Movie Rating Prediction

Team 15: 哦伊

Team Members: 杜霙笙 H2408032 陳詠翰 H24081163 戰孅 D54099012

1. Brief Introduction of the Problem

Nowadays, with the rise of online streaming platforms like NETFLIX, Disney Plus and many more, predicting movie ratings has become increasingly important because it can serve as an important asset in not only marketing (content-based filtering), but also improving the user experience of streaming platforms (user-based filtering). With all that said, our goal is to find the indicators that affect movie ratings and use them to train our model in order to predict the ratings of movies based on their information alone. There are so many movies that are directed by different directors, different cast, and other attributes. As a result, we want to predict movies rating based on movies content through machine learning algorithms. After that, since it will be hard to create an interface to predict future movies without their info, we have decided to implement a simple user database to use our prediction as a base to predict the rating that the current user will give based on the ratings they have already given.

2. Data Description and Preprocessing

Data Description

For our dataset, we use three csv files including movies.csv, rating.csv and links.csv from the infamous MovieLen Dataset. The first csv file is the TMDB data set which includes the movie information as follows:

1. movies.csv:

Id: The id of the movie for reference.

Title: The entire title of the movie.

Genres: Represents the movie’s genre categories. Each movie has two or three genres and there are twenty different genres in whole dataset.

Date: The release date of the movie. YYYY - MM - DD

Collection: If the movie has sequels, it will be shown in this column, whereas the others will show NA.

Runtime: The length of the movie (in minutes).

Revenue and Budget: The money profited and spent on the production of the movie. They are mostly 0.

Cast: The actors that star in the movie. There are over ten thousand different actors in entire dataset.

Production companies: The companies that participated in the production of the movie.

Production countries: The countries involved in the production of the movie.

Language: The verbal language used throughout the movie.

Popularity: The popularity of the movie calculated into a metric.

Average vote: A number between 0 – 10 which is the average rating of the movie. (We have renamed this column to Rating to avoid confusion)

Number of votes: The number of ratings this movie has.

IMDB id: The movie id from the IMDB dataset.

2. rating.csv:

includes user id, a random movie id and rating, which shows the ratings different users give to each movie. The rating ranges from 0.5 to 5.

3. links.csv:

Includes each movie's random id used in rating.csv, IMDB id and TMDB id.

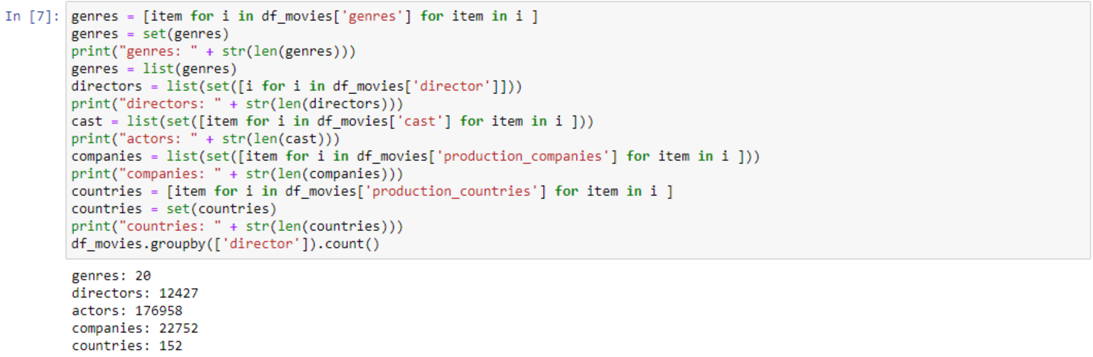
Preprocessing

Merging Data: First, we calculate the average rating of each movie in the rating.csv file then map the data to links.csv to find each movie's IMDB id, then merge it with the data derived from movies.csv based on each movie's IMDB id, renamed the average rating column "other_rating" (IMDB) to not get confused with our target variable, "Rating". After merging the three csv files to one data frame, we now have one data frame to perform the next part of our preprocessing.

Type conversion: The genres, cast, production countries, and production companies are of type string, so we split each of these by comma and stripped the whitespaces from the front ant the back, saving them as lists.

Data refactoring: For convenience, we refactored collection and language into binary variables. For collection, 1 means that the movie a sequel and 0 means it is a solo film. For language, we observed that more than half of the movies were in English, and the second most as well as third most took up lower than 15 percent, therefor we chose to use 1 to represent English, and 0 to represent non-English movies. As for the date, we believed that the year of the movie would be sufficient representation of the date of the movie and therefore replaces the date column with only the year.

Encoding: Our next move was to determine, how the genres, directors, actors, cast, production companies and production countries could be represented as features for our model. To determine this, first, we needed to know how many different directors, actors, genres, production companies and production countries values were in the data frame. The result was (genres: 20, directors: 12.4k, actors: 177k, companies: 22.7k, countries: 152)



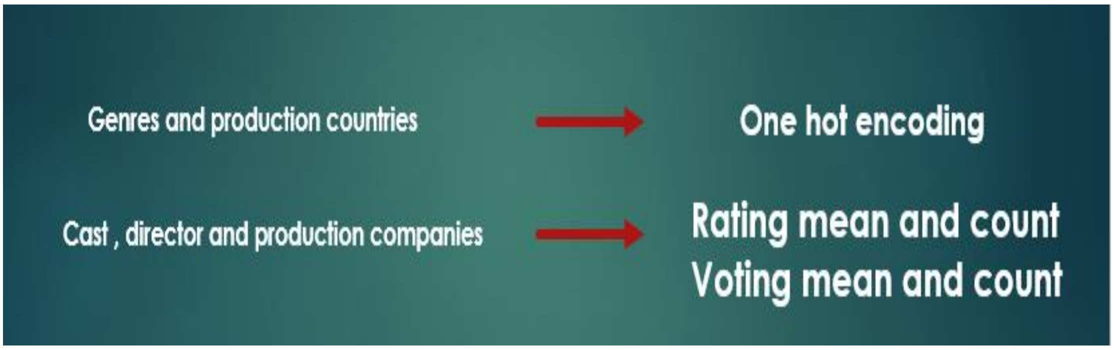
```
In [7]: genres = [item for i in df_movies['genres'] for item in i ]
        genres = set(genres)
        print("genres: " + str(len(genres)))
        genres = list(genres)
        directors = list(set([i for i in df_movies['director']]))
        print("directors: " + str(len(directors)))
        cast = list(set([item for i in df_movies['cast'] for item in i ]))
        print("actors: " + str(len(cast)))
        companies = list(set([item for i in df_movies['production_companies'] for item in i ]))
        print("companies: " + str(len(companies)))
        countries = [item for i in df_movies['production_countries'] for item in i ]
        countries = set(countries)
        print("countries: " + str(len(countries)))
        df_movies.groupby(['director']).count()

        genres: 20
        directors: 12427
        actors: 176958
        companies: 22752
        countries: 152
```

Number of genres, directors, actors, companies, and countries

For genres, since there were 20 in total, we decided to use **One Hot Encoding** to transfer each genre into a column and binarize the values, removing the genre column and adding 20 columns, each a different genre, and using 1s and 0s to show whether this movie falls into the category of the genre or not. For production countries, we wanted to do the same thing, however, we thought that 152 extra columns were too many, so we did some filtering, keeping only the countries that have participated in more than 100 movies. We were left with 36 countries and processed and binarized them in similar fashion with the genres.

As for the directors, cast, and production companies, We found out that there were too many of each to perform one hot encoding for cast and companies.

Therefore, after digging into a few online resources, we decided to use another way to encode the directors, cast, and production companies, target encoding. **Target Encoding** is the method of saving each value of the variable you wish to encode in the form of the mean of the target value. Therefore, we used the values rating_mean, rating_count, vote_mean, and vote_count to represent each director, actor, and company as possible features for our model. These values were created by calculating the average rating (IMDB) and votes of all the movies each director, actor, or company has participated in, which are the means; the counts are the number of movies each director, actor, and company has participated in and the total number of votes from all the movies they have participated in. Our argument for these variables is that the average ratings and votes reflect their performance and the higher the count, the more famous or experienced they are. The target encoded variables are saved in the form of 3 dictionaries, Director_Dict, Cast_Dict, and Companies_Dict. These dictionaries are created by functions defined in our code and have each director, actor, and company name as their keys and lists containing rating_mean, rating_count, vote_mean, and vote_count as values for reference. Creating these dictionaries is extremely time consuming due to the large amount of data and calculation required. ***(Due to this, the total time needed to train and run the entire model will take approximately 1 hour.)***



Genres and production countries → One hot encoding

Cast , director and production companies → Rating mean and count
Voting mean and count

Feature Selection for target encoding:

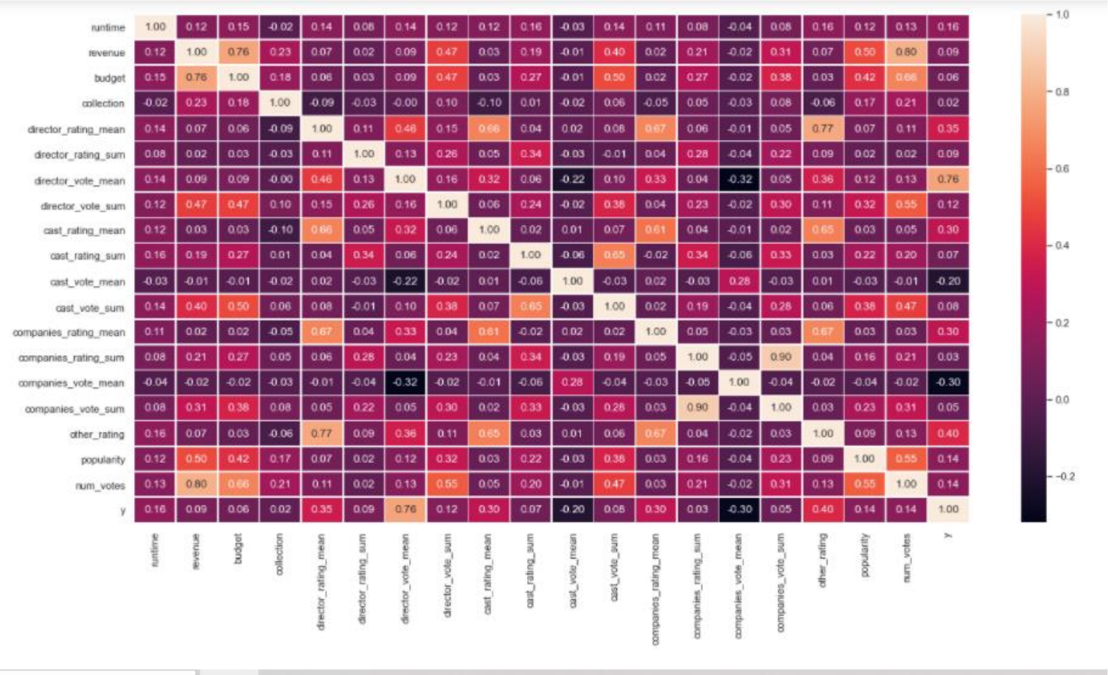
After encoding, we defined more functions to help our feature selection extract features. Since each movie has only one director, the Director_Dict can be directly used to find the features of that movie's director. However, there are more than one actor and company for each movie. So, we chose to find the average of the data that correspond to each actor and company in the Cast_Dict and Companies_Dict. Since we realized that the list of actors in the cast column is based the order of the importance of their role in the film, we decided to take only the average of the top 3 actors.

During feature extraction, if the director, cast or company cannot be found in our dictionary (due to train test splitting and possible when predicting new movies), our feature extraction function will fill in the values of director with the average of all directors and 0 for counts, whereas the cast and companies will be filled in with the mean of the values that can be found, and will only fill in the average when none of the data of the actors and companies can be found.

Conclusion:

The above concludes our work of preprocessing the data for feature extraction. It was challenging to determine how to process each piece of data in order to find the optimal way to extract useful information. In the end, during feature extraction, we left out those such as popularity, revenue, and votes-related features because the popularity and revenue can be hard to calculate for new movies and the votes-related features, if added, could result in multicollinearity, which would lower the performance of our model, which we will talk about in the next section of our report.

3. Insights Discovered from Data



	runtime	revenue	budget	collection	director_rating_mean	director_rating_sum	director_vote_mean	director_vote_sum	cast_rating_mean	cast_rating_sum	cast_vote_mean	cast_vote_sum	companies_rating_mean	companies_rating_sum	companies_vote_mean	companies_vote_sum	other_rating	popularity	num_votes	y
runtime	1.00	0.12	0.15	-0.02	0.14	0.08	0.14	0.12	0.12	0.16	-0.03	0.14	0.11	0.08	-0.04	0.08	0.16	0.12	0.13	0.16
revenue	0.12	1.00	0.76	0.23	0.07	0.02	0.09	0.47	0.03	0.19	-0.01	0.40	0.02	0.21	-0.02	0.31	0.07	0.50	0.80	0.09
budget	0.15	0.76	1.00	0.18	0.06	0.03	0.09	0.47	0.03	0.27	-0.01	0.50	0.02	0.27	-0.02	0.38	0.03	0.42	0.66	0.06
collection	-0.02	0.23	0.18	1.00	-0.09	-0.03	-0.00	0.10	-0.10	0.01	-0.02	0.06	-0.05	0.05	-0.03	0.08	-0.06	0.17	0.21	0.02
director_rating_mean	0.14	0.07	0.06	-0.09	1.00	0.11	0.46	0.15	0.66	0.04	0.02	0.08	0.67	0.06	-0.01	0.05	0.77	0.07	0.11	0.35
director_rating_sum	0.08	0.02	0.03	-0.03	0.11	1.00	0.13	0.26	0.05	0.34	-0.03	-0.01	0.04	0.28	-0.04	0.22	0.09	0.02	0.02	0.09
director_vote_mean	0.14	0.09	0.09	-0.00	0.46	0.13	1.00	0.16	0.32	0.06	-0.22	0.10	0.33	0.04	-0.32	0.05	0.36	0.12	0.13	0.76
director_vote_sum	0.12	0.47	0.47	0.10	0.15	0.26	0.16	1.00	0.06	0.24	-0.02	0.38	0.04	0.23	-0.02	0.30	0.11	0.32	0.55	0.12
cast_rating_mean	0.12	0.03	0.03	-0.10	0.66	0.05	0.32	0.06	1.00	0.02	0.01	0.07	0.61	0.04	-0.01	0.02	0.65	0.03	0.05	0.30
cast_rating_sum	0.16	0.19	0.27	0.01	0.04	0.34	0.06	0.24	0.02	1.00	-0.06	0.65	-0.02	0.34	-0.06	0.33	0.03	0.22	0.20	0.07
cast_vote_mean	-0.03	-0.01	-0.01	-0.02	0.02	-0.03	-0.22	-0.02	0.01	-0.06	1.00	-0.03	0.02	-0.03	0.28	-0.03	0.01	-0.03	-0.01	-0.20
cast_vote_sum	0.14	0.40	0.50	0.06	0.08	-0.01	0.10	0.38	0.07	0.65	-0.03	1.00	0.02	0.19	-0.04	0.28	0.06	0.38	0.47	0.08
companies_rating_mean	0.11	0.02	0.02	-0.05	0.67	0.04	0.33	0.04	0.61	-0.02	0.02	0.02	1.00	0.05	-0.03	0.03	0.67	0.03	0.03	0.30
companies_rating_sum	0.08	0.21	0.27	0.05	0.06	0.28	0.04	0.23	0.04	0.34	-0.03	0.19	0.05	1.00	-0.05	0.90	0.04	0.16	0.21	0.03
companies_vote_mean	-0.04	-0.02	-0.02	-0.03	-0.01	-0.04	-0.32	-0.02	-0.01	-0.06	0.28	-0.04	-0.03	-0.05	1.00	-0.04	-0.02	-0.04	-0.02	-0.30
companies_vote_sum	0.08	0.31	0.38	0.08	0.05	0.22	0.05	0.30	0.02	0.33	-0.03	0.28	0.03	0.90	-0.04	1.00	0.03	0.23	0.31	0.05
other_rating	0.16	0.07	0.03	-0.06	0.77	0.09	0.36	0.11	0.65	0.03	0.01	0.06	0.67	0.04	-0.02	0.03	1.00	0.09	0.13	0.40
popularity	0.12	0.50	0.42	0.17	0.07	0.02	0.12	0.32	0.03	0.22	-0.03	0.38	0.03	0.16	-0.04	0.23	0.09	1.00	0.55	0.14
num_votes	0.13	0.80	0.66	0.21	0.11	0.02	0.13	0.55	0.05	0.20	-0.01	0.47	0.03	0.21	-0.02	0.31	0.13	0.55	1.00	0.14
y	0.16	0.09	0.06	0.02	0.35	0.09	0.76	0.12	0.30	0.07	-0.20	0.08	0.30	0.03	-0.30	0.05	0.40	0.14	0.14	1.00

Through the heatmap, we observed that some of the variables are highly correlated with each other, such as revenue, budget, director votes sum, cast vote sum, company vote sum, popularity and num votes. Revenue has high correlation with budget, and revenue also has high correlation with director votes sum. Due to the multicollinearity, if we put those seven variables in the regression model at the same time, the performance metrics of the model such as MSE or RMSE will be significantly higher, because we will have taken too many similar variables in the model. As a result, when we start to drop variables in the model, those variables will be taken in consideration first.

In addition, cast rating mean, director rating mean, and companies rating mean are highly correlated with each other. It is easier to understand that phenomenon because these variables are evaluated by IMDB ratings and popular actors work with popular director, which makes sense.

With these insights, we will have a clearer direction as to what our features should be and how to choose which features to keep and which features to drop.

4. Methodology details

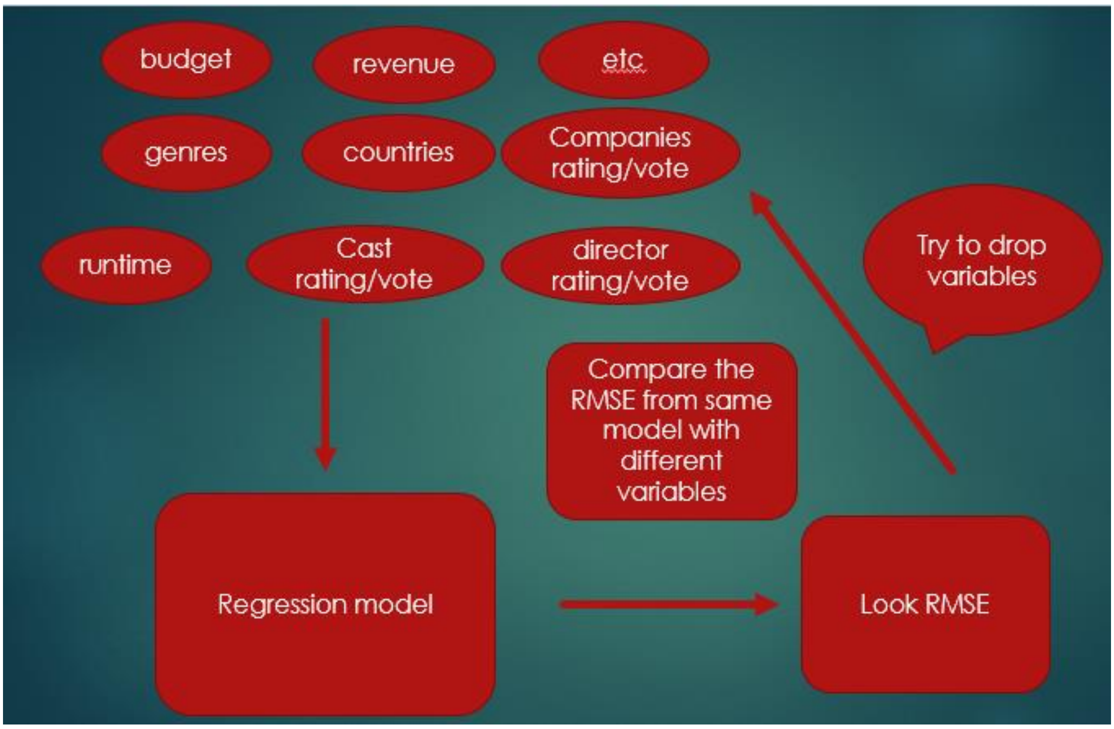
Train Model

To find the optimal model, we took the following steps. For easy implementation, modification, and reusability, we have defined most of the following actions in the form of functions.

1. Train-test split: we need to split the data to train data and test data. Use train data to train and use test data to evaluate the model.

   Why train- test split? To avoid overfitting and it serves as a better way to simulate the situation in real life since the train-test split procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model.
2. Create Dictionaries of Directors, Cast, and Companies based on the train data to calculate the rating mean and vote mean correspondent to each director, actor, and production company for feature extraction.
3. Choose the features and machine learning algorithm you want. First, we put all the features in the model, and we will explain why later.
4. Put the data (selected features) into the model. Use the test data to evaluate the model. Because it is a regression model, we have decided to use RMSE to evaluate the model.
5. We start to drop variables to see whether the performance of regression model is better. This is similar to the backward elimination process in regression analysis.

   Why we use backward elimination method? Because we do not know which variable is the most important in the model, so we start with putting all variables in the model and then dropping the most dependent variable first.
6. Return to 3, trying different combinations of selected features and different models. (As shown in the figure below)



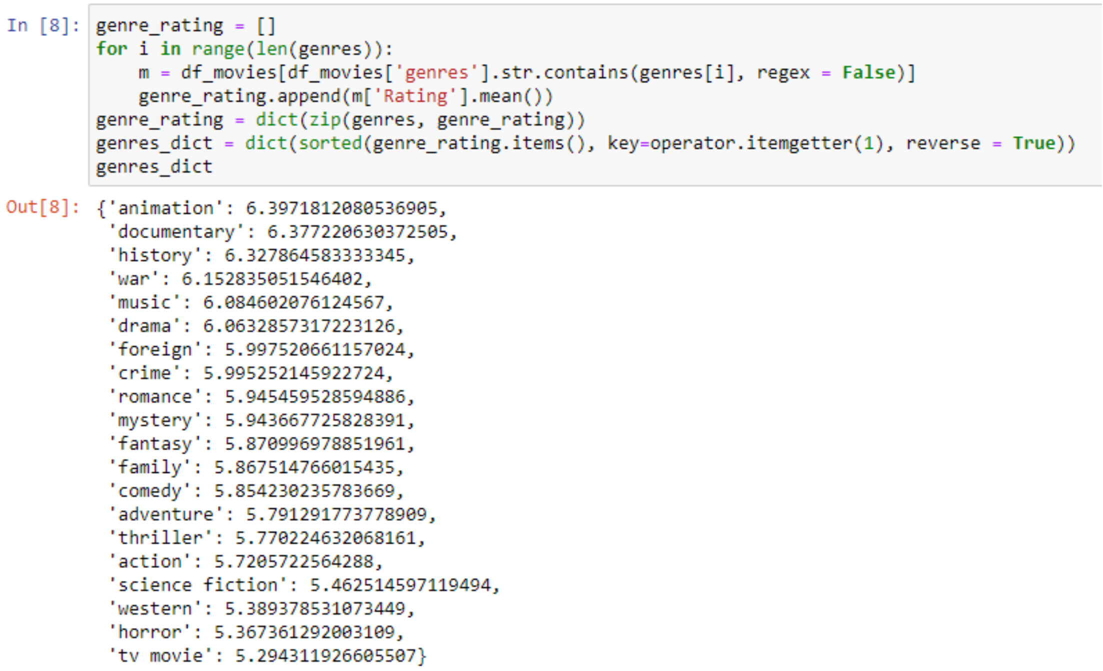
Model training process.

Interactive Interface

After the model is trained, we have decided to implement an interactive interface with a simple user database for users to see our results.

The Database: Implemented to keep track of each user's ratings, which are used to update the director and cast dictionaries according to each user. The reason we modify the director and cast dictionaries is because we believe that the director and the cast have a larger impact on a person as to whether they like a movie or not. With every rating a user gives to a movie, the database uses the rating to update the values of the director and cast dictionary. The higher the rating, the more the value of the director and cast is raised, affecting the predicted rating of movies with the same director or actors.

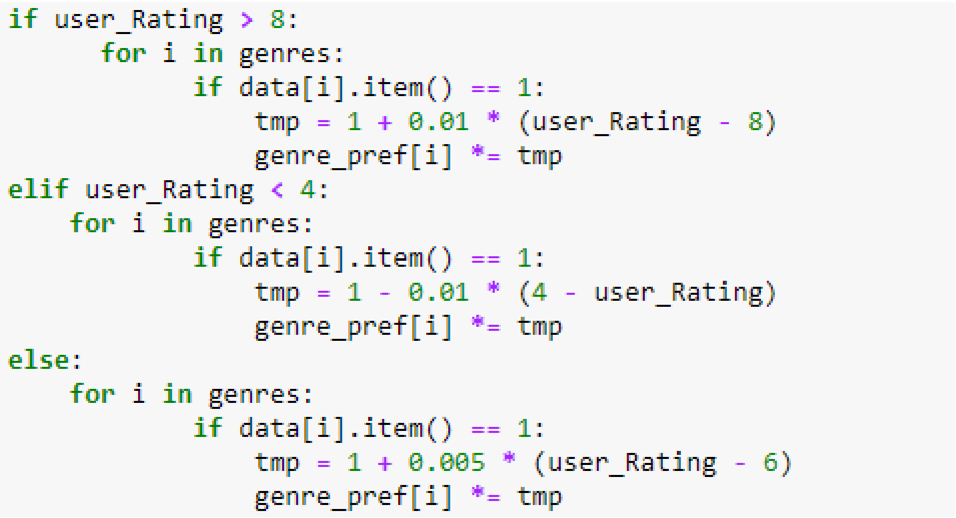
Genre Preference: Since we also believe that genre preference also affects a user's rating, we have added a weighting system for the genres initialized at 1 for all 20 genres. And for every rating the user gives, the weight is scaled. The way we scale the weight is based on the figure below (the average rating for movies of each genre).



```
In [8]: genre_rating = []
        for i in range(len(genres)):
            m = df_movies[df_movies['genres'].str.contains(genres[i], regex = False)]
            genre_rating.append(m['Rating'].mean())
        genre_rating = dict(zip(genres, genre_rating))
        genres_dict = dict(sorted(genre_rating.items(), key=operator.itemgetter(1), reverse = True))
        genres_dict
```

```
Out[8]: {'animation': 6.3971812080536905,
         'documentary': 6.377220630372505,
         'history': 6.327864583333345,
         'war': 6.152835051546402,
         'music': 6.084602076124567,
         'drama': 6.0632857317223126,
         'foreign': 5.997520661157024,
         'crime': 5.995252145922724,
         'romance': 5.945459528594886,
         'mystery': 5.943667725828391,
         'fantasy': 5.870996978851961,
         'family': 5.867514766015435,
         'comedy': 5.854230235783669,
         'adventure': 5.791291773778909,
         'thriller': 5.770224632068161,
         'action': 5.7205722564288,
         'science fiction': 5.462514597119494,
         'western': 5.389378531073449,
         'horror': 5.367361292003109,
         'tv movie': 5.294311926605507}
```

As we can see, the average rating of all genres is approximately 6. Therefore, we use 4, 6, and 8 (6 + - 2) as thresholds for whether a person likes a movie or not and scale the weights of the genres of this movie accordingly. (the code is as follows)



```
if user_Rating > 8:
   for i in genres:
      if data[i].item() == 1:
        tmp = 1 + 0.01 * (user_Rating - 8)
        genre_pref[i] *= tmp
elif user_Rating < 4:
  for i in genres:
      if data[i].item() == 1:
        tmp = 1 - 0.01 * (4 - user_Rating)
        genre_pref[i] *= tmp
else:
  for i in genres:
      if data[i].item() == 1:
        tmp = 1 + 0.005 * (user_Rating - 6)
        genre_pref[i] *= tmp
```

Predicted Rating: Using our trained model and constantly updating database, we generate an initial prediction, then multiply it by the weights of the genres of this movie.

The Interface: An interactive interface that allows the user to search for movies, see our predicted rating and give ratings. Once the user logs out, the current user's ratings will be printed out and the data is erased. The database is reinitialized every time you rerun the interface.

5. Evaluation

Our prediction is evaluated by Root Mean Square Error (RMSE).

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - y_i\text{^})^2$$

MSE is calculated by the sum of square of prediction error which is real output minus predicted output and then divide by the number of data points. It gives you an absolute number on how much your predicted results deviate from the actual number. You cannot interpret many insights from one single result but it gives you a real number to compare against other model results and help you select the best regression model.

Root Mean Square Error(RMSE) is the square root of MSE. It is used more commonly than MSE because firstly sometimes MSE value can be too big to compare easily. Secondly, MSE is calculated by the square of error, and thus square root brings it back to the same level of prediction error and makes it easier for interpretation.

In summary, RMSE value is calculated by the mean square of the difference between predicted rating and real rating.

Feature extraction: The only way we could know the best features for the model was though trial and error, process mentioned in part 4. The following is a table of the features we dropped, and the results yielded.

All features include: ['date', 'runtime', 'revenue', 'budget', 'collection', 'director_rating_mean', 'director_rating_count', 'director_vote_mean', 'director_vote_count', 'cast_rating_mean', 'cast_rating_count', 'cast_vote_mean', 'cast_vote_count', 'companies_rating_mean', 'companies_rating_count', 'companies_vote_mean', 'companies_vote_count', 'other_rating', 'popularity', 'num_votes'] + [20 genres + 36 production countries] (binary values)

Dropped Features	RMSE
['title', 'id', 'revenue', 'collection', 'Rating', 'popularity', ’budget’, 'director_vote_count', 'cast_vote_count', 'companies_rating_count', 'companies_vote_count']	2.5305
['title', 'id', 'revenue', 'collection',	2.2451

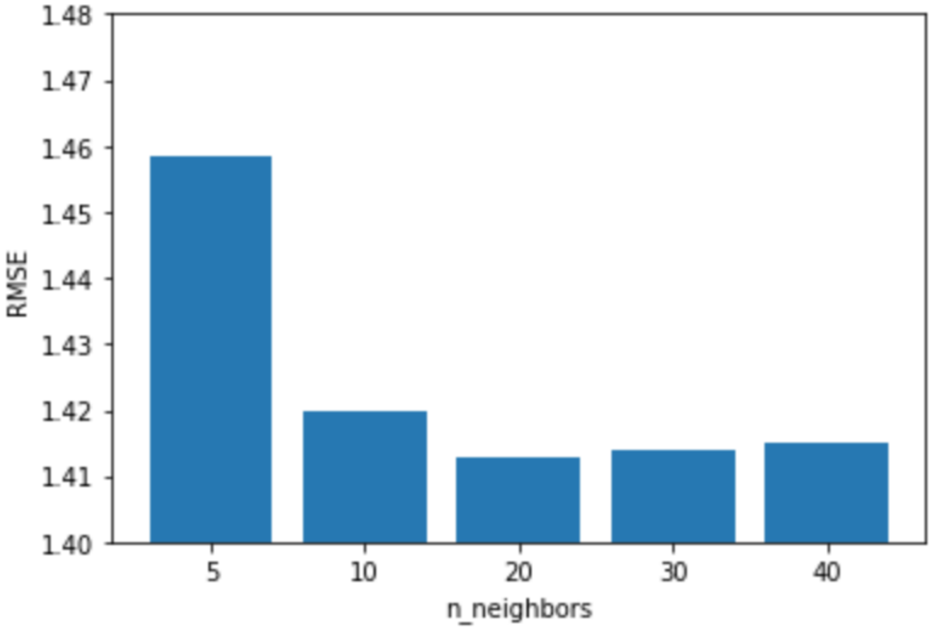
'Rating', 'popularity', ’budget’, 'director_vote_count', 'cast_vote_count', 'companies_rating_count', 'companies_vote_count', 'companies_vote_mean']	
['title', 'id', 'Rating', 'revenue', 'budget', 'collection', 'popularity', 'cast_vote_mean', 'cast_vote_count', 'companies_rating_count', 'companies_vote_mean', 'companies_vote_count']	2.2192
['title', 'id', 'Rating', 'revenue', 'budget', 'collection', 'popularity', 'cast_vote_mean', 'cast_vote_count', 'director_vote_mean', 'companies_rating_count', 'companies_vote_mean', 'companies_vote_count']	0.9002
['title', 'id', 'Rating', 'revenue', 'budget', 'cast_vote_mean', 'director_vote_mean', 'popularity', 'companies_vote_mean']	0.9105
['title', 'id', 'Rating', 'revenue', 'budget', 'collection', 'popularity', 'director_vote_mean', 'director_vote_sum', 'cast_vote_mean', 'cast_vote_count', 'companies_rating_count', 'companies_vote_mean', 'companies_vote_count']	0.8885 (the best one!)

Due to multicollinearity, once we dropped all columns related to votes, the performance of the model was raised tremendously.

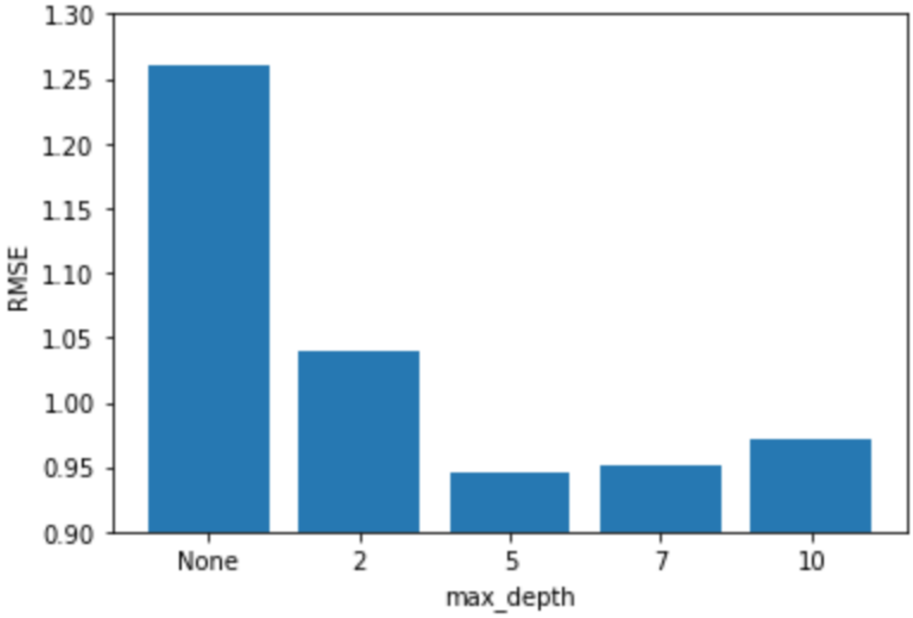
For our final model, the features we selected were:
['date', 'runtime', 'director_rating_mean', 'director_rating_count', 'cast_rating_mean', 'cast_rating_count', 'companies_rating_mean', 'other_rating', 'num_votes'] + [20 genres and 36 production countries] = 65 in total.

Different Machine Learning Algorithms:

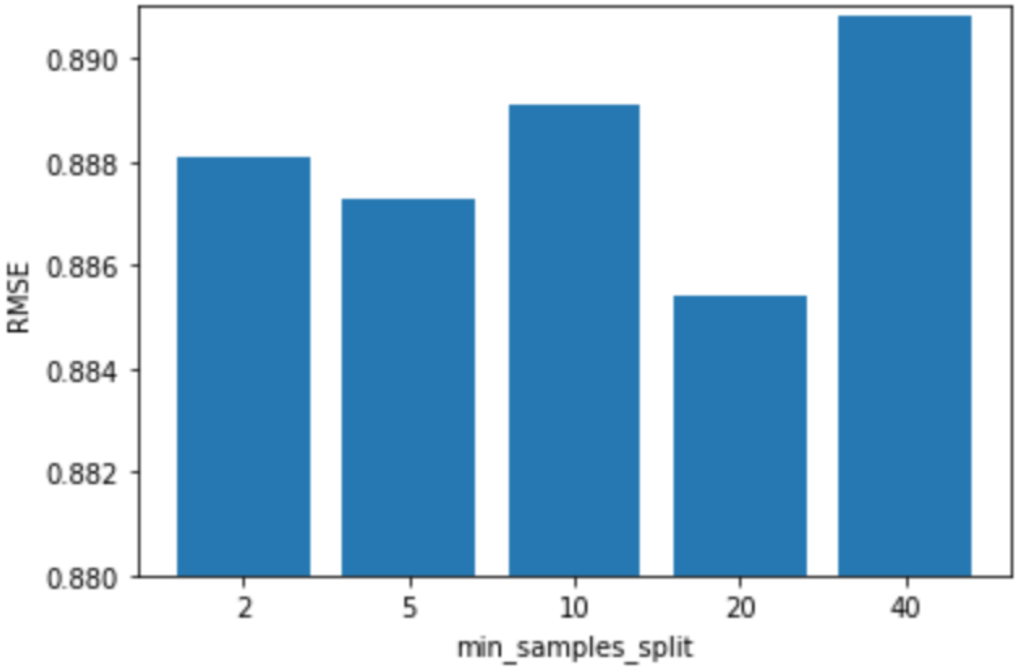
KNN



Decision Tree Regressor



Random Forest Regressor



Tuning Parameters of Algorithm:

For KNN, we tried changing the number of neighbors; for Decision Tree Regressor, we tried different max depths and for Random Forest Regressor, we tried different minimum samples split. Each algorithm had a situation where they performed at their best, respectively.

Performance:

Random Forest > Decision Tree > KNN

Therefore, we selected Random Forest Regressor with min_samples_split = 20 to train our final model.

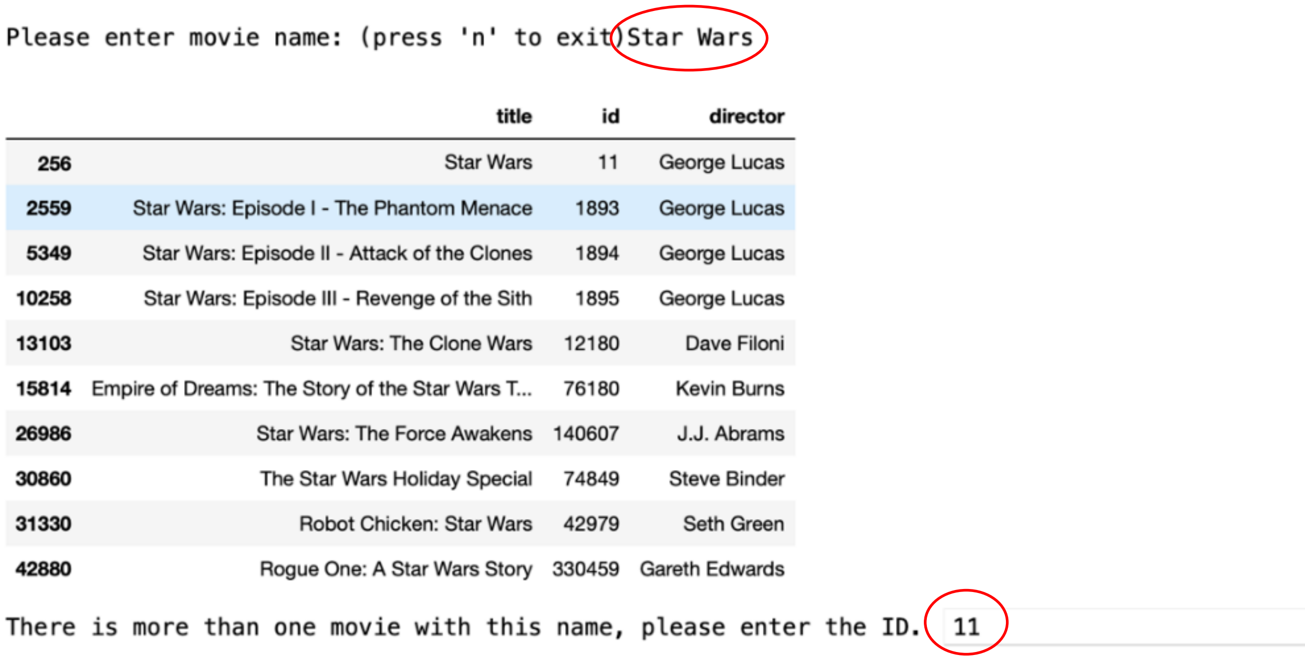
6. Results

Our result is an interactive interface defined in a function that has the preprocessed Data Frame and trained model as parameters. When the user enters a movie name, we will find detailed information of the given movie from data frame. Passing the information and previously created director, cast, and company dictionaries to functions, previously mentioned in part 4 of the report, to perform feature extraction. Next, we input the features into our trained model to predict the movie's average rating. Finally, the result is scaled based on the movie genre preference to personalize the movie rating predictions based on the user. Once the result is shown, the user will be asked to give this movie a rating for the database to update the data used for our model, as well as modify the scales of our genre preference.

Theoretically, the more ratings the user gives, the more precise our predictions will get, since the database will continue to update itself as long as the user gives the database new movie ratings. In addition, we regard ‘logging out’ and rerunning the function as switching to another user. Thus, the database will be reinitialized while you rerun the function.

Here is a quick Demo of our Result.
We will be taking the movie Star Wars as an example:

Step 1: Enter movie name “Star Wars”. Since there is more than one movie matched, we will enter the movie id “11”.

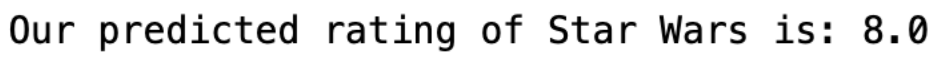


```
Please enter movie name: (press 'n' to exit)Star Wars
```

	title	id	director
256	Star Wars	11	George Lucas
2559	Star Wars: Episode I - The Phantom Menace	1893	George Lucas
5349	Star Wars: Episode II - Attack of the Clones	1894	George Lucas
10258	Star Wars: Episode III - Revenge of the Sith	1895	George Lucas
13103	Star Wars: The Clone Wars	12180	Dave Filoni
15814	Empire of Dreams: The Story of the Star Wars T...	76180	Kevin Burns
26986	Star Wars: The Force Awakens	140607	J.J. Abrams
30860	The Star Wars Holiday Special	74849	Steve Binder
31330	Robot Chicken: Star Wars	42979	Seth Green
42880	Rogue One: A Star Wars Story	330459	Gareth Edwards

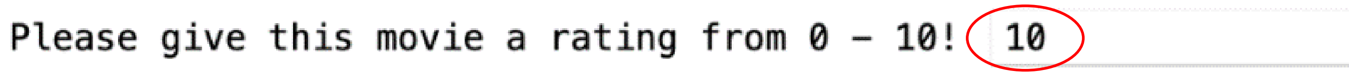
```
There is more than one movie with this name, please enter the ID. 11
```

Step 2: You will see our predicted rating of the movie.



```
Our predicted rating of Star Wars is: 8.0
```

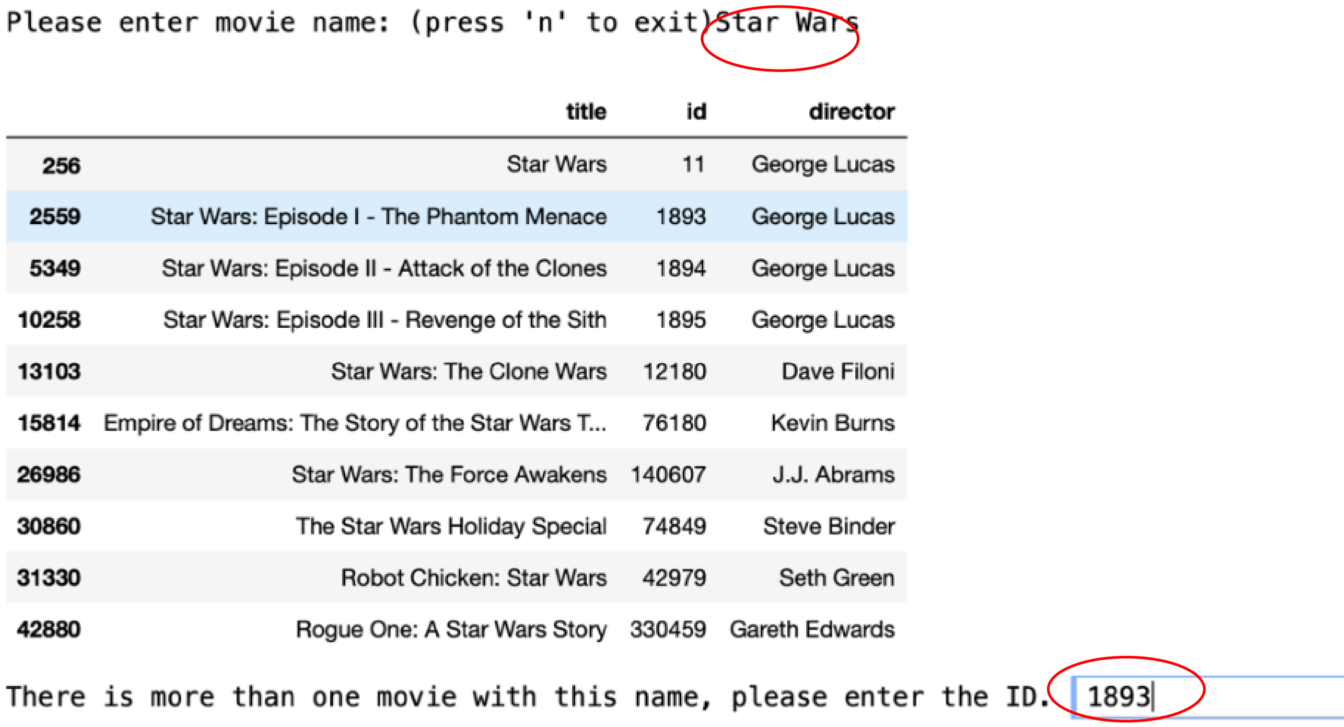
Step 3: Give this movie a rating from 0.0 to 10.0. (We will be giving 10 to show more apparent differences.)



```
Please give this movie a rating from 0 - 10! 10
```

(The rating “10” will be used to update the database and genre preference to personalize movie rating predictions.)

Step 4: Enter movie name, according to our example, we will choose "Star Wars" again, and select the movie id "1893", has the same director as the previously rated movie. (Keep in mind that it very likely also has the same genres!)

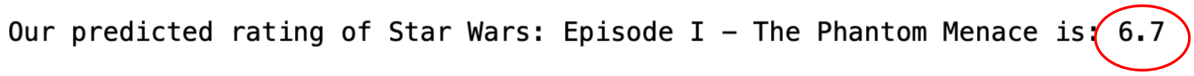


```
Please enter movie name: (press 'n' to exit)Star Wars
```

	title	id	director
256	Star Wars	11	George Lucas
2559	Star Wars: Episode I - The Phantom Menace	1893	George Lucas
5349	Star Wars: Episode II - Attack of the Clones	1894	George Lucas
10258	Star Wars: Episode III - Revenge of the Sith	1895	George Lucas
13103	Star Wars: The Clone Wars	12180	Dave Filoni
15814	Empire of Dreams: The Story of the Star Wars T...	76180	Kevin Burns
26986	Star Wars: The Force Awakens	140607	J.J. Abrams
30860	The Star Wars Holiday Special	74849	Steve Binder
31330	Robot Chicken: Star Wars	42979	Seth Green
42880	Rogue One: A Star Wars Story	330459	Gareth Edwards

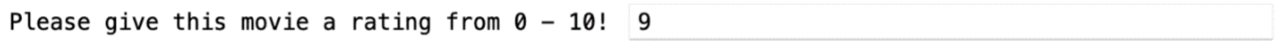
```
There is more than one movie with this name, please enter the ID. 1893
```

Step 5: Remember this prediction rating!!



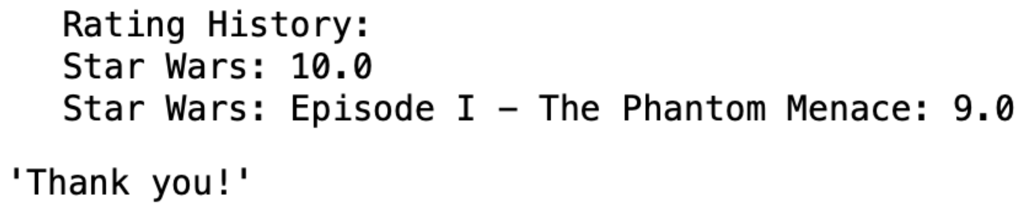
```
Our predicted rating of Star Wars: Episode I - The Phantom Menace is: 6.7
```

Step 6: Give this movie a rating.



```
Please give this movie a rating from 0 - 10! 9
```

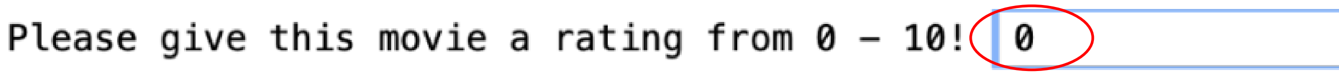
Step 7: Press 'n' to exit and get your rating history.



```
    Rating History:
    Star Wars: 10.0
    Star Wars: Episode I - The Phantom Menace: 9.0

'Thank you!'
```

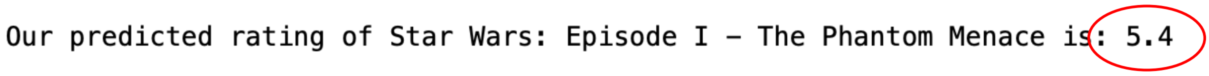
!! If you repeat Step1 to Step7, but entering different rating in Step3, you will get different predicted rating in Step5!!

(Give the movie a rating of 0 in step 3 for more apparent results)



```
Please give this movie a rating from 0 - 10! 0
```

You will get:

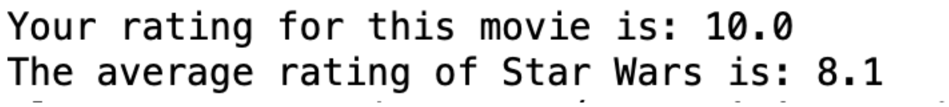


```
Our predicted rating of Star Wars: Episode I - The Phantom Menace is: 5.4
```

5.4 is significantly lower than 6.7, the result if you gave the other movie a rating of 10.

This is the result we want because the 2 Star Wars movies are similar (they have the same genre and director). Therefore, if you like one of them, the chances you will like the other is higher, and vise versa.

Lastly, if you choose a movie for the second time, you will get your rating for the movie and the average rating of the movie.



```
Your rating for this movie is: 10.0
The average rating of Star Wars is: 8.1
```

7. Conclusions and Novelty

In the beginning, we settled on the project of predicting movie ratings based on the data obtainable in the dataset, hoping to develop a model to predict the ratings of future movies. However, after a few tips from Prof. Li, we decided to crop up the difficulty of the project and further design a system that would be able to predict each user's personal movie rating of a given movie, and we accomplished exactly that. Our prediction is based on a simple database and scaled based on the user's movie genre preference. There are many ways to predict personalized movie ratings, such as web scraping, face detection from movie posters, and so on. But when we were discussing such methods, we found many problems, like when you predict based on movie posters, there is a problem that if a poster has one or no human faces, we cannot tell if the movie is great simply from poster. All in all, movie rating prediction is an interesting and promising project. In the future, we will try more advanced methods to predict movie ratings.

8. Contributions of Each Team Member

Code Contribution: 杜霙笙 65%, 陳詠翰 30%, 戰嬍 5%
Oral Presentation: 戰嬍
Written Report: 陳詠翰 40%, 戰嬍 35%, 杜霙笙 25%

9. Bibliography and Resources

https://thesai.org/Downloads/Volume11No8/Paper_49Movie_Rating_Prediction.pdf

https://medium.com/swlh/movie-recommendation-and-rating-prediction-using-k-nearest-neighbors-704ca8ccaff3

https://nycdatascience.com/blog/student-works/web-scraping/movie-rating-prediction/

https://www.analyticsvidhya.com/blog/2020/08/types-of-categorical-data-encoding/