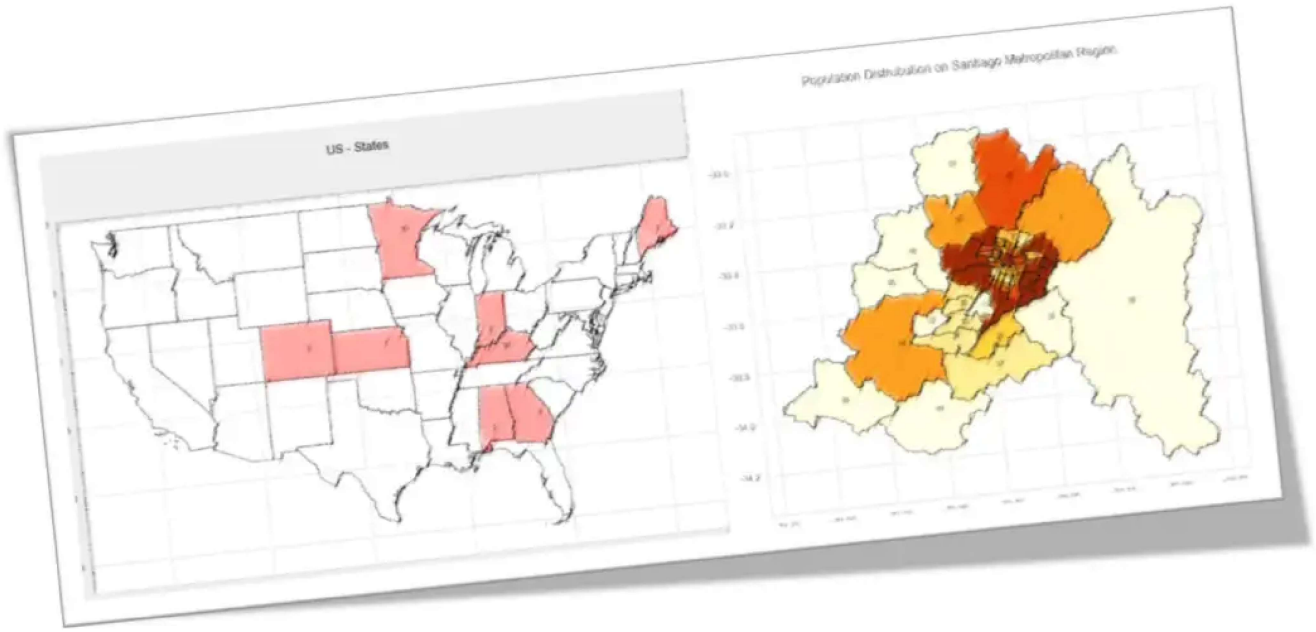
Published in Towards Data Science

Marcelo Rovai Follow

Nov 20, 2018 · 13 min read · Listen

Save

Mapping Geograph Data in Python



One great help when working in Data Science, is to visualize your data on a geo map and for that, several packages can take care of it, as GeoPandas for example.

> *You can learn how to use GeoPandas, reading my article: How Safe are the Streets of Santiago.*

Sometimes install Geopandas packages can be complicated, depending on what environment you are working. Or, simplesly you need take the control seat of your

code! So, on this article we will explore on "the hard way", how to construct our own "geo map functions", using "Shapefiles" and basic Python libraries.

1. Shapefiles

Developed and regulated by Esri as a (mostly) open specification, the shapefile format spatially describes geometries as either ‘points’, ‘polylines’, or ‘polygons’. In OpenStreetMap terms these can be considered as ‘nodes’, ‘ways’ and ‘closed ways’,

respectively. Each geometry has a set of associated attributes. Broadly speaking these are a bit like OSM's tags.

The shapefile is in fact a grouping of several files formatted to represent different aspects of geodata:

- .shp — shape format; the feature geometry itself.
- .shx — shape index format; a positional index of the feature geometry to allow seeking forwards and backwards quickly.
- .dbf — attribute format; columnar attributes for each shape, in dBase IV format.

There are also several optional files in the shapefile format. The most significant of these is the .prj file which describes the coordinate system and projection information used. Although not part of the Esri shapefile standard, the .lyr file is often included as it contains specifications of how to display the data (colour, labelling, etc) in ArcGIS software.

> For more info see wikipedia

2. Installing Python Shapefile Library (PyShp)

The Python Shapefile Library (pyshp) provides read and write support for the Esri Shapefile format. The Shapefile format is a popular Geographic Information System vector data format created by Esri.

To Install *pyshp,* execute below instruction in your Terminal:

```
pip install pyshp
```

3. Importing and initializing main Python libraries

```
import numpy as np
import pandas as pd
import shapefile as shp
import matplotlib.pyplot as plt
import seaborn as sns
```

Initializing vizualization set

```
sns.set(style=”whitegrid”, palette=”pastel”, color_codes=True)
sns.mpl.rc(“figure”, figsize=(10,6))
```

and if you are using a Jupyter Notebook:

```
%matplotlib inline
```

4. Opening a Vector Map

As described on 1., a vector map is a group of several files, with name.shp being the main one, where the geographic features are saved. Important that all other files as 'name.shx', ' name.dbf', etc., must be at same folder.

On this tutorial, we will work with maps related to the cities ("Comunas") that together, make the Santiago Metropolitan Region. On INE (Chilean National Institute of Statistics), is possible to download a group of shapefiles related with maps, created for the last national 2017 census :

- Comuna.cpg
- Comuna.shp
- Comuna.dbf
- Comuna.shp.xml
- Comuna.prj
- Comuna.shx
- Comuna.sbn
- Comuna.sbx

```
shp_path = “./Comunas_RM_Mapas_Vectoriales/Comuna.shp”
sf = shp.Reader(shp_path)
```

Let's check how many different "shapes" were imported by our function *shp.Reader:*

```
len(sf.shapes())
```

The result will be: 52

This means that exist 52 shapes on our shape files, what make sense, once the Santiago Metropolitan Region has 52 "comunas" as shown at below map (do not worry, before the end of this article, you will learn how to create a map like this one, directly from your data):

Let's also explore one of the shapes (or "records"):

```
sf.records()[1]
```

The result will be an array with 6 elements:

```
Out:

['13',
 '131',
 '13115',
 'REGIÓN METROPOLITANA DE SANTIAGO',
```

```
'SANTIAGO',
'LO BARNECHEA']
```

The element [5] is the name of the 'comuna', in this case: 'LO BARNECHEA', a 'comuna' located on oriental part of the city, where the Andes montains are located (and also my home! ;-)

You can get its name directlly:

```
sf.records()[1][5]
```

The most central 'comuna' of Santiago Metropolitan Region is exactly the Comuna of Santiago (little confuse?), where you can find the Metropolitan Catedral, La Moneda Presidential Palace (That was heavy bombed on 73), Pablo Neruda's house, etc.

But, Let' s end sightseeing and take a look on Santiago's comuna data structure (id: 25):

```
sf.records()[25]
```

Out[]:

```
['13',
 '131',
 '13101',
 'REGIÓN METROPOLITANA DE SANTIAGO',
 'SANTIAGO',
 'SANTIAGO']
```

We can see that some data changed and most importantly, the name of the 'comuna', that is now, 'SANTIAGO'.

> *Note that you can apply what will be describe on this tutorial to any shapfile.*

5. Converting shapefile data on Pandas dataframe

In the last example, I previously knew that Santiago" id was '25'. But how to find such id, starting from a comuna's name? Let's first create a usefull function to convert our 'shapefile' format on a more commun Pandas dataframe format:

```
def read_shapefile(sf):
    """
    Read a shapefile into a Pandas dataframe with a 'coords'
    column holding the geometry information. This uses the pyshp
    package
    """
    fields = [x[0] for x in sf.fields][1:]
    records = sf.records()
    shps = [s.points for s in sf.shapes()]

    df = pd.DataFrame(columns=fields, data=records)
    df = df.assign(coords=shps)

    return df
```

So, let's convert sf data on a dataframe and see how it looks like:

```
df = read_shapefile(sf)
df.shape
```

The dataframe has a shape of (52, 7). What means that we we have 7 diferent features (columns) for each line ('comuna'). Remember that previosly we saw 6 of those features. Seems that an extra one was added now. Let's see a sample:

```
df.sample(5)
```

	REGION	PROVINCIA	COMUNA	NOM_REGION	NOM_PROVIN	NOM_COMUNA	coords
32	13	136	13602	REGIÓN METROPOLITANA DE SANTIAGO	TALAGANTE	EL MONTE	[(-70.98764514099997, -33.616396522999935), (-...
12	13	131	13131	REGIÓN METROPOLITANA DE SANTIAGO	SANTIAGO	SAN RAMÓN	[(-70.63741773399994, -33.51877349899996), (-7...
38	13	132	13203	REGIÓN METROPOLITANA DE SANTIAGO	CORDILLERA	SAN JOSÉ DE MAIPO	[(-70.09163445799999, -33.052354270999956), (-...
14	13	131	13111	REGIÓN METROPOLITANA DE SANTIAGO	SANTIAGO	LA GRANJA	[(-70.60985596899997, -33.537431253999955), (-...
8	13	131	13119	REGIÓN METROPOLITANA DE SANTIAGO	SANTIAGO	MAIPÚ	[(-70.77557241299996, -33.45816236699994), (-7...

The last column is exactelly the coordinates, latitude and longitude, of every point that was used to create a specific map shape.

Confuse? Let's dig a little bit more.

How we can locate the Santiago comuna's id? Now with Pandas is very simple:

```
df[df.NOM_COMUNA == 'SANTIAGO']
```

	REGION	PROVINCIA	COMUNA	NOM_REGION	NOM_PROVIN	NOM_COMUNA	coords
25	13	131	13101	REGIÓN METROPOLITANA DE SANTIAGO	SANTIAGO	SANTIAGO	[(-70.66527655199997, -33.42827810699998), (-7...

We can easily see that 25 is exactelly the dataframe index, where our comuna shape is located.

With simple Pandas' s commands you can related the index (or id) with the comuna's name:

```
df.NOM_COMUNA
Out:
```

```
0 LAS CONDES
1 LO BARNECHEA
2 VITACURA
3 HUECHURABA
4 PUDAHUEL

…

49 ALHUÉ
50 LAMPA
51 TILTIL
```

6. Plotting a specific shape

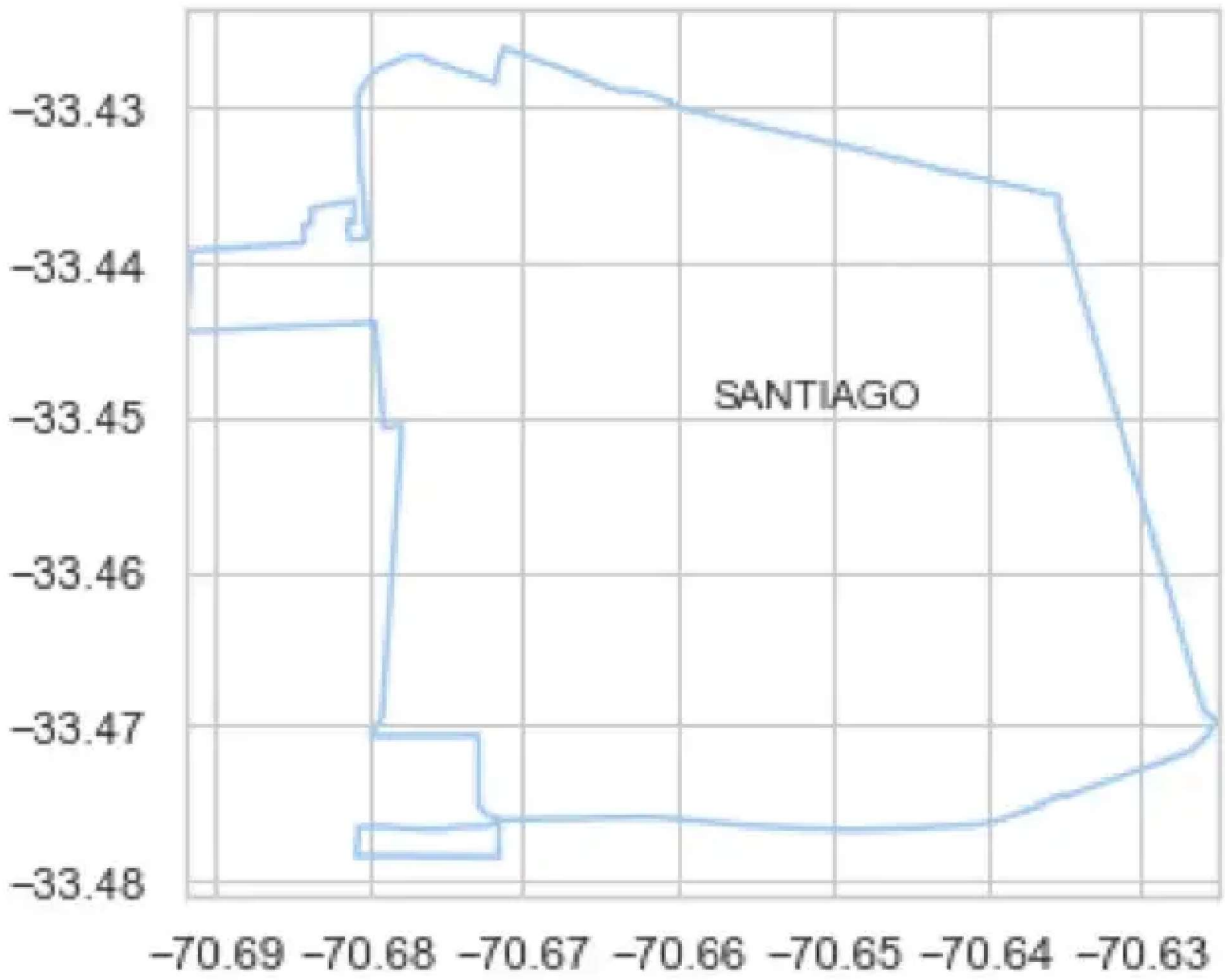
Finally, we will see what a shape really is. For that, we should create a function to plot it. We will use the Python MatPlotLib library:

```
def plot_shape(id, s=None):
    """ PLOTS A SINGLE SHAPE """
    plt.figure()
    ax = plt.axes()
    ax.set_aspect('equal')
    shape_ex = sf.shape(id)
    x_lon = np.zeros((len(shape_ex.points),1))
    y_lat = np.zeros((len(shape_ex.points),1))
    for ip in range(len(shape_ex.points)):
        x_lon[ip] = shape_ex.points[ip][0]
        y_lat[ip] = shape_ex.points[ip][1]

    plt.plot(x_lon,y_lat)
    x0 = np.mean(x_lon)
    y0 = np.mean(y_lat)
    plt.text(x0, y0, s, fontsize=10)
    # use bbox (bounding box) to set plot limits
    plt.xlim(shape_ex.bbox[0],shape_ex.bbox[2])
    return x0, y0
```

The above function does two things: a) Plot the shape (polygon) based on the comuna's coordinates and, b) calculate and return the medium point of that specific shape (x0, y0). This medium point was also used to define where to print the comuna's name.

For example, for our famous Santiago's comuna:

```
comuna = 'SANTIAGO'
com_id = df[df.NOM_COMUNA == comuna].index.get_values()[0]
plot_shape(com_id, comuna)
```



Santiago — Shape only

Note that we must know the shape id (index) to plot it, but we entered with the Comuna's name: SANTIAGO. Using Pandas was ease to calculate the id as you can see on the second line of the previous code.

7. Plotting a complete map

Plotting a single shape was basically to work around this small part of the code:

```
sf.shape(id)
```

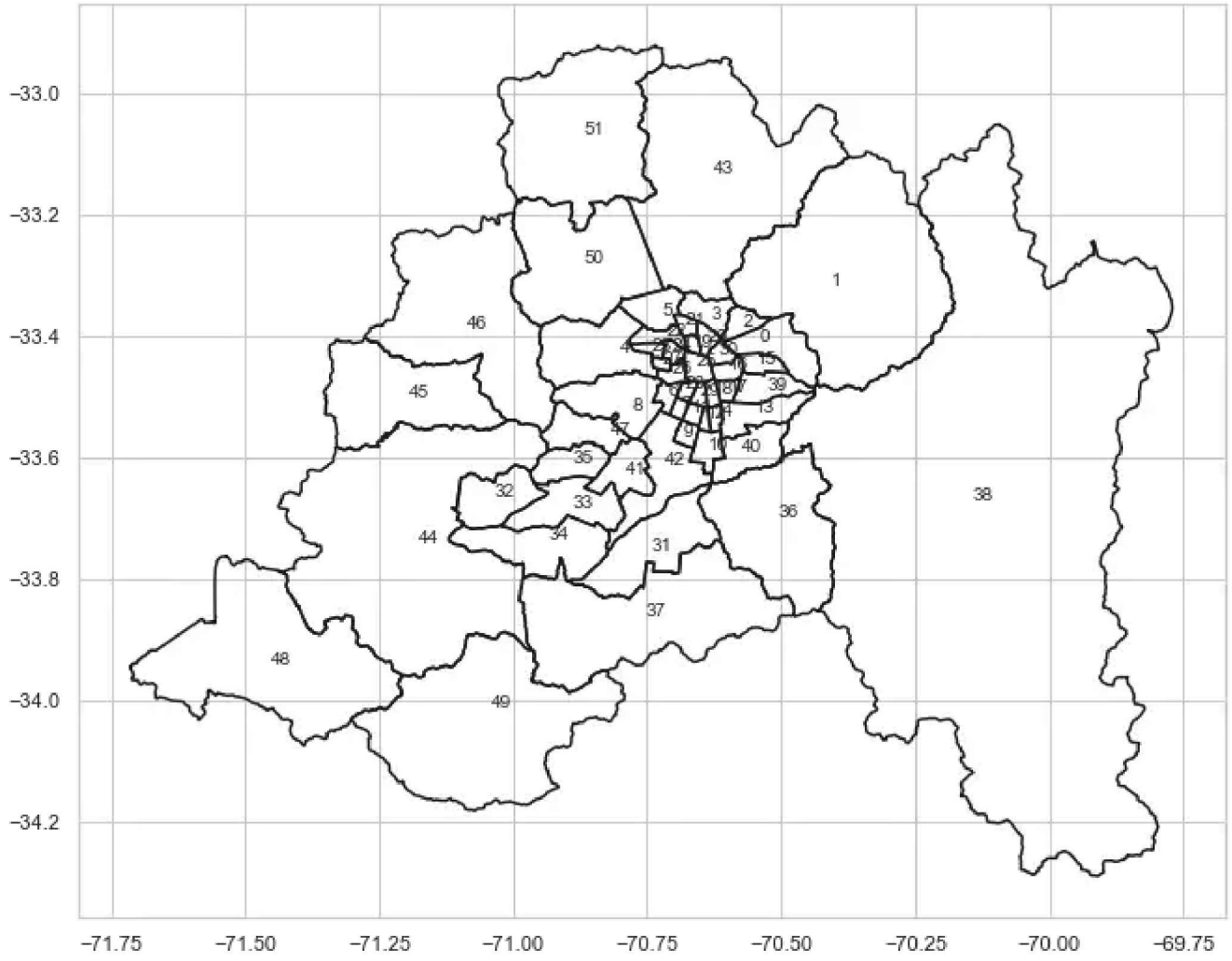
Now, we must plot at same picture, all the shapes that are on our dataframe. For that, we will use the following function:

```
def plot_map(sf, x_lim = None, y_lim = None, figsize = (11,9)):
    '''
    Plot map with lim coordinates
    '''
    plt.figure(figsize = figsize)
    id=0
    for shape in sf.shapeRecords():
        x = [i[0] for i in shape.shape.points[:]]
        y = [i[1] for i in shape.shape.points[:]]
        plt.plot(x, y, 'k')

        if (x_lim == None) & (y_lim == None):
            x0 = np.mean(x)
            y0 = np.mean(y)
            plt.text(x0, y0, id, fontsize=10)
        id = id+1

    if (x_lim != None) & (y_lim != None):
        plt.xlim(x_lim)
        plt.ylim(y_lim)
```

The above function, by default, plot all shapes on a given 'df' file, including its shape id at middle of it. Or a zoomed map will be plotted (w/o ids). You can change the function to print or not the ids.

Plotting a full map:

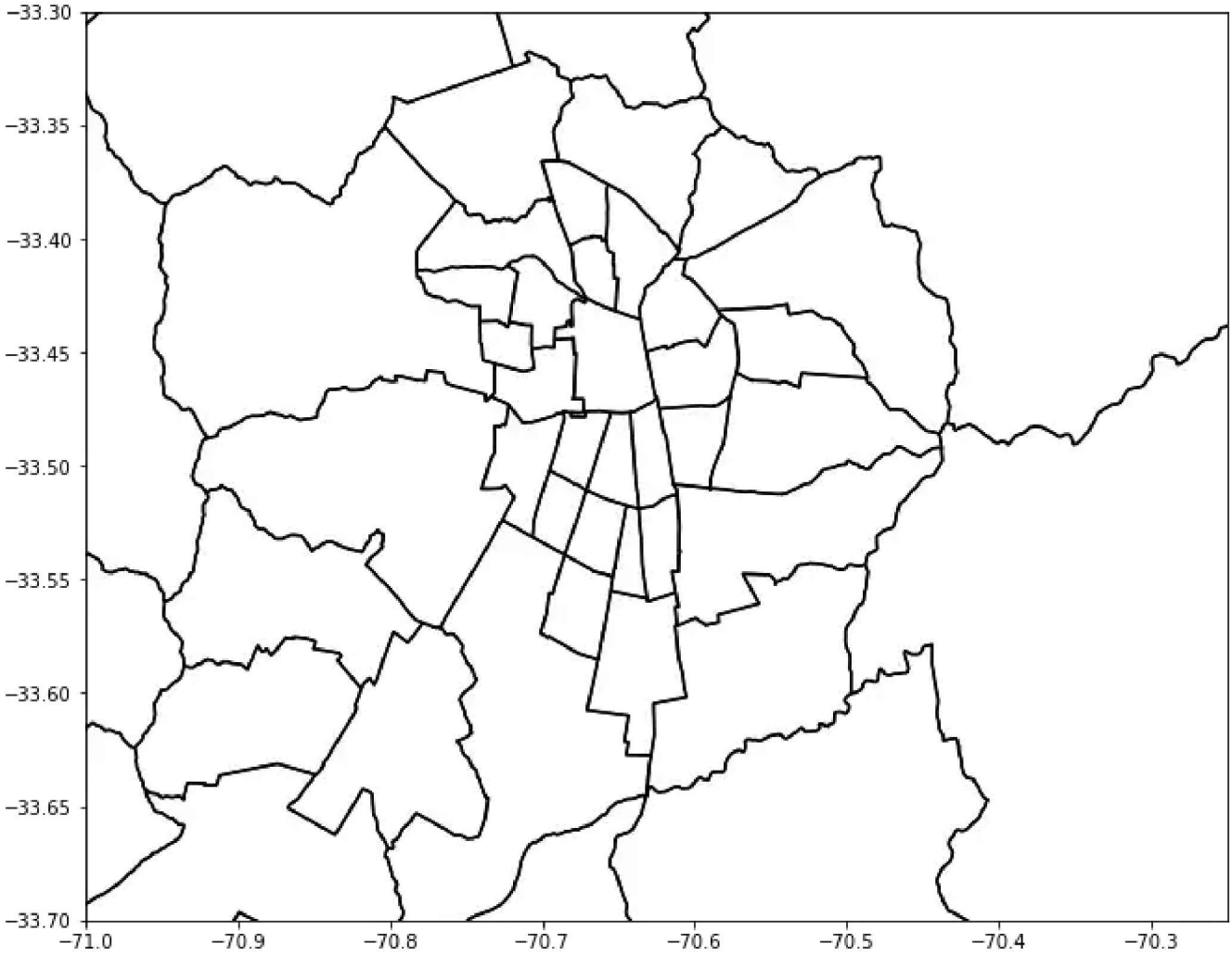
```
plot_map(sf)
```



Full Map

Plotting a zommed map:

```
y_lim = (-33.7,-33.3) # latitude
x_lim = (-71, -70.25) # longitude

plot_map(sf, x_lim, y_lim)
```



Full Map with Zoom

8. Plotting a single shape over a complete map

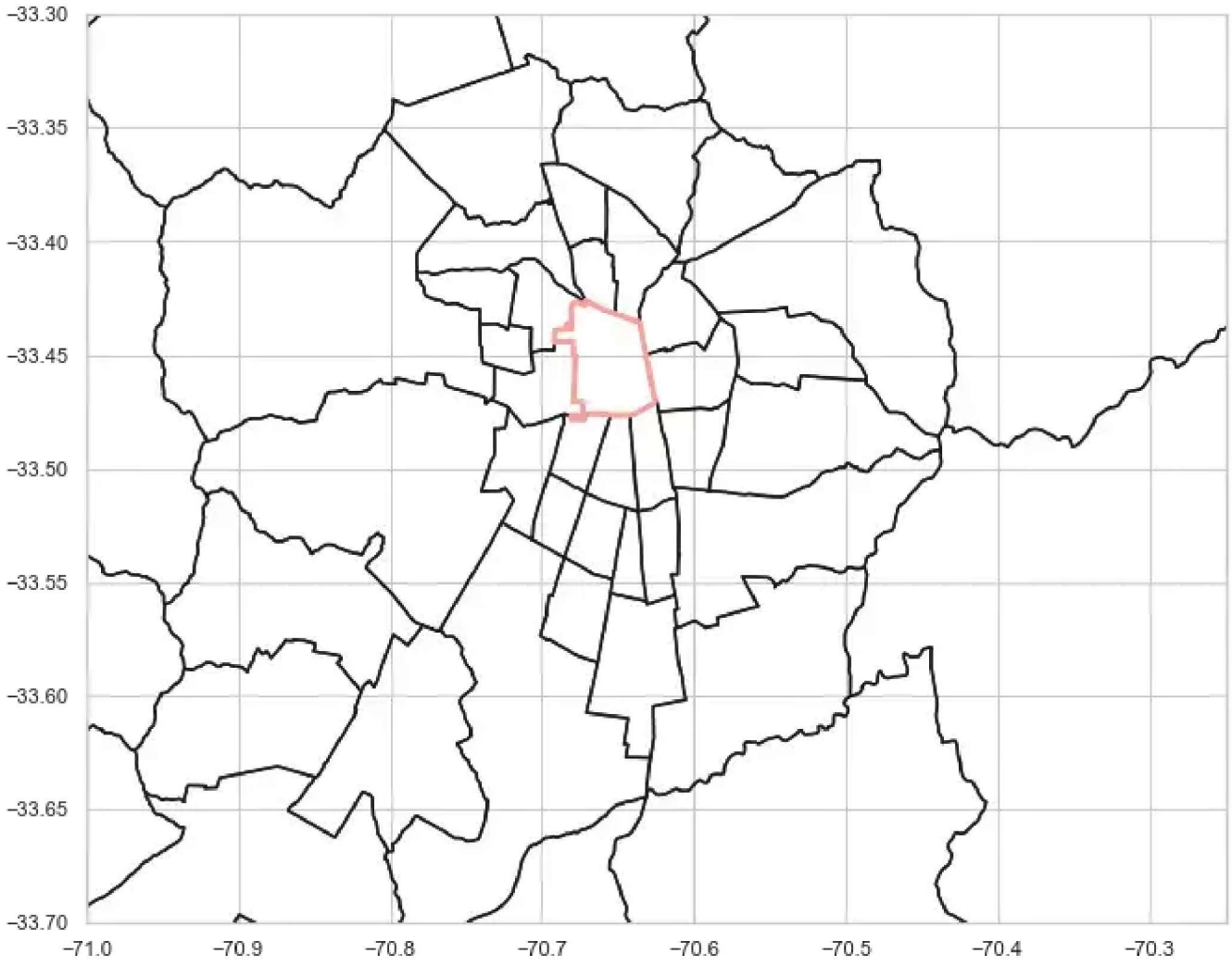
We can "merge" the two previous functions and "plot" a single shape inside a full map. For that, let's write a new function, where the shape id is now an input parameter:

```
def plot_map2(id, sf, x_lim = None, y_lim = None, figsize=(11,9)):
    '''
    Plot map with lim coordinates
    '''

    plt.figure(figsize = figsize)
    for shape in sf.shapeRecords():
        x = [i[0] for i in shape.shape.points[:]]
        y = [i[1] for i in shape.shape.points[:]]
        plt.plot(x, y, 'k')

    shape_ex = sf.shape(id)
    x_lon = np.zeros((len(shape_ex.points),1))
```

```
    y_lat = np.zeros((len(shape_ex.points),1))
    for ip in range(len(shape_ex.points)):
        x_lon[ip] = shape_ex.points[ip][0]
        y_lat[ip] = shape_ex.points[ip][1]
    plt.plot(x_lon,y_lat, 'r', linewidth=3)

    if (x_lim != None) & (y_lim != None):
        plt.xlim(x_lim)
        plt.ylim(y_lim)
```

Plotting the Santiago's comuna in "red":

```
plot_map2(25, sf, x_lim, y_lim)
```



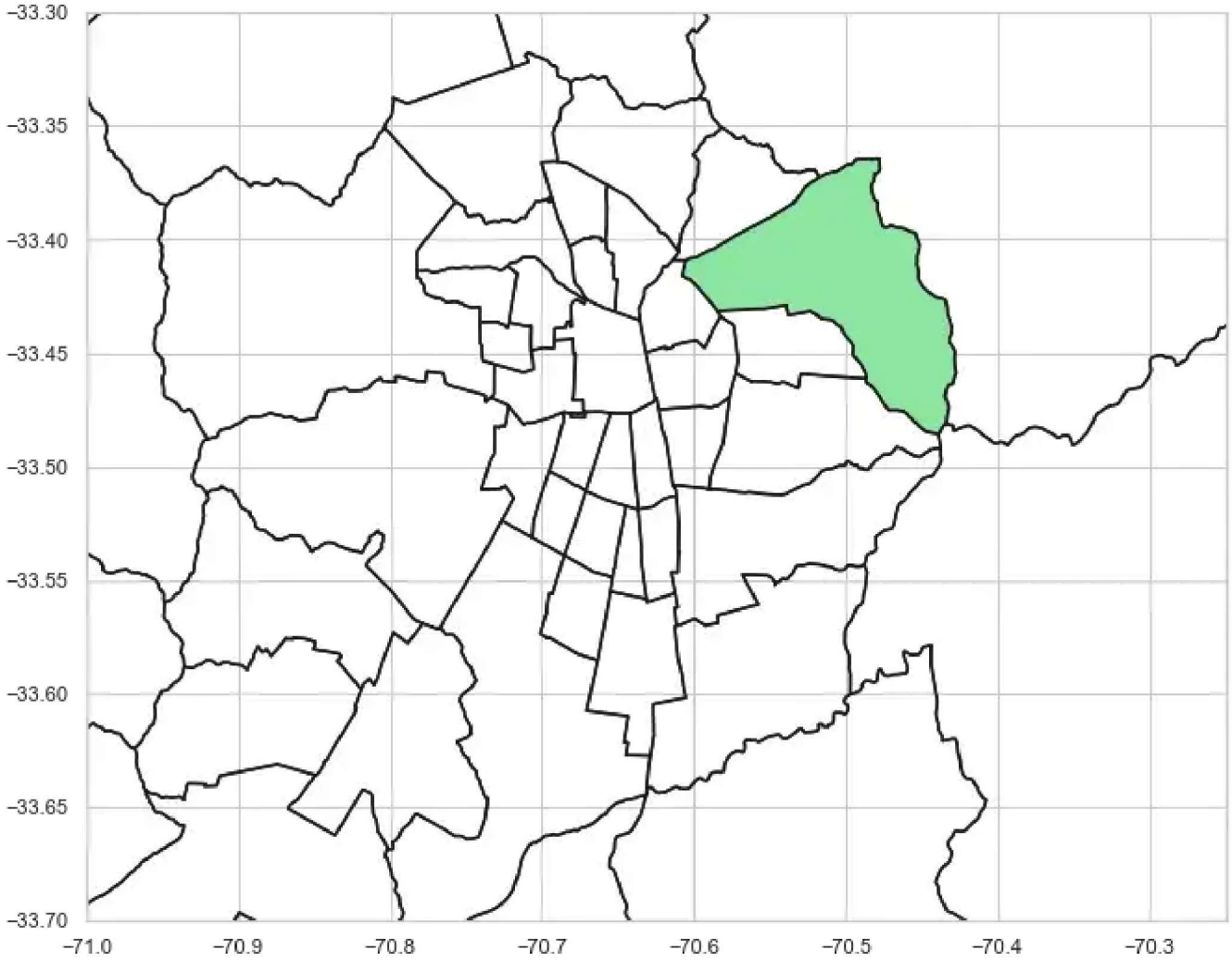
Santiago

And if we want to "fill" a single shape with a specific color? Simple! We can use plt.fill for that. The function can be rewriten:

```
def plot_map_fill(id, sf, x_lim = None,
                          y_lim = None,
                          figsize = (11,9),
                          color = 'r'):
    '''
    Plot map with lim coordinates
    '''

    plt.figure(figsize = figsize)
    fig, ax = plt.subplots(figsize = figsize)

    for shape in sf.shapeRecords():
        x = [i[0] for i in shape.shape.points[:]]
        y = [i[1] for i in shape.shape.points[:]]
        ax.plot(x, y, 'k')

    shape_ex = sf.shape(id)
    x_lon = np.zeros((len(shape_ex.points),1))
    y_lat = np.zeros((len(shape_ex.points),1))
    for ip in range(len(shape_ex.points)):
        x_lon[ip] = shape_ex.points[ip][0]
        y_lat[ip] = shape_ex.points[ip][1]
    ax.fill(x_lon,y_lat, color)

    if (x_lim != None) & (y_lim != None):
        plt.xlim(x_lim)
        plt.ylim(y_lim)
```

Plotting the comuna of "Las Condes" (id=0) in green ('g'):

```
plot_map_fill(0, sf, x_lim, y_lim, color='g')
```



Las Condes

9. Plotting multiple shapes on a full map

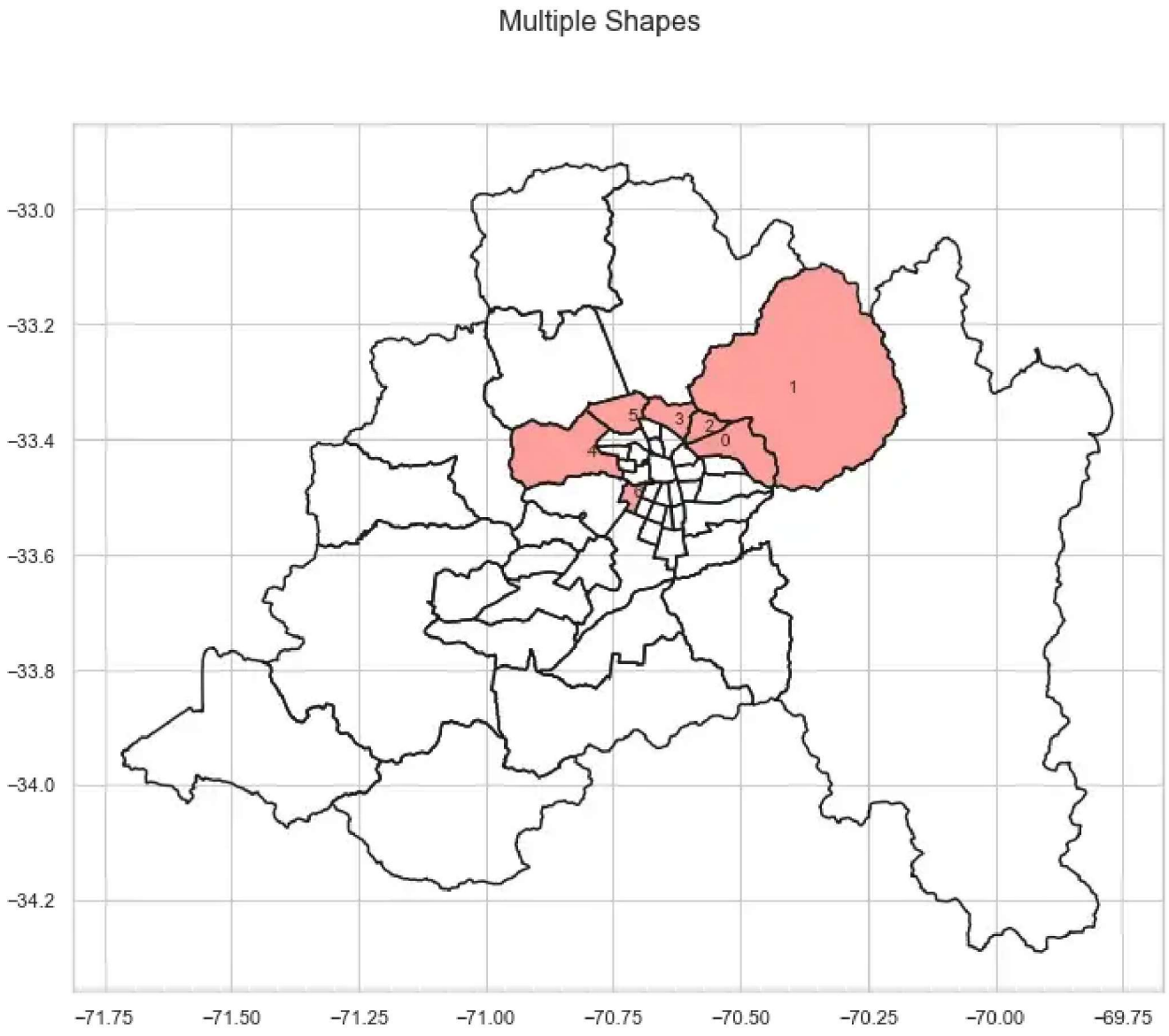
The next natural step on our "hard way mapping journey", will be create a map where several shapes are selected. For that, insteady of having an id as input parameter, we will have a list of ids, and will use a for loop to fill with color each one of them. The modified function is shown below:

```
def plot_map_fill_multiples_ids(title, comuna, sf,
                                                x_lim = None,
                                                y_lim = None,
                                                figsize = (11,9),
                                                color = 'r'):
    '''
    Plot map with lim coordinates
    '''

    plt.figure(figsize = figsize)
```

```
    fig, ax = plt.subplots(figsize = figsize)
    fig.suptitle(title, fontsize=16)

    for shape in sf.shapeRecords():
        x = [i[0] for i in shape.shape.points[:]]
        y = [i[1] for i in shape.shape.points[:]]
        ax.plot(x, y, 'k')

    for id in comuna:
        shape_ex = sf.shape(id)
        x_lon = np.zeros((len(shape_ex.points),1))
        y_lat = np.zeros((len(shape_ex.points),1))
        for ip in range(len(shape_ex.points)):
            x_lon[ip] = shape_ex.points[ip][0]
            y_lat[ip] = shape_ex.points[ip][1]
        ax.fill(x_lon,y_lat, color)

        x0 = np.mean(x_lon)
        y0 = np.mean(y_lat)
        plt.text(x0, y0, id, fontsize=10)

    if (x_lim != None) & (y_lim != None):
        plt.xlim(x_lim)
        plt.ylim(y_lim)
```

On the above function, "comuna" is now a list of ids:

```
comuna_id = [0, 1, 2, 3, 4, 5, 6]
plot_map_fill_multiples_ids("Multiple Shapes",
                            comuna_id, sf, color = 'r')
```



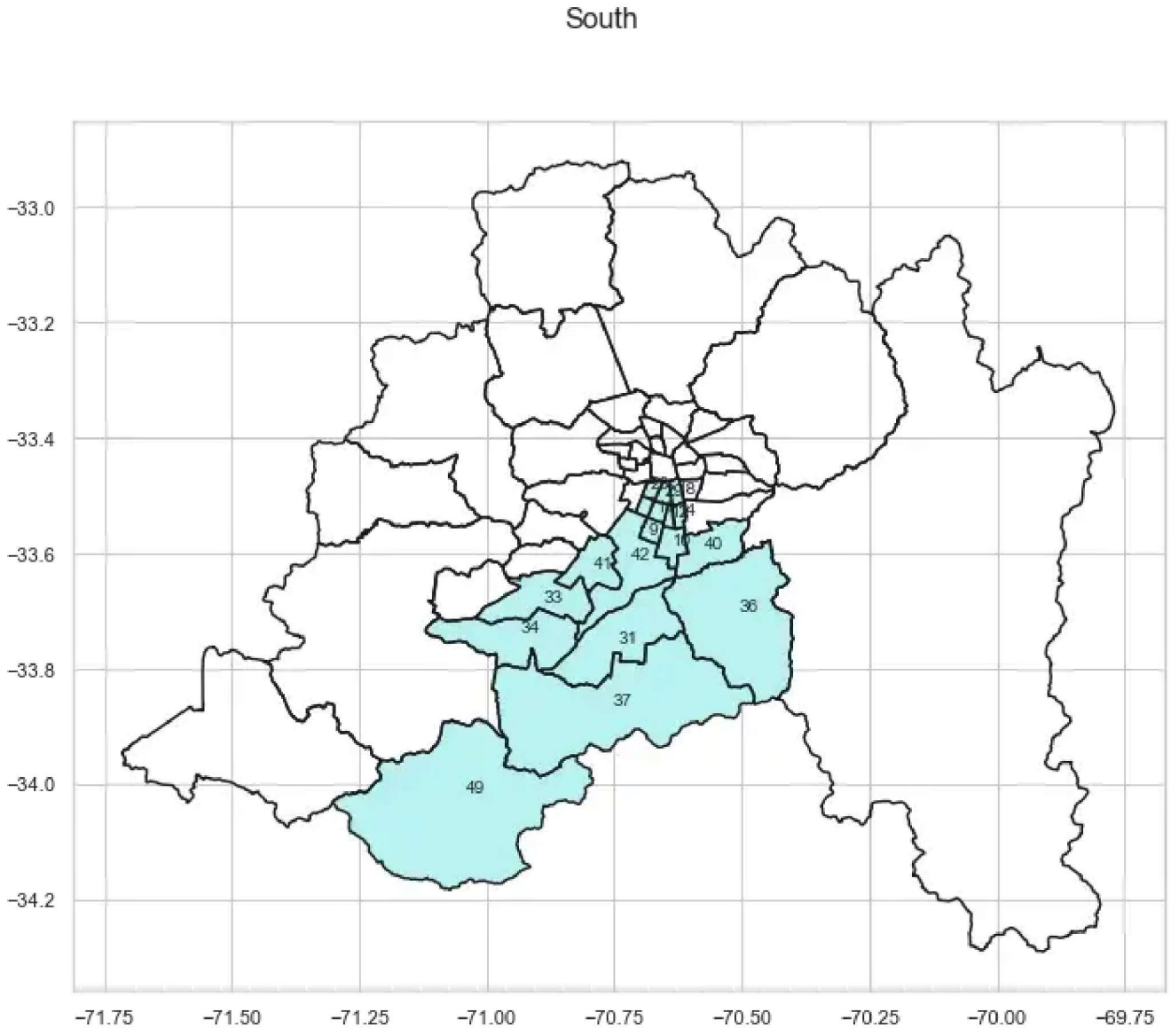
Taking opportunity of our previous pandas dataframe, let's create a simple function where the input is the name of comuna, insteady of its id:

```
def plot_comunas_2(sf, title, comunas, color):
    '''
    Plot map with selected comunes, using specific color
    '''

    df = read_shapefile(sf)
    comuna_id = []
    for i in comunas:
        comuna_id.append(df[df.NOM_COMUNA == i.upper()]
                         .index.get_values()[0])
    plot_map_fill_multiples_ids(title, comuna_id, sf,
                                       x_lim = None,
                                       y_lim = None,
```

```
                                    figsize = (11,9),
                                    color = color);
```

Plotting the southern comunes of Santiago Metropolitan Region:

```
south = ['alhué', 'calera de tango', 'buin', 'isla de maipo', 'el
bosque', 'paine', 'la granja', 'pedro aguirre cerda', 'lo espejo',
'puente alto', 'san joaquín', 'san miguel', 'pirque', 'san bernardo',
'san ramón', 'la cisterna', 'talagante', 'la pintana']

plot_comunas_2(sf, 'South', south, 'c')
```



10. Creating 'Heat Maps'

A very useful type of map is to fill a specific shape with a color, which "intensity" is proportional to a given value. Doing that, is possible to have a general overview about data distribution on a specific geographic area. For example, population distibution.

First we will create a function that once receiving a list of data, will split them on "bins". For each one of those beans will get a specific color assigned. For experience, usually 5 to 7 bins are good to have a good feeling of data distribution. We will use 6 bins and 4 different color palettes associated with those bins. You must select one of those bins at time.

```
def calc_color(data, color=None):
        if color   == 1: color_sq =
                         ['#dadaebFF','#bcbddcF0','#9e9ac8F0',
                         '#807dbaF0','#6a51a3F0','#54278fF0'];
                         colors = 'Purples';
        elif color == 2: color_sq =
                         ['#c7e9b4','#7fcdbb','#41b6c4',
                         '#1d91c0','#225ea8','#253494'];
                         colors = 'YlGnBu';
        elif color == 3: color_sq =
                         ['#f7f7f7','#d9d9d9','#bdbdbd',
                         '#969696','#636363','#252525'];
                         colors = 'Greys';
        elif color == 9: color_sq =
                         ['#ff0000','#ff0000','#ff0000',
                         '#ff0000','#ff0000','#ff0000']
        else:             color_sq =
                         ['#ffffd4','#fee391','#fec44f',
                         '#fe9929','#d95f0e','#993404'];
                         colors = 'YlOrBr';
        new_data, bins = pd.qcut(data, 6, retbins=True,
        labels=list(range(6)))
        color_ton = []
        for val in new_data:
            color_ton.append(color_sq[val])
        if color != 9:
            colors = sns.color_palette(colors, n_colors=6)
            sns.palplot(colors, 0.6);
            for i in range(6):
                print ("\n"+str(i+1)+': '+str(int(bins[i]))+
                       " => "+str(int(bins[i+1])-1), end =" ")
            print("\n\n   1   2   3   4   5   6")
        return color_ton, bins;
```

Both functions *plot_comunas()* and *plot_map_fill_multiples_ids* should be adapted to take advantage of this new colored scheme:

```
  def plot_comunas_data(sf, title, comunas, data=None,
                        color=None, print_id=False):
      '''
      Plot map with selected comunes, using specific color
      '''

      color_ton, bins = calc_color(data, color)
      df = read_shapefile(sf)
      comuna_id = []
      for i in comunas:
          i = conv_comuna(i).upper()
          comuna_id.append(df[df.NOM_COMUNA ==
                              i.upper()].index.get_values()[0])
      plot_map_fill_multiples_ids_tone(sf, title, comuna_id,
                                       print_id,
                                       color_ton,
                                       bins,
                                       x_lim = None,
                                       y_lim = None,
                                       figsize = (11,9));
```

and,

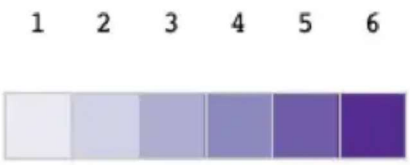
```
  def plot_map_fill_multiples_ids_tone(sf, title, comuna,
                                       print_id, color_ton,
                                       bins,
                                       x_lim = None,
                                       y_lim = None,
                                       figsize = (11,9)):
      '''
      Plot map with lim coordinates
      '''

      plt.figure(figsize = figsize)
      fig, ax = plt.subplots(figsize = figsize)
      fig.suptitle(title, fontsize=16)

  for shape in sf.shapeRecords():
          x = [i[0] for i in shape.shape.points[:]]
          y = [i[1] for i in shape.shape.points[:]]
          ax.plot(x, y, 'k')
```

```
    for id in comuna:
        shape_ex = sf.shape(id)
        x_lon = np.zeros((len(shape_ex.points),1))
        y_lat = np.zeros((len(shape_ex.points),1))
        for ip in range(len(shape_ex.points)):
            x_lon[ip] = shape_ex.points[ip][0]
            y_lat[ip] = shape_ex.points[ip][1]
        ax.fill(x_lon,y_lat, color_ton[comuna.index(id)])
        if print_id != False:
            x0 = np.mean(x_lon)
            y0 = np.mean(y_lat)
            plt.text(x0, y0, id, fontsize=10)
    if (x_lim != None) & (y_lim != None):
        plt.xlim(x_lim)
        plt.ylim(y_lim)
```

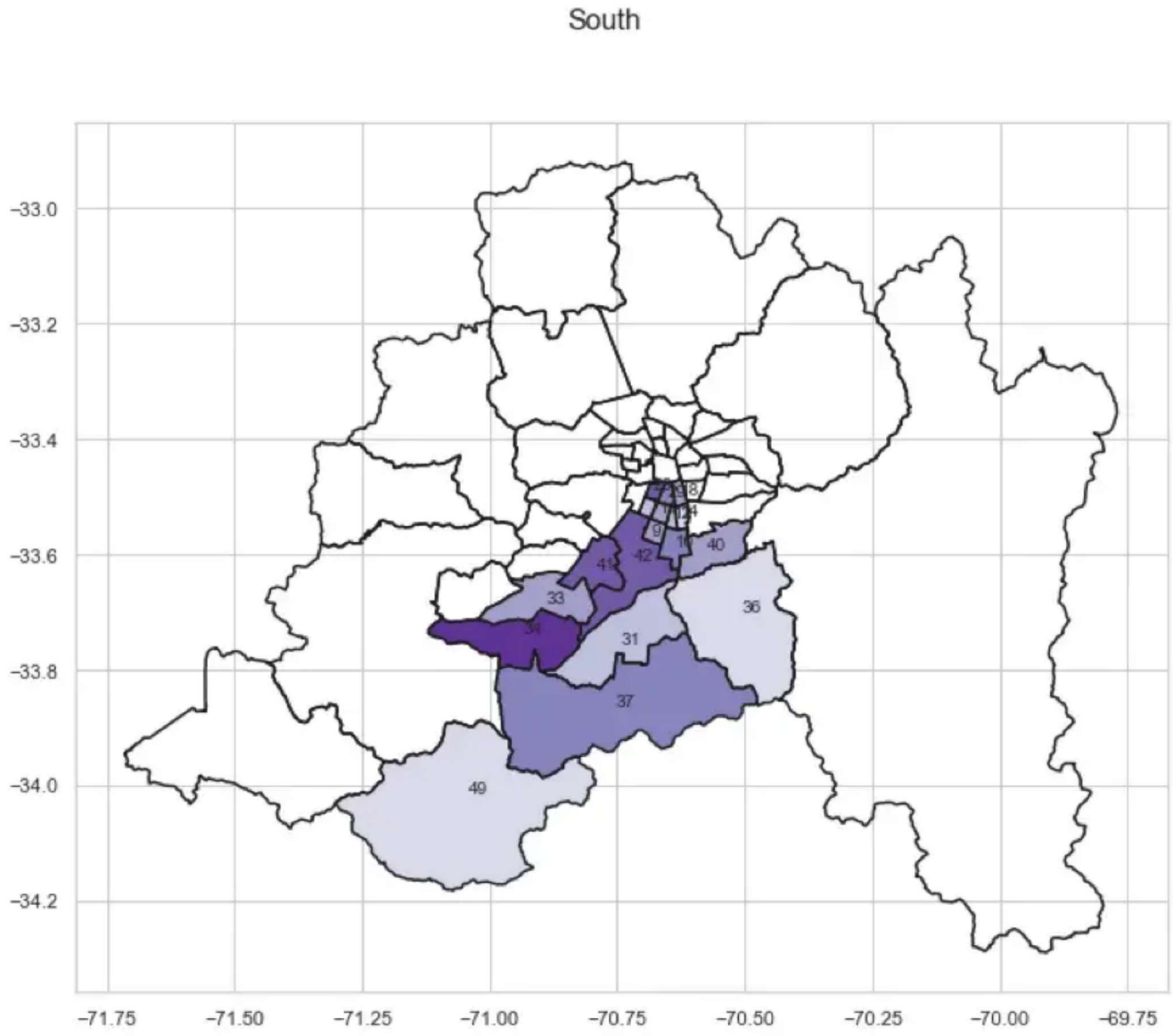
In order to test our new functions, let's take the previous list of shapes for the southern region of Santiago, associating a general value for each one of them. We will use the "color pallete #1 ('Purples'):

```
south = ['alhué', 'calera de tango', 'buin', 'isla de maipo', 'el
bosque', 'paine', 'la granja', 'pedro aguirre cerda', 'lo espejo',
'puente alto', 'san joaquín', 'san miguel', 'pirque', 'san bernardo',
'san ramón', 'la cisterna', 'talagante', 'la pintana']

data = [100, 2000, 300, 400000, 500, 600, 100, 2000, 300, 400, 500,
600, 100, 2000, 300, 400, 500, 600]

print_id = True # The shape id will be printed
color_pallete = 1 # 'Purples'

plot_comunas_data(sf, 'South', south, data, color_pallete, print_id)
```

```
1: 100 => 265
2: 266 => 365
3: 366 => 499
4: 500 => 599
5: 600 => 1999
6: 2000 => 399999
```




```
<Figure size 792x648 with 0 Axes>
```



Cool! Isn't ? ;-)

11. Plotting real data

To finish our overview regarding how to handle maps using Python, let's take some real data from last Chilean Census 2017 and apply those functions developed on part 10.

Reading dataset:

```
census_17 = pd.read_excel('./data/CENSO_2017_COMUNAS_RM.xlsx')
census_17.shape
```

```
Out:
(52,7)
```

Our dataset has 52 lines, what make sense once each line contains data related to each one of Santiago's comunas.

Taking a look at dataset:

	COMUNA	NOM_COMUNA	INMIGRANTES	PERSONAS	INM_PERC	TOTAL_VIV	PERS_VIV
0	13101	santiago	110732	404495	27.375369	193628	2.089032
1	13102	cerrillos	3620	80832	4.478424	24547	3.292948
2	13103	cerro navia	6022	132622	4.540725	38020	3.488217
3	13104	conchali	9223	126955	7.264779	37759	3.362245
4	13105	el bosque	3368	162505	2.072552	47941	3.389687

The column "PERSONAS" for example is related to the number of persons that live on that especific comuna. "TOTAL_VIV" is the total number of homes on that comuna and so on.

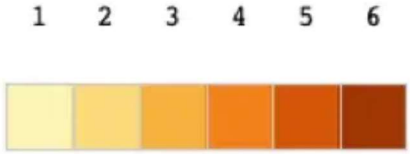
Plotting:

Let's apply our Map functions to analyze how the population is distributed on Santiago Metropolitan area.

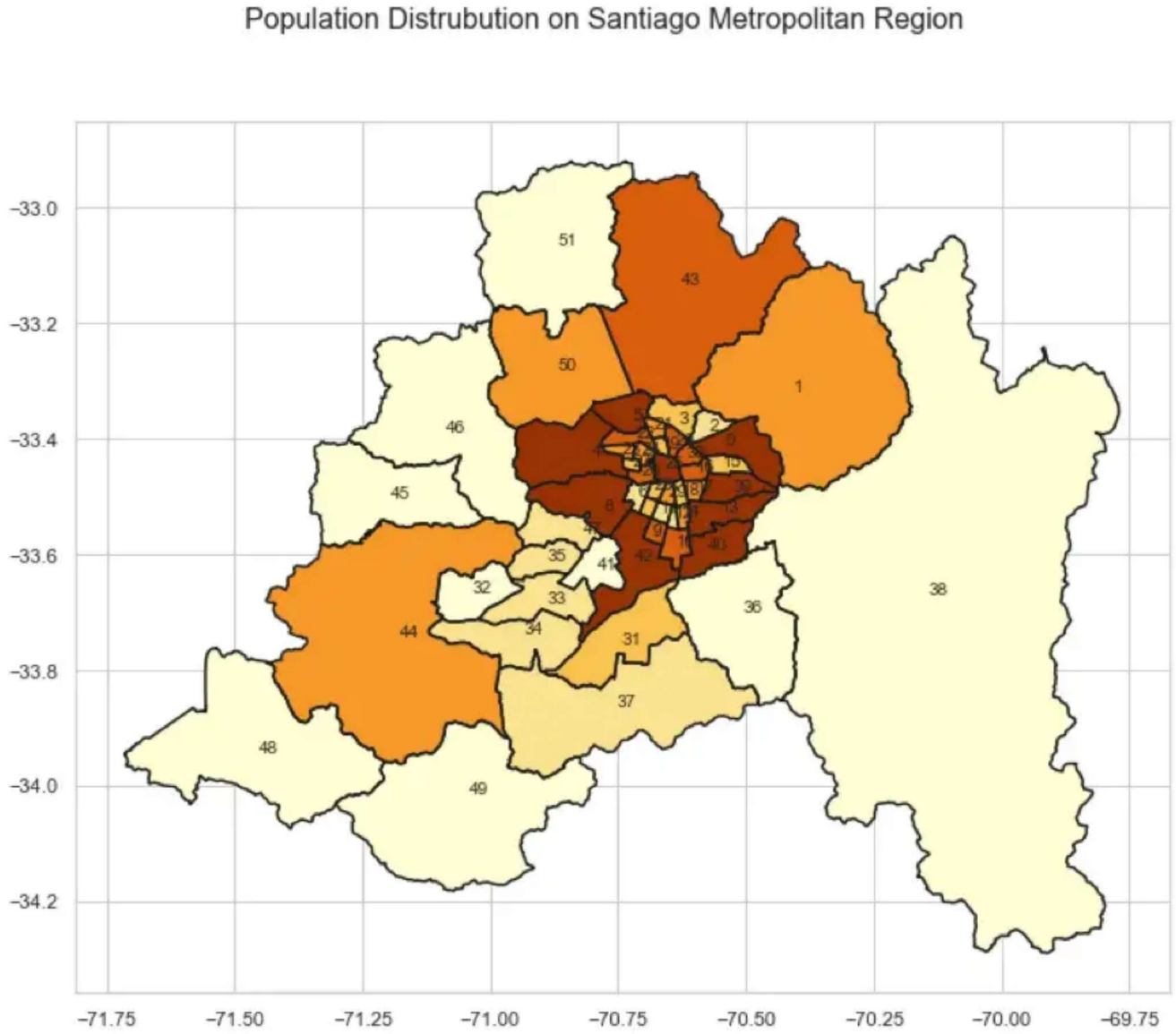
```
title = 'Population Distrubution on Santiago Metropolitan Region'
data = census_17.PERSONAS
```

```
names = census_17.NOM_COMUNA

plot_comunas_data(sf, title, names, data, 4, True)
```

```
1: 6444 => 36070
2: 36071 => 90200
3: 90201 => 101603
4: 101604 => 132621
5: 132622 => 209322
6: 209323 => 568105
```




```
<Figure size 792x648 with 0 Axes>
```



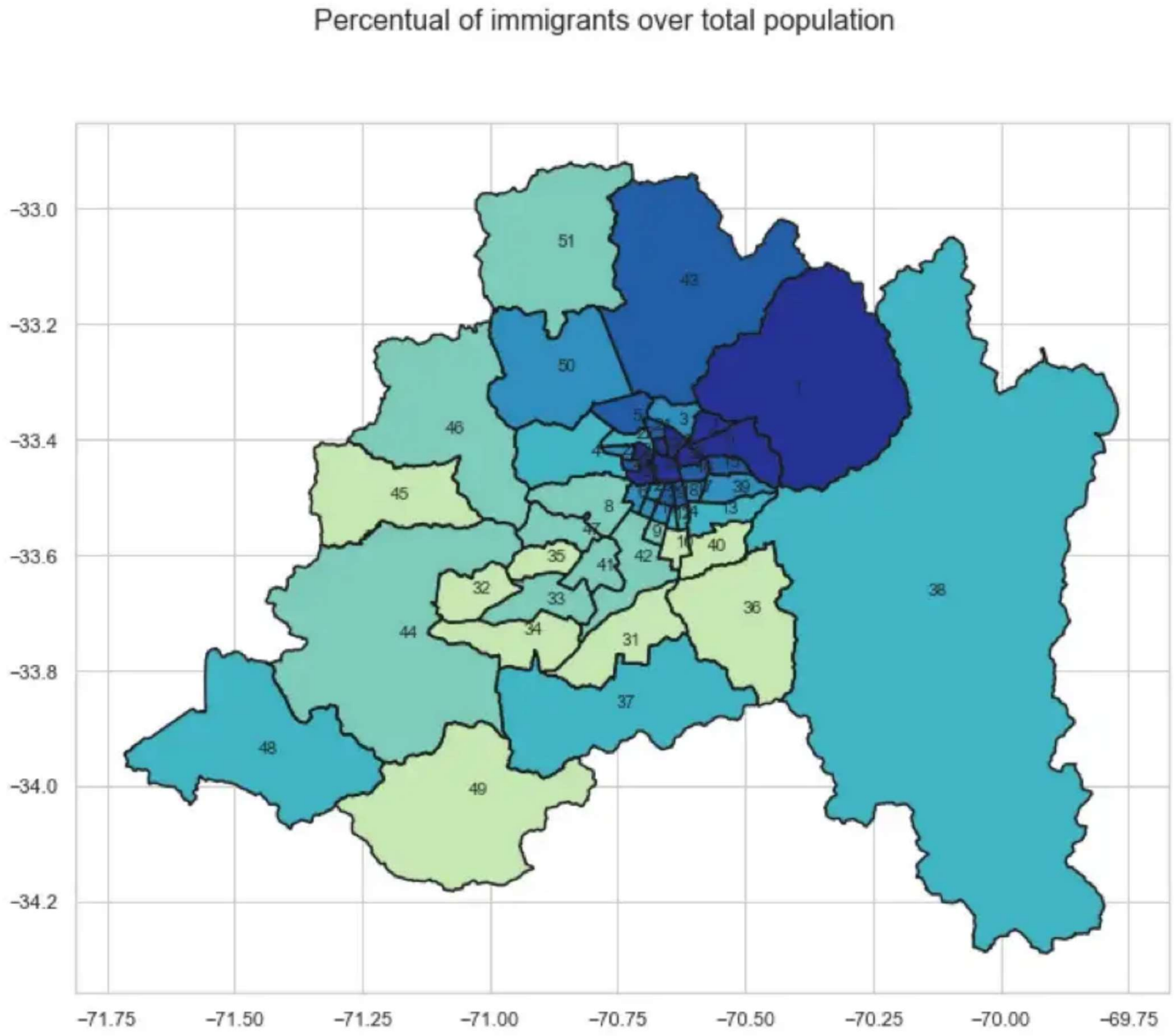
Great! We can see that population is heavily distributed around the center region of Metropolitan area! To the east (right on map), the population are sparce, what is logical, because this is the great Andes Montains! To the west and south are agricultural regions.

One more! Let's plot the percentual of immigrants over total population at Metropolitan Region of Santiago:

```
title = 'Percentual of immigrants over total population'
data = census_17.INM_PERC
names = census_17.NOM_COMUNA

plot_comunas_data(sf, title, names, data, 2, True)
```

```
1: 0 => 0
2: 1 => 1
3: 2 => 2
4: 3 => 4
5: 5 => 7
6: 8 => 28
```


```
<Figure size 792x648 with 0 Axes>
```



12. Conclusion

You can realize that at end you can create a map with only one line of code, using the 3 funcions developed on this article:

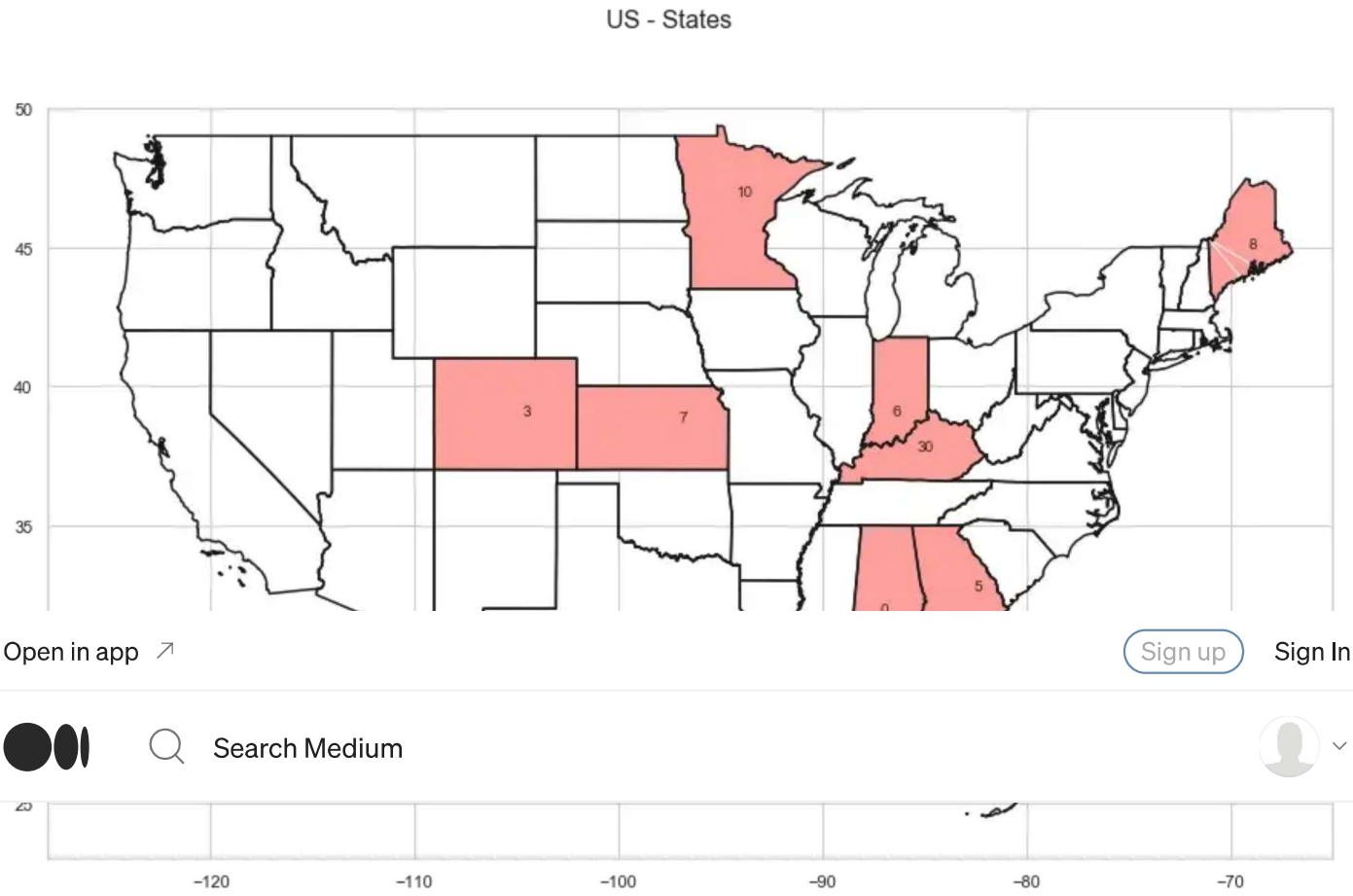
```
plot_comunas_data() .... that calls:

plot_map_fill_multiples_ids_tone() ... that calls:

calc_color()
```

What was developed on this article for Santiago Matropolitan area, can be easily adapted to be used with any vectorial map available at internet!

For example, you can go to US Census Bureau and download the Cartographic Boundary Shapefiles for US-States. Following what was done for Santiago,

```
shp_path = "./cb_2017_us_state_5m/cb_2017_us_state_5m.shp"
sf = shp.Reader(shp_path)

# Continental US
y_lim = (23, 50) # lat
x_lim = (-128, -65) # long

state_id = [0, 10, 3, 5, 6, 7, 8, 30]
plot_map_fill_multiples_ids("US - States", state_id, sf, x_lim,
                            y_lim, color = 'r', figsize = (15,9))
```

you can plot:



That's all folks!

Hope you have learned more about the fantastic world of Data Science!

The Jupyter Notebook and all data used on this article can be downloaded from my GitHub.

See you on my next article!

Saludos from the south of the world!

Marcelo

Data Science Python Geography Mapping Visualization

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. Take a look.