Midterm 2_8

February 22, 2021

0.1 Tracking Water Bill Debt by Demographics across California

0.1.1 Load Libraries and Data

First, I will load the libraries

[1]:
```
import pandas as pd
import plotly.express as px
```

Now, I am going to upload the water bill data

[2]:
```
acs = pd.read_csv('Data/Updated Bill Data 2_22.csv')
```

0.1.2 Explore the Data

Now that my data are uploaded, I need to get a sense of how they look.

[3]:
```
acs.shape
```

[3]:
```
(1073, 52)
```

This dataset has 1073 entries and 52 columns - it's big!

I also want to get a sense of the data itsef: missing data, data type, etc.

[4]:
```
acs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1073 entries, 0 to 1072
Data columns (total 52 columns):
 #   Column                                                          Non-Null Count
Dtype
---  ------                                                          --------------
-----
 0   Zip Codes                                                       1073 non-null
object
 1   Count of Zip Code                                               1072 non-null
float64
 2   Sum of Less than $100                                           934 non-null
float64
 3   Sum of $100-$200                                                965 non-null
```

```
float64
 4   Sum of $200-$300                                        965 non-null
float64
 5   Sum of $300-$400                                        954 non-null
float64
 6   Sum of $400-$500                                        941 non-null
float64
 7   Sum of $500-$600                                        917 non-null
float64
 8   Sum of $600-$700                                        914 non-null
float64
 9   Sum of $700-$800                                        906 non-null
float64
 10  Sum of $800-$900                                        895 non-null
float64
 11  Sum of $900-$1000                                       894 non-null
float64
 12  Sum of More than $1000                                  936 non-null
float64
 13  Sum of Total number of delinquent residential accounts  1063 non-null
float64
 14  pop                                                     1030 non-null
float64
 15  nhw                                                     1030 non-null
float64
 16  black                                                   1030 non-null
float64
 17  hisp                                                    1030 non-null
float64
 18  asian                                                   1030 non-null
float64
 19  noncitizen                                              1030 non-null
float64
 20  immigrants                                              1030 non-null
float64
 21  ohu                                                     1030 non-null
float64
 22  lep_hh                                                  1030 non-null
float64
 23  dpov                                                    1030 non-null
float64
 24  npov                                                    1030 non-null
float64
 25  mhhi                                                    1030 non-null
float64
 26  overcrowded                                             1030 non-null
float64
 27  no_veh_hh                                               1030 non-null
```

```
float64
 28  w_broadband                                             1030 non-null
float64
 29  pop_19_64                                               1030 non-null
float64
 30  uninsured_19_64                                         1030 non-null
float64
 31  pct_nhw                                                 1030 non-null
float64
 32  pct_black                                               1030 non-null
float64
 33  pct_hisp                                                1030 non-null
float64
 34  pct_asian                                               1030 non-null
float64
 35  pct_noncitizen                                          1030 non-null
float64
 36  pct_immigrants                                          1030 non-null
float64
 37  pct_lep_hh                                              1029 non-null
float64
 38  pct_povt                                                1029 non-null
float64
 39  pct_overcrowded                                         1029 non-null
float64
 40  pct_no_veh_hh                                           1029 non-null
float64
 41  pct_broadband                                           1029 non-null
float64
 42  pct_no_broadband                                        1029 non-null
float64
 43  pct_uninsured_19_64                                     1030 non-null
float64
 44  aggveh                                                  1030 non-null
float64
 45  pct_no_hins                                             1030 non-null
float64
 46  veh_person                                              1030 non-null
float64
 47  Total Population in Occupied Housing Units: Renter Occupied  1030 non-null
float64
 48  Owner Occupied Pop                                      1030 non-null
float64
 49  % Renter Pop                                            1030 non-null
float64
 50  % Owner Pop                                             1030 non-null
float64
 51  Households                                              1031 non-null
```

```
float64
dtypes: float64(51), object(1)
memory usage: 436.0+ KB
```

So here we can see that there are fewer datapoints here than total zip codes in California. There are over 1,700 zips in the state. This can largely be attributed to the fact that the water bill debt data was conducted via a survey distribted by the California State Water Resources Control Board. Survey responses have their limitations in that they are completed on a voluntary basis. It is worth keeping in mind as I continue with my analysis that this is not a complete dataset of all zip codes in the state. Hopefullly, this dataset is complete enough though to draw some conclusions about water bill debt trends and demographics.

Note that all the data types are floats as well - so I shouldn't have problems conducting quantative analyses.

0.1.3 Cleaning the Data

Next, I will only keep the columns of interest, including debt-related columns, as well as racial/ethnic factors, percent poverty, percent renter, percent immigrant, and median household income. I have a sense that these facors will contribute to making an area more financially vulnerable, and thus, might highlight trends in water bill debt data.

[5]:
```
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',
 'pct_immigrants',
 '% Renter Pop']
```

[6]:
```
acs = acs[refined_columns]
```

Now I have just saved the new data frame and will check my work.

[7]:
```
acs.head()
```

[7]:

```
  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  …  Sum of More than $1000  \
0             137.0             132.0  …                   709.0
1             281.0             231.0  …                  1779.0
2             513.0             482.0  …                  3437.0
3             223.0             206.0  …                  1116.0
4              95.0              95.0  …                   426.0

   Sum of Total number of delinquent residential accounts      pop     mhhi  \
0                                            14902.0       58975.0  38521.0
1                                            10755.0       53111.0  35410.0
2                                            13669.0       72741.0  37226.0
3                                             8589.0       61586.0  48754.0
4                                             5094.0       39479.0  35149.0

   pct_black  pct_hisp  pct_asian  pct_povt  pct_immigrants  % Renter Pop
0   0.088648  0.900144   0.002187  0.287524        0.407630      0.647257
1   0.194950  0.784640   0.006063  0.328603        0.350737      0.621999
2   0.221828  0.770542   0.003533  0.306597        0.375442      0.692044
3   0.038856  0.511139   0.251437  0.180601        0.495437      0.803900
4   0.061374  0.492338   0.350794  0.280593        0.591352      0.912764

[5 rows x 21 columns]
```

Just the cell columns if interest have saved.

0.1.4 Normalize the Data

I need the percentage of the deliquent population each debt "bucket" represents.

I also want to know what percent of the population in each zip code has water bill debt, broadly. To do this, I need to create new columns for each of these columns as a percentage of the deliquent population and total population, respectively.

Further I want to convert all of the % values from decimals to percents so they are clearer on my maps.

[8]:
```python
list(acs)
```

[8]:
```
['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',
 'pct_immigrants',
 '% Renter Pop']
```

[9]:
```python
acs['Percent Delinquent'] = acs['Sum of Total number of delinquent residential␣
↪accounts']/acs['pop']*100


pct_debt_buckets = ['Percent Less than $100', 'Percent $100-$200', 'Percent␣
↪$200-$300' ,
                 'Percent $300-$400', 'Percent $400-$500', 'Percent $500-$600',␣
↪'Percent $600-$700',
                 'Percent $700-$800', 'Percent $800-$900', 'Percent $900-$1000',␣
↪'Percent More than $1000']

debt_buckets =  ['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_total = 'Sum of Total number of delinquent residential accounts'

demographics = ['pct_black', 'pct_hisp', 'pct_asian', 'pct_povt',␣
↪'pct_immigrants', '% Renter Pop']
```

```
for pct, debt in zip(pct_debt_buckets, debt_buckets):
    acs[pct] = acs[debt] / acs[sum_total]*100

for dem in demographics:
    acs[dem] = acs[dem]*100
```

To simplify my code, I divided up much of the repetitive calculations into loops. My standardized buckets were listed together and looped, and I did the same for my demographic factors to multiply them by 100 for standardized percents in my data.

[10]:
```
acs.head()
```

[10]:
```
   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  …  Percent $100-$200  \
0             137.0             132.0  …          33.129781
1             281.0             231.0  …          20.009298
2             513.0             482.0  …          13.753749
3             223.0             206.0  …          19.315403
4              95.0              95.0  …          21.731449

   Percent $200-$300  Percent $300-$400  Percent $400-$500  Percent $500-$600  \
0          10.616025           4.589988           2.650651           1.899074
1           9.781497           6.582985           4.583914           3.579730
2          11.427317           8.552198           6.884191           5.720974
3          13.028292           8.557457           5.844685           4.505763
4          11.739301           7.872006           5.516294           3.435414

   Percent $600-$700  Percent $700-$800  Percent $800-$900  \
0           1.476312           0.919340           0.885787
1           3.189214           2.612738           2.147838
2           4.923550           3.753018           3.526227
3           3.248341           2.596344           2.398417
4           2.866117           1.864939           1.864939
```

```
   Percent $900-$1000  Percent More than $1000
0            0.650919                 4.757751
1            1.868898                16.541144
2            2.933645                25.144488
3            1.909419                12.993364
4            1.138594                 8.362780

[5 rows x 33 columns]
```

Above you'll see a quick check of my data. Next, I only wanted to keep columns of interest.

[11]:
```python
columns_drop = ['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']

acs = acs.drop(columns_drop, axis = 1)
```

The code I ran was a little shorter than listing the columns in their entirety. Then I checked my work.

[12]:
```python
acs.head()
```

[12]:
```
  Zip Codes      pop     mhhi  pct_black  pct_hisp  pct_asian   pct_povt  \
0     90001  58975.0  38521.0   8.864773  90.014413   0.218737  28.752380
1     90002  53111.0  35410.0  19.495020  78.463972   0.606277  32.860252
2     90003  72741.0  37226.0  22.182813  77.054206   0.353308  30.659739
3     90004  61586.0  48754.0   3.885623  51.113890  25.143701  18.060146
4     90005  39479.0  35149.0   6.137440  49.233770  35.079409  28.059292

   pct_immigrants  % Renter Pop  Percent Delinquent  …  Percent $100-$200  \
0       40.763035     64.725731           25.268334  …          33.129781
1       35.073714     62.199921           20.250042  …          20.009298
2       37.544164     69.204438           18.791328  …          13.753749
3       49.543727     80.390024           13.946351  …          19.315403
4       59.135236     91.276375           12.903062  …          21.731449

   Percent $200-$300  Percent $300-$400  Percent $400-$500  Percent $500-$600  \
0          10.616025           4.589988           2.650651           1.899074
```

```
1          9.781497           6.582985           4.583914           3.579730
2         11.427317           8.552198           6.884191           5.720974
3         13.028292           8.557457           5.844685           4.505763
4         11.739301           7.872006           5.516294           3.435414

   Percent $600-$700  Percent $700-$800  Percent $800-$900  \
0           1.476312           0.919340           0.885787
1           3.189214           2.612738           2.147838
2           4.923550           3.753018           3.526227
3           3.248341           2.596344           2.398417
4           2.866117           1.864939           1.864939

   Percent $900-$1000  Percent More than $1000
0            0.650919                 4.757751
1            1.868898                16.541144
2            2.933645                25.144488
3            1.909419                12.993364
4            1.138594                 8.362780

[5 rows x 21 columns]
```

0.1.5 Summary Statistics

Now that my data is cleaned and normalized, I want to get a better sense of how it looks. I'll calculate some summary stats.

[63]:
```python
Columns = ['pop',
 'mhhi',
 'pct_black',
 'pct_hisp',
 'pct_asian',
 'pct_povt',
 'pct_immigrants',
 '% Renter Pop',
 'Percent Delinquent',
 'Percent Less than $100',
 'Percent $100-$200',
 'Percent $200-$300',
 'Percent $300-$400',
 'Percent $400-$500',
 'Percent $500-$600',
 'Percent $600-$700',
 'Percent $700-$800',
 'Percent $800-$900',
 'Percent $900-$1000',
 'Percent More than $1000']
```

```
acs[Columns].describe()
```

[63]:

	pop	mhhi	pct_black	pct_hisp	pct_asian
count	376.000000	376.000000	376.000000	376.000000	376.000000
mean	28184.731383	82000.944149	4.188481	28.370609	13.309808
std	21148.586559	36947.501222	5.912946	22.156196	14.473204
min	202.000000	0.000000	0.000000	0.000000	0.000000
25%	11313.500000	55786.750000	0.814591	11.902809	2.674377
50%	26088.000000	74415.000000	2.005617	20.297812	8.145666
75%	39401.000000	100338.500000	4.819766	40.371188	18.738314
max	99284.000000	226450.000000	43.836772	97.971152	67.856784

	pct_povt	pct_immigrants	% Renter Pop	Percent Delinquent
count	376.000000	376.000000	376.000000	366.000000
mean	13.053572	22.855914	40.826504	0.372288
std	8.330521	13.311688	18.082471	0.298633
min	0.000000	0.000000	0.000000	0.000000
25%	7.162339	12.851028	27.183280	0.095965
50%	11.279916	21.250227	38.228502	0.322388
75%	17.379622	31.288143	53.074624	0.613493
max	66.666667	64.147483	92.803598	0.990099

	Percent Less than $100	Percent $100-$200	Percent $200-$300
count	314.000000	329.000000	327.000000
mean	21.994885	23.360832	17.070812
std	24.347467	21.544359	18.475945
min	0.000000	0.000000	0.000000
25%	0.000000	8.333333	7.310222
50%	16.397849	19.444444	13.636364
75%	32.948537	33.245383	19.667713
max	100.000000	100.000000	100.000000

	Percent $300-$400	Percent $400-$500	Percent $500-$600
count	319.000000	315.000000	303.000000
mean	13.055340	57.238690	5.937663
std	15.218510	844.713460	6.863027
min	0.000000	0.000000	0.000000
25%	5.509189	2.245201	1.316017
50%	9.615385	6.060606	4.382470
75%	15.849282	11.817279	8.021382
max	100.000000	15000.000000	54.545455

	Percent $600-$700	Percent $700-$800	Percent $800-$900
count	300.000000	298.000000	291.000000
mean	5.066759	3.559053	3.232929
std	8.567253	7.339987	9.000086

```
min             0.000000           0.000000           0.000000
25%             0.573367           0.000000           0.000000
50%             3.041869           1.914782           1.665622
75%             6.107955           4.469320           3.341308
max           100.000000         100.000000         100.000000

       Percent $900-$1000  Percent More than $1000
count          288.000000               313.000000
mean             2.596548                15.206207
std              8.837908                23.972368
min              0.000000                 0.000000
25%              0.000000                 1.136364
50%              1.015464                 5.555556
75%              2.376241                18.340611
max            100.000000               100.000000
```

This is a lot of information in one place, but it can help me benchmark averages for debt and dempographics across the state. Thus, in my further analysis of mapping demographis, I have this table as a baseline to see which Zip Codes are above, below the mean and median, and which Zips are towards the lower and upper bounds of the distribution. These stats will help guide my mapping as I determine Zip codes that have demographic factors that might put them at a higher risk of increased deliquent bill debt. Below I plot some of these relationships to better visualize them.

0.1.6 Incorporation of Demographics

I tried to use a for loop for these, but plotly.express did not produce the interactive maps within the function. The code ran, but none of the plots printed into the notebook, for reference, this is what I used:

def dem_scatter(dem): px.scatter(acs, x = dem, y = 'Percent Delinquent')

Demographics = ['mhhi', 'pct_black', 'pct_hisp', 'pct_asian', 'pct_povt', 'pct_immigrants', '% Renter Pop',]

for dem in Demographics: dem_scatter(dem)

Below you can see where I ran the plots by hand.

[24]:
```
acs.drop(acs[acs['Percent Delinquent'] > 1].index, inplace = True)

px.scatter(acs,
           x ='mhhi',
           y = 'Percent Delinquent')
```

There is a very high conentration of zip codes with some degree of water bill debt between 30,000 and 100,000. This is a clear trend in the data, wherein lower income zip codes tend to have more water bill debt than higher income zip codes.

Now let's see if there are trends in the other demographic factors.

[25]:
```
px.scatter(acs,
           x ='pct_povt',
           y = 'Percent Delinquent')
```

Looking at the percentage of poverty, there is not necessarily a clear trend. There are definitely a high conentration of deliquent households in zip codes with poverty levels between 10-20%. However, when we dive deeper into this data, there might be a clerer trend.

The poverty rate in the US is around 12%, so if we condier anything higher than 12% as a high poverty area, there is some indication that there are a high concentration of zip codes with some degree of deliquency on their bills in moderately-high poverty rate zip codes. Of course, there needs to be bit more examination of the data before we make a clearer judgement call.

[26]:
```
px.scatter(acs,
           x ='pct_black',
           y = 'Percent Delinquent')
```

There is a very slight trend here, but not a very obvious one. There are some zip codes with a higher percentage of Black populations that have higher bill debt (as you can see on the right hand side of the scatterplot). However, more analysis is needed.

[27]:
```
px.scatter(acs,
           x ='pct_hisp',
           y = 'Percent Delinquent')
```

It definitely seems like there are many more datapoints with a larger hispanic population than black population. It also does appear that populations with a greater percentage of hispanic populations also have fairly high levels of water bill debt deliqiency (when compared to black populations) I think more analysis should be done, but there seems to be a clearer trend in this data than with the % black population.

[28]:
```
px.scatter(acs,
           x ='pct_asian',
           y = 'Percent Delinquent')
```

With the percent asian population, there is a less clear trend in the data, with a cluster towards the left hand side of the plot. If anything, the percent black, hispanic, and asian plots serve as clear comparisons and contrasts to the diffent ways these groups experience bill debt across the state.

[29]:
```
px.scatter(acs,
           x ='pct_immigrants',
           y = 'Percent Delinquent')
```

There is a fairly clear trend in this scatterplot, and I will incirporate immigrant popualtions into my map in the future.

[30]:
```
px.scatter(acs,
           x ='% Renter Pop',
           y = 'Percent Delinquent')
```

There also appears to be a trend with renter population, and I will incorprate this into my mapping analysis as well.