What’s trending in North America?

ETL Project on YouTube data



**Team Members:**

Carly Osborne | Kristen Mason | Rima Philip

Table of Contents

**Background**

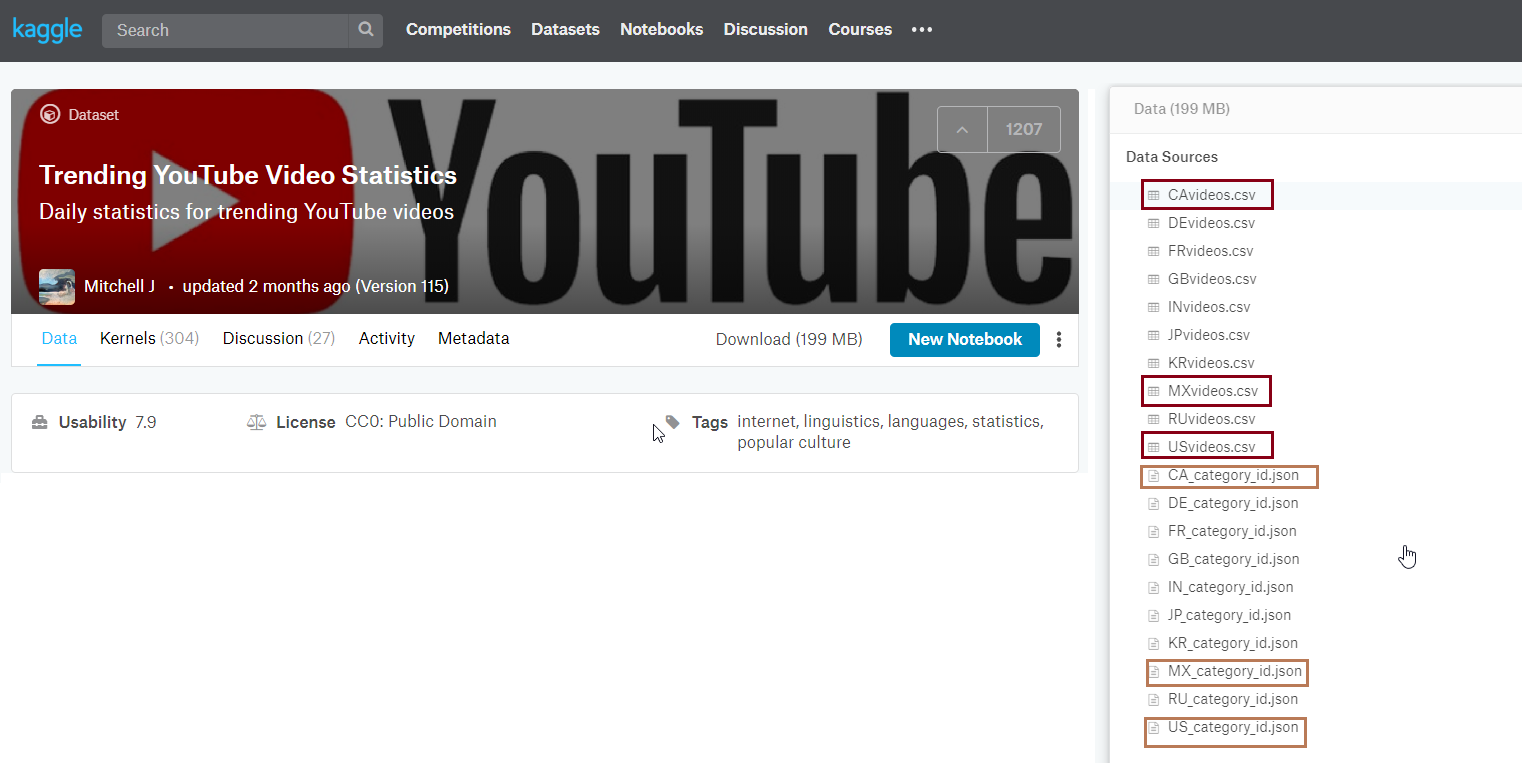
Our extraction process started with reviewing available datasets from kaggle.com. Our goal was to find datasets with large volume of records with relevant data which can be used for analysis.

We came across a collection of datasets on trending YouTube video statistics which had datasets for different countries in both csv and json formats. Each dataset had over 41k records and had a lot of data(columns) with some good potential for cleanup as well.

All these aligned with our goal for this project and we decided to proceed with this collection and to limit our analysis within North America, we picked the datasets for US, Canada and Mexico.

**Data Source**

<https://www.kaggle.com/datasnaek/youtube-new/data>



|  |  |
| --- | --- |
| **File Name** | **Type** |
| US\_videos | csv |
| CA\_videos | csv |
| MX\_videos | csv |
| US\_category\_id | json |

**Data Sets**



**Database Setup:**

Next step was to visualize a database setup with the available data and we chose the opensource relational database management system ‘Postgres’ to setup our database

We decided to set up the youtube\_db with the below tables.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Db Name** | **Table#** | **Table Name** | **Primary Key** | **Data Source** |
| youtube\_db | 1 | videos | video\_id | csv |
| 2 | country | country\_id | csv |
| 3 | popularity | id | csv |
| 4 | category | category\_id | json |

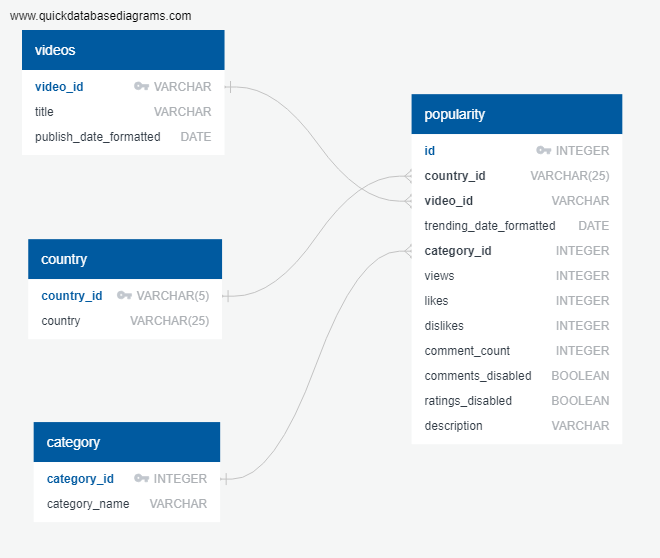
**Table Schema | ERD**

Using quickdatabasediagrams.com, we designed the below entity relationship diagram and derived the schema to setup the tables in the youtube\_db.

Schema SQL:



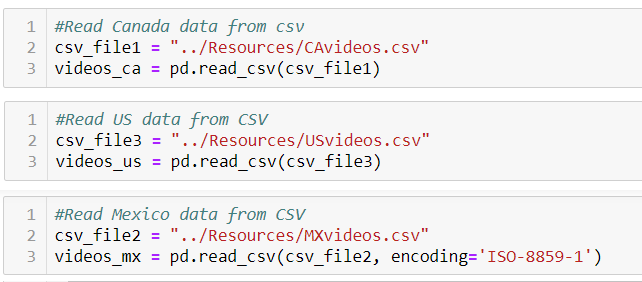
Link to schema: <https://app.quickdatabasediagrams.com/#/d/LHP3ZS>

****

**Extraction from CSV**

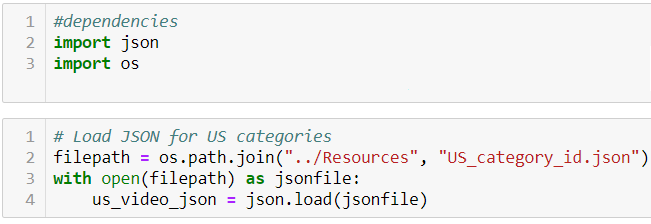
After retrieving the csv datasets from Kaggle.com, reading the file using python and jupyter notebook was the next step.

We used the *read\_csv()* function for this.



**Extraction from json**

After retrieving the json datasets from Kaggle.com, next step was to read the json file using python and jupyter notebook. The json files for all the three countries were the same (YouTube categories are the same for all countries), so we used only the US\_category\_id json file to extract all the categories.



**Transformation**

The transformation aspect dealt with reviewing the dataframes created by reading the csv files and the json file. The steps we executed to transform out datasets into dataframes which were ready to load into the tables are below.

1. **Concatenation** 
   1. Joined the three dataframes (US+CA+MX) into one (North America) and added a new column with country\_id to distinguish between each country.
   2. Used the function *concat()* to join the three datasets into one
2. **Retaining required columns alone**
   1. Dropped all unnecessary columns and retained only the required ones.
   2. Retained columns: "country\_id","country","video\_id","category\_id","trending\_date","title","publish\_time","likes","dislikes","views", "comment\_count", "comments\_disabled","ratings\_disabled","description"
3. **Removal of blank records**
   1. There were many records without valid video\_id which were dropped
4. **Formatting**
   1. Formatted date column data to standardize the date fields.
   2. Used to\_datetime() function to format the dates.
5. **Dataframes**

Created 4 different dataframes to align with our 4 tables

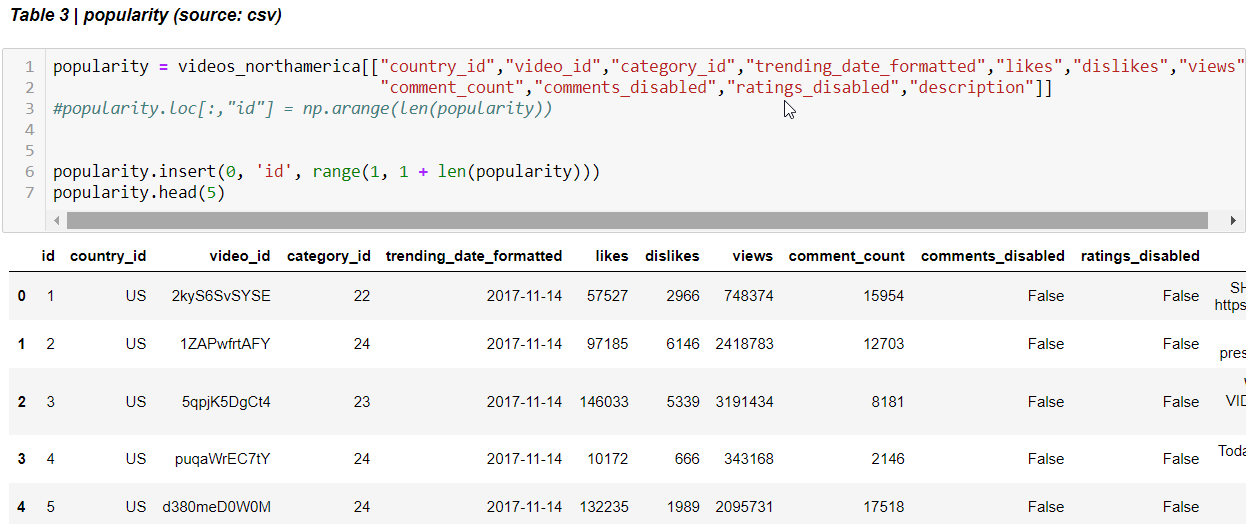
* 1. videos dataframe
     1. Removed the duplicates from the master dataset and picked video\_id, title and published\_date\_formatted fields to create the videos dataframe.



* 1. country dataframe
     1. Created country dataframe with country\_id and a new column country\_name



* 1. popularity dataframe
     1. The master dataframe with all North America data was retained as the popularity dataframe.
     2. This dataframe had the details like, trending\_date, views, likes, dislikes, comment\_count etc. apart from video\_id, country\_id and category\_id which were retained to build foreign key dependencies in the database between the tables.



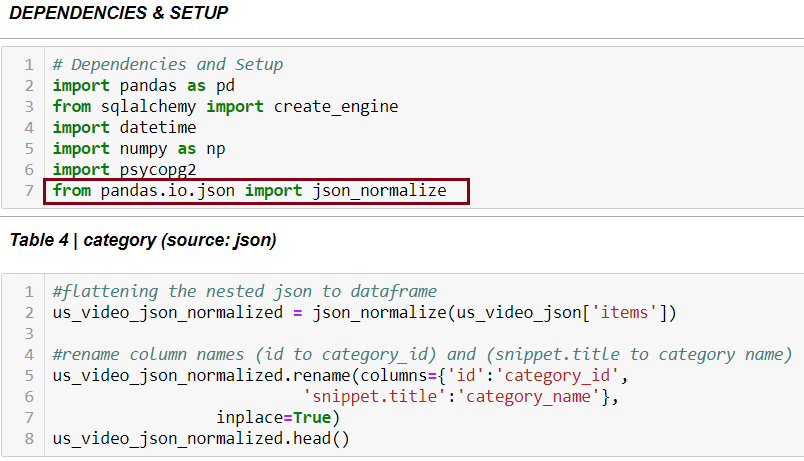
* 1. category dataframe
     1. This dataframe was created from the json file US\_category\_id and had all the YouTube video category information.
     2. category\_id and category\_name were the only fields extracted to create the category dataframe.



**Note:**

The process of extraction and transformation was slightly different for the dataframes created by reading the csv files and the json file.

* + **CSV** - Transforming the dataframes created by reading the csv file after cleanup was direct. It involved selection of the required columns for each table.
  + **Json** - Transforming dataframes extracted from the json file involved flattening the nested json into the dataframe.
  + *json\_normalize ()* function was used to retrieve data from the nested json file.



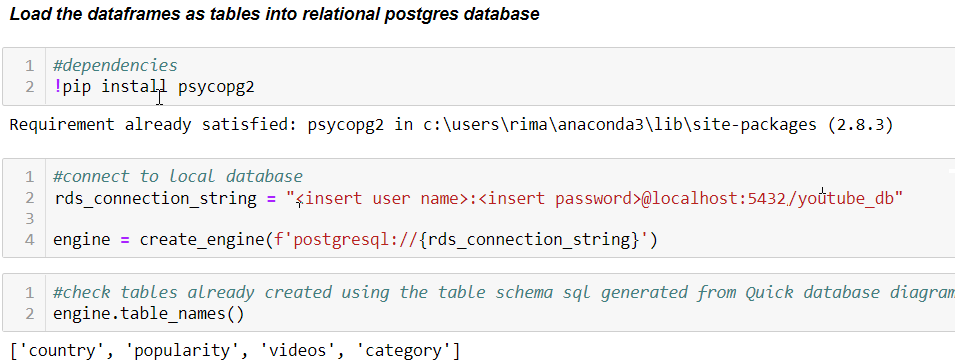
**Preconditions:**

* The youtube\_db database was setup in Postgres relational database management system.
* The table schema was derived using the Entity Relationship diagram created using quickdatabasediagram.com website.
* set up the dependencies and relationships between the tables.
* Executed the schema sql and created the 4 tables in youtube\_db with the respective columns matching the corresponding dataframes.
* More details are given on Page 2

**Loading Process**

Loading the tables involved two steps:

1. **Connecting to the local database (postgres)**
   * Installed Psycopg2 which is a DB API 2.0 compliant PostgreSQL driver. It is one of the most popular PostgreSQL database adapters for the Python programming language.
   * Created a connection string to pass the username and password for the postgres database.



1. **Loading the dataframe into corresponding tables.**
   * Used the to\_sql() function to load the records from the dataframes into the tables.

****

We verified that the data is loaded correctly by querying the databases.

