# Exercise: Analyzing the 2020 Contributions Data

## Summary

This exercise cleans up the election data from the earlier assignment, joins on some additional information from the FEC, and then does a little analysis.

## Input Data

The main input file is **contrib_by_zip.zip**, which is available from the course Google Drive. It's a zipped version of the file produced in the previous FEC assignment. The additional files are provided in the Google Drive folder as well: **pocodes.csv**, a list of states that will be used for filtering the data; **fec_committees.csv**, FEC information about campaign committees; and **fec_candidates.csv**, FEC information about candidates. Note that **fec_committees.csv** and **fec_candidates.csv** contain information about House and Senate races, not just the Presidential election, and there are records for some years other than 2020. All of that will be filtered out.

## Deliverables

The deliverables are three scripts: **contrib_clean.py**, which removes some unneeded records from the aggregated contributions data; **com_cand_info.py**, which builds a file of information about committees and candidates; and **by_place_cand.py** which builds a joined dataset that links contributions to candidates and does a little analysis. Instructions for each are provided below.

## Instructions

### A. Script contrib_clean.py

1. Set `contrib` to the result of using `pd.read_csv()` to read file `"contrib_by_zip.zip"` using `dtype=str`.
2. Make `contrib['amt']` numeric using the `.astype(float)` method.
3. Set `po` to `pd.read_csv()` applied to file `"pocodes.csv"`.
4. Drop the `"Name"` column from `po`.
5. To filter out all the state codes that aren't in the 50 states plus Washington, DC, and Puerto Rico (PR), we'll join on a list of the state codes we want to keep, which are in `po`. That will let us remove the other records shortly using the merge indicator. To do the join, set `contrib` to the result of calling the `.merge()` method of `contrib` with the following arguments: `po`, `left_on='STATE'`, `right_on='PO'`, `how='outer'`, `validate='m:1'`, and `indicator=True`. This will use the two-letter postal codes as the keys in the join. The `left_on` and `right_on` arguments are needed because the column names for the postal codes are not the same in the two datasets.

   Even though we eventually only want to keep records appearing the right dataset, this is done as an outer join rather than a right join so we can add up the contributions that will be left out.
6. Print the result of calling `.value_counts()` on the `'_merge'` column of `contrib`. Records for places in `"pocodes.csv"` will be listed as `'both'`. Those in the FEC data but not in `pocodes.csv` will be listed as `'left_only'`. Anything that was in `"pocodes.csv"` but not in the FEC data would be shown as `'right_only'` but that should be 0 in this case.

   Please note that this step will come up after most or all of the merges. From here on out the instruction *print the merge indicator* will mean to do something like this for the merge that was just done.
7. Set `state_bad` to `contrib['_merge'] != 'both'`. We'll use that to remove the records that didn't match the geographic entities in `po`. We'll refer to those as bad states because we're focusing on the states plus DC and PR. However, a lot of them are actually legitimate postal codes for things other than US states, such as US military addresses, US territories, Canadian provinces, and foreign country codes.
8. Drop the `'_merge'` and `'PO'` columns from `contrib`. We're done with them.

9. Next we'll tabulate the data that's going to be dropped when we exclude records with bad state codes. Start by picking out the bad records and grouping them by state as shown below:

   ```
   bad_recs = contrib[state_bad].groupby('STATE')
   ```

   This is doing two things in one step that we've often done in the past in two steps. The `contrib[state_bad]` selects the records where `state_bad` is True and `.groupby('STATE')` then groups those records by state code.

10. Sum up the contributions in those states by setting `state_bad_amt` to the result of applying the `.sum()` method to `bad_recs['amt']`.

11. Print `state_bad_amt` to show the state codes and total contributions. Then print `state_bad_amt.sum()` to show the total contributions from those states. It's important to have a concrete idea about how much data is lost when filtering out records.

12. Now filter out the records by setting `contrib` to the rows of `contrib` where `state_bad == False`.

13. Now we'll look for bad zip codes by finding any that aren't purely numeric. To do that, set `num_zip` to the result of calling the `pd.to_numeric()` function with the following arguments: `contrib['zip']` and `errors='coerce'`. That tells Pandas to build a new series by converting the `zip` column into its numeric equivalent. The important feature is the `errors='coerece'` argument: that tells Pandas to put in the missing data code for anything that can't be converted to a number rather than stopping with an error message.

14. Set `zip_bad` to the result of applying the `.isna()` method to `num_zip`. The result will be a series with `True` where the corresponding value of `num_zip` is missing and `False` everywhere else.

15. Do an analysis similar to that for `state_bad`. Set `bad_recs` to `contrib[zip_bad].groupby('zip')`. Then set `zip_bad_amt` to the result of summing `bad_recs['amt']`, print `zip_bad_amt`, and print the result of summing it.

16. Now filter out the records by setting `contrib` to the rows of `contrib` where `zip_bad == False`.

17. Use the `.to_pickle()` method of `contrib` to write a file called `contrib_clean.pkl`.

18. Now we'll compute total contributions by committee, which will be useful later. Start by summing the contributions to each committee. Set `by_com` to `contrib` grouped by the committee `'CMTE_ID'` and then create `com_total` by applying the `.sum()` method to `by_com['amt']`.

19. Then change the name of the data in the series to `'total_amt'` to reflect that it is the total by committee. That's done by setting the `name` attribute of the series to `'total_amt'` as follows:

    ```
    com_total.name = 'total_amt'
    ```

20. Finally, use the `.to_csv()` method to write `com_total` to file `com_total.csv`.

### B. Script com_cand_info.py

1. Import any modules needed.

2. Set `contrib` to the result of using `pd.read_pickle()` to reload the data written by the previous script.

3. Set `com_total` to the result of using `pd.read_csv()` on `com_total.csv`.

4. Set `com_info` to the result of applying `pd.read_csv()` to file `"fec_committees.csv"` using the argument `dtype=str`.

5. Trim down `com_info` to the following columns: `'CMTE_ID'`, `'CMTE_NM'`, `'CMTE_PTY_AFFILIATION'`, `'CAND_ID'`. (FAQ1)

6. Now we'll join the total contributions onto `com_info`. Set `com_merged` to the result of calling the `.merge()` method of `com_info` with the following arguments: `com_total`, `how='right'`, `validate='m:1'`, and

indicator=True. We're using a right join because we only want the committees that had contributions in the presidential race.

7. Print the merge indicator to verify that all of the committees with contributions were found in com_info and then drop '_merge'.
8. In principle, a committee could fund multiple candidates, which would be a problem because we wouldn't know how the committee split its donations between the candidates. We'll check whether that's an issue. Set numcan to com_info grouped by 'CMTE_ID', and then apply the .size() method to the result by placing it at the end of the line after the .groupby() call. The .size() method counts the number of entries in each group, so the result will be a series with the number of times each committee appears in com_info.
9. Print numcan for rows where numcan > 1. If all has gone well the result will be an empty series. That indicates that there aren't any committees with more than one candidate.
10. Now read the information about candidates. Set pres to the result of using pd.read_csv() to read fec_candidates.csv using dtype=str.
11. Next we'll filter out everyone who wasn't running for President in 2020. Start by setting is_pres to pres['CAND_OFFICE'] == 'P'. That will be true for Presidential candidates and false for everyone else.
12. Set is_2020 in a similar manner but use 'CAND_ELECTION_YR' and '2020'.
13. Set keep to is_pres & is_2020. That will be true for Presidential candidates in 2020 and false everywhere else.
14. Set pres to the subset pres[keep]. That will eliminate all the other candidates and election years.
15. Drop 'CAND_OFFICE' and 'CAND_ELECTION_YR' from pres: we're done with them.
16. Now we'll join the candidate date onto the committee information. Set com_cand to the result of applying the .merge() method of com_merged with the following arguments: pres, how='left', validate='m:1', indicator=True. Because we're not specifying any join keys, Pandas will default to using all the columns having identical names in both dataframes. Here, that's only CAND_ID, which is exactly what we want.
17. Print the merge indicator. You should see that some committees were eliminated: those were committees for candidates from previous elections who happened to have some transactions in the 2020 election cycle.
18. Set com_cand to its subset where com_cand['_merge'] == 'both' and then drop '_merge' from com_cand.
19. Use the .to_csv() method to write com_cand to com_cand_info.csv using the index=False argument since the index is just row numbers that we don't need to keep.

### C. Script by_place_cand.py

1. Import modules as needed.
2. Set contrib to the result of using pd.read_pickle() to reload contrib_clean.pkl, and set com_cand to the result of using pd.read_csv() to read com_cand_info.csv.
3. Create merged by joining com_cand onto contrib by using the .merge() method of contrib with the following arguments: com_cand, on='CMTE_ID', validate='m:1', and indicator=True.
4. Print the merge indicator to verify that all records matched and then drop '_merge' from merged.
5. Since candidates may have more than one committee, we'll now aggregate the data to candidates. Set group_by_place_cand to the result of grouping merged by the following columns: 'STATE', 'zip', and 'CAND_NAME'.

6. Set by_place_cand to the result of applying the .sum() method to group_by_place_cand['amt']. That will total up the contributions to each candidate by each place (where a place is a state and zipcode combination).
7. Use .to_csv() to write by_place_cand out to file by_place_cand.csv.
8. Now we'll do a little analysis to see which places provided the largest contributions to each candidate. Start by summing things up to the state and candidate level by setting mil equal to the result of calling .groupby() on by_place_cand with the argument ["STATE","CAND_NAME"], calling .sum() on the result, and then dividing that 1e6 to convert it to millions of dollars.
9. Next, compute overall totals by candidate by setting by_cand to the result of applying .groupby() to mil with the argument "CAND_NAME" and applying .sum() to the result.
10. Select the top 10 candidates by setting top_cand to the result of calling the .sort_values() method of by_cand and then using [-10:] to select the last 10 entries. Then print top_cand.
11. Follow a similar procedure to compute by_state, the overall totals by state, and top_state, the top ten states. Then print top_state.
12. Create a new two-panel figure by setting fig, (ax1,ax2) to the result of calling plt.subplots(1,2,dpi=300).
13. Set the figure title by calling fig.suptitle() with the argument "Top Candidates and States, Millions of Dollars".
14. Call .plot.barh() on top_cand with arguments ax=ax1 and fontsize=7.
15. Set the Y axis label of ax1 to "" (and empty string) to turn it off.
16. Call .plot.bar() on top_state with arguments ax=ax2 and fontsize=7.
17. Set the X axis label of ax2 to "State".
18. Tighten the layout and then save the figure as "top.png".
19. Now we'll build a heatmap of the top candidates and states. Start by setting reset to the result of calling the .reset_index() method on mil. That will convert the index to columns and reset the index itself to sequential numbers.
20. Create keep_cand by calling the .isin() method on reset["CAND_NAME"] using the argument top_cand.index. That will pick out the rows of reset for the top candidates.
21. Create keep_state by calling the .isin() method on reset["STATE"] using the argument top_state.index. That will pick out the rows of reset for the top states.
22. Set keep equal to keep_cand & keep_state. That will be true for rows where both keep_cand and keep_state are true, and it will be false everywhere else.
23. Set sub to the result of using keep to select rows from reset.
24. Now we'll sum things up over the zip codes. Start by setting grouped equal to the result of grouping sub by "STATE" and "CAND_NAME".
25. Next, set summed to the result of applying .sum() to the "amt" column of grouped.
26. Now unstack the date to make columns by state. Set grid to the result of calling the .unstack() method on summed using "STATE" as the argument. If all has gone well, grid should have one row per top-ten candidate and one column per top-ten state.
27. Create a new single-panel figure. (FAQ2)
28. Call .suptitle() on fig to set the figure's title to "Contributions in Millions".
29. Call sns.heatmap() with the following arguments: grid, annot=True, fmt=".0f", and ax=ax1. The annot and fmt arguments cause the cells in the heatmap to be labeled with values rounded to integers.

30. Set the X axis label to `"State"` and the Y axis label to `"Candidate"`.
31. Tighten the layout and then save the figure as `"heatmap.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

- A subsequent assignment will map some of the detailed data saved in `by_place_cand.csv`. In the mean time, you might find it interesting to look up a few zip codes you know to see what contributions looked like in those areas.

## FAQS

1. *How do I trim a dataframe down to a subset of its columns?*

   To trim dataframe `D` down to the columns in list `L` use `D = D[L]`.

2. *How do I create a new single-panel figure?*

   Set `fig, ax1` to the result of calling `plt.subplots(dpi=300)`.