In [455]: # Find the Largest connected component and make a subgraph
          top_comp = max(nx.connected_components(G), key=len)
          SG = G.subgraph(top_comp)
          print(f"Largest connected component:\nNodes: {SG.number_of_nodes()}\nEdges: {SG.number_of_edges()}")

          # Find the average connected component size
          avg_nodes = np.mean([G.subgraph(comp).number_of_nodes() for comp in nx.connected_components(G)])
          avg_edges = np.mean([G.subgraph(comp).number_of_edges() for comp in nx.connected_components(G)])
          print(f"Average of connected components:\nNodes: {avg_nodes}\nEdges: {avg_edges}")

          Largest connected component:
          Nodes: 794
          Edges: 1502
          Average of connected components:
          Nodes: 202.0
          Edges: 378.25
```

Connected Component Measure of Graph for Jun'21

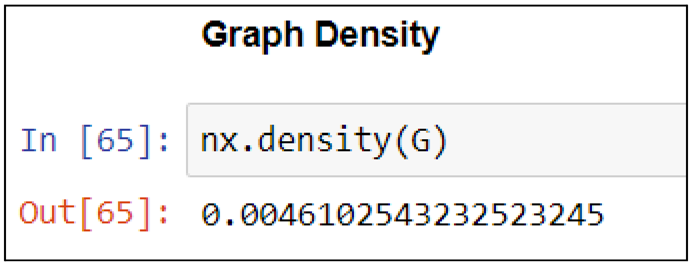


Degree of Centrality - (Top 6 user nodes with greatest degree of centrality)

```
In [64]: res = {k: v for k, v in sorted(nx.degree_centrality(G).items(), key=lambda item: item[1], reverse=True)}
         dict(list(res.items())[0:6])

Out[64]: {1275458276432285698: 0.3995067817509248,
          914738730740715521: 0.17139334155363747,
          3291830170: 0.059186189889025895,
          1382571790677921793: 0.03945745992601726,
          941092772500533248: 0.036991368680641186,
          1368509175769092096: 0.032059186189889025}
```

Degree of Centrality for Top 6 user nodes with greatest values for Jun'21 time period



Graph Density

```
In [65]: nx.density(G)

Out[65]: 0.0046102543232523245
```

Graph Density for Jun'21 time period

4. References

1. Python libraries used snscrape, np, re, textblob, pandas, matplotlib, networx
2. Tutorials referred
    a. https://github.com/ugis22/analysing_twitter/blob/master/Jupyter%20Notebook%20files/Interaction%20Network.ipynb
3. NFT use case link https://coinmarketcap.com/currencies/ethermon/
4. Master thread of similar nft use case https://twitter.com/zachxbt/status/1465464817813176324?t=8aEHr1wgWhf65K91e7DWrw&s=19