Lab Report

Title: Exploring Data Scraping and Mapping Locations
Notice: Dr. Bryan Runck
Author: Lauren Roach
Date: October 9, 2022

Project Repository: https://github.com/L-roach/GIS5571/tree/main/Lab1
Google Drive Link: *N/A*
Time Spent: 20 hours

Abstract

This project explores the different conceptual models of APIs regarding the following sources: MN Geospatial Commons, Google Places, and NDAWN. Additionally, a pipeline is built using Jupyter Notebooks that extracts data from each of these sources. Following importation, the data was joined together, and the coordinate reference systems were made the same.

Problem Statement

Different organizations utilize different ways of structuring their APIs. Understanding how these APIs are constructed is a useful skill for spatial data scientists, and essential for certain data extractions. Also, certain research problems require sourcing data from various places. For example, exploring the accessibility of libraries in the metropolitan area of Minneapolis requires sourcing the locations of libraries and examining accessibility via transportation. This is explored in the following report.

Table 1. Data Needed

#	Requirement	Defined As	(Spatial) Data	Attribute Data	Dataset	Preparation
1	Library locations	Raw input data via data scraping	Point location		Google Places	N/A
2	Transit routes	Transit routes in Minneapolis metro area	Line		MN Geospatial Commons	
3	Weather data	Weather data of multiple variables			NDAWN	

Input Data

The data used in this report is sourced from three different online sources. The first is Google Places. The second is the MN Geospatial Commons. The third is the NDAWN site. The data is of various types: point data via latitude and longitude locations, shapefile data, and weather data.

Table 2. Data Purposes

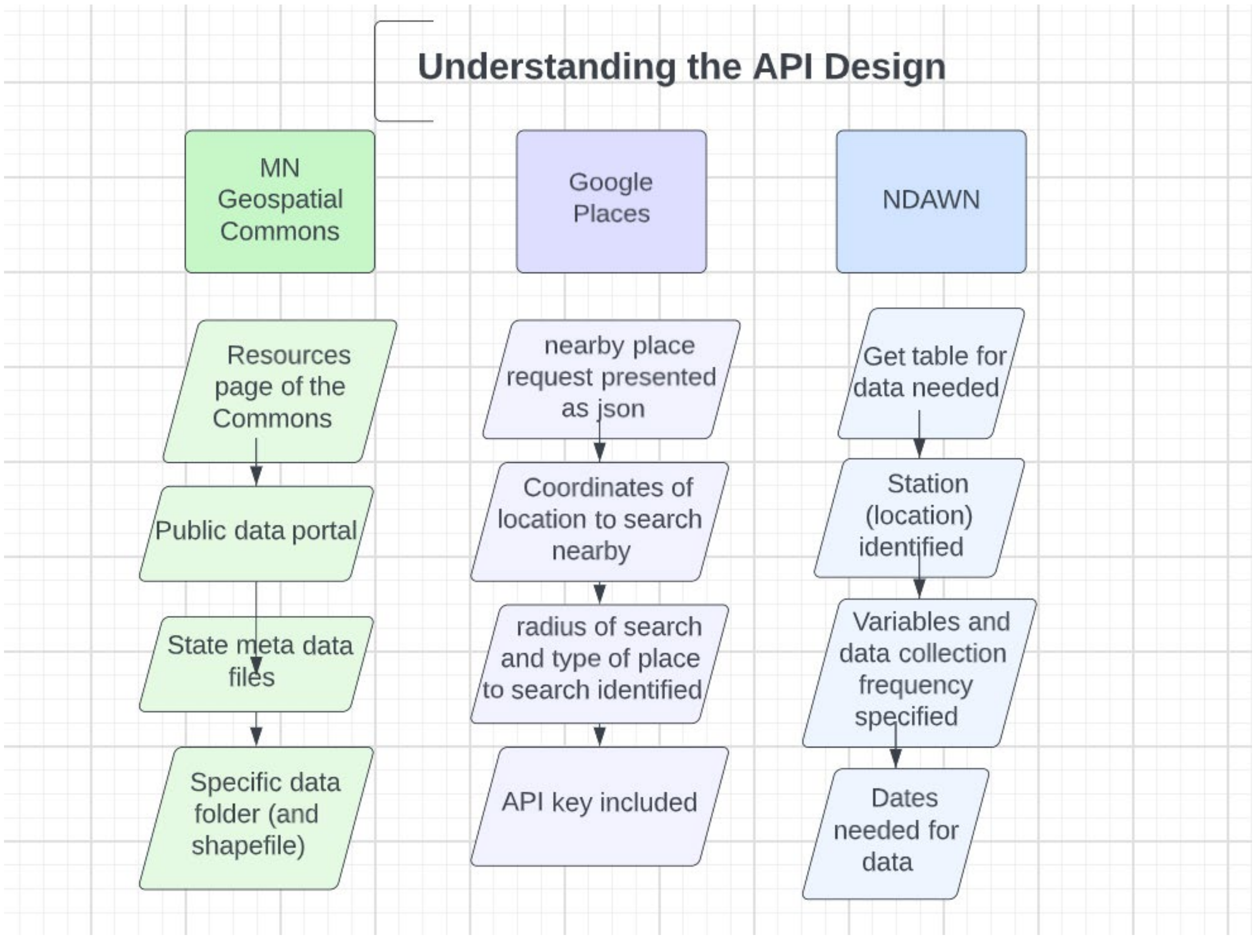
#	Title	Purpose in Analysis	Link to Source

1	Library locations	Raw point location used to show the locations of public libraries in the metropolitan area of Minneapolis	Google Places
2	Transit Routes	Line data showing routes in metro area which are used to show the transportation routes to examine where the library locations are in relation	Mn GeoSpatial Commons
3	NDAWN Weather data	Data used to learn how to extract this type of data from online sources	NDAWN

Methods

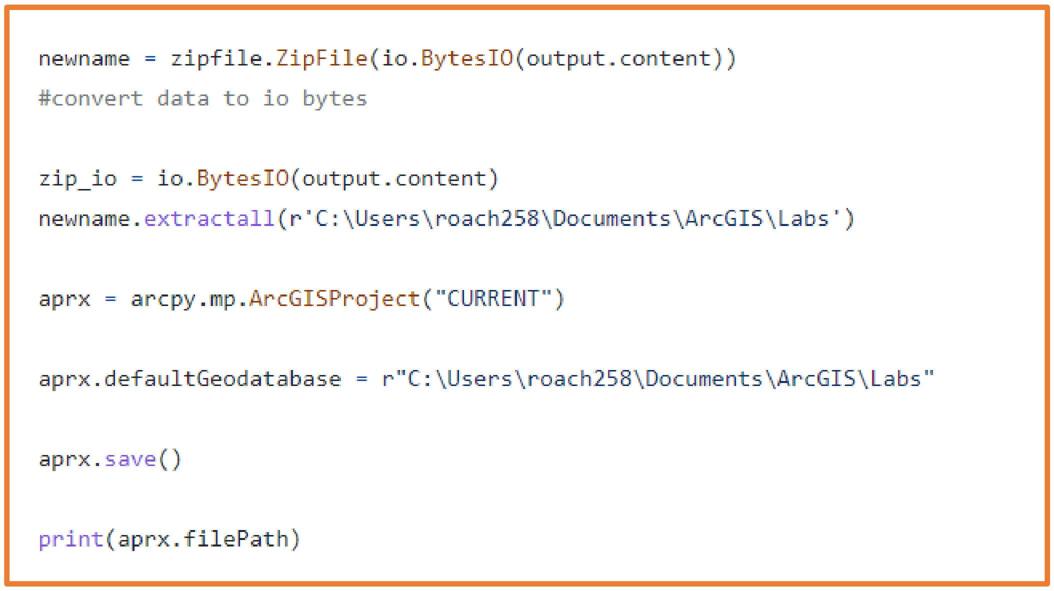
First, the APIs from the three sources were examined. An API key was created in Google Cloud Console and a locations request made using python code in ArcGIS Pro Jupyter Notebooks. The locations requested were “public libraries” within the metro area of Minneapolis, MN. The transit routes shapefile was requested from the MN Geospatial Commons using python code. Weather data for cities in North Dakota was requested in Jupyter Notebooks by requesting the API related to the export CSV function on the NDAWN website.

Figure 1 Understanding APIs


Understanding the API Design
MN Geospatial Commons
Google Places
NDAWN
Resources page of the Commons
Public data portal
State meta data files
Specific data folder (and shapefile)
nearby place request presented as json
Coordinates of location to search nearby
radius of search and type of place to search identified
API key included
Get table for data needed
Station (location) identified
Variables and data collection frequency specified
Dates needed for data

For the coding, the arcpy library was imported and used to complete the necessary functions. The pandas, io, and zipfile libraries were also imported and used. Since the shapefiles on the MN Geospatial Commons are in the form of zipfiles, the zipfile library was used to extract the requested shapefile. To display the shapefile in the map, the map was set to the current context and the data was added from a specified path.

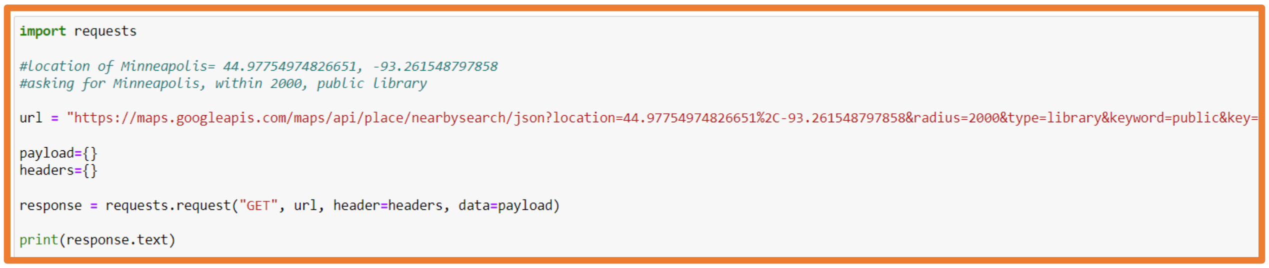
Figure 2 Code to bring in the shapefile from MN Geospatial Commons



```
newname = zipfile.ZipFile(io.BytesIO(output.content))
#convert data to io bytes

zip_io = io.BytesIO(output.content)
newname.extractall(r'C:\Users\roach258\Documents\ArcGIS\Labs')

aprx = arcpy.mp.ArcGISProject("CURRENT")

aprx.defaultGeodatabase = r"C:\Users\roach258\Documents\ArcGIS\Labs"

aprx.save()

print(aprx.filePath)
```

Requesting location data from Google Places involved using an API key and specifying parameters. These parameters included the type of location, the latitude and longitude for the center of the search, the distance of search, and other parameters if wanted. The code was sourced from the Google Cloud Console.

Figure 3 API Request Code from Google Places



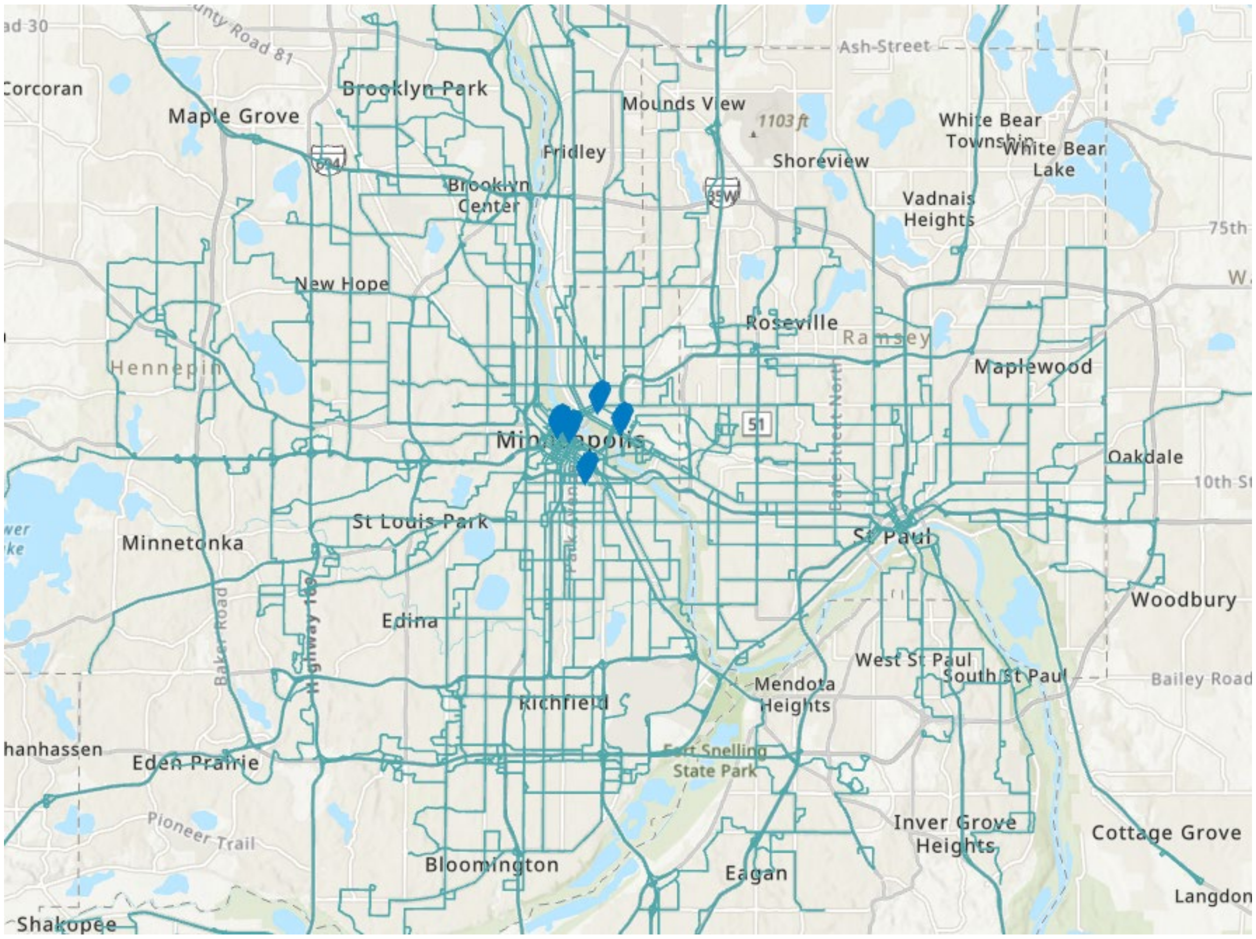
```
import requests

#location of Minneapolis= 44.97754974826651, -93.261548797858
#asking for Minneapolis, within 2000, public library

url = "https://maps.googleapis.com/maps/api/place/nearbysearch/json?location=44.97754974826651%2C-93.261548797858&radius=2000&type=library&keyword=public&key=

payload={}
headers={}

response = requests.request("GET", url, header=headers, data=payload)

print(response.text)
```

Once the latitude and longitude coordinates were returned via the Google Places API request, the coordinates were geoprocessed so that they were turned into a feature layer on the map. During this process, the coordinate system was set to match that of the transit routes. Next, the library location layer and transit routes layer were joined together using a spatial join code in ArcGIS Pro and displayed in a map.

Results

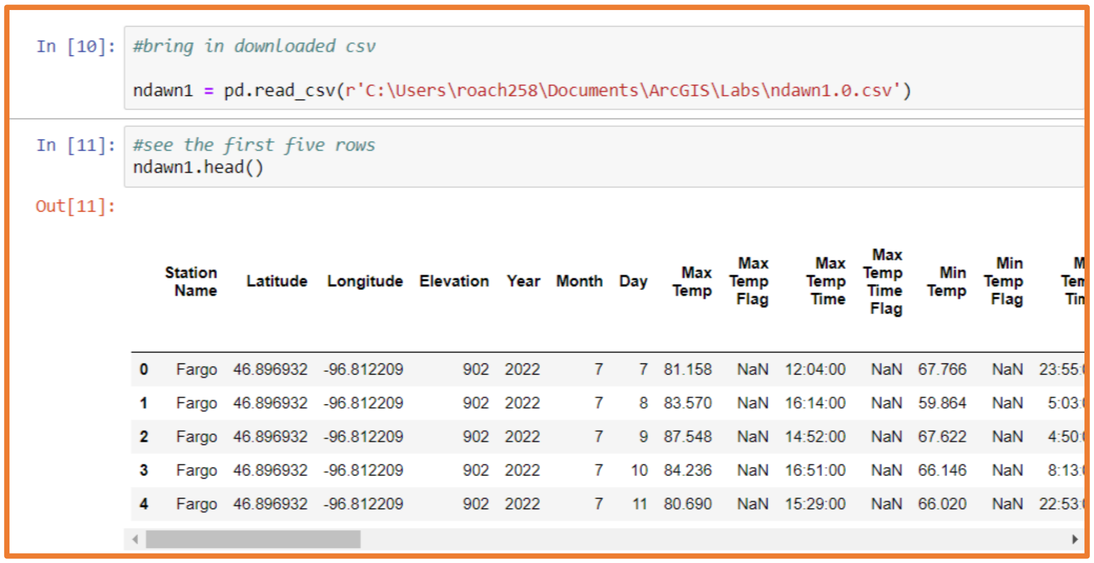
The map displays the locations of the public libraries related to public transit. This displays the joined layer of the features.

Figure 4 Map of Library Locations and Transit Routes



The following image shows the imported NDAWN data.

Figure 5 Code and Data from NDAWN



```
In [10]: #bring in downloaded csv

         ndawn1 = pd.read_csv(r'C:\Users\roach258\Documents\ArcGIS\Labs\ndawn1.0.csv')
```

```
In [11]: #see the first five rows
         ndawn1.head()
```

Out[11]:

	Station Name	Latitude	Longitude	Elevation	Year	Month	Day	Max Temp	Max Temp Flag	Max Temp Time	Max Temp Time Flag	Min Temp	Min Temp Flag	M Tem Tin
0	Fargo	46.896932	-96.812209	902	2022	7	7	81.158	NaN	12:04:00	NaN	67.766	NaN	23:55:
1	Fargo	46.896932	-96.812209	902	2022	7	8	83.570	NaN	16:14:00	NaN	59.864	NaN	5:03:
2	Fargo	46.896932	-96.812209	902	2022	7	9	87.548	NaN	14:52:00	NaN	67.622	NaN	4:50:
3	Fargo	46.896932	-96.812209	902	2022	7	10	84.236	NaN	16:51:00	NaN	66.146	NaN	8:13:
4	Fargo	46.896932	-96.812209	902	2022	7	11	80.690	NaN	15:29:00	NaN	66.020	NaN	22:53:

Results Verification

The results of the joined feature layer was recreated using the geoprocessing tools built into ArcGIS Pro. This produced the same result as the coding. Therefore, the results of the coding analysis was verified by a visual check between the map created using python code in Jupyter Notebooks and the map created using tools in ArcGIS Pro.

Discussion and Conclusion

Even though the Google Places API request was successful and returned the latitude and longitude of library locations, I was unable to find the code necessary to convert the dictionary into a format to plot the latitude and longitude (despite hours of searching the internet). Instead, I was able to copy and paste the coordinates into a csv file, upload that file into Jupyter Notebooks, and then put those coordinates onto the map using the XY Table to Point code. While that code did not initially work for me, once I ran the geoprocessing tool, I was able to copy the Python code and then it worked in Jupyter Notebooks.

Concerning the NDAWN data, I was only able to download a csv with the data and upload that csv to the Jupyter Notebook. I tried to make the various code options work that I had found online after hours of searching, but I could not successfully bring in the data; I kept receiving error messages. Yet, I was able to better understand the structure of the API through these failed attempts. When considering finding data that could be joined to the NDAWN data, I tried to make a Google Places API request for location coordinates of schools in an area of North Dakota, using coordinate locations for a city in North Dakota. However, when I ran the code, it returned locations for Peoria, IL.

The easiest data to bring in using Python code was the shapefile data from MN Geospatial Commons. Even though this required more steps than the Google Places API request, it was the most successful in multiple runs, and the easiest code to create a clean pipeline.

References

N/A

Self-score

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	20
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	24

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	18
		100	90