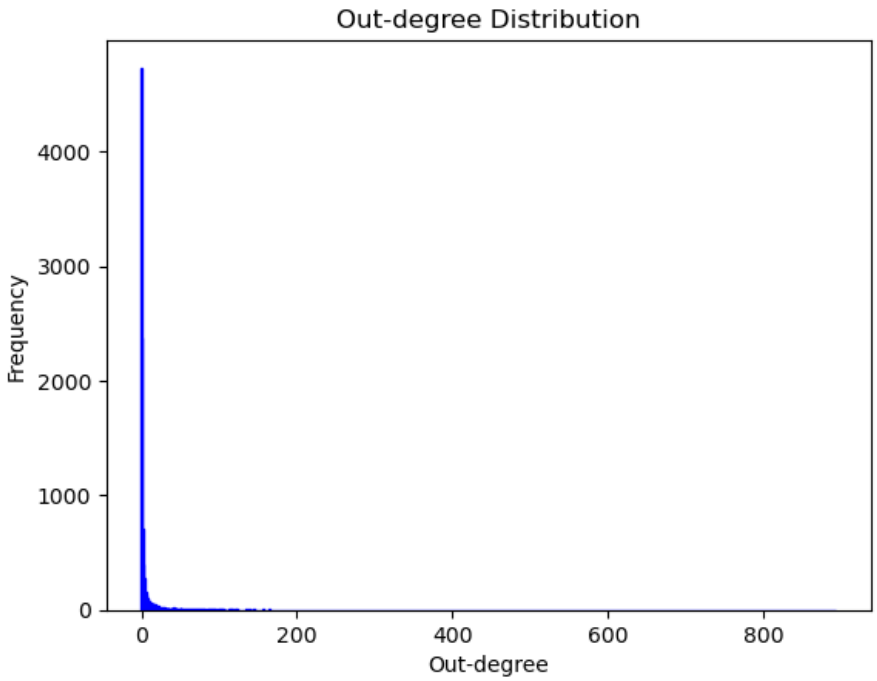
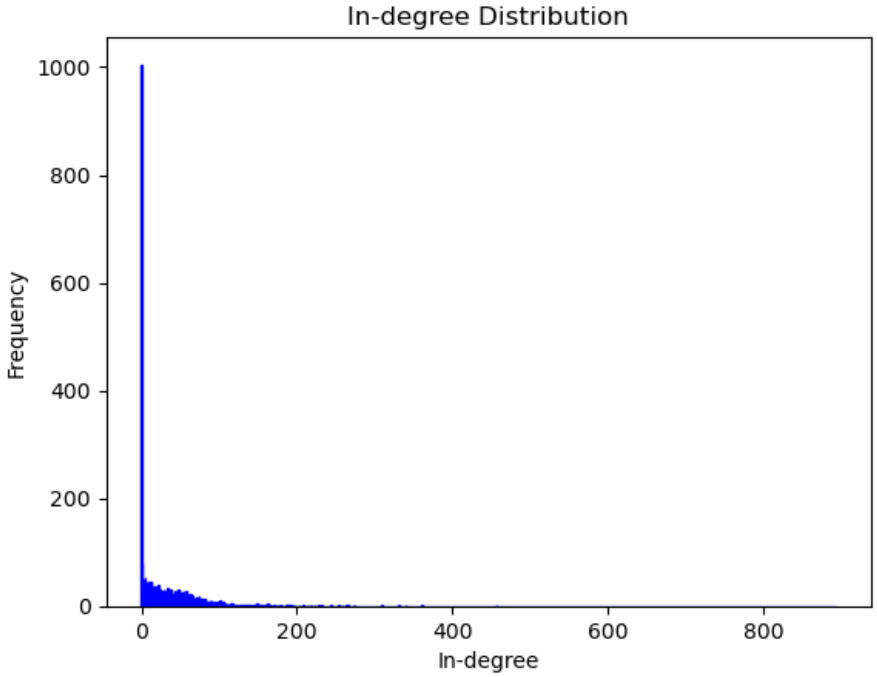
Ans 1)
Part 1-

1. read_edges : takes a file path as input, opens the file, reads edge data, and returns a list of tuples representing the edges. It skips lines starting with "#" to ignore comments.
2. create_adjacency_list : Given the list of edges, we construct an adjacency list representation of the network. It initializes an empty dictionary and iterates through the edges, adding them to the dictionary as key-value pairs.
3. number_of_nodes : calculates the total number of unique nodes in the network. It first creates a set of keys from the adjacency list and then iterates through the values, updating the set with new nodes as necessary. The length of the set is the total number of nodes.
4. number_of_edges : returns the length of the input edge list, which corresponds to the total number of edges in the network
5. in_out_degrees : initializes two dictionaries and iterates through the edges, updating the degree counts for each node accordingly.
6. avg_in_out_degrees : calculates the average in-degrees and out-degrees for the network. It takes the in-degree and out-degree dictionaries and the total number of nodes as input, sums the degree values, and divides them by the number of nodes to find the averages. (total in/out degree divided by number of nodes)
7. Max_in_out_degree : identifies the nodes with the maximum in-degree and out-degree values. It takes the in-degree and out-degree dictionaries as input and uses the max function with the key argument to find the nodes with the highest degrees.
8. network_density : estimates the density of the network, a measure of how interconnected the nodes are. It takes the total number of nodes and edges as input and calculates the density using the formula density = num_edges / (num_nodes * (num_nodes - 1)).

<u>1.Plot degree distribution of the network (in case of a directed graph, plot in-degree and out-degree separately).</u>

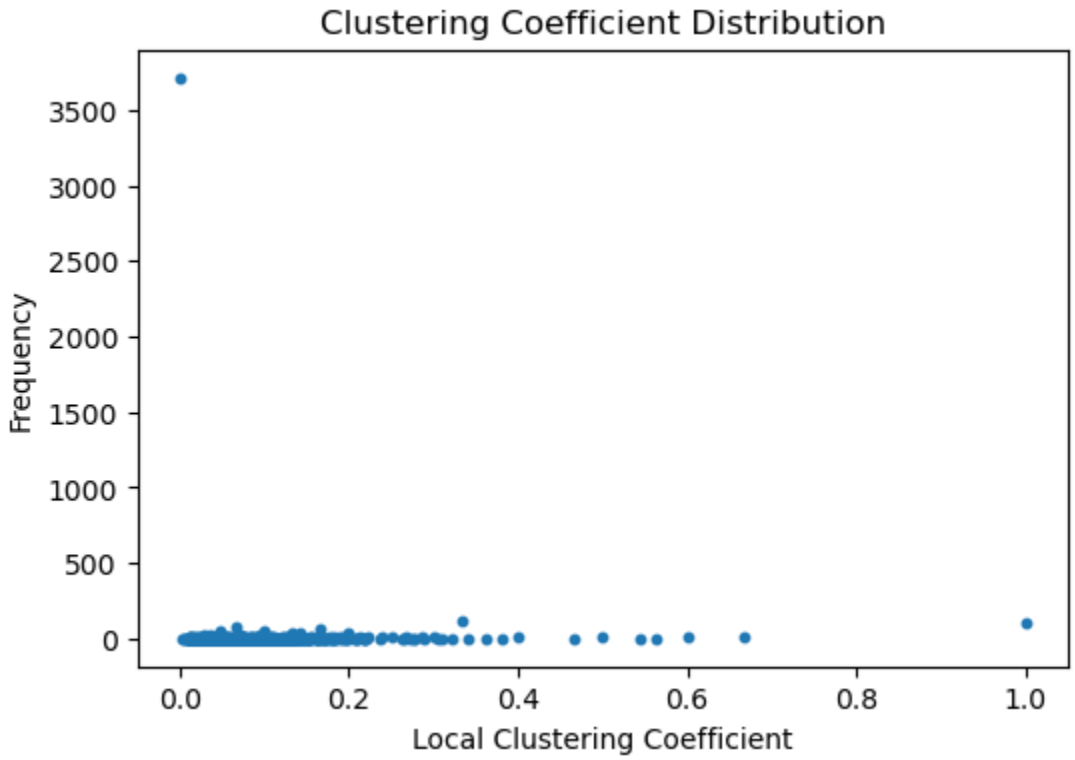




Here when we iterate through the file, we consider source and targets nodes and keep a track of these nodes in dictionaries in_degree and out_degree.
The in_degree and out_degree dictionaries store the in-degree and out-degree of each node respectively, where in-degree is the number of incoming edges to a node and out-degree is the number of outgoing edges from a node.
We also calculate the maximum degree that a node has in the graph.
Then we compute the frequency of both in and out degree using the above dictionary.

Then we plot the above graph using matplotlib library.

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



Here we are also iterating through the file and find target and source node.We also keep track of neighbours by checking if the source to target relationship is already saved or not.
For each node in this dictionary we check if the number of neighbors is less than 2. if yes we set the local clustering coefficient to 0 and continues to the next node.
Otherwise we initialize a counter for the number of edges that connect pairs of neighbors of the node. We iterate over all pairs of neighbors, checking whether they are connected by an edge. If yes we increase the count.
We compute the distribution of local clustering coefficients by counting the number of occurrences of each coefficient value in the dictionary.
We then plot the graph using matplotlib library.

Q2.

Here we are analyzing networks. It takes a directed graph that is stored in an edge list file named "Wiki-Vote.txt" and reads it into memory using the NetworkX library. It then calculates several graph metrics for each node in the graph, including PageRank, Authority, and Hub scores. The scores are then saved to separate output files, and the code prints the number of

nodes and edges in the graph to the console. Finally, it identifies and displays the top 10 nodes based on the three metrics on the console. To make it easier for comparison.

The code is particularly useful in analyzing the importance of nodes in a directed graph based on different metrics. It has numerous applications, including identifying influential individuals in social networks or ranking web pages based on their importance in a web graph. Furthermore, the code is well-commented and straightforward to understand, making it accessible to users with varying levels of programming experience.

Comparison Of Results

The results obtained from the PageRank, Authority, and Hub algorithms reveal that different algorithms rank the importance of nodes in the network differently. While some nodes, like Node 4037 and Node 2398, appear in the top 10 lists for multiple algorithms, the lists for each algorithm are not identical. This is due to the fact that these algorithms measure different aspects of the network: PageRank focuses on the importance of a node based on the number and quality of incoming links; Authority, similar to PageRank, considers incoming links, but also accounts for the Hub scores as part of the HITS algorithm; and Hub measures the importance of a node based on the number and quality of outgoing links. The observed differences in the rankings indicate that the three algorithms capture distinct perspectives on node importance within the network.