ETL Project

June 12th, 2021

—

Ándres Estrada

Roberto Miranda

Juan José Fernández

Ángeles Cruz

Rafael Deyta

Data Cleanup & Analysis

Business Requirement: Create an app to tell the final user where a movie is streaming on. And also, know the most important information about the movie, such as *director*, *genre* and other specifications.

1. **Extract:**

   Source 1

   The first dataset we selected on Kaggle.com has a large list of movies and indicates whether they are on Netflix, Hulu, Prime Video or Disney+, and the basic information such as Title, Year, Age and more, up to 17 columns.

   The format of the extracted file was on *csv*.

   https://www.kaggle.com/ruchi798/movies-on-netflix-prime-video-hulu-and-disney

   Source 2

   The second dataset we selected on Kaggle.com combines data from Netflix, Rotten Tomatoes, IMBD, posters, box office information, trailers on YouTube, and much more, up to 29 columns for series and movies.

   The format of the extracted file was on *csv*.

   https://www.kaggle.com/ashishgup/netflix-rotten-tomatoes-metacritic-imdb

   Both datasets were lacking information that could be found in the other, that's why we selected them and began our transformation process.

Disclaimer on Data Freshness. The Source 1 dataset was updated a year ago. The Source 2 dataset (Netflix) is updated constantly. The last update on Kaggle was in April 2021.

2. Transform:

2.1. A Jupyter Notebook called *Netflix_Extract_Transform_Load.ipynb* was created.

2.2. Dependencies indicated.

```
# Dependencies and libraries

import pandas as pd
import datetime as dt
from sqlalchemy import create_engine
import requests
import pymongo
from IPython.display import clear_output
```

2.3. With the function pd.read_csv we loaded the file *MoviesOnStreamingPlatforms.csv* and converted it to a dataframe called *movies_df*.

```
movies_file="MoviesOnStreamingPlatforms.csv"
movies_df=pd.read_csv(movies_file)
```

2.4. We explored the dataframe columns.

```
movies_df.columns
```

```
Index(['Unnamed: 0', 'ID', 'Title', 'Year', 'Age', 'IMDb', 'Rotten Tomatoes',
       'Netflix', 'Hulu', 'Prime Video', 'Disney+', 'Type', 'Directors',
       'Genres', 'Country', 'Language', 'Runtime'],
      dtype='object')
```

2.5. We created a copy of the dataframe called *new_movies_df*. This new dataframe contained the columns we were going to work, these were renamed too: *'Title'*, *'Year'*,*'Directors'*, *'Age'*,*'Genres'*, *'Language'*, *'Hulu'*, *'Prime Video'*, *'Disney+'* . The Netflix column was not selected as it was part of the plan to obtain that data from the second source.

```
new_movies_df=movies_df[['Title', 'Year','Directors', 'Age','Genres', 'Language','Hulu', 'Prime Video', 'Disney+']]
new_movies_df.rename(columns={'Title':'title','Year':'year','Genres':'genre','Language':'languages','Directors':'di
                              'Prime Video':'prime_video', 'Disney+':'disney_plus'},inplace=True)
new_movies_df.head()

```

	title	year	director	age	genre	languages	hulu	prime_video	disney_plus
0	Inception	2010	Christopher Nolan	13+	Action,Adventure,Sci-Fi,Thriller	English,Japanese,French	0	0	0
1	The Matrix	1999	Lana Wachowski,Lilly Wachowski	18+	Action,Sci-Fi	English	0	0	0
2	Avengers: Infinity War	2018	Anthony Russo,Joe Russo	13+	Action,Adventure,Sci-Fi	English	0	0	0
3	Back to the Future	1985	Robert Zemeckis	7+	Adventure,Comedy,Sci-Fi	English	0	0	0
4	The Good, the Bad and the Ugly	1966	Sergio Leone	18+	Western	Italian	0	1	0

2.6. A new column was generated to be the key to be prepared to join both dataframes. This new key was the concatenation of *title* and *year*. This value was passed as *string*.

```
new_movies_df['key']=new_movies_df['title']+"-"+new_movies_df['year'].astype('string')
```

2.7. With the function pd.read_csv we loaded the file *netflix-dataset.csv* and converted it to a dataframe called *netflix_df*.

```
netflix_file="netflix-dataset.csv"
netflix_df=pd.read_csv(netflix_file)
```

2.8. We explored the dataframe columns.

```
netflix_df.columns
```

```
Index(['Title', 'Genre', 'Tags', 'Languages', 'Series or Movie',
       'Hidden Gem Score', 'Country Availability', 'Runtime', 'Director',
       'Writer', 'Actors', 'View Rating', 'IMDb Score',
       'Rotten Tomatoes Score', 'Metacritic Score', 'Awards Received',
       'Awards Nominated For', 'Boxoffice', 'Release Date',
       'Netflix Release Date', 'Production House', 'Netflix Link', 'IMDb Link',
       'Summary', 'IMDb Votes', 'Image', 'Poster', 'TMDb Trailer',
       'Trailer Site'],
      dtype='object')
```

2.9. We created a filter selecting only the value *Movie*.

```
new_netflix_df = netflix_df[netflix_df['Series or Movie']=='Movie'].copy()
```

2.10. The columns we were going to work on were selected and renamed: *'Title'*, *'Genre'*, *'Languages'*, *'Director'*, *'Actors'*, *'IMDb Score'*, *'Release Date'*, *'IMDb Link'*, *'Poster'*,*'Summary'*

```

new_netflix_df=new_netflix_df[['Title', 'Genre', 'Languages', 'Director','Actors', 'IMDb Score',
                'Release Date', 'IMDb Link', 'Poster','Summary']]

new_netflix_df.rename(columns = {'Title':'title', 'Genre':'genre', 'Languages':'languages',
                                 'Director':'director','Actors':'actors', 'IMDb Score':'imdb_score',
                                 'Release Date':'release_date', 'IMDb Link':'imdb_link',
                                 'Summary':'summary', 'Poster':'poster'},inplace=True)


```

2.11. As in the *new_movies_df* was indicated with the value 1 when a movie was found in Hulu, Prime Video and Disney+, we created a new column *Netflix* with the value 1 for each row.

```
new_netflix_df['netflix'] = 1
```

2.12. The release_date column was not in the correct format. We used the function pd.to_datetime to convert it to a format with year, month and day. And afterwards we generated a new column *year* as an integer datatype.

```
new_netflix_df['release_date'] = pd.to_datetime(new_netflix_df['release_date'], format='%d %b %Y')
new_netflix_df.dropna(subset=['release_date'],inplace=True)
new_netflix_df['year']=new_netflix_df.loc[:,'release_date'].dt.year.astype(int)
```

```
new_netflix_df=new_netflix_df[['title','year', 'netflix', 'genre', 'languages', 'director', 'actors',
       'imdb_score', 'release_date', 'imdb_link', 'poster', 'summary']]
```

2.13. As in the first datasource, we created the key, we had to drop movies with the same title to create the same key in this dataframe. We identified the duplicated values and dropped them.

```
new_netflix_df = new_netflix_df.drop_duplicates(subset='title',keep='first')

print(f"There are {new_netflix_df['title'].duplicated().sum()} duplicate titles in the Netflix dataset")
```

```
There are 0 duplicate titles in the Netflix dataset
```

2.14. Then we created the key as a string.

```
new_netflix_df['key']=new_netflix_df['title']+"-"+new_netflix_df['year'].astype('string')
```

2.15. As both dataframes were cleaned with this process and both ready with the *key* to be merged, we used the function pd.merge to create the join and created a new dataframe: *new_bd*.

```
new_bd=pd.merge(new_netflix_df,new_movies_df, how='outer', on='key')
```

2.16. When merging both datasets, there were many NAN values on the title_x column and we filled them with the function *.fillna(0)*.

Then we created a new dataframe based on the *title_x* that had values (*temporal2*) and another one for the rows with *title_x* that were empty (*temporal*). We replaced the columns *title_x*, *year_x*, *director_x*, *genre_x*, *languages_x* with the columns that already contained this information.

As for the poster we indicated specific information, such as a NO IMAGE AVAILABLE url and for imdb_link we specified *"No info available"*. In the last column netflix, we indicated 0.

```
new_bd['title_x'] = new_bd['title_x'].fillna(0)

temporal2=new_bd.loc[new_bd['title_x']!=0,:].copy()
temporal=new_bd.loc[new_bd['title_x']==0,:].copy()

temporal['title_x']=temporal['title_y']
temporal['year_x']=temporal['year_y']
temporal['director_x']=temporal['director_y']
temporal['genre_x']=temporal['genre_y']
temporal['languages_x']=temporal['languages_y']
temporal['poster']='https://www.google.com/imgres?imgurl=https://www.athousakis.gr/images/usrImage/29-04-2021-10-25
temporal['imdb_link']='No info available'
temporal['netflix']=0
temporal.head()
```

2.17. After setting the format for the movies that were not on the column netflix, we were ready to add it to the dataframe temporal2. We used the function pd.concat and drop duplicates, in case they were already some of them. This created the final dataframe called *good_df*.

For the NAN values in the columns *hulu*, *prime_video*, *disney_plus*, we used the funcion *.fillna(0)* and also, *year_x* and *netflix* had to be converted as integers.

For the final step, the columns *title_x*, *year_x*, *genre_x*, *languages_x* and *director_x* were renamed, eliminating the *_x*.

```
good_df=pd.concat([temporal2,temporal])
good_df= good_df.drop_duplicates(subset='key',keep='first')

good_df=good_df[['title_x', 'year_x', 'genre_x', 'languages_x', 'director_x','actors','age',
       'netflix', 'hulu', 'prime_video', 'disney_plus','imdb_score', 'imdb_link', 'poster','summary']]
good_df['hulu'] = good_df['hulu'].fillna(0).astype(int)
good_df['prime_video'] = good_df['prime_video'].fillna(0).astype(int)
good_df['disney_plus'] = good_df['disney_plus'].fillna(0).astype(int)
good_df['year_x']=good_df['year_x'].astype(int)
good_df['netflix']=good_df['netflix'].astype(int)

good_df=good_df.rename(columns={'title_x':'title','year_x':'year','genre_x':'genre','languages_x':'languages','dire
good_df.head(2)
```

3. Load:

3.1. We chose a non - relational (**MongoDB**) to load our transformed data in order to be queried through an app by our users.

MongoDB was our selected mechanism for storage and retrieval of data because of its:

(1) *Simplicity of design*: not having to deal with the "impedance mismatch" between the object-oriented approach to write applications and the schema-based tables and rows of a relational database. In this case, storing all the movies information in one document as opposed to having to join many tables together, resulting in less code to write, debug, and maintain, and (2) *Flexibility* to easily and quickly accommodate any new type of data (schemaless database) allowing us to freely add fields to our JSON base as required.

For example, a movie record that may or may not contain fields like Age rating (not rated yet) or even having a duplicate name requiring a new field to differentiate (i.e. Joker as movie title corresponds either to the 2019 DC villain movie or the 2012 Hindu comedy hit).

3.2. We created our local connection to MongoDB and defined our database (*ETL_project*) and collection (*Movies*) .

Create Mongodb connection

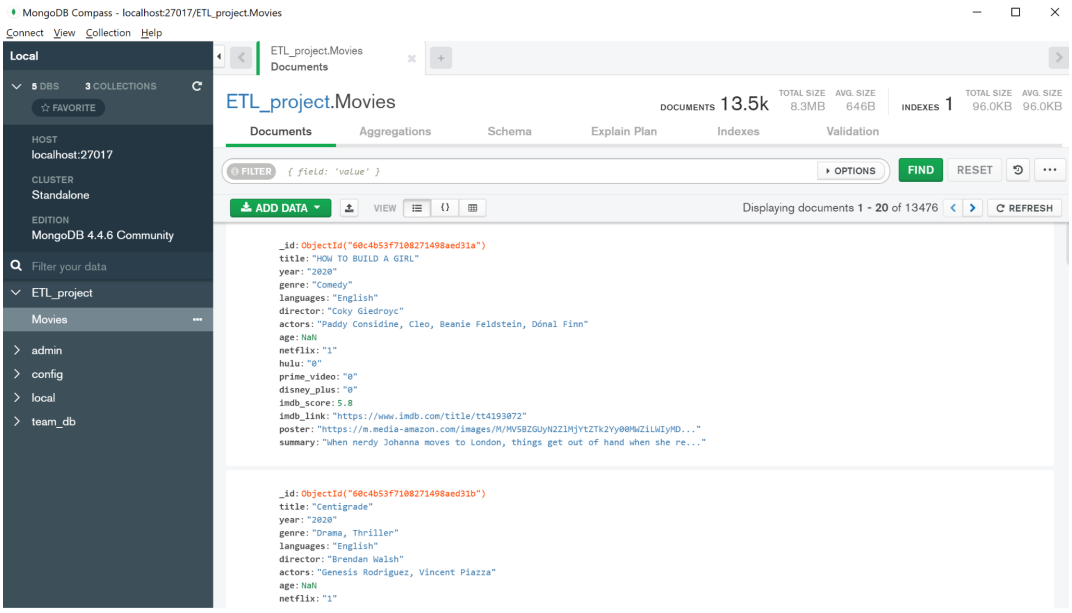
```
# Initialize PyMongo to work with MongoDBs
conn = 'mongodb://localhost:27017'
client = pymongo.MongoClient(conn)

# Define database and collection
db = client.ETL_project
collection = db.Movies
```

3.3. To load our data we performed a *For loop* to create a dictionary form database to simplify queries from our flask app. We also added a drop() instruction and a counter to ensure all of our data was getting through to MongoDB.

```
# Making sure database is empty
collection.drop()

# Creating the dictionary for Mongo
# We want title as object, hence the for loop

# Initializing variables and loop parameters
rows=len(good_df)
movies_dict={}
x=0 # This is just a counter to make sure loop parameters are correct

for row in range(0,rows):
    movies_dict={'title':good_df.iloc[row,:][0],
                 'year':(good_df.iloc[row,:][1]).astype(str),
                 'genre':good_df.iloc[row,:][2],
                  'languages':good_df.iloc[row,:][3],
                  'director':good_df.iloc[row,:][4],
                  'actors':good_df.iloc[row,:][5],
                  'age':good_df.iloc[row,:][6],
                  'netflix':good_df.iloc[row,:][7].astype(str),
                  'hulu':good_df.iloc[row,:][8].astype(str),
                  'prime_video':good_df.iloc[row,:][9].astype(str),
                  'disney_plus':good_df.iloc[row,:][10].astype(str),
                  'imdb_score':good_df.iloc[row,:][11],
                  'imdb_link':good_df.iloc[row,:][12],
                  'poster':good_df.iloc[row,:][13],
                  'summary':good_df.iloc[row,:][14]}
    x += 1

    # Saving dictionary to MongoDB
    collection.insert_one(movies_dict)

    # Progress counter
    clear_output(wait=True)
    print(f'Processing {x} of {rows}')
print(x)
```

```
Processing 9228 of 24681
```

3.4. We used MongoDB Compass to review that our loaded data is ready to be used on our flask app. Note that every entry has a different object id as a result of our load process using a *For loop*.



3.5. The database is now created on MongoDB. Now the App Development team will render this database on an API so final users can look for movies and check out on which streaming platform they can watch them.