# final_report_notebook

June 10, 2020

## 1 Final Report Notebook

*This is our main program for importing data, manipulating it, and exporting it to .CSV files used for creating a network graph of COVID infection. We chose to import data from a John's Hopkins dataset that we can derive infection rate of different U.S. counties from. In this project we are specifically looking at the state of New York*

[125]:
```
from IPython.display import Image

Image(filename="new-york-county-map.gif")
```

[125]:
```
<IPython.core.display.Image object>
```

[123]:
```
import pandas as pd
import matplotlib.pyplot as plt
import networkx as nx
import datetime as dt
import numpy as np
import math
from sklearn import linear_model
import joblib
import warnings
from IPython.display import display
from IPython.display import Video
from IPython.core.interactiveshell import InteractiveShell
```

*The method below is used to filter data from a Pandas dataframe based on the state parameter given. This is useful when looking to pull specific state data.*

[102]:
```
# useful for filtering data from a certain state or province
def filter_data_by_state(data, state):
    data = data[data['Province_State'] == state]
    return data[data['Admin2'] != 'Unassigned']
```

*The create_nodes_for_data method takes in a Pandas dataframe, and populates a Networkx graph with nodes created with county names.*

[103]:
```
# makes nodes in graph for each row in data
def create_nodes_for_data(data):
    G = nx.Graph()
    for index, row in data.iterrows():
        G.add_node(str(row.iloc[5]))
    return G
```

*The populate_data_into_nodes_state method is used to pull data from a Pandas dataframe and populate each node in the initialized Networkx graph 'G' with important data like latitude, longitude, time of first infection, and so on. The method will also assign each node a relative time, based on first infection.*

[104]:
```
# fills nodes with data and assign relative_time
def populate_data_into_nodes_state(G, data, logistic_curve=False,␣
↪pop_data=None, load_curves=False,
                                   max_iterations=10000):
    already_infected = set()
    logistic_curves = dict()
    time = 0
    start_date = dt.datetime(2020, 1, 22)
    current_date = dt.datetime(2020, 1, 22)
    end_date = dt.datetime(2020, 5, 27)  # change as more data comes in
    while current_date <= end_date:
        day_data = data[data[current_date.strftime('%#m/%#d/%y')] > 0]
        was_added = False
        for index, row in day_data.iterrows():
            county = str(row.iloc[5])
            if county not in already_infected:
                G.nodes[county]['relative_time'] = time
                G.nodes[county]['real_time'] = current_date.strftime('%#m/%#d/
↪%y')
                G.nodes[county]['lat'] = row.iloc[8]
                G.nodes[county]['long'] = row.iloc[9]
                G.nodes[county]['data'] = row.iloc[12:]
                if pop_data is not None:
                    pop = pop_data[pop_data['County'] == county]
                    if not pop.empty:
                        G.nodes[county]['pop'] = pop['Population']
                    else:
                        G.nodes[county]['pop'] = 2000  # give unknown counties␣
↪2000 population
                if logistic_curve:
                    if load_curves:
                        load_logistic_curve(county, logistic_curves)
                    else:
                        logistic_curves[county] = fit_logistic_curve(G.
↪nodes[county], max_iterations)
```

```
                already_infected.add(county)
                was_added = True
        if was_added:
            time += 1
        current_date += dt.timedelta(days=1)
    return time, logistic_curves
```

*The create_edges_for_graph_first_infection method is used to create edged between the nodes in a Networkx graph, based on the relative time of infection, and the distance between the two counties.*

[105]:
```
# uses relative_time field to determine where to place edges, distance can be␣
↪limited but by default is not
def create_edges_for_graph_first_infection(G, max_relative_time,␣
↪distance_limit):
    time = 0
    prev_nodes = []
    temp = []
    while time <= max_relative_time:
        for node in G.nodes(data=True):
            if 'relative_time' in node[1] and node[1]['relative_time'] == time:
                for prev_node in prev_nodes:
                    distance = calculate_distance(prev_node, node)
                    if distance <= distance_limit:
                        G.add_edge(prev_node[0], node[0], weight=distance)
                temp.append(node)
        prev_nodes = temp.copy()
        temp.clear()
        time += 1
```

*Much like the above method, the create_edges_for_graph_threshold_distance method checks if the number of infected in the county, with proportion to the distance to the next county, meets a certain specified threshold. If it does meet this threshold, the method will create an edge between the two counties.*

[106]:
```
# uses real_time and distance to determine where to place edges threshold and␣
↪distance_limit must be provided
def create_edges_for_graph_threshold_distance(G, threshold, distance_limit):
    start_date = dt.datetime(2020, 1, 22)
    current_date = dt.datetime(2020, 1, 22)
    end_date = dt.datetime(2020, 5, 27)
    prev_nodes = []
    while current_date <= end_date:
        for node in G.nodes(data=True):
            if 'real_time' in node[1] and node[1]['real_time'] == current_date.
↪strftime('%#m/%#d/%y'):
                for prev_node in prev_nodes:
                    distance = calculate_distance(prev_node, node)
```

```
                    day = int(current_date.strftime('%#j')) - int(start_date.
↪strftime('%#j'))
                    weight = prev_node[1]['data'][int(day)]/distance
                    if weight >= threshold and distance <= distance_limit:
                        G.add_edge(prev_node[0], node[0], weight=weight)
                prev_nodes.append(node)
        current_date += dt.timedelta(days=1)
```

*This method uses a logistic regression model from SciKit-Learn to predict a fitted logistic curve for each county, and then runs a simulation to create edges in the network.*

*This simulation is based off a radius of infection, calculated by the logistic regression model. If a county is within the infection radius distance of an already infected county, that county will also begin to become infected.*

[107]:
```
# uses regression to fit a logistic growth curve to the each county and then␣
↪run a simulation to create edges
def create_edges_for_graph_logistic_simulation(G, logistic_curves, time_limit,␣
↪radius_weight):
    time = np.zeros((1, 1), dtype=int)
    infected = []
    found_first = False
    while time[0][0] <= time_limit:
        if not found_first:
            for node in G.nodes(data=True):
                if node[0] in logistic_curves and logistic_curves[node[0]].
↪predict(time)[0] > 1:
                    infected.append(node)
                    found_first = True
                    break
        time[0][0] += 1
        if found_first:
            break
    while time[0][0] <= time_limit:
        for node1 in infected:
            radius = radius_weight * logistic_curves[node1[0]].predict(time)[0]␣
↪ # / float(node1[1]['pop'])
            for node2 in G.nodes(data=True):
                if node2 in infected:
                    continue
                if 'lat' in node2[1]:
                    distance = calculate_distance(node1, node2)
                    if distance < radius:
                        G.add_edge(node1[0], node2[0], weight=distance)
                        infected.append(node2)

        time[0][0] += 1
```

*The below method is key for creating our infection simulation. This will take in a node from the Networkx graph, and use the Scikit-Learn python library to create a logistic regression model to fit logistic curves to the infection data in that node. It will then return the logistic curve.*

*The logistic regression model will the node to fit a curve based on the data within that node. It will optimize the data to the logistic function:*

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

*where $x_0$ is the $x$ value of thei sigmoid midpoint, $L$ is the curves maximum value, and $k$ is the growth rate of the curve.*

[108]:
```
# uses sklearn to fit a logistic growth curve to the data in the node
def fit_logistic_curve(node, max_iterations):
    x = np.linspace(0, len(node['data'])-1, len(node['data']), dtype=int)
    y = np.zeros(len(x), dtype=int)
    for i in range(0, len(y)):
        y[i] = int(node['data'][i])
    x = x[:, np.newaxis]
    warnings.filterwarnings("ignore")
    clf = linear_model.LogisticRegression(C=1e5, max_iter=max_iterations)
    clf.fit(x, y)
    # # used to plot logistic curves
    # plt.figure(1, figsize=(4, 3))
    # # plt.clf()
    # x = np.linspace(0, 250, 251)
    # x = x[:, np.newaxis]
    # plt.plot(x, clf.predict(x))
    # plt.show()
    return clf
```

*This method is used to save the produced logistic curves to a certain directory. The default directory is logistic_curves, but this can be modified to save to different locations.*

[109]:
```
# saves logistic_curves to a directory named logistic_curves and a modifier can␣
 ↪be added for different save locations
def save_logistic_curves(logistic_curves, modifier=''):
    if modifier is not '':
        modifier = '_' + modifier
    for key in logistic_curves:
        joblib.dump(logistic_curves[key], 'logistic_curves%s/%s.pkl' %␣
 ↪(modifier, key))
```

*This method will load logistic curves from a certain directory. The default directory is logistic_curves, but this can be modified to load from different locations.*

[110]:
```
# loads logistic curves from a directory named logistic_curves and a modifier␣
 ↪can be added for different load locations
def load_logistic_curve(county, logistic_curves, modifier=''):
```

```
    if modifier is not '':
        modifier = '_' + modifier
    logistic_curves[county] = joblib.load('logistic_curves%s/%s.pkl' %␣
↪(modifier, county))
```

*This method will run previously defined methods to create a Networkx graph of the spread of Covid-19 based on first infection in a location and connects location which were infected next.*

[111]:
```
def create_graph_first_infected(data, distance_limit=30000):
    G = create_nodes_for_data(data)
    max_time = populate_data_into_nodes_state(G, data)[0]
    create_edges_for_graph_first_infection(G, max_time, distance_limit)
    return G
```

*This method will run previously defined methods in order to create a networkx graph of the spread Covid-19 based on a threshold and distance_limit. This method may be better on larger sample sizes so far locations are not infecting each other*

[112]:
```
def create_graph_infected_distance(data, threshold=2.5, distance_limit=75):
    G = create_nodes_for_data(data)
    populate_data_into_nodes_state(G, data)
    create_edges_for_graph_threshold_distance(G, threshold=threshold,␣
↪distance_limit=distance_limit)
    return G
```

*This method will run previously defined methods in order to create a networkx graph of the spread Covid-19 based on a Logistic simulation*

[113]:
```
def create_graph_logistic_simulation(data, time_limit=250, radius_weight=.5):
    G = create_nodes_for_data(data)
    pop_data = pd.read_csv('pop_ny_data_2020.csv')
    max_relative_time, logistic_curves = populate_data_into_nodes_state(G,␣
↪data, logistic_curve=True, pop_data=pop_data,
                                                                        ␣
↪max_iterations=50000)
    create_edges_for_graph_logistic_simulation(G, logistic_curves, time_limit,␣
↪radius_weight)
    save_logistic_curves(logistic_curves)
    return G
```

*This method will run previously defined methodsin order to create a networkx graph of the spread Covid-19 based on a logistic simulation loaded from saved logistic curves.*

[114]:
```
def create_graph_logistic_simulation_load_lc(data, time_limit=250,␣
↪radius_weight=.5):
    G = create_nodes_for_data(data)
    pop_data = pd.read_csv('pop_ny_data_2020.csv')
```

```
    max_relative_time, logistic_curves = populate_data_into_nodes_state(G,␣
↪data, logistic_curve=True, pop_data=pop_data,
                                                                        ␣
↪load_curves=True)
    create_edges_for_graph_logistic_simulation(G, logistic_curves, time_limit,␣
↪radius_weight)
    return G
```

*This method converts the networkx graph created to two csv files which can be imported into cytoscape.*

[115]:
```
def convert_to_csv(G):
    output_node =␣
↪'county,relative_time,real_time,lat,long,degree,degree_centrality,closeness_centrality\n'
    output_edge = 'source,target,distance\n'
    for node in G.nodes(data=True):
        if 'relative_time' in node[1]:
            output_node += '%s,%s,%s,%s,%s,%s,%s,%s\n' % (node[0],␣
↪node[1]['relative_time'], node[1]['real_time'],
                                                        node[1]['lat'],␣
↪node[1]['long'], node[1]['degree'],
                                                                      ␣
↪node[1]['degree_centrality'], node[1]['closeness_centrality'])
    for edge in G.edges(data=True):
        output_edge += '%s,%s,%s\n' % (edge[0], edge[1], edge[2]['weight'])

    return output_node, output_edge
```

*The convert_to_cyto_layout method is used to convert the cytoscape data and network csv files into csv files usable for the cytoscape coordinateLayout plugin.*

[116]:
```
def convert_to_cyto_layout(node_file_in, edge_file_in, node_file_out,␣
↪edge_file_out, scale=25):
    nodes_out_text = ''
    edges_out_text = ''
    nodes_in = pd.read_csv(node_file_in)
    cities = dict()
    i = 1
    for index, row in nodes_in.iterrows():
        cities[row.iloc[0]] = i
        nodes_out_text += '%s %s %s %s %s\n' % (i, str(row.iloc[0]).replace('␣
↪', '_'), '1', int(row.iloc[3])/scale,
                                                int(row.iloc[4])/scale)
        i += 1
    edges_in = pd.read_csv(edge_file_in)
    for index, row in edges_in.iterrows():
```

```
        edges_out_text += '%s %s %s\n' % (cities[row.iloc[0]], cities[row.
↪iloc[1]], row.iloc[2])

    with open(node_file_out, 'w') as f:
        f.write(nodes_out_text)
        f.close()

    with open(edge_file_out, 'w') as f:
        f.write(edges_out_text)
        f.close()
```

*These two methods are used to calculate the distance between two nodes. This calculation uses the latitude and longitude of the counties that the nodes represent.*

[117]:
```
# calculates the distance between two nodes
def calculate_distance(node1, node2):
    lat1 = node1[1]['lat']
    lat2 = node2[1]['lat']
    lon1 = node1[1]['long']
    lon2 = node2[1]['long']
    R = 3958.5  # radius of earth in miles
    dlat = deg_to_rad(lat2 - lat1)
    dlon = deg_to_rad(lon2 - lon1)
    a = (math.sin(dlat/2))**2 + math.cos(deg_to_rad(lat1)) * math.
↪cos(deg_to_rad(lat2)) * (math.sin(dlon/2))**2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    return math.floor(R * c)


# helper function for calculate distance
def deg_to_rad(deg):
    return deg * (math.pi / 180)
```

*This method is used to calculate useful statistics of nodes in the graph such as, centrality and degree.*

[118]:
```
def calculate_node_stats(G):
    for item in nx.degree_centrality(G).items():
        G.nodes[item[0]]['degree_centrality'] = item[1]
    for item in nx.closeness_centrality(G).items():
        G.nodes[item[0]]['closeness_centrality'] = item[1]
    for item in G.degree():
        G.nodes[item[0]]['degree'] = item[1]
```

*This method is used to write data from the Networkx graph to a csv format, and will then write it to a local directory file.*

[119]:
```
def write_csv(filename, G):
    csv = convert_to_csv(G)
    with open('%s_network.csv' % filename, 'w') as f:
        f.write(csv[1])
        f.close()

    with open('%s_data.csv' % filename, 'w') as f:
        f.write(csv[0])
        f.close()
```

*The main method of our program. This will read in the John's Hopkins csv data into a Pandas dataframe, will filter that dateframe to the state of New York, and will use the methods above to create various visualizations.*

[120]:
```
def main():
    data = pd.read_csv('time_series_covid19_confirmed_US.csv')

    new_york = filter_data_by_state(data, 'New York')
    display(new_york)

    # visualizes data in graph vs time
    dates = list(range(0, len(data.columns[12:].values)))
    plt.figure()
    for index, row in new_york.iterrows():
        plt.plot(dates, row.iloc[12:], label=row.iloc[5])
    plt.title(str.title('NY Counties cases vs. time'))
    plt.legend(framealpha=2, frameon=True, ncol=3, loc='upper left')
    plt.show()

    # creates graph of first infection
    G = create_graph_first_infected(new_york)
    calculate_node_stats(G)
    write_csv('first_infection', G)

    # creates graph of threshold distance
    H = create_graph_infected_distance(new_york)
    calculate_node_stats(H)
    write_csv('threshold_distance', H)

    # creates graph of logistic simulation
    J = create_graph_logistic_simulation(new_york)
    # J = create_graph_logistic_simulation_load_lc(new_york, radius_weight=1)
    calculate_node_stats(J)
    write_csv('logistic_simulation', J)
```

```
    # this will make files for the coordinateLayout plugin for cytoscape for␣
↪each graph
    convert_to_cyto_layout('first_infection_data.csv', 'first_infection_network.
↪csv',
                           'first_infection_data_cyto_layout.csv',␣
↪'first_infection_network_cyto_layout.csv')
    convert_to_cyto_layout('threshold_distance_data.csv',␣
↪'threshold_distance_network.csv',
                           'threshold_distance_data_cyto_layout.csv',␣
↪'threshold_distance_network_cyto_layout.csv')
    convert_to_cyto_layout('logistic_simulation_data.csv',␣
↪'logistic_simulation_network.csv',
                           'logistic_simulation_data_cyto_layout.csv',␣
↪'logistic_simulation_network_cyto_layout.csv')


if __name__ == '__main__':
    main()
```

| | UID | iso2 | iso3 | code3 | FIPS | Admin2 | Province_State | \ |
|---|---|---|---|---|---|---|---|---|
| 1833 | 84036001 | US | USA | 840 | 36001.0 | Albany | New York | |
| 1834 | 84036003 | US | USA | 840 | 36003.0 | Allegany | New York | |
| 1835 | 84036005 | US | USA | 840 | 36005.0 | Bronx | New York | |
| 1836 | 84036007 | US | USA | 840 | 36007.0 | Broome | New York | |
| 1837 | 84036009 | US | USA | 840 | 36009.0 | Cattaraugus | New York | |
| 1838 | 84036011 | US | USA | 840 | 36011.0 | Cayuga | New York | |
| 1839 | 84036013 | US | USA | 840 | 36013.0 | Chautauqua | New York | |
| 1840 | 84036015 | US | USA | 840 | 36015.0 | Chemung | New York | |
| 1841 | 84036017 | US | USA | 840 | 36017.0 | Chenango | New York | |
| 1842 | 84036019 | US | USA | 840 | 36019.0 | Clinton | New York | |
| 1843 | 84036021 | US | USA | 840 | 36021.0 | Columbia | New York | |
| 1844 | 84036023 | US | USA | 840 | 36023.0 | Cortland | New York | |
| 1845 | 84036025 | US | USA | 840 | 36025.0 | Delaware | New York | |
| 1846 | 84036027 | US | USA | 840 | 36027.0 | Dutchess | New York | |
| 1847 | 84036029 | US | USA | 840 | 36029.0 | Erie | New York | |
| 1848 | 84036031 | US | USA | 840 | 36031.0 | Essex | New York | |
| 1849 | 84036033 | US | USA | 840 | 36033.0 | Franklin | New York | |
| 1850 | 84036035 | US | USA | 840 | 36035.0 | Fulton | New York | |
| 1851 | 84036037 | US | USA | 840 | 36037.0 | Genesee | New York | |
| 1852 | 84036039 | US | USA | 840 | 36039.0 | Greene | New York | |
| 1853 | 84036041 | US | USA | 840 | 36041.0 | Hamilton | New York | |
| 1854 | 84036043 | US | USA | 840 | 36043.0 | Herkimer | New York | |
| 1855 | 84036045 | US | USA | 840 | 36045.0 | Jefferson | New York | |
| 1856 | 84036047 | US | USA | 840 | 36047.0 | Kings | New York | |
| 1857 | 84036049 | US | USA | 840 | 36049.0 | Lewis | New York | |
| 1858 | 84036051 | US | USA | 840 | 36051.0 | Livingston | New York | |
| 1859 | 84036053 | US | USA | 840 | 36053.0 | Madison | New York | |

```
1860  84036055   US  USA    840  36055.0        Monroe       New York
1861  84036057   US  USA    840  36057.0    Montgomery       New York
1862  84036059   US  USA    840  36059.0        Nassau       New York
...        ...  ...  ...    ...      ...           ...            ...
1866  84036067   US  USA    840  36067.0      Onondaga       New York
1867  84036069   US  USA    840  36069.0       Ontario       New York
1868  84036071   US  USA    840  36071.0        Orange       New York
1869  84036073   US  USA    840  36073.0       Orleans       New York
1870  84036075   US  USA    840  36075.0        Oswego       New York
1871  84036077   US  USA    840  36077.0        Otsego       New York
1872  84036079   US  USA    840  36079.0        Putnam       New York
1873  84036081   US  USA    840  36081.0        Queens       New York
1874  84036083   US  USA    840  36083.0    Rensselaer       New York
1875  84036085   US  USA    840  36085.0      Richmond       New York
1876  84036087   US  USA    840  36087.0      Rockland       New York
1877  84036089   US  USA    840  36089.0  St. Lawrence       New York
1878  84036091   US  USA    840  36091.0      Saratoga       New York
1879  84036093   US  USA    840  36093.0   Schenectady       New York
1880  84036095   US  USA    840  36095.0     Schoharie       New York
1881  84036097   US  USA    840  36097.0      Schuyler       New York
1882  84036099   US  USA    840  36099.0        Seneca       New York
1883  84036101   US  USA    840  36101.0       Steuben       New York
1884  84036103   US  USA    840  36103.0       Suffolk       New York
1885  84036105   US  USA    840  36105.0      Sullivan       New York
1886  84036107   US  USA    840  36107.0         Tioga       New York
1887  84036109   US  USA    840  36109.0      Tompkins       New York
1888  84036111   US  USA    840  36111.0        Ulster       New York
1889  84036113   US  USA    840  36113.0        Warren       New York
1890  84036115   US  USA    840  36115.0    Washington       New York
1891  84036117   US  USA    840  36117.0         Wayne       New York
1892  84036119   US  USA    840  36119.0   Westchester       New York
1893  84036121   US  USA    840  36121.0       Wyoming       New York
1894  84036123   US  USA    840  36123.0         Yates       New York
3179  84080036   US  USA    840  80036.0      Out of NY       New York

     Country_Region        Lat      Long_  ... 5/29/20  5/30/20  5/31/20  \
1833             US  42.600603 -73.977239  ...    1834     1843     1860
1834             US  42.257484 -78.027505  ...      45       45       45
1835             US  40.852093 -73.862828  ...       0        0        0
1836             US  42.159032 -75.813261  ...     557      561      566
1837             US  42.247782 -78.679231  ...      86       87       88
1838             US  42.912617 -76.557316  ...      89       90       91
1839             US  42.227692 -79.366918  ...      82       83       84
1840             US  42.138911 -76.763880  ...     137      137      137
1841             US  42.494300 -75.608876  ...     130      132      133
1842             US  44.745309 -73.678754  ...      95       95       95
1843             US  42.248193 -73.630891  ...     382      383      387
1844             US  42.595092 -76.070489  ...      39       41       41
```

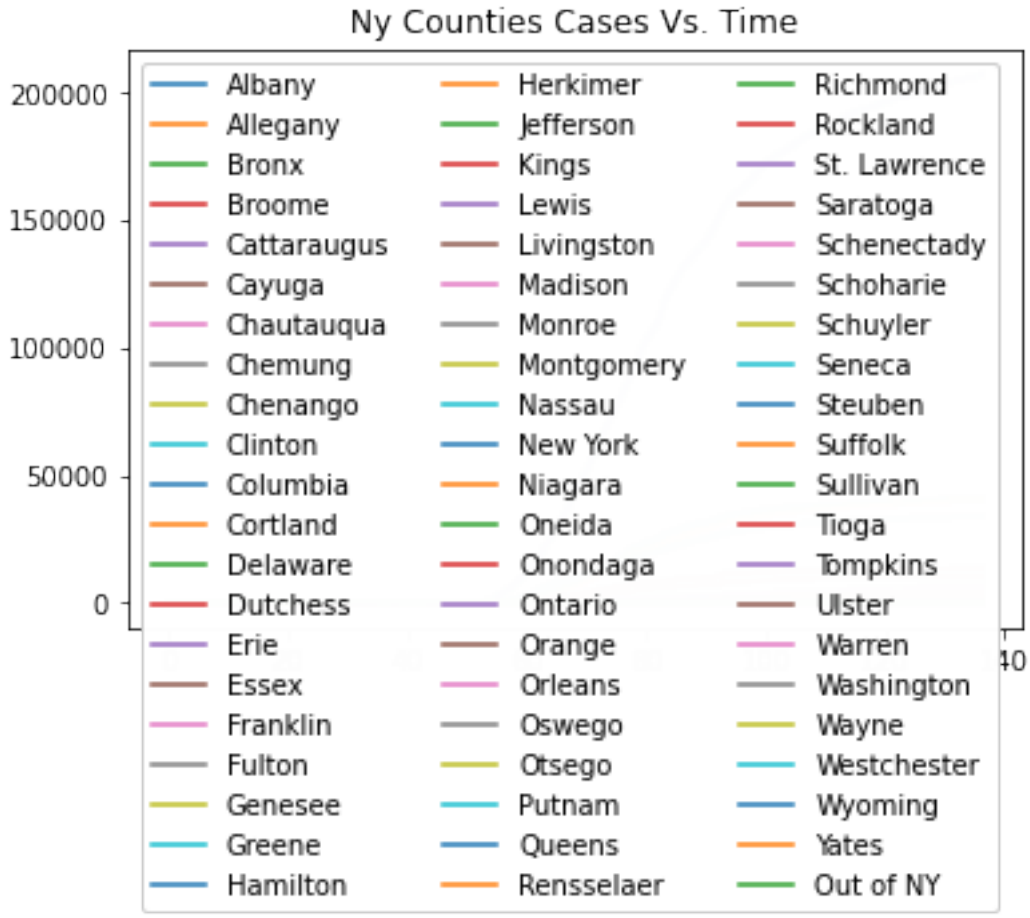
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1845 | US | 42.198376 | -74.967093 | ... | 79 | 79 | 79 |
| 1846 | US | 41.764861 | -73.743567 | ... | 3887 | 3899 | 3909 |
| 1847 | US | 42.762490 | -78.730637 | ... | 5935 | 6014 | 6070 |
| 1848 | US | 44.116308 | -73.772978 | ... | 36 | 36 | 37 |
| 1849 | US | 44.590409 | -74.299260 | ... | 21 | 22 | 22 |
| 1850 | US | 43.113639 | -74.417988 | ... | 206 | 207 | 208 |
| 1851 | US | 43.002260 | -78.191352 | ... | 199 | 200 | 202 |
| 1852 | US | 42.275797 | -74.123849 | ... | 237 | 238 | 238 |
| 1853 | US | 43.661466 | -74.497220 | ... | 5 | 5 | 5 |
| 1854 | US | 43.420342 | -74.961453 | ... | 107 | 107 | 108 |
| 1855 | US | 44.042010 | -75.946535 | ... | 74 | 74 | 74 |
| 1856 | US | 40.636183 | -73.949356 | ... | 0 | 0 | 0 |
| 1857 | US | 43.784416 | -75.449040 | ... | 20 | 20 | 20 |
| 1858 | US | 42.725963 | -77.779662 | ... | 118 | 118 | 119 |
| 1859 | US | 42.916539 | -75.672666 | ... | 310 | 311 | 316 |
| 1860 | US | 43.146389 | -77.693229 | ... | 2860 | 2909 | 2942 |
| 1861 | US | 42.901235 | -74.440116 | ... | 84 | 89 | 91 |
| 1862 | US | 40.740665 | -73.589419 | ... | 40226 | 40307 | 40396 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 1866 | US | 43.004919 | -76.199712 | ... | 2092 | 2133 | 2170 |
| 1867 | US | 42.851457 | -77.308744 | ... | 205 | 206 | 207 |
| 1868 | US | 41.403375 | -74.302408 | ... | 10361 | 10389 | 10406 |
| 1869 | US | 43.251698 | -78.232007 | ... | 216 | 228 | 229 |
| 1870 | US | 43.427789 | -76.146352 | ... | 108 | 108 | 109 |
| 1871 | US | 42.634926 | -75.031514 | ... | 71 | 71 | 72 |
| 1872 | US | 41.426301 | -73.749655 | ... | 1241 | 1248 | 1252 |
| 1873 | US | 40.710881 | -73.816847 | ... | 0 | 0 | 0 |
| 1874 | US | 42.713481 | -73.510899 | ... | 478 | 486 | 491 |
| 1875 | US | 40.585822 | -74.148086 | ... | 0 | 0 | 0 |
| 1876 | US | 41.150279 | -74.025605 | ... | 13100 | 13128 | 13151 |
| 1877 | US | 44.497618 | -75.065500 | ... | 202 | 202 | 204 |
| 1878 | US | 43.109042 | -73.866539 | ... | 473 | 477 | 480 |
| 1879 | US | 42.816688 | -74.052783 | ... | 680 | 686 | 689 |
| 1880 | US | 42.588317 | -74.443390 | ... | 49 | 50 | 50 |
| 1881 | US | 42.391840 | -76.877330 | ... | 11 | 11 | 11 |
| 1882 | US | 42.780810 | -76.824971 | ... | 57 | 58 | 58 |
| 1883 | US | 42.268914 | -77.382992 | ... | 241 | 241 | 241 |
| 1884 | US | 40.883201 | -72.801217 | ... | 39445 | 39532 | 39643 |
| 1885 | US | 41.715795 | -74.763946 | ... | 1364 | 1375 | 1387 |
| 1886 | US | 42.168528 | -76.308358 | ... | 126 | 128 | 130 |
| 1887 | US | 42.449458 | -76.472298 | ... | 157 | 161 | 164 |
| 1888 | US | 41.890279 | -74.262521 | ... | 1663 | 1678 | 1685 |
| 1889 | US | 43.561730 | -73.843370 | ... | 254 | 255 | 255 |
| 1890 | US | 43.311538 | -73.430434 | ... | 232 | 235 | 235 |
| 1891 | US | 43.154944 | -77.029765 | ... | 113 | 113 | 115 |
| 1892 | US | 41.162784 | -73.757417 | ... | 33349 | 33429 | 33481 |
| 1893 | US | 42.701451 | -78.221996 | ... | 82 | 84 | 86 |
| 1894 | US | 42.635055 | -77.103699 | ... | 39 | 39 | 39 |

```
3179            US   0.000000   0.000000  ...       0        0        0
```

| | 6/1/20 | 6/2/20 | 6/3/20 | 6/4/20 | 6/5/20 | 6/6/20 | 6/7/20 |
|---|---|---|---|---|---|---|---|
| 1833 | 1882 | 1900 | 1920 | 1930 | 1941 | 1953 | 1961 |
| 1834 | 48 | 48 | 49 | 49 | 51 | 51 | 51 |
| 1835 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1836 | 574 | 578 | 589 | 593 | 605 | 613 | 620 |
| 1837 | 89 | 89 | 90 | 91 | 92 | 92 | 94 |
| 1838 | 92 | 93 | 96 | 96 | 98 | 100 | 101 |
| 1839 | 85 | 86 | 89 | 95 | 97 | 99 | 101 |
| 1840 | 137 | 137 | 137 | 137 | 137 | 137 | 137 |
| 1841 | 133 | 133 | 133 | 133 | 133 | 133 | 134 |
| 1842 | 96 | 97 | 97 | 97 | 97 | 97 | 97 |
| 1843 | 389 | 391 | 399 | 400 | 411 | 412 | 414 |
| 1844 | 41 | 41 | 41 | 41 | 41 | 41 | 41 |
| 1845 | 81 | 82 | 82 | 82 | 84 | 85 | 85 |
| 1846 | 3924 | 3936 | 3951 | 3962 | 3984 | 3995 | 4000 |
| 1847 | 6123 | 6173 | 6234 | 6308 | 6359 | 6429 | 6486 |
| 1848 | 37 | 38 | 38 | 38 | 38 | 38 | 38 |
| 1849 | 23 | 23 | 23 | 23 | 23 | 23 | 23 |
| 1850 | 210 | 212 | 213 | 215 | 219 | 222 | 224 |
| 1851 | 202 | 202 | 202 | 205 | 205 | 206 | 208 |
| 1852 | 239 | 241 | 241 | 241 | 242 | 246 | 246 |
| 1853 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 1854 | 109 | 111 | 113 | 113 | 115 | 122 | 123 |
| 1855 | 74 | 74 | 74 | 74 | 75 | 75 | 77 |
| 1856 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1857 | 20 | 20 | 20 | 20 | 20 | 20 | 20 |
| 1858 | 119 | 119 | 120 | 120 | 121 | 121 | 121 |
| 1859 | 316 | 317 | 319 | 319 | 323 | 325 | 327 |
| 1860 | 2964 | 2989 | 3048 | 3081 | 3117 | 3167 | 3190 |
| 1861 | 91 | 93 | 94 | 96 | 96 | 98 | 100 |
| 1862 | 40479 | 40572 | 40644 | 40713 | 40797 | 40853 | 40904 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 1866 | 2197 | 2228 | 2256 | 2295 | 2329 | 2375 | 2392 |
| 1867 | 208 | 209 | 215 | 218 | 219 | 220 | 221 |
| 1868 | 10422 | 10449 | 10460 | 10471 | 10484 | 10508 | 10514 |
| 1869 | 236 | 241 | 246 | 247 | 247 | 254 | 255 |
| 1870 | 110 | 110 | 112 | 112 | 112 | 114 | 115 |
| 1871 | 72 | 73 | 73 | 73 | 73 | 74 | 74 |
| 1872 | 1257 | 1262 | 1264 | 1268 | 1270 | 1274 | 1277 |
| 1873 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1874 | 492 | 493 | 495 | 495 | 497 | 499 | 502 |
| 1875 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1876 | 13185 | 13223 | 13259 | 13280 | 13297 | 13315 | 13325 |
| 1877 | 205 | 205 | 206 | 207 | 207 | 209 | 209 |
| 1878 | 483 | 484 | 489 | 493 | 496 | 501 | 502 |
| 1879 | 693 | 696 | 701 | 701 | 706 | 710 | 711 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1880 | 50 | 51 | 51 | 51 | 51 | 54 | 54 |
| 1881 | 11 | 12 | 12 | 12 | 12 | 12 | 12 |
| 1882 | 59 | 59 | 60 | 60 | 61 | 61 | 61 |
| 1883 | 241 | 242 | 243 | 245 | 246 | 251 | 251 |
| 1884 | 39705 | 39980 | 40062 | 40153 | 40239 | 40278 | 40329 |
| 1885 | 1389 | 1392 | 1393 | 1405 | 1409 | 1411 | 1415 |
| 1886 | 130 | 131 | 133 | 133 | 134 | 134 | 134 |
| 1887 | 164 | 164 | 165 | 165 | 167 | 171 | 171 |
| 1888 | 1691 | 1696 | 1701 | 1704 | 1711 | 1714 | 1718 |
| 1889 | 255 | 255 | 256 | 256 | 256 | 257 | 257 |
| 1890 | 235 | 237 | 238 | 238 | 240 | 240 | 240 |
| 1891 | 116 | 116 | 120 | 121 | 122 | 123 | 124 |
| 1892 | 33552 | 33633 | 33691 | 33767 | 33854 | 33924 | 33954 |
| 1893 | 86 | 86 | 87 | 87 | 88 | 89 | 89 |
| 1894 | 39 | 39 | 39 | 39 | 39 | 39 | 39 |
| 3179 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

[63 rows x 149 columns]



*Here is a video of our final simulation visualization in cytoscape. It shows of COVID theoretically spread from county to county in the state of New York.*

[126]:
```
Video('NewGraphDemo.mp4')
```

[126]:
```
<IPython.core.display.Video object>
```

[ ]:
```

```