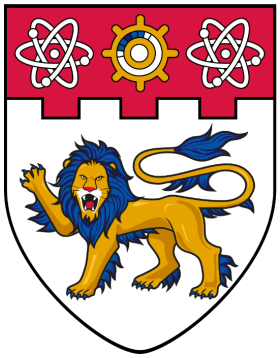


NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE

AY2021-22, Semester 2

CZ4071 Network Science

Project 1

Members	Matriculation Number
Phoe Chuan Bin	U1821679J
Chen Gangzhe	U1822840E
Bose Rajeswari	U1822509G
Kapur Rohan	U1923800A

1. Extracting Network Data

We are given an excel file *DataScientists.xls* and are required to build a collaboration network to understand their temporal history and the network properties. Our data extraction process is robust - it is able to extract data solely based on the input file, without any form of hard coding. The full implementation of this process can be found under *data_collection.ipynb.*

Analysing the input excel file

We observed that every author has a unique URL and a corresponding XML page in the form:

https://dblp.org/pid/**<dblp>**.html
https://dblp.org/pid/**<dblp>**.xml

However, from the input file given, some URLs under the *dblp* column did not follow this format. Instead, they look like:

https://dblp.org/pers/**<first letter of name>**/**<name>**.html

Fortunately when accessing such name-based URLs, we noticed that they were all eventually directed to their respective PID-based URLs. As such, we will exploit this by using the redirected address to get the unique PID to identify each unique author. We will use the Author PID XML for our data scraping process.

Example of data inconsistencies

X	Input file URL	https://dblp.org/pers/e/Elmore:Aaron_J=.html
X	Author PID URL	https://dblp.org/pid/75/9436.html
✓	Author PID XML	https://dblp.org/pid/75/9436.xml

Data Scraping Approach

a. Strategy to traverse data - Iteration
   To scrape all the links given in the input excel file, we first read the data into a DataFrame before iterating and sending a get request to

each URL. From the redirected address, we will obtain the unique PID of the author. Lastly, we will send another GET request to get the author's XML page.

b. Prevent duplicate records
During this process, we kept a PID seen list, *pid_list*. This list is updated whenever a new unique PID is seen and will skip the scraping process if the current PID has already been seen before. This helps us to prevent scraping duplicate author records.

Examples of duplicate PIDs

```
Total link addresses: 1220
Processed link address 1
Processed link address 2
Processed link address 3
Processed link address 4
Processed link address 5
Processed link address 6
Processed link address 7
Processed link address 8
Skipping repeated PID 04/7892
Processed link address 10
Processed link address 11
Processed link address 12
Processed link address 13
Processed link address 14
Processed link address 15
Processed link address 16
Skipping repeated PID k/AlfonsKemper
Processed link address 18
Processed link address 19
Processed link address 20
Processed link address 21
Processed link address 22
Processed link address 23
Processed link address 24
Processed link address 25
Skipping repeated PID h/AlonYHalevy
Processed link address 27
Processed link address 28
Processed link address 29
Processed link address 30
Processed link address 31
Skipping repeated PID 136/7882
Processed link address 33
```

c. BeautifulSoup4
From the Author XML page, we used BeautifulSoup 4 to parse the content and extract the details that we wanted.
    1. Author PID

2. Co-author PID
3. Year
4. Title

We then append a new row to the *network_df* DataFrame for each unique author-coauthor-year-title relationship.

d. Error handling
We noticed that some links provided by the input excel file were non-existent, as such we catch these errors in try-except blocks to ensure that the whole process is reliable

Example of invalid PID

```
Processed link address 365
Processed link address 366
Skipping invalid PID 98/5721 . No such person record.
Processed link address 368
Processed link address 369
Processed link address 370
```

e. Saving scraped data
We saved this data as *RawNetworkDataFrame.csv*

Pseudocode of data scraping implementation

For each http address in the dblp column of *DataScientists.xls*:

1. Get the PID from the redirected address
2. Skip if seen this PID before (prevent duplicates), else update seen list (pid_list) and name mapping (pid_name_mapping.
3. Use bs4 to extract XML details
4. Skip if this PID is non-existent (prevent error if no such records)
5. Get all articles and extract ['author_pid','coauthor_pid','year','title']
6. Add to DataFrame network_df

Data Cleaning Approach

a. Remove unwanted nodes
   Since it was stated that "the network should only contain individuals in the input file as nodes. No other individual should be part of this network." In the assignment, we removed rows where the PIDs were not present in the *pid_list* which we extracted earlier. To recall, this list contains the unique PID of all authors in the input excel file. Our DataFrame was reduced from the original 362952 rows to 64474 rows.

b. Mapping PIDs to Author Names
   To make our data more user-friendly, we mapped each unique PID to the corresponding Author Name. This dictionary was created whilst traversing each Author's XML page during the data scraping process.

c. Check for duplicates arising from bidirectional relationships
   Since author-coauthor relationships are bidirectional, our data will have multiple entries of data where the title and year are the same with the author and co-author values swapped.

   Row `<authorA>,<authorB>,<title1>,<year1>`
   should be treated the same as
   Row `<authorB>,<authorA>,<title1>,<year1>`

   As such, we created a neat function to check for duplicates by converting each row's values into a string *check_duplicates* and then compare the sorted author-coauthor pairing. Rows whose strings are found to be duplicates will be removed.

   `<sorted[author_pid,coauthor_pid]><title><year>`.

Example of duplicates arising from bi-directional relationships

```
# Check for duplicates
network_df['check_duplicates'].sort_values(ascending=True)
```

```
225853    00/110705/1478Secure Personal Data Servers: a ...
304969    00/110705/1478Secure Personal Data Servers: a ...
136310    00/110773/6460A Scalable Search Engine for Mas...
225799    00/110773/6460A Scalable Search Engine for Mas...
136318    00/110773/6460A Secure Search Engine for the P...
                                ...
19736     z/AoyingZhouz/XiaofangZhouEffective Data Densi...
20177     z/AoyingZhouz/XiaofangZhouFiltering Duplicate ...
333133    z/AoyingZhouz/XiaofangZhouFiltering Duplicate ...
19531     z/AoyingZhouz/XiaofangZhouQuality-aware schedu...
332474    z/AoyingZhouz/XiaofangZhouQuality-aware schedu...
Name: check_duplicates, Length: 64474, dtype: object
```

Our DataFrame was reduced to 31030 rows.

d. Saving processed data
   We saved this data as *ProcessedNetworkDataFrame.csv*

2. Assignment Questions

In order to answer questions 1 to 3, we have created a web application that reads the *ProcessedNetworkDataFrame.csv* file to generate the corresponding network, so the network only contains individuals from the input file as nodes. The *Networkx* library was used to generate the network and its properties.

To start the web application, change directory to the *Input* folder and run *py project.py*. More information is provided in the read-me file. Please note that as the network is large, the web application may take a long time to respond. Hence, we have recorded a short video that demonstrates how the web application works here.

Question 1

We have chosen to display the following properties of the collaboration network, using the corresponding *Networkx* functions or derived otherwise:

Property	***Networkx* Function**
Number of isolates	number_of_isolates()
Number of nodes	number_of_nodes()
Number of edges	number_of_edges()
Average degree	number_of_nodes() ÷ number_of_edges()
Highest degree	sorted(G.degree, key=lambda x: x[1], reverse=True)[0][1], where G is the network
Average Clustering Coefficient	average_clustering()
Number of nodes in largest connected component	Obtain the largest connected component using connected_components(), then use number_of_nodes()
Diameter of largest connected component	diameter()
Average shortest path of largest connected component	average_shortest_path_length()
Node with highest degree centrality	degree_centrality()
Node with highest eigenvector centrality	eigenvector_centrality()
Node with highest betweenness centrality	betweenness_centrality()
Node with highest closeness centrality	closeness_centrality()

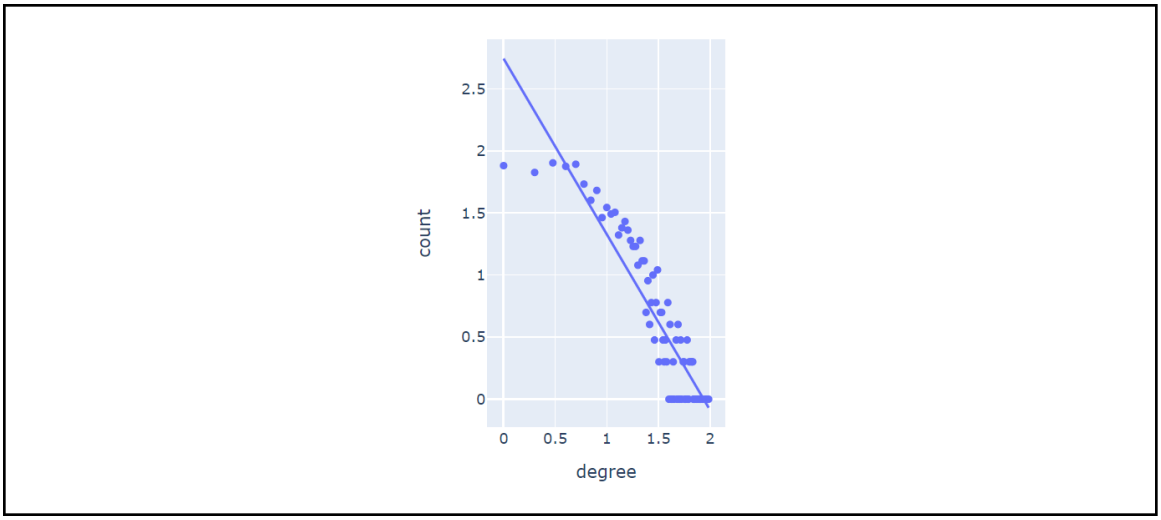
Example of network properties

Network Properties

Properties	Results
Number of isolates	0
Number of nodes	989
Number of edges	6447
Average degree	6.518705763397371
Highest degree	97
Average Clustering Coefficient	0.3263719949949877
Number of nodes in largest connected component	983
Diameter of largest connected component	9
Average shortest path of largest connected component	3.4075453793926487
Node with highest degree centrality	z/XiaofangZhou
Node with highest eigenvector centrality	z/XiaofangZhou
Node with highest betweenness centrality	z/XiaofangZhou
Node with highest closeness centrality	z/XiaofangZhou

In addition, we have also included the log-log degree distribution of the network by drawing a scatter-plot using the *Plotly* library.

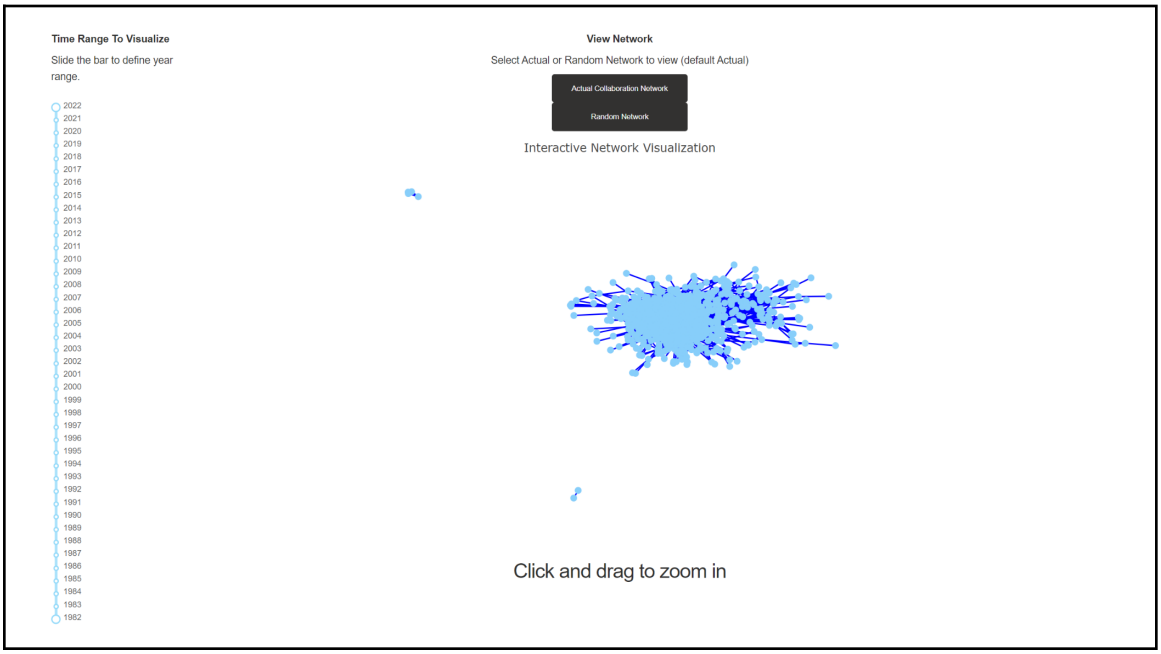
Example of degree distribution plot

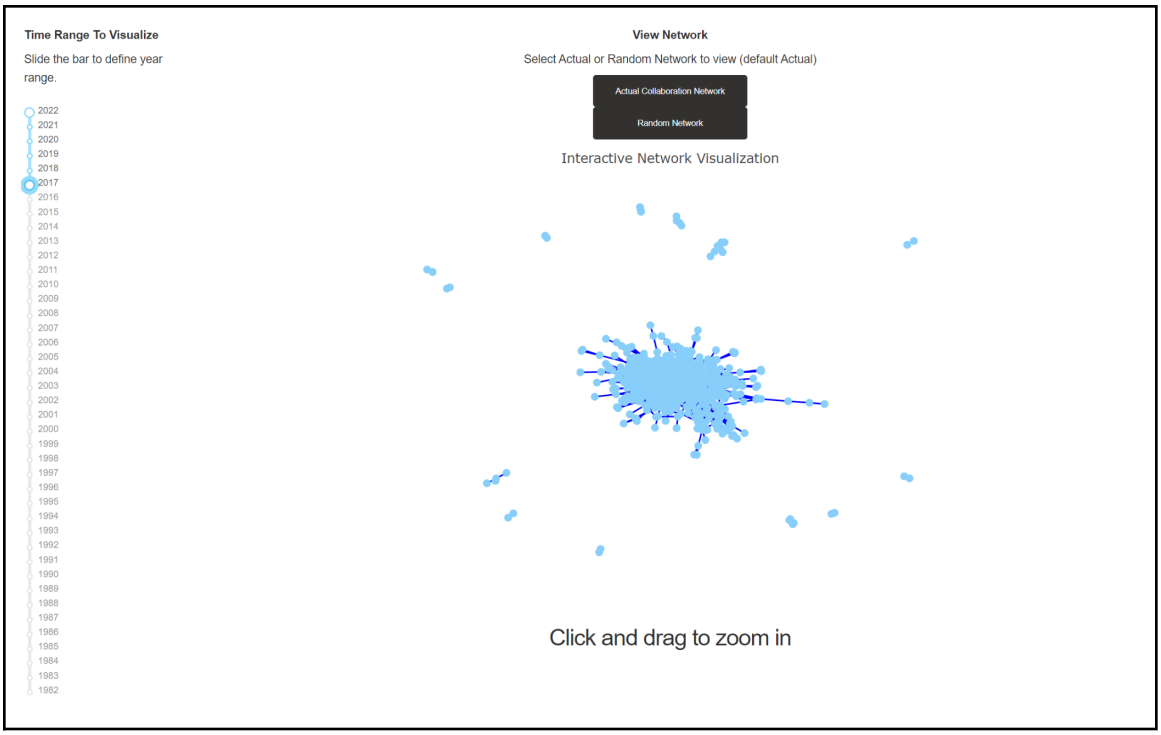


Question 2

To better visualise how the collaboration network and its properties evolved over time, we have included a year slider in our web application. After adjusting the range of years, the network updates automatically, as well as its properties and log-log degree distribution, which details are described in Question 1.

Example of using the year slider



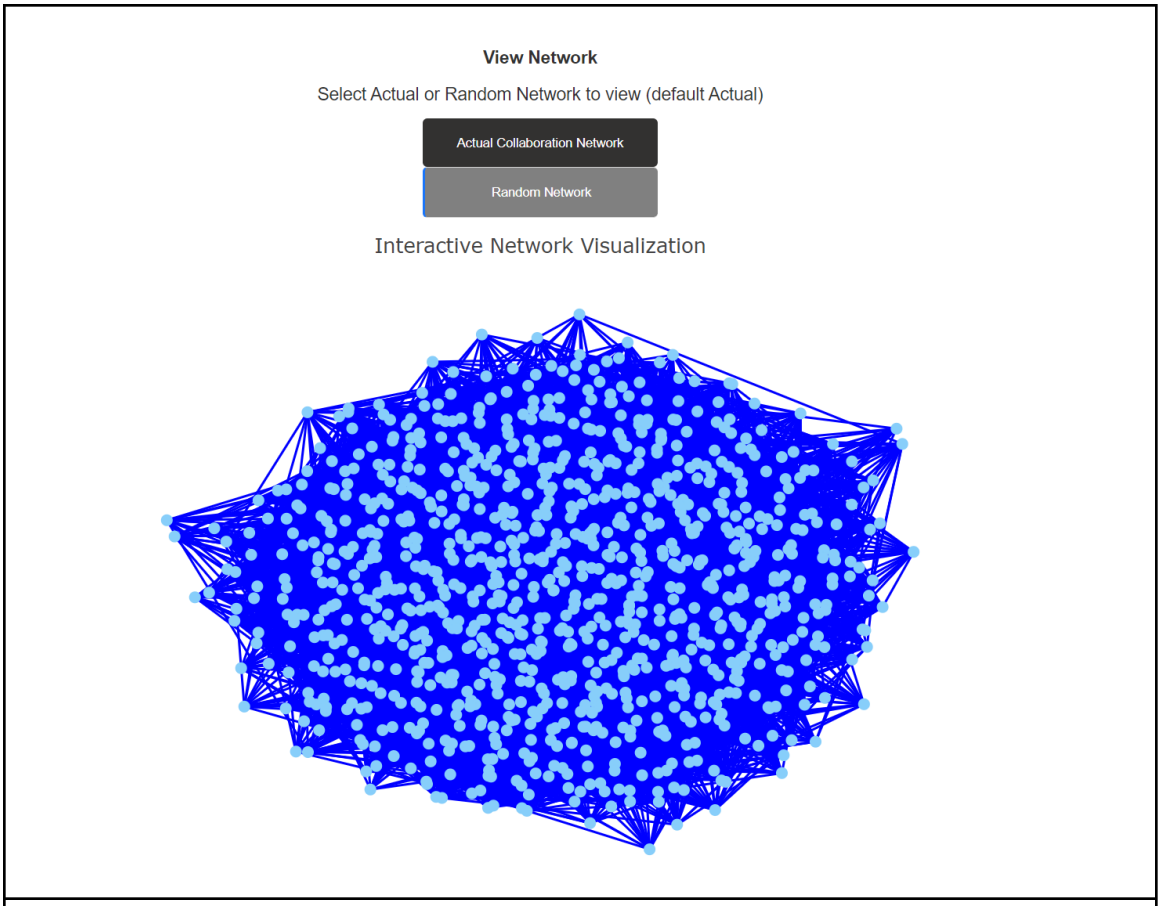


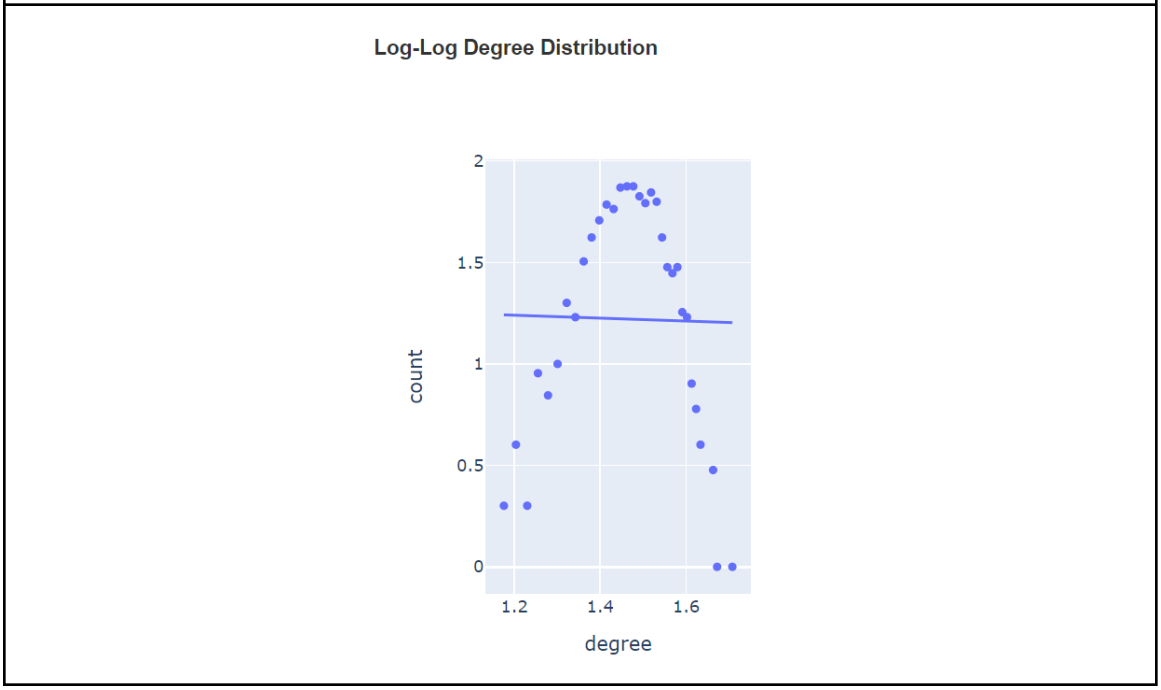
Question 3

As the input file that we are using has a total of 989 individuals, we have created an $G(n, p)$ random graph (or Erdős-Rényi graph) using the *Networkx* function *fast_gnp_random_graph()* with $n = 989$, $p = 0.03$ (an arbitrary value that we have decided).

In order to better visualise the random network and how its properties differ from the real collaboration network, we have included a toggle in our web application to switch between the actual and random networks. For the random network, the year slider will be irrelevant.

Random network and its properties





Network Properties

Properties	Results
Number of isolates	0
Number of nodes	989
Number of edges	14847
Average degree	15.012133468149646
Highest degree	51
Average Clustering Coefficient	0.029684060078913596
Number of nodes in largest connected component	989
Diameter of largest connected component	3
Average shortest path of largest connected component	2.359517444930675
Node with highest degree centrality	988
Node with highest eigenvector centrality	988
Node with highest betweenness centrality	988
Node with highest closeness centrality	988

We can clearly see that the random network's degree distribution follows that of a binomial or poisson distribution, while the actual collaboration network's degree distribution (example in question 1) follows the power law.

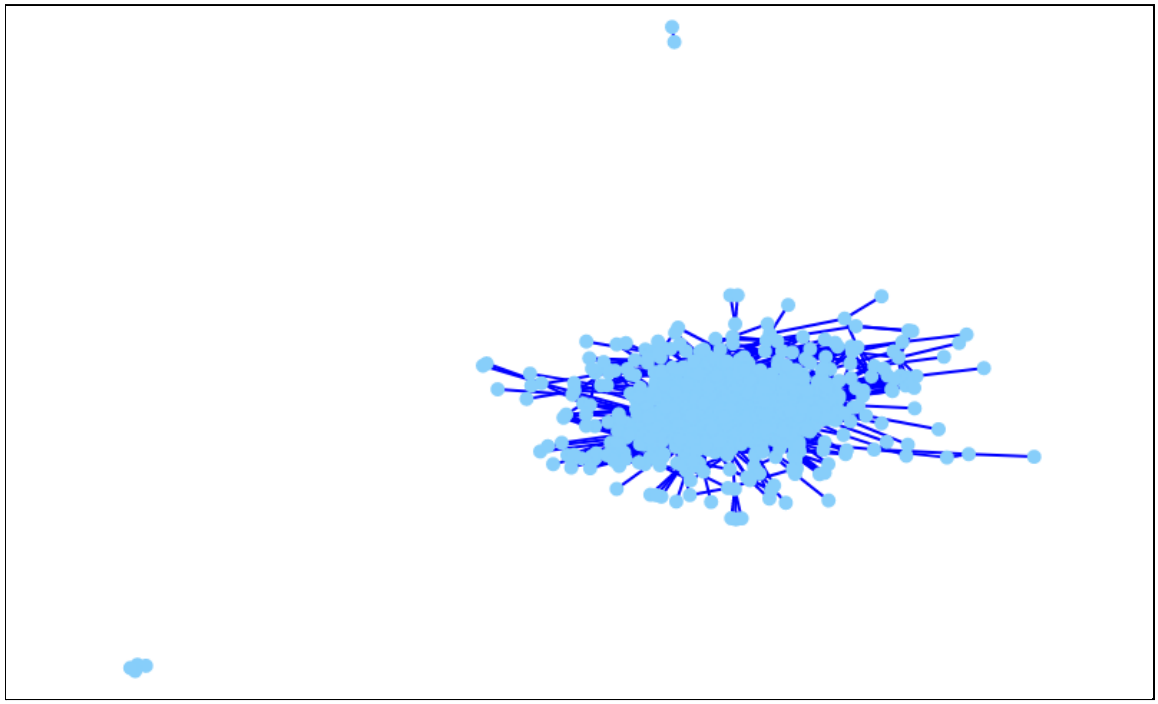
Additionally, the average clustering coefficient of the random network is much lower than the actual collaboration network. However, the distance (average shortest path of the largest connected component) is similar for both networks.

Question 4

The first thing we do for this special question is to load our cleaned and processed dataset, *ProcessedNetworkDataframe.csv*. The schema looks like this:

	author_pid	coauthor_pid	year	title	author_name	coauthor_name
0	75/9436	163/0545	2021	Decomposed Bounded Floats for Fast Compression...	Aaron J. Elmore	John Paparrizos
1	75/9436	163/0545	2021	VergeDB: A Database for IoT Analytics on Edge ...	Aaron J. Elmore	John Paparrizos
2	75/9436	f/MJFranklin	2021	VergeDB: A Database for IoT Analytics on Edge ...	Aaron J. Elmore	Michael J. Franklin
3	75/9436	147/1189	2021	VergeDB: A Database for IoT Analytics on Edge ...	Aaron J. Elmore	Sanjay Krishnan
4	75/9436	117/3757	2021	Version Reconciliation for Collaborative Datab...	Aaron J. Elmore	Zechao Shang
...	...	...	...	...	...	...
31025	189/2405	20/3716	2019	Balance-Aware Distributed String Similarity-Ba...	Zeyuan Shang	Zhifeng Bao
31026	189/2405	20/3716	2018	DITA: Distributed In-Memory Trajectory Analytics.	Zeyuan Shang	Zhifeng Bao
31027	189/2405	20/3716	2018	DITA: A Distributed In-Memory Trajectory Analy...	Zeyuan Shang	Zhifeng Bao
31028	189/2405	20/3716	2017	Dima: A Distributed In-Memory Similarity-Based...	Zeyuan Shang	Zhifeng Bao
31029	169/3426	169/3419	2015	SINGA: A Distributed Deep Learning Platform.	Zhaojing Luo	Zhongle Xie

The saved_attrs.pkl file is a dictionary created beforehand using DataScientists.xls and contains the institute,country, and expertise of each pid. This is used to provide node attributes for each node in the network. Then, using the NetworkX library in python, we plot the original collaboration network, to set as a benchmark:



Note: This is where we add attributes to the graph, based on the country, institution and expertise(randomised) information provided to us in DataScientists.xls as the pickled file saved_attrs.pkl

Here we can see that the network is quite heavily clustered, due to a large giant component, and the influence of certain hubs. As per the requirements of the task, our goal is to increase the number of isolates and introduce diversity into our network.

Currently, the graph has the following metadata:

Number of Isolates	0
Number of connected components	3
Size of giant component	983
Average degree	6.5187

As we can see, there are no isolates and the average degree is quite high, therefore we implement an algorithm to ensure that we have more isolates, increased diversity and a lower average degree.

The first step we take is calculating the diversity of a connected component in the graph. We implement a function in our program called *calculate_diversity.*

Pseudocode of calculate_diversity function:

1. For each node in the connected component:
    a. Add node[country] to list ‘countries’
    b. Add node[institute] to list ‘institutes’
    c. Add node[expertise] to list ‘expertise’
2. Diversity = (no. of unique countries/no. of nodes)*(no. of unique institutes/no. of nodes)*(no. of unique expertise/no. of nodes)

The basic premise behind this is that a connected component has a greater diversity per node if there are a greater number of unique countries, institutes and expertise values.

The next step in our algorithmic process is to go through the network bridge by bridge, removing each bridge and checking if the diversity of our network increases.
The way we do this is in the following steps:

1. **Select bridge** in G
2. **Calculate diversity** of the connected component bridge belongs to
3. **Remove** bridge
4. **Calculate new diversity** of each connected component
5. If new diversity <= old diversity, then **add the bridge back**.

Hence, we remove all those bridges that do not decrease the diversity of our network with respect to the three metrics, rather than keeping only those bridges that increase diversity.

Further use of diversity and its implementation in the algorithm is as follows:
For the transformation, there are two things that are implemented in order of decreasing priority while removing edges in a judicious manner (in order of priority):

1. Nodes with common country, institution or expertise (diversity metrics)
2. Decreasing order of the other node's degree

The explanation for 1. Is quite straightforward. The heuristic can be described as follows:

- We remove nodes that are similar to one another. For this, we use a simple variable called *neighbour_difference* in our algorithm.

- The *neighbour_difference* variable is calculated by comparing the characteristics of a node to its neighbour. If any attribute amongst country, expertise or institute are not the same, *neighbour_difference* is incremented by 1 for each difference.

- We then store the indices of neighbouring nodes with zero differences, one difference and two differences in variables *zero_diff, one_diff* and *two_diff* respectively.

- After we are done with the segregation, we start to remove the edges from the least difference, i.e the ones in *zero_diff.*

- Now, we check if the degree of the node is less than k_{max} . If it is, we stop for that node and move to the next, if not, we continue.

Collaboration Cutoff: In this blurb, we will be explaining what the value of collaboration cutoff for our experiment was, and how we arrived at it. For the calculation of k_{max}, we considered the average degrees or **<k>**. This was simply calculated in the following way:

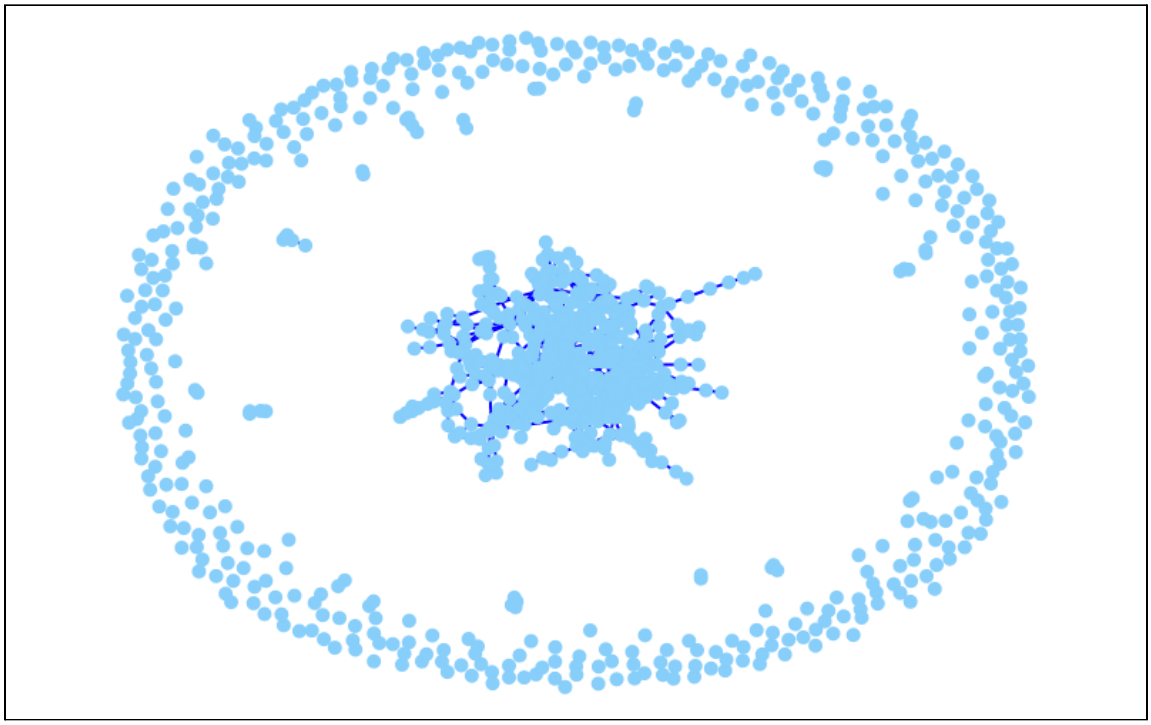
$$\mathbf{k_{max} = <k> = no.of\ edges/no.\ of\ nodes}$$

For the network given to us, this value stood at **6.437.**

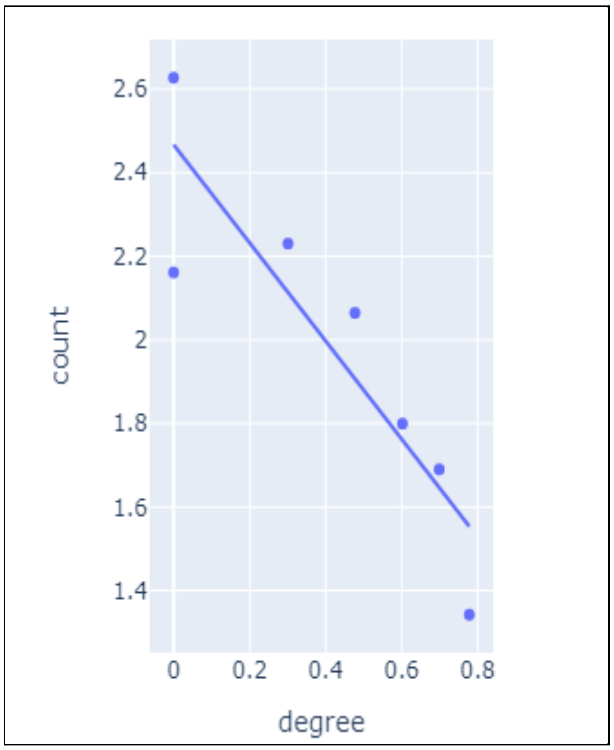
- We repeat these steps for *one_diff* and *two_diff*, only going to the next set of nodes if we still have the degree of the node to be less than k_{max}

- After exhausting these variables, we now start to remove edges in decreasing order of the neighbour degrees, until we find that the maximum degree of the node does not exceed k_{max}. *Note: This is done after sorting the nodes in decreasing order of their degrees, and popping the top element from the stack.*

Post the implementation of this sophisticated algorithm, we plot the network graph once more. We get a figure like this:



Log-Log Degree Distribution Plot



Note that we can already see the pictorial difference in how sparse the graph is, with a clear difference in the number of influential nodes in the graph itself.

But now for the testing of whether we actually achieve the objective set out (smaller giant component and more isolates). Upon calculation for the transformed network, we get the following result (in comparison to original network as well):

Metric	**Original network**	**Transformed network**
Number of Isolates	**0**	**424**
Number of connected components	**3**	**448**
Size of giant component	**983**	**505**
Average degree	**6.5187**	**0.739**

We can draw the following conclusions to prove the success of our experiment:

- The **number of isolates** has gone up from **0** to **424**, increasing significantly
- The size of **giant component** has reduced by **49%,** down to almost half of the original value
- The **average degree** is **1/9** of the original network

Therefore, we are successful in reducing the influence of a few powerful nodes in the network. We have accomplished the following objectives:

- A smaller giant component and larger number of isolates than the original collaboration network
- The maximum degree of any node does not exceed beyond a user-specified kmax, referred to as collaboration cutoff, which is typically smaller than the degrees of hubs.

- A rich diversity of individuals (country, expertise, institutions) in the transformed network.

Conclusion

From the findings above, we can see that each network exhibits different network properties which have been summarised. Despite the number of nodes (N) being the same, there are a large variety of networks that can be constructed. For instance, the random network constructed showed a much higher average degree (15) as compared to the original collaboration network (6). In this particular example of data scientists, a simple network reconstruction has shown how it is possible to change the structure to a decentralised system. This is also useful for studying the behaviour and modelling ideal collaboration networks. However, implementing such networks in practice might prove to be a much more difficult task.