Software skills for librarians

Module 2: Open Refine
Reference material

Creating a project

To use open refine you must first create a project based on the data file you wish to edit. You can import data in the following formats: TSV, CSV, Excel, XML, JSON and Google Spreadsheets. OpenRefine will try to select the format based on the file extension and the contents, and then prompt you to select options depending on the format.

Once the data has been imported, you will be presented with a tabular display, like a spreadsheet. However, there are two crucial differences: OpenRefine does not modify your original data file; instead all changes are saved with the project, allowing for infinite 'undo' and 'redo' operations. Secondly, the entire file does not necessarily need to be held in memory, making it easier to handle large data sets (up to hundreds of thousands of rows).

Manual editing

To make changes to a single cell, place the mouse pointer over that cell and click the small edit button which appears. You can change the value in the window which appears, while there is also an option to apply the same change to all cells in the selected column which have the same initial value. You can also set the data type (text, number or date) of the cell.

Common global changes

There are a number of common operations which work on the entire data set. These can easily be accessed from the menu at the top of each column (click on the downward pointing arrow). These are:

- **Reordering columns**. Selected from the first column, labelled 'all', it allows you to change the order of columns, also to rename and remove them.
- **Sorting**. Selected from the top of the column you wish to sort on, you can select options like normal and reverse order from the pop-up window. It works like sorting in a spreadsheet, except that the sort only affects the display of data; your original file remains unchanged.
- **Faceting**. This is a useful tool for to gain an overview of your data set. It works best with columns which can take only a limited range of values from a controlled vocabulary (like MARC language codes, degree courses or students' colleges). Select facet from the top of the column you're interested in and you will see a list of possible values; click on one of these to select it. You can also make global changes to a heading in this way. The available types of facet include text, numeric, date and custom.
- **Filtering**. This works like faceting, except that it limits your view of the data to those rows which contain a specified word or phrase in a particular column. It works well with free text fields, like titles, where faceting cannot easily be applied. You can also filter using regular expressions.
- **Clustering**. This can be used with data which exhibits small variations, such as names which may contain differences in spelling and punctuation. It attemps to automatically group similar values into clusters, and allows you to make these consistent. There are different methods of clustering which may be selected according to the range of variation in your data. You can also select clustering from within a text facet.

Custom facets

If the availabe default facets are insufficient, you can create your own. This is particularly useful if you have data split across several columns, or you only need part of a column. Esentially you enter an expression in GREL which works on the data to produce a result which is related to the input values in some way. This will be clearer once you have a good grasp of GREL expressions. Some possibilities are:

- Boolean facets. Create a facet with two values, for example: those items which do and do not contain a particular word. `value.contains("XML")`
- Substrings. Create a facet on part of a string, for example: extracting institutions from e-mail addresses. `value.match(/@(\w+)\./)[0]`
- Multiple columns. Create a facet which depends on data from several columns. `join([ cells["col1"].value, cells["col2"].value ], " ")`
- Mathematical functions. With numeric data, transform the values before faceting, for example: convert data to a linear scale. `log(value)`

Transformations

Much of the power of of OpenRefine comes from the ability to apply transformations to make changes to data. The concept is similar to formulae in a spreadsheet, but the language is more flexible allowing changes to be made completely automatically. Transformations can be applied from the menu at the top of a column and by entering a GREL expression.

All GREL expressions will operate on one of the following predefeined variables:

`value` the data in the current cell.

`cell` as above but containing extra fields. The notation cell.value is a synonym for value (as above).

`cells` the cells in the current row. You can use `cells.colname` to get a single cell object, or, if the column names have spaces in, `cells["column name"]`

`row` as above but containing additional fields. The notation row.cells is the same as the cells variable (above)

GREL differentiates between the following data types: text, numbers, dates, arrays and boolean (true or false) values. The last two are not encountered directly in your data, but may be produced as the result of an expression. For example, the split method operates on a string to produce an array of substrings.

String literals can be written in inverted commas, like `"Some text"`. Arrays can be written as a comma separated list of values enclosed in square brackets. Functions or methods can be applied to your data in one of two ways:

`split(value,",")`, or
`value.split(",")`

Functions on strings

`contains(`*s, sub*`)`
Returns true if the string contains the substring sub.

`toLowercase(`*s*`)`
Converts a string to lowercase.

`toUppercase(`*s*`)`
Converts a string to uppercase.

`substring(`*s, from, to*`)`
Returns the substring taken from s, between character positions specified by from and to, inclusive.

`indexOf(`*s, sub*`)`
Returns the location of the substring sub.

`replace(`*s, f, r*`)`
Search the string s for substring f and replace it with r.

`match(`*s, regexp*`)`
Matches regexp against the string and returns an array containing all the captured groups.

`split(`*s, sep*`)`
Returns an array containing the individual fields of s separated by sep.

Functions on arrays

`slice(`*a, from, to*`)`
: Returns a section of the array a, between the elements given by from and to inclusive.

`reverse(`*a*`)`
: Reverses the order of elements in the array a.

`sort(`*a*`)`
: Sorts the array a into order.

`join(`*a, sep*`)`
: Returns a string containing all the elements of a joined together with sep between them.

Functions on dates

`toString(`*d, format*`)`
: Converts a date to a string using the specified format.

`diff(`*d1, d2, timeUnit*`)`
: Returns the difference between two dates. If timeUnit is not specified it defaults to hours.

`datePart(`*d, unit*`)`
: Returns a single field from a date. The unit parameter is a string like `"day"` specifying which field.