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 three input files: **near-parcels.gpkg**, the output geopackage file from the previous exercise; **tl_2018_36_bg.zip**, a shapefile of Census block groups for New York State; and **class_200_by_bg.csv**, a CSV file giving the count of residential properties (property class 200) in each block group in Onondaga County. The shapefile will need to be downloaded from the Census. Be sure to get the 2018 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: scripts **pop_by_bg.py** and **near_road.py** and graphs **prob_by_race.png** and **odds_ratio.png**.

Instructions

Script pop_by_bg.py

1. Import modules as needed.
2. Following an approach similar to that of previous exercises, set up a request to the Census API server to collect data by block group for Onondaga County for two variables: "B02001_001E" and "B02001_002E". The first is the total population and the second is the population that reports its race as "white alone". Use the 2018 ACS5 endpoint (same as in previous exercises), use `"block group:*"` as the `for_clause`, and use `"state:36 county:067"` as the `in_clause`.
3. Use the response from the server to construct a dataframe called `pop`.
4. Use a dictionary to rename the columns of `pop` for `"B02001_001E"` and `"B02001_002E"` to `"total"` and `"white"`.
5. 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.
6. Set the index of `pop` to `"GEOID"`.
7. Create a variable called `keep_list` that is equal to a list containing the strings `"total"` and `"white"`.
8. Trim down `pop` by setting it to the result of using `keep_list` to select the two columns from `pop`.
9. Save `pop` to `"pop_by_bg.csv"`.

Script near_road.py

1. Import modules as needed.
2. Set `bg_data` to the result of using `pd.read_csv()` 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.
3. Set the index of `bg_data` to `"GEOID"`.
4. Set column `"poc"` in `"bg_data"` to the difference between the `"total"` and `"white"` columns.

5. Set count_res to the result of using pd.read_csv() to read "class_200_by_bg.csv". As above, be sure to read "GEOID" as a string.

6. Set the index of count_res to "GEOID".

7. Set near to the result of using geopandas.read_file() to read "near-parcels.gpkg" layer "parcels". These are the residential properties from the previous exercise that are within 2000 m of the interstate.

8. Set prop_class equal to the result of applying the .astype(float) method to the "PROP_CLASS" column of near.

9. Set is_res equal to the result of calling the .between() method on prop_class with arguments 200 and 299. That's the range of residential property classes so the result will be a series indicating whether each row of the dataframe is a residential property.

10. Set houses equal to the result of using is_res to select the residential rows of near.

11. Set bg_geo to the result of using geopandas.read_file() to read "tl_2018_36_bg.zip".

12. Filter bg_geo down to Onondaga County by setting bg_geo to the result of using its .query() method to select the records where "COUNTYFP" is equal to "067".

13. Now set bg_geo to the result of using its .to_crs() method to change its projection to epsg:26918. As in a previous exercise, that will speed up a subsequent spatial join by matching the projection used in the parcel file.

14. Create a variable called keep_list that is equal to a list containing the column names "GEOID" and "geometry".

15. Set bg_geo to the result of selecting the columns in keep_list from bg_geo.

16. Now use a spatial join to add an appropriate block group geoid to the attributes for each house. Set houses_by_bg to the result of calling geopandas.overlay() with arguments houses, bg_geo, and how="intersection".

17. Set grouped to the result of calling .groupby() on houses_by_bg using the list ["GEOID","radius"] as the argument. Add a call to .size() at the end of the line. The result will be a dataframe with the count of houses in each combination of block group and radius.

18. Set near_counts to the result of calling .unstack() on grouped with the argument fill_value=0. The fill_value= argument will fill in zeros for any block group and distance combinations that have no houses.

19. Set column "near" of the earlier dataframe count_res to the result of applying the .sum() method to near_counts with the argument axis="columns". That will be the total number of houses in the block group that are in one of the rings near the interstates.

20. Then set column "near" of count_res equal to the result of calling .fillna(0) on itself to set any missing values to 0. This step is necessary because many block groups in the county are more than 2000 m from any interstate and thus have no houses in any of the rings.

21. Set column "far" of count_res to the difference between the "count" and "near" columns of count_res. This is the number of houses in each block group outside the outer ring.

22. Set houses to the result of calling pd.DataFrame() with the argument index=count_res.index. Using the index= argument ensures that the index will have an entry for every inhabited block group in the county (not just those near the highways) since county_res was built from the Census population data.

23. Set houses equal to the result of calling the .join() method on houses with argument near_counts. The .join() method is a streamlined version of .merge() that can be used when joining objects based on their indexes rather than their columns. That's why no on= argument is used here.

24. Set column 2999 in houses to the "far" column of count_res. Note that 2999 is a number, not a string. It will be used for all residences in each block group that are beyond the 2000 m ring.

25. Set `houses` to the result of calling `.fillna(0)` on itself to fill in some cells where there are no houses.
26. Set `shares` to the result of calling `.div()` on `houses` using the arguments `count_res["count"]` and `axis="index"`. The result will be a set of shares showing the fractions of each block group's houses that fall in each zone.
27. Set `stacked` to the result of calling `.stack()` on `shares`. This format will be convenient for a step below where we'll estimate the racial populations of each ring.
28. Set `by_race` to the result of calling `pd.DataFrame()` with no arugments.
29. Set the `"white"` column of `by_race` to the result of multiplying `stacked` by `bg_data["white"]`. The result will be the estimated number of residents living in each zone who identify as white alone.
30. Use a similar process to set the `"poc"` column of `by_race`.
31. Now set `by_race_ring` to the result of calling `.sum()` on `by_race` with argument `level=1`. This will aggregate over block groups and the result will be the total number of residents of each race living in each zone. The `level=1` argument indicates that the sum is to be over index level 1, which is the radius.
32. Set `prob` to the result of dividing `by_race_ring` by the result of calling `.sum()` on `by_race_ring`. The result will be the probability that a random resident of each race will live in each zone.
33. Set `ratio` to the result of dividing the `"poc"` column of `prob` by the `"white"` column. The result will be the odds ratio for a person of color living in a particular zone relative to a white person.
34. Use `fig, ax1 = plt.subplots()` to set up a single panel figure using `dpi=300`. Then use `fig.suptitle()` to set the title to `"Probability of Location, by Race"`.
35. Call the `.plot.bar()` method on `prob` with arguments `y=["poc","white"]` and `ax=ax1`.
36. Set the X axis label to `"Distance from Nearest Interstate"` and set the Y axis label to `"Probability"`.
37. Tighten the layout and save the figure as `"prob_by_race.png"`.
38. Use a similar approach to set up a second figure. This time, use `"Odds Ratio of Location, POC to White"` as the title.
39. Call the `.plot.bar()` method on `ratio` using argument `ax=ax1`.
40. Set the X axis label as in the previous graph. Set the Y axis label to `"Ratio"`.
41. Call the `.axhline()` method on `ax1` with argument `1` to add a horizontal line where the odds ratio is 1, which is the value at which a person of each race would have an equal chance of ending up in the zone.
42. Tighten the layout and save the figure as `"odds_ratio.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. If all has gone well, your results will show that about 8% of the county's population (about 35,000 people) live in that area, and that people of color are at about double the risk of being in those zones as the white population.
- These results could be refined to account for the fact that some properties are two- and three-family houses, and to account for apartment buildings.