Experiment No.6
Develop Structure based social media analytics model for any business.
Date of Performance: 18/02/2025
Date of Submission: 25/02/2025

Aim: Develop Structure based social media analytics model for any business.

Objective: Develop a robust social media analytics model leveraging NetworkX in Python to analyze and visualize the structure of social networks for businesses, enabling insights into relationships, influences, and potential marketing strategies based on graph-based analysis of user interactions and connections.

Theory:

NetworkX is a Python package for complex graph network analysis. In order to understand NetworkX functionality, we first need to understand graphs.

What are Graphs?

Graphs are mathematical structures used to model many types of relationships and processes in physical, biological, social and information systems. A graph consists of nodes or vertices (representing the entities in the system) that are connected by edges (representing relationships between those entities). Working with graphs is a function of navigating edges and nodes to discover and understand complex relationships and/or optimize paths between linked data in a network.

There are many uses of graph network analysis, such as analyzing relationships in social networks, cyber threat detection, and identifying the people most likely to buy a product based on shared preferences.

In the real world, nodes can be people, groups, places, or things such as customers, products, members, cities, stores, airports, ports, bank accounts, devices, mobile phones, molecules, or web pages.

Examples of edges, or relationships between nodes, include friendships, network connections, hyperlinks, roads, routes, wires, phone calls, emails, "likes," payments, transactions, phone calls, and social networking messages. Edges can have a one-way direction arrow to represent a relationship from one node to another, like if Janet "liked" a social media post of Jeanette's. But they can also be non-directional, like if Bob is a Facebook friend of Alice, then Alice is also a friend of Bob.

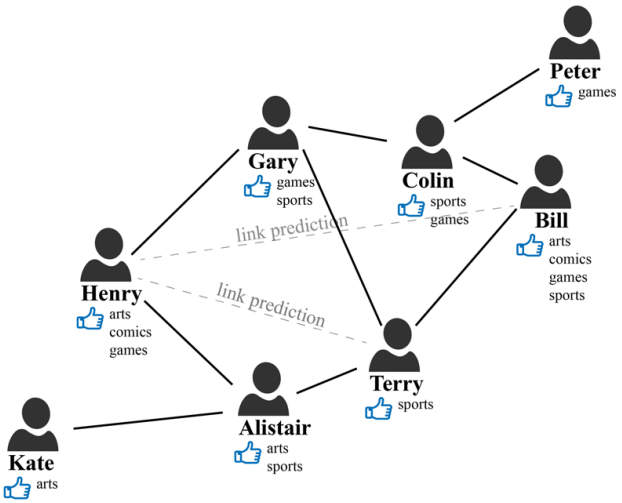


fig. 1: Graph based example

NetworkX nodes can be any object that is hashable, meaning that its value never changes. These can be text strings, images, XML objects, entire graphs, and customized nodes. The base package includes many functions to generate, read, and write graphs in multiple formats. NetworkX has the capacity to operate on very large graphs with more than 10 million nodes and 100 million edges. The core package, which is free software under the BSD license, includes data structures for representing such things as simple graphs, directed graphs, and graphs with parallel edges and self-loops. NetworkX also has a large community of developers who maintain the core package and contribute to a third-party ecosystem.
Among the principal uses of NetworkX are:

- Study of the structure and dynamics of social, biological, and infrastructure networks
- Standardized programming environment for graphs
- Rapid development of collaborative, multidisciplinary projects
- Integration with algorithms and code written in C, C++, and FORTRAN
- Working with large nonstandard data sets

NetworkX is considered relatively easy to install and use, particularly for Python developers.
Some functions in Networkx are:-

1) Graph.add_edge :- Used to add an edge between two nodes in a graph, allowing the creation or modification of the graph structure.
2) Graph.add_node :- Used to add a node to a graph, allowing users to specify node attributes if needed.

3) Graph.edges :- Returns a list of edges in the graph, where each edge is represented as a tuple of nodes.

4) networkx.draw(G, with_labels=1) :- Used to visualize the graph G with node labels displayed.

5) networkx.degree_centrality(G) :- Calculates the degree centrality for each node in the graph G, which is the fraction of nodes a node is connected to.

6) networkx.closeness_centrality(G) :- calculates the closeness centrality for each node in a graph, measuring the reciprocal of the average shortest path distance from a node to all other nodes in the graph.

7) networkx.degree(G) :- Returns a dictionary where keys are nodes in the graph G and values are their respective degrees, representing the number of edges incident to each node.

8) m_influential=networkx.degree_centrality(G)
networkx.eigenvector_centrality(G)
:- m_influential is a variable storing the degree centrality of nodes in graph G calculated using networkx.degree_centrality(). The function networkx.eigenvector_centrality(G) computes the eigenvector centrality of nodes in the graph, measuring their influence based on the principle that connections to high-scoring nodes contribute more to a node's score.

9) networkx.betweenness_centrality(G) :- It computes the betweenness centrality for each node in the graph G, measuring the fraction of shortest paths that pass through a node, identifying nodes that act as critical bridges in the network.

10) list(networkx.find_cliques(G)) :- Returns all cliques (complete subgraphs) in the graph G.

11) networkx.ego_graph :- It constructs the ego graph of a specified node, which includes the node and its neighbors within a given radius or distance.

12) networkx.spring_layout :- It implements a force-directed layout algorithm, positioning nodes in a graph based on attractive and repulsive forces between them to create visually appealing network visualizations.

13) networkx.density(G) :- It calculates the density of a graph G, representing the proportion of actual edges to the total number of possible edges in the graph.

14) networkx.has_bridges(G) :- It determines whether the graph G contains any bridges (edges whose removal increases the number of connected components) and returns a boolean value accordingly.

Code and Output:

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

# Step 1: Create a social network graph
G = nx.Graph()

# Step 2: Adding users (nodes)
users = ["Aarav", "Rahul", "Chirag", "Meenakshi", "Esha", "Saniya"]
G.add_nodes_from(users)

# Step 3: Establishing relationships (edges)
connections = [("Aarav", "Rahul"), ("Aarav", "Chirag"), ("Rahul", "Meenakshi"),
               ("Chirag", "Esha"), ("Meenakshi", "Esha"), ("Esha", "Saniya"),
               ("Rahul", "Esha"), ("Chirag", "Meenakshi")]
G.add_edges_from(connections)


# Step 4: Calculate centrality metrics
degree_centrality = nx.degree_centrality(G)
closeness_centrality = nx.closeness_centrality(G)
betweenness_centrality = nx.betweenness_centrality(G)
eigenvector_centrality = nx.eigenvector_centrality(G)
```

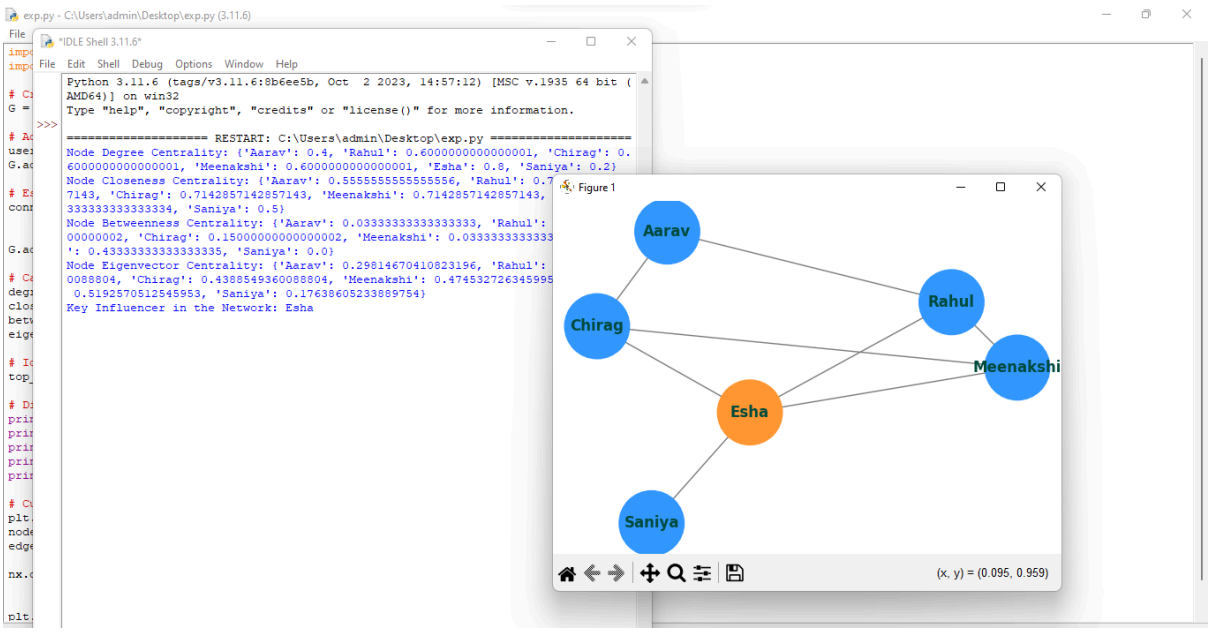
```
# Step 5: Identify the most influential user based on degree centrality
top_influencer = max(degree_centrality, key=degree_centrality.get)

# Step 6: Display results
print("Node Degree Centrality:", degree_centrality)
print("Node Closeness Centrality:", closeness_centrality)
print("Node Betweenness Centrality:", betweenness_centrality)
print("Node Eigenvector Centrality:", eigenvector_centrality)
print(f"Key Influencer in the Network: {top_influencer}")

# Step 7: Customize Visualization
plt.figure(figsize=(8, 6))
node_colors = ['#ff9933' if node == top_influencer else '#3399ff' for node in G.nodes()]
edge_colors = '#808080'

nx.draw(G, with_labels=True, node_color=node_colors, edge_color=edge_colors,
        node_size=2800, font_size=12, font_color="#004d40", font_weight="bold")

plt.title("Modified Social Media Network Representation", fontsize=14, color="#333333")
plt.show()
```



Conclusion:

This experiment uses NetworkX to analyze social media networks by identifying key users and relationships. It computes centrality measures to find influential users and visualize connections. Businesses can use this model for marketing, customer segmentation, and fraud detection. The graph-based approach helps in understanding user influence and engagement. Future improvements can include real-time data analysis and AI-driven predictions.