Augmenting Activity 3: Using Add Column by Fetching URLs

You may have noticed from Activity 2 that sometimes you searched for a property to add from Wikidata, but got an error. Other times you might want to augment your dataset with data that isn't provided by a reconciliation service. In both cases, you might want to try calling an API directly through the Add Column by Fetching URLs feature.

The goal of this activity is to try out using the Add Column By Fetching URLs feature by using the Wikidata API to add the property "first line" to our data (the first sentence of a particular book). You could also add the property "first line" using the *Add columns from reconciled values...* feature, as we did in Activity 2, but we will use this property as an example of a different way to access this data. We will make an API call, and then parse the resulting JSON data for the property information we want. This is a normal workflow when augmenting data with APIs.

Note: Complete Augmenting Activities 1 & 2 first before attempting this activity.

In this activity, you are going to:

A. Add a column of JSON data resulting from a Wikidata API call
B. Parse this JSON data column to obtain the property "first line"

A. Add a column of JSON data resulting from a Wikidata API call

1. Wikidata not only provides a reconciliation service, but also a web API that we can use. To learn more about how to use this API, see: https://www.wikidata.org/w/api.php.
2. We want to "get" some data, so **try doing a find (CTRL-F) on this web page for "get".** Scrolling through the list of "get" actions, we should see that there is an action called ***wbgetentities***. If you **click on its link**, you get a page that describes how to make the API call. If you scroll to the bottom, you see some examples. We are going to use these examples to formulate our API call.
3. We used the 4th example on this page to construct most of this API call. To add the JSON format part, we went back to the main page of the API instructions, and searched for "format". If you click on the JSON link, you will see a page with more instructions and examples of how to specify JSON format.
   In order to get the JSON data for each book (all the data from the Wikidata page in JSON format), we need to use the following API call:
   `https://www.wikidata.org/w/api.php?action=wbgetentities&ids=<BookID>&languages=en&format=json`
   where the **<BookID>** part that is bolded would be replaced with the ID of each book. How do we know the IDs? We know because we got the IDs of the book when we reconciled them – that is one of the results of reconciling data. So this is a special situation where we needed to both reconcile our data and use an API call. We have already reconciled the data,

linking books with their Wikidata entries, so now we will call an API to add information about each book (specifically, its first line) and then clean the results.

4. Before we use this URL in OpenRefine, let's check that it works. We will trying using it to get the JSON data for our *Pride and Prejudice* page. But first we need to know the ID for this book. If you **go to the *Pride and Prejudice* Wikidata page**, the number in parentheses is the ID. So **take that number and substitute it into our API call** to get:

   `https://www.wikidata.org/w/api.php?action=wbgetentities&ids=Q170583&languages=en&format=json`

5. **Put this URL into your browser**. If it is correct, you should see JSON data describing Pride and Prejudice displayed in your browser. The JSON data is hard to read in this format, but it will be easier to work with OpenRefine. We need to make this call for each book in our list using their particular reconciliation ID numbers. There is a special GREL command that allows us to get the ID of an item after it has been reconciled: **`cell.recon.match.id`**. Now we have everything we need to construct a URL for the API call. We will use the concatenate functions to add the ID to the middle of this URL.

6. **From the *title* column *pull down menu*, *select edit columns->add column by fetching URLs*** and give it the name ***JSONData***. **Change the throttle delay to 2000** to shorten the time between API calls to make it go a bit faster. *Note: Certain APIs require there to be delays between making calls – so don't just always make the time shorter – you may be blocked! Remember to read the documentation!*

7. **For the URL expression, type:**
   `"https://www.wikidata.org/w/api.php?action=wbgetentities&ids=" + cell.recon.match.id + "&languages=en&format=json"`

8. You should see a preview of the result from the API call. If all looks correct, **click on OK.**

B. Parse this JSONdata column to obtain the property "first line"

9. Now you should have a column with all the properties for each book in JSON format. The last step is to parse out the JSON to find the particular property you want. We need to refer to the *title* column in the process, so let's quickly rename it to make this easier. **From the _ - *title* column *pull down menu*, select *edit columns->rename this column*** and give it the name ***title.***

10. We are going to create a new column to extract out the *first line* from the JSON data. To do this, we need to know what the property ID is for first lines. You should still have your *Pride and Prejudice* Wikidata page open **– go to it and click on *first line*.** At the top of that page is the property number in parentheses – ***P1922***. We will use this parse our data.

11. **From the *JSONData* column *pull down menu*, select *edit columns->add column based on this column*** and give it the name ***First Line.***

12. For the expression, as we type, we will look at the preview window to see how we are narrowing down our search through the JSON to find the data we are looking for. To parse the JSON, we can use the function ***parseJson().*** So we start with our expression **`value.parseJson()`**

13. Our JSON data is hard to read and understand, so here is a trick that can help us in this particular situation. **Put the URL for the API call (from Step 4) back into your browser, but add "fm" to the end of it to make "jsonfm".** Now you should be able to read the JSON

easier, as it is nicely formatted in HTML. This does not work for all API calls, but it is an option for this one. Now we can refer back to this as we parse the data to help us understanding the nesting.

14. Back in OpenRefine, in the preview window, you will see attribute names, such as "entities", and then a colon and a curly bracket and then a new attribute name, etc. You need to follow the attribute names, colons and curly bracket trail to find the property you are looking for, as you type. To dive into the curly brackets of an attribute name, you add *["<attributename>"]* to the end of the expression. Note: The next six steps will guide you through the JSON data's structure until you have just the first line of each entry. If you would like further instruction on JSON, click here: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON
15. So if we select the data about "entities", our expression would be
    `value.parseJson()["entities"]`
16. The next attribute is trickier, as it is actually the reconciliation match ID we used earlier in the API call. OpenRefine helps us refer back to these variables, in this case by writing **`cells.title.recon.match.id`** This is slightly different than before, because when we made the API call we were in the title column, this time we are in the JSONData column, so we have to refer back to the title column by using *cells.title*. The *recon.match.id* part is the same as before. As we are using a formula for the name of the attribute, we don't put it in double quotation marks. So our expression is now
    `value.parseJson()["entities"][cells.title.recon.match.id]`
17. Next we see different attribute options, such as *aliases* and *claims*. We can see that there are "P" values within the *claims* curly brackets, so we can guess that all the properties are in that attribute. Check your "pretty" JSON preview in your browser to confirm. So our expression is now
    `value.parseJson()["entities"][cells.title.recon.match.id]["claims"]`
18. Then we want our particular property, P1922, so we add that to the expression, so it is now
    `value.parseJson()["entities"][cells.title.recon.match.id][ "claims"]["P1922"]`
19. Now we have a square bracket instead of more curly brackets. That signals that what we have is an array, but if you take a look at the remaining JSON, you'll see that the first item is what we want: the first line in English. So to go into the square brackets to get the first item of this array, we type *[0].* So our expression is now
    `value.parseJson()["entities"][cells.title.recon.match.id][ "claims"]["P1922"][0]`
20. Finally, we just continue to work through the attributes to get to the text for the *first line* property. So our final expression is now
    `value.parseJson()["entities"][cells.title.recon.match.id][ "claims"]["P1922"][0]["mainsnak"]["datavalue"]["value"]["text"]`
21. If everything is correct, you should see the first line appear in the preview. If so, **click on OK**.
22. This expression may seem daunting at first, but the preview window really helps you to parse the JSON and see what you are doing. It might take some trial and error, but eventually you will be able to extract what you want. This was quite a complicated and nested JSON file – often it is not as bad as this one was!

Now you're ready for Activity 4!