Exercise: Mapping 2020 Political Contributions in Pennsylvania

Summary

This exercise maps contributions to presidential candidates by party in the 2020 election for zip codes in Pennsylvania.

Input Data

Several input files are available from the class Google Drive folder: **contrib_clean.pkl**, a pickled version of the aggregated individual contribution data from the previous exercise on campaign contributions; **com_cand_info.csv**, another file from the previous assignment that links committees, candidates, and parties; and **cb_2019_42_zcta510_500k.gpkg**, a geopackage file containing two layers: `state`, which is the boundary for Pennsylvania itself, and `zip`, which has boundaries for all of the state's ZCTAs (zip code tabulation areas, or just zip codes below). See the tips section if you're interested in what the `_500k` in the file name means.

Deliverables

There are seven deliverables: a script called **pop.py** that retrieves population data from the Census API server; one called **by_party.py** that aggregates the contribution data to parties rather than candidates (different from what was done in the previous exercise); a script called **join.py** that joins the population and contribution results onto the zip layer in the geopackage file to build a new geopackage file called `"joined.gpkg"`; a QGIS project file called **contrib.qgz**, and three images: **map_party.png**, **map_pc.png** and **map_funds.png**.

Instructions

A. Script pop.py

1. *Get the populations of Pennsylvania zip codes.* Follow the approach used in a couple of the previous exercises to retrieve Census variable `B01001_001E`, the total population, using a `for` clause of `"zip code tabulation area:*"` and an `in` clause of `"state:42"`. This time omit `"NAME"` from the query: it just restates the zip code and is not helpful here.
2. Build a dataframe called `pop` from the server's response.
3. Use a dictionary called `new_names` to rename columns `"B01001_001E"` to `"pop"` (same as the dataframe itself) and `"zip code tabulation area"` to `"zip"`.
4. Set the index of `pop` to `"zip"` and then use `.sort_index()` to sort the dataframe by zip code.
5. Save the dataframe to `"pop.csv"`.

B. Script by_party.py

1. Import modules as needed.
2. Read the contributions data in `"contrib_clean.pkl"` into a variable called `contrib`.
3. Read the committee data in `"com_cand_info.csv"` into a dataframe called `com_cand`.
4. *Join the committee information onto the contributions.* Create a variable called `merged` by using an inner join to merge `com_cand` onto `contrib` using `"CMTE_ID"` as the join key and specifying `validate="m:1"`. Since it's an inner join, omit the indicator variable since all the output records will always be `"both"`. The inner join eliminates a few 2019 contributions to a committee that didn't actually field a candidate in 2020.
5. Use a dictionary to rename `"CAND_PTY_AFFILIATION"` to `"party"`.

6. *Select the Pennsylvania data from the full dataset.* Create a variable called `pa` by using `.query()` to select the rows of `merged` where `"STATE"` is equal to `"PA"`.
7. *Group the Pennsylvania data by zip code and political party.* Create a variable called `grouped` by using `.groupby()` to group `pa` by `"zip"` and `"party"`.
8. *Find total contributions by zip code and party.* Create variable `amount` by applying `.sum()` to column `"amt"` of `grouped`.
9. Some of the amounts are negative because campaigns sometimes repay contributors. We'll use the `.where()` method to set those to zero. Set `amount` to the result of calling the `.where()` method on `amount` with the arguments `amount>0` and `0`. (FAQ 1)
10. *Summarize contributions by zip code.* Set `wide` to the result of calling `.unstack()` on `amount` using the argument `"party"`. The result will be a dataframe with one row per zip code and one column per party.
11. A lot of the values in `wide` will be missing because there aren't contributions to every party in every zip code. Set those to zero by calling `.fillna()` on `wide` with arguments `0` and `inplace=True`.
12. *Find total contributions in each zip code.* Add a column to `wide` called `"total"` that is equal to applying `.sum()` to `wide` with `axis="columns"`.
13. Save `wide` to `"by_party.csv"`. Have a look at it to make sure it is roughly what you'd expect. (However, if you're familiar with Pennsylvania you may notice that some of the zip codes are clearly wrong. Actual Pennsylvania zip codes are in the range 15xxx to 19xxx but some of the original contributions that list Pennsylvania as the state have zip codes outside that range. Those will be removed in a later step.)

C. Script join.py

1. Import the numerical python module `numpy` as `np` and import other modules as needed
2. *Read the population data.* Create variable `pop` by reading `pop.csv`. Use a dictionary to set `dtype` for `"state"` and `"zip"` to `str`. The `dtype` setting is less crucial here than in other contexts because neither the state code nor any of the legitimate zip codes start with leading zero. However, if you work with US data you'll make your life easier if you get in the habit of **always** reading FIPS and zip codes as strings.
3. *Read the contribution data.* Create variable `contrib` by reading `"by_party.csv"`. Set the `dtype` for `"zip"` to `str`.
4. *Merge the contribution data onto the population data.* Create variable `both` by merging `contrib` onto `pop` using an outer one-to-one join with `"zip"` as the join key and `indicator` set to `True`.
5. *Check the merge.* Print the value counts for the merge indicator. Not everything will be `"both"`: expect to see counts for both `"left_only"` (map zip codes that had no contributions) and `"right_only"` (contributions for zip codes that aren't in Pennsylvania).
6. *Check the contributions by merge result.* Create `grouped` by grouping `both` by `"_merge"`. Then create `summary` by applying the `.agg()` method to column `"total"` of `grouped` using the argument `["count","sum","mean"]`. The result will be a dataframe with three columns of aggregate information: the record count for each case of `"_merge"`, and the sum and the mean as well. Note that these are totals by zip code, so the means are much larger than a typical individual contribution. See the Tips section if you receive a FutureWarning message at grouping step.
7. Print `summary`.

8. *Select the usable data.* Now set `trim` to the result of using `.query()` to pick out the records of `both` where `"_merge"` is not equal to `"right_only"` (that is, keep the `"both"` and "`left_only"` records), and then add `.copy()` to the end of the statement to cause Pandas to create a copy of the data rather than a view.
9. Drop `"_merge"` from `trim`.
10. Set the amounts in zip codes with no contributions to 0 by calling `.fillna()` on `trim` with arguments `0` and `inplace=True`.
11. Compute per capita contributions by zip code by setting column `"pc"` of `trim` to the `"total"` column divided by the `"pop"` column.
12. *Ignore per capita results for small counties.* The per capita values are very large in a few zip codes with small populations and large contributions. To focus on larger counties, set the per capita column to numpy's missing data value, `np.nan`, for counties with less than `100` people by setting `trim["pc"]` equal to the result of applying `.where()` to `trim["pc"]` using arguments `trim["pop"]>=100` and `np.nan`.
13. Next create a column called `"d_share"` in `trim` giving the share of total contributions that went to Democratic candidates.
14. Sort `trim` by `"zip"` using `.sort_values()`.
15. Save `trim` as `"join.csv"` using `index=False`. Note that the name is `"join.csv"` to match the name of the script for clarity when looking at the directory in the future.
16. *Read the GIS layer of zip code boundaries.* Now read the zip code layer of geopackage file by setting `geo_zip` to the result of calling `gpd.read_file()` with arguments `"cb_2019_42_zcta510_500k.gpkg"` and `layer="zip"`.
17. *Join the contributions data onto the GIS data.* Set `joined` to the result of doing a left 1:1 join of `trim` onto `geo_zip`. Use `left_on="ZCTA5CE10"` and `right_on="zip"` to set the join keys since the column names are different. Also, set the indicator to `True`. A left join is appropriate because we want to keep all of the zip codes in the shape file and want to discard any zip codes that don't match (the bad zip codes discussed above).
18. Print the value counts of the merge indicator and then drop it.
19. *Save the joined layer.* Write out the joined layer to layer `"zip"` of geopackage `"joined.gpkg"` by calling `.to_file()` on `joined` using arguments `"joined.gpkg"` and `layer="zip"`. Note that if `"joined.gpkg"` already exists, this call will add (or replace) layer `"zip"` but won't affect any other layers. To be sure that you get a clean copy of the file, you may want to remove any existing version of `"joined.gpkg"` before running this part of the script.
20. *Save the state border.* For convenience, now read the state boundary layer from the original geopackage file by setting `geo_state` to the result of using `gpd.read_file()` to read `"cb_2019_42_zcta510_500k.gpkg"` with `layer="state"`. Then use the `.to_file()` method to write it out to `"joined.gpkg"` using `layer="state"`.

D. Maps map_party.png, map_pc.png and map_funds.png

1. Start QGIS and load both layers from `"joined.gpkg"`.
2. *Build a map showing the party balance in contributions.* Start by dragging the state layer to the bottom of the list. Then set its fill color to black and its fill style to "FDiagonal". That will be useful because some of

the zip codes have missing data, which has the effect of creating holes in layers that use graduated styling. The shading will make it easy to tell where that has happened because it will only be visible where the overlying data is missing. If you look carefully, you'll see that happen right away because there are two small parts of the state that have no zip code. See the Tips section for a brief explanation about places with no zip code.

3. Right-click on the zip layer and rename it to `party` for clarity. Then set its style to "Graduated" using `"d_share"` as the value, set the color ramp to `RdBu`, and set the mode to **Pretty Breaks** using 5 classes. This is where the striped state layer really pays off: it makes it easy to tell the difference between missing data (striped) and places where the `d_share` variable leads to the area being white (not striped).
4. The map will a little look better if you make lines showing the zip codes thinner. To do that, click on the colored area in the "Symbol" section of the "Graduated" settings (click on the color, not the drop down arrow). Then click on the "Simple Fill" symbol, scroll down until "Stroke width" is visible, and then click the down arrow next to it a couple of times until "Hairline" appears. Then click "Apply".
5. Export the map as `"map_party.png"`.
6. *Build a map showing per capita contributions.* The party map shows the mix of contributions but it's only part of the story because it doesn't say anything about the intensity of contributions from each zip code. To get at that, we'll look at per capita contributions. Duplicate the `party` layer and rename it `pc`. Then turn off the `party` layer by unchecking its box and turn on the new `pc` layer. Change the variable driving the "Graduated" style to `"pc"`, change the color ramp to "Greens", set the mode to **Natural Breaks**, and use 4 classes. If all goes well, you should see that per capita contributions are actually very small for most of the state. However, they are very high for some zip codes near Philadelphia (on the east), and fairly high for some near Pittsburgh (on the west) and at a few other places scattered around the state.
7. Export the map as an image to `"map_pc.png"`.
8. *Build a map showing overall funding by party and zip code.* The maps above show the preferences of people in different zip codes but they don't convey overall fundraising. To get at that, we'll build a map with pie charts indicating total funding by party from each zip code. Duplicate the `pc` layer and rename it `funds`. Then turn off `pc` and turn on `funds`. Set the style for `funds` to "Single Symbol" and set the fill color to something pretty light. Then add pie diagrams. For the pie attributes, choose `"DEM"` and `"REP"` and set the colors to blue and red. Then set the size of the pies to "Scaled Size" using the `"total"` attribute for scaling. Click "Find" to find the maximum value of `"total"`, and then set the "Size" to 15. Make sure that the "Size Units" selector at the top of the dialog is set to millimeters.
9. Export the map to `"map_funds.png`".
10. Save the QGIS project file as `"contrib.qgz`".

Submitting

Once you're happy with everything and have committed all of the changes to your local repository, please push the changes to GitHub. At that point, you're done: you have submitted your answer.

Tips

- The cartographic boundary files are lower resolution, and hence smaller in file size, than TIGER/Line files. They are very suitable for thematic maps (such as the one in this exercise) where precise distance measurements are not needed. The files are available at several scales: the 500k here indicates that the file is at a scale of 1:500,000, which is the highest resolution among the cartographic boundary options. The US zip code map is about 60 MB at this resolution; for comparison, the TIGER/Line version is about nine times larger: about 530 MB.

- Some areas of the United States do not have ZCTAs. They are usually rural areas with very low population density. There are a couple of spots in Pennsylvania without ZCTAs. For comparison, there are considerably more areas without them in New York, largely in the Adirondack region, and vastly more in the western US.
- This exercise is just scratching the surface of what could be done to analyze the contributions data. Many other variables could be added as well, including median income, race, education, employment status, and so on.

FAQs

1. *How does the `.where()` method work?*

   It's a little counterintuitive, at least at first. When called like this, `B = A.where(test,C)`, it says that each element of `B` should be equal to the corresponding element of `A` when the corresponding element of `test` is True; otherwise, the value of `B` should be set to `C` instead of what was in `A`.

2. *What causes the FutureWarning about observed=False?*

   This is due to an internal bug in pandas as of about version 2.1.4. There's nothing you can do about it and it can be ignored.