Geog 675

Yifeng Ai

Final Project Executive Summary

Introduction

This project is done as an extension of, or an addition to, a web map I made for Geography 699 I am taking this semester with Professor Gartner. One of the data sources of this project comes from the work I've done for Geog 699, and other data is collected for this course. The result of the data analysis done for this project is added to, and presented in the web map I've made for Geog 699. However, without the big data analysis skills I learned in this course, including writing GeoJSON, using GeoPandas and shapely in performing spatial operations, and using OSMnx for doing street network analysis, I would not have been able to add this functionality to the web map.

The project goal is to add a functionality to the web map, with the purpose of visualizing number of surrounding restaurants of each apartment within 5-minute and 10-minute walking distance, which hopefully can be an important reference for students looking for off-campus housing.

Data

There are two base datasets needed for completing this project—a GeoJSON containing all apartment polygons and a GeoJSON containing all restaurants and their respective attributes. The first GeoJSON contains 796 apartment polygons, which was generated before I started this project, so I will not report about details in generating this GeoJSON.

The data of the second GeoJSON comes from Yelp. For the first step, I developed a python program to scrape the full HTML source code of each single one of the 990 Madison restaurant web pages from Yelp using Selenium; then, I parsed this data into a csv file with multiple columns including FID (the only FID for each restaurant, used as Primary Key), Name (of the restaurant), URL (to the corresponding web page on Yelp), Address (street address of the restaurant), City, State (these two field are included for an accurate geocoding), rating count and rating score. This csv was sent to Google Sheets to geocode using an add-on called "Geocode by Awesome Table", and I downloaded the new csv with two addition fields—longitude and latitude. Then I the parsed the new csv into a GeoJSON file using another Python program, and added a new field—the opening hour of the restaurant to it; because this field cannot be conveniently stored in a csv, I chose to add it directly to the GeoJSON in this later stage using the json lib in Python.

Method

The data collection procedure is already done at this step. Next, I will create 2 new GeoJSON files, with all features being 5-minute and 10-minute walking boundary of all 796 apartments. I did it on Jupyter Notebook, because the amount of calculation was large at every step, and using Jupyter Notebook to do the programming in multiple parts can make mistakes easier to detect and quicker to correct (in this case, every time I correct a mistake, I don't need to start all over again and take time to import libraries, to download the entire street network for Madison and to find the nearest nodes for every apartment).

The first two steps imports packages and correct environment issues with OSMnx, and download a street network graph for Madison, as well as define walking distance and time.

In [1]:
```
import geopandas as gpd
import matplotlib.pyplot as plt
import networkx as nx
import osmnx as ox
from shapely.geometry import Point, LineString, Polygon, mapping
import json

ox.config(log_console=True, use_cache=True)
ox.__version__

import os
os.environ["PROJ_LIB"] = "C:/Users/osvob/Anaconda3/Library/share" #corrects the environment issue with OSMnx
print('packages imported')
```

```
packages imported
```

In [2]:
```
# configure the place, network type, trip times, and travel speed
place = 'Madison, WI, USA'
network_type = 'walk'
trip_times = [5, 10] #in minutes
travel_speed = 4.5 #walking speed in km/hour

# download the street network
G = ox.graph_from_place(place, network_type=network_type)
print('downloaded')
```

```
downloaded
```

This step reads polygons into a list from the apartment polygons GeoJSON.

In [3]:
```
# add an edge attribute for time in minutes required to traverse each edge
meters_per_minute = travel_speed * 1000 / 60 #km per hour to m per minute
for u, v, k, data in G.edges(data=True, keys=True):
    data['time'] = data['length'] / meters_per_minute
print('travel time range defined')
```

```
travel time range defined
```

In [4]:
```
import re
with open('D:/Senior 2nd semester/Geog_565/Map!!!!!!/leaflet-lab-Lab5B - Copy (2)/data/all_apts_222444.geojson','r') as fr:
    # this file contains all 796 apartment polygons. Read it into a single string
    all = fr.readlines()[0]

# get the coordinates and FID of each apartment polygon
pat = re.compile('"coordinates": \[\[(.*?)\]\]')
pat_fid = re.compile('"FID":(.*?),')

allPoly = re.findall(pat, all)
allFID = re.findall(pat_fid, all)
for FID in allFID:
    FID = int(FID.replace(' ',''))

allPolyNew = []
for i in range(0, len(allPoly)):
    nodesNew = []
    nodes = allPoly[i].split('], [')[:-1]

    # Read every node in each polygon, parse it into a collection of 795 polygons.
    # This step can actually be done using the json lib.
    for node in nodes:
        node = node.strip().replace('[','').replace(']','')
        x = float(node.split(',')[0])
        y = float(node.split(',')[1])
        nodeNew = (x,y)
        nodesNew.insert(0,nodeNew)
    allPolyNew.append(nodesNew)
print(len(allPolyNew),allPolyNew[0])
print(allPolyNew[795])  # check if the new polygon list is successfully generated
```

```
796 [(-89.50905053873521, 43.022593809815554), (-89.50900264029855, 43.02256247152154), (-89.50902421817999, 43.0225447168564
9), (-89.50903211223985, 43.02254987882688), (-89.50902664535903, 43.0225543776112), (-89.50904600444932, 43.02256704361293),
(-89.50905191434666, 43.02256218303569), (-89.50907219252191, 43.02257544837762), (-89.50907749205078, 43.022571089229416), (-8
```

This step uses convex hull function provided in OSMnx to create 5-minute and 10-minute walk distance boundary GeoJSON and print it out. I can then copy the print-out and paste in a text editor to create two new GeoJSON files.

In [8]:
```
# make the isochrone polygons: SIMPLE
import shapely
from shapely.geometry import Point, Polygon

isochrone_polys_5min = []
isochrone_polys_10min = []

for i in range(0,796):#range(len(nearestNodeL)):
    center_node = nearestNodeL[i]
    count = 0
    for trip_time in sorted(trip_times, reverse=True):
        subgraph = nx.ego_graph(G, center_node, radius=trip_time, distance='time')
        node_points = [Point((data['x'], data['y'])) for node, data in subgraph.nodes(data=True)]
        bounding_poly = gpd.GeoSeries(node_points).unary_union.convex_hull

        if count == 0:
            isochrone_polys_10min.append(bounding_poly)
        if count == 1:
            isochrone_polys_5min.append(bounding_poly)
        count += 1

print(len(isochrone_polys_5min))

goL = []
goL2 = []
for i in range(0, 796):
    goL.append(isochrone_polys_5min[i])
    goL2.append(isochrone_polys_10min[i])
print(gpd.GeoSeries(goL).to_json())
print('\n\n\n\n\n\n\n\n\n\n\n\n\n\n')
print(gpd.GeoSeries(goL2).to_json())
```

```
796
{"type": "FeatureCollection", "features": [{"id": "0", "type": "Feature", "properties": {}, "geometry": {"type": "Polygon",
"coordinates": [[[-89.5114258, 43.0156868], [-89.5129484, 43.016867], [-89.516063, 43.0195967], [-89.5169012, 43.0204491], [-
89.5172477, 43.0220055], [-89.5168492, 43.0232358], [-89.5151197, 43.0254757], [-89.5098736, 43.0283724], [-89.5079166, 43.02
90805], [-89.505452, 43.025928], [-89.5042869, 43.0242352], [-89.503155, 43.019727], [-89.5057019, 43.0168212], [-89.5086998,
43.0160741], [-89.5114258, 43.0156868]]]}, "bbox": [-89.5172477, 43.0156868, -89.503155, 43.0290805]}, {"id": "1", "type": "F
eature", "properties": {}, "geometry": {"type": "Polygon", "coordinates": [[[-89.4001647, 43.0261127], [-89.4049226, 43.02631
72], [-89.40477, 43.0285089], [-89.4041514, 43.0307254], [-89.40397, 43.0310197], [-89.402668, 43.032649], [-89.396429, 43.03
5079], [-89.394632, 43.030001], [-89.3949603, 43.0265705], [-89.4001647, 43.0261127]]]}, "bbox": [-89.4049226, 43.0261127, -8
9.394632, 43.035079]}, {"id": "2", "type": "Feature", "properties": {}, "geometry": {"type": "Polygon", "coordinates": [[[-8
9.4074966, 43.0332655], [-89.4076062, 43.0333047], [-89.4119551, 43.0352655], [-89.4124231, 43.0361365], [-89.411129, 43.0373
```

Then, in a new Jupyter Notebook, I read the 5-min and 10-min walking boundary GeoJSON, and the restaurant GeoJSON into geoPandas, and find all restaurants for each walking boundary polygon.

```
In [2]: from osgeo import ogr
        from shapely.geometry import Point, Polygon
        import os
        import geopandas as gpd

        boundary_df = gpd.read_file('D:/Senior 2nd semester/Geog_565/Map!!!!!!/leaflet-lab-Lab5B - Copy (2)/data/all_apts_5_min.geojson'
        res_df = gpd.read_file('D:/Senior 2nd semester/Geog_565/Map!!!!!!/leaflet-lab-Lab5B - Copy (2)/data/all_restaurants.geojson')
```

```
In [ ]: coveredL = []
        for i in range(0, len(boundary_df)):
            covered = []
            print (i) # collection is slow, print out i to see progress
            for k in range(0, len(res_df)):
                # collect all restaurants within 5-min walking boundaries of each apartment
                if (res_df.iloc[k].geometry).within(boundary_df.iloc[i].geometry):
                    covered.append(k)
                    # FIDs of the restaurant geojson is from 0 to 796, so just use k as the FID to append to the list of covered restaur
            coveredL.append(covered)
        print(len(coveredL), len(boundary_df))
```

```
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
```

Then generate a new 5-minute and 10-minute walking boundary GeoJSON containing a new property field—FIDs of all restaurants within each polygon boundary.

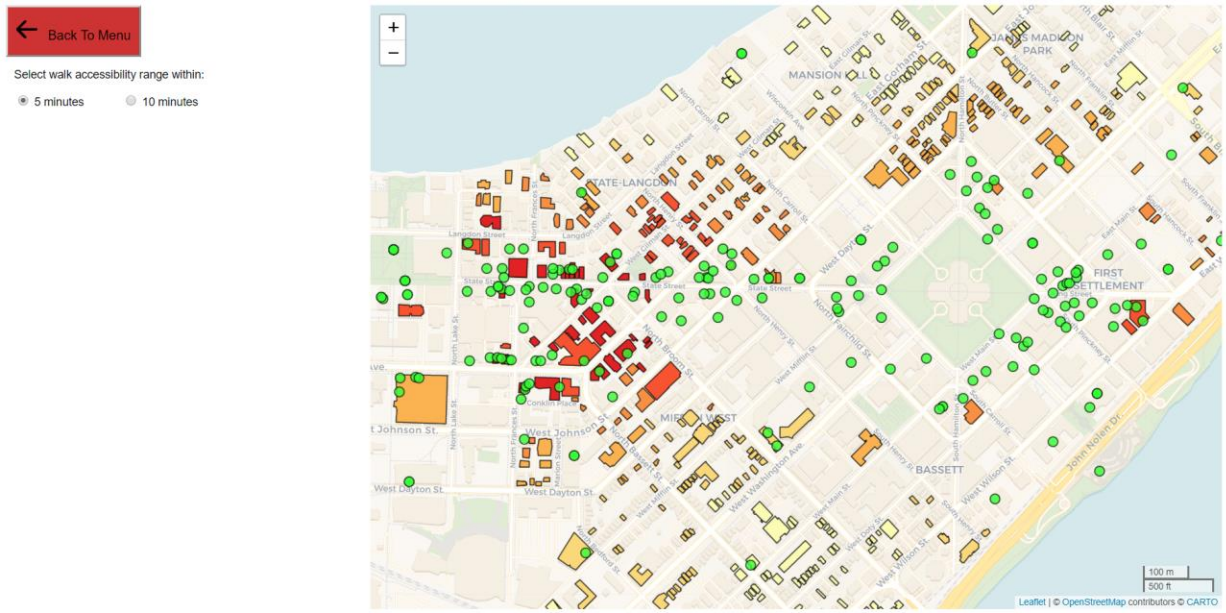
```
In [8]: import fiona
        boundary_df2 = boundary_df.assign(reachable_restaurants = coveredL)

        print(boundary_df2.to_json())
        #boundary_df2.to_file("D:/output.geojson", driver="GeoJSON")

        # ----- Writing a geojson somehow failed, so I directly print out the string into a geojson format
        # and copy and paste it into a text editor to create a 5-min apartment walking boundary polygon geojson, with an
        # additional property field-- a list of FIDs of all restaurants within each boundary, so that later I can retrieve
        # information from corresponding features in the restaurant geojson that was created in the data preparation stage.
```

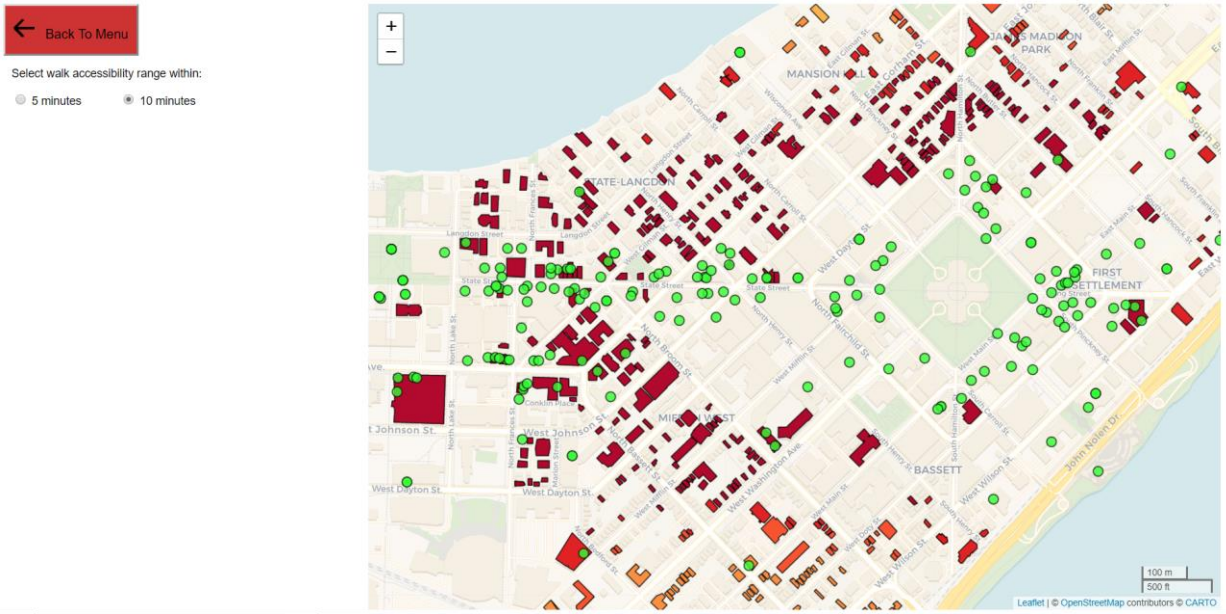

```
{"type": "FeatureCollection", "features": [{"id": "0", "type": "Feature", "properties": {"id": "0", "reachable_restaurants":
[]}, "geometry": {"type": "Polygon", "coordinates": [[[-89.508584, 43.01826], [-89.5120674, 43.0187468], [-89.512906, 43.0192
15], [-89.5142203, 43.0210419], [-89.5134287, 43.024578], [-89.5126168, 43.02563], [-89.5100289, 43.0261063], [-89.507512, 4
3.024434], [-89.5071529, 43.0192482], [-89.5071779, 43.0190057], [-89.508584, 43.01826]]]}}, {"id": "1", "type": "Feature",
"properties": {"id": "1", "reachable_restaurants": []}, "geometry": {"type": "Polygon", "coordinates": [[[-89.4001647, 43.026
1127], [-89.4022149, 43.0266804], [-89.4024458, 43.0272355], [-89.401214, 43.03014], [-89.396459, 43.032615], [-89.394632, 4
3.030001], [-89.3949603, 43.0265705], [-89.4001647, 43.0261127]]]}}, {"id": "2", "type": "Feature", "properties": {"id": "2",
```


The result of the collected data is used for generation of a new functionality of an apartment searching website (actually it's currently only available on a local server and has not been published online). The code was long and is integrated into a 1500+ line JavaScript file, a 400+ line CSS file and a 100+ line HTML file, thus is not convenient to present here. Here I will briefly explain the website and provide some screenshots of the final product website.

Users can click on the left-hand side panel to select walk accessibility range. The color of each polygon was given based on Jenks natural color break classification method I implemented in JavaScript. The benefit of this method is that given a mandatory number of class breaks, it can generate a class break

at certain values so that the variation within each group is minimized while variation between each group is maximized. I implemented a 6-class break for apartment polygons according to the number of accessible restaurants within 5-minutes. The number of restaurants within each class are: 0-4 (lightest, yellow), 5-11, 12-24, 25-39, 40-56, 57-77 (darkest, red). For ease of comparison, I used the same class break for the 5-min walking boundary rather than implement a new class break for number of restaurants accessible within 10-min walking boundary, while added those buildings having more than 77 restaurants within 10-minute walking boundary as a new class and assigned it a darker red, thus 7 classes for 10-min walking boundary.





For specific method of use, the user can click on each single apartment polygon to trigger the walking boundary for this specific apartment. User can now see points representing all restaurants within this walking boundary highlighted with a different color. Everything updates as the user clicks on other polygons. The user can also see the number of reachable restaurants on the left-side panel, and details about the clicked apartment. Users can also hover mouse over restaurants within this walking boundary (but not restaurants without this boundary) to open popup, and upon clicking the point, the user can also retrieve information about this restaurant; everything will also update when user clicks on new ones and old records will not pile up on the panel.

One thing to note is that this functionality involves four GeoJSON files—the apartment GeoJSON containing all apartment polygons and detailed attributes (as is displayed on the left side panel), the restaurant GeoJSON containing all restaurant points and detailed attributes, the 5-min walking boundary GeoJSON containing walking boundary polygon for every apartment and a list of FIDs of all restaurants within this boundary, and a 10-minute walking boundary GeoJSON formatted in the same way as the 5-min one. The interaction between these four GeoJSON files are linked by unique apartment FIDs and restaurant FIDs.