THE STATE UNIVERSITY OF NEW JERSEY

RUTGERS

MITA CAPSTONE PROJECT
(20209:544:688:05)

Movie Recommender System

By Akshata Abhay Bhende

RUID: 197007541

Under the guidance of
Prof. Michail Xyntarakis

INTRODUCTION

With the rise of YouTube, Netflix, Amazon, and many other web services over the last few decades, recommendation systems have gained more and more positions in our lives.

From e-commerce (suggesting customers products that might interest them) to online ads (offering users the right content, matching their preferences), recommendation systems are unavoidable in our everyday online journeys today.

In a very general way, recommended systems aim to recommend related things to the user (objects being movies to watch, goods to buy, or something else depending on industry).

This project focuses on implementing different recommendation engines, namely the Simple Recommender, the Content-Based Engine Recommender, and Collaborative Recommender Engine. And to further personalized our system, we combined content-based and collaborative recommender to obtain a Hybrid Recommender Engine.

GOALS

- To give users a better experience so that they can spend more time watching than wasting time searching.
- To provide a recommender engine to streaming services like Netflix to attract movie lovers who will increase their revenue and Sales.

DATA COLLECTION

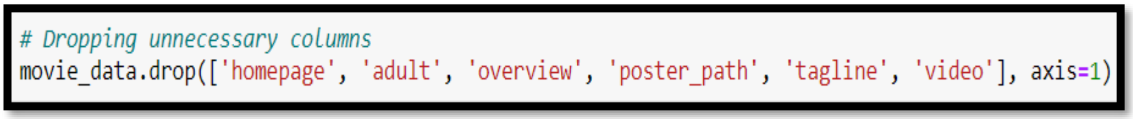
Dataset is taken from MovieLense website. It contains 26 million ratings and 750,000 tag applications applied to 45,000 movies by 270,000 users.

Also, a small dataset containing 10,000 ratings for 9000 movies from 700 users is available. Files used in the project are as follows:

- **movie_metadata.csv**: - The metadata file has the TMDB reviews of 45,000 films, with many features like genre, ratings, budget, revenue, released date, etc.
- **credits.csv:** - Credits of any movie, i.e., cast, crew, producer, director, etc.
- **keywords.csv:** - plot keywords associated with the film.
- **links.csv:** - Contains a list of movies included in the small subset of Full Movie Dataset.
- **ratings.csv:** - This file contains 100,000 ratings on 9,000 movies from 700 users.

DATA WRANGLING

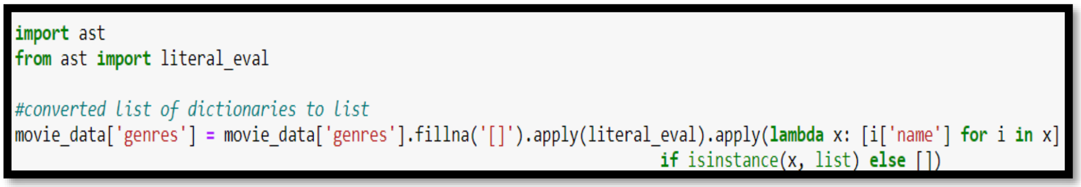
- The dataset has a total of 24 columns. Some features such as Adult, Homepage, Overview, poster_path, tagline, video were unnecessary attributes and were dropped to reduce the dimensionality of the dataset.



```
# Dropping unnecessary columns
movie_data.drop(['homepage', 'adult', 'overview', 'poster_path', 'tagline', 'video'], axis=1)
```

- The dataset had a lot of features that had 0's for values it did not possess. These values were replaced by NaN.

- There were few attributes like genre, cast, belongs_to_collection, which were in the form of Stringified JSON Object. They were converted to a list since we did not require their ID and other attributes.



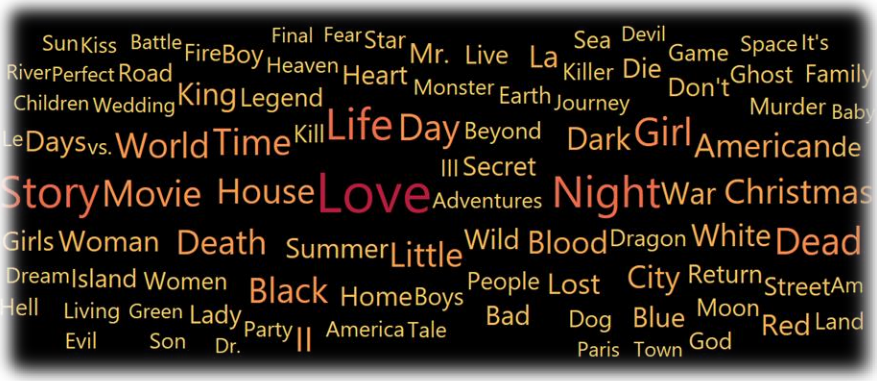
```
import ast
from ast import literal_eval

#converted list of dictionaries to list
movie_data['genres'] = movie_data['genres'].fillna('[]').apply(literal_eval).apply(lambda x: [i['name'] for i in x]
                                                                                    if isinstance(x, list) else [])
```

- Finally, most of the columns were converted into basic python data types like integers, strings, float, etc.

- Features of our dataset:
  - **belongs_to_collection:** Contains the name of the movie franchise
  - **Budget:** Budget of the movie in dollars.
  - **Genres:** list of the genres associated with the movie.
  - **Id:** ID of the movie
  - **Imdb_id**: contains IMDB id of the movie
  - **Original_language:** Language in which the movie was originally shot.
  - **Original_title:** the original title of the movie
  - **Popularity:** popularity scores as assigned by IMDB.
  - **Production_companies:** list of production companies involved with the making of the movie.
  - **Production_countries:** list of countries where the movie was produced in.
  - **Release_date:** Theatrical Release Date of the movie.
  - **Revenue:** total revenue of the movie in dollars.
  - **Runtime:** runtime of the movie in minutes.
  - **Spoken_languages:** list of spoken languages in the film.
  - **Status:** status of the movie (Released, To Be Released, Announced, etc.)
  - **Title:** Official Title of the movie.
  - **Vote_average:** average rating of the movie.
  - **Vote_count:** number of votes by users, as counted by TMDB.

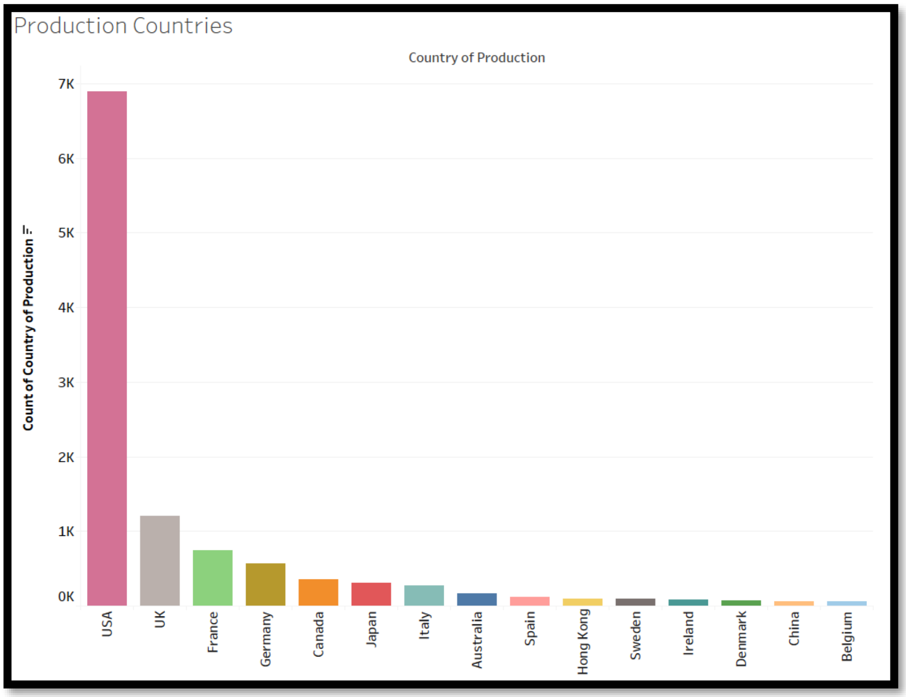
DATA VISUALIZATIONS

Title and Overview Word clouds



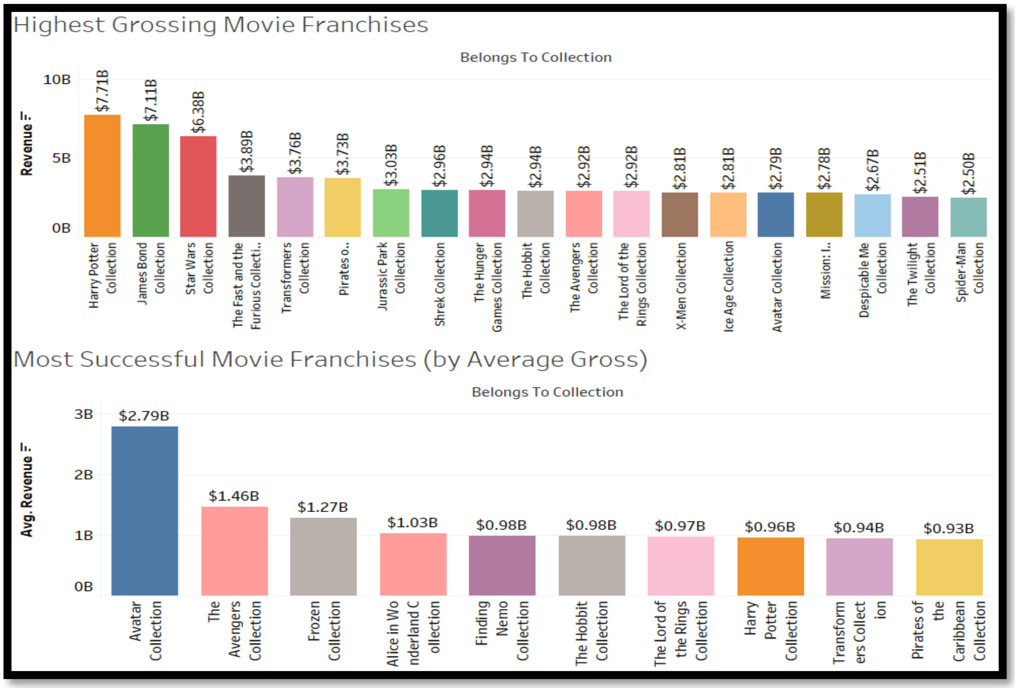
- The word Love is the term most used in film titles. Girl, Story, and Night are among the most frequently used words, too.

Production Countries



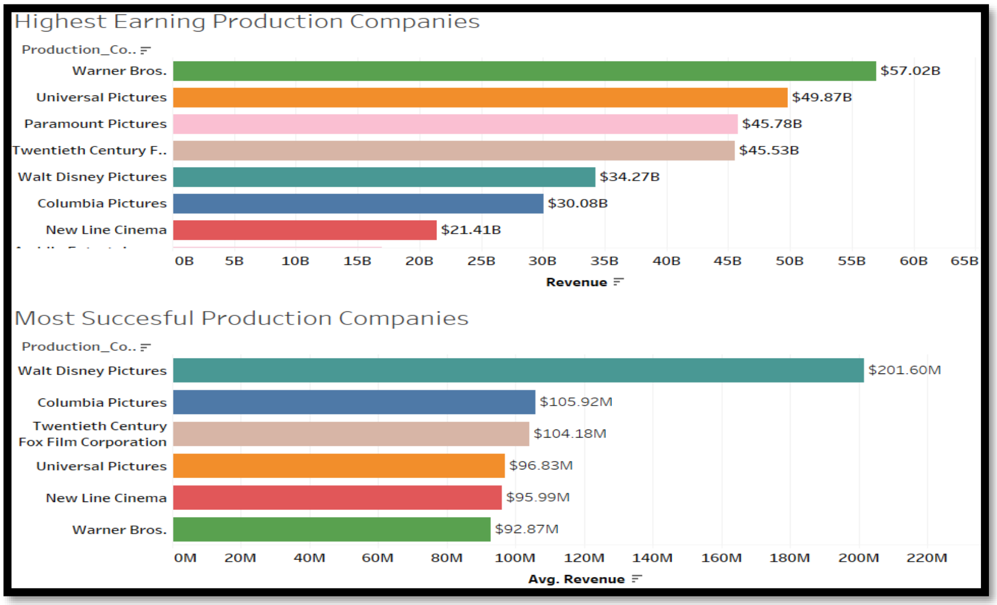
- The US is the most expected movie production destination because our dataset consists mostly of English films. Europe is also ubiquitous, with the UK, France, Germany, and Italy among the top 5.

Franchise Movies



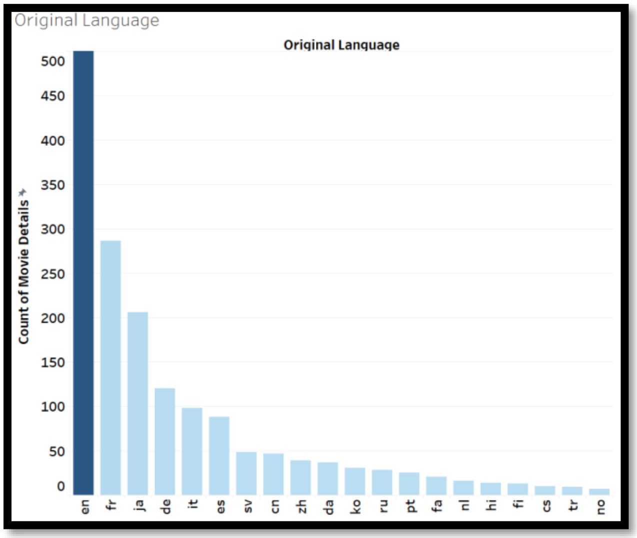
- The Harry Potter Series is the most profitable film franchise, ranking in more than $7.707 billion from 8 films. The Star war Movies are coming in a close second, too, with $7.403 billion from 8 movies. James Bond is sixth, but the franchise has a slightly higher gross average than the others on the list.
- While the Avatar Series currently consists of only one film, it is the most profitable franchise of all time, with the single film raking in nearly $3 billion. For at least five movies, the Harry Potter series is the most successful franchise.

Production Companies



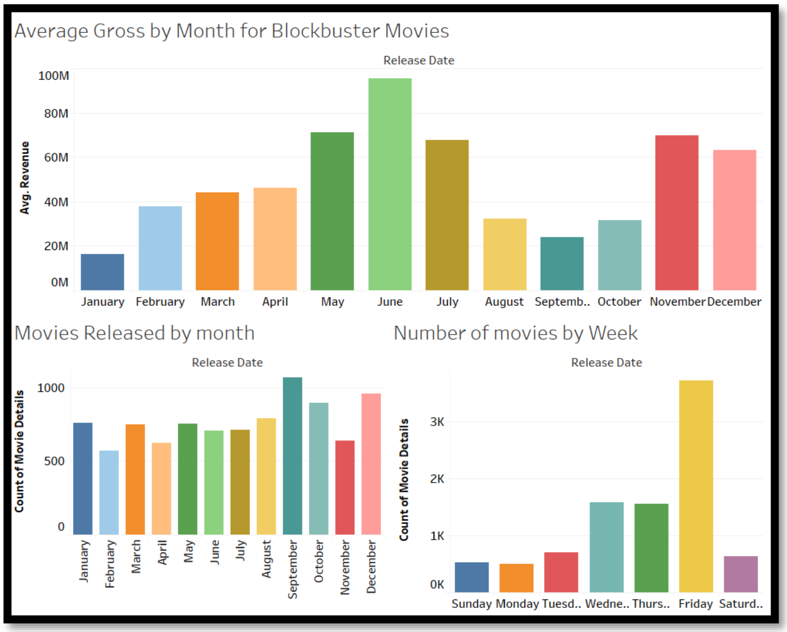
- Warner Bros is the highest-earning production company of all time, earning a staggering 63.5 billion dollars from close. Universal Pictures and Paramount Pictures are the second and the third highest-earning companies with 55 billion dollars and 48 billion dollars.
- Walt Disney Pictures has produced the most successful movies, on average.

Original Languages



- Our dataset contains over 93 languages. English language films form the vast majority, as we had expected. Both French and Japanese films come in very distant seconds and thirds.

Movie Release Dates



- The months of May and June have the highest gross average of the top-grossing films. The hit movies are usually released in the summer when the children are out of school and the parents are on holiday. Thus, the audience is more likely to spend their discretionary income on entertainment.

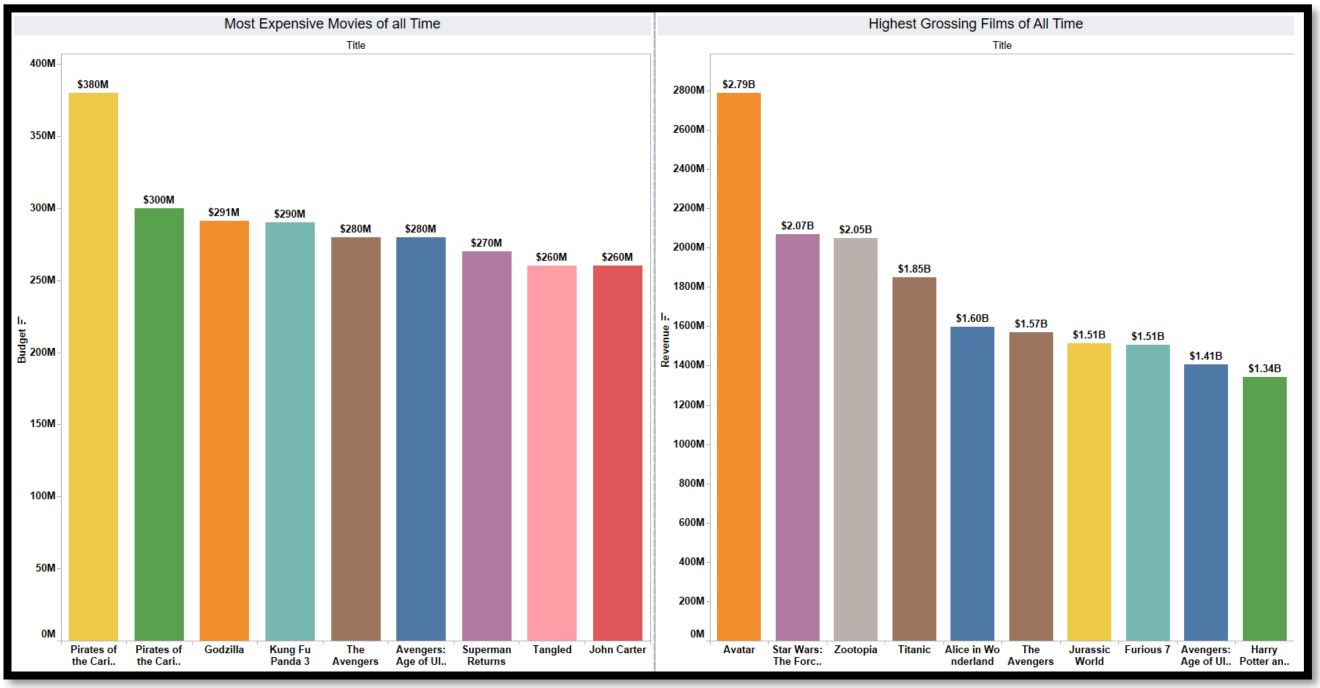
- Friday is probably the most famous day for film releases. That’s understandable given the fact that it typically denotes the weekend started. Sunday and Monday are the least-popular days and can be due to the same cause.

Genres

Animation
Music
Science Fiction
Family
Action
Romance
Fantasy
Foreign
Thriller
Drama
War
TV Movie
History
Crime
Adventure
Horror
Comedy
Documentary
Mystery
Western

- Drama is the most prevalent genre, with almost half of the films describing themselves as a drama. Comedy comes in at a distant second, with 25% of the movie having sufficient comedy doses.
- Action, Horror, Crime, Mystery, Science Fiction, Animation, and Fantasy are other major genres included in the top 10.

Budget & Revenue



- Budget is always a crucial feature for film performance. Two Caribbean Pirates films hold the top slots in this list with a massive $300 million budget.
- The second most critical numerical number associated with a film is the revenue. Avatar is the highest-grossing movie of all time with a revenue of $2.79 billion.

MOVIE RECOMMENDER SYSTEM

Simple Recommender System

Simple recommender offers generic recommendations based on movie popularity and genre to every user. This recommender's fundamental idea is that more successful and critically acclaimed movies should have a higher chance of being enjoyed by the average viewer. This model does not offer user-based, custom recommendations.

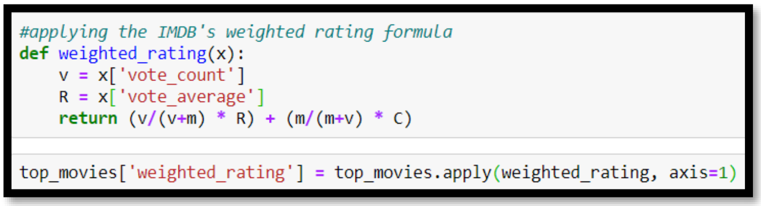
We have used the TMDB Ratings and IMDB's *weighted rating* formula to develop our Top Movies Chart.

$$\boldsymbol{Weighted\ Rating(WR)} = \left(\frac{\boldsymbol{v}}{\boldsymbol{v+m}}.\boldsymbol{R}\right) + \left(\frac{\boldsymbol{m}}{\boldsymbol{v+m}}.\boldsymbol{C}\right)$$

where,

- ***v*** : no. of votes for the movie
- ***m*** : minimum votes required to be listed in the chart
- ***R*** : average rating of the movie
- ***C*** :mean vote across the whole report

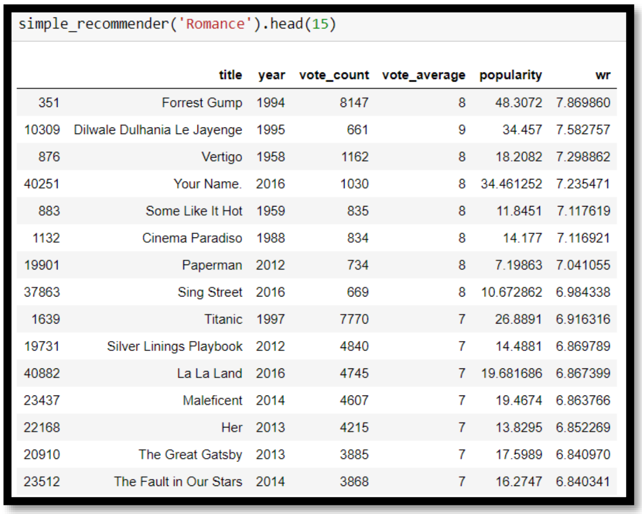
The next step was to decide a suitable value for 'm', the minimum votes needed to appear in the chart. We have used the 95th percentile as our limit. In other words, for a movie to be included in the charts, it must have more votes in the list than at least 95th percent of the movies.



```python
#applying the IMDB's weighted rating formula
def weighted_rating(x):
    v = x['vote_count']
    R = x['vote_average']
    return (v/(v+m) * R) + (m/(m+v) * C)
```

```python
top_movies['weighted_rating'] = top_movies.apply(weighted_rating, axis=1)
```

Once all the parameters were obtained, we build a function that gave us movie recommendation with respect to the weighted average and genre as the feature.



```python
simple_recommender('Romance').head(15)
```

	title	year	vote_count	vote_average	popularity	wr
351	Forrest Gump	1994	8147	8	48.3072	7.869860
10309	Dilwale Dulhania Le Jayenge	1995	661	9	34.457	7.582757
876	Vertigo	1958	1162	8	18.2082	7.298862
40251	Your Name.	2016	1030	8	34.461252	7.235471
883	Some Like It Hot	1959	835	8	11.8451	7.117619
1132	Cinema Paradiso	1988	834	8	14.177	7.116921
19901	Paperman	2012	734	8	7.19863	7.041055
37863	Sing Street	2016	669	8	10.672862	6.984338
1639	Titanic	1997	7770	7	26.8891	6.916316
19731	Silver Linings Playbook	2012	4840	7	14.4881	6.869789
40882	La La Land	2016	4745	7	19.681686	6.867399
23437	Maleficent	2014	4607	7	19.4674	6.863766
22168	Her	2013	4215	7	13.8295	6.852269
20910	The Great Gatsby	2013	3885	7	17.5989	6.840970
23512	The Fault in Our Stars	2014	3868	7	16.2747	6.840341

The top romance movie, according to our metrics, is Forrest Gump. It was followed by Bollywood's movie Dilwale Dulhania Le Jayenge.

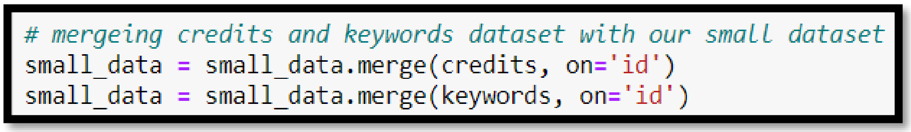
Content-Based Recommender System

In the Simple Recommender engine, we suffered from some significant limitations. It offers the same suggestion to everyone, no matter the user's taste. If our top 15 list were to be looked at by a person who loves romantic films (and hates action), they would probably not like any films.

To personalize our recommendations more, we have built an engine that takes in a movie that a user currently likes as input. It then analyzes the contents (storyline, genre, cast, director, etc.) to find out other movies with similar content. It ranks identical movies according to their similarity scores and recommends the most relevant movies to the user. Since we will be using content to build this engine, this is also known as **Content-Based Filtering.**

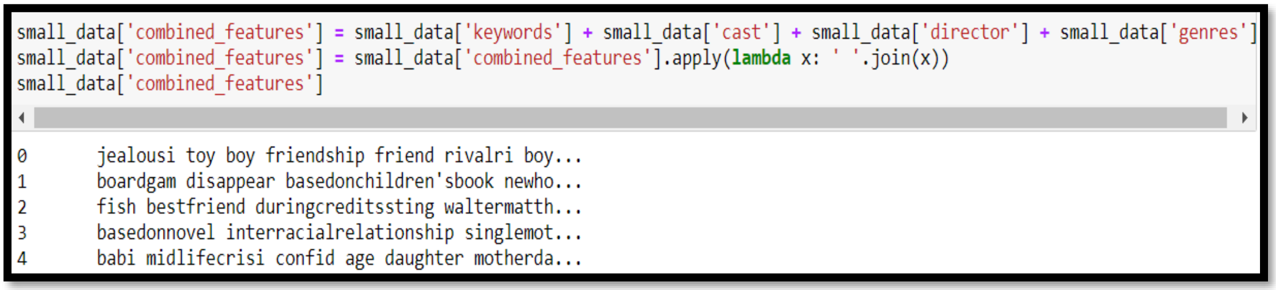
Content-Based Recommender System was implemented as follows:

Step 1: As mentioned earlier, for our personalized recommender engines, we will be using a smaller dataset. Since we must include the keyword, cast, and direction to this engine, we had to merge our dataset with **credits.csv** data, which contains the cast and direction details, and **keywords.csv,** which consist of keywords associated with the movies present in our dataset.



```
# mergeing credits and keywords dataset with our small dataset
small_data = small_data.merge(credits, on='id')
small_data = small_data.merge(keywords, on='id')
```

Step 2: Once all necessary features are present in a single dataset, we then combined our content features, i.e., keyword, genre, cast, and director, in a single column.

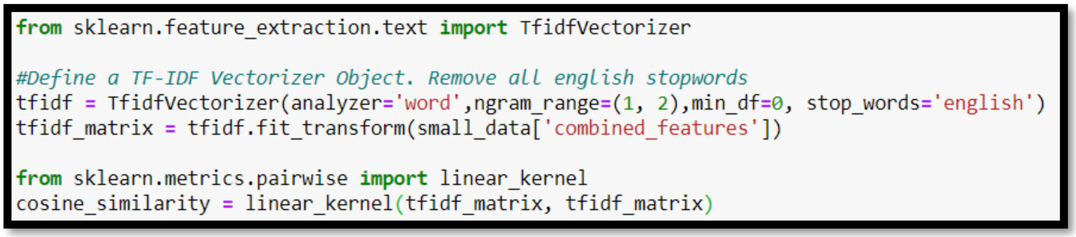


```
small_data['combined_features'] = small_data['keywords'] + small_data['cast'] + small_data['director'] + small_data['genres']
small_data['combined_features'] = small_data['combined_features'].apply(lambda x: ' '.join(x))
small_data['combined_features']
```

```
0      jealousi toy boy friendship friend rivalri boy...
1      boardgam disappear basedonchildren'sbook newho...
2      fish bestfriend duringcreditssting waltermatth...
3      basedonnovel interracialrelationship singlemot...
4      babi midlifecrisi confid age daughter motherda...
```

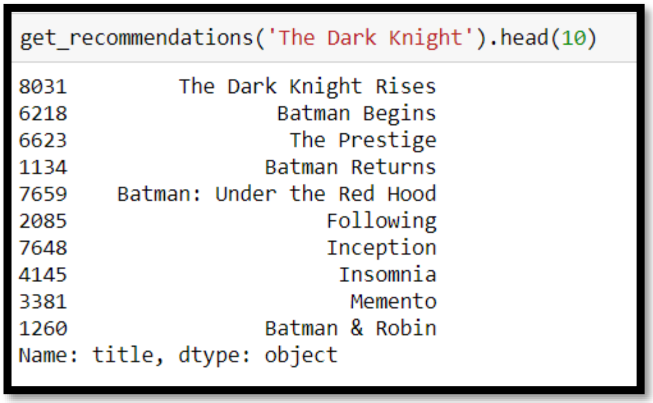
Step 3: Converted our combined features using **TFIDF (Term Frequency–Inverse Document Frequency) Vectorizer** and obtained a similar score of each movie using cosine similarity, which is mathematically defined as follows:

$$cosine(x, y) = \frac{x.y}{||x||.||y||}$$



```
from sklearn.feature_extraction.text import TfidfVectorizer

#Define a TF-IDF Vectorizer Object. Remove all english stopwords
tfidf = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words='english')
tfidf_matrix = tfidf.fit_transform(small_data['combined_features'])

from sklearn.metrics.pairwise import linear_kernel
cosine_similarity = linear_kernel(tfidf_matrix, tfidf_matrix)
```

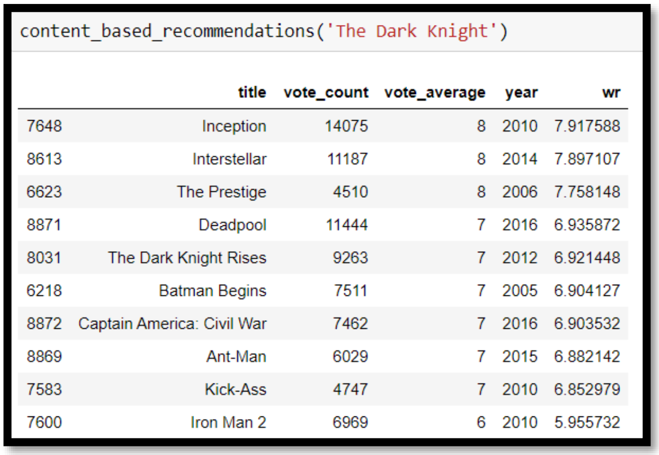
Step 4: Once we have obtained similarity scores for each movie, we build a function that recommends the most similar movies based on cosine similarity scores.



```
get_recommendations('The Dark Knight').head(10)

8031         The Dark Knight Rises
6218                 Batman Begins
6623                  The Prestige
1134                Batman Returns
7659    Batman: Under the Red Hood
2085                     Following
7648                     Inception
4145                      Insomnia
3381                       Memento
1260                Batman & Robin
Name: title, dtype: object
```

In our recommendation system, we see that, regardless of ratings and popularity, it recommends movies. Batman and Robin, indeed compared to The Dark Knight, have many similar characters, but it was a bad movie that should not be recommended to anyone.

To achieve that, we have integrated the weighted average function that we previously implemented in our simple recommender system to remove terrible movies and return movies that have a strong critical response.



```
content_based_recommendations('The Dark Knight')
```

	title	vote_count	vote_average	year	wr
7648	Inception	14075	8	2010	7.917588
8613	Interstellar	11187	8	2014	7.897107
6623	The Prestige	4510	8	2006	7.758148
8871	Deadpool	11444	7	2016	6.935872
8031	The Dark Knight Rises	9263	7	2012	6.921448
6218	Batman Begins	7511	7	2005	6.904127
8872	Captain America: Civil War	7462	7	2016	6.903532
8869	Ant-Man	6029	7	2015	6.882142
7583	Kick-Ass	4747	7	2010	6.852979
7600	Iron Man 2	6969	6	2010	5.955732

We can observe that movies recommended to us now have a good rating and strong critical response. Also, Batman and Robin's earlier recommendation is not present as it has a terrible vote average, i.e., 4.

Collaborative Filtering Recommender System

Our content-based engine suffers from some significant constraints. It is only able to recommend movies that are similar to a particular film. That is, it cannot identify the user's tastes and make suggestions across genres. The engine we designed isn't personal in that it doesn't absorb a user's personal preferences and biases. Anyone who queries our engine for recommendations based on a movie will provide the same recommendations for that movie.

In this recommender system, We will therefore use a technique called Collaborative Filtering to make more personalized suggestions to Movie Lovers. Collaborative filtering is based on the concept that it is possible to consider users similar to me to predict which movies I would like which those users have already watched, but I have not.

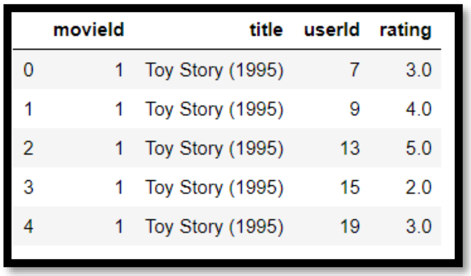
We will be implementing two different technique for Collaborative Filtering Recommender Engine:

1) Using Pearson's Correlation

Pearson's Coefficient of correlation is used to get the statistical relationship between two variables. In the collaborative filtering recommender system, we will use Pearson's correlation to get us the correlation coefficient between similar movies, which will help us reach more personalized movie recommendations.

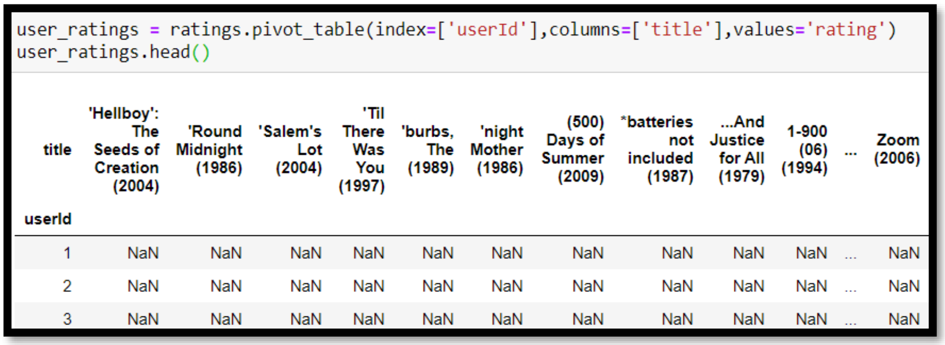
Steps to be followed for the building of this recommender engine:

Step 1: For collaborative filtering, we will be using the **ratings.csv** data file, which contains around 100,000 ratings on 7,000 movies from approx. 700 users. We merged the rating.csv with our small dataset to get all the necessary features required for this recommender engine.



	movieId	title	userId	rating
0	1	Toy Story (1995)	7	3.0
1	1	Toy Story (1995)	9	4.0
2	1	Toy Story (1995)	13	5.0
3	1	Toy Story (1995)	15	2.0
4	1	Toy Story (1995)	19	3.0

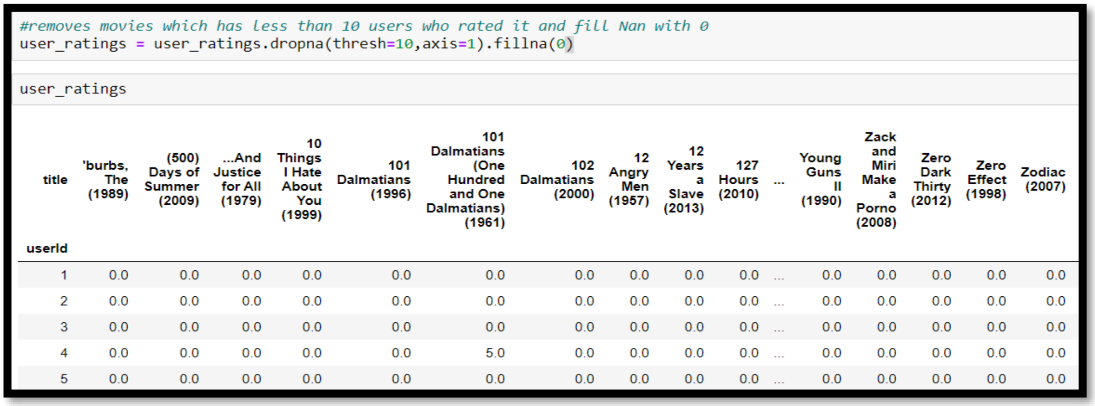
Step 2: Once the data frame was obtained, we pivot the table to get **userId** as our table index, movie **titles** as our columns, and **user_ratings** as our value.



```
user_ratings = ratings.pivot_table(index=['userId'],columns=['title'],values='rating')
user_ratings.head()
```

title	'Hellboy': The Seeds of Creation (2004)	'Round Midnight (1986)	'Salem's Lot (2004)	'Til There Was You (1997)	'burbs, The (1989)	'night Mother (1986)	(500) Days of Summer (2009)	*batteries not included (1987)	...And Justice for All (1979)	1-900 (06) (1994)	...	Zoom (2006)
userId												
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN

Step 3: Remove movies which have less than ten users who rated it and then fill the remaining NaN values with 0

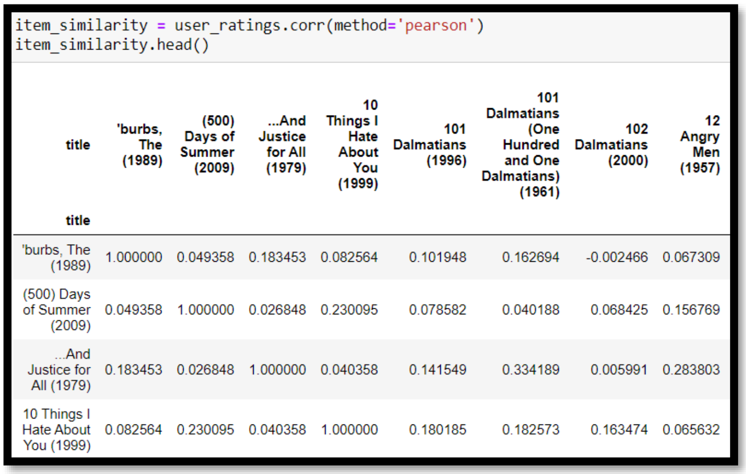


```
#removes movies which has less than 10 users who rated it and fill Nan with 0
user_ratings = user_ratings.dropna(thresh=10,axis=1).fillna(0)
```

```
user_ratings
```

title	'burbs, The (1989)	(500) Days of Summer (2009)	...And Justice for All (1979)	10 Things I Hate About You (1999)	101 Dalmatians (1996)	101 Dalmatians (One Hundred and One Dalmatians) (1961)	102 Dalmatians (2000)	12 Angry Men (1957)	12 Years a Slave (2013)	127 Hours (2010)	...	Young Guns II (1990)	Zack and Miri Make a Porno (2008)	Zero Dark Thirty (2012)	Zero Effect (1998)	Zodiac (2007)
userId																
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0

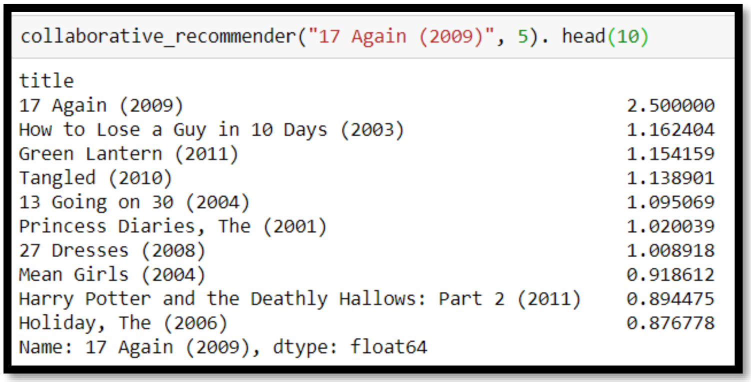
Step 4: Building a similarity matrix that gives us a similarty score between two movies using the Pearson correlation method.



```
item_similarity = user_ratings.corr(method='pearson')
item_similarity.head()
```

title	'burbs, The (1989)	(500) Days of Summer (2009)	...And Justice for All (1979)	10 Things I Hate About You (1999)	101 Dalmatians (1996)	101 Dalmatians (One Hundred and One Dalmatians) (1961)	102 Dalmatians (2000)	12 Angry Men (1957)
title								
'burbs, The (1989)	1.000000	0.049358	0.183453	0.082564	0.101948	0.162694	-0.002466	0.067309
(500) Days of Summer (2009)	0.049358	1.000000	0.026848	0.230095	0.078582	0.040188	0.068425	0.156769
...And Justice for All (1979)	0.183453	0.026848	1.000000	0.040358	0.141549	0.334189	0.005991	0.283803
10 Things I Hate About You (1999)	0.082564	0.230095	0.040358	1.000000	0.180185	0.182573	0.163474	0.065632

Step 5: Implementation of function which will give us movie recommendation based on collaborative filtering technique.



```
collaborative_recommender("17 Again (2009)", 5). head(10)
```

```
title
17 Again (2009)                                     2.500000
How to Lose a Guy in 10 Days (2003)                 1.162404
Green Lantern (2011)                                1.154159
Tangled (2010)                                      1.138901
13 Going on 30 (2004)                               1.095069
Princess Diaries, The (2001)                        1.020039
27 Dresses (2008)                                   1.008918
Mean Girls (2004)                                   0.918612
Harry Potter and the Deathly Hallows: Part 2 (2011) 0.894475
Holiday, The (2006)                                 0.876778
Name: 17 Again (2009), dtype: float64
```

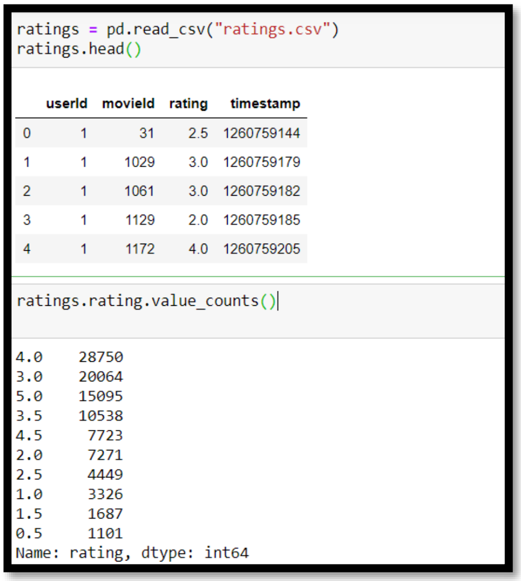
We can see that our **collaborative_recommender** function takes movie *title* and *rating* as its input and displays those movies with similar ratings concerning similar scores we got in earlier steps.

2)Using Scikit Learn's Surprise Library

The **surprise** is a Python **scikit** for building and analyzing recommender systems that deal with explicit rating data.

Steps to be followed for the building of this recommender engine:

Step 1: The first step is to pre-process our data.



```
ratings = pd.read_csv("ratings.csv")
ratings.head()
```

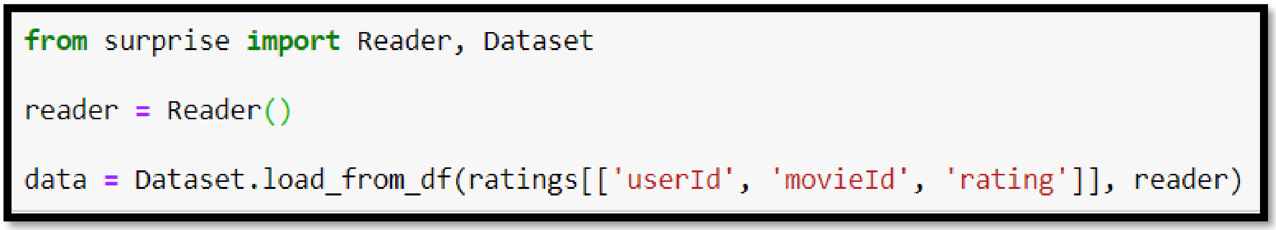
	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205

```
ratings.rating.value_counts()
```

```
4.0    28750
3.0    20064
5.0    15095
3.5    10538
4.5     7723
2.0     7271
2.5     4449
1.0     3326
1.5     1687
0.5     1101
Name: rating, dtype: int64
```

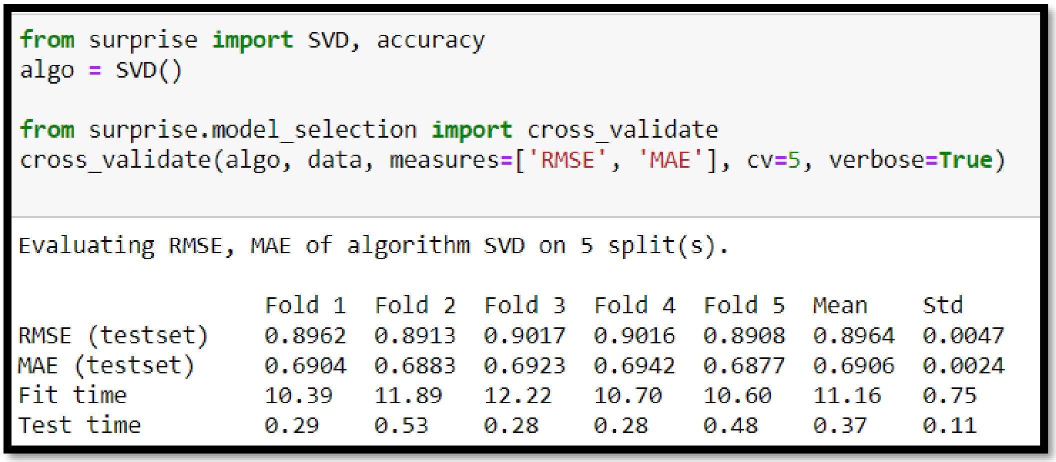
From the above snippet, we can see that we have four columns *userId, movieId, ratings, timestamp*, and the value counts of rating. We observe that ratings of 4.0 have the highest value count, which means more users rated a movie 4.0.

Step 2: To load a dataset from the pandas data frame, we have used the **load_from_df()** method. We have also used a Reader object, but only the **rating_scale** the parameter must be specified the default rating_scale is (2,5).



```
from surprise import Reader, Dataset

reader = Reader()

data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)
```

Step 3: We will be using the famous **Singular Value Decomposition (SVD)** algorithm.
SVD is a Matrix Factorization technique that is usually more effective because it allows us to discover the latent features underlying the interactions between users and items.

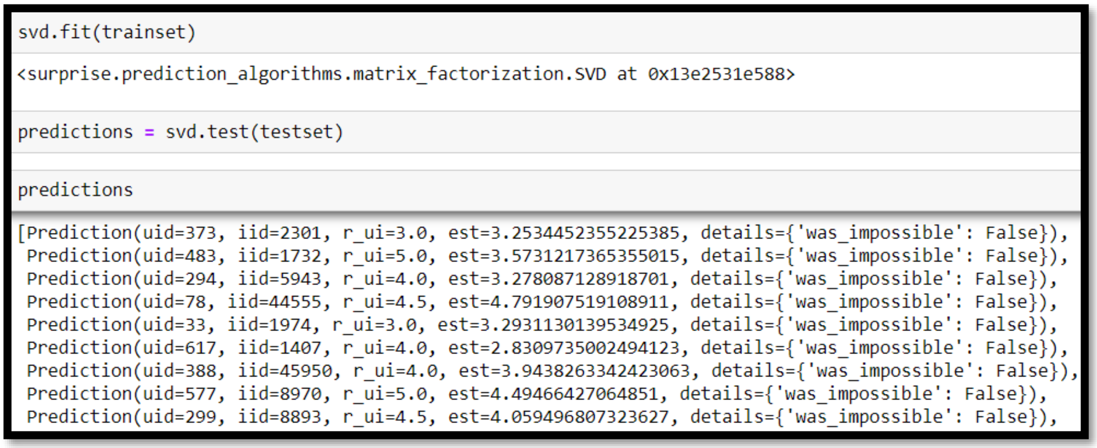


```
from surprise import SVD, accuracy
algo = SVD()

from surprise.model_selection import cross_validate
cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std     
RMSE (testset)    0.8962  0.8913  0.9017  0.9016  0.8908  0.8964  0.0047  
MAE (testset)     0.6904  0.6883  0.6923  0.6942  0.6877  0.6906  0.0024  
Fit time          10.39   11.89   12.22   10.70   10.60   11.16   0.75    
Test time         0.29    0.53    0.28    0.28    0.48    0.37    0.11    
```

For our case, we get a mean Root Mean Square Error of **0.8964,** which is more than good enough.

Step 4: Our final step is to fit the svd algorithm on trainset and later predict our values using the test set.



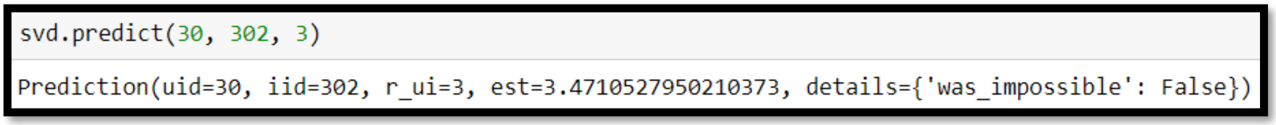
```
svd.fit(trainset)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x13e2531e588>
```

```
predictions = svd.test(testset)
```

```
predictions
```

```
[Prediction(uid=373, iid=2301, r_ui=3.0, est=3.2534452355225385, details={'was_impossible': False}),
 Prediction(uid=483, iid=1732, r_ui=5.0, est=3.5731217365355015, details={'was_impossible': False}),
 Prediction(uid=294, iid=5943, r_ui=4.0, est=3.278087128918701, details={'was_impossible': False}),
 Prediction(uid=78, iid=44555, r_ui=4.5, est=4.791907519108911, details={'was_impossible': False}),
 Prediction(uid=33, iid=1974, r_ui=3.0, est=3.2931130139534925, details={'was_impossible': False}),
 Prediction(uid=617, iid=1407, r_ui=4.0, est=2.8309735002494123, details={'was_impossible': False}),
 Prediction(uid=388, iid=45950, r_ui=4.0, est=3.9438263342423063, details={'was_impossible': False}),
 Prediction(uid=577, iid=8970, r_ui=5.0, est=4.49466427064851, details={'was_impossible': False}),
 Prediction(uid=299, iid=8893, r_ui=4.5, est=4.059496807323627, details={'was_impossible': False}),
```



```
svd.predict(30, 302, 3)
```

```
Prediction(uid=30, iid=302, r_ui=3, est=3.4710527950210373, details={'was_impossible': False})
```

We got an average prediction of **3.941** for the film with ID 302. One surprising aspect of this recommender method is that what the film is (or what it contains) doesn't matter. It operates solely based on an allocated film ID and attempts to predict ratings based on how the other users predicted the film.

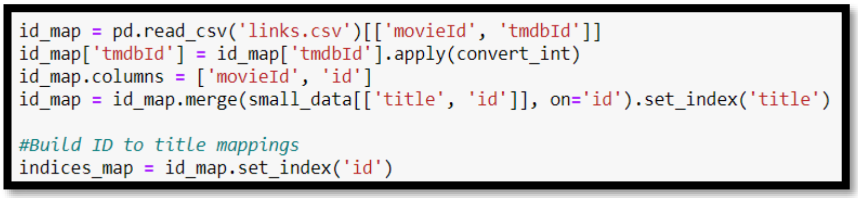
Hybrid Recommender

In the Hybrid recommender system, we have combined the techniques we have implemented in the content-based recommender engine and the collaborative filter recommender engine to provide personalized Similar Movie Recommendations to Users based on their taste.

The Hybrid Recommender system was implemented as follows:

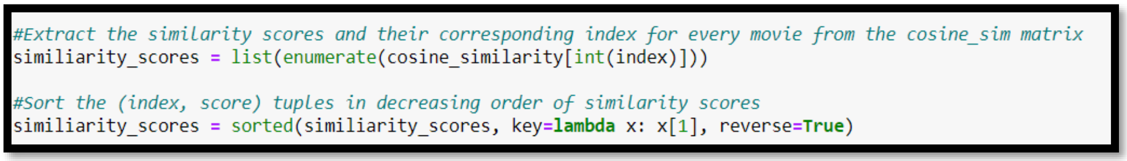
Step 1: Computing the similarity matrix that gives us a similarity score between two movies using Cosine similarity function as performed in the **content-based recommender system**.

Step 2: Computing the cosine similarity mapping matrix.



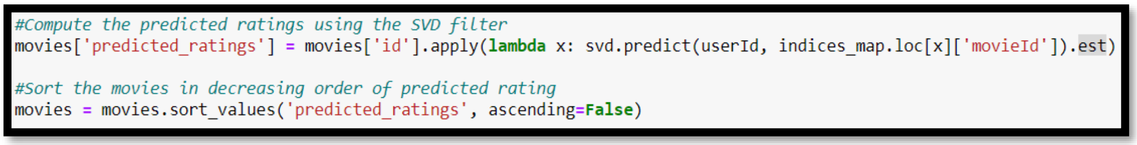
```
id_map = pd.read_csv('links.csv')[['movieId', 'tmdbId']]
id_map['tmdbId'] = id_map['tmdbId'].apply(convert_int)
id_map.columns = ['movieId', 'id']
id_map = id_map.merge(small_data[['title', 'id']], on='id').set_index('title')

#Build ID to title mappings
indices_map = id_map.set_index('id')
```

Step 3: Extract the similarity scores and their corresponding index for every movie from the **cosine_similarity** matrix.



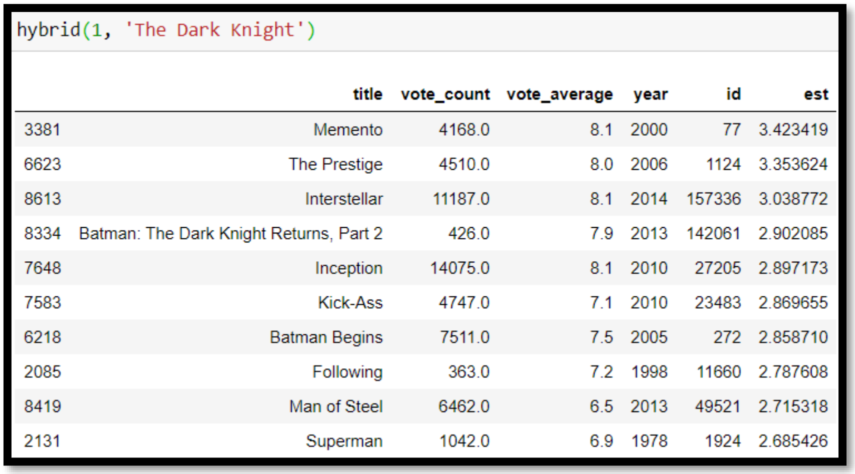
```
#Extract the similarity scores and their corresponding index for every movie from the cosine_sim matrix
similiarity_scores = list(enumerate(cosine_similarity[int(index)]))

#Sort the (index, score) tuples in decreasing order of similarity scores
similiarity_scores = sorted(similiarity_scores, key=lambda x: x[1], reverse=True)
```

Step 4: Compute the predicted ratings using the SVD filter as performed in a collaborative filtering recommender system



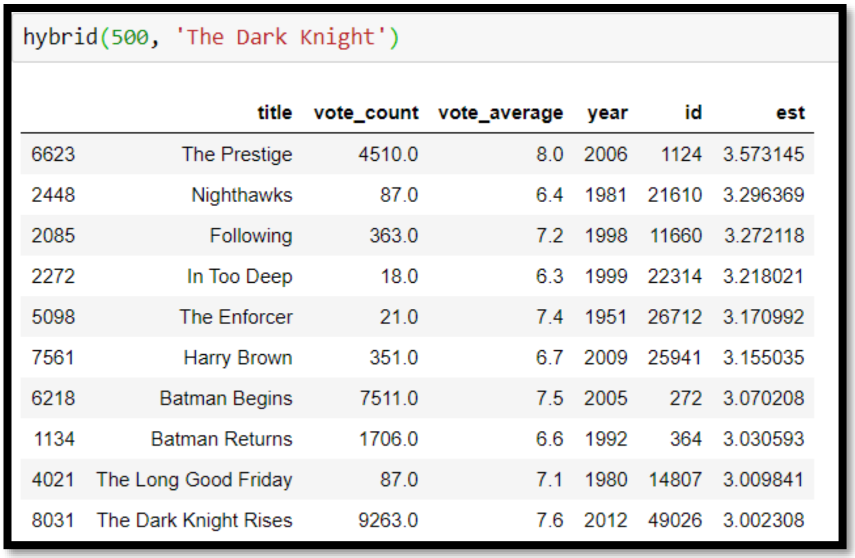
```
#Compute the predicted ratings using the SVD filter
movies['predicted_ratings'] = movies['id'].apply(lambda x: svd.predict(userId, indices_map.loc[x]['movieId']).est)

#Sort the movies in decreasing order of predicted rating
movies = movies.sort_values('predicted_ratings', ascending=False)
```

Step 5: Finally, recommending the movies by sorting them in descending order of their predicted rating.



```
hybrid(1, 'The Dark Knight')
```

	title	vote_count	vote_average	year	id	est
3381	Memento	4168.0	8.1	2000	77	3.423419
6623	The Prestige	4510.0	8.0	2006	1124	3.353624
8613	Interstellar	11187.0	8.1	2014	157336	3.038772
8334	Batman: The Dark Knight Returns, Part 2	426.0	7.9	2013	142061	2.902085
7648	Inception	14075.0	8.1	2010	27205	2.897173
7583	Kick-Ass	4747.0	7.1	2010	23483	2.869655
6218	Batman Begins	7511.0	7.5	2005	272	2.858710
2085	Following	363.0	7.2	1998	11660	2.787608
8419	Man of Steel	6462.0	6.5	2013	49521	2.715318
2131	Superman	1042.0	6.9	1978	1924	2.685426



```
hybrid(500, 'The Dark Knight')
```

	title	vote_count	vote_average	year	id	est
6623	The Prestige	4510.0	8.0	2006	1124	3.573145
2448	Nighthawks	87.0	6.4	1981	21610	3.296369
2085	Following	363.0	7.2	1998	11660	3.272118
2272	In Too Deep	18.0	6.3	1999	22314	3.218021
5098	The Enforcer	21.0	7.4	1951	26712	3.170992
7561	Harry Brown	351.0	6.7	2009	25941	3.155035
6218	Batman Begins	7511.0	7.5	2005	272	3.070208
1134	Batman Returns	1706.0	6.6	1992	364	3.030593
4021	The Long Good Friday	87.0	7.1	1980	14807	3.009841
8031	The Dark Knight Rises	9263.0	7.6	2012	49026	3.002308

From the above results, we can observe that the hybrid recommender engine gives us a very different recommendation compared to the same movie but for a different user. Hence, our recommendations are more personalized and targeted towards users with similar personal taste.

CONCLUSION

Thus, we have built four different recommendation systems based on different ideas and algorithms. They are as follows:

1)Simple Recommender: This system used TMDB Vote Count and Vote Averages to create Top Movies Charts, in general, and for a particular genre. The IMDB Weighted Ranking System was used to determine ratings for which final sorting was performed.

2)Content-Based Recommender: We have used contents (storyline, genre, cast, director, etc.) of the movie to find out other movies that have similar content. It ranks identical movies according to their similarity scores and recommends the most relevant movies to the user.

3)Collaborative Filtering Recommender: We have built two collaborative engines; one that uses the powerful Surprise Library to create a collaborative filter based on a decomposition of a single value. The obtained RMSE was less than one, and the engine received approximate ratings for a given user and film. Another engine that uses Pearson's correlation algorithm to recommend movies that was liked by users with similar taste.

4)Hybrid Recommender: We put together ideas from content and collaborative filtering to create an algorithm that gave recommendations of movies to a specific user based on the average ratings it had internally measured for that user.

REFERENCES

- “MovieLens”, https://grouplens.org/datasets/movielens/

- “How to build a Movie Recommender System” (Ramya Vidiyala), https://towardsdatascience.com/how-to-build-a-movie-recommendation-system-67e321339109

- “Surpr!se: A Python scikit for Recommender System”, http://surpriselib.com/

- “Cosine similarity: How does it measure the similarity, Math’s behind and usage in Python” (Varun), https://towardsdatascience.com/cosine-similarity-how-does-it-measure-the-similarity-maths-behind-and-usage-in-python-50ad30aad7db

- “Cosine Similarity – Understanding the math and how it works (with python codes)” (Selva Prabhakaran), https://www.machinelearningplus.com/nlp/cosine-similarity/

- “TF-IDF Vectorizer scikit-learn” (Mukesh Chaudhary), https://medium.com/@cmukesh8688/tf-idf-vectorizer-scikit-learn-dbc0244a911a

- “How to calculate Correlation between variables in Python” (Jason Brownlee), https://machinelearningmastery.com/how-to-use-correlation-to-understand-the-relationship-between-variables/