Data Wrangling

At a basic level, three pieces of information are needed to build a recommendation system: (1) information about the items, (2) information about item users, and (3) information about the relationship between items and users. As such, the goal of the data wrangling task was to obtain these three pieces of information. I have divided the data wrangling process into 12 parts. See figure 1 for a broad summary of the data wrangling steps.

Part 1. Data Extraction

File	Description	Source	Type	Quantity of Files
Million Song Dataset	Collection of 1M .h5 files. Each file represents data for one song.	Using the terminal: rsync -avzuP publicdata.opensciencedatacloud.org::ark:/31807/osdc-c1c763e4/remotefile /path/to/local_copy	.h5	1M
Million Song Summary File	Song metadata information.	http://labrosa.ee.columbia.edu/millionsong/sites/default/files/AdditionalFiles/msd_summary_file.h5	.h5	1
Artist Terms and Tags	Artist terms and tags obtained from the Echo Nest API and the Music Brainz.	Using the terminal: rsync -avzuP publicdata.opensciencedatacloud.org::ark:/31807/osdc-c1c763e4/remotefile /path/to/local_copy ***Located in file A*	SQLite	1
Million Song Duplicates	List of known song duplicates.	http://labrosa.ee.columbia.edu/millionsong/sites/default/files/AdditionalFiles/msd_duplicates.txt	.txt	1
Million Song Mismatches	List of known song mismatches.	http://labrosa.ee.columbia.edu/millionsong/sites/default/files/tasteprofile/sid_mismatches.txt	.txt	1

Part 2. Song Metadata Summary File

The song metadata summary file is an .h5 file that has metadata information for 1M song files. The file is divided into three groups: (1) metadata, (2) analysis and (3) musicbrainz. Each group contains numpy arrays for each song.[1]

[1] The aggregate song metadafile was created by the researches of Columbia's LabRosa throught iteration. The code they used can be found here: https://github.com/tbertinmahieux/MSongsDB/blob/master/PythonSrc/create_aggregate_file.py

The **song metadata** group contains arrays that describe basic features of a song such as id, song title, song hotness, artist id, and release album. The **analysis metadata** group contains song's musical features such as key, length, danceability. The **musicbrainz** group includes the year of song release. The wrangling steps for each group of arrays are as follows:

- Song Metadata Dataset: The group's arrays were converted into a pandas dataframe. Once in the dataframe, object type columns were converted to strings and columns with b' ' string were removed. These strings were the result of python parsing the string data into Unicode from another format. There were also removed to make song identifying features consistent since these strings are not found in other complimentary datasets. The resulting data had missing values in some of the columns. The dataframe has 1M entries.

- Analysis Metadata Dataset: The group's arrays were converted into a pandas dataframe. Once in the dataframe, object type columns were converted to strings and columns with b' ' string were removed. These strings were the result of python parsing the string data into Unicode from another format. There were also removed to make song identifying features consistent. The resulting data had no missing values. The dataframe has 1M entries.

- Musicbrainz: The group's arrays were converted into a pandas dataframe. The dataset has two columns, with one of these columns representing year of song release. Zeroes in the year column represent missing. As such, zeroes were replaced with 'NaN' using df.replace and np.nan. The dataframe has 1M entries.

An alternative approach to using the song metadata summary file was to use an iterator to extract the information of interest from each of the individual song's .h5 files. The difference between the summary file and the full dataset, is that the full dataset contains artist similarity analysis and in-depth song analysis. This approach was explored; and a function, as well for loop were built to extract the data from the larger dataset. However, during testing the amount of time that it took to complete iterating through the files was substantial. Using the summary metadata file showed to be more efficient since the iteration work for the song's metadata has already been done. Moreover, for the purposes of building a recommendation system prototype, song metadata is sufficient and using in-depth song analysis or artist similarity is not required.

Part 3. Artist Terms and Tags

The artist terms and tags are contained in a SQLite database. The database has five tables. The tables artist_mbtag and and artist_term were queried and converted to a pandas dataframe. The resulting dataframes were merged on artist id and using a left merge approach. Missing values in the merged dataframe were filled using the pandas method fillna.

Part 4. User Listening Data

The user listening data text file was converted to a dataframe. . The dataframe read the several columns of data as one. Separating by delimiter was used to separate data into several columns. The result was a dataframe with user_id, song_id and play_count. A column called "play" with value 1 was added to indicate the user played the song. The resulting dataframe has ~48.4M entries.

Part 5. Song Mismatches List

There is known song mismatches list. This means songs that were labeled incorrectly and matched incorrectly when the dataset was created. The song mismatches text file was converted to a dataframe. The dataframe read the several columns of data as one. Separating by delimiter was used to separate data into several columns. As a result, a dataframe with four columns was created: song_id, track_id, song_name, and does_not_equal_to.

Part 6. Song Duplicates List

There is known song duplicates list. The song duplicate text file was converted to a dataframe. The dataframe read the several columns of data as one. The structure of this data column is a song name followed by track_ids below.

The method series.str.split was used to split the track_ids and the song sames. As a result of this a blank column was created, which was dropped. Two columns remained: track_id and song names. However, blanks were created in the track_id column because song names were previously there. These blanks were filled with NaN using replace method and np.nan. Further, the first five rows were dropped because they had dataset contact information and not actual duplicate information.

Moreover, the song names column has blanks cells below song names (i.e. they are parallel to track_ids). Using the pandas fillna with the forward fill method, song names were propagated forward and to its respective track_ids. Track ids with blanks rows were dropped, extrenous number digits in front of the song names were removed, and the index was reset.

This list is the result of finding 131,661 items of 53,471 song objects. This means that 53,471 are unique values to the overall song dataset. To take this into account, the pandas duplicated method was used to identify unique and duplicated values. Finally, the dataframe was filtered to include only values that were found to be duplicates and remove the ones that are unique.

Part 7. Concatenating Song Metadata

The Song Metadata Dataset, Analysis Metadata Dataset and Musicbrainz datasets were concatenated on the index to crerate one large song metadata dataframe. The resulting dataframe has 1M entries. These columns have missing values: artist_familiarity, artist_hotttnesss, artist_longitude, song_hotttnesss and year.

Part 8. Removing Song Mismatches from Song Metadata

The concatenated song metadata file was merged with the song mismatches dataframe using a left merge approach and on song_id. Due to the merge new rows and columns were created, including a track_id_y which came from the song mismatches list. The resulting dataframe have ~1.0004M entries.

The track_id_y blanks were replaced with the string ‘not mismatched’. This was done to indicate songs that did not merge with the mismatched items list. The dataframe was filtered by the ‘not mismatched’ string to remove mismatched items. The resulting dataframe has 981,022 entries.

The merge approach was used to filter duplicates rather than comparing the song_id series of each dataframe directly because each series’ length and indexes are different making the comparison challenging and not obvious.

Part 9. Using Duplicates List to Remove Duplicates

The dataframe frame with mismatches removed was merged with the song duplicates dataframe using a left merge approach and on track_id. Due to the merge new rows and columns were created, including a song_name_y which came from the song duplicates. The resulting dataframe have 981,022 entries.

The song_name_y blanks were replaced with the string ‘not duplicate’. This is to indicate songs that did not merged with the duplicates items list. The dataframe was filtered by the ‘not duplicate’ string to remove mismatched items. The resulting dataframe has 905,712 entries.

The merge approach was used to filter duplicates rather than comparing the track_id series of each dataframe directly because each series’ length and indexes are different making the comparison challenging and not obvious.

Part 10. Removing columns created during merges and filling blanks

In this step, columns created by the merged process were dropped. As well blank entries were replaced with ‘NaN’ using two approaches: fillna and replace. The latter created two blank columns: analyzer version and genre. These blank columns were removed.

Part 11. Converting Datasets into Tidy Format[2]

As part of the wrangling process three main dataframes were created: (1) song metadata, (2) user listening, and (3) artist tags. Song metadata dataframe was melted using song_id as the identifying variable. User listening dataframe was melted using user_id as the identifying variable (the song_id is part of the values). Artist tags and terms dataframe was melted using artist_id as the identifying variable.

Part 12. Output Datasets into CSV files

The song metadata, user listening data and artist tags were exported to a local CSV file using chunksize 500K and the mode append to speed up exporting.

[2] More info on tidy data here: https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html

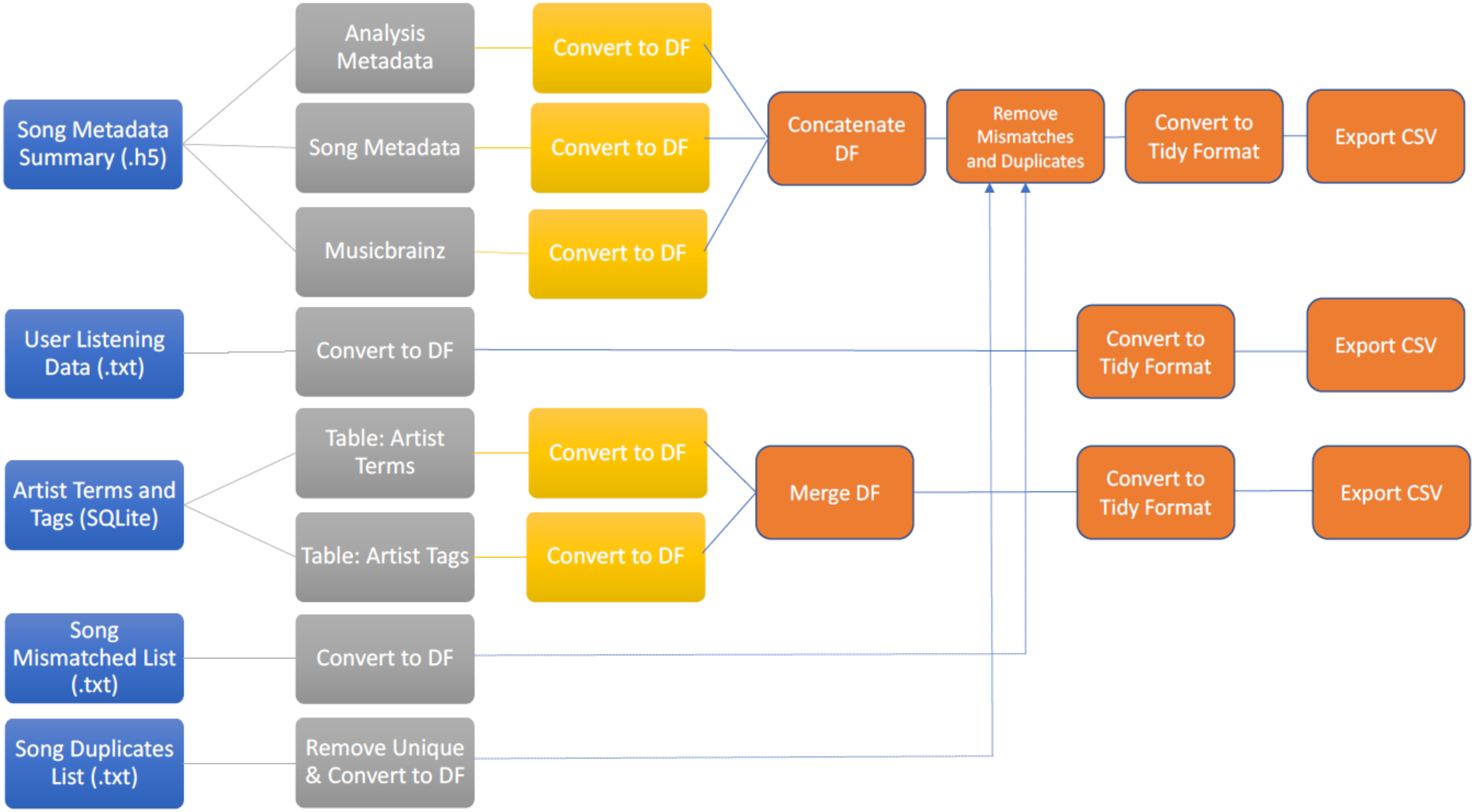

Figure 1. Summary of Data Wrangling Steps