Social Network Analysis with Python

1. Preparing Data

Getting all the emails

Entrée [1]:

```
# Import needed libraries
import glob
import os
import re
from pprint import pprint
import numpy as np
import pandas as pd
import codecs as co
from email.parser import Parser
import networkx as nx
import matplotlib.pyplot as plt
import community
```

Entrée [2]:

```
# Get paths to all subdirectories 'inbox'
usernames = []
email = []
usernames = os.listdir("C:/Users/dell/Desktop/M3/SNA/maildir")
#pprint(usernames)
for email in glob.iglob('C:/Users/dell/Desktop/M3/SNA/maildir/**/inbox/*', recursive=True):
    pprint(email)
```

```
'C:/Users/dell/Desktop/M3/SNA/maildir\\allen-p\\inbox\\1'
'C:/Users/dell/Desktop/M3/SNA/maildir\\allen-p\\inbox\\10'
'C:/Users/dell/Desktop/M3/SNA/maildir\\allen-p\\inbox\\11'
'C:/Users/dell/Desktop/M3/SNA/maildir\\allen-p\\inbox\\12'
'C:/Users/dell/Desktop/M3/SNA/maildir\\allen-p\\inbox\\13'
'C:/Users/dell/Desktop/M3/SNA/maildir\\allen-p\\inbox\\14'
'C:/Users/dell/Desktop/M3/SNA/maildir\\allen-p\\inbox\\15'
'C:/Users/dell/Desktop/M3/SNA/maildir\\allen-p\\inbox\\16'
'C:/Users/dell/Desktop/M3/SNA/maildir\\allen-p\\inbox\\17'
'C:/Users/dell/Desktop/M3/SNA/maildir\\allen-p\\inbox\\18'
'C:/Users/dell/Desktop/M3/SNA/maildir\\allen-p\\inbox\\19'
'C:/Users/dell/Desktop/M3/SNA/maildir\\allen-p\\inbox\\2'
'C:/Users/dell/Desktop/M3/SNA/maildir\\allen-p\\inbox\\20'
'C:/Users/dell/Desktop/M3/SNA/maildir\\allen-p\\inbox\\21'
'C:/Users/dell/Desktop/M3/SNA/maildir\\allen-p\\inbox\\22'
'C:/Users/dell/Desktop/M3/SNA/maildir\\allen-p\\inbox\\23'
'C:/Users/dell/Desktop/M3/SNA/maildir\\allen-p\\inbox\\24'
'C:/Users/dell/Desktop/M3/SNA/maildir\\allen-p\\inbox\\25'
'C:/Users/dell/Desktop/M3/SNA/maildir\\allen-p\\inbox\\26'
```

Entrée [5]:

```
# Keep only Employees who sent emails
users = []
userSend = []
for userSend in glob.iglob('C:\\Users\\dell\\Desktop\\M3\\SNA\\maildir\\**\\sent\\1', recur
    if userSend :
        users.append(userSend)
print(len(users))

ListUserSend=[]
for i in range(len(users)):
    x=users[i].split('\\')
    ListUserSend.append(x[7])
pprint(ListUserSend)
```

```
89
['allen-p',
 'arora-h',
 'bass-e',
 'beck-s',
 'benson-r',
 'brawner-s',
 'buy-r',
 'campbell-l',
 'carson-m',
 'cash-m',
 'cuilla-m',
 'dasovich-j',
 'davis-d',
 'dean-c',
 'delainey-d',
 'derrick-j',
 'dickson-s',
 'donohoe-t',
 'dorland-c',
 'ermis-f',
 'farmer-d',
 'fischer-m',
 'fossum-d',
 'gay-r',
 'germany-c',
 'giron-d',
 'grigsby-m',
 'haedicke-m',
 'hayslett-r',
 'heard-m',
 'hernandez-j',
 'hodge-j',
 'horton-s',
 'hyvl-d',
 'jones-t',
 'kaminski-v',
 'kean-s',
 'keavey-p',
 'king-j',
 'kuykendall-t',
 'lavorato-j',
 'lay-k',
```

```
 'lenhart-m',
 'lewis-a',
 'lokay-m',
 'love-p',
 'maggi-m',
 'mann-k',
 'martin-t',
 'may-l',
 'mcconnell-m',
 'mckay-b',
 'mclaughlin-e',
 'mims-thurston-p',
 'neal-s',
 'nemec-g',
 'pereira-s',
 'perlingiere-d',
 'presto-k',
 'quenet-j',
 'ring-a',
 'rodrique-r',
 'rogers-b',
 'ruscitti-k',
 'sager-e',
 'sanders-r',
 'scott-s',
 'shackleton-s',
 'shankman-j',
 'shapiro-r',
 'shively-h',
 'skilling-j',
 'smith-m',
 'south-s',
 'stclair-c',
 'stokley-c',
 'sturm-f',
 'symes-k',
 'taylor-m',
 'tholt-j',
 'townsend-j',
 'tycholiz-b',
 'ward-k',
 'weldon-c',
 'whalley-g',
 'whalley-l',
 'white-s',
 'ybarbo-p',
 'zipper-a']
```

Entrée [ ]:

```
# Read all the emails into a dictionary or a data structure of your choice
# You can use the path or username as key and the content of the email as value.
content=[]
dict={}
file_to_read=[]

for i in range(len(ListUserSend)):
    file_to_read = os.listdir('C:/Users/dell/Desktop/M3/SNA/maildir/'+ListUserSend[i])
    #pprint(len(file_to_read))
    for j in range(len(file_to_read)):
        f = co.open('C:/Users/dell/Desktop/M3/SNA/maildir/'+ListUserSend[i]+'/sent/'+file_t
        dict[ListUserSend[i]+file_to_read[j]] = f.read()

for key,values in dict.items():
    print(key,values)
```

Getting Employees username/emails

Entrée [7]:

```
# Get a list of usernames / emails from Sent emails
emails=[]
usernames=[]

from email.parser import Parser
usernames = os.listdir("C:\\Users\\dell\\Desktop\\M3\\SNA\\maildir")
for i in range(len(usernames)):
    for file_to_read in glob.iglob('C:\\Users\\dell\\Desktop\\M3\\SNA\\maildir\\'+usernames
        with open(file_to_read, "r") as f:
            data = f.read()
            email = Parser().parsestr(data)
            emails.append(email['from'])

            for j in range(len(emails)):

                print("email: " , emails[j],"to: ",usernames[i])
```

```
email:  phillip.allen@enron.com to:  allen-p
email:  phillip.allen@enron.com to:  arora-h
email:  harry.arora@enron.com to:  arora-h
email:  phillip.allen@enron.com to:  bass-e
email:  harry.arora@enron.com to:  bass-e
email:  eric.bass@enron.com to:  bass-e
email:  phillip.allen@enron.com to:  beck-s
email:  harry.arora@enron.com to:  beck-s
email:  eric.bass@enron.com to:  beck-s
email:  sally.beck@enron.com to:  beck-s
email:  phillip.allen@enron.com to:  benson-r
email:  harry.arora@enron.com to:  benson-r
email:  eric.bass@enron.com to:  benson-r
email:  sally.beck@enron.com to:  benson-r
email:  robert.benson@enron.com to:  benson-r
email:  phillip.allen@enron.com to:  brawner-s
email:  harry.arora@enron.com to:  brawner-s
email:  eric.bass@enron.com to:  brawner-s
email:  sally.beck@enron.com to:  brawner-s
```

Entrée [8]:

```
# Transform data structure to have emails rather than usernames or paths
emailsArobase=[]
for i in range(len(emails)):
    emailsCleaned= emails[i].split('@')
    emailsArobase.append(emailsCleaned[0])
pprint(emailsArobase)
```

```
['phillip.allen',
 'harry.arora',
 'eric.bass',
 'sally.beck',
 'robert.benson',
 'sandra.brawner',
 'rick.buy',
 'larry.campbell',
 'mike.carson',
 'michelle.cash',
 'martin.cuilla',
 'jeff.dasovich',
 'dana.davis',
 'clint.dean',
 'david.delainey',
 'james.derrick',
 'stacy.dickson',
 'tom.donohoe',
 'chris.dorland',
 'frank.ermis',
 'daren.farmer',
 'mark.fisher',
 'drew.fossum',
 'rob.gay',
 'chris.germany',
 'darron.giron',
 'mike.grigsby',
 'mark.haedicke',
 'rod.hayslett',
 'marie.heard',
 'judy.hernandez',
 'jenny.helton',
 'stanley.horton',
 'dan.hyvl',
 'tana.jones',
 'vince.kaminski',
 'steven.kean',
 'peter.keavey',
 'jeff.king',
 'tori.kuykendall',
 'angela.mcculloch',
 'rosalee.fleming',
 'matthew.lenhart',
 'andrew.lewis',
 'michele.lokay',
 'phillip.love',
 'mike.maggi',
 'kay.mann',
 'laura.vuittonet',
 'larry.may',
 'mike.mcconnell',
```

```
 'brad.mckay',
 'errol.mclaughlin',
 'patrice.mims',
 'scott.neal',
 'gerald.nemec',
 'susan.pereira',
 'debra.perlingiere',
 'tamara.black',
 'joe.quenet',
 'andrea.ring',
 'robin.rodrigue',
 'benjamin.rogers',
 'kevin.ruscitti',
 'elizabeth.sager',
 'richard.sanders',
 'susan.scott',
 'kaye.ellis',
 'jeffrey.shankman',
 'richard.shapiro',
 'hunter.shively',
 'sherri.reinartz',
 'matt.smith',
 'steven.south',
 'carol.clair',
 'fletcher.sturm',
 'kate.symes',
 'mark.taylor',
 'jane.tholt',
 'judy.townsend',
 'barry.tycholiz',
 'kim.ward',
 'v.weldon',
 'greg.whalley',
 'greg.whalley',
 'zionette.vincent',
"paul.y'barbo",
 'andy.zipper']
```

Getting Sender and Receiver information

Entrée [20]:

```
# Create a list of [sender, receiver]
# Use Regex to extract the email of the Sender of each email ('From')

lines1=[]
lines2=[]

for file_to_read1 in glob.iglob('C:\\Users\\dell\\Desktop\\M3\\SNA\\maildir\\**\\sent\\1',
    hand1 = open(file_to_read1, "r")
    for line1 in hand1:
        line = line1.rstrip()
        if re.search('^From:.+@', line1) :
            lines1.append(line1)

emailsArobaseFrom=[]
for i in range(len(lines1)):
    emailsCleanedfrom= lines1[i].split('@')
    emailsArobaseFrom.append(emailsCleanedfrom[0])
#pprint(emailsArobaseFrom)


for file_to_read2 in glob.iglob('C:\\Users\\dell\\Desktop\\M3\\SNA\\maildir\\**\\sent\\1',
    hand2 = open(file_to_read2, "r")
    for line2 in hand2:
        line2 = line2.rstrip()
        if re.search('^To:.+@', line2) :
            lines2.append(line2)

emailsArobaseTo=[]
for i in range(len(lines2)):
    emailsCleanedto= lines2[i].split('@')
    emailsArobaseTo.append(emailsCleanedto[0])
#pprint(emailsArobaseTo)

lines12 = [[]]
for i in range(len(emailsArobaseFrom)):
    for j in range (len(emailsArobaseTo)):
        lines12.append([ emailsArobaseFrom[i], emailsArobaseTo[j]])
pprint (lines12)
#print(len(lines12))
```

```
[[],
 ['From: phillip.allen', 'To: christi.nicolay'],
 ['From: phillip.allen', 'To: yolanda.roberts'],
 ['From: phillip.allen', 'To: shanna.husser'],
 ['From: phillip.allen', 'To: KATHY BASS <daphneco64'],
 ['From: phillip.allen', 'To: <blake_veal_2000'],
 ['From: phillip.allen', 'To: tjacobs'],
 ['From: phillip.allen', 'To: "\'Sally Beck\'" <Sally.Beck'],
 ['From: phillip.allen', 'To: bsunsurf'],
 ['From: phillip.allen', 'To: tammi.depaolis'],
 ['From: phillip.allen', 'To: sarah-joy.hunter'],
 ['From: phillip.allen', 'To: george.robinson'],
 ['From: phillip.allen', 'To: Larry Campbell/ET&S/Enron'],
 ['From: phillip.allen', 'To: tara.sweitzer'],
 ['From: phillip.allen', 'To: david.oxley'],
 ['From: phillip.allen', 'To: Michelle Cash/HOU/ECT'],
 ['From: phillip.allen', 'To: David Oxley/HOU/ECT'],
```

```
['From: phillip.allen', 'To: s.nadalin'],
```

Creating the Dataframe

Entrée [24]:

```
# Create a pandas Dataframe based on the list. Use pandas documentation if needed
df = pd.DataFrame()
df = pd.DataFrame(lines12,columns=['sender','receiver'])
df
```

Out[24]:

	sender	receiver
0	None	None
1	From: phillip.allen	To: christi.nicolay
2	From: phillip.allen	To: yolanda.roberts
3	From: phillip.allen	To: shanna.husser
4	From: phillip.allen	To: KATHY BASS <daphneco64
5	From: phillip.allen	To: <blake_veal_2000
6	From: phillip.allen	To: tjacobs
7	From: phillip.allen	To: "'Sally Beck'" <Sally.Beck
8	From: phillip.allen	To: bsunsurf
9	From: phillip.allen	To: tammi.depaolis
10	From: phillip.allen	To: sarah-joy.hunter
11	From: phillip.allen	To: george.robinson
12	From: phillip.allen	To: Larry Campbell/ET&S/Enron
13	From: phillip.allen	To: tara.sweitzer
14	From: phillip.allen	To: david.oxley
15	From: phillip.allen	To: Michelle Cash/HOU/ECT
16	From: phillip.allen	To: David Oxley/HOU/ECT
17	From: phillip.allen	To: s.nadalin
18	From: phillip.allen	To: "Martin Cuilla (E-mail)" <mcuilla
19	From: phillip.allen	To: steve.montovano
20	From: phillip.allen	To: nicole.mendez
21	From: phillip.allen	To: Dana Davis/HOU/ECT
22	From: phillip.allen	To: kim.melodick
23	From: phillip.allen	To: Clint Dean/Corp/Enron
24	From: phillip.allen	To: janet.dietrich
25	From: phillip.allen	To: David W Delainey/HOU/ECT
26	From: phillip.allen	To: Brett R Wiggs/SA/Enron
27	From: phillip.allen	To: jkubin
28	From: phillip.allen	To: jeffrey.hodge
29	From: phillip.allen	To: hunaid.engineer
...	...	...
10091	From: andy.zipper	To: rebecca.quenet
10092	From: andy.zipper	To: jared.kaiser

	sender	receiver
10093	From: andy.zipper	To: brian.kristjansen
10094	From: andy.zipper	To: brandon.neff
10095	From: andy.zipper	To: todd.johnson.b
10096	From: andy.zipper	To: fmdutton
10097	From: andy.zipper	To: rtellis
10098	From: andy.zipper	To: Richard B Sanders/HOU/ECT
10099	From: andy.zipper	To: ahebert
10100	From: andy.zipper	To: lisa.mackey
10101	From: andy.zipper	To: ben
10102	From: andy.zipper	To: cbone
10103	From: andy.zipper	To: ina.rangel
10104	From: andy.zipper	To: markskilling
10105	From: andy.zipper	To: jeanne.seward
10106	From: andy.zipper	To: bsitz
10107	From: andy.zipper	To: larry.hunter
10108	From: andy.zipper	To: rogers.herndon
10109	From: andy.zipper	To: frank.hayden
10110	From: andy.zipper	To: jonalan.page
10111	From: andy.zipper	To: marc.r.cutler
10112	From: andy.zipper	To: outlook.team
10113	From: andy.zipper	To: cynthia.franklin
10114	From: andy.zipper	To: bradb
10115	From: andy.zipper	To: eolhelp
10116	From: andy.zipper	To: james_pelland
10117	From: andy.zipper	To: paul.chivers
10118	From: andy.zipper	To: paul.chivers
10119	From: andy.zipper	To: jasultan
10120	From: andy.zipper	To: chris.edmonds

10121 rows × 2 columns

Entrée [30]:

```
# Add Frequency column to your dataframe
new_df=df.groupby(['sender','receiver']).size().reset_index()
new_df.columns = ['sender', 'receiver', 'frequency']
new_df
```

Out[30]:

	sender	receiver	frequency
0	From: <RMSSVS	To: "'Andrew.H.Lewis	1
1	From: <RMSSVS	To: "'Sally Beck'" <Sally.Beck	1
2	From: <RMSSVS	To: "Martin Cuilla (E-mail)" <mcuilla	1
3	From: <RMSSVS	To: 8777208398.4891940	1
4	From: <RMSSVS	To: <blake_veal_2000	1
5	From: <RMSSVS	To: Brett R Wiggs/SA/Enron	1
6	From: <RMSSVS	To: Clint Dean/Corp/Enron	1
7	From: <RMSSVS	To: Dana Davis/HOU/ECT	1
8	From: <RMSSVS	To: Daren J Farmer/HOU/ECT	1
9	From: <RMSSVS	To: Darlene C Forsyth/HOU/ECT	1
10	From: <RMSSVS	To: David Oxley/HOU/ECT	1
11	From: <RMSSVS	To: David W Delainey/HOU/ECT	1
12	From: <RMSSVS	To: Joseph W Sutton/ENRON_DEVELOPMENT	1
13	From: <RMSSVS	To: Julie Meyers/HOU/ECT	1
14	From: <RMSSVS	To: KATHY BASS <daphneco64	1
15	From: <RMSSVS	To: Kay Mann/Corp/Enron	1
16	From: <RMSSVS	To: Kenneth Lay/Corp/Enron	1
17	From: <RMSSVS	To: Larry Campbell/ET&S/Enron	1
18	From: <RMSSVS	To: Mark Fisher/EWC/Enron	1
19	From: <RMSSVS	To: Melissa Graves/HOU/ECT	2
20	From: <RMSSVS	To: Michelle Cash/HOU/ECT	1
21	From: <RMSSVS	To: Richard B Sanders/HOU/ECT	1
22	From: <RMSSVS	To: Scott Adams/CAL/ECT	1
23	From: <RMSSVS	To: Scott Neal/HOU/ECT	1
24	From: <RMSSVS	To: Steven J Kean/HOU/EES	1
25	From: <RMSSVS	To: ahebert	1
26	From: <RMSSVS	To: alove770	1
27	From: <RMSSVS	To: becky.young	1
28	From: <RMSSVS	To: ben	1
29	From: <RMSSVS	To: bradb	1
...	...	...	...
9525	From: zionette.vincent	To: marc.r.cutler	1
9526	From: zionette.vincent	To: mark.taylor	1

	sender	receiver	frequency
9527	From: zionette.vincent	To: markskilling	1
9528	From: zionette.vincent	To: monique.sanchez	1
9529	From: zionette.vincent	To: nicole.mendez	1
9530	From: zionette.vincent	To: nytasha.sims	1
9531	From: zionette.vincent	To: outlook.team	1
9532	From: zionette.vincent	To: paul.chivers	2
9533	From: zionette.vincent	To: pereira	1
9534	From: zionette.vincent	To: rebecca.carter	1
9535	From: zionette.vincent	To: rebecca.quenet	1
9536	From: zionette.vincent	To: robert.jones	1
9537	From: zionette.vincent	To: roger.balog	1
9538	From: zionette.vincent	To: rogers.herndon	1
9539	From: zionette.vincent	To: rtellis	1
9540	From: zionette.vincent	To: rusty.stevens	1
9541	From: zionette.vincent	To: s.nadalin	1
9542	From: zionette.vincent	To: sarah-joy.hunter	1
9543	From: zionette.vincent	To: scott.sefton	1
9544	From: zionette.vincent	To: shanna.husser	1
9545	From: zionette.vincent	To: shemeika.landry	1
9546	From: zionette.vincent	To: stephen.stock	1
9547	From: zionette.vincent	To: steve.montovano	1
9548	From: zionette.vincent	To: tammi.depaolis	1
9549	From: zionette.vincent	To: tara.sweitzer	1
9550	From: zionette.vincent	To: theresa.staab	1
9551	From: zionette.vincent	To: tjacobs	1
9552	From: zionette.vincent	To: tknight	1
9553	From: zionette.vincent	To: todd.johnson.b	1
9554	From: zionette.vincent	To: yolanda.roberts	1

9555 rows × 3 columns

Entrée [35]:

```
#Creating a list for frequency
#We will need it for the graph creation
frequency=[]
freq=new_df['frequency']
frequency.append(freq)
#frequency
```

2. Creating the Network

Entrée [37]:

```
# Create an undirected weighted graph using your Dataframe
# Initialize a Graph object
G = nx.Graph()
# Add nodes to the Graph
G.add_nodes_from(df)
# Add edges to the Graph
for i in range(len(lines1)):
    for j in range (len(lines2)):
        for k in range (len(frequency)):
            G.add_edge(lines1[i] ,lines2[j],weight=frequency[k])
print(nx.info(G))
```

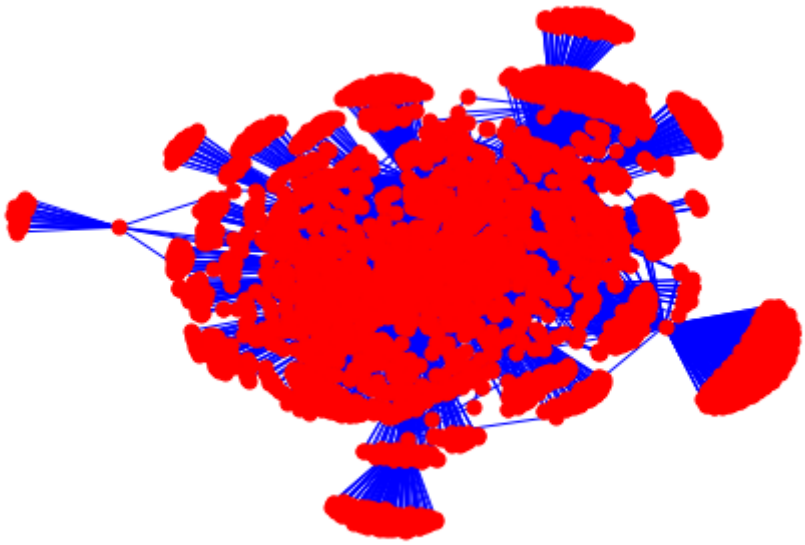
```
Name:
Type: Graph
Number of nodes: 198
Number of edges: 9555
Average degree:  96.5152
```

Entrée [ ]:

```
#creat the graph
nx.write_gexf(G, 'MailGraph.gexf')
```

Entrée [2]:

```
# Display the network
G=nx.read_gexf('C:\\Users\\dell\\Desktop\\MailGraph.gexf')
nx.draw(G,pos=nx.spring_layout(G),node_size=50,edge_color='b')
plt.show()
```



3. Analyzing your Network

Calculate network metrics

Entrée [46]:

```
## Macro-level analysis
print(nx.is_connected(G))

# Density
density = nx.density(G)
print("Network density:", density)
```

```
True
Network density: 0.0017222585567909308
```

Entrée [47]:

```
#Assortativity : measures the similarity of connections in the graph with respect to the no
r=nx.degree_assortativity_coefficient(G)
print("%3.1f"%r)
```

```
-0.5
```

Entrée [48]:

```
# Clustring coefficient
nx.clustering(G)
```

Out[48]:

```
{'adam.bayer': 0,
 'andrea.ring': 0.08374384236453201,
 'alejandra.chavez': 0.26666666666666666,
 'amanda.rybarski': 0.07692307692307693,
 'announcements.enron': 0.04915514592933948,
 'bob.shults': 0.09941520467836257,
 'christina.sanchez': 0,
 'dana.jones': 0,
 'david.oxley': 0.07586206896551724,
 'edward.brady': 0.1,
 'frank.vickers': 0.15555555555555556,
 'h..cook': 0,
 'infrastructure.ubsw': 0.07509881422924901,
 'james.hungerford': 0.3333333333333333,
 'julie.clyatt': 0.08333333333333333,
 'kimberly.bates': 0.14285714285714285,
 'kulvinder.fowler': 0.1111111111111111,
```

Entrée [49]:

```
# Diameter
#if not connected, finding the largest component and calculating diameter on that component
print(nx.is_connected(G))
# Next, use nx.connected_components to get the list of components,
# then use the max() command to find the largest one:
components = nx.connected_components(G)
largest_component = max(components, key=len)
# Create a "subgraph" of just the largest component
# Then calculate the diameter of the subgraph, just like you did with density.
subgraph = G.subgraph(largest_component)
diameter = nx.diameter(subgraph)
print("Network diameter of largest component:", diameter)
```

```
True
Network diameter of largest component: 5
```

Entrée [50]:

```
# clustering coefficient
#Triadic closure supposes that if two people know the same person, they are likely to know
triadic_closure = nx.transitivity(G)
print("Triadic closure:", triadic_closure)
```

```
Triadic closure: 0.019101936900958467
```

Entrée [57]:

```
## Micro-level analysis

#create a dictionary of degrees
from operator import itemgetter
degree_dict = dict(G.degree(G.nodes()))
print(degree_dict)
#sort the dictionary of degrees
sorted_degree = sorted(degree_dict.items(), key=itemgetter(1), reverse=True)
print(sorted_degree)
#print the top 20 nodes
print("Top 20 nodes by degree:")
for d in sorted_degree[:20]:
    print(d)
```

```
{'adam.bayer': 2, 'andrea.ring': 29, 'alejandra.chavez': 10, 'amanda.rybar
ski': 13, 'announcements.enron': 63, 'bob.shults': 19, 'christina.sanche
z': 1, 'dana.jones': 1, 'david.oxley': 30, 'edward.brady': 5, 'frank.vicke
rs': 10, 'h..cook': 3, 'infrastructure.ubsw': 23, 'james.hungerford': 7,
'julie.clyatt': 24, 'kimberly.bates': 15, 'kulvinder.fowler': 10, 'liz.tay
lor': 22, 'melissa.videtto': 11, 'michele.winckowski': 1, 'neil.davies': 2
4, 'no.address': 79, 'resources.human': 37, 'scott.goodell': 4, 'scott.nea
l': 104, 'stephanie.miller': 6, 'stephanie.sever': 21, 'suzanne.calcagno':
7, 'victor.lamadrid': 9, '40ect': 2, 'andrew.lewis': 41, '40enron': 42, 'a
drianne.engler': 12, 'airam.arteaga': 5, 'alex.villarreal': 6, 'bart.bur
k': 2, 'cheryl.johnson': 9, 'd..hogan': 10, 'danielle.marcinkowski': 5, 'd
avid.forster': 25, 'denver.plachy': 3, 'di.mu': 1, 'enron.chairman': 5, 'e
nron.expertfinder': 6, 'georgeanne.hodges': 15, 'huy.dinh': 2, 'iris.mac
k': 15, 'jin.guo': 1, 'k..allen': 5, 'kirk.mcdaniel': 3, 'legalonline-comp
liance': 2, 'martin.cuilla': 20, 'monique.sanchez': 7, 'r..harrington': 1
4, 'robyn.zivic': 2, 'savita.puthigai': 22, 'sonya.johnson': 4, 'technolog
y.enron': 22, 'todd.peikert': 3, 'transportation.parking': 6, 'andy.zippe
r': 76, 'a..bibi': 3, 'a..shankman': 6, 'adam.johnson': 9, 'bob.lee': 3,
'carl.carter': 1, 'chris.bowling': 1, 'claudio.ribeiro': 1, 'coo.jeff': 8,
```

Entrée [55]:

```
#Eigenvector centrality cares if you are a hub, but it also cares how many hubs you are con
eigenvector_dict = nx.eigenvector_centrality(G)
pprint(eigenvector_dict)
```

```
{'2.ews': 0.0037602589701925533,
 '40ect': 0.0046949821896925805,
 '40ees': 0.0021090842790367987,
 '40enron': 0.1251763277257142,
 'a..allen': 0.0022862356081075103,
 'a..aune': 0.0011075133494264004,
 'a..bibi': 0.011643858409243112,
 'a..bowen': 0.0015490237304019608,
 'a..casas': 0.003951058362137417,
 'a..davis': 0.008300722692296616,
 'a..garcia': 0.008297873323351204,
 'a..gomez': 0.002726036817723818,
 'a..hope': 0.0025016787802791185,
 'a..howard': 0.0038162416093736025,
 'a..hudler': 0.010279050258408508,
 'a..hughes': 0.00851677527816802,
 'a..knudsen': 0.0025016787802791185,
 'a..lee': 0.0003659819148389361,
 'a..lindholm': 0.007113606741111981.
```

Entrée [59]:

```
#Betweenness centrality looks at all the shortest paths that pass through a particular node
betweenness_dict = nx.betweenness_centrality(G)
print(betweenness_dict)
#Sort Betweenness centrality
sorted_betweenness = sorted(betweenness_dict.items(), key=itemgetter(1), reverse=True)
print("Top 20 nodes by betweenness centrality:")
for b in sorted_betweenness[:20]:
    print(b)
```

```
{'adam.bayer': 4.6266012704420694e-07, 'andrea.ring': 0.00329516636314486
9, 'alejandra.chavez': 0.0002460745682028448, 'amanda.rybarski': 0.0002419
033057732572, 'announcements.enron': 0.0731742983449279, 'bob.shults': 0.0
013260693423665473, 'christina.sanchez': 0.0, 'dana.jones': 0.0, 'david.ox
ley': 0.009041480891334745, 'edward.brady': 2.4218283250866722e-05, 'fran
k.vickers': 0.0006185275748845689, 'h..cook': 5.720216890155504e-05, 'infr
astructure.ubsw': 0.0019627823602997372, 'james.hungerford': 4.18888041007
4748e-05, 'julie.clyatt': 0.0037076785508265795, 'kimberly.bates': 0.00082
65671067836297, 'kulvinder.fowler': 0.00015916645931605681, 'liz.taylor':
0.004055081400702491, 'melissa.videtto': 0.00034696398744020457, 'michele.
winckowski': 0.0, 'neil.davies': 0.003039165632892864, 'no.address': 0.129
1895042206206, 'resources.human': 0.01864940372350025, 'scott.goodell': 9.
707901925458061e-06, 'scott.neal': 0.022540417568967595, 'stephanie.mille
r': 0.00032910301455880915, 'stephanie.sever': 0.0012981631175088269, 'suz
anne.calcagno': 8.322416662861927e-05, 'victor.lamadrid': 7.13552155685814
6e-05, '40ect': 4.181996854191785e-06, 'andrew.lewis': 0.004453041068556337
, '40enron': 0.05053980925345601, 'adrianne.engler': 0.000202895524906430
04, 'airam.arteaga': 4.340752261627777e-05, 'alex.villarreal': 8.031258243
099883e-05, 'bart.burk': 7.4882572256312285e-06, 'cheryl.johnson': 0.00081
```

Entrée [60]:

```
#First get the top 20 nodes by betweenness as a list
top_betweenness = sorted_betweenness[:20]
#Then find and print their degree
for tb in top_betweenness: # Loop through top_betweenness
    degree = degree_dict[tb[0]] # Use degree_dict to access a node's degree
    print("Name:", tb[0], "| Betweenness Centrality:", tb[1], "| Degree:", degree)
```

```
Name: sherri.reinartz | Betweenness Centrality: 0.15975688326653045 | Degre
e: 281
Name: rosalee.fleming | Betweenness Centrality: 0.1550920331218245 | Degree:
304
Name: no.address | Betweenness Centrality: 0.1291895042206206 | Degree: 79
Name: sally.beck | Betweenness Centrality: 0.10074839481245033 | Degree: 228
Name: rick.buy | Betweenness Centrality: 0.07725843839060766 | Degree: 218
Name: gerald.nemec | Betweenness Centrality: 0.07661340499244368 | Degree: 2
04
Name: announcements.enron | Betweenness Centrality: 0.0731742983449279 | Deg
ree: 63
Name: vince.kaminski | Betweenness Centrality: 0.06999058225717317 | Degree:
132
Name: kaye.ellis | Betweenness Centrality: 0.06566506147467412 | Degree: 194
Name: rod.hayslett | Betweenness Centrality: 0.05332096197298508 | Degree: 1
09
Name: paul.y'barbo | Betweenness Centrality: 0.05276954358388109 | Degree: 1
09
Name: 40enron | Betweenness Centrality: 0.05053980925345601 | Degree: 42
Name: jeff.dasovich | Betweenness Centrality: 0.046022471537877585 | Degree:
124
Name: jenny.helton | Betweenness Centrality: 0.043151822766835754 | Degree:
170
Name: michelle.cash | Betweenness Centrality: 0.04178703501255959 | Degree:
118
Name: tana.jones | Betweenness Centrality: 0.04071620318116479 | Degree: 148
Name: mark.haedicke | Betweenness Centrality: 0.04034561158241739 | Degree:
148
Name: clint.dean | Betweenness Centrality: 0.034266412229062085 | Degree: 83
Name: laura.vuittonet | Betweenness Centrality: 0.03411483724783643 | Degre
e: 133
Name: angela.mcculloch | Betweenness Centrality: 0.033893338145284564 | Degr
ee: 110
```

Entrée [61]:

```
#clustering : measures the density of links in node i’s immediate neighborhood
clustering=nx.clustering(G)
print(clustering)
```

{'adam.bayer': 0, 'andrea.ring': 0.08374384236453201, 'alejandra.chavez':
0.26666666666666666, 'amanda.rybarski': 0.07692307692307693, 'announcement
s.enron': 0.04915514592933948, 'bob.shults': 0.09941520467836257, 'christi
na.sanchez': 0, 'dana.jones': 0, 'david.oxley': 0.07586206896551724, 'edwa
rd.brady': 0.1, 'frank.vickers': 0.15555555555555556, 'h..cook': 0, 'infra
structure.ubsw': 0.07509881422924901, 'james.hungerford': 0.33333333333333
33, 'julie.clyatt': 0.08333333333333333, 'kimberly.bates': 0.1428571428571
4285, 'kulvinder.fowler': 0.1111111111111111, 'liz.taylor': 0.077922077922
07792, 'melissa.videtto': 0.21818181818181817, 'michele.winckowski': 0, 'n
eil.davies': 0.09057971014492754, 'no.address': 0.05322947095098994, 'reso
urces.human': 0.05405405405405406, 'scott.goodell': 0.6666666666666666, 's
cott.neal': 0.04835698282300224, 'stephanie.miller': 0.3333333333333333,
'stephanie.sever': 0.0761904761904762, 'suzanne.calcagno': 0.3809523809523
8093, 'victor.lamadrid': 0.3333333333333333, '40ect': 0, 'andrew.lewis':
0.02926829268292683, '40enron': 0.0627177700348432, 'adrianne.engler': 0.1
9696969696969696, 'airam.arteaga': 0, 'alex.villarreal': 0, 'bart.burk':
0, 'cheryl.johnson': 0.05555555555555555, 'd..hogan': 0.06666666666666667,
'danielle.marcinkowski': 0.1, 'david.forster': 0.08666666666666667, 'denve
r.plachy': 0, 'di.mu': 0, 'enron.chairman': 0.3, 'enron.expertfinder': 0.1

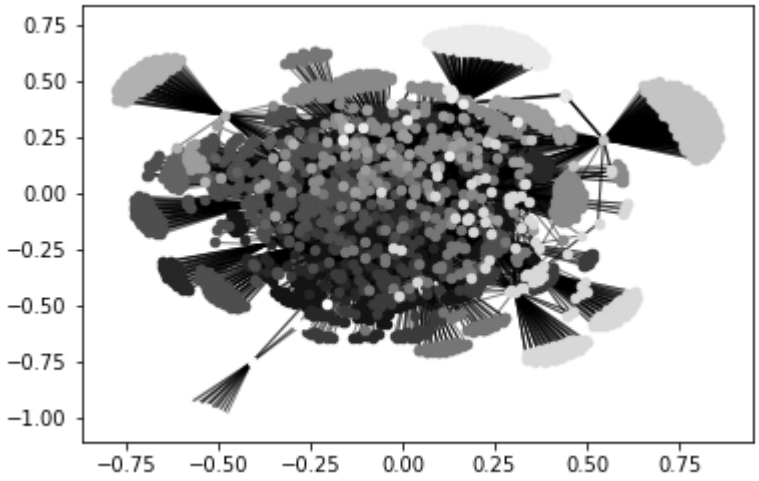
Entrée [63]:

```
## Meso-level analysis

#first compute the best partition
partition = community.best_partition(G)

#drawing
size = float(len(set(partition.values())))
pos = nx.spring_layout(G)
count = 0.
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 = 20,
                                node_color = str(count / size))


nx.draw_networkx_edges(G,pos, alpha=0.5)
plt.show()
```



Entrée [67]:

```
#List of nodes in periphery
p=nx.periphery(G)
pprint(p)
```

```
['brian.wood',
 'carol.carter',
 'chris.foster',
 'christopher.czuppon',
 'darren.cross',
 'dean.drozdiak',
 'dolores.muzzy',
 'eric.le',
 'gerry.hrap',
 'howard.sangwine',
 'hr.taylor',
 'jake.thomas',
 'jennifer.mcclain',
 'john.malowney',
 'kevin.heal',
 'kimberly.hardy',
 'morris.clark',
 'nicole.laporte',
 'rov.hartstein'.
```

Entrée [ ]: