**ETL Project**

Gideon Schultz, Martha Karran, Jenny Yi

**OVERVIEW**:

Extracting, cleaning, transforming, and unionizing Netflix and Hulu TV show data to see which shows are available on which streaming service(s). It would be interesting to use this database to see how many TV shows Netflix has versus Hulu, and what content differences there are by genre (e.g., Comedy, Drama, Documentaries, etc.) and TV rating (e.g. G, PG, Mature, etc.)

**EXTRACT:**

We extracted the following datasets in csv. formats:

1. A data set of 6,234 movies and TV shows available on Netflix as of 2019. The dataset was collected by Flixable, a third-party Netflix search engine.
   1. <https://www.kaggle.com/shivamb/netflix-shows>
2. A dataset of Hulu’s top 1,000 most popular TV shows up until 2017.
   1. <https://data.world/chasewillden/top-1-000-most-popular-hulu-shows>

**TRANSFORM:**

Some challenges with the data included filtering out the Netflix file as there were both movies and TV shows, and removing duplicate TV shows from the Hulu file. We also found a few (under 5) duplicate TV show names in the Netflix file, but saw through the cast and director data that the repeats were different versions of the same TV show (e.g. The Office from the U.S. and The Office from the U.K.)

Luckily, similar variables existed in both datasets, making it quick to identify the variables we wanted to include in our final dataset. We chose to extract the following columns from each dataset: “Show\_ID,” “Show\_Name,” “Genre,” “TV\_Rating,” and “Description.”

We reformatted the column names and used underscores instead of spaces to ensure consistency between the two datasets, being mindful of the order of the columns when we unionized the two datasets, which made it easier to upload into PostgreSQL later.

Transforming the Netflix CSV file:

1. Choosing the columns we wanted and renaming them:
   1. The following columns were extracted from the original netflix dataset: show\_id, title, listed\_in, rating, description.
   2. These columns were renamed “Show\_ID,” “Show\_Name,” “Genre,” “TV\_Rating,” and “Description,” respectively.
2. Since this data set included both movies and TV shows, we filtered out all 4,265 movies

Transforming the Hulu CSV file:

1. Transforming columns
   1. The following columns were extracted from the original Hulu dataset: show/id, show/name, show/genre, show/show\_rollups/subscriber/highest\_rating, show/description
   2. These columns were renamed “Show\_ID,” “Show\_Name,” “Genre,” “TV\_Rating,” and “Description,” respectively.
2. We sorted the new dataset alphabetically by “Show\_Name” to visualize any duplicates.
   1. To assess the duplicates further, we counted the number of instances of each show in the “Show\_Name” column.
   2. Using the drop\_duplicates function in pandas, we dropped duplicates, keeping the last instance of each.
   3. To confirm that duplicates were successfully removed, we counted the number of instances of each show again, looking for only one instance of each.

Unionizing and adding data to the combined dataframe:

We unionized the two dataframes as opposed to merging because we had already set up each dataframe to be identical. Unionizing meant we would be placing all the data on top of each other, as opposed to combining columns through merging.

Prior to the union, we created a new column in each dataframe that would capture whether each TV show was available on each streaming service (see below).

* netflix/hulu\_df\_2["Available on Netflix/Hulu"] = "Yes"

However, once we unionized both datasets, we realized all the empty rows of each competitor column contained “NaN,” so we also replaced all the “NaN” with “No” so that each cell in the availability columns contained either “Yes” or “No.”

**LOAD:**

1. We created a relational database in SQL because our merged CSV file is highly structured.
   1. This database contains the following headers:
      1. “Show\_ID”, Show\_Name”, “Genre”, “TV\_Rating”, “Description”, “Available\_on\_Hulu”, “Available\_on\_Netflix”
   2. We used the query tool to create the table framework in PgAdmin/Posgres
   3. Using python we converted and saved the dataframe “netflix\_hulu\_tvshows”
   4. We created a connection to posgres and loaded our data using df.to\_sql.
      1. We used “if\_exists=replace” because our table was already created and the default “if\_exists=fail” would see the table and stop the process.

Because we planned ahead and transformed this data, we can quickly conduct analysis for any future projects using this data as it would not need any further cleaning.