

UNIVERSITY OF  
**Waterloo**



## ECE 356 Project - Movies

Group 19

Adam Lam, a87lam

Eddy Wu, e33wu

Matthew Deng, m29deng

Github link: <https://git.uwaterloo.ca/ece356-f2023/Team-19>

# Table of Contents

<b>Introduction:</b>	<b>3</b>
<b>ER Design:</b>	<b>4</b>
Diagram:	4
ER Model design choices:	4
<b>Relational Schema:</b>	<b>6</b>
<b>Test Plan:</b>	<b>8</b>
<b>Client Application:</b>	<b>10</b>
<b>Data Mining Investigation:</b>	<b>12</b>

## Introduction:

The purpose of this project was to create a database that models relationships between various facts and information about a select dataset. For our project, we have selected movies. In this report, there will consist of an ER model, a corresponding relational schema, client application and a data mining investigation. The ER model and relational schema that we implement in our SQL database will be derived from provided raw CSV files/Kaggle data sources, figuring out what information will be important to include within our project. The ER model gives us an overall view of the data we will be working with and what kind of relationships each entity has. The schema will translate this model into a working database, in which we will create all our tables from the entities in the ER model and load all necessary information into them. Our client application will be written in Python that allows users to interact with our database to get information about movies. And lastly, our data mining investigation, in which we will be investigating the relationship between the budget of a movie vs the amount of income it grosses.

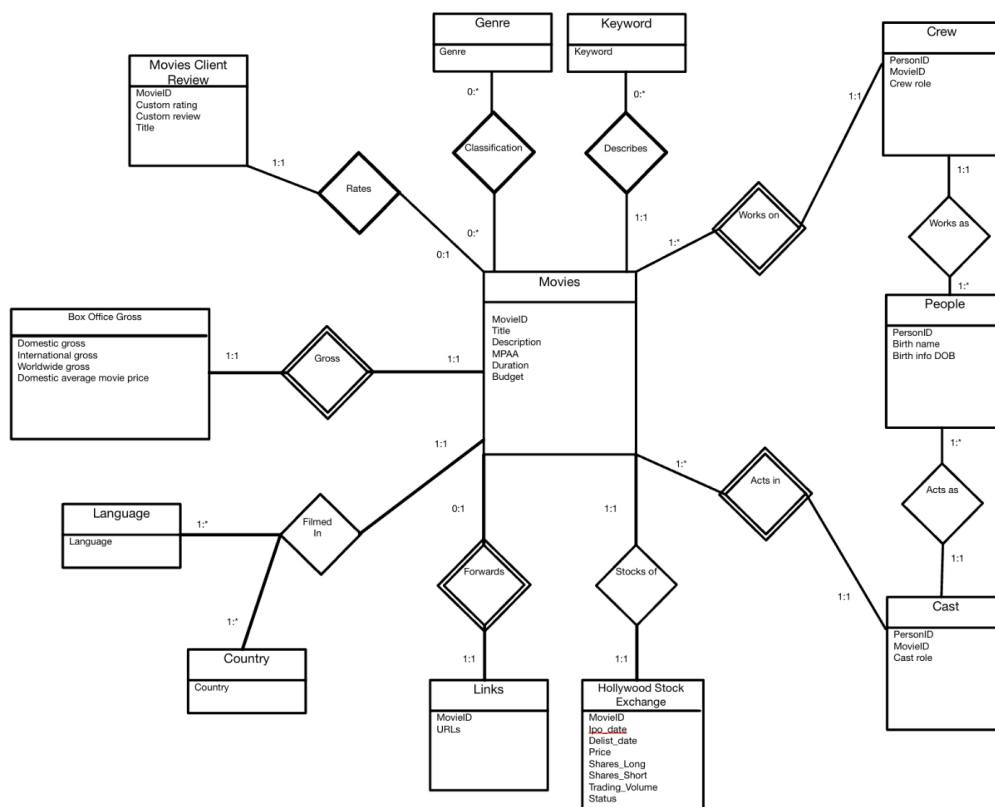
All files related to this project can be found in the gitLab link provided below:

<https://git.uwaterloo.ca/ece356-f2023/Team-19>

# ER Design:

## Diagram:

An image of the ER model diagram has been included below, however, a PDF file of the model has also been included in the repository. We see that we have 11 entities: Box Office Gross, Genre, Keyword, Crew, People, Cast, Links, Country, Language, Movies Client Review, Hollywood Stock Exchange and Links. We also have 11 relationships that relate each entity to some other, 4 out of the 11 being weak entity relationships due to the entity it's related to being a weak entity, this is seen by a double diamond box. Cardinalities between entities and relationships are also indicated on the lines that connect entities together.



## ER Model design choices:

- Links, Languages, Country, Keywords, and genre are all separate entities since they don't necessarily reflect an attribute of a movie. An example would be like Language is not related to a movie itself, it is just its own description/portion.

A language is part of every movie, but it is not necessarily an attribute of said movie.

- We have a general People entity set that is related to the Crew and Cast entities, and the Crew and Cast entities are related to Movies itself. This design choice was implemented for understanding and not necessarily simplicity. It is easier to understand that Cast and Crew are both related to People, but the two roles are not the same. It can be argued that the two roles can be grouped together under one generalization, but under CSV files, they can be found under different rows, and hence it is easier to split into two entities.
- Language and Country entities have the same relationship with Movies. This is because they're relationship can be seen as relatively similar.
- Some data has been scrapped from our model since we are under the assumption that clients are making simple and basic searches about movies and information related to the movie. Information like links can be seen as irrelevant to users who just want to find basic information on certain movies.
- Box Office Gross, links, Cast and Crew have a weak entity relationship with Movies itself because it cannot be uniquely identified by its own attributes alone.

## Relational Schema:

All the relational schema related code can be found in the GitLab repository under the file `Server.sql`. Having a relational schema is very important for designing the database. We created our relational schema by translating it from the ER model we created before. The relational schema creates the required tables and views. Each table created will also have appropriate primary keys, foreign keys, and integrity constraints, the proper data from the dataset CSVs will also be loaded into the tables. Indexes will also be created as necessary for the query operations in which we will use in the client application and data mining exercise.

For most entities that we have, we are using `MovieID` as a foreign key. This helps us keep a clean working schema that minimizes duplicates and if necessary, join our entities together for any work to be done through there. That being said, the foreign keys that we choose to use otherwise are pretty self-explanatory. For weak entities, the primary key would just be a variation of the primary key of the entity it depends on.

For the entities `Crew` and `Cast`, we chose to have 2 foreign keys: `MovieID` and `personID`. `MovieID` would be used for relations with the main/center entity. `PersonID` is used to relate the `People` entity.

```
CREATE TABLE crew (  
    personID VARCHAR(10) PRIMARY KEY,  
    crewRoleName VARCHAR(64),  
    FOREIGN KEY (personID) REFERENCES people(personID),  
    FOREIGN KEY (movieID) REFERENCES movie(movieID)  
);  
  
CREATE TABLE cast (  
    personID VARCHAR(10) PRIMARY KEY,  
    castRoleName VARCHAR(64),  
    FOREIGN KEY (personID) REFERENCES people(personID),  
    FOREIGN KEY (movieID) REFERENCES movie(movieID)  
);
```

One thing to note in our SQL schema is the use of dummies. Here below is a code snippet of loading the `mojo_budget_update.csv` into the `Movies` table. We see that there is the utilization of dummies. This is due to the fact that the `mojo_budget_update.csv` file has more columns/information that we intend to use. In

which case we just replace them with a dummy. This can be seen throughout a few more loads in our schema.

```
-- MOVIES
LOAD DATA INFILE 'CSV/archive/Mojo_budget_update.csv'
  INTO TABLE movie
  FIELDS TERMINATED BY ','
  ENCLOSED BY '"'
  LINES TERMINATED BY '\r\n'
  IGNORE 1 LINES
  (
    Movie_id,
    Title,
    @dummy,
    Trivia,
    Mpaa,
    Release_date,
    Run_time,
    @dummy, @dummy, @dummy, @dummy, @dummy, @dummy, @dummy, @dummy, @dummy,
    Budget
  );
```

One problem that we encountered in our schema is missing data. Some csv files are missing information. This however is fixed with the amount of files that we were provided with. Movies have more files/data to work with, so the gaps of information that are missing in our entities can be filled through multiple sources/CSVs.

# Test Plan:

It is important to create a test plan that helps verify that the server code works as expected and that the data is loaded correctly into the tables.

1. Database Creation:
  - a. Execute the SQL code on the server to create a new database
  - b. Confirm that the database is created successfully
2. Table Creation:
  - a. Execute the SQL code to create tables (movies, people, crew, cast, links, country, movieLanguage, boxOfficeGross, genre, keywords).
  - b. Check that each table is created without errors
  - c. Verify the table structures (columns, data types, primary keys, foreign keys)
3. Loading Data:
  - a. For each LOAD DATA INFILE statement, execute the code to load data into the respective table
  - b. Verify that the data is loaded without error
  - c. Create manual queries to check a few records in each table to ensure that the right data is shown and ensure that its accurate and matches the source files
    - i. Check the length and make sure it matches the CSV lengths
4. Data Integrity:
  - a. Check for any constraints or relationships between tables
  - b. Confirm that primary keys are unique
  - c. Verify that foreign keys reference existing records in related tables
5. Query:
  - a. Run a few simple SELECT queries on each tables to retrieve data
  - b. Verify that the queried data shows the correct data
  - c. Create more complex queries (joins, filtering, etc. )



## 6. Testing With Client:

- a. Error Checking
  - i. If there is any error, return to the main menu instead of exiting the program. Use of try-catch blocks here, and make the interface easy to understand for the user.
- b. Search for movies
  - i. Filter based on the relevant columns. User should be able to choose 1 or more filters, and the client should return the matching movies
- c. Add movie
  - i. Users should only be able to add if they have all the required information. If they don't, return back to the menu
- d. Create reviews
  - i. Create simple review for movie that exists
- e. Create a new review for movie that does not exist
- f. Create a rating for the movie between 1 to 10
- g. Modify existing reviews
  - i. Client will show users old review, and prompt them to create a new review in place of the old review
  - ii. Client is not able to modify a review that does not exist
- h. Delete review
  - i. Remove a review that exists
    - 1. Entry in database will be wiped
  - ii. Client should not be able to remove a review that does not exist

## 7. Cleanups:

- a. Clean up the code, remove an unimportant statements or tables

# Client Application:

Our client application is written in Python, and consists of two files, `client.py` and `constants.py`. For the client application to properly work, the data must first be loaded properly into the schema database.

The 2 key requirements that the client application should be able to do are:

1. Querying the data in a way that a customer in the domain would do
2. Modifying the data in a way that a customer in the domain would do

Our client application was created with simple customer requests in mind, since it is not possible to consider every possible client request that a customer may have in mind. It does however cover the most basic and important functions, such as:

1. Searching a movie by title, genre, people, year released, budget, keyword, duration and country
2. Add/Modify/View/Delete Client-Reviews

A README file is included in the repository that serves as a manual that helps users navigate the application.

In our `constants.py` code, we declare all the functions that would be usable by the users, including:

1. Help

Provides existing functionality of the application, and how to use said function. It outputs a list of available functions.

2. `search_m`

Search for a specific movie based on filters and user input. It will output the movie that users search for.

3. `add_m`

Add a movie to the database given the user has all the necessary information to input.

4. `search_p`

Search for a specific person based on filters and user input. It will output the person the users search for.

#### 5. create\_r

Create a movie review or other information. A new movie review will be added to the database after users call this function.

#### 6. modify\_r

Use this function to modify existing movies reviews and any information that users wish to be changed.

#### 7. view\_r

View the reviews for a movie with a given movieID.

#### 8. delete\_r

Used to delete existing movie reviews from the database if users wish to do so.

#### 9. exit

Will exit out of the client application.

Our client.py will hold all the logic and function definitions that users will use in the application.

One major function that we didn't feel the need to add is deleting and modifying existing movies. As users of our application, they shouldn't be able to change and modify existing movies that are already in the database. This kind of information should be restricted only to the admins of the application. All these other functions are like said above, kept simple for the use of users, and only having simple queries in mind. Search\_m and search\_p are both necessary functions to include as users must be able to query and search for information they need. Modify\_r, view\_r and delete\_r could be argued as unnecessary, but we thought it was information that a user would want to see (view\_r). And if they are able to view information made by other users, they should be able to modify (modify\_r) and delete (delete\_r) their reviews if they chose to do so.

# Data Mining Investigation:

For our investigation, we will be looking at how different factors contribute to a movie's overall gross. For this, we will consider the gross to be the `worldwide_gross`, which is in `boxofficemojo_releases.csv`. We first examine each of the factors and how they relate to gross. We then either manually posit a correlation, or use a mathematical correlation function to determine its effectiveness.

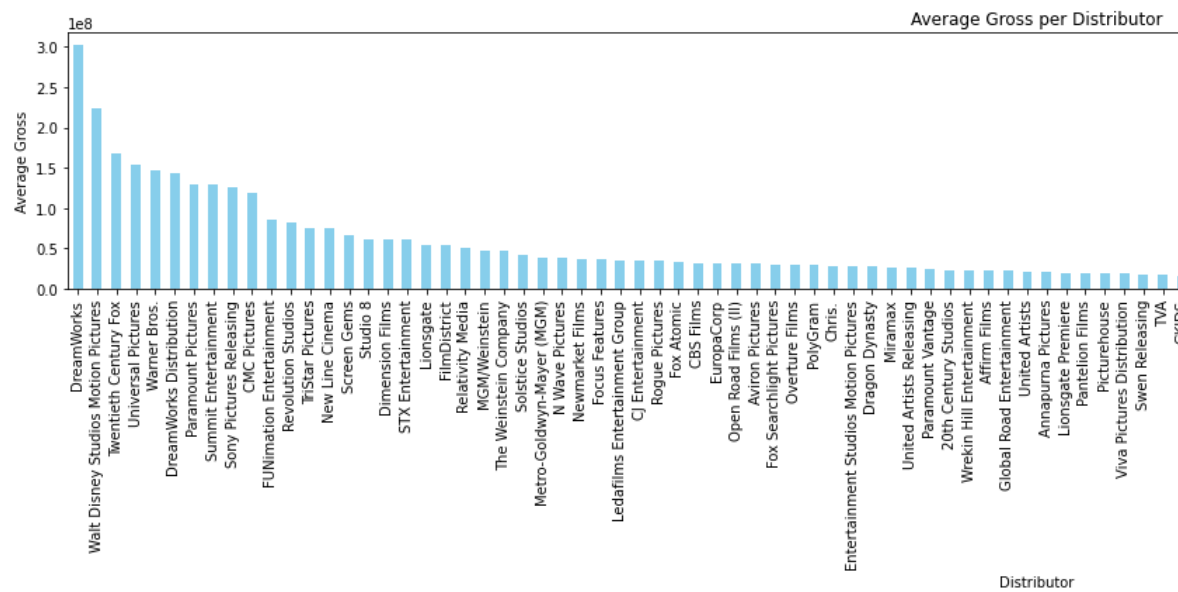
Our overall approach was to use Pandas. This is because it has a lot of functionality, both in pre-processing, modelling, post-processing, and validation. As well, matplotlib was used for the graphs.

The next part of this report will detail each of the factors and their process, validation, and result.

```
"distributor_name", "budget", "release_date", "worldwide_gross", "genres",  
"mpaa_rating", "running_time"
```

1. `Distributor_name`
  - a. When examining the data manually, we had the question - do certain movie studios tend to produce large movies? Intuitively, the answer was yes, but we had to find out.
  - b. First, we removed any n/a data. This is because it would be hard to interpolate, use moving averages, or polynomial functions for this type of categorical data.
  - c. Then, we used a Pandas groupby function to aggregate the distributor names with the average gross of each movie produced.

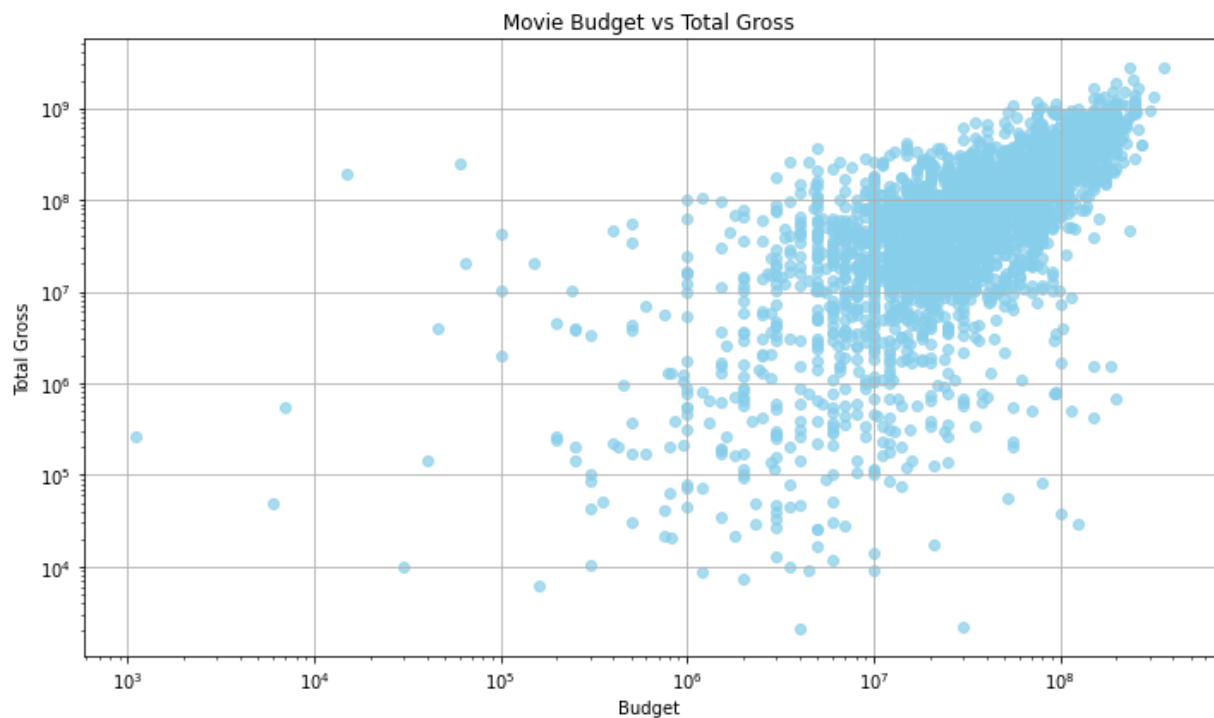
d. The final graph is here:



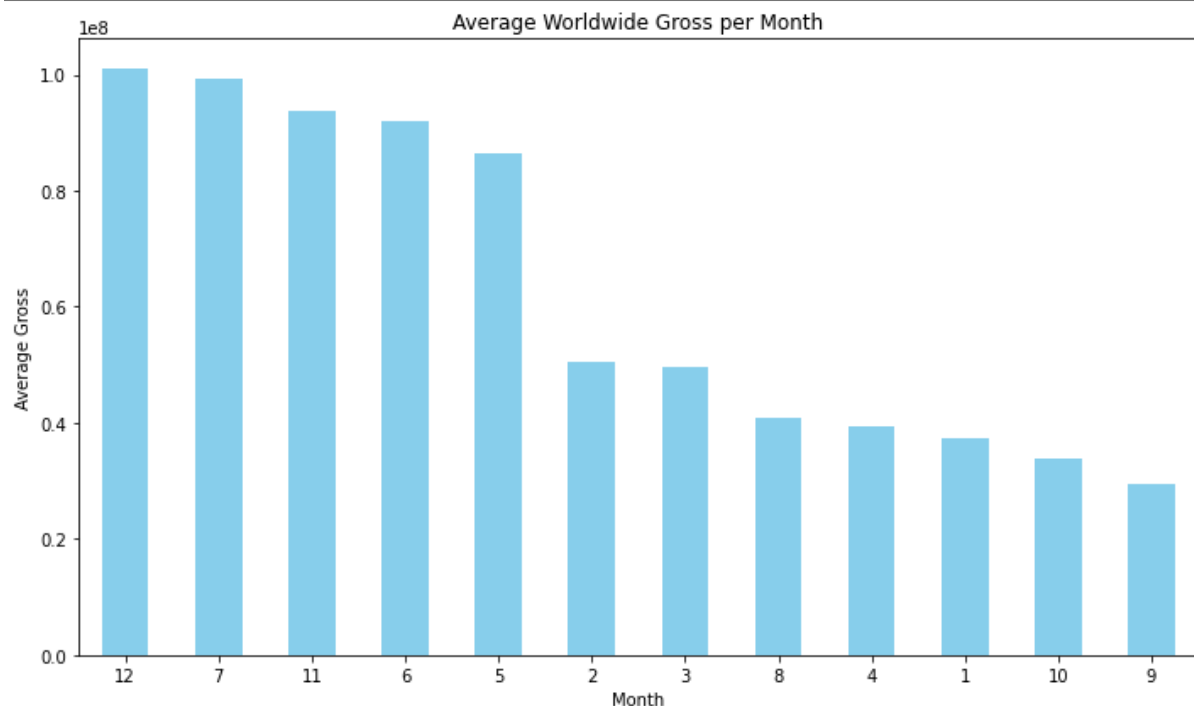
- The right half of the graph is cut off because we deemed these distributors irrelevant to the rest.
- We can see clearly that the first 10 or so distributors have significantly higher gross than the rest. There is a strong correlation, but this does not necessarily mean causation.

2. Our second variable looked at is budget.

- Many entries had 0.0 for their budget, so we removed these as well. It would not be useful to interpolate these either.
- Upon graphing, we get this graph:

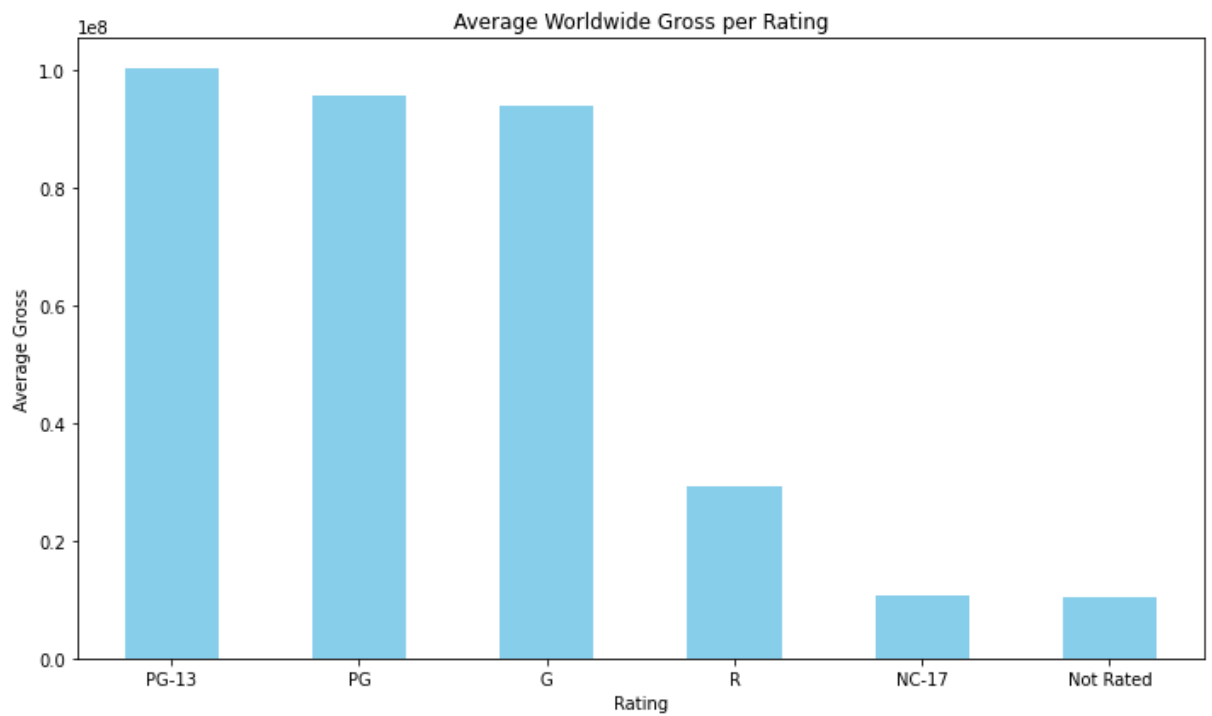


- c. Because of the numerical nature of both axes, we were able to calculate the Pearson coefficient, which in this case, is 0.713, a fairly strong correlation.
3. The next factor we analyzed was release date.
- a. This required some preprocessing. We had to turn the release dates into months, to see if any specific month had a higher average grossing compared to the others.
  - b. Some advanced pandas datetime module usage, as well as another groupby statement gives us this graph:



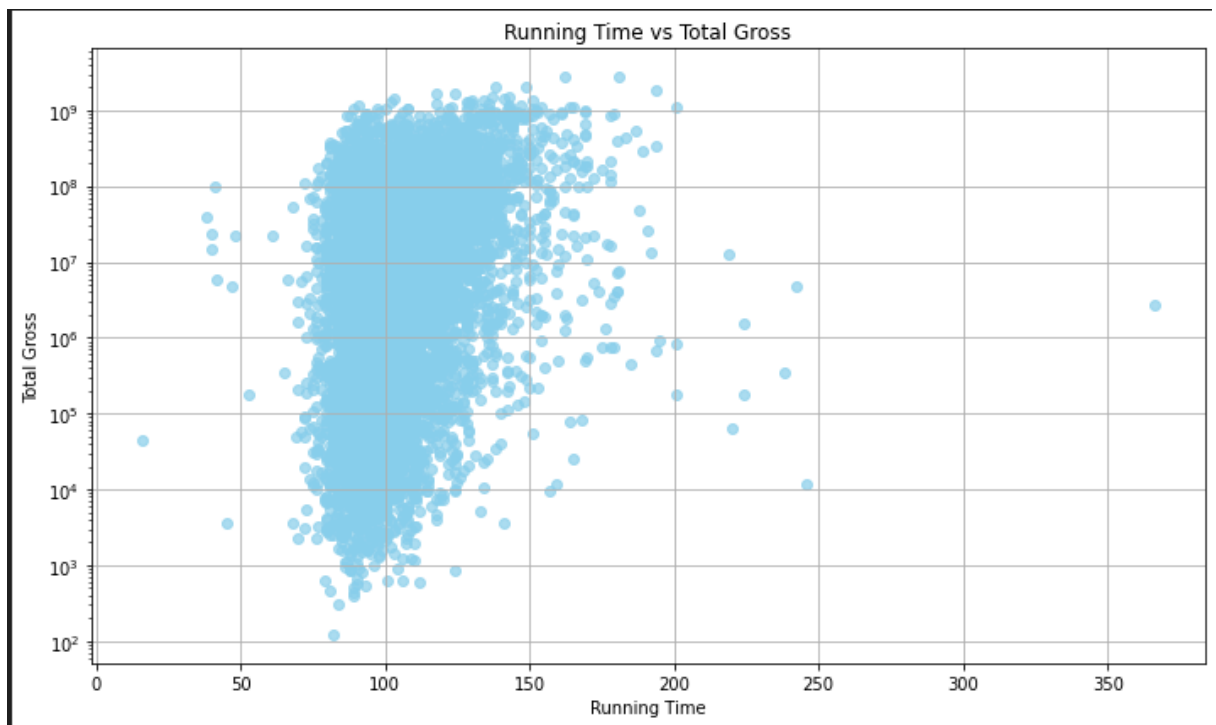
- c. As we can see, the 12th month (December), July, November, June and May have the highest grossing. This is significantly higher than the other months. We might extrapolate this due to the summer months, Christmas holidays, and other seasonal factors.
4. MPAA Rating is the next factor to determine. For this, again, we had to clean up our missing/improper data. After this, we were able to graph it vs

worldwide\_grossing average. Here is the result:

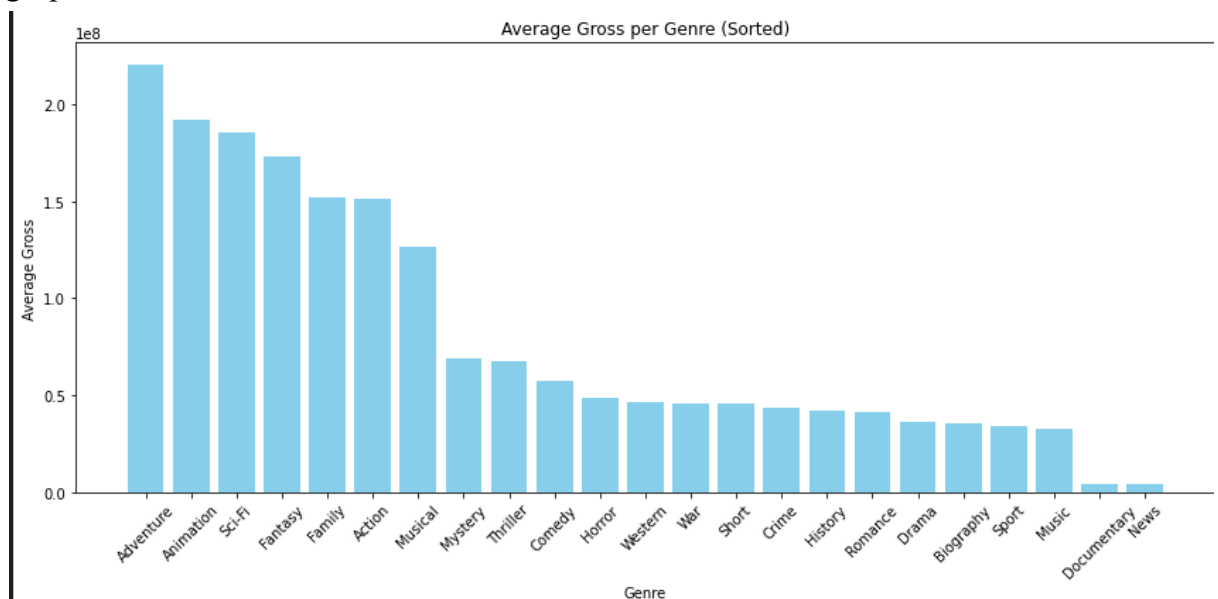


- a. It is worth noting that R rated movies have significantly lower average gross. This makes sense, as many theatres actually adhere to age restrictions, meaning that anyone under 18 cannot buy a R-rated movie ticket.
5. The next factor is running time. We followed a similar process in data pre-processing. We had to turn the running time format (X hr Y min) into total

minutes using a pandas apply function. The graph is here:



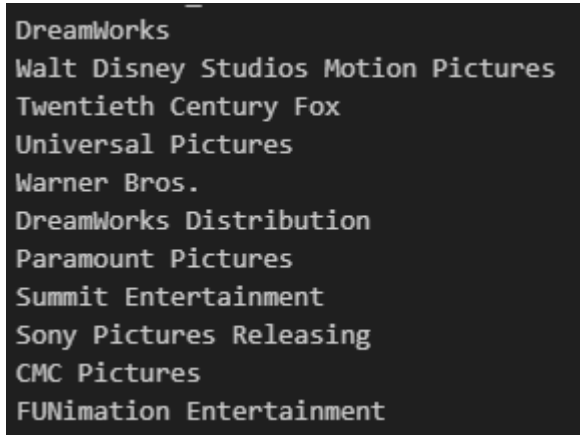
- a. We were able to calculate a Pearson coefficient of 0.28, showing a weak positive correlation. The graph is relatively scattered, however, as the running time increases, the frequency of higher gross also increases.
6. Lastly, we analyzed the genres of movies. This was slightly more difficult as movies could have multiple genres. We used dictionary logic to iterate over the dataframe and keep track of the total, count, and average of the gross. The graph is as follows:





- a. This shows that adventure, animation, sci-fi, fantasy, family, action, and musicals tend to have higher average grossing. There is a drop-off after this

Conclusion:

- After our analysis of 6 different factors, we saw correlations of different strengths all across the board. In brief, these were:
    - Released by one of the top 10 distributors
- 

```
DreamWorks
Walt Disney Studios Motion Pictures
Twentieth Century Fox
Universal Pictures
Warner Bros.
DreamWorks Distribution
Paramount Pictures
Summit Entertainment
Sony Pictures Releasing
CMC Pictures
FUNimation Entertainment
```
- A higher budget (average is 47,900,985)
  - Released in May, June, July, November or December
  - Doesn't have a rating of R
  - Relatively higher run-time (average is 104.9 minutes)
  - Genre of adventure, animation, sci-fi, fantasy, family, action, or musicals
- To validate this, let us check the top 10 movies with the highest grossing.

The following table checks these conditions:

Title	Top 10 Distributor?	Higher than average budget?	Release in May, June, July, November, December?	Doesn't have a rating of R?	Higher than average run-time?	Adventure, animation, sci-fi, fantasy, family, action, musical?
Avengers: Endgame	✓	✓		✓	✓	3/4
Avatar	✓	✓	✓	✓	✓	4/4
Star Wars - The Force Awakens	✓	✓	✓	✓	✓	3/3
Avengers: Infinity War	✓	✓		✓	✓	3/3
Titanic	✓	✓	✓	✓	✓	0/2
Jurassic World	✓	✓	✓	✓	✓	3/3
The Lion King	✓	✓	✓	✓	✓	4/5
The Avengers	✓	✓	✓	✓	✓	3/3
Furious 7	✓	✓		✓	✓	2/3
Frozen II	✓	✓	✓	✓		5/6

In conclusion, we can see that the grossing of movies definitely is impacted by these 6 factors.