Project: Code Challenge by Insight Data Engineer **Winter 2018**

Report: Repeat Donors for Multiply Years

Kuan Zhou *Date: Feb 09*

This file serves as a report for code challenge - repeat donors for multiple years - designed by Insight Data Engineer, which covers methods, dependencies, conclusions etc.

1.1 Task Description

In order to gain more insights on campaign contributions for political candidates, a file listing individual campaign contributions for multiple years is used to determine which ones came from repeat donors and calculate a few statistics.

1.2 Analysis

(1) **Grouped Zip order:**

The major features of this implementation are:

- Use `defaultdict()` with list for DONARS to complete search in O(1) time;
- Use `defaultdict()` with set for ZIPS to complete search in O(1) time;
- Use a framework of `zip-donar-donor` to extract the repeat donors and output in a grouped zip code order;
- Use of `numpy.percentile()` with `interpolation='lower'` for percentile calculation;
- Donation with amount less than 1 dollars is neglected.

As in `donation_analytics.py`, the running time can be approximated to be at most $O(nlg(n))$.

RESULT: For `itcont.txt` of `indiv16` with $3.83G$, running time: $419.71s$

(2) **Streaming Input order:**

Another method with no grouped zip order is implemented in `donation_analytics_stream.py`. For this implementation, the results for contribution amounts and percentiles are also grouped for zip code and year, but they are calculated and written into output file with streaming input order.

Because the major restriction in my PC is speed not memory, the input is read in and stored in a defaultdict `hash_donars`. When memory is a restriction, it can easily be changed to work in a streaming way.

As in `donation_analytics_stream.py`, the running time can be approximated to be within $O(n)$.

RESULT: For `itcont.txt` of `indiv16` with $3.83G$, running time: $448.16s$

1.3 Dependencies

Operating System: Ubuntu 16.04 LTS(Linux) 64 bit

Memory: 31.3 GiB

Processor: Intel Core i7

Packages: Python 3, Numpy.

1.4 How to Run

In `./` directory:

```
./run.sh
```

In `./src/` directory:

```
python3 ./donation_analytics.py ../input/itcont.txt ../input/percentile.txt
  ../output/repeat_donors.txt
```

- `./src/donation_analytics.py`: the codes to process the extraction in grouped zip order;
- `./src/donation_analytics_stream.py`: the codes to process the extraction in streaming order;
- `./input/`: input directory with `itcont.txt` and `percentile.txt` files;
- `./output/`: output directory with `repeat_donors.txt` output file;
- The ipython notebooks in `./notebooks` are some unit cases for testing and debugging.

1.5 Conclusion

A well organized framework to extract the repeat donors with percentile statistics is completed within $O(n)$ time.

1.6 Future Improvements

- **CPython** Modules in CPython can be used to boost the speed;
- **Multi Threads** Multi threads `multiprocessing` module with insights into codes can be used to parallel the computing;
- **PyPy** A new package PyPy with even more speed can be used to further boost the calculation.

1.7 Possible Errors

- Since relation of zips and DONARS are stored using a `defaultdict()` with set, the order of zips and DONARS are in random. The output order can be random for DONARS for each zip. In this case, the results are correct, but not exactly as in test sample.
- Some other malformed input not listed as in task requirements.