Batch: HDA2	**Roll No.: 16010123012**
Experiment No.: 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.

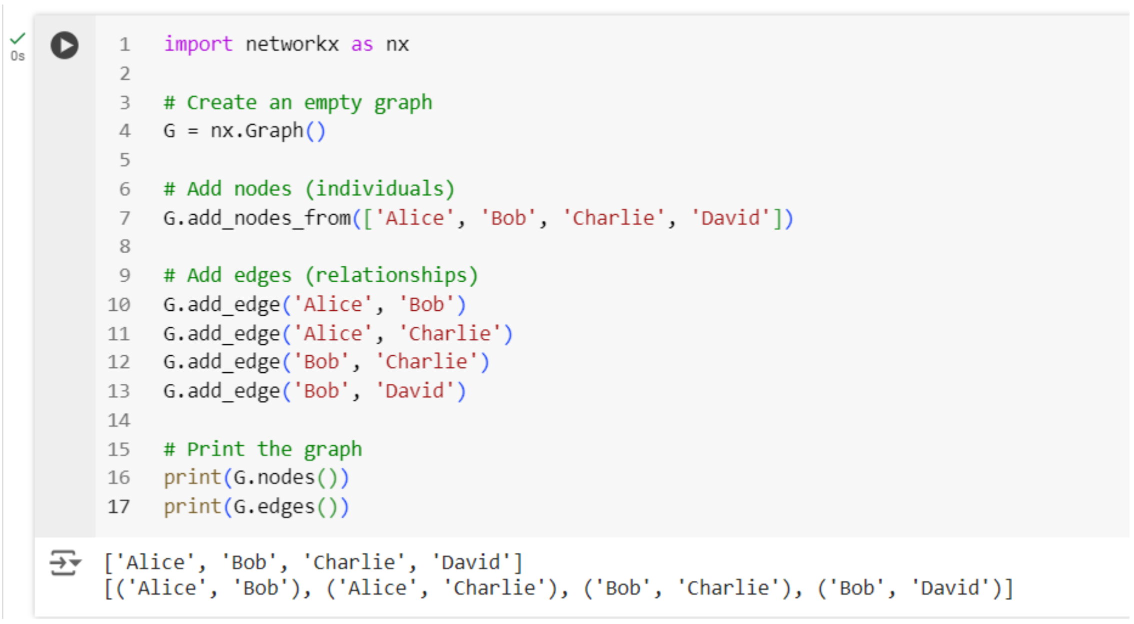
Expected OUTCOME of Experiment:
CO3: Perform the social data analytics

Books/ Journals/ Websites referred:
https://www.analyticsvidhya.com/blog/2018/04/introduction-to-graph-theory-network-analysis-python-codes/

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:

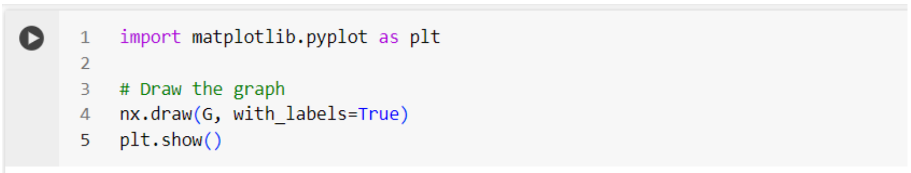
Building a Social Network Graph with NetworkX



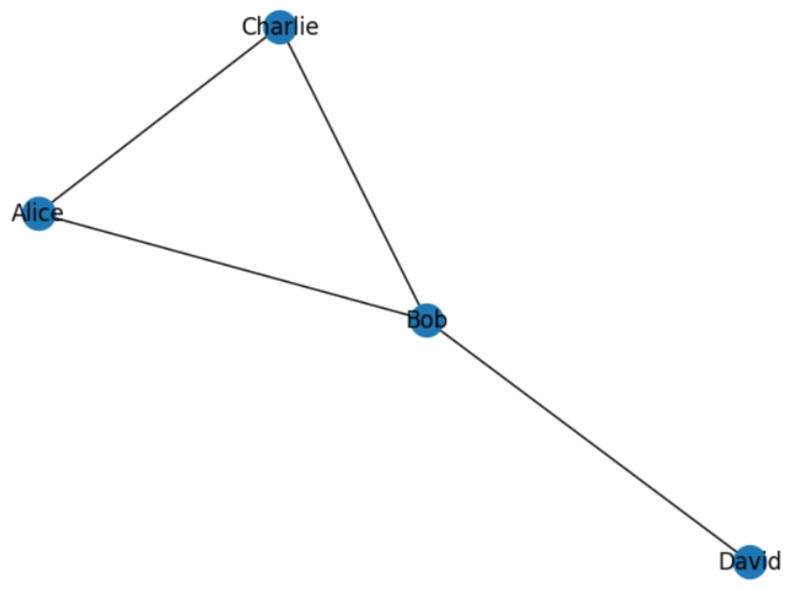
```python
import networkx as nx

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

# Add nodes (individuals)
G.add_nodes_from(['Alice', 'Bob', 'Charlie', 'David'])

# Add edges (relationships)
G.add_edge('Alice', 'Bob')
G.add_edge('Alice', 'Charlie')
G.add_edge('Bob', 'Charlie')
G.add_edge('Bob', 'David')

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

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

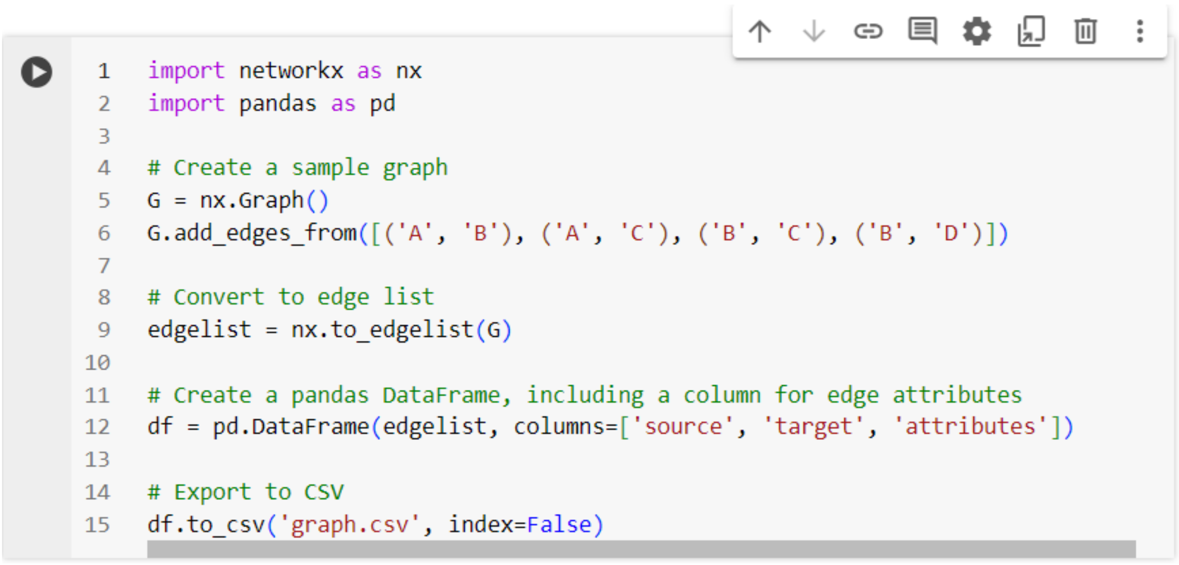
Visualizing the Graph



```python
import matplotlib.pyplot as plt

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

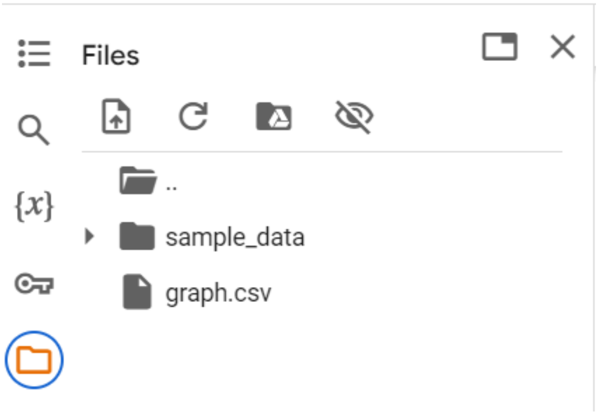


Exporting a NetworkX Graph to CSV

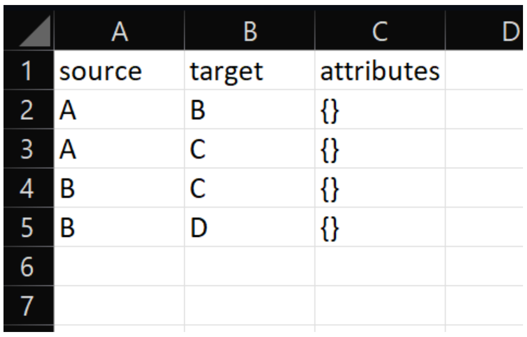


```python
import networkx as nx
import pandas as pd

# Create a sample graph
G = nx.Graph()
G.add_edges_from([('A', 'B'), ('A', 'C'), ('B', 'C'), ('B', 'D')])

# Convert to edge list
edgelist = nx.to_edgelist(G)

# Create a pandas DataFrame, including a column for edge attributes
df = pd.DataFrame(edgelist, columns=['source', 'target', 'attributes'])

# Export to CSV
df.to_csv('graph.csv', index=False)
```

The csv file gets created

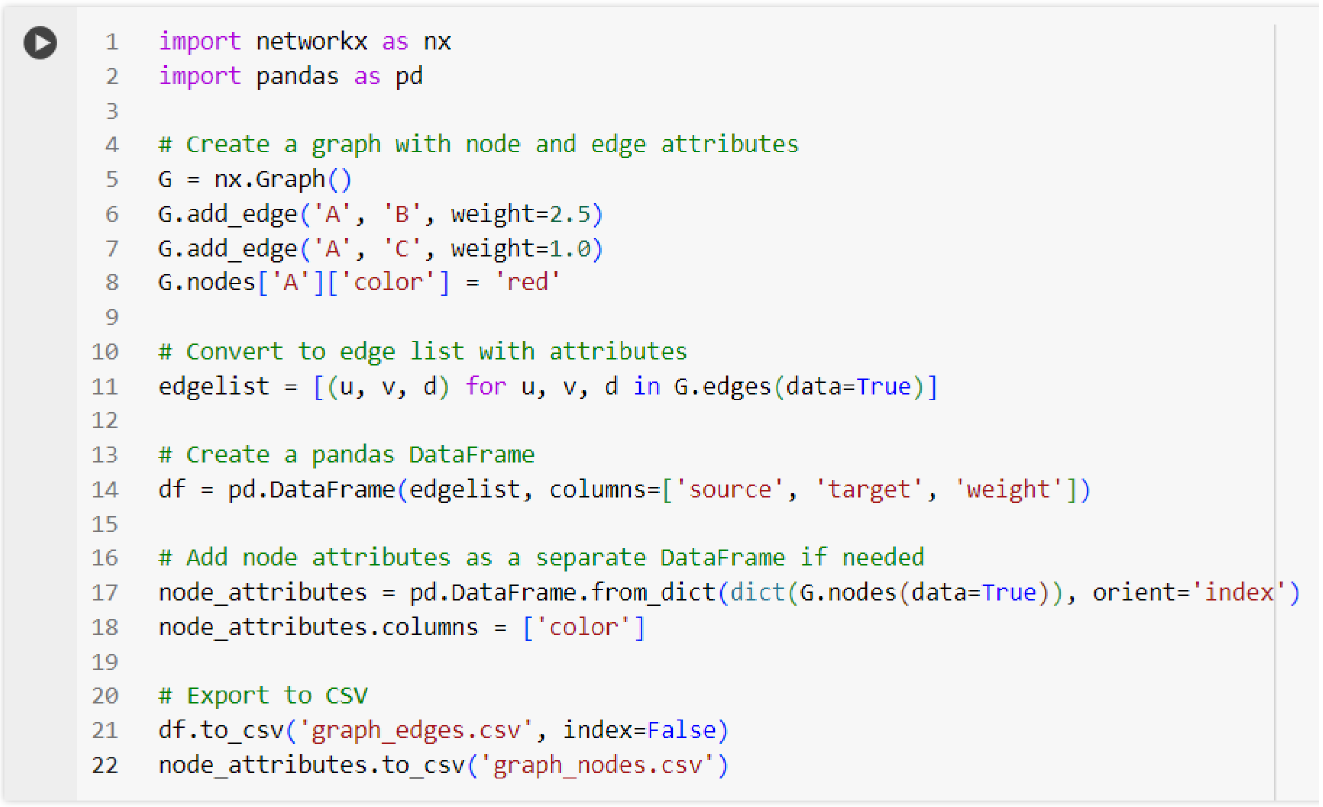


Contents of the csv file



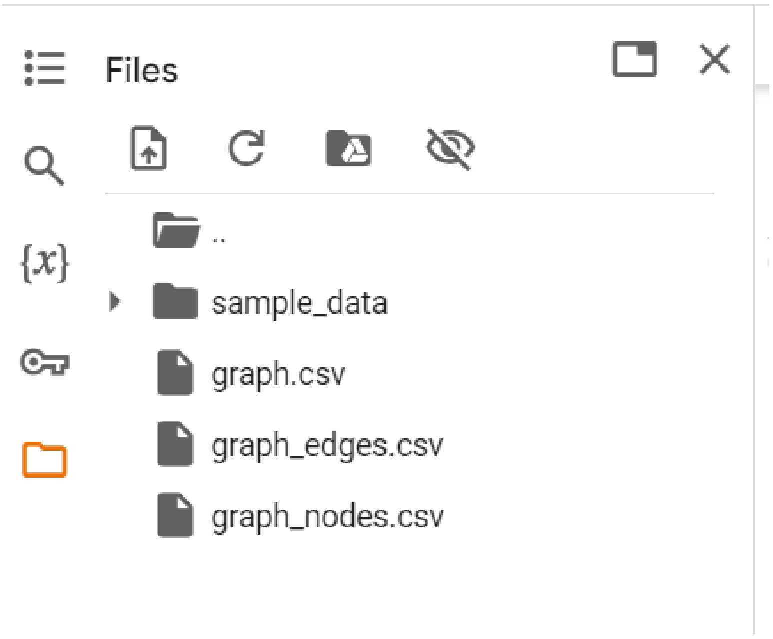
	A	B	C	D
1	source	target	attributes	
2	A	B	{}	
3	A	C	{}	
4	B	C	{}	
5	B	D	{}	
6				
7				

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

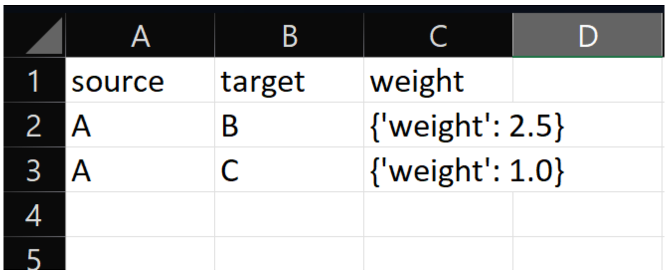


```python
import networkx as nx
import pandas as pd

# Create a graph with node and edge attributes
G = nx.Graph()
G.add_edge('A', 'B', weight=2.5)
G.add_edge('A', 'C', weight=1.0)
G.nodes['A']['color'] = 'red'

# Convert to edge list with attributes
edgelist = [(u, v, d) for u, v, d in G.edges(data=True)]

# Create a pandas DataFrame
df = pd.DataFrame(edgelist, columns=['source', 'target', 'weight'])

# Add node attributes as a separate DataFrame if needed
node_attributes = pd.DataFrame.from_dict(dict(G.nodes(data=True)), orient='index')
node_attributes.columns = ['color']

# Export to CSV
df.to_csv('graph_edges.csv', index=False)
node_attributes.to_csv('graph_nodes.csv')
```

csv files get created

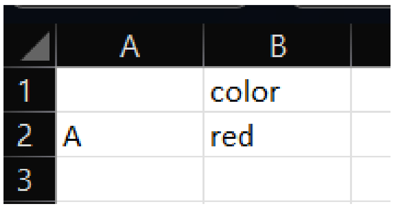


Contents of graph_edges.csv



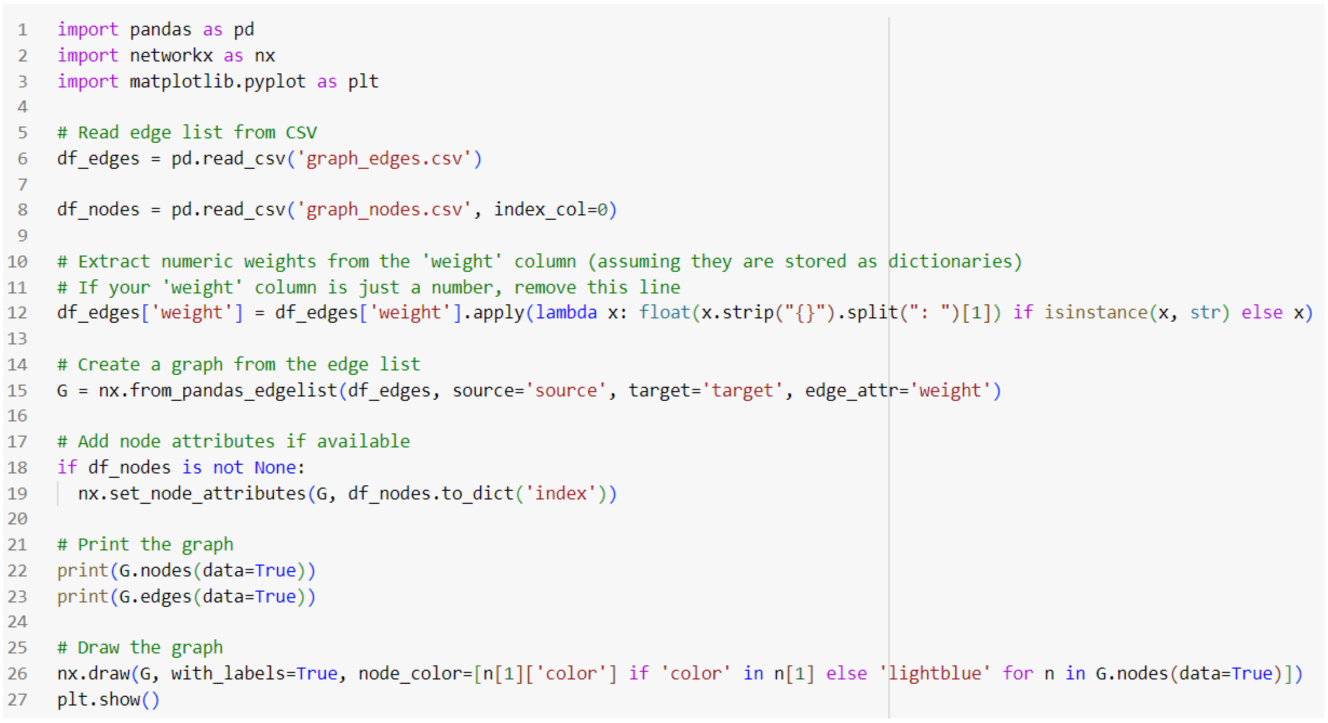
	A	B	C	D
1	source	target	weight	
2	A	B	{'weight': 2.5}	
3	A	C	{'weight': 1.0}	
4				
5				

Contents of graph_nodes.csv



	A	B
1		color
2	A	red
3		

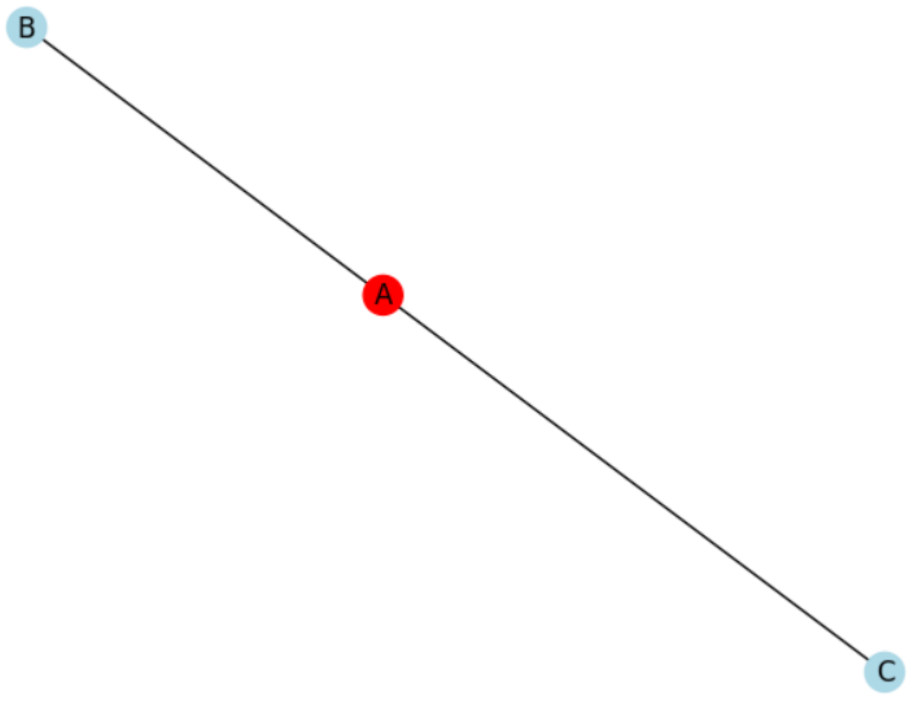
Importing a graph from a csv file



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

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

df_nodes = pd.read_csv('graph_nodes.csv', index_col=0)

# Extract numeric weights from the 'weight' column (assuming they are stored as dictionaries)
# If your 'weight' column is just a number, remove this line
df_edges['weight'] = df_edges['weight'].apply(lambda x: float(x.strip("{}").split(": ")[1]) if isinstance(x, str) else x)

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

# Add node attributes if available
if df_nodes is not None:
    nx.set_node_attributes(G, df_nodes.to_dict('index'))

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

# Draw the graph
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)])
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})]
```

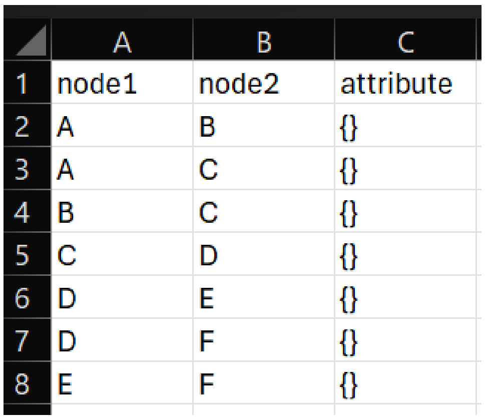


<u>Graph Analytics</u>

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:



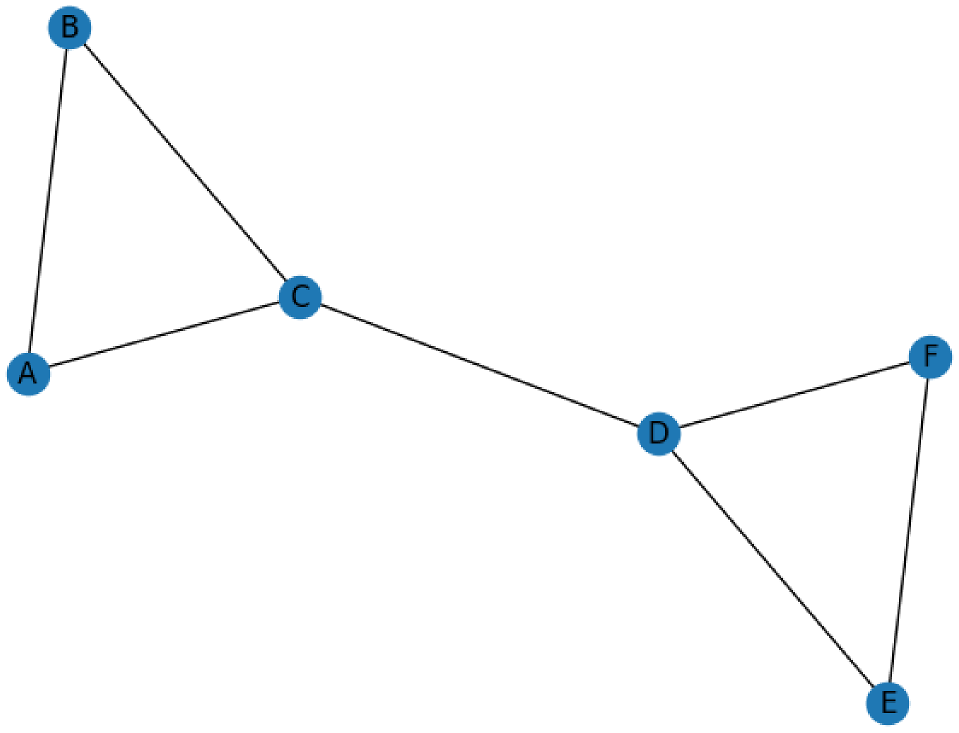
	A	B	C
1	node1	node2	attribute
2	A	B	{}
3	A	C	{}
4	B	C	{}
5	C	D	{}
6	D	E	{}
7	D	F	{}
8	E	F	{}

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

```python
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_betwenness = nx.edge_betweenness_centrality(G,normalized=False)
print("\nEdge Betweenness Centrality:", e_betwenness)
e_betwenness = nx.edge_betweenness_centrality(G)
print("Normalized Edge Betweenness Centrality:", e_betwenness)

# Community detection (Girvan-Newman)
communities = nx.algorithms.community.girvan_newman(G)
```

```
try:
    top_level_communities = next(communities)
    print("\nCommunities after 1 step:", top_level_communities)

    top_level_communities = next(communities)
    print("\nCommunities after 2 steps:", top_level_communities)

    top_level_communities = next(communities)
    print("\nCommunities after 3 steps:", top_level_communities)

    top_level_communities = next(communities)
    print("\nCommunities after 4 steps:", top_level_communities)

    top_level_communities = next(communities)
    print("\nCommunities after 5 steps:", top_level_communities)

except StopIteration:
    print("\nNo more splits are possible.")
```

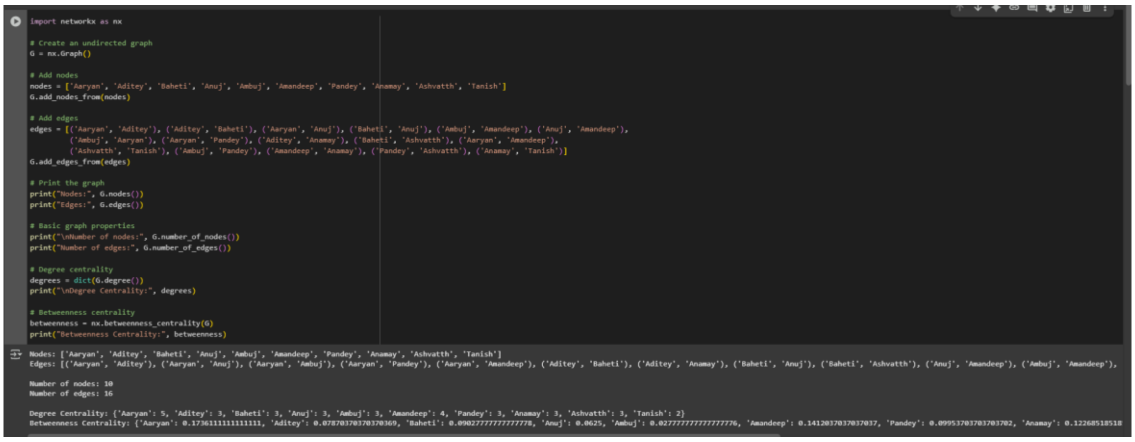
Output:

```
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 after 1 step: ({'A', 'C', 'B'}, {'E', 'F', 'D'})

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

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

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

Communities after 5 steps: ({'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:

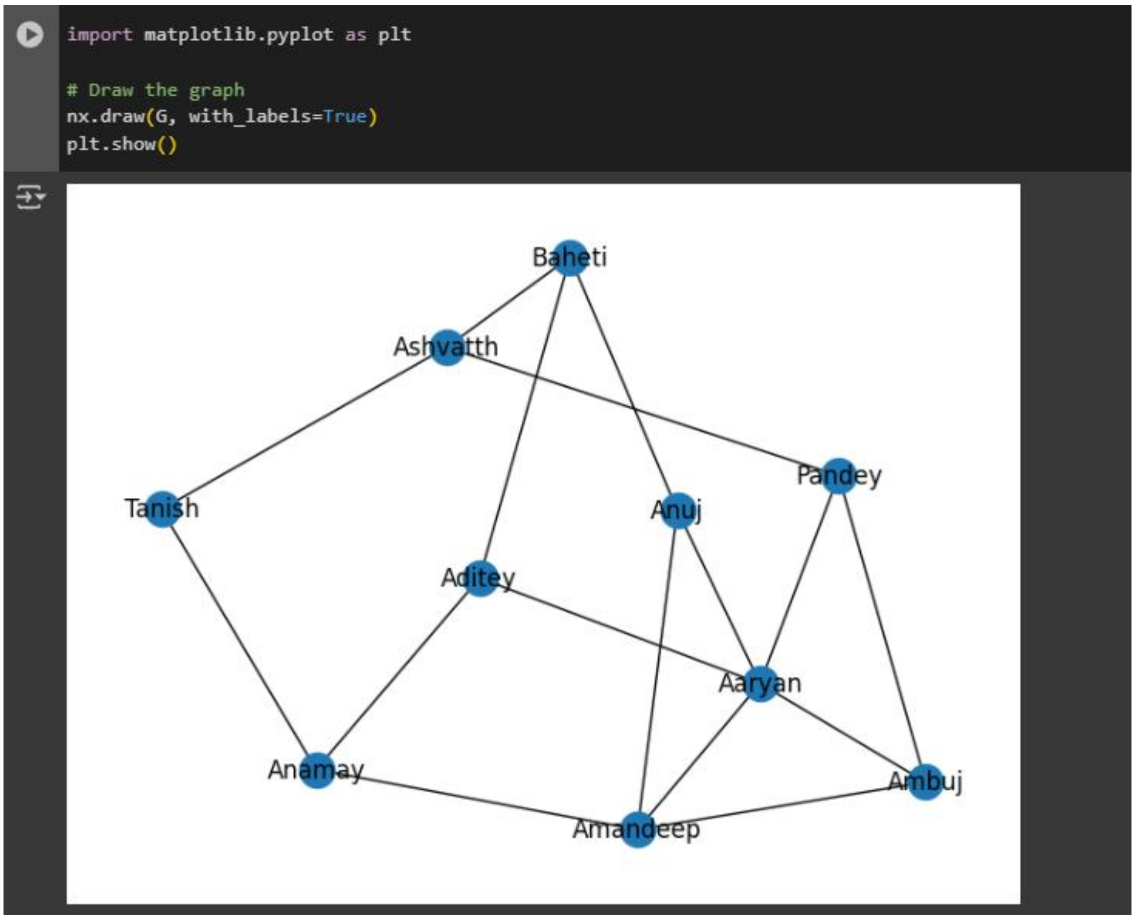
Building a Social Network Graph with NetworkX



```
import networkx as nx

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

# Add nodes
nodes = ['Aaryan', 'Aditey', 'Baheti', 'Anuj', 'Ambuj', 'Amandeep', 'Pandey', 'Anamay', 'Ashvatth', 'Tanish']
G.add_nodes_from(nodes)

# Add edges
edges = [('Aaryan', 'Aditey'), ('Aditey', 'Baheti'), ('Aaryan', 'Anuj'), ('Baheti', 'Anuj'), ('Ambuj', 'Amandeep'), ('Anuj', 'Amandeep'),
         ('Ambuj', 'Aaryan'), ('Aaryan', 'Pandey'), ('Aditey', 'Anamay'), ('Baheti', 'Ashvatth'), ('Aaryan', 'Amandeep'),
         ('Ashvatth', 'Tanish'), ('Ambuj', 'Pandey'), ('Amandeep', 'Anamay'), ('Pandey', 'Ashvatth'), ('Anamay', 'Tanish')]
G.add_edges_from(edges)

# Print the graph
print("Nodes:", G.nodes())
print("Edges:", G.edges())

# Basic graph properties
print("\nNumber 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)
print("Betweenness Centrality:", betweenness)
```

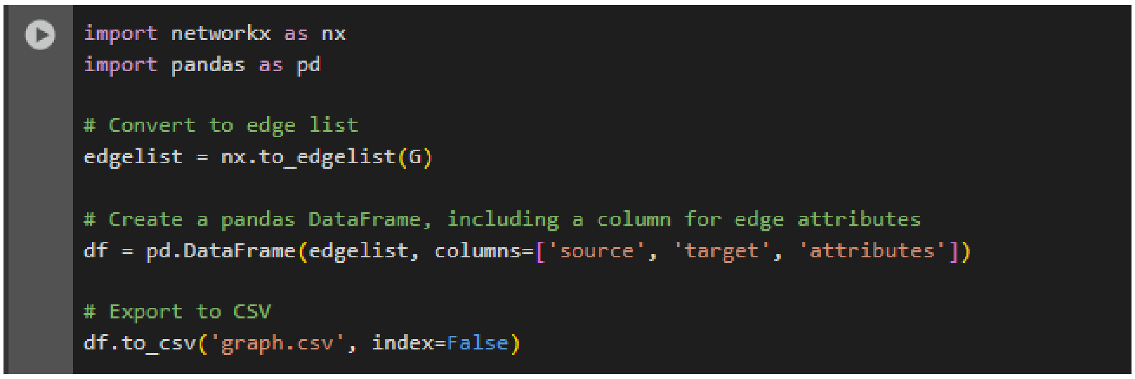
```
Nodes: ['Aaryan', 'Aditey', 'Baheti', 'Anuj', 'Ambuj', 'Amandeep', 'Pandey', 'Anamay', 'Ashvatth', 'Tanish']
Edges: [('Aaryan', 'Aditey'), ('Aaryan', 'Anuj'), ('Aaryan', 'Ambuj'), ('Aaryan', 'Pandey'), ('Aaryan', 'Amandeep'), ('Aditey', 'Baheti'), ('Aditey', 'Anamay'), ('Baheti', 'Anuj'), ('Baheti', 'Ashvatth'), ('Anuj', 'Amandeep'), ('Ambuj', 'Amandeep'),

Number of nodes: 10
Number of edges: 16

Degree Centrality: {'Aaryan': 5, 'Aditey': 3, 'Baheti': 3, 'Anuj': 3, 'Ambuj': 3, 'Amandeep': 4, 'Pandey': 3, 'Anamay': 3, 'Ashvatth': 3, 'Tanish': 2}
Betweenness Centrality: {'Aaryan': 0.1736111111111111, 'Aditey': 0.07870370370370369, 'Baheti': 0.09027777777777778, 'Anuj': 0.0625, 'Ambuj': 0.027777777777777776, 'Amandeep': 0.1412037037037037, 'Pandey': 0.09953703703703702, 'Anamay': 0.12268518518
```

Visualizing the Graph



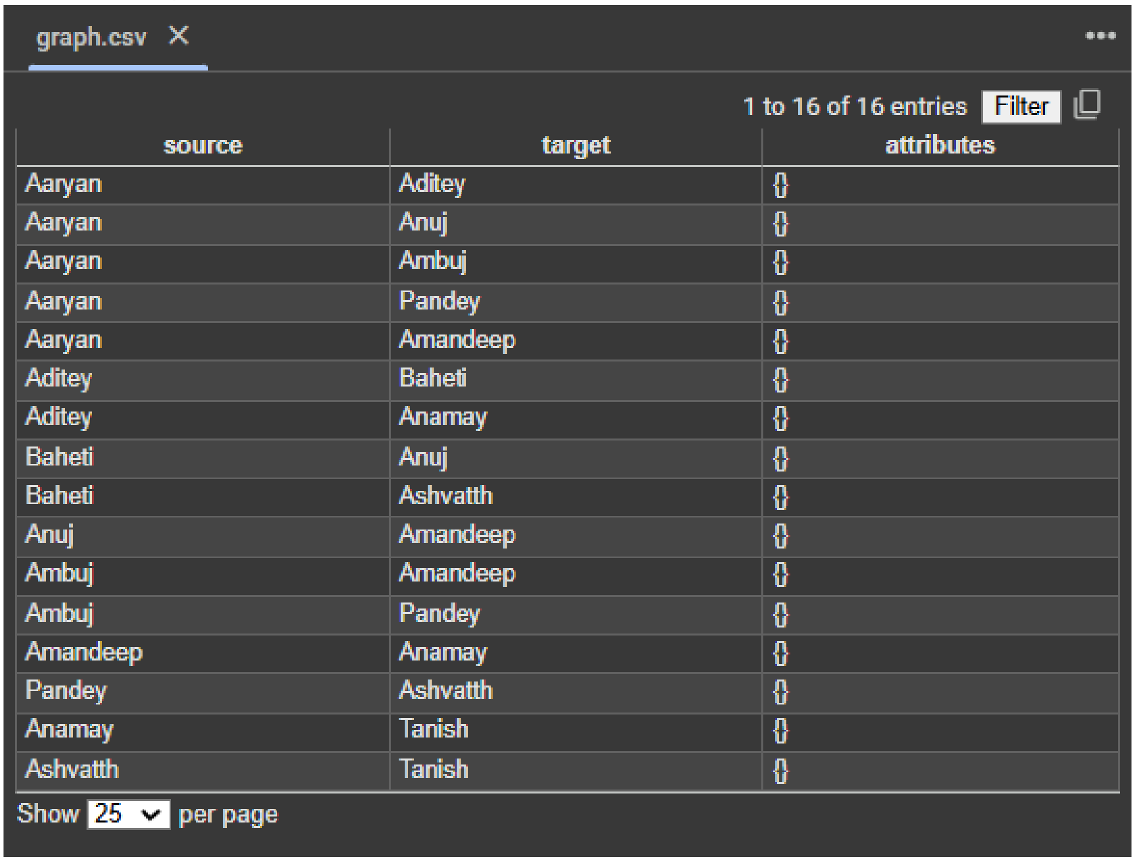
```
import matplotlib.pyplot as plt

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

Exporting a NetworkX Graph to CSV



```
import networkx as nx
import pandas as pd

# Convert to edge list
edgelist = nx.to_edgelist(G)

# Create a pandas DataFrame, including a column for edge attributes
df = pd.DataFrame(edgelist, columns=['source', 'target', 'attributes'])

# Export to CSV
df.to_csv('graph.csv', index=False)
```

The csv file gets created and Contents of the csv file



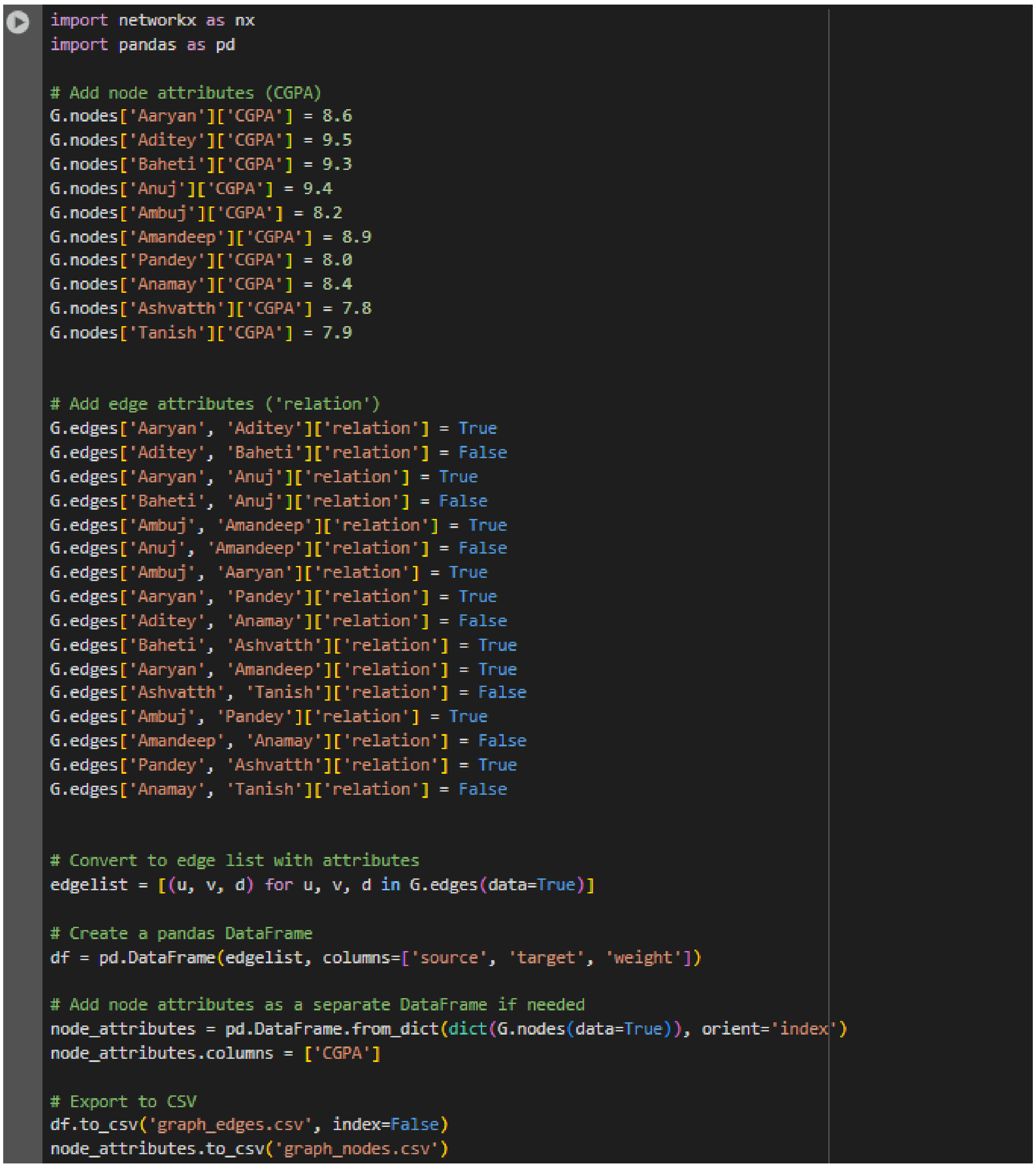
graph.csv

1 to 16 of 16 entries

source	target	attributes
Aaryan	Aditey	{}
Aaryan	Anuj	{}
Aaryan	Ambuj	{}
Aaryan	Pandey	{}
Aaryan	Amandeep	{}
Aditey	Baheti	{}
Aditey	Anamay	{}
Baheti	Anuj	{}
Baheti	Ashvatth	{}
Anuj	Amandeep	{}
Ambuj	Amandeep	{}
Ambuj	Pandey	{}
Amandeep	Anamay	{}
Pandey	Ashvatth	{}
Anamay	Tanish	{}
Ashvatth	Tanish	{}

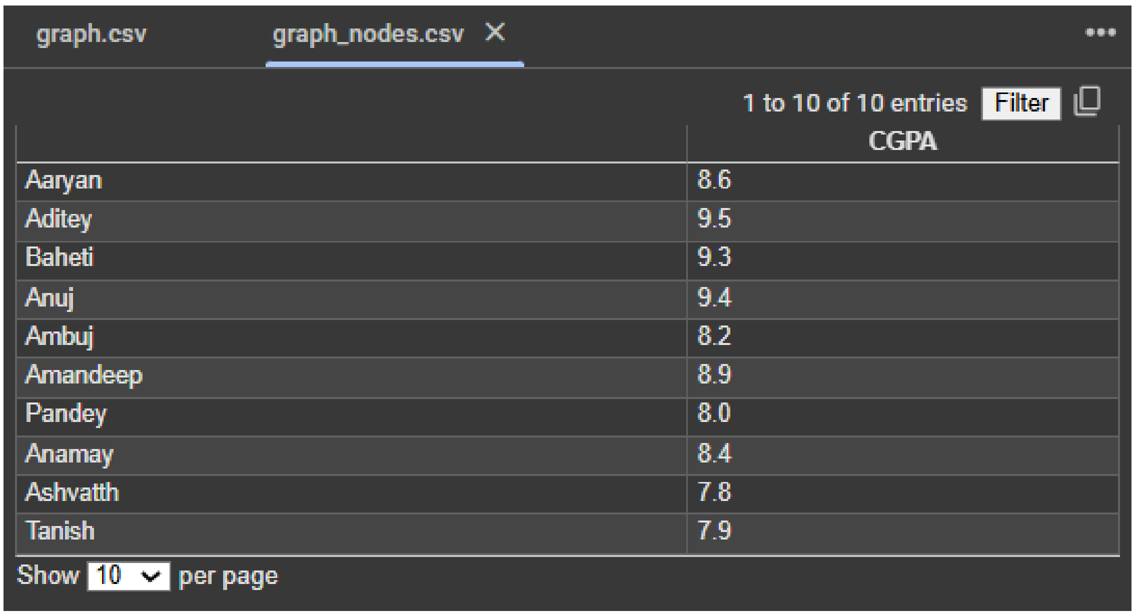
Show 25 per page

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



```python
import networkx as nx
import pandas as pd

# Add node attributes (CGPA)
G.nodes['Aaryan']['CGPA'] = 8.6
G.nodes['Aditey']['CGPA'] = 9.5
G.nodes['Baheti']['CGPA'] = 9.3
G.nodes['Anuj']['CGPA'] = 9.4
G.nodes['Ambuj']['CGPA'] = 8.2
G.nodes['Amandeep']['CGPA'] = 8.9
G.nodes['Pandey']['CGPA'] = 8.0
G.nodes['Anamay']['CGPA'] = 8.4
G.nodes['Ashvatth']['CGPA'] = 7.8
G.nodes['Tanish']['CGPA'] = 7.9


# Add edge attributes ('relation')
G.edges['Aaryan', 'Aditey']['relation'] = True
G.edges['Aditey', 'Baheti']['relation'] = False
G.edges['Aaryan', 'Anuj']['relation'] = True
G.edges['Baheti', 'Anuj']['relation'] = False
G.edges['Ambuj', 'Amandeep']['relation'] = True
G.edges['Anuj', 'Amandeep']['relation'] = False
G.edges['Ambuj', 'Aaryan']['relation'] = True
G.edges['Aaryan', 'Pandey']['relation'] = True
G.edges['Aditey', 'Anamay']['relation'] = False
G.edges['Baheti', 'Ashvatth']['relation'] = True
G.edges['Aaryan', 'Amandeep']['relation'] = True
G.edges['Ashvatth', 'Tanish']['relation'] = False
G.edges['Ambuj', 'Pandey']['relation'] = True
G.edges['Amandeep', 'Anamay']['relation'] = False
G.edges['Pandey', 'Ashvatth']['relation'] = True
G.edges['Anamay', 'Tanish']['relation'] = False


# Convert to edge list with attributes
edgelist = [(u, v, d) for u, v, d in G.edges(data=True)]

# Create a pandas DataFrame
df = pd.DataFrame(edgelist, columns=['source', 'target', 'weight'])

# Add node attributes as a separate DataFrame if needed
node_attributes = pd.DataFrame.from_dict(dict(G.nodes(data=True)), orient='index')
node_attributes.columns = ['CGPA']

# Export to CSV
df.to_csv('graph_edges.csv', index=False)
node_attributes.to_csv('graph_nodes.csv')
```

Contents of graph_nodes.csv



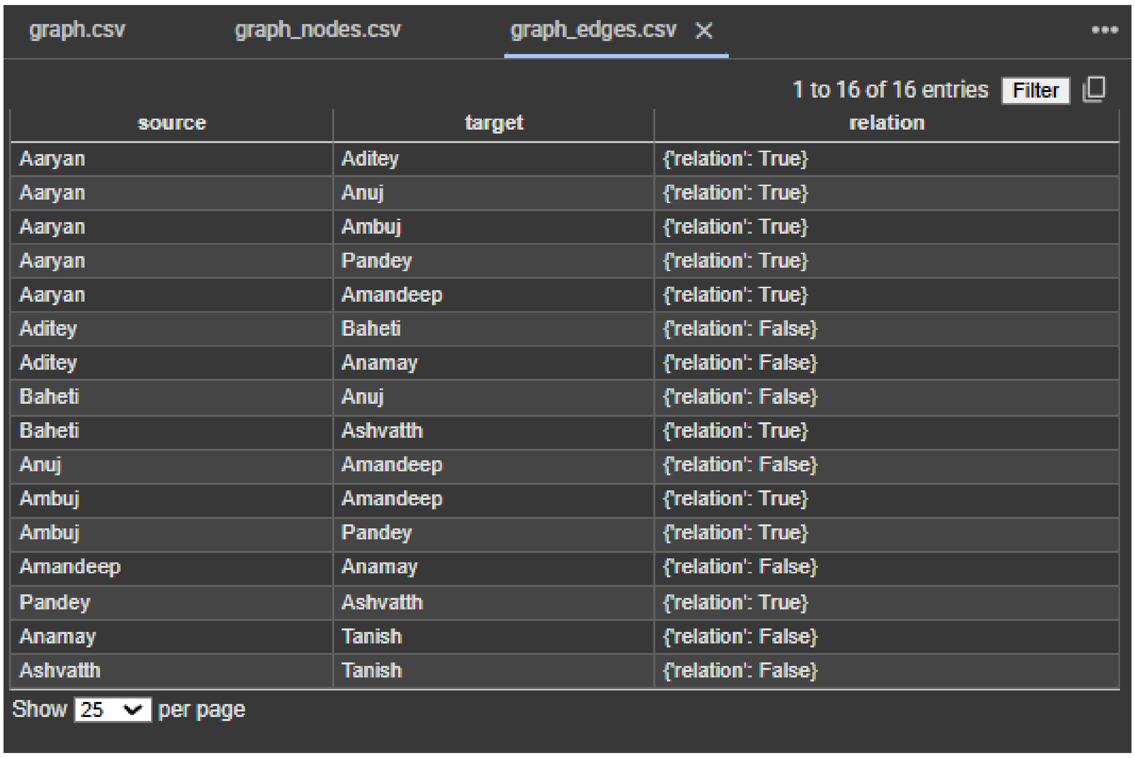
graph.csv | graph_nodes.csv

1 to 10 of 10 entries Filter

	CGPA
Aaryan	8.6
Aditey	9.5
Baheti	9.3
Anuj	9.4
Ambuj	8.2
Amandeep	8.9
Pandey	8.0
Anamay	8.4
Ashvatth	7.8
Tanish	7.9

Show 10 per page

Contents of graph_edges.csv



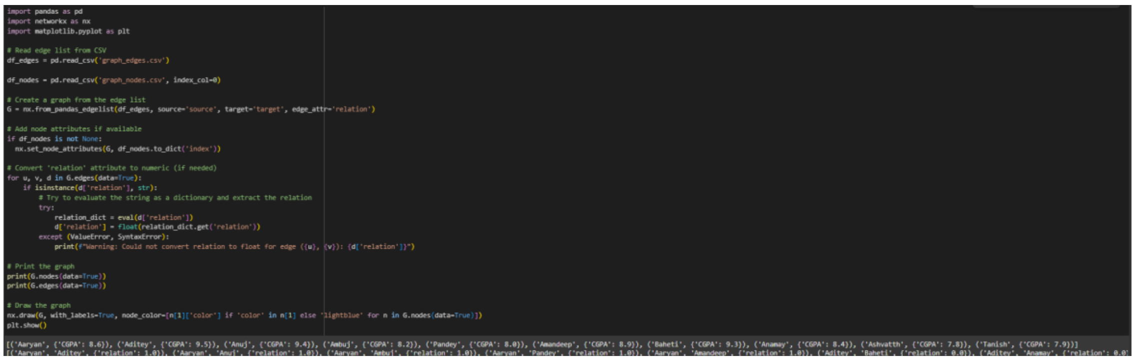
graph.csv | graph_nodes.csv | graph_edges.csv

1 to 16 of 16 entries Filter

source	target	relation
Aaryan	Aditey	{'relation': True}
Aaryan	Anuj	{'relation': True}
Aaryan	Ambuj	{'relation': True}
Aaryan	Pandey	{'relation': True}
Aaryan	Amandeep	{'relation': True}
Aditey	Baheti	{'relation': False}
Aditey	Anamay	{'relation': False}
Baheti	Anuj	{'relation': False}
Baheti	Ashvatth	{'relation': True}
Anuj	Amandeep	{'relation': False}
Ambuj	Amandeep	{'relation': True}
Ambuj	Pandey	{'relation': True}
Amandeep	Anamay	{'relation': False}
Pandey	Ashvatth	{'relation': True}
Anamay	Tanish	{'relation': False}
Ashvatth	Tanish	{'relation': False}

Show 25 per page

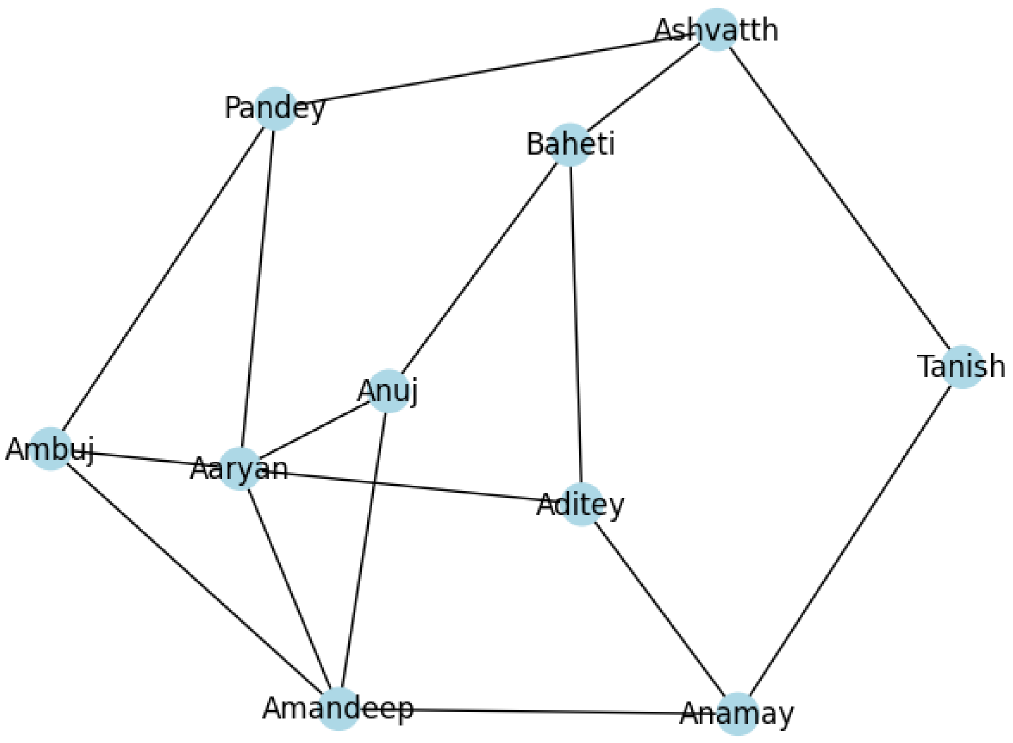
Importing a graph from a csv file



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

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

df_nodes = pd.read_csv('graph_nodes.csv', index_col=0)

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

# Add node attributes if available
if df_nodes is not None:
  nx.set_node_attributes(G, df_nodes.to_dict('index'))

# Convert 'relation' attribute to numeric (if needed)
for u, v, d in G.edges(data=True):
    if isinstance(d['relation'], str):
        # Try to evaluate the string as a dictionary and extract the relation
        try:
            relation_dict = eval(d['relation'])
            d['relation'] = float(relation_dict.get('relation'))
        except (ValueError, SyntaxError):
            print(f"Warning: Could not convert relation to float for edge ({u}, {v}): {d['relation']}")

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

# Draw the graph
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)])
plt.show()
```

```
[('Aaryan', {'CGPA': 8.6}), ('Aditey', {'CGPA': 9.5}), ('Anuj', {'CGPA': 9.4}), ('Ambuj', {'CGPA': 8.2}), ('Pandey', {'CGPA': 8.0}), ('Amandeep', {'CGPA': 8.9}), ('Baheti', {'CGPA': 9.3}), ('Anamay', {'CGPA': 8.4}), ('Ashvatth', {'CGPA': 7.8}), ('Tanish', {'CGPA': 7.9})]
[('Aaryan', 'Aditey', {'relation': 1.0}), ('Aaryan', 'Anuj', {'relation': 1.0}), ('Aaryan', 'Ambuj', {'relation': 1.0}), ('Aaryan', 'Pandey', {'relation': 1.0}), ('Aaryan', 'Amandeep', {'relation': 1.0}), ('Aditey', 'Baheti', {'relation': 0.0}), ('Aditey', 'Anamay', {'relation': 0.0
```

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



Graph Analytics:

```
# 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
```

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

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

# Community detection (Girvan-Newman)
communities = nx.algorithms.community.girvan_newman(G)
try:
    top_level_communities = next(communities)
    print("\nCommunities after 1 step:", top_level_communities)

    top_level_communities = next(communities)
    print("\nCommunities after 2 steps:", top_level_communities)

    top_level_communities = next(communities)
    print("\nCommunities after 3 steps:", top_level_communities)

    top_level_communities = next(communities)
    print("\nCommunities after 4 steps:", top_level_communities)

    top_level_communities = next(communities)
    print("\nCommunities after 5 steps:", top_level_communities)

except StopIteration:
    print("\nNo more splits are possible.")
```

```
Number of nodes: 10
Number of edges: 16

Degree Centrality: {'Aaryan': 5, 'Aditey': 3, 'Anuj': 3, 'Ambuj': 3, 'Pandey': 3, 'Amandeep': 4, 'Baheti': 3, 'Anamay': 3, 'Ashvatth': 3, 'Tanish': 2}

Betweenness Centrality: {'Aaryan': 6.25, 'Aditey': 2.833333333333333, 'Anuj': 2.25, 'Ambuj': 1.0, 'Pandey': 3.583333333333333, 'Amandeep': 5.083333333333333, 'Baheti': 3.25, 'Anamay': 4.416666666666666, 'Ashvatth': 4.833333333333333, 'Tanish': 1.5}
Normalized Betweenness Centrality: {'Aaryan': 0.1736111111111111, 'Aditey': 0.07870370370370369, 'Anuj': 0.0625, 'Ambuj': 0.027777777777777776, 'Pandey':
```

```
0.09953703703703702, 'Amandeep': 0.1412037037037037, 'Baheti': 0.09027777777777778, 'Anamay': 0.12268518518518516, 'Ashvatth': 0.13425925925925924, 'Tanish': 0.041666666666666664}

Edge Betweenness Centrality: {('Aaryan', 'Aditey'): 5.833333333333333, ('Aaryan', 'Anuj'): 3.75, ('Aaryan', 'Ambuj'): 3.0, ('Aaryan', 'Pandey'): 5.583333333333333, ('Aaryan', 'Amandeep'): 3.333333333333333, ('Aditey', 'Baheti'): 4.25, ('Aditey', 'Anamay'): 4.583333333333333, ('Anuj', 'Baheti'): 5.25, ('Anuj', 'Amandeep'): 4.5, ('Ambuj', 'Amandeep'): 4.25, ('Ambuj', 'Pandey'): 3.75, ('Pandey', 'Ashvatth'): 6.833333333333333, ('Amandeep', 'Anamay'): 7.083333333333332, ('Baheti', 'Ashvatth'): 6.0, ('Anamay', 'Tanish'): 6.166666666666666, ('Ashvatth', 'Tanish'): 5.833333333333333}
Normalized Edge Betweenness Centrality: {('Aaryan', 'Aditey'): 0.12962962962962962, ('Aaryan', 'Anuj'): 0.08333333333333334, ('Aaryan', 'Ambuj'): 0.06666666666666667, ('Aaryan', 'Pandey'): 0.12407407407407407, ('Aaryan', 'Amandeep'): 0.07407407407407407, ('Aditey', 'Baheti'): 0.09444444444444444, ('Aditey', 'Anamay'): 0.10185185185185185, ('Anuj', 'Baheti'): 0.11666666666666667, ('Anuj', 'Amandeep'): 0.1, ('Ambuj', 'Amandeep'): 0.09444444444444444, ('Ambuj', 'Pandey'): 0.08333333333333334, ('Pandey', 'Ashvatth'): 0.15185185185185185, ('Amandeep', 'Anamay'): 0.15740740740740738, ('Baheti', 'Ashvatth'): 0.13333333333333333, ('Anamay', 'Tanish'): 0.13703703703703704, ('Ashvatth', 'Tanish'): 0.12962962962962962}

Communities after 1 step: ({'Ambuj', 'Amandeep', 'Aaryan', 'Anuj', 'Pandey'}, {'Tanish', 'Baheti', 'Ashvatth', 'Anamay', 'Aditey'})

Communities after 2 steps: ({'Ambuj', 'Amandeep', 'Aaryan', 'Anuj', 'Pandey'}, {'Anamay', 'Aditey'}, {'Tanish', 'Baheti', 'Ashvatth'})

Communities after 3 steps: ({'Ambuj', 'Aaryan', 'Pandey', 'Amandeep'}, {'Anamay', 'Aditey'}, {'Anuj'}, {'Tanish', 'Baheti', 'Ashvatth'})

Communities after 4 steps: ({'Ambuj', 'Aaryan', 'Pandey', 'Amandeep'}, {'Anamay', 'Aditey'}, {'Anuj'}, {'Baheti'}, {'Tanish', 'Ashvatth'})

Communities after 5 steps: ({'Ambuj', 'Aaryan', 'Amandeep'}, {'Anamay', 'Aditey'}, {'Anuj'}, {'Pandey'}, {'Baheti'}, {'Tanish', 'Ashvatth'})
```

Date: 11/08/2025 **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?**
   Nodes with the highest centrality values were most influential. Degree centrality showed well-connected individuals, betweenness revealed bridges between groups, closeness identified nodes that can quickly reach others, and eigenvector highlighted connections to other influential nodes. In the social network, these represent key communicators or connectors.

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 found tightly connected communities by removing high-betweenness edges. These groups reflect natural clusters (e.g., friend circles or teams) where members interact more within the group than outside, showing the network's underlying structure.

3. **Discuss the implications of identifying influential nodes in the network. How can this information be used?**
   Influential nodes help in spreading information, strengthening communities, and assessing network resilience. They can be targeted for marketing, leadership roles, or security measures, making them crucial for understanding and managing the network.