Problem Set 4

```
import pandas as pd
import numpy as np
from scipy import stats
from scipy.stats import norm
from numpy.linalg import inv
import matplotlib.pyplot as plt
import os
from pathlib import Path
import re
import statistics as stat
import seaborn as sns
import random
import networkx as nx

os.chdir('e:/MIT4/statistics-Computation/pset4')
```

4.2 Investigating a time-varying criminal network

In this problem, you will study a time-varying criminal network that is repeatedly disturbed by police forces. The data for this problem can be found in CAVIAR.zip.

(a)

For each of the 11 phases, compute and list the: • degree, • betweenness centrality • eigenvector centrality of the actors under investigation.

degree

```
# the list of actors under investigation
under_investigation = [1,3,83,86,85,6,11,88,106,89,84,5,8,76,77,87,82,96,12,17,80,33,16]
# build the network
n = 110
phases = 11
graphs = {}

for p in range(phases):
    file_name = "data/phase" + str(p+1) + ".csv"
    data = pd.read_csv( file_name, header = 0, index_col= 0)
    actors = data.index.values.tolist()
    M = np.zeros((n+1, n+1))
    for i in actors:
        for j in actors:
            M[i,j] = data[str(j)][i]
    G = nx.DiGraph(M)
    G.remove_node(0)
    graphs[p+1] = G
```

```
# degree
in_degree_df = pd.DataFrame(0, index = under_investigation, columns = range(1, phases+1))

for p in range(1, phases+1):
    in_degrees = graphs[p].in_degree(nbunch = under_investigation)
    for i in under_investigation:
        in_degree_df.loc[i, p] = in_degrees[i]
in_degree_df

out_degree_df = pd.DataFrame(0, index = under_investigation, columns = range(1, phases+1))

for p in range(1, phases+1):
    out_degrees = graphs[p].out_degree(nbunch = under_investigation)
    for i in under_investigation:
        out_degree_df.loc[i, p] = out_degrees[i]
out_degree_df
```

in-degrees:

	1	2	3	4	5	6	7	8	9	10	11
1	5	8	10	11	11	17	15	10	5	8	6
3	2	1	6	4	3	9	2	8	5	1	0
83	0	2	3	4	2	2	3	1	1	1	0
86	0	0	3	2	1	0	0	1	0	1	1
85	2	0	4	5	2	3	3	2	3	2	3
6	3	0	3	1	2	0	1	0	1	0	0
11	0	2	1	1	0	0	1	1	1	0	1
88	4	2	0	2	1	0	2	0	0	0	1
106	0	0	0	2	0	0	0	0	0	0	0
89	2	2	1	2	3	0	0	0	1	0	0
84	0	0	3	1	0	0	0	1	0	1	0
5	1	1	2	1	2	1	1	0	0	0	0
8	1	2	2	2	1	4	1	3	2	1	0
76	0	2	1	2	1	6	3	2	3	3	3
77	0	0	0	0	0	2	2	1	0	0	0
87	0	0	0	0	0	0	1	4	5	3	3
82	0	0	0	0	1	1	0	2	4	5	3
96	0	0	0	0	0	0	0	0	3	2	1
12	0	0	2	1	5	7	5	8	5	3	9
17	0	0	0	0	0	1	0	0	1	1	2
80	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	1	1	1

out-degrees:

	1	2	3	4	5	6	7	8	9	10	11
1	11	17	26	21	18	13	18	19	9	13	6
3	2	2	6	6	4	9	9	9	11	1	1
83	2	0	7	6	1	1	1	0	0	2	1
86	0	1	4	3	1	0	0	1	0	0	0
85	3	2	0	4	2	3	2	3	2	2	2
6	1	1	2	1	1	1	1	1	0	0	0
11	0	0	2	1	1	2	2	2	0	0	1
88	4	2	2	1	1	0	0	0	1	0	0
106	0	0	0	0	0	0	0	0	0	0	0
89	2	2	1	6	0	0	0	0	0	0	0
84	0	0	2	2	1	2	0	0	0	1	2
5	0	0	0	1	0	3	0	0	0	0	0
8	0	2	0	2	1	1	0	0	2	2	0
76	0	0	2	1	2	4	5	5	3	1	7
77	0	0	0	0	0	2	2	0	0	0	0
87	0	0	0	0	0	2	3	9	7	11	4
82	0	0	0	0	1	1	0	0	5	3	4
96	0	0	0	0	0	0	0	0	2	1	2
12	0	1	0	1	5	10	1	4	7	6	9
17	0	0	0	0	1	1	1	1	1	0	1
80	0	0	0	0	0	0	0	1	0	0	0
33	0	0	0	0	0	0	0	1	0	0	0
16	0	0	0	0	0	0	1	1	1	1	1

Betweeness Centrality

```
between_df = pd.DataFrame(0, index = under_investigation, columns = range(1, phases+1))
for p in range(1, phases+1):
    b_cen = nx.betweenness_centrality(graphs[p])
    for i in under_investigation:
        between_df.loc[i, p] = b_cen[i]
# between_df
```

betweeness centrality:\n

	1	2	3	4	5	6	7	8	9	10	11
1	0.005989	0.014866	0.028174	0.036145	0.029392	0.024875	0.032195	0.041907	0.008339	0.028797	0.014229
3	0	0	0.004318	0.005437	0.007263	0.006909	0.002421	0.017513	0.028132	0	0
83	0	0	0.001798	0.005479	0.001529	0	0	0	0	0.001444	0
86	0	0	0.000849	0.002223	0	0	0	0.000127	0	0	0
85	0.000127	0	0	0.007277	0.003313	0.000127	0.002591	0	0.002931	0.002209	0.00034
6	0.000042	0	0.002421	0	0	0	0	0	0	0	0
11	0	0	0.000042	0	0	0	0.001444	0.002039	0	0	0
88	0.001317	0.001911	0	0	0	0	0	0	0	0	0
106	0	0	0	0	0	0	0	0	0	0	0
89	0.001529	0.001402	0	0.006966	0	0	0	0	0	0	0
84	0	0	0.000028	0.001034	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0
8	0	0.001444	0	0.001529	0	0	0	0	0.000297	0.001444	0
76	0	0	0	0	0	0.004049	0.007815	0.005253	0.008792	0.004587	0.00429
77	0	0	0	0	0	0	0	0	0	0	0
87	0	0	0	0	0	0	0	0.012714	0.013549	0.009854	0.001444
82	0	0	0	0	0	0	0	0	0.007914	0.006881	0.004162
96	0	0	0	0	0	0	0	0	0.000623	0	0.000595
12	0	0	0	0	0.008877	0.016876	0.00034	0.023785	0.016834	0.002124	0.015998
17	0	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0

Eigenvector Centrality

```
eigen_df = pd.DataFrame(0, index = under_investigation, columns = range(1, phases+1))
for p in range(1, phases+1):
    e_cen = nx.eigenvector_centrality(graphs[p], max_iter = 1000)
    for i in under_investigation:
        eigen_df.loc[i, p] = e_cen[i]
# eigen_df
eigen_l_df = pd.DataFrame(0, index = under_investigation, columns = range(1, phases+1))
reverse_g = {}
for p in range(1, phases+1):
    reverse_g[p] = graphs[p].reverse()
    e_lcen = nx.eigenvector_centrality(reverse_g[p], max_iter = 1000)
    for i in under_investigation:
        eigen_l_df.loc[i, p] = e_lcen[i]
eigen_l_df
```

Table of right eigenvector centrality:

	1	2	3	4	5	6	7	8	9	10	11
1	4.70E-01	4.77E-01	4.63E-01	4.42E-01	4.70E-01	5.38E-01	5.14E-01	4.52E-01	4.25E-01	5.06E-01	4.26E-01
3	3.35E-01	1.89E-01	2.96E-01	2.75E-01	3.06E-01	4.01E-01	1.84E-01	4.02E-01	3.22E-01	1.44E-01	3.53E-39
83	5.01E-10	1.89E-01	2.16E-01	2.37E-01	1.73E-01	1.99E-01	2.71E-01	1.15E-01	1.19E-01	1.44E-01	3.53E-39
86	5.01E-10	1.67E-08	2.52E-01	1.84E-01	5.70E-02	3.16E-11	2.39E-11	1.02E-01	7.71E-16	4.07E-02	2.23E-37
85	3.35E-01	1.67E-08	2.27E-01	3.79E-01	1.73E-01	1.99E-01	2.09E-01	2.17E-01	2.09E-01	1.84E-01	2.44E-01
6	2.89E-01	1.67E-08	1.76E-01	1.20E-01	1.55E-01	3.16E-11	1.44E-01	8.79E-12	1.19E-01	4.86E-17	3.53E-39
11	5.01E-10	1.63E-01	5.59E-02	1.20E-01	6.36E-10	3.16E-11	1.44E-01	1.15E-01	1.19E-01	4.86E-17	1.38E-01
88	4.96E-01	1.89E-01	3.06E-08	2.23E-01	5.70E-02	3.16E-11	1.10E-01	8.79E-12	7.71E-16	4.86E-17	1.05E-01
106	5.01E-10	1.67E-08	3.06E-08	1.84E-01	6.36E-10	3.16E-11	2.39E-11	8.79E-12	7.71E-16	4.86E-17	3.53E-39
89	1.85E-01	2.24E-01	1.20E-01	1.29E-01	2.05E-01	3.16E-11	2.39E-11	8.79E-12	9.02E-02	4.86E-17	3.53E-39
84	5.01E-10	1.67E-08	2.07E-01	6.42E-02	6.36E-10	3.16E-11	2.39E-11	1.15E-01	7.71E-16	1.44E-01	3.53E-39
5	1.63E-01	1.89E-01	1.20E-01	1.20E-01	2.05E-01	1.14E-01	1.44E-01	8.79E-12	7.71E-16	4.86E-17	3.53E-39
8	1.63E-01	2.63E-01	1.79E-01	1.94E-01	1.55E-01	2.66E-01	1.44E-01	2.73E-01	2.09E-01	1.44E-01	3.53E-39
76	5.01E-10	1.89E-01	1.20E-01	2.23E-01	1.55E-01	3.41E-01	2.62E-01	1.15E-01	2.11E-01	1.96E-01	3.25E-01
77	5.01E-10	1.67E-08	3.06E-08	1.86E-11	6.36E-10	1.99E-01	1.95E-01	1.15E-01	7.71E-16	4.86E-17	3.53E-39
87	5.01E-10	1.67E-08	3.06E-08	1.86E-11	6.36E-10	3.16E-11	1.44E-01	2.80E-01	4.04E-01	2.92E-01	3.33E-01
82	5.01E-10	1.67E-08	3.06E-08	1.86E-11	1.55E-01	1.14E-01	2.39E-11	2.17E-01	3.73E-01	3.81E-01	2.75E-01
96	5.01E-10	1.67E-08	3.06E-08	1.86E-11	6.36E-10	3.16E-11	2.39E-11	8.79E-12	2.74E-01	1.39E-01	8.92E-02
12	5.01E-10	1.67E-08	1.96E-01	7.46E-02	2.86E-01	2.81E-01	2.35E-06	6.36E-02	1.18E-01	2.84E-05	3.30E-01
17	5.01E-10	1.67E-08	3.06E-08	1.86E-11	6.36E-10	5.97E-02	2.39E-11	8.79E-12	3.30E-02	1.36E-05	1.73E-01
80	5.01E-10	1.67E-08	3.06E-08	1.86E-11	6.36E-10	3.16E-11	2.39E-11	8.79E-12	7.71E-16	4.86E-17	3.53E-39
33	5.01E-10	1.67E-08	3.06E-08	1.86E-11	6.36E-10	3.16E-11	2.39E-11	8.79E-12	7.71E-16	4.86E-17	3.53E-39
16	5.01E-10	1.67E-08	3.06E-08	1.86E-11	6.36E-10	3.16E-11	2.39E-11	8.79E-12	3.30E-02	1.36E-05	1.07E-01

Table of left eigenvector centrality:

	1	2	3	4	5	6	7	8	9	10	11
1	5.26E-01	6.59E-01	5.74E-01	6.41E-01	6.15E-01	5.30E-01	5.73E-01	5.82E-01	4.06E-01	6.06E-01	5.47E-01
3	3.63E-01	1.03E-01	2.90E-01	2.73E-01	3.34E-01	3.74E-01	3.89E-01	3.89E-01	5.74E-01	1.72E-01	2.89E-05
83	2.45E-01	9.37E-10	3.87E-01	2.22E-01	1.69E-05	1.13E-01	3.49E-06	1.66E-10	5.21E-12	1.72E-01	1.21E-37
86	9.88E-11	2.61E-01	3.24E-01	2.34E-01	1.69E-05	1.04E-12	4.26E-10	2.32E-09	5.21E-12	1.64E-16	1.93E-39
85	4.25E-01	1.03E-01	4.03E-09	2.90E-01	1.23E-01	1.88E-01	1.60E-01	2.47E-01	2.74E-01	2.21E-01	3.13E-01
6	1.80E-01	1.50E-08	2.24E-01	1.74E-01	2.02E-01	1.13E-01	1.60E-01	1.48E-01	5.21E-12	1.64E-16	1.93E-39
11	9.88E-11	9.37E-10	2.24E-01	1.74E-01	2.02E-01	1.37E-01	2.05E-01	9.90E-02	5.21E-12	1.64E-16	1.77E-01
88	5.19E-01	2.61E-01	5.79E-02	7.87E-02	4.05E-02	1.04E-12	4.26E-10	1.66E-10	7.68E-02	1.64E-16	1.93E-39
106	9.88E-11	9.37E-10	4.03E-09	6.51E-11	2.06E-09	1.04E-12	4.26E-10	1.66E-10	5.21E-12	1.64E-16	1.93E-39
89	2.08E-01	4.60E-05	4.84E-08	1.88E-01	2.06E-09	1.04E-12	4.26E-10	1.66E-10	5.21E-12	1.64E-16	1.93E-39
84	9.88E-11	9.37E-10	1.49E-01	2.34E-01	2.02E-01	1.53E-01	4.26E-10	1.66E-10	5.21E-12	6.27E-02	2.79E-01
5	9.88E-11	9.37E-10	4.03E-09	1.74E-01	2.06E-09	2.16E-01	4.26E-10	1.66E-10	5.21E-12	1.64E-16	1.93E-39
8	9.88E-11	2.61E-01	4.03E-09	1.89E-09	2.89E-08	1.13E-01	4.26E-10	1.66E-10	2.74E-01	1.72E-01	1.93E-39
76	9.88E-11	9.37E-10	1.49E-01	7.87E-02	2.02E-01	2.44E-01	2.52E-01	2.85E-01	2.65E-01	1.72E-01	4.17E-01
77	9.88E-11	9.37E-10	4.03E-09	6.51E-11	2.06E-09	1.92E-01	2.25E-01	1.66E-10	5.21E-12	1.64E-16	1.93E-39
87	9.88E-11	9.37E-10	4.03E-09	6.51E-11	2.06E-09	1.92E-01	2.69E-01	2.84E-01	3.72E-01	4.18E-01	4.27E-01
82	9.88E-11	9.37E-10	4.03E-09	6.51E-11	2.02E-01	1.13E-01	4.26E-10	1.66E-10	2.44E-01	2.11E-01	3.53E-01
96	9.88E-11	9.37E-10	4.03E-09	6.51E-11	2.06E-09	1.04E-12	4.26E-10	1.66E-10	2.08E-02	5.98E-02	1.14E-01
12	9.88E-11	2.61E-01	4.03E-09	7.39E-02	1.98E-01	3.15E-01	3.49E-06	1.01E-01	2.10E-01	4.93E-05	7.40E-05
17	9.88E-11	9.37E-10	4.03E-09	6.51E-11	6.51E-02	6.69E-02	3.49E-06	2.58E-02	5.88E-02	1.64E-16	2.89E-05
80	9.88E-11	9.37E-10	4.03E-09	6.51E-11	2.06E-09	1.04E-12	4.26E-10	2.58E-02	5.21E-12	1.64E-16	1.93E-39
33	9.88E-11	9.37E-10	4.03E-09	6.51E-11	2.06E-09	1.04E-12	4.26E-10	2.58E-02	5.21E-12	1.64E-16	1.93E-39
16	9.88E-11	9.37E-10	4.03E-09	6.51E-11	2.06E-09	1.04E-12	3.49E-06	2.58E-02	5.88E-02	2.35E-05	2.89E-05

(b)

Describe which actors are central and which actors are only peripheral. Explain and validate your reasoning. Feel free to compute other graph parameters, in addition to the centrality measures in (a), to aid you in validating your answer. Who seem to be the three principal traffickers?

```
# We sort the degree and centrality measures of (a):
in_degree_df.mean(1).sort_values(ascending = False)[:5]
out_degree_df.mean(1).sort_values(ascending = False)[:5]
between_df.mean(1).sort_values(ascending = False)[:5]
between_df.mean(1).sort_values(ascending = False)[-10:]
```

```
eigen_df.mean(1).sort_values(ascending = False)[:10]
eigen_l_df.mean(1).sort_values(ascending = False)[:10]
```

The betweeness centrality can grasp the "connection" of an actor in a criminal network. The top 5 central players are: n1, n12, n3, n87, n76. We noticed that They also have high degrees and eigenvector centrality values. The peripheral actors are those with low betweeness centrality values. The min value of betweeness centrality is 0, and the corresponding actors are: n16, n106, n33, n77, n17, n80, n5.

The eigenvector centrality can measure an actor's connection with other important actors. As the edge in our directed graph means a phone call, we think the left and right eigenvalue centrality could mean different types of "importance". But in our case, the top players usually have high score in both. They are n1, n3, n85, and n76.

However, all these measures cannot find the real central actor without further information about the behavioral and organizational pattern of this criminal network. It's possible that the bosses will call others to give orders but never receive calls, or only receive calls but issue orders in another way, or simply do not use telephone at all and do business through their agents. We can only know n1, n3, n12, n87, n76, n85 are important traffickers.

(c)

Are there other actors that play an important role but are not on the list of investigation? List them, and explain why they are important.

```
## repeat (a) and include all
# degree
in_degree_alldf = pd.DataFrame(0, index = range(1, n+1), columns = range(1, phases+1))
out_degree_alldf = pd.DataFrame(0, index = range(1, n+1), columns = range(1, phases+1))
between_alldf = pd.DataFrame(0, index = range(1, n+1), columns = range(1, phases+1))
eigen_alldf = pd.DataFrame(0, index = range(1, n+1), columns = range(1, phases+1))
eigen_l_alldf = pd.DataFrame(0, index = range(1, n+1), columns = range(1, phases+1))


for p in range(1, phases+1):
    in_degrees = graphs[p].in_degree()
    out_degrees = graphs[p].out_degree()
    b_cen = nx.betweenness_centrality(graphs[p])
    e_cen = nx.eigenvector_centrality(graphs[p], max_iter = 1000)
    e_lcen = nx.eigenvector_centrality(reverse_g[p], max_iter = 1000)

    for i in range(1, n+1):
        in_degree_alldf.loc[i, p] = in_degrees[i]
        out_degree_alldf.loc[i, p] = out_degrees[i]
        between_alldf.loc[i, p] = b_cen[i]
        eigen_alldf.loc[i, p] = e_cen[i]
        eigen_l_alldf.loc[i, p] = e_cen[i]

in_degree_alldf.mean(1).sort_values(ascending = False)[:5]
out_degree_alldf.mean(1).sort_values(ascending = False)[:5]
between_alldf.mean(1).sort_values(ascending = False)[:10]
between_df.mean(1).sort_values(ascending = False)[:10]
eigen_alldf.mean(1).sort_values(ascending = False)[:10]
eigen_l_alldf.mean(1).sort_values(ascending = False)[:10]
eigen_l_df.mean(1).sort_values(ascending = False)[:10]

np.setdiff1d(between_df.mean(1).sort_values(ascending = False)[:10].index.values.tolist(), between_alldf.mean(1).sort
np.setdiff1d(eigen_df.mean(1).sort_values(ascending = False)[:10].index.values.tolist(), eigen_alldf.mean(1).sort_val
np.setdiff1d(eigen_l_df.mean(1).sort_values(ascending = False)[:10].index.values.tolist(), eigen_l_alldf.mean(1).sort
```

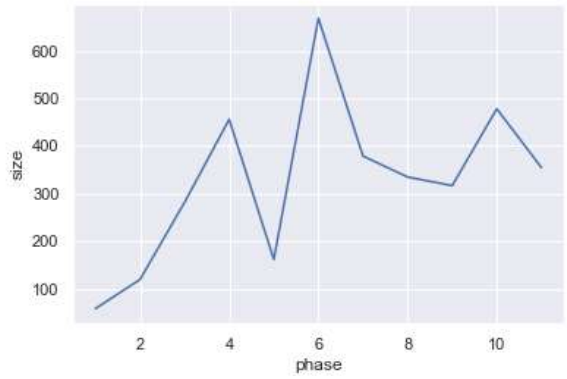
The n8, n83, n89 are in betweeness centrality top 10 list of all actors but not investigated. The n88 is in right eigenvector centrality top 10 list of all actors but not investigated. The n6, n11 are in left eigenvector centrality top 10 list of all actors but not investigated.

(d)

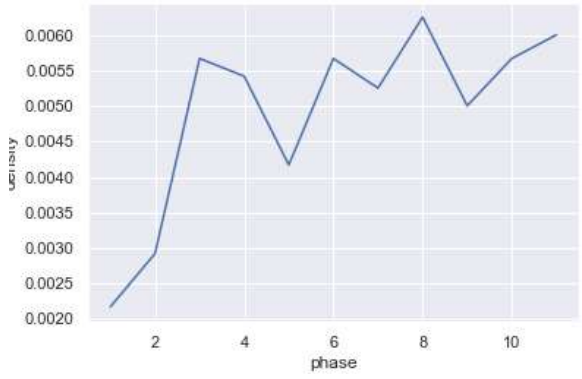
Describe the coarse pattern(s) you observe as the network evolves through the phases. Does the network evolution reect the background story? Explain.

```
# track the changes of the network
change_df = pd.DataFrame(0, index = range(1, phases+1), columns = ['phase', 'size','density'])
change_df['phase'] = range(1, phases+1)
for p in range(1, phases+1):

    change_df.loc[p, 'size'] =  graphs[p].size(weight = 'weight')
    change_df.loc[p, 'density'] = nx.density(graphs[p])

# change_df
import seaborn as sns; sns.set()
ax1 = sns.lineplot(x = "phase", y = "size", data = change_df)
ax2 = sns.lineplot(x = "phase", y = "density", data = change_df)
fig1 = ax1.get_figure()
fig1.savefig('figure/sizechange.png')
fig2 = ax2.get_figure()
fig2.savefig('figure/densechange.png')
```

Plot of network size change:



Plot of network density change:



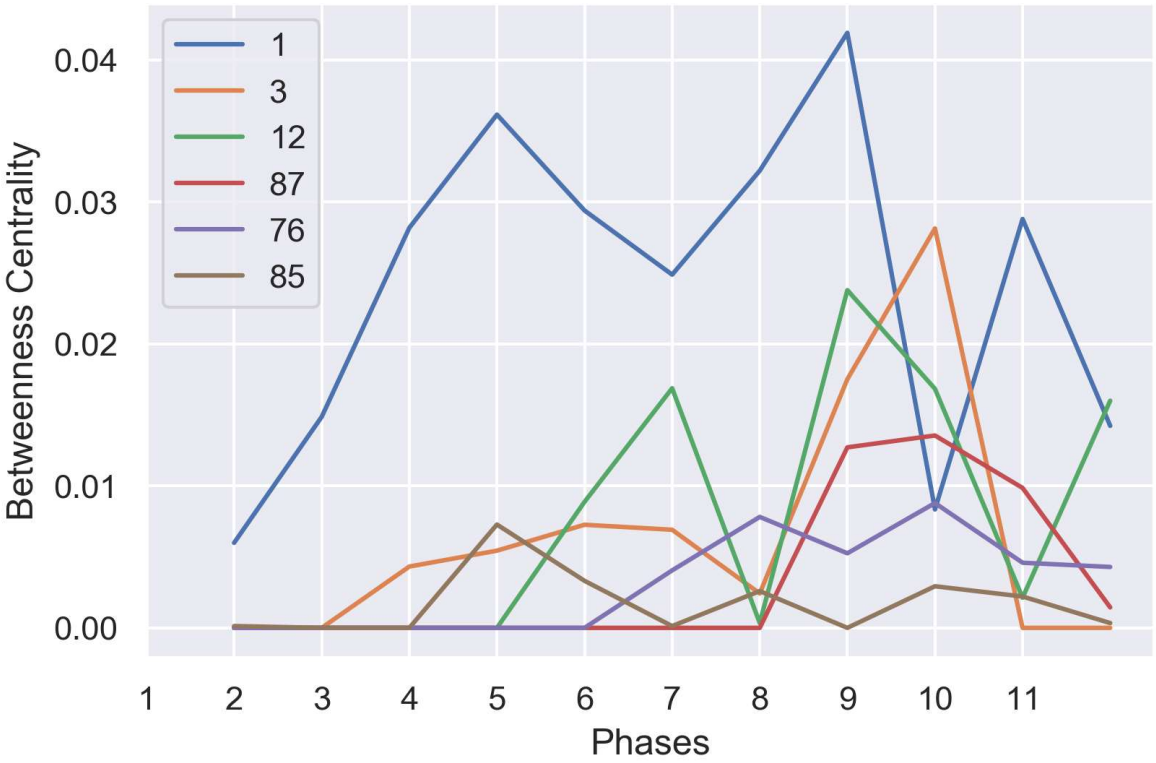
The network size and density decreased a lot between phase 4 and phase 5. It is caused by the seizure in phase 4. Phase 5 had no seizure, which caused the increase in network size and density in phase 6. The fluctuations after phase 6 were caused by seizures.

(e)

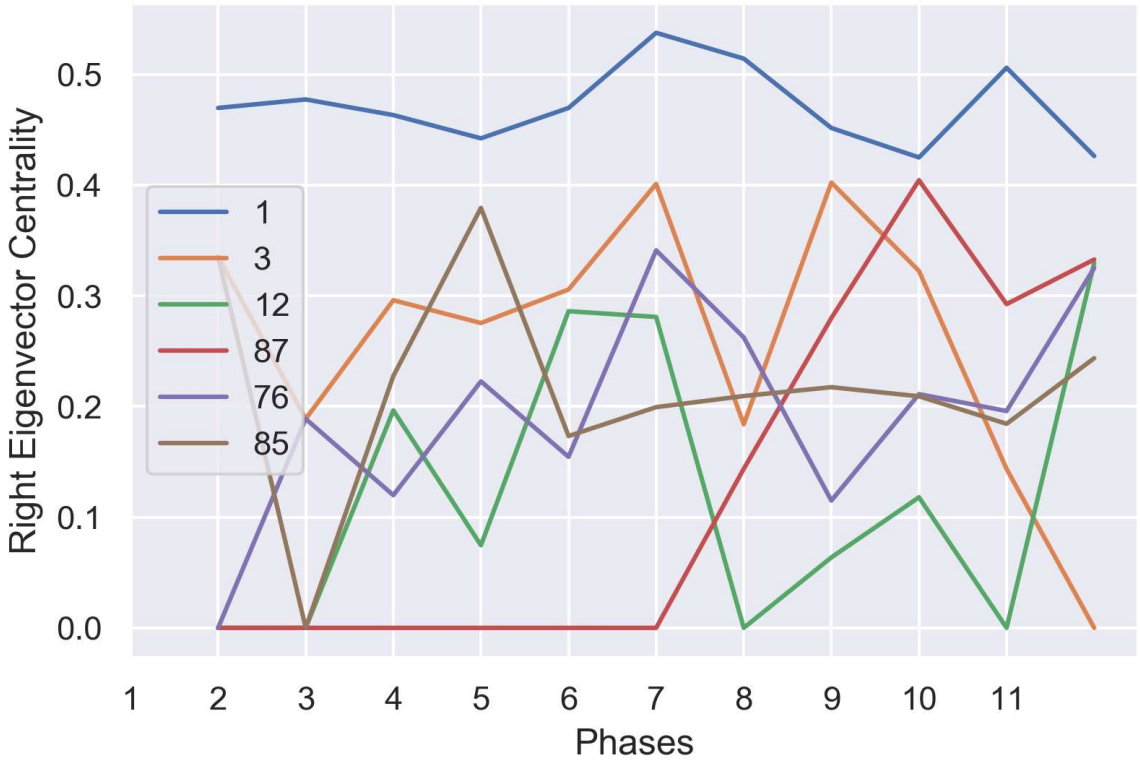
Describe and interpret the evolution of the role of the central actors found in (b). At which phases are they active? When do they withdraw? Find indices in the network evolution that reflect the description given to them.

```
central_actors = [1,3,12,87,76,85]

central_actors_bc = between_df.loc[central_actors]
central_actors_ec = eigen_df.loc[central_actors]
central_actors_elc = eigen_l_df.loc[central_actors]


# fig, ax = plt.subplots()
ax3 = central_actors_bc.T.plot()
ax3.set(xlabel = "Phases", ylabel = "Betweenness Centrality")
ax3.set_xticks(range(11))
ax3.set_xticklabels(central_actors_bc.columns)
plt.savefig('figure/central_actors_bc.png', dpi = 300)
plt.show()

ax4 = central_actors_ec.T.plot()
ax4.set(xlabel = "Phases", ylabel = "Right Eigenvector Centrality")
ax4.set_xticks(range(11))
ax4.set_xticklabels(central_actors_ec.columns)
plt.savefig('figure/central_actors_ec.png', dpi = 300)
plt.show()

ax5 = central_actors_elc.T.plot()
ax5.set(xlabel = "Phases", ylabel = "Left Eigenvector Centrality")
ax5.set_xticks(range(11))
ax5.set_xticklabels(central_actors_elc.columns)
plt.savefig('figure/central_actors_elc.png', dpi = 300)
plt.show()
```

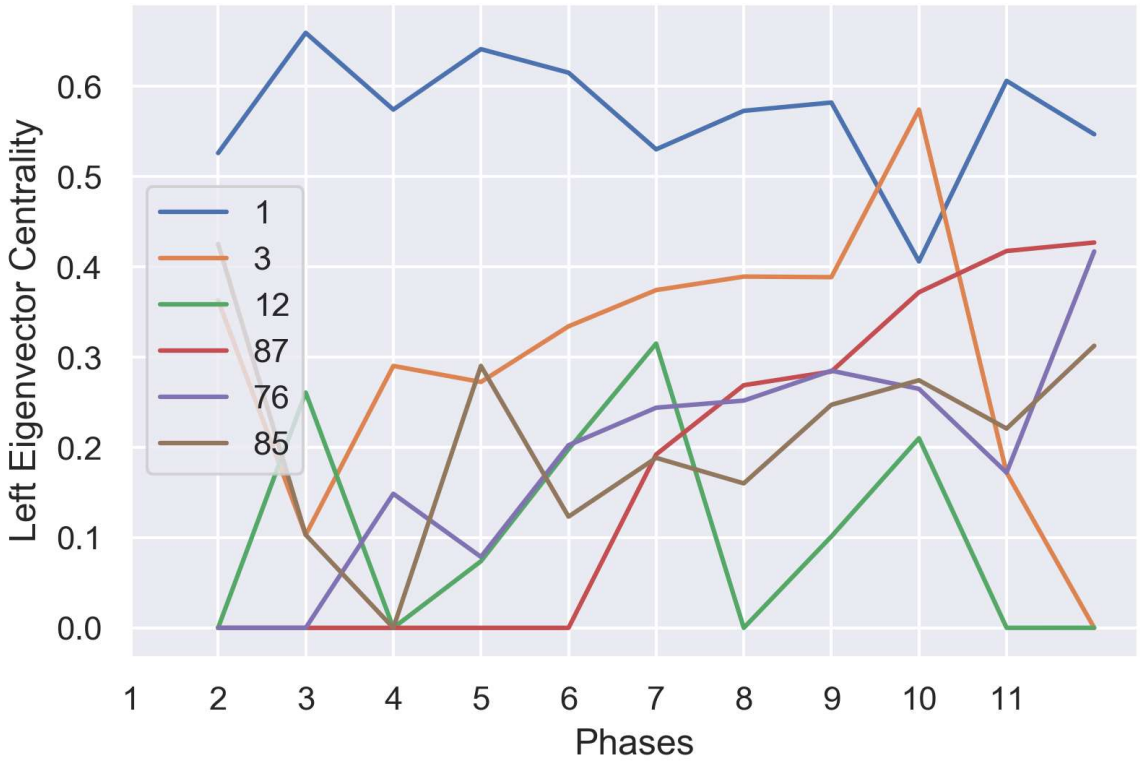
Plot of betweeness centrality change of selected central actors:



Plot of right eigenvector centrality change of selected central actors:



Plot of left eigenvector centrality change of selected central actors:



From the plot, we can see that n1 was in central position throughout the 11 phases, although we observed a drop in betweeness centrality in phase 9. n3 and n12's centrality dropped in phase 7 but increased after phase 7. n76 was steadly becoming more central, especially since phase 6. n87 started to become central since phase 8. n85 had a drop in phase 3 but generally highly connected with important actors.

(f)

Examine the frequency and the directions of the communications of (n1) as the network evolves. Any contrast or pattern(s) you observe? Describe, explain and interpret.

```
n1_df = pd.DataFrame(0, index = range(1, phases+1), columns = ['in degree', 'out degree','degree'])
n1_df.loc[1,'out degree']

for p in range(1, phases+1):
    n1_df.loc[p,'degree'] = graphs[p].degree(1)
    n1_df.loc[p, 'in degree'] = in_degree_df.loc[1,p]
    n1_df.loc[p,'out degree'] = out_degree_df.loc[1,p]
# n1_df

ax6 = n1_df.plot()
ax6.set(xlabel = "Phases", ylabel = "Degrees of n1")
ax6.set_xticks(range(1,12))
ax6.set_xticklabels(n1_df.index)
plt.savefig('n1d.png', dpi = 300)
plt.show()
```

We plot the in and out degree changes of n1:

Generally, phone calls from n1 (out degree) were more than calls to n1 (in degree).In phase 6, n1 received more calls than making calls to others. His communication frequency dropped a lot in phase 9 corresponding to 2 seizures.

(g)

Would you consider that the particular strategy adopted by the police had an impact on the criminal network throughout the different phases of the investigation? What kind of impact? Explain.

```
ax1 = sns.lineplot(x = "phase", y = "size", data = change_df)
ax2 = sns.lineplot(x = "phase", y = "density", data = change_df)
```

From the lineplot of network size and edge density, we found that the network is highly sensitive to seizures at first, but become less and less affected. Also, by investigating the betweeness/eigenvector centrality of the central actors, we think that the position of central actors may shift to resist the threats to the network. Therefore, the police should keep monitoring the communications of the network, make one or two seizures to disturb their organization and leadership, and pay attention to the actors with increasing centrality.

4.3 Co-offending Network

The data for this problem set consists of individuals who were arrested in Quebec between 2003 and 2010. Some of the individuals have always acted solo, and have been arrested alone throughout their 'career'. Others co-offended with other individuals, and have been arrested in groups. The goal of this problem set is to construct and analyze the co-oﬀender network. The nodes in the network are the oﬀenders, and two offenders share a (possibly weighted) edge whenever they are arrested for the same crime event.

build the whole co-offender network. Discard the isolated nodes, thus every node will have degree>=2. Given the size of the network, be careful regarding computational and memory constraints. Be sure to use sparse representations of the data whenever possible.

```
cooffend_df = pd.read_csv('data/Cooffending.csv').to_sparse()
cooffend_df.shape
cooffend_df.head()
```

```
# find cooffending cases
import collections

case_l = cooffend_df['SeqE'].tolist()

coofending_case_l = []
for item, count in collections.Counter(case_l).items():
    if count > 1:
        coofending_case_l.append(item)

len(coofending_case_l)
len(set(case_l))
# there were 1164836 unique cases, in which 84239 cases were co-offending cases
# remove individual cases
net_df0 = cooffend_df[cooffend_df['SeqE'].isin(coofending_case_l)].sort_values(by = ['SeqE']).to_sparse()

# the whole network took forever to build, we use a reduced network instead.
net_df = net_df0.iloc[1:1000].to_sparse()

# build the network
G = nx.Graph()

case_num = 0
offender_l = []

for i in range(len(net_df)):
    # extract one observation of offender and case
    seq = net_df['SeqE'][i]
    offender = net_df['NoUnique'][i]
    # add the offender into graph
    G.add_node(offender, Naissance = net_df['Naissance'][i], Sex = net_df['SEXE'][i])
    # as the data we use here is sorted by case number, we only need to go case by case
```

```
    if seq == case_num:
        # if the offender is still in current case
        for x in offender_l:

            # two possible cases: this offender co-offend other cases that have beed recorded in our graph with the s
            if G.has_edge(offender, x):
                G[offender][x]['weight'] += 1
            else:
                G.add_edge(offender, x, weight = 1)
        # add this offender in the case offender list
        offender_l.append(offender)

    # if we go to another case, reset case number and offerders list
    else:
        case_num = seq
        offender_l = [offender]
```

(e)

How many nodes does the network have? How many solo offenders are there in the data set? How many (unweighted) edges does the graph contain?

```
num_node = nx.number_of_nodes(G)
num_node
# to calculate the solo offenders, we use the number of unique offenders in the dataset minus the number of unique of
num_solo = len(set(cooffend_df['NoUnique'])) - len(set(net_df0['NoUnique']))
num_solo

unweighted_size = G.size()
unweighted_size
```

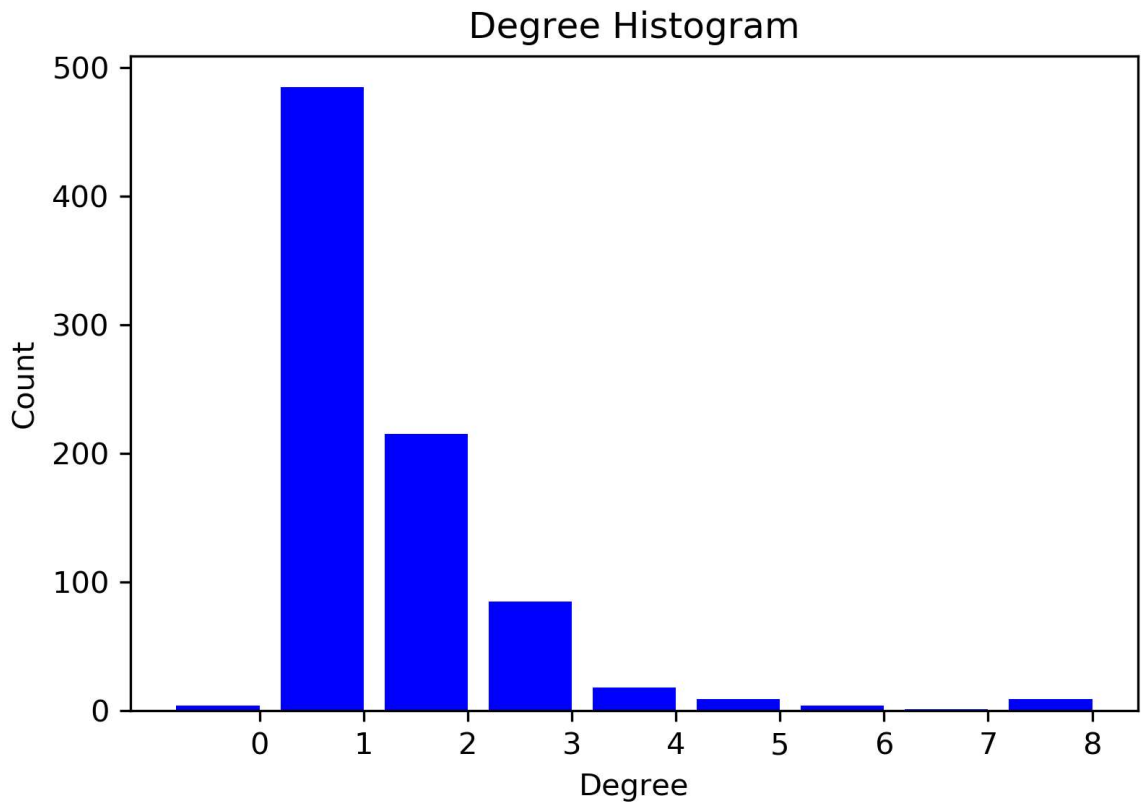
In our reduced network, we have 830 nodes, and the unweighted network size is 695; There are 418281 solo offenders in the dataset

(f)

Plot the degree distribution (or an approximation of it if needed) of the network.

```
coof_degree = G.degree()

def PlotDegree(G, filename):
    degree_sequence = sorted([d for n, d in G.degree()], reverse=True)  # degree sequence
    degreeCount = collections.Counter(degree_sequence)
    deg, cnt = zip(*degreeCount.items())
    fig, ax = plt.subplots()
    plt.bar(deg, cnt, width=0.80, color='b')
    plt.title("Degree Histogram")
    plt.ylabel("Count")
    plt.xlabel("Degree")
    ax.set_xticks([d + 0.4 for d in deg])
    ax.set_xticklabels(deg)
    plt.savefig(filename, dpi = 300)

PlotDegree(G, 'figure/degree_histogram.png')
```

Degree Histogram of our reduced network:



(g)

How many connected components does the network have?

```
num_compo = nx.number_connected_components(G)
num_compo
```

There are 336 components in our reduced network

(h)

We will now isolate the largest connected component and focus on it. This brings us down to a more manageable size. How many nodes does the largest connected component have?

```
max_compo = max(nx.connected_components(G), key=len)
max_compo
G_max_compo = G.subgraph(list(max_compo))
num_node_max_compo = nx.number_of_nodes(G_max_compo)
num_node_max_compo
```

Our largest connected component has 9 nodes

(i)

Compute the degree of the nodes, and plot the degree distribution (or an approximation of it if needed) for the largest connected component. Comment on the shape of the distribution.

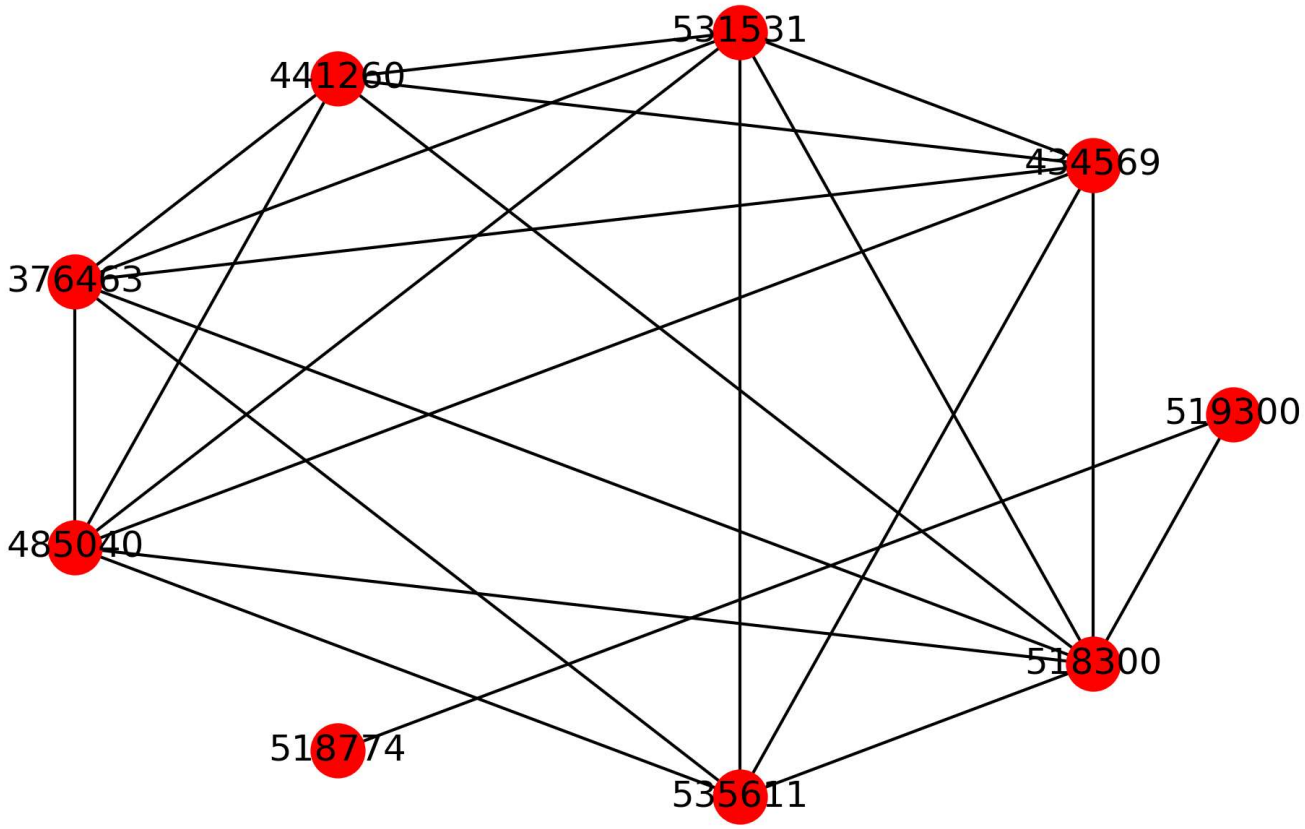
```
PlotDegree(G_max_compo,'figure/max_compo_degree_histogram.png' )
```

(j)

Describe the general shape of the largest connected component. Use the degree distribution from above, and compute statistics of the network to obtain an overview of its characteristics. You may want to consider the edge density, clustering, diameter, etc. Comment on the results.

```
nx.draw_shell(G_max_compo, with_labels=True)
plt.savefig('figure/max_compo.png',dpi = 300)
den = nx.density(G_max_compo)
den
dia = nx.diameter(G_max_compo)
dia
```

Network plot of the max component in reduced network:



This component's density is 0.61, and the diameter is 3.

(k)

Thisfinal section involves some free form investigation. The following parts are optional for undergraduates. How many crime events are executed only by young offenders?

```
young_offenders_df = cooffend_df[cooffend_df['Adultes'] == 0]
young_offenders_df.head(20)
len(set(young_offenders_df['SeqE']))
```

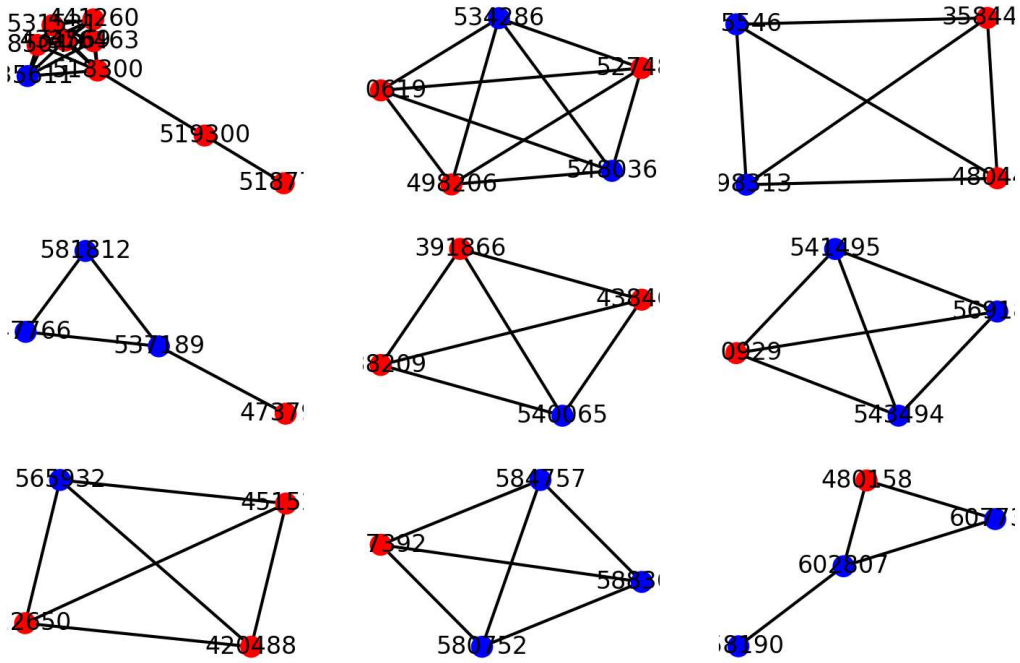
There are no crime events only by young offenders.

(l)

Investigate the relationship between young offenders and adult offenders. Study the structure of the crimes that include both young and adult offenders. Discuss any patterns you observe.

```
# crime of both young and adult offenders:
co_l = net_df[(net_df['Adultes']!=0) & (net_df['Jeunes']!=0)]
co_l
# the data is from 2003 to 2010 so we set 1985 as threashold
test = co_l[(co_l['Naissance'] < 1985)]
test.shape

compos = sorted(nx.connected_components(G), key=len, reverse=True)


i = 0
num_plots = 9
for compo in compos:
    if i>= num_plots:
        break
    c = G.subgraph(list(compo))

    code = int('33'+str(i+1))
    young = 0
    adult = 0
    color_map = []
    # check if it has both young and agult
    for node, data in c.nodes(data=True):
        if data['Naissance'] < 1985:
            color_map.append('red')
            young += 1
        else:
            color_map.append('blue')
            adult += 1

    if young * adult == 0:
        continue

    plt.subplot(code)
    i += 1
    nx.draw(c, with_labels=True,node_size=40,font_size=8,node_color = color_map)

plt.savefig('figure/compos.png', dpi = 300)
```

9 components containing both young and adult offenders in reduced network:



We can oberve 2 main patterns: 1. one adult and multiple young offenders; 2. one young offender and multiple adults. It could be possible that some closely linked groups have few adult leaders and many young "soldiers", or groups of adult offenders will bring one or two young guy with them.

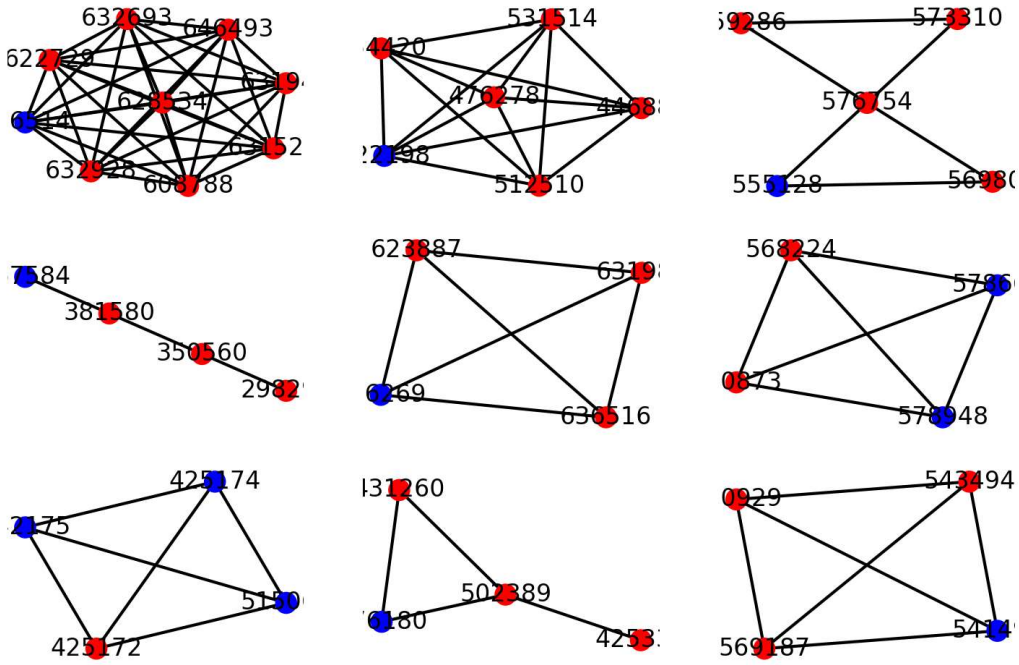
(m)

Ask your own question, build new separate networks if needed, and get as much insight as you like. Feel free to focus on either the whole network, or the largest connected component.

```
i = 0
num_plots = 9
for compo in compos:
    if i>= num_plots:
        break
    c = G.subgraph(list(compo))

    code = int('33'+str(i+1))
    male = 0
    female = 0
    color_map = []
    # check if it has both young and agult
    for node, data in c.nodes(data=True):
        if data['Sex'] == 'M':
            color_map.append('red')
            male += 1
        else:
            color_map.append('blue')
            female += 1

    if male * female == 0:
        continue

    plt.subplot(code)
    i += 1
```

```
        nx.draw(c, with_labels=True,node_size=40,font_size=8,node_color = color_map)

    plt.savefig('figure/compos_sex.png', dpi = 300)
```

9 components containing both male and female offenders in reduced network:



We ask the question about the sex of the offender. We found that major components of the cooffending network contain much less female offenders than male offenders.