

# ASSIGNMENT 3

Aditya Dixit (MT21004)

Shivang Kanwar (MT21084)

**Libraries Used:** Pandas, NumPy, Itertools, Matplotlib, Collections, Networkx, heapq

## Question 1 - [45 Points] Link Analysis

We are taking the WikiVote dataset. It contains two columns. The first column contains node numbers from where the edge starts, and the second column contains node numbers where the edge ends.

FromNodeId	ToNodeId
------------	----------

Reading dataset and calculating the number of nodes and edges in the network.

```
1 file = pd.read_csv("/content/drive/MyDrive/WikiVote.csv") #importing the dataset
2 start = file.iloc[:,0].to_numpy() #taking the first column
3 end = file.iloc[:,1].to_numpy() #taking the second column
4
5 in_dict={} #dictionaries for storing node as key value and neighbours list as value
6 out_dict={}
7
8 edge_list=[] #storing edge list
9 adjacency_matrix=[] #storing adjacency matrix
10 node_index_map={} #for mapping node to numbers starting from 0
11
12 n=np.append(start,end)
13 nodes = np.unique(n) #nodes store all the nodes present in graph
14 n=len(nodes) #n and e stores number of nodes and edges
15 e=len(start)
```

Mapping and calculating edge list, in\_dict, and out\_dict.

```
17 for i in range(0,n): #nodes are unevenly numbered so we are mapping them to numbers starting from zero to n
18     node_index_map[nodes[i]]=i
19
20 for i in range(0,e): #building edge list
21     el=[]
22     el.append(start[i])
23     el.append(end[i])
24     edge_list.append(el)
25
26 #creating in_dict containing edges coming to a node and out_dict containing edges going out of node
27 in_dict={key : [v[0] for v in val] for key, val in groupby(sorted(edge_list, key = lambda ele: ele[1]), key = lambda ele: ele[1])}
28 out_dict = {key : [v[1] for v in val] for key, val in groupby(sorted(edge_list, key = lambda ele: ele[0]), key = lambda ele: ele[0])}
29
```

Creating Adjacency Matrix and Printing all outputs

```
30 #creating adjacency matrix
31 for i in range(0,n):
32     temp=[0]*n
33     if nodes[i] in out_dict.keys():
34         for j in out_dict[nodes[i]]:
35             temp[node_index_map[j]]=1
36     adjacency_matrix.append(temp)
37
38 #printing output
39 print("Adjacency List Top 20 Rows:- ")
40 for i in adjacency_matrix[:20]:
41     print(i)
42 print("Edge List:- ", edge_list)
43 print("Total Number of Nodes are:-",n)
44 print("Total Number of edges are:-",e)
```

### OUTPUT

**Preview of Adjacency Matrix for 10 nodes and 10 columns:**

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
[0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
[0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

**Preview of Edge List for 5 edges:**

```
Edge List:-  [[30, 1412], [30, 3352], [30, 5254], [30, 5543], [30, 7478],
```

```
Total Number of Nodes are:- 7115
```

```
Total Number of edges are:- 103689
```

A common dictionary is built, containing node number as key and list of nodes connected to it by both in and out edge as value.

```
common_dict={}      #common dictionary cotaining both in and out node numbers for a node
avg_in_deg=0        #average degrees
avg_out_deg=0
density=0           #density

#loops for calculating common dictionary
for k in in_dict.keys():
    if k in out_dict.keys():
        common_dict[k]=list(set(in_dict[k]) | set(out_dict[k]))
        common_dict[k].sort()
    else:
        common_dict[k]=in_dict[k]
        common_dict[k].sort()

for k in out_dict.keys():
    if k not in common_dict.keys():
        common_dict[k]=out_dict[k]
        common_dict[k].sort()

common_dict=dict(sorted(common_dict.items())) #sorting
```

Average in and out degrees is calculated by adding all in degrees and out degrees and dividing them by the total number of nodes. Density and node with max in degree and out-degree are also calculated.

Average in degree = Sum of in degrees/total number of nodes

Average out-degree = Sum of out degrees/total number of nodes

DENSITY = no. of edges/((no. of nodes)\*(no. of nodes-1))

```
22 #finding in degree and out degree for each node
23 for key,val in in_dict.items():
24     in_dict[key]=len(val)
25     avg_in_deg+=len(val)
26 for key,val in out_dict.items():
27     out_dict[key]=len(val)
28     avg_out_deg+=len(val)
29
30 avg_in_deg/=n      #calculating average
31 avg_out_deg/=n
32
33 #printing output
34 print("Average In Degree:- ",avg_in_deg)
35 print("Average Out Degree:- ",avg_out_deg)
36 print("Node With Maximum In Degree:- ",max(zip(in_dict.values(), in_dict.keys()))[1])
37 print("Node With Maximum Out Degree:- ",max(zip(out_dict.values(), out_dict.keys()))[1])
38 print("Density of Network is:- ", e/(n*(n-1)))
```

## OUTPUT

Average In Degree:- 14.573295853829936  
Average Out Degree:- 14.573295853829936  
Node With Maximum In Degree:- 4037  
Node With Maximum Out Degree:- 2565  
Density of Network is:- 0.0020485375110809584

Finding probability or the fraction of time a degree appeared for a node.

We divide no of times a degree appeared from the total number of nodes to find the probability of a degree.

Probability(Degree)=no. Of times degree appeared/total number of nodes

```
1 #taking in degree and out degree for each node and sorting them
2 in_degrees=list(in_dict.values())
3 out_degrees=list(out_dict.values())
4 in_degrees.sort()
5 out_degrees.sort()
6
7 #taking count of how many times each degree appear and sorting
8 num_in_deg=Counter(in_degrees)
9 num_out_deg=Counter(out_degrees)
10 num_in_deg=sorted(num_in_deg.items())
11 num_out_deg=sorted(num_out_deg.items())
12
13 #finding probability of times each degree appeared
14 prob_in=[]
15 prob_out=[]
16 for i in num_in_deg:
17     p=i[1]/n
18     prob_in.append(p)
19
20 for i in num_out_deg:
21     p=i[1]/n
22     prob_out.append(p)
23
```

Plotting degree distribution for both in-degree and out-degree using matplotlib. We used a stem plot to show degree distribution.

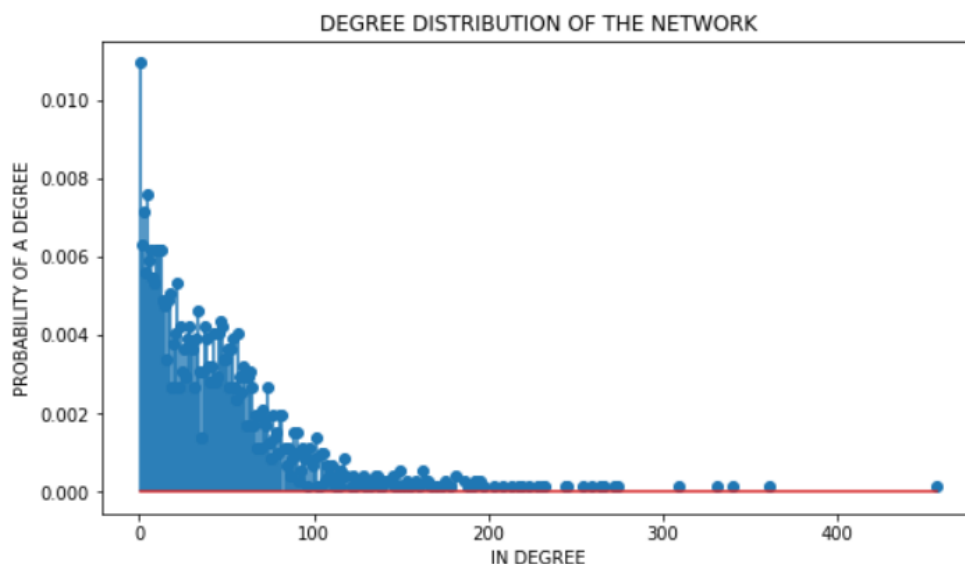
```
#taking unique values
in_degrees = list(set(in_degrees))
out_degrees = list(set(out_degrees))

#plotting graph
plt.figure(figsize=(9, 5))
plt.stem(in_degrees,prob_in,use_line_collection=True)
plt.title("DEGREE DISTRIBUTION OF THE NETWORK")
plt.xlabel("IN DEGREE")
plt.ylabel("PROBABILITY OF A DEGREE")

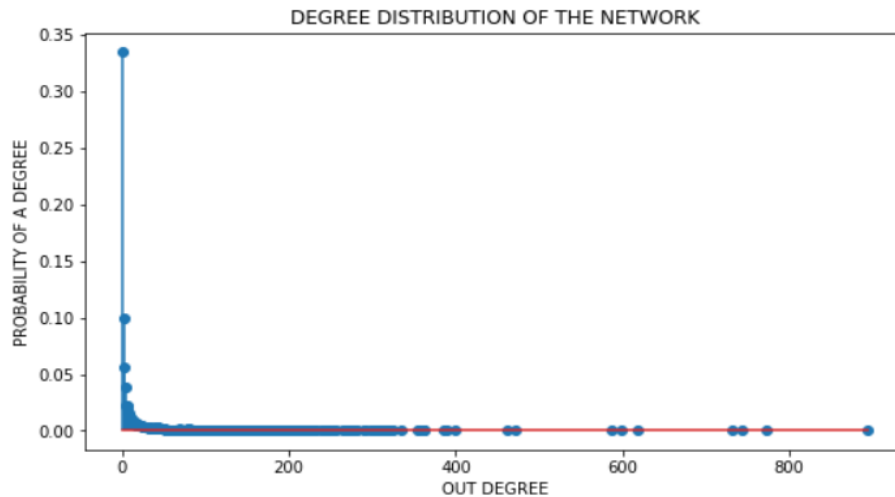
plt.figure(figsize=(9, 5))
plt.stem(out_degrees,prob_out,use_line_collection=True)
plt.title("DEGREE DISTRIBUTION OF THE NETWORK")
plt.xlabel("OUT DEGREE")
plt.ylabel("PROBABILITY OF A DEGREE")
plt.show()
```

## OUTPUT

### In-Degree Distribution Curve:



## Out-Degree Distribution Curve:



Finding the degree of each node and their neighbors link between themselves

```
1 deg=[]          #list which stores degree of each node
2 neigh_link=[]   # stores link between neighbours for each node
3
4 for i in common_dict.keys():      #taking degrees from common_dict
5     deg.append(len(common_dict[i]))
6
7 for i in common_dict.values():    #storing number of neighbours having link between them in neigh_link
8     a=0
9     for j in range(0,len(i)-1):
10        for k in range(j+1,len(i)):
11            if adjacency_matrix[node_index_map[i[j]]][node_index_map[i[k]]]==1:
12                a+=1
13    neigh_link.append(a)
```

Calculating local cluster coefficient for each node.

If the degree of a node  $< 2$ , then cluster coefficient = 0; else, cluster coefficient =  $(2 * \text{no. of neighbors of a node having a link between them}) / (\text{degree of the node} * (\text{degree of the node} - 1))$

```
clus_coeff=[]    #list containing localclustering coefficient of each node
clus_freq=[]     #list containing frequency of each local clustering coefficient
count_cc1=[]     #list containing unique local cluster coefficient

#calculating local cluster coefficient
for i in range(0,len(deg)):
    if deg[i]<2:
        clus_coeff.append(0)
    else:
        ab=(2*neigh_link[i])/(deg[i]*(deg[i]-1))
        clus_coeff.append(ab)
```

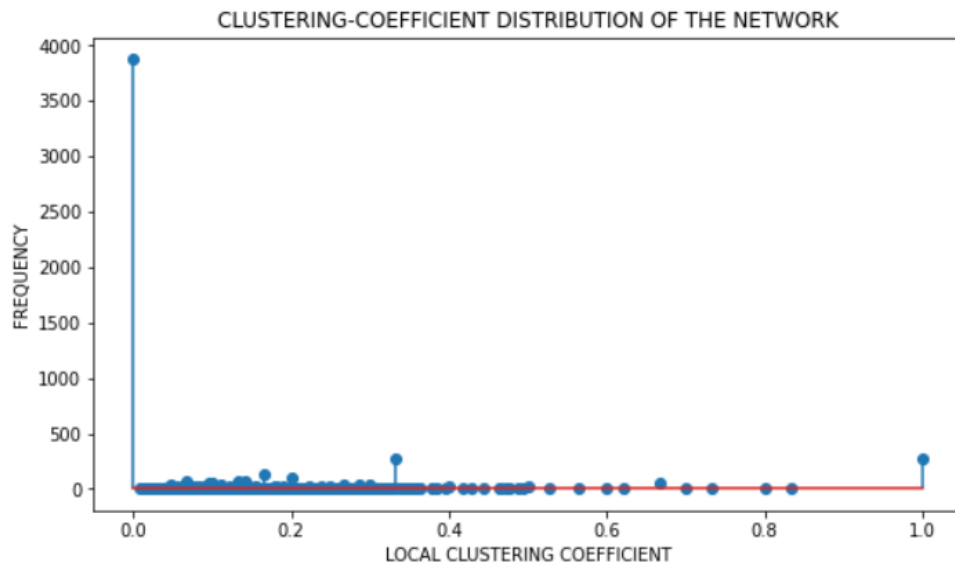
Plotting stem plot between local clustering coefficient and its frequency using matplotlib.

```
#number of times each local cluster coefficient appears is stored in dictionary count_cc
count_cc = sorted(Counter(clus_coeff).items())

#storing local cluster coefficient in count_cc1 and its frequency in clus_freq
for i in count_cc:
    clus_freq.append(i[0])
    count_cc1.append(i[1])

#plotting graph
plt.figure(figsize=(9, 5))
plt.stem(clus_freq,count_cc1,use_line_collection=True)
plt.title("CLUSTERING-COEFFICIENT DISTRIBUTION OF THE NETWORK")
plt.xlabel("LOCAL CLUSTERING COEFFICIENT")
plt.ylabel("FREQUENCY")
plt.show()
```

## OUTPUT



## Question 2 - [35 points] PageRank, Hubs, and Authority

Read data from the dataset “WikiVote.txt” and then create the node list and the adjacency list for all the pages by traversing through the CSV.

```
df = df.rename(columns=df.iloc[0]).drop(df.index[0])
df.head()
```

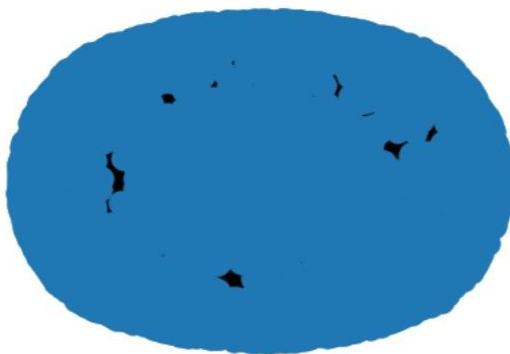
	FromNodeId	ToNodeId
1	30	1412
2	30	3352
3	30	5254
4	30	5543
5	30	7478

Analyzed the node list by finding the node count, smallest/largest node, and edge counts.

```
Nodes count: 7115
Smallest node: 3
Largest node: 8297
Edges count: 103689
Nodes: [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
Adjacency list: {30: [1412, 3352, 5254, 5543, 7478], 3: [28, 30, 39, 54, 108, 152, 178, 182, 214, 271, 286, 300, 348, 349, 371, 567, 581
```

Created graph using the node list and adjacency list for finding edges and drawing them.

```
nx.draw(G, cmap = plt.get_cmap('jet'))
```





## 1. [15 points] PageRank score for each node

Calculated Pagerank using the networkx API for each node of the graph.

### 1. PageRank score for each node

```
[18] #Calculate Pagerank using networkx
pagerank = nx.pagerank(G)
print("Pagerank: ", pagerank)

Pagerank: {1: 4.7660786889344714e-05, 2: 4.7660786889344714e-05, 3: 0.00019405306220767607, 4: 4.7660786889344714e-05, 5: 4.7660786889344714e-05, 6: 0.00029428173544419
```

## 2. [15 points] Authority and Hub score for each node

Calculated hub and authority using the networkx API for each graph node.

### 2. Authority and Hub score for each node

```
[11] #Calculate Hub, Authority using networkx
hub, authority = nx.hits(G)
print("Hub: ", hub)
print("Authority: ", authority)

Hub: {1: 0.0, 2: 0.0, 3: 4.021031639777644e-05, 4: 7.31960768582419e-05, 5: 3.5017884744336435e-05, 6: 0.001053987286176362, 7: 8.200618013274951e-05, 8: 0.000320012333
Authority: {1: 0.0, 2: 0.0, 3: 9.501171858460756e-05, 4: 0.0, 5: 0.0, 6: 6.398065594290167e-05, 7: 0.0, 8: 0.00018776680194921425, 9: 0.0, 10: 6.848818550186213e-05, 11
```

## 3. [5 points] Compare the results obtained from both the algorithms in parts 1 and 2 based on the node scores.

Numerical analysis:

Found the top 10 highest scoring PageRank, hub, and authority nodes.

```
#Find top 10 highest pagerank score nodes
highestPageranks = nlargest(10, pagerank, key=pagerank.get)
for node in highestPageranks:
    print(node, ': ', pagerank.get(node))
```

```
4037 : 0.004349070572777368
15 : 0.0034730989804857898
6634 : 0.0033100848020831634
2625 : 0.0030997039963222037
2398 : 0.002458151746777155
2470 : 0.0023869661176755936
2237 : 0.0023599269720822776
4191 : 0.002136846007631464
7553 : 0.0020490084362567926
5254 : 0.002029058301350638
```

```
[25] #Find top 10 highest authority score nodes
highestAuthorities = nlargest(10, authority, key=authority.get)
for node in highestAuthorities:
    print(node, ': ', authority.get(node))
```

2398 : 0.0025801471780088738  
4037 : 0.002573241124229792  
3352 : 0.0023284150914976826  
1549 : 0.002303731480457179  
762 : 0.0022558748562871394  
3089 : 0.0022534066884511627  
1297 : 0.0022501446366627233  
2565 : 0.0022235641039536126  
15 : 0.002201543492565581  
2625 : 0.002197896803403073

```
[26] #Find top 10 highest hub score nodes
highestHubs = nlargest(10, hub, key=hub.get)
for node in highestHubs:
    print(node, ': ', hub.get(node))
```

2565 : 0.007940492708143145  
766 : 0.007574335297501249  
2688 : 0.006440248991029864  
457 : 0.006416870490261073  
1166 : 0.006010567902411202  
1549 : 0.005720754058269244  
11 : 0.004921182063808105  
1151 : 0.004572040717564085  
1374 : 0.00446788792711107  
1133 : 0.003918881732057349

Picked the highest PageRank and authority scoring node and compared their incoming and outgoing links.

```
[44] print(G.in_edges(4037))
      print(len(G.in_edges(4037)))
```

[ (6, 4037), (15, 4037), (47, 4037), (72, 4037), (71, 4037), (87, 4037), (68, 4037), (109, 4037), (113, 4037), (127, 4037), (136, 4037), (143, 4

457

```
[45] print(G.out_edges(4037))
      print(len(G.out_edges(4037)))
```

[ (4037, 15), (4037, 825), (4037, 1385), (4037, 2014), (4037, 2958), (4037, 3498), (4037, 3717), (4037, 4138), (4037, 4256), (4037, 4402), (4037, 4520), (4037, 4638), (4037, 4756), (4037, 4874), (4037, 4992), (4037, 5110), (4037, 5228), (4037, 5346), (4037, 5464), (4037, 5582), (4037, 5700), (4037, 5818), (4037, 5936), (4037, 6054), (4037, 6172), (4037, 6290), (4037, 6408), (4037, 6526), (4037, 6644), (4037, 6762), (4037, 6880), (4037, 6998), (4037, 7116), (4037, 7234), (4037, 7352), (4037, 7470), (4037, 7588), (4037, 7706), (4037, 7824), (4037, 7942), (4037, 8060), (4037, 8178), (4037, 8296), (4037, 8414), (4037, 8532), (4037, 8650), (4037, 8768), (4037, 8886), (4037, 9004), (4037, 9122), (4037, 9240), (4037, 9358), (4037, 9476), (4037, 9594), (4037, 9712), (4037, 9830), (4037, 9948), (4037, 10066), (4037, 10184), (4037, 10302), (4037, 10420), (4037, 10538), (4037, 10656), (4037, 10774), (4037, 10892), (4037, 11010), (4037, 11128), (4037, 11246), (4037, 11364), (4037, 11482), (4037, 11600), (4037, 11718), (4037, 11836), (4037, 11954), (4037, 12072), (4037, 12190), (4037, 12308), (4037, 12426), (4037, 12544), (4037, 12662), (4037, 12780), (4037, 12898), (4037, 13016), (4037, 13134), (4037, 13252), (4037, 13370), (4037, 13488), (4037, 13606), (4037, 13724), (4037, 13842), (4037, 13960), (4037, 14078), (4037, 14196), (4037, 14314), (4037, 14432), (4037, 14550), (4037, 14668), (4037, 14786), (4037, 14904), (4037, 15022), (4037, 15140), (4037, 15258), (4037, 15376), (4037, 15494), (4037, 15612), (4037, 15730), (4037, 15848), (4037, 15966), (4037, 16084), (4037, 16202), (4037, 16320), (4037, 16438), (4037, 16556), (4037, 16674), (4037, 16792), (4037, 16910), (4037, 17028), (4037, 17146), (4037, 17264), (4037, 17382), (4037, 17500), (4037, 17618), (4037, 17736), (4037, 17854), (4037, 17972), (4037, 18090), (4037, 18208), (4037, 18326), (4037, 18444), (4037, 18562), (4037, 18680), (4037, 18798), (4037, 18916), (4037, 19034), (4037, 19152), (4037, 19270), (4037, 19388), (4037, 19506), (4037, 19624), (4037, 19742), (4037, 19860), (4037, 19978), (4037, 20096), (4037, 20214), (4037, 20332), (4037, 20450), (4037, 20568), (4037, 20686), (4037, 20804), (4037, 20922), (4037, 21040), (4037, 21158), (4037, 21276), (4037, 21394), (4037, 21512), (4037, 21630), (4037, 21748), (4037, 21866), (4037, 21984), (4037, 22102), (4037, 22220), (4037, 22338), (4037, 22456), (4037, 22574), (4037, 22692), (4037, 22810), (4037, 22928), (4037, 23046), (4037, 23164), (4037, 23282), (4037, 23400), (4037, 23518), (4037, 23636), (4037, 23754), (4037, 23872), (4037, 23990), (4037, 24108), (4037, 24226), (4037, 24344), (4037, 24462), (4037, 24580), (4037, 24698), (4037, 24816), (4037, 24934), (4037, 25052), (4037, 25170), (4037, 25288), (4037, 25406), (4037, 25524), (4037, 25642), (4037, 25760), (4037, 25878), (4037, 25996), (4037, 26114), (4037, 26232), (4037, 26350), (4037, 26468), (4037, 26586), (4037, 26704), (4037, 26822), (4037, 26940), (4037, 27058), (4037, 27176), (4037, 27294), (4037, 27412), (4037, 27530), (4037, 27648), (4037, 27766), (4037, 27884), (4037, 28002), (4037, 28120), (4037, 28238), (4037, 28356), (4037, 28474), (4037, 28592), (4037, 28710), (4037, 28828), (4037, 28946), (4037, 29064), (4037, 29182), (4037, 29300), (4037, 29418), (4037, 29536), (4037, 29654), (4037, 29772), (4037, 29890), (4037, 30008), (4037, 30126), (4037, 30244), (4037, 30362), (4037, 30480), (4037, 30598), (4037, 30716), (4037, 30834), (4037, 30952), (4037, 31070), (4037, 31188), (4037, 31306), (4037, 31424), (4037, 31542), (4037, 31660), (4037, 31778), (4037, 31896), (4037, 32014), (4037, 32132), (4037, 32250), (4037, 32368), (4037, 32486), (4037, 32604), (4037, 32722), (4037, 32840), (4037, 32958), (4037, 33076), (4037, 33194), (4037, 33312), (4037, 33430), (4037, 33548), (4037, 33666), (4037, 33784), (4037, 33902), (4037, 34020), (4037, 34138), (4037, 34256), (4037, 34374), (4037, 34492), (4037, 34610), (4037, 34728), (4037, 34846), (4037, 34964), (4037, 35082), (4037, 35200), (4037, 35318), (4037, 35436), (4037, 35554), (4037, 35672), (4037, 35790), (4037, 35908), (4037, 36026), (4037, 36144), (4037, 36262), (4037, 36380), (4037, 36498), (4037, 36616), (4037, 36734), (4037, 36852), (4037, 36970), (4037, 37088), (4037, 37206), (4037, 37324), (4037, 37442), (4037, 37560), (4037, 37678), (4037, 37796), (4037, 37914), (4037, 38032), (4037, 38150), (4037, 38268), (4037, 38386), (4037, 38504), (4037, 38622), (4037, 38740), (4037, 38858), (4037, 38976), (4037, 39094), (4037, 39212), (4037, 39330), (4037, 39448), (4037, 39566), (4037, 39684), (4037, 39802), (4037, 39920), (4037, 40038), (4037, 40156), (4037, 40274), (4037, 40392), (4037, 40510), (4037, 40628), (4037, 40746), (4037, 40864), (4037,

Similarly picked the highest hub scoring nodes and verified that their outgoing links were greater than the incoming links.

```
[46] print(G.out_edges(2565))
      print(len(G.out_edges(2565)))
```

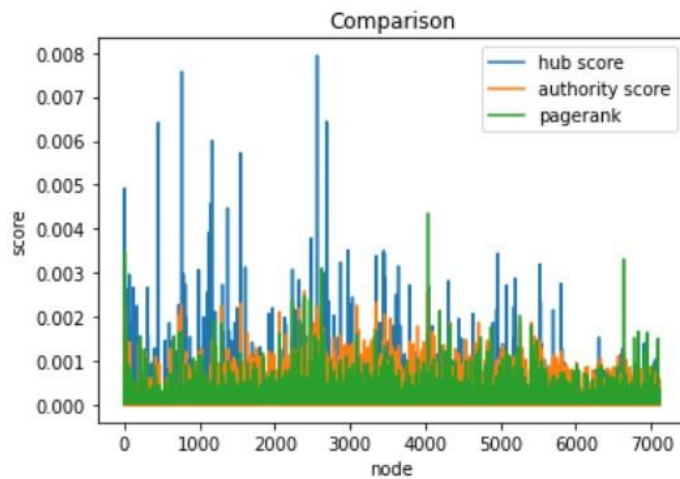
[(2565, 56), (2565, 155), (2565, 204), (2565, 214), (2565, 285), (2565, 290), (2565, 346), (2565, 403), (2565, 417), (2565, 425), (2565, 893)

```
[47] print(G.in_edges(2565))
      print(len(G.in_edges(2565)))
```

[(11, 2565), (14, 2565), (20, 2565), (47, 2565), (29, 2565), (72, 2565), (87, 2565), (99, 2565), (68, 2565), (122, 2565), (127, 2565), (1

### Visual Comparison:

Plotted a graph for all nodes against their PageRank, hub, and authority scores.



Plotted scatter plots for comparing PageRank and hits algo by plotting graphs for

(1).pagerank and hub score

(2).pagerank and authority score

