**The Top Ten List**

**Collaborators**

* Mike Lygas
* Smita Sharma
* Arjun Subramaniam
* Gretel Uptegrove

**Summary**

Our ETL project provides a pipeline for entertainment fans to find the top ten movies for a given year between 2008-2018 based on gross box office sales and music albums based on number sold. This information is then matched with review and rating information. For example, a single record will show you that the movie Black Panther from BV studio was ranked at #2 in 2018 based on box office sales with a user review average of 6.5 out of 10 based on 2829 reviews and a critic score of 88 out of 100 based on 55 reviews on [Metacritic](https://www.metacritic.com/). A timestamp reflecting when reviews were scraped is included.

**System Requirements**

* Chrome Web Browser
* Chromedriver
* MongoDB
* Python environment running Python 3.7 with the following installations:
  + beautifulsoup4
  + numpy
  + pandas
  + pymongo
  + requests
  + splinter
  + yaml

**Steps to run the pipeline:**

1. If running on MacOS, no configuration changes are needed. To run on Windows, change the chromedriver path in config.yml.
2. Run mongod to open a local connection.
3. Run top\_ten.py.
4. At the prompt, enter the year between 2008-2018 to scrape.
5. After finished executing, a mongo database for the provided year will be available on a local connection.

**Narrative**

We wanted to create a database that reflects entertainment in popular culture over the last decade. We focused on music and movies, as those forms of entertainment create large amounts of revenue, and we included rating information to show both popular and critical opinions on entertainment culture.

**How to use the data**

Each item is entered as a document in two different collections--movie and music album--in a mongo database. Each document in the collection includes year, rank in that year, title, studio name/artist name, user rating, number of user reviews, critic score, number of critical reviews, and the time the review data was scraped. If review information was not available, review values were populated with null values.

The reason we chose to use Mongo over SQL is for its faster performance features, such as the read/write scanning for handling data, and flexibility for adding additional data at a later date. MongoDB benefits include handling unstructured data and integrating with analytical tools such as Spark and BI.

This database can be used to see what sorts of entertainment are popular and profitable over time and how both users and critics feel about them.

**Data Sources**

We gathered movie ranking data from [BoxOfficeMojo](https://www.boxofficemojo.com/), which reports box-office revenue. It is affiliated with IMDb and is available for general public use.

We gathered music ranking data for albums from [billboard](https://www.billboard.com/), which reports popularity of music albums and songs based on sales.

For each movie and album, we gathered rating information from [Metacritic](https://www.metacritic.com/). We decided to use metacritic as it provides consistent review information across movies and music. It provides a platform for users to provide ratings from 0-10. It also aggregates critical reviews for and summarizes them into a single score, called a metascore, that ranges from 0-100. Information on how score is calculated can be viewed at [About Metascores](https://www.metacritic.com/about-metascores). Metacritic gathers data from multiple sources, which can be viewed at [Metacritic FAQ](https://www.metacritic.com/faq#item12).

**Schema**

top\_10\_db.movies:

{

\_id: int # Unique ObjectID assigned by mongodb

rank: int # Rank of movie in given year

title: string # Title of movie

studio: string # Production studio

year: int # Year of ranking

user\_rev\_count: int # Number of user reviews

user\_rev\_avg: float # Average user review (on scale 0-10)

critic\_rev\_count: int # Number of critical reviews

critic\_rev\_score:float # Critic score (on scale 0-100)

scrape\_time: string # Timestamp of metacritic scrape

}

top\_10\_db.albums:

{

\_id: int # Unique ObjectID assigned by mongodb

rank: int # Rank of album in given year

title: string # Title of album

artist: string # Album's artist

year: int # Year of ranking

user\_rev\_count: int # Number of user reviews

user\_rev\_avg: float # Average user review (on scale 0-10)

critic\_rev\_count: int # Number of critical reviews

critic\_rev\_score: float # Critic score (on scale 0-100)

scrape\_time: string # Timestamp of metacritic scrape

{

**Transformation Steps**

* We ran necessary pip installs.
* We created four separate jupyter notebooks to develop our python code and then converted each notebook into functions to be run by a single master script:
  + Movie scraping script (Arjun):
    - Used requests module to get content from boxofficemojo.com.
    - Used Beautiful Soup to parse content.
    - Cleaned data by stripping special characters.
    - Entered data into dataframe.
    - Added year column.
    - Dropped unnecessary columns and rows.
    - Converted dataframe into a list of dictionaries.
  + Album scraping script (Mike):
    - Used requests module to get content from Billboard.com.
    - Used Beautiful Soup to parse content.
    - Filtered out unnecessary rows based on rank.
    - Created a list of dictionaries.
  + Metacritic scraping script (Smita):
    - For each movie/album dictionary in the provided list of dictionaries from the scraping scripts:
      * Created a URL based on provided movie/album data.
      * Used Splinter to create a browser object for that URL to gather content from associated metacritic page.
      * Used Beautiful Soup to parse content.
      * If review information was unavailable, populated review variables with null values.
      * Created timestamp reflecting time page was scraped.
      * Added rating data to provided dictionary.
  + Master script (Gretel):
    - Made necessary configurations.
    - Created local connection to MongoDB.
    - Used pymongo to create a new top\_ten\_db mongo database and movies and albums collections.
    - Created user prompt for year.
    - Checked if information for provided year was already entered into database.
    - Invoked each script to gather data, resulting in two lists of dictionaries.
    - Inserted a document in associated collection from each dictionary in movies and albums
    - Inserted print statements with scraping progress.
  + YAML configuration file (Gretel):
    - Created configuration file specifying path used to connect to Chromedriver.
* After extensive testing, we exported the final script notebook into a single python script called top\_ten.py.

**Exception and Error Handling**

Error handling increases the robustness of our code by guarding against potential failures that would cause the program to exit in an uncontrolled fashion. The scraping script is written to execute with certain condition holding at run-time, but if these conditions do not hold, our scraping might not return and instead return an error. Since we are dealing with dynamic pages, they may change at any time, resulting in these errors.

We applied the following runtime exception handeling:

* IndexError and AttributeError: within the review scraping functions metacritic\_movie\_scraper and metacritic\_album\_scraper, this error is returned when the page we seek to parse does not exist or when html tags are not appropriately assigned.
  + When this exception occurs, we populate the following with null values only for that particular movie's/album's review and scores:
    - user\_rev\_count
    - critic\_rev\_count
    - user\_rev\_avg
    - critic\_rev\_score