**Lab Report**

Title: Lab 1- API Deconstruction

Notice: Dr. Bryan Runck

Author: Taryn Reitsma

Date: October 8, 2024

**Project Repository:** <https://github.com/taryn-reitsma/GIS5571/tree/main/Lab1>

**Google Drive Link:** N/A

**Time Spent:** 20

**Abstract**

In this lab, I explored the process of deconstructing different APIs to extract data, clean and prepare it, and then spatially join two datasets. The problem centers around understanding how various API structures differ and how to effectively extract and transform their data. I worked with three datasets: a GeoJSON file from the ArcGIS REST API, a CSV file from the NDAWN API, and a shapefile from the Minnesota Geospatial Commons CKAN API. Each dataset required different methods for extraction and processing. I used Python libraries such as `requests` to retrieve the data, and Pandas and ArcGIS features modules to clean and format it. The workflows involved preparing spatial geometries and normalizing attribute data to make them compatible for a spatial join. After extracting the data, I focused on spatially joining the NDAWN weather station data with the Minnesota counties dataset from the ArcGIS REST API. The NDAWN data was converted to a spatially enabled DataFrame (SEDF), and the county geometry was modified to fit ArcGIS requirements.

The results were verified by checking for errors, ensuring correct geometries, and confirming that all columns from both datasets were included after the spatial join. The success of the join was visualized in a table showing the combined datasets. Overall, this lab demonstrated how APIs vary in structure and complexity, and how to effectively navigate their differences to extract and integrate spatial data. The exercise improved my understanding of API data extraction and spatial analysis workflows.

**Problem Statement**

APIs vary significantly in their structure and functionality across different platforms, making it essential to understand how to break them down and extract meaningful data. In this lab, I will deconstruct three different APIs, each with unique interfaces, to extract spatial and attribute data. My goal is to build an ETL pipeline that handles these differences and prepares the datasets for spatial analysis. After processing, I will perform a spatial join on two of the datasets using Jupyter Notebooks. This process will deepen my understanding of API structures and their role in geospatial workflows.

Table 1.Target Data for Analysis

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **#** | **Requirement** | **Defined As** | **(Spatial) Data** | **Attribute Data** | **Dataset** | **Preparation** |
| 1 | GeoJSON from ArcGIS REST API | Raw input data from ArcGIS REST API (mn.gov) | County geometry | County name and location | [MN Counties](https://www.arcgis.com/home/webmap/viewer.html?url=https%3A%2F%2Fwebgis.dot.state.mn.us%2F65agsf1%2Frest%2Fservices%2Fsdw_govnt%2FCOUNTY%2FFeatureServer%2F0&source=sd) | Extract from API, reproject, isolate geometry |
| 2 | CSV from NDAWN | Raw Ada County yearly weather station data from NDAWN | Point geometry of weather station location (lat/long) | Station name, lat/long, year, average temp, g | [NDAWN Data](https://ndawn.ndsu.nodak.edu/get-table.html?station=78&variable=ydmxt&ttype=yearly) | Extract from API, isolate geometry |
| 3 | Shapefile from Minnesota Geospatial Commons | Roadway weather Information Sites data from Minnesota Geospatial Commons | Point Geometry of Weather info sites | Site location (lat/long), temperature, pressure, etc. | [Minnesota Geospatial Commons](https://gisdata.mn.gov/dataset/struc-roadway-weather-sites) | Extract from API, isolate geometry |

**Input Data**

I will be using three datasets from three different sources with varying APIs. I will be extracting data from NDAWN, and ArcGIS REST API, and the Minnesota Geospatial commons, which uses the CKAN API. The NDAWN dataset is weather station data from Ada, Minnesota, the ArcGIS data is Minnesota counties, and the Minnesota Geospatial Commons data is a road weather conditions point layer. Each dataset will be extracted from their respective API, cleaned and prepared for a spatial join.

Table 2. Datasets Used in Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Title** | **Purpose in Analysis** | **Link to Source** |
| 1 | County | This layer will be extracted from an ArcGIS REST API, then spatially joined to the NDAWN csv file | [MN Counties](https://www.arcgis.com/home/webmap/viewer.html?url=https%3A%2F%2Fwebgis.dot.state.mn.us%2F65agsf1%2Frest%2Fservices%2Fsdw_govnt%2FCOUNTY%2FFeatureServer%2F0&source=sd) |
| 2 | NDAWN Ada Weather Station table | This table will be extracted from the NDAWN API, then spatially joined to the County GeoJSON extracted from an ArcGIS REST API | [NDAWN Data](https://ndawn.ndsu.nodak.edu/get-table.html?station=78&variable=ydmxt&ttype=yearly) |
| 3 | Roadway Weather Sites, Minnesota | This layer will be extracted from the Minnesota Geospatial Commons API CKAN | [Minnesota Geospatial Commons](https://gisdata.mn.gov/dataset/struc-roadway-weather-sites) |

**Methods**

Figure 1. API Deconstruction Data Flow Diagram

*A screenshot of a diagram

Description automatically generated*

The process for deconstructing each API varied due to the differing structures. I first began deconstructing the CKAN API from the Minnesota Geospatial commons. This API required use of a call “package\_show?’ to pull a specific dataset. Once I obtained this URL, I used the Python requests module to access the URL using ‘requests.get(url).’ This gave me access to the metadata, where I was able to locate the zipped file. I was then able to save and extract the zip file locally to obtain the shapefile of Minnesota Counties.

For the ArcGIS REST API, the process was slightly different. I manually entered a query using the ArcGIS Services Directory, where I was then redirected to a page with the GeoJSON data. I took the URL from the redirect to pull the data using ‘requests.get(url)’. I performed one more step, which was to extract the features from the JSON and normalize it into a Pandas DataFrame to isolate the geometry and attributes.

Lastly, I accessed the NDAWN API by deconstructing the URL into each attribute to determine what I needed to obtain the data. I downloaded a .csv table of yearly maximum temperatures from the Ada weather station using ‘requests.get(url)’. This left me with the raw .csv file directly from the NDAWN website.

Figure 2. Spatial Join Data Flow Diagram

A diagram of a company

Description automatically generated

I chose to spatially join the NDAWN csv and GeoJSON from the ArcGIS REST API for this exercise. The first step was ensuring both files were in a spatially enabled data frame that ArcGIS would accept. The NDAWN csv had columns for latitude and longitude, so I was able to convert the .csv to a SEDF using the GeoAccessor package from the arcgis.features module, and I used the function ‘GeoAccessor.from\_xy’ to convert the .csv into a SEDF.

For the GeoJSON file, I first had to convert the geometry into a format ArcGIS would accept. I took the raw geometry column ‘geometry.coordinates’ and applied a lambda function to separate the coordinates for each row. Then, I converted the new coordinate structure to ‘rings’ geometry, which ArcGIS would accept as polygons. I simultaneously set the spatial reference to WKID 4326 and set the resulting geometry to a new column ‘SHAPE’ in the DataFrame, which would now be accepted as a SEDF.

Lastly, I used the ‘spatial.join’ function to spatially join the two datasets, using a left join and intersect, and saved the resulting dataset to a new file geodatabase named ‘FGDB.gdb’ that I created using ArcPy.

**Results**

Figure 3. Spatial Join Table Preview

*A screenshot of a computer

Description automatically generated*

This figure is the result of using the head() function to print out the first few lines of the spatial join between the NDAWN dataset and the ArcGIS REST dataset. The purpose of this figure is to display the success of the spatial join between the two datasets. The columns leading up to ‘SHAPE’ are the columns from the NDAWN dataset, and the following columns are from the ArcGIS REST dataset. The joined columns show that the spatial join of the datasets was successful.

**Results Verification**

The results of my spatial join were correct because the columns from both datasets were successfully included (Fig. 3), and no errors or warnings arose from the join process. Throughout the workflow, I performed several checks to ensure accuracy and data integrity.

First, I periodically printed out intermediate datasets during the cleaning process to verify that the correct transformations were applied. This included checking that unnecessary columns were removed and that the geometry and projections were standard among datasets. I then verified the geometry by ensuring both datasets were using the correct spatial reference system (WKID 4326). After ensuring that both datasets were properly aligned, I used ArcGIS tools to inspect the geometries and visually confirmed that the NDAWN points aligned correctly with the county polygons.

Lastly, I performed a series of checks after the spatial join was completed. I confirmed that all expected columns were present in the resulting dataset and that each weather station from the NDAWN .csv had been correctly matched with its corresponding county from the ArcGIS REST data. I also rechecked that no rows were lost or incorrectly duplicated during the join process. By regularly checking my work and addressing issues like unnecessary columns or geometry problems early on, I ensured that the spatial join was accurate and that the final output could be uploaded to my file geodatabase without any issues.

**Discussion and Conclusion**

Web APIs can vary in structure and useability, and the APIs that I deconstructed in this exercise displayed this variation quite obviously. The NDAWN API was least structured and required the most trial and error to understand how the structure worked, and the ArcGIS REST API was the most structured and straightforward. I found the ArcGIS REST API the easiest to work with, but it is important to note that I was already taught how to use the query pages to scrape data in my work, so this also gave me a level of comfort when working with it.

I found the CKAN API from the Minnesota Geospatial Commons the most difficult to work with because it was the least straight forward to me. The NDAWN API was frustrating at times but using the developer tools and the web inspector were incredibly helpful in deconstructing the URL. The CKAN documentation was difficult for me to understand, and once I was able to get the correct URL, parsing through the metadata to find the zip file was tedious. My primary suggestion for CKAN would be to not have the information nested so deep within the metadata, but I am not sure how to advise NDAWN. I find the ArcGIS REST API the most user friendly, if you have someone walk you through it the first time. Overall, I found this exercise to be very helpful in understanding how APIs work and how different interfaces operate.

**References**

Esri. (2019). Introduction to the Spatially Enabled DataFrame | ArcGIS API for Python. Arcgis.com. <https://developers.arcgis.com/python/latest/guide/introduction-to-the-spatially-enabled-dataframe/>?

Esri. (2022). arcgis.gis module | ArcGIS API for Python. Arcgis.com. <https://developers.arcgis.com/python/latest/api-reference/arcgis.gis.toc.html>

Feature Class To Geodatabase (Conversion)—ArcGIS Pro | Documentation. (2024). Arcgis.com. <https://pro.arcgis.com/en/pro-app/latest/tool-reference/conversion/feature-class-to-geodatabase.htm>

Minnesota Geospatial Commons. (2024). API Developer Resources | gisdata.mn.gov. Mn.gov. <https://gisdata.mn.gov/content/?q=help/api>

**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 | **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** |