OSCARS, RATINGS, AND FILMS- OH PY!

PROJECT TEAM: LAUREN PARRISH, ZEHRA TOKATLI, CHASE WEBB

Table of Contents:

Project Objective:

It seems like a bygone activity of yesteryear, but yes people did once go to movie theaters. With the advent of streaming services and, well Covid, most of us have been enjoying films from the comfort of our couch, but there are so many options! How do we know if what we select is “quality” and if something is “quality” is it something that we actually want to watch?

In this project, the team will perform ETL on data extracted from an Academy Awards Database and a Streaming Services database to capture information that would enable us to determine:

- Do award winning films actually have high user ratings?
- Do awards impact film popularity on streaming services?
- Do award winning directors receive higher ratings?
- Do films with winning soundtracks have higher popularity?

Extraction:

The first dataset was extracted from Kaggle in CSV format. This data included details on movie titles, user ratings, streaming services supporting the film, director, and release year, and genres.

The team then leveraged DataHub for the second dataset extraction in CSV format. This data set provided details year, award category, winner (T/F), and winner name.

A third data set was then leveraged from Kaggle to support joining of the Streaming and Awards data sets because there were more common data elements.

Datasets downloaded and imported to Python for cleaning.

- https://www.kaggle.com/ruchi798/movies-on-netflix-prime-video-hulu-and-disney
- https://datahub.io/rufuspollock/oscars-nominees-and-winners
- https://www.kaggle.com/unanimad/the-oscar-award

Transformation:

Summary of Streaming Data Cleanse:

- Drop empty columns;
- Remove rows that did not contain rating information;
- Split First and Second Director, drop any additional director information;
- Split Location cell into Primary and Secondary Location, drop other locations;
- Split Language into Primary and Secondary Language, drop other languages;
- Split Genres into eight separate columns:
- Convert streaming services cells to Booleans.

The greatest challenge in the Streaming dataset was that several cells contained multiple data points separated by commas. Some of these cells were not applicable for the team's analysis, and thus the decision was made to drop any additional data elements post the primary and secondary listing. The genres information was the exception. This cell was split and added to eight new genre columns.

```
#split the genres into seperate genre columns
ordered_df[['First Genre', 'Second Genre', 'Third Genre', 'Fourth Genre', 'Fifth Genre', 'Sixth Genre', 'Seventh Genre', 'Eigth Genr
e']] = ordered_df.Genres.str.split(",", expand=True)
ordered_df
```

```
#Delete Genres column and confirm order of remaining columns
del ordered_df['Genres']
ordered_df.columns
```

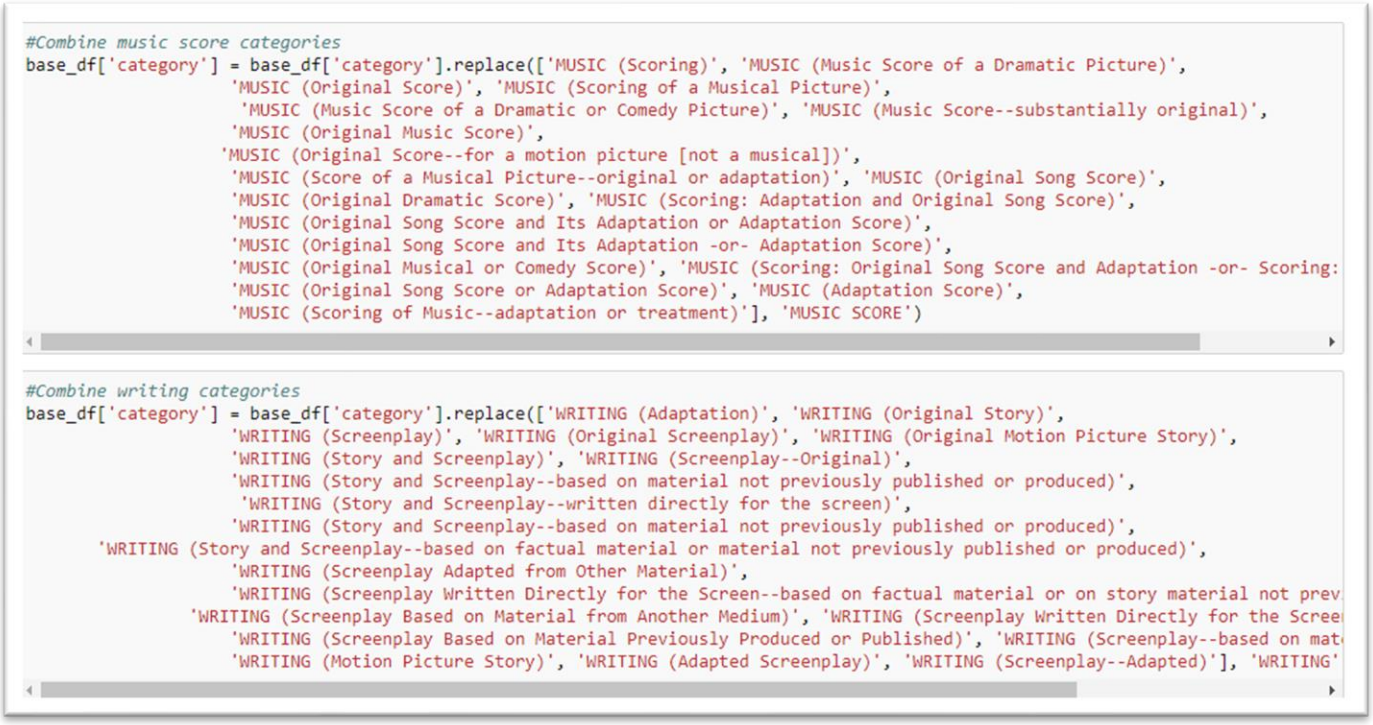
```
Index(['ID', 'Title', 'Year', 'Age', 'IMDb', 'Rotten Tomatoes', 'Netflix',
       'Hulu', 'Prime Video', 'Disney+', 'First Director', 'Second Director',
       'Primary Location', 'Secondary Location', 'Primary Language',
       'Secondary Language', 'Runtime', 'First Genre', 'Second Genre',
       'Third Genre', 'Fourth Genre', 'Fifth Genre', 'Sixth Genre',
       'Seventh Genre', 'Eigth Genre'],
      dtype='object')
```

```
streams_df = ordered_df.astype({"Netflix":bool, "Hulu":bool, "Prime Video":bool, "Disney+":bool})
streams_df
```

Awards Data Cleanse:

- Identify unique fields;
- Combine data with similar music categories;
- Combine data with similar short film categories;
- Combine data with similarities for cinematography, art direction;
- Identify winning actors;
- Remove “year_ceremony” and “ceremony” columns;
- Identify winning best pictures.

This data set had the opposite challenge where there were several columns containing similar categories. This level of detail was not required for the team’s assessment and therefore the data was merged.



```
#Combine music score categories
base_df['category'] = base_df['category'].replace(['MUSIC (Scoring)', 'MUSIC (Music Score of a Dramatic Picture)',
                    'MUSIC (Original Score)', 'MUSIC (Scoring of a Musical Picture)',
                     'MUSIC (Music Score of a Dramatic or Comedy Picture)', 'MUSIC (Music Score--substantially original)',
                    'MUSIC (Original Music Score)',
                   'MUSIC (Original Score--for a motion picture [not a musical])',
                    'MUSIC (Score of a Musical Picture--original or adaptation)', 'MUSIC (Original Song Score)',
                    'MUSIC (Original Dramatic Score)', 'MUSIC (Scoring: Adaptation and Original Song Score)',
                    'MUSIC (Original Song Score and Its Adaptation or Adaptation Score)',
                    'MUSIC (Original Song Score and Its Adaptation -or- Adaptation Score)',
                    'MUSIC (Original Musical or Comedy Score)', 'MUSIC (Scoring: Original Song Score and Adaptation -or- Scoring:
                    'MUSIC (Original Song Score or Adaptation Score)', 'MUSIC (Adaptation Score)',
                    'MUSIC (Scoring of Music--adaptation or treatment)'], 'MUSIC SCORE')
```

```
#Combine writing categories
base_df['category'] = base_df['category'].replace(['WRITING (Adaptation)', 'WRITING (Original Story)',
                    'WRITING (Screenplay)', 'WRITING (Original Screenplay)', 'WRITING (Original Motion Picture Story)',
                    'WRITING (Story and Screenplay)', 'WRITING (Screenplay--Original)',
                    'WRITING (Story and Screenplay--based on material not previously published or produced)',
                     'WRITING (Story and Screenplay--written directly for the screen)',
                    'WRITING (Story and Screenplay--based on material not previously published or produced)',
       'WRITING (Story and Screenplay--based on factual material or material not previously published or produced)',
                    'WRITING (Screenplay Adapted from Other Material)',
                    'WRITING (Screenplay Written Directly for the Screen--based on factual material or on story material not prev
                'WRITING (Screenplay Based on Material from Another Medium)', 'WRITING (Screenplay Written Directly for the Scree
                    'WRITING (Screenplay Based on Material Previously Produced or Published)', 'WRITING (Screenplay--based on mat
                    'WRITING (Motion Picture Story)', 'WRITING (Adapted Screenplay)', 'WRITING (Screenplay--Adapted)'], 'WRITING'
```

Data Joins:

As we progressed through the initial dataset, we found that it would be difficult to join to the other dataset, so a new source file was used which listed film titles. The new data set contained the Academy Award Winners plus the film name for which they were nominated. This was critical as the data could now be joined on film name. Fortunately, the new data set contained similar headers to the original data set so limited changes were required to the cleaning code.

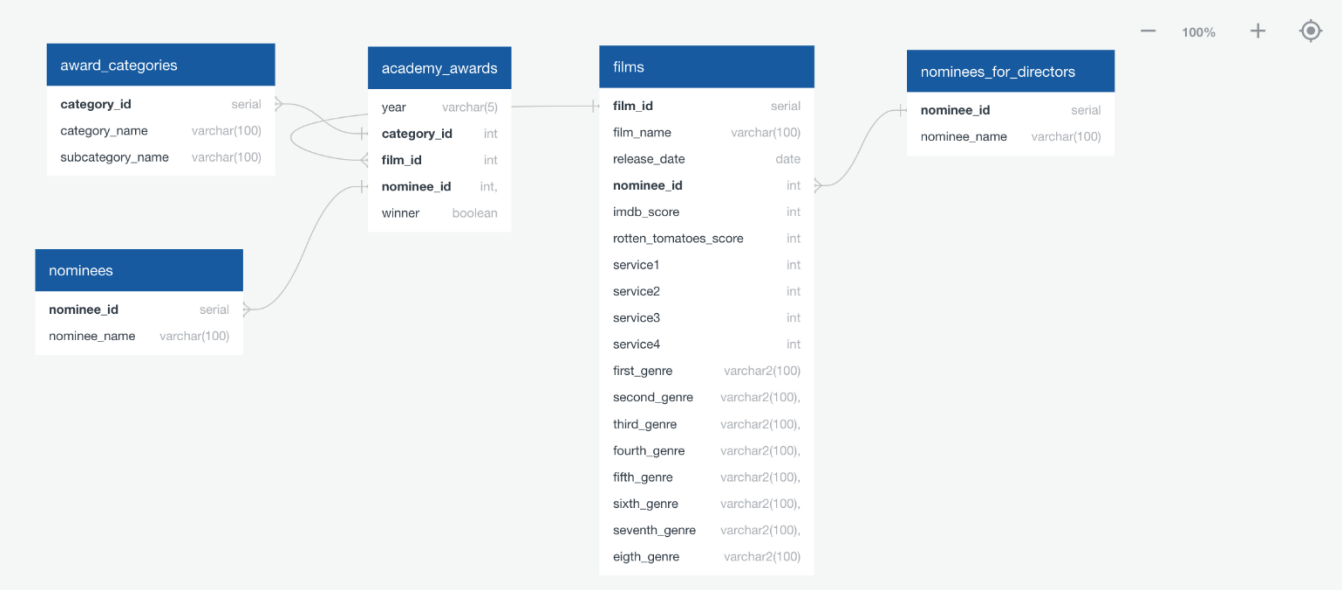
Third Data source link:

https://www.kaggle.com/unanimad/the-oscar-award

ERD Construction:

The introduction of the third data set with film names as a common identifier allowed for the construction of an ERD to assist relational database table design.

The ERD supported the construction of five tables which were created in PGAdmin. Primary and Foreign keys established for category_id, nominee_id, film_id.



The first three table were constructed leveraging the Awards data set. The awards_categories table was created to isolate the award types themselves. The nominees table was created to isolate those nominated. These two tables were then leveraged in the creation of the academy_awards table which combined the date while also adding the film id. The film id being the join to the Streaming data set.

The films table captured all the data from the streaming data set with corresponding keys, film_id and nominee_id. The nominees_for_director table was created as director was another element shared in both data sets yet not sufficiently unique to be its own key.

These tables would allow an individual to perform the analysis required to answer the team's original questions by merging and calling the data as required.

Examples of this analysis at a high level would include:

- Do award winning films actually have high user ratings?
    - Merge Academy Awards and Films data on Film_ID
    - Compare ratings for the award winners
- Do awards impact film popularity on streaming services?
    - Using above merge, sort ratings by descending order.
    - Analyze how many are award winners
- Do award winning directors receive higher ratings?
    - Merge Academy Awards and Nominees for Director tables
    - Group the films by those directors and analyze average ratings
- Do films with winning soundtracks have higher popularity?
    - Merge Academy Awards and Films data on Film_ID
    - Filter by category score, compare winners vs. non-winners.

Loading:

The data was loaded into a relational database because the team wanted to be able to make comparisons between the two data sources using commonalities such as film name. The team approached data loading by enhancing the original cleaning code to include loading elements.

```
In [2]: # import dbconnection files
        from config import conn_str
        from dboperations import connect2db, insertvalues
```

```
In [3]: conn = connect2db(conn_str)
        Connection Succesful
```

The original code was then updated to ensure consistency with the PGAdmin tables by renaming headers where appropriate.

In [18]:
```
#award_categories prepare df
category_df = pd.DataFrame({'category_name' : base_df['category'].unique()})

category_df = category_df.reset_index()
category_df['index'] = category_df['index']  + 1
category_df['subcategory_name'] = ""
category_df['description'] = ""

category_df = category_df.rename(columns={'index': 'category_id'})

category_df.head()
```

Out[18]:

	category_id	category_name	subcategory_name	description
0	1	ACTOR		
1	2	ACTRESS		
2	3	ART DIRECTION		
3	4	CINEMATOGRAPHY		
4	5	DIRECTING		

In [19]:
```
#award_categories insert df to table
insertvalues(conn, "award_categories", category_df)
```

```
Values Succesfully Insert To award_categories
```

Indexes were also applied to align to the database tables.

In [24]:
```
#nominees prepare df
nominees_df = pd.concat([actors, actress, directing], ignore_index=True)
nominees_df = pd.DataFrame({'nominee_name': nominees_df['entity']})
nominees_df.head()
```

Out[24]:

	nominee_name
0	Emil Jannings
1	Warner Baxter
2	George Arliss
3	Lionel Barrymore
4	Wallace Beery

In [25]:
```
#nominees prepare df
nominees_df = nominees_df.reset_index()
nominees_df = nominees_df.rename(columns={'index': 'nominee_id'})
nominees_df.head()
```

Out[25]:

	nominee_id	nominee_name
0	0	Emil Jannings
1	1	Warner Baxter
2	2	George Arliss
3	3	Lionel Barrymore
4	4	Wallace Beery

In [27]:
```
##award_categories insert df to table
insertvalues(conn, "nominees", nominees_df)
```

```
Values Succesfully Insert To nominees
```

Additional Python code was also created to assist in data insert.

```
def insertvalues(conn, table, values):

    try:
        values.to_sql(name = table, con = conn, if_exists = 'append', index = False)
    except Exception as error:
        print("Error While Insert Data: " + error)
    print("Values Succesfully Insert To " + table)
```

The combination of these coding elements culminated in successful data insert into the database tables.

```
select * from award_categories
```

Messages | Query History | Explain | Data Output | Notifications

	category_id bigint	category_name text	subcategory_name text	description text
1	1	ACTOR		
2	2	ACTRESS		
3	3	ART DIRECTION		
4	4	CINEMATOGRAPHY		
5	5	DIRECTING		
6	6	ENGINEERING EFFECTS		
7	7	OUTSTANDING PICTURE		
8	8	UNIQUE AND ARTISTIC PICTURE		
9	9	WRITING		

Lessons Learnt:

The team identified several take-aways from the project:

- Data rarely comes in a Goldilocks form even when it looks clean to start. It can have too much information, too little, or just be nuanced enough to cause problems.
- Common data fields are critical when constructing relational databases. Strings can be used, but unique numeric identifiers are preferred.
- The detail required in the Pandas coding library to export data frames to the database tables can cause numerous errors and delays. The team witnessed numerous issues with indexing and primary/foreign keys.
- Don’t create code errors that duplicate your original data set from 5,000 rows to 15,000. You’ll lose an hour just trying to run it.