DATA SCIENCE REPORT

STAGE 4

INTEGRATING MATCHED TUPLES AND PERFORMING ANALYSIS

<u>**Team Members**</u>

1. **Saranya Baskaran**
2. **Shivanee Nagarajan**
3. **Varun Ramesh**

For project stage 4, we have integrated tuples that matched in Table A and Table B to form Table E. Then we created tables in a star schema fashion in Postgres DB and performed OLAP style exploration to extract interesting information.

1. How did we combine Table A and Table B

We wrote a python script to merge the tuples which was predicted as a match in the previous match tables.

Merging Methods:

Name, Year, Directors, Actors:

For all these attributes, we always chose the values from A table (IMDB). This is because the IMDB website is more reliable source of information and maintains standard across all the other website on the internet.

Rating:

The E table movie rating is obtained by averaging the rating values from A table and B table. In IMBDB the rating is given out of 5 whereas in Filmcrave rating is given out of 10 for a movie. Hence, we convert the IMDB ratings to a scale of 10 and then take the average of these two to get the final rating.

Eg: IMDB - 3.7 Filmcrave - 7.6
then E table will have
Rating = (Filmcrave * 2.5 + imdb)/2
= 8.425

Genre:

Movie Genre is obtained by taking union of genres from both the tables.

For example,
IMDB - Horror/Drama
FilmCrave - Thriller, Drama, Horror
E table - Horror, Drama, Thriller

Other Tables added:

We obtained table E by merging A and B tables. We did not use any other table to populate E table.

Issues:

While merging data we hit into the issue of duplicate matches.

For Example,
Dark Night and The Dark Night were a match
Dark Night Part 1 and The Dark Night were a match

We should add IronMan1 to the table E only once. To avoid such duplicate entries of the same tuple in the final merged table we maintained a map of all merged tuples in the python script. Before adding any new tuple into the merged csv, it will be checked if it is already present in the final map. This way we avoid duplicate entries of same movies in Table E.

2. Statistics on Table E

- **Schema of Table E**

Attribute Name	Attribute Type	Description
Name	String	Title of the movie
Rating	Float	Overall rating of the movie
Year	Integer	Year of Release
Genres	String	Genres of the film
Directors	String	Directors of the movie
Actors	String	Stars of the movie

- **Number of tuples in E**

Table E has 1022 merged tuples after removing all duplicate matches.

- **<u>Sample tuples from Table E</u>**

Name	Rating	Year	Genres	Directors	Actors
The Conjuring 2	7.31	2016	Mystery, Horror, Thriller	James Wan	Vera Farmiga,Patrick Wilson,Madison Wolfe,Frances O'Connor
Sausage Party	6.30	2016	Comedy, Animation, Adventure	Greg Tiernan, Conrad Vernon	Seth Rogen, Kristein Wilig, Jonah Hill, Alistair Abell
Happy End	7.2	2017	Drama	Michael Haneka	Isabelle Huppert, Jean-Louis, Mathew, Fantine Harduin
La La Land	7.93	2016	Music, Drama, comedy	Damien Chazelle	Ryan Gosling, Emma stone, Rosemarie DeWitt, J.K Simmons

3. Data Analysis Task

For the data analysis task, we did OLAP style exploration on the final merged table E. We created a star schema with movie table as a fact table and different dimension tables such as Directors, Actors and Genres. Each of the dimension table have the movie id as a foreign key. Following is the schema of the star and fact tables.

Star Schema for OLAP Exploration:

Actors
A_M_Id (Foreign Key)
A_id
A_Name

Movie Table
M_id
M_Name
M_Rating
M_Year
M_Genres
M_Directors
M_Actors

Directors
D_M_id
D_id
D_Name

Genres
G_M_id
G_id
G_Name

Observation 1: Slice for the top 5 directors, actors and genres based on rating

Query to select top 5 directors based on rating:

Select d_name from directors where d_m_id in (Select m_id from movies ORDER BY m_rating DESC) limit 5;

```
     d_name
-------------------
 Chan-wook Park
 Damien Chazelle
 Xavier Dolan
 Christopher Nolan
 Quentin Tarantino
(5 rows)
```

Query to select top 5 actors based on movie rating:

Select a_name from actors where a_m_id in (Select m_id from movies ORDER BY m_rating DESC) group by a_name having count(*) > 1 limit 5;

```
     a_name
--------------------
 Tom Hardy
 J.K. Simmons
 DiCaprio
 Brie Larson
 Matt Damon
```

Query to select the genre which has most of the highly rated movies:

Select g_name from generes where g_m_id in (Select m_id from movies ORDER BY m_rating DESC) limit 5;

```
g_name
---------
 Crime
 Romance
 Mystery
 Drama
 Music
(5 rows)
```

Observation 2: If a highly rated pair of director and actor combination creates a highly rated movies always?

Dice:

Select movies based on the best actor and director to see if the combination always produces movies with at least a minimum threshold of movie rating. Since we have the list of top 5 best directors and actors, we can choose a combination and check if that pair always produces highly rated movies.

Example: Combination of a good director and good actor gives a movie with higher rating

```
myproject=# Select m_name,m_rating,  m_directors, m_actors from movies where m_directors like '%Nolan%' and m_actors li
;
  m_name    | m_rating |    m_directors     |                              m_actors
------------+----------+--------------------+-----------------------------------------------------------------
 Inception |     8.54 | Christopher Nolan | Leonardo DiCaprio,Joseph Gordon-Levitt,Ellen Page,Ken Watanabe
(1 row)
```

Similarly, a combination of not so good director and actor gives a poorly rated movie

```
myproject=# Select m_name,m_rating,  m_directors, m_actors from movies where m_directors like '%Brett%' and m_actors like '%Johnson%'
;
  m_name  | m_rating | m_directors  |                      m_actors
----------+----------+--------------+-----------------------------------------------------
 Hercules |     6.19 | Brett Ratner | Dwayne Johnson,John Hurt,Ian McShane,Joseph Fiennes
(1 row)
```

Observtion 3: Identify the best genre and best actor or director in that particular genre.

Roll up:

We first aggregate on rating and then find which genre of movies have the highest rating.

Select avg(m_rating) as avg_rating, m_genres from movies GROUP BY M_GENRES ORDER BY avg_rating DESC limit 1;

```
    avg_rating     |                 m_genres
--------------------+--------------------------------------------
 8.2400000000000000 | Crime,Romance,Mystery,Drama
```

Drill down:

Now we can drill down based on genre dimension. For example, we can see that in genre crime who is the best director.

select avg(m_rating) as avg_rating, m_directors from movies where m_id in (Select g_m_id from GENERES where g_name = 'Crime') group by m_directors having count(m_id) > 1 order by avg_rating desc limit 1;

```
    avg_rating     |  m_directors
--------------------+-----------------
```

8.2350000000000000 | Martin Scorsese

(1 row)

Similarly, we found best directors and actors for each genre that had scored the highest rating.

4. Learnings/Conclusion

- From the data analysis, we see that mostly a popular combination of movie director and actor when working together will create a highly rated movie. So, we can infer that when the building blocks of the product is good, the end product will also be good.
- However, most movies of a highly rated director or actor is also highly rated.

5. Future work

- **Build a recommender system:** Based on the final merged data we wanted to build a recommender system were people could search for an actor, director combination and there will be movie suggestions according to their searches. Most of our data analysis part was leading towards inferring such information which will form the backend of the system and a good web UI using react framework would help getting movie suggestions based on actors, directors, genre would be lot easier.
- **Search based on movie plot**: In stage 3 we had extracted an additional column called ‘Plot’ which describes the movie plot. During the entity merging stage we had removed that column from final table as it is difficult to find a candidate match for a long text field. In future, based on such summary of movies, we would like to give movie suggestions for users.

Appendix: Python Scripts

data_merger.py

```
import pandas as pd
from merger_methods import merger, nomerge
from populate_foreigntables import *

movie_data_columns = ['movie_name', 'movie_rating', 'movie_year', 'movie_genres',
            'movie_directors', 'movie_actors']

genre_data_columns = ['movie_id','genre_id', 'genre_name']
director_data_columns = ['movie_id', 'director_id', 'director_name']
actor_data_columns = ['movie_id', 'actor_id', 'actor_name']

candidate_dataframe = pd.read_csv('../data/Candidate_Matches.csv')
predicted_dataframe = pd.read_csv('../data/Predicted.csv')

# Getting index range
possible_matches = len(predicted_dataframe[
                (predicted_dataframe['predicted'] == 1)])

print('Total tuple combinations: ',len(predicted_dataframe))

# id's for tables
id_count = 0
director_id = 0
actor_id = 0
genre_id = 0

# Map to maintain list of already added movies
added_movies = []
added_directors = []
added_genres = []
added_actors = []

# Create new dataframes
movies = pd.DataFrame(columns=movie_data_columns)
genres = pd.DataFrame(columns=genre_data_columns)
directors = pd.DataFrame(columns=director_data_columns)
```

```
actors = pd.DataFrame(columns=actor_data_columns)


# Iterate through all the predictions_dataframe
for i, row in predicted_dataframe.iterrows():

    # Check if both ltable and rtable id not already added
    l_id = int(row['ltable_id'])
    r_id = int(row['rtable_id'])

    merged_movie = []
    if int(row['predicted']) == 1:
        # Check if one of the ids is already in added_movies
        if l_id not in added_movies and r_id not in added_movies:
            # call merger for l_id and r_id
            merged_movie = merger(candidate_dataframe, l_id, r_id)
            added_movies.append(l_id)
            added_movies.append(r_id)

    if merged_movie:
        # Append to movies dataframe
        movies = movies.append({
        'movie_name':merged_movie[0],
        'movie_rating':merged_movie[1],
        'movie_year':merged_movie[2],
        'movie_genres':merged_movie[3],
        'movie_directors':merged_movie[4],
        'movie_actors':merged_movie[5]
        }, ignore_index=True)

        genres, genre_id = populate_genre_table(genres, id_count, genre_id,
                    merged_movie[3], added_genres)

        directors, director_id = populate_director_table(directors, id_count,
                        director_id, merged_movie[4], added_directors)
```

```python
        actors, actor_id = populate_actor_table(actors, id_count, actor_id,
                                               merged_movie[5], added_actors)

        id_count += 1

movies = movies[pd.notnull(movies['movie_name'])]
genres = genres[pd.notnull(genres['genre_name'])]
directors = directors[pd.notnull(directors['director_name'])]
actors = actors[pd.notnull(actors['actor_name'])]


print('Total number of movies ', len(movies))
print('Total number of genres ', genre_id)
print('Total number of directors ', director_id)
print('Total number of actors ', actor_id)


movies.to_csv('../Data/movie_table.csv')
genres.to_csv('../Data/genre_table.csv')
directors.to_csv('../Data/director_table.csv')
actors.to_csv('../Data/actor_table.csv')
```

Merger_methods.py

```python
def merger(candidate_dataframe, l_id, r_id):

    r_index = candidate_dataframe.index[candidate_dataframe['rtable_id'] == r_id]
    l_index = candidate_dataframe.index[candidate_dataframe['ltable_id'] == l_id]

    # adding movie title - Always take imdb movie title
    movie_name = candidate_dataframe.loc[r_index]['rtable_Title'].values[0]

    # Get average rating - (Filmcrave * 2.5 + imdb)/2
    filmcrave_rating_series = candidate_dataframe.loc[l_index]['ltable_Overall Rating'].values[0]
    imdb_rating_series = candidate_dataframe.loc[r_index]['rtable_Overall Rating'].values[0]
```

```
# Some cleaning stuff
filmcrave_rating_list = str(filmcrave_rating_series).split('/')
filmcrave_rating = float(filmcrave_rating_list[0])
imdb_rating = float(imdb_rating_series)
average_rating = (filmcrave_rating * 2.5 + imdb_rating) / 2

# Get movie year. Always take IMDB year.
movie_year_series = candidate_dataframe.loc[r_index]['rtable_Year'].values[0]
movie_year = int(movie_year_series)

# Get Movie Genres - Take union of both genres
filmcrave_genre_series = candidate_dataframe.loc[l_index]['ltable_Genre'].values[0]
imdb_genre_series = candidate_dataframe.loc[r_index]['rtable_Genre'].values[0]

# Getting individual genre fields
filmcrave_genres = str(filmcrave_genre_series).split('/')
imdb_genres = str(imdb_genre_series).split(',')
genre_set = set()

# cleaning genre and getting union
for val in imdb_genres:
    val = str(val).strip()
    genre_set.add(val)

for val in filmcrave_genres:
    val = str(val).strip()
    genre_set.add(val)

movie_genre_list = list(genre_set)
movie_genres = ','.join(movie_genre_list)

# Get movie directors - Get from IMDB more reliable
imdb_director_series = candidate_dataframe.loc[r_index]['rtable_Directors'].values[0]
movie_directors = str(imdb_director_series)

# Get movie actors - Get from IMDB more reliable
```

```
imdb_actor_series = candidate_dataframe.loc[r_index]['rtable_Actors'].values[0]
movie_actors = str(imdb_actor_series)

result = [movie_name, average_rating, movie_year, movie_genres,
        movie_directors, movie_actors]

return result
```