MTH4322 - Internship Seminar

Final Project

Dan Kolonay

Introduction

In order to compute the 'hospitalizability' of cities, I utilized open source data from OSM (OpenStreetMap) to create a node graph of cities with nodes and intersections and edges as roads between them. This was largely done with the help of osmnx, a python library used to work with OSM data as NetworkX graphs.

Initializing the Data

I started by initializing the roadway data based on the target city as seen below. Note that the edges of the graph were then reversed. This will come in handy later when calculating distances to the hospitals. The weights associated with the edges of the graph are travel times calculated based on the distance between nodes and the posted speed limits on the roads. With the data available to me, this was the best estimate I was able to get to travel times, although I admit more data about average speeds collected from the area and traffic data would strengthen the analysis.

After roadway data was added to the graph, hospital data was imported and snapped to the nearest node based on a single point to identify its latitude and longitude location.

```
# first create the graph and the features
def initialize_data(location):
    place = location
    G_initial = ox.graph.graph_from_place(place, network_type="drive")
    G = G_initial.reverse(copy=True)
    #Reverse the graph so Djikstra's algorithm can be calculated from hospitals instead of to them

    ox.routing.add_edge_speeds(G, hwy_speeds=None, fallback=None, agg=np.mean)
    ox.routing.add_edge_travel_times(G)

    features = ox.features.features_from_place(place, {"amenity": "hospital"})

    feature_points = features.representative_point()
    nn = ox.distance.nearest_nodes(G, feature_points.x, feature_points.y)
    useful_tags = ["name", "emergency"]

    for node, feature in zip(nn, features[useful_tags].to_dict(orient="records")):
        feature = {k: v for k, v in feature.items() if pd.notna(v)}
        G.nodes[node].update({"hospital": feature})

    return G

G = initialize_data("New York, NY, USA")
```

I then collected a list of the node ids of nodes that contained a hospital to be used in distance calculations from every other node in the network

```
hospital_node_ids = []

for node in list(G.nodes):
    attr_name = 'hospitals'
    attr = set()

    G.nodes[node][attr_name] = attr

    if 'hospital' in G.nodes[node]:
        hospital_node_ids.append(node)
```

Calculating Distances to Hospitals

The method I used to calculate the closest distance to a hospital from every location was Dijkstra's algorithm. This is the reason the edges on the graph were reversed earlier. Rather than calculate the distance to every hospital from each node on the graph, I calculated the distance to every node from every hospital. The distinction here is that, since some roads are one-way, running the algorithm on the reversed graph from the perspective of the hospitals is like running it from every node in the original graph.

Dijkstra's algorithm was my choice for this task because it can very efficiently calculate the distance from a single point to all other points on the graph. By using this algorithm on each hospital node, I was able to easily calculate distances from every location on the map to each hospital. The values for travel time to each hospital were then input directly into the NetworkX graph nodes for each location where the lowest values would indicate the travel time to the closest and second closest hospitals.

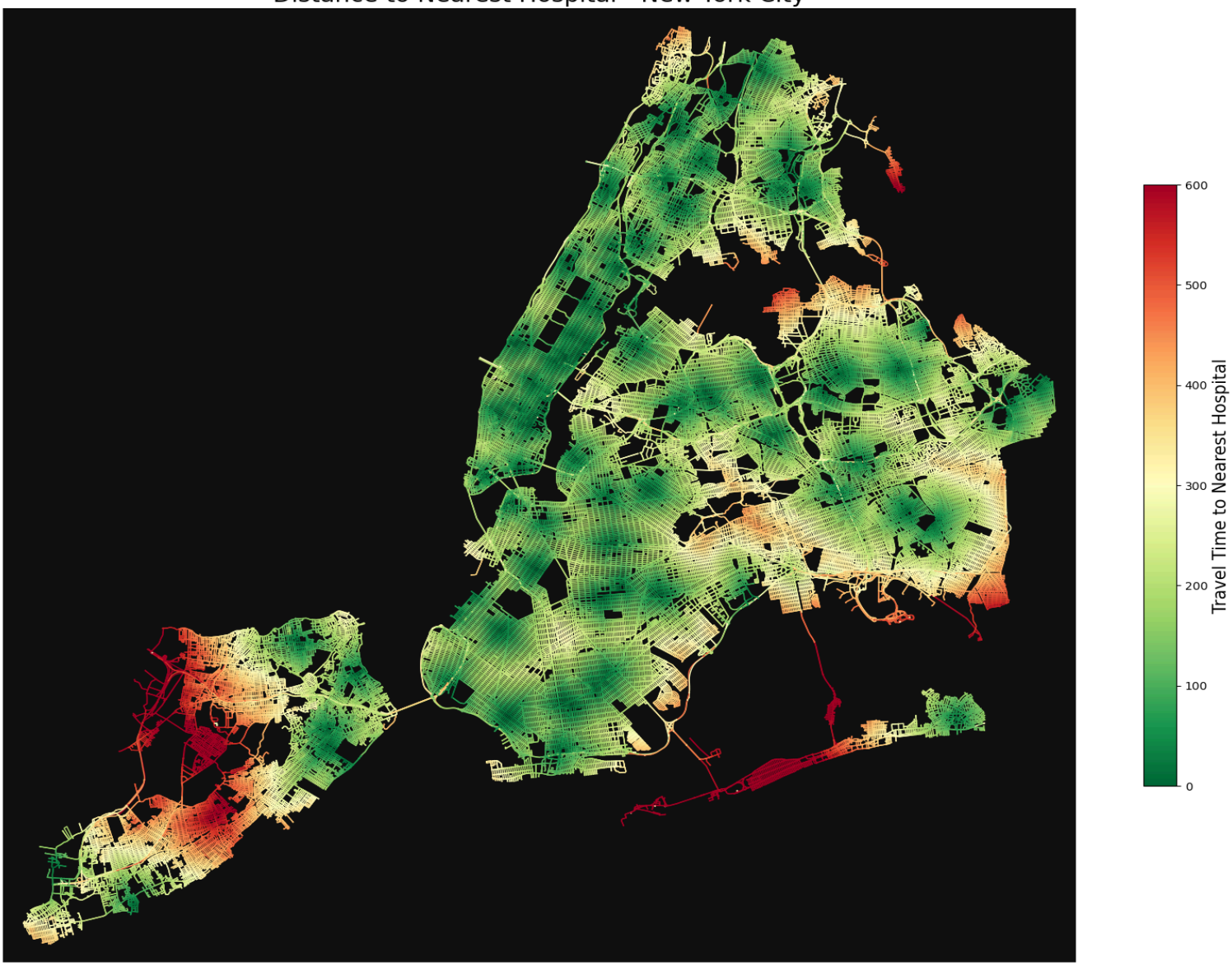
Once these values were found, calculating the average travel time to hospitals across all nodes in the city was done by simply adding up the travel time to the closest hospital and dividing by the number of nodes.

```python
def dijkstra_to_hospital(graph, hospital_node):
    travel_times = {node: float('inf') for node in graph.nodes}
    travel_times[hospital_node] = 0
    pq = [(0, hospital_node)]

    while pq:
        current_travel_time, current_node = heapq.heappop(pq)

        if current_travel_time > travel_times[current_node]:
            continue

        for neighbor in graph.neighbors(current_node):
            edge_data = graph.get_edge_data(current_node, neighbor)
            min_edge_time = min(attr['travel_time'] for attr in edge_data.values())

            travel_time = current_travel_time + min_edge_time

            if travel_time < travel_times[neighbor]:
                travel_times[neighbor] = travel_time
                heapq.heappush(pq, (travel_time, neighbor))

    return travel_times
```

Visualizing the Data

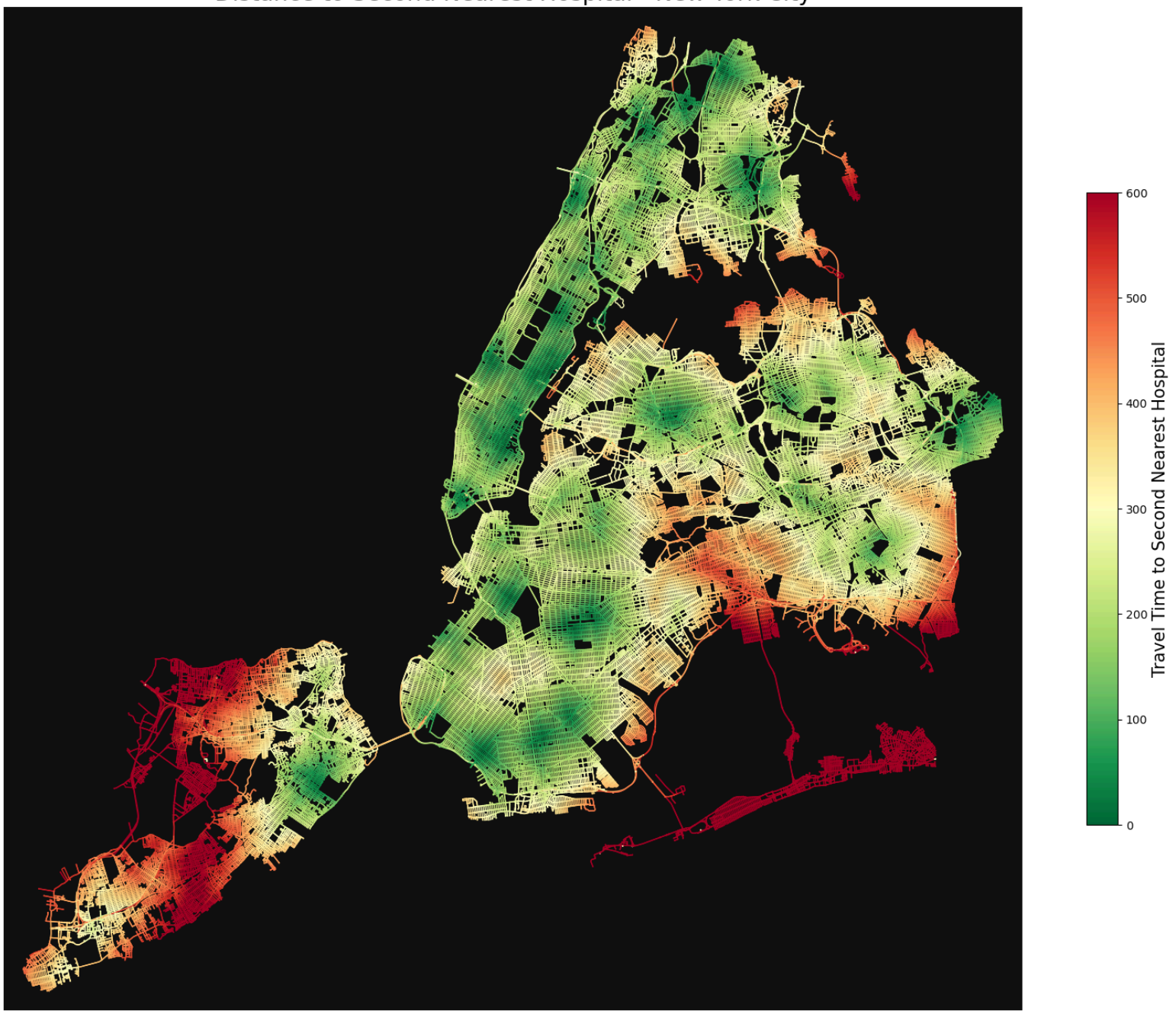
To visualize the data, I used the matplotlib-based plotting feature built into osmnx to display the raw node graphics and Folium to represent the data superimposed onto an interactive map. A custom cmap was made to display travel times to the hospital greater than 10 minutes in red. To avoid a cluttered map, I opted to make the nodes themselves hidden and instead added the color from the nodes to the nearest edge. The full code for the mapping process can be found in the Jupyter notebook, but it wouldn't fit nicely in this document. The folium map in this document is static, so if you would like to view the interactive version, view it in the Jupyter notebook.

Distance to Nearest Hospital - New York City

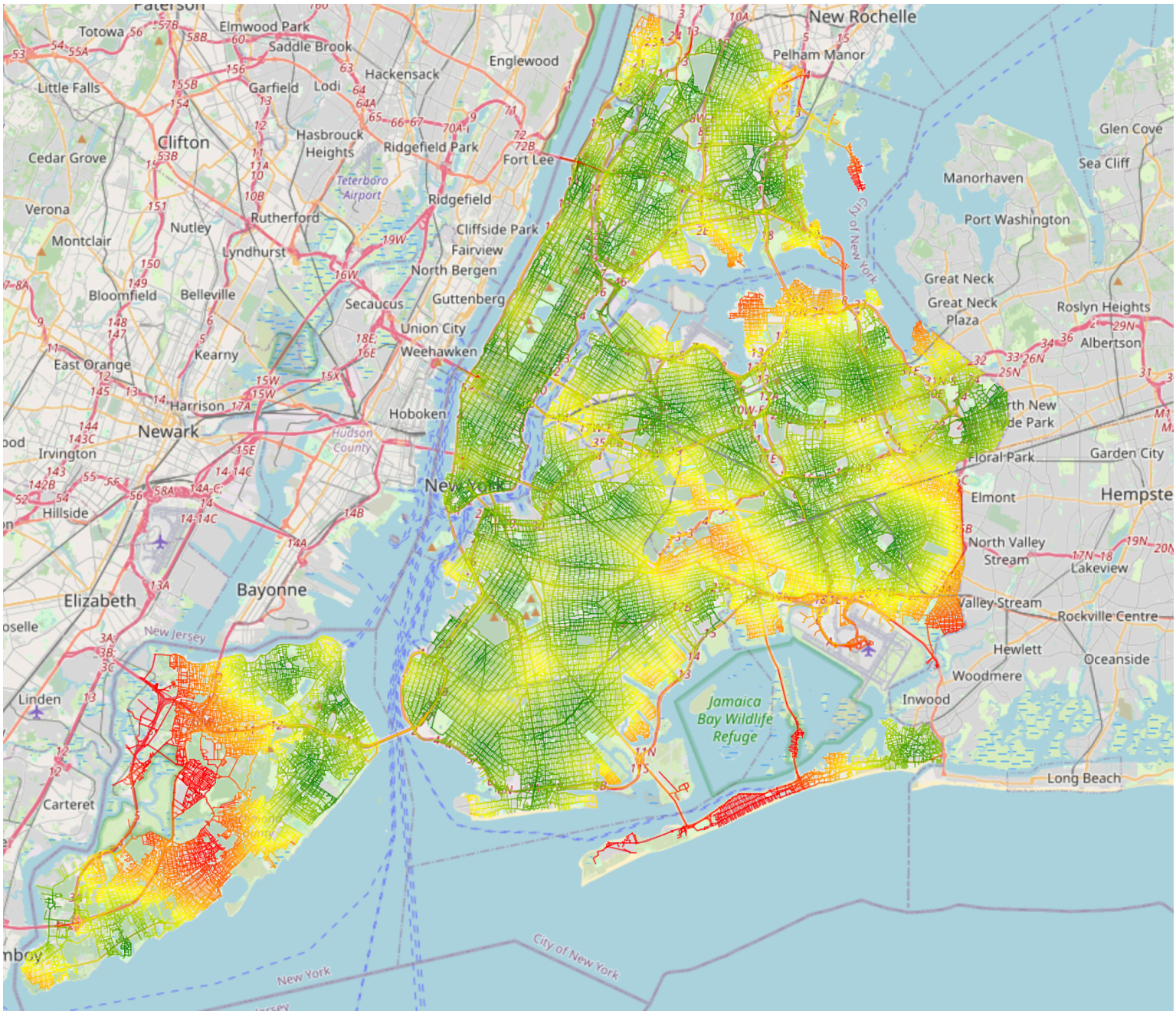


Average travel time to nearest hospital: 3 minutes 42 seconds

Distance to Second Nearest Hospital - New York City



Average travel time to second hospital: 5 minutes 13 seconds


New Rochelle
Pelham Manor
Englewood
Hackensack
Garfield
Lodi
Little Falls
Clifton
Cedar Grove
Hasbrouck Heights
Ridgefield Park
Fort Lee
Teterboro Airport
Ridgefield
Verona
Nutley
Rutherford
Montclair
Lyndhurst
Cliffside Park
Fairview
North Bergen
Bloomfield
Belleville
Secaucus
Guttenberg
Union City
Weehawken
East Orange
Kearny
Harrison
Hoboken
Newark
Hudson County
Irvington
Hillside
Elizabeth
Bayonne
New Jersey
Linden
Carteret
Glen Cove
Sea Cliff
Manorhaven
Port Washington
City of New York
Great Neck
Great Neck Plaza
Roslyn Heights
Albertson
Floral Park
Garden City
Elmont
North Valley Stream
Lakeview
Valley Stream
Rockville Centre
Hewlett
Oceanside
Woodmere
Inwood
Jamaica Bay Wildlife Refuge
Long Beach
New York
City of New York