NETFLIX MOVIE RECOMMENDATION SYSTEM

Netflix is an application that keeps growing bigger and faster with its popularity, shows and content. This is a story telling through its data along with a content-based recommendation system and a wide range of different graphs and visuals

Problem Statement :

Recommendation systems have been around with us for a while now, and they are so powerful. They do have a strong influence on our decisions these days. From movie streaming services to online shopping stores, they are almost everywhere we look. If you are wondering how they know what you might buy after adding an “x” item to your cart, the answer is simple: Power of Data.

Recommendation systems are a very interesting field of machine learning. Recommendation system recommends the movie based on the users movie choices.

Data Source :

MovieLens Dataset : https://grouplens.org/datasets/movielens/

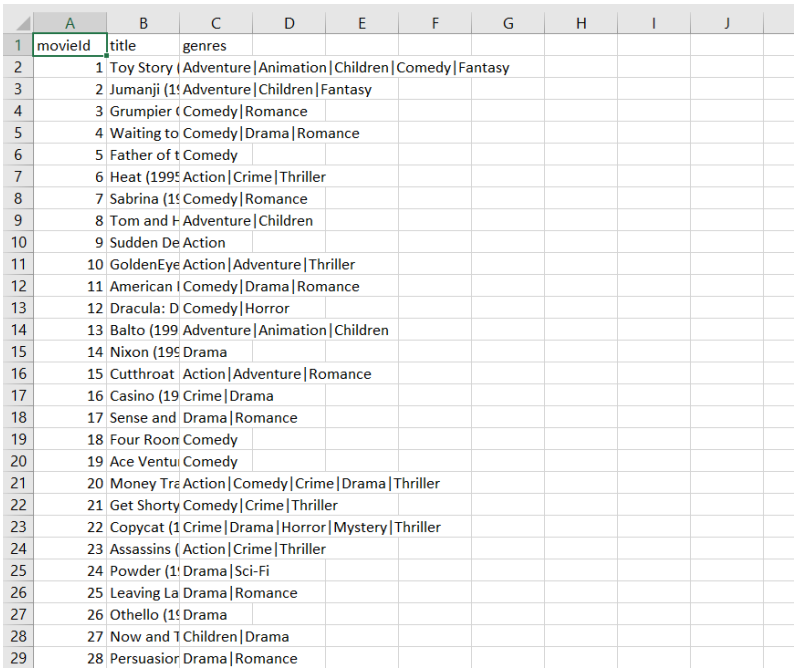
This dataset contains 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users.

FEATURE DESCRIPTION :

- MovieId (Quantitative) : Uniquely Identifies the movie
- Title (Qualitative) : Title of the Movie
- Genres (Categorical) : Defines a movie based on narrative elements
- UserId (Quantitative): Uniquely Identifies the user
- Ratings (Quantitative) : Rating of the movie
- Tag (Qualitative): Defines the type of movie
- TimeStamp (Quantitative): Timestamp of the rating

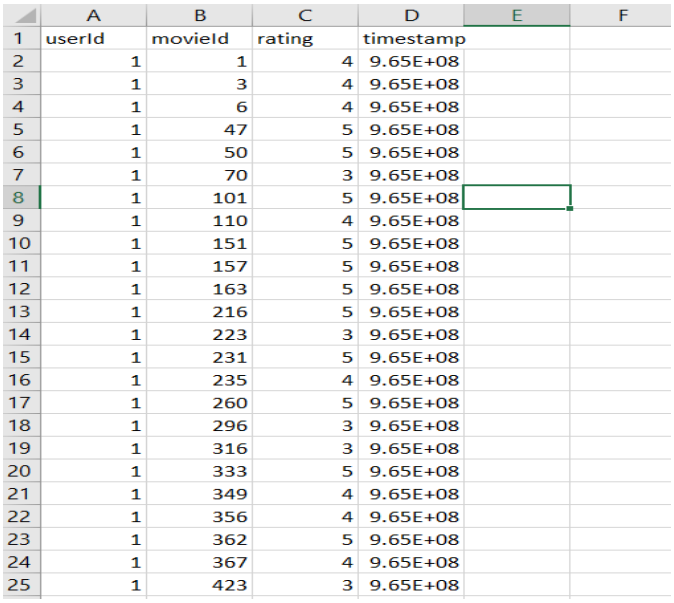
SAMPLE DATASET :

Movies.csv



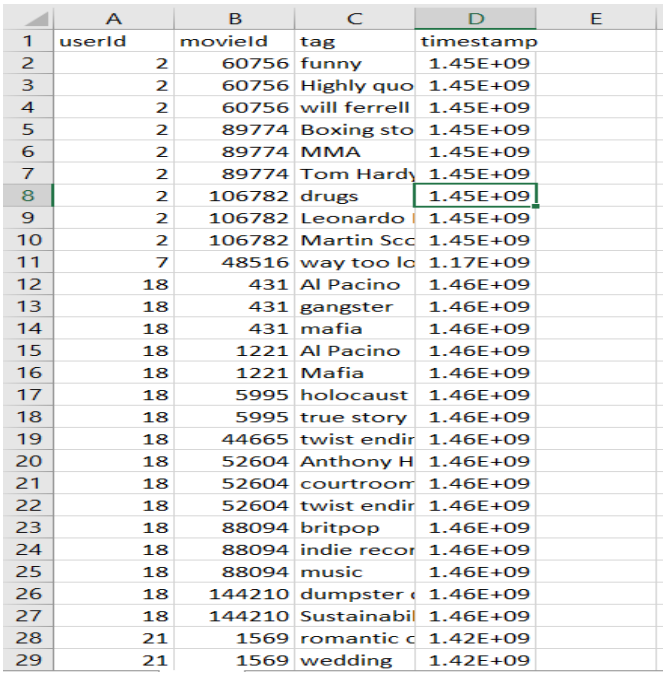
	A	B	C	D	E	F	G	H	I	J
1	movieId	title	genres							
2	1	Toy Story (	Adventure|Animation|Children|Comedy|Fantasy							
3	2	Jumanji (1	Adventure|Children|Fantasy							
4	3	Grumpier (	Comedy|Romance							
5	4	Waiting to	Comedy|Drama|Romance							
6	5	Father of t	Comedy							
7	6	Heat (1995	Action|Crime|Thriller							
8	7	Sabrina (1	Comedy|Romance							
9	8	Tom and H	Adventure|Children							
10	9	Sudden De	Action							
11	10	GoldenEye	Action|Adventure|Thriller							
12	11	American I	Comedy|Drama|Romance							
13	12	Dracula: D	Comedy|Horror							
14	13	Balto (199	Adventure|Animation|Children							
15	14	Nixon (199	Drama							
16	15	Cutthroat	Action|Adventure|Romance							
17	16	Casino (19	Crime|Drama							
18	17	Sense and	Drama|Romance							
19	18	Four Roon	Comedy							
20	19	Ace Ventu	Comedy							
21	20	Money Tra	Action|Comedy|Crime|Drama|Thriller							
22	21	Get Shorty	Comedy|Crime|Thriller							
23	22	Copycat (1	Crime|Drama|Horror|Mystery|Thriller							
24	23	Assassins (	Action|Crime|Thriller							
25	24	Powder (1	Drama|Sci-Fi							
26	25	Leaving La	Drama|Romance							
27	26	Othello (1	Drama							
28	27	Now and T	Children|Drama							
29	28	Persuasior	Drama|Romance							

Ratings.csv



	A	B	C	D	E	F
1	userId	movieId	rating	timestamp		
2	1	1	4	9.65E+08		
3	1	3	4	9.65E+08		
4	1	6	4	9.65E+08		
5	1	47	5	9.65E+08		
6	1	50	5	9.65E+08		
7	1	70	3	9.65E+08		
8	1	101	5	9.65E+08		
9	1	110	4	9.65E+08		
10	1	151	5	9.65E+08		
11	1	157	5	9.65E+08		
12	1	163	5	9.65E+08		
13	1	216	5	9.65E+08		
14	1	223	3	9.65E+08		
15	1	231	5	9.65E+08		
16	1	235	4	9.65E+08		
17	1	260	5	9.65E+08		
18	1	296	3	9.65E+08		
19	1	316	3	9.65E+08		
20	1	333	5	9.65E+08		
21	1	349	4	9.65E+08		
22	1	356	4	9.65E+08		
23	1	362	5	9.65E+08		
24	1	367	4	9.65E+08		
25	1	423	3	9.65E+08		

Tags.csv



	A	B	C	D	E
1	userId	movieId	tag	timestamp	
2	2	60756	funny	1.45E+09	
3	2	60756	Highly quo	1.45E+09	
4	2	60756	will ferrell	1.45E+09	
5	2	89774	Boxing sto	1.45E+09	
6	2	89774	MMA	1.45E+09	
7	2	89774	Tom Hardy	1.45E+09	
8	2	106782	drugs	1.45E+09	
9	2	106782	Leonardo	1.45E+09	
10	2	106782	Martin Scc	1.45E+09	
11	7	48516	way too lc	1.17E+09	
12	18	431	Al Pacino	1.46E+09	
13	18	431	gangster	1.46E+09	
14	18	431	mafia	1.46E+09	
15	18	1221	Al Pacino	1.46E+09	
16	18	1221	Mafia	1.46E+09	
17	18	5995	holocaust	1.46E+09	
18	18	5995	true story	1.46E+09	
19	18	44665	twist endir	1.46E+09	
20	18	52604	Anthony H	1.46E+09	
21	18	52604	courtroom	1.46E+09	
22	18	52604	twist endir	1.46E+09	
23	18	88094	britpop	1.46E+09	
24	18	88094	indie recor	1.46E+09	
25	18	88094	music	1.46E+09	
26	18	144210	dumpster	1.46E+09	
27	18	144210	Sustainabil	1.46E+09	
28	21	1569	romantic c	1.42E+09	
29	21	1569	wedding	1.42E+09	

All these datasets are merged through and to be used for our recommendation system.

TOOLS USED :

Python : Python is an interpreted, high-level, general-purpose programming language used for performing the statistical analysis. When applying the technique of Web Scraping, Python coding will scrap the internet for selected data.

Open CV : OpenCV is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then It sees. The library is cross-platform and free for use under the open-source BSD license.

Pandas : Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

Numpy : NumPy is a python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices.

Seaborn : Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics

Methodology:

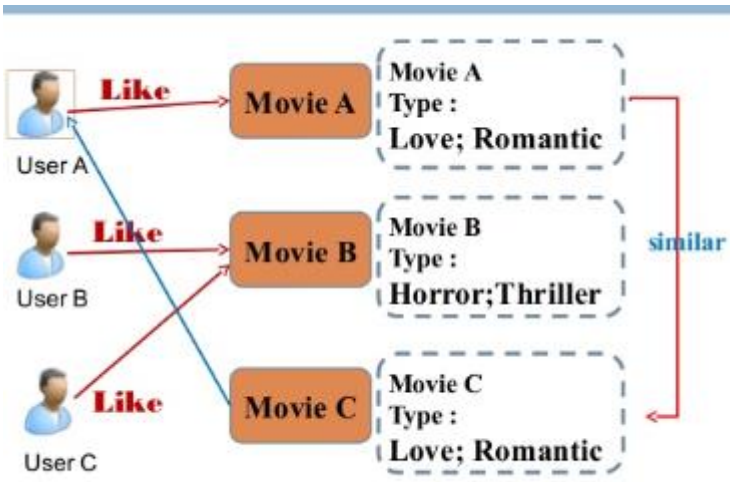
There are 5 major steps involved in the building a ML model for Movie Recommendation System.This encapsulates the following steps:

- Data loading
- Data cleaning
- Data Analysis
- Recommendation model

Demographic Filtering :

Contenet Based Filtering :

In this recommender system the content of the movie (overview, cast, crew, keyword, tagline etc) is used to find its similarity with other movies. Then the movies that are most likely to be similar are recommended.



Collaborative Filtering :

Collaborative Filtering

Our content based engine suffers from some severe limitations. It is only capable of suggesting movies which are close to a certain movie. That is, it is not capable of capturing tastes and providing recommendations across genres.

Also, the engine that we built is not really personal in that it doesn't capture the personal tastes and biases of a user. Anyone querying our engine for recommendations based on a movie will receive the same recommendations for that movie, regardless of who she/he is.

Therefore, in this section, we will use a technique called Collaborative Filtering to make recommendations to Movie Watchers. It is basically of two types:-

User based filtering- These systems recommend products to a user that similar users have liked. For measuring the similarity between two users we can either use pearson correlation or cosine similarity. This filtering technique can be illustrated with an example. In the following matrixes, each row represents a user, while the columns correspond to different movies except the last one which records the similarity between that user and the target user. Each cell represents the rating that the user gives to that movie. Assume user E is the target.

Item Based Collaborative Filtering - Instead of measuring the similarity between users, the item-based CF recommends items based on their similarity with the items that the target user rated. Likewise, the similarity can be computed with Pearson Correlation or Cosine Similarity. The major difference is that, with item-based collaborative filtering, we fill in the blank vertically, as oppose to the horizontal manner that user-based CF does.

EVALUATION METRIC :

Personalization :

Personalization is a great way to assess if a model recommends many of the same items to different users. It is the dissimilarity (1- cosine similarity) between user's lists of recommendations.

Intra-list Similarity:

Intra-list similarity is the average cosine similarity of all items in a list of recommendations. This calculation uses features of the recommended items (such as movie genre) to calculate the similarity.

Coverage:

Coverage is the percent of items in the training data the model is able to recommend on a test set. In this example, the popularity recommender has only 0.05% coverage, since it only ever recommends 10 items. The random recommender has nearly 100% coverage as expected. Surprisingly, the collaborative filter is only able to recommend 8.42% of the items it was trained on.

EXPLORATORY DATA ANALYSIS :

Table :

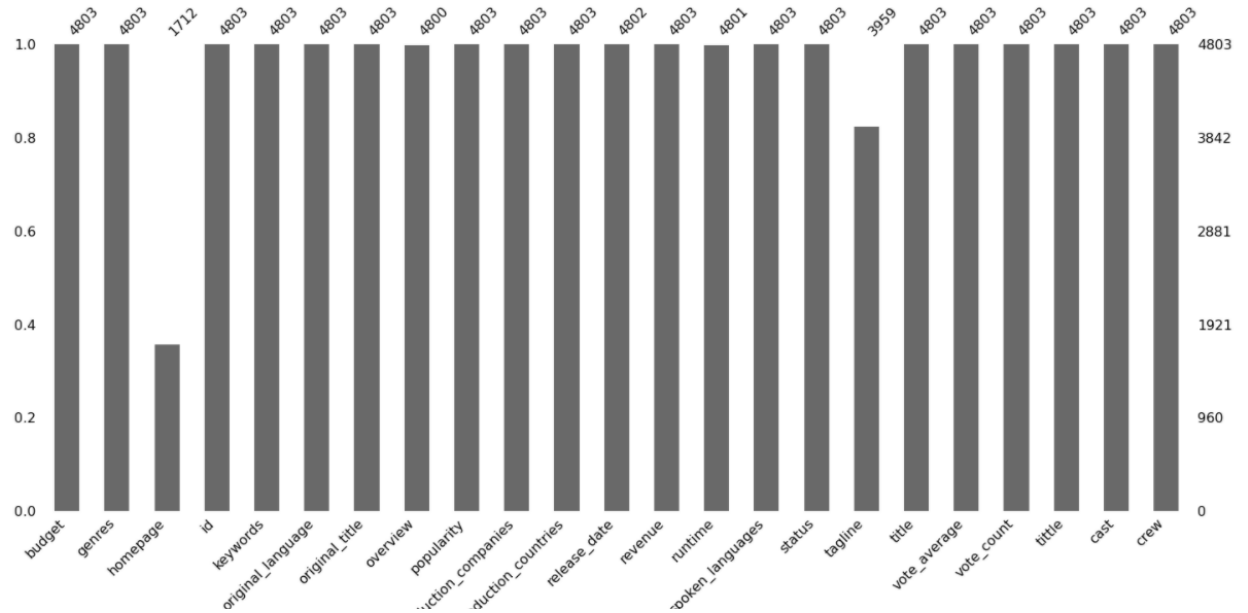
```
df_movies.describe()
```

	budget	id	popularity	revenue	runtime	vote_average	vote_count
count	4.803000e+03	4803.000000	4803.000000	4.803000e+03	4801.000000	4803.000000	4803.000000
mean	2.904504e+07	57165.484281	21.492301	8.226064e+07	106.875859	6.092172	690.217989
std	4.072239e+07	88694.614033	31.816650	1.628571e+08	22.611935	1.194612	1234.585891
min	0.000000e+00	5.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000
25%	7.900000e+05	9014.500000	4.668070	0.000000e+00	94.000000	5.600000	54.000000
50%	1.500000e+07	14629.000000	12.921594	1.917000e+07	103.000000	6.200000	235.000000
75%	4.000000e+07	58610.500000	28.313505	9.291719e+07	118.000000	6.800000	737.000000
max	3.800000e+08	459488.000000	875.581305	2.787965e+09	338.000000	10.000000	13752.000000

The above table gives the complete description of our data . We can check the mean , standard deviation , Min and Max of all the columns in our data.

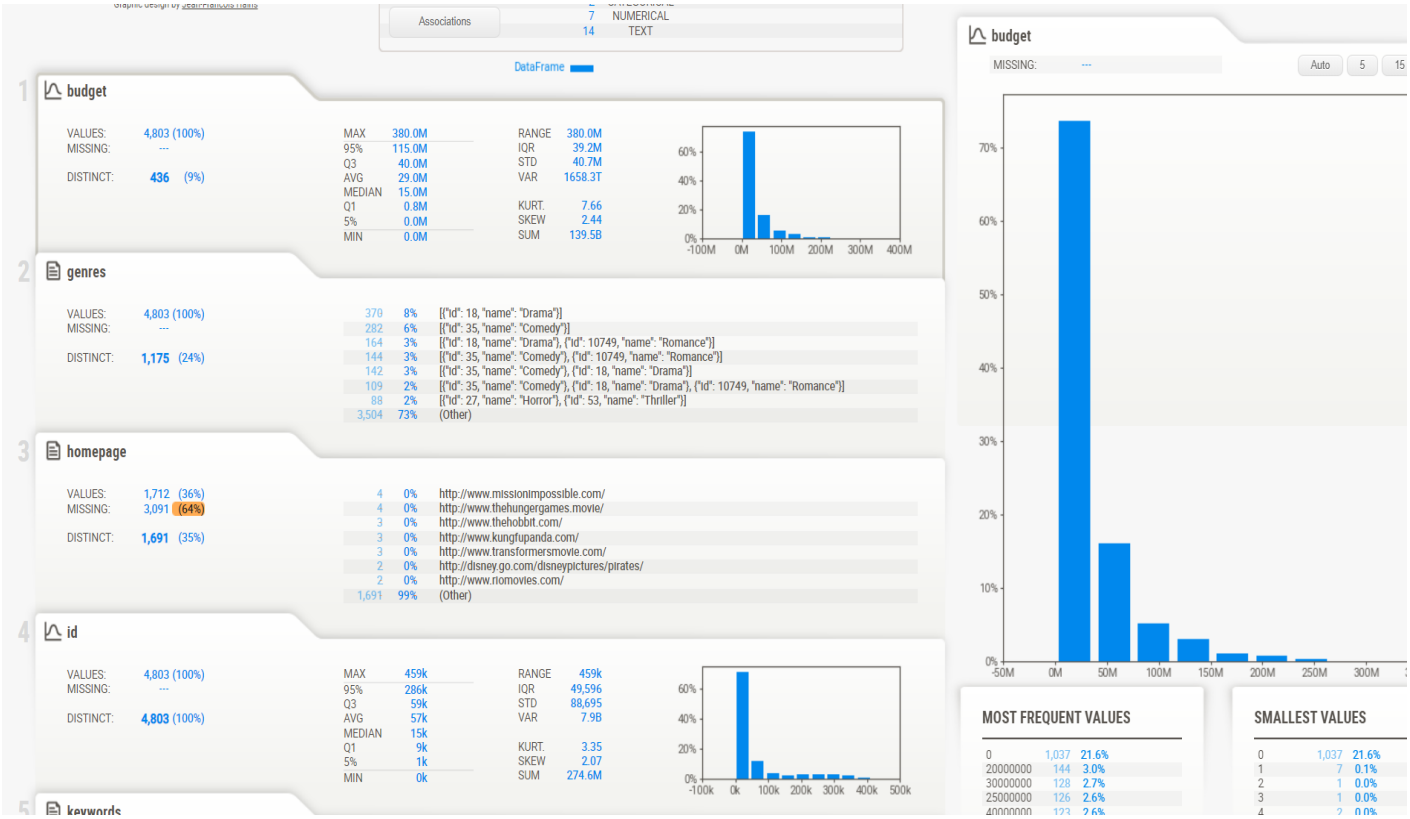
Bar Graph :

```
import missingno as msno
msno.bar(df_movies);
```



The above bar chart gives Information about the missing values in all the columns of the data. We can observe that homepage and tagline column has more missing values than any other column.

Sweetviz :

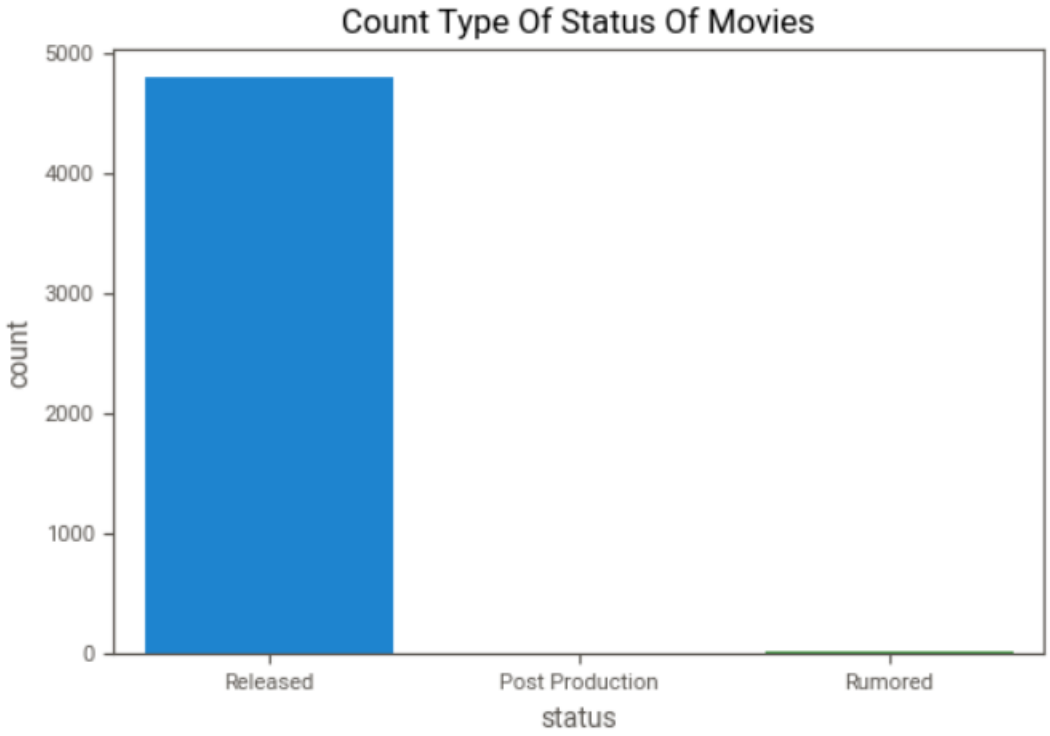


The above chart using the sweetviz gives the detailed information about each column in the dataframe . It gives the the count of each value and maximum,minimum values of each column.

Count Plot :

```
d = sns.countplot(x='status',data=df_movies)
d.axes.set_title("Count Type Of Status Of Movies")
```

```
Text(0.5, 1.0, 'Count Type Of Status Of Movies')
```



The above count plot gives the count of Different status of Movies . Most of the movies have Released status and very few movies have Rumored status.

Table :

Title	Budget
Chasing Papi	0
The Savages	0
Breakin' All the Rules	0
Jab Tak Hai Jaan	0
Red Dog	0
Kites	0
Richard III	0
Good Intentions	0
Partition	0
It's a Wonderful Afterlife	0
Jungle Shuffle	0
Fifty Dead Men Walking	0
White Noise 2: The Light	0
Eulogy	0
The Reef	0
Free Style	0

The above table gives information about the lowest budget movies . There are movies that are made with zero budget.

Title	Budget
Pirates of the Caribbean: At World's End	300000000
Avengers: Age of Ultron	280000000
Superman Returns	270000000
John Carter	260000000
Tangled	260000000
Spider-Man 3	258000000
The Lone Ranger	255000000
X-Men: Days of Future Past	250000000
The Hobbit: The Desolation of Smaug	250000000
Captain America: Civil War	250000000
Batman v Superman: Dawn of Justice	250000000
The Hobbit: An Unexpected Journey	250000000
The Hobbit: The Battle of the Five Armies	250000000
Harry Potter and the Half-Blood Prince	250000000
The Dark Knight Rises	250000000
Spectre	245000000

We can observe from the table that the maximum budget of the movie is 3000000000 and next highest is 28

FEATURE ENGINEERING :

Feature engineering is the process of using domain knowledge to extract features from raw data via data mining techniques. These features can be used to improve the performance of machine learning algorithms. Feature engineering can be considered as applied machine learning itself.

```
# Some intervals were determined according to the vote_count and these were assigned categorical variables.
Newvotecount = pd.Series(["Low", "Normal", "Average", "High"], dtype = "category")

df_movies["new_vote_count"] = Newvotecount
df_movies.loc[df_movies["vote_count"] <= 70, "new_vote_count"] = Newvotecount[0]
df_movies.loc[(df_movies["vote_count"] > 70) & (df_movies["vote_count"] <= 99), "new_vote_count"] = Newvotecount[1]
df_movies.loc[(df_movies["vote_count"] > 99) & (df_movies["vote_count"] <= 126), "new_vote_count"] = Newvotecount[2]
df_movies.loc[df_movies["vote_count"] > 126 ,"new_vote_count"] = Newvotecount[3]
```

ONE HOT ENCODING:

A one hot encoding is a representation of categorical variables as binary vectors. This first requires that the categorical values be mapped to integer values.

ONE HOT ENCODING

```
# pandas.get_dummies() is used for data manipulation. It converts categorical data into dummy variables.
df_movies = pd.get_dummies(df_movies, columns =["new_vote_count"], drop_first = True)
```

FILTERING TECHNIQUES :

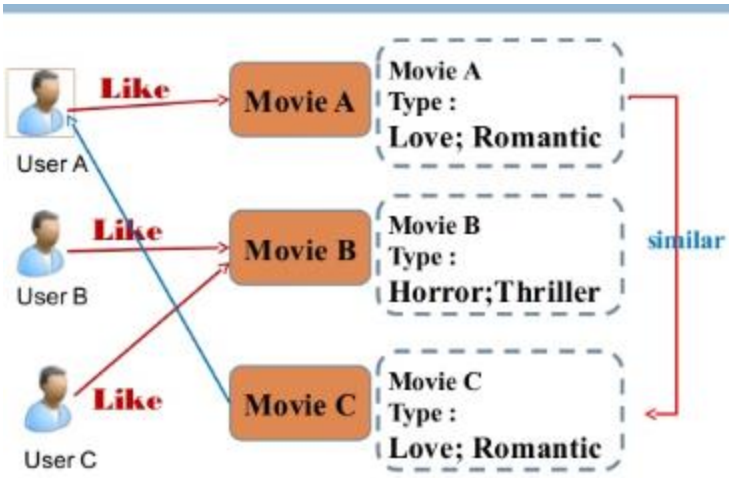
Content Based Filtering :

In this recommender system the content of the movie (overview, cast, crew, keyword, tagline etc) is used to find its similarity with other movies. Then the movies that are most likely to be similar are recommended.

Now Term Frequency-Inverse Document Frequency (TF-IDF) vectors for each overview is calculated .Now if you are wondering what is term frequency , it is the relative frequency of a word in a document and is given as (term instances/total instances). Inverse Document Frequency is the relative count of documents containing the term is given as log(number of documents/documents with term) The overall importance of each word to the documents in which they appear is equal to TF * IDF

This will give a matrix where each column represents a word in the overview vocabulary (all the words that appear in at least one document) and each row represents a movie, as before.This is

done to reduce the importance of words that occur frequently in plot overviews and therefore, their significance in computing the final similarity score.



With this matrix in hand, we can now compute a similarity score. There are several candidates for this; such as the euclidean, the Pearson and the cosine similarity scores. There is no right answer to which score is the best. Different scores work well in different scenarios and it is often a good idea to experiment with different metrics.

We will be using the cosine similarity to calculate a numeric quantity that denotes the similarity between two movies. We use the cosine similarity score since it is independent of magnitude and is relatively easy and fast to calculate. Mathematically, it is defined as follows:

Since we have used the TF-IDF vectorizer, calculating the dot product will directly give us the cosine similarity score. Therefore, we will use sklearn's linear_kernel() instead of cosine_similarities() since it is faster.

```python
# Function that takes in movie title as input and outputs most similar movies
def get_recommendations(title, cosine_sim=cosine_sim):
    # Get the index of the movie that matches the title
    idx = indices[title]

    # Get the pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return df_movies['title'].iloc[movie_indices]
```

```python
get_recommendations('The Dark Knight Rises')
```

```
65                          The Dark Knight
119                           Batman Begins
9       Batman v Superman: Dawn of Justice
2193                 Secret in Their Eyes
1398                             Max Payne
979                   Free State of Jones
2416                              Beastly
2274                             Survivor
790                       American Sniper
4649                      Shotgun Stories
Name: title, dtype: object
```

Collaborative Filtering :

User based filtering- These systems recommend products to a user that similar users have liked. For measuring the similarity between two users we can either use pearson correlation or cosine similarity.

Item Based Collaborative Filtering - Instead of measuring the similarity between users, the item-based CF recommends items based on their similarity with the items that the target user rated. Likewise, the similarity can be computed with Pearson Correlation or Cosine Similarity.

```
ratings[ratings['userId'] == 1]
```

	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
5	1	1263	2.0	1260759151
6	1	1287	2.0	1260759187
7	1	1293	2.0	1260759148
8	1	1339	3.5	1260759125
9	1	1343	2.0	1260759131
10	1	1371	2.5	1260759135
11	1	1405	1.0	1260759203
12	1	1953	4.0	1260759191
13	1	2105	4.0	1260759139
14	1	2150	3.0	1260759194
15	1	2193	2.0	1260759198
16	1	2294	2.0	1260759108
17	1	2455	2.5	1260759113
18	1	2968	1.0	1260759200
19	1	3671	3.0	1260759117

```
svd.predict(1, 302, 3)
```

```
]: Prediction(uid=1, iid=302, r_ui=3, est=2.629630982047076, details={'was_impossible': False})
```

Single Value Decomposition

One way to handle the scalability and sparsity issue created by CF is to leverage a latent factor model to capture the similarity between users and items. Essentially, we want to turn the recommendation problem into an optimization problem. We can view it as how good we are in predicting the rating for items given a user. One common metric is Root Mean Square Error (RMSE). The lower the RMSE, the better the performance.

It is a broad idea which describes a property or concept that a user or an item have. For instance, for music, latent factor can refer to the genre that the music belongs to. SVD decreases the dimension of the utility matrix by extracting its latent factors. Essentially, we map each user and each item into a latent space with dimension r. Therefore, it helps us better understand the relationship between users and items as they become directly comparable. The below figure illustrates this idea.

```
In [51]: svd = SVD()

In [52]: cross_validate(svd, 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)    0.8963  0.8971  0.9005  0.9033  0.8946  0.8984  0.0031
MAE (testset)     0.6890  0.6929  0.6914  0.6950  0.6918  0.6920  0.0020
Fit time          4.46    4.27    4.05    4.44    4.21    4.28    0.15
Test time         0.11    0.23    0.22    0.13    0.13    0.16    0.05

Out[52]: {'test_rmse': array([0.8962738 , 0.89711062, 0.90054653, 0.90328481, 0.89463493]),
          'test_mae': array([0.68898873, 0.69285016, 0.6914444 , 0.69500823, 0.69177713]),
          'fit_time': (4.458118200302124,
           4.271575450897217,
           4.045140743255615,
           4.441352367401123,
           4.206753730773926),
          'test_time': (0.11465001106262207,
           0.23137950897216797,
           0.21850323677062988,
           0.13161158561706543,
           0.12861371040344238)}

In [53]: trainset = data.build_full_trainset()
         svd.fit(trainset)

Out[53]: <surprise.prediction_algorithms.matrix_factorization.SVD at 0x1a5cc3fe3a0>
```

WEB APP:

Conclusion : Hybrid Systems can take advantage of content-based and collaborative filtering as the two approaches are proved to be almost complimentary.In this Recommendation system I am using content based Filtering as it compares the similarity of movies.