

**Name: Syed Umair Andrabi**

**Reg No: 22MCB0015**

**Subject: Social Network Analysis**

**ASSIGNMENT 1**

**Question 1:**

Given edge\_list.csv

1. Load data
2. Plot directed and undirected graph.
3. Print adjacency matrix.

**Input**:

src,dst,weight

A,B,5.0

B,C,2.0

C,D,7.5

D,E,1.2

A,E,4.6

B,F,3.0

C,G,6.4

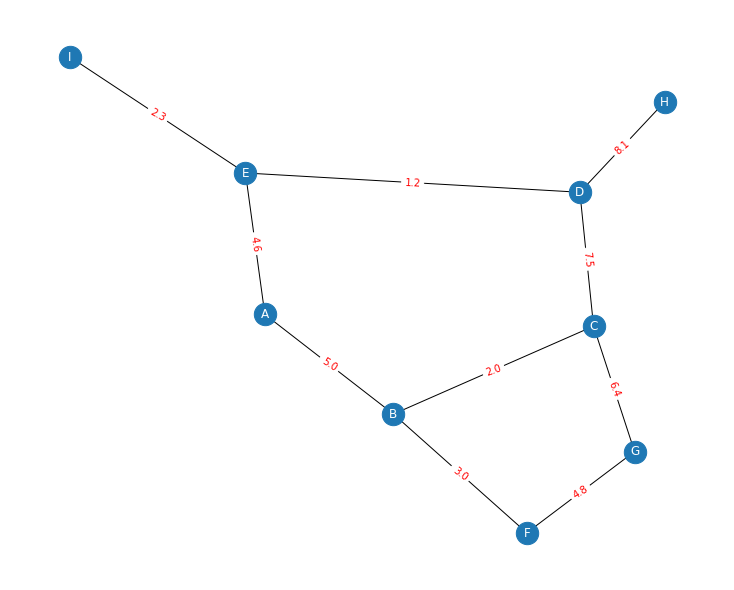
D,H,8.1

E,I,2.3

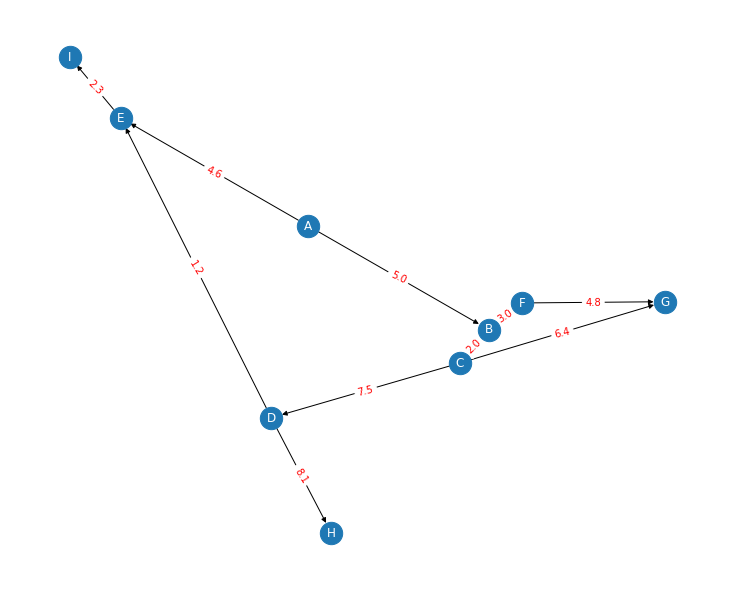
F,G,4.8

**OUTPUT:**

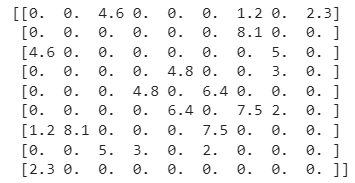
Undirected graph plot



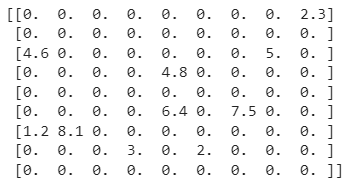
Directed graph plot:



**Undirected Graph Adjacency Matrix:**

****

**Directed Graph Adjacency Matrix:**

****

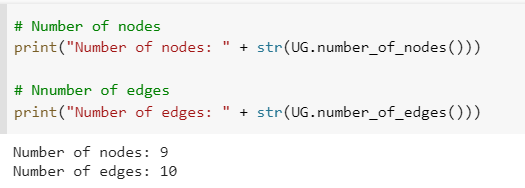
**Explanation:**

We are creating a graph from a CSV file that contains a list of edges with weights. We then create an adjacency matrix from the edges list, which is used to create an undirected and directed graph using the NetworkX library. We label the nodes of the graph with the corresponding node names and then draw the undirected and directed graphs using the NetworkX library. The adjacency matrices of the undirected and directed graphs are printed to the console at the end of the code.

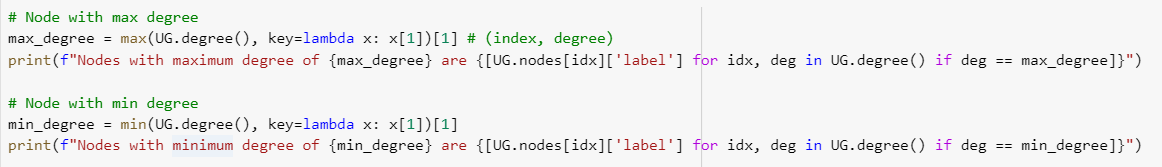
**Question 2:**

**For undirected graph:**

1. Print number of nodes.
2. Number of edges.
3. Nodes with max degree
4. Nodes with min degree.
5. Number of nodes and edges:



1. Nodes with max degree and min degree



Output:

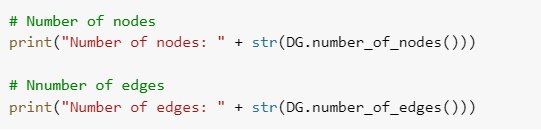
Nodes with maximum degree of 3 are ['B', 'E', 'C', 'D']

Nodes with minimum degree of 1 are ['I', 'H']

**For directed graph:**

Print number of nodes, number of edges, nodes with max out-degree, nodes with min out-degree, nodes with max in-degree, nodes with min in-degree.

1. Number of nodes and edges:

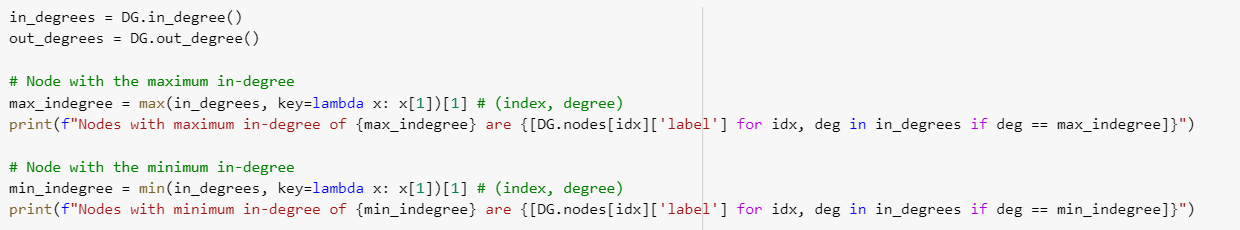


OUTPUT

Number of nodes: 9

Number of edges: 10

1. Nodes with min and max in-degree

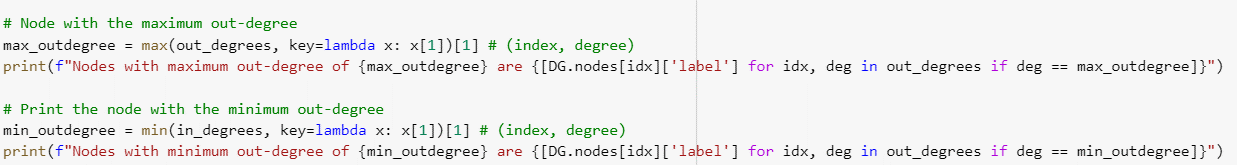


OUTPUT

Nodes with maximum in-degree of 2 are ['E', 'G']

Nodes with minimum in-degree of 0 are ['A']

1. Nodes with min and max out-degree



OUTPUT:

Nodes with maximum out-degree of 2 are ['A', 'B', 'C', 'D']

Nodes with minimum out-degree of 0 are ['I', 'G', 'H']

**Question 3:**

**For undirected graph:**

Print sum of all weights of all outgoing edges for each vertex.

**For directed graph:**

Print sum of weights of all incoming and outgoing edges for each vertex.

**For undirected graph:**

For undirected graph we defined a function sum\_of\_edges\_weight()

def sum\_of\_edges\_weight():

  # Get the sum of weights of all outgoing edges for each vertex

  outgoing\_weights = {}

  for node in undirectedGraph.nodes():

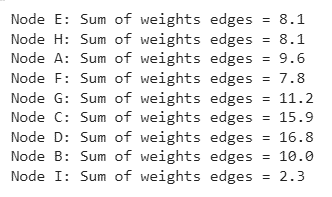
    outgoing\_weights[node] = sum([edge[2]['weight'] for edge in undirectedGraph.edges(node, data=True)])

  # Print the sum of weights of outgoing edges for each vertex

  for node, weight in outgoing\_weights.items():

    print(f"Node {undirectedGraph.nodes[node]['label']}: Sum of weights edges = {round(weight, 2)}")

**OUTPUT**



**For directed graph:**

For undirected graph we are calculating the sum of weights of incoming and outgoing edges for each node in a directed graph. We create two dictionaries to store the sum of weights of outgoing and incoming edges for each node, respectively. Finally, we print the sum of weights of incoming and outgoing edges for each node along with the node label using formatted strings.

# Get the sum of weights of all outgoing edges for each vertex

outgoing\_weights = {}

for node in directedGraph.nodes():

    outgoing\_weights[node] = sum([edge[2]['weight'] for edge in directedGraph.out\_edges(node, data=True)])

# Get the sum of weights of all incoming edges for each vertex

incoming\_weights = {}

for node in directedGraph.nodes():

    incoming\_weights[node] = sum([edge[2]['weight'] for edge in directedGraph.in\_edges(node, data=True)])

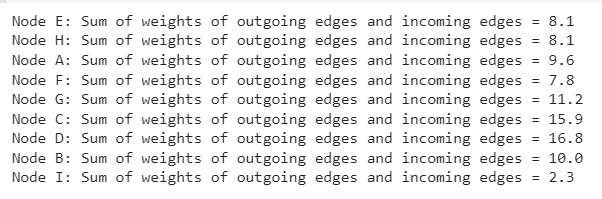
# Print the sum of weights of incoming and outgoing edges for each vertex

for node, outgoing\_weight in outgoing\_weights.items():

    incoming\_weight = incoming\_weights[node]

    print(f"Node {directedGraph.nodes[node]['label']}: Sum of weights of outgoing edges and incoming edges = {round(outgoing\_weight + incoming\_weight, 2)}")

**OUTPUT**

****

| **Node** | **Neighbor Weights** | **Sum of Weights** |
| --- | --- | --- |
| A | B: 0, C: 4.6, G: 1.2, I: 2.3 | 9.6 |
| B | A: 0, C: 0, D: 0, G: 8.1 | 10.0 |
| C | A: 4.6, B: 0, D: 0, H: 5.0 | 15.9 |
| D | B: 0, C: 0, E: 4.8, H: 3.0 | 16.8 |
| E | D: 4.8, F: 6.4 | 8.1 |
| F | E: 6.4, G: 7.5, H: 2.0 | 7.8 |
| G | A: 1.2, B: 8.1, F: 7.5 | 11.2 |
| H | C: 5.0, D: 3.0, F: 2.0 | 8.1 |
| I | A: 2.3 | 2.3 |

The summation equation for the sum of weights for each node remains the same:

, where E is the set of edges in the graph, w(i, j) is the weight of the edge connecting node i and node j.

**Question 4:**

**Calculate and print values for various centralities for each node. Get node with minimum and maximum of each centrality. For undirected as well as directed.**

We calculate various centrality measures (Degree, Betweenness, Closeness, PageRank, Eigenvector) for each node in an undirected graph using the NetworkX library. We then store the calculated centrality values in a dictionary, which is converted to a Pandas DataFrame. Finally, we print the sorted DataFrame displaying the centrality measures for each node.

1. Degree Centrality: For an undirected graph, the degree centrality of a node **v** is defined as:

where deg(v) is the degree of node **v**, and n is the total number of nodes in the graph.

1. Betweenness Centrality: The betweenness centrality of a node **v** is defined as:

where σ(s, t) is the total number of shortest paths between nodes **s** and **t**, and σ(s, t | v) is the number of those paths that pass through node **v**. The summation is taken over all pairs of nodes (s, t) in the graph.

1. Closeness Centrality: The closeness centrality of a node **v** is defined as:

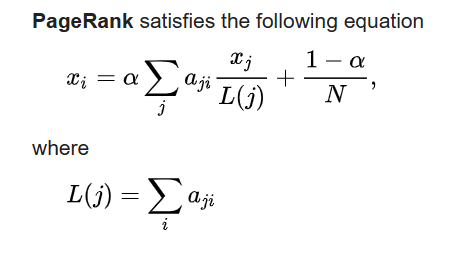
where d(v, u) is the shortest path length between nodes **v** and **u**, and the summation is taken over all nodes **u** in the graph.

1. PageRank: The PageRank of a node **v** can be calculated iteratively using the following formula:

where d is the damping factor (usually set to 0.85), n is the total number of nodes in the graph, L(u) is the number of outgoing links from node **u**, and the summation is taken over all nodes **u** that link to node **v**.

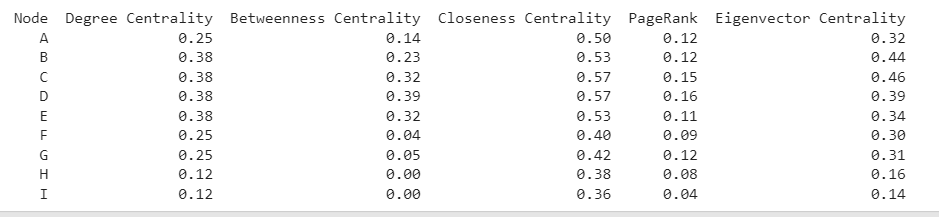
1. Eigenvector Centrality: The eigenvector centrality of a node **v** is defined as the principal eigenvector of the adjacency matrix **A** of the graph. If **x** is the eigenvector corresponding to the largest eigenvalue, then the eigenvector centrality of node **v** is given by:

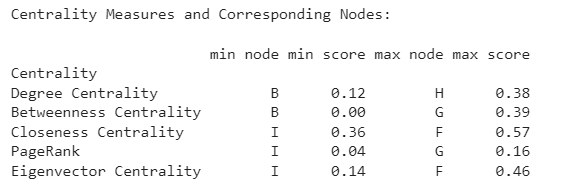
where x\_v is the component of the eigenvector **x** corresponding to node **v**.



**OUTPUT:**

**For undirected**

****

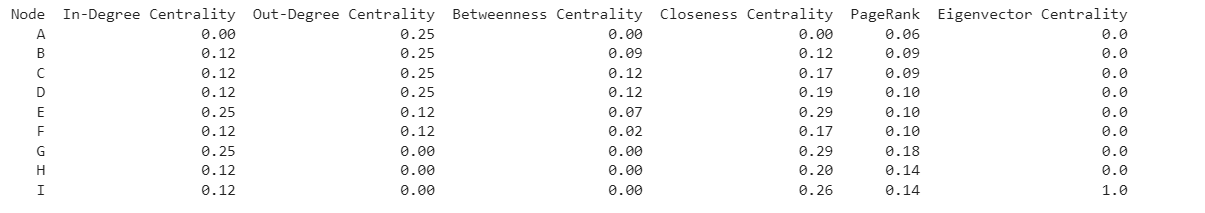
****

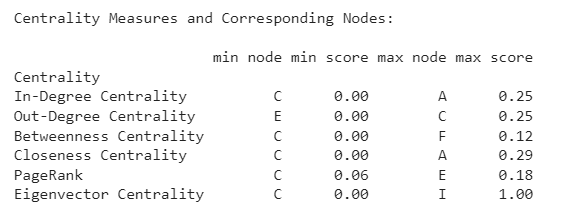
It signifies the relative importance of nodes in the network based on different criteria:

1. Degree Centrality: Node B has the lowest degree centrality (0.12), while Node H has the highest (0.38). This means that H has more connections than B in the network, making H more central based on the number of direct connections.
2. Betweenness Centrality: Node B has the lowest betweenness centrality (0.00), while Node G has the highest (0.39). This means that G is more often found on the shortest paths between pairs of nodes in the network, indicating that G serves as a bridge or connector between different parts of the network.
3. Closeness Centrality: Node I has the lowest closeness centrality (0.36), while Node F has the highest (0.57). This means that F is, on average, closer to all other nodes in the network than I. A higher closeness centrality indicates that a node can reach other nodes more quickly, suggesting a more central position in the network.
4. PageRank: Node I has the lowest PageRank (0.04), while Node G has the highest (0.16). This means that G is considered more important in the network based on the link structure and the relative importance of its neighbors. PageRank is often used to rank web pages based on their incoming links.
5. Eigenvector Centrality: Node I has the lowest eigenvector centrality (0.14), while Node F has the highest (0.46). This means that F is more central based on its connections to other well-connected nodes. Eigenvector centrality takes into account not only the number of connections but also the quality of connections in the network.

These values provide insights into the structure of the network and can help identify influential nodes, bridges, or bottlenecks within the network based on different criteria.

**For directed**

****

****