1. Name: Jayaa Emekar
2. CSCI-651-Assignment-3

Imports Required for all functions

```
import networkx as nx
from scipy import stats
import matplotlib.pyplot as plt
import numpy as np
import math
import random
import operator
```

.

Question1

1. For the Openflights airport network (2016), the high-energy physics citation network, and the H-I-05 (human protein-protein interaction) network, determine three separate values of γ for each network using a simple log-log plot, logarithmic binning, and the complementary cumulative distribution. For each estimate, include a figure with an appropriate fitted curve and the associated r 2 value. There are a number of tools for curve fitting in Python. scipy.optimize.curve_fit or scipy.stats.linregress might be useful here.

Simple log-log ploting for networks

Function to find degree distribution of graph

```
#Function to find degree distribution of graph
def degreeDistrGraph(G):
  degrees = {}
  degreeList = [G.degree(v) for v in G.nodes()]
  for deg in degreeList:
    degrees[deg] = degrees.get(deg, 0) + 1
  (X, Y) = zip(*[(key, degrees[key]/len(G)) for key in degrees])
  return X,Y

```

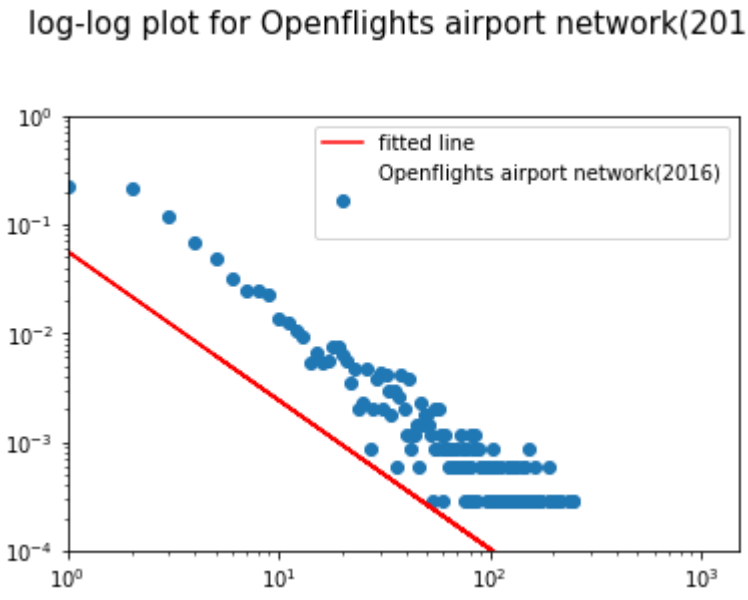
Function to plot simple log-log plot for degree distribution of graph

```
#Function to plot simple log-log plot for degree distribution of graph
def plotLogLogPlot(G, data_set_name):
  X, Y = degreeDistrGraph(G)
  x = np.log(np.asarray(X).astype(np.float))
  y = np.log(np.asarray(Y).astype(np.float))
  res = stats.linregress(x, y)
  print(f"R-squared: {res.rvalue**2:.6f}")
  print ('Slope of line', res.slope)
  print()
  plt.scatter(X, Y, label=data_set_name)
  plt.plot(np.power(10,x), np.power(10,(res.intercept + res.slope*x)), 'r', label='f
  plt.yscale('log')
  plt.xscale('log')
  plt.title('log-log plot for '+data_set_name, fontsize ='15')
  plt.xlim(1, 1500)
  plt.ylim(1/10000, 1)
  plt.legend()
  plt.show()
```

Generate simple log-log plot for openflight network

```
# Generate simple log-log plot for openflight network
G = nx.read_edgelist("/content/out.openflights")
plotLogLogPlot(G,"Openflights airport network(2016)\n\n")
```

```
R-squared: 0.900237
Slope of line -1.3663124225854362
```



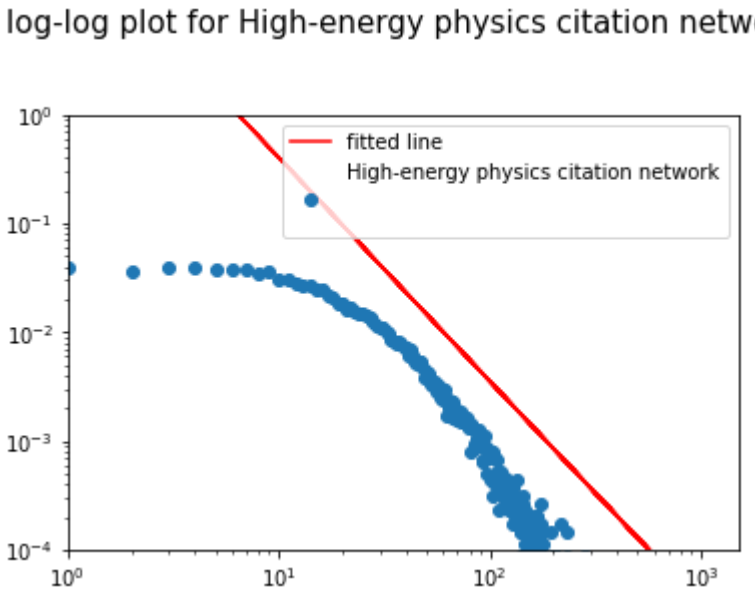
Values for Open flight network for Log-Log plot are as below:

1. R-squared: 0.900237
2. Gamma: -1.3663124225854362

Generate simple log-log plot for High-energy physics citation network

```
# Generate simple log-log plot for High-energy physics citation network
G = nx.read_edgelist("/content/Cit-HepPh.txt")
plotLogLogPlot(G,"High-energy physics citation network\n\n")
```

```
R-squared: 0.883175
Slope of line -2.053670459410214
```



Values for High-energy physics citation network for Log-Log plot are as below:

1. R-squared: 0.883175
2. Gamma: -2.053670459410214

.

.

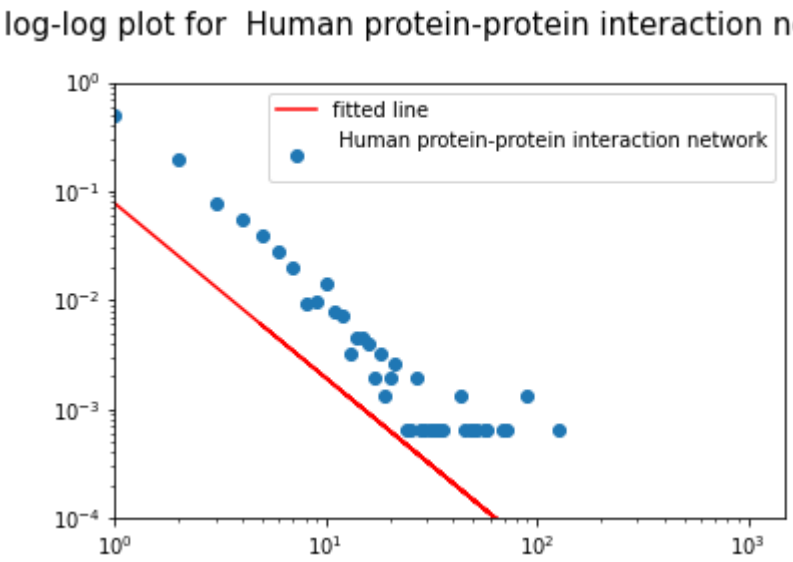
.

.

Generate simple log-log plot for H-I-05 (human protein-protein interaction) network

```
# Generate simple log-log plot for H-I-05 (human protein-protein interaction) networ
G = nx.read_edgelist("/content/H-I-05.tsv")
plotLogLogPlot(G," Human protein-protein interaction network\n")
```

```
R-squared: 0.883469
Slope of line -1.6027654485191132
```



Values for High-energy physics citation network for Log-Log plot are as below:

1. R-squared: 0.883469
2. Gamma: -1.6027654485191132

.

.

.

.

.

.

Logarithmic binning for networks

This function is to generate log-binning for network

```
#This function is to generate log-binning for network
def log_binning(x, y, bin_count=24):
    max_x = np.log10(max(x))
    max_y = np.log10(max(y))
    max_base = max([max_x,max_y])
    xx = [i for i in x if i>0]
    min_x = np.log10(np.min(xx))
    bins = np.logspace(min_x,max_base,num=bin_count)
    hist = np.histogram(x,bins)[0]
    nonzero_mask = np.logical_not(hist==0)
    hist[hist==0] = 1
    bin_means_y = (np.histogram(x,bins,weights=y)[0] / hist)
    bin_means_x = (np.histogram(x,bins,weights=x)[0] / hist)
    return bin_means_x[nonzero_mask],bin_means_y[nonzero_mask]
```

This function is to plot log-binning for network

```
#This function is to plot log-binning for network
def plotLogBinPlot(G, data_set_name):
  X, Y = degreeDistrGraph(G)
  lk, lebk = log_binning(np.array(X,dtype=np.float64), np.array(Y), bin_count=60)
  x = np.log10(np.asarray(lk).astype(np.float))
  y = np.log10(np.asarray(lebk).astype(np.float))
  res = stats.linregress(x, y)
  print(f"R-squared: {res.rvalue**2:.6f}")
  print ('Slope of line', res.slope)
  print()
  plt.scatter(lk,lebk,label= data_set_name)
  plt.plot(np.power(10,x), np.power(10,(res.intercept + res.slope*x)), 'r', label='f
  plt.title('log-binning plot for '+data_set_name, fontsize ='15')
  plt.yscale('log')
  plt.xscale('log')
  plt.xlim(1, 150)
  plt.ylim(1/1000, 1)
  plt.legend()
  plt.show()
```

.

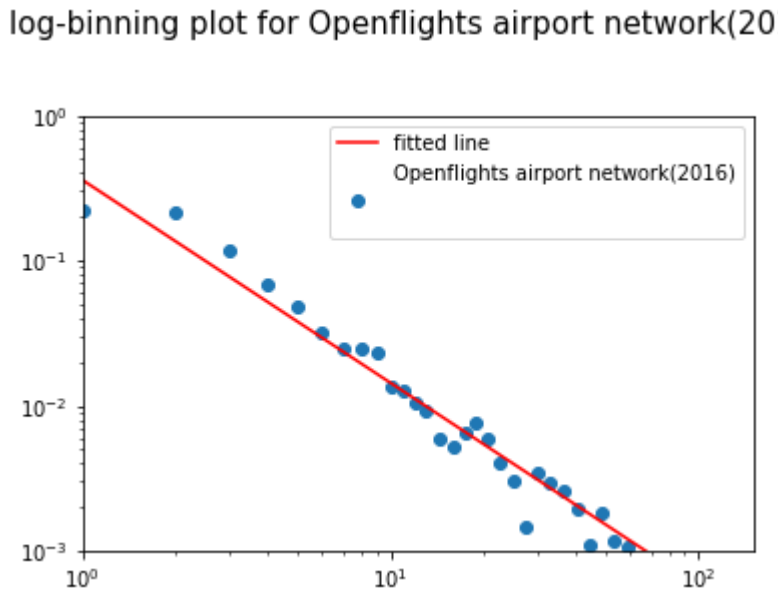
.

.

Generate simple log-binning for openflight network

```
# Generate simple log-binning for openflight network
G = nx.read_edgelist("/content/out.openflights")
plotLogBinPlot(G,"Openflights airport network(2016)\n\n")
```

```
R-squared: 0.975902
Slope of line -1.3972900128657209
```



Values for Open flight network for log-binning plot are as below:

1. R-squared: 0.975902
2. Gamma: -1.3972900128657209

.

.

.

.

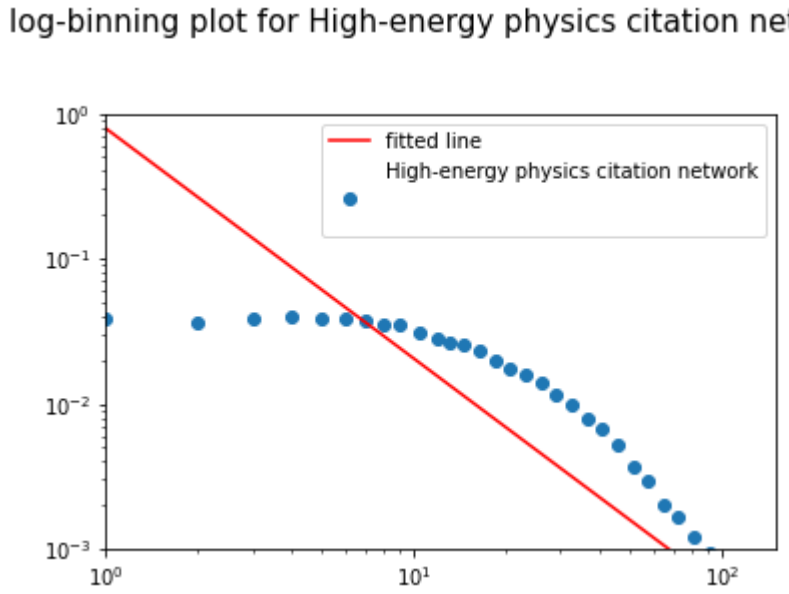
.

.

Generate simple log-binning plot for High-energy physics citation network

```
# Generate simple log-binning plot for High-energy physics citation network
G = nx.read_edgelist("/content/Cit-HepPh.txt")
plotLogBinPlot(G,"High-energy physics citation network\n\n")
```

```
R-squared: 0.890094
Slope of line -1.5851066497246415
```



Values for High-energy physics citation network for log-binning plot are as below:

1. R-squared: 0.890094
2. Gamma: -1.5851066497246415

.

.

.

.

.

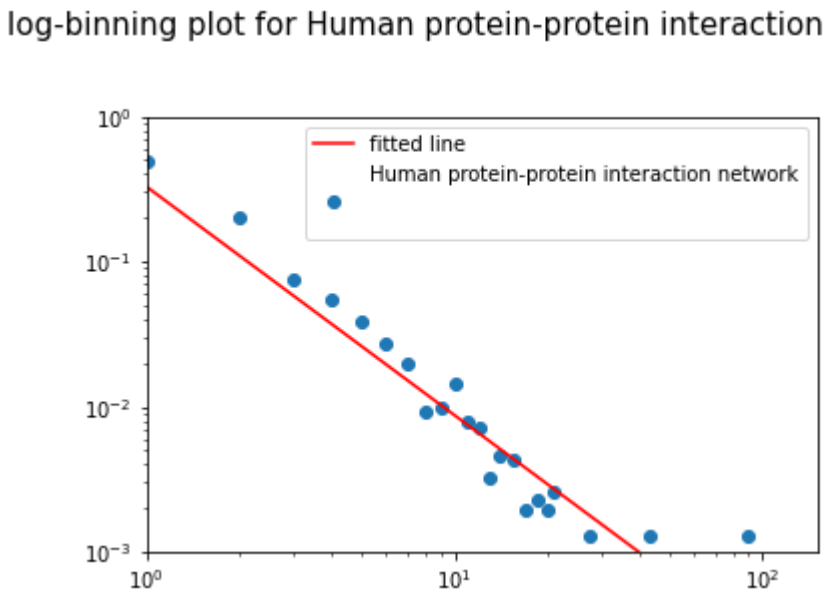
.

.

Generate simple log-binning plot for H-I-05 (human protein-protein interaction) network

```
# Generate simple log-binning plot for H-I-05 (human protein-protein interaction) ne
G = nx.read_edgelist("/content/H-I-05.tsv")
plotLogBinPlot(G,"Human protein-protein interaction network\n\n")
```

```
R-squared: 0.893449
Slope of line -1.5741381368614518
```



Values for Human protein-protein interaction network for log-binning plot are as below:

1. R-squared: 0.893449
2. Gamma: -1.5741381368614518

.

.

.

.

.

Function to Calculate the cumulative complementary distribution function

```
# Calculate the cumulative complementary distribution function
def calculate_cumulative_distribution(G):
  # Initialize all the parameters for the network
  n = G.number_of_nodes() # Total number of nodes
  attach_cont = [i for i in range(1,5)] # Uniform attachment contribution
  c = 2 # k_out

  ccdfs = []
  indegrees = []
  for r in attach_cont:
      p = c/(c+r) # Attachment probability
      x = [0] * (c*n) # Store the complete network
      x[0:11] = [2, 3, 4, 1, 3, 4, 1, 2, 4, 1, 2, 3] # Initial clique seed
      x.pop()

      # Iterate through all the vertices
      for t in range(5,n+1):
          # Iterate through each out-edge
          for j in range(0,c):
              # Generate a random number between 0 and 1
              # and check if it is less than p
              if (random.uniform(0,1) < p):
                  # choose an element uniformly at random
                  # from the list of targets
                  d = x[random.randint(0, c*(t-1))]
                  #print(d)
              else:
                  # choose a vertex uniformly at random
                  # from the set of all vertices
                  d = random.randint(1, t-1)
                  #print(d)
              x[c*(t-1) + j] = d

      # Initialize dictionary for counting the number of times
      # a node appears in the target list
      target_count = {}
      for i in x:
          if (i not in target_count):
              target_count[i] = 1
          else:
              target_count[i] += 1

```

```
    # Calculate the in-degree of nodes and store them in a dictionary
    ct = list(target_count.values())
    ct_dict = {}
    for c in ct:
        if (c not in ct_dict):
            ct_dict[c] = 1
        else:
            ct_dict[c] += 1


    # Calculate the cumulative complementary distribution function
    ccdf_dict = {}
    for key, value in ct_dict.items():
        #larger = [i for i in list(ct_dict.keys()) if key <= i]
        larger = []
        lt = list(ct_dict.keys())
        for i in lt:
            if (key <= i):
                larger.append(i)
        su = 0
        for x in larger:
            su += ct_dict[x]
        ccdf_dict[key] = su

    # Store the x and y values for plotting the ccdf
    x = []
    y = []
    for key, value in ccdf_dict.items():
        x.append(key)
        y.append(value/(n))

    # Store the graphs for different values of r
    indegrees.append(x)
    ccdfs.append(y)
    return indegrees,ccdfs
```

This function is to plot complementary cumulative distribution for network

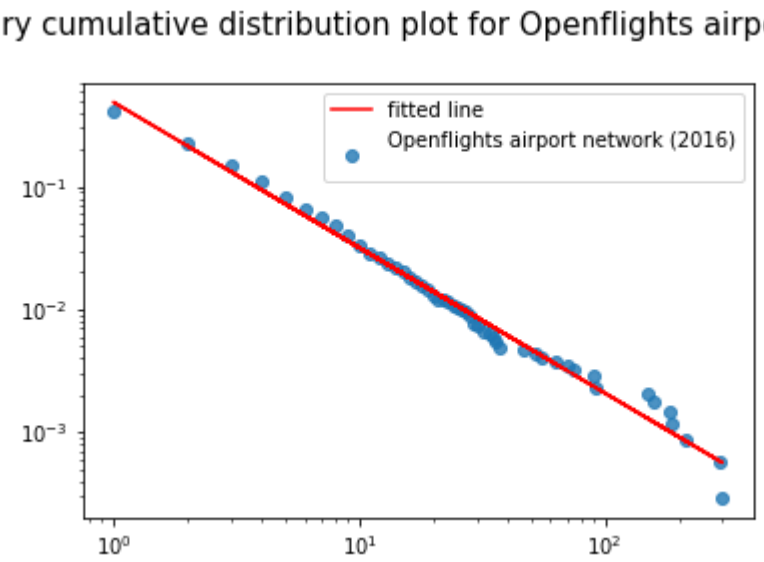
```
#This function is to plot complementary cumulative distribution for network
def plotCommulativeDistributionPlot(G,data_set_name):
  # Calculate complementary cumulative distribution
  indegrees,ccdfs =calculate_cumulative_distribution(G)
  for x,y in zip(indegrees, ccdfs):
    x = np.log10(np.asarray(x).astype(np.float))
    y = np.log10(np.asarray(y).astype(np.float))
  res = stats.linregress(x,y)
  print(f"R-squared: {res.rvalue**2:.6f}")
```

```
  print ('Slope of line', res.slope)
  print()
  ax = plt.gca()
  ax.set_xscale('log')
  ax.set_yscale('log')
  plt.plot(np.power(10,x), np.power(10,(res.intercept + (res.slope)*x)), 'r', label=
  for x,y in zip(indegrees, ccdfs):
    ax.scatter(x,y, alpha = 0.8,label= data_set_name)
  plt.title('Complementary cumulative distribution plot for '+data_set_name, fontsiz
  plt.yscale('log')
  plt.xscale('log')
  plt.legend()
  plt.show()
```

Generate simple complementary cumulative distribution for openflight network

```
# Generate simple complementary cumulative distribution  for openflight network
G = nx.read_edgelist("/content/out.openflights")
plotCommulativeDistributionPlot(G,"Openflights airport network (2016)\n")
```

```
R-squared: 0.986347
Slope of line -1.1843150308141723
```



Values for simple complementary cumulative distribution for openflight network

1. R-squared: 0.986439
2. Gamma: -1.2449401510843168

.

.

.

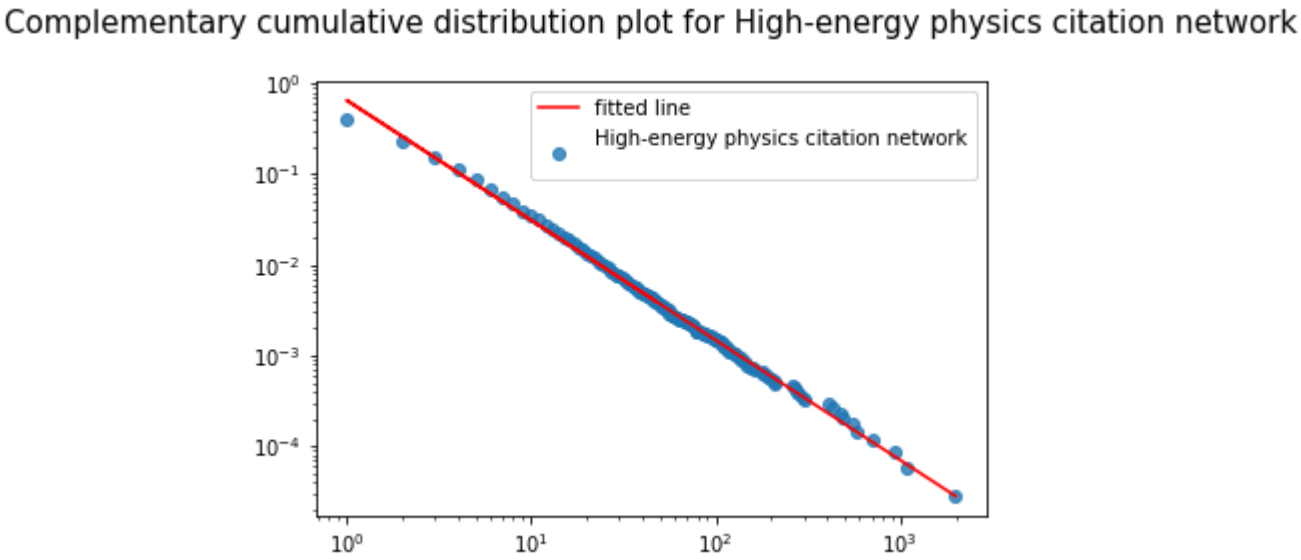
.

.

Generate simple complementary cumulative distribution plot for High-energy physics citation network

```
# Generate simple complementary cumulative distribution plot for High-energy physics
G = nx.read_edgelist("/content/Cit-HepPh.txt")
plotCommulativeDistributionPlot(G,"High-energy physics citation network\n")
```

```
R-squared: 0.997470
Slope of line -1.323225946025517
```



Values for simple complementary cumulative distribution plot for High-energy physics citation network :

1. R-squared: 0.996355
2. Gamma: -1.3154947837057414

.

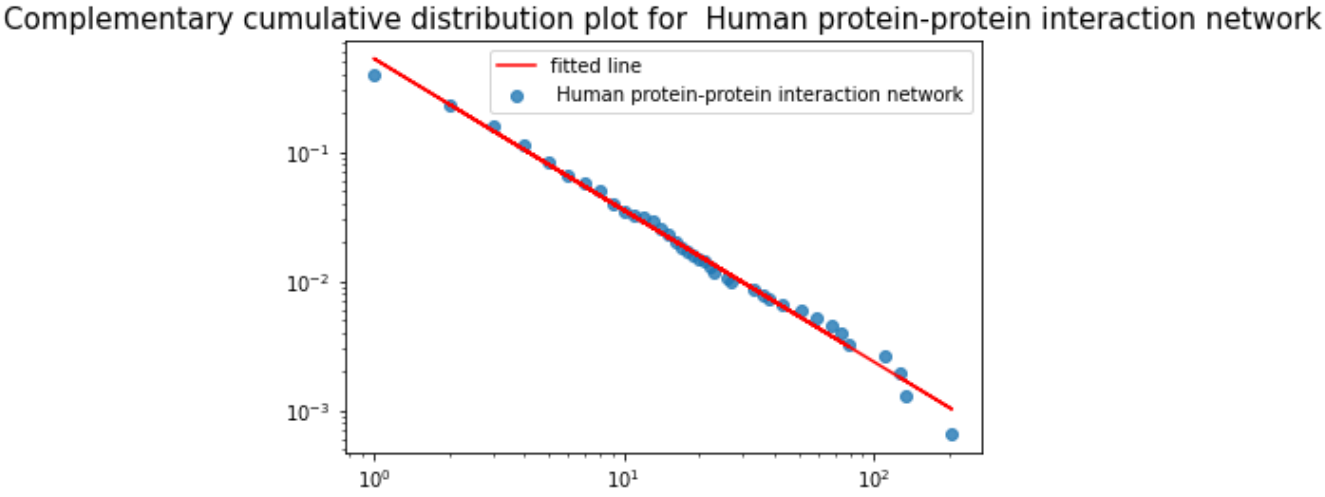
.

.

Generate simple complementary cumulative distribution plot for H-I-05 (human protein-protein interaction) network

```
# Generate simple complementary cumulative distribution plot
#for H-I-05 (human protein-protein interaction) network
G = nx.read_edgelist("/content/H-I-05.tsv")
plotCommulativeDistributionPlot(G," Human protein-protein interaction network")
```

```
R-squared: 0.991479
Slope of line -1.1748178067124613
```



Values for simple complementary cumulative distribution plot for H-I-05 (human protein-protein interaction) network

1. R-squared: 0.991479
2. Gamma: -1.1748178067124613

.

.

.

.

Question2

2. (a) Implement the preferential attachment mixture model discussed in class.

```
#function to implement the preferential attachement model
def preferential_attachement_model(G,m,p):
  n = G.number_of_nodes()
  # For probabilty zero, The graph is complete perferntial attachment
  if p==0:
    # List of existing nodes, with nodes repeated once for each adjacent edge
    G = nx.complete_graph(m)
    repeated_nodes = [n for n, d in G.degree() for _ in range(d)]
    # Start adding the other n - m0 nodes.
    source = len(G)
    while source < n:
        # Now choose m unique nodes from the existing nodes
        # Pick uniformly from repeated_nodes (preferential attachment)
        targets = set()
        while len(targets) < m:
            x = random.choice(repeated_nodes)
            targets.add(x)
        # Add edges to m nodes from the source.
        G.add_edges_from(zip([source] * m, targets))
        # Add one node to the list for each new edge just created.
        repeated_nodes.extend(targets)
        # And the new node "source" has m edges to add to the list.
        repeated_nodes.extend([source] * m)

        source += 1
  # For probabilty One, The graph is complete random ,No preferential attchement
  elif p==1:
    degree_sequence = [d+2 for n, d in G.degree()]
    G = nx.configuration_model(degree_sequence)
  else:
    # Add m initial nodes (m0 in barabasi-speak)
    G = nx.complete_graph(m)
    # List of nodes to represent the preferential attachment random selection.
    preferential_attachment = []
    preferential_attachment.extend(range(m))

    # Start adding the other n-m nodes. The first node is m.
    new_node = m
    while new_node < n:
        # Total number of edges of a Clique of all the nodes
        g_degree = len(G) - 1
        g_size = (len(G) * g_degree) / 2
```

```
        g_size = (len(G) * g_degree) // 2
        # Adding m new edges, if there is room to add them
        if random.random() < p and G.size() <= g_size - m:
            # Select the nodes where an edge can be added
            elligible_nodes = [nd for nd, deg in G.degree() if deg < g_degree]
            for i in range(m):
                # Choosing a random source node from elligible_nodes
                src_node = random.choice(elligible_nodes)
                # Picking a possible node that is not source node or neighbor with s
                prohibited_nodes = list(G[src_node])
                prohibited_nodes.append(src_node)
                # This will raise an exception if the sequence is empty
                dest_node = random.choice([nd for nd in preferential_attachment if n
                G.add_edge(src_node, dest_node) # Adding the new edge

                # Appending both nodes to add to their preferential attachment
                preferential_attachment.append(src_node)
                preferential_attachment.append(dest_node)

                # Adjusting the elligible nodes. Degree may be saturated.
                if G.degree(src_node) == g_degree:
                    elligible_nodes.remove(src_node)
                if (G.degree(dest_node) == g_degree and dest_node in elligible_nodes
                    elligible_nodes.remove(dest_node)
        # Adding new node with m edges
        else:
            # Select the edges' nodes by preferential attachment
            targets = set()
            while len(targets) < m:
                x = random.choice(preferential_attachment)
                targets.add(x)
            G.add_edges_from(zip([new_node] * m, targets))
            # Add one node to the list for each new edge just created.
            preferential_attachment.extend(targets)
            # The new node has m edges to it, plus itself: m + 1
            preferential_attachment.extend([new_node] * (m + 1))
            new_node += 1
    return G
```

.

.

.

.

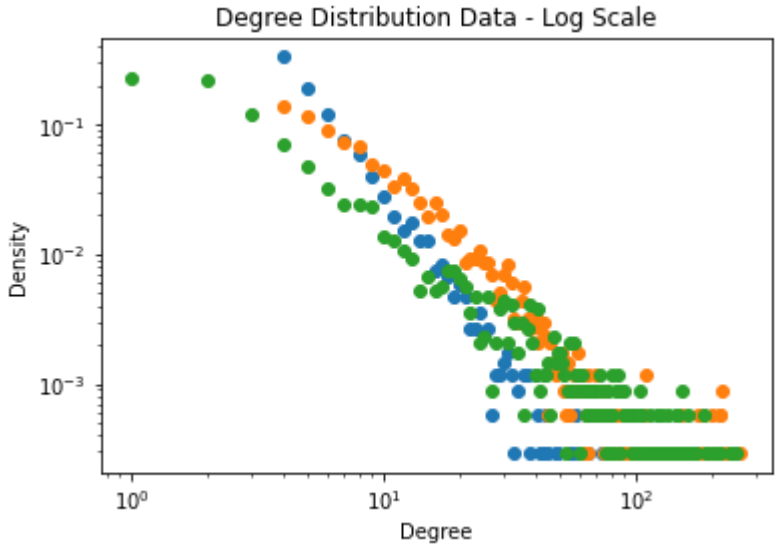
.

2. **(b)Generate three random networks using this model with m0 = m = 4, an initially complete graph, and α ∈ {0, 1/2, 1}. Plot the degree distributions for these networks on the same set of axes. Describe the differences between these distributions and discuss why these differences make sense in the context of preferential attachment.**

Function to show the degree distribution of Graph

```
#Function to show the degree distribution of Graph
def degreeDistribution(G):
  degrees = {}
  degreeList = [G.degree(v) for v in G.nodes()]
  for deg in degreeList:
    degrees[deg] = degrees.get(deg, 0) + 1
  (X, Y) = zip(*[(key, degrees[key]/len(G)) for key in degrees])
  return X,Y

```

```
G = nx.read_edgelist("/content/out.openflights")

# graph with probability zero
G1= preferential_attachement_model(G,4, 0)
X1,Y1 = degreeDistribution(G1)
plt.scatter(X1, Y1,label='alpha = 0')

# graph with probability 0.5
G2= preferential_attachement_model(G,4, 0.5)
X2,Y2 = degreeDistribution(G2)
plt.scatter(X2, Y2,label='alpha = 0.5')

# graph with probability 1
G3 = preferential_attachement_model(G,4, 1)
X3,Y3 = degreeDistribution(G)
plt.scatter(X3, Y3,label="alpha = 1")

plt.yscale('log')
plt.xscale('log')
plt.title('Degree Distribution Data - Log Scale')
plt.xlabel('Degree')
plt.ylabel('Density')
plt.show()
```



Based on generated three graphs for α with {0, 1/2, 1}.

1. For α as 0, we see a linear curve or close to it when there is preferential attachment and on the contrary
2. when α is 1 there is no linear curve on log-log plot.
3. With α as 1/2, we see a mixture of both and hence, can say that preferential attachment is partially present but not across all vertices and some were chose based on random uniformity. Which is also sufficient to state that this could be a possible candidate for scale-free network as it supports both growth and preferential attachment with bit of a deviation.

.

.

.

.

.

.

.

.

Question3

3. Implement a function betweenness(G, v) in Python that calculates the betweenness centrality of a given vertex v in the graph G. The all_shortest_paths function in NetworkX may be helpful.

Function to Compute the betweenness centrality for each family

```
#Computing the betweenness centrality for each family
def betweenness(graph_2, node='6'):
    betweenness_centralities = {} # Dictionary for storing the betweenness centralit
    total_edges = len(graph_2.edges())
    total_paths = 0
    list_of_paths = []
    passes_through_i = 0
    #total number of vertices in a graph
    total_vertices = len(graph_2.nodes())
    vertices = list(list(graph_2.nodes()))

    num_of_nodes=len(vertices)
    bc=0
    for i in range(0,num_of_nodes):
      if vertices[i]==node:
        continue
      for j in range(i+1, num_of_nodes):
        if vertices[j]==node:
          continue
        #Calculate shorest path distance
        shortest_paths=nx.all_shortest_paths(graph_2, vertices[i], vertices[j])
        num_of_shortest_passing_through_node=0
        num_of_paths=-1
        #process the shortest path lengths
        for path in shortest_paths:
          if node in path:
            num_of_shortest_passing_through_node+=1
            num_of_paths+=1
          if num_of_paths == 0:
            num_of_paths+=1
        bc+=(num_of_shortest_passing_through_node/num_of_paths)
    sbc=(2*bc)/((num_of_nodes-1)*(num_of_nodes-2))
    return sbc
```

.

Verification of betweenness function with networkx in built function

Using the betweenness (G,v) function I was able to determine the betweenness centrality of medici database, which matches with networkx function. Please see the code demo as below.

```
#Load the data
G = nx.read_adjlist("/content/medici_edge_list.txt")

#Find betweenness centrality with node 8
centrality_8 = betweenness(G,'8')

#Betweenness centrality processing
centrality_betweenness = nx.betweenness_centrality(G)
sorted_betweenness = sorted(((v, '{:0.2f}'.format(c)) for v, c in centrality_between
             key=lambda x: x[1],reverse=True )

#Verify with betweenness centrality function of networkx
print('betweenness centrality with node 8 :',centrality_8)
print('betweenness centrality with node 8 (NetworkX Function)',sorted_betweenness[0]
```

```
betweenness centrality with node 8 : 0.5494505494505495
betweenness centrality with node 8 (NetworkX Function) ('8', '0.52')
```

The values for node 8 matches with original networkx function

.

.

.

.

.

.

.

.

Question4

4. a) Determine the degree centrality, harmonic centrality, eigenvector centrality, and betweenness centrality of each vertex (you may use functions available in NetworkX). For each measure, make a table with the families ranked by importance. Show these tables side by side.

```
#Read data from medici adj list
G = nx.read_adjlist("/content/medici_adj_list.txt")

#Determine the degree Centrality
centrality = nx.degree_centrality(G)
sorted_centrality = sorted(((v, '{:0.2f}'.format(c)) for v, c in centrality.items())
             key=lambda x: x[1],reverse=True )

#Determine the eigenvector Centrality
eigenvector= nx.eigenvector_centrality(G)
sorted_eigenvector = sorted(((v, '{:0.2f}'.format(c)) for v, c in eigenvector.items(
             key=lambda x: x[1],reverse=True )

#Determine the betweenness Centrality
betweenness = nx.betweenness_centrality(G)
sorted_betweenness = sorted(((v, '{:0.2f}'.format(c)) for v, c in betweenness.items(
             key=lambda x: x[1],reverse=True )

#Determine the harmonic Centrality
harmonic = nx.harmonic_centrality(G)
sorted_harmonic = sorted(((v, '{:0.2f}'.format(c)) for v, c in harmonic.items()),
             key=lambda x: x[1],reverse=True )

#Logic to print the values in the form of table
family_ids = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
families = ["Acciaiuoli", "Albizzi", "Barbadori", "Bischeri", "Castellani",
            "Ginori", "Guadagni", "Lamberteschi", "Medici", "Pazzi",
            "Peruzzi", "Pucci", "Ridolfi", "Salviati", "Strozzi",
            "Tornabuoni"]

family = dict(zip(family_ids, families))

list_of_deg_cent = []
list_of_harm_cent = []
list_of_eig_cent = []
list_of_bet_cent = []

#Degree centrality processing
sz = np.shape(sorted_centrality)
```

```
for i in range(sz[0]):
    temp = sorted_centrality[i]
    ids = int(temp[0])
    list_of_deg_cent.append((families[ids], temp[1]))

#Harmonic centrality processing
sz = np.shape(sorted_harmonic)
for i in range(sz[0]):
    temp = sorted_harmonic[i]
    ids = int(temp[0])
    list_of_harm_cent.append((families[ids], temp[1]))

#Eigenvector centrality processing
sz = np.shape(sorted_eigenvector)
for i in range(sz[0]):
    temp = sorted_eigenvector[i]
    ids = int(temp[0])
    list_of_eig_cent.append((families[ids], temp[1]))

#Betweenness centrality processing
sz = np.shape(sorted_betweenness)
for i in range(sz[0]):
    temp = sorted_betweenness[i]
    ids = int(temp[0])
    list_of_bet_cent.append((families[ids], temp[1]))

#Plot data in the form of table
print("Degree Centrality \t\t| Harmonic Centrality \t\t| EigenVector Centrality \t |
res = "\n".join("{}\t\t | {}\t\t | {}\t\t | {}".format(w, x, y, z) for w, x, y, z
                in zip(list_of_deg_cent, list_of_harm_cent, list_of_eig_cent, list_o
print(res)
```

```
Degree Centrality               | Harmonic Centrality           | EigenVector Centrality
('Medici', '0.40')               | ('Medici', '9.50')            | ('Medici', '0.43')
('Guadagni', '0.27')             | ('Guadagni', '8.08')          | ('Strozzi', '0.36')
('Strozzi', '0.27')              | ('Ridolfi', '8.00')           | ('Ridolfi', '0.34')
('Albizzi', '0.20')              | ('Strozzi', '7.83')           | ('Tornabuoni', '0.33
('Castellani', '0.20')           | ('Albizzi', '7.83')           | ('Guadagni', '0.29')
('Bischeri', '0.20')             | ('Tornabuoni', '7.83')                | ('Bischeri',
('Peruzzi', '0.20')              | ('Bischeri', '7.20')          | ('Peruzzi', '0.28')
('Tornabuoni', '0.20')           | ('Barbadori', '7.08')                 | ('Castellani
('Ridolfi', '0.20')              | ('Castellani', '6.92')                | ('Albizzi',
('Barbadori', '0.13')            | ('Peruzzi', '6.78')           | ('Barbadori', '0.21')
('Salviati', '0.13')             | ('Salviati', '6.58')          | ('Salviati', '0.15')
('Acciaiuoli', '0.07')           | ('Acciaiuoli', '5.92')                | ('Acciaiuoli
('Ginori', '0.07')               | ('Lamberteschi', '5.37')              | ('Lamberteschi
('Lamberteschi', '0.07')                    | ('Ginori', '5.33')            | ('Ginori', '0
('Pazzi', '0.07')                | ('Pazzi', '4.77')             | ('Pazzi', '0.04')
('Pucci', '0.00')                | ('Pucci', '0.00')             | ('Pucci', '0.00')
```

.

Below is the table for Centralities of a network : table with the families ranked by importance

Degree Centrality	Harmonic Centrality	EigenVector Centrality	Betweenness Centrality
('Medici', '0.40')	('Medici', '9.50')	('Medici', '0.43')	('Medici', '0.45')
('Guadagni', '0.27')	('Guadagni', '8.08')	('Strozzi', '0.36')	('Guadagni', '0.22')
('Strozzi', '0.27')	('Ridolfi', '8.00')	('Ridolfi', '0.34')	('Albizzi', '0.18')
('Albizzi', '0.20')	('Strozzi', '7.83')	('Tornabuoni', '0.33')	('Salviati', '0.12')
('Castellani', '0.20')	('Tornabuoni', '7.83')	('Guadagni', '0.29')	('Ridolfi', '0.10')
('Bischeri', '0.20')	('Albizzi', '7.83')	('Bischeri', '0.28')	('Bischeri', '0.09')
('Peruzzi', '0.20')	('Bischeri', '7.20')	('Peruzzi', '0.28')	('Strozzi', '0.09')
('Tornabuoni', '0.20')	('Barbadori', '7.08')	('Castellani', '0.26')	('Barbadori', '0.08')
('Ridolfi', '0.20')	('Castellani', '6.92')	('Albizzi', '0.24')	('Tornabuoni', '0.08')
('Barbadori', '0.13')	('Peruzzi', '6.78')	('Barbadori', '0.21')	('Castellani', '0.05')
('Salviati', '0.13')	('Salviati', '6.58')	('Salviati', '0.15')	('Peruzzi', '0.02')
('Acciaiuoli', '0.07')	('Acciaiuoli', '5.92')	('Acciaiuoli', '0.13')	('Acciaiuoli', '0.00')
('Ginori', '0.07')	('Lamberteschi', '5.37')	('Lamberteschi', '0.09')	('Ginori', '0.00')
('Lamberteschi', '0.07')	('Ginori', '5.33')	('Ginori', '0.07')	('Lamberteschi', '0.00')
('Pazzi', '0.07')	('Pazzi', '4.77')	('Pazzi', '0.04')	('Pazzi', '0.00')
('Pucci', '0.00')	('Pucci', '0.00')	('Pucci', '0.00')	('Pucci', '0.00')

4. (b) Discuss your results. How important was the Medici family with respect to this network? What family was the second most important?

1. From the table, it is evident that Medici family is the most important family in the network given the fact that within each centrality 'Medici' family occupies highest rank or makes to top of the list. Hence, it is safe to state that 'Medici' family owns a structurally important position in the network as per Padgett and Ansell's claim.
2. For second position, there are couple of contenders based on all data across the tables. As per degree, harmonic and betweenness centrality column values, 'Guadagni' is second most important family in the network. On the other hand, 'Ridolfi' is second most important families as per eigenvector centrality column values. I am more inclined towards 'Guadagni' as it has second rank acorss 3 different centrality measures (degree, harmonic and betweenness centrality).

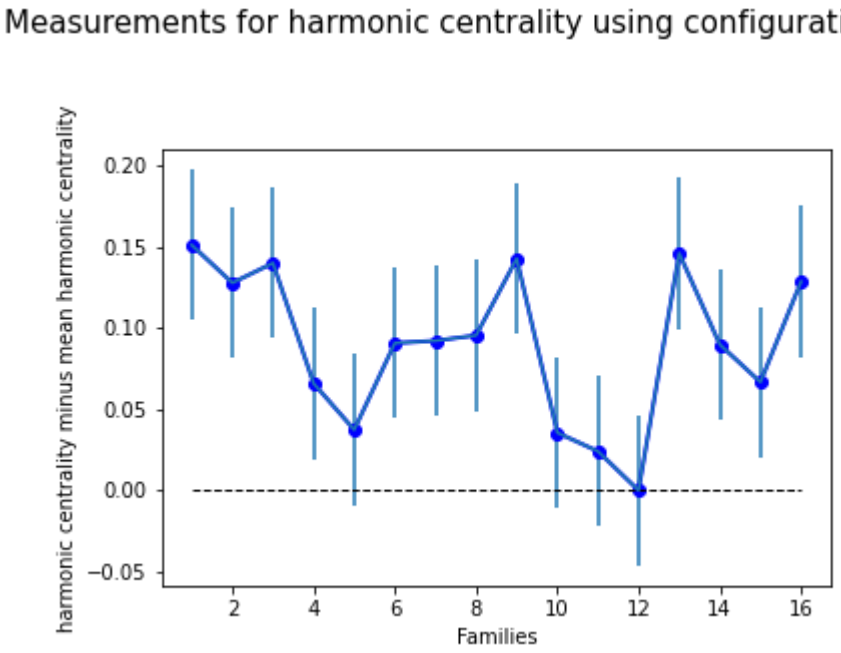
4. (c)Are our measurements for harmonic centrality unusual given the degree distribution of this network? Using the degree sequence of this network, generate 10,000 random networks using the configuration model and 10,000 random networks using degree preserving randomization. Make two figures, one for each type of random network, with families on the x-axis and harmonic centrality minus mean harmonic centrality on the y-axis. Include error bars representing one standard deviation for each point. Discuss what you find.

Yes Looking at the table created in 4(a) the measurements of harmonic centrality unusual given the degree distribution of this network. The values of harmonic centralities are far away from other centralities measurements

10,000 random networks using configuration model

```
#Load data from Medici adj list
graph_A = nx.read_adjlist('/content/medici_adj_list.txt')

#Calculate harmanic centrality of graph
harmonic= nx.harmonic_centrality(graph_A)

#perform data preprocessing
hc_A = sorted(harmonic.items(), key=operator.itemgetter(0))
hc_B = np.zeros((len(graph_A),2))
data = np.zeros((len(graph_A), 10000))

#degree sequence
deg_seq = [1,3,2,3,3,1,4,1,6,1,3,0,3,2,4,3]

#configuration model
graph=nx.configuration_model(deg_seq)

dict_nodes_harm = {}
sum_arr = np.zeros((len(graph),5))
sum_arr[:,0] = range(len(graph))
count = 1

# Generating the 10,000 random graphs from the degree sequence using the configurati
while (count <= 10000):
    graph=nx.configuration_model(deg_seq)
    harmonic= nx.harmonic_centrality(graph)
    hc = sorted(harmonic.items(), key=operator.itemgetter(0))
    for i in range(len(graph)):
        sum_arr[i,1] += hc[i][1]
        data[i, count-1] = hc[i][1]
    count += 1
```

```
32
33 sum_arr[:,1] = sum_arr[:,1]/10000
34
35 for i in range(len(graph_A)):
36     hc_B[i,0]=i;
37     ind = int(hc_A[i][0])
38     hc_B[ind,1] = hc_A[i][1]
39
40 #Calculate the mean value
41 for i in range(len(data)):
42     sum_arr[i,4] =  (hc_B[i,1] - sum_arr[i,1])/10 # Mean Value
43
44 # Plotting the graph
45 fig, ax = plt.subplots()
46 ax.plot(sum_arr[:,0]+1.,sum_arr[:,4],'-o',lw=2,color='blue')
47 ax.plot(np.linspace(1, 16, 100, endpoint=True),np.zeros((100,1)),'--',lw=1,color='bl
48 ax.errorbar(sum_arr[:,0]+1., sum_arr[:,4], yerr=np.std(sum_arr[:,4]))
49 plt.xlabel('Families')
50 plt.ylabel('harmonic centrality minus mean harmonic centrality')
51 plt.title('Measurements for harmonic centrality using configuration model\n\n\n',fon
52 plt.show()
```



.

.

.

.

Function to determine the degree preserving randamization

```
#Function to determine the degree preserving randamization
def degPresRand(G, rewires=24):
    n = 0
    swapcount = 0
    keys, degrees = zip(*G.degree())  # keys, degree
    cdf = nx.utils.cumulative_distribution(degrees)  # cdf of degree
    discrete_sequence = nx.utils.discrete_sequence
    while swapcount < rewires:
        # if random.random() < 0.5: continue # trick to avoid periodicities?
        # pick two random edges without creating edge list
        # choose source node indices from discrete distribution
        (ui, xi) = discrete_sequence(2, cdistribution=cdf, seed=random)
        if ui == xi:
            continue  # same source, skip
        u = keys[ui]  # convert index to label
        x = keys[xi]
        # choose target uniformly from neighbors
        v = random.choice(list(G[u]))
        y = random.choice(list(G[x]))
        if v == y:
            continue  # same target, skip
        if (x not in G[u]) and (y not in G[v]):  # don't create parallel edges
            G.add_edge(u, x)
            G.add_edge(v, y)
            G.remove_edge(u, v)
            G.remove_edge(x, y)
            swapcount += 1
        n += 1
    return G
```

.

.

.

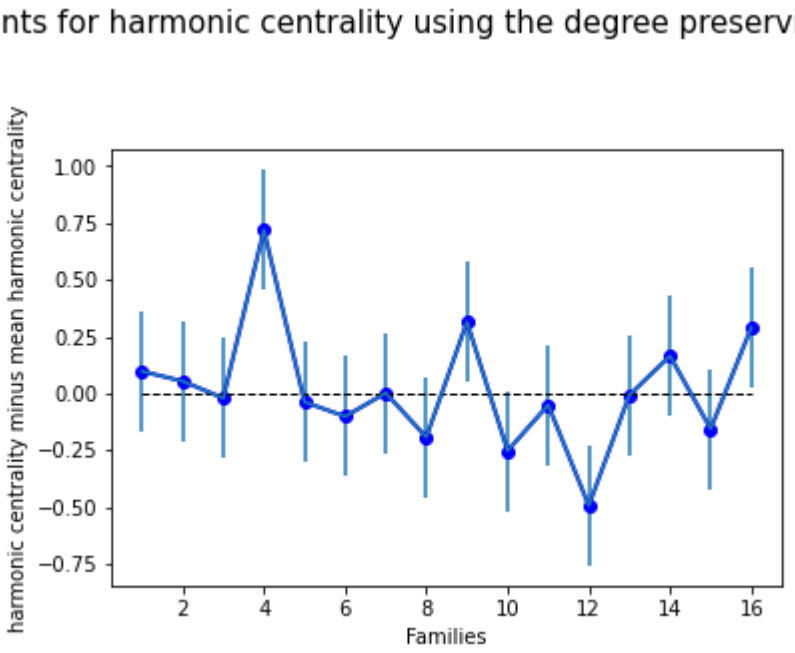
.

.

10,000 random networks using degree preserving randomization

```
#Load data from Medici adj list
graph_A = nx.read_adjlist('/content/medici_adj_list.txt')

#Calculate harmanic centrality of graph
harmonic= nx.harmonic_centrality(graph_A)

#perform data preprocessing
hc_A = sorted(harmonic.items(), key=operator.itemgetter(0))
hc_B = np.zeros((len(graph_A),2))
data = np.zeros((len(graph_A), 10000))
#Degree Sequence
deg_seq = [1,3,2,3,3,1,4,1,6,1,3,0,3,2,4,3]

#degree preserving randamization
graph=degPresRand(graph_A)

dict_nodes_harm = {}
sum_arr = np.zeros((len(graph),5))
sum_arr[:,0] = range(len(graph))
count = 1

# Generating the 10,000 random graphs from the degree sequence using the degree pres
while (count <= 10000):
    graph=degPresRand(graph)
    harmonic= nx.harmonic_centrality(graph)
    hc = sorted(harmonic.items(), key=operator.itemgetter(0))
    for i in range(len(graph)):
        sum_arr[i,1] += hc[i][1]
        data[i, count-1] = hc[i][1]
    count += 1

sum_arr[:,1] = sum_arr[:,1]/10000

for i in range(len(graph_A)):
    hc_B[i,0]=i;
    ind = int(hc_A[i][0])
    hc_B[ind,1] = hc_A[i][1]

#Calculate the mean value
for i in range(len(data)):
    sum_arr[i,4] =  (hc_B[i,1] - sum_arr[i,1])/10 # Mean Value

# Plotting the graph
fig, ax = plt.subplots()
ax.plot(sum_arr[:,0]+1.,sum_arr[:,4],'-o',lw=2,color='blue')
```

```
46
47
48 ax.plot(np.linspace(1, 16, 100, endpoint=True),np.zeros((100,1)),'--',lw=1,color='bl
49 ax.errorbar(sum_arr[:,0]+1., sum_arr[:,4], yerr=np.std(sum_arr[:,4]))
50 plt.xlabel('Families')
51 plt.ylabel('harmonic centrality minus mean harmonic centrality')
52 plt.title('Measurements for harmonic centrality using the degree preserving randamiz
53 plt.show()
```



Yes, our harmonic centrality measurements are little off compared to degree distribution of this network.

1. Based on Y-axis plot using config model for harmonic centrality - mean harmonic centrality, all networks are above the mean except for family Ridolfi which is an exactly equivalent to mean value of harmonic centrality.
2. Based on Y-axis plot using degree preserving randomization for harmonic centrality - mean harmonic centrality, 3 families Bischeri, Lamberteschi, and Salviati are exactly equivalent to mean value of harmonic centrality. For first plot with config model, there are no families which fall below mean harmonic centrality but in second plot with degree preserving randomization, there are 7 families which fall below mean harmonic centrality.
3. Harmonic centrality across different networks is well sparsed with degree preserving randomization compared to config model as it has equal distribution across families below and above mean value of harmonic centrality.

.

.

.

✓ 2s completed at 7:26 PM