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 (the same information as `STATEFP` and `COUNTYFP` in the other file).

Deliverables

There are two deliverables. The first is a script called **county_merge.py** that joins the population data onto the geographic data, calculates several variables, including 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.

The second deliverable is an updated version of the **results.md** file that contains answers to several questions.

Instructions

Please prepare a script called `county_merge.py` as described below.

1. Import `pandas` as `pd`
2. Set `geo_data` to the result of calling `pd.read_csv()` on `"county_geo.csv"` using `dtype=str`. See the tips section for why `dtype=str` is useful.
3. Set `pop_data` to the result of calling `pd.read_csv()` on `"county_pop.csv"` using `dtype=str`.
4. Now we'll do the left join of `pop_data` onto `geo_data` using the state and county FIPS codes. It's a little involved because the columns with the FIPS codes have different names in the two files.

   Create a new variable called `merged` that is equal to the result of calling `geo_data.merge()` with the following arguments: `pop_data`, `left_on=["STATEFP","COUNTYFP"]`, `right_on=["state","county"]`, `how="left"`.
5. Set the index of `merged` by setting `merged` to the result of calling `merged.set_index()` with the Python list `["STATEFP","COUNTYFP"]` as the call's argument.
6. Now create a new column called `"sq_km"` in `merged` that is equal to the result of calling the `.astype(float)` on `merged["ALAND"]` and then dividing by `1e6` to convert square meters to square kilometers.
7. Create a new column called `"pop"` in `merged` that is equal to the result of calling the `.astype(float)` method on `merged["B01001_001E"]`.
8. Create a new column called `"density"` in `merged` that is the result of dividing the `"pop"` column of `merged` by the `"sq_km"` column.

9. Now we'll compute the total population by state. Start by creating a new variable `group_by_state` equal to the result of calling `merged.groupby()` using `"STATEFP"` as the argument.
10. Create new variable `state_pop` by calling the `.sum()` method on `group_by_state["pop"]`.
11. Print `state_pop` and before proceeding further, verify that the first line is state 01 with a population of 4864680. State 01 is Alabama.
12. Now we'll compute each county's share in the state's total population and report the results as percentages. Create a new column called `"percent"` in `merged` that is equal to 100 times the value of `merged["pop"]` divided by `state_pop`. Pandas does all the heavy lifting to ensure that each county is correctly matched with its state's population by using `"STATEFP"` in each index to line up the data.
13. Now we'll print out information about the 5 counties with the largest values of key variables. Use a `for` loop with loop variable `var` to loop over the list '["sq_km","pop","density","percent"]. Within the loop do the following:
    1. Create a variable called `sort_by_var` that is equal to the result of calling the `merged.sort_values()` with argument `var`.
    2. Create a variable called `last5` that is equal to the result of using `[-5:]` to pick out the last 5 rows of `sort_by_var`. Those will be the rows with the largest values of `var`.
    3. Create a variable called `selected` that is equal to the `"NAME"` and `var` columns of `last5`. See the tips section if you're not sure how to do this.
    4. Print `selected`. You may want to include a `"\n"` as the first argument to `print()` for readability,.
14. Sort the data by FIPS code by setting `merged` equal to the result of calling `merged.sort_index()`. No argument is needed: Pandas will automatically sort the data by `"STATEFP"` and then by `"COUNTYFP"` since that's what's in the dataframe's index.
15. Call the `.to_csv()` method of `merged` to write the data out to a file called `"county_merge.csv"`.
16. Finally, update the `results.md` file to replace the TBD place holders with the answers to the questions it contains. Please try to leave the rest of the Markdown intact (i.e., the indenting and the * that cause each answer to be indented and bulleted.)

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

1. Using `dtype=str` is handy when dealing with files where only a few variables are going to be used in computations. It saves Pandas from having to convert all the data to numeric form. That's especially handy when some of the columns contain FIPS codes, since they look like numbers to Pandas but must be kept as strings to preserve leading 0's.
2. An easy way to select a subset of columns from a Pandas dataframe is to use a list of the names you want:

   ```
   varlist = ["alpha","omega"]
   selected = original[varlist]
   ```

   To do the selection in a single line, use **two** levels of square brackets. The outer level indicates that it's a column selection and the inner level groups the names into a Python list:

   ```
   selected = original[["alpha","omega"]]
   ```