Cleaning the Cornelia Dataset Using OpenRefine

First assignment for 2021 – 2022 Introduction to Digital Humanities class

Lara Peeters **GitHub:** https://github.com/ejgenc/cornelia-dataset
0462266
lara.peeters@student.kuleuven.be
Wangzhi Xi
0867619
wangzhi.xi@student.kuleuven.be
Eren Janberk Genç
0866892
erenjanberk.genc@student.kuleuven.be

Brief desciption of the dataset

The dataset which we worked on is a subset of the data collected within *Project Cornelia*. *Project Cornelia* is a multi-disciplinary project that bridges computer science and art history with the aim of providing a hybrid research engine. The main dataset that powers this research engine is related to 17th century Flemish tapestry and painting. Further details about the specifics of the dataset that powers *Project Cornelia* can be found in an article written by Koenraad Brosens et al.[1]

The subset that we were tasked to clean for this assignment contains data entries that are sourced from a register of the guild of painters, goldbeaters and stained-glass makers in Brussels. These registers contain information about the apprentices and masters which were present in the guild within the timeframe between 1599 and 1706.

In order to fully understand the technical and semantic qualities of the subset we aimed to clean, we followed a programmatic approach using Python and two well-known Python packages: Jupyter Notebooks and Pandas. The dataset was loaded into a Jupyter Notebook environment, and several queries were ran on it to understand its extent and uncover any possible problems. We chose to follow such a methodology alongside using OpenRefine for checking values because we believed that only a systematic study would reveal all the problems that were present in the dataset. This Jupyter Notebook is attached to the assignment submission file and can also be found on the assignment's GitHub page.

Size of the dataset

We found that the dataset consisted of eleven columns and 2830 rows in total. Right away a data quality problem became apparent: there were null values present in some crucial columns such as date_day, date_month, actor_first_name, actor_surname and phase.

"source" column

The source column indicates which guild register book the entry was taken from. There were no null values in this column and all the rows had the same entry "BRGA818" as a value. This indicates that all the entries in the dataset were sourced from a guild register book recovered from Brussels.

"source_entry" column

This column aims to inform about which exact entry in the register book the information present in the row comes from. While there were no null values in this column, there were inconsistencies in the way in which this data was presented. We discerned more than four different notations for this column. Upon further inquiry, we realized that this column was fed by the three columns that followed

[1] Brosens, Beernes, Cardoso and Truyen, "The Brussels guild of painters, goldbeaters and stained-glass makers, 1599-1706: A Prismatic analysis", *Zeitschrift fur kunstgeschichte* 4,82 (2019): 531-553.

it: “date_day”, “date_month” and “date_year”. The first two to four digits of some “source_entry” values had the exact same value as “date_day” and “date_month” columns. We noted this (a string with the *ddmmyyyy[A-Z])* notation as the canon notation.

“date_day”, “date_month” and “date_year” columns

These three columns indicate exactly when the information in any given row was entered into the registrar. The columns “date_day” and “date_month” included some null values whereas the “date_year” column did not.

“actor_id”, “actor_first_name” and “actor_surname” columns

These columns serves to uniquely identify each artist that was represented in the dataset. While every row had a value for the “actor_id” column and every “actor_id” was mapped uniquely to only one name and surname combination, the two other columns had null values. However, there were no “totally anonymous” actors in the dataset: each actor either had a surname or a name. It was also noted that three different ways of indicating missing information was utilized for the “actor_first_name” column: some rows used the string “*anonymous*”, some rows used the string *“[nn]”* (for not known) and some rows utilized null values.

“role” and “status” columns

These two columns give detailed information about the artistic character of a given actor. They had no missing values. The “role” column had ten unique values and the “status” column had eight. However, we noted that the roles and statuses were presented in both English, Dutch and Old Dutch. It was also apparent that some unique values could be collapsed into one. We also realized that the generic role of “member” could be replaced with another more appropriate role for actors that appear more than once in the dataset.

“phase” column

This column gives information about the reason why an actor was noted in the register: either at the start of their duty or at its end (quitting or dying.)

Selected problems

In the light of our analysis above, we decided to take the following actions to improve the quality of the dataset:

1. The “source_entry” column should be unified with a single notation using the “DDMMYYYY[alphabetic character]” notation.
2. The null-proxy values of the “actor_first_name” column should be replaced with proper null values to comply with the rest of the dataset standards.
3. The values in the “role” column should be translated into English and redundant roles should be collapsed into one.
4. The values in the “status” column should be translated into English and redundant roles should be collapsed into one.

We decided not to take the following actions that might have potentially improved the quality of the dataset:

1. We decided against filling the null values present in various columns using the information in other columns. For example, we did not fill in the first names and last names of the actors in the dataset by looking at similarly named actors with similar roles.
2. We decided against trying to replace the “role” value of the rows who had “member” as their role by looking at similarly named entries.

Our justification for not taking these actions was twofold: as we had no “base truth” database, we feared lowering the quality of the dataset instead of improving it. We also faced a technical constraint: we could not get our heads around working with multiple columns and comparing rows by grouping them in OpenRefine.

Steps taken to solve the problems

“source_entry” problem

As we knew that the information in this column could be constructed by looking at the three subsequent columns, we decided to wipe all information except the part that could not be extrapolated and reconstruct the column from scratch. We executed these commands in the following order:

1. Remove all the digits of the “source_entry” values except the last digit if the last digit is an alphabetic character

   Select “source_entry” column → edit cells → transform → Execute *if((isNumeric(value[-1]) == false), value[-1], "")*

2. In order to recreate the “source_entry” column, we first had to normalize all the values in the “date_day” column. More specifically: null values needed to be mapped to “00”, and single-digit dates need to be mapped to the “0x” format.

   Select “date_day” column → edit cells → transform → Execute *if(isNull(value), "00", value)*, then *value.replace(/^\d$/,'0$0')*

3. We repeated the above operation, this time for the “date_month” column.
4. After these operations, we recreated the “source_entry” column using the following commands:

   edit column → join columns → *reorder by dragging* to the following order: date_day, date_month, date_year, source_entry → *select* all four → OK.

5. Lastly, we replaced the “00” values in the “date_day” and “date_month” column with null again to insure consistency.

   Select [“date_day” / “date_month”] column → edit cells → transform → Execute *if((value == "00"), "", value)* then *if(value[0]=='0', value[1],value)*

“actor_first_name” problem

We executed the following GREL commands to get rid of additional ways of indicating null values:

Select “actor_first_name” column → edit cells → transform → Execute *value.replace("[NN]", "")* and *value.replace("[anonymous]", "")*

“role” problem

The problem with this column was a problem of translation. We executed the following commands via the “transform” option, only this time selecting the “role” column.

value.replace("apotheker","pharmacist")

value.replace("vergulder","gilder")

value.replace("glasschilder","glass painter")

value.replace("gelaesschryver","glass painter")

value.replace("goudslager","goldsmith")

value.replace("plaatslager","plate craftsman")

“status” problem

As with the “role” column, we provided our own translations by executing the following commands:

value.replace("leermeester","tutor")

value.replace("meesterszoon","master's son")

value.replace("ouderman","dean")

value.replace("recognitie","non-sworn in master")

value.replace("cortosie","non-sworn in master")

Description of the result

In the end, we made operations on 6 columns to fix the problems that we found in the following 4 columns: “source_entry”, “actor_first_name”, “role” and “status. The result is a dataset of the same size (2830 rows, 11 columns) because no rows were dropped for lack of data. The resulting dataset shares a common way of indicating null values, and all the values except actor names are in English instead of Dutch and Old Dutch.

Description of the workflow

In producing this assignment, we followed the “meet – delegate – meet” approach: we met to discuss the details, delegated the work in between us and met again to review and re-delegate. We had three meetings (two face-to-face, one virtually) following this pattern on the dates 03/11/2021, 09/11/2021 and on 17/11/2021.

In the first meeting, we worked on the requirements of the assignment and decided on the general course to follow. The decision to follow a problem-centric approach was taken. To this end, we took up the task of creating a Jupyter Notebook and systematically investigating the dataset. The Jupyter Notebook was placed in a GitHub repository along with other files such as the non-mutable original dataset. The GitHub repository allowed the team members to work asynchronously and to use version control. Moreover, it was seen as a tool that could be used to smoothly start the second assignment.

After the problems were presented, the team decided on which problems to tackle. Some problems were left untreated for reasons previously stated above. The problems were once again delegated to different group members. It was decided that each team member would document their OpenRefine implementation of the selected problem solutions. The solution documentation was then aggregated by a specific team member and the dataset was updated accordingly by the said member. At the last meeting, the solutions were reviewed and the report writing task was separated into different parts. The different parts were once again delegated, aggregated after completion and then edited by a single team member.

OpenRefine Evaluation

The following section includes our brief remarks about working with OpenRefine.

“OpenRefine was very easy to use with its built-in functions. It was easy and fast to change cells and clean a column. However, it was also limited in the columns you can change per function. We couldn’t find a function that looked for multiple equal cells in a single row to find duplicate rows. Because of this, there are probably still some duplicate entries within the dataset.” **Lara Peeters**

“My experience with OpenRefine is overall good. The interface is clear with few unnecessary elements, and most functions are very straightforward. For me, the best function is Undo/Redo, by which I can jump to every step I have made.” **Wangzhi Xi**

“OpenRefine offers an easy-to-use interface. However, it lacks the expressiveness of scripting languages. It also does not play well with principles of replicability and version control: it is hard to replicate the results, and you cannot follow a complex workflow with branches, merges and revisions.” **Eren Janberk Genç**