Q1

1. All the important and necessary libraries are imported.

Initial Data Preparation
Pick a real-world directed network dataset (with number of nodes > 100) .

2. The dataset used is Wikipedia vote network from the link :

https : //snap.stanford.edu/data/wiki-Vote.html

3. The dataset is loaded into memory.
4. The dataset has two columns named FromNodeId and ToNodeId.
5. The dataset statistics as per the source of the data :
    i. Nodes : 7115
    ii. Edges : 103689
    iii. Nodes in largest WCC : 7066 (0.993)
    iv. Edges in largest WCC : 103663 (1.000)
    v. Nodes in largest SCC : 1300 (0.183)
    vi. Edges in largest SCC : 39456 (0.381)
    vii. Average clustering coefficient : 0.1409
    viii. Number of triangles : 608389
    ix. Fraction of closed triangles : 0.04564
    x. Diameter (longest shortest path) : 7
    xi. 90-percentile effective diameter : 3.8

6. Find the unique nodes in FromNodeId and ToNodeId.
7. Concatenate the nodes in FromNodeId and ToNodeId to find complete set of nodes.
8. Create duplicate columns for FromNodeId and ToNodeId.
9. For a total of 7115 nodes in the dataset, the node IDs range from 3 to 8297. Therefore, some nodes are not represented in the data and so we will be creating new ids for the source and the target nodes ranging from 0 to 7114.
10. Create a dictionary to assign unique id to each node.
11. Then, map the newly assigned ids to the earlier given node ids in the network.

Question 1 : Link Analysis

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

1. Create a matrix named ad_matrix of shape number_of_nodes x number_of_nodes filled with 0s.
2. Initiate a loop to iterate over all the entries in the data file, which is also the number of edges in the network. For each row extracts the source and target node IDs.
3. Set ad_matrix[source][target] to 1, indicating that there is a directed edge from the source node to the target node.

```
[[0. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 1. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
```

[0. 0. 0. ... 0. 0. 0.]]

Edge list

1. Extract the columns "FromNodeId1" and "ToNodeId1" from a Pandas DataFrame df and converts them into a list of lists.

[[0, 6110], [0, 2716], [0, 4124], [0, 4314], [0, 6111]]

Briefly describe the dataset chosen and report the following:

Wikipedia is a website that allows users from all over the world to create and edit encyclopedia articles for free. Some of these users are administrators who have extra tools to help maintain the website. To become an administrator, a user must go through a process called Request for adminship (RfA), where the Wikipedia community votes on whether or not to promote them. We used data from the complete dump of Wikipedia page edits in January 2008 to extract information on all administrator elections and vote history. The data showed that there were 2,794 elections with a total of 103,663 votes, and 7,066 users participated either by voting or being voted on. Of these, 1,235 elections resulted in a successful promotion, while 1,559 did not. About half of the votes came from existing admins, while the other half were from ordinary Wikipedia users. The network represents all Wikipedia voting data until January 2008, where nodes represent Wikipedia users and directed edges show which user voted on which other user.

1. **Number of Nodes**
    a. Initialize the number of nodes n_nodes to 0.
    b. Find all the unique nodes.
    c. Iterate over all nodes by incrementing n_nodes by 1 in each step.
    d. After the loops are complete, print the value of "n_nodes".
    e. The number of nodes is 7115.

2. **Number of Edges**
    a. Initialize the number of edges n_edges to 0.
    b. Use a for loop to iterate over all the nodes in the graph. This loop starts from 0 and goes up to (n_nodes - 1).
    c. Inside the first for loop, use another for loop to iterate over all the nodes in the graph. This loop also starts from 0 and goes up to (n_nodes - 1).
    d. Inside the nested for loop, check if the value of the element at index (i, j) in the adjacency matrix is not equal to 0.
    e. If the value of the element at index (i, j) is not equal to 0, increment the value of "n_edges" by 1.
    f. After the loops are complete, print the value of "n_edges".
    g. The number of edges is 103689
3. **Avg In-degree**

    Formula : average_In_degree = sum of the In-degrees of all nodes / total nodes

    a. Create an array In_Degree of size n_nodes with all elements initialized to 0.

b. Loop over all columns j of the adjacency matrix ad_matrix, where n_nodes is the number of nodes in the graph.
c. For each column j, loop over all rows i of the adjacency matrix.
d. If the value of the adjacency matrix at (i, j) is not zero, increment the value of In_Degree at index j.
e. Loop over all elements in In_Degree and sum them to get the total in-degree of the graph.
f. Calculate the average in-degree of the graph by dividing the total in-degree by the number of nodes n_nodes.
g. Return the calculated average in-degree.
h. The average in-degree is 14.573295853829936

4. **Avg. Out-Degree**

Formula : average_out_degree = sum of the out-degrees of all nodes / total nodes

a. Create a numpy array of zeros with size n_nodes) to represent the out-degree of each node
b. Loop over all nodes and their adjacent nodes in the adjacency matrix and increment the out-degree of each node for every non-zero entry in the matrix.
c. Calculate the total out-degree by summing up the out-degree of all nodes in the Out_Degree array.
d. Calculate the average out-degree per node by dividing the total out-degree by the number of nodes.
e. The average out-degree is 14.573295853829936

5. **Node with Max In-degree**

a. Initialize two variables, node_max_in_deg and max_in_deg, to keep track of the node with the highest in-degree and its in-degree, respectively. Both are initially set to -1 and 0.
b. Loop through all the nodes in the graph represented by the adjacency matrix.
c. Check if the in-degree of the current node is greater than or equal to the current maximum in-degree max_in_deg.
d. If it is, update the value of max_in_deg to updated to this value, and set the value of node_max_in_deg to the current node index.
e. After the loop completes, the node_max_in_deg and max_in_deg variables contain the node with the highest in-degree and its in-degree, respectively.
f. Finally, the code prints out the node_max_in_deg and max_in_deg values to show which node has the maximum in-degree.
g. Inverse map to get the original node id to find the node with max in-degree
h. NodeID 4037 has max in-degree of 457

6. **Node with Max out-degree**

a. Initialize two variables, node_max_out_deg and max_out_deg, to keep track of the node with the highest in-degree and its in-degree, respectively. Both are initially set to -1 and 0.
b. The code iterates through each node in the graph, and checks if the out-degree of that node is greater than or equal to the current max_out_deg. If it is, the max_out_deg variable is updated to this value and node_max_out_deg is updated to the index of the current node.

c. Once all nodes have been checked, the node_max_out_deg variable contains the index of the node with the maximum out-degree and max_out_deg contains the value of that out-degree.
d. Finally, the code prints out the node_max_out_deg and max_out_deg values to show which node has the maximum in-degree.
e. Inverse map to get the original node id to find the node with max out-degree
f. NodeID 2565 has max out-degree of 893.

7. The density of the network

The network density represents the fraction of total number of edges present in the network to the number of edges possible.
Formula : network_density=no_of_edges/(no_of_nodes x (no_of_nodes-1))

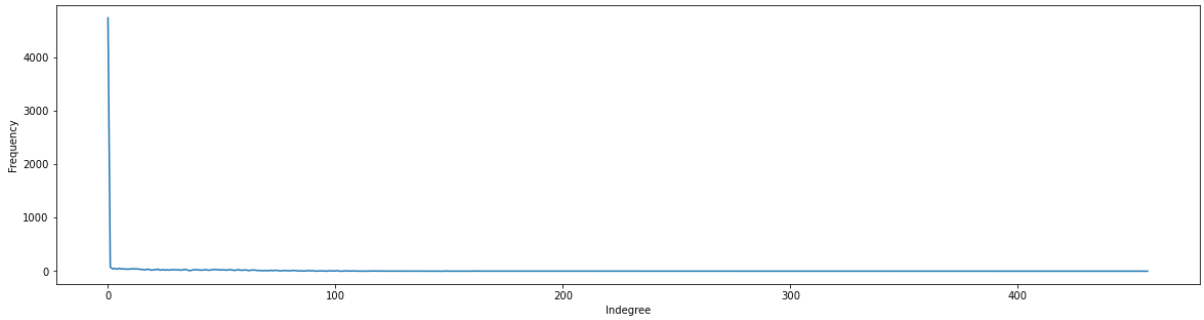
a. Calculate network density by dividing the total edges by the total number of possible edges (n_nodes * (n_nodes - 1))

b. The density of the network is 0.0020485375110809584

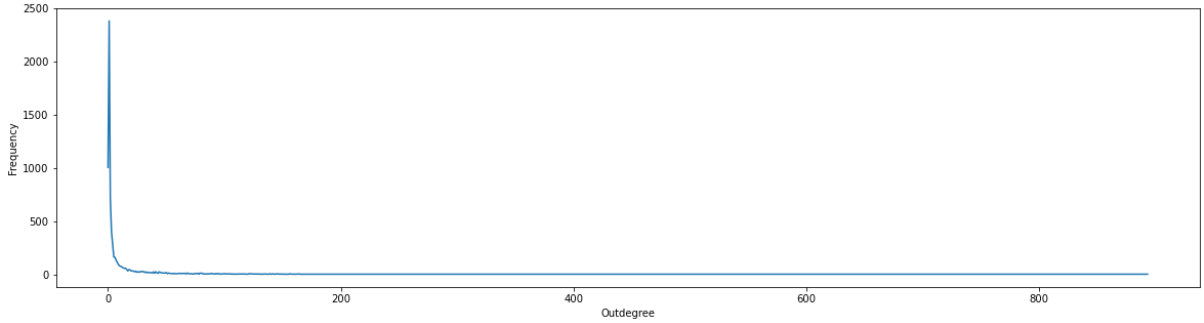
Further tasks to be performed:

1. **Plot degree distribution of the network (in case of a directed graph, plot in-degree and out-degree separately).**
   a. Create a dataframe with in-degree and out-degrees of nodes with their corresponding frequencies.
   b. Sort the dataframe by Indegree and Outdegree values in increasing order.
   c. Plot in-degree and out-degree distribution using matplotlib library.

Plotting in-degree distribution



Plotting out-degree distribution



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

Formula: local clustering coefficient for node i = C(i) = |{e(jk) : v(j), v(k) ∈ N(i), e(jk)∈ E}|/(deg(i) * (deg(i) - 1))

a. First, creates a set of all the neighbors of the input node.
b. If the number of neighbors is less than or equal to 1, the function returns 0 as the clustering coefficient.
c. Otherwise, the function counts the number of triangles formed by the node and its neighbors.
d. For each neighbor, the function iterates through its neighbors to find common neighbors with the input node.
e. If a common neighbor is found, the function increments the count of triangles.
f. Finally, the function returns the clustering coefficient of the node, which is calculated by dividing the number of triangles by the product of the number of neighbors and the number of neighbors minus 1.
   Following is the list of Local clustering coefficient of five nodes:

Local clustering coefficient of node 30: 0.05
Local clustering coefficient of node 3: 0.07509881422924901
Local clustering coefficient of node 25: 0.10074906367041199
Local clustering coefficient of node 4: 0.13669950738916256
Local clustering coefficient of node 5: 0.191699604743083

Plotting clustering-coefficient distribution