Topic: Web Map on Student Off-Campus Housing Options and Their Related Urban Landscape Near UW-Madison Campus

Abstract

Off-campus housing is a major way of housing for UW-Madison students. However, information, especially geographic aspects of information that is important in the apartments searching process, is often not available in a well-synthesized way in many popular online apartment searching websites targeting Madison apartment rental market. In order to present apartment information in a more intuitive, better-synthesized way with an emphasis on geographic data in order to make apartment seeking easier for UW students, I collected data and made a web map using resources downloaded from online and a variety of computer and GIS technologies, such as Python Selenium library, Python regular expression, writing GeoJSON with Python, OSMnx, Google Maps API, ArcMap, Postgres database, jQuery, and Leaflet. In the process of making this map, I attempted to develop frameworks for visualizing different types of geographic data concerning apartment searching, and have developed a well-functioning website. The process creating this web map also shows the potential of future expansion of the functionalities of this web map to present more types of information that would be helpful for apartment seekers.

Table of Contents

1. Introduction

The City of Madison, especially around downtown and surrounding areas of UW Campus, has witnessed surge of high-rise apartment buildings in recent years. According to the report of "State of Downtown Madison 2017", the number of apartment units as increased by 35% since 2011 ("State of Downtown Madison 2017", p. 12). At the same time,

the vacancy rate of apartments has also been increasing since 2013, the vacancy rate of multi-family homes (with a vast majority being apartments for rental) has increased from the lowest vacancy rate of 2.56% in 2013 to the highest of 4.95% in 2017 ("State of Downtown Madison 2017", p. 13); even so, the increase in vacancy rate is unmatched by the increase in number of units, which implies that an increasing amount of people are now renting apartments or houses in downtown Madison.

With downtown Madison providing a large percentage of apartments amongst all off-campus housing oriented towards students and young-professionals working and studying at UW-Madison, the rise in the number of apartments and apartment vacancy rate is shifting downtown Madison apartment rental market away from a sellers' market to renters' market, and is certainly offering students and young professionals more choices when looking for housing. Apart from that, considering the prospective amount of student renters, new apartments constructed close to campus often take students' need into mind, and offer amenities that often cater to student need, including 12-month lease based on academic year, fully furnished apartment room convenient for settling-in, study rooms and on-site laundry.

Having all these features that cater to students' need is certainly a good thing for apartment seekers, and such benefit is further promoted by search operators available on several major rental websites targeting UW-Madison renters, including https://www.madisoncampusanddowntownapartments.com/search.j?, https://campusareahousing.wisc.edu/, https://www.rentcollegepads.com/off-campus-housing/madison/search, https://www.abodo.com/madison-wi/university-wisconsin-madison-apartments/campus, and https://www.apartments.com/madison-wi/, where apartment seekers are provided filters of various features, including price range, floorplans, apartment amenities

and pet policies that allow them to narrow down their selection. The following screenshot is the filter interface of Abodo, it shows how manifolded the filtering options can be:

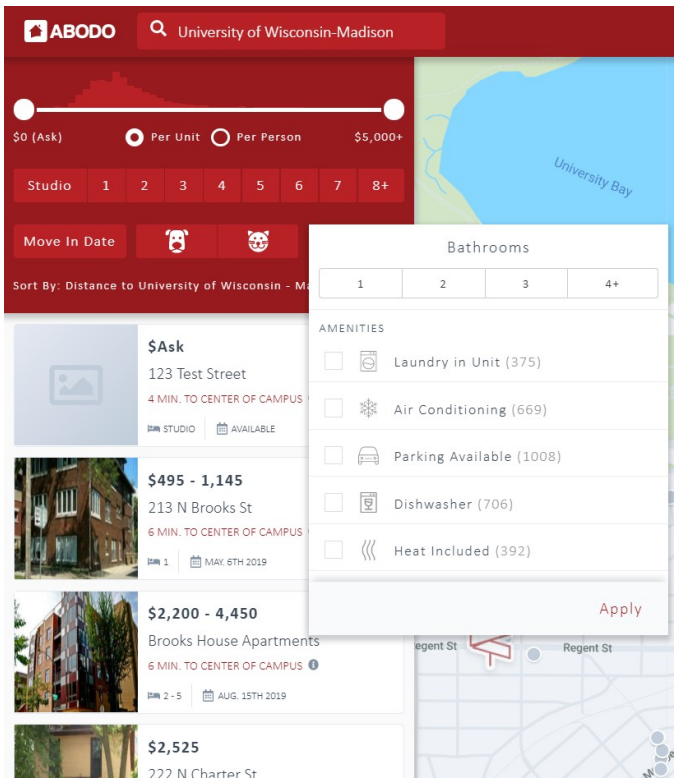


Figure 1

However, renters are often not well informed of information about the implication of location of each apartment, the surroundings of each apartment, and how different types of apartments are distributed over space—all of which should become an important consideration for people looking for housing. Apartment rental websites and individual rental webpages often talk about the location of these apartment in very general terms, some examples include “close to bus lines”, “close to supermarkets”, without giving renters specific impression of the surroundings and what does the location of the apartment bring them about. In this case, students would not able to know specific things like: how close is the apartment to a bus line? What is the frequency of the bus service? How convenient is it to

get to campus or libraries? How close the apartment is to different restaurants? Are these restaurants good or not? If the student frequently cooks for himself/herself, he/she might also want to know things like how close the apartment is to a supermarket so that he/she can get fresh vegetables and meat conveniently. Getting to know all these things requires the unveiling of urban landscape using GIS technology.

Apart from finding the most suitable place to live by uncovering proximity, insights on geographical patterns such as the distribution of certain floorplans and price is also important for student looking for housing. When retrieval functionality in an interactive map is applied appropriately, users can get just the amount of data they will need upon request. In the apartment search, if filter and re-expression are applied appropriately, necessary info can be quickly acquired and become helpful for students by using an interactive map.

To reveal these geographic elements in apartment seeking, I will make an interactive web map, with the aim of providing apartment seekers from UW-Madison a more comprehensive overview of apartment choices by visualizing key apartment features and by providing intuitive visual representation of spatial features—including surroundings and proximities of each apartment—which are often not included in details on several major apartment searching websites. Specifically, I will record and visualize on a web map key info about apartments including price, area for each floorplan available at each apartment, difference in travel time to campus, restaurants in areas surrounding apartments, and all bus routes that grant passengers access to campus; all of these features should aim to enforce a more comprehensive comparison of spatial elements of apartments.

2. Literature Review

2.1. On student off-campus housing

According to “Comprehensive Housing Market Analysis for Madison, Wisconsin”, during the 2014-2015 academic year, amongst 43,250 students, only 8100 live on-campus while a number of the rest of the students live off-campus (“Comprehensive Housing Market Analysis for Madison, Wisconsin”, p. 10). This analysis also estimated that students comprises about more than 35% of all renter households in the city of Madison(“Comprehensive Housing Market Analysis for Madison, Wisconsin”, p. 11). US News also mentioned that 75% of all UW-Madison student live off-campus. These facts make off-campus housing options around UW-Madison campus an important topic for both UW students and the city of Madison.

Also, several studies indicated the importance of location, especially the proximity to campus. The “2016 City of Madison Biennial Housing Report” stated in a section on student housing, that “proximity to campus is a top demand driver for students and increasing the opportunities for students to live close to campus should be a primary goal” (“2016 City of Madison Biennial Housing Report”, p. 12). In Muhammad Hilmy Muslim et al., in an overview of relevant studies about student housing, the authors indicated that “location and proximity to campus” is one of several important elements highlighted by those authors (2012 Muhammad Hilmy Muslim et al, p. 608). In Fields (p. 38, 2011), the author concluded that location is an important consideration for students to value their housing options. These studies support point that, for housing around campuses like UW-Madison, the location of apartments to campus is a key consideration

for students. Thus, I am implementing a travel-time-to-campus visualization and exploration functionality, and a surrounding-restaurants visualization and exploration functionality in my web map.

On specific feature of student apartments. Fields (p. 38, p. 2011) argued that students put high value on the number of bedrooms. Number of bedrooms is also an important information provided on a large number of apartment rental websites. Thus, I made sure in my web map that the number of bedrooms is clearly indicated and presented.

2.2. On web map

Schneiderman (1996) put forth a "Visual Information Seeking Mantra" that could be considered as a design philosophy for revealing insights into detailed information amongst big data: "Overview first, zoom and filter, then details-on-demand". In the design of my web map, I paid attention to adhere to this principle; I always ensured that the web map will present an overview of all information, allow the users to filter out information they don't need and highlight those they might be of concern, and then allow the users to retrieve more detailed information on individual features that they might be interested in.

3. Method

Because this project aims to develop from scratch a web map that visualizes rental options from scratch, this section documents the obtaining and processing of every piece of data used in the map.

3.1. Data

3.1.1. Data for apartments

This data is the core of this project. Having a dataset that records the apartment rental information as completely as possible is important for this project because an important goal of this map is to offer users, the apartment seekers, an overview—as complete as possible— of all off-campus rental information available.

3.1.1.1. Selection of rental resource websites

Because the time of collecting this data is in spring, when many apartments had already been rented out for the coming academic year and some of those resources had gone offline, my dataset might not be ideally complete.

An important issue is that there isn't a single website that hosts a comprehensive list of all apartments. There are several apartment rental websites that host leasing information for UW Campus area housing resources, ranging from the UW official rental resource collection—Campus Area Housing, to other commercial rental resource websites that host resources on a regional scale—with Madison area being their core market, or on a national scale and having Madison area apartments as a subset of their resources. However, because of the fact that each landlord can choose to lease with one, several, or even probably none of the websites, records of rental information are unevenly spread across those websites.

In this project, I attempted to develop a method that combined information from several rental websites in order to get a fuller list; however, because of time limit for this project and the large amount of work needed to develop algorithms that suit each individual website and additional manual corrections, I was only able to combine rental resources from two websites: https://www.madisoncampusanddowntownapartments.com/search.j? and https://www.rentcollegepads.com/off-campus-housing/madison/search. I selected these two websites first because amongst many websites that has rental listing in areas surrounding UW Campus, these two probably contain the most complete information on rental resources, especially high-rise apartments. Campus Area Housing on https://campusareahousing.wisc.edu/, is a UW official collection of rental resources that also maintains a database of roughly 700-800 rental information entries. This website contains many rental units that are not covered by the previous two websites. This figure shows the differences between Campus Area Housing apartments (shown as green circles) and apartment resources offered by the two rental websites I selected (shown as blue dots):

Figure 2

However, due to the limited time for completing this project, I was not able to include this dataset in the final web map. Even so, I have proofed the feasibility of combining resources from multiple websites and have developed a functional website that supports visualization of apartments (although with some minor bug and other deficiencies that I am not able to solve per my current level of programming skills), thus adding additional data in the future will just be a matter of time.

3.1.1.2. Scraping and processing data

The final product of this step is to be a csv file with several columns of attributes of apartments, including coordinates.

The methods for getting the data from these two websites are somewhat similar. The general idea is to get the HTML source code for each webpage that list rental resources and get URLs to each individual

webpage that contains detailed information for every single apartment. Next, after acquiring the HTML source code of each individual apartment webpage, I can extract essential information from the source code using regular expression, including fields like name of the rental (if there is one), address, price, area, URL, etc. and in the end clean up and correct the format of the data through python programming, and in the end, through manual correction to get rid of extra words in each cell, so that the file can be written into a new csv with formatting standardized, so that the basic properties can be further processed parsed into GeoJSON file that is ready to use in a web map. One thing to note is that all these steps, apart from the manual examination and correction of data in the last step, are done by Python programming.

For the first step, getting data from these two websites: https://www.madisoncampusanddowntownapartments.com/search.j? and https://www.rentcollegepads.com/off-campus-housing/madison/search is done by using Selenium and the re (regular expression) libs. There are multiple ways to do website scraping, however, for these two specific rental resource websites, Selenium is needed because the rental listing pages, which contain URLs to each specific apartment info webpage, from which we will scrape data, cannot be accessed using a specific URL, but rather, new contents are loaded only when user click on buttons on the website. Selenium is a web browser automation tool that is available in multiple programming languages including Python. It basically simulates

human web browsing behavior and can drive the browser to do things like clicking buttons and filling in forms. By implementing this tool, I was able to drive the browser to automatically flip through listing pages and get HTML source code for each page (see Appendix 1). One advantage of doing so is that, as there might be multiple listing pages on a rental website and they can be updated frequently, I can get the most up-to-date updates of this website at any time, without having to manually click through pages and copy and paste source codes.

For the second step, after getting the HTML source code, I can use the Python re lib to parse the source code and get a list of all URLs to each individual apartment page. Here comes into play another scraping tool—the Python requests lib. This is a lighter tool for website scraping than the Selenium lib. As long as the website does not strictly ban scraping, I can use the requests lib to get HTML source code without having to simulate human web browsing behavior using Selenium; this can be actually faster than using Selenium. By passing the URLs obtained in the first step as parameter in the requests function, I was able to get HTML source codes for each apartment page (see Appendix 2). Then I parsed each individual HTML source code and convert it into a csv file (see Appendix 3, 4), with the first several columns look like this:

	B	C	D	E	F	G	H	I	J	K	L
1	Address	Name	Latitude	Longitude	Type_of_rent	URL	Prc1	Prc2	Prc3	Prc4	Prc5
2	215 North Frances Street	The Frances	43.0713688	-89.395550	Apartment	https://www.rentcolle	1 Bedroom(s):$780-810#	2 Bedroom(s):$910#393sf			
3	1323 West Dayton Street	Vantage Point	43.0708486	-89.408515	Apartment	https://www.rentcolle	1 Bedroom(s):PRICE N/A	2 Bedroom(s):$2	3 Bedroom(s):PR	4 Bedroom(s):PR	5 Bedroom(s
4	508 Farley Avenue	Farley Arms	43.0726724	-89.433843	Apartment	https://www.rentcolle	1 Bedroom(s):PRICE N/A	2 Bedroom(s):$9	2 Bedroom(s)(Den):$895		
5	4 North Park Street	Park Regent A	43.0679294	-89.401265	Apartment	https://www.rentcolle	1 Bedroom(s):PRICE N/A	2 Bedroom(s):$1	3 Bedroom(s):$2	4 Bedroom(s):PR	5 Bedroom(s
6	110 South Brooks Street	College Statio	43.0660034	-89.402782	Apartment	https://www.rentcolle	1 Bedroom(s):$895#450s	2 Bedroom(s):$1	4 Bedroom(s):$2470#1345sf		
7	633 East Johnson Street	NO-NAME	43.0812965	-89.380379	House	https://www.rentcolle	2 Bedroom(s):$1495#14	4 Bedroom(s):$1495#1152sf			

Figure 3

For the address column, I change any unstandardized forms of address into standard formats, using the official Postal Service standard suffix abbreviations (https://pe.usps.com/text/pub28/28apc_002.htm), including changing “Street” to “St”, “Avenue” to “Ave”, “Court” to “Ct”, “Boulevard” to “Blvd”, etc.; I also changed “East” to “E”, “South” to “S”, “West” to “W”, “North” to “N”. I also added a city and state name column— ‘Madison, WI’ for the accuracy of Geocoding. After this is done, I send the csv file to Google Sheets, where there is a free add-on called “Geocode by Awesome Table”. I then selected the address and city/state columns, and was able to get all my addresses geocoded into latitudes and longitudes in decimal degree. Here I get a csv file with geocoded locations for further process.

3.1.13. Adding geometry info to the data and convert it GeoJSON format

The final product of this step is to be a GeoJSON file ready to be exported into a map. The structure of this GeoJSON file is specifically designed so that necessary data can be easily accessed within the JavaScript file.

However, before writing the files into GeoJSON, I will need to have a layer of polygons of all apartments or houses for rent. Most rental websites

only used symbols representing a geocoded point to show the location of the apartment, which can mis out spatial information that is important for apartment seekers. Apartment seekers will probably want to know how large area the apartment occupies, whether it occupies a whole block, a 2 or 3-storey high townhouse or a small detached single-family house. Using GIS, it is possible to get this more accurate data on apartment location and spatial property, and present users a more accurate representation of the reality.

To do this, I will need to get several GIS layers from this geospatial data repository, GeoData@Wisconsin at http://geodata.wisc.edu/opengeoportal/. Including a Madison municipal boundary shapefile from https://gisdata.wisc.edu/public/Madison_CityLimit_2017.zip, a polygon shapefile for all buildings in Dane County from https://gisdata.wisc.edu/public/Dane_Buildings_2018.zip. By using the overlay tool in ArcMap, I got all buildings within the boundary of the City of Madison. Then I loaded the geocoded csv into ArcMap using the "Convert Coordinate Notation" tool in the "Projections and Transformations" tool set. Next, I compared the points and with each building polygon and overlaid each point on the corresponding polygon. Because this polygon file does not have an address filed, I had to rely on external source to check whether each point is moved onto the correct polygon, like this:

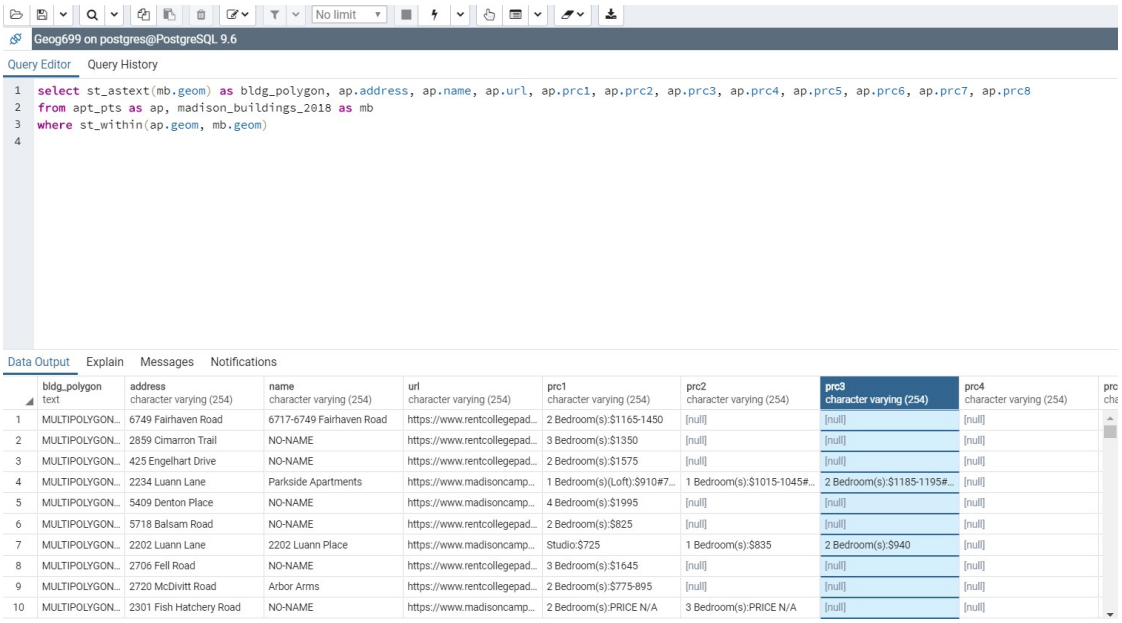
https://cityofmadison.maps.arcgis.com/apps/webappviewer/index.html?id=8b840c56c3794af5ab3d68aca9ce901f&query=Parcels, an official ArcGIS application that hosts all Madison parcel data as a reference, although later I found that I can also use this Madison parcel shapefile from City of Madison Open Data at http://data-cityofmadison.opendata.arcgis.com/datasets/tax-parcels-assessor-property-information. For points whose address format do not conform with the parcel detail displayed on the website, I changed the address to the same as what is displayed on the parcel info website to make my records as standardized as possible (For example, one address column comes from scraped data displays "134 State Street", yet the parcel info website shows "134-136 State Street" for the corresponding parcel, I will change the address to the later one). I then used the overlay tool in ArcMap to check if all each corrected point is within one polygon, and correct those that are not.

The next step is to import the polygon layer and the point layer into PostgreSQL and perform an ST_Within() function using a query shown below. After the query, the apartment property data will be correlated with corresponding building polygons. Then I used a Python program to generate a GeoJSON file (see Appendix 6). In this Python program, I also combined two fields of data, from the Madison Tax Parcel Assessor Property Information table from http://data-cityofmadison.opendata.arcgis.com/datasets/tax-parcels-assessor-property-

information that was exported as a txt file, with corresponding apartment records; these two columns are—

1. the type of each property, for most cases, this data will note the number of units in each building, if it's a residential building; I am including this field because there are no official ways to distinguish specific types of building, whether it's a townhouse, a duplex, an apartment complex, or something else. In this official parcel assessor record, the number of units are included under a large category—apartment. Availability of info of different types of housing is important for a wide range of student renters, who may have different preferences in choosing housing; and such info can be first distinguished by number of unit, and students can be further directed to detailed info of apartments with that basic information in mind.

2. the year the building was built. This field should be important because in recent years, a lot of new high-rises have been built in Madison downtown area, equipped with brand-new amenities and features. While according to the data I scraped, the prices per person of some of their floorplans are not necessarily higher than other rentals. Students might probably want to know where these newly constructed high-rises are and how new they are.



	bldg_polygon text	address character varying (254)	name character varying (254)	url character varying (254)	prc1 character varying (254)	prc2 character varying (254)	prc3 character varying (254)	prc4 character varying (254)
1	MULTIPOLYGON...	6749 Fairhaven Road	6717-6749 Fairhaven Road	https://www.rentcollegepad...	2 Bedroom(s):$1165-1450	[null]	[null]	[null]
2	MULTIPOLYGON...	2859 Cimarron Trail	NO-NAME	https://www.rentcollegepad...	3 Bedroom(s):$1350	[null]	[null]	[null]
3	MULTIPOLYGON...	425 Engelhart Drive	NO-NAME	https://www.rentcollegepad...	2 Bedroom(s):$1575	[null]	[null]	[null]
4	MULTIPOLYGON...	2234 Luann Lane	Parkside Apartments	https://www.madisoncamp...	1 Bedroom(s)(Loft):$910#7...	1 Bedroom(s):$1015-1045#...	2 Bedroom(s):$1185-1195#...	[null]
5	MULTIPOLYGON...	5409 Denton Place	NO-NAME	https://www.madisoncamp...	4 Bedroom(s):$1995	[null]	[null]	[null]
6	MULTIPOLYGON...	5718 Balsam Road	NO-NAME	https://www.rentcollegepad...	2 Bedroom(s):$825	[null]	[null]	[null]
7	MULTIPOLYGON...	2202 Luann Lane	2202 Luann Place	https://www.madisoncamp...	Studio:$725	1 Bedroom(s):$835	2 Bedroom(s):$940	[null]
8	MULTIPOLYGON...	2706 Fell Road	NO-NAME	https://www.rentcollegepad...	3 Bedroom(s):$1645	[null]	[null]	[null]
9	MULTIPOLYGON...	2720 McDivitt Road	Arbor Arms	https://www.rentcollegepad...	2 Bedroom(s):$775-895	[null]	[null]	[null]
10	MULTIPOLYGON...	2301 Fish Hatchery Road	NO-NAME	https://www.madisoncamp...	2 Bedroom(s):PRICE N/A	3 Bedroom(s):PRICE N/A	[null]	[null]

Figure 4

3.1.1.4. Structure of the apartment GeoJSON file

The structure of the apartment GeoJSON is designed in a way that different columns of data are clearly organized and can be retrieved with ease when needed.

```
    "properties": {
      "Address": "520 E Johnson St",
      "Year_built": 1885,
      "Detailed_type": "5 unit Apartment",
      "Simplified_type": 5,
      "FID": 700,
      "Name": "NO-NAME",
      "Type_of_housing": "House",
      "URL1": "https://www.rentcollegepads.com/city/madi
      "URL2": "https://www.madisoncampusanddowntownapartr
      "Floorplans": {
        "Floorplan1": {
          "Title": "2 Bedroom(s)",
          "Price": [
            1050,
            1350
          ],
          "Area": [
            "N/A"
          ],
          "Price/Unit_area": [
            "N/A"
          ],
          "Individual_Average": {
            "avg_price": 600,
            "avg_area": "N/A",
            "avg_price_unit_area": "N/A"
          }
```


Figure 5

This what the property field of a feature in this GeoJSON file looks like up to this step. For every apartment feature, its properties include basic information: address, year constructed, FID (which is a unique primary key for every apartment), type of the rental (whether it's a housing or a building), URLs to apartment rental websites. The "floorplan" field includes all floor plans of the apartment; each floor plan includes: title (name of the floor plan), price range, area range, price per square feet range, average price per person, and average area per person. The average price per unit area (per square feet) will be calculated as price/area, and if a range in either price or area exists, I used the highest-price/largest-area, and lowest-price/smallest-area as the range of the price/unit_area field. The average prices per person are calculated as (lowest price + highest price) / 2 / number-of-rooms, the average areas are calculated as (smallest area +

largest area) / 2 / number-of-rooms. For price per unit area, price per person, and area per person, if either price or area is N/A, the field that requires at least one of these two for calculation will be marked as N/A too. If a floor plan contains a sub-field like den or penthouse, it will be added as an "Extra Feature" of a floorplan, and each extra feature will have its own price range, area range, and price per unit area range.

3.1.2. Data for restaurants and walking boundary

These two layers—restaurants and walking boundary are closely connected, because the purpose of this functionality is to visualize the number of surrounding restaurants of each apartment within 5-minute and 10-minute walking distance, which hopefully can be an important reference for students to compare differences between surrounding amenities of those many apartments. I selected restaurants as one type of amenity, and developed a model so that representing more types of amenities that might be of interest to apartment finders—like gyms, convenience stores, and supermarkets—would be possible in the future.

This part was originally done for the final project of Geog 675 that I am taking this semester. The spatial analysis of restaurant data and 5/10 min walking boundary map could not be done without skills I learnt in that course. Because the Geog 675 project and this web map project are closely related, I will introduce the how the data acquired from that class was integrated into this web map; however, because the spatial operation and analysis of data was not done for this class, I will not go

deep into the technical specifications of how I processed the data into a new layer containing walking range polygons.

3.1.2.1. Restaurant Data

All restaurant data for the city of Madison comes from Yelp. I scraped 990 records of data. 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, a browser automation tool available in Python (the HTML source code obtained using the Python requests library will often be incomplete). After geocoding and parsing, I got a GeoJSON file with each feature look like this:


```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [
          -89.3862015,
          43.0742022
        ]
      },
      "properties": {
        "FID": 0,
        "address": "18 N Carroll St,Madison WI 53703",
        "name": "Graft",
        "url": "https://www.yelp.com/biz/graft-madison",
        "opening_hours": {
          "Mon": [
            [
              "Closed"
            ]
          ],
          "Tue": [
            [
              "5:00pm",
              "10:00pm"
            ]
          ],
          "Wed": [
            [
              "5:00pm",
              "10:00pm"
            ]
          ],
```


Figure 6

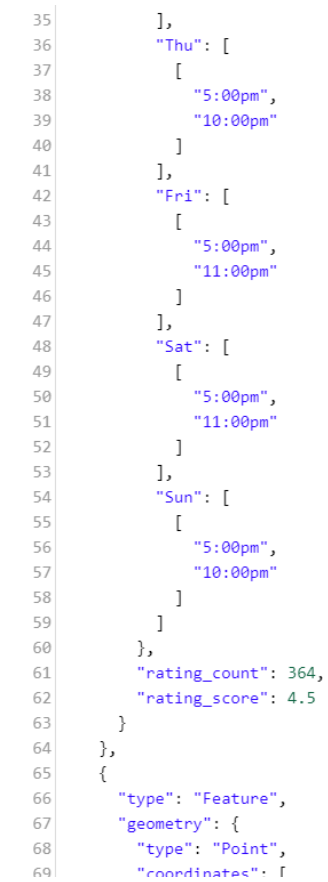


Figure 7

The properties of each properties field contain: 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), opening hours every day of the week, rating count and rating score.

3.1.2.2. Walking boundary data.

I collected 5-minute and 10-minute walking boundary for every apartment using OSMnx. Walking boundary polygon of 5-minute walk for all polygons look like this:

Figure 8

3.1.2.3. Restaurants within each boundary data.

I conducted spatial overlay using the GeoPandas library for Python and got a list of all restaurants within each 5-minute and 10-minute walking boundary polygon for all apartments. Then I appended the list of FIDs of restaurants within boundary as a new property for each feature to the walking boundary GeoJSON file, so that I can link the walking boundary polygon to retrieve information of corresponding restaurants within the boundary.

3.1.3. Data for travel time to campus.

For this step, I collected the on-foot and on-bike travel time from each apartment to four UW campus landmarks—the Gordon Dining Hall, Bascom Hall, Union South, and UW Natatorium. The data was obtained using Google Maps Directions API. I used Shapely, a Python library for analyzing geometric objects to get the coordinates of centroids of each apartment polygon as the start point of the trip, and used the coordinates of these four campus landmarks as the end point of the trip.

Then I parsed the URL for getting each travel time information and used the Python requests library to get response from Google Maps API server. A sample request URL looks like this:

https://maps.googleapis.com/maps/api/directions/json?origin=**<latitude,longitude>**&mode=walking&destination=**<latitude,longitude>**&key=**<USER'S KEY>**

In this URL, the bolded texts need to be replaced with actual values.

The responses from Google Maps API are in JSON format. Then I parsed response columns that are relevant for the task—travel time and distance, and added the parsed data into the apartments GeoJSON file. The newly added property looks like this:

```
},
"to_campus": {
  "to_gordon_walk": [
    "1.1 mi",
    "23 mins"
  ],
  "to_gordon_bicycling": [
    "1.4 mi",
    "9 mins"
  ],
  "to_bascom_walk": [
    "1.3 mi",
    "27 mins"
  ],
  "to_bascom_bicycling": [
    "1.8 mi",
    "12 mins"
  ],
  "to_unionS_walk": [
    "1.6 mi",
    "32 mins"
  ],
  "to_unionS_bicycling": [
    "1.6 mi",
    "10 mins"
  ],
  "to_nat_walk": [
    "2.2 mi",
    "44 mins"
  ],
  "to_nat_bicycling": [
    "2.3 mi"
```


Figure 9

3.1.4. Bus route data

Bus routes are important for students who live far from campus. In this case, I collected all bus lines having at least one stop within the boundary of UW campus and all the stops on these lines. The bus lines and stops data were collected from City of Madison Open Data. The campus boundary, which is an approximate boundary for future campus area plan, is also collected from City of Madison Open Data; this is the only available campus boundary shapefile that I could find. Then I did a 65-meter buffer of the boundary so that closer bus stops can also be included. The selected routes and stops are then transformed into two GeoJSON files on mapshaper.org. The properties field of each feature in the bus route GeoJSON contains URL to corresponding webpage on Madison

Metro Transit for this route, so that users can use this URL to discover more detailed information; it also contains the route name. The properties field of each feature in the bus stop GeoJSON contains a unique stop ID, stop name, and bus lines that pass through this stop.

4. Web map—design methods and results

The web map is the final product of this project. The web map is available on GitHub at https://a-yifeng.github.io/Geog699_apartments_web_map/. The web map makes use of all the data collected in section 5.2 and presented them to the users.

I chose Carto Voyager—a free base map released by Carto. This base map has an overall color theme that is light in color values and low in color saturation; this characteristic makes it easier for polygons, lines, markers and circles with higher color values and saturations to stand out and occupy a high visual hierarchy. This base map does not have an exceeding amount of text and other map features, which also makes it easier to let my data occupy a higher visual hierarchy.

The web map interface is divided into four parts that are all accessed from a main menu. On the main menu interface, a base map with apartment polygons is displayed when the user is on the main menu; when the user clicks on one of the four green buttons, the map will be updated with new features to display; there is also a back-to-menu button on each functionality page, when the user clicks on this button, all contents displayed on the map will be cleared.

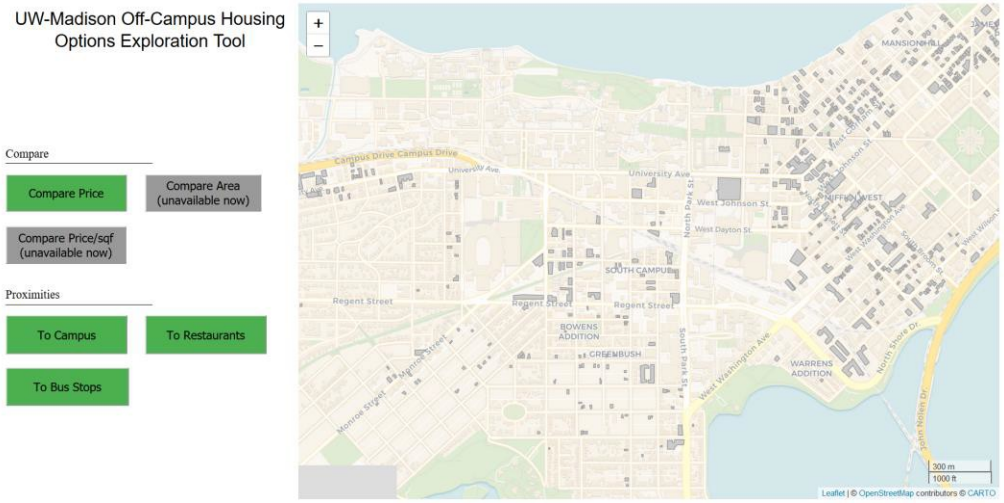


Figure 10

Each one performs one functionality in the conceptual design—the first one is the price comparison that allows users to compare pricing between different apartments and floor plans, the second one is an overview of surrounding areas of each apartment, the third one is the proximity to campus, and the last one is the proximity to bus stops.

For the first part—price comparison window, I designed the interface like this:

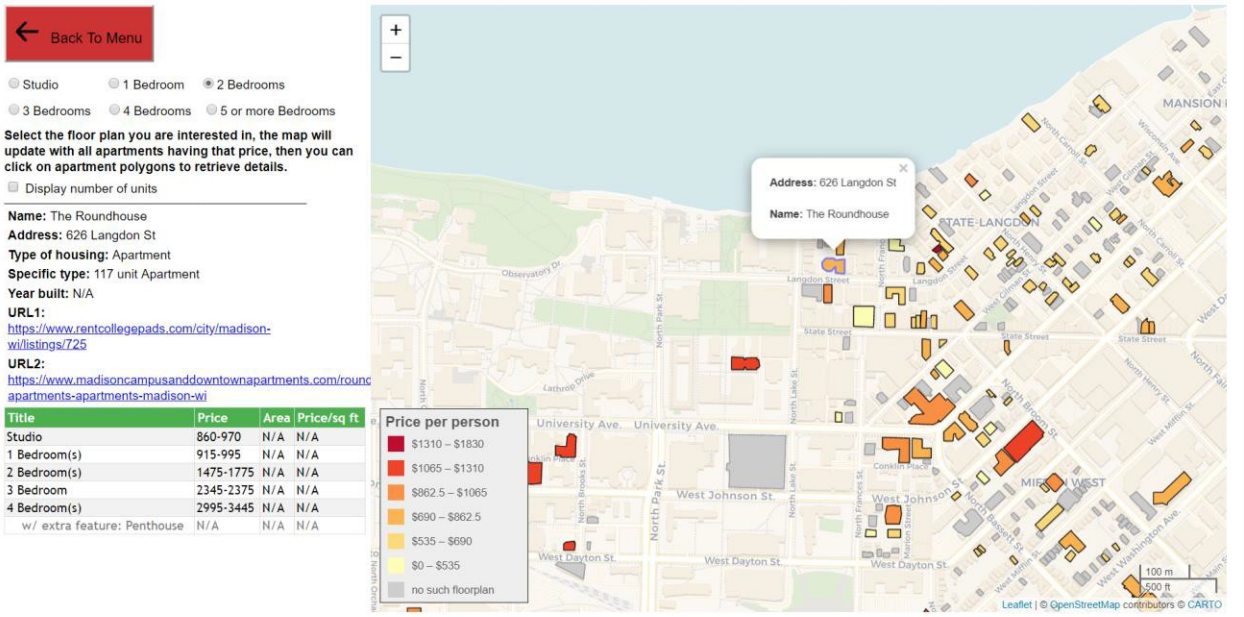


Figure 11

The user must first select a floor plan in the radio button. Radio button is a graphical control element that only allow people to select one in a group of options. By clicking one radio button, all the apartment having that floor plan is given a color; all apartments that do not have this floor plan will remain gray; apartments that do have this floor plan but do not have a price area also remain gray.

The color rank is given based on the position of the aparment's price in per-person prices of all floor plans of all apartments. Classifying numeric values of spatial data is necessary for representing them on a map and visually understanding them (Osaragi, 2002). Because whether the per-person price of a rental unit is cheap or expensive, and what is the specific level of price, depends on the distribution of prices of all rental units, rather than by any arbitrarily defined break in the amount of rent (for example, I cannot arbitrarily claim that $600 is cheap, $800 is intermediate, and $1000 is expensive—$800 might be about 20% higher than the average per-person prices for all apartment units in Madison, while might be really cheap if someone is renting a similar unit in New York; such criteria should depend on how all prices are actually distributed), I need to use some statistical methods to make each class in the classification more representative. Here, I used Jenks natural color break to break the prices of all per-person prices into 6 groups. Jenks color break minimizes variation of values in each classified group (Jenks 1967). By implementing a Jenks natural color break, each color can better represent each price range, with all prices within each category having the smallest possible variation, while avoid the risk of having values very different from each other being placed into one class by implementing an arbitrary break at, for example, every $100 (for instance, this method avoids the risk of having 20 floor plans priced at $601 per

person and 20 floor plans priced at $699 per person being classified into one category, with very few floor plans having prices between $601 and $699, while another 15 floor plans priced at $701 categorized into another category (in which case, obviously placing these 20 floor plans at $699 and 15 floor plans at $701 in one class would enable the classification better represent the actual price level and distribution of prices)).

The six colors that are used to represent the break used “6-class YlOrRd” obtained from ColorBrewer at http://colorbrewer2.org, a website that offers some nice color combinations for color ramps needing different colors and number of classes. In the end, a 6-class color ramp from light yellow to dark red would sequentially represent classes of apartment having prices ranging from $0-535, $535-690, $690-862.5, $862.5-1065, $1065-1310, $1310-1830. The user can use colors of different apartment polygons to get an overview of price distribution, and the approximate level of per-person price. For example, when the user selects “1 bedroom”, he will see this:

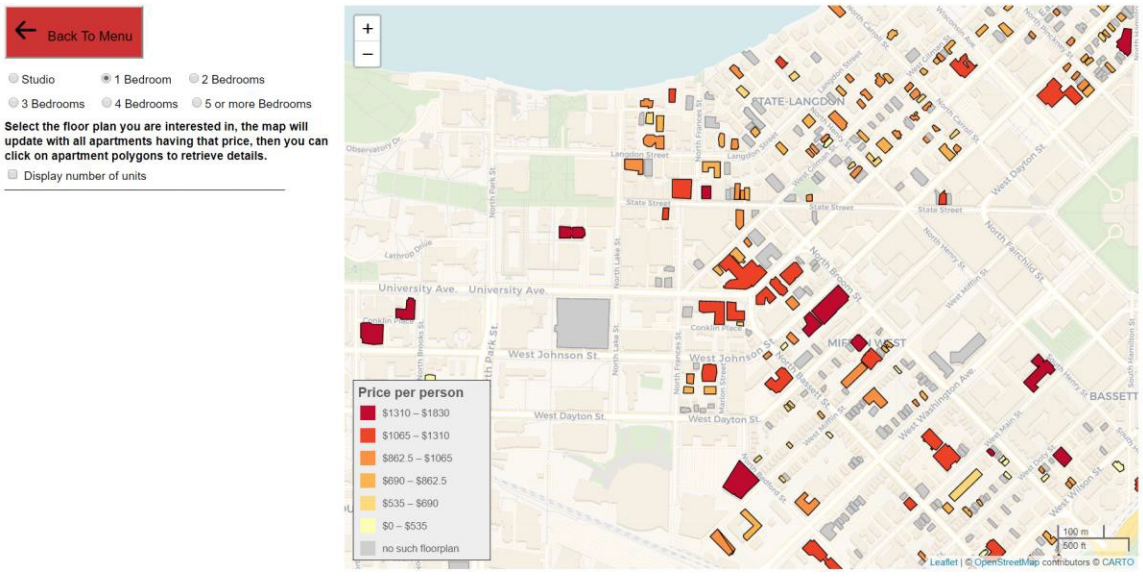


Figure 12

While when he selects “3 bedrooms”, he would see this:

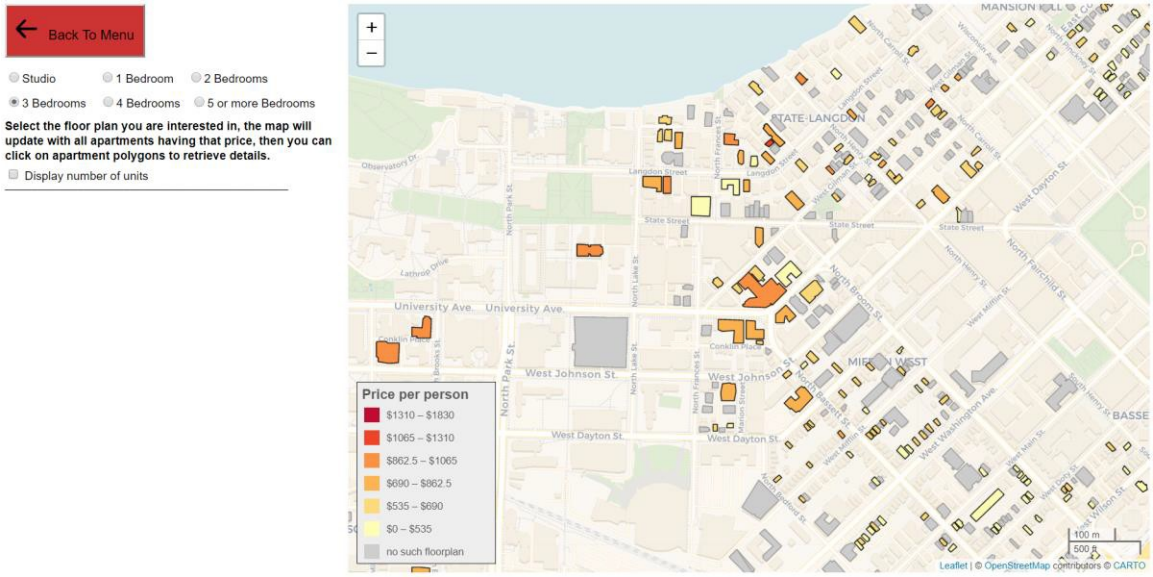


Figure 13

The distinction between these two screenshots clearly shows that the per-person prices of 1-bedroom units more tend to be represented by darker red colored polygons in the downtown area, while the per-person prices of 4-bedroom units more tend to be represented by orange or light yellow colored polygons in the same area. In this way, the users can use this functionality to get an overview of price distribution of all floor plans. One thing to note is that floor plans with some extra features like den or penthouse, and those with more complicated pricing policies like different pricing for single and double are not included in this classification, because implementing this will be complicated—their information are available in the price window on the side window.

If the user discovers some cheaply priced apartment, he can click on an apartment polygon to retrieve more detailed information. The apartment will be highlighted upon mouse hovering, and back to original when he moves mouse out of the polygon area. For

now, I did not implement highlighting upon click, because it involves resetting the entire map when the user needs to de-highlight it, and this interactive functionality is hard to achieve; but as a functionality that could help users to focus on their interested features, I will implement it later. Currently, upon user click, the info window on the left side will be updated with detailed information of the apartment, including name, address, type, year constructed, URLs to rental websites, and a table listing all floorplans, their corresponding price, area and price/square feet. All units listed under a certain number of bedrooms that has extra features like den or penthouse, or different pricing policies for use as single and double, are shown like this, with gray text and a header "w/ extra features:" in the title part.

Title	Price	Area	Price/sq ft
Studio			
w/ extra feature: Single	1250	N/A	N/A
w/ extra feature: Double	1600	N/A	N/A
1 Bedroom(s)			
w/ extra feature: Single	1350	N/A	N/A
w/ extra feature: Double	1700	N/A	N/A
2 Bedroom(s)			
w/ extra feature: Single	1700	N/A	N/A
w/ extra feature: Double	2200	N/A	N/A
3 Bedroom(s)			
w/ extra feature: Single	2550	N/A	N/A
w/ extra feature: Double	3300	N/A	N/A
4 Bedroom(s)			
w/ extra feature: Single	3400	N/A	N/A
w/ extra feature: Double	4400	N/A	N/A

Figure 14

The users can also check the "Display number of units" checkbox to show the number of units of every apartment. This can be helpful for students interested in some specific types of housing (townhouse, single-family homes, or large apartment complexes).

When the user clicks on the "Back to menu" button, all info currently displayed on the info window and on the map will be wiped from screen, and the user can select a new functionality to explore.

The “To campus” functionality is another functionality directed from the main menu. A working interface looks like this:

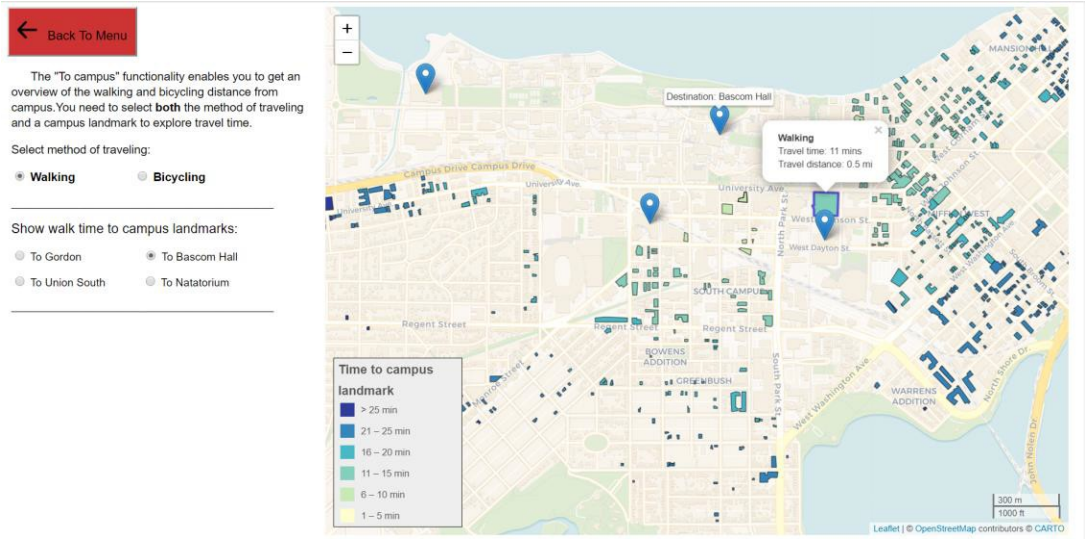


Figure 15

The user will read the prompt on the window upon entering this interface. When they first selected a method of traveling and then a destination landmark, they will see the destination with a tooltip on top of it, and all the apartments will be given a color based on the travel time to this destination, either by walking or by bicycling, based on user selection. The color ramp is based on travel time, and every 5 minutes is a class (the starting time is 1 minute, because it is the minimum travel time that can be obtained from Google Maps API. The highest class is for those whose travel time is beyond 25 minutes, beyond which the travel time is not classified, as for any time longer than 25 minutes, students will likely take public transportation for commute. The user will be able to have an overview of time range that it would take them to get from each apartment to a campus landmark. The user can also hover over each apartment polygon and retrieve details of traveling for every single apartment to a designated campus landmark. For an integrated user experience, I also enable

the user to retrieve information about each apartment upon click, if they see the travel time is within a desirable range and want to know more about the apartment.

The “To restaurants” functionality, another major functionality, enables students to discover restaurants within 5-minute and 10-minute walking range.

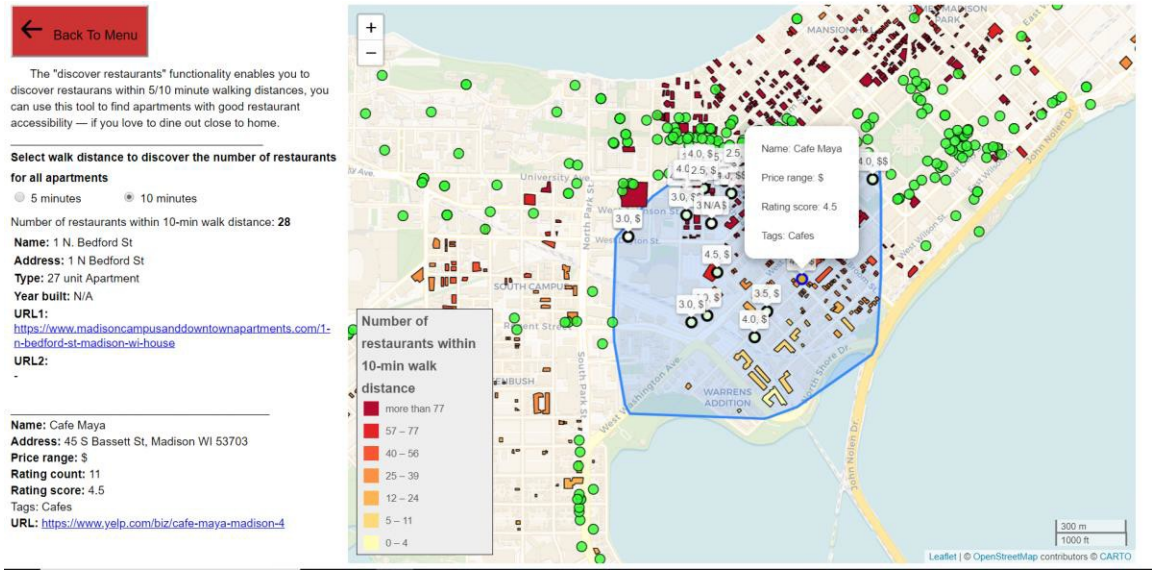


Figure 16

Upon entering the interface, the user will select 5 minutes or 10 minutes based on prompt. Then each apartment will be assigned a color based on number of restaurants within selected walking distance, with light yellow representing less restaurants within walking distance, while dark red representing more restaurants within walking distance. The legend bar will also load based on 5-minute or 10-minute walking boundary. The class break used Jenks natural break based on the number of restaurants within 5-minute walking boundary. For ease of comparison, a same class break is applied to number of restaurants within 5-minute and 10-minute walking boundary; in this case, the legend for 5-minute walking boundary has 6 classes, while the legend for 10-minute walking boundary has one more

class—those apartments having more than 77 restaurants (the largest number of accessible restaurants of 5-minute walking boundary of all apartments).

The user can click on each apartment polygon to retrieve details of the apartment on the left-side window, and display the boundary calculated using OSMnx on the map. The restaurants within this boundary will be highlighted using a different color and have a tooltip on top displaying ranking scores and price level; I chose these two things to show because they are probably some important first things to consider about a restaurant for students—first students will want to know is a restaurant is rated good or not, and second they want to know whether the restaurant is cheap enough for everyday meal. The exact number of restaurants will also display on the side window. Users can also click on restaurant points within the boundary to retrieve detailed information about the restaurant. The map will update with new boundaries and restaurants within the boundary when the user clicks on a new apartment polygon or a new restaurant point.

The last major functionality is a bus route explorer. As introduced in the data processing stage, each bus route here has at least one stop within a 65-meter buffered polygon area of campus area; also, each bus stop displayed here has at least one route going to campus area. The user can explore the bus routes by interacting either with bus stop symbols map or bus line radio buttons on the side window.

For bus stop symbols, the user can hover cursor over a bus stop symbol to show a tooltip containing all bus routes at this stop. If they click the symbol, all bus routes that pass through this stop will be displayed on the screen; and the routes will update when they click on new bus stop symbols. Here, I applied a low opacity value (0.4), so that if multiple routes overlay

on top of each other, the user will see the color of the route is less opaque, which can to certain extents indicate the approximate number of bus lines on this road.

For bus route radio buttons, it is displayed on the side window, with a “Show on map” radio button and a URL that links to the webpage of this route on Madison Metro Transit website for each route. When the user clicks on one radio button, the map will update with corresponding line.

This functionality will help users who are looking for housing farther-off from campus to discover bus route accessibility more intuitively. For a better-integrated user experience, adding a side window that displays detailed info of each apartment upon user request will still be necessary (after all, the goal of the user is to find apartments); however, because the bus route radio buttons are occupying a large part of the screen real-estate and cannot be simply removed, the integration of such an apartment info window will need some special processing, which hopefully, will be improved in the future.

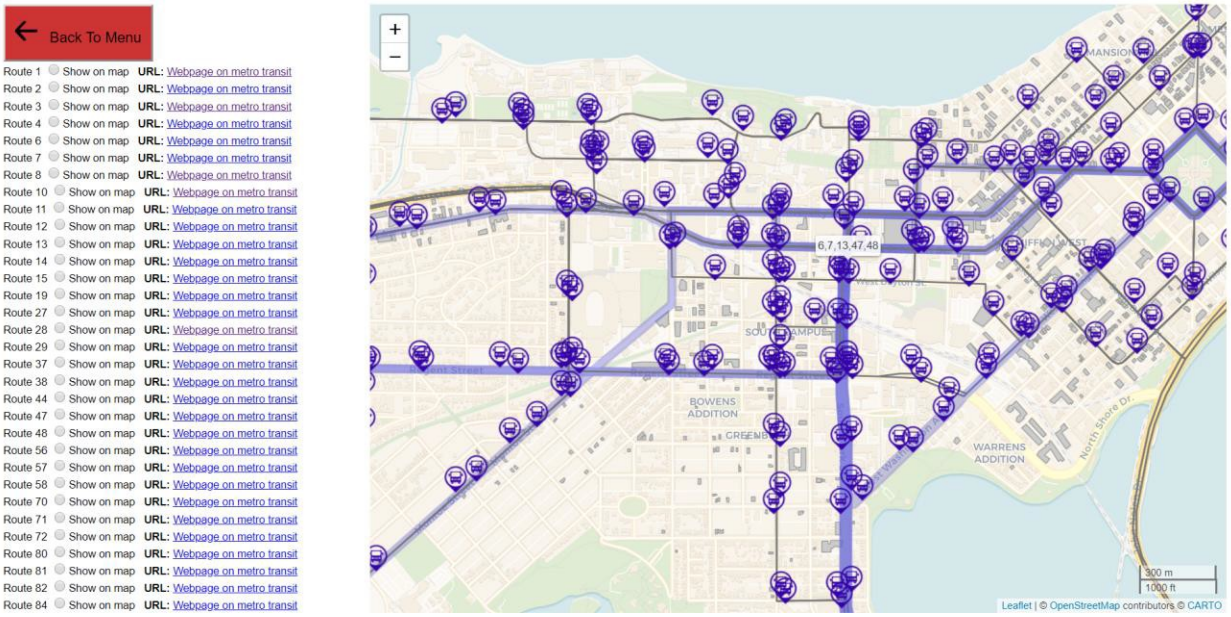


Figure 17

5. Conclusion

The initial goal of this project focuses on creating a reference map for students looking for housing. Reflecting on the goals in the conceptual development stage, several utilities have already been realized and frameworks for each functionality have been built, and they are presenting a useful reference for the users—student apartment seekers.

In the data collection process, I was able to collect various kinds of data, data on basic info, floor plans and prices of apartments (796 records), restaurant data (990 records), travel time data and bus route data; I was also able to process them into GeoJSON so that these different types of data can be imported to a Leaflet web map. The process of collecting apartment data was the most time-consuming part of the project, as the quality of data available on some rental websites are poor and the formatting of data was not standardized;

because of this, specific Python programs needed to be written to standardize the data formatting before parsing the data into GeoJSON; manual correction was also needed. Due to the difficulty in parsing the data into GeoJSON and the limited time for this project, I was only able to collect about 2/3 of all apartment data and only data for restaurants amongst all kinds of amenities. More work is required in the future to collect a more complete dataset to make this tool more useful.

For the web map, I was able to represent the information on each apartment, especially the floor plans information, on the web map, in an intuitive way for the users. The information on each apartment's proximity to restaurants and campus is also represented on the web map in a satisfactory way. The bus line exploration functionality is working well, but it is currently just a tool for straightforward exploration and representation of bus route into, and more work needs to be done to make it truly useful for students looking for apartments.

Given more time, I would also work on visualizing differences between per-person areas and price/square foot of different rental units, as well as visualizing crime data. The current framework of the project proofs the potential of representing these new features.

6. Bibliography

C1 Street Suffix Abbreviations - Postal Explorer - USPS.com, https://pe.usps.com/text/pub28/28apc_002.htm. Last retrieved 4/22/2019.

City of Madison Property Lookup Application, https://cityofmadison.maps.arcgis.com/apps/webappviewer/index.html?id=8b840c56c3794af5ab3d68aca9ce901f&query=Parcels. Last retrieved 5/2/2019.

ColorBrewer, http://colorbrewer2.org

Comprehensive Housing Market Analysis for Madison, Wisconsin. 2015.

Google Maps Directions API. https://developers.google.com/maps/documentation/directions/start. Last Retrieved 5/1/2019.

Fields, T. J. (2011). A hedonic model for off-campus student housing: the value of location, location, location.

Jenks G F. (1967). "The Data Model Concept in Statistical Mapping", *International Yearbook of Cartography, Vol.7*, pp.186-190.

Muslim, M. H., Karim, H. A., & Abdullah, I. C. (2012). Satisfaction of Students' Living Environment between On-Campus and Off-Campus Settings: A Conceptual Overview. *Procedia-Social and Behavioral Sciences*, 68, 601-614.

Osaragi, T. (2002). Classification methods for spatial data representation.

Schneiderman, B. (1996). The Eyes Have It - A Task by Data Type Taxonomy for Information Visualization.

Tax Parcels – City of Madison Open Data, http://data-cityofmadison.opendata.arcgis.com/datasets/tax-parcels-assessor-property-information. Last Retrieved 4/25/2019.

US News. https://www.usnews.com/best-colleges/university-of-wisconsin-3895/student-life. Last Retrieved 5/2/2019.

UW Campus Plan – City of Madison Open Data, http://data-cityofmadison.opendata.arcgis.com/datasets/uw-campus-plan. Last Retrieved 4/27/2019.

2016 City of Madison Biennial Housing Report. 2016.

7. Appendix

7.1. Appendix 1: Python program for getting HTML source code of apartments from cdliving.com and parsing into csv file

```
import re
import requests
from requests.exceptions import RequestException

from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException

browser = webdriver.Chrome('C:/Users/osvob/Downloads/chromedriver_win32/chromedriver.exe')
wait = WebDriverWait(browser,25)

def search(pageNum, url, totPage):
    try:
        if pageNum >= 1:
            if (pageNum < 6):
                submit = wait.until(EC.element_to_be_clickable((By.CSS_SELECTOR,
'#searchResultsListContainer > div.paginationHolderBottom.text-center > nav > ul > li:nth-child(12) > a')))
                submit.click()
                print('1: ',pageNum)
            elif ((pageNum >= 6) and (pageNum <= totPage - 7)):
                submit = wait.until(EC.element_to_be_clickable((By.CSS_SELECTOR,
'#searchResultsListContainer > div.paginationHolderBottom.text-center > nav > ul > li:nth-child(13) > a')))
                submit.click()
                print('1-2: ',pageNum)
            elif (pageNum == totPage - 6):
                submit = wait.until(EC.element_to_be_clickable((By.CSS_SELECTOR,
'#searchResultsListContainer > div.paginationHolderBottom.text-center > nav > ul > li:nth-child(11) > a')))
                submit.click()
                print('2: ',pageNum)
            elif (pageNum <= totPage):
                selector = '#searchResultsListContainer > div.paginationHolderBottom.text-center > nav > ul >
li:nth-child(' + str(pageNum - 13) + ') > a'
```

```
            print ('3: ',pageNum)
            submit = wait.until(EC.element_to_be_clickable((By.CSS_SELECTOR, selector)))
            submit.click()
        else:
            submit = wait.until(EC.element_to_be_clickable((By.CSS_SELECTOR, '#searchResultsListContainer > div.paginationHolderBottom.text-center > nav > ul > li:nth-child(12) > a')))
            submit.click()
            submit = wait.until(EC.element_to_be_clickable((By.CSS_SELECTOR, '#searchResultsListContainer > div.paginationHolderBottom.text-center > nav > ul > li:nth-child(1) > a')))
            submit.click()

        pageHTML = browser.page_source
        print('will call PARSE', pageNum)
        parseEachPage(pageHTML, pageNum, totPage)

    except TimeoutException:
        print ('Have tried for ', pageNum, ' times.')
        search(pageNum,url,totPage)

def get_one_page(urlFull):
    try:
        response = requests.get(urlFull)
        if response.status_code == 200:
            return response.text
        else:
            return None
    except RequestException:
        return None

def parseEachPage(pageHTML, pageNum, totPage):

    pattern = re.compile('<h3><a href="(.*?)" onclick=', re.S)
    urlPart = re.findall(pattern, pageHTML)
    print("URLPART", urlPart)

    addrL = []
    typeL = []
```

```
numPriceL = []
titleL = []

priceD_L = []
urlL = []

if pageNum < totPage:
    pageItems = 30
else:
    pageItems = 21
for i in range(0, pageItems):
    print(urlPart[i][-6:])
    if (urlPart[i][-6:] == '-house'):
        # STORAGE
        typeL.append('House')
    else:
        # STORAGE: typeL
        typeL.append('Apartment')

    urlFull = 'https://www.madisoncampusanddowntownapartments.com/' + urlPart[i]
    eachAptHtml = get_one_page(urlFull)
    urlL.append(urlFull)

    addrPattern = re.compile('<a class=\'showMap yesprint\'.*?href=\'javascript:;\'>(.*?)</a>')
    addrThisApt = re.findall(addrPattern, eachAptHtml)

    # Get apartment ADDRESS:
    addrThisApt[0] = (addrThisApt[0].replace(' ', ' ')).strip()
    subStr = addrThisApt[0]
    print('SUBSTR-1:', subStr)
    subStr = subStr.replace(',','')
    if (subStr[-2:] == 'WI'):
        subStr = subStr[:-11]
    elif (subStr[-1].isdigit()):
       subStr = subStr[:-17]
    print('SUBSTR', subStr)
```

```
# Get apartment NAME:
titlePattern = re.compile('<h1 class.*?>(.*?)<')
titleThisApt = re.findall(titlePattern, eachAptHtml)[0]
if titleThisApt == subStr:
    titleL.append('NO-NAME')
else:
    titleL.append(titleThisApt)

# STORAGE: addrL
addrL.append(subStr)

priceReg = ''
priceListPattern = re.compile('<ul class=\'arrow_list\'(.*?)</ul>', re.S)
priceListHtml = (re.findall(priceListPattern, eachAptHtml))[0]

print('PRICELIST_HTML: ',priceListHtml, type(priceListHtml))

# STORAGE: numPriceL
numPrice = priceListHtml.count('<li')
numPriceL.append(numPrice)

for i in range(0, numPrice):
    priceReg += '<li.*?>(.*?)<.*?'

pricePattern = re.compile(priceReg, re.S)

priceT = re.findall(pricePattern, priceListHtml)[0]
print('PRICE-T', priceT, type(priceT))

priceL = []
priceL_F = []
priceF_D = {}

if type(priceT) == tuple:
    for j in range(0, len(priceT)):
        priceL.append(priceT[j].strip())
        print("PRICE-L-mid", priceL)
```

```
        priceL[len(priceL) - 1] = priceL[len(priceL) - 1].replace(' ', ' ').strip()
    elif type(priceT) == str:
        priceT = priceT.replace(' ', ' ').strip()
        priceL.append(priceT)

    print('Price-L', priceL, type(priceL), len(priceL))

    # STORAGE: PriceL_F
    for j in range(0, len(priceL)):
        print('j', j)
        priceL_F.append(re.split('-', priceL[j]))
        print('len', len(priceL_F))
        for j in range(0, len(priceL_F[len(priceL_F) - 1])):
            priceL_F[len(priceL_F) - 1][j] = priceL_F[len(priceL_F) - 1][j].strip()
            print ('Price-F', priceL_F, '\n')

    # Store Price-List into dict
    for n in range(0, len(priceL_F)):
        print (n)
        newItem = ''
        if len(priceL_F[n]) < 2:
            priceF_D[priceL_F[n][0]] = 'PRICE N/A'
        else:
            print ('AVAL', priceL_F[n])
            for k in range(1, len(priceL_F[n])):
                toDict = priceL_F[n][k].replace(',','')
                if k == len(priceL_F[n]) - 1:
                    newItem += toDict
                else:
                    newItem += toDict + '-'

            priceF_D[priceL_F[n][0]] = newItem

    priceD_L.append(priceF_D)
    print("PRICE_Dict",len(priceF_D),priceF_D)
    print("PRICE_LiSTTTTTT", priceD_L)
```

```
    filename = 'all_apts-2700.csv'
    print('PAGE\n', pageNum)

    with open(filename, 'a', encoding='utf-8') as fa:
        if (pageNum == 1):
            for i in range(0, len(typeL) + 1):
                if (i == 0):
                    strW = ''
                    for m in range(0, 22):
                        if m < 21:
                            strW += '%s,'
                        else:
                            strW += '%s\n'
                    fa.writelines(strW %
('Address','Name','City','State','Country','Type_of_rental','URL','Number_of_floorplans','Prc1','Prc2','Prc3
','Prc4','Prc5','Prc6','Prc7','Prc8','Prc9','Prc10','Prc11','Prc12','Prc13','Prc14'))
                else:
                    fa.writelines('{},{},{},{},{},{},{},{},{},{}\n'.format(addrL[i - 1],titleL[i - 1],'Madison','WI','United
States',typeL[i - 1],urlL[i - 1],numPriceL[i - 1],priceD_L[i -1]))
        else:
            for i in range(0, len(typeL)):

                fa.writelines('{},{},{},{},{},{},{},{},{},{}\n'.format(addrL[i],titleL[i], 'Madison', 'WI', 'United States',
typeL[i],urlL[i], numPriceL[i],priceD_L[i]))



def main():
    url = 'https://www.madisoncampusanddowntownapartments.com/search.j?'
    browser.get(url)

    totPage = 24
    for i in range(1, 25):
        search(i, url, totPage)

if name_== '_main_':
    main()
```

7.2. Appendix 2: Python program for getting and saving HTML source code from https://www.rentcollegepads.com/off-campus-housing/madison/search

```
import re

from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

browser = webdriver.Chrome()
wait = WebDriverWait(browser,25)

def search(url):
    lastLen = -1111
    print('entered search')
    homeH_S = browser.page_source
    hasButton = '<div data-records="407" class="loadMoreContainer">'
    ct = 0
```

```
    while True:
        print('new homeH1', lastLen)
        if ct == 0:
            lastLen = 0
        else:
            lastLen = homeH_S.find(hasButton)
        homeH = browser.page_source
        print('new homeH3',lastLen)
        homeH_S = homeH[lastLen:]
        print ('Position of Button:',homeH_S.find(hasButton))
        if (homeH_S.find(hasButton) != -1):
            submit = wait.until(EC.element_to_be_clickable((By.CSS_SELECTOR, '#loadMore')))
            ct += 1
            print('This is the ', ct, ' th click.')
            submit.click()

        else:
            print('ended while')
            break

    allPageHtml = browser.page_source
    pat1 = re.compile('<div data-listing-outer.*?>(.*)</li>.*?</div>' ,re.S)
    chosenH = re.findall(pat1, allPageHtml)[0]
    parseAll(chosenH)

def parseAll(chosenH):
    pat2 = re.compile('\shref="(https://www.rentcollegepads.com/city/madison-wi/listings/\d*)">')
    allItems = re.findall(pat2, chosenH)
    i = 0
    while i < len(allItems) - 1:
        if allItems[i] == allItems[i + 1]:
            allItems.pop(i)
        else:
            i += 1

def main():
    url = 'https://www.rentcollegepads.com/off-campus-housing/madison/search'
```

```
    browser.get(url)
    search(url)

if name_== '_main_':
    main()
```

7.3. Appendix 3: Python program for parsing HTML source code of the rental listing page acquired in the last step, get URLs to each individual rental webpage, and getting and saving source code for these individual rental webpages

```
import requests
from requests.exceptions import RequestException

def openUrl():
    with open('urls.txt','r') as fr:
        container = fr.readlines()[0]
        conte = fr.read
    print(container)
    container = container.replace('\'','')
    listUrl = container.split(',')
    print (listUrl, len(listUrl))

    return listUrl

def get_each_page(urlThisP):
    try:
        response = requests.get(urlThisP)
        if response.status_code == 200:
            return response.text
        else:
            return None
    except RequestException:
        return None

def writeToFile(htmlL):
    for i in range(0, len(htmlL)):
        fileName = 'Apt' + str(i) + '.txt'
        print('Currently writing: ' + fileName)
```

```
        with open(fileName,'w', encoding='utf-8') as fw:
            fw.writelines(htmlL[i])


def main():
    htmlList = []
    listUrl = openUrl()
    for i in range(0, len(listUrl)):
        print("Getting html of page ", i)
        inter = get_each_page(listUrl[i])
        if inter != None:
            htmlList.append(inter)
    writeToFile(htmlList)

if name_== '_main_':
    main()
```

7.4. Appendix 4: Python program for parsing HTML source code from https://www.rentcollegepads.com/off-campus-housing/madison/search obtained from the last step and parsing it into csv

```
import re

def readFiles(page):
    contnt = ''
```

```
    fileName = 'Apt' + str(page) + '.txt'
    with open(fileName, 'r') as fr:
        while True:
            line = fr.readline()
            if len(line) == 0:
                break
            else:
                line=line.strip('\n')
                contnt += line
    return contnt

def parse_this_page(html):
    strForPat3 = ''
    pat1 = re.compile('<h1>(.*?)<span.*</div>(.*?)</span></h1>',re.S)
    aptName = re.findall(pat1, html)[0][0]
    aptAddr = re.findall(pat1, html)[0][1]

    pat2 = re.compile('<ul class="tabs-menu">(.*?)</ul>')

    try:
        allFlpHtml = re.findall(pat2, html)[0]
    except:
        allFlpHtml = ''

    numFlp = allFlpHtml.count('</li>')
    for i in range(0, numFlp):
        strForPat3 += '<li.*?><a href=.*?>(.*?)</a></li>'
    pat3 = re.compile(strForPat3)
    allFlp = re.findall(pat3, allFlpHtml)

    return aptName, aptAddr, allFlp

def get_urls():
    with open ('urls.txt','r') as fr:
        allUrlStr = (fr.readlines())[0]
    allUrlL = allUrlStr.split(',')
```

```
    newAllUrlL = []
    for url in allUrlL:
        newUrl = url.strip('\"')
        newAllUrlL.append(newUrl)
        print (newUrl)
    return newAllUrlL

def main():

    allUrls = get_urls()

    for i in range(0, 405):
        #i == 61 or i == 226 or i == 264 or i == 303 or i == 330 or

        if i == 61 or i == 226 or i == 264 or i == 303 or i == 331 or i == 339 or i == 364 or i == 374:
          continue
        writeF = 'CP_all_apts_51.csv'

        thisUrl = allUrls[i]

        htmlThisP = readFiles(i)
        aptName, aptAddr, allFlp = parse_this_page(htmlThisP)
        print("\npassed Flp", allFlp)

        # strip of city, state, country name
        ind_sep = aptAddr.find('Madison')
        if ind_sep != -1:
            aptAddr_sep = aptAddr[:ind_sep]
            aptAddr_sep = aptAddr_sep.strip(', ')
        else:
            aptAddr_sep = aptAddr

        # get zip code
        zip_pat = re.compile('53\d\d\d')
        try:
            zip_find = re.findall(zip_pat, aptAddr)[0]
            addrLv2 = 'Madison,WI ' + zip_find + ',USA'
```

```
except:
    addrLv2 = 'Madison,WI,USA'
# append city, state, country and zip code separately
print('addrLv2', addrLv2)
# Get Flp details
## get level 1
priceL_pat = re.compile('<tbody>(.*?)</tbody>', re.S)
priceL_this_all = re.findall(priceL_pat, htmlThisP)[1]
print('PRICE_List', priceL_this_all)
## get level 2
priceL_pat2 = re.compile('<tr.*?>(.*?)</tr>')
priceL_this_all2 = re.findall(priceL_pat2, priceL_this_all)
print('Price_List', priceL_this_all2)

## get level 3
thisFlpLL = []
for item in priceL_this_all2:
    thisFlpL = []
    priceL_pat3 = re.compile('<td>(.*?)</td>')
    # priceL_this_det contains all info of a single floorplan
    priceL_this_det = re.findall(priceL_pat3, item)

    bed_this_det = priceL_this_det[1]
    bath_this_det = priceL_this_det[2]
    try:
        price_this_det = int(priceL_this_det[3].strip('$'))
    except:
        price_this_det = -1

    area_this_det = priceL_this_det[4].split('-')

    thisFlpL.append(bed_this_det)
    thisFlpL.append(bath_this_det)
    thisFlpL.append(price_this_det)
    thisFlpL.append(area_this_det) # list of area range under each floorplan

    print("detail", bed_this_det, price_this_det, area_this_det)
```

```
    thisFlpLL.append(thisFlpL)

# list of same floorplan price list
sfpL = []
sfpLL = []
sfpAL = []
sfpALL = []

# Append the first flp item
sfpL.append(thisFlpLL[0][0])
sfpL.append(thisFlpLL[0][2])

if len(thisFlpLL[0][3]) == 1:
    sfpAL.append(int(thisFlpLL[0][3][0]))
elif len(thisFlpLL[0][3]) == 2:
    sfpAL.append(int(thisFlpLL[0][3][0]))
    sfpAL.append(int(thisFlpLL[0][3][1]))

if len(thisFlpLL) == 1:
    sfpLL.append(sfpL)
    sfpALL.append(sfpAL)

for j in range(1, len(thisFlpLL)):
    if thisFlpLL[j][0] != thisFlpLL[j - 1][0]:
        sfpLL.append(sfpL)
        sfpL = []
        sfpL.append(thisFlpLL[j][0])
        sfpL.append(thisFlpLL[j][2])

        sfpALL.append(sfpAL)
        sfpAL = []
        sfpAL.append(thisFlpLL[j][3][0])
        if len(thisFlpLL[j][3]) == 2:
            sfpAL.append(thisFlpLL[j][3][1])
    else:
        # else, the last item is the same as this
```

```
            # append the this item to the same L
            sfpL.append(thisFlpLL[j][2])
            sfpAL.append(thisFlpLL[j][3][0])
            if len(thisFlpLL[j][3]) == 2:
                sfpAL.append(thisFlpLL[j][3][1])

        if j == len(thisFlpLL) - 1:
            sfpLL.append(sfpL)

            sfpALL.append(sfpAL)
        print('Prices under this flp\n', sfpL)

print('price list of all flp of this apartment', sfpLL)

# GET MAX/MIN PRICES of each flp
thisFlpLL = []
# get price range for each floorplan (only consider number of bedrooms)
for thisFlpPrc in sfpLL:
    print('checking prices under this flp',thisFlpPrc)
    maxPrc = -10
    minPrc = 10000
    for k in range(1, len(thisFlpPrc)):
        if thisFlpPrc[k] < minPrc:
            minPrc = thisFlpPrc[k]
        if thisFlpPrc[k] > maxPrc:
            maxPrc = thisFlpPrc[k]
    title = thisFlpPrc[0]

    thisFlpL = []

    thisFlpL.append(title)
    if minPrc == maxPrc:
        thisFlpL.append(minPrc)
    else:
        thisFlpL.append(minPrc)
        thisFlpL.append(maxPrc)
```

```
    thisFlpLL.append(thisFlpL)

# GET MAX/MIN AREAS of each flp
thisFlpAS = ''
thisFlpASL = []
print('CHECKING area', )
for thisFlpA in sfpALL:
    print('checking AREA under this flp',thisFlpA)
    maxA = -1
    minA = 10000
    for k in range(0, len(thisFlpA)):
        if thisFlpA[k] != '':
            if int(thisFlpA[k]) < minA:
                minA = int(thisFlpA[k])
            if int(thisFlpA[k]) > maxA:
                maxA = int(thisFlpA[k])

    thisFlpAS = ''

    if maxA == -1:
        thisFlpAS += ''
    else:
        if minA == maxA:
            thisFlpAS += str(minA)
        else:
            thisFlpAS += str(minA) + '-' + str(maxA)

    thisFlpASL.append(thisFlpAS)

    print("MIN-MAX area",thisFlpAS)
print("AREA LIST",thisFlpASL,len(thisFlpASL))

# NEXT
aptName.replace('Street','St')
aptAddr_sep.replace('Street','St').replace('Avenue','Ave')

aptType = 'Apartment'
```

```
        if (aptName == aptAddr_sep) or (aptName == ''):
            aptName = 'NO-NAME'
            aptType = 'House'

        with open(writeF, 'a') as fa:
            print("TITLE", i)
            if i == 0:
                fa.writelines('{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{}\n'.format('Name', 'Address', 'City', 'State',
'Country', 'Type_of_rental', 'URL',
'Number_of_floorplans','Prc1','Prc2','Prc3','Prc4','Prc5','Prc6','Prc7','Prc8'))
            flpD = {}
            for k in range(0,len(thisFlpLL)):
                if thisFlpLL[k][0] != 'Studio':
                    key = str(thisFlpLL[k][0]) + ' Bedroom(s)'
                else:
                    key = str(thisFlpLL[k][0])

                if len(thisFlpLL[k]) == 2:

                    if thisFlpASL[k] != '':
                        flpD[key] = '$' + str(thisFlpLL[k][1]) + '#' + thisFlpASL[k] + 'sf'
                    else:
                        flpD[key] = '$' + str(thisFlpLL[k][1])
                elif len(thisFlpLL[k]) == 3:
                    if thisFlpLL[k][1] == -1:
                        thisFlpLL[k][1] = 'Ask'
                    if thisFlpLL[k][2] == -1:
                        thisFlpLL[k][2] = 'Ask'

                    print(thisFlpASL[k],thisFlpASL[k])

                    if thisFlpASL[k] != '':
                        flpD[key] = '$' + str(thisFlpLL[k][1]) + '-' + str(thisFlpLL[k][2]) + '#' + thisFlpASL[k] + 'sf'
                    else:
                        flpD[key] = '$' + str(thisFlpLL[k][1]) + '-' + str(thisFlpLL[k][2])

            fa.writelines('{},{},{},{},{},{},{},{}\n'.format(aptName, aptAddr_sep, addrLv2, aptType, thisUrl, len(thisFlpLL),
```

```
flpD))
        print("This FLP", thisFlpLL)
        #print('INFO:')
        #print(aptName, aptAddr)
        #aptNameL.append(aptName)
        #aptAddrL.append(aptAddr)
        #allFlpL.append(thisFlpLL)

if name_== '_main_':
    main()
```

7.5. Appendix 5: Python program for combining csv obtained from two rental resources

```
import re

data1 = []
data2 = []
addrL1 = []
addrL2 = []

with open('MCADA_FIXED.csv','r') as fr:
    while True:
        line = fr.readline()
        if len(line) != 0:
            list = line.split(',')
            data1.append(list)
        else:
            break

with open('CP_FIXED.csv','r') as fr:
    while True:
        line = fr.readline()
        if len(line) != 0:
            list = line.split(',')
            data2.append(list)
        else:
            break

# Get address list of MCADA
```

```
for det in data1:
    addrL1.append(det[1])
# Get address list of CP

for det in data2:
    addrL2.append(det[1])

for i in range(0, len(data2)):
    for j in range(8, len(data2[i])):
        data2[i][j] = data2[i][j].replace('\'', '').replace('{', '').replace('}', '').replace('\n', '')

# add up similar ones
counter = 0
listComb = []
remove1 = []
remove2 = []
i = 0
j = 0
while i < len(addrL1):
    j = 0
    while j < len(addrL2):
        if addrL1[i] == addrL2[j]:
            counter += 1
            str1 = ''
            for k in range(0, len(data1[i])):
                if k < len(data1[i]) - 1:
                    str1 += data1[i][k] + ','
                else:
                    str1 += data1[i][k]

            str2 = ''
            for k in range(0, len(data2[j])):
                if k >= 8 and k < len(data2[j]) - 1:
                    str2 += data2[j][k] + ','
                else:
                    str2 += data2[j][k]
            itemComb = []
```

```
# Combine records from two sources into one.
titleNew = data1[i][0] + '&' + data2[j][0]
itemComb.append(titleNew)
itemComb.append(data1[i][1])

if data2[j][2] == 'NO-NAME' and data1[i][2] != 'NO-NAME':
    itemComb.append(data1[i][2])
elif data2[j][2] != 'NO-NAME' and data1[i][2] == 'NO-NAME':
    itemComb.append(data2[j][2])
elif data2[j][2] == 'NO-NAME' and data1[i][2] == 'NO-NAME':
    itemComb.append('NO-NAME')
elif data2[j][2] != 'NO-NAME' and data1[i][2] != 'NO-NAME' and data2[j][2] == data1[i][2]:
    itemComb.append(data2[j][2])
elif data2[j][2] != 'NO-NAME' and data1[i][2] != 'NO-NAME' and data2[j][2] != data1[i][2]:
    itemComb.append(data2[j][2] + ' -AND- ' + data1[i][2])

for k in range(3, 7):
    itemComb.append(data2[j][k])

linkStr = ''
data2[j][7] = data2[j][7].replace('\"','')
linkStr += (data2[j][7]) + ','
linkStr += (data1[i][7])# + '\n'
itemComb.append(linkStr)

for k in range(8, len(data1[i])):
    data1[i][k] = data1[i][k].replace('\n','')
    itemComb.append(data1[i][k])# + '\n')
#itemComb.append('\n←MC|CP→\n')
for k in range(8, len(data2[j])):
    itemComb.append(data2[j][k])# + '\n')

# append string to itemStr
itemStr = ','.join(itemComb)
listComb.append(itemStr)

remove2.append(data2[j])
```

```
            remove1.append(data1[i])

        j += 1
    i += 1

lenComb = len(listComb)

for item1 in remove1:
    if item1 in data1:
        data1.remove(item1)

for item2 in remove2:
    if item2 in data2:
        data2.remove(item2)

for i in range(0, len(data1)):
    str = ''
    for j in range(0, len(data1[i])):
        if j < len(data1[i]) - 1:
            str += data1[i][j] + ','
        else:
            str += data1[i][j]
        str = str.replace('\n','')
    listComb.append(str)

# Here add remaining unrepeated items in CP and MC to listComb
for i in range(0, len(data2)):
    str = ''
    for j in range(0, len(data2[i])):
        if j < len(data2[i]) - 1:
            str += data2[i][j] + ','
        else:
            str += data2[i][j]
    listComb.append(str)

counter2 = 0
for item in listComb:
```

```
    print(counter2, item)
    counter2 += 1

print(len(data1) + len(data2) + lenComb)
print(counter, len(listComb))

with open('MC-CP-combine.csv', 'w') as fw:
    for item in listComb:
        fw.writelines(item + '\n')
```

7.6. Appendix 6: Python program for parsing csv table from Postgis and two columns from the tax-parcels-assessor-property-information table and into apartments GeoJSON

```
import json, copy
import re
import utm

city_pop_dict = {}
elemLL = []
allLat = []
allLon = []
PK = 0
LL = []
L2= []
L3 = []
with open('1.txt', 'r') as fr:
    while True:
        L= []
        coordL2 = []
        L2 = []
        line = fr.readline()
        if len(line) == 0:
            break
        else:
```

```
            line = line.replace('\t',' ')
            pat = re.compile('\"\s*\"')
            line = re.sub(pat, ',', line)

            yesCoord = line.split(')))),')[0]
            noCoord = line.split(')))')[1]

            # noCoord
            noL = noCoord.split(',')
            LL.append(noL)

            L = yesCoord.replace('\"','').replace('MULTIPOLYGON','').replace('(','').replace(')','')
            coordL= L.split(',')
            #print('split',coordL)
            for xy in coordL:
                coord_new = []
                #print('raw',xy)
                coords = xy.split(' ')
                coords_new = utm.to_latlon(float(coords[0]), float(coords[1]), 16, 'U')
                lat = coords_new[0]
                lon = coords_new[1]
                coord_new.append(lon)
                coord_new.append(lat)

                allLon.append(lon)
                allLat.append(lat)

                # single coord
                coordL2.append(coord_new)

            # one polygon
            L2.append(coordL2)

        # polygons
        L3.append(L2)

for i in range(0, len(LL)):
```

```
    LL[i][0] = i
    print(LL[i],'AND', L3[i])
url1 = []
url2 = []
for i in range(0, len(LL)):
    url = LL[i][5].split('|AND|')
    if len(url) == 2:
        url1.append(url[0])
        url2.append(url[1])
    else:
        url1.append(url[0])
        url2.append('-')

    for j in range(6, len(LL[i])):
        if j == len(LL[i]) - 1:
            LL[i][j] = LL[i][j].replace('\"', '').replace('\n','').strip(' ')

########################################

elemLL = LL
for item in elemLL:
    item[1] = item[1].replace('West','W').replace('East','E').replace('South','S').replace('North','N')
    item[1] =
item[1].replace('Avenue','Ave').replace('Street','St').replace('Place','Pl').replace('Road','Rd').replace('Court','C
t').replace('Drive','Dr')

aptD_dict = {} # city_pop_dict is the top-level dictionary, which contains dictionaries indicating
# the type of 'FeatureCollection' and 'features'

eachAptD = {} # eachCity, a dict that stores a dict indicating the type of 'Feature', a dict of geometry,
# and a dict of properties

geom = {} # a dict that stores a dict indicating the type of 'Point', and a list of coordinates

coord = [] # a list of two coordinates

features = [] # a list of features to be added under the key of 'features' in city_pop_dict
```

```
aptD_dict['type'] = "FeatureCollection" # add the item of 'type'

yrL = []
layoutL = []
addrAptL = []


for i in range(1, len(elemLL)):
    #print('\nNEW--- ',i,len(elemLL))
#    eachCity = {}
    geom = {}
#    years = {}
#    yrKeyList = []
    eachAptD = {}
    eachAptD['type'] = 'Feature' # set the item whose key is 'type' in the eachCity dict

    geom['type'] = 'Polygon' # set the item whose key is 'type' in the geom dict

    coord = [] # restart an empty coordinate list

    # append geom
    try:
        coord.append(L3[i][0])
    except:
        print('mistake',i)
    # append the item whose key is 'coordinates' to the geom dict
    geom['coordinates'] = coord

    # append the item whose key is 'geometry' to the eachCity dict
    eachAptD['geometry'] = geom

    flpD = {}
    flpDL_sub = []
    flpDL = []

    # assign the years dict, city id, name, label under the item whose key is 'properties' under the eachCity dict
```

```
eachAptD['properties'] = {}
eachAptD['properties']['Address'] = elemLL[i][2]

yrL = []
layoutL = []
addrAptL = []

with open('bldg_det.txt', 'r') as fr:
    while True:
        line = fr.readline()
        if len(line) == 0:
            break
        thisL = line.split(',')
        # yrL.append()
        yrL.append(thisL[61])
        layoutL.append(thisL[54])
        addrAptL.append(thisL[6])

k = 0
while k < len(yrL) - 1:
    if addrAptL[k] == addrAptL[k + 1]:
        del addrAptL[k]
        del layoutL[k]
        del yrL[k]
    else:
        k += 1

addrAptL = addrAptL[1:]
layoutL = layoutL[1:]
yrL = yrL[1:]

simpleTy = ''
for j in range(len(yrL)):
    if elemLL[i][2] == addrAptL[j]:
        if yrL[j] == '0':
            eachAptD['properties']['Year_built'] = 'N/A'
        else:
```

```
            eachAptD['properties']['Year_built'] = yrL[j]
        eachAptD['properties']['Detailed_type'] = layoutL[j]

        simpleTy = layoutL[j].replace(' unit Apartment', '').replace('Single family','1').replace(' Unit','')
        try:
            int(simpleTy)
        except:
            simpleTy = 'other'
        eachAptD['properties']['Simplified_type'] = simpleTy

eachAptD['properties']['FID'] = i
eachAptD['properties']['Name'] = elemLL[i][1]
eachAptD['properties']['Type_of_housing'] = elemLL[i][4]
eachAptD['properties']['URL1'] = url1[i]
eachAptD['properties']['URL2'] = url2[i]

eachAptD['properties']['Floorplans'] = {}

flpKeyNum = 0
flpRmMainPrv = ''
exKeyNum = 0
for j in range(6, len(elemLL[i])):

    # Flp list
    flpRm = ''
    flpRmMain = ''
    flpRmMainL = []

    flpRmSub = ''
    flpRmSubL = []

    flpRmL = []

    flpDet = ''
    flpPrc = [] # lowest and highest price
    flpPrcL = [] # list of all price ranges
```

```
flpArea = [] # smallest and largest area
flpAreaL = [] # list of all area ranges

flpUnitPrc = [] # lowest and highest price/unit area
flpUnitPrcL = [] # list of all price/area ranges

flpInfo = elemLL[i][j].split(':')

# Name of the flp
flpRm = flpInfo[0]
print("NAMEEEEEEE",i, j)
flpSub = ''
flpSub_pat = re.compile('\(.*?\)')

# Evaluate for every row, whether it's an extra feature or not (by default NOT)
exFtr = False

try:
    # When there is sub (Den/Penthouse...)
    # When an extra bracket other than (s) is found, implying an extra feature exists
    if ('Studio' in flpRm) == False:
        flpSub_p = re.findall(flpSub_pat, flpRm)[1] # has to be 1 (0 is (s))
    else:
        flpSub_p = re.findall(flpSub_pat, flpRm)[0]

    try:
        flpSub_p2 = re.findall(flpSub_pat, flpRm)[2]
    except:
        flpSub_p2 = ''
    print(flpSub_p)
    flpSub = flpSub_p.replace('(','').replace(')','')

    flpRmMain = flpRm.replace(flpSub_p,'').replace(flpSub_p2,'')
    print('truncating',flpSub_p,flpRmMain)

    # when this flp title is different from the last one, key# +1
    print('before compare1', flpRmMain, flpRmMainPrv)
```

```
    if flpRmMain != flpRmMainPrv:
        flpKeyNum += 1
        exKeyNum = 1
        flpKey = 'Floorplan' + str(flpKeyNum)

        eachAptD['properties']['Floorplans'][flpKey] = {}
        eachAptD['properties']['Floorplans'][flpKey]['Title'] = flpRmMain

    else:
        exKeyNum  += 1

    # to be the next key
    flpRmMainPrv = flpRmMain

    exftr = True

except:
    exKeyNum = 0
    # when there is no sub (NO Den/Penthouse...)
    flpRmMain = flpRm

    #print('before compare2', flpRmMain, flpRmMainPrv)

    if flpRmMain != flpRmMainPrv:

        flpKeyNum += 1
        flpKey = 'Floorplan' + str(flpKeyNum)

        eachAptD['properties']['Floorplans'][flpKey] = {}

    # to be the next key

    flpRmMainPrv = flpRmMain
    #flpRmSubL.append('')
    ##
    exftr = False
```

```
        #print('\nMAIN: ',len(flpRmMainL),': ', flpRmMain)
        #print('SUB: ----------',len(flpRmSubL),'----------  ',flpSub)

        #print('flp-KEY', flpKeyNum)
        #print('extra-KEY', exKeyNum)

        # Price and area of the flp
        try:
            flpDet = flpInfo[1]
        except:
            flpDet = ''

        # area
        area_pat = re.compile('#(.*?)sf')
        try:
            area_all = re.findall(area_pat, flpDet)[0]
        except:
            area_all = ''

        flpArea = area_all.split('-')
        print("FLP",flpArea)

        newFlp = []
        for item in flpArea:
            if item != '':
                newFlp.append(int(item))
            else:
                newFlp.append(item)

        flpAreaL.append(newFlp)

        # price
        prc_all = flpDet.replace(area_all,'').replace('sf','').replace('#','').replace('$Ask-','').replace('
','').replace('\n','').replace('$Ask','N/A').replace('PRICE N/A','N/A').replace('PRICEN/A','N/A')
        try:
            flpPrcL = [int(f) for f in (prc_all.replace('$','')).split('-')]
        except:
```

```
    flpPrcL = (prc_all.replace('$','')).split('-')
#flpPrcL = flpPrc # price range (a list of 2 values)

# unit price
if flpArea[0] == 'N/A':
    flpUnitPrc = 'N/A'
    flpUnitPrcL.append(flpUnitPrc)
else:
    #print('testttttt', flpAreaL[0], flpPrcL)

    if len(flpPrcL) == 1 and len(flpAreaL[0]) == 1 and flpAreaL[0][0] != '':
        try:
            flpUnitPrc.append(int(flpPrcL[0])/int(flpAreaL[0][0]))
        except:
            pass
            #print("ERROR 1-2",i,flpPrcL[0],flpAreaL[0])
    elif len(flpPrcL) == 1 and len(flpAreaL[0]) == 2 and flpAreaL[0][0] != '':
        try:
            fup1 = (int(flpPrcL[0])/int(flpAreaL[0][0]))
            fup2 = (int(flpPrcL[0])/int(flpAreaL[0][1]))
            if fup1 > fup2:
                flpUnitPrc.append(fup2)
                flpUnitPrc.append(fup1)
            else:
                flpUnitPrc.append(fup1)
                flpUnitPrc.append(fup2)
        except:
            pass
            #print("ERROR 1-22",i)
    elif len(flpPrcL) == 2 and len(flpAreaL[0]) == 1 and flpAreaL[0][0] != '':
        try:
            fup1 = int(flpPrcL[0])/int(flpAreaL[0][0])
            fup2 = int(flpPrcL[1])/int(flpAreaL[0][0])
            if fup1 > fup2:
                flpUnitPrc.append(fup2)
                flpUnitPrc.append(fup1)
            else:
```

```
                flpUnitPrc.append(fup1)
                flpUnitPrc.append(fup2)
            except:
                pass
                #print("ERROR 2-1",i)
        elif len(flpPrcL) == 2 and len(flpAreaL[0]) == 2 and flpAreaL[0][0] != '':
            try:
                fup1 = (int(flpPrcL[0])/int(flpAreaL[0][0]))
                fup2 = (int(flpPrcL[1])/int(flpAreaL[0][1]))
                if fup1 > fup2:
                    flpUnitPrc.append(fup2)
                    flpUnitPrc.append(fup1)
                else:
                    flpUnitPrc.append(fup1)
                    flpUnitPrc.append(fup2)
            except:
                pass
                #print("ERROR 2-2",i)
        else:
            flpUnitPrc.append('N/A')

        #if flpUnitPrc[0] != 'PRICE/UNIT_AREA N/A':
         # flpUnitPrc = 0
        flpUnitPrcL.append(flpUnitPrc) # price/unit area range (a list of 2 values)

    # create a list of dicts for all CURRENT flp (Studio/ 1 Bedroom(s)/ 2 Bedroom(s), etc.)
    for k in range(0, len(flpRmL)):
        flpD = {}
        flpD['Room'] = flpRmL[k]
        print('p')
        flpD['Price'] = flpPrcL[k]
        flpD['Area'] = flpAreaL[k]
        flpD['Unit_Price'] = flpUnitPrcL[k].sort(reverse=True)
        flpDL.append(flpD) #with a length of k (count of all flps)

    for m in range(0, len(flpRmMainL)):
        main = flpRmMainL[m]
```

```
            ftr_num = 0
            for s in range(0, len(flpRmSubL)):
                sub = flpRmSubL[s]
                if (sub == main) and (sub != ''):
                    ftr_num += 1
                    flpDL[m][sub] = {}
                    flpDL.remove(flpDL[s])
                    flpDL[m][sub] = {}

        # every flp is a dict

        if flpUnitPrcL[0] == []:
            flpUnitPrcL[0] = ['N/A']

        if flpAreaL[0][0] == '':
            flpAreaL[0] = ['N/A']

        # when the row is a new flp, NOT an extra feature
        if exftr == False:

            print('LLLLLL',flpPrcL, len(flpAreaL), flpUnitPrcL)
            if len(flpAreaL) != 0:
                print('NOOOO',len(flpRmMain),type(flpRmMain))

            eachAptD['properties']['Floorplans'][flpKey]['Title'] = flpRmMain
            eachAptD['properties']['Floorplans'][flpKey]['Price'] = flpPrcL
            eachAptD['properties']['Floorplans'][flpKey]['Area'] = flpAreaL[0]

            try:
                eachAptD['properties']['Floorplans'][flpKey]['Price/Unit_area'] = [round(flpUnitPrcL[0][0],
2),round(flpUnitPrcL[0][1], 2)]
            except:
                try:
                    eachAptD['properties']['Floorplans'][flpKey]['Price/Unit_area'] = [round(flpUnitPrcL[0][0], 2)]
                except:
                    eachAptD['properties']['Floorplans'][flpKey]['Price/Unit_area'] = ['N/A']
```

```
        # determine number of residence
        if flpRmMain == 'Studio' or flpRmMain == 'Individual' or flpRmMain == 'Individual Leases':
            numPeople = 1
            extracted = 1
        elif flpRmMain == 'Shared Leases':
            numPeople = 2
            extracted = 2
        else:
            indNum = flpRmMain.index(' ')
            print("ORIG",flpRmMain)
            extracted = flpRmMain[:indNum].replace('+','')
            numPeople = int(extracted)
        print('ROOM',extracted,'FROM',flpRmMain)

        eachAptD['properties']['Floorplans'][flpKey]['Individual_Average'] = {}
        # average price
        try:
            eachAptD['properties']['Floorplans'][flpKey]['Individual_Average']['avg_price'] = round((flpPrcL[0]
+ flpPrcL[1]) / 2 / numPeople, 1)
        except:
            try:
                eachAptD['properties']['Floorplans'][flpKey]['Individual_Average']['avg_price'] =
round(flpPrcL[0] / numPeople, 1)
            except:
                eachAptD['properties']['Floorplans'][flpKey]['Individual_Average']['avg_price'] = 'N/A'

        # average area
        try:
            eachAptD['properties']['Floorplans'][flpKey]['Individual_Average']['avg_area'] =
round((flpAreaL[0][0] + flpPrcL[0][1]) / 2 / numPeople, 1)
        except:
            try:
                eachAptD['properties']['Floorplans'][flpKey]['Individual_Average']['avg_area'] =
round(flpPrcL[0][0] / numPeople, 1)
            except:
                eachAptD['properties']['Floorplans'][flpKey]['Individual_Average']['avg_area'] = 'N/A'
        # average flp
```

```
            try:
                eachAptD['properties']['Floorplans'][flpKey]['Individual_Average']['avg_price_unit_area'] =
round((round(flpUnitPrcL[0][0], 2) + round(flpUnitPrcL[0][1], 2)) / 2, 1)
            except:
                try:
                    eachAptD['properties']['Floorplans'][flpKey]['Individual_Average']['avg_price_unit_area'] =
round(flpUnitPrcL[0][0], 1)
                except:
                    eachAptD['properties']['Floorplans'][flpKey]['Individual_Average']['avg_price_unit_area'] =
'N/A'

        else:
            exKey = 'Extra_Feature_' + str(exKeyNum)

            print('KEYS', flpKey, exKey)
            if i == 90:
                print('before',eachAptD['properties']['Floorplans'][flpKey])
            eachAptD['properties']['Floorplans'][flpKey][exKey] = {}
            if i == 90:
                print('mid',eachAptD['properties']['Floorplans'][flpKey])
            if i == 90:
                print (eachAptD['properties']['Floorplans'][flpKey], flpKey)

            eachAptD['properties']['Floorplans'][flpKey][exKey]['Sub_title'] = flpSub
            eachAptD['properties']['Floorplans'][flpKey][exKey]['Price'] = flpPrcL
            eachAptD['properties']['Floorplans'][flpKey][exKey]['Area'] = flpAreaL

            try:
                eachAptD['properties']['Floorplans'][flpKey][exKey]['Price/Unit_area'] = [round(flpUnitPrcL[0][0],
2),round(flpUnitPrcL[0][1], 2)]
            except:
                try:
                    eachAptD['properties']['Floorplans'][flpKey][exKey]['Price/Unit_area'] = [round(flpUnitPrcL[0][0],
2)]
                except:
                    eachAptD['properties']['Floorplans'][flpKey][exKey]['Price/Unit_area'] = ['N/A']
```

```
    print(eachAptD['properties']['Floorplans'])

    # deepcopy the dictionary of eachCity to create a new compound dict, while append as a method of direct
    # assignment will only refer to the original eachCity, which is a compound dict
    dictDpCpy = copy.deepcopy(eachAptD)
    features.append(dictDpCpy)

    #print(len(features))
ct = 0
for item in features:
    ct += 1
    #print(ct)
    #print (item),
#print(len(features))

# assign feature to the key of 'features' under city_pop_dict
city_pop_dict['features'] = features
city_pop_geojson = json.dumps(city_pop_dict)

#print('typeeeeee', city_pop_geojson)
city_pop_geojson = city_pop_geojson

#write to a new geojson file
with open('all_apts_222333.geojson','w') as fw:
    fw.writelines(city_pop_geojson)
```

7.7. Appendix 7: Python program for parsing bus routes and stops into two GeoJSON files

7.7.1.

```
import re
LL = []
routeL = []

with open('campus_routes.txt') as fr:
    while True:
        line = fr.readline()
        if len(line) == 0:
```

```
            break
        else:
            print(line)
            pat = re.compile('\"(.*?)\"')
            all = re.findall(pat,line)[0]
            thisL = all.split(',')
            thisL = [int(f.replace(' ','')) for f in thisL]
            LL.append(thisL)

for row in LL:
    print (row)
    for item in row:
        if (item in routeL) == False:
            routeL.append(item)
routeL.sort()
print(routeL,len(routeL))
```

7.7.2.

```
import re
import requests
from requests.exceptions import RequestException

def get_one_page(url):
    try:
        response = requests.get(url)
        if response.status_code == 200:
            return response.text
        else:
            return None
    except RequestException:
        return None

def parse_one_page(htmlpage):
    pattern = re.compile('<th scope="col".*?>(.*?)</th>')
    items = re.findall(pattern,htmlpage)
    items = [f.replace('amp; ',' ').replace(' ',' ') for f in items]
    i = 0
```

```
    while i < len(items):
        if items[i][0] == '<':
            print(items[i])
            print(len(items))
            del items[i]
        else:
            i += 1
    return items

def main():
    autoCt = 0
    items = []
    urlList = []
    fullurl = []
    routeL = [1, 2, 3, 4, 6, 7, 8, 10, 11, 12, 13, 14, 15, 19, 27, 28, 29, 37, 38, 44, 47, 48, 56, 57, 58, 70, 71, 72, 80, 81,
82, 84]
    for i in routeL:
        print(i)
        url = 'https://www.cityofmadison.com/metro/routes-schedules/route-' + str(i)
        htmlpage = get_one_page(url)
        stopList = parse_one_page(htmlpage)
        print(stopList)


if name_== '_main_':
    main()
```

7.7.3.

```
import re
import json

fidL = []
stopCodeL = []
stopNameL = []

stop_latL = []
stop_lonL = []
```

```
routeLL = []

maxLon = -89.26874744446569
minLon = -89.50914111999082
minLat = 43.003953112230484
maxLat = 43.134706405395235

routes = [1, 2, 3, 4, 6, 7, 8, 10, 11, 12, 13, 14, 15, 19, 27, 28, 29, 37, 38, 44, 47, 48, 56, 57, 58, 70, 71, 72, 80, 81, 82,
84]
with open('ALL_MSN_STOPS_to_GeoJSON.txt','r') as fr:
    while True:
        line = fr.readline()
        if len(line) == 0:
            break
        else:
            thisL = line.split(',')
            fid = thisL[0]
            stopCode = thisL[1]
            stopName = thisL[2]

            stop_lat = thisL[3]
            stop_lon = thisL[4]

            routeL = thisL[6:-2]
            print('START:',thisL[6],thisL[-3],routeL)
            try:
                routeL = [int(f.strip().replace('\"','')) for f in routeL]
                routeLL.append(routeL)
                fidL.append(fid)
                stopCodeL.append(stopCode)
                stopNameL.append(stopName)
                stop_latL.append(stop_lat)
                stop_lonL.append(stop_lon)
            except:
                print('MISTAKE',line)
```

```
print(routeLL)
i = 0
print("BEFORE",len(routeLL),len(fidL),len(stopCodeL),len(stopNameL),len(stop_latL),len(stop_lonL))

while i < len(routeLL):
    hasOne = False
    for r in routeLL[i]:
        if (r in routes) == True:
            hasOne = True
        print(r,len(routeLL),routeLL[i])
    if hasOne == False:
        del routeLL[i]
        del fidL[i]
        del stopCodeL[i]
        del stopNameL[i]
        del stop_latL[i]
        del stop_lonL[i]
    else:
        i += 1
        print('evaluating',i)
print("AFTER",len(routeLL),len(fidL),len(stopCodeL),len(stopNameL),len(stop_latL),len(stop_lonL))
for i in range(0, len(routeLL)):
    print (i, routeLL[i],fidL[i],stopCodeL[i],stopNameL[i],round(float(stop_latL[i]),6),round(float(stop_lonL[i]),6))

g = {}
g['type'] = 'FeatureCollection'

features = []
ct = 0
for i in range(0, len(routeLL)):
    eachD = {}
    eachD['type'] = 'Feature'

    geomD = {}
    geomD['type'] = 'Point'

    lon = round(float(stop_lonL[i]),6)
```

```
    lat = round(float(stop_latL[i]),6)
    geomD['coordinates'] = [lon, lat]
    eachD['geometry'] = geomD

    propD = {}
    propD['Stop_code'] = stopCodeL[i]
    propD['Routes'] = routeLL[i]
    propD['Stop_name'] = stopNameL[i]
    eachD['properties'] = propD

    if (lon < minLon or lon > maxLon or lat < minLat or lat > maxLat) == False:
        ct += 1
        features.append(eachD)

g['features'] = features
print('COUNTING',ct)

gjStr = json.dumps(g)
print(type(gjStr),len(gjStr),gjStr)

with open('all_stops_of_campus_routes_BOUNDING_BOX.geojson','w') as fw:
    fw.writelines(gjStr)
```

7.8. Appendix 8: Python program for getting travel time and writing it into the apartments.geojson

```
import re
import requests
import geopandas as gpd
import matplotlib.pyplot as plt
from shapely.geometry import Point, LineString, Polygon, mapping
import json

with open('D:/Senior 2nd semester/Geog_565/Map!!!!!!/leaflet-lab-Lab5B - Copy
(2)/data/all_apts_222666.geojson','r') as fr:
    all = fr.readlines()[0]
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(' ', ''))
# print(len(allPoly),allPoly[0],len(allFID),allFID[9])

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

pat_coords = re.compile('\((.*?)\)')
bascomL = []
natL = []
gordonL = []
unionSL = []
reachDL = []

nearestNodeL = []
for i in range(0, len(allPolyNew)):
    reachD = {}
    pt = allPolyNew[i]
    lnstr = LineString(pt).centroid.wkt
    coords = re.findall(pat_coords, lnstr)[0]
    lon = float(coords.split(' ')[0])
    lat = float(coords.split(' ')[1])
    print(lat, lon)
```

```
    Bascom_url = 'https://maps.googleapis.com/maps/api/directions/json?origin=' + str(lat) + ',' + str(
        lon) + '&mode=walking&destination=43.075343,-
89.404142&key=AIzaSyDh9bK0ydRSsFmxW2NuSlknvlD5ZznEgKk'
    json_return_bas = requests.get(Bascom_url).text
    Bascom_pat = re.compile('"legs".*?distance.*?"text" : "(.*?)".*?duration.*?"text" : "(.*?)"', re.S)
    Bascom_dist = re.findall(Bascom_pat, json_return_bas)[0]
    # bascomL.append(Bascom_dist)
    # print(i, '\n', Bascom_dist)

    nat_url = 'https://maps.googleapis.com/maps/api/directions/json?origin=' + str(lat) + ',' + str(
        lon) + '&mode=walking&destination=43.076953,-
89.420260&key=AIzaSyDh9bK0ydRSsFmxW2NuSlknvlD5ZznEgKk'
    json_return_nat = requests.get(nat_url).text
    ##########nat_pat = re.compile('"legs".*?distance.*?"text" : "(.*?)".*?duration.*?"text" : "(.*?)"', re.S)
    nat_dist = re.findall(Bascom_pat, json_return_nat)[0]
    # natL.append(nat_dist)
    # print(nat_dist)

    gordon_url = 'https://maps.googleapis.com/maps/api/directions/json?origin=' + str(lat) + ',' + str(
        lon) + '&mode=walking&destination=43.071159,-
89.398348&key=AIzaSyDh9bK0ydRSsFmxW2NuSlknvlD5ZznEgKk'
    json_return_gordon = requests.get(gordon_url).text
    ############gordon_pat = re.compile('"legs".*?distance.*?"text" : "(.*?)".*?duration.*?"text" : "(.*?)"', re.S)
    gordon_dist = re.findall(Bascom_pat, json_return_gordon)[0]
    # gordonL.append(gordon_dist)
    # print(gordon_dist)

    unionSL_url = 'https://maps.googleapis.com/maps/api/directions/json?origin=' + str(lat) + ',' + str(
        lon) + '&mode=walking&destination=43.071902,-
89.408028&key=AIzaSyDh9bK0ydRSsFmxW2NuSlknvlD5ZznEgKk'
    json_return_unionS = requests.get(unionSL_url).text
    unionS_dist = re.findall(Bascom_pat, json_return_unionS)[0]

    Bascom_url2 = 'https://maps.googleapis.com/maps/api/directions/json?origin=' + str(lat) + ',' + str(
        lon) + '&mode=bicycling&destination=43.075343,-
89.404142&key=AIzaSyDh9bK0ydRSsFmxW2NuSlknvlD5ZznEgKk'
    json_return_bas2 = requests.get(Bascom_url2).text
```

```
    Bascom_pat2 = re.compile('"legs".*?distance.*?"text" : "(.*?)".*?duration.*?"text" : "(.*?)"', re.S)
    Bascom_dist2 = re.findall(Bascom_pat2, json_return_bas2)[0]

    nat_url2 = 'https://maps.googleapis.com/maps/api/directions/json?origin=' + str(lat) + ',' + str(
        lon) + '&mode=bicycling&destination=43.076953,-
89.420260&key=AIzaSyDh9bK0ydRSsFmxW2NuSlknvlD5ZznEgKk'
    json_return_nat2 = requests.get(nat_url2).text
    nat_dist2 = re.findall(Bascom_pat2, json_return_nat2)[0]

    gordon_url2 = 'https://maps.googleapis.com/maps/api/directions/json?origin=' + str(lat) + ',' + str(
        lon) + '&mode=bicycling&destination=43.071159,-
89.398348&key=AIzaSyDh9bK0ydRSsFmxW2NuSlknvlD5ZznEgKk'
    json_return_gordon2 = requests.get(gordon_url2).text
    gordon_dist2 = re.findall(Bascom_pat2, json_return_gordon2)[0]

    unionSL_url2 = 'https://maps.googleapis.com/maps/api/directions/json?origin=' + str(lat) + ',' + str(
        lon) + '&mode=bicycling&destination=43.071902,-
89.408028&key=AIzaSyDh9bK0ydRSsFmxW2NuSlknvlD5ZznEgKk'
    json_return_unionS2 = requests.get(unionSL_url2).text
    unionS_dist2 = re.findall(Bascom_pat2, json_return_unionS2)[0]

    reachD['to_gordon_walk'] = gordon_dist
    reachD['to_gordon_bicycling'] = gordon_dist2

    reachD['to_bascom_walk'] = Bascom_dist
    reachD['to_bascom_bicycling'] = Bascom_dist2

    reachD['to_unionS_walk'] = unionS_dist
    reachD['to_unionS_bicycling'] = unionS_dist2

    reachD['to_nat_walk'] = nat_dist
    reachD['to_nat_bicycling'] = nat_dist2

    reachDL.append(reachD)
print(len(reachDL))

import json
```

```
with open('D:/Senior 2nd semester/Geog_565/Map!!!!!!/leaflet-lab-Lab5B - Copy
(2)/data/all_apts_222666.geojson',
        'r') as fr:
    geoj = fr.read()
    boundary_df = json.loads(geoj)
print(boundary_df['features'][2]['properties'])
for i in range(0, len(reachDL)):
    boundary_df['features'][i]['properties']['to_campus'] = reachDL[i]

to_write = json.dumps(boundary_df)
with open('D:/apartments_WITH_travel_time_to_campus_2.geojson', 'w') as fw:
    fw.writelines(to_write)
```

7.9. Appendix 9: main.js

```
// Declare global variables
var yrSelectedArr = new Array
var circle2
var markerG = new Array
var data2Yrs = new Array
var checkNotif

var gg2
var gg5
var gg5_b

var gg10
var gg10_b

var gg0
var gg_res

var gg_routes
var gg_stops

var selected_ress = []
var busIcon = L.Icon.extend({
```

```
    options: {
      iconSize: [30,30],
        iconAnchor: [15, 30],
        shadowAnchor: [4, 62],
        popupAnchor:  [-3, -76]
    }
})

var pass_routes = []
var gg_route_sel

var gg_campus_walk
var gg_campus_bike

var  gg_gordon
var  gg_bascom
var  gg_unionS
var gg_nat

var legend1 = L.control({position: 'bottomleft'});
var legend2 = L.control({position: 'bottomleft'});
var legend3 = L.control({position: 'bottomleft'});

//function to instantiate the Leaflet map
function createMap() {
    //create the map using the L.map method
    var map = L.map('map', {
        center: [43.07, -89.403],
        zoom: 15
   });

   L.control.scale({'position':'bottomright'}).addTo(map);

   L.tileLayer('https://{s}.basemaps.cartocdn.com/rastertiles/voyager_labels_under/{z}/{x}/{y}{r}.png', {
      attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors
&copy; <a href="https://carto.com/attributions">CARTO</a>',
      subdomains: 'abcd',
```

```
    maxZoom: 19
  }).addTo(map)

  $('#det_window').hide()
  $('#price_window').hide()
  $('#res_window').hide()
  $('#lib_window').hide()
  $('#bus_window').hide()
  $('#campus_window').hide()
   //call getData function
  getData(map);

};

function onEachFeature(feature, layer) {
   //no property named popupContent; instead, create html string with all properties
   var popupContent = "";
   if (feature.properties) {
      //loop to add feature property names and values to html string
      for (var property in feature.properties) {
        popupContent += "<p>" + property + ": " + feature.properties[property] + "</p>";
      }
      layer.bindPopup(popupContent);
   };
};

function calcR(attrValue) {
  // calculate
  return Math.sqrt((attrValue * 0.5)/Math.PI) * 0.8
}

function pointToLayer(latlng) {

  var options = {
    fillColor: "#ff7800",
    color: "#00aa00",
    weight: 1,
```

```
    opacity: 1,
    fillOpacity: 0.6,
    radius: 2
  }

  // "Circle-Marker" layer is added as a leaflet layer
  var cmLayer = L.circleMarker(latlng,options)

  return cmLayer
}

function set_det_window() {
  $('#menu').hide()
  $('#det_window').show()
  $('.back_to_menu').off('click')
}
function set_campus_window() {
  $('#menu').hide()
  $('#campus_window').show()
  $('.back_to_menu').off('click')
}

function bound_res_10(map,fid1,data10,data_res) {
  //var gg5 = L.geoJson(data5)

  console.log(map.hasLayer(gg10_b))
  if (map.hasLayer(gg10_b)) {
    console.log('HASSSS')
    map.removeLayer(gg10_b)
  }
  if (map.hasLayer(gg5_b)) {
    console.log('HASSSS')
    map.removeLayer(gg5_b)
  }
  gg10_b = L.geoJson(data10, {
    filter: function(feature, layer) {
      if (parseInt(feature.id) === fid1) {
```

```
            var ress10 = feature.properties.reachable_restaurants
            all_ress_10(map, data_res, ress10)
          }
          return (parseInt(feature.id) === fid1)
        }
      }).addTo(map)

      gg10_b.bringToBack()
    }

    function all_ress_5(map, data_res, ress5) {
      console.log('RESSSSSSSSSSSSSSSS calleddddddddddddd')
      if (map.hasLayer(gg5)) {
        map.removeLayer(gg5)
      }
      if (map.hasLayer(gg10)) {
        map.removeLayer(gg10)
      }
      selected_ress = []

      gg5 = L.geoJson(data_res, {
        pointToLayer: function(feature,latlng) {
          var options = {
            radius: 5.5,
            fillColor: "#ffffff",
            color: "#000",
            weight: 1,
            opacity: 1,
            fillOpacity: 0.8
          };
          return L.circleMarker(latlng, options)
        },
        filter: function(feature, layer) {
          if (ress5.includes(parseInt(feature.properties.FID))) {
            console.log('selected', parseInt(feature.properties.FID))
            selected_ress.push(feature)
          }
```

```
      return (ress5.includes(parseInt(feature.properties.FID)))
    },
    onEachFeature(feature, layer) {
      layer.setStyle({color:'#000',weight:'3'})
      if ((feature.properties.rating_score) != '-') {
        var toolTipContent = '<div>' + feature.properties.rating_score + ', ' + feature.properties.price_range +
'</div>'
      } else {
        var toolTipContent = '<div>' + 'N/A' + '</div>'
      }
      layer.bindTooltip(toolTipContent,{'permanent':true,'direction':'top','offset':new L.Point(0,-4)}).openTooltip()

      var popupContent = '<div><p>Name: ' + feature.properties.name + '</p><p>Price range: '
      + feature.properties.price_range + '</p><p>Rating score: ' + feature.properties.rating_score
      + '</p><p>Tags: ' + feature.properties.tags + '</p></div>'

      layer.bindPopup(popupContent, {
        offset: new L.Point(0, -5),
        closeButton: false,
        opacity: 0.8
      })

      layer.on({
        mouseover: function() {
          this.openPopup()
          this.setStyle({color:'blue',fillColor:'orange',weight: 3})
        },
        mouseout: function() {
          this.closePopup()
          this.setStyle({color: "#000",fillColor:'white',weight: 3})
        },
        click: function() {
          $('#ress_info').remove()
          var windowContent = "<div id='ress_info' style='padding-top:-10px; line-
height:20px'>_____<br><b>Name:</b> " + feature.properties.name +
'<br><b>Address:</b> ' + feature.properties.address + '<br><b>Price range:</b> '
          + feature.properties.price_range + '<br><b>Rating count:</b> ' + feature.properties.rating_count +
```

```
'<br><b>Rating score:</b> ' + feature.properties.rating_score
            + '<br>Tags: ' + feature.properties.tags + '<br><b>URL: </b><a href="url">' + feature.properties.url +
'</a></p></div>'
            $('#res_window').append(windowContent)
          }
        })
      }

  }).addTo(map)

  //set_res_window.append()
  $('#ress_notif').remove()
  var panelContent = "<div id='ress_notif'><p>Number of restaurants within 5-min walk distance: <b>" +
ress5.length + '<b></p>'
  $('#res_window').append(panelContent)

  resInfoPopup(selected_ress)
  gg5.bringToFront()
}
function bound_res_5(map,fid1,data5,data_res) {

  console.log(map.hasLayer(gg5_b))
  if (map.hasLayer(gg5_b)) {
    console.log('HASSSS')
    map.removeLayer(gg5_b)
  }
  if (map.hasLayer(gg10_b)) {
    console.log('HASSSS')
    map.removeLayer(gg10_b)
  }
  gg5_b = L.geoJson(data5, {
    filter: function(feature, layer) {
      if (parseInt(feature.id) === fid1) {
        var ress5 = feature.properties.reachable_restaurants
        all_ress_5(map, data_res, ress5)
      }
      return (parseInt(feature.id) === fid1)
```

```
    }
  }).addTo(map)

  gg5_b.bringToBack()
}

function all_ress_10(map, data_res, ress10) {

  console.log('RESSSSSSSSSSSSSSSS calledddddddddddd')
  if (map.hasLayer(gg10)) {
    map.removeLayer(gg10)
  }
  if (map.hasLayer(gg5)) {
    map.removeLayer(gg5)
  }
  selected_ress = []

  var count = 0

  gg10 = L.geoJson(data_res, {
    pointToLayer: function(feature,latlng) {
      var options = {
        radius: 5.5,
        fillColor: "#ffffff",
        color: "#000",
        weight: 1,
        opacity: 1,
        fillOpacity: 0.8
      };
      return L.circleMarker(latlng, options)
    },
    filter: function(feature, layer) {
      if (ress10.includes(parseInt(feature.properties.FID))) {
        console.log('selected', parseInt(feature.properties.FID))
        selected_ress.push(feature)
      }
      return (ress10.includes(parseInt(feature.properties.FID)))
```

```
    },
    onEachFeature(feature, layer) {
      layer.setStyle({color:'#000',weight:'3'})
      if ((feature.properties.rating_score) != '-') {
        var toolTipContent = '<div>' + feature.properties.rating_score + ', ' + feature.properties.price_range +
'</div>'
      } else {
        var toolTipContent = '<div>' + 'N/A' + '</div>'
      }
      layer.bindTooltip(toolTipContent,{'permanent':true,'direction':'top','offset':new L.Point(0,-4)}).openTooltip()

      var popupContent = '<div><p>Name: ' + feature.properties.name + '</p><p>Price range: '
      + feature.properties.price_range + '</p><p>Rating score: ' + feature.properties.rating_score
      + '</p><p>Tags: ' + feature.properties.tags + '</p></div>'

      layer.bindPopup(popupContent, {
        offset: new L.Point(0, -5),
        closeButton: false,
        opacity: 0.8
      })

      layer.on({
        mouseover: function() {
          this.openPopup()
          this.setStyle({color:'blue',fillColor:'orange',weight: 3})
        },
        mouseout: function() {
          this.closePopup()
          this.setStyle({color: "#000",fillColor:'white',weight: 3})
        },
        click: function() {
          $('#ress_info').remove()
          var windowContent = "<div id='ress_info' style='padding-top:-10px; line-
height:20px'>_____<br><b>Name:</b> " + feature.properties.name +
'<br><b>Address:</b> ' + feature.properties.address + '<br><b>Price range:</b> '
          + feature.properties.price_range + '<br><b>Rating count:</b> ' + feature.properties.rating_count +
'<br><b>Rating score:</b> ' + feature.properties.rating_score
```

```
            + '<br>Tags: ' + feature.properties.tags + '<br><b>URL: </b><a href="url">' + feature.properties.url +
'</a></div>'
              $('#res_window').append(windowContent)
            }
          })
        }
    }).addTo(map)

    $('#ress_notif').remove()
    var panelContent = "<div id='ress_notif'><p>Number of restaurants within 10-min walk distance: <b>" +
ress10.length + '<b></p>'
    $('#res_window').append(panelContent)

    resInfoPopup(selected_ress)
    gg10.bringToFront()
}
function resInfoPopup(selected_ress) {
    for (i = 0; i < selected_ress.length; i ++) {
      console.log("RES_DET_WINDOW++++++++++",(selected_ress)[i])
    }
    console.log('ALL RESSSSS WITIN:==========',selected_ress)
}

function set_price_window() {
    $('#menu').hide()
    $('#price_window').show()
    $('.back_to_menu').off('click')
    $("input[type='radio']").off('click')
}
function set_res_window() {
    $('#menu').hide()
    $('#res_window').show()
    $('.back_to_menu').off('click')
    $("input[type='radio']").off('click')
}
function set_bus_window() {
    console.log('setting bus')
```

```
  $('#menu').hide()
  $('#bus_window').show()
  $('.back_to_menu').off('click')
}

function campusSymbols(data, data_uw, map) {
  //L.geoJson(data_uw).addTo(map)

  var styles3 = [{"color": "#000000", "fillColor": "#ffffcc","fillOpacity": '0.95',"weight": 0.5},
  {"color": "#000000", "fillColor": "#c7e9b4","fillOpacity": '0.95',"weight": 0.5},
  {"color": "#000000", "fillColor": "#7fcdbb","fillOpacity": '0.95',"weight": 0.5},
  {"color": "#000000", "fillColor": "#41b6c4","fillOpacity": '0.95',"weight": 0.5},
  {"color": "#000000", "fillColor": "#2c7fb8","fillOpacity": '0.95',"weight": 0.5},
  {"color": "#000000", "fillColor": "#253494","fillOpacity": '0.95',"weight": 0.5},
  ]
  var color3 = ["#ffffcc","#c7e9b4","#7fcdbb","#41b6c4","#2c7fb8","#253494"]
  legend3.remove()

  legend3.onAdd = function (map) {
    var div = L.DomUtil.create('div', 'info legend')
    div.innerHTML = '<div id="legend_title"><b>Time to campus landmark</b></div>'
      var grades = [0, 5, 10, 15, 20, 25]
    for (var i = grades.length - 1; i >= 0; i--) {
      if (i == grades.length - 1) {
        div.innerHTML += '<i style="background:' + color3[i] + '"></i>'
          + '> ' + (grades[i]) + ' min<br>';
      } else {
        div.innerHTML += '<i style="background:' + color3[i] + '"></i>'
          + (grades[i] + 1) + (grades[i + 1] ? ' &ndash; ' + grades[i + 1] + ' min<br>' : '+');
      }
    }
    return div;
  };

  legend3.addTo(map);
```

```
$('.back_to_menu').on({
  click: function() {
    if (map.hasLayer(gg2)) {
      map.removeLayer(gg2)
    }
    legend3.remove()
    if (map.hasLayer(gg_campus_walk)) {
      gg_campus_walk.clearLayers()
    }
    if (map.hasLayer(gg_campus_bike)) {
      gg_campus_bike.clearLayers()
    }
    if (map.hasLayer(gg_gordon)) {
      gg_gordon.clearLayers()
    }
    if (map.hasLayer(gg_bascom)) {
      gg_bascom.clearLayers()
    }
    if (map.hasLayer(gg_unionS)) {
      gg_unionS.clearLayers()
    }
    if (map.hasLayer(gg_nat)) {
      gg_nat.clearLayers()
    }

    $('#menu').show()
    $('#campus_window').hide()
  }
})
console.log("UW_landmarks", data_uw)

$("input[name='travel_mode']").click(function() {
  var rv1 = $("input[name='travel_mode']:checked").val()
  console.log('RV1:', rv1, typeof rv1)
  // I don't know why this is not working
  $("input[name='campus']").attr("checked", false);
```

```
if (map.hasLayer(gg_campus_walk)) {
  gg_campus_walk.clearLayers()
}
if (map.hasLayer(gg_campus_bike)) {
  gg_campus_bike.clearLayers()
}
if (map.hasLayer(gg2)) {
  gg2.clearLayers()
}
$("input[name='campus']").click(function() {
  var rv2 = $("input[name='campus']:checked").val();
  // Walk time
  if (rv1 == 'walk') {

    if (map.hasLayer(gg_gordon)) {
      map.removeLayer(gg_gordon)
    }
    if (map.hasLayer(gg_bascom)) {
      map.removeLayer(gg_bascom)
    }
    if (map.hasLayer(gg_unionS)) {
      map.removeLayer(gg_unionS)
    }
    if (map.hasLayer(gg_nat)) {
      map.removeLayer(gg_nat)
    }

    if (map.hasLayer(gg_campus_walk)) {
      gg_campus_walk.clearLayers()
    }
    if (map.hasLayer(gg_campus_bike)) {
      gg_campus_bike.clearLayers()
    }

    if (rv2 == 'gordon') {
      if (map.hasLayer(gg_gordon)) {
        map.removeLayer(gg_gordon)
```

```
        }
        if (map.hasLayer(gg_bascom)) {
          map.removeLayer(gg_bascom)
        }
        if (map.hasLayer(gg_unionS)) {
          map.removeLayer(gg_unionS)
        }
        if (map.hasLayer(gg_nat)) {
          map.removeLayer(gg_nat)
        }

        gg_gordon = L.geoJson(data_uw, {
          onEachFeature(feature, layer) {
            console.log("ADDDING BASOM")
            layer.unbindTooltip()
            if (feature.properties.name == 'Gordon Dining and Event Center') {
              var popupContent = 'Destination: ' + feature.properties.name
              layer.bindTooltip(popupContent, {'permanent':true,'direction':'top','offset':new L.Point(0,-
4),'opacity':0.9}).openTooltip()
            }
          }
        }).addTo(map)
      }
      if (rv2 == 'bascom') {
        if (map.hasLayer(gg_gordon)) {
          map.removeLayer(gg_gordon)
        }
        if (map.hasLayer(gg_bascom)) {
          map.removeLayer(gg_bascom)
        }
        if (map.hasLayer(gg_unionS)) {
          map.removeLayer(gg_unionS)
        }
        if (map.hasLayer(gg_nat)) {
          map.removeLayer(gg_nat)
        }
```

```
        gg_bascom = L.geoJson(data_uw, {
          onEachFeature(feature, layer) {
            layer.unbindTooltip()
            if (feature.properties.name == 'Bascom Hall') {
              var popupContent = 'Destination: ' + feature.properties.name
              layer.bindTooltip(popupContent, {'permanent':true,'direction':'top','offset':new L.Point(0,-
4),'opacity':0.9}).openTooltip()
            }
          }
        }).addTo(map)
      }
      if (rv2 == 'unionS') {
        if (map.hasLayer(gg_gordon)) {
          map.removeLayer(gg_gordon)
        }
        if (map.hasLayer(gg_bascom)) {
          map.removeLayer(gg_bascom)
        }
        if (map.hasLayer(gg_unionS)) {
          map.removeLayer(gg_unionS)
        }
        if (map.hasLayer(gg_nat)) {
          map.removeLayer(gg_nat)
        }

        gg_unionS = L.geoJson(data_uw, {
          onEachFeature(feature, layer) {
            layer.unbindTooltip()
            if (feature.properties.name == 'Union South') {
              var popupContent = 'Destination: ' + feature.properties.name
              layer.bindTooltip(popupContent, {'permanent':true,'direction':'top','offset':new L.Point(0,-
4),'opacity':0.9}).openTooltip()
            }
          }
        }).addTo(map)
      }
      if (rv2 == 'nat') {
```

```
            if (map.hasLayer(gg_gordon)) {
              map.removeLayer(gg_gordon)
            }
            if (map.hasLayer(gg_bascom)) {
              map.removeLayer(gg_bascom)
            }
            if (map.hasLayer(gg_unionS)) {
              map.removeLayer(gg_unionS)
            }
            if (map.hasLayer(gg_nat)) {
              map.removeLayer(gg_nat)
            }

            gg_nat = L.geoJson(data_uw, {
              onEachFeature(feature, layer) {
                layer.unbindTooltip()
                if (feature.properties.name == 'UW Natatorium') {
                  var popupContent = 'Destination: ' + feature.properties.name
                  layer.bindTooltip(popupContent, {'permanent':true,'direction':'top','offset':new L.Point(0,-
4),'opacity':0.9}).openTooltip()
                }
              }
            }).addTo(map)
          }

          gg_campus_walk = L.geoJson(data, {
            style: function(feature) {
              console.log('INTO')
              if (rv2 == 'gordon') {
                var gordon_walk = parseInt(feature.properties.to_campus.to_gordon_walk[1].slice(0,
feature.properties.to_campus.to_gordon_walk[1].indexOf(' ')))
                if (gordon_walk > 0 && gordon_walk <= 5) {
                  return styles3[0]
                }
                if (gordon_walk > 5 && gordon_walk <= 10) {
                  return styles3[1]
                }
```

```
            if (gordon_walk > 10 && gordon_walk <= 15) {
              return styles3[2]
            }
            if (gordon_walk > 15 && gordon_walk <= 20) {
              return styles3[3]
            }
            if (gordon_walk > 20 && gordon_walk <= 25) {
              return styles3[4]
            }
            if (gordon_walk > 25) {
              return styles3[5]
            }
          }
          if (rv2 == 'bascom') {
            var bascom_walk = parseInt(feature.properties.to_campus.to_bascom_walk[1].slice(0,
feature.properties.to_campus.to_bascom_walk[1].indexOf(' ')))
            if (bascom_walk > 0 && bascom_walk <= 5) {
              return styles3[0]
            }
            if (bascom_walk > 5 && bascom_walk <= 10) {
              return styles3[1]
            }
            if (bascom_walk > 10 && bascom_walk <= 15) {
              return styles3[2]
            }
            if (bascom_walk > 15 && bascom_walk <= 20) {
              return styles3[3]
            }
            if (bascom_walk > 20 && bascom_walk <= 25) {
              return styles3[4]
            }
            if (bascom_walk > 25) {
              return styles3[5]
            }

          }
          if (rv2 == 'unionS') {
```

```
            var unionS_walk = parseInt(feature.properties.to_campus.to_unionS_walk[1].slice(0,
feature.properties.to_campus.to_unionS_walk[1].indexOf(' ')))
            if (unionS_walk > 0 && unionS_walk <= 5) {
              return styles3[0]
            }
            if (unionS_walk > 5 && unionS_walk <= 10) {
              return styles3[1]
            }
            if (unionS_walk > 10 && unionS_walk <= 15) {
              return styles3[2]
            }
            if (unionS_walk > 15 && unionS_walk <= 20) {
              return styles3[3]
            }
            if (unionS_walk > 20 && unionS_walk <= 25) {
              return styles3[4]
            }
            if (unionS_walk > 25) {
              return styles3[5]
            }
          }
          if (rv2 == 'nat') {
            var nat_walk = parseInt(feature.properties.to_campus.to_nat_walk[1].slice(0,
feature.properties.to_campus.to_nat_walk[1].indexOf(' ')))
            if (nat_walk > 0 && nat_walk <= 5) {
              return styles3[0]
            }
            if (nat_walk > 5 && nat_walk <= 10) {
              return styles3[1]
            }
            if (nat_walk > 10 && nat_walk <= 15) {
              return styles3[2]
            }
            if (nat_walk > 15 && nat_walk <= 20) {
              return styles3[3]
            }
            if (nat_walk > 20 && nat_walk <= 25) {
```

```
            return styles3[4]
          }
          if (nat_walk > 25) {
            return styles3[5]
          }
        }
      },
      onEachFeature(feature, layer) {
        var ppct = ''

        if (rv2 == 'nat') {
          ppct = '<div><b>Walking</b><br>Travel time: ' + feature.properties.to_campus.to_nat_walk[1] +
'<br>Travel distance: ' + feature.properties.to_campus.to_nat_walk[0] + '</div>'
        }
        if (rv2 == 'gordon') {
          ppct = '<div><b>Walking</b><br>Travel time: ' + feature.properties.to_campus.to_gordon_walk[1] +
'<br>Travel distance: ' + feature.properties.to_campus.to_gordon_walk[0] + '</div>'
        }
        if (rv2 == 'bascom') {
          ppct = '<div><b>Walking</b><br>Travel time: ' + feature.properties.to_campus.to_bascom_walk[1] +
'<br>Travel distance: ' + feature.properties.to_campus.to_bascom_walk[0] + '</div>'
        }
        if (rv2 == 'unionS') {
          ppct = '<div><b>Walking</b><br>Travel time: ' + feature.properties.to_campus.to_unionS_walk[1] +
'<br>Travel distance: ' + feature.properties.to_campus.to_unionS_walk[0] + '</div>'
        }

        layer.bindPopup(ppct,{
          offset:new L.Point(0,-8)
        })
        layer.on({
          mouseover: function() {
            layer.setStyle({color:"#5555cc",weight:3})
            layer.openPopup()
          },
          mouseout: function() {
            layer.setStyle({"color": "#000000", "weight": 1})
```

```
            layer.closePopup()
          },
          click: function() {
            updateInfoWindow(feature)
          }
        })
      }
    }).addTo(map)
  }
  // Bike time
  if (rv1 == 'bike') {
    if (map.hasLayer(gg_gordon)) {
      map.removeLayer(gg_gordon)
    }
    if (map.hasLayer(gg_bascom)) {
      map.removeLayer(gg_bascom)
    }
    if (map.hasLayer(gg_unionS)) {
      map.removeLayer(gg_unionS)
    }
    if (map.hasLayer(gg_nat)) {
      map.removeLayer(gg_nat)
    }

    if (rv2 == 'gordon') {
      if (map.hasLayer(gg_gordon)) {
        map.removeLayer(gg_gordon)
      }
      if (map.hasLayer(gg_bascom)) {
        map.removeLayer(gg_bascom)
      }
      if (map.hasLayer(gg_unionS)) {
        map.removeLayer(gg_unionS)
      }
      if (map.hasLayer(gg_nat)) {
        map.removeLayer(gg_nat)
      }
```

```
        gg_gordon = L.geoJson(data_uw, {
          onEachFeature(feature, layer) {
            layer.unbindTooltip()
            if (feature.properties.name == 'Gordon Dining and Event Center') {
              var popupContent = 'Destination: ' + feature.properties.name
              layer.bindTooltip(popupContent, {'permanent':true,'direction':'top','offset':new L.Point(0,-
4),'opacity':0.9}).openTooltip()
            }
          }
        }).addTo(map)
      }
      if (rv2 == 'bascom') {
        if (map.hasLayer(gg_gordon)) {
          map.removeLayer(gg_gordon)
        }
        if (map.hasLayer(gg_bascom)) {
          map.removeLayer(gg_bascom)
        }
        if (map.hasLayer(gg_unionS)) {
          map.removeLayer(gg_unionS)
        }
        if (map.hasLayer(gg_nat)) {
          map.removeLayer(gg_nat)
        }

        gg_bascom = L.geoJson(data_uw, {
          onEachFeature(feature, layer) {
            layer.unbindTooltip()
            if (feature.properties.name == 'Bascom Hall') {
              var popupContent = 'Destination: ' + feature.properties.name
              layer.bindTooltip(popupContent, {'permanent':true,'direction':'top','offset':new L.Point(0,-
4),'opacity':0.9}).openTooltip()
            }
          }
        }).addTo(map)
      }
```

```
        if (rv2 == 'unionS') {
          if (map.hasLayer(gg_gordon)) {
            map.removeLayer(gg_gordon)
          }
          if (map.hasLayer(gg_bascom)) {
            map.removeLayer(gg_bascom)
          }
          if (map.hasLayer(gg_unionS)) {
            map.removeLayer(gg_unionS)
          }
          if (map.hasLayer(gg_nat)) {
            map.removeLayer(gg_nat)
          }

          gg_bascom = L.geoJson(data_uw, {
            onEachFeature(feature, layer) {
              layer.unbindTooltip()
              if (feature.properties.name == 'Union South') {
                var popupContent = 'Destination: ' + feature.properties.name
                layer.bindTooltip(popupContent, {'permanent':true,'direction':'top','offset':new L.Point(0,-
4),'opacity':0.9}).openTooltip()
              }
            }
          }).addTo(map)
        }
        if (rv2 == 'nat') {
          if (map.hasLayer(gg_gordon)) {
            map.removeLayer(gg_gordon)
          }
          if (map.hasLayer(gg_bascom)) {
            map.removeLayer(gg_bascom)
          }
          if (map.hasLayer(gg_unionS)) {
            map.removeLayer(gg_unionS)
          }
          if (map.hasLayer(gg_nat)) {
            map.removeLayer(gg_nat)
```

```
        }

        gg_nat = L.geoJson(data_uw, {
          onEachFeature(feature, layer) {
            layer.unbindTooltip()
            if (feature.properties.name == 'UW Natatorium') {
              var popupContent = 'Destination: ' + feature.properties.name
              layer.bindTooltip(popupContent, {'permanent':true,'direction':'top','offset':new L.Point(0,-
4),'opacity':0.9}).openTooltip()
            }
          }
        }).addTo(map)
      }
      if (map.hasLayer(gg_campus_walk)) {
        gg_campus_walk.clearLayers()
      }
      if (map.hasLayer(gg_campus_bike)) {
          gg_campus_bike.clearLayers()
      }
      gg_campus_bike = L.geoJson(data, {
        style: function(feature) {
          console.log('INTO')
          if (rv2 == 'gordon') {
            var gordon_bike = parseInt(feature.properties.to_campus.to_gordon_bicycling[1].slice(0,
feature.properties.to_campus.to_gordon_bicycling[1].indexOf(' ')))
            if (gordon_bike > 0 && gordon_bike <= 5) {
              console.log('BIKE TIME: GORDON', gordon_bike)
              return styles3[0]
            }
            if (gordon_bike > 5 && gordon_bike <= 10) {
              return styles3[1]
            }
            if (gordon_bike > 10 && gordon_bike <= 15) {
              return styles3[2]
            }
            if (gordon_bike > 15 && gordon_bike <= 20) {
              return styles3[3]
```

```
        }
        if (gordon_bike > 20 && gordon_bike <= 25) {
          return styles3[4]
        }
        if (gordon_bike > 25) {
          return styles3[5]
        }
      }
      if (rv2 == 'bascom') {
        var bascom_bike = parseInt(feature.properties.to_campus.to_bascom_bicycling[1].slice(0,
feature.properties.to_campus.to_bascom_bicycling[1].indexOf(' ')))
        if (bascom_bike > 0 && bascom_bike <= 5) {
          return styles3[0]
        }
        if (bascom_bike > 5 && bascom_bike <= 10) {
          return styles3[1]
        }
        if (bascom_bike > 10 && bascom_bike <= 15) {
          return styles3[2]
        }
        if (bascom_bike > 15 && bascom_bike <= 20) {
          return styles3[3]
        }
        if (bascom_bike > 20 && bascom_bike <= 25) {
          return styles3[4]
        }
        if (bascom_bike > 25) {
          return styles3[5]
        }

      }
      if (rv2 == 'unionS') {
        var unionS_bike = parseInt(feature.properties.to_campus.to_unionS_bicycling[1].slice(0,
feature.properties.to_campus.to_unionS_bicycling[1].indexOf(' ')))
        if (unionS_bike > 0 && unionS_bike <= 5) {
          return styles3[0]
        }
```

```
            if (unionS_bike > 5 && unionS_bike <= 10) {
              return styles3[1]
            }
            if (unionS_bike > 10 && unionS_bike <= 15) {
              return styles3[2]
            }
            if (unionS_bike > 15 && unionS_bike <= 20) {
              return styles3[3]
            }
            if (unionS_bike > 20 && unionS_bike <= 25) {
              return styles3[4]
            }
            if (unionS_bike > 25) {
              return styles3[5]
            }
          }
          if (rv2 == 'nat') {
            var nat_bike = parseInt(feature.properties.to_campus.to_nat_bicycling[1].slice(0,
feature.properties.to_campus.to_nat_bicycling[1].indexOf(' ')))
            if (nat_bike > 0 && nat_bike <= 5) {
              return styles3[0]
            }
            if (nat_bike > 5 && nat_bike <= 10) {
              return styles3[1]
            }
            if (nat_bike > 10 && nat_bike <= 15) {
              return styles3[2]
            }
            if (nat_bike > 15 && nat_bike <= 20) {
              return styles3[3]
            }
            if (nat_bike > 20 && nat_bike <= 25) {
              return styles3[4]
            }
            if (nat_bike > 25) {
              return styles3[5]
            }
```

```
        }
      },
      onEachFeature(feature, layer) {
        var ppct = ''

        if (rv2 == 'nat') {
          ppct = '<div><b>Bicycling</b><br>Travel time: ' + feature.properties.to_campus.to_nat_bicycling[1]
+ '<br>Travel distance: ' + feature.properties.to_campus.to_nat_bicycling[0] + '</div>'
        }
        if (rv2 == 'gordon') {
          ppct = '<div><b>Bicycling</b><br>Travel time: ' +
feature.properties.to_campus.to_gordon_bicycling[1] + '<br>Travel distance: ' +
feature.properties.to_campus.to_gordon_bicycling[0] + '</div>'
        }
        if (rv2 == 'bascom') {
          ppct = '<div><b>Bicycling</b><br>Travel time: ' +
feature.properties.to_campus.to_bascom_bicycling[1] + '<br>Travel distance: ' +
feature.properties.to_campus.to_bascom_bicycling[0] + '</div>'
        }
        if (rv2 == 'unionS') {
          ppct = '<div><b>Bicycling</b><br>Travel time: ' +
feature.properties.to_campus.to_unionS_bicycling[1] + '<br>Travel distance: ' +
feature.properties.to_campus.to_unionS_bicycling[0] + '</div>'
        }

        layer.bindPopup(ppct,{
          offset:new L.Point(0,-8)
        })
        layer.on({
          mouseover: function() {
            layer.setStyle({color:"#5555cc",weight:3})
            layer.openPopup()
          },
          mouseout: function() {
            layer.setStyle({"color": "#000000", "weight": 1})
            layer.closePopup()
          },
```

```
            click: function() {
              updateInfoWindow(feature)
            }
          })
        }
      }).addTo(map)
    }
  })
})
}

function resSymbols(data, data5, data10, data_res, map) {
  if (map.hasLayer(gg2)) {
    gg2.clearLayers()
  }
  var domain_all = []

  L.geoJson(data5, {
    onEachFeature(feature, layer) {
      res_count_5 = feature.properties.reachable_restaurants.length
      console.log('THIS_1010101010:', res_count_5)
      domain_all.push(res_count_5)
    }
  })

  var clusters = ss.ckmeans(domain_all, 6)
  var clusterBreaks = [0]
  console.log(clusters)

  // Get the largest value for each cluster as cluster breaks
  for (i = 0; i < clusters.length; i ++) {
    clusterBreaks.push(clusters[i][clusters[i].length - 1])
  }
  console.log('Class breaks', clusterBreaks)

  var gg_res = L.geoJson(data_res, {
    pointToLayer: function(feature,latlng) {
```

```
    var options = {
      radius: 6,
      fillColor: "#33ff33",
      color: "#000",
      weight: 1,
      opacity: 1,
      fillOpacity: 0.8
    };
    return L.circleMarker(latlng, options)
  }
}).addTo(map)
gg_res.bringToFront()

var styles = [{"color": "#000000", "fillColor": "#ffffb2","fillOpacity": '0.95',"weight": 1},
{"color": "#000000", "fillColor": "#fed976","fillOpacity": '0.95',"weight":1},
{"color": "#000000", "fillColor": "#feb24c","fillOpacity": '0.95',"weight":1},
{"color": "#000000", "fillColor": "#fd8d3c","fillOpacity": '0.95',"weight":1},
{"color": "#000000", "fillColor": "#fc4e2a","fillOpacity": '0.95',"weight":1},
{"color": "#000000", "fillColor": "#e31a1c","fillOpacity": '0.95',"weight":1},
{"color": "#000000", "fillColor": "#b10026","fillOpacity": '0.95',"weight": 1},
]

$("input[name='ress']").click(function(){
  if (map.hasLayer(gg0)) {
    gg0.clearLayers()
  }
  var radioValue = $("input[name='ress']:checked").val();
  console.log("CLIKED?" + radioValue);
  if (map.hasLayer(gg10_b)) {
    map.removeLayer(gg10_b)
  }
  if (map.hasLayer(gg5_b)) {
    map.removeLayer(gg5_b)
  }
  if (map.hasLayer(gg10)) {
    map.removeLayer(gg10)
  }
```

```
  if (map.hasLayer(gg5)) {
    map.removeLayer(gg5)
  }
  if (radioValue == '10_ress') {
    color10_ress(gg_res, clusterBreaks, styles, data, data10, data_res, map)
  } else if (radioValue == '5_ress') {
    color5_ress(gg_res, clusterBreaks, styles, data, data5, data_res, map)
  }
})
$('.back_to_menu').on({
  click: function() {
    //console.log("BACK?",map.hasLayer(gg2))
    legend2.remove()
    if (map.hasLayer(gg2)) {
      map.removeLayer(gg2)
    }
    if (map.hasLayer(gg10_b)) {
      map.removeLayer(gg10_b)
    }
    if (map.hasLayer(gg5_b)) {
      map.removeLayer(gg5_b)
    }
    if (map.hasLayer(gg10)) {
      map.removeLayer(gg10)
    }
    if (map.hasLayer(gg5)) {
      map.removeLayer(gg5)
    }
    if (map.hasLayer(gg0)) {
      map.removeLayer(gg0)
    }
    if (map.hasLayer(gg_res)) {
      map.removeLayer(gg_res)
    }
    //gg_res.clearLayers()
    $('#menu').show()
    $('#res_window').hide()
```

```
    }
  })
}
function color5_ress(gg_res, clusterBreaks, styles, data, data5, data_res, map) {

  var colors = ["#ffffb2","#fed976","#feb24c","#fd8d3c","#fc4e2a","#e31a1c"]
  legend2.remove()

  legend2.onAdd = function (map) {
    var div = L.DomUtil.create('div', 'info legend')
    div.innerHTML = '<div id="legend_title"><b>Number of restaurants within 5-min walk distance</b></div>'
      var grades = [-1, 4, 11, 24, 39, 56, 77]
    for (var i = grades.length - 2; i >= 0; i--) {
      div.innerHTML += '<i style="background:' + colors[i] + '"></i>'
        + (grades[i] + 1) + (grades[i + 1] ? ' &ndash; ' + grades[i + 1] + '<br>' : '+');
    }
    return div;
  };

  legend2.addTo(map);

  clusterBreaks.push(999)
  gg0 = L.geoJson(data, {
    style: function(feature) {
      console.log(data5, feature.properties.FID)
      for (i = 0; i < 6; i ++) {
        var resCt = data5.features[feature.properties.FID].properties.reachable_restaurants.length
        if (resCt >= clusterBreaks[i] && resCt <= clusterBreaks[i + 1]) {
          return styles[i]
        }
      }
    },
    onEachFeature(feature, layer) {
      layer.on({
        mouseover: function() {
          layer.setStyle({color:"#5555cc",weight:3})
```

```
        },
        mouseout: function() {
          layer.setStyle({"color": "#000000", "weight": 1})
        },
        click: function() {
          $('#ress_info').remove()

          console.log('BUILDING BOUND:',feature)
          bound_res_5(map,feature.properties.FID, data5, data_res)
          updateInfoWindow(feature)

        }
      })
    }
  }).addTo(map)

  gg_res.bringToFront()
  // create basic a basic point layer containing all restaurants
}

function color10_ress(gg_res, clusterBreaks, styles, data, data10, data_res, map) {

  var colors = ["#ffffb2","#fed976","#feb24c","#fd8d3c","#fc4e2a","#e31a1c","#b10026"]
  legend2.remove()

  legend2.onAdd = function (map) {
    var div = L.DomUtil.create('div', 'info legend')
    div.innerHTML = '<div id="legend_title"><b>Number of restaurants within 10-min walk distance</b></div>'
      grades = [-1, 4, 11, 24, 39, 56, 77, 'more than 77'],
      labels = [];
    for (var i = grades.length - 2; i >= 0; i--) {
      if (i == grades.length - 2) {
        div.innerHTML += '<i style="background:' + colors[i] + '"></i>'
          + grades[i +1] + '<br>'
      } else {
        div.innerHTML += '<i style="background:' + colors[i] + '"></i>'
          + (grades[i] + 1) + (grades[i + 1] ? ' &ndash; ' + grades[i + 1] + '<br>' : '+');
```

```
    }
  }
  return div;
};

legend2.addTo(map);
clusterBreaks.push(999)
gg0 = L.geoJson(data, {
  /*style: function() {
    return ({"color": "#999999", "fillColor": "#cccccc","fillOpacity": 1,"weight": 1})
  },*/
  style: function(feature) {
    console.log(data10, feature.properties.FID)
    /*for (i = 0; i < 796; i ++) {
      if (data5.features[i].id == feature.properties.FID) {*/
    for (i = 0; i < 7; i ++) {
      var resCt = data10.features[feature.properties.FID].properties.reachable_restaurants.length
      if (resCt >= clusterBreaks[i] && resCt <= clusterBreaks[i + 1]) {
        return styles[i]
      }
    }
  },
  onEachFeature(feature, layer) {
    layer.on({
      mouseover: function() {
        layer.setStyle({color:"#5555cc",weight:3})
      },
      mouseout: function() {
        layer.setStyle({"color": "#000000", "weight": 1})
      },
      click: function() {
        $('#ress_info').remove()

        console.log('BUILDING BOUND:',feature)
        bound_res_10(map,feature.properties.FID, data10, data_res)
        updateInfoWindow(feature)
```

```
        }
      })
    }
  }).addTo(map)

  gg_res.bringToFront()
}

function priceSymbols(data, map) {
  var domainDA = []
  var domainL = []

  for (var i = 0; i < 796; i ++) {
    prop = data.features[i]['properties']
    console.log('prop',prop['Floorplans'])
    for (var keyFlp in prop['Floorplans']) {
      if (prop['Floorplans'][keyFlp]['Individual_Average']) {
        var val = parseFloat(prop['Floorplans'][keyFlp]['Individual_Average']['avg_price'])
        console.log(val)
        if (isNaN(val) === false) {
          var key = prop['Floorplans'][keyFlp]['Title']
          console.log("TITLE", key)
          var Dict = {}
          Dict[key] = val
          domainDA.push(Dict)
          domainL.push(val)
        }
      }
    }
  }
  console.log('length', domainDA, domainDA.length)
  for (i = 0; i < domainDA.length; i ++) {
    console.log('DOMAIN',JSON.stringify(domainDA[i]))
  }

  clusters = ss.ckmeans(domainL, 6)
  clusterBreaks = [0]
```

```
console.log(clusters)

for (i = 0; i < clusters.length; i ++) {
  clusterBreaks.push(clusters[i][clusters[i].length - 1])
}
clusterBreaks.push(9999)
console.log('Class breaks', clusterBreaks)

var styles = [{"color": "#000000", "fillColor": "#ffffb2","fillOpacity": '0.95',"weight": 1},
{"color": "#000000", "fillColor": "#fed976","fillOpacity": '0.95',"weight": 1},
{"color": "#000000", "fillColor": "#feb24c","fillOpacity": '0.95',"weight": 1},
{"color": "#000000", "fillColor": "#fd8d3c","fillOpacity": '0.95',"weight": 1},
{"color": "#000000", "fillColor": "#f03b20","fillOpacity": '0.95',"weight": 1},
{"color": "#000000", "fillColor": "#bd0026","fillOpacity": '0.95',"weight": 1},
]
var colors = ["#ffffb2", "#fed976","#feb24c","#fd8d3c","#f03b20","#bd0026"]

legend1.onAdd = function (map) {

  var div = L.DomUtil.create('div', 'info legend')
  div.innerHTML = '<div id="legend_title"><b>Price per person</b></div>'
    grades = [0, 535, 690, 862.5, 1065, 1310, 1830],
    labels = [];

  for (var i = grades.length - 2; i >= 0; i--) {
    div.innerHTML += '<i style="background:' + colors[i] + '"></i> $'
      + grades[i] + (grades[i + 1] ? ' &ndash; $' + grades[i + 1] + '<br>' : '+')
  }
  div.innerHTML += '<i style="background:' + '#cccccc' + '"></i>no such floorplan<br>';

  return div;
};

legend1.addTo(map);

gg2 = L.geoJson(data)
```

```
  var noDataStyle = {"color": "#999999", "fillColor": "#cccccc","fillOpacity": 1,"weight": 1}
  $("input[type='radio']").click(function(){

    if (map.hasLayer(gg2)) {
      gg2.clearLayers()
    }
    var radioValue = $("input[name='bedrooms']:checked").val();
    console.log("Your are a - " + radioValue);
    gg2 = L.geoJson(data, {
      style: function(feature) {
        style1 = {}
        var keyL = Object.keys(feature['properties']['Floorplans'])
        for (i = 0; i < keyL.length; i ++) {
          var flpKey = keyL[i]
          if (feature['properties']['Floorplans'][flpKey]['Title'] == radioValue &&
feature['properties']['Floorplans'][flpKey]['Individual_Average']) {
            var avgPrice = parseFloat(feature['properties']['Floorplans'][flpKey]['Individual_Average']['avg_price'])
            if (isNaN(avgPrice) === false) {
              for (i = 0; i < clusterBreaks.length - 1; i ++) {
                if (parseFloat(avgPrice) > clusterBreaks[i] && (parseFloat(avgPrice) < clusterBreaks[i + 1])) {
                  style1 = styles[i]
                  return style1
                }
              }
            }
          }
          else if ((feature['properties']['Floorplans'][flpKey]['Title'] == '6 Bedroom(s)' ||
feature['properties']['Floorplans'][flpKey]['Title'] == '7 Bedroom(s)' ||
feature['properties']['Floorplans'][flpKey]['Title'] == '8 Bedroom(s)' ||
feature['properties']['Floorplans'][flpKey]['Title'] == '9 Bedroom(s)' ||
feature['properties']['Floorplans'][flpKey]['Title'] == '10 Bedroom(s)' ||
feature['properties']['Floorplans'][flpKey]['Title'] == '11 Bedroom(s)' ||
feature['properties']['Floorplans'][flpKey]['Title'] == '12 Bedroom(s)' ||
feature['properties']['Floorplans'][flpKey]['Title'] == '13 Bedroom(s)') &&
feature['properties']['Floorplans'][flpKey]['Individual_Average']) {
            var avgPrice = parseFloat(feature['properties']['Floorplans'][flpKey]['Individual_Average']['avg_price'])
            if (isNaN(avgPrice) === false) {
```

```
            for (i = 0; i < clusterBreaks.length - 1; i ++) {
              if (parseFloat(avgPrice) > clusterBreaks[i] && (parseFloat(avgPrice) < clusterBreaks[i + 1])) {
                style1 = styles[i]
                return style1
              }
            }
          }
        }
      }
      return noDataStyle
    },
    onEachFeature(feature, layer) {
      var stylePass = style1

      if (Object.keys(stylePass).length === 0) {
        var stylePass = {"color": "#999999", "fillColor": "#cccccc","fillOpacity": 1,"weight": 1}
      }
      console.log('passssssss', style1, Object.values(stylePass)[0])
      createInfoPopup(feature, layer, stylePass, radioValue)
    }
  }).addTo(map)
});

$('.back_to_menu').on({
  click: function() {
    if (map.hasLayer(gg2)) {
      map.removeLayer(gg2)
    }
    legend1.remove()
    $('#menu').show()
    $('#price_window').hide()
  }
})
}

function createInfoPopup(feature,layer,style,radioValue) {
```

```
console.log('passed style', style)
prt = feature.properties
var addr = prt.Address
var name = prt.Name
var typeH = prt.Type_of_housing
var url1 = prt.URL1
var  url2 = prt.URL2
var flp = prt.Floorplans
var det = prt.Detailed_type
var simp = prt.Simplified_type
var yrB = prt.Year_built
var popupCntnt = '<p><b>Address:</b> ' + addr + '</p><p><b>Name:</b> ' + name

layer.bindPopup(popupCntnt,{
  offset:new L.Point(0,-10)
})

$('#unitTT').change(function() {
  var content = ''
  if (this.checked) {
    layer.bindTooltip(simp, {'permanent':true,'direction':'center','opacity':'0.8'})
  } else {
    layer.unbindTooltip()
  }
})

$(layer).on({
  mouseover: function() {
    layer.openPopup()
    layer.setStyle({color:"#7777ff",weight:2})
  },
  mouseout: function() {
    layer.closePopup()
    layer.setStyle(style)
  },
  click: function() {
    layer.openPopup()
```

```
        layer.setStyle({color: "#7777ff"})
        updateInfoWindow(feature)
      }
    })
}

function updateInfoWindow(feature) {
    $('.updatable').remove()
    $('#tb_flp').remove()

    var prt = feature.properties
    var addr = prt.Address
    var name = prt.Name
    //var typeH = prt.Type_of_housing
    var url1 = prt.URL1
    var  url2 = prt.URL2
    var flp = prt.Floorplans
    var det = prt.Detailed_type
    var simp = prt.Simplified_type
    var yrB = prt.Year_built
    var popupCntnt = '<p><b>Address:</b> ' + addr + '</p><p><b>Name:</b> ' + name

    var panelCntnt = '<div class="updatable"><p id="updatable-basic"><b>Name: </b>' + name + '</p>'
    +'<p id="updatable-basic"><b>Address: </b>' + addr + '</p>'
    //+'<p id="updatable-basic"><b>Type of housing: </b>' + typeH + '</p>'
    +'<p id="updatable-basic"><b>Type: </b>' + det + '</p>'
    +'<p id="updatable-basic"><b>Year built: </b>' + yrB + '</p>'
    +'<p id="updatable-basic"><b>URL1: </b><br><a href="url">' + url1 + '</a></p>'

    if (url2 != '-') {
      panelCntnt += '<p id="updatable-basic"><b>URL2: </b><br><a href="url">' + url2 + '</a></p>'
    }
    else {
      panelCntnt += '<p id="updatable-basic"><b>URL2: </b><br>' + url2 + '<br><br></p>'
    }

    tableC = '<table id="tb_flp">'
```

```
var keyDict = Object.keys(flp)
tableC += '<tr>' + '<th>Title</th>' + '<th>Price</th>' + '<th>Area</th>' +'<th>Price/sq ft</th>' + '</tr>'
for (i = 0; i < keyDict.length; i ++) {

  var flpN = flp[keyDict[i]]
  var title = flpN['Title']

  tableC += '<tr><td>' + title + '</td>'
  // Append basic floorplan info to table
  if ('Price' in flpN) {
    var price = flpN['Price']
    var area = flpN['Area']
    var ppa = flpN['Price/Unit_area']

    if (price.length == 2) {
      priceStr = price[0] + '-' + price[1]
    } else {
      priceStr = price[0]
    }

    if (area.length == 2) {
      areaStr = area[0] + '-' + area[1]
    } else {
      areaStr = area[0]
    }

    if (ppa.length == 2) {
      ppaStr = ppa[0] + '-' + ppa[1]
    } else {
      ppaStr = ppa[0]
    }

    tableC += '<td>' + priceStr + '</td><td>' + areaStr + '</td><td>' + ppaStr + '</td>'
  }
  tableC += '</tr>'

  for (ex = 0; ex < 4; ex ++) {
```

```
        var extraKey = 'Extra_Feature_' + ex.toString()
        //console.log('thisssssss',flpN)
        if (extraKey in flpN) {
          var extra = flpN[extraKey]
          var extraTitle = extra['Sub_title']

          var extraPrice = extra['Price']
          if (extraPrice.length == 2) {
            var extraPriceStr = extraPrice[0] + '-' + extraPrice[1]
          } else {
            var extraPriceStr = extraPrice[0]
          }

          var extraArea = extra['Area']
          //console.log('AREAAAAAAAAAAA', extraArea.length ,extraArea)
          if (extraArea.length == 2) {
            var extraAreaStr = extraArea[0] + '-' + extraArea[1]
          } else {
            var extraAreaStr = extraArea[0]
          }

          var extraPPA = extra['Price/Unit_area']
            var extraPPAStr = extraPPA[0]

          tableC += '<tr id="tb_extra">'/* + '<td>Extra Layout' + ex*/ + '<td>   &nbspw/ extra
feature: ' + extraTitle + '</td><td>'
          + extraPriceStr + '</td><td>' + extraAreaStr + '</td><td>' + extraPPAStr + '</td></tr>'
        }
      }
    }

  tableC += '</table>'
  $('#price_window').append(panelCntnt)
  $('#price_window').append(tableC)
  $('#res_window').append(panelCntnt)
  $('#campus_window').append(panelCntnt)
}
```

```
function createBaseSymbols(data, map) {
  var basic = L.geoJson(data, {
    style: function(feature) {
      var style1 = {"color": "#999999", "fillColor": "#cccccc","fillOpacity": 1,"weight": 1}
      return style1
    }
  }).addTo(map)
}

function updatePropSymbols(map, attribute) {
  // map.eachLayer is used to iterate through each single Circle Marker
  map.eachLayer(function(layer) {
    // proceed if feature, and properties with the name of attribute(the column name that starts with "gdp_") exist
    // in this layer (represented by this Circle Marker)
    if (layer.feature && layer.feature.properties[attribute]) {
      // get all properties -- all gdp data under this layer)
      var proprts = layer.feature.properties

      // calculate radius of the reset Circle Marker
      var radius = calcR(proprts[attribute])

      createPopupCntnt(layer, radius, proprts, attribute)
    }
  })
}

function updateNotif(index) {
  $('#notif').remove()
  iNum = parseInt(index)
  // add a notification that shows current year the slider is on
  // If index of the slider is greater than 5, use a different equation, because 2008, 2009, and 2010 does not
  // follow the previous pattern of intervals of 5 (1980, 1985, 1990, 1995, 2000 and 2005).
   var year = (iNum > 5) ? (2002 + iNum): (1980 + 5 * iNum)
   var notif = '<div id="notif">Showing GDP of year ' + year + '</div>'
   // If '#updatable'(the info panel that shows data of clicked layer in the '#panel') is still empty (still initial value in
index.html),
```

```
  // append notification at the end of the panel, otherwise append before '#updatable'
  // This ensure that the relative position of "#updatable" and "#notif" remains unchanged when the user performs
any
  // clicking or sliding

   if (!$('#updatable').is(':empty')) {
        $(notif).insertBefore('#updatable')
   } else {
        $('#panel').append(notif)
   }
}

function enableMenuButtons(response, map) {
  // append a series of buttons to '#panel', '#compare' and '#compareResult' interfaces
  $('.back_to_menu').off('click')

  $('#menu').append("<p id='title_menu'>UW-Madison Off-Campus Housing Options Exploration Tool</p>")

  $('#menu').append("<p id='title_compare'>Compare<br>_____</p>")

  // Call price-compare function
  $('#menu').append("<br><button id='price_compare'>Compare Price </button>")
  $('#price_compare').off('click')
  $('#price_compare').on({
    'click': function() {
      console.log('Now comparing prices')
      set_price_window()
      priceSymbols(response, map)
    }
  })

  $('#menu').append("<br><button id='area_compare'>Compare Area<br>(unavailable now)</button>")
  $('#menu').append("<br><button id='ppa_compare'>Compare Price/sqf<br>(unavailable now)</button>")

  $('#menu').append("<p id='title_proximity'>Proximities<br>_____</p>")
  $('#menu').append("<br><button id='campus_proximity'>To Campus</button>")
  $('#campus_proximity').on({
```

```
    'click': function() {
      set_campus_window()
      getUW(response, map)
    }
  })

  $('#menu').append("<br><button id='res_proximity'>To Restaurants</button>")
  // Call restaurant-proximity function
  $('#res_proximity').on({
    'click': function() {
      console.log('Now getting restaurant proximities')
      set_res_window()
      getData2(response, map)
    }
  })

  $('#menu').append("<br><button id='bus_proximity'>To Bus Stops</button>")
  $('#bus_proximity').on({
    'click': function() {
      set_bus_window()
      getData3(response, map)
    }
  })
  // Only shows '#panel' at the inital stage, '#compare' and '#compareResult' interfaces are fired later upon clicking
  // certain buttons.
  $('#compare').hide()
  $('#compareResult').hide()
}

function busSymbols(data, dataR, dataS, map) {
  if (map.hasLayer(gg2)) {
    gg2.clearLayers()
  }
  console.log('CORRECT!')
  $('#bus_side_window').empty()
  gg_routes = L.geoJson(dataR, {
    onEachFeature(feature, layer) {
```

```
        layer.setStyle({'color':'#555555', 'weight':2})
        var window_item = 'Route ' + feature.properties.route_shor
        + '</a>  <label><input type="radio" name="routes" value='
        + feature.properties.route_shor + '>Show on map</label>'
        + '   <b>URL:</b> <a href="'+ feature.properties.route_url + '">Webpage on metro
transit<br><br>'
        $('#bus_side_window').append(window_item)
      }
    }).addTo(map)

    $("input[name='routes']").click(function() {
      map.removeLayer(gg_routes)

      if (map.hasLayer(gg_route_sel)) {
        map.removeLayer(gg_route_sel)
      }

      var radioValue = parseInt($("input[name='routes']:checked").val());

      gg_route_sel = L.geoJson(dataR, {
        onEachFeature(feature, layer) {
          if (feature.properties.route_shor == radioValue) {
            console.log('settin')
            layer.setStyle({'color':'#6666dd', 'weight':10, 'opacity': 0.9})
          } else {
            layer.setStyle({'color':'#555555', 'weight':2})
          }
        }
      }).addTo(map)
    })

    gg_stops = L.geoJson(dataS, {

      pointToLayer: function(feature, latlng) {
        var stopIcon = new busIcon({iconUrl: 'data/icon/stop3.png'})
        return L.marker(latlng, {icon: stopIcon})
      },
```

```
    onEachFeature: function(feature, layer) {

      var tooltipContent = '<div>' + feature.properties.Routes + '</div>'

      layer.bindTooltip(tooltipContent, {
        'direction':'top', 'offset':new L.Point(0, -25)
      }).openTooltip()
      layer.on({
        click: function() {
          pass_routes = []
            console.log('Line Properties:', feature.properties.Routes)
          pass_routes = feature.properties.Routes
          updateRoutes(pass_routes, dataR, map)
        }
      })
    }
  }).addTo(map)
  $('.back_to_menu').on({
    click: function() {
      if (map.hasLayer(gg2)) {
        map.removeLayer(gg2)
      }
      if (map.hasLayer(gg_stops)) {
        gg_stops.clearLayers()
      }
      if (map.hasLayer(gg_routes)) {
        gg_routes.clearLayers()
      }
      if (map.hasLayer(gg_route_sel)) {
        gg_route_sel.clearLayers()
      }

      $('#menu').show()
      $('#bus_window').hide()
    }
  })
}
```

```
function resetRoutes(dataR, map) {
   if (map.hasLayer(gg_routes)) {
     map.removeLayer(gg_routes)
   }
   gg_routes = L.geoJson(dataR, {
     onEachFeature(feature, layer) {
       layer.setStyle({'color':'#777777', 'weight':2})
     }
   }).addTo(map)
}

function updateRoutes(pass_routes, dataR, map) {
   console.log("PASSSINGG:", pass_routes, dataR)
   if (map.hasLayer(gg_routes)) {
     map.removeLayer(gg_routes)
   }
   if (map.hasLayer(gg_route_sel)) {
     console.log('REMOVING?????', gg_route_sel)
     map.removeLayer(gg_route_sel)
   }
   gg_routes = L.geoJson(dataR, {
     onEachFeature: function(feature,layer) {
       if (pass_routes.includes(feature.properties.route_shor)) {
         console.log('UPDATING????', pass_routes,feature.properties.route_shor)
         layer.setStyle({'color':'#6666dd', 'weight':10, 'opacity': 0.4})
       } else {
         layer.setStyle({'color': '#777777', 'weight':2})
       }
     }
   }).addTo(map)
}

function createRess(data, map) {
   L.geoJson(data, {
     pointToLayer: function(feature,latlng) {
       var options = {
```

```
            radius: 2,
            fillColor: "#ff7800",
            color: "#000",
            weight: 1,
            opacity: 1,
            fillOpacity: 0.8
          };
          return L.circleMarker(latlng, options)
        }
    }).addTo(map)
}

function getData3(data, map) {
    console.log('getting-data',data)
    $.when(ajax_routes(), ajax_stops()).then(function(a1, a2) {
        console.log('DONE', a1)
        console.log(a2)
        busSymbols(data, a1, a2, map)
    })
}

function ajax_routes() {
    return $.ajax("data/routes.geojson", {
        dataType: "json",
        success: function(response) {
            return response
        }
    })
}
function ajax_stops() {
    return $.ajax("data/stops.geojson", {
        dataType: "json",
        success: function(response) {
            return response
        }
    })
}
```

```
// Getting data through AJAX
function getData2(data, map) {
  console.log('getting-data',data)
  $.when(ajax5(), ajax10(), ajax_res()).then(function(a1, a2, a3) {
    console.log('DONE',a1, a2, a3)
    resSymbols(data, a1[0], a2[0], a3[0], map)
  })
}

function ajax5() {
  return $.ajax("data/5_min_walk_polygon+reachable_restaurant_id.geojson", {
    /* specify the file format the ajax method is to call*/
    dataType: "json",
    success: function(response) {
      //var attributes = processData(response)
      console.log('getting ajax5')
      return response
    }
  })
}

function ajax10() {
  return $.ajax("data/10_min_walk_polygon+reachable_restaurant_id.geojson", {
    /* specify the file format the ajax method is to call*/
    dataType: "json",
    success: function(response) {
      //var attributes = processData(response)
      console.log('getting ajax10')
      return response
    }
  })
}

function ajax_res() {
  return $.ajax("data/all_restaurants.geojson", {
    /* specify the file format the ajax method is to call*/
```

```
    dataType: "json",
    success: function(response) {
      //var attributes = processData(response)
      console.log('getting res')
      return response
    }
  })
}

//function to retrieve the data and place it on the map
function getData(map) {
  $.ajax("data/WITH_to_campus_2.geojson", {
    /* specify the file format the ajax method is to call*/
    dataType: "json",
    success: function(response) {
      //var attributes = processData(response)
      enableMenuButtons(response, map)
      createBaseSymbols(response, map)
    }
  })
}
function getUW(data, map) {
  console.log('UWUWgetting')
  $.ajax("data/uw_landmarks.geojson", {
    /* specify the file format the ajax method is to call*/
    dataType: "json",
    success: function(response) {
      console.log("UW0", response)
      campusSymbols(data, response, map)
    }
  })
}

$(document).ready(createMap);
```

7.10. Appendix 10: style.css

```
/* Stylesheet by Yifeng Ai, 2019 */
/*#mapid {
  width: 80%;
  height: 800px;
}*/

html {
  width: 98%;
  height: 98%;
}

body {
  width: 100%;
  height: 100%;
}

.leaflet-popup-content p {
   margin: 0.2em 0;
}

/*.range-slider {
  width: 35%;
}

#forward {
  float: right;
}

#reverse {
  float: left;
}*/
#map {
  height: 100%;
  width: 70%;
```

```
  /*display: inline-block;
  */float:right
}


#menu {
  position: absolute;
  width: 29%;
  height: 98%;
  display: inline-block;
  vertical-align: top;
  text-align: left;
  line-height:0px
}

#det_window {
  position: absolute;
  width: 29%;
  height: 98%;
  display: inline-block;
  vertical-align: top;
  text-align: left;
  line-height:0px
}

#price_window {
  position: absolute;
  width: 29%;
  height: 98%;
  display: inline-block;
  vertical-align: top;
  text-align: left;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 14px;
  line-height: 0px
}
```

```
#area_window {
  position: absolute;
  width: 29%;
  height: 98%;
  display: inline-block;
  vertical-align: top;
  text-align: left;
  line-height:0px
}


#ppa_window {
  position: absolute;
  height: 98%;
  width: 70%;
  right:0px;
  display: inline-block;
}

#campus_window {
  position:  absolute;
  width: 29%;
  height: 98%;
  display: inline-block;
  vertical-align: top;
  text-align: left;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 14px;
  line-height:16px
}
#res_window {
  /*position: absolute;*/
  width: 29%;
  height: 98%;
  display: inline-block;
  vertical-align: top;
  text-align: left;
```

```
  font-family: Arial, Helvetica, sans-serif;
  font-size: 14px;
  line-height:5px;
}
#bus_window {
  position: absolute;
  width: 29%;
  height: 98%;
  display: inline-block;
  vertical-align: top;
  text-align: left;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 14px;
  line-height:2px
}

.updatable {
  /*position:absolute;*/
  top:325px;
  width: 90%;
  text-align: left !important;
  line-height: 5px;
}

#updatable-basic {
  width: 90%;
  margin-top: 50%;
  margin:4px;
  line-height: 18px
}

#tb_flp {
  position:absolute;
  width:70%;
  margin-top: 0%;
  font-family: "Trebuchet MS", Arial, Helvetica, sans-serif;
  border-collapse: collapse;
```

```
  width: 100%;
  overflow-wrap: normal;
  line-height: 16px;
}

#tb_flp td, #tb_flp th {
  border: 1px solid #ddd;
  padding:2px;
  /*word-wrap: break-word*/
}


#tb_flp tr:nth-child(even){background-color: #f2f2f2;}

#tb_flp tr:hover {background-color: #ddd;}

#tb_flp th {
 padding-top: 3px;
 padding-left: 3px;
 padding-bottom: 3px;
 text-align: left;
 background-color: #4CAF50;
 color: white;
}

#tb_extra {
  color: grey
}


/* Below shows the compare operator formatting*/

#compare {
  width: 28%;
  height: 100%;
  padding: 4px;
  display: inline-block;
```

```
  vertical-align: top;
  top: 10px;
  text-align: left;
  line-height: 3px;
}

#title_menu {
  line-height:30px;
  position: absolute;
  margin-top: 1.5%;
  margin-left: 10px;
  text-align: center;
  font-size: 24px;
  font-family: Arial, Helvetica, sans-serif
}
.back_to_menu {
  text-align: center;
  font-size:16px;
  background-color: #cc3333;
  display: inline-block;
  padding: 15px 11px;
}
/* Explore*/
#title_explore {
  line-height:5px;
  position: absolute;
  margin-top: 20%;
  margin-left: 10px;
  text-align: left;
  font-size: 16px;
}
#detail_explore {
  position: absolute;
  margin-top: 27.5%;
  margin-left:10px;
  text-align: center;
  font-size: 16px;
```

```
  background-color: #4CAF50;
  display: inline-block;
  padding: 15px 29px
}
/* Compare*/
#title_compare {
  line-height: 5px;
  position: absolute;
  margin-top: 50%;
  margin-left: 10px;
  text-align: left;
  font-size: 16px;
}
#price_compare {
  position: absolute;
  margin-top: 57.5%;
  margin-left: 10px;
  text-align: center;
  font-size: 16px;
  background-color: #4CAF50;
  display: inline-block;
  padding: 15px 30px;
  line-height: 18px
}
#radio_group {
  width:98%;
}

#area_compare {
  position: absolute;
  margin-top: 57.5%;
  margin-left: 200px;
  text-align: center;
  font-size: 16px;
  background-color: #999999;
  display: inline-block;
  padding: 4.3px 15px
```

```
}
#ppa_compare {
  position: absolute;
  margin-top: 75%;
  margin-left: 10px;
  text-align: center;
  font-size: 16px;
  background-color: #999999;
  display: inline-block;
  padding: 4.3px 16px
}

/*proximities*/
#title_proximity {
  line-height: 5px;
  position: absolute;
  margin-top: 97.5%;
  margin-left: 10px;
  text-align: left;
  font-size: 16px;
}
#campus_proximity {
  position: absolute;
  margin-top: 105%;
  margin-left: 10px;
  text-align: center;
  font-size: 16px;
  background-color: #4CAF50;
  display: inline-block;
  padding: 15px 42px
}
#res_proximity {
  position: absolute;
  margin-top: 105%;
  margin-left: 200px;
  text-align: center;
  font-size: 16px;
```

```
  background-color: #4CAF50;
  display: inline-block;
  padding: 15px 28px
}
#bus_proximity {
  position: absolute;
  margin-top: 122.5%;
  margin-left:10px;
  text-align: center;
  font-size: 16px;
  background-color: #4CAF50;
  display: inline-block;
  padding: 15px 36.5px
}

.yr_text {
  position: relative !important;
  left:36px;
  bottom:50px;
  text-align:justify;
  color: white !important;
  font-size:50px;
  text-shadow: 4px 4px 15px red;
}

.notif {
  margin-top:300px;
}

.range-slide-new {
  width: 200px;
}

.legend-circle {
  padding-top: 200px;
}
```

```
#TT {
  width: 100%;
  line-height: 5px;
  margin-top: -10px;
  margin-bottom: 10px;
}

#bus_side_window {
  width: 100%;
  line-height: 4px !important;
  font-size: 12px;
}

.legend {
  width: 162px;
   line-height: 27px;
  color: #555;
  background-color: #eeeeee;
  border: solid #111111 1px;
  margin-top: 10px;
  margin-bottom:10px
}
.legend2 {
  width: 132px;
   line-height: 27px;
  color: #555;
  background-color: #eeeeee;
  border: solid #111111 1px;
  margin-top: 10px;
  margin-bottom:10px
}
.legend3 {
  width: 155px;
   line-height: 27px;
  color: #555;
  background-color: #eeeeee;
  border: solid #111111 1px;
```

```
  margin-top: 10px;
  margin-bottom:10px
}
.legend i {
  width: 18px;
  height: 18px;
  float: left;

  margin-top: 4px;
  margin-bottom: 4px;
  margin-left: 8px;
  margin-right: 8px;
  opacity: 0.95;
}

#legend_title {
  margin-left: 5px;
  margin-bottom: -2px;
  font-size: 16px;
  float: center;
}
```

7.11. Appendix 11: index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title></title>

    <link rel="stylesheet" href="css/style.css">
    <link rel="stylesheet" href="lib/leaflet/leaflet.css">
    <!--[if IE<9]>
      <link rel="stylesheet" href="css/style.ie.css">
```

```
    <![endif]-->
  </head>

  <body>

    <script type="text/javascript" src="lib/jquery-3.3.1.js"></script>
    <script type="text/javascript" src="lib/leaflet/leaflet-src.js"></script>
    <script src='https://unpkg.com/simple-statistics@7.0.2/dist/simple-statistics.min.js'>
    </script>

    <div id='menu'></div>

    <div id='price_window'>
      <button class='back_to_menu'><img class='arrow' src='data/icon/left_arrow2.png'>  &nbspBack
To Menu</button>
      <div id='radio_group'>
        <p><label><input type="radio" name="bedrooms" value="Studio">Studio</label>
                      <label><input
type="radio" name="bedrooms" value="1 Bedroom(s)">1 Bedroom</label>
             <label><input type="radio" name="bedrooms" value="2 Bedroom(s)">2
Bedrooms</label></p>
        <p><label><input type="radio" name="bedrooms" value="3 Bedroom(s)">3 Bedrooms</label>
             <label><input type="radio" name="bedrooms" value="4 Bedroom(s)">4
Bedrooms</label>
            <label><input type="radio" name="bedrooms" value="5 Bedroom(s)">5 or more
Bedrooms</label></p>
      </div>

      <p style='line-height:18px;margin-top:3px;font-size:14px;font-weight:bold'>
        Select the floor plan you are interested in, the map will update with all apartments having that price, then you
can click on apartment polygons to retrieve details.
      </p>

      <div id='TT'>
        <label>
          <input type='checkbox' class='showTT' id='unitTT' name='show_TT' value='number_of_units'> Display
number of units
```

```
        </label>
        <div>_____</div>
        <!--<label>
          <input type='checkbox' class='showTT' id='yrTT' name='show_TT' value='number_of_units'> Display year
built
        </label>-->
      </div>

      <div class='updatable'></div>
    </div>

    <div id='campus_window'>
      <button class='back_to_menu'><img class='arrow' src='data/icon/left_arrow2.png'>  &nbspBack
To Menu</button>
      <div id='radio_group_4' style='padding-left:5px; font-family:arial; font-size: 15px; line-height: 20px'>
        <p>     &nbspThe "To campus" functionality enables you to get an overview of
the walking and bicycling distance from
          campus.You need to select <b>both</b> the method of traveling and a campus landmark to explore travel
time.
        </p>
        <p>Select method of traveling: </p>
        <p><b>
          <label><input type="radio" name="travel_mode" value='walk'> Walking</label>
                           
          <label><input type="radio" name="travel_mode" value="bike"> Bicycling</label>
        </b></p>
      </div>

      <div id='radio_group_3' style='padding-left:5px'>
        <p>_____</p>
        <p style='font-size: 17px'>Show walk time to campus landmarks: </p><p><label><input type="radio"
name="campus" value='gordon'> To Gordon</label>

                   &n
bsp
          <label><input type="radio" name="campus" value="bascom"> To Bascom Hall</label></p>
        <p><label><input type="radio" name="campus" value="unionS"> To Union South</label>
```

```
                      
          <label><input type="radio" name="campus" value="nat"> To Natatorium</label></p>
        <p>_____</p>
      </div>

    </div>

    <!--Restaurant Proximity Window-->
    <div id='res_window'>
      <button class='back_to_menu'><img class='arrow' src='data/icon/left_arrow2.png'>  &nbspBack
To Menu</button>
      <div id='radio_group_2' style='padding-top: 10px;padding-top:-150px;padding-bottom:-15px;line-
height:20px;font-size:14px'>
          <div>     &nbspThe "discover restaurants" functionality enables you to
discover restaurans within 5/10 minute walking distances, you can use this tool to
            find apartments with good restaurant accessibility — if you love to dine out close to home.</div>
          <div>_____</div>
          <div style='line-height: 25px'>

            <b>Select walk distance to discover the number of restaurants for all apartments</b><br>
            <label><input type="radio" name="ress" value='5_ress'> 5 minutes</label>
                      
            <label><input type="radio" name="ress" value="10_ress"> 10 minutes</label>

          </div>
      </div>

      <div class='updatable'></div>
    </div>

    <div id='bus_window'>
      <button class='back_to_menu'><img class='arrow' src='data/icon/left_arrow2.png'>  &nbspBack
To Menu</button>
      <div id='bus_side_window'></div>
    </div>

    <div id='map'></div>
```

```
    <div id='compareResult'></div>

    <script type="text/javascript" src="js/geojson_MyOwnData.js"></script>

  </body>
</html>
```