Lab Report

Title: Lab1
Notice: Dr. Bryan Runck
Author: Greg Kohler
Date: 10/10/2023

Project Repository: *<if applicable weblink to public repository>*
Time Spent: 20

Abstract

This lab involved downloading data from three APIs, and then using two of these datasets, transform the CRS and perform a spatial join. The three APIs used were MN Geospatial Commons, Google Places, and NDAWN. The datasets accessed from these APIs were Transit Centers, Bus Garages, and Weather Stations. The methods used to download the data, transform the CRS, and perform the spatial data are shown in flow charts. The results are showcased in maps showing each downloaded dataset, as well as a map of the spatially joined dataset. The differences and similarities in the three APIs are showcased in a table. The results are verified by showing the successful code, the downloaded datasets, and the output table that showcases the spatially joined field. The final section of this report discusses the obstacles and lessons learned as I completed this lab.

Problem Statement

APIs are a powerful tool to access specific data in a streamlined way. For the first section of this lab, the purpose is to compare and contrast three different APIs to obtain spatial data. For the second section of this lab, the purpose is to take two datasets, change their coordinate system, spatially join the two, print out the table with the joined attributes, and save the joined dataset to a geodatabase. This lab will help to establish an understanding of how to extract, transform, and load data.

Table 1 - Resources Used

#	Requirement	Defined As	(Spatial) Data	Attribute Data	Dataset	Preparation
1	MN Geospatial Commons API	URL to download/access shapefile from the commons.	Shapefile, Point Data	Information about Park and Rides	Mn GeoSpatial Commons	Had to find shapefile url in API.
2	Google Places API	URL to download/access JSON information from the API.	JSON with coordinates	Name of Bus Garages	Google Places API	Had to create query to search API.

3	NDAWN API	URL to download/access CSV informaton from the API.	XY coordinates of stations.	Average Annual Rainfall	NDAWN API	Had to adjust URL to get information from multiple stations.
4	ArcGIS Pro Notebook	Python Notebooks to write code for obtaining and manipulating data from the API.	N/A	N/A	N/A	N/A

Input Data

The data for this lab was obtained from three sources. The first source was the MN Geospatial Commons, through the API link I was able to download a shapefile of transit centers and park and rides. The second source was the Google Places API. From this API, I was able to query this API and get a JSON of Bus Garages in the Twin Cities. This data had to be converted into a shapefile. The third data source was the NDAWN API. The data from this site was a CSV containing information on average rainfall for a few different weather stations in North Dakota. This data also had to be converted into a shapefile.

Table 2 - Data Sources

#	Title	Purpose in Analysis	Link to Source
1	Transit Centers & Park & Rides	Point data of transit centers in the Twin Cities from the MN Geospatial Commons. Data was used to perform the spatial join.	Mn GeoSpatial Commons
2	Bus Garages in Twin Cities	JSON of locations of Bus Garages in the Twin Cities. Locations were determined by a query to the API. Needed to be converted to shapefile.	'Google Places API'
3	Average Rainfall of NDAWN Stations	CSV of average annual rainfall in North Dakota at various weather stations. Needed to be converted to shapefile.	NDAWN API

Methods

Accessing and downloading the data from each API had different approaches. Accessing the data from the Minnesota Geospatial Commons was probably the simplest approach. The data was already stored in a shapefile, so it just required accessing the zip file, writing it to my local folder, and then extracting the shapefiles using the zipfile module.

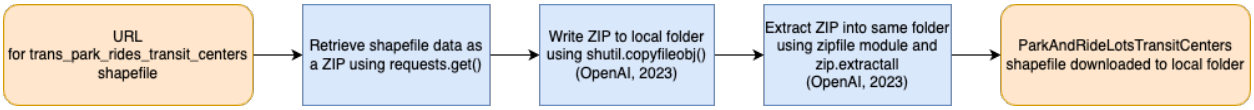

Figure 1 - Downloading Data from MN Geospatial API

For the Google Places API, the process was similar but had a few more steps. To obtain data from the API URL, I needed to create a search query and have a key. With this, I was able to pull the data from the API and format it as a JSON. Once it was formatted, it was a matter of accessing the name of the bus garages and their coordinates. With this information in a list, I was able to convert it to a GeoDataFrame and write it to a shapefile.

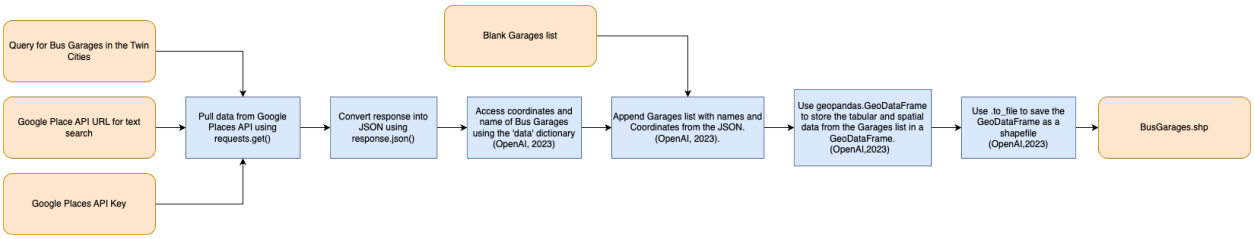

Figure 2 - Downloading Data from Google Places API

The final API to download data from was NDAWN. This started with a URL from NDAWN with all the stations I was interested in. The next step was to get the data from the API URL using the requests module. This data was written to a CSV file in my local folder. Next, I opened up this CSV with pandas and renamed the columns to make more sense. After saving these changes, I created point geometry using the XY coordinates of the weather stations. With this geometry defined, I opened up a GeoDataFrame with the names of the bus garages and their coordinates. I filtered this GeoDataFrame to only have average rainfall results from 2020 and then saved this refined version to a shapefile.

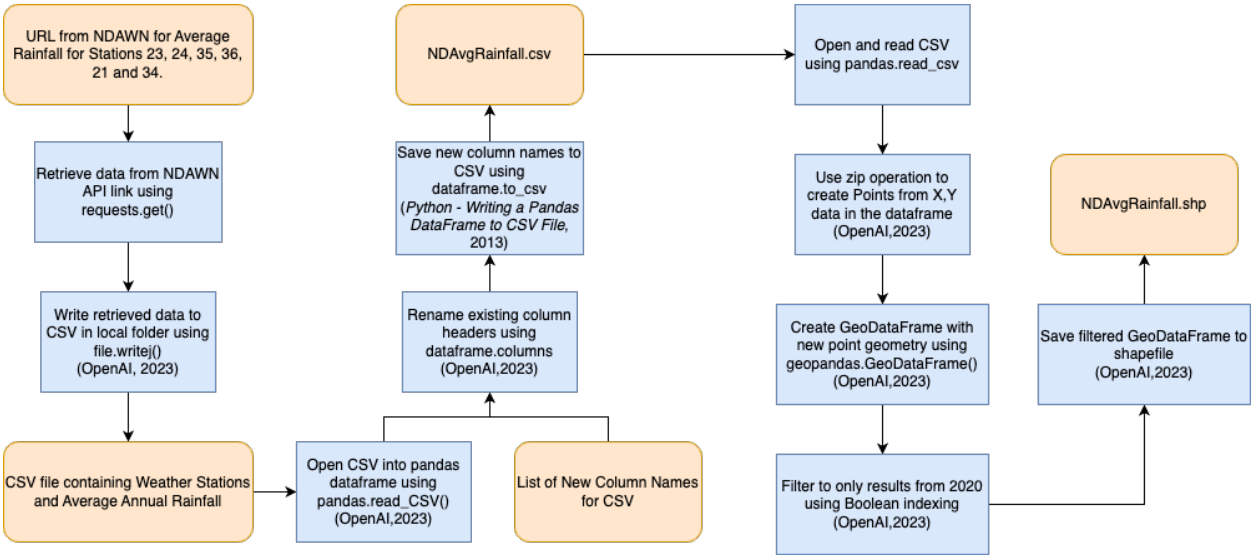

Figure 3 - Downloading Data from NDAWN API

After downloading the data from all three 3 APIS, the next step was to take two of these datasets, change them to the same coordinate system, spatially join them, and then print out the joined table. I chose to use the Park and Ride Center data from MN Geospatial Commons and the Bus Garage locations from the Google Places API. To change both datasets to the same

coordinates, the first step was to open up the downloaded shapefiles into a GeoDataFrame. Then using GeoDataFrame.to_crs() I converted both GeoDataFrames to the same coordinate reference system. For the dataset with Park and Rides and Transit Centers, I chose to clean up the data. This required creating a list of columns I did not want from the dataset and dropping them. Once this was completed, I wrote the GeoDataFrames for both datasets to new shapefiles.

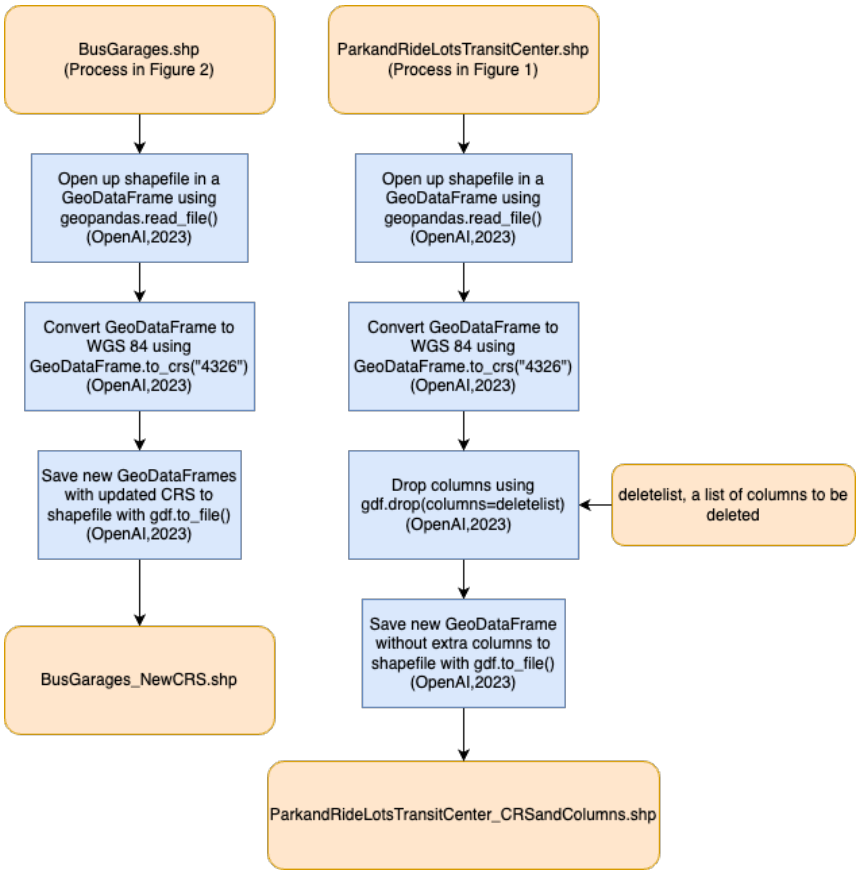

Figure 4 - Changing CRS

After changing the CRS, the next process was to perform a spatial join of the two datasets using arcpy. This tool required the input feature, which was Park and Rides and Transit Centers, and the join feature, which was Bus Garages in the Twin Cities. For the tool parameters, I chose to do a spatial join for Park and Rides and Transit Centers within 5 kilometers of Bus Garages. I chose to do a one-to-many join, as multiple bus garages can be near a transit center. I also chose to only keep transit centers that were within 5 kilometers of the Bus Garages. The joined dataset was saved to the geodatabase as a feature class.

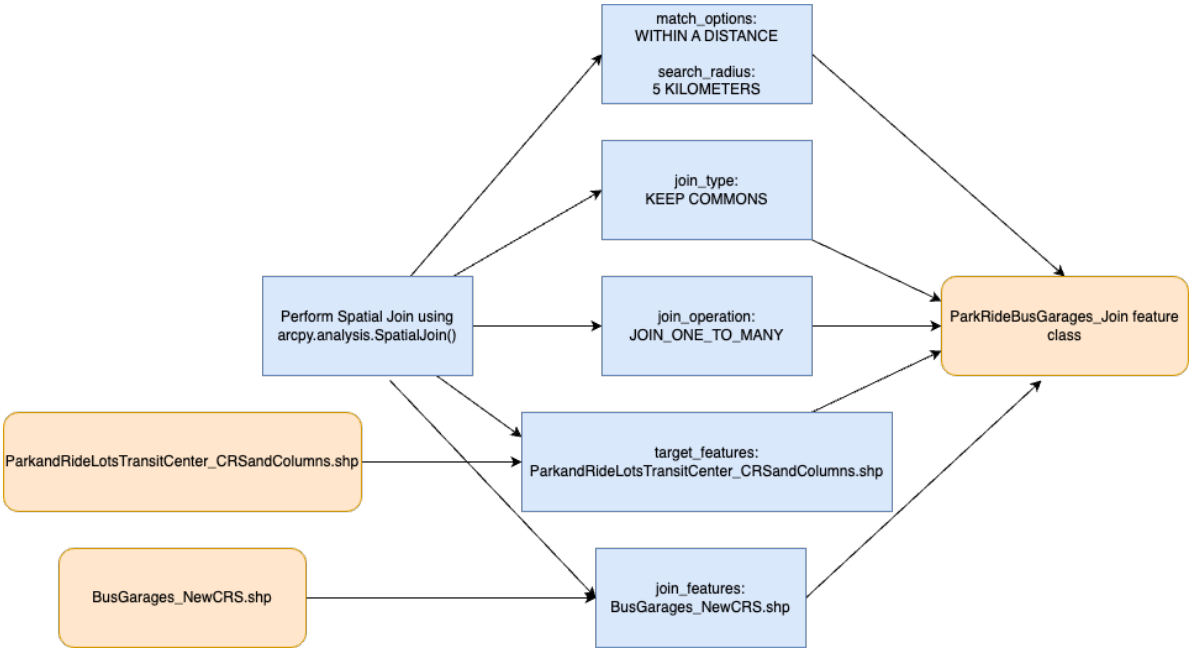

Figure 5 - Spatial Join

The final step for this lab was to print out the joined table. I started an editing session in the workspace. Then, I renamed the joined field to BusGaragewithin5km using the arcpy AlterField tool. Next, with the joined feature class, I extracted the data and fields, put them in a data frame, and printed the table out to the screen.

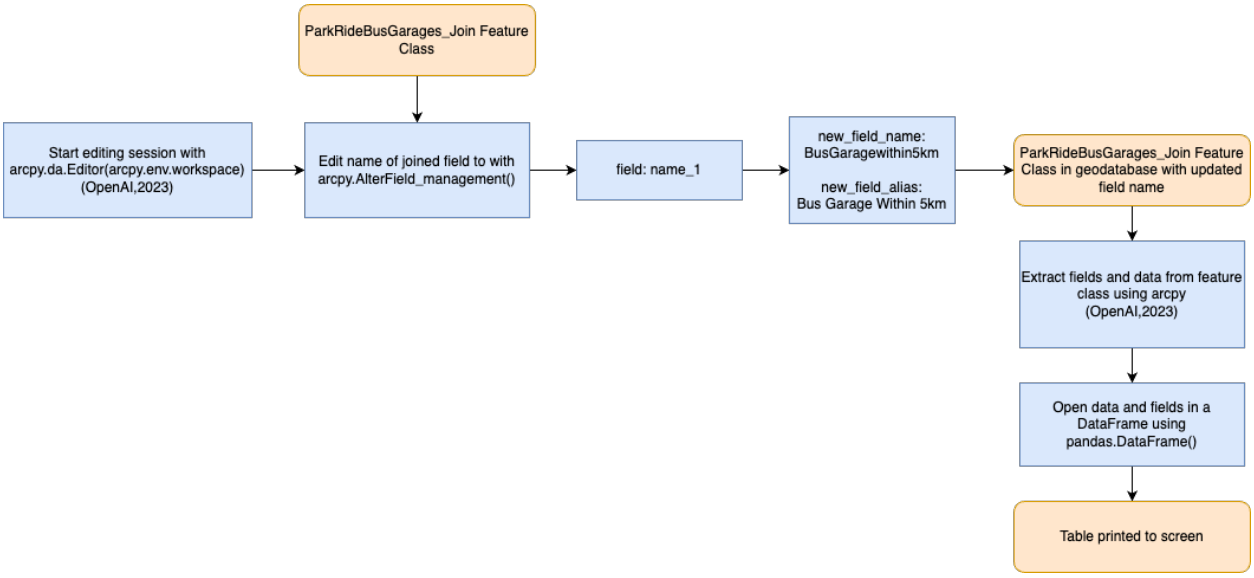

Figure 6 - Print Out Table to Screen

Results

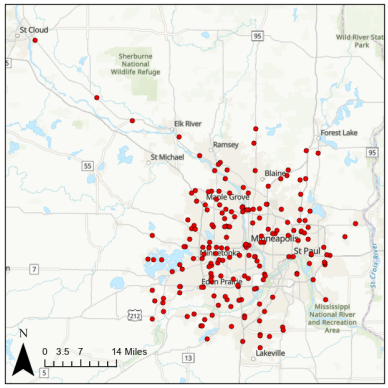

Figure 7 - Park and Rides and Transit Centers from MN Geospatial Commons

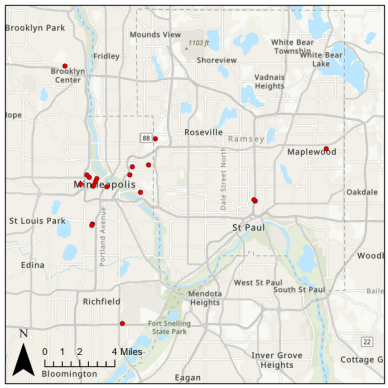

Figure 8 - Bus Garages in Twin Cities from Google Places API

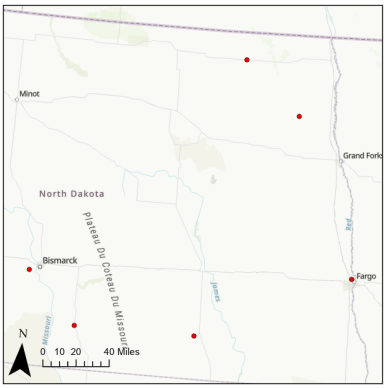

Figure 9 - Weather Stations in North Dakota from NDAWN API

The first section of this lab generated three different data sets from three different APIs. While the process of obtaining spatial data from the three APIs was different, they all produced point layers, see Figures 7, 8, and 9. Each layer had a different method of downloading the data and converting it into a displayable file format. See Table 3 for a breakdown of the similarities and differences in obtaining data from the different APIs.

Table 3 - Comparing the Three APIs

	MN Geospatial Commons API	**Google Places API**	**NDAWN API**
Output Type(s)	Shapefile	JSON	CSV
Uses requests.get()	Yes	Yes	Yes
API Key	No	Yes	No
Location	Minnesota	Worldwide	North Dakota
Uses CKAN	Yes	No	No
Needed to be Converted to Geometry	No	Yes	Yes
Geometry	Point	Point	Point
Data Points Represents	Transit Centers	Bus Garages in Query	Weather Stations
Had to Add to URL	No	Yes, with search query	Yes, to add stations

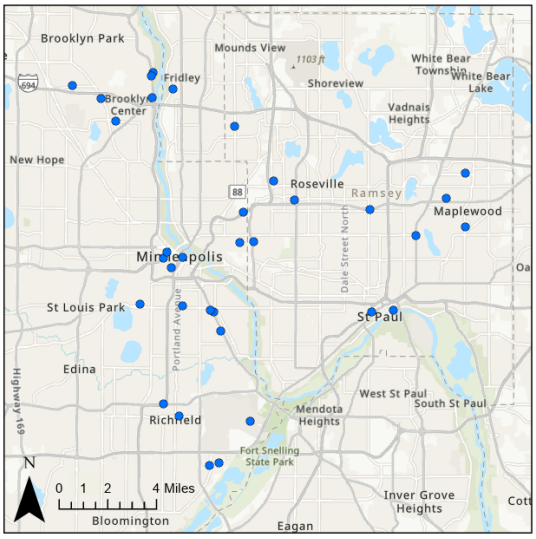

Figure 10 - Spatial Join - Transit Centers within 5 km of Bus Garage

For the second part of the lab, a new layer was created by performing a spatial join. This spatial join displayed Park and Rides and Transit Centers within 5 km of a Bus Garage, see Figure 10. Both of the layers that were joined had their coordinate system converted to the same coordinate reference system. See Figure 11 for the table printed on the notebook console. This table shows the joined field, BusGaragewithin5km.

Name	FcltyType	Status	Address	City	Provider	YearOpen	FcltyOwnr	SharedUse	x_coord	y_coord	BusGaragewithin5km
30th Ave Station Park & Ride	PR	Open	8101 Winstead Way	Bloomington	Metro Transit Rail	2004.0	Metropolitan Council	N	-93.231054	44.855682	Metro Transit South Garage
38th Street Station Transit Center	TC	Open	2902 38th St E	Minneapolis	Metro Transit	2004.0	Metropolitan Council	N	-93.229644	44.934772	Greyhound: Bus Stop
65th Ave & Brooklyn Blvd	PR	Open	6503 Brooklyn Blvd	Brooklyn Center	Metro Transit	1995.0	Metropolitan Council	N	-93.331100	45.073067	Metro Transit Martin J Ruter Garage
Hwy 252 & 73rd Ave N	PR	Inactive	7317 West River Rd	Brooklyn Park	Metro Transit	1991.0	Metropolitan Council	N	-93.287365	45.088404	Metro Transit Martin J Ruter Garage
Aldrich Arena	PR	Closed	1850 White Bear Ave	Maplewood		2002.0			-93.024402	44.996888	ISD 622 Transportation
...	...	...	...	...	...	...	...	...	...	...	...
Uptown Transit Station	TC	Open	2855 Hennepin Ave S	Minneapolis	Metro Transit	2001.0	Metropolitan Council	N	-93.298064	44.950252	Fourth Street North Transit Station
Uptown Transit Station	TC	Open	2855 Hennepin Ave S	Minneapolis	Metro Transit	2001.0	Metropolitan Council	N	-93.298064	44.950252	Marquette Av S At 5 St S Sw
Uptown Transit Station	TC	Open	2855 Hennepin Ave S	Minneapolis	Metro Transit	2001.0	Metropolitan Council	N	-93.298064	44.950252	Ramp A/7th St Transit Center

Figure 11 - Table Print Out for Spatial Join

Results Verification

For the first section of this lab, we can verify the results by seeing that data from each API was downloaded, converted to a shapefile, and displayed on a map (Refer back to Figures 7,

8 and 9). We know the process was successful because each map shows point data that corresponds with the data set.

Convert to Same Coordinate Reference System

```
# Opens Up Both Datasets
garagesgdf = geopandas.read_file(r'C:\Users\gregkohler1\Documents\GIS5571_Files\Lab1\Lab1Data\BusGarages.shp')
parkridegdf = geopandas.read_file(r'C:\Users\gregkohler1\Documents\GIS5571_Files\Lab1\Lab1Data\ParkAndRideLotsTransitCenters.shp')
#Converts Both Datasets to Same CRS
garagesgdf = garagesgdf.to_crs("4326")
parkridegdf = parkridegdf.to_crs("4326")
#Saves Both Datasets to New Name with New CRS
garagesgdf.to_file(r'C:\Users\gregkohler1\Documents\GIS5571_Files\Lab1\Lab1Data\BusGarages_NewCRS.shp')
```

Figure 12 - Code to Convert CRS

Join Bus Garages and Park and Rides and saves to Lab1 Geodatabase

```
arcpy.analysis.SpatialJoin(
    target_features=r'C:\Users\gregkohler1\Documents\GIS5571_Files\Lab1\Lab1Data\ParkAndRideLotsTransitCenters_CRSandColumns.shp',
    join_features=r'C:\Users\gregkohler1\Documents\GIS5571_Files\Lab1\Lab1Data\BusGarages_NewCRS.shp',
    out_feature_class=r"C:\Users\gregkohler1\Documents\GIS5571_Files\Lab1\Lab1.gdb\ParkRideBusGarages_Join",
    join_operation="JOIN_ONE_TO_MANY",
    join_type="KEEP_COMMON",
    match_option="WITHIN_A_DISTANCE",
    search_radius="5 Kilometers",
    distance_field_name=""
)
```

Messages

Start Time: Sunday, September 24, 2023 2:17:26 PM

Succeeded at Sunday, September 24, 2023 2:17:27 PM (Elapsed Time: 1.71 seconds)

Figure 13 - Code to Perform Spatial Join

To verify the results of the second section of this lab, all of the code ran without error. It successfully converted the coordinate reference system and performed the spatial join. See Figures 12 and 13 to see the code without errors. We can also see that the spatial join was successful by looking at the joined table in Figure 11. This table shows that a field was added to the Park and Rides and Transit Centers dataset. This new field shows what bus garages were within 5 kilometers of the transit centers. This field shows the successful join relationship between the two datasets.

Discussion and Conclusion

Overall, this lab helped me gain a greater understanding of how to access and download data from APIs. It also helped me grasp how to manipulate spatial data after obtaining it from these APIs. Obtaining the data from each API site was fairly simple. While each API had a slightly different way of showcasing the data, I made use of requests.get() to extract the data from each API.

Downloading data from Minnesota Geospatial Commons was the most straightforward. It did teach me how to use requests.get() to extract the data and write it to my local disk. The greatest challenge with this was extracting the shapefiles from the zip file after it had been written to my local drive, but this was achieved by using the zipfile module (Open AI, 2023).

Obtaining data from Google Places API taught me new concepts. The biggest thing I learned was using coordinates (in this case from the JSON) and formatting it into a shapefile. I have used JSON before but never used it to make geometry. However, the process is fairly simple after some exploration. It required appending a list with each search result that contained

the name and coordinates. Once this was setup, the geometry in the list was able to be converted into a shapefile. This API process helped me grasp how geometry can be created.

For NDAWN, I got stuck on actually obtaining the link to the data. The data came in a CSV format, but I was at first unsure of how to get more than one stations' data. However, once I read through the site's URL, I discovered you can manipulate the URL to add more stations. Once I got this, the rest of the process was very similar to the Google Places API. The only difference was using Point(xy) to tuple the coordinates.

Most of the learning for this lab came from downloading the datasets from each API. I did not previously know how to change the CRS of a dataset with Python, but it ended up being a simple command (.to_csv). Performing the actual spatial join was also fairly simple, as it is already a tool used in ArcGIS Pro. My biggest obstacle for the second part of the lab was how I wanted to actually perform the spatial join. There are several methods to perform a spatial join and I wanted to choose one that made the most sense for the datasets I had. I decided that it would make the most sense to perform a join that kept transit centers that were within 5 kilometers of bus garages. This meant the end results only kept certain transit centers and added a field to show what bus garage was near. With this method, the spatial join outputted something informative about transit centers in the Twin Cities.

References

References

OpenAI. (2023). *ChatGPT*. Chat.openai.com; OpenAI. https://chat.openai.com/

python - Writing a pandas DataFrame to CSV file. (2013, June 4). Stack Overflow; Stack Exchange Inc. https://stackoverflow.com/questions/16923281/writing-a-pandas-dataframe-to-csv-file

Self-score

Fill out this rubric for yourself and include it in your lab report. The same rubric will be used to generate a grade in proportion to the points assigned in the syllabus to the assignment.

Category	Description	Points Possible	Score
Structural Elements	All elements of a lab report are included **(2 points each)**: Title, Notice: Dr. Bryan Runck, Author, Project Repository, Date, Abstract, Problem Statement, Input Data w/ tables, Methods w/ Data, Flow Diagrams, Results, Results Verification, Discussion and Conclusion, References in common format, Self-score	28	**28**
Clarity of Content	Each element above is executed at a professional level so that someone can understand the goal, data, methods, results, and their validity and implications in a 5 minute reading at a cursory-level, and in a 30 minute meeting at a deep level **(12 points)**. There is a clear connection from data to results to discussion and conclusion **(12 points)**.	24	**24**
Reproducibility	Results are completely reproducible by someone with basic GIS training. There is no ambiguity in data flow or rationale for data operations. Every step is documented and justified.	28	**28**

Verification	Results are correct in that they have been verified in comparison to some standard. The standard is clearly stated **(10 points)**, the method of comparison is clearly stated **(5 points)**, and the result of verification is clearly stated **(5 points)**.	20	**20**
		100	**100**