mapscaler

Release 0.0.3

Jun 26, 2020

Contents

CHAPTER 1

Contents

1.1 Installation

1.1.1 Installing from PyPI

```
pip install mapscaler
```

1.1.2 Dependencies

- NumPy
- GeoPandas
- Shapely

1.1.3 View Source on Github

View mapscaler on github

1.2 Loading Common Maps

1.2.1 Simple Example

Import MapScaler and load a map of the US Counties:

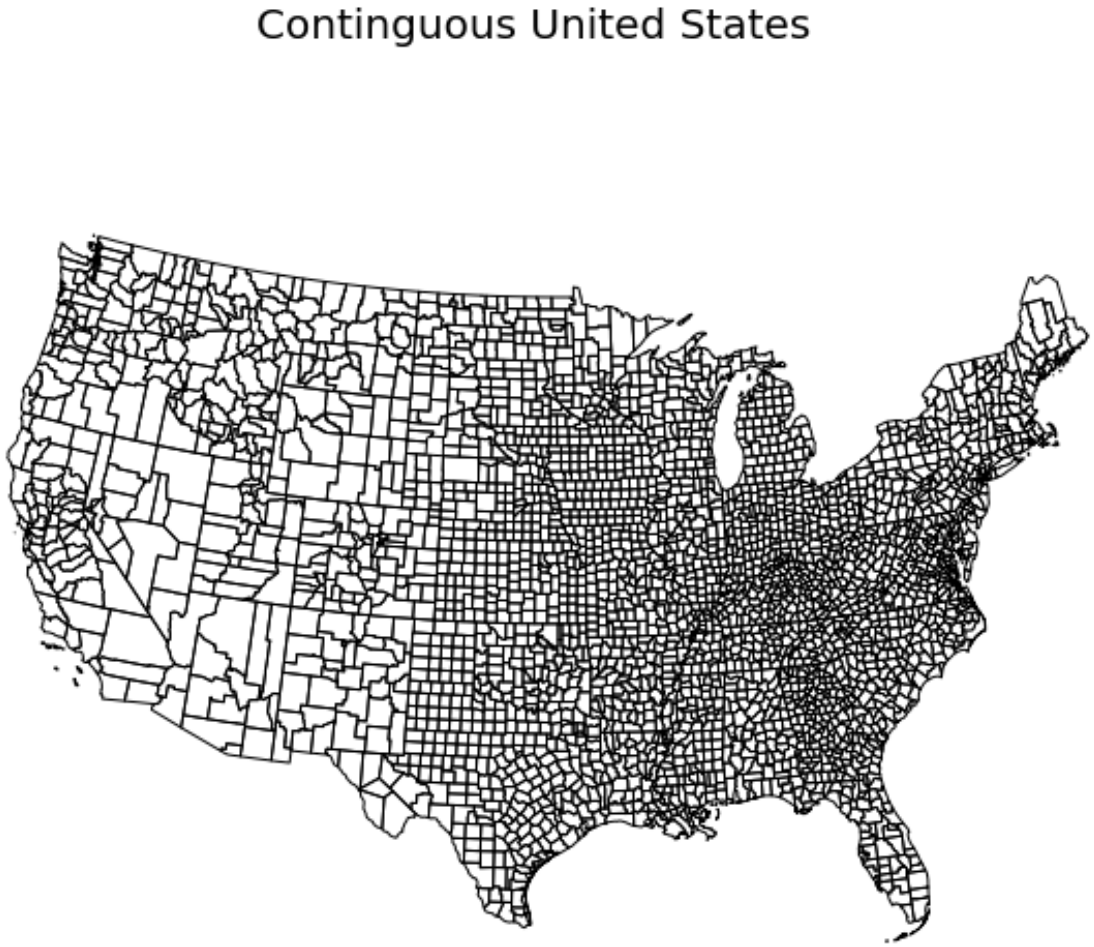
```
import mapscaler as ms
loader = ms.MapLoader()
df = loader.fetch_counties()['df']
```

That's it! You now have a map of the counties.

```
import matplotlib.pyplot as plt
import geoplot as gplt

#Reduce to the lower 48 for an easier demonstration
df = df[df.STATE_FIPS != '02'] # remove AK
df = df[df.STATE_FIPS != '15'] # remove HI
df = df[df.STATE_FIPS != '72'] # remove PR

gplt.polyplot(
    df,
    projection=gplt.crs.AlbersEqualArea(),
    figsize=(15,8),
    )
plt.suptitle('Contiguous United States', fontsize=20, ha='center')
plt.show()
```



1.2.2 Documentation

class mapscaler.**MapLoader**

Bases: object

Quickly load common maps of the US as GeoPandas Dataframes.

Note: In most cases, the least detailed map options are used for performance. All MapLoader methods return a `sources` item with links to find maps with higher detail, if available.

fetch_counties(*state_fips=None*)
Load a map of the US counties.

Parameters **state_fips** (*str*) – *Optional* - State FIPS code as a string. Default is `None` which loads all states.

Returns

`dict` with 2 keys:

df is a GeoPandas DataFrame, including a column of shape objects.

sources is a dict of links to the original map source.

Return type `dict`

fetch_states()
Load a map of the US states, including Puerto Rico.

Returns

`dict` with 2 keys:

df is a GeoPandas DataFrame, including a column of shape objects.

sources is a dict of links to the original map source.

Return type `dict`

1.3 Creating Shape Scalars from a Variable

In this tutorial, we'll use a GeoPandas dataframe with 2 columns: a `geometry` column of US county shapes, and a population column of numeric values. (Of course, the following steps will work for other variables besides population - any numeric input will do.

1.3.1 Data Prep and Cleaning

Import MapScaler and load a map of the US Counties:

```
import mapscaler as ms
loader = ms.MapLoader()
df = loader.fetch_counties()['df']
```

We'll restrict this example to the continuous 48 for easier visualizations, so let's drop Alaska, Hawaii, and Puerto Rico.

```
df = df[df.STATE_FIPS != '02'] # AK
df = df[df.STATE_FIPS != '15'] # HI
df = df[df.STATE_FIPS != '72'] # PR
```

1.3.2 Step 1: Choose a 'base' for scale

First, store the area of each shape as a columns in your dataframe. You can pull this in from an official GIS source (Such as the Census Bureau), or you can calculate the areas directly using the GeoPandas Shapely objects, which we'll do here:

```
df['area'] = [ shape.area for shape in df['geometry'] ]
```

While scalars clearly define proportions, they do not specify any sizes specifically. For example, Queens County, NY has about double the population of Mecklenburg, NC. This could be satisfied by respective areas of 2 and 1, 20 and 10, 2 million and 1 million, etc. You must specify some anchor for the overall scale of the new map.

There's no reliable programmatic way to choose this scale, because it depends highly on the distribution of shapes in your chart. Here's the extremes:

- You could choose the densest county for your base (New York County). New York County would remain the original size, and ALL other counties would shrink. The result would be mathematically accurate proportions between counties, but they would all be very small and spaced out, an ugly visualization.
- You could choose the least dense counties for your base (Garfield County, Montana). Garfield County would remain the same size, and ALL the other counties would expand dramatically. The increased counties would all overlap each other, and mapscaler will take a very long time to find a solution with no overlapping states, and by that point, the counties will have been rearranged so much that the solution may not be recognizeable as a map of the US.

In short, it's a tradeoff between having large individual shapes, and having a map with an overall shape that still resembles the original. Therefore, the best rule of thumb is to pick a base shape somewhere in the 60th-80th percentile of variable density. This will result in over half your shapes shrinking and less than half of them increasing. Start there and tweak as needed.

In this tutorial, we'll use Washtenaw County, Michigan. This was determined with a small amount of trail & error by running Shape Scaler, and inspecting the output, which is covered in the next section. Store the variable value (population) and the area of your base:

```
base_fips = '26161'
base_population = df[df.FIPS==base_fips]['EST_POP_2019'].values[0]
base_area = df[df.FIPS==base_fips]['area'].values[0]
```

1.3.3 Step 2: Create Scalars

Given base values for area and a variable `var`, this formula will calculate the appropriate scalar by which to adjust the area of given shape `abc`:

Formula Explanation

The goal is to scale the area of shape `abc` until it has the same variable-to-area ratio as the base, like so:

Solving for `scalar` gives:

Finally, when you scale a shape's coordinates by some scalar x, the area scales at $x2$ (see Additional Reading below). Therefore, to scale the area of a shape, the area scalar is the square root of each coordinate scalar.

```
df['scaleby'] = ( (df['EST_POP_2019']*base_area) / (df['area']*base_population) )**.5
```

This `scaleby` column is ready to be passed as input to *Shape Scaler* or *Bubble Scaler*, which will be covered next.

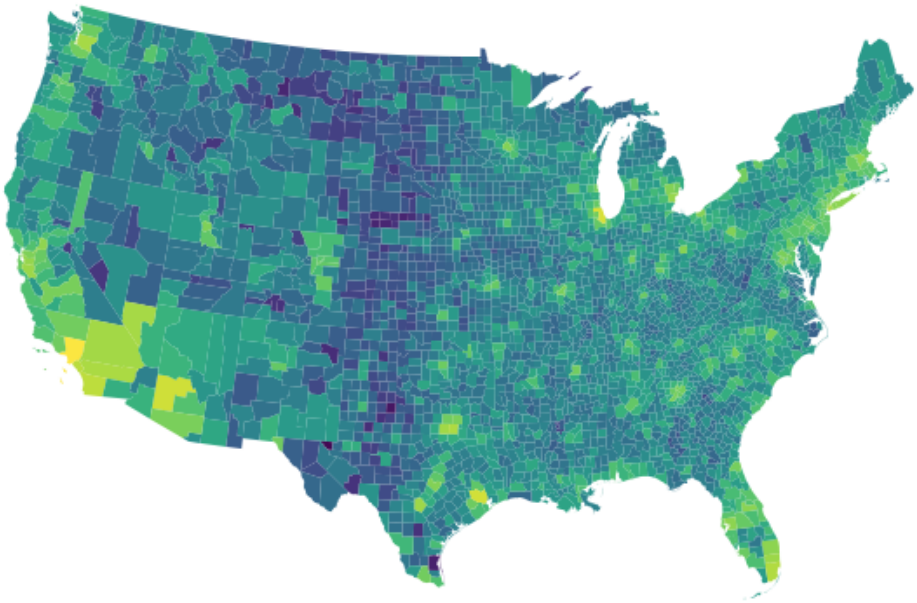
1.3.4 Additional Reading

- Square-Cube Law
- Calculating the scale factor to resize a polygon to a specific size

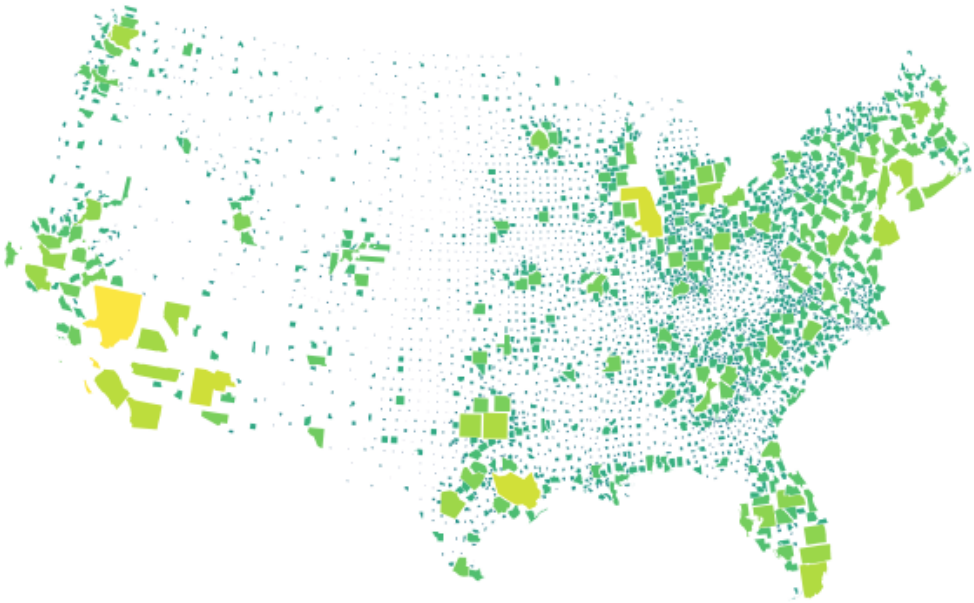
1.4 Shape Scaler

1.4.1 Simple Example

BEFORE



AFTER



NOTE: This example uses the variable `df` as seen in the example in *Loading Common Maps*.

```
import mapscaler as ms

ss = ms.ShapeScaler()
scaled_df = ss.scale_map(df, 'scaleby', map_vel=.001, group_vel=.15, verbose=True)
```

If `verbose`, *`scale_map()`* will print progress as it goes (truncated below):

```
Iteration 1
--73 overlapping groups remaining
Iteration 2
--79 overlapping groups remaining
...
[truncated]
...
Iteration 62
--2 overlapping groups remaining
Iteration 63
--1 overlapping groups remaining
Iteration 64
Separated in 64 iterations
```

Now, let's visualize the output, `scaled_df`:

```
import matplotlib.pyplot as plt
import geoplot as gplt
import numpy as np

gplt.choropleth(
    df, hue=[np.log(x) for x in df.EST_POP_2019],
    projection=gplt.crs.AlbersEqualArea(), cmap='viridis',
    figsize=(15,8),
)
plt.title('BEFORE', fontsize=30, loc='left')
plt.show()

gplt.choropleth(
    scaled_df, hue=[np.log(x) for x in df.EST_POP_2019],
    projection=gplt.crs.AlbersEqualArea(), cmap='viridis',
    figsize=(15,8),
)
plt.title('AFTER', fontsize=30, loc='left')
plt.show()
```

1.4.2 Documentation

class mapscaler.**ShapeScaler**

Bases: `mapscaler.mapscaler.BaseScaler`

get_group_centroid(*obj_list*)

Calculate the geometric centroid of a group of objects.

Parameters **obj_list** (*list*) – Iterable of Shapely objects (Polygon or MultiPolygon)

Returns [x,y] coordinates of the entire group's geometric centroid

Return type `list`

get_group_members(*original_df*, *property_col*)
Returns the members of each group, given a column from the original dataframe to print. Useful for debugging / inspecting groups that are too slow to separate.

Parameters

- **original_df** (*GeoPandas DataFrame*) – GeoPandas Dataframe previously passed to *scale_map()* method
- **property_col** (*str*) – String name of column in original_df that identifies each shape; Typically a name or ID

Returns key, value pairs where key is an arbitrary group number and value is a list of **property_col** values describing the group members

Return type dict

get_overlapping_groups(*df*, *geo*, *buffer*)
Return all groups of overlapping shapes in a map.

Parameters

- **df** (*GeoPandas DataFrame*) – GeoPandas Dataframe
- **geo** (*str*) – string name of the geometry column in **df**
- **buffer** (*float*) – Euclidean distance required between shapes before they are considered non-overlapping

Returns key, value pairs where key is is the group id and value is a list of shape ids.

Return type dict

index_geo_col(*df*, *geo*)
Returns a mapping of Shape IDs to their initial index in the dataframe.

Parameters

- **df** (*GeoPandas DataFrame*) – GeoPandas Dataframe
- **geo** (*str*) – string name of the geometry column in **df**

Returns key, value pairs where key is is a shape id, and value is its row index in **df**

Return type dict

index_overlapping_groups()
Return a mapping of Shape IDs to their current overlapping groups; Inverse of *get_overlapping_groups()*.

Returns key, value pairs where key is a shape id, and value is a group id

Return type dict

move_shape(*shape*, *movement*)
Move a Shapely Polygon by a given movement vector.

Parameters

- **shape** (*Shapely Polygon*) – Shape to be moved
- **movement** (*list or tuple*) – vector [x,y] describing the movement

Returns Shape with updated coordinates

Return type Shapely Polygon

nudge_shapes(*df*, *geo*, *map_vel*, *group_vel*)
Nudge overlapping shapes away from group and map centroids.

Parameters

- **df** (*GeoPandas DataFrame*) – GeoPandas Dataframe
- **geo** (*str*) – string name of the geometry column in **df**
- **map_vel** (*float*) – Velocity at which each shape is nudged away from the centroid of the whole map
- **group_vel** (*float*) – Velocity at which each shape is nudged away from the centroid of its respective group of overlapping shapes

Returns Dataframe with updated geometry column

Return type GeoPandas `DataFrame`

scale_map(*df*, *scaleby*, *geo='geometry'*, *map_vel=0.01*, *group_vel=0.1*, *buffer=0*, *max_iter=100*, *verbose=False*)
Automatically scale the parts of any map by any variable, without any overlapping shapes and with minimal distortion.

Parameters

- **df** (*GeoPandas DataFrame*) – GeoPandas Dataframe
- **scaleby** (*str*) – string name of the column in **df** with scalar values
- **geo** (*str*) – *Optional* - string name of the geometry column in **df**; default is `'geometry'`
- **map_vel** (*float*) – *Optional* - Velocity at which each shape is nudged away from the centroid of the whole map; dfault is `.01`
- **group_vel** (*float*) – *Optional* - Velocity at which each shape is nudged away from the centroid of its respective group of overlapping shapes; default is `.1`
- **buffer** (*float*) – *Optional* - Euclidean distance required between shapes before they are considered non-overlapping; default is `0`
- **max_iter** (*int*) – *Optional* - Maximum number of attempts to nudge shapes away from each other; default is `100`
- **verbose** (*boolean*) – *Optional* - Whether to print progress as shapes are separated; default is `False`

Returns Dataframe with updated geometry column

Return type GeoPandas `DataFrame`

scale_shapes(*df*, *scaleby*, *geo*)
Scale the Coordinates of all map shapes by their respective scalars.

Parameters

- **df** (*GeoPandas DataFrame*) – GeoPandas Dataframe
- **scaleby** (*str*) – string name of the column in **df** with scalar values
- **geo** (*str*) – string name of the geometry column in **df**

Warning: This function scales by coordinates. Scaling coordinates by x will scale the area of the shape by x^2. More info: *Creating Shape Scalars from a Variable*

separate_map(*df*, *geo*, *map_vel*, *group_vel*, *buffer*, *max_iter*, *verbose*)
Reposition shapes on a map so that none of them overlap.

Parameters

- **df** (*GeoPandas DataFrame*) – GeoPandas Dataframe
- **geo** (*str*) – String name of the geometry column in **df**
- **map_vel** (*float*) – Velocity at which each shape is nudged away from the centroid of the whole map
- **group_vel** (*float*) – Velocity at which each shape is nudged away from the centroid of its respective group of overlapping shapes
- **buffer** (*float*) – Euclidean distance required between shapes before they are considered non-overlapping
- **max_iter** (*int*) – Maximum number of attempts to nudge shapes away from each other
- **verbose** (*boolean*) – Whether to print progress as shapes are separated

Returns Dataframe with updated geometry column

Return type GeoPandas `DataFrame`

update_group_centroids(*df*, *geo*)
Calculate the geometric centroid of all overlapping groups.

Parameters

- **df** (*GeoPandas DataFrame*) – GeoPandas Dataframe
- **geo** (*str*) – string name of the geometry column in **df**

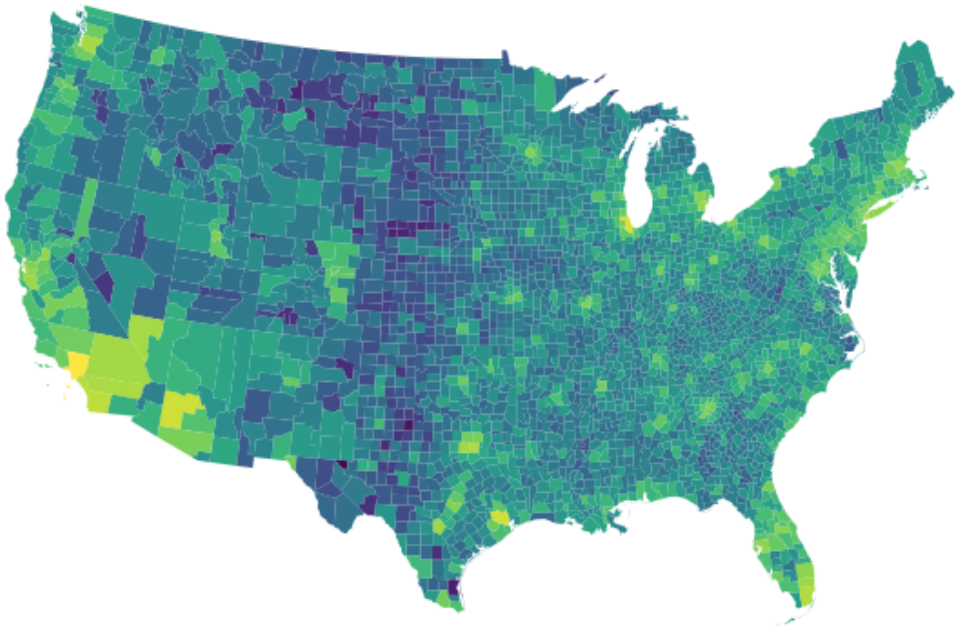
Returns key, value pairs where key is the group id, and value is the group centroid

Return type `dict`

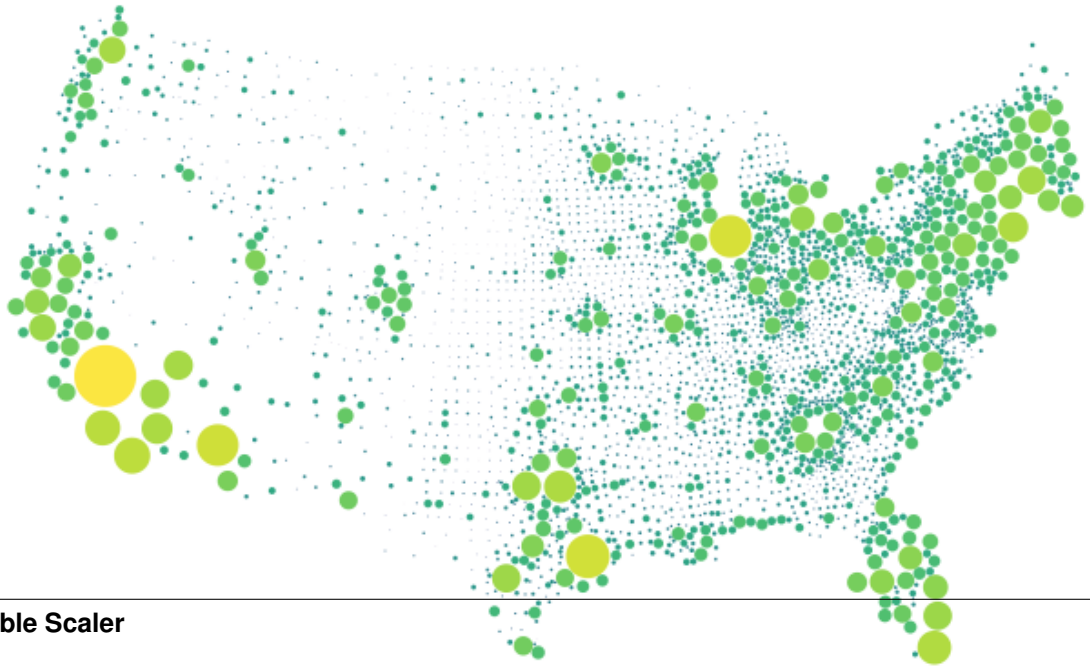
1.5 Bubble Scaler

1.5.1 Simple Example

BEFORE



AFTER



NOTE: This example uses the variable `df` as seen in the example in *Loading Common Maps*.

```
import mapscaler as ms
import numpy as np

bs = ms.BubbleScaler()
bubble_df = bs.scale_map(df, 'scaleby', usa_albers=True,
                map_vel=.001, group_vel=.15, verbose=True)
```

If `verbose`, *`scale_map()`* will print progress as it goes (truncated below):

```
Iteration 1
--93 overlapping groups remaining
Iteration 2
--89 overlapping groups remaining
...
[truncated]
...
Iteration 42
--2 overlapping groups remaining
Iteration 43
--2 overlapping groups remaining
Iteration 44
Separated in 44 iterations
```

Now, let's visualize the output `bubble_df`:

```
import matplotlib.pyplot as plt
import geoplot as gplt
import numpy as np

gplt.choropleth(
    df, hue=[np.log(x) for x in df.EST_POP_2019],
    projection=gplt.crs.AlbersEqualArea(), cmap='viridis',
    figsize=(15,8),
)
plt.title('BEFORE', fontsize=30, loc='left')
plt.show()

gplt.choropleth(
    bubble_df, hue=[np.log(x) for x in df.EST_POP_2019],
    figsize=(15,8), linewidth=0, cmap='viridis',
)
plt.title('AFTER', fontsize=30, loc='left')
plt.show()
```

1.5.2 Documentation

class mapscaler.**BubbleScaler**

Bases: `mapscaler.mapscaler.BaseScaler`

convert_to_bubbles(*df*, *geo*)

Convert all shapes in a geopandas dataframe to circles, retaining areas and centroid coordinates.

Parameters

- **df** (*GeoPandas DataFrame*) – GeoPandas Dataframe

- **geo** (*str*) – String name of the geometry column in **df**

Returns Dataframe with updated geometry column, each shape converted to a same-area circle

Return type GeoPandas `DataFrame`

get_group_centroid(*obj_list*)
Calculate the geometric centroid of a group of objects.

Parameters **obj_list** (*list*) – Iterable of Shapely objects (Polygon or MultiPolygon)

Returns [x,y] coordinates of the entire group's geometric centroid

Return type `list`

get_group_members(*original_df*, *property_col*)
Returns the members of each group, given a column from the original dataframe to print. Useful for debugging / inspecting groups that are too slow to separate.

Parameters

- **original_df** (*GeoPandas DataFrame*) – GeoPandas Dataframe previously passed to *scale_map()* method
- **property_col** (*str*) – String name of column in original_df that identifies each shape; Typically a name or ID

Returns key, value pairs where key is an arbitrary group number and value is a list of **property_col** values describing the group members

Return type `dict`

get_overlapping_groups(*df*, *geo*, *buffer*)
Return all groups of overlapping shapes in a map.

Parameters

- **df** (*GeoPandas DataFrame*) – GeoPandas Dataframe
- **geo** (*str*) – string name of the geometry column in **df**
- **buffer** (*float*) – Euclidean distance required between shapes before they are considered non-overlapping

Returns key, value pairs where key is is the group id and value is a list of shape ids.

Return type `dict`

index_geo_col(*df*, *geo*)
Returns a mapping of Shape IDs to their initial index in the dataframe.

Parameters

- **df** (*GeoPandas DataFrame*) – GeoPandas Dataframe
- **geo** (*str*) – string name of the geometry column in **df**

Returns key, value pairs where key is is a shape id, and value is its row index in **df**

Return type `dict`

index_overlapping_groups()
Return a mapping of Shape IDs to their current overlapping groups; Inverse of *get_overlapping_groups()*.

Returns key, value pairs where key is a shape id, and value is a group id

Return type `dict`

move_shape(*shape*, *movement*)
Move a Shapely Polygon by a given movement vector.

Parameters

- **shape** (*Shapely Polygon*) – Shape to be moved
- **movement** (*list or tuple*) – vector [x,y] describing the movement

Returns Shape with updated coordinates

Return type Shapely `Polygon`

nudge_shapes(*df*, *geo*, *map_vel*, *group_vel*)
Nudge overlapping shapes away from group and map centroids.

Parameters

- **df** (*GeoPandas DataFrame*) – GeoPandas Dataframe
- **geo** (*str*) – string name of the geometry column in **df**
- **map_vel** (*float*) – Velocity at which each shape is nudged away from the centroid of the whole map
- **group_vel** (*float*) – Velocity at which each shape is nudged away from the centroid of its respective group of overlapping shapes

Returns Dataframe with updated geometry column

Return type GeoPandas `DataFrame`

scale_map(*df*, *scaleby*, *geo='geometry'*, *usa_albers=False*, *map_vel=0.01*, *group_vel=0.1*, *buffer=0*, *max_iter=100*, *verbose=False*)
Convert all shapes in a map to circles, and automatically scale the parts of any map by any variable, without any overlapping shapes and with minimal distortion.

Parameters

- **df** (*GeoPandas DataFrame*) – GeoPandas Dataframe
- **scaleby** (*str*) – string name of the column in **df** with scalar values
- **geo** (*str*) – *Optional* - string name of the geometry column in **df**; default is `'geometry'`
- **usa_albers** (*boolean*) – *Optional* - Whether to apply an albers projection prior to converting to bubbles. More Info on Projections. Default is `False`
- **map_vel** (*float*) – *Optional* - Velocity at which each shape is nudged away from the centroid of the whole map; dfault is `.01`
- **group_vel** (*float*) – *Optional* - Velocity at which each shape is nudged away from the centroid of its respective group of overlapping shapes; default is `.1`
- **buffer** (*float*) – *Optional* - Euclidean distance required between shapes before they are considered non-overlapping; default is `0`
- **max_iter** (*int*) – *Optional* - Maximum number of attempts to nudge shapes away from each other; default is `100`
- **verbose** (*boolean*) – *Optional* - Whether to print progress as shapes are separated; default is `False`

Returns Dataframe with updated geometry column

Return type GeoPandas `DataFrame`

scale_shapes (*df*, *scaleby*, *geo*)
Scale the Coordinates of all map shapes by their respective scalars.

Parameters

- **df** (*GeoPandas DataFrame*) – GeoPandas Dataframe
- **scaleby** (*str*) – string name of the column in **df** with scalar values
- **geo** (*str*) – string name of the geometry column in **df**

> **Warning:** This function scales by coordinates. Scaling coordinates by x will scale the area of the shape by x^2. More info: *Creating Shape Scalars from a Variable*

separate_map (*df*, *geo*, *map_vel*, *group_vel*, *buffer*, *max_iter*, *verbose*)
Reposition shapes on a map so that none of them overlap.

Parameters

- **df** (*GeoPandas DataFrame*) – GeoPandas Dataframe
- **geo** (*str*) – String name of the geometry column in **df**
- **map_vel** (*float*) – Velocity at which each shape is nudged away from the centroid of the whole map
- **group_vel** (*float*) – Velocity at which each shape is nudged away from the centroid of its respective group of overlapping shapes
- **buffer** (*float*) – Euclidean distance required between shapes before they are considered non-overlapping
- **max_iter** (*int*) – Maximum number of attempts to nudge shapes away from each other
- **verbose** (*boolean*) – Whether to print progress as shapes are separated

Returns Dataframe with updated geometry column

Return type GeoPandas `DataFrame`

update_group_centroids (*df*, *geo*)
Calculate the geometric centroid of all overlapping groups.

Parameters

- **df** (*GeoPandas DataFrame*) – GeoPandas Dataframe
- **geo** (*str*) – string name of the geometry column in **df**

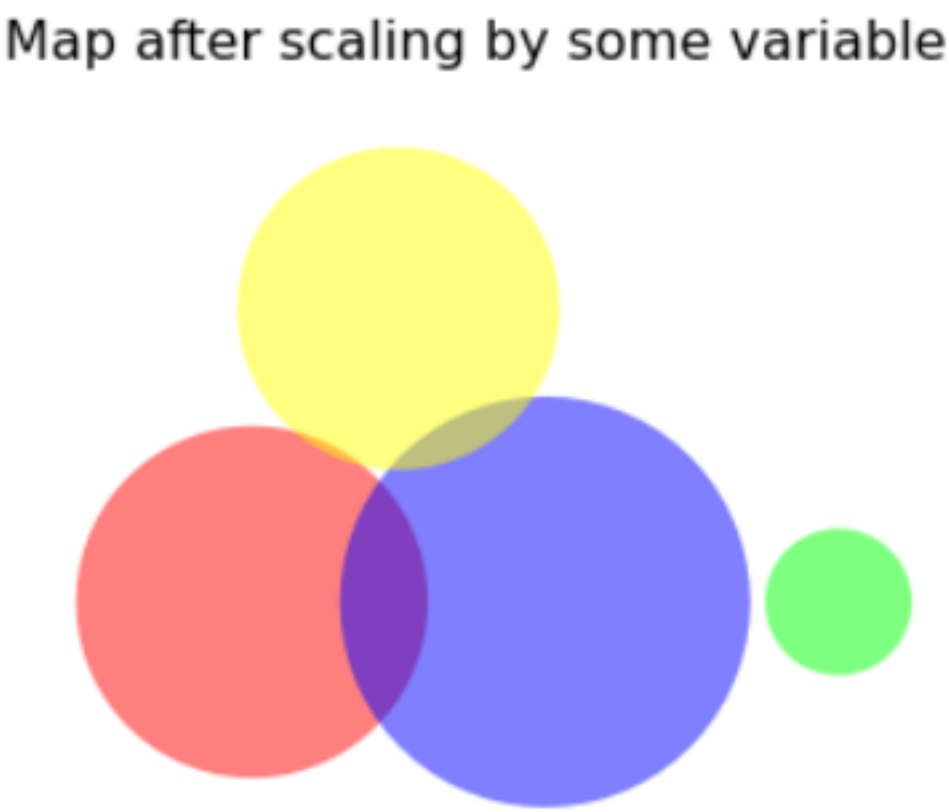
Returns key, value pairs where key is the group id, and value is the group centroid

Return type `dict`

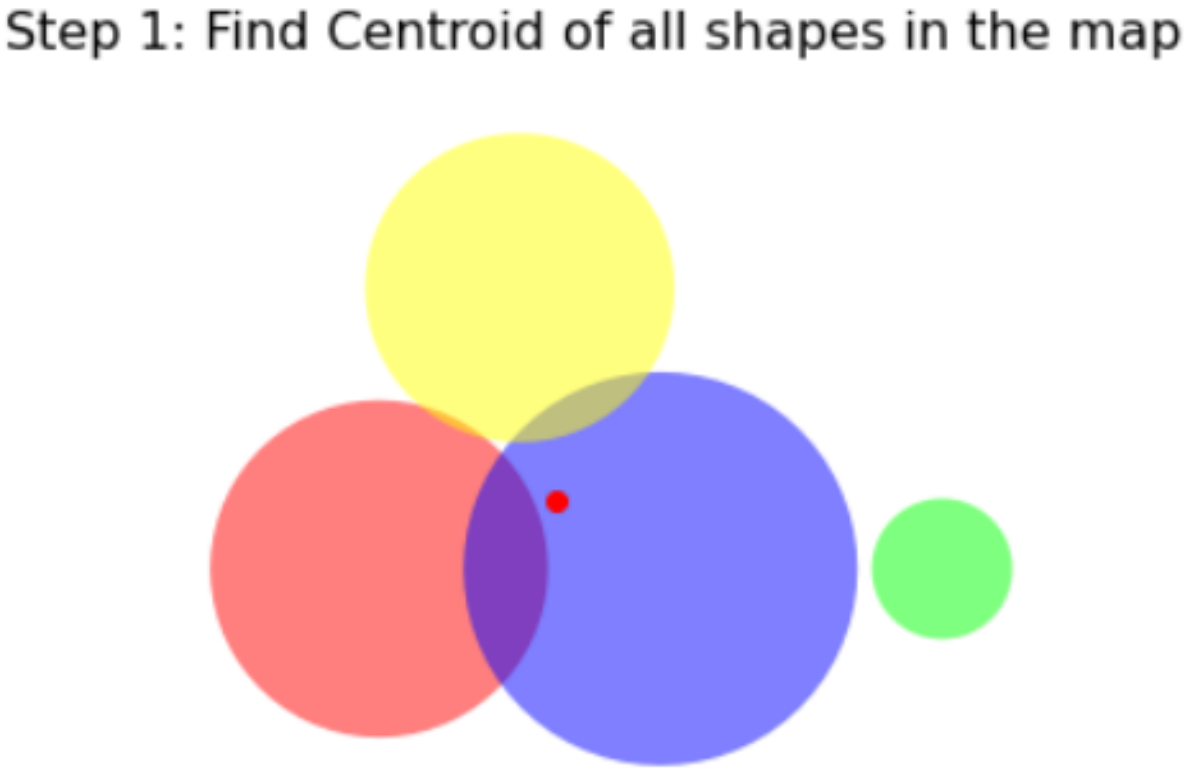
1.6 Technical Details: Rearranging shapes

Shape Scaler works iteratively, gently and strategically nudging the polygons in your map until they don't overlap. Below is a quick overview, using 4 circles to create a minimalist example.

Suppose these circles are scaled by a variable, and as a consequence, there's some unwanted overlap:

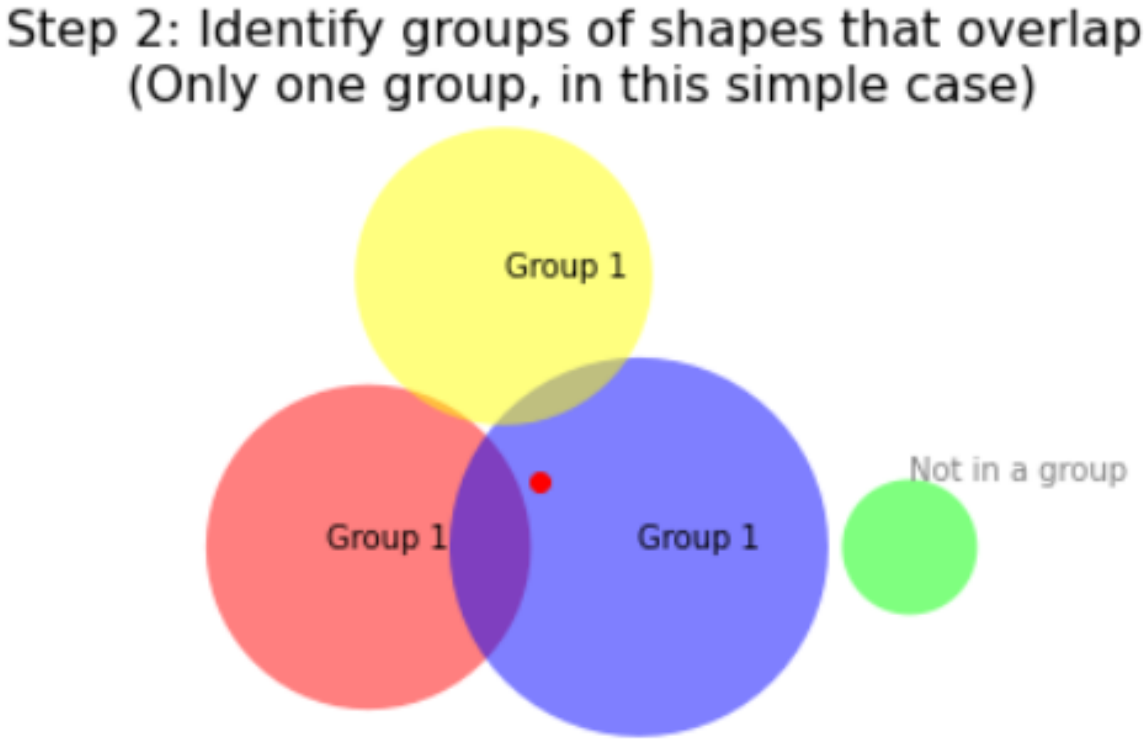


The first step to removing overlap is to find the geometric centroid of the map, all shapes included:

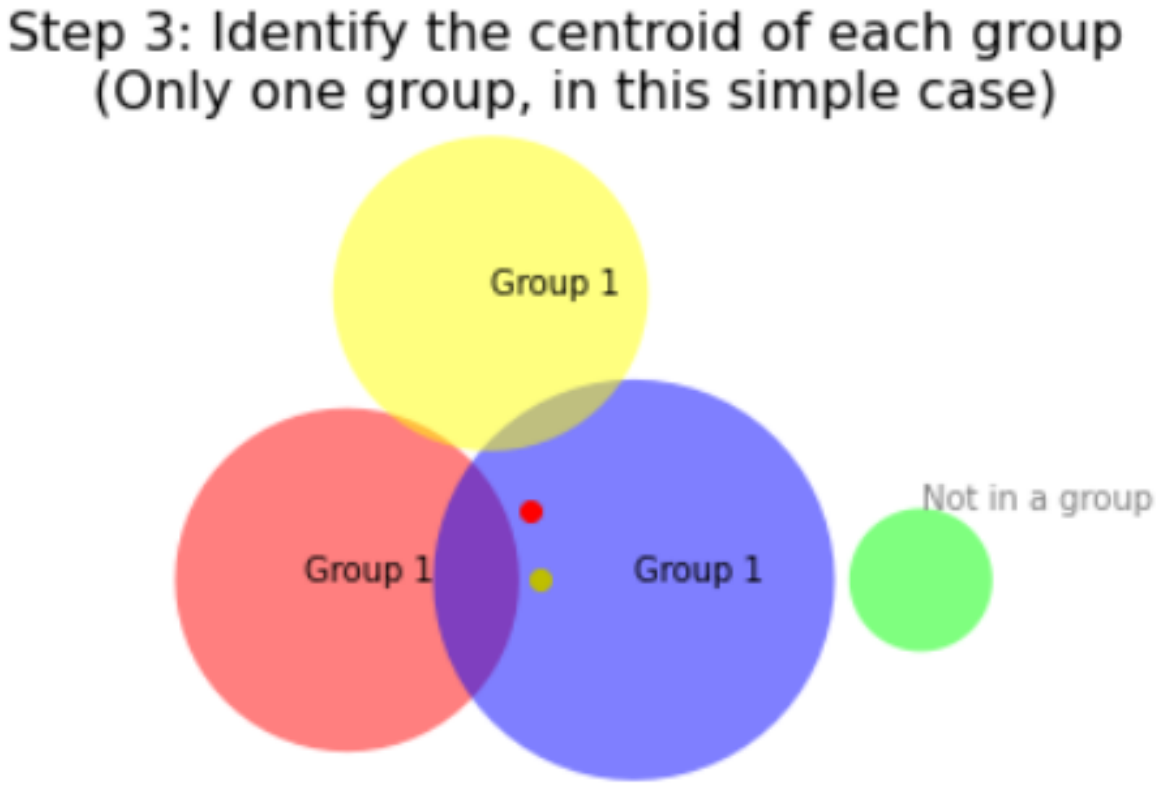


Next, organize all the shapes into groups with other shapes they overlap. Shapes that don't overlap any others do not

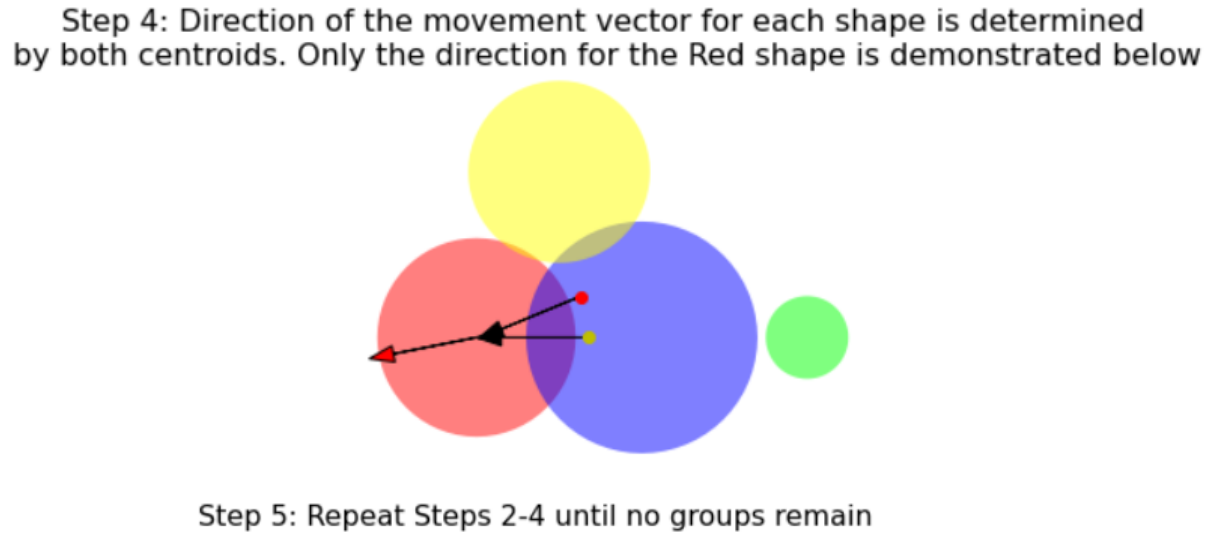
need to be moved, so they are not members of any group.



Next, find the centroid of each group:



The direction that each shape will move is a function of its position in relation to BOTH the map centroid, and its group centroid. The degree to which each centroid relation is considered in the formation of the direction vector is determined by the arguments `map_vel` and `group_vel`. In most cases, the group centroid should be the primary driver of the direction vector, and the map centroid should just be included minimally to achieve faster resolution in maps with dense groups.



The *Bubble Scaler* works the same way after first converting all shapes in your map to circles with the equivalent area.

1.7 License

mapscaler is licensed under the **GNU General Public License v3.0**. This license can be read here.

1.8 Contact

Hi! if you have questions, you can email me at conditg@gmail.com.

Index