Social Network Analysis

Network Visualization

Dr. Chun-Hsiang Chan

Department of Geography,
National Taiwan Normal University

Photo credit: midjourney

Outline

– Why We Need Graph Visualization?
– Task 1: Create a Random Graph
– Task 2: Visualize with Networkx
– Task 3: Visualize with pyVis
– Task 4: Visualize with Jaal
– Task 5: Visualize with Gephi

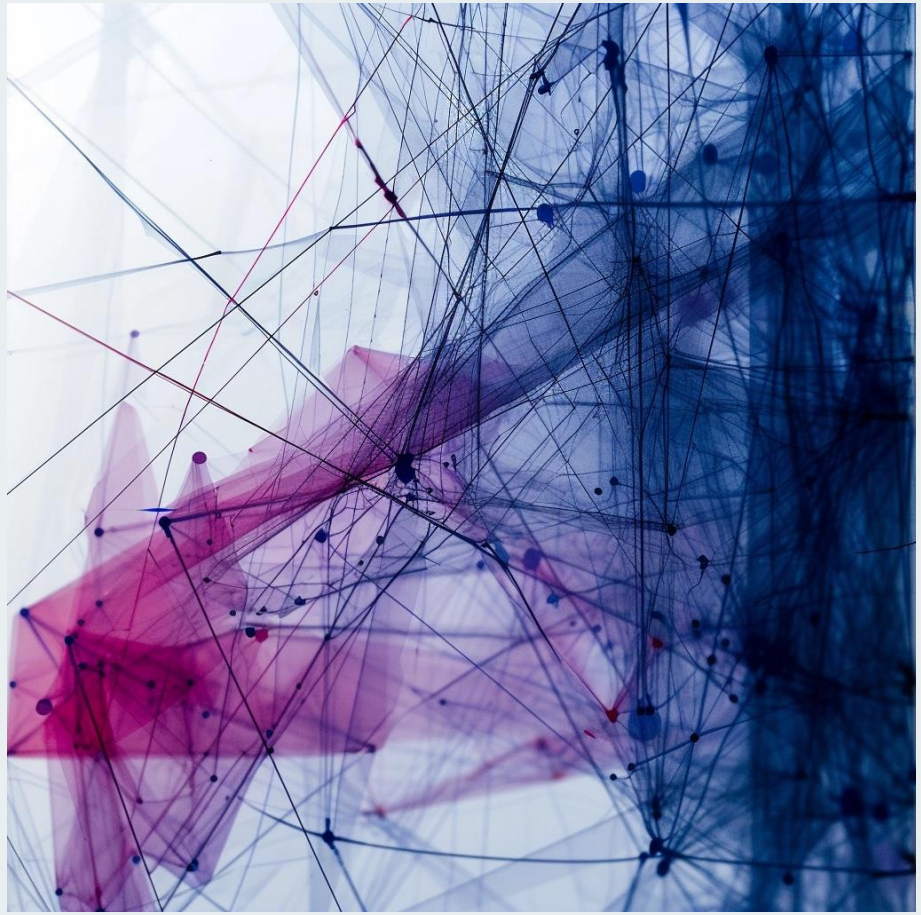


Photo credit: midjourney

Why We Need Graph Visualization?

- Graph visualization is used to demonstrate the relationship between nodes with node-edge quantification.
- Different to map visualization, the closer nodes does not indicate that these nodes have geographical proximity, only means that there relationship might be stronger than others.
- However, different graph visualization may illustration different story of your graph; therefore, you need to choose the best way to present your graph.

Why We Need Graph Visualization?

- In today's class, we are going to introduce graph-based visualization, e.g., networkx, pyvis, and Jaal.
- If you want to know how to plot a graph onto a map, please refer to Python Programming lectures. Basically, we use plt.plot and plt.scatter to draw the straight lines and nodes based on the attributes and geometry of provided GeoDataFrame, respectively.

Task 1: Create a Random Graph

– First, we need to import necessary packages:

```
# import packages
import numpy as np
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
```

Task 1: Create a Random Graph

– Create a random graph with **Erdos Renyi**:

```
# create Erdos Renyi random graph
# a directed graph with 50 nodes and 0.2 edge-link probability
G = nx.erdos_renyi_graph(50, 0.2, seed=21, directed=False)
```

Task 2: Visualize with Networkx

– **Networkx** is a useful Python graph packages, which is convenient for social/complex network analysis via numerous quantitative indicators.

– **Installation**

```
pip install networkx
```

Task 2: Visualize with Networkx

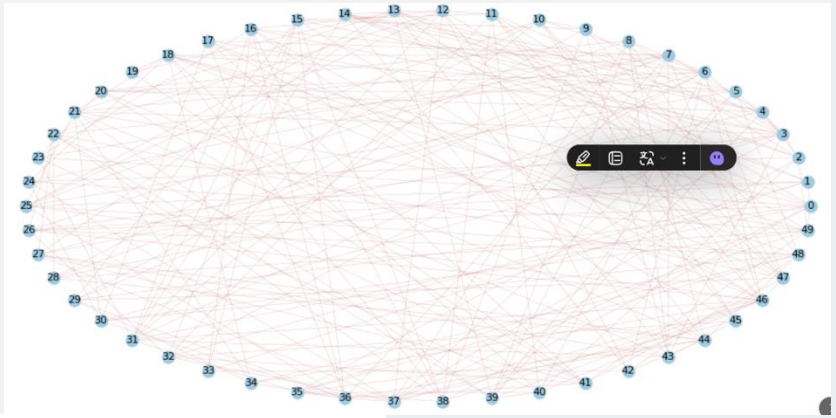
– Configuration for visualization

```
# set options
options = {
    "font_size": 8,
    "node_size": 100,
    "node_color": "#A0CBE2",
    "edge_color": "brown",
    "linewidths": 0.1,
    "width": 0.08,
}
```

Task 2: Visualize with Networkx

– **Circular Style**

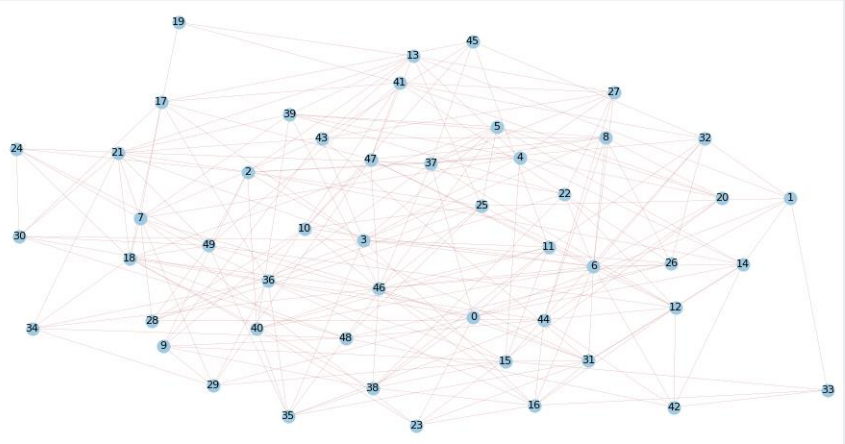
```
# circular
plt.subplots(figsize=[12,6], dpi=100)
nx.draw_circular(G,  **options, with_labels = True)
ax = plt.gca()
ax.margins(0.001)
plt.axis("off")
plt.show()
```



Task 2: Visualize with Networkx

– **Kamada Kawai Style**

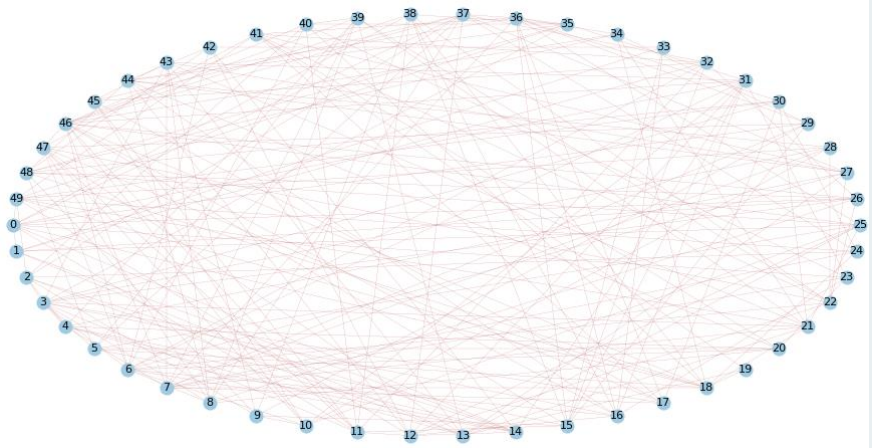
```
# kamada kawai
plt.subplots(figsize=[12,6], dpi=100)
nx.draw_kamada_kawai(G,  **options, with_labels = True)
ax = plt.gca()
ax.margins(0.001)
plt.axis("off")
plt.show()
```



Task 2: Visualize with Networkx

– **Shell Style**

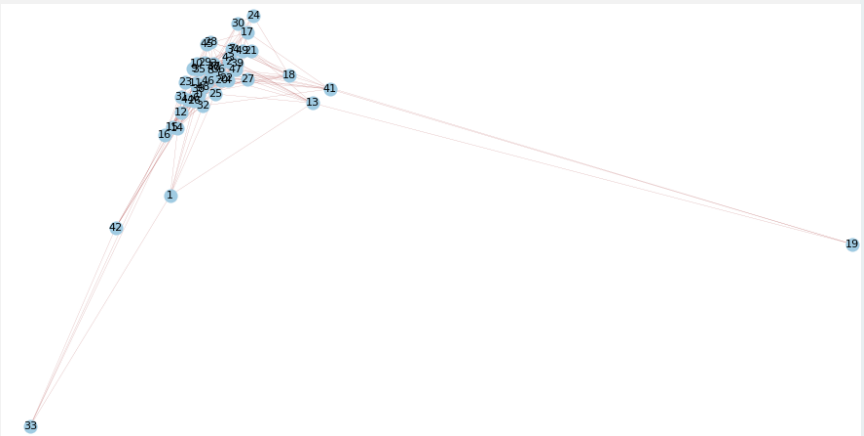
```
# Shell
plt.subplots(figsize=[12,6], dpi=100)
nx.draw_shell(G,  **options, with_labels = True)
ax = plt.gca()
ax.margins(0.001)
plt.axis("off")
plt.show()
```



Task 2: Visualize with Networkx

– **Spectral Style**

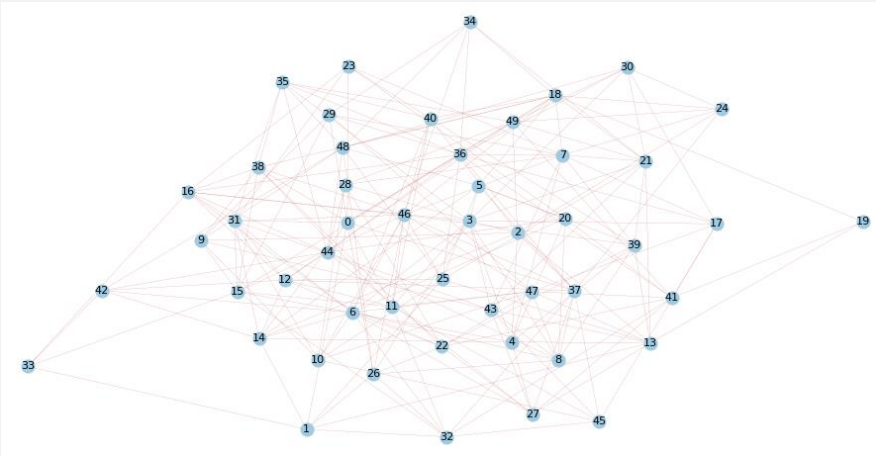
```
# Spectral
plt.subplots(figsize=[12,6], dpi=100)
nx.draw_spectral(G,  **options, with_labels = True)
ax = plt.gca()
ax.margins(0.001)
plt.axis("off")
plt.show()
```



Task 2: Visualize with Networkx

– **Spring Style**

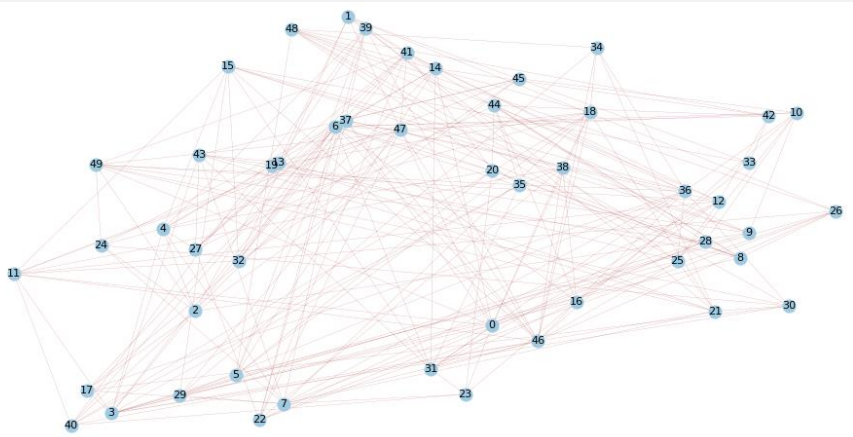
```
# Spectral
plt.subplots(figsize=[12,6], dpi=100)
nx.draw_spring(G,  **options, with_labels = True)
ax = plt.gca()
ax.margins(0.001)
plt.axis("off")
plt.show()
```



Task 2: Visualize with Networkx

– **Random Style**

```
# Random
plt.subplots(figsize=[12,6], dpi=100)
nx.draw_random(G,  **options, with_labels = True)
ax = plt.gca()
ax.margins(0.001)
plt.axis("off")
plt.show()
```



Task 3: Visualize with pyVis

- The developers of Networkx emphasize that the library is designed for quantifying and analyzing network structures and topological properties, rather than for visualization purposes.
- In many cases, however, visualization is essential for understanding interactions among nodes. Static visualization methods often fall short in effectively conveying these relationships.
- To address this limitation, we introduce PyVis, a dynamic visualization tool that enables interactive exploration of graphs. It can display networks directly within a Jupyter Notebook or export them as standalone HTML files for broader accessibility.

Task 3: Visualize with pyVis

- **Installation**

```
pip install pyvis
```

- Before starting to visualize our graph, pyvis only conducts its own node and edge format.

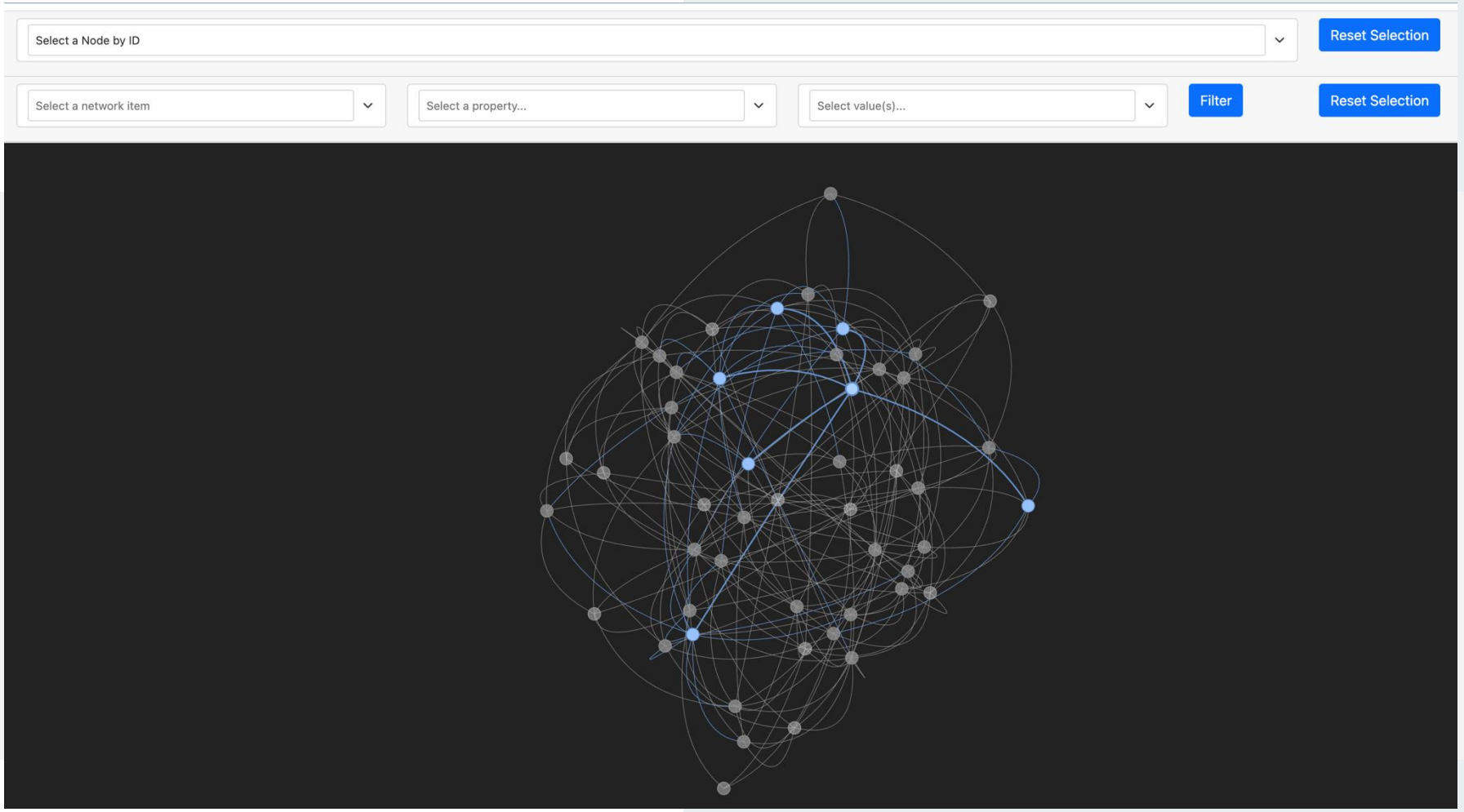
Task 3: Visualize with pyVis

– Import graph from network

```
# create vis network
visG = net.Network(notebook=True, height="800px", width="100%",
                   bgcolor="#222222", font_color="white",
                   filter_menu=True, select_menu=True,
                   cdn_resources='remote')
# import graph from networkx
visG.from_nx(G)
# show
visG.show_buttons()
visG.show('pyvis_ex.html')
```

Task 3: Visualize with pyVis

– **Results**



Task 3: Visualize with pyVis

- **Node Color and Size with Group Label and Importance**
- To demonstrate the **size** attribute, we use a random number to assign the size to each node.

```
# extract networkx node to dataframe
G_nodes = pd.DataFrame.from_records(np.vstack(list(G.nodes)),
                                    columns=['ID'])
# add size with random values
G_nodes['importance'] = abs(np.random.normal(15, 12,
                                    G_nodes.shape[0]))
```

Task 3: Visualize with pyVis

- **Node Color and Size with Group Label and Importance**
- To demonstrate the **group** attribute, we use a random number to assign the group label to each node.

```
# add group with random values
G_nodes['group'] = np.random.randint(5, size=G_nodes.shape[0])+1
# add node label with random values
G_nodes['code'] = 'S_'+G_nodes['ID'].astype('str')
# preview data
G_nodes.head()
```

	ID	importance	group	code
0	0	22.095438	3	S_0
1	1	4.494944	2	S_1
2	2	10.828733	1	S_2
3	3	13.368127	3	S_3
4	4	24.870871	4	S_4

Task 3: Visualize with pyVis

- **Edge Width with Weight**
- To demonstrate the **edge** attribute, we use a random number to assign the weight to each edge.

```
# extract networkx edge to dataframe
G_edges = pd.DataFrame.from_records(np.vstack(list(G.edges)),
columns=['source','target'])
# add weight with random values
G_edges['weight'] = abs(np.random.normal(100, 25,
G_edges.shape[0]))**2/1000
```

	source	target	weight
0	0	1	5.494500
1	0	11	3.121376
2	0	14	8.873845
3	0	18	6.916639
4	0	23	14.928490

Task 3: Visualize with pyVis

– **Assign Colors and Settings**

```
color_map = {1  :'#CCFF00',  2  :'#CCCC33', 3:'#CC9966',  4:'#CC6699',
             5  :'#CC33CC', 6  :'#CC00FF',  7:'#9900FF',   8:'#6600CC',
             9  :'#660066',   10:'#660033',  11:'#CC99FF',12:'#FFCCFF',
             13:'#FFFFCC', 14:'#FF3300', 15:'#333333',  16:'#666666'}

options = {'edge_color': '#FFDEA2', 'width': .5, 'with_labels': False,
           'font_weight': 'regular'}
```

Task 3: Visualize with pyVis

– **Starting to Plot**

```
# initialize
g = net.Network(height="800px", width="100%", bgcolor="#222222",
                font_color="white",  filter_menu=True, select_menu=True,
                cdn_resources='remote')
# add node
for i in range(len(G_nodes['code'])):
    g.add_node(int(G_nodes['ID'][i]), size=int(G_nodes['importance'][i]),
    label=G_nodes['code'][i], group=int(G_nodes['group'][i]))
```

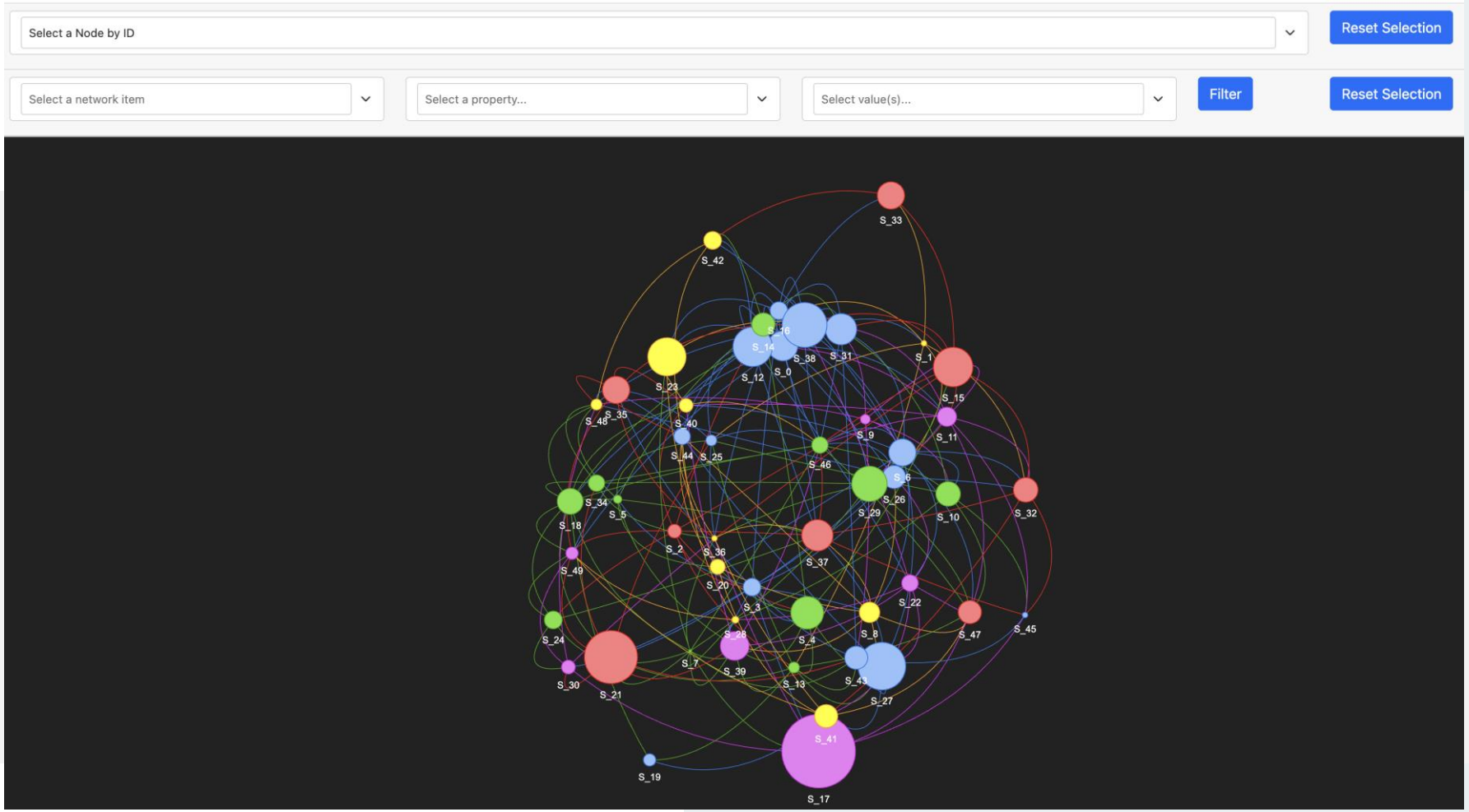
Task 3: Visualize with pyVis

– **Starting to Plot**

```
# add edge
for i in range(len(G_edges)):
        g.add_edge(int(G_edges['source'][i]), int(G_edges['target'][i]),
        weight=G_edges['weight'][i])
# add options
g.set_template_dir = None
g.show_buttons()
# export
g.show('adv_ex.html', notebook=False)
```

Task 3: Visualize with pyVis

– **Results**



Task 4: Visualize with Jaal

- Sometimes we do not need several parameters to adjust the visualization. Here, we are going to introduce another graph visualization package – Jaal.
- This package is designed to plot your graph inside your Jupyter Notebook directly with minimal (critical) filters and functions for visualization adjustment.

Task 4: Visualize with Jaal

- In Jaal, the allowed input argument names are very restricted, especially for column names; for example, the **source** and **target** should be named **from** and **to**, respectively.
- Additionally, the term of "**size**" cannot be used for edge or node column names.

```
# rename columns for Jaal
# Jaal reinforce source & target should be named as from and to
G_edges.columns = ['from', 'to', 'weight']
G_nodes.columns = ['id', 'importance', 'group', 'code']
```

Task 4: Visualize with Jaal

– Group number or label should be a string.

```
# change the data type of the group column into String
G_nodes['group'] = G_nodes['group'].astype(str)
```

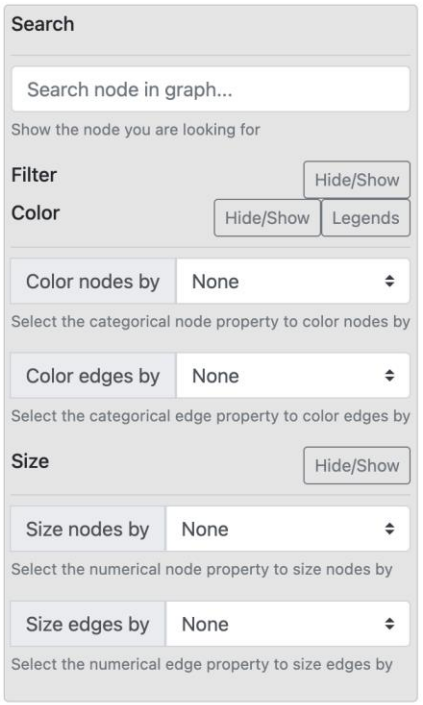
– Define edge weight divisions via mean and standard deviations

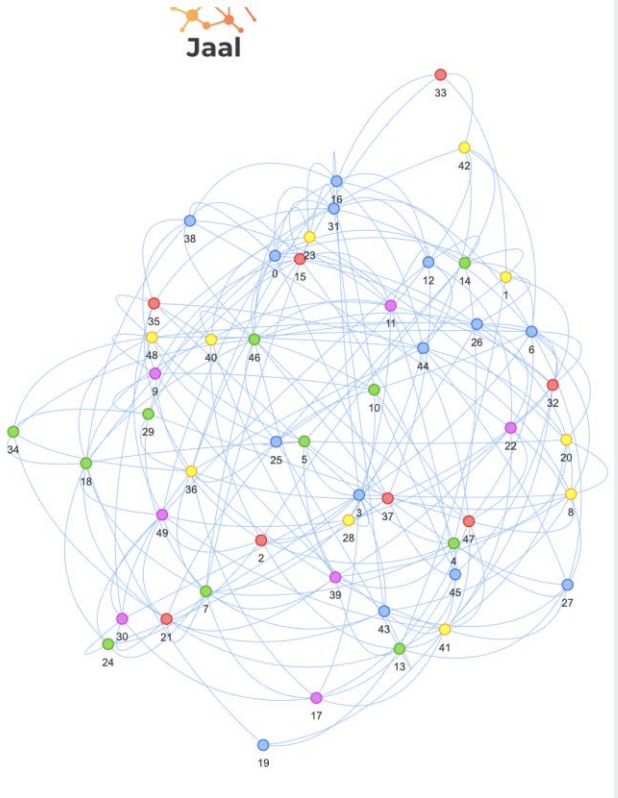
```
# compute the mean and standard deviation for division
mean = np.mean(G_edges['weight'])
stddev = np.std(G_edges['weight'], ddof=1)
```

Task 4: Visualize with Jaal

– **Simple Plot**

```
# simple demo
Jaal(G_edges, G_nodes).plot()
```





Task 4: Visualize with Jaal

– Adjustment

```
Parsing the data...Done
```

Search

Search node in graph...

Show the node you are looking for

Filter Hide/Show

Color Hide/Show Legends

Color nodes by group

Select the categorical node property to color nodes by

Color edges by strength

Select the categorical edge property to color edges by

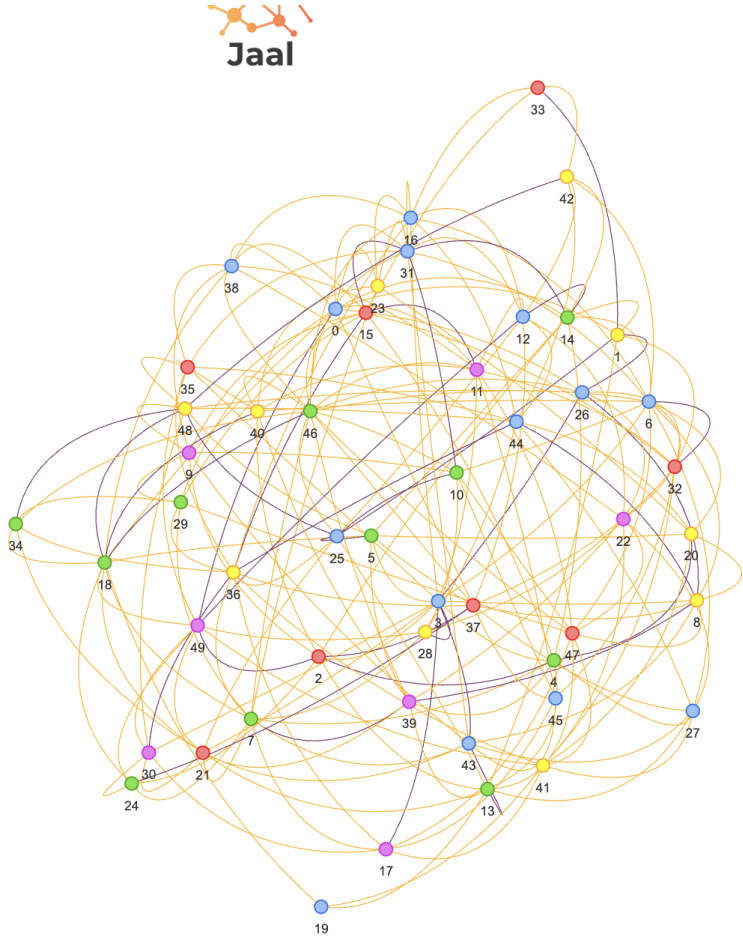
Size Hide/Show

Size nodes by None

Select the numerical node property to size nodes by

Size edges by None

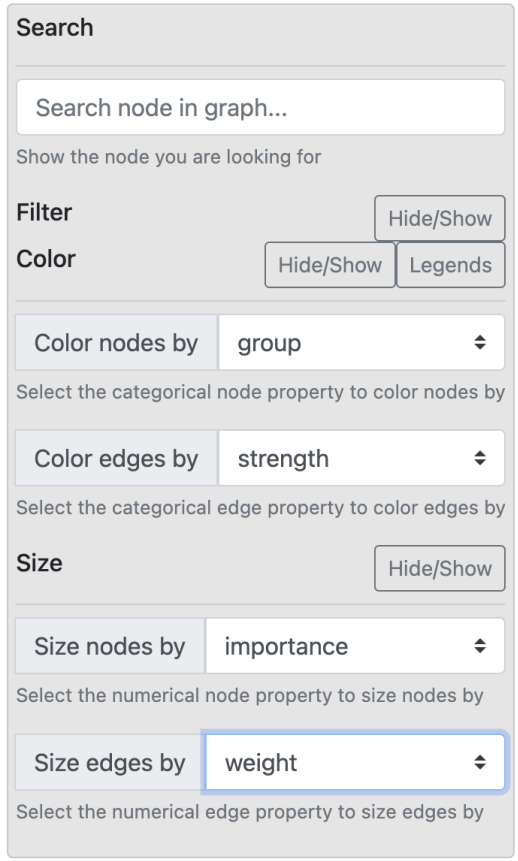
Select the numerical edge property to size edges by

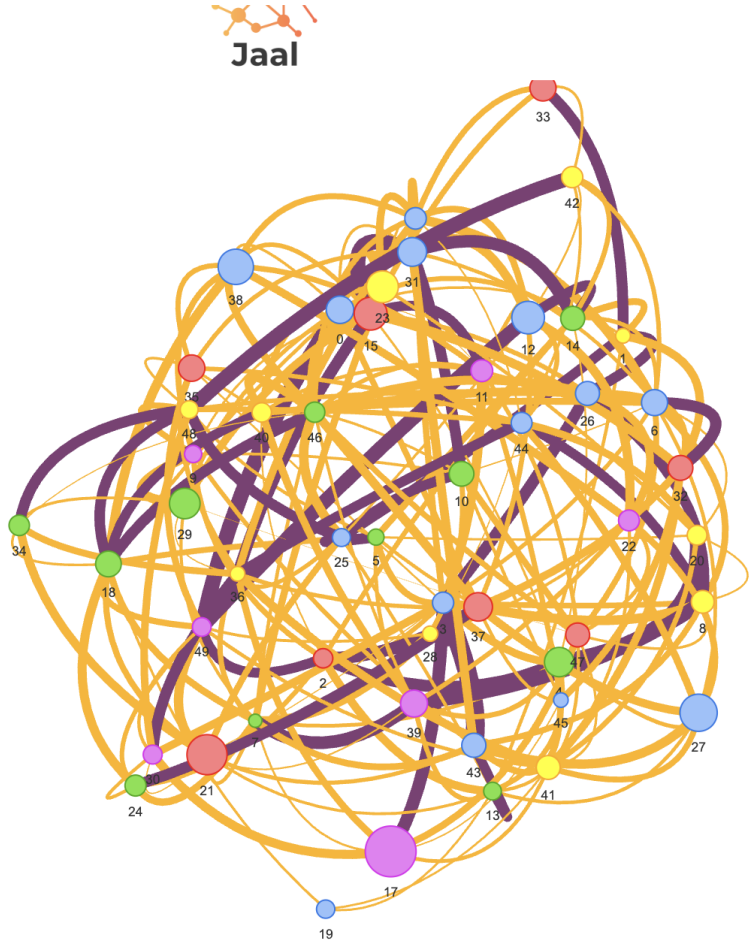


Task 4: Visualize with Jaal

– Adjustment

```
Parsing the data...Done
```

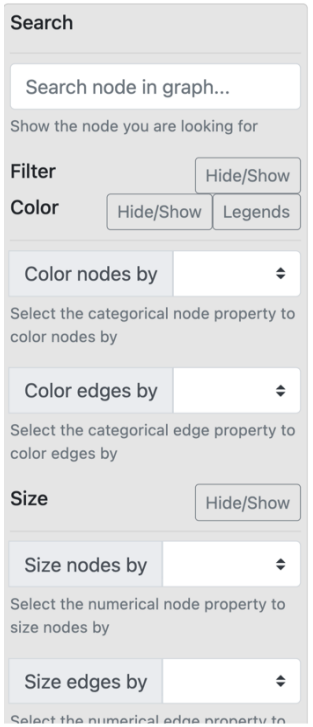


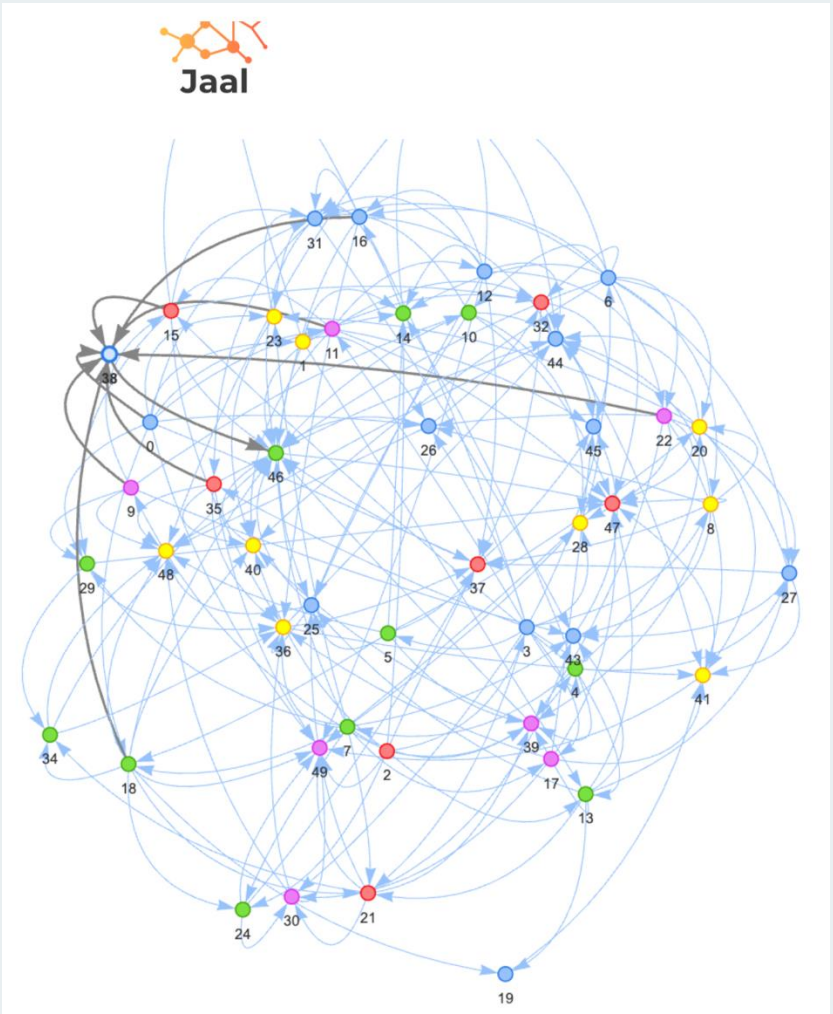


Task 4: Visualize with Jaal

– For Directed Graph

```
# set for directed graph
Jaal(G_edges, G_nodes).plot(directed=True)
```





Task 4: Visualize with Jaal

– Adjustment

Task 4: Visualize with Jaal

– **Add More Settings**

```
# init Jaal and run server
Jaal(G_edges, G_nodes).plot(vis_opts={'height': '600px', # change height
                                      'interaction':{'hover': True}, # turn on-off the hover
                                      'physics':{'stabilization':{'iterations': 100}}},
                            ) # define the convergence iteration of network
```

Task 4: Visualize with Jaal

– Results

```
Parsing the data...Done
```

Search

Search node in graph...

Show the node you are looking for

Filter Hide/Show

Color Hide/Show Legends

Color nodes by

Select the categorical node property to color nodes by

Color edges by

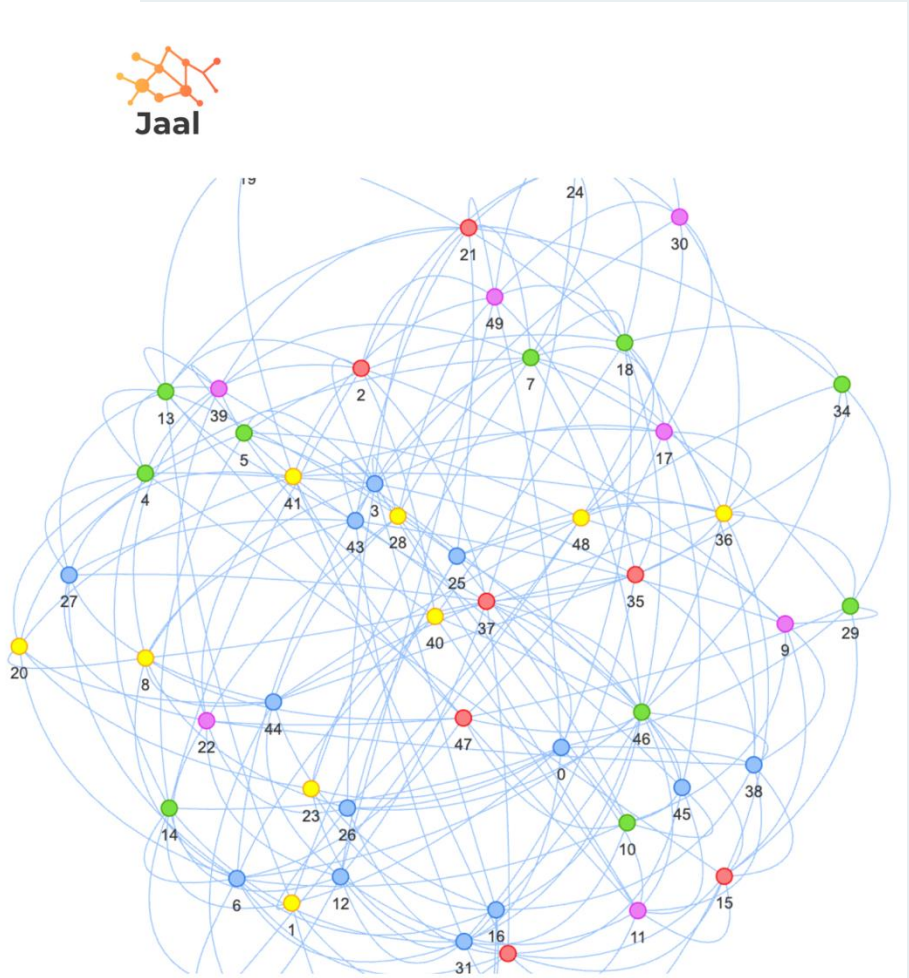
Select the categorical edge property to color edges by

Size Hide/Show

Size nodes by

Select the numerical node property to size nodes by

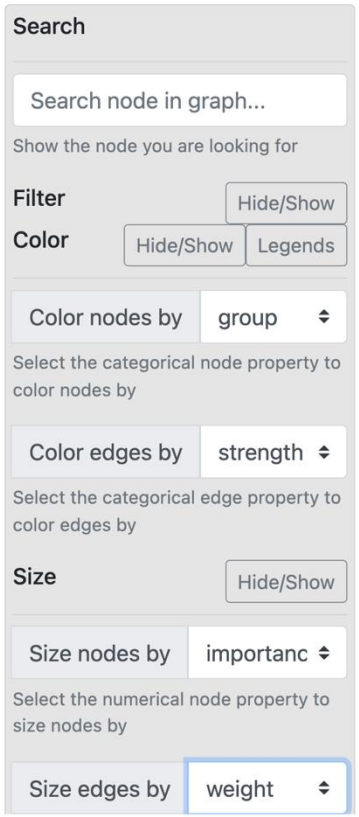
Size edges by



Task 4: Visualize with Jaal

– Results

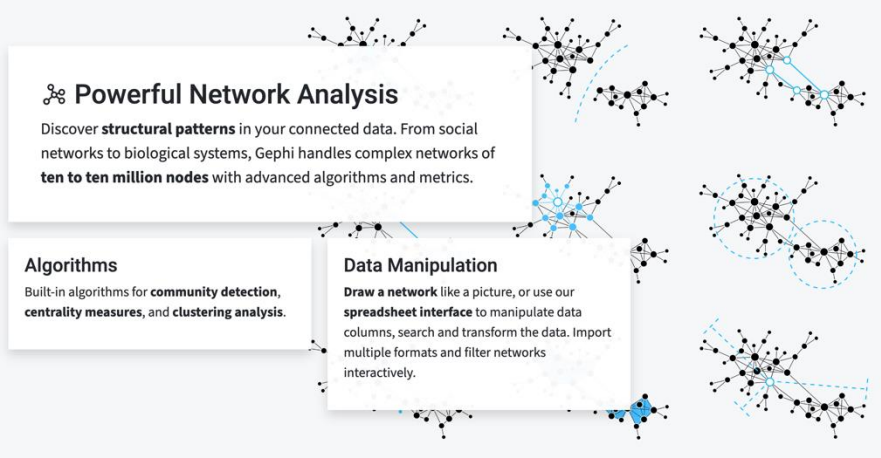
```
Parsing the data...Done
```

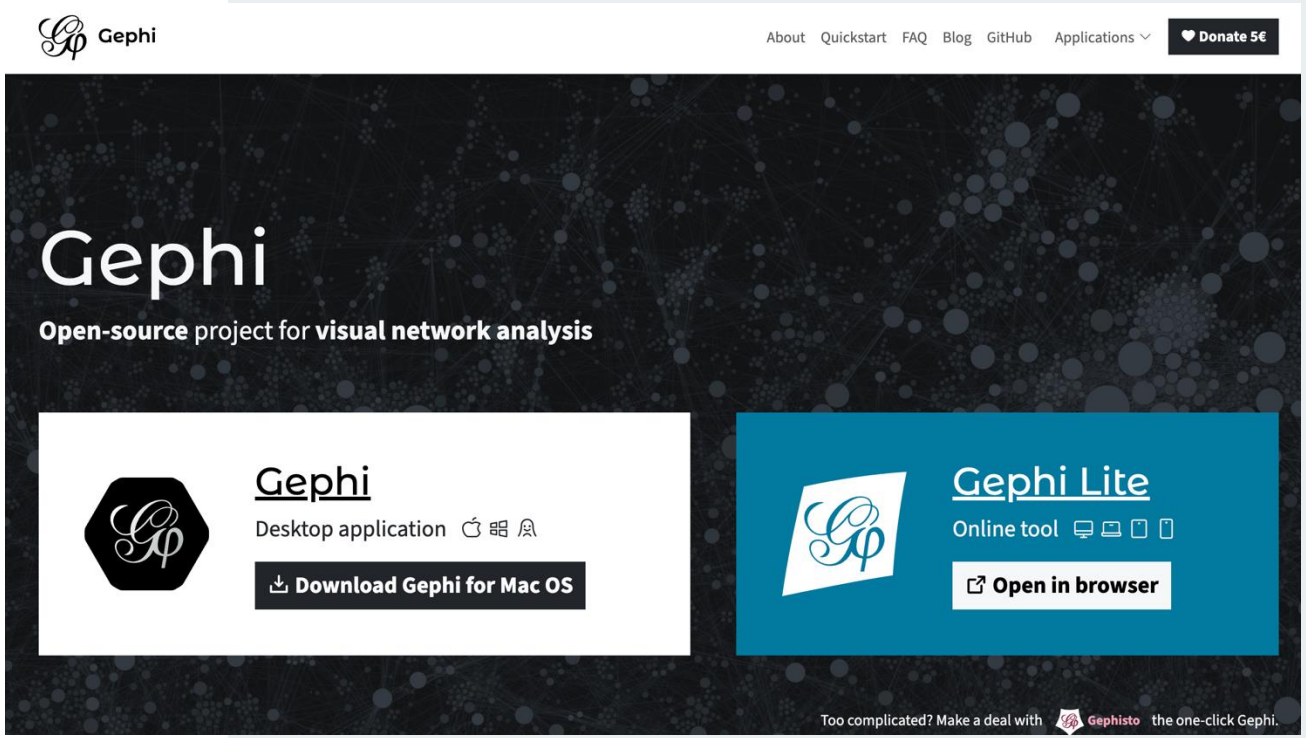


Task 5: Visualize with Gephi

– In addition to Python-based visualization, we introduce Gephi, a software tool for graph visualization.

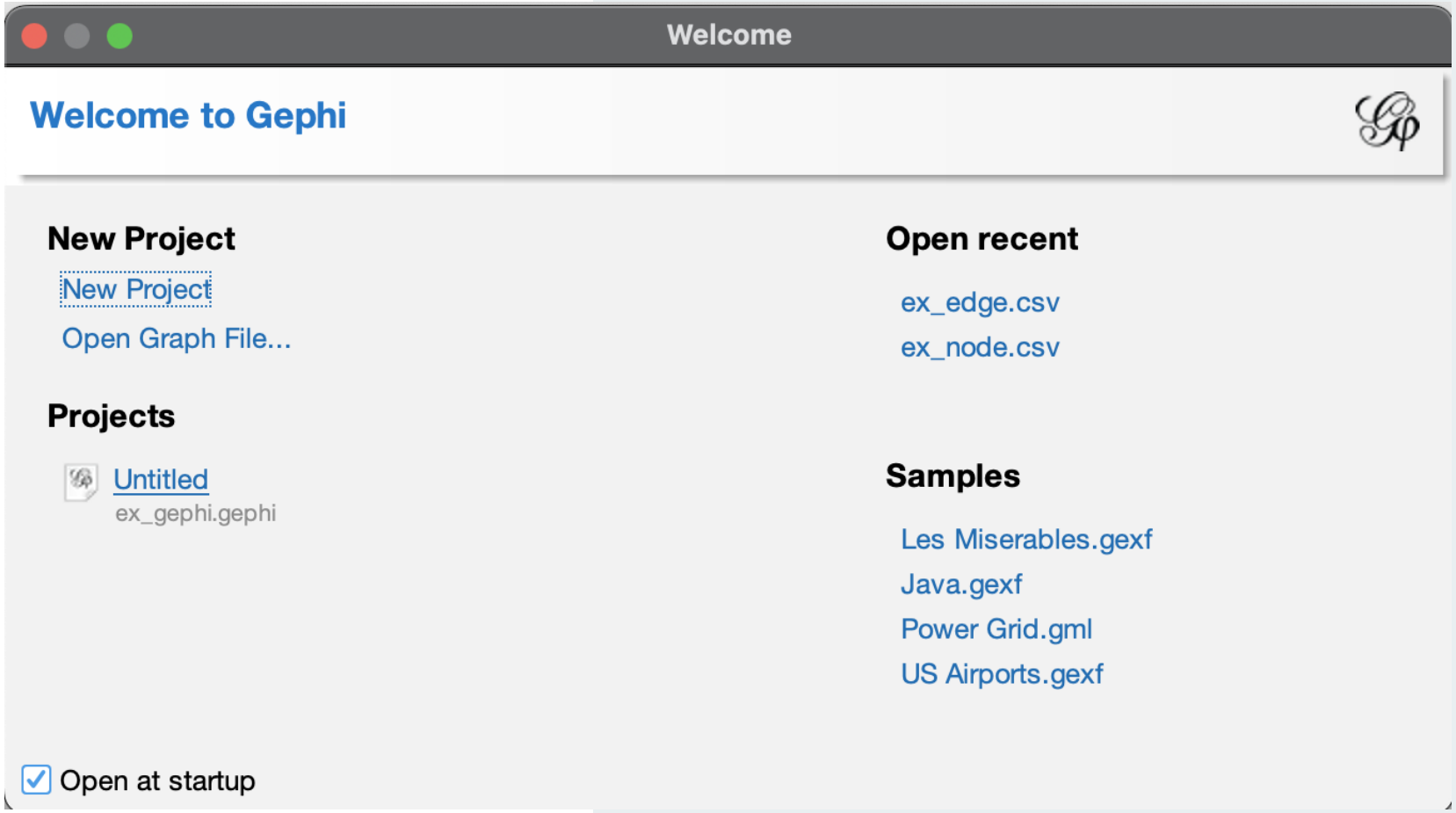
– **Installation:** https://gephi.org/



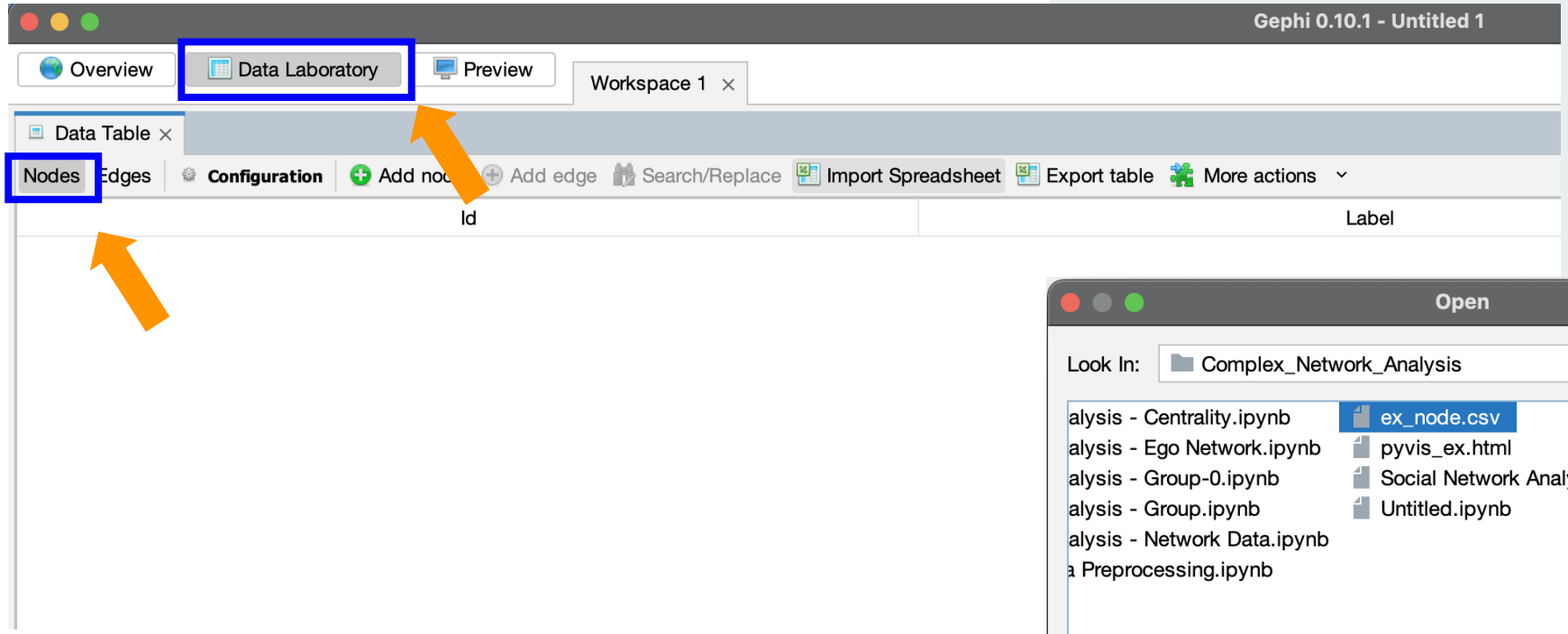


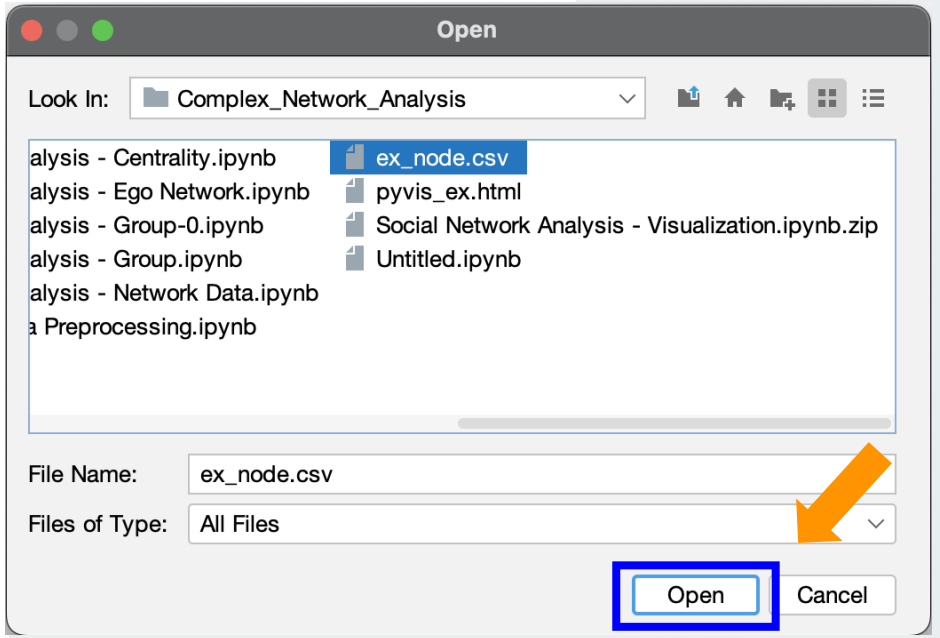
Task 5: Visualize with Gephi

– New Project

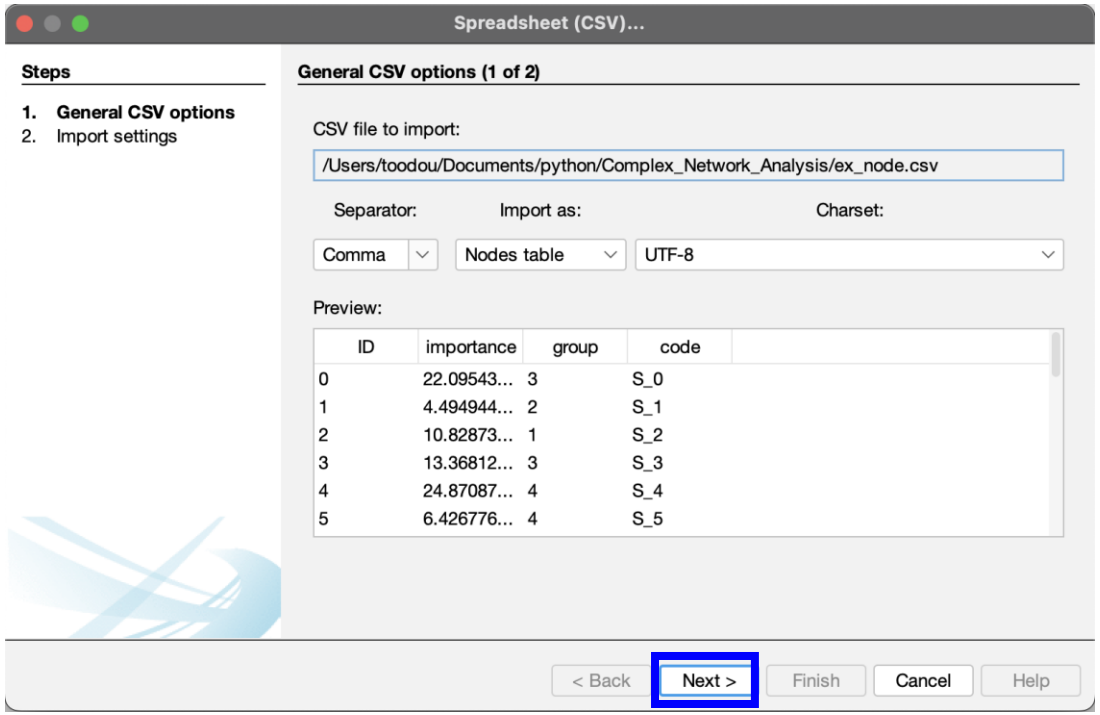


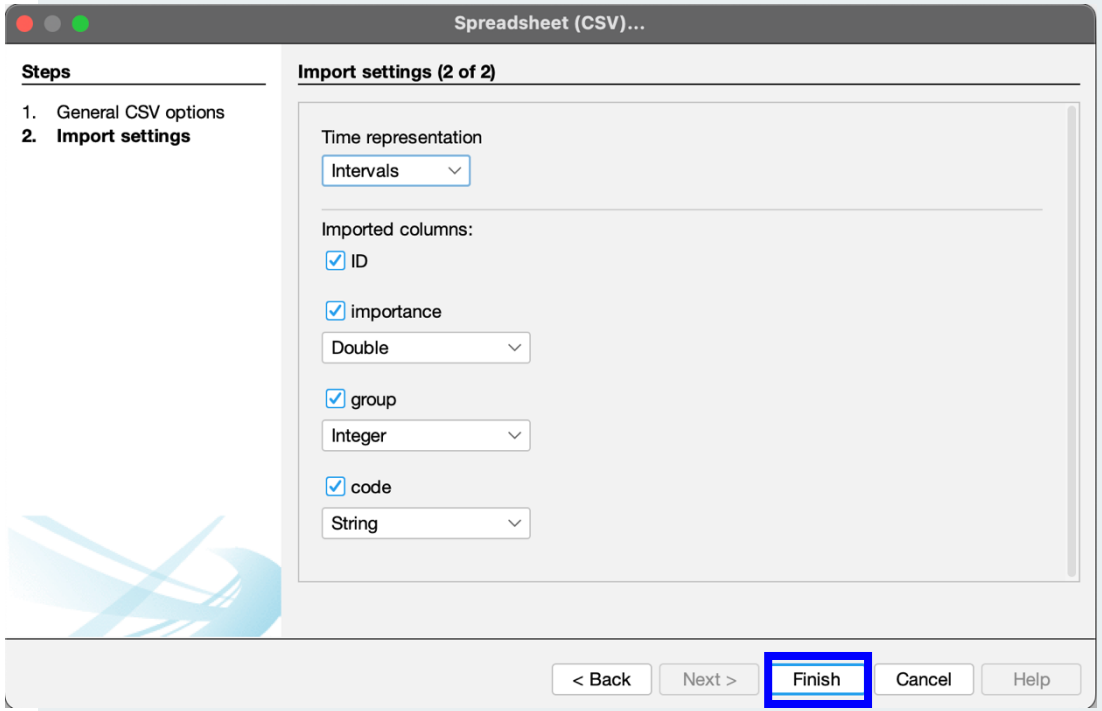
Task 5: Visualize with Gephi



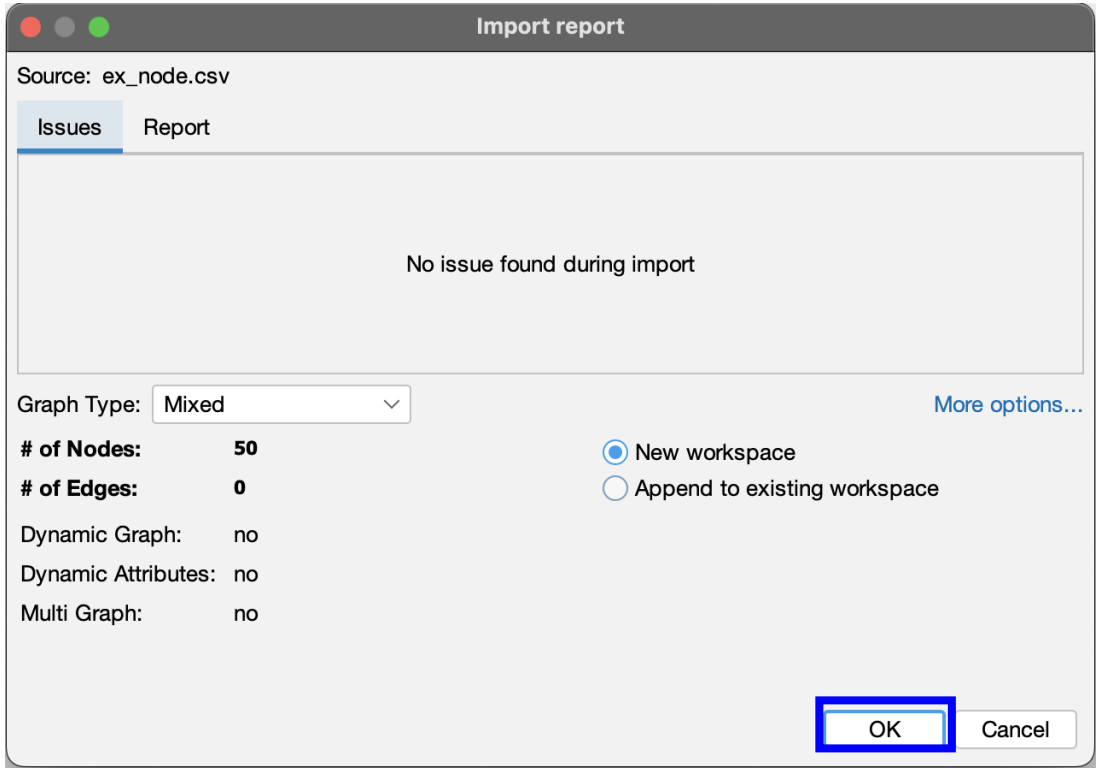


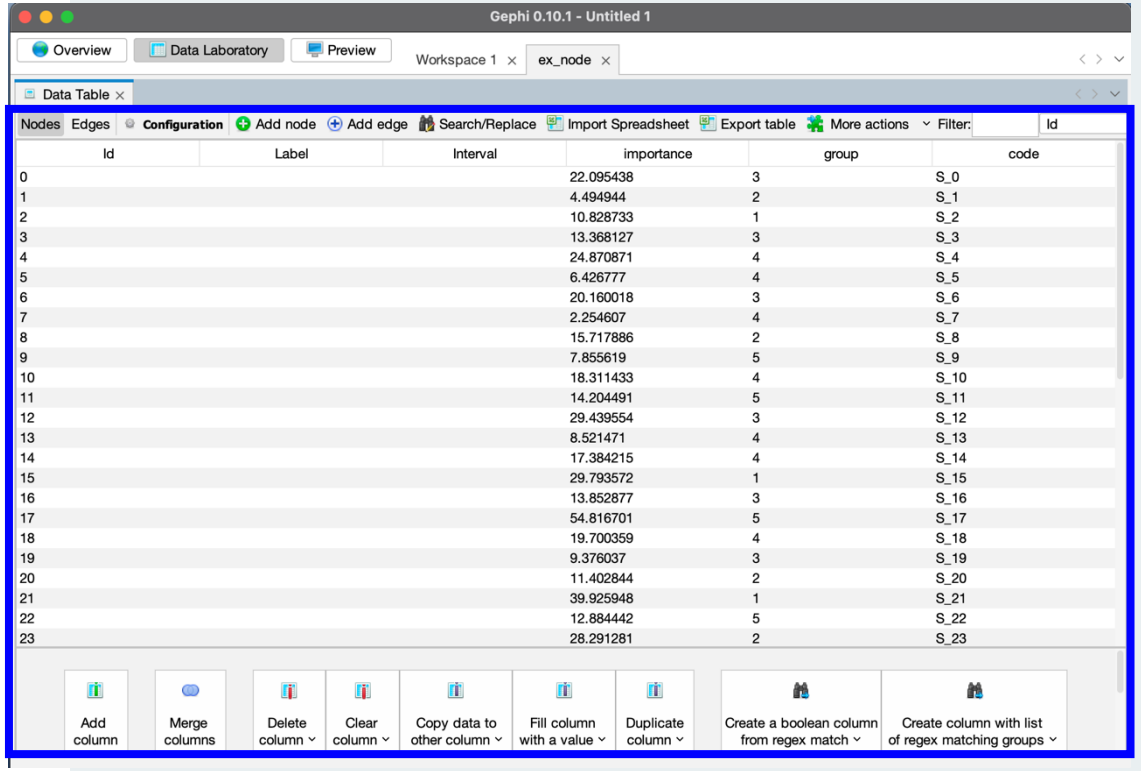
Task 5: Visualize with Gephi



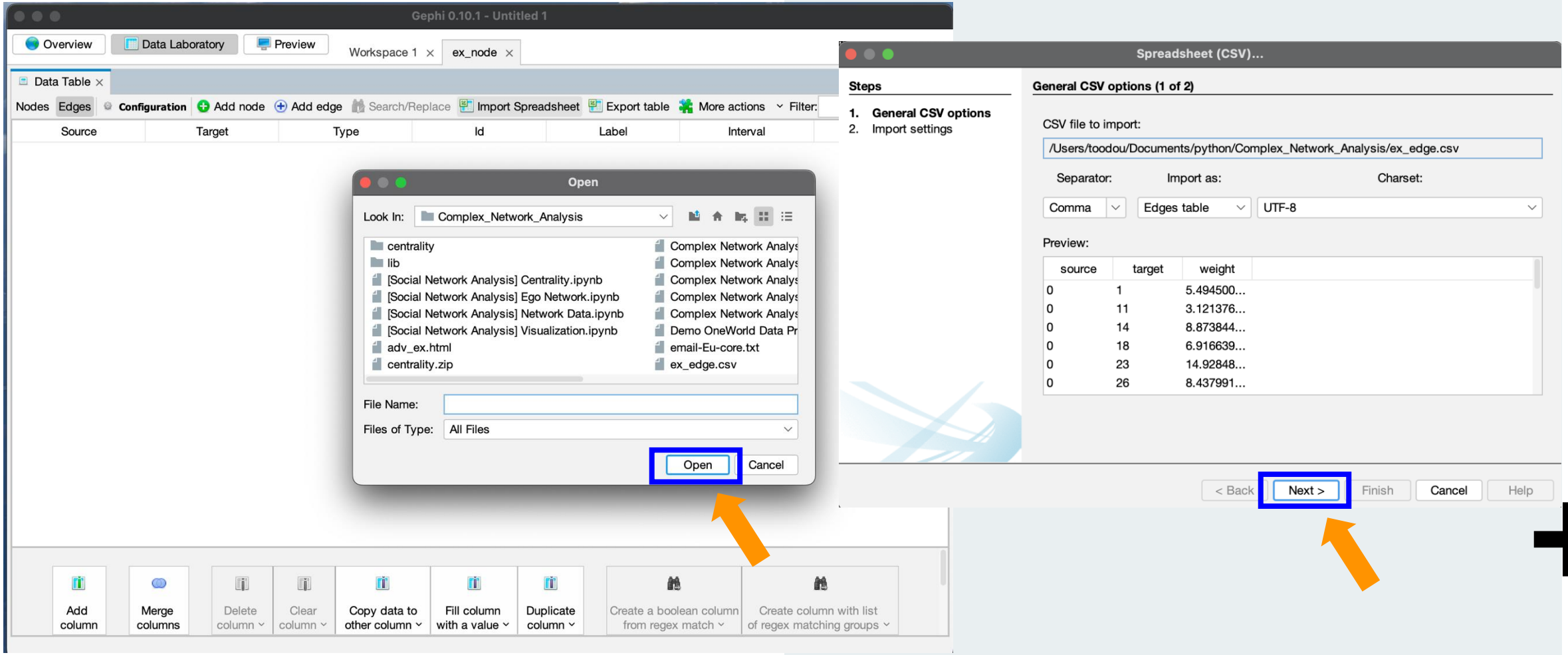


Task 5: Visualize with Gephi

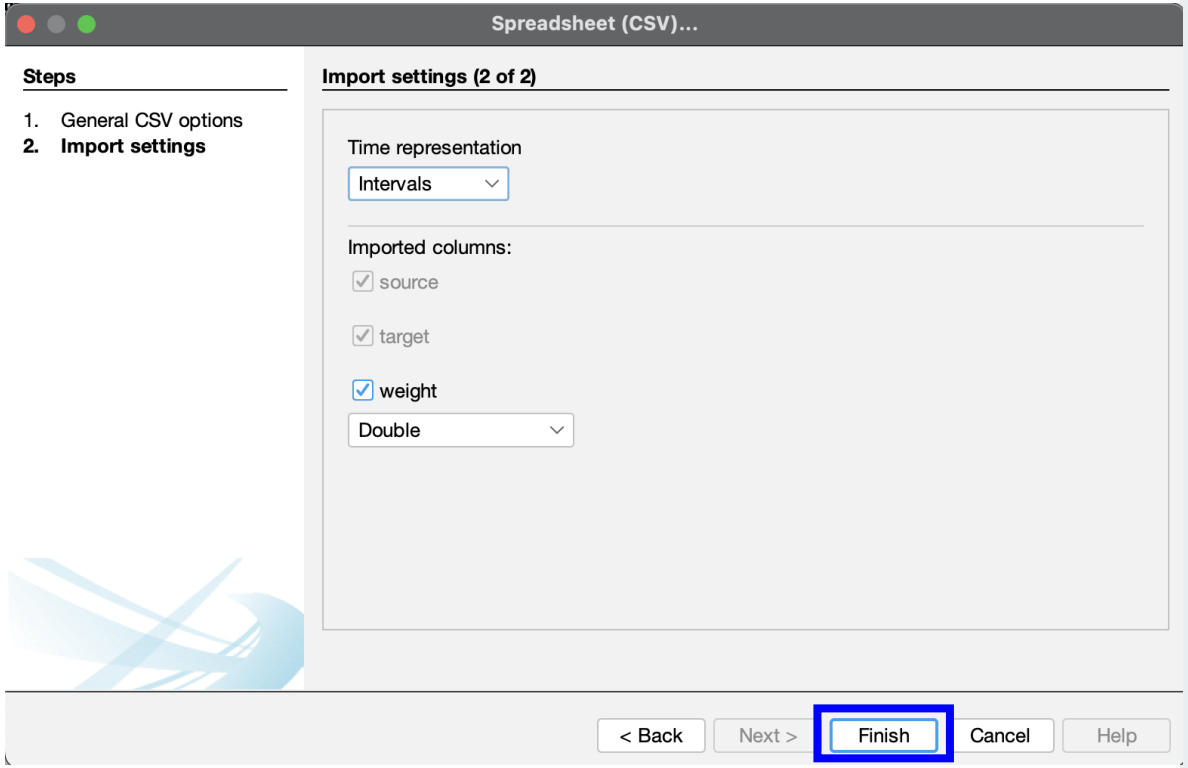


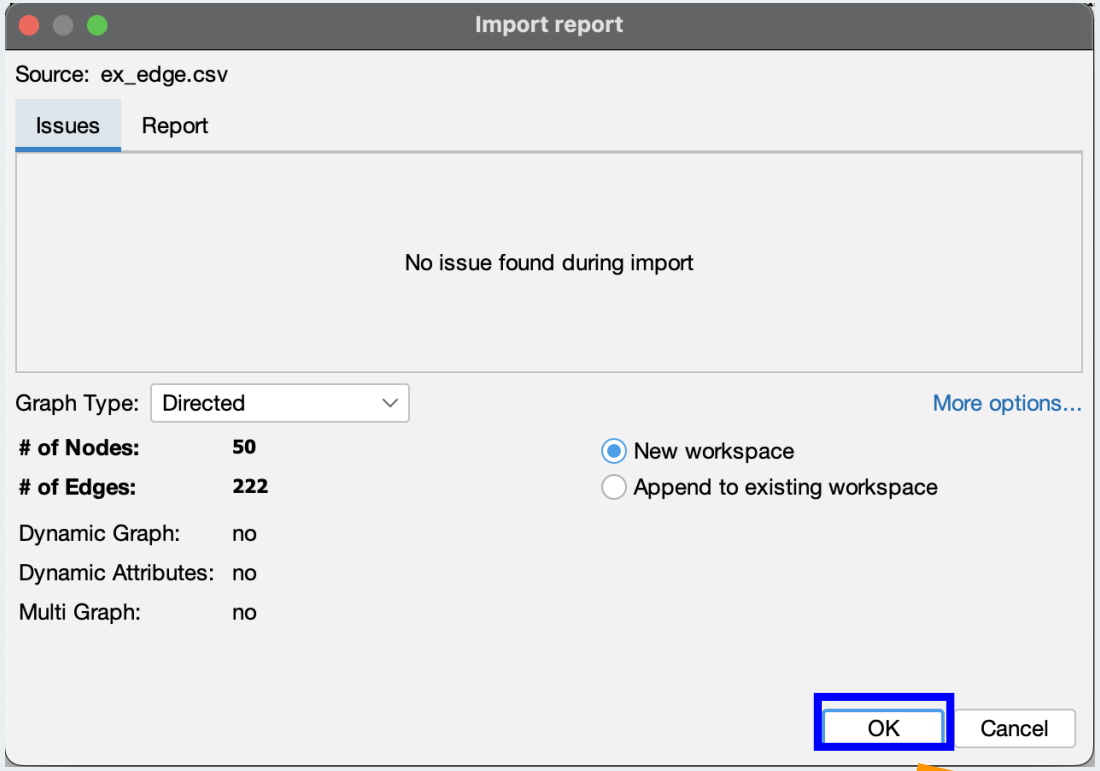


Task 5: Visualize with Gephi

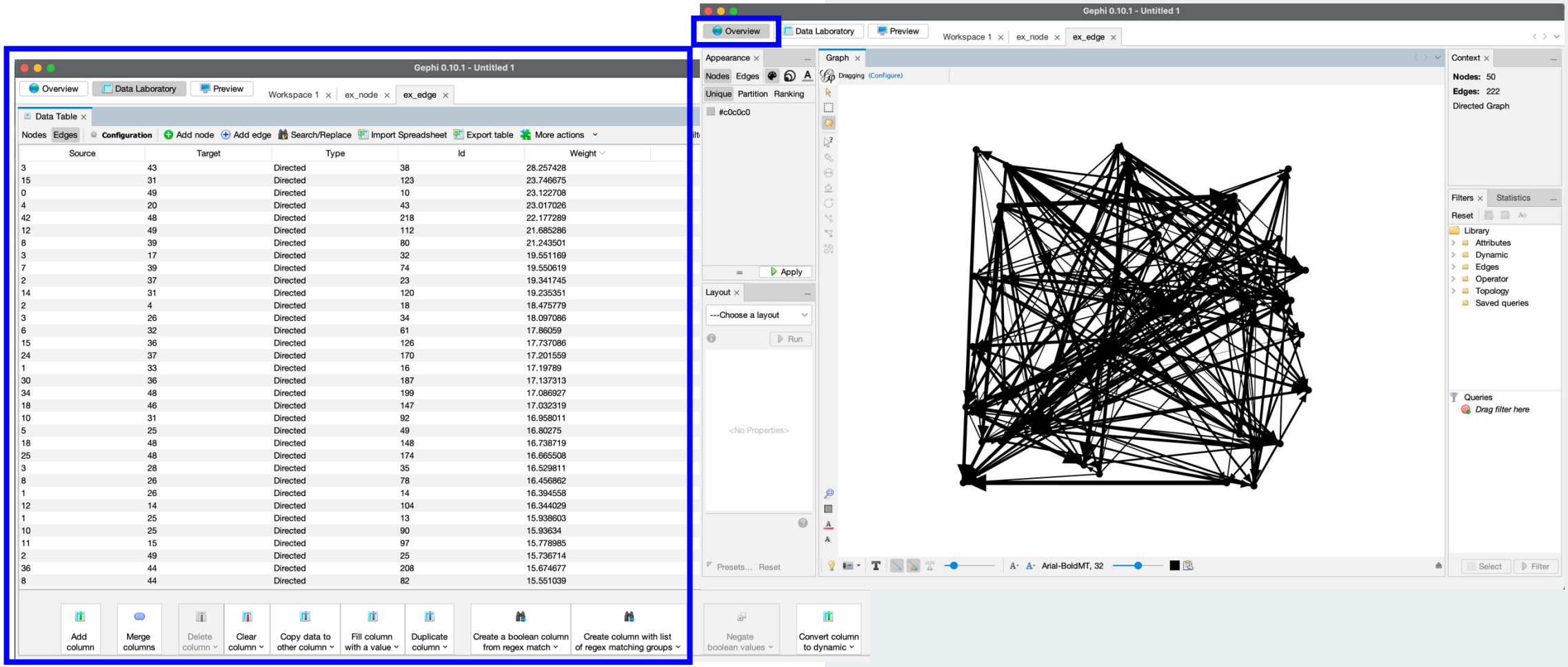


Task 5: Visualize with Gephi

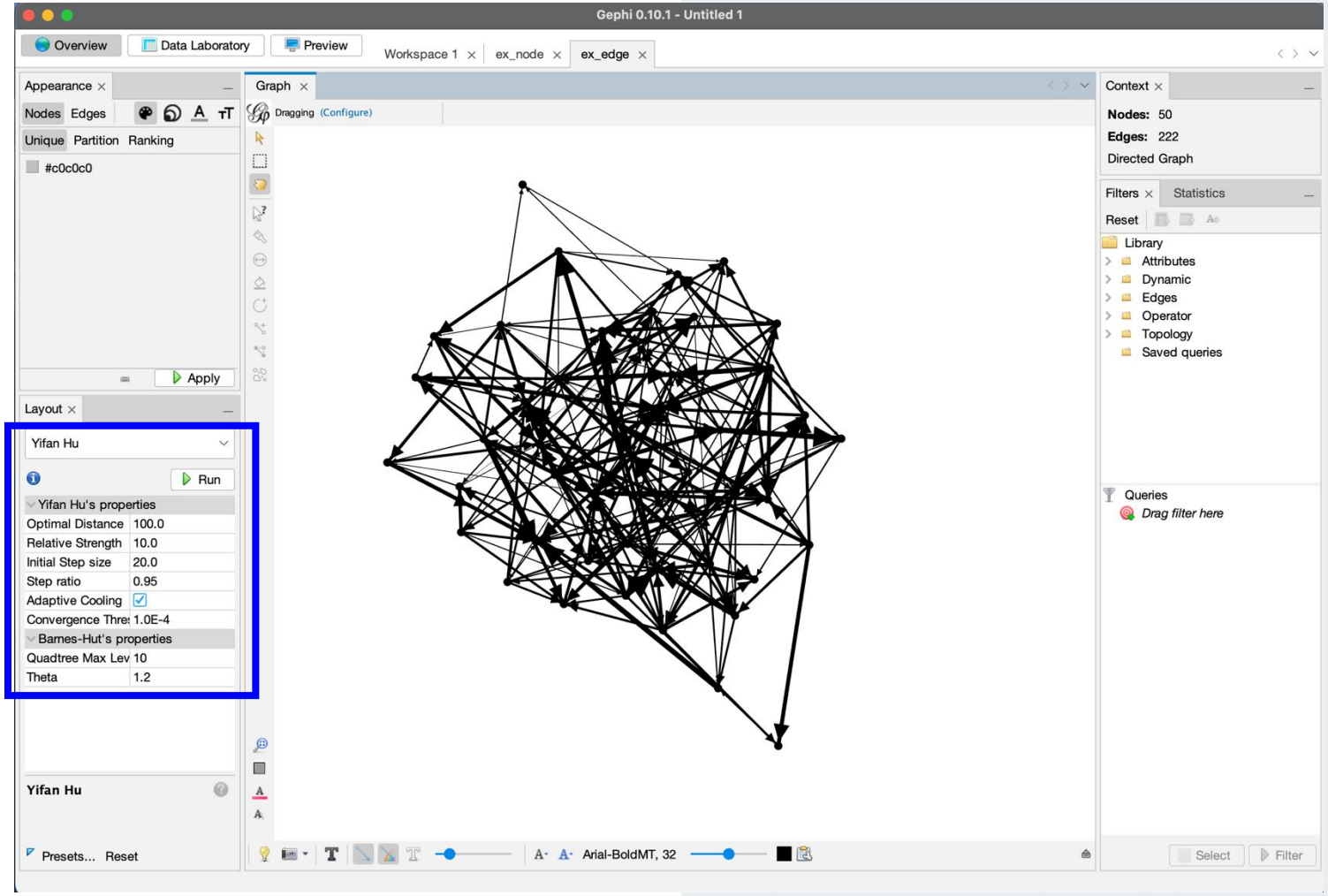




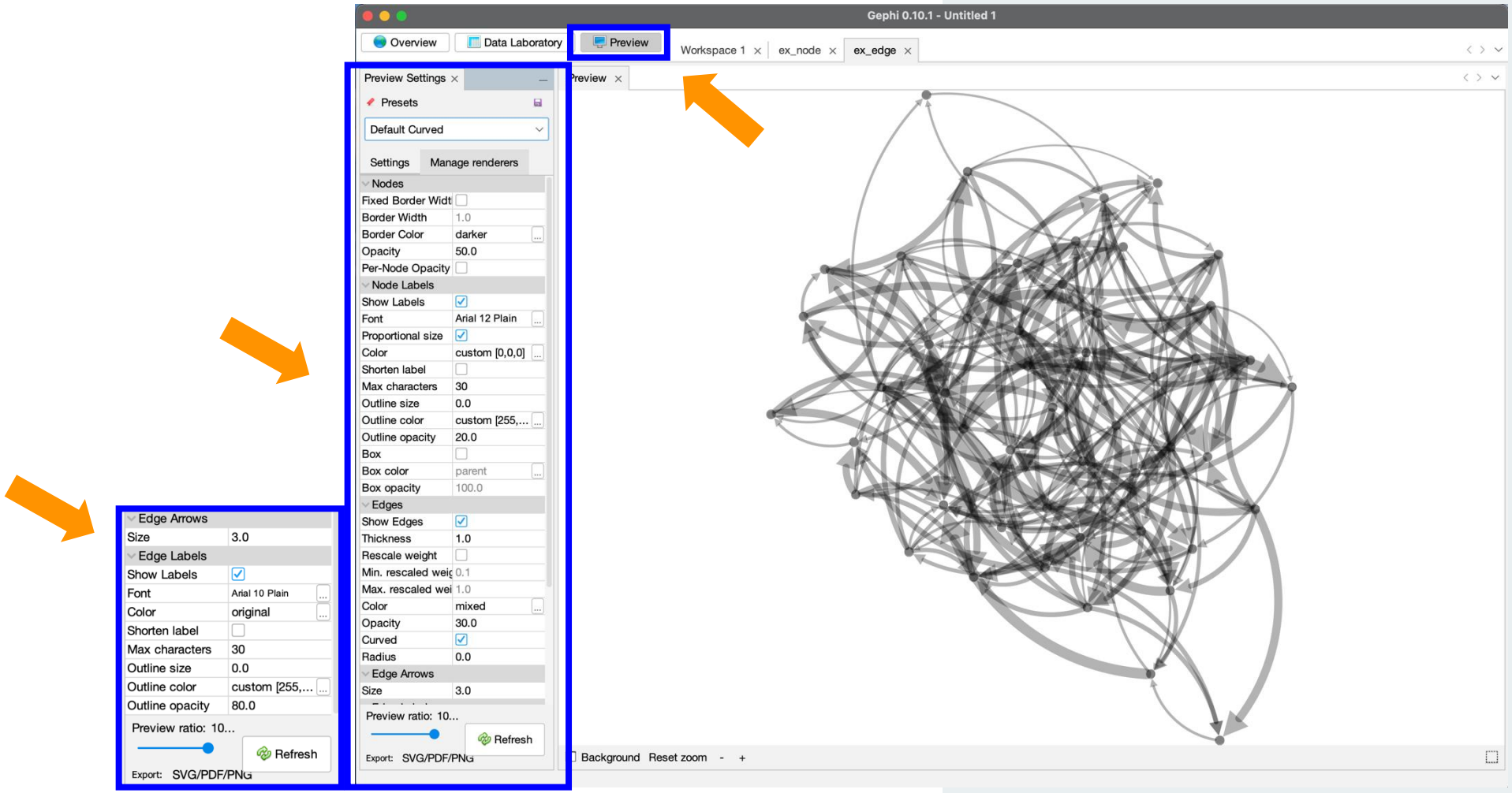
Task 5: Visualize with Gephi



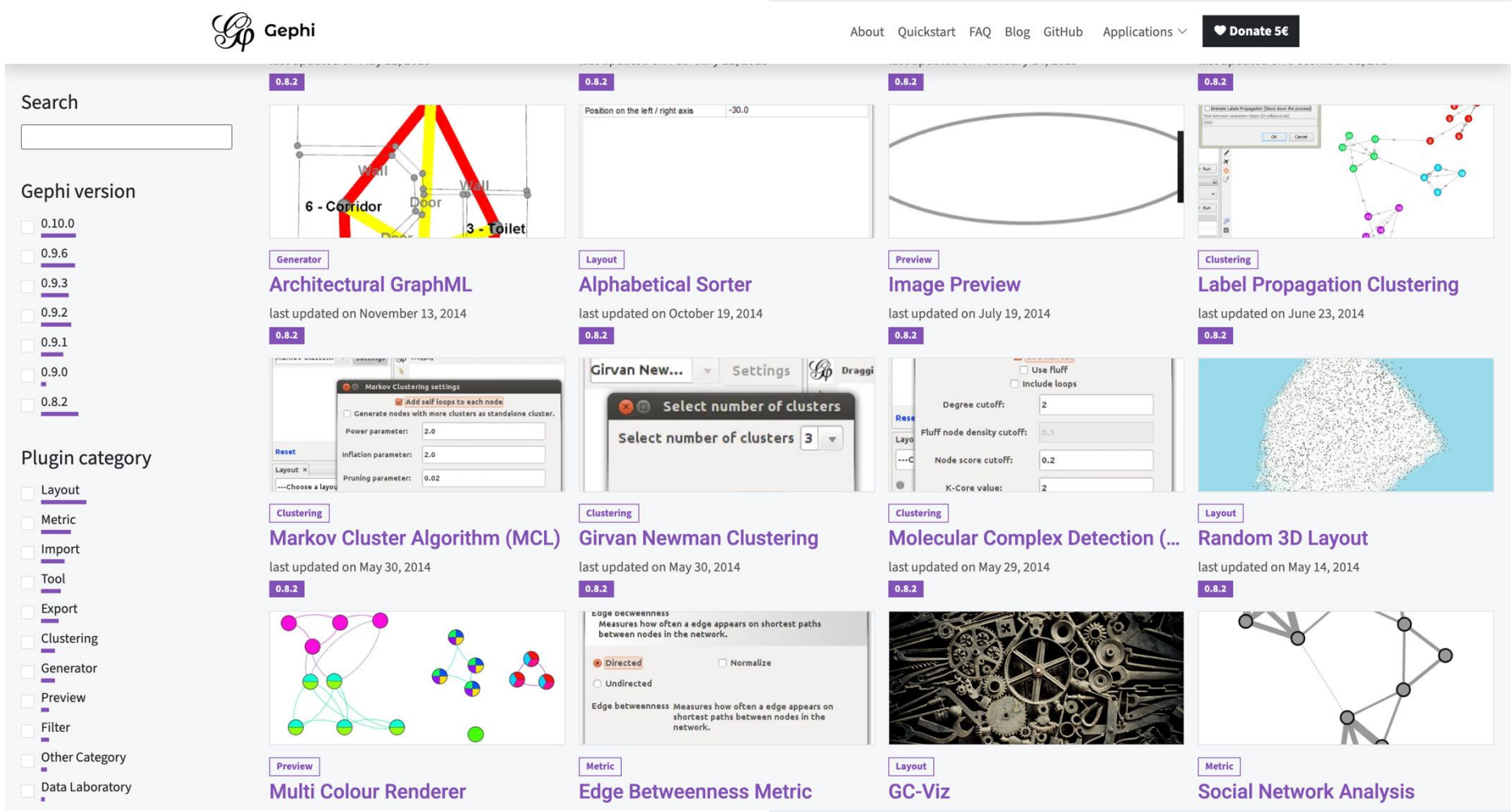
Task 5: Visualize with Gephi



Task 5: Visualize with Gephi

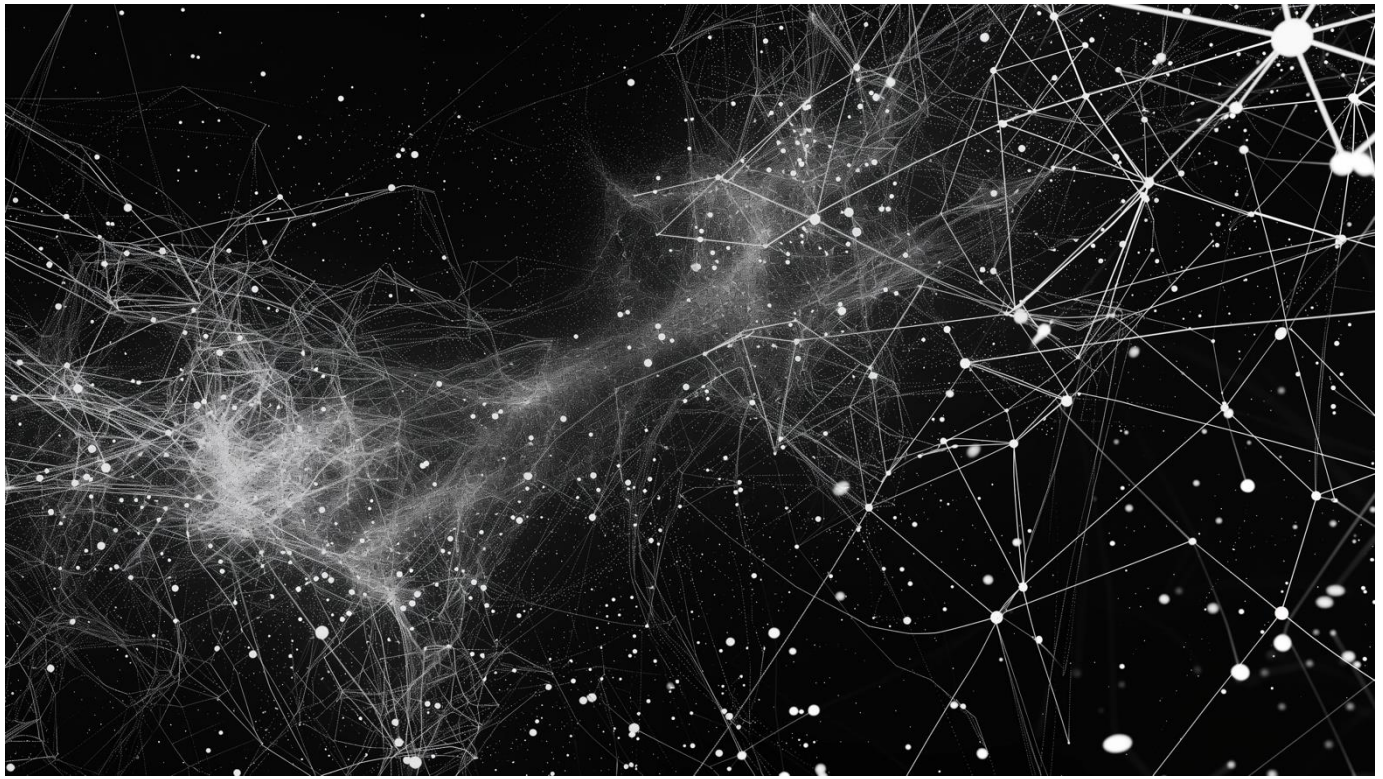


Task 5: Visualize with Gephi



The End

Thank you for your attention!



Email: chchan@ntnu.edu.tw
Website: toodou.github.io

Photo credit: midjourney