# OpenRefine for Health Sciences: Part 2

## Learning Objectives

1. Apply transformations to strings, numbers, dates and Booleans to programmatically edit data and prepare data for analysis.
2. Write expressions using General Refine Expression Language (GREL)

## Logistics/Housekeeping

Welcome to Part 2 of the OpenRefine for Health Sciences. This is part 2 of a series focused on OpenRefine, today’s session is Transformations. This series of classes is based on material from The Carpentries which is known for live coding. If you were able to download and install OpenRefine on your device, we ask that you try to follow along if possible with the live coding.

## Reopening OpenRefine Project

Hopefully everyone was able to download and install OpenRefine. I am using version 3.9.3 on Windows. If you weren’t able to install OpenRefine, that’s okay! We encourage you to follow along as best you can and take notes of things to try on your own. The slide deck we shared has some key points we are covering about different features and tasks you can do in OpenRefine.

Now I’m going to launch OpenRefine. From the **blue diamond** shortcut.

This will open in a web browser, but remember it's not actually connected to the internet. It just uses the web browser as the interface. *NOTE: If OpenRefine does not open in a browser window, open your browser and type the address* [*127.0.0.1:3333*](https://127.0.0.1:3333/) *to take you to the OpenRefine interface.*

### Steps for Reopening Previous Project

If you were with us for Part 1 of this class, you have already created a project in OpenRefine. You can open a past project by clicking **Open project**, where you’ll see a list of your past projects on this device. I also mentioned at the end of last week that you can export your entire OpenRefine project as a file and under **Import project** you can browse and open a project file, OR open a project from a URL. This is really helpful if you want to share projects with collaborators. I can open my project from last week, and I’ll show you if you were with us last week:

*Select OpenRefinePart1\_NNLM\_2025-09-09*

You can see the same dataset and our history of steps under Undo/Redo. But I also want to show folks who maybe weren’t with us last week how to create a new project file. So to return to that initial menu, I click the OpenRefine logo which is the blue diamond icon. And then **Create project**. You hopefully received the link and were able to download the dataset we are using today which is the same dataset we saved at the end of last week’s class. Again, we click **Browse**, and then select the dataset file wherever it is saved on your computer. Click **Open** and **Next**.

## Introducing Transformations

*Based on* [*Library Carpentry: OpenRefine Lesson 7*](https://librarycarpentry.github.io/lc-open-refine/instructor/07-introduction-to-transformations.html)

Through facets, filters and clusters OpenRefine offers relatively straightforward ways of getting an overview of your data, and making changes where you want to standardize terms used to a common set of values.

However, sometimes there will be changes you want to make to the data that cannot be achieved in this way. Such types of changes include:

* Splitting data that is in a single column into multiple columns (e.g. splitting an address into multiple parts)
* Standardizing the format of data in a column without changing the values (e.g. removing punctuation or standardizing a date format)
* Extracting a particular type of data from a longer text string (e.g. finding ISBNs in a bibliographic citation)

To support this type of activity OpenRefine supports ‘Transformations’ which are ways of manipulating data in columns. Transformations are normally written in a special language called “GREL” (General Refine Expression Language). To some extent GREL expressions are similar to Excel Formula, although they tend to focus on text manipulations rather than numeric functions.

Full documentation for the GREL is available in [OpenRefine Help Documentation](https://docs.openrefine.org/manual/grelfunctions). This tutorial covers only a small subset of the commands available.

## Common Transformations

Some transformations are used regularly and are accessible directly through drop-down menu options, without having to type them directly.

Examples of some of these common transformations are given in the table below, with their GREL equivalents. We’ll see how to use the GREL version later in this lesson.

|  |  |  |
| --- | --- | --- |
| **Common Transformation** | **Action** | **GREL expression** |
| Trim leading and trailing whitespace | Removes any ‘whitespace’ characters (e.g. spaces, tabs) from the start and end of the current value | value.trim() |
| Collapse consecutive whitespace | Removes consecutive whitespace between words/characters | value.replace(/[\p{Zs}\s]+/,' ') |
| To titlecase | Converts the current value to titlecase (i.e. each word starts with an uppercase character and all other characters are converted to lowercase) | value.toTitlecase() |
| To uppercase | Converts the current value to uppercase | value.toUppercase() |
| To lowercase | Converts the current value to lowercase | value.toLowercase() |

#### Exercise #1: Transform Language column

1. Create a text facet on the Language column
   1. Note that in the values there are some entries with multiple entries – why do these two values appear separately rather than as a single value?
   2. Point out that some of the dual language entries have a space between them
2. On the Language column use the dropdown menu to select Edit cells 🡪 Common transforms 🡪 Collapse consecutive whitespace
3. Look at the text facet again for the Language column now – has it changed? (if it hasn’t changed try clicking the Refresh option to make sure it updates)

#### Practice Question #1 Remove whitespace from Publication Type column

1. How many cells have extra white space?

**Solution Steps**

* Create a Text facet on the **Publication Type** column
* On the Publication Type column use the dropdown menu to select Edit cells 🡪 Common transforms 🡪 Collapse consecutive whitespace
  + 1,316 cells were transformed

#### Practice Question #2 Transform to Titlecase

1. Use the common transform option “To titlecase” to change all the values in the Title column to Title Case.
2. How many columns were transformed?

**Solution Steps**

* On the Title column use the dropdown menu to select Edit cells 🡪 Common transforms 🡪 To titlecase
  + 3,176 cells were transformed

These are a couple examples using **Common Transformations**, now we’ll look at writing more complex, specific transformations.

### Key Points

* Common transformations are available through the drop-down menu option

## Writing Transformations

*Based on* [*Library Carpentry: OpenRefine Lesson 8*](https://librarycarpentry.github.io/lc-open-refine/instructor/08-writing-transformations.html)

Writing transformations in OpenRefine allows you to make complicated changes to your data, including editing values, removing characters, or extracting pieces of a value to form a new column.

GREL functions are written by giving a value of some kind (a text string, a date, a number etc.) to a GREL function. Some GREL functions take additional parameters or options which control how the function works.

There are two ways to write GREL expressions, with value.function(and in parentheses any augments) or function – for example like toTitlecase, toDate, split, join, or replace – followed by parentheses with value (which is the original value), followed by the new value or format etc. You can see the two ways of writing the syntax on this slide:

* value.function(options)
* function(value, options)

Either one is valid, and which is used is completely down to personal preference. Today we’re going to use the first syntax.

Let’s look at the **Keyword Heading** column. If we want to change these values to Title Case, we could do the **Common Transformation** as Javier showed us, or we can write our own transformation. To start writing transformations, select the column on which you wish to perform a transformation and choose Edit cells 🡪 Transform….

In the screen that displays you have a place to write a transformation and then the ability to preview the effect the transformation would have on 10 rows of your data.

The transformation you type into the ‘**Expression’** box has to be a valid GREL expression. The default expression is the word value by itself – which means the value that is currently in the column – that is: make no change.

Next to the ‘**Preview’** tab are tabs to view:

* **History**: a list of transformations you’ve previously used with the option to reuse them immediately or to “star” them for easy access
* **Starred**: a list of transformations you’ve “starred” via the History view
* **Help**: a list of all the GREL functions and brief information on how to use them

Now to transform this column, we type value.toTitlecase() in the Expression box and click **OK**. That transformed 882 cells in the Keyword Heading column.

#### Exercise #2: Put MESH Subject Headings into Title Case

Now let’s look at another example transforming the **MESH Subject Headings** column to Title Case. Click the drop-down menu for the **MESH Subject Headings** column, and select Edit cells 🡪 Transform…

In the **Expression** box type value.toTitlecase() – we can already see a **Preview** in the box below what effect this will have. You’ll notice that not all subject heading words are being capitalized. This is because **toTitlecase** only capitalizes the first letter when it is the first character, but the vertical bar or separator pipe (|) character means the computer is interpreting the first word after the pipe as part of the previous word. We can remove the vertical bar and add a space between each word.

To do this, type value.replace("|", ", ") and click **OK**. This now ***replaces*** the vertical bar with a comma as the separator and adds a space. Now we can see the expression recognizes and capitalizes the first character for the first word in a **Subject Heading**. However, subject headings that start with the star character **(\*)** are still lowercase. This had me a little stumped, but I looked at the **Help** tab to see if I could understand more about the toTitlecase expression.

*Click* ***Help*** *tab, scroll down to* ***toTitlecase****.*

In the **Help** for **toTitlecase**, I can see the two arguments that the expression requires – the “**string s**” which is our original value – and the “**string delimiters (optional)**”.

Leaving these parentheses empty allows the default to take place, but we need to write in our delimiters as part of the expression.

Now I’m going to use the second way of writing transformations, type toTitlecase(value, ", \*"). Remember we replaced the **|** so now our delimiter is the **comma** and **space**. I can add the **\*** character as part of the delimiter and now in the **Preview**, we see that all words are capitalized, even those with the \*. We click **OK**. This transformed 2,774 cells, and when we scroll down we see Subject Headings are capitalized but wait – some headings are still not capitalized because they are in brackets. This is our next Practice Question.

#### Practice Question #3 Transform MESH Subject Headings in brackets to Title Case

1. Write a transformation to transform MESH Subject Headings written in [brackets] to Title Case.
2. How many cells are transformed?

**Solution Steps**

* On the **MESH Subject Headings** column use the dropdown menu to select Edit cells 🡪 Transform…
* Type in the Expression box: toTitlecase(value, ", \*[")
  + *We include the both the \* and [ characters to transform both \*starred entries and [bracketed] entries*
* Click **OK**
* 2,897 cells are transformed

We can see this transformed additional cells in the **MESH Subject Heading** column, before it was 2,774 cells and using the bracket as part of the delimiter transformed 2,897 cells.

#### Exercise #3 Facet by Database

Let’s do another exercise together now, looking at how transformations can extract part of a value from a cell, for example if we wanted to shorten the **Database** names. First, we look at the facet list for our **Database** column.

1. Select Ovid MEDLINE(R). Let’s say that we consider that too long, and just want it to read “Ovid.”
2. Choose Edit cells 🡪 Transform…
3. In the Expression box type value.get(0,4)
   1. OpenRefine reads from left to right, starting with position “0”. In OpenRefine GREL, “get” is a function that will return characters in a string based on their position in the string, starting from “0”
4. What does the **Preview** pane under the Expression box show? Click OK
5. Close the **Facet/Filter** box and do a new **Text facet**. What changes does it show?

### **Key Points**

* You can alter data in OpenRefine based on specific instructions
* You can preview the results of your GREL expression
* You can expand the data editing functions that are built-in into OpenRefine by building your own transformation

## Data Types

*Based on* [*Library Carpentry: OpenRefine Lesson 10*](https://librarycarpentry.github.io/lc-open-refine/instructor/10-data-transformation.html)

Understanding data types can help you write a wider variety of transformations using GREL.

Every piece of data in OpenRefine has a ‘type’. The most common ‘type’ is a ‘string’ – that is a piece of text. However, there are other data types available and transformations let you convert data from one type to another where appropriate. The data types supported are:

* String
* Boolean
* Number
* Date
* Array (which we’re not going to talk about)

### Booleans

A “Boolean” is a binary value that can either be “true” or “false”. Boolean values can be used directly in OpenRefine cells, but are more often used in transformations as part of a GREL expression. For example, the GREL expression: value.contains("test") generates a Boolean value of either TRUE or FALSE depending on whether the current value in the cell contains the text “test” anywhere.

Such tests can be combined with other GREL expressions to create more complex transformations. For example, the GREL transformation if(value.contains("test"), “test”, value) can be used to change a specific phrase or word in a cell value and replace it with something else.

Facets are intended to group together common values and OpenRefine limits the number of values allowed in a single facet to ensure the software does not perform slowly or run out of memory. If you create a facet where there are many unique values (for example, a facet on our DOI or article title column in a dataset that has one row per article) the facet created will be very large and may either slow down the application, or OpenRefine will not create the facet.

Now you may run into an issue here, where OpenRefine tells you “choices total, too many to display”. This happens when there are too many unique entries in a column and it would bog down OpenRefine too much. Rather than pushing through and potentially crashing, like what excel does, it will display this warning. So there are too many for us to facet.

Instead, let’s try a BOOLEAN to find what we are looking for. On the **MESH Subject Headings** column drop-down menu, select Edit column 🡪 Add column based on this column. Using this function, you can create a new column while preserving the old column. This opens a similar box to the **Expression** box we saw before when writing our transformations. The first thing we enter is the **New column name** which I’m naming Anesthesia.

And in the Expression box, type: if(value.contains("Anesthesia"), "Include", “Remove”))

Make sure you don’t have any extra spaces or periods in the phrase. This means that if the cell has the value “Anesthesia” in it – where Anesthesia is TRUE, it will be labeled as “Include” in the new column. If Anesthesia is not present or FALSE, the value will become “Remove”. Click **OK**. All 3,189 rows are modified and we now have a new column where we can more easily facet for what articles to include or remove from our dataset.

Now try this on your own, using the if and contains functions to create a new column for “Pediatric”

#### Practice Question #4 Use Boolean to add a new column

1. Use a Boolean to find how many rows include the MeSH Subject Heading ‘Pediatric’. How many articles contain Pediatric in the MeSH Subject Headings?

**Solution Steps**

* On the **MeSH Subject Headings** column, Edit column 🡪 Add column based on this column
* New column name: Pediatric
* if(value.contains("Pediatric"), "Include", “Remove”))
* Click **OK**
* Facet 🡪 Text facet
  + 1160 articles contain "Pediatric"

## Why Are Dates Messy?

Before we get into working with dates, let’s talk a little about why they can be so messy. Take a moment and type in the chat – what’s today’s date?

## Working with Dates

So far, we’ve been looking only at “string” type data. Much of the time it is possible to treat numbers and dates as strings. For example, in the **Date Accessed** column we have dates that may look date formatted, but OpenRefine considers these dates to be represented as a string. However, some operations and transformations only work on what OpenRefine considers “number” or “date” typed data, such as sorting values in numeric or date order. To carry out these functions we need to convert the values to a date or number first.

### Specifying Date Formatting in GREL Expressions

GREL allows us to specify date and time using pattern strings, which are letters that have some specific representation in the function call.

Pattern strings are case sensitive, therefore capital and lower-case letters have a different meaning and usage. The table below shows letters related to date and time representation.

| **Letter** | **Date or Time Representation** |
| --- | --- |
| y | Year |
| M | Month in year |
| D | Day in year |
| d | Day in month |
| F | Day of week in month |
| E | Day name in week |
| u | Day number of week |
| a | AM/PM marker |

The table below presents examples on how to use the patterns as input and the obtained output.

|  |  |
| --- | --- |
| **Date and Time Pattern Input** | **Output** |
| "yyyy-MM-dd" | 2025-04-05 |
| "dd MMM yyyy" | 05 Apr 2025 |
| "EEE, MMM d, ''yy" | Sat, Apr 5, ’25 |
| "yyyy.MMMM.dd hh:mm a" | 2025.April.05 12:10 PM |
| "EEE, d MMM yyyy HH:mm:ss" | Sat, 5 Apr 2025 12:10:10 |

For a more detailed explanation, review the [OpenRefine Documentation](https://docs.openrefine.org/manual/grelfunctions#date-functions).

### Reformat a Column of Dates

Make sure you remove all Facets and Filters so we’re working with our entire dataset again. On our two date columns, we’ll use the Edit cells 🡪 Common transforms 🡪 To date again. Note how the values are now displayed in green and follow a standard convention for their display format (ISO 8601) – this indicates they are now stored as “date” data types in OpenRefine.

When we do this on the **Date Published** column, you’ll see that only 517 cells were transformed and as we scroll, we still see many non-date data types in this column. This is because the toDate function defaults to the ISO 8601 format and that’s what we saw in the **Date Accessed** column. We will have to manually transform the different date formats represented in the Date Published column.

1. First, we do Facet 🡪 Text facet to see how many ways the date is written.
2. On the **Date Published** column use the dropdown menu to select Edit cells 🡪 Transform…
3. In the ‘Expression’ box type the GREL expression value.toDate("yyyy”, “yyyy MMMM”, “MMMM yyyy”)
   1. This will look for date formats where there is just the year, there is Year and Month, or Month and Year, and convert all of these to the same standardized date format.
4. Click **OK**
5. This transformed 2,247 rows which is the majority, and probably the best we can do using these mass transformations.

### Timeline Facet

*Based on* [*Library Carpentry: OpenRefine Lesson 4*](https://librarycarpentry.github.io/lc-open-refine/instructor/04-faceting-and-filtering.html#other-types-of-facet)

Not that the majority of our values in this column are recognized as dates, we can carry out functions that are specific to dates, and we can do the **Timeline Facet**.

To look at the **Timeline Facet**, click on **Date Published** column drop-down menu, and select Facet 🡪 Timeline facet. Now on the right, we can look at a timeline of our publication dates. You can use the sliding scales at either end to limit which articles you are looking at.

We can also do our **Timeline Facet** on the **Date Accessed**, and you can see different trends in each. Let’s say I want to look at only articles published between 2016 and 2019. I have about 1,246 rows. Now give this a try yourself.

#### Practice Question #5 Timeline Facet on the Date columns

1. How many articles were accessed between 2022 and 2023?
2. How many articles were published between 2003 and 2013?

**Solution Steps**

1. How many articles were accessed between 2022 and 2023?
   * On the **Date Accessed** column, Facet 🡪 Timeline facet
   * Limit dates to Jan 01 2022 and Dec 16 2023
     + 1,546 articles access between 2022 and 2023

\***Note**: Make sure to remove or reset the **Timeline Facet** from one column before faceting another column.

1. How many articles were published between 2003 and 2013?
   * On the **Date Published** column, Facet 🡪 Timeline facet
   * Limit dates to Dec 25 2002 and Dec 15 2013
     + 1,645 articles published between 2002 and 2013

You can add additional facets on top of the timeline. Add a Text facet to the **Publisher** column.

* How many publishers are listed between 2003 and 2013? 164 different publishers
* What about between 2010 and 2013? 144 different publishers

#### Exercise #4 Creating a column based on Another Column

Let’s say the year published is actually the most important factor on if we want to include an article in our dataset and we want to be able to more easily facet to just the years in a particular timeframe. We can use transformations to create a new column that is just the year the article was published.

On the **Date Published** column drop-down menu, select Edit column 🡪 Add column based on this column. Using this function, you can create a new column while preserving the old column. This opens a similar box to the **Expression** box we saw before when writing our transformations. The first thing we enter is the **New column name** which I’m labeling **YearPublished**. And we can use another transformation we looked at today, the get function: type value.get(0,4) because the year is always the first four digits in our date data type.

This transformed all 3,189 rows, and now we have a new column. You’ll notice it’s not in green - these are not considered dates by OpenRefine. But now we can use our **Text facet** again on the **YearPublished** column, and we have a much cleaner list of years to look at in our facet list. However, this didn’t perfectly convert everything to a year as some of our entries were not converted to dates before. But for these, I think the easiest thing would be to manually enter the year of publication, based on the information in the **Date Published** and **Source** columns.

Using this Facet list, we can include or exclude articles/rows based on the Year Published. I could include just articles from 2015 to 2020, by clicking “include” beside each of these years.

### Converting Data Types: From Date to String

And lastly, we’ll work through one more example together converting from the Date data type back to the String data type. We can create a new column feature to once again modify our **Date Accessed** column and make the dates more human-readable. On the **Date Accessed** column drop-down menu, select Edit column 🡪 Add column based on this column.

For New column name, enter “FormattedDateAccessed”. In the ‘Expression’ box type the GREL expression: value.toString("dd MMMM yyyy")

We now see the date expressed in a more human-readable format, with a two-digit date, the name of the month spelled out, and the four-digit year.

## Export (Final Step)

*Based on* [*Library Carpentry: OpenRefine Lesson 12*](https://librarycarpentry.github.io/lc-open-refine/instructor/12-export-transformation.html)

### **Exporting Data**

Now we have finished working with our data set for today. And it’s time to save and export. The export options are accessed through the **Export** button at the top right of the OpenRefine interface.

Export formats support include HTML, Excel and comma- and tab-separated value (csv and tsv). You can also write a custom export, selecting to export specific fields, adding a header or footer and specifying the exact format.

*Click* ***Export*** *and select the file format for saving the dataset.*