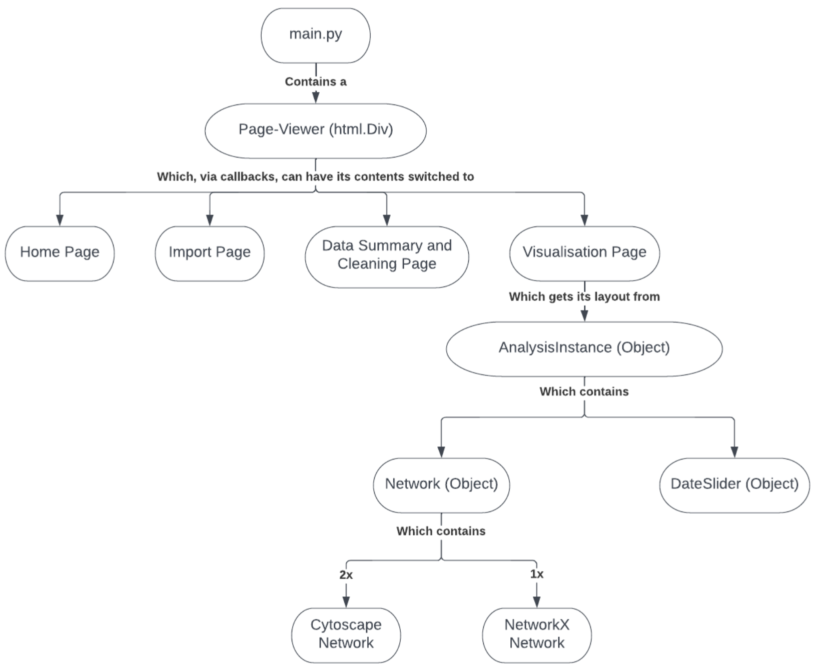
Documentation

App Files:

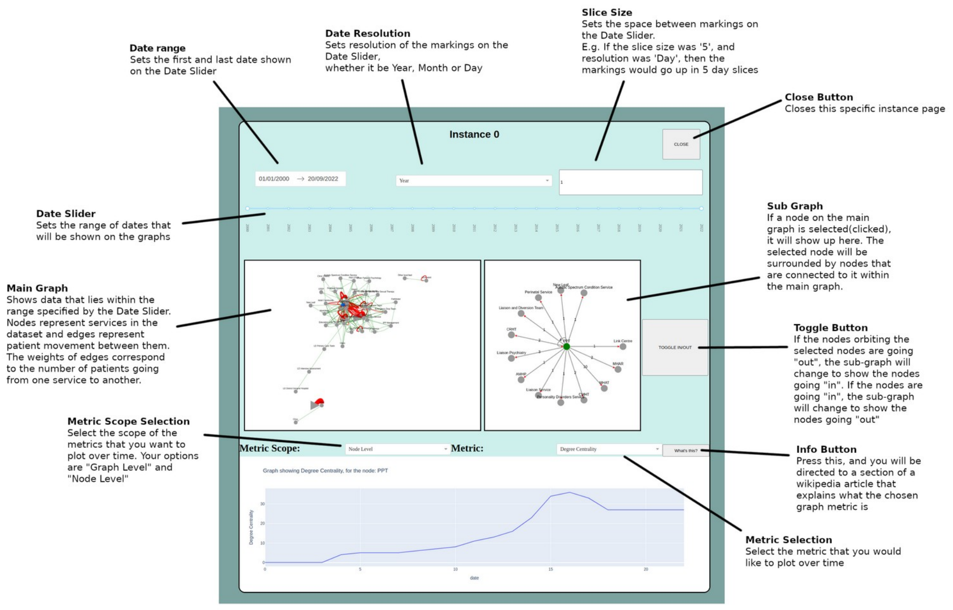
- main.py
- AnalysisInstance.py
- NetworkCreation.py
- DateSlider.py

There is also a folder called "uploaded_files", which is the location that imported files are contained

App Structure



The main part of the program is self contained within the AnalyisInstance class. Within the class, is code to create this layout:



Date range
Sets the first and last date shown on the Date Slider

Date Resolution
Sets resolution of the markings on the Date Slider,
whether it be Year, Month or Day

Slice Size
Sets the space between markings on the Date Slider.
E.g. If the slice size was '5', and resolution was 'Day', then the markings would go up in 5 day slices

Close Button
Closes this specific instance page

Date Slider
Sets the range of dates that will be shown on the graphs

Sub Graph
If a node on the main graph is selected(clicked), it will show up here. The selected node will be surrounded by nodes that are connected to it within the main graph.

Main Graph
Shows data that lies within the range specified by the Date Slider. Nodes represent services in the dataset and edges represent patient movement between them. The weights of edges correspond to the number of patients going from one service to another.

Toggle Button
If the nodes orbiting the selected nodes are going "out", the sub-graph will change to show the nodes going "in". If the nodes are going "in", the sub-graph will change to show the nodes going "out"

Metric Scope Selection
Select the scope of the metrics that you want to plot over time. Your options are "Graph Level" and "Node Level"

Info Button
Press this, and you will be directed to a section of a wikipedia article that explains what the chosen graph metric is

Metric Selection
Select the metric that you would like to plot over time

The AnalysisInstance class contains a DataSlider object, which allows the user to choose a range of dates using a dcc.RangeSlider. The AnalysisInstance class also contains a Network object, which deals with everything to do with networks. Within the Network class, there are 2 cytoscape graphs: one “main” graph, and one “sub” graph for displaying information for a specific node. The class also contains a NetworkX graph, for use in calculating graph metrics.

Multiple Pages

This web application allows users to switch to different pages. Although dash does provide functionality for this, I have chosen not to utilise it. The way that multiple pages works in this application is:

- main.py contains the app layout
- There are multiple (dcc.Link)s at the top of the page which act as the navigation bar
- Within the app layout there is an html.Div component with ID, “page-viewer”
- When a link on the navigation bar is pressed, the url changes
- A callback “listens” to the change in url, and gets the required page from a function
- The contents of “page-viewer” is modified with the return value of the function

The reasoning behind this approach is to make sharing information between pages easy, as they are all contained within the same main.py file.

The app has 4 pages:

- Home page
    - This is just a welcome page. It could be used to give the user a brief overview of how to use the application
- Importing page
    - This page allows the user to import a dataset
- Data summary and cleaning page
    - This page gives the user a summary of the dataset that they have imported. It prompts the user if there is a problem with the data
- Data visualisation page
    - This page is used for visualising the data within a network graph, and for plotting graph metrics over time

NetworkCreation.py

Classes:

Network

The Network contains code to initialise and manage cytoscape and networkX graphing. The class contains 2 cytoscape graphs (one for the "main" view and another for the "selected node" view), as well as a networkX graph.

Initialisation:

- **__init__()**
    - DESCRIPTION: Does nothing but create a Network object. Use the initialise() method to set up the Network

Attributes:

- PRIVATE
    - **__dataset**: pd.DataFrame
    - **__services**: list
    - **__cytoscape_nodes**: list of dicts
    - **__cytoscape_edges**: list of dicts
    - **__networkx_graph**: NetworkX object
    - **__adjacency_matrix**: 2x2 array with elements in the form of lists of tuples
    - **__selected_node**: str
- PUBLIC
    -

Methods:

- PRIVATE
    - **__get_services()**
        - DESCRIPTION: Gets the services from the imported dataset
        - RETURN: list of services with no duplicate services within it
    - **__create_adjacency_matrix()**
        - DESCRIPTION: Initialises a 2x2 array, where the elements are in the form of a list of tuples. The width and height of the array is equal to the number of services within the dataset. The tuples in the array represent a movement from one service to another. The tuple is structured as such:
          (start date, end date)
    - **__create_networkx_nodes_and_edges()**
        - DESCRIPTION: Initialises a directed networkX graph. Adds the nodes and edges from the cytoscape graph, putting them in a format that networkX can read.
- PUBLIC
    - **initialise(dataset: pd.DataFrame)**
        - DESCRIPTION: Sets up the cytoscape and networkX graphs
        - PARAMETERS: The imported dataset
    - **create_cytoscape_nodes_and_edges(all_nodes_and_edges: bool, start_date: date, end_date: date)**
        - DESCRIPTION:
        - PARAMETERS:
            - all_nodes_and_edges → Use all of the data from the dataset (True or False)
            - start_date → If all_nodes_and_edges is false, this is the first date in the range of data to be added to the cytoscape graph
            - end_date → If all_nodes_and_edges is false, this is the last date in the range of data to be added to the cytoscape graph
    - **get_cytoscape_nodes()**
        - DESCRIPTION: Returns the cytoscape nodes from the main graph
        - RETURN: list of dicts → This is a list of nodes in cytoscape format
    - **get_cytoscape_edges()**
        - DESCRIPTION: Returns the cytoscape edges from the main graph
        - RETURN: list of dicts → This is a list of edges in cytoscape format
    - **get_cytoscape_graph(graph_name: str)**
        - DESCRIPTION: Returns the current main cytoscape graph
        - PARAMETERS:

        - graph_name → This is used as the cytoscape main graph's ID, for callbacks
      - RETURN: cyto.Cytoscape
    - **get_specific_node_cytoscape_graph(graph_name: str, in_or_out: str)**
      - DESCRIPTION: Returns a cytoscape graph showing nodes connected to the selected node, along with the weights of the edges
      - PARAMETERS:
        - graph_name → This is used as the cytoscape sub-graphs ID, for use in callbacks
        - in_or_out → Whether the sub-graph should display nodes going into the selected node, or out of it
      - RETURN: cyto.Cytoscape
    - **get_specific_node_elements(in_or_out: str)**
      - DESCRIPTION: Retruns the nodes and edges within the sub-graph for selected node
      - PARAMETERS:
        - in_or_out → 'in' or 'out' string determines if the nodes shown on the sub-graph are going into the selected node, or out of it. String should be lowercase.
      - RETURN: list of dicts → The selected node is in the centre, with connected nodes orbiting it
    - **set_selected_node(selected_node: str)**
      - DESCRIPTION: Sets the node that has been selected
      - PARAMETERS:
        - selected_node → The label of the node that has been clicked
    - **iterate(start_date: date, end_date: date, slice_resolution: int, slice_size: int, metric_scope: int, metric: int)**
      - DESCRIPTION: Iterates over the range of dates specified, and returns a list of graph metrics for each time slice
      - PARAMETERS:
        - start_date → The first date in the range of dates
        - end_date → The last date in the range of dates
        - slice_resolution → The resolution of the time slices → 1-Year, 2-Month, 3-Day
        - slice_size →The size of each slice
        - metric_scope →The scope at which to view metrics → 1- Graph Level, 2 – Node Level
        - metric → The metric to plot
      - RETURN: list of values

Functions:

- **get_radial_layout(node_id, nodes_, centre, radius)**
  - DESCRIPTION: Returns the nodes entered, but in a radial layout where the selected node is in the centre and the other nodes orbit around it. Note – the only cytoscape elements that should be parsed into this function are nodes. This function is independent of edges (edges are not affected by this method)
  - PARAMETERS:
    - node_id →The label of the selected node, which will be placed in the centre
    - nodes_ → All of the nodes, including the selected node
    - centre → The centre position of the radial layout. In the form of a tuple (x, y) or a list [x, y]
    - radius → The radius of the radial layout
  - RETURN: list of dicts → The nodes arranged in a radial layout

DateSlider.py

The DataSlider class consists of a RangeSlider dash object, and methods to set up the RangeSlider to allow for selection of a range of dates.

Classes:

DateSlider

Initialisation:

- **__init__(id: str, start_date: date, end_date: date, slice_size: int, slice_resolution: int)**
  - DESCRIPTION: Initialises the DateSlider object
  - PARAMETERS:
    - id → The ID of the RangeSlider object within the DateSlider class. Used for callbacks
    - start_date → The first date to be shown on the slider

        - end_date → The last date to be shown on the slider
        - slice_size → The difference in time between one mark on the slider and another. E.g. If the slice size was “5”, and the slice resolution was “day”, then the slider marks would go up in 5 day chunks.
        - slice_resolution → The resolution of the slider – Either 1-Year, 2-Month or 3-Day

Attributes:

- PRIVATE
    - **__start_date**: date
    - **__end_date**: date
    - **__slice_resolution**: int
    - **__slice_size**: int
    - **__id**: str
    - **__slider**: dcc.RangeSlider
    - **__marks**: dict
    - **__length**: int
- PUBLIC
    -

Methods:

- PRIVATE
    - **__create_new_dateslider()**
        - DESCRIPTION: Creates a new RangeSlider after properties of the DateSlider object have been modified
- PUBLIC
    - **update_slider(start_date: date, end_date: date, slice_size: int, slice_resolution: int)**
        - DESCRIPTION: Updates the slider properties
        - PARAMETERS:
            - start_date → The first date on the slider
            - end_date → The last date on the slider
            - slice_size → The size of the time slice
            - slice_resolution → 1-Year, 2-Month, 3-Day
    - **get_start_date()**
        - DESCRIPTION: Returns the first date on the slider
        - RETURN: date
    - **get_end_date()**
        - DESCRIPTION: Returns the last date on the slider
        - RETURN: date
    - **get_slice_resolution()**
        - DESCRIPTION: Returns the slice resolution
        - RETURN: int
    - **get_slice_size()**
        - DESCRIPTION: Returns the slice size
        - RETURN: int
    - **get_marks()**
        - DESCRIPTION: Returns the marks on the slider
        - RETURN: dict
    - **get_length()**
        - DESCRIPTION: Returns the number of marks on the slider
        - RETURN: int
    - **get_slider()**
        - DESCRIPTION: Returns the slider object
        - RETURN: dcc.RangeSlider
    - **get_date_at_pos(pos: int)**
        - DESCRIPTION: Gets the date at a certain position on the slider
        - PARAMETERS:
            - pos → index to get the date at
        - RETURN: date

AnalysisInstance.py

The AnalysisInstance class encapsulates all of the functionality for analysing data.

Classes:

AnalysisInstance

Initialisation:

- **__init__(self)**
    - DESCRIPTION: Empty __init__(self) → Does nothing but create an AnalysisInstance object. To set up the AnalysisInstance object, use the initialise() method

Attributes:

- PRIVATE
    - **__instance_id**: str
    - **__dataset**: pd.DataFrame
    - **__slider**: DateSlider
    - **__network**: Network
    - **__toggle_value**: str
    - **__selected_node**: str
    - **__slider_start_pos**: int
    - **__slider_end_pos**: int
    - **__selected_metric_scope**: int
    - **__selected_metric**: int
- PUBLIC
    -

Methods:

- PRIVATE
    -
- PUBLIC
    - **initialise(instance_id: str, dataset: pd.DataFrame)**
        - DESCRIPTION: Sets up the AnalysisInstance. Creates a DateSlider and Network object.
        - PARAMETERS:
            - instance_id → The instance ID. This will be used to set the ID of all the elements within the AnalysisInstance object. The ID is used for callback purposes
            - dataset → The imported dataset
    - **set_date_range(start_date: date, end_date: date)**
        - DESCRIPTION: Sets the first and last date on the date slider.
        - PARAMETERS:
            - start_date → The first date
            - end_date → The last date
    - **set_slice_resolution(slice_resolution: int)**
        - DESCRIPTION: Make the resolution of the slider either Year, Month or Day
        - PARAMETERS:
            - slice_resolution → The resolution → 1 = Year, 2 = Month, 3 = Day
    - **set_slice_size(slice_size: int)**
        - DESCRIPTION: Sets the size of the time slice on the slider to view as a graph
        - PARAMETERS:
            - slice_size → The size of the time slice
    - **set_start_pos(pos: int)**
        - DESCRIPTION: Sets the starting date
        - PARAMETERS:
            - pos → Starting position on slider
    - **set_end_pos(pos: int)**
        - DESCRIPTION: Sets the ending date
        - PARAMETERS:
            - pos → Ending position on slider
    - **set_selected_node(node: str)**
        - DESCRIPTION: Sets the selected node on the main graph
        - PARAMETERS:
            - node → The label of the selected node
    - **toggle()**

        - DESCRIPTION: Switches the toggle to “out” if it is already “in”, or “in” if it is already “out”
    - **set_metric_scope(value: int)**
        - DESCRIPTION: Sets the scope of the metrics, either “Graph Level” or “Node Level”
        - PARAMETERS:
            - value → The metric scope → 1 = Graph Level, 2 = Node Level
    - **set_metric(value: int)**
        - DESCRIPTION: Sets the metric to view
        - PARAMETERS:
            - value → The metric to view
            - If metric scope = 1:
                - 1 = number of nodes
                - 2 = number of edges
                - 3 = average degree
                - 4 = graph density
                - 5 = graph modularity
            - If metric scope = 2:
                - 1 = degree centrality
                - 2 = betweenness centrality
                - 3 = modularity centrality
                - 4 = eigenvector centrality
    - **get_slider()**
        - DESCRIPTION: Gets the current slider object held within the date slider
        - RETURN: The RangeSlider object within the DateSlider, within the AnalysisInstance
    - **get_main_graph_elements()**
        - DESCRIPTION: Gets the elements of the current main graph. Used to update the graph after making modifications to dates
        - RETURN: A list of dicts
    - **get_sub_graph_elements()**
        - DESCRIPTION: Gets the elements connected to the selected node and puts them in a radial layout
        - RETURN: A list of dicts
    - **get_plot(metric_name: str)**
        - DESCRIPTION: Gets a line graph showing the selected graph metric over time
        - PARAMETERS:
            - metric_name: The name of the metric. This will be displayed on the graph plot at the bottom of the instance
        - RETURN: px.line
    - **get_layout()**
        - DESCRIPTION: Gets the layout of the analysis instance to be displayed on screen
        - RETURN: html.Div() with layout inside

main.py

The main python script:

- Initialises dash
- Configures the dash uploader to put uploaded files into a folder
- Sets the app layout → to be modified later by callbacks
- Contains functions to summarise and clean data
- Contains functions to return the different pages of the app
- Deals with the switching of pages
- Deals with all callbacks
- Starts the app

Functions:

- **summarise_data(dataset: pd.DataFrame)**
    - DESCRIPTION: Collect a summary of info from the dataset. This summary includes:
        - number of rows
        - number of headers
        - header names

    - PARAMETERS:
        - dataset → The imported dataset
    - RETURN: A string of the summary
- **get_column_completeness(dataset: pd.DataFrame)**
    - DESCRIPTION: Gets the completeness of each column
    - PARAMETERS:
        - dataset → The imported dataset
    - RETURN: list of completeness' for each column
- **columns_complete(dataset: pd.DataFrame)**
    - DESCRIPTION: Checks if the columns of the dataset are complete
    - PARAMETERS:
        - dataset → The imported dataset
    - RETURN: bool value → True if columns are complete
- **headers_labelled_correctly(dataset: pd.DataFrame)**
    - DESCRIPTION: Checks if the headers of the dataset are labelled correctly. The headers should be:
        - service
        - id
        - referraldate
        - dischargedate
    - PARAMETERS:
        - dataset → The imported dataset
    - RETURN: A bool value indicating if the headers of the dataset are labelled correctly
- **id_formatted(dataset: pd.DataFrame)**
    - DESCRIPTION: Checks if the ID's are positive
    - PARAMETERS:
        - dataset → The imported dataset
    - RETURN: A bool value indicating if the ID's are all positive
- **date_time_formatted(dataset: pd.DataFrame)**
    - DESCRIPTION: Checks if the dates in the dataset are valid. Makes sure days, months and years are positive, and within their valid range (days should be between 1 and 31, months should be between 1 and 12)
    - PARAMETERS:
        - dataset → The imported dataset
    - RETURN: A bool value indicating if the dates are in the correct format
- **get_home_page()**
    - DESCRIPTION: Gets the html layout of the home page
    - RETURN: A html layout
- **get_import_page()**
    - DESCRIPTION: Gets the html layout of the import page
    - RETURN: A html layout
- **get_cleaning_page()**
    - DESCRIPTION: Gets the html layout of the data summary and cleaning page
    - RETURN: A html layout
- **get_visualisation_page()**
    - DESCRIPTION: Gets the html layout of the data visualisation page. This function gets the layout of the visualisation page from the AnalysisInstance objects
    - RETURN: A html layout
- **get_404_not_found_page()**
    - DESCRIPTION: Gets the html layout of the 404 not found page. This page is displayed when the user tries to visit a URL that does not exist
    - RETURN: A html layout

Callbacks:

- **change_page(pathname: str)**
    - DESCRIPTION: This callback listens to a change in URL, and changes the contents of the page-viewer html.Div
    - PARAMETERS:
        - pathname → The new URL
    - RETURN: The new page to put into the page-viewer
- **uploaded(status: du.UploadStatus)**

    - DESCRIPTON: This is a dash-uploader callback that puts the latest imported dataset into the global variable “file”
    - PARAMETERS:
        - status → du.UploadStatus → The upload status of the file
    - RETURN: Once the file has finished uploading, the function returns a string “THE FILE HAS BEEN UPLOADED”
- **analysis_instance_callbacks(start_date0, end_date0, date_resolution0, slice_size0, slider_values0, selected_node0, toggle0, metric_scope0, metric0, info_button0, current_slider0, current_main_graph_elements0, current_sub_graph_elements0, current_plot0, current_metric_options0, current_metric_value0)**
    - DESCRIPTION: Modifies contents of an AnalysisInstance object
    - PARAMETERS:
        - start_date0 → The start date for instance 0
        - end_date0 → The end date for instance 0
        - date_resolution0 →The date resolution for instance 0
        - slice_size0 →The slice size for instance 0
        - slider_values0 →The values of the RangeSlider for instance 0
        - selected_node0 →The clicked node on the main graph of instance 0
        - toggle0 →The toggle button for instance 0
        - metric_scope0 →The selected metric scope for instance 0
        - metric0 →The selected metric for instance 0
        - info_button0 →The info buttton for instance 0
        - current_slider0 →The current RangeSlider for instance 0
        - current_main_graph_elements0 →The current elements in the main graph of instance 0
        - current_sub_graph_elements0 → The current elements in the sub graph of instance 0
        - current_plot0 →The current plot in instance 0
        - current_metric_options0 →The current metric options of instance 0
        - current_metric_value0 → The current metric value of instance 0
    - RETURN:
        - The function returns multiple values:
            - A slider
            - Elements for the main graph
            - Elements for the sub graph
            - A plot
            - A list of metric options
            - A metric value

TODO

Doesn’t work / Not fully implemented

- Callback Updates
    - Currently, the graphs and the plot only update when the slider is clicked. So even if the user selects a new date range with the date picker and slice resolution dropdown box, nothing will update until the date slider has been clicked.
    - When the app first launches and a dataset is imported – the plot doesn’t show anything straight away, when it should. It should by default plot the number of nodes for the graph, over time. To get the plot to update you first need to select something else from the “metric” dropdown box, then change it back.
- Data cleaning
    - Some of the data cleaning functions exist, but are not yet used
    - The app currently assumes the dataset is clean, and all in the correct format
- Metrics
    - Modularity centrality is not yet implemented

    - Eigenvector centrality sometimes fails to work
    - Average degree does not work
    - Network modularity is not yet implemented
- Close button
    - The close button is not yet implemented
    - It should remove the chosen AnalysisInstance from the array of instances.
    - An add button should be added at the bottom of the page to add more instances
- Slice Size
    - The slice size functionality has not yet been added to the iterate() method within the Network class in NetworkCreation.py
    - At the moment, only a slice size of 1 works with the graph
- Analysis Instances
    - Only Instance 0 works at the moment, as it is the only instance with callbacks implemented

Could do with improvement

- Callbacks
    - Callbacks are messy in there current state. There is a single function that handles every single callbacks
    - Callbacks could probably be split up into multiple different functions
    - The idea of using instances means that every instance would have to have its own version of the callbacks. Every callback would need to be duplicated for however many instances that there are on screen.
    -
- Main graph layout
    - The current layout of the main graph is unclear. It would be worth experimenting with the different settings for the "cose" display layout. E.g. "gravity" or "node repulsion"
- Data Summary and Cleaning page
    - The app currently
- Code
    - Make sure that all methods that potentially could be private, are. For ease of development, programmers should mostly only be interfacing with public methods. Public methods should be clear in what they do.
    - Data cleaning code could be placed outside of main.py, in its own class possibly. This would help ensure encapsulation and make code cleaner and easier to debug.
- Speed
    - The code to ensure that the dataset is clean could be improved in terms of performance. Cleaning makes use of multiple methods that work in a similar way – loop through the entire dataset, and check for any anomalys in the data. This could almost definetely be done in a single loop, where every column is checked per row. This could improve speed, as well as clean up the code.

Would be good to add

- Multiple plots shown on one graph
- Dates on the x axis of the time series graph
- Better UI
    - Better colour scheming
    - Nicer looking layout, buttons, ect
- Options
    - Allow the user to change the layout of the graphs and plots
- Forecasting Data