Geosnap Tutorial

Stephen Francisco, Kate Oxx

April 14, 2021

This tutorial will showcase tools from the Geosnap package for Python. Geosnap helps spatial analysts identify, explore, and model neighborhoods statically or temporally based on parameters chosen by the user. It contains built-in data from the past three censuses (1990, 2000, and 2010) as well a series of other similar datasets containing American geographic and demographic data but also allows the user to import their own data.

In this tutorial we will cover boundary harmonization for standardizing geographic units over time, geodemographics and regionalization, and simulating neighborhood change – just part of the functionality of the Geosnap package.

0.0.1 Geosnap Installation & Data Preparation

Installation The makers of Geosnap recommend installing with anaconda. We recommend you try and use the python-unstable env where we installed pysal last week. (Alternately, you might want to use the gus5031 env.) To install, open the Miniconda prompt, navigate to the proper environment, and use the following commands:

[ ]:
```
conda activate <your_env>
conda install -c conda-forge geosnap
```

Be sure to confirm it's installed, or if you installed it previously, make sure you're running version 0.6.0.

0.0.2 Built-In Data

Geosnap comes "batteries included" with multiple demographic datasets, including state, county, and micropolitan and metropolitan statistical (MSA) boundaries. In order to view the list of datasets, run the following commands:

[3]:
```
from geosnap import datasets
dir(datasets)
```

[3]:
```
['acs',
 'blocks_2000',
 'blocks_2010',
 'codebook',
 'counties',
 'ltdb',
 'msa_definitions',
 'msas',
 'ncdb',
 'states',
 'tracts_1990',
 'tracts_2000',
 'tracts_2010']
```

Each dataset is a pandas (or geopandas) dataframe, whose attributes and methods can be used.

Next, we will import geosnap's central data structure: **Community** >"A Community is a dataset that stores information about a collection of neighborhoods over several time periods, including each neighborhood's physical, socioeconomic, and demographic attributes and its demarcated boundaries. Under the hood, each Community is simply a long-form geopandas geodataframe with some associated metadata."1

[4]:
```python
from geosnap import Community
```

Processing times can be fairly long when working with large datasets (such as large cities or entire states). Therefore, we recommend storing the census data locally to improve performance using the following commands:

[5]:
```python
from geosnap.io import store_census
store_census()
```

```
Loading manifest:
100%|                                     | 8/8
[00:00<00:00, 4.04k/s]
Copying objects:
100%|                                  | 253M/253M
[00:07<00:00, 34.3MB/s]

Successfully installed package 'census/tracts_cartographic', tophash=e848bc9
from s3://spatial-ucr

Loading manifest:
100%|                                     | 9/9
[00:00<00:00, 9.03k/s]
Copying objects:
100%|                                  | 185M/185M
[00:03<00:00, 51.3MB/s]

Successfully installed package 'census/administrative', tophash=3eecdec from
s3://spatial-ucr
```

To harness the census data built into Geosnap, we will have to assign it to a variable, thus creating our own Community. In this case, we will use the Community.from_census constructor to obtain demographic data for our county of interest (Philadephia) using FIPS county codes:

[6]:
```
phl = Community.from_census(county_fips="42101")
```

To access the underlying data, call its gdf attribute to return a geodataframe (gdf). Here, we use the head command to show the first five rows in the gdf:

[7]:
```
phl.gdf.head()
```

[7]:
```
             geoid  n_mexican_pop  n_cuban_pop  n_puerto_rican_pop  \
29740  42101036500         9185.0          9.0                79.0
29758  42101035800         6012.0         15.0                30.0
29773  42101035900         5354.0         16.0                39.0
29775  42101036400          528.0          1.0                17.0
29785  42101035700         8529.0         15.0                55.0

       n_total_housing_units  n_vacant_housing_units  \
29740                 3684.0                   248.0
29758                 2344.0                   196.0
29773                 2235.0                    82.0
29775                    2.0                     0.0
29785                 3869.0                   169.0

       n_occupied_housing_units  n_owner_occupied_housing_units  \
29740                    3436.0                          2352.0
29758                    2148.0                          1865.0
29773                    2153.0                          1538.0
29775                       2.0                             1.0
29785                    3700.0                          1361.0

       n_renter_occupied_housing_units  n_white_persons  …  \
29740                           1084.0        8880102.0  …
29758                            283.0         572268.0  …
29773                            615.0         504466.0  …
29775                              1.0           3233.0  …
29785                           2339.0         796592.0  …

       p_irish_born_pop  p_italian_born_pop  p_poverty_rate_children  \
29740               NaN                 NaN                      NaN
29758               NaN                 NaN                      NaN
29773               NaN                 NaN                      NaN
29775               NaN                 NaN                      NaN
29785               NaN                 NaN                      NaN
```

```
       p_poverty_rate_hispanic  p_russian_born_pop  p_scandanavian_born_pop  \
29740                      NaN                 NaN                      NaN
29758                      NaN                 NaN                      NaN
29773                      NaN                 NaN                      NaN
29775                      NaN                 NaN                      NaN
29785                      NaN                 NaN                      NaN

       p_scandanavian_pop  n_total_pop_sample  p_female_labor_force  \
29740                 NaN                 NaN                   NaN
29758                 NaN                 NaN                   NaN
29773                 NaN                 NaN                   NaN
29775                 NaN                 NaN                   NaN
29785                 NaN                 NaN                   NaN

       p_black_persons
29740              NaN
29758              NaN
29773              NaN
29775              NaN
29785              NaN

[5 rows x 195 columns]
```

In order to view all column names, we can iterate through the series and print each column name:

[8]:
```python
columns = phl.gdf.columns
for col in columns:
    print (col)
```

```
geoid
n_mexican_pop
n_cuban_pop
n_puerto_rican_pop
n_total_housing_units
n_vacant_housing_units
n_occupied_housing_units
n_owner_occupied_housing_units
n_renter_occupied_housing_units
n_white_persons
n_nonhisp_white_persons
n_black_persons
n_nonhisp_black_persons
n_hispanic_persons
n_native_persons
n_hawaiian_persons
n_asian_indian_persons
n_chinese_persons
n_filipino_persons
```

n_japanese_persons
n_korean_persons
n_asian_persons
n_vietnamese_persons
n_white_age_distribution
n_white_under_15
n_white_over_60
n_white_over_65
n_black_age_distribution
n_black_under_15
n_black_over_60
n_black_over_65
n_hispanic_age_distribution
n_hispanic_under_15
n_hispanic_over_60
n_hispanic_over_65
n_native_age_distribution
n_native_under_15
n_native_over_60
n_native_over_65
n_asian_age_distribution
n_asian_under_15
n_total_pop
n_russian_pop
n_italian_pop
n_german_pop
n_irish_pop
n_foreign_born_pop
n_recent_immigrant_pop
n_naturalized_pop
n_age_5_older
n_other_language
n_limited_english
median_home_value
median_contract_rent
n_structures_30_old
n_occupied_housing_units_sample
n_household_recent_move
n_persons_under_18
n_persons_over_60
n_persons_over_75
n_persons_over_15
n_persons_over_25
n_married
n_widowed_divorced
n_total_families
n_female_headed_families
n_female_over_16

n_female_labor_force
n_labor_force
n_unemployed_persons
n_employed_over_16
n_employed_professional
n_employed_manufacturing
n_employed_self_employed
n_civilians_over_16
n_veterans
n_civilians_16_64
n_disabled
median_household_income
n_total_households
n_white_households
n_black_households
n_hispanic_households
n_asian_households
per_capita_income
n_poverty_determined_persons
n_poverty_persons
n_poverty_over_65
n_poverty_determined_families
n_poverty_determined_white
n_poverty_white
n_poverty_determined_black
n_poverty_black
n_poverty_determined_native
n_poverty_native
n_poverty_determined_asian
n_poverty_asian
n_edu_college_greater
n_edu_hs_less
p_mexican_pop
p_cuban_pop
p_puerto_rican_pop
p_russian_pop
p_italian_pop
p_german_pop
p_irish_pop
p_foreign_born_pop
p_recent_immigrant_pop
p_naturalized_pop
p_other_language
p_limited_english
n_housing_units_multiunit_structures_denom
n_total_housing_units_sample
p_vacant_housing_units
p_owner_occupied_units

```
p_structures_30_old
p_household_recent_move
p_persons_under_18
p_persons_over_60
p_persons_over_75
p_married
p_widowed_divorced
p_female_headed_families
p_nonhisp_white_persons
p_nonhisp_black_persons
p_hispanic_persons
p_native_persons
p_asian_persons
p_hawaiian_persons
p_asian_indian_persons
p_chinese_persons
p_filipino_persons
p_japanese_persons
p_korean_persons
p_vietnamese_persons
p_white_under_15
p_white_over_60
p_white_over_65
p_black_under_15
p_black_over_60
p_black_over_65
p_hispanic_under_15
p_hispanic_over_60
p_hispanic_over_65
p_native_under_15
p_native_over_60
p_native_over_65
p_asian_under_15
p_edu_hs_less
p_edu_college_greater
p_unemployment_rate
p_employed_professional
p_employed_manufacturing
p_employed_self_employed
p_veterans
p_disabled
p_poverty_rate
p_poverty_rate_over_65
p_poverty_rate_white
p_poverty_rate_black
p_poverty_rate_native
p_poverty_rate_asian
geometry
```

```
year
median_income_asianhh
median_income_blackhh
median_income_hispanichh
median_income_whitehh
n_asian_over_60
n_asian_over_65
n_civilians_over_18
n_german_born_pop
n_housing_units_multiunit_structures
n_irish_born_pop
n_italian_born_pop
n_poverty_determined_hispanic
n_poverty_families_children
n_poverty_hispanic
n_russian_born_pop
n_scandaniavian__born_pop
n_scandaniavian_pop
p_asian_over_60
p_asian_over_65
p_german_born_pop
p_housing_units_multiunit_structures
p_irish_born_pop
p_italian_born_pop
p_poverty_rate_children
p_poverty_rate_hispanic
p_russian_born_pop
p_scandanavian_born_pop
p_scandanavian_pop
n_total_pop_sample
p_female_labor_force
p_black_persons
```

0.0.3 Boundary Harmonization with Areal Interpolation

One of the most fascinating features of Geosnap is its ablilty to harmonize data within geographical boundaries that have changed shape over time. A common example of this is US census boundary changes that can occur from one census period to the next. These boundary changes make it difficult to accurately compare census areas over time, which makes harmonization an incredibly useful tool. Geosnap has two ways of harmonizing boundaries: areal interpolation and dasymetric interpolation.

With the areal interpolation method, Geosnap uses the area of overlap between polygons of different time periods to create a weighted sum of the overlapped area. The defining factor of this approach compared to dasymetric interpolation is that it **assumes a constant density for each polygon**. In the case of census areas, this would mean that the population is evenly spread throughout the census area. Therefore, this method is best used on small, homogenous polygons (urban areas).

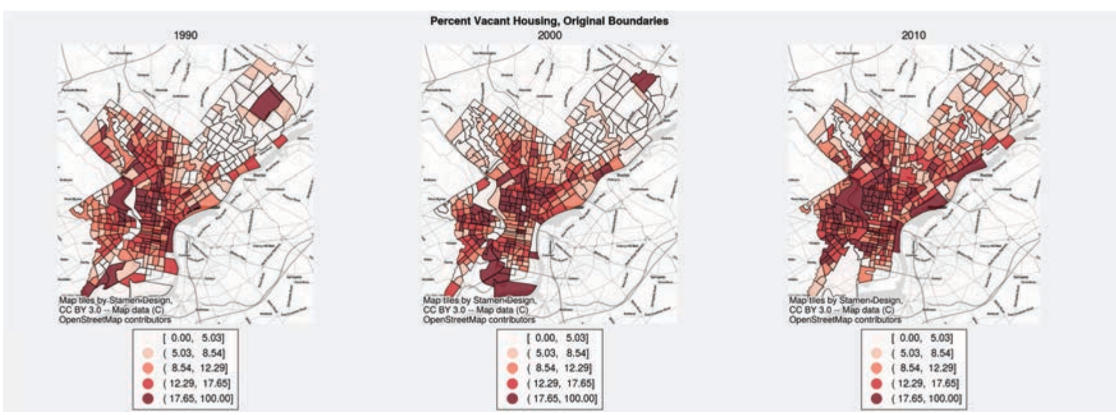
As with many spatial analysis procedures, we must set a reasonable coordinate reference system for the area of focus. In this case, we will use European Petroleum Survery Group (EPSG) 2272.

[9]:
```
phl.gdf = phl.gdf.to_crs(2272)
```

To show the Philadelphia census tract boundaries for each time period before harmonization, we will create a time series showing choropleth maps representing a variable for the available census years. The first argument for this command is the name of the column of interest written as a string. cmap refers to the map color, which can be assigned to a theme such as "Reds". dpi sets the resolution of the plot, edgecolor creates and colors boundary lines for each census tract polygon. alpha sets the opacity of the choropleth over the basemap, on a scale from 0-1. figsize sets the scale of the plot using inch units (w,h).

[10]:
```
phl.plot_timeseries(
    "p_vacant_housing_units", title="Percent Vacant Housing, Original␣
 ↪Boundaries",
    cmap='Reds',
    dpi=200,
    edgecolor = 'black',
    alpha = 0.7,
    figsize=(14,5)
)
```

[10]:
```
SubplotsContainer([CartesianAxesSubplot(0.0537579,0.260667;0.225818x0.690944),
CartesianAxesSubplot(0.387085,0.260667;0.225829x0.690944),
CartesianAxesSubplot(0.722026,0.260667;0.222615x0.690944)])
```



The output displays three plots showing percent vacant housing per census tract in Philadelphia using each year's respective census demarcation. You can see that from 1990 to 2010, as the population grew, census tracts were subdivided in some cases, creating new boundaries and therefore hard-to-track demographic changes.

In order to show change more accurately, we will harmonize the data for all three years based on the 2010 boundaries. It is important to note the intensive (the magnitude of which is independent of the size of the system aka mean, median, temperature, other ratios, etc) and extensive (add to or subtract from the existing system aka population, counts, etc) variables. weights_method assigns the interpolation type, and target_year sets the desired set of polygons for comparison across time.

[11]:
```
phl_harm10 = phl.harmonize(
    intensive_variables=["p_vacant_housing_units"],
    extensive_variables=["n_total_pop"],
    weights_method="area",
    target_year=2010,
)
```

```
Converting 2 time periods:   0%|          | 0/2 [00:00<?, ?it/s]
```

```
Harmonizing 1990
```

```
C:\Miniconda3\envs\gus5031\lib\site-packages\tobler\util\util.py:28:
UserWarning: nan values in variable: p_vacant_housing_units, replacing with 0
  warn(f"nan values in variable: {column}, replacing with 0")
```

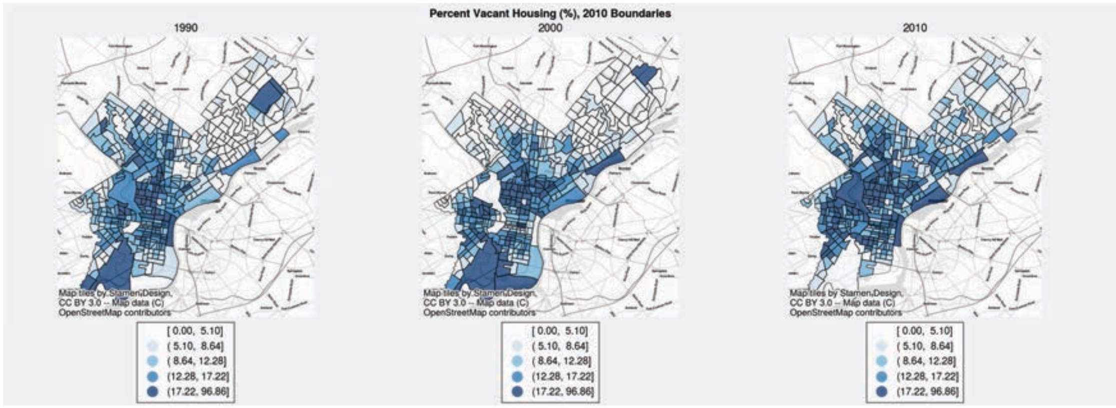
```
Harmonizing 2000
```

```
C:\Miniconda3\envs\gus5031\lib\site-packages\tobler\util\util.py:28:
UserWarning: nan values in variable: p_vacant_housing_units, replacing with 0
  warn(f"nan values in variable: {column}, replacing with 0")
```

Plot the resulting maps:

[12]:
```
phl_harm10.plot_timeseries(
    "p_vacant_housing_units", title="Percent Vacant Housing (%), 2010␣
↪Boundaries",
    dpi=200,
    edgecolor = 'black',
    alpha = 0.7,
    figsize=(14,5)
)
```

[12]:
```
SubplotsContainer([CartesianAxesSubplot(0.053474,0.260667;0.226385x0.690944),
CartesianAxesSubplot(0.386807,0.260667;0.226385x0.690944),
CartesianAxesSubplot(0.721674,0.260667;0.223319x0.690944)])
```


Percent Vacant Housing (%), 2010 Boundaries
1990
2000
2010
Map tiles by Stamen Design, CC BY 3.0 -- Map data (C) OpenStreetMap contributors
[0.00, 5.10]
(5.10, 8.64]
(8.64, 12.28]
(12.28, 17.22]
(17.22, 96.86]

1 Part II: Modelling Neighborhood & Neighborhood Types

Though the importance of the neighborhood and its centrality to urban life and spatial anaylsis is nearly indisputable, defining “neighborhood” is tricky for many reasons. They’re wildly complex, different sizes and shapes, and they contain disparate and incomperable objects, with interactions and networks between them that increase exponentially (Claude Fisher). What’s more, a countless number of parties and stakeholders, often with opposing interests and yielding unequal degrees of power, are involved in the process. In Philadelphia, given the city’s age, size, and demographics, these challenges can be both more formidable and more urgent. A 2019 Philadelphia Department of Public Health report, for example, outlined the difficulty the department and its institutional partners had in addressing the health needs of city residents because of the different ways other entities defined and characterized neighborhoods; just months later covid-19 proved their worst-case scenarios were grave underestimates. Researchers faced with these challenges need better tools with which to analyze and make sense of neighborhood change and dynamics both temporally and spatially, as a recent co-authored article by members of the geosnap team claimed (Knaap, 121). And that’s where geosnap comes in: it provides various tools that aim to clarify and measure neighborhood dynamics. In this section, we’ll explore what it provides to analyze **geodemographics,** or the population characteristics that can be divided into geographical areas, and **regionalization,** or the process of identifying and delineating the neighborhoods using those characteristics. GEOSNAP provides both “classic” and spatial clustering algorithms in order to do both. Below, we’ll create a Community object again and use some of geosnap’s algorithms as methods on it.

In the above potion of the tutorial we used census data to create a Community object; here we’ll use the Longitudinal Tract Database (LTDB). Although the LTDB data is derived from the census, I’m using it being it is already harmonized (though this might not be as significant as it apprears), it contains data going back to 1970, and it gives us the chance to use different data.

Let’s start with our imports.

```
import os
import sys
import matplotlib.pyplot as plt
import seaborn as sns
from geosnap import Community, datasets
```

2 Modelling Neighborhood Types

The geosnap.analyze module has a cluster function which takes: 1. a list of columns/variables, 2. a clustering algorithm, and 3. a number of clusters.

It splits the dataset by decade and standardizes each variable. It then "re-pools" the time periods back together and runs an algorithm. In doing so, it shows how the distribution of each variable evolves over time. For this part of my demo, I used variables that could indicate the historical effects of redlining, with the aim of importing redlining data into geosnap for comparsion (which is possible, but beyond our scope).

Begin by creating the Community object 'phl'.

```
phl = Community.from_ltdb(county_fips='42101')
```

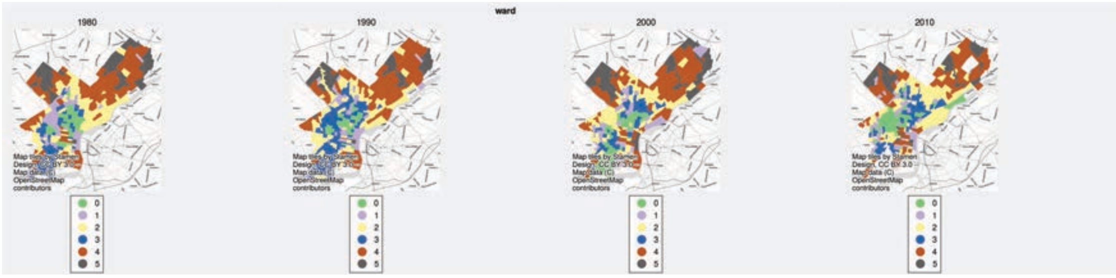
Below, we'll use the **ward** clustering method, which measures cluster proximity based on the least increase in the difference between the sum of squares in the joint cluster and the combined sum of squares in the given clusters. (Ward's method is similar to k-means, although it is thought by some to be more accurate regarding clusters of varying physical sizes and clusters "thrown about space very irregularly.") The cluster method returns a Community class with cluster labels appended as a new column so we can visualize how the clusters have evolved in space over time.

```
phl = phl.cluster(columns=[
    "median_household_income",
        "p_poverty_rate",
        "p_vacant_housing_units",
        "p_unemployment_rate",
    ],
    method="ward",
)
```

We'll plot it with the parameter "categorical=True" (as opposed to continuous) and the built-in map.

```
phl.plot_timeseries(
    "ward", years=[1980, 1990, 2000, 2010], categorical=True, cmap="Accent"
)
```

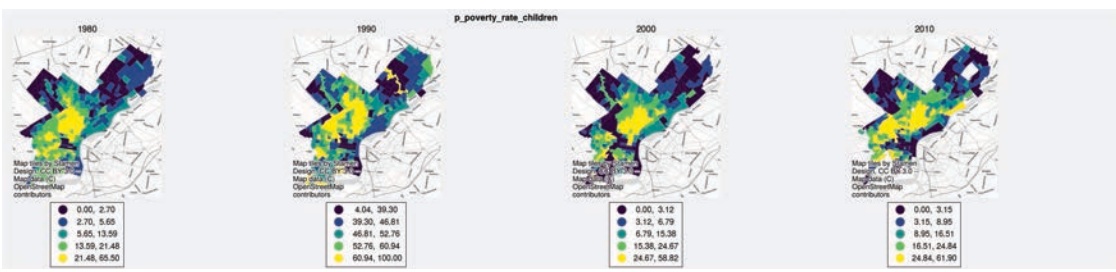
```
SubplotsContainer([CartesianAxesSubplot(0.0280197,0.351445;0.139451x0.591327),
CartesianAxesSubplot(0.301884,0.351445;0.139451x0.591327),
CartesianAxesSubplot(0.575748,0.351445;0.139451x0.591327),
CartesianAxesSubplot(0.851416,0.351445;0.137563x0.591327)])
```



Because cluster labels are appended to the input database we can compute and visualize statistics by cluster, even with variables we didn't use to derive the clusters. Below, I map children's poverty rate (viridis is a built-in map in matplotlib) and then compute the median of the ward clusters using the variable 'p_recent_immigrant_pop'.

```
phl.plot_timeseries(
    "p_poverty_rate_children", years=[1980, 1990, 2000, 2010], cmap="viridis"
)
```

```
SubplotsContainer([CartesianAxesSubplot(0.0280197,0.322186;0.139451x0.618005),
CartesianAxesSubplot(0.301884,0.322186;0.139451x0.618005),
CartesianAxesSubplot(0.575748,0.322186;0.139451x0.618005),
CartesianAxesSubplot(0.851416,0.322186;0.137563x0.618005)])
```




```
    phl.gdf.groupby("ward")
```

```
["p_recent_immigrant_pop"].median()[52]: ward
 0    0.650289
 1    1.882773
 2    2.178842
 3    1.366304
 4    1.816481
 5    1.169264
 Name: p_recent_immigrant_pop, dtype: float64
```

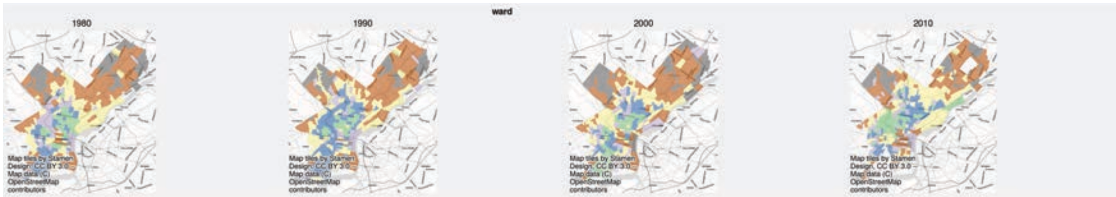
2.1 Clustering Algorithms in GEOSNAP

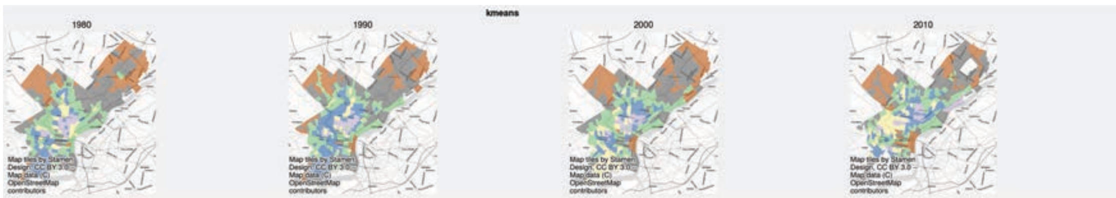
GEOSNAP provides other clustering algorithms in addition to ward that we can pass to geosnap.analyze.cluster to compare. kmeans, affinity_propagation, gaussian_mixture, spectral and hdbscan are the "classic" ones provided; the spatial algorithms are azp, kmeans_spatial, ward_spatial, max_p, skater, and spenc. (See the geosnap api for further info https://spatialucr.github.io/geosnap/api.html) We're not going to use the hdbscan because it requires installation of another package, but below I use a **for loop** to compare our data using 3 different algorithms.

```
types = ["ward", "kmeans", "affinity_propagation"]
```

```
for algo_type in types:

    phl = phl.cluster(
        columns=[
            "median_household_income",
        "p_poverty_rate",
        "p_vacant_housing_units",
        "p_unemployment_rate",
        ],
        method=algo_type,
    )
    phl.plot_timeseries(
        column=algo_type,
        years=[1980, 1990, 2000, 2010],
        cmap="Accent",
        categorical=True,
        legend=False,
        alpha=0.6
    )
```

```
C:\Users\tuh42842\AppData\Local\Continuum\anaconda3\envs\gus5031\lib\site-
packages\sklearn\cluster\_kmeans.py:786: FutureWarning: 'precompute_distances'
was deprecated in version 0.23 and will be removed in 1.0 (renaming of 0.25). It
has no effect
  warnings.warn("'precompute_distances' was deprecated in version "
C:\Users\tuh42842\AppData\Local\Continuum\anaconda3\envs\gus5031\lib\site-
packages\sklearn\cluster\_affinity_propagation.py:148: FutureWarning:
'random_state' has been introduced in 0.23. It will be set to None starting from
1.0 (renaming of 0.25) which means that results will differ at every function
call. Set 'random_state' to None to silence this warning, or to 0 to keep the
behavior of versions <0.23.
  warnings.warn(
```





2.2 EXERCISES

1. (Easier) Use a different clustering algorithm and/or different variables to compare. One way to do this is to use different variables in the for-loop Try: "gaussian_mixture" and "spectral."

2. (More challenging) Since you have already harmonized the census data, go back and model neighborhoods using a regionalization approach. Geosnap will aggregate them into demarcated neighborhoods with internal similarity. To do this, use the **cluster_spatial** method instead of the classic one we used above. If you are working with larger areas, this might be very helpful to you. Start by creating the new community from the census with the code below:

```
#phl = Community.from_census(county_fips='42101')
```

1 Part III

1.1 Simulating Nieghborhood Sociospatial Dynamics

One of the aspects of geosnap I found exciting is that it allows the user to simulate future neighborhood change. It uses a **spatially-conditioned Markov transition model** which measures change by determining what a certain condition will become based on the distribution of prior transitions between what it was and what it became. Using neighborhood types, geosnap can predict what type a neighborhood is likely to be in the future, provided we know what type a neighborhood was as well as the types of its surrounding neighborhoods.

First, import modules and built in datasets.

```
from geosnap import datasets, Community
import matplotlib.pyplot as plt
```

1.2 Cluster Model

As we did above, we start by developing a cluster model of neighborhood types. I chose as variables proven risk factors for severe or terminal cases of covid and/or common characteristics of persons who have had severe or fatal cases of covid using a number of peer-reviewed secondary sources from the past few months. My aim was to get a sense of how so many Philadephians' vulnerability to and death from covid has accumulated over time as well as to be able to compare actual covid maps with geosnap's simulations. As I mentioned above, the city's Department of Public Health struggled to capture and analyze neighborhood data, and I can't help but think that had we been able to model this better in the past, the city could have targeted populations most at risk for the severe effects of highly communicable infectious diseases. We know this is not the last one we'll experience in our lifetimes; perhaps we can find a way to mitigate its effects before the next one arrives. (Obviously, this is not intended to be "scientific" in any real sense, not least because I had to make decisions about what variables and indicators were in the LTDB that approximated researchers' findings.)

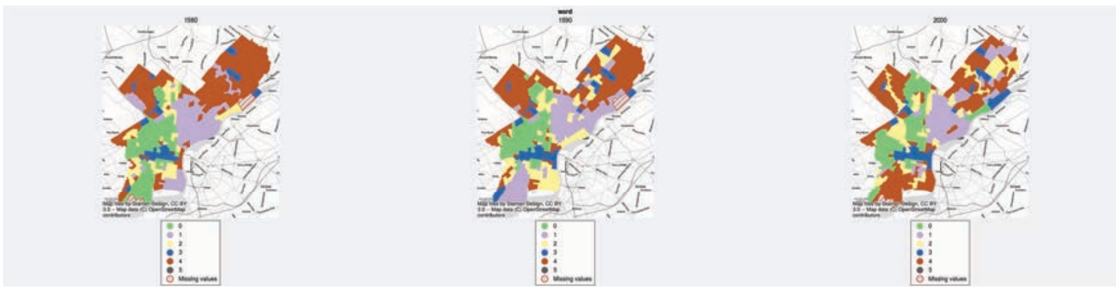
Because we're using the LTDB again, we don't have as many variable options as in the census, but we do have an extra historical decade, which will be important for our simulation below.

[ ]:
```
VARS = ['p_black_over_60', 'p_asian_over_60',␣
↪'p_housing_units_multiunit_structures',␣
↪'p_employed_manufacturing','p_disabled']
```

```
phl_clust = Community.from_ltdb(county_fips='42101', years = [1980, 1990, 2000])
```

```
phl_clust = phl_clust.cluster(method='ward', k=6, columns=VARS)
phl_clust.plot_timeseries('ward', categorical=True, figsize=(24,6))
```

```
SubplotsContainer([CartesianAxesSubplot(0.0885011,0.278056;0.156331x0.68162),
CartesianAxesSubplot(0.421834,0.278056;0.156331x0.68162),
CartesianAxesSubplot(0.755168,0.278056;0.156331x0.68162)])
```



2 Transition Model¶

Now that we have the cluster model for the neighborhood types, we can create the Markov transition model which shows which neighborhoods are likely to transition into different neighborhood types. This is crucial for our simulation.

phl_clust.plot_transition_matrix(‘ward’, n_rows=2, n_cols=4, figsize=(16,8))

3 Simulating Neighborhood Change¶

Now let’s simulate the changes. We’ll use geosnap’s predict method, which takes a cluster model name and simulates it forward by the number of time steps we specify. It returns a new Community. We’ll use queen contiguity (which, as we learned last week, defines neighbors as spatial units with a common edge or a common vertices) and 4 time stamps.

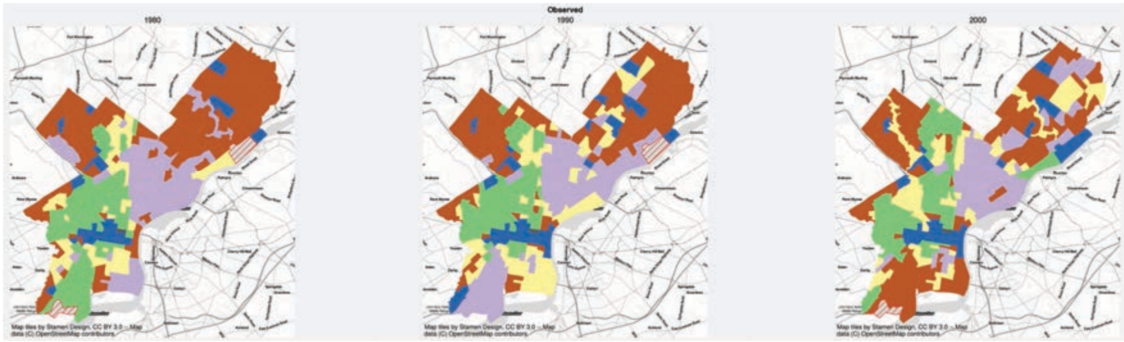
```
predicted = phl_clust.simulate('ward', base_year=2000, w_type='queen',␣
↪time_steps=4)
```

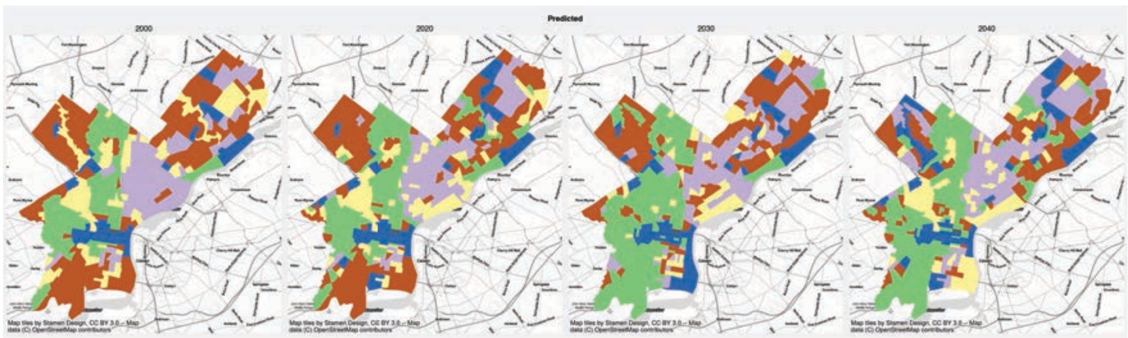
```
C:\Users\tuh42842\AppData\Local\Continuum\anaconda3\envs\gus5031\lib\site-
packages\geopandas\geodataframe.py:1322: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
```

```
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  super(GeoDataFrame, self).__setitem__(key, value)
C:\Users\tuh42842\AppData\Local\Continuum\anaconda3\envs\gus5031\lib\site-
packages\geopandas\geodataframe.py:1322: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  super(GeoDataFrame, self).__setitem__(key, value)
C:\Users\tuh42842\AppData\Local\Continuum\anaconda3\envs\gus5031\lib\site-
packages\geopandas\geodataframe.py:1322: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  super(GeoDataFrame, self).__setitem__(key, value)
C:\Users\tuh42842\AppData\Local\Continuum\anaconda3\envs\gus5031\lib\site-
packages\geopandas\geodataframe.py:1322: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  super(GeoDataFrame, self).__setitem__(key, value)
```

```python
phl_clust.plot_timeseries('ward',categorical=True,legend=False,␣
↪figsize=(20,6), title='Observed')
predicted.gdf = predicted.gdf[predicted.gdf.year!=2010] # the geosnap people␣
↪say not to use tthe base year you used above here
predicted.plot_timeseries('ward', categorical=True,␣
↪legend=False,figsize=(20,6), title='Predicted')
```

```
SubplotsContainer([CartesianAxesSubplot(0.0230417,0.0719981;0.236687x0.859985),
CartesianAxesSubplot(0.268757,0.0719981;0.236687x0.859985),
CartesianAxesSubplot(0.514472,0.0719981;0.236687x0.859985),
CartesianAxesSubplot(0.760188,0.0719981;0.236687x0.859985)])
```





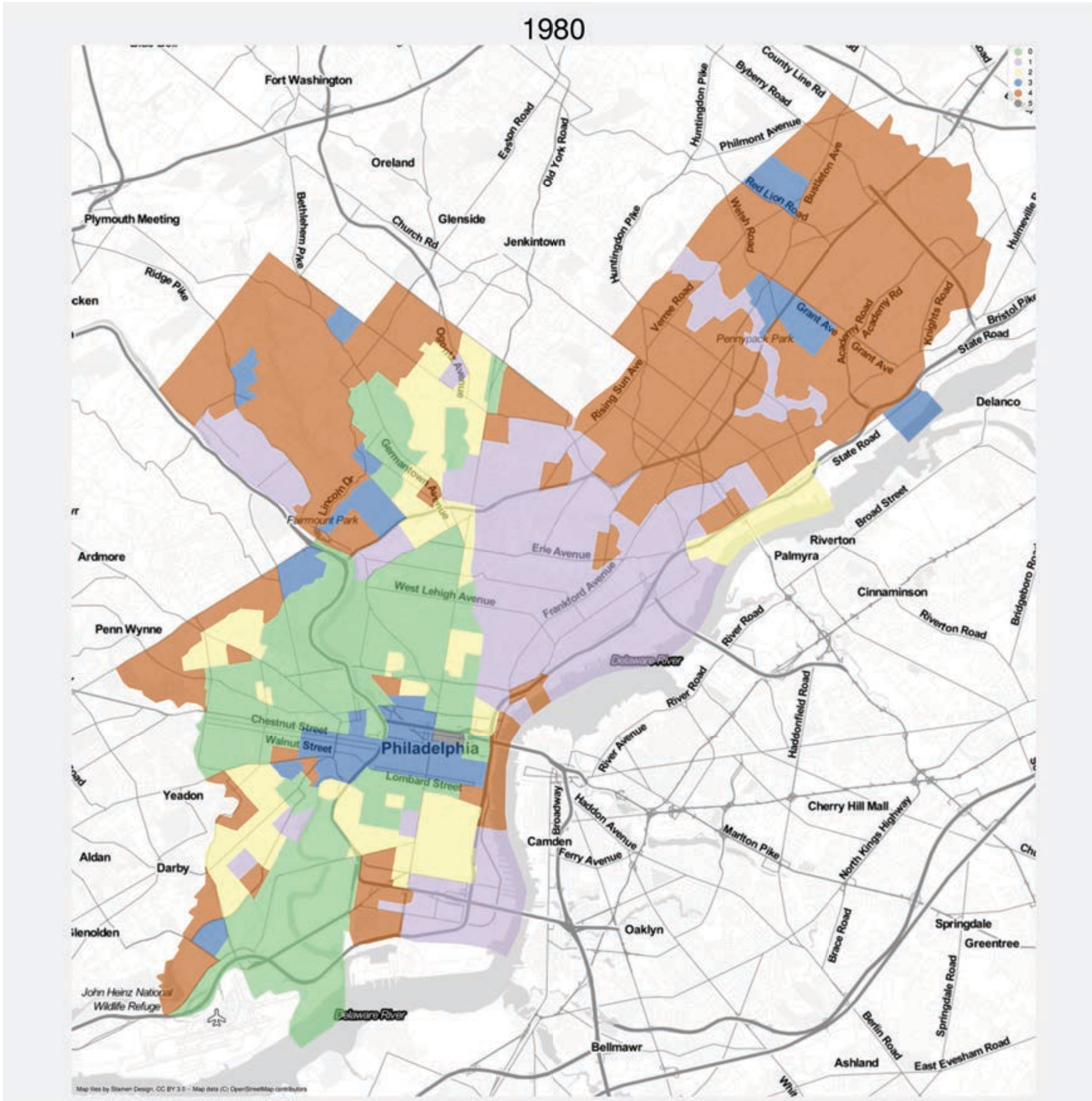
Now, I could conceivably compare the 2020 simulation above with actual covid maps.

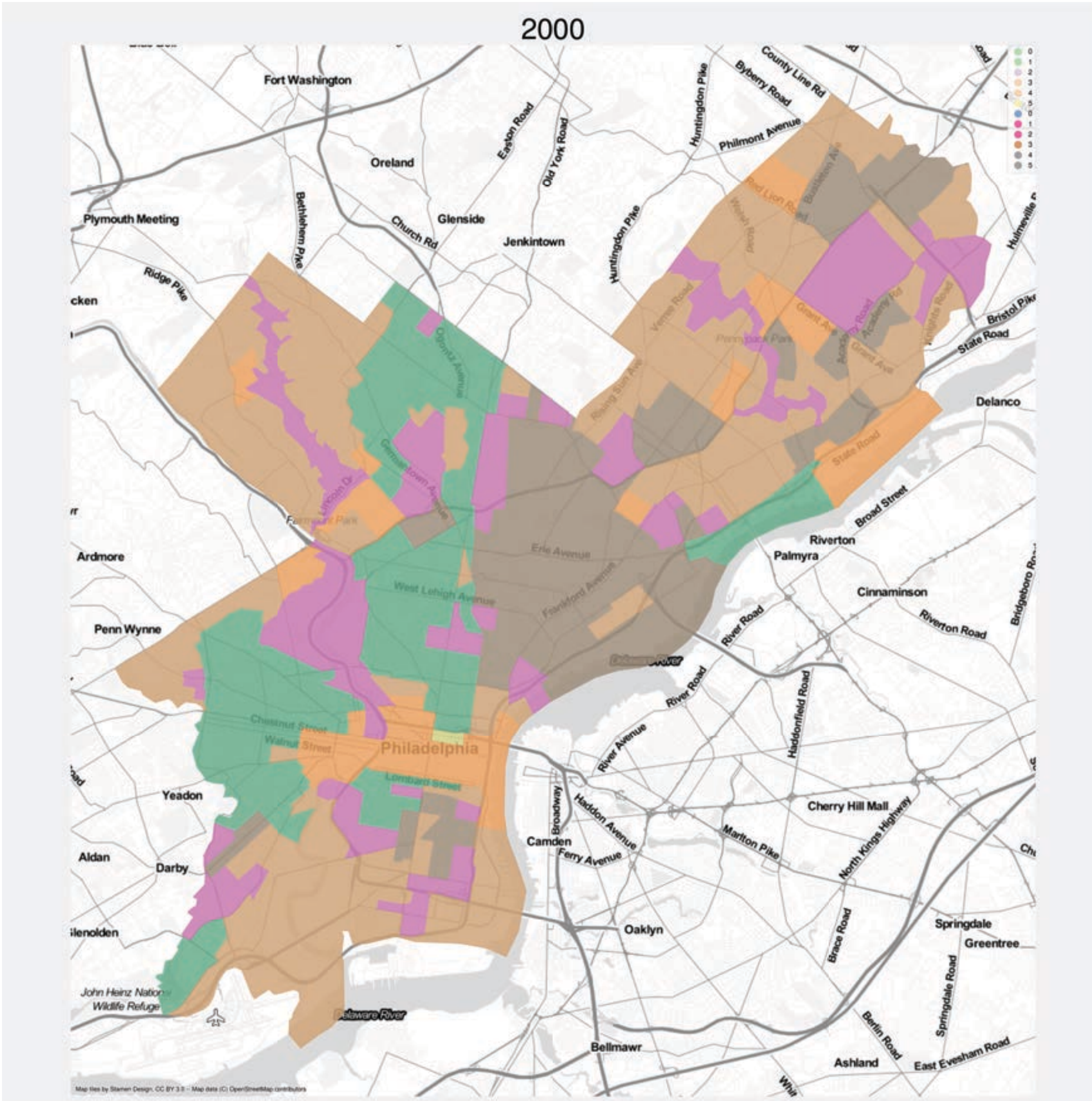
3.1 Lastly, let's use geosnap's animate method to put it in a gif.

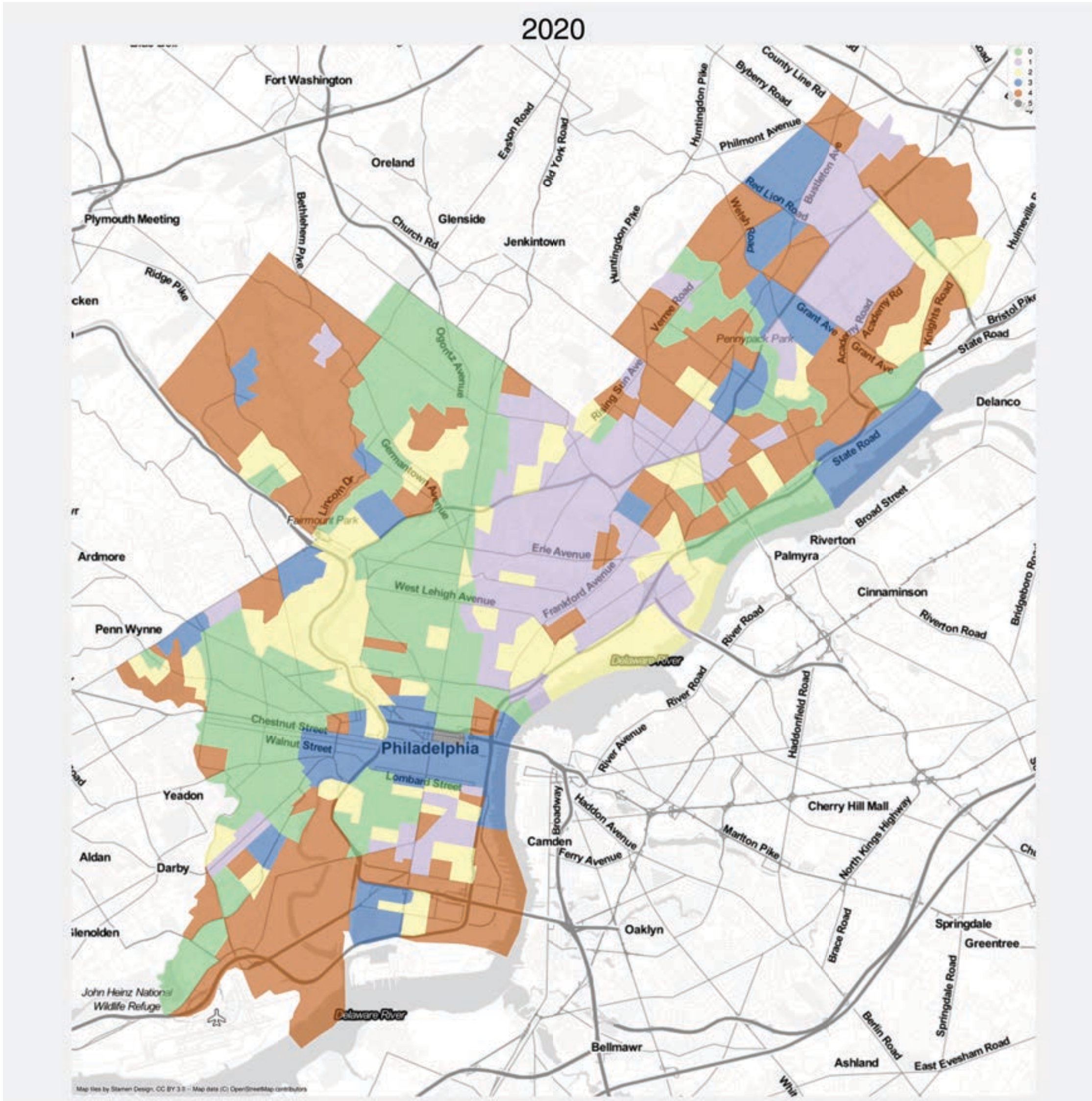
```
phl_clust.gdf = phl_clust.gdf[phl_clust.gdf.year!= 1990]
combined_sequence = Community.from_geodataframes([phl_clust.gdf, predicted.gdf])
```

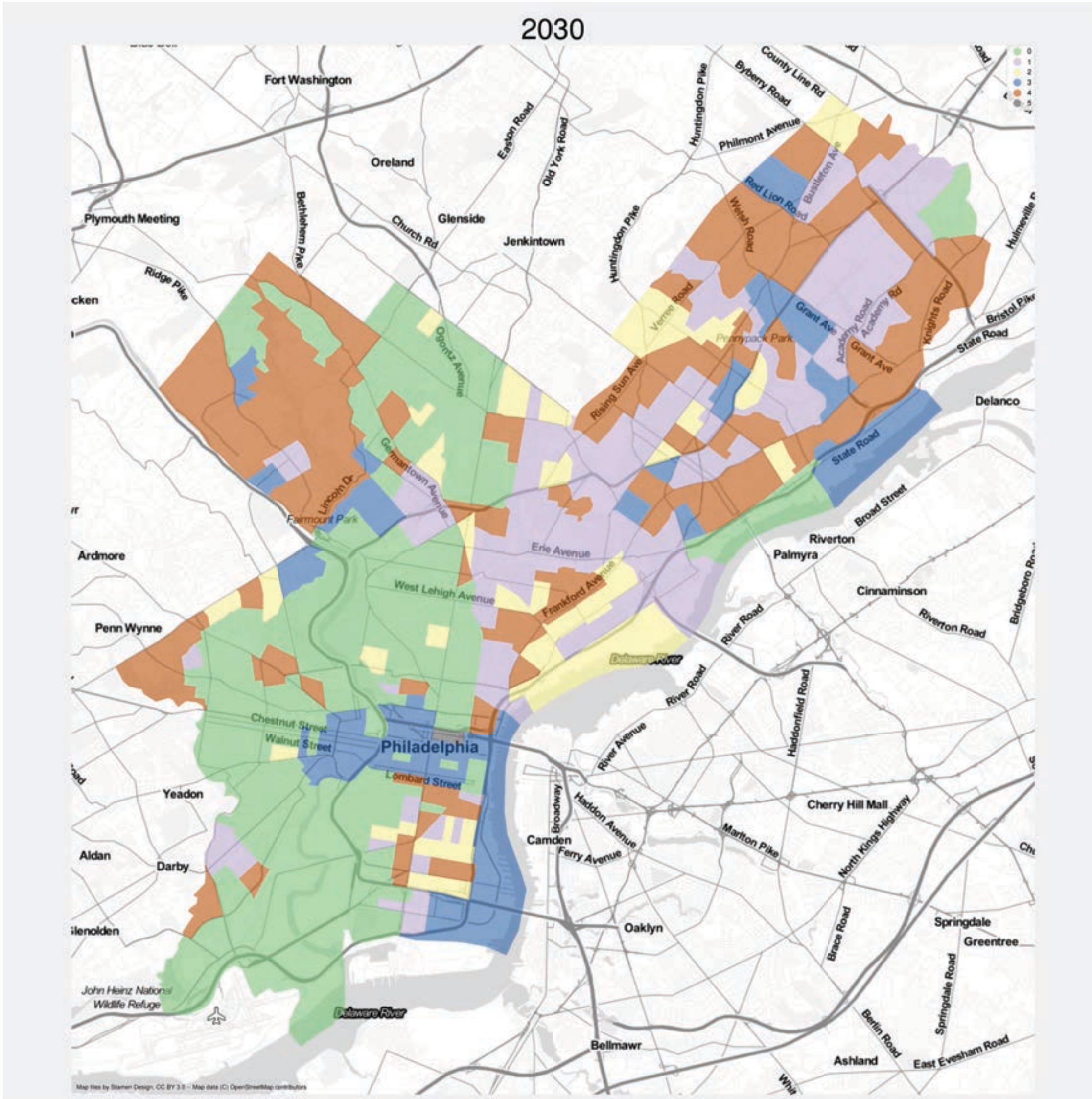
In the line below, the filename should be a path to where you willl store the gif file. (Note that the execution of this will

```
combined_sequence.animate_timeseries('ward', categorical = True, filename='C://
↪geosnap-master/examples/images/phl_sim.gif', dpi=200)
```


1980
0
1
2
3
4
5
Fort Washington
Oreland
Plymouth Meeting
Glenside
Jenkintown
Easton Road
Old York Road
Huntingdon Pike
Church Rd
Bethlehem Pike
Ridge Pike
County Line Rd
Byberry Road
Philmont Avenue
Red Lion Road
Bustleton Ave
Verree Road
Pennypack Park
Grant Ave
Academy Road
Knights Road
Rising Sun Ave
Bristol Pike
State Road
Delanco
Lincoln Dr
Germantown Avenue
Fairmount Park
Ardmore
Erie Avenue
Riverton
Palmyra
Broad Street
West Lehigh Avenue
Frankford Avenue
Cinnaminson
Riverton Road
Penn Wynne
Delaware River
River Road
Haddonfield Road
Chestnut Street
Walnut Street
Philadelphia
Lombard Street
Yeadon
River Avenue
Cherry Hill Mall
Haddon Avenue
Camden
Broadway
Ferry Avenue
Marlton Pike
North Kings Highway
Aldan
Darby
Oaklyn
Springdale
Greentree
Brace Road
John Heinz National Wildlife Refuge
Springdale Road
Bellmawr
Berlin Road
Ashland
East Evesham Road

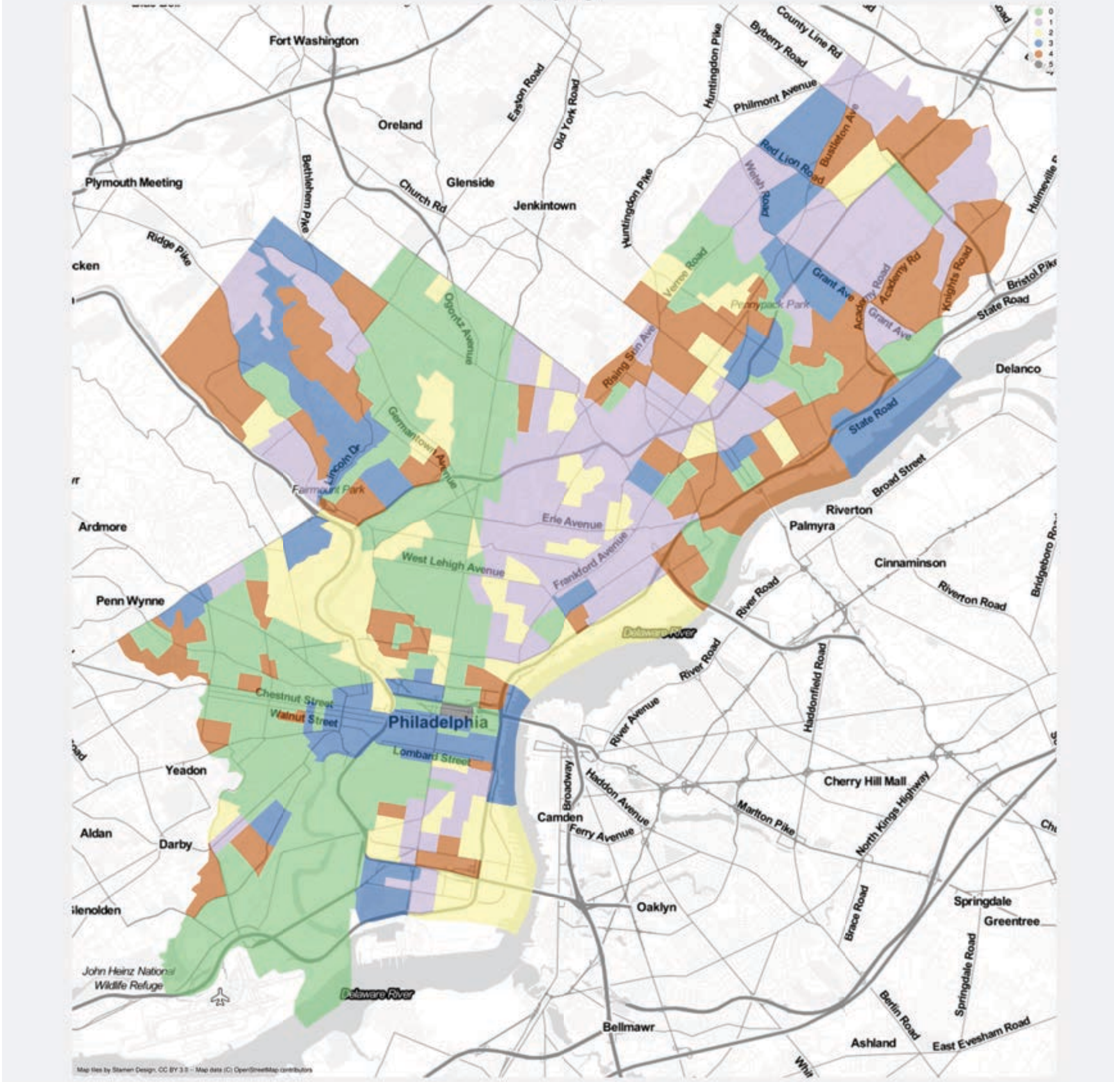

2000
Fort Washington
Oreland
Glenside
Jenkintown
Plymouth Meeting
Ardmore
Penn Wynne
Yeadon
Aldan
Darby
Philadelphia
Camden
Oaklyn
Bellmawr
Ashland
Springdale
Greentree
Cherry Hill Mall
Cinnaminson
Riverton
Palmyra
Delanco
Bethlehem Pike
Church Rd
Easton Road
Old York Road
Huntingdon Pike
Philmont Avenue
County Line Rd
Byberry Road
Ridge Pike
State Road
Broad Street
Riverton Road
River Road
Haddonfield Road
Marlton Pike
North Kings Highway
Brace Road
Berlin Road
Springdale Road
East Evesham Road
Haddon Avenue
Ferry Avenue
Broadway
Erie Avenue
West Lehigh Avenue
Frankford Avenue
Chestnut Street
Walnut Street
Lombard Street
Delaware River
John Heinz National Wildlife Refuge

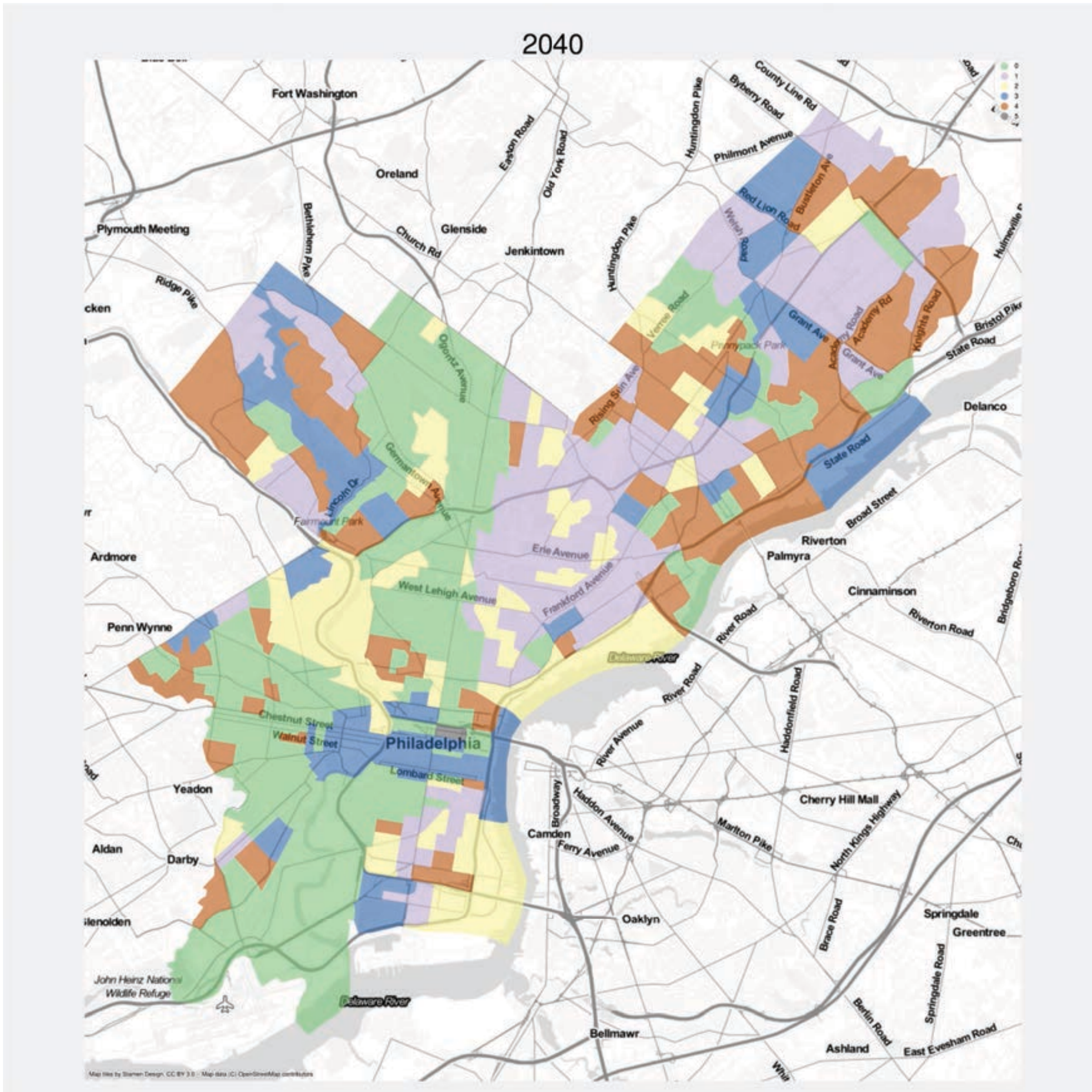

2020
Fort Washington
Oreland
Plymouth Meeting
Glenside
Jenkintown
Church Rd
Bethlehem Pike
Easton Road
Old York Road
Huntingdon Pike
Ridge Pike
Ardmore
Penn Wynne
Yeadon
Aldan
Darby
John Heinz National Wildlife Refuge
Philadelphia
Chestnut Street
Walnut Street
Lombard Street
West Lehigh Avenue
Erie Avenue
Frankford Avenue
Germantown Avenue
Ogontz Avenue
Lincoln Dr
Fairmount Park
Rising Sun Ave
Verree Road
Pennypack Park
Grant Ave
Academy Road
Academy Rd
Knights Road
Red Lion Road
Bustleton Ave
Philmont Avenue
Byberry Road
County Line Rd
State Road
Bristol Pike
Delanco
Riverton
Palmyra
Broad Street
Cinnaminson
Riverton Road
River Road
Delaware River
Haddonfield Road
Cherry Hill Mall
Marlton Pike
North Kings Highway
Haddon Avenue
Ferry Avenue
Camden
Broadway
Oaklyn
Bellmawr
Springdale
Greentree
Springdale Road
Berlin Road
Ashland
East Evesham Road
Brace Road
River Avenue



2030
Fort Washington
Oreland
Glenside
Jenkintown
Plymouth Meeting
Ardmore
Penn Wynne
Yeadon
Aldan
Darby
Philadelphia
Camden
Oaklyn
Delanco
Riverton
Palmyra
Cinnaminson
Cherry Hill Mall
Springdale
Greentree
Bellmawr
Ashland

2040





3.2 Exercise

1. Choose different variables and/or a different dataset and/or different basemap or color scheme to create your own animation.

1 WORKS CITED

An, Li, and Stephen Crook. "Spatiotemporal Analysis." *International Encyclopedia of Geography*, 1–13. American Cancer Society, 2017. https://doi.org/10.1002/9781118786352.wbieg0635.

Carethers, J. M. "Insights into Disparities Observed with COVID-19." *Journal of Internal Medicine* 289, no. 4 (2021): 463–73. https://doi.org/10.1111/joim.13199.

Fischer, Claude S. "Toward a Subcultural Theory of Urbanism." *American Journal of Sociology* 80, no. 6 (1975): 1319–41.

Gao, Ya-dong, Mei Ding, Xiang Dong, Jin-jin Zhang, Ahmet Kursat Azkur, Dilek Azkur, Hui Gan, et al. "Risk Factors for Severe and Critically Ill COVID-19 Patients: A Review." *Allergy* 76, no. 2 (2021): 428–55. https://doi.org/10.1111/all.14657.

Gerwen, Maaike, Mathilda Alsen, Christine Little, Joshua Barlow, Eric Genden, Leonard Naymagon, and Douglas Tremblay. "Risk Factors and Outcomes of COVID-19 in New York City; a Retrospective Cohort Study." *Journal of Medical Virology* 93, no. 2 (February 2021): 907–15. https://doi.org/10.1002/jmv.26337.

Knaap, Elijah, Levi Wolf, Sergio Rey, Wei Kang, and Su Han. "The Dynamics of Urban Neighborhoods: A Survey of Approaches for Modeling Socio-Spatial Structure." *Journal of GIS & Quantitative Geography* (2019). https://doi.org/10.31235/osf.io/3frcz.

Knaap, Elijah, Wei Kang, Sergio Rey, Levi John Wolf, Renan Xavier Cortes, and Su Han. *geosnap: The Geospatial Neighborhood Analysis Package.* 2019. doi = {10.5281/ZENODO.3526163}.

Logan, John R., Zengwang Xu, and Brian Stults. "Interpolating US Decennial Census Tract Data from as Early as 1970 to 2010: A Longitudinal Tract Database." *Professional Geographer* (2012) 66,3: 412-420.

Mitchell, Andy. *The ESRI Guide to GIS Analysis Volume 2: Spatial Measurements and Statistics.* First edition. Redlands, CA: ESRI, 2005.

Moore, Kari. "Philadelphia Department of Public Health (PDPH) 500 Cities Project: Linking Local Data for Targeted Action Among Philadelphia's Most Vulnerable Populations." Philadelphia Department of Public Health, Urban Health Collaborative, Dornsife School of Public Health, and Drexel University, 2019.

Rey, Sergio, Su Yeon Han, Wei Kang, Elijah Knaap, and Renan Xavier Cortes. "A Visual Analytics System for Space–Time Dynamics of Regional Income Distributions Utilizing Animated

Flow Maps and Rank-Based Markov Chains." *Geographical Analysis* 52, no. 4 (2020): 537–57. https://doi.org/10.1111/gean.12239.

Silver, Daniel, and Thiago H. Silva. "A Markov Model of Urban Evolution: Neighbourhood Change as a Complex Process." *PLOS ONE* 16, no. 1 (January 15, 2021). https://doi.org/10.1371/journal.pone.0245357.

Soni, Devi. "Introduction to Markov Chains." *Towards Data Science* (March 5, 2019). Medium.

Steuteville, Robert. "The Once and Future Neighborhood." *Public Square* (JAN. 29, 2019). Washington, DC: Congress for the New Urbanism.