Name: Jeimy Martinez De La Hoz
Date: 12/04/25

Final Project: Project 1

URL:
https://github.com/Jeimymdelahoz/geog4057_JeimyMartinez/tree/fd552d3fbd26314209013728d02d3c894a7581c3/Project1

1. Introduction

1.1 Project Summary

This project focuses on converting a land value dataset for New Orleans, originally downloaded from *data.gov*, into a GIS-compatible format. The dataset is provided as a JSON file containing polygon geometries encoded in **Well-Known Text (WKT)** along with multiple attribute fields. Although ArcGIS Pro includes tools for importing JSON, it cannot directly process this specific WKT-based structure using tools such as *JSON to Features*.

To address this limitation, a custom Python script was developed using **ArcPy** to:

- Read and parse the JSON file
- Convert WKT geometry strings into polygon feature objects
- Create a new shapefile with appropriate fields
- Insert all records into the shapefile
- Visualize the resulting dataset in ArcGIS Pro through a map layout

The final output includes both the generated shapefile and a formatted map layout exported as a PDF.

1.2 Data Format and Description: The data is available in project/data as "no_tax.json". The data file is in the JavaScript Object Notion (JSON) format. Using the python json package can help interpret the file. Two parts in the json file are useful: "meta" and "data". 'Meta" includes descrpitons of fields. "Data" includes geometries and field value.

2. Methodology

The methodology for this project followed a structured workflow beginning with understanding the dataset, developing a Python-based conversion routine, and completing the visualization of the output in ArcGIS Pro. The process integrated exploratory analysis, geoprocessing automation, and cartographic design to produce a functional shapefile and a final layout.

2.1 Explanation of data sources

The dataset used in this project originates from the *2018 Land Value for New Orleans* collection published on data.gov. It is provided as a JSON file containing both attribute fields and polygon geometries expressed in Well-Known Text (WKT). The JSON structure consists primarily of two relevant components: the "meta" section, which defines field names and metadata descriptors, and the "data" section, which stores the actual records. Because ArcGIS Pro does not directly ingest JSON files containing WKT geometry, a preliminary inspection was conducted in a Jupyter Notebook to examine the structure of the file, confirm the index of the geometry field, and evaluate how attributes should be mapped into a resulting shapefile. This exploration ensured that the subsequent conversion script would correctly interpret and process the dataset.

2.2 processing steps

After the dataset was examined, a custom Python script was created using the ArcPy library to automate the conversion from JSON to a GIS-compatible shapefile. The script began by loading and parsing the JSON file using Python's built-in json module. WKT geometry strings were extracted and converted into ArcPy polygon objects using the arcpy.FromWKT() function. A new shapefile was then generated with the appropriate spatial reference system (EPSG 4326), followed by the creation of attribute fields derived from the metadata definitions in the JSON file. Because ArcGIS imposes restrictions on field name length and characters, names were cleaned and standardized before being added to the feature class. Each record from the JSON file was subsequently inserted into the shapefile using an InsertCursor, ensuring the accurate transfer of both geometric and tabular information.

2.3 Analysis Approach

Following the creation of the shapefile, the dataset was imported into ArcGIS Pro to complete visualization and cartographic analysis. The feature layer was symbolized to improve interpretability and visually distinguish land value polygons from the basemap. A layout was designed that incorporated a map frame, legend, title, scale bar, and north arrow, reflecting standard cartographic conventions. This layout provided a clear and coherent representation of the converted dataset. The final analytical product, a formatted PDF map, served as the primary visual output and demonstrated the successful transformation of the original JSON data into a usable GIS layer.

3. Code Documentation

This section provides a high-level overview of the Python script developed to convert the JSON dataset into a polygon shapefile. The script was written using the ArcPy library and is organized around a single primary function, json2shape (), which encapsulates all major processing steps. The design emphasizes modularity, readability, and compatibility with ArcGIS Pro's geoprocessing environment.

- **Function Overview: json2shape ():** The core of the project is the json2shape () function, which automates the conversion workflow. The function accepts four parameters:

- input_json: path to the source JSON file
- workspace: output directory where the shapefile will be saved
- fcname: name of the new shapefile
- wkid: spatial reference identifier (default EPSG 4326)

These inputs allow the script to be reused for similar datasets with minimal modification.

- **Loading and Parsing the JSON File:** The script begins by loading the JSON file using Python's built-in json module. This step extracts both the "meta" section (field definitions) and the "data" section (attribute values and geometry), allowing the script to dynamically generate attribute fields and populate them with valid content.
- **Geometry Conversion Using arcpy. FromWKT ():** Because ArcGIS Pro cannot ingest WKT directly, the script converts each WKT string into an ArcPy polygon object: polygon = arcpy.FromWKT(row[8]). These geometry objects are stored temporarily and later inserted into the feature class. This conversion step is essential for transforming text-based geometry into a native GIS format compatible with ArcGIS.
- **Creating the Shapefile:** The script uses arcpy.management.CreateFeatureclass() to generate an empty polygon feature class:

```
arcpy.management.CreateFeatureclass(out_path=workspace,out_name=fcname,
                                    geometry_type='POLYGON',
                                    spatial_reference=4236)
```

0.1s Python

This establishes the shapefile structure, geometry type, and coordinate system before attribute fields are added.

- **Adding Attribute Field:** Field names are extracted from the "meta" section. Because ArcGIS imposes restrictions on field length and characters, field names are standardized (e.g., truncated to 10 characters, spaces removed). The script adds each field using:

```
for ind,field_name in enumerate(field_names):
    arcpy.management.AddField(fc_fullname,
                              field_name=field_name,
                              field_type=field_type[ind])
```

0.7s

This ensures that all attributes from the JSON file are preserved and stored appropriately in the shapefile.

- **Inserting Records with InsertCursor:** The final step involves populating the feature class. For each row in the JSON "data" array:
    - All non-geometry attribute values are collected
    - A corresponding polygon geometry is appended
    - The combined record is inserted into the shapefile

This is done using:

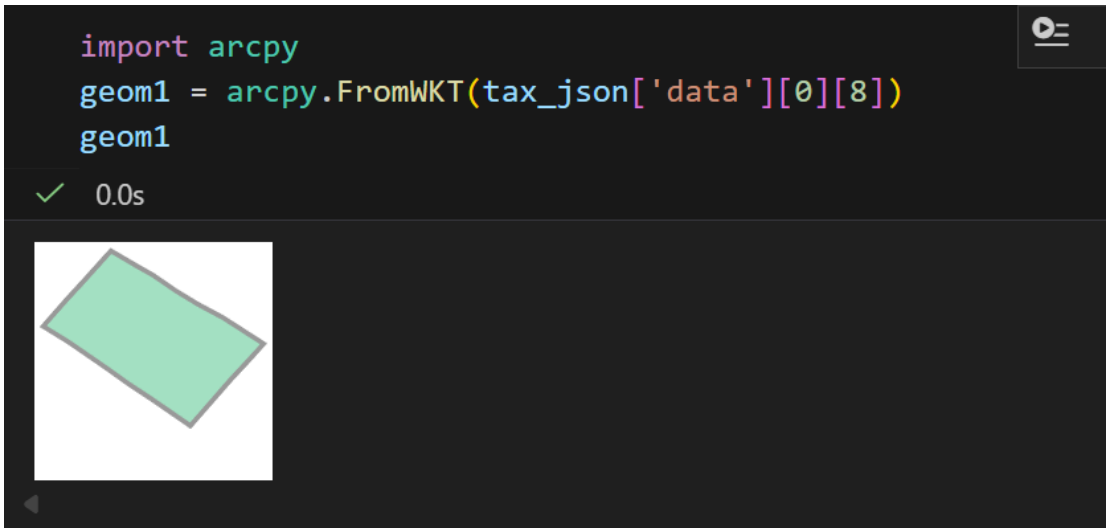
```
with arcpy.da.InsertCursor(fc_fullname, field_names=field_names) as cursor:
    for row in tax_json['data']:
        new_row_vals = []

        for ind, value in enumerate(row):
            if ind == 8:
                continue
            if value is None:
                value = ""
            new_row_vals.append(value)
```

The cursor guarantees that each row correctly aligns attributes with its associated polygon geometry.

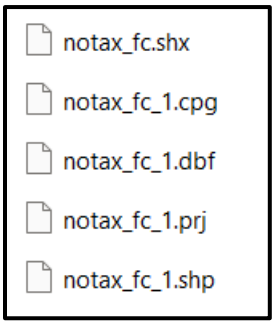
- **Script Execution:** The script concludes with an if __name__ == "__main__": block that provides example inputs and allows the user to run the function directly. This design supports both standalone execution and reuse within other geoprocessing workflows.

4. Results and Visualization

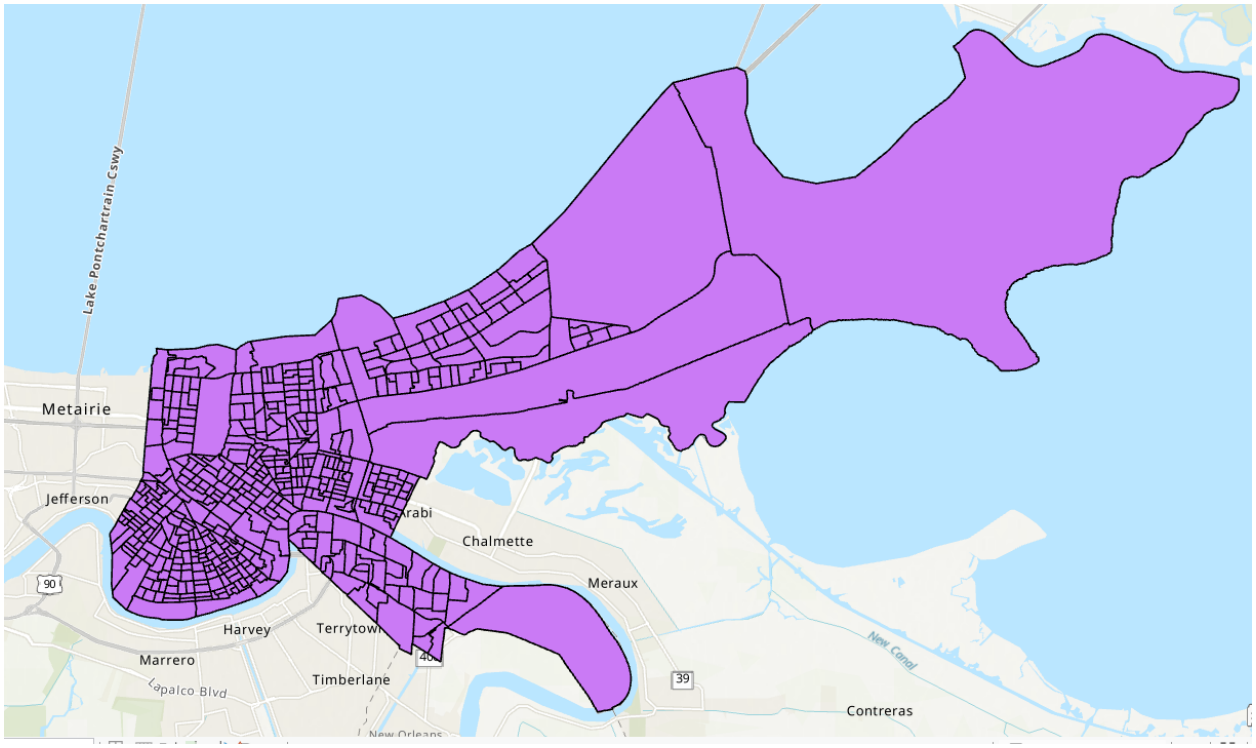
The execution of the Python script successfully produced a polygon shapefile representing the 2018 land value geometry for New Orleans.



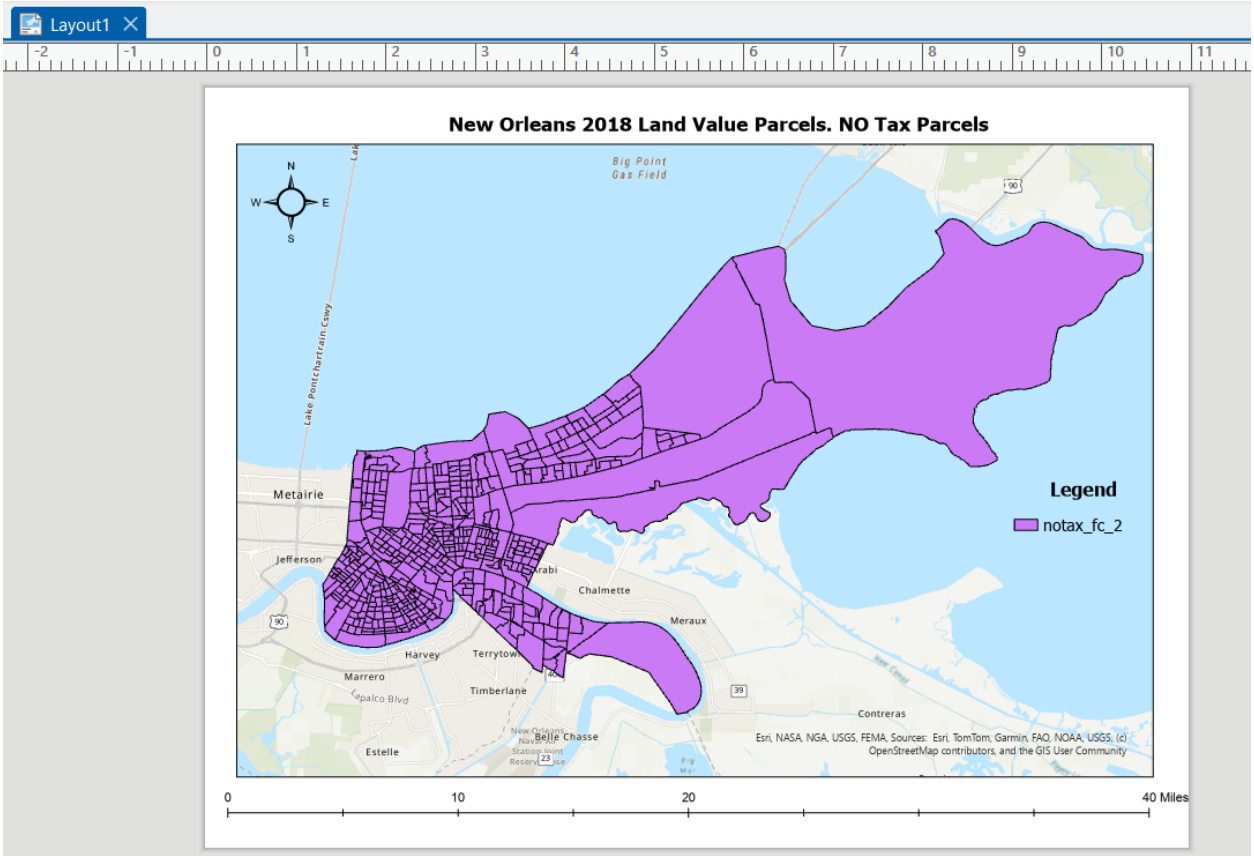
All WKT geometries from the JSON file were correctly converted into ArcPy polygon objects and inserted into a feature class with their associated attributes. Verification in ArcGIS Pro confirmed that the spatial features aligned properly and that all attribute fields were transferred without errors. The resulting shapefile displayed the expected spatial coverage and polygon boundaries, demonstrating the accuracy of both the parsing logic and the geometry conversion process.



Following the creation of the shapefile, the dataset was visualized within a new ArcGIS Pro project. The shapefile was added to a map and symbolized using a single-symbol color scheme for clear representation of the study area. Additional symbology adjustments, such as outline color and transparency, were applied to enhance legibility. The visualization process also included framing the map extent to ensure that all polygon features were fully visible and centered within the layout.



A final layout was created to present the results in a cartographic format. Standard layout elements including title, map frame, legend, scale bar, and north arrow, were added to communicate the geographic context and improve interpretability. The completed layout was exported as a PDF, providing a polished visual summary of the converted data. This exported map serves as the formal output of the project and demonstrates the successful integration of Python-based data conversion with ArcGIS Pro visualization capabilities.



5. Challenges and Future Work.

Several challenges were encountered during the development and implementation of the JSON-to-shapefile conversion workflow. Many of these issues were related to character encoding and ensuring that the standalone Python script correctly reused functions initially developed in the Jupyter Notebook environment. Additional difficulties involved attribute type casting and ensuring that the generated shapefile complied with ESRI format limitations, including restrictions on field name length and supported data types. Assigning the correct spatial reference and validating polygon geometries also required extra verification steps to ensure the accuracy of the final dataset.

Future improvements could enhance both the robustness and scalability of the tool. One potential direction is to generalize the script so that it can automatically detect and process any JSON dataset containing WKT geometries, rather than only the specific land-value dataset used in this project. Incorporating more comprehensive error-handling routines for malformed geometries

or missing attributes would further increase reliability. Finally, integrating spatial analysis capabilities, such as clustering, zoning assessments, or land-value heatmaps, directly into the workflow would allow the tool to function not only as a data-conversion utility but also as the foundation for more advanced GIS-based analytical applications.