Exercise: Computing County Dissimilarity Indexes

Summary

This exercise uses Census data at the block group level to compute a measure of racial segregation known as a dissimilarity index for large counties in the US. It also demonstrates how to work with zip archives containing multiple files.

Input Data

There are two input files that will need to be downloaded from the course Google Drive folder: **bg_by_state.zip** and **county_names.csv**. Please note that `bg_by_state.zip` should not be unzipped: you'll read it directly using Pandas.

The first file, `bg_by_state.zip`, contains 52 individual CSV files of block group data: one for each state plus the District of Columbia and Puerto Rico. The files have names like "bg36.csv", where the digits indicate the FIPS code of the state. Each of the files has six columns: "B02001_001E", "B02001_002E", "state", "county", "tract", and "block group". The first two are the total population of the block group ("B02001_001E") and the population of people in the block group who identify as white alone ("B02001_002E"). The remaining columns are all components of the FIPS code identifying the block group. If you'd like to see what the files look like without unpacking the zip archive, you can download the New York file, **bg36.csv**, from the Google Drive folder.

The second file, `county_names.csv`, contains the names of US counties. It has three columns: "state", "county", and "NAME". The first two are FIPS codes.

Deliverables

The deliverables for the assignment are two scripts, **append.py** and **dissim.py**. They produce two output files, "append.csv" and "dissim.csv", and one figure, "pop_by_bin.png". Note that the CSV files are not deliverables and will not be uploaded to GitHub: they will be rebuilt when your script is run.

Instructions

Please note that some instructions for common operations are brief because they're probably becoming pretty familiar. However, they may have additional text in an FAQ section at the bottom in case it's useful.

A. Script append.py

1. Import `pandas` as `pd`.
2. Import `zipfile`.
3. Open the zip archive by setting variable `ziparc` to the result of calling `zipfile.ZipFile()` with the argument `"bg_by_state.zip"`. The `ziparc` variable will be a ZipFile object with a range of helpful methods for working with the archive.
4. Create a dictionary called `fips` that will be used to make sure the following four FIPS codes are read as strings: `"state"`, `"county"`, `"tract"`, and `"block group"`.
5. Create an empty list called `files`. It will be used to hold the input data for each state as it is read.
6. Use variable `f` to loop over the result of calling the `.namelist()` method on `ziparc`. The `.namelist()` method iterates over the names of the files in the zip archive. Within the loop do the following:
    1. Open a file handle for reading file `f` by setting variable `fh` to the result of calling the `.open()` method on `ziparc` with argument `f`.

   2. Create dataframe `cur` by calling `pd.read_csv()` with arguments `fh` and `dtype=fips`.

   3. Print a message giving the name of the file just read (`f`) and the number of block groups it contained (the length of `cur`).

   4. Append the new dataframe to the `files` list using the `.append()` method with the argument `cur`.

7. Now build a single dataframe called `combined` by setting `combined` equal to the result of calling `pd.concat()` with the argument `files` (that is, the list of dataframes).

8. Set variable `n_files` equal to the length of `files`.

9. Set variable `n_bgs` equal to the length of `combined`.

10. Print a message giving the two values just computed: the number of files read and the total number of block groups found.

11. The tract code (6 digits) and the block group code (1 digit) together provide a unique code for the block group within the county. It will be convenient to replace the two of them with the combined code. Create a new column `"bg"` in `combined` that is equal to the result of concatenating the `"tract"` and `"block group"` columns of `combined`. (FAQ1)

12. Drop the `"tract"` and `"block group"` columns from `combined`. (FAQ2)

13. Create a dictionary called `varmap` and use it to rename the Census variables in `combined`. Rename `"B02001_001E"` to `"total"` and `"B02001_002E"` to `"white"`.

14. Set column `"poc"` of `combined` equal to the difference between the `"total"` and `"white"` columns.

15. Save the result to `"append.csv"` by calling the `.to_csv()` method on `combined` with the argument `"append.csv"` and `index=False`. The `index` keyword omits the index from the output file. That's useful in this context because here the index is just the row number and we don't need to retain it.

B. Script dissim.py

1. Import `pandas` and `matplotlib.pyplot`.

2. Set the default DPI for plots to 300.

3. Create a dictionary called `fips` to keep the `"state"`, `"county"`, and `"bg"` FIPS codes as strings when `"append.csv"` is read.

4. Set `dat` to the result of calling `pd.read_csv()` with arguments `"append.csv"` and `dtype=fips`.

5. Set the index of `dat` to a list consisting of `"state"`, `"county"`, and `"bg"`.

6. Set `by_co` to the result of grouping `dat` by `"state"` and `"county"`.

7. Compute each county's total population in each racial group by setting `by_co_tot` to the result of calling `.sum()` on `by_co`.

8. Keep a list of the racial groups in the data by setting `races` equal to the names of the columns of `by_co_tot`. It will be handy later in the script.

9. Compute each block group's share of the county's total population of each racial group by setting `shr` equal to `dat` divided by `by_co_tot`.

10. Check the calculation by setting `check` equal to the result of grouping `shr` by `"state"` and `"county"` and then calling `.sum()` on the result.
11. Print a random sample of the results for 20 counties by printing the outcome of calling the `.sample()` method on `check` with argument `20`. If all has gone well, you should get 20 rows of 1's.
12. Calculate each block group's absolute difference between the population shares of white people and people of color by setting `abs_diff` equal to the `abs()` function called on the difference between the `"white"` and `"poc"` columns of `shr`.
13. Set `sum_by_co` to the result of grouping `abs_diff` by `"state"` and `"county"` and then applying `.sum()`.
14. Calculate the dissimilarity index for each county by setting `dissim` equal to `100` times `0.5` times `sum_by_co`. The result will be dissimilarity index values measured in percentages.
15. Now we'll build a dataframe of information by county. Start by setting `all_co_results` equal to the result of calling `.copy()` on `by_co_tot`. That will copy the county populations into `all_co_results`.
16. Next, store the number of block groups in each county by setting column `"num_bg"` of `all_co_results` to the result of calling `.size()` of `by_co`.
17. Now store a rounded version of the dissimilarity index by setting column `"dissim"` of `all_co_results` to the result of calling `.round(2)` of `dissim`.
18. Compute and print the total population by race in millions by setting `tot_pop` to the result of calling `.sum()` on `all_co_results[races]` and dividing the result by `1e6`. Then print `tot_pop`.
19. Now filter down the counties to those that have at least 50 block groups and at least 10,000 residents of color. Do that by setting `large_co_results` to the result of calling `.query()` on `all_co_results` with the argument `"num_bg >= 50 and poc >= 10000"`. Note that the argument is a string and column names within the string are NOT quoted.
20. Compute the population in the filtered data by setting `large_pop` equal to the result of summing `large_co_results[races]` and then dividing by `1e6`. Then print `large_pop`.
21. In addition, print the large county populations as shares of the national totals by printing `100` times `large_pop` divided by `tot_pop`. This step is important to verify that we haven't filtered out too much of the total population or too many people of color. Here you should see that about 78% of the total population and 89% of the people of color live in large counties.
22. Next we'll merge on the county names. As a first step, create dataframe `names` by calling `pd.read_csv()` on `"county_names.csv"` using `dtype=str`.
23. Now join the names onto the data by setting `res` equal to the result of calling `.merge()` on `large_co_results` using the following arguments: `names`, `on=["state", "county"]`, `how="left"`, `validate="1:1"`, and `indicator=True`. It's a left join because we want to keep all of the records in `large_co_results` and don't want to include any of the records that are in `names` but not in `large_co_results`, since those are for small counties.
24. Print the value counts for the `"_merge"` column of `res` and then drop it from the dataframe. (FAQ3, FAQ2)
25. Sort `res` by `"dissim"`.

26. Save the results by calling `.to_csv()` on `res` with arguments `"dissim.csv"` using `index=False`.
27. Now have a look at the results for some counties in New York by setting `nys` to the result of calling `.query()` on `res` with argument `"state == '36'"`. Notice that the state FIPS code MUST be quoted since it's a string: writing `"state == 36"` will not work.
28. Print `nys` and look to see where Onondaga County falls.
29. Next, set up bins for the dissimilarity index by setting the `"bin"` column of `res` to the result of calling `.round(-1)` on the `"dissim"` column.
30. Set `by_bin` equal to the result of grouping `res` by `"bin"`.
31. Set `pop_by_bin` to the result of applying the `.sum()` method to `by_bin[races]` and then dividing by `1e6`. The `[races]` selector picks out the population columns and is needed to avoid summing the names, FIPS codes, and so on.
32. Calculate the percentage of each racial group in each bin by setting `pct_by_bin` equal to `100` times `pop_by_bin` divided by `large_pop`.
33. Print an informative heading and then print `pop_by_bin`.
34. Print an informative heading and then print `pct_by_bin`.
35. Begin a new figure by setting `fig1, ax1` to the result of calling `plt.subplots()`.
36. Set `bars` to a list consisting of `"white"` and `"poc"`. It will define the columns in a bar graph below.
37. Call `.plot.bar()` on `pct_by_bin[bars]` using the argument `ax=ax1`.
38. Set the figure title by calling `.suptitle()` on `fig1` with the argument `"Degree of Segregation in Large US Countie`
39. Set the X axis label by calling `.set_xlabel()` on `ax1` with the argument `"Dissimilarity Index"`.
40. Set the Y axis label by calling `.set_ylabel()` on `ax1` with the argument `"Percent of Overall Population"`.
41. Adjust the figure's spacing by calling `.tight_layout()` on `fig1`.
42. Save the figure by calling `.savefig()` on `fig1` with arguments `"pop_by_bin.png"`.

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.

FAQ

1. *How do I concatenate the strings in two Pandas Series?*

   To concatenate the strings in dataframe `D` columns `D["X"]` and `D["Y"]` into a new column `D["Z"]`, use `D["Z"] = D["X"] + D["Y"]`. Each element of `D["Z"]` will be equal to the result of concatenating the corresponding elements of `D["X"]` and `D["Y"]`.

2. *How do I drop one or more columns from a dataframe?*

   To drop a single column `"C"` from dataframe `D` use `D = D.drop(columns="C")`. To drop several columns, use a list: `D = D.drop(columns=["E","F"])`.

3. *How do I print the value counts of `"_merge"`?*

   For dataframe `D` print `D["_merge"].value_counts()`.