Intersect Australia Ltd
PO Box H58, Australia Square
Sydney NSW 1215

T +61 2 8079 2500
F +61 2 9262 3040

ABN 67 131 752 657

Cleaning and exploring your data
with OpenRefine

Dr Luc Small & Dr Jeff Christiansen & Dr Richard Berry | 10 Apr 2017 | 2.1

This tutorial includes material from *Library Carpentry: OpenRefine Lessons for Librarians*, June 2016, https://data-lessons.github.io/library-openrefine/. Licence: (cc) BY https://creativecommons.org/licenses/by/4.0/

1. Overview

OpenRefine is a powerful free tool for exploring, normalising and cleaning up datasets. It is most useful where you have data in a simple tabular format, but with internal inconsistencies either in data formats, or where data appears, or in terminology used. It can help you:

- Get an overview of a data set

- Resolve inconsistencies in a data set

- Help you split data up into more granular parts

- Match local data up to other data sets

- Enhance a data set with data from other sources

In this tutorial we'll work through the various features of Refine, including importing data, faceting, clustering, and calling into remote a API, by working on a fictional but plausible humanities research project. We'll start with a research question in mind and use the features of OpenRefine to gain insights and find answers.

The research question relates to NSW police stations — finding out what we can about where they are located, their heritage status, and identifying and removing duplicate entries.

2. Resources

OpenRefine:

- http://openrefine.org/

OpenRefine Documentation:

- http://openrefine.org/documentation.html

- https://github.com/OpenRefine/OpenRefine/wiki/Documentation-For-Users

- https://github.com/OpenRefine/OpenRefine/wiki/Recipes

The Google Geolocation API:

- https://developers.google.com/maps/documentation/geolocation/intro

Source of the original dataset as hosted by the NSW Office of Environment and Heritage:

- http://www.environment.nsw.gov.au/heritageapp/heritagesearch.aspx (then Basic Search for "police station")
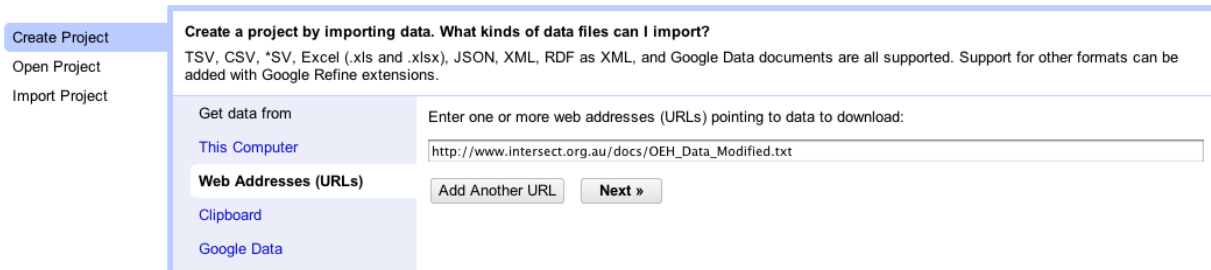
3. Installing OpenRefine

To install OpenRefine:

1. Go to the main OpenRefine website:

   - http://openrefine.org/

2. Browse to the **Download OpenRefine** section.

3. Choose the appropriate download for your operating system. Windows, Mac and Linux are all supported.

4. Follow the installation procedures for your operating system.

4. Starting a Project

1. Launch OpenRefine. It will open in your default web browser. Note: OpenRefine is not a cloud application; it runs locally, using your web browser as its primary interface.

2. Select the **Create Project** tab.

3. Select the **Web Addresses (URLs)** option.

4. Enter http://bit.ly/zonJkP (or, alternatively,

   http://www.intersect.org.au/docs/OEH_Data_Modified.txt) in the **Data file URL** field.
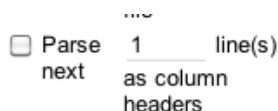
5. Click **Next >>**.

6. On the resultant screen, set Project name to The Bill:
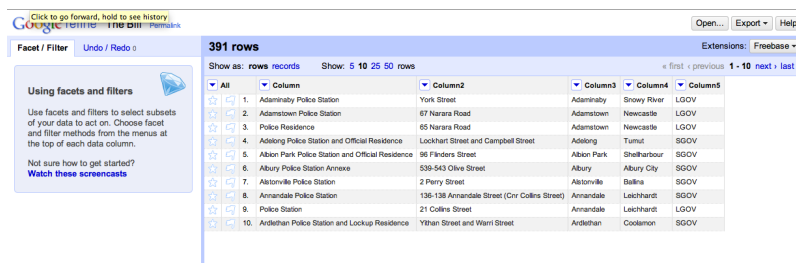
7. Untick **Parse next _1____ line(s) as column headers**:

8. Click **Create Project >>**.

9. The project will open with 391 rows:

10. By default OpenRefine only displays 10 lines of text at one time. You can adjust the number of lines shown by selecting **5, 10 , 25 or 50** from the navigation bar at the top of the screen.

391 rows

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

11. You can also select **<< first**, **< previous**, **next >** or **last>>** to scroll between different pages of data.

Extensions:

« first ‹ previous **1 - 10** next › last »
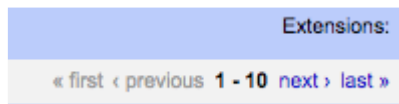
2. Getting Organised

Renaming columns

1. For *Column 1* select **Column menu > Edit column > Rename this column**.
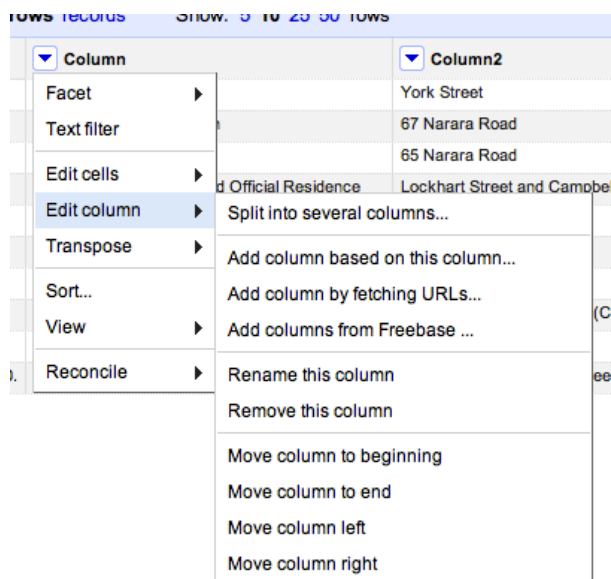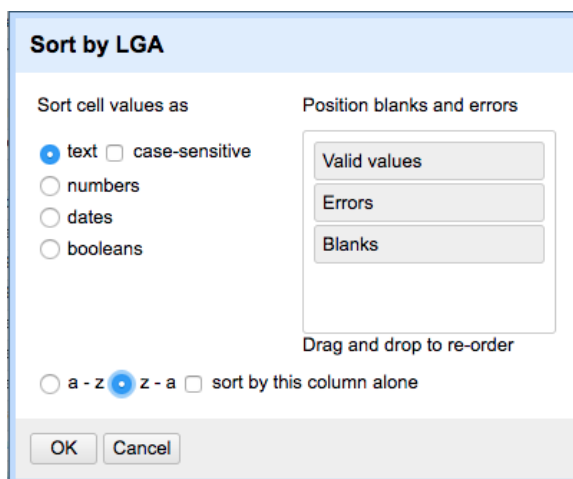
2. Rename to *Station Name*. Click **OK**

3. Rename *Column2* to *Address* as above.

4. Rename *Column3* to *Suburb* as above.

5. Rename *Column4* to *LGA* as above.

6. Rename *Column5* to *Source_Heritage*

3. Exploring and cleaning LGA and Suburb data

Sorting by the LGA column

1. Select **LGA menu > Sort**

2. Select **text** and **z-a** and click **OK**. All entries now appear in reverse order based on LGA.

3. Notice the different convention used for the LGA field of the first two entries. Mouse over *Yass Val.* on the second entry and select **Edit**.

4. Replace the text with `Yass Valley` and select **Apply**.

Facet on LGA

1. Select **LGA menu > Facet > Text Facet**. Note that a **LGA** facet will appear on the left hand side of the screen. This shows a list of unique LGAs in the data.

2. Click on **126 choices**. A text box will appear so we can copy and paste our list of unique suburbs into, say, a document.

3. You can click on any given LGA to view only the records matching the LGA. For instance, click on *Albury City* to filter to this LGA. Click **exclude** to remove the filter.

4. Scroll to the bottom of the LGA facet. Note that there are two separate entries with the LGA *Yass Valley*.

5. Select **LGA menu > Edit cells > Common transforms > Trim leading and trailing whitespace**.

6. The LGA facet now shows a single entry for *Yass Valley*.

7. Select **LGA menu > Sort > Remove Sort**. The data is now returned to the original order.

8. Select **Remove All** to remove the LGA facet.

Facet and cluster on Suburbs

1. Select **Suburb menu > Edit cells > Common transforms > Trim leading and trailing whitespace**. This will ensure any trailing whitespace is removed from values in the Suburb column.

2. Select **Suburb menu > Facet > Text Facet**. Note that a **Suburb** facet will appear on the left hand side of the screen. This shows a list of unique suburbs in the data.

3. Click **Cluster** to reveal and fix some consistency issues with the dataset. Select, for instance **nearest neighbor** as the method. You'll see that Refine finds some near matches. Now try some of the other methods.

4. You can make your data more consistent by typing the correct value into **New Cell Value** and ensuring the **Merge?** checkbox is selected. Use the **Merge Selected & Re-Cluster** function to actually modify the dataset.

Note that OpenRefine will suggest the most frequent value in the cluster as the suggested "New Cell Value". For example, "Murwillumbah" will appear in "New Cell Value" because it appears in 2 rows of the dataset, while the misspelled "Murwilumbah" only appears in one.

4. Separating source and heritage into different columns

1. Select **Source_Heritage menu > Edit column > Add column based on this column...**

2. Set **New column name** to *Heritage Listed*.

3. Set **Expression** to:

   ```
   if(value == "Yes", "Yes", "No")
   ```

   OpenRefine uses GREL (Google Refine Expression Language). More info on GREL syntax can found here http://code.google.com/p/google-refine/wiki/UnderstandingExpressions.

   The GREL expression used in step 3 is being used to add values to a new column, and these values are based on the existing values found in the Source_Heritage column. This column is messy and contains various values of mixed meanings (i.e. the **Source of information in the row:** "LGOV (Local Government)", "SGOV (State Government)", "GAZ (NSW Government Gazette)"; and **if the Police Station is Heritage listed:** "Yes". Here we're going to start cleaning the data by separating the mixed meaning of the Source_Heritage column into 2 new columns with specific meanings: the Source of the data and the Heritage Status of each Police Station.

   The first GREL expression is related to the Heritage Status of each Police Station:

   GREL Expression: `if(value == "Yes", "Yes", "No")`

   Meaning: for every row, **if the value of the cell in column 5 equals "Yes"**, **then in the new column add a value of "Yes",** otherwise in the new column add a value of "No".

   ---

4. Select **Source_Heritage menu > Edit column > Add column based on this column...**

5. Set **New column name** to *Source*.

6. Set **Expression** to:

   ```
   if(value != "Yes", value, "")
   ```

   The second GREL expression is to denote the source of information ("LGOV", "SGOV" or "GAZ")

   GREL Expression: `if(value != "Yes", value, "")`
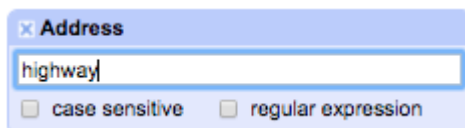
   Meaning: for every row, **if the value of the cell in Source_Heritage does not equal "Yes"** then in the new **column,** insert a value in the new column of "" (i.e. whatever the value was that appeared in Source_Heritage)

   ---

7. Select **Source_Heritage menu > Edit column > Remove this column**

5. Exploring address information

Using filters and finding addresses spanning multiple street numbers

1. Select **Address menu > Text Filter**. Type *highway* in the text box that appears on the left hand side of the screen. Only stations located on a highway now appear in our data set.

> ✕ **Address**
>
> highway|
>
> ☐ case sensitive   ☐ regular expression

2. Select the regular expression check box in the Address field. Replace the text in the filter with `^\d`.

> ✕ **Address**
>
> ^\d.
>
> ☐ case sensitive   ☑ regular expression

3. The Regular Expression shown here finds all addresses that starts with a digit, for example *67 Narara Road*. Regular expressions are a whole topic to themselves, but can be very useful for complex pattern matching.

4. Select **Remove All** to remove the filter.

Finding duplicate addresses

1. Select **Address menu > Facet > Customised facets > Duplicates facet**. A facet containing the entries *true* and *false* appears on the left hand side of the screen.
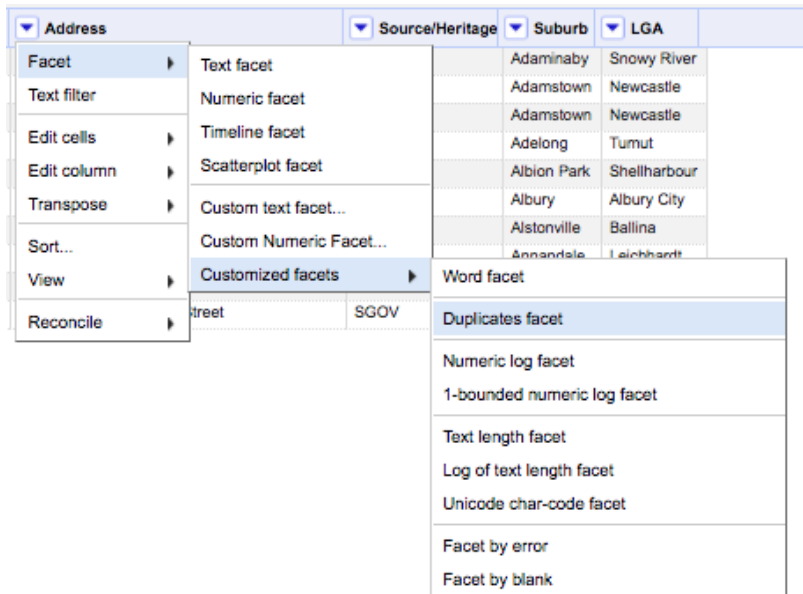
2. Click on `true` to only show entries where there is more than one entry with exactly the same address.

3. Select **Address menu > Sort.** Select **text** and **a-z** and click **OK**. Sorting by address makes it easier to identify entries with similar addresses.

4. What can you tell from looking at the data? Are there multiple entries for the same station (perhaps from different data sources?). Are there entries that have the same address but different station or suburb details?

5. Select **Remove All** to remove the filter.

6. Select **Address menu > Sort > Remove Sort**. The data is now returned to the original order.

6. Collapsing duplicate rows into a single record

In this and the following section we will work through a process to identify as many duplicate entries from our dataset as we can. Consider the following two entries:

	Station Name	Address	Suburb	LGA	Source	Heritage Listed
273	Paddington Police Station	16 Jersey Road	Paddington	Woollahrah	SGOV	No
274	Police Station and Court House	16 Jersey Road	Paddington	Wollahra	LGOV	No

Although the station name is different, we can be confident that they are duplicates because they refer to the same address, suburb and LGA. However we can also see that the LGA is slightly different, with an extra 'o' in the first entry. OpenRefine's clustering ability is perfect for dealing with these inconsistencies, but it does not allow us to use clustering across multiple columns at the same time.

In order to identify and cleanup our duplicates entries we need to create another **compound** column that is a combination of the `Address`, `Suburb` and `LGA` columns. Using a combination of three locations fields allows us to make sure that, for example, 16 Jersey Road in Paddington is not accidentally merged with 16 Jersey Road in another suburb.

The steps to do this are:

1. Create the compound column - for example "`16 Jersey Road | Paddington | Woollahrah`"
2. Use clustering on this column to identify and merge similar entries - merging "`16 Jersey Road | Paddington | Woollahrah`" with "`16 Jersey Road | Paddington | Wollahrah`"
3. Use a feature of OpenRefine called **Records** to identify entries with exactly the same value in this field.

Let's get started.

Creating a composite column

1. Select **Address menu > Edit Column > Add column based on this column...**

2. Set **New column name** to *Compound Key*

3. Set **Expression** to:

```
value  +  "  |  "  +  cells["Suburb"].value  +  "  |  "  +
cells["LGA"].value
```

> The second GREL expression is to combine the values across three columns (Address, Suburb and LGA)
>
> **GREL Expression:** *value + " | ” + "cells[“Suburb”].value + " | ” + cells[“LGA”].value*
>
> **Meaning:** for every row**, get the value of the cell in the "Address" column add spaces and a "pipe" symbol to this to clearly separate this from the next field,** add the value of the cell in the Suburb column **add more spaces and another pipe,** then add the value of the cell in the LGA column

4. Click **OK** to create the column.

Remove any extra whitespace

1. Select **Compound Key menu > Edit Cells > Common Transforms > Collapse Consecutive Whitespace**. This will remove all extra whitespace and ensure there is only a single space between all words and the special "|" (pipe) separator.

2. Select **Compound Key menu > Edit Cells > Cluster and edit**. Select **Nearest Neighbour** as the method and **PPM** as the distance function. Choose **Select All** then **Merge Selected & Close** to actually modify the dataset.

Reorder and remove excess columns

1. Select **All > Edit Columns > Re-Order / remove columns…**

2. Left click and drag **Compound Key** to the top of *Columns to re-order*. Drag **Address**, **Suburb** and **LGA** to *Columns to remove*. Click OK.

Re-order / Remove Columns

Drag columns to re-order	Drop columns here to remove
Compound Key	Suburb
Station Name	LGA
Source	Address
Heritage Listed	

Merging rows into records

1. Select **Compound Key menu > Sort.** Select **text** and **a-z** and click **OK**. Sorting by the composite column will allow us to identify entries with exactly the same address, suburb and LGA.

2. Select **Sort > Reorder rows permanently**. Until this option is selected, sorting operations in OpenRefine are temporary and do not affect the actual order of the saved data. As the name suggests, selecting this option makes the reorder permanent.

3. Select **Compound Key menu > Edit Cells > Blank down**. This will delete the contents of any adjacent cells with identical values, leaving only the topmost row.

4. Select **Records** from the navigation bar at the top of the screen. You should now see rows referring to the same address, suburb and LGA grouped together under a single 'record'. Note also that the total number of records is less than the total number of rows. In this case, we have 315 Records, as opposed to 391 rows. This indicates we have many duplicate entries (the exact number of records you have in your dataset will depend on how many rows you clustered and merged earlier).

7. Removing duplicate rows

Now that we have identified duplicate entries and cleaned the `Compound Key` column, we still have to clean the rest of the fields then remove the duplicates. Consider the following example again:

Compound Key	Station Name	Source	Heritage Listed
16 Jersey Road | Paddington | Woollahra	Paddington Police Station	SGOV	No
16 Jersey Road | Paddington | Woollahra	Police Station and Court House	LGOV	No

Thanks to clustering, the `Compound Key` column is now consistent. However we still have different values for the `Station Name` and `Source` columns. If we were to delete one of these rows we would lose this potentially valuable data. In order to prevent this, we will flatten this information into a single row by joining the contents of two cells together before we delete the duplicates. We also need to restore our `Address`, `Suburb` and `LGA` fields that we used to create the `Compound Key` column. The end result we want to achieve is something like this:

Station Name	Address	Suburb	LGA	Source	Heritage Listed
Paddington Police Station | Police Station and Court House	16 Jersey Road	Paddington	Woollahra	SGOV | LGOV	No

The overall steps are:

1. Collapse the values from the `Station Name` and `Source` columns into a single row
2. Cleanup some inconsistencies in the `Heritage Listed` column
3. Remove the duplicate entries
4. Split the `Compound Key` column back into separate columns

Collapsing data in the Source column

1. While still in **Records** view, select **Source menu > Edit cells > Join multi-valued cells…**

2. Replace the comma with " | " (note the spaces between the pipe characters) and select **OK**.

Where a record contains multiple rows from different sources this has now been merged into a single row, with different sources separated by a comma value.

Collapsing data in the Station Name menu

1. Select **Station Name menu > Edit Cells > Blank down**. This removes adjacent identical values.

2. Selection **Station Name menu > Edit Cells > Join multi-valued cells…**.

3. Replace the comma with " | " (note the spaces between the pipe characters) and select **OK** to merge the data from multiple rows in a record into the only top-most row.

Fixing errors in heritage status

The data for this dataset was combined from two different sources, with only one of these sources containing information on whether or not the station is heritage listed. We therefore have some duplicate entries for the same station, with some entries giving correctly giving the heritage status as "Yes", and others as "No". We need to fix this inconsistency in order to be able to merge the duplicates to a single entry.
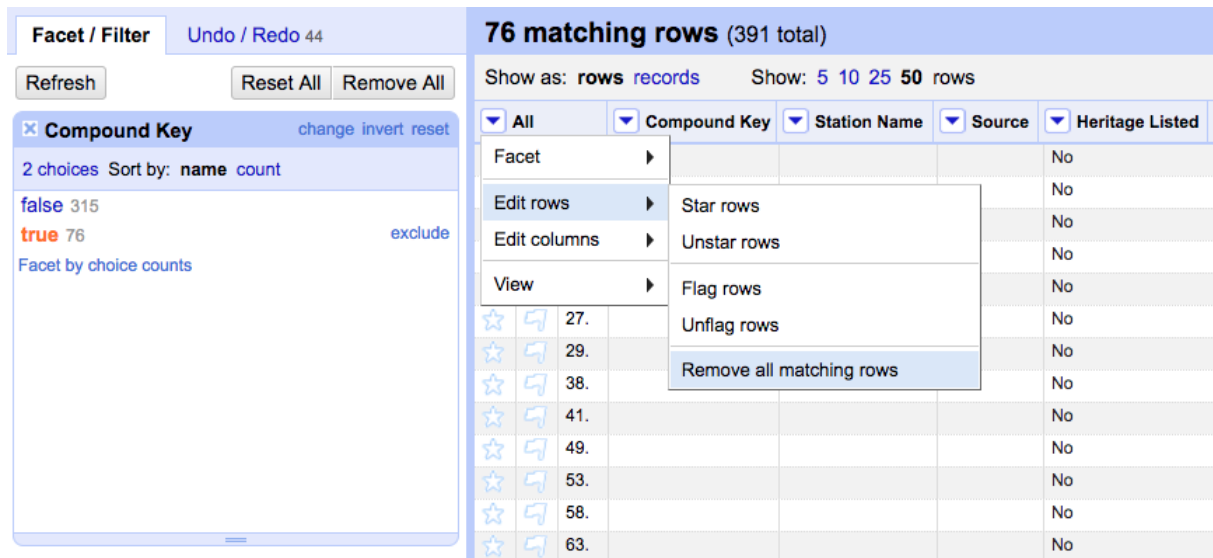
1. While still in **Records** view, select **Heritage Listed menu > Facet > Text Facet**.

2. Select **Yes** from the **Heritage Listed facet** menu to filter. **Note that facets and filters behave differently in *Records* mode than in *Rows* mode**. When set to display records, a filter will show the complete record when any one row in that record matches the filter. In this case, we have multiple rows referring to the same police station, but only some rows are correctly identified as being heritage listed.

3. With the **Heritage Listed filter** still active, select **Heritage Listed > Edit Cells > Transform…**.

4. Set **Expression** to:

   ```
   "Yes"
   ```

5. Click **OK** to set all Heritage Listed values to yes for any listed row shown. Note that this will only be applied to the records included within the filter.

6. Select **Remove All** to remove all active facets and filters.

Removing duplicate rows

We are now finally ready to remove the duplicate rows.

1. Switch to a **Rows** view.

2. Select **Compound Key > Facet > Customized Facet > Facet by blank**. Select **true** from the Facet menu.

3. Select **All > Edit rows > Remove all matching rows**. We have now removed all the duplicated rows we have found in our data set.

4. Select **Remove All** to remove all active facets and filters.

Normalising the Compound Key column

The Compound Key column we created to identify duplicates now contains multiple types of information. We need to re-split this back into separate columns.

1. Select **Compound key menu > Edit columns > Split into several columns**.

2. Replace the comma character with " | " and select **OK**.

3. Rename the resulting columns as *Address*, *Suburb* and *LGA* respectively.

8. Supplementing the data by calling into an API

It's relatively straightforward to draw in data from an external API with OpenRefine. In this case we'll call in to Google's Geolocation API to get the longitude and latitude of all our police stations.

1. In **Rows** view, ensure all rows are displayed by selecting **Reset All**.

2. Select **Suburb menu > Edit column > Add column by fetching URLs**.

3. Type `Geocoding Response` into **New column name**.

4. Type `50` in **Throttle delay**.

5. Type the following exactly into the **Expression** box:

   `"https://maps.googleapis.com/maps/api/geocode/json?sensor=false`
   `&address="+escape(value + ", New South Wales", "url")`

The language used to query the Google Geocoding Application Programmer Interface (API) is found here: https://developers.google.com/maps/documentation/geocoding/ .

These state that the format of the request to this API must be:
http://maps.googleapis.com/maps/api/geocode/output?parameters and the minimum parameters required are "address" (the address that you want to get geo-co-ordinates for) and "sensor" (which indicates whether or not the geocoding request comes from a device with a location sensor. This value must be either true or false). Parameters are separated by the ampersand (&) symbol.

Our Expression:

`"https://maps.googleapis.com/maps/api/geocode/json?sensor=false&address="+escape(value + ", New South Wales", "url")`

Meaning:

This is telling your computer to visit ***https://maps.googleapis.com/maps/api/geocode/*** and get output in JSON (JavaScript Object Notation) language, from a query to the API. We are telling it that the API query request does not come from a device where there is a location sensor (it's just coming from a list of text terms), and that the query address to use is the value found in the suburb column, and limit the results to those that only include "New South Wales" somewhere in the information compiled by Google.

The *escape* and *url* refer to GREL string functions where we can basically invoke a function and then 'escape out' in url mode. See https://github.com/OpenRefine/OpenRefine/wiki/GREL-String-Functions#escapestring-s-string-mode.

---

6. Click **OK**. OpenRefine will query the API for each row in the dataset.

7. Select **Geocoding Response menu > Edit column > Move column to end**.

8. Select **Geocoding Response menu > Edit column > Add column based on this column**.

9. Type *Lat* into **New column name**.

10. Type *parseJson(value).results[0].geometry.location.lat* into **Value**.

The results from this query for every row in the table come back from calling the Google Geocoding API with all the geolocation information compiled by Google. We asked for it to be in JSON format which looks like this on your screen in the OpenRefine table (This is the info for "Bellata"):

```
{ "results" : [ { "address_components" : [ { "long_name" : "Bellata",
"short_name" : "Bellata", "types" : [ "locality", "political" ] }, { "long_name"
:    "New    South    Wales",    "short_name"    :    "NSW",    "types"    :    [
"administrative_area_level_1", "political" ] }, { "long_name" : "Australia",
"short_name" : "AU", "types" : [ "country", "political" ] }, { "long_name" :
"2397",    "short_name"    :    "2397",    "types"    :    [    "postal_code"    ]    }    ],
"formatted_address" : "Bellata NSW 2397, Australia", "geometry" : { "bounds" : {
"northeast" : { "lat" : -29.77446510, "lng" : 150.07246010 }, "southwest" : {
"lat"   :   -30.066420,   "lng"   :   149.47932990   }   },   "location"   :   {   "lat"   :
-29.91820250,    "lng"    :    149.79118540    },    "location_type"    :    "APPROXIMATE",
"viewport" : { "northeast" : { "lat" : -29.77446510, "lng" : 150.07246010 },
"southwest" : { "lat" : -30.066420, "lng" : 149.47932990 } } }, "types" : [
"locality", "political" ] } ], "status" : "OK" }
```

The information in JSON format is hierarchical (or nested) and can be displayed in this way:

```
{

   "results" : [

      {

         "address_components" : [

            {

               "long_name" : "Bellata",

               "short_name" : "Bellata",

               "types" : [ "locality", "political" ]

            },

            {

               "long_name" : "New South Wales",
```

```
            "short_name" : "NSW",

            "types" : [ "administrative_area_level_1", "political" ]

        },

        {

            "long_name" : "Australia",

            "short_name" : "AU",

            "types" : [ "country", "political" ]

        },

        {

            "long_name" : "2397",

            "short_name" : "2397",

            "types" : [ "postal_code" ]

        }

    ],

    "formatted_address" : "Bellata NSW 2397, Australia",

    "geometry" : {

        "bounds" : {

            "northeast" : {

                "lat" : -29.77446510,

                "lng" : 150.07246010

            },

            "southwest" : {

                "lat" : -30.066420,

                "lng" : 149.47932990

            }

        },

        "location" : {

            "lat" : -29.91820250,

            "lng" : 149.79118540

        },
```

```
        "location_type" : "APPROXIMATE",

        "viewport" : {

          "northeast" : {

            "lat" : -29.77446510,

            "lng" : 150.07246010

          },

          "southwest" : {

            "lat" : -30.066420,

            "lng" : 149.47932990

          }

        }

      },

      "partial_match" : true,

      "types" : [ "locality", "political" ]

    }

  ],

  "status" : "OK"

}
```

Note you can call the Google API for each row individually by doing a similar query but including a specific place name (the example here is just for Bellata):

http://maps.googleapis.com/maps/api/geocode/json?sensor=false&address="Bellata", New South Wales", "url".

There is a lot of information returned from Google and in this example we want to filter out just the "Latitude" value.

Expression: *parseJson(value).results[0].geometry.location.lat*

Meaning: Parse the JSON results in each cell of the table to show the latitude (lat) value, which is nested under the location information, which is nested under the geometry information. If more than one result is returned (e.g. if there are 3 "Richmond"s in NSW, we're only interested in getting the data from the first (denoted by [0]) result that was retrieved.

11. Click **OK**.

12. Select **Geocoding Response menu > Edit column > Add column** based on this column.

13. Type *Long* into **New column name**.

14. Type *parseJson(value).results[0].geometry.location.lng* into **Value**.

As above, now we're just filtering out the Longitude Value Google has on each place:

Expression: `parseJson(value).results[0].geometry.location.lng`

Meaning: Parse the JSON results in each cell of the table to only show the longitude (lng) value, which is nested under the location information, which is nested under the geometry information. Again, we're only interested in getting the data from the first ([0]) result that was retrieved.

---

15. Click **OK**.

16. Select **Geocoding Response menu > Edit column > Remove this column**.

Scatterplot Facet

We now have the geospatial lat/long coordinates of the suburb of every police station in the dataset. We can use the scatterplot facet to hone in on a subset of these:

1. Select **Long menu > Facet > Scatterplot Facet**.

2. Click the highlighted facet area. The scatterplot facet will appear on the left-hand side. Notice it looks uncannily like a map of NSW.

3. Use click-and-drag to select an area of NSW (say the Sydney Basin). Observe the subset of results you get. Note that the **Suburb** and **Address** facets are narrowed down to the matching area.

4. Click **Reset All** to display the entire set of rows.

9. Undo/Redo History

OpenRefine has an infinite undo history. To access:

1. Click on the **Undo/Redo** tab. You'll see every action you've done since creating the project.

2. You can undo to any step by clicking on the step you want to revert back to. Similarly, you can redo every step.

If you wish to save a set of steps to be re-applied later, or to a different project, you can click the **Extract** button. This gives you the option to select the steps you want to save, and to copy the transformations included in the selected steps in a format called 'JSON'

To apply a set of steps you have copied or saved in this 'JSON' format use the **Apply** button and paste in the JSON. In this way you can share transformations between projects and each other.

10. Exporting the dataset

1. Click **Export** in the top right-hand corner.

2. Select **Comma-separated variable** from the drop-down menu. A CSV dump of your data will be downloaded.