Exercise: Joining Geographic and Census Data

Summary

This exercise carries out a basic left join (a type of merge) of two datasets: one extracted from a geographic information system (GIS) file that gives some basic physical data on US counties, and the other obtained from the US Census that gives each county's population. It shows the internal details of joins that we'll use a lot during the GIS portion of the course later in the semester.

Input Data

The first input file is **county_geo.csv**. It has five fields: `STATEFP`, `COUNTYFP`, `GEOID`, `ALAND`, and `AWATER`. The first two, `STATEFP` and `COUNTYFP`, are the state FIPS code (2 digits) and the county FIPS code within the state (3 digits). The third, `GEOID`, is a 5-digit FIPS code that uniquely identifies the county within the US. It consists the state and county codes concatenated together. The last two variables are the areas of land and water in the county in square meters.

The second input file is **county_pop.csv**. It has four fields: `NAME`, `B01001_001E`, `state`, `county`. The first is the name of the county, including its state. The second, `B01001_001E`, is the Census variable giving the total population of the county. The third and fourth fields give the FIPS codes for the state and the county within the state.

Deliverables

A script called **county_merge.py** that joins the population data onto the geographic data, calculates each county's percentage of its state's total population, and writes out the result as a new CSV file. Although it's not part of this assignment, the output file could be imported into GIS software for mapping.

Instructions

Please prepare a script called `county_merge.py` as described below. See the accompanying file `notation.pdf` for brief examples showing how some common instructions in this exercise should be translated into code.

1. Import `csv`
2. Define a function for reading the input data called `read_file()`. It should take a single argument, `filename`. Within the function do the following:
    1. Create an empty list called `records`.
    2. Open `filename` for reading using `fh` as the file handle.
    3. Set `reader` to the result of calling `csv.DictReader()` on `fh`.
    4. Use variable `rec` to loop over `reader`. Within the loop append `rec` to `records`.
    5. After the end of the loop, close `fh` and return `records`.
3. Following the function, set `geo_list` to the result of calling `read_file()` on `county_geo.csv`.
4. Create an empty dictionary called `geo_data`. This will become an indexed version of the geographic data organized by each county's unique 5-digit FIPS code.
5. Use `geo_rec` to loop over `geo_list`. Within the loop do the following:
    1. Set `this_geoid` to the value of `"GEOID"` in the `geo_rec` dictionary.
    2. Set the value of `this_geoid` in the new `geo_data` dictionary to `geo_rec`. This stores the full geographic record for this county in the dictionary under the county's unique geoid.
6. Following the loop, set `pop_list` to the result of calling `read_file()` on `county_pop.csv`.

7. Create an empty dictionary called `pop_data`. This will become an indexed version of the population data. Like `geo_data`, it will be organized by each county's unique FIPS code. However, it's a little more involved since the unique geoid isn't provided in the input data and must be built along the way.
8. Use `pop_rec` to loop over `pop_list`. Within the loop do the following:
    1. Set `fips_state` to the value of `state` in the `pop_rec` dictionary.
    2. Set `fips_county` to the value of `county` in `pop_rec`.
    3. Set `this_geoid` to the result of concatenating `fips_state` and `fips_county`. That produces the county's 5-digit FIPS code.
    4. Set the value of `this_geoid` in the new `pop_data` dictionary to `pop_rec`. This stores the full population record for this county in the dictionary under the county's unique geoid.
9. Following the loop, join the population data onto the geographic data. To do that, use `geoid` to loop over the keys of `geo_data`. Within the loop do the following:
    1. Set `geo_rec` to the value of `geoid` in `geo_data`. That looks up the geographic data for the current value of `geoid`.
    2. Set `pop_rec` to the value of `geoid` in `pop_data`. That looks up the population data for the current value of `geoid`.
    3. Now use `k` to loop over the keys of `pop_rec`. Within the loop, set the value of `k` in `geo_rec` to the value of `k` in `pop_rec`. The effect will be to copy all of the entries in the current population record to the current geographic record. The population data will then have been merged onto the geographic data.
    4. Finally, at the end of the loop set the value of `"pop"` in `geo_rec` to the value of `float()` called on the value of `"B01001_001E"` in `geo_rec`. That will create a numeric version with a more convenient name for use in the next step.
10. After the loop, aggregate the population data to the state level as follows. First, create an empty dictionary called `state_total`. It will be used to sum up the populations within each state. Then use `rec` to loop over the values of `geo_data`. Within the loop do the following:
    1. Set `state` to the value of `"STATEFP"` in the `rec` dictionary.
    2. Set `pop` to the value of `"pop"` in `rec`.
    3. Use an `if` statement to test whether `state` is already in `state_total`. If it is, update `state_total[state]` by adding `pop` to it. Otherwise, set `state_total[state]` to `pop`.
11. Now compute each county's share in the state's total population and report the results as percentages. Start by using `rec` to loop over the values of `geo_data`. Within the loop do the following:
    1. Set `state` to the value of `"STATEFP"` in `rec`. That identifies the state for the current county record.
    2. Set `state_pop` to the value of `state` in `state_total`. That looks up the state population for the current county's state.
    3. Set `pct` to 100 times the value of `rec["pop"]` divided by `state_pop`.
    4. Set the value of `"percent"` in `rec` to `pct`. The percentage is now merged into the county dataset.
12. Finally, write out the joined dataset as follows. First, set `geoids` to the result of calling `sorted()` on the keys of `geo_data`. That will cause `geoids` to be an ordered list of all of the 5-digit county FIPS codes in the joined dataset.
13. Next, build a list of fields in the database. First, pick out the record for one of the counties by setting `onondaga` to the value of `"36067"` in `geo_data` (36067 is the FIPS code for Onondaga County). Remember that the code should be a string.

14. Set `fields` to the keys of `onondaga`. This will be used when calling `DictWriter()` to indicate which fields should be written to the output file, and the order in which they should appear. Just FYI, any county would work since all of the entries have the same fields.
15. Open `county_merge.csv` for writing using file handle `fh`. Include additional argument `newline=""` (two quotes with no space between them) in the call to `open()`. Without that, the output file will have extra newlines at the end of each line, which will cause it to be double spaced.
16. Set `writer` to the result of calling `csv.DictWriter()` on `fh` and `fields`.
17. Call `writer.writeheader()` to write the field names to the first line of the output file.
18. Use `geoid` to loop over `geoids`. Within the loop call `writer.writerow()` on the value of `geoid` in `geo_data`.
19. Call `fh.close()`.

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

- It's not required but you might find it interesting to open up the output file and look at the counties that have the largest and smallest percentages of their state's population. Eight counties have 40% or more of their state's population and 28 counties have less than 0.01%.