> Geographic Data Science

with

PySAL

and the

pydata stack

Sergio J. Rey

Dani Arribas-Bel

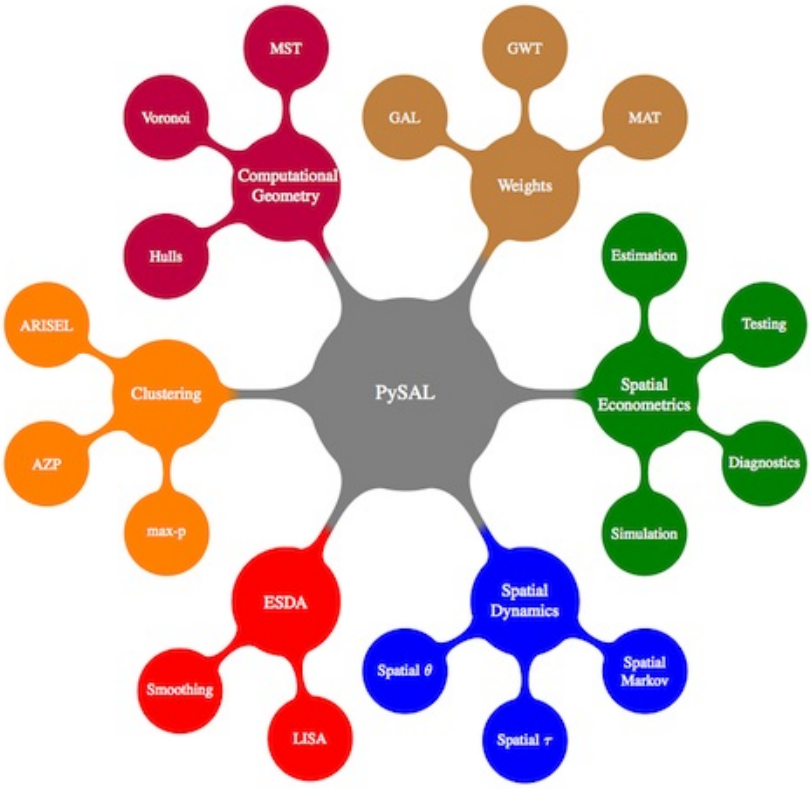
Table of Contents

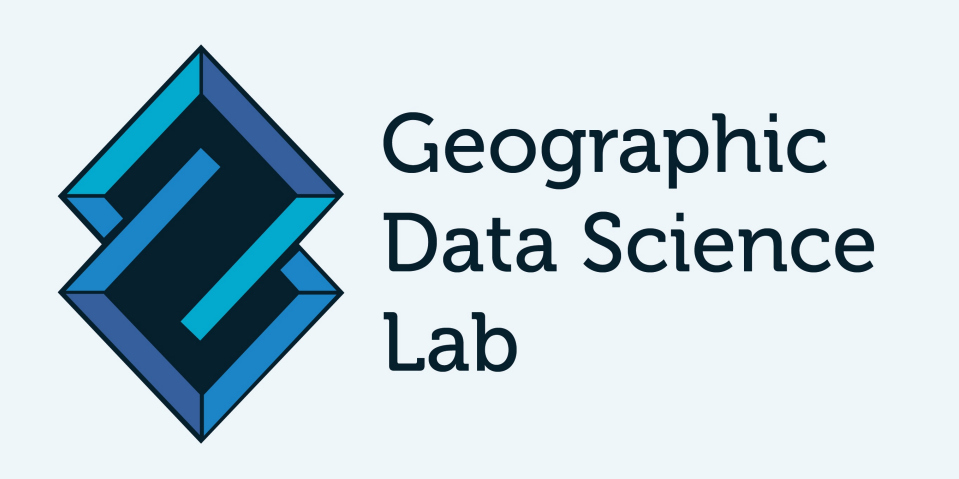
Introduction 1.1
Distribution 1.2
About the authors 1.3
Outline 1.4
Data 1.5
Part I 1.6
- Software and Tools Installation 1.6.1
- Spatial data processing with PySAL 1.6.2
- Spatial weights in PySAL 1.6.3
- Geovisualization 1.6.4
- ESDA with PySAL 1.6.5
- Space-time analysis 1.6.6

Part II 1.7
- Points 1.7.1
- Spatial clustering 1.7.2
- Spatial Regression 1.7.3

Geographic Data Science with PySAL and the pydata stack

This two-part tutorial will first provide participants with a gentle introduction to Python for geospatial analysis, and an introduction to version `PySAL` 1.11 and the related eco-system of libraries to facilitate common tasks for Geographic Data Scientists. The first part will cover munging geo-data and exploring relations over space. This includes importing data in different formats (e.g. shapefile, GeoJSON), visualizing, combining and tidying them up for analysis, and will use libraries such as `pandas`, `geopandas`, `PySAL`, or `rasterio`. The second part will provide a gentle overview to demonstrate several techniques that allow to extract geospatial insight from the data. This includes spatial clustering and regression and point pattern analysis, and will use libraries such as `PySAL`, `scikit-learn`, or `clusterpy`. A particular emphasis will be set on presenting concepts through visualization, for which libraries such as `matplotlib`, `seaborn`, and `folium` will be used.



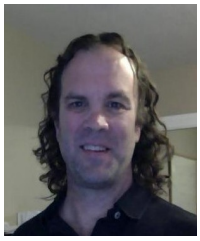

Geographic
Data Science
Lab

Distribution

> [URL] [PDF] [EPUB] [MOBI] [IPYNB]

License

About the authors



Sergio Rey is professor of geographical sciences and core faculty member of the GeoDa Center for Geospatial Analysis and Computation at the Arizona State University. His research interests include open science, spatial and spatio-temporal data analysis, spatial econometrics, visualization, high performance geocomputation, spatial inequality dynamics, integrated multiregional modeling, and regional science. He co-founded the Python Spatial Analysis Library (PySAL) in 2007 and continues to direct the PySAL project. Rey is a fellow of the spatial econometrics association and editor of the journal Geographical Analysis.

Dani Arribas-Bel is Lecturer in Geographic Data Science and member of the Geographic Data Science Lab at the University of Liverpool (UK). Dani is interested in undestanding cities as well as in the quantitative and computational methods required to leverage the power of the large amount of urban data increasingly becoming available. He is also part of the team of core developers of PySAL, the open-source library written in Python for spatial analysis. Dani regularly teaches Geographic Data Science and Python courses at the University of Liverpool and has designed and developed several workshops at different levels on spatial analysis and econometrics, Python and open source scientific computing.

Outline

Part I

1. Software and Tools Installation (10 min)
2. Spatial data processing with PySAL (45 min)
   a. Input-output
   b. Visualization and Mapping
   c. Spatial weights
3. Exercise (10 min.)
4. ESDA with PySAL (45 min)
   a. Global Autocorrelation
   b. Local Autocorrelation
   c. Space-Time exploratory analysis
5. Exercise (10 min)

Part II

1. Point Patterns (30 min)
   a. Point visualization
   b. Centrography and distance based statistics
2. Exercise (10 min)
3. Spatial clustering a (30 min)
   a. Geodemographic analysis
   b. Regionalization
4. Exercise (30 min)
5. Spatial Regression (30 min)
   a. Overview
   c. Basic spatial regression: spatial lag and error model
6. Exercise (10 min)

Data

This tutorial makes use of a variety of data sources. Below is a brief description of each dataset as well as the links to the original source where the data was downloaded from. For convenience, we have repackaged the data and included them in the compressed file with the notebooks. You can download it here.

AirBnb listing for Austin (TX)

Source: Inside AirBnb's extract of AirBnb locations in Austin (TX).

Part I

Software and Tools Installation

Dependencies

Participants should have installed the following dependencies:

- Anaconda or MiniConda Python distributions for Python 2.7. See installation instructions on the links.
- `git`
- A `conda` environment loaded with all the dependencies can be installed by running the `pydata.sh` script available as part of the `envs` repository (Github link). To install it, follow these instructions:
    - Clone the repository on your machine:

      `> git clone https://github.com/darribas/envs.git`
    - Navigate into the folder:

      `> cd envs`
    - Run the script:

      `> bash pydata.sh`

Once installed, you need to activate the environment to run the notebooks. In Windows, open up PowerShell and type:

`> activate pydata`

And if you are on GNU/Linux or OSX:

`> source activate pydata`

Get started

Instructions to fire up a notebook here.

```
#by convention, we use these shorter two-letter names
import pysal as ps
import pandas as pd
import numpy as np
```

Spatial Data Processing with PySAL & Pandas

PySAL has two simple ways to read in data. But, first, you need to get the path from where your notebook is running on your computer to the place the data is. For example, to find where the notebook is running:

```
!pwd
```

```
/home/serge/Dropbox/p/pysal/workshops/scipy16/gds_scipy16/content/part1
```

PySAL has a command that it uses to get the paths of its example datasets. Let's work with a commonly-used dataset first.

```
dbf_path = ps.examples.get_path('NAT.dbf')
print(dbf_path)
```

```
/home/serge/anaconda2/envs/scipy16/lib/python3.5/site-packages/pysal/examples/nat/NAT.dbf
```

For the purposes of this part of the workshop, we'll use the `NAT.dbf` example data, and the `usjoin.csv` data.

```
csv_path = ps.examples.get_path('usjoin.csv')
```

Working with shapefiles

To read in a shapefile, we will need the path to the file.

```
shp_path = ps.examples.get_path('NAT.shp')
print(shp_path)
```

```
/home/serge/anaconda2/envs/scipy16/lib/python3.5/site-packages/pysal/examples/nat/NAT.shp
```

Then, we open the file using the `ps.open` command:

```
f = ps.open(shp_path)
```

`f` is what we call a "file handle." That means that it only *points* to the data and provides ways to work with it. By itself, it does not read the whole dataset into memory. To see basic information about the file, we can use a few different methods.

For instance, the header of the file, which contains most of the metadata about the file:

```
f.header
```

```
{'BBOX Mmax': 0.0,
 'BBOX Mmin': 0.0,
 'BBOX Xmax': -66.9698486328125,
 'BBOX Xmin': -124.7314224243164,
 'BBOX Ymax': 49.371734619140625,
 'BBOX Ymin': 24.95596694946289,
 'BBOX Zmax': 0.0,
 'BBOX Zmin': 3.754550197104843e+72,
 'File Code': 9994,
 'File Length': 731108,
 'Shape Type': 5,
 'Unused0': 0,
 'Unused1': 0,
 'Unused2': 0,
 'Unused3': 0,
 'Unused4': 0,
 'Version': 1000}
```

To actually read in the shapes from memory, you can use the following commands:

```
f.by_row(14) #gets the 14th shape from the file
```

```
<pysal.cg.shapes.Polygon at 0x7f7ea0604668>
```

```
all_polygons = f.read() #reads in all polygons from memory
```

```
len(all_polygons)
```

```
3085
```

So, all 3085 polygons have been read in from file. These are stored in PySAL shape objects, which can be used by PySAL and can be converted to other Python shape objects.]

They typically have a few methods. So, since we've read in polygonal data, we can get some properties about the polygons. Let's just have a look at the first polygon:

```
all_polygons
```

```
[<pysal.cg.shapes.Polygon at 0x7f7ea06045f8>,
 <pysal.cg.shapes.Polygon at 0x7f7e8489e9b0>,
 <pysal.cg.shapes.Polygon at 0x7f7e8489ea20>,
 <pysal.cg.shapes.Polygon at 0x7f7e8489ea90>,
 <pysal.cg.shapes.Polygon at 0x7f7e8489eb00>,
 <pysal.cg.shapes.Polygon at 0x7f7e8489eb70>,
 <pysal.cg.shapes.Polygon at 0x7f7e8489ebe0>,
 <pysal.cg.shapes.Polygon at 0x7f7e8489ec50>,
 <pysal.cg.shapes.Polygon at 0x7f7e8489ecc0>,
 <pysal.cg.shapes.Polygon at 0x7f7e8489ed30>,
 <pysal.cg.shapes.Polygon at 0x7f7e8489eda0>,
 <pysal.cg.shapes.Polygon at 0x7f7e8489ee10>,
 <pysal.cg.shapes.Polygon at 0x7f7e8489ee80>,
 <pysal.cg.shapes.Polygon at 0x7f7e8489eef0>,
 <pysal.cg.shapes.Polygon at 0x7f7e8489ef60>,
 <pysal.cg.shapes.Polygon at 0x7f7e8489efd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce080>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce0f0>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce160>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce1d0>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce240>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce2b0>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce320>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce390>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce400>,
 <pysal.cg.shapes.Polygon at 0x7f7ea0604710>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce470>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce4e0>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce550>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce5c0>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce630>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce6a0>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce710>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce780>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce7f0>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce860>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce8d0>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce940>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ce9b0>,
 <pysal.cg.shapes.Polygon at 0x7f7e745cea20>,
 <pysal.cg.shapes.Polygon at 0x7f7e745cea90>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ceb00>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ceb70>,
 <pysal.cg.shapes.Polygon at 0x7f7e745cebe0>,
 <pysal.cg.shapes.Polygon at 0x7f7e745cec50>,
 <pysal.cg.shapes.Polygon at 0x7f7e745cecc0>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ced30>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ceda0>,
 <pysal.cg.shapes.Polygon at 0x7f7e745cee10>,
 <pysal.cg.shapes.Polygon at 0x7f7e8489e240>,
 <pysal.cg.shapes.Polygon at 0x7f7e745cedd8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745cef28>,
```

```
 <pysal.cg.shapes.Polygon at 0x7f7e745cef98>,
 <pysal.cg.shapes.Polygon at 0x7f7e745cefd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa0b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa128>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa198>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa208>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa278>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa2e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa358>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa3c8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa438>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa4a8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa518>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa588>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa5f8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa668>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa6d8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa748>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa7b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa828>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa898>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa908>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa978>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fa9e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745faa58>,
 <pysal.cg.shapes.Polygon at 0x7f7e745faac8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fab38>,
 <pysal.cg.shapes.Polygon at 0x7f7e745faba8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fac18>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fac88>,
 <pysal.cg.shapes.Polygon at 0x7f7e745facf8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fad68>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fadd8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fae48>,
 <pysal.cg.shapes.Polygon at 0x7f7e745faeb8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745faf28>,
 <pysal.cg.shapes.Polygon at 0x7f7e745faf98>,
 <pysal.cg.shapes.Polygon at 0x7f7e745fafd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ad0b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ad128>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ad198>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ad208>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ad278>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ad2e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ad358>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ad3c8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ad438>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ad4a8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ad518>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ad588>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ad5f8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ad668>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ad6d8>,
```

```
 <pysal.cg.shapes.Polygon at 0x7f7e745ad748>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ad7b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ad828>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ad7f0>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ad978>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ad9e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ada58>,
 <pysal.cg.shapes.Polygon at 0x7f7e745adac8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745adb38>,
 <pysal.cg.shapes.Polygon at 0x7f7e745adba8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745adc18>,
 <pysal.cg.shapes.Polygon at 0x7f7e745adc88>,
 <pysal.cg.shapes.Polygon at 0x7f7e745adcf8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745add68>,
 <pysal.cg.shapes.Polygon at 0x7f7e745addd8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745ade48>,
 <pysal.cg.shapes.Polygon at 0x7f7e745adeb8>,
 <pysal.cg.shapes.Polygon at 0x7f7e745adf28>,
 <pysal.cg.shapes.Polygon at 0x7f7e745adf98>,
 <pysal.cg.shapes.Polygon at 0x7f7e745adfd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e745540b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554128>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554198>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554208>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554278>,
 <pysal.cg.shapes.Polygon at 0x7f7e745542e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554358>,
 <pysal.cg.shapes.Polygon at 0x7f7e745543c8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554438>,
 <pysal.cg.shapes.Polygon at 0x7f7e745544a8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554518>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554588>,
 <pysal.cg.shapes.Polygon at 0x7f7e745545f8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554668>,
 <pysal.cg.shapes.Polygon at 0x7f7e745546d8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554748>,
 <pysal.cg.shapes.Polygon at 0x7f7e745547b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554828>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554898>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554908>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554978>,
 <pysal.cg.shapes.Polygon at 0x7f7e745549e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554a58>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554ac8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554b38>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554ba8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554c18>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554c88>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554cf8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554d68>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554dd8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554e48>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554eb8>,
```

```
 <pysal.cg.shapes.Polygon at 0x7f7e74554f28>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554f98>,
 <pysal.cg.shapes.Polygon at 0x7f7e74554fd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c0b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c128>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c198>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c208>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c278>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c2e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c358>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c3c8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c438>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c4a8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c518>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c588>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c5f8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c668>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c6d8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c748>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c7b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c828>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c898>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c908>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c978>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457c9e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457ca58>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457cac8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457cb38>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457cba8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457cc18>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457cc88>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457ccf8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457cd68>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457cdd8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457ce48>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457ceb8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457cf28>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457cf98>,
 <pysal.cg.shapes.Polygon at 0x7f7e7457cfd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e745200b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520128>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520198>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520208>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520278>,
 <pysal.cg.shapes.Polygon at 0x7f7e745202e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520358>,
 <pysal.cg.shapes.Polygon at 0x7f7e745203c8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520438>,
 <pysal.cg.shapes.Polygon at 0x7f7e745204a8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520518>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520588>,
 <pysal.cg.shapes.Polygon at 0x7f7e745205f8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520668>,
```

```
 <pysal.cg.shapes.Polygon at 0x7f7e745206d8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520748>,
 <pysal.cg.shapes.Polygon at 0x7f7e745207b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520828>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520898>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520908>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520978>,
 <pysal.cg.shapes.Polygon at 0x7f7e745209e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520a58>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520ac8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520b38>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520ba8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520c18>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520c88>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520cf8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520d68>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520dd8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520e48>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520eb8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520f28>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520f98>,
 <pysal.cg.shapes.Polygon at 0x7f7e74520fd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd0b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd128>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd198>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd208>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd278>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd2e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd358>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd3c8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd438>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd4a8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd518>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd588>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd5f8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd668>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd6d8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd748>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd7b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd828>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd898>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd908>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd978>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cd9e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cda58>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cdac8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cdb38>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cdba8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cdc18>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cdc88>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cdcf8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cdd68>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cddd8>,
```

```
 <pysal.cg.shapes.Polygon at 0x7f7e744cde48>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cdeb8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cdf28>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cdf98>,
 <pysal.cg.shapes.Polygon at 0x7f7e744cdfd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f00b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0128>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0198>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0208>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0278>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f02e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0358>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f03c8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0438>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f04a8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0518>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0588>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f05f8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0668>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f06d8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0748>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f07b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0828>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0898>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0908>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0978>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f09e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0a58>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0ac8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0b38>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0ba8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0c18>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0c88>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0cf8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0d68>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0dd8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0e48>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0eb8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0f28>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0f98>,
 <pysal.cg.shapes.Polygon at 0x7f7e744f0fd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e744950b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495128>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495198>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495208>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495278>,
 <pysal.cg.shapes.Polygon at 0x7f7e744952e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495358>,
 <pysal.cg.shapes.Polygon at 0x7f7e744953c8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495438>,
 <pysal.cg.shapes.Polygon at 0x7f7e744954a8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495518>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495588>,
```

```
 <pysal.cg.shapes.Polygon at 0x7f7e744955f8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495668>,
 <pysal.cg.shapes.Polygon at 0x7f7e744956d8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495748>,
 <pysal.cg.shapes.Polygon at 0x7f7e744957b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495828>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495898>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495908>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495978>,
 <pysal.cg.shapes.Polygon at 0x7f7e744959e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495a58>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495ac8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495b38>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495ba8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495c18>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495c88>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495cf8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495d68>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495dd8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495e48>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495eb8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495f28>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495f98>,
 <pysal.cg.shapes.Polygon at 0x7f7e74495fd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b60b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6128>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b60f0>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6240>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b62b0>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6320>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6390>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6400>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6470>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b64e0>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6550>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b65c0>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6630>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b66a0>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6710>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6780>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b67f0>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6860>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b68d0>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6940>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b69b0>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6a20>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6a90>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6b00>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6b70>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6be0>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6c50>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6cc0>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6d30>,
```

```
 <pysal.cg.shapes.Polygon at 0x7f7e744b6da0>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6e10>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6e80>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6ef0>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6f60>,
 <pysal.cg.shapes.Polygon at 0x7f7e744b6fd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459080>,
 <pysal.cg.shapes.Polygon at 0x7f7e744590f0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459160>,
 <pysal.cg.shapes.Polygon at 0x7f7e744591d0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459240>,
 <pysal.cg.shapes.Polygon at 0x7f7e744592b0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459320>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459390>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459400>,
 <pysal.cg.shapes.Polygon at 0x7f7e744593c8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459518>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459588>,
 <pysal.cg.shapes.Polygon at 0x7f7e744595f8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459668>,
 <pysal.cg.shapes.Polygon at 0x7f7e744596d8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459748>,
 <pysal.cg.shapes.Polygon at 0x7f7e744597b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459828>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459898>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459908>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459978>,
 <pysal.cg.shapes.Polygon at 0x7f7e744599e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459a58>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459ac8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459b38>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459ba8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459c18>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459c88>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459cf8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459d68>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459dd8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459e48>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459eb8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459f28>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459f98>,
 <pysal.cg.shapes.Polygon at 0x7f7e74459fd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447e0b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447e128>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447e198>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447e208>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447e278>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447e2e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447e358>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447e3c8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447e438>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447e4a8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447e518>,
```

```
 <pysal.cg.shapes.Polygon at 0x7f7e7447e588>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447e5f8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447e668>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447e6d8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447e748>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447e7b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447e828>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447e898>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447e908>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447e978>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447e9e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447ea58>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447eac8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447eb38>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447eba8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447ec18>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447ec88>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447ecf8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447ed68>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447edd8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447ee48>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447eeb8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447ef28>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447ef98>,
 <pysal.cg.shapes.Polygon at 0x7f7e7447efd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b0b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b128>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b198>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b208>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b278>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b2e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b358>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b3c8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b438>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b4a8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b518>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b588>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b5f8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b668>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b6d8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b748>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b7b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b828>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b898>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b908>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b978>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441b9e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441ba58>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441bac8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441bb38>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441bba8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441bc18>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441bc88>,
```

```
 <pysal.cg.shapes.Polygon at 0x7f7e7441bcf8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441bd68>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441bdd8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441be48>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441beb8>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441bf28>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441bf98>,
 <pysal.cg.shapes.Polygon at 0x7f7e7441bfd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e744370b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437128>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437198>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437208>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437278>,
 <pysal.cg.shapes.Polygon at 0x7f7e744372e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437358>,
 <pysal.cg.shapes.Polygon at 0x7f7e744373c8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437438>,
 <pysal.cg.shapes.Polygon at 0x7f7e744374a8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437518>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437588>,
 <pysal.cg.shapes.Polygon at 0x7f7e744375f8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437668>,
 <pysal.cg.shapes.Polygon at 0x7f7e744376d8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437748>,
 <pysal.cg.shapes.Polygon at 0x7f7e744377b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437828>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437898>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437908>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437978>,
 <pysal.cg.shapes.Polygon at 0x7f7e744379e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437a58>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437ac8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437b38>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437ba8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437c18>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437c88>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437cf8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437d68>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437dd8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437e48>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437eb8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437f28>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437f98>,
 <pysal.cg.shapes.Polygon at 0x7f7e74437fd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d80b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8128>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8198>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8208>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8278>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d82e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8358>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d83c8>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8438>,
```

```
 <pysal.cg.shapes.Polygon at 0x7f7e743d84a8>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8518>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8588>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d85f8>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8668>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d86d8>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8748>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d87b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8828>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8898>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8908>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8978>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d89e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8a58>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8ac8>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8b38>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8ba8>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8c18>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8c88>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8cf8>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8d68>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8dd8>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8e48>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8eb8>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8f28>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8f98>,
 <pysal.cg.shapes.Polygon at 0x7f7e743d8fd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f70b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7128>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7198>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7208>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7278>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f72e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7358>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7390>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7320>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7550>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f75c0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7630>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f76a0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7710>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7780>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f77f0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7860>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f78d0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7940>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f79b0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7a20>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7a90>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7b00>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7b70>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7be0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7c50>,
```

```
 <pysal.cg.shapes.Polygon at 0x7f7e743f7cc0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7d30>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7da0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7e10>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7e80>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7ef0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7f60>,
 <pysal.cg.shapes.Polygon at 0x7f7e743f7fd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d080>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d0f0>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d160>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d1d0>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d240>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d2b0>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d320>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d390>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d400>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d470>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d4e0>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d550>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d5c0>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d630>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d6a0>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d710>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d780>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d7f0>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d860>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d8d0>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d940>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438d9b0>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438da20>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438da90>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438db00>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438db70>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438dbe0>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438dc50>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438dcc0>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438dd30>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438dda0>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438de10>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438de80>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438def0>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438df60>,
 <pysal.cg.shapes.Polygon at 0x7f7e7438dfd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aa080>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aa0f0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aa160>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aa1d0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aa240>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aa2b0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aa320>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aa390>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aa400>,
```

```
 <pysal.cg.shapes.Polygon at 0x7f7e743aa470>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aa4e0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aa550>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aa5c0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aa630>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aa6a0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aa710>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aa780>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aa748>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aa898>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aa908>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aa978>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aa9e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aaa58>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aaac8>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aab38>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aaba8>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aac18>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aac88>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aacf8>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aad68>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aadd8>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aae48>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aaeb8>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aaf28>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aaf98>,
 <pysal.cg.shapes.Polygon at 0x7f7e743aafd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743440b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344128>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344198>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344208>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344278>,
 <pysal.cg.shapes.Polygon at 0x7f7e743442e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344358>,
 <pysal.cg.shapes.Polygon at 0x7f7e743443c8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344438>,
 <pysal.cg.shapes.Polygon at 0x7f7e743444a8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344518>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344588>,
 <pysal.cg.shapes.Polygon at 0x7f7e743445f8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344668>,
 <pysal.cg.shapes.Polygon at 0x7f7e743446d8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344748>,
 <pysal.cg.shapes.Polygon at 0x7f7e743447b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344828>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344898>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344908>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344978>,
 <pysal.cg.shapes.Polygon at 0x7f7e743449e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344a58>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344ac8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344b38>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344ba8>,
```

```
 <pysal.cg.shapes.Polygon at 0x7f7e74344c18>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344c88>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344cf8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344d68>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344dd8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344e48>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344eb8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344f28>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344f98>,
 <pysal.cg.shapes.Polygon at 0x7f7e74344fd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743580b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358128>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358198>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358208>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358278>,
 <pysal.cg.shapes.Polygon at 0x7f7e743582e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358358>,
 <pysal.cg.shapes.Polygon at 0x7f7e743583c8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358438>,
 <pysal.cg.shapes.Polygon at 0x7f7e743584a8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358518>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358588>,
 <pysal.cg.shapes.Polygon at 0x7f7e743585f8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358668>,
 <pysal.cg.shapes.Polygon at 0x7f7e743586d8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358748>,
 <pysal.cg.shapes.Polygon at 0x7f7e743587b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358828>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358898>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358908>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358978>,
 <pysal.cg.shapes.Polygon at 0x7f7e743589e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358a58>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358ac8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358b38>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358ba8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358c18>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358c88>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358cf8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358d68>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358dd8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358e48>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358eb8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358f28>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358f98>,
 <pysal.cg.shapes.Polygon at 0x7f7e74358fd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743770b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377128>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377198>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377208>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377278>,
 <pysal.cg.shapes.Polygon at 0x7f7e743772e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377358>,
```

```
 <pysal.cg.shapes.Polygon at 0x7f7e743773c8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377438>,
 <pysal.cg.shapes.Polygon at 0x7f7e743774a8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377518>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377588>,
 <pysal.cg.shapes.Polygon at 0x7f7e743775f8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377668>,
 <pysal.cg.shapes.Polygon at 0x7f7e743776d8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377748>,
 <pysal.cg.shapes.Polygon at 0x7f7e743777b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377828>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377898>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377908>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377978>,
 <pysal.cg.shapes.Polygon at 0x7f7e743779e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377a58>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377ac8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377b38>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377ba8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377c18>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377c88>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377cf8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377d68>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377dd8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377e48>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377eb8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377f28>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377f98>,
 <pysal.cg.shapes.Polygon at 0x7f7e74377fd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743120b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312128>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312198>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312208>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312278>,
 <pysal.cg.shapes.Polygon at 0x7f7e743122e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312358>,
 <pysal.cg.shapes.Polygon at 0x7f7e743123c8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312438>,
 <pysal.cg.shapes.Polygon at 0x7f7e743124a8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312518>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312588>,
 <pysal.cg.shapes.Polygon at 0x7f7e743125f8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312668>,
 <pysal.cg.shapes.Polygon at 0x7f7e743126d8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312748>,
 <pysal.cg.shapes.Polygon at 0x7f7e743127b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312828>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312898>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312908>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312978>,
 <pysal.cg.shapes.Polygon at 0x7f7e743129e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312a58>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312ac8>,
```

```
 <pysal.cg.shapes.Polygon at 0x7f7e74312b38>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312ba8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312c18>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312c88>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312cf8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312d68>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312dd8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312e48>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312eb8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312f28>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312f98>,
 <pysal.cg.shapes.Polygon at 0x7f7e74312fd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e743280b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328128>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328198>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328208>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328278>,
 <pysal.cg.shapes.Polygon at 0x7f7e743282e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328358>,
 <pysal.cg.shapes.Polygon at 0x7f7e743283c8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328438>,
 <pysal.cg.shapes.Polygon at 0x7f7e743284a8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328518>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328588>,
 <pysal.cg.shapes.Polygon at 0x7f7e743285f8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328668>,
 <pysal.cg.shapes.Polygon at 0x7f7e743286d8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328748>,
 <pysal.cg.shapes.Polygon at 0x7f7e743287b8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328828>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328898>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328908>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328978>,
 <pysal.cg.shapes.Polygon at 0x7f7e743289e8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328a58>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328ac8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328b38>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328ba8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328c18>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328c88>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328cf8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328d68>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328dd8>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328da0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328ef0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328f60>,
 <pysal.cg.shapes.Polygon at 0x7f7e74328fd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5080>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c50f0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5160>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c51d0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5240>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c52b0>,
```

```
 <pysal.cg.shapes.Polygon at 0x7f7e742c5320>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5390>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5400>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5470>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c54e0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5550>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c55c0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5630>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c56a0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5710>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5780>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c57f0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5860>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c58d0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5940>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c59b0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5a20>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5a90>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5b00>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5b70>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5be0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5c50>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5cc0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5d30>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5da0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5e10>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5e80>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5ef0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5f60>,
 <pysal.cg.shapes.Polygon at 0x7f7e742c5fd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0080>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e00f0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0160>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e01d0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0240>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e02b0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0320>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0390>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0400>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0470>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e04e0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0550>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e05c0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0630>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e06a0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0710>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0780>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e07f0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0860>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e08d0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0940>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e09b0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0a20>,
```

```
 <pysal.cg.shapes.Polygon at 0x7f7e742e0a90>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0b00>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0b70>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0be0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0c50>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0cc0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0d30>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0da0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0e10>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0e80>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0ef0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0f60>,
 <pysal.cg.shapes.Polygon at 0x7f7e742e0fd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282048>,
 <pysal.cg.shapes.Polygon at 0x7f7e742820f0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282160>,
 <pysal.cg.shapes.Polygon at 0x7f7e742821d0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282240>,
 <pysal.cg.shapes.Polygon at 0x7f7e742822b0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282320>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282390>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282400>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282470>,
 <pysal.cg.shapes.Polygon at 0x7f7e742824e0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282550>,
 <pysal.cg.shapes.Polygon at 0x7f7e742825c0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282630>,
 <pysal.cg.shapes.Polygon at 0x7f7e742826a0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282710>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282780>,
 <pysal.cg.shapes.Polygon at 0x7f7e742827f0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282860>,
 <pysal.cg.shapes.Polygon at 0x7f7e742828d0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282940>,
 <pysal.cg.shapes.Polygon at 0x7f7e742829b0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282a20>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282a90>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282b00>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282b70>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282be0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282c50>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282cc0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282d30>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282da0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282e10>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282e80>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282ef0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282f60>,
 <pysal.cg.shapes.Polygon at 0x7f7e74282fd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294048>,
 <pysal.cg.shapes.Polygon at 0x7f7e742940f0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294160>,
 <pysal.cg.shapes.Polygon at 0x7f7e742941d0>,
```

```
 <pysal.cg.shapes.Polygon at 0x7f7e74294240>,
 <pysal.cg.shapes.Polygon at 0x7f7e742942b0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294320>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294390>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294400>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294470>,
 <pysal.cg.shapes.Polygon at 0x7f7e742944e0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294550>,
 <pysal.cg.shapes.Polygon at 0x7f7e742945c0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294630>,
 <pysal.cg.shapes.Polygon at 0x7f7e742946a0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294710>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294780>,
 <pysal.cg.shapes.Polygon at 0x7f7e742947f0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294860>,
 <pysal.cg.shapes.Polygon at 0x7f7e742948d0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294940>,
 <pysal.cg.shapes.Polygon at 0x7f7e742949b0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294a20>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294a90>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294b00>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294b70>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294be0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294c50>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294cc0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294d30>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294da0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294e10>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294e80>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294ef0>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294f60>,
 <pysal.cg.shapes.Polygon at 0x7f7e74294fd0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742b9080>,
 <pysal.cg.shapes.Polygon at 0x7f7e742b90f0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742b9160>,
 <pysal.cg.shapes.Polygon at 0x7f7e742b91d0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742b9240>,
 <pysal.cg.shapes.Polygon at 0x7f7e742b92b0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742b9320>,
 <pysal.cg.shapes.Polygon at 0x7f7e742b9390>,
 <pysal.cg.shapes.Polygon at 0x7f7e742b9400>,
 <pysal.cg.shapes.Polygon at 0x7f7e742b9470>,
 <pysal.cg.shapes.Polygon at 0x7f7e742b94e0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742b9550>,
 <pysal.cg.shapes.Polygon at 0x7f7e742b95c0>,
 <pysal.cg.shapes.Polygon at 0x7f7e742b9630>,
 <pysal.cg.shapes.Polygon at 0x7f7e742b96a0>,
 ...]
```

```
all_polygons[0].centroid #the centroid of the first polygon
```

```
(-94.90336786329912, 48.771730563701574)
```

```
all_polygons[0].area
```

```
0.565411079543992
```

```
all_polygons[0].perimeter
```

```
4.055313773836516
```

While in the Jupyter Notebook, you can examine what properties an object has by using the tab key.

```
polygon = all_polygons[0]
```

```
polygon. #press tab when the cursor is right after the dot
```

```
  File "<ipython-input-16-aa03438a2fa8>", line 1
    polygon. #press tab when the cursor is right after the dot
                                                 ^
SyntaxError: invalid syntax
```

Working with Data Tables

When you're working with tables of data, like a `csv` or `dbf`, you can extract your data in the following way. Let's open the dbf file we got the path for above.

```
f = ps.open(dbf_path)
```

Just like with the shapefile, we can examine the header of the dbf file

```
f.header
```

```
['NAME',
 'STATE_NAME',
 'STATE_FIPS',
 'CNTY_FIPS',
 'FIPS',
 'STFIPS',
 'COFIPS',
 'FIPSNO',
 'SOUTH',
```

```
 'HR60',
 'HR70',
 'HR80',
 'HR90',
 'HC60',
 'HC70',
 'HC80',
 'HC90',
 'PO60',
 'PO70',
 'PO80',
 'PO90',
 'RD60',
 'RD70',
 'RD80',
 'RD90',
 'PS60',
 'PS70',
 'PS80',
 'PS90',
 'UE60',
 'UE70',
 'UE80',
 'UE90',
 'DV60',
 'DV70',
 'DV80',
 'DV90',
 'MA60',
 'MA70',
 'MA80',
 'MA90',
 'POL60',
 'POL70',
 'POL80',
 'POL90',
 'DNL60',
 'DNL70',
 'DNL80',
 'DNL90',
 'MFIL59',
 'MFIL69',
 'MFIL79',
 'MFIL89',
 'FP59',
 'FP69',
 'FP79',
 'FP89',
 'BLK60',
 'BLK70',
 'BLK80',
 'BLK90',
 'GI59',
```

```
 'GI69',
 'GI79',
 'GI89',
 'FH60',
 'FH70',
 'FH80',
 'FH90']
```

So, the header is a list containing the names of all of the fields we can read. If we were interested in getting the `['NAME', 'STATE_NAME', 'HR90', 'HR80']` fields.

If we just wanted to grab the data of interest, `HR90`, we can use either `by_col` or `by_col_array`, depending on the format we want the resulting data in:

```
HR90 = f.by_col('HR90')
print(type(HR90).__name__, HR90[0:5])
HR90 = f.by_col_array('HR90')
print(type(HR90).__name__, HR90[0:5])
```

```
list [0.0, 15.885623511, 6.4624531472, 6.9965017491, 7.4780332772]
ndarray [[  0.        ]
 [ 15.88562351]
 [  6.46245315]
 [  6.99650175]
 [  7.47803328]]
```

As you can see, the `by_col` function returns a list of data, with no shape. It can only return one column at a time:

```
HRs = f.by_col('HR90', 'HR80')
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-20-1fef6a3c3a50> in <module>()
----> 1 HRs = f.by_col('HR90', 'HR80')


TypeError: __call__() takes 2 positional arguments but 3 were given
```

This error message is called a "traceback," as you see in the top right, and it usually provides feedback on why the previous command did not execute correctly. Here, you see that one-too-many arguments was provided to `__call__`, which tells us we cannot pass as many arguments as we did to `by_col`.

If you want to read in many columns at once and store them to an array, use `by_col_array`:

```
HRs = f.by_col_array('HR90', 'HR80')
```

```
HRs
```

```
array([[  0.       ,   8.85582713],
       [ 15.88562351,  17.20874204],
       [  6.46245315,   3.4507747 ],
       ...,
       [  4.36732988,   5.2803488 ],
       [  3.72771194,   3.00003   ],
       [  2.04885495,   1.19474313]])
```

It is best to use `by_col_array` on data of a single type. That is, if you read in a lot of columns, some of them numbers and some of them strings, all columns will get converted to the same datatype:

```
allcolumns = f.by_col_array(['NAME', 'STATE_NAME', 'HR90', 'HR80'])
```

```
allcolumns
```

```
array([['Lake of the Woods', 'Minnesota', '0.0', '8.8558271343'],
       ['Ferry', 'Washington', '15.885623511', '17.208742041'],
       ['Stevens', 'Washington', '6.4624531472', '3.4507746989'],
       ...,
       ['York', 'Virginia', '4.3673298769', '5.2803488048'],
       ['Prince William', 'Virginia', '3.7277119437', '3.0000300003'],
       ['Gallatin', 'Montana', '2.0488549462', '1.1947431302']],
      dtype='<U20')
```

Note that the numerical columns, `HR90` & `HR80` are now considered strings, since they show up with the single tickmarks around them, like `'0.0'`.

These methods work similarly for `.csv` files as well

Using Pandas with PySAL

A new functionality added to PySAL recently allows you to work with shapefile/dbf pairs using Pandas. This *optional* extension is only turned on if you have Pandas installed. The extension is the `ps.pdio` module:

```
ps.pdio
```

```
<module 'pysal.contrib.pdutilities' from '/home/serge/anaconda2/envs/scipy16/lib/python3.5/site-packa
ges/pysal/contrib/pdutilities/__init__.py'>
```

To use it, you can read in shapefile/dbf pairs using the `ps.pdio.read_files` command.

```
data_table = ps.pdio.read_files(shp_path)
```

This reads in *the entire database table* and adds a column to the end, called `geometry`, that stores the geometries read in from the shapefile.

Now, you can work with it like a standard pandas dataframe.

```
data_table.head()
```

	NAME	STATE_NAME	STATE_FIPS	CNTY_FIPS	FIPS	STFIPS
0	Lake of the Woods	Minnesota	27	077	27077	27
1	Ferry	Washington	53	019	53019	53
2	Stevens	Washington	53	065	53065	53
3	Okanogan	Washington	53	047	53047	53
4	Pend Oreille	Washington	53	051	53051	53

5 rows × 70 columns

The `read_files` function only works on shapefile/dbf pairs. If you need to read in data using CSVs, use pandas directly:

```
usjoin = pd.read_csv(csv_path)
#usjoin = ps.pdio.read_files(csv_path) #will not work, not a shp/dbf pair
```

```
usjoin.head()
```

	Name	STATE_FIPS	1929	1930	1931	1932	1933	1934
0	Alabama	1	323	267	224	162	166	211
1	Arizona	4	600	520	429	321	308	362
2	Arkansas	5	310	228	215	157	157	187
3	California	6	991	887	749	580	546	603
4	Colorado	8	634	578	471	354	353	368

5 rows × 83 columns

The nice thing about working with pandas dataframes is that they have very powerful baked-in support for relational-style queries. By this, I mean that it is very easy to find things like:

The number of counties in each state:

```
data_table.groupby("STATE_NAME").size()
```

```
STATE_NAME
Alabama                67
Arizona                14
Arkansas               75
California             58
Colorado               63
Connecticut             8
Delaware                3
District of Columbia    1
Florida                67
Georgia               159
Idaho                  44
Illinois              102
Indiana                92
Iowa                   99
Kansas                105
Kentucky              120
Louisiana              64
Maine                  16
Maryland               24
Massachusetts          12
Michigan               83
Minnesota              87
Mississippi            82
Missouri              115
Montana                55
Nebraska               93
Nevada                 17
New Hampshire          10
New Jersey             21
New Mexico             32
New York               58
North Carolina        100
North Dakota           53
Ohio                   88
Oklahoma               77
Oregon                 36
Pennsylvania           67
Rhode Island            5
South Carolina         46
South Dakota           66
Tennessee              95
Texas                 254
Utah                   29
Vermont                14
Virginia              123
Washington             38
West Virginia          55
Wisconsin              70
Wyoming                23
dtype: int64
```

Or, to get the rows of the table that are in Arizona, we can use the `query` function of the dataframe:

```
data_table.query('STATE_NAME == "Arizona"')
```

	NAME	STATE_NAME	STATE_FIPS	CNTY_FIPS	FIPS	STF
1707	Navajo	Arizona	04	017	04017	4
1708	Coconino	Arizona	04	005	04005	4
1722	Mohave	Arizona	04	015	04015	4
1726	Apache	Arizona	04	001	04001	4
2002	Yavapai	Arizona	04	025	04025	4
2182	Gila	Arizona	04	007	04007	4
2262	Maricopa	Arizona	04	013	04013	4
2311	Greenlee	Arizona	04	011	04011	4
2326	Graham	Arizona	04	009	04009	4
2353	Pinal	Arizona	04	021	04021	4
2499	Pima	Arizona	04	019	04019	4
2514	Cochise	Arizona	04	003	04003	4
2615	Santa Cruz	Arizona	04	023	04023	4
3080	La Paz	Arizona	04	012	04012	4

14 rows × 70 columns

Behind the scenes, this uses a fast vectorized library, `numexpr`, to essentially do the following.

First, compare each row's `STATE_NAME` column to `'Arizona'` and return `True` if the row matches:

```
data_table.STATE_NAME == 'Arizona'
```

```
0      False
```

```
1      False
2      False
3      False
4      False
5      False
6      False
7      False
8      False
9      False
10     False
11     False
12     False
13     False
14     False
15     False
16     False
17     False
18     False
19     False
20     False
21     False
22     False
23     False
24     False
25     False
26     False
27     False
28     False
29     False
       ...
3055   False
3056   False
3057   False
3058   False
3059   False
3060   False
3061   False
3062   False
3063   False
3064   False
3065   False
3066   False
3067   False
3068   False
3069   False
3070   False
3071   False
3072   False
3073   False
3074   False
3075   False
3076   False
3077   False
```

```
3078    False
3079    False
3080     True
3081    False
3082    False
3083    False
3084    False
Name: STATE_NAME, dtype: bool
```

Then, use that to filter out rows where the condition is true:

```
data_table[data_table.STATE_NAME == 'Arizona']
```

	NAME	STATE_NAME	STATE_FIPS	CNTY_FIPS	FIPS	STFI
1707	Navajo	Arizona	04	017	04017	4
1708	Coconino	Arizona	04	005	04005	4
1722	Mohave	Arizona	04	015	04015	4
1726	Apache	Arizona	04	001	04001	4
2002	Yavapai	Arizona	04	025	04025	4
2182	Gila	Arizona	04	007	04007	4
2262	Maricopa	Arizona	04	013	04013	4
2311	Greenlee	Arizona	04	011	04011	4
2326	Graham	Arizona	04	009	04009	4
2353	Pinal	Arizona	04	021	04021	4
2499	Pima	Arizona	04	019	04019	4
2514	Cochise	Arizona	04	003	04003	4
2615	Santa Cruz	Arizona	04	023	04023	4
3080	La Paz	Arizona	04	012	04012	4

14 rows × 70 columns

We might need this behind the scenes knowledge when we want to chain together conditions, or when we need to do spatial queries.

This is because spatial queries are somewhat more complex. Let's say, for example, we want all of the counties in the US to the West of `-121` longitude. We need a way to express that question. Ideally, we want something like:

```
SELECT
      *
FROM
      data_table
WHERE
      x_centroid < -121
```

So, let's refer to an arbitrary polygon in the the dataframe's geometry column as `poly`. The centroid of a PySAL polygon is stored as an `(X,Y)` pair, so the longidtude is the first element of the pair, `poly.centroid[0]`.

Then, applying this condition to each geometry, we get the same kind of filter we used above to grab only counties in Arizona:

```
data_table.geometry.apply(lambda x: x.centroid[0] < -121)
```

```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
9     False
10    False
11    False
12    False
13    False
14    False
15    False
16    False
17    False
18    False
19    False
20    False
21    False
22    False
23    False
24    False
25    False
26    False
```

```
27     True
28    False
29    False
      ...
3055  False
3056  False
3057  False
3058  False
3059  False
3060  False
3061  False
3062  False
3063  False
3064  False
3065  False
3066  False
3067  False
3068  False
3069  False
3070  False
3071  False
3072  False
3073  False
3074  False
3075  False
3076  False
3077  False
3078  False
3079  False
3080  False
3081  False
3082  False
3083  False
3084  False
Name: geometry, dtype: bool
```

If we use this as a filter on the table, we can get only the rows that match that condition, just like we did for the `STATE_NAME` query:

```
data_table[data_table.geometry.apply(lambda x: x.centroid[0] < -121)]
```

	NAME	STATE_NAME	STATE_FIPS	CNTY_FIPS	FIPS	ST
27	Whatcom	Washington	53	073	53073	53
31	Skagit	Washington	53	057	53057	53
44	Clallam	Washington	53	009	53009	53
47	Snohomish	Washington	53	061	53061	53

48	Island	Washington	53	029	53029	53
57	Jefferson	Washington	53	031	53031	53
71	Kitsap	Washington	53	035	53035	53
80	King	Washington	53	033	53033	53
85	Mason	Washington	53	045	53045	53
92	Grays Harbor	Washington	53	027	53027	53
107	Pierce	Washington	53	053	53053	53
116	Thurston	Washington	53	067	53067	53
130	Pacific	Washington	53	049	53049	53
131	Lewis	Washington	53	041	53041	53
169	Skamania	Washington	53	059	53059	53
170	Cowlitz	Washington	53	015	53015	53
171	Wahkiakum	Washington	53	069	53069	53
181	Clatsop	Oregon	41	007	41007	41
183	Columbia	Oregon	41	009	41009	41
193	Clark	Washington	53	011	53011	53
216	Tillamook	Oregon	41	057	41057	41
217	Washington	Oregon	41	067	41067	41
224	Multnomah	Oregon	41	051	41051	41
225	Hood River	Oregon	41	027	41027	41
226	Wasco	Oregon	41	065	41065	41
240	Clackamas	Oregon	41	005	41005	41

253	Marion	Oregon	41	047	41047	41
269	Polk	Oregon	41	053	41053	41
272	Lincoln	Oregon	41	041	41041	41
...	...	...	...	...	...	...
493	Curry	Oregon	41	015	41015	41
517	Josephine	Oregon	41	033	41033	41
594	Siskiyou	California	06	093	06093	6
601	Del Norte	California	06	015	06015	6
687	Humboldt	California	06	023	06023	6
709	Trinity	California	06	105	06105	6
735	Shasta	California	06	089	06089	6
907	Tehama	California	06	103	06103	6
974	Butte	California	06	007	06007	6
1005	Mendocino	California	06	045	06045	6
1047	Glenn	California	06	021	06021	6
1090	Yuba	California	06	115	06115	6
1101	Lake	California	06	033	06033	6
1141	Colusa	California	06	011	06011	6
1172	Sutter	California	06	101	06101	6
1262	Yolo	California	06	113	06113	6
1281	Napa	California	06	055	06055	6
1282	Sonoma	California	06	097	06097	6

1301	Sacramento	California	06	067	06067	6
1401	Marin	California	06	041	06041	6
1404	San Joaquin	California	06	077	06077	6
1447	Solano	California	06	095	06095	6
1462	Contra Costa	California	06	013	06013	6
1514	Alameda	California	06	001	06001	6
1530	San Francisco	California	06	075	06075	6
1559	San Mateo	California	06	081	06081	6
1609	Santa Clara	California	06	085	06085	6
1664	Santa Cruz	California	06	087	06087	6
1731	San Benito	California	06	069	06069	6
1745	Monterey	California	06	053	06053	6

69 rows × 70 columns

This works on any type of spatial query.

For instance, if we wanted to find all of the counties that are within a threshold distance from an observation's centroid, we can do it in the following way.

First, specify the observation. Here, we'll use Cook County, IL:

```
data_table.query('(NAME == "Cook") & (STATE_NAME == "Illinois")')
```

	NAME	STATE_NAME	STATE_FIPS	CNTY_FIPS	FIPS	STFIPS
3044	Cook	Illinois	17	031	17031	17

1 rows × 70 columns

```
geom = data_table.query('(NAME == "Cook") & (STATE_NAME == "Illinois")').geometry
```

```
geom.values[0].centroid
```

```
(-87.82107391263027, 41.84346628270174)
```

```
cook_county_centroid = geom.values[0].centroid
```

```
import scipy.spatial.distance as d
def near_target_point(polygon, target=cook_county_centroid, threshold=1):
    return d.euclidean(polygon.centroid, target) < threshold
```

```
data_table[data_table.geometry.apply(near_target_point)]
```

	NAME	STATE_NAME	STATE_FIPS	CNTY_FIPS	FIPS	STF
631	Will	Illinois	17	197	17197	17
633	Kendall	Illinois	17	093	17093	17
634	Lake	Indiana	18	089	18089	18
635	Porter	Indiana	18	127	18127	18
686	Grundy	Illinois	17	063	17063	17
718	Kankakee	Illinois	17	091	17091	17
731	Newton	Indiana	18	111	18111	18
3010	Racine	Wisconsin	55	101	55101	55
3019	Kenosha	Wisconsin	55	059	55059	55
3028	McHenry	Illinois	17	111	17111	17
3030	Lake	Illinois	17	097	17097	17
3044	Cook	Illinois	17	031	17031	17
3045	Kane	Illinois	17	089	17089	17
3046	De Kalb	Illinois	17	037	17037	17
3059	Du Page	Illinois	17	043	17043	17

15 rows × 70 columns

Moving in and out of the dataframe

Most things in PySAL will be explicit about what type their input should be. Most of the time, PySAL functions require either lists or arrays. This is why the file-handler methods are the default IO method in PySAL: the rest of the computational tools are built around their datatypes.

However, it is very easy to get the correct datatype from Pandas using the `values` and `tolist` commands.

`tolist()` will convert its entries to a list. But, it can only be called on individual columns (called `Series` in `pandas` documentation)

So, to turn the `NAME` column into a list:

```
data_table.NAME.tolist()
```

```
['Lake of the Woods',
 'Ferry',
 'Stevens',
 'Okanogan',
 'Pend Oreille',
 'Boundary',
 'Lincoln',
 'Flathead',
 'Glacier',
 'Toole',
 'Liberty',
 'Hill',
 'Sheridan',
 'Divide',
 'Burke',
 'Renville',
 'Bottineau',
 'Rolette',
 'Towner',
 'Cavalier',
 'Pembina',
 'Kittson',
 'Roseau',
 'Blaine',
 'Phillips',
 'Valley',
 'Daniels',
 'Whatcom',
 'Bonner',
 'Ward',
 'Koochiching',
 'Skagit',
 'Williams',
 'McHenry',
```

```
'St. Louis',
'Roosevelt',
'Mountrial',
'Marshall',
'Ramsey',
'Walsh',
'Beltrami',
'Pierce',
'Chelan',
'Pondera',
'Clallam',
'Benson',
'Chouteau',
'Snohomish',
'Island',
'Sanders',
'Lake',
'Nelson',
'Grand Forks',
'Polk',
'Pennington',
'Douglas',
'McKenzie',
'Jefferson',
'Richland',
'Teton',
'McCone',
'Shoshone',
'Spokane',
'Lake',
'Clearwater',
'Kootenai',
'Garfield',
'Red Lake',
'Grant',
'Lincoln',
'Lewis and Clark',
'Kitsap',
'Itasca',
'Sheridan',
'Wells',
'McLean',
'Eddy',
'Dunn',
'Fergus',
'Dawson',
'King',
'Cascade',
'Griggs',
'Steele',
'Traill',
'Mason',
'Missoula',
```

```
'Petroleum',
'Powell',
'Kittitas',
'Foster',
'Mercer',
'Grays Harbor',
'Norman',
'Mahnomen',
'Mineral',
'Cass',
'Aroostook',
'Judith Basin',
'Hubbard',
'Benewah',
'Wibaux',
'Golden Valley',
'Billings',
'Stutsman',
'Kidder',
'Burleigh',
'Pierce',
'Oliver',
'Adams',
'Whitman',
'Barnes',
'Cass',
'Prairie',
'Becker',
'Clay',
'Thurston',
'Latah',
'Meagher',
'Yakima',
'Aitkin',
'Stark',
'Morton',
'Bayfield',
'Clearwater',
'Custer',
'Rosebud',
'Granite',
'Wadena',
'Crow Wing',
'Pacific',
'Lewis',
'Broadwater',
'Carlton',
'Golden Valley',
'Douglas',
'Musselshell',
'Wheatland',
'Franklin',
'Benton',
```

```
'Grant',
'Otter Tail',
'Garfield',
'Fallon',
'Idaho',
'Ravalli',
'Ashland',
'Logan',
'Emmons',
'La Moure',
'Slope',
'Hettinger',
'Ransom',
'Richland',
'Wilkin',
'Nez Perce',
'Columbia',
'Walla Walla',
'Iron',
'Somerset',
'Piscataquis',
'Jefferson',
'Yellowstone',
'Treasure',
'Lewis',
'Asotin',
'Sioux',
'Pine',
'Penobscot',
'Skamania',
'Cowlitz',
'Wahkiakum',
'Todd',
'Morrison',
'McIntosh',
'Dickey',
'Sargent',
'Bowman',
'Adams',
'Deer Lodge',
'Mille Lacs',
'Clatsop',
'Sweet Grass',
'Columbia',
'Silver Bow',
'Washburn',
'Sawyer',
'Burnett',
'Kanabec',
'Carter',
'Stillwater',
'Grant',
'Douglas',
```

```
'Clark',
'Klickitat',
'Big Horn',
'Traverse',
'Umatilla',
'Wallowa',
'Price',
'Campbell',
'Harding',
'McPherson',
'Perkins',
'Corson',
'Brown',
'Beaverhead',
'Marshall',
'Roberts',
'Morrow',
'Union',
'Madison',
'Benton',
'Gilliam',
'Powder River',
'Stearns',
'Tillamook',
'Washington',
'Pope',
'Stevens',
'Isanti',
'Chisago',
'Sherman',
'Polk',
'Multnomah',
'Hood River',
'Wasco',
'Lemhi',
'Washington',
'Franklin',
'Barron',
'Rusk',
'Carbon',
'Walworth',
'Edmunds',
'Day',
'Big Stone',
'Sherburne',
'Ziebach',
'Dewey',
'Clackamas',
'Wright',
'Yamhill',
'Anoka',
'Kandiyohi',
'Swift',
```

```
 'Taylor',
 'Oxford',
 'Grant',
 'Meeker',
 'Coos',
 'Washington',
 'Chippewa',
 'Marion',
 'Lac Qui Parle',
 'Adams',
 'Potter',
 'Faulk',
 'Hennepin',
 'Spink',
 'St. Croix',
 'Dunn',
 'Butte',
 'Valley',
 'Clark',
 'Codington',
 'Chippewa',
 'Ramsey',
 'Baker',
 'Polk',
 'Wheeler',
 'Meade',
 'Lincoln',
 'Clark',
 'Essex',
 'Grand Isle',
 'Franklin',
 'Orleans',
 'Clinton',
 'Park',
 'Crook',
 'Big Horn',
 'Campbell',
 'Sheridan',
 'Franklin',
 'Grant',
 'McLeod',
 'Carver',
 'Deuel',
 'Dakota',
 'Yellow Medicine',
 'Sully',
 'Hyde',
 'Hand',
 'Renville',
 'Pierce',
 'Custer',
 'Eau Claire',
 'Washington',
```

```
'Jefferson',
'Scott',
'Hamlin',
'Lamoille',
'Linn',
'Stanley',
'Caledonia',
'Waldo',
'Haakon',
'Fremont',
'Sibley',
'Chittenden',
'Kennebec',
'Goodhue',
'Benton',
'Redwood',
'Wood',
'Pepin',
'Teton',
'Beadle',
'Lyon',
'Lincoln',
'Lawrence',
'Buffalo',
'Trempealeau',
'Jackson',
'Clark',
'Crook',
'Johnson',
'Hughes',
'Essex',
'Le Sueur',
'Rice',
'Kingsbury',
'Brookings',
'Pennington',
'Gem',
'Brown',
'Washington',
'Androscoggin',
'Nicollet',
'Wabasha',
'Malheur',
'Hancock',
'Grafton',
'Deschutes',
'Knox',
'Boise',
'Lincoln',
'Addison',
'Carroll',
'Lane',
'Blue Earth',
```

```
'Juneau',
'Butte',
'Lyman',
'Orange',
'Waseca',
'Buffalo',
'Jerauld',
'Moody',
'Pipestone',
'Dodge',
'Sanborn',
'Murray',
'Cottonwood',
'Steele',
'Olmsted',
'Miner',
'Lake',
'Winona',
'Weston',
'Jones',
'Cumberland',
'Washakie',
'Monroe',
'Payette',
'Hamilton',
'Watonwan',
'Herkimer',
'La Crosse',
'Elmore',
'Hot Springs',
'Jefferson',
'Harney',
'Sagadahoc',
'Fremont',
'Jackson',
'Blaine',
'Teton',
'Windsor',
'Douglas',
'Aurora',
'Brule',
'Madison',
'Canyon',
'Mellette',
'Camas',
'Custer',
'Rutland',
'Faribault',
'Minnehaha',
'Rock',
'Freeborn',
'Nobles',
'Jackson',
```

```
'Martin',
'Houston',
'Mower',
'Fillmore',
'Davison',
'Hanson',
'McCook',
'York',
'Washington',
'Ada',
'Warren',
'Tripp',
'Belknap',
'Vernon',
'Shannon',
'Owyhee',
'Bonneville',
'Bingham',
'Klamath',
'Lake',
'Coos',
'Merrimack',
'Sullivan',
'Strafford',
'Richland',
'Natrona',
'Gregory',
'Niobrara',
'Charles Mix',
'Turner',
'Lincoln',
'Worth',
'Mitchell',
'Allamakee',
'Winnebago',
'Winneshiek',
'Converse',
'Osceola',
'Dickinson',
'Kossuth',
'Howard',
'Emmet',
'Lyon',
'Douglas',
'Hutchinson',
'Fall River',
'Sublette',
'Crawford',
'Saratoga',
'Bennett',
'Todd',
'Bennington',
'Lincoln',
```

```
'Fulton',
'Rockingham',
'Sioux',
'Windham',
"O'Brien",
'Cerro Gordo',
'Clay',
'Hancock',
'Palo Alto',
'Floyd',
'Chickasaw',
'Iowa',
'Grant',
'Hillsborough',
'Lincoln',
'Gooding',
'Minidoka',
'Cheshire',
'Yankton',
'Bon Homme',
'Power',
'Union',
'Clay',
'Fayette',
'Clayton',
'Montgomery',
'Caribou',
'Bannock',
'Sioux',
'Dawes',
'Sheridan',
'Jackson',
'Keya Paha',
'Boyd',
'Cherry',
'Curry',
'Rensselaer',
'Schenectady',
'Plymouth',
'Cherokee',
'Bremer',
'Butler',
'Buena Vista',
'Twin Falls',
'Pocahontas',
'Humboldt',
'Wright',
'Franklin',
'Otsego',
'Holt',
'Essex',
'Knox',
'Cedar',
```

```
'Jerome',
'Schoharie',
'Brown',
'Lafayette',
'Albany',
'Rock',
'Josephine',
'Dixon',
'Berkshire',
'Franklin',
'Middlesex',
'Worcester',
'Dubuque',
'Cassia',
'Webster',
'Delaware',
'Buchanan',
'Black Hawk',
'Goshen',
'Platte',
'Bear Lake',
'Woodbury',
'Ida',
'Sac',
'Calhoun',
'Hamilton',
'Hampshire',
'Hardin',
'Grundy',
'Dakota',
'Delaware',
'Jo Daviess',
'Columbia',
'Oneida',
'Greene',
'Suffolk',
'Carbon',
'Pierce',
'Box Butte',
'Antelope',
'Albany',
'Franklin',
'Jackson',
'Wayne',
'Hampden',
'Jones',
'Benton',
'Linn',
'Tama',
'Thurston',
'Sweetwater',
'Plymouth',
'Norfolk',
```

```
'Monona',
'Crawford',
'Carroll',
'Greene',
'Boone',
'Marshall',
'Story',
'Ulster',
'Cuming',
'Bristol',
'Stanton',
'Madison',
'Grant',
'Loup',
'Hooker',
'Garfield',
'Thomas',
'Wheeler',
'Blaine',
'Dutchess',
'Barnstable',
'Burt',
'Litchfield',
'Hartford',
'Clinton',
'Tolland',
'Windham',
'Sullivan',
'Providence',
'Cache',
'Siskiyou',
'Garden',
'Box Elder',
'Rich',
'Scotts Bluff',
'Morrill',
'Wayne',
'Del Norte',
'Humboldt',
'Elko',
'Modoc',
'Washoe',
'Cedar',
'Boone',
'Jasper',
'Polk',
'Poweshiek',
'Harrison',
'Guthrie',
'Shelby',
'Audubon',
'Iowa',
'Dallas',
```

```
'Johnson',
'Rock Island',
'Scott',
'Bristol',
'Kent',
'Colfax',
'Dodge',
'Platte',
'Arthur',
'Greeley',
'McPherson',
'Logan',
'Custer',
'Valley',
'Will',
'Lucas',
'Kendall',
'Lake',
'Porter',
'Fulton',
'Geauga',
'New London',
'Williams',
'Banner',
'Washington',
'Newport',
'Fairfield',
'Laramie',
'Wyoming',
'Washington',
'Middlesex',
'New Haven',
'Lackawanna',
'La Salle',
'Orange',
'Elk',
'Cuyahoga',
'Venango',
'Forest',
'Ottawa',
'Cameron',
'Wood',
'Pike',
'Lycoming',
'Muscatine',
'Sullivan',
'Bureau',
'Henry',
'Uinta',
'Noble',
'De Kalb',
'Putnam',
'Nance',
```

```
'Lorain',
'Mahaska',
'Pottawattamie',
'Washington',
'Marion',
'Madison',
'Warren',
'Keokuk',
'Cass',
'Adair',
'Sandusky',
'Trumbull',
'Mercer',
'Marshall',
'Henry',
'Clinton',
'Grundy',
'Humboldt',
'Butler',
'Saunders',
'Erie',
'Kosciusko',
'Starke',
'Clarion',
'Cheyenne',
'Defiance',
'Weber',
'Louisa',
'Luzerne',
'Polk',
'Douglas',
'Sherman',
'Howard',
'Lincoln',
'Merrick',
'Keith',
'Kimball',
'Jefferson',
'Morgan',
'Trinity',
'Westchester',
'Sussex',
'Summit',
'Portage',
'Mercer',
'Rockland',
'Putnam',
'Columbia',
'Kankakee',
'Whitley',
'Jasper',
'Huron',
'Allen',
```

```
'Medina',
'Summit',
'Centre',
'Clearfield',
'Monroe',
'Seneca',
'Paulding',
'Stark',
'Newton',
'Deuel',
'Passaic',
'Sarpy',
'Shasta',
'Lassen',
'Northumberland',
'Fulton',
'Butler',
'Armstrong',
'Pulaski',
'Montour',
'Hancock',
'Mills',
'Putnam',
'Montgomery',
'Adams',
'Clarke',
'Wapello',
'Jefferson',
'Union',
'Henry',
'Hamilton',
'Lucas',
'Monroe',
'Davis',
'Knox',
'Marshall',
'Union',
'Carbon',
'Mahoning',
'Lawrence',
'Bergen',
'Livingston',
'Warren',
'Tooele',
'Morris',
'Des Moines',
'Henderson',
'Warren',
'Cass',
'Ashland',
'Wabash',
'Seward',
'Lancaster',
```

```
 'Buffalo',
 'Dawson',
 'York',
 'Hall',
 'Huntington',
 'Iroquois',
 'Miami',
 'Ford',
 'Moffat',
 'Weld',
 'Jackson',
 'Perkins',
 'Logan',
 'Sedgwick',
 'Routt',
 'Larimer',
 'Daggett',
 'Crawford',
 'Lander',
 'Eureka',
 'Richland',
 'Wayne',
 'Van Wert',
 'Wyandot',
 'Stark',
 'Peoria',
 'Northampton',
 'Pershing',
 'Schuylkill',
 'Adams',
 'Wells',
 'Woodford',
 'Columbiana',
 'Cass',
 'White',
 'Salt Lake',
 'Allen',
 'Indiana',
 'Fremont',
 'Page',
 'Taylor',
 'Ringgold',
 'Nassau',
 'Davis',
 'Van Buren',
 'Decatur',
 'Wayne',
 'Essex',
 'Appanoose',
 'Snyder',
 'Uintah',
 'Beaver',
 'Mifflin',
```

```
 'Duchesne',
 'Hudson',
 'Lee',
 'Hardin',
 'Otoe',
 'Lehigh',
 'Hunterdon',
 'Suffolk',
 'McLean',
 'Somerset',
 'Carroll',
 'Tazewell',
 'Blair',
 'Benton',
 'Phillips',
 'Huntingdon',
 'Cambria',
 'Union',
 'Mercer',
 'Carroll',
 'Fulton',
 'Morrow',
 'Marion',
 'Saline',
 'Adams',
 'Clay',
 'Fillmore',
 'Juniata',
 'Hayes',
 'Chase',
 'Frontier',
 'Gosper',
 'Auglaize',
 'Kearney',
 'Wasatch',
 'Phelps',
 'Berks',
 'Westmoreland',
 'Allegheny',
 'Grant',
 'Holmes',
 'Dauphin',
 'Tuscarawas',
 'Hancock',
 'McDonough',
 'Perry',
 'Hancock',
 'Bucks',
 'Clark',
 'Scotland',
 'Schuyler',
 'Middlesex',
 'Jefferson',
```

```
'Blackford',
'Putnam',
'Atchison',
'Jay',
'Utah',
'Nodaway',
'Mercer',
'Howard',
'Harrison',
'Tippecanoe',
'Worth',
'Knox',
'Nemaha',
'Lebanon',
'Logan',
'Gage',
'Johnson',
'Morgan',
'Union',
'Vermilion',
'Warren',
'Shelby',
'Washington',
'Grand',
'Coshocton',
'Tehama',
'Monmouth',
'Plumas',
'Delaware',
'Montgomery',
'Mason',
'Clinton',
'Yuma',
'Washington',
'Harrison',
'Mercer',
'Tipton',
'Champaign',
'Brooke',
'Madison',
'Delaware',
'Gentry',
'Sullivan',
'Fountain',
'Darke',
'Thayer',
'Jefferson',
'Adair',
'Dundy',
'Franklin',
'Webster',
'Nuckolls',
'Hitchcock',
```

```
'Harlan',
'Furnas',
'Red Willow',
'Logan',
'Bedford',
'Cumberland',
'Randolph',
'Lancaster',
'Knox',
'Franklin',
'Piatt',
'De Witt',
'Somerset',
'Schuyler',
'Licking',
'Champaign',
'Holt',
'Richardson',
'Pawnee',
'Grundy',
'Boulder',
'Lewis',
'Chester',
'Hamilton',
'York',
'Montgomery',
'Rio Blanco',
'Guernsey',
'Adams',
'Miami',
'Ohio',
'Boone',
'Burlington',
'Vermillion',
'Menard',
'Belmont',
'Ocean',
'Fulton',
'Muskingum',
'Butte',
'Daviess',
'Franklin',
'Fayette',
'Philadelphia',
'Andrew',
'Cass',
'White Pine',
'Madison',
'Brown',
'Garfield',
'Henry',
'Adams',
'Delaware',
```

```
'Macon',
'De Kalb',
'Macon',
'Clark',
'Linn',
'Marshall',
'Greene',
'Wayne',
'Juab',
'Churchill',
'Norton',
'Phillips',
...]
```

To extract many columns, you must select the columns you want and call their `.values` attribute.

If we were interested in grabbing all of the `HR` variables in the dataframe, we could first select those column names:

```
HRs = [col for col in data_table.columns if col.startswith('HR')]
```

We can use this to focus only on the columns we want:

```
data_table[HRs]
```

	HR60	HR70	HR80	HR90
0	0.000000	0.000000	8.855827	0.000000
1	0.000000	0.000000	17.208742	15.885624
2	1.863863	1.915158	3.450775	6.462453
3	2.612330	1.288643	3.263814	6.996502
4	0.000000	0.000000	7.770008	7.478033
5	0.000000	0.000000	4.573101	4.000640
6	7.976390	5.536179	5.633168	5.720497
7	1.011173	1.689475	4.490115	2.814460
8	11.529039	9.273857	28.227324	5.500096
9	0.000000	5.708740	0.000000	6.605892
10	0.000000	0.000000	0.000000	0.000000
11	3.574045	3.840688	7.413585	1.888146
12	0.000000	0.000000	0.000000	0.000000
13	0.000000	0.000000	0.000000	0.000000
14	0.000000	0.000000	0.000000	0.000000

15	0.000000	0.000000	0.000000	0.000000
16	2.945942	0.000000	0.000000	0.000000
17	0.000000	0.000000	19.161808	5.219752
18	0.000000	0.000000	0.000000	0.000000
19	0.000000	8.117213	0.000000	0.000000
20	0.000000	0.000000	0.000000	0.000000
21	0.000000	4.864050	0.000000	0.000000
22	0.000000	0.000000	0.000000	2.218377
23	4.119804	9.910312	14.287755	14.863258
24	5.530668	0.000000	12.421589	0.000000
25	5.854801	5.811757	6.504065	4.045798
26	0.000000	10.811980	0.000000	0.000000
27	1.422131	2.439530	4.061193	2.086920
28	2.138534	2.142245	5.518079	3.756292
29	0.708135	1.138434	0.570854	1.150993
...	...	...	...	...
3055	0.000000	0.000000	2.107526	0.739919
3056	0.000000	0.970572	4.400324	0.825491
3057	0.611430	2.568301	1.974919	2.828994
3058	2.022286	2.033140	0.000000	3.987956
3059	0.850723	1.981012	2.782690	2.260130
3060	1.790825	3.732470	5.757329	4.007173
3061	0.556604	1.060271	1.010560	2.769193
3062	0.000000	1.756836	5.505395	2.907653
3063	0.427592	3.688132	2.625587	0.000000
3064	0.896660	2.704530	4.229303	2.938915
3065	1.051403	5.379304	7.057466	3.424679
3066	2.095434	6.671377	9.243279	7.285916
3067	1.872835	3.951663	5.339935	3.201065
3068	1.917913	6.382639	1.304631	0.000000
3069	1.939789	4.960564	0.000000	6.072530
3070	5.452765	15.156192	24.841012	28.268787
3071	7.520089	9.163383	10.590286	6.443839
3072	7.202448	9.746302	11.850014	12.561604
3073	8.253379	15.655752	21.173432	16.479507

3074	2.181802	3.074760	3.191133	3.300700
3075	4.902862	11.782264	7.680787	18.362582
3076	18.513376	17.133324	15.034136	12.027015
3077	4.159907	4.126434	3.967782	6.585273
3078	5.403098	5.970974	4.127839	2.586787
3079	1.121183	1.096311	2.442074	2.806112
3080	5.046682	13.152054	13.251761	5.521552
3081	3.411368	7.393533	11.453817	8.691999
3082	1.544425	6.023552	5.280349	4.367330
3083	9.302820	1.800148	3.000030	3.727712
3084	3.396162	2.284879	1.194743	2.048855

3085 rows × 4 columns

With this, calling `.values` gives an array containing all of the entries in this subset of the table:

```
data_table[['HR90', 'HR80']].values
```

```
array([[  0.       ,   8.85582713],
       [ 15.88562351,  17.20874204],
       [  6.46245315,   3.4507747 ],
       ...,
       [  4.36732988,   5.2803488 ],
       [  3.72771194,   3.00003   ],
       [  2.04885495,   1.19474313]])
```

Using the PySAL pdio tools means that if you're comfortable with working in Pandas, you can continue to do so.

If you're more comfortable using Numpy or raw Python to do your data processing, PySAL's IO tools naturally support this.

Spatial Weights

Spatial weights are mathematical structures used to represent spatial relationships. They characterize the relationship of each observation to every other observation using some concept of proximity or closeness that depends on the weight type.

They can be build in PySAL from shapefiles, as well as some types of files.

```
import pysal as ps
import numpy as np
```

There are functions to construct weights directly from a file path.

```
shp_path = ps.examples.get_path('NAT.shp')
```

Weight Types

Contiguity:

Queen Weights

A commonly-used type of weight is a queen contigutiy weight, which reflects adjacency relationships as a binary indicator variable denoting whether or not a polygon shares an edge or a verted each another polygon. These weights are symmetric, in that when polygon A neighbors polygon B, both $w*{AB} = 1$ and $w*{BA} = 1$.

To construct queen weights from a shapefile, use the `queen_from_shapefile` function:

```
qW = ps.queen_from_shapefile(shp_path)
dataframe = ps.pdio.read_files(shp_path)
```

```
qW
```

```
<pysal.weights.weights.W at 0x7f17686af320>
```

All weights objects have a few traits that you can use to work with the weights object, as well as to get information about the weights object.

To get the neighbors & weights around an observation, use the observation's index on the weights object, like a dictionary:

```
qW[4] #neighbors & weights of the 5th observation
```

```
{2: 1.0, 5: 1.0, 28: 1.0, 62: 1.0}
```

By default, the weights and the pandas dataframe will use the same index. So, we can view the observation and its neighbors in the dataframe by putting the observation's index and its neighbors' indexes together in one list:

```
self_and_neighbors = [4]
self_and_neighbors.extend(qW.neighbors[4])
print(self_and_neighbors)
```

```
[4, 2, 28, 5, 62]
```

and grabbing those elements from the dataframe:

```
dataframe.loc[self_and_neighbors]
```

	NAME	STATE_NAME	STATE_FIPS	CNTY_FIPS	FIPS	STFIPS
4	Pend Oreille	Washington	53	051	53051	53
2	Stevens	Washington	53	065	53065	53
28	Bonner	Idaho	16	017	16017	16
5	Boundary	Idaho	16	021	16021	16
62	Spokane	Washington	53	063	53063	53

5 rows × 70 columns

A full, dense matrix describing all of the pairwise relationships is constructed using the `.full` method, or when `pysal.full` is called on a weights object:

```
Wmatrix, ids = qW.full()
#Wmatrix, ids = ps.full(qW)
```

```
Wmatrix
```

```
array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  1., ...,  0.,  0.,  0.],
       [ 0.,  1.,  0., ...,  0.,  0.,  0.],
       ...,
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.]])
```

Note that this matrix is binary, in that its elements are either zero or one, since an observation is either a neighbor or it is not a neighbor.

However, many common use cases of spatial weights require that the matrix is row-standardized. This is done simply in PySAL using the `.transform` attribute

```
qW.transform = 'r'
```

Now, if we build a new full matrix, its rows should sum to one:

```
Wmatrix, ids = qW.full()
```

```
Wmatrix.sum(axis=1) #numpy axes are 0:column, 1:row, 2:facet, into higher dimensions
```

```
array([ 1.,  1.,  1., ...,  1.,  1.,  1.])
```

Since weight matrices are typically very sparse, there is also a sparse weights matrix constructor:

```
qW.sparse
```

```
<3085x3085 sparse matrix of type '<class 'numpy.float64'>'
   with 18168 stored elements in Compressed Sparse Row format>
```

By default, PySAL assigns each observation an index according to the order in which the observation was read in. This means that, by default, all of the observations in the weights object are indexed by table order. If you have an alternative ID variable, you can pass that into the weights constructor.

For example, the `NAT.shp` dataset has a possible alternative ID Variable, a `FIPS` code.

```
dataframe.head()
```

	NAME	STATE_NAME	STATE_FIPS	CNTY_FIPS	FIPS	STFIPS
0	Lake of the Woods	Minnesota	27	077	27077	27
1	Ferry	Washington	53	019	53019	53
2	Stevens	Washington	53	065	53065	53
3	Okanogan	Washington	53	047	53047	53
4	Pend Oreille	Washington	53	051	53051	53

5 rows × 70 columns

The observation we were discussing above is in the fifth row: Pend Oreille county, Washington. Note that its FIPS code is 53051.

Then, instead of indexing the weights and the dataframe just based on read-order, use the `FIPS` code as an index:

```
qW = ps.queen_from_shapefile(shp_path, idVariable='FIPS')
```

```
ps.knnW_from_shapefile??
```

Now, Pend Oreille county has a different index:

```
qW[4] #fails, since no FIPS is 4.
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

<ipython-input-17-1d8a3009bc1e> in <module>()
----> 1 qW[4] #fails, since no FIPS is 4.


/home/serge/anaconda2/envs/scipy16/lib/python3.5/site-packages/pysal/weights/weights.py in __getite
m__(self, key)
    504         {1: 1.0, 4: 1.0, 101: 1.0, 85: 1.0, 5: 1.0}
    505         """
--> 506         return dict(list(zip(self.neighbors[key], self.weights[key])))
    507
    508     def __iter__(self):


KeyError: 4
```

Note that a `KeyError` in Python usually means that some index, here `4`, was not found in the collection being searched, the IDs in the queen weights object. This makes sense, since we explicitly passed an `idVariable` argument, and nothing has a `FIPS` code of 4.

Instead, if we use the observation's `FIPS` code:

```
qW['53051']
```

```
{'16017': 1.0, '16021': 1.0, '53063': 1.0, '53065': 1.0}
```

We get what we need.

In addition, we have to now query the dataframe using the `FIPS` code to find our neighbors. But, this is relatively easy to do, since pandas will parse the query by looking into python objects, if told to.

First, let us store the neighbors of our target county:

```
self_and_neighbors = ['53051']
self_and_neighbors.extend(qW.neighbors['53051'])
```

Then, we can use this list in `.query`:

```
dataframe.query('FIPS in @self_and_neighbors')
```

	NAME	STATE_NAME	STATE_FIPS	CNTY_FIPS	FIPS	STFIPS
2	Stevens	Washington	53	065	53065	53
4	Pend Oreille	Washington	53	051	53051	53
5	Boundary	Idaho	16	021	16021	16
28	Bonner	Idaho	16	017	16017	16
62	Spokane	Washington	53	063	53063	53

5 rows × 70 columns

Note that we have to use `@` before the name in order to show that we're referring to a python object and not a column in the dataframe.

```
#dataframe.query('FIPS in neighs') will fail because there is no column called 'neighs'
```

Of course, we could also reindex the dataframe to use the same index as our weights:

```
fips_frame = dataframe.set_index(dataframe.FIPS)
fips_frame.head()
```

	NAME	STATE_NAME	STATE_FIPS	CNTY_FIPS	FIPS	ST
FIPS						
27077	Lake of the Woods	Minnesota	27	077	27077	27
53019	Ferry	Washington	53	019	53019	53
53065	Stevens	Washington	53	065	53065	53
53047	Okanogan	Washington	53	047	53047	53
53051	Pend Oreille	Washington	53	051	53051	53

5 rows × 70 columns

Now that both are using the same weights, we can use the `.loc` indexer again:

```
fips_frame.loc[self_and_neighbors]
```

	NAME	STATE_NAME	STATE_FIPS	CNTY_FIPS	FIPS	ST
FIPS						
53051	Pend Oreille	Washington	53	051	53051	53
53065	Stevens	Washington	53	065	53065	53
16017	Bonner	Idaho	16	017	16017	16
16021	Boundary	Idaho	16	021	16021	16
53063	Spokane	Washington	53	063	53063	53

5 rows × 70 columns

Rook Weights

Rook weights are another type of contiguity weight, but consider observations as neighboring only when they share an edge. The rook neighbors of an observation may be different than its queen neighbors, depending on how the observation and its nearby polygons are configured.

We can construct this in the same way as the queen weights, using the special `rook_from_shapefile` function

```
rW = ps.rook_from_shapefile(shp_path, idVariable='FIPS')
```

```
rW['53051']
```

```
{'16017': 1.0, '16021': 1.0, '53063': 1.0, '53065': 1.0}
```

These weights function exactly like the Queen weights, and are only distinguished by what they consider "neighbors."

Bishop Weights

In theory, a "Bishop" weighting scheme is one that arises when only polygons that share vertexes are considered to be neighboring. But, since Queen contiguigy requires either an edge or a vertex and Rook contiguity requires only shared edges, the following relationship is true:

$\mathcal{Q} = \mathcal{R} \cup \mathcal{B}$

where $\mathcal{Q}$ is the set of neighbor pairs *via* queen contiguity, $\mathcal{R}$ is the set of neighbor pairs *via* Rook contiguity, and $\mathcal{B}$ *via* Bishop contiguity. Thus:

$\mathcal{Q} \setminus \mathcal{R} = \mathcal{B}$

Bishop weights entail all Queen neighbor pairs that are not also Rook neighbors.

PySAL does not have a dedicated bishop weights constructor, but you can construct very easily using the `w_difference` function. This function is one of a family of tools to work with weights, all defined in `ps.weights`, that conduct these types of set operations between weight objects.

```
bW = ps.w_difference(qW, rW, constrained=False, silent_island_warning=True) #silence because there will be a lot of warnings
```

```
bW.histogram
```

```
[(0, 2359), (1, 531), (2, 150), (3, 31), (4, 14)]
```

Thus, the vast majority of counties have no bishop neighbors. But, a few do. A simple way to see these observations in the dataframe is to find all elements of the dataframe that are not "islands," the term for an observation with no neighbors:

```
islands = bW.islands
```

```
dataframe.query('FIPS not in @islands')
```

	NAME	STATE_NAME	STATE_FIPS	CNTY_FIPS	FIPS	ST
32	Williams	North Dakota	38	105	38105	38
35	Roosevelt	Montana	30	085	30085	30
56	McKenzie	North Dakota	38	053	38053	38
58	Richland	Montana	30	083	30083	30
123	Bayfield	Wisconsin	55	007	55007	55
135	Douglas	Wisconsin	55	031	55031	55
148	Emmons	North Dakota	38	029	38029	38
153	Richland	North Dakota	38	077	38077	38
166	Sioux	North Dakota	38	085	38085	38
167	Pine	Minnesota	27	115	27115	27
176	Sargent	North Dakota	38	081	38081	38
178	Adams	North Dakota	38	001	38001	38
181	Clatsop	Oregon	41	007	41007	41
183	Columbia	Oregon	41	009	41009	41
185	Washburn	Wisconsin	55	129	55129	55
186	Sawyer	Wisconsin	55	113	55113	55
188	Kanabec	Minnesota	27	065	27065	27
191	Grant	Minnesota	27	051	27051	27
192	Douglas	Minnesota	27	041	27041	27
200	Campbell	South Dakota	46	021	46021	46

204	Corson	South Dakota	46	031	46031	46
205	Brown	South Dakota	46	013	46013	46
207	Marshall	South Dakota	46	091	46091	46
208	Roberts	South Dakota	46	109	46109	46
216	Tillamook	Oregon	41	057	41057	41
217	Washington	Oregon	41	067	41067	41
218	Pope	Minnesota	27	121	27121	27
219	Stevens	Minnesota	27	149	27149	27
220	Isanti	Minnesota	27	059	27059	27
...	...	...	...	...	...	...
2976	Ozaukee	Wisconsin	55	089	55089	55
2978	Montcalm	Michigan	26	117	26117	26
2979	Gratiot	Michigan	26	057	26057	26
2992	Waukesha	Wisconsin	55	133	55133	55
2993	Milwaukee	Wisconsin	55	079	55079	55
2998	Ionia	Michigan	26	067	26067	26
2999	Clinton	Michigan	26	037	26037	26
3012	Livingston	Michigan	26	093	26093	26
3013	Ingham	Michigan	26	065	26065	26
3027	Winnebago	Illinois	17	201	17201	17
3029	Boone	Illinois	17	007	17007	17
3032	Washtenaw	Michigan	26	161	26161	26

3033	Jackson	Michigan	26	075	26075	26
3034	Kalamazoo	Michigan	26	077	26077	26
3035	Calhoun	Michigan	26	025	26025	26
3036	Van Buren	Michigan	26	159	26159	26
3042	Ogle	Illinois	17	141	17141	17
3045	Kane	Illinois	17	089	17089	17
3046	De Kalb	Illinois	17	037	17037	17
3050	Branch	Michigan	26	023	26023	26
3051	St. Joseph	Michigan	26	149	26149	26
3052	Cass	Michigan	26	027	26027	26
3055	Warren	Pennsylvania	42	123	42123	42
3057	McKean	Pennsylvania	42	083	42083	42
3059	Du Page	Illinois	17	043	17043	17
3065	La Porte	Indiana	18	091	18091	18
3068	Lagrange	Indiana	18	087	18087	18
3069	Steuben	Indiana	18	151	18151	18
3074	Fairfax	Virginia	51	059	51059	51
3079	Oconto	Wisconsin	55	083	55083	55

726 rows × 70 columns

Distance

There are many other kinds of weighting functions in PySAL. Another separate type use a continuous measure of distance to define neighborhoods. To use these measures, we first must extract the polygons' centroids.

For each polygon `poly` in `dataframe.geometry`, we want `poly.centroid`. So, one way to do this is to make a list of all of the centroids:

```
centroids = [list(poly.centroid) for poly in dataframe.geometry]
```

```
centroids[0:5] #let's look at the first five
```

```
[[-94.90336786329912, 48.771730563701574],
 [-118.51718120712802, 48.46959353253665],
 [-117.85532452342407, 48.39591039096631],
 [-119.73943524482668, 48.5484338901435],
 [-117.27400489644516, 48.53279719845048]]
```

If we were working with point data, this step would be unncessary.

KnnW

If we wanted to consider only the `k`-nearest neighbors to an observation's centroid, we could use the `knnW` function in PySAL.

This specific type of distance weights requires that we first build a `KDTree`, a special representation for spatial point data. Fortunately, this is built in to PySAL:

```
kdtree = ps.cg.KDTree(centroids)
```

Then, we can use this to build a spatial weights object where only the closest `k` observations are considered "neighbors." In this example, let's do the closest 5:

```
nn5 = ps.knnW(kdtree, k=5)
```

```
nn5.histogram
```

```
[(5, 3085)]
```

So, all observations have exactly 5 neighbors. Sometimes, these neighbors are actually different observations than the ones identified by contiguity neighbors.

For example, Pend Oreille gets a new neighbor, Kootenai county:

```
nn5[4]
```

```
{2: 1.0, 5: 1.0, 28: 1.0, 62: 1.0, 65: 1.0}
```

```
dataframe.loc[nn5.neighbors[4] + [4]]
```

	NAME	STATE_NAME	STATE_FIPS	CNTY_FIPS	FIPS	STFIPS
2	Stevens	Washington	53	065	53065	53
28	Bonner	Idaho	16	017	16017	16
5	Boundary	Idaho	16	021	16021	16
62	Spokane	Washington	53	063	53063	53
65	Kootenai	Idaho	16	055	16055	16
4	Pend Oreille	Washington	53	051	53051	53

6 rows × 70 columns

```
fips_frame.loc[qW.neighbors['53051'] + ['53051']]
```

	NAME	STATE_NAME	STATE_FIPS	CNTY_FIPS	FIPS	ST
FIPS						
53065	Stevens	Washington	53	065	53065	53
16017	Bonner	Idaho	16	017	16017	16
16021	Boundary	Idaho	16	021	16021	16
53063	Spokane	Washington	53	063	53063	53
53051	Pend Oreille	Washington	53	051	53051	53

5 rows × 70 columns

Kernel W

Kernel Weights are continuous distance-based weights that use kernel densities to provide an indication of neighborliness.

Typically, they estimate a `bandwidth`, which is a parameter governing how far out observations should be considered neighboring. Then, using this bandwidth, they evaluate a continuous kernel function to provide a weight between 0 and 1.

Many different choices of kernel functions are supported, and bandwidth can be estimated at each point or over the entire map.

For example, if we wanted to use a single estimated bandwidth for the entire map and weight according to a gaussian kernel:

```
kernelW = ps.Kernel(centroids, fixed=True, function='gaussian')
#ps.Kernel(centroids, fixed=False, function='gaussian') #same kernel, but bandwidth changes at each observation
```

```
dataframe.loc[kernelW.neighbors[4] + [4]]
```

	NAME	STATE_NAME	STATE_FIPS	CNTY_FIPS	FIPS	STFII
1	Ferry	Washington	53	019	53019	53
69	Lincoln	Washington	53	043	53043	53
2	Stevens	Washington	53	065	53065	53
110	Whitman	Washington	53	075	53075	53
62	Spokane	Washington	53	063	53063	53
4	Pend Oreille	Washington	53	051	53051	53
65	Kootenai	Idaho	16	055	16055	16
100	Benewah	Idaho	16	009	16009	16
28	Bonner	Idaho	16	017	16017	16
5	Boundary	Idaho	16	021	16021	16
4	Pend Oreille	Washington	53	051	53051	53

11 rows × 70 columns

As you can see, this provides our target observation, Pend Oreille, with many new neighbors. \

Exploratory Geovisualization with PySAL

Introduction

When PySAL was originally planned, the intention was to focus on the computational aspects of exploratory spatial data analysis and spatial econometric methods, while relying on existing GIS packages and visualization libraries for visualization of computations. Indeed, we have partnered with esri and QGIS towards this end.

However, over time we have received many requests for supporting basic geovisualization within PySAL so that the step of having to interoperate with an exertnal package can be avoided, thereby increasing the efficiency of the spatial analytical workflow.

In this notebook, we demonstrate several approaches towards geovisualization within a self-contained exploratory workflow. The idea here is the support quick generation of different views of your data to complement the statistical and econometric work in PySAL. Once your work has progressed to the publication stage, we point you to resources that can be used for publication quality output.

PySAL Viz Module

Contributors:

- Dani Arribas-Bel daniel.arribas.bel@gmail.com
- Serge Rey sjsrey@gmail.com

This document describes the main structure, components and usage of the mapping module in PySAL. The is organized around three main layers:

- A lower-level layer that reads polygon, line and point shapefiles and returns a Matplotlib collection.
- A medium-level layer that performs some usual transformations on a Matplotlib object (e.g. color code polygons according to a vector of values).
- A higher-level layer intended for end-users for particularly useful cases and style preferences pre-defined (e.g. Create a choropleth).

```
%matplotlib inline
import numpy as np
import pysal as ps
import random as rdm
from pysal.contrib.viz import mapping as maps

from pylab import *
```

Lower-level component

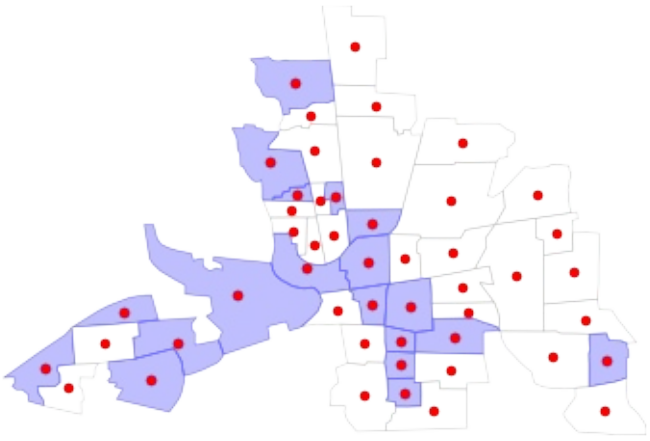
This includes basic functionality to read spatial data from a file (currently only shapefiles supported) and produce rudimentary Matplotlib objects. The main methods are:

- map_poly_shape: to read in polygon shapefiles
- map_line_shape: to read in line shapefiles
- map_point_shape: to read in point shapefiles

These methods all support an option to subset the observations to be plotted (very useful when missing values are present). They can also be overlaid and combined by using the setup_ax function. the resulting object is very basic but also very flexible so, for minds used to matplotlib this should be good news as it allows to modify pretty much any property and attribute.

Example

```
shp_link = ps.examples.get_path('columbus.shp')
shp = ps.open(shp_link)
some = [bool(rdm.getrandbits(1)) for i in ps.open(shp_link)]

fig = figure()

base = maps.map_poly_shp(shp)
base.set_facecolor('none')
base.set_linewidth(0.75)
base.set_edgecolor('0.8')
some = maps.map_poly_shp(shp, which=some)
some.set_alpha(0.5)
some.set_linewidth(0.)
cents = np.array([poly.centroid for poly in ps.open(shp_link)])
pts = scatter(cents[:, 0], cents[:, 1])
pts.set_color('red')

ax = maps.setup_ax([base, some, pts], [shp.bbox, shp.bbox, shp.bbox])
fig.add_axes(ax)
show()
```



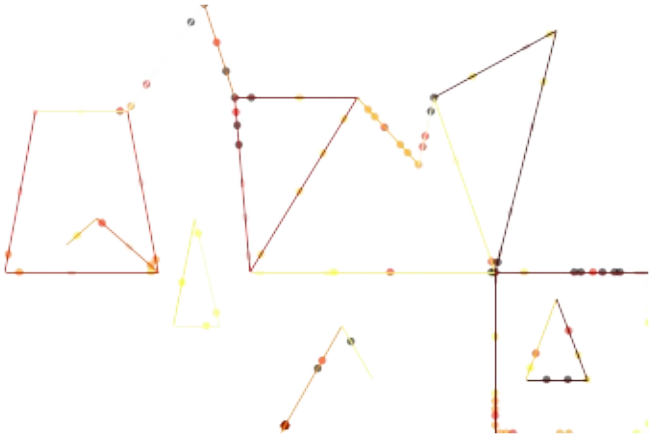
Medium-level component

This layer comprises functions that perform usual transformations on matplotlib objects, such as color coding objects (points, polygons, etc.) according to a series of values. This includes the following methods:

- base_choropleth_classless
- base_choropleth_unique

Example

```
net_link = ps.examples.get_path('eberly_net.shp')
net = ps.open(net_link)
values = np.array(ps.open(net_link.replace('.shp', '.dbf')).by_col('TNODE'))

pts_link = ps.examples.get_path('eberly_net_pts_onnetwork.shp')
pts = ps.open(pts_link)

fig = figure()

netm = maps.map_line_shp(net)
netc = maps.base_choropleth_unique(netm, values)

ptsm = maps.map_point_shp(pts)
ptsm = maps.base_choropleth_classif(ptsm, values)
ptsm.set_alpha(0.5)
ptsm.set_linewidth(0.)

ax = maps.setup_ax([netc, ptsm], [net.bbox, net.bbox])
fig.add_axes(ax)
show()
```



```
maps.plot_poly_lines(ps.examples.get_path('columbus.shp'))
```

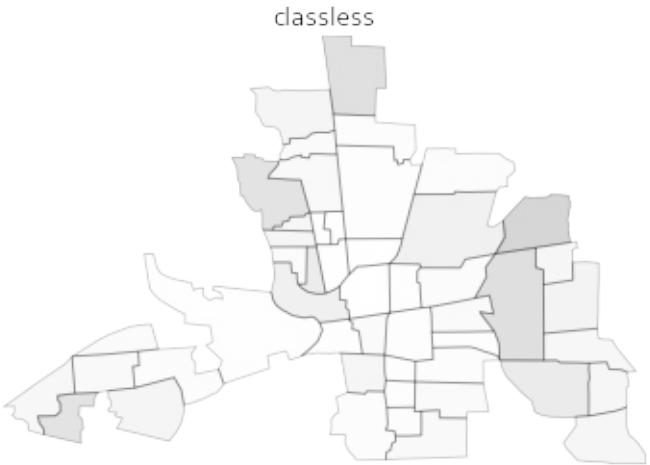
```
callng plt.show()
```

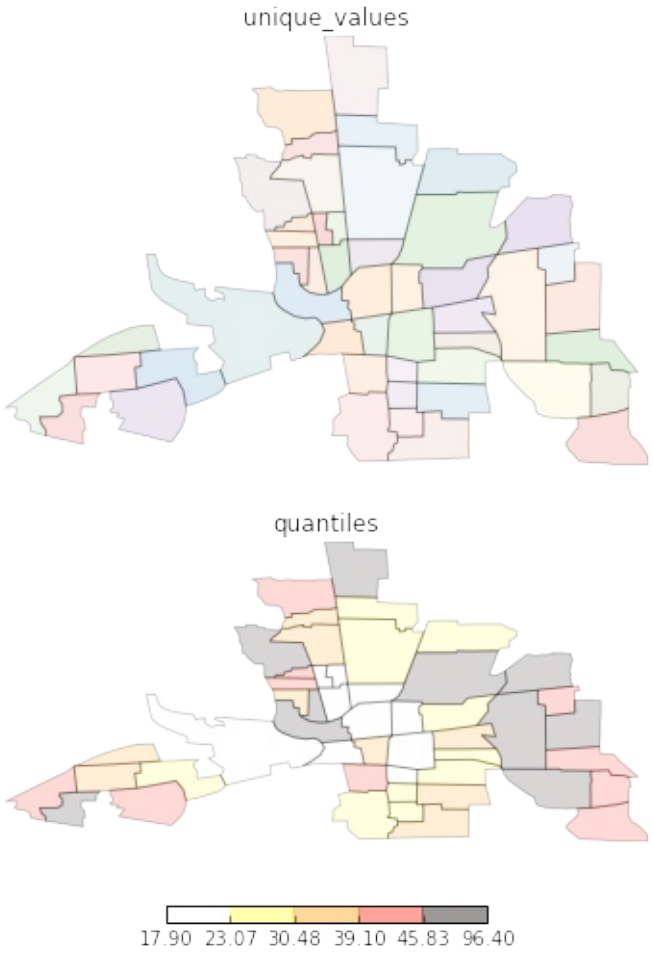
Higher-level component

This currently includes the following end-user functions:

- plot_poly_lines: very quick shapfile plotting

```
shp_link = ps.examples.get_path('columbus.shp')
values = np.array(ps.open(ps.examples.get_path('columbus.dbf')).by_col('HOVAL'))

types = ['classless', 'unique_values', 'quantiles', 'equal_interval', 'fisher_jenks']
for typ in types:
    maps.plot_choropleth(shp_link, values, typ, title=typ)
```




unique_values
quantiles
17.90 23.07 30.48 39.10 45.83 96.40
equal_interval
17.9 33.6 49.3 65.0 80.7 96.4
fisher_jenks
17.90 25.70 36.30 47.73 61.95 96.40

Folium

Contributors:

- Levi Wolf levi.john.wolf@gmail.com
- Serge Rey sjsrey@gmail.com

In addition to using matplotlib, the viz module includes components that interface with the folium library which provides a Pythonic way to generate Leaflet maps.

```
import pysal as ps
import geojson as gj
from pysal.contrib.viz import folium_mapping as fm
```

First, we need to convert the data into a JSON format. JSON, short for "Javascript Serialized Object Notation," is a simple and effective way to represent objects in a digital environment. For geographic information, the GeoJSON standard defines how to represent geographic information in JSON format. Python programmers may be more comfortable thinking of JSON data as something akin to a standard Python dictionary.

```
filepath = ps.examples.get_path('south.shp')[:-4]
shp = ps.open(filepath + '.shp')
dbf = ps.open(filepath + '.dbf')
```

```
js = fm.build_features(shp, dbf)
```

Just to show, this constructs a dictionary with the following keys:

```
js.keys()
```

```
dict_keys(['type', 'features', 'bbox'])
```

```
js.type
```

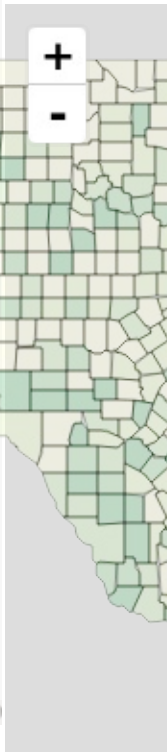
```
'FeatureCollection'
```

```
js.bbox
```

```
[-106.6495132446289, 24.95596694946289, -75.0459976196289, 40.63713836669922]
```

```
js.features[0]
```

{"bbox": [-80.6688232421875, 40.39815902709961, -80.52220916748047, 40.63713836669922], "geometry": {"coordinates": [[[-80.6280517578125, 40.39815902709961], [-80.60203552246094, 40.480472564697266], [-80.62545776367188, 40.504398345947266], [-80.6336441040039, 40.53913879394531], [-80.6688232421875, 40.568214416503906], [-80.66793060302734, 40.58207321166992], [-80.63754272460938, 40.61391830444336], [-80.61175537109375, 40.619998931884766], [-80.57462310791016, 40.615909576416016], [-80.52220916748047, 40.63713836669922], [-80.52456665039062, 40.47871780395508], [-80.52377319335938, 40.4029655456543], [-80.6280517578125, 40.39815902709961]]], "type": "Polygon"}, "properties": {"BLK60": 3.839454752, "BLK70": 3.2554278095, "BLK80": 2.5607402642, "BLK90": 2.5572616581, "CNTY_FIPS": "029", "COFIPS": 29, "DNL60": 6.1681225056, "DNL70": 6.1714993547, "DNL80": 6.1714631077, "DNL90": 6.0508978146, "DV60": 2.2779893943, "DV70": 2.5591397849, "DV80": 5.0619350519, "DV90": 7.2636377003, "FH60": 9.9812973718, "FH70": 7.8, "FH80": 9.7857968181, "FH90": 12.604551644, "FIPS": "54029", "FIPSNO": 54029, "FP59": 9.6, "FP69": 5.9, "FP79": 6.5327526442, "FP89": 10.17311807, "GI59": 0.2236450331, "GI69": 0.2953773833, "GI79": 0.3322512119, "GI89": 0.3639335641, "HC60": 0.6666666667, "HC70": 1.6666666667, "HC80": 2.6666666667, "HC90": 0.3333333333, "HR60": 1.6828642349, "HR70": 4.1929776011, "HR80": 6.5977204876, "HR90": 0.9460827444, "MA60": 28.9, "MA70": 30.0, "MA80": 31.4, "MA90": 37.7, "MFIL59": 8.8410143105, "MFIL69": 9.2471543451, "MFIL79": 10.073356901, "MFIL89": 10.327970666, "NAME": "Hancock", "PO60": 39615, "PO70": 39749, "PO80": 40418, "PO90": 35233, "POL60": 10.586963113, "POL70": 10.590339963, "POL80": 10.607030509, "POL90": 10.469738422, "PS60": 1.218684208, "PS70": 1.1368342185, "PS80": 1.0385705291, "PS90": 0.8964534429, "RD60": -1.394676863, "RD70": -1.307438562, "RD80": -1.159302086, "RD90": -0.399028376, "SOUTH": 1, "STATE_FIPS": "54", "STATE_NAME": "West Virginia", "STFIPS": 54, "UE60": 3.1, "UE70": 2.7, "UE80": 7.0763827919, "UE90": 6.8578070515}, "type": "Feature"}



Then, we write the json to a file:

```
with open('./example.json', 'w') as out:
    gj.dump(js, out)
```

Mapping

Let's look at the columns that we are going to map.

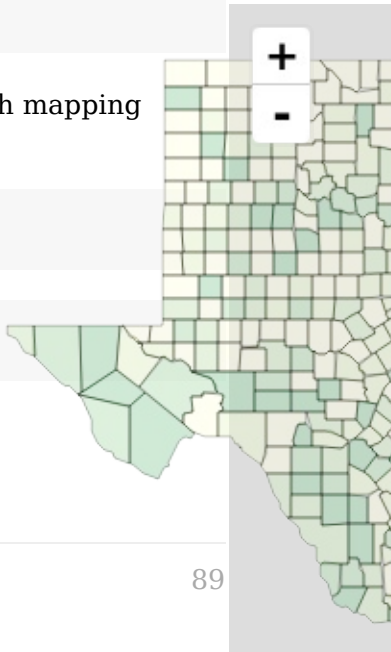
```
list(js.features[0].properties.keys())[:5]
```

```
['PO60', 'COFIPS', 'UE90', 'DV80', 'DNL80']
```

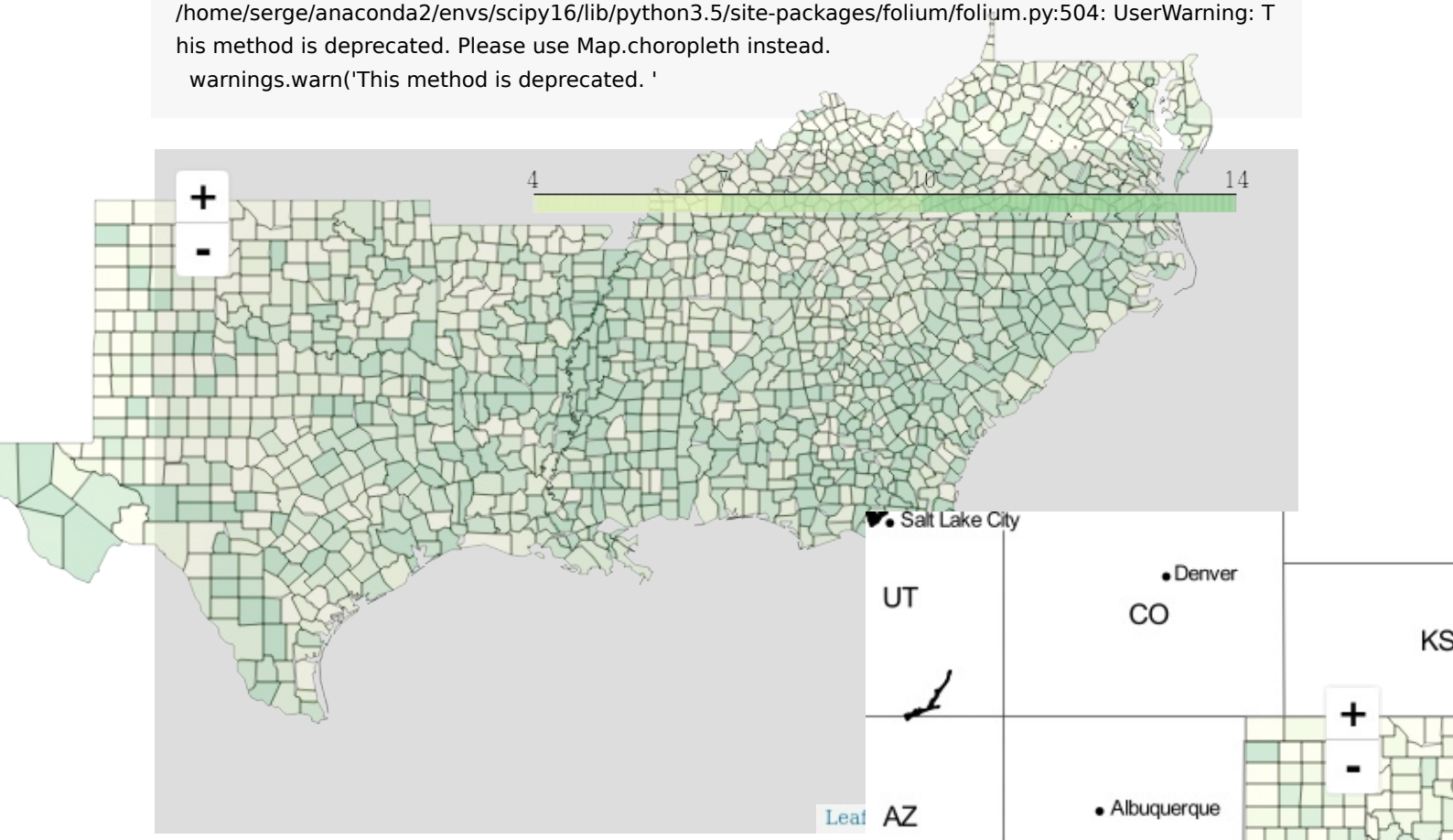
We can map these attributes by calling them as arguments to the choropleth mapping function:

```
import folium as fol
```

```
fm.choropleth_map('./example.json', 'FIPS', 'HR90')
```



```
/home/serge/anaconda2/envs/scipy16/lib/python3.5/site-packages/folium/folium.py:504: UserWarning: T
his method is deprecated. Please use Map.choropleth instead.
  warnings.warn('This method is deprecated. '
```

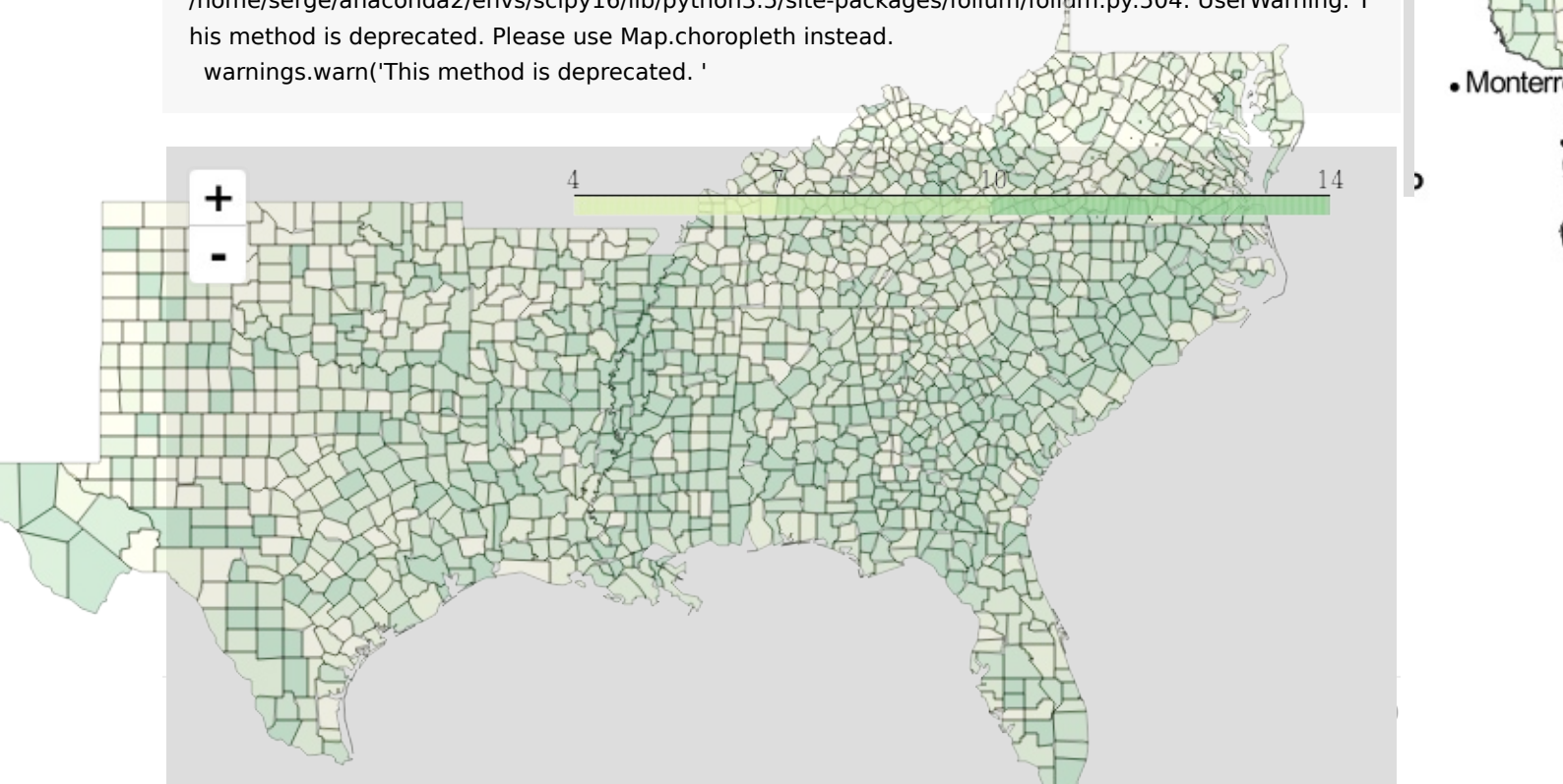


This produces a map using default classifications and col
file. We set the function to have sane defaults. However, i
control, we have many options available.

There are arguments to change the classification scheme:

```
fm.choropleth_map('./example.json', 'FIPS', 'HR90', classification = 'Quantiles')
```

```
/home/serge/anaconda2/envs/scipy16/lib/python3.5/site-packages/folium/folium.py:504: UserWarning: T
his method is deprecated. Please use Map.choropleth instead.
  warnings.warn('This method is deprecated. '
```

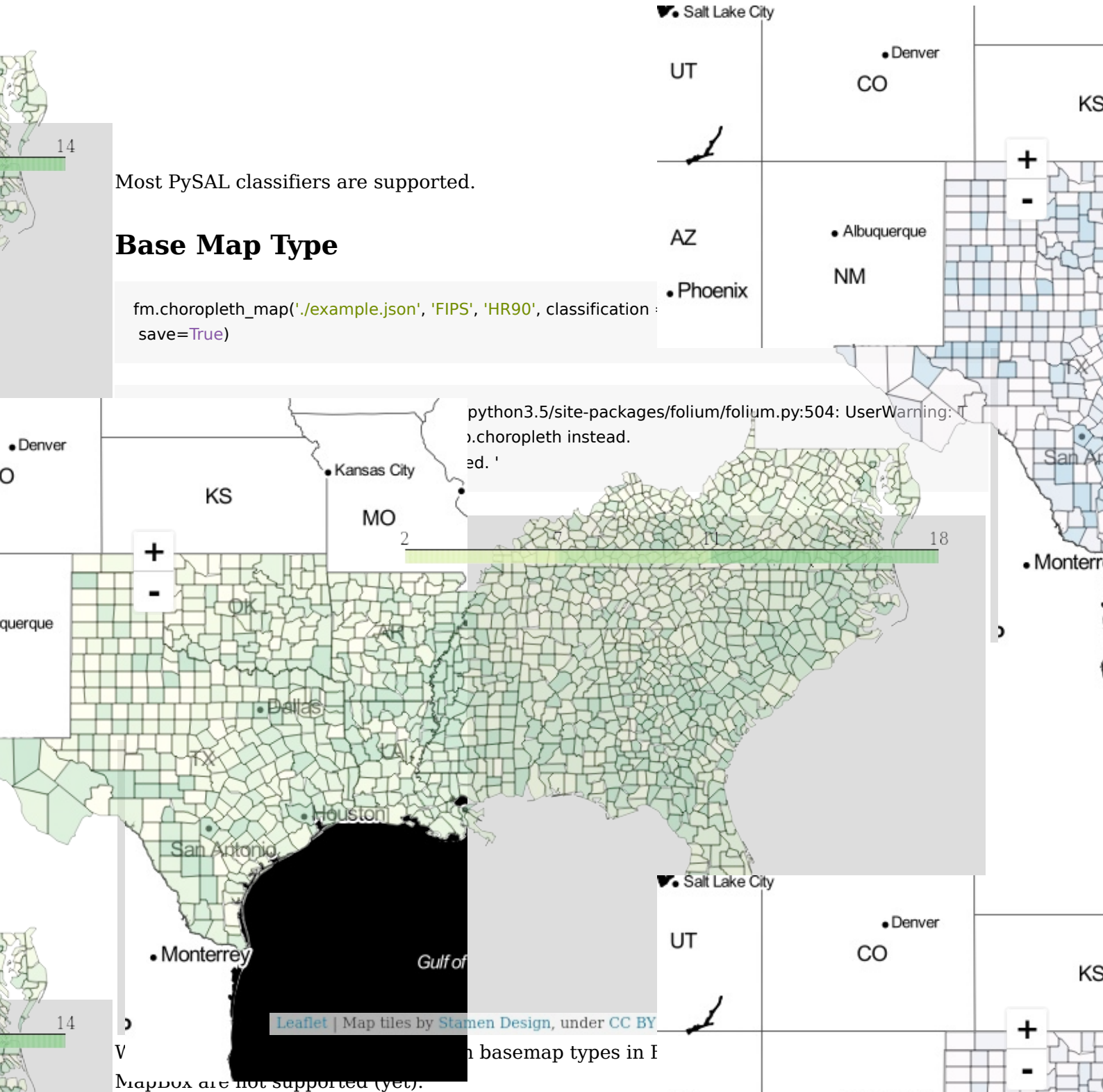


Most PySAL classifiers are supported.

Base Map Type

```
fm.choropleth_map('./example.json', 'FIPS', 'HR90', classification =
 save=True)
```

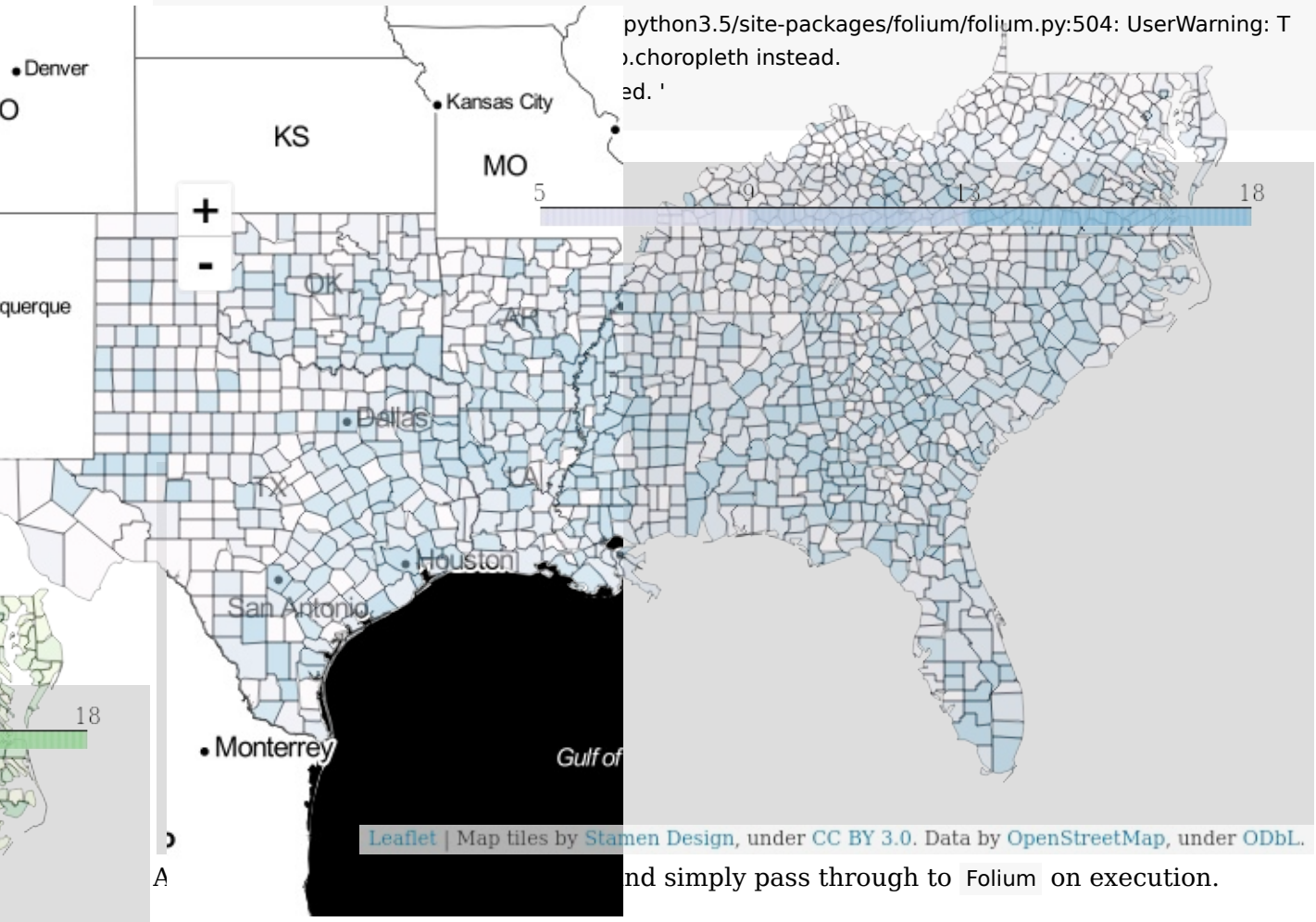
```
python3.5/site-packages/folium/folium.py:504: UserWarning: T
).choropleth instead.
ed. '
```



W n basemap types in F
MapBox are not supported (yet).

Color Scheme

```
fm.choropleth_map('./example.json', 'FIPS', 'HR80', classification =
 fill_color = 'PuBuGn', save=True)
```

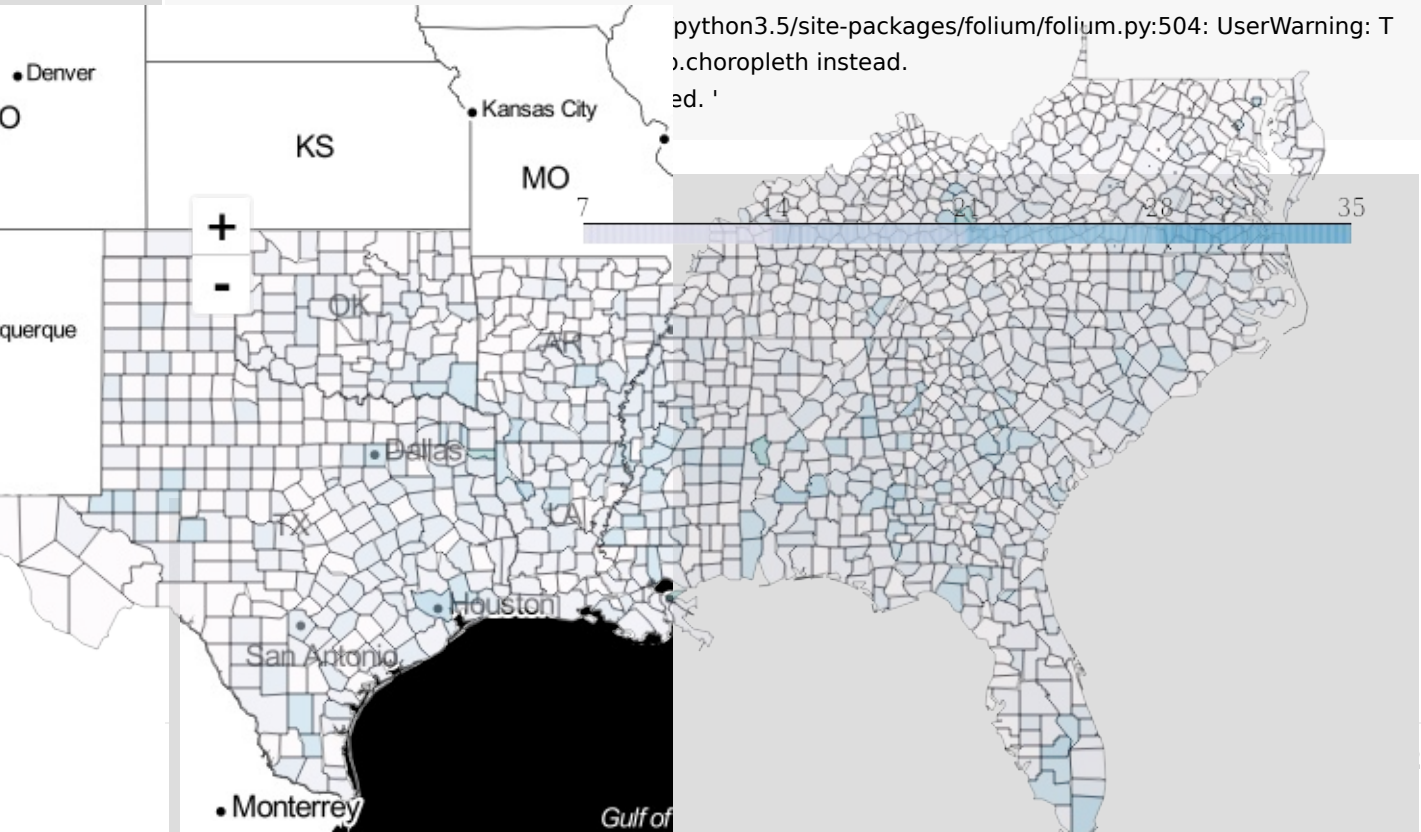


python3.5/site-packages/folium/folium.py:504: UserWarning: T
.choropleth instead.
ed. '

A nd simply pass through to `Folium` on execution.

Class numbers

```
fm.choropleth_map('./example.json', 'FIPS', 'HR80', classification = 'Equal Interval', classes=6, tiles='St
amen Toner', fill_color='PuBuGn',save=True)
```



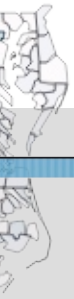
python3.5/site-packages/folium/folium.py:504: UserWarning: T
.choropleth instead.
ed. '

18

Folium supports up to 6 classes.

Cartopy

Althought we don't have time to go into the details here today, we note that for for publication ready maps one can turn to Cartopy. For an example of a recent publication and code see Rey (2016).



35

ESDA with `PySAL`

```
%matplotlib inline

import pysal as ps
```

```
/home/dani/anaconda/envs/pydata/lib/python2.7/site-packages/matplotlib/font_manager.py:273: UserW
arning: Matplotlib is building the font cache using fc-list. This may take a moment.
  warnings.warn('Matplotlib is building the font cache using fc-list. This may take a moment.')
```

```
import pysal as ps
import numpy as np
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
f = ps.open(ps.examples.get_path('usjoin.csv'), 'r')
```

To determine what is in the file, check the `header` attribute on the file object:

```
f.header
```

```
['Name',
 'STATE_FIPS',
 '1929',
 '1930',
 '1931',
 '1932',
 '1933',
 '1934',
 '1935',
 '1936',
 '1937',
 '1938',
 '1939',
 '1940',
 '1941',
 '1942',
 '1943',
 '1944',
 '1945',
 '1946',
 '1947',
 '1948',
 '1949',
 '1950',
 '1951',
 '1952',
 '1953',
 '1954',
 '1955',
 '1956',
 '1957',
 '1958',
 '1959',
 '1960',
 '1961',
 '1962',
```

```
 '1963',
 '1964',
 '1965',
 '1966',
 '1967',
 '1968',
 '1969',
 '1970',
 '1971',
 '1972',
 '1973',
 '1974',
 '1975',
 '1976',
 '1977',
 '1978',
 '1979',
 '1980',
 '1981',
 '1982',
 '1983',
 '1984',
 '1985',
 '1986',
 '1987',
 '1988',
 '1989',
 '1990',
 '1991',
 '1992',
 '1993',
 '1994',
 '1995',
 '1996',
 '1997',
 '1998',
 '1999',
 '2000',
 '2001',
 '2002',
 '2003',
 '2004',
 '2005',
 '2006',
 '2007',
 '2008',
 '2009']
```

Ok, lets pull in the name variable to see what we have

```
name = f.by_col('Name')
```

```
name
```

```
['Alabama',
 'Arizona',
 'Arkansas',
 'California',
 'Colorado',
 'Connecticut',
 'Delaware',
 'Florida',
 'Georgia',
 'Idaho',
 'Illinois',
 'Indiana',
 'Iowa',
 'Kansas',
 'Kentucky',
 'Louisiana',
 'Maine',
 'Maryland',
 'Massachusetts',
 'Michigan',
 'Minnesota',
 'Mississippi',
 'Missouri',
 'Montana',
 'Nebraska',
 'Nevada',
 'New Hampshire',
 'New Jersey',
 'New Mexico',
 'New York',
 'North Carolina',
 'North Dakota',
 'Ohio',
 'Oklahoma',
 'Oregon',
 'Pennsylvania',
 'Rhode Island',
 'South Carolina',
 'South Dakota',
 'Tennessee',
 'Texas',
 'Utah',
 'Vermont',
 'Virginia',
 'Washington',
 'West Virginia',
 'Wisconsin',
 'Wyoming']
```

Now obtain per capital incomes in 1929 which is in the column associated with `1929`

```
y1929 = f.by_col('1929')
```

```
y1929
```

```
[323,
 600,
 310,
 991,
 634,
 1024,
 1032,
 518,
 347,
 507,
 948,
 607,
 581,
 532,
 393,
 414,
 601,
 768,
 906,
 790,
 599,
 286,
 621,
 592,
 596,
 868,
 686,
 918,
 410,
 1152,
 332,
 382,
 771,
 455,
 668,
 772,
 874,
 271,
 426,
 378,
 479,
 551,
 634,
 434,
 741,
 460,
 673,
 675]
```

And now 2009

```
y2009 = f.by_col("2009")
```

```
y2009
```

```
[32274,
 32077,
 31493,
 40902,
 40093,
 52736,
 40135,
 36565,
 33086,
 30987,
 40933,
 33174,
 35983,
 37036,
 31250,
 35151,
 35268,
 47159,
 49590,
 34280,
 40920,
 29318,
 35106,
 32699,
 37057,
 38009,
 41882,
 48123,
 32197,
 46844,
 33564,
 38672,
 35018,
 33708,
 35210,
 38827,
 41283,
 30835,
 36499,
 33512,
 35674,
 30107,
 36752,
 43211,
 40619,
 31843,
 35676,
 42504]
```

These are read into regular Python lists which are not particularly well suited to efficient data analysis. So let's convert them to numpy arrays

```
y2009 = np.array(y2009)
```

```
y2009
```

```
array([32274, 32077, 31493, 40902, 40093, 52736, 40135, 36565, 33086,
       30987, 40933, 33174, 35983, 37036, 31250, 35151, 35268, 47159,
       49590, 34280, 40920, 29318, 35106, 32699, 37057, 38009, 41882,
       48123, 32197, 46844, 33564, 38672, 35018, 33708, 35210, 38827,
       41283, 30835, 36499, 33512, 35674, 30107, 36752, 43211, 40619,
       31843, 35676, 42504])
```

Much better. But pulling these in and converting them a column at a time is tedious and error prone. So we will do all of this in a list comprehension.

```
Y = np.array( [ f.by_col(str(year)) for year in range(1929,2010) ] ) * 1.0
```

```
Y.shape
```

```
(81, 48)
```

```
Y = Y.transpose()
```

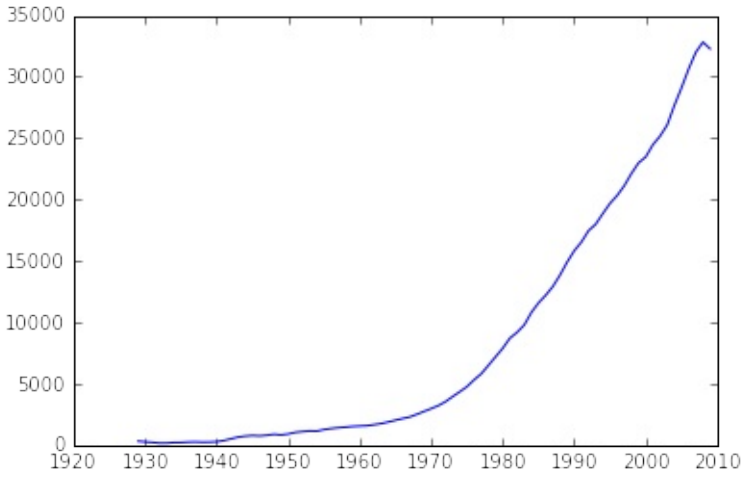
```
Y.shape
```

```
(48, 81)
```

```
years = np.arange(1929,2010)
```

```
plot(years,Y[0])
```

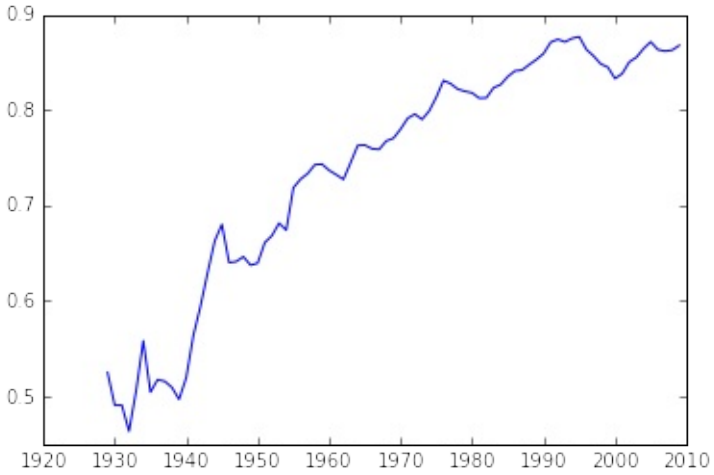
```
[<matplotlib.lines.Line2D at 0x7febfbf24e80>]
```




```
RY = Y / Y.mean(axis=0)
```

```
plot(years,RY[0])
```

```
[<matplotlib.lines.Line2D at 0x7febfb9d02b0>]
```



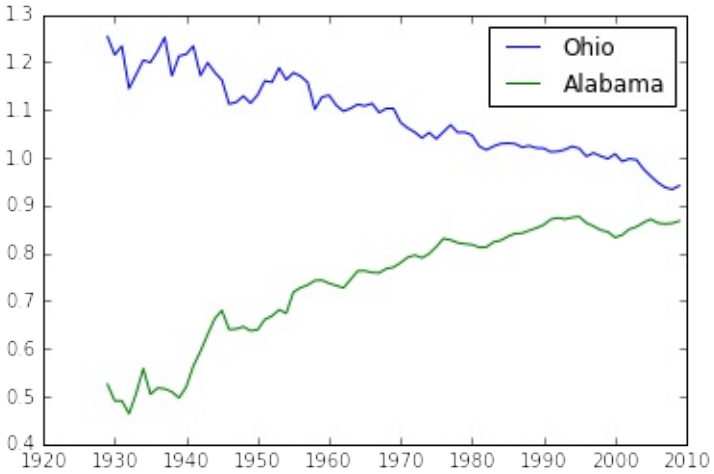

```
name = np.array(name)
```

```
np.nonzero(name=='Ohio')
```

```
(array([32]),)
```

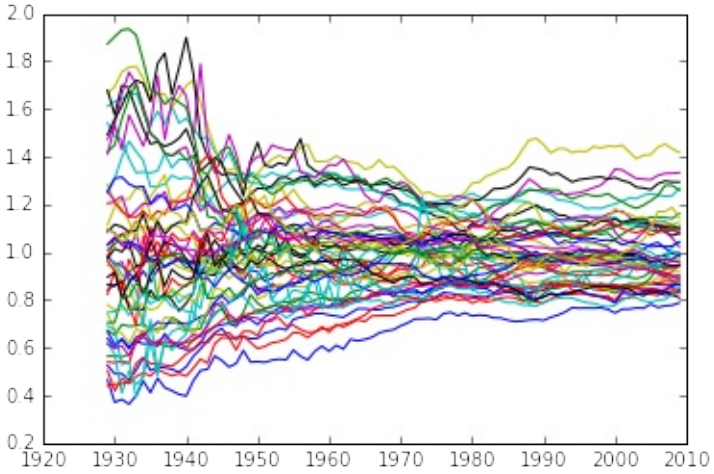
```
plot(years, RY[32], label='Ohio')
plot(years, RY[0], label='Alabama')
legend()
```

```
<matplotlib.legend.Legend at 0x7febfb9b7278>
```



Spaghetti Plot

```
for row in RY:
    plot(years, row)
```



Kernel Density (univariate, aspatial)

```
from scipy.stats.kde import gaussian_kde
```

```
density = gaussian_kde(Y[:,0])
```

```
Y[:,0]
```

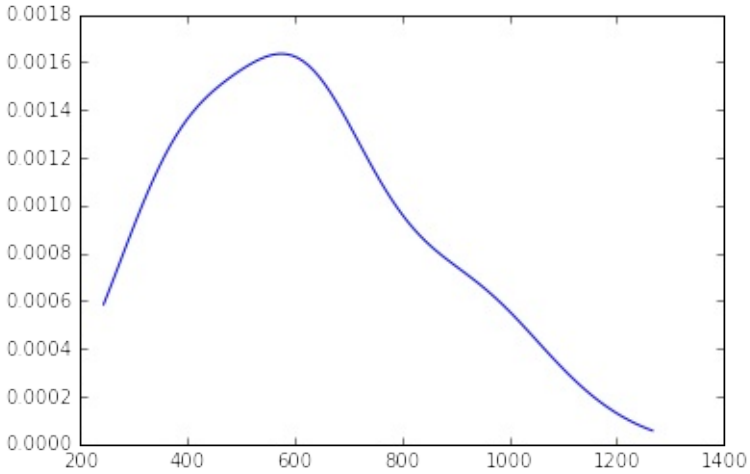
```
array([ 323.,  600.,  310.,  991.,  634., 1024., 1032.,  518.,
        347.,  507.,  948.,  607.,  581.,  532.,  393.,  414.,
        601.,  768.,  906.,  790.,  599.,  286.,  621.,  592.,
        596.,  868.,  686.,  918.,  410., 1152.,  332.,  382.,
        771.,  455.,  668.,  772.,  874.,  271.,  426.,  378.,
        479.,  551.,  634.,  434.,  741.,  460.,  673.,  675.])
```

```
density = gaussian_kde(Y[:,0])
```

```
minY0 = Y[:,0].min()*.90
maxY0 = Y[:,0].max()*1.10
x = np.linspace(minY0, maxY0, 100)
```

```
plot(x,density(x))
```

```
[<matplotlib.lines.Line2D at 0x7febfb841be0>]
```

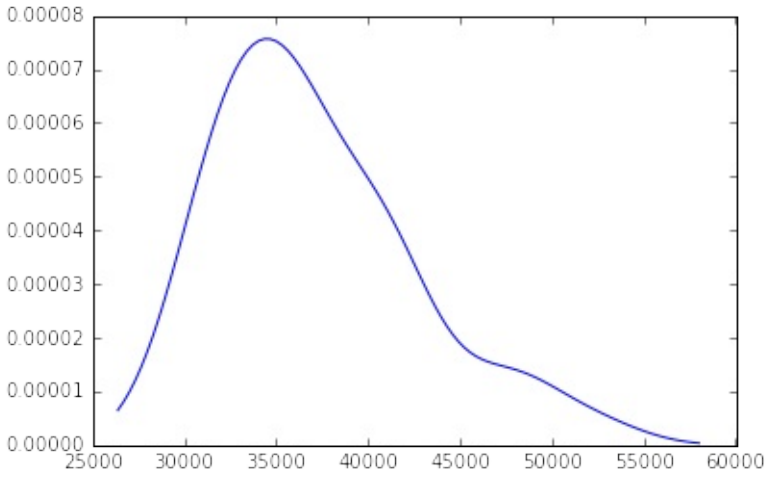



```
d2009 = gaussian_kde(Y[:,-1])
```

```
minY0 = Y[:,-1].min()*.90
maxY0 = Y[:,-1].max()*1.10
x = np.linspace(minY0, maxY0, 100)
```

```
plot(x,d2009(x))
```

```
[<matplotlib.lines.Line2D at 0x7febfb861748>]
```




```
minR0 = RY.min()
```

```
maxR0 = RY.max()
```

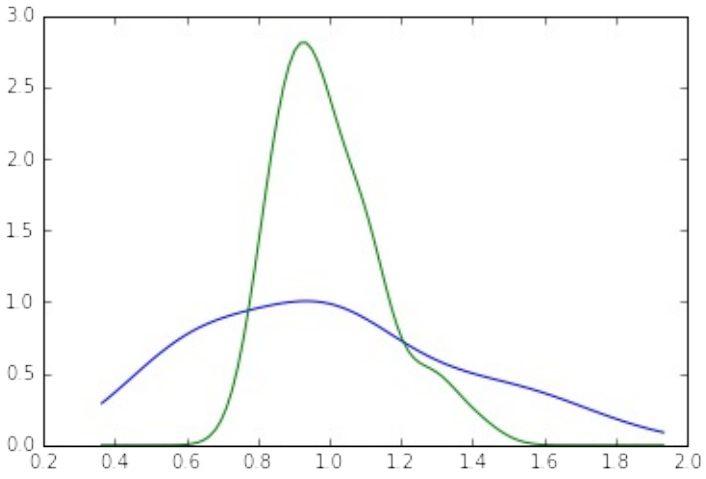
```
x = np.linspace(minR0, maxR0, 100)
```

```
d1929 = gaussian_kde(RY[:,0])
```

```
d2009 = gaussian_kde(RY[:,-1])
```

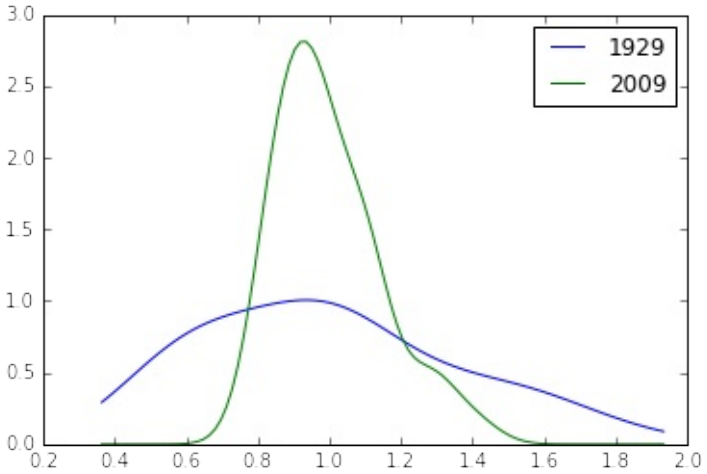
```
plot(x, d1929(x))
plot(x, d2009(x))
```

```
[<matplotlib.lines.Line2D at 0x7febfb91be10>]
```

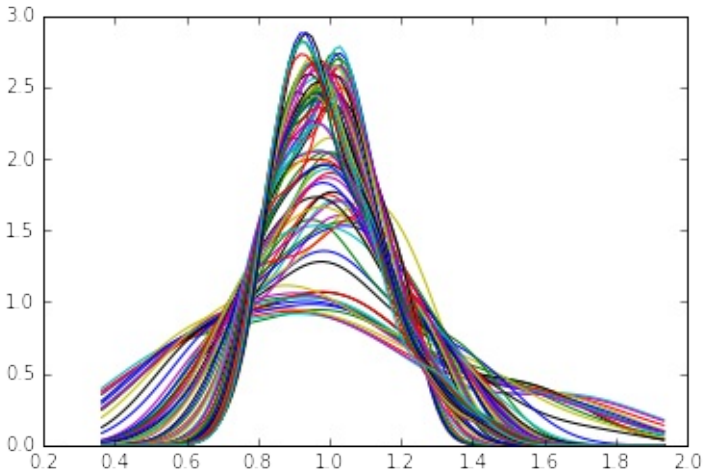


```
plot(x, d1929(x), label='1929')
plot(x, d2009(x), label='2009')
legend()
```

```
<matplotlib.legend.Legend at 0x7febfb8c8128>
```

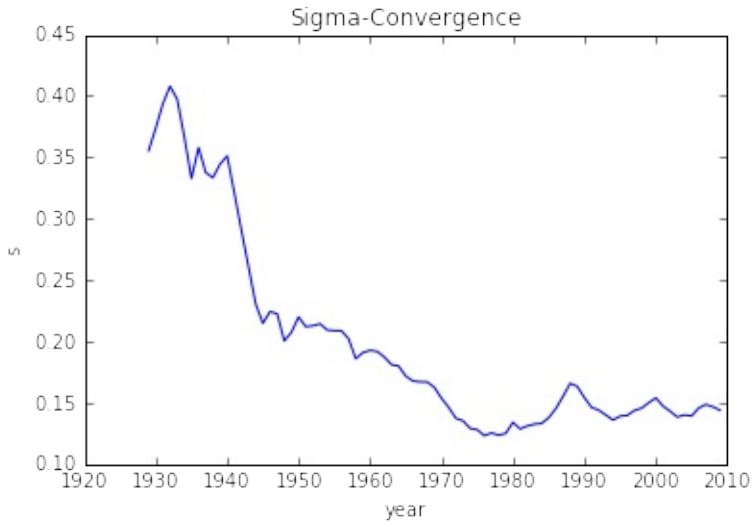



```
for cs in RY.T: # take cross sections
    plot(x, gaussian_kde(cs)(x))
```




```
sigma = RY.std(axis=0)
plot(years, sigma)
ylabel('s')
xlabel('year')
title("Sigma-Convergence")
```

```
<matplotlib.text.Text at 0x7febf9c9f780>
```



So the distribution is becoming less dispersed over time.

But what about internal mixing? Do poor (rich) states remain poor (rich), or is there movement within the distribuiton over time?

Markov Chains

```
c = np.array([['b','a','c'],
['c','c','a'],
['c','b','c'],
['a','a','b'],
['a','b','c']])
```

```
c
```

```
array([['b', 'a', 'c'],
       ['c', 'c', 'a'],
       ['c', 'b', 'c'],
       ['a', 'a', 'b'],
       ['a', 'b', 'c']],
      dtype='<U1')
```

```
m = ps.Markov(c)
```

```
m.classes
```

```
array(['a', 'b', 'c'],
      dtype='<U1')
```

```
m.transitions
```

```
array([[ 1.,  2.,  1.],
       [ 1.,  0.,  2.],
       [ 1.,  1.,  1.]])
```

```
m.p
```

```
matrix([[ 0.25      ,  0.5       ,  0.25      ],
        [ 0.33333333,  0.        ,  0.66666667],
        [ 0.33333333,  0.33333333,  0.33333333]])
```

State Per Capita Incomes

```
f = ps.open(ps.examples.get_path("usjoin.csv"))
pci = np.array([f.by_col[str(y)] for y in range(1929,2010)])
pci.shape
```

```
(81, 48)
```

Put series into cross-sectional quintiles (i.e., quintiles for each year)

```
q5 = np.array([ps.Quantiles(y).yb for y in pci]).transpose()
```

```
q5.shape
```

```
(48, 81)
```

```
q5[:,0]
```

```
array([0, 2, 0, 4, 2, 4, 4, 1, 0, 1, 4, 2, 2, 1, 0, 1, 2, 3, 4, 4, 2, 0, 2,
       2, 2, 4, 3, 4, 0, 4, 0, 0, 3, 1, 3, 3, 4, 0, 1, 0, 1, 2, 2, 1, 3, 1,
       3, 3])
```

```
pci.shape
```

```
(81, 48)
```

```
pci[0]
```

```
array([ 323,  600,  310,  991,  634, 1024, 1032,  518,  347,  507,  948,
        607,  581,  532,  393,  414,  601,  768,  906,  790,  599,  286,
        621,  592,  596,  868,  686,  918,  410, 1152,  332,  382,  771,
        455,  668,  772,  874,  271,  426,  378,  479,  551,  634,  434,
        741,  460,  673,  675])
```

we are looping over the rows of y which is ordered Txn (rows are cross sections, row 0 is the cross-section for period 0

```
m5 = ps.Markov(q5)
```

```
m5.classes
```

```
array([0, 1, 2, 3, 4])
```

```
m5.transitions
```

```
array([[ 729.,   71.,    1.,    0.,    0.],
       [  72.,  567.,   80.,    3.,    0.],
       [   0.,   81.,  631.,   86.,    2.],
       [   0.,    3.,   86.,  573.,   56.],
       [   0.,    0.,    1.,   57.,  741.]])
```

```
m5.p
```

```
matrix([[ 0.91011236,  0.0886392 ,  0.00124844,  0.        ,  0.        ],
        [ 0.09972299,  0.78531856,  0.11080332,  0.00415512,  0.        ],
        [ 0.        ,  0.10125   ,  0.78875   ,  0.1075    ,  0.0025    ],
        [ 0.        ,  0.00417827,  0.11977716,  0.79805014,  0.07799443],
        [ 0.        ,  0.        ,  0.00125156,  0.07133917,  0.92740926]])
```

```
m5.steady_state
```

```
matrix([[ 0.20774716],
        [ 0.18725774],
        [ 0.20740537],
        [ 0.18821787],
        [ 0.20937187]])
```

```
fmpt = ps.ergodic.fmpt(m5.p)
```

```
fmpt
```

```
matrix([[  4.81354357,  11.50292712,  29.60921231,  53.38594954,
        103.59816743],
       [ 42.04774505,   5.34023324,  18.74455332,  42.50023268,
         92.71316899],
       [ 69.25849753,  27.21075248,   4.82147603,  25.27184624,
         75.43305672],
       [ 84.90689329,  42.85914824,  17.18082642,   5.31299186,
         51.60953369],
       [ 98.41295543,  56.36521038,  30.66046735,  14.21158356,
          4.77619083]])
```

For a state with income in the first quintile, it takes on average 11.5 years for it to first enter the second quintile, 29.6 to get to the third quintile, 53.4 years to enter the fourth, and 103.6 years to reach the richest quintile.

But, this approach assumes the movement of a state in the income distribution is independent of the movement of its neighbors or the position of the neighbors in the distribution. Does spatial context matter?

Dynamics of Spatial Dependence

Read in a GAL file to construct our W

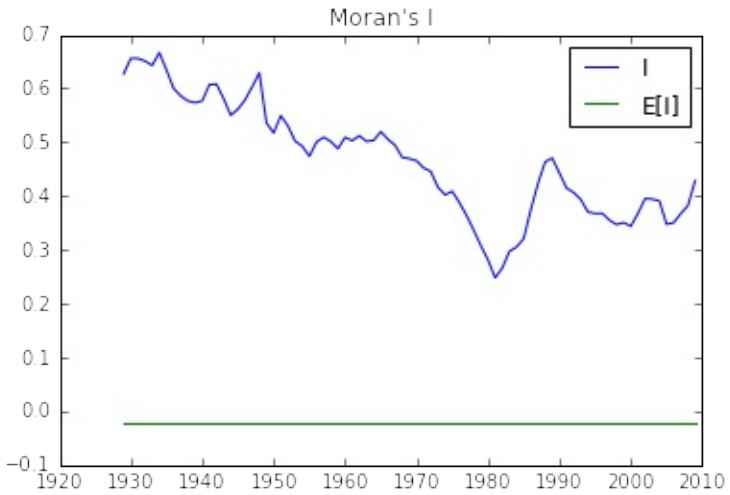
```
w = ps.open(ps.examples.get_path("states48.gal")).read()

w.n

w.transform = 'R'
```

```
mits = [ps.Moran(cs, w) for cs in RY.T]
```

```
res = np.array([(m.I, m.EI, m.p_sim, m.z_sim) for m in mits])
```

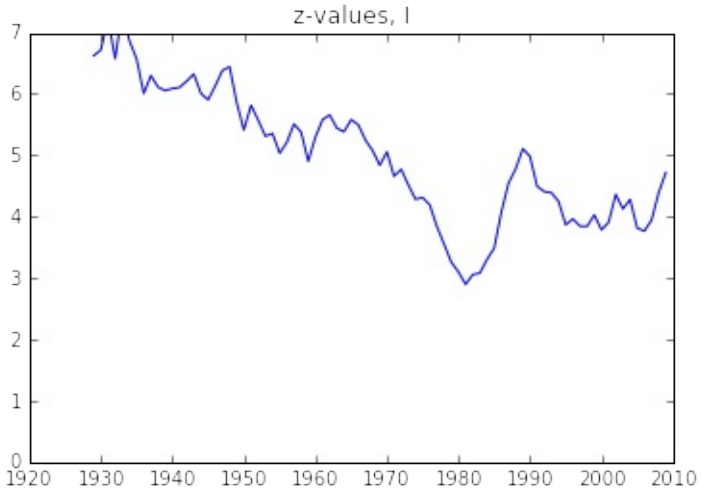
```
plot(years, res[:,0], label='I')
plot(years, res[:,1], label='E[I]')
title("Moran's I")
legend()
```

```
<matplotlib.legend.Legend at 0x7febf9bdf240>
```




```
plot(years, res[:,-1])
ylim(0,7.0)
title('z-values, I')
```

```
<matplotlib.text.Text at 0x7febf9cdc160>
```



Spatial Markov

```
pci.shape
```

```
(81, 48)
```

```
pci = pci.T
```

```
pci.shape
```

```
(48, 81)
```

```
rpci = pci / pci.mean(axis=0)
```

```
rpci[:,0]
```

```
array([ 0.5250254 ,  0.97527938,  0.50389434,  1.61083644,  1.03054521,
        1.6644768 ,  1.67748053,  0.8419912 ,  0.56403657,  0.82411107,
        1.54094142,  0.98665764,  0.94439553,  0.86474771,  0.63880799,
        0.67294277,  0.97690484,  1.2483576 ,  1.47267186,  1.28411785,
        0.97365391,  0.46488317,  1.00941416,  0.96227565,  0.96877751,
        1.41090417,  1.11506942,  1.49217745,  0.66644091,  1.8725364 ,
        0.53965459,  0.62092787,  1.253234  ,  0.73958686,  1.08581104,
        1.25485946,  1.42065696,  0.44050119,  0.69244836,  0.61442601,
        0.77859804,  0.89563156,  1.03054521,  0.70545208,  1.20447003,
        0.74771419,  1.09393837,  1.0971893 ])
```

```
rpci[:,0].mean()
```

```
0.99999999999999989
```

```
sm = ps.Spatial_Markov(rpci, w, fixed=True, k=5)
```

```
sm.p
```

```
matrix([[ 0.91461837,  0.07503234,  0.00905563,  0.00129366,  0.        ],
        [ 0.06570302,  0.82654402,  0.10512484,  0.00131406,  0.00131406],
        [ 0.00520833,  0.10286458,  0.79427083,  0.09505208,  0.00260417],
        [ 0.        ,  0.00913838,  0.09399478,  0.84856397,  0.04830287],
        [ 0.        ,  0.        ,  0.        ,  0.06217617,  0.93782383]])
```

```
for p in sm.P:
    print(p)
```

```
[[ 0.96341463  0.0304878   0.00609756  0.        0.        ]
 [ 0.06040268  0.83221477  0.10738255  0.        0.        ]
 [ 0.        0.14        0.74        0.12      0.        ]
 [ 0.        0.03571429  0.32142857  0.57142857  0.07142857]
 [ 0.        0.        0.        0.16666667  0.83333333]]
[[ 0.79831933  0.16806723  0.03361345  0.        0.        ]
 [ 0.0754717   0.88207547  0.04245283  0.        0.        ]
 [ 0.00537634  0.06989247  0.8655914   0.05913978  0.        ]
 [ 0.        0.        0.06372549  0.90196078  0.03431373]
 [ 0.        0.        0.        0.19444444  0.80555556]]
[[ 0.84693878  0.15306122  0.        0.        0.        ]
 [ 0.08133971  0.78947368  0.1291866   0.        0.        ]
 [ 0.00518135  0.0984456   0.79274611  0.0984456   0.00518135]
 [ 0.        0.        0.09411765  0.87058824  0.03529412]
 [ 0.        0.        0.        0.10204082  0.89795918]]
[[ 0.8852459   0.09836066  0.        0.01639344  0.        ]
 [ 0.03875969  0.81395349  0.13953488  0.        0.00775194]
 [ 0.0049505   0.09405941  0.77722772  0.11881188  0.0049505 ]
 [ 0.        0.02339181  0.12865497  0.75438596  0.09356725]
 [ 0.        0.        0.        0.09661836  0.90338164]]
[[ 0.33333333  0.66666667  0.        0.        0.        ]
 [ 0.0483871   0.77419355  0.16129032  0.01612903  0.        ]
 [ 0.01149425  0.16091954  0.74712644  0.08045977  0.        ]
 [ 0.        0.01036269  0.06217617  0.89637306  0.03108808]
 [ 0.        0.        0.        0.02352941  0.97647059]]
```

```
sm.S
```

```
array([[ 0.43509425,  0.2635327 ,  0.20363044,  0.06841983,  0.02932278],
       [ 0.13391287,  0.33993305,  0.25153036,  0.23343016,  0.04119356],
       [ 0.12124869,  0.21137444,  0.2635101 ,  0.29013417,  0.1137326 ],
       [ 0.0776413 ,  0.19748806,  0.25352636,  0.22480415,  0.24654013],
       [ 0.01776781,  0.19964349,  0.19009833,  0.25524697,  0.3372434 ]])
```

```
for f in sm.F:
    print(f)
```

```
[[  2.29835259  28.95614035  46.14285714  80.80952381  279.42857143]
 [ 33.86549708   3.79459555  22.57142857  57.23809524  255.85714286]
 [ 43.60233918   9.73684211   4.91085714  34.66666667  233.28571429]
 [ 46.62865497  12.76315789   6.25714286  14.61564626  198.61904762]
 [ 52.62865497  18.76315789  12.25714286   6.         34.1031746 ]]
[[  7.46754205   9.70574606  25.76785714  74.53116883  194.23446197]
 [ 27.76691978   2.94175577  24.97142857  73.73474026  193.4380334 ]
 [ 53.57477715  28.48447637   3.97566318  48.76331169  168.46660482]
 [ 72.03631562  46.94601483  18.46153846   4.28393653  119.70329314]
 [ 77.17917276  52.08887197  23.6043956    5.14285714   24.27564033]]
[[  8.24751154   6.53333333  18.38765432  40.70864198  112.76732026]
 [ 47.35040872   4.73094099  11.85432099  34.17530864  106.23398693]
 [ 69.42288828  24.76666667   3.794921    22.32098765   94.37966594]
 [ 83.72288828  39.06666667  14.3         3.44668119   76.36702977]
 [ 93.52288828  48.86666667  24.1         9.8          8.79255406]]
[[ 12.87974382  13.34847151  19.83446328  28.47257282  55.82395142]
 [ 99.46114206   5.06359731  10.54545198  23.05133495  49.68944423]
 [117.76777159  23.03735526   3.94436301  15.0843986   43.57927247]
 [127.89752089  32.4393006   14.56853107   4.44831643  31.63099455]
 [138.24752089  42.7893006   24.91853107  10.35         4.05613474]]
[[  56.2815534    1.5        10.57236842  27.02173913  110.54347826]
 [  82.9223301    5.00892857   9.07236842  25.52173913  109.04347826]
 [  97.17718447  19.53125     5.26043557  21.42391304  104.94565217]
 [ 127.1407767   48.74107143  33.29605263   3.91777427   83.52173913]
 [ 169.6407767   91.24107143  75.79605263  42.5          2.96521739]]
```

LISA Markov

```
lm = ps.LISA_Markov(pci,w)
```

```
lm.classes
```

```
array([1, 2, 3, 4])
```

```
lm.transitions
```

```
array([[ 1.08700000e+03,  4.40000000e+01,  4.00000000e+00,
         3.40000000e+01],
       [ 4.10000000e+01,  4.70000000e+02,  3.60000000e+01,
         1.00000000e+00],
       [ 5.00000000e+00,  3.40000000e+01,  1.42200000e+03,
         3.90000000e+01],
       [ 3.00000000e+01,  1.00000000e+00,  4.00000000e+01,
         5.52000000e+02]])
```

```
np.set_printoptions(3, suppress=True)
```

```
lm.transitions
```

```
array([[ 1087.,    44.,     4.,    34.],
       [   41.,   470.,    36.,     1.],
       [    5.,    34.,  1422.,    39.],
       [   30.,     1.,    40.,   552.]])
```

```
lm.p
```

```
matrix([[ 0.93 ,  0.038,  0.003,  0.029],
        [ 0.075,  0.858,  0.066,  0.002],
        [ 0.003,  0.023,  0.948,  0.026],
        [ 0.048,  0.002,  0.064,  0.886]])
```

```
lm.steady_state
```

```
matrix([[ 0.286],
        [ 0.142],
        [ 0.405],
        [ 0.168]])
```

```
ps.ergodic.fmpt(lm.p)
```

```
matrix([[  3.501,  37.93 ,  40.558,  43.174],
        [ 31.728,   7.047,  28.682,  49.915],
        [ 52.445,  47.421,   2.47 ,  43.756],
        [ 38.768,  51.518,  26.316,   5.969]])
```

Test of independence of own chains and lag chains

```
lm.transitions
```

```
array([[ 1087.,    44.,     4.,    34.],
       [   41.,   470.,    36.,     1.],
       [    5.,    34.,  1422.,    39.],
       [   30.,     1.,    40.,   552.]])
```

```
lm.expected_t
```

```
array([[ 1123.281,    11.538,     0.348,    33.834],
       [    3.503,   528.474,    15.918,     0.106],
       [    0.154,    23.216,  1466.907,     9.723],
       [    9.608,     0.099,     6.235,   607.058]])
```

```
lm.chi_2
```

```
(1058.2079036003051, 0.0, 9)
```

Part II

Point Patterns

> IPYNB

This notebook covers a brief introduction on how to visualize and analyze point patterns. To demonstrate this, we will use a dataset of all the AirBnb listings in the city of Austin (check the Data section for more information about the dataset).

Before anything, let us load up the libraries we will use:

```
%matplotlib inline

import numpy as np
import pandas as pd
import geopandas as gpd
import seaborn as sns
import matplotlib.pyplot as plt
import mplleaflet as mpll
```

```
/home/dani/anaconda/envs/pydata/lib/python2.7/site-packages/matplotlib/font_manager.py:273: UserW
arning: Matplotlib is building the font cache using fc-list. This may take a moment.
  warnings.warn('Matplotlib is building the font cache using fc-list. This may take a moment.')
```

Data preparation

Let us first set the paths to the datasets we will be using:

```
# Adjust this to point to the right file in your computer
listings_link = '/home/dani/Desktop/listings.csv'
```

The core dataset we will use is `listings.csv`, which contains a lot of information about each individual location listed at AirBnb within Austin:

```
lst = pd.read_csv(listings_link)
lst.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5835 entries, 0 to 5834
Data columns (total 92 columns):
id                              5835 non-null int64
listing_url                     5835 non-null object
scrape_id                       5835 non-null int64
last_scraped                    5835 non-null object
name                            5835 non-null object
summary                         5373 non-null object
space                           4475 non-null object
```

```
description                       5832 non-null object
experiences_offered               5835 non-null object
neighborhood_overview             3572 non-null object
notes                             2413 non-null object
transit                           3492 non-null object
thumbnail_url                     5542 non-null object
medium_url                        5542 non-null object
picture_url                       5835 non-null object
xl_picture_url                    5542 non-null object
host_id                           5835 non-null int64
host_url                          5835 non-null object
host_name                         5820 non-null object
host_since                        5820 non-null object
host_location                     5810 non-null object
host_about                        3975 non-null object
host_response_time                4177 non-null object
host_response_rate                4177 non-null object
host_acceptance_rate              3850 non-null object
host_is_superhost                 5820 non-null object
host_thumbnail_url                5820 non-null object
host_picture_url                  5820 non-null object
host_neighbourhood                4977 non-null object
host_listings_count               5820 non-null float64
host_total_listings_count         5820 non-null float64
host_verifications                5835 non-null object
host_has_profile_pic              5820 non-null object
host_identity_verified            5820 non-null object
street                            5835 non-null object
neighbourhood                     4800 non-null object
neighbourhood_cleansed            5835 non-null int64
neighbourhood_group_cleansed      0 non-null float64
city                              5835 non-null object
state                             5835 non-null object
zipcode                           5810 non-null float64
market                            5835 non-null object
smart_location                    5835 non-null object
country_code                      5835 non-null object
country                           5835 non-null object
latitude                          5835 non-null float64
longitude                         5835 non-null float64
is_location_exact                 5835 non-null object
property_type                     5835 non-null object
room_type                         5835 non-null object
accommodates                      5835 non-null int64
bathrooms                         5789 non-null float64
bedrooms                          5829 non-null float64
beds                              5812 non-null float64
bed_type                          5835 non-null object
amenities                         5835 non-null object
square_feet                       302 non-null float64
price                             5835 non-null object
weekly_price                      2227 non-null object
monthly_price                     1717 non-null object
```

```
security_deposit                 2770 non-null object
cleaning_fee                     3587 non-null object
guests_included                  5835 non-null int64
extra_people                     5835 non-null object
minimum_nights                   5835 non-null int64
maximum_nights                   5835 non-null int64
calendar_updated                 5835 non-null object
has_availability                 5835 non-null object
availability_30                  5835 non-null int64
availability_60                  5835 non-null int64
availability_90                  5835 non-null int64
availability_365                 5835 non-null int64
calendar_last_scraped            5835 non-null object
number_of_reviews                5835 non-null int64
first_review                     3827 non-null object
last_review                      3829 non-null object
review_scores_rating             3789 non-null float64
review_scores_accuracy           3776 non-null float64
review_scores_cleanliness        3778 non-null float64
review_scores_checkin            3778 non-null float64
review_scores_communication      3778 non-null float64
review_scores_location           3779 non-null float64
review_scores_value              3778 non-null float64
requires_license                 5835 non-null object
license                          1 non-null float64
jurisdiction_names               0 non-null float64
instant_bookable                 5835 non-null object
cancellation_policy              5835 non-null object
require_guest_profile_picture    5835 non-null object
require_guest_phone_verification 5835 non-null object
calculated_host_listings_count   5835 non-null int64
reviews_per_month                3827 non-null float64
dtypes: float64(20), int64(14), object(58)
memory usage: 4.1+ MB
```

It turns out that one record displays a very odd location and, for the sake of the illustration, we will remove it:

```
odd = lst.loc[lst.longitude>-80, ['longitude', 'latitude']]
odd
```

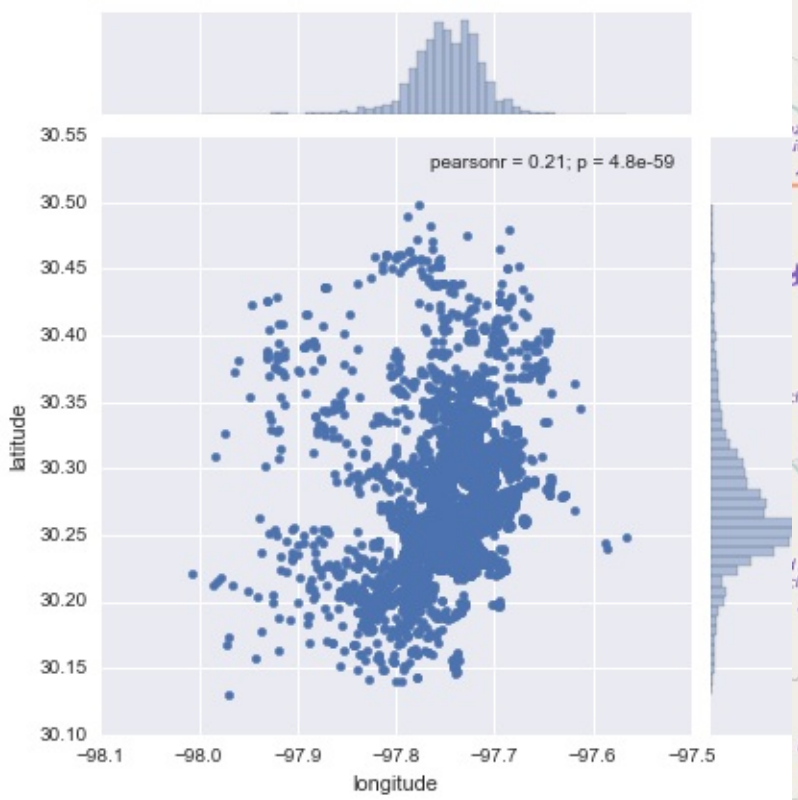
	longitude	latitude
5832	-5.093682	43.214991

```
lst = lst.drop(odd.index)
```

Point Visualization

The most straighforward way to get a first glimpse of the d
their latitude and longitude:

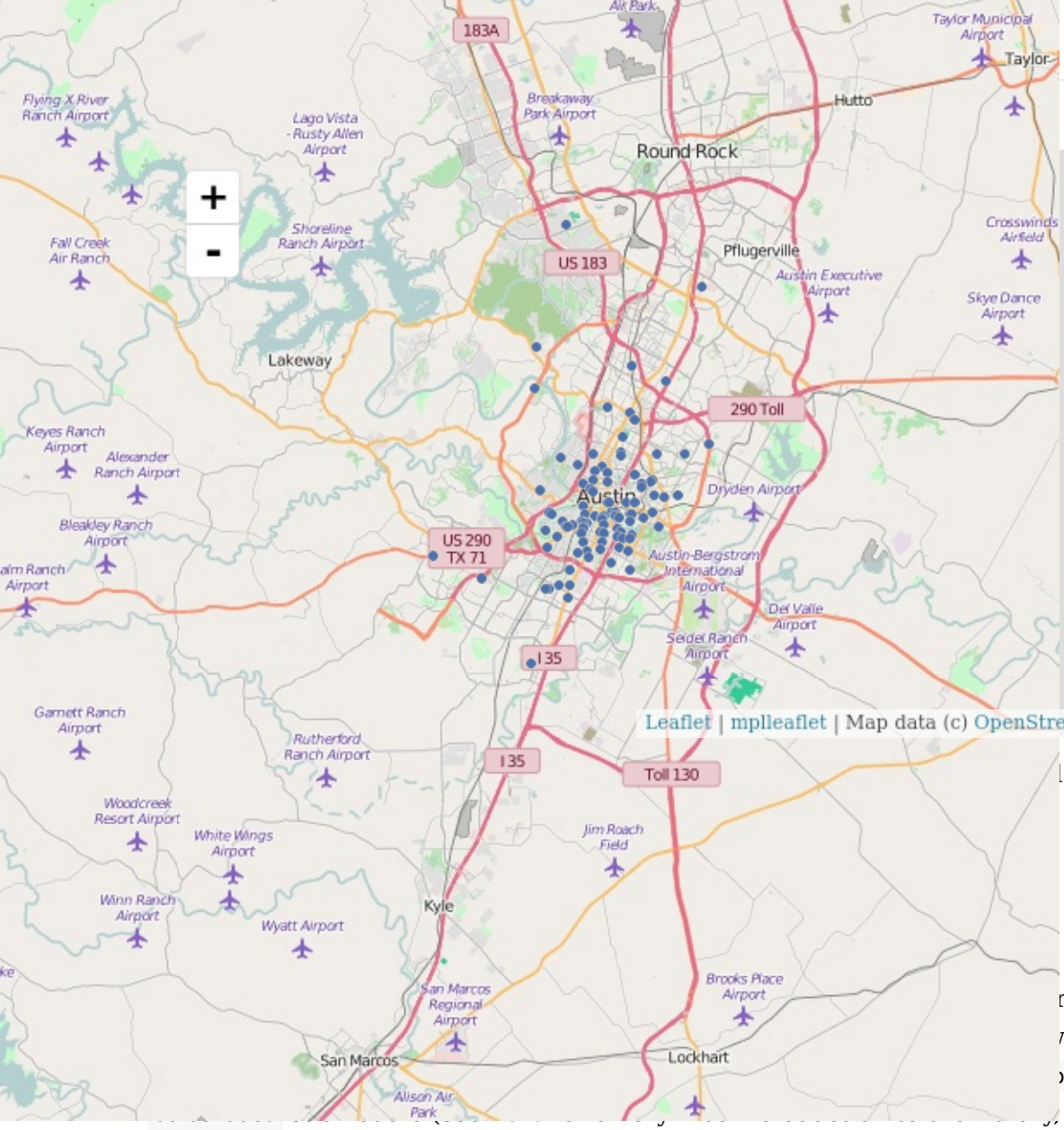
```
sns.jointplot(x="longitude", y="latitude", data=lst);
```



Now this does not neccesarily tell us much about the datas
within Austin. There are two main challenges in interpreting the plot: one, there is lack of context, which means the points are not identifiable over space (unless you are so familiar with lon/lat pairs that they have a clear meaning to you); and two, in the center of the plot, there are so many points that it is hard to tell any patter other than a big blurb of blue.

Let us first focus on the first problem, geographical context. The quickest and easiest way to provide context to this set of points is to overlay a general map. If we had an image with the map or a set of several data sources that we could aggregate to create a map, we could build it from scratch. But in the XXI Century, the easiest is to overlay our point dataset on top of a web map. In this case, we will use Leaflet, and we will convert our underlying `matplotlib` points with `mplleaflet`. The full dataset (+5k observations) is a bit too much for leaflet to plot it directly on screen, so we will obtain a random sample of 100 points:

```
rids = np.arange(lst.shape[0])
np.random.shuffle(rids)
f, ax = plt.subplots(1, figsize=(6, 6))
lst.iloc[rids[:100], :].plot(kind='scatter', x='longitude', y='latitude', \
                             s=30, linewidth=0, ax=ax)
mpll.display(fig=f,)
```



what type of
blurb has to do

robably the
e need to resort
okeh and its
from where this
example has been adapted).

Before we delve into `bokeh`, let us reproject our original data (lon/lat coordinates) into Web Mercator, as `bokeh` will expect them. To do that, we turn the coordinates into a `GeoSeries`:

```
from shapely.geometry import Point
xys_wb = gpd.GeoSeries(lst[['longitude', 'latitude']].apply(Point, axis=1), \
                       crs="+init=epsg:4326")
xys_wb = xys_wb.to_crs(epsg=3857)
x_wb = xys_wb.apply(lambda i: i.x)
y_wb = xys_wb.apply(lambda i: i.y)
```

Now we are ready to setup the plot in `bokeh`:

```
from bokeh.plotting import figure, output_notebook, show
from bokeh.tile_providers import STAMEN_TERRAIN
output_notebook()

minx, miny, maxx, maxy = xys_wb.total_bounds
y_range = miny, maxy
x_range = minx, maxx

def base_plot(tools='pan,wheel_zoom,reset',plot_width=600, plot_height=400, **plot_args):
    p = figure(tools=tools, plot_width=plot_width, plot_height=plot_height,
        x_range=x_range, y_range=y_range, outline_line_color=None,
        min_border=0, min_border_left=0, min_border_right=0,
        min_border_top=0, min_border_bottom=0, **plot_args)

    p.axis.visible = False
    p.xgrid.grid_line_color = None
    p.ygrid.grid_line_color = None
    return p

options = dict(line_color=None, fill_color='#800080', size=4)
```

contributors

```
<div class="bk-banner">
    <a href="http://bokeh.pydata.org" target="_blank" class="bk-logo bk-logo-small bk-logo-notebook">
</a>
    <span id="a0125a61-aa40-4d88-9db2-d2b800c39acb">Loading BokehJS ...</span>
</div>
```

And good to go for mapping!

```
# NOTE: `show` turned off to be able to compile the website,
#       comment out the last line of this cell for rendering.
p = base_plot()
p.add_tile(STAMEN_TERRAIN)
p.circle(x=x_wb, y=y_wb, **options)
#show(p)
```

```
<bokeh.models.renderers.GlyphRenderer at 0x7f6cf6ce8c90>
```

As you can quickly see, `bokeh` is substantially faster at rendering larger amounts of data.

The second problem we have spotted with the first scatter is that, when the number of points grows, at some point it becomes impossible to discern anything other than a big blur of color. To some extent, interactivity gets at that problem by allowing the user to zoom in until every point is an entity on its own. However, there exist techniques that allow to summarize the data to be able to capture the overall pattern at once. Traditionally, kernel density estimation (KDE) has been one of the most common solutions by approximating a

continuous surface of point intensity. In this context, however, we will explore a more recent alternative suggested by the `datashader` library (see the paper if interested in more details).

Arguably, our dataset is not large enough to justify the use of a reduction technique like datashader, but we will create the plot for the sake of the illustration. Keep in mind, the usefulness of this approach increases the more points you need to be plotting.

```
# NOTE: `show` turned off to be able to compile the website,
#       comment out the last line of this cell for rendering.

import datashader as ds
from datashader.callbacks import InteractiveImage
from datashader.colors import viridis
from datashader import transfer_functions as tf
from bokeh.tile_providers import STAMEN_TONER

p = base_plot()
p.add_tile(STAMEN_TONER)

pts = pd.DataFrame({'x': x_wb, 'y': y_wb})
pts['count'] = 1
def create_image90(x_range, y_range, w, h):
    cvs = ds.Canvas(plot_width=w, plot_height=h, x_range=x_range, y_range=y_range)
    agg = cvs.points(pts, 'x', 'y',  ds.count('count'))
    img = tf.interpolate(agg.where(agg > np.percentile(agg,90)), \
                         cmap=viridis, how='eq_hist')
    return tf.dynspread(img, threshold=0.1, max_px=4)

#InteractiveImage(p, create_image90)
```

The key advandage of `datashader` is that is decouples the point processing from the plotting. That is the bit that allows it to be scalable to truly large datasets (e.g. millions of points). Essentially, the approach is based on generating a very fine grid, counting points within pixels, and encoding the count into a color scheme. In our map, this is not particularly effective because we do not have too many points (the previous plot is probably a more effective one) and esssentially there is a pixel per location of every point. However, hopefully this example shows how to create this kind of scalable maps.

Centrography and distance based statistics

Spatial Clustering

```
%matplotlib inline

import pandas as pd
```

```
/home/dani/anaconda/envs/pydata/lib/python2.7/site-packages/matplotlib/font_manager.py:273: UserW
arning: Matplotlib is building the font cache using fc-list. This may take a moment.
  warnings.warn('Matplotlib is building the font cache using fc-list. This may take a moment.')
```

```
link = '/home/dani/Desktop/listings.csv'
db = pd.read_csv(link)
db.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5835 entries, 0 to 5834
Data columns (total 92 columns):
id                              5835 non-null int64
listing_url                     5835 non-null object
scrape_id                       5835 non-null int64
last_scraped                    5835 non-null object
name                            5835 non-null object
summary                         5373 non-null object
space                           4475 non-null object
description                     5832 non-null object
experiences_offered             5835 non-null object
neighborhood_overview           3572 non-null object
notes                           2413 non-null object
transit                         3492 non-null object
thumbnail_url                   5542 non-null object
medium_url                      5542 non-null object
picture_url                     5835 non-null object
xl_picture_url                  5542 non-null object
host_id                         5835 non-null int64
host_url                        5835 non-null object
host_name                       5820 non-null object
host_since                      5820 non-null object
host_location                   5810 non-null object
host_about                      3975 non-null object
host_response_time              4177 non-null object
host_response_rate              4177 non-null object
host_acceptance_rate            3850 non-null object
host_is_superhost               5820 non-null object
host_thumbnail_url              5820 non-null object
host_picture_url                5820 non-null object
host_neighbourhood              4977 non-null object
host_listings_count             5820 non-null float64
```

```
host_total_listings_count          5820 non-null float64
host_verifications                 5835 non-null object
host_has_profile_pic               5820 non-null object
host_identity_verified             5820 non-null object
street                             5835 non-null object
neighbourhood                      4800 non-null object
neighbourhood_cleansed             5835 non-null int64
neighbourhood_group_cleansed       0 non-null float64
city                               5835 non-null object
state                              5835 non-null object
zipcode                            5810 non-null float64
market                             5835 non-null object
smart_location                     5835 non-null object
country_code                       5835 non-null object
country                            5835 non-null object
latitude                           5835 non-null float64
longitude                          5835 non-null float64
is_location_exact                  5835 non-null object
property_type                      5835 non-null object
room_type                          5835 non-null object
accommodates                       5835 non-null int64
bathrooms                          5789 non-null float64
bedrooms                           5829 non-null float64
beds                               5812 non-null float64
bed_type                           5835 non-null object
amenities                          5835 non-null object
square_feet                        302 non-null float64
price                              5835 non-null object
weekly_price                       2227 non-null object
monthly_price                      1717 non-null object
security_deposit                   2770 non-null object
cleaning_fee                       3587 non-null object
guests_included                    5835 non-null int64
extra_people                       5835 non-null object
minimum_nights                     5835 non-null int64
maximum_nights                     5835 non-null int64
calendar_updated                   5835 non-null object
has_availability                   5835 non-null object
availability_30                    5835 non-null int64
availability_60                    5835 non-null int64
availability_90                    5835 non-null int64
availability_365                   5835 non-null int64
calendar_last_scraped              5835 non-null object
number_of_reviews                  5835 non-null int64
first_review                       3827 non-null object
last_review                        3829 non-null object
review_scores_rating               3789 non-null float64
review_scores_accuracy             3776 non-null float64
review_scores_cleanliness          3778 non-null float64
review_scores_checkin              3778 non-null float64
review_scores_communication        3778 non-null float64
review_scores_location             3779 non-null float64
review_scores_value                3778 non-null float64
```

```
requires_license                  5835 non-null object
license                           1 non-null float64
jurisdiction_names                0 non-null float64
instant_bookable                  5835 non-null object
cancellation_policy               5835 non-null object
require_guest_profile_picture     5835 non-null object
require_guest_phone_verification  5835 non-null object
calculated_host_listings_count    5835 non-null int64
reviews_per_month                 3827 non-null float64
dtypes: float64(20), int64(14), object(58)
memory usage: 4.1+ MB
```

Spatial Regression