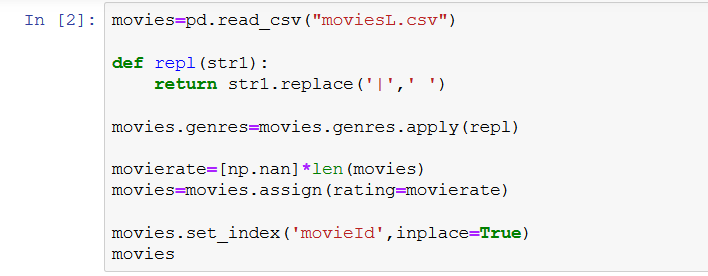
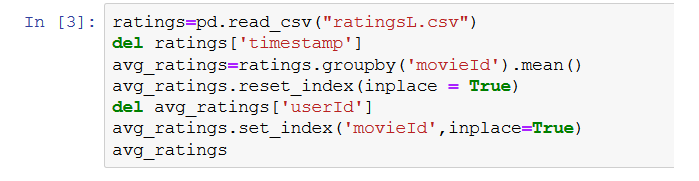
**Content based Filtering -**

**Output -**

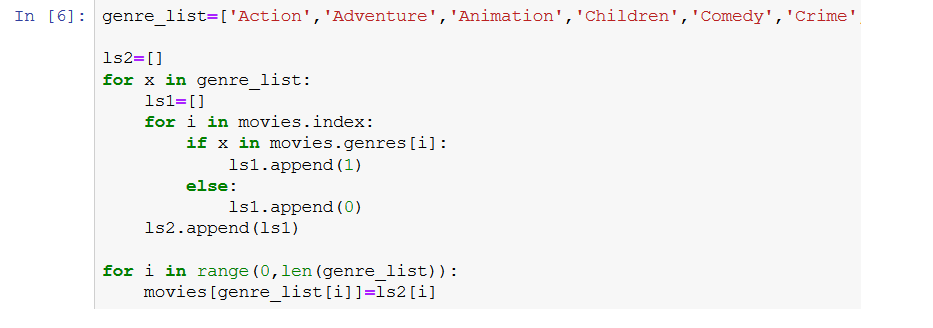
Initially, the movies dataset is quite inconvenient for our application. So, a few steps must be taken beforehand to ensure the analysis goes smoothly. In the genres column of the dataframe, the ‘|’ between the genres listed are replaced by spaces, making further manipulation easier.



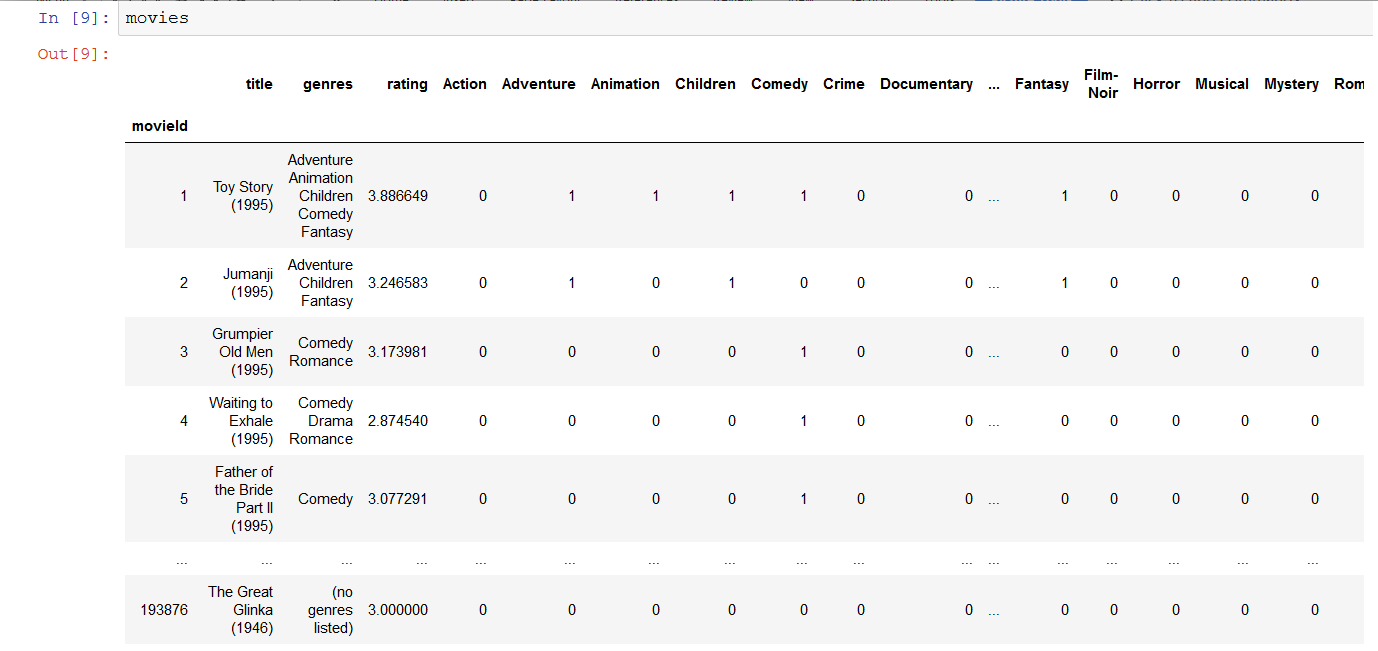
The next step is to add the average ratings of each movie as calculated from the several ratings given by thousands of users.



This code is a manual implementation of the one hot encoding technique. It creates a separate column for each movie genre.

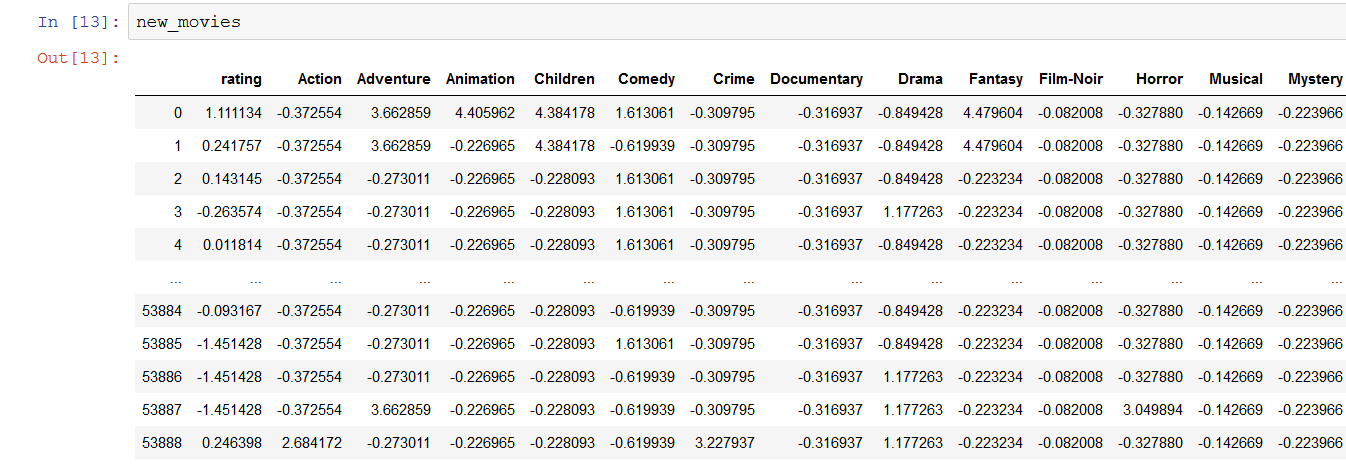


The movies dataframe now looks like this -



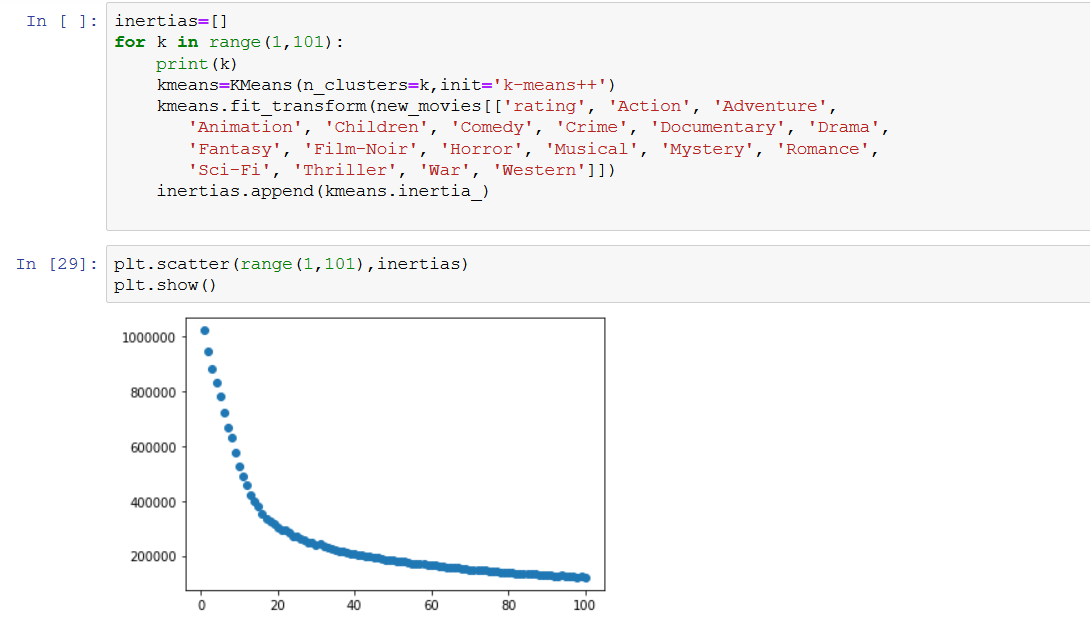
The only step left before applying the K - means algorithm is to standardize the data. This is accomplished using StandardScaler. It ensures that no single column will overshadow the rest of the columns since k - means works based on Euclidean distances.





This is how the movies dataframe looks after scaling. The mean of each individual column is 0 and the standard deviation is 1.

The ‘k’ value used in the method is determined by using the elbow method. K is the number of clusters to be used. This method tells us the optimal number of clusters after which any more clusters would contribute more cost than benefit. We choose k = 29.



Result -

Thus, k-means algorithm gives us clusters of movies based on their ratings and genres. Movies belonging to the same cluster are similar. Using fewer clusters would have been an example of underfitting i.e. the algorithm does not know the data well enough. The opposite end of the spectrum is overfitting i.e. the algorithm knows the data too well and finds complex patterns of similarity instead of more obvious ones.



Fig: Cosine similarity calculation

Tfidf is used to vectorize the genres so that it can be interpreted by the computer. Cosine similarity for every movie with respect to every other movie is calculated to find similar movies. Cosine similarity for every movie with respect to itself is 1 as there angle between two same vectors is and cos 0 is 1

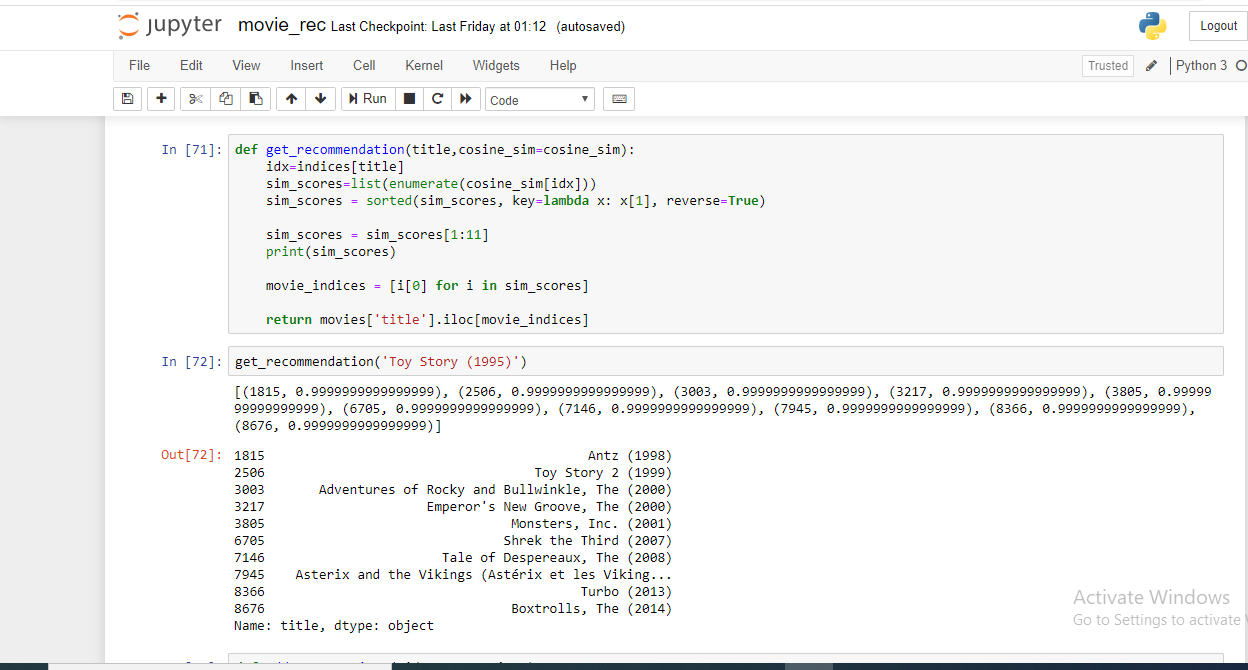


Fig: Finding similar movies using cosine similarity

Using the cosine similarity matrix for every movie it’s possible to find a sorted list of movies with descending. In the above figure we see that toy story has high cosine similarity values for movies like antz and toy story 2 which makes sense cause all of them are animated

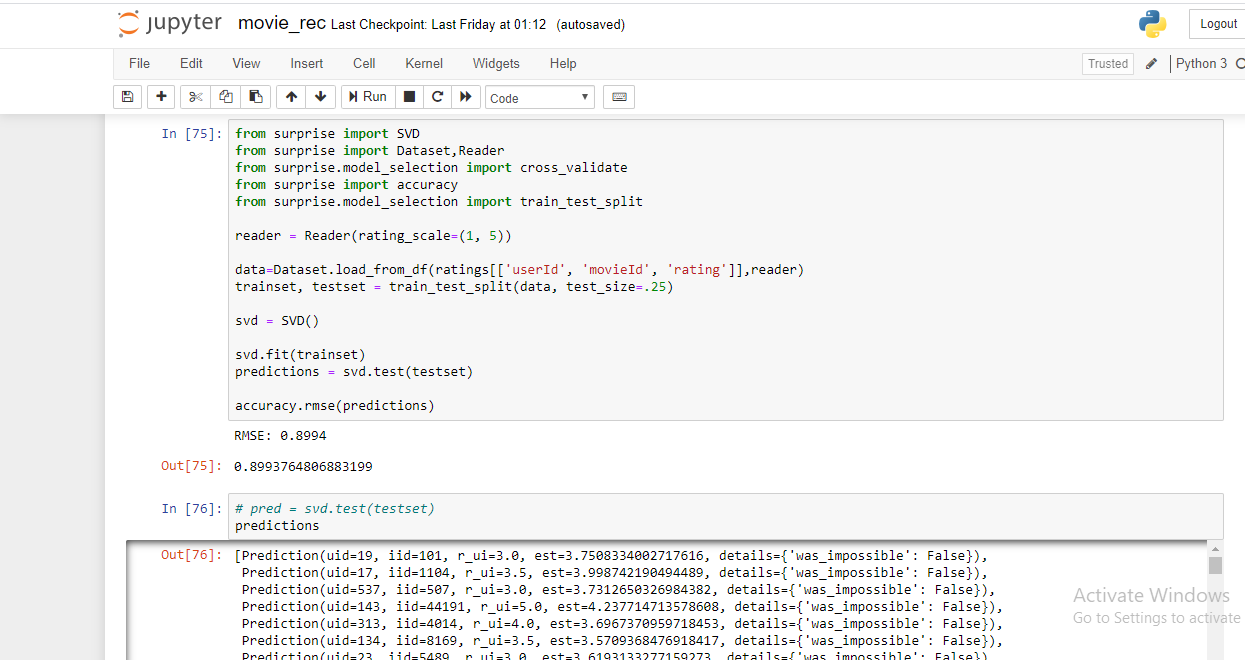


Fig- SVD to compute predictions

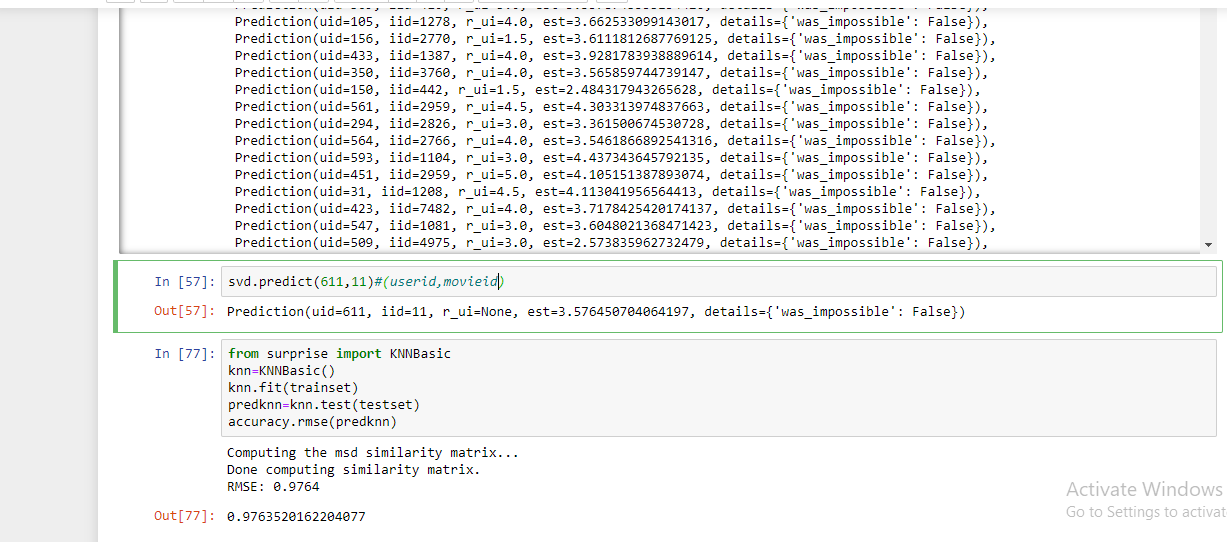


Fig- SVD to compute predictions

SVD is used to predict ratings for movies not rated by the users based on movies rated by them by finding users similar to them. This is done by factorizing the ratings matrix into A=USV**ᵀ** where U and V are orthogonal matrices with orthonormal eigenvectors chosen from AAᵀand AᵀA, respectively. S is a diagonal matrix with r elements equal to the root of the positive eigenvalues of AAᵀ or Aᵀ A. U is an n × r user-latent feature matrix, V is an m × r movie-latent feature matrix.

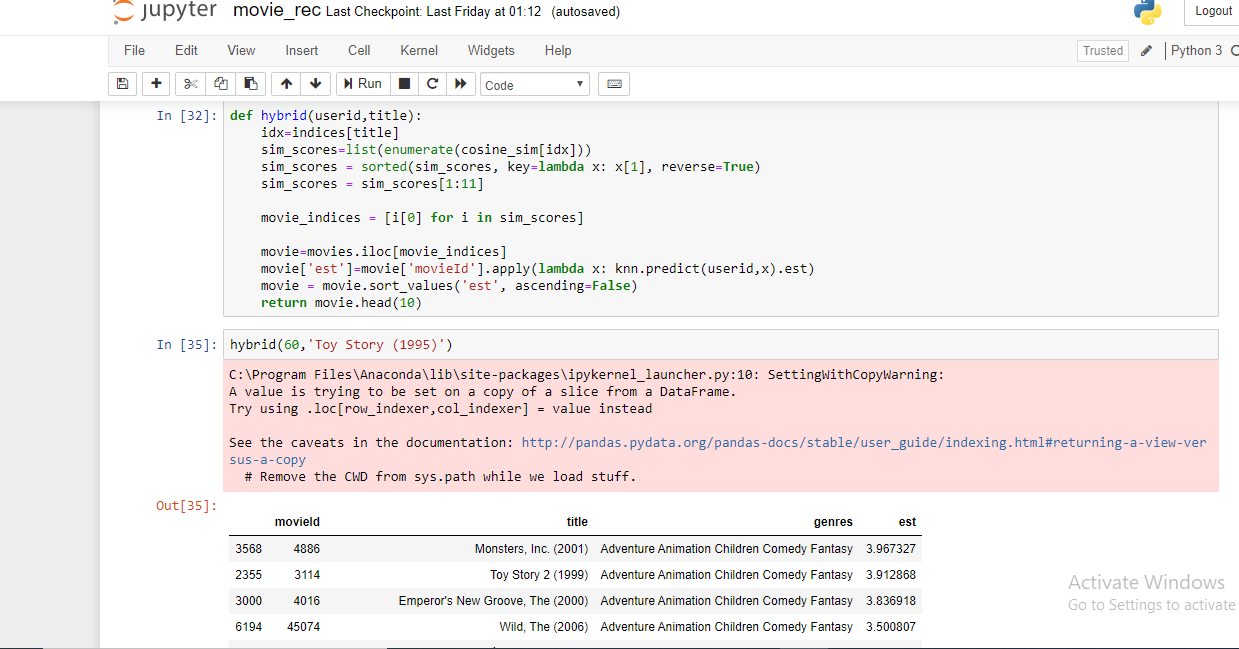


Fig: Combination of content based and collaborative filtering

Collaborative filtering and content based filtering both have their disadvantages when used separately which is why we use hybrid filtering which is a combination of both to create a more efficient and accurate method for predicting the estimated ratings for a user with a certain movie preference.

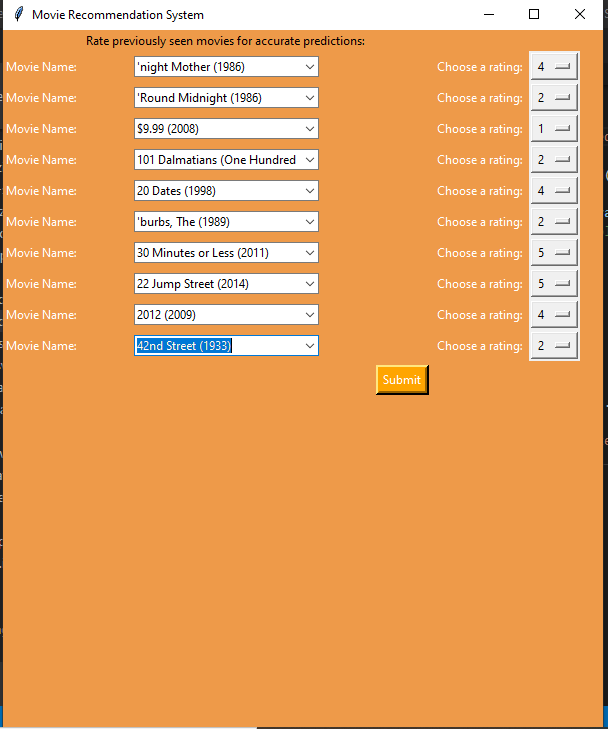


Fig: Finding predictions for new user based on previously seen movies using SVD

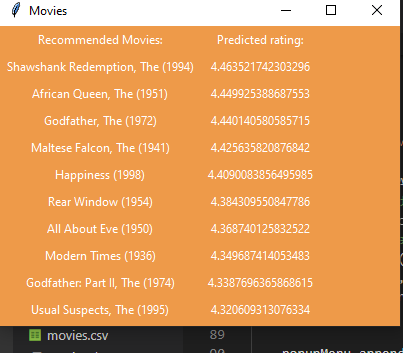


Fig: Prediction for new user using SVD

In the above figure we have taken ratings from a new user to avoid the cold start problem and added it to the ratings table. SVD is then used on this new ratings table to calculate ratings for the new user to predict ratings that user would give to the movies and use these ratings to recommend 10 movies that the user would like.