# Purpose

This package provides a solution to AlixPartners Challenge Problem #3: "Fuzz-ography". It takes as input a list of city name/country code combinations, some of which contain errors, and attempts to match these inputs against a known third party master list consisting of validated world city names. The output is a list of “validated” city names, with a best guess as to the correct city spelling (in upper case ASCII, to match the input) as provided by the master list, plus a validation quality code, the full country name spelling, and estimated latitude/longitude location.

# Package Contents

When fully assembled, the solution set will consist of eight files, summarized below:

|  |  |  |
| --- | --- | --- |
| **File Name** | **File Type** | **Source** |
| Problem 3 Input Data.txt | Input | <https://alixpartners.box.com/problem3inputdata> |
| Problem 3 Input Data -- Country Map.txt | Input | <https://alixpartners.box.com/problem3inputdata> |
| README.pdf | User Manual | Included in .zip file |
| worldcitiespop.txt | Input | Must be downloaded from: <http://download.maxmind.com/download/worldcities/worldcitiespop.txt.gz> |
| airports.dat | Input | Included in .zip file, or alternatively, it may be downloaded from: <http://openflights.svn.sourceforge.net/viewvc/openflights/openflights/data/airports.dat> |
| validatecities.py | Script | Included in .zip file |
| unique\_cities.csv | Output | Included in .zip file, or it may be regenerated by running validatecities.py |
| processed\_cities.csv | Output | Download it from: <https://www.box.com/s/4nofvrszf0hprs2jd2c0> or it may be regenerated by running validatecities.py |

# Installation

The Python script which is included in the .zip file is valid under the Python 2.7.x language standard, but is not valid under 3.x (text I/O—a crucial function in this problem context—is somewhat different under the newer standard). Therefore, if you want to actually run the code, please start by making sure that you have a Python 2.7 interpreter installed. If you don't have the 2.7 version, a free distribution may be found at: <https://www.enthought.com/downloads/>.

Once you have Python 2.7 installed, perform the following steps:

1. Unpack the .zip file into a working directory.
2. Download the main input files from <https://alixpartners.box.com/problem3inputdata>
3. Download the third party city spelling validation file (an additional required input) from: [http://download.maxmind.com/download/worldcities/worldcitiespop.txt.gz](http://download.maxmind.com/download/worldcities/worldcitiespop.txt.gz%20) (see <http://www.maxmind.com/en/worldcities> also for additional info).
4. (Optional) Download the “long form” (~200 MByte) sample output file processed\_cities.csv from: <https://www.box.com/s/4nofvrszf0hprs2jd2c0>
5. (Optional) Rename both of the sample output files (unique\_cities.csv and processed\_cities.csv) or else move them to a different directory so that they don't get overwritten when you run the script.

# Execution

The Python script is designed to be simple to run. From the command line, cd to your working directory, and then, assuming that python is in your path and points to version 2.7, type:

python validatecities.py

The script may also be run interactively, e.g. within IDLE, IPython, Enthought Canopy, or similar:

%run validatecities

The script contains some text pattern matching code, which is used to perform spelling auto-correction, but this part of the code unfortunately is not particularly fast. The script takes about 4 hours to run to completion on a 2008-era Windows Vista PC laptop (it will likely run appreciably faster on a more recent vintage Windows 7 or 8 desktop) and provides progress updates every few minutes. It is because of this speed issue that sample output files have also been included as part of the problem solution; please review those as an alternative if that would be preferable to waiting up to four hours for the script to finish. For running on a Windows PC in particular, it is recommended that the user turn off any screen saver and/or other power management functionality designed to cause the screen to go blank after a set period of time without user activity; the anecdotal evidence appears to be that in limiting power consumption, these functions can also cause application code to run more slowly in the background when the computer sits for long periods without interactive use. In Windows, the screen saver options may be accessed via “Start -> Control Panel -> Classic View -> Personalization -> Screen Saver” and the power management options via “Start -> Control Panel -> Classic View -> Power Options -> Change Plan Settings”.

# Output File Format

Both of the output files are formatted identically into 8 columns, the only difference is that in the unique\_cities.csv file, all the rows are unique, whereas in the processed\_cities.csv file, the rows reflect the structure of the original input file, which contained many repeats. The names and meanings of the columns are:

|  |  |
| --- | --- |
| **Column Name** | **Contents** |
| InputCity | City name as given in original input file |
| InputCountryAbbrev | Two letter country code as given in original input file |
| Quality | An integer value (1-9) indicating how reliably the input was able to be pattern-matched against its output |
| OutputCityASCII | The estimated correct city spelling, in upper case ASCII, as provided by the third party city spelling validation file: worldcitiespop.txt |
| OutputCityAccent | The estimated correct city spelling, in mixed case, using a limited palette of foreign characters where appropriate |
| OutputCountryName | The full country name, derived from the two letter input code |
| Latitude | Estimated city latitude |
| Longitude | Estimated city longitude |

The integer values in the third output column, the so-called "Quality" indicator, take on the following interpretations:

|  |  |
| --- | --- |
| **Quality Value** | **Meaning** |
| 1 | Exact match to precisely one city in worldcitiespop.txt spelling validation file |
| 2 | Exact match to multiple cities within the same country (presumably from multiple states, provinces or geographic regions); lat/lon values are selected based on airports.dat which presumably represents the most prominent among them |
| 3 | Exact match to multiple records in worldcitiespop.txt file, but no matches in airports.dat; lat/lon is therefore not uniquely resolvable |
| 4 | Input city approximately matches one unique record in worldcitiespop.txt, except for extra white space and/or stray non-alphabetic characters |
| 5 | Input city matches multiple records in worldcitiespop.txt except for extra white space or non-alphabetic characters; lat/lon values are taken from airports.dat file |
| 6 | Input city matches multiple cities except for extra white space or non-alphabetic characters, does not appear in airports.dat; lat/lon is not uniquely resolvable |
| 7 | Input city name appears to be misspelled; spelling auto-correction algorithm has taken a guess at what was intended |
| 8 | Input city name is completely unrecognizable (typically either blank, street address, numerical postal code, or two-letter country code which is valid, but not the one the user intended) |
| 9 | Invalid two-letter country code |

In general, the higher the “Quality” index, the less reliable the output in columns 4 through 8; i.e., a value of “1” indicates a city name and country pairing which was well-recognized and could be conclusively tied to known latitude/longitude values, while values of 8 or 9 indicate input values which essentially were completely illegible.

# Tips for Viewing the Output File

For a human reviewer, the better output file to look at is unique\_cities.csv. Microsoft Excel is a fairly satisfactory viewing tool. After starting up the Excel application, choose “Data -> Get External Data -> From Text” and select the correct file location. A “Text Import Wizard” popup window should appear; select “Delimited” instead of “Fixed Width”, and then on the next screen choose “Comma” as the delimiter and a single quote mark (i.e., as opposed to double quotes or none) as the text qualifier. OpenOffice Calc also has a similar interface, however it seems to be accessed through the “File -> Open” menu rather than the “Data” menu.

When reviewing the unique\_cities.csv file, please note that the rows are sorted in order of “Quality” (column 3) first, then alphabetically by two-letter country code (column 2) and then alphabetically by city name (column 1). By far the most interesting results to look at are those with a “Quality” value of 7. These generally correspond to inputs which were misspelled or otherwise mangled (for example by swapping the order of two letters, or substituting one for another, or dropping a letter completely), and it is instructive to see how well the spelling auto-correction algorithm did on the really tough cases.

# FAQ

**What is the purpose of the worldcitiespop.txt file?**

It’s a third party file, freely available on the web, which provides the master list of validated city spellings against which each line of the input file is checked.

**O.K., then what’s the purpose of the airports.dat file?**

The names of many large cities often are not unique within a given country. For example, in the U.S., although the city of Dallas, TX, is well known and has a population of over a million people, there is also a Dallas, GA, with a population of around 10,000. Because the input data set for the problem did not include region coding (i.e., information such as state or province) there is no way to know for sure which one was intended, and therefore no way to know for sure which is the correct latitude and longitude to assign to the validated output. The airports.dat file is used as a proxy estimator for city size and prominence; in ambiguous cases with multiple matches, the approach taken within the code is then to select as a “best guess” whichever city was large enough to have its own airport.

**Does the airports.dat file ever fail to return the correct result?**

Yes, the lack of region coding affects the airports.dat file as well. Often, a given city will have multiple entries in the airports.dat file; for example, Chicago, IL has both O’Hare (ORD) and Midway (MDW) as well as some smaller regional airports. Thus, in order to have a single latitude/longitude value associated with a given city, the script must average together the lat/lon values for all airports serving the city. This works just fine until you come to Portland, US: the file contains entries for Portland International Airport (PDX), serving Portland, OR, and also Portland Union Station, a train station serving the same city as well, but on the other hand, there is also Portland International Jetport (PWM), serving Portland, ME. Since the script logic currently averages together all known lat/lon values for each city in the airports.dat file, this approach leads to erroneous lan/lon geocode values for any matches to Portland, US, because it happens to be an unusual case in which *both* the cities in question were large enough to have an airport. The erroneous lat/lon value reported for Portland, US, turns out to be an average of the three unique values associated with PDX, Union Station, and PWM, which actually corresponds to a point (44.92129433, -105.194527) somewhere in northeast Wyoming!

**Could the ambiguity problem in airports.dat be fixed?**

Sure, you could rewrite the code so that it averages together only the lat/lon values for airports which appear to be in relatively close proximity to one another (for example, when averaging, one could require that each constituent airport must lie within +/- 1 degree of its neighbor in order to be included within the average). This would eliminate the problem of *incorrect* lat/lon values associated with the city of Portland, US, but it still wouldn’t take away the fundamental underlying ambiguity issue: i.e., whether to assume Portland, OR or Portland, ME.

For now though, this suggested fix has been left for implementation in a future update.

**What’s the deal with Puerto Rico?**

Puerto Rico is treated as a separate country in the Problem 3 Input Data – County Map.txt file, but as a part of the US in the worldcitiespop.txt file. Thus, all cities in Puerto Rico currently return an “invalid country code” (Quality = 9) result because the country code “PR” does not exist in wolrdcitiespop.txt. It probably wouldn’t be all that hard to fix, but, again, at some point, you do need to cut your effort short somewhere.

**Are there any city names which are particularly frequently misspelled?**

Buenos Aires, Argentina, appears to have a rather astonishing number of incorrect yet still perfectly recognizable variations. The spelling auto-correction code in the validatecities.py script seems to catch a lot of them (these entries are mostly sequential and will all have their validation “Quality” value set to “7” within the unique\_cities.csv output file).

**Are there any other ways that the script could be improved?**

Sure—the best thing to do would be to provide geographic region codes (e.g., state or province names) as part of the input, to make the data less “fuzzy”.

Short of actually improving the input data itself, however (which is really the ideal solution) another good thing to do would be to add processing that recognizes certain common abbreviations, such as “LA”, “NYC”, or “CEDEX” (the latter phrase, which is associated with postal codes, appears in many of the city names in France) or “ST.” (i.e., as an abbreviation for “SAINT”, not “Street”). Also, it would help to add some processing in order to recognize cases where people had inadvertently entered a commonly recognized region code instead of the correct country code, for example: “Birmingham, AL” (which is clearly intended to represent “Birmingham, Alabama, United States” but instead is misinterpreted as “Birmingham, Albania” which doesn’t come close to matching any real city names in Albania at all).

# Author Contact Information

Name: Andrew L. Stachyra

Email: [andrewlstachyra@gmail.com](mailto:andrewlstachyra@gmail.com)

Telephone: 781 354 5658