MOVIE RECOMMENDATION SYSTEM

By Priyadarshani Kamble

Whether you realize it or not, recommendations drive so many of our decisions on daily basis. Be it obvious recommendations such as suggestions of new restaurants from friends, or a certain model of camera discussed in a blog, to less direct recommendations such as Netflix promoting shows you are likely to enjoy, or Amazon proposing other purchases that go well with what you are buying.

Let's take Netflix as an example. Instead of having to browse through thousands of box sets and movie titles, Netflix presents you with a much narrower selection of items that you are likely to enjoy. This capability saves you time and delivers a better user experience. With this function, Netflix achieved lower cancellation rates, saving the company around a billion dollars a year.

What is a Recommendation Engine?

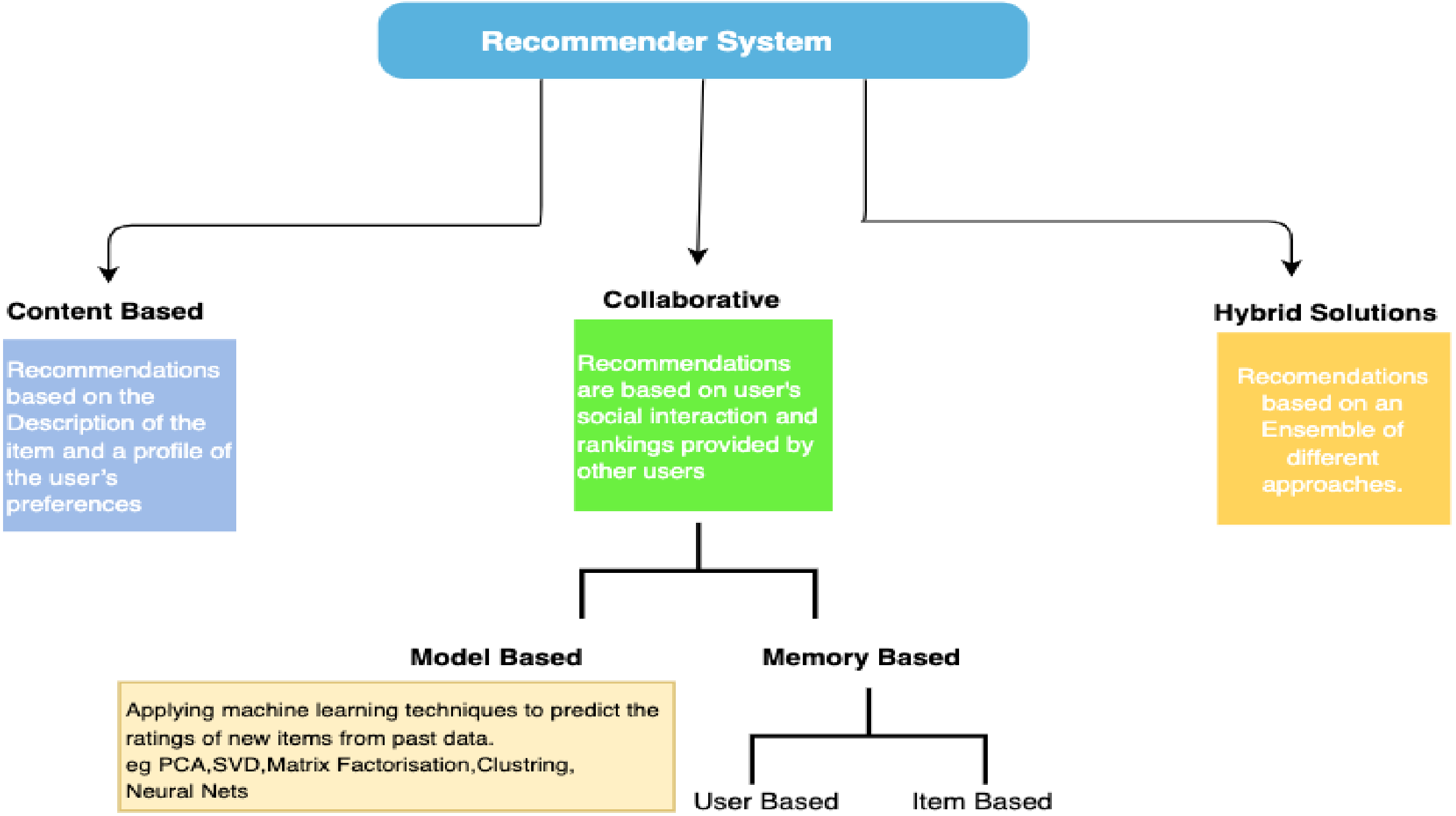
Simply put a **Recommendation System** is a filtration program whose prime goal is to predict the "rating" or "preference" of a user towards a domain-specific item.

A recommendation engine can be of three types- collaborative recommendation engine, content-based recommendation engine, and hybrid recommendation engine.

Collaborative Recommendation Engine: In collaborative filtering, a recommendation system recommends a user the products based on the preferences of the other users with similar tastes

Content Based Recommendation Engine: As the name suggests, a content-based recommendation engine recommends the relevant content to the users based on their preferred features of other content.

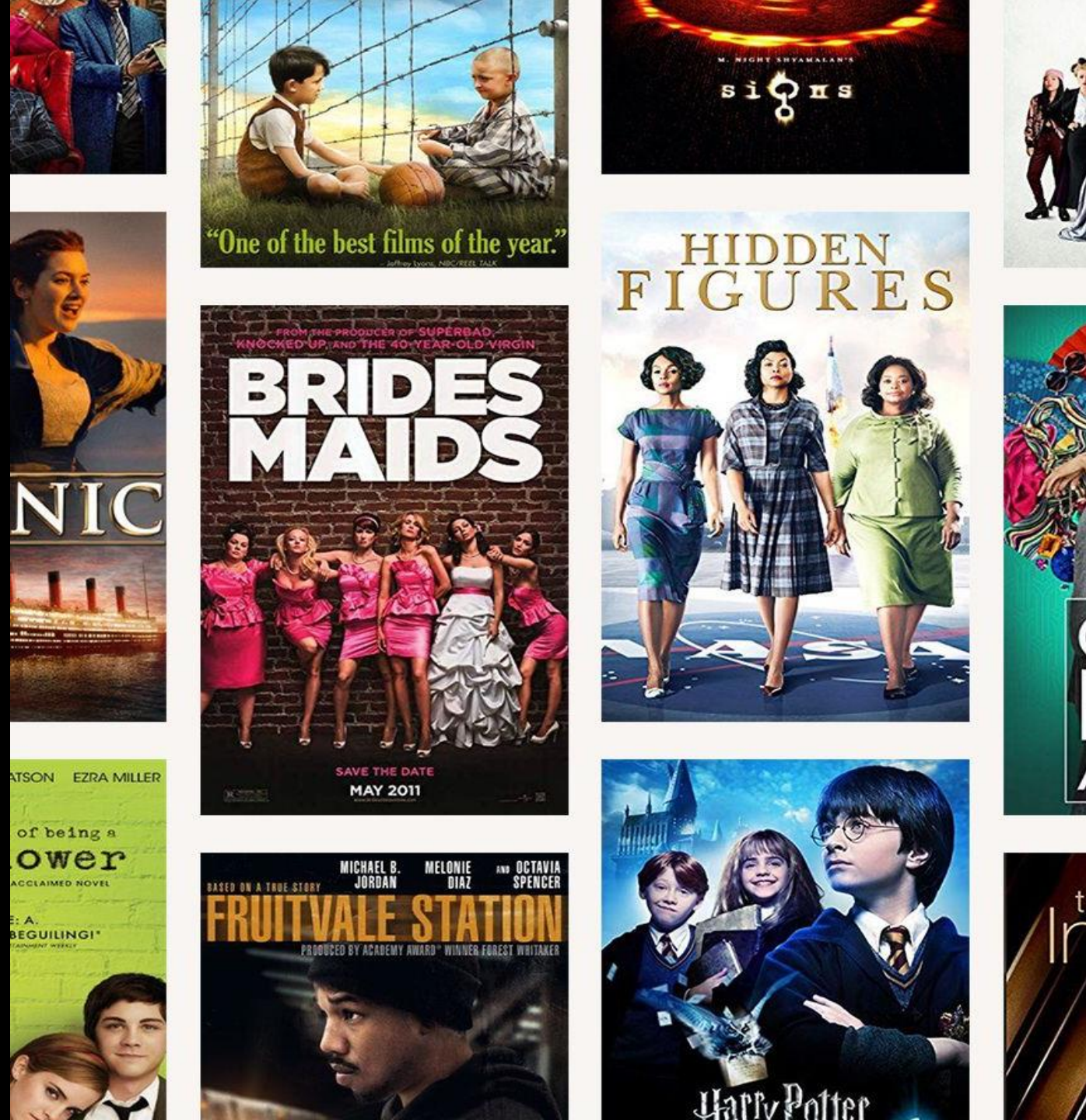
Hybrid recommendation engine: Collaborative filtering and content-based filtering both are used widely in the recommendation systems. A recommendation engine with both the collaborative filtering and content-based filtering is called a hybrid recommendation engine.


Recommender System
Content Based
Recommendations based on the Description of the item and a profile of the user's preferences
Collaborative
Recommendations are based on user's social interaction and rankings provided by other users
Hybrid Solutions
Recomendations based on an Ensemble of different approaches.
Model Based
Applying machine learning techniques to predict the ratings of new items from past data.
eg PCA,SVD,Matrix Factorisation,Clustring, Neural Nets
Memory Based
User Based
Item Based

For this Project, I will build Content based and Collaborative recommendation engine for a movie database.

I will be using below datasets from Kaggle.

Dataset Link:
https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset



Exploratory Data Analysis

Let's explore the dataset to find some insights about the data. There are close to 670 users and 7000 movies and 10 unique values of ratings.

The data show that Forrest Gump is the most popular film.

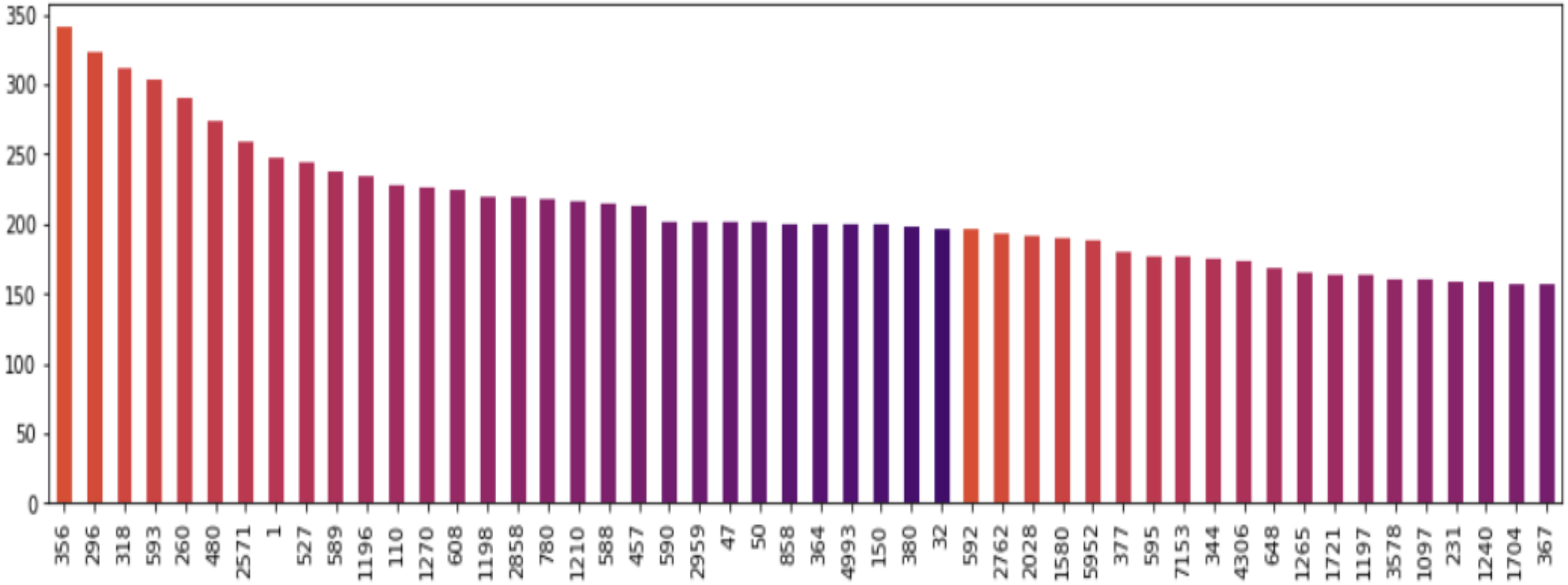
```
df.head()
```

	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

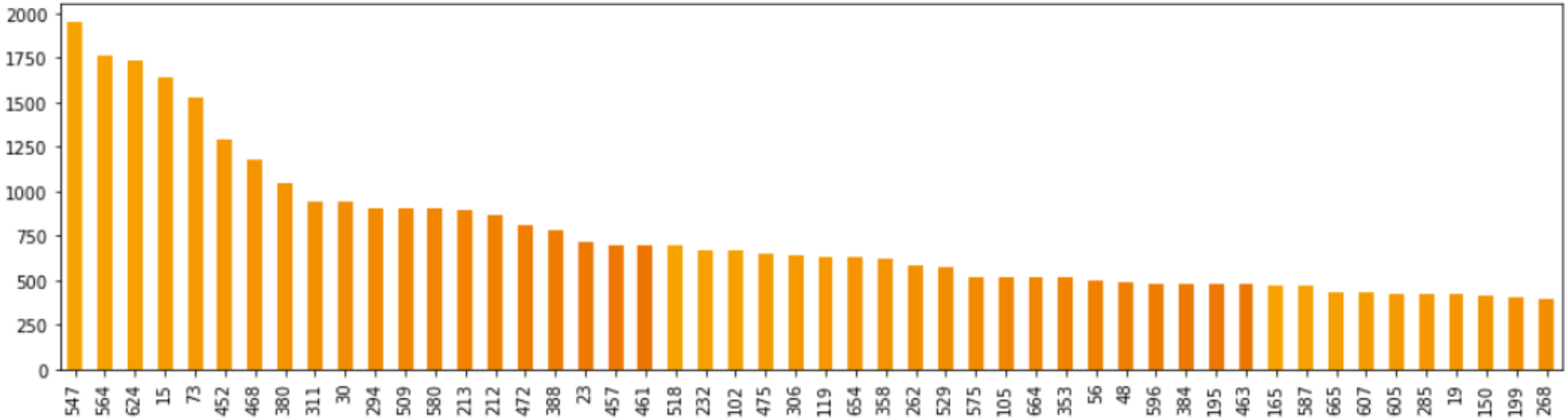
```
print(movie_popularity)
```

```
Forrest Gump (1994)                          341
Pulp Fiction (1994)                          324
Shawshank Redemption, The (1994)             311
Silence of the Lambs, The (1991)             304
Star Wars: Episode IV - A New Hope (1977)    291
Name: title, dtype: int64
```

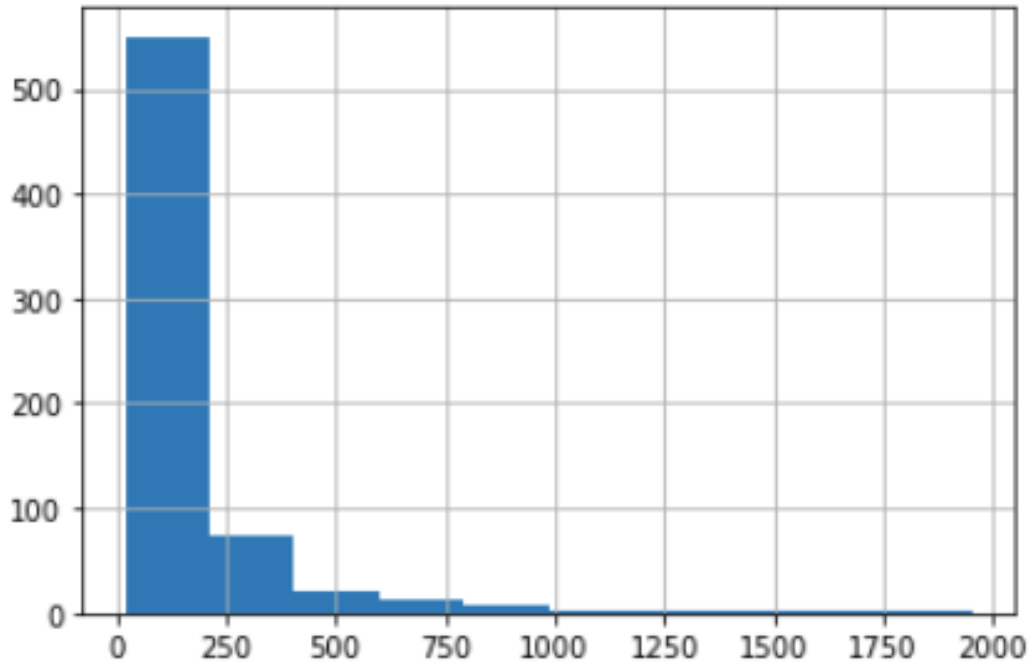
Here we can see that ratings are not evenly distributed among movies. The movie with id 356 [Forrest Gump] is the most rated movie, however it does not have more that 341 ratings.



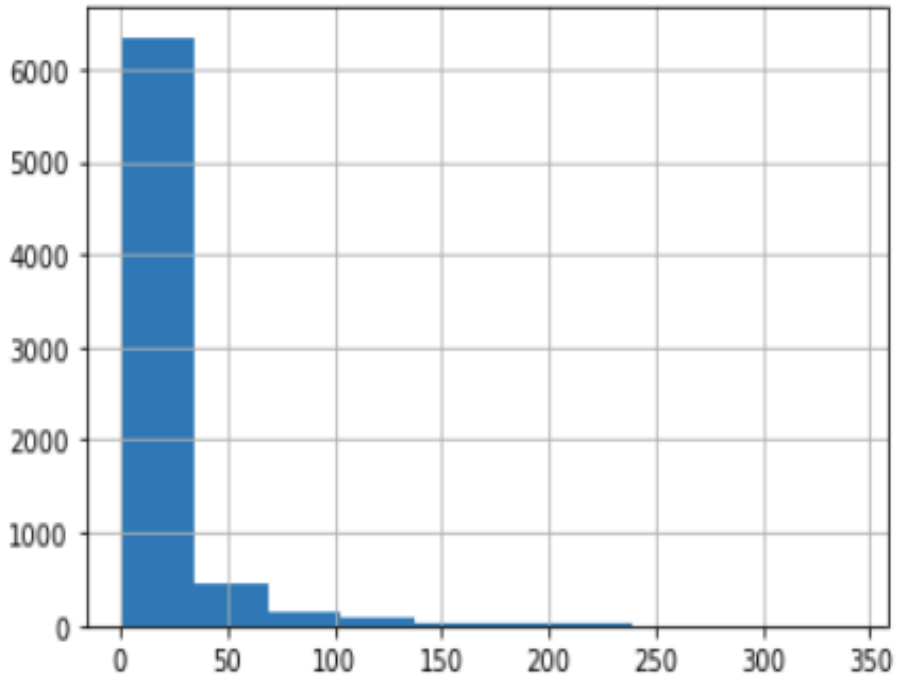
Users have provided 1953 ratings.



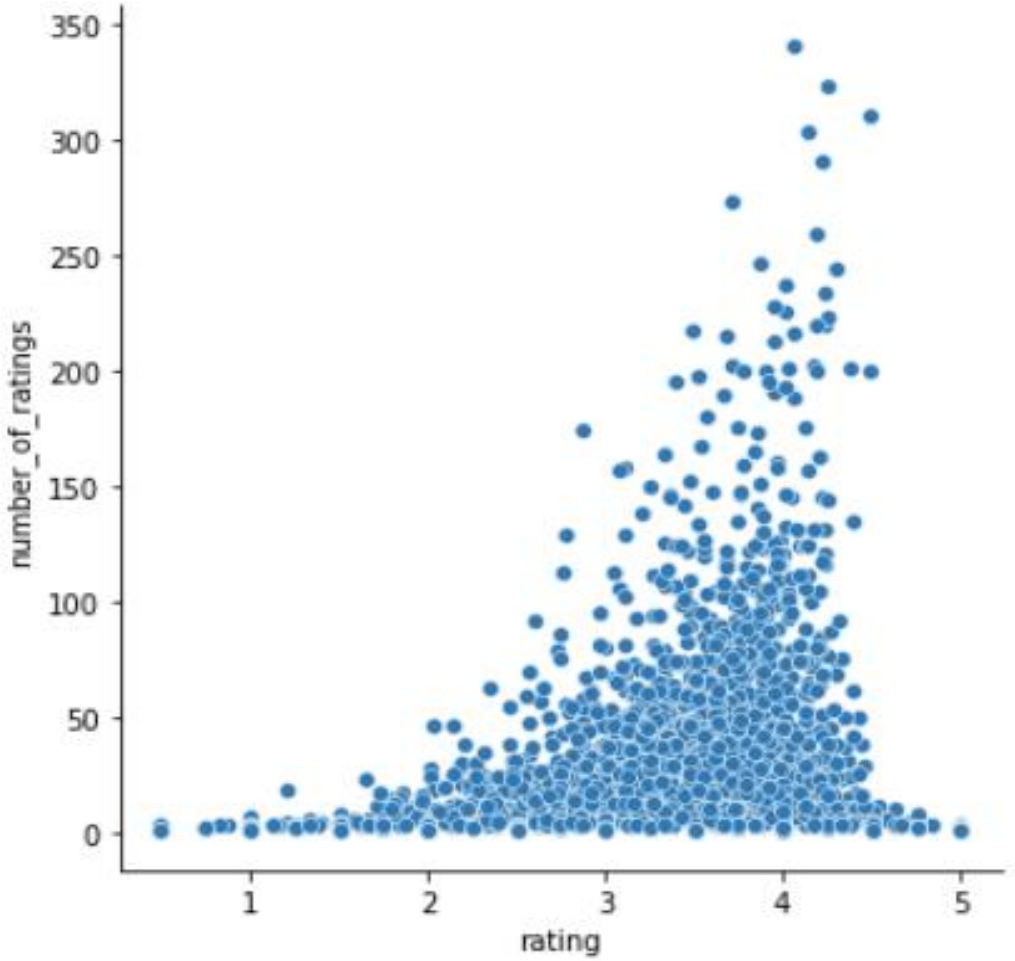
The histogram shows that most users (roughly 560 out of 671 –80%) have less than 250 ratings.



The histogram shows that most movies (roughly 6500 out of 7063 –90%) have less than 30 ratings.



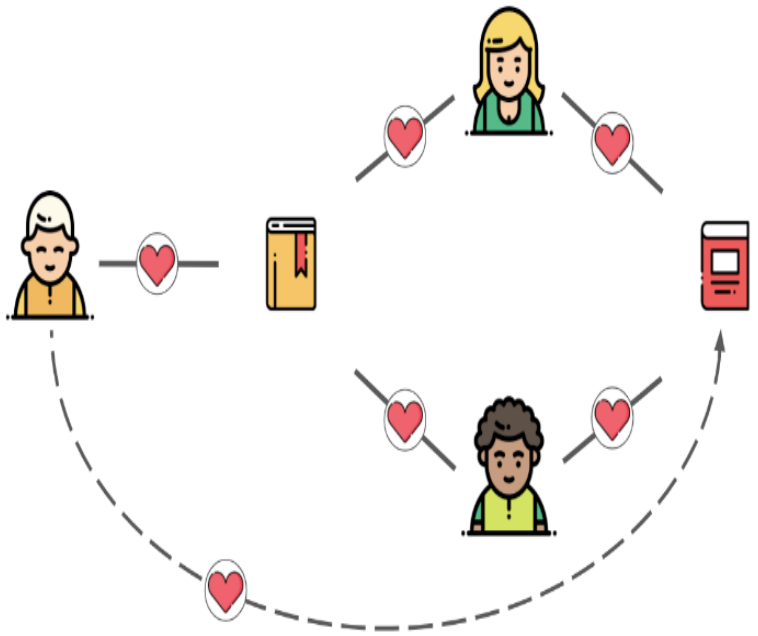
The plot shows many movies having 0 ratings. Less number of users have rated movies.
Most of the users have given rating between 3 to 4.
Average rating given to frequently watched films.



title	rating
Godfather, The (1972)	4.487500
Shawshank Redemption, The (1994)	4.487138
Maltese Falcon, The (1941)	4.387097
Godfather: Part II, The (1974)	4.385185
Usual Suspects, The (1995)	4.370647

COLLABORATIVE FILTERING :

Collaborative filtering is used by most recommendation systems to find similar patterns or information of the users, this technique can filter out items that users like based on the ratings or reactions by similar users.

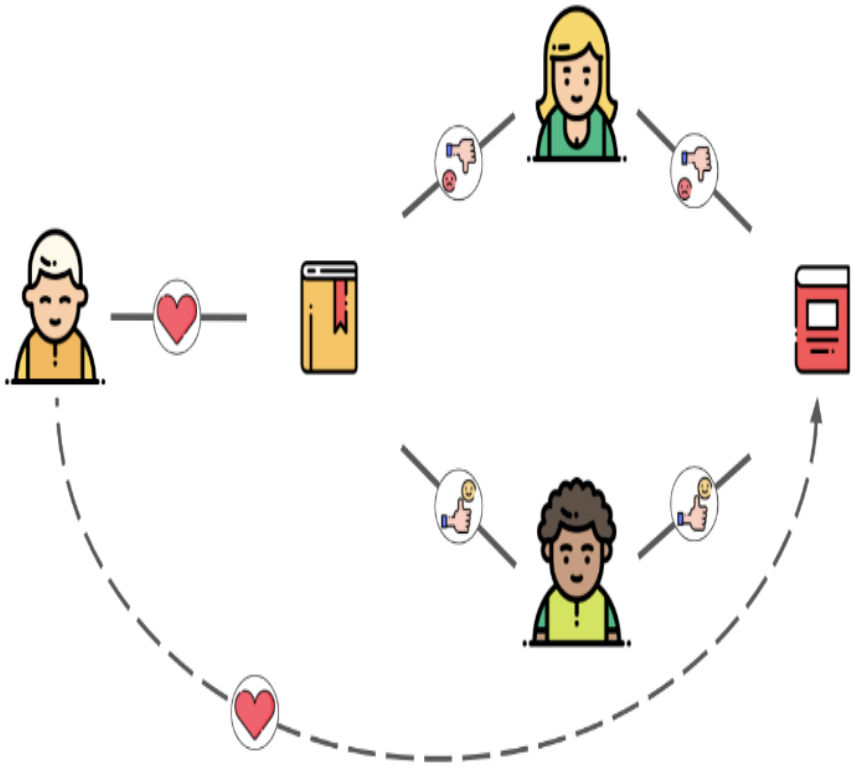


Memory Based Approach :

Memory-based collaborative filtering uses all the data in the database to generate a prediction

User-User-Based Collaborative Filtering

In this type of filtering, we find the users that have the most similar preferences to the user we are making recommendations for and based on that group's preferences, make suggestions. It works around the premise that person A has similar tastes to person B and C. and both person B and C also like a certain item, then it is likely that person A would also like that new item.



Item-Item-based Collaborative Filtering

It assumes, if Item A and B receive similar reviews, either positive or negative, then however other people feel about A, they should feel the same way about B.

we need to convert this data into matrix of item for machine to understand. As we see here , there are many NaN values.

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)	Zootopia (2016)	Zulu (1964)	Zulu (2013)	[REC] (2007)	eXistenZ (1999)	(2
userId																		
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	

5 rows × 7062 columns

This is expected as a user will rarely rate every movie. and its rare that every movie is rated by every user. This is called as data sparsity. We need to handle this missing values either by replacing them or dropping them . Dropping is not an option as the data this sparse, we will end up losing all the data.

We can not replace missing values with 0 as it can create issues with recommendation engines. As 0 imply that the user hated/greatly disliked the movie which we can not say for sure.

One alternative is to center each user's ratings around 0 by deducting the row average and then fill in the missing values with 0. This means the missing data is replaced with neutral scores.
We will fill in missing data with information that should not bias the data that we have.

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)	Zootopia (2016)	Zulu (1964)	Zulu (2013)	[REC] (2007)	eXistenZ (1999)	(2
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	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	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	0.0	
4	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	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	0.0	

Using corrwith function in Pandas, I calculated pair-wise correlations between rows and columns of a dataframe and used it for our prediction.

Making recommendation for movie 'Shawshank Redemption, The (1994)' :
we will see which movies are similar to Shawshank Redemption, The (1994) movie based on rating.

title	Corr	number_of_ratings
Shawshank Redemption, The (1994)	1.000000	311
Usual Suspects, The (1995)	0.278356	201
Forrest Gump (1994)	0.258404	341
Good Will Hunting (1997)	0.244178	157
Braveheart (1995)	0.235068	228
Beautiful Mind, A (2001)	0.223111	114
Pulp Fiction (1994)	0.216315	324
Schindler's List (1993)	0.214576	244
Green Mile, The (1999)	0.214251	103
Swingers (1996)	0.205387	42

Model Based Filtering :

Model-based collaborative filtering uses the data in the database to create a model that can then be used to generate predictions.

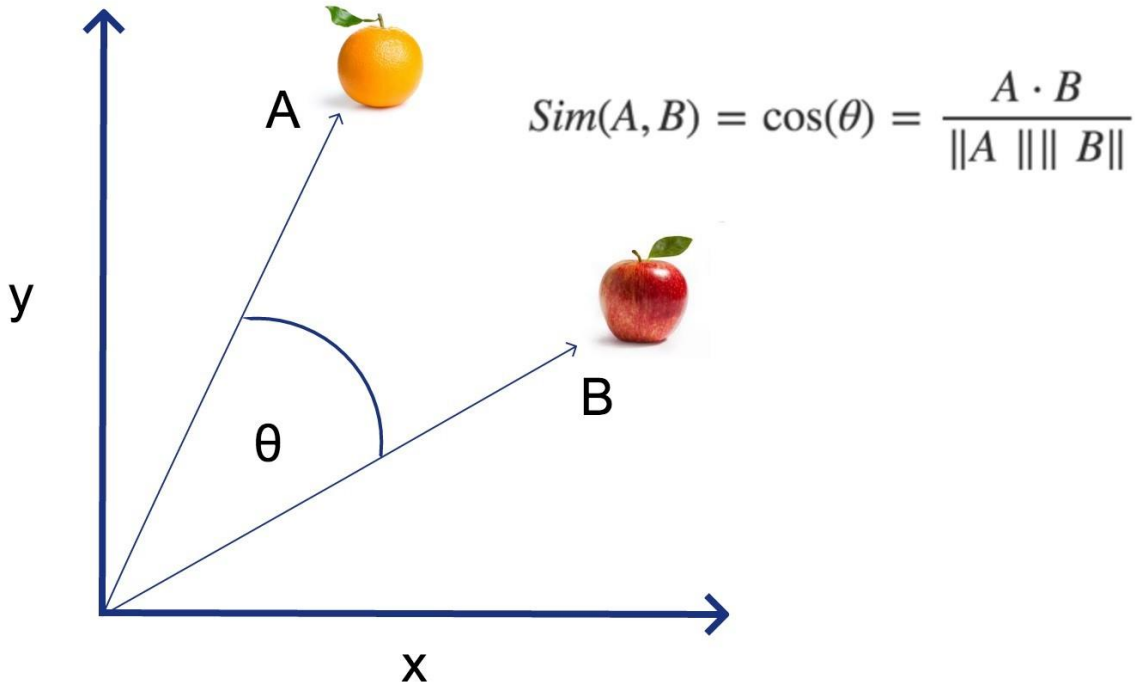
Collaborative filtering technique represents recommender system as a regression model, where the output is a numeric rating value. So, we can apply regression evaluation metrics to our recommendation system.

I used 2 collaborative-based filtering algorithms — K nearest neighbor and Singular value decomposition.

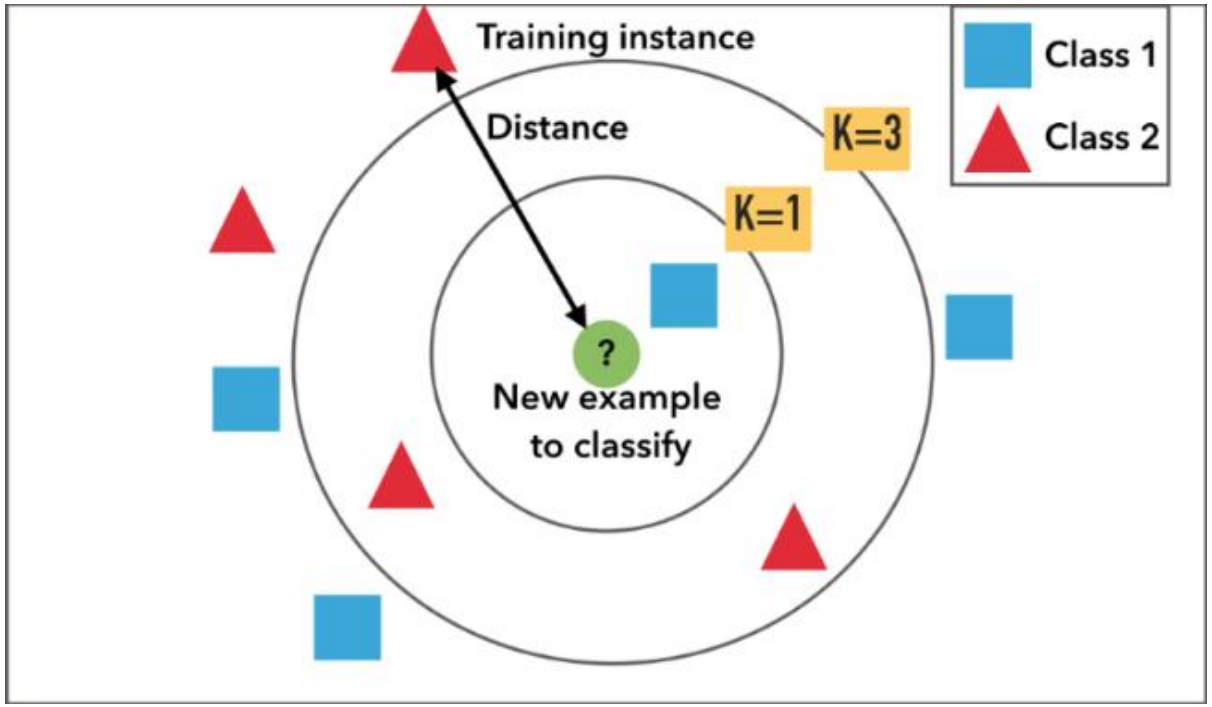
K Nearest Neighbor (KNN) :

k nearest neighbour algorithm is used to represent movies. The distance among points are calculated based on **cosine similarity** — which is determined by the angle between two vectors (as shown in the diagram). Cosine similarity is preferred instead of Euclidean distance, because it suffers less when the dataset is high in dimensionality.

Cosine Similarity



$$Sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$



Matrix sparsity

A common challenge with real-world ratings data is that most users will not have rated most items, and most items will only have been rated by a small number of users. This results in a very empty or sparse DataFrame.
I found that our DataFrame is over 97% empty. This means that less than 3% of the DataFrame includes any data. This is actually a common concern in real-world rating data as the number of users and items are generally quite high and the number of reviews are quite low.

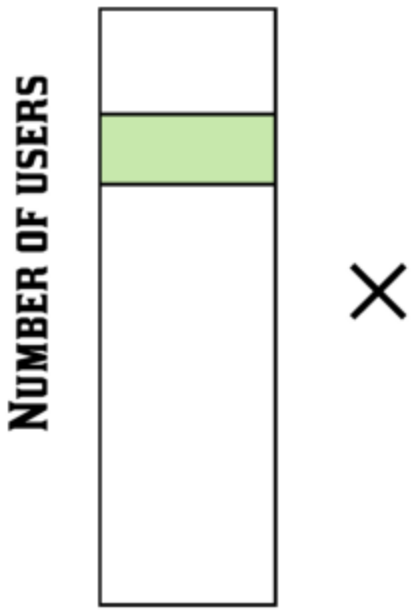
This can create problems if we were to use KNN with sparse data because KNN requires you to find the K nearest users that have rated the item and if there are not many neighbors therefore, we would have to return an average of all reviews because there is no other data. This will not actually take the similarities into account.

We will leverage the power of matrix factorization to deal with this sparsity. Matrix factorization is when we decompose the user-rating matrix into the product of two lower dimensionality matrices.

Matrix Factorization — Singular Value Decomposition

Singular Value Decomposition is a matrix factorization technique that decomposes the matrix into the product of lower dimensionality matrices, and then extracts the latent features from highest importance to lowest. Instead of iterating through individual ratings like KNN, it views the rating matrix as a whole. Therefore, it has less computation cost compared to KNN but also makes it less interpretable. SVD extract the latent features (which is not an actual features contained in the dataset, but what the algorithm magically discovered as valuable hidden features) to form the factorized matrices U and V transposed and placed them in a descending feature importance order. It then fills in the blank ratings by taking the product of U and V transposed in a weighted approach based on feature importance. These latent feature parameters are learned iteratively through minimizing the error.

Matrix Factorization



Predict

Items

Users

5	?	?	3
4	?	?	2
?	1	3	1

item latent vector

user latent vector

$$u_i^T v_j = \hat{r}_{ij}$$

predicted ratings

The surprise library allows us to implement both algorithms in just several lines of code.
The cross_validate function from surprise library executes cross validation automatically.
The result shows the comparison between KNN and SVD.

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

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9140  0.9280  0.9169  0.9231  0.9122  0.9188  0.0059
MAE (testset)     0.7037  0.7083  0.7032  0.7066  0.7005  0.7045  0.0027
Fit time          9.87    10.01   9.45    9.17    11.44   9.99    0.79
Test time         6.00    4.95    5.14    6.27    5.17    5.51    0.53
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8873  0.8927  0.9007  0.8978  0.8908  0.8938  0.0048
MAE (testset)     0.6834  0.6861  0.6947  0.6917  0.6866  0.6885  0.0041
Fit time          4.28    4.41    3.93    3.79    3.64    4.01    0.29
Test time         0.22    0.10    0.14    0.10    0.10    0.13    0.05
```

SVD has smaller RMSE, MAE values, hence performs better than KNN, and also takes significantly less time to compute.

Train-Test Split Evaluation

Here we will split the dataset into 80% for training and 20% for testing. Instead of iterating the model build 5 times as in cross validation, it will only train the model once and test it once.
The result shows that SVD has less error.

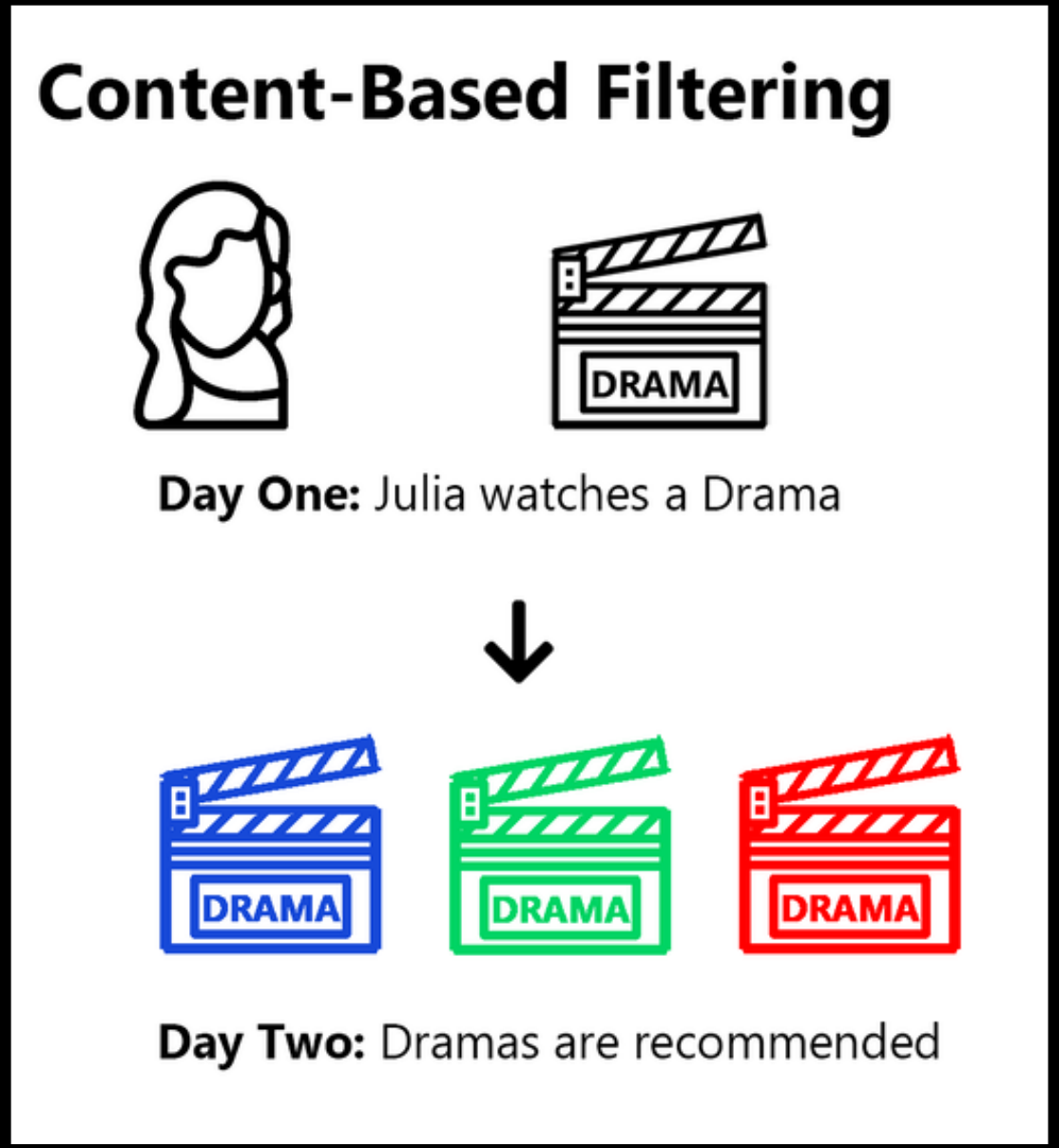
```
KNN RMSE - 0.9180918562232379
KNN MAE - 0.7054351557425212
KNN MSE - 0.8428926564634305
```

```
SVD RMSE - 0.8912746762213019
SVD MAE - 0.6875084776882803
SVD MSE - 0.7943705484733865
```

Using this model, we can now predict rating for a user for a movie.

For movie with ID 100, we get an estimated prediction of 2.43.

This recommender system doesn't care what the movie is (or what it contains). It works purely on the basis of an assigned movie ID and tries to predict ratings based on how the other users have predicted the movie.



CONTENT BASED RECOMMENDATION SYSTEM:

Content based filtering makes predictions of what the audience is likely to prefer based on the content properties, e.g., genres, language, video length.

Let's build Collaborative recommendation engine using movie database.
For this Project, I will build Content based recommendation engine using movie database. I will be using TMDB 5000 Movie Database from kaggle (https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata)

It contains 2 CSV files.
•tmdb_5000_credits.csv
•tmdb_5000_movies.csv

In this recommender system,
I will first find similarities based on only 'Overview' feature of the movie. As it consists of most of the description about the movie.

Next, I will use below features to find its similarity with other movies and predict similar movies.
cast - The name of lead and supporting actors.
crew - The name of Director, Editor, Composer, Writer etc
genre - The genre of the movie, Action, Comedy ,Thriller etc.
keywords - The keywords or tags related to the movie.

Let's use the 'overview' column first to recommend movies to the user. This column contains the plot of movies.

```
movies['overview'].head()
```

```
0    In the 22nd century, a paraplegic Marine is di...
1    Captain Barbossa, long believed to be dead, ha...
2    A cryptic message from Bond's past sends him o...
3    Following the death of District Attorney Harve...
4    John Carter is a war-weary, former military ca...
Name: overview, dtype: object
```

As you can see, we need to remove stopwords [a, the etc) before feeding them into the vectorizer.
We will use TfidfVectorizer here.

TfidfVectorizer:

Simply using the word count as a feature value of a word really doesn't reflect the importance of that word in a document. For example, if a word is present frequently in all documents in a corpus, then its count value in different documents is not helpful in discriminating between different documents. On other hand, if a word is present only in a few of documents, then its count value in those documents can help discriminating them from the rest of the documents. Thus, *the importance of a word, i.e. its feature value, for a document not only depends upon how often it is present in that document but also how is its overall presence in the corpus*. This notion of importance of a word in a document is captured by a scheme, known as the *term frequency-inverse document frequency (tf-idf*) weighting scheme.

The term frequency is a ratio of the count of a word's occurrence in a document and the number of words in the document. Thus, it is a normalized measure that takes into consideration the document length. Let us show the count of word i in document j by tf_{ij}. The document frequency of word i represents the number of documents in the corpus with word i in them. Let us represent document frequency for word i by df_i. With N as the number of documents in the corpus, the tf-idf weight w_{ij} for word i in document j is computed by the following formula:

$$w_{ij} = tf_{ij} \times (1 + \log \frac{1+N}{1+df_{ij}})$$

We fed the Overview feature to the **TfidfVectorizer** . We used linear_kernel to calculate the matrix values. We can now get the top recommendations for a few movies.

```
movie_recommend('The Dark Knight')
```

```
3                              The Dark Knight Rises
428                                   Batman Returns
3854    Batman: The Dark Knight Returns, Part 2
299                                   Batman Forever
1359                                          Batman
119                                    Batman Begins
1181                                            JFK
9             Batman v Superman: Dawn of Justice
2507                                      Slow Burn
210                                   Batman & Robin
Name: title, dtype: object
```

We will now use "cast", "crew", "keywords", "genres" features to find similarities

Data Wrangling :

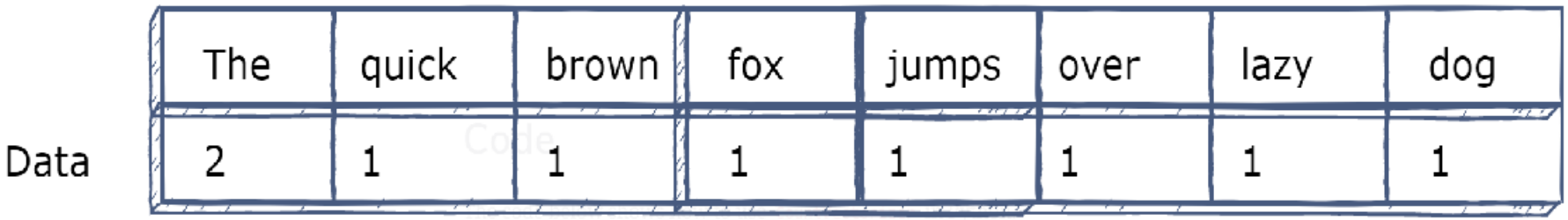
In this data set, the movie data is present in the form of lists containing strings.
Using few python function, I converted the stringfield data into a safe and usable structure.
I am going to build a recommender based on top 3 actors, the director, genres and keywords.
Let's see how the data looks like after the above transformations. The next step would be to convert the above feature instances into lowercase and remove all the spaces between them.

cast	director	keywords	genres
[samworthington, zoesaldana, sigourneyweaver]	jamescameron	[cultureclash, future, spacewar]	[action, adventure, fantasy]
[johnnydepp, orlandobloom, keiraknightley]	goreverbinski	[ocean, drugabuse, exoticisland]	[adventure, fantasy, action]
[danielcraig, christophwaltz, léaseydoux]	sammendes	[spy, basedonnovel, secretagent]	[action, adventure, crime]
[christianbale, michaelcaine, garyoldman]	christophernolan	[dccomics, crimefighter, terrorist]	[action, crime, drama]
[taylorkitsch, lynncollins, samanthamorton]	andrewstanton	[basedonnovel, mars, medallion]	[action, adventure, sciencefiction]

In order to use textual data for predictive modeling, the text must be parsed to remove certain words – this process is called **tokenization**. These words need to then be encoded as integers, or floating-point values, for use as inputs in machine learning algorithms. This process is called **feature extraction (or vectorization)**.

Scikit-learn's CountVectorizer is used to convert a collection of text documents to a vector of term/token counts. It also enables the pre-processing of text data prior to generating the vector representation.

```
Data = ['The', 'quick', 'brown', 'fox', 'jumps', 'over', ' the', 'lazy', 'dog']
```



	The	quick	brown	fox	jumps	over	lazy	dog
Data	2	1	1	1	1	1	1	1

Our movie recommendation engine works by suggesting movies to the user based on the metadata information. The similarity between the movies is calculated and then used to make recommendations. For that, our text data should be preprocessed and converted into a vectorizer using the CountVectorizer.

As the name suggests, CountVectorizer counts the frequency of each word and outputs a 2D vector containing frequencies.

There exist several similarity score functions such as cosine similarity, Pearson correlation coefficient, etc.
Here, I used the cosine similarity score as this is just the dot product of the vector output by the CountVectorizer.

Corpus creation:
Let's create a corpus containing all of the metadata information extracted to input into the vectorizer.

```
0     cultureclash future spacewar samworthington zo...
1     ocean drugabuse exoticisland johnnydepp orland...
2     spy basedonnovel secretagent danielcraig chris...
3     dccomics crimefighter terrorist christianbale ...
4     basedonnovel mars medallion taylorkitsch lynnc...
Name: corpus, dtype: object
```

Our recommender system is now ready to recommend the similar movies based on the similar features.

```
get_recommendations("The Dark Knight", cosine_sim)
```

```
3                The Dark Knight Rises
119                        Batman Begins
4638    Amidst the Devil's Wings
2398                                Hitman
1720                             Kick-Ass
1740                         Kick-Ass 2
3326                    Black November
1503                              Takers
1986                              Faster
303                             Catwoman
Name: title, dtype: object
```