ECE356 Project

Movie Data

Student Name: Zixuan Wang, Wenzhe Zhao
Student ID: 21111769, 21111897
Group Number: 07

Table of Contents

Project Description

Our project's objective was to build a sizable movie database where visitors could view details about their favorite movies and various stars and other crew members of their favorite movies. In order to accomplish this, we combined the information we obtained from Kaggle about TMDB and IMDB to build an extensive database of films, actors, and crew. Users can access the database and make queries to search for movies or people in our client application. The query is sent to the database based on the information we have on movies or people. Then, the app generates an information page. What is more, the client application allows users to rate movies and stores the records in the database.

We made the following assumptions about the user behavior and our data:

(1) Assume users only search by movie title, person name, or keyword.
(2) Assume the data is accurate.

Entity Relationship Design

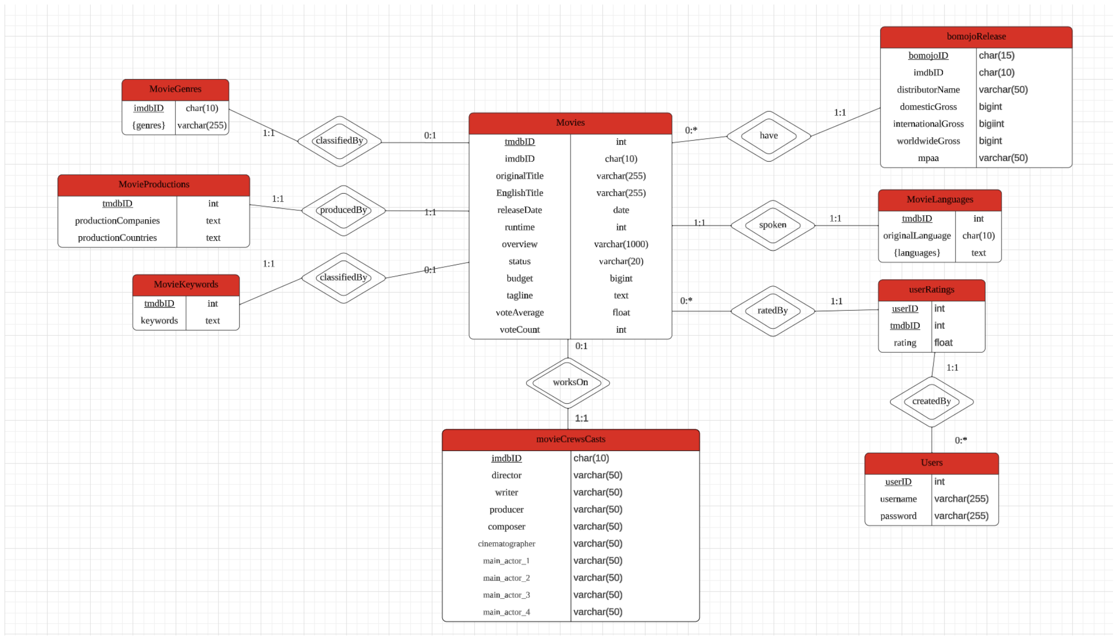


Figure 1. ER Diagram

We looked closely at the CSVs that Kaggle provided for us in order to figure out how to incorporate as much project-related data as possible into our ER architecture while still being able to integrate it. Here is the explanation of the ER diagram.

1. The relationships between movies are all weak entity sets. This is because, in a movie dataset, most entities have no meaning if the movie does not exist. Without movies, it does not make sense to say the genres of the movie have any meaning. The same applies to production companies and countries, movie keywords, movie crews, movie casts, movie information on ‘bomojo’, and movie language. For the ‘Users’ entity and ‘userRatings’ entity, if there is no user, the ratings of users also do not exist at all.

2. The Movies entity is the most important entity in the diagram. It has the tmdbID as the primary key and has unique imdbID values. As you can see, there are two attributes about the title of the movie: originalTitle and EnglishTitle. The difference is original title is written in multiple languages, and the English title is the translation of the original title, which is only written in English. This provides various choices for global users to search for the title of a movie.

3. The reason why we consider the genres as a new entity instead of an attribute of movies is that genre is multi-valued and complicated. Some movies have two genres, but others may have four or none. So, to simplify the movie entity and make the genres of a movie clearer, we create a separate entity for the movie genres. The same reason also applies to movie production, keywords, and languages.

4. The entity ‘bomojoRelease’ includes information about master data of movie releases from BoxOfficeMojo.com. It uses the bomojoID as its primary key and imdbID as the foreign key that references the imdbID in movies.

5. The entity ‘userRating’ is used to record users’ ratings of the movies. Each user will be assigned a userID, which is used as the primary key in entity Users.

6. Alternatives we considered:

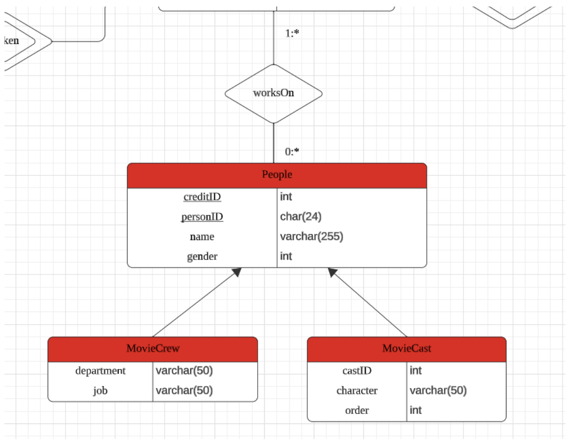
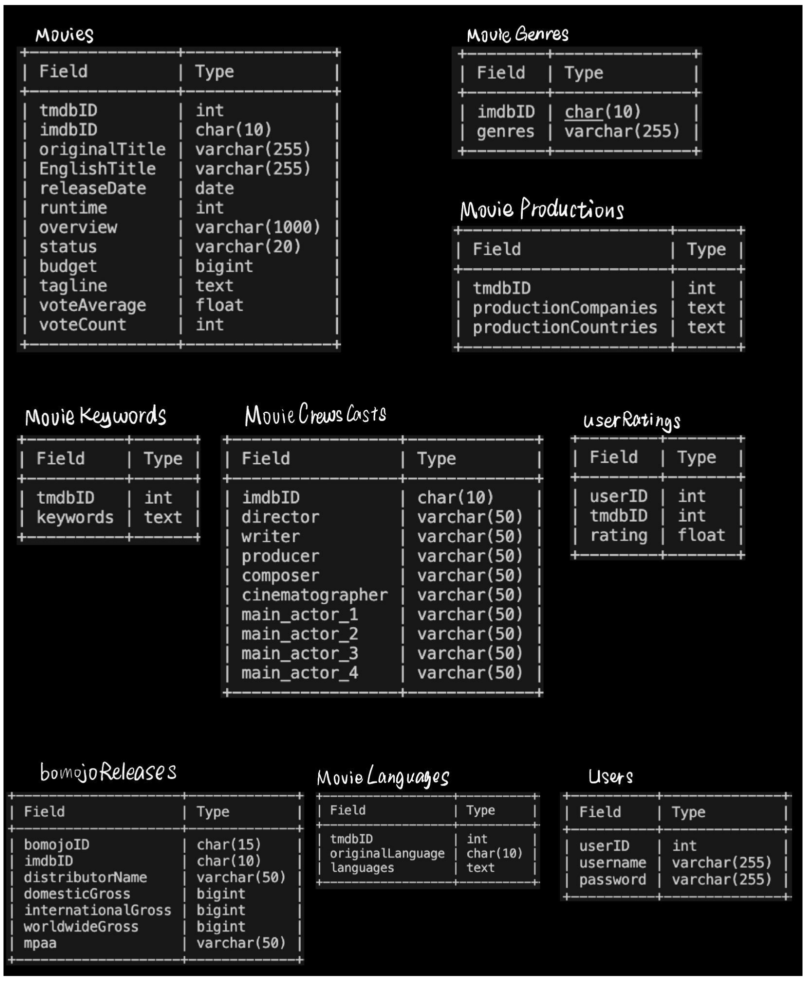


Figure 2. Alternative ER model design

Why we chose the design that we did rather than the alternative:
At first, we designed the crew and cast entity like the diagram above, which has a total overlapping specialization. However, when we investigated the dataset further, we found not all crews have the personID, and not all casts have creditID. As a result, the primary key of People is invalid. After discussing, we decided to use the imdbID as the primary key and list all the jobs as attributes. This entity was clearer and simpler, so we chose to use it in the final design.

7. Final Relational Schema Tables:


Movies

```
+----------------+----------------+
| Field          | Type           |
+----------------+----------------+
| tmdbID         | int            |
| imdbID         | char(10)       |
| originalTitle  | varchar(255)   |
| EnglishTitle   | varchar(255)   |
| releaseDate    | date           |
| runtime        | int            |
| overview       | varchar(1000)  |
| status         | varchar(20)    |
| budget         | bigint         |
| tagline        | text           |
| voteAverage    | float          |
| voteCount      | int            |
+----------------+----------------+
```

Movie Genres

```
+--------+--------------+
| Field  | Type         |
+--------+--------------+
| imdbID | char(10)     |
| genres | varchar(255) |
+--------+--------------+
```

Movie Productions

```
+---------------------+------+
| Field               | Type |
+---------------------+------+
| tmdbID              | int  |
| productionCompanies | text |
| productionCountries | text |
+---------------------+------+
```

Movie Keywords

```
+----------+------+
| Field    | Type |
+----------+------+
| tmdbID   | int  |
| keywords | text |
+----------+------+
```

Movie Crews Casts

```
+-----------------+-------------+
| Field           | Type        |
+-----------------+-------------+
| imdbID          | char(10)    |
| director        | varchar(50) |
| writer          | varchar(50) |
| producer        | varchar(50) |
| composer        | varchar(50) |
| cinematographer | varchar(50) |
| main_actor_1    | varchar(50) |
| main_actor_2    | varchar(50) |
| main_actor_3    | varchar(50) |
| main_actor_4    | varchar(50) |
+-----------------+-------------+
```

userRatings

```
+--------+-------+
| Field  | Type  |
+--------+-------+
| userID | int   |
| tmdbID | int   |
| rating | float |
+--------+-------+
```

bomojoReleases

```
+--------------------+-------------+
| Field              | Type        |
+--------------------+-------------+
| bomojoID           | char(15)    |
| imdbID             | char(10)    |
| distributorName    | varchar(50) |
| domesticGross      | bigint      |
| internationalGross | bigint      |
| worldwideGross     | bigint      |
| mpaa               | varchar(50) |
+--------------------+-------------+
```

Movie Languages

```
+------------------+----------+
| Field            | Type     |
+------------------+----------+
| tmdbID           | int      |
| originalLanguage | char(10) |
| languages        | text     |
+------------------+----------+
```

Users

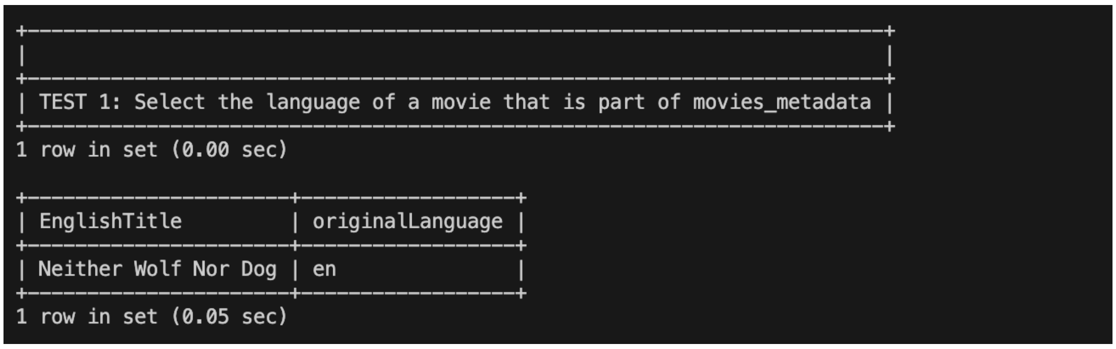
```
+----------+--------------+
| Field    | Type         |
+----------+--------------+
| userID   | int          |
| username | varchar(255) |
| password | varchar(255) |
+----------+--------------+
```

Figure 3. Relational Schema Tables

8. Data type choosing: Attributes like imdbID and bomojoID have fixed lengths, so we chose char as their datatype. As for titles and overviews, their lengths are not fixed. Some titles are very long, while others may be short. So, we chose varchar as their data type in order to save space.

Testcases

We created 5 test cases to guarantee basic Select commands, basic unique items, and basic foreign key constraints.
The first test is to select the language of a movie that is part of movies_metadata.csv. The expected output should return a row that has the title of the movie in the first column and the language in the second column. This is shown in Figure 4 below.

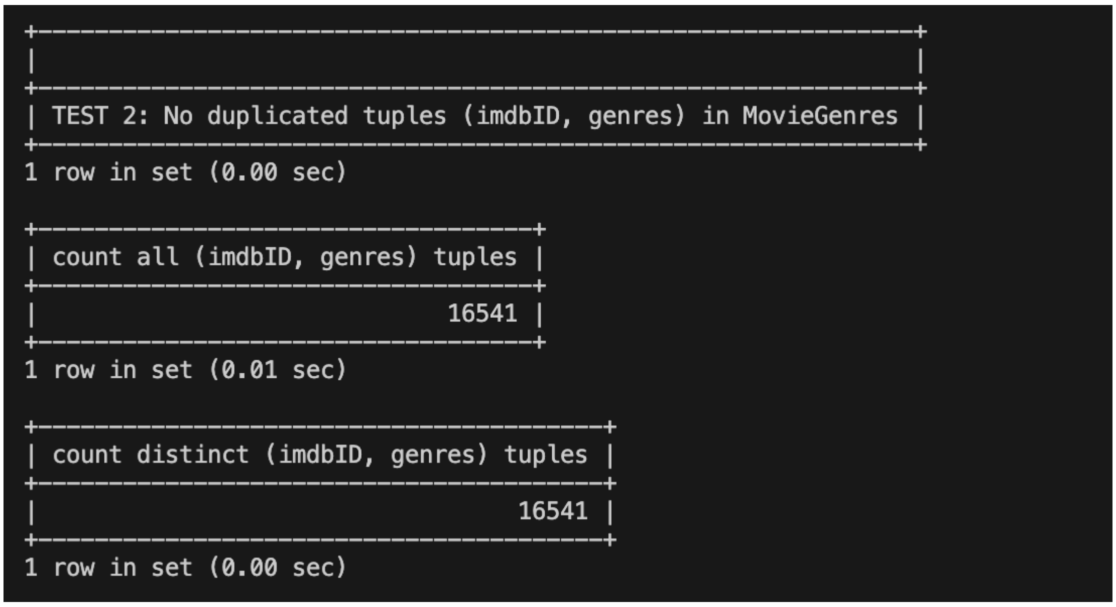


```
+---------------------------------------------------------------------+
|                                                                     |
+---------------------------------------------------------------------+
| TEST 1: Select the language of a movie that is part of movies_metadata |
+---------------------------------------------------------------------+
1 row in set (0.00 sec)

+----------------------+------------------+
| EnglishTitle         | originalLanguage |
+----------------------+------------------+
| Neither Wolf Nor Dog | en               |
+----------------------+------------------+
1 row in set (0.05 sec)
```

Figure 4: Test 1

The second test is to make sure there are no duplicated tuples (imdbID, genres) in MovieGenres. The expected output should be that the Counts results are the same for the two commands, as all duplicated entries should be deleted due to the primary key imdbID. This is shown in Figure 5 below.



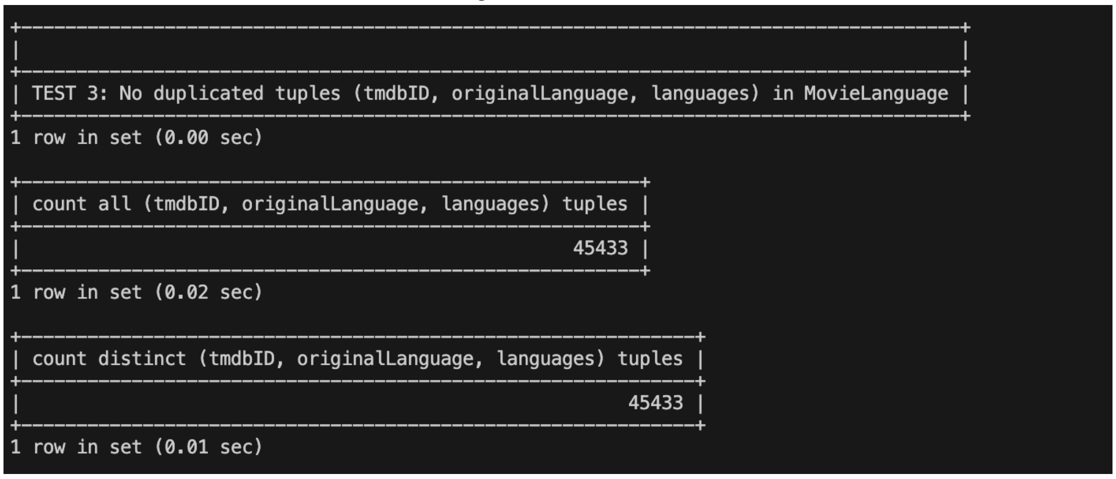
```
+-----------------------------------------------------------------+
|                                                                 |
+-----------------------------------------------------------------+
| TEST 2: No duplicated tuples (imdbID, genres) in MovieGenres |
+-----------------------------------------------------------------+
1 row in set (0.00 sec)

+-----------------------------------+
| count all (imdbID, genres) tuples |
+-----------------------------------+
|                             16541 |
+-----------------------------------+
1 row in set (0.01 sec)

+----------------------------------------+
| count distinct (imdbID, genres) tuples |
+----------------------------------------+
|                                  16541 |
+----------------------------------------+
1 row in set (0.00 sec)
```

Figure 5: Test 2

The third test is the same as test 2. The only difference is that we test MovieLanguages this time. The test result is shown in Figure 6:



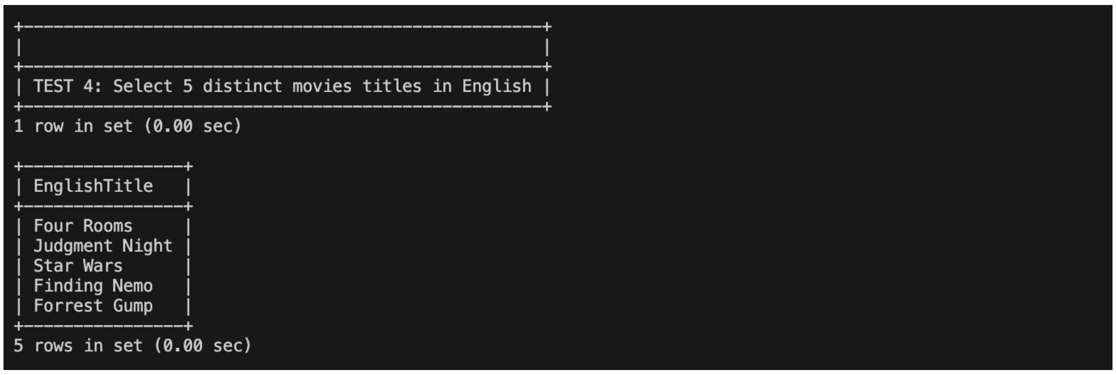
```
+---------------------------------------------------------------------------------------+
|                                                                                       |
+---------------------------------------------------------------------------------------+
| TEST 3: No duplicated tuples (tmdbID, originalLanguage, languages) in MovieLanguage |
+---------------------------------------------------------------------------------------+
1 row in set (0.00 sec)

+---------------------------------------------------------+
| count all (tmdbID, originalLanguage, languages) tuples |
+---------------------------------------------------------+
|                                                   45433 |
+---------------------------------------------------------+
1 row in set (0.02 sec)

+--------------------------------------------------------------+
| count distinct (tmdbID, originalLanguage, languages) tuples |
+--------------------------------------------------------------+
|                                                        45433 |
+--------------------------------------------------------------+
1 row in set (0.01 sec)
```

Figure 6: Test 3

The fourth test is to select 5 distinct movies titles in English. Actually, there are 11320 movies in English. But in order to make the interface clean and clear, we only chose five of them.
The output is shown below in Figure 7.

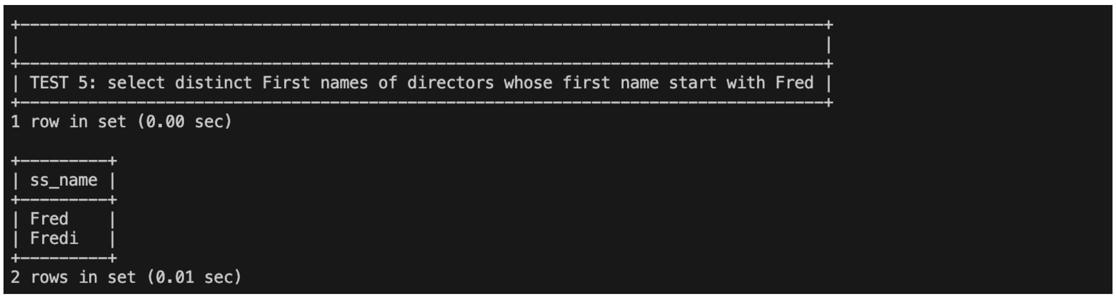


```
+-----------------------------------------------------+
|                                                     |
+-----------------------------------------------------+
| TEST 4: Select 5 distinct movies titles in English |
+-----------------------------------------------------+
1 row in set (0.00 sec)

+----------------+
| EnglishTitle   |
+----------------+
| Four Rooms     |
| Judgment Night |
| Star Wars      |
| Finding Nemo   |
| Forrest Gump   |
+----------------+
5 rows in set (0.00 sec)
```

Figure 7: Test 4

The last test case is about selecting distinct first names of directors whose first name start with ‘Fred’. The output in Figure 8 is what we expected.



```
+--------------------------------------------------------------------------------+
|                                                                                |
+--------------------------------------------------------------------------------+
| TEST 5: select distinct First names of directors whose first name start with Fred |
+--------------------------------------------------------------------------------+
1 row in set (0.00 sec)

+---------+
| ss_name |
+---------+
| Fred    |
| Fredi   |
+---------+
2 rows in set (0.01 sec)
```

Figure 8: Test 5

Client Application

For our client application, we chose to develop a simple Python-based command line program to allow users to view information about movies and the people who worked on them. Additionally, users can register an account so they can add their own reviews to the movies they love and hate. Based on our preliminary application design, we were able to implement most of the desired features.

Functionalities

After starting the program, the user will be greeted by a menu. For all the options, the program will check whether the input is valid, and we also let the user decide the maximum number of rows of the output query in case the matching result is of a huge number. Moreover, all the queries used the “LIKE” keyword in MySql; thus, similar matching results will also be returned. We also imported the tabulate library to pretty-print the output

The basic functionalities could be divided as follows:

1. Search for movies

After selecting to search for movies, the user can further specify which keyword the search is based on. We offer a total of three options: To Search based on the English title of the movie, the director of the movie, and the actor involved. The output will contain the basic information about the movie, plus the search keywords. For example, if the search is based on the English title, the English title, tmdbID, year, and vote average scraped from The Movies Dataset will be returned. If the search is based on the director, in addition to the information above, the output will also contain the director's name. Lastly, if we search using the actor, the output will contain four main actors in this movie, plus the basic information, as above.

2. Create/Modify/Remove the review

We offer the functionality for the user to make their own review of the movie. The program will first check whether the user has logged in. After checking that the user has done so, to create or modify a review, it will ask for the tmdbID of the movie together with the rating. To delete a review, the user only needs to enter the tmdbID. We would check whether the query is successful. If not successful, a logging warning will be printed on the console.

3. Add new movies

Inside the database, we have already registered an Administrator account, which is the only account that can add new movies. The program will first check whether the user is Admin. After that, it will ask the user to input tmdbID and the English Title to create the new movie record. We would check whether the query is successful. If not successful, a logging warning will be printed on the console.

4. Sign up/ Login

We need the user to input the username and password in order to sign up for a new account or log in. After logging in, the userID, which is an auto-increment type in our database, will be saved inside the program for the review functionality mentioned above. We would check whether the query is successful. If not successful, a logging warning will be printed on the console.

Reflect:

Ideally, the client should be like a website containing frontend and backend, which is our initial design. However, both two of our team members have no previous experience in writing React code. We spent a lot of time testing our frontend code after implementing the backend in Express successfully. However, we still have no solution as the deadline is approaching. Finally, we decided to rewrite all of our code from scratch in Python.

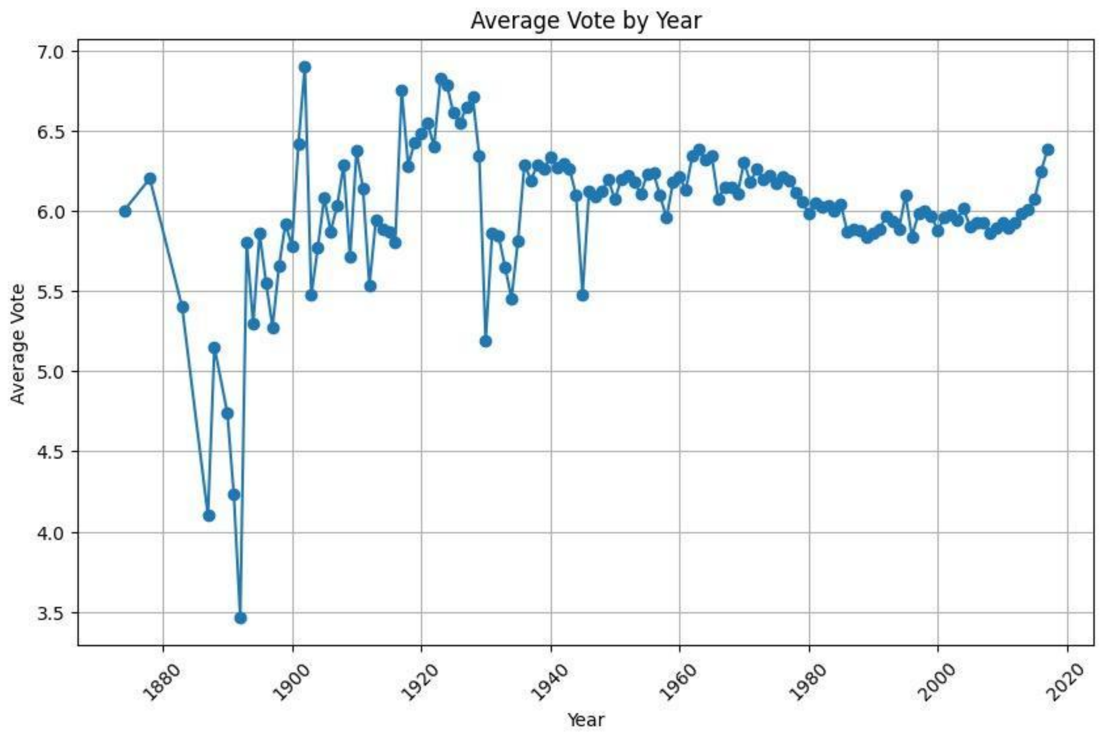
We didn’t implement the logout function as it’s quite simple in this scenario to exit the program and start again. If the client is a website, then a logout page should be ideal.

Data Mining

Domain question:
What factors determine the votes that a movie receives?

Technique:
grouping and aggregation, association discovery

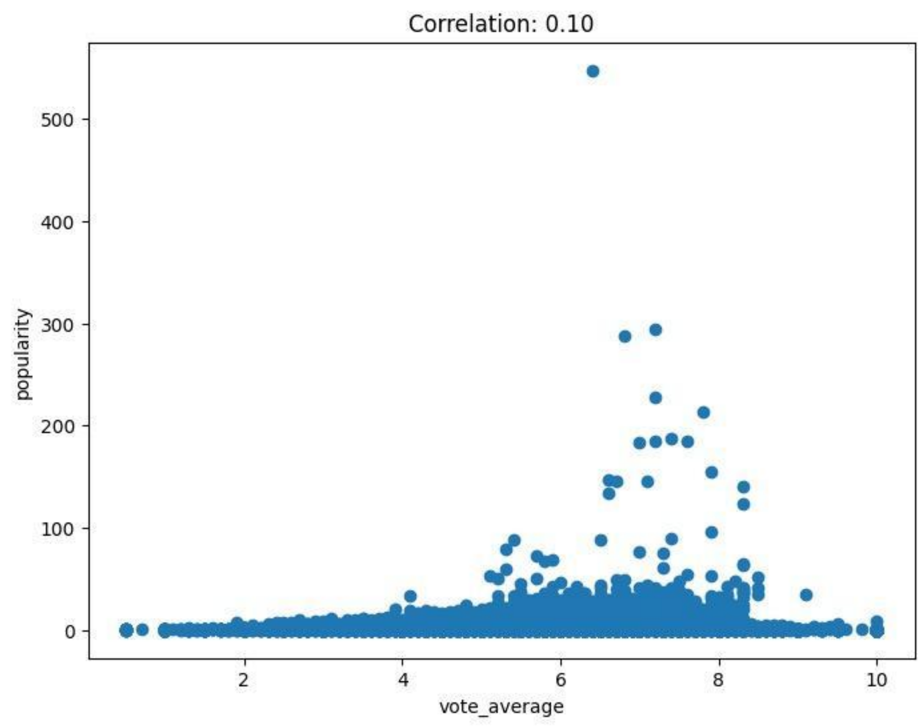
Implementation:
Our group chose the release year, popularity, revenue, and runtime to find whether they had an impact on the movie's votes. We used Jupyter Notebook to build the model and plot the graph to show the result directly.



Graph1: Average Vote by Year

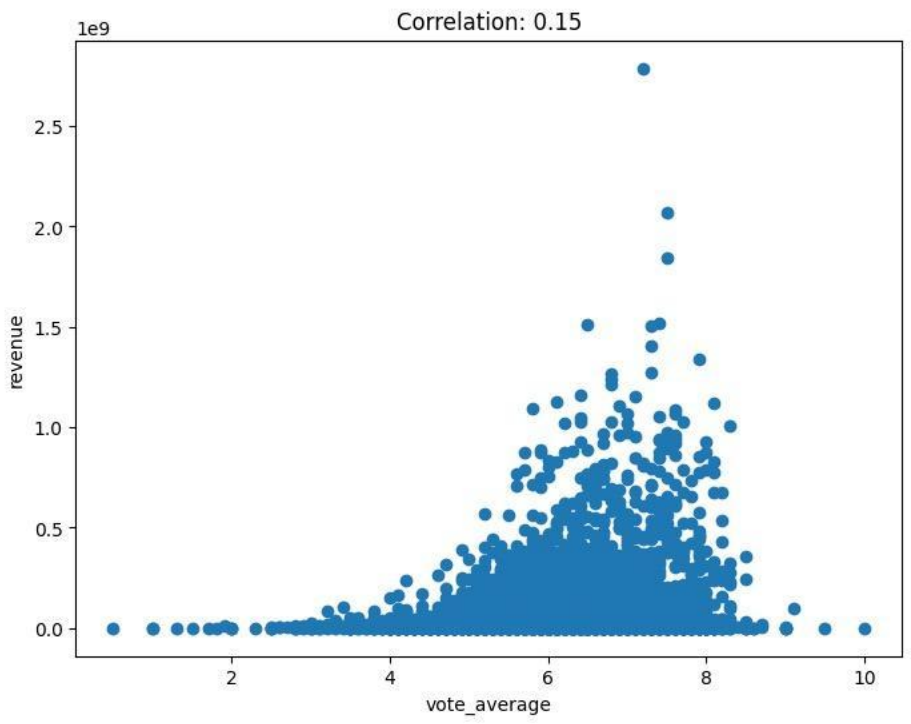
Every movie has many ratings and an average rating. The y-axis means that each year, a lot of movies are released. The average vote means calculating the mean of all average ratings of the movies that were produced in that year. So, what the graph tells us is which years produced the high-quality movies and low-quality movies. The high quality here means the movie received high average votes, and low quality means low average votes. If our result is a flat line, then we can conclude that the released year has nothing to do with the average votes.

However, it is clear that from 1880 to 1940, the average vote is not stable at all. So, the release year of the movie determines the votes in that period. But since 1940, the line seems to be flat. So, we can conclude that since 1940, the released year does not affect the average vote much.



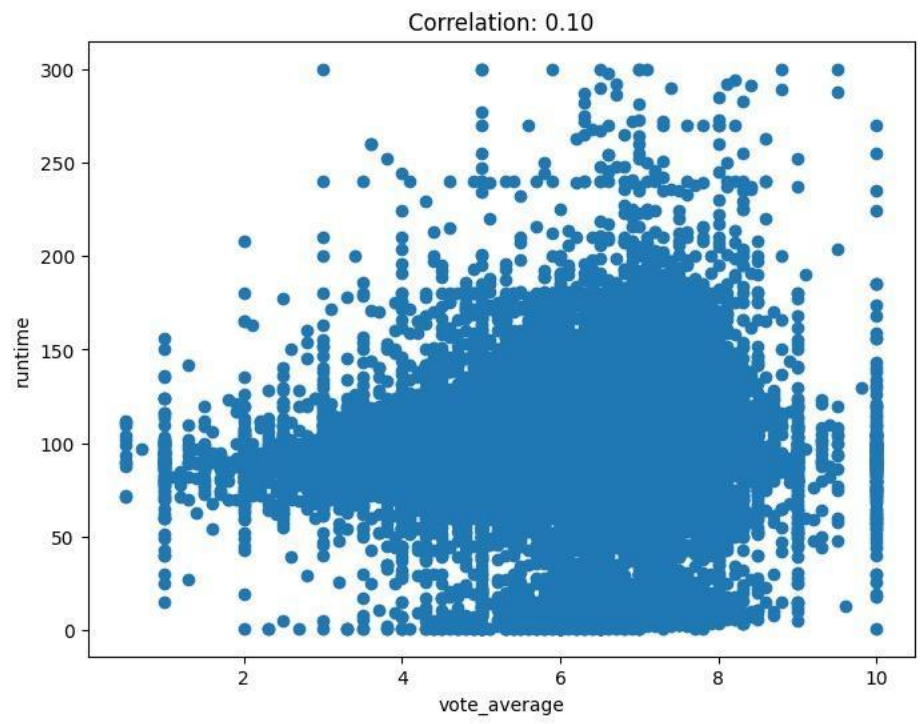
Graph 2: Correlation between vote_average and popularity

From the shape of the diagram, we can see that the most popular movies tend to have an average vote from 6 to 8. So, we can say that popularity influences the average vote to some degree.



Graph3: correlation between revenue and vote_average

As the revenue of the movie becomes higher, the average vote becomes higher, too. Although for some movies that have an average rate higher than 8, their revenue may not be very high. However, we can still conclude that revenue has a great impact on the average vote because the dots are not distributed uniformly.



Graph4: Correlation between runtime and vote_average

However, for this graph, the dots are distributed uniformly, and they do not have distinguishing features. As a consequence, the runtime of the movie does not influence the average vote.

So, we can draw the conclusion that the released year, popularity, and revenue have some connection with the votes of a movie. The runtime of a movie has nothing to do with the movie votes.