Midterm Pt 2

February 8, 2021

0.1 Midterm Project Part 2: Tracking Water Bill Debt across California

My previous file ran out of memory, so I am picking up with mapping on this second file…

I need to pick back up really quick by importing my libraries and uploading my data

[29]:
```
import pandas as pd
import geopandas as gpd
import contextily as ctx
import matplotlib.pyplot as plt
import urllib.request, json
import plotly.express as px
from ipywidgets import interact
from keplergl import KeplerGl
```

[30]:
```
acs = pd.read_csv('Data/URBNPL206A Dataset - Sheet1.csv')

zip = gpd.read_file('Data/ca_california_zip_codes_geo.min.json')
```

0.1.1 Cleaning Data

Before I can map my data, I need to clean it a bit as I did in my first notebook

[31]:
```
refined_columns = ['Zip Codes',
 'Sum of Less than $100',
 'Sum of $100-$200',
 'Sum of $200-$300',
 'Sum of $300-$400',
 'Sum of $400-$500',
 'Sum of $500-$600',
 'Sum of $600-$700',
 'Sum of $700-$800',
 'Sum of $800-$900',
 'Sum of $900-$1000',
 'Sum of More than $1000',
 'Sum of Total number of delinquent residential accounts',
 'pop',
 'mhhi',
 'pct_black',
```

```
'pct_hisp',
'pct_asian',
'pct_povt',
'% Renter Pop',
'% Owner Pop',
'Households']
```

Quickly, I'm just re-refining my columns

[32]:
```
acs = acs[refined_columns]
```

[33]:
```
acs['Percent Deliquent'] = acs['Sum of Total number of delinquent residential␣
↪accounts']/acs['pop']
acs['Percent Less than $100'] = acs['Sum of Less than $100']/acs['Sum of Total␣
↪number of delinquent residential accounts']
acs['Percent $100 - $200'] = acs['Sum of $100-$200']/acs['Sum of Total number␣
↪of delinquent residential accounts']
acs['Percent $200 - $300'] = acs['Sum of $200-$300']/acs['Sum of Total number␣
↪of delinquent residential accounts']
acs['Percent $300 - $400'] = acs['Sum of $300-$400']/acs['Sum of Total number␣
↪of delinquent residential accounts']
acs['Percent $400 - $500'] = acs['Sum of $400-$500']/acs['Sum of Total number␣
↪of delinquent residential accounts']
acs['Percent $500 - $600'] = acs['Sum of $500-$600']/acs['Sum of Total number␣
↪of delinquent residential accounts']
acs['Percent $600 - $700'] = acs['Sum of $600-$700']/acs['Sum of Total number␣
↪of delinquent residential accounts']
acs['Percent $700 - $800'] = acs['Sum of $700-$800']/acs['Sum of Total number␣
↪of delinquent residential accounts']
acs['Percent $800 - $900'] = acs['Sum of $800-$900']/acs['Sum of Total number␣
↪of delinquent residential accounts']
acs['Percent $900 - $1000'] = acs['Sum of $900-$1000']/acs['Sum of Total number␣
↪of delinquent residential accounts']
acs['Percent More than $1000'] = acs['Sum of More than $1000']/acs['Sum of␣
↪Total number of delinquent residential accounts']
```

And here I've added in the percent columns again… .

Now time to map.

0.1.2 Maps

Now it is finally time to create some maps with this data. The first thing I need to do is merge the GeoJson file with my .csv file.

The first thing I need to do is get a sense of my zip boundary file, and figure out where I can merge it with my water bill debt file.

[34]:
```
list(zip)
```

[34]:
```
['STATEFP10',
 'ZCTA5CE10',
 'GEOID10',
 'CLASSFP10',
 'MTFCC10',
 'FUNCSTAT10',
 'ALAND10',
 'AWATER10',
 'INTPTLAT10',
 'INTPTLON10',
 'PARTFLG10',
 'geometry']
```

I just need to rename the Zip Code column so that I can easily match it.

[35]:
```
zip.columns = ['STATEFP10',
 'Zip Codes',
 'GEOID10',
 'CLASSFP10',
 'MTFCC10',
 'FUNCSTAT10',
 'ALAND10',
 'AWATER10',
 'INTPTLAT10',
 'INTPTLON10',
 'PARTFLG10',
 'geometry']
```

I changed the column titled 'ZCTA5CE10' to 'Zip Codes'

Now let me check my work really quick….

[36]:
```
zip.head()
```

[36]:
```
  STATEFP10 Zip Codes  GEOID10 CLASSFP10 MTFCC10 FUNCSTAT10   ALAND10  \
0        06     94601  0694601        B5   G6350          S   8410939
1        06     94501  0694501        B5   G6350          S  20539466
2        06     94560  0694560        B5   G6350          S  35757865
3        06     94587  0694587        B5   G6350          S  51075108
4        06     94580  0694580        B5   G6350          S   8929836

   AWATER10   INTPTLAT10    INTPTLON10 PARTFLG10  \
0    310703  +37.7755447  -122.2187049         N
1   9005303  +37.7737968  -122.2781230         N
2     60530  +37.5041413  -122.0323587         N
3         0  +37.6031556  -122.0186382         N
```

```
4     17052  +37.6757312  -122.1330170         N

                                            geometry
0  POLYGON ((-122.22717 37.79197, -122.22693 37.7…
1  POLYGON ((-122.29181 37.76301, -122.30661 37.7…
2  POLYGON ((-122.05499 37.54959, -122.05441 37.5…
3  POLYGON ((-122.06515 37.60485, -122.06499 37.6…
4  POLYGON ((-122.12999 37.68445, -122.12995 37.6…
```

Looks good! We now have 'Zip Codes' columns in both data sets.

Now to merge the datasets....

[37]:
```
merged = acs.merge(zip,
                   on='Zip Codes')
```

This function indicates that I will merge the 2 datasets on the column titled 'Zip Codes'

Now I want to check to make sure it worked.

[38]:
```
merged.head()
```

[38]:
```
  Zip Codes  Sum of Less than $100  Sum of $100-$200  Sum of $200-$300  \
0     90001                 5726.0            4937.0            1582.0
1     90002                 3130.0            2152.0            1052.0
2     90003                 1829.0            1880.0            1562.0
3     90004                 2199.0            1659.0            1119.0
4     90005                 1712.0            1107.0             598.0

   Sum of $300-$400  Sum of $400-$500  Sum of $500-$600  Sum of $600-$700  \
0             684.0             395.0             283.0             220.0
1             708.0             493.0             385.0             343.0
2            1169.0             941.0             782.0             673.0
3             735.0             502.0             387.0             279.0
4             401.0             281.0             175.0             146.0

   Sum of $700-$800  Sum of $800-$900  …  GEOID10  CLASSFP10  MTFCC10  \
0             137.0             132.0  …  0690001         B5    G6350
1             281.0             231.0  …  0690002         B5    G6350
2             513.0             482.0  …  0690003         B5    G6350
3             223.0             206.0  …  0690004         B5    G6350
4              95.0              95.0  …  0690005         B5    G6350

   FUNCSTAT10  ALAND10  AWATER10   INTPTLAT10    INTPTLON10 PARTFLG10  \
0           S  9071361         0  +33.9740268  -118.2495088         N
1           S  7930685         0  +33.9490988  -118.2467371         N
2           S  9197642       403  +33.9641307  -118.2727831         N
3           S  7894533         0  +34.0761981  -118.3107225         N
4           S  2807559         0  +34.0591634  -118.3068924         N
```

```
                                          geometry
0  POLYGON ((-118.23795 33.96015, -118.23853 33.9…
1  POLYGON ((-118.23737 33.95852, -118.23738 33.9…
2  POLYGON ((-118.28270 33.96417, -118.28270 33.9…
3  POLYGON ((-118.31160 34.06896, -118.31233 34.0…
4  MULTIPOLYGON (((-118.30285 34.06236, -118.3028…

[5 rows x 45 columns]
```

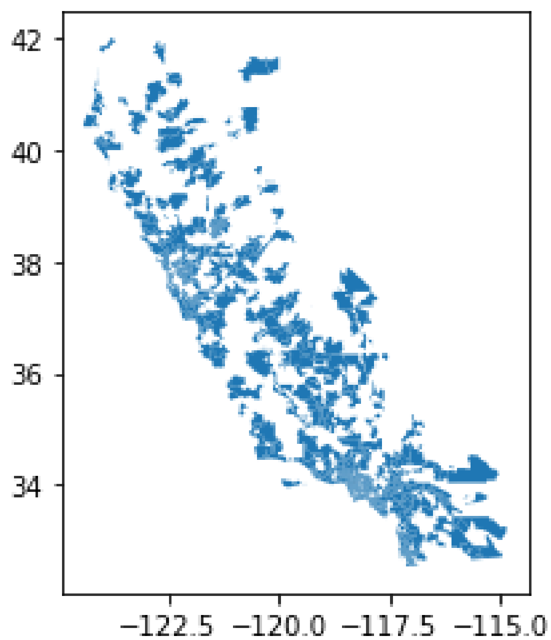
It worked! Now I need to get ready to plot.

[39]:
```
merged = gpd.GeoDataFrame(merged)
```

First I have to change the dataset to a Geo Data Frame so it will plot the zip code boundaries.

[40]:
```
merged.plot()
```

[40]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f058100e520>
```

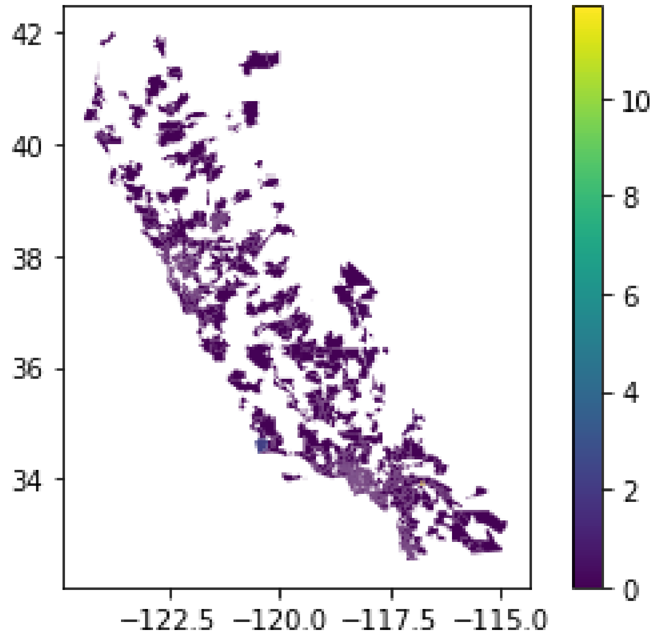


That does look like California (somewhat)! I knew my dataset was incomplete, but it looks like there are a good number of counties here.

To start, let's plot some water bill debt to see if we can identify some trends.

[41]:
```
merged.plot('Percent Deliquent', legend=True)
```

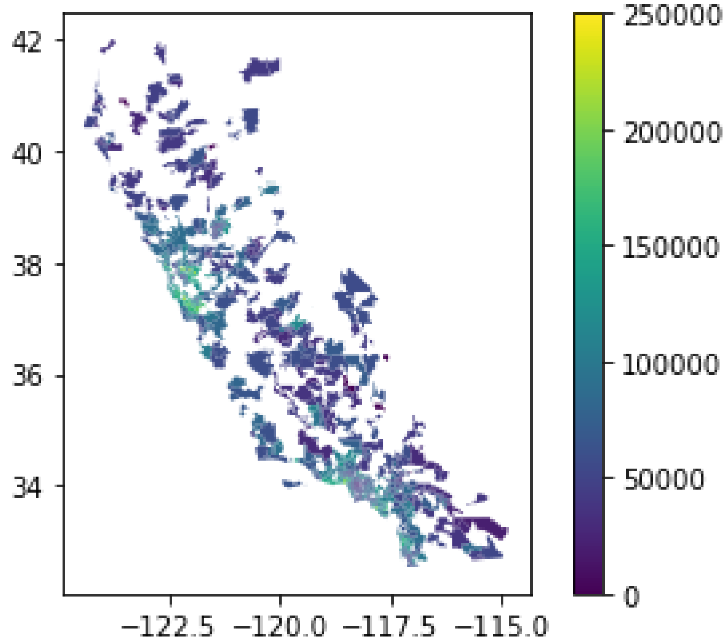
[41]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0580cecf40>



You can see from the above map that there is a bit of variation in the percent deliquent across zip codes, but you cannot really infer much with a static map.

[42]:
```
merged.plot('mhhi', legend=True)
```

[42]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0580c14730>



Again, there's some variation in median household income across zip codes as well, but again, it is difficult to draw any conclusions from a static map.

Next, I am going to use a Kepler map to layer Percent Delinquent, Median Household Income, and Percent Poverty (since those are the scatterplots that had the most obvious trends).

[16]:
```
map = KeplerGl(height=600,width=800)
map
```

```
User Guide: https://docs.kepler.gl/docs/keplergl-jupyter
```

```
KeplerGl(height=600)
```

The next section of code takes you through how I mapped my data.

However, I ran the map and added in the appropriate demographic factors and wanted to make sure my edits and configurations remained the same even as I cleared the kernel and reran the notebook. This is why I saved the map as "water_data_map.py" and added the %run function to ensure my saved file would run.

[43]:
```
%run water_data_map.py

map.add_data(data=merged, name='geometry')

map.config = config
```

You can see that I mapped my merged data based on the "geometry" shapefiles that outline zip code.

You can also see that I added layers for the demographic factors of interest mentioned above.

Next, I saved the file.

[44]:
```
map.save_to_html(file_name='Delinquent and Demographics.html',read_only=True)
```

```
Map saved to Delinquent and Demographics.html!
```

Here the file is now saved as an html.

Then, I saved this configuration (which I ran at the beginning of this section using the %run function) to my Jupyter folder

[45]:
```
config = map.config
with open('water_data_map.py', 'w') as f:
    f.write('{}'.format(config))
```

The scatterplots I created earlier helped me to detect some key trends in the data (notably between Median Household Income and Percent Poverty). With this knowledge, I was able to create a more interactive map so I could hone in on specific zip codes and get a more visual (and interactive) break down of these demographic percentages of interest. You can see from my Kelpler map that it not only color codes the demographic facors statewide, but also shows a break down by zip code of those factors as well - thus making it easier to draw conclusions and compare factors.