Exercise: Analyzing Residential Property by Zip Code

Summary

This exercise uses parcel-level property tax data from New York State to explore residential property values by zip code in Onondaga County.

Input Data

The input data is a file called **onondaga-tax-parcels.zip** that will need to be downloaded from the course Google Drive folder. It *does not* need to be unzipped: you'll read the data from it directly using Pandas.

The zip file contains a single large CSV file called `onondaga-tax-parcels.csv`. That file has one record (line) for every parcel of land in Onondaga County. It is derived from the New York State master database for the county but some fields have been removed to make the file smaller. There are a bit more than 180,000 parcels in total and just over 144,000 of them are residential.

Each of the records has 20 fields (columns). The ones we'll use are: "TOTAL_AV", the total assessed value of the parcel in dollars (it's the total in the sense that it is the sum of the values of the land itself and the buildings on it); "SQFT_LIV", the size of the living area in square feet; "NBR_BEDRM", the number of bedrooms; "YR_BLT", the year the house was built; "CALC_ACRES", the size of the property in acres; "PROP_CLASS", which indicates the classification of the parcel as agricultural, commercial, residential, or various other categories; and "ZCTA5CE10", which is the 5-digit zip code for the property. ZCTA5 is a US Census designation that stands for "zip code tabulation area at the 5-digit level" and CE10 indicates that the zip code boundaries correspond to those that were in place during the 2010 census. There are other variables in the file (and even more in the original master) but those are the only ones we'll use for this exercise. The first few lines are available in `firstlines.csv` in the Google Drive folder.

Deliverables

A script called **parcels.py** that goes through the steps below. It will produce a figure and a CSV file called `parcels.csv` with summary information about residential housing in each zip code in the county. You'll also submit a short Markdown file called **results.md** with some observations about the results.

Instructions

1. Import `pandas` as `pd` and import `matplotlib.pyplot` as `plt`. In case you're curious, import statements can be included in any order.
2. Set `use_type` to a small dictionary with one key: `"ZCTA5CE10"`. The value for the key should be `str`. Please note that it's just `str` with no quotes: it represents the string data type. The dictionary will be used in the next step to tell Pandas what datatype to use for reading the `"ZCTA5CE10"` column.
3. Set `raw` to the result of calling `pd.read_csv()` with the arguments `"onondaga-tax-parcels.zip"` and `dtype=use_type`. See the tips section for an explanation of the `dtype` argument and why it's important for zip codes.
4. One way to get the number of records in a dataframe is to use the `len()` function. Print `len(raw)` to show the number of records in `raw`. Please use the same approach in other places below where the number of records in a dataframe is needed.
5. Now create a new column in `raw` called `"zipcode"` that is equal to the `"ZCTA5CE10"` column of `raw`. This is just a quick way to be able to use a clearer name for the zip code. As you'll see below and in future exercises, it's also possible to rename variables rather than duplicating them.
6. Now create a column in `raw` called `"value_k"` that is equal to the `"TOTAL_AV"` column of `raw` divided by `1000`. That will allow the property value results below to be reported in thousands of dollars, which is clearer and more convenient that just plain dollars.

7. Create a list called `key_vars` that consists of the three strings `"zipcode"`, `"value_k"`, and `"PROP_CLASS"`.
8. Now create a new dataframe called `usable` by calling the `.dropna()` method on `raw` with the argument `subset=key_vars`. The effect will be to drop all the records that have missing data for one or more of the variables in `key_vars`.
9. Print the number of records in `usable`. It should be around 300 fewer than `raw`.
10. Next we'll skip the parcels that aren't residential. Residential parcels have property class values from 200 to 299. Set `is_res` to the result of calling the `.between()` method on `usable` column `"PROP_CLASS"`. Use arguments 200 and 299 in the call to `.between()`. The result will be a series of `True` and `False` values, one for each record, indicating whether each record is in the residential range.
11. Create a new dataframe called `res` that is equal to `usable[ is_res ]`. That will pick out each of the records in `usable` where `is_res` is `True`.
12. Set `tot_n` to the number of records in `res`.
13. Set `mean_k` to the value of calling `.mean()` on `res` column `"value_k"`.
14. Set `tot_bil` to the value of calling `.sum()` on the `"value_k"` column of `res` and then dividing by `1e6`. The result will be the total value of property in Onondaga County in billions of dollars. Note that the division is by 1e6 rather than 1e9 because the values are already in thousands.
15. Use three print statements to write out the three values just calculated, each with a short informative message indicating what it is.
16. Now group the records by zip code by setting `by_zip` to the result of calling `.groupby()` on `res` with the argument `"zipcode"`.
17. Now create a list called `attr` that consists of the strings `"SQFT_LIV"`, `"NBR_BEDRM"`, `"YR_BLT"`, and `"CALC_ACRES"`. It will be used to pick out those characteristics of each residential parcel.
18. Create a dataframe called `summary` by calling `.median()` on `by_zip[attr]`. That will aggregate the grouped data by zip code: the result will have one column for each variable in `attr` and one row for each zip code. As you might expect, each value will be the median for corresponding variable in the corresponding zip code.
19. Round the medians to two decimal places by setting `summary` to the result of calling `.round()` on `summary` with argument 2.
20. Now we'll sort the data by the vintage of the buildings. Set `summary` to the result of calling `.sort_values()` on `summary` with argument `"YR_BLT"`.
21. Print `summary`. The result gives a rough picture of how development in the county proceeded over time.
22. Now we'll take a detailed look at property values. Set `value` to the result of calling `.describe()` on `by_zip` column `"value_k"`. The result will be a dataframe with summary statistics about property values by zip code.
23. Round the data to two decimal places by setting `value` to the result of calling `.round()` on `value` with argument 2.
24. Now sort the data in `value` from lowest to highest median property value by setting `value` to the result of calling `.sort_values()` on `value` using the argument `"50%"`.
25. Print `value`.
26. Now we'll merge the data in `value` into the `summary` dataframe. Since the dataframes have the same index (the zip codes), an easy way to do that is to go through the columns in `value` and insert them into `summary` one by one. Set up a `for` loop using variable `c` to loop over the columns of `value`. Within the loop include a single statement that sets `summary` column `c` to `value` column `c`.
27. Put the data back in zip code order by setting `summary` to the result of calling `.sort_index()` on `summary`.

28. Write the results to the output file by calling `.to_csv()` on `summary` with argument `"parcels.csv"`.
29. Now set a tuple consisting of variables `fig1` and `ax1` to the result of calling `plt.subplots()`. That creates a new blank figure (`fig1`) with a single set of drawing axes (`ax1`).
30. Then call `.plot.scatter()` on `summary` with four arguments: `"YR_BLT"`, `"SQFT_LIV"`, `s="50%"`, and `ax=ax1`. Note that `s` is a keyword argument and should not be in quotes. The last argument, `ax=ax1` causes the plot to be drawn on the axes set up in the previous step. The result will be a scatter plot with `"YR_BLT"` on the horizontal axis, `"SQFT_LIV"` on the vertical axis, and with the size of the dots scaled by the median value, `"50%"`.
31. Set the title by calling `.set_title()` on `ax1` with argument `"Characteristics of Zip Codes"`.
32. Finalize the formatting of the figure by calling `.tight_layout()` on `fig1`. That fine-tunes the spacing of labels and axes in figures to make sure that everything fits properly. It's not really needed here but is definitely needed in later assignments and it's a good practice in general.
33. Save the figure by calling `fig1.savefig()` with two arguments: `"parcels.png"` and `dpi=300`. The `dpi` argument sets the resolution of the figure to 300 dots per inch, which is sharper than the default.
34. Edit the `results.md` file and replace the TBD placeholder with a few notes on the results. There's nothing specific you need to mention; rather, just spend a few minutes looking over the output and say a little about some things you think are interesting.

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 `dtype` argument is used to tell `read_csv()` what datatype it should use for some or all of the columns in the file. When given a dictionary, `read_csv()` will use the indicated datatype for any columns that have keys in the database, and will determine on its own what the data type should be for all the other columns. Here we're telling it to use strings for the zip codes, which it would otherwise treat as numeric data. It's a very good practice to always treat zip codes as strings because some zip codes begin with 0's. It's the same reason that FIPS codes should always be strings.
- In later exercises we'll join this data to information from the Census and other sources to examine the county in more detail. We'll also map it using GIS.
- Data on all tax parcels in New York State is available from the NYS geographic information system (GIS) data repository. A link is available from the resources section of the class web page.