Hemanta Pokharel

Dr. Lei Wang

GEOG 4057

6 Apr 2025

<p align="center">Project-1</p>

Introduction

This project is all about converting the JSON file to shape file using the python and the python toolbox. It is necessary to do such because of the problem that 'the JSON file is not directly readable in the ArcGIS'. The JSON file contains WKT (Well-Known Text) format of geometries and attributes that could be used in the GIS, which will be imported to a shapefile using the script tool and visualized.

Exploration of the Data, procedure, and results

1. Initial JSON Structure Exploration

This will show you the main components of your JSON structure like 'data' and 'meta'.

```python
import json
with open('no_tax.json') as file:
    tax_json=json.load(file)

print("Top-level keys:", tax_json.keys())
[1]  ✓ 0.0s
...  Top-level keys: dict_keys(['meta', 'data'])
```

<p align="center">Figure 1: Load the data</p>

2. Explore metadata

```
2 Data structure

    # How many records are there?
    print("Number of rcords:", len(tax_json['data']))

    # what does the first record look like?
    print(tax_json['data'][0])
[1]   ✓ 0.0s

Number of rcords: 496
['row-69eh-dt2h-vwz3', '00000000-0000-0000-A344-B176ECD7FE9B', 0, 1628101573, None, 1628101573, None,
```

Figure 2: data structure

3. Counting the number of items in the columns list and displaying them

```
    # get information about the columns
    columns = tax_json['meta']['view']['columns']
    len(columns)
[24]   ✓ 0.0s

14

    # Print all field names
    print("Field names:")
    for column in columns:
        print("-", column['name'])
[23]   ✓ 0.0s

Field names:
- sid
- id
- position
- created_at
- created_meta
- updated_at
- updated_meta
- meta
- the_geom
- OBJECTID
- ID
- Cluster Letter
- Shape.STArea()
- Shape.STLength()
```

Figure 3: understanding the fields (columns)

4. Here, we find which column in the columns list contains the geometry. One method is by directly viewing the data and finding it manually. Second method is to find it using the word 'geom'

```
tax_json['data']
[26]    ✓ 0.2s
...    [['row-69eh-dt2h-vwz3',
         '00000000-0000-0000-A344-B176ECD7FE9B',
         0,
         1628101573,
         None,
         1628101573,
         None,
         '{ }',
         'MULTIPOLYGON (((-90.092842237961 29.969376832976, -90.09206523793 29.970255834178, -90.091376237438
         '101',
         '220710050002',
         'C',
         '3085170.5967876972',
         '7251.2480743083825'],
        ['row-7new-5v4m~u4mk',
```

```
# Find which field conta (type) columns = Any
for i, col in enumerate(columns):
    if 'geom' in col['name'].lower():
        print(f"Geometry is in field {i}: {col['name']}")
        break
[29]    ✓ 0.0s

...    Geometry is in field 8: the_geom
```

Figure 4: Field, where the geometry is located.

5. Here we import the required library "arcpy" and load the modules necessary. Then we convert the WKT geometry to ArcPy Geometry by assuming the 9th field (index 8) contains the geomeries in WKT format. Arcpy.FromWKT() converts the strings to ArcPy geometry objects.

Figure 5: Creating the arcPy geometry from the WKT

6. In this step we define the output path under "workspace" and filename "notax_fc.shp" under the "fcname". Then combine the workspace and the filename into a fullpath. It also deletes the shapefile if it already exists with the same name to avoid the conflicts. Then it initializes an empty shape file to store the polygon features with the spatial reference of WGS 1984. This will include FID (unique feature ID) and Shape (geometry field). Lastly, it also checks if the shapefile was created correctly and lists its default fields or not. arcpy.da.Describe() returns metadata about the shapefile.

```
import os
fcname = 'notax_fc.shp'
workspace = r'C:\Users\hpokha1\OneDrive - Louisiana State University\Documents\GitHub\GEOG_4057\Guided_Project\Project_1\data'
fc_fullname=os.path.join(workspace, fcname)
if arcpy.Exists(fc_fullname):
    arcpy.management.Delete(fc_fullname)
arcpy.management.CreateFeatureclass(workspace, fcname, 'POLYGON', spatial_reference=4236)

desc = arcpy.da.Describe(fc_fullname)
for field in desc['fields']:
    print(field.name)
```
[1] ✓ 0.3s

```
FID
Shape
Id
```

Figure 6: Creating feature class

7. This code processes the field metadata from the JSON file to prepare it for creating shapefile attributes. First, it extracts all field definitions from tax_json['meta']['view']['columns'] and filters out the geometry field ('the_geom'). It then defines data types for each field (TEXT for strings, LONG for integers) and processes the field names to meet shapefile requirements: renaming reserved names like 'id' to 'id_0', truncating names to 10 characters, and replacing spaces and periods with underscores. The cleaned field names are stored in field_names, which will later be used with arcpy.management.AddField() to build the shapefile's attribute table structure.

```
fields=tax_json['meta']['view']['columns']
fields=[field for field in fields if field['name']!='the_geom']
for field in fields:
    print(field['name'])

field_type=['TEXT','TEXT','LONG','LONG','TEXT','LONG','TEXT','TEXT','TEXT','TEXT','TEXT','TEXT']
field_names=[]
for ind,field in enumerate(fields):
    name=field['name']
    if name =='the_geom':
        continue
    if name.lower() =='id':
        name=f'id_{ind}'
    max_len=min(10,len(name))
    name=name[:max_len]


    field_names.append(name)
field_names=[field.replace(" ","_") for field in field_names]
field_names=[field.replace(".","_") for field in field_names]
field_names
```
[57] ✓ 0.0s

```
sid
id
position
```

Figure 7: processes the field metadata from the JSON file

8. This section adds the attribute fields to the empty shapefile. The loop goes through each field name in field_names list and uses "enumerate()" tracks the position "(ind)" of each field. Then builds the attribute table structure before loading the data.


```python
Adds attribute fields

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

field_names
```
✓ 1.7s

```
['sid',
 'id_1',
 'position',
 'created_at',
 'created_me',
 'updated_at',
 'updated_me',
 'meta',
 'OBJECTID',
 'id_9',
 'Cluster_Le',
 'Shape_STAr',
 'Shape_STLe']
```

Figure 8: Adding attribute fields to shapefile

9. "field_names.append('SHAPE@')" adds the special geometry field token 'SHAPE@' to the list. This will tell the arcpy where to put the polygon shapes when inserting the data. Also, now thw list becomes 14 items.

```
      field_names
[76]  ✓  0.0s

...   ['sid',
       'id_1',
       'position',
       'created_at',
       'created_me',
       'updated_at',
       'updated_me',
       'meta',
       'OBJECTID',
       'id_9',
       'Cluster_Le',
       'Shape_STAr',
       'Shape_STLe']


      field_names.append('SHAPE@')

[77]  ✓  0.0s


      len(field_names)
[78]  ✓  0.0s

...   14
```

Figure 9: Adding the special geometry field token

10. Here, the "arcpy.da.InsertCursor()" creates a data writer for the shapefile. Then it iterates through each feature in the JSON dataset and prepares the attribute for the insertion. "ind==8" skips the original WKT geometry at index 8, "None" converts the null to empty strings and builds a new list with only attribute values.

"new_row.append(row[8])" adds the pre-processed geometry. Row[8] contains the geometry object created earlier by arcpy.WKT(). This matches the 'SHAPE@' field position in the field list.

It also ensures the new row has exactly 14 attributes. Now the 'insertRow()' performs a binary write to the .shp file and attributes to the .dbf.

```
with arcpy.da.InsertCursor(fc_fullname,field_names=field_names) as cursor:
    for row in tax_json['data']:
        new_row=[]
        for ind,value in enumerate(row):
            if ind==8:
                continue
            if value ==None:
                value=""
            new_row.append(value)
        new_row.append(row[8])
        print(len(new_row))
        cursor.insertRow(new_row)
```

[64] ✓ 1.1s

```
14
14
14
14
14
14
14
14
```

Figure 10: Adding data to the feature class

∨ DATA

⊞ boundary.csv
🐍 convert_json_to_fc.py
📓 json_to_shapefile_conversion.ipynb
{} no_tax.json
≣ notax_fc.cpg
≣ notax_fc.dbf
≣ notax_fc.prj
≣ notax_fc.shp
≣ notax_fc.shp.GEOG-121-2.17044.15804.sr.lock
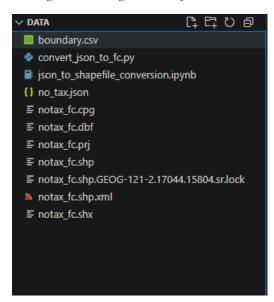⧹ notax_fc.shp.xml
≣ notax_fc.shx

Figure 11:Shape files

Layout map of Guided Project1

Figure 12: Map created from the shape file that is generated using the JSON

Python toolbox run

Previously created python file is defined as a working function, and it is called in the pyt file created from the ArcGIS. Then the parameters are set as Workspace, Input, and Output. Then the tool is run by selecting the respective files. Finally, the working tool to convert the JSON to shape files is create and run.


```
C: > Users > hpokha1 > OneDrive - Louisiana State University > Documents > programming > geog4057_hemanta > Guided_Project > Project_1 > ⊕ JsonToShapefile.py
1    #Exploring the json file
2    import json
3    import arcpy
4    import os
5
6    def importNoTaxJSON(workspace=r'C:\Users\hpokha1\OneDrive - Louisiana State University\Documents\GitHub\GEOG_4057\Guided_Project
7        with open(json_file,'r') as file:
8            tax_json=json.load(file)
9
10       arcpy.FromWKT(tax_json['data'][8][8])
11       for row in tax_json['data']:
12           row[8]=arcpy.FromWKT(row[8])
13
14       #Create feature class and write fields
15       fcname=out_fc
16
17       fc_fullname=os.path.join(workspace,fcname)
18       if arcpy.Exists(fc_fullname):
19           arcpy.management.Delete(fc_fullname)
20       arcpy.management.CreateFeatureclass(out_path=workspace, out_name=fcname,geometry_type="POLYGON",spatial_reference=4236)
21
22       fields=tax_json['meta']['view']['columns']
23       fields=[field for field in fields if field['name']!='the_geom']
24
```

Figure 13: Creating a function for toolbox


```
C: > Users > hpokha1 > OneDrive - Louisiana State University > Documents > programming > geog4057_hemanta > Guided_Project > Project_1 > ⊕ json_featureclass.p
1    |# -*- coding: utf-8 -*-
2
3    import arcpy
4    from JsonToShapefile import importNoTaxJSON
5
6    class Toolbox:
7        def __init__(self):
8            """Define the toolbox (the name of the toolbox is the name of the
9            .pyt file)."""
10           self.label = "Toolbox"
11           self.alias = "toolbox"
12
13           # List of tool classes associated with this toolbox
14           self.tools = [Tool]
15
16
17   class Tool:
18       def __init__(self):
19           """Define the tool (tool name is the name of the class)."""
20           self.label = "Tool"
21           self.description = ""
22
23       def getParameterInfo(self):
```

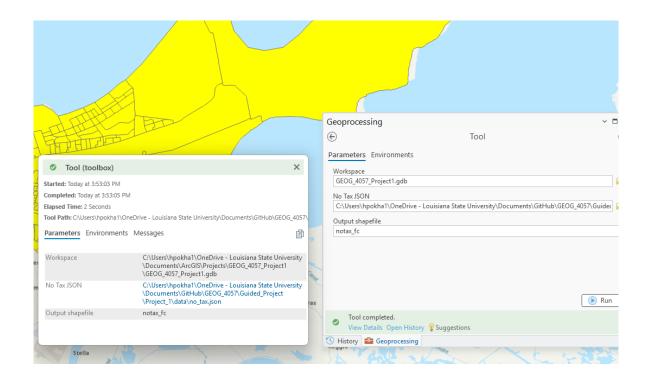*Figure 14: Creating a python toolbox*

Figure 15: Python toolbox interface and successful run

Conclusion

This project successfully demonstrated the conversion of JSON data containing WKT geometries into an ArcGIS shape file using Jupiter notebook and Python toolbox.