Census Exploration Lab 1_24

January 25, 2021

0.1 Exploring Median Household Income, and Owner v. Renter status of households across California

Jacqueline Adams 1/24/2021

This notebook will explore some of the Census data relevant to my final project, which will ultimately at water bill debt across the state of California

0.1.1 Uploading Libraries

I'm going to upload 4 different libraries to help me in my analysis: * Pandas for wrangling data * Geopandas for data visualization * Contextily for basemaps * Matplotlib.pyplot for plots and figures

The following code uploads these libraries and shortens their names to make them a bit easier to use when coding:

[31]:
```
import pandas as pd

import geopandas as gpd

import contextily as ctx

import matplotlib.pyplot as plt
```

Now that the libraries are uploaded I can move on to the data itself…

0.1.2 Import Census Data

First, I went to censusreporter.org and obtained a dataset that contained aggregate household income in the past 12 months, number of owner occupued houses, and number of renter occupied houses. This data was divided by ZIP code (as my water debt data is also divided by ZIP). I then downloaded this data as a Geojson file, unzipped the file, and uploaded it to my Jupyter folder titled "Data." Below you'll find the code for how I uploaded this data file to this Notebook:

[18]:
```
acs = gpd.read_file('Data/acs2019_5yr_B25120_86000US93673.geojson')
```

I uploaded the data as "acs" for American Community Survey. After a bit of a wait, the data file itself uploaded to Jupyter, and the above code copied it into this notebook. Now, I'll start getting a sense of how the data look…

0.1.3 First Look at the Data

Next I am going to get a sense of how the data look: size, type, missing data, the first five rows, and get a visual understanding.

[19]:
```
acs.shape
```

[19]:
```
(1776, 13)
```

[23]:
```
acs.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 1776 entries, 0 to 1775
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   geoid             1776 non-null   object
 1   name              1776 non-null   object
 2   B25120001         1540 non-null   float64
 3   B25120001, Error  1540 non-null   float64
 4   B25120002         1528 non-null   float64
 5   B25120002, Error  1528 non-null   float64
 6   B25120003         1523 non-null   float64
 7   B25120003, Error  1523 non-null   float64
 8   B25120004         1523 non-null   float64
 9   B25120004, Error  1523 non-null   float64
 10  B25120005         1525 non-null   float64
 11  B25120005, Error  1525 non-null   float64
 12  geometry          1776 non-null   geometry
dtypes: float64(10), geometry(1), object(2)
memory usage: 180.5+ KB
```

[22]:
```
acs.head()
```

[22]:
```
          geoid        name     B25120001  B25120001, Error     B25120002  \
0     04000US06  California  1.394637e+12      5.695883e+09  9.639479e+11
1  86000US89010       89010  1.133300e+07      2.496212e+06  7.468800e+06
2  86000US89019       89019  1.046821e+08      6.567377e+07  9.497140e+07
3  86000US89046       89046  8.360700e+06      2.646964e+06  5.915400e+06
4  86000US89060       89060  2.258843e+08      2.558462e+07  1.888915e+08

   B25120002, Error     B25120003  B25120003, Error     B25120004  \
0      6.573099e+09  7.428685e+11      4.804820e+09  2.210794e+11
1      2.079910e+06  1.335500e+06      9.682500e+05  6.133200e+06
2      6.527523e+07  1.598290e+07      8.185754e+06  7.898850e+07
3      2.589367e+06  2.101900e+06      1.889446e+06  3.813500e+06
4      2.539575e+07  1.204472e+08      2.794166e+07  6.844430e+07
```

```
   B25120004, Error     B25120005  B25120005, Error  \
0      2.419329e+09  4.306887e+11      1.966668e+09
1      1.884313e+06  3.864200e+06      1.515361e+06
2      6.545474e+07  9.710700e+06      3.190459e+06
3      1.901507e+06  2.445300e+06      1.294253e+06
4      1.352304e+07  3.699270e+07      1.213679e+07

                                            geometry
0  MULTIPOLYGON (((-124.13656 41.46445, -124.1378…
1  MULTIPOLYGON (((-118.43518 37.90129, -118.4301…
2  MULTIPOLYGON (((-115.83450 35.95483, -115.8342…
3  MULTIPOLYGON (((-115.22538 35.47589, -115.2231…
4  MULTIPOLYGON (((-116.30350 36.41833, -116.3008…
```

The above code has told me the size of the data, the type of each column of data as well as if there are any missing data (I can see this varies across columns), given me a preview of the first 5 lines of data, and visually plotted all of the data (which is, as expected, the state of California). Now that I have a general sense of the size of the data and what they look like, I need to make some minor adjustments to the data itself.

0.1.4 Data Adjustments

Next, I'll need to make a few edits to the data so that it is a bit more readable and easier to work with.

[24]:
```
acs = acs.drop([0])
```

[25]:
```
acs.head()
```

[25]:
```
          geoid   name    B25120001  B25120001, Error    B25120002  \
1  86000US89010  89010   11333000.0         2496212.0    7468800.0
2  86000US89019  89019  104682100.0        65673767.0   94971400.0
3  86000US89046  89046    8360700.0         2646964.0    5915400.0
4  86000US89060  89060  225884300.0        25584617.0  188891500.0
5  86000US89061  89061  173267200.0        21036604.0  154278400.0

   B25120002, Error    B25120003  B25120003, Error   B25120004  \
1         2079910.0    1335500.0          968250.0   6133200.0
2        65275226.0   15982900.0         8185754.0  78988500.0
3         2589367.0    2101900.0         1889446.0   3813500.0
4        25395747.0  120447200.0        27941660.0  68444300.0
5        22497739.0  101005200.0        20957293.0  53273200.0

   B25120004, Error   B25120005  B25120005, Error  \
1         1884313.0   3864200.0         1515361.0
2        65454741.0   9710700.0         3190459.0
3         1901507.0   2445300.0         1294253.0
4        13523044.0  36992700.0        12136793.0
```

```
5        15928689.0  18988800.0         8833788.0

                                          geometry
1  MULTIPOLYGON (((-118.43518 37.90129, -118.4301…
2  MULTIPOLYGON (((-115.83450 35.95483, -115.8342…
3  MULTIPOLYGON (((-115.22538 35.47589, -115.2231…
4  MULTIPOLYGON (((-116.30350 36.41833, -116.3008…
5  MULTIPOLYGON (((-115.90394 35.97005, -115.8994…
```

The first line of data showed median income and owner/renter status for the whole state of California. I deleted this line so that my analysis was no skewed by these statewide estimates. Then, I checked the data to ensure the line was deleted.

Next, I am going to remove certain columns.

[26]:
```
list(acs)
```

[26]:
```
['geoid',
 'name',
 'B25120001',
 'B25120001, Error',
 'B25120002',
 'B25120002, Error',
 'B25120003',
 'B25120003, Error',
 'B25120004',
 'B25120004, Error',
 'B25120005',
 'B25120005, Error',
 'geometry']
```

[27]:
```
columns_to_keep = ['geoid',
 'name',
 'B25120001',
 'B25120002',
 'B25120003',
 'B25120004',
 'B25120005',
 'geometry']
```

[28]:
```
acs = acs[columns_to_keep]
```

[29]:
```
acs.head()
```

[29]:
```
          geoid   name    B25120001    B25120002    B25120003    B25120004  \
1  86000US89010  89010   11333000.0    7468800.0    1335500.0    6133200.0
2  86000US89019  89019  104682100.0   94971400.0   15982900.0   78988500.0
3  86000US89046  89046    8360700.0    5915400.0    2101900.0    3813500.0
```

```
4  86000US89060  89060  225884300.0  188891500.0  120447200.0  68444300.0
5  86000US89061  89061  173267200.0  154278400.0  101005200.0  53273200.0

    B25120005                                          geometry
1   3864200.0  MULTIPOLYGON (((-118.43518 37.90129, -118.4301…
2   9710700.0  MULTIPOLYGON (((-115.83450 35.95483, -115.8342…
3   2445300.0  MULTIPOLYGON (((-115.22538 35.47589, -115.2231…
4  36992700.0  MULTIPOLYGON (((-116.30350 36.41833, -116.3008…
5  18988800.0  MULTIPOLYGON (((-115.90394 35.97005, -115.8994…
```

Above, you'll see that I obtained a list of all the columns, and then selected which ones I wanted to keep (i.e. removed all the standard error columns), and once again checked my data to ensure the proper columns had been removed.

The column names themselves are a bit confusing and meaningless... so I also need to change them to be a little more understandable

[30]:
```python
list(acs)
```

[30]:
```
['geoid',
 'name',
 'B25120001',
 'B25120002',
 'B25120003',
 'B25120004',
 'B25120005',
 'geometry']
```

[32]:
```python
acs.columns = ['geoid',
 'name',
 'Aggregate household income in the past 12 months',
 'Owner occupied',
 'Housing units with a mortgage',
 'Housing units without a mortgage',
 'Renter occupied',
 'geometry']
```

[22]:
```python
acs.head()
```

[22]:
```
         geoid   name  Aggregate household income in the past 12 months  \
1  86000US89010  89010                                        11333000.0
2  86000US89019  89019                                       104682100.0
3  86000US89046  89046                                         8360700.0
4  86000US89060  89060                                       225884300.0
5  86000US89061  89061                                       173267200.0

   Owner occupied  Housing units with a mortgage  \
1       7468800.0                      1335500.0
```

```
2      94971400.0                       15982900.0
3       5915400.0                        2101900.0
4     188891500.0                      120447200.0
5     154278400.0                      101005200.0

   Housing units without a mortgage  Renter occupied  \
1                         6133200.0        3864200.0
2                        78988500.0        9710700.0
3                         3813500.0        2445300.0
4                        68444300.0       36992700.0
5                        53273200.0       18988800.0

                                            geometry
1  MULTIPOLYGON (((-118.43518 37.90129, -118.4301…
2  MULTIPOLYGON (((-115.83450 35.95483, -115.8342…
3  MULTIPOLYGON (((-115.22538 35.47589, -115.2231…
4  MULTIPOLYGON (((-116.30350 36.41833, -116.3008…
5  MULTIPOLYGON (((-115.90394 35.97005, -115.8994…
```

Above, I listed each column, and then renamed them based on the column titles in the Metadata.json file. Finally, I checked the data again to make sure the columns were titled appropriately.

0.1.5 Summary Statistics

Next, I want to get a sense of the mean, median, and overall disribution of the data. My three columns of interest are aggregate household income, owner occupied, and renter occupied.

[33]:
```
acs['Aggregate household income in the past 12 months'].describe().apply(lambda␣
↪x: format(x, 'f'))
```

[33]:
```
count          1539.000000
mean      905974798.375569
std       854282445.991729
min         1845200.000000
25%       116631750.000000
50%       759512100.000000
75%      1414723900.000000
max      4840138700.000000
Name: Aggregate household income in the past 12 months, dtype: object
```

[34]:
```
acs['Owner occupied'].describe().apply(lambda x: format(x, 'f'))
```

[34]:
```
count          1527.000000
mean      631519280.091683
std       622936158.653026
min          752400.000000
25%        88930050.000000
50%       472182400.000000
```

```
75%        968458600.000000
max       3171893300.000000
Name: Owner occupied, dtype: object
```

[35]:
```
acs['Renter occupied'].describe().apply(lambda x: format(x, 'f'))
```

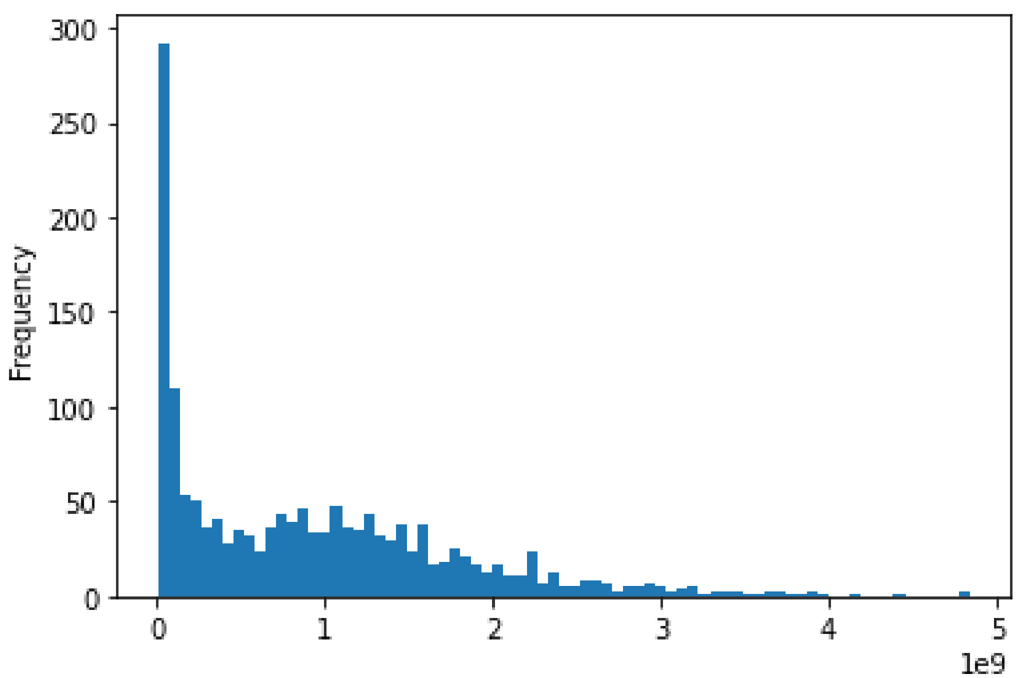
[35]:
```
count          1524.000000
mean      282129442.388451
std       326636806.224488
min          294200.000000
25%        26840125.000000
50%       202712150.000000
75%       425378650.000000
max      3516740800.000000
Name: Renter occupied, dtype: object
```

The above outputs give me a snapshot of how the data look: mean, standard deviation, min and max, as well as quartiles. The output was originally in scientific notation, but I found some code via a quick Google search, to remove that notation and make my results a bit more readable.

I am also going to get a visual representation of the spread.

[36]:
```
acs['Aggregate household income in the past 12 months'].plot.hist(bins=75)
```

[36]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9353436c10>
```



Above, we see there is a really high concentration of household income centered around 0… which suggests a really high number of low income Zip code areas across the seate. We know none of these Zip codes have a population of 0, because we can obtain the number of owner and renter occupied households from the data as well.

However, it might be a good idea to determine the % of households that are owner and renter occupied in each area. This would prove a more useful metric to better understand the population makeup in each Zip code area.

0.2 Normalizing and Benchmarking Housing Characteristics

I decided, since there is no population data in this set other than the number of reported owner occupied and renter occupied houses, that I would create a percentage of each (owner and renter) in each area based on the given data. I created two new columns.

[37]:
```
acs['Percent Owner Occupied'] = acs['Owner occupied'] / (acs['Owner occupied']␣
↪+ acs['Renter occupied'])
```

[38]:
```
acs['Percent Renter Occupied'] = acs['Renter occupied'] / (acs['Owner␣
↪occupied'] + acs['Renter occupied'])
```

[39]:
```
acs.head()
```

[39]:
```
           geoid   name  Aggregate household income in the past 12 months  \
1  86000US89010  89010                                        11333000.0
2  86000US89019  89019                                       104682100.0
3  86000US89046  89046                                         8360700.0
4  86000US89060  89060                                       225884300.0
5  86000US89061  89061                                       173267200.0

   Owner occupied  Housing units with a mortgage  \
1       7468800.0                      1335500.0
2      94971400.0                     15982900.0
3       5915400.0                      2101900.0
4     188891500.0                    120447200.0
5     154278400.0                    101005200.0

   Housing units without a mortgage  Renter occupied  \
1                         6133200.0        3864200.0
2                        78988500.0        9710700.0
3                         3813500.0        2445300.0
4                        68444300.0       36992700.0
5                        53273200.0       18988800.0

                                            geometry  Percent Owner Occupied  \
1  MULTIPOLYGON (((-118.43518 37.90129, -118.4301…                0.659031
2  MULTIPOLYGON (((-115.83450 35.95483, -115.8342…                0.907236
3  MULTIPOLYGON (((-115.22538 35.47589, -115.2231…                0.707524
```

```
4  MULTIPOLYGON (((-116.30350 36.41833, -116.3008…                0.836232
5  MULTIPOLYGON (((-115.90394 35.97005, -115.8994…                0.890407

   Percent Renter Occupied
1                 0.340969
2                 0.092764
3                 0.292476
4                 0.163768
5                 0.109593
```

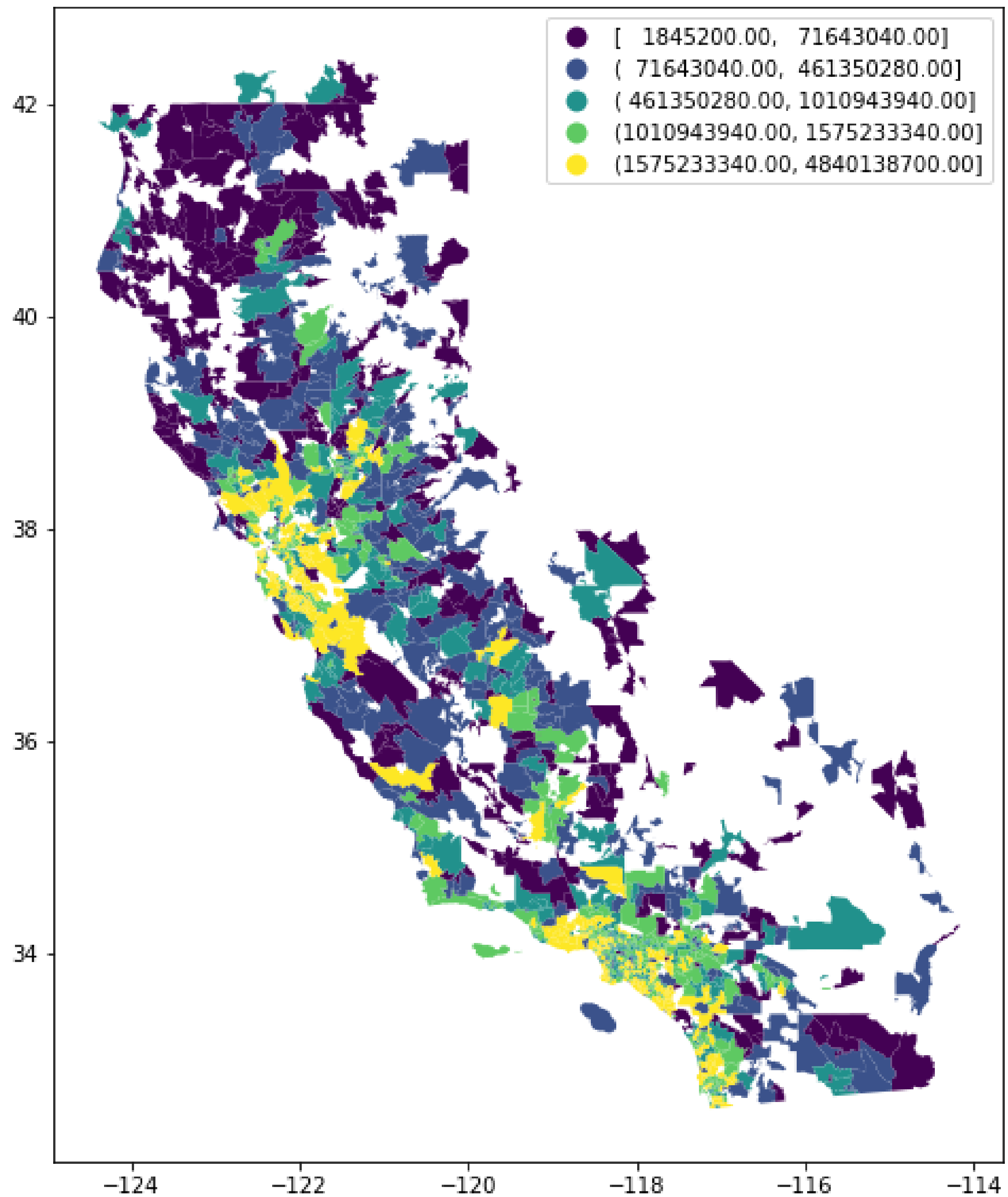
Above, I created 2 new cells that now tells me the percent of owner occupied and percent of renter occupied units in each zip code. This is, of course, in reference to the total number of reporter owner and renter occupied households reported in each area. This is a benchmark reference to estimate the makeup of housing in a given area.

0.2.1 Maps

Next, I want to create a couple of maps to get a sense of income and housing characteristics across the state. First, I am going to create a map that gives a sense of statewide income distribution:

[40]:
```
acs.plot(figsize=(12,10),
                 column='Aggregate household income in the past 12 months',
                 legend=True,
                 scheme='quantiles')
```

[40]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9353274310>
```



Above you'll see a statewide map of aggregate income across the state divided by quantiles. Note that aggregate income is consumption expenditure plus net profits (hence why the minimum is so high when compared to a simple median household income…though this data would also be interesting and I might incorporate that into my project as well). You can clearly see the concentration of high aggregate incomes around Los Angeles/San Diego and the San Francisco/Sacramento areas.

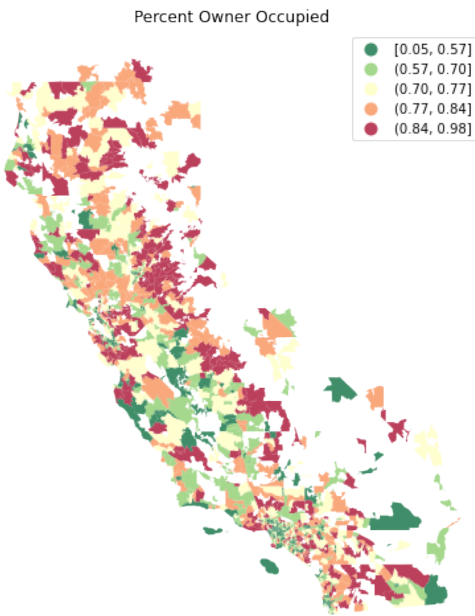
Next I wanted to compare the spread of owner vs renter occupied households.

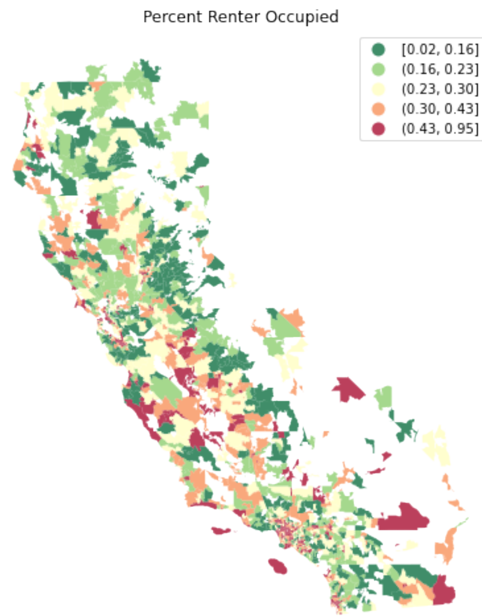
[42]:

```python
# create the 1x2 subplots
fig, axs = plt.subplots(1, 2, figsize=(15, 12))

# name each subplot
ax1, ax2 = axs

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

ax1.axis("off")
ax1.set_title("Percent Owner Occupied")

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

ax2.axis("off")
ax2.set_title("Percent Renter Occupied")
```

[42]:

```
Text(0.5, 1.0, 'Percent Renter Occupied')
```





Above you can see the comparison of owner v renter occupied households. I was surprised at the very high concentration of owner occupied households statewide - this number was significantly larger than I expected. I think a future analysis should see how these values compare to aggregate income and median household income.