Spatial Autocorrelation with ADU Data

December 7, 2020

Table of Contents

1 Group Assignment #4: Spatial Analysis

- This assignment will focus on some of the more advanced spatial analyses techniques learned in class and will utilize ADU permit data from the LA Data Portal.
- We will find tendencies for spatial clustering in your data by conducting a spatial autocorrelation analysis.
- Our results will include a global Moran's I statistic, followed by a local spatial autocorrelation with a moran's plot that indicates a P-value and a scatterplot with HH, HL, LH, and LL values.
- We will produce a final output in the form of a map that indicates the location of statistically significant spatial clusters.

1.1 Methodology

- In this study, we will look at ADU data from January 2017 - November 2020. Do ADU development locations have a statistical significant tendency to cluster in certain communities?
- To answer this question, we will look at the location of recorded ADU permits in the city, and compare these locations with developments nearby. We are seeking to see where spatial correlations occur based on the data. Our approach is:

1. import census block group boundaries for Los Angeles
2. import ADU data from the LA Open Data Portal
3. spatially join the two datasets
4. normalize the data to create ADUs per 1000
5. conduct global spatial autocorrelation using Moran's I
6. conduct local spatial autocorrelation using Local Indicators of Spatial Association (LISAs)

1.2 Libraries to use

[199]:
```
# to read and wrangle data
import pandas as pd

# to import data from LA Data portal
from sodapy import Socrata

# to create spatial data
import geopandas as gpd

# for basemaps
import contextily as ctx

# For spatial statistics
import esda
from esda.moran import Moran, Moran_Local

import splot
```

```
from splot.esda import moran_scatterplot, plot_moran,␣
↪lisa_cluster,plot_moran_simulation

import libpysal as lps

# Graphics
import matplotlib.pyplot as plt
import plotly.express as px
```

1.3 Block Groups

Our first task is to bring in a geography that will allow us to summarize the location of ADU permits. The smaller geography that the census block groups provides a human scale. Additionally, working with census geographies will allow for future analyses that may include census data.

- Date source:
  - Census Reporter: ACS 2018 5 year: Table B01003: Total Population in Los Angeles: Census Block Groups

[200]:
```
# read downloaded geojson file from census reporter
gdf = gpd.read_file('data/acs2018_5yr_B01003_15000US060372711003.geojson')
```

[201]:
```
gdf.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 2516 entries, 0 to 2515
Data columns (total 5 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   geoid             2516 non-null   object
 1   name              2516 non-null   object
 2   B01003001         2516 non-null   float64
 3   B01003001, Error  2516 non-null   float64
 4   geometry          2516 non-null   geometry
dtypes: float64(2), geometry(1), object(2)
memory usage: 98.4+ KB
```

[202]:
```
# trim the data to the bare minimum columns
gdf = gdf[['geoid','B01003001','geometry']]

# rename the columns
gdf.columns = ['FIPS','TotalPop','geometry']
```

[203]:
```
# last rows
gdf.tail()
```

[203]:
```
                      FIPS  TotalPop  \
2511  15000US060379800261      37.0
```

```
2512  15000US060379800281       0.0
2513  15000US060379800311    1113.0
2514  15000US060379902000       0.0
2515        16000US0644000  3959657.0

                                                geometry
2511  MULTIPOLYGON (((-118.35173 34.28034, -118.3517…
2512  MULTIPOLYGON (((-118.45246 33.94315, -118.4464…
2513  MULTIPOLYGON (((-118.29105 33.75378, -118.2905…
2514  MULTIPOLYGON (((-118.63598 34.03255, -118.6325…
2515  MULTIPOLYGON (((-118.66818 34.18987, -118.6681…
```

[204]:
```
# delete last column which is for the entire city of LA
gdf=gdf.drop(2515)
```

[205]:
```
# fix FIPS code
gdf['FIPS'] = gdf['FIPS'].str.replace('15000US','')
gdf.tail()
```

[205]:
```
              FIPS  TotalPop  \
2510  060379800241     264.0
2511  060379800261      37.0
2512  060379800281       0.0
2513  060379800311    1113.0
2514  060379902000       0.0

                                                geometry
2510  MULTIPOLYGON (((-118.51849 34.18389, -118.5184…
2511  MULTIPOLYGON (((-118.35173 34.28034, -118.3517…
2512  MULTIPOLYGON (((-118.45246 33.94315, -118.4464…
2513  MULTIPOLYGON (((-118.29105 33.75378, -118.2905…
2514  MULTIPOLYGON (((-118.63598 34.03255, -118.6325…
```

One more data cleanup: get rid of census blocks groups with less than 100 total population.

[206]:
```
# sort by total pop
gdf.sort_values(by='TotalPop').head(20)
```

[206]:
```
              FIPS  TotalPop  \
2514  060379902000       0.0
2506  060379800201       0.0
2358  060372772002       0.0
2512  060379800281       0.0
2509  060379800231       0.0
2508  060379800221       0.0
2501  060379800091       5.0
2503  060379800141      10.0
```

```
2507  060379800211      12.0
2511  060379800261      37.0
2500  060379800081      90.0
2499  060379302002     118.0
2505  060379800191     151.0
2502  060379800101     189.0
2510  060379800241     264.0
2360  060372774002     294.0
2498  060379302001     301.0
2284  060372736003     310.0
2092  060372640001     320.0
1868  060372371013     321.0

                                               geometry
2514  MULTIPOLYGON (((-118.63598 34.03255, -118.6325…
2506  MULTIPOLYGON (((-118.34412 34.21700, -118.3438…
2358  MULTIPOLYGON (((-118.38597 33.94734, -118.3859…
2512  MULTIPOLYGON (((-118.45246 33.94315, -118.4464…
2509  MULTIPOLYGON (((-118.64870 34.23120, -118.6480…
2508  MULTIPOLYGON (((-118.50266 34.30809, -118.5026…
2501  MULTIPOLYGON (((-118.33707 34.14160, -118.3361…
2503  MULTIPOLYGON (((-118.26088 33.76850, -118.2602…
2507  MULTIPOLYGON (((-118.40183 34.26509, -118.4017…
2511  MULTIPOLYGON (((-118.35173 34.28034, -118.3517…
2500  MULTIPOLYGON (((-118.50267 34.22121, -118.5015…
2499  MULTIPOLYGON (((-118.51028 34.34504, -118.5102…
2505  MULTIPOLYGON (((-118.59919 34.07436, -118.5991…
2502  MULTIPOLYGON (((-118.25165 34.08038, -118.2515…
2510  MULTIPOLYGON (((-118.51849 34.18389, -118.5184…
2360  MULTIPOLYGON (((-118.37868 33.95180, -118.3786…
2498  MULTIPOLYGON (((-118.41035 34.29197, -118.4102…
2284  MULTIPOLYGON (((-118.46583 33.99098, -118.4657…
2092  MULTIPOLYGON (((-118.49381 34.05010, -118.4938…
1868  MULTIPOLYGON (((-118.29148 33.98586, -118.2914…
```

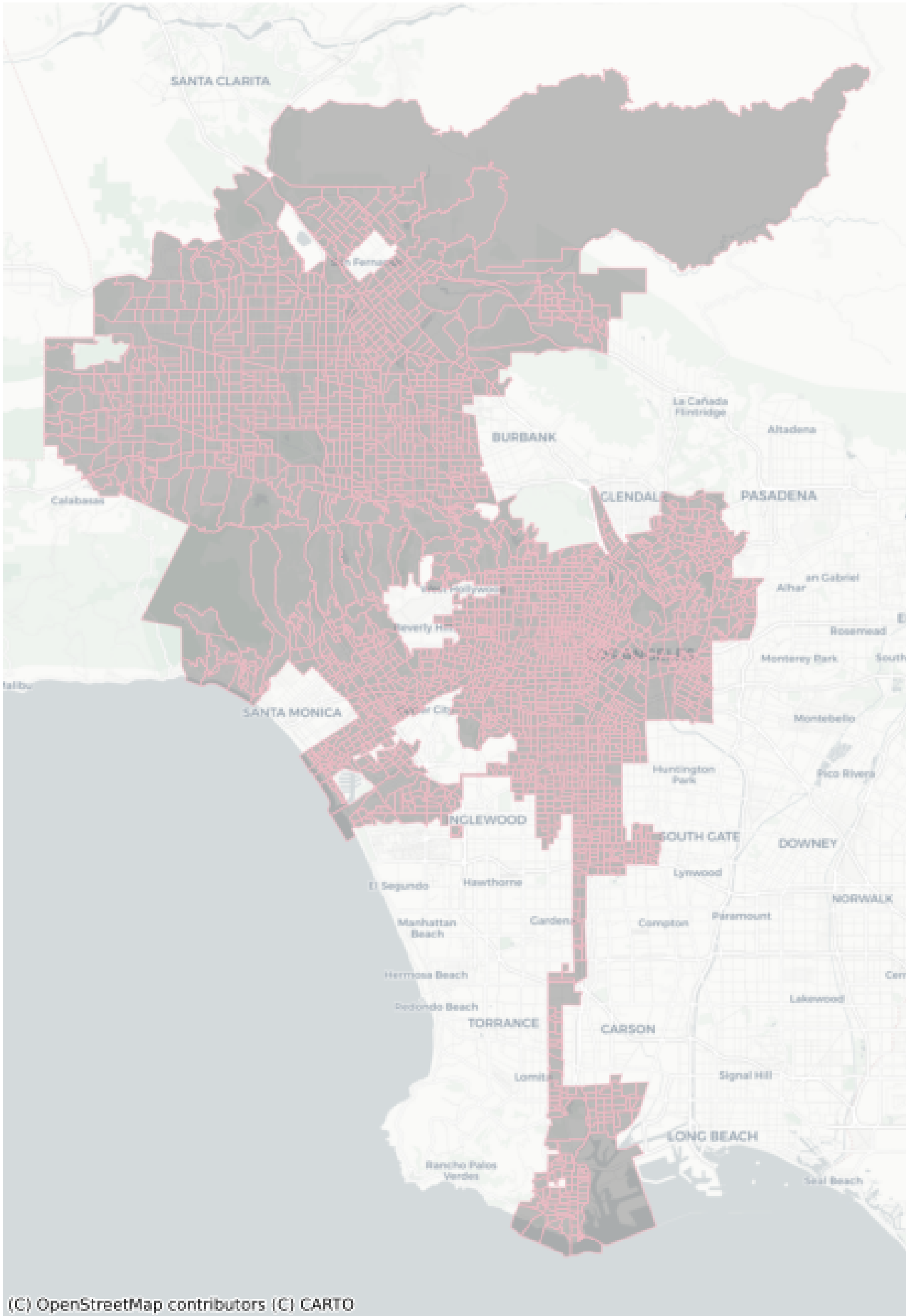
[207]:
```
# delete zero population geographies
gdf = gdf[gdf['TotalPop']>10]
```

1.4 Map the census block groups

[208]:
```
# get the layers into a web mercator projection
# reproject to web mercator
gdf = gdf.to_crs(epsg=3857)
```

[209]:
```
# plot it!
ax=gdf.plot(figsize=(12,12),
                      color='gray',
```

```
                        edgecolor='pink',
                        alpha=0.5)

# no axis
ax.axis('off')

# add a basemap
ctx.add_basemap(ax,source=ctx.providers.CartoDB.Positron)
```


SANTA CLARITA
BURBANK
La Cañada Flintridge
Altadena
PASADENA
Calabasas
Rosemead
Monterey Park
SANTA MONICA
Montebello
Huntington Park
Pico Rivera
SOUTH GATE
DOWNEY
Lynwood
El Segundo
Hawthorne
NORWALK
Manhattan Beach
Compton
Paramount
Hermosa Beach
Redondo Beach
Lakewood
TORRANCE
CARSON
Signal Hill
LONG BEACH
Rancho Palos Verdes
Seal Beach
(C) OpenStreetMap contributors (C) CARTO

1.5 Get ADU Data from LA Open Data Portal

Next, we acquire the data using the socrata API. Use the socrata documentation to grab the code syntax for our ADU data. - https://data.lacity.org/A-Prosperous-City/ADU-info/hyem-e7yr

[210]:
```
# connect to the data portal
client = Socrata("data.lacity.org", None)

results = client.get("hyem-e7yr",
                     limit=50000,
                     where = "issue_date between '2017-01-01T00:00:00' and␣
 ↪'2020-11-30T00:00:00'",
                     order='issue_date desc')

# Convert to pandas DataFrame
adus = pd.DataFrame.from_records(results)
```

```
WARNING:root:Requests made without an app_token will be subject to strict
throttling limits.
```

[211]:
```
adus.shape
```

[211]:
```
(2153, 53)
```

1.5.1 Convert data to a geodataframe

Geopandas allows us to convert different types of data into a spatial format. - https://geopandas.org/gallery/create_geopandas_from_pandas.html

[212]:
```
# convert pandas dataframe to geodataframe
adu = gpd.read_file('https://data.lacity.org/resource/hyem-e7yr.geojson')

adu.head()
```

[212]:
```
  assessor_parcel zip_code location_1_address  \
0             034    91367                None
1             046    91316                None
2             024    90025                None
3             014    90034                None
4             027    91436

                                    work_description  \
0  NEW FIRE SPRINKLER SYSTEM FOR ADU  PER NFPA 13…
1  NFPA13D FOR ADU.  EXISTING 1'' DOMESTIC WATER …
2  NFPA 13D SYSTEM . 1" DOMESTIC METER SEVRVES TH…
3  NEW FIRE SPRINKLER SYSTEM FOR PER NFPA 13D FOR…
4  New fire sprinkler system for ADU per NFPA-13D…
```

```
  :@computed_region_2dna_qi2s applicant_address_3  \
0                        None          ARLETA, CA
1                        None      SUN VALLEY, CA
2                        None                None
3                        None  WOODLAND HILLS, CA
4                          62        WEST HILLS,CA

  floor_area_l_a_zoning_code_definition address_fraction_end project_number  \
0                                  None                 None           None
1                                  None                 None           None
2                                  None                 None           None
3                                  None                 None           None
4                                  None                 None           None

  suffix_direction  … event_code reference_old_permit  \
0             None  …       None                 None
1             None  …       None                 None
2             None  …       None                 None
3             None  …       None                 None
4             None  …       None                 None

  applicant_relationship :@computed_region_k96s_3jcv contractor_state  \
0             Contractor                        None               CA
1             Contractor                        None               CA
2             Contractor                        None               CA
3             Contractor                        None               CA
4   Agent for Contractor                         327               CA

  license_expiration_date :@computed_region_qz3q_ghft applicant_address_2  \
0     2021-06-30T00:00:00                        None                None
1     2021-12-31T00:00:00                        None              UNIT G
2     2021-10-31T00:00:00                        None                None
3     2021-01-31T00:00:00                        None                None
4     2021-10-31T00:00:00                       19737                None

          permit_sub_type                     geometry
0  1 or 2 Family Dwelling                         None
1  1 or 2 Family Dwelling                         None
2  1 or 2 Family Dwelling                         None
3  1 or 2 Family Dwelling                         None
4  1 or 2 Family Dwelling  POINT (-118.49822 34.14598)

[5 rows x 65 columns]
```

[213]:

```
adu = adu[['issue_date','geometry']]

# print it with .sample, which gives you random rows
```

```
adu.head()
```

[213]:
```
            issue_date                     geometry
0  2020-12-03T00:00:00                         None
1  2020-10-30T00:00:00                         None
2  2020-10-27T00:00:00                         None
3  2020-09-30T00:00:00                         None
4  2020-09-18T00:00:00  POINT (-118.49822 34.14598)
```

[214]:
```
list(adu)
```

[214]:
```
['issue_date', 'geometry']
```

[215]:
```
adu.crs
```

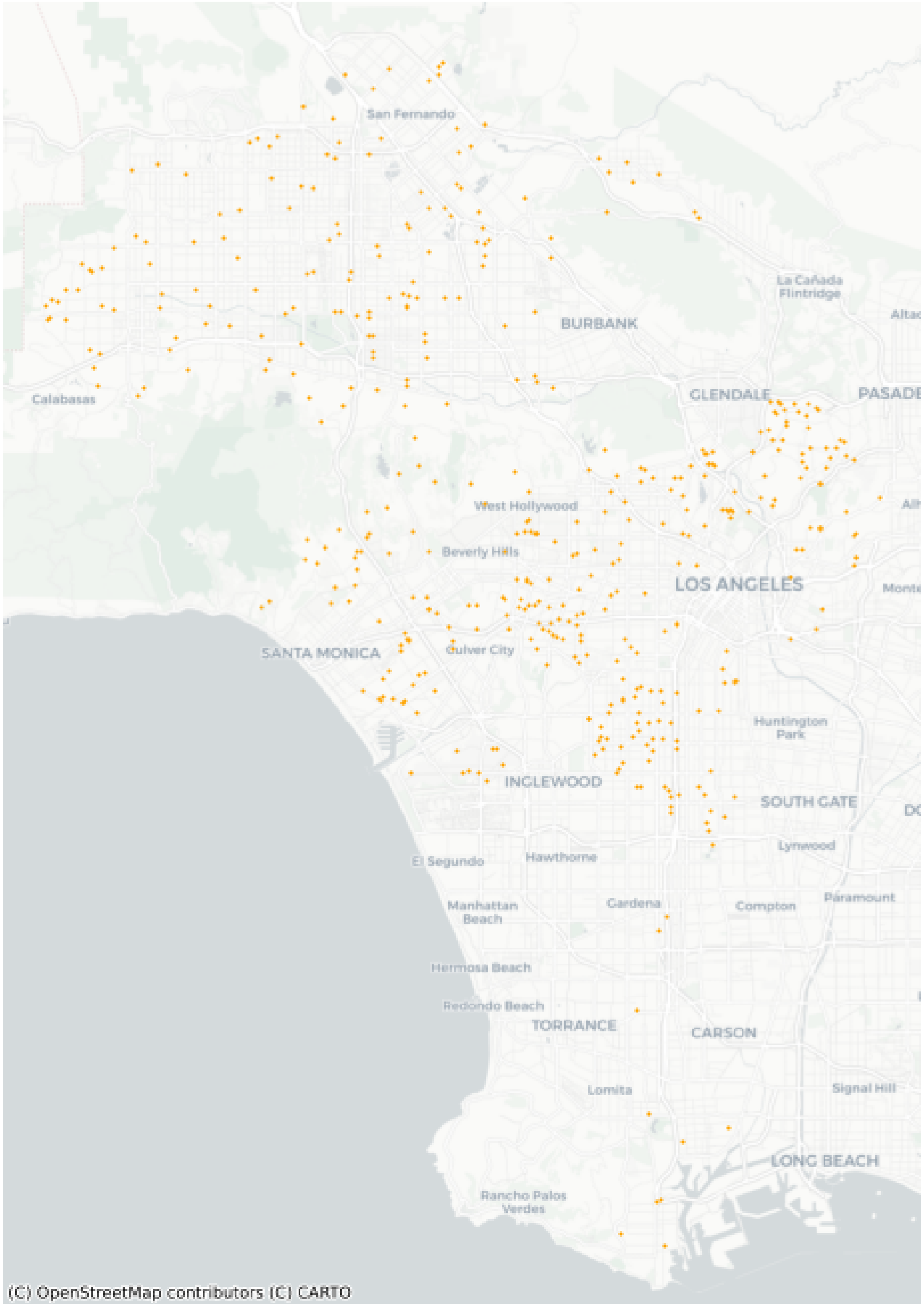
[215]:
```
<Geographic 2D CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- name: World
- bounds: (-180.0, -90.0, 180.0, 90.0)
Datum: World Geodetic System 1984
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

[216]:
```
#We can a latitude and longtitude column so that we can map it
adu['x'] = adu.geometry.x
adu['y'] = adu.geometry.y
```

[217]:
```
adu = adu.dropna()
```

[218]:
```
# get the layers into a web mercator projection
# reproject to web mercator
adu = adu.to_crs('EPSG:3857')
```

[219]:
```
# map it!
ax = adu.plot(figsize=(12,12),
                  color='orange',
                  markersize=1)

# no axis
ax.axis('off')

# add a basemap
ctx.add_basemap(ax,source=ctx.providers.CartoDB.Positron)
```


San Fernando
BURBANK
La Cañada
Flintridge
GLENDALE
PASADE
Calabasas
West Hollywood
Beverly Hills
LOS ANGELES
SANTA MONICA
Culver City
Huntington
Park
INGLEWOOD
SOUTH GATE
Lynwood
El Segundo
Hawthorne
Manhattan
Beach
Gardena
Compton
Paramount
Hermosa Beach
Redondo Beach
TORRANCE
CARSON
Lomita
Signal Hill
LONG BEACH
Rancho Palos
Verdes
(C) OpenStreetMap contributors (C) CARTO

1.6 Create a two layer map

- https://geopandas.org/mapping.html

Since we want to zoom to the extent of the ADU layer (and not the block groups), get the bounding coordinates for our axis.

[220]:
```
# get the bounding box coordinates for the ADU data
minx, miny, maxx, maxy = adu.geometry.total_bounds
print(minx)
print(maxx)
print(miny)
print(maxy)
```

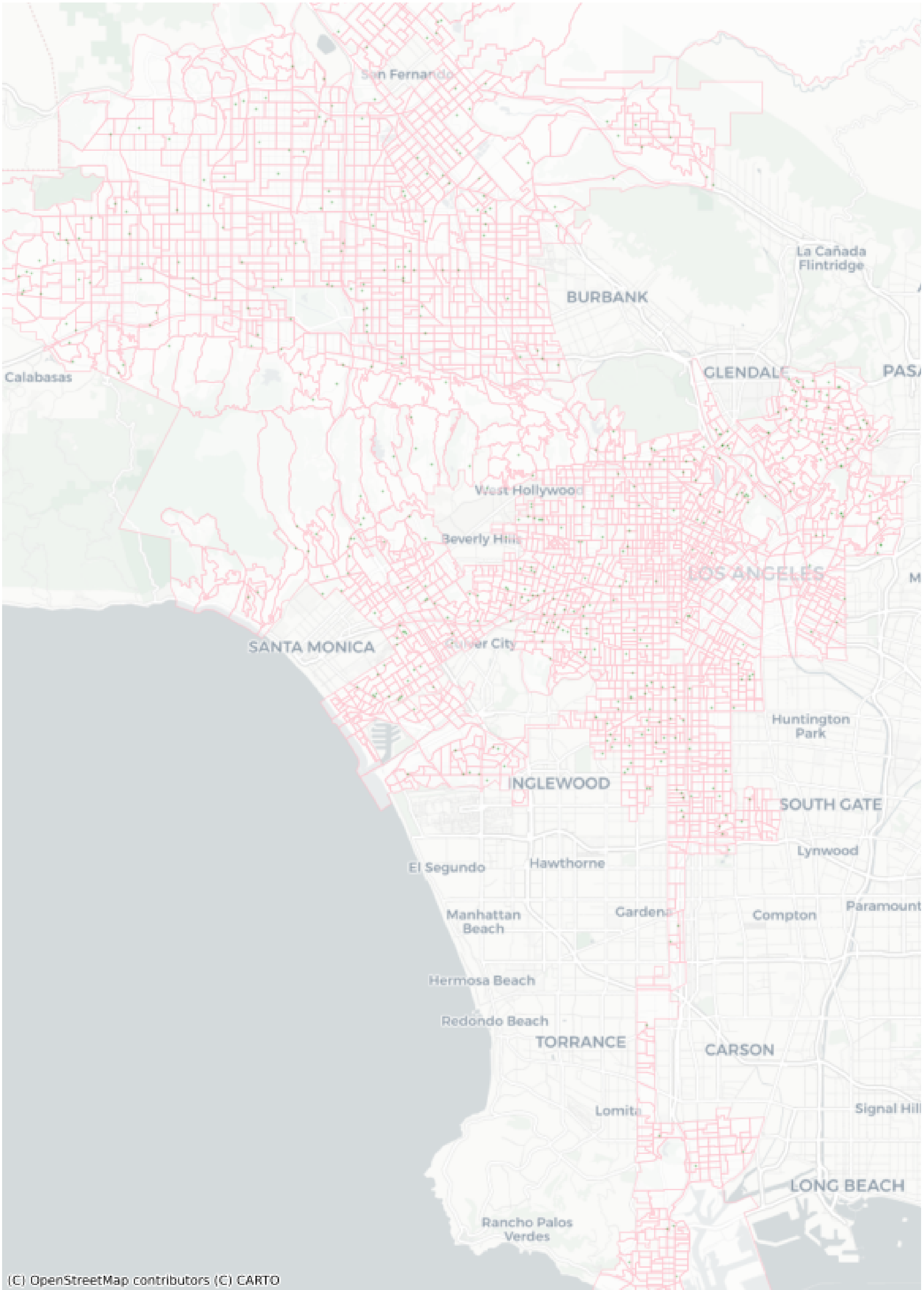
```
-13208653.141897654
-13153433.108489651
3992182.022248056
4070591.396704316
```

1.7 Subplots for multi-layered maps

For our multi-layered maps, we are taking it one step further from our previous lab using matplotlib's `subplots`. `subplots` allows the creation of multiple plots on a gridded canvas. For our map, we only need a single subplot, but we are layering multiple datasets *on top of one another* on that subplot. To specify which subplot to put the layer on, you use the `ax` argument.

[221]:
```
# set up the plot canvas with plt.subplots
fig, ax = plt.subplots(figsize=(15, 15))

# block groups
gdf.plot(ax=ax, # this puts it in the ax plot
        color='white',
        edgecolor='pink',
        alpha=0.5)

# ADUs
adu.plot(ax=ax, # this also puts it in the same ax plot
            color='green',
            markersize=1,
            alpha=0.2)

# use the bounding box coordinates to set the x and y limits
ax.set_xlim(minx - 1000, maxx + 1000) # added/substracted value is to give some␣
 ↪margin around total bounds
ax.set_ylim(miny - 1000, maxy + 1000)

# no axis
ax.axis('off')
```

```
# add a basemap
ctx.add_basemap(ax,source=ctx.providers.CartoDB.Positron)
```



1.8 The spatial join

- https://geopandas.org/mergingdata.html?highlight=spatial%20join

In a Spatial Join, two geometry objects are merged based on their spatial relationship to one another.

While the official documentation may seem confusing, consider the following as a rule of thumb. When you do a spatial join with `gpd.sjoin()`, you feed it three arguments: a left dataframe, a right dataframe, and a how statement.

- **Left dataframe**: identify the layer that you want to get information *from* to attach to the other layer
- **Right dataframe**: identify the layer you want *to* attach infomation that will come from the other layer

Once you identify your left and right dataframes, use `how="right"` to spatially join the two layers (think: "I'm sending data from the left to the right"). Note that this will result in a dataframe with the same number of rows as the RIGHT layer.

[222]:
```
# Do the spatial join
join = gpd.sjoin(gdf, adu, how='right')
join.head()
```

[222]:
```
     index_left          FIPS  TotalPop          issue_date  \
195           6  060371012102    3143.0  2020-10-28T00:00:00
503          10  060371013002    1279.0  2019-09-16T00:00:00
588          10  060371013002    1279.0  2020-01-24T00:00:00
729          14  060371014002    1540.0  2020-02-24T00:00:00
837          16  060371021031    1771.0  2020-06-09T00:00:00

                             geometry          x         y
195  POINT (-13168046.018 4063201.652) -118.29057  34.25580
503  POINT (-13165406.633 4060288.844) -118.26686  34.23417
588  POINT (-13165741.705 4060738.577) -118.26987  34.23751
729  POINT (-13169831.583 4062652.161) -118.30661  34.25172
837  POINT (-13175267.313 4058918.211) -118.35544  34.22399
```

This creates a dataframe that has every ADU record with the corresponding FIPS code.

Next, we create another dataframe that counts crime by their corresponding block group:

[223]:
```
adu_by_gdf = join.FIPS.value_counts().rename_axis('FIPS').
↪reset_index(name='adu_count')
```

[224]:
```
adu_by_gdf.head()
```

[224]:

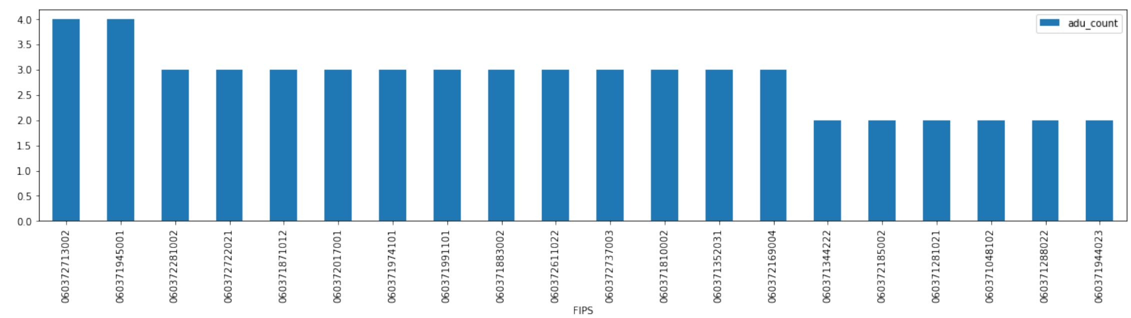
```
           FIPS  adu_count
0  060372713002          4
1  060371945001          4
2  060372281002          3
3  060372722021          3
4  060371871012          3
```

[225]:

```python
# make a bar chart of top 20 geographies
adu_by_gdf[:20].plot.bar(figsize=(20,4),
                         x='FIPS',
                         y='adu_count')
```

[225]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5cd2522070>
```



1.9 Join the value counts back to the gdf

How many people know their census block number? The bar chart is nice, but it is not informative. Without spatial awareness, the data chart does little to convey knowledge. What we want is a choropleth map to accompany it. To do so, we merge the counts back to the block group gdf.

[226]:

```python
# join the summary table back to the gdf
gdf=gdf.merge(adu_by_gdf,on='FIPS')
```

Now the block group gdf has a new column for ADU counts:

[227]:

```python
# our neighborhood table now has a count column
gdf.head()
```

[227]:

```
           FIPS  TotalPop                                           geometry  \
0  060371012102    3143.0  MULTIPOLYGON (((-13169034.646 4063225.625, -13…
1  060371013002    1279.0  MULTIPOLYGON (((-13166473.296 4061829.859, -13…
2  060371014002    1540.0  MULTIPOLYGON (((-13171587.314 4062370.826, -13…
3  060371021031    1771.0  MULTIPOLYGON (((-13176367.818 4059552.748, -13…
4  060371021072    1602.0  MULTIPOLYGON (((-13176542.701 4060290.729, -13…

   adu_count
```

```
0            1
1            2
2            1
3            1
4            1
```

1.10 ADUs per 1000 people

Rather than proceeding with an absolute count of ADUs, let's normalize it by number of people who live in the census block group.

[228]:
```
gdf['adu_per_1000'] = gdf['adu_count']/gdf['TotalPop']*1000
```

[229]:
```
gdf.sort_values(by="adu_per_1000").tail()
```

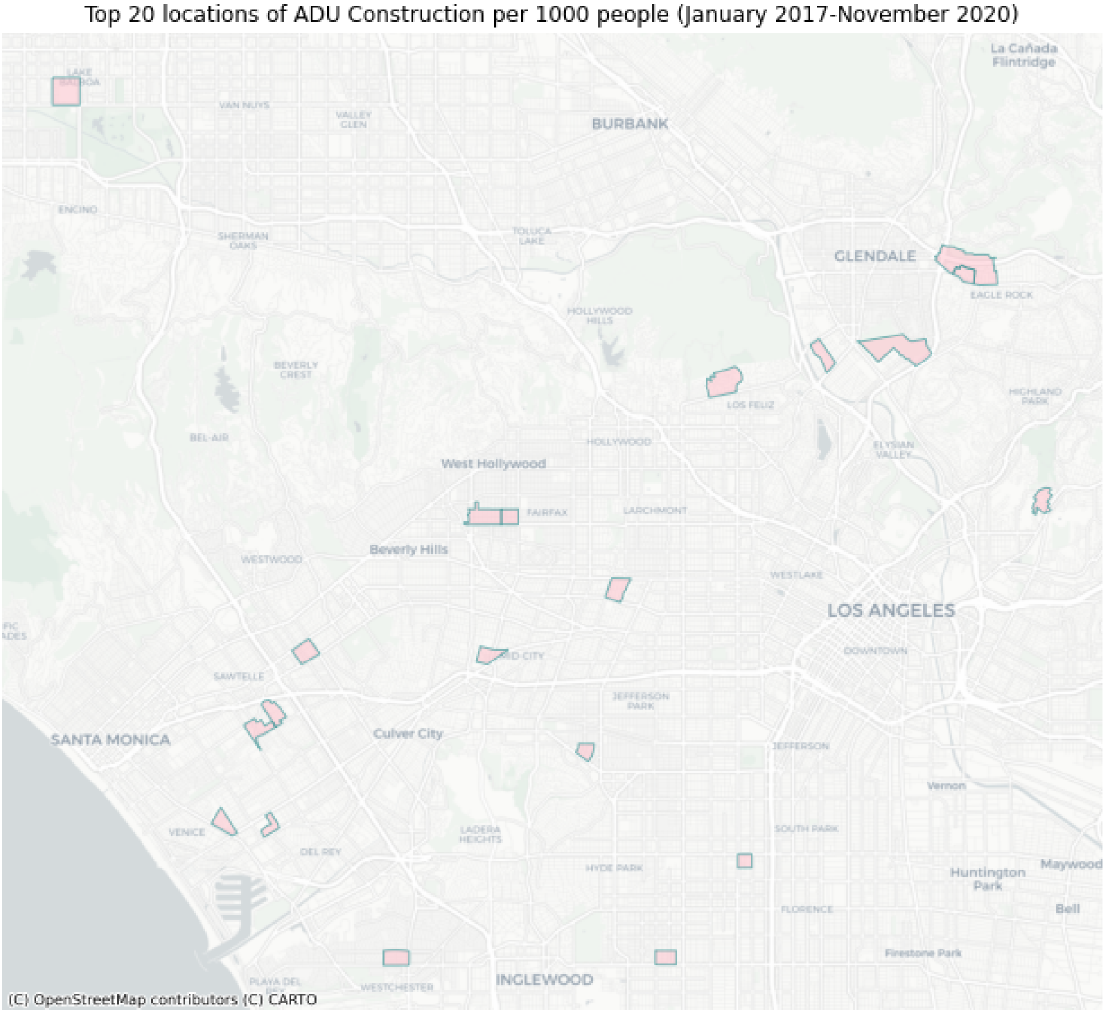
[229]:
```
             FIPS  TotalPop  \
360  060372737003     802.0
347  060372713002    1066.0
348  060372713005     532.0
183  060371945001    1044.0
281  060372361005     351.0

                                              geometry  adu_count  \
360  MULTIPOLYGON (((-13186803.464 4028447.410, -13…          3
347  MULTIPOLYGON (((-13185005.876 4032732.144, -13…          4
348  MULTIPOLYGON (((-13185660.880 4031904.885, -13…          2
183  MULTIPOLYGON (((-13176544.371 4039567.550, -13…          4
281  MULTIPOLYGON (((-13173862.461 4030997.106, -13…          2

     adu_per_1000
360      3.740648
347      3.752345
348      3.759398
183      3.831418
281      5.698006
```

Here, we sort the values by descending ADU production rate, and only show a slice of the data, the top 20 geographies using the handy `[:20]`.

[230]:
```
# map the top 20 geographies
ax = gdf.sort_values(by='adu_per_1000',ascending=False)[:20].
 ↪plot(figsize=(12,10),
                                                              color='pink',
                                                              edgecolor='teal',
                                                              alpha=0.
 ↪5,legend=True)
```

```
# title
ax.set_title('Top 20 locations of ADU Construction per 1000 people (January 2017-November 2020)')

# no axis
ax.axis('off')

# add a basemap
ctx.add_basemap(ax,source=ctx.providers.CartoDB.Positron)
```

Top 20 locations of ADU Construction per 1000 people (January 2017-November 2020)

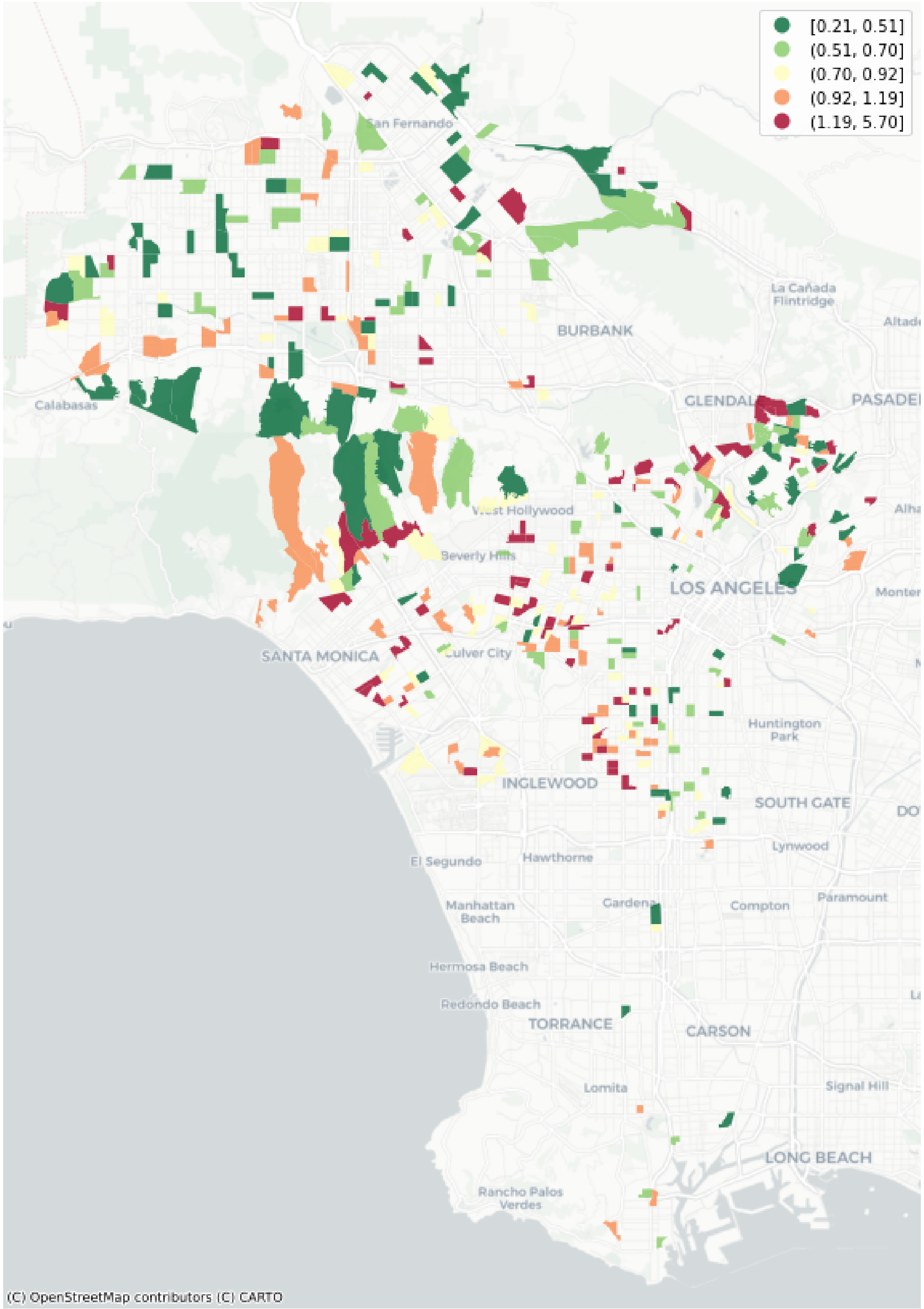


1.11 Choropleth map of ADUs

Finally, we are ready to generate a choropleth map of ADU permits.

[231]:
```
ax = gdf.plot(figsize=(15,15),
                        column='adu_per_1000',
                        legend=True,
                        alpha=0.8,
                        cmap='RdYlGn_r',
                        scheme='quantiles')

ax.axis('off')
ax.set_title('2017 January to 2020 November ADUs per 1000 people',fontsize=22)
ctx.add_basemap(ax,source=ctx.providers.CartoDB.Positron)
```

2017 January to 2020 November ADUs per 1000 people



The map above is a good way to begin exploring spatial patterns in our data. What does this map tell you? Is it informative? Do you notice any significant clusters? What if you change the map?

Notice the `scheme` argument is set to `naturalbreaks`. Experiment with other map classfications such as `equalinterval`, `quantiles`. How does each classification change the map?

2 Global Spatial Autocorrelation

We have imported two datasets. Cleaned them up, spatialized them, and connected them spatially. We successfully mapped them to show the location of ADUs per 1000 people by census block groups. The resulting map intuitively and visually tells us that there does appear to be spatial clusters of where ADU production is more prevalent, but to what degree of certainty can we say so? Actually, very little, without statitistically backing up our determinations. Could this exact pattern be a matter of chance? Or is the pattern so distinct that there is no way it could have happened randomly?

In order to answer this question, we conduct spatial autocorrelation, a process that determines to what degree an existing pattern is or is not random.

Global Moran's I statistic is a way to *quantify* the degree to which similar geographies are clustered. To do so, we compare each geography based on a given value (in this case ADU permit counts) with that of its neighbors. The first step of this process is to define a "spatial weight."

For this lab, we will use the KNN weight, where `k` is the number of "nearest neighbors" to count in the calculations. Let's proceed with `k=8` for our KNN spatial weights.

- https://geographicdata.science/book/notebooks/04_spatial_weights.html#distance-based-weights

[232]:
```
# calculate spatial weight
wq =  lps.weights.KNN.from_dataframe(gdf,k=8)
wq.transform = 'r'
```

2.0.1 Spatial lag

Now that we have our spatial weights assigned, we use it to calculate the spatial lag. While the mathematical operations are beyond the scope of this lab, you are welcome to check it out here. Simply put, the spatial lag is a calculated assignment to each geography in your data, which takes into account the data values from others in their "neighborhood" as defined by the spatial weight. This operation can be done with a single line of code which is part of the pysal module, but the underlying calculations are not that difficult to understand: it takes the average of all the neighbors as defined by the spatial weight to come up with a single associated value.

[233]:
```
# create a new column for the spatial lag
gdf['adu_per_1000_lag'] = lps.weights.lag_spatial(wq, gdf['adu_per_1000'])
```

[234]:
```
gdf.sort_values(by='adu_per_1000',ascending=False).sample(100)
```

[234]:
```
             FIPS  TotalPop  \
317  060372621002     515.0
81   060371284004     972.0
245  060372199021    2428.0
```

```
201  060372011203     791.0
178  060371927001    2951.0
..            …        …
78   060371281021    3462.0
174  060371910003    1242.0
227  060372168002    1233.0
364  060372764001    1079.0
128  060371810002    1442.0

                                              geometry  adu_count  \
317  MULTIPOLYGON (((-13186389.132 4039143.921, -13…          1
81   MULTIPOLYGON (((-13188443.200 4053750.612, -13…          1
245  MULTIPOLYGON (((-13176714.467 4032451.414, -13…          1
201  MULTIPOLYGON (((-13153794.563 4041968.635, -13…          1
178  MULTIPOLYGON (((-13168172.922 4040031.382, -13…          1
..                                                 …        …
78   MULTIPOLYGON (((-13185651.196 4054871.503, -13…          2
174  MULTIPOLYGON (((-13172065.765 4042455.554, -13…          1
227  MULTIPOLYGON (((-13177152.732 4036461.797, -13…          2
364  MULTIPOLYGON (((-13181744.327 4024958.364, -13…          1
128  MULTIPOLYGON (((-13161147.772 4048542.321, -13…          3

     adu_per_1000  adu_per_1000_lag
317      1.941748          0.908729
81       1.028807          0.901541
245      0.411862          0.988507
201      1.264223          0.821440
178      0.338868          0.760559
..            …               …
78       0.577701          0.613689
174      0.805153          0.892107
227      1.622060          1.314962
364      0.926784          1.135060
128      2.080444          1.188762

[100 rows x 6 columns]
```

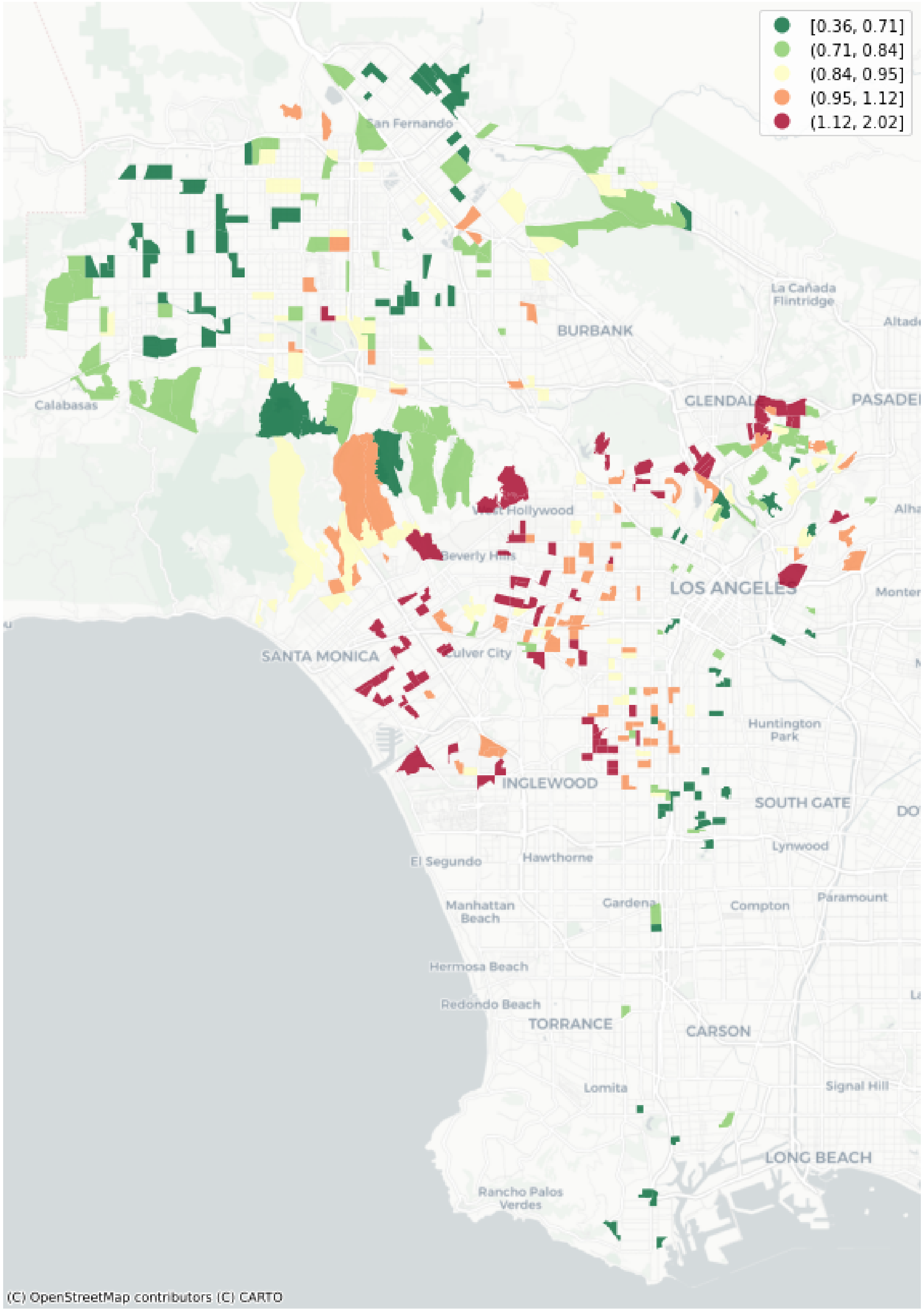
2.1 Spatial lag map

But we digress. Let's map the entire dataframe by the newly created spatial lag column.

[235]:
```
# use subplots that make it easier to create multiple layered maps
fig, ax = plt.subplots(figsize=(15, 15))

# spatial lag choropleth
gdf.plot(ax=ax,
         figsize=(15,15),
```

```
        column='adu_per_1000_lag',
        legend=True,
        alpha=0.8,
        cmap='RdYlGn_r',
        scheme='quantiles')

# uncomment this to see the actual point locations of ADUs
# adu.plot(ax=ax,
#               color='blue',
#               markersize =1,
#               alpha=0.2,
#               legend=True)

ax.axis('off')
ax.set_title('2017 January to 2020 November ADUs per 1000 people',fontsize=22)

ctx.add_basemap(ax,source=ctx.providers.CartoDB.Positron)
```

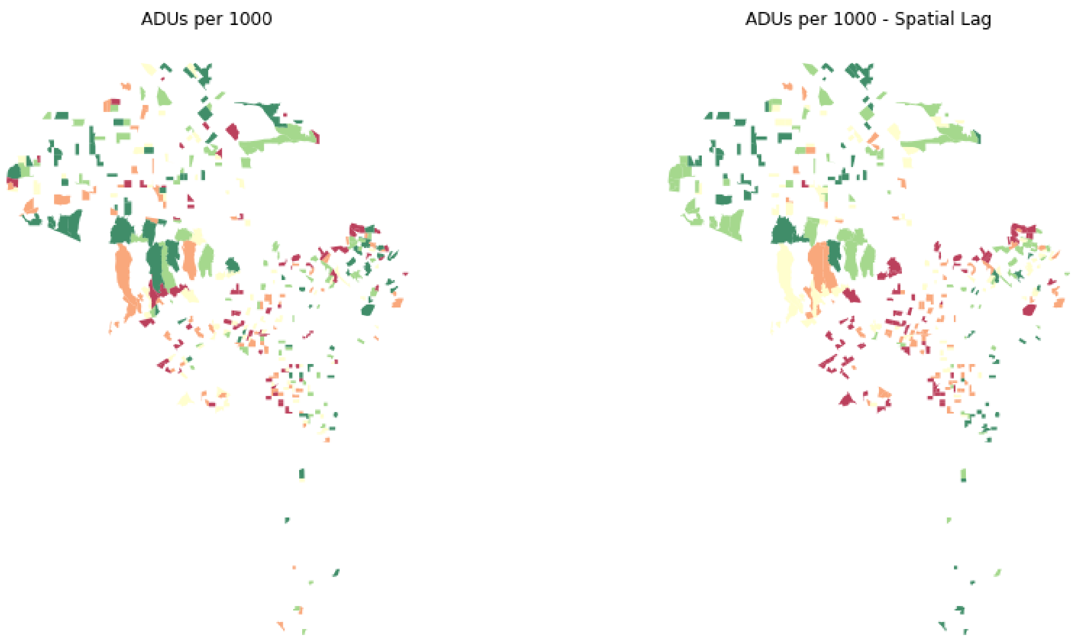
2017 January to 2020 November ADUs per 1000 people



2.2 Side-by-side maps

We can now compare these two map outputs side by side. Notice that the syntax is a bit different from past labs where we have only worked with one figure at a time. This output produces 1 row, and 2 columns of figures in `subplots`. - subplots documentation

[236]:
```python
# create the 1x2 subplots
fig, axs = plt.subplots(1, 2, figsize=(15, 8))

# name each subplot
ax1, ax2 = axs

# regular count map on the left
gdf.plot(column='adu_per_1000',
            cmap='RdYlGn_r',
            scheme='quantiles',
            k=5,
            edgecolor='white',
            linewidth=0.,
            alpha=0.75,
            ax=ax1 # this assigns the map to the subplot
           )


ax1.axis("off")
ax1.set_title("ADUs per 1000")

# spatial lag map on the right
gdf.plot(column='adu_per_1000_lag',
            cmap='RdYlGn_r',
            scheme='quantiles',
            k=5,
            edgecolor='white',
            linewidth=0.,
            alpha=0.75,
            ax=ax2 # this assigns the map to the subplot
           )

ax2.axis("off")
ax2.set_title("ADUs per 1000 - Spatial Lag")

plt.show()
```



2.3 Interactive spatial lag satellite map

Buidling the equivalent map as an interactive javascript map is a bit more challenging. While there are several options to choose from, this lab will use plotly express's choropleth_mapbox feature. - https://plotly.com/python/mapbox-county-choropleth/#

[237]:
```
# interactive version needs to be in WGS84
gdf_web = gdf.to_crs('EPSG:4326')
```

[238]:
```
# what's the centroid?
minx, miny, maxx, maxy = gdf_web.geometry.total_bounds
center_lat_gdf_web = (maxy-miny)/2+miny
center_lon_gdf_web = (maxx-minx)/2+minx
```

Unlike the matplotlib map, plotly's mapbox map only gives us a continuous scale option (there is no magical `scheme` option). To produce a similar quantile map, we need to calculate the values manually.

As we want to produce a choropleth map based on our spatial lag column, let's get some simple stats:

[239]:
```
# some stats
gdf_web.adu_per_1000_lag.describe()
```

[239]:
```
count    380.000000
mean       0.936097
```

```
std         0.268536
min         0.364564
25%         0.746395
50%         0.896920
75%         1.084832
max         2.017196
Name: adu_per_1000_lag, dtype: float64
```

[240]:
```
# set the mapbox access token
token = 'pk.eyJ1IjoieW9obWFuIiwiYSI6IkxuRThfNFkifQ.u2xRJMiChx914U7mOZMiZw'
px.set_mapbox_access_token(token)
```

[241]:
```
# grab the median
median = gdf_web.adu_per_1000_lag.median()
```

[242]:
```
fig = px.choropleth_mapbox(gdf_web,
                     geojson=gdf_web.geometry,
                     locations=gdf_web.index,
                     mapbox_style="satellite-streets",
                     zoom=9,
                     color='adu_per_1000_lag',
                     color_continuous_scale='RdYlGn_r',
                     color_continuous_midpoint =median,
                     range_color =(0,median*2),
                     hover_data=['adu_count','adu_per_1000','adu_per_1000_lag'],
                     center = {"lat": center_lat_gdf_web, "lon":␣
↪center_lon_gdf_web},
                     opacity=0.8,
                     width=1000,
                     height=800,
                     labels={
                             'adu_per_1000_lag':'ADUs per 1000 (Spatial Lag)',
                             'adu_per_1000':'ADUs per 1000',
                     })
fig.update_traces(marker_line_width=0.1, marker_line_color='white')
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
```

2.4 Moran's Plot

We now have a spatial lag map: a map that displays geographies weighted against the values of its neighbors. The clusters are much clearer and cleaner than the original ADU count map. Up to this point we still have not *quantified* the degree of the spatial correlations.

- To begin this process, we test for global autocorrelation for a continuous attribute (ADU counts).

[243]:
```
y = gdf.adu_per_1000
moran = Moran(y, wq)
moran.I
```

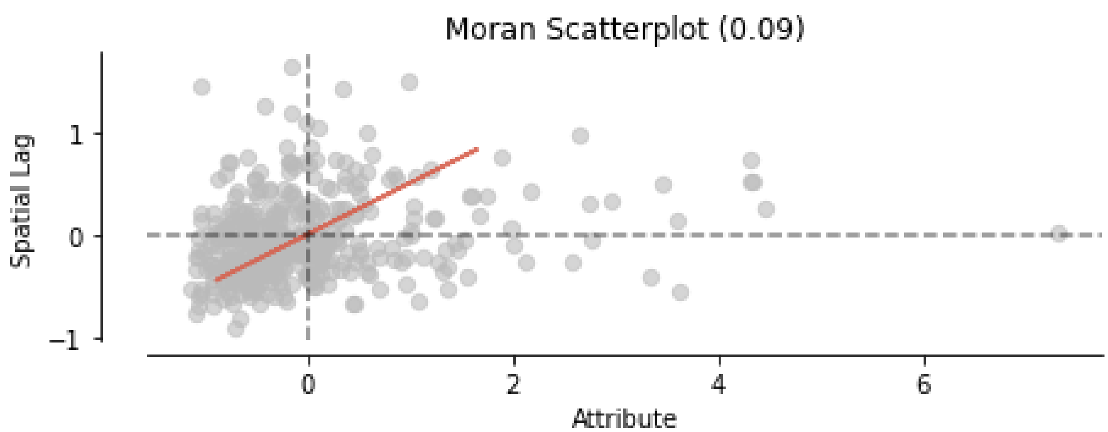
[243]: 0.08569127145430425

The moran's I value is nothing more than the calculated slope of the scatterplot of our "ADUs per 1000" and "ADUs per 1000 spatial lag" columns. It does indicate whether or not you have a positive or negative autocorrelation. Values will range from positive one, to negative one.

- **Positive** spatial autocorrelation: high values are close to high values, and/or low values are close to low values
- **Negative** spatial autocorrelation (less common): similar values are far from each other; high values are next to low values, low values are next to high values

You can output a scatterplot:

[244]:
```
fig, ax = moran_scatterplot(moran, aspect_equal=True)
plt.show()
```



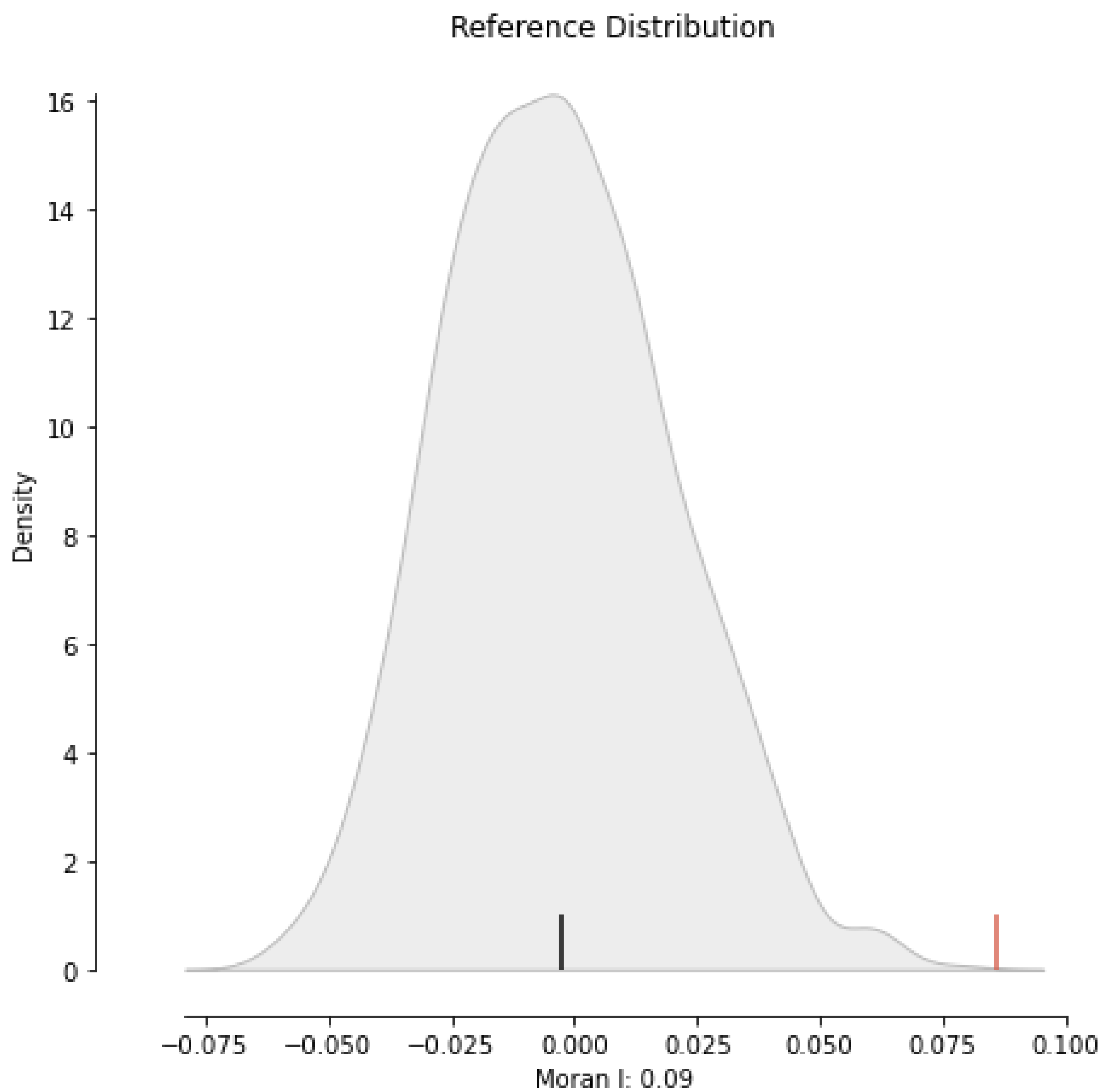
So what is the significance of our Moran value of 0.09? In other words, **how likely is our observed pattern on the map generated by an entirely random process?** To find out, we compare our value with a simulation of 999 permutations that randomly shuffles the ADU permit data throughout the given geographies. The output is a sampling distribution of Moran's I values under the (null) hypothesis that attribute values are randomly distributed across the study area. We then compare our observed Moran's I value to this "Reference Distribution."

[245]:
```
plot_moran_simulation(moran,aspect_equal=False)
```

```
/opt/conda/lib/python3.8/site-packages/splot/_viz_esda_mpl.py:47:
MatplotlibDeprecationWarning:


The set_smart_bounds function was deprecated in Matplotlib 3.2 and will be
```

```
removed two minor releases later.

/opt/conda/lib/python3.8/site-packages/splot/_viz_esda_mpl.py:48:
MatplotlibDeprecationWarning:


The set_smart_bounds function was deprecated in Matplotlib 3.2 and will be
removed two minor releases later.
```

[245]:
```
(<Figure size 504x504 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f5cd0283c70>)
```



We can compute the P-value:

[246]:
```
moran.p_sim
```

[246]:
```
0.001
```

The value is calculated as an empirical P-value that represents the proportion of realisations in the simulation under spatial randomness that are more extreme than the observed value. A small enough p-value associated with the Moran's I of a map allows to reject the hypothesis that the map is random. In other words, we can conclude that the map displays more spatial pattern than we would expect if the values had been randomly allocated to a locations.

That is a very low value, particularly considering it is actually the minimum value we could have obtained given the simulation behind it used 999 permutations (default in PySAL) and, by standard terms, it would be deemed statistically significant. We can ellaborate a bit further on the intuition behind the value of p_sim. If we generated a large number of maps with the same values but randomly allocated over space, and calculated the Moran's I statistic for each of those maps, only 0.01% of them would display a larger (absolute) value than the one we obtain from the observed data, and the other 99.99% of the random maps would receive a smaller (absolute) value of Moran's I.

3 Local Spatial Autocorrelation

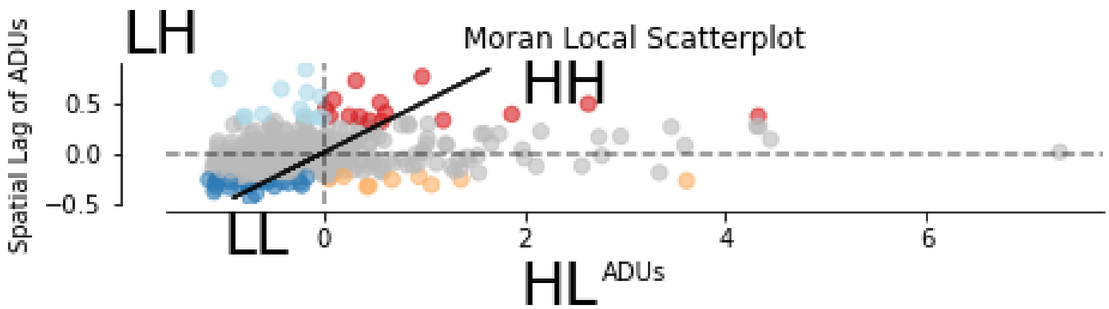
So far, we have only determined that there is a positive spatial autocorrelation between the price of properties in neighborhoods and their locations. But we have not detected where clusters are. Local Indicators of Spatial Association (LISA) is used to do that. LISA classifies areas into four groups: high values near to high values (HH), Low values with nearby low values (LL), Low values with high values in its neighborhood, and vice-versa.

- HH: high ADU production rate geographies near other high ADU production rate neighbors
- LL: low ADU production geographies near other low ADU production rate neighbors
- LH (donuts): low ADU production rate geographies surrounded by high ADU production rate neighbors
- HL (diamonds): high ADU production geographies surrounded by low ADU production rate neighbors

3.1 Moral Local Scatterplot

[247]:
```
# calculate local moran values
lisa = esda.moran.Moran_Local(y, wq)
```

[248]:
```
# Plot
fig, ax = moran_scatterplot(lisa, p=0.05)
ax.set_xlabel("ADUs")
ax.set_ylabel('Spatial Lag of ADUs')
plt.text(1.95, 0.5, "HH", fontsize=25)
plt.text(1.95, -1.5, "HL", fontsize=25)
plt.text(-2, 1, "LH", fontsize=25)
plt.text(-1, -1, "LL", fontsize=25)
plt.show()
```

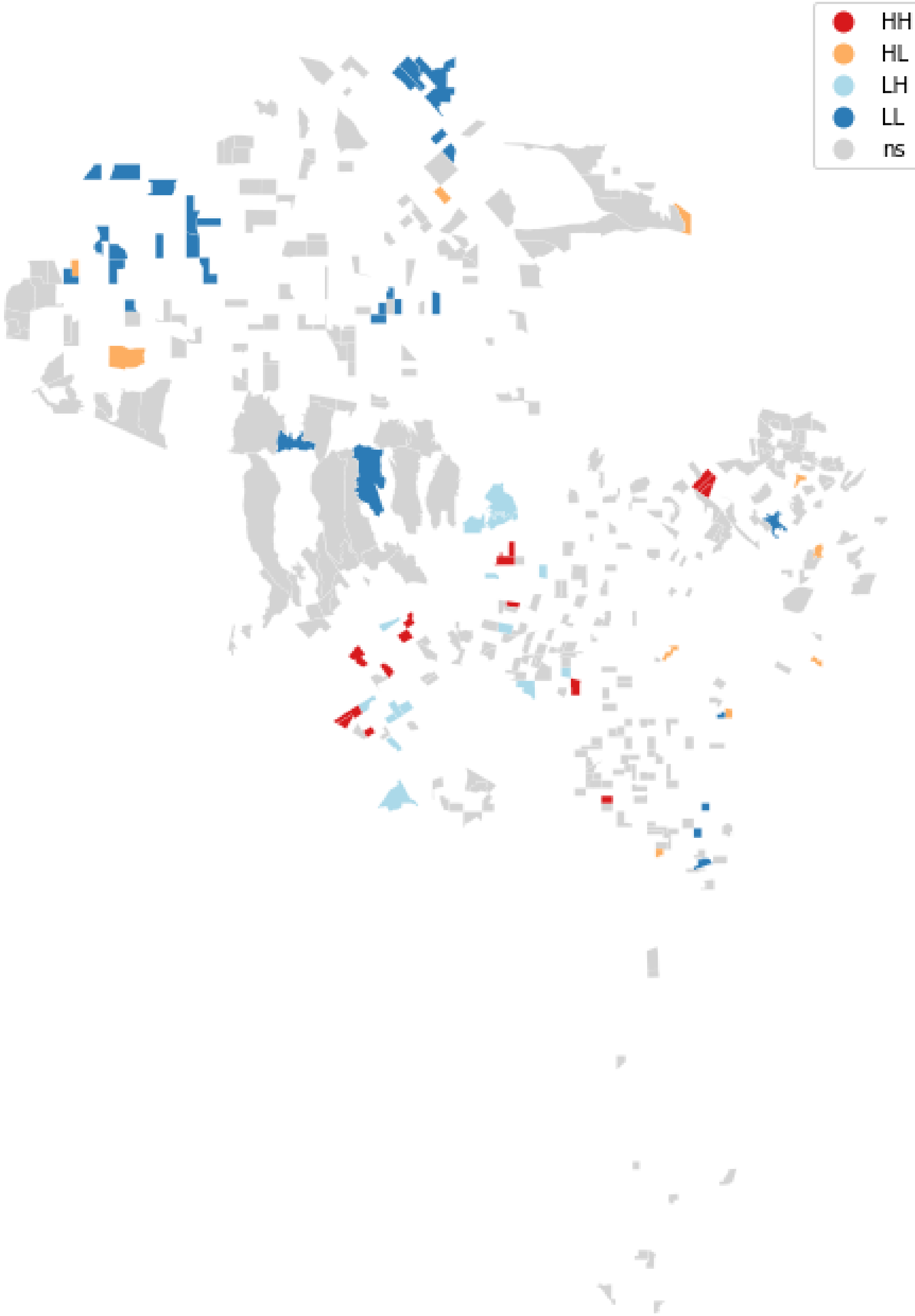


In the scatterplot above, the colored dots represents the rows that have a P-value less that 0.05 in each quadrant. In other words, these are the statisticaly significantly, spatially autocorrelated geographies.

3.2 Spatial Autocorrelation Map

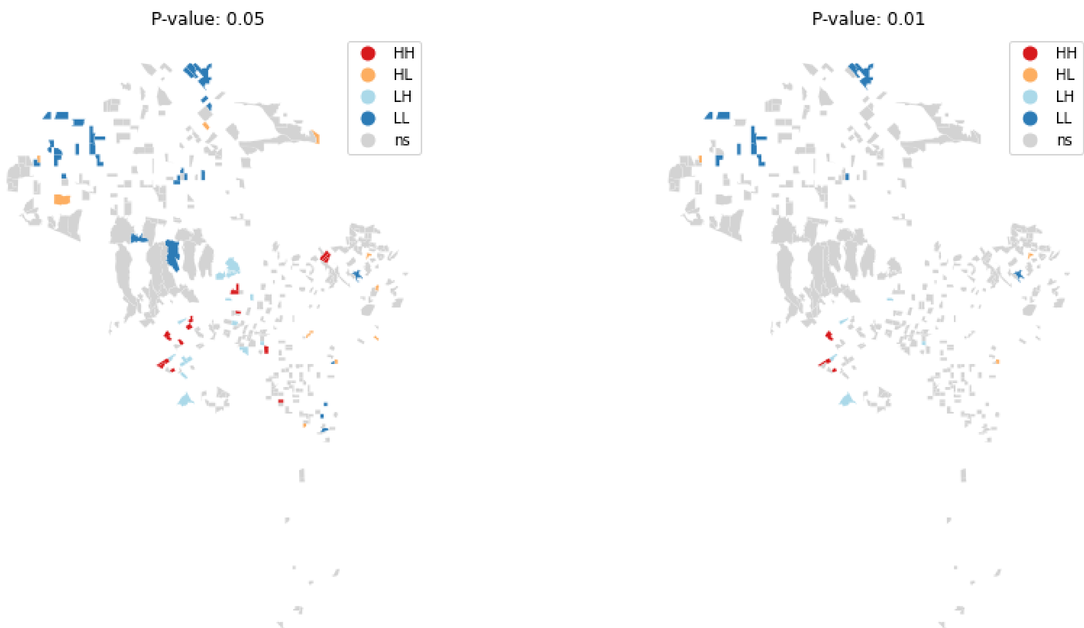
Finally, you can visually these statistically significant clusters using the `lisa_cluster` function:

[249]:
```
fig, ax = plt.subplots(figsize=(14,12))
lisa_cluster(lisa, gdf, p=0.05, ax=ax)
plt.show()
```


HH
HL
LH
LL
ns

And create a map comparing different p-values

[250]:
```
# create the 1x2 subplots
fig, axs = plt.subplots(1, 2, figsize=(15, 8))

# name each subplot
ax1, ax2 = axs

# regular count map on the left
lisa_cluster(lisa, gdf, p=0.05, ax=ax1)

ax1.axis("off")
ax1.set_title("P-value: 0.05")

# spatial lag map on the right
lisa_cluster(lisa, gdf, p=0.01, ax=ax2)
ax2.axis("off")
ax2.set_title("P-value: 0.01")

plt.show()
```



3.3 Creating an interactive version of the LISA map

The lisa function produces additional values that can be obtained:

- `lisa.y`: original value list
- `lisa.q`: quadrant list

- `lisa.p_sim`: p-value list

[251]:
```python
# original value list
lisa.y[:5]
```

[251]:
```
array([0.31816736, 1.56372166, 0.64935065, 0.56465274, 0.62421973])
```

[252]:
```python
# quadrant list
lisa.q[:5]
```

[252]:
```
array([3, 4, 3, 3, 3])
```

[253]:
```python
# p sim list
lisa.p_sim[:5]
```

[253]:
```
array([0.425, 0.045, 0.199, 0.46 , 0.267])
```

[254]:
```python
# add quadrant numbers to the dataframe
gdf['q'] = lisa.q.tolist()
```

[255]:
```python
# add individual p-values to the dataframe
gdf['p_sim'] = lisa.p_sim.tolist()
```

[256]:
```python
gdf.head()
```

[256]:
```
          FIPS  TotalPop                                           geometry  \
0  060371012102    3143.0  MULTIPOLYGON (((-13169034.646 4063225.625, -13…
1  060371013002    1279.0  MULTIPOLYGON (((-13166473.296 4061829.859, -13…
2  060371014002    1540.0  MULTIPOLYGON (((-13171587.314 4062370.826, -13…
3  060371021031    1771.0  MULTIPOLYGON (((-13176367.818 4059552.748, -13…
4  060371021072    1602.0  MULTIPOLYGON (((-13176542.701 4060290.729, -13…

   adu_count  adu_per_1000  adu_per_1000_lag  q  p_sim
0          1      0.318167          0.942949  3  0.425
1          2      1.563722          0.637621  4  0.045
2          1      0.649351          0.751918  3  0.199
3          1      0.564653          0.925931  3  0.460
4          1      0.624220          0.786867  3  0.267
```

3.4 Create a hotspot map

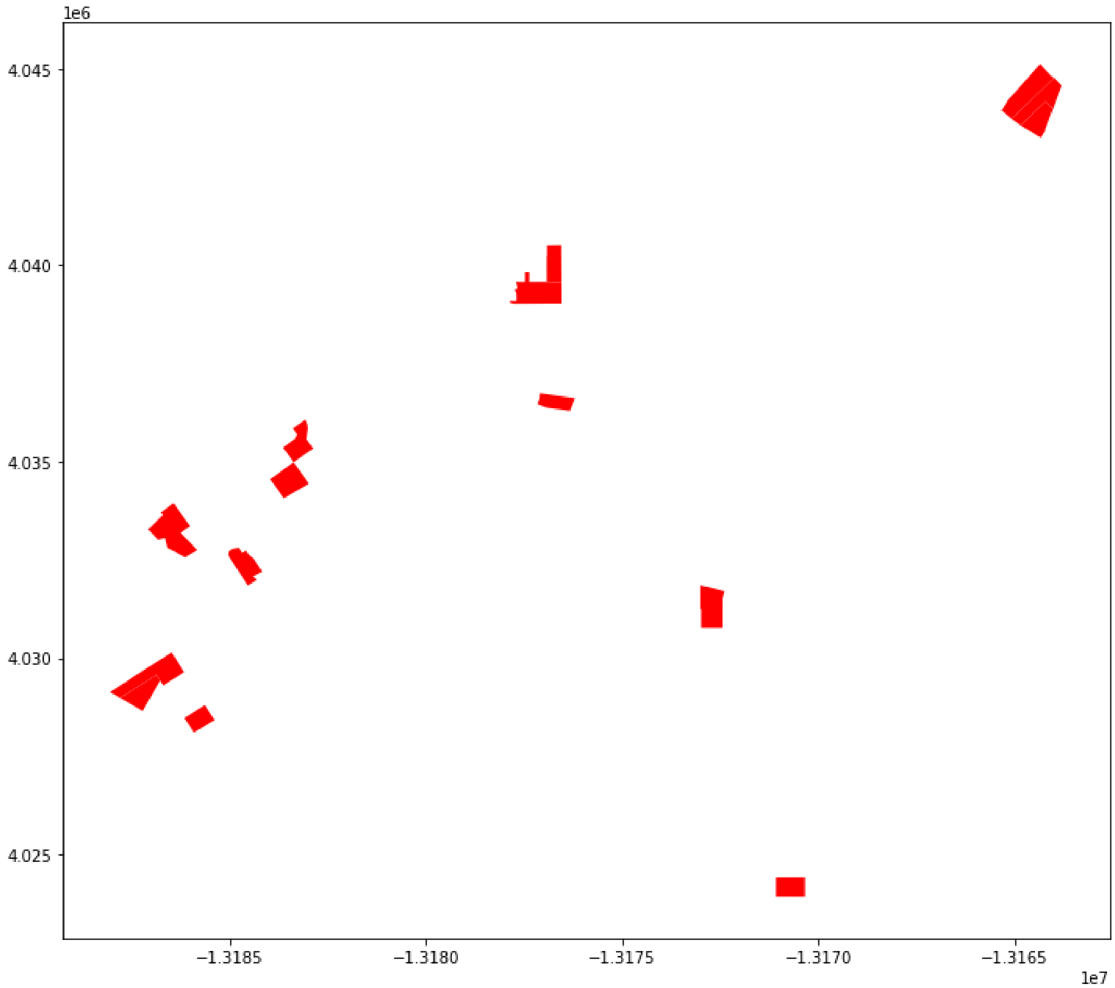
[257]:
```python
# identify just the hotspot geographies
hot_spots = gdf[(gdf.p_sim < 0.05) & (gdf.q == 1)]
```

[258]:
```python
hot_spots.shape
```

[258]:
```
(15, 8)
```

[259]:
```python
# quick plot... not very informative
hot_spots.plot(figsize=(12,12),color='red',legend=True,categorical=True)
```

[259]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5cd1b26610>



[260]:
```python
# interactive version needs to be in WGS84
hot_spots = hot_spots.to_crs('EPSG:4326')
```

[261]:
```python
# what's the centroid?
minx, miny, maxx, maxy = hot_spots.geometry.total_bounds
center_lat_hot_spots = (maxy-miny)/2+miny
center_lon_hot_spots = (maxx-minx)/2+minx
```

[262]:
```python
fig = px.choropleth_mapbox(hot_spots,
                     geojson=hot_spots.geometry,
                     locations=hot_spots.index,
```

```
                        mapbox_style="satellite-streets",
                        center = {"lat": center_lat_hot_spots, "lon":␣
↪center_lon_hot_spots},
                        zoom=9,
                        opacity=0.6,
                        color='adu_per_1000_lag',
                        color_continuous_scale='RdYlGn_r',
                        color_continuous_midpoint =median,
                        range_color =(0,median*2),
                        hover_data=['adu_count','adu_per_1000','adu_per_1000_lag'],
                        labels={
                                'adu_per_1000_lag':'ADUs per 1000 (Spatial Lag)',
                                'adu_per_1000':'ADUs per 1000',
                        })
fig.update_traces(marker_line_width=1, marker_line_color='white')
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
```

4 Map Interpretation

- Moran value of 0.09
- P-value of 0.003
- We are visualizing ADU permit approvals in the City of LA. From the statistical data it is clear that the data is statistically significant. I think the data confirms that ADUs are being produced in high home value neighborhoods.
- ADUs are still a new housing typology (only 3 years old) so there is not a lot of data. However, the hotspots remained mostly the same from the regular to the "lag" maps as seen above.
- For our own research – Atwater Village is a hot spot. This is surprising because the permit numbers are still low (5) but it shows that this is a significant amount of units relative to other patterns in the City. There is also a lot more permits being issued in the west side of LA versus the other neighborhoods.

[ ]:
```

```