Lab Report

Title: Lab 1 Report
Notice: Dr. Bryan Runck
Author: Kyle Olson
Date: 10/7/2024

Project Repository: https://github.com/kolson5581/GIS5571/tree/main/Lab%201
Google Drive Link:N/A
Time Spent: 10 hours

Abstract

For Lab 1, I created an ETL pipeline for three datasets using different API methods, from different sources - the NDAWN website, the Minnesota Geospatial Commons, and using Esri's ArcGIS Online REST API. Each dataset is then projected to the same coordinate reference system, and two sets are spatially joined. The end result is a file geodatabase that contains the joined datasets, and the transformed versions of all the input datasets. This lab demonstrates the utility of this approach to loading data into a GIS for analysis, particularly for repeatable tasks.

Problem Statement

In this lab, the requirement was to build an ETL pipeline for three different datasets. In the *extract* phase, I pulled the data using the unique API structures each uses to enable users to access the data. From there, I *transformed* the data, confirming the spatial reference systems were in alignment with each other before performing a spatial join between two of the datasets. Lastly, the data was *loaded* into a new file geodatabase for additional analysis, visualization, or other uses.

Table 1. Project requirements

#	Requirement	Defined As	(Spatial) Data	Attribute Data	Dataset	Preparation
1	Build ETL Pipeline	A program which can take in data from an API and make it usable for a spatial join and export to .gdb	See Rows 2-4			
2	Use NDAWN as data source	NDAWN website, easiest to generate a .csv file	Lat, lon coordinates	Average temperature	NDAWN table of MN Weather Stations	Download as csv via URL, clean unnecessary rows
3	Use a dataset from Minnesota Geospatial Commons	Use CKAN API protocol to access a dataset provided by the State of Minnesota	Polygons stored in shapefile, geometry stored in file	County name, FIPS code, other identifying features	County Boundaries in Minnesota	Download via CKAN API call, reproject to WGS 84

4	Use a dataset using ArcGIS Online REST API	Define a query using Esri's tools and call the API	Polygon features stored in json	Municipality name, city code, other identifying features	County Boundaries in Minnesota	Download via REST API call

Input Data

The data for this lab is drawn from three separate sources: the Minnesota Geospatial Commons, the North Dakota Agricultural Weather Network (NDAWN), and Hennepin County's ArcGIS servers. The datasets were all chosen by me, as no particular problem or outcome was expected beyond demonstrating an understanding of the APIs involved and building the pipeline.

Table 2. Data Sources for olso5581 Lab 1

#	Title	Purpose in Analysis	Link to Source
1	NDAWN weather stations in Minnesota	Dataset accessed via NDAWN's query url structure to download specific data	NDAWN table of MN Weather Stations
2	Hennepin County Municipalities	Dataset accessed using ArcGIS Online REST API	Hennepin County Municipalities
3	County Boundaries in Minnesota	Dataset accessed using MN Geospatial Commons CKAN API	County Boundaries in Minnesota

Methods

API Construction:

Below, I'll be borrowing the color scheme from Dr. Runck's Week 3 lecture as a way to model the different parts of the urls used in this lab to access the three datasets (Runck 2024):

Main URL of host of API

Specific service and function

Response Type

Function inputs for search

NDAWN:

The NDAWN API has no specific documentation available for the public to view in order to understand how to structure the queries. However, playing around with the website, by clicking on specific stations, and generating tables to display in a web browser, provided hints as to how data should be accessed. Additionally, the website provides a "Export .csv" button on the web tables, which will create a url that will download a .csv file.

https://ndawn.ndsu.nodak.edu/table.csv?station=78&station=174&station=118&station=87&station=124&station=226&station=219&station=227&station=184&station=2&station=220&station

=223&station=183&station=156&station=70&station=173&station=185&station=187&station=119&station=4&station=82&station=225&station=120&station=71&station=103&station=116&station=114&station=115&station=61&station=181&station=60&station=122&station=5&station=182&station=117&station=6&station=222&station=92&station=123&station=95&station=148&variable=wdavt&ttype=weekly&quick_pick=&begin_date=2024-09-01&count=1

In yellow is the NDAWN website. In green, we can see where the response type is defined. I have used .csv because it is easy to use with pandas in my notebook, but if I changed that to "get-table.html", I would instead just load the web version of the table.

Everything in Blue afterwards defines what is going into that table. I chose a long list of stations, so that is the bulk of the url, but at the end there are also parameters for variable (&variable=wdavt) , type of table (&ttype=weekly), start date (&begin_date=2024-09-01), and number of weeks I want (&count=1). NDAWN does not appear to have different specific services or functions in the url, so there is no purple text in this url.

MN Geospatial Commons CKAN:
The Minnesota Geospatial Commons uses the CKAN API to organize their data requests. This is unique among the data used in this lab, as we use multiple calls to fully access the data. Understanding and exploring this API took more time, but demonstrates how this can be used to ensure that any data accessed from this site is fully up to date.

The first url I used is below. In yellow is the State of Minnesota website where the data is housed. In purple is the API service we are accessing, and in blue is the identifier of the package I want to access. The blue text identifies which package we want to see.

This package id is found at the end of the url for the package's traditional website (see https://gisdata.mn.gov/dataset/bdry-counties), or can alternatively be found by looking at a complete package list (see: https://gisdata.mn.gov//api/3/action/package_list)

https://gisdata.mn.gov//api/3/action/package_show?id=bdry-counties

This loads as a json which can then be read by python code to extract the url which will provide us with the shapefile used in the rest of the analysis.

Esri ArcGIS Online REST API:
Lastly, the Esri ArcGIS Online REST API was used to access a dataset from Hennepin county which contained all the municipalities in the county. In yellow we see that this is being accessed by a Hennepin county website. In purple we see that the service is the ArcGIS REST services, and then in blue the parameters that define the dataset. At the end in green, we can see the type of response I get back.

https://gis.hennepin.us/arcgis/rest/services/HennepinData/BOUNDARIES/MapServer/4/query?where=1%3D1&text=&objectIds=&time=&timeRelation=esriTimeRelationOverlaps&geometry=&geometryType=esriGeometryEnvelope&inSR=&spatialRel=esriSpatialRelIntersects&distance=&units=esriSRUnit_Foot&relationParam=&outFields=*&returnGeometry=true&returnTrueCur

ves=false&maxAllowableOffset=&geometryPrecision=&outSR=4326&havingClause=&returnIdsOnly=false&returnCountOnly=false&orderByFields=&groupByFieldsForStatistics=&outStatistics=&returnZ=false&returnM=false&gdbVersion=&historicMoment=&returnDistinctValues=false&resultOffset=&resultRecordCount=&returnExtentOnly=false&sqlFormat=none&datumTransformation=&parameterValues=&rangeValues=&quantizationParameters=&featureEncoding=esriDefault&f=geojson

While building the query using the GUI that is provided for these services, I only filled out certain fields (where=1%3D1, outFields=*, outSR=4326), and the rest are left blank. I specifically set the spatial reference at WGS 84 so I would not need to reproject it later. I also set it to return a geojson, knowing that the format plays well with pandas, and could easily be converted into whatever I needed later on.

Process

The NDAWN data was the easiest to manipulate once I learned how to work the API. I used pandas to read the csv directly into a dataframe, bypassing the need to save the .csv to my file system. Because the .csv has some unnecessary rows and merged cells that the dataframe does not like, I just skipped the problematic rows and had a readable dataframe. From there, I converted it to a spatial dataframe by setting the longitude and latitude fields, and it was ready to be used.

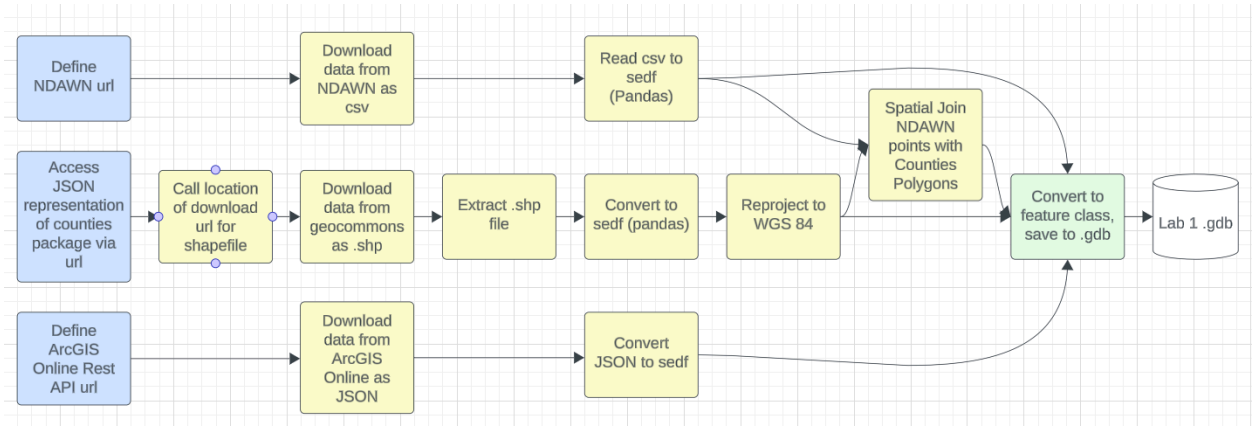
The counties data from the Minnesota Geospatial Commons was more involved. After using requests.get on the url detailed above, I needed to extract the download url from the package_show json that was returned. I chose to use a shapefile of this dataset for this stage. I needed to save it to my file system, and then extract the dataframe. I used the spatial.from_featureclass function to read the shapefile directly into a spatially enabled dataframe. This datatset's spatial reference was 26915, or UTM Zone 15N, so I needed to reproject it to WGS 84. It was now ready to be joined.

Lastly the Hennepin Cities data was accessed using the ArcGIS Online REST API. I read the json as a feature set, and then converted it to a spatially enabled dataframe. All three of my datasets were in spatially enabled dataframes and were able to be joined.

From there I joined the counties and the NDAWN dataframes together, resulting in the merged table in Figure 2 (below). I then used pandas' groupedby function to create a new dataframe that has the number of stations in each county, and joined that back to my county dataframe so I could more easily use this

Lastly, I created file geodatabase and exported each dataset into it as a feature class, using the to_featureclass function to deposit it in the geodatabase.

Figure 1. Data flow diagram.



If appropriate, add in pseudo-code describing model algorithms and/or objects. If using mathematical equations, create a clear mapping between the reference equation, pseudo-code, and actual implementation in a programming language.

Results

For this Lab, the requested final product was the merged join table for two of the three datasets. That table can be seen in Figure 2 below

Figure 2: printed merged table

	Station Name	Latitude	Longitude	Elevation	Year	Month	Day	Avg Temp	Number Missing	Number Estimated	SHAPE	index_right	FID	COUNTY_NAM	COUNTY_COD	COUNTY_FIP	COUNTY_GN
0	Ada	47.321190	-96.514060	910	2024	9	1	60.161	0	0	{"spatialReference": {"wkid": 4326}, "x": -96....	82	82	Norman	54	107	65949
1	Waukon	47.325810	-96.132580	1146	2024	9	1	60.634	0	0	{"spatialReference": {"wkid": 4326}, "x": -96....	82	82	Norman	54	107	65949
2	Alvarado	48.245940	-97.021530	809	2024	9	1	61.180	0	0	{"spatialReference": {"wkid": 4326}, "x": -97....	57	57	Marshall	45	089	65948
3	Stephen	48.456740	-96.853670	834	2024	9	1	62.138	0	0	{"spatialReference": {"wkid": 4326}, "x": -96....	57	57	Marshall	45	089	65948
4	Becker	45.343990	-93.850140	942	2024	9	1	63.109	0	0	{"spatialReference": {"wkid": 4326}, "x": -93....	3	3	Sherburne	71	141	65951

I wanted to change how the count of weather stations was displayed so created a new dataframe which showed the totals per county, for counties that had at least one station (Figure 3), then joined this to the counties dataset so it could all appear in one table and could for easier visualization (Figure 4)

Figure 3: the station count intermediate dataframe

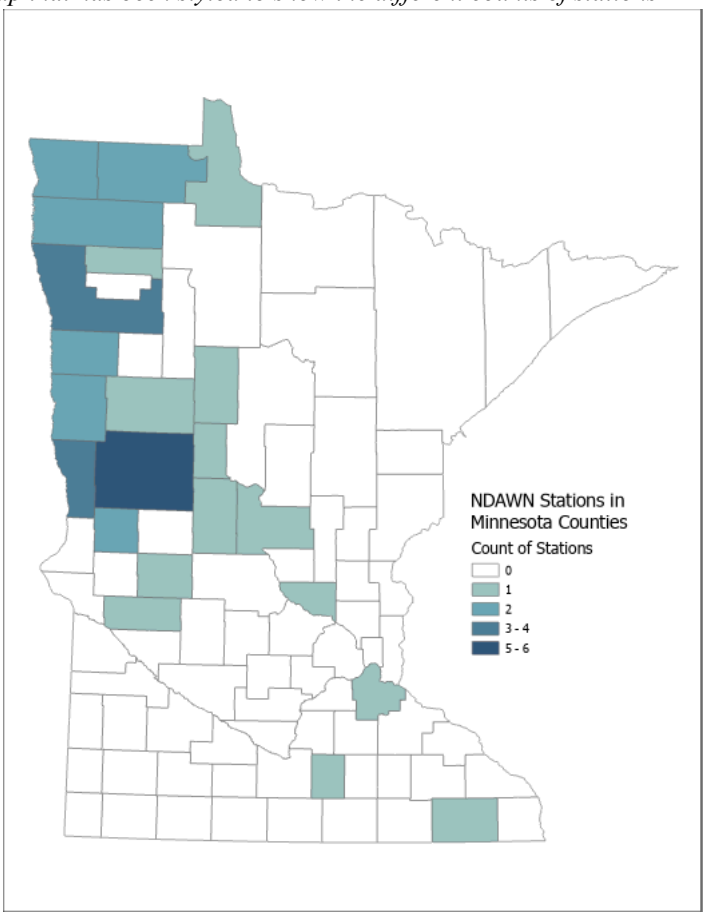
	COUNTY_NAM	station_count
0	Becker	1
1	Clay	2
2	Dakota	1
3	Fillmore	1
4	Grant	2
5	Hubbard	1
6	Kittson	2
7	Lake of the Woods	1
8	Marshall	2
9	Morrison	1
10	Norman	2
11	Otter Tail	6
12	Pennington	1
13	Polk	4
14	Pope	1
15	Roseau	2
16	Sherburne	1
17	Swift	1
18	Todd	1
19	Wadena	1
20	Waseca	1
21	Wilkin	3

Figure 4: the final table for merged dataset showing station count as an attribute for each county

	FID	COUNTY_NAM	COUNTY_COD	COUNTY_FIP	COUNTY_GNI	ATP_CODE	SHAPE_Leng	SHAPE_Area	SHAPE	station_count
0	0	Red Lake	63	125	659508	2	160725.772889	1121198845.29	{"rings": [[[-96.29203868018837, 47.9629636673...	0
1	1	Hennepin	27	053	659472	M	189829.833881	1570490144.08	{"rings": [[[-93.49957384649944, 45.2395687982...	0
2	2	Stearns	73	145	659517	3	280898.930397	3598882111.63	{"rings": [[[-95.1397848063617, 45.77343063417...	0
3	3	Sherburne	71	141	659515	3	163652.509384	1166966708.25	{"rings": [[[-94.03581736621345, 45.5590780377...	1
4	4	Murray	51	101	659496	8	173803.04716	1864170293.52	{"rings": [[[-95.46245846222979, 43.8479537270...	0

After this dataset was in my .gdb file, it was automatically added to the map. I wanted to visualize the count of stations in each county as a choropleth map so I quickly styled it (Figure 5).

Figure 5: a choropleth map that has been styled to show the different counts of stations



Results Verification

I can test against a more traditional (and labor intensive) means of importing these data sets into ArcGIS Pro and confirming that the data accessed via my code appears the same by using the other method. Additionally, I can verify the station counts associated with each county by displaying the county polygons and the weather station points at the same time and counting a few of the counties manually to confirm that the results are correct.

Discussion and Conclusion

Over the course of this lab I learned how to build an ETL pipeline for three different types of APIs. The API calls were handled differently in each of the required sources, but each method demonstrated the utility of using this method to access data and prepare it for further analysis or uses. While the NDAWN API and the Esri REST API both were ultimately structured via queries in a url, they were different in how those queries are generated and how easy it is to learn about which query parameters are used. The Minnesota Geospatial Commons CKAN API was structured very differently, however, and is a good example of an alternative approach to making this data available.

Each of these methods are much quicker to execute than any method that relies on hand-download operation and manual loading into ArcGIS Pro. This lab demonstrates the utility

of using APIs and ETL pipelines for working with spatial data, especially when the process requires repeatability.

References

panadas. *pandas.DataFrame.groupby.* The pandas development team.
Retrieved October 6, 2024, from
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html

The CKAN API. The Open Knowledge Foundation
Retrieved September 28, 2024 from https://docs.ckan.org/en/ckan-2.1.5/api.html

Esri. *ArcGIS API for Python.* Esri.
Retrieved October 6, 2024 from https://developers.arcgis.com/python/latest/

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	**23**
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	**26**
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	**95**