In [1]:

```python
# Import packages
import numpy as np
import scipy as sp
import networkx as nx
import pandas as pd
import pickle

# Plotting
import matplotlib.pyplot as plt
import seaborn as sns

# HodgeRank
from Hodge import *
```

Set the random seed so results can replicate. I used random.org to generate the seed.

In [2]:

```python
seed = 833913377
```

Load data

In [3]:

```python
# Load networks
infile = open("../data/data_dict.pickle",'rb')
data_dict = pickle.load(infile)
infile.close()
```

In [4]:

```python
# Load self-assessment data
df_SA = pd.read_csv("../data/self_assessed_wealth.csv")
```

In [5]:

```python
# Poverty rankings determined at community meetings
df_CBT = pd.read_csv("../data/community_meeting.csv")
```

Friend-Based Ranking vs Community-Based Targeting

Steps:

1. Extract only the nine nodes networks, plot those to check for variation
2. Apply HodgeRank
    - Need three colum numpy array.
3. Compare to consumption figures

In [6]:

```python
def extract_dataframe_of_comparisons(hamlet, data_dict):
    '''
    hamlet: str, name of hamlet, eg. hamlet_1
    data_dict: dictionary of extracted data

    return: Pandas dataframe
    '''
    chosen = data_dict[hamlet]["chosen"]

    edge_list = []
    for i in chosen:
        for j in chosen:
            if i<j:
                edge_list.append((i,j))

    df = pd.DataFrame(index=pd.MultiIndex.from_tuples(edge_list, names=["i","j"]),
                      columns=chosen)

    for ind in chosen:
        for edge in edge_list:
            rank_i = data_dict[hamlet]["guess_rank"][ind][edge[0]]
            rank_j = data_dict[hamlet]["guess_rank"][ind][edge[1]]
            if (rank_i!=999) & (rank_j!=999):
                df.loc[edge][ind] = (rank_j - rank_i)//abs(rank_j - rank_i)

    id_to_hodge = {}
    hodge_to_id = {}

    x = 0
```

```python
    for ind in chosen:
        id_to_hodge[ind] = x
        hodge_to_id[x] = ind
        x+=1

    df.reset_index(inplace=True)
    df["i_hodge"] = df.i.map(id_to_hodge)
    df["j_hodge"] = df.j.map(id_to_hodge)

    return (df, id_to_hodge, hodge_to_id, chosen)
```

In [7]:

```python
def use_HodgeRank(i):
    '''
    i: integer for hamlet number


    '''
    np.random.seed(seed=seed)

    hamlet = "hamlet_" + str(i)
    (df, id_to_hodge, hodge_to_id, chosen) = extract_dataframe_of_comparisons(hamlet, data_dict)

    hodge_scores = pd.DataFrame()
    for ind in chosen:
        included = list(set(chosen)-set([ind]))
        df["mean_over_ind"] = df[included].mean(axis=1)
        # df["sum_over_ind"] = df[included].mean(axis=1, min_count=1)
        R = df[df.mean_over_ind.notnull()][["i_hodge","j_hodge"]].values
        Y = df[df.mean_over_ind.notnull()].mean_over_ind.values
        W = np.ones(len(Y))
        (s, I, H) = doHodge(R, W, Y)


        hs_ind = pd.DataFrame()
        hs_ind["hodge_id"] = np.array(range(0,len(s)))
        hs_ind["hodge_s"] =  s
        hs_ind["id"] = hs_ind.hodge_id.map(hodge_to_id) # Need to be careful this mapping is done accurately
        hs_ind["excluded"] = ind

        (a, b, c) = getConsistencyRatios(Y, I, H, W)
        hs_ind["local_inconsistency"] = b
        hs_ind["global_inconsistency"] = c

        hodge_scores = pd.concat([hodge_scores,hs_ind])

    consumption = pd.DataFrame.from_dict(data_dict[hamlet]["consumption"], orient='index', columns=["consumpt
    consumption.reset_index(inplace=True)
    consumption.rename(columns={'index':'id'}, inplace=True)

    result = pd.merge(consumption,hodge_scores,on='id')

    result = pd.merge(result,df_SA[df_SA.hamlet==i][['id','self_assessed_wealth']],on='id')

    try:
        result = pd.merge(result,df_CBT[df_CBT.hamlet==i],on='id')
    except:
        result['ranking_meeting'] = np.nan
        result['quota_final'] = np.nan
        result['nhhrank'] = np.nan

    return result
```

Complete analysis

In [8]:

```python
df = pd.DataFrame()
for h in range(1,640):
    try:
        df_h = use_HodgeRank(h)
        df_h = df_h[df_h.id==df_h.excluded].copy()
        df = pd.concat([df,df_h])
    except:
        continue
;
```

```
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
```

```
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
C:\Users\molc0001\repos\fbr-in-practice\notebooks\Hodge.py:127: RuntimeWarning: invalid value encountered in
double_scalars
  a = (normD0s/normY)**2
C:\Users\molc0001\repos\fbr-in-practice\notebooks\Hodge.py:128: RuntimeWarning: invalid value encountered in
double_scalars
  b = (normI/normY)**2
C:\Users\molc0001\repos\fbr-in-practice\notebooks\Hodge.py:129: RuntimeWarning: invalid value encountered in
double_scalars
  c = (normH/normY)**2
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
The exact solution is  x = 0
```

Out[8]: ''

A few of the hamlets did not work:

In [9]:
```
df_CBT.hamlet.nunique() - df.hamlet.nunique()
```

Out[9]: 8

Clean up rounding errors for some of measures.

In [10]:
```
df[['hodge_s',
    'local_inconsistency',
    'global_inconsistency']] = df[['hodge_s','local_inconsistency','global_inconsistency']].round(10)
```

In [11]:
```
df['ranking_meeting_norm'] = df.ranking_meeting/df.nhhrank
```

Descriptive statistics

In [12]:
```
df[['consumption','hodge_s','local_inconsistency','hamlet',
    'maintreatment','elite_meeting','ranking_meeting_norm']].describe()
```

Out[12]:

	consumption	hodge_s	local_inconsistency	hamlet	elite_meeting	ranking_meeting_norm
count	3522.000000	3522.000000	3522.000000	3522.000000	3513.000000	3522.000000
mean	531.447516	0.003976	0.154635	313.071834	0.495588	0.533532
std	538.087668	0.475024	0.065444	183.708111	0.500052	0.283264
min	55.447090	-0.888889	0.000000	1.000000	0.000000	0.012195
25%	267.381975	-0.375000	0.124069	157.000000	0.000000	0.295905
50%	398.583750	0.000000	0.146472	312.000000	0.000000	0.544949
75%	608.593525	0.384921	0.168747	473.000000	1.000000	0.780488
max	12460.380000	0.888889	0.807143	638.000000	1.000000	1.000000

In [13]:
```
df[['consumption','self_assessed_wealth','hodge_s','ranking_meeting_norm']].corr(method="spearman")
```

Out[13]:

	consumption	self_assessed_wealth	hodge_s	ranking_meeting_norm
consumption	1.000000	0.340417	0.332634	0.343914
self_assessed_wealth	0.340417	1.000000	0.469688	0.454537
hodge_s	0.332634	0.469688	1.000000	0.739825
ranking_meeting_norm	0.343914	0.454537	0.739825	1.000000

In [14]:

```
df[df.elite_meeting==0][['consumption','self_assessed_wealth','hodge_s','ranking_meeting_norm']].corr(method=
```

Out[14]:

	consumption	self_assessed_wealth	hodge_s	ranking_meeting_norm
consumption	1.000000	0.302098	0.331063	0.338186
self_assessed_wealth	0.302098	1.000000	0.478671	0.458613
hodge_s	0.331063	0.478671	1.000000	0.743119
ranking_meeting_norm	0.338186	0.458613	0.743119	1.000000

In [15]:

```
df[df.elite_meeting==1][['consumption','self_assessed_wealth','hodge_s','ranking_meeting_norm']].corr(method=
```

Out[15]:

	consumption	self_assessed_wealth	hodge_s	ranking_meeting_norm
consumption	1.000000	0.378553	0.331728	0.347391
self_assessed_wealth	0.378553	1.000000	0.461793	0.450564
hodge_s	0.331728	0.461793	1.000000	0.734936
ranking_meeting_norm	0.347391	0.450564	0.734936	1.000000

Targeting

In [16]:

```
df[df.ranking_meeting>df.quota_final].consumption.describe()
```

Out[16]:

```
count     2438.000000
mean       601.488724
std        610.583536
min         65.706350
25%        296.706850
50%        440.261900
75%        690.730700
max      12460.380000
Name: consumption, dtype: float64
```

In [17]:

```
df[df.ranking_meeting<=df.quota_final].consumption.describe()
```

Out[17]:

```
count    1084.000000
mean      373.919412
std       257.950556
min        55.447090
25%       217.145575
50%       310.749900
75%       460.392775
max      3457.083000
Name: consumption, dtype: float64
```

In [18]:

```
cutoff = -0.274
```

The exact hodge scores depend on the random seed in the least squares solver.

In [19]:

```
df[df.hodge_s<cutoff].consumption.describe()
```

Out[19]:

```
count    1084.000000
mean      389.022715
std       300.335349
min        55.447090
25%       226.134625
50%       319.865100
75%       467.895525
max      4087.845000
Name: consumption, dtype: float64
```

In [20]:
```python
df[df.hodge_s>=cutoff].consumption.describe()
```

Out[20]:
```
count     2438.000000
mean       594.773392
std        604.324086
min         96.809520
25%        294.732400
50%        438.015900
75%        678.798875
max      12460.380000
Name: consumption, dtype: float64
```

In [21]:
```python
df[["hodge_s","ranking_meeting","consumption"]].describe()
```

Out[21]:

	hodge_s	ranking_meeting	consumption
count	3522.000000	3522.000000	3522.000000
mean	0.003976	27.907155	531.447516
std	0.475024	21.517776	538.087668
min	-0.888889	1.000000	55.447090
25%	-0.375000	12.000000	267.381975
50%	0.000000	23.000000	398.583750
75%	0.384921	38.000000	608.593525
max	0.888889	169.000000	12460.380000

And now self assessed

In [22]:
```python
df[df.ranking_meeting>df.quota_final].self_assessed_wealth.describe()
```

Out[22]:
```
count    2424.000000
mean        2.945132
std         1.031830
min         1.000000
25%         2.000000
50%         3.000000
75%         4.000000
max         6.000000
Name: self_assessed_wealth, dtype: float64
```

In [23]:
```python
df[df.ranking_meeting<=df.quota_final].self_assessed_wealth.describe()
```

Out[23]:
```
count    1083.000000
mean        2.163435
std         0.990748
min         1.000000
25%         1.000000
50%         2.000000
75%         3.000000
max         6.000000
Name: self_assessed_wealth, dtype: float64
```

In [24]:
```python
df[df.hodge_s<cutoff].self_assessed_wealth.describe()
```

Out[24]:
```
count    1082.000000
mean        2.108133
std         0.987596
min         1.000000
25%         1.000000
50%         2.000000
75%         3.000000
max         6.000000
Name: self_assessed_wealth, dtype: float64
```

In [25]:
```python
df[df.hodge_s>cutoff].self_assessed_wealth.describe()
```

Out[25]:
```
count    2425.000000
mean        2.969485
std         1.013470
min         1.000000
25%         2.000000
50%         3.000000
```

```
75%          4.000000
max          6.000000
Name: self_assessed_wealth, dtype: float64
```

Create variables for targeting

In [26]:
```
df["HodgeCBT_targets"] = np.where(df.hodge_s<cutoff,"Included","Excluded")
```

In [27]:
```
df["tradCBT_targets"] = np.where(df.ranking_meeting<=df.quota_final,"Included","Excluded")
```

In [28]:
```
pd.crosstab(df.HodgeCBT_targets,df.tradCBT_targets)
```

Out[28]:

tradCBT_targets HodgeCBT_targets	Excluded	Included
Excluded	2016	422
Included	422	662

HodgeCBT scores by hamlet

Even though the HodgeRank algorithm sets the mean of the scores to zero, we exclude the reports of i when determining the score of i. This mean each observation is based on slightly different ranking data and the mean scores do not always exactly equal zero.

In [29]:
```
df.groupby('hamlet').hodge_s.describe().round(4).sample(10)
```

Out[29]:

hamlet	count	mean	std	min	25%	50%	75%	max
19	9.0	0.0000	0.5029	-0.6726	-0.2679	0.0000	0.3929	0.8393
457	8.0	0.0157	0.4598	-0.8444	-0.1212	-0.0317	0.3397	0.5661
540	9.0	-0.0000	0.5754	-0.8889	-0.3492	0.2222	0.3492	0.8571
42	9.0	-0.0000	0.5233	-0.8254	-0.2857	0.0317	0.4762	0.5397
337	9.0	0.0000	0.5519	-0.8571	-0.2857	-0.0635	0.3810	0.8889
353	9.0	0.0000	0.5931	-0.8571	-0.5079	-0.0317	0.5079	0.8889
58	8.0	0.0000	0.4391	-0.6122	-0.3367	0.0000	0.2245	0.7347
167	8.0	-0.0040	0.5979	-0.8148	-0.3757	-0.0476	0.3810	0.7937
594	9.0	-0.0000	0.5501	-0.8889	-0.2222	0.0000	0.4127	0.7302
71	8.0	0.0000	0.5500	-0.8750	-0.3438	0.1875	0.2656	0.8125

Export dataframe

In [30]:
```
df.to_csv("../data/analysis.csv", index=False)
```

Hamlet level

In [31]:
```
df_ham = pd.DataFrame()
```

In [32]:
```
df_ham["count_hh"] = df.groupby('hamlet').id.count()
```

In [33]:
```
df_ham["corr_SAW_hodge"] = df.groupby('hamlet').apply(lambda df: df['self_assessed_wealth'].corr(df['hodge_s'
                                                                                                     method='spea
```

```
C:\Users\molc0001\AppData\Local\Continuum\anaconda3\envs\fbr-in-practice\lib\site-packages\scipy\stats\stats.
py:4196: SpearmanRConstantInputWarning: An input array is constant; the correlation coefficent is not define
d.
  warnings.warn(SpearmanRConstantInputWarning())
```

In [34]:
```
df_ham["corr_SAW_meet"] =df.groupby('hamlet').apply(lambda df: df['self_assessed_wealth'].corr(df['ranking_me
                                                                                       method='spearm
```

In [35]:
```
df_ham["corr_consum_hodge"] = df.groupby('hamlet').apply(lambda df: df['consumption'].corr(df['hodge_s'],
                                                                                         method='pear
```

In [36]:
```
df_ham["corr_consum_meet"] =df.groupby('hamlet').apply(lambda df: df['consumption'].corr(df['ranking_meeting_
                                                                                       method='pearso
```

In [37]:
```
df_ham["consum_sd"] =df.groupby('hamlet').consumption.std()
```

Cycle ratio

In [38]:
```
df_ham["local_incon"]  = df.groupby('hamlet').local_inconsistency.mean()
```

In [39]:
```
df_ham["global_incon"]  = df.groupby('hamlet').global_inconsistency.mean().round(3)
```

Note that in complete ranking graphs, the measure of global inconsistencies is zero by definition. This is because the algorithm first measures cycles of length 3 and then considers any remaining cycles.

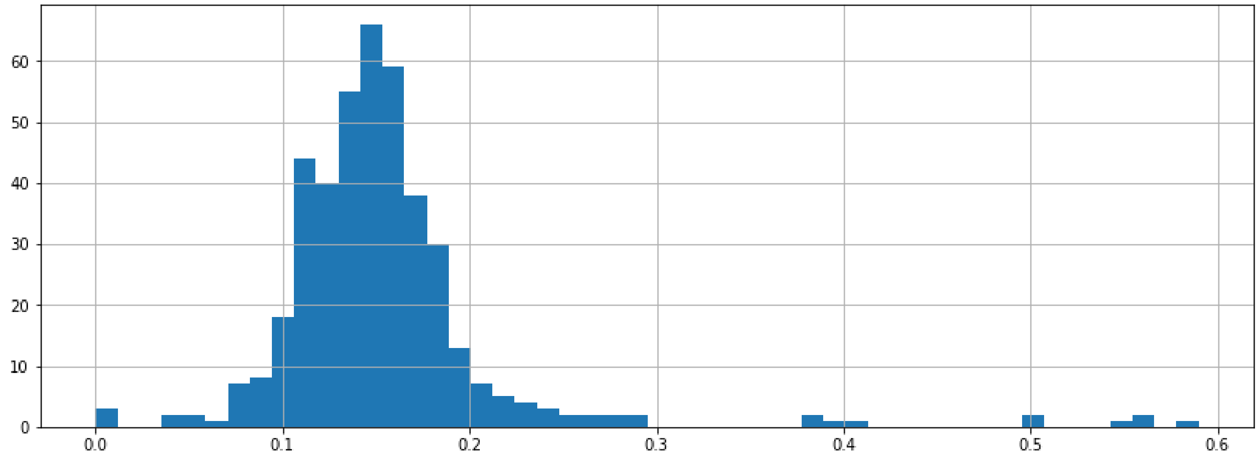
In [40]:
```
df_ham["cycle_ratio"] = df_ham["local_incon"] + df_ham["global_incon"]
```

In [41]:
```
df_ham[["corr_consum_hodge","corr_SAW_hodge","cycle_ratio","consum_sd"]].corr()
```

Out[41]:

	corr_consum_hodge	corr_SAW_hodge	cycle_ratio	consum_sd
corr_consum_hodge	1.000000	0.269436	-0.051486	0.097690
corr_SAW_hodge	0.269436	1.000000	-0.089129	0.035973
cycle_ratio	-0.051486	-0.089129	1.000000	-0.002050
consum_sd	0.097690	0.035973	-0.002050	1.000000

In [42]:
```
plt.figure(figsize=(14,5))
df_ham.cycle_ratio.hist(bins=50) ;
```



Notice the outliers with high cycle ratios. Inspect these hamlets.

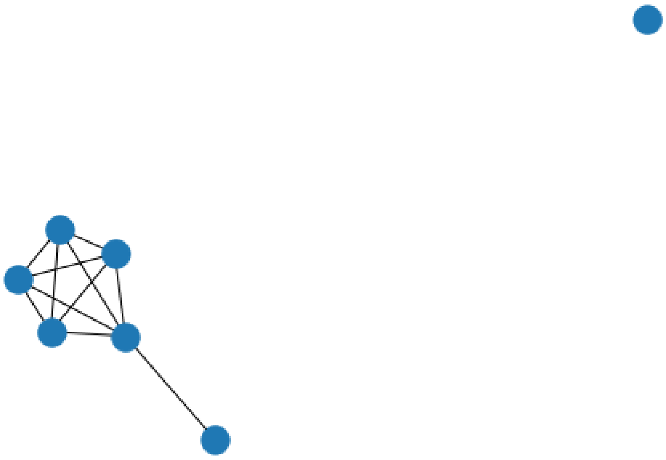
In [43]:
```
list(df_ham[df_ham.local_incon>0.5].index.values)
```

Out[43]: [18, 486, 487, 488, 622]

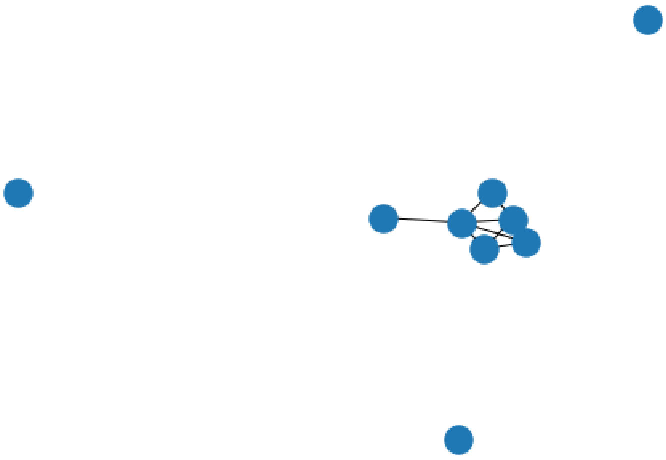
In [44]:
```
for num in list(df_ham[df_ham.local_incon>0.5].index.values):
        try:
            key = "hamlet_" + str(num)
            graph = data_dict[key]["graph"]
```

```
            chosen = data_dict[key]["chosen"]
            print("---------------------------------------------------------------------------")
            print(key)
            print("---------------------------------------------------------------------------")

            nx.draw(graph.subgraph(chosen))

            plt.show()
        except:
            print(key)
```

```
---------------------------------------------------------------------------
hamlet_18
---------------------------------------------------------------------------
```



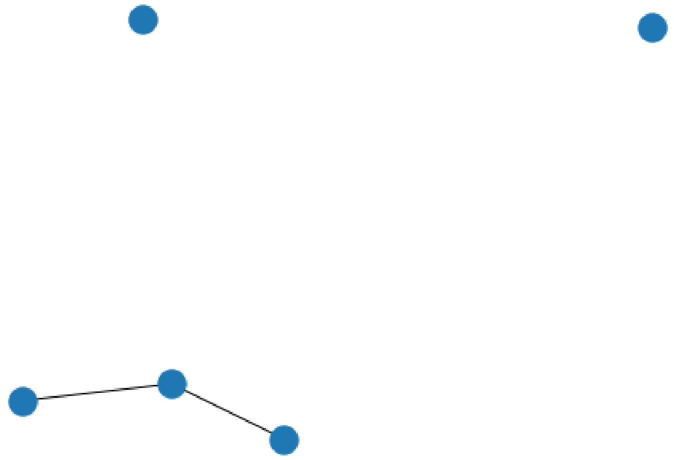
```
---------------------------------------------------------------------------
hamlet_486
---------------------------------------------------------------------------
```



```
---------------------------------------------------------------------------
hamlet_487
---------------------------------------------------------------------------
```

```
---------------------------------------------------------------------------
hamlet_488
---------------------------------------------------------------------------
```

```
---------------------------------------------------------------------------
hamlet_622
---------------------------------------------------------------------------
```



Is the connectivity of the outlier networks lower than the other networks?

In [45]:
```
def extract_network_stats(key_num):
    '''
    Extract network statistics from the subgraphs of 9 households who participate
    in the individualised wealth ranking.
    '''
    key = "hamlet_" + str(key_num)
    result = {}
    g = data_dict[key]["graph"]
    chosen = data_dict[key]["chosen"]
    sg = g.subgraph(chosen)
    result["density"] = nx.density(sg)
    return result
```

In [46]:
```
network_stats = {}
for num in list(df_ham.index.values):
    network_stats[num] = extract_network_stats(num)
```

In [47]:
```
df_ham["density"] = pd.DataFrame.from_dict(network_stats, orient='index')
```

In [48]:
```
df_ham[['cycle_ratio','density']].corr()
```

Out[48]:

	cycle_ratio	density
cycle_ratio	1.000000	-0.215492
density	-0.215492	1.000000

In [49]:
```
df_ham[df_ham.cycle_ratio>0.3].density.describe()
```

```
count    10.000000
```

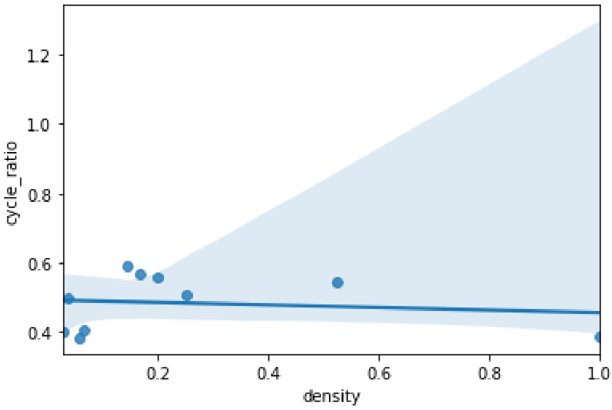
Out[49]:
```
mean      0.246905
std       0.302769
min       0.027778
25%       0.058333
50%       0.154762
75%       0.237500
max       1.000000
Name: density, dtype: float64
```

In [50]:
```
df_ham[df_ham.cycle_ratio<0.3].density.describe()
```

Out[50]:
```
count    413.000000
mean       0.695042
std        0.299372
min        0.027778
25%        0.472222
50%        0.777778
75%        1.000000
max        1.000000
Name: density, dtype: float64
```

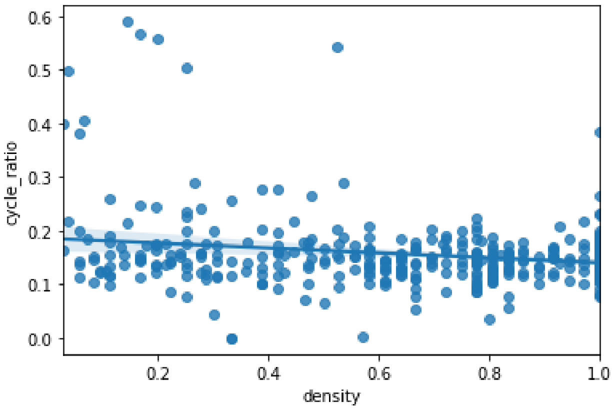
In [51]:
```
sns.regplot(x='density',y='cycle_ratio',data=df_ham[df_ham.cycle_ratio>0.3])
```

Out[51]: <AxesSubplot:xlabel='density', ylabel='cycle_ratio'>



In [52]:
```
sns.regplot(x='density',y='cycle_ratio',data=df_ham)
```

Out[52]: <AxesSubplot:xlabel='density', ylabel='cycle_ratio'>



Export data

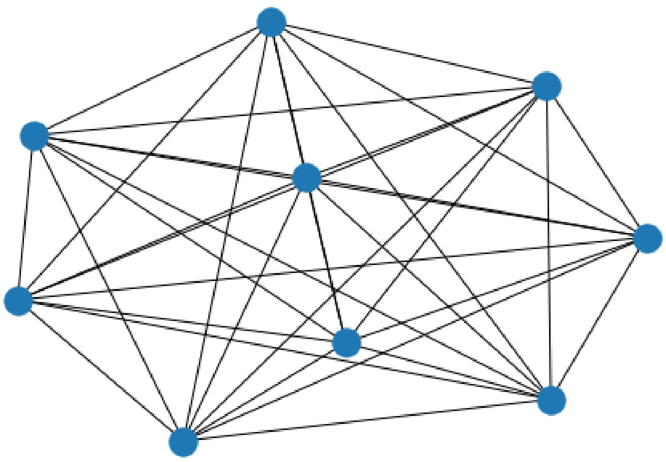
In [53]:
```
df_ham.to_csv("../data/analysis_hamlet_level.csv", index=False)
```

Graph networks

In [54]:
```
x = 0
for key in data_dict.keys():
    if x <= 5:
        try:
            graph = data_dict[key]["graph"]
```

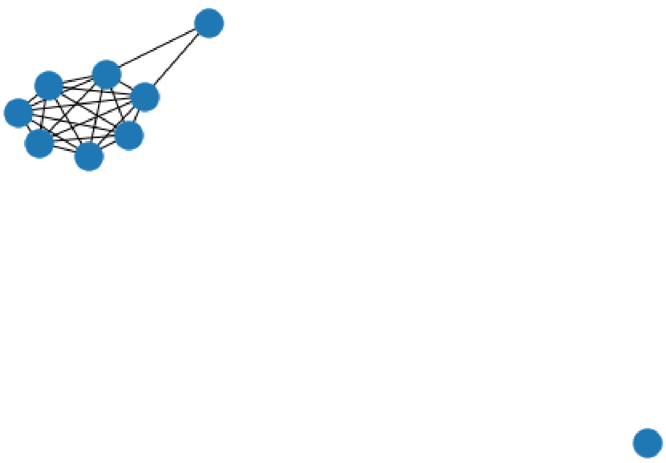
```
            chosen = data_dict[key]["chosen"]
            print("----------------------------------------------------------------------------")
            print(key)
            print("----------------------------------------------------------------------------")

            nx.draw(graph.subgraph(chosen))

            plt.show()
        except:
            print(key)
            continue
        x+= 1
```

```
----------------------------------------------------------------------------
hamlet_1
----------------------------------------------------------------------------
```



```
----------------------------------------------------------------------------
hamlet_2
----------------------------------------------------------------------------
```

```
----------------------------------------------------------------------------
hamlet_3
----------------------------------------------------------------------------
```



```
----------------------------------------------------------------------------
hamlet_4
```

----

hamlet_5

----

hamlet_6