Working with Geodata in Stata

Chao Liu*

November 12, 2022

Stata is able to perform some basic analysis using georeferenced data. This note will walk you through four tasks: drawing maps, geocoding, matching locations to polygons, and finding neighboring polygons.[1]

I Drawing Maps

To create a map in Stata, we need two necessary packages: **shp2dta** and **spmap**. You can download them by typing the following in your Stata command line:

```
ssc install shp2dta
ssc install spmap
```

Below, I will draw a map of the share of limited English proficient people at the county level in the U.S.

Step 1: Use **shp2dta** to translate files

```
//use shp2dta to convert
shp2dta using "$data\us_county\cb_2019_us_county_500k.shp", ///
database("$data\us_county\usdb.dta") ///
coordinates("$data\us_county\uscoord.dta") ///
genid(id) replace
```

shp2dta reads a shape (.shp) and dBase (.dbf) file from disk and converts them into Stata datasets. Here I use a shapefile of U.S. counties from https://www.census.gov/cgi-bin/geo/shapefiles/index.php. The shape and dBase files must have the

*Liu: Kellogg School of Management, Northwestern University. Email: chao.liu1@kellogg.northwestern.edu.

[1]Do files and data can be downloaded from https://github.com/chaoliu-kellogg/geodata_stata.

same name and be saved in the same directory. **database()** specifies that we want the database file to be named usdb.dta. **coordinates()** specifies that we want the coordinate file to be named uscoord.dta. **genid()** specifies that we want the ID variable created in usdb.dta to be named id.

Step 2: Merge geodata with the variable you want to plot (e.g. LEP share). In this step, you need the database file that you got in step 1.

```
//merge with the variable you want to plot
use "$data\us_county\usdb.dta", clear
destring STATEFP, replace force
drop if STATEFP > 56
drop if STATEFP == 2
drop if STATEFP == 15
destring GEOID, gen(fips) force
merge 1:1 fips using "$data\county_badeng.dta", keep(3) nogen
```

Step 3: Use **spmap** to draw maps. In this step, you need the coordinate file that you got in step 1.

```
//use spmap to draw maps
format (badeng) %5.2f
spmap badeng using "$data\us_county\uscoord.dta", ///
id(id) cln(5) fcolor(Blues) ndf(gs13) ///
legend(pos(4) size(medium)) title("Share of LEP Population")
graph export "$output\lepshare.png", as(png) replace
```

The above code will generate Figure 1. You can check the help file of **spmap** to see how you modify the map.

II Geocoding

If you search for geocoding in Stata on Google, you'll find several solutions for both forward and reverse geocoding. However, by the time of this writing, most of them are no longer in service.

I find **opencagegeo** a stable tool to do geocoding tasks. It requires an OpenCage Data API key which can be obtained by signing up at https://geocoder.opencagedata.com/users/sign_up. The user can choose among a number of customer plans with different daily rate limits. To use this package in Stata, you will install three required

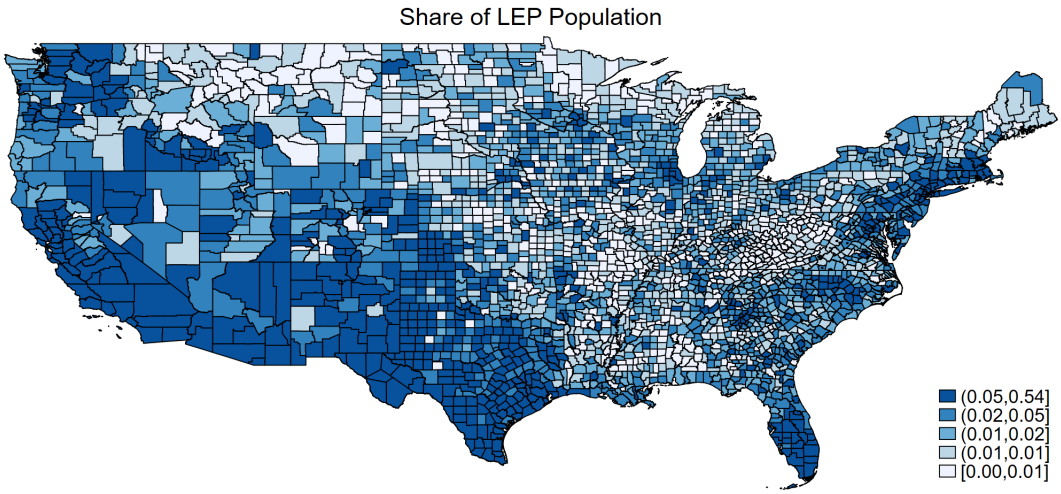

Figure 1. LEP Share in the U.S.

libraries:

```
* Install required libraries:
ssc install opencagegeo
ssc install libjson
ssc install insheetjson
```

Then it is straightforward to use **opencagegeo** to do forward and reverse geocoding.

```
//forward geocoding: from address to coordinate
/* If you have a dataset of addresses stored
in a single string variable 'address' */
opencagegeo, key(YOUR_API_KEY) fulladdress(address)

/* If your addresses are stored in separate variables,
e.g. house number in 'num', street name in 'str',
city in 'city', and country in 'ctry': */
opencagegeo, key(YOUR_API_KEY) number(num) street(str) ///
city(city) country(ctry)

//reverse geocoding: from coordinate to address
/* To geocode coordinates stored in a single
variable 'coords' in the following format:
latitude,longitude */
opencagegeo, key(YOUR_API_KEY) coordinates(coords)
```

```
/* If your coordinates are stored in two
separate variables 'lat' and 'lng' */
opencagegeo, key(YOUR_API_KEY) latitude(lat) longitude(lng)
```

III Matching Locations to Polygons

Suppose you have a list of coordinates for bank branches, how do you know which county each branch is located in. You can solve this problem by using **opencagegeo** to get the detailed address, but a quicker and easier way is to use **geoinpoly**.

After you install this package by "**ssc install geoinpoly**", you can use the following code to determine the county of each bank branch.

```
//use shp2dta to convert
shp2dta using "$data\us_county\cb_2019_us_county_500k.shp", ///
database("$data\us_county\usdb.dta") ///
coordinates("$data\us_county\uscoord.dta") ///
genid(id) replace

//use geoinpoly
use "$data\branch_2019.dta", clear
drop if sims_latitude == . | sims_longitude == .
geoinpoly sims_latitude sims_longitude ///
using "$data\us_county\uscoord.dta"
merge m:1 _ID using "$data\us_county\usdb.dta", keep(1 3) nogen
```

You can get the locations of bank branches in 2019 from https://www7.fdic.gov/sod/dynaDownload.asp?barItem=6. Again, we first use **shp2dta** to convert shapefiles to Stata data files. We then ask **geoinpoly** to use the latitude and longitude information in the bank branch file and the coordinate file that we created by **shp2dta**. The output contains a variable named **_ID**, which indicates the identity of counties. You can further merge the output with the database file (i.e., **usbd.dta**), and then you have FIPS codes for each bank branch.

The data we downloaded from FDIC actually include the county of each bank branch, so we can compare our results to the records provided by FDIC. The accuracy rate of using **geoinpoly** is over 99%.

```
//check
destring GEOID, gen(fips) force
gen correct = stcntybr == fips
tab correct
```

correct	Freq.	Percent	Cum.
0	702	0.81	0.81
1	85,688	99.19	100.00
Total	86,390	100.00	

IV Finding Neighboring Polygons

In this example, I show how you can find neighboring Public Use Microdata Areas (PUMAs) for each PUMA in the 2000 census.[2] The 2000 version of Public Use Microdata Areas (PUMAs) is the lowest level of geography identified in the 2000 census 5% sample and the ACS/PRCS samples from 2005 to 2011. You can download the corresponding boundary files from https://usa.ipums.org/usa/volii/2000pumas.shtml.

Again, we first need to use **shp2dta** to read shapefiles and translate them to Stata data files.

```
shp2dta using "$data\ipums_puma_2000.shp", ///
genid(_ID) data("$data\puma2000.dta") ///
coor("$data\puma2000_coor.dta") replace
```

The main idea is to find points that exist in multiple polygons. We can find these points in **puma2000_coor.dta** using the following code:

```
use "$data\puma2000_coor.dta", clear
/* remove duplicates within each polygon
and missing coordinates that indicate
the start of a new polygon */
drop if mi(_Y, _X)
```

[2] I thank Robert Picard for sharing this method. Original source: https://www.statalist.org/forums/forum/general-stata-discussion/general/1377956-creating-neighbor-information-variables-in-country-panel-data.

```
gduplicates drop _Y _X _ID, force
/* reduce to coordinates that
appear in more than one polygon */
bys _Y _X: keep if _N > 1
```

Once you have a list of points that are located in boundaries, you only need to form all possible pairs of adjacent PUMAs.

```
/* switch to wide form and reduce to
one obs per conterminous country set */
by _Y _X: gen j = _n
greshape wide _ID, i(_Y _X) j(j)
keep _ID*
gduplicates drop _ID*, force

/* switch back to long form and form
all pairwise combinations within the set */
gen set = _n
greshape long _ID, i(set)
drop if mi(_ID)
drop _j
save "$data\puma_sets.dta", replace
rename _ID _ID_pair
joinby set using "$data\puma_sets.dta"
drop if _ID == _ID_pair

/* remove duplicates and merge
with database to get the name */
bysort _ID _ID_pair: keep if _n == 1
merge m:1 _ID using "$data\puma2000.dta", ///
assert(match using) keep(match) nogen

rename (_ID _ID_pair STATEFIP PUMA GISMATCH) ///
(_ID_pair _ID STATEFIP1 PUMA1 GISMATCH1)
merge m:1 _ID using "$data\puma2000.dta", ///
assert(match using) keep(match) nogen
rename (STATEFIP PUMA GISMATCH) ///
```

(STATEFIP2 PUMA2 GISMATCH2)

We can create two data sets. The first one contains all pairs of neighboring PUMAs, and the second contains neighboring PUMAs in different states.

```
/* final list */
isid GISMATCH1 GISMATCH2, sort
save "$output\neighboring_puma2000.dta", replace

keep if STATEFIP1 != STATEFIP2
save "$output\bordering_puma2000.dta", replace
```