# Week 1

### **Installing OpenRefine on Windows**

* [Official instructions](https://github.com/OpenRefine/OpenRefine/wiki/Installation-Instructions#windows)

To install on Windows, [download the zip file](https://github.com/OpenRefine/OpenRefine/releases/download/3.2/openrefine-win-3.2.zip), browse to the downloaded zip file, then [extract the files](https://support.microsoft.com/en-us/help/14200/windows-compress-uncompress-zip-files) to a new folder placed anywhere you like on your computer (Downloads, Desktop, etc.).

Check to make sure that it works by double clicking on the OpenRefine icon in your Applications folder. You should see a command window open, followed by tab in your default web browser running the application. If you don't, try visiting <http://127.0.0.1:3333/>, which is the web address you can use to visit OpenRefine any time the application is running.

If the application installs, but no browser window opens, make sure that you have a [Java Runtime Environment (JRE)](https://github.com/OpenRefine/OpenRefine/wiki/Setup-JAVA) installed. This the number one problem you're likely to run into. Once you're sure that's installed and running, try closing your browser and reopening OpenRefine.

## **Week 1 Excel File**

Use this file for the exercises and assignment this week.

Click [Week1Excel.xlsx](https://www.libraryjuiceacademy.com/moodle/pluginfile.php/64974/mod_resource/content/1/Week1Excel.xlsx) link to view the file.

### **Understanding the Basics of OpenRefine**

Some basic things to know about what works well and what doesn’t work well in OpenRefine:

* It’s not a spreadsheet or a database, though it has aspects of each. Don’t try to use it for data entry. Add your data in Excel or another program first, then import.
* You can view items as rows or records. If you see results that don’t look right, check to make sure you have the one selected that you think you should have (almost always rows).
* You get infinite undos!
* Don’t try to do anything with huge datasets. Break them into smaller sections if possible.

You might want to watch this introduction before getting started to get a sense of what you can do.

<https://youtu.be/B70J_H_zAWM>

### **Starting your first project by importing an Excel file**

Download the Week 1 Excel File. You will use this throughout this exercise, and hopefully by the end of the exercise you will be ready to submit your completed assignment.

1. Click on Create Project tab from the opening OpenRefine screen (this may already be selected).
2. In the "Get data from" menu you will see that "This computer" is selected. Next to that is a "Choose file" button. Click this, and browse to the file you just downloaded. Once it has loaded, click Next >>.
3. The program will spend a moment importing the data. Once it has finished, you may rename it in the "Project Name" field (not necessary). Everything else should be defaults. Make sure the header includes the correct information, and if it doesn't, make sure you are including the first line as column headers (click on check box next to parse line 1 as column headers).
4. Click on Create Project.

### **Understanding the Interface**

Spend some time familiarizing yourself with the OpenRefine interface before you jump in to any cleanup. We'll go over the most important pieces of it right now, but if you have additional questions, please ask on the discussion board.

Note that you can rename your project at any time by clicking on the title next to the OpenRefine icon. You can also copy a link to the Permalink, which you can bookmark to get to immediately as long as OpenRefine is running.

A screenshot of a cell phone

Description automatically generated

*Screenshot of top OpenRefine project, top left corner, showing project name and permalink*

### **Facet/Filter and Undo Tabs**

A screenshot of a cell phone

Description automatically generated

Facet/Filter is the major workspace you'll be using in your time working with OpenRefine.

Undo allows you to perform infinite undos on your work, as well as to copy and paste the JSON (way of describing data) that describes your project (Click on the Extract... button for this), which you'll need to do when you turn in assignments. You can use this tab to copy and paste in other JSON snippets that you find online or have saved to perform some functions automatically (Click on Apply... for this).

### **Open and Export options**

A screenshot of a cell phone

Description automatically generated

Open... lets you open a new project.

Export allows you to export your project, either the entire project, or a version of your data such as Excel. You'll do this at the end when you turn in your completed Excel file.

*Open project: Using OpenRefine,*Chapter 1: Diving Into OpenRefine, Recipe 2 - Creating a new project

*Export project: Using OpenRefine,*Chapter 1: Diving Into OpenRefine, Recipe 6 - Exporting a project

### **Dropdown Menus**

Using OpenRefine, Chapter 1: Diving Into OpenRefine, Recipe 3 - exploring your data

Take a look at all the dropdown menus carefully. The first column has different options than the rest of them.

The All column includes options that will affect an entire row or allow you to rearrange columns.

The rest of the columns we'll go through in more detail in the following lessons.

### **Working with Data**

Let's get to work on cleaning up your name and address data. Spend a few moments looking through the dataset to see if you can spot any obvious problems. (Click on "Show 50" to see everything at once).

Sometimes you know what the errors are, sometimes you don't. This dataset has some obvious inconsistencies and errors as would be normal in data entry, as well as some invisible errors that wouldn't cause a problem until you tried to, for instance, copy it into another system.

Let's look at an example of that now.

### **Built in cleanup tools**

Using OpenRefine, Chapter 2: Analyzing and Fixing Data, Recipe 5

Click on the dropdown menu in first\_name, then Edit Cells, then Common Transforms. Select Trim leading and trailing whitespaces. You should see the following message at the top of your screen:

A close up of a logo

Description automatically generated

This tells you that OpenRefine ran a function called "Trim" on the first name column that removed extra whitespace that would be invisible to you, but could cause huge problems if you were trying to run additional processing on this file.

Run through the rest of the columns with this same function to make sure you've removed all other spaces.

### **Facets**

Using OpenRefine, Chapter 2, Analyzing and Fixing Data, Recipe 2 - faceting data

Facets are the magic of OpenRefine. They allow you to focus in on matching or nearly matching data, and ensure that everything is consistent, among other things.

Click on the dropdown menu for the "State," column, then Facet, then Text Facet. You'll see a box in the Facet/Filter tab on the left of the screen with 24 choices. Sort them by count. Scroll down to the items with only one count. These are always good to check first, because they are items with only one instance, meaning that they might be items with typos and errors.

Find "OY". This clearly isn't a state, but there are several states it could be, so you can't just edit it. Click on it, and your dataset will limit to just the row that includes this. This should give you enough clues to know what it should really be. Hover over OY in your Facet menu, and click on Edit, and change it to the correct state. You'll note that after you do that your focused data will change to include all the states with that same abbreviation.

There's another error state. See if you can figure out which one it is in your Facet menu, and edit it.

### **Filters**

Using OpenRefine, Chapter 2 Analyzing and Fixing Data, Recipe 4 - applying a text filter

Filters allow you to search across an entire column for a certain piece of text, or can be used within Facets to search within a smaller part of your data.

A few of the names in first\_name include titles, which you don't want to include in your final dataset. You can find these more easily by clicking on first\_name, then Text Filter. In your Filter menu box, type in the letter M, and then click on the Case Sensitive box. You should find two names with titles. To get rid of those, hover over a box with data you want to change, and then click on Edit. Delete the title and press Apply.

SNEAK PREVIEW: Click on Undo and then go back one step so you have both the extra titles back. Now copy this into your Text Filter box: **M[a-z].\s**, and click on Regular Expression. We'll start looking at regular expressions more in coming weeks, but this is a preview of their power. Breaking it down:

M = the letter M

[a-z] = any lower case letter

. = a period

\s a space character, meaning there's a space rather than any other characters in a word.

### **Bringing it All Together**

Use facets, filters, and built in transformations to clean up the rest of your data. Once done, you'll be ready to turn it in to complete your weekly assignment. Some tips:

* Check to make sure email addresses and websites will work.
* Make sure all the abbreviations are consistent. If only one item in a column is abbreviated, unabbreviate it, and vice versa.
* Make sure phone numbers are consistently formatted, and that there aren't any obvious errors in area codes.

# Week 2

## **Starting a project by importing a text file or copying and pasting text**

### **What is different from an Excel file?**

An Excel file has data that's already structured, with columns and rows. You'll notice that when you import a file you are told that OpenRefine supports "TSV, CSV, \*SV, Excel (.xls and .xlsx), JSON, XML, RDF as XML, and Google Data". Those are all structured data formats. TSV and CSV files are text, but they contain tabs or commas to differentiate columns, so are easy for OpenRefine to process.

To import a file without any structure, or with an unpredictable structure, we need to take a slightly different approach. Libraries often have to do something with a list of citations which we really would rather have in a spreadsheet. Rather than copying and pasting manually into columns, we can use OpenRefine to create structure out of those citations.

### **What is different from an Excel file?**

An Excel file has data that's already structured, with columns and rows. You'll notice that when you import a file you are told that OpenRefine supports "TSV, CSV, \*SV, Excel (.xls and .xlsx), JSON, XML, RDF as XML, and Google Data". Those are all structured data formats. TSV and CSV files are text, but they contain tabs or commas to differentiate columns, so are easy for OpenRefine to process.

To import a file without any structure, or with an unpredictable structure, we need to take a slightly different approach. Libraries often have to do something with a list of citations which we really would rather have in a spreadsheet. Rather than copying and pasting manually into columns, we can use OpenRefine to create structure out of those citations.

## **Starting a project by importing a text file or copying and pasting text**

### **Getting your text file ready**

Before you attempt to put your file into OpenRefine, I suggest you take a close look at it in a text editor to fix any obvious errors ahead of time. Sometimes you won't be able to find those errors until you have it imported, so you may need to come back to this after you find them.

### **Text Editors**

A good text editor is going to be a key tool for you to use as you work with OpenRefine. I personally like [Notepad++](https://notepad-plus-plus.org/) for Windows, and [Brackets](http://brackets.io/) for Apple. The main thing you'll want to have is an ability to see spaces and paragraph breaks in your text, since that will visually indicate to you if there are some extra invisible characters that are causing problems. Advanced find and replace using regular expressions will also come in handy once you're more comfortable with using those.

### **Pre-editing the File**

For this week's exercise, you're going to use a CV or resume, either your own or someone you know. My CV is available for download [here](https://docs.google.com/document/d/1Hptd7AlYcPf2nVT1U6lmrxrx_GOJiQnHpOggJwrlXRk/edit?usp=sharing), which I'll use as an example. You may want to follow along with the same text. I'm going to copy and paste all the text I want to include into Brackets.

For this exercise, I'll focus on the "Presentations" section only.

If I paste this text into OpenRefine using the Clipboard option to start a project, I'll get this. Note that there is a blank row every place there was a blank line in the Word document.

A screenshot of a cell phone

Description automatically generated

To get rid of those, I can unclick "Store blank rows", and those will be removed. If I had any non-blank rows that I didn't want to import, or data errors, it is easier to fix in a text editor. So in my text editor I removed all the data I didn't want, fixed a few things, and tried again, with the store blank rows box unchecked.

A screenshot of a cell phone

Description automatically generated

This looks much easier to deal with, so I'll go ahead and click on Create Project>>.

### **Try it Yourself**

Find a segment of a CV or resume to turn into structured data. It could be your own, or a friend's, or mine. Just make sure there are enough items on it that you can get at least 5-10 items to include.

First copy it into a text editor, and see if there are any obvious errors. Fix those. Then open OpenRefine, click on Create Project, then Clipboard. Import your text, and experiment with the check boxes to see how your data changes (click on Update Preview if nothing happens). Once you're satisfied, create your project.

## **Introduction to regular expressions**

### **Attempt: 1**

### **What are regular expressions?**

Using OpenRefine pp. 107-118

You already used a regular expression last week, and while it may have looked unusual or even scary, you need to know how to use regular expressions to get the full power of OpenRefine. Basically you use regular expressions to stand in for text patterns, the same way that you would with regular find/replace, but with a lot more power. They are used in programming to do things like check to make sure an email address is properly formed.

I'm not going to go into a lot of detail--we'll just go over a few different patterns this week that are particularly relevant to the work we'll be doing, so I urge you to spend some time after the class learning more about regular expressions. It's a good idea to print out a cheat sheet and leave that at your desk when you're trying to construct a new one. Some good resources are:

* [Fear No Longer Regular Expressions (Bohyun Kim)](http://acrl.ala.org/techconnect/post/fear-no-longer-regular-expressions)
* [RegEx Tutorials](http://regextutorials.com/)

### **Regular Expressions for characters and words**

Let's start with the basics, characters and words.

* **[d]** means find a lowercase "d".
* **[a-z]** means find any lowercase alphabetical character.
* **[D]** means find any uppercase "D".
* **[A-Z]** means find any uppercase alphabetical character.
* **[a-zA-Z]** means find any lowercase or uppercase alphabetical character.

You normally will enclose units of your search in square brackets, but there are also a number of shortcuts you can use.

* **\w** means any word. This means a series of alphabetical characters or underscores, separated by spaces. We'll look at more words soon.

### **Digits**

* **25** means the number "25".
* **\d** means any digit character.
* **2\d** means any number that begins with 2 and then contains any digit.

It's helpful in particular with digits to know about the curly braces, which allow you to specify the number of digits that should exist in your search (you can use these with characters as well). There are more to these, but for now to keep it simple, let's leave it at this:

* **\d{4}** means match any number with four digits. This is super helpful for finding years.

### **Additional operators**

There are other operators that are useful to know, though they are definitely going to require more practice to really understand than we'll go over in this class.

* **\b** means a word boundary.
* **\bLa** will match "Labs" but not "Collaborative".
* \* matches the thing you just typed zero or more times.
* **+**matches what you just typed one or more times.
* **?**matches what you just typed zero or one time.
* **[d]\***will match anything with or without a d.
* **[d]+** will match something that has at least one d.

We'll look more at look-ahead and look behind next week, but you might want to look at [this page](http://www.rexegg.com/regex-lookarounds.html) to get a peek at it.

### **Finishing up Regular Expressions**

We'll use regular expressions more in the coming lessons, but let's take a look at an example from my CV. I wasn't always consistent in labeling the years. In some cases, I included the date in parentheses like (2012), but in other cases I listed it after a comma. How can I find all these times?

Think about it...

**, [\d]{4}**will find a comma, followed by a four digits. I can use this to identify the line with an error and update it.

## **Introduction to Google Refine Expression Language (GREL)**

### **What is GREL?**

Using OpenRefine pp. 115-118

Google Refine (the old name for OpenRefine) Expression Language, or GREL, is a simple programming language similar to Javascript that works inside OpenRefine. It allows you to do more complex and custom transformations on your data than you can with the built in regular expressions and transformations.

We will cover the basics of GREL in this class, but you can get the GREL reference anywhere when you are transforming you data by clicking on the help tab. You can also look on the OpenRefine wiki for some sample syntax, though this generally is written for people with strong computer science skills and takes some unpacking.

### **GREL Syntax**

The basic format for constructing GREL functions is:

* Function(arguments)

You can use "value" to stand in for the cell you're running the function on, which is easier to follow.

* Example value.replace(“dog”, “cat”)

**value** = the current cell

**.replace() =**the replace function

**"dog"** = the word (aka string) to find in the cell

**"cat"** = the string to replace it with

You could have also written this **replace(value,"dog","cat")**

Some additional functions that we'll need to use this week are:

* **split("fire, water, earth, air", ",")** becomes "fire", "water", "earth" , and "air“. You construct this as **value.split("THING YOU WANT TO SPLIT ON")**. Note that this doesn't do anything else with the cells, it just splits them into an array (a list) that you can do something else with. You usually want to combine this with something like join.
* **join([ "magic", "capes", "swords" ], ";")** becomes magic;capes;swords. Note that the items inside the square brackets are an array that you can create with the split function, and then join with the join function. I.e. join([value.split("//")], ";")

Let's try an example. In the Presentations section of my CV, I have my name listed as "G" or "G.". I'd like to spell it out as Greer. How would I do that?

The replace function will work for this. To use this function, I will click on the column with names (the first column), then Edit Cells, then Transform.

A screenshot of a cell phone

Description automatically generated

In the box, I get the word "value", and then a preview screen below. This is really helpful, since you can use the preview to show you if you're getting it right or not *before* you press OK.

There are a few options for what I could enter. The simplest is this:

**value.replace("Martin, G","Martin, Greer")**

This simply says any place you see "Martin, G" replace with "Martin, Greer".

A screenshot of a cell phone

Description automatically generated

You'll note that I still have a period after "Greer." Why? Because it's not included in the string--the part in the quotes. If I wanted to change the period to a comma, I could include the period in the "value" string and a comma in the "replace" string. Since the period was left outside of the expression altogether, it is ignored completely, and therefore it remains.

Let's say that I have my first name initial elsewhere in the CV, and it's not always preceded by "Martin,". Sometimes it's "G. Martin." To change **all** occurrences of my first name initial to my full first name, I can use a regular expressions inside of GREL to do a more complex find/replace operation.

The following GREL find/replace expression will first look for a capital "G" and then look ahead to make sure it is not followed by a letter. This means we are making sure that we don't replace any capital G with "Greer" (yikes!).

**value.replace(/([G][.]?)(?a-z])/,"Greer")**

Let's break this down:

* To use regular expressions in GREL, you must enclose them in //
* () indicates groups.
* [M] indicates match an uppercase G.
* [.]? means optionally match a period if it follows G.
* (?![a-z]) includes lookahead operators. This is a sneak preview for later. This means "check ahead to see if there are any more letters after this. If so, don't match this". This means that it won't match anything that just starts with an uppercase G.

This is great... but note that it will change any other names with "G" initials to Greer. I could go back and fix those manually. This is a common occurrence, a batch edit operation such as this will edit data that you don't necessarily want to edit. So, you need to decide if it is less work to automate edits, go back and fix these issues later, versus making all of the edits manually. In this example, I decide that it is less work to use GREL to change my initial than it is to change all the different versions of my initial manually.

### **Separating all the columns**

Using OpenRefine pp. 83-84

In my example, I used the built in "split columns by separator" function. This requires you to identify a common data element which you can use to divide columns. Earlier, the first thing I did was to separate the date into its own column using **(** (results pictured below) and then **).**

![A screenshot of a cell phone

Description automatically generated

You can use regular expressions in this box as well, and it's probably the go-to function. But there is one important caveat. You lose that element in your data. That doesn't always matter. For instance, in the above example, I have lost the open parentheses "(". So, I'll want to remove the close parentheses as well.

However, there is an issue! If I split the data using parenthesis, it will split every parenthesis, including those in the presentation description, such as in Row 3, "(GLUG)." This might be edited manually later.

Another way to split out segments of the presentations into individual columns is to split on periods. The citation style that I have use separates segments by period, so this would work nicely. But, there 's a problem: there will be a column split inside of the name segment, because middle initials are followed by a period. As we already know, this will not only split the data across two columns, but it will remove the period, and we don't want either of these things to happen. So what do I do?

Here's where we need those look-ahead and look-behind operators again. Last time I used it to check ahead for any letters. This time I'm going to check behind with the following function:

**value.split(/(?<A-Z])[.]/)**

Let's break this one down.

* Again, use // to show you're using a regular expression.
* () show one group that you're assessing. You have to use these with look-ahead and look-behind operators.
* ?<![A-Z] says "when you get past this section, check back to see if there's an uppercase letter. If so, don't use it."
* [.] matches a period. But in this case, *only* a period that doesn't follow an uppercase letter.

As you recall, the split function doesn't split anything, it only creates an array of splits. You can use the join function to join your splits with a character that you're sure doesn't exist elsewhere in your data. I've chosen "%".

**join(value.split(/(?<![A-Z])[.]/),"%")**

Now I can use the built in split columns by separator function to split on "% ". Here's my final result:

![A screenshot of a social media post

Description automatically generated

Note that it doesn't always go perfectly. You'll note that the close quotes for the presentation titles have moved over to the conference column. It would have been wise to delete these ahead of time, since I know that the titles will be in their own column. But, this is pretty easy to fix manually. Also, Row 9 did not split, because the period follows a capital letter ("Getting to know the ArchivesSpace API."). And, Row 10 didn't split because the presentation title ends with an exclamation point. I can edit these manually too. Finally, I'll need to run the trim leading and ending whitespaces built-in transform as well.

Last step is to rename the columns into something that makes sense. Here's my final product:

A screenshot of a social media post

Description automatically generated

## **Introduction to Google Refine Expression Language (GREL)**

### **Working on your own CV**

Use the same techniques on a segment of your own CV.

1. Identify the data you want in the final version, which elements separate them, and which you need to keep and which get rid of.
2. Plan your actions: visualize the final product and work backwards.
3. Run trim leading and trailing whitespaces.
4. Fix any obvious data errors.
5. Rename the columns.

### **More clustering and filtering**

Now we’ve gotten this split up, we can look at some things you might want to do. For instance, what about seeing how many presentations given per year? I can do this by using the timeline filter on the Date column. But first, we'll need to reformat the dates, which are currently formatted as text and written as "[Month, YYYY]", as dates.

Although there is a reformatting shortcut (Edit Cell > Common Transformations > To date), this won't work because the data is not numeric. First we'll have to transform the written-out months into numbers. The steps are:

1. Separate month and year into separate columns (Edit Column > Split Into Several Columns, use "," separator).
2. Rename the new columns "Month" and "Year."
3. Transform month names into numbers: Edit cells > Transform... and write in expression: value.replace("January", "01").replace("February", "02").replace("March", "03").replace("April", "04").replace("May", "05").replace("June", "06").replace("July", "07").replace("August", "08").replace("September", "09").replace("October", "10").replace("November", "11").replace("December", "12")
4. Combine month and year columns\*: On Year column, select Edit cells > Transform... and write in expression: value+"-"+cells['Month'].value
5. Now the months and years are combined into one column. Now you can select Edit cells > Common transforms > To date.

(\*The column names might change as you add and join column data, so you may need to change the column name back to "Month" and "Year".)

The dates should have turned green and have time information appended. Now you can filter as dates.

Click on the timeline facet, and facet to 2013-4.

A screenshot of a cell phone

Description automatically generated

Now, we can add another filter. Let's filter by text in the Title column. If I type in "metadata," titles without "metadata" will be filtered out.

A screenshot of a social media post

Description automatically generated

In general, clustering and filtering can help you to identify patterns, and then make small adjustments within those patterns. If you have a large dataset, this allows you to focus on just a small subset of data. Try this on your own data to see what patterns emerge.

### **Clustering**

Using OpenRefine pp. 104-108

The button we haven’t yet discussed in the Facet menu is a little button labeled Cluster. This is another OpenRefine magic trick. This will find pieces of your data that are really close and probably the same, and allow you to do editing en masse.

Here's an example using a different set of data. If I do a text facet on "Journal," and then click on Cluster, the built-in algorithms will "cluster" identical text strings. This essentially will give you a list of unique text strings. This is really useful for identifying typos. In evaluating the list, you may find some false matches, but in this case I find an error within the middle of a word. I can click on the Merge box to fix that.

A screenshot of a cell phone

Description automatically generated

In the above example, the two unique text strings will merge together, meaning that their values will become whatever is written in the "New Cell Value" box. Merge in this case means to merge the clusters, not merge the cells.

Now go over some columns that look like they contain duplicate data values. Can you find any typos or outliers using the Text Facet "Cluster" option?

# Week 3

Week 3 API

What is an API?

Introduction

The intention of the lessons this week is to give you an introduction to how to use APIs in general if that is is new to you, and how to use one specifically for data you might use in library work. The [SHERPA/RoMEO](http://www.sherpa.ac.uk/romeo/index.php) API is reasonably easy to use, but the advantage to OpenRefine is that you don't have to worry about a lot of the mechanics of dealing with data. OpenRefine does it for you.

[SHERPA/RoMEO API information](http://www.sherpa.ac.uk/romeo/apimanual.php?la=en&fIDnum=|&mode=simple)

### What is an API?

API Stands for Application Programming (or Programmers’) Interface, and is a way to access data in a structured format so that it can be used for other purposes. Common uses for APIs in libraries are to present a custom front end for an OPAC or ILS, move data between your ILS and another system, such as a self checkout machine. They are available for many services we use for storing or obtaining data about our services. For instance, [Springshare provides APIs](https://buzz.springshare.com/producthighlights/libguidescms-as-website/features/search-widgets) for their products, which allow you to create widgets on your website for LibGuides or LibCal. [OCLC has many APIs](https://www.oclc.org/developer/develop/web-services.en.html), one of which we’ll learning about next week. Often you can obtain this data by manually searching a database, but with APIs you can automate database lookup. This can change hours of work into minutes.

[List of Library APIs](http://www.programmableweb.com/category/library%2Breference/apis?category=20272%2C20066)

## is an API?

### How do you use an API?

The service programmers should have provided documentation for how you you need to query (or "call") the API. Usually this is in the form of a special URL. You may have seen an OAI base URL before, and this is basically what we are talking about here. It looks just like a URL that you would use to get to a website, but if you put it into a browser you would see XML or data in some other form (often JSON). What you do with that data is up to you, which can be very frustrating if you aren't a programmer, so this is where tools like OPenRefine come in.

To use an API in OpenRefine, first identify the column containing the data we want to search for. Click on Edit Column, then, "Create new column by fetching URL.” This will allow you to enter all the parameters for your API call.

Important things to note here include  the throttle section. You need to be respectful of the load you are putting on the server. Doing all your queries without a break can overload the server, so the throttle allows you to separate them. Sometimes the API will have a requirement about this, but in general try to err on the side of caution if you have a lot of queries to make. You may need to start your query, go for coffee, and come back.

Other thing is is the preview window. Just as with transformations, this shows what OR will do with what you typed in. This is especially helpful for using an API, because you can copy and and paste a sample URL into a browser and see if you get back what you expect.

We'll get into what you actually need to type into this box in a bit.

### Getting an API Key

Before we go any further, we need to stop to talk about API keys. These are usually either required by the service provider or recommended. They allow the service to control who has access to the data, as well as communicate with those users who rely on the service. A example of why this matters: one day the library hours disappeared from our library website. It turned out that this functionality relied on a version of the Google Calendar API that was no longer supported. We didn't realize this because it was possible to use it without an API key, so we didn't get a notification when it went away. So even if there is free API, always register if you rely on the service.

Note! API keys are passwords. So please don't post them anywhere publicly. If you share code, always remove your API key before you post it. You can put in "API KEY HERE" or similar.

Before you go on to the next lesson, [please register for a free SHERPA/RoMEO key](http://www.sherpa.ac.uk/romeo/apiregistry.php).

### Using the SHERPA/RoMEO API

For those who aren't familiar, SHERPA/RoMEO is a database of permissions for archiving open access versions of journal publications. So if you wanted to self-archive your articles in an open access repository (or someone else's articles), you could search by journal title to learn of its policies. By using the SHERPA/RoMEO API, you can automate that process. Here's how you'd do it.

Create a new project in OpenRefine using the week3 - Editable.xlsx spreadsheet.

First, we know we want to check the Journal column. So we select that, then click on “Add column by fetching URLs…”.

Name the column SHERPA, and update the throttle delay to 500 millisecond to start with.

The expression you want to use is as follows: **'http://www.sherpa.ac.uk/romeo/api29.php?ak=[APIKEY]&qtype=starts&jtitle=' + escape(value,'url')'**.

Let's break this down:

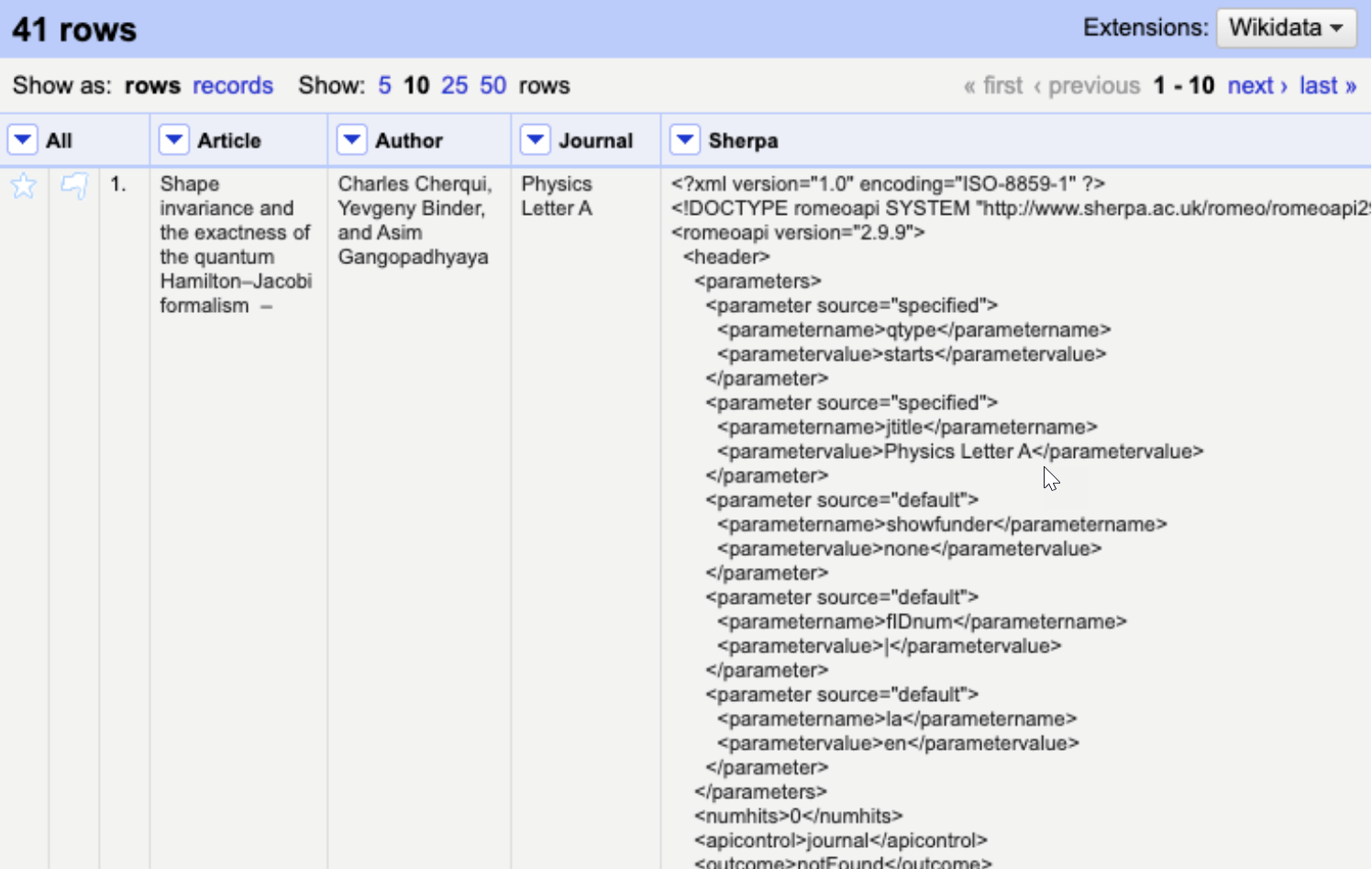
* Enclose your entire URL in **''**.
* **http://www.sherpa.ac.uk/romeo/api29.php** is the base URL. This is the site that will understand the query.
* **?** is the way you say "I'm starting my query now."
* **ak=[APIKEY]** is where you put your API key. Don't actually enclose it in square brackets.
* **&** is the way you say "I'm adding more to my query."
* **qtype=starts**means you want to search for something that starts with this text.
* **jtitle=**is the journal title. The quote after this is showing that you need to add the dynamic part of the query.
* **+**indicates you're adding an OpenRefine function.
* **escape(value,'url')**means that you want to add in the value from the cell, but you want to "escape" the characters for a URL.

Your end result is a URL that looks something like **http://www.sherpa.ac.uk/romeo/api29.php?ak=[APIKEY]&qtype=starts&jtitle=Reference+%26+User+Services+Quarterly**. Fill in your API key and paste this into a browser. You should get some XML with the information from SHERPA/RoMEO.

What would happen if you put in just **='+value'**? It would give you this:

**http://www.sherpa.ac.uk/romeo/api29.php?ak=[APIKEY]&qtype=starts&jtitle=Reference & User Services Quarterly**. If you try that URL, you'll see it doesn't work. The ampersand throws off the search since this is a character that is--you guessed it--also used in URLs! So you have to escape the characters to make sure any special characters or spaces don't mess up your search.

I'll run my fetch URL function, and after a few seconds I get a new column with a bunch of XML:



### Getting what you want from the XML

I am really interested in knowing what version of the PDF I can use, so I really want to make a column for each section of the XML which includes that information. The first step in parsing XML in OpenRefine is to look carefully at all the XML elements you want to use.

In this case, the ones I want are:

<prearchiving>, which is the pre-print version

<postarchiving>, which is the post-print version

 <pdfarchiving>, which is the published version

and

<condition>, which are any stated conditions.

Note that for the first three, there's only one occurrence. For the last one, conditions, there may be many of them. These require different approaches, so we'll take them one by one.

OpenRefine has a few different built-in parsing functions. They are:

|  |  |
| --- | --- |
| parseHtml | (string s) |
|  | returns: HTML object |
|  | Parses a string as HTML |
| parseJson | (string s) |
|  | returns: JSON object |
|  | Parses a string as JSON |

In practice, you can use parseHtml() for XML, since it has the <> characters. JSON is also very helpful, and we'll use that next week.

Here's how to grab just one element. Click on SHERPA, and then "Add column based on this column." Type into the box: **value.parseHtml().select('pdfarchiving')[0].htmlText()**

Let's break it down:

**value.parseHtml()**runs the parseHTML function on the cell value.

**.select('pdfarchiving')[0]**says "I want to select the first element from within the PDFarchiving element." 0 starts lists--if there were multiple and you just wanted one, you could pick whatever number corresponds with the right one.

**.htmlText()**says "just give me the HTML text, not the full thing."

A screenshot of a cell phone

Description automatically generated

*Note: You may see a parsing error if no data is available for the first row. It does not necessarily meant that the expression won't work.*  
  
Here is the result:

A screenshot of a social media post

Description automatically generated

You'll see that I now have a new column called "pdfarchiving." It does not contain data for every row, but for some (as pictured) it will contained the value "cannot," which means that I cannot archive the published version of the PDF.

I can continue to do this for all the other data elements that I'm interested in (prearchiving, postarchiving, and condition).

Using this same construction for multiple elements won't work as well. Let's see what would happen if you tried using it on the "condition" element.

**value.parseHtml().select("condition")[0].htmlText()**

For *Reference & User Services Quarterly*, this gets us the first condition, but there are two conditions. We can change the [0] to a [1] and get the second one, but this means you would have to make a new column for each condition, and you wouldn't know in advance how many columns you needed.

Luckily, there's a built-in solution to this problem called a "for loop," which if you are unfamiliar with this just means a way to run the same function until you run out of elements to run it on. Here's what this looks like:

**forEach(value.parseHtml().select("condition"),v,v.htmlText()).join(". ")**

Let's break it down:

* **forEach(a, v, e)**contains everything that happens in this whole statement before the join function. This just says "keep doing this same thing until you run out of items." It needs three arguments, *a*, *v*, and *e*. The official wording is "Evaluates expression a to an array. Then for each array element, binds its value to variable name v, evaluates expression e, and pushes the result onto the result array." All this means is that it takes the first function and does whatever it says to do and puts those results in a list. The list gets put into the letter v, which then gets one more thing done to do it.
* **value.parseHtml().select("condition")** does exactly what our previous function did for parsing elements. This is "a" in the forEach function.
* **,v** is the list of all the conditions found, and of course is "v" in the forEach function.
* **v.htmlText()).join(". ")**is the most confusing looking, so let's take it step by step:
  + **v.htmlText()**says "everything that was in that list of conditions--just grab the text out of them." This is "e" in the forEach function.
  + If we left this alone, we would see the correct list of items in the preview window, but it would look like this: *[ "On author's personal website, institutional website or institutional repository", "Published source must be acknowledged" ].*As we saw before, this doesn't actually do anything. The **.join(". ")**runs on the entire list and says "join up all the items with a period, followed by a space. You could put anything you wanted in those quotes, and that would separate your list items. In this case, a period works well.

You can use a forEach() function on anything that contains a number of items and do the same thing on all the items. But for this particular data, "condition" was the most important thing to grab.

After this, we could get additional information if we wanted it (like the ISSN), or simply remove the SHERPA column. You can also use the "move column" options to move the columns wherever you want them. An image of my final result is below. Note that there are still errors and missing data, but we've come a long way from a web page!

A screenshot of a cell phone

Description automatically generated

## Week 3 file

You can use this data if your Week 2 data didn't have journal titles in it. Note that this does need some cleanup before you use it, in particular removing trailing and leading whitespaces. You are welcome and encouraged to use a data set that is more meaningful to you, but this one will allow you to complete the assignment.

Click [week3 - Editable.xlsx](https://www.libraryjuiceacademy.com/moodle/pluginfile.php/64984/mod_resource/content/1/week3%20-%20Editable.xlsx) link to view the file.

## **Using the SHERPA/RoMEO API on your CV**

Follow through the steps in this week's lesson on your own data.

1. Get a SHERPA/RoMEO API key, and use it to construct a query to search for journals in the CV you processed last week.
2. Use the parseHtml() and forEach() functions to grab the data elements you want--you're not limited just to the ones in the example.
3. Organize your file into something that will be useful for you.
4. Turn in your JSON and exported Excel file.

# Week 4

## Sources of Bibliographic Data

In this week's class, we'll look specifically at bibliographic data. One of the things we've been working on in discussions is the types of data cleanup you may have to do on a daily basis, and what  tools you normally use. One of the issues with working with library data is that it usually is in a format that's not quite like a spreadsheet that's easy to import into OpenRefine. We have learned a lot of tricks for converting text into structured data over the last few weeks, but here are specific tips on other tools that you can use to help you get your data into and out of OpenRefine in a structure that makes sense. We won't go over all of these in detail, but hopefully this will give you an idea about what approaches you may take.

1. MARC
   * [MARCEdit](http://marcedit.reeset.net/) allows you to do batch editing of MARC files. You can use it in a few different ways with OpenRefine. Use the **Export as Tab Delimited**feature to create a TSV file you can import into OpenRefine. If your data isn't going back into MARC, you can get just the fields you need and process it (using the tools below, perhaps). If it is, you can use **Delimited Text Translator**to process your OpenRefine results back into MARC for loading into your ILS or another system.
2. ContentDM
   * Use MARCEdit to get your data out of the catalog, or simply start with an Excel file. OCLC [provides instructions](https://www.oclc.org/content/dam/training/CONTENTdm/pdf/Tutorials/Importing%20Items-Creating%20Objects/Importing%20from%20Tab-Delimited%20Files.pdf) to create spreadsheets for metadata import into ContentDM.
3. Digital Commons
   * You can use [Digital Commons Excel Bulk Upload, Export, and Revise](https://www.bepress.com/reference_guide_dc/batch-upload-export-revise/) to process metadata in bulk. Again, you can use MARCEdit to create TSV files from MARC records that you process through OpenRefine into the proper format or start with another spreadsheet.

For this week, I'll provide a Digital Commons bulk revise file for practice, and you can work on obtaining your own data for your final project. Please [email me](mailto:greer.martin@gmail.com) if you need help getting your data in the right format.

## **Using the FAST API**

### **What we're trying to do**

For this week, I provided a list of 12 math faculty publications in our institutional repository, which is also harvested into the library catalog with an OAI feed. The "keywords" column contains user-assigned tags for the article. We might want to improve those tags for better discovery in our library catalog so that they match up with other library-controlled vocabularies. One approach might be to use OpenRefine to automatically verify which of those keywords are valid FAST headings and don't need further review.

In this lesson, we'll see how we'd go about doing that. First, download the Practice Bibliographic Data File, and import the Excel file into OpenRefine to create a new project. We'll use this to practice the skills that you can apply to other data.

OCLC's [Faceted Application of Subject Terminologies (FAST) subject headings data](http://fast.oclc.org/searchfast/) has several APIs. We'll be using the [assignFAST API](https://www.oclc.org/developer/develop/web-services/fast-api/assign-fast.en.html" \t "_blank) for this exercise. This is an API designed to perform autocomplete searches for FAST terms. We can look at an example.

The first item in the practice file is called "Deformations Associated with Rigid Algebras." The first term in the keyword column is "cohomology." A [search for cohomology](http://fast.oclc.org/searchfast/?&limit=keywords&facet=all&query=Cohomology&sort=usage+desc&start=0#&single=fst00959720&fullview=simple&sep=click) reveals that "Homology theory" is the term used for cohomology in FAST, and so if we wanted to ensure that clicking on this term in the catalog found related items, we might want to consider changing to or at least adding "Homology theory" to the list of keywords. You could look up every single term manually, of course, but using the API would allow us to do a lot of the lookup automatically. As with the SHERPA/RoMEO API this won't remove all the work, but it will allow you to focus your efforts on the more complex problems.

### **Constructing your API call**

The [required parameters for assignFAST](https://www.oclc.org/developer/develop/web-services/fast-api/assign-fast.en.html) are as follows:

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| query | The query to search |
| queryIndex | The index corresponding to the FAST facet. (see below) |
| queryReturn | Information requested list, comma separated. (See below) |
| numRows | headings to return maximum restricted to 20 |
| callback | the callback function name for jsonp |

Those indices mentioned are here:

|  |  |
| --- | --- |
| **name** | **description** |
| suggestall | All facets |
| suggest00 | Personal names |
| suggest10 | Corporate names |
| suggest11 | Event |
| suggest30 | Uniform title |
| suggest50 | Topical |
| suggest51 | Geographic names |
| suggest55 | Form/Genre |

The possible Query Returns are:

|  |  |
| --- | --- |
| idroot | FAST Number |
| Auth | Authorized Heading, formatted for display with—as subfield separator |
| Type | alt or auth—indicates whether the match on the queryIndex was to an Authorized or See heading |
| Tag | MARC Authority tag number for the heading—100= Personal name, 150 = Topical, etc. |
| Raw | Authorized Heading, with subfield indicators. Blank if this is identical to auth (i.e., no subfields) |
| breaker | Authorized Heading in marcbreaker format. Blank if this is identical to raw (i.e., no diacritics) |
| indicator | Indicator 1 from the Authorized Heading |

The format this should take is:

http://fast.oclc.org/searchfast/fastsuggest?&query=[query]&queryIndex=[queryIndex]&queryReturn=[queryReturn]&suggest=autosuggest&rows=[numRows]&callback=[callbackFunction]

Note that this doesn't require or even request an API key. This is an experimental service, and that's something to be wary of if you plan to use it for a major on-going project.

With all that in mind, let's construct a URL that will work.

**[query]**will be the **value**from an OpenRefine cell.

**[queryIndex]**can be "suggestall".

**[queryReturn]**can be suggestall, auth,and type (you have to %2C to separate them)

**[rows]**can be 1 for now.

**[callbackFunction]** is a name we choose, but we don't strictly need it.

So our final URL is:

http://fast.oclc.org/searchfast/fastsuggest?&query=value&queryIndex=suggestall&queryReturn=suggestall%2Cauth%2Ctype&rows=1

You can type that into a browser and see the raw JSON it returns.

### **Putting this into OpenRefine**

Let's first look at our data structure in OpenRefine. Each keyword is separated a comma, and there are multiple keywords per cell. So just typing in something similar to the SHERPA/RoMEO call won't work:

**'http://fast.oclc.org/searchfast/fastsuggest?&query='+escape(value,"url")+'&queryIndex=suggestall&queryReturn=suggestall%2Cauth%2Ctype&rows=1**'

This will give us something like this, which obviously will never match anything!

**http://fast.oclc.org/searchfast/fastsuggest?&query=Cohomology%2C+Deformations%2C+Diagrams+of+algebras%2C+Rigidity%2C+Punctured+sphere%2C+Weyl+algebra&queryIndex=suggestall&queryReturn=suggestall%2Cauth%2Ctype&rows=1&callback=mathtest'**

There are several ways we could approach this, but to keep things simple, we'll first split the columns by separator of comma, and now have keyword columns 1-11. We'll run the URL on each column. Let's do the first one:

A screenshot of a social media post

Description automatically generated

A screenshot of a cell phone

Description automatically generated You'll see we get JSON that looks like this. While there's a lot we could get out of this, let's focus on the one in bold, cohomology theory. We want to isolate just this piece.

{ "responseHeader":{ "status":0, "QTime":1, "params":{ "fl":"suggestall,auth,type", "q":"suggestall:Cohomology", "rows":"1"}}, "response":{"numFound":8,"start":0,"docs":[ { "suggestall":"**Cohomology theory**"}] }}

The parseJson() function isn't all that easy to figure out from the documentation, but basically we can "walk through" the different levels to pull out just the text we want.

**value.parseJson()["response"]["docs"][0]["suggestall"]**

Let's break it down:

* **value.parseJson()** shows you the JSON. If you type just that piece in, you'll see all the JSON results.
* **["response"]** says "focus on just the set of results dealing with the response."
* **["docs"]**says "within response, just look at the docs array."
* **[0]**says grab just the first item within docs.
* **["suggestall"]**grabs just the text from within suggestall.

As you'll recall, we only got one row back, so we don't have a lot of items to deal with. If we did, we'd have to use a for loop to process all of them one by one and return joined, just as we did with the parseHTML function.

A screenshot of a social media post

Description automatically generated

Now we have one result back from FAST. Let's say that we want to change any keyword to the FAST heading it matched (of course this might not be such a great idea if the responses were wrong!)

First, create a text length facet on the FAST 1 column, and unselect the "errors" box, leaving only "Numeric" checked. This will only include cells that found something.

A screenshot of a social media post

Description automatically generated

Now click on the Keywords 1 column, and transform. In this case, we don't need a function, we are just finding a particular value from another cell.

**cells["FAST1"].value**

This gets the value of the cell in the FAST1 column for each row. We can now remove the FAST1 column since we have everything we needed from it.

A screenshot of a social media post

Description automatically generated

You would follow this same process for each column. Eventually, you would have a series of Keyword columns that you need to put back together into one column for reuploading to Digital Commons. We can use the same replacement technique to create a new Keywords column.

First though we need to put some kind of value in the blank columns. For each keywords column, select "Custom Facet", then "Facet by Blank" (the opposite of what we did above). Select "True" from the facet list to filter to just those cells. Transform the blank cells by click on Transform, then typing " " into the box. That will make a white space that you can't see, but won't break an attempt to combine cells.

A screenshot of a cell phone

Description automatically generated

Now we'll create a new column call "keywords" based on the "keywords 1" column. In the box, type

**value + ", " + cells["keywords 2"].value+ ", " + cells["keywords 3"].value+ ", " + cells["keywords 4"].value+ ", " + cells["keywords 5"].value+ ", " + cells["keywords 6"].value+ ", " + cells["keywords 7"].value+ ", " + cells["keywords 8"].value+ ", " + cells["keywords 9"].value+ ", " + cells["keywords 10"].value+ ", " + cells["keywords 10"].value**

This will create a new column with all the keyword columns. The extra commas can be removed with this regular expression:

**value.replace(/(?<![a-z])[,]/,"")**

This says:

**value.replace()** in the value, replace as follows.

**/(?<![a-z])[,]/** means "when you come to a comma, look back see if a letter came before it. If so, don't match this. If you don't see a letter, go ahead and match."

**,""**replaces with nothing.

(You might feel that you remove those extra columns more quickly in Excel. That is valid too!)

## **Reconciliation Services in OpenRefine**

### **What is reconciliation?**

Reconciliation is the process of matching values in an OpenRefine cell to linked data vocabularies and seeing what matches. It's similar to what we just did with FAST, but actually a little simpler. Essentially, it will search a vocabulary, determine whether a match is likely, and then let you create a link to the vocabulary. Again, this automates some of the lookup process, and can be used to extend your metadata. [The Rockefeller Archive Center's Digital Team blog](http://blog.rockarch.org/?p=1230) has a post about one use for this.

### **Reconciling against Getty AAT**

Some organizations have published their data as linked open data and have also created reconciliation services. One such organization is Getty for their [vocabularies](https://www.getty.edu/research/tools/vocabularies/obtain/openrefine.html). The services is available by a URI, which when plugged into OpenRefine, enables a user to perform reconciliation against a dataset in OpenRefine.

Skim the list of other reconcilable data sources [here](https://github.com/OpenRefine/OpenRefine/wiki/Reconcilable-Data-Sources), under the "hosted services" section.

Let's try it.

Note: Firefox works better for reconciliation than Chrome, which obscures some important information when using OpenRefine. Copy and paste the OpenRefine address into Firefox, your project should still be there.

To add a Getty reconciliation service, first open a new project using the spreadsheet reconciliation.csv in this week's course materials. We'll reconcile the terms in the "Collection" column, which do not adhere to any controlled vocabulary, against the Getty AAT vocabulary. The AAT (Art & Architecture Thesaurus) is a controlled vocabulary for art, architecture, and material culture.

Once the project has been created in OpenRefine, we'll make a copy of the column that we want to reconcile. This is because reconciliation changes data, so it is wise to make a copy, so that you may go back to the original if need be.

On the Collection column, select Edit Column > Add column based on this column. Select a new column name in the next window, and leave everything else as is. We'll use "collection\_reconciled" for our copied column.

On our new column, select the drop down menu > reconcile > add standard service. Copy and paste URI: <http://services.getty.edu/vocab/reconcile/> and select "AAT" from the radio buttons. Click "start reconciling" and wait a few minutes for the program to run.

When it is finished you will see a list of suggested matches beneath each term in the column. The suggested terms come from AAT, they presented as candidates based on the original term.

If you hover over the suggestions, a window will appear with a definition of the term. If you click on a candidate, a web page will open in AAT which defines the term.

So far, you only have candidates for reconciliation. To choose one and make a reconciliation requires user action. If click the single blue check box, that instance of the term will be reconciled to the AAT vocabulary term. If the double check box is select, every instance of that term in the column will be reconciled to that candidate.

Go through and try to reconcile a few. You won't have enough information in the dataset to be sure of the correct candidate, but make your best guess (this is just practice).

### **Reconciling against Wikidata**

Another thing that we can do with reconciled data is to pull in other related properties from the vocabulary. For instance, once we have reconciled our collection terms against AAT, we can add identifiers (also from AAT) for each reconciled term. This is called extending data, and many reconciliation services provide this via OpenRefine.

Let's try this with the same dataset, reconciled.csv. This time we'll reconcile university names against Wikidata, and then add geodata from Wikidata for the reconciled names.

First name a copy of the "Institutional Affiliation" column (Edit column > Add a Column Based on this Column). Select the drop down menu for your new column of copied data, Reconcile, then "Start reconciling." Wikidata should already be available as a service. In the next window, you have the opportunity to choose the type of objects that you are reconciling. It is recommended to select the broadest term. If an appropriate type is not in presented as an option, search for a better term in Wikidata. For instance, the institutions in this column are not only universities, there are secondary schools as well. So, "university" is too narrow, as is "Private not-for-profit educational institution," two of the presented choices. Let's try a broader term, like "educational institution." Type that into the "reconcile against type" field in the reconciliation window in OpenRefine. Suggestions from Wikipedia will auto-populate (see below).

A screenshot of a cell phone

Description automatically generated

Select Start reconciling and wait for the job to finish. Once it has, select matches for as many institutions as you can. If a term is blue and linked, it has automatically reconciled, meaning it found a candidate that it was so confident it could match, it did it for you. Once you have finished, we'll add in our geodata.

### **Extending Data**

Now that most or all of our institution names are matched to Wikidata, we can add in anything other properties of that Wikidata object. For instance, the Wikidata page for the first institution, McGill University, shows several properties under the "statement" section. Below is a screenshot of a few of these. These are examples of other information about McGill University that we can add to our dataset.

A screenshot of a social media post

Description automatically generated

One of these properties is "coordinate location." It will supply us with geo-coordinates for this, and all the other reconciled university names.

To add this in, select the Institutional Affiliation drop down menu, Edit Column and "Add column from reconciled column." This will open a new window. In the "Add property" field type in "coordinate location." You will see that the available properties will auto-populate. You will also be given a preview of the data that will be pulled in. Click OK, and a new column will be added with coordinate data.

## **Your Cleaned Up Dataset**

Throughout the weeks, you've been discussing data sets that you use regularly, and issues you have with them. For your final project, you should pick some data that needs cleaning--just a small portion of it for testing--and do the following:

1. Write down all the issues you have with the data. What is wrong with it? What could be improved? For instance, do you need to create structure out of data that has no structure? Clean up years of character encoding issues? Or do you need to know what is a valid subject heading?
2. Once you have an end goal in mind, create a project in OpenRefine and import your data.
3. Work through the steps we've discussed throughout the past weeks to create your end goal data. Try to use regular expressions, GREL, and an API on your data. If you need suggestions, ask on the discussion board, or email me.
4. Once your data cleanup is complete, describe the steps you took, including any errors or missteps you had along the way. Post this on the Week 4 Discussion board.
5. Turn in your original data (or a description of it), your JSON extract, and your final extracted Excel file.