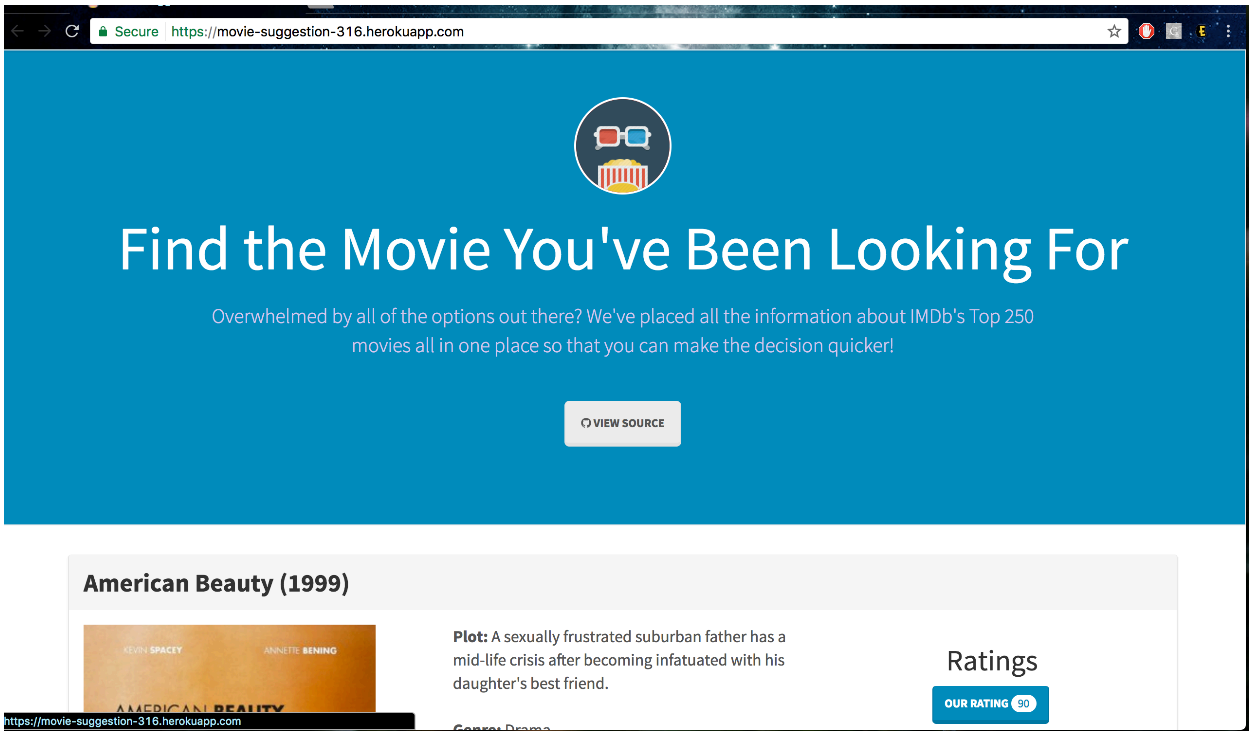
Final Project Report
Group 17, Movie Recommendation

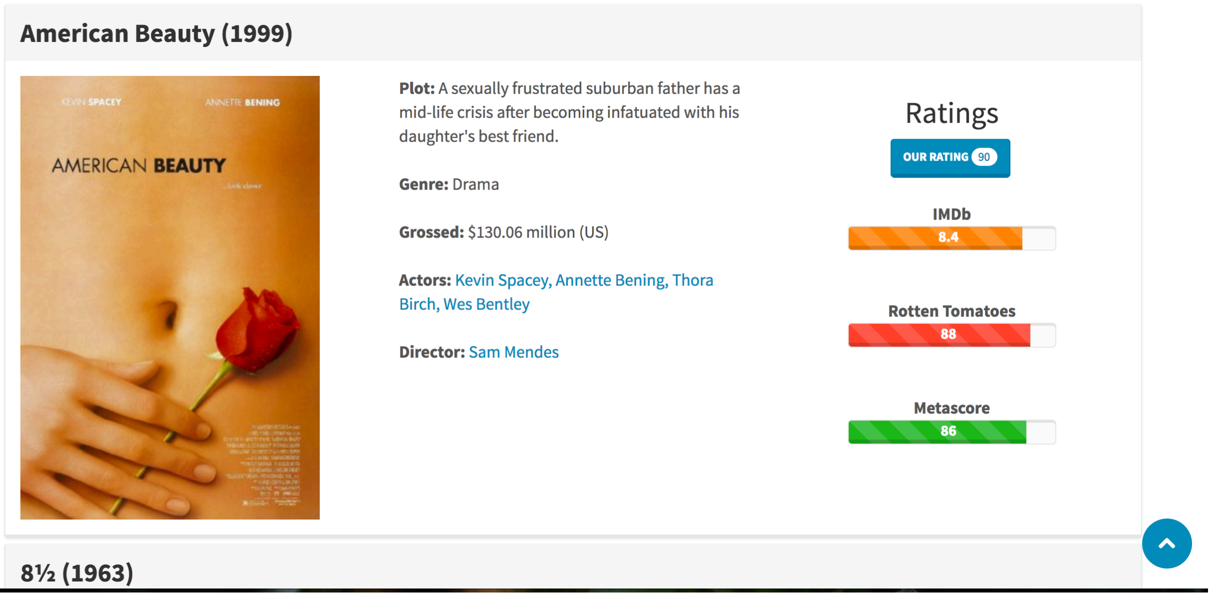
Brief description of our application

The following is our site. The url to access it is https://movie-suggestion-316.herokuapp.com/


Secure https://movie-suggestion-316.herokuapp.com
Find the Movie You've Been Looking For
Overwhelmed by all of the options out there? We've placed all the information about IMDb's Top 250 movies all in one place so that you can make the decision quicker!
VIEW SOURCE
American Beauty (1999)
Plot: A sexually frustrated suburban father has a mid-life crisis after becoming infatuated with his daughter's best friend.
Ratings
OUR RATING 90

On the top of the page, we highlight the theme which is also the purpose of our web app-helping people to find the movies they are looking for. Instead of looking through different movie rating sites, we summarize the rating information from different popular sites for the users. One of the functionality of our sites is user can rank movies by any of the provided movie rating site or by default our rating. The rating we currently included are imdb, rotton tomatoes, matascore. Our rating score is based on an algorithm to average those scores based on weights we give to each sites. The weighs for each site is based on the popularity and completeness of that site. If a movie does not have rating on some rating sites, the available rating sites' rating becomes weighted more heavily.

In addition to the first function which is compare and generate our own rating, we also provide movie recommendation options for the user if user choose to create an account and log in. We store user account information in our database. User will also indicate the movies they like and we will find the similar movies to recommend to users. We will find the movies that are directed by the same director, or similar plot, or same genre and rank them by our rating score as our movie suggestions for the users. The movie recommendation process is a dynamic process which will change as user likes more movies.


American Beauty (1999)

Plot: A sexually frustrated suburban father has a mid-life crisis after becoming infatuated with his daughter's best friend.

Genre: Drama

Grossed: $130.06 million (US)

Actors: Kevin Spacey, Annette Bening, Thora Birch, Wes Bentley

Director: Sam Mendes

Ratings

OUR RATING 90

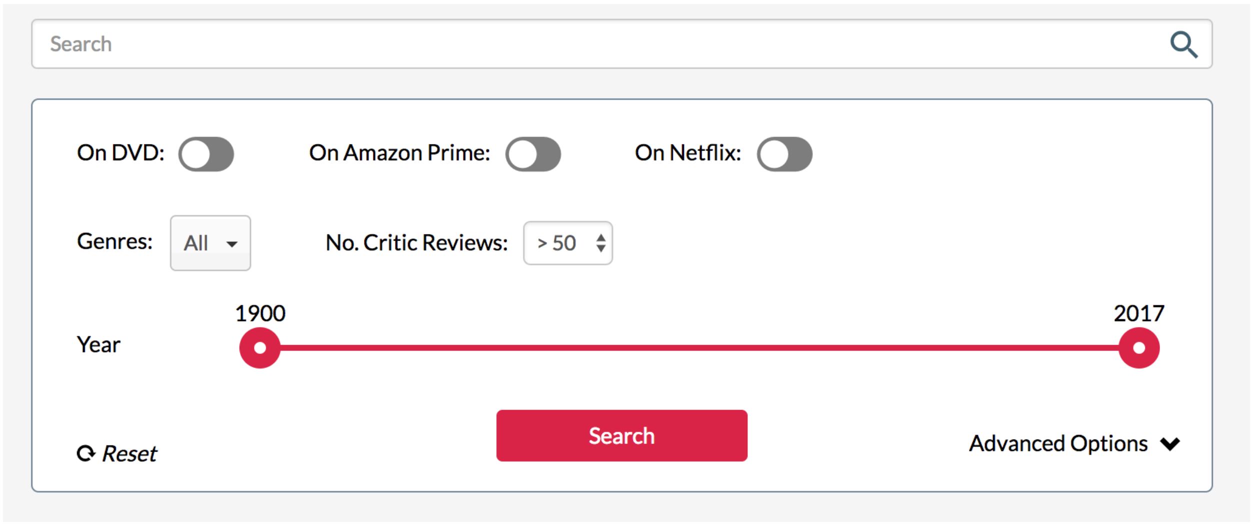
IMDb 8.4

Rotten Tomatoes 88

Metascore 86

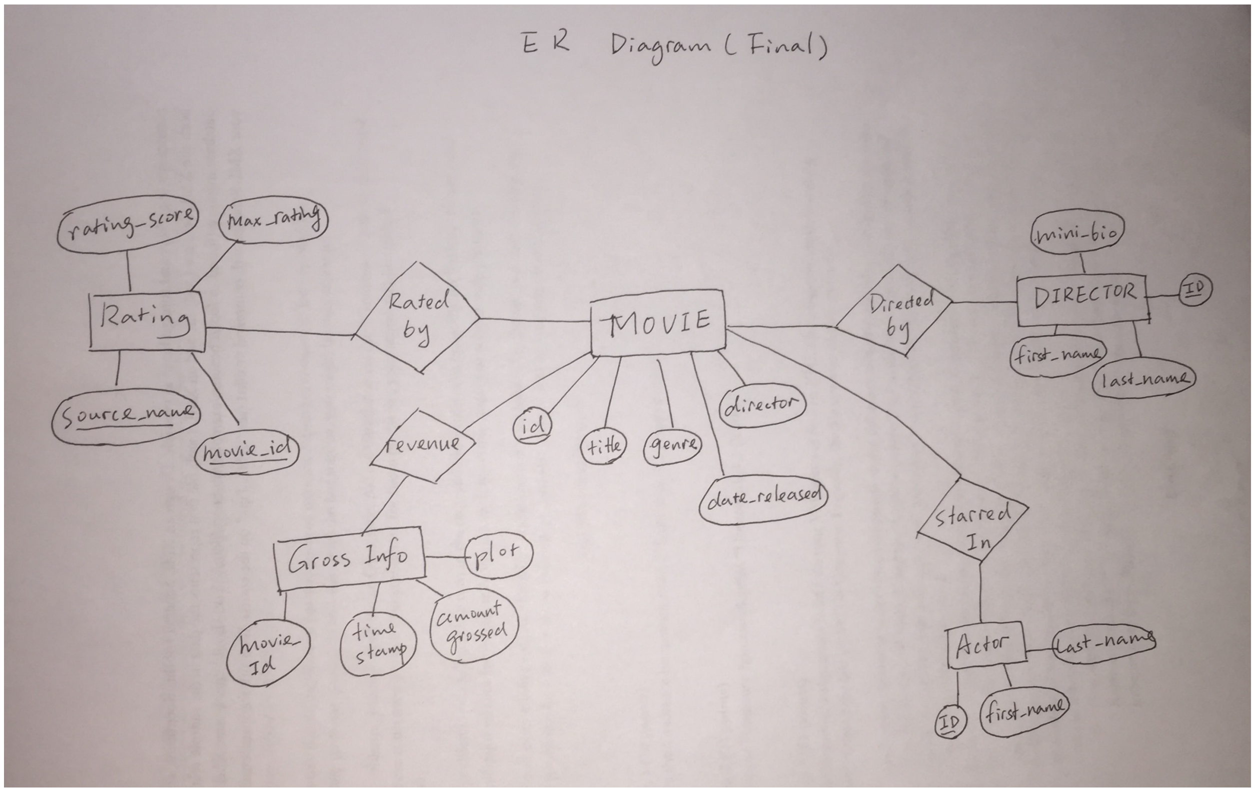
8½ (1963)

Another feature which we offer is run user defined queries. User can search movies based on key words in title, director, or genre or year of the movie. The backend will get the user input and we will generate the queries to get the corresponding information from the database and then show the results on the front end.



In terms of how we build this web app. All our source code is public on git via this url: git@github.com:yuminz/316Spring2017.git. We used postgre sql to establish our database and scape the movie data from popular sites mostly by api. For more details, please see the python code in the extra folder which includes how we used each site to insert data into each table. We use heroku to host our web app, and it comes with aws to store our database. For the frame work, we used Django. For front end, we used html, css, and javascript to make the UI. For back end we use python to connect to database and generate queries and return and render the result on the UI. All the queries are dynamically generated as the user writes input into UI. Our back end function will be called and respond correspondingly.

The updated E/R diagram for the final database design



Assumptions that you are making about the data being modeled

We had more assumptions earlier, but now we kind of settle down in terms of how to handle incomplete or inconsistent data. I don't think we have much assumptions now about the data, but let me clarify how we handle those incompleteness and inconsistency. For the movie information, we are mainly relying on imdb, for couple reasons. First imdb has a pretty complete database comparing to other movie sites and it is update frequently. Thus, we decided not to double checking movie information with other movie sites to ensure the consistency. Most of our data incompleteness from the rating part, because some sites are missing the rating for some movies. We handle that by creating our weighted average rating function. In the function, if a site's rating is missing, the other sites the rating will increase proportionally to ensure it will still all add up to 100%.

The list of database tables with descriptions

I think all the tables are pretty intuitive with its name and attributes. Thus I directly attached our create_db.sql in the following. And those information matches perfectly with our E/R diagram above, since we updated the E/R diagram. The only table that is not represented here is the user account and password information. For security reason, we do not directly store user account information in the database, but rather go through some hashing and cryptographic method to encapsulated those info.

```
CREATE TABLE IF NOT EXISTS Director
(
  id INTEGER NOT NULL PRIMARY KEY,
  first_name VARCHAR(256) NOT NULL,
  last_name VARCHAR(256) NOT NULL
  mini_bio TEXT
);

CREATE TABLE IF NOT EXISTS Movie
(
  id INTEGER NOT NULL PRIMARY KEY,
  title VARCHAR(256) NOT NULL,
  genre VARCHAR(256) NOT NULL,
  director INTEGER NOT NULL REFERENCES Director(id),
  date_released TIMESTAMP NOT NULL
);

CREATE TABLE IF NOT EXISTS Rating
(
  id UUID NOT NULL PRIMARY KEY,
  source_name VARCHAR(256) NOT NULL,
  movie_id INTEGER NOT NULL REFERENCES Movie(id),
  max_rating INTEGER NOT NULL,
  rating_score FLOAT NOT NULL CHECK(rating_score >= 0 AND rating_score <= max_rating)
);

--Amount grossed in is millions
CREATE TABLE IF NOT EXISTS GrossingInfo
(
  movie_id INTEGER NOT NULL REFERENCES Movie(id),
  date_timestamp TIMESTAMP NOT NULL,
  amount_grossed FLOAT NOT NULL,
  plot TEXT
);

CREATE TABLE IF NOT EXISTS Actor
```

```
(
  id INTEGER NULL PRIMARY KEY,
  first_name VARCHAR(256) NOT NULL,
  last_name VARCHAR NOT NULL
);

CREATE TABLE IF NOT EXISTS StarredIn
(
  actor_id INTEGER NOT NULL REFERENCES Actor(id),
  movie_id INTEGER NOT NULL REFERENCES Movie(id)
);
```