**T/Th | PROJECT ELT WRITE-UP**

Jason Bell, Leanne Stacey and Seblewongel Wolde

Nov. 23, 2019

**Premise: Cataloguing the Metropolitan Museum of Art**

We want to create a virtual gallery for paintings in the Met, but their cataloguing practices for their artworks suck!! So we had to create our own database from which to maintain and support this virtual gallery website. (Note: We didn’t actually build the website….yet).

**Initial Extraction & Transformation:** We chose to use a database (.csv) downloaded from Kaggle (link). This dataset was pulled via open access from the Met, which we could have pulled from directly; however, the csv appeared to be accurate and was much more readily available.

We used jupyter notebook to do our initial extraction and transformation. This included:

**Extracting desired information** – We identified 12 fields that were relevant to our objective.   
We loaded up the csv and pulled only the identified fields into a pandas data frame. We then  
filtered only to keep painting objects. This gave us an initial data set of ~5,500 records.

**Cleaning** – We identified columns with missing information. Paintings that did not have any ‘Title’ were dropped. There were also missing artists and creation/finish dates, but we didn’t want to drop these as well, so instead we would account for them with a fill of ‘Unknown.’ To get to our desired data for our SQL tables, we needed heavy cleaning on artist name and dates:

* **Artist Name:** The original data set contained two artist fields: full name (first and last, together) and then a ‘alpha sort’ (last, first). We used the latter in the hope that the alpha sort was meant for a search or index function, in which case it would be standardized….but it wasn’t.  
    
  Within artist names, there were many instances of non-names, e.g., ‘French Painter’, or Mono-named artists, e.g., ‘Goya’, ‘Raphael,’. Additionally, some artists had nicknames or notes includes in parentheses (e.g., ‘Surname, First Name (also occasionally known as different name)’). Additionally, we encountered artist ‘citations’ (e.g., when an artist’s work heavily uses another painting as a reference) that included multiple artists names delimited by a ‘|.’  
    
  First, we dumped any citation artists by splitting all artist fields (name, birth, death) by the ‘|’ and only keeping the first instance (actual artist). We then split the artist name using the ‘,’ delimiter into two new columns (first name and surname); there were special characters hidden within, but by increasing our ‘holding columns’ to include two dummy variables (‘Blank1’ & ’Blank2’), which we then simply dropped after the split. This had an additional benefit of helping us split and delete these hidden characters. Once first name and surname were isolated, we performed yet another split to remove any text encased in parenthesis for both the first and surname variables—this helped remove many of notes, aliases, etc.   
    
    
  Before we could concatenate the cleaned first and surname together in a master full name, we needed to account for the mono-name artists. So that their name would also pull over into our new variable, we needed to set their non-existent first name with a dummy variable (‘blank\_’). We then concatenated the two together, then stripped the dummy text from the variable to clean the mono-names. Throughout this process we made generous use of the .str.strip() function to constantly remove any spaces, etc.

* **Dates:** Artist birth dates were left blank when unknown, so they were easily accounted for with a .fillna. Some unknown death dates were filled with dummy values of ‘9999,’ which we replaced with ‘unnknown’ to be consistent throughout the data set.

**Transforming into tables** – We decided that the artist information could be stored in a separate ‘Artist’ table. Since multiple paintings could be produced by the same artist, separating the two would be more efficient as we’d be minimizing duplicative data.   
  
We pulled all artist-related variables into a separate data frame and dropped duplicates. This reduced the number of records by about half, which confirmed that we had many repeated artists. We reset the index and copied it into a new column to create an ‘Artist\_id,’ which would become our key for linking back to our paintings. We saved this new data frame as our Artist table, then created a working data frame to merge the Artist\_id back to its respective paintings, using the ‘full name’ variable as our shared value.   
  
We then repeated these steps to save off department as a separate table, linked to the paintings as a ‘Dept\_id’. All tables were saved out to csv.

Artist bio from the original dataset was very inconsistent and sparse, so for our second data source we wanted to extract a brief paragraph and an artist image from the web, to supplement our database. We created a python script that would read our artist table and extract this information from Wikipedia.

However, throughout this process we ran into many challenges with Wikipedia and honestly regret choosing this source. Some of the challenges:

* **Inconsistent formatting** – Sometimes the first <p> tag will contain desired data, other times it was blank and our desired data was in the second <p> tag.
* **Missing artists** – Wikipedia is by no means a comprehensive source. We had many lesser known artists that were not published on Wikipedia (or only on foreign Wikipedia sites). Regardless of whether an artist page was successfully found or not, it could still find an image, header and paragraph text that it would pull in (i.e., wouldn’t trigger our exception handling)
* **Duplicative names** – If an artist shared the same exact name as another historical figured, Wikipedia appends occupation tags (e.g., (painter)). However, we were unable to account for which names might have shared or not.
* **Special characters** – Many foreign artists from our original dataset included special character to account for accents, etc. There was no good way we felt we could account for this in our initial cleaning, so it just had to be passed through the scraper and see if Wikipedia url could account for it.

Nevertheless, we still used the scraper to append information to our artist table. We were somewhat able to account for some of the inconsistent formatting by reading in all the paragraphs within our target div to a list. We then sent up conditional to read through the list and only pull out the first instance of the paragraph following a <b>, as we noticed that many of our desired blurb started with the individuals name in bold text.

If we had more time, we would have also liked to create something similar, that only pulled in images in a .jpeg format, as we found when an ‘non-existent page’ returned, it pulled a .svg file of the Wikipedia logo. A similar conditional could have filtered for this.

Because of these challenges, we knew there would be problems with accuracy. We setup a manner of try-catch blocks to at least track when we encountered a ‘hard error’ (e.g., couldn’t find anything close to what we were looking for). We had 158 records that were logged for later review.   
  
We also performed some post cleaning on bio paragraphs that returned standard messages like ‘Other reasons this message may be displayed’ or ‘…may refer to’. Another way that we could account for this is incorporating a user-based flag system for our virtual gallery; i.e., if something is incorrect, allow users to flag it for review and send a message to an administrator (Ricky the jr. analysts) to review and fix as appropriate.

**Loading the Data:** We read our artist table and the scrape results into data frames using jupyter notebook. We did additionally cleaning to remove unwanted characters from the biography paragraphs (“[‘” & “’]”). We then also did a replace on the two boilerplate error returns (‘Other reasons this message may be displayed’ or ‘…may refer to’) to a standard ‘Biography could not be found.’ The same process was done for all images that were not .jpg (i.e., replaced with ‘Image could not be found’). We dropped duplicates and filled in na, then exported as our final artist csv.

We created out schema use QuickDB and exported (see Architecture image). This was used to create a database and table in postgresSQL, and we imported our csv’s into our final tables.

**WHY we chose SQL:** We chose a relational database because the whole point of our ETL was to create more standardization and structure, which was lacking in the original data source. We wanted something that could speak back and forth between shared commonalities (departments, artists), to improve searchability in the proposed gallery and more readily lend itself to various analyses.