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.

```
file = pd.read_csv("/content/drive/MyDrive/WikiVote.csv") #importing the dataset
start = file.iloc[:,0].to_numpy()  #taking the first column
end = file.iloc[:,1].to_numpy()    #taking the second column

in_dict={}    #dictionaries for storing node as key value and neighbours list as value
out_dict={}

edge_list=[]      #storing edge list
adjacency_matrix=[]  #storing adjacancy matrix
node_index_map={}   #for mapping node to numbers starting from 0

n=np.append(start,end)
nodes = np.unique(n)               #nodes store all the nodes present in graph
n=len(nodes)                       #n and e stores number of nodes and edges
e=len(start)
```

Mapping and calculating edge list, in_dict, and out_dict.

```
for i in range(0,n):           #nodes are unevenly numbered so we are maping them to numbers starting from zero to n
  node_index_map[nodes[i]]=i

for i in range(0,e):           #building edge list
  el=[]
  el.append(start[i])
  el.append(end[i])
  edge_list.append(el)

#creating in_dict contaning edges coming to a node and out_dict containing edges going out of node
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])}
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])}
```

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))

```
#finding in degree and out degree for each node
for key,val in in_dict.items():
  in_dict[key]=len(val)
  avg_in_deg+=len(val)
for key,val in out_dict.items():
  out_dict[key]=len(val)
  avg_out_deg+=len(val)

avg_in_deg/=n    #calculating average
avg_out_deg/=n

#printing output
print("Average In Degree:- ",avg_in_deg)
print("Average Out Degree:- ",avg_out_deg)
print("Node With Maximum In Degree:- ",max(zip(in_dict.values(), in_dict.keys()))[1])
print("Node With Maximum Out Degree:- ",max(zip(out_dict.values(), out_dict.keys()))[1])
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

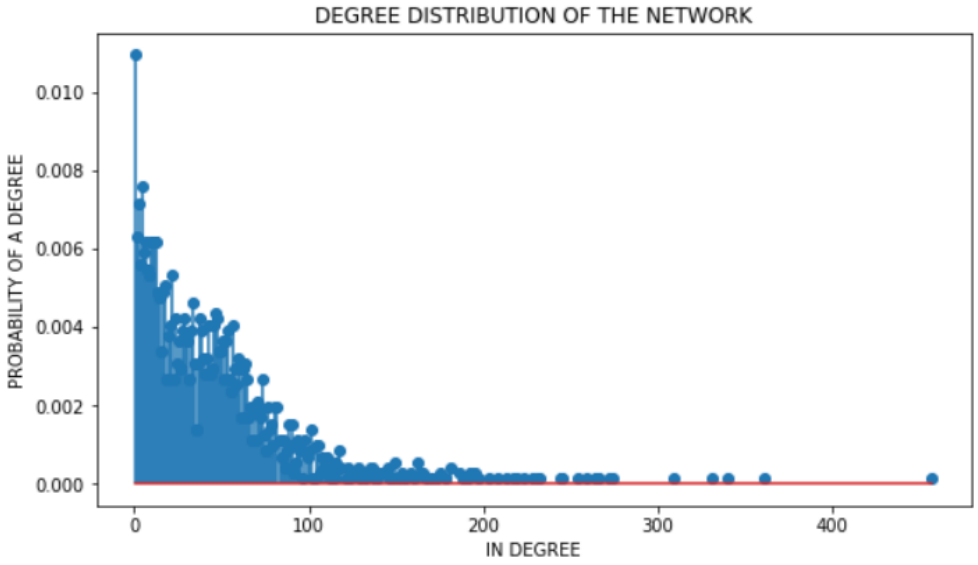
```
#taking in degree and out degree for each node and sorting them
in_degrees=list(in_dict.values())
out_degrees=list(out_dict.values())
in_degrees.sort()
out_degrees.sort()

#taking count of how many times each degree appear and sorting
num_in_deg=Counter(in_degrees)
num_out_deg=Counter(out_degrees)
num_in_deg=sorted(num_in_deg.items())
num_out_deg=sorted(num_out_deg.items())

#finding probability of times each degree appeared
prob_in=[]
prob_out=[]
for i in num_in_deg:
  p=i[1]/n
  prob_in.append(p)

for i in num_out_deg:
  p=i[1]/n
  prob_out.append(p)
```

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

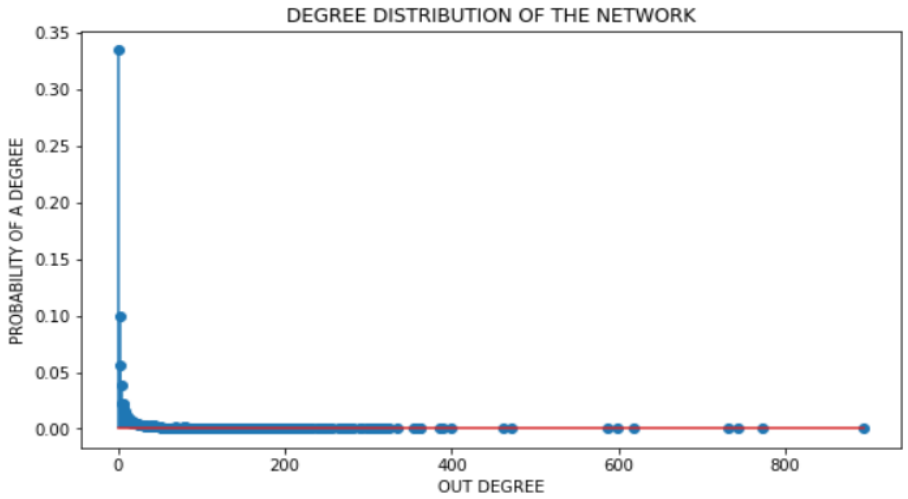
```
#taking unique values
in_degrees = list(set(in_degrees))
out_degrees = list(set(out_degrees))

#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

```
deg=[]         #list which stores degree of each node
neigh_link=[]  # stores link between neighbours for each node

for i in common_dict.keys():        #taking degrees from commmon_dict
  deg.append(len(common_dict[i]))

for i in common_dict.values():  #storing number of neighbours having link between them in neigh_link
  a=0
  for j in range(0,len(i)-1):
    for k in range(j+1,len(i)):
      if adjacency_matrix[node_index_map[i[j]]][node_index_map[i[k]]]==1:
        a+=1
  neigh_link.append(a)
```

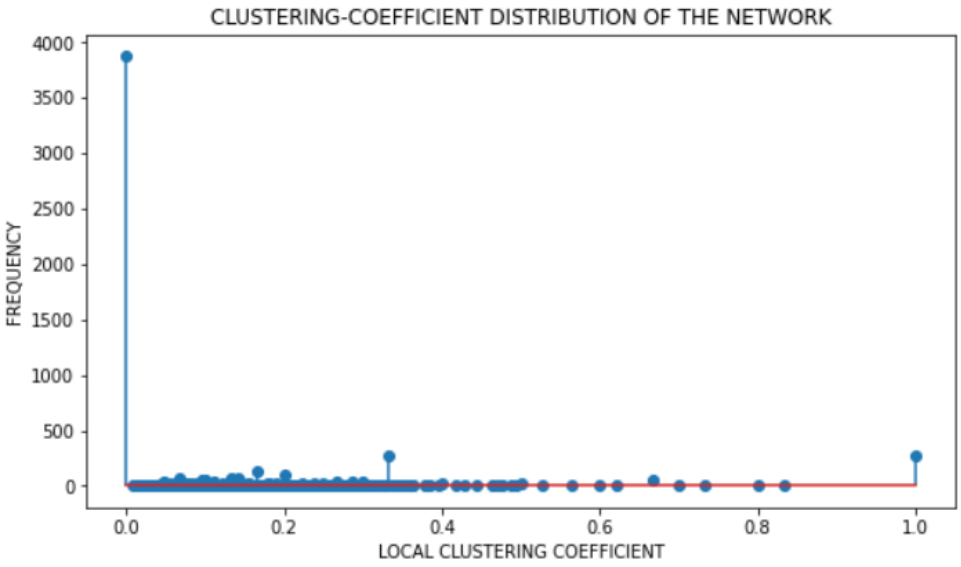
Calculating local cluster coefficient for each node.
If the degree of a node <2, then cluster coefficient = 0; else, cluster coefficient = (2*no. of neighbors of a node having a link between them)/(degree of the node)/(degree of the node -1)

```
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()
```

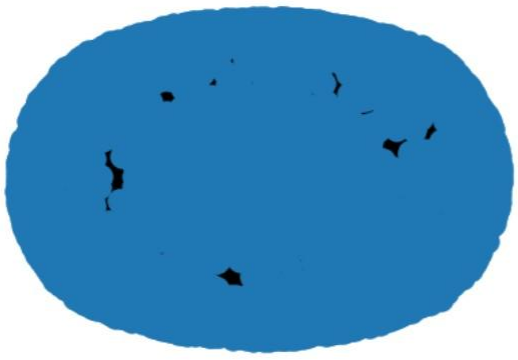
	FromNodeId	ToNodeId
1	30	1412
2	30	3352
3	30	5254
4	30	5543
5	30	7478

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 : 0.0045720407017564085
1374 : 0.004467888792711107
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
15
```

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)))
```

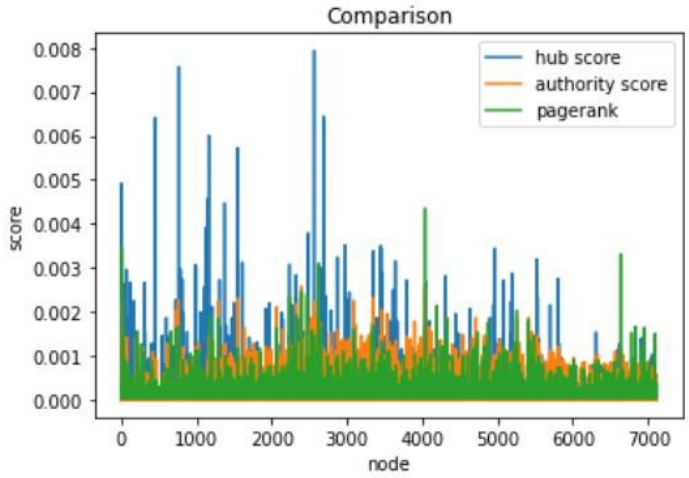
```
[(2565, 56), (2565, 155), (2565, 204), (2565, 214), (2565, 285), (2565, 290), (2565, 346), (2565, 403), (2565, 417), (2565, 425), (2565,
893
```

```
[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
274
```

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