OpenRefine Tutorial

From Enipedia
(Redirected from Google Refine Tutorial)

Contents

Overview

The shortened URL for this page is http://is.gd/refine

This is a tutorial on using OpenRefine (formerly Google Refine), and has been developed to teach students in a statistics class how this tool can be used to clean up data.

The data used is here (http://enipedia.tudelft.nl/enipedia/images/f/ff/UniversityData.zip) , which is a zip file containing only the file universityData.csv, which is a plain text CSV file. Save the file directly to your computer. Don't open it first in Excel, since saving it again may disturb the layout.

The example used shows how we can use Wikipedia data to see if there is a relationship between the number of students at a university and the size of the university's endowment.

- **Main Page** - https://github.com/OpenRefine/OpenRefine/wiki - This contains several videos that give an overview of the tool.
- **Download** - https://code.google.com/p/google-refine/downloads/list - Available for Windows, Mac and Linux
- **Documentation** - https://github.com/OpenRefine/OpenRefine/wiki/Documentation-For-Users - Almost everything you need to know. Search Google and you'll find everything else.
- **Advanced Documentation** - Google Refine Expression Language (GREL) reference (https://github.com/OpenRefine/OpenRefine/wiki/Documentation-For-Users#reference) - This shows how you can use a programming language to perform some very sophisticated tasks.

Download OpenRefine from the download page. The google-refine window should be kept open, while working in your browser with google-refine.

The data is sourced from a SPARQL query to DBpedia which extracts information about universities from Wikipedia infoboxes.

What you can learn

The data contains quite a few issues, and this tutorial shows how to do things like:

- Cleaning up inconsistent spelling of terms (i.e. "USA", "U.S.A", "U.S.", etc).
- Converting values that are text descriptions of numeric values (i.e. $123 million) to actual numeric values (i.e. 123000000) which are usable for analysis.
- Identifying which rows of a specific column contain a search term
- Extracting and cleaning values for dates
- Removing duplicate rows
- Using a scatterplot to visualize relationships between values in different columns
- Finding geographic coordinates for a list of place names (i.e. the names of universities, etc.)
- Exporting cleaned data to Excel

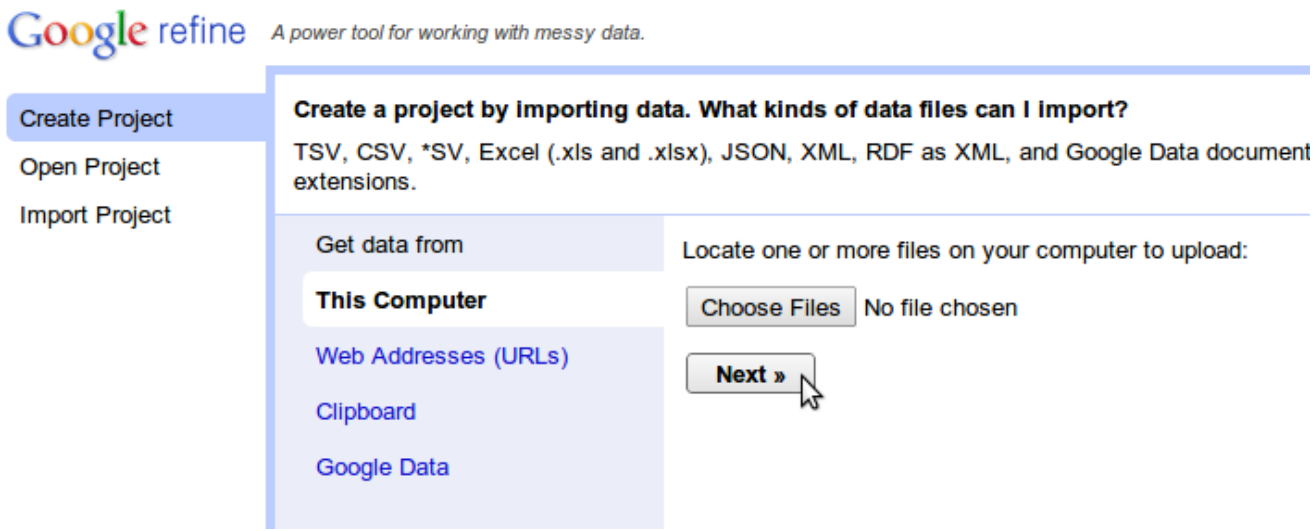
Steps

Installation

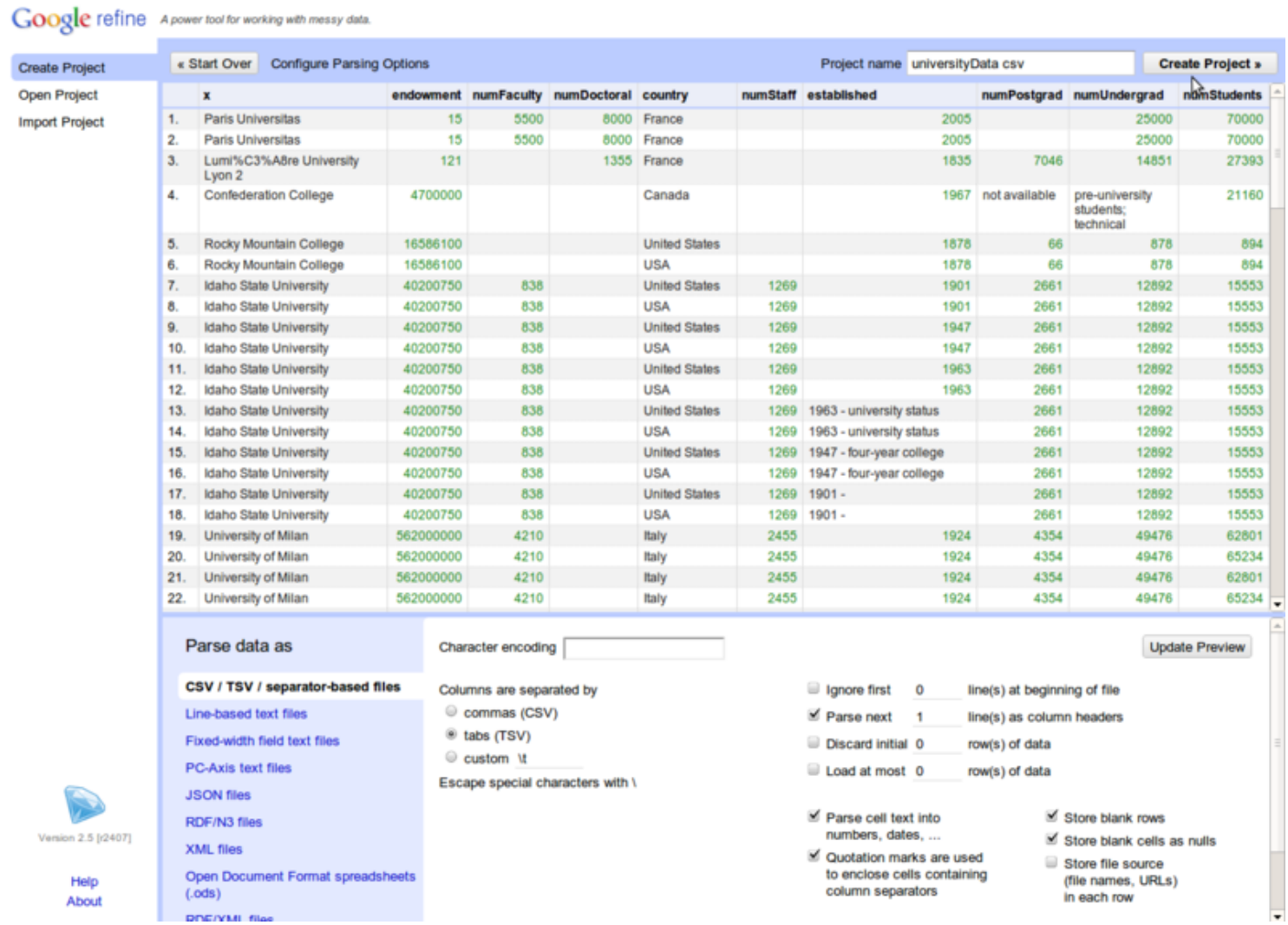
Once you download OpenRefine, you need to unzip it. After this, follow the instructions on the download page (http://code.google.com/p/google-refine/wiki/Downloads) to run it. OpenRefine runs in a web browser, and when you start it, it should automatically open up a web browser window. If this does not happen, open a web browser yourself and go to http://127.0.0.1:3333, and you should see it.

Load file and create project

Click on "Create Project", then "Choose Files". Select the file from your computer (universityData.csv), then click on "Next"

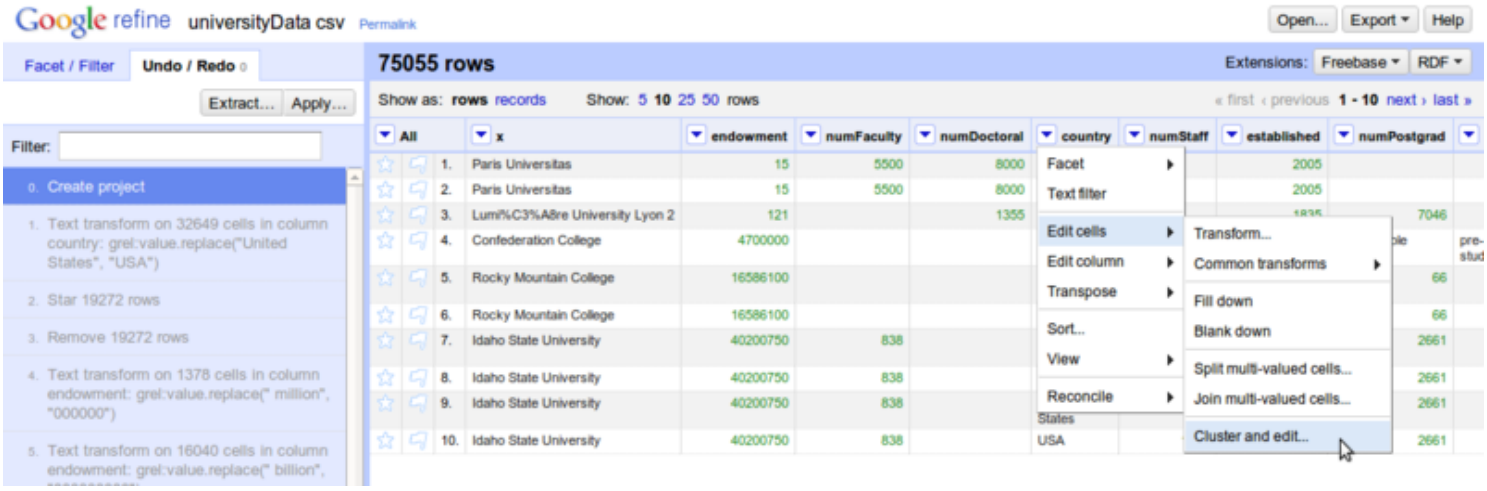


The data should be read in OK, so you can go ahead and click on Create Project. Note: Remember to check that the box, "Parse cell text into numbers, dates, etc", is ticked!

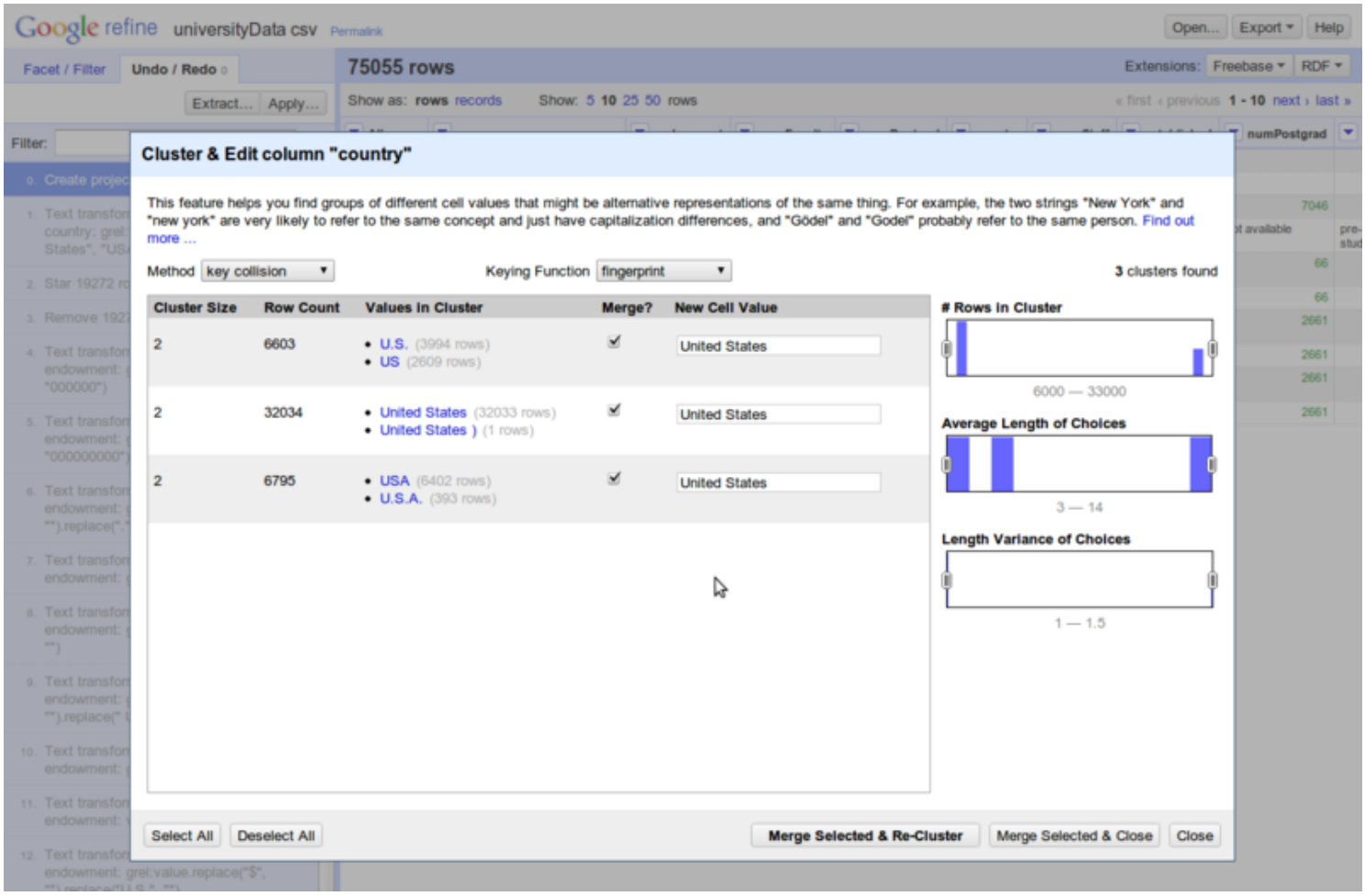


Clean up country names

The data contains variants of the names for several countries. To fix this, use **Edit cells->Cluster and edit** on the country column.



We already see an issue here where there is both the full name of a country (United States) and its abbreviation (US). To fix this, we can just copy/paste "United States" as the new cell value.

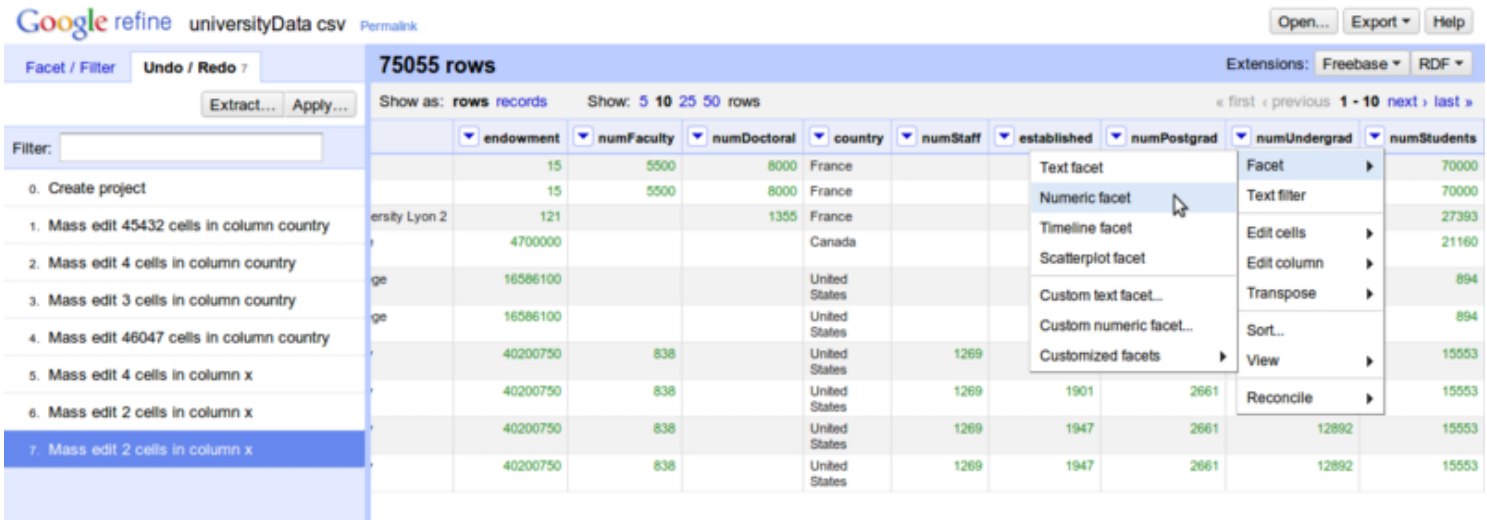


It's not just the US that has different names specified for it. By using the different string comparison algorithms (choose a different method and/or keying function), you can find issues with other countries as well.

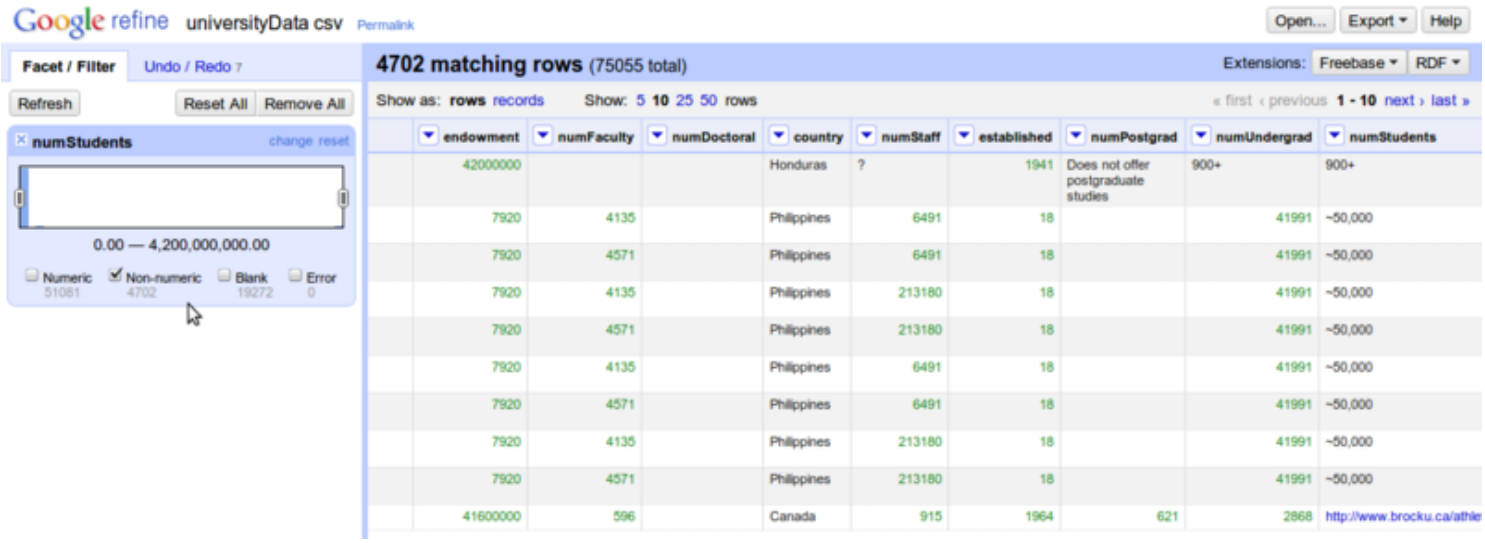
Clean up values for the number of students

We need to clean the data for the number of students. Not all of the values are numeric, and many of them contain bits of text in addition to the actual number of the students.

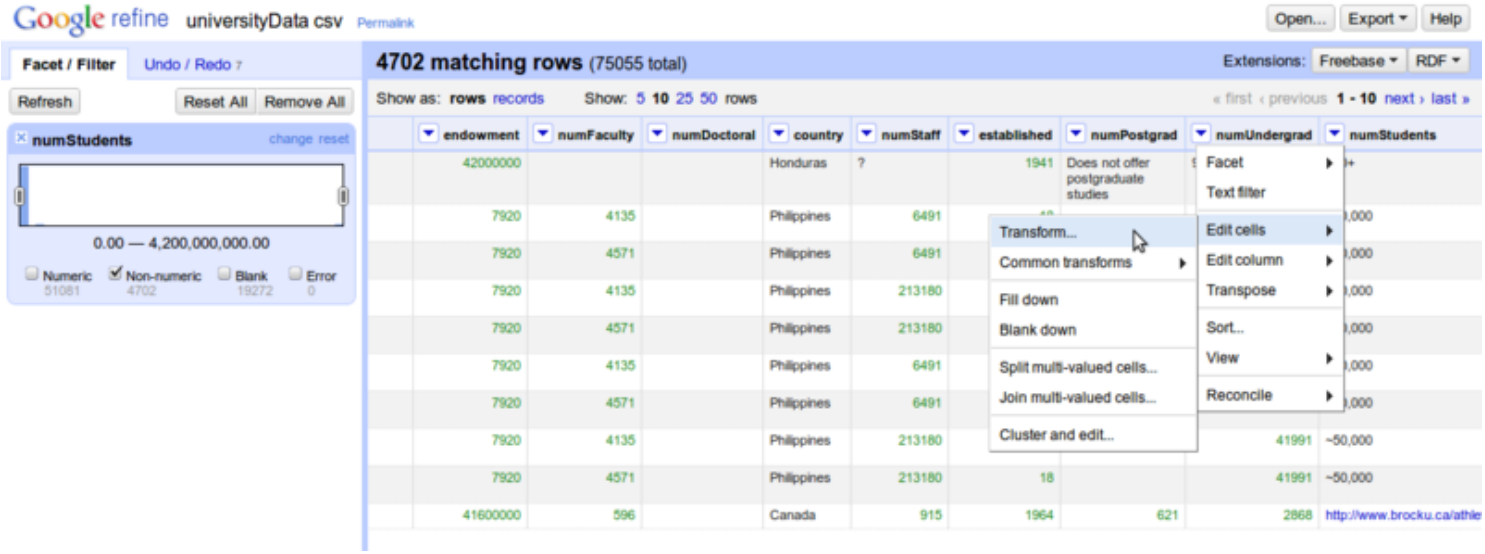
To figure out which entries need to be fixed, we need to use a Numeric facet:



This shows us a histogram of the values, and also lists the number of entries per type (numeric, non-numeric, blank, error, etc). Make sure that only the non-numeric rows are selected:



We can see some problems already, as some cells have "+" and "~" in them. To fix this, we need to do **Edit cells -> Transform**

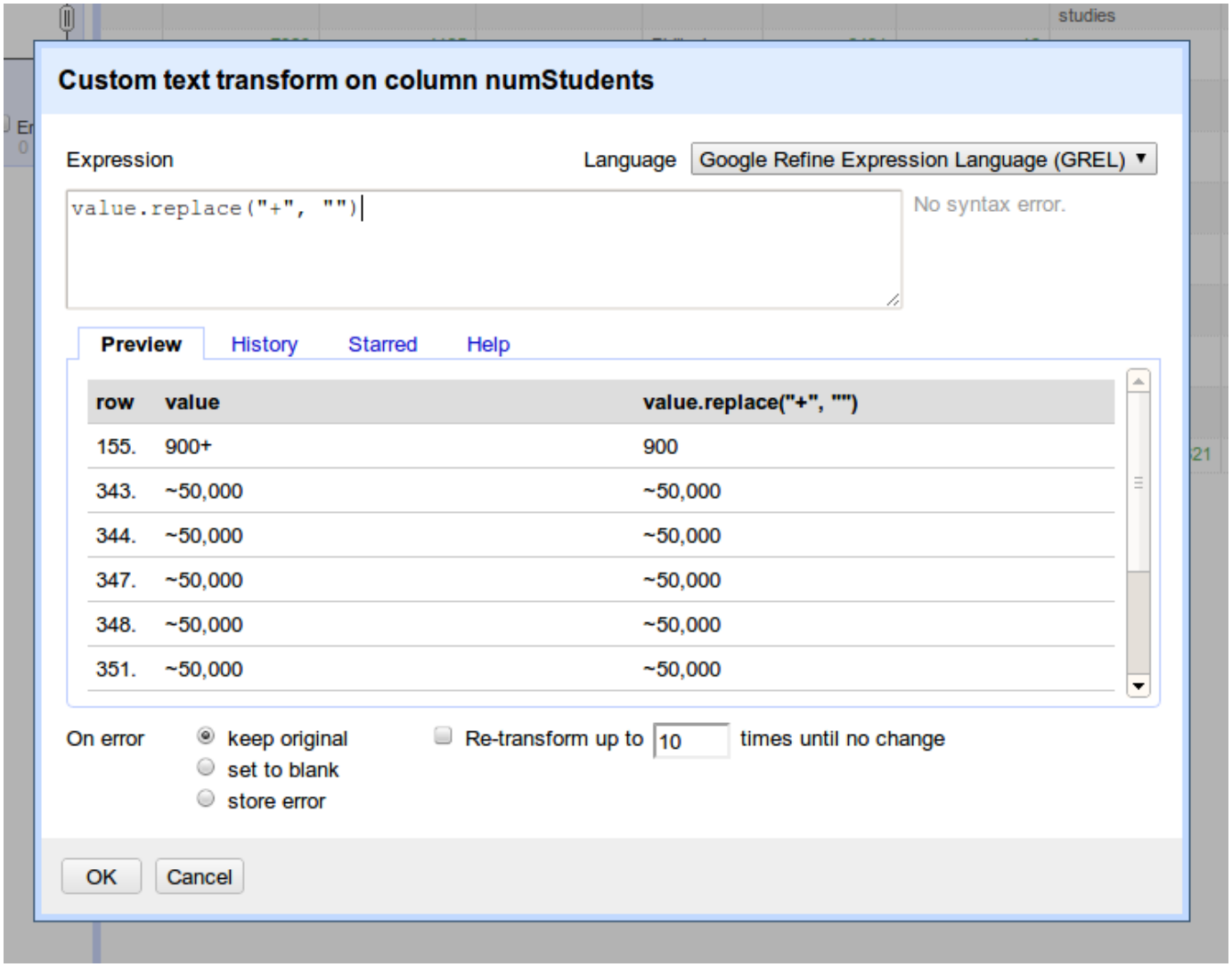


This allows us to now type in commands that can replace sequences of characters:

```
value.replace("+", "")
```

Also, if you see entries with strange symbols like "Lumi%C3%A8re University Lyon 2" in the "x" column (should be "Lumière University Lyon 2"), you can fix this via **Edit cells -> Transform** with this command:

```
value.unescape('url')
```

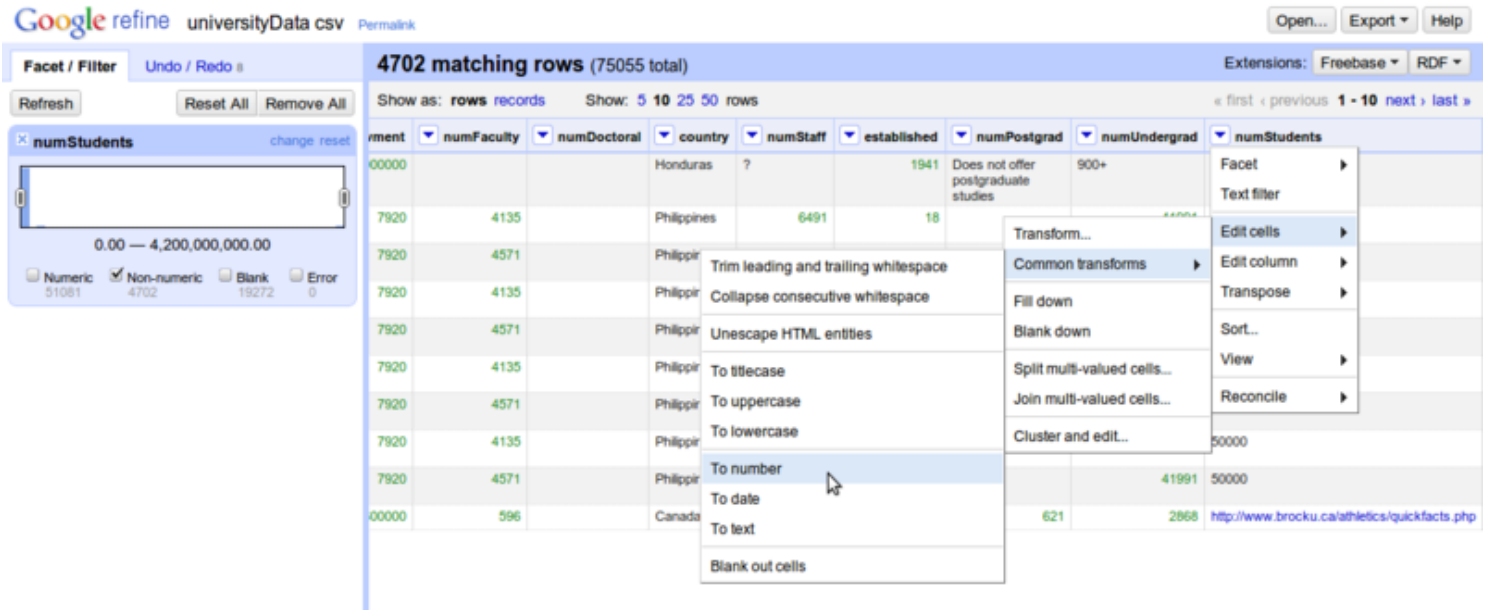


In doing this, you're actually using bits of a programming language. A lot of advanced features are available (not covered in this tutorial), and if you want to understand this further, you can refer to the Google Refine Expression Language (GREL) reference (http://code.google.com/p/google-refine/wiki/DocumentationForUsers#Reference)

If you find multiple things that need to be replaced, you don't have to keep clicking **Edit cells -> Transform** for every single issue. You can chain these commands together to fix several issues at once:

```
value.replace("~", "").replace(",","")
```

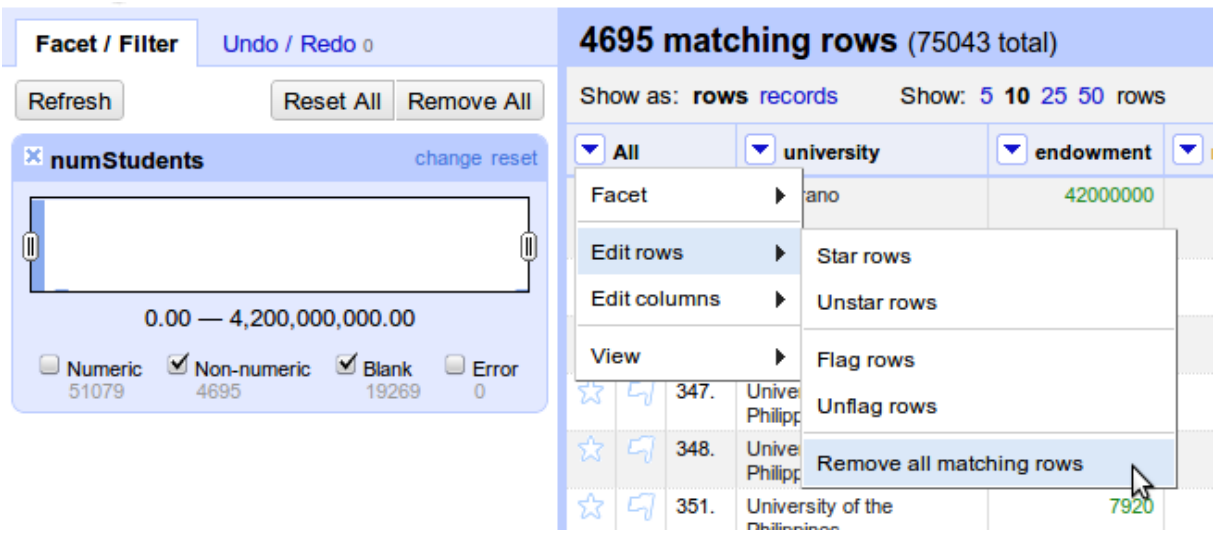
In order to update the selection of non-numeric values, it's sometimes necessary to convert the values of the columns to numbers - **Edit cells -> Common transforms -> To number**. Once you do this, you should see that there are fewer non-numeric values.



More issues can be cleaned up via:

```
value.replace(" total", "").replace(" -", "")
```

You can continue cleaning up the data, but for this exercise we will move on and remove all the rows that do not have numeric values for the number of students. To do this, use a numeric facet again on numStudents to select only the non-numeric and blank values. Then do **All -> Edit rows -> Remove all matching rows**

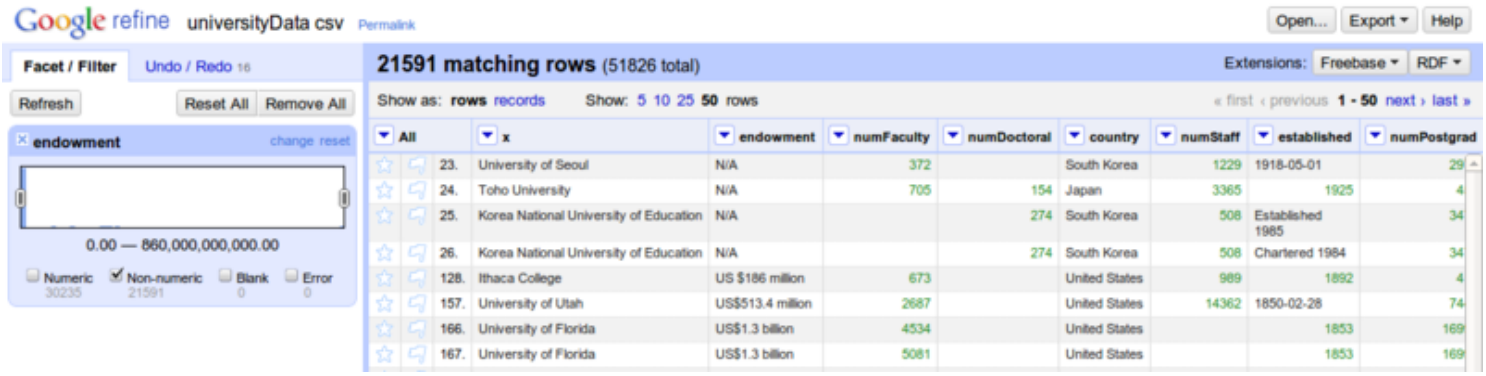


Clean up values for the endowment

It's possible to have multiple facets in use at once. When you do this, each additional facet makes a sub-selection of the data selected by the previous facet. If you find that the number of rows you have selected and are working with is smaller than expected, then check to see if you still have facets in use which are not needed any more.

First remove the numeric facet for numStudents and create a new numeric facet for endowment. Select only the non-numeric values, as was done for the number of students.

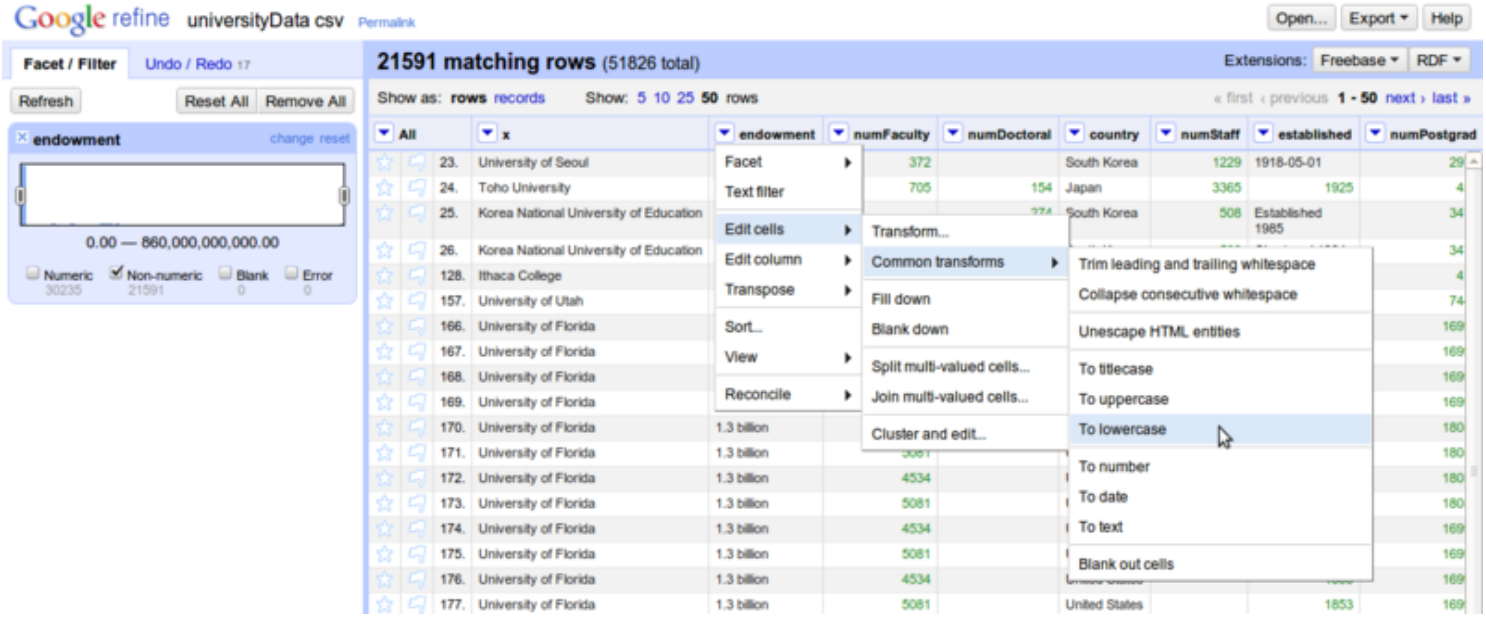
Already we see issues like "US$1.3 billion" and "US $186 million"



Assuming that everything is in $ (a somewhat bogus assumption), we can clean up the data similarly to how we did it before. Click on the endowment column -> **Edit cells -> Transform**

```
value.replace("US $","").replace("US$", "")
```

Both "million" and "Million" are in the values, so it's useful to convert all the values to lowercase instead of cleaning this up twice.



Click on the endowment column again, and create a custom text facet to locate all the rows with the word "million" in them: **Facet -> Custom text facet**

```
value.contains("million")
```

Then **Edit cells -> Transform**. It's not advisable to just replace "million" by "000000" since you have some values like "$13.8 million", which would be converted to "$13.8000000". It's better to first remove "million" from the text, convert the remaining text to a number, and then multiply this by 1000000:

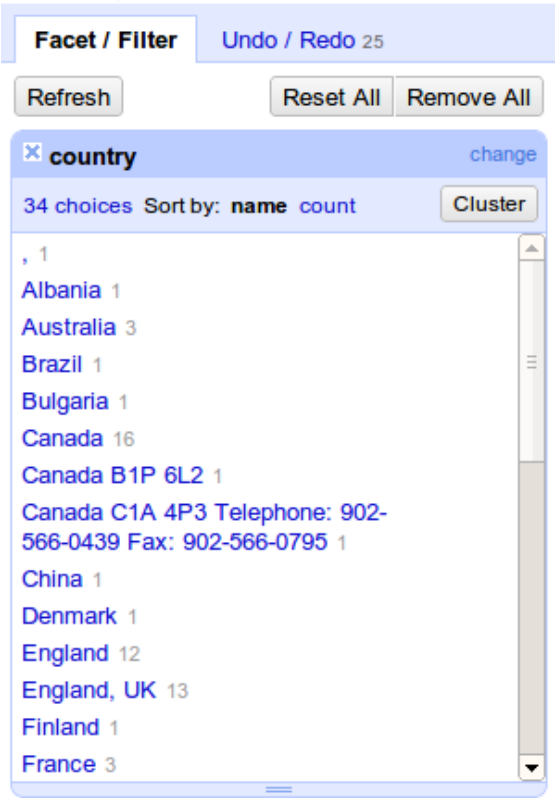
```
toNumber(value.replace(" million", ""))*1000000
```

The term "billion" is in the values as well, so remove previous facet for endowment, and create a new one for billion, and repeat process described above.

After most of this has been cleaned up, select the non-numeric values, and delete them, just as was done for the numStudents.

Finding issues in other columns

OpenRefine has plenty of features that can help clean up the other columns as well. For example, if you do a text facet on the column with country names, you will find issues such as entries for both "England" and "England, UK", along with entries for Canada that contains parts of the university address.



Cleaning up dates

The dates are a mess as well, but there's a few techniques that can be used to help clean them up.

First we want to convert everything to text - **Edit cells -> Common transformations -> To text**, and then you need to **Edit cells -> Common transformations -> To date**. If you did not convert all the values to text first, then you may find that some of the years are represented as numbers, and have not been converted.

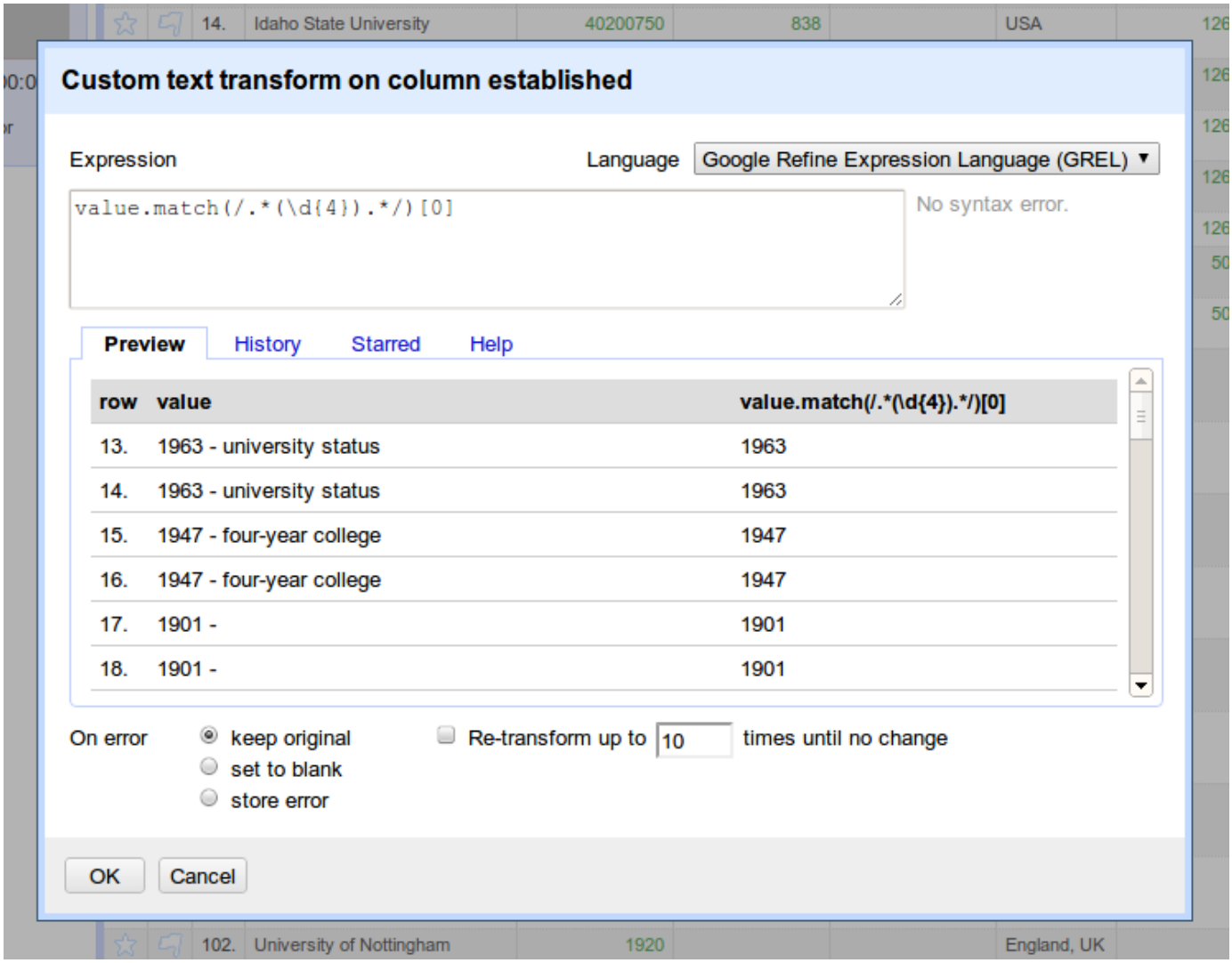
If only a year is listed, then the date created will use January 1st as the month and day. We will clean this up later to use only the year.

To further clean up the dates, we need to use **Facet -> Timeline facet** and select only "Non-Time" values. This shows us that we have a bit of a tricky situation as years are mixed in with text such as "Established 1985". We need some way to recognize a sequence of four numbers in a section of text and extract only the numbers. To do this, we need to use regular expressions (http://code.google.com/p/google-refine/wiki/UnderstandingRegularExpressions) . This is a very powerful technique that allows you to specify very complex patterns that you wish to match. For this tutorial, you don't need to know how to write regular expressions, but you should at least know that they exist, and that they can be used to help you with seemingly impossible tasks.

Regular Expressions. Source:XKCD (http://xkcd.com/208/) , CC BY-NC 2.5 (http://creativecommons.org/licenses/by-nc/2.5/) license.

We now want to do **Edit cells -> Transform**, and use the code below. The ".*" means a sequence of zero or more characters (letters, numbers, symbols, etc). The "\d" indicates that we're looking for a digit. The "{4}" shows that we want to match exactly 4 digits. The value.match function returns an array of results, so we use "[0]" to retrieve only the first match.

```
value.match(/.*(\d{4}).*/)[0]
```



We can now convert these extracted values to dates - **Edit cells -> Common transformations -> To date**. At this point, we've done almost everything we can to track down usable dates, and we now want to just extract the years. To do this, we want to **Edit cells -> Transform** with the code below:

```
value.toString('yyyy')
```

What's happening here is that we're using a string ('yyyy' in this case) to specify what parts of the date we want, and how it should be displayed. The documentation here (http://docs.oracle.com/javase/1.4.2/docs/api/java/text/SimpleDateFormat.html) describes this in much more detail. As illustrated in the table below, you can experiment with different commands to get different formats of dates.

Command	Result
value.toString('M')	1
value.toString('MM')	01
value.toString('MMM')	Jan
value.toString('MMMM')	January

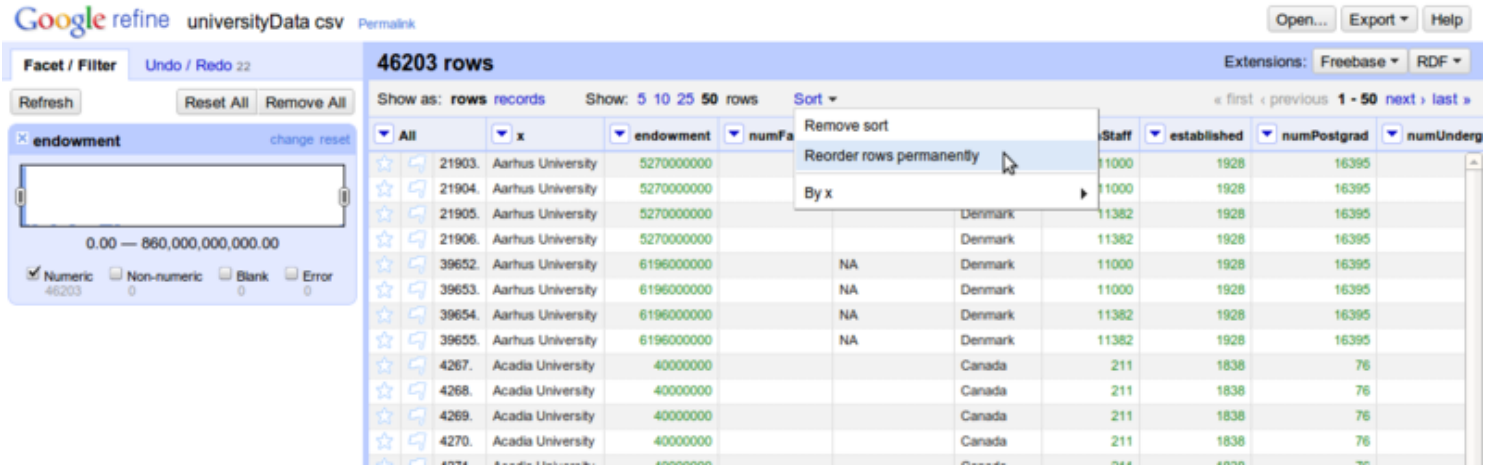
As described here (http://code.google.com/p/google-refine/wiki/GRELDateFunctions) , you can use code such as that below to reformat multiple date formats into a single format.

```
value.toDate('MM/yy','MMM-yy').toString('yyyy-MM')
```

Deduplicate entries

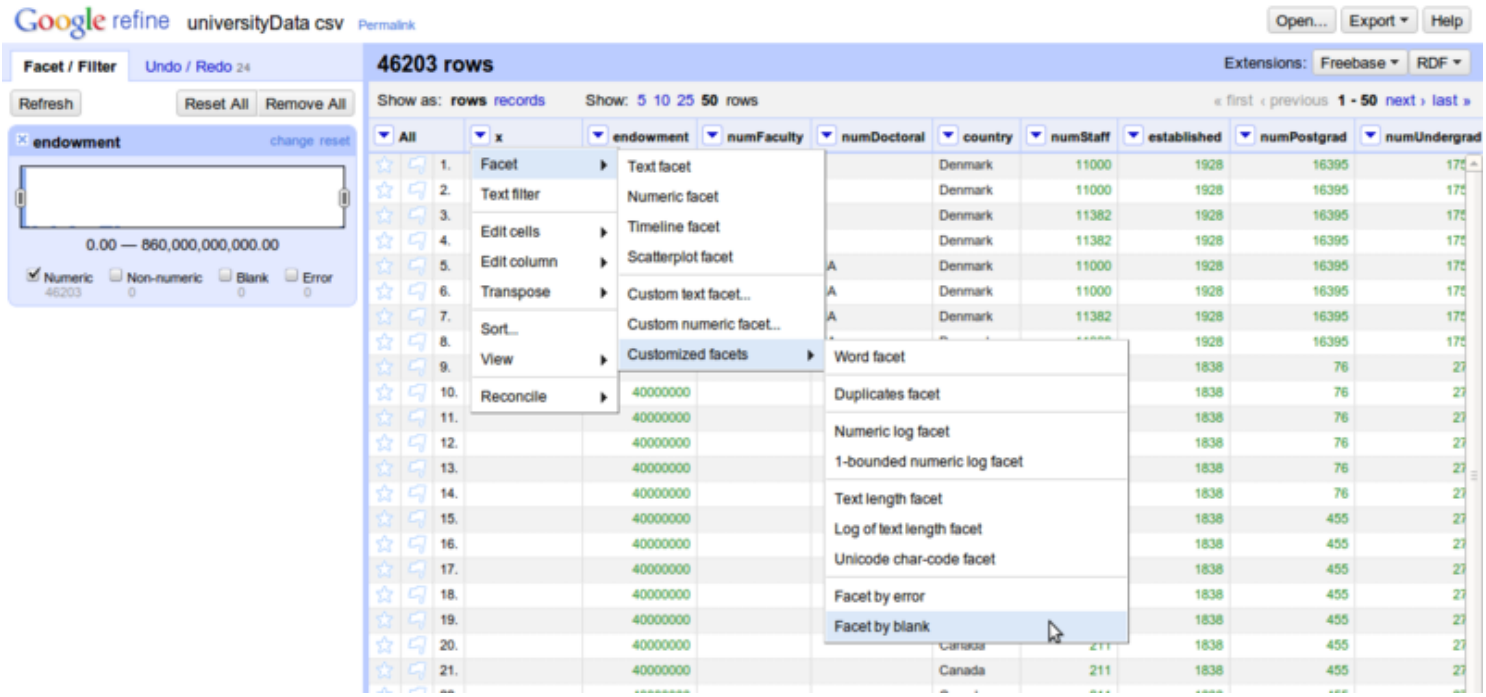
There's a lot of (nearly) duplicate rows in the data. Why this happens is a bit of a long story, and is due to Wikipedia having things like multiple numbers of students listed for different years. When the data is retrieved, permutations of these values are returned. To make things simple, we want to just keep the first row of data for each university.

To do this (based on documentation here (http://googlerefine.blogspot.nl/2011/08/remove-duplicate.html)), click on the column with the university names, and then click on "Sort". Once you do this, you will notice that there is a new "Sort" menu at the top. Click on this and select "Reorder rows permanently". This may take a while as it renumbers the rows in which the entries appear.



Then on the column with university names, **Edit cells -> Blank down**

Then on the same column, **Facet -> Customized facets -> Facet by blank**

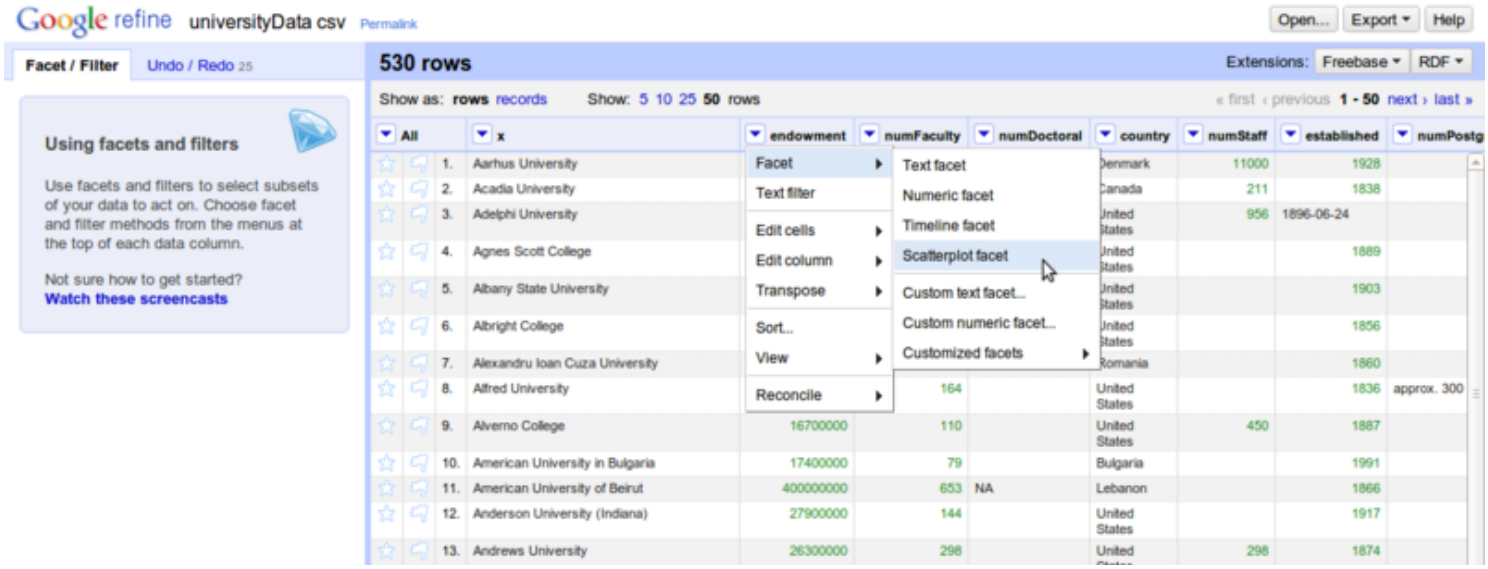


Now we want to remove all the blank rows, so select true, then on the "All" column on the left, Edit rows -> Remove all matching rows, like you have done when working with the numStudents and endowment columns.

Once you remove all the facets, and you now have a (mostly) cleaned data set.

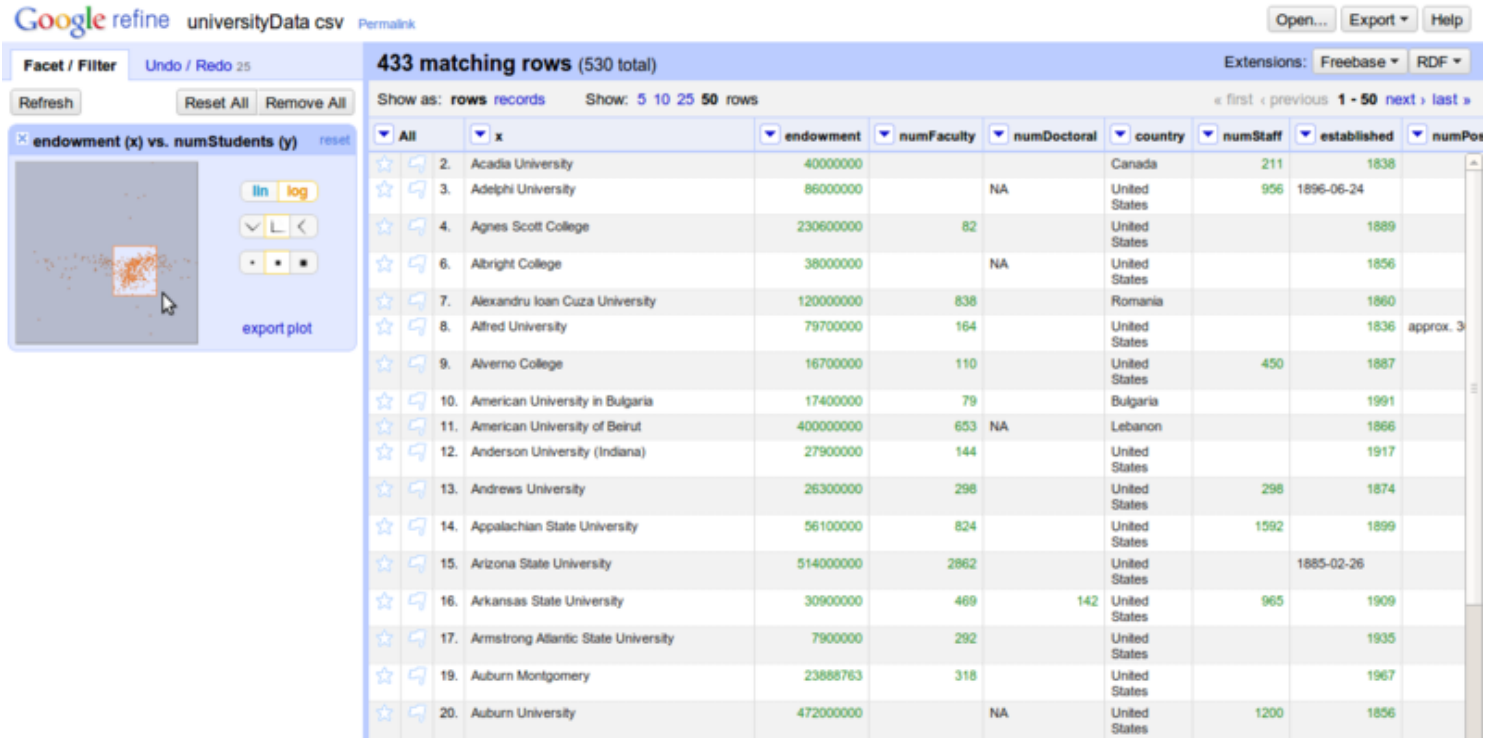
Exploring the data with scatter plots

Click on the "endowment" column, **Facet -> Scatterplot facet**.



This shows the relationships between all of the numeric values in each of the columns. Click on "log" to get a better view.

Click on the plot for endowment vs. numStudents. You can now drag select a portion of the plot, and then see the rows corresponding to that selection.

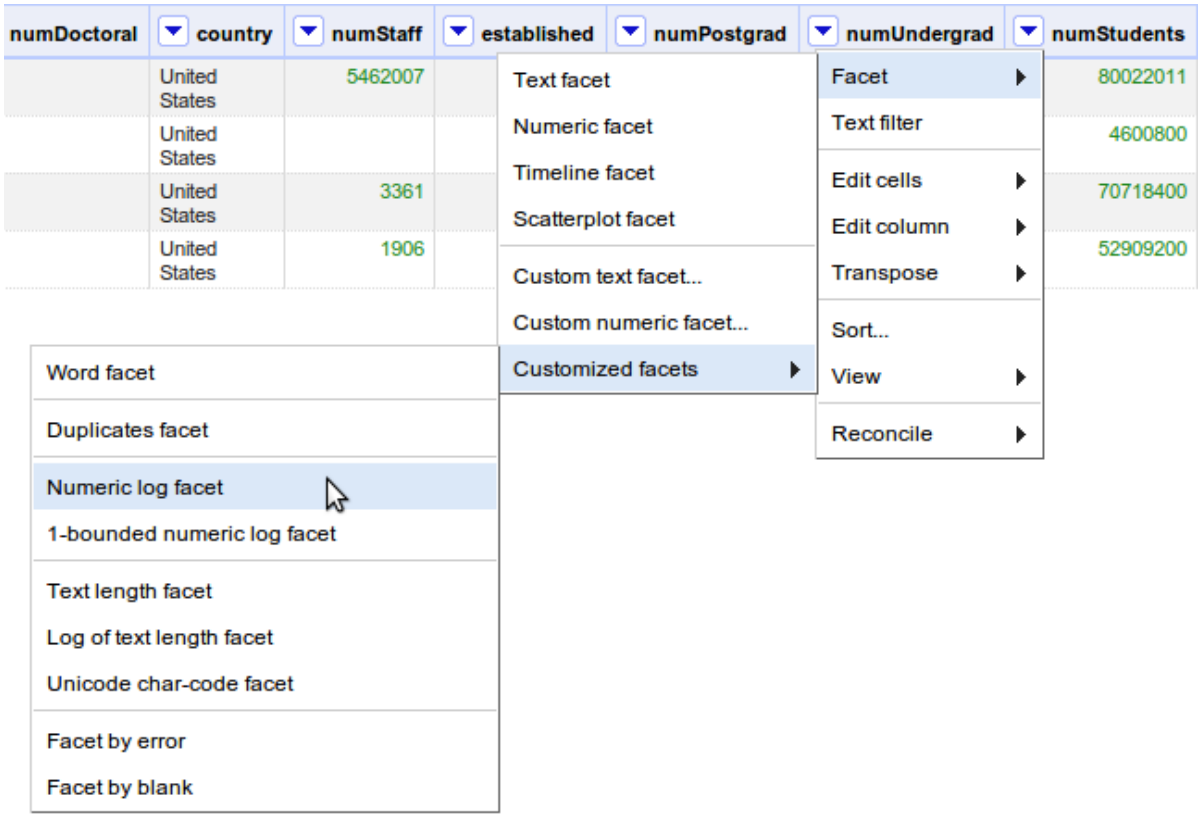


Geocoding names and addresses

This next part shows (based on documentation here (http://code.google.com/p/google-refine/source/browse/wiki/Geocoding.wiki?r=1342)) how to go from a description of a place (i.e. the name of a university) to values for its (likely) geographic coordinates. Behind the scenes, this uses Google Maps to figure out what is the most likely location you are asking for.

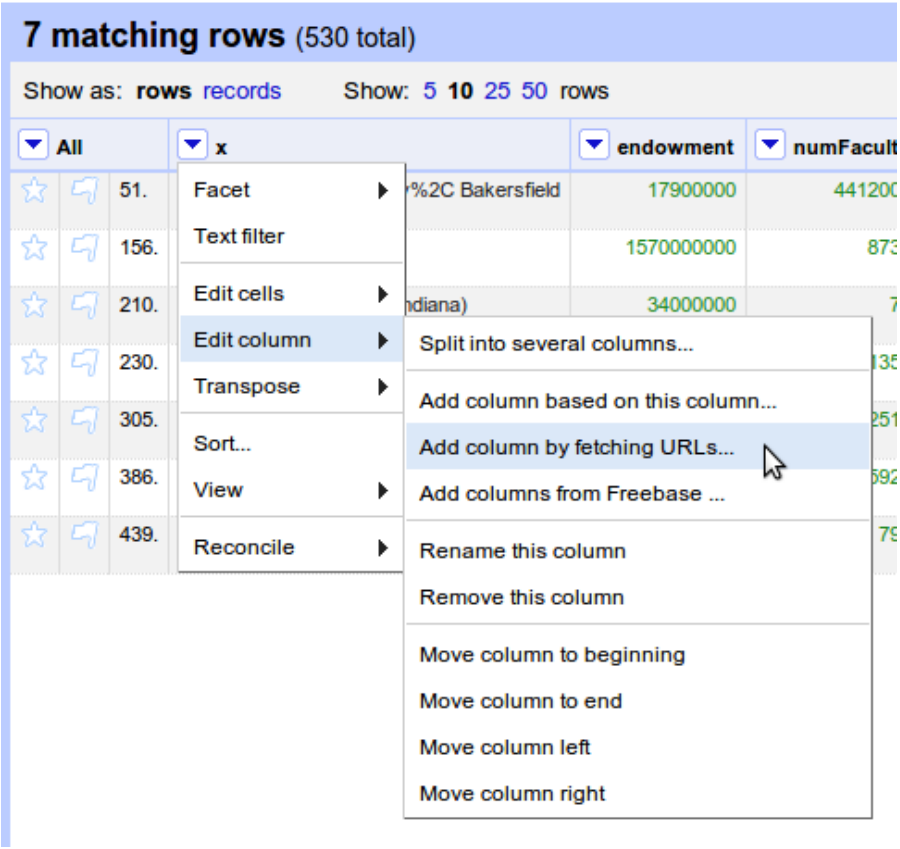
To learn how to do this, you don't need to do process the whole data set. This can take a while, and Google limits you to 2000 requests per day. It's better to just select around 10 rows and verify that it works.

An easy way to get a limited set of rows is by using a numeric log facet of the number of students, so use **Facet -> Customized facets -> Numeric log facet**



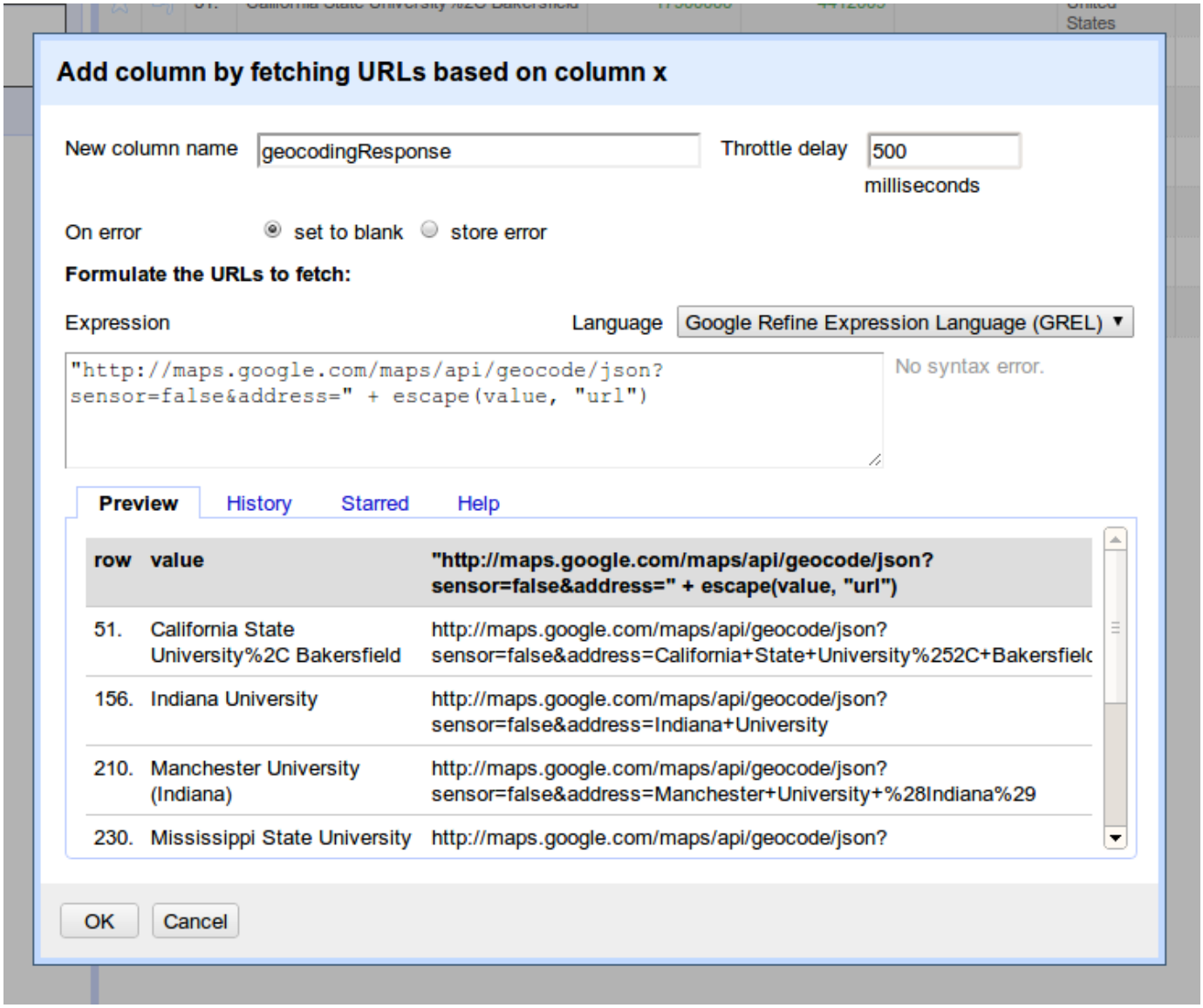
Use this facet to make a selection of around ten rows, and then check the **matching rows** number to verify that you have a reasonable selection size:

Now the fun begins and we want to do **Edit column -> Add column by fetching URLs**. In other words, the values of the cells in the new column are based on data that is retrieved from the Internet.



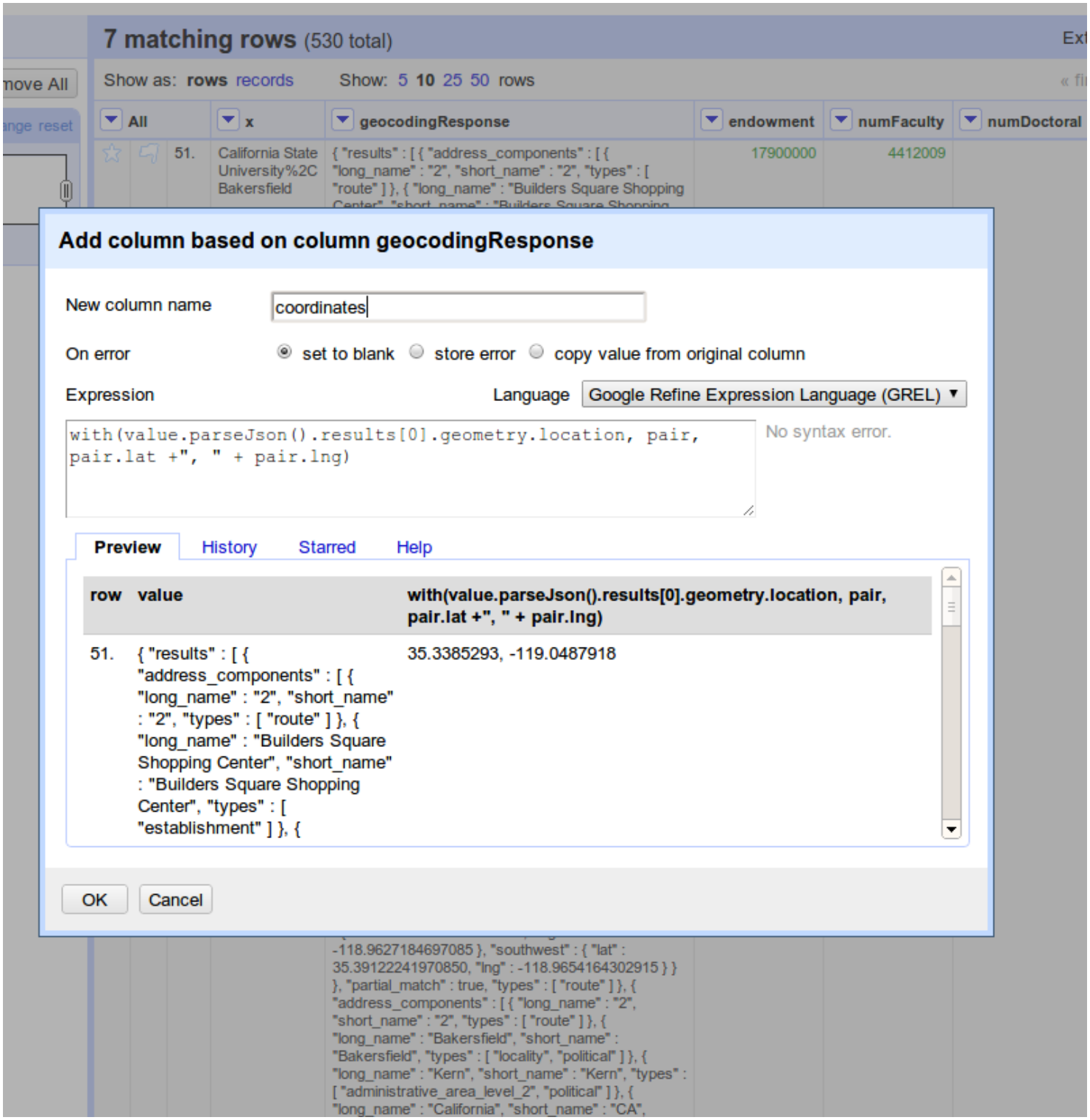
Enter in the expression below, and you should see a list of URLs with the names of the universities at the end of the URLs. Specify a new column name such as "geocodingResponse", and set the throttle delay to around 500 milliseconds.

```
"http://maps.google.com/maps/api/geocode/json?sensor=false&address=" + escape(value, "url")
```



You should get a bunch of data back. To convert this into a more readable format, you need to click on the geocodingResponse column, and then on **Edit column -> Add column based on this column**. Enter in the expression below

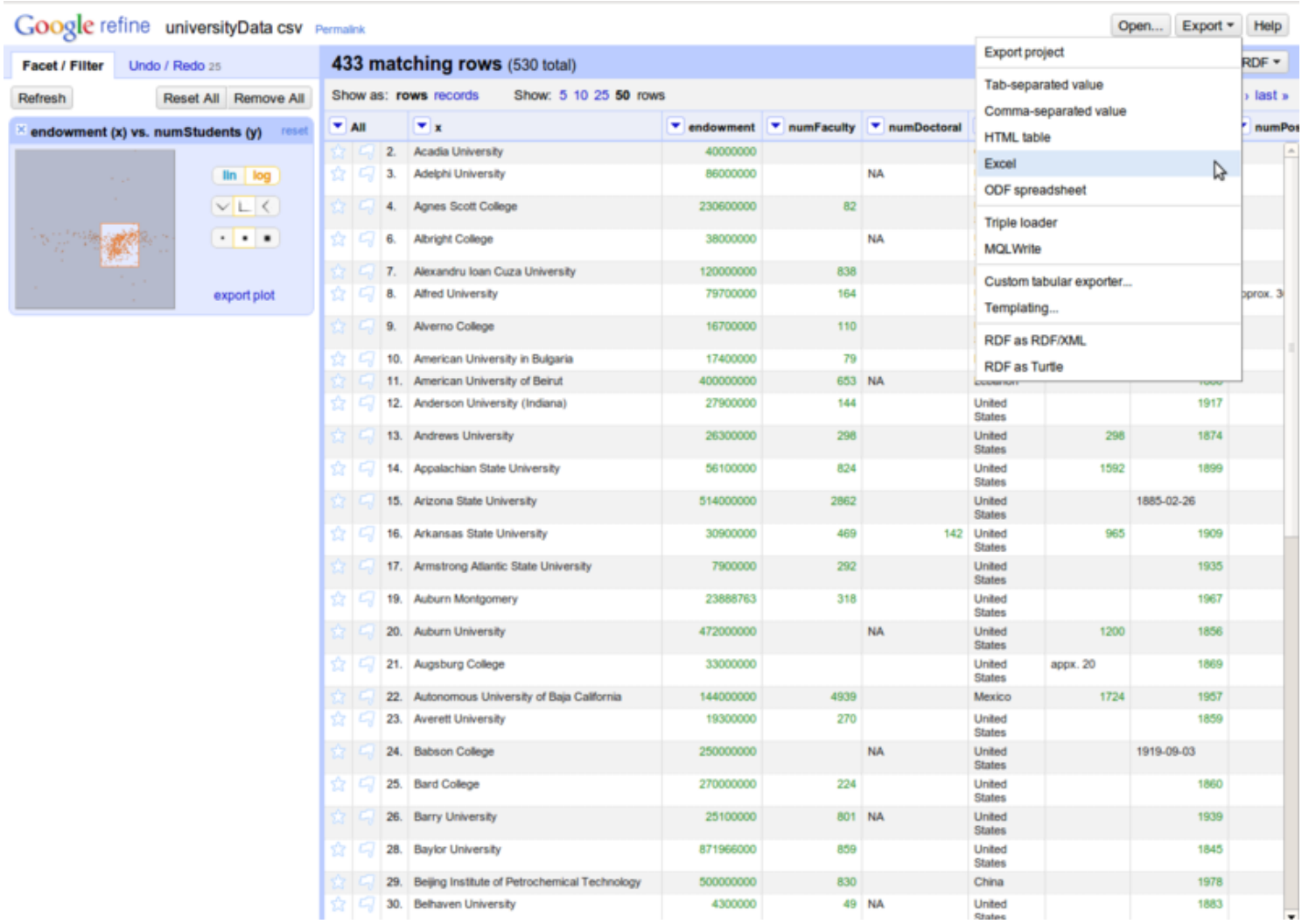
```
with(value.parseJson().results[0].geometry.location, pair, pair.lat +", " + pair.lng)
```



Now you have a single column with coordinates. You can split this into columns for latitude and longitude by selecting **Edit Column -> Split into several columns** and specifying a separator of ",". These columns can then be renamed using **Edit Column -> Rename this column**.

Export Data

The data can be exported to formats such as Excel. If you read this into tools such as SPSS and notice that the last column is missing, then open the file up in Excel, re-save it, and try to open it up again in SPSS.



Original Data Source

To learn more about how the data was retrieved and how to write your own queries, refer to the tutorials listed on Using SPARQL with Enipedia.

The query used to retrieve the data is shown below and was run at the DBPedia live SPARQL endpoint at http://live.dbpedia.org/sparql. The value for OFFSET is incremented by 10000 with multiple queries, as more than 10000 results are returned.

There's quite a bit of duplication in the results since permutations of the values in different rows are returned. For example, it is common to find that there are multiple values for the numbers of students, which is likely the result of the Wikipedia article mentioning different numbers of students for different years.

```
PREFIX dbpprop: <http://dbpedia.org/property/>
select * where {
       {?x dbpprop:wikiPageUsesTemplate <http://dbpedia.org/resource/Template:Infobox_University> } UNION
       {?x dbpprop:wikiPageUsesTemplate <http://dbpedia.org/resource/Template:Infobox_university> }.
       ?x dbpprop:endowment ?endowment .
       OPTIONAL{?x dbpprop:faculty ?numFaculty }.
       OPTIONAL{?x dbpprop:doctoral ?numDoctoral }.
       ?x dbpprop:country ?country .
       OPTIONAL{?x dbpprop:staff ?numStaff }.
       ?x dbpprop:established ?established .
       OPTIONAL{?x dbpprop:postgrad ?numPostgrad }.
       ?x dbpprop:undergrad ?numUndergrad .
       OPTIONAL{?x dbpprop:students ?numStudents }.
} LIMIT 10000 OFFSET 0
```

More Data Sets - Is the 27 Club Real?

Following the death of Amy Winehouse in 2011, the media declared that she was the latest member of the 27 Club (http://en.wikipedia.org/wiki/27_Club) , which consists of musicians who died at the age of 27. Commonly cited members of this club include Jim Morrison, Jimi Hendrix, Kurt Cobain, and Janis Joplin, which is hardly a representative sample (n=5) given the many thousands of musicians that are out there.

The spreadsheet File:Musicians.xlsx contains data sourced from Wikipedia about artists, their musical genres (one entry per row), and their birth and death dates. Multiple columns exist for birth and death dates as this data is semi-structured on Wikipedia and different techniques are needed to find these values.

The instructions in the tutorial above show step-by-step many types of functions that you will need to use when cleaning up the data. The instructions below show several of the formulas that you may find useful.

Create a new column named birthdate that uses the first value it encounters when scanning from birthdate1 to birthdate2 to birthdate3

```
forNonBlank(cells.birthdate1.value, v1, v1, forNonBlank(cells.birthdate2.value, v2, v2, forNonBlank(cells.birthdate3.value, v3, v3, null
```

create a new column named deathdate in the same fashion as with the birthdate column

```
forNonBlank(cells.deathdate1.value, v1, v1, forNonBlank(cells.deathdate2.value, v2, v2, forNonBlank(cells.deathdate3.value, v3, v3, null
```

Extract the year value for the birthdate and deathdate columns

```
value.match(/.*(\d{4}).*/)[0]
```

Create a new column showing the approximate age at which they died.

```
cells.deathdate.value - cells.birthdate.value
```

Additional Documentation

David François Huynh

- Google Refine Tutorial