Readme File

Initially, follow the below sequence for executing the code

- Upload all the dataset files on google colab.
- Change the directory to the drive folder where all the files are present as displayed below:
- Follow the cell sequence for execution.

```
os.chdir('/content/drive/MyDrive/Assignment-3-IR/')
```

Dataset: The dataset used in this assignment is '`soc-sign-bitcoinotc.csv`'.

Question 1:
This question is in the code file **'Assignment_3_Q1.ipynb'**

Pre-processing:

- Import libraries then Read the dataset as shown

```
import pandas as pd
import os
import matplotlib.pyplot as plt
import math
import numpy as np
import collections
from scipy.sparse import coo_matrix
from matplotlib import style
```

```
os.chdir('/content/drive/MyDrive/Assignment-3-IR/')
```

```
# !tar -xvf "twitter.tar.gz"
```

```
colnames=['source', 'target', 'rating', 'time']
df = pd.read_csv("soc-sign-bitcoinotc.csv",names =colnames,header=None)
df.head
```

Assumptions: No assumptions

Methodology:

- Firstly, we created a weighted adjacency matrix and and unweighted adjacency matrix from the dataset

```
n_nodes = len(node_list)
A = np.zeros((n_nodes, n_nodes))
for row in df.itertuples():
    tt.at[row.source,row.target] = row.rating

print("Weighted directed Adjacency Matrix:" + "\n")
tt
```

```
adj_mat = pd.crosstab(df.source, df.target)
idx = adj_mat.columns.union(adj_mat.index)
adj_mat = adj_mat.reindex(index = idx, columns=idx, fill_value=0)

print("\n")
print("Undirected Adjacency Matrix:" + "\n")

adj_mat
```

- Then, number of nodes, number of edges, edge list were calculated

```
print("Number of Nodes: ",len(np.unique(node_list)))
```

```
Number of Nodes:  5881
```

```
num_edges = np.count_nonzero(tt)
print("Number of Edges: ",num_edges)
```

```
Number of Edges:  35592
```

```
edges_list2 = []
for edge in edge_list.itertuples():
  edges_list2.append([edge.source,edge.target])
print("Edge List: ",str(edges_list2))
```

```
Edge List:  [[6, 2], [6, 5], [1, 15], [4, 3], [13, 16], [13, 10], [7, 5], [2, 21], [2, 20], [21, 2], [
```

- Average In-degree and average Out-degree were calculated using the below code

```python
avg_in_deg = np.sum(np.count_nonzero(tt, axis=1))/len(node_list)
print("Average In-Degree of Network: ",avg_in_deg)
```

```
Average In-Degree of Network:  6.052031967352491
```

```python
avg_out_deg = np.sum(np.count_nonzero(tt, axis=0))/len(node_list)
print("Average Out-Degree of Network: ",avg_out_deg)
```

```
Average Out-Degree of Network:  6.052031967352491
```

- Maximum in-degree, out-degree as well as density of the network were found out using the following logic:

```python
dict_in_deg={}
dict_out_deg = {}
for i in tt.index:
  dict_in_deg[i] = sum(adj_mat.loc[i])
  dict_out_deg[i] = sum(adj_mat[i])
in_deg_max = max(dict_in_deg.values())
for key,val in dict_in_deg.items():
  if val == in_deg_max:
    print("Node ID: "+str(key)+"  |  Maximum In-Degree: "+str(val))
```

```
Node ID: 35  |  Maximum In-Degree: 763
```

```python
out_deg_max = max(dict_out_deg.values())
for key,val in dict_out_deg.items():
  if val == out_deg_max:
    print("Node ID: "+str(key)+"  |  Maximum Out-Degree: "+str(val))
```

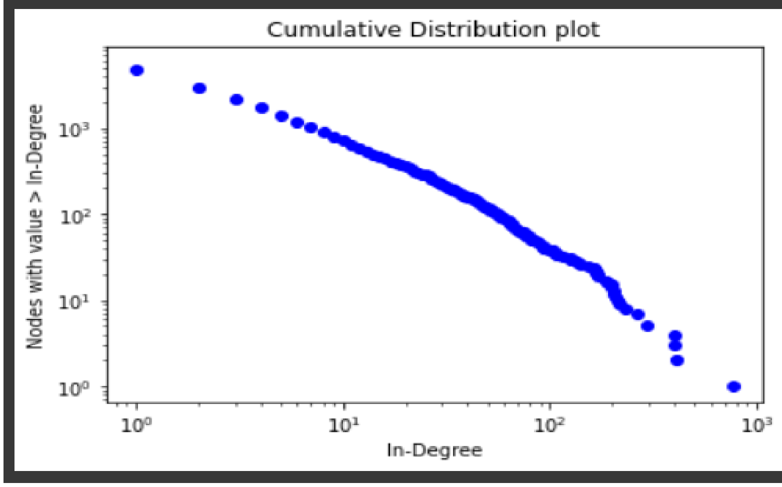
```
Node ID: 35  |  Maximum Out-Degree: 535
```

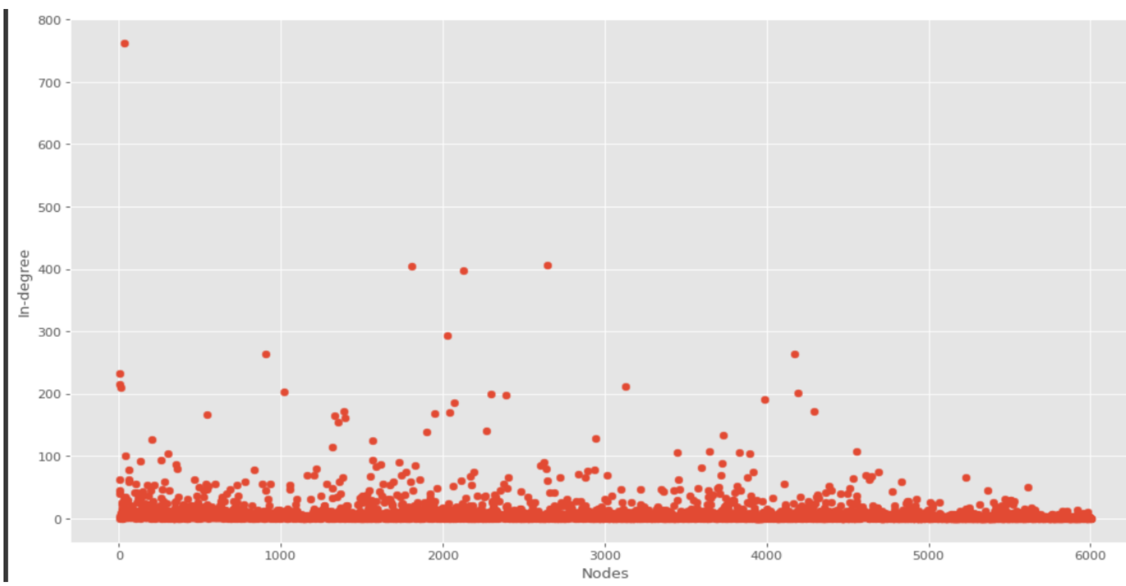
```python
print("Density of Network", num_edges/(len(node_list)*(len(node_list)-1)))
```

```
Density of Network 0.0010292571373048454
```

- In-degree distribution of the network came to be

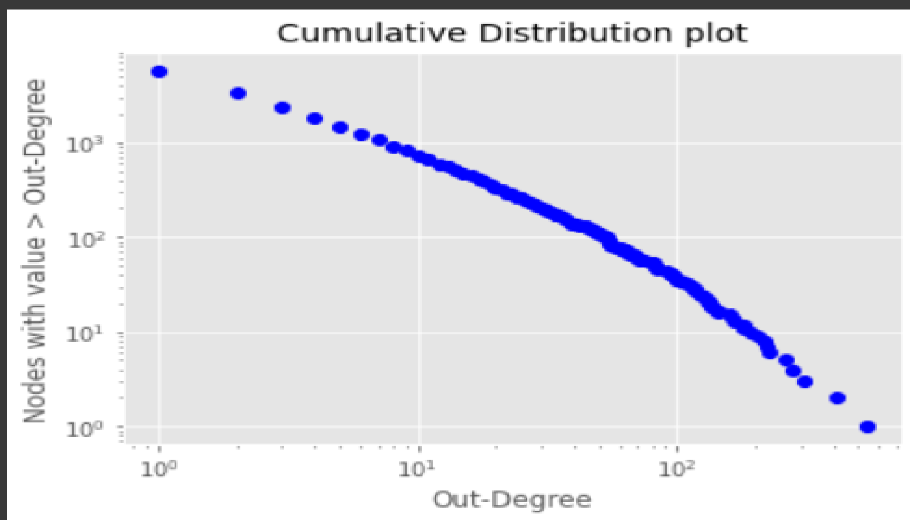
```
degree_sequence = sorted(in_degree_count, reverse=True)
degreeCount = collections.Counter(degree_sequence)
deg, cnt = zip(*degreeCount.items())
cs = np.cumsum(cnt)
plt.loglog(deg, cs, 'bo')
plt.title("Cumulative Distribution plot")
plt.ylabel("Nodes with value > In-Degree")
plt.xlabel("In-Degree")
plt.show()
```

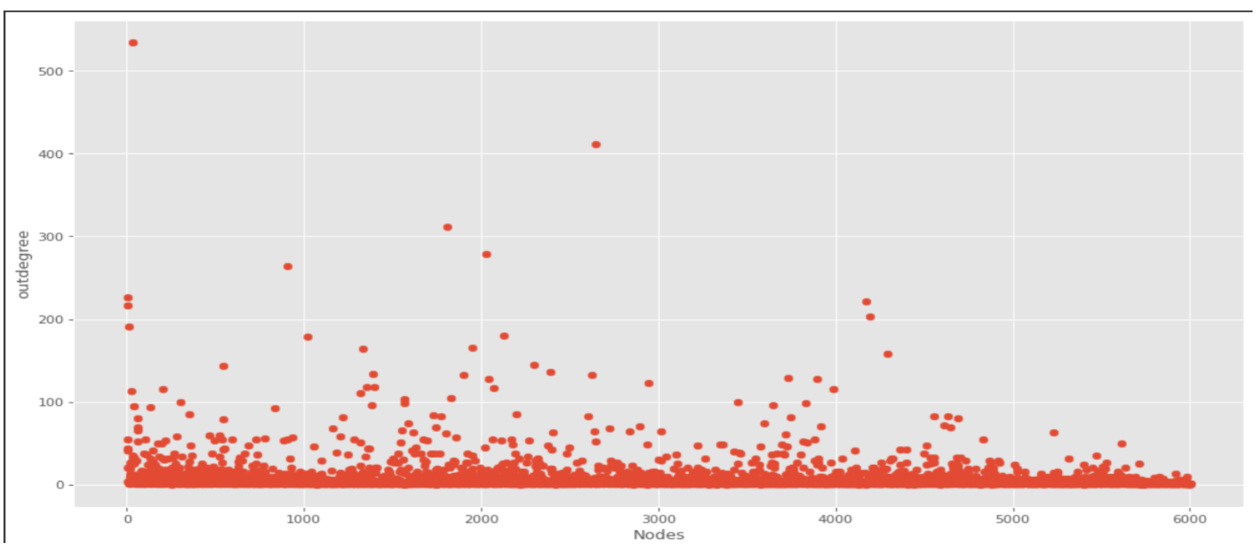




- Out-degree distribution of the network was evaluated as

```
degree_sequence_out = sorted(out_degree_count, reverse=True)
degreeCountOut = collections.Counter(degree_sequence_out)
deg_out, cnt_out = zip(*degreeCountOut.items())
cs_out = np.cumsum(cnt_out)
plt.loglog(deg_out, cs_out, 'bo')
plt.title("Cumulative Distribution plot")
plt.ylabel("Nodes with value > Out-Degree")
plt.xlabel("Out-Degree")
plt.show()
```

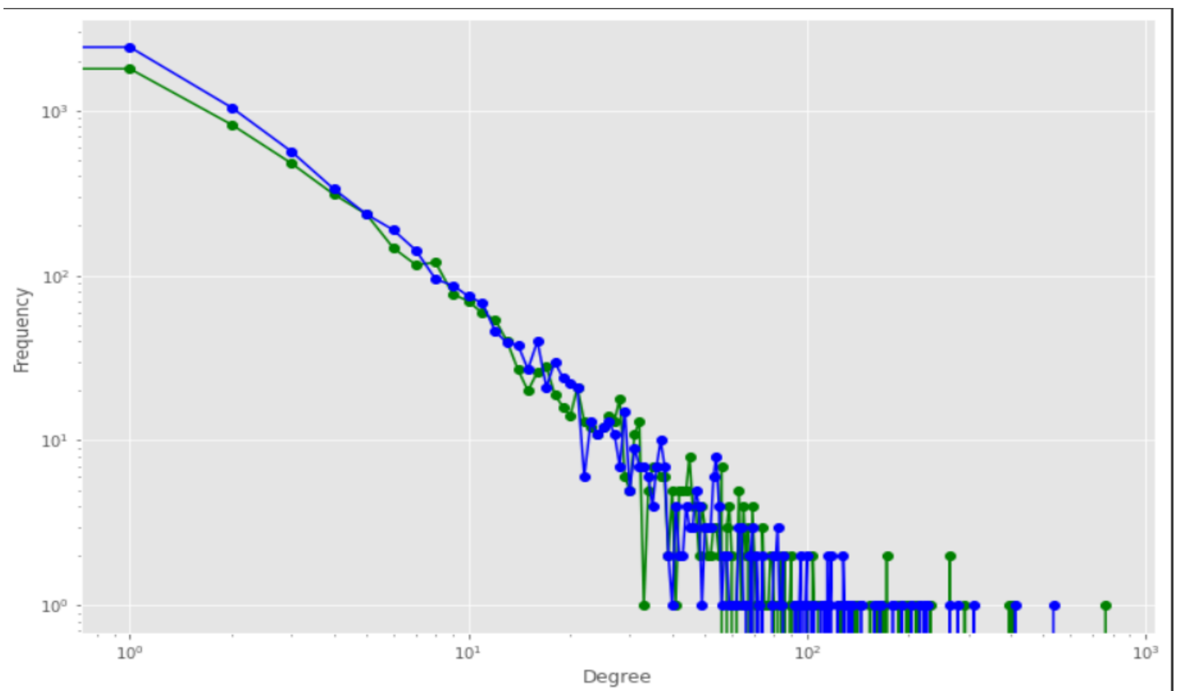




- The frequency was calculated for the each of the appearing in-degree and out-degree in the network as

```
in_max=max(in_degree_count)+1
in_degree_freq= [ 0 for deg in range(in_max) ]
for deg in in_degree_count:
    in_degree_freq[deg] += 1
```

```
out_max=max(out_degree_count)+1
out_degree_freq= [ 0 for d in range(out_max) ]
for deg in out_degree_count:
    out_degree_freq[deg] += 1
```



- Local Clustering coefficient was calculated for each of the node using the neighbour count and the formula

  ```
  clustering_coefficient=n_links/(0.5*n_neighbors*(n_neighbors-1))
  ```

  Where n_links denotes the count of neighours of the node that are also connected to each other.

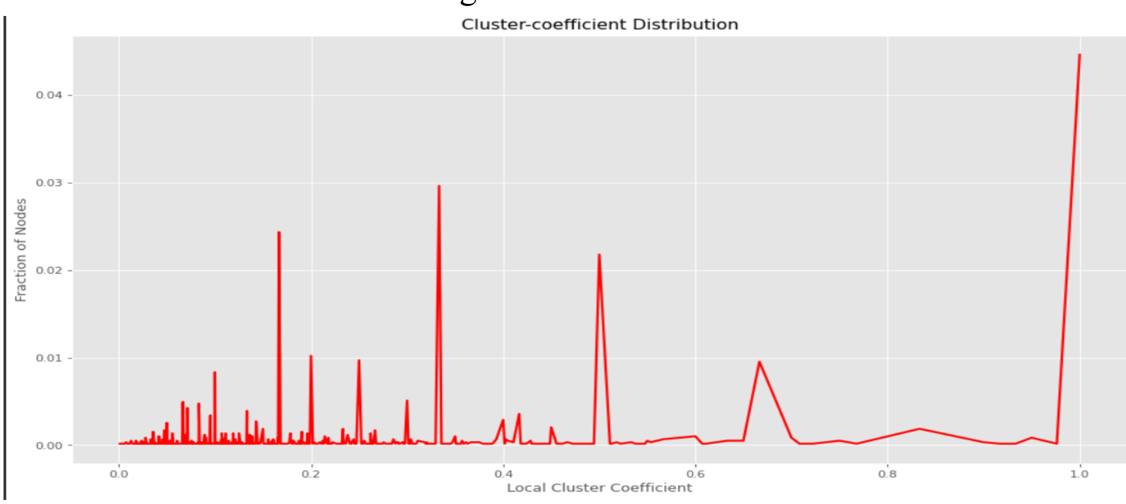
```
temp_clust_dict = {}
for node in node_list:
  if node in temp_list:
      neighbours = res_neighbor_list.loc[node].index.tolist()
      n_neighbors=len(neighbours)
      n_links=0
      if n_neighbors>1:
          for node1 in neighbours:
              for node2 in neighbours:
                  if adj_mat.at[node1,node2] > 0:
                      n_links+=1
          n_links/=2
          clustering_coefficient=n_links/(0.5*n_neighbors*(n_neighbors-1))
          if node not in temp_clust_dict.keys():
            temp_clust_dict[node] = clustering_coefficient
      else:
          temp_clust_dict[node] = 0
  else:
      temp_clust_dict[node] = 0
```

- Then the frequency of each of the clustering coefficients that appeared in our network was calculated as

```
frac_nodes = {}
for keys,val in temp_clust_dict.items():
  if val not in frac_nodes:
    frac_nodes[val] = 1

  else:
    frac_nodes[val] += 1
for node in frac_nodes:
    frac_nodes[node] = frac_nodes[node] / len(node_list)
```

- Distribution of the local clustering coefficients in the network is



Question 2:

The Solution to this question is in code file **Assignment3_Q2.** Here we need to find the Page rank and Hits of a node.
We have done this question using both approaches. First, we did it using the inbuilt function, available in the Network x library and then we implemented it from scratch for a single iteration.

Here are the results that we obtained from the NetworkX library and the Scratch Code

Page Rank

Node	Page Rank from Networkx	Page Rank scratch using single Iteration
6	0.0007741085917228506	0.000516795865633075
2	0.0009774710321327727	0.0002645502645502645
5	9.298616272940449e-05	0.0011574074074074073
1	0.005029048679852529	0.0004894762604013706
15	0.00032293239289585003	0.00041288191577208916
4	0.0012898358110030761	0.004545454545454545
3	0.00038277895200766233	0.0012987012987012987
13	0.004285772484391748	0.0006510416666666666
16	5.235060037934621e-05	0.002369668246445498
10	0.00013403416569389068	0.004065040650406504

Hubs and Authority

Hubs

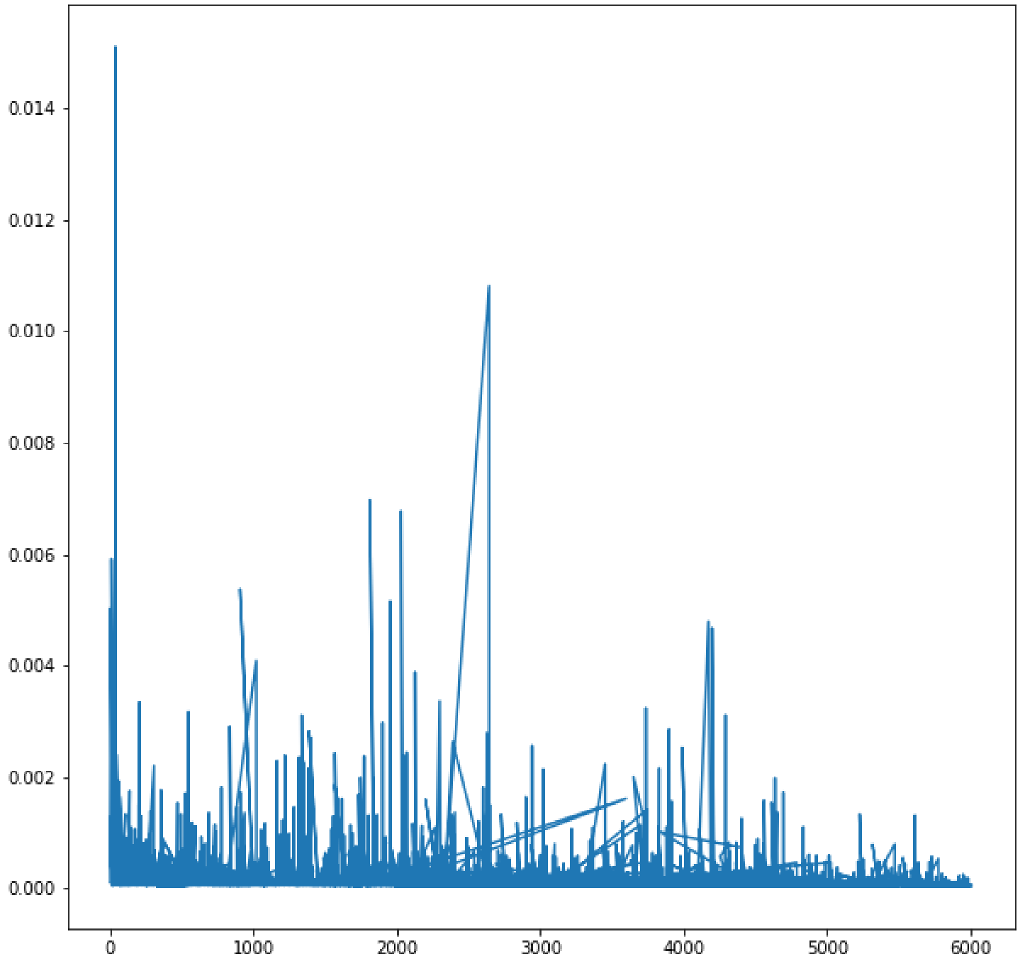
Node	Hub value from Networkx	Hub from scratch using single iteration

6	0.0014629173309573145	0.0016566245981467244
2	0.0007758275426393981	0.0008970134956435646
5	0.00020879948188316468	0.00023679986902920823
1	0.004636831266948586	0.0040787072091430085
15	0.00030249489456763605	0.0002567768127127423
4	0.0015073564036747497	0.0015469950291517205,
3	-0.0	0.0
13	0.004512775520459806	0.0036825790331743947
16	5.235060037934621e-05	0.0
10	0.0003013670388846936	0.00027724099892514296

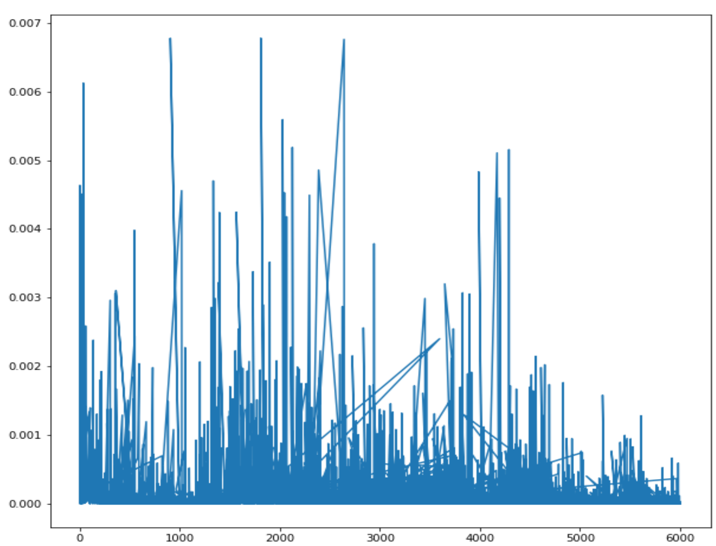
Authority

Node	Auth value from Networkx	Auth from scratch using single iteration
6	0.0015718821282270107	0.001576832330044468
2	0.0005890168397930876	0.0006052237914779686
5	0.00016970301811764727	0.000172063039375231
1	0.004496189948700376	0.003480831753438964
15	0.0002946753929369685	0.00022364661996821604
4	0.0011197026235601761	0.0009383971921573172
3	0.0005475613411447125	0.00047873802536640243
13	0.0038135198037455163	0.0028367436204183357
16	8.329469180041478e-05	7.419556112689632e-05
10	0.0002481369752860905	0.00020739425895946733

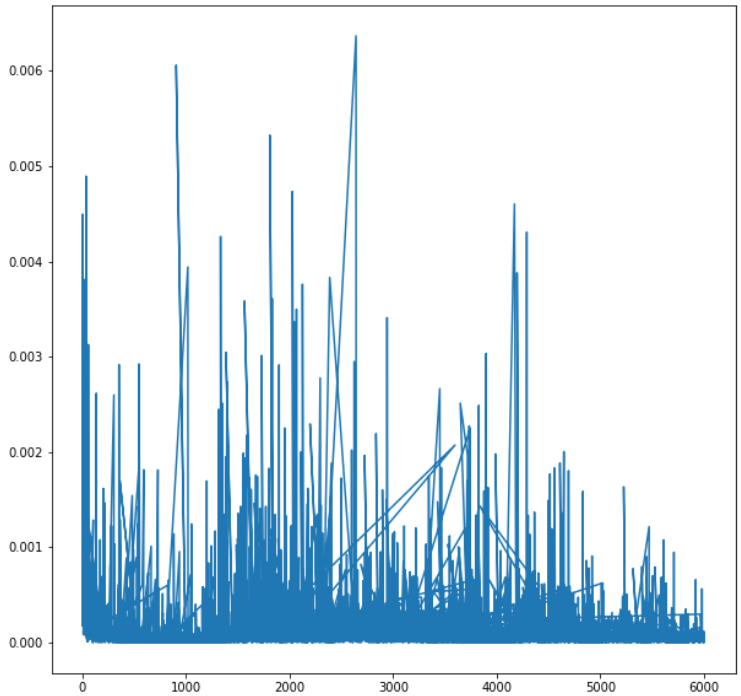
Pagerank from NetworkX graph



Hub and Auth from Network X graph

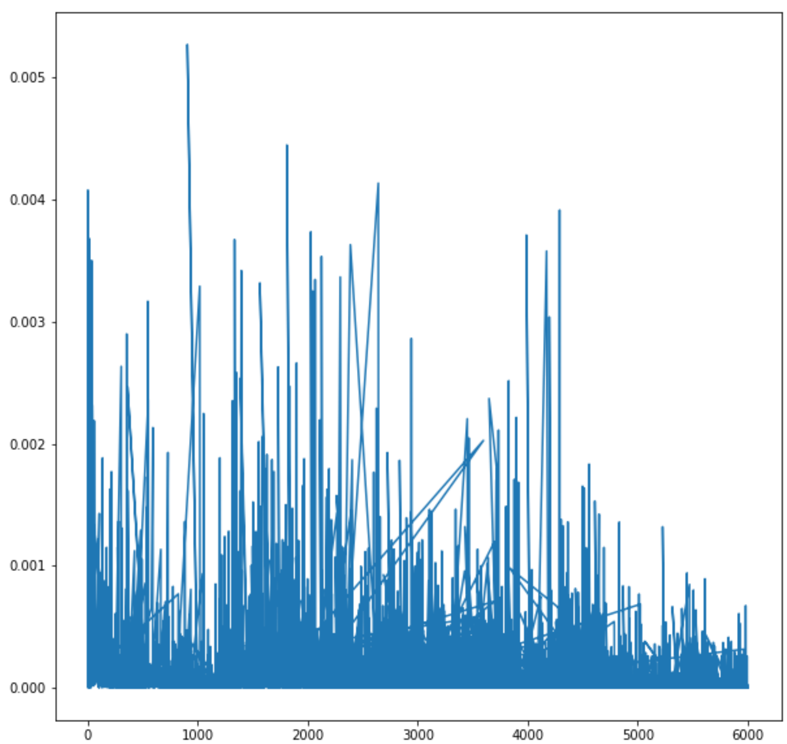


Hub Graph

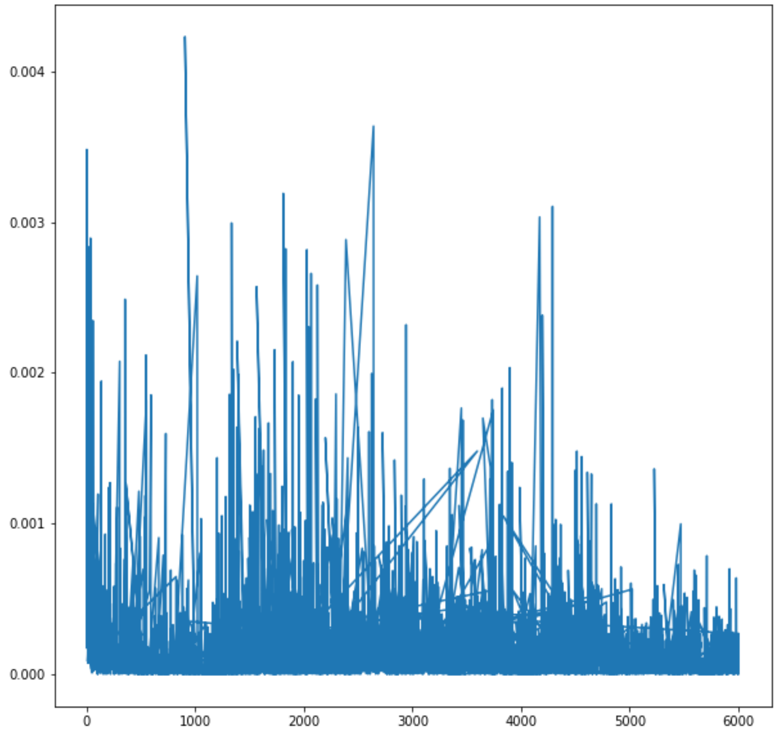


Auth Graph

Hub and Auth from Scratch Algorithm



Hub Graph



Auth Graph

Methodologies

Page Rank

Step 1: Find the indegree node of a node

Step 2: Calculate initial P(Node) as 1 / length of Indegree Node

Step 3: Now iterate through the Indegree Nodes of and node and find their respective prob by dividing the value which we get from the step 2 from the length of the outgoing degree nodes

Step 4: Sum all the values

Step 5: Store the page rank value of a node in a dictionary

```
pagerankValue = {}
for node in G.nodes():
 if lenDict[node] != 0:
   len1 = 1.0/(lenDict[node] + 1)
   pageRank
   for values in inedgeDict[node]:
```

```
    x = 0
    if lenDict2[values] != 0:
      x += len1/(lenDict2[values] + 1)
    pageRank = x
pagerankValue[node] = pageRank
```

Hubs and Authority

Hubs :- Outdegree

Authority :- Indegree

Algorithm

Step 1:- Calculate the sum of the Outdegree of Indegrees and strore it in a intNodeAuth dictionary

Step 2:- Calculate the sum of the Indegree of Outdegrees and strore it in a intNodeHub dictionary

Step 3:- Calculate thes auth sum of all the nodes and store it in authSum

Step 4:- Calculate thes hub sum of all the nodes and store it in hubSum

Step 5:- Normalize the auth and the hub of each nodes by dividing the auth value of a node with the authSum and hub by hub value of a node with hubSum

Step 6:- Store it in a dictionary

```
intNodeAuth = {}
intNodeHub = {}

for node in G.nodes():
 sum = 0
 for value in inedgeDict[node]:
   sum += lenDict2[value]
 intNodeAuth[node] = sum

for node in G.nodes():
 sum = 0
 for value in outedgeDict[node]:
   sum += lenDict[value]
 intNodeHub[node] = sum
authSum = 0
hubSum = 0
```

```
for key in intNodeAuth:
 authSum += intNodeAuth[key]
for key in intNodeHub:
 hubSum += intNodeHub[key]
finalAuthDict = {}
finalHubDict = {}

for node in G.nodes():
 nodeAuth = intNodeAuth[node]/authSum
 finalAuthDict[node] = nodeAuth

for node in G.nodes():
 nodeHub = intNodeHub[node]/hubSum
 finalHubDict[node] = nodeHub
```

Assumptions :- No Assumptions