CSC111 project proposal: ScreenSelect

Aastha Sharma, Narges Movahedian Nezhad, Sidharth Sawhney, Dogyu Lee

08 March 2023

Part 1: Goal

Due to the advent of streaming services, which provide customers immediate access to a vast collection of information, the entertainment industry has undergone a vital change. Finding the right things to watch, for example, might just be more difficult than ever for viewers. This is due to the overflow of available content. As a solution to this issue, recommendation systems have been created; personalizing and recommending material based on users' viewing patterns and interests. With this in mind, **our project intends to develop a movie customization and recommendation app that takes inspiration from Netflix's recommendation system, leveraging watching patterns, user input, and clever algorithms to provide tailored movie suggestions.**

Our project's objective is to boost user engagement by means of providing a highly-customized watching experience that changes depending on the user. We will offer a platform that generates suggestions according to a content-based filtering algorithm. We will examine data including but not limited to plot keywords, cast, and genre, to gauge the effectiveness of our app.

Our system will identify the best recommendations for each user every time by taking into account contextual elements—such as the last 10 watched movies, keywords, genre, and language—going beyond Netflix's straightforward suggestions of related material based on user watch history. We want to provide a highly engaging experience that keeps people riveted to the screen and coming back for more by putting a strong emphasis on tailored suggestions.

The limits of conventional content-based filtering algorithms, which only consider the characteristics of each piece of material, such as cast, and director, are yet another issue that our study aims to solve. Although these algorithms are helpful, they do not account for consumers' changing preferences and interests. As the user's preferences change over time, our strategy makes use of both the user's viewing history and contextual information to provide recommendations that are more individualized.

In conclusion, the goal of our project is to develop a highly individualized, entertaining, and impactful movie recommendation app. We seek to offer consumers a watching experience that is customized to their interests and keeps them hooked with our platform over time. This is done through machine learning algorithms and contextual elements. Our ultimate purpose is to enhance user engagement and commitment. We will gauge our progress using a variety of test cases that reflect this goal.

Part 2: Computation

1. Since we are making a movie recommendation system, the entire data including the users and movies will be represented using Graphs. The Graph class consists of a dictionary containing all the vertices, which are both the user and the movies. The Vertices of the graph are objects of the Vertex abstract superclass, where the subclass are different for the **movie vertex** and **user vertex** *(Refer to the appendix)*. The edges in the graph will be made between the user and the movies once the user chooses one movie from the given recommendations. This way the edges will help us suggest more accurate movies on the basis of user input and what movie the user has chosen before; it also prevents us from recommending the same movie.

   The ScreenSelect system requires us to compute real-life data, so an example data set found on Kaggle is the Movie Dataset: tmdb_5000_movies. This dataset consists of two CSV files *(both of which are cited in Part 4: References)*. The image of both the CSV files are below, representing the sample data in both CSV files. The first CSV file contains, the movie's id, title, cast, and crew (figure 1, under Part 5: Appendix), and the second contains multiple statics about each movie (Figures 2, under Part 5: Appendix)

2. As the data for the system comes from a csv file, we need to load our graph with Movie instance objects initialized from observations of the Movie dataset. Every time a new User is created, Graph vertices will be updated with the username and its corresponding User instance. If the user already exists in the system, we will search by username and retrieve the associated User instance, so that we can obtain the previous movies chosen, ie, recommended before. If it is a new user then the user can input their desires for what movie they would like the system to recommend (genre, language, keywords , director), and the system would not take their previous movies watched into consideration. If the user already exists in the system, we will mutate the existing instance attributes with the new values. We have decided to recommend the movies by an associated weight based on the importance of the attribute which will help the system decide which movie to recommend. For example, We have fixed genre as 10 (if the genre desired matches with the genre of the movie) else 5 (if the genre desired doesn't match with the genre of the movie).

   As we traverse through all movie vertices in the graph for the case of a new user, each movie will receive a score depending on whether the movie has its attributes matching with respect to the desires of the user. This total score will be the sum of the weights depending if it's a match or not. For an existing user, the chosen movies will also contribute to the total score by adding the sum obtained by matching the attribute of the chosen movies with the movies we traverse through. For an existing user, we will not traverse through movies in the system which the user has chosen so that we may avoid recommending them again. On determining the scores of all possible movies that can get recommended, we recommend the top 5 based on score. The user will choose one movie from the 5 recommendations. This chosen movie will be added to the neighbors of the user and vice versa for the movie which would represent a newly formed edge.

   For a more in-depth implementation look into the appendix.

3. The user interface will be done with PyQt6 library module in python described below. The user will first see a log-in page that will help them create a new account or log into an existing one. Then they will see a screen of their chosen movies in the past and an option for getting a new recommendation. Upon clicking on the recommendation option, they will fill in multiple inputs that are required for the system and then the loading page will appear. After the loading page, the top 5 recommendations will be shown with their images, stats, and an option to choose this movie. If they choose that movie, it will take them back to the screen of chosen movies in the past until they log off. The color scheme will particularly be dark with hints of bright color to engage the user plus give them a cinema experience.

   See Figure 3 in Part 5: Appendix for the diagrams

4. Our project's user interface will be created by installing the PyQt6 python library and importing the QtWidgets and QtCore modules. QtWidgets module will create various UI elements such as buttons, textboxes, progress bars, and drop-down lists. To use QtWidgets, we will create subclasses that inherit it. For the display of the program, QMainWindow will first be constructed without a parent. Since QMainWindow has a built-in layer, we will create an empty new class that inherits QWidget and use QGridLayout to effectively divide the screen and arrange the created UI elements after they are added to the cell grid using addWidget(). QLineEdit will only be used for user input such as preferences, usernames, and passwords with the help of echoMode(). QTextEdit will be used to display movie recommendations of the program using setHtml(). Using the built-in signal functions of each module's class, such as returnPressed() of QLineEdit, the classes we implemented will communicate with each other and execute "events". Additionally, we plan to use QObject to give distinct names to each object we create and use QMetaObject's connectSlotsByName(), connect(), disconnect(), and emit() to pass in those names to connect objects and trigger them to work depending on the signal they received

Part 4: References—IEEE Format

Course Notes

- D. Liu and M. Badr. "Course Notes for CSC110 and CSC111." Foundations of Computer Science. https://www.teach.cs.toronto.edu/ csc110y/fall/notes/ (accessed Mar. 5, 2023)

Data Set

- A. Dattatray, tmdb_5000_credits, vol.1, Kaggle: Kaggle, 2022. [Dataset]. Available: https://www.kaggle.com/datasets/akshaydattatraykhare/movies-dataset. [Accessed: Mar. 5, 2022].

- A. Dattatray, tmdb_5000_movies, vol.1, Kaggle: Kaggle, 2022. [Dataset].

  Available: https://www.kaggle.com/datasets/akshaydattatraykhare/movies-dataset. [Accessed: Mar. 5, 2022].

Module Documentation

- Riverbank Computing and The Qt Company. "Reference Guide — PyQt Documentation v6.4.1." Riverbank Computing. https://www.riverbankcomputing.com/static/Docs/PyQt6/ (Accessed: Mar. 5, 2022).
- J. Bodnar. "Python PyQt6." ZetCode. https://zetcode.com/pyqt6/introduction/ (Accessed Mar. 7, 2023)
- The Qt Company. "Qt for Python" Qt. https://doc.qt.io/qtforpython/index.html (Accessed Mar. 8, 2023)

Idea Research

- M. Gavira. "How Netflix uses AI and Data to conquer the world." LinkedIn. https://www.linkedin.com/pulse/how-netflix-uses-ai-data-conquer-world-mario-gavira/ (accessed Mar. 7, 2023)
- J. Ciancutti. "How We Determine Product Success." Netflix Technology Blog. https://netflixtechblog.com/how-we-determine-product-success-980f81f0047e (accessed Mar. 7, 2023)
- D. Chong. "Deep Dive into Netflix's Recommender System." Towards Data Science. https://towardsdatascience.com/deep-dive-into-netflixs-recommender-system-341806ae3b48 (accessed Mar. 7, 2023)

Part 5: Appendix

Implementation:

1. **Graph Class:** _vertices : dict[str or int , _User or _Movie]

   Key: str or int. It would be str if the value is a _User and int if the value is _Movie. Str would represent the usernames and int would represent the IDs of the movie. Value: _Movie or _User. _Movie and _User classes are subclasses of the abstract class _Vertex. Therefore The value can be represented as _Vertex. On initializing an empty graph, all of the above instance attributes are empty as well.

2. **_Vertex (abstract class)** item: int or str ID (for the movie) or Username (for the user)

   genre: str (a genre of the movie or genre the user wants to watch)

   lang: str (language of the movie or the language of the movie the user wants to watch)

   keywords: Optional[set[str]] Note: themes/keywords associated for a movie or themes/keywords for a movie that the user wants to watch.

   Director: Optional[str]

   neighbours : lst[_Movie] or dict[str , _User] The type contract lst[_Movie] will be used for _User instances. dict[str , _User] will be used for _Movie instances. The _Movie is added to the neighbors of the user if the user accepts that movie recommendation. lst[_Movie] will have a size of not more than 10. Every time a _Movie is added once the size of neighbors is 10, the oldest _Movie instance at the 0th index will be popped out. The corresponding chosen movie's neighbors will add the user which chose that movie.

   Representation Invariant:

   all(self in u.neighbours for u in self.neighbours) This ensures symmetry between 2 vertices that share an edge.

   self not in self.neighbors This will ensure that the same vertex is never in the neighbors of that vertex.

3. **_Movie (_Vertex superclass)**:

   Title: str

   Vote_average: float

   Overview: str

   Runtime: int (in minutes)

   Release Date: str

   _total_score : int = 0

Representation Invariant

all(type(self.neighbours[u]) is not type(self) for u in self.neighbours)

This ensures that an edge exists only between a _Movie and _User instance.

4. **_User (_Vertex superclass):**

   all_neighbours : set[_Movie] Stores the all movies that the user chooses

   _top_scores : dict[int, _Movie] This dictionary will always have a size of not more than 5. This dictionary will contain the top 5 movies to be recommended based on the highest total scores. The keys represent the total score and the value is the associated _Movie instance.

   Representation Invariant _top_scores will never contain a movie recommendation at any point in time for a movie that has already been chosen by that user (that exists in the neighbors attribute or non_top_neighbours attribute).

   all(type(u) is not type(self) for u in self.neighbours) This ensures that an edge exists only between a _Movie and _User instance.

5. Create an empty Graph. The first vertex added to a graph must be a _Movie instance, not a _User instance. We will first load our graph with _Movie instance initialized from observations of the Movie dataset. After loading the graph, the graph instance _vertices attribute will contain all movie ID's with their associated _Movie instances.

   To load the graph with vertices, we will filter the data in the csv files by column and transform the data types to match the instance attribute data types of _Movie. Then, each observation/row's data will be stored in a _Movie vertex and mutated in the Graph's _vertices in a key: value form.

   Every time a new _User is created, Graph._vertices will be updated with the username and its corresponding _User instance. If the user already exists in the system, we will search by username and retrieve the associated _User instance. If it is a new user then the user can input their desires for what movie they would like the system to recommend (genre, language, keywords , director). If the user already exists in the system, we will mutate the existing instance attributes with the new values.

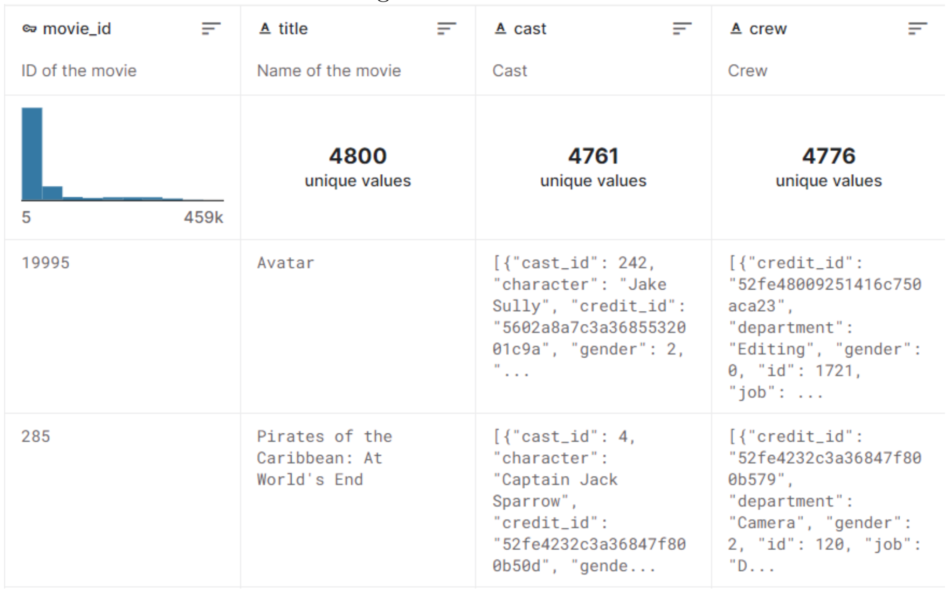
   When a previous user returns, we will perform mutation or variable reassignment on the user class to add another movie to its neighbour and possibly add or replace other instance attributes depending on its input.

   Each attribute of _Vertex (and therefore _User and _Movie subclasses) has an associated weight based on the importance of the attribute which will help the system decide which movie to recommend. We have fixed genre and language attribute = 10 (if the genre desired matches with the genre of the movie) else 5 (if the genre desired doesnt match with the genre of the movie) , for every keyword that the user entered = 5 (if the keyword matches with the keyword of the movie), else 0. Items attribute will always = 0 and director = 5 (if the director matches with the director of the movie else), it will be 0.

   For a new or existing user, we will compare the desires of the user with the existing parameters of each movies in the system. Given the weights for each case, we will add the weights to get a total score for a movie and we will do this comparison for all movies. As we compute the score for each movie, we will add to _User._top_scores for the first 5 movies, thereafter it will add the _Movie instance if only the score is greater than the scores of any one of the existing movies _top_scores, and remove the element whose key has the lowest score in the dictionary. After all comparisons, we display the titles of the movies in _top_scores to the user. The user will then choose the movie they want to watch from the recommended options by title of the movie. After choosing the choosing, we will add the movie chosen as an element of the neighbors attribute of the _User instance and all_neighbours attribute of the _User instance and add the user to the neighbors attribute of the corresponding chosen _Movie instance.

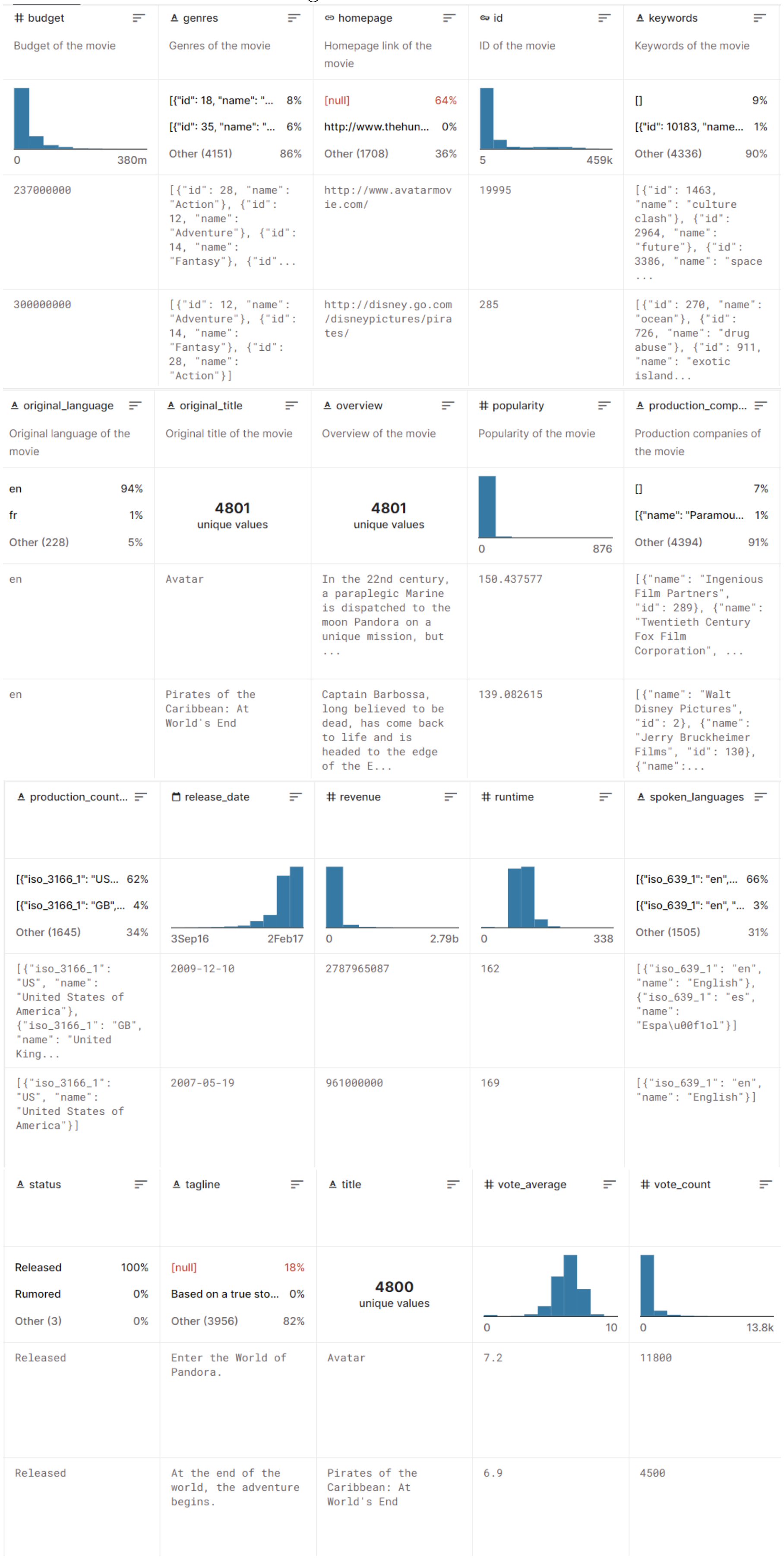
   We will aggregate the weights of matching parameters to obtain a summed score for each _Movie vertex. As the scores are calculated and stored, the system will add the 5 highest scoring vertices into _User._top_scores by performing a comparison between the first five vertices' scores and the next vertex's score. If any of the five vertices' scores are lower, the lower vertex will be deleted from _top_scores and the higher vertex will be mutated into _top_scores. Resultantly, only the top 5 highest scoring _Movie vertices are left in _top_scores to be recommended to the user.

Figure 1: CSV File 1



movie_id	title	cast	crew
ID of the movie	Name of the movie	Cast	Crew
5 459k	4800 unique values	4761 unique values	4776 unique values
19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "credit_id": "5602a8a7c3a3685532001c9a", "gender": 2, "...	[{"credit_id": "52fe48009251416c750aca23", "department": "Editing", "gender": 0, "id": 1721, "job": ...
285	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Sparrow", "credit_id": "52fe4232c3a36847f800b50d", "gende...	[{"credit_id": "52fe4232c3a36847f800b579", "department": "Camera", "gender": 2, "id": 120, "job": "D...

Figure 2: CSV File 2



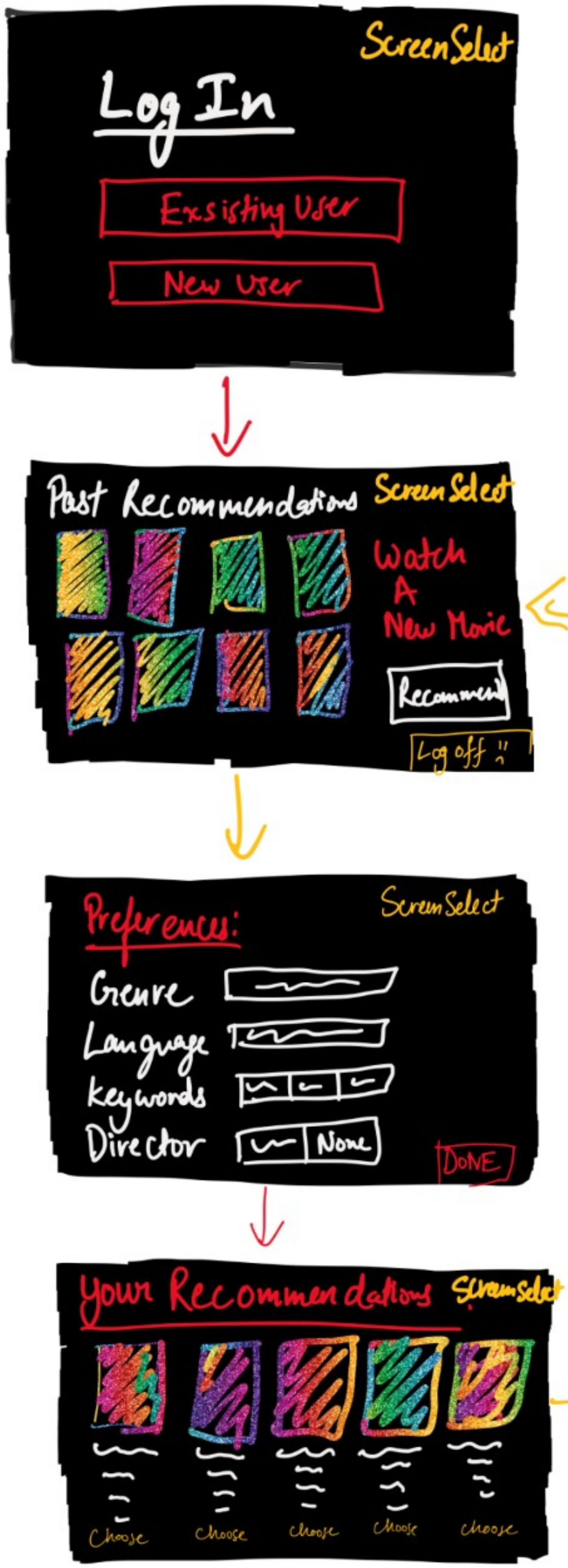
# budget	genres	homepage	id	keywords
Budget of the movie	Genres of the movie	Homepage link of the movie	ID of the movie	Keywords of the movie
0 380m	[{"id": 18, "name": "... 8% [{"id": 35, "name": "... 6% Other (4151) 86%	[null] 64% http://www.thehun... 0% Other (1708) 36%	5 459k	[] 9% [{"id": 10183, "name... 1% Other (4336) 90%
237000000	[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {"id"...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id": 2964, "name": "future"}, {"id": 3386, "name": "space ...
300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {"id": 28, "name": "Action"}]	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "name": "drug abuse"}, {"id": 911, "name": "exotic island...

original_language	original_title	overview	# popularity	production_comp...
Original language of the movie	Original title of the movie	Overview of the movie	Popularity of the movie	Production companies of the movie
en 94% fr 1% Other (228) 5%	4801 unique values	4801 unique values	0 876	[] 7% [{"name": "Paramou... 1% Other (4394) 91%
en	Avatar	In the 22nd century, a paraplegic Marine is dispatched to the moon Pandora on a unique mission, but ...	150.437577	[{"name": "Ingenious Film Partners", "id": 289}, {"name": "Twentieth Century Fox Film Corporation", ...
en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, has come back to life and is headed to the edge of the E...	139.082615	[{"name": "Walt Disney Pictures", "id": 2}, {"name": "Jerry Bruckheimer Films", "id": 130}, {"name":...

production_count...	release_date	# revenue	# runtime	spoken_languages
[{"iso_3166_1": "US... 62% [{"iso_3166_1": "GB",... 4% Other (1645) 34%	3Sep16 2Feb17	0 2.79b	0 338	[{"iso_639_1": "en",... 66% [{"iso_639_1": "en", "... 3% Other (1505) 31%
[{"iso_3166_1": "US", "name": "United States of America"}, {"iso_3166_1": "GB", "name": "United King...	2009-12-10	2787965087	162	[{"iso_639_1": "en", "name": "English"}, {"iso_639_1": "es", "name": "Espa\u00f1ol"}]
[{"iso_3166_1": "US", "name": "United States of America"}]	2007-05-19	961000000	169	[{"iso_639_1": "en", "name": "English"}]

status	tagline	title	# vote_average	# vote_count
Released 100% Rumored 0% Other (3) 0%	[null] 18% Based on a true sto... 0% Other (3956) 82%	4800 unique values	0 10	0 13.8k
Released	Enter the World of Pandora.	Avatar	7.2	11800
Released	At the end of the world, the adventure begins.	Pirates of the Caribbean: At World's End	6.9	4500

Figure 3: User Interface Diagram


Screen Select
Log In
Exsisting User
New User
Past Recommendations
Screen Select
Watch A New Movie
Recommend
Log off !!
Preferences:
Screen Select
Genre
Language
keywords
Director
None
DONE
Your Recommendations
Screen Select
Choose
Choose
Choose
Choose
Choose