Network Analysis

August 2, 2019

0.0.1 Network Analysis

A simple (scalable) network analysis with a focus on the weights between nodes. It makes use of the package *networkx*. Throughout the analysis, I will use the analogy of *shops* and the *owners* of the shops, which can also be another shop, meaning there can be an ultimate owner who sits at the top of what is essentially a tree. Example:

```
In [1]: import pandas as pd
        data = pd.DataFrame({'shop': ['S1', 'S2', 'S3', 'S1', 'S1', 'S4', 'S2', 'S3'],
                             'owner': ['O1', 'O2', 'S2', 'O3', 'S4', 'O4', 'O5', 'O6']})
        data.head(10)
```

Out[1]:

	shop	owner
0	S1	O1
1	S2	O2
2	S3	S2
3	S1	O3
4	S1	S4
5	S4	O4
6	S2	O5
7	S3	O6

In this example shops 2 and 4 have some ownership over shops 3 and 4 respectively. As a first step, I can use networkx to create a graph off the relationships, and find the ultimate owners of the shops. First, define a function for creating the graph:

```
In [2]: import networkx as nx
        def all_descendants_nx():
            DiG = nx.from_pandas_edgelist(data,'owner','shop',
                                          create_using=nx.DiGraph())
            return pd.DataFrame.from_records([(n1,n2) for n1 in DiG.nodes()
                for n2 in nx.ancestors(DiG, n1)], columns=['shop','ult_owners'])
```

Now I simply call the function which will create a new data frame with all the final owners of the shops.

```
In [3]: desc_df = all_descendants_nx()
        desc_df = desc_df.loc[desc_df.ult_owners.str.startswith("O")]
        desc_df.head(10)
```

```
Out[3]:    shop ult_owners
        0    S1         O3
        1    S1         O1
        3    S1         O4
        4    S2         O5
        5    S2         O2
        6    S3         O5
        7    S3         O2
        9    S3         O6
        10   S4         O4
```

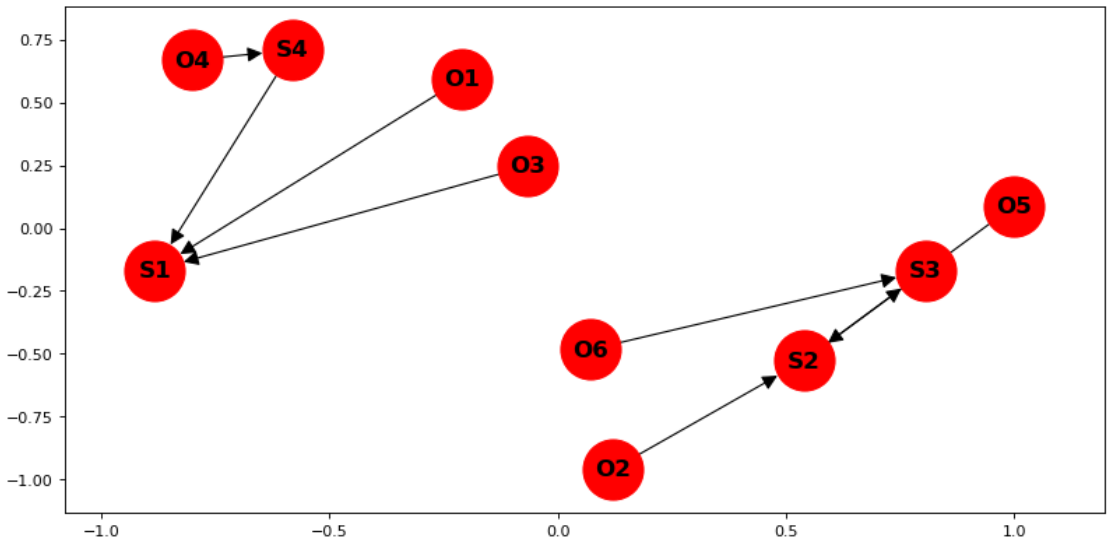
Finally, I can group by the shops to put all the owners together, and also get the links between them.

```
In [4]: desc_df['Links'] = desc_df.shop.astype('str'
                            ) + ' - '+ desc_df.ult_owners.astype('str')
        desc_df = desc_df.groupby('shop').agg(lambda x:
                                              sorted(x.tolist())).reset_index()
        desc_df.head()
```

```
Out[4]:   shop    ult_owners                          Links
        0   S1  [O1, O3, O4]  [S1 - O1, S1 - O3, S1 - O4]
        1   S2      [O2, O5]           [S2 - O2, S2 - O5]
        2   S3  [O2, O5, O6]  [S3 - O2, S3 - O5, S3 - O6]
        3   S4          [O4]                    [S4 - O4]
```

In fact, if creating the graph means I can plot the result:

```
In [5]: %matplotlib inline
        from matplotlib.pyplot import figure
        figure(num=None, figsize=(12, 6), dpi=80, facecolor='w', edgecolor='k')
        import numpy as np
        np.random.seed(1)
        G = nx.from_pandas_edgelist(data,'owner','shop',
                                    create_using=nx.DiGraph())
        pos=nx.spring_layout(G, k = 0.5, iterations = 20)
        node_labels = {node:node for node in G.nodes()}
        nx.draw_networkx(G, pos, labels = node_labels, arrowstyle = '-|>',
                     arrowsize = 20,  font_size = 15, font_weight = 'bold', node_size = 1500)
```



Graphing in this way makes it easy to see who has ownership of each shop. NB, I can check which shops also have some ownership over other shops like this:

```
In [6]: print(set(data.shop) & set(data.owner))
```

```
{'S2', 'S4'}
```

As a next step, I want to introduce percentage shares to the data frame. This example is slightly simplified but with weights added. In this simple example the weights all add to 1.

```
In [7]: data = pd.DataFrame({'shop': ['S1', 'S1', 'S1', 'S2', 'S2', 'S3', 'S3', 'S3'],
                             'owner': ['O1', 'O2', 'S2', 'S3', 'O3', 'O4', 'O5', 'O6'],
                             'share': [0.2,   0.2,  0.6,  0.5,  0.5,  0.1,  0.1,  0.8]})
        data.head(10)
```

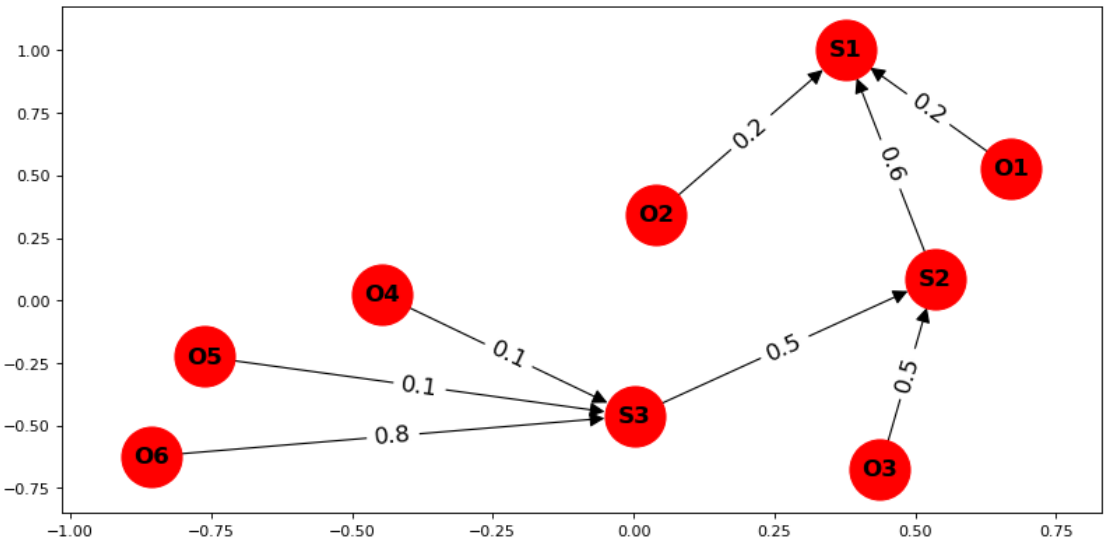
```
Out[7]:   shop owner  share
        0   S1    O1    0.2
        1   S1    O2    0.2
        2   S1    S2    0.6
        3   S2    S3    0.5
        4   S2    O3    0.5
        5   S3    O4    0.1
        6   S3    O5    0.1
        7   S3    O6    0.8
```

In this example all the shops and owners are connected as part of the same graph. I can draw the graph and add the weights to it:

```
In [8]: G = nx.from_pandas_edgelist(data,'owner', 'shop', edge_attr = ('share'),
                                    create_using=nx.DiGraph())
```

```
figure(num=None, figsize=(12, 6), dpi=80, facecolor='w', edgecolor='k')
np.random.seed(4)
pos=nx.spring_layout(G, k = 0.5, iterations = 20)
node_labels = {node:node for node in G.nodes()}
edge_labs = nx.get_edge_attributes(G,'share')
nx.draw_networkx(G, pos, labels = node_labels,
                 arrowstyle = '-|>', edge_labels = edge_labs,
             arrowsize = 20,  font_size = 15,
                 font_weight = 'bold', node_size = 1500)
nx.draw_networkx_edge_labels(G, pos, edge_labels = edge_labs,
                             font_size = 15)
```

```
Out[8]: {('O1', 'S1'): Text(0.523397,0.76389,'0.2'),
         ('O2', 'S1'): Text(0.208536,0.671963,'0.2'),
         ('S2', 'S1'): Text(0.456213,0.543579,'0.6'),
         ('S3', 'S2'): Text(0.268799,-0.186819,'0.5'),
         ('O3', 'S2'): Text(0.486679,-0.293083,'0.5'),
         ('O4', 'S3'): Text(-0.221858,-0.219201,'0.1'),
         ('O5', 'S3'): Text(-0.379704,-0.34279,'0.1'),
         ('O6', 'S3'): Text(-0.426816,-0.541575,'0.8')}
```



The next step is to calcuate the percentage share owned by each owner. For the shops owned by other shops, I need to multiply through the percentages:

```
In [9]: owners = set(data['owner'])
        shops  = set(data['shop'])
        summary = {}
        for owner in owners:
            for shop in shops:
                paths = list(nx.all_simple_paths(G, owner, shop))
```

```
                if len(paths):
                    for path in paths:
                        for start, end in zip(path[:-1], path[1:]):
                            summary[(shop, owner)] = summary.get((shop,owner),
                                                        1) * G[start][end]['share']

        summary = pd.DataFrame.from_dict(summary, orient = 'index',
                                         columns = 'share'.split())
        print(summary)
```

```
           share
(S1, O2)    0.20
(S1, S2)    0.60
(S3, O5)    0.10
(S2, O5)    0.05
(S1, O5)    0.03
(S1, O1)    0.20
(S2, S3)    0.50
(S1, S3)    0.30
(S3, O6)    0.80
(S2, O6)    0.40
(S1, O6)    0.24
(S2, O3)    0.50
(S1, O3)    0.30
(S3, O4)    0.10
(S2, O4)    0.05
(S1, O4)    0.03
```

This has created a long data frame giving each combination of owners and shares. I want to get the final ownership for each, and their weights:

```
In [10]: # For viewing the whole column
         pd.set_option('max_colwidth', -1)

In [11]: # Turn index to column
         summary = summary.reset_index()

         # Turn into 2 separate stripped columns
         summary["index"] = summary["index"].astype(str)
         summary['shop'], summary['owner'] = summary['index'
                                             ].str.split(' ', 1).str
         summary["shop"] = summary.shop.str.replace('\W', '')
         summary["owner"] = summary.owner.str.replace('\W', '')

         # Tunr into percentage and add to owners
         summary["share"] = summary["owner"] + ": " + (
                                summary["share"]*100).map(str) + "%"
```

```
# Keep only final owners
summary = summary[summary.owner.str.startswith('O')]

# Final aggregation
summary = summary.sort_values(by = ["shop", "share"])
summary = pd.DataFrame(summary.groupby('shop')['share'].apply(list))
summary = summary.reset_index()
summary.head()
```

Out[11]:

	shop	share
0	S1	[O1: 20.0%, O2: 20.0%, O3: 30.0%, O4: 3.0%, O5: 3.0%, O6: 24.0%]
1	S2	[O3: 50.0%, O4: 5.0%, O5: 5.0%, O6: 40.0%]
2	S3	[O4: 10.0%, O5: 10.0%, O6: 80.0%]

All 6 owners have some share over S1, and they all neatly add to 100%. Next, I want to introduce a situation where there is a loop across the network. I'll create a simple new data frame for this.

```
In [12]: data = pd.DataFrame({'shop': ['S1', 'S1', 'S2', 'S2', 'S3'],
                              'owner': ['S2', 'S3', 'O1', 'O2', 'O1'],
                              'share': [0.8,   0.2,  0.5,  0.5, 1.0]})
         data.head()
```

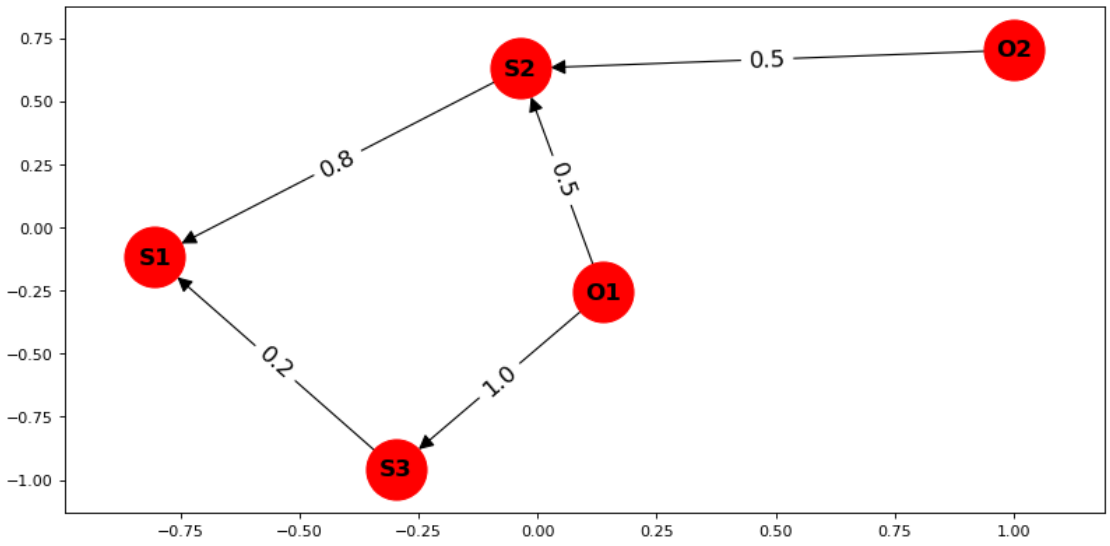
Out[12]:

	shop	owner	share
0	S1	S2	0.8
1	S1	S3	0.2
2	S2	O1	0.5
3	S2	O2	0.5
4	S3	O1	1.0

I can graph the new network:

```
In [13]: G = nx.from_pandas_edgelist(data,'owner', 'shop', edge_attr = ('share'),
                                     create_using=nx.DiGraph())
         figure(num=None, figsize=(12, 6), dpi=80, facecolor='w', edgecolor='k')
         np.random.seed(1)
         pos=nx.spring_layout(G, k = 0.5, iterations = 20)
         node_labels = {node:node for node in G.nodes()}
         edge_labs = nx.get_edge_attributes(G,'share')
         nx.draw_networkx(G, pos, labels = node_labels, arrowstyle = '-|>',
                          edge_labels = edge_labs,
                      arrowsize = 20,  font_size = 15,
                          font_weight = 'bold', node_size = 1500)
         nx.draw_networkx_edge_labels(G, pos, edge_labels = edge_labs,
                                      font_size = 15)
```

```
Out[13]: {('S2', 'S1'): Text(-0.419962,0.255394,'0.8'),
          ('S3', 'S1'): Text(-0.550972,-0.537394,'0.2'),
          ('O1', 'S2'): Text(0.0509718,0.186287,'0.5'),
          ('O1', 'S3'): Text(-0.0800378,-0.606501,'1.0'),
          ('O2', 'S2'): Text(0.482076,0.665609,'0.5')}
```



Here in order to find the share that owner 1 has over shop 1, the 2 paths need to be multiplied and then added:

```
In [14]: from operator import mul
         from functools import reduce

         # Get all unique from each column
         owners = set(data['owner'])
         shops  = set(data['shop'])

         # Open data frames
         result = []
         summary = {}

         # Create loop
         for shop in shops:
             for owner in owners:
                 for path in nx.all_simple_paths(G, owner, shop):
                     # Grab and add shares for all paths
                     share = reduce(mul, (G[start][end]['share'] for start,
                                          end in zip(path[:-1], path[1:])), 1)
                     summary[(shop, owner)] = summary.get((shop, owner), 0) + share

         # Create final df
         summary = pd.DataFrame.from_dict(summary, orient = 'index',
                                          columns = 'share'.split())
         print(summary)

                share
(S3, O1)  1.0
```

```
(S2, O1)  0.5
(S2, O2)  0.5
(S1, S3)  0.2
(S1, O1)  0.6
(S1, S2)  0.8
(S1, O2)  0.4
```

This method will work for many different networks within the same data frame. The code multiplies through all paths and then adds together the results. I can turn this into final ownership in the same way as before:

```
In [15]: # Turn index to column
         summary = summary.reset_index()

         # Turn into 2 separate stripped columns
         summary["index"] = summary["index"].astype(str)
         summary['shop'], summary['owner'] = summary['index'].str.split(' ', 1).str
         summary["shop"] = summary.shop.str.replace('\W', '')
         summary["owner"] = summary.owner.str.replace('\W', '')

         # Tunr into percentage and add to owners
         summary["share"] = summary["owner"] + ": " + round(
             summary["share"]*100).map(str) + "%"

         # Keep only final owners
         summary = summary[summary.owner.str.startswith('O')]

         # Final aggregation
         summary = summary.sort_values(by = ["shop", "share"])
         summary = pd.DataFrame(summary.groupby('shop')['share'].apply(list))
         summary = summary.reset_index()
         summary.head()
```

```
Out[15]:   shop                    share
         0  S1    [O1: 60.0%, O2: 40.0%]
         1  S2    [O1: 50.0%, O2: 50.0%]
         2  S3    [O1: 100.0%]
```

Owner 1 has (100% x 20%) + (50% x 80%) = 60%. As a final example, I introduce a larger network with some more complicated loops.

```
In [16]: data = pd.DataFrame({
             'shop': ['S1','S1','S1','S1','S2','S2','S2','S3','S3','S3','S4','S4','S4',
                      'S5','S5','S5','S6','S7','S7','S7','S8','S8','S8','S9','S9','S9',
                      'S10','S10','S10','S10','S11','S11','S11','S11','S11','S12','S12',
                      'S12','S13','S13','S13','S14','S14','S14','S15','S15','S15','S16',
                      'S16','S16','S17','S17','S17'],
             'owner':['O1','O2','S2','S3','O1','S3','S4','O4','O3','O13','O2','O3','O4',
```

```
                    'S3','O5','O6','O5','O5','O6','S7','S5','S7','O6','S5','S7','O7',
                    'O7', 'O8', 'O9', 'O12', 'O8', 'O9','O10','O11','S10', 'O8','O11',
                    'O12','O10','O11','O13','O9','O10','O13', 'O15','O16','O17','O13',
                    'O15','S15','O14','O15','S16'],
            'share':[0.2, 0.2, 0.3, 0.3, 0.1, 0.5, 0.4, 0.15, 0.15, 0.7, 0.3, 0.3, 0.4,
                   0.2, 0.3, 0.5, 1.0, 0.2, 0.2, 0.6, 0.1, 0.1, 0.8, 0.25, 0.25, 0.5,
                   0.25, 0.25, 0.25, 0.25, 0.1,  0.2,  0.3,  0.15, 0.25,  0.3,  0.3,
                   0.4,  0.2,  0.2,  0.6,  0.4,  0.5,  0.1, 0.4, 0.4, 0.2, 0.3,
                   0.3, 0.4, 0.2, 0.2, 0.6]})
data.head()
```

Out[16]:

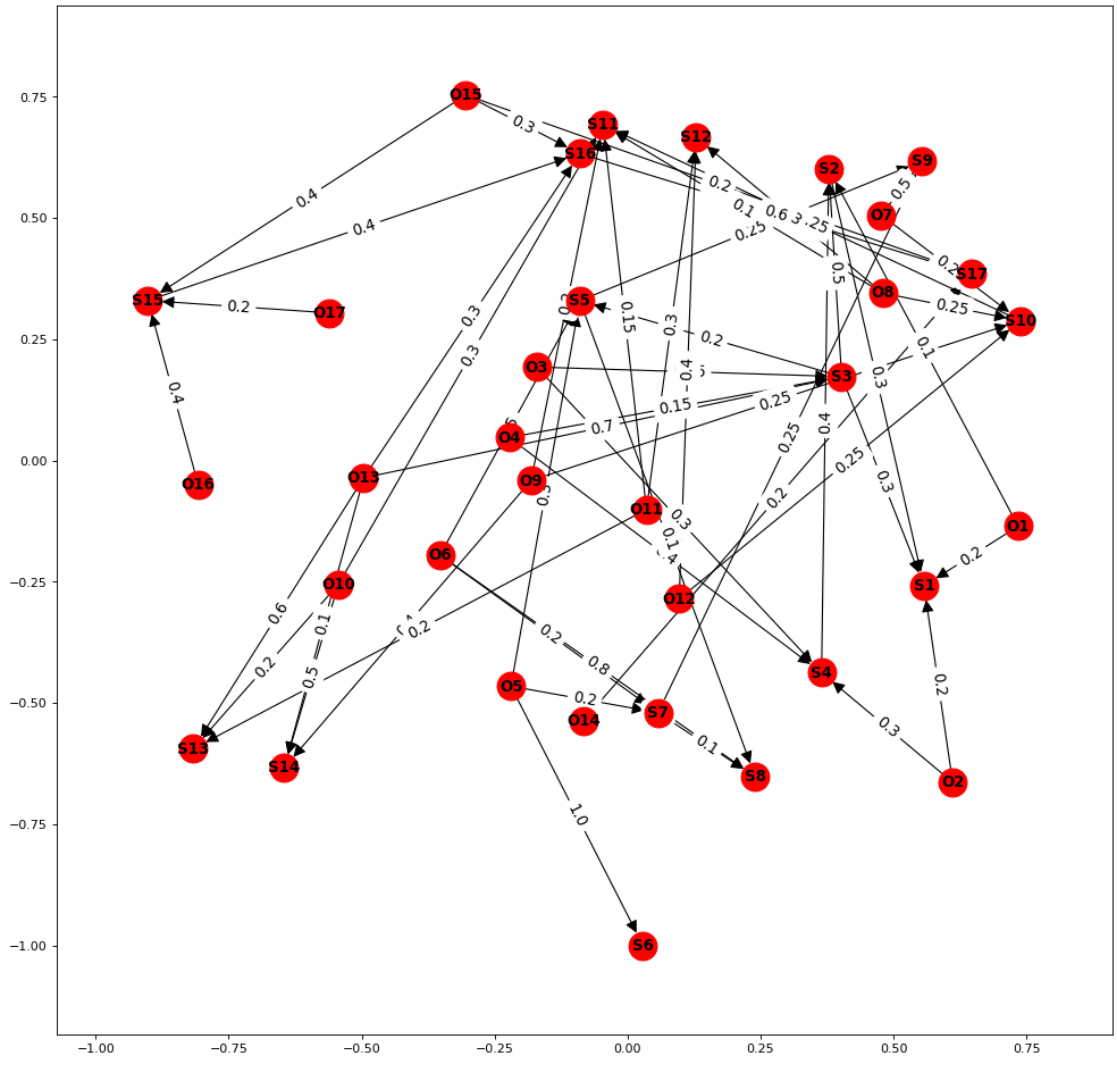
	shop	owner	share
0	S1	O1	0.2
1	S1	O2	0.2
2	S1	S2	0.3
3	S1	S3	0.3
4	S2	O1	0.1

And draw the network:

```
In [17]: G = nx.from_pandas_edgelist(data,'owner', 'shop',
                                     edge_attr = ('share'),
                                     create_using=nx.DiGraph())
         figure(num=None, figsize=(15, 15), dpi=80, facecolor='w', edgecolor='k')
         np.random.seed(2019)
         pos=nx.spring_layout(G, k = 0.5, iterations = 20)
         node_labels = {node:node for node in G.nodes()}
         edge_labs = nx.get_edge_attributes(G,'share')
         nx.draw_networkx(G, pos, labels = node_labels, arrowstyle = '-|>',
                          edge_labels = edge_labs,
                      arrowsize = 20,  font_size = 12,
                          font_weight = 'bold', node_size = 500)
         nx.draw_networkx_edge_labels(G, pos, edge_labels = edge_labs,
                                      font_size = 12)
```

```
Out[17]: {('O1', 'S1'): Text(0.645677,-0.197081,'0.2'),
          ('O1', 'S2'): Text(0.556105,0.232598,'0.1'),
          ('O2', 'S1'): Text(0.584159,-0.461142,'0.2'),
          ('O2', 'S4'): Text(0.487843,-0.550658,'0.3'),
          ('S2', 'S1'): Text(0.467671,0.171116,'0.3'),
          ('S3', 'S1'): Text(0.47984,-0.0428346,'0.3'),
          ('S3', 'S2'): Text(0.390268,0.386844,'0.5'),
          ('S3', 'S5'): Text(0.156345,0.251196,'0.2'),
          ('S4', 'S2'): Text(0.371355,0.0815997,'0.4'),
          ('O4', 'S3'): Text(0.0898721,0.110075,'0.15'),
          ('O4', 'S4'): Text(0.0709595,-0.19517,'0.4'),
          ('O3', 'S3'): Text(0.115887,0.18277,'0.15'),
          ('O3', 'S4'): Text(0.0969748,-0.122475,'0.3'),
          ('O13', 'S3'): Text(-0.046802,0.0685039,'0.7'),
```

```
('O13', 'S13'): Text(-0.656135,-0.315086,'0.6'),
('O13', 'S14'): Text(-0.570838,-0.334729,'0.1'),
('O13', 'S16'): Text(-0.292824,0.298434,'0.3'),
('S5', 'S8'): Text(0.075095,-0.16134,'0.1'),
('S5', 'S9'): Text(0.232034,0.473451,'0.25'),
('O5', 'S5'): Text(-0.154543,-0.0679852,'0.3'),
('O5', 'S6'): Text(-0.0951804,-0.732735,'1.0'),
('O5', 'S7'): Text(-0.0810062,-0.492817,'0.2'),
('O6', 'S5'): Text(-0.221249,0.0670801,'0.5'),
('O6', 'S7'): Text(-0.147712,-0.357752,'0.2'),
('O6', 'S8'): Text(-0.0564061,-0.423759,'0.8'),
('S7', 'S7'): Text(0.0573252,-0.520165,'0.6'),
('S7', 'S8'): Text(0.148631,-0.586172,'0.1'),
('S7', 'S9'): Text(0.30557,0.0486184,'0.25'),
('O7', 'S9'): Text(0.515414,0.561641,'0.5'),
('O7', 'S10'): Text(0.607799,0.396973,'0.25'),
('S10', 'S11'): Text(0.346267,0.489929,'0.25'),
('O8', 'S10'): Text(0.609102,0.31662,'0.25'),
('O8', 'S11'): Text(0.216784,0.518482,'0.1'),
('O8', 'S12'): Text(0.304212,0.505803,'0.3'),
('O9', 'S10'): Text(0.278549,0.123357,'0.25'),
('O9', 'S11'): Text(-0.113769,0.325219,'0.2'),
('O9', 'S14'): Text(-0.413561,-0.337463,'0.4'),
('O12', 'S10'): Text(0.417776,0.0012894,'0.25'),
('O12', 'S12'): Text(0.112886,0.190473,'0.4'),
('O10', 'S11'): Text(-0.294848,0.218256,'0.3'),
('O10', 'S13'): Text(-0.679937,-0.424782,'0.2'),
('O10', 'S14'): Text(-0.59464,-0.444426,'0.5'),
('O11', 'S11'): Text(-0.00526666,0.295345,'0.15'),
('O11', 'S12'): Text(0.0821617,0.282666,'0.3'),
('O11', 'S13'): Text(-0.390356,-0.347693,'0.2'),
('O15', 'S15'): Text(-0.602935,0.541672,'0.4'),
('O15', 'S16'): Text(-0.196916,0.69312,'0.3'),
('O15', 'S17'): Text(0.171055,0.569013,'0.2'),
('S15', 'S16'): Text(-0.495626,0.481306,'0.4'),
('O16', 'S15'): Text(-0.854457,0.139905,'0.4'),
('O17', 'S15'): Text(-0.7309,0.317027,'0.2'),
('S16', 'S17'): Text(0.278364,0.508647,'0.6'),
('O14', 'S17'): Text(0.281519,-0.0762429,'0.2')}
```



The network looks extremely complicated but the code can still find the owners of the network:

```
In [18]: # Get all unique from each column
         owners = set(data['owner'])
         shops  = set(data['shop'])

         # Open data frames
         result = []
         summary = {}

         # Create loop
         for shop in shops:
             for owner in owners:
                 for path in nx.all_simple_paths(G, owner, shop):
                     # Grab and add shares for all paths
```

```
                share = reduce(mul, (G[start][end]['share'] for start,
                                     end in zip(path[:-1], path[1:])), 1)
                summary[(shop, owner)] = summary.get((shop, owner), 0) + share

        # Create final df
        summary = pd.DataFrame.from_dict(summary, orient = 'index',
                                         columns = 'share'.split())
        summary.head(10)
```

Out[18]:

```
                share
(S2, O13)  0.350
(S2, S4)   0.400
(S2, O4)   0.235
(S2, O2)   0.120
(S2, O1)   0.100
(S2, O3)   0.195
(S2, S3)   0.500
(S8, O13)  0.014
(S8, S7)   0.100
(S8, O5)   0.050
```

Finally, turn into a proper data frame and print, like before:

In [19]:

```
# Turn index to column
summary = summary.reset_index()

# Turn into 2 separate stripped columns
summary["index"] = summary["index"].astype(str)
summary['shop'], summary['owner'] = summary['index'
                                            ].str.split(' ', 1).str
summary["shop"] = summary.shop.str.replace('\W', '')
summary["owner"] = summary.owner.str.replace('\W', '')

# Tunr into percentage and add to owners
summary["share"] = summary["owner"] + ": " + round(
    summary["share"]*100,5).map(str) + "%"

# Keep only final owners
summary = summary[summary.owner.str.startswith('O')]

# Final aggregation
summary = summary.sort_values(by = ["shop", "share"])
summary = pd.DataFrame(summary.groupby('shop')['share'].apply(list))
summary = summary.reset_index()
summary.head(20)
```

Out[19]:

```
    shop  \
0   S1
1   S10
```

```
2   S11
3   S12
4   S13
5   S14
6   S15
7   S16
8   S17
9   S2
10  S3
11  S4
12  S5
13  S6
14  S7
15  S8
16  S9
```

```
                                                                                share
0   [O13: 31.5%, O1: 23.0%, O2: 23.6%, O3: 10.35%, O4: 11.55%]
1   [O12: 25.0%, O7: 25.0%, O8: 25.0%, O9: 25.0%]
2   [O10: 30.0%, O11: 15.0%, O12: 6.25%, O7: 6.25%, O8: 16.25%, O9: 26.25%]
3   [O11: 30.0%, O12: 40.0%, O8: 30.0%]
4   [O10: 20.0%, O11: 20.0%, O13: 60.0%]
5   [O10: 50.0%, O13: 10.0%, O9: 40.0%]
6   [O15: 40.0%, O16: 40.0%, O17: 20.0%]
7   [O13: 30.0%, O15: 46.0%, O16: 16.0%, O17: 8.0%]
8   [O13: 18.0%, O14: 20.0%, O15: 47.6%, O16: 9.6%, O17: 4.8%]
9   [O13: 35.0%, O1: 10.0%, O2: 12.0%, O3: 19.5%, O4: 23.5%]
10  [O13: 70.0%, O3: 15.0%, O4: 15.0%]
11  [O2: 30.0%, O3: 30.0%, O4: 40.0%]
12  [O13: 14.0%, O3: 3.0%, O4: 3.0%, O5: 30.0%, O6: 50.0%]
13  [O5: 100.0%]
14  [O5: 20.0%, O6: 20.0%]
15  [O13: 1.4%, O3: 0.3%, O4: 0.3%, O5: 5.0%, O6: 87.0%]
16  [O13: 3.5%, O3: 0.75%, O4: 0.75%, O5: 12.5%, O6: 17.5%, O7: 50.0%]
```

All the ownership shares add to 100%. The code will scale to work with thousands of different networks and owners/shops or whatever the example might be.