

**Batch :- D-2 Roll No. :- 16010122151**

**Experiment :- 04**

**TITLE : Performing Graph Analytics**

**AIM:** To analyze the structural properties of a real-world social network by constructing a graph representation, identifying key players and influential individuals through centrality measures, and detecting communities within the network using appropriate algorithms.

---

**Expected OUTCOME of Experiment:**

CO3: Perform the social data analytics

---

**Books/ Journals/ Websites referred:**

Students have to list.

---

**Pre Lab/ Prior Concepts:**

Students should have a basic understanding of:

Graph theory: Nodes, edges, directed and undirected graphs, weighted graphs.

Data structures: Lists, dictionaries.

Python programming: Basic syntax, data manipulation, libraries like NetworkX.

Statistical concepts: Mean, standard deviation, correlation.

Visualization techniques: Basic plotting using libraries like Matplotlib.

## Procedure:

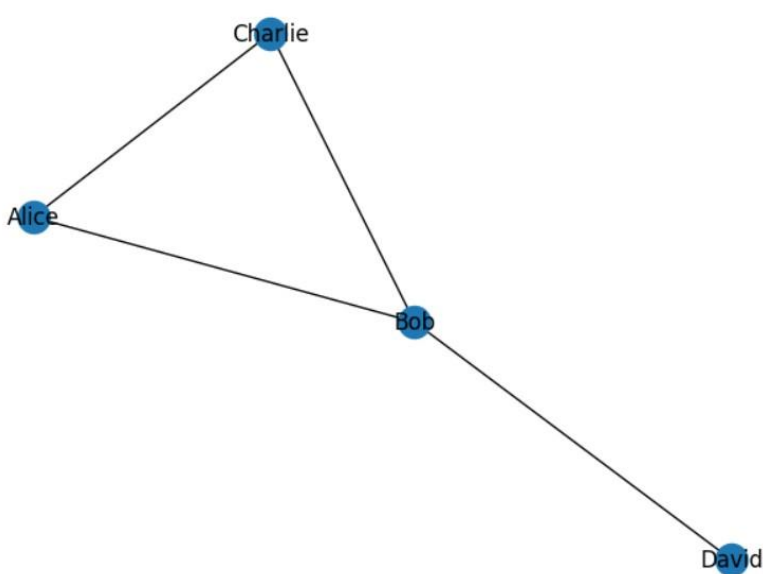
### Building a Social Network Graph with NetworkX

```
1 import networkx as nx
2
3 # Create an empty graph
4 G = nx.Graph()
5
6 # Add nodes (individuals)
7 G.add_nodes_from(['Alice', 'Bob', 'Charlie', 'David'])
8
9 # Add edges (relationships)
10 G.add_edge('Alice', 'Bob')
11 G.add_edge('Alice', 'Charlie')
12 G.add_edge('Bob', 'Charlie')
13 G.add_edge('Bob', 'David')
14
15 # Print the graph
16 print(G.nodes())
17 print(G.edges())
```

[ 'Alice', 'Bob', 'Charlie', 'David' ]  
[ ('Alice', 'Bob'), ('Alice', 'Charlie'), ('Bob', 'Charlie'), ('Bob', 'David') ]

### Visualizing the Graph

```
1 import matplotlib.pyplot as plt
2
3 # Draw the graph
4 nx.draw(G, with_labels=True)
5 plt.show()
```



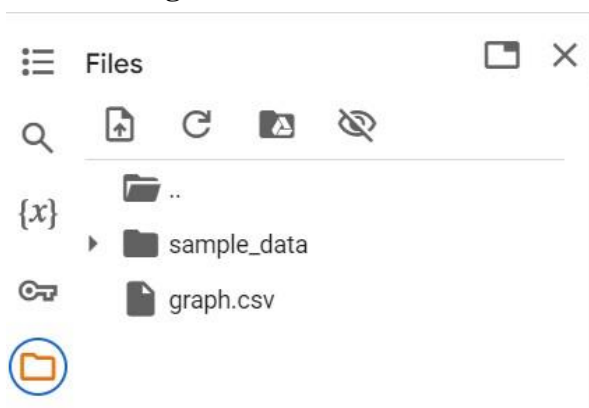
```
graph TD
    Alice --- Bob
    Alice --- Charlie
    Bob --- Charlie
    Bob --- David
    Charlie --- Alice
    Charlie --- Bob
    David --- Bob
```

## Exporting a NetworkX Graph to CSV

```

1  import networkx as nx
2  import pandas as pd
3
4  # Create a sample graph
5  G = nx.Graph()
6  G.add_edges_from([('A', 'B'), ('A', 'C'), ('B', 'C'), ('B', 'D')])
7
8  # Convert to edge list
9  edgelist = nx.to_edgelist(G)
10
11 # Create a pandas DataFrame, including a column for edge attributes
12 df = pd.DataFrame(edgelist, columns=['source', 'target', 'attributes'])
13
14 # Export to CSV
15 df.to_csv('graph.csv', index=False)
  
```

The csv file gets created



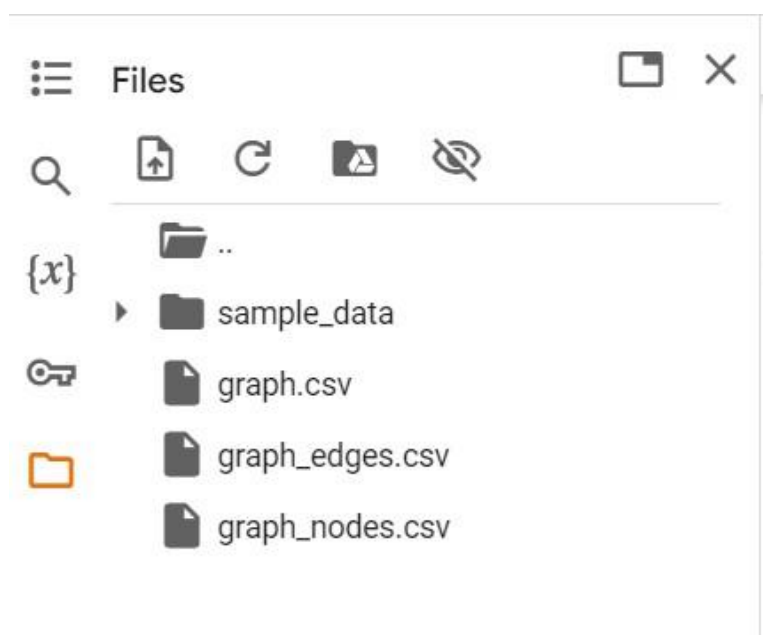
Contents of the csv file

	A	B	C	D
1	source	target	attributes	
2	A	B	{}	
3	A	C	{}	
4	B	C	{}	
5	B	D	{}	
6				
7				

## Creating and exporting a NetworkX Graph with edge attributes and node attributes to a csv file

```
1 import networkx as nx
2 import pandas as pd
3
4 # Create a graph with node and edge attributes
5 G = nx.Graph()
6 G.add_edge('A', 'B', weight=2.5)
7 G.add_edge('A', 'C', weight=1.0)
8 G.nodes['A']['color'] = 'red'
9
10 # Convert to edge list with attributes
11 edgelist = [(u, v, d) for u, v, d in G.edges(data=True)]
12
13 # Create a pandas DataFrame
14 df = pd.DataFrame(edgelist, columns=['source', 'target', 'weight'])
15
16 # Add node attributes as a separate DataFrame if needed
17 node_attributes = pd.DataFrame.from_dict(dict(G.nodes(data=True)), orient='index')
18 node_attributes.columns = ['color']
19
20 # Export to CSV
21 df.to_csv('graph_edges.csv', index=False)
22 node_attributes.to_csv('graph_nodes.csv')
```

## csv files get created



### Contents of graph\_edges.csv

	A	B	C	D
1	source	target	weight	
2	A	B	{'weight': 2.5}	
3	A	C	{'weight': 1.0}	
4				
5				

### Contents of graph\_nodes.csv

	A	B	
1		color	
2	A	red	
3			

### Importing a graph from a csv file

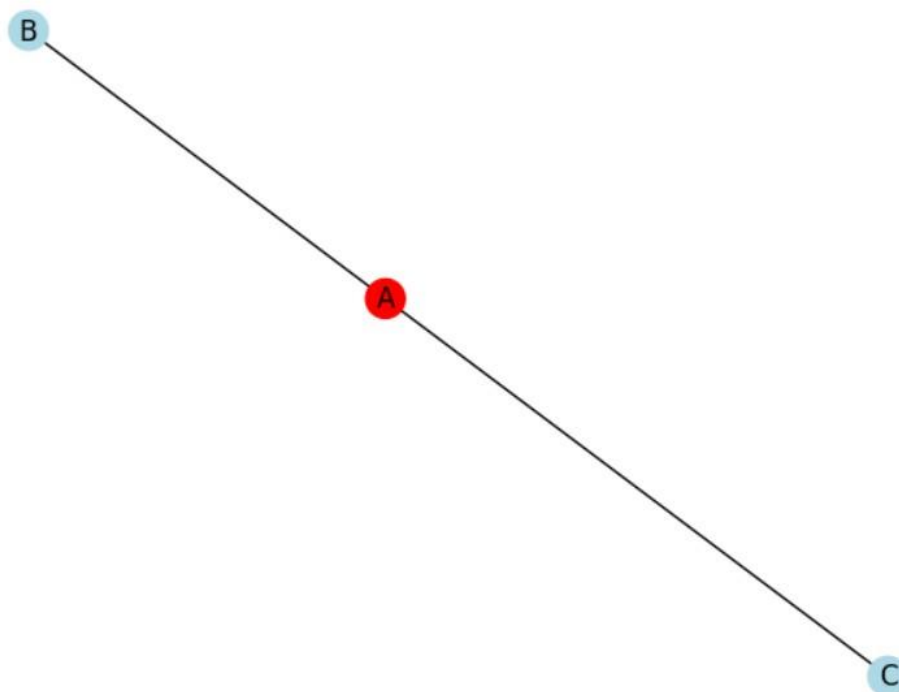
```

1  import pandas as pd
2  import networkx as nx
3  import matplotlib.pyplot as plt
4
5  # Read edge list from CSV
6  df_edges = pd.read_csv('graph_edges.csv')
7
8  df_nodes = pd.read_csv('graph_nodes.csv', index_col=0)
9
10 # Extract numeric weights from the 'weight' column (assuming they are stored as dictionaries)
11 # If your 'weight' column is just a number, remove this line
12 df_edges['weight'] = df_edges['weight'].apply(lambda x: float(x.strip("{}").split(": ")[1]) if isinstance(x, str) else x)
13
14 # Create a graph from the edge list
15 G = nx.from_pandas_edgelist(df_edges, source='source', target='target', edge_attr='weight')
16
17 # Add node attributes if available
18 if df_nodes is not None:
19     nx.set_node_attributes(G, df_nodes.to_dict('index'))
20
21 # Print the graph
22 print(G.nodes(data=True))
23 print(G.edges(data=True))
24
25 # Draw the graph
26 nx.draw(G, with_labels=True, node_color=[n[1]['color'] if 'color' in n[1] else 'lightblue' for n in G.nodes(data=True)])
27 plt.show()

```

### Output (List of nodes and edges, and visualizing the imported graph)

```
[('A', {'color': 'red'}), ('B', {}), ('C', {})]  
[('A', 'B', {'weight': 2.5}), ('A', 'C', {'weight': 1.0})]
```



### **Graph Analytics**

1. Degree centrality : The degree centrality for a node  $v$  is the fraction of nodes it is connected to. The degree centrality values are normalized by dividing by the maximum possible degree in a simple graph  $n-1$  where  $n$  is the number of nodes in  $G$ .
2. Betweenness centrality : Betweenness centrality of a node  $v$  is the sum of the fraction of all-pairs shortest paths that pass through  $v$ . The betweenness centrality is normalized by dividing by the total number of shortest paths.
3. Edge betweenness centrality : Betweenness centrality of a node  $e$  is the sum of the fraction of all-pairs shortest paths that pass through  $e$ . The betweenness centrality is normalized by dividing by the maximum possible number of edges in a graph  $G$ .
4. Communities can be identified using the Girvan Newman algorithm, by successively deleting the edges with the highest betweenness centrality values.

## Importing a graph from csv file and performing graph analytics

The graph in csv file:

	A	B	C
1	node1	node2	attribute
2	A	B	{}
3	A	C	{}
4	B	C	{}
5	C	D	{}
6	D	E	{}
7	D	F	{}
8	E	F	{}

Importing the graph, printing its edge list and visualizing it:

```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt

# Read edge list from CSV
df_edges = pd.read_csv('new_graph_edges.csv')

# Create a graph from the edge list
G = nx.from_pandas_edgelist(df_edges, source='node1', target='node2')

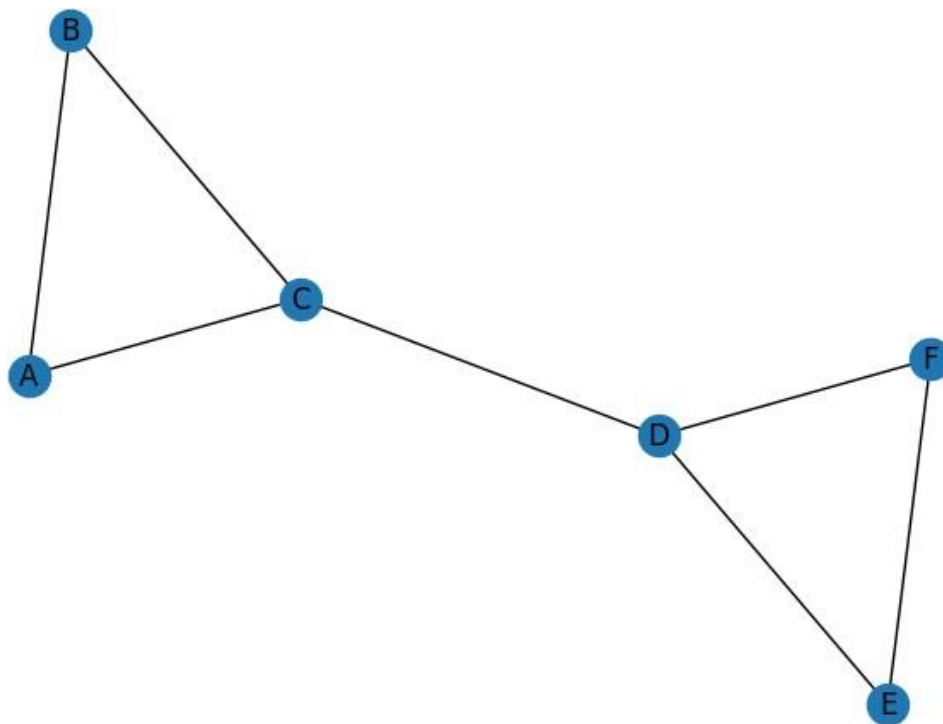
# Print the graph
print(G.nodes(data=True))
print(G.edges(data=True))

# Draw the graph
nx.draw(G, with_labels=True)
plt.show()
```

Output (graph details and visualization):

```
[('A', {}), ('B', {}), ('C', {}), ('D', {}), ('E', {}), ('F', {})]
[('A', 'B', {}), ('A', 'C', {}), ('B', 'C', {}), ('C', 'D', {}), ('D', 'E', {}), ('D', 'F', {}), ('E', 'F', {})]
```





**Performing analytics on this graph:**

```
# Basic graph properties
print("Number of nodes:", G.number_of_nodes())
print("Number of edges:", G.number_of_edges())

# Degree centrality
degrees = dict(G.degree())
print("\nDegree Centrality:", degrees)

# Betweenness centrality
betweenness = nx.betweenness_centrality(G, normalized=False)
print("\nBetweenness Centrality:", betweenness)
betweenness = nx.betweenness_centrality(G)
print("Normalized Betweenness Centrality:", betweenness)

# Closeness centrality
e_betweenness = nx.edge_betweenness_centrality(G, normalized=False)
print("\nEdge Betweenness Centrality:", e_betweenness)
```



```
e_betweenness = nx.edge_betweenness centrality(G)
print("Normalized Edge Betweenness Centrality:", e_betweenness)

# Community detection (Girvan-Newman)
communities = nx.algorithms.community.girvan_newman(G)
top_level_communities = next(communities)
print("\nCommunities level 1:", top_level_communities)
top_level_communities = next(communities)
print("\nCommunities level 2:", top_level_communities)
top_level_communities = next(communities)
print("\nCommunities level 3:", top_level_communities)
top_level_communities = next(communities)
print("\nCommunities level 4:", top_level_communities)
top_level_communities = next(communities)
print("\nCommunities level 5:", top_level_communities)
```

### Output:

Number of nodes: 6

Number of edges: 7

Degree Centrality: {'A': 2, 'B': 2, 'C': 3, 'D': 3, 'E': 2, 'F': 2}

Betweenness Centrality: {'A': 0.0, 'B': 0.0, 'C': 6.0, 'D': 6.0, 'E': 0.0, 'F': 0.0}

Normalized Betweenness Centrality: {'A': 0.0, 'B': 0.0, 'C': 0.6000000000000001, 'D': 0.6000000000000001, 'E': 0.0, 'F': 0.0}

Edge Betweenness Centrality: {('A', 'B'): 1.0, ('A', 'C'): 4.0, ('B', 'C'): 4.0, ('C', 'D'): 9.0, ('D', 'E'): 4.0, ('D', 'F'): 4.0, ('E', 'F'): 1.0}

Normalized Edge Betweenness Centrality: {('A', 'B'): 0.06666666666666667, ('A', 'C'): 0.26666666666666666, ('B', 'C'): 0.26666666666666666, ('C', 'D'): 0.6, ('D', 'E'): 0.26666666666666666, ('D', 'F'): 0.26666666666666666, ('E', 'F'): 0.06666666666666667}

Communities level 1: ({'B', 'A', 'C'}, {'E', 'D', 'F'})

Communities level 2: ({'A'}, {'B', 'C'}, {'E', 'D', 'F'})

Communities level 3: ({'A'}, {'B'}, {'C'}, {'E', 'D', 'F'})

Communities level 4: ({'A'}, {'B'}, {'C'}, {'D'}, {'E', 'F'})

Communities level 5: ({'A'}, {'B'}, {'C'}, {'D'}, {'E'}, {'F'})

**Students have to perform all the tasks illustrated above by creating a social network graph with nodes labelled with their own names and their friends' names. The graph should have at least 10 nodes. Students have to paste their code and screenshots of output and csv file below.**

Implementation details:

```
import networkx as nx

# Create an empty graph
G = nx.Graph()

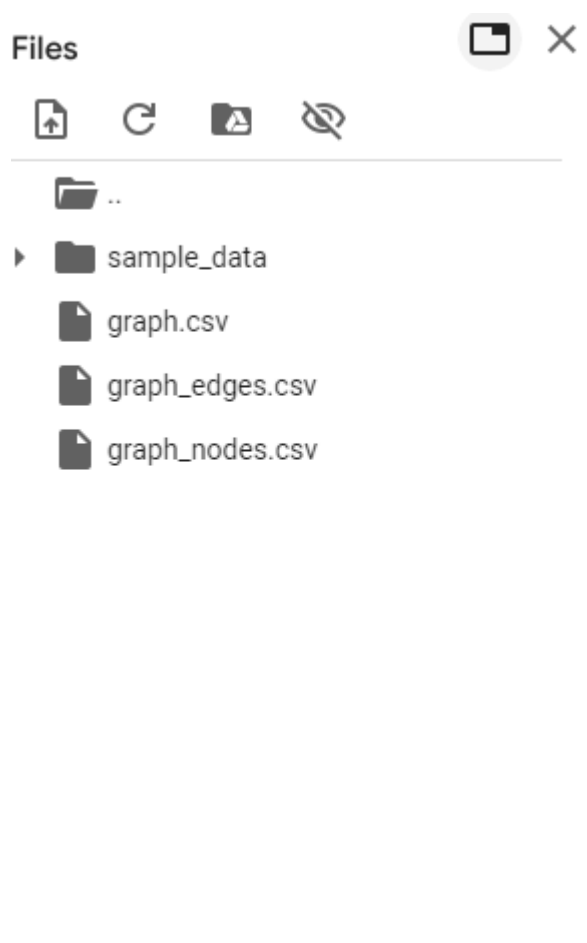
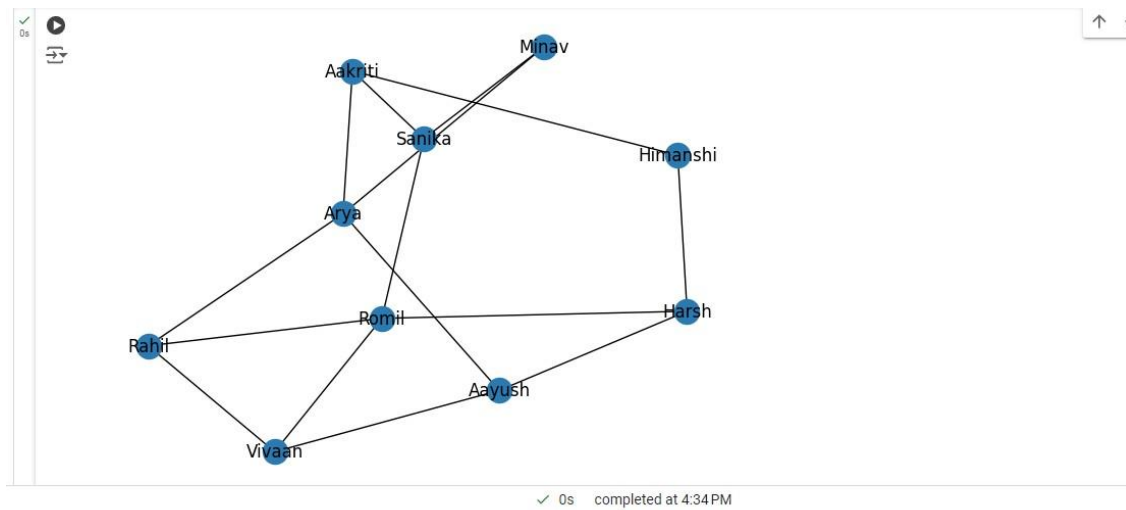
# Add nodes (individuals)
G.add_nodes_from(['Rahil', 'Romil', 'Arya', 'Vivaan', 'Minav', 'Sanika', 'Aakriti', 'Aayush', 'Himanshi', 'Harsh'])

# Add edges (relationships)
G.add_edge('Rahil', 'Romil')
G.add_edge('Rahil', 'Arya')
G.add_edge('Rahil', 'Vivaan')
G.add_edge('Romil', 'Vivaan')
G.add_edge('Romil', 'Sanika')
G.add_edge('Arya', 'Minav')
G.add_edge('Arya', 'Aakriti')
G.add_edge('Vivaan', 'Aayush')
G.add_edge('Minav', 'Sanika')
G.add_edge('Sanika', 'Aakriti')
G.add_edge('Aakriti', 'Himanshi')
G.add_edge('Aayush', 'Harsh')
G.add_edge('Himanshi', 'Harsh')
G.add_edge('Harsh', 'Romil')

G.add_edge('Sanika', 'Aakriti')
G.add_edge('Aakriti', 'Himanshi')
G.add_edge('Aayush', 'Harsh')
G.add_edge('Himanshi', 'Harsh')
G.add_edge('Harsh', 'Romil')
G.add_edge('Aayush', 'Arya')

# Print the graph
print(G.nodes())
print(G.edges())
```

['Rahil', 'Romil', 'Arya', 'Vivaan', 'Minav', 'Sanika', 'Aakriti', 'Aayush', 'Himanshi', 'Harsh']  
[('Rahil', 'Romil'), ('Rahil', 'Arya'), ('Rahil', 'Vivaan'), ('Romil', 'Vivaan'), ('Romil', 'Sanika'), ('Romil', 'Harsh'), ('Arya', 'Minav'), ('Arya', 'Aakriti'), ('Sanika', 'Aakriti'), ('Aakriti', 'Himanshi'), ('Aayush', 'Harsh'), ('Himanshi', 'Harsh'), ('Harsh', 'Romil'), ('Aayush', 'Arya')]





1 to 10 of 15 entries

source	target	weight
Rahil	Romil	{'weight': 1.5}
Rahil	Arya	{'weight': 2.0}
Rahil	Vivaan	{'weight': 1.0}
Romil	Vivaan	{'weight': 2.5}
Romil	Sanika	{'weight': 1.2}
Romil	Harsh	{'weight': 1.6}
Arya	Minav	{'weight': 3.0}
Arya	Aakriti	{'weight': 1.8}
Arya	Aayush	{'weight': 1.3}
Vivaan	Aayush	{'weight': 2.1}

Show  per page

1 entry

	color
Rahil	red

Show  per page

```
1 import pandas as pd
2 import networkx as nx
3 import matplotlib.pyplot as plt
4
5 # Read edge list from CSV
6 df_edges = pd.read_csv('graph_edges.csv')
7
8 df_nodes = pd.read_csv('graph_nodes.csv', index_col=0)
9
10 # Extract numeric weights from the 'weight' column (assuming they are stored as dictionaries)
11 # If your 'weight' column is just a number, remove this line
12 df_edges['weight'] = df_edges['weight'].apply(lambda x: float(x.strip("{}").split(": ")[1]) if isinstance(x, str) else x)
13
14 # Create a graph from the edge list
15 G = nx.from_pandas_edgelist(df_edges, source='source', target='target', edge_attr='weight')
16
17 # Add node attributes if available
18 if df_nodes is not None:
19     nx.set_node_attributes(G, df_nodes.to_dict('index'))
20
21 # Print the graph
22 print(G.nodes(data=True))
23 print(G.edges(data=True))
24
25 # Draw the graph
26 nx.draw(G, with_labels=True, node_color=[n[1]['color'] if 'color' in n[1] else 'lightblue' for n in G.nodes(data=True)])
27 plt.show()
```



Number of edges: 15

Betweenness Centrality: {'Rahil': 2.1666666666666665, 'Romil': 7.166666666666667, 'Arya': 8.333333333333332, 'Vivaan': 1.25, 'Sanika': 4.166666666666666, 'Harsh': 4.25, 'Minav': 0.75, 'Aakriti': 4.25, 'Aayush': 3.666666666666666, 'Himanshi': 1.0}

Edge Betweenness Centrality: {'Rahil', 'Romil': 4.5, ('Rahil', 'Arya'): 6.166666666666667, ('Rahil', 'Vivaan'): 2.6666666666666666, ('Romil', 'Vivaan'): 4.4166666666666666, ('Romil',



'Sanika'): 7.666666666666667, ('Romil', 'Harsh'): 6.75, ('Arya', 'Minav'): 5.916666666666667, ('Arya', 'Aakriti'): 6.916666666666667, ('Arya', 'Aayush'): 6.666666666666667, ('Vivaan', 'Aayush'): 4.416666666666667, ('Sanika', 'Minav'): 4.583333333333333, ('Sanika', 'Aakriti'): 5.083333333333333, ('Harsh', 'Aayush'): 5.25, ('Harsh', 'Himanshi'): 5.5, ('Aakriti', 'Himanshi'): 5.5}

Normalized Edge Betweenness Centrality: ({'Rahil', 'Romil'): 0.1, ('Rahil', 'Arya'): 0.13703703703703704, ('Rahil', 'Vivaan'): 0.05925925925925925, ('Romil', 'Vivaan'): 0.09814814814814814, ('Romil', 'Sanika'): 0.1703703703703704, ('Romil', 'Harsh'): 0.15, ('Arya', 'Minav'): 0.13148148148148148, ('Arya', 'Aakriti'): 0.1537037037037037, ('Arya', 'Aayush'): 0.14814814814814814, ('Vivaan', 'Aayush'): 0.09814814814814814, ('Sanika', 'Minav'): 0.10185185185185185, ('Sanika', 'Aakriti'): 0.11296296296296296, ('Harsh', 'Aayush'): 0.11666666666666667, ('Harsh', 'Himanshi'): 0.12222222222222223, ('Aakriti', 'Himanshi'): 0.12222222222222223}

Communities after 1 step: ({'Harsh', 'Aayush', 'Vivaan', 'Rahil', 'Romil'}, {'Aakriti', 'Arya', 'Sanika', 'Minav', 'Himanshi'})

Communities after 2 steps: ({'Harsh', 'Aayush', 'Vivaan', 'Rahil', 'Romil'}, {'Minav', 'Arya', 'Sanika', 'Aakriti'}, {'Himanshi'})

Communities after 3 steps: ({'Vivaan', 'Romil', 'Rahil'}, {'Minav', 'Arya', 'Sanika', 'Aakriti'}, {'Harsh', 'Aayush'}, {'Himanshi'})

Communities after 4 steps: ({'Vivaan', 'Romil', 'Rahil'}, {'Aakriti', 'Arya'}, {'Minav', 'Sanika'}, {'Harsh', 'Aayush'}, {'Himanshi'})

Communities after 5 steps: ({'Rahil'}, {'Vivaan', 'Romil'}, {'Aakriti', 'Arya'}, {'Minav', 'Sanika'}, {'Harsh', 'Aayush'}, {'Himanshi'})

**Date:** \_\_\_\_\_

**Signature of faculty in-charge**

### **Post Lab Descriptive Questions:**

1. Analyze the centrality measures you calculated. Which nodes were identified as the most influential? What does this mean in the context of the social network?

### **Centrality Measures and Influential Nodes**

Centrality measures help us understand the importance or influence of nodes within a network. Here's how various centrality measures are typically analyzed:

- **Degree Centrality:** This measure counts the number of direct connections a node has. Nodes with high degree centrality are often considered influential because they have more direct connections to other nodes.
- **Closeness Centrality:** This measure evaluates how quickly a node can access other nodes in the network. Nodes with high closeness centrality are often central and can spread information quickly throughout the network.
- **Betweenness Centrality:** This measure identifies nodes that act as bridges between other nodes. Nodes with high betweenness centrality are important for the flow of information or resources across the network.
- **Eigenvector Centrality:** This measure considers not only the number of connections a node has but also the quality of those connections. Nodes connected to other well-connected nodes are considered more influential.

#### **Interpretation:**

- **High Degree Centrality Nodes:** These nodes have many direct connections, indicating they are well-connected and potentially influential within their immediate network.
- **High Closeness Centrality Nodes:** These nodes can reach other nodes quickly, suggesting they are central to the network and can efficiently disseminate information.
- **High Betweenness Centrality Nodes:** These nodes control the flow of information between other nodes, making them crucial for network connectivity and communication.
- **High Eigenvector Centrality Nodes:** These nodes are connected to other influential nodes, indicating they hold substantial influence in the network.

2. Describe the communities identified using the Girvan-Newman algorithm. What are the characteristics of these communities? How do they relate to the social network's structure?

The Girvan-Newman algorithm detects communities by progressively removing edges with the highest betweenness centrality. Communities are groups of nodes that are more densely connected internally than with the rest of the network.

#### **Characteristics and Structure:**

- **Community Structure:** The resulting communities represent subgroups within the network where nodes are more interconnected with each other than with nodes outside their community.



- **Relation to Social Network:** In a social network context, these communities might represent clusters of people who interact more frequently or share common interests. The structure can reveal groups or social circles within the network.

3. Discuss the implications of identifying influential nodes in the network. How can this information be used?

#### **Strategic Importance:**

- **Information Dissemination:** Influential nodes are key for spreading information quickly across the network. Understanding who these nodes are can help in designing effective communication strategies.
- **Resource Allocation:** In business or organizational contexts, resources can be allocated to or through influential nodes to maximize impact.
- **Marketing and Outreach:** For marketing campaigns, targeting influential nodes can increase the reach and effectiveness of the campaign.
- **Network Resilience:** Identifying influential nodes can also help in understanding network resilience. If these nodes are removed, it might affect the network's connectivity significantly.

#### **Applications:**

- **Social Media:** Influential users (e.g., influencers) can drive trends and influence public opinion.
- **Public Health:** Identifying key individuals can help in the rapid dissemination of health information.
- **Organizational Management:** Recognizing central figures in an organization can aid in improving communication and collaboration.