Dataset: Rotten Tomatoes

The dataset I will be using is the Rotten Tomatoes dataset. This data was obtained from Kaggle, where a user published this dataset after web scarping the official Rotten tomatoes website (https://www.rottentomatoes.com/). The dataset includes reviews of movies from all around the world and comprises two distinct review types: ratings from regular members and rating from professional critics, known as the 'Tomatometer'. With this dataset, my objective is to process the data and extract valuable insights to address my research questions.

The dataset was obtained from: https://www.kaggle.com/datasets/andrezaza/clapper-massive-rotten-tomatoes-movies-and-reviews

Quality

I believe the dataset is in great shape when it comes to quality. The data is scraped from the official Rotten Tomatoes website. The website is globally recognized and a highly trusted source for movie recommendations. A point that needs to be highlighted is that the certain fields in the dataset are absent because the website does not provide the details. Most of the empty cells are replaced by NumPy NaN . Moreover, the data is meticulously arranged in the correct rows and columns, resulting in a dataset of high quality.

However, I would like to point out that the original version of the database includes an 'id' column. The id column is in string format. Initially, I used this column as the primary key for my table. But I soon discovered that this column is not unique, as there were some duplicated entries. To address this issue, I had to set an auto-incrementing integer primary key for my table to serve as a reliable identifier for each movie entry.

Level of details

In terms of details, the dataset contains all the important information necessary to provide responses to my questions. The column such as genre, director, language, distributor, and scores, are examples of fields required to get back result of my questions, and all of them are included inside the dataset. While the dataset is already in high usability, its utility could be further enhanced if it were to be linked with another dataset. One potential dataset that could be linked is the Rotten Tomatoes reviews dataset, which contains records of every review from the user.

Discovery

I came across this dataset while web searching. At first, I had intention of finding dataset related to K-drama or K-pop. However, the dataset available on Kaggle for these categories did not include the fields I required, or they were in Korean, making the analysis of the dataset difficult. So, I further looked through Kaggle and I found this dataset. The dataset that perfectly meets all the fields I require and having enough records and at the same time subject of my interest.

Terms of use of the dataset

The dataset was available on Kaggle under the CC0: Public Domain License. This license signifies that the provider of the dataset has handed over their rights to the work according to copyright law, making it freely available to the public domain worldwide. As a result, we can use, modify, distribute, and perform the work without the need for permission.

Interest

I find this dataset fascinating since I'm a movie lover. I frequently rely on the Rotten Tomatoes website to discover movies to watch. I believe my taste in movies closely align with the critics on the website, as I enjoyed most of the movies with high Tomatometer score. However, even after filtering for my preference for specific genres and ensuring they meet the minimum Tomatometer score, there still have vast amount of availability. Which result in, struggling to find the next movie to watch. So instead of browsing the website every time, I thought it would be good idea to combine my passion for movie with my schoolwork. This way at the end of the day I could finish my work and have list of movies I can enjoy.

As I am not a huge fan of reading subtitles and dubbed movies, I prefer to watch movies in English or Korea. And I prefer the movies with runtime of at least 90 minutes, as short movies tend to feel more like drama. Most of the time, I like to watch movies that are genres of either comedy, drama, action, romance, crime, adventure, sci-fi. Lastly, I specifically look for movies with scores more than 80 marks.

I always enjoyed movies directed by Christopher Nolan and a curiosity rose of which writer did he work together with most frequently. So, I will be looking into this.

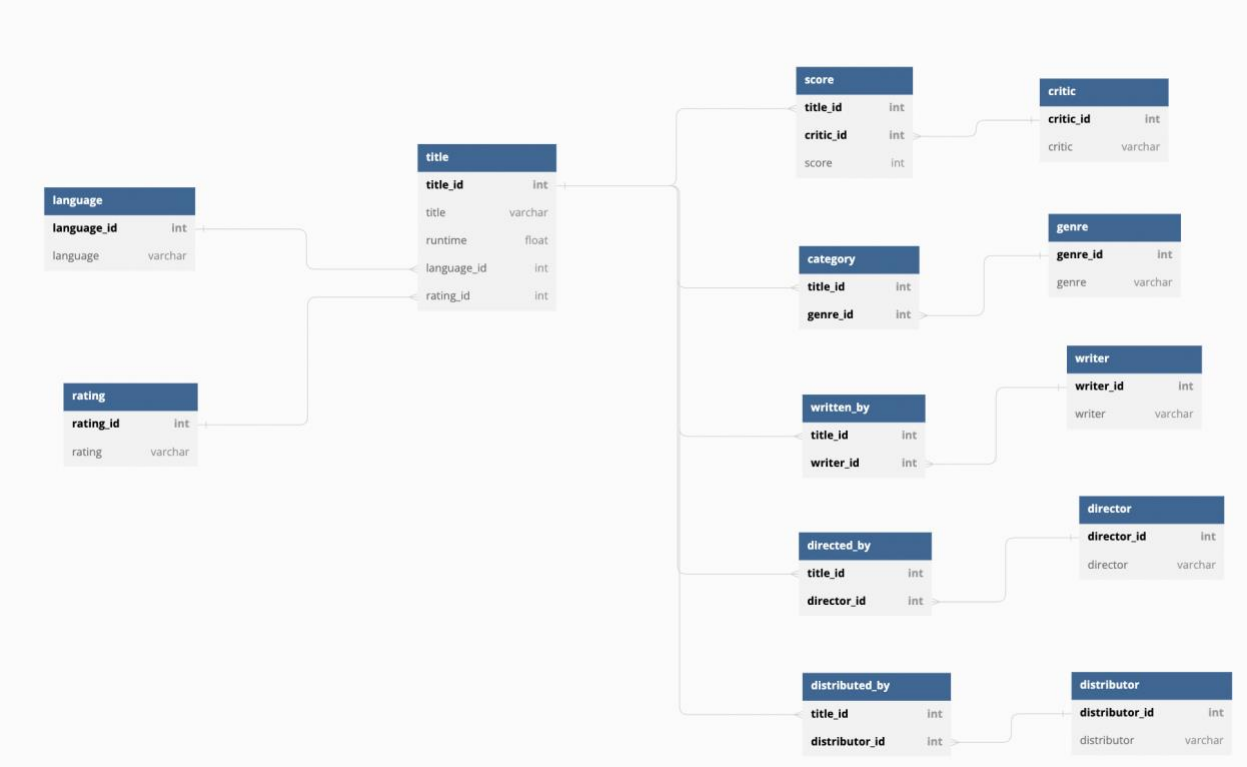
Additionally, I'm interested in finding the steaming platform that offers the most movies of my likings. Since subscribing to multiple streaming services can be costly, I aim to minimize my expenses by identifying the platform that has the highest number of movies I could enjoy and willing to subscribe to that particular platform. I will be considering the following platforms: Netflix, Amazon Prime Video, Disney+, Hulu, HBO Max, and Paramount+.

Based on these interests, I have come up with the following question:

1. Which movies have a Tomatometer or audience score exceeding 90, are in English or Korea, have a runtime greater than 80 minutes, and fall under the genres of comedy, drama, action, romance, crime, adventure, or sci-if?
2. Among the movies directed by Christopher Nolan, which writer has he collaborated with most frequently?
3. Which distributor offers the highest number of movies that match my preferences?

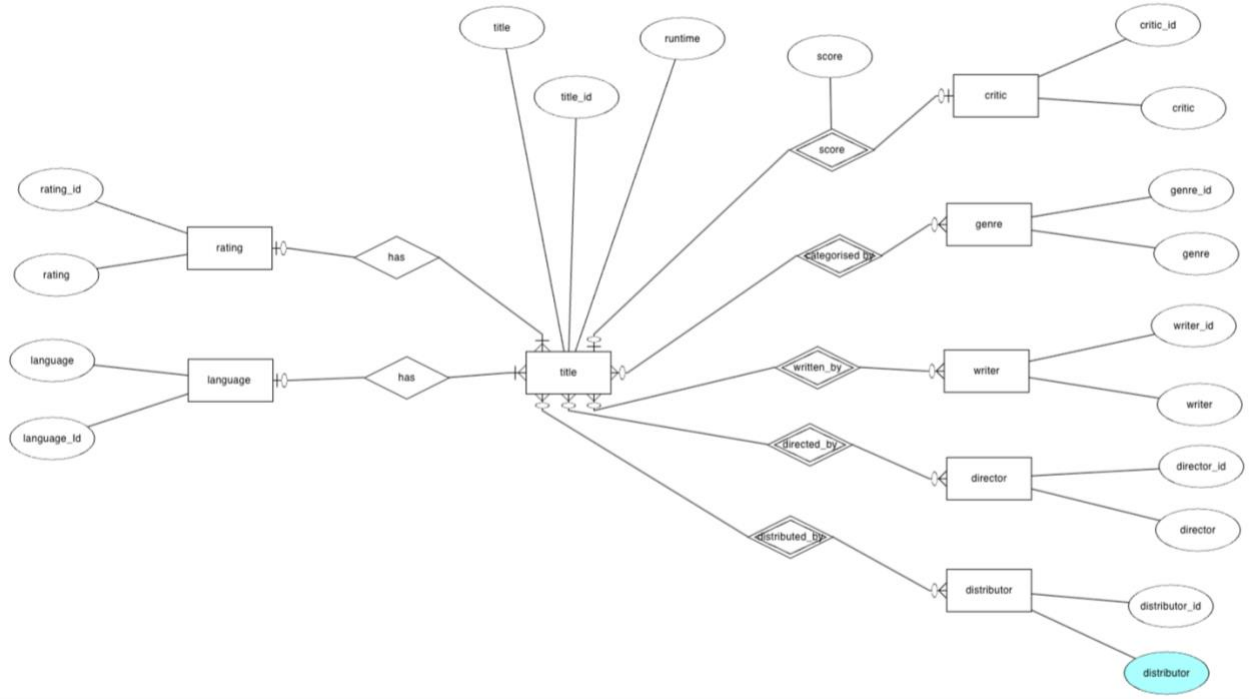
Relational Schema and ER Diagram:

Relational Schema



Link to relational schema: https://dbdiagram.io/d/6476205c7764f72fcf1d66b3

Entity - Relationship Diagram



List database tables and fields:

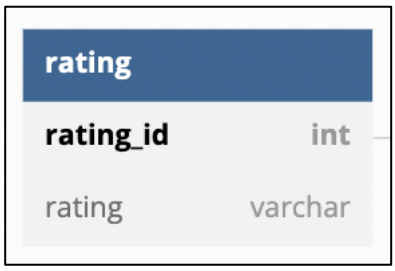
There is total 13 tables in my database. The tables are:

- language
- rating
- title
- score
- category
- written_by
- directed_by
- distributed_by
- Critic
- Genre
- Writer
- Director
- distributor

Language table:

The language table has primary key of "language_id". Each "language_id" unique identifies each language.

Rating table:

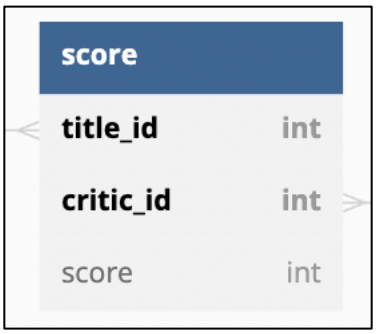


The rating table has a primary key of "rating_id". This table is used for storing ratings such as PG-13 and TV-14. Each unique "rating_id" identifies a specific rating in the table.

Title table:

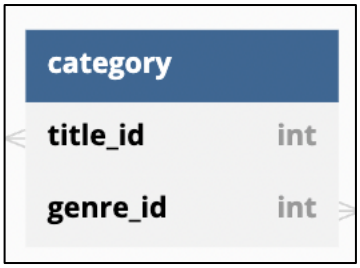
The title table has a primary key of "title_id". This table includes title, runtime, "language_id" and "rating_id". The "language_id" and "rating_id" are foreign keys of the table. The "language_id" references the language table which links each movie to corresponding language. Similarly, the "rating_id" references the rating table linking each movie to correct rating category.

Score table:



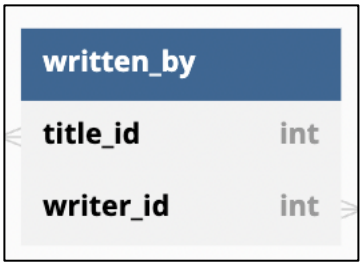
This Score table is a junction table connecting the title table and the critic table. The primary keys of the table are “title_id” and “critic_id”, and they are foreign keys as well. The “title_id” and “critic_id” references respective tables. This score table enables connection between specific title with individual critics, making it easier to retrieve scores assigned by each critic.

Category table:



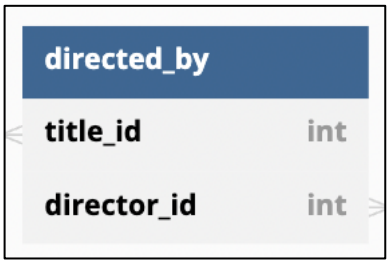
This category table is a junction table connecting the title table and the genre table. The primary keys of the table are “title_id” and “genre_id”, and they are foreign keys as well. The “title_id” references title table while “genre_id” references genre table.

Written_by table:



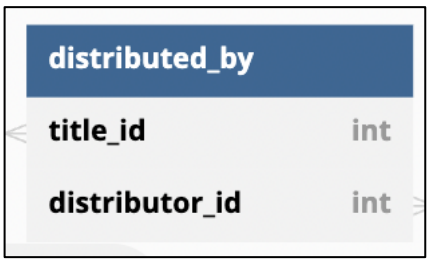
This written_by table is a junction table connecting the title table and the writer table. The primary keys of the table are “title_id” and “writer_id”, and they are foreign keys as well. The “title_id” references title table while “writer_id” references writer table.

Directed_by table:



This directed_by table is a junction table connecting the title table and the director table. The primary keys of the table are “title_id” and “director_id”, and they are foreign keys as well. The “title_id” references title table while “director_id” references director table.

Distributed_by table:



This distributed_by table is a junction table connecting the title table and the distributor table. The primary keys of the table are “title_id” and “distributor_id”, and they are foreign keys as well. The “title_id” references title table while “distributor _id” references distributor table.

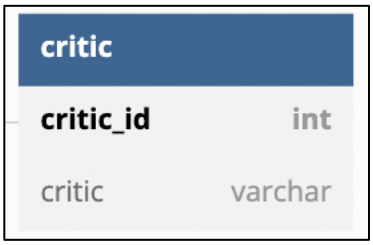
Genre table:

The genre table contains two columns the genre_id and genre, where “genre_id” is primary key of the table. Each “genre_id” provides a unique identifier for each genre.

Writer table:

The writer table contains two columns the writer_id and writer, where “writer_id” is primary key of the table. Each “writer_id” provides a unique identifier for each writer.

Critic table:



The critic table contains two columns the critic_id and critic, where “critic _id” is primary key of the table. Each “critic _id” provides a unique identifier for each critic. The critic is either “Tomatometer” or “AudienceScore”

Director table:

The director table contains two columns the director_id and director, where “director_id” is primary key of the table. Each “director_id” provides a unique identifier for each director.

Distributor table:

The distributor table contains two columns the distributor _id and distributor, where “distributor _id” is primary key of the table. Each “distributor _id” provides a unique identifier for each distributor.

Evaluating the tables (Normalisation):

1NF

The requirement for table to be in first normal form are:

- Each attribute in the table is single-valued attributes.
- Each column int the table should have unique column names.
- Each row in a table must be uniquely identifiable.
- Each column should contain values of the same type.

The original form of the dataset I have retrieved fulfills the second, third, fourth rules for being in the first normal form. However, the table does not meet first requirement, which is to have single-valued values. Specifically, the columns *genre, director, writer*, *distributor,* they have multi values.

genre	ori
Comedy, Horror, Sci-fi	En
Drama	En
Drama	Ko
Action, Mystery & thriller	En
Fantasy, Adventure, Animation	En
Adventure, Drama, Romance	En

Fig 1.1

For instance, each record contained different genre instead of one genre. As can be seen on fig1.1.

Therefore, to bring the table into the first normal form, some data cleanings steps need to be performed. These steps were carried out using Jupyter Notebook.In the later part of the document, I will further explain about what actions I have taken during the data cleaning process. However, for the time being, let’s assume that the table has been cleaned and it is in first normal form.

2NF

Moving on to the second normal form, it requires the table to be in first normal form, and each non-key attribute should depend on the entire primary key. I can confirm that my tables mee the criteria, ensuring that the table is in second normal form (2NF).

3NF

For the third normal form, the table should be in second normal form and should not have transitive functional dependencies. My tables fulfill these rules, so it is in third normal form.

Boyce – Codd Normal Form (BCNF)

title_id	title	audienceScore	tomatoMeter	rating	runtime	genre	language	director	writer	distributor
1	Space Zombie Bingo	50			75	Comedy	English	George Orm	George Ormrod	
1	Space Zombie Bingo	50			75	Comedy	English	George Orm	John Sabotta	
1	Space Zombie Bingo	50			75	Horror	English	George Orm	George Ormrod	
1	Space Zombie Bingo	50			75	Horror	English	George Orm	John Sabotta	
1	Space Zombie Bingo	50			75	Sci-fi	English	George Orm	George Ormrod	
1	Space Zombie Bingo	50			75	Sci-fi	English	George Orm	John Sabotta	
2	The Green Grass				114	Drama	English	Tiffany Edwa	Tiffany Edwards	
3	Love Lies	43			120	Drama	Korean	Park Heung-S	Ha Young-Joon	
3	Love Lies	43			120	Drama	Korean	Park Heung-S	Jeon Yun-su	
3	Love Lies	43			120	Drama	Korean	Park Heung-S	Song Hye-jin	
3	Love Lies	43			120	Drama	Korean	Heung-Sik Pa	Ha Young-Joon	

The current table is not in BCNF, in order to meet the requirement for BCNF the table must have functional dependency X-> Y and where X is super key. So, to meet the requirement the decomposition of the table was undergone.
The given example is focusing on the director column of the title table.

Title table:

title

title_id	title	runtime	language_id	rating_id
1	Space Zombie Bingo	75.0	0	
2	The Green Grass	114.0	0	
3	Love Lies	120.0	1	

*Cadinate key of the title table is title_id

Director table :

director

director_id	director
0	George Ormrod
1	Tiffany Edwards
2	Park Heung-Sik

*Cadinate key of the title table is title_id

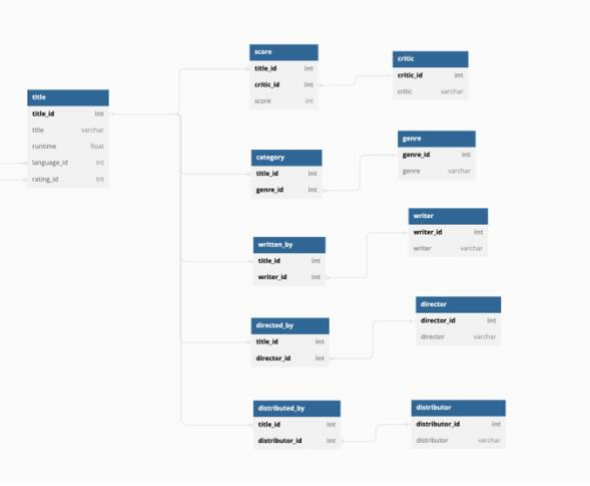
Directed_by table:

directed_by

title_id	director_id
1	0
2	1
3	2
3	3
4	4

*Cadinate key of the title table is title_id, director_id

After decomposition of the original table, it is now in BCNF as it meets the requirement to have functional dependency X-> Y and where X is super key. This is not the only table that has been decomposed to meet the requirement of BCNF from the title table, 10 additional tables were created for it. The following tables are, for score critic column score and critic table was created, For genre column, category and genre table was created. Furthermore, for writer column, written_by table and writer table was created, for distributor column, distributor and distributed table was created. Including the director and directed_by table total 10 tables were created to satisfy BCNF.



As can be seen they all have the functional dependency. And from this it eliminates the redundancy of storing, critic, genre, writer, director, and distributor in the title table.

Creating database:

I have created the database using Coursera lab terminal.

```
CREATE DATABASE rotten_tomatoes;
USE rotten_tomatoes;
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  SQL CONSOLE

mysql> create database rotten_tomatoes;
Query OK, 1 row affected (0.01 sec)

mysql> use rotten_tomatoes;
Database changed
```

```
CREATE TABLE `writer` (
 `writer_id` int PRIMARY KEY,
 `writer` varchar(255)
);
```

```
mysql> CREATE TABLE `writer` (
    ->   `writer_id` int PRIMARY KEY,
    ->   `writer` varchar(255)
    -> );
Query OK, 0 rows affected (0.15 sec)
```

```
CREATE TABLE `director` (
 `director_id` int PRIMARY KEY,
 `director` varchar(255)
);
```

```
mysql> CREATE TABLE `director` (
    ->   `director_id` int PRIMARY KEY,
    ->   `director` varchar(255)
    -> );
Query OK, 0 rows affected (0.15 sec)
```

```
CREATE TABLE `distributor` (
 `distributor_id` int PRIMARY KEY,
 `distributor` varchar(255)
);
```

```
mysql> CREATE TABLE `distributor` (
    ->   `distributor_id` int PRIMARY KEY,
    ->   `distributor` varchar(255)
    -> );
Query OK, 0 rows affected (0.12 sec)
```

```
CREATE TABLE `title` (
```

```
 `title_id` int PRIMARY KEY,
 `title` varchar(255),
 `runtime` float,
 `language_id` int,
 `rating_id` int,
 FOREIGN KEY (`language_id`) REFERENCES `language` (`language_id`),
 FOREIGN KEY (`rating_id`) REFERENCES `rating` (`rating_id`)
);
```

```
mysql> CREATE TABLE `title` (
    ->   `title_id` int PRIMARY KEY,
    ->   `title` varchar(255),
    ->   `runtime` float,
    ->   `language_id` int,
    ->   `rating_id` int,
    ->   FOREIGN KEY (`language_id`) REFERENCES `language` (`language_id`),
    ->   FOREIGN KEY (`rating_id`) REFERENCES `rating` (`rating_id`)
    -> );
Query OK, 0 rows affected (0.21 sec)
```

```
CREATE TABLE `rating` (
 `rating_id` int PRIMARY KEY,
 `rating` varchar(255)
);
```

```
mysql> CREATE TABLE `rating` (
    ->   `rating_id` int PRIMARY KEY,
    ->   `rating` varchar(255)
    -> );
Query OK, 0 rows affected (0.14 sec)
```

```
CREATE TABLE `genre` (
 `genre_id` int PRIMARY KEY,
 `genre` varchar(255)
);
```

```
mysql> CREATE TABLE `genre` (
    ->   `genre_id` int PRIMARY KEY,
    ->   `genre` varchar(255)
    -> );
Query OK, 0 rows affected (0.13 sec)
```

```
CREATE TABLE `language` (
  `language_id` int PRIMARY KEY,
  `language` varchar(255)
);
```

```
mysql> CREATE TABLE `language` (
    ->   `language_id` int PRIMARY KEY,
    ->   `language` varchar(255)
    -> );
Query OK, 0 rows affected (0.15 sec)
```

```
CREATE TABLE `category` (
  `title_id` int,
  `genre_id` int,
  PRIMARY KEY (`title_id`, `genre_id`),
  FOREIGN KEY (`title_id`) REFERENCES `title` (`title_id`),
  FOREIGN KEY (`genre_id`) REFERENCES `genre` (`genre_id`)
);
```

```
mysql> CREATE TABLE `category` (
    ->   `title_id` int,
    ->   `genre_id` int,
    ->   PRIMARY KEY (`title_id`, `genre_id`),
    ->   FOREIGN KEY (`title_id`) REFERENCES `title` (`title_id`),
    ->   FOREIGN KEY (`genre_id`) REFERENCES `genre` (`genre_id`)
    -> );
Query OK, 0 rows affected (0.20 sec)
```

```
CREATE TABLE `written_by` (
  `title_id` int,
  `writer_id` int,
  PRIMARY KEY (`title_id`, `writer_id`),
  FOREIGN KEY (`title_id`) REFERENCES `title` (`title_id`),
  FOREIGN KEY (`writer_id`) REFERENCES `writer` (`writer_id`)
);
```

```
mysql> CREATE TABLE `written_by` (
    ->   `title_id` int,
    ->   `writer_id` int,
    ->   PRIMARY KEY (`title_id`, `writer_id`),
    ->   FOREIGN KEY (`title_id`) REFERENCES `title` (`title_id`),
    ->   FOREIGN KEY (`writer_id`) REFERENCES `writer` (`writer_id`)
    -> );
Query OK, 0 rows affected (0.19 sec)
```

```
CREATE TABLE `directed_by` (
 `title_id` int,
 `director_id` int,
 PRIMARY KEY (`title_id`, `director_id`),
 FOREIGN KEY (`title_id`) REFERENCES `title` (`title_id`),
 FOREIGN KEY (`director_id`) REFERENCES `director` (`director_id`)
);
```

```
mysql> CREATE TABLE `directed_by` (
    ->   `title_id` int,
    ->   `director_id` int,
    ->   PRIMARY KEY (`title_id`, `director_id`),
    ->   FOREIGN KEY (`title_id`) REFERENCES `title` (`title_id`),
    ->   FOREIGN KEY (`director_id`) REFERENCES `director` (`director_id`)
    -> );
Query OK, 0 rows affected (0.19 sec)
```

```
CREATE TABLE `distributed_by` (
 `title_id` int,
 `distributor_id` int,
 PRIMARY KEY (`title_id`, `distributor_id`),
 FOREIGN KEY (`title_id`) REFERENCES `title` (`title_id`),
 FOREIGN KEY (`distributor_id`) REFERENCES `distributor` (`distributor_id`)
);
```

```
mysql> CREATE TABLE `distributed_by` (
    ->   `title_id` int,
    ->   `distributor_id` int,
    ->   PRIMARY KEY (`title_id`, `distributor_id`),
    ->   FOREIGN KEY (`title_id`) REFERENCES `title` (`title_id`),
    ->   FOREIGN KEY (`distributor_id`) REFERENCES `distributor` (`distributor_id`)
    -> );
Query OK, 0 rows affected (0.19 sec)
```

```
CREATE TABLE `score` (
 `title_id` int,
 `critic_id` int,
```

```
 `score` int,
 PRIMARY KEY (`title_id`, `critic_id`),
 FOREIGN KEY (`title_id`) REFERENCES `title` (`title_id`),
 FOREIGN KEY (`critic_id`) REFERENCES `critic` (`critic_id`)
);
```

```
mysql> CREATE TABLE `score` (
    ->   `title_id` int,
    ->   `critic_id` int,
    ->   `score` int,
    ->   PRIMARY KEY (`title_id`, `critic_id`),
    ->   FOREIGN KEY (`title_id`) REFERENCES `title` (`title_id`),
    ->   FOREIGN KEY (`critic_id`) REFERENCES `critic` (`critic_id`)
    -> );
Query OK, 0 rows affected (0.20 sec)
```

```
CREATE TABLE `critic` (
 `critic_id` int PRIMARY KEY,
 `critic` varchar (255)
);
```

```
mysql> CREATE TABLE `critic` (
    ->   `critic_id` int PRIMARY KEY,
    ->   `critic` varchar(255)
    -> );
Query OK, 0 rows affected (0.13 sec)
```

Query to insert CSV file into corresponding tables.

Just like creating the tables, I used the Coursera lab terminal to load the CSV files into their respective tables

```
DELETE FROM language;

LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/language.csv'
INTO TABLE language
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

```
mysql>
mysql> LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/language.csv'
    -> INTO TABLE language
    -> FIELDS TERMINATED BY ','
    -> ENCLOSED BY '"'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 ROWS;
Query OK, 110 rows affected (0.04 sec)
Records: 110  Deleted: 0  Skipped: 0  Warnings: 0
```

DELETE FROM distributor;

LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/distributor.csv'
INTO TABLE distributor
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

```
mysql>
mysql> LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/distributor.csv'
    -> INTO TABLE distributor
    -> FIELDS TERMINATED BY ','
    -> ENCLOSED BY '"'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 ROWS;
Query OK, 3141 rows affected (0.26 sec)
Records: 3141  Deleted: 0  Skipped: 0  Warnings: 0
```

DELETE FROM rating;

LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/rating.csv'
INTO TABLE rating
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS

```
mysql>
mysql> LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/rating.csv'
    -> INTO TABLE rating
    -> FIELDS TERMINATED BY ','
    -> ENCLOSED BY '"'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 ROWS;
Query OK, 10 rows affected (0.03 sec)
Records: 10  Deleted: 0  Skipped: 0  Warnings: 0
```

DELETE FROM genre;

LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/genre.csv'
INTO TABLE genre
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

```
mysql>
mysql> LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/genre.csv'
    -> INTO TABLE genre
    -> FIELDS TERMINATED BY ','
    -> ENCLOSED BY '"'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 ROWS;
Query OK, 68 rows affected (0.03 sec)
Records: 68  Deleted: 0  Skipped: 0  Warnings: 0
```

DELETE FROM writer;

LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/writer.csv'
INTO TABLE writer
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

```
mysql>
mysql> LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/writer.csv'
    -> INTO TABLE writer
    -> FIELDS TERMINATED BY ','
    -> ENCLOSED BY '"'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 ROWS;
Query OK, 74111 rows affected (1.65 sec)
Records: 74111  Deleted: 0  Skipped: 0  Warnings: 0
```

DELETE FROM director;

LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/director.csv'
INTO TABLE director
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

```
mysql>
mysql> LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/director.csv'
    -> INTO TABLE director
    -> FIELDS TERMINATED BY ','
    -> ENCLOSED BY '"'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 ROWS;
Query OK, 64718 rows affected (1.66 sec)
Records: 64718  Deleted: 0  Skipped: 0  Warnings: 0
```

DELETE FROM title;

LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/title.csv'
INTO TABLE title
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(title_id, title, @runtime, @language_id, @rating_id)
SET rating_id = NULLIF(@rating_id, ''),
  language_id = NULLIF(@language_id, ''),
  runtime = NULLIF(@runtime, '');

```
mysql>
mysql> LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/title.csv'
    -> INTO TABLE title
    -> FIELDS TERMINATED BY ','
    -> ENCLOSED BY '"'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 ROWS
    -> (title_id, title, @runtime, @language_id, @rating_id)
    -> SET rating_id = NULLIF(@rating_id, ''),
    ->     language_id = NULLIF(@language_id, ''),
    ->     runtime = NULLIF(@runtime, '');

Query OK, 143258 rows affected (3.90 sec)
Records: 143258  Deleted: 0  Skipped: 0  Warnings: 0
```

DELETE FROM distributor;

LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/distributor.csv'
INTO TABLE distributor
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

```
mysql> LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/distributor.csv'
    -> INTO TABLE distributor
    -> FIELDS TERMINATED BY ','
    -> ENCLOSED BY '"'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 ROWS;
Query OK, 3141 rows affected (0.07 sec)
Records: 3141  Deleted: 0  Skipped: 0  Warnings: 0
```

DELETE FROM category;

LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/category.csv'
INTO TABLE category
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

```
mysql> LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/category.csv'
    -> INTO TABLE category
    -> FIELDS TERMINATED BY ','
    -> ENCLOSED BY '"'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 ROWS;

Query OK, 204236 rows affected (4.51 sec)
Records: 204236  Deleted: 0  Skipped: 0  Warnings: 0
```

DELETE FROM written_by;

LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/written_by.csv'
INTO TABLE written_by
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

```
mysql> LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/written_by.csv'
    -> INTO TABLE written_by
    -> FIELDS TERMINATED BY ','
    -> ENCLOSED BY '"'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 ROWS;

Query OK, 136284 rows affected (4.30 sec)
Records: 136284  Deleted: 0  Skipped: 0  Warnings: 0
```

DELETE FROM directed_by;

LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/directed_by.csv'
INTO TABLE directed_by

FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

```
mysql> LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/directed_by.csv'
    -> INTO TABLE directed_by
    -> FIELDS TERMINATED BY ','
    -> ENCLOSED BY '"'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 ROWS;

Query OK, 150745 rows affected (4.22 sec)
Records: 150745  Deleted: 0  Skipped: 0  Warnings: 0
```

DELETE FROM distributed_by;

LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/distributed_by.csv'
INTO TABLE distributed_by
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

```
mysql> LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/distributed_by.csv'
    -> INTO TABLE distributed_by
    -> FIELDS TERMINATED BY ','
    -> ENCLOSED BY '"'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 ROWS;

Query OK, 26494 rows affected (2.20 sec)
Records: 26494  Deleted: 0  Skipped: 0  Warnings: 0
```

DELETE FROM critic;

LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/critic.csv'
INTO TABLE critic
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

```
mysql> LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/critic.csv'
    -> INTO TABLE critic
    -> FIELDS TERMINATED BY ','
    -> ENCLOSED BY '"'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 ROWS;
Query OK, 2 rows affected (0.03 sec)
Records: 2  Deleted: 0  Skipped: 0  Warnings: 0
```

DELETE FROM score;

```
LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/score.csv'
INTO TABLE score
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

```
mysql> LOAD DATA INFILE '/home/coder/project/mid-term/rotten_tomatoes/web-app/data/score.csv'
    -> INTO TABLE score
    -> FIELDS TERMINATED BY ','
    -> ENCLOSED BY '"'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 ROWS;
Query OK, 107125 rows affected (3.19 sec)
Records: 107125  Deleted: 0  Skipped: 0  Warnings: 0
```

Data cleaning

For the data cleaning process of the dataset, I have used Jupyter Notebook. Several tasks were performed to ensure data quality, such as removing special characters, dropping unnecessary columns, replacing empty values, and splitting columns with multiple values. For more detailed information, please refer to the appendix section.

Reflect on how well the database reflects the data.

I believe that my database reflects by data very well. To ensure of data accuracy, I have taken measures to address special characters and empty values by removing or replacing them appropriately. I have also ensured that no entries were dropped, and all the data has been successfully stored in the respective tables. The database's design enables each entry to be uniquely identified, facilitating efficient search operations, and further enhancing the database's ability to faithfully reflect the data.

Queries for getting answers to my questions:

1. **Question1**: Which movies have a Tomatometer or audience score exceeding 90, are in English or Korea, have a runtime greater than 80 minutes, and fall under the genres of comedy, drama, action, romance, crime, adventure, or sci-if?

```
select title.title_id, title.title, genre.genre,language.language, score.score, critic.critic, title.runtime from title\
    join score on title.title_id = score.title_id\
    join critic on score.critic_id = critic.critic_id\
```

```
join language on title.language_id = language.language_id\
join category on title.title_id = category.title_id\
join genre on category.genre_id = genre.genre_id\
where title.runtime > 80 and score.score >90\
and genre.genre in ("comedy", "drama", "action", "romance", "crime", "adventure", "sci-if")\
and language.language in ("English","Korean");
```

To retrieve the list of movies meeting my preferences, the above SQL command was used. The title table was joined with necessary tables, including score, critic, language, category, and genre. Then, a where statement was used to filter the result based on the requirements mentioned in my question. As a result, total 2517 movies were found that matches my preferences.

2. **Question2:** Among the movies directed by Christopher Nolan, which writer has he collaborated with most frequently?

Command1:

```
select writer.writer, count(distinct title.title) as Number_of_movies\
    from title\
    join directed_by on title.title_id = directed_by.title_id\
    join director on directed_by.director_id = director.director_id\
    join written_by on title.title_id = written_by.title_id\
    join writer on written_by.writer_id = writer.writer_id\
    where director.director = 'Christopher Nolan'\
    group by writer.writer;
```

Command2:

```
select title.title_id, title.title, director.director, writer.writer from title\
    join directed_by on title.title_id=directed_by.title_id\
    join director on directed_by.director_id = director.director_id\
    join written_by on title.title_id = written_by.title_id\
    join writer on  written_by.writer_id = writer.writer_id\
    where director.director in ('Christopher Nolan');
```

For this question I used two command to provide more details. The first command returns the number of times ‘Christopher Nolan’ has worked together with each writer. The second command returns the number of movies directed by ‘Christopher Nolan’ along with the corresponding writers. For both commands, title table was joined with director, writer table, and where statement was used filter out movies directed by ‘Christopher Nolan’.

The first command has additional statement of ‘group by writer.writer’. This statement allows me to obtain desired result, which is number of times the writer has contributed.

The first command returned:

```
mysql> select writer.writer, count(distinct title.title) as Number_of_movies\
    ->         from title\
    ->         join directed_by on title.title_id = directed_by.title_id\
    ->         join director on directed_by.director_id = director.director_id\
    ->         join written_by on title.title_id = written_by.title_id\
    ->         join writer on written_by.writer_id = writer.writer_id\
    ->         where director.director = 'Christopher Nolan'\
    ->         group by writer.writer;
+------------------+------------------+
| writer           | Number_of_movies |
+------------------+------------------+
| Bob Kane         |                1 |
| Christopher Nolan |               12 |
| David S. Goyer   |                1 |
| Hillary Seitz    |                1 |
| Jonathan Nolan   |                5 |
+------------------+------------------+
5 rows in set (0.01 sec)
```

The second command returned:

```
mysql> select title.title_id, title.title, director.director, writer.writer from title\
    ->         join directed_by on title.title_id=directed_by.title_id\
    ->         join director on directed_by.director_id = director.director_id\
    ->         join written_by on title.title_id = written_by.title_id\
    ->         join writer on  written_by.writer_id = writer.writer_id\
    ->         where director.director in ('Christopher Nolan');
+----------+----------------------+-------------------+-------------------+
| title_id | title                | director          | writer            |
+----------+----------------------+-------------------+-------------------+
|     6842 | The Dark Knight      | Christopher Nolan | Jonathan Nolan    |
|     6842 | The Dark Knight      | Christopher Nolan | Christopher Nolan |
|    11164 | The Dark Knight Rises | Christopher Nolan | Jonathan Nolan    |
|    11164 | The Dark Knight Rises | Christopher Nolan | Christopher Nolan |
|    13446 | The Prestige         | Christopher Nolan | Jonathan Nolan    |
|    13446 | The Prestige         | Christopher Nolan | Christopher Nolan |
|    17152 | Dunkirk              | Christopher Nolan | Christopher Nolan |
|    28475 | Tenet                | Christopher Nolan | Christopher Nolan |
|    31928 | Batman Begins        | Christopher Nolan | Christopher Nolan |
|    31928 | Batman Begins        | Christopher Nolan | David S. Goyer    |
|    31928 | Batman Begins        | Christopher Nolan | Bob Kane          |
|    71529 | Insomnia             | Christopher Nolan | Hillary Seitz     |
|    80193 | Interstellar         | Christopher Nolan | Jonathan Nolan    |
|    80193 | Interstellar         | Christopher Nolan | Christopher Nolan |
|    84543 | Following            | Christopher Nolan | Christopher Nolan |
|   116263 | Doodlebug            | Christopher Nolan | Christopher Nolan |
|   116542 | Memento              | Christopher Nolan | Jonathan Nolan    |
|   116542 | Memento              | Christopher Nolan | Christopher Nolan |
|   122210 | Inception            | Christopher Nolan | Christopher Nolan |
|   137025 | Oppenheimer          | Christopher Nolan | Christopher Nolan |
+----------+----------------------+-------------------+-------------------+
20 rows in set (0.02 sec)
```

Result:

The output shows that Christopher Nolan has collaborated mostly with himself, a total of 12 times out of 20, in both the roles of director and writer. If we exclude Christopher Nolan himself, then he has collaborated mostly with Jonathan Nolan, a total of 5 times.

3. **Question3:** Which distributor offers the highest number of movies that match my preferences?

Command1:

```
select distributor.distributor, COUNT(*) as movie_count\
    from title\
    join score on title.title_id = score.title_id\
    join critic on score.critic_id = critic.critic_id\
    join distributed_by on title.title_id = distributed_by.title_id\
    join distributor on distributed_by.distributor_id =  distributor.distributor_id\
    join language on title.language_id = language.language_id\
    join category on title.title_id = category.title_id\
    join genre on category.genre_id = genre.genre_id\
```

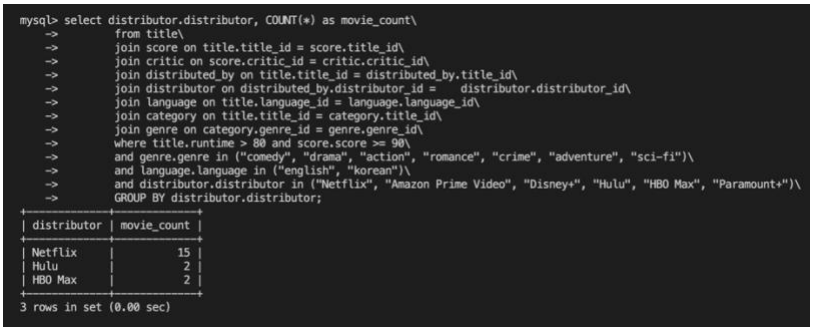
```
      where title.runtime > 80 and score.score >= 90\
      and genre.genre in ("comedy", "drama", "action", "romance", "crime", "adventure", "sci-fi")\
      and language.language in ("english", "korean")\
      and distributor.distributor in ("Netflix", "Amazon Prime Video", "Disney+", "Hulu", "HBO Max",
"Paramount+")\
      GROUP BY distributor.distributor;
```

Command2:

```
select title. title_id, title.title, genre.genre, language.language, score.score,\
       critic.critic, title.runtime, distributor.distributor\
      from title\
      join score on title.title_id = score.title_id\
      join critic on score.critic_id = critic.critic_id\
      join distributed_by on title.title_id = distributed_by.title_id\
      join distributor on distributed_by.distributor_id = distributor.distributor_id\
      join language on title.language_id = language.language_id\
      join category on title.title_id = category.title_id\
      join genre on category.genre_id = genre.genre_id\
      where title.runtime > 80 and score.score >= 90\
      and genre.genre in ("comedy", "drama", "action", "romance", "crime", "adventure", "sci-fi")\
      and distributor.distributor in ("Netflix", "Amazon Prime Video", "Disney+", "Hulu", "HBO Max",
"Paramount+")\
      and language.language in ("english", "korean");
```

For question3, I as well used two commands. The first command returns the count of movies available on each distributor of my choice that align with my likings of movie. The second command returns the list of movies along with the corresponding distributors.

The first command returned:



```
mysql> select distributor.distributor, COUNT(*) as movie_count\
    ->         from title\
    ->         join score on title.title_id = score.title_id\
    ->         join critic on score.critic_id = critic.critic_id\
    ->         join distributed_by on title.title_id = distributed_by.title_id\
    ->         join distributor on distributed_by.distributor_id =    distributor.distributor_id\
    ->         join language on title.language_id = language.language_id\
    ->         join category on title.title_id = category.title_id\
    ->         join genre on category.genre_id = genre.genre_id\
    ->         where title.runtime > 80 and score.score >= 90\
    ->         and genre.genre in ("comedy", "drama", "action", "romance", "crime", "adventure", "sci-fi")\
    ->         and language.language in ("english", "korean")\
    ->         and distributor.distributor in ("Netflix", "Amazon Prime Video", "Disney+", "Hulu", "HBO Max", "Paramount+")\
    ->         GROUP BY distributor.distributor;
+-------------+-------------+
| distributor | movie_count |
+-------------+-------------+
| Netflix     |          15 |
| Hulu        |           2 |
| HBO Max     |           2 |
+-------------+-------------+
3 rows in set (0.00 sec)
```

The second command returned:

```
mysql> select title.title_id, title.title, genre.genre, language.language, score.score,\
    ->          critic.critic, title.runtime, distributor.distributor\
    ->         from title\
    ->         join score on title.title_id = score.title_id\
    ->         join critic on score.critic_id = critic.critic_id\
    ->         join distributed_by on title.title_id = distributed_by.title_id\
    ->         join distributor on distributed_by.distributor_id = distributor.distributor_id\
    ->         join language on title.language_id = language.language_id\
    ->         join category on title.title_id = category.title_id\
    ->         join genre on category.genre_id = genre.genre_id\
    ->         where title.runtime > 80 and score.score >= 90\
    ->         and genre.genre in ("comedy", "drama", "action", "romance", "crime", "adventure", "sci-fi")\
    ->         and distributor.distributor in ("Netflix", "Amazon Prime Video", "Disney+", "Hulu", "HBO Max", "Paramount+")\
    ->         and language.language in ("english", "korean");
+----------+-----------------------------------------+-----------+----------+-------+---------------+---------+-------------+
| title_id | title                                   | genre     | language | score | critic        | runtime | distributor |
+----------+-----------------------------------------+-----------+----------+-------+---------------+---------+-------------+
|   112765 | Son of Monarchs                         | Drama     | English  |   100 | audienceScore |      97 | HBO Max     |
|   134254 | Red Notice                              | Action    | English  |    92 | audienceScore |     117 | Netflix     |
|    39563 | Hustle                                  | Drama     | English  |    93 | audienceScore |     118 | Netflix     |
|    17026 | Faraway                                 | Romance   | English  |    91 | audienceScore |     109 | Netflix     |
|     4186 | The Gray Man                            | Action    | English  |    90 | audienceScore |     129 | Netflix     |
|   112765 | Son of Monarchs                         | Drama     | English  |    92 | tomatoMeter   |      97 | HBO Max     |
|   106385 | Rye Lane                                | Romance   | English  |    98 | tomatoMeter   |      82 | Hulu        |
|    22558 | Palm Springs                            | Romance   | English  |    94 | tomatoMeter   |      90 | Hulu        |
|   132904 | The Two Popes                           | Drama     | English  |    90 | tomatoMeter   |     126 | Netflix     |
|   116501 | The Lost Daughter                       | Drama     | English  |    94 | tomatoMeter   |     122 | Netflix     |
|   113051 | The Meyerowitz Stories New and Selected | Comedy    | English  |    93 | tomatoMeter   |     110 | Netflix     |
|   103242 | The Little Prince                       | Adventure | English  |    92 | tomatoMeter   |     106 | Netflix     |
|    87620 | The Irishman                            | Crime     | English  |    95 | tomatoMeter   |     209 | Netflix     |
|    70239 | Marriage Story                          | Drama     | English  |    95 | tomatoMeter   |     136 | Netflix     |
|    69153 | The FortyYearOld Version                | Comedy    | English  |    99 | tomatoMeter   |     123 | Netflix     |
|    41489 | El Camino A Breaking Bad Movie          | Crime     | English  |    92 | tomatoMeter   |     122 | Netflix     |
|    40667 | Private Life                            | Comedy    | English  |    94 | tomatoMeter   |     124 | Netflix     |
|    39563 | Hustle                                  | Drama     | English  |    93 | tomatoMeter   |     118 | Netflix     |
|     1360 | Passing                                 | Drama     | English  |    90 | tomatoMeter   |      98 | Netflix     |
+----------+-----------------------------------------+-----------+----------+-------+---------------+---------+-------------+
```

Result:

Among the considered distributors, Netflix offers the largest number of movies that align with my preferences, followed by HULU and HBO Max. However, this question remains unanswered as the accuracy of the "distributor" column does not align with my question's need. For this question, the column 'distributor' was used, but my inquiry focused more on streaming platforms rather than distributors. The considered platforms were Netflix, Amazon Prime Video, Disney+, Hulu, HBO Max, and Paramount+, which serve as both streaming platforms and distributors. However, the majority of movies available on these platforms come from different distributors making ‘distributor’ column not suitable for this case. The given answer would only make sense if I were seeking for which distributor is more likely to offer movies of my preference in the future. Therefore, the question remains unanswered.

Images of my webpags:

Index.html

Home Question1 Question2 Question3

Table with all the fields

Title_id	Title	Genre	Language	Rating	score	Critic	Genre	Writer	Director	Distributor
134288	Charlies Angels	Comedy	English	PG-13	78	audienceScore	Comedy	Evan Spiliotopoulos	Elizabeth Banks	Sony Pictures Entertainment
134288	Charlies Angels	Adventure	English	PG-13	78	audienceScore	Adventure	Evan Spiliotopoulos	Elizabeth Banks	Sony Pictures Entertainment
134288	Charlies Angels	Action	English	PG-13	78	audienceScore	Action	Evan Spiliotopoulos	Elizabeth Banks	Sony Pictures Entertainment
134334	SpiderMan 3	Fantasy	English	PG-13	51	audienceScore	Fantasy	Ivan Raimi	Sam Raimi	Sony Pictures Entertainment
134334	SpiderMan 3	Adventure	English	PG-13	51	audienceScore	Adventure	Ivan Raimi	Sam Raimi	Sony Pictures Entertainment
134334	SpiderMan 3	Action	English	PG-13	51	audienceScore	Action	Ivan Raimi	Sam Raimi	Sony Pictures Entertainment
134334	SpiderMan 3	Fantasy	English	PG-13	51	audienceScore	Fantasy	Sam Raimi	Sam Raimi	Sony Pictures Entertainment
134334	SpiderMan 3	Adventure	English	PG-13	51	audienceScore	Adventure	Sam Raimi	Sam Raimi	Sony Pictures Entertainment
134334	SpiderMan 3	Action	English	PG-13	51	audienceScore	Action	Sam Raimi	Sam Raimi	Sony Pictures Entertainment
134334	SpiderMan 3	Fantasy	English	PG-13	51	audienceScore	Fantasy	Alvin Sargent	Sam Raimi	Sony Pictures Entertainment
134334	SpiderMan 3	Adventure	English	PG-13	51	audienceScore	Adventure	Alvin Sargent	Sam Raimi	Sony Pictures Entertainment
134334	SpiderMan 3	Action	English	PG-13	51	audienceScore	Action	Alvin Sargent	Sam Raimi	Sony Pictures Entertainment
135179	The Unholy	Horror	English	PG-13	56	audienceScore	Horror	Evan Spiliotopoulos	Evan Spiliotopoulos	Sony Pictures Entertainment
135179	The Unholy	Mystery & thriller	English	PG-13	56	audienceScore	Mystery & thriller	Evan Spiliotopoulos	Evan Spiliotopoulos	Sony Pictures Entertainment

Question1.html

Home Question1 Question2 Question3

Which movies have a Tomatometer or audience score exceeding 90, are in English or Korea, have a runtime greater than 100 minutes, and fall under the genres of comedy, drama, action, romance, crime, adventure, or sci-if?

Title_id	Title	Genre	Language	Score	Runtime	Critic
39111	My Fiona	Drama	English	100	88	audienceScore
39101	Snatch	Comedy	English	93	98	audienceScore
38843	SelfDelusion and Other Obstacles	Comedy	English	100	90	audienceScore
38484	Shadowlands	Drama	English	92	90	audienceScore
38287	Beauty Mark	Drama	English	94	87	audienceScore
38259	The Big Lebowski	Comedy	English	93	117	audienceScore
38170	Chasing Yesterday	Comedy	English	100	91	audienceScore
38005	10E	Comedy	English	100	94	audienceScore
37943	Trainspotting	Comedy	English	93	94	audienceScore
37738	Stroszek	Drama	English	92	108	audienceScore
37646	Lifted	Drama	English	91	108	audienceScore
37595	Ill Never Forget My High School Friends	Comedy	English	94	98	audienceScore
37260	Sex Weather	Comedy	English	99	90	audienceScore
37109	Annie Hall	Romance	English	92	93	audienceScore
36541	Brothers Keeper	Drama	English	100	85	audienceScore
36095	Popovich and the Voice of the Fabled American West	Comedy	English	100	90	audienceScore
35969	The Killer	Action	Korean	91	94	audienceScore
35736	The Clark Sisters First Ladies of Gospel	Drama	English	100	90	audienceScore
35500	The Ditchdiggers Daughters	Drama	English	93	97	audienceScore
35381	The Biscuit Eater	Drama	English	100	83	audienceScore
35203	Paper Moon	Comedy	English	94	102	audienceScore
34742	American Bistro	Comedy	English	97	95	audienceScore
34670	Lantern Hill	Drama	English	92	111	audienceScore

Question2.html

Home Question1 Question2 Question3

Among the movies directed by Christopher Nolan, which writer has he collaborated with most frequently?

Number of times 'Christopher Nolan' has collaborated with each writer

Writer	Number_of_movies
Bob Kane	1
Christopher Nolan	12
David S. Goyer	1
Hillary Seitz	1
Jonathan Nolan	5

List of movies directed by 'Christopher Nolan'

Title_id	Title	Director	Writer
6842	The Dark Knight	Christopher Nolan	Jonathan Nolan
6842	The Dark Knight	Christopher Nolan	Christopher Nolan
11164	The Dark Knight Rises	Christopher Nolan	Jonathan Nolan
11164	The Dark Knight Rises	Christopher Nolan	Christopher Nolan
13446	The Prestige	Christopher Nolan	Jonathan Nolan
13446	The Prestige	Christopher Nolan	Christopher Nolan
17152	Dunkirk	Christopher Nolan	Christopher Nolan
28475	Tenet	Christopher Nolan	Christopher Nolan
31928	Batman Begins	Christopher Nolan	Christopher Nolan
31928	Batman Begins	Christopher Nolan	David S. Goyer
31928	Batman Begins	Christopher Nolan	Bob Kane
71529	Insomnia	Christopher Nolan	Hillary Seitz
80193	Interstellar	Christopher Nolan	Jonathan Nolan

Question3.html

Home Question1 Question2 Question3

Which streaming Platform offers the highest number of movies that match my preferences?

Number of movies of my preferences each platform provides

Distributor	Movie_count
Netflix	15
Hulu	2
HBO Max	2

List of movies of my preferences and their distributor

Title_id	Title	Genre	Language	Score	Critic	Runtime	Distributor
112765	Son of Monarchs	Drama	English	100	audienceScore	97	HBO Max
134254	Red Notice	Action	English	92	audienceScore	117	Netflix
39563	Hustle	Drama	English	93	audienceScore	118	Netflix
17026	Faraway	Romance	English	91	audienceScore	109	Netflix
4186	The Gray Man	Action	English	90	audienceScore	129	Netflix
112765	Son of Monarchs	Drama	English	92	tomatoMeter	97	HBO Max
106385	Rye Lane	Romance	English	98	tomatoMeter	82	Hulu
22558	Palm Springs	Romance	English	94	tomatoMeter	90	Hulu
132904	The Two Popes	Drama	English	90	tomatoMeter	126	Netflix
116501	The Lost Daughter	Drama	English	94	tomatoMeter	122	Netflix
113051	The Meyerowitz Stories New and Selected	Comedy	English	93	tomatoMeter	110	Netflix
103242	The Little Prince	Adventure	English	92	tomatoMeter	106	Netflix
87620	The Irishman	Crime	English	95	tomatoMeter	209	Netflix
70239	Marriage Story	Drama	English	95	tomatoMeter	136	Netflix
69153	The FortyYearOld Version	Comedy	English	99	tomatoMeter	123	Netflix
41489	El Camino A Breaking Bad Movie	Crime	English	92	tomatoMeter	122	Netflix
40667	Private Life	Comedy	English	94	tomatoMeter	124	Netflix

Link to my lab:

https://hub.labs.coursera.org:443/connect/sharedufstlwpb?forceRefresh=false&path=%2F%3Ffolder%3D%2Fhome%2Fcoder%2Fproject&isLabVersioning=true

Appendix:

The following document is the data cleaning process done in Jupyter notebook:

Database_rotten_tomatotes

June 25, 2023

Database mid term

Data cleaning

Cleaning the rotten_tomatoes dataset and droping some columns that are not required.

[1]:
```
import pandas as pd
import numpy as np
import re
```

[2]:
```
rotten_tomatoes = pd.read_csv('rotten_tomatoes.csv')
rotten_tomatoes.columns
```

[2]:
```
Index(['id', 'title', 'audienceScore', 'tomatoMeter', 'rating',
       'ratingContents', 'releaseDateTheaters', 'releaseDateStreaming',
       'runtimeMinutes', 'genre', 'originalLanguage', 'director', 'writer',
       'boxOffice', 'distributor', 'soundMix'],
      dtype='object')
```

[3]:
```
rotten_tomatoes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 143258 entries, 0 to 143257
Data columns (total 16 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   id                    143258 non-null  object
 1   title                 142891 non-null  object
 2   audienceScore         73248 non-null   float64
 3   tomatoMeter           33877 non-null   float64
 4   rating                13991 non-null   object
 5   ratingContents        13991 non-null   object
 6   releaseDateTheaters   30773 non-null   object
 7   releaseDateStreaming  79420 non-null   object
 8   runtimeMinutes        129431 non-null  float64
 9   genre                 132175 non-null  object
 10  originalLanguage      129400 non-null  object
 11  director              139064 non-null  object
 12  writer                90116 non-null   object
 13  boxOffice             14743 non-null   object
```

1

```
 14  distributor           23005 non-null   object
 15  soundMix              15917 non-null   object
dtypes: float64(3), object(13)
memory usage: 17.5+ MB
```

I am deleting the columns 'boxoffice', 'soundmix', 'releaseDateStreaming', and 'releaseDateTheaters' from the dataset. These columns are not relevant for providing answers to my questions.

[4]:
```
delete_columns =␣
  ↪['boxOffice','soundMix','ratingContents','releaseDateStreaming','releaseDateTheaters']
rotten_tomatoes = rotten_tomatoes.drop(columns=delete_columns, axis=1)
rotten_tomatoes.columns
```

[4]:
```
Index(['id', 'title', 'audienceScore', 'tomatoMeter', 'rating',
       'runtimeMinutes', 'genre', 'originalLanguage', 'director', 'writer',
       'distributor'],
      dtype='object')
```

[5]:
```
rotten_tomatoes.rename(columns={"originalLanguage": "languages"}, inplace=True)
rotten_tomatoes.rename(columns={"runtimeMinutes": "runtime"}, inplace=True)
rotten_tomatoes.rename(columns={"id": "title_id"}, inplace=True)
rotten_tomatoes.rename(columns={"languages": "language"}, inplace=True)
rotten_tomatoes.rename(columns={"languages": "language"}, inplace=True)

rotten_tomatoes.head()
```

[5]:

	title_id	title	audienceScore	tomatoMeter	rating	runtime	genre	language	director	writer	distributor
0	space-zombie-bingo	Space Zombie Bingo!	50.0	NaN	NaN	75.0	Comedy, Horror, Sci-fi	English	George Ormrod	George Ormrod,John Sabotta	
1	the_green_grass	The Green Grass	NaN	NaN	NaN	114.0	Drama	English	Tiffany Edwards	Tiffany Edwards	
2	love_lies	Love, Lies	43.0	NaN	NaN	120.0	Drama	Korean	Park Heung-Sik,Heung-Sik Park	Ha Young-Joon,Jeon Yun-su,Song Hye-jin	
3	the_sore_losers_1997	Sore Losers	60.0	NaN	NaN	90.0	Action, Mystery & thriller	English	John Michael McCarthy	John Michael McCarthy	
4	dinosaur_island_2002	Dinosaur Island	70.0	NaN	NaN	80.0	Fantasy, Adventure, Animation	English	Will Meugniot	John Loy	

```
0         NaN
1         NaN
2         NaN
3         NaN
4         NaN
```

Spliting the columns that are not single valued.

These columns are director, genre, writer, distributor

Even though the data types of the columns are marked as "object", the split function cannot be performed if there are NaN values present. This error occurs because the split function expects a string as input, and NaN values cannot be split. Therefore, it is important to handle these missing values appropriately before applying the split operation.To resolve this issue, I will be replacing NaN values with empty strings.

[6]:
```
list_columns=['audienceScore','tomatoMeter','rating','runtime','genre','language','director','
rotten_tomatoes[list_columns] = rotten_tomatoes[list_columns].fillna('')
```

[7]:
```
rotten_tomatoes['genre'] = rotten_tomatoes["genre"].apply(lambda x: x.
 ↪split(","))
rotten_tomatoes['director'] = rotten_tomatoes["director"].apply(lambda x: x.
 ↪split(","))
rotten_tomatoes['writer'] = rotten_tomatoes["writer"].apply(lambda x: x.
 ↪split(","))
rotten_tomatoes['distributor'] = rotten_tomatoes["distributor"].apply(lambda x:␣
 ↪x.split(","))
rotten_tomatoes.head(5)
```

[7]:
```
               title_id                title audienceScore tomatoMeter rating  \
0    space-zombie-bingo  Space Zombie Bingo!          50.0
1       the_green_grass      The Green Grass
2             love_lies           Love, Lies          43.0
3  the_sore_losers_1997          Sore Losers          60.0
4  dinosaur_island_2002      Dinosaur Island          70.0

  runtime                                genre language  \
0    75.0         [Comedy,  Horror,  Sci-fi]  English
1   114.0                            [Drama]  English
2   120.0                            [Drama]   Korean
3    90.0      [Action,  Mystery & thriller]  English
4    80.0  [Fantasy,  Adventure,  Animation]  English

                          director  \
0                  [George Ormrod]
1                [Tiffany Edwards]
2  [Park Heung-Sik, Heung-Sik Park]
3           [John Michael McCarthy]
```

```
4                    [Will Meugniot]

                                        writer distributor
0                [George Ormrod, John Sabotta]          []
1                             [Tiffany Edwards]          []
2  [Ha Young-Joon, Jeon Yun-su, Song Hye-jin]          []
3                      [John Michael McCarthy]          []
4                                   [John Loy]          []
```

In order to ensure that the language filters only recognize the language and not the region, it is necessary to remove the brackets indicating the part of the language used in the movie. Only the language itself should be considered

[8]:
```
rotten_tomatoes['language'].iloc[35:45]
```

[8]:
```
35            English
36            English
37            English
38            English
39            English
40            Spanish
41            English
42    French (Canada)
43            Chinese
44
Name: language, dtype: object
```

[9]:
```
rotten_tomatoes['language'] = rotten_tomatoes['language'].astype(str)
rotten_tomatoes['language'] = rotten_tomatoes['language'].apply(lambda x: re.
 ↪sub("\(.*?\)|\[.*?\]","",x))
rotten_tomatoes['language'].iloc[35:45]
```

[9]:
```
35    English
36    English
37    English
38    English
39    English
40    Spanish
41    English
42    French
43    Chinese
44
Name: language, dtype: object
```

I also found out that the movie titles contains some special characters so they needed to be removed.

[10]:
```
rotten_tomatoes['title']=rotten_tomatoes['title'].astype(str)
```

```
rotten_tomatoes['title'] =rotten_tomatoes['title'].apply(lambda x:re.
↪sub(r'[^a-zA-Z0-9\s]+', '', x))
```

I have found out that there are duplicate entries in the "title_id" field, leading to non-unique records. In order to resolve this issue, I have implemented an auto-incrementing integer for the "title_id" column, ensuring that each record has a unique identifier.

[11]:
```
rotten_tomatoes['title_id'] = [x for x in range(1, len(rotten_tomatoes.values)␣
↪+ 1)]
rotten_tomatoes.head(10)
```

[11]:

	title_id	title	audienceScore	tomatoMeter	rating	runtime	genre	language	director
0	1	Space Zombie Bingo	50.0			75.0	[Comedy, Horror, Sci-fi]	English	[George Ormrod]
1	2	The Green Grass				114.0	[Drama]	English	[Tiffany Edwards]
2	3	Love Lies	43.0			120.0	[Drama]	Korean	[Park Heung-Sik, Heung-Sik Park]
3	4	Sore Losers	60.0			90.0	[Action, Mystery & thriller]	English	[John Michael McCarthy]
4	5	Dinosaur Island	70.0			80.0	[Fantasy, Adventure, Animation]	English	[Will Meugniot]
5	6	Adrift	65.0	69.0	PG-13	120.0	[Adventure, Drama, Romance]	English	[Baltasar Kormákur]
6	7	Scrambled Beer	55.0			88.0	[Comedy]	Spanish	[Cristobal Valderrama]
7	8	Kakabakaba ka ba Will Your Heart Beat Faster	88.0				[]		[Mike de Leon]
8	9	Sundowning				123.0	[Drama]	English	[Jim Comas Cole]
9	10	Born to Kill	74.0	83.0		92.0	[Crime, Drama]	English	[Robert Wise]

```
                                                writer  distributor
0                        [George Ormrod, John Sabotta]           []
1                                    [Tiffany Edwards]           []
2            [Ha Young-Joon, Jeon Yun-su, Song Hye-jin]           []
3                              [John Michael McCarthy]           []
4                                           [John Loy]           []
5  [Aaron Kandell, Jordan Kandell, David Branson …  [STX Films]
6                              [Cristobal Valderrama]           []
7                                                  []           []
8                                                  []           []
9                        [Eve Greene, Richard Macaulay]           []
```

[12]:
```
rotten_tomatoes=rotten_tomatoes.explode('director')
rotten_tomatoes=rotten_tomatoes.explode('genre')
rotten_tomatoes=rotten_tomatoes.explode('writer')
rotten_tomatoes=rotten_tomatoes.explode('distributor')
rotten_tomatoes.head(10)
```

[12]:
```
   title_id              title audienceScore tomatoMeter rating runtime  \
0         1  Space Zombie Bingo          50.0                        75.0
0         1  Space Zombie Bingo          50.0                        75.0
0         1  Space Zombie Bingo          50.0                        75.0
0         1  Space Zombie Bingo          50.0                        75.0
0         1  Space Zombie Bingo          50.0                        75.0
0         1  Space Zombie Bingo          50.0                        75.0
1         2     The Green Grass                                      114.0
2         3           Love Lies          43.0                       120.0
2         3           Love Lies          43.0                       120.0
2         3           Love Lies          43.0                       120.0

     genre language         director           writer distributor
0   Comedy  English    George Ormrod    George Ormrod
0   Comedy  English    George Ormrod     John Sabotta
0   Horror  English    George Ormrod    George Ormrod
0   Horror  English    George Ormrod     John Sabotta
0   Sci-fi  English    George Ormrod    George Ormrod
0   Sci-fi  English    George Ormrod     John Sabotta
1    Drama  English  Tiffany Edwards  Tiffany Edwards
2    Drama   Korean   Park Heung-Sik    Ha Young-Joon
2    Drama   Korean   Park Heung-Sik      Jeon Yun-su
2    Drama   Korean   Park Heung-Sik     Song Hye-jin
```

Only then, the empty rows are replaced with NaN with the condition of columns that are not data type of integer or float .

[13]:
```
columnlist1 = ['title','genre', 'language', 'director', 'writer',␣
 ↪'distributor', 'rating']
rotten_tomatoes[columnlist1] = rotten_tomatoes[columnlist1].replace('', np.nan)
```

```
rotten_tomatoes.head(10)
```

[13]:

	title_id	title	audienceScore	tomatoMeter	rating	runtime
0	1	Space Zombie Bingo	50.0		NaN	75.0
0	1	Space Zombie Bingo	50.0		NaN	75.0
0	1	Space Zombie Bingo	50.0		NaN	75.0
0	1	Space Zombie Bingo	50.0		NaN	75.0
0	1	Space Zombie Bingo	50.0		NaN	75.0
0	1	Space Zombie Bingo	50.0		NaN	75.0
1	2	The Green Grass			NaN	114.0
2	3	Love Lies	43.0		NaN	120.0
2	3	Love Lies	43.0		NaN	120.0
2	3	Love Lies	43.0		NaN	120.0

	genre	language	director	writer	distributor
0	Comedy	English	George Ormrod	George Ormrod	NaN
0	Comedy	English	George Ormrod	John Sabotta	NaN
0	Horror	English	George Ormrod	George Ormrod	NaN
0	Horror	English	George Ormrod	John Sabotta	NaN
0	Sci-fi	English	George Ormrod	George Ormrod	NaN
0	Sci-fi	English	George Ormrod	John Sabotta	NaN
1	Drama	English	Tiffany Edwards	Tiffany Edwards	NaN
2	Drama	Korean	Park Heung-Sik	Ha Young-Joon	NaN
2	Drama	Korean	Park Heung-Sik	Jeon Yun-su	NaN
2	Drama	Korean	Park Heung-Sik	Song Hye-jin	NaN

Creating and exporting to CSV file

The cleaned rotten_tomatoes database is saved as csv file.

[14]:
```
DATA_PATH1='/Users/julie/Desktop/Database_mid_term '
rotten_tomatoes.to_csv(f"{DATA_PATH1}/processed_rotten_tomatoes.csv",␣
↪index=False)
```

For the remaining part of the file, I will create suitable dataframes that can be exported as CSV files and then inserted into the corresponding SQL tables.

[15]:
```
DATA_PATH='/Users/julie/Desktop/database_sub_csvfiles'
```

Language table

[16]:
```
language = pd.DataFrame(data=rotten_tomatoes['language'].unique(),␣
↪columns=['language'])
language.reset_index(inplace=True)
language.columns = ['language_id', 'language']
language.dropna(inplace=True)
language = language.drop_duplicates(subset=['language'])
language.to_csv(f"{DATA_PATH}/language.csv", index=False)
language.head(10)
```

[16]:

	language_id	language
0	0	English
1	1	Korean
2	2	Spanish
4	4	Tagalog
5	5	Hindi
6	6	French
7	7	Japanese
8	8	Tamil
9	9	Chinese
10	10	Arabic

Rating table

[17]:
```
rating = pd.DataFrame(data=rotten_tomatoes['rating'].unique(),␣
↪columns=['rating'])
rating.reset_index(inplace=True)
rating.columns = ['rating_id', 'rating']
rating.dropna(inplace=True)
rating.to_csv(f"{DATA_PATH}/rating.csv", index=False)
rating.head(10)
```

[17]:

	rating_id	rating
1	1	PG-13
2	2	TVPG
3	3	R
4	4	PG
5	5	TV14
6	6	NC-17
7	7	TVG
8	8	TVMA
9	9	TVY7
10	10	G

Genre table

[18]:
```
genre = pd.DataFrame(data=rotten_tomatoes['genre'].unique(), columns=['genre'])
genre.reset_index(inplace=True)
genre.columns = ['genre_id', 'genre']
genre.dropna(inplace=True)
genre = genre.drop_duplicates(subset='genre')
genre.reset_index(drop=True, inplace=True)
genre.to_csv(f"{DATA_PATH}/genre.csv", index=False)
genre.head(10)
```

[18]:

	genre_id	genre
0	0	Comedy
1	1	Horror

2	2	Sci-fi
3	3	Drama
4	4	Action
5	5	Mystery & thriller
6	6	Fantasy
7	7	Adventure
8	8	Animation
9	9	Adventure

Director table

[19]:
```
director = pd.DataFrame(data=rotten_tomatoes['director'].unique(),␣
↪columns=['director'])
director.reset_index(inplace=True)
director.columns = ['director_id', 'director']
director.dropna(inplace=True)
director = director.drop_duplicates(subset='director')
director.reset_index(drop=True, inplace=True)
director.to_csv(f"{DATA_PATH}/director.csv", index=False)
director.head(10)
```

[19]:

	director_id	director
0	0	George Ormrod
1	1	Tiffany Edwards
2	2	Park Heung-Sik
3	3	Heung-Sik Park
4	4	John Michael McCarthy
5	5	Will Meugniot
6	6	Baltasar Kormákur
7	7	Cristobal Valderrama
8	8	Mike de Leon
9	9	Jim Comas Cole

Distributor table

[20]:
```
distributor = pd.DataFrame(data=rotten_tomatoes['distributor'].unique(),␣
↪columns=['distributor'])
distributor.reset_index(inplace=True)
distributor.columns = ['distributor_id', 'distributor']
distributor.dropna(inplace=True)
distributor_unique = distributor.drop_duplicates(subset=['distributor'])
distributor.to_csv(f"{DATA_PATH}/distributor.csv", index=False)
distributor.head(10)
```

[20]:

	distributor_id	distributor
1	1	STX Films
2	2	Passport Pictures
3	3	Big Pictures

```
4                4          Freestyle Releasing
5                5                Kino Pictures
6                6        Reliance Big Pictures
7                7                  Saban Films
8                8             Outsider Pictures
9                9  Sony Pictures Entertainment
10              10              20th Century Fox
```

Writer table

[21]:
```
writer = pd.DataFrame(data=rotten_tomatoes['writer'].unique(),␣
↪columns=['writer'])
writer.reset_index(inplace=True)
writer.columns = ['writer_id', 'writer']
writer.dropna(inplace=True)
writer = writer.drop_duplicates(subset='writer')
writer.reset_index(drop=True, inplace=True)
writer.to_csv(f"{DATA_PATH}/writer.csv", index=False)
writer.head(10)
```

[21]:

	writer_id	writer
0	0	George Ormrod
1	1	John Sabotta
2	2	Tiffany Edwards
3	3	Ha Young-Joon
4	4	Jeon Yun-su
5	5	Song Hye-jin
6	6	John Michael McCarthy
7	7	John Loy
8	8	Aaron Kandell
9	9	Jordan Kandell

Category table

[22]:
```
category = rotten_tomatoes.merge(genre, how='left', left_on='genre',␣
↪right_on='genre')
category = category[['title_id', 'genre_id']]
category['genre_id'] = category['genre_id'].astype(float).astype('Int64')
category.drop_duplicates(subset=['title_id', 'genre_id'], inplace=True)
category.dropna(inplace=True)
category.to_csv(f"{DATA_PATH}/category.csv", index=False)
category.head(10)
```

[22]:

	title_id	genre_id
0	1	0
2	1	1
4	1	2
6	2	3

7	3	3
13	4	4
14	4	5
15	5	6
16	5	7
17	5	8

Directed_by table

[23]:
```
directed_by = rotten_tomatoes.merge(director, how='left', left_on='director',␣
↪right_on='director')
directed_by = directed_by[['title_id', 'director_id']]
directed_by['director_id'] = directed_by['director_id'].astype(float).
↪astype('Int64')
directed_by.drop_duplicates(subset=['title_id', 'director_id'], inplace=True)
directed_by.dropna(inplace=True)
directed_by.to_csv(f"{DATA_PATH}/directed_by.csv", index=False)
directed_by.head(10)
```

[23]:

	title_id	director_id
0	1	0
6	2	1
7	3	2
10	3	3
13	4	4
15	5	5
18	6	6
27	7	7
28	8	8
29	9	9

written_by table

[24]:
```
written_by = rotten_tomatoes.merge(writer, how='left', left_on='writer',␣
↪right_on='writer')
written_by = written_by[['title_id', 'writer_id']]
written_by['writer_id'] = written_by['writer_id'].astype('Int64')
written_by.drop_duplicates(subset=['title_id', 'writer_id'], inplace=True)
written_by.dropna(inplace=True)
written_by.to_csv(f"{DATA_PATH}/written_by.csv", index=False)
written_by.head(10)
```

[24]:

	title_id	writer_id
0	1	0
1	1	1
6	2	2
7	3	3
8	3	4

9	3	5
13	4	6
15	5	7
18	6	8
19	6	9

Distributed_by table

[25]:
```
distributed_by = rotten_tomatoes.merge(distributor, how='left',␣
 ↪left_on='distributor', right_on='distributor')
distributed_by = distributed_by[['title_id', 'distributor_id']]
distributed_by['distributor_id'] = distributed_by['distributor_id'].
 ↪astype('Int64')
distributed_by.drop_duplicates(subset=['title_id', 'distributor_id'],␣
 ↪inplace=True)
distributed_by.dropna(inplace=True)
distributed_by.to_csv(f"{DATA_PATH}/distributed_by.csv", index=False)
distributed_by.head(10)
```

[25]:

	title_id	distributor_id
18	6	1
37	13	2
49	18	3
52	21	4
73	32	5
94	46	6
114	50	7
168	74	8
179	83	9
182	85	10

Title table

[26]:
```
title = rotten_tomatoes.copy()
dropcolumns = ['language', 'audienceScore', 'tomatoMeter', 'genre', 'director',␣
 ↪'writer', 'distributor', 'rating']
title = title.merge(language, how='left', left_on='language',␣
 ↪right_on='language')
title = title.merge(rating, how='left', left_on='rating', right_on='rating')
title = title.drop(columns=dropcolumns)


title['language_id'] = title['language_id'].astype('Int64')
title['rating_id'] = title['rating_id'].astype('Int64')

title['rating_id'] = title['rating_id'].astype(str)
title['language_id'] = title['language_id'].astype(str)
```

```
title['rating_id'] = title['rating_id'].apply(lambda x: None if x == "<NA>"␣
 ↪else x)
title['language_id'] = title['language_id'].apply(lambda x: None if x == "<NA>"␣
 ↪else x)

title.drop_duplicates(inplace=True)

title.to_csv(f"{DATA_PATH}/title.csv", index=False)
title.head(10)
```

[26]:

	title_id	title	runtime	\
0	1	Space Zombie Bingo	75.0	
6	2	The Green Grass	114.0	
7	3	Love Lies	120.0	
13	4	Sore Losers	90.0	
15	5	Dinosaur Island	80.0	
18	6	Adrift	120.0	
27	7	Scrambled Beer	88.0	
28	8	Kakabakaba ka ba Will Your Heart Beat Faster		
29	9	Sundowning	123.0	
30	10	Born to Kill	92.0	

	language_id	rating_id
0	0	None
6	0	None
7	1	None
13	0	None
15	0	None
18	0	1
27	2	None
28	None	None
29	0	None
30	0	None

Critics table

[27]:
```
critic = pd.DataFrame({'critic_id': [1, 2], 'critic': ['audienceScore',␣
 ↪'tomatoMeter']})
critic.to_csv(f"{DATA_PATH}/critic.csv", index=False)
critic.head()
```

[27]:

	critic_id	critic
0	1	audienceScore
1	2	tomatoMeter

Score table

The audienceScore and tomatoMeter columns of rotten_tomatoes are joined in to a single column called score in score dataframe.

[28]:
```
score = pd.DataFrame(data=rotten_tomatoes['title_id'].unique(),␣
 ↪columns=['title_id'])

audience_scores = pd.DataFrame()
audience_scores['title_id'] = rotten_tomatoes['title_id']
audience_scores['title'] = rotten_tomatoes['title']
audience_scores['score_type'] = 'audience'
audience_scores['score'] = rotten_tomatoes['audienceScore']

tomato_meters = pd.DataFrame()
tomato_meters['title_id'] = rotten_tomatoes['title_id']
tomato_meters['title'] = rotten_tomatoes['title']
tomato_meters['score_type'] = 'tomato'
tomato_meters['score'] = rotten_tomatoes['tomatoMeter']

score = pd.concat([audience_scores, tomato_meters], ignore_index=True)
score.drop_duplicates(subset=['title_id', 'title','score_type','score'],␣
 ↪inplace=True)
score = score[score["score"] != ""]
score.sort_values(by=['title']).head(10)
```

[28]:

	title_id	title	score_type	score
267876	109796	And They Lived Happily Ever After	audience	71.0
617089	109796	And They Lived Happily Ever After	tomato	57.0
554612	83925	Dollars	tomato	86.0
205399	83925	Dollars	audience	49.0
269993	110697	Propos de Nice	audience	74.0
260059	106470	Tick Tick Tick	audience	57.0
283446	116251	Vos Marques Party	audience	72.0
188535	76995	coup sr	audience	11.0
121124	49753	ing	audience	86.0
121535	49915	ing	audience	86.0

The critic column is created by referencing the score_type column. After that, unnecessary columns are dropped

[29]:
```
score["critic"] = score["score_type"].apply(lambda x: "audienceScore" if x ==␣
 ↪"audience" else "tomatoMeter" if x == "tomato" else "Others")
score = score[["title_id", "critic", "score"]]
score = score[score["score"] != " "]
score["score"] = score["score"].astype(float)
score.sort_values(by=['title_id']).head(10)
```

[29]:

	title_id	critic	score
0	1	audienceScore	50.0
7	3	audienceScore	43.0
13	4	audienceScore	60.0
15	5	audienceScore	70.0

```
349231         6    tomatoMeter   69.0
18             6  audienceScore   65.0
27             7  audienceScore   55.0
28             8  audienceScore   88.0
30            10  audienceScore   74.0
349243        10    tomatoMeter   83.0
```

Now, I can merge the data with the critic table to display the correct critic_id corresponding to each entry's critic

[31]:
```
score = score.merge(critic, how="left", left_on='critic', right_on='critic')
score = score[["title_id", "critic_id", "score"]]
score['score']=score['score'].astype('Int64')
score.to_csv(f"{DATA_PATH}/score.csv", index=False)
score.head(10)
```

[31]:

	title_id	critic_id	score
0	1	1	50
1	3	1	43
2	4	1	60
3	5	1	70
4	6	1	65
5	7	1	55
6	8	1	88
7	10	1	74
8	11	1	19
9	14	1	86

[ ]: