Data cleaning with Open Refine

LAB COURSE NOTES & EXERCISES

Pere-Pau Vázquez | Information Visualization | 2020

Índex

1. Introduction

Open Refine is a free open source tool designed to clean data. It was born as Freebase Gridworks (and later Google Refine) as a tool to performing different tasks known as data wrangling. Essentially, all data sources commonly have some defects, and before working with them, it is necessary to remove them.

As a tool, it has the look and feel of a spreadsheet, but it behaves more like a database, since complex operations such as filtering or clustering can be applied. However, everything happens in a spreadsheet layout, where the data is arranged in rows and columns.

Open Refine can be downloaded form https://openrefine.org/, and the webpage also contains several videos that helps users understand the features. This tutorial does not assume that the readers have watched the videos. However, it might be interesting to give them a try before starting with the exercises.

1.1 TUTORIAL OBJECTIVES

The goal of this tutorial is to give the reader a gentle introduction to Open Refine through the step-by-step description of several useful features. However, the tutorial is not intended to be a deep technical document covering all the functions, only the most common will be explained, and the readers are intended to further explore other ones.

There are many other methods to clean data. You could ever use any popular spreadsheet software. However, proper use of Open Refine may save you a lot of time.

1.2 ORGANIZATION

In this tutorial we start with an overview of the tool, and the detailed description of several available procedures. In order to illustrate those, concrete data files will be used, mostly from the Barcelona Open Data site. Finally, we will propose some small exercises that revisit some of the features, and a final task where you will have to clean a data file from scratch.

2. Overview

As already mentioned, Open Refine is a free tool that provides data wrangling features. The software can be downloaded from https://openrefine.org/download.html.

The way to work with Open Refine consists in the following steps:

1. Open the file
2. Explore the data
3. Cleanse the data
4. Save
5. Optionally, apply the same modifications to other files.

When opening the file, we need to be aware of some information, such as the file format or the encoding, since the software can perform different modifications upon loading that can save us a ton of effort.

The initial step before we perform different data wrangling tasks is to understand how the data looks. We need to be aware of several issues such as whether it contains typos, whether the data formats were interpreted properly (e.g. numbers loaded as numbers and not text), and so on and so forth.

Once we have a feeling of how the data looks and the potential defects to correct, we can proceed to modify the data.

Finally, the file can be saved.

Eventually, if the modifications were performed systematically and in a general manner, those can be applied (in the same order) to a different file with a single operation.

Open Refine has powerful operations that let you modify a lot of records at the same time. But it also has very useful undoing commands that let you undo (and redo) multiple editions. Moreover, the system also provides a very powerful previsualization of the results before applying a command, so that we can interactively check whether the command we are writing will perform as expected.

Before we dive into the different possibilities, let's give a small overview on the layout and organization of the application.

2.1 LAYOUT

After downloading and installing, running Open Refine will open a new tab in your browser (everything shown here has been tested with Mozilla Firefox), and you can start working. Initially, the software looks like as the screen shown in Figure 1.

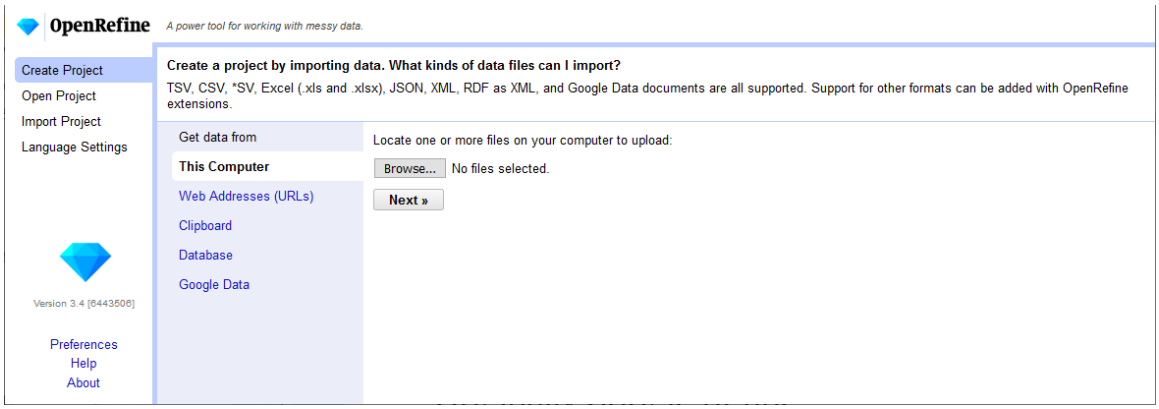

Figure 1: Initial Open Refine screen.

Although the application runs as a tab in a browser, no information is sent through the network (unless activating certain functions that require so). Therefore, your data will stay safe in your computer.

Note that the software may load different data formats besides the popular XML or CSV. You can even load files directly from URLs.

After opening a file, the application looks like the image in Figure 2.

The left pane will contain some data on the operations performed, as we will see later. The top menu lets you change or export the file, and the operations will be mostly carried out upon selection of the dropdown menu shown as a small arrow pointing down that appears next to each column and next to the initial one, called "All".

Operations executed on a single column, triggered by the dropdown menu appearing next to a column, will affect only the data in that column (see Figure 3). The menu that appears when using the dropdown of the "All" column is slightly different and contains operations that can be applied to the whole dataset at once.

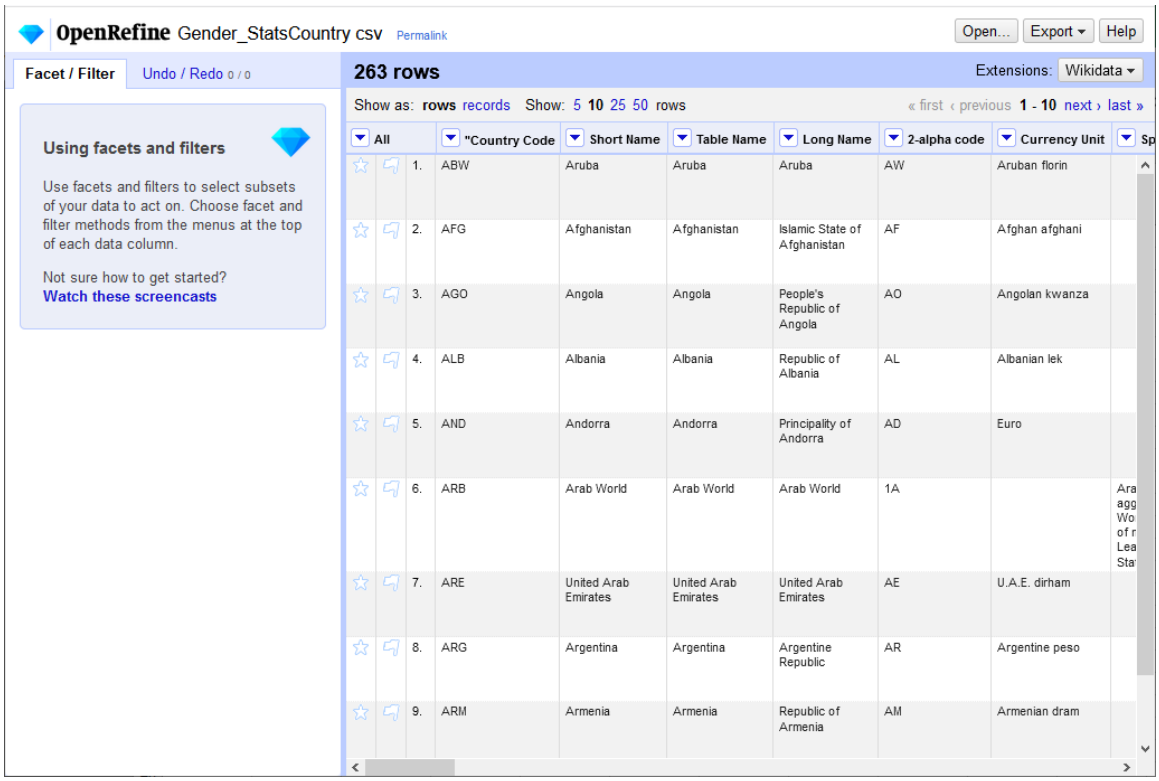

Figure 2: Open Refine initial view after loading a file.

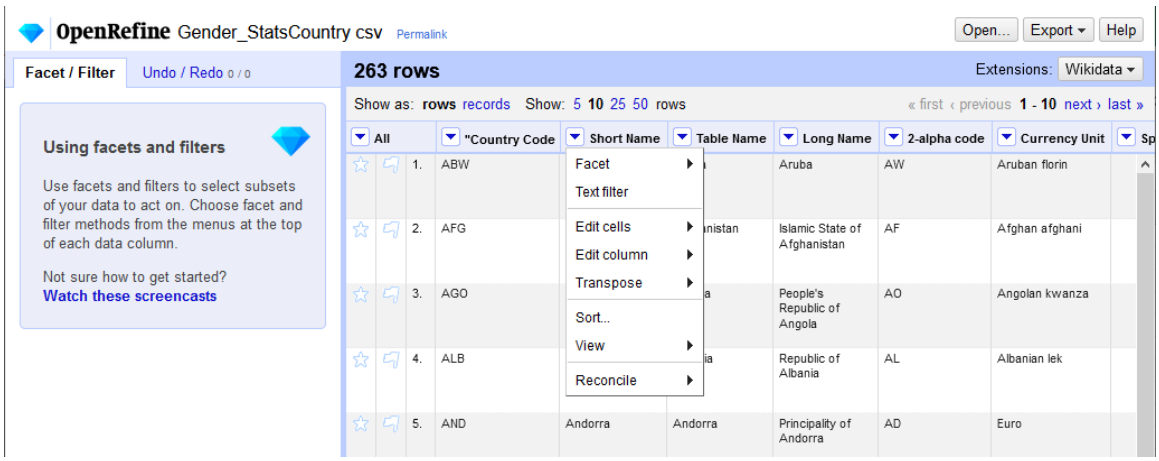

Figure 3: Menu showing the commands to apply to a single column.

Above the column titles, there is a row of operations that lets the user change the view to explore more (or less) records at once (the option of 10 records is the default) and navigating through the rest of the records (options “next”, “last” on the right).

2.2 EXPLORING DATA

For the rest of the examples, we encourage you to replicate everything that is described here. The data files used for the examples are available through the Racó.

With the aforementioned buttons, as well as the sliders, you can explore the dataset. This exploration is crucial since it has two objectives: get an idea on what the data contains, analyze the potential data defects.

In order to properly explore the data, we will load a real dataset. The first task you have to do is to open Open Refine and load the file *Gender_StatsCountry.csv*.

After the file is selected through the "Browse" button, you will find out the first problem: all the data appears in the same row.

The reason is that, despite being a CSV file, it has its odds that results in two issues:

1. The loader has not detected line changes.
2. The loader has not inferred row names.

There are several ways of addressing this problem using Open Refine. The simplest one is letting Open Refine know that the file contains the symbol " to enclose cells containing column separators. We need to disable the option highlighted in Figure 4.

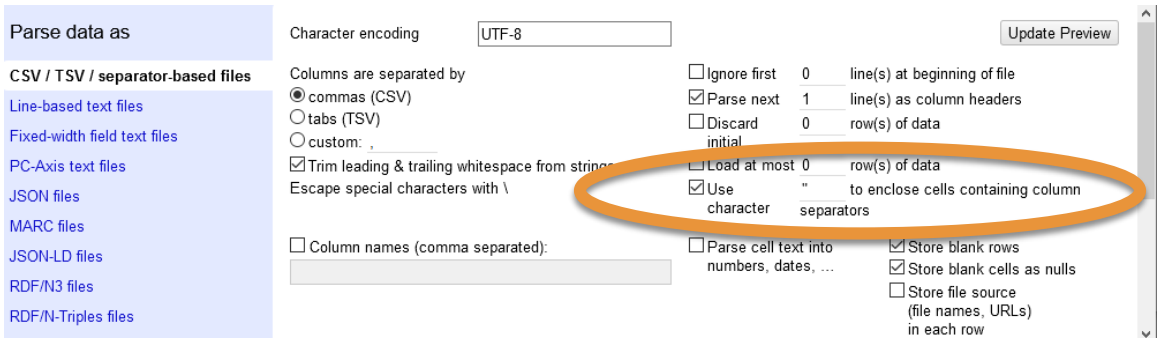

Figure 4: Ensuring Open Refine parses data properly.

The result will be the one appearing in Figure 5. Note that the result is not completely perfect, because we see some columns where the header contains the symbol ", such as "Country Code". We will deal with this later.

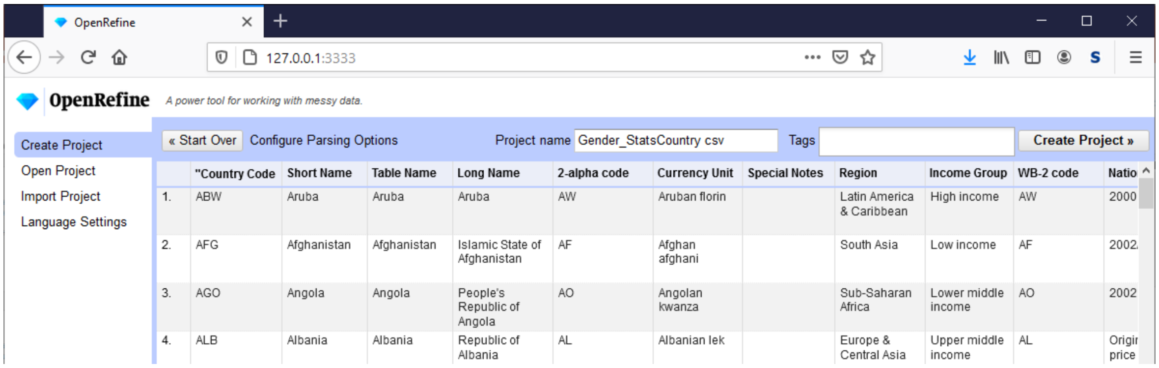

Figure 5: Loaded data after the previous modification.

Another parsing option is the one that let's Open Refine the input as a line-based file. In such case, changing the option to the one on the left as shown in Figure 6 may solve the problem.

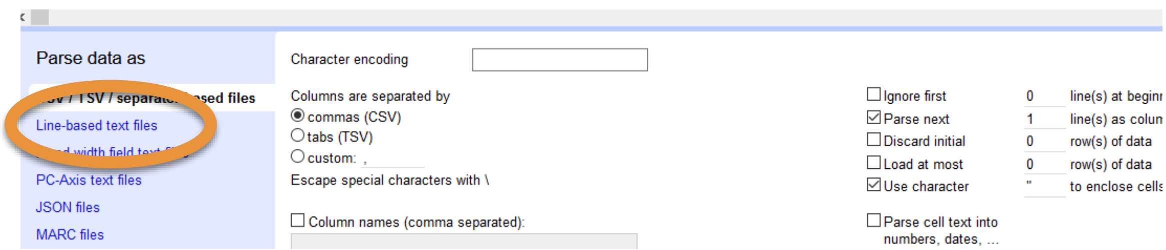

Figure 6: Loaded data after the previous modification.

Unfortunately, just doing so for our concrete example file, does not solve the problem properly, since, in this case, it does not detect file headers. The result of the reading procedure using this option, instead of the previous one would be the one shown in Figure 7. Note that each entry contains a line where all the values are stored as a single record.

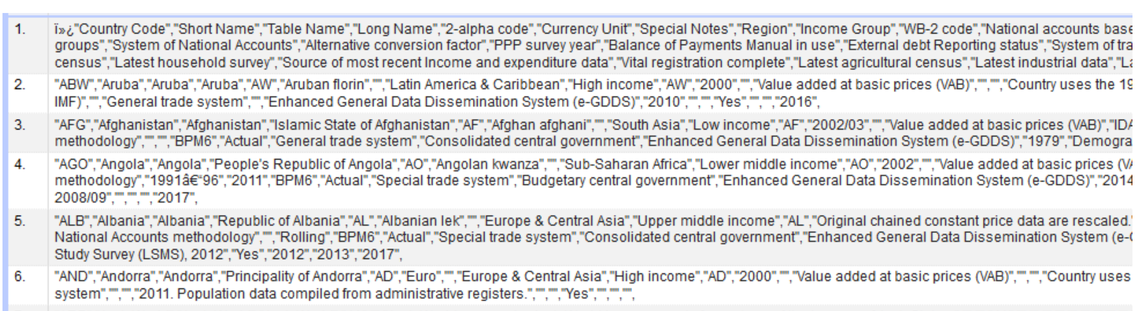

Figure 7: Result after loading the same file as a line-based one.

3. Data wrangling

In this section, we will introduce several data edition options. In order to illustrate those, we will start from the previous example, that is the file *Gender_StatsCountry.csv* loaded by using as parsing option the "line-based file". This will give us the opportunity of showing the most basic edition tasks.

3.1 BASIC DATA EDITIONS

The first thing we want to correct in this version of the file is to separate the data in different columns. Note that we only have two columns we can operate with: "All" and "Column 1". We can achieve this data splitting through (shown in Figure 8):

Dropdown Column 1 → Edit column → Split into several columns

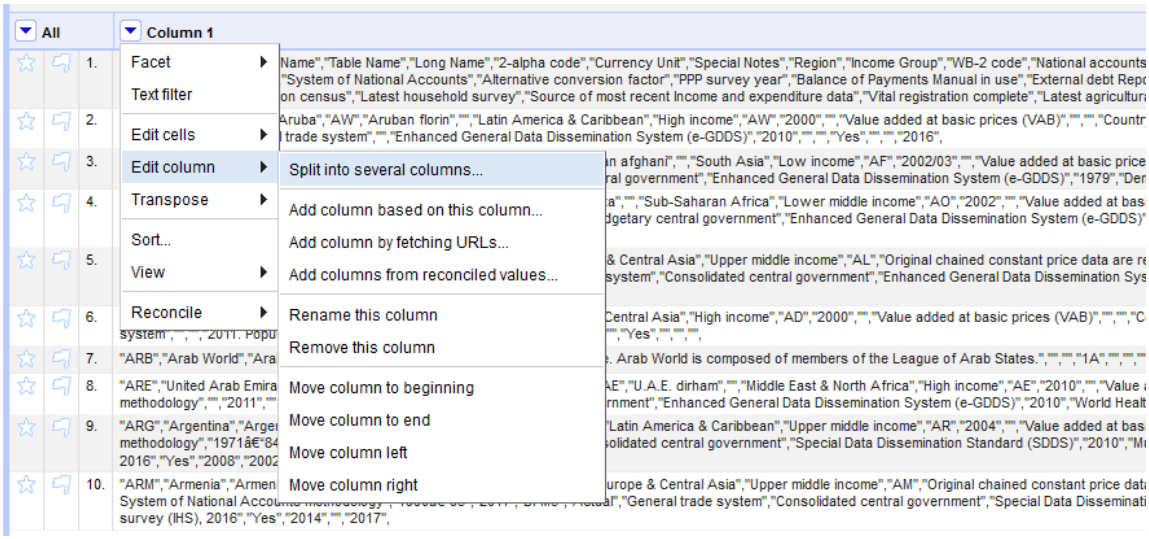

Figure 8: Column splitting.

In order to properly separate, we further need to tell Open Refine what separator is the one that delimits columns. In our case, it is the comma. Thus, we fill the form that appears upon selecting the previous option accordingly, as illustrated in Figure 9.

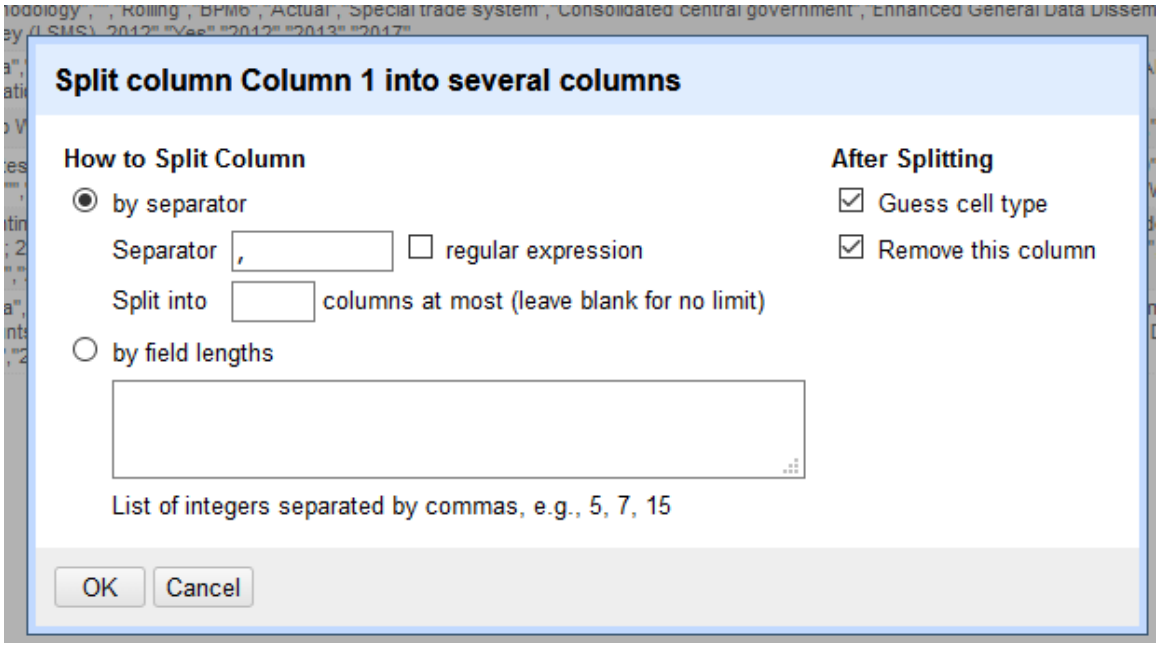

Figure 9: Configuring the separator.

The result will be a new version where the columns will be properly separated, as shown in Figure 10.

	Column 1 1	Column 1 2	Column 1 3	Column 1 4	Column 1 5	Column 1 6	Column 1 7	Column 1 8	Column 1 9
1.	ï»¿"Country Code"	"Short Name"	"Table Name"	"Long Name"	"2-alpha code"	"Currency Unit"	"Special Notes"	"Region"	"Income Group"
2.	"ABW"	"Aruba"	"Aruba"	"Aruba"	"AW"	"Aruban florin"	""	"Latin America & Caribbean"	"High income"
3.	"AFG"	"Afghanistan"	"Afghanistan"	"Islamic State of Afghanistan"	"AF"	"Afghan afghani"	""	"South Asia edit"	"Low income"
4.	"AGO"	"Angola"	"Angola"	"People's Republic of Angola"	"AO"	"Angolan kwanza"	""	"Sub-Saharan Africa"	"Lower middle income"
5.	"ALB"	"Albania"	"Albania"	"Republic of Albania"	"AL"	"Albanian lek"	""	"Europe & Central Asia"	"Upper middle income"

Figure 10: Result after column splitting.

Now we can proceed to clean some obvious typos: the inverted commas that plague all the records. Recall that this would be mostly avoided if we had properly loaded the file, but this example is only for didactic purposes.

To replace across the file a symbol, we can select the option "Transform" in the "All" column dropdown. This will show a form that can be completed with a function. In our case, it will be *value.replace(symbol to replace, replacing*

symbol). Note that inputting certain symbols in any programming language can be tricky. Since the symbol needs to be enclosed with inverted commas, and the symbol to replace is inverted commas too, the symbol to replace would be interpreted as the closing mark unless we modify it in a certain way. This is achieved using the symbol "\", that tells the language not to interpret what is coming next. Finally, in order to eliminate the symbol, we must tell the application that it must be replaced with the null string, which is represented by opening and closing inverted commas. Therefore, the concrete command to use is the one shown in Figure 11.

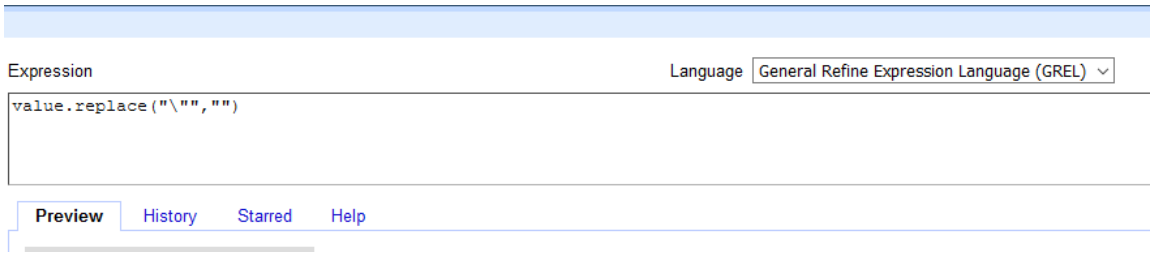

Figure 11: Replacing inverted commas across the file.

For the following examples, let's open the file called educ_uoe_enra02.tsv that contains the information regarding education levels in different countries of Europe.

Note that, after opening the file, among other problems, several fields have a white space and the "e" letter after the value (see Figure 12).

14443 rows

Show as: **rows** records Show: 5 10 25 **50** rows

All		unit,age,sex,isce	2017	2016	2015	2014	2013	2012
1.		NR,TOTAL,F,ED0,AT	148592	145862	142867	136544	133104	:
2.		NR,TOTAL,F,ED0,BE	222236	224351	224855	225519	222723	:
3.		NR,TOTAL,F,ED0,BG	108189	112067	116644	116675	113762	:
4.		NR,TOTAL,F,ED0,CH	83642	83676	81552	79858	76270	:
5.		NR,TOTAL,F,ED0,CY	15254	14733	14469	11163	11056	:
6.		NR,TOTAL,F,ED0,CZ	175894	178304	178234	176635	172195	:
7.		NR,TOTAL,F,ED0,DE	1533257	1499471	1462794	1441677	1399210	:
8.		NR,TOTAL,F,ED0,DK	134687	137656	141703	146392	147649	:
9.		NR,TOTAL,F,ED0,EE	32830	33122	33293	33275	32493	:
10.		NR,TOTAL,F,ED0,EL	:	107055	110728	152856 e	105661 e	:
11		NR,TOTAL,F,ED0,ES	861937	875499	890134	911643	925778	

Figure 12: Extra text in different fields.

We can modify this with different strategies, in this case, we are going to use the option "Edit cells" of the column to edit. Therefore, what we need to do is:

Dropdown of the column → Edit cells → Transform

This is illustrated in Figure 13.

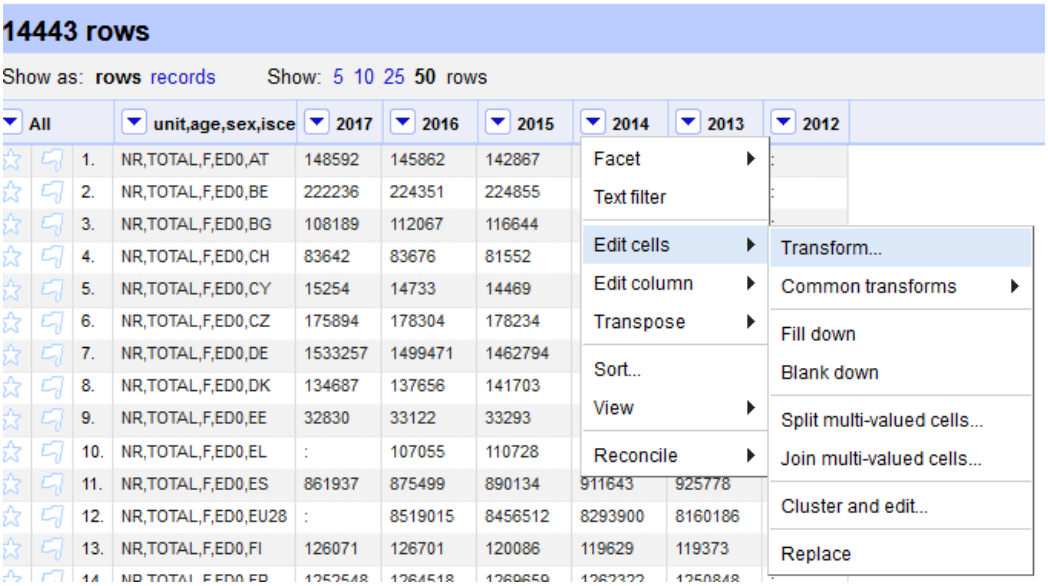

Figure 13: Transforming fields of a column.

Then, we can use the *replace* custom function and see in the bottom part the preview of the operation, as shown in Figure 14.

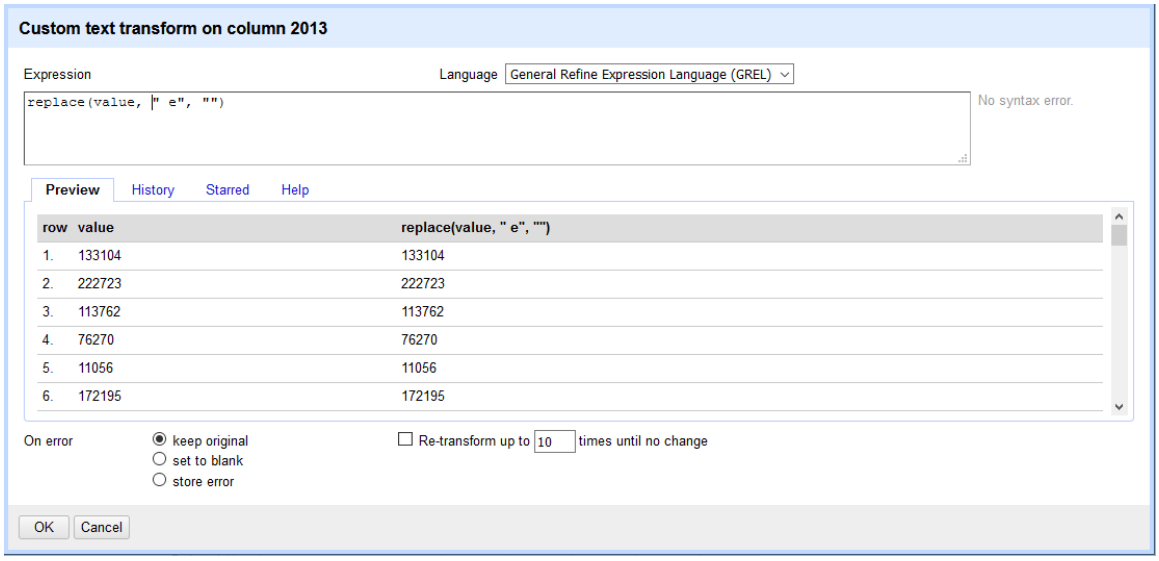

Figure 14: Replacing function with preview.

The same can be achieved with the replace command, that would not provide a preview, as shown in Figure 15. Note that in both cases, the whitespace is important, otherwise, it would replace the letter "e", not the extra whitespace and the letter.

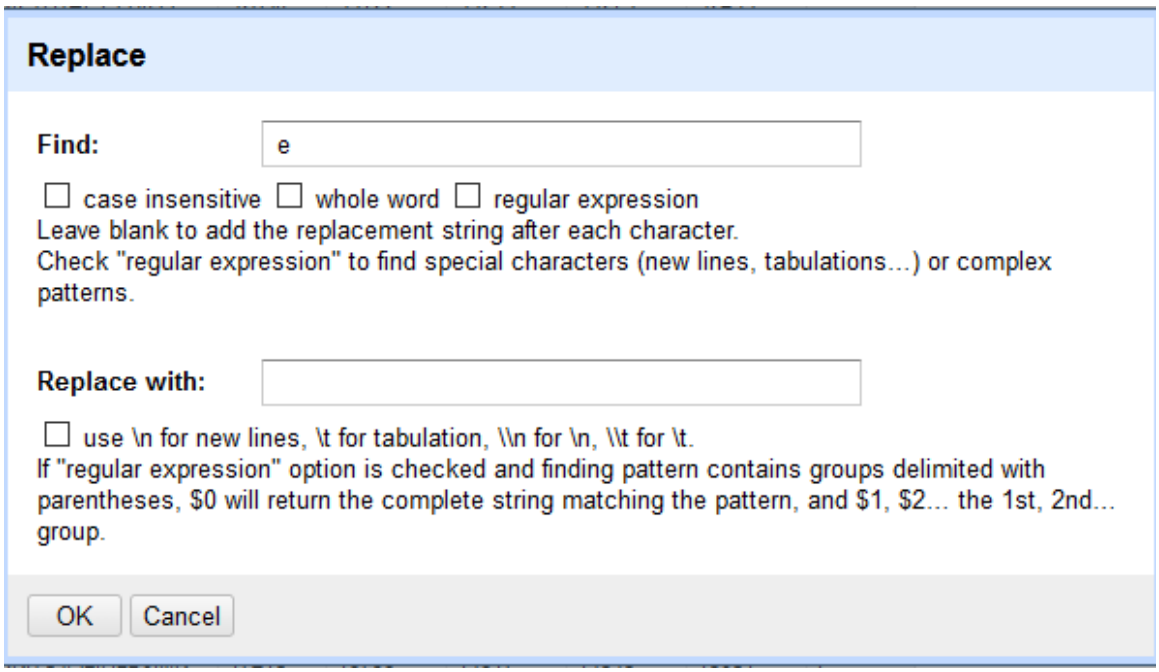

Figure 15: Replacing command. Observe the "e" is preceded by a white space.

We can now proceed with the same operations for the other columns.

To remove incomplete data, we can completely erase a full column. In our current example, the data of year 2012 is only available for Germany. If our problem lets us so, we can completely erase the year 2012 column. In other cases, we should complete the missing data with real values.

To erase a column, we just need to go to "Edit column" and select "Remove this column".

Another thing that we see is that the years are not ordered in ascending order. We can move columns with the "Edit column" option. In this case, we move the year 2017 to the end by using "Edit column" and "Move column to end", as illustrated in Figure 16.

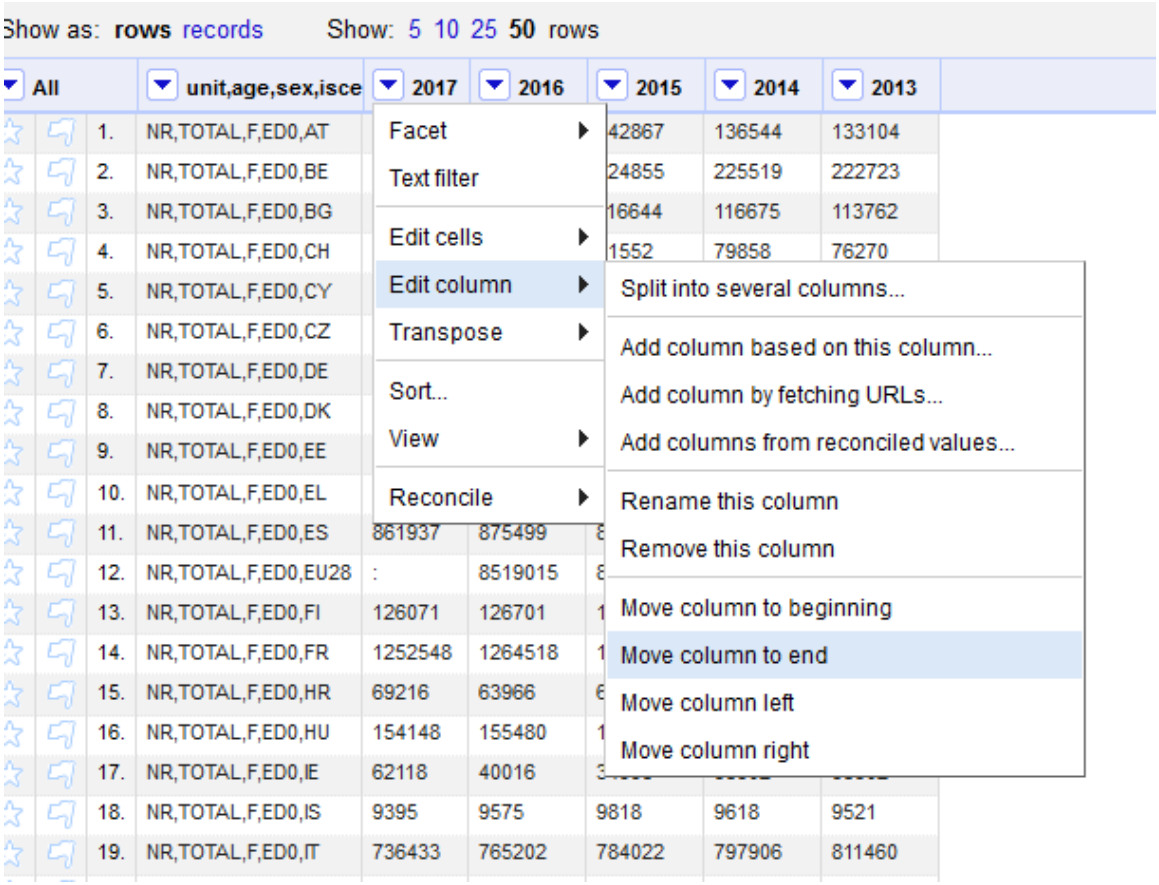

Figure 16: Moving the 2017 column to the end.

We can reorder the other columns with the moving commands ("Move column to beginning", "Move column to end", "Move column left", and "Move column right").

Other minor editions consist in erasing trailing whitespaces, collapsing consecutive whitespaces, and changing the data type (e.g. text to number). You can access all of these transformations by selecting the "Common transforms" option under the "Edit" menu, as depicted in Figure 17.

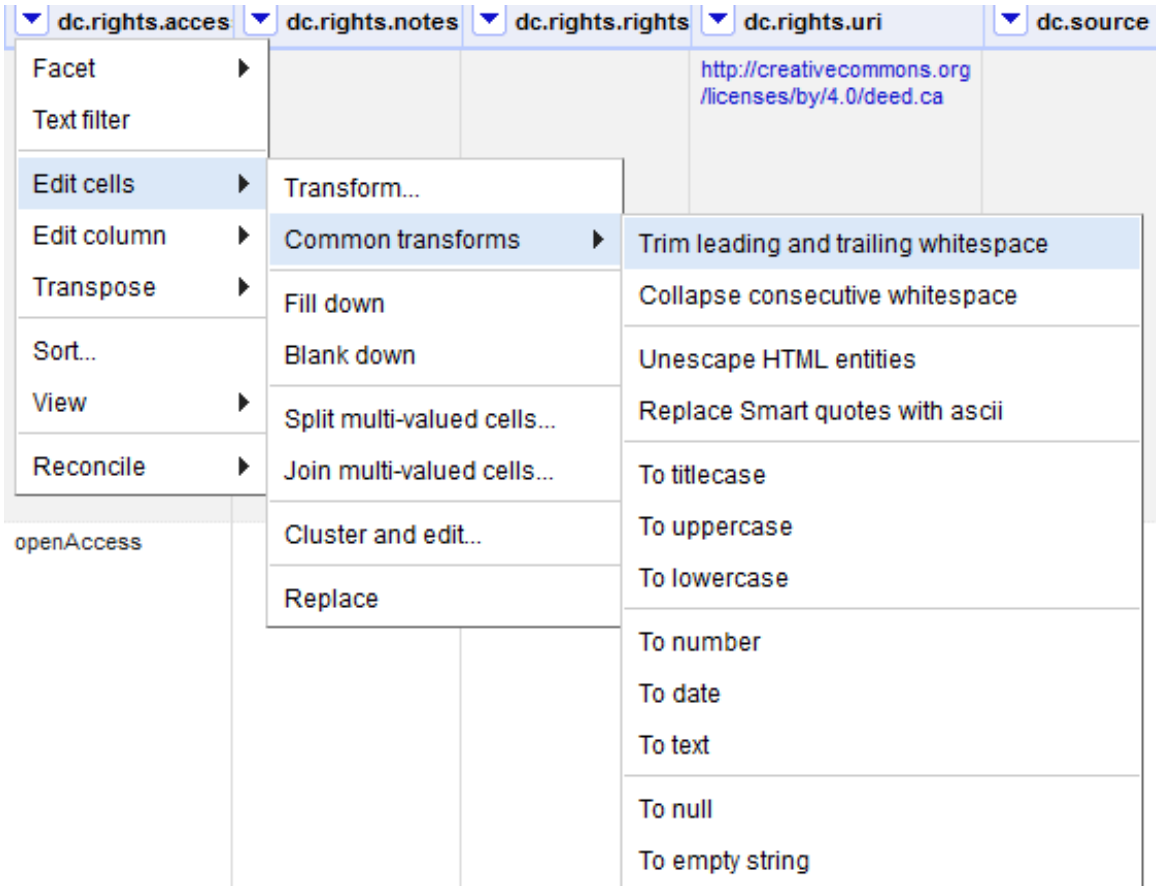

Figure 17: Simple data transformations.

Note that you can also change data to uppercase lowercase, and so on, with the same menu.

Other not so obvious operations consist on filling blank values with the previous value that appears in a column, or filling the following values with blanks. Both of those are also accessible with "Edit cells" + "Fill down" or "Blank down".

So far, the presented operations are rather simple, and many software packages such as spreadsheets or text editors may have some available function that might do the job as easily. In the following section, we will see some complex operations that are typically only available through data wrangler applications or programmatically.

3.2 ADVANCED DATA EDITIONS

In the following section, we will deal with some more advanced transformations that can lead to both detecting defective data and its proper correction. The complexity of most of the following techniques can also be tackled by using a custom program to clean the data. However, data wrangling software provides

some features such as previews of the operation effect, or the possibility of undoing the editions that are not simple with a custom program.

The first operation we are going to see is the "Cluster & Edit" function. Its main purpose is to collectively inspect and edit cells. A common problem in data analysis happens when the same value appears written differently for many cells. This is not easy to detect, since we need to inspect the data and this may become unfeasible if the file is very large.

The "Cluster & Edit" function finds groups of representations and opens a detail view that provides some insights on the groups, such as the number of rows that have each representation, and offers the possibility of merging and editing values of multiple rows (even with original different values) at once.

In order to use the function, we will move to the file named Publicacions_369.csv. It stores information on publications released by different departments, with the names of the authors, title, and all sorts of other data.

If we cluster by column "dc.contributor.author", we will find that some names were written differently, as shown in Figure 18.

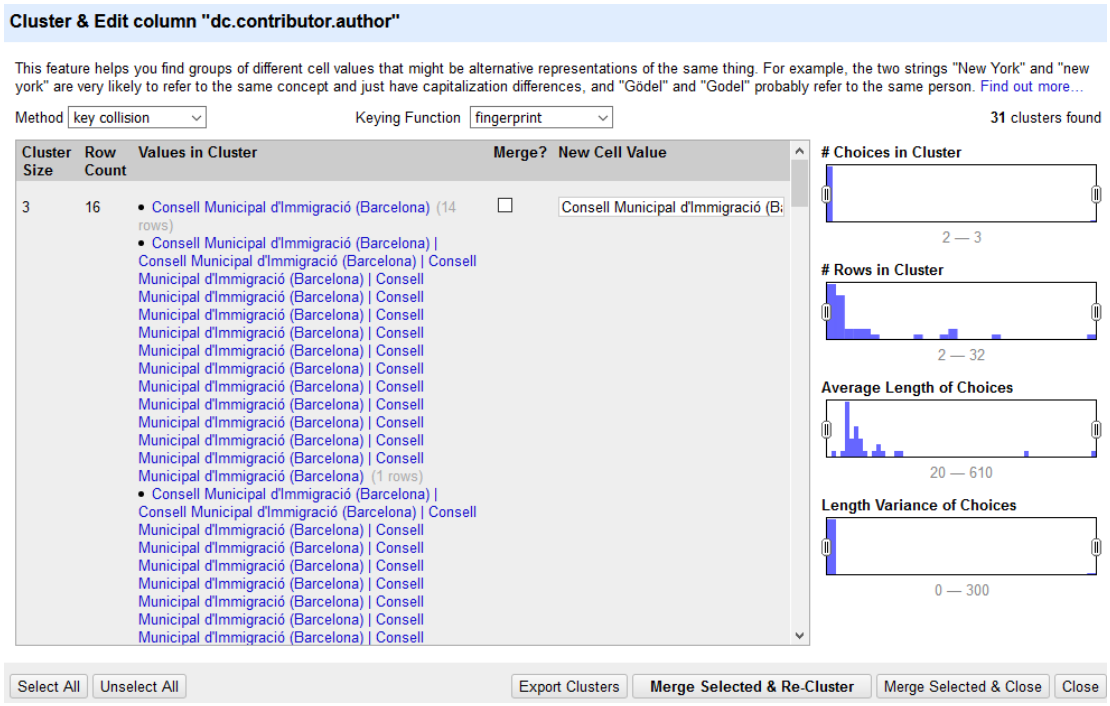

Figure 18: Clustering column.

The charts on the right show the statistics of the found values. The left part shows the different names that were applied to the values in the column. The algorithm tries to find names that are repeated but written wrongly (e.g. due to a typo, or

because wrongly placed trailing spaces, and so on). As a result, we will be offered a set of candidate values that the algorithm believes that should be the same, and thus, we can fix it.

In our example, we can see that the "Consell Municipal d'Immigració (Barcelona)" is wrongly written (repeating the name several times) in some rows. By clicking on the "Merge" checkbox and the "Merge Selected & Re-Cluster", the rows will be updated and will disappear from the view since all the values are the same now (see Figure 19). Similar names might not be detected as belonging to the same cluster. By iteratively correcting the wrong ones, you should converge to the proper solution. The next iteration shows some examples with rows that have the same authors, but they were ordered differently. If the order does not matter (in some cases it might), we can fix this easily. In this case, we can fix the four at once.

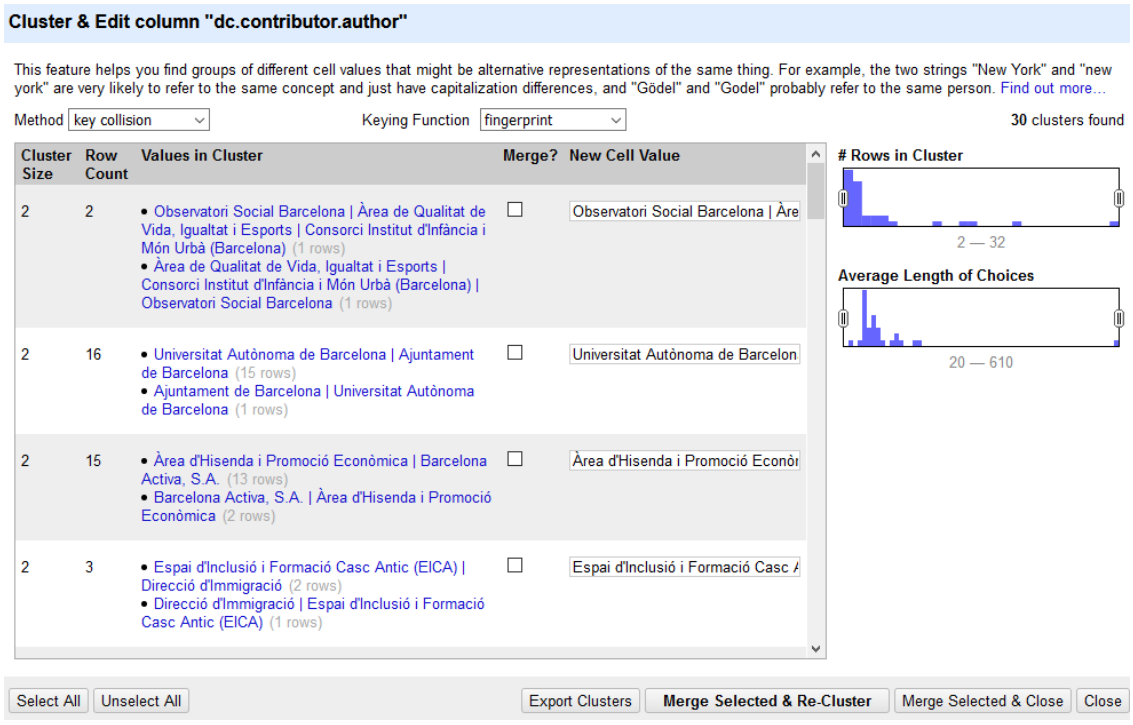

Figure 19: Re-clustering result after fixing the first issue.

The clustering operation is not only useful to detect typos, but also to detect anomalies in values.

For example, in columns that store numerical values, the cluster operation will show the distribution, and thus we may be able to detect some values that are out of the expected range.

Another operation that is complementary to the clustering is the "Facet". It can show us the distribution of values. We are going to use the file IndicadorsCOVIDBarcelona.csv downloaded from the Barcelona Open Data site on the 19th of September.

We want to track the how frequently the data was gathered. In order to do so, we are going to facet the date ("Data_Indicador"). Unfortunately, the loading process did not detect the column as a date. We can fix it with the proper transformation option (to date).

Then, we can facet the column with "Facet" and selecting the "Timeline gacet" option. We will see something strange: some indicators were captured in 2001 (see Figure 20).

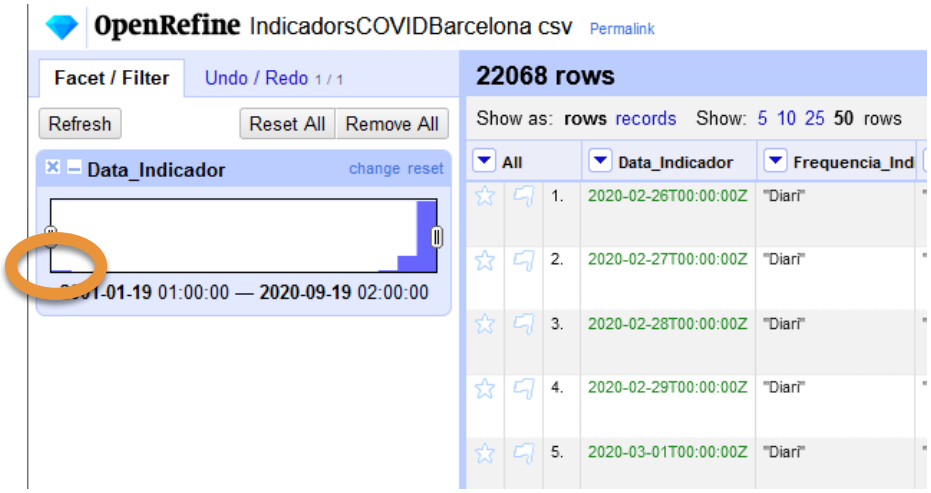

Figure 20: Faceting the date column.

We can inspect what happens here by adjusting the right view to the rows in this range. This can be easily done by dragging the controllers at the left and right borders of the facet view (shown in Figure 21).

Figure 21: Widgets to resize the facet preview.

If we drag the right widget to leave the 2001 year elements within the selection, we can see the problem, as depicted in Figure 22.

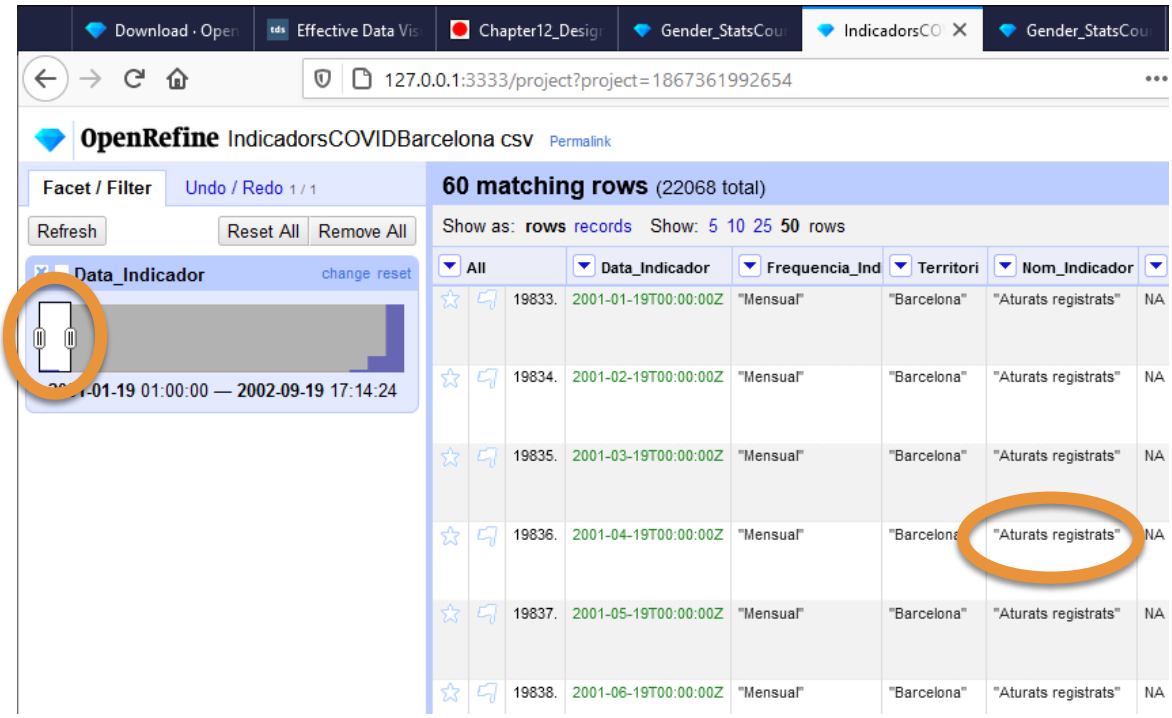

Figure 22: Rows of 2001 belong to another database, since they show the number of unemployed people.

We can remove those rows that are obviously wrong with the "All" option named "Remove matching rows". Upon this change, we will see that there are still some rows with non-valid dates.

Finally, a more complex operation consists in augmenting the data. Basically, we need this when there is missing data, or we need to derive new values. Open Refine provides a powerful mechanism through the use of web queries from URL addresses.

For example, in the file named 2018_padro_nivell_academic.csv, we have neighborhood names. If we want to use values of this file and put them in a map, we would need some sort of location to generate the geometric elements. So what we are going to do now is to get a GPS position from a neighborhood.

In order to do so, we go to the column named "Nom_Barri" and select "Edit column" and "Add column by fetching URLs". The form that will open looks like the one in Figure 23.

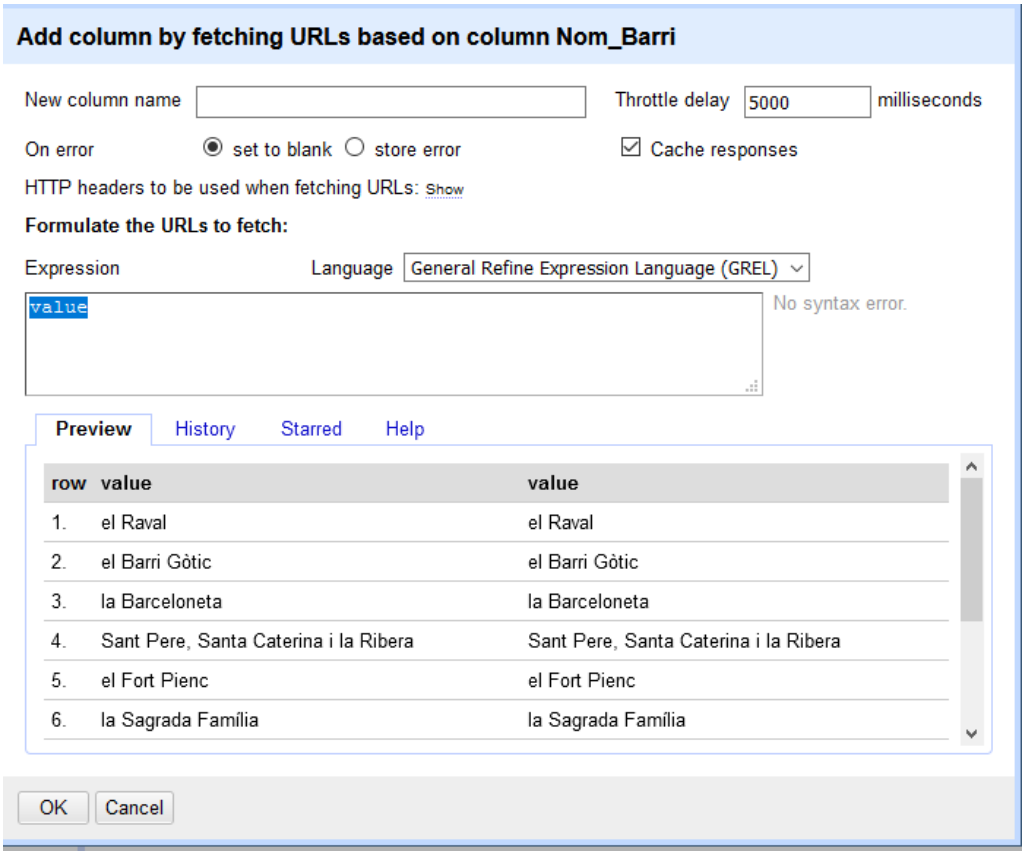

Figure 23: Data augmentation through fetching form a URL.

Then, we need to define a new column name (top box), change the delay to something faster (e.g. 1500ms, since Openstreetmap does not want more than one call per second), and enter the url of the query. It should be written as in figure 24.

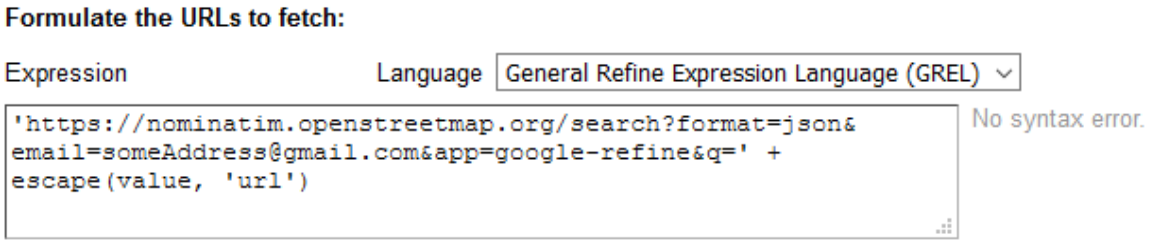

Figure 24: Openstreetmap query.

Note that in this case, the preview window is unable to fully resolve the query, but it shows what query will be applied for each value.

The result will be a new column, where each cell contains a json code that contains (among a lot of other information), the GPS position of the neighborhood.

To extract the information of the proper fields, we can add a new column using this information with the command "Edit column" and "Add column based on this column", as shown in Figure 25.

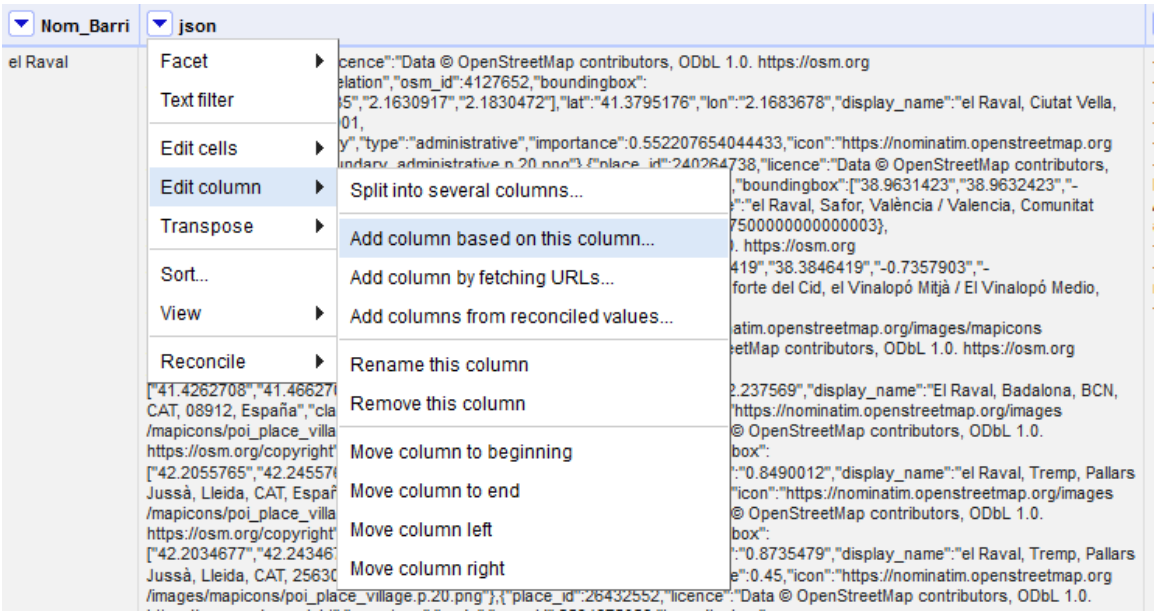

Figure 25: Adding a new column to extract the GPS coordinates from the json code.

In this case, we extract the latitude and longitude using the *parseJson* function and the *with* expression. The code is illustrated in Figure 26.

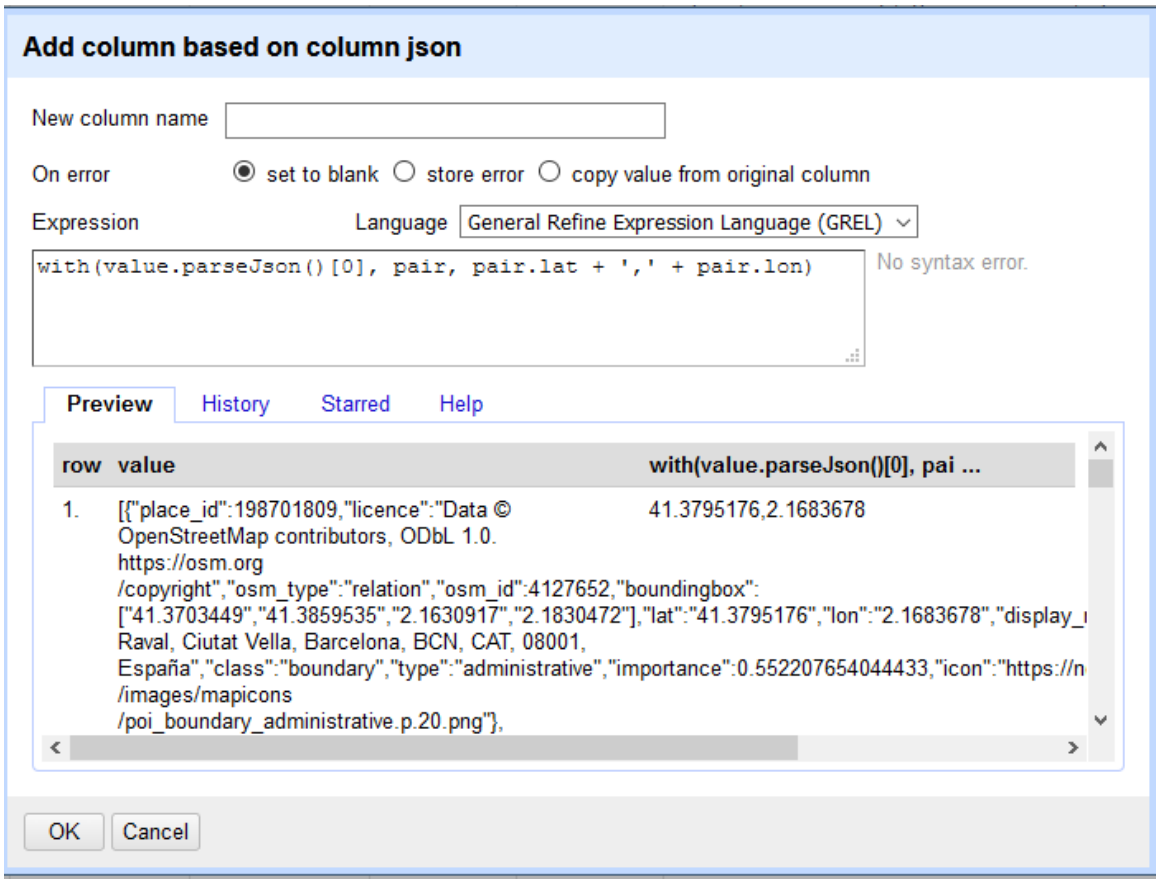

Figure 26: Extracting the GPS position from the json code.

Note how the preview already shows the result.

Then, this result may be split in two different columns.

Recall that you can export the current file at any moment with the "Export" menu on top.

4. Exercises

1. Analyze and clean the file "inca_od_20191115.csv".
2. Analyze and clean the file "airQualityDailySummary.csv".