QUES 1 REPORT

In question 1 we were asked to pick a real-world directed network dataset (with number of nodes > 100) from here and represent the network in terms of its ‘adjacency matrix’ as well as ‘edge list’.

We chose the Wiki-Vote data and then performed a few operations to determine all the aspects we were asked to find.

First we downloaded the data from the link given and analysed the file then extracted the data using python’s OS library and formed created an edge list where first node is the source node and second node is the destination node. This basically means that there is an edge directed from the first node to the second node..

After reading the file downloaded, we will just simply add all of the nodes in a set, that way we will get the total number of nodes in the network.

After this we defined a function create_adj_matrix that takes two arguments: edges and nodes.

- The function first calculates the maximum integer value in the nodes list and stores it in the variable n. It then creates an n+1 by n+1 matrix filled with zeros using a nested list comprehension. This matrix will be the adjacency matrix for the graph.
- The function then iterates through each tuple in the edges list and sets the corresponding entry in the adjacency matrix to 1.
- Finally, the function returns the completed adjacency matrix.

After defining the function, the code calls it with the edges and nodes variables as arguments and assigns the resulting matrix to the variable adj_matrix. This will allow us to analyze the graph using its adjacency matrix.

After that we defined a function create_adj_list that takes two arguments: edges and nodes.

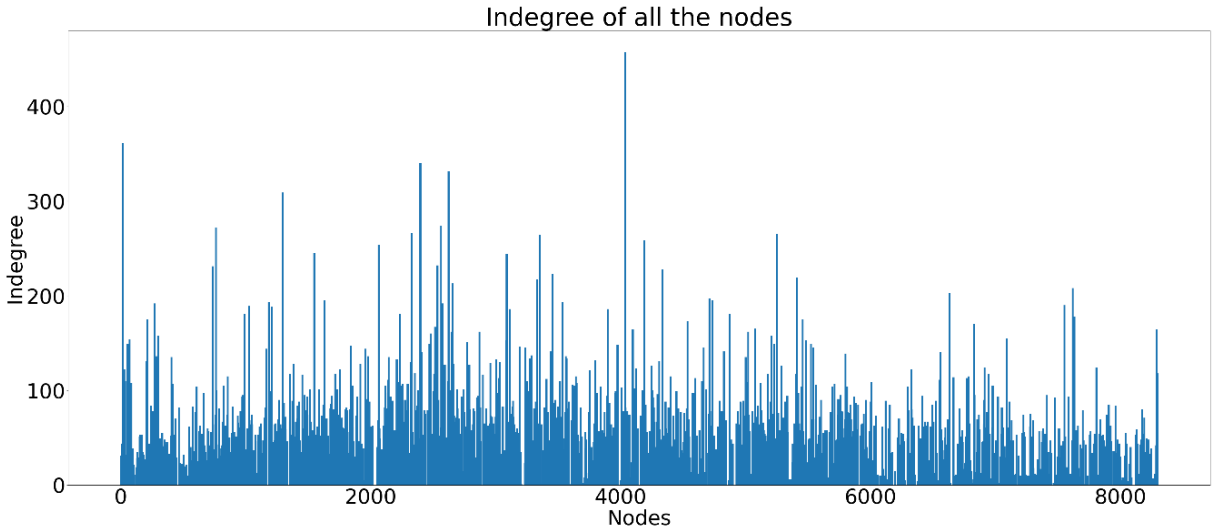
- The function first creates an empty dictionary adj_list, which will be the adjacency list for the graph.
- The function then iterates through each tuple in the edges list. For each tuple, it checks if the first element (edge[0]) is already a key in the adj_list dictionary. If it is, the second element (edge[1]) is appended to the list of adjacent nodes for that key. If it isn't, a new key-value pair is added to the adj_list dictionary with the key as the first element of the tuple and the value as a list containing only the second element of the tuple.
- After adding all the edges to the adj_list dictionary, the function then iterates through each node in the nodes list. If a node is not already a key in the adj_list dictionary, a new key-value pair is added with the key as the node and the value as an empty list. This ensures that all nodes in the graph have an entry in the adjacency list, even if they have no edges.
- Finally, the function returns the completed adjacency list.

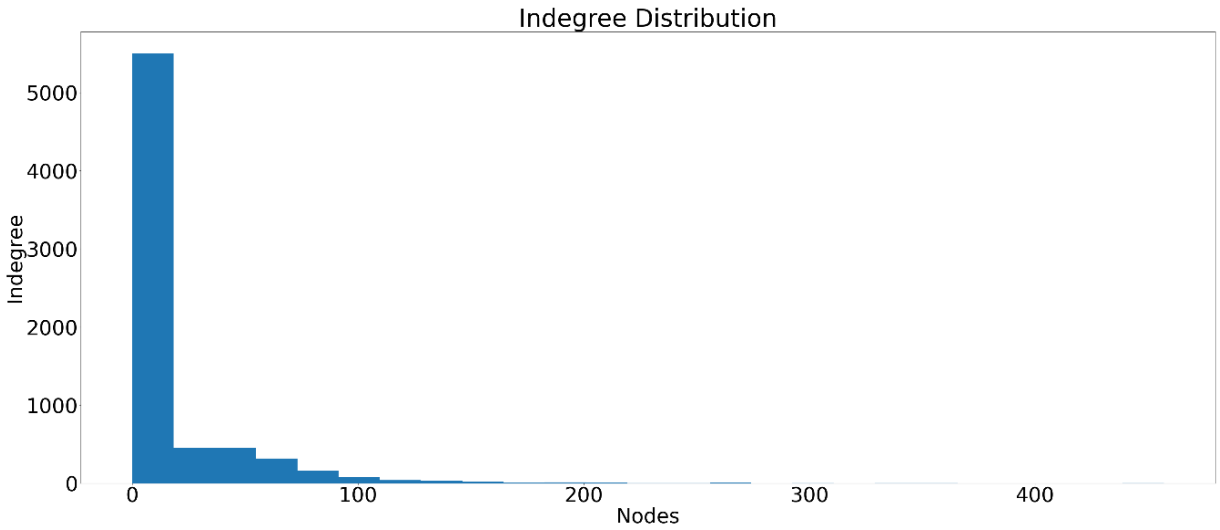
After defining the function, the code calls it with the edges and nodes variables as arguments and assigns the resulting dictionary to the variable adj_list. This will allow us to analyze the graph using its adjacency list.

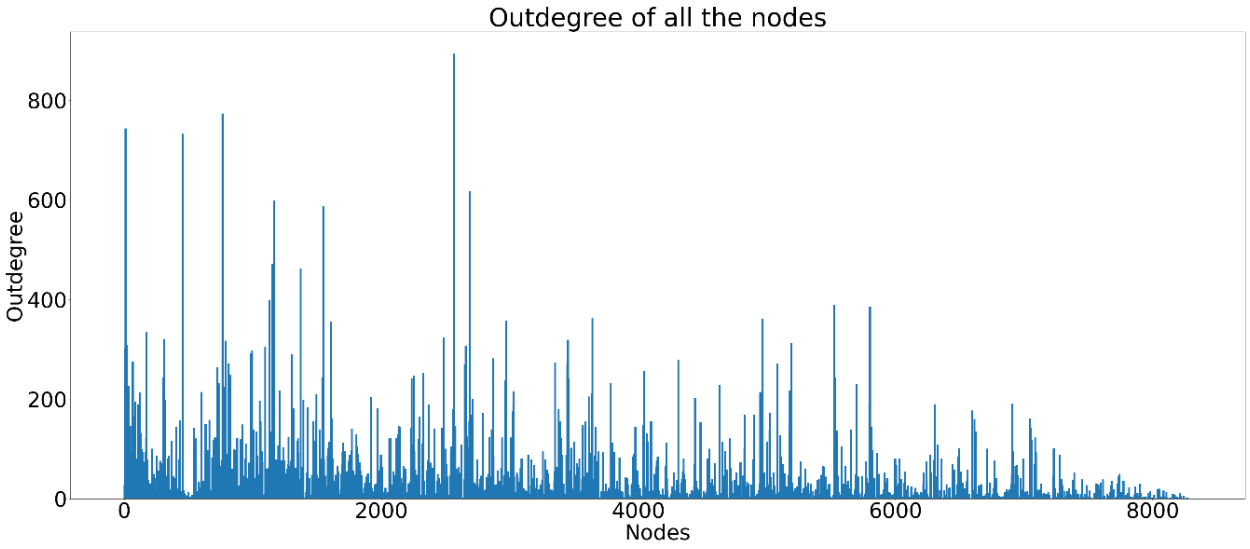
After this we defined a function calculate_indegrees that takes two arguments: adj_matrix and nodes. To find out the indegree of a particular node present in list nodes, we just need to calculate the sum of that particular node column in adjacency matrix. That will be the indegree of that particular node.

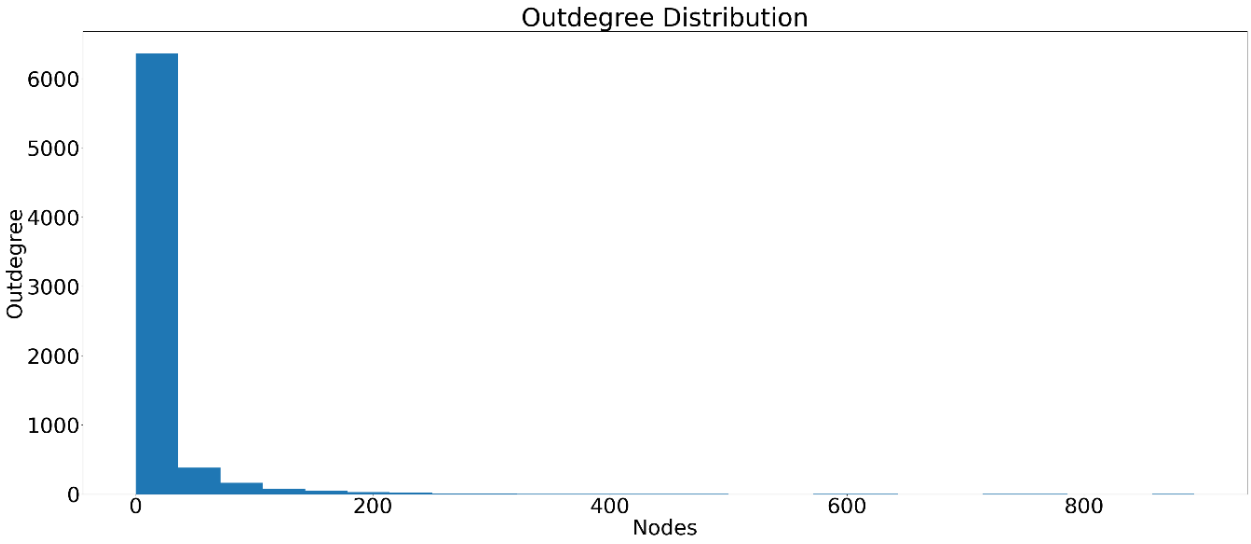
After this we defined a function calculate_outdegrees that takes two arguments: adj_list and nodes. To find out the outdegree of a particular node present in list nodes, we just check the length of the adjacency list of that particular node. That will be the outdegree of that particular node.

After this we just calculated average indegrees and outdegree using simple mean formula and printed it and bar plotted the indegree and outdegree distribution of the network using matplotlib library.





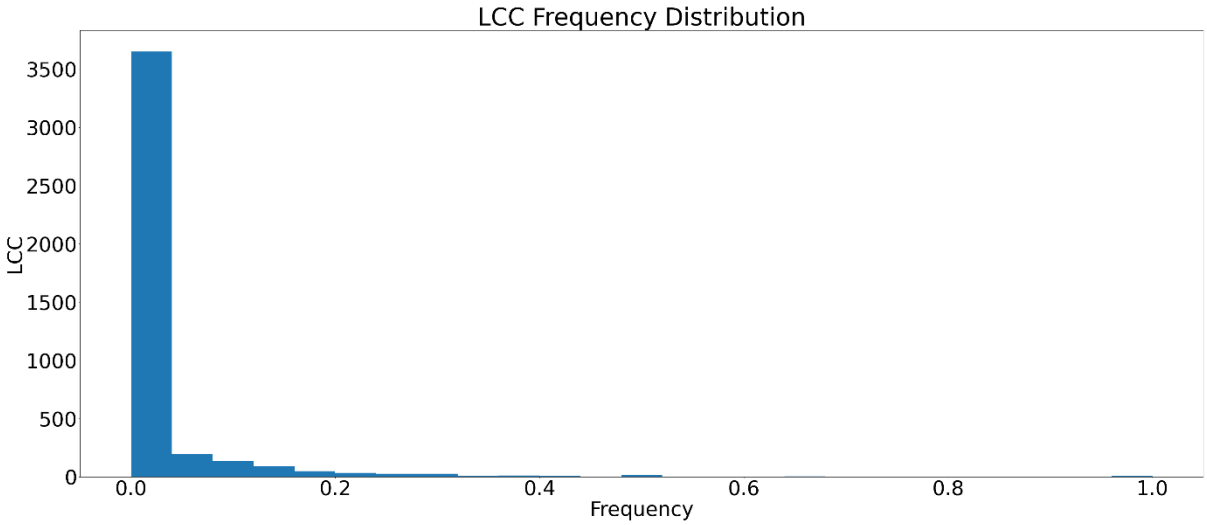




And further to find out the node with maximum indegree and outdegree we simply just traverse through the indegrees and outdegrees dictionaries and find out the highest number of indegree and outdegree and the nodes associated to them.

To calculate the density of network, we first found out the total number of maximum possible edges in the network which is n*(n-1) where n is the total number of nodes. We then divide the total number of edges present in the network by the total number of maximum possible edges to calculate the density of the network.

And finally, to calculate local cluster coefficient of each node we first initialized dictionary to store local clustering coefficient for each node. Then we looped over nodes in adjacency list and checked if the node has neighbours less than 2. If the node has less than 2 neighbours, the clustering coefficient is undefined, so we just skipped that node. After that we counted the number of edges between the node's neighbours and then calculated the local clustering coefficient for the node


LCC Frequency Distribution
LCC
3500
3000
2500
2000
1500
1000
500
0
0.0
0.2
0.4
0.6
0.8
1.0
Frequency