Exercise: Aggregating 2020 Political Contributions

Summary

This exercise is the first of several that examine data from the US Federal Election Commission (FEC) on contributions by individuals to 2020 presidential campaigns.

Input Data

The input data is a large file called **contributions.zip** that will need to be downloaded from the course Google Drive folder. Be sure to leave it zipped because it's very large (about 850 MB) when unzipped.

The zip file contains a single CSV file called **contributions.csv**. It's a trimmed-down version of 2020 election data available on the FEC website. The full file contains contributions to House, Senate, and Presidential campaigns, and it has a lot of data about each contribution. It's almost 9 GB zipped and about 14 GB unzipped. To keep the size down for this assignment, **contributions.csv** contains only contributions to Presidential campaigns and omits some variables.

Each record in **contributions.csv** contains the following fields: `'CMTE_ID'`, `'NAME'`, `'CITY'`, `'STATE'`, `'ZIP_CODE'`, `'EMPLOYER'`, `'OCCUPATION'`, `'TRANSACTION_PGI'`, `'TRANSACTION_AMT'`, and `'TRANSACTION_DT'`. The first field, `'CMTE_ID'`, is an FEC identifier indicating the campaign committee that received the contribution. The `'TRANSACTION_PGI'` field is a five character string where the first character indicates the type of election and the remaining four characters indicate the election year. The types that are relevant for this assignment are `'P'`, which indicates a primary, and `'G'`, which indicates a general election. The `'TRANSACTION_AMT'` field gives the contribution, in dollars, and the `'TRANSACTION_DT'` field is an eight character string giving the date in the form MMDDYYYY. The other variables are self-explanatory. In this exercise, we'll aggregate contributions up to the committee level; in a subsequent exercise we'll map the committees to candidates.

Because the main file is very large, and processing it can be slow, there is also a file on the course drive called **sample.zip**. It's a random sample of 1 percent of the records in the main file. You may want to use it when developing your script until you have things working smoothly. The first few lines are also provided in **firstlines.csv** if you'd like to see what the records look like. If you use the sample file be sure to remember to run your final script on **contributions.zip**.

Deliverables

There are two deliverables for this assignment. The first is a script called **contrib_all.py** that reads the original data, builds a Pandas dataframe, and writes the dataframe out in pickled form. A pickled dataframe is not human-readable but it can be reloaded by Python much more quickly than rereading the CSV file. Pickling is a very useful way to pass large datasets between scripts without the extra overhead of converting them to and from human-readable form.

The second deliverable is a script called **contrib_by_zip.py** that reads in the pickled file, aggregates the data, and writes out the results as a CSV file called `"contrib_by_zip.csv"`. As in several earlier exercises, `"contrib_by_zip.csv"` is not a deliverable: I'll reproduce it by running your scripts.

Instructions

A. Script contrib_all.py

1. Import modules as needed.
2. Create a new Pandas dataframe called `raw` by calling `pd.read_csv()` on `"contributions.zip"`. Use the keyword `dtype=str` to tell Pandas to load the data in string form without trying to infer which variables

are numeric. That makes things a little faster and it avoids messing up the zip codes, which can have leading zeros.

3. Set variable `n_now` to `len(raw)`. Then print a message indicating the number of records read.
4. Rename the column called `"TRANSACTION_PGI"` to `"PGI"` for convenience later in the script.
5. Create a numeric version of the transaction amount by creating a new column in `raw` called `'amt'` that is equal to the result of calling the `.astype(float)` method on the `'TRANSACTION_AMT'` column.
6. Set variable `ymd` equal to the result of calling `pd.to_datetime()` with two arguments: the `"TRANSACTION_DT"` column of `raw` and `format="%m%d%Y"`. The `format` argument tells Pandas that the dates are stored in MMDDYYYY form.
7. Create a new column in `raw` called `"date"` that is equal to the result of calling the `.dt.to_period("M")` method on `ymd`. That converts the full date into one with monthly ("M") frequency, which will be handy later for grouping the data by month.
8. Set variable `n_last` equal to `n_now`.
9. Set variable `year_ok` equal to `ymd.dt.year >= 2019`. Then create a new dataframe called `contrib` that is equal to the rows of `raw` where `year_ok` is true. This filters out a small number of contributions from earlier years.
10. Set `n_now` equal to the number of records in `contrib`.
11. Indicate the number of records just dropped by printing an informative message about the filtering that was just applied, and then print the reduction in records, `n_last-n_now`.
12. Set variable `n_last` equal to `n_now`. Yep, it needs to be done again.
13. Pick out the records for the primary and general elections in 2020 by setting `contrib` equal to the result of calling the `.query()` method on `contrib` with the string `"PGI == 'P2020' or PGI == 'G2020'"`.
14. Set `n_now` equal to the number of records in `contrib`.
15. As before, indicate the number of records just dropped by printing an informative message about the filtering that was just applied and then printing the reduction in records, `n_last-n_now`.
16. Print an informative message indicating the final number of records and then print `n_now`.
17. Create a list called `keepvars` containing `'CMTE_ID'`, `'STATE'`, `'ZIP_CODE'`, `'PGI'`, `'date'` and `'amt'`.
18. Create a new variable called `trimmed` that is equal to the `keepvars` columns of `contrib`.
19. Save a pickled version of `trimmed` by calling its `.to_pickle()` method. Use `'contrib_all_pkl.zip'` as the name of the pickle file. The `.zip` extension causes the file to be compressed, which reduces its size by about 85%. If all goes well, you won't have to rerun the script once you've successfully built the pickle file.
20. Print an informative message and then print `len(trimmed)`, the number of records in the pickled dataframe.
21. Print an informative message and the print `list(trimmed.columns)` to verify that it has the correct variables.
22. Now we'll build a figure showing the pattern of contributions over time. Start by setting `grouped` to the value of grouping `trimmed` by `"date"` and `"PGI"`.

23. Create variable `by_date_pgi` by applying `.sum()` to the `"amt"` column of `grouped` and then dividing the result by `1e6` to convert it to millions of dollars.
24. Unstack the data to create separate columns for the primary and general elections by setting variable `by_date_wide` equal to the result of calling `.unstack()` on `by_date_pgi` with the argument `"PGI"`.
25. Begin a new figure by setting `fig1, ax1` equal to the result of calling `plt.subplots()` with the argument `dpi=300`.
26. Set the figure's title to `"Individual Contributions"`.
27. Plot the data by calling the `.plot()` method on `by_date_wide` using the argument `ax=ax1`.
28. Set the label for the X axis to `"Date"`.
29. Set the label for the Y axis to `"Million Dollars"`
30. Use `.tight_layout()` to adjust the figure's spacing.
31. Save the figure using the name `"by_month.png"`.

B. Script contrib_by_zip.py

1. Import modules as needed.
2. Create a new dataframe called `contrib` by calling `pd.read_pickle()` on the pickle file produced by the previous script.
3. Create a variable called `zip_all` equal to the `'ZIP_CODE'` column in `contrib`. As you'll see, we'll need to do some work to standardize the zip codes.
4. Create a variable called `ziplen` equal to the result of applying the `.str.len()` method to `zip_all`. That will produce a series with the length, in characters, of each zip code. The `.str` tells Pandas that you're intending to apply a string operation to each item in the series. The `.len()` tells it the operation to apply is `len()`.
5. Print the result of applying the `.value_counts()` method to `ziplen` using the argument `dropna=False`. That will produce a table of counts of zip code lengths. The two most common lengths will be 9 and 5, but there will be a few others. The `dropna=False` argument tells Pandas to report a count of missing zip codes in addition to those with actual lengths.
6. Create a variable called `zip_9` equal to `ziplen == 9` and create a similar one called `zip_5` for length 5. These variables will have `True` or `False` values for each element of `ziplen`. For example, the value of `zip_9` for a given index will indicate whether the length of the zip code in `zip_all` at the same index is 9.
7. Create a variable called `zip_ok` that is equal to `zip_5 | zip_9`. The result will be a series with `True` wherever the zip code is either 5 or 9 characters, and `False` everywhere else.
8. Create a variable called `zip_bad` that is equal to `~ zip_ok` (note the tilde). That will be equal to `True` for any records where the zip code isn't 5 or 9 characters.
9. Create a variable called `zip5` that is equal to `zip_all` with the `.copy()` method applied.
10. Set the values of `zip5` where the original zip code is 9 digits to the first five of them. Use `zip5[zip_9]` on the left side of an equals sign and `zip5[zip_9].str[:5]` on the right side. As before, the `.str` tells Pandas that you're applying a string operation to each string in the series. In this case the operation is `[:5]`, which selects the first five characters from the string.

11. Set the values of `zip5[zip_bad]` to `None`, which will mark them as missing data.
12. Create a variable called `zip5len` that is equal to the lengths of the zip codes in `zip5`. Then print the value counts using `dropna=False` and check that they are consistent with the earlier counts.
13. Create a new column in `contrib` called `'zip'` equal to `zip5`.
14. Create a variable called `grouped` equal to `contrib` grouped by a list of the following columns: `'CMTE_ID'`, `'STATE'`, `'zip'`.
15. Create a variable called `contrib_by_zip` equal to the `.sum()` method applied to the `'amt'` column of `grouped`.
16. Use the `.to_csv()` method to write `contrib_by_zip` to a file called `contrib_by_zip.csv`.

C. Cleaning up

1. Once you're happy with `contrib_by_zip.csv` you should probably delete both `contributions.zip` and `contrib_all_pkl.zip`. That will save you about 230 MB of disk space and the files won't be needed for the other FEC assignments. Be sure to keep `contrib_by_zip.csv`, though: that one will be used.

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

- Don't be surprised by strange values in the state and zip code fields. The data is in the form it was reported to the FEC and there contributions from overseas as well as data entry errors.