```
#https://stackoverflow.com/questions/48293601/how-to-install-libspatialindex-on-google-col
#https://colab.research.google.com/drive/1N7i9zmOwVcUzd4eHWZux4p_WTBMZHi8C

######################################################################
### Rodar toda vez que for usar o Osmnx no Google Colaboratory ###
######################################################################

import os
!pip install geopandas
!pip install rtree
!pip install osmnx
!mount -o remount,exec /content
!apt-get install -qq curl g++ make
!curl -L http://download.osgeo.org/libspatialindex/spatialindex-src-1.8.5.tar.gz | tar xz
os.chdir('spatialindex-src-1.8.5')
!./configure
!make
!make install
!pip install rtree
!ldconfig
os.chdir('/content/')
clear_output()


from networkx.algorithms.community import greedy_modularity_communities
from sklearn.metrics.cluster import normalized_mutual_info_score
from networkx.algorithms.community import LFR_benchmark_graph
from sklearn.preprocessing import StandardScaler
from networkx.algorithms import community
from IPython.display import clear_output
from community import community_louvain
from sklearn.decomposition import PCA
from scipy.linalg import expm
from rtree.index import Rtree
from random import randint
from rtree import index
from numpy  import *
import matplotlib.pyplot as plt
import statistics as stats
import scipy.stats as st
import networkx as nx
import seaborn as sns
import pandas as pd
import osmnx as ox
import numpy as np
import warnings
import random
import scipy
import os
warnings.filterwarnings('ignore')
```

1 – For the following networks:

a) E-road network (http://konect.cc/networks/subelj_euroroad),

b) Hamsterster friendships (http://konect.uni-koblenz.de/networks/petsterfriendships-hamst

c) C. elegans neural network (http://wwwpersonal.umich.edu/~mejn/netdata/celegansneural.

d) US airport network (http://toreopsahl.com/datasets/#usairports)

Construct a correlation matrix between the centrality measures:

(i) degree,

(ii) kcore,

(iii) closeness centrality,

(iv) betweenness centrality,

(v) eigenvector centrality,

(vi) pagerank,

(vi) random walk accessibility,

(v) communicability centrality.

Discuss the highest correlations and interpret the results.

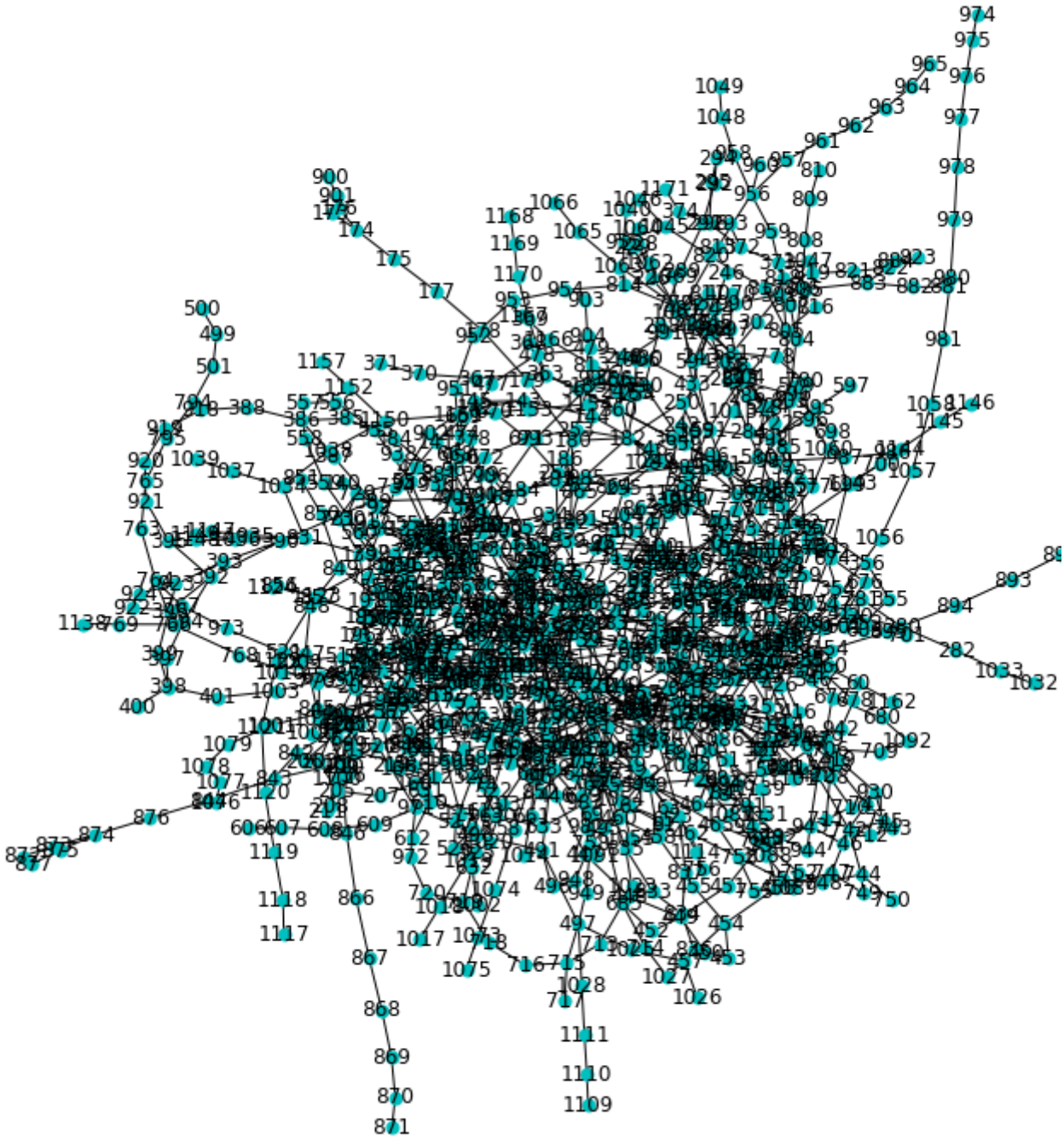
```
def refaz_multigraph(G):
  Gnew = nx.Graph()
  for edge in G.edges():
    Gnew.add_edge(edge[0], edge[1])
  return Gnew

def refaz_grafo_ponderado(G):
  Gnew = nx.Graph()
  for (u,v) in G.edges:
    Gnew.add_edge(u, v)
  return Gnew

def trata_rede(G                                    # Se True:
               , undirected = True                  # Deixa o Grafo bidirecionado
               , convert_to_int = True              # Troca os labels dos nós para inteir
               , most_connected_component = True    # Extrai o componente com maior numer
               , remove_selfedges = True            # Remove loops
               , multigraph = False                 # Transforma de Multigrafo para grafo
               , remove_weights = True):            # Remove pesos

  if multigraph : G = refaz_multigraph(G)
  if remove_selfedges : G.remove_edges_from(G.selfloop_edges())
```

```
  if most_connected_component : G = sorted(nx.connected_component_subgraphs(G), key = len,
  if convert_to_int : G = nx.convert_node_labels_to_integers(G, first_label=0)
  if undirected : G = G.to_undirected()
  if remove_weights : G = refaz_grafo_ponderado(G)
  return G


def plota_rede(G, fig_size = 6, nd_size = 500, n_color = 0):
  colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w']
  plt.figure(figsize=(fig_size,fig_size))
  pos=nx.spring_layout(G)
  nx.draw(G, with_labels = True, pos = pos, node_size = nd_size, node_color = colors[n_col
  plt.show(True)


def degree_centrality(G):
  vk = dict(G.degree())
  return list(vk.values())


def closeness_centrality(G):
  CLC = dict(nx.closeness_centrality(G))
  return list(CLC.values())


def betweenness_centrality(G):
  B = dict(nx.betweenness_centrality(G))
  return list(B.values())


def eigenvector_centrality(G):
  EC = dict(nx.eigenvector_centrality(G, max_iter = 1000))
  return list(EC.values())


def pagerank(G):
  PR = dict(nx.pagerank(G, alpha=0.85))
  PR = list(PR.values())
  return np.array(PR)


def kcore(G):
  return list(dict(nx.core_number(G)).values())


def communicability_centrality(G):
  return np.asarray(list(nx.subgraph_centrality_exp(G).values()))


def random_walk_accessibility(G):
    N = len(G.nodes())
    vk = dict(G.degree())
    vk = list(vk.values())
    A = nx.adjacency_matrix(G)
    P = np.zeros((N,N), dtype = 'float')
    for i in np.arange(0, N):
        for j in np.arange(i, N):
            P[i,j] = A[i,j]/vk[i]
            P[j,i] = P[i,j]
    P2 = expm(P)/np.exp(1)
```

```
    vacc = np.zeros(N, dtype = float)
    for i in np.arange(0, N):
        acc = 0
        for j in np.arange(0,N):
            if(P2[i,j] > 0):
                acc = acc + P2[i,j]*log(P2[i,j])
        acc = np.exp(-acc)
        vacc[i] = acc
    return vacc


def correlation_matrix(df, type = 1):
  if type == 1 :
    corr = df.corr()
    plt.figure(figsize=(7, 7))
    mask = np.zeros_like(corr)
    mask[np.triu_indices_from(mask)] = True
    with sns.axes_style("white"):ax = sns.heatmap(corr, mask=mask, vmax=1, square=True)
    plt.show()

  elif type == 2 :
    corr = df.corr()
    #Plot Correlation Matrix using Matplotlib
    plt.figure(figsize=(8,8))
    plt.imshow(corr, cmap='Blues', interpolation='none', aspect='auto')
    plt.colorbar()
    plt.xticks(range(len(corr)), corr.columns, rotation='vertical', fontsize=20)
    plt.yticks(range(len(corr)), corr.columns, fontsize=20);
    plt.suptitle('Correlation between centrality measures', fontsize=20)
    plt.grid(False)
    plt.show()
  return


listaGraph = list()
listaIndex = ["E-road network", "Hamsterster friendships", "C.elegans neural  network", "

G_eroad = nx.read_edgelist("eroad.edges", nodetype=int, data=(('weight',int),))
listaGraph.append(trata_rede(G_eroad))

G_hamster = nx.read_edgelist("Hamsterster.edges", nodetype=int, data=(('weight',int),))
listaGraph.append(trata_rede(G_hamster))

G_celneural = nx.read_gml("celegansneural.gml")
listaGraph.append(trata_rede(G_celneural, multigraph = True))

G_usaairport = nx.read_edgelist("usaairport.edges", nodetype=int, data=(('weight',int),))
listaGraph.append(trata_rede(G_usaairport))


def ex1(G,T):
  VK = degree_centrality(G)
  KC = kcore(G)
  CLC = closeness_centrality(G)
  B = betweenness_centrality(G)
  EC = eigenvector_centrality(G)
  PR = pagerank(G)
```

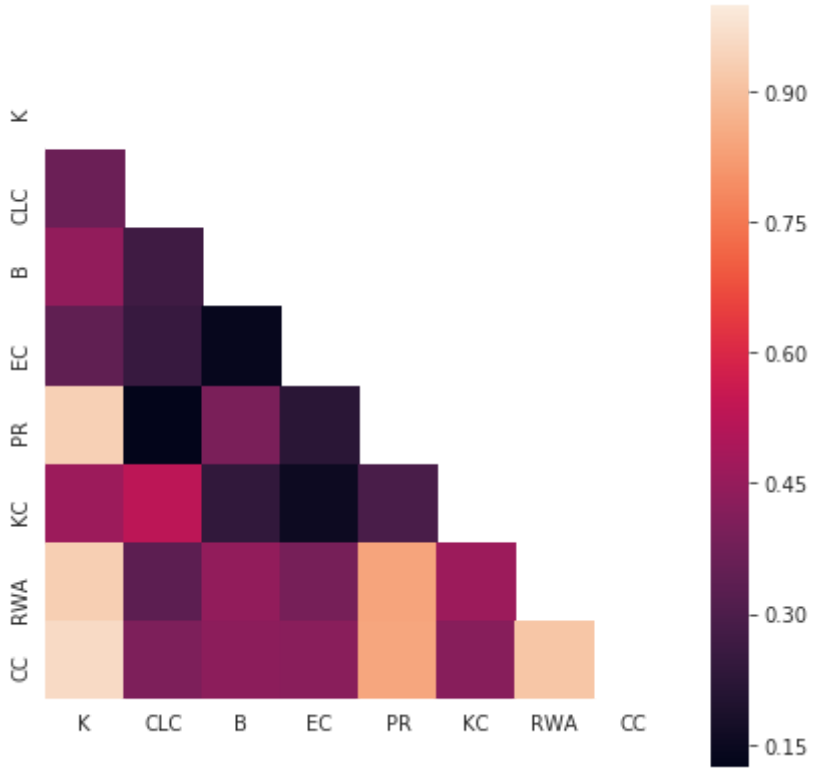
```
  RWA = random_walk_accessibility(G)
  CC = communicability_centrality(G)
  df = pd.DataFrame({'K':VK,'CLC':CLC,'B':B,'EC':EC,'PR':PR,'KC':KC, 'RWA':RWA, 'CC': CC})
  #df = pd.DataFrame({'K':VK,'CLC':CLC,'B':B,'EC':EC,'PR':PR,'KC':KC, 'RWA':RWA})
  print("Matriz de correlações entre \nK Degreee Centrality\nKC Kcore\nCLC Closeness Centr
  print("EC Eigenvector Centrality\nPR Pagerank\nRWA Random Walk Accessibility\nCC Communi
  correlation_matrix(df,1)


T = listaIndex[0]
G = listaGraph[0]
print("Rede: ", T)
plota_rede(G, nd_size=50, fig_size=10, n_color = random.randint(0,8))
```

```
Rede:  E-road  network
```



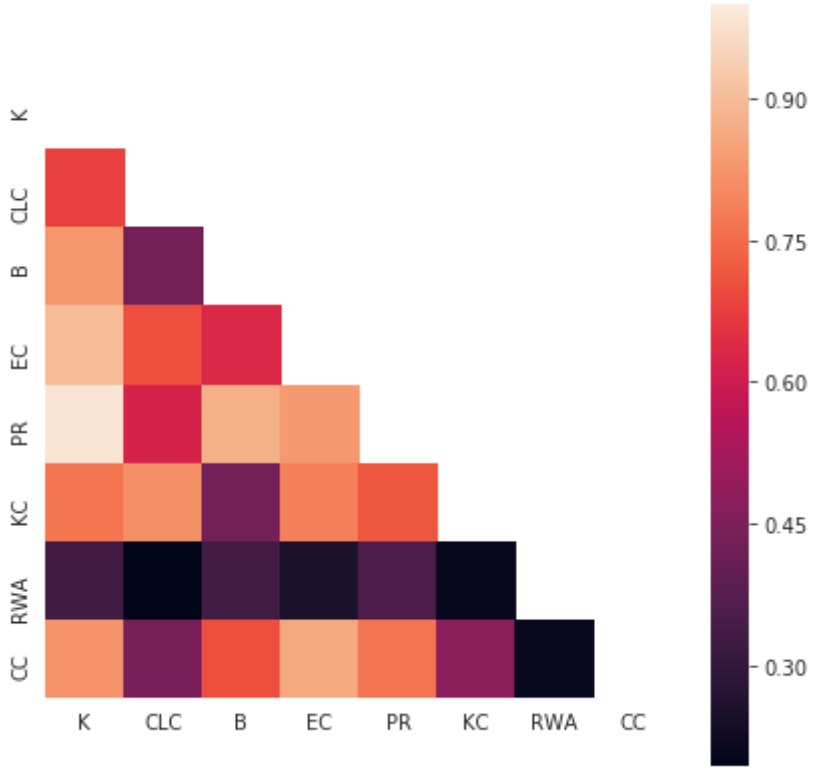
```
ex1(G,T)
```

```
Matriz de correlações entre
K Degreee Centrality
KC Kcore
CLC Closeness Centrality
B Betwenesses Centrality
EC Eigenvector Centrality
PR Pagerank
RWA Random Walk Accessibility
CC Communicability Centrality
```



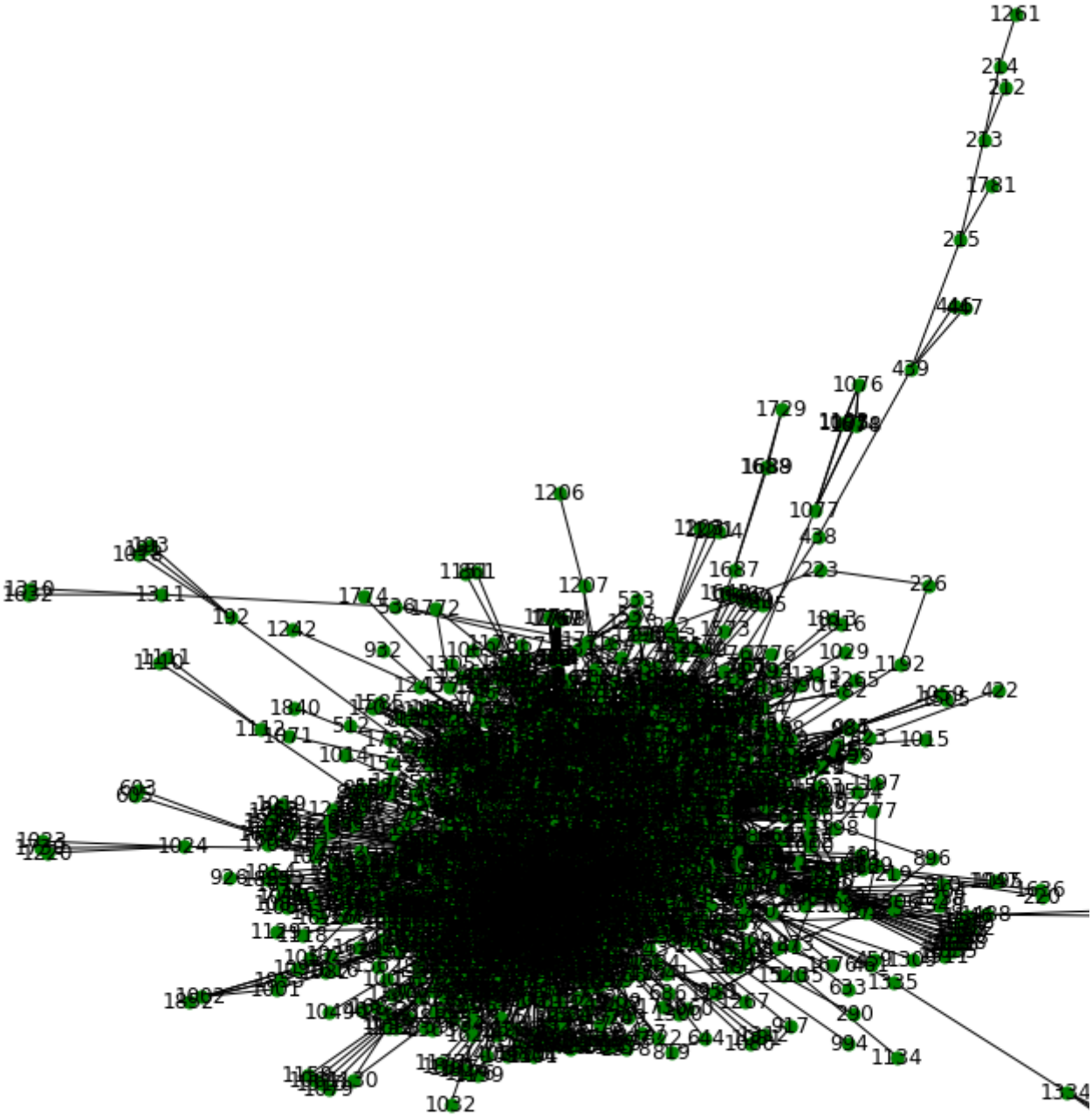

```
T = listaIndex[1]
G = listaGraph[1]
ex1(G,T)
```

```
Matriz de correlações entre
K Degreee Centrality
KC Kcore
CLC Closeness Centrality
B Betwenesses Centrality
EC Eigenvector Centrality
PR Pagerank
RWA Random Walk Accessibility
CC Communicability Centrality
```



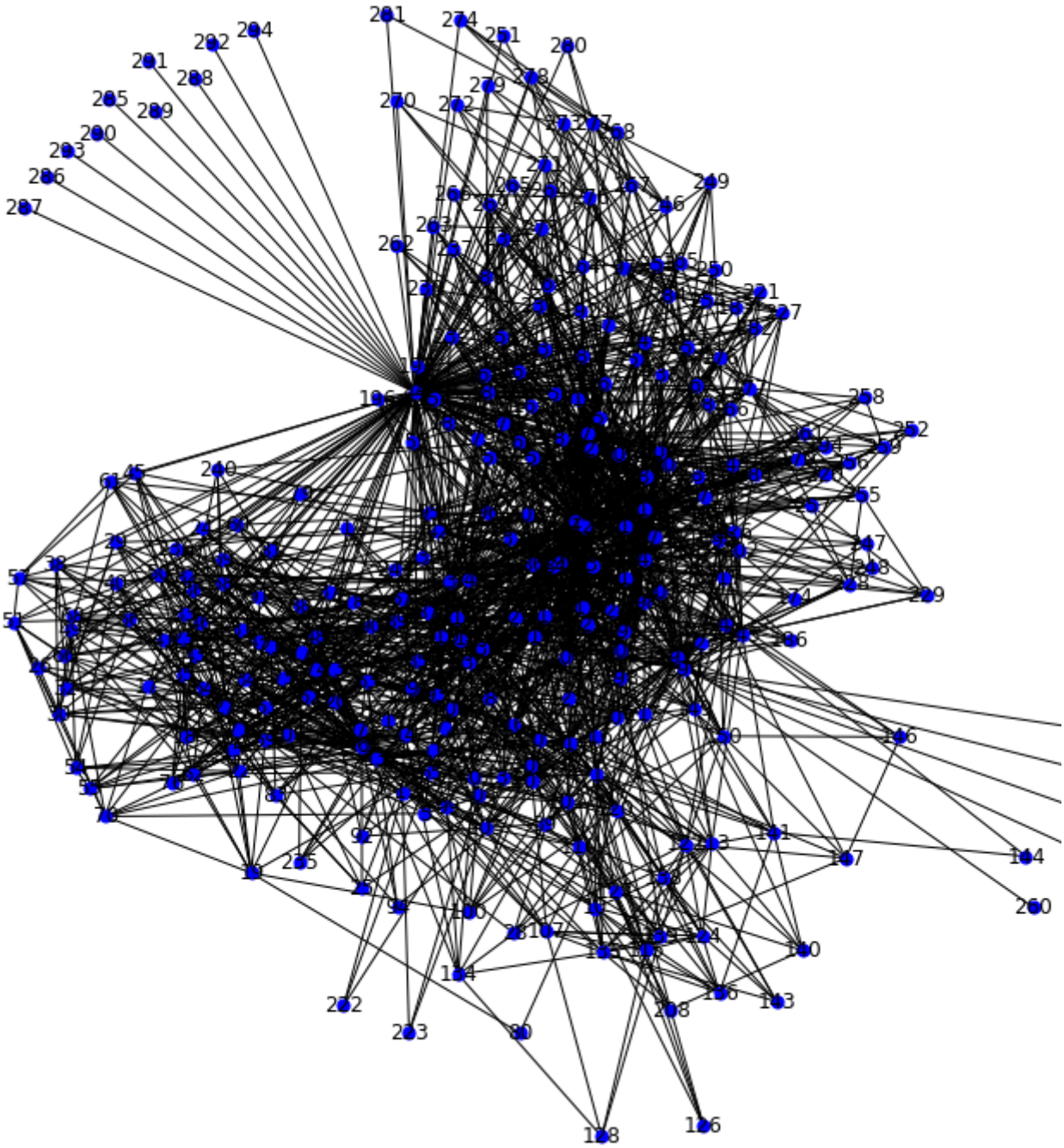

```
print("Rede: ", T)
plota_rede(G, nd_size=50, fig_size=10, n_color = random.randint(0,8))
```

```
Rede:  Hamsterster friendships
```



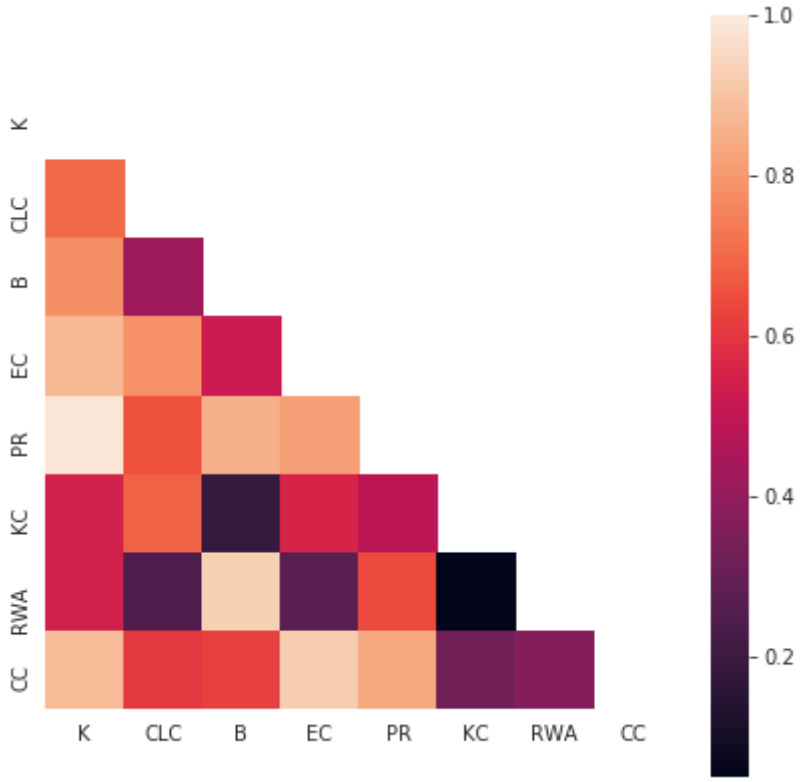
```
T = listaIndex[2]
G = listaGraph[2]
print("Rede: ", T)
plota_rede(G, nd_size=50, fig_size=10, n_color = random.randint(0,8))
```

```
Rede:  C.elegans        neural  network
```



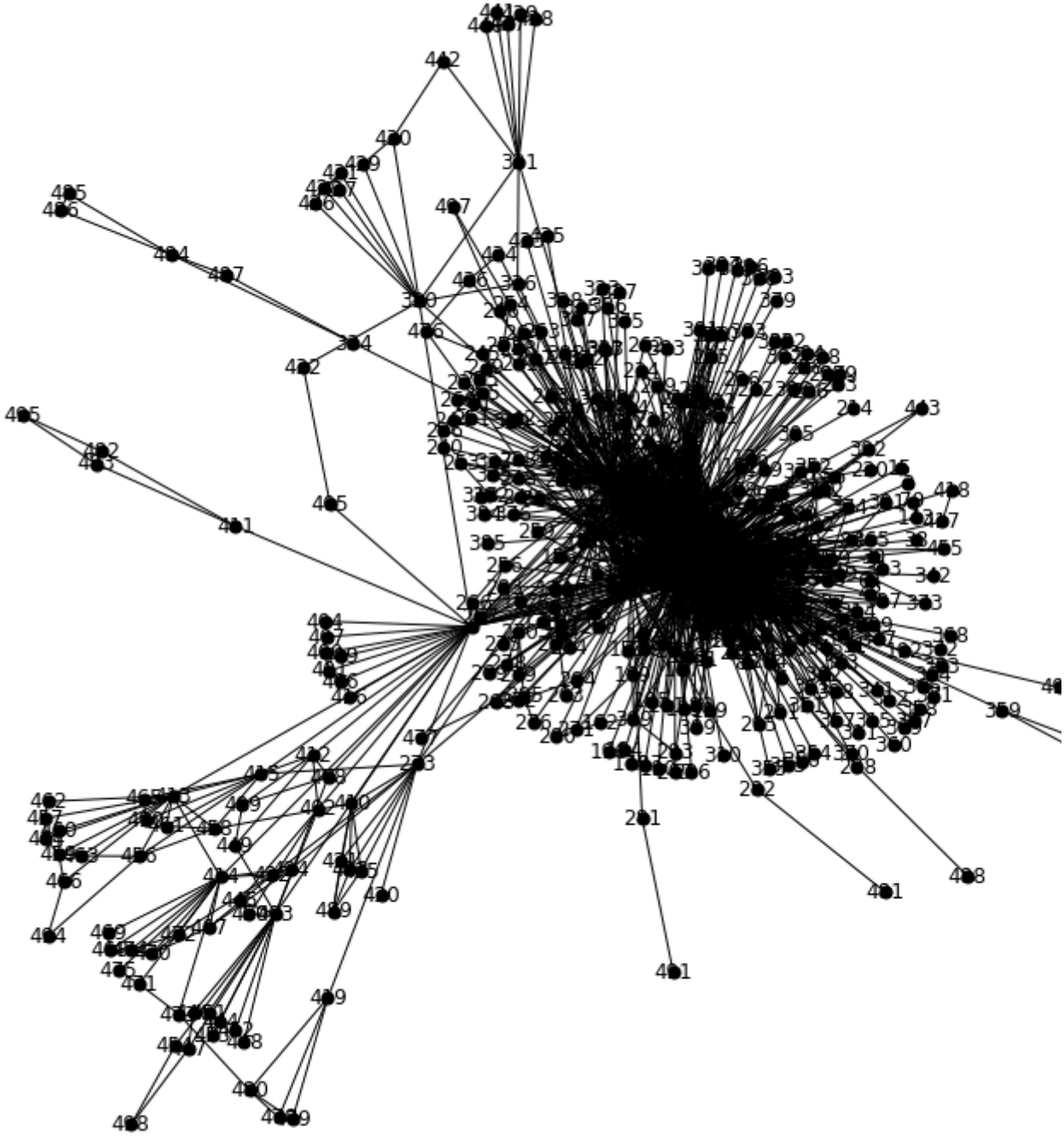
```
ex1(G,T)
```

```
Matriz de correlações entre
K Degreee Centrality
KC Kcore
CLC Closeness Centrality
B Betwenesses Centrality
EC Eigenvector Centrality
PR Pagerank
RWA Random Walk Accessibility
CC Communicability Centrality
```



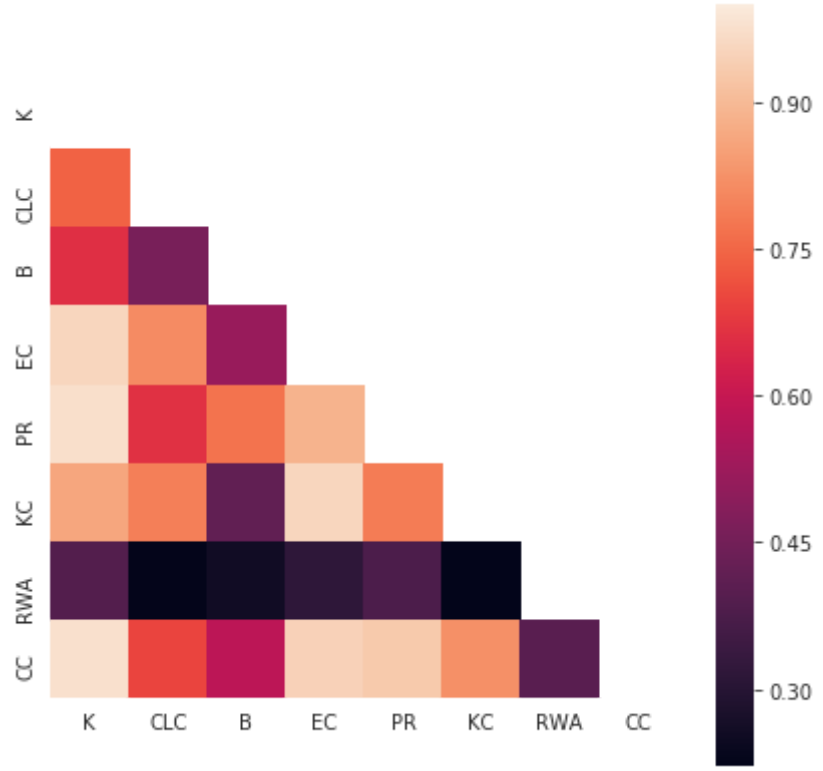

```
T = listaIndex[3]
G = listaGraph[3]
print("Rede: ", T)
plota_rede(G, nd_size=50, fig_size=10, n_color = random.randint(0,8))
```

Rede: US airport network



ex1(G,T)

```
Matriz de correlações entre
K Degreee Centrality
KC Kcore
CLC Closeness Centrality
B Betwenesses Centrality
EC Eigenvector Centrality
PR Pagerank
RWA Random Walk Accessibility
CC Communicability Centrality
```



2 – Choose four dataset of cities (from OSMX or any other dataset) and c centrality measures. That is, construct the histogram of

(i) degree,

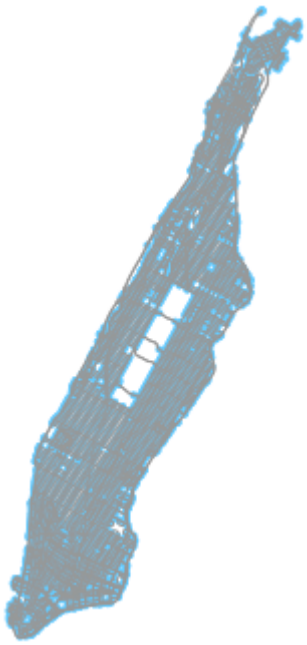
(ii) closeness centrality and

(iii) betweenness centrality. Discuss which city is easier to navigate in terms of these distribut

```
listaGraph = list()
listaIndex = ["Manhattan - EUA", "São Carlos - Brasil", "Brest - França", "Veneza - Itália
```

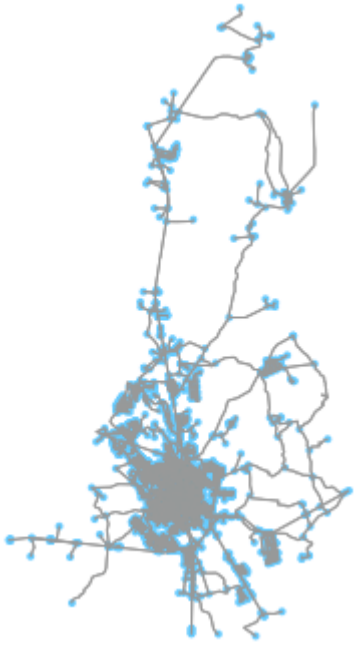
```
G_manhattan = ox.graph_from_place('Manhattan Island, New York City, New York, USA', networ
print("Ilha de Manhattan")
ox.plot_graph(G_manhattan)
listaGraph.append(trata_rede(G_manhattan, multigraph= True, remove_selfedges=False))
```

Ilha de Manhattan



```
G_sanca = ox.graph_from_place('São Carlos', network_type='drive')
print("Cidade de São Carlos")
ox.plot_graph(G_sanca)
listaGraph.append(trata_rede(G_sanca, multigraph= True, remove_selfedges=False))
```

Cidade de São Carlos

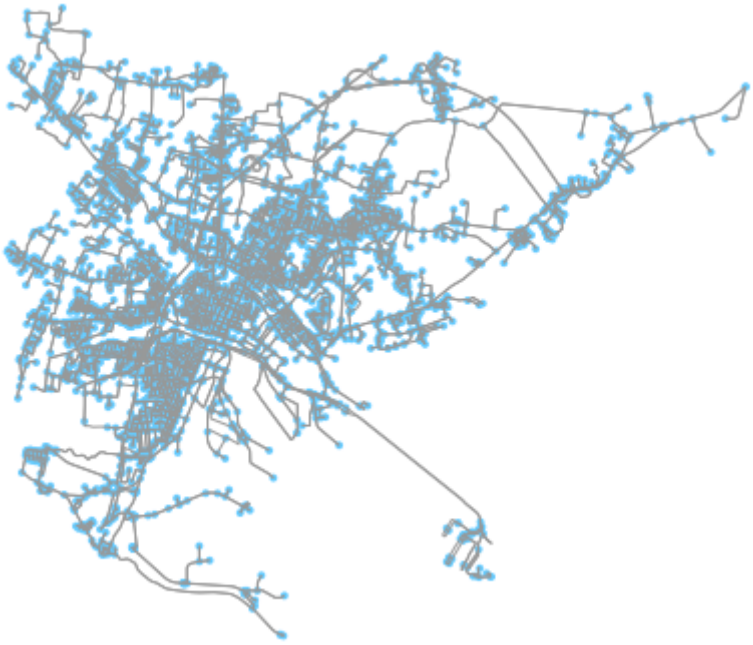


```
G_brest = ox.graph_from_place('Brest, France', network_type='drive')
print("Cidade de Brest na Bretanha francesa")
ox.plot_graph(G_brest)
listaGraph.append(trata_rede(G_brest, multigraph= True, remove_selfedges=False))
```

Cidade de Brest na Bretanha francesa

```
G_venice = ox.graph_from_place('Venice', network_type='drive')
print("Cidade de Veneza na Itália")
ox.plot_graph(G_venice)
listaGraph.append(trata_rede(G_venice, multigraph= True, remove_selfedges=False))
```

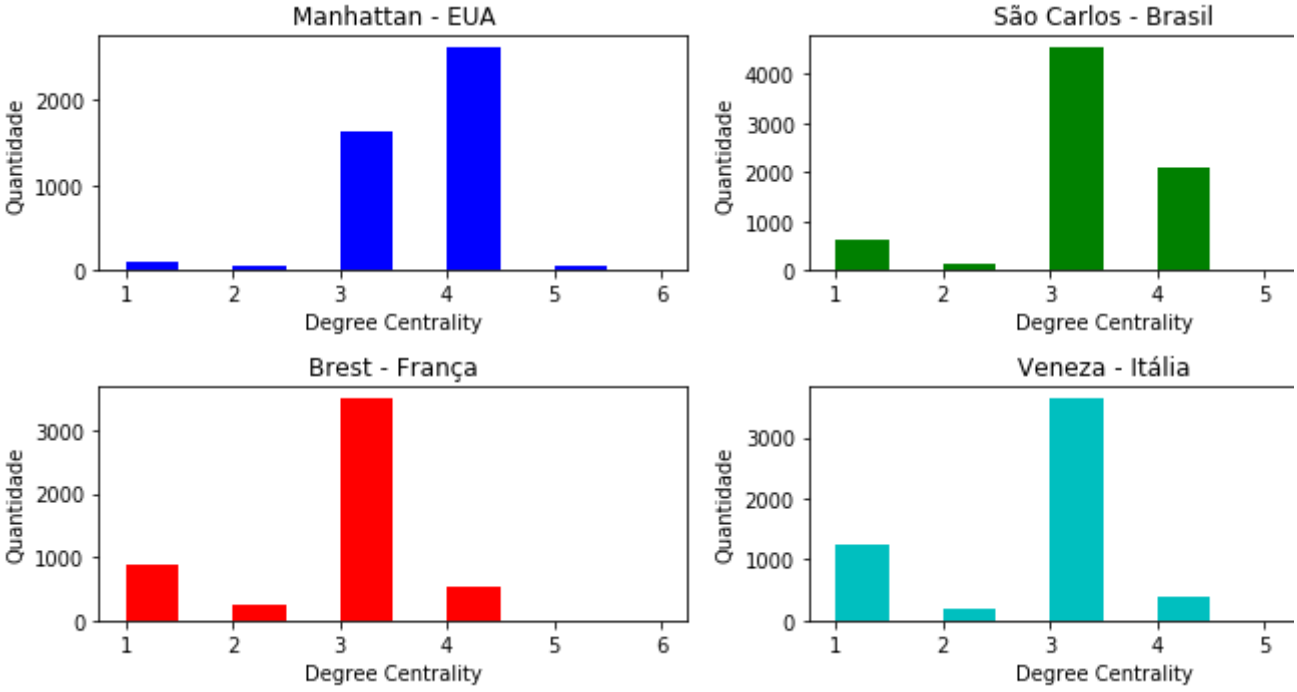
Cidade de Veneza na Itália



```
print('Histogramas da distribuição de grau de acordo com a cidade')
f,a = plt.subplots(2,2, figsize=(10,5))
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w']
a = a.ravel()
for idx,ax in enumerate(a):
    ax.hist(degree_centrality(listaGraph[idx]),  facecolor=colors[idx])
    ax.set_title(listaIndex[idx])
    ax.set_xlabel("Degree Centrality")
```

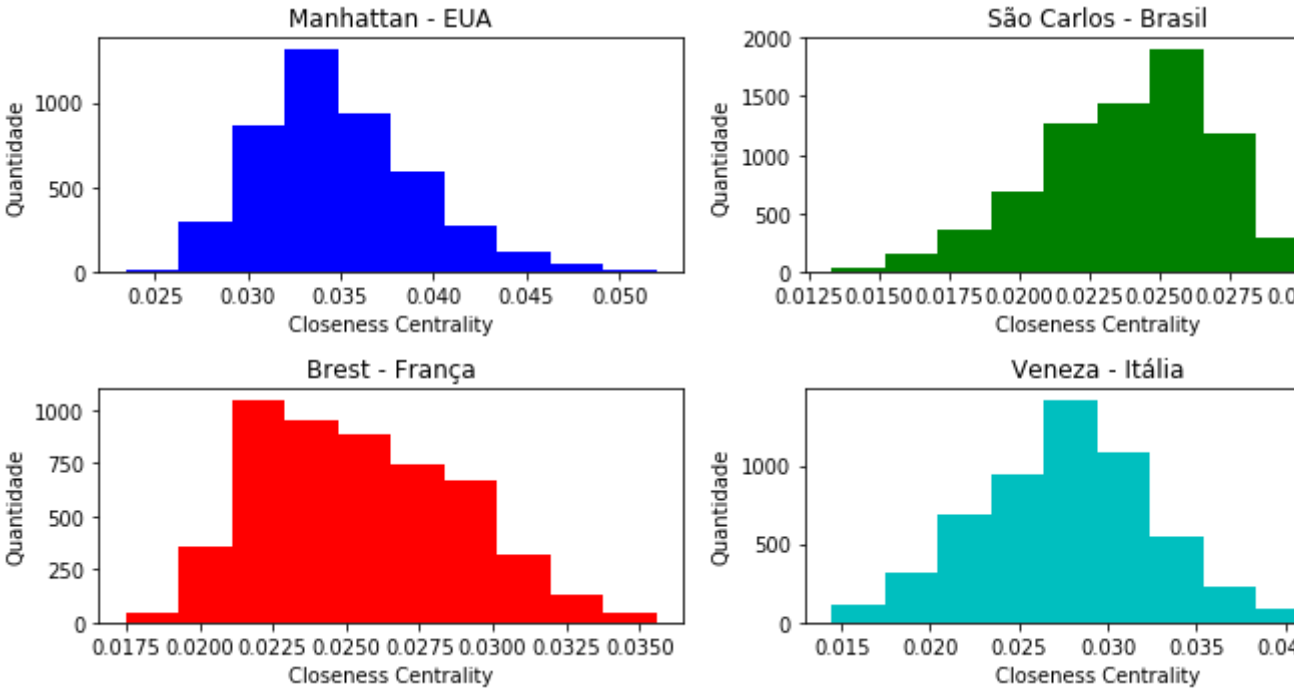
```
    ax.set_ylabel("Quantidade")
plt.tight_layout()
```

```
Histogramas da distribuição de grau de acordo com a cidade
```



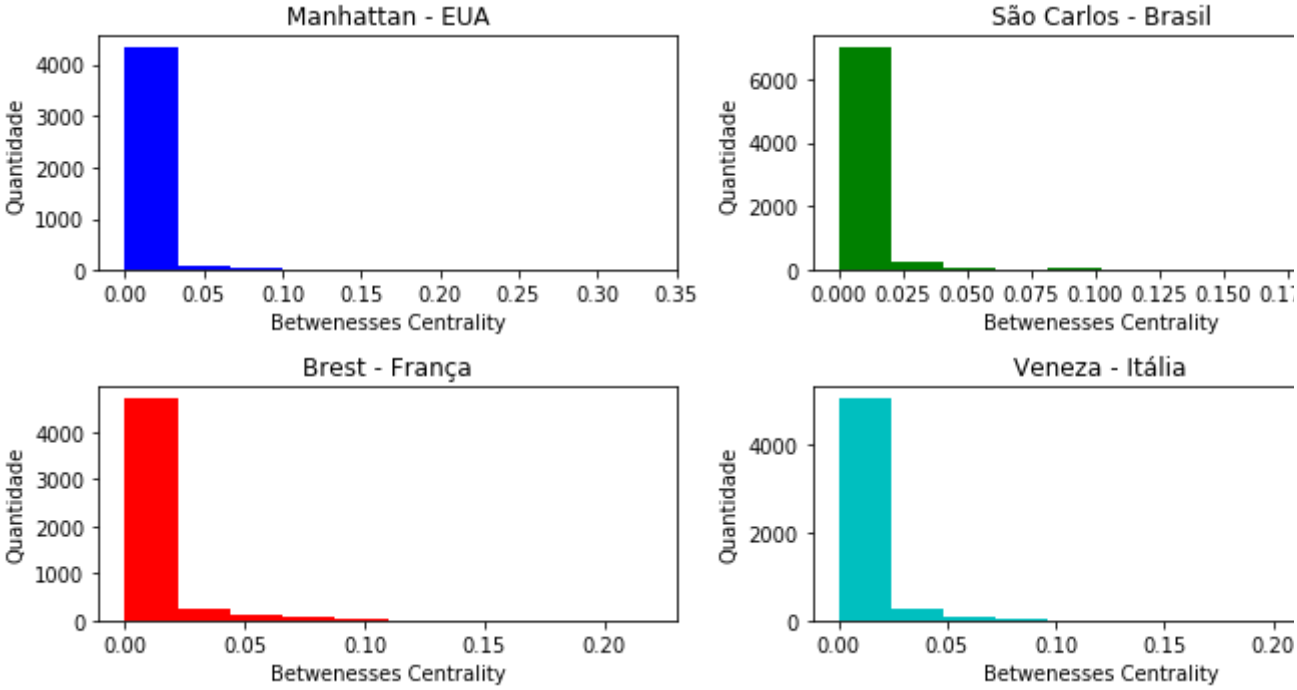

```
print('Histogramas da distribuição de closeness centrality de acordo com a cidade')
f,a = plt.subplots(2,2, figsize=(10,5))
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w']
a = a.ravel()
for idx,ax in enumerate(a):
    print(round(100*(idx/4),2),"%")
    ax.hist(closeness_centrality(listaGraph[idx]),  facecolor=colors[idx])
    ax.set_title(listaIndex[idx])
    ax.set_xlabel("Closeness Centrality")
    ax.set_ylabel("Quantidade")
    clear_output()
print('Histogramas da distribuição de closeness centrality de acordo com a cidade')
plt.tight_layout()
```

Histogramas da distribuição de closeness centrality de acordo com a cidade




```
print('Histogramas da distribuição de betwenesses centrality de acordo com a cidade')
f,a = plt.subplots(2,2, figsize=(10,5))
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w']
a = a.ravel()
for idx,ax in enumerate(a):
    print(round(100*(idx/4),2),"%")
    ax.hist(betweenness_centrality(listaGraph[idx]),  facecolor=colors[idx])
    ax.set_title(listaIndex[idx])
    ax.set_xlabel("Betwenesses Centrality")
    ax.set_ylabel("Quantidade")
    clear_output()
print('Histogramas da distribuição de betwenesses centrality de acordo com a cidade')
plt.tight_layout()
```

Histogramas da distribuição de betwenesses centrality de acordo com a cidade



3 – For the following networks:

a) Human protein network (http://konect.cc/networks/maayan-vidal)

b) C. elegans protein network 2007 (http://interactome.dfci.harvard.edu/C_elegans/index.php

c) C. elegans protein network 2004 (http://interactome.dfci.harvard.edu/C_elegans/index.php

d) Western US power grid (http://wwwpersonal.umich.edu/~mejn/netdata/power.zip)

e) R dependency network (http://www.mas.ncl.ac.uk/~ncsg3/blog/dependencies.csv)

Construct the histogram of the centrality measures and compare them. Use the measures

(i) degree,

(ii) eigenvector centrality,

(iii) pagerank and

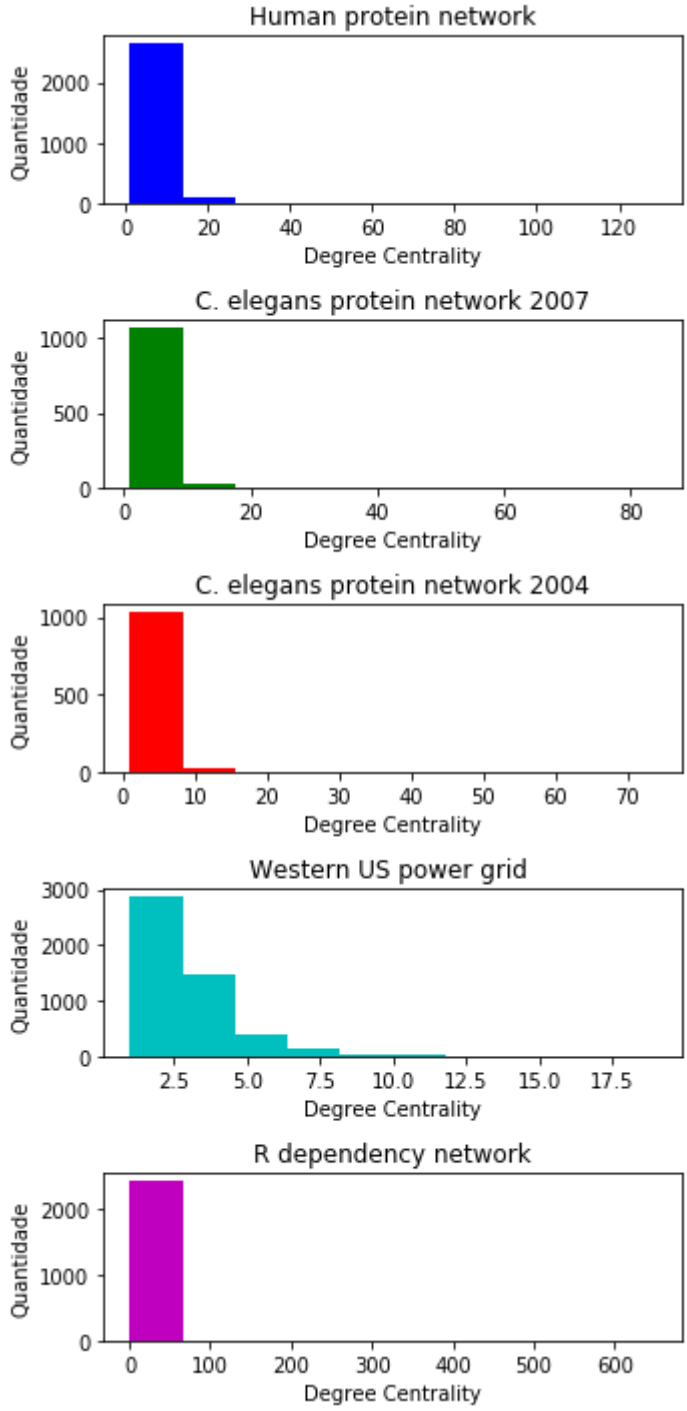
(v) closeness centrality.

Are there similarities between networks of the same type, such as among technological or biological networks

```
listaGraph = list()
listaIndex = ["Human protein network", "C. elegans protein network 2007", "C. elegans prot
 "Western US power grid ", "R dependency network"]

G_maayan_vidal = nx.read_edgelist("maayan-vidal.edges", nodetype=int)
```

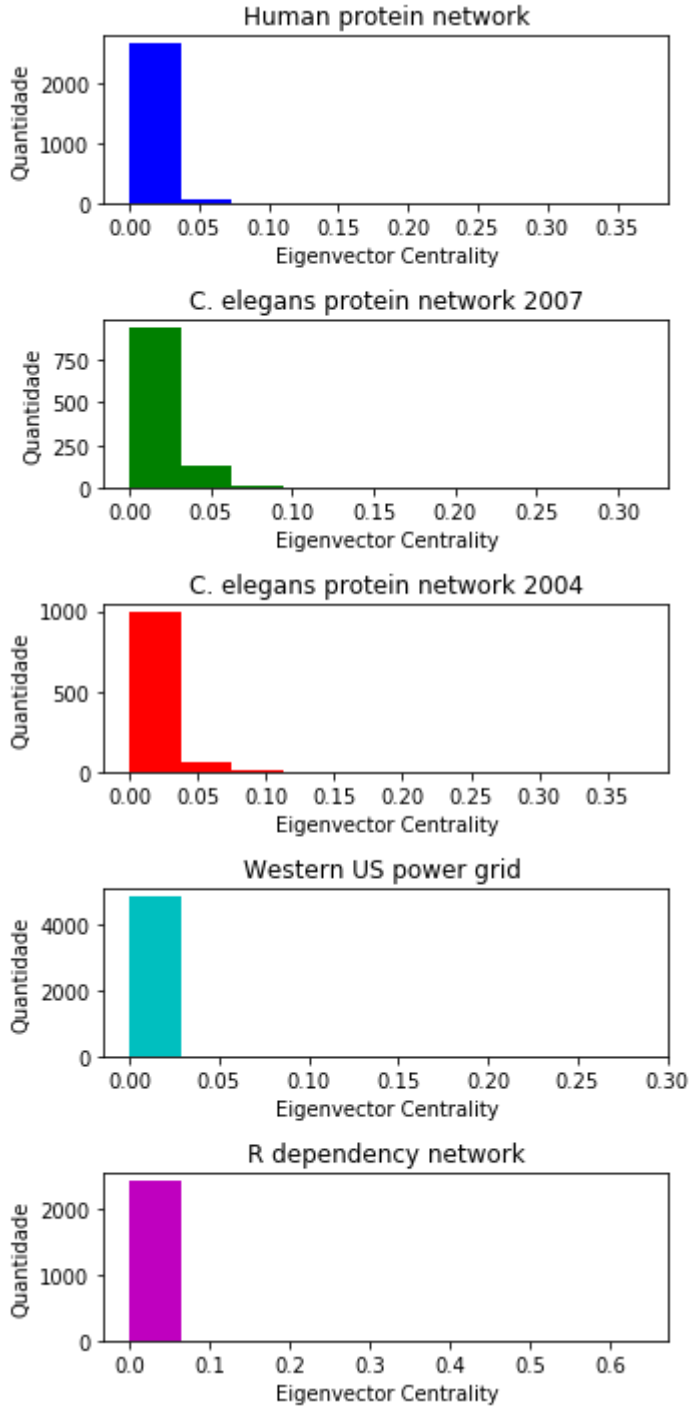
```
listaGraph.append(trata_rede(G_maayan_vidal))

G_protein_2007 = nx.read_edgelist("protein2007.edges")
listaGraph.append(trata_rede(G_protein_2007))

G_protein_2005 = nx.read_edgelist("protein2005.edges")
listaGraph.append(trata_rede(G_protein_2005))

G_powergrid = nx.read_edgelist("powergrid.edges", nodetype = int)
listaGraph.append(trata_rede(G_powergrid))

df = pd.read_csv("dependencies.csv")
G_R_dep = nx.Graph()
for i in range(len(df)): G_R_dep.add_edge(df.iloc[i,1],df.iloc[i,2])
listaGraph.append(trata_rede(G_R_dep))


print('Histogramas da distribuição de grau de acordo com cada rede')
f,a = plt.subplots(5,1, figsize=(5,10))
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w']
a = a.ravel()
for idx,ax in enumerate(a):
    ax.hist(degree_centrality(listaGraph[idx]), facecolor=colors[idx])
    ax.set_title(listaIndex[idx])
    ax.set_xlabel("Degree Centrality")
    ax.set_ylabel("Quantidade")
plt.tight_layout()
```

```
Histogramas da distribuição de grau de acordo com cada rede
```



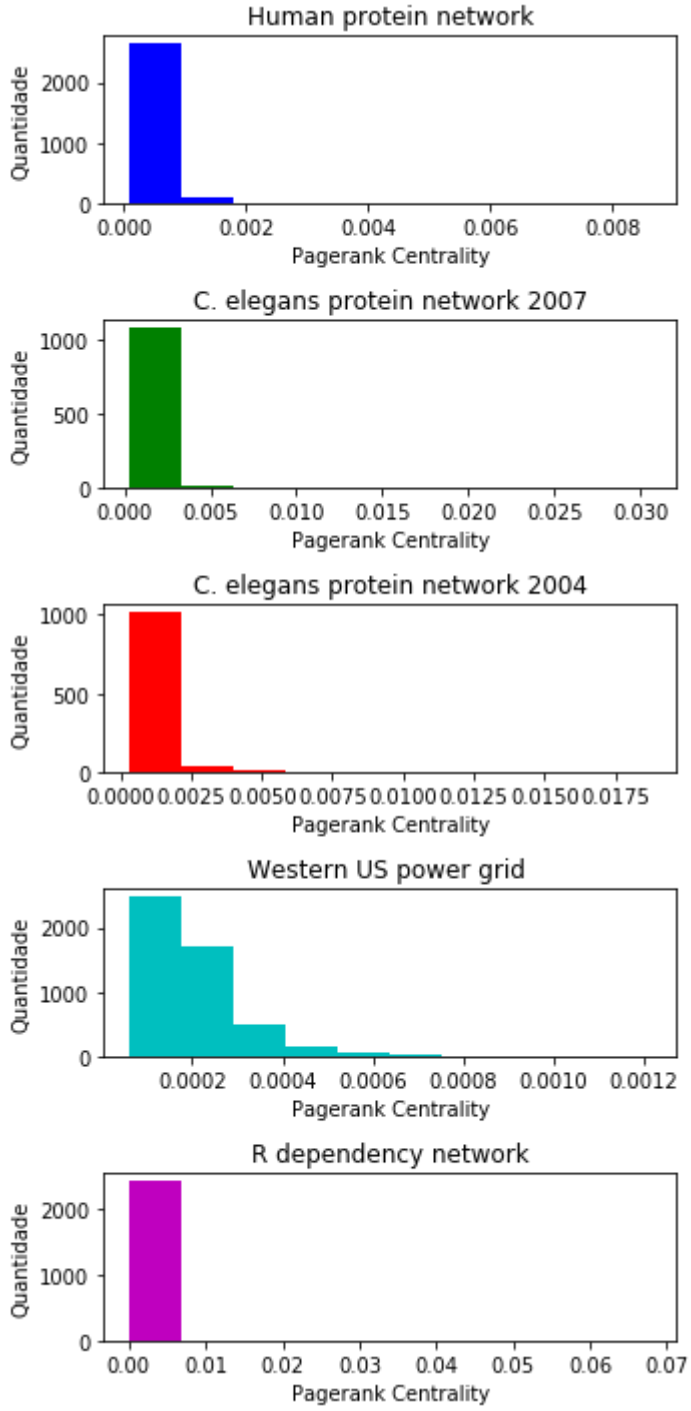

```
print('Histogramas da distribuição de eigenvector centrality de acordo com cada rede')
f,a = plt.subplots(5,1, figsize=(5,10))
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w']
a = a.ravel()
for idx,ax in enumerate(a):
    print(round(100*(idx/4),2),"%")
    ax.hist(eigenvector_centrality(listaGraph[idx]),  facecolor=colors[idx])
    ax.set_title(listaIndex[idx])
    ax.set_xlabel("Eigenvector Centrality")
    ax.set_ylabel("Quantidade")
    clear_output()
print('Histogramas da distribuição de eigenvector centrality de acordo com cada rede')
plt.tight_layout()
```

Histogramas da distribuição de eigenvector centrality de acordo com cada rede



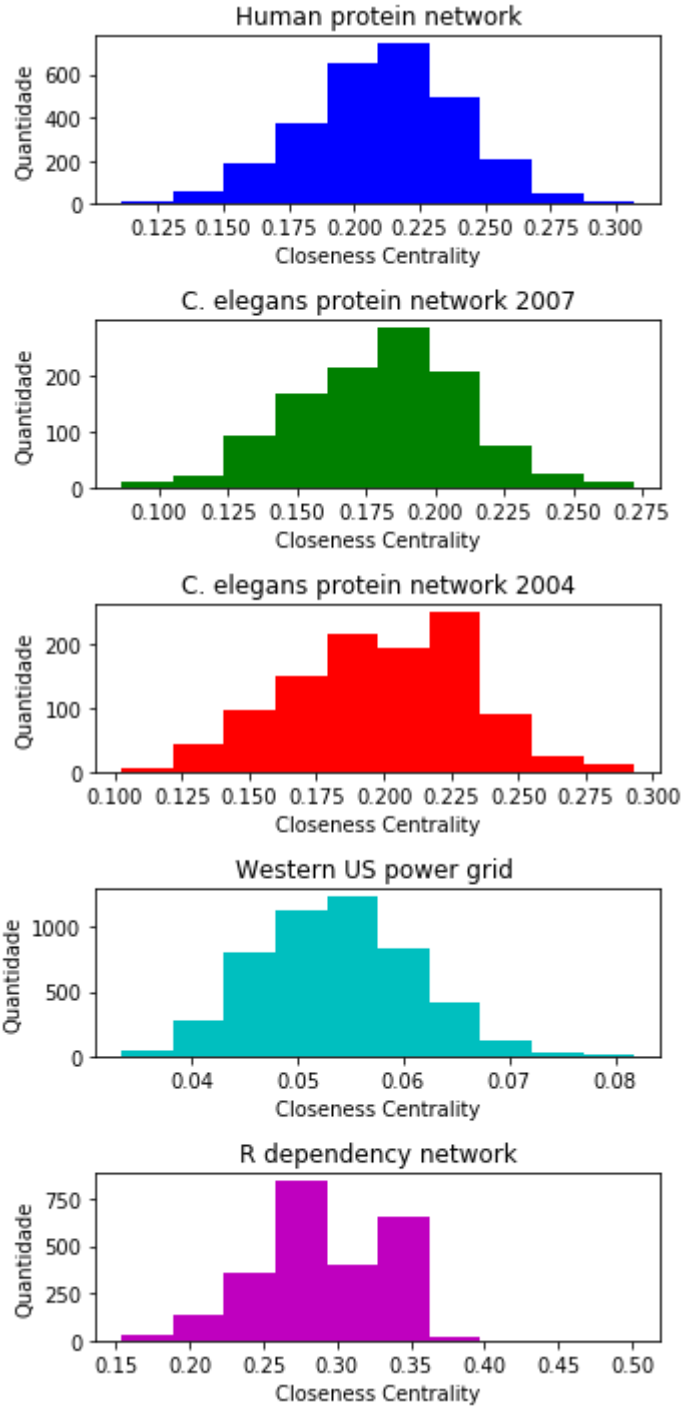

```
print('Histogramas da distribuição de pagerank centrality de acordo com cada rede')
f,a = plt.subplots(5,1, figsize=(5,10))
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w']
a = a.ravel()
for idx,ax in enumerate(a):
    print(round(100*(idx/4),2),"%")
    ax.hist(pagerank(listaGraph[idx]), facecolor=colors[idx])
    ax.set_title(listaIndex[idx])
    ax.set_xlabel("Pagerank Centrality")
    ax.set_ylabel("Quantidade")
    clear_output()
print('Histogramas da distribuição de pagerank centrality de acordo com cada rede')
plt.tight_layout()
```

Histogramas da distribuição de pagerank centrality de acordo com cada rede




```
print('Histogramas da distribuição de closeness centrality de acordo com cada rede')
f,a = plt.subplots(5,1, figsize=(5,10))
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w']
a = a.ravel()
for idx,ax in enumerate(a):
    print(round(100*(idx/4),2),"%")
    ax.hist(closeness_centrality(listaGraph[idx]), facecolor=colors[idx])
    ax.set_title(listaIndex[idx])
    ax.set_xlabel("Closeness Centrality")
    ax.set_ylabel("Quantidade")
    clear_output()
print('Histogramas da distribuição de closeness centrality de acordo com cada rede')
plt.tight_layout()
```

Histogramas da distribuição de closeness centrality de acordo com cada rede



4 – Consider the networks in the last exercise. For each network, conside

(i) degree,

(ii) k-core,

(iii) closeness centrality,

(iv) betweenness centrality,

(v) eigenvector centrality,

(vi) pagerank,

(vi) random walk accessibility,

(v) communicability centrality.

And for each measure, calculate:

(i) average,

(ii) standard deviation,

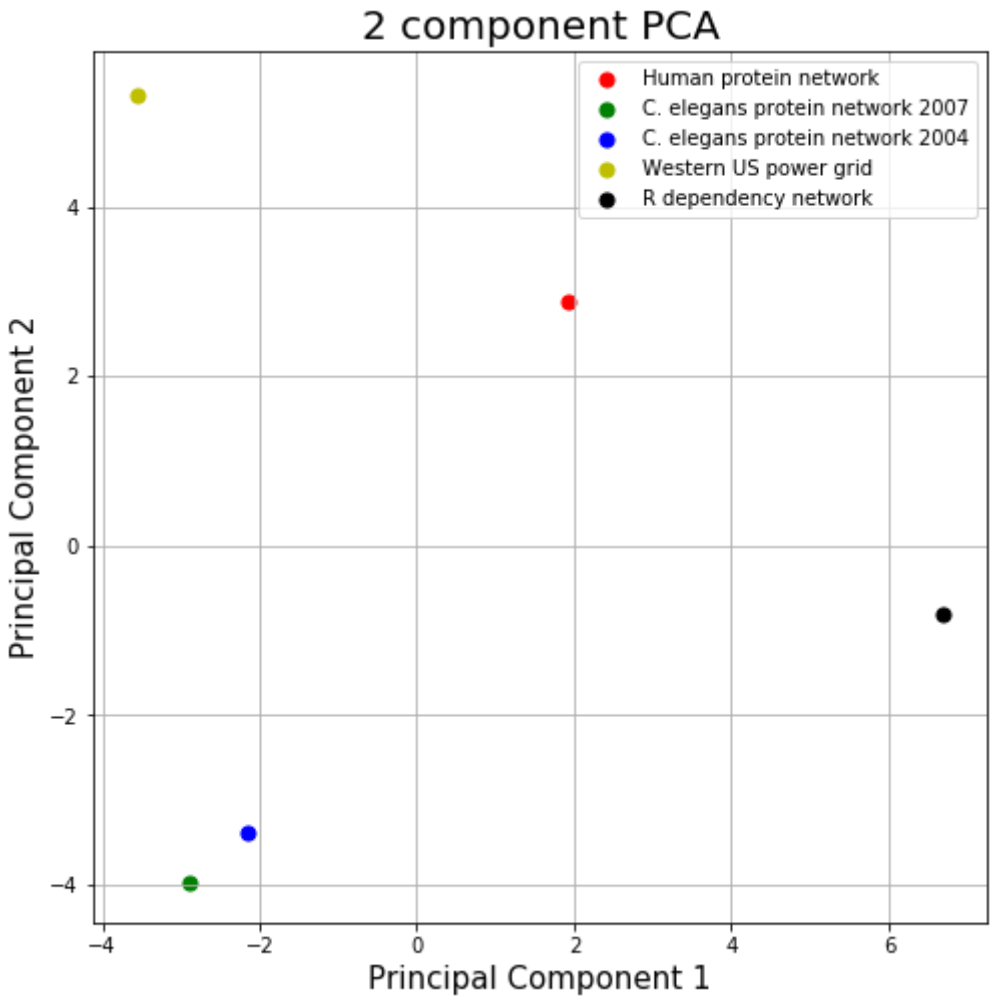
(iii) second moment,

(iii) Shannon entropy.

Thus, each network will be represented by a feature vector with 20 elements. Project the netw Principal Component Analysis (PCA). Discuss the similarities of the networks in terms of their

```
def momment_of_degree_distribution(m):
  return(np.sum(m**2)  / len(m))


def shannon_entropy(m):
  value,counts = np.unique(m, return_counts=True)
  pk = counts / counts.sum()
  H = st.entropy(counts, base=2)
  return H


def ex4(measure):
  return np.mean(measure), stats.stdev(measure) , momment_of_degree_distribution(measure)


matrix_values = np.zeros((5, 8*4))

for i in range(len(listaGraph)):
  G = listaGraph[i]
  print("Network ",listaIndex[i])

  VK = np.asarray(degree_centrality(G))
  matrix_values[i,0],matrix_values[i,1],matrix_values[i,2],matrix_values[i,3] = ex4(VK)
  print("\n Vk", VK)

  KC = np.asarray(kcore(G))
  matrix_values[i,4],matrix_values[i,5],matrix_values[i,6],matrix_values[i,7] = ex4(KC)
  print("\n KC", KC)

  CLC = np.asarray(closeness_centrality(G))
  matrix_values[i,8],matrix_values[i,9],matrix_values[i,10],matrix_values[i,11] = ex4(CLC)
  print("\n CLC", CLC)

  B = np.asarray(betweenness_centrality(G))
  matrix_values[i,12],matrix_values[i,13],matrix_values[i,14],matrix_values[i,15] = ex4(B)
  print("\n B", B)
```

```
  EC = np.asarray(eigenvector_centrality(G))
  matrix_values[i,16],matrix_values[i,17],matrix_values[i,18],matrix_values[i,19] = ex4(EC
  print("\n EC", EC)

  PR = np.asarray(pagerank(G))
  matrix_values[i,20],matrix_values[i,21],matrix_values[i,22],matrix_values[i,23] = ex4(PR
  print("\n PR", PR)

  RWA = np.asarray(random_walk_accessibility(G))
  matrix_values[i,24],matrix_values[i,25],matrix_values[i,26],matrix_values[i,27] = ex4(RW
  print("\n RWA", RWA)

  CC = np.asarray(communicability_centrality(G))
  matrix_values[i,28],matrix_values[i,29],matrix_values[i,30],matrix_values[i,31] = ex4(CC
  print("\n CC", CC)

  clear_output()


df = pd.DataFrame(matrix_values)
names = pd.DataFrame(listaIndex)

# Standardizing the features
x = StandardScaler().fit_transform(df)

pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents
                           , columns = ['principal component 1', 'principal component 2'])

finalDf = pd.concat([principalDf, names],axis=1)


fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets = ["Human protein network", "C. elegans protein network 2007", "C. elegans proteir
 "Western US power grid ", "R dependency network"]
colors = ['r', 'g', 'b', 'y', 'k']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf[0] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```



5 – For the following networks:

a) E-road network (http://konect.cc/networks/subelj_euroroad),

b) C. elegans neural network (http://www-personal.umich.edu/~mejn/netdata/celegansneural

c) US airport network (http://toreopsahl.com/datasets/#usairports)

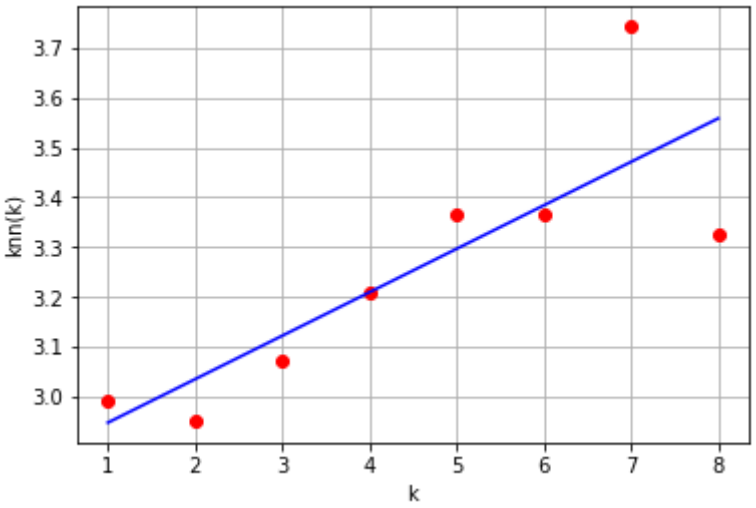
d) Human protein network (http://konect.cc/networks/maayan-vidal)

Obtain the scatterplot of knn(k) X k and calculate the Pearson correlation coefficient between assortativity coefficient. Comment the results.

```
listaGraph = list()
listaIndex = ["E-road network", "C. elegans neural network", "US airport network ", "Humar

G_eroad = nx.read_edgelist("eroad.edges", nodetype=int, data=(('weight',int),))
listaGraph.append(trata_rede(G_eroad))

G_celneural = nx.read_gml("celegansneural.gml")
listaGraph.append(trata_rede(G_celneural, multigraph = True))

G_usaairport = nx.read_edgelist("usaairport.edges", nodetype=int, data=(('weight',int),))
```

```
listaGraph.append(trata_rede(G_usaairport))

G_maayan_vidal = nx.read_edgelist("maayan-vidal.edges", nodetype=int)
listaGraph.append(trata_rede(G_maayan_vidal))



def ex5(G):
  knn = []
  for i in G.nodes():
      aux =  nx.average_neighbor_degree(G, nodes = [i])
      knn.append(float(aux[i]))
  knn = np.array(knn)
  vk = dict(G.degree())
  vk = list(vk.values())
  knnk = list()
  ks = list()
  for k in arange(np.min(vk), np.max(vk)):
      aux = vk == k
      if(len(knn[aux]) > 0):
          av_knn = mean(knn[aux]) #average clustering among all the nodes with degree k
          knnk.append(av_knn)
          ks.append(k)
  plt.plot(ks, knnk, 'ro')
  #plt.loglog(ks,knnk,'bo',basex=10,basey=10)
  #plt.title("Average neighborhood connectivity vs degree")
  plt.ylabel("knn(k)")
  plt.xlabel("k")
  plt.grid(True)
  #plt.savefig('knnk.eps')

  # determine best fit line
  par = np.polyfit(ks, knnk, 1, full=True)
  slope=par[0][0]
  intercept=par[0][1]
  xl = [min(ks), max(ks)]
  yl = [slope*xx + intercept  for xx in xl]
  plt.plot(xl, yl, '-b')

  #plt.savefig('knn.eps') #save the figure into a file
  plt.show(True)

  rho = corrcoef(ks, knnk)[0,1]
  print('Pearson correlation coefficient:', rho)

  r=nx.degree_assortativity_coefficient(G)
  print("Assortativity = ","%3.4f"%r)
  return rho,r


matrix_values = np.zeros((4,2))

print(listaIndex[0])
matrix_values[0,0],matrix_values[0,1] = ex5(listaGraph[0])
```

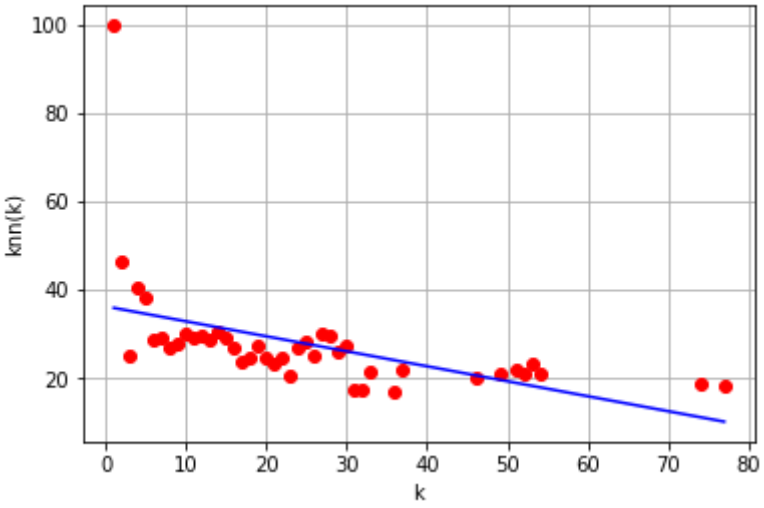
E-road network




```
Pearson correlation coefficient: 0.8296047846722235
Assortativity =  0.0900
```

```
print(listaIndex[1])
matrix_values[1,0],matrix_values[1,1] = ex5(listaGraph[1])
```

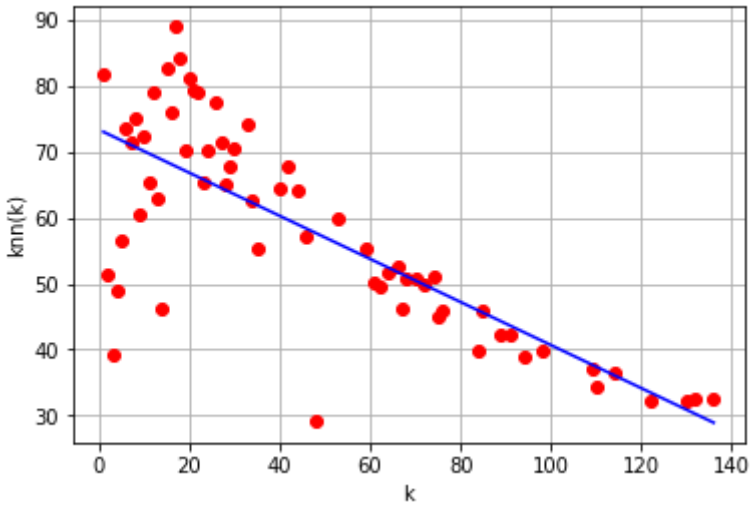
C. elegans neural network




```
Pearson correlation coefficient: -0.4947811319324024
Assortativity =  -0.1632
```

```
print(listaIndex[2])
matrix_values[2,0],matrix_values[2,1] = ex5(listaGraph[2])
```

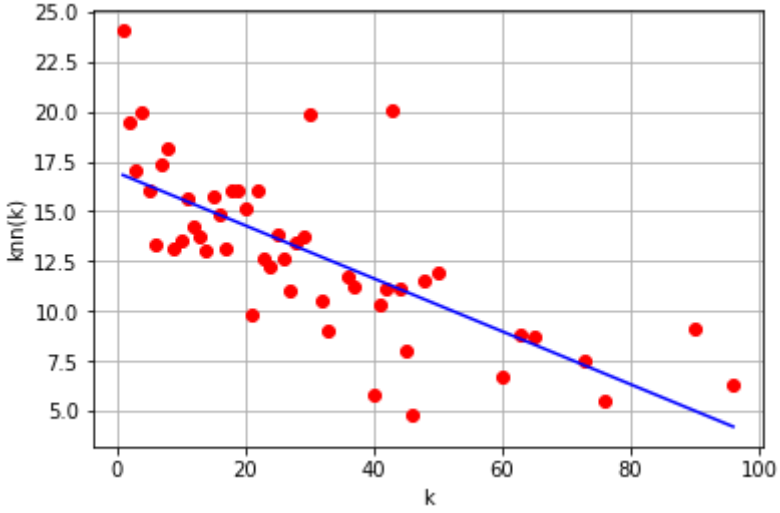
```
US airport network
```




```
Pearson correlation coefficient: -0.7756491513624391
Assortativity =  -0.2679
```

```
print(listaIndex[3])
matrix_values[3,0],matrix_values[3,1] = ex5(listaGraph[3])
```

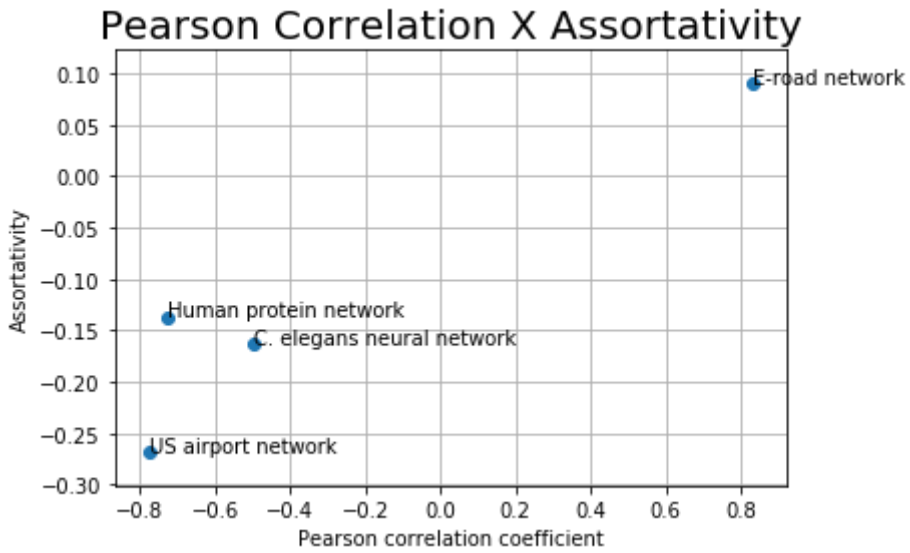
```
Human protein network
```




```
Pearson correlation coefficient: -0.7264791351157311
Assortativity =  -0.1366
```

```
df = pd.DataFrame(matrix_values, columns = ["Pearson correlation coefficient", "Assortativ
names = pd.DataFrame(listaIndex, columns = ["Networks"])
df = pd.concat([df, names],axis=1)
```

```
fig, ax = plt.subplots()
ax.scatter(df["Pearson correlation coefficient"], df["Assortativity"])
ax.set_title('Pearson Correlation X Assortativity', fontsize = 20)
plt.ylabel("Assortativity")
plt.xlabel("Pearson correlation coefficient")
plt.grid(True)
for i, txt in enumerate(listaIndex):
    ax.annotate(txt, (df["Pearson correlation coefficient"][i], df["Assortativity"][i]))
```



6 – Construct the Givan-Newman benchmark (N=128, kin+kout = 16 and

You can use:
https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.a
s.LFR_benchmark_graph.html

Compare the following methods:

Fastgreedy:
https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.a
y_modularity_communities.html#networkx.algorithms.community.modularity_max.greedy_mo

Label propagation:
https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.a
el_propagation_communities.html#networkx.algorithms.community.label_propagation.label_p

Givan Newman:
https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.a
man.html#networkx.algorithms.community.centrality.girvan_newman

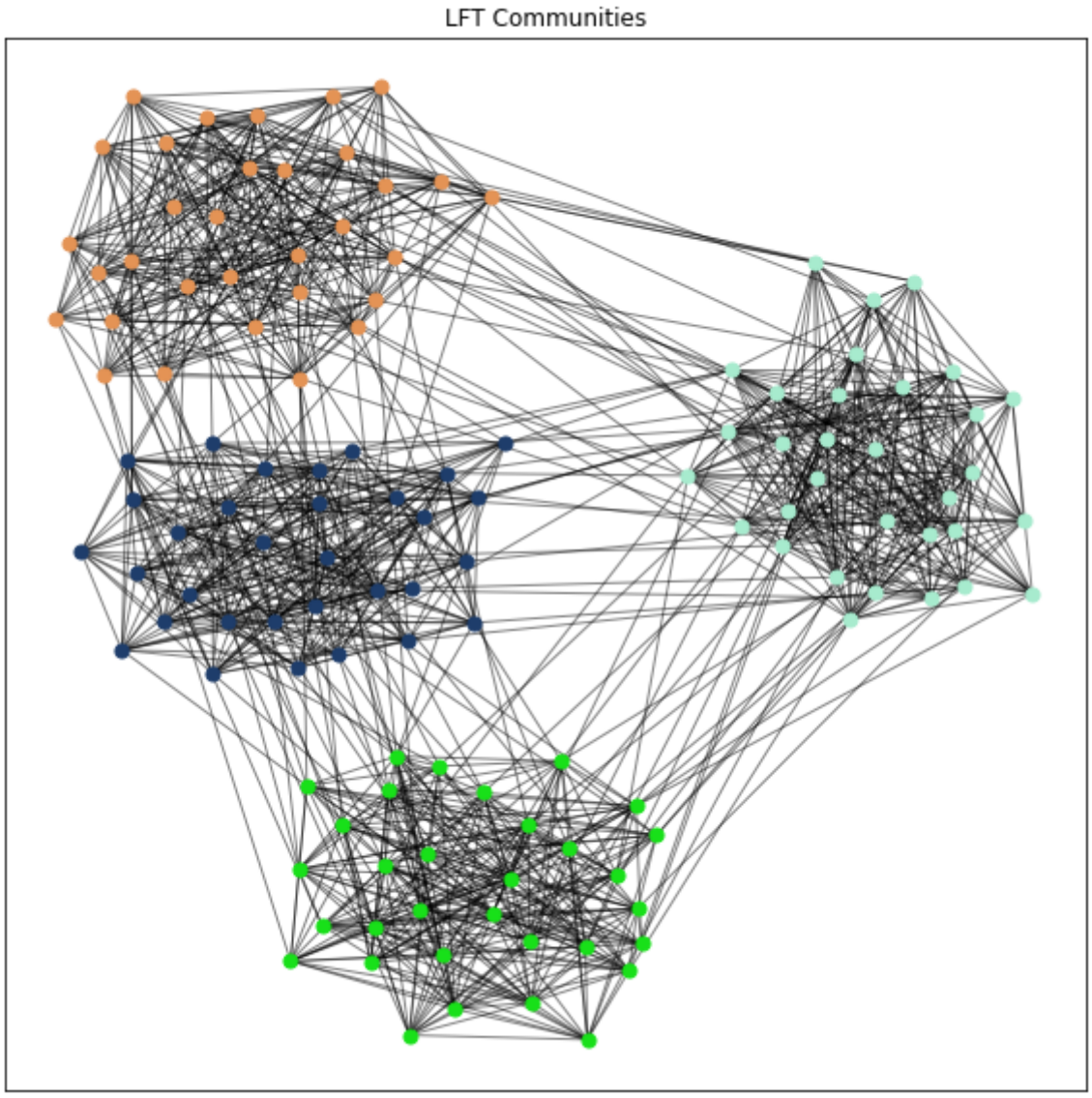
Louvain: https://github.com/taynaud/python-louvain

Use the normalized mutual information as a measure of accuracy.

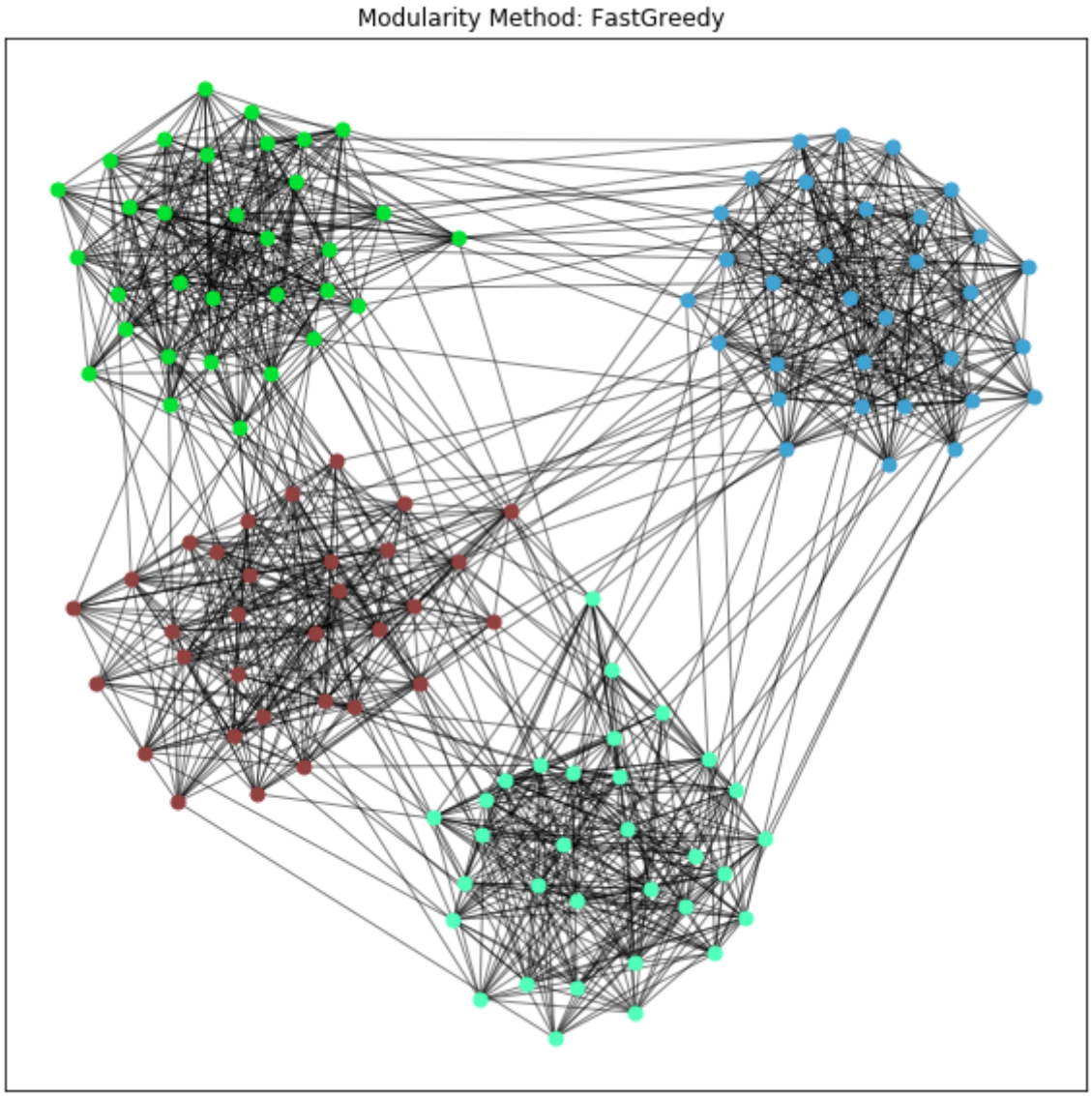
```
def drawing_graph_and_communities(G,partition, T):
  #drawing
  plt.figure(figsize=(10,10))
  colors = []
  for i in range(max(list(partition.values())) + 5):
    colors.append('#%06X' % randint(0, 0xFFFFFF))
  #colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w']
  size = float(len(set(partition.values())))
```

```
  count = 0.
  pos=nx.spring_layout(G)
  for com in set(partition.values()) :
      count = count + 1.
      list_nodes = [nodes for nodes in partition.keys() if partition[nodes] == com]
      nx.draw_networkx_nodes(G, pos, list_nodes, node_size = 50, node_color = colors[int(c
  nx.draw_networkx_edges(G, pos, alpha=0.5)
  plt.title(T)
  plt.show()


def greedy(G, draw = True):
  p = greedy_modularity_communities(G)

  p = ([list(x) for x in p])

  partition_greedy = {}
  for i in range(len(p)):
    for j in range(len(p[i])):
      partition_greedy[p[i][j]] = i

  if draw: drawing_graph_and_communities(G,partition_greedy,"Modularity Method: FastGreedy
  return partition_greedy


def label(G, draw = True):
  p = nx.algorithms.community.label_propagation.label_propagation_communities(G)
  p = ([list(x) for x in p])

  partition_label = {}
  for i in range(len(p)):
    for j in range(len(p[i])):
      partition_label[p[i][j]] = i

  if draw: drawing_graph_and_communities(G,partition_label, "Modularity Method: Label Prop
  return partition_label


def givan_newman_method(G,k, draw = True):
  communities = community.girvan_newman(G)
  for i in arange(0, k-1):
      next_level_communities = next(communities)
  p = sorted(map(sorted, next_level_communities))

  partition_givan_newman = {}
  for i in range(len(p)):
    for j in range(len(p[i])):
      partition_givan_newman[p[i][j]] = i
  if draw :  drawing_graph_and_communities(G,partition_givan_newman, "Modularity Method: G
  return partition_givan_newman


def louvain(G, draw = True):
  partition_lovain = community_louvain.best_partition(G)
  if draw: drawing_graph_and_communities(G,partition_lovain, "Modularity Method: Louvain "
  return partition_lovain
```

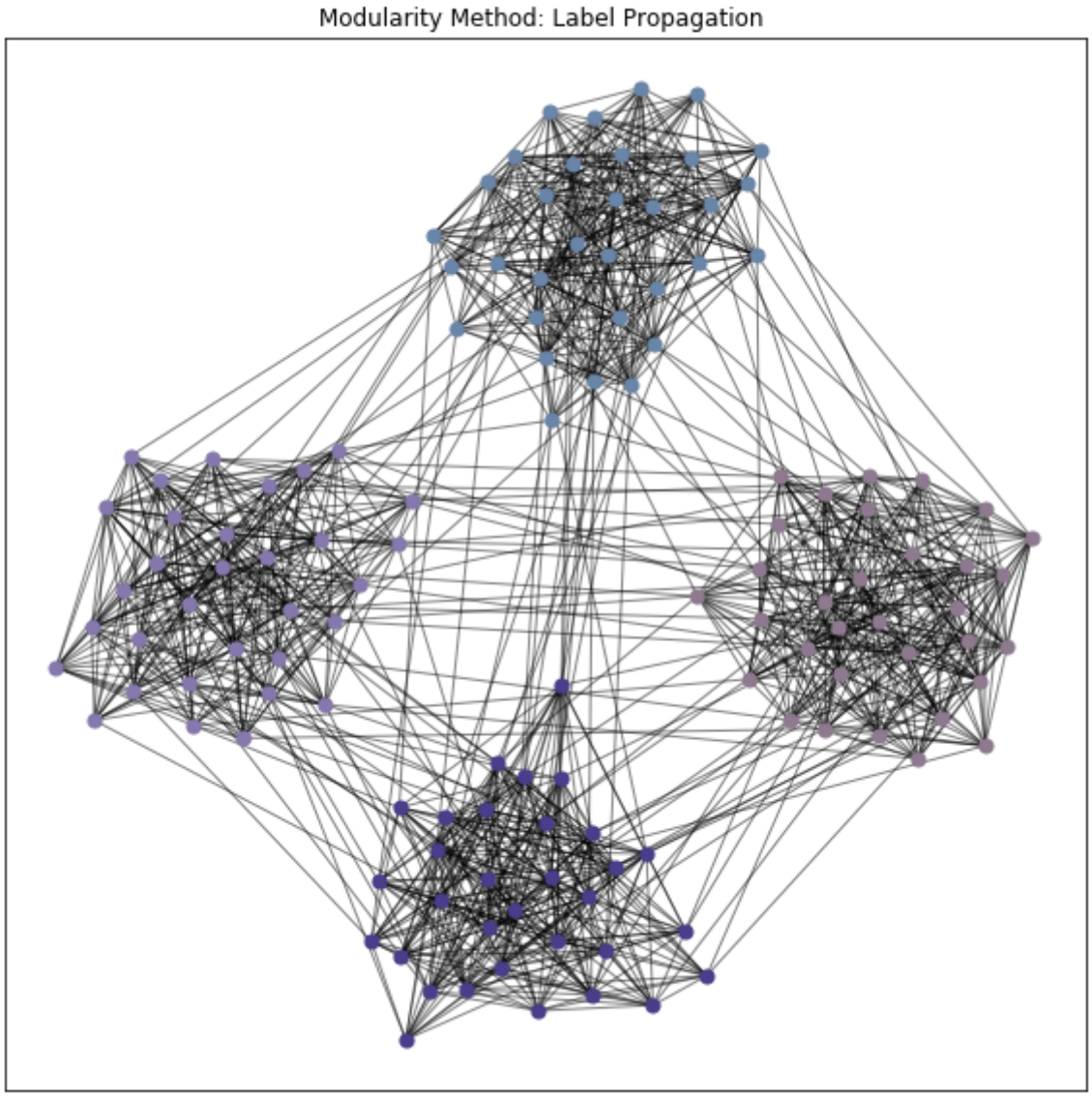
```
N = 128
tau1 = 3
tau2 = 1.5
mu = 0.04
k = 16
minc = 32
maxc = 32
G_benchmark_graph = LFR_benchmark_graph(n = N, tau1 = tau1, tau2 = tau2, mu = mu, min_degr
                        max_degree = k, min_community=minc, max_community = maxc, seed = 1
communities = {frozenset(G_benchmark_graph.nodes[v]['community']) for v in G_benchmark_gra
p = ([list(x) for x in communities])

partition_communities = {}
for i in range(len(p)):
  for j in range(len(p[i])):
    partition_communities[p[i][j]] = i
drawing_graph_and_communities(G_benchmark_graph,partition_communities, "LFT Communities")
```



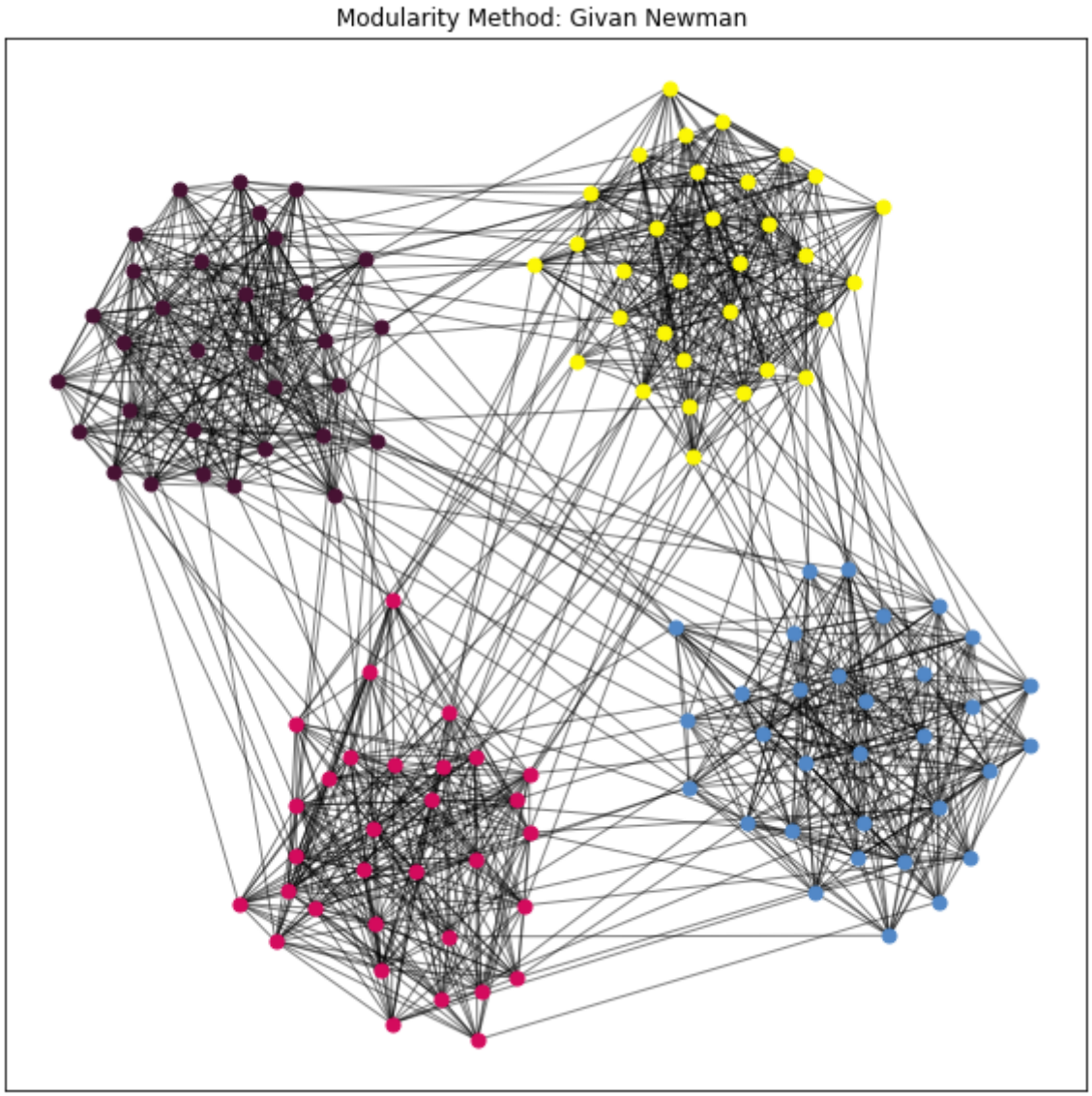

```
listaClusters = list()
partition_greedy = greedy(G_benchmark_graph)
listaClusters.append(partition_greedy)
```



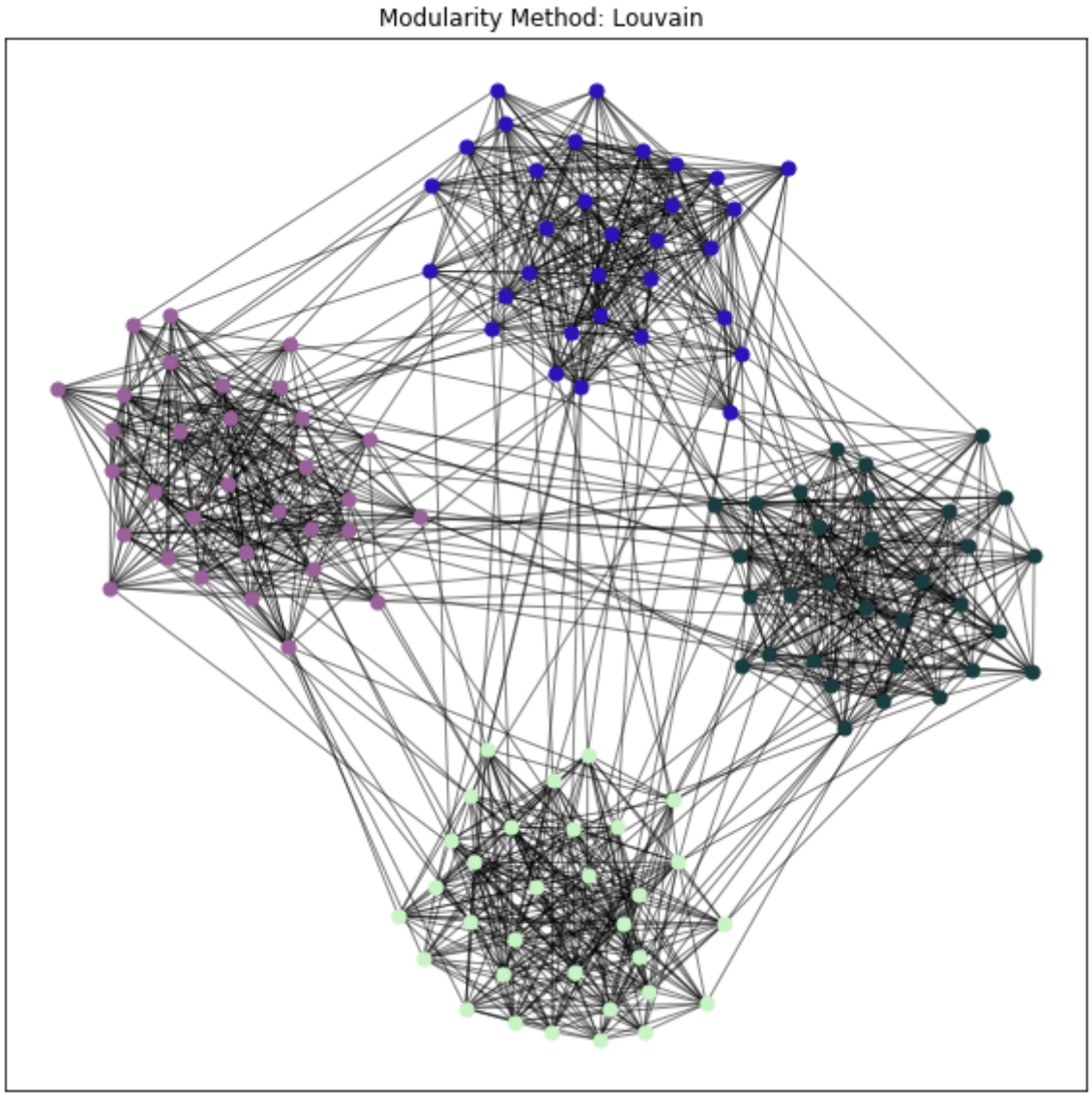

```
partition_label = label(G_benchmark_graph)
listaClusters.append(partition_label)
```




```
partition_givan_newman = givan_newman_method(G_benchmark_graph,4)
listaClusters.append(partition_givan_newman)
```




```
partition_louvain = louvain(G_benchmark_graph)
listaClusters.append(partition_louvain)
```




```
matrix_values = np.zeros((len(listaClusters[0]),len(listaClusters) ))
for i in range(len(listaClusters)):
  for j in range(len(listaClusters[i])):
    matrix_values[j,i] = listaClusters[i][j]
df = pd.DataFrame(matrix_values)
matrix_values = np.zeros((len(partition_communities)))
for i in range(len(partition_communities)):
  matrix_values[i] = partition_communities[i]
matrix_final = np.zeros((4))
for i in range(4):
  matrix_final[i] = normalized_mutual_info_score(df[i],matrix_values)
final_df = pd.DataFrame(matrix_final, columns = ["Communities"], index = ["FastGreedy", "L
print("Normalized Mutual Information Score between each clustering method")
final_df
```

Normalized Mutual Information Score between each clustering method

	Communities
FastGreedy	1.0
Label_propagation	1.0
Givan_Newman	1.0
Louvain	1.0

7 –Consider the methods for community identification:

Netcarto: https://amaral.northwestern.edu/resources/software/netcarto

Informap: https://www.mapequation.org/code.html

Fastgreedy:
https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.a
y_modularity_communities.html#networkx.algorithms.community.modularity_max.greedy_mo

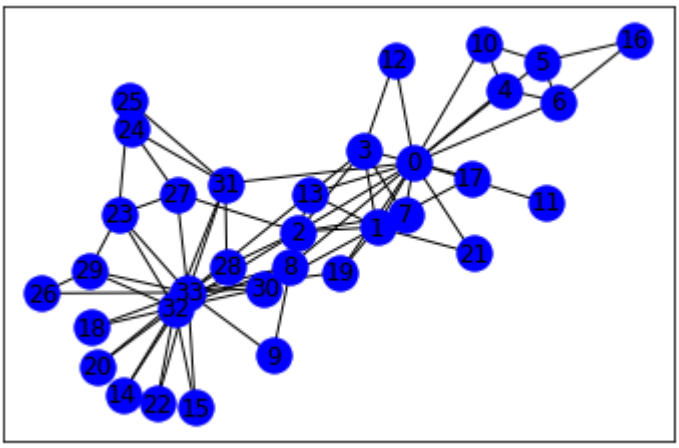
Label propagation:
https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.a
el_propagation_communities.html#networkx.algorithms.community.label_propagation.label_p

Givan Newman:
https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.a
man.html#networkx.algorithms.community.centrality.girvan_newman

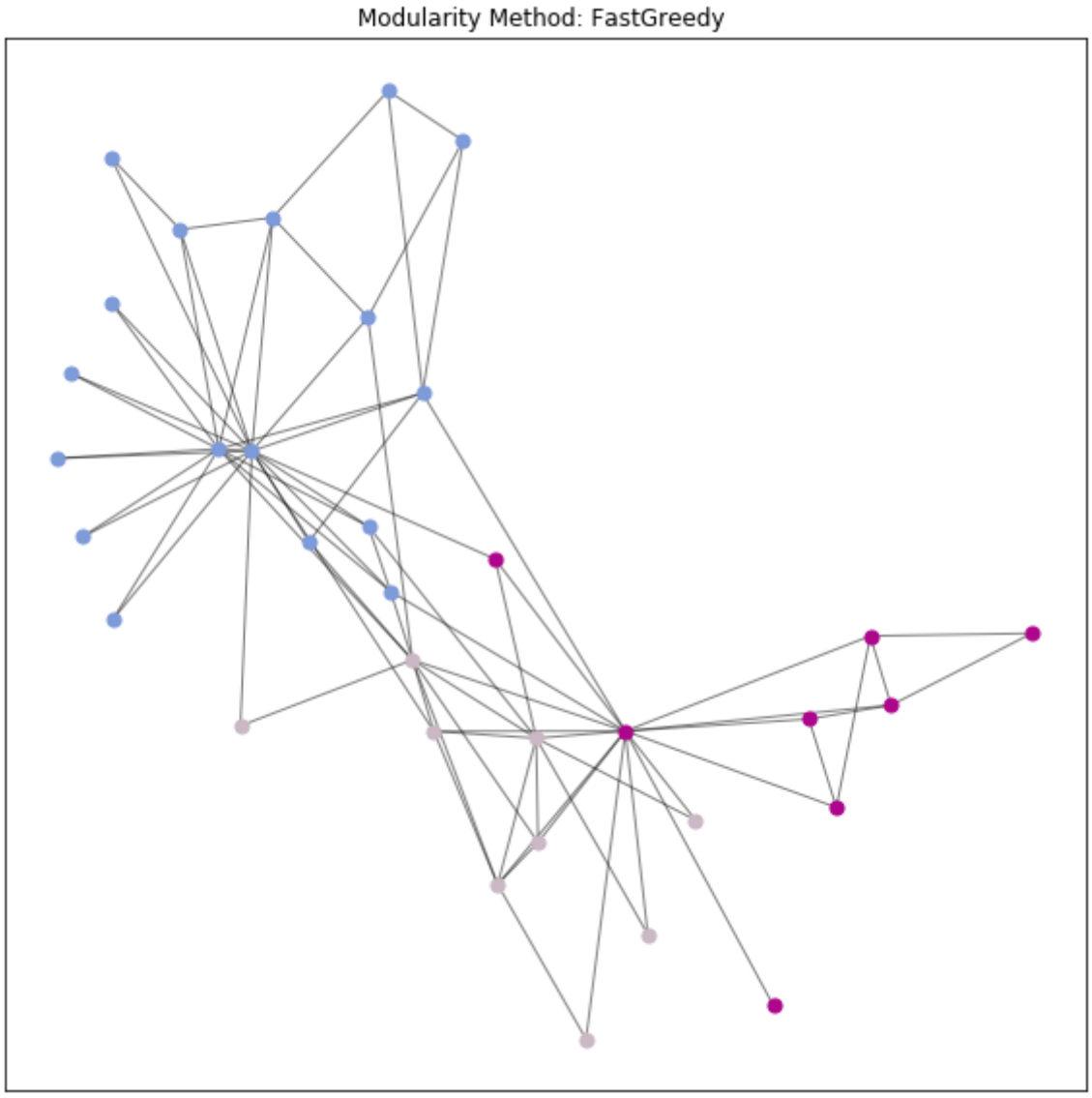
Louvain: https://github.com/taynaud/python-louvain

Show the partition of the Zachary karate club for these networks.

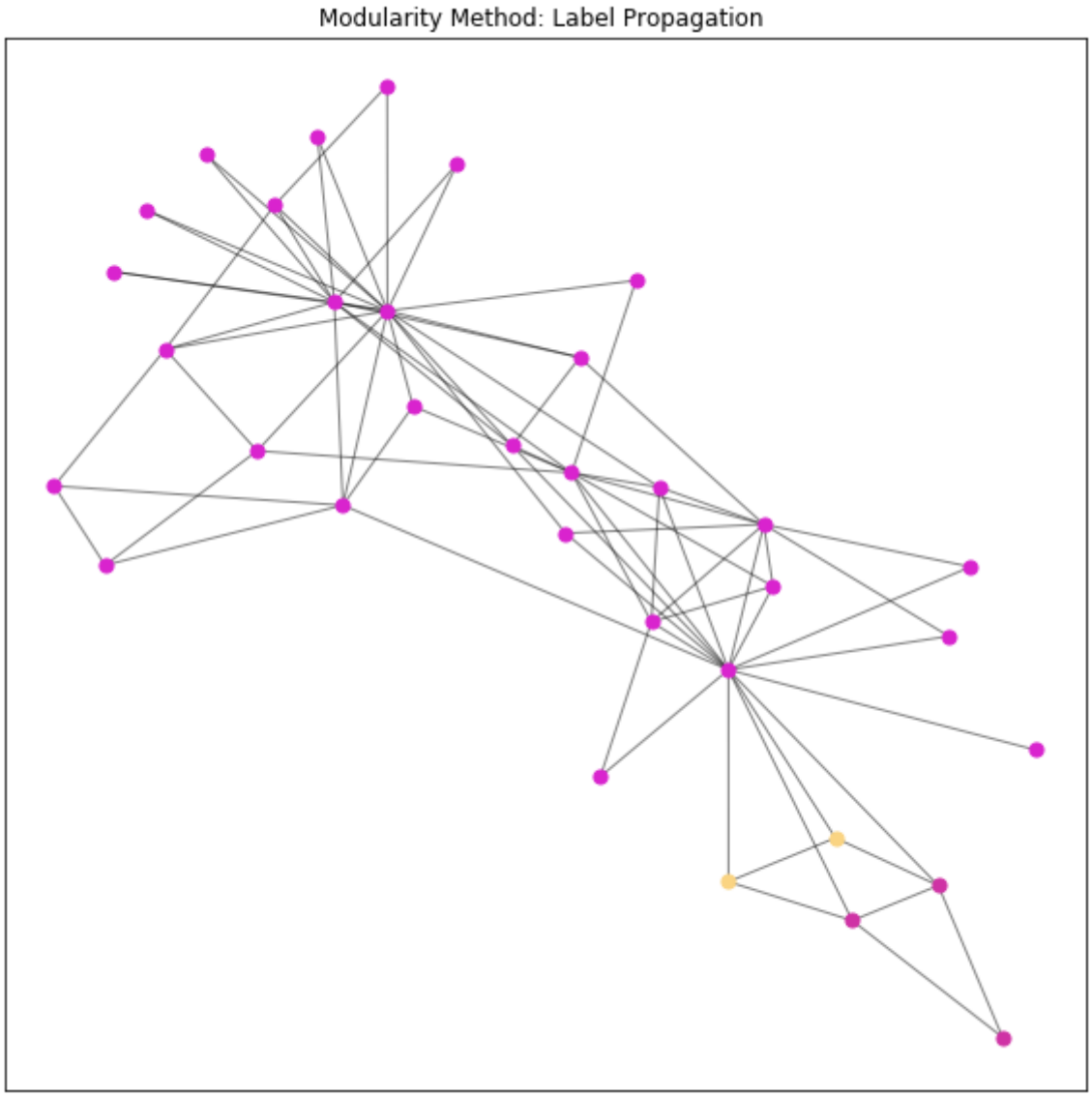
```
G=nx.karate_club_graph()
G = trata_rede(G)
pos=nx.nx.fruchterman_reingold_layout(G)
nx.draw_networkx(G, pos=pos, node_color = 'b')
plt.show(True)
```



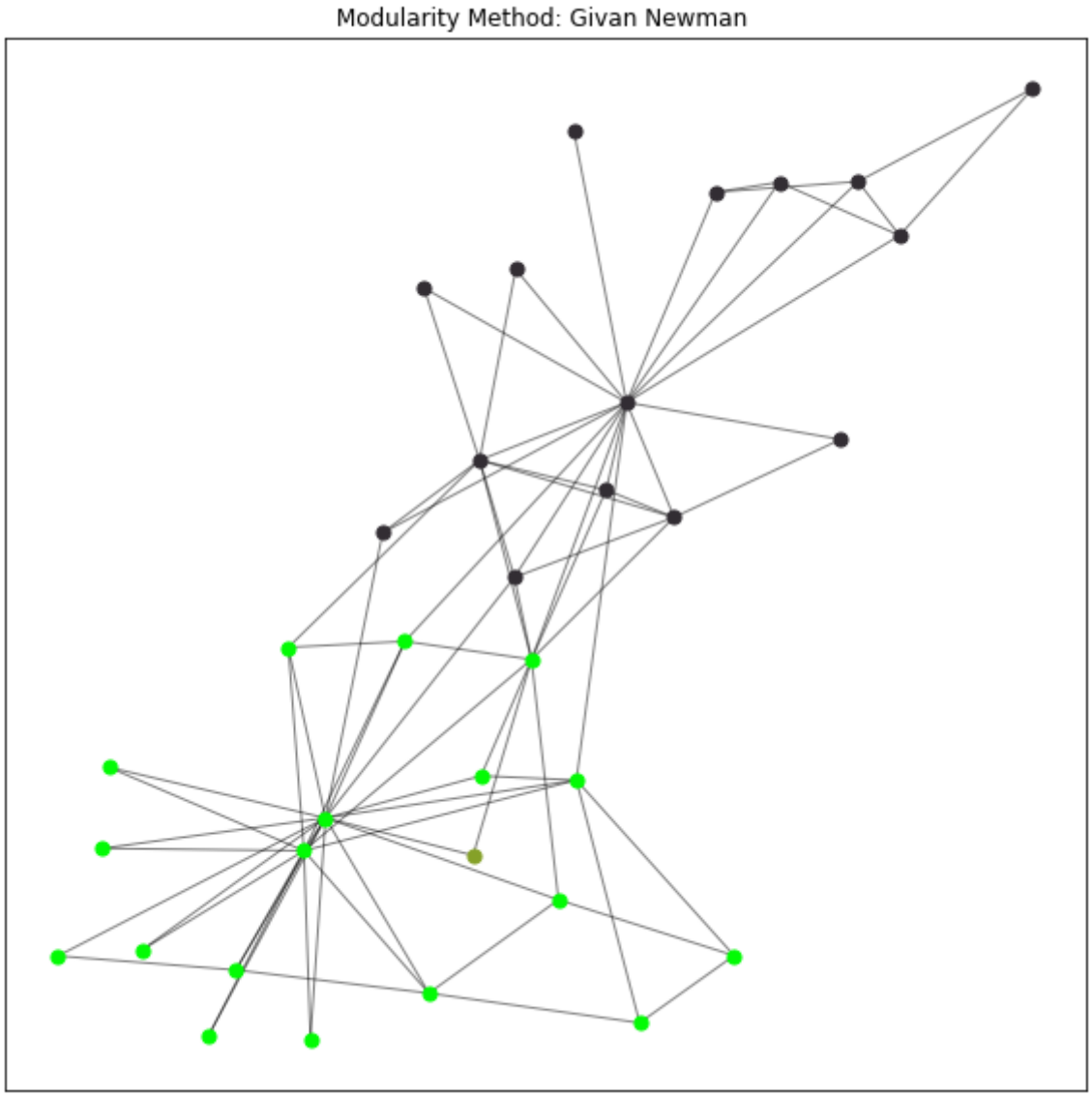

```
listaClusters = list()
partition_netcarto = np.zeros((len(G)))
listaClusters.append(partition_netcarto)


partition_informap = np.zeros((len(G)))
listaClusters.append(partition_informap)


partition_greedy = greedy(G)
listaClusters.append(partition_greedy)
```



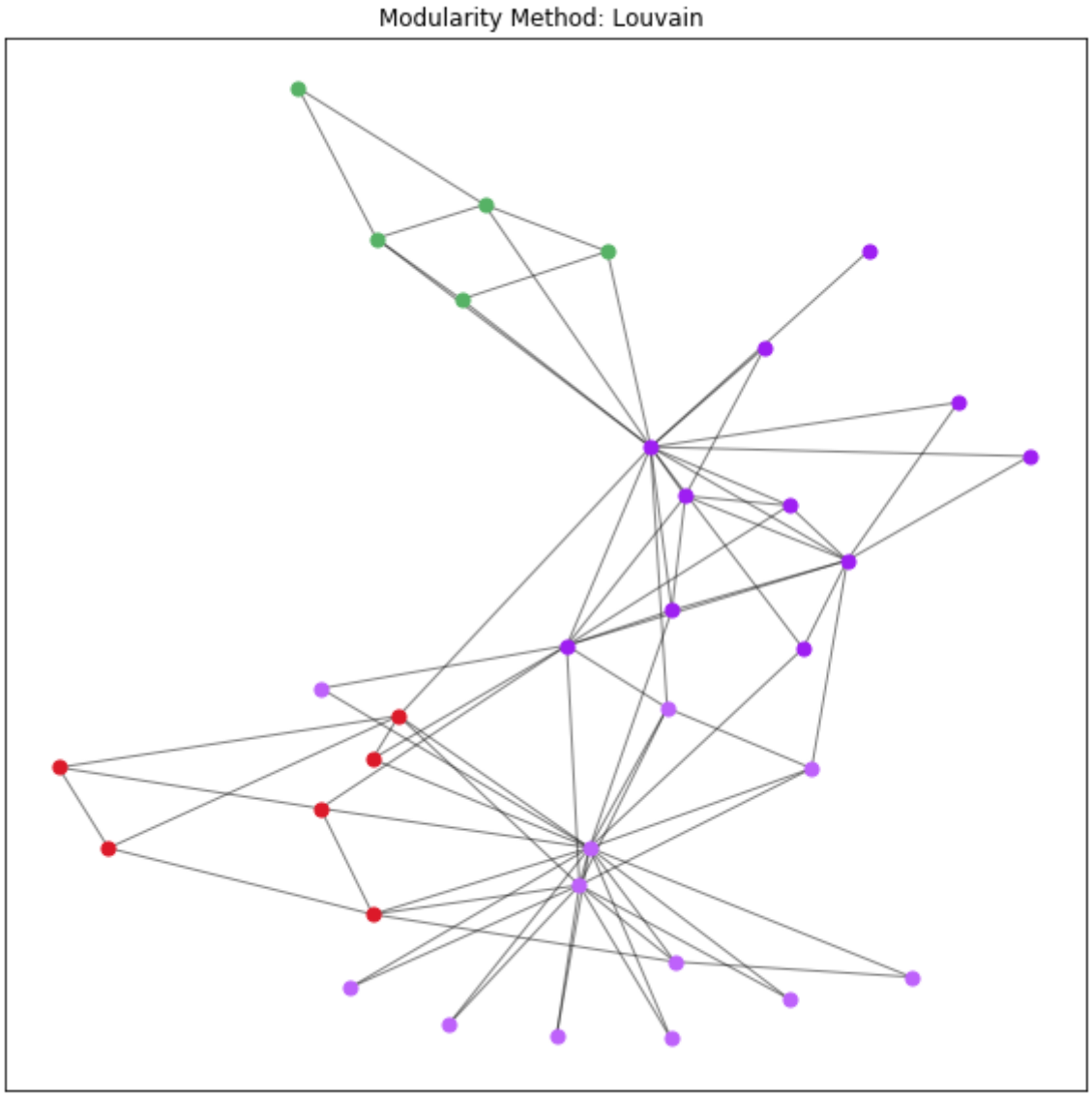

```
partition_label = label(G)
listaClusters.append(partition_label)
```




```
partition_givan_newman = givan_newman_method(G,3)
listaClusters.append(partition_givan_newman)
```

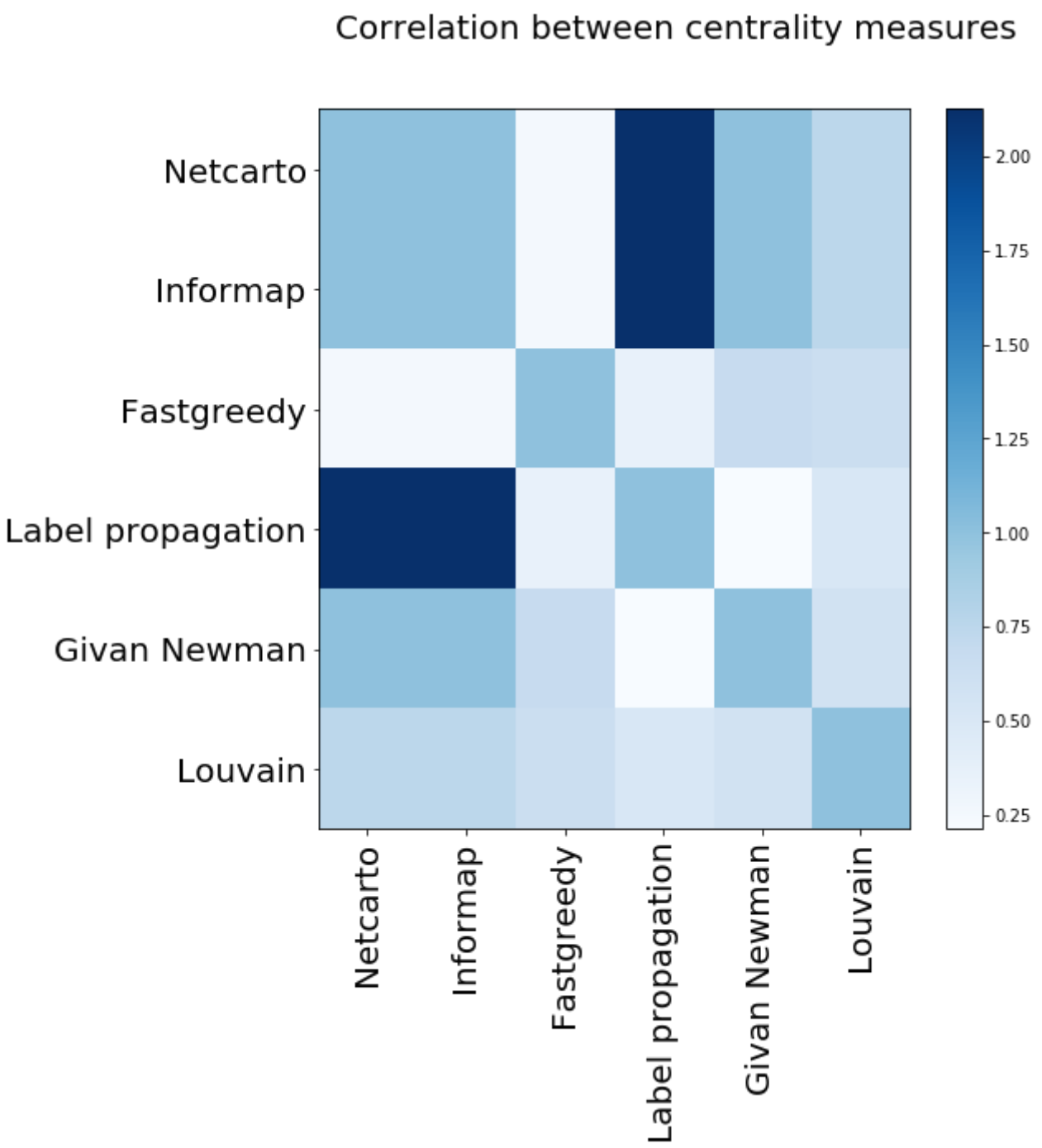



```
partition_louvain = louvain(G)
listaClusters.append(partition_louvain)
```




```
def corrplot_NMI_between_modularities(listaClusters, T):
  matrix_values = np.zeros((len(listaClusters[0]),len(listaClusters) ))
  for i in range(len(listaClusters)):
    for j in range(len(listaClusters[i])):
      matrix_values[j,i] = listaClusters[i][j]
  df = pd.DataFrame(matrix_values)
  from sklearn.metrics.cluster import normalized_mutual_info_score
  matrix_values = np.zeros((len(listaClusters),len(listaClusters)))
  for i in range(len(listaClusters)):
    for j in range(len(listaClusters)):
      matrix_values[i,j] = normalized_mutual_info_score(df[i],df[j])
  final_df = pd.DataFrame(matrix_values, index = T, columns = T)


  corr = final_df
  plt.figure(figsize=(8,8))
  plt.imshow(corr, cmap='Blues', interpolation='none', aspect='auto')
  plt.colorbar()
  plt.xticks(range(len(corr)), corr.columns, rotation='vertical', fontsize=20)
  plt.yticks(range(len(corr)), corr.columns, fontsize=20);
  plt.suptitle('NMI between centrality measures', fontsize=20)
  plt.grid(False)
```

```
  plt.show()

corrplot_NMI_between_modularities(listaClusters
          ,["Netcarto", "Informap","Fastgreedy", "Label propagation", "Givan Newman", "Lou
```



8 – Consider the Fortunato benchmark:

https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.as.LFR_benchmark_graph.html

Compare the following methods. Consider the Normalized Mutual Information (NMI) measure

Netcarto: https://amaral.northwestern.edu/resources/software/netcarto

Informap: https://www.mapequation.org/code.html

Fastgreedy:
https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.a
y_modularity_communities.html#networkx.algorithms.community.modularity_max.greedy_mo

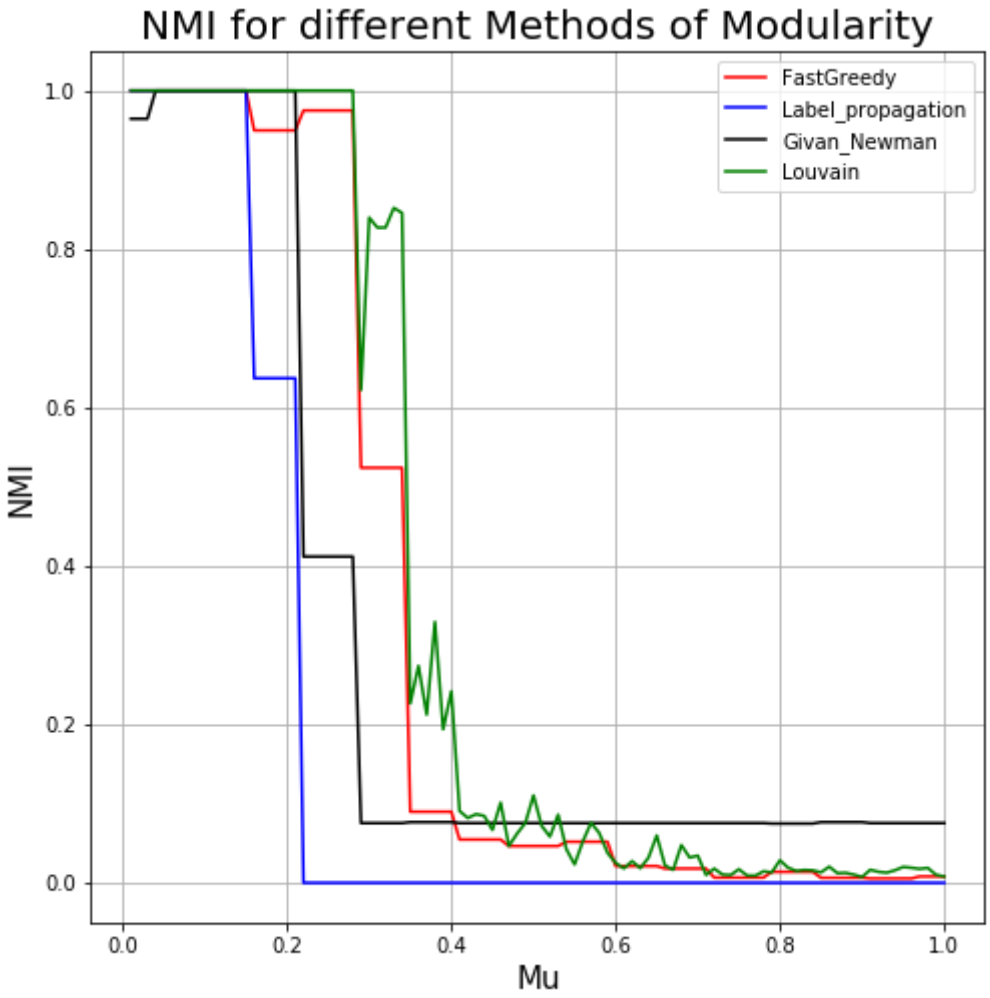
Label propagation:
https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.a
el_propagation_communities.html#networkx.algorithms.community.label_propagation.label_p

Givan Newman:
https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.a
man.html#networkx.algorithms.community.centrality.girvan_newman

Louvain: https://github.com/taynaud/python-louvain

```
N = 128
tau1 = 3
tau2 = 1.5
#mu = 0.04
k = 16
minc = 32
maxc = 32
listaGraph = list()
listaMu_x = list()

for i in range(0,100):
  mu = i/100
  listaMu_x.append(mu)
  G_benchmark_graph = LFR_benchmark_graph(n = N, tau1 = tau1, tau2 = tau2, mu = mu, min_de
                        max_degree = k, min_community=minc, max_community = maxc, seed = 1
  communities = {frozenset(G_benchmark_graph.nodes[v]['community']) for v in G_benchmark_g


N = 128
tau1 = 3
tau2 = 1.5
mu = 0.04
k = 16
minc = 32
maxc = 32
points_x = list()
points_y = list()

for mu in range(1,101):
  mu = mu/100
  print( round(mu*100,2), "% Concluded")
  G_benchmark_graph = LFR_benchmark_graph(n = N, tau1 = tau1, tau2 = tau2, mu = mu, min_de
                          max_degree = k, min_community=minc, max_community = maxc, seed =
  communities = {frozenset(G_benchmark_graph.nodes[v]['community']) for v in G_benchmark_g
  p = ([list(x) for x in communities])

  partition_communities = {}
  for i in range(len(p)):
```

```
    for i in range(len(p)):
      for j in range(len(p[i])):
        partition_communities[p[i][j]] = i
    #drawing_graph_and_communities(G_benchmark_graph,partition_communities, "LFT Communities

    listaClusters = list()
    partition_greedy = greedy(G_benchmark_graph,False)
    listaClusters.append(partition_greedy)

    partition_label = label(G_benchmark_graph,False)
    listaClusters.append(partition_label)

    partition_givan_newman = givan_newman_method(G_benchmark_graph,4,False)
    listaClusters.append(partition_givan_newman)

    partition_louvain = louvain(G_benchmark_graph,False)
    listaClusters.append(partition_louvain)

    matrix_values = np.zeros((len(listaClusters[0]),len(listaClusters) ))
    for i in range(len(listaClusters)):
      for j in range(len(listaClusters[i])):
        matrix_values[j,i] = listaClusters[i][j]
    df = pd.DataFrame(matrix_values)

    matrix_values = np.zeros((len(partition_communities)))
    for i in range(len(partition_communities)):
      matrix_values[i] = partition_communities[i]

    matrix_final = np.zeros((4))
    for i in range(4):
      nmi = normalized_mutual_info_score(df[i],matrix_values)
      if nmi > 1 or nmi < 0 : nmi = 0
      matrix_final[i] = nmi
    points_x.append(mu)
    points_y.append(matrix_final)
    clear_output()
points_y = np.asmatrix(points_y)
points_y = pd.DataFrame(points_y, columns = ["FastGreedy", "Label_propagation", "Givan_New



fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Mu', fontsize = 15)
ax.set_ylabel('NMI', fontsize = 15)
ax.set_title('NMI for different Methods of Modularity', fontsize = 20)
targets = ["FastGreedy", "Label_propagation", "Givan_Newman", "Louvain"]
colors = ['r', 'g', 'b', 'y', 'k']

ax.plot(points_x ,points_y["FastGreedy"], color = 'r')
ax.plot(points_x ,points_y["Label_propagation"], color = 'b')
ax.plot(points_x ,points_y["Givan_Newman"], color = 'k')
ax.plot(points_x ,points_y["Louvain"], color = 'g')
ax.legend(targets)
ax.grid()
```



9 – Consider the following networks:

a) E-road network (http://konect.cc/networks/subelj_euroroad),

b) C. elegans neural network (http://www-personal.umich.edu/~mejn/netdata/celegansneural

c) US airport network (http://toreopsahl.com/datasets/#usairports)

d) Human protein network (http://konect.cc/networks/maayan-vidal)

Construct a table showing the following properties of these networks:

N,

Average degree,

assortativity coefficient,

average shortest path length,

modularity.

In the case of modularity, show the values for the following methods:

(i) Fastgreedy,

(ii) Label propagation,

(iii) Givan Newman,

(iv)Louvain.

(a) Compare the results in terms of the modularity.

(b) Are there some relation between assortativity and other network measures?

```
listaGraph = list()
listaIndex = ["E-road network", "C. elegans neural network", "US airport network ", "Humar

G_eroad = nx.read_edgelist("eroad.edges", nodetype=int, data=(('weight',int),))
listaGraph.append(trata_rede(G_eroad))

G_celneural = nx.read_gml("celegansneural.gml")
listaGraph.append(trata_rede(G_celneural, multigraph = True))

G_usaairport = nx.read_edgelist("usaairport.edges", nodetype=int, data=(('weight',int),))
listaGraph.append(trata_rede(G_usaairport))

G_maayan_vidal = nx.read_edgelist("maayan-vidal.edges", nodetype=int)
listaGraph.append(trata_rede(G_maayan_vidal))
```

```
matrix_values = np.zeros((4, 4))
for i in range(len(listaGraph)):
  G = listaGraph[i]
  matrix_values[i,0] = len(G)
  matrix_values[i,1] = mean(np.array(list(dict(G.degree()).values())))
  matrix_values[i,2] = nx.degree_assortativity_coefficient(G)
  matrix_values[i,3] = nx.average_shortest_path_length(G)
df = pd.DataFrame(matrix_values, columns = ["N", "Avg Degree", "AssortCoef", "Avg SPL"],
                  index = ["E-road network", "C. elegans neural network", "US airport netv
```

```
df
```

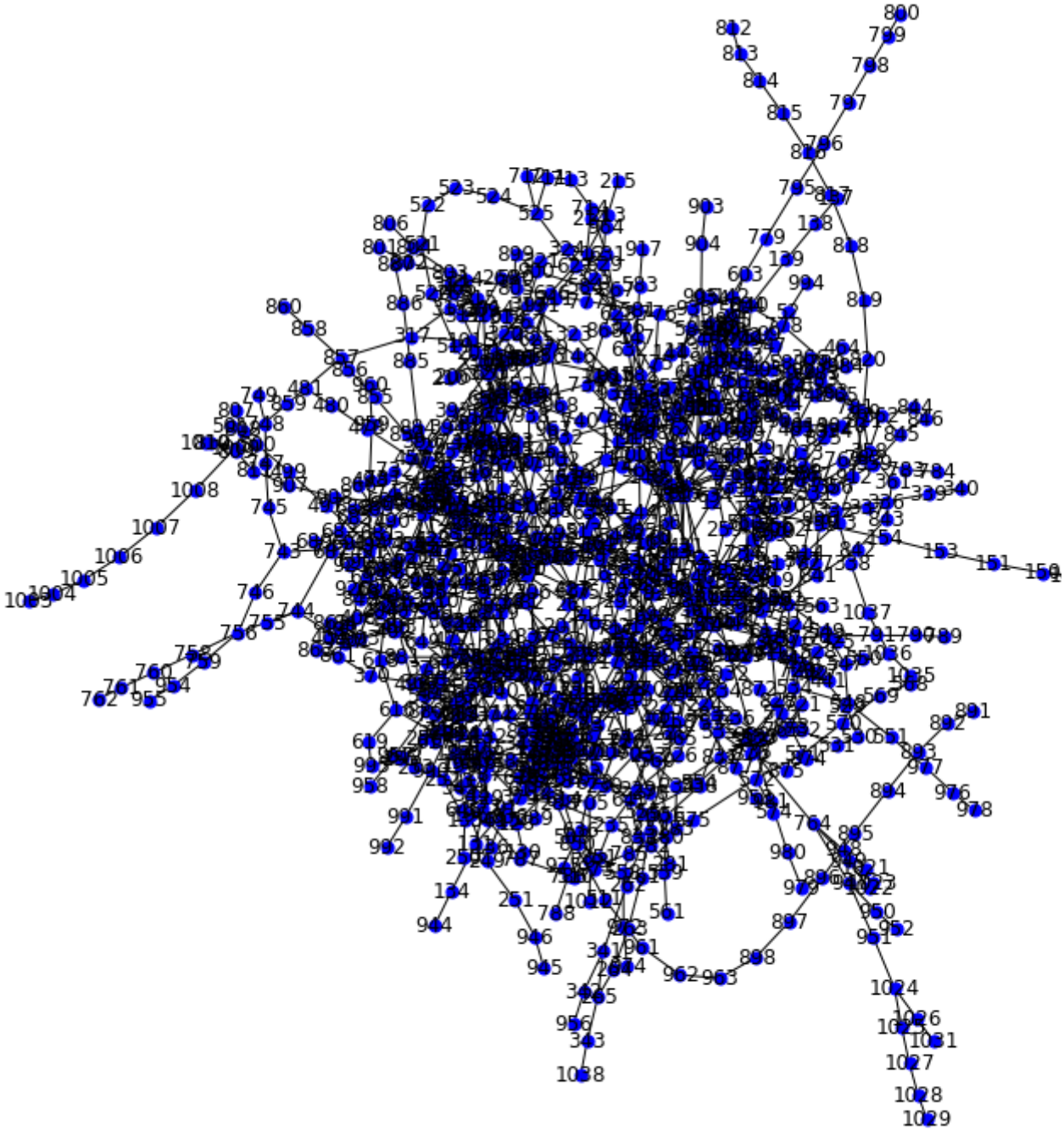
	N	Avg Degree	AssortCoef	Avg SPL
E-road network	1039.0	2.512031	0.090040	18.395146
C. elegans neural network	297.0	14.464646	-0.163199	2.455319
US airport network	500.0	11.920000	-0.267863	2.991030
Human protein network	2783.0	4.316924	-0.136560	4.839802

```
print("Network : ", listaIndex[0],"\n")
G = listaGraph[0]
```

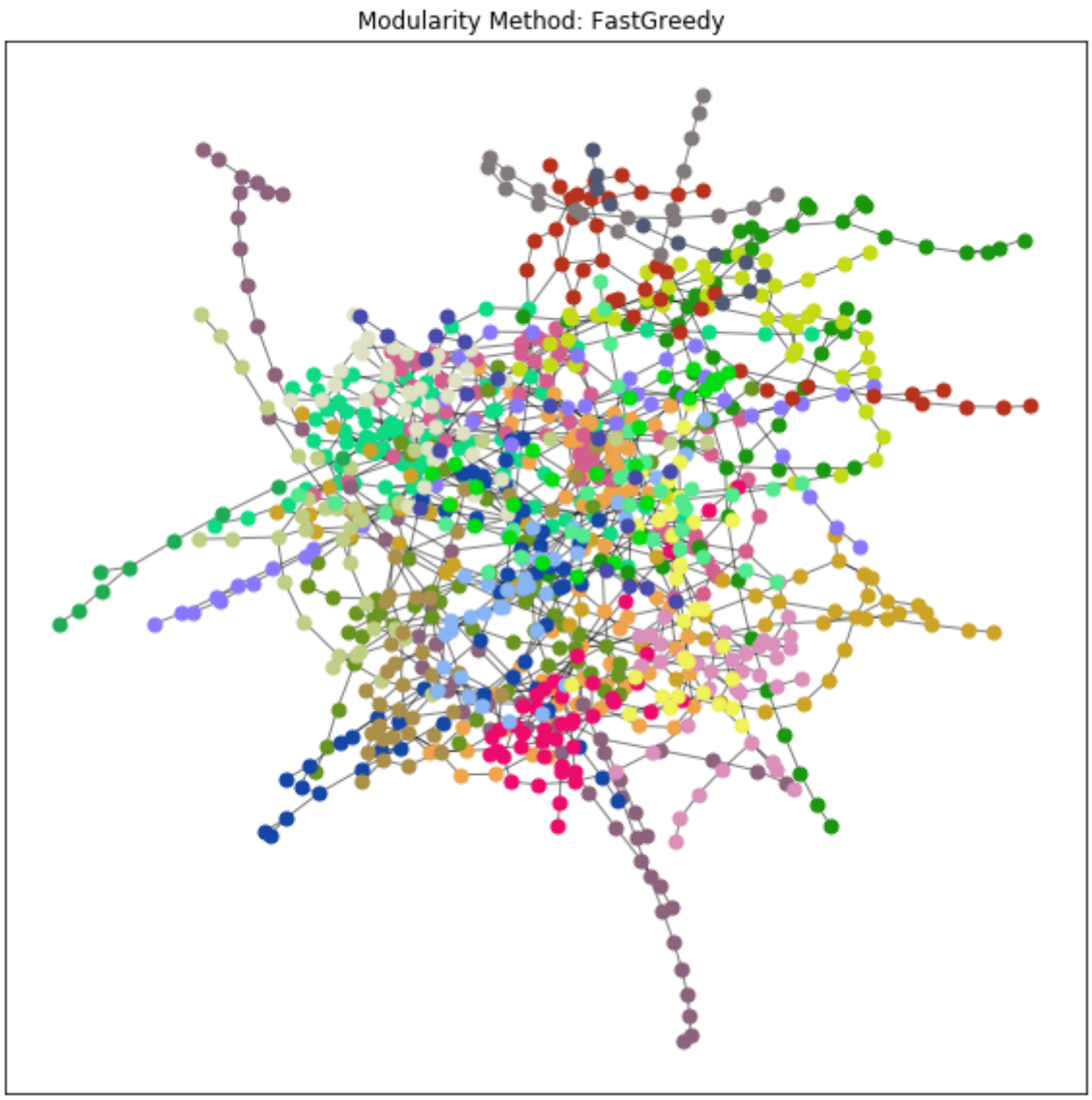
```
df.loc["E-road network",:]
```

```
Network :  E-road network

N             1039.000000
Avg Degree       2.512031
AssortCoef       0.090040
Avg SPL         18.395146
Name: E-road network, dtype: float64
```

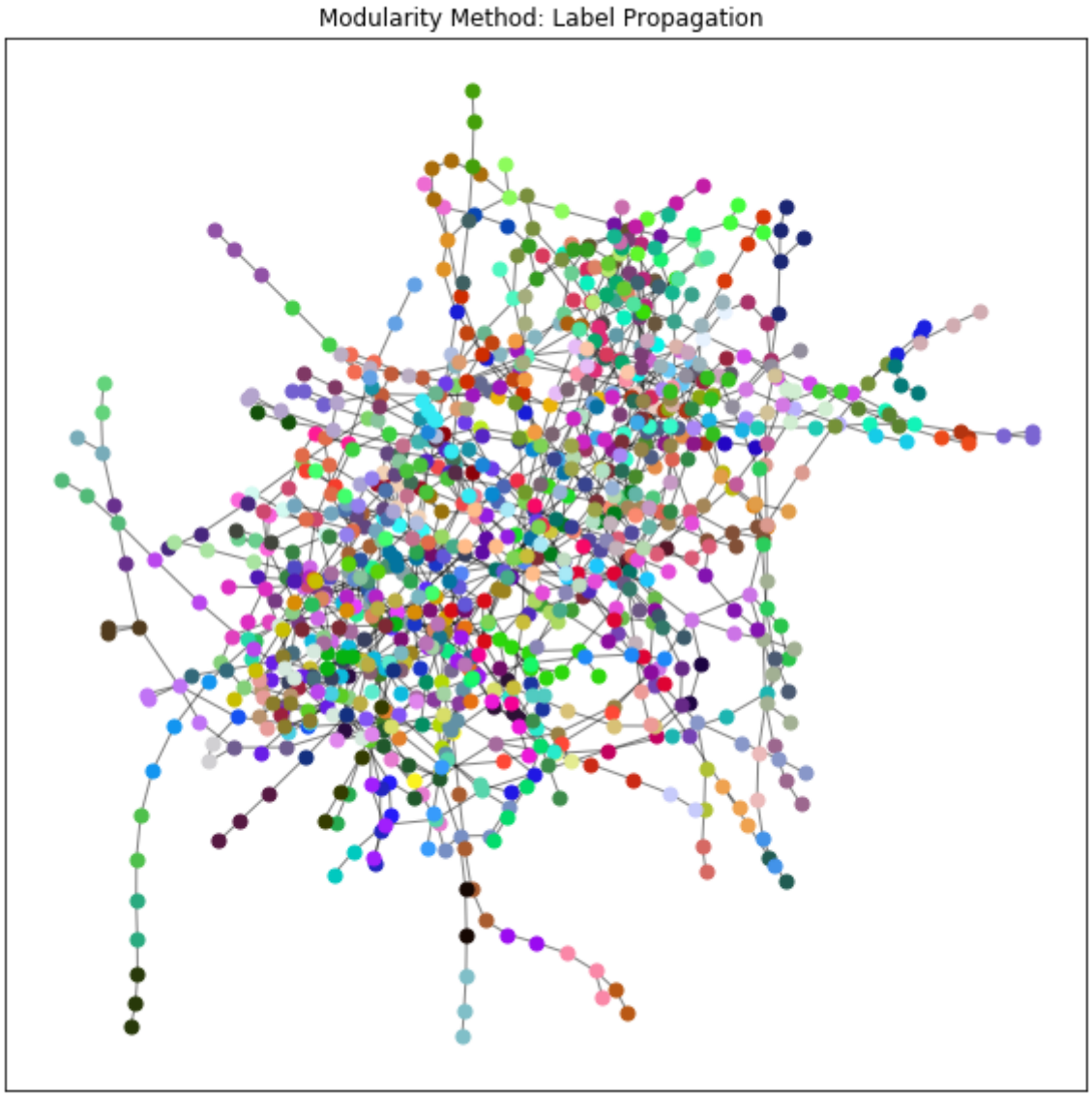
```
plota_rede(G, nd_size=50, fig_size=10, n_color = random.randint(0,8))
```



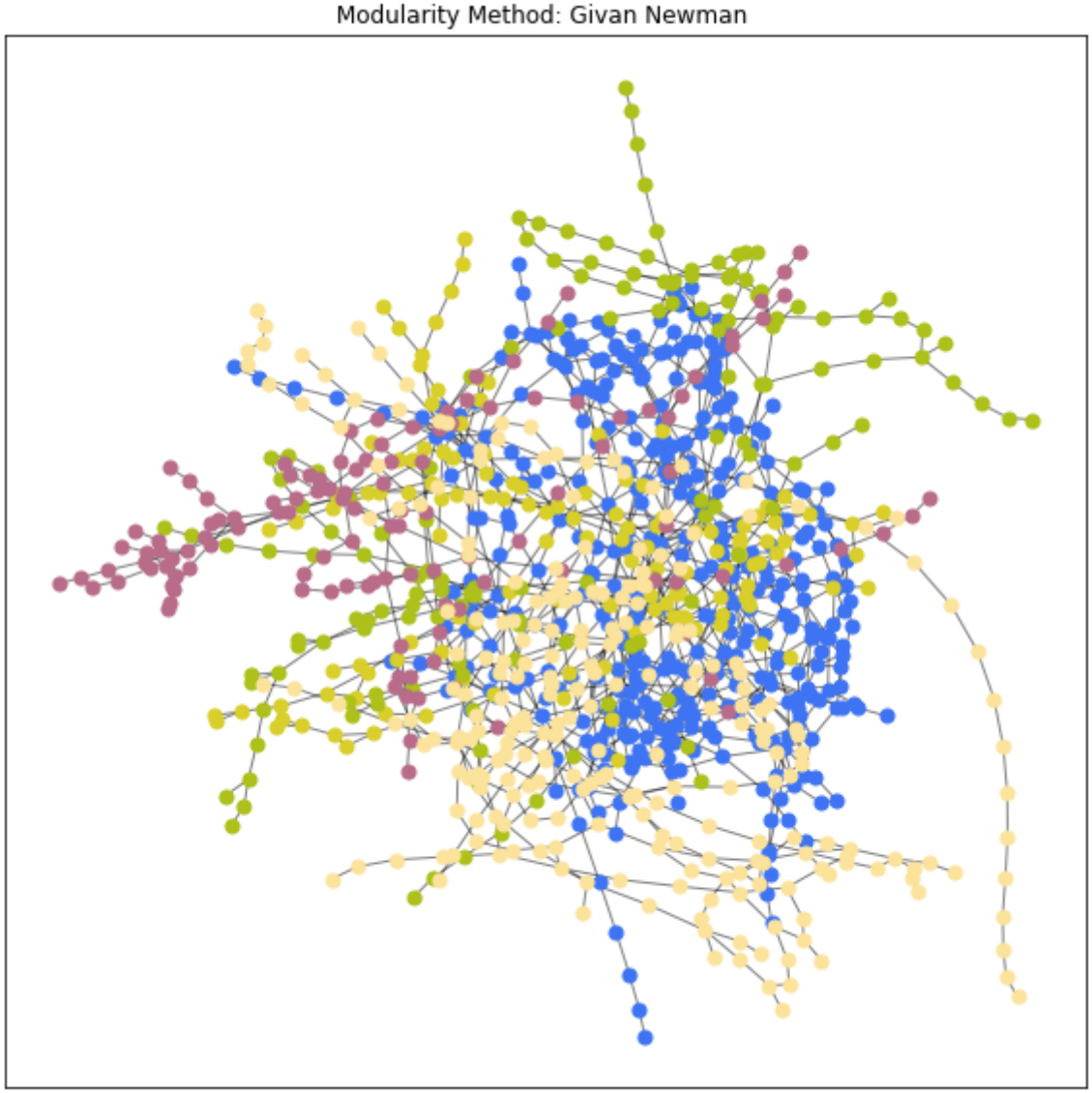
```
listaClusters = list()
partition_greedy = greedy(G)
listaClusters.append(partition_greedy)
```



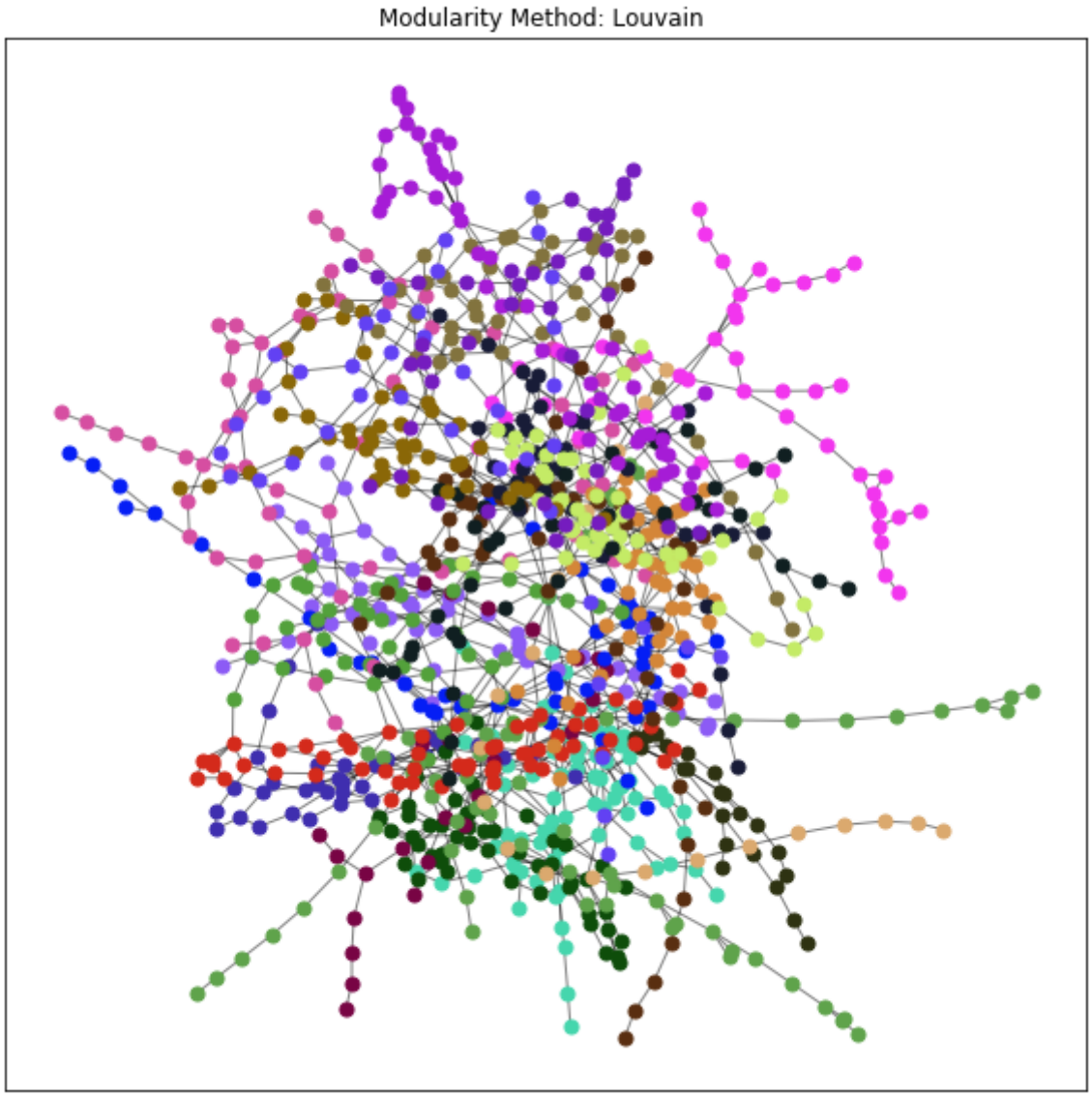

```
partition_label = label(G)
listaClusters.append(partition_label)
```



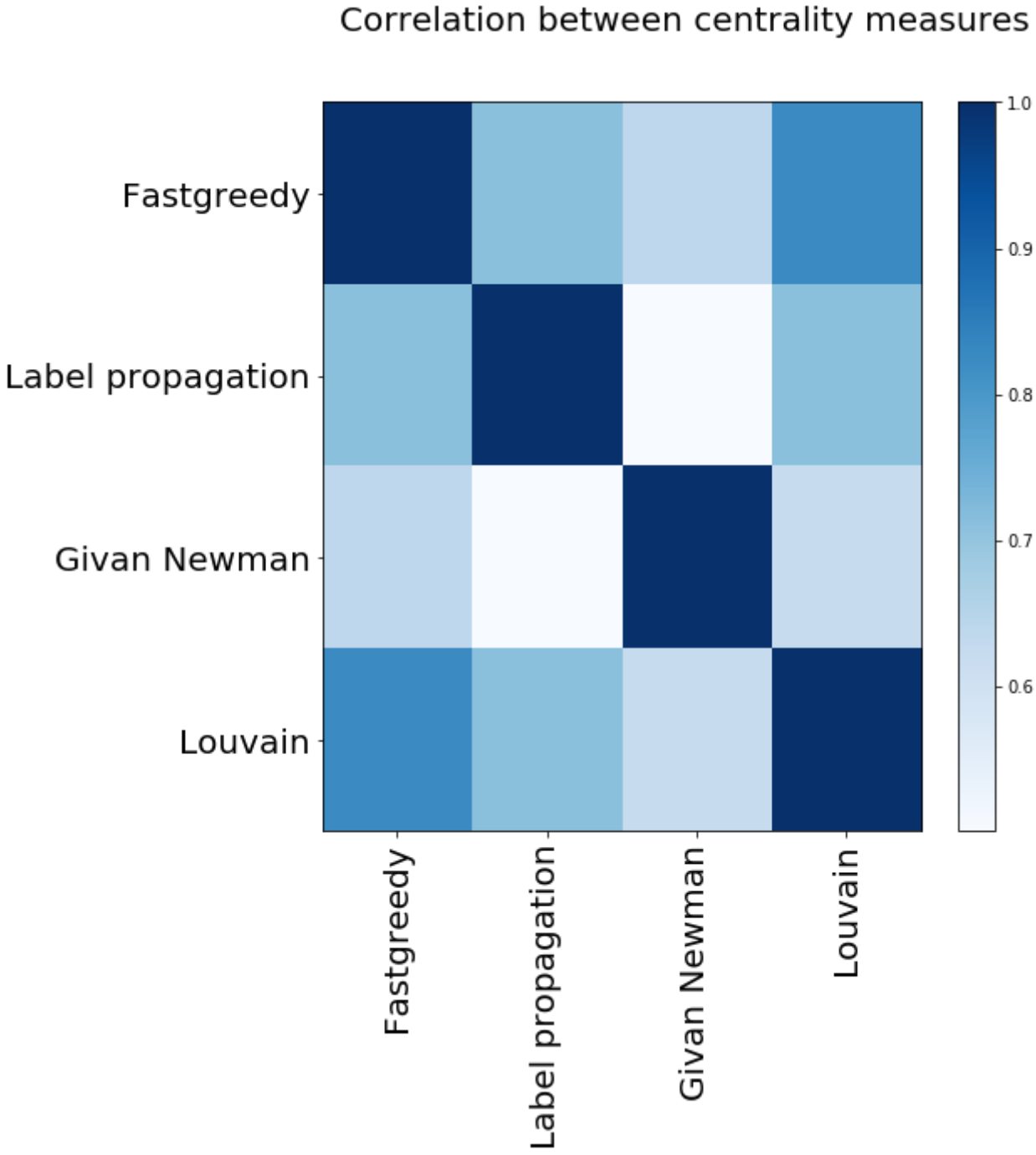

```
partition_givan_newman = givan_newman_method(G,5)
listaClusters.append(partition_givan_newman)
```




```
partition_louvain = louvain(G)
listaClusters.append(partition_louvain)
```



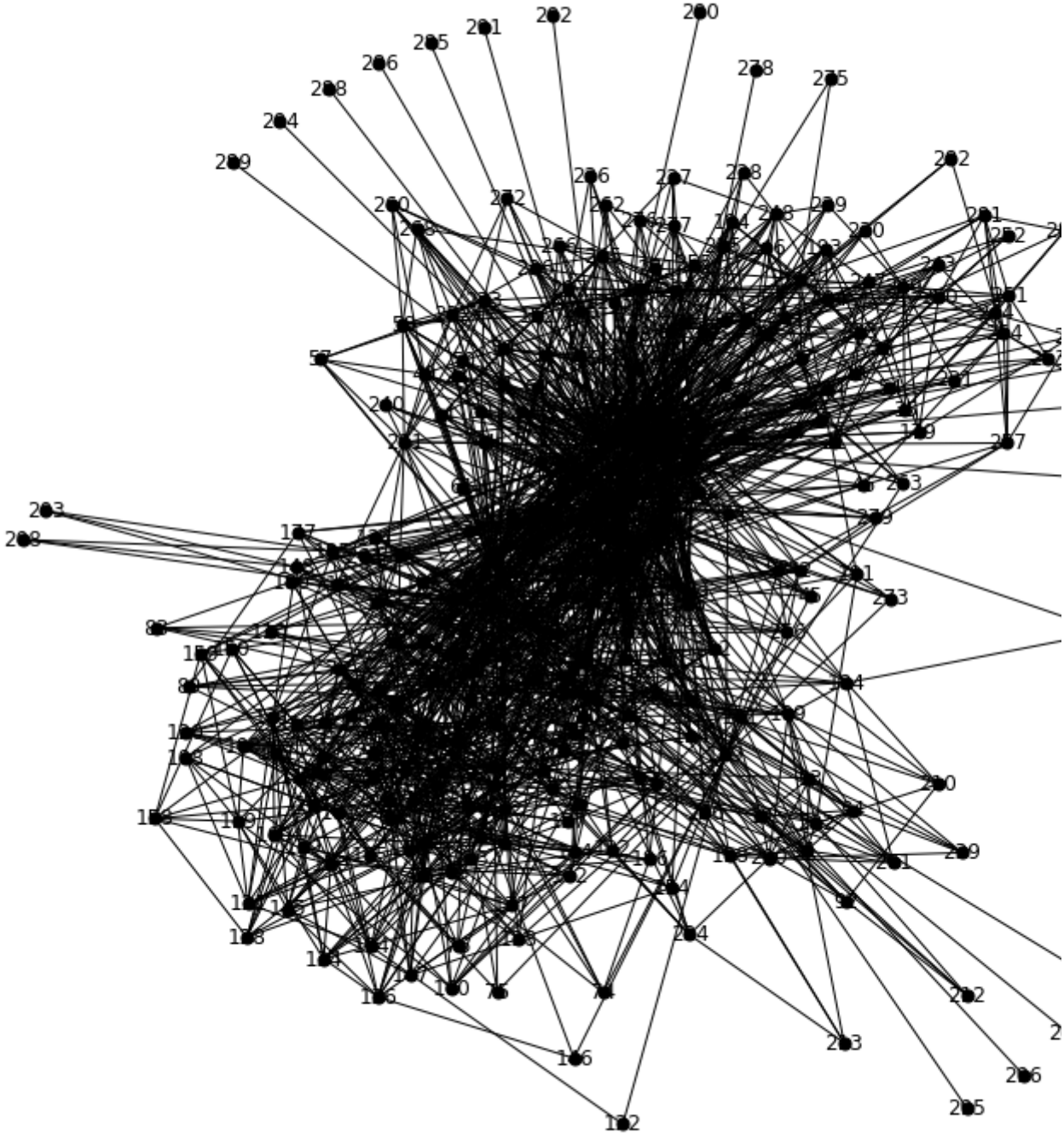

```
corrplot_NMI_between_modularities(listaClusters
          ,["Fastgreedy", "Label propagation", "Givan Newman", "Louvain"])
```



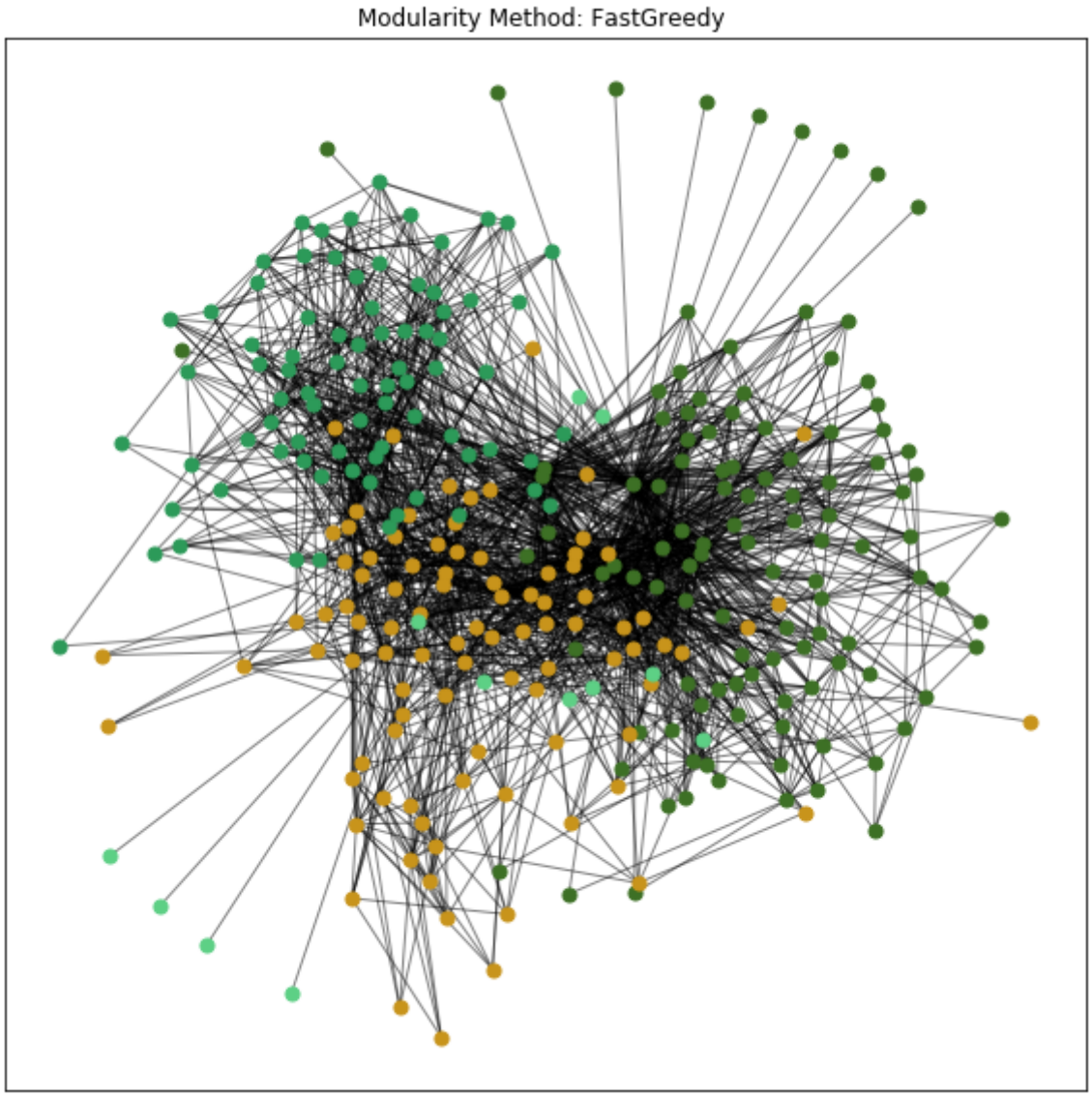

```
print("Network : ", listaIndex[1],"\n")
G = listaGraph[1]
df.loc["C. elegans neural network",:]
```

```
Network :  C. elegans neural network

N             297.000000
Avg Degree     14.464646
AssortCoef     -0.163199
Avg SPL         2.455319
Name: C. elegans neural network, dtype: float64
```

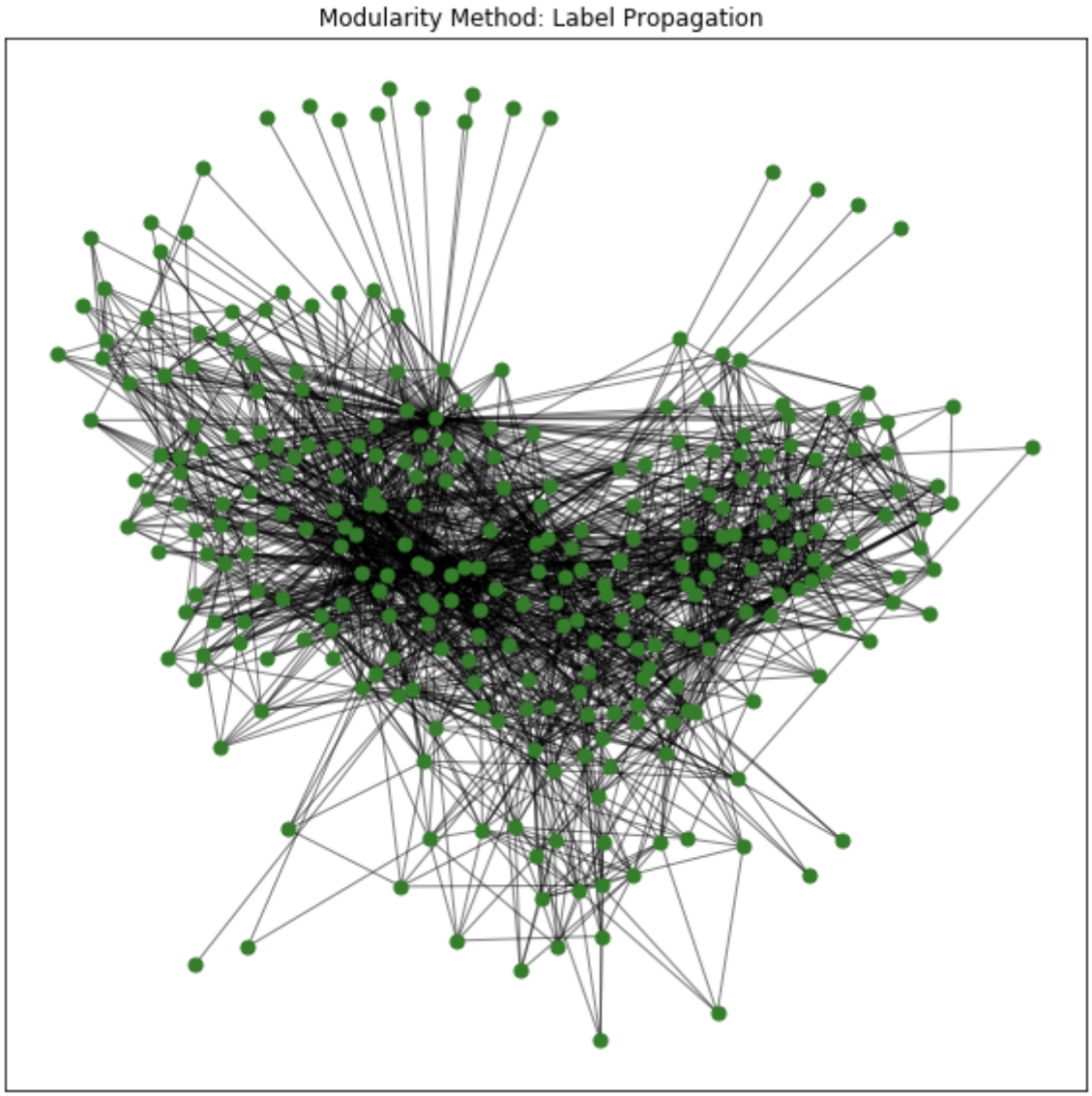
```
plota_rede(G, nd_size=50, fig_size=10, n_color = random.randint(0,8))
```



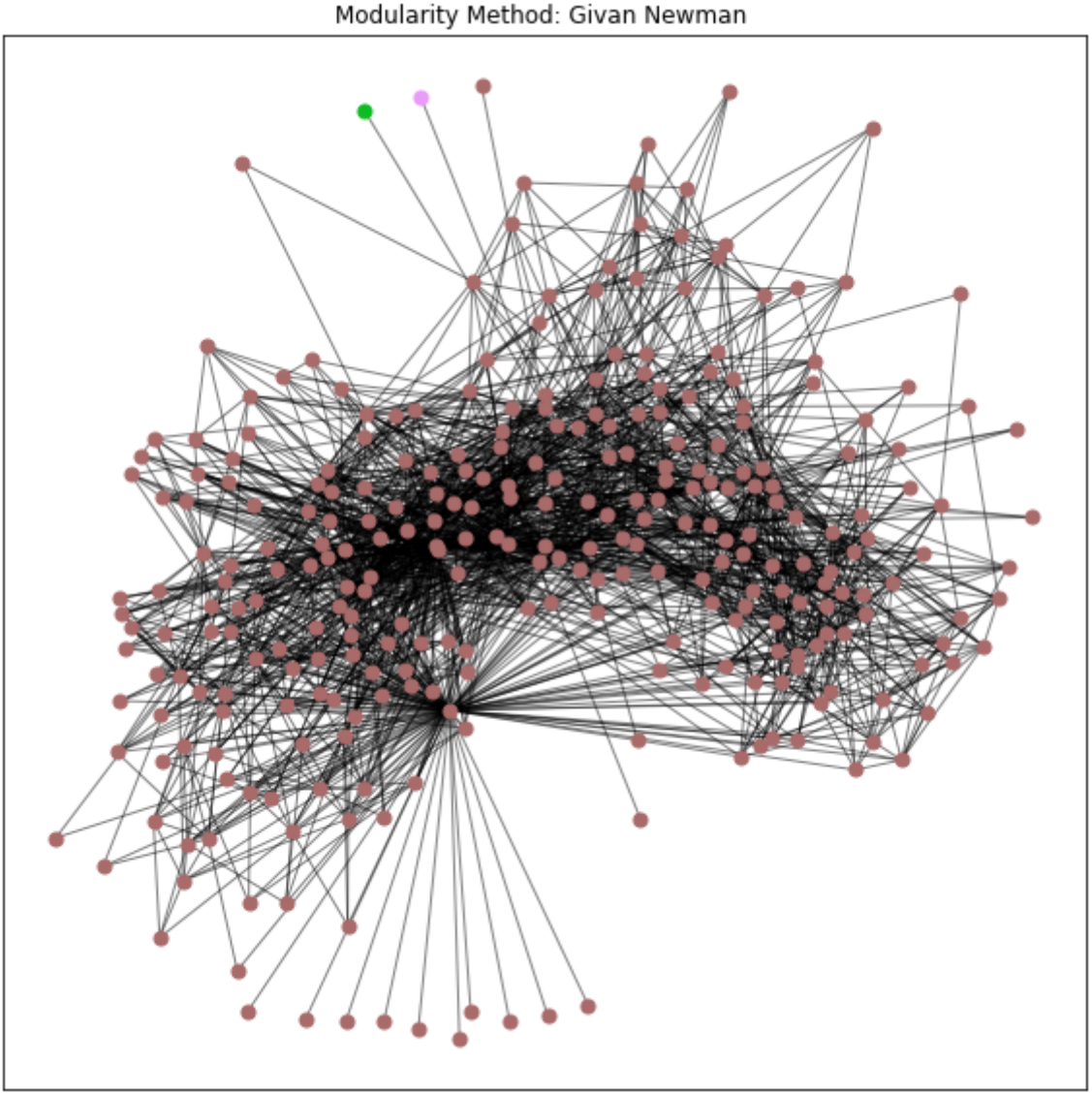
```
listaClusters = list()
partition_greedy = greedy(G)
listaClusters.append(partition_greedy)
```



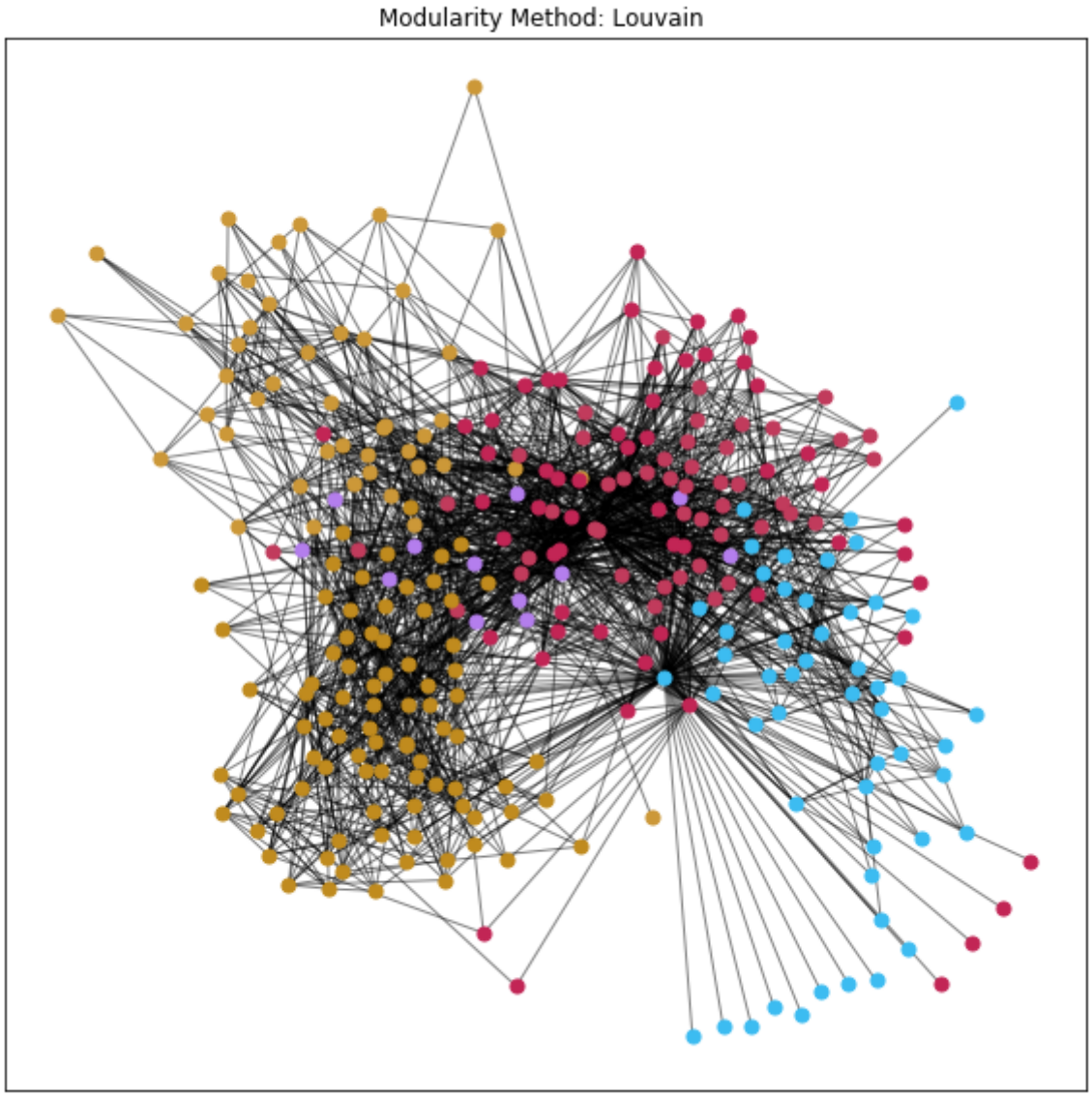

```
partition_label = label(G)
listaClusters.append(partition_label)
```



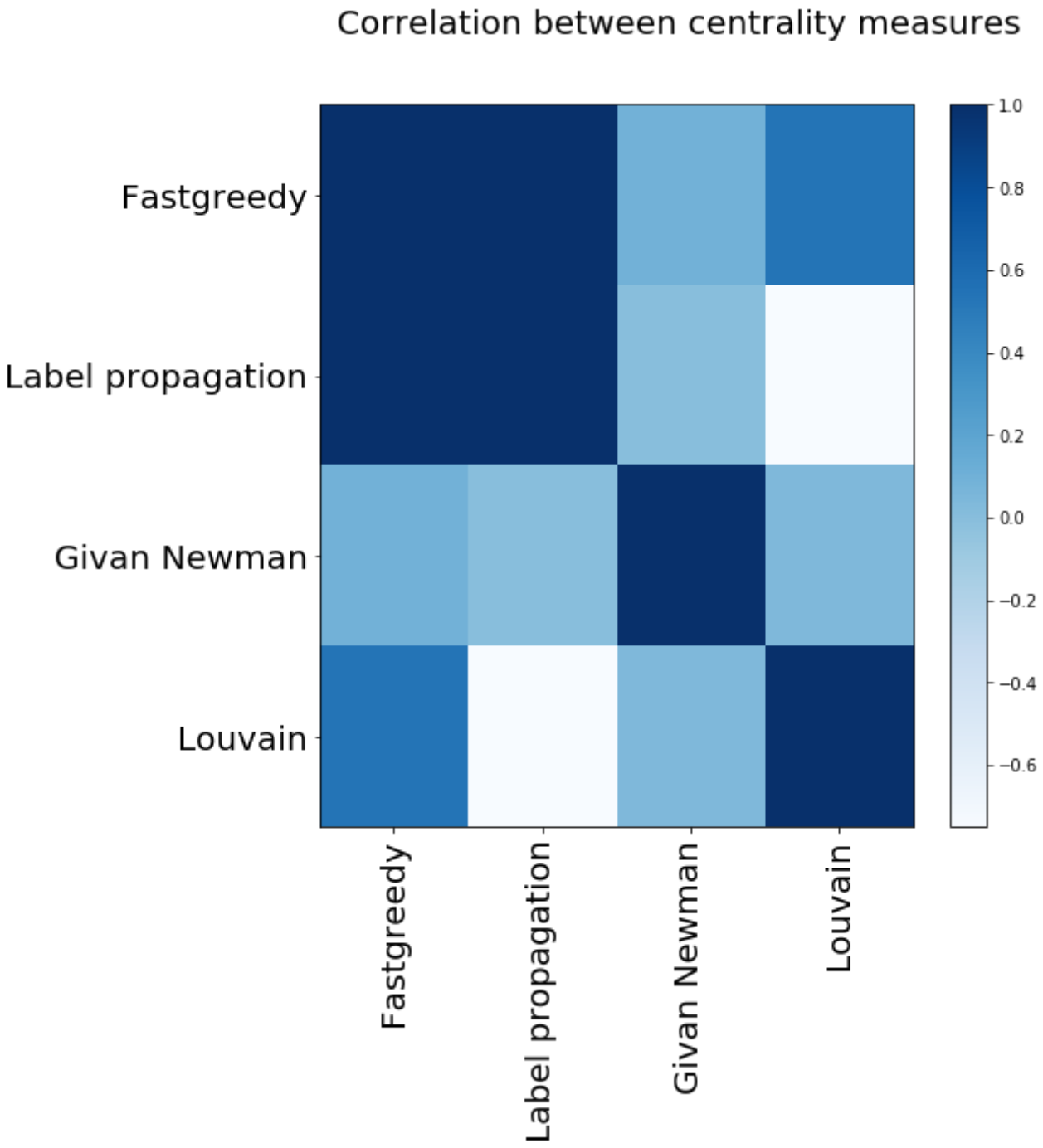

```
partition_givan_newman = givan_newman_method(G,3)
listaClusters.append(partition_givan_newman)
```




```
partition_louvain = louvain(G)
listaClusters.append(partition_louvain)
```



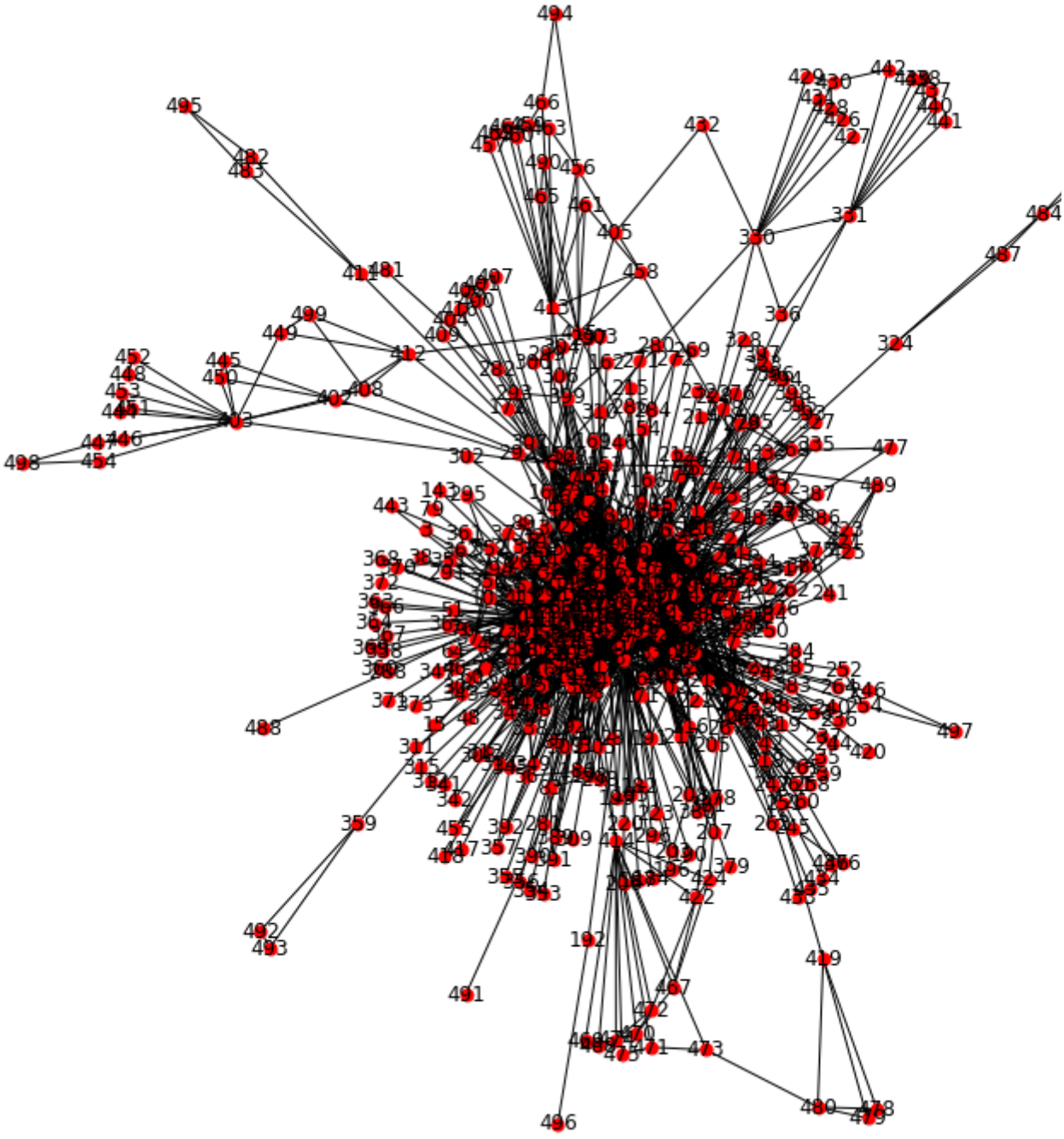

```
corrplot_NMI_between_modularities(listaClusters
          ,["Fastgreedy", "Label propagation", "Givan Newman", "Louvain"])
```



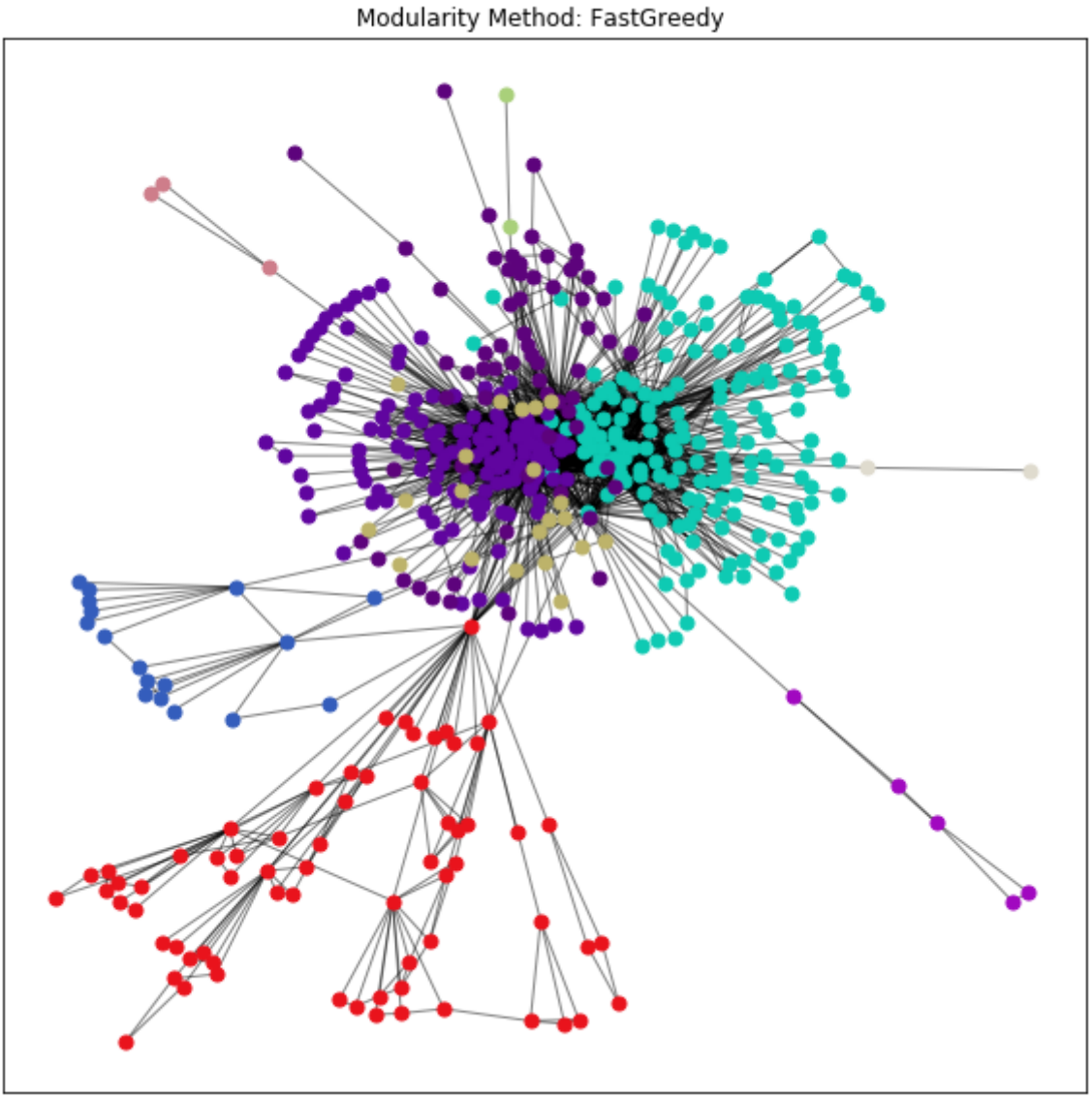

```
print("Network : ", listaIndex[2],"\n")
G = listaGraph[2]
df.loc["US airport network ",:]
```

```
Network :  US airport network

N             500.000000
Avg Degree     11.920000
AssortCoef     -0.267863
Avg SPL         2.991030
Name: US airport network , dtype: float64
```

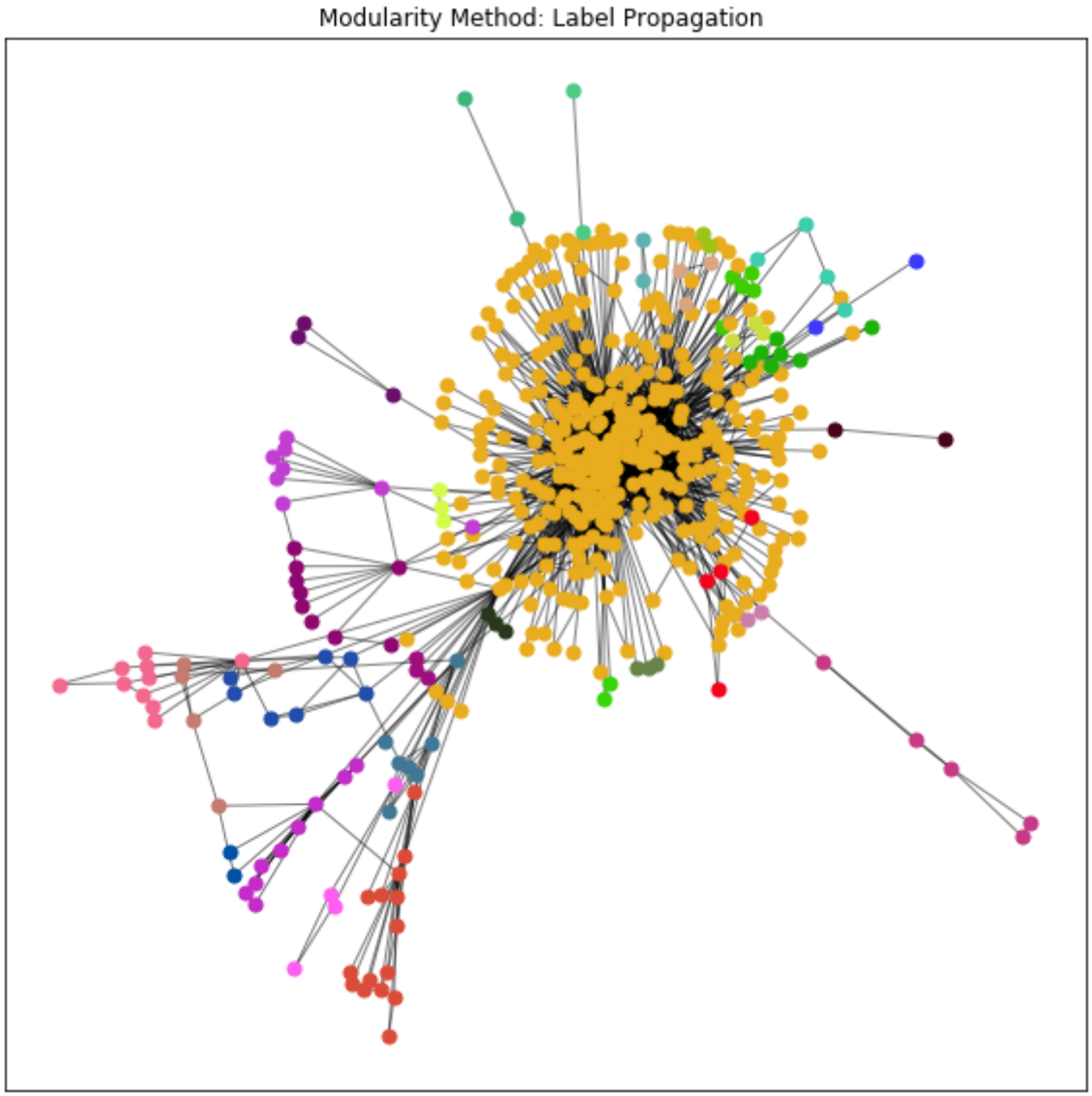
```
plota_rede(G, nd_size=50, fig_size=10, n_color = random.randint(0,8))
```



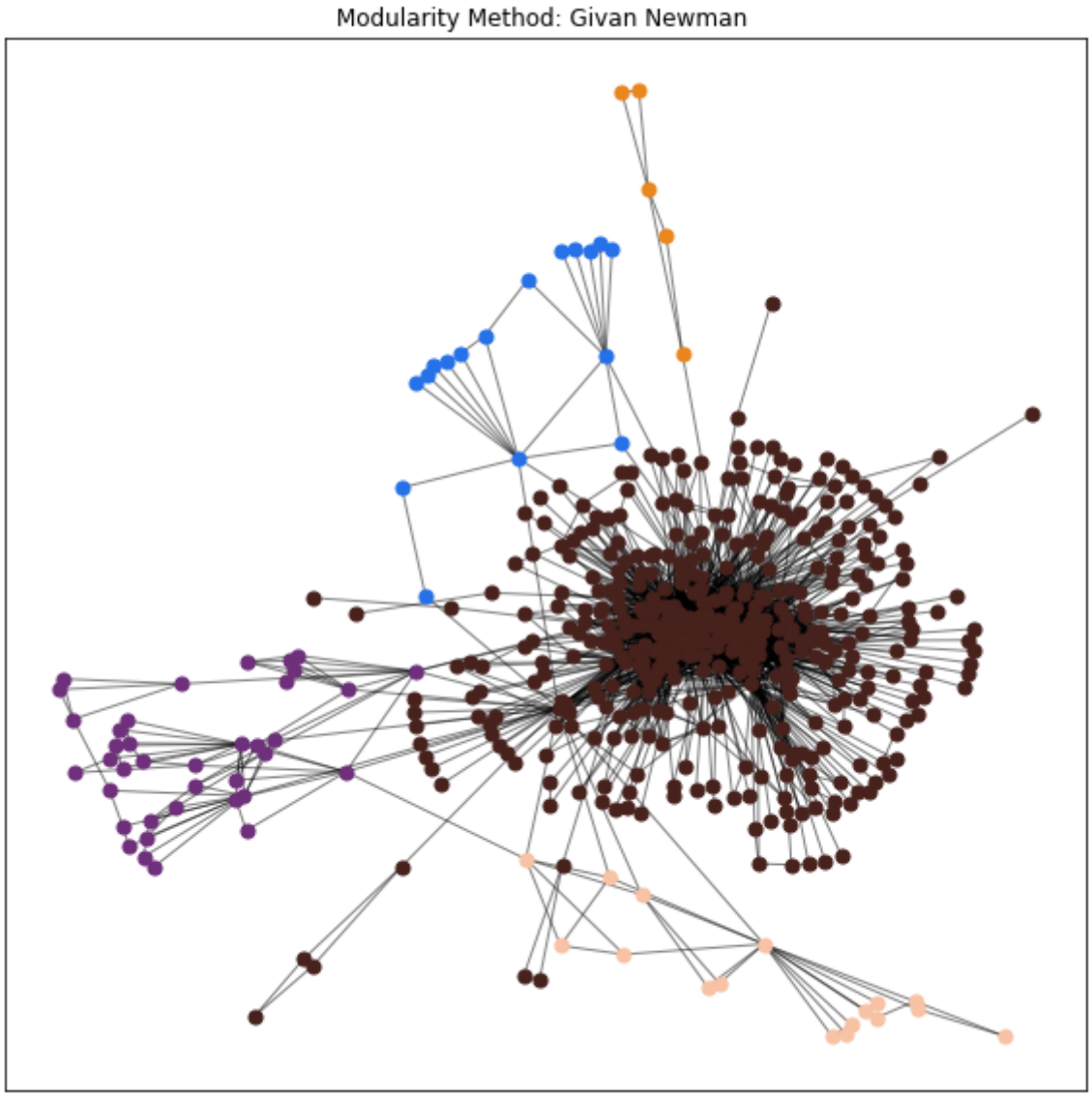
```
listaClusters = list()
partition_greedy = greedy(G)
listaClusters.append(partition_greedy)
```



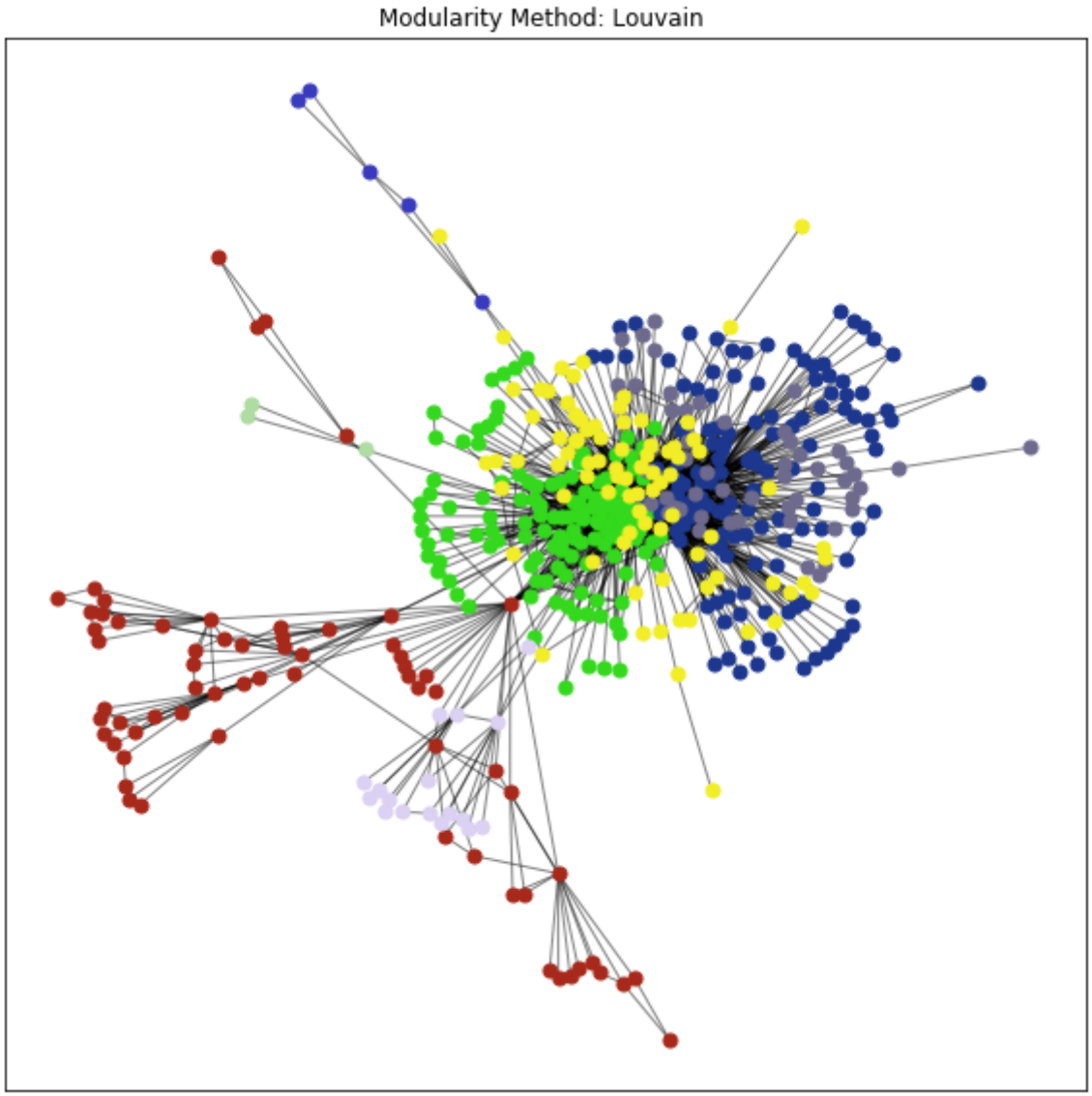

```
partition_label = label(G)
listaClusters.append(partition_label)
```



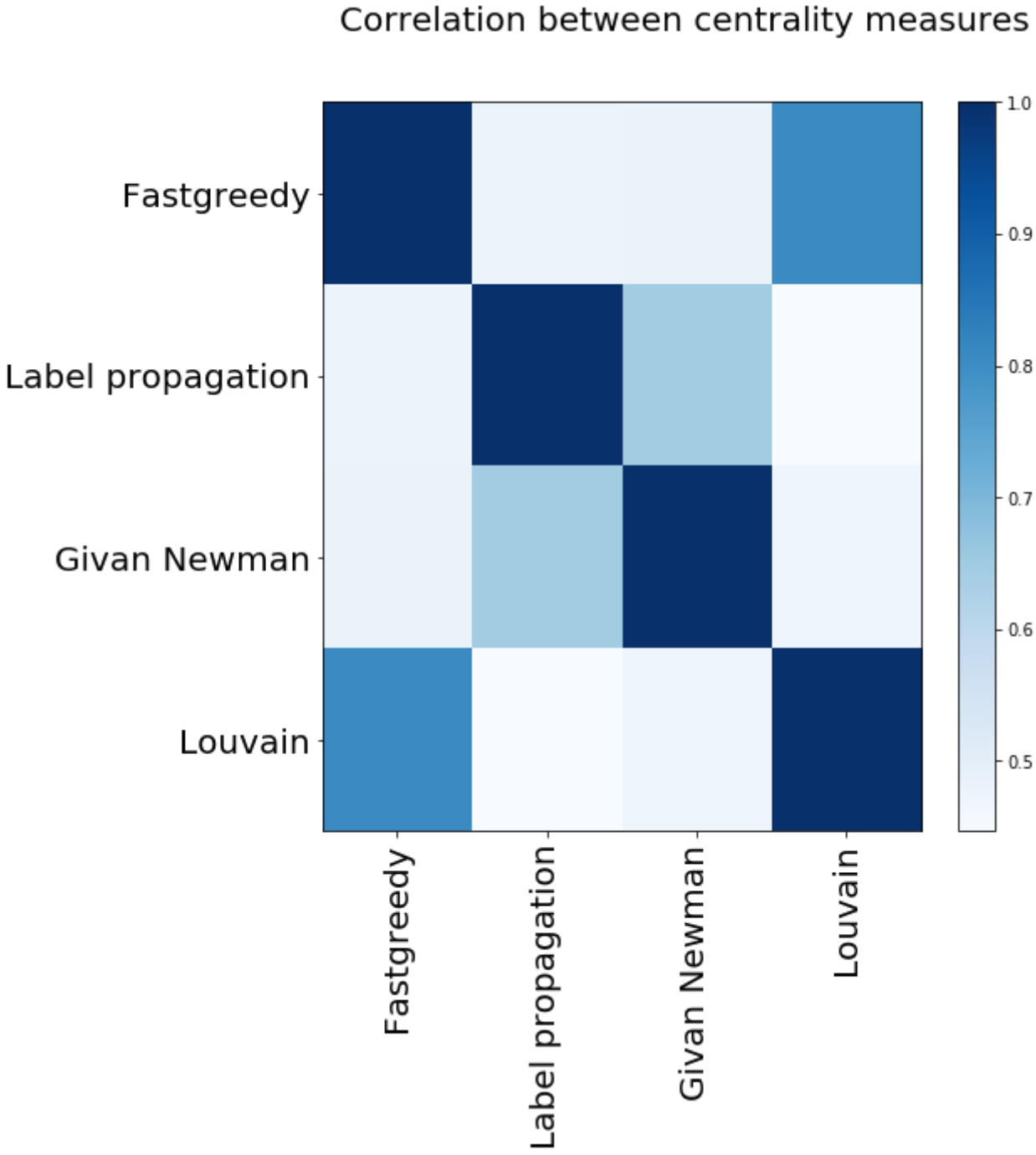

```
partition_givan_newman = givan_newman_method(G,5)
listaClusters.append(partition_givan_newman)
```




```
partition_louvain = louvain(G)
listaClusters.append(partition_louvain)
```



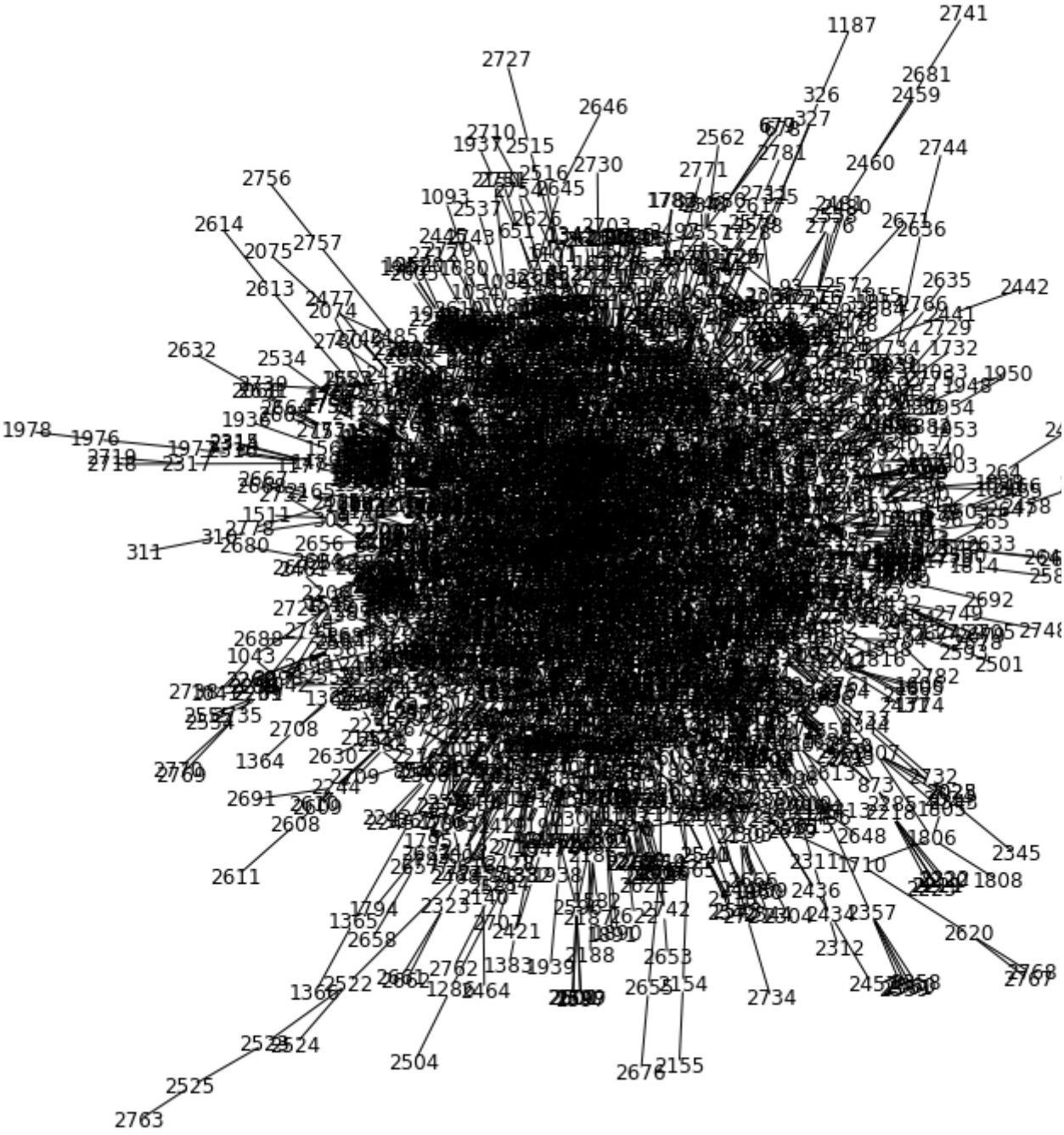

```
corrplot_NMI_between_modularities(listaClusters
          ,["Fastgreedy", "Label propagation", "Givan Newman", "Louvain"])
```



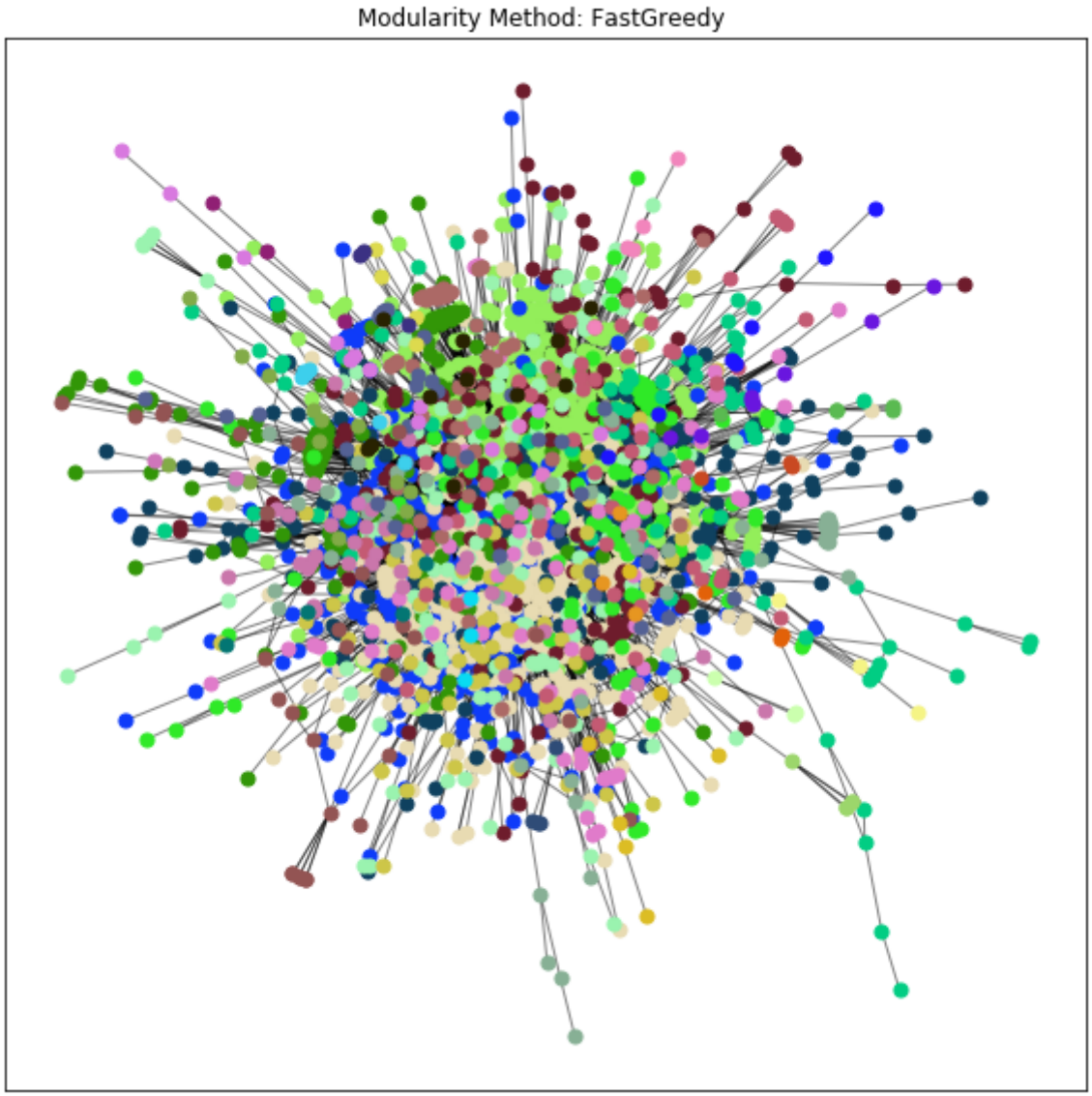

```
print("Network : ", listaIndex[3],"\n")
G = listaGraph[3]
df.loc["Human protein network",:]
```

```
Network :  Human protein network

N             2783.000000
Avg Degree       4.316924
AssortCoef      -0.136560
Avg SPL          4.839802
Name: Human protein network, dtype: float64
```

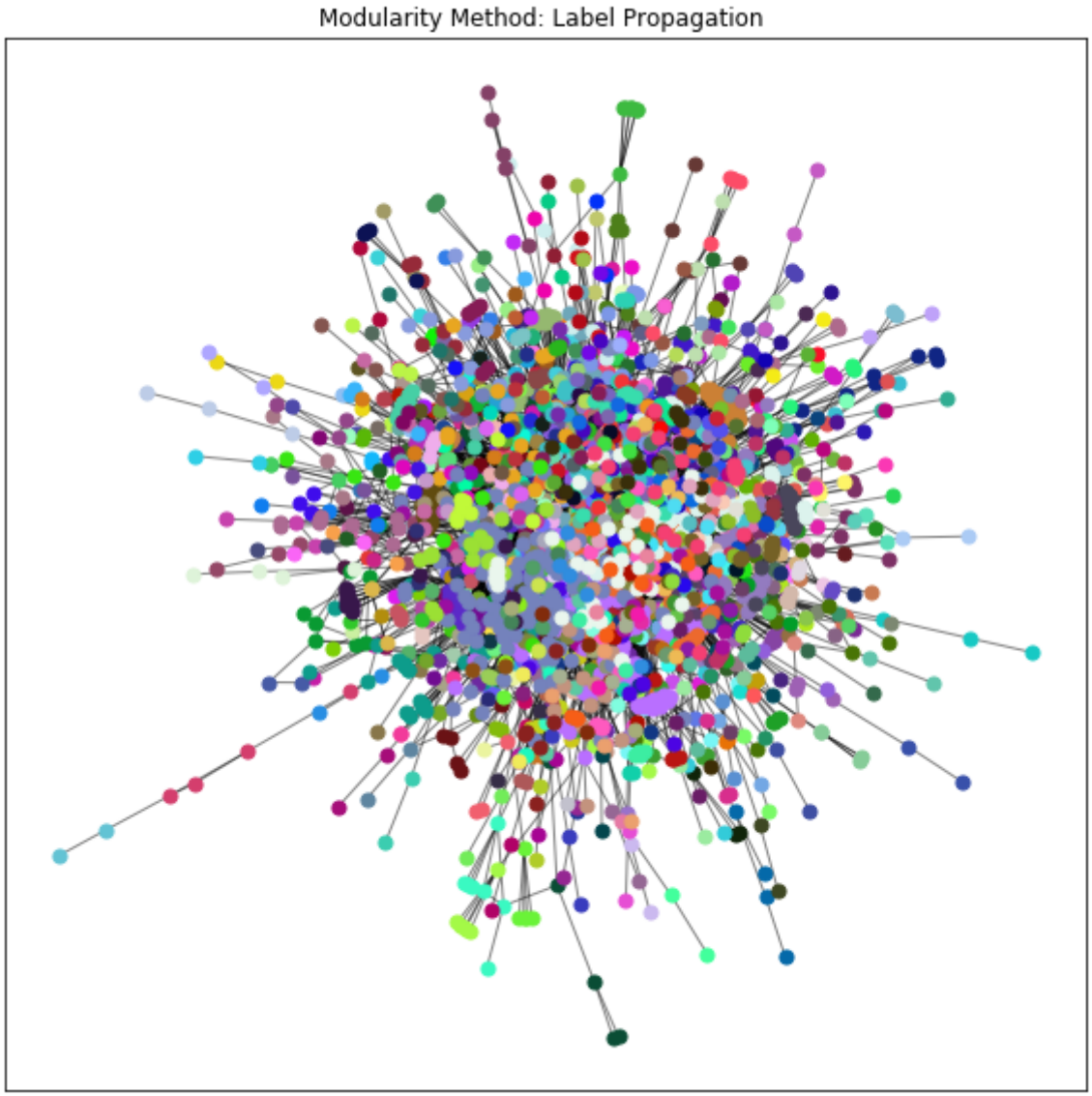
```
plota_rede(G, nd_size=50, fig_size=10, n_color = random.randint(0,7))
```



```
listaClusters = list()
partition_greedy = greedy(G)
listaClusters.append(partition_greedy)
```




```
partition_label = label(G)
listaClusters.append(partition_label)
```




```
partition_givan_newman = givan_newman_method(G,10)
listaClusters.append(partition_givan_newman)


partition_louvain = louvain(G)
listaClusters.append(partition_louvain)


corrplot_NMI_between_modularities(listaClusters
          ,["Fastgreedy", "Label propagation", "Givan Newman", "Louvain"])


correlation_matrix(df,2)
```