MIDS W205

|  |  |  |  |
| --- | --- | --- | --- |
| **Lab #** | 10 | **Lab Title** | OpenRefine -- Introduction |
| **Related Module(s)** | 10 | **Goal** | Get you started on OpenRefine and Edit Distance |
| **Last Updated** | 9/27/15 | **Expected duration** | 60 minutes |

# Introduction

This Lab has 3 parts. The first two are involves using OpenRefine to clean up some data files. The third one involves calculating the Levenshtein distance between two strings.

OpenRefine is an open source tool for working with bad data. In this Lab we will give you a quick tour of how you can use it to clean data. This is purposely a short introduction to just get you introduced to the tool. If you want a more comprehensive tutorial you can follow any of the tutorials listed in the resources section. The second part of this lab involves understanding Levenshtein distance calculation using the dynamic programming method.

For the OpenRefine portion we will be using two data sets. One data set has earthquake data and one contains customer complaint data. You can access the dataset from the Github (both datasets are available if you clone or pull) or use the links in the text below. The links are also in the resources sections later in this document.

The first data set contains customer complaints; you can download that data set [here](https://github.com/UCBerkeleyISchool/w205labsexercises/blob/master/l%20ab_10/dataset/Consumer_Complaints.csv).

The second data set is the eq2015 data set which data about earthquakes of magnitude 3 or more during the first 6 months of 2015. You can download the Earthquake data set [here](https://github.com/UCBerkeleyISchool/w205labsexercises/blob/master/lab_10%20/dataset/eq2015.csv). You can fine an earthquake data attribute glossary [here](http://earthquake.usgs.gov/earthquakes/feed/v1.0/glossary.php#net). OpenRefine is to a large extent menu driven. But it also allows you to use a language for doing certain types of transformations.

# Open Refine

The basic idea in OpenRefine is that you should think of exploring your data in terms of patterns, called facets. Facets helps by characterizing data and give you an overview of value ranges, missing values etc. There are a number of facets for different data types as well as plots such as scatter plots. Once you understand the data you can transform the data using pattern matching and transformations. To support this OpenRefine has functions for doing transformations of data. These transformations can be expressed in the language GREL although there are a few other options as well. As an example, you can decide to create a new column based on an existing column but with a transformation applied to the data. GREL allows you to match in regular expressions, but also do common operations like trimming blanks, splitting strings etc. In addition it has control structures such as if-statements. You can even have OpenRefine call out to URLS and insert the results in a column. OpenRefine also support fuzzy matching (clustering) of attribute values. It will suggest values to merge and will let you choose which one you like to use. You can adjust the way clustering works using parameters such as radius and character block matching.

# Instructions, resources and prerequisites

For Step-1 and Step-2 install OpenRefine from [here](http://openrefine.org/). The lab describe a number of commands to try, but is also poses a few questions we like you to consider and experiment with. For the submissions you should answer the SUBMISSION questions embedded in the Lab description.

For Step-3 make sure you have a working Python installation.

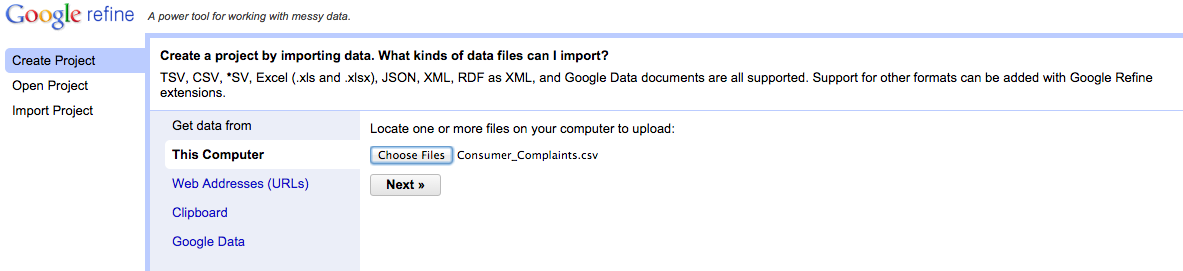
Below are a number of resources that are referred to in this lab, or may be of general interest to you during or after this lab.

|  |  |
| --- | --- |
| **Resource** | **What** |
| http://openrefine.org/ | This is where you download OpenRefine. |
| http://arcadiafalcone.net/GoogleRefineCheatSheets.pdf | A short description of OpenRefine commands. |
| http://enipedia.tudelft.nl/wiki/OpenRefine\_Tutorial | Another tutorial on OpenRefine. |
| http://davidhuynh.net/spaces/nicar2011/tutorial.pdf | Another tutorial on OpenRefine. |
| http://schoolofdata.org/handbook/recipes/cleaningdatawithrefine/ | Programming guide for the Spark Context object. Here you can find actions available on the Spark Contexts. |
| https://github.com/OpenRefine/OpenRefine/wiki/General-Refine-Expression-Language | GREL is the language used in OpenRefine for data refinements. This is a reference guide for the GREL language. |
| http://earthquake.usgs.gov/eart | Explanation of the Earthquake data. |
| https://pypi.python.org/pypi/python-Levenshtein/0.12.0 | A Levenshtein module you can use to check your results in a Python shell. |
| https://github.com/OpenRefine/OpenRefine/wiki/Clustering-In-Depth | A good quick read on some clustering methods. |
| Earth Quake Data set | [https://github.com/UC­Berkeley­I­School/w205­labs­exercises/blob/master/lab\_10 /dataset/eq2015.csv](https://github.com/UCBerkeleyISchool/w205labsexercises/blob/master/lab_10%20/dataset/eq2015.csv) |
| Earthquake Data Glossary. | <http://earthquake.usgs.gov/earthquakes/feed/v1.0/glossary.php#net> |
| Customer Complaints Data. | https://github.com/UC­Berkeley­I­School/w205­labs­exercises/blob/master/l ab\_10/dataset/Consumer\_Complaints.csv |

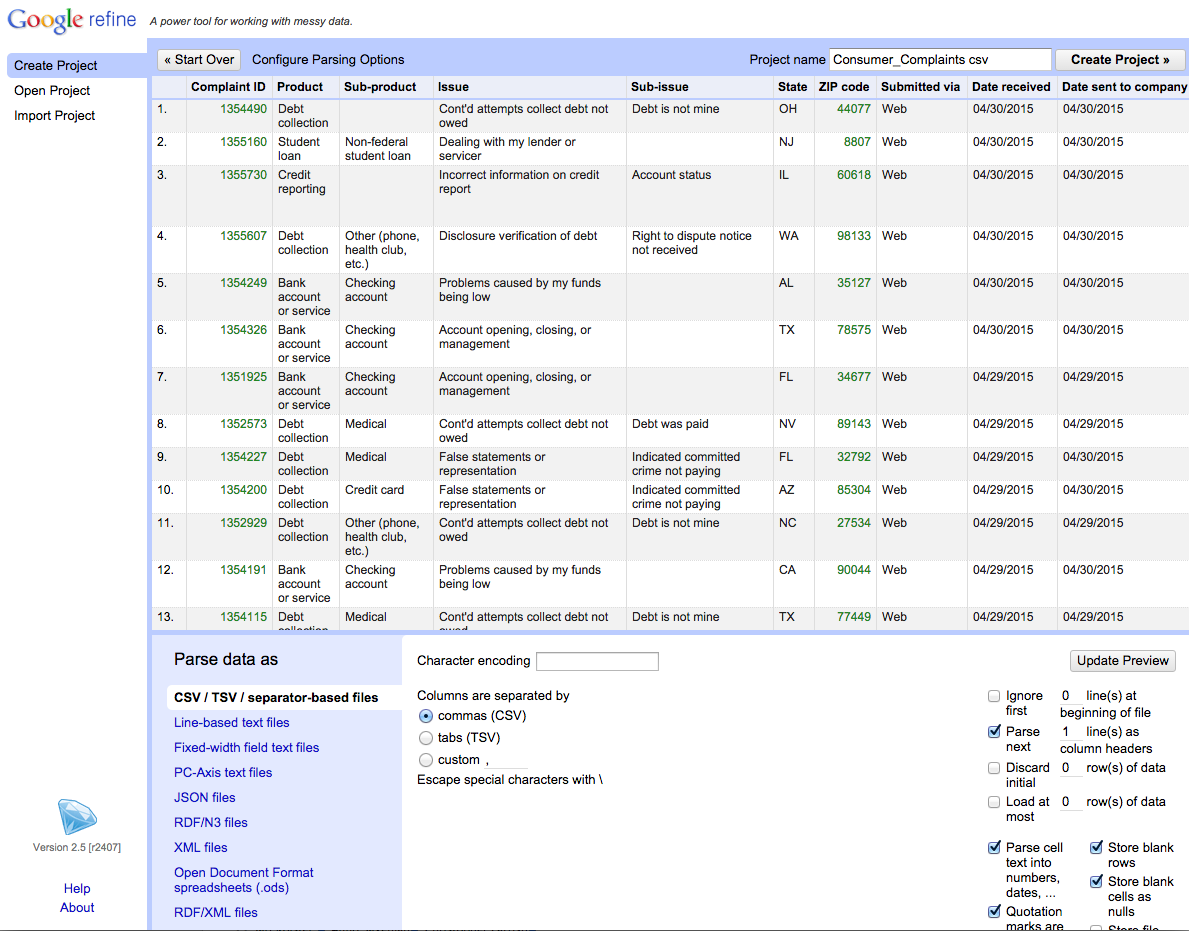
# Step-1. Wrangling the Customer Complaints Data

## Uploading data

After you started OpenRefine you can pick a data set. For this first step choose the Customer Complaints Data set (dataset/Consumer\_Complaints.csv)



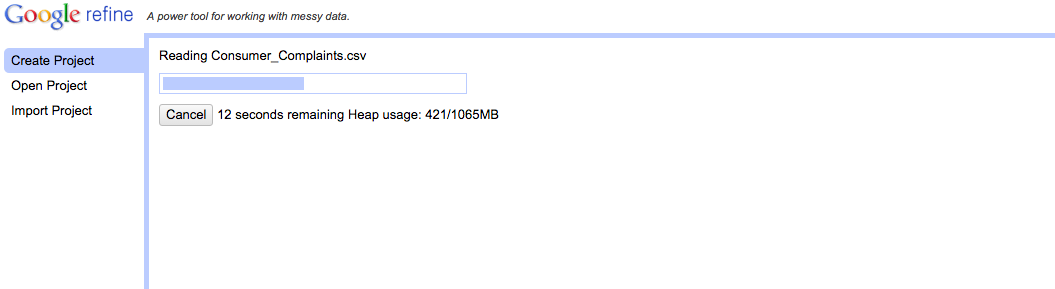
Once the data is read you can inspect it. In this case it looks ok. But lets say that it would have been tab separated rather than comma separate, then OpenRefine would not have identified the structure correctly. You have the opportunity to look at the data here and confirm it is ok. Since we think it looks good and we will now click the “Create Project” button.



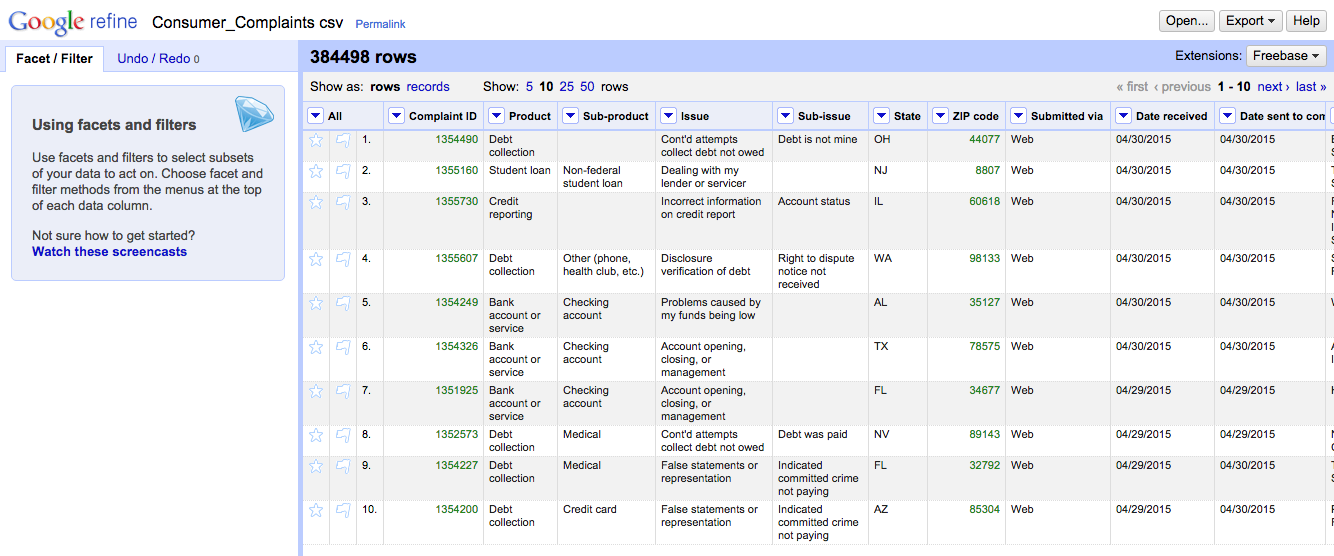
Also note that we specified that the first line should be parsed as column headers.

## Creating a project

Next creating the project can take a little time since there are more that 300,000 lines in this file.

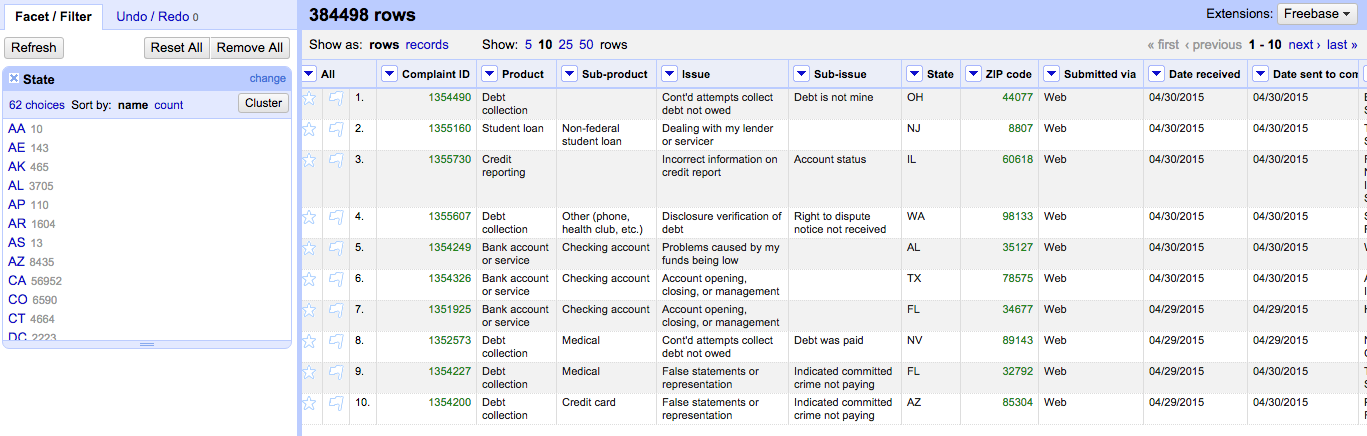


Once the project is created you can see that it has 384498 rows.



## Check states with text facet

If you select text facet for ‘State” attribute you will see a summary in the left column pane. It indicates you we have 62 different state values. Try to figure out why.



SUBMISSION 1: How many rows are missing value in the “State” column? Explain how you came up with the number?

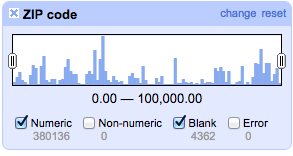
## Checking zip codes

Try the text facet on “ZIP code”, what happens? You can see that there are 24748 different zip codes in this data set. Is that reasonable? Eye ball the data, does all zip codes look valid? You may need to research valid zip codes on the internet to determine if the values are reasonable.

Now try the numeric facet. With the numeric facet the zip code attribute is treated is a numeric value. What would you say the scalar type is for zip codes, can be treated as a numeric attribute? Histogram below shows the distribution when the attribute is treated as numeric.

By unchecking numeric you can get a list of row that are missing.

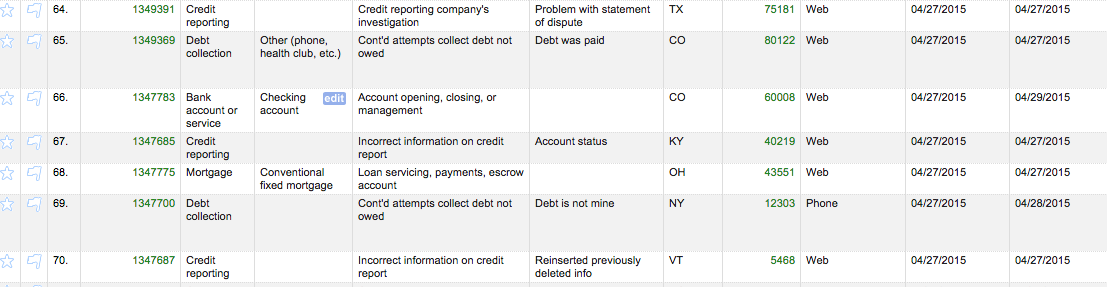
SUBMISSION 2: How many rows with missing zip codes do you have?



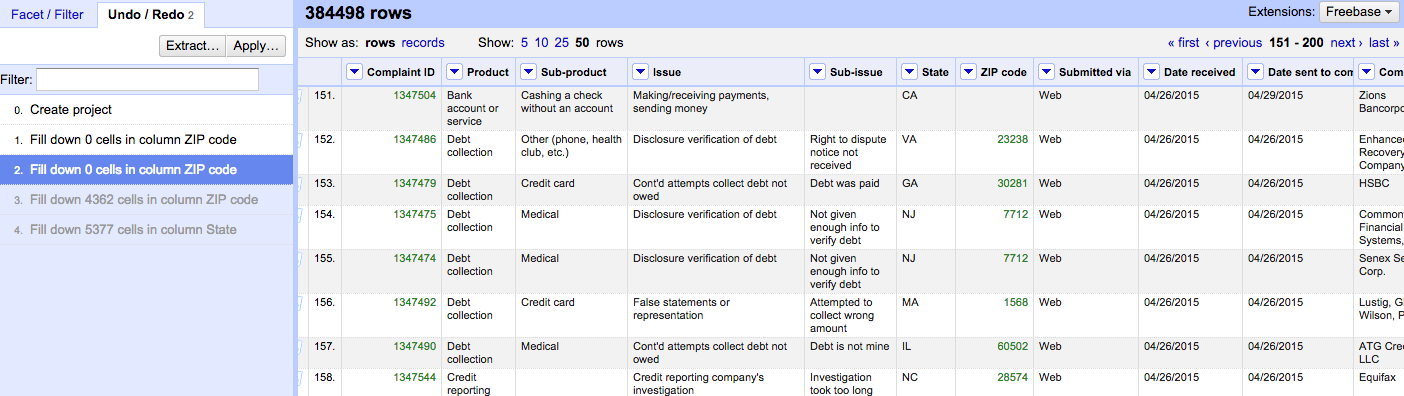
One way of filling in missing values is to take the previous value and use that to set subsequent empty cells. In OpenRefine it is called fill-down. Find a row that is blank. Apply fill down to the fill down by:

Edit Cell->Fill Down

What happened to the empty cell? Is this a valid way filling in missing Zip codes. Can you think of a better way?



If you need to undo the operation, switch to the Undo/Redo tab. Select the previous state for the data. In this example I went back to state 2. As you can see in this screen shot row 151 has a missing zip code, so presumably the fill downs for Zip code and State has been un-done. Observe that the list in Undo/Redo can look different for you if you have been issuing more or different commands than we have so far in this lab.



Lets create a new column called “ZipCode5” with all zip codes that contain 5 digits preserved. All other rows should have the zip code 99999. (Technically speaking the 4-digit zip codes may be valid zip codes; we do this more to illustrate transformations).

Transformations are generally expressed in some language. OpenRefine supports a few alternative languages for transform, we will be using GREL. You can find a link to a language reference in the resources section. For this simple transformation we will be using an if-statement.



For the ZIP code column select:

Edit Column -> Add column based on this column.

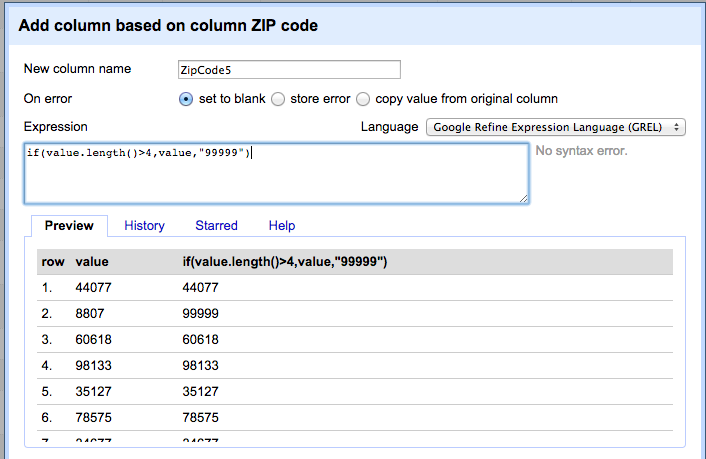
You will get the dialogue below. Insert the name of the new column and the expression:

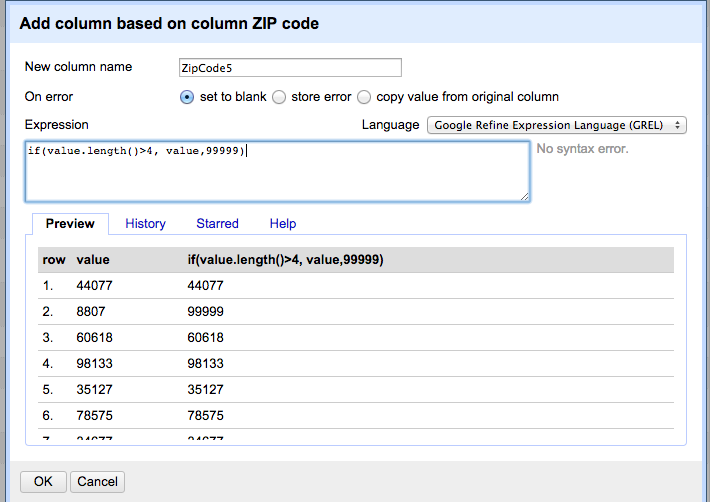
If(value.length() > 4, value, “99999”)

This expression states that if the length of value is more than 4 insert value, otherwise insert the string “99999”.

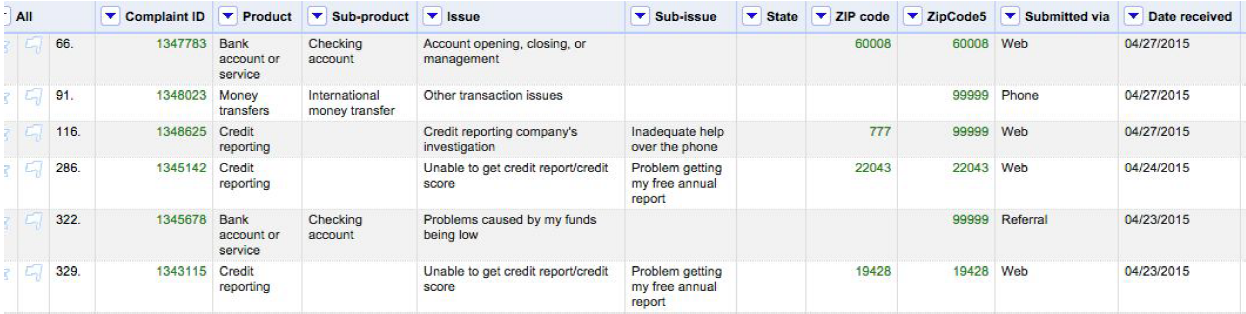
Look at the result, this it do what you wanted? What seems to be wrong with that? What happens if you instead insert a numeric value using the following expression?

If(value.length() > 4, value, 99999)





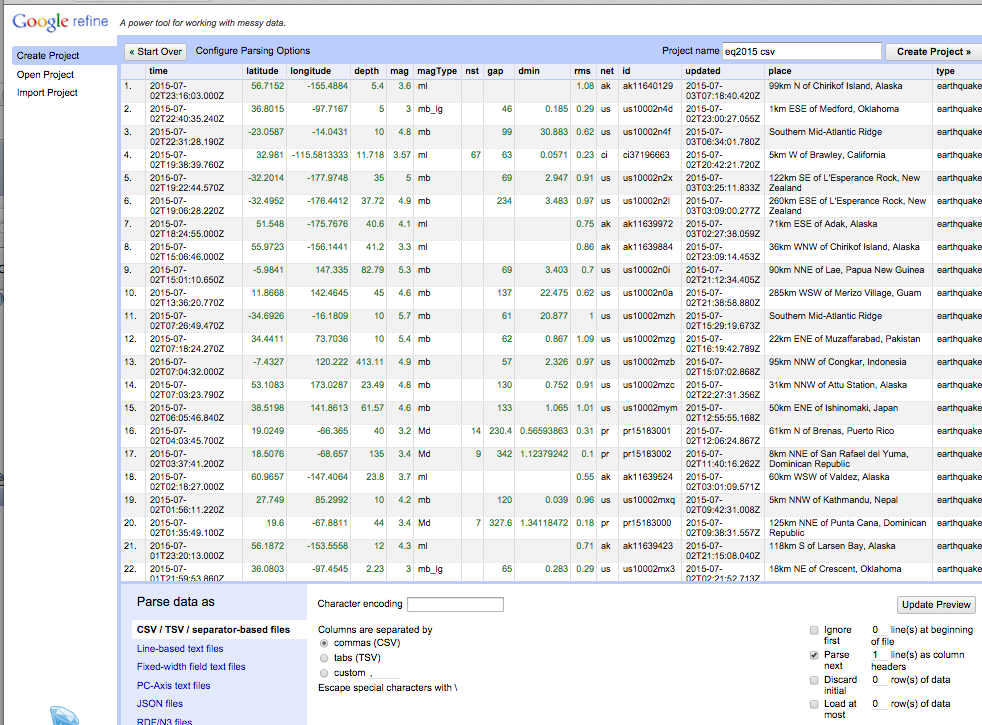
You should now have the same type for all cells in the created column. As an example the result should look something like the following.



SUBMISSION 3: If you consider all zip codes less than 99999 valid zip codes. How many valid and invalid zip codes do you have respectively?

# Step-2. Cleaning up eq2015 Data.

Upload the data file dataset/eq2015.csv. After you checked that the data looks ok, create the project.



As you can see the column “nst” is missing quite a few values. Look up the “nst” attribute in the glossary. What would happen if we just ignored row with missing values? Is there an obvious strategy for filling in the missing values? What would you suggest we do with the column?

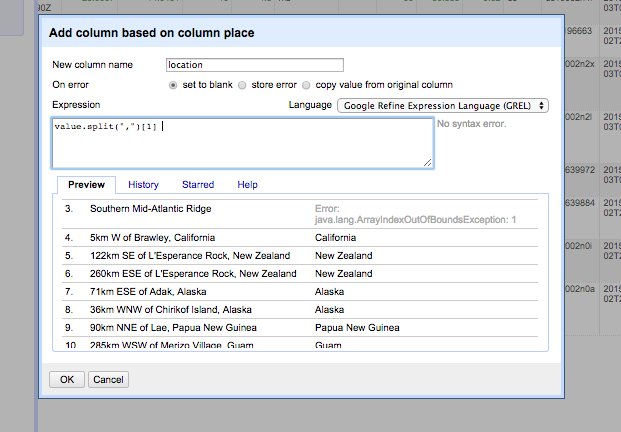
Next we like to extract an approximate area from the “place” column. We would like to have a State or Country, and we like to store that information in a separate column we like to call “location”.

As we review the “place” column we notice that the cell seems to be consisting two commas separated components. The components are: a direction and distance; and a general location.

Select the command:

Edit Column -> Add Column based on this column

You should see a pop-up dialogue box looking as follows. We typed in the column name of the new column “location”



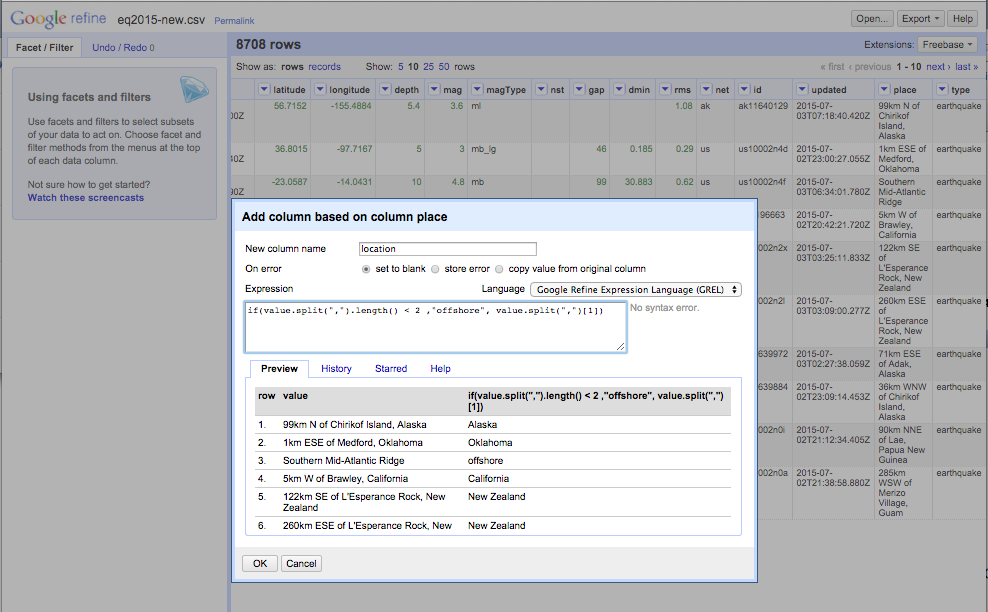
Since we noticed the cells have two comma-separated components, and the second is a location we defined the following expression.

value.split(",")[1]

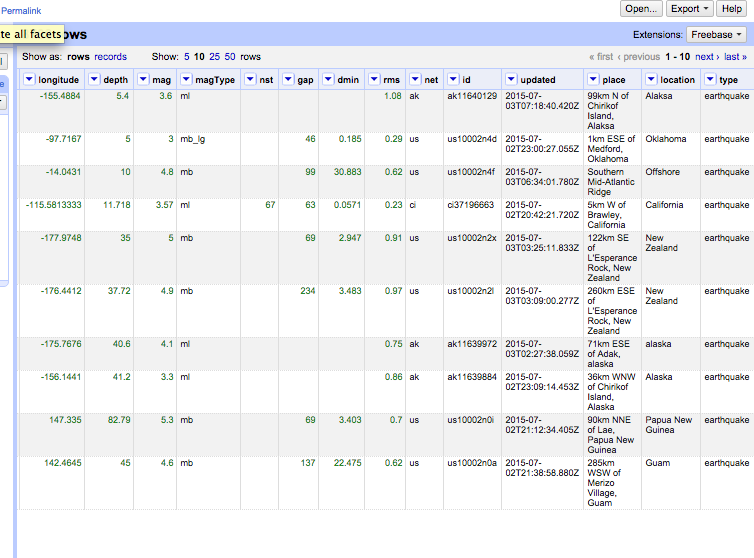
But as you probably notice that did not work so well. In fact, not all cells have the two components. If you look at the data more closely it seems that if the place was offshore the location component was missing. So we modify the expression as follows.

if(value.split(",").length() < 2 ,"Offshore", value.split(",")[1])

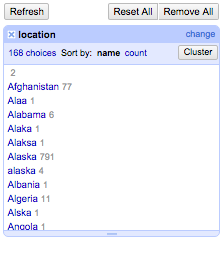
If only has one component we assume it was Offshore and put that value in the location column.



Check the resulting data. Does it seem reasonable, or are more adjustments needed?



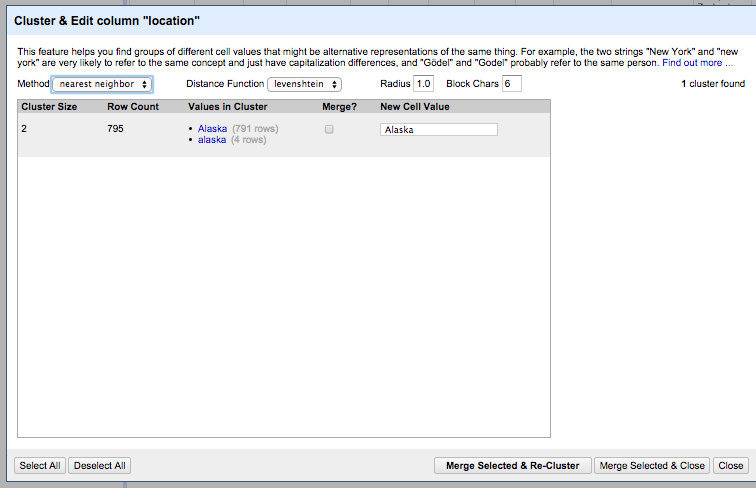
Check the value by using a text facet on the column. You may notice that there appears to be multiple strings that look like Alaska, but appears to be misspelled.



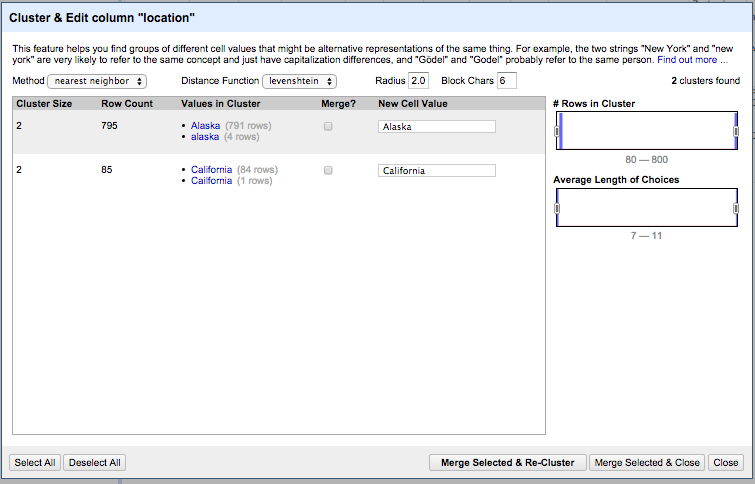
Clustering may help us detect more of these kinds of situations. Run clustering by pushing the Cluster button on the facet or use the column pop-up menu and select:

Edit Cell->Cluster and edit

Try key collision what do you see? Try nearest neighbor, Levenshtein what do you see? You can change the parameters such as Radius and Block Chars. Radius provides a threshold for how close (in terms of distance measure) the strings should be to be considered representing the same entity. The Block Char parameter may behave a little counter intuitive. Blocking defines blocks within which the string distance method is applied. It helps with scalability, as we will not compare strings cross the whole data set. OpenRefine blocking parameter defines the size of a substring S, such that all strings that share S will be in a common block. So a smaller S will likely result in bigger blocks and more computation required.



Change radius to 2.0, what happens?

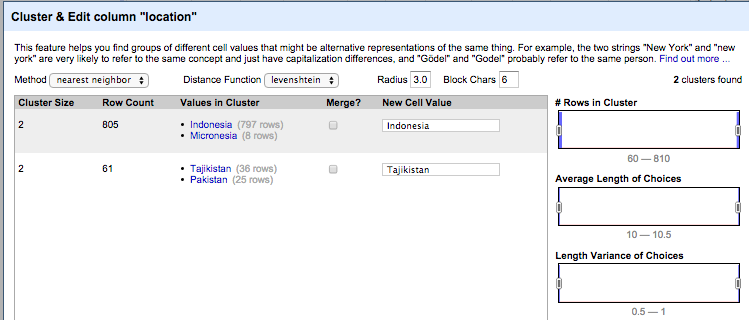


SUBMISSION 5: Change radius to 3.0, what happens? Do you want to merge any of the resulting matches?

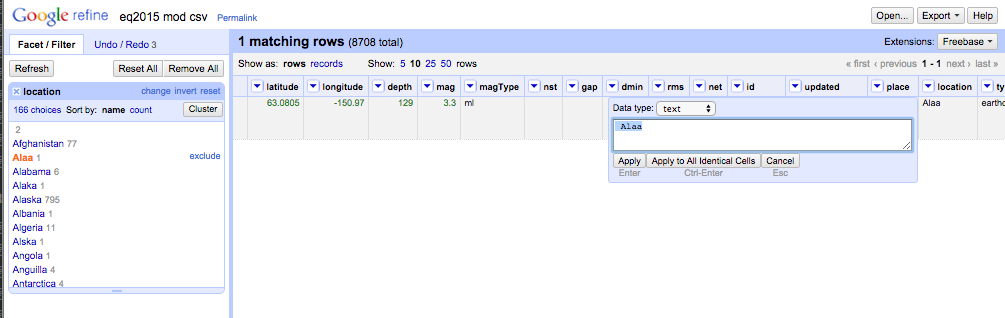
Change block size to 2 and run the clustering.

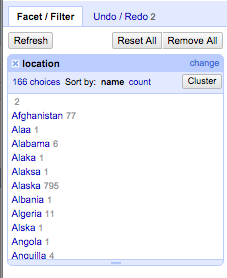
SUBMISSION 6: Change block size to 2. Give 2 examples of new clusters that may be worthwhile merging.

You can try different parameters to see if you can catch the issues you see. If not you can also note that there are a few misspellings of Alaska that only occurs once. Hence, it is doable to go in and edit them by hand.



If you review the facet you may still see values that seems wrong and that were not caught. If these are single values the easiest if probably just to go in an edit those cells and fix the values. You can access the values by clicking on the facet widget.





Consider if you had a very large data set and you wanted to automate the cleaning. Expecting manual editing would not be feasible.

The place column strings are significantly longer that the strings for location. Try to do nearest neighbor clustering on “place’ column. What happens and why? How does the user experience compare with the clustering of location?

SUBMISSION 7: Explain in words what happends when you cluster place, and why you think that happened? What additional functionality could OpenRefine provide to possibly deal with the situation.

Hint…it takes a long time; you may in fact want to cancel the run.

# Step-3 Levenshtein Distance

## Introduction

In this part of the lab we will go over a simple example of Levenshtein distance calculation. We will then ask you to calculate the distance for two strings gumbarrel" and "gunbarell". We will point you to a python implementation of the Levenshtein distance that you can use to check your result.

## Installing Levenshtein python module

The following steps will just clone and build a Python Levenshtein module in a directory. It does not fully install the module. But you can use it to run a distance function from your shell to check your results by running the python shell in the Levenshtein sub-directory.

$ git clone <https://github.com/ztane/python-Levenshtein/>

$ cd python-Levenshtein/

$ python setup.py build

$ cd Levenshtein/

$ python

>>> from Levenshtein import \*

>>> distance("hej","hei")

1

>>> distance("monthgomery st","montgomery street")

5

## Example: Levenshtein Calculation

Lets step through the calculation of distance between the words LOYOLA and LAJOLLA. We will denote a cell with the d[i,j], where i is the row and j is the column. The dark column and row indicates the indice number we will be using for the actual calculation matrix.

As a reminder the algorithms is as follows:

Denote the row by r and column by c. We have n rows and m columns.

d[i,j] denotes the value on row i and columns j.

d[i,j] is to be set to the minimum of: d[i-1,j]+1 or d[i,j-1]+1 or d[i-1, j-1]+cost[i,j]

Distance is found in the resulting value d[n,m]

We first set up the matrix. The dark row and column just contains the i and j values. We then insert values 0-m in first row i==1 and 0-n in the column j==1.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|  |  |  | L | O | Y | O | L | A |
| 1 |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 2 | L | 1 |  |  |  |  |  |  |
| 3 | A | 2 |  |  |  |  |  |  |
| 4 | J | 3 |  |  |  |  |  |  |
| 5 | O | 4 |  |  |  |  |  |  |
| 6 | L | 5 |  |  |  |  |  |  |
| 7 | L | 6 |  |  |  |  |  |  |
| 8 | A | 7 |  |  |  |  |  |  |

Lets calculate the d[i,2]. Meaning the value for each row in the column 2.

d[2,2], cost is 0, minimum is d[1,1]+0=>0

d[3,2], cost is 1, minimum is d[2,2]+1=>1

d[4,2], cost is 1, minimum is d[3,2]+1=>2

d[5,2], cost is 1, minimum is d[4,2]+1=>3

d[6,2], cost is 0, minimum is d[5,1]+0=>4, or d[5,2]+1

d[7,2], cost is 0, minimum is d[6,2]+0=>5, or d[6,2]+1

d[8,2], cost is 1, minimum is d[7,2]+1=>6

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|  |  |  | L | O | Y | O | L | A |
| 1 |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 2 | L | 1 | 0 |  |  |  |  |  |
| 3 | A | 2 | 1 |  |  |  |  |  |
| 4 | J | 3 | 2 |  |  |  |  |  |
| 5 | O | 4 | 3 |  |  |  |  |  |
| 6 | L | 5 | 4 |  |  |  |  |  |
| 7 | L | 6 | 5 |  |  |  |  |  |
| 8 | A | 7 | 6 |  |  |  |  |  |

Lets calculate the d[i,3]. Meaning the value for each row in the column 3.

d[2,3], cost is 1, minimum is d[2,2]+1=>1

d[3,3], cost is 1, minimum is d[2,2]+1=>1

d[4,3], cost is 1, minimum is d[3,2]+1=>2, or d[3,3]+1

d[5,3], cost is 0, minimum is d[4,2]+0=>2

d[6,3], cost is 1, minimum is d[5,3]+1=>3

d[7,3], cost is 1, minimum is d[6,2]+1=>4, or d[6,3]+1

d[8,3], cost is 1, minimum is d[7,2]+1=>4

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|  |  |  | L | O | Y | O | L | A |
| 1 |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 2 | L | 1 | 0 | 1 |  |  |  |  |
| 3 | A | 2 | 1 | 1 |  |  |  |  |
| 4 | J | 3 | 2 | 2 |  |  |  |  |
| 5 | O | 4 | 3 | 2 |  |  |  |  |
| 6 | L | 5 | 3 | 3 |  |  |  |  |
| 7 | L | 6 | 3 | 4 |  |  |  |  |
| 8 | A | 7 | 4 | 4 |  |  |  |  |

If you do the same thing for the remaining columns we will get the following matrix. You see the calculated edit distance in the cell d[8,7].

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|  |  |  | L | O | Y | O | L | A |
| 1 |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 2 | L | 1 | 0 | 1 | 2 | 3 | 4 | 5 |
| 3 | A | 2 | 1 | 1 | 2 | 3 | 4 | 4 |
| 4 | J | 3 | 2 | 2 | 2 | 3 | 4 | 5 |
| 5 | O | 4 | 3 | 2 | 3 | 2 | 3 | 4 |
| 6 | L | 5 | 3 | 3 | 3 | 3 | 2 | 3 |
| 7 | L | 6 | 3 | 4 | 4 | 4 | 3 | 3 |
| 8 | A | 7 | 4 | 4 | 5 | 5 | 4 | **3** |

If you use the Levenshtein function to check the result you will see the following result.

>>> distance("loyola","lajolla")

3

So we are assuming we got the calculation right.

## Calculation: gumbarrel v.s gunbarell

Now calculate the edit distance between the words: “gumbarrel” and “gunbarell". After you done that use the python Levenshtein function to check you result.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  |  |  | G | U | M | B | A | R | R | E | L |
| 1 |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 2 | G | 1 |  |  |  |  |  |  |  |  |  |
| 3 | U | 2 |  |  |  |  |  |  |  |  |  |
| 4 | N | 3 |  |  |  |  |  |  |  |  |  |
| 5 | B | 4 |  |  |  |  |  |  |  |  |  |
| 6 | A | 5 |  |  |  |  |  |  |  |  |  |
| 7 | R | 6 |  |  |  |  |  |  |  |  |  |
| 8 | E | 7 |  |  |  |  |  |  |  |  |  |
| 9 | L | 8 |  |  |  |  |  |  |  |  |  |
| 10 | L | 9 |  |  |  |  |  |  |  |  |  |

SUBMISSION 8: submit a representation of the resulting matrix from the leveshtein edit distance calculation. The resulting value should be correct.