Field Session Final Report

Fuzzy Matching Profiles

FullContact Inc.

Members

John Honan, Gregory Connelly, Austin Sonju, Kepler Novotny, Daniel Winternitz

Introduction

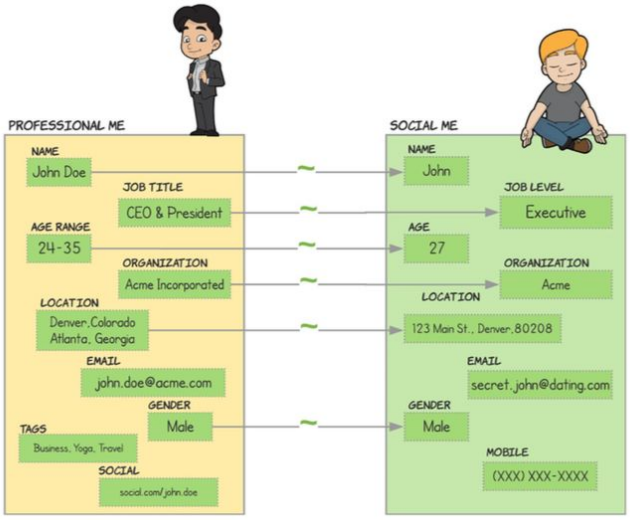
FullContact is a contact management company, whose focus is building meaningful relationships between clients and the people who matter most to them: contacts, connections, or customers. In addition, FullContact uses publicly available datasets to provide information and insights to foster connections with people in more meaningful and authentic ways.

FullContact believes that life is all about people and the relations between them. On their website, under the about us tab, they say “Our core purpose is to make relationships better. We believe that relationships, are driven by trust; trust is driven by empathy and understanding; understanding is driven by insights; and insights are driven by data.”

FullContact works with everyone from an individual who has to manage a few contacts to multinational corporations who have to track and maintain thousands.

Our project is to develop a program that can determine a probability whether two online profiles belong to the same person. These profiles have information such as: name, location, age, etc. This is done using a process called fuzzy matching. Fuzzy matching is a technique that matches two objects even if all of the information isn’t present or exact matches.

Our projects success in identifying matches is measured with precision and recall. Precision is the number of true positives divided by the sum of the true positives and false positives. In the terms of this project, it is the number of correctly identified matches divided by the total number identified matches. Recall is the number of true positives divided by the sum of the true positives and false negatives. In the terms of this project, it is the number of correctly identified matches divided by the total number of matching profiles.



Requirements

Functional Requirements

The code functions overall to take in two profiles as an input and returns the probability that they belong to the same person. This uses a variety of sub functions to process different factors into a final probability:

- A sub-function that compares names of two profiles and returns the probability that they are from the same person.
- A sub-function that compares locations from two profiles and returns the probability that they are from the same person (Eg: One profile with Denver,CO and other with Golden,CO could belong to the same person - one being home and the other being work).
- A sub-function that compares two job titles and their categories which returns the probability that they are the same.
- A sub-function that compares organization titles and their variations which returns the probability that they are the same.
- Other data from online datasets, such as census data by region to supplement/define multivariate probability calculations will supplement this probability as well.
- A merge function brings all of the sub-functions together returning a final probability.

Non-Functional Requirements

- The code is written in Java.
- Git is used for version control.
- The input is a csv file with multiple profiles and each profile contains pertinent information (name, location, etc.)
- Intermediate and final results are returned as probabilities [0-1.0].
- Precision and recall are calculated in each step.
- Code is thoroughly tested, using unit tests that mimic actual usage in production.
- Project can be compiled into a jar file.
- Code should be portable.
    - Able to be integrated with other FullContact products.
    - Able to run on many machines seamlessly.

Potential Risks

There are certain technology risks that have to be considered when writing and using the code. Working with sensitive data is an issue especially with GDPR regulations in place. It is also not uncommon for insufficient information to exist (no name, location, job title, etc.) so there is an issue with always confidently predicting matches in this case. The code is optimized to produce

the best results possible but there will always be some incorrect results (false negative and false positive). There are also some skill risks associated with this project. Statistical methods have to be applied to code effectively and was challenging at times deriving the best/correct equations to use. Correctly documenting the code is a skill everyone actively practiced.

Definition of Done

The project required that we create separate functions to determine probabilities based on individual components (name, location, organization, job) and combine the intermittent probabilities in another function to determine the probability of whether or not the profiles are connected. The completed project consists of a .jar file along with the source code. In addition to the project, we created a presentation to demonstrate it to the FullContact engineering team. All parts of the project are covered by unit tests. All parts are documented internally with the why, what, and how of that component's operation. The precision and recall was documented for each field comparison and the probability merge functions throughout the project.

System Architecture

Our project is designed to take in a series of pairs of profiles which contain basic information about an individual. This information includes a person's name, age, gender, location, the organization they work for, and their job title. Our project uses this information to determine whether or not the profiles in each pair match. We chose to only look at two profiles at a time so that we could quickly run the program on one machine and not have memory issues when attempting to work with several profiles at once.

Since we are only comparing two profiles at a time we needed to structure the input file to reflect our design decision. Our input comes from a csv file that contains pairs of profiles for the program to compare. The first item in our appendix is a screenshot of a sample input file. From this image, it can be seen that the first line is the header, containing the names of all the different attributes for the profiles. After the header file we have the different profiles. These profiles are in pairs, which the program looks at to calculate match probabilities. It compares the first profile with the second, the third with the forth, and so on until we reach the end of the file.

Determining whether each pair of profiles match is done using a series of functions called comparators. These comparators look at different attributes of the profile and determine how likely it is that the attribute matches across the the different profiles.

The first comparator is the name comparator. As the name suggests, this is the function that determines if the names are the same across the different profiles. This is not as simple as just comparing two different strings. We use data from around the world to determine which names are the most popular by region and then use that information to determine the likelihood that the names match. We also needed to determine if it is possible that one profile is using a nickname

while the other uses a full name. If one profile has the first name Joe and the other is Joseph then it is still possible that the profiles match since Joe is sometimes interchangeable with Joseph.

The location comparator is similar to the name comparator. We can not just compare strings to determine if two profiles are in the same location. It is possible for a person to have profiles from two different locations. A person could live in Denver and go to school in Golden, for instance. This could result in two profiles, one with a location in Denver and the other with a location in Golden, corresponding to the same person. The comparison is done using a dataset that contains the global population distribution as well as the latitude and longitude for every city. This allows us to determine the probability that the locations match based on the population distribution as well as the distance between the two locations.

Comparing organizations across the different profiles is a very different process. The key to this process is normalization. We needed to ensure that if the same organization was represented differently across two profiles, they can still be matched as the same organization. It is entirely possible one profile working at IBM and another profile working at International Business Machines represent a match. These are simply different ways to represent the same company, so we had to ensure we accounted for this in our algorithm.

The fourth is the gender comparator. This is the simplest of the comparators because the listed genders should be exact matches. Either the genders match across the profiles or they do not match, and based on FullContact's dataset, this is normalized to allow direct string comparison.

Age is not always disclosed exactly; some profiles will have an age while some only have an age range attribute associated with them. We needed a way to check whether these attributes are the same across the different profiles. It is entirely possible that one profile could have an age and the other could only have an age range, so we needed to ensure that the ages matched with the age range to determine the possible connection between the profiles.

The final comparator looks at a person's job title and decides if the job titles are the same across the different profiles. Similar to the organization comparator, the job comparator needs to account for different representations of the same job. It possible that one profile could say 'C.E.O.' and the other simply lists 'executive'; these are the same job title despite being represented differently.

Each of the above comparators determines the likelihood that its associated attribute is the same across the two profiles. After gathering all of that information, we need a way to combine all these probabilities into something useful. This is done in something we call the merge function. This takes all the values returned by the comparators and determines whether the two profiles are the same or not, by calculating the probability that the profiles match.

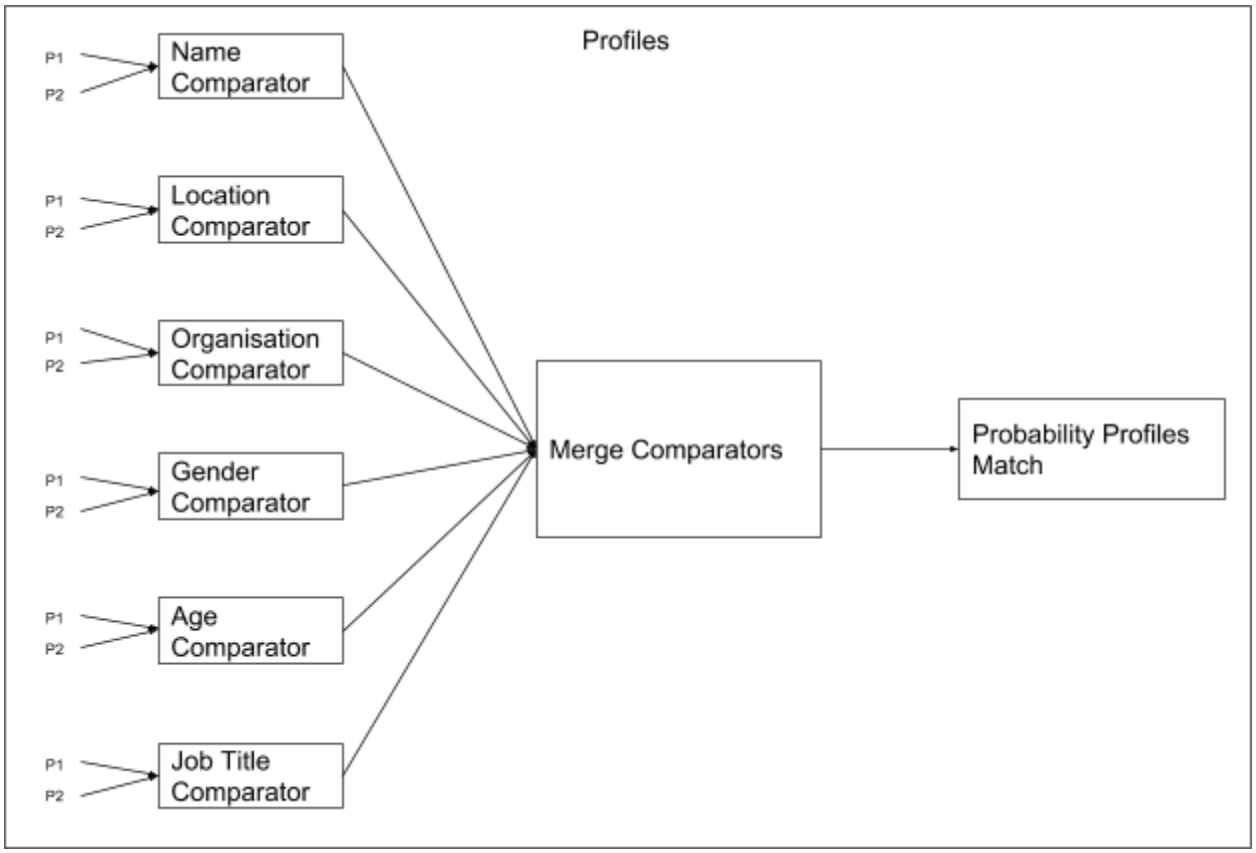
A simple diagram of this overall project structure is included below within the Technical Design specifications.

Technical Design

Data passing/management

To handle the data after it was read in, Profile objects were created, which each hold all the information about an individual profile. Two consecutive profiles were read in and used to create a profile comparator object, which holds information about the two potentially matching profiles. This object performs each comparison function, testing the probability of the items matching, before using these results in the merge function. The merge function then uses mathematical statistics procedures to calculate a final probability.

This data management scheme was made to minimize unnecessary data passing, and allow objects to hold all information individually that they would need. The downside is a slight decrease in modularity; swapping out the function of any comparator must be done from the function within the object itself. While this makes an arguably bloated class structure, the comparisons against profile data as class attributes reduces the data handling, simplifying the process of passing pieces through this structure.



Handling US vs non-US locations

Our location and name comparators depend on various data files in order to produce meaningful results. We were able to find more meaningful data for the United States than for other countries.

The dataset for locations inside the US has every zip code, along with the city, state, location, and population for the zipcode. Item 4 in the appendix shows a piece of the csv used in the location comparator for locations inside the United States. Of the data in the csv, the latitude/longitude locations and populations are very helpful in our calculations. Given two locations we can compare their latitude and longitude locations to determines how close they are. The data allows us to determine the population of cities the profiles are from, and the populations of the areas around the cities. This dataset is complete for every zip code in the US, allowing us to calculate accurate probabilities for all profiles located inside the US.

The dataset for world wide locations can be seen in Item 6 in the appendix. It has data broken down by city, not zip code. In addition, the population info is incomplete. It has the population for large cities, which we can use to create relatively accurate calculations if the profiles are located in large cities. For profiles in smaller cities, our calculations must rely purely on the distance between the profiles. As a result, calculations made using this dataset are not as precise as those using the former dataset.

We also had more meaningful name data for locations inside the US than those outside. We have data on the frequency of each name for each state in the US. For places inside the US, we can calculate the number of people in the state with a specific name. This information is used to calculate the probability the two profiles belong to the same person, based on the relative frequency of the name.

For countries outside the US, we do not have data on the frequency of each name. We only have a list of common names in each country. Rather than knowing the frequency of each name, like we do for profiles in the US, we only know whether the name is common for profiles outside the US.

In order to maximize accuracy, profiles located inside and outside the United States are handled using separate processes, allowing us to utilize the more complete datasets when possible.

Reducing File I/O

The project requires extensive use of datasets in order to make meaningful calculations. Reading in the datasets can be time extensive, most notably reading in the 180 MB “Global Location Data.csv” file used by the location comparator for profiles located outside the United States. Initially, everytime a profile was located outside the US, the program would comb through the file finding every city located in the same country as the profiles, store the info of each city in an

object, and store all the objects in an array list. This process would be repeated for every set of profiles located outside the US, and resulted in a lot of time being spent on file I/O. The code for this implementation can be seen in Item 3 in the appendix.

In order to speed up the program, we changed the program so that after the information for the cities in a country are read in and stored in the array list, the array list is stored as the value of a map, with the country being the key. This removed the unnecessary file I/O of reading in the same information multiple times. The code for the new implementation can be seen in Item 4 in the appendix.

Decisions

Language Choice

When our project began we were given free reign to decide what language we wanted to use. However, our client prefered that we use one of the JVM languages. The two he talked about were Scala and Java. Our client did not care which of these two languages we used (because the source code underneath is the same). In the end we decided to use Java over Scala. We chose Java because everyone in the group was familiar with Java and how to use it. This does not matter to our client because the source code for Java and the source code for Scala are the same so our client can easily convert our Java code to Scala.

Data Design

This project uses a lot of external data. The data is saved in several .csv files. The reason we choose to use csv was because we needed to combine a lot of smaller datasets to make the large one that was used, and because .csv files are easy to parse and extract information from in code. This was especially true for the location matching dataset; we needed to take several separate datasets and combine them in to one in order to get a dataset that would have the location (latitude and longitude), city name, state name, zip code, and the population broken down by zip code. We did not use java to combine the datasets, but we chose to have the datasets saved as a csv to allow us to easily import the data in to the java program.

The other way we stored our data is through json files. These files would then be imported into the project. Our field session client provided, for us to use, a python file that contained a dictionary. This dictionary had a list name, for each name there were variants of each name and a list of common nicknames. As an example for the name John the list would contain Jon and Jack, since Jon is a variant of John and Jack can be interchanged with John. This information is essential for getting the name comparator to work so we needed to find a way to move this into our project. We ended up using python to convert the dictionary to a json file. We then imported the json file into our program utilizing the gson library.

Reuse/Library Choice

We are writing this project from scratch. This means that when we started the project we started with an empty java file. This file did not contain anything for the project. We met with our client and they provided us with the overall concept for the project and we started from there. Due to this project structure, we did not reuse any significant amount of code from any source.

This project does use some java libraries. The first that we used is the Gson library developed by Google. This library assists with reading in json files and eventually converting this format to a java object. This library was used to import the json file described above. The JUnit library was also imported to be used with a suite of test to ensure the code was fully functional.

Implementation Discussion

- This project is broken down into two major parts: merging and comparing.
- Comparing is in reference to the different comparators, the four major ones we have written for the program are the Name Comparator, Location Comparator, Organization Comparator, and Job Comparator.
    - **Name Comparator**: This is a function that returns the probability that the name field of the two profiles are the same.
    - **Location Comparator**: This function returns the probability that the location fields are the same.
    - **Organization Comparator**: Similar to the other two functions this function returns the probability that the organization field/fields are the same across the different profiles.
    - **Job Comparator**: Returns the probability that the job titles are same or similar based on the raw string and category of the job.
    - **Simple Comparators**: These are comparators which we wrote but they are not that complicated. These comparators can return 3 different values, based on whether the attributes are matching, non-matching, or inconclusive/missing data.
- The merging is what pulls the different comparators together. This is the function that is responsible for assigning weights to the probabilities, meaning that the results from each comparator does not have the same weight when determining the final probability. If we have two profiles then the probability that the names match is more important (and should therefore have a higher weight) than the probability that the exact locations match across the profiles.
- Note: For the above, if we have values missing or not specified in the input then the comparator returns 1. Due to the mathematical structure of the calculation, comparators with a value of 1 do not affect the final merge calculation.

Results

Missing Features

We were able to implement all the features we hoped to at the beginning of this project. We also found time to implement more features that were not outlined at the beginning of this project. Originally we only thought we would be able to get the name comparator and the location comparator done. However we were able to finish both of the above comparators and then some other ones as well. We were able to finish the organization, gender, age, and job title comparators.

Performance Testing Results

In this project the precision is more important than the recall. FullContact can use a human in the loop verify the matches generated by this algorithm before implementing them in production. They want the matches sent to the human in the loop to have as high a confidence as possible. In testing we found we had a .919 precision and a .819 recall. These are results that we are happy with, but that does not mean that this result cannot be improved upon later. We also found that the project used less than 2 gigabytes of memory to run. There is a lot of data going in to this project and memory use was a issue that we ran into while working on the project.

Summary of Testing

We have done a lot of testing of our code. Every function has tests associated with it to ensure that the function is working correctly. We also implemented tests to ensure we were reading in files correctly. Finally we made a dataset containing 1,000 different profiles (500 pairs) to use as a testing dataset. It was compiled from a sample dataset provided by our client. This allowed us to test the performance of our function and compare it with the actual results from the table.

Usability Tests

Since our project is a proof of concept there was not a lot of usability tests being performed. However we did ensure that using the project was simple (assuming you installed the correct libraries). The main issue we ran into when it came to usability resulted from incorrectly formatted input files. Our client also specified the use of a java executable file so we needed to test and ensure that file was working correctly.

Future Work

There is a lot of work that can still be done to this project. The most obvious is to add more comparators. The more comparators that you have the more accurate the results are going to be. There is a lot of organization that can be done to the project. The way we have things set up now, the majority of our codebase exists within a single class. In retrospect, it may have been easier to divide this across multiple files and classes to increase the level of abstraction for each part of the program. This would allow future developers to more easily swap out components of the code with an improved section if future development resulted in better versions of each module.

Machine learning was also discussed as a potential option to implement with the merge function. Unfortunately, we did not have enough data to execute this correctly as test data had to be manually generated, as sample profiles did not inherently contain whether the profiles match. We therefore had to use a with a more statistically driven approach. In the future, machine learning would be a very effective implementation to make this piece of software even more accurate.

Lessons Learned

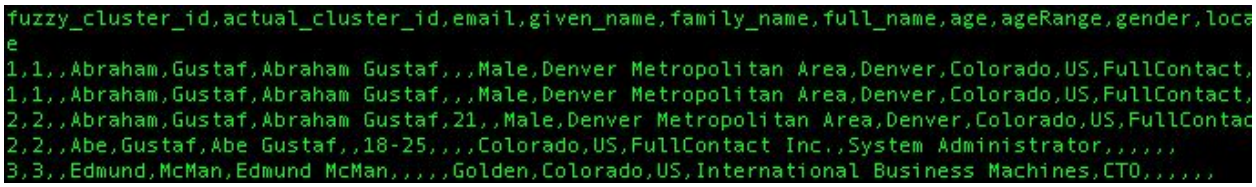
For most of us in this group this was the first time we worked at a company. As a result we gained good experience about the daily life of a software developer outside of school and what to expect when we graduate and get a job. Specific aspects such as working in teams and utilizing the agile process were definitely emphasized.

We also worked with a variety of data of different sizes and learned that data can often be incomplete or unreliable therefore it is important your code knows how to deal with various situations.

As part of this project, we were able to attend the Connect 2018 conference. FullContact invited us to attend this conference and we learned a lot about the field. There were guest speakers from around the country. The speakers who presented at this conference were men and women who are at the top of the computer science field. It was really incredible to listen to their presentations and see how the skills we are learning here at Mines will be applied when we start to work in the tech industry.

Appendix

Item 1: Screenshot of a input file.



```
fuzzy_cluster_id,actual_cluster_id,email,given_name,family_name,full_name,age,ageRange,gender,loca
e
1,1,,Abraham,Gustaf,Abraham Gustaf,,,Male,Denver Metropolitan Area,Denver,Colorado,US,FullContact,
1,1,,Abraham,Gustaf,Abraham Gustaf,,,Male,Denver Metropolitan Area,Denver,Colorado,US,FullContact,
2,2,,Abraham,Gustaf,Abraham Gustaf,21,,Male,Denver Metropolitan Area,Denver,Colorado,US,FullContac
2,2,,Abe,Gustaf,Abe Gustaf,,18-25,,,,Colorado,US,FullContact Inc.,System Administrator,,,,,,
3,3,,Edmund,McMan,Edmund McMan,,,,,Golden,Colorado,US,International Business Machines,CTO,,,,,,
```

Item 2: Original function for reading in the location data for outside the US

```java
public void readInGlobalDataset(String CountryCode) {
    InternationalCityInfo myRow = new InternationalCityInfo();
    globalCities.clear();
    try {
        //Set up file reader
        BufferedReader br = new BufferedReader(new FileReader("src/resources/Global Location Data.csv"));
        String line;
        line = br.readLine(); // eats the header
        //Read in each line
        while ((line = br.readLine()) != null) {
            myRow = new InternationalCityInfo();
            String[] info = line.split(",");
            myRow.setCountry(info[1].toLowerCase());
            myRow.setCity(info[2].toLowerCase());
            myRow.setAccentCity(info[3].toLowerCase());
            myRow.setRegion(info[4]);
            if(!info[5].equals("")) {
                myRow.setPopulation((int) Double.parseDouble(info[5]));
            } else {
                myRow.setPopulation(-1);
            }
            myRow.setLat(Double.parseDouble(info[6]));
            myRow.setLon(Double.parseDouble(info[7]));

            if (myRow.getCountry().equals(CountryCode))
            {
                globalCities.add(myRow);
            }
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Item 3: Revised function for reading in location data for outside the US

```
public void readInGlobalDataset() {
    if (globalCityMap == null) {
        globalCityMap = new HashMap<String, List<InternationalCityInfo>>();
        InternationalCityInfo myRow;
        try {
            //Set up file reader
            BufferedReader br = new BufferedReader(new FileReader("src/resources/Global Location Data.csv"));
            String line;
            line = br.readLine(); // eats the header
            //Read in each line
            while ((line = br.readLine()) != null) {


                myRow = new InternationalCityInfo();
                String[] info = line.split(",");
                myRow.setCountry(info[1].toLowerCase());
                myRow.setCity(info[2].toLowerCase());
                myRow.setAccentCity(info[3].toLowerCase());
                myRow.setRegion(info[4]);
                if (!info[5].equals("")) {
                    myRow.setPopulation((int) Double.parseDouble(info[5]));
                } else {
                    myRow.setPopulation(-1);
                }
                myRow.setLat(Double.parseDouble(info[6]));
                myRow.setLon(Double.parseDouble(info[7]));

                if (!globalCityMap.containsKey(myRow.getCountry())) {
                    globalCityMap.put(myRow.getCountry(), new ArrayList<InternationalCityInfo>());
                }
                globalCityMap.get(myRow.getCountry()).add(myRow);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Item 4: Piece of US location data

	country code	zip code	city	state	state abriavation	county	county code	lat	lon	accuracy	population
0	US	99553	Akutan	Alaska	AK	Aleutians East	13	54.143	-165.7854	1	1027
1	US	99571	Cold Bay	Alaska	AK	Aleutians East	13	55.1858	-162.7211	1	160
2	US	99583	False Pass	Alaska	AK	Aleutians East	13	54.8542	-163.4113	1	35
3	US	99612	King Cove	Alaska	AK	Aleutians East	13	55.0628	-162.3056	1	938
4	US	99661	Sand Point	Alaska	AK	Aleutians East	13	55.3192	-160.4914	1	978
5	US	99546	Adak	Alaska	AK	Aleutians West ((	16	51.874	-176.634	1	326
6	US	99547	Atka	Alaska	AK	Aleutians West ((	16	52.1961	-174.2006	1	61
7	US	99591	Saint George Isla	Alaska	AK	Aleutians West ((	16	56.5944	-169.6186	1	102

Item 5: Piece of data file from outside the US

	Country	City	AccentCit	Region	Populatio	Latitude	Longitude
0	ad	aixas	AixÃ s	6		42.48333	1.466667
1	ad	aixirivali	Aixirivali	6		42.46667	1.5
2	ad	aixirivall	Aixirivall	6		42.46667	1.5
3	ad	aixirvall	Aixirvall	6		42.46667	1.5
4	ad	aixovall	Aixovall	6		42.46667	1.483333
5	ad	andorra	Andorra	7		42.5	1.516667