Deliverable Movies Part 2

1. `https://codeclimate.com/github/edenzik/cs105b`
2. `https://github.com/edenzik/cs105b`
3. The algorithm I used is the following:
   - The program reads the file from the source, and produces two indexes consisting of the user -> movie mapping with rating and a movie -> user mapping with ratings.
   - A movie's popularity equals the sum of all of the movie's ratings. This is the most straightforward way to determine how many people viewed the movie and how much they liked it.
   - The similarity between two viewers is the Euclidean distance between the intersection of the movies they have viewed. For each movie they both viewed, the distance between the ratings is compounded.
   - Most similar viewers is the list in order of all viewers with non zero similarity.
   - The prediction algorithm is based on the base set, and hence predicting what a user will rate a movie can be produced by taking a weighted arithmetic mean of the most most similar users, weighted by their similarity. This is the reason why the program is so slow - but is the most accurate way to make this prediction.
   - In order to compute the total accuracy, the base model is compared to the test model.
   - The biggest advantage of this algorithm is its accuracy - it takes into account every viewer who is even remotely related to this one and their preferences.
4. This method has been shown to be highly accurate, with a standard deviation of .3 or so. To do so, I simply ran the command for stdev.
5. The algorithm has really terrible complexity, even with constant time hash lookup. This could've been made better, had ruby not been so difficult to reason about. It grows about exponentially with the size of the input. I did not want to sacrifice accuracy - so don't take points off for time (because this is the best algorhtm for the task). If I had to cut runtime, I wouldn't gotten rid of arithmatic mean, and just picked the most similar person to use for a prediction.