CS-GY 6313 / CUSP-GX 6006: Data Visualization - Spring '24

Homework #2

This homework is due three weeks after the release of this assignment: April 3rd, 2024.

> *This homework consists of 2 required segments, totaling up to 15 points. Points are more likely to be awarded if code implementations come with comments explaining your process, even if the answer is incorrect. Submit your code as a `.zip` file together with your code and any other submittable items such as figures or writeups. You are permitted to use external Python libraries for your work.*
>
> *Adhere to NYU's Academic Integrity rules. Your answers and implementation must be your own. If you cooperated with another student from the course, please list either their name or NetID either as a comment in the code implementation or in Markdown. If you relied on external resources, websites, publicly-available code, etc. to answer any part of this assignment, you are expected to add an attribution to the original source material (ex. a comment, or a URL). **Using ChatGPT to generate code is explicitly not allowed and will result in an automatic 0 points for this assignment.***

Geographical Data Visualization (15 points)

In this homework, you will visualize bike trip data as a heat map with `pandas` and `geopandas`. Feel free to make ready use of online resources to familiarize yourself either Python packages. You will be working with 3 datasets:

- Bike trip data for July 2020: `datasets\202007-divvy-tripdata.csv`
- Bike station locations: `datasets\station-locations.csv`
- Community areas from the city of Chicago: `datasets\chicago-community-areas.geojson`

Data Pre-Processing (3/15 points)

Some of the data in these datasets are not in the format we want and may be missing important data points. We're going to change that by pre-processing each dataset to ensure that they're in the proper format. Complete the tasks below:

1. **Bike trip Pre-Processing (1 point):** Within the bike trip data that we loaded (`trips_df`), get rid of missing (`NaN`) start and end station IDs, and convert those columns to integer columns. Make sure the modified dataframe is referenced as `trips_pr_df`.
2. **Community Areas Pre-processing (1 point):** Within the geojson data for the Chicago community areas (`community_df`), rename the column `area_numbe` to `area_number`, and convert that column to an integer column. Make sure to reference the modified geojson data as `community_pr_df`.
3. **Stations Pre-processing (1 point):** Within the bike station location data (`stations_df`), convert it to a `GeoDataFrame` and set its geometry to the point specified by the longitude and latitude pair. Make sure to reference the modified data as `stations_pr_df`.

Hint: use the `geopandas.points_from_xy()` method to construct longitudinal and latitudinal points and specify the `geometry` arg when constructing the `GeoDataFrame`.

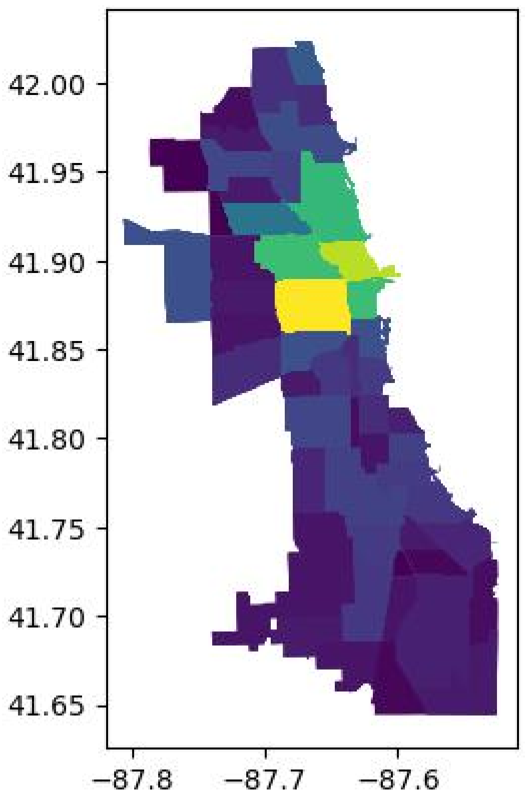
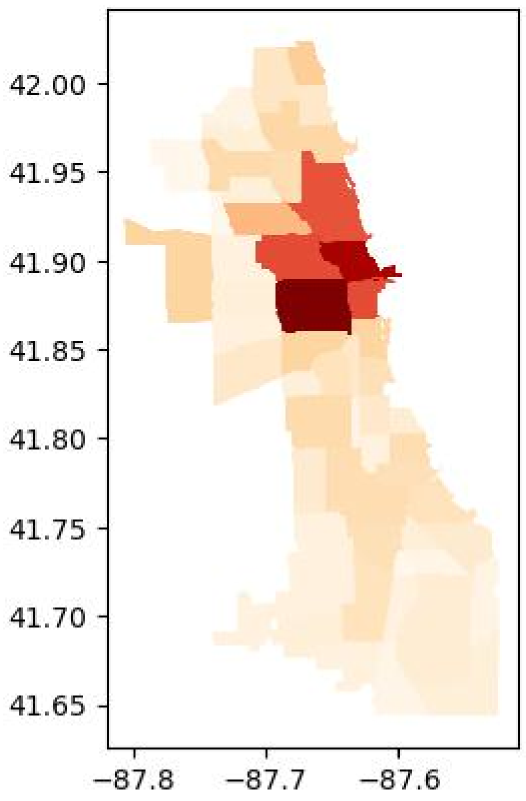
Grading Metric: We'll run your code to generate `trips_pr_df`, `community_pr_df`, and `stations_pr_df`. Points are assigned based on how accurately the implementation follows the instructions. We'll use a code implementation of our own to test whether the changes have been appropriately made to each dataframe.

Geographical Visualization (12/15 points)

We will now analyze the relationship between community area, trips, and station location. Some of these tasks involve merging `DataFrames` and `GeoDataFrames` in a manner similar to SQL table joins. We will also visualize one of these relationships as a spatial heat map.

1. **Spatial Join (2 points):** Given points from station locations, we want to find out which community areas those points are in. This can be accomplished using a sjoin in `geopandas`. After joining the two datasets, you should be able to find the area_number for each `station_id`. Save your joined results as `station_community_df`. ***Hint:*** *If you see an error about CRS of frames not matching, you can set the `crs` parameter of the `GeoDataFrame` to the same as the community areas.*
2. **Add Community Areas to Trips (4 points):** Use the updated dataframe from the previous part with the bike trip dataset to add columns specifying the start and end community area numbers (`start_ca_num` and `end_ca_num`) for each trip. Remove any entries in your final results that have `NaN` values for either `start_ca_num` or `end_ca_num`. Save your results in `trips_community_df`. ***Hint:*** *Use the `start_station_name` and `end_station_name` and the joined dataframe from the previous part to set these. You could try to use two normal `pandas` merges (one for start and one for end) as opposed to the `sjoin` method.*
3. **Explaining the Joins (2 points):** In a short (no more than a paragraph) description, please briefly answer the following inquiries. You can write either in Markdown or in code comments in the space provided in the notebook file.
    1. For each join conducted in steps 1 and 2, what was your rationale for using these particular join types?
    2. Did your final `trips_community_df` end up a different size from the original `trips_pr_df` dataframe? If so, what do you think caused this difference in size?
4. **Visualize Station Distribution (4 points):** We want to understand which community areas have bike stations. Using `geopandas`, generate a plot of the number of stations per community area. This can be accomplished by aggregating the stations by community area. Then use the `plot()` command to generate a chloropleth map. Examples of possible chloropleth maps are provided below. You are allowed to define a colormap for your chloropleth map via the `cmap` parameter. ***Hint(s):*** *If you end up using `groupby`, you should recreate the `GeoDataFrame` before plotting. If your plot appears less like a geographic map and more like a collection of dots, make sure check that the geometry of your `GeoDataFrame` uses `MultiPolygon` rather than `Point`; this will depend largely on how you joined the `stations_community_df`.*

Default Choropleth	Orange-Red Choropleth

Default Choropleth	Orange-Red Choropleth
	

Grading Metric: We'll run the code to check if all code blocks are properly executed - this means joins are executed correctly regardless of join type and that the visualization produces a choropleth map. Your rationale for the chosen join types must provide insight beyond random choice. The resulting choropleth visualization must clearly highlight which communities have the greatest and least number of bike stations.