Exercise: Aggregating 2016 Political Contributions

Summary

This exercise is the first of two that examine data from the US Federal Election Commission (FEC) on contributions by individuals to presidential campaigns in 2016.

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 400 MB) when unzipped.

The zip file contains a single CSV file called **contributions.csv**. It's a trimmed-down version of 2016 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 more than 1.4 GB zipped and almost 4 GB unzipped. To keep the size down for this assignment, **contributions.csv** only contains individual 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_AMT'. Because contributions are made to campaign committees rather than individuals, the first field, CMTE_ID, is an FEC identifier indicating the committee that received the contribution. 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 10 lines of it are provided in **sample-head.csv** if you'd like to see what the records look like. If you use the sample file, though, be sure not to forget 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. However, please make sure to use that name in your script so that the correct file will be produced when I run the script.

Instructions

A. Script contrib_all.py

1. Import modules as needed.
2. Create a new Pandas dataframe called `contrib` by calling `pd.read_csv()` on `contributions.zip`. In the call give 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 (e.g., 02138) and are better left as strings.
3. Create a new column in `contrib` called 'amt' that is equal to the 'TRANSACTION_AMT' column with the `.astype(float)` method called on it to convert it to numeric form.
4. Create a list called `keepvars` containing 'CMTE_ID', 'STATE', 'ZIP_CODE', and 'amt'.
5. Create a new variable called `trimmed` that is equal to the `keepvars` columns of `contrib`.

6. Print an informative message and then print `len(trimmed)`, which will be the number of records in the dataframe.
7. Print an informative message and the print `list(trimmed.columns)` to verify that it has the correct variables.
8. Save a pickled version of `trimmed` by calling its `.to_pickle()` method. Use 'contrib_all.pkl' as the name of the pickle file. If all goes well you won't have to rerun the script once you've successfully built the pickle file.

B. Script contrib_by_zip.py

1. Include an import of `numpy` as `np` with any other imports that are 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(dropna=False)` method to `ziplen`. That will produce a table of counts of zipcode lengths. The two most common will be 9 and 5, but there will be a few others. The `dropna=False` tells Pandas to include missing values in the output.
6. Create a variable called `zip_9` equal to `ziplen == 9`. Create a similar one called `zip_5` for length 5. These variables will have a 0 or 1 for each element to indicate whether the length of the corresponding zipcode is 9 or 5. That is, the value of `zip_9` for a given index will indicate whether the length of the zipcode in `zip_all` at the same index is 9.
7. Create a variable called `zip_ok` that is equal to `zip_5 | zip_9`. In this case we want Python's bitwise `or` operator because Pandas will apply it pairwise to the elements of `zip_5` and `zip_9`. It will work correctly because each element of `zip_5` and `zip_9` is 0 or 1, so a bitwise `or` will correctly determine whether one or the other of the conditions is true.
8. Create a variable called `zip_bad` that is equal to `~zip_ok` (note the tilde). That will be equal to 1 for any records where the zipcode 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[0:5]` 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 `[0:5]`, which selects the first 5 characters from the string.
11. Set the values of `zip5[zip_bad]` to `np.NaN`, which is the Numpy and Pandas code for 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`. Doing so will save you about 250 MB of disk space and the files won't be needed for the second FEC assignment. 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.