Project Title:

Movie Recommendation System Project

Objective:

This mini project aims to develop a movie recommendation system using machine learning algorithms. The goal of the system is to provide personalized recommendations to users based on their past movie experiences .The goal of this project is to provide useful and personalized movie recommendation system for users.

Introduction:

Movies are a universal source of entertainment, enjoyed by people of all ages, genders, races, colors, and from different parts of the world. Despite our common interest in movies, each person has their unique preferences and combinations when it comes to the type of movies they enjoy. Some people prefer a specific genre, such as thrillers, romance, or science fiction, while others focus on their favorite actors and directors.

It is challenging to generalize a movie and claim that everyone will like it, given the variety of tastes and preferences. However, certain movies tend to be more popular with specific groups of people who share similar interests. Movie recommendation systems have become an integral part of the modern world of entertainment. With the rise of online streaming platforms, there is an overwhelming amount of content available to users.

As a result, users often struggle to find new movies or TV shows that match their interests.

This is where recommendation systems come in. And where data scientists come into play and extract the juice out of all the behavioral patterns of not only the audience but also from the movies themselves. In this project, we have implemented a movie recommendation system using machine learning techniques.

The project is not only useful for online streaming platforms but also for other businesses that rely on personalized recommendations to increase their revenue and customer satisfaction. The system

can be easily scaled to handle many users and movies, making it a powerful tool for businesses of all sizes.

Background:

The movie industry has seen significant growth in recent years, with the number of movies released each year increasing steadily. With this growth, there has been a need for movie recommendation systems to help users navigate the vast amount of content available. These systems use machine learning algorithms to provide personalized movie recommendations to users based on their past movie ratings and preferences.

Movie recommendation systems have been successfully used by major movie streaming platforms such as Netflix, Hulu, and Amazon Prime Video.

What is a Recommendation System?

A Recommendation System is a **filtration program** whose prime goal is to predict the **preference** of user towards domain-specific item or item. In this case, this domain specific item is movie, therefore the focus of recommendation system is to filter and predict only those movies which a user

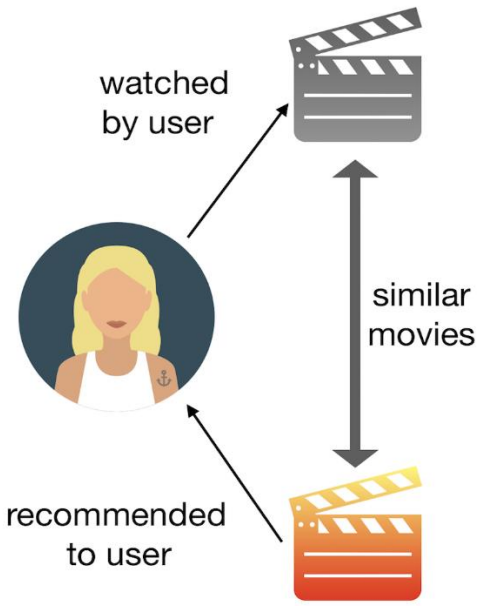
would prefer given some data about the user him or herself.

What are Different Filtration Strategies?

Let's have a look at Different Filtration Strategies and how they work using movie recommendation systems as a base.

1) Content-Based Movie Recommendation Systems:

This filtration strategy is based on the data provided about the items. The algorithm recommends products that are like the ones that a user has liked in the past. This similarity (generally cosine similarity) is computed from the data we have about the items as well as the user's past preferences.



Assumption:

This approach assumes that users who have liked a particular item in the past are likely to enjoy similar items in the future.

For example: If a user watches a comedy movie starring Adam Sandler, the system will recommend them movies in the same genre or starring the same actor, or both. The input for building a content-based recommender system is movie attributes.

2) Collaborative-Based Movie Recommendation System:

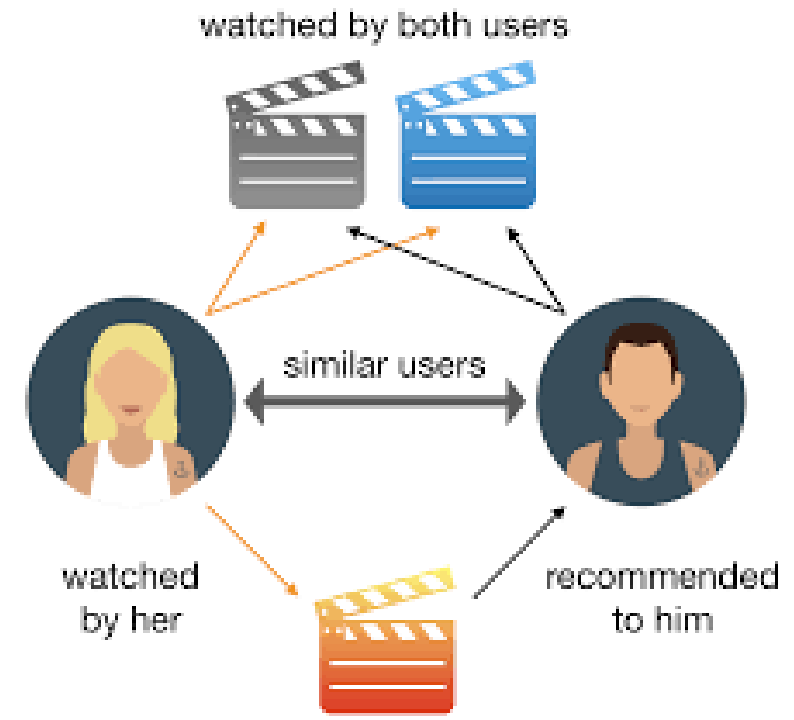
This filtration strategy is based on the combination of the user's behavior and comparing the user's behavior with other users' behavior in the database. The history of all users plays an important role in this algorithm.

The main difference between content-based filtering and collaborative filtering that in the latter, the interaction of all users with the items influences the recommendation algorithm while for content-based filtering only the concerned user's data is considered.

Assumption:

This approach assumes that users who have similar preferences in the past are likely to have similar preferences in the future.

There are multiple ways to implement collaborative filtering but the main concept is that in collaborative filtering multiple user's data influences the outcome of the recommendation. and doesn't depend on only one user's data for modeling.



There are 2 types of collaborative filtering algorithms:

- **User-based Collaborative Filtering:**

The basic idea here is to find users that have similar past preference patterns as the user 'A' has had and then recommending him or her items liked by those similar users which 'A' has not encountered yet.

This is achieved by making a matrix of items each user has rated or viewed or liked depending upon the task at hand, and then computing the similarity score between the users and finally recommending items that the concerned user isn't aware of but users like him/her are and liked it.

For Example: If the user 'A' likes 'Batman Begins', 'Justice League' and 'The Avengers' while the user 'B' likes 'Batman Begins', 'Justice League' and 'Thor' then they have similar interests because we know that these movies belong to the super-hero genre. So, there is a high probability that the user 'A' would like 'Thor' and the user 'B' would like The Avengers.

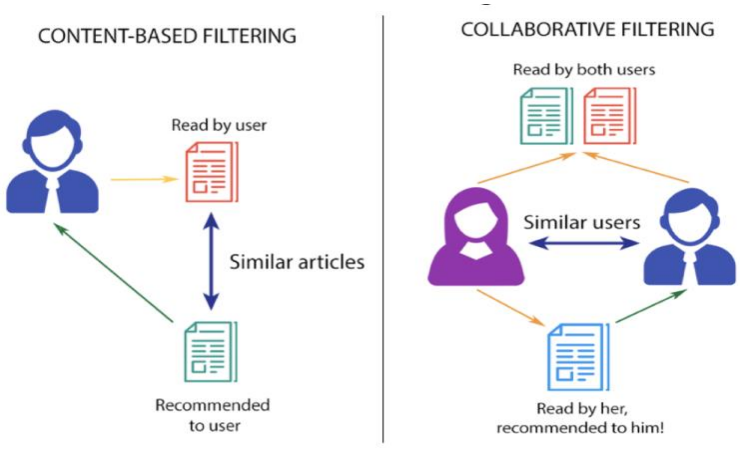
- **Item-based Collaborative Filtering:**

The concept in this case is to find similar movies instead of similar users and then recommending similar movies to 'A' that has had in his

or her past preferences. This is executed by finding every pair of items that were rated or viewed or liked by the same user, then measuring the similarity of those rated or viewed or liked across all user who rated/viewed/liked/clicked both, and finally recommending them based on similarity scores.

For Example: We take 2 movies 'A' and 'B' and check their ratings by all users who have rated both the movies and based on the similarity of these ratings, and based on this rating similarity by users who have rated both we find similar movies. So, if most common users have rated 'A' and 'B' both similarly and it is highly probable that 'A' and 'B' are similar, therefore if someone has watched and liked 'A' they should be recommended 'B' and vice versa.

Broadly Seeing the Filtration Strategies:



Coding Part:

In this implementation, when the user searches for a movie we will recommend the top 10 similar movies using movie recommendation system. We will be using an **item-based collaborative filtering** algorithm for our purpose.

Exploring The Data Sets:

First, we need to import libraries which we'll be using in our movie recommendation system. Also, we'll import the dataset by adding the path of the CSV files. Here We use two datasets one being **movies** and other being **ratings.**

Movie Dataset has

- **movieId** – once the recommendation is done, we get a list of all similar movieId and get the title for each movie from this dataset.
- **genres**– which is not required for this filtering approach.

The movies dataset is having 9742 rows and 3 columns

```
In [1]: import pandas as pd
        import numpy as np
        from scipy.sparse import csr_matrix
        from sklearn.neighbors import NearestNeighbors
```

```
In [2]: movies = pd.read_csv("movies.csv")
        ratings = pd.read_csv("ratings.csv")
```

```
In [3]: print("The size of the movies dataset:",movies.shape)
        movies.head()
```

```
The size of the movies dataset: (9742, 3)
```

Out[3]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

Ratings Dataset has

- **userId** – unique for each user.
- **movieId** – using this feature, we take the title of the movie from the movies dataset.
- **rating** – Ratings given by each user to all the movies using this we are going to predict the top 10 similar movies.

The ratings dataset is having 100836 rows and 4 columns.

In [1]:
```
import pandas as pd
import numpy as np
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
```

In [2]:
```
movies = pd.read_csv("movies.csv")
ratings = pd.read_csv("ratings.csv")
```

In [3]:
```
print("The size of the ratings dataset:",ratings.shape)
ratings.head()
```

```
The size of the ratings dataset: (100836, 4)
```

Out[3]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

Here, we can see that userId 1 has watched movieId 1 & 3 and rated both 4.0 but has not rated movieId 2 at all. This interpretation is harder to extract from this dataframe.

Therefore, to make things easier to understand and work with, we are going to make a new dataframe where each column would represent each unique userId and each row represents each unique movieId.

Now, for creating new dataframe we use **pivot()** function which takes parameters index, columns, values.

- **index** represents the rows of new dataframe.
- **Columns** represents columns of the dataframe.
- **Values** represents the values of the variable of that combination.

And normally pivot function gives NaN value when there is any missing data.

In [4]:
```
# pivot is an function which returns reshaped dataframe and normally
# index---> rows of the dataframe
# coloumns---> coloumn of the dataframe
# values ---> represents the values of the varaiable of that combination and
# returns NaN values when any missing data is found

final_dataset = ratings.pivot(index='movieId',columns='userId',values='rating')
final_dataset
```

Out[4]:

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
movieId																					
1	4.0	NaN	NaN	NaN	4.0	NaN	4.5	NaN	NaN	NaN	...	4.0	NaN	4.0	3.0	4.0	2.5	4.0	2.5	3.0	5.0
2	NaN	NaN	NaN	NaN	NaN	4.0	NaN	4.0	NaN	NaN	...	NaN	4.0	NaN	5.0	3.5	NaN	NaN	2.0	NaN	NaN
3	4.0	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.0	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
193581	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193583	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193585	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193587	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193609	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

9724 rows × 610 columns

Now, it's much easier to interpret that userId 1 has rated movieId 1& 3 4.0 but has not rated movieId 3,4,5

at all (therefore they are represented as NaN) and therefore their rating data is missing.

Let's fix this and impute NaN with 0 to make things understandable for the algorithm and making the data more eye-soothing. So, replacing this NaN values with 0 we use **fillna()** function.

```
In [7]: final_dataset.fillna(0,inplace=True)
        final_dataset
```

Out[7]:

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
movieId																					
1	4.0	0.0	0.0	0.0	4.0	0.0	4.5	0.0	0.0	0.0	...	4.0	0.0	4.0	3.0	4.0	2.5	4.0	2.5	3.0	5.0
2	0.0	0.0	0.0	0.0	0.0	4.0	0.0	4.0	0.0	0.0	...	0.0	4.0	0.0	5.0	3.5	0.0	0.0	2.0	0.0	0.0
3	4.0	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	2.0	0.0	0.0
4	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
5	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	3.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
193581	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	0.0	0.0	0.0	0.0
193583	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	0.0	0.0	0.0	0.0
193585	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	0.0	0.0	0.0	0.0
193587	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	0.0	0.0	0.0	0.0
193609	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	0.0	0.0	0.0	0.0

9724 rows × 610 columns

Now Let us see the plot showing the **count of the no of movies belonging to each genre in the dataframe.**

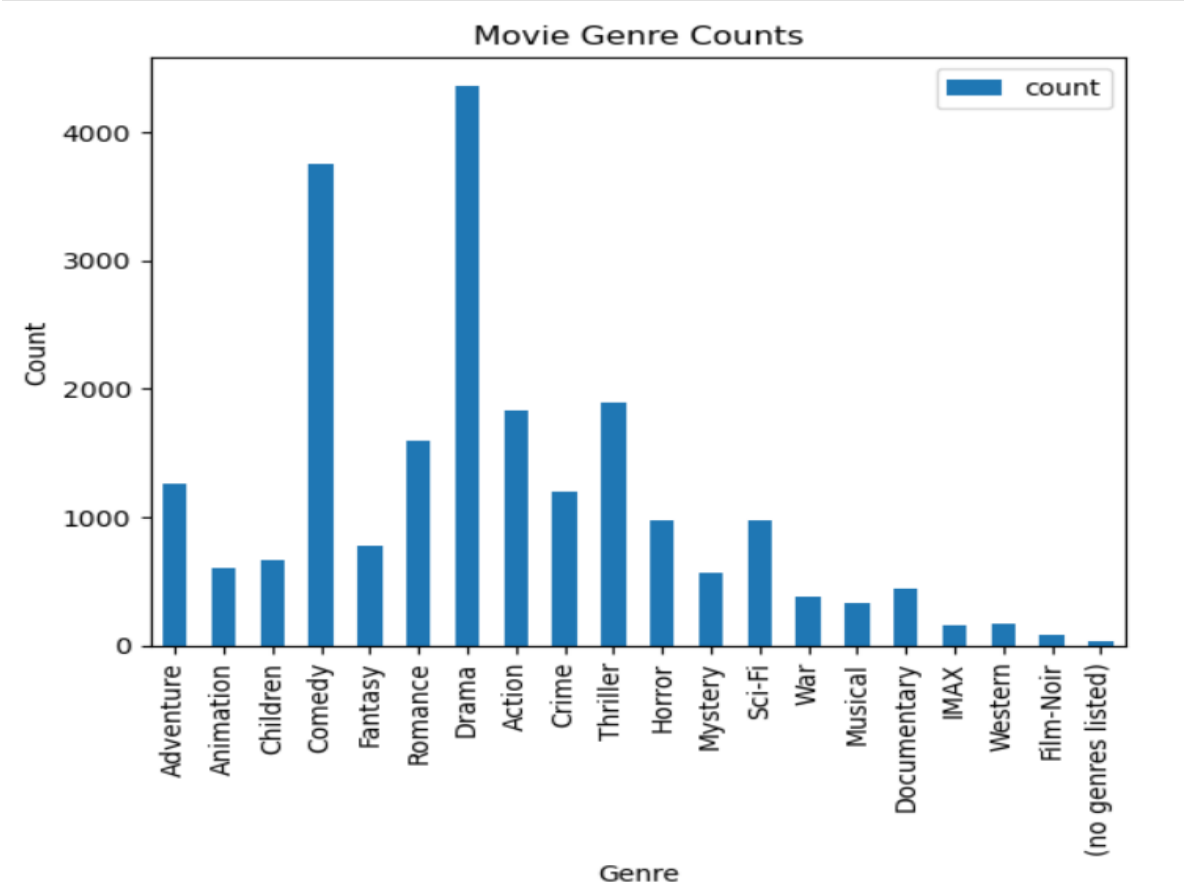
In [27]:
```
# now let us see the count of movies in each genre
# extract the genres column from the dataframe

genres_series = movies['genres']

# create a dictionary to count the movies in each genre
genre_counts = {}
for genres in genres_series:
    # split the string of genres into a list of individual genres
    genre_list = genres.split('|')
    for genre in genre_list:
        # increment the count for the genre in the dictionary
        if genre in genre_counts:
            genre_counts[genre] += 1
        else:
            genre_counts[genre] = 1


# convert the dictionary to a pandas dataframe
genre_counts_df = pd.DataFrame.from_dict(genre_counts, orient='index', columns=['count'])

# create a bar plot of the genre counts
genre_counts_df.plot(kind='bar')
plt.title('Movie Genre Counts')
plt.xlabel('Genre')
plt.ylabel('Count')
plt.show()
```



Data Preprocessing:

In the real-world, ratings are very sparse and data points are mostly collected from very popular movies and highly engaged users. We wouldn't want movies that were rated by a small number of users because it's not credible enough. Similarly, users who have rated only a few handfuls of the movies should also not be considered.

The data is preprocessed by

- To qualify a movie, a minimum of **10 users** should have voted a movie.
- To qualify a user, a minimum of **50 movies** should have voted by the user.

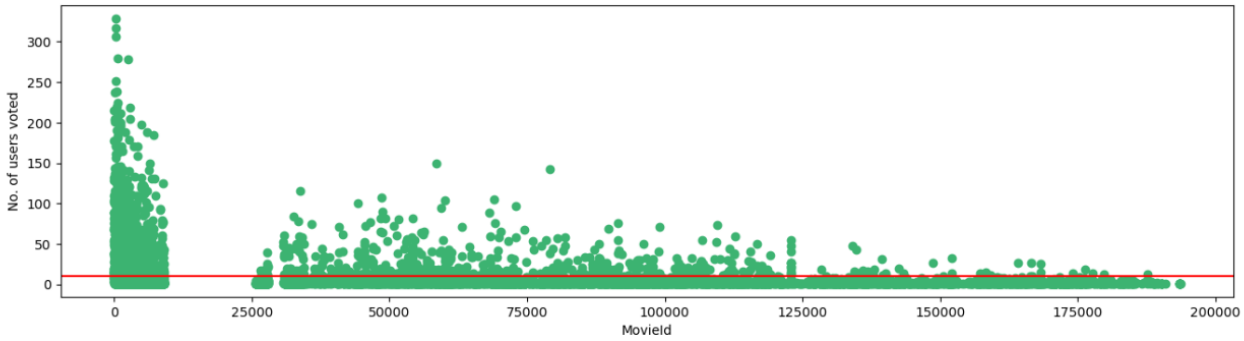
This ensures that the dataset contains enough data to accurately recommend movies to users.

- Aggregating the number of users who voted and the number of movies that were voted.
- Then Let's visualize the no of users who have voted above our threshold with a value of 10. Making the necessary modifications as per the threshold set.

In [8]:
```
no_user_voted = ratings.groupby('movieId')['rating'].agg('count')
no_movies_voted = ratings.groupby('userId')['rating'].agg('count')
```

In [23]:
```
f,ax = plt.subplots(1,1,figsize=(16,4))
# ratings['rating'].plot(kind='hist')

plt.scatter(no_user_voted.index,no_user_voted,color='mediumseagreen')
plt.axhline(y=10,color='r')
plt.xlabel('MovieId')
plt.ylabel('No. of users voted')
plt.show()
```



And let's See final dataset with the above mentioned specifications:

In [11]:
```
final_dataset = final_dataset.loc[no_user_voted[no_user_voted > 10].index,:]
final_dataset
```

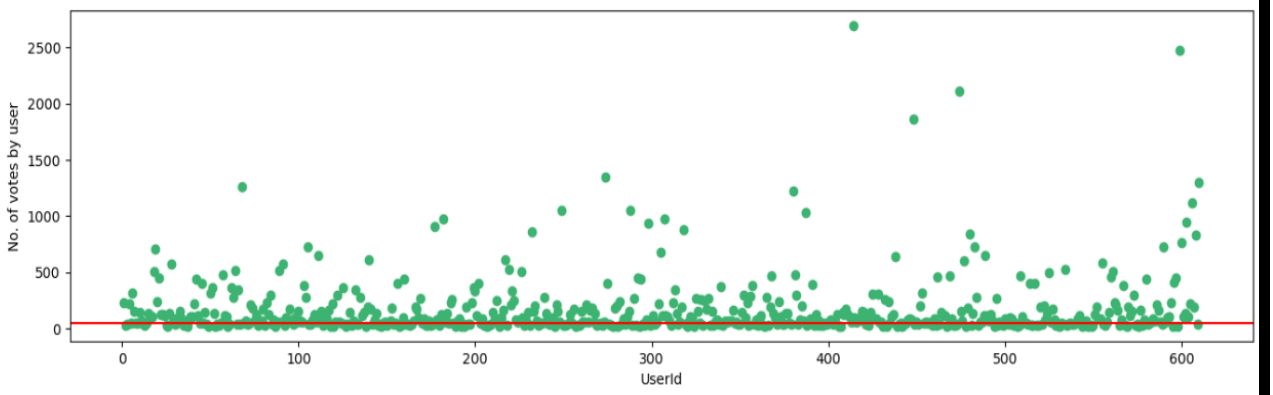
Out[11]:

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
movieId																					
1	4.0	0.0	0.0	0.0	4.0	0.0	4.5	0.0	0.0	0.0	...	4.0	0.0	4.0	3.0	4.0	2.5	4.0	2.5	3.0	5.0
2	0.0	0.0	0.0	0.0	0.0	4.0	0.0	4.0	0.0	0.0	...	0.0	4.0	0.0	5.0	3.5	0.0	0.0	2.0	0.0	0.0
3	4.0	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	2.0	0.0	0.0
5	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	3.0	0.0	0.0	0.0	0.0	0.0	0.0
6	4.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	...	0.0	3.0	4.0	3.0	0.0	0.0	0.0	0.0	0.0	5.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
174055	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
176371	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
177765	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	4.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
179819	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	0.0	0.0	0.0	0.0
187593	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	0.0	0.0	0.0	0.0

2121 rows × 610 columns

- Let's visualize the number of votes by each user with our threshold of 50. Making the necessary modifications as per the threshold set.

```
In [25]: f,ax = plt.subplots(1,1,figsize=(16,4))
         plt.scatter(no_movies_voted.index,no_movies_voted,color='mediumseagreen')
         plt.axhline(y=50,color='r')
         plt.xlabel('UserId')
         plt.ylabel('No. of votes by user')
         plt.show()
```



And let's See final dataset with the above-mentioned specifications:

```
In [26]: final_dataset=final_dataset.loc[:,no_movies_voted[no_movies_voted > 50].index]
         final_dataset
```

Out[26]:

userId	1	4	6	7	10	11	15	16	17	18	...	600	601	602	603	604	605	606	607	608	610
0	4.0	0.0	0.0	4.5	0.0	0.0	2.5	0.0	4.5	3.5	...	2.5	4.0	0.0	4.0	3.0	4.0	2.5	4.0	2.5	5.0
1	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	...	4.0	0.0	4.0	0.0	5.0	3.5	0.0	0.0	2.0	0.0
2	4.0	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	2.0	0.0
3	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	2.5	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0
4	4.0	0.0	4.0	0.0	0.0	5.0	0.0	0.0	0.0	4.0	...	0.0	0.0	3.0	4.0	3.0	0.0	0.0	0.0	0.0	5.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2116	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
2117	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
2118	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	4.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2119	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	0.0	0.0	0.0	0.0
2120	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	0.0	0.0	0.0	0.0

2121 rows × 378 columns

Feature Engineering:

Our final_dataset has dimensions of **2121 * 378** where most of the values are sparse. We are using only a small dataset but for the original large dataset of movies which has more than **100000** features, our system may run out of computational resources when that is feed to the model.

To reduce the sparsity, we used the feature engineering techniques include the creation of a sparse matrix of user ratings and the incorporation of movie genre information. The sparse matrix is used to represent the user ratings data and is used as input to the machine learning algorithms. The movie genre information is used to incorporate movie characteristics into the recommendation system.

So, for this we have used the **csr_matrix** function from the scipy library. Applying the csr_matrix method to the dataset along with **reset_index** to set integer index to default:

In [30]:
```
# for reducing the sparsity from the data we use csr_matrix function from the scipy library.
# it creates a Compressed Sparse Row matrix representation of a 2D array.
# where each row of the matrix is stored as a sequence of indices and values, instead of storing every element of the matrix.

csr_data = csr_matrix(final_dataset.values)

# it resets the index of the final_dataset DataFrame to the default integer index and modifies the DataFrame in-place
# that is starting from 0 and increment by 1.

final_dataset.reset_index(inplace=True)
```

Now the final dataset is ready and we must perform this dataset on the machine learning algorithm model.

Developing Movie Recommendation System Model:

Here, we have used a **KNN algorithm** in a movie recommendation system. The KNN algorithm can be used in a movie recommendation system to find similar users or movies based on their ratings. In this approach, the system first creates a matrix of ratings where the rows represent users and the columns represent movies. The values in the matrix correspond to the ratings given by the users to the movies.

To generate recommendations for a particular user, the KNN algorithm first identifies the K nearest neighbors of that user based on their rating patterns. The algorithm calculates the similarity between users using a distance metric such as **Euclidean distance or cosine similarity.**

Once the nearest neighbors are identified, the algorithm recommends movies that are highly rated by the neighbors but not yet watched by the user. The recommendations can be ranked based on the average rating given by the neighbors.

We will use KNN algorithm along with cosine similarity which is very must fast compared to others.

In [31]:
```
# we will use knn algorithm along with cosine similaritty which is much better than pearson coefficient

# cosine distance is similarity measure between two vectors that measures cosine of angle between them it is used here.
# brute alogithm is used here to find the k nearest neighbors by computing the distance between each pair of points
# in the dataset.
# number of neighbors to 20 using the n_neighbors parameter.
# the number of jobs to use for parallelization to -1 using the n_jobs parameter.
# it specifies the number of CPU cores to use for parallel processing. A value of -1 means to use all available cores.

knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20, n_jobs=-1)

# Fits the NearestNeighbors model to the data using the fit() method of the knn object.
# This trains the model on the input data, which is a sparse matrix created from the final_dataset DataFrame.
knn.fit(csr_data)
```

Out[31]:
```
NearestNeighbors(algorithm='brute', metric='cosine', n_jobs=-1, n_neighbors=20)
```

We are using the scikit-learn library to create a Nearest Neighbors model using the k-Nearest Neighbors algorithm with cosine distance as the metric for measuring similarity. The algorithm used to find the nearest neighbors is the brute-force algorithm, which computes distances between all pairs of samples, and selects the k neighbors with the smallest distance.

The **"n_neighbors"** parameter is set to 20, which means that the model will return the 20 closest neighbors for each data point in the dataset. The "n_jobs" parameter is set to -1, which means that the model will use all available CPU cores to perform parallel computation for finding the nearest neighbors,

which can speed up the process when dealing with large datasets.

The data used for training the model is in the form of a sparse matrix represented as a **Compressed Sparse Row (CSR) matrix**, denoted by "csr_data". This type of matrix is a memory-efficient way of representing large matrices that contain mostly zeros, which is common in many machine learning applications.

Overall, the code is creating a Nearest Neighbors model that can be used to find the k most similar items to a given query item based on cosine similarity, using parallel processing to speed up the computation.

Making the Recommendation Function:

The working principle is very simple. We first check if the movie name input is in the database and if it is we use our recommendation system to find similar movies and sort them based on their similarity distance and output only the top 10 movies with their distances from the input movie.

Here we define a function called **get_movie_recommendation()** that takes a movie

name as an input and returns Data Frame of movie recommendations based on that input. The function utilizes a dataset of movies and their corresponding genres and ratings.

In [17]:
```
def get_movie_recommendation(movie_name):
    # it says the no of movies to recommend
    n_movies_to_reccomend = 10
    # it stores all the movies with name begin susstring of given movie name
    movie_list = movies[movies['title'].str.contains(movie_name)]

    if len(movie_list):
        movie_idx= movie_list.iloc[0]['movieId']
        movie_idx = final_dataset[final_dataset['movieId'] == movie_idx].index[0]

        # it finds all the n nearest neighbours
        distances, indices = knn.kneighbors(csr_data[movie_idx], n_neighbors=n_movies_to_reccomend+1)

        # then it sorts in ascending order of the distance and remove the gIen movie from the list
        rec_movie_indices = sorted(list(zip(indices.squeeze().tolist(), distances.squeeze().tolist())),
                                   key=lambda x: x[1],reverse=False)[1::1]
        recommend_frame = []

        # now we append all the movie name,genre ,distance to recommend frame
        for val in rec_movie_indices:
            movie_idx = final_dataset.iloc[val[0]]['movieId']
            idx = movies[movies['movieId'] == movie_idx].index
            genres = movies.iloc[idx]['genres'].values[0]
            recommend_frame.append({
                'Title': movies.iloc[idx]['title'].values[0],
                'Distance': val[1],
                'Genres': genres
            })
        df = pd.DataFrame(recommend_frame, index=range(1, n_movies_to_reccomend+1))
        return df
    else:
        return "No movies found. Please check your input"
```

The first step in the function is to define the number of movies to recommend (**n_movies_to_recommend**) which is set to 10. The function then creates a Data Frame called **movie_list** that contains all movies in the dataset that have a title containing the given input **movie_name.**

The function then checks if there are any movies in the **movie_list.** If there are, it proceeds to find the index of the given movie in the dataset (final_dataset) and then finds the n_movies_to_recommend nearest neighbors based on the distance between their ratings using a K-Nearest Neighbors (KNN) algorithm.

The distances and indices variables store the distance and index of the n_movies_to_recommend nearest neighbors to the given movie. The code then sorts the **rec_movie_indices** list in ascending order of distance, removes the given movie from the list, and then creates a list of movie recommendations called **recommend_frame.**

Each **item in recommend_frame contains the recommended movie's title, its distance from the given movie, and its genres.** The list is then converted to a DataFrame called df and returned as output.

If there are no movies in the movie_list, the function returns a string stating "No movies found. Please check your input".

Overall, this function uses a KNN algorithm to find similar movies to the given input

movie based on their ratings and returns a DataFrame of recommendations based on those similar movies.

Now we can use this function to recommend movies for the movie_name user has seen.

Finally, Let's Recommend Some Movies!!

```
In [26]: get_movie_recommendation('Iron Man')
```

Out[26]:

	Title	Distance	Genres
1	Avengers, The (2012)	0.285319	Action|Adventure|Sci-Fi|IMAX
2	Dark Knight, The (2008)	0.285835	Action|Crime|Drama|IMAX
3	WALL·E (2008)	0.298138	Adventure|Animation|Children|Romance|Sci-Fi
4	Iron Man 2 (2010)	0.307492	Action|Adventure|Sci-Fi|Thriller|IMAX
5	Avatar (2009)	0.310893	Action|Adventure|Sci-Fi|IMAX
6	Batman Begins (2005)	0.362759	Action|Crime|IMAX
7	Star Trek (2009)	0.366029	Action|Adventure|Sci-Fi|IMAX
8	Watchmen (2009)	0.368558	Action|Drama|Mystery|Sci-Fi|Thriller|IMAX
9	Guardians of the Galaxy (2014)	0.368758	Action|Adventure|Sci-Fi
10	Up (2009)	0.368857	Adventure|Animation|Children|Drama

I personally think the results are pretty good. All the movies at top are superhero or animation movies which are ideal for kids as is the input movie **"Iron Man".**

Let's try another one :

```
In [27]: get_movie_recommendation('Memento')
```

Out[27]:

	Title	Distance	Genres
1	Fight Club (1999)	0.272380	Action|Crime|Drama|Thriller
2	Lord of the Rings: The Fellowship of the Ring,...	0.316777	Adventure|Fantasy
3	Matrix, The (1999)	0.326215	Action|Sci-Fi|Thriller
4	Eternal Sunshine of the Spotless Mind (2004)	0.346196	Drama|Romance|Sci-Fi
5	Lord of the Rings: The Two Towers, The (2002)	0.348358	Adventure|Fantasy
6	Kill Bill: Vol. 1 (2003)	0.350167	Action|Crime|Thriller
7	Lord of the Rings: The Return of the King, The...	0.371622	Action|Adventure|Drama|Fantasy
8	Pulp Fiction (1994)	0.386235	Comedy|Crime|Drama|Thriller
9	American History X (1998)	0.388615	Crime|Drama
10	American Beauty (1999)	0.389346	Drama|Romance

All the movies in the top 10 are serious and mindful movies just like **"Memento"** itself, therefore I think the result, in this case, is also good.

Similarly!! We can recommend all other movies as per user's history.

Advantages:

- Movie recommendation system improves user experience by providing personalized recommendations that match their interests.
- It can help businesses to increase their revenue and customer satisfaction by providing better recommendations to their users.

- It can provide insights into user preferences and interests that can be used for marketing and advertising purposes.

Disadvantages:

- Heavy reliance on user ratings and feedback is a major disadvantage of this system, which may result in inaccurate recommendations if sufficient data is not available.
- The system may suffer from the problem of overfitting, where the recommendations become too specific to a particular user or a small group of users, leading to poor generalization.
- The system does not take into account contextual information such as the time of the day, day of the week, or current events, which may influence a user's movie preferences.

Applications:

- Movie recommendation systems can be used in various online streaming platforms such as Netflix, Amazon Prime, Hulu, etc. to provide personalized recommendations to their users.

- This system can be used to recommend similar products or items on e-commerce websites like Amazon or Flipkart.
- This project can be used as a basis for building more complex recommendation systems for various other domains.

Conclusion:

In this project, we successfully implemented **a movie recommendation system using the KNN algorithm.** We explored and learned various concepts and techniques related to machine learning, such as building a recommendation system, computing similarity scores and indexes, calculating distance between vectors using Euclidean distance and cosine similarity. We also gained valuable experience in data analysis and visualization.

References:

1) https://techvidvan.com/tutorials/movie-recommendation-system-python-machine-learning/

2) https://www.kaggle.com/code/ibtesama/getting-started-with-a-movie-recommendation-system