Exercise 4

Due Friday June 11 2021.

Some files are provided that you need below: E4.zip. **You may not write any loops** in your code.

Movie Title Entity Resolution

For this question, we will use some movie rating data derived from the MovieTweetings *(https://github.com/sidooms/MovieTweetings)* data set.

The first thing you're given (`movie_list.txt` in the ZIP file) is a list of movie titles, one per line, like this:

```
Bad Moms
Gone in Sixty Seconds
Raiders of the Lost Ark
```

The second file (`movie_ratings.csv`) contains users' rating of movies: title and rating pairs. Well… the title, except the users have misspelled the titles.

```
title,rating
Bad Mods,8
Gone on Sixty Seconds,6.5
Gone in Six Seconds,7
```

The task for this question is to **determine the average rating for each movie**, compensating for the bad spelling in the ratings file. We will assume that the movie list file contains the correct spellings.

There are also some completely-incorrect titles that have nothing to do with the movie list. Those should be ignored. e.g.

```
Uhhh some movie i watched,7
```

Your command line should take the movie title list, movie ratings CSV, and the output CSV filename, like this:

```
python3 average_ratings.py movie_list.txt movie_ratings.csv output.csv
```

It should produce a CSV file (`output.csv` in the example above) with 'title' as the first column, and (average) 'rating' **rounded to two decimal places** as the second. The output should be **sorted by title** and include only movies with ratings in the data set.

```
title,rating
Bad Moms,8.0
Gone in Sixty Seconds,6.75
```

Hints

- To read line-at-a-time input like in `movie_list.txt`, you could use `open(movie_list).readlines()` to get a list of lines, and construct a DataFrame from there. Or maybe use read_csv *(http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html)* somehow but **not** with the default parameters, since this isn't comma-separated data and movie titles can contain commas.
- Finding similar strings is much easier than I thought it was going to be: in the Python standard library, you'll find Python difflib.get_close_matches *(https://docs.python.org/3/library/difflib.html#difflib.get_close_matches)* . You can use the default cutoff for what is “too

far" to count as a match (0.6).

- To get the entity resolution done, add a column to your ratings data with the best-match real movie title ("real title" or something). That will give you a value to filter and group with.

Cities: Temperatures and Density

This question will combine information about cities from Wikidata *(https://www.wikidata.org/wiki/Q24639)* with a different subset of the Global Historical Climatology Network *(https://www.ncdc.noaa.gov/data-access/land-based-station-data/land-based-datasets/global-historical-climatology-network-ghcn)* data.

The question I have is: **is there any correlation between population density and temperature**? I admit it's an artificial question, but one we can answer. In order to answer the question, we're going to need to get population density of cities matched up with weather stations.

The program for this part should be named `temperature_correlation.py` and take the station and city data files in the format provided on the command line. The last command line argument should be the filename for the plot you'll output (described below).

```
python3 temperature_correlation.py stations.json.gz city_data.csv output.svg
```

The Data

The collection of weather stations is quite large: it is given as a line-by-line JSON file that is gzipped. That is a fairly common format in the big data world. You can read the gzipped data directly with Pandas and it will automatically decompress as needed:

```
stations = pd.read_json(stations_file, lines=True)
```

The `'avg_tmax'` column in the weather data is °C×10 (because that's what GHCN distributes): it needs to be divided by 10. The value is the average of TMAX values from that station for the year: the average daily-high temperature.

The city data is in a nice convenient CSV file. There are many cities that are missing either their area or population: we can't calculate density for those, so they can be removed. Population density is population divided by area.

The city area is given in m², which is hard to reason about: convert to km². There are a few cities with areas that I don't think are reasonable: exclude cities with area greater than 10000 km².

Entity Resolution

Both data sets contain a latitude and longitude, but they don't refer to exactly the same locations. A city's "location" is some point near the centre of the city. That is very unlikely to be the exact location of a weather station, but there is probably one nearby.

Find the weather station that is closest to each city. We need it's `'avg_tmax'` value. This takes an $O(mn)$ kind of calculation: the distance between every city and station pair must be calculated. Here's a suggestion of how to get there:

- Write a function `distance(city, stations)` that calculates the distance between **one** city and **every** station. You can probably adapt your function from the GPS question last week. [1]
- Write a function `best_tmax(city, stations)` that returns the best value you can find for `'avg_tmax'` for that one city, from the list of all weather stations. Hint: use `distance` and `numpy.argmin` (or `Series.idxmin` or `DataFrame.idxmin`) for this. [2]
- Apply that function across all cities. You can give extra arguments when applyling in Pandas like this: `cities.apply(best_tmax, stations=stations)`.

[1] When working on the collection of stations, make sure you're using Python operators on Series/arrays or NumPy ufuncs *(https://docs.scipy.org/doc/numpy/reference/ufuncs.html#math-operations)* , which are much faster than `DataFrame.apply` or `np.vectorize`. Your program should take a few seconds on a reasonably-fast computer, not a few minutes. [Aside: Python operators on Series/arrays are overloaded to call those ufuncs.]

[2] Note that there are a few cities that have more than one station at the same minimum distance. In that case, we'll use the station that is **first** in the input data. That choice happens to match the behaviour of `.argmin` and `.idxmin`, so if you ignore the ambiguity, you'll likely get the right result.

Output

Produce a scatterplot of average maximum temperature against population density (in the file given on the command line).

Let's give some nicer labels than we have in the past: see the included `sample-output.svg`. The strings used are beautified with a couple of Unicode characters: `'Avg Max Temperature (\u00b0C)'` and `'Population Density (people/km\u00b2)'`.

Questions

Answer these questions in a file `answers.txt`. [Generally, these questions should be answered in a few sentences each.]

1. Based on your results for the last question, do you think daily temperatures are a good way to predict population density? Briefly explain why or why not.
2. The larger data file (`stations.json.gz`) was kept compressed on disk throughout the analysis. Decompressing every time we run the program seems inefficient. Why might this be faster than working with an uncompressed `.json` data?

Submitting

Submit your files through CourSys for Exercise 4.

Updated Wed April 07 2021, 09:16 by ggbaker.