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 Open Refine

Dr Luc Small & Dr Jeff Christiansen | 09 August 2013 | 1.4

1 Overview

Open Refine is a powerful free tool for exploring, normalising and cleaning up datasets. In this tutorial we'll work through the various features of Refine, including importing data, faceting, clustering, and calling into remote APIs, by working on a fictional but plausible humanities research project. We'll start with a research question in mind and use the features of Refine 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 the kinds of archival records State Records NSW holds on them.

2 Resources

Open Refine:

- http://openrefine.org/

Open Refine Documentation:

- http://openrefine.org/OpenRefine/documentation
- https://github.com/OpenRefine/OpenRefine/wiki/Documentation-For-Users

The Google Geolocation API:

- http://code.google.com/apis/gears/api_geolocation.html

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

- http://www.heritage.nsw.gov.au/07_subnav_04.cfm (then Basic Search for "police station")

The NSW State Records API:

- http://search.records.nsw.gov.au/

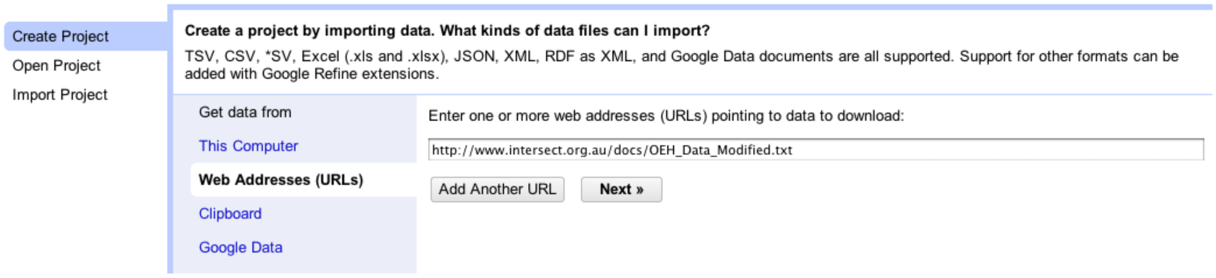
3 Installing Open Refine

To install Open Refine:

1. Go to the main Open Refine 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 Open Refine. It will open in your default web browser. Note: Open Refine 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 >>**.



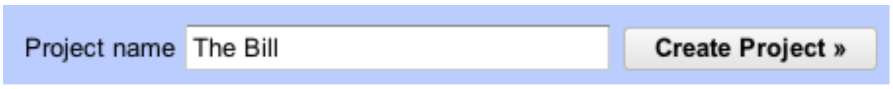
Create Project
Open Project
Import Project

Create a project by importing data. What kinds of data files can I import?
TSV, CSV, *SV, Excel (.xls and .xlsx), JSON, XML, RDF as XML, and Google Data documents are all supported. Support for other formats can be added with Google Refine extensions.

Get data from
This Computer
Web Addresses (URLs)
Clipboard
Google Data

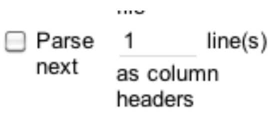
Enter one or more web addresses (URLs) pointing to data to download:
http://www.intersect.org.au/docs/OEH_Data_Modified.txt

6. On the resultant screen, set **Project name** to `The Bill`:



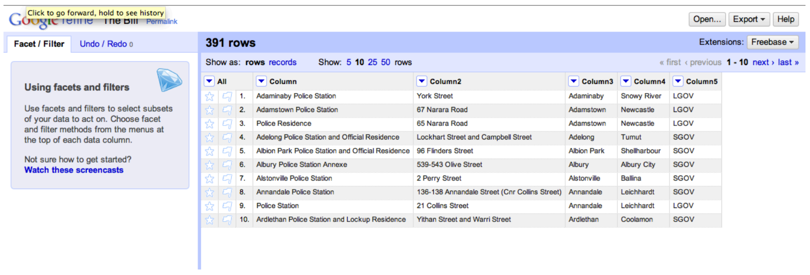
Project name: The Bill

7. Untick **Parse next _1___ line(s) as column headers**:



☐ Parse next 1 line(s) as column headers

8. Click **Create Project >>**.
9. The project will open with 391 rows:



Click to go forward, hold to see history
Google refine The Bill Permalink
Open...
Export
Help
Facet / Filter
Undo / Redo 0
Extensions: Freebase

Using facets and filters

Use facets and filters to select subsets of your data to act on. Choose facet and filter methods from the menus at the top of each data column.

Not sure how to get started?
Watch these screencasts

391 rows

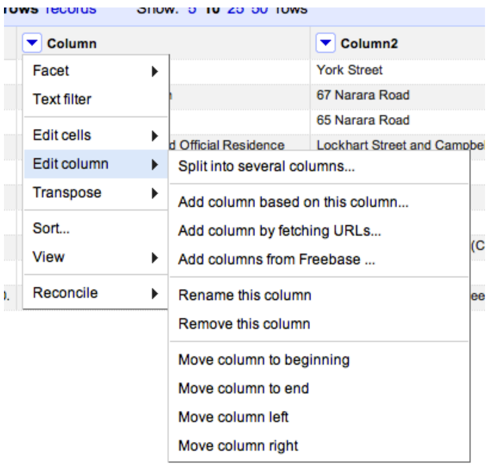
Show as: rows records   Show: 5 10 25 50 rows   « first ‹ previous 1 - 10 next › last »

All	Column	Column2	Column3	Column4	Column5
1.	Adaminaby Police Station	York Street	Adaminaby	Snowy River	LGOV
2.	Adamstown Police Station	67 Narara Road	Adamstown	Newcastle	LGOV
3.	Police Residence	65 Narara Road	Adamstown	Newcastle	LGOV
4.	Adelong Police Station and Official Residence	Lockhart Street and Campbell Street	Adelong	Tumut	SGOV
5.	Albion Park Police Station and Official Residence	96 Flinders Street	Albion Park	Shellharbour	SGOV
6.	Albury Police Station Annexe	539-543 Olive Street	Albury	Albury City	SGOV
7.	Alstonville Police Station	2 Perry Street	Alstonville	Ballina	SGOV
8.	Annandale Police Station	136-138 Annandale Street (Cnr Collins Street)	Annandale	Leichhardt	SGOV
9.	Police Station	21 Collins Street	Annandale	Leichhardt	LGOV
10.	Ardlethan Police Station and Lockup Residence	Yithan Street and Warri Street	Ardlethan	Coolamon	SGOV

5 Getting Organised

Renaming columns

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.

Splitting columns

1. Select **Column5 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")
```

Open Refine 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 column 5. Column 5 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 Column 5 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 **Column5 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, "")`***

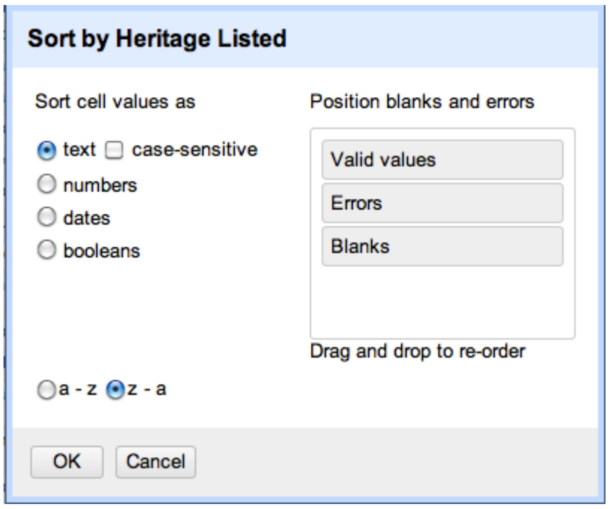
Meaning: for every row**, if the value of the cell in column 5 does not equal "Yes" then in the new column, insert a value in the new column of "" (i.e. whatever the value was that appeared in column 5)**

7. Select **Column5 menu > Edit column > Remove this column**

6 Exploring the data

Sorting Columns

1. Select **Heritage Listed menu > Sort**
2. Select **text** and **z-a** and click **OK**. All the heritage-listed police stations will appear first.



Sort by Heritage Listed

Sort cell values as: [✓ text] ☐ case-sensitive, numbers, dates, booleans

Position blanks and errors: Valid values, Errors, Blanks

Drag and drop to re-order

a - z [✓ z - a]

Facet and cluster on Suburbs

1. 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.
2. Click on **265 choices**. A text box will appear so we can copy and paste our list of unique suburbs into, say, a document.
3. Click **count** to order to list the most frequently occurring suburbs first.
4. 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.
5. 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.

Find duplicate addresses

1. Take the same approach as above to create a text facet on the **Address** column.
2. Sort by **count**.
3. You can click on any given address to view only the records matching the address. For instance, click on `281 Clarence Street`.
4. Click on **Reset All** to restore the listing to display all 391 rows.

7 Undo/Redo History

Open Refine 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.

8 Calling into an API

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

1. Ensure all 391 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:

```
"http://maps.googleapis.com/maps/api/geocode/json?sensor=false&addres
s="+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:

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

Meaning:

This is telling your computer to visit ***http://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**. Open Refine 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 Open Refine 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 391 records.

9 Supplementing the data by calling into another API

Searching the State Records NSW Archives

1. Select **Suburb menu > Edit column > Add column by fetching URLs**.
2. Type `Search Response` into **New column name**.
3. Type `200` in **Throttle delay**.
4. Type the following exactly into the **Expression** box:

```
"http://search.records.nsw.gov.au/search?entities=Agency&q="+escape(v
alue +" Police Station", "url")
```

Now we are querying the State Records Office of NSW using their API. More info about it is available here: http://search.records.nsw.gov.au/usage.

Expression:

```
"http://search.records.nsw.gov.au/search?entities=Agency&q="+escape(v
alue +" Police Station", "url")
```

Meaning:

The syntax has some similarities to the Google API call we used previously.

This is telling your computer, for every row, to visit **http://search.records.nsw.gov.au and search all data from the State Records for Agencies (i.e. administrative or business units which have responsibility for carrying out some designated activity (which includes Police Stations)), and to use a query term of "the value in the Suburb column Police Station".**

Once again, 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. *+* refers to a space, an illegal character in urls.

What is returned for each row with this query that used the value for the Suburb, and the terms "Police" and "Station" is a html formatted list of all the Police Stations in NSW where there is a record in the State Records Office that contain any of the 3 search terms, and they are ranked in order of similarity to the search terms.

You can see the same thing, neatly formatted in a web page for "Adaminaby Police Station" if you put this into your web browser:

http://search.records.nsw.gov.au/search?entities=Agency&q=adaminaby+police+station

5. Click **OK**. Open Refine will query the API for each row in the dataset.
6. Select **Search Response menu > Edit column > Move column to end**.
7. Select **Search Response menu > Edit column > Add column based on this column**.
8. Type `State Records Title` into **New column name**.
9. Type `value.parseHtml().select("#content table tr td div:eq(0) a")[0].htmlText()` in **Expression**. Click **OK**.

Now we want to parse the html results for each row to only list the name of the police station that appears first in the results list.

The relevant part of the html which shows the first two lines of the search results in the results table looks something like this:

```
<table>
 <tr>
  <td width="20%" class="searchlist">
   <div class="collection_search_result">
    <p>
     <img src="/resources/images/agencies.png;jsessionid=02A41D8315656564F3DAF7CAAD8489A1" alt="agencies"/>
     <a href="/agencies/118;jsessionid=02A41D8315656564F3DAF7CAAD8489A1">
       Adaminaby Police Station
     </a>
    </p>
   </div>
   <div class="collection_search_result">
    <p>
     <img src="/resources/images/agencies.png;jsessionid=02A41D8315656564F3DAF7CAAD8489A1" alt="agencies"/>
     <a href="/agencies/966;jsessionid=02A41D8315656564F3DAF7CAAD8489A1">
       Nimmitabel Police Station
     </a>
    </p>
   </div>
```

Expression:

```
value.parseHtml().select("#content table tr td div:eq(0) a")[0].
htmlText()
```

Meaning:

Parse the html formatted table of results to select the htmlText value of "a" in the first row of the results.

tr, **td**, **div** and **(0)** refer to the html **table row**, **table cell**, **table divider**, and the **1st divider.** **()** denotes calling a function (the select function in this case). When using the select function, even if there is only 1 result, we need to use **[0]** to specify the first result.

10. Select **Search Response menu > Edit column > Add column based on this column**.
11. Type `Agency URL` into **New column name**.
12. Type `value.parseHtml().select("#content table tr td div:eq(0) a")[0].htmlAttr("href").split(";")[0]` into Expression. Click **OK**.

We also want to parse the html results for each row just to give us the State Records Agency number for the Police Station record that appears first in the results list

Expression:

```
value.parseHtml().select("#content table tr td div:eq(0) a")[0].
htmlAttr("href").split(";")[0]
```

Meaning:

Parse the html formatted table of results to get the **value of "a"** in the **first row** of the results, then **find the html attribute "href", and take everything in this value prior to the ";".**

i.e. for each row in the table, get me the NSW State Records ID for each Police Station. This is indicated by the **href** value of **a for the first row** (i.e. **take everything prior to the ;** in the URL).

Again, **tr**, **td**, **div** and **(0)** refer to the html **table row**, **table cell**, **table divider**, **and the 1st divider. ()** denotes calling a function (the **select** function in this case). When using the **select** function, even if there is only 1 result, we need to use **[0]** to specify the first result.

13. Select **Search Response menu > Edit column > Remove this column**.

Digging Deeper

14. Select **Agency URL menu > Edit column > Add column by fetching URLs**.
15. Type *`Agency Response`* in **New column name**.
16. Set **Throttle Delay** to *`50`*.
17. Type *`"http://search.records.nsw.gov.au" + value + ".xml"`* in **Expression**. Click **OK**.

Expression: `"http://search.records.nsw.gov.au" + value + ".xml"`

Meaning: go to http://search.records.nsw.gov.au and append the value from the Agency URL column and then append .xml

i.e. This formulates a URL that points to an xml file for every row:

The Adaminaby example is **http://search.records.nsw.gov.au/agencies/118.xml** and this returns a xml file with various attributes about the Police Station:

```
<agency version="1.0">
 <id>118</id>
 <title>Adaminaby Police Station</title>
 <administrativeHistoryNote/>
 <abolition/>
 <category>Police Station</category>
 <creation/>
 <endDate/>
 <endDateQualifier/>
 <startDate>1889-01-01 00:00:00.0</startDate>
 <startDateQualifier>by</startDateQualifier>
 <lastAmendmentDate>2005-07-21 00:00:00.0</lastAmendmentDate>
</agency>
```

18. Select **Agency Response menu > Add column based on this column**.
19. Type `History Note` in **New column name.**
20. Type `value.parseHtml().select("administrativehistorynote")[0].htmlText()` in **Expression**. Click **OK**.

Expression:

```
value.parseHtml().select("administrativehistorynote")[0].htmlText()
```

Meaning: **Parse the XML information** just returned and just give me the **value of the Administrative History Note** (there is no value for this in the Adaminaby example) as **text**.

Note that there is no parseXml function in GREL so we use parseHtml. Again, when using the **select** function, even if there is only 1 result, we need to use **[0]** to specify the first result.

21. Select **Agency Response menu > Add column based on this column**.
22. Type `Start Year` in **New column name**.
23. Type `value.parseHtml().select("startDate")[0].htmlText().substring(0,4)` in **Expression**. Click **OK**.

Expression:

```
value.parseHtml().select("startDate")[0].htmlText().substring(0,4)
```

Meaning: Give me the date that the Police Station was established (in the Adaminaby example it's 1889). **Parse the XML information** just returned and **select** the **1st 4 characters (start at the *0* value and take *4* characters)** of the value of the **start Date** as **text.**

Again, when using the **select** function, even if there is only 1 result, we need to use **[0]** to specify the first result.

24. Select **Agency Response menu > Add column based on this column**.
25. Type `End Year` in **New column name**.
26. Type `value.parseHtml().select("endDate")[0].htmlText().substring(0,4)` in **Expression**. Click **OK**.

Expression

```
value.parseHtml().select("endDate")[0].htmlText().substring(0,4)
```

Meaning: Give the date the Police Station was closed

27. Select **Agency Response menu > Edit column > Remove this column**.

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.