

## Project Code

[https://github.com/tallenj/si507\\_w23\\_final\\_project](https://github.com/tallenj/si507_w23_final_project)

I ended up not using any of my coworkers' code! I did learn some of the basics from him and made improvements where I could, but his application turned out to be very different from what I wanted to do. I learned more from the OSMnx documentation tbh. That said, I did get a headstart in learning folium thanks to his use of CircleMarker but I ended up using PolyLine instead of ColorLine.

## Data Sources

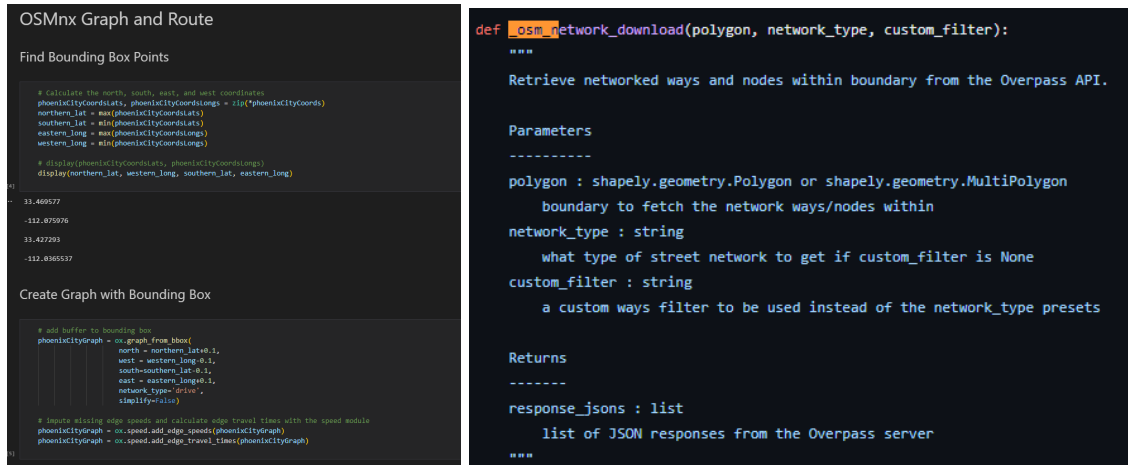
### OSMnx

- OSMnx is the main tool I'm trying to learn in the final project. It handles a lot of the grunt work that this project calls for. Once I read the documentation and studied the examples I found that it was fairly easy to get the map data and if I use the Google API it is also easy to get the elevation data. I was able to spend more time on creating the visualizations I wanted thanks to this tool.
- That said, I did have to learn about folium and leaflet libraries too. Learning OSMnx, folium, and leaflet has been the main focus, and with OSMnx I also learn about NetworkX. All in all this project has been a great learning experience so far, this checkpoint is mainly me trying to make sure I'm going to get credit for the project along the way :)
- For data summary, I get a graph from Open Street Maps thanks to OSMnx, that has nodes and edges. I use the 'osmids' of the nodes and edges, the lat and long of the nodes. Once I add elevation data I also use the node elevation and the edge grade.
- I put some screenshots of the code from the libraries below, and a screenshot of the documentation stating that caching is default True.

### Open Street Maps API

- I'm using the OSMnx library which handles the access and caching of Open Street Maps data for me. Specifically, I'm using the bounding box function where I give four coordinates (that I find from a route) and OSMnx handles the API call, download, and caching:

tallenj



use\_cache : bool

If True, cache HTTP responses locally instead of calling API repeatedly for the same request. Default is True.

- The above images should hopefully show that caching is being used for the Open Street Maps data. In my dev notebook I'm using a small route, my final submission will be a 80+ stretch of highway which will have more nodes until I filter by the route.

## Google Elevation API

```
def add_node_elevations_google(
    G,
    api_key,
    max_locations_per_batch=350,
    pause_duration=0,
    precision=3,
    url_template="https://maps.googleapis.com/maps/api/elevation/json?locations={}&key={}",
):
    """# pragma: no cover
    """
    Add "elevation" (meters) attribute to each node using a web service.

    By default, this uses the Google Maps Elevation API but you can optionally
    use an equivalent API with the same interface and response format, such as
    Open Topo Data. The Google Maps Elevation API requires an API key but
    other providers may not.

    For a free local alternative see the 'add_node_elevations_raster'
    function. See also the 'add_edge_grades' function.

    Parameters
    -----
    G : networkx.MultiDiGraph
        input graph
    api_key : string
        a valid API key
    max_locations_per_batch : int
        max number of coordinate pairs to submit in each API call (if this is
        too high, the server will reject the request because its character
        limit exceeds the max allowed)
    pause_duration : float
        time to pause between API calls, which can be increased if you get
        rate limited
    precision : int
        decimal precision to round elevation values
    url_template : string
        a URL string template for the API endpoint, containing exactly two
        parameters: 'locations' and 'key'; for example, for Open Topo Data:
        "https://api.opentopodata.org/v1/aster30m?locations={}&key={}"

    Returns
    -----
    G : networkx.MultiDiGraph
        graph with node elevation attributes
    """
```

```
# check if this request is already in the cache (if global use_cache=True)
cached_response_json = downloader.retrieve_from_cache(url)
if cached_response_json is not None:
    response_json = cached_response_json
```

tallenj

- OSMnx has a function that accesses the Google API for me and it utilizes caching by default. I created an API key and tested it in my dev notebook. I am currently downloading elevation data for every node in the graph but for my final submission I plan to make a graph of only the route nodes and just get that elevation data.
- OSMnx checks the cache and creates a networkX graph for me with the added elevation data using the graph I passed in from the original Open Street Maps as the original.

## Data Structures

- OSMnx already created the graph for me.
- The interactive html in my github link should show progress :)

## Interaction and Presentation Plans

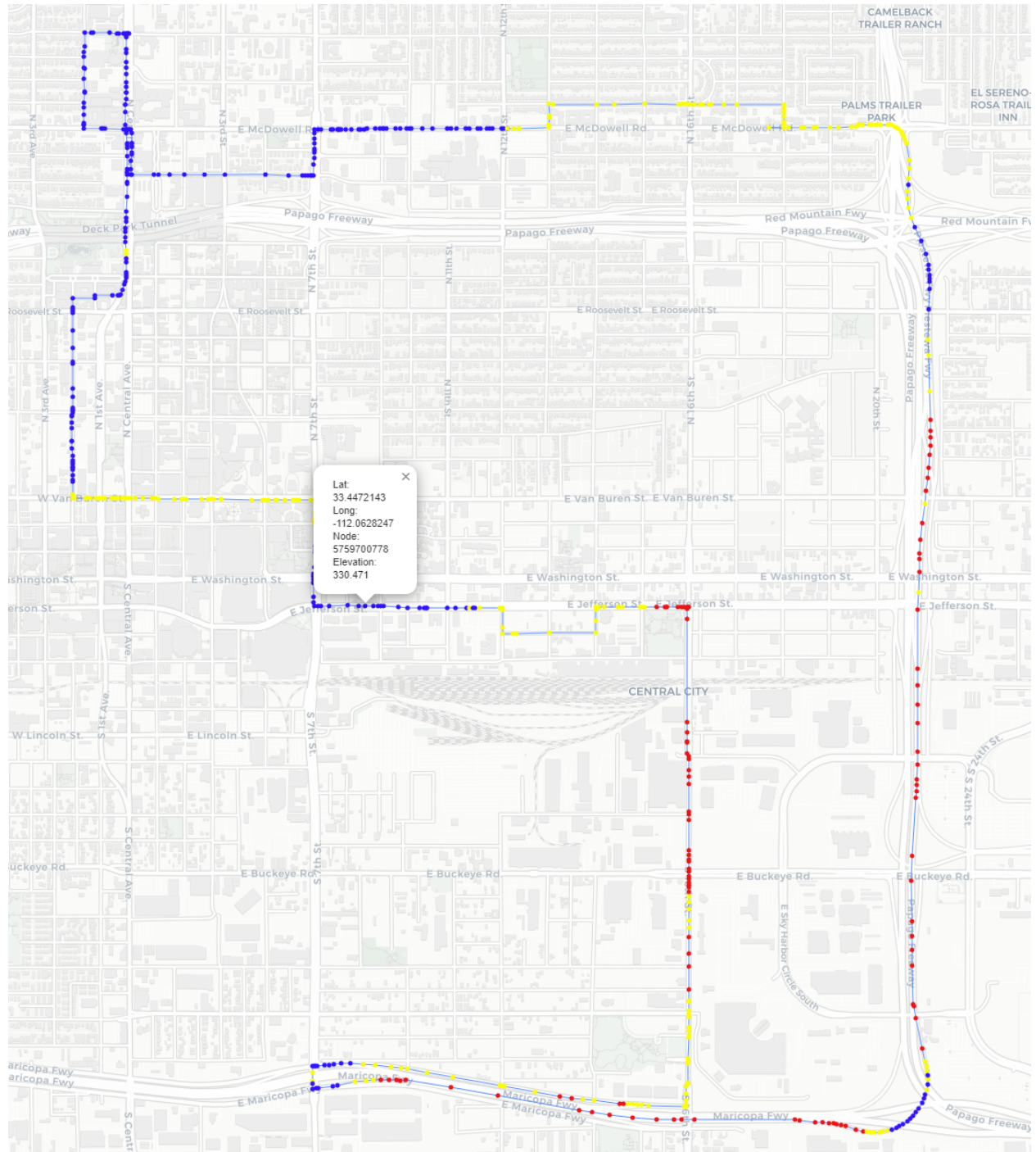
- The html file in the github shows the vision I had for my display.
- There is a zoomed in map of the area that the route is in, with the default view showing the route.
- There are three other layers, the first adding nodes on the route with their elevation.
- The second shows all of the edges in the route with their grade.
- The third shows a grade filter.
  - This filter is currently coded in the notebook and is locked in for the html.
  - I plan to look into adding a legend and putting the coded filter value into the html output.
  - If my team at work thinks it valuable, I'll make the filter more dynamic, possibly with the leaflet slider control plugin :)
- The checkpoint stops here, but I decided to show screenshots of the html file below. This isn't the final route I plan to use but it shows the functionality is possible and in place!

tallenj

## Route

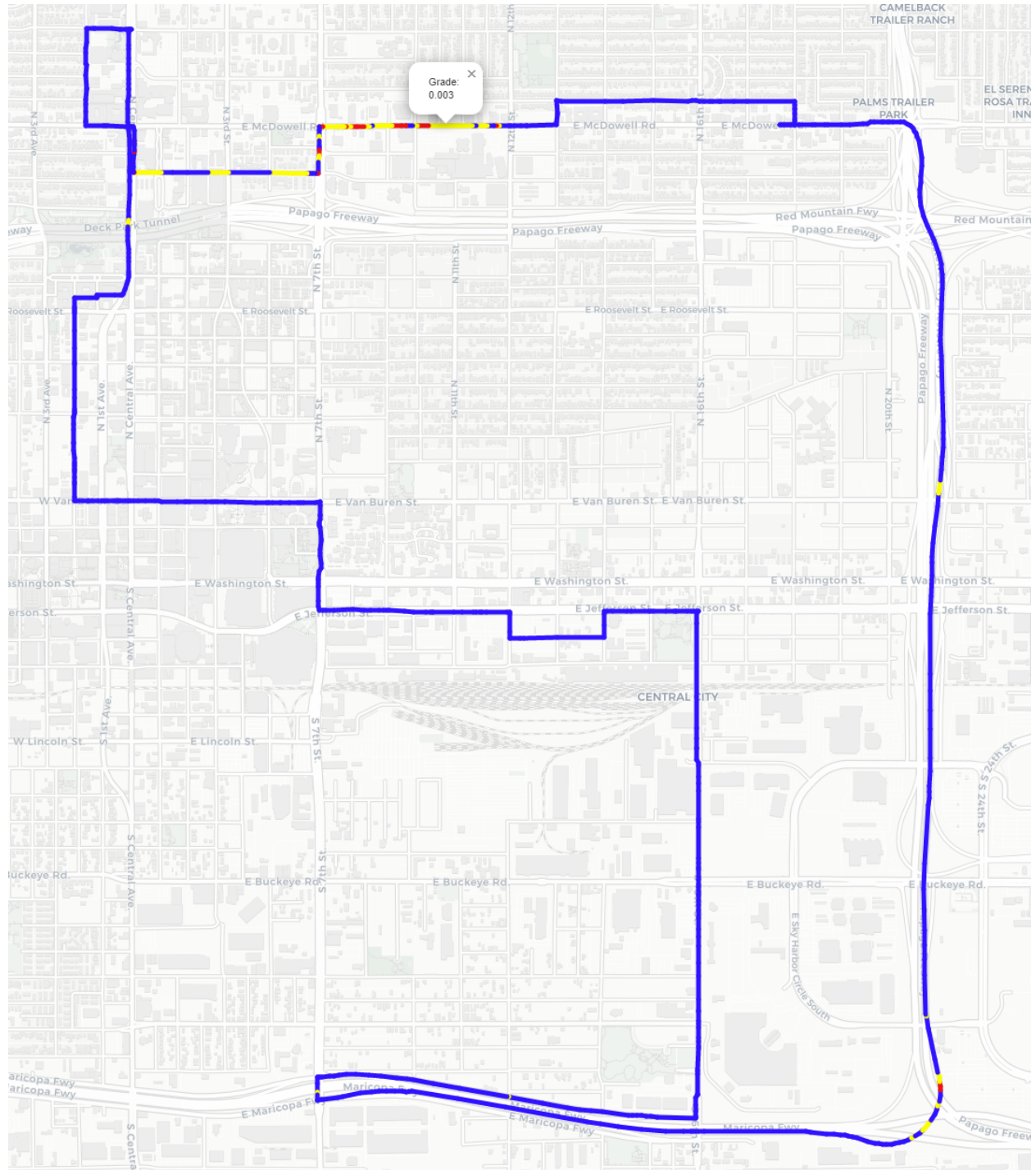
tallenj

## Node Elevation



tallenj

## Edge Grade





tallenj

## Edge Grade Filtered

