

|  |
| --- |
| ETL PROJECT-DATA ENGINEERING |
|  |
| May 27  COMPANY NAME: AMRITA VISHWA VIDYAPEETHAM  Authored by: MAHALAKSHMI B |

# SUMMARY

|  |
| --- |
| OBJECTIVE The objective of this project was to extract data from websites and available APIs. The following datasets were then transformed by cleaning, joining, and filtering into nine tables. The object-relational database (PostgreSQL) was used to load the datasets into pgAdmin.  logo-placeholder |
| EXTRACT-TRANSFORM-LOAD |
| **Extract:**  The following Data Sources were used below:  **IMDb Website**  ● ***Method***: Web Scraping  ● ***Link***: <https://www.imdb.com/chart/top/?ref_=nv_mv_250>  ● ***Used for:*** Collecting the Top 250 IMDB rated movie list  ● ***Description:*** We used the url and pandas code to get the HTML table on the webpage. The data contains a list of top 250 imdb movie titles along with imdb rank and imdb rating.    **OMDb API**  ● ***Method:*** API Extraction  ● ***Link***: <http://www.omdbapi.com/>  ● ***Used for***: Collecting IMDb id and other movie related details like actor, director, etc. ● ***Description***: We used OMDB API to get the movie details by the title of the movie. The endpoint for the API took the title name and returned the data in JSON format. 250 requests were made to this API (one for each movie in the IMDb top 250 movie data) to get 250 json return files. We appended all the json files in one json file and used it to create a pandas dataframe containing the movie title, imdb id, year rating, actor, director, production, etc.    **Utelly API**  ● ***Method:*** API Extraction  ● ***Link:*** [https://rapidapi.com/utelly/api/utelly?endpoint=apiendpoint\_3cad787b-ca7b-449a -84b4-23b40d64fd73](https://rapidapi.com/utelly/api/utelly?endpoint=apiendpoint_3cad787b-ca7b-)  ● ***Used For:*** Collecting streaming options for Top 250 IMDb movies  ● ***Description:*** Utelly API is an API available on rapidapi.com. A personal account was created to get the api\_key required to request data from this API. The API had limits on the number of requests that we can make in a day. The endpoint used was GET/idlookup and it took the imdb\_id to return a JSON file for each movie. We used the imdb\_id obtained from the OMDb API. 250 requests were made using this API to obtain 250 json files, one for each movie. The json files contained the streaming options that are available for a particular movie. The returned json files were searched to look for the information that we needed and it was saved in a pandas dataframe. The information was collected and saved in such a way that we could get the dataframe in first normal form.    uNoGS API  ● ***Method:*** API Extraction  ● ***Link:*** <https://rapidapi.com/unogs/api/unogs/endpoints>  ● ***Used For:*** Collecting movies on Netflix in released in the United States which have an IMDb rating greater than or equal to 7  ● ***Description:*** uNoGS (unofficial Netflix online Global Search) allows anyone to search the global Netflix catalog. Netflix no longer provides an API for its data. So an alternative Netflix API was used. This API is available on rapidapi.com. A personal account had to be created on rapidapi.com to get the api\_key needed to extract data from this API. The API provides several API endpoints. Each of the endpoints were analysed using sample requests to figure out the API that could be most useful for the purpose of the project. The endpoint used was GET Advanced Search. This endpoint provided an option to build an advanced custom query. Search could be based on a number of different parameters including netflix country list, movie/ series type, genre, subtitles, audio, imdb rating, dates on which the movie became available on Netflix and some other parameters.  The parameters that we used for this search were  1. Movie or Series: We used to code only to get a list of movies  2. Country code: We used the code to get only the movies that are available in the United States (country code: 78 on API found using a different query)  3. IMDb rating: We used to code to get only the movies that had the IMDb rating between 7 and 10.  4. Movie year range was year 1900 to 2020 to get almost all the movies that satisfy the above conditions.  5. No filtering was done for genre, subtitles, audio, etc. The API has a limit to return only 100 results per page per request. The return results of the query returned around 925 total results. Hence the request was made 10 times (ie, to get 10 pages of result) to get all the results that satisfy all the conditions listed above. The data obtained was in json format. The data on all the pages was combined in a json file and was loaded in a pandas dataframe using pd.read\_json().    **Google Search Engine**  ● ***Method:*** Web Scraping  ● ***Link:*** A custom link was generated for each movie The titles of the movies (top 250 imdb movies) were converted to lowercase and the space between any two words in the movie title was replaced by ‘+’. Also the string ‘+watch+movie’ was appended at the end of the url to get the data that we need.  ● ***Used for:*** Collecting viewing Streaming Service availability and price.  ● ***Description:*** Beautiful Soup library was used to scrape the data using css class names. Google has a limitation on the number of requests that can be made per hour. It was found through ‘Google Search’ that this number is 8 requests per hour per IP address. There is a possibility that if the number of requests exceed 10, Google will block the IP address. So scraping the data involved a lot of effort in running the code in sets of 8, multiple times during the day. The data obtained was transformed in pandas data frame.    **Transform**  ● Data extracted were formatted in CSV and JSON files  ● The data formatted files were manipulated in pandas to clean, join, and filter nine tables.  **TABLES**  ***top\_imdb***    Transformation steps:  ● The data table obtained from the IMDb website link has IMDb rank, Movie title and Year of Release in a single column. These were converted in separate columns to get the list of movies. This list was then used as an input for the OMDb API.  ● Displayed just the IMDb ID and movie title from the OMDb API data  ● Saved the DataFrame to a csv file.  movie    Transformation steps: ● Displayed IMDb ID, movie title, year, release year, runtime, movie rating, IMDB rating, and production from the OMDb API data ● Reset the index to make the IMDB rank column ● Transformed the decimal significant figure from 1 to 0 ● Split the runtime string to only include the number ● Renamed columns of the runtime data frame to drop the unwanted string from runtime ● Renamed the columns of the movie data frame to include only meaningful names ● Combined runtime and movie data frame ● Displayed IMDb ID, movie title, year, release year, runtime, movie rating, IMDB rating, and production from movie data frame ● Saved the DataFrame to a csv file.  **Movie\_actor**    Transformation steps: ● Displayed IMDb ID, and actors from the OMDb API data ● Dropped any N/A values ● Creating an actor list to set up a conversion into a DataFrame ● Iterated through actors dataframe to clean up data into a dictionary ● Converted actor list into a Dataframe ● Saved the DataFrame to a csv file.  movie\_director    Transformation steps:  ● Displayed IMDb ID, and directors from the OMDb API data  ● Dropped any N/A values  ● Creating a director list to set up a conversion into a DataFrame  ● Iterated through directors dataframe to clean up data into a dictionary  ● Converted director list into a Dataframe  ● Saved the DataFrame to a csv file.  **Utelly\_streaming\_service**    Transformation steps:  ● Displayed streaming services from the Utelly API data  ● Renamed column header to have meaningful names  ● Filtered to see unique values in Streaming service column of dataframe to set up for a unique service id  ● Splitting service name column by splitting string  ● Grouped Streaming services  ● Renamed columns to meaningful names  ● Dropping unwanted column that did not include streaming name  ● Declared a list for unique service id column  ● Created a service id column and added to the service\_id list  ● Show service\_id and service\_name of the utelly\_streaming\_Service table  ● Saved the DataFrame to a csv file.  **google\_streaming\_service**    Transformation steps:  ● Displayed streaming services from the Google web scraped data  ● Renamed column header to have meaningful names  ● Filtered to see unique values in Streaming service column of dataframe to set up for a unique service id  ● Grouped Streaming services  ● Renamed columns to meaningful names  ● Dropping unwanted column that did not include streaming name  ● Declared a list for unique service id column  ● Created a service id column and added to the service\_id list  ● Show service\_id and service\_name of the google\_streaming\_service table  ● Saved the DataFrame to a csv file.  **Utelly\_movie\_streaming**    Transformation steps:  ● Displayed streaming services from the Utelly API data  ● Transformed streaming service column by splitting string  ● Dropped unwanted column that does not include important information for service dataframe  ● Renamed columns to meaningful names  ● Combined rename dataframe with service dataframe  ● Created a blank series for the service\_id column  ● Ran a for loop and assigning values to service\_id series  ● Inserted new column from the service\_id values  ● Showed service id and IMDB id of the utelly\_movie\_streaming table  ● Saved the DataFrame to a csv file  **Google\_movie\_streaming**    Transformation steps:  ● Displayed streaming services from the Google web scraped data  ● Renamed column header to have meaningful names  ● Filtered to see unique values in Streaming service column of dataframe to set up for a unique service id  ● Created a blank series for the service\_id column  ● Ran a for loop and assigning values to service\_id series  ● Inserted new column from the service\_id values  ● Showed movie title, google service id and price of the google\_movie\_streaming table ● Saved the DataFrame to a csv file.  **Netflix\_movie**    Transformation steps:  ● Displayed Netflix ID, IMDB ID, and movie title from the Rapid API data  ● Renames columns into meaningful names  ● Saved the DataFrame to a csv file  **Load**  ● The object-relational database (PostgreSQL) was used to load the datasets into pgAdmin  ● We selected a relational database as the data was in a structured format  ERD DIAGRAM    Sample Queries Query:  Joining movie table with google\_movie\_streaming table    Query:  Specific movie title, streaming service availability, price Query in google\_movie\_streaming table for movie (“Inception”), streaming service available, and price:    Thank you |