!pip install snap-stanford

import pandas as pd

import networkx as nx

df = pd.read\_csv("ga\_edgelist.csv")

#showing top 5 data

df.head(5)

#By using the nx.from\_pandas\_edgelist function with your DataFrame,

#you are creating a graph where the 'from' column represents the source nodes,

#the 'to' column represents the target nodes, and each row in the DataFrame represents an edge in the graph.

us\_graph=nx.from\_pandas\_edgelist(df,source='from',target='to')

# Get the in-degrees and out-degrees for each node

#node name

us\_graph.nodes()

#number of nodes

num\_nodes = len(us\_graph.nodes)

print("Number of nodes:", num\_nodes)

#number of edges

num\_edges= len(us\_graph.edges)

print("Number of edges:", num\_edges)

# Average degree

degree = sum(dict(us\_graph.degree()).values())

average\_degree = degree / num\_nodes

print("Average degree (average degree):", average\_degree)

#edge details

us\_graph.edges()

#if there is any edge between two node

if us\_graph.has\_edge('grey', 'colin'):

  print("already connected")

else:

  us\_graph.add\_edge('grey','colin')

#Visualization

import matplotlib.pyplot as plt

%matplotlib inline

nx.draw(us\_graph,with\_labels=True,node\_color='b')

#Degree information of each graph

nx.degree(us\_graph)

import matplotlib.pyplot as plt

degrees = [degree for node, degree in us\_graph.degree()]

# Plot the degree distribution

plt.hist(degrees, bins=20, alpha=0.5, color='b', edgecolor='k')

plt.title("Degree Distribution")

plt.xlabel("Degree")

plt.ylabel("Number of Nodes")

plt.show()

# Centralinty is used to measure influential nodes in the graph

# 0: means isolated nodes

# 1: means that node is connected with all other nodes in the graph

nx.degree\_centrality(us\_graph)

# Showing influence as per descending order

degree\_centrality\_inf=nx.degree\_centrality(us\_graph)

for w in sorted(degree\_centrality\_inf, key=degree\_centrality\_inf.get,reverse=True):

    print(w, degree\_centrality\_inf[w])

between\_cen\_inf = nx.betweenness\_centrality(us\_graph)

for w in sorted(between\_cen\_inf, key=between\_cen\_inf.get,reverse=True):

    print(w, between\_cen\_inf[w])

ev\_cen\_inf = nx.eigenvector\_centrality(us\_graph)

for w in sorted(ev\_cen\_inf, key=ev\_cen\_inf.get,reverse=True):

    print(w, ev\_cen\_inf[w])

#nodes that are all reachable from a person

group1 = nx.bfs\_tree(us\_graph,'karev')

nx.draw\_networkx(group1)

#nodes that are all reachable from a person using connected component

nx.node\_connected\_component(us\_graph,'torres')

#Finding the cluster in a graph

connected\_components = list(nx.connected\_components(us\_graph))

# Print the different connected components

for i, component in enumerate(connected\_components, 1):

  print(f"Connected Component {i}: {component}")

#Finding diameter of each component

connected\_components = list(nx.connected\_components(us\_graph))

# Print the different connected components

for i, component in enumerate(connected\_components, 1):

   subgraph = us\_graph.subgraph(component)

   diameter = nx.diameter(subgraph)

   print(f"Connected Component {i}: {component} (Diameter: {diameter})")

#Clique is a subgraph that are fully connected

clique = nx.find\_cliques(us\_graph)

# Find all cliques in the graph of length grater than 2

all\_cliques = list(nx.find\_cliques(us\_graph))

# Filter cliques with size greater than 2

cliques\_greater\_than\_2 = [clique for clique in all\_cliques if len(clique) > 1]

# Print the cliques with size greater than 2

for i, clique in enumerate(cliques\_greater\_than\_2, 1):

    print(f"Clique {i}: {clique}")

import networkx as nx

import matplotlib.pyplot as plt

G = nx.karate\_club\_graph()

nx.draw(G,with\_labels=True,node\_color='b')

plt.show()

hits\_scores = nx.hits(G, max\_iter=100)

hub\_scores, authority\_scores = hits\_scores

print("Hub Scores:", hub\_scores)

print("Authority Scores:", authority\_scores)

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.hist(list(hub\_scores.values()), bins=20, alpha=0.5, color='b')

plt.title("Hub Scores Distribution")

plt.xlabel("Hub Score")

plt.ylabel("Frequency")

plt.tight\_layout()

plt.show()

plt.subplot(1, 2, 2)

plt.hist(list(authority\_scores.values()), bins=20, alpha=0.5, color='g')

plt.title("Authority Scores Distribution")

plt.xlabel("Authority Score")

plt.ylabel("Frequency")

plt.tight\_layout()

plt.show()

#Page rank algorithm

import networkx as nx

import matplotlib.pyplot as plt

# Create a directed graph representing the hyperlink structure

G = nx.DiGraph()

G.add\_edges\_from([(1, 2), (1, 4), (2, 3), (3, 1), (3, 4), (4, 2), (1,4)])

# Calculate PageRank

pagerank\_scores = nx.pagerank(G, alpha=0.85)

# Print PageRank scores

print("PageRank scores:", pagerank\_scores)

# Draw the graph with node sizes proportional to their PageRank scores

node\_sizes = [pagerank\_scores[node] \* 5000 for node in G.nodes]

pos = nx.circular\_layout(G)

nx.draw(G, pos, with\_labels=True, node\_size=node\_sizes, node\_color='skyblue', font\_size=10, font\_color='black', font\_weight='bold', arrowsize=20)

plt.show()

import networkx as nx

import matplotlib.pyplot as plt

# Create a sample graph

G = nx.karate\_club\_graph()

# Select a focal node (ego)

# We can select any node here

ego\_node = 1

# Extract the ego network

ego\_network = nx.ego\_graph(G, ego\_node)

# Plot the ego network

pos = nx.spring\_layout(G)  # You can choose different layout algorithms

nx.draw(G, pos, with\_labels=True, font\_weight='bold', node\_color='lightblue', node\_size=800)

nx.draw(ego\_network, pos, with\_labels=True, font\_weight='bold', node\_color='salmon', node\_size=800)