Python for Bioinformatics

adventures in bioinformatics

Wednesday, June 24, 2020

GeoJSON

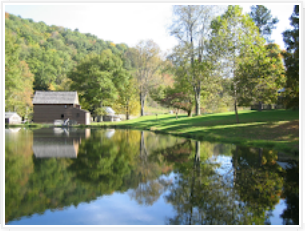
GeoJSON is a JSON representation of geographic data. Although JSON was developed for javascript, the format should be very familiar to any Python programmer. There are collections: dicts and lists, as well as simple types like: string, number, boolean or null. Dictionaries are called objects, and may also be values contained in lists or other dictionaries.

A typical GeoJSON file looks like this:

```
{"type":"FeatureCollection",
 "features":
  [
    {"type":"Feature",
     "properties":{"GEO_ID":"0500000US01001",
                   "STATE":"01",
                   "COUNTY":"001",
                   "NAME":"Autauga",
                   "LSAD":"County",
                   "CENSUSAREA":594.436
                  }
     "geometry": {"type":"Polygon",
                  "coordinates": [[ [-86.496774, 32.344437],
                                                ...
                                    [-86.496774, 32.344437]
                                  ]]
                 }
     "id":"01001"
    }

    {
    "type":"Feature",
    ...
                   "STATE": "01",
                   "COUNTY": "009",
                   "NAME": "Blount"
    ...
    }

  ]
}
```



Jackson's Mill WV

Search This Blog

Search

Labels

- 16S rRNA (10)
- alignments (7)
- bayes (17)
- binary (5)
- bindings (1)
- Bioconductor (7)
- bioinformatics (77)
- BLAST (8)
- book (1)
- C (8)
- calculus (14)
- command line (15)
- cool stuff (1)
- COVID-19 (17)
- crypto (10)
- ctypes (5)
- Cython (3)
- dental project (6)
- distributions (20)
- DNA binding sites (6)
- duly quoted (31)
- EMBOSS (3)
- fun (16)
- Geometry (26)
- go (4)

(I've formatted whitespace to my taste, YMMV).

At top level, it is a dict with two keys. The `"type"` is `"FeatureCollection"`.

The second key is `"features"`, which is a list of Feature objects.

The `"id"` is a FIPS code for the Feature. This Feature is Autauga County, Alabama. The FIPS code for the state is `"01"` and the full FIPS for the county is `"01001"`.

Each Feature is a dict with four keys. The `"properties"` and `"geometry"` keys yield dicts in turn. The `"geometry"` dict has a key `"coordinates"` which gives a list of longitude and latitudes for each vertex of the Feature's map.

The coordinates are nested inconsistently. It it's two deep: a list with one element that is a list with many 2-element vertices. Sometimes it's three deep.

[Update: this is a great over-simplification. The problem is that these are Multipolygons, and they do mean multi. That will have to wait for another post. Here's a link to the Mapbox documentation.]

The number of vertices depends on the resolution. For some counties at high resolution, it could run to 300 or more tuples.

For use with `plotly.express.choropleth`, there's no reason to parse this stuff or to generate a file with only the desired Features.

Just load the data

```
import json
with open(fn,'r') as fh:
    counties = json.load(fh)
```

then make a pandas data frame with one or more elements like

```
    fips  value
0  01001      1
1  06071      2
```

by

```
import pandas ad pd
pd.DataFrame({'fips':list_of_fips, 'value': list_of_values})
```

(or just pass `{'fips':list_of_fips, 'value': list_of_values}` as the `df` argument to `choropleth`).

```
fig = px.choropleth(
    df,
    geojson=counties,
    locations='fips',
    color=["Autauga","San Bernardino"],
    color_discrete_sequence=cL,
    scope='usa')

fig.show()
```

The locations argument says to match the `"fips"` from the data frame with the default value in the GeoJSON, which is its `"id"`. The colors can be specified as a color list like

```
cL = ['green','magenta']
```

See the plotly tutorial section on colors for more detail. The scope argument limits the region plotted on the map.

States are even easier, since plotly already knows the state GeoJSON data.

```
import plotly.express as px
fig = px.choropleth(
    locations=['CA','TX','SC'],
    locationmode="USA-states",
    color=[1,2,3],
    scope="usa")

fig.show()
```

The county GeoJSON data is available from plotly, see the first example for the URL).

I came across a collection of GeoJSON files on the web. This data is originally from the US Census, and has been converted to GeoJSON by the author of that post.

One slight hiccup is that the GeoJSON file includes Puerto Rico, which has municipalities with Spanish names. These are encoded with ISO-8859-1.

This uses high-order bytes to represent Spanish ñ, í and so on. Some of these byte sequences are not valid UTF-8.

```
> python3
Python 3.7.7 (default, Mar 10 2020, 15:43:33)
..
>>> fn = 'gz_2010_us_050_00_20m.json'
>>> fh = open(fn)
>>> data = fh.read()
```


- what we're saying (1)
- What we're thinking (2)
- Xcode (9)
- Xgrid (12)
- XML (9)

Unique visitors since Feb 21, 2011

Blog Archive

- 2022 (2)
- 2021 (13)
- 2020 (34)
  - December (2)
  - July (2)
  - June (6)
    - Shapefiles
    - Pythagorean theorem redux

```
Traceback (most recent call last):
..
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xed in position 935286: invalid continuat
```

Take a look:

```
>>> fh = open(fn,'rb')
>>> data = fh.read()
>>> data[935280:935290]
b'"Comer\xedo",'
>>> list(data[935280:935290])
[34, 67, 111, 109, 101, 114, 237, 111, 34, 44]
```

This is for Comerío Puerto Rico. The byte (237, 0xed) is mapped to the accented í But this and the following two bytes are

```
>>> '{:08b}'.format(237)
'11101101'
>>> '{:08b}'.format(111)
'01101111'
>>> '{:08b}'.format(34)
'00100010'
```

In UTF-8 the first byte starts with 1110, which indicates that this is a three-byte sequence, but the third byte of the group does not begin with 01 and so is not valid.

There is a nice module to document such issues. I do `pip3 install validate-utf8`.

```
> validate-utf8 src.json
invalid continuation byte
  COUNTY": "045", "NAME": "Comerío", "LSAD": "Muno", "CENSUSARE
                                  ^
...
```

To fix this

```
>>> fn = 'gz_2010_us_050_00_5m.json'
>>> fh = open(fn,'rb')
>>> s = fh.decode('ISO-8859-1')
>>> fn = 'counties.json'
>>> fh = open(fn,'w')
>>> fh.write(s)
>>> fh.close()
```

For my later explorations, I found it useful to parse the GeoJSON to a flat format with elements like:

```
01001
```

Albers projection
Plotting polygons
GeoJSON
COVID-19 data analysis

- May (19)
- April (4)
- March (1)

- 2018 (1)
- 2017 (7)
- 2016 (1)
- 2015 (16)
- 2014 (3)
- 2013 (2)
- 2012 (77)
- 2011 (174)
- 2010 (224)
- 2009 (265)
- 2008 (76)

About Me

telliott99

I'm retired, but used to teach and do research in Microbiology. This blog started as a record of my adventures learning bioinformatics and using Python. It has expanded to include Cocoa, R, simple math and assorted topics. As bbum says, it's so "google can organize my head." The programs here are developed on OS X using R and Python plus other software as noted. YMMV. I've had to turn comments off for the blog. Nothing but spam anymore. The intrepid reader will be able to find me. Hint: +"9" and I use gmail.

View my complete profile

```
Autauga
Alabama
-86.496774      32.344437
-86.717897      32.402814
-86.814912      32.340803
-86.890581      32.502974
-86.917595      32.664169
-86.71339       32.661732
-86.714219      32.705694
-86.413116      32.707386
-86.411172      32.409937
-86.496774      32.344437
```

Shapefiles and conversions are complicated enough that we'll leave them for another post.

Posted by telliott99 at 6/24/2020 12:13:00 PM 
Labels: maps

Newer Post Home Older Post