Assignment No. 3
Report*(Group 97)*

MT21074	**2020020**	**MT22067**
Samiksha Garg	Akshat Tilak	Shambhavi Pathak

Q1.

The dataset used: `"email-Eu-core.txt.gz"`

Dataset Description:

The email-Eu-core.txt.gz is a compressed text file containing a dataset of email communication networks. The dataset was created by crawling a large European research institution's email server, which included emails sent and received between members of the institution over a period of several months.

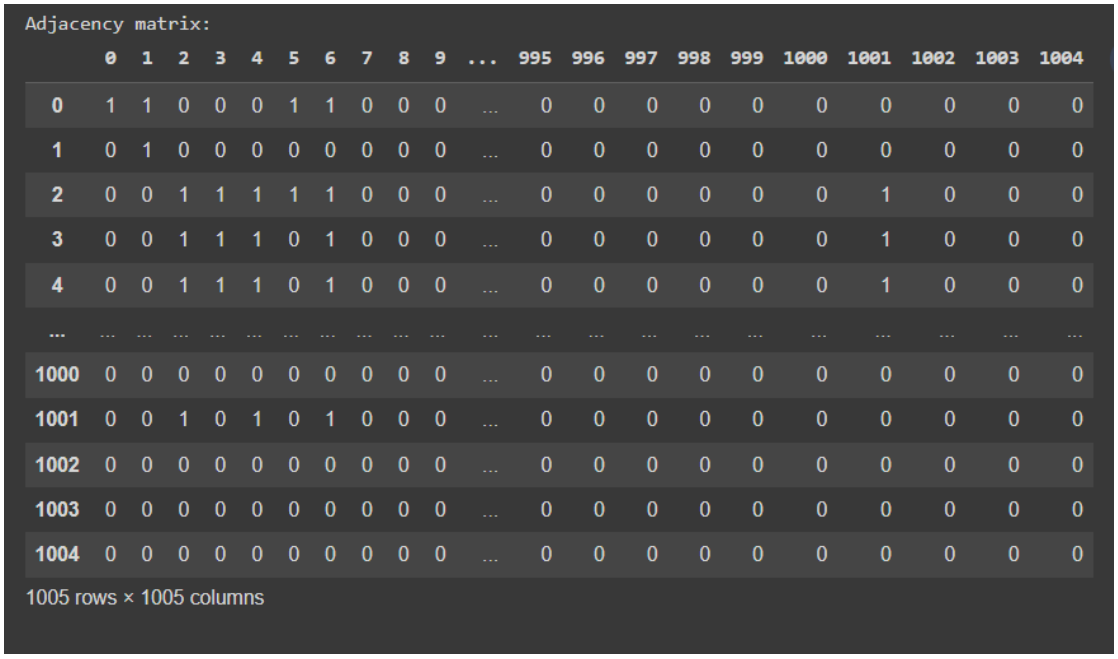
The file contains a list of email addresses and their corresponding numerical IDs, representing the individuals who sent or received emails. Each line of the file represents a single email and contains two IDs separated by a space representing the sender and recipient of the email. The dataset includes a total of 1,005,506 emails sent between 986 different individuals.

The dataset additionally includes information about the departmental affiliations of each individual, with each person belonging to one of 42 departments at the research institute. This information serves as the "ground-truth" community memberships of the nodes.
It should be noted that this dataset corresponds to the "core" of the email-EuAll network, which includes links not only within the institution but also to individuals outside the institution (although their node IDs differ from those in this dataset).

This dataset has been used in various research studies related to social network analysis and email communication, including studies on community detection, link prediction, and influence maximization. It is a valuable resource for researchers interested in studying email communication patterns and networks.

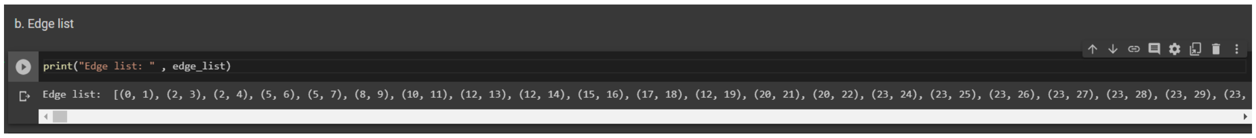
Represent the network in terms of its 'adjacency matrix' as well as 'edge list'

Adjacency Matrix:



```
Adjacency matrix:
         0  1  2  3  4  5  6  7  8  9  ...  995  996  997  998  999  1000  1001  1002  1003  1004
   0     1  1  0  0  0  1  1  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     0     0     0
   2     0  0  1  1  1  1  1  0  0  0  ...    0    0    0    0    0     0     1     0     0     0
   3     0  0  1  1  1  0  1  0  0  0  ...    0    0    0    0    0     0     1     0     0     0
   4     0  0  1  1  1  0  1  0  0  0  ...    0    0    0    0    0     0     1     0     0     0
  ...   ... ... ... ... ... ... ... ... ... ... ...  ...  ...  ...  ...  ...   ...   ...   ...   ...   ...
 1000    0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0     0     0     0     0     0
 1001    0  0  1  0  1  0  1  0  0  0  ...    0    0    0    0    0     0     0     0     0     0
 1002    0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0     0     0     0     0     0
 1003    0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0     0     0     0     0     0
 1004    0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0     0     0     0     0     0
1005 rows × 1005 columns
```

Edge list:

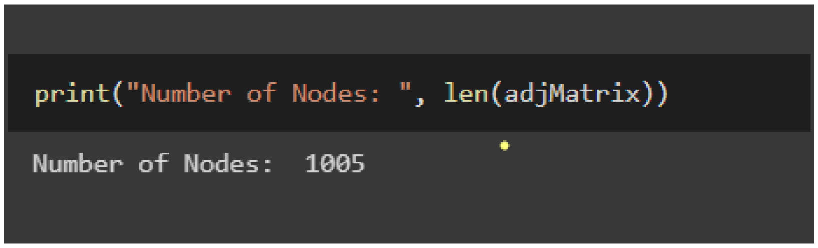


b. Edge list

```
print("Edge list: " , edge_list)
```

```
Edge list:  [(0, 1), (2, 3), (2, 4), (5, 6), (5, 7), (8, 9), (10, 11), (12, 13), (12, 14), (15, 16), (17, 18), (12, 19), (20, 21), (20, 22), (23, 24), (23, 25), (23, 26), (23, 27), (23, 28), (23, 29), (23,
```

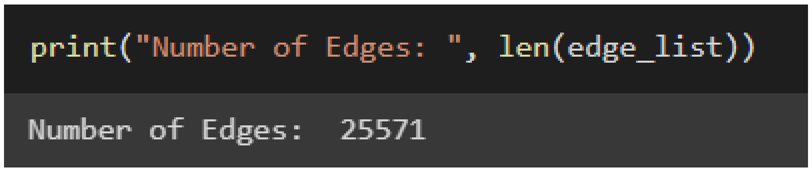
1. Number of Nodes:



```
print("Number of Nodes: ", len(adjMatrix))
```

```
Number of Nodes:  1005
```

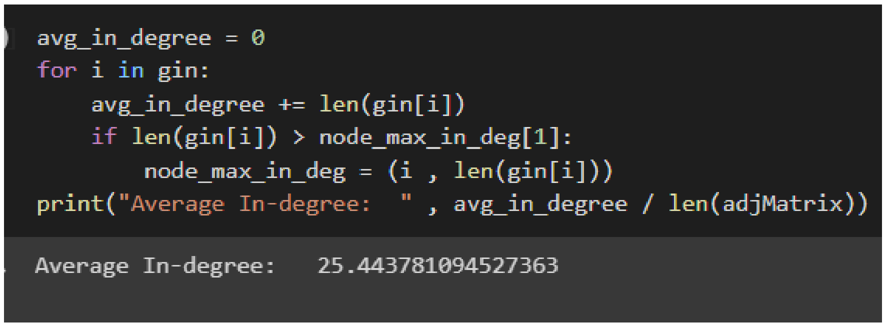
2. Number of Edges:



```
print("Number of Edges: ", len(edge_list))
```

```
Number of Edges:  25571
```

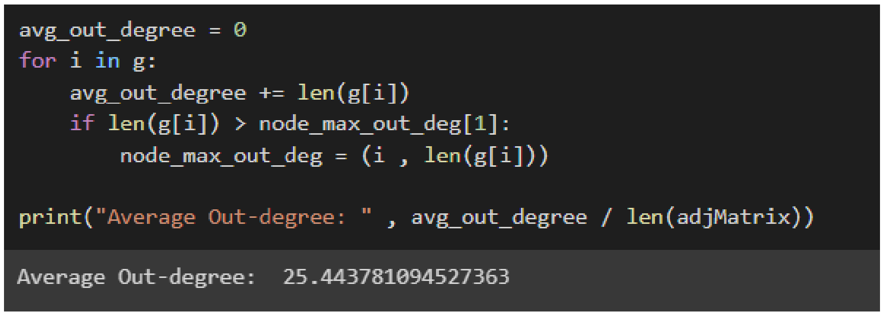
3. Avg In-degree:



```
avg_in_degree = 0
for i in gin:
    avg_in_degree += len(gin[i])
    if len(gin[i]) > node_max_in_deg[1]:
        node_max_in_deg = (i , len(gin[i]))
print("Average In-degree:  " , avg_in_degree / len(adjMatrix))
```

```
Average In-degree:   25.443781094527363
```

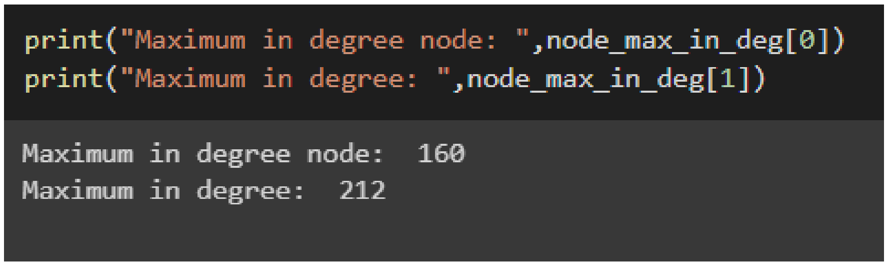
4. Avg. Out-Degree:



```
avg_out_degree = 0
for i in g:
    avg_out_degree += len(g[i])
    if len(g[i]) > node_max_out_deg[1]:
        node_max_out_deg = (i , len(g[i]))

print("Average Out-degree: " , avg_out_degree / len(adjMatrix))
```

```
Average Out-degree:  25.443781094527363
```

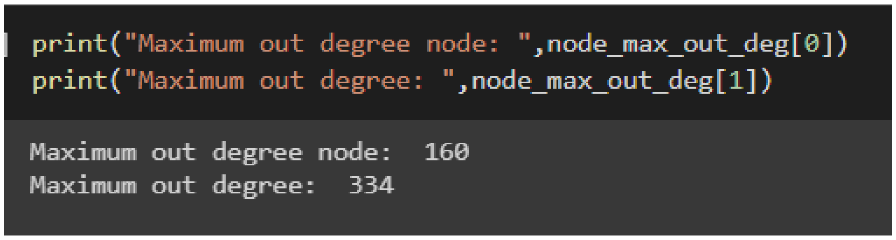
5. Node with Max In-degree:



```
print("Maximum in degree node: ",node_max_in_deg[0])
print("Maximum in degree: ",node_max_in_deg[1])
```

```
Maximum in degree node:  160
Maximum in degree:  212
```

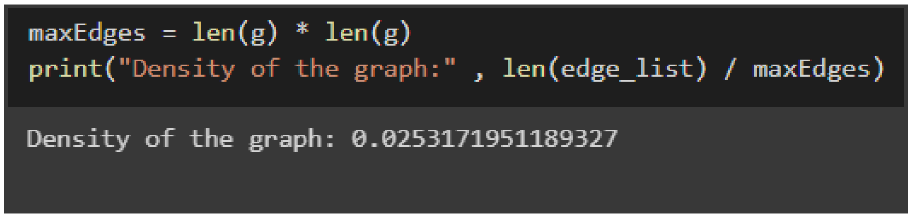
6. Node with Max out-degree:



```
print("Maximum out degree node: ",node_max_out_deg[0])
print("Maximum out degree: ",node_max_out_deg[1])
```

```
Maximum out degree node:  160
Maximum out degree:  334
```

7. The density of the network:

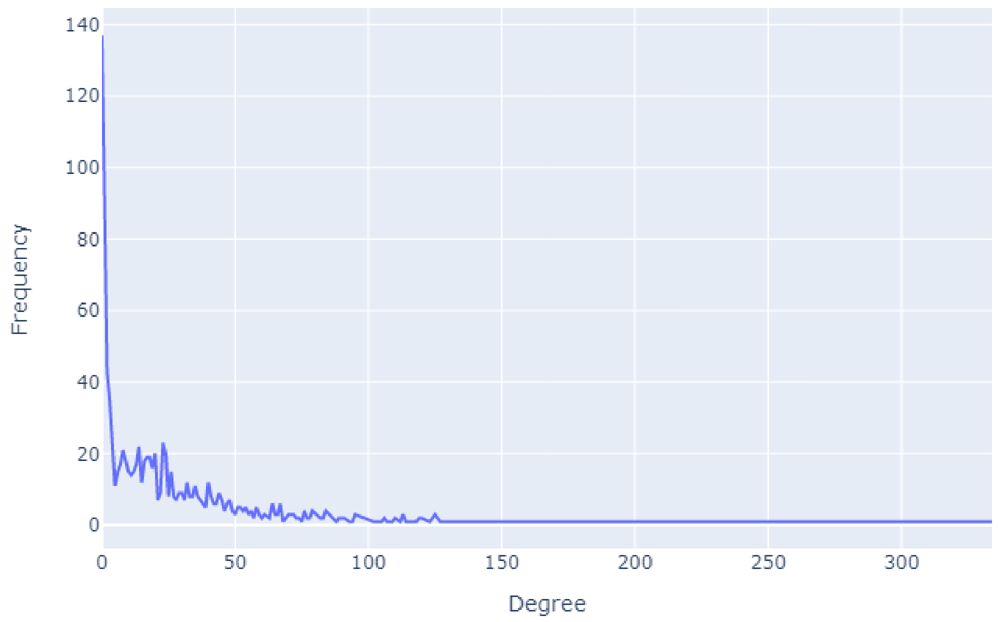


```
maxEdges = len(g) * len(g)
print("Density of the graph:" , len(edge_list) / maxEdges)
```

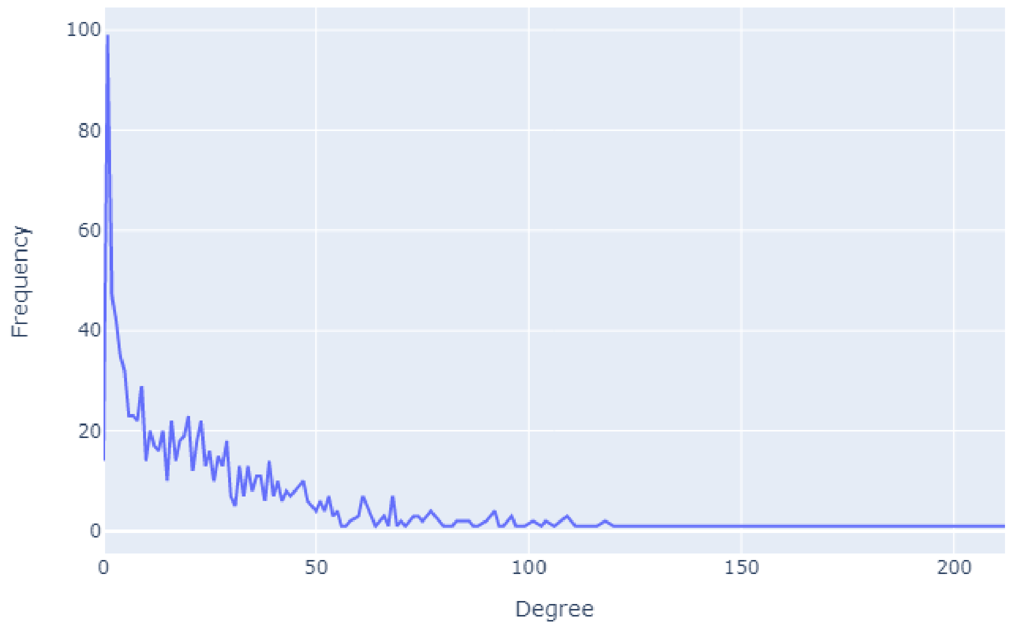
```
Density of the graph: 0.0253171951189327
```

Plot degree distribution of the network (in case of a directed graph, plot in-degree and out-degree separately)

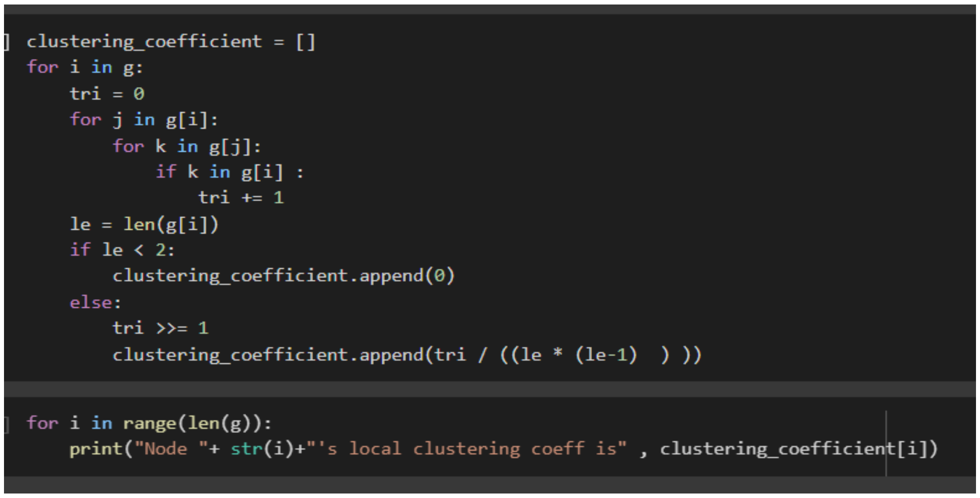
Plot of Out-degree distribution



Plot of In-degree distribution

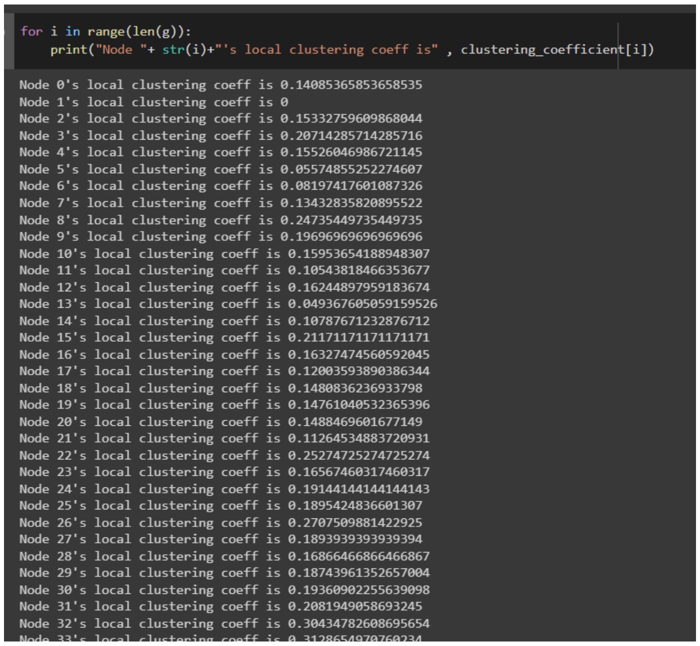


Calculate the local clustering coefficient of each node and plot the clustering-coefficient distribution (lcc vs frequency of lcc) of the network



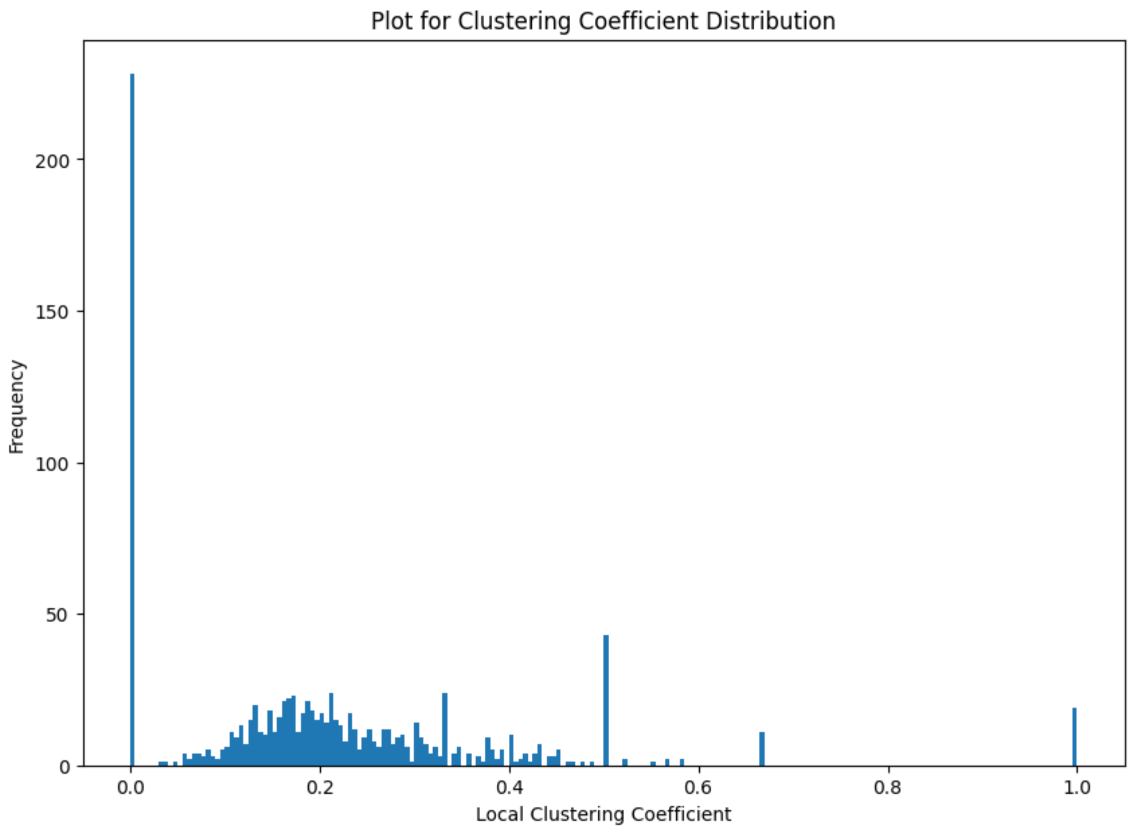
```python
clustering_coefficient = []
for i in g:
    tri = 0
    for j in g[i]:
        for k in g[j]:
            if k in g[i] :
                tri += 1
    le = len(g[i])
    if le < 2:
        clustering_coefficient.append(0)
    else:
        tri >>= 1
        clustering_coefficient.append(tri / ((le * (le-1)  ) ))
```

```python
for i in range(len(g)):
    print("Node "+ str(i)+"'s local clustering coeff is" , clustering_coefficient[i])
```



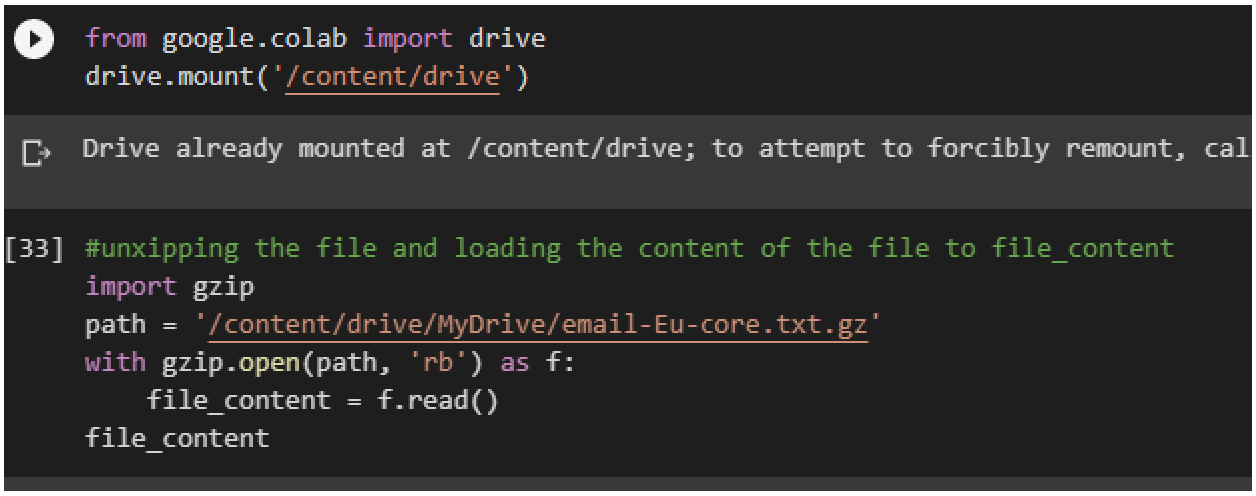
```python
for i in range(len(g)):
    print("Node "+ str(i)+"'s local clustering coeff is" , clustering_coefficient[i])
```

```
Node 0's local clustering coeff is 0.14085365853658535
Node 1's local clustering coeff is 0
Node 2's local clustering coeff is 0.15332759609868044
Node 3's local clustering coeff is 0.20714285714285716
Node 4's local clustering coeff is 0.15526046986721145
Node 5's local clustering coeff is 0.05574855252274607
Node 6's local clustering coeff is 0.08197417601087326
Node 7's local clustering coeff is 0.13432835820895522
Node 8's local clustering coeff is 0.24735449735449735
Node 9's local clustering coeff is 0.19696969696969696
Node 10's local clustering coeff is 0.15953654188948307
Node 11's local clustering coeff is 0.10543818466353677
Node 12's local clustering coeff is 0.16244897959183674
Node 13's local clustering coeff is 0.049367605059159526
Node 14's local clustering coeff is 0.10787671232876712
Node 15's local clustering coeff is 0.21171171171171171
Node 16's local clustering coeff is 0.16327474560592045
Node 17's local clustering coeff is 0.12003593890386344
Node 18's local clustering coeff is 0.1480836236933798
Node 19's local clustering coeff is 0.14761040532365396
Node 20's local clustering coeff is 0.1488469601677149
Node 21's local clustering coeff is 0.11264534883720931
Node 22's local clustering coeff is 0.25274725274725274
Node 23's local clustering coeff is 0.16567460317460317
Node 24's local clustering coeff is 0.19144144144144143
Node 25's local clustering coeff is 0.1895424836601307
Node 26's local clustering coeff is 0.2707509881422925
Node 27's local clustering coeff is 0.1893939393939394
Node 28's local clustering coeff is 0.16866466866466867
Node 29's local clustering coeff is 0.18743961352657004
Node 30's local clustering coeff is 0.19360902255639098
Node 31's local clustering coeff is 0.2081949058693245
Node 32's local clustering coeff is 0.30434782608695654
Node 33's local clustering coeff is 0.3128654970760234
```



Q2.Question 2 - [35 points] PageRank, Hubs and Authority
For the dataset chosen in the above question, calculate the following:
Steps:

1. Mounting Drive
2. Unzipping the file and reading the content

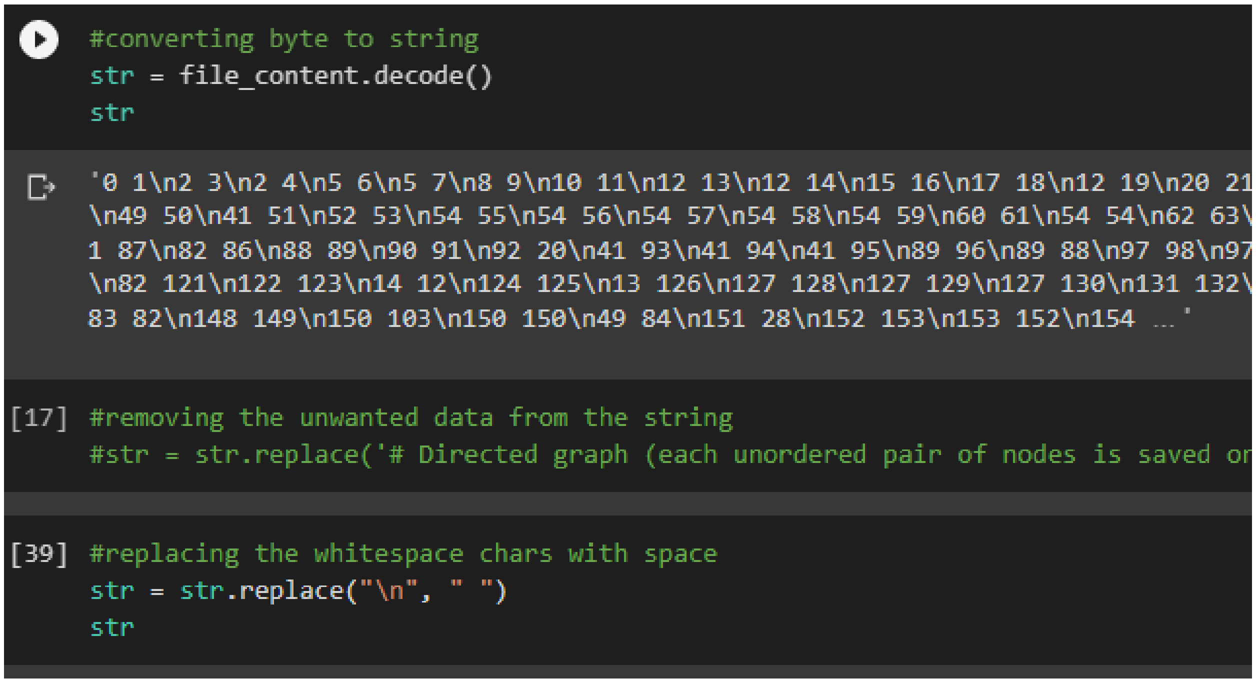


```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, cal
```

```
[33] #unxipping the file and loading the content of the file to file_content
     import gzip
     path = '/content/drive/MyDrive/email-Eu-core.txt.gz'
     with gzip.open(path, 'rb') as f:
         file_content = f.read()
     file_content
```

3. Converting the file to string
4. Removing whitespaces from the string



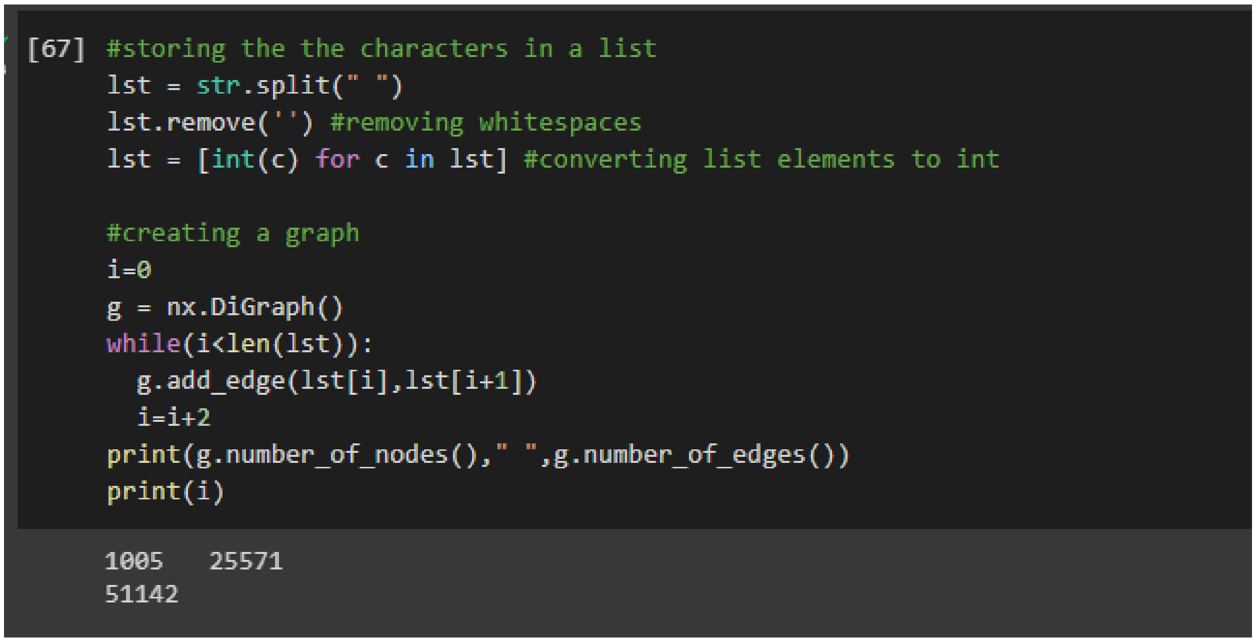
```
#converting byte to string
str = file_content.decode()
str
```

```
'0 1\n2 3\n2 4\n5 6\n5 7\n8 9\n10 11\n12 13\n12 14\n15 16\n17 18\n12 19\n20 21
\n49 50\n41 51\n52 53\n54 55\n54 56\n54 57\n54 58\n54 59\n60 61\n54 54\n62 63\
1 87\n82 86\n88 89\n90 91\n92 20\n41 93\n41 94\n41 95\n89 96\n89 88\n97 98\n97
\n82 121\n122 123\n14 12\n124 125\n13 126\n127 128\n127 129\n127 130\n131 132\
83 82\n148 149\n150 103\n150 150\n49 84\n151 28\n152 153\n153 152\n154 ...'
```

```
[17] #removing the unwanted data from the string
     #str = str.replace('# Directed graph (each unordered pair of nodes is saved or
```

```
[39] #replacing the whitespace chars with space
     str = str.replace("\n", " ")
     str
```

5. Splitting the string on the basis of space and storing as a list using .split() function
6. Creating the directed graph and using NetworkX function DiGraph and adding edges
7. On printing the number of edges and vertices returning the same value as cited by the website



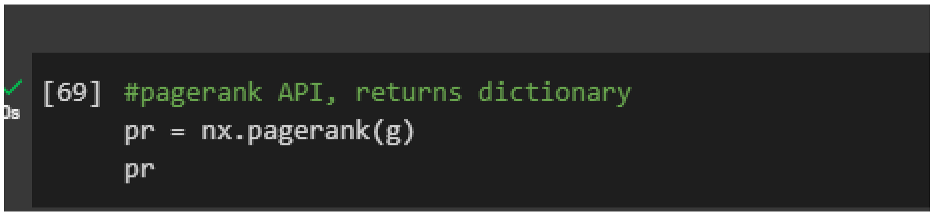
```
[67] #storing the the characters in a list
     lst = str.split(" ")
     lst.remove('') #removing whitespaces
     lst = [int(c) for c in lst] #converting list elements to int

     #creating a graph
     i=0
     g = nx.DiGraph()
     while(i<len(lst)):
       g.add_edge(lst[i],lst[i+1])
       i=i+2
     print(g.number_of_nodes()," ",g.number_of_edges())
     print(i)
```

```
1005   25571
51142
```

1. [15 points] PageRank score for each node

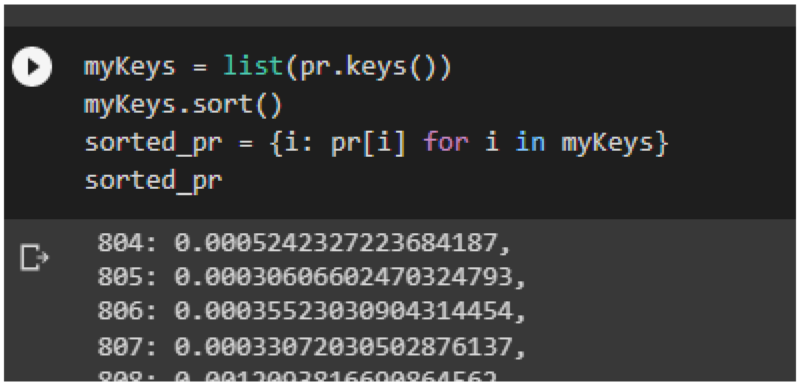
Steps:

1. Called the pagerank function and passed the graph, it returns a dictionary where keys are nodes and values are it corresponding pagerank score



```
[69] #pagerank API, returns dictionary
     pr = nx.pagerank(g)
     pr
```

2. Sorting the dictionary based on the keys



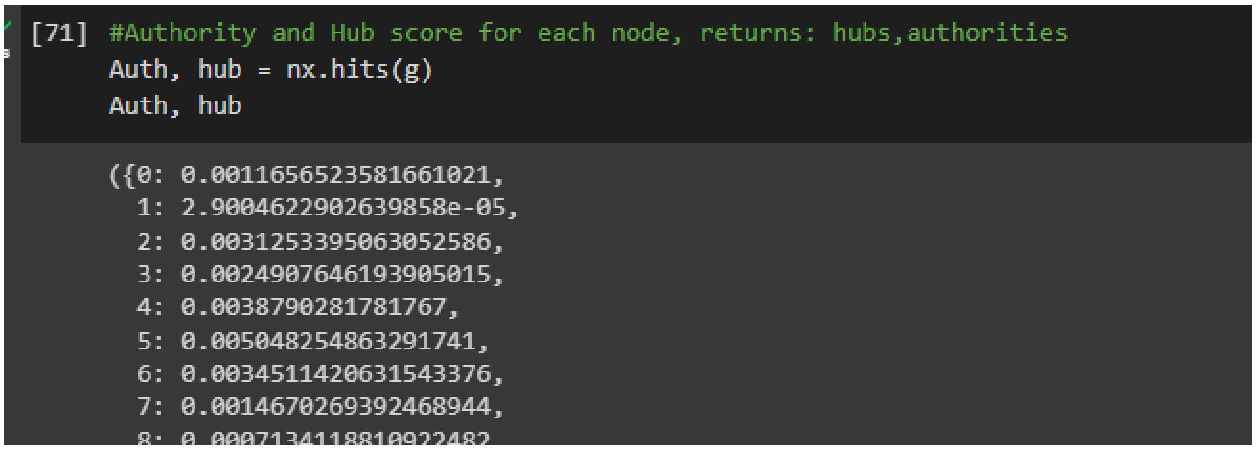
```
myKeys = list(pr.keys())
myKeys.sort()
sorted_pr = {i: pr[i] for i in myKeys}
sorted_pr
```

```
804: 0.0005242327223684187,
805: 0.00030606602470324793,
806: 0.00035523030904314454,
807: 0.00033072030502876137,
808: 0.0012092816609864562
```

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

Steps:

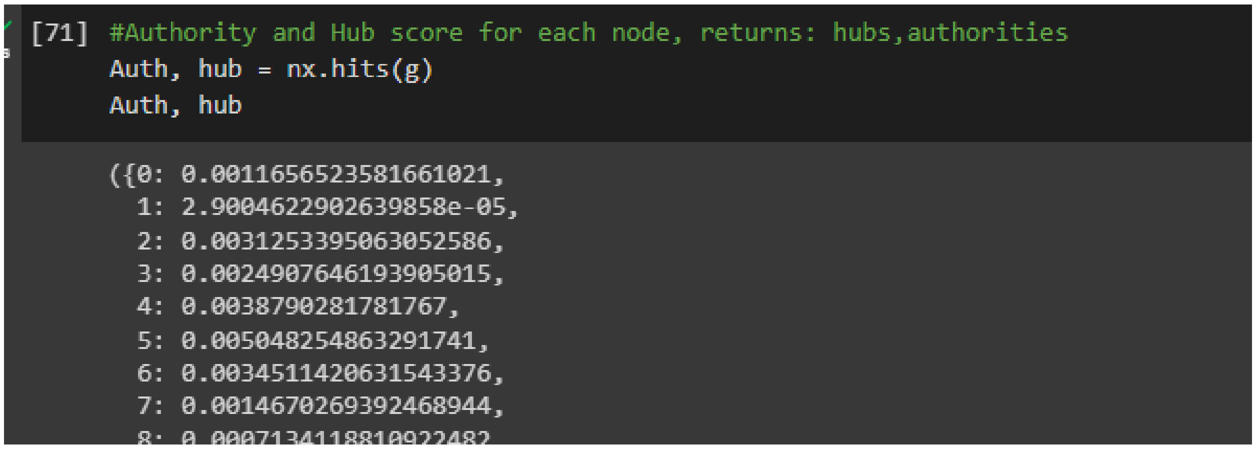
1. Called the hits function from the NetworkX, which returns to tuples of dictionaries (hubs, authorities)



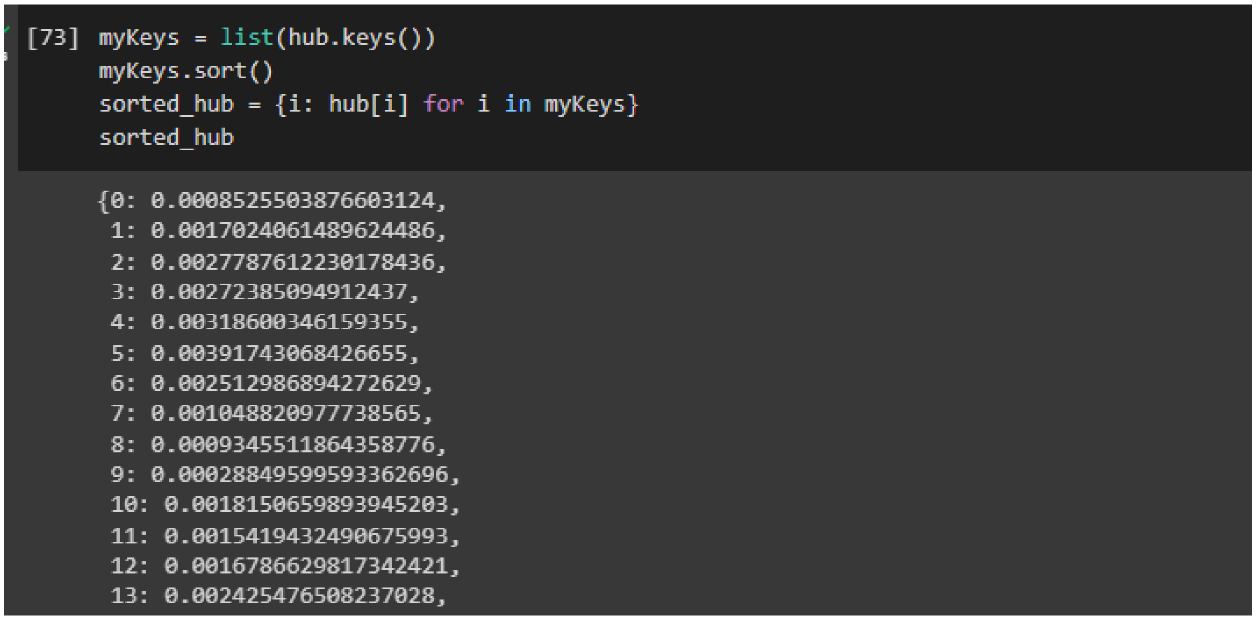
```
[71] #Authority and Hub score for each node, returns: hubs,authorities
     Auth, hub = nx.hits(g)
     Auth, hub
```

```
({0: 0.0011656523581661021,
  1: 2.9004622902639858e-05,
  2: 0.0031253395063052586,
  3: 0.0024907646193905015,
  4: 0.0038790281781767,
  5: 0.005048254863291741,
  6: 0.0034511420631543376,
  7: 0.0014670269392468944,
  8: 0.0007134118810922482
```

2. Sorted the auth dictionary and hub dictionary



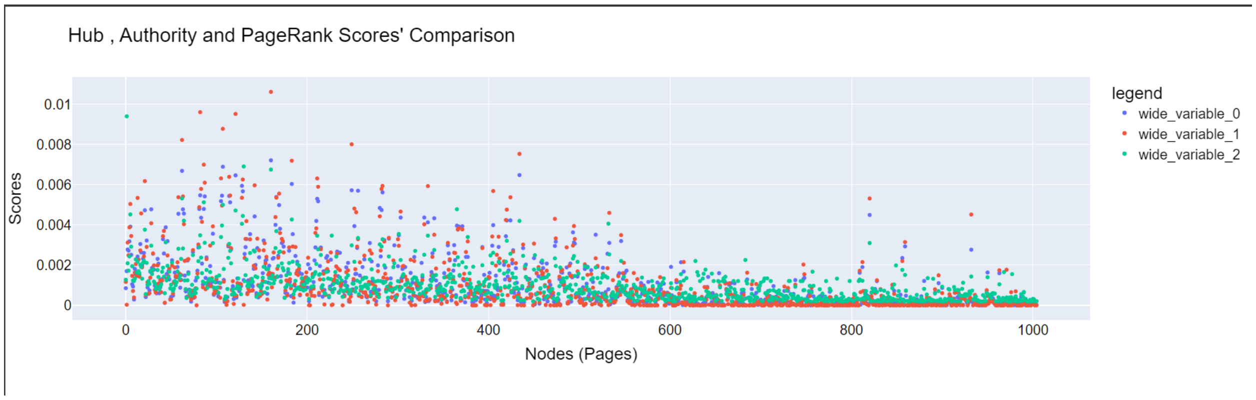
```
[71] #Authority and Hub score for each node, returns: hubs,authorities
     Auth, hub = nx.hits(g)
     Auth, hub

     ({0: 0.0011656523581661021,
       1: 2.9004622902639858e-05,
       2: 0.0031253395063052586,
       3: 0.0024907646193905015,
       4: 0.0038790281781767,
       5: 0.005048254863291741,
       6: 0.0034511420631543376,
       7: 0.0014670269392468944,
       8: 0.0007134118810922482
```



```
[73] myKeys = list(hub.keys())
     myKeys.sort()
     sorted_hub = {i: hub[i] for i in myKeys}
     sorted_hub

     {0: 0.0008525503876603124,
      1: 0.0017024061489624486,
      2: 0.0027787612230178436,
      3: 0.00272385094912437,
      4: 0.00318600346159355,
      5: 0.00391743068426655,
      6: 0.002512986894272629,
      7: 0.0010488209777738565,
      8: 0.0009345511864358776,
      9: 0.00028849599593362696,
      10: 0.0018150659893945203,
      11: 0.0015419432490675993,
      12: 0.0016786629817342421,
      13: 0.0024254765082370228,
```

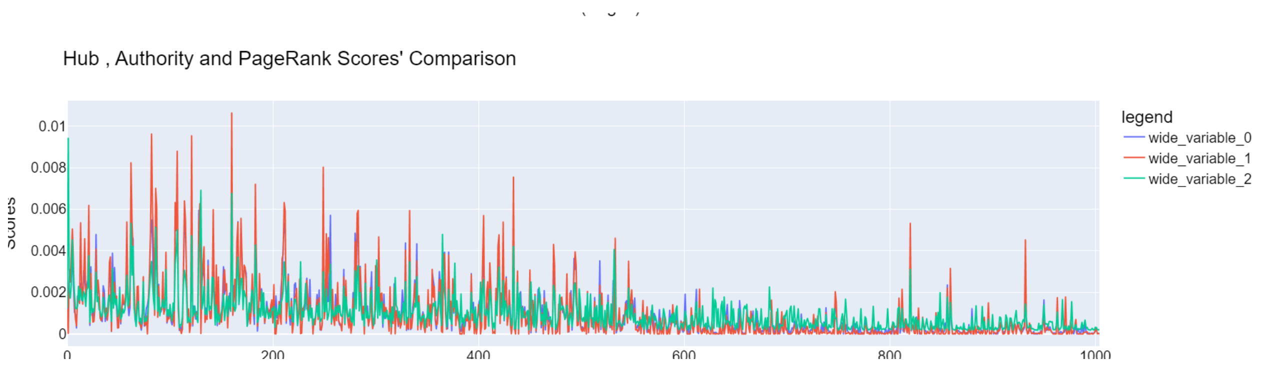
[5 points] Compare the results obtained from both the algorithms in parts 1 and 2 based on the node

A scatter plot depicting the Hub, Authority and PageRank Scores



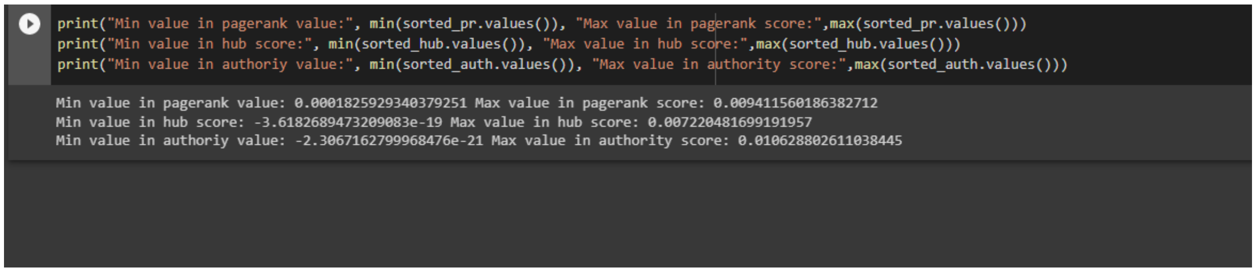
In the following graph, wide_variable1 refers to the Hub Score, wide_variable2 Authority score and wide_variabl3 refers to PageRank score

Line Plot depicting the Hub, Authority and PageRank Scores



In the following graph, wide_variable1 refers to the Hub Score, wide_variable2 Authority score and wide_variable3 refers to PageRank score

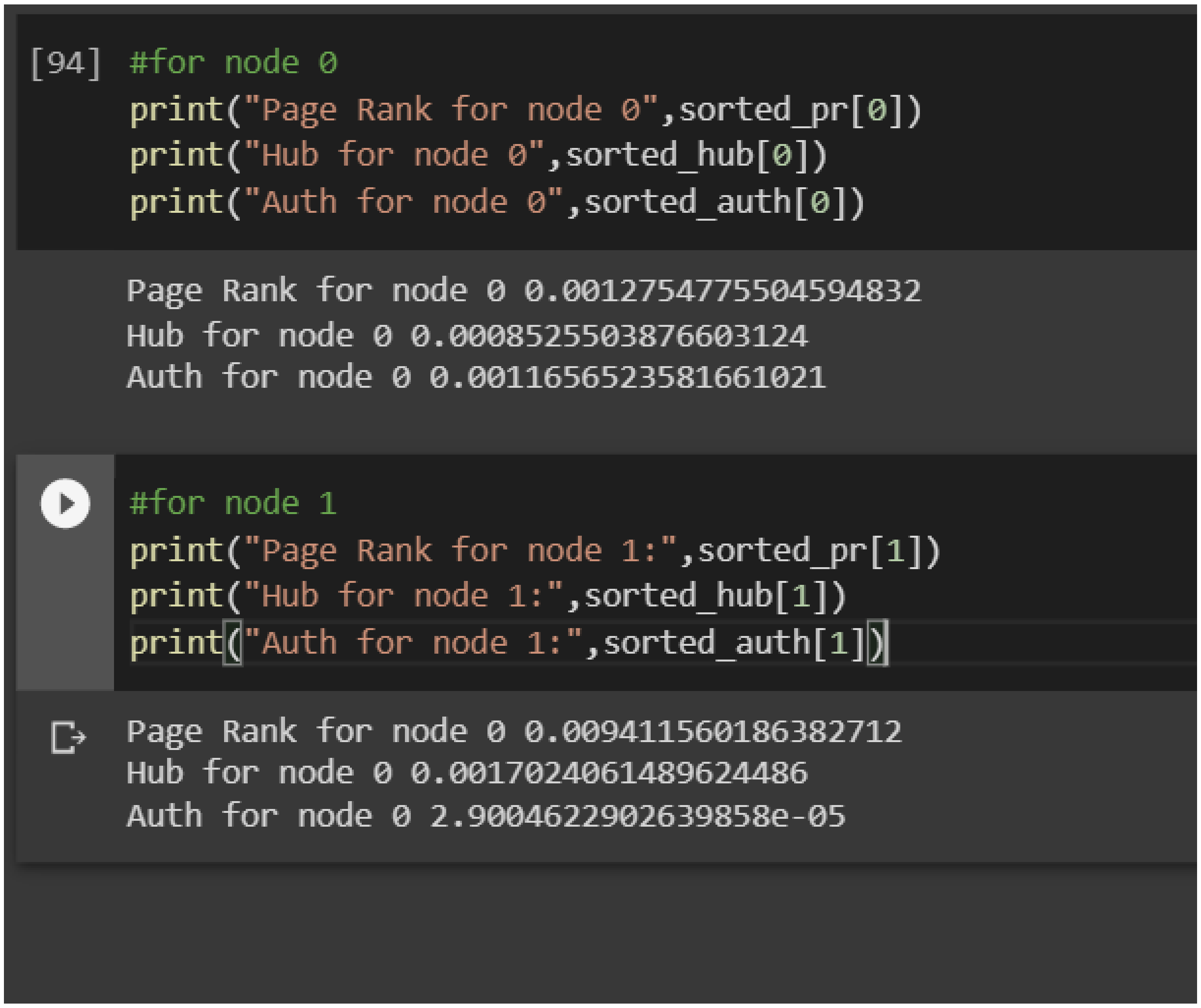
Minimum and maximum values for PageRank, Auth and hub



```
print("Min value in pagerank value:", min(sorted_pr.values()), "Max value in pagerank score:",max(sorted_pr.values()))
print("Min value in hub score:", min(sorted_hub.values()), "Max value in hub score:",max(sorted_hub.values()))
print("Min value in authoriy value:", min(sorted_auth.values()), "Max value in authority score:",max(sorted_auth.values()))
```

```
Min value in pagerank value: 0.0001825929340379251 Max value in pagerank score: 0.009411560186382712
Min value in hub score: -3.6182689473209083e-19 Max value in hub score: 0.007220481699191957
Min value in authoriy value: -2.3067162799968476e-21 Max value in authority score: 0.010628802611038445
```

A comparison between the Pagerank, Auth and Hub scores for node 0 and 1



```
[94] #for node 0
     print("Page Rank for node 0",sorted_pr[0])
     print("Hub for node 0",sorted_hub[0])
     print("Auth for node 0",sorted_auth[0])
```

```
Page Rank for node 0 0.0012754775504594832
Hub for node 0 0.0008525503876603124
Auth for node 0 0.0011656523581661021
```

```
#for node 1
print("Page Rank for node 1:",sorted_pr[1])
print("Hub for node 1:",sorted_hub[1])
print("Auth for node 1:",sorted_auth[1])
```

```
Page Rank for node 0 0.009411560186382712
Hub for node 0 0.0017024061489624486
Auth for node 0 2.9004622902639858e-05
```

Observation
The **authority** score measures the overall quality and credibility of a web page based on the quality and relevance of its content, as well as the quantity and quality of links pointing to it from other authoritative websites.

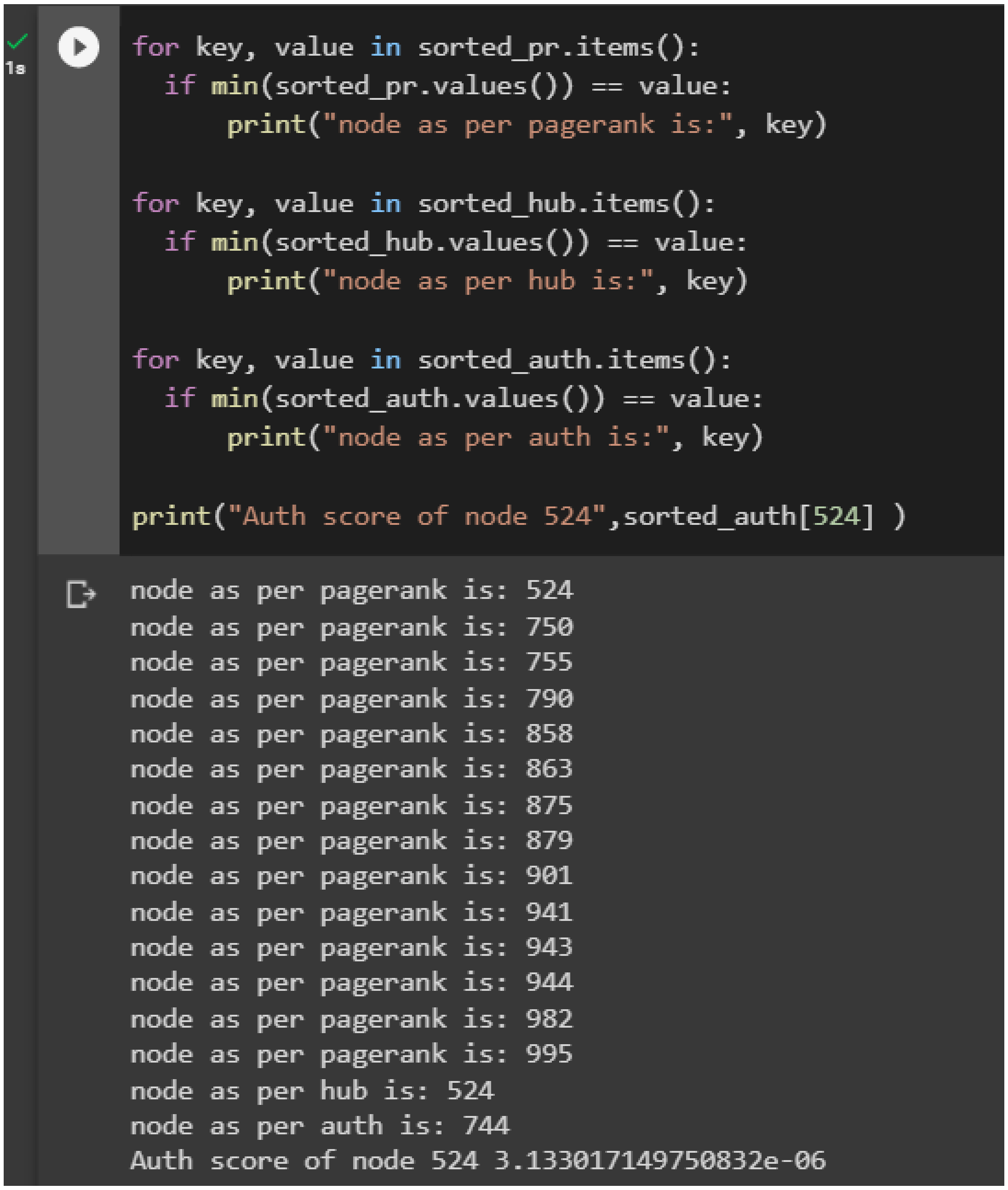
The **hub** score, on the other hand, measures the degree to which a web page is a central hub for information on a particular topic, based on the quality and relevance of its content as well as the number and quality of links pointing to other authoritative pages on the same topic.

PageRank works by assigning a score to each web page based on the number and quality of links pointing to it from other web pages. The more links a page has from other high-quality, relevant pages, the higher its PageRank score will be.

An authority score that is negative may indicate that the website has low-quality or spammy content, or that it has been penalized by search engines for violating their guidelines.

It may also suggest that the website has very few links pointing to it or that the links pointing to it are of low quality.

a negative hub score may indicate that the website is not seen as a valuable source of information on a particular topic, either because it lacks high-quality content or because it has few links pointing to other authoritative pages on the same topic.

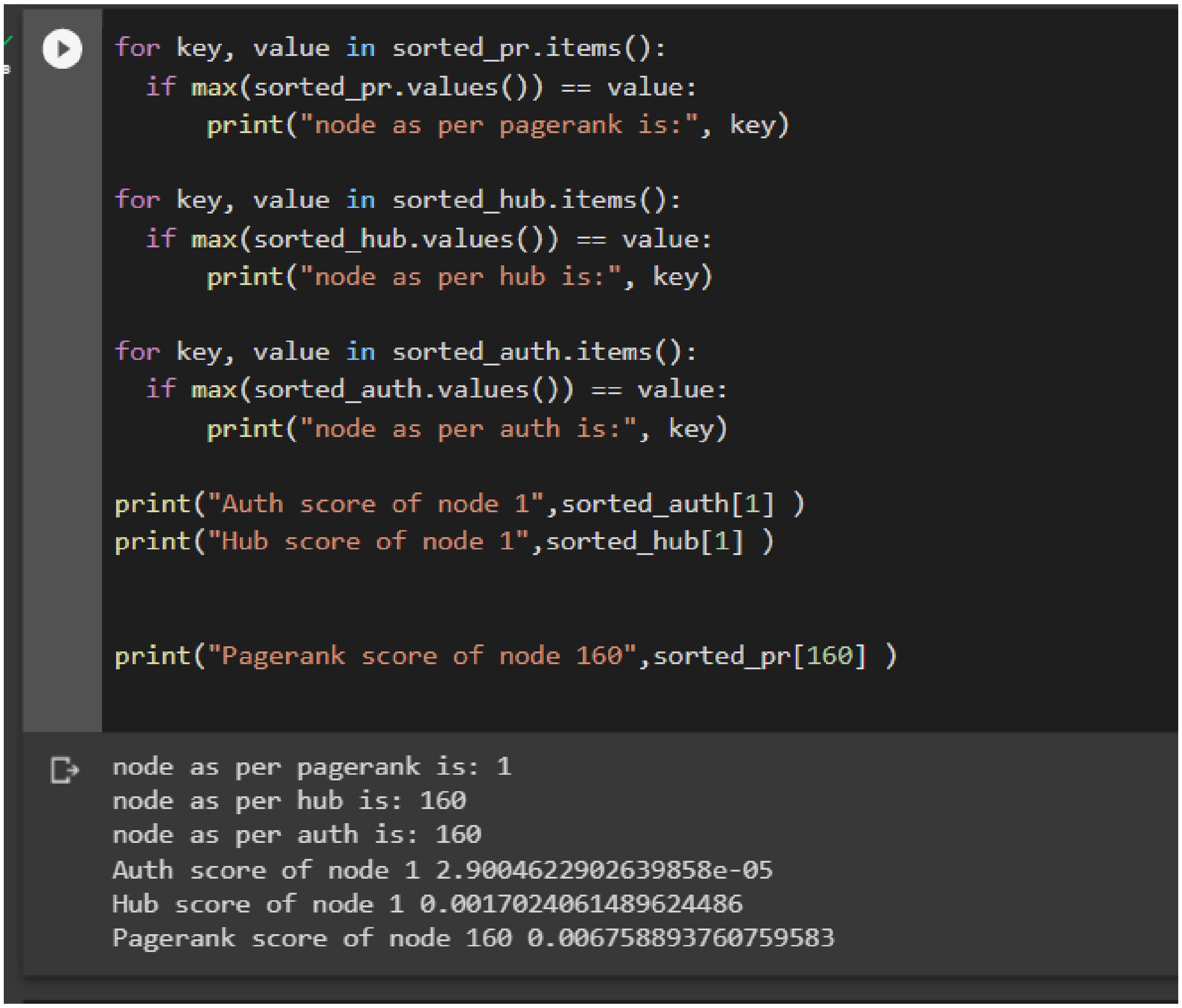


```python
for key, value in sorted_pr.items():
  if min(sorted_pr.values()) == value:
      print("node as per pagerank is:", key)

for key, value in sorted_hub.items():
  if min(sorted_hub.values()) == value:
      print("node as per hub is:", key)

for key, value in sorted_auth.items():
  if min(sorted_auth.values()) == value:
      print("node as per auth is:", key)

print("Auth score of node 524",sorted_auth[524] )
```

```
node as per pagerank is: 524
node as per pagerank is: 750
node as per pagerank is: 755
node as per pagerank is: 790
node as per pagerank is: 858
node as per pagerank is: 863
node as per pagerank is: 875
node as per pagerank is: 879
node as per pagerank is: 901
node as per pagerank is: 941
node as per pagerank is: 943
node as per pagerank is: 944
node as per pagerank is: 982
node as per pagerank is: 995
node as per hub is: 524
node as per auth is: 744
Auth score of node 524 3.133017149750832e-06
```

In the above figure I have printed the minimum score's key and found that pagerank and hub score is minimum on page 524 whereas the auth score is not the minimum for the same.

Following shows the nodes for the max score value



```
for key, value in sorted_pr.items():
  if max(sorted_pr.values()) == value:
      print("node as per pagerank is:", key)

for key, value in sorted_hub.items():
  if max(sorted_hub.values()) == value:
      print("node as per hub is:", key)

for key, value in sorted_auth.items():
  if max(sorted_auth.values()) == value:
      print("node as per auth is:", key)

print("Auth score of node 1",sorted_auth[1] )
print("Hub score of node 1",sorted_hub[1] )


print("Pagerank score of node 160",sorted_pr[160] )
```

```
node as per pagerank is: 1
node as per hub is: 160
node as per auth is: 160
Auth score of node 1 2.9004622902639858e-05
Hub score of node 1 0.0017024061489624486
Pagerank score of node 160 0.006758893760759583
```

Also the corresponding values of the node specified by auth/hub