Faxi Yuan · Follow

7 min read · Jan 7

Listen Share

A guide to assign aggregated demographic information with a specific address

Credit: https://givingcompass.org/article/why-demographic-data-is-important-for-funding-purposes

Equity relevant issues such as access equity to critical infrastructures, facilities, groceries and medical centers, etc., are receiving increasing attention form both academia and other public institutions. According to these issues, research questions like which point of interest (POI)s such as a grocery and hospital mainly serve which group of people in terms of their demographic characters need to be resolved. With that, assigning aggregated (e.g., county-, city-, ZIP code-, census tract- and census block group-level) demographic information with a specific point of interest (POI) such as a

grocery and hospital becomes necessary. Here, I am sharing with you a method I developed to assign a specific address with its corresponding aggregated demographic information, including 1) how to stream different demographic variables at different aggregation levels from US Census Database; 2) how to geocode an address and get its geo-coordinate; and 3) how to create a shapefile with geo-coordinates, check the coordinate reference systems (CRSs) of different shapefiles, and do the spatial join of different shapefiles.

Increasing studies in equity relevant issues such as equity of access to critical infrastructures and facilities, healthy food and environment, and medical centers, etc., have brought more attention to the aggregated demographic data. With such aggregated demographic data, researchers can investigate the distribution of critical facilities within a region such as a **ZIP Code Tabulation Area** (ZCTA), census tract (CT), and census block group (CBG), and further explore the impacts of their demographic information on such distributions.

More specifically, given a scenario, where addresses of a set of point of interests (POIs) like grocery stores are known and we want to know which group of people (in terms of their demographic characters) are mainly served by these grocery stores, what shall we do? The first step is to define which aggregation-level demographic data will be used such as ZCTA, CT, or CBG. Secondly, we need to stream the demographic data from census database with our defined aggregated levels. Thirdly, each grocery address will be assigned with a region ID (i.e., GEOIDs as defined by the US Census Data) such as ZIP code, CT GEOID or CBG GEOID. The last step is to join grocery store data with their aggregated demographic data through their region GEOIDs, and perform the analysis.

This tutorial will guide you to assign the aggregated demographic data with a specific address through the following steps. This tutorial mainly serves for researchers who are interested in using US census data from U.S. Census Bureau.

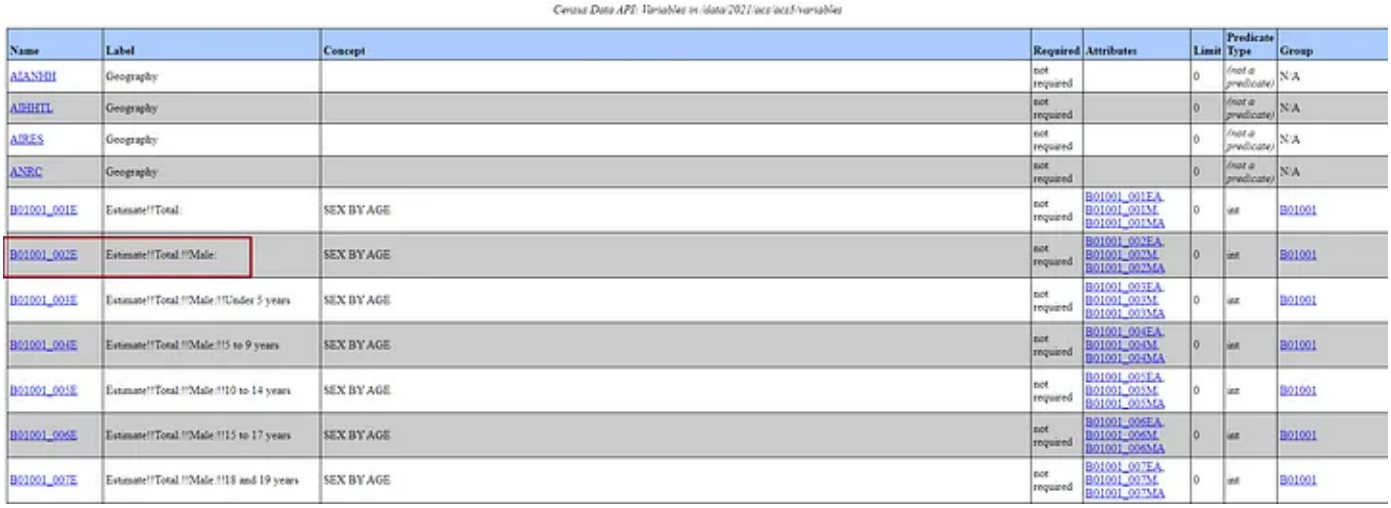
Step 1. Defining aggregated level

U.S. Census Bureau provides aggregated demographic data across country, state, county, city, ZCTA, CT, and CBG. Each level has its own GEOIDs. This tutorial takes a

fine spatial resolution level, CBG, as an example. More details about GEOIDs for different levels can be referred from U.S. Census Bureau.

Step 2. Streaming aggregated-level demographic data from US Census Bureau Database

Figure 1 shows a screenshot of the table head. Red frame highlights the variable name 'B01001_002E' and its label as 'total male'. With this table, users can select to stream the demographic variables per their interests.

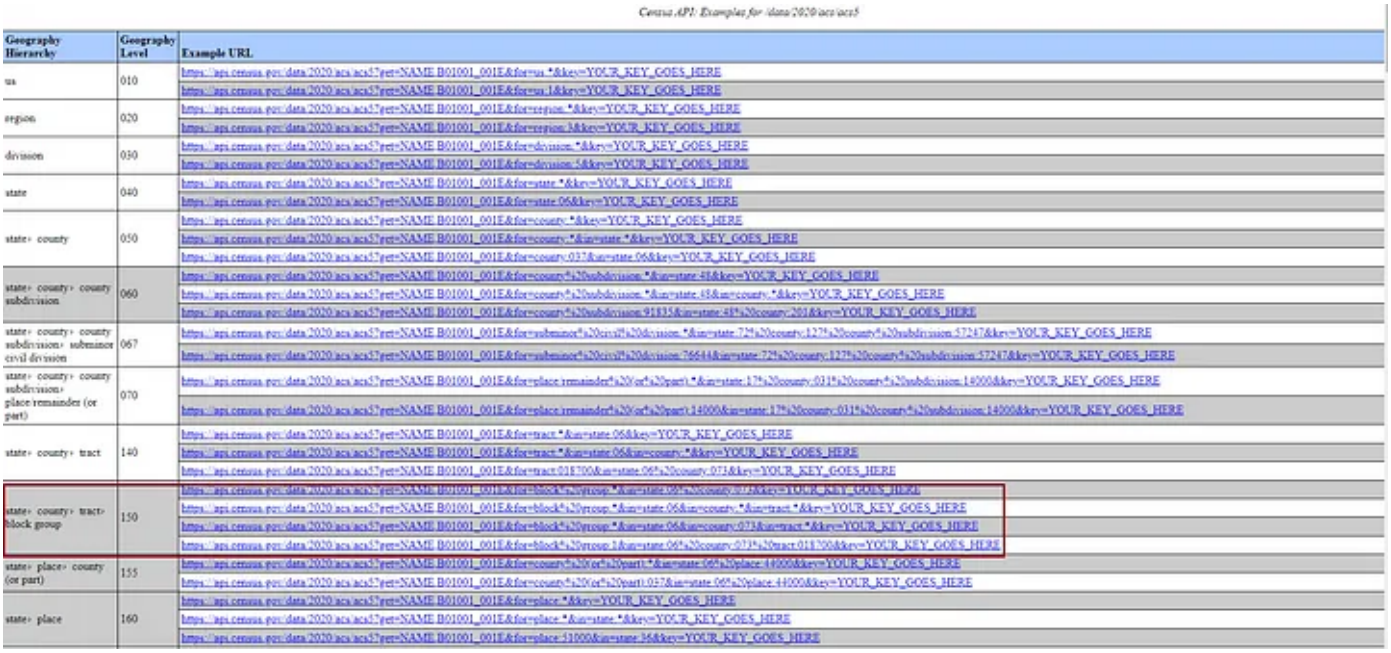


Census Data API: Variables in /data/2021/acs/acs5/variables

Name	Label	Concept	Required	Attributes	Limit	Predicate Type	Group
AIANHH	Geography		not required		0	(not a predicate)	N/A
AIHHTL	Geography		not required		0	(not a predicate)	N/A
AIRES	Geography		not required		0	(not a predicate)	N/A
ANRC	Geography		not required		0	(not a predicate)	N/A
B01001_001E	Estimate!!Total:	SEX BY AGE	not required	B01001_001EA, B01001_001M, B01001_001MA	0	int	B01001
B01001_002E	Estimate!!Total!!Male:	SEX BY AGE	not required	B01001_002EA, B01001_002M, B01001_002MA	0	int	B01001
B01001_003E	Estimate!!Total!!Male!!Under 5 years	SEX BY AGE	not required	B01001_003EA, B01001_003M, B01001_003MA	0	int	B01001
B01001_004E	Estimate!!Total!!Male!!5 to 9 years	SEX BY AGE	not required	B01001_004EA, B01001_004M, B01001_004MA	0	int	B01001
B01001_005E	Estimate!!Total!!Male!!10 to 14 years	SEX BY AGE	not required	B01001_005EA, B01001_005M, B01001_005MA	0	int	B01001
B01001_006E	Estimate!!Total!!Male!!15 to 17 years	SEX BY AGE	not required	B01001_006EA, B01001_006M, B01001_006MA	0	int	B01001
B01001_007E	Estimate!!Total!!Male!!18 and 19 years	SEX BY AGE	not required	B01001_007EA, B01001_007M, B01001_007MA	0	int	B01001

Figure 1. Screenshot of variable dictionary

2) Census data API: This table illustrates the request link structure of different aggregation levels including the input of GEOIDs of state, county, census tract and census block group, and variable names per users' interests. Figure 2 presents the example request URLs at different aggregation levels. Red frame highlights the URL examples of CBG level, which includes the input of GEOIDs of state, county, tract, and block group, as well as the variable names.



Geography Hierarchy	Geography Level	Example URL
us	010	https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=us:*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=us:1&key=YOUR_KEY_GOES_HERE
region	020	https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=region:*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=region:3&key=YOUR_KEY_GOES_HERE
division	030	https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=division:*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=division:5&key=YOUR_KEY_GOES_HERE
state	040	https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=state:*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=state:06&key=YOUR_KEY_GOES_HERE
state› county	050	https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=county:*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=county:*&in=state:*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=county:037&in=state:06&key=YOUR_KEY_GOES_HERE
state› county› county subdivision	060	https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=county%20subdivision:*&in=state:48&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=county%20subdivision:*&in=state:48&in=county:*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=county%20subdivision:91835&in=state:48%20county:201&key=YOUR_KEY_GOES_HERE
state› county› county subdivision› submincr civil division	067	https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=submincr%20civil%20division:*&in=state:72%20county:127%20county%20subdivision:57247&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=submincr%20civil%20division:76644&in=state:72%20county:127%20county%20subdivision:57247&key=YOUR_KEY_GOES_HERE
state› county› county subdivision› place remainder (or part)	070	https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=place remainder%20(or%20part):*&in=state:17%20county:031%20county%20subdivision:14000&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=place remainder%20(or%20part):14000&in=state:17%20county:031%20county%20subdivision:14000&key=YOUR_KEY_GOES_HERE
state› county› tract	140	https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=tract:*&in=state:06&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=tract:*&in=state:06&in=county:*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=tract:018700&in=state:06%20county:073&key=YOUR_KEY_GOES_HERE
state› county› tract› block group	150	https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=block%20group:*&in=state:06%20county:073&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=block%20group:*&in=state:06&in=county:*&in=tract:*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=block%20group:*&in=state:06&in=county:073&in=tract:*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=block%20group:1&in=state:06%20county:073%20tract:018700&key=YOUR_KEY_GOES_HERE
state› place› county (or part)	155	https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=county%20(or%20part):*&in=state:06%20place:44000&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=county%20(or%20part):037&in=state:06%20place:44000&key=YOUR_KEY_GOES_HERE
state› place	160	https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=place:*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=place:*&in=state:*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=place:51000&in=state:36&key=YOUR_KEY_GOES_HERE

Figure 2. Example request URLs provided by US census database

More details of URL structure components of CBG level were illustrated in Figure 3.

Figure 3. Explanations of CBG URL structure

Here, users need to apply and get their own API keys from the US census website.

2.2 Use case of streaming demographic data

```
In [1]: import pandas as pd
        from urllib import request
        import json
```

```
In [2]: ### Read cbg geoids for City of Boston
        CBGs = pd.read_csv('Updated_Boston_CBGs.csv')
        CBGs_head = CBGs.head(10)
        CBGs_sel = CBGs_head[['bos_cbgid','STATE','COUNTY','TRACT','BLKGRP']]
        CBGs_sel
```

Out[2]:

	bos_cbgid	STATE	COUNTY	TRACT	BLKGRP
0	250250001011	25	25	101	1
1	250250001012	25	25	101	2
2	250250001021	25	25	102	1
3	250250001022	25	25	102	2
4	250250002011	25	25	201	1
5	250250002012	25	25	201	2
6	250250002013	25	25	201	3
7	250250002014	25	25	201	4
8	250250002021	25	25	202	1
9	250250002022	25	25	202	2

Figure 4. Loading Python libraries and CBGs of Boston

With CBGs of Boston, this tutorial takes ten of them for demonstration. The US Census Bureau provides the CBG GEOIDs in terms of various data formats like shapefiles. Figure 5 defined the URL structure for these ten CBGs of Boston with the variable 'B01001_002E' (total male). 'census_api_key' refers to your personal API key which you can request from US Census Database.

```
In [3]: ### Build the request URL structure at CBG level

        ## Base URL
        url_base = 'https://api.census.gov/data/2020/acs/acs5?get=NAME,'

        ## Varibae list
        ### Here we take 'B01001_002E' total male count as an example
        demovar_list = ['B01001_002E']

        ## Use your API key
        census_api_key = '[redacted]'

        ### Set IDs for state, county, census tract and block group
        url_cbg_id = '&for=block%20group:'
        url_state_id = '&in=state:'
        url_county_id = '%20county:'
        url_censustract_id = '%20tract:'

        ### Set your API key
        url_key = '&key='
```

Figure 5. Request URL structure at CBG level

```
In [4]: for demo_var in demovar_list:

            CBGs_sel[demo_var] = ""
            print('Streaming variable: ' +demo_var)

            for i in range(len(CBGs_sel)):

                if len(str(CBGs_sel['STATE'][i])) ==2:
                    state_fip = str(CBGs_sel['STATE'][i])
                elif len(str(CBGs_sel['STATE'][i])) <2:
                    state_fip = str(0) + str(CBGs_sel['STATE'][i])

                if len(str(CBGs_sel['COUNTY'][i])) == 3:
                    county_fip = str(CBGs_sel['COUNTY'][i])
                elif len(str(CBGs_sel['COUNTY'][i])) == 2:
                    county_fip = str(0) + str(CBGs_sel['COUNTY'][i])
                elif len(str(CBGs_sel['COUNTY'][i])) == 1:
                    county_fip = str(0) + str(0) + str(CBGs_sel['COUNTY'][i])

                if len(str(CBGs_sel['TRACT'][i])) == 6:
                    census_tract_fip = str(CBGs_sel['TRACT'][i])
                elif len(str(CBGs_sel['TRACT'][i])) == 5:
                    census_tract_fip = str(0) + str(CBGs_sel['TRACT'][i])
                elif len(str(CBGs_sel['TRACT'][i])) == 4:
                    census_tract_fip = str(0) + str(0) + str(CBGs_sel['TRACT'][i])
                elif len(str(CBGs_sel['TRACT'][i])) == 3:
                    census_tract_fip = str(0) + str(0) + str(0) + str(CBGs_sel['TRACT'][i])

                block_group_fip = str(CBGs_sel['BLKGRP'][i])

                url_str_ele = url_base + demo_var + url_cbg_id+block_group_fip +url_state_id+state_fip +url_county_id+county_fip
                +url_censustract_id+census_tract_fip +url_key +census_api_key

                response_ele = request.urlopen(url_str_ele)
                html_str_ele = response_ele.read().decode("utf-8")
                if (html_str_ele):
                    json_data_ele = json.loads(html_str_ele)

                    CBGs_sel[demo_var][i] = json_data_ele[1][1]

            print('Streaming variable: ' +demo_var +'is completed')
```

Figure 6. Streaming variable information with user defined URL structure

```
In [5]: CBGs_sel
```

Out[5]:

	bos_cbgid	STATE	COUNTY	TRACT	BLKGRP	B01001_002E
0	250250001011	25	25	101	1	414
1	250250001012	25	25	101	2	513
2	250250001021	25	25	102	1	644
3	250250001022	25	25	102	2	900
4	250250002011	25	25	201	1	444
5	250250002012	25	25	201	2	425
6	250250002013	25	25	201	3	805
7	250250002014	25	25	201	4	416
8	250250002021	25	25	202	1	364
9	250250002022	25	25	202	2	431

Figure 7. Results check of variable ‘B01001_002E’

Step 3. Assign an address with a GEOID

3.1 Geocode addresses into coordinates

To assign an address with a CBG GEOID, we will start with geocoding such address. There are many APIs for geocoding addresses such as ESRI, Google, Microsoft, etc., where you can start with free trials with certain count of free geocoding addresses

requests and later make subscriptions for further use. Here, I take GeoPy, a free Python package, to do the geocoding of addresses, though its efficiency may not be good compared with those paid API services. Figure 8 shows how GeoPy was used to geocode five randomly selected addresses of hospitals and groceries in Massachusetts.

```
[1]: from geopy.extra.rate_limiter import RateLimiter
     from geopy.geocoders import Nominatim
     import pandas as pd
     Last executed at 2023-01-05 16:31:48 in 280ms

[2]: ### Create an empty dataframe
     df_address_MA = pd.DataFrame()
     Last executed at 2023-01-05 16:31:49 in 2ms

[3]: ### Random select five MA addresses
     address = ['300 Longwood Ave, Boston, MA 02115','26 Oxford St, Cambridge, MA 02138','200 Boylston St, Chestnut Hill, MA 02467',
                '55 Russell St, Waltham, MA 02451','40 Highland St, Worcester, MA 01609']
     Last executed at 2023-01-05 16:32:26 in 2ms

[4]: df_address_MA['ADDRESS']=address
     Last executed at 2023-01-05 16:32:36 in 5ms

[5]: # 1 - conveneint function to delay between geocoding calls
     locator = Nominatim(user_agent="application")
     geocode = RateLimiter(locator.geocode, min_delay_seconds=1)
     # 2- - create location column
     df_address_MA['location'] = df_address_MA['ADDRESS'].apply(geocode)
     # 3 - create longitude, laatitude and altitude from location column (returns tuple)
     df_address_MA['point'] = df_address_MA['location'].apply(lambda loc: tuple(loc.point) if loc else None)
     # 4 - split point column into latitude, longitude and altitude columns
     df_address_MA[['latitude', 'longitude', 'altitude']] = pd.DataFrame(df_address_MA['point'].tolist())
     Last executed at 2023-01-05 16:32:46 in 4.35s

[6]: df_address_MA
     Last executed at 2023-01-05 16:32:47 in 21ms
```

[6]:

	ADDRESS	location	point	latitude	longitude	altitude
0	300 Longwood Ave, Boston, MA 02115	(Boloco, 300, Longwood Avenue, Fenway / Kenmor...	(42.3376838, -71.1045659, 0.0)	42.337684	-71.104566	0.0
1	26 Oxford St, Cambridge, MA 02138	(26, Oxford Street, Agassiz, Cambridge, Middle...	(42.37836705092443, -71.11635923525951, 0.0)	42.378367	-71.116359	0.0
2	200 Boylston St, Chestnut Hill, MA 02467	(200, Boylston Street, Thompsonville, Newton, ...	(42.32000235849056, -71.17605732075471, 0.0)	42.320002	-71.176057	0.0
3	55 Russell St, Waltham, MA 02451	(55, Russell Street, Banks Square, Riverview, ...	(42.37539322079867, -71.2471568554421, 0.0)	42.375393	-71.247157	0.0
4	40 Highland St, Worcester, MA 01609	(40, Highland Street, Clinton, Worcester Count...	(42.426055257525384, -71.6986434987836, 0.0)	42.426055	-71.698643	0.0

Figure 8. GeoPy for geocoding addresses

3.2 Creating a shapefile with geocoded coordinates

With the geocoded addresses, I will show you how we can create a point shapefile for further spatial join with the CBG shapefile. Figure 9 shows the reading of the geocoded addresses and their coordinates.

```
In [1]: import pandas as pd
        import fiona

In [2]: address_df = pd.read_csv('sample_addrress_geocoded.csv')
        address_df
```

Out[2]:

	Unnamed: 0	address	lat	long
0	0	300 Longwood Ave, Boston, MA 02115	42.337684	-71.104566
1	1	26 Oxford St, Cambridge, MA 02138	42.378367	-71.116359
2	2	200 Boylston St, Chestnut Hill, MA 02467	42.320002	-71.176057
3	3	55 Russell St, Waltham, MA 02451	42.375393	-71.247157
4	4	40 Highland St, Worcester, MA 01609	42.426055	-71.698643

Figure 9. Reading geocoded sample addresses

Figure 10 starts with presenting how I defined the schema to create the point shapefile, then creating a fiona object where I specified the shapefile name as 'geocodedAddressPoints.shp', driver, and CRS, and finally writing out each geocoded addresses as a point within the point shapefile.

```
In [3]: # define schema
        schema = {
            'geometry':'Point',
            'properties':[('Name','str')]
        }

In [4]: #open a fiona object
        address_Shp = fiona.open('geocodedAddressPoints.shp', mode='w', driver='ESRI Shapefile',schema = schema, crs = "EPSG:4326")

In [5]: #iterate over each row in the dataframe and save record
        for index, row in address_df.iterrows():
            rowDict = {
                'geometry' : {'type':'Point',
                              'coordinates': (row.long,row.lat)},
                'properties': {'Name' : row.address},
            }
            address_Shp.write(rowDict)
        #close fiona object
        address_Shp.close()
```

Figure 10. Creating point shapefile with geocoded addresses

3.3 Spatial joining geocoded addresses with CBG shapefiles

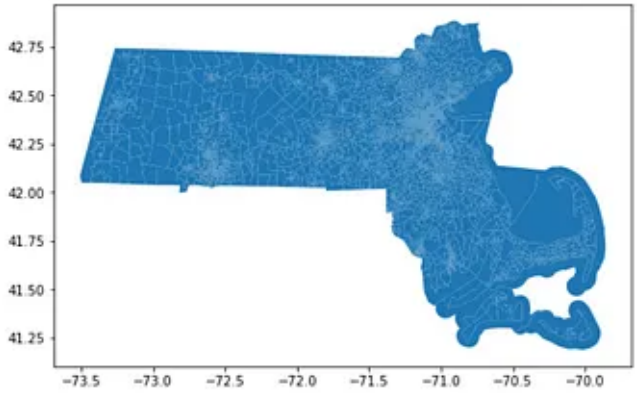
This section joins the point shapefile of geocoded addresses and the shapefile (polygon) of Masschusstes at CBG level, and finally assigns the CBG GEOID with each specific address.

```
In [1]: import geopandas as gpd
        import pandas as pd
        import fiona
        import csv
        import matplotlib.pyplot as plt

        import rtree
        import pygeos
        from pyproj import CRS

In [2]: ### Read Mass shaefile at CBG level
        geo_df = gpd.read_file("                                                    tl_2017_25_bg.shp")
        ###Read geocoded address shapefile
        address_gdf = gpd.read_file(r"geocodedAddressPoints.shp")
```

Figure 11. Reading shapefiles of geocoded addresses and Mass

Figure 12 shows the maps of Mass at CBG level and sample geocoded addresses.

```
In [3]: ### Draw Mass map by CBGs
        fig0, ax0 = plt.subplots(figsize = (8, 6))
        geo_df.plot(ax = ax0);
```




```
In [4]: ### Draw geocoded addresses
        fig1, ax1 = plt.subplots(figsize = (8, 6))
        address_gdf.plot(ax = ax1, markersize = 20, color = "red");
```

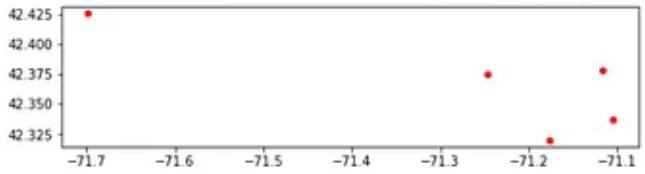


Figure 12. Drawing maps of Mass at CBG level and geocoded sample addresses

Figure 13 illustrates how to check the Coordinate Reference Systems (CRSs) of both shapefiles before joining them. More details of CRSs can be referred from here. When two shapefiles have different CRSs, here I converted the CRS of the geocoded address shapefile into the CRS of the Mass shapefile. With that, I draw them into one map. Red nodes denote the locations of my geocoded address samples.

```
In [5]: ### Check the CRS systems for both shapefiles
        crs1 = address_gdf.crs
        crs2 = geo_df.crs

        print(crs1)
        print(crs2)

        epsg:4326
        epsg:4269

In [6]: ### Make CRSs of both shapefiles consistent
        address_gdf_con = address_gdf.to_crs(crs2)

In [7]: ## Draw geocoded addresses with Mass map
        fig, ax = plt.subplots(figsize = (8, 6))
        address_gdf_con.plot(ax = ax, markersize = 25, color = "red")
        geo_df.plot(ax = ax)

Out[7]: <AxesSubplot:>
```

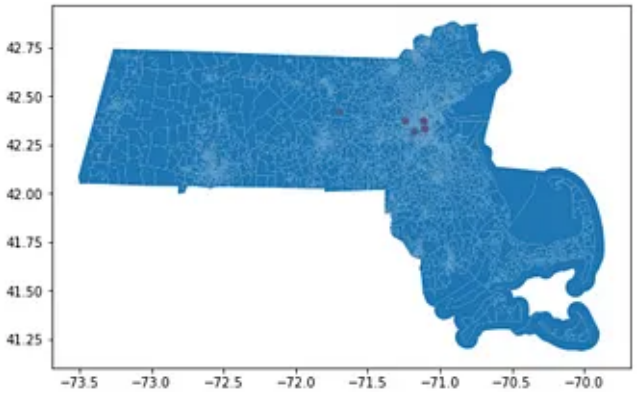


Figure 13. Checking CRSs of both shapefiles and making them consistent

From Figure 13, we can see their CRSs are consistent so that spatial join of them can be performed. Figure 14 shows how I did the spatial join. As a result, you can see each sample address has a CBG's GEOID.

```
In [8]: ### Spatial join geocoded address shapefile and Mass shapefile
        ### Assign each address with a CBG GEOID
        address_cbg = gpd.sjoin(address_gdf_con, geo_df, how="left", op='intersects')
        address_cbg
```

Out[8]:

	Name	geometry	index_right	STATEFP	COUNTYFP	TRACTCE	BLKGRPCE	GEOID	NAMELSAD	MTFCC	FUNCSTAT	ALAND	AWATER	INT
0	300 Longwood Ave, Boston, MA 02115	POINT (-71.10457 42.33768)	238	25	025	010300	2	250250103002	Block Group 2	G5030	S	360526	0	+42.3
1	26 Oxford St, Cambridge, MA 02138	POINT (-71.11636 42.37837)	1867	25	017	353600	5	250173536005	Block Group 5	G5030	S	327187	0	+42.3
2	200 Boylston St, Chestnut Hill, MA 02467	POINT (-71.17606 42.32000)	1982	25	017	373900	1	250173739001	Block Group 1	G5030	S	993154	0	+42.3
3	55 Russell St, Waltham, MA 02451	POINT (-71.24716 42.37539)	660	25	017	368300	5	250173683005	Block Group 5	G5030	S	208018	0	+42.3
4	40 Highland St, Worcester, MA 01609	POINT (-71.69864 42.42605)	3666	25	027	716200	2	250277162002	Block Group 2	G5030	S	1222219	0	+42.4

Figure 14. Spatial join of geocoded address sample with Mass shapefile

Final Step. Join address sample with demographic data

For better visualization in Figure 15, I selected the GEOIDs relevant information from the 'address_cbg' data frame in Figure 14. Following that, I read the 'B01001_002E' variable table saved from Step 2 (Figure 7). Lastly, I joined sample addresses table with the CBG-level demographic data through their CBGs' GEOIDs. 'NaN' denotes the situation where varibale 'B01001_002E' for that CBG is not avilable.

```
In [9]: address_cbg_sel = address_cbg[['Name','GEOID','STATEFP','COUNTYFP','TRACTCE','BLKGRPCE']]
address_cbg_sel
```

Out[9]:

	Name	GEOID	STATEFP	COUNTYFP	TRACTCE	BLKGRPCE
0	300 Longwood Ave, Boston, MA 02115	250250103002	25	025	010300	2
1	26 Oxford St, Cambridge, MA 02138	250173536005	25	017	353600	5
2	200 Boylston St, Chestnut Hill, MA 02467	250173739001	25	017	373900	1
3	55 Russell St, Waltham, MA 02451	250173683005	25	017	368300	5
4	40 Highland St, Worcester, MA 01609	250277162002	25	027	716200	2

```
In [10]: cdb_male_count = pd.read_csv('cbg_male_count.csv')
cdb_male_count.head(15)
```

Out[10]:

	Unnamed: 0	bos_cbgid	STATE	COUNTY	TRACT	BLKGRP	B01001_002E
0	0	250250001011	25	25	101	1	414.0
1	1	250250001012	25	25	101	2	513.0
2	2	250250001021	25	25	102	1	644.0
3	3	250250001022	25	25	102	2	900.0
4	4	250250002011	25	25	201	1	444.0
5	5	250250002012	25	25	201	2	425.0
6	6	250250002013	25	25	201	3	805.0
7	7	250250002014	25	25	201	4	416.0
8	8	250250002021	25	25	202	1	364.0
9	9	250250002022	25	25	202	2	431.0
10	10	250250103002	25	25	10300	2	1254.0
11	11	250173536005	25	17	353600	5	NaN
12	12	250173739001	25	17	373900	1	NaN
13	13	250173683005	25	17	368300	5	355.0
14	14	250277162002	25	27	716200	2	545.0

```
In [11]: address_cbg_sel['GEOID']=address_cbg_sel['GEOID'].astype(str)
cdb_male_count['bos_cbgid']=cdb_male_count['bos_cbgid'].astype(str)
```

```
In [12]: sample_address_cbg_count = pd.merge(address_cbg_sel,cdb_male_count,left_on='GEOID',right_on='bos_cbgid',how='left')
sample_address_cbg_count
```

Out[12]:

	Name	GEOID	STATEFP	COUNTYFP	TRACTCE	BLKGRPCE	Unnamed: 0	bos_cbgid	STATE	COUNTY	TRACT	BLKGRP	B01001_002E
0	300 Longwood Ave, Boston, MA 02115	250250103002	25	025	010300	2	10	250250103002	25	25	10300	2	1254.0
1	26 Oxford St, Cambridge, MA 02138	250173536005	25	017	353600	5	11	250173536005	25	17	353600	5	NaN
2	200 Boylston St, Chestnut Hill, MA 02467	250173739001	25	017	373900	1	12	250173739001	25	17	373900	1	NaN
3	55 Russell St, Waltham, MA 02451	250173683005	25	017	368300	5	13	250173683005	25	17	368300	5	355.0
4	40 Highland St, Worcester, MA 01609	250277162002	25	027	716200	2	14	250277162002	25	27	716200	2	545.0

Figure 15. Final join of sample address with CBG-level demographic data

Summary

This tutorial shows you a process flow of how to assign the aggregated demographic information with a specific address. There are other methods to achieve this. Here I provided you one of them. Any comments and suggestions on this and other methods are warmly welcome!

Geospatial Data Science | Data Science | Demographics

Follow

Written by Faxi Yuan

222 Followers

Data Scientist @ MAPFRE; All views are my own. LinkedIn: https://www.linkedin.com/in/faxi-yuan-bb2167104

More from Faxi Yuan