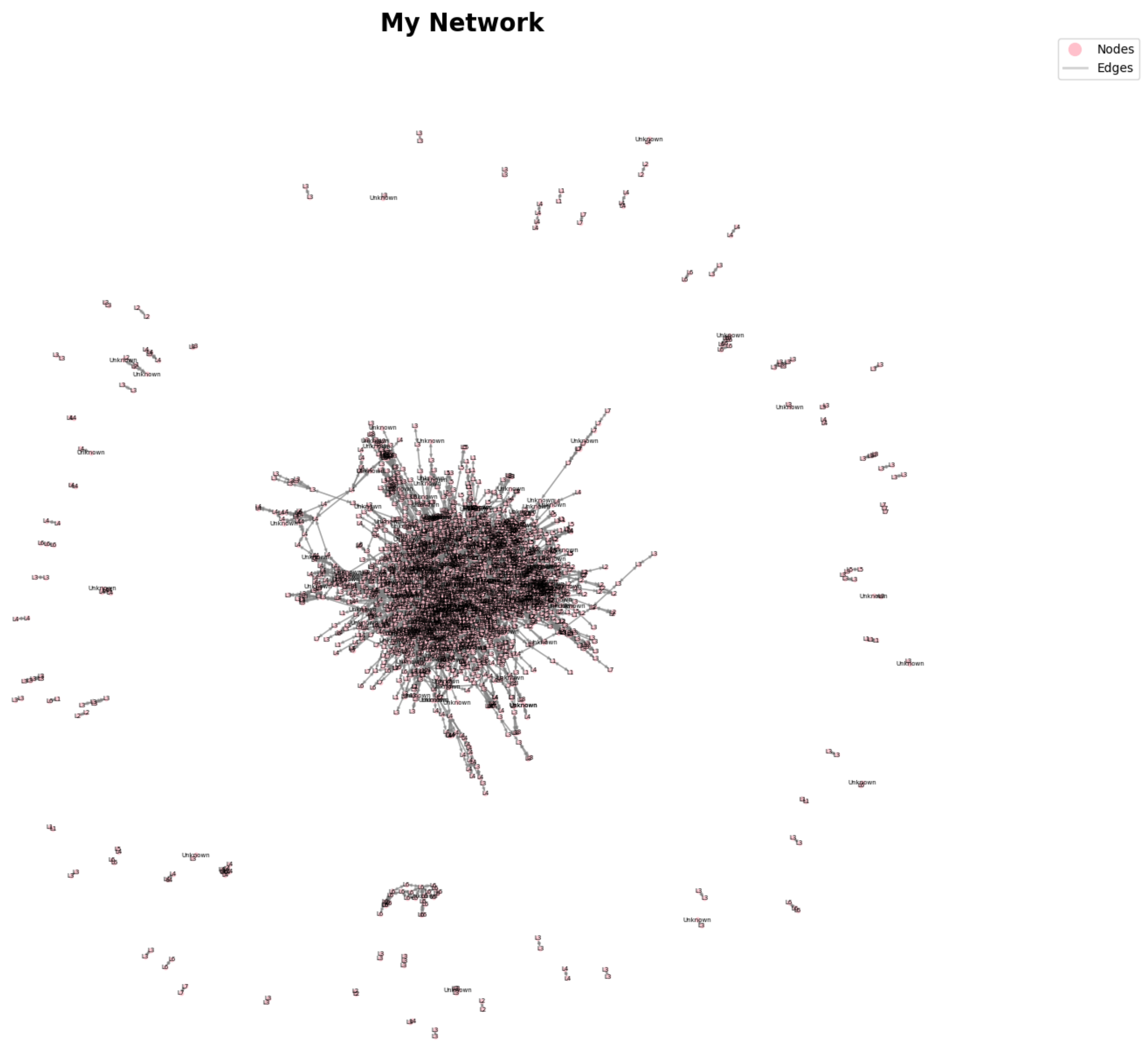
Step 1:

A. plotting the Graph:

reads in data from two Excel files, creates a directed graph using NetworkX, creates an empty directed graph using the NetworkX library then adds the nodes to the graph. Each node ID is paired with a dictionary containing a `label` key and the node label as the value then adds the edges to the graph. The resulting plot shows the nodes as pink circles and the edges as gray lines, with labels for each node. A legend is included to show the meaning of the colors and markers used in the plot. visualizes the graph using matplotlib.

Plot the directed graph with their Label:



B. Statistical Metrics:

Printing some important statistical Features:

Our statistics are:
Total Degree: 21112
Total Edges: 10556
Average Degree: 7.796159527326441
Average shortest path: 1.170483515230952
Transitivity: 0.09349725626661058
Clustering Coefficient: 0.24067329850193744
Density: 0.0014399999126942077
Diameter: 19
Assortativity: -0.06587087427227857

Note: there is no path between some pairs of nodes in the graph and The average_shortest_path_length() function from networkx can only be used on strongly connected graphs. One way to handle this situation is to use the weakly_connected_components() function from networkx to decompose the graph into its weakly connected components, and then compute the average shortest path length for each component separately.

We first calculate the weakly connected components of the graph using the weakly_connected_components() function. If there is only one component, we can simply compute the average shortest path length using the average_shortest_path_length() function. If there are multiple components, we compute the average shortest path length for each component using a loop, and then take the average of these values.

The total degree of the network is 21112, which indicates that the network is relatively large. **The total number of edges in the network is 10556**, which suggests that the network is not really dense compared to dense networks.

The transitivity of the network is 0.0935, which indicates a relatively **low level of clustering**. This means that nodes in the network are less likely to form **triangles** or clusters compared to a network with higher transitivity.

The clustering coefficient of the network is 0.2407, which is relatively high and suggests that the **degree of clustering within the network is relatively strong.** This means that nodes in the network tend to form triangles or clusters more frequently than in a network with lower clustering coefficient.

The density of the network is 0.0014, which indicates that the network is sparse. This can be explained by the relatively large size of the network compared to the total number of edges.

The diameter of the network is 19,which is really high compared to real NetWorks. Maybe it is not a social network of people.

The average degree of the network is 7.7962, which indicates that the average number of edges per node is relatively low. **This suggests that the network is not heavily interconnected.**

The average shortest path of the network is 1.1705, which suggests that the network is relatively well-connected. This means that information can be transmitted quickly and efficiently within the network.

The assortativity of the network is -0.0659, which indicates that nodes in the network tend to be connected to nodes with different degrees. This means that high-degree nodes are less likely to be connected to other high-degree nodes, and low-degree nodes are less likely to be connected to other low-degree nodes.

Conclusion: the network appears to have a relatively low level of clustering, but a high degree of clustering coefficient. The network is relatively well-connected, but not heavily interconnected. The assortativity suggests that nodes tend to be connected to nodes with different degrees, which may have implications for information flow and **network resilience.**

C. Centrality Measures:

Degree Centrality:
The five nodes with the highest degree centrality in this network are nodes 35, 6213, 1365, 3229, and 910. Node 35 has the highest degree centrality with a value of 0.12, which indicates that **it is an important hub in the network**. Nodes with high degree centrality tend to have a large number of connections to other nodes in the network, and can play **important roles in facilitating the flow of information or resources.**

Closeness Centrality:
The five nodes with the highest closeness centrality in this network are nodes 35, 6213, 3229, 887, and 4584. These nodes all have the same closeness centrality value of 0.22, which indicates that they are all equally important in terms of transmitting information quickly throughout the network. Nodes with high closeness centrality tend to be located close to other nodes in the network, and can play important roles in facilitating the flow of information or resources.

Betweenness Centrality:
The five nodes with the highest betweenness centrality in this network are nodes 35, 3229, 4330, 1365, and 6213. Node 35 has the highest betweenness centrality value of 0.23, which indicates that it is an important bridge between different parts of the network. Nodes with high betweenness centrality tend to act as intermediaries between different groups of nodes in the network, and can play important roles in facilitating the flow of information or resources.

The top five nodes by degree centrality, closeness centrality, and betweenness centrality are all different, which suggests that different nodes play important roles in different aspects of the network. Node 35 appears to be particularly important in all three measures of centrality, which indicates that it is a key node in the network and may have a significant impact on the network's structure and function.

It is common to have a possessive correlation between centrality measures of specific nodes but we got a negative correlation, It is possibly an interesting pattern.
Here are the 100 lowest nodes in centrality measures:

Degree	Betweenness	Closeness
(1128945, 0.0007388252678241596),	(72805, 0.0003694126339120798),	(1131420, 0.0),
(1129021, 0.0007388252678241596),	(73972, 0.0003694126339120798),	(1131421, 0.0),
(1129040, 0.0007388252678241596),	(114966, 0.0003694126339120798),	(1131466, 0.0),
(1129106, 0.0007388252678241596),	(115188, 0.0003694126339120798),	(1131550, 0.0),
(1129368, 0.0007388252678241596),	(116512, 0.0003694126339120798),	(1131607, 0.0),
(1129369, 0.0007388252678241596),	(121792, 0.0003694126339120798),	(1131734, 0.0),
(1129443, 0.0007388252678241596),	(133628, 0.0003694126339120798),	(1131741, 0.0),
(1129494, 0.0007388252678241596),	(155277, 0.0003694126339120798),	(1131745, 0.0),
(1129798, 0.0007388252678241596),	(180301, 0.0003694126339120798),	(1131752, 0.0),
(1130069, 0.0007388252678241596),	(187260, 0.0003694126339120798),	(1131754, 0.0),
(1130080, 0.0007388252678241596),	(219218, 0.0003694126339120798),	(1131828, 0.0),
(1130243, 0.0007388252678241596),	(228990, 0.0003694126339120798),	(1132073, 0.0),
(1130356, 0.0007388252678241596),	(228992, 0.0003694126339120798),	(1132285, 0.0),
(1130568, 0.0007388252678241596),	(232860, 0.0003694126339120798),	(1132385, 0.0),
(1130586, 0.0007388252678241596),	(245955, 0.0003694126339120798),	(1132406, 0.0),
(1130637, 0.0007388252678241596),	(294145, 0.0003694126339120798),	(1132416, 0.0),
(1130653, 0.0007388252678241596),	(302545, 0.0003694126339120798),	(1132857, 0.0),
(1130657, 0.0007388252678241596),	(321004, 0.0003694126339120798),	(1132864, 0.0),
(1130929, 0.0007388252678241596),	(368657, 0.0003694126339120798),	(1132887, 0.0),
(1130934, 0.0007388252678241596),	(376704, 0.0003694126339120798),	(1132922, 0.0),
(1131163, 0.0007388252678241596),	(380341, 0.0003694126339120798),	(1133008, 0.0),
(1131164, 0.0007388252678241596),	(384428, 0.0003694126339120798),	(1133028, 0.0),
(1131165, 0.0007388252678241596),	(400455, 0.0003694126339120798),	(1133047, 0.0),
(1131172, 0.0007388252678241596),	(408885, 0.0003694126339120798),	(1133338, 0.0),
(1131180, 0.0007388252678241596),	(430574, 0.0003694126339120798),	(1133390, 0.0),
(1131189, 0.0007388252678241596),	(458439, 0.0003694126339120798),	(1133428, 0.0),
(1131192, 0.0007388252678241596),	(529165, 0.0003694126339120798),	(1133469, 0.0),
(1131195, 0.0007388252678241596),	(594119, 0.0003694126339120798),	(1133846, 0.0),
(1131230, 0.0007388252678241596),	(594483, 0.0003694126339120798),	(1133930, 0.0),
(1131266, 0.0007388252678241596),	(604073, 0.0003694126339120798),	(1134031, 0.0),
(1131330, 0.0007388252678241596),	(610529, 0.0003694126339120798),	(1134056, 0.0),
(1131345, 0.0007388252678241596),	(617378, 0.0003694126339120798),	(1134197, 0.0),
(1131348, 0.0007388252678241596),	(617575, 0.0003694126339120798),	(1134346, 0.0),
(1131414, 0.0007388252678241596),	(621555, 0.0003694126339120798),	(1135108, 0.0),
(1131734, 0.0007388252678241596),	(628458, 0.0003694126339120798),	(1135345, 0.0),
(1131754, 0.0007388252678241596),	(628459, 0.0003694126339120798),	(1135368, 0.0),

(1131828, 0.0007388252678241596),	(633030, 0.0003694126339120798),	(1135455, 0.0),
(1132073, 0.0007388252678241596),	(633031, 0.0003694126339120798),	(1135899, 0.0),
(1132285, 0.0007388252678241596),	(642593, 0.0003694126339120798),	(1135955, 0.0),
(1132406, 0.0007388252678241596),	(646412, 0.0003694126339120798),	(1136040, 0.0),
(1132416, 0.0007388252678241596),	(648369, 0.0003694126339120798),	(1136110, 0.0),
(1132864, 0.0007388252678241596),	(653628, 0.0003694126339120798),	(1136310, 0.0),
(1132887, 0.0007388252678241596),	(654519, 0.0003694126339120798),	(1136342, 0.0),
(1132922, 0.0007388252678241596),	(662250, 0.0003694126339120798),	(1136442, 0.0),
(1133008, 0.0007388252678241596),	(688824, 0.0003694126339120798),	(1136631, 0.0),
(1133028, 0.0007388252678241596),	(709518, 0.0003694126339120798),	(1136634, 0.0),
(1133047, 0.0007388252678241596),	(711994, 0.0003694126339120798),	(1137140, 0.0),
(1133390, 0.0007388252678241596),	(752684, 0.0003694126339120798),	(1137466, 0.0),
(1133469, 0.0007388252678241596),	(754594, 0.0003694126339120798),	(1138027, 0.0),
(1133930, 0.0007388252678241596),	(779960, 0.0003694126339120798),	(1138091, 0.0),
(1134031, 0.0007388252678241596),	(814836, 0.0003694126339120798),	(1138619, 0.0),
(1134056, 0.0007388252678241596),	(817774, 0.0003694126339120798),	(1138755, 0.0),
(1134197, 0.0007388252678241596),	(820661, 0.0003694126339120798),	(1138970, 0.0),
(1135345, 0.0007388252678241596),	(824245, 0.0003694126339120798),	(1139009, 0.0),
(1135368, 0.0007388252678241596),	(899085, 0.0003694126339120798),	(1139195, 0.0),
(1135455, 0.0007388252678241596),	(1102548, 0.0003694126339120798),	(1140230, 0.0),
(1135955, 0.0007388252678241596),	(1104191, 0.0003694126339120798),	(1140231, 0.0),
(1136040, 0.0007388252678241596),	(1104435, 0.0003694126339120798),	(1140547, 0.0),
(1136110, 0.0007388252678241596),	(1104749, 0.0003694126339120798),	(1140548, 0.0),
(1136310, 0.0007388252678241596),	(1105622, 0.0003694126339120798),	(1152179, 0.0),
(1136342, 0.0007388252678241596),	(1107728, 0.0003694126339120798),	(1152379, 0.0),
(1136442, 0.0007388252678241596),	(1107808, 0.0003694126339120798),	(1152394, 0.0),
(1136634, 0.0007388252678241596),	(1108570, 0.0003694126339120798),	(1152436, 0.0),
(1137140, 0.0007388252678241596),	(1110628, 0.0003694126339120798),	(1152508, 0.0),
(1137466, 0.0007388252678241596),	(1110950, 0.0003694126339120798),	(1152569, 0.0),
(1138091, 0.0007388252678241596),	(1112417, 0.0003694126339120798),	(1152663, 0.0),
(1138619, 0.0007388252678241596),	(1113084, 0.0003694126339120798),	(1152711, 0.0),
(1138755, 0.0007388252678241596),	(1117049, 0.0003694126339120798),	(1152821, 0.0),
(1138970, 0.0007388252678241596),	(1119654, 0.0003694126339120798),	(1152858, 0.0),
(1139009, 0.0007388252678241596),	(1120019, 0.0003694126339120798),	(1152859, 0.0),
(1139195, 0.0007388252678241596),	(1120880, 0.0003694126339120798),	(1152910, 0.0),
(1140231, 0.0007388252678241596),	(1123493, 0.0003694126339120798),	(1152917, 0.0),
(1140548, 0.0007388252678241596),	(1124837, 0.0003694126339120798),	(1152944, 0.0),
(1152179, 0.0007388252678241596),	(1125258, 0.0003694126339120798),	(1152959, 0.0),
(1152379, 0.0007388252678241596),	(1126315, 0.0003694126339120798),	(1153031, 0.0),
(1152394, 0.0007388252678241596),	(1129021, 0.0003694126339120798),	(1153064, 0.0),
(1152436, 0.0007388252678241596),	(1129040, 0.0003694126339120798),	(1153065, 0.0),
(1152508, 0.0007388252678241596),	(1129494, 0.0003694126339120798),	(1153106, 0.0),
(1152569, 0.0007388252678241596),	(1130069, 0.0003694126339120798),	(1153169, 0.0),
(1152663, 0.0007388252678241596),	(1130080, 0.0003694126339120798),	(1153254, 0.0),
(1152821, 0.0007388252678241596),	(1130243, 0.0003694126339120798),	(1153736, 0.0),
(1152858, 0.0007388252678241596),	(1131163, 0.0003694126339120798),	(1153784, 0.0),
(1152859, 0.0007388252678241596),	(1131195, 0.0003694126339120798),	(1153786, 0.0),
(1152910, 0.0007388252678241596),	(1131330, 0.0003694126339120798),	(1153866, 0.0),
(1152959, 0.0007388252678241596),	(1131414, 0.0003694126339120798),	(1153877, 0.0),
(1153031, 0.0007388252678241596),	(1131754, 0.0003694126339120798),	(1153879, 0.0),

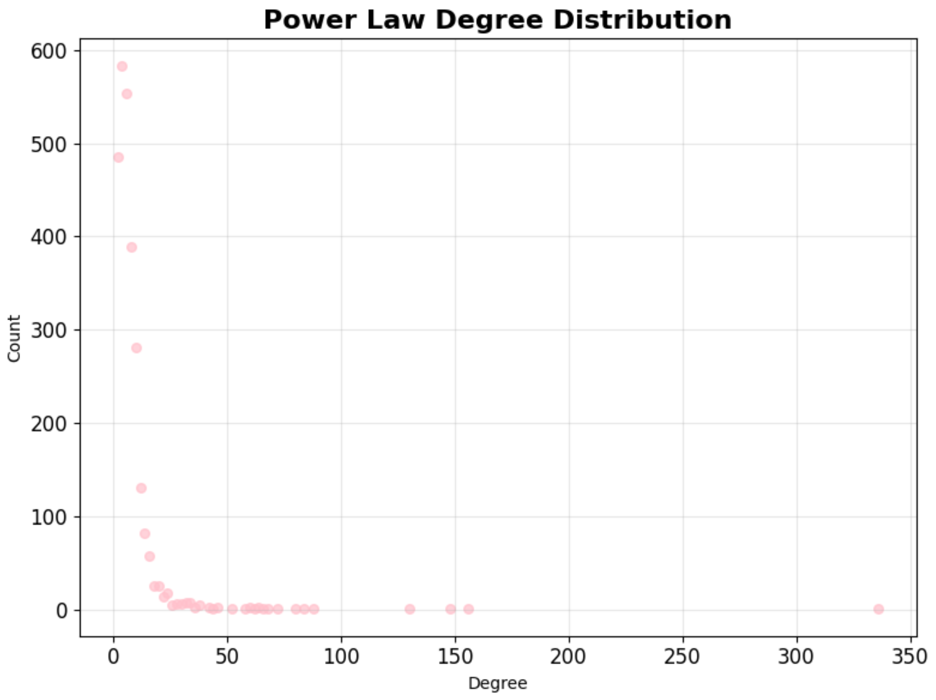
(1153106, 0.0007388252678241596),	(1132073, 0.0003694126339120798),	(1153889, 0.0),
(1153254, 0.0007388252678241596),	(1133008, 0.0003694126339120798),	(1153896, 0.0),
(1153736, 0.0007388252678241596),	(1133930, 0.0003694126339120798),	(1153945, 0.0),
(1153786, 0.0007388252678241596),	(1134056, 0.0003694126339120798),	(1153946, 0.0),
(1153877, 0.0007388252678241596),	(1134197, 0.0003694126339120798),	(1154012, 0.0),
(1153879, 0.0007388252678241596),	(1135955, 0.0003694126339120798),	(1154042, 0.0),
(1153889, 0.0007388252678241596),	(1136040, 0.0003694126339120798),	(1154071, 0.0),
(1153946, 0.0007388252678241596),	(1137140, 0.0003694126339120798),	(1154074, 0.0),
(1154071, 0.0007388252678241596),	(1138619, 0.0003694126339120798),	(1154173, 0.0),
(1154074, 0.0007388252678241596),	(1139009, 0.0003694126339120798),	(1154230, 0.0),
(1154173, 0.0007388252678241596),	(1152394, 0.0003694126339120798),	(1154233, 0.0),
(1154230, 0.0007388252678241596),	(1152858, 0.0003694126339120798),	(1154500, 0.0),
(1154500, 0.0007388252678241596),	(1153946, 0.0003694126339120798),	(1154520, 0.0),
(1154520, 0.0007388252678241596)	(1154173, 0.0003694126339120798)	(1154524, 0.0)

Couldn't find an interesting pattern in the 100 lowest centralized nodes.

Step 2:

Show the most important nodes

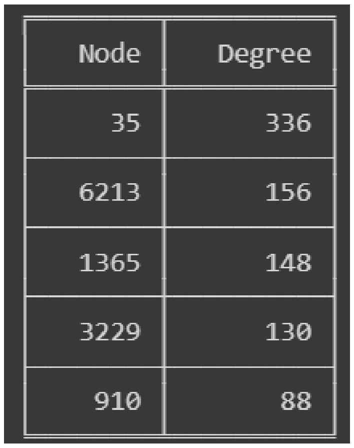
A. Degree Centrality:



Since the degree distribution follows a power law, we know that there are relatively few nodes with very high degrees, and many nodes with low degrees. This implies that the network may have a few highly connected nodes (i.e., "hubs") that are critical to the network's structure and function and according to **“algha= 0.9002920001438559”** this difference is very tense.

In the power law formulation p(x) = C x^{-\alpha}, the parameter α represents the exponent of the power law. It determines the degree of the power-law behavior, with larger values of α indicating a more steeply decaying distribution.
Duo to power law degree distribution our network in resilience to the random attack but not resilience to the intentional one.

These are the most important nodes according to Degree



Node	Degree
35	336
6213	156
1365	148
3229	130
910	88

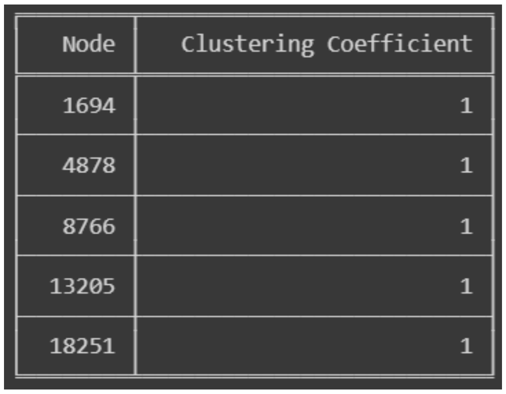
B. Clustering Coefficient:

Nodes with high clustering coefficients are important because they tend to be part of highly connected groups or clusters in the network. The clustering coefficient of a node measures the extent to which its neighbors are also connected to each other. Nodes with high clustering coefficients are often located in tightly knit groups or communities, which can be important for understanding the structure and function of the network.

In social networks, for example, nodes with high clustering coefficients are often part of tightly knit groups of friends or colleagues, which can be important for the spread of information or ideas. In biological networks, nodes with high clustering coefficients are often part of functional modules, such as groups of genes or proteins that work together to carry out a specific biological function.

Nodes with high clustering coefficients **can also be important for identifying bottlenecks** or vulnerabilities in the network. If a node with a high clustering coefficient is removed from the network, it can disrupt the connectivity of the surrounding nodes and potentially fragment the network into smaller disconnected components.

These are the most important nodes according to high Clustering Coefficient



Node	Clustering Coefficient
1694	1
4878	1
8766	1
13205	1
18251	1

C. Betweenness and closeness:

Betweenness centrality and closeness centrality are two important measures of node centrality that can be used to identify nodes that are critical for the flow of information and communication within a network.

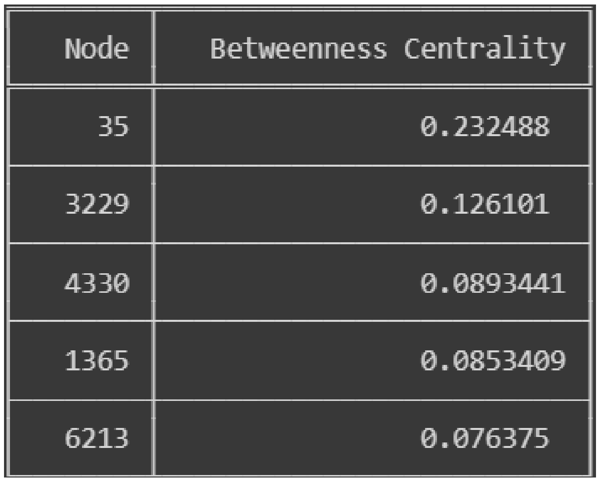
Betweenness centrality measures the extent to which a node lies on the shortest paths between other nodes in the network. Nodes with high betweenness centrality are important for facilitating communication and information transfer between different parts of the network. These nodes act as "bridges" or "hubs" that connect different groups of nodes and play a critical role in the overall structure and function of the network. Removing nodes with high betweenness centrality can have a significant impact on the connectivity and efficiency of the network.

Closeness centrality, on the other hand, measures the distance of a node to all other nodes in the network. Nodes with high closeness centrality are able to communicate more efficiently with other nodes in the network. These nodes act as "central" nodes that are well-connected to many other nodes in the network and are often important for information flow and transfer. Removing nodes with high closeness centrality can also have a significant impact on the connectivity and efficiency of the network.

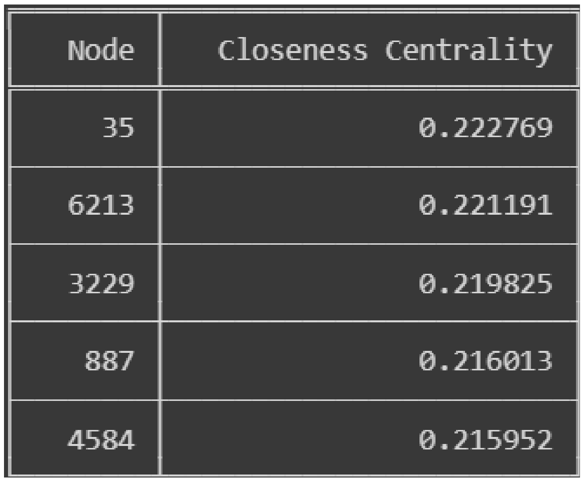
The role of betweenness and closeness centrality in determining node importance will depend on the specific characteristics of the network and the research question being asked. In general, nodes with high betweenness centrality are important for facilitating communication and information.

Note: Generally centrality measures are positively correlated but if we can find the negative correlation of a node in centrality measures, we can assume that some interesting pattern occurred.

These are the most important nodes according to high Betweenness and Closeness



Node	Betweenness Centrality
35	0.232488
3229	0.126101
4330	0.0893441
1365	0.0853409
6213	0.076375

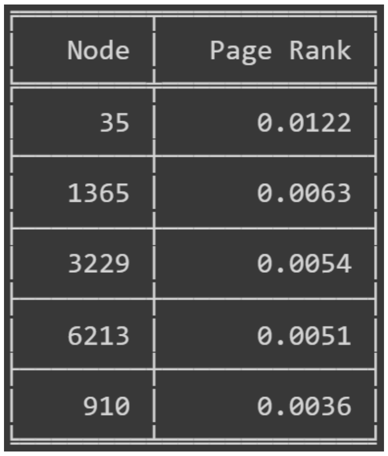


Node	Closeness Centrality
35	0.222769
6213	0.221191
3229	0.219825
887	0.216013
4584	0.215952

D. PageRank:

Page rank algorithm is to compute the centrality scores of the nodes based on random walks. To perform random walks, we start at a given node and randomly move to a neighboring node at each step. We repeat this process for a large number of steps, and the nodes that are visited most frequently are considered to be the most important based on their centrality.

These are the most important nodes according random walk based page rank



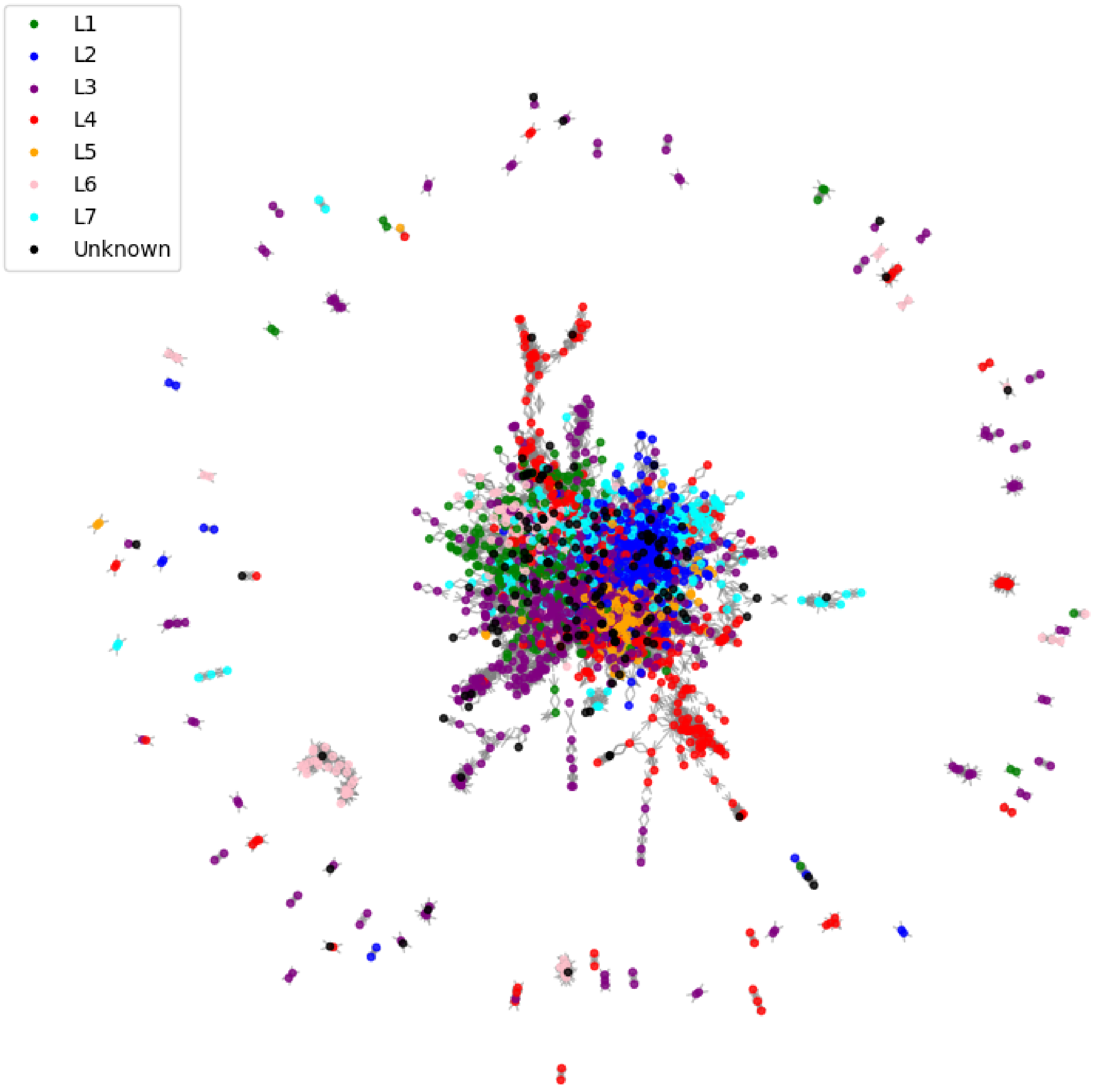
Node	Page Rank
35	0.0122
1365	0.0063
3229	0.0054
6213	0.0051
910	0.0036

Conclusion: As you can see, node 35 is really important in all metrics. We have some common nodes too.

Step 3:

A. Coloring the nodes with their labels:

As you can see, Labels can be used to identify clusters of related nodes, which can in turn be used to define communities within the graph. In this way, the labels serve as indicators of relatedness, and can help uncover underlying structures and patterns in the network.
The label 3 is all over the graph. Almost everywhere.
L6 has an isolated small community.



B.1 Community detection using louvain greedy method:

Girvan newman is not really scalable compared to other community detection methods and louvain is the best known method that has a great speed.

We got the modularity of 0.5708727024501035 using louvain. And could make 710 community

B.2 Community detection using greedy modularity optimization:

There are lots of variations of greedy methods except louvain in community detection. Here is another method which could achieve the **modularity of 0.801645002434549 which is really high and 104 communities.**

C. Is it possible to classify nodes based on identified clusters using community detection methods?

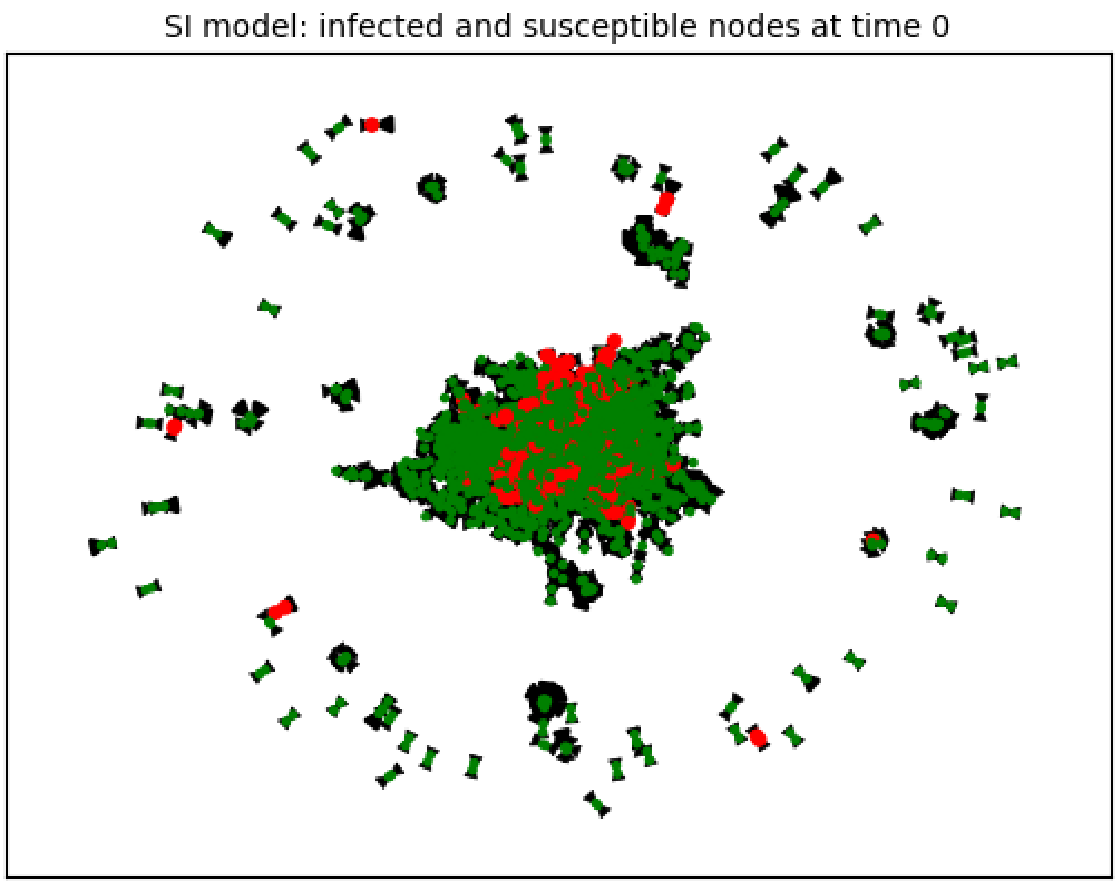
First I put all the nodes with the same label in a specific component. Now we have 8 communities. Then I got a modularity of my community detection based on labels. Modularity of communities **using labels: 0.5928930113933577.** Surprisingly it is near the Louvain modularity. We can assume that these are good large communities but can't really determine that it is really the best partition or not; <u>however we can conclude that using labels as communities is a reasonable approach and may not be significantly inferior to more advanced community detection methods.</u>

Step 4:

A. Using SI Epidemic model to determine most suspicious nodes.

The model divides the population into two compartments: susceptible (S) and infected (I). Individuals in the S compartment are susceptible to the disease, while individuals in the I compartment are infected and can transmit the disease to susceptible individuals. The model assumes that there is no recovery or immunity from the disease, so once an individual is infected, they remain infected and infectious indefinitely.

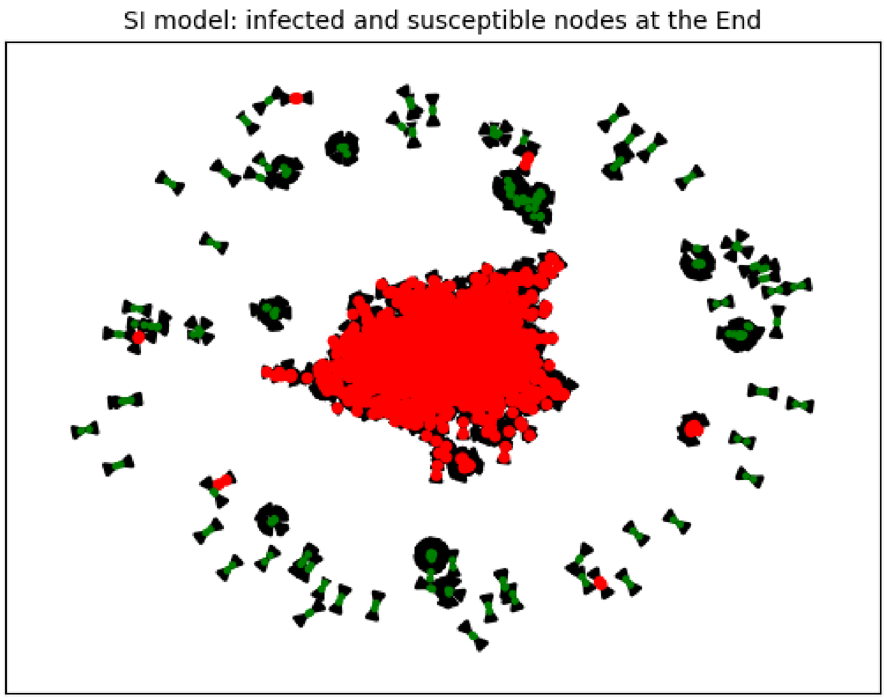
Here is the plot in time 0 which indicate the infected and susceptible nodes at time 0.
The red nodes are the infected nodes and green are the susceptible.



In each time step represented with ”time = []”. We have different numbers and probability of infections.

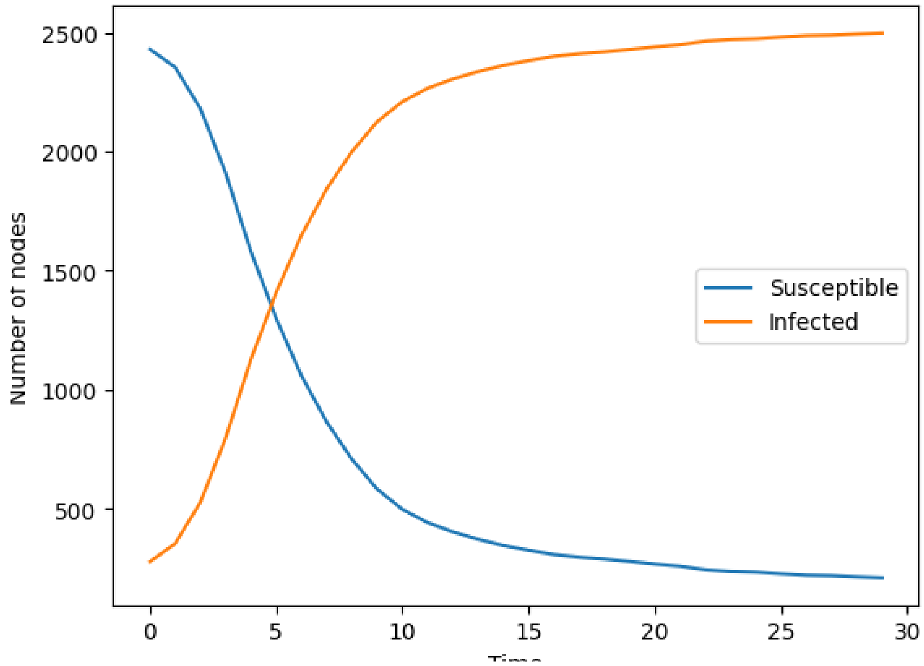
For example here in this plot we have the SI model plot in the last time step.

After this time step we do not have any new infected nodes. The green nodes are kinda isolated so that they couldn't get affected by the virus or whatever.



I didn't have any background of the identification of the nodes and edges, so assume the probability of getting affected of nodes equal to **p=0.3**
According to the project, **the initial_infected_nodes are the nodes with label L1.**

Now we can simulate the rate of S and I over the time



We have almost 29 steps from zero time to the last one until all the red nodes in the previous figure are affected.

To determine the people with the highest probability of being diagnosed with this disease among the Unknown Labeled nodes we must run our SI model in different simulations and make an average of these simulations.

For Example:
Here is the Unknown label nodes that fell sick in the **first simulation** in each step:

```
1:  [1152896, 1153166, 1153703, 1152308, 1153724, 1153728, 1152448, 1152975, 1153879, 1153891,
1152244, 1153150]
2:  [1152564, 1154176, 1136814, 1153264, 1153889, 1138027, 1153943, 1154459, 1138091, 1152436,
1153003]
3:  [1153064, 1153148, 1136791, 1152194, 1152740, 1139195, 1153280, 1152307, 1137466, 1152358,
1153942, 1152421, 1154500]
4:  [1153056, 1153065, 1153106, 1153160, 1153275, 1153577, 1153811, 1153816, 1153853, 1153866,
1153897, 1153922, 1154042, 1154251, 1152277, 1140040, 1152663, 1152673, 1136393, 1136397,
1136422, 1152944]
5:  [1154076, 1154123, 1154124, 1154169, 1154276, 1153262, 1152272, 1135894, 1152290, 1136442,
1153900, 1152379, 1152917, 1152508]
6:  [1152569, 1152633, 1135746, 1152143, 1153169, 1153183, 1153736, 1139928, 1153784, 1152761,
1138968, 1136446, 1153933, 1152910, 1154520]
7:  [1155073, 1140289, 1153091, 1152162, 1152676, 1154233, 1153287, 1140547, 1153877, 1153899]
8:  [1154103, 1152150, 1154229, 1136310, 1152714, 1136342, 1136447, 1140543, 1153860, 1153861,
1153896, 1152958, 1152490]
9:  [1153101, 1136449, 1152904, 1136631]
10: [1140230, 1152075, 1135750, 1154230, 1154232, 1153254, 1152821, 1153945, 1136110, 1136634]
11: [1135899, 1153195, 1152959, 1140548, 1138755]
12: [1140231, 1154068, 1135589]
13: [1153031]
14: []
15: [1138970, 1153014]
16: [1152259]
17: [1152179, 1138043]
18: [1152991]
19: []
20: []
21: []
22: []
23: []
24: [1154071]
25: [1153024]
26: [1153097]
27: []
28: [1152859]
29: []
```

So I repeated this simulation for 10 times and then gave an average between unknown nodes that got sick in the first Time Step.

```
# 10 times simulation
for i in range(10):
   this_infected_nodes, this_new_infected_nodes_step = si_model(G, initial_infected_nodes, i)
   infected_nodes.append(this_infected_nodes)
   new_infected_nodes.append(this_new_infected_nodes_step)
```

In 10 iterations of the simulation the chance of nodes falling sick in the first step were:

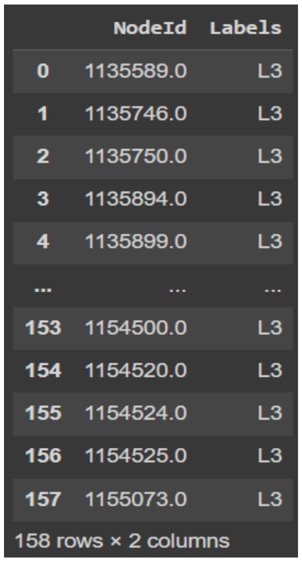
1138027: 40.0%
1138091: 30.0%
1152244: 50.0%
1152307: 40.0%
1152308: 90.0%
1152436: 40.0%
1152448: 30.0%
1152564: 30.0%
1152896: 70.0%
1152975: 30.0%
1153101: 10.0%
1153148: 50.0%
1153150: 30.0%
1153166: 60.0%
1153169: 30.0%
1153703: 30.0%
1153724: 40.0%
1153728: 80.0%
1153811: 30.0%
1153877: 20.0%
1153879: 40.0%
1153889: 20.0%
1153891: 90.0%
1154500: 40.0%

Step 5: Strategies for node Classification.

SVM:

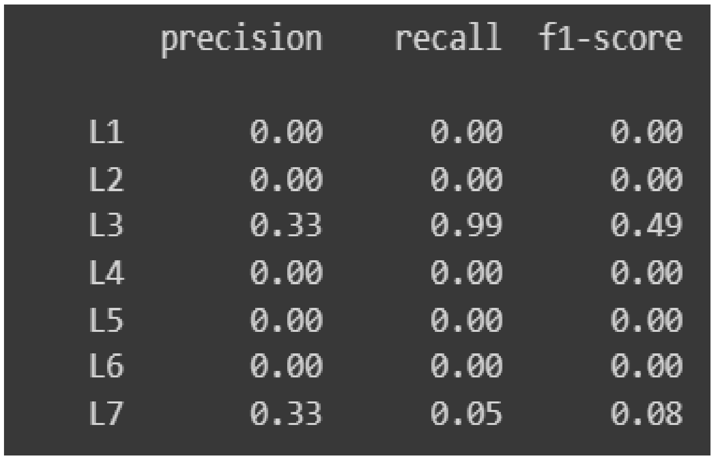
Used degree and clustering coefficient of nodes as features of the vectors. I applied the 90%-10% training and testing but due to lack of training data I got really a bad prediction. All of the Unknown nodes got the label L3.

So print a Classification Record so that it determines how low accurate the SVM is.



	NodeId	Labels
0	1135589.0	L3
1	1135746.0	L3
2	1135750.0	L3
3	1135894.0	L3
4	1135899.0	L3
...	...	...
153	1154500.0	L3
154	1154520.0	L3
155	1154524.0	L3
156	1154525.0	L3
157	1155073.0	L3

158 rows × 2 columns



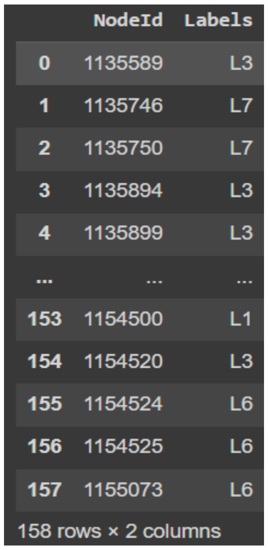
```
      precision    recall  f1-score

L1       0.00       0.00      0.00
L2       0.00       0.00      0.00
L3       0.33       0.99      0.49
L4       0.00       0.00      0.00
L5       0.00       0.00      0.00
L6       0.00       0.00      0.00
L7       0.33       0.05      0.08
```

So I used SVM using node2vec.

SVM with Node2Vec:

Node2vec is a vectorized method that uses random walk.
I determined the size of dimensions, walk_length and num_walks initially and trained my model. based on this method. Surprisingly i got the better result



	NodeId	Labels
0	1135589	L3
1	1135746	L7
2	1135750	L7
3	1135894	L3
4	1135899	L3
...	...	...
153	1154500	L1
154	1154520	L3
155	1154524	L6
156	1154525	L6
157	1155073	L6

158 rows × 2 columns

DecisionTree Classification using one hot encoding:

This method again couldn't be satisfactory. Because of the same reasons as the SVM model.
All the predicted Labels are L5 and it is not really a True prediction.
So I used another node2vec method with the naive bayes Classification.

Naive Bayes Classification using Node2Vec:

In this method we again used the model node2vec on training the Naive Bayes and the result is not really bad.
Better than not having the node2vec in the two previous methods.

To analyze the result we must know what is the node and the edge in this network but overlay most of the unknown nodes got Label 3.
Label 3 is really all over the network.
The results of the two methods using Node2Vec are slightly different.