Exercise: Residential Demographics Near Interstates

Summary

Highways are a significant source of air pollution and create health risks for people living in the area. That can raise environmental justice concerns if nearby residents aren't representative of the broader community. This exercise examines that issue by estimating racial differences in residential proximity to interstate highways in Onondaga County.

Input Data

There are two input files: **near-parcels.gpkg**, the output geopackage file from the previous exercise, which will be needed for the `"rings"` layer, and **tl_2020_36_bg.zip**, a shapefile of Census block groups for New York State. The shapefile will need to be downloaded from the Census. Be sure to get the 2020 version for consistency with data that will be downloaded from the API server. If you had trouble with the last exercise, you can download a copy of "near-parcels.gpkg" from the Google Drive folder for this exercise.

Deliverables

There are four deliverables: two scripts, **pop_by_bg.py** and **realloc.py**, one file of results, **realloc.csv**, and one figure, **char_by_ring.png**.

Instructions

Script pop_by_bg.py

1. Import `requests`, `pandas` and `numpy` in their usual ways.
2. Set up a dictionary called `variables` containing the following five key:value pairs: `'B02001_001E':'pop_total','B02001_00`
   It will be a convenient way to rename the variables later on.
3. Set `var_list` to the result of calling the `.keys()` method on `variables`.
4. Set `var_string` to the result of using `.join()` to join the elements of `var_list` with commas.
5. Set `api` to `"https://api.census.gov/data/2020/acs/acs5"`, the endpoint for the ACS 5-year results for 2020. *This differs from previous exercises, which used an earlier year*.
6. Set up an appropriate payload and call the Census API. Use `var_string` as the `get` clause, `"block group:*"` as the `for_clause`, and `"state:36 county:067"` as the `in_clause`.
7. Use the response from the server to construct a dataframe called `pop`.
8. Set `pop` to the result of calling `.replace()` on `pop` using two arguments: `"-666666666"` and `np.nan`). That value is used by the API to indicate missing data, and it has nine 6s.
9. Use the dictionary of variable names created earlier to rename the columns of `pop` by setting `pop` to the value of using the `.rename()` method of `pop` with the argument `columns=variables`
10. Create a new column in `pop` called `"GEOID"` that is the result of concatenating the columns for `"state"`, `"county"`, `"tract"` and `"block group"`. There should be no spaces between the parts.
11. Set the index of `pop` to `"GEOID"`.
12. Create a variable called `keep_cols` that is equal to the result of calling the `.values()` method on `variables`.
13. Trim down `pop` by setting it to the result of using `keep_cols` to select the desired columns from `pop`. The main effect is to discard the component parts of the GEOID.
14. Save `pop` to `"pop_by_bg.csv"`. Do not use `index=False` since the index is meaningful and will be needed later.

Script near_road.py

1. Import `geopandas`, `pandas`, `matplotlib` and `seaborn` in their usual ways.
2. Set variable `rings` to the result of using `geopandas` to read layer `"rings"` from `"near-parcels.gpkg"`.
3. Set `pop` to the result of using `pandas` to read `"pop_by_bg.csv"`. Be sure to use the `dtype` argument with an appropriate dictionary so `"GEOID"` will be read as a string.
4. Set the index of `pop` to `"GEOID"`.
5. Set column `"pop_poc"` in `pop` to the difference between the `"pop_total"` and `"pop_white"` columns.
6. Set `bgs` to the result of using `geopandas` to read `"tl_2020_36_bg.zip"`.
7. Filter `bgs` down to Onondaga County by setting `bgs` to the result of using its `.query()` method to select the records where `"COUNTYFP"` is equal to `"067"`.
8. Set variable `keep_cols` to a list consisting of the strings `'GEOID'`,`'COUNTYFP'`,`'geometry'`. Then set `bgs` to the result of using `keep_cols` to select the desired columns from `bgs`.
9. Reproject `bgs` so that it matches the projection used with the rings by setting `bgs` to the result of calling `.to_crs()` on `bgs` with the argument `rings.crs`.
10. Calculate the area of each block group by setting `bgs['bg_area']` to `bgs.area`. Because `.area` is an attribute rather than a function, no parentheses should be used. (FAQ 1)
11. Now merge on the population data by setting `bgs` to the result of calling `.merge()` on `bgs` with arguments `pop`, `on='GEOID'`, `validate='1:1'`, and `indicator=True`.
12. Print the result of using `.value_counts()` to summarize the merge indicator and verify that all records matched in both datasets. Then delete the merge indicator.
13. Now intersect the rings with the block groups. Set variable `slices` to the result of calling the `.overlay()` method on `bgs` with arguments `rings`, `how='union'`, and `keep_geom_type=True`. Setting `how` to `'union` indicates that all areas should be retained even if they were not present in both datasets. That's needed to pick up the areas outside the rings.
14. Filter out pieces that are outside the county (largely the round end caps of the rings) by setting `slices` to the result of calling `.dropna()` on `slices` with argument `subset='COUNTYFP'`.
15. Use `.fillna(9999)` to replace the missing values of column `'radius'` in `slices`. Those are correspond to slices of the block groups (or whole block groups) that are outside the largest ring. It will be a lot more convenient to track them with a radius of 9999 than with missing data.
16. Set the index of `slices` to `['GEOID','radius']`
17. Create a column in `slices` called `'s_area'` that is equal to `slices.area`. That will be the area of each slice.
18. Determine each slice's share of its block group. Set `area_share` to the result of dividing the `'s_area'` column of `slices` by the `'bg_area'` column.
19. Now use `area_share` to split up the Census variables. Start by creating an empty dataframe called `realloc` that is equal to the result of calling `pd.DataFrame()` with no arguments. Then use variable `v` to loop over `pop.columns`. Within the loop there should be one statement that does the following:
    1. Set column `v` in `realloc` to the result of calling `.mul()` on `slices[v]` using two arguments: `area_share` and `axis='index'`. That will create a column where each slice has its share of the corresponding block group variable.
20. Now aggregate the whole thing up to the ring level. Set `ring_info` to the result of calling `.groupby('radius').sum()` on `slices`.

21. For convenience in graphing the results later, calculate the totals for the county by setting `co_totals` equal to `ring_info.sum()`. If all has gone well, the values in `ring_info` should match the result of adding up the original data via `pop.sum()`.
22. Set `co_pct_poc` equal to 100 times the result of dividing the `'pop_poc'` element of `co_totals` by the `'pop_total'` element.
23. Set `co_pct_rental` equal to 100 times the result of dividing the `'housing_rental'` element of `co_totals` by the `'housing_total'` element.
24. Now set the `'pct_poc'` column of `ring_info` to 100 times the result of dividing the `'pop_poc'` element of `ring_info` by the `'pop_total'` element.
25. Set the `'pct_rental'` column of `ring_info` to 100 times the result of dividing the `'housing_rental'` element of `ring_info` by the `'housing_total'` element.
26. Use `.to_csv()` to write `ring_info` to a CSV file called `"realloc.csv"`. Don't use `index=False` since the index is important.
27. Now we'll tweak the results for plotting. Create a variable called `plot_data` equal to the result of calling the `.reset_index()` method on `ring_info`.
28. Create a column in `plot_data` called `'Distance'` that is equal to the result of calling `.astype(int).astype(str)` on the `'radius'` column of `plot_data`. That makes set of string labels for the distances.
29. Set the `'Distance'` column just created to the value of calling the `.replace()` method on it with arguments `'9999'` and `'>3200'`. That will improve the label for the area beyond the outer ring.
30. Create a column in `plot_data` called `'Population'` that is equal to the `'pop_total'` column. That's just a quick way to fix the case of a label that will appear in a figure legend.
31. Set `plt.rcParams['figure.dpi']` to 300 to improve the resolution of an upcoming figure.
32. Now set `fg` equal to the result of calling `sns.relplot()` with the following fairly long list of arguments: `data=plot_data, x='pct_poc', y='pct_rental', hue='Distance', size='Population', sizes=(10,200), facet_kws={'despine':False,'subplot_kws':{ 'title':'Characteristics of Areas by Distance'}}`. The `sizes` argument sets the minimum and maximum sizes that will be used for points (which are scaled by the population of each ring). The `facet_kws` argument is a dictionary of tweaks allowed by Seaborn's FacetGrid objects: the `'despine'` argument says not to remove the top and right borders of the graph, and the `subplot_kws` is a dictionary of tweaks that are passed on to the underlying matplotlib routines (here it just adds a title).
33. Call the `.set_axis_labels()` method on `fg` with arguments `'Percent People of Color'` and `'Percent Rental'`.
34. Call the `.refline()` method on `fg` with two arguments: `x=co_pct_poc` and `y=co_pct_rental`. That adds reference lines at the county's overall percentages.
35. Call the `.tight_layout()` method on `fg`.
36. Save the figure by calling the `.savefig()` method on `fg` with argument `'char_by_ring.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.

Tips

- The air quality issue is most acute within a couple of hundred meters of the road: roughly the 200 and 400 meter rings. These results indicate that about 52,000 people, or about 11% of the county's population, live in that area and it is disproportionately skewed toward people of color and renters.

- These results could be refined to account for the locations of residential buildings within each block group. However, doing so would require estimating the occupancy of each building, which is especially tricky with apartments.

FAQs

1. Census shapefiles contain areas in square meters in the "ALAND" and "AWATER" fields. However, it's a good idea to recalculate them to ensure that the units match those in the current projection, which might differ from meters for projections used by state and local governments.