import networkx as nx

import numpy as np

import pandas as pd

print("All Modules Imported Successfully")

data = pd.read\_csv('ga\_edgelist.csv')

print(df)

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

#Visualization

import matplotlib.pyplot as plt

%matplotlib inline

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

#number of nodes

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

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

#number of edges

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

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

# Average degree

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

average\_degree = degree / num\_nodes

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

print("All Nodes of the Network: ")

for i in pp\_graph.nodes:

    print(i)

pp\_graph.edges()

pp\_graph.nodes()

#if there is any edge between two node

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

  print("already connected")

else:

  pp\_graph.add\_edge('grey','colin')

#Degree information of each graph

nx.degree(pp\_graph)

import matplotlib.pyplot as plt

degrees = [degree for node, degree in pp\_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(pp\_graph)

degree\_centrality\_inf=nx.degree\_centrality(pp\_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(pp\_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(pp\_graph)

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

    print(w, ev\_cen\_inf[w])

#EigenVector Centrality

top\_5\_nodes\_ev= sorted(ev\_cen\_inf, key=ev\_cen\_inf.get, reverse=True)[:5]

for w in top\_5\_nodes\_ev:

    print(w, ev\_cen\_inf[w])

#Degree Centrality

top\_5\_nodes\_dc= sorted(degree\_centrality\_inf, key=degree\_centrality\_inf.get,reverse=True)[:5]

for w in top\_5\_nodes\_dc:

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

#Between Ness Centrality

top\_5\_nodes\_bc= sorted(between\_cen\_inf, key=between\_cen\_inf.get,reverse=True)[:5]

for w in top\_5\_nodes\_bc:

    print(w, between\_cen\_inf[w])

# Calculate the top 5 nodes for each centrality measure

top\_5\_nodes\_ev = sorted(ev\_cen\_inf, key=ev\_cen\_inf.get, reverse=True)[:5]

top\_5\_nodes\_dc = sorted(degree\_centrality\_inf, key=degree\_centrality\_inf.get, reverse=True)[:5]

top\_5\_nodes\_bc = sorted(between\_cen\_inf, key=between\_cen\_inf.get, reverse=True)[:5]

# Find the union of the top 5 nodes from all centrality measures

union\_top\_5 = set(top\_5\_nodes\_ev) | set(top\_5\_nodes\_dc) | set(top\_5\_nodes\_bc)

# Find the intersection of the top 5 nodes from all centrality measures

intersection\_top\_5 = set(top\_5\_nodes\_ev) & set(top\_5\_nodes\_dc) & set(top\_5\_nodes\_bc)

# Print the union and intersection

print("Union of the top 5 nodes from all centrality measures:", union\_top\_5)

print("Intersection of the top 5 nodes from all centrality measures:", intersection\_top\_5)

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

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

#Finding the cluster in a graph

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

# Print the different connected components

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

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