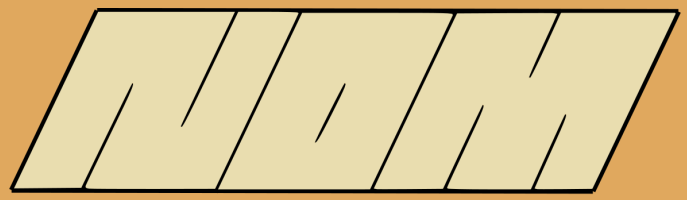
This project is funded and supported by:

THIS.Institute NIHR | Applied Research Collaboration South West Peninsula UNIVERSITY OF EXETER



Network-based Operational Modelling

Advanced network analysis session 2
Graph Visualisation

Dr Sean Manzi - University of Exeter, UK

Health Service Modelling Associates Programme 2021

Session structure

- How graph visualisations work
- Standardised algorithms and predefined coordinate systems
- Visualising graphs with NetworkX
- Visualising graphs with Cytoscape
- Visualising graphs with Gephi

Learning objectives

- Understand how graph algorithms seek to represent data
- Know how to visualise graphs using a variety of open source approaches
- Be able to customise a graph visualisation to improve its usefulness

How graph visualisation works

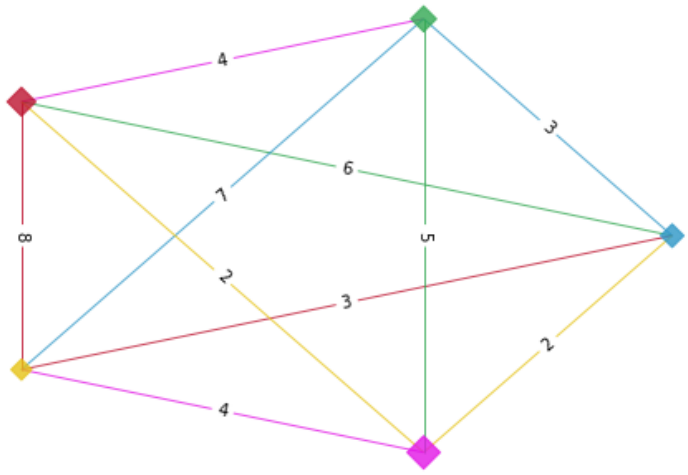
Network graphs provide a flexible way to visualise multiple attributes of the data. The three main components of the graph that can be customised to represent different aspects of the data are:

- Nodes
- Edges
- Layout

How graph visualisation works

Node customisation

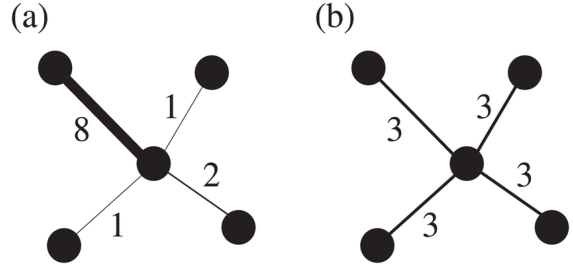
- Size - for continuous attributes
- Colour - for discrete attributes
- Shape - for discrete attributes
- Opacity - for continuous attributes
- Label - for discrete or continuous attributes



How graph visualisation works

Edge customisation

- Thickness - for continuous attributes
- Colour - for discrete attributes
- Label - for continuous or discrete attributes

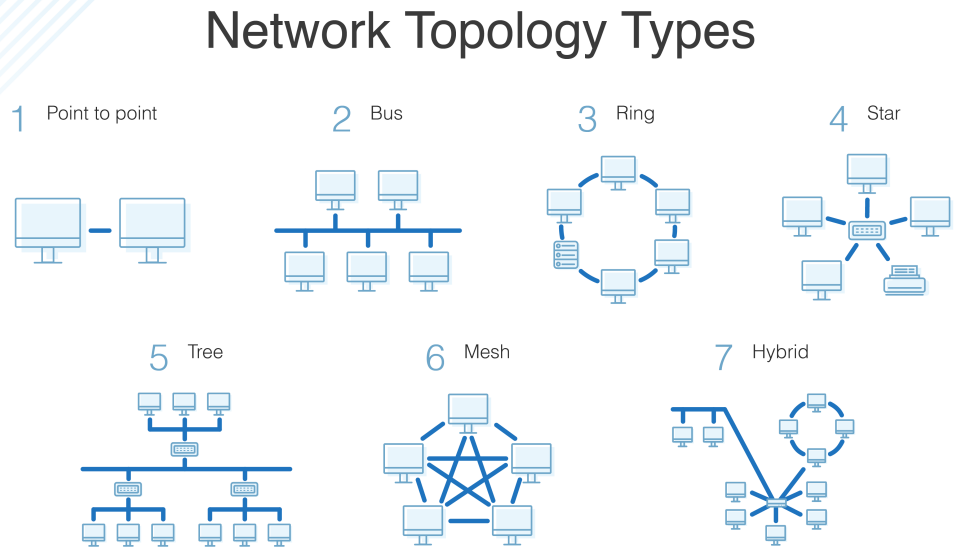


How graph visualisation works

Layout customisation

The basic layout of the graph is determined by the structure of the network, also called the network topology. Some common network structures are:

- Heirarchical
- Tree/Forest
- Ring
- Mesh
- Star
- Bus



How graph visualisation works

Simple and highly structured systems might conform to a simple topology. Complex systems are often a hybrid of two or more topologies making them more difficult to visualise. There are algorithms that help to visualise complex networks with non-standard topologies.

How graph visualisation works

NetworkX layouts

- Circular layout - nodes positioned in a circle
- Kamada kawai layout - nodes positioned using a path-length cost-function
- Random layout - nodes positioned uniformly at random
- Spring layout/Fruchterman-Reingold force directed layout - a physics based algorithm
- Spectral layout - nodes are positioned using the eigenvectors of the Laplacian matrix
- Spiral layout - nodes positioned in a spiral

For detailed layout documentation see https://networkx.org/documentation/stable/reference/drawing.html

How graph visualisation works

Cytoscape standard layouts

- Preset - nodes positioned by pre-determined coordinates
- Random - nodes are randomly positioned
- Grid - nodes are positioned in a square or rectangular grid
- Circle layout - nodes positioned in a circle
- Concentric - nodes are positioned in multiple concentric circles
- Breadthfirst - nodes are positioned in a heirarchical tree structure
- Cose - a physics based spring simulation

For detailed layout information see
https://dash.plotly.com/cytoscape/reference and
https://js.cytoscape.org/#layouts

How graph visualisation works

Cytoscape external layouts
Requires the line 'cyto.load_extra_layouts()'

- Cose-bilkent - extended cose layout for nested structures
- Cola - force directed physics layout with good generic applicability
- Euler - force directed physics layout with good generic applicability
- Spread - force directed physics layout aiming for an even spread
- Dagre - for use with directed acyclic graphs and trees
- Klay - discrete layout algorithm with good generic applicability

For detailed layout information see
https://dash.plotly.com/cytoscape/reference and
https://js.cytoscape.org/#layouts

Visualising graphs with NetworkX

NetworkX

```
pos = nx.'layout'('graph_object')
draw_networkx('graph_object', 'position_dict')
or
draw_networkx('graph_object', nx.'layout', **kwds)
```

Visualising graphs with NetworkX

NetworkX key word arguments (kwds/kwargs)

- arrows, arrowstyle, arrowsize
- with_labels, node_size, node_color, node_shape, cmap
- width, edge_color, edge_cmap, style
- font_size, font_color, font_weight, font_family

NetworkX example

Let's look at a simple example of producing a network visualisation in NetworkX. Open the 'networkx_vis_example.py' file.
The code breaks down as:

- Create some data
- Function to create the graph
- Define the layout
- Define the node and edge attributes: note the number of values for each attribute
- Draw the plot

Visualising graphs with Plotly Cytoscape

Plotly Dash is a python framework for building web applications
Cytoscape is a powerful network graph library for creating interactive graphs within web applications
The two examples that we looked at in the introduction to network analysis were both built using Dash and Cytoscape
We will now look at how to create a simple Dash application that contains a Cytoscape network graph

The structure of a Dash application

A simple Dash application is constructed from six core sections:

- Python library imports
- Persistent local variable creation
- Application initialisation
- The application layout
- Application callback functions - enables object interactivity
- The application run command

Python library imports

Like with any python script we need to import the required libraries at the beginning of our script
The Dash and Cytoscape libraries most commonly needed are:

```
import dash
import dash_html_components as html
import dash_core_components as dcc
import dash_cytoscape as cyto
import networkx as nx
from dash.dependencies import Input, Output, State
```

Persistent local variable creation

Any code that is run before the app initialisation section is only run once when the app is first loaded

This can be useful for simple applications that don't required generalised operability or where you have local variables that never change

In the example we will look at, all of the data upload and preprocessing is done when the app first loads. We also set the style options for the network graph in a dictionary known as a stylesheet (nomenclature from web development - css stylesheets).

Application initialisation

Creating the application object requires only two lines of code to setup an application object and a server object

```
app = dash.Dash(__name__)
server = app.server
```

The first line creates a Dash application object
The second line creates the server object linked to the application object

Application layout

The application layout contains the information on how the objects used in the application are to be structured and setup on app initialisation

This information is passed to the layout component of the application object

The layout is based on HTML components and syntax which has been adapted for use in python

```
app.layout = html.Div([
    html.Div(className='site-title text-center', children=[
        html.H1(className='padheader', children=[
            'Network of Thrones'
            ]),
        ]),
    html.Div(className='text-center', children=[
        html.P(children=[
            'Data from https://networkofthrones.wordpress.com/'
            ]),
        ]),
    html.Div(className='row', children=[
        html.Div(className='col-3', children=[
            html.Div(className='center-items', children=[
                html.Div(className='text-center', children=[
                    html.H3(children=['Select season']),
                    ]),
                html.Div(className='slider-container', children=[
                    dcc.Slider(className='slider', id='season-slider',
```

Application layout

It is in the layout section that we build up the layout structure using html components, Dash core components and additional component libraries such as cytoscape and dash bootstrap components

HTML component examples

- Div - Divider/container
- H1 - Headers, sizes 1-5
- P - Paragraph
- A - Hyperlink
- UL - Unordered list
- Img - Image

Dash core component examples

- Dropdown
- Input
- Slider
- Button
- Graph
- Checklist

Application callback functions

Callback functions come after the layout section. These functions take inputs from the app components defined in the layout and return outputs to other objects in the layout. The syntax is an adapted version of the standard python user defined function syntax.

```
@app.callback(Output('dynamic-net', 'elements'),
              [Input('hidden-elements-dict', 'children'),
               Input('graph-slider', 'value')])
def dynamic_network_select(data, selection):
    if data != None:
        el_dict = json.loads(data)
        selection = str(selection)
        selected = el_dict[selection]
        #elements = list(selected)

        return selected
    else:
        return list()
```

Application run command

The application run command does just that. It comes at the end of the script and runs the app server in the development environment

```
if __name__ == "__main__":
    app.run_server(debug=False)
```

The Cytoscape graph object

An example of a Cytoscape graph object in the app layout

```
app.layout = html.Div(children=[
                cyto.Cytoscape(
                    id='cyto-graph',
                    className='net-obj',
                    elements=elements,
                    style={'width':'80%', 'height':'600px'},
                    layout={'name': 'cose',
                            'padding': '200',
                            'nodeRepulsion': '7000',
                            'gravityRange': '6.0',
                            'nestingFactor': '0.8',
                            'edgeElasticity': '50',
                            'idealEdgeLength': '200',
                            'nodeDimensionsIncludeLabels': 'true',
                            'numIter': '6000',
                           },
                    stylesheet=default_stylesheet
                    )
                ])
```

The Cytoscape graph object

The main Cytoscape graph objects arguments can be summarised as:

- id = str - a unique identifier within the app layout
- className = str - a CSS class with standardised styling options
- elements = list of dicts - nodes and edge data
- style = dict - style options alternately placed in the stylesheet
- layout = dict - graph layout name and the layout attributes
- stylesheet = dict - node and edge style options

The Cytoscape graph object

An example of a stylesheet for a Cytoscape graph

```
default_stylesheet=[
            {'selector': 'node',
                'style': {
                        'width': '20',
                        'height': '20',
                        'background-color': 'blue',
                        'content': 'data(label)',
                        'font-size': '40px',
                        'text-outline-color': 'white',
                        'text-outline-opacity': '1',
                        'text-outline-width': '8px',
                    }
                },
            {'selector': 'edge',
                'style': {
                        'line-color': 'black'
                    }
                }
        ]
```

A simple cytoscape visualisation

Let's now have a look at a simple Dash app displaying a network graph made using cytoscape

Open the 'cytoscape_vis_simple.py' file. This example imports two csv files for the node and edge data so you will need to set your working directory accordingly.

A more advanced cytoscape visualisation

Now let's look at a more advanced version of the simple Dash app. This app not only displays the network graph made using cytoscape but also incorporates an analysis and visualises this as node size and colour. This example uses a callback to enable interactivity in the app.

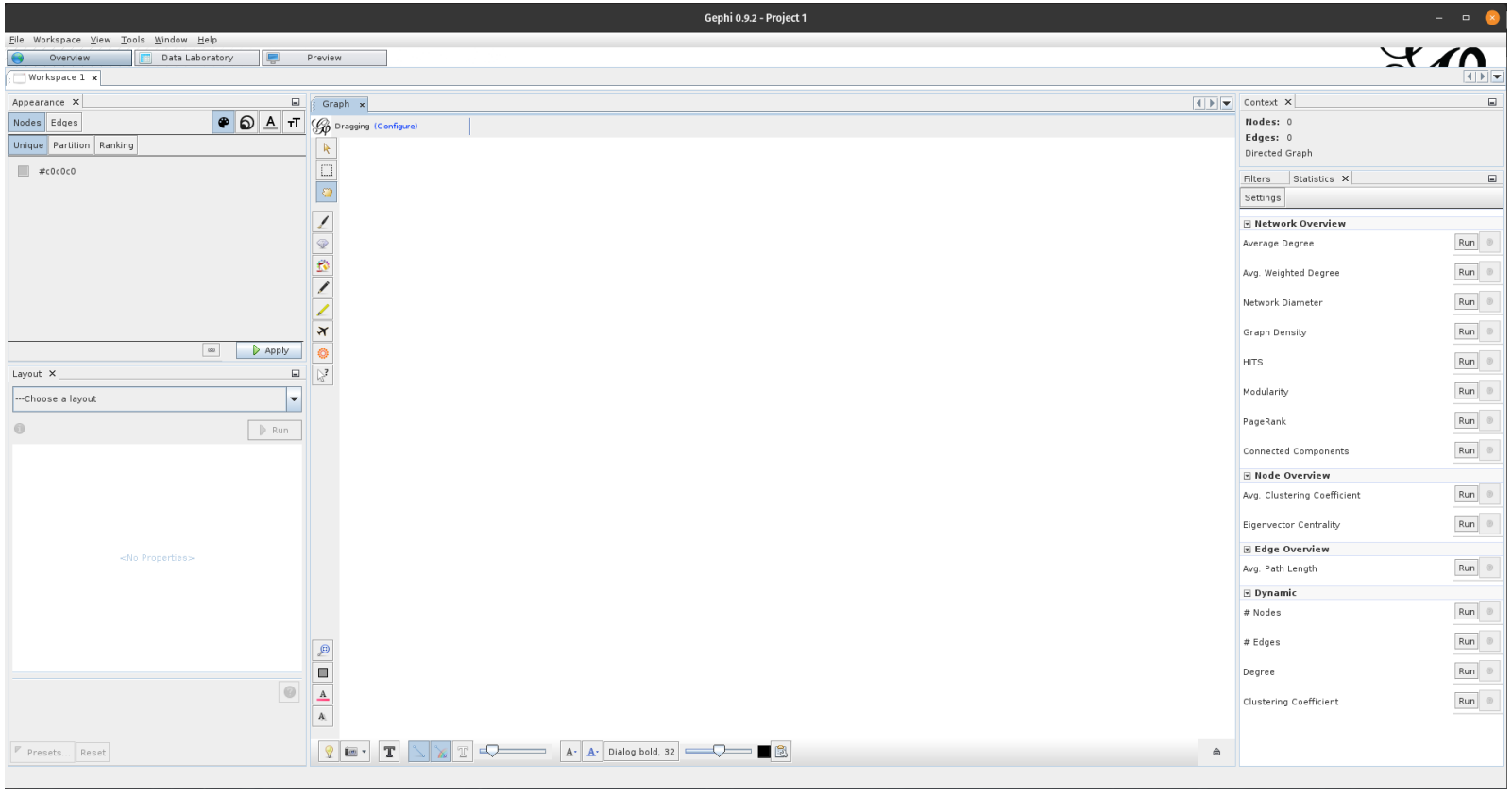
Open the 'cytoscape_vis_advanced.py' file. This example also imports the same two csv files for the node and edge data so you will need to set your working directory accordingly.

Visualising graphs with Gephi

- Gephi is a piece of software developed specifically for network visualisation and analysis
- It is the best visualisation engine for producing great looking network graphs
- The built in analysis functionality includes some useful graph metrics
- It is most effectively used when the analysis is conducted with NetworkX and any node and edge metrics are attached to the node and edge lists, exported to CSV and imported into Gephi to produce specific visualisations for presentations and publications

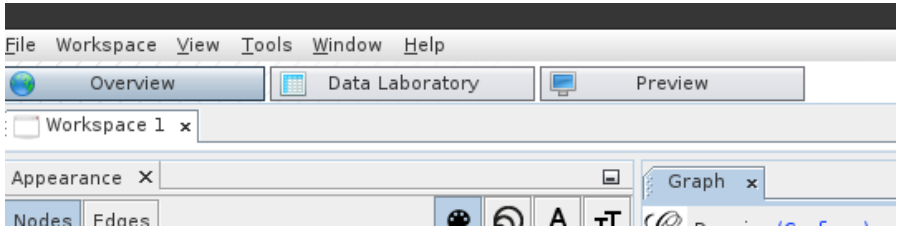
Gephi orientation - Openning screen

Openning screen



Gephi orientation - Main work areas

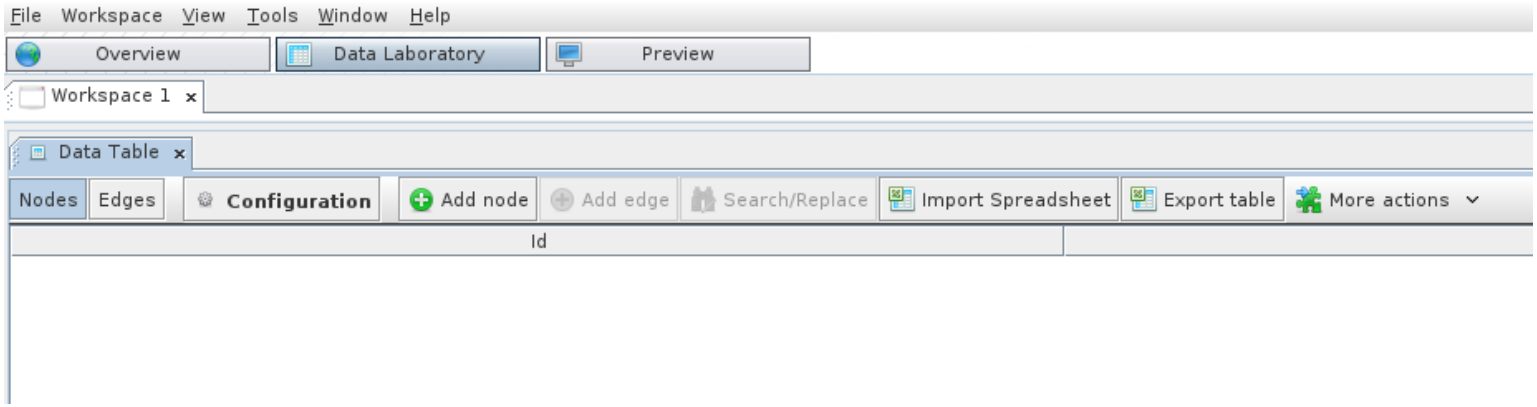
Three main screens: overview, Data Laboratory and Preview



- Overview: Layout, appearance and analysis
- Data Laboratory: Enter or upload data for visualisation
- Preview: refined appearance parameters and image export

Gephi orientation - Data laboratory

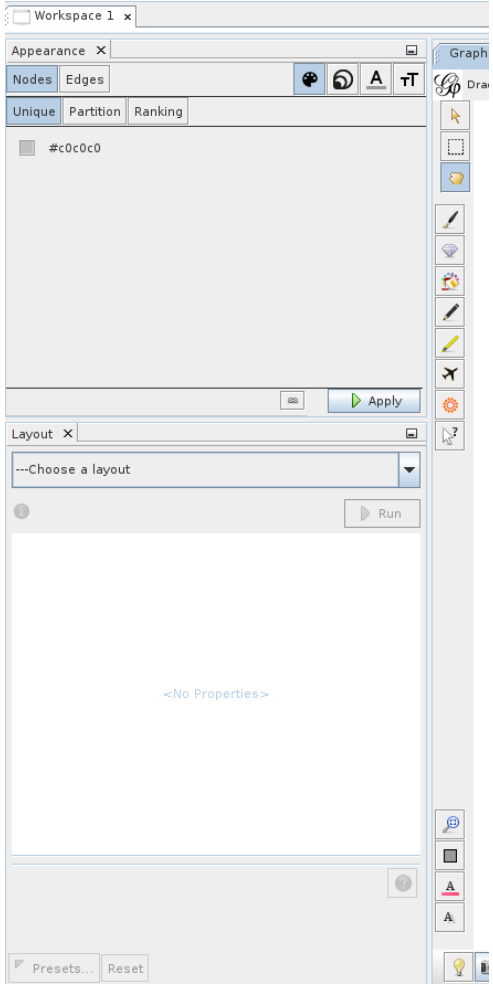
Data laboratory data import



- Separate node and edge data entry
- Enter data manually or import spreadsheet CSV, xls, xlsx
- Nodes: Id and Label columns required
- Edges: Source, Target, Type (directed or undirected) and Id required
- Edge source and target must match node Id

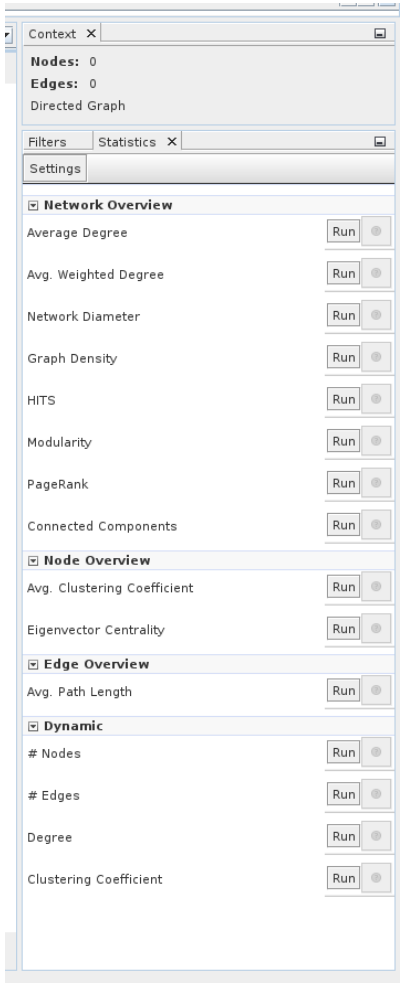
Gephi orientation - Appearance and layout

- Appearance
  - Node and edge formatting
  - Colour, shape, size, text
  - User determined or data determined
- Layout
  - Node position algorithms
  - Each algorithm has its own properties
  - One or more algorithms can be used in combination



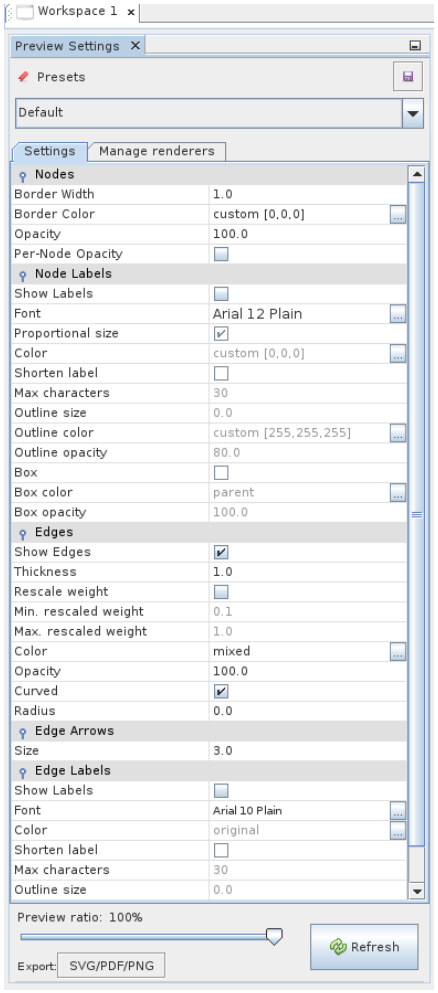
Gephi orientation - Graph metrics

- Network overview
  - Metrics describing the structure of the whole network
  - e.g. average degree, graph density, modularity
- Node overview
  - Average clustering coefficient - grouping across all nodes
  - Eigenvector centrality - relative importance for connectivity
- Edge overview
  - Average path length



Gephi orientation - Preview

- Detailed visual formatting parameters
- Adjust for nodes, node labels, edges, edge arrows and edge labels
- Takes the input from the overview view and allows refinement
- Need to use the refresh button to update the preview view when changes made to overview
- Export the visualisation view in SVG, PDF and PNG formats
- Presets can be established for repetition



Task time

1. Create a graph of the got_s08 dataset using Gephi
   - Add node labels
   - Colour nodes by modularity class
   - Edge thickness based on weight
   - Determine an appropriate layout
2. Create a graph of the got_s08 dataset using holoviews
   - Adapt the earlier example and use the documentation to see if you can make other changes to the visualisation

Resources

- https://gephi.org/
- https://dash.plotly.com/cytoscape
- https://dash.plotly.com/introduction
- https://networkx.org/documentation/stable/index.html

Thank you

Thank you for paying attention
Hope you enjoyed the session
Checkout https://www.project-nom.com for more information and training on the use of network-based operational modelling for whole system modelling in healthcare