Exercise: Joining Geographic and Census Data

Summary

This exercise carries out a basic left join 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. It also illustrates how to use a loop to produce plots of several variables quickly.

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 is one deliverable: 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, writes out the result as a new CSV file, and draws four graphs. 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.

A. Initial setup

1. Import `pandas` as `pd` and `matplotlib.pyplot` as `plt`.
2. As in previous exercises, use `plt.rcParams` to set the default resolution of plots to `300`. Future exercises will be more terse and simply say to set the default resolution to 300.
3. Create a list called `fips_list` that consists of the following five elements: `"STATEFP"`,`"COUNTYFP"`, `"GEOID"`, `"state"` and `"county"`.
4. Set variable `fips_cols` to the result of using a dictionary comprehension to build a dictionary with the elements of `fips_list` as keys and `str` as each entry's value. This will be used with `.read_csv()` to keep the FIPS codes as strings.
5. Set `geo_data` to the result of calling `pd.read_csv()` on `"county_geo.csv"` using `dtype=fips_cols`.
6. Set `pop_data` to the result of calling `pd.read_csv()` on `"county_pop.csv"` using `dtype=fips_cols`.

B. Checking which state codes are present

We're eventually going to do an left join on the two files, which will keep exactly the states that are present in left file. To see how the data will match up, we'll have a quick look at the states in each file.

1. Build a list of state codes in `geo_data` by setting `geo_states` equal to the result of calling the `set()` function with the `"STATEFP"` column of `geo_data` as the function's argument.

2. Use a similar statement to build a variable called `pop_states` that is the set of states in the `"state"` column of `pop_data`.
3. Print an informative heading and then print the result of `geo_states - pop_states`. The result is a set difference: all the entries in `geo_states` less any entries that are also in `pop_states`. If all goes well, this should produce an empty set, which will print as "set()". That's good: it means that there's a population entry for every geography record. Note that this method doesn't report entries that are in `pop_states` but not in `geo_states`: that will be computed next.
4. Print another informative heading and then print the result of `pop_states - geo_states` as the argument. The result will be all the entries in `pop_states` that are not in `geo_states`. This won't be the empty set because the population file includes Puerto Rico, which is not in this particular geography file. It will be dropped in the join.

C. Joining the data

Now we'll do the left join of `pop_data` onto `geo_data` using the state and county FIPS codes.

1. Create a new variable called `merged` that is equal to the result of calling `.merge()` on `geo_data` with the following arguments: `pop_data`, `left_on=["STATEFP","COUNTYFP"]`, `right_on=["state","county"]`, `how="left"`. The left database is `geo_data` and the right database is `pop_data`.
2. Set the index of `merged` by setting `merged` to the result of calling `.set_index()` on `merged` with the Python list `["STATEFP","COUNTYFP"]` as the call's argument. Future exercises will be more terse and simply say to set the dataframe's index to a specific column or list of columns.
3. Using an approach like that in earlier exercises, use the `.rename()` method of `merged` to rename column `"B01001_001E"` to `"pop"`.
4. Now create a new column called `"sq_km"` in `merged` that is equal to the result of dividing `merged["ALAND"]` by `1e6` to convert square meters to square kilometers.
5. Create a new column called `"density"` in `merged` that is the result of dividing the `"pop"` column of `merged` by the `"sq_km"` column.
6. Create a new column called `"pop_mil"` in `merged` that is the result of dividing the `"pop"` column by `1e6`.
7. Now we'll compute the total population by state. Start by creating a new variable `group_by_state` equal to the result of calling using `.groupby()` to group `merged` by `"STATEFP"`.
8. Create new variable `state_pop` by calling the `.sum()` method on the `"pop"` column of `group_by_state`.
9. 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`. As in earlier assignments, 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.
10. Sort the data by FIPS code by setting `merged` equal to the result of calling the `.sort_index()` method of `merged`. 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.
11. Call the `.to_csv()` method of `merged` to write the data out to a file called `"county_merge.csv"`.

D. Quick analysis of several variables

Now we'll do some quick analysis and plotting of some of the results. We'll do it with a loop to show how to analyze a list of variables in a quick way that scales well.

1. Create a list called `plot_info` that consists of tuples that will describe the graphs to be drawn. Each tuple will have three elements: a column name to be plotted, the units of that column, and some information for the graph's title. The role of the elements will become clearer when the graphs are set up in

subsequent steps. For `plot_info`, the first tuple should be `("sq_km", "Square km", "area")`, the second should be `("pop_mil", "Millions of people", "population")`, the third should be `("density", "People per square km", "density")`, and the last one should be `("percent", "Percent", "share of state population")`. The code will be nicest if there's one tuple per line, with the closing `]` of the list at the very end.

2. Set variable `nfig` equal to 1. This will be used as a figure number.
3. Set up a `for` loop to loop through the elements in `plot_info`. Use the tuple `var,units,ftitle` for the loop variable. At each trip through the loop, `var`, `units`, and `ftitle` will be set to the elements of the corresponding tuple in `plot_info`.
    1. Create a variable called `sort_by_var` that is equal to the result of calling the `.sort_values()` of `merged` with argument `var`.
    2. Create a variable called `last10` that is equal to the result of using `[-10:]` to pick out the last 10 rows of `sort_by_var`. Those will be the rows with the largest values of `var`.
    3. Set `fig,ax` equal to the result of calling `plt.subplots()`.
    4. Set the overall figure title by calling the `.suptitle()` method of `fig` with the argument `f"Figure {nfig}: Top 10 counties for {ftitle}"`.
    5. Add a horizontal bar graph to the Axes object by calling the `.plot.barh()` method of `last10` with four arguments: `x="NAME",y=var,ax=ax,legend=None`.
    6. Turn off the Y axis label by calling the `.set_ylabel()` method of `ax` with argument `None`.
    7. Set the X axis label to the appropriate units (passed in via the tuple) by calling the `.set_xlabel()` method of `ax` with argument `units`.
    8. As in previous exercises, use the `.tight_layout()` to finalize the figure's layout.
    9. Use the `.savefig()` method of `fig` to save the figure. Use the following string as the argument: `f"fig{nfig}_{var}.png"`. That will save the image with both the figure number and variable in the filename, which is handy when referring to files later.
    10. Increment the figure count by adding 1 to `nfig`.

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. Matplotlib supports SVG (scalable vector graphics) as an output format in `.savefig()`. SVG is a great format to use for the web or publication quality graphics because it doesn't degrade when scaled up or down. You should consider using it for images in the repository for your project. To save an image in SVG format just use a file name with `".svg"` as the extension: `"some_name.svg"`. However, some common image browsing tools don't support it so we'll generally stick with PNG for class assignments.