

**Movie Recommendation System**

**Movie Recommendation System**

**Objective:** To recommend movies to the users using content based and Collaborative filtering Methods.

**Dataset:** TMDB Movies dataset

It consists of Movies and Credits csv tables

**URL:** <https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata>

**Methodology:-**

* Movies and credits datasets are used.
* Importing required python modules
* Preprocessing the dataset
* Visualizing the data
* Recommendation based on rating and genre
* Recommendation based on Content Based Filtering
* Predicting Ratings of the movies using Collaborative Filtering.

Columns:-

Dataset 1: tmdb\_5000\_credits

* Movie\_id
* Title
* Cast
* Crew

Dataset 2: tmdb\_5000\_movies

* Budget
* Genres
* Homepage
* Id
* Keywords
* Original\_title
* Overview
* Popularity
* Production\_companies
* Production\_countries
* Release\_date
* Revenue
* Runtime
* Spoken\_language
* Status
* Tagline
* Title
* Vote\_average
* Vote\_count

**Scope of the Project**

The objective of this project is to provide accurate movie recommendations to users. The goal of the project is to improve the quality of movie recommendation system, such as accuracy, quality and scalability of system than the pure approaches. This is done using the content based filtering and collaborative filtering methods. There is a huge scope of exploration in this field for improving scalability, accuracy and quality of movie recommendation systems. Movie Recommendation system is very powerful and important system. But, due to the problems associated with pure collaborative approach, movie recommendation systems also suffers with poor recommendation quality and scalability issues.

**Agile Methodology**:

1**.Collecting the data sets**: Collecting all the required data set from Kaggle web site.in this project we require *tmdb\_5000\_credits.csv* and *tmdb\_5000\_movies.csv*

2.**Data Analysis**: make sure that that the collected data sets are correct and analysing the data in the csv files. i.e. checking whether all the column Felds are present in the data sets.

3.**Algorithms:** in our project we have only two algorithms one is cosine similarity and other is KNN are used to build the machine learning recommendation model.

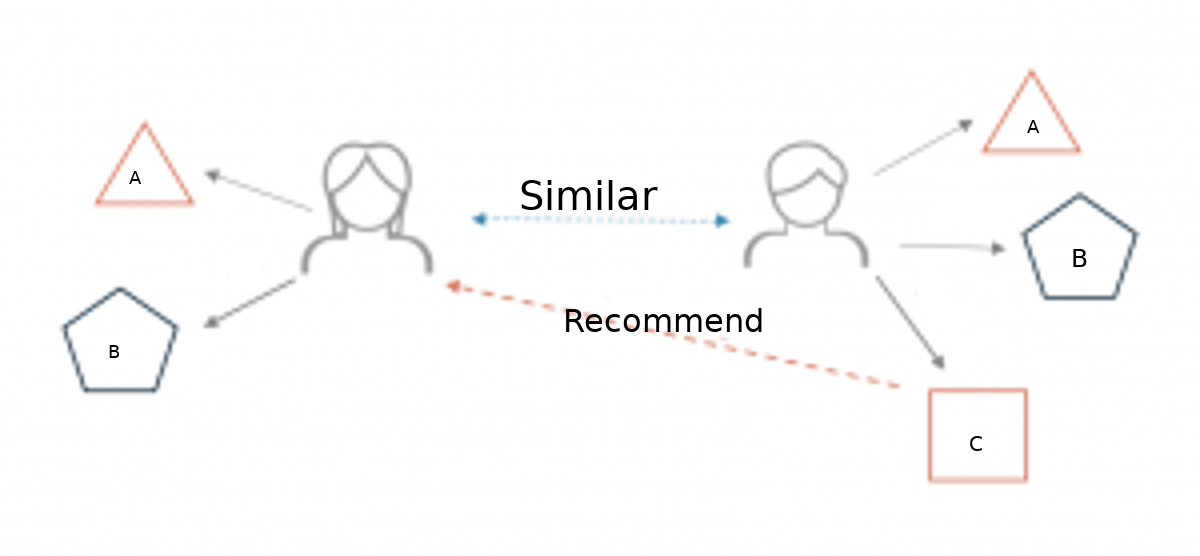
4.**Training and Testing the model**: once the implementation of algorithm is completed . we have to train the model to get the result. We have tested it several times the model is recommend different set of movies to different users.

5.**Improvements in the project**: In the later stage we can implement different algorithms and methods for better recommendation.

**What is a recommender system?**

A recommender system is a simple algorithm whose aim is to provide the most relevant information to a user by discovering patterns in a dataset. The algorithm rates the items and shows the user the items that they would rate highly. An example of recommendation in action is when you visit Amazon and you notice that some items are being recommended to you or when Netflix recommends certain movies to you. They are also used by Music streaming applications such as Spotify and Youtube to recommend music that you might like.

Below is a very simple illustration of how recommender systems work in the context of an e-commerce site.

****

Two users buy the same items A and B from an e-commerce store. When this happens the similarity index of these two users is computed. Depending on the score the system can recommend item C to the other user because it detects that those two users are similar in terms of the items they purchase.

**Different types of recommendation engines**

The most common types of recommendation systems are content-based and collaborative filtering recommender systems. In collaborative filtering, the behavior of a group of users is used to make recommendations to other users. The recommendation is based on the preference of other users. A simple example would be recommending a movie to a user based on the fact that their friend liked the movie. There are two types of collaborative models Memory-based methods and Model-based methods. The advantage of memory-based techniques is that they are simple to implement and the resulting recommendations are often easy to explain. They are divided into three:

**Content-based systems**

These filtering methods are based on the description of an item and a profile of the user’s preferred choices. In a content-based recommendation system, keywords are used to describe the items, besides, a user profile is built to state the type of item this user likes. In other words, the algorithms try to recommend products that are similar to the ones that a user has liked in the past.**Content-based systems are based on the idea that if you liked a certain item you are most likely to like something that is similar to it.**

**Collaborative-based systems**

In Collaborative Filtering, we tend to find similar users and recommend what similar users like. In this type of recommendation system, we don’t use the features of the item to recommend it, rather we classify the users into the clusters of similar types, and recommend each user according to the preference of its cluster.

**Hybrid Recommendation Systems**

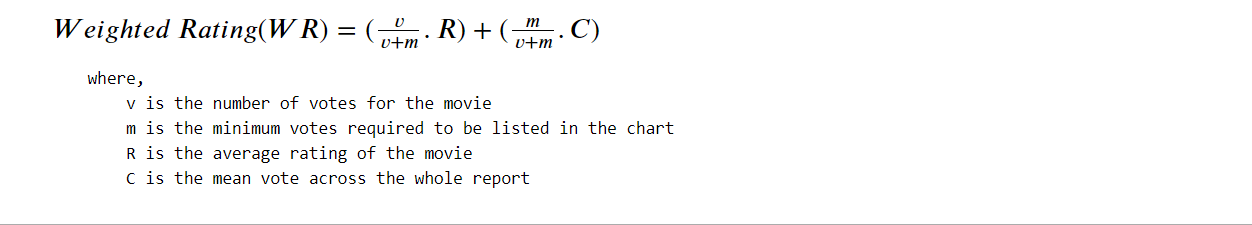
Hybrid recommendations combines both content based and collaborative filterin algorithms. Hybrid approaches can be implemented in several ways, by making content-based and collaborative-based predictions separately and then combining them, by adding content-based capabilities to a collaborative-based approach (and vice versa), or by unifying the approaches into one model.

**IMPLEMENTATION**

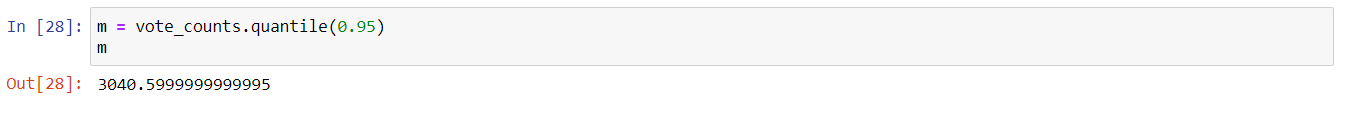
**Simple Recommendation System:**

**Recommending the top 15 movies based on popularity and ratings.**

* The Simple Recommender offers **generalized recommendations** to every user **based on movie popularity and (sometimes) genre**.
* The **basic idea** behind this recommender is that **movies that are more popular and more critically acclaimed will have a higher probability of being liked by the average audience.**
* This model **does not give personalized recommendations** based on the user.

****

* The next step, we need to determine an appropriate value for m, the minimum votes required to be listed in the chart.
* We will use 95th percentile as our cutoff. In other words, for a movie to feature in the charts, it must have more votes than at least 95% of the movies in the list.

****

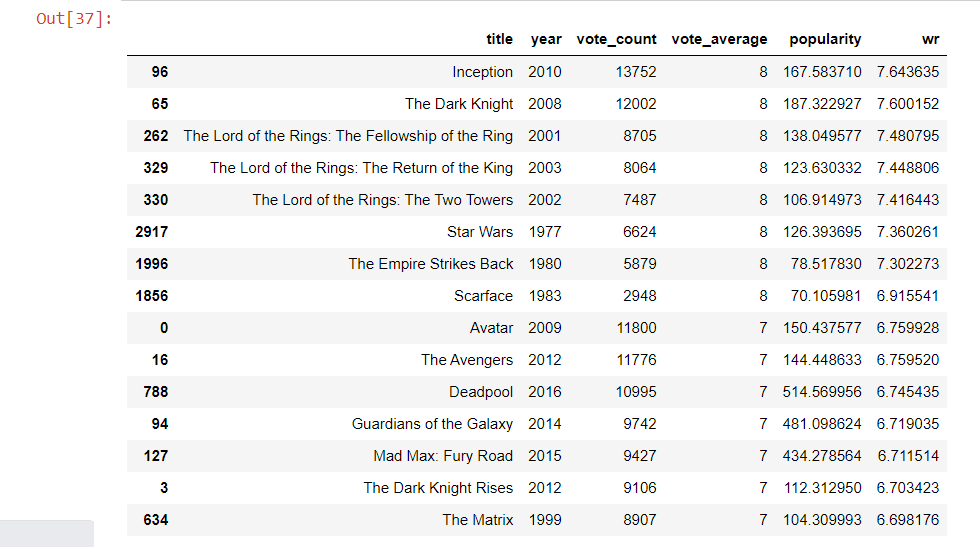
We have categorized the movies as **qualified** if it has the vote count greater than 3040,

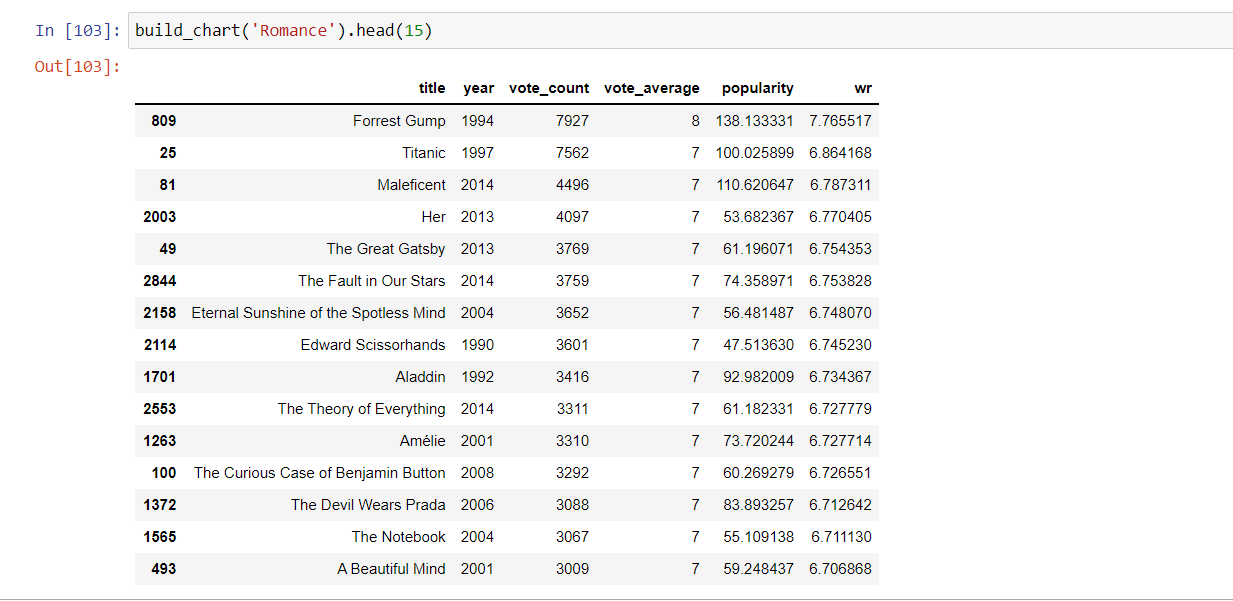
which is the 95th percentile.



Here we have defined a function which will accept the genre as a parameter and will be displaying the top 15 movies based on popularity.







**Conclusion:** **• The Simple Recommender provides every user with generalised suggestions based on the popularity and (sometimes) genre of movies.**

**• The fundamental tenet of this recommender is that more well-known and highly acclaimed films are more likely to be enjoyed by the general public.**

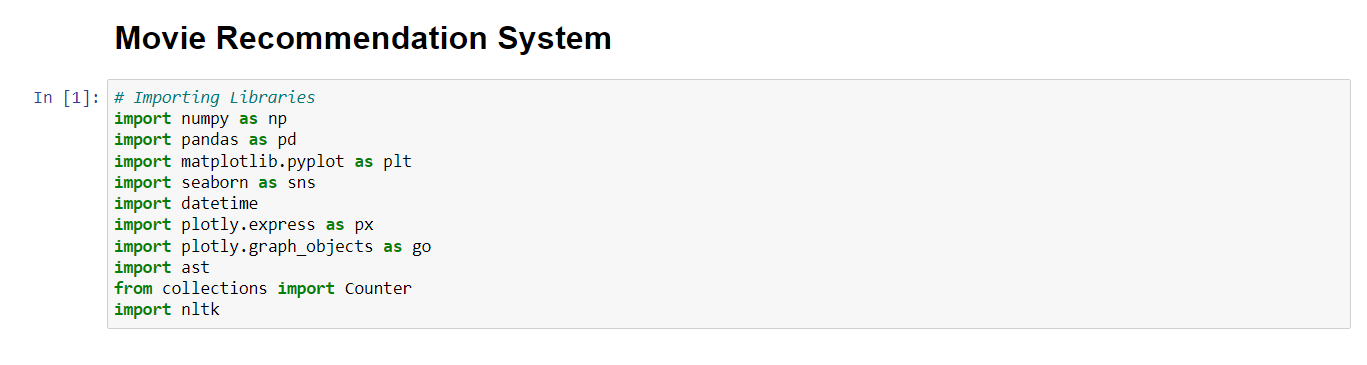
**This model does not provide user-specific recommendations.**

**This type of recommendation can be applied to Netflix, Prime, and other OTT services to provide users with the most well-liked material.**

***RECOMMENDATION MODELS***

1. **Content Based Recommendation System-**Using cosine similarity

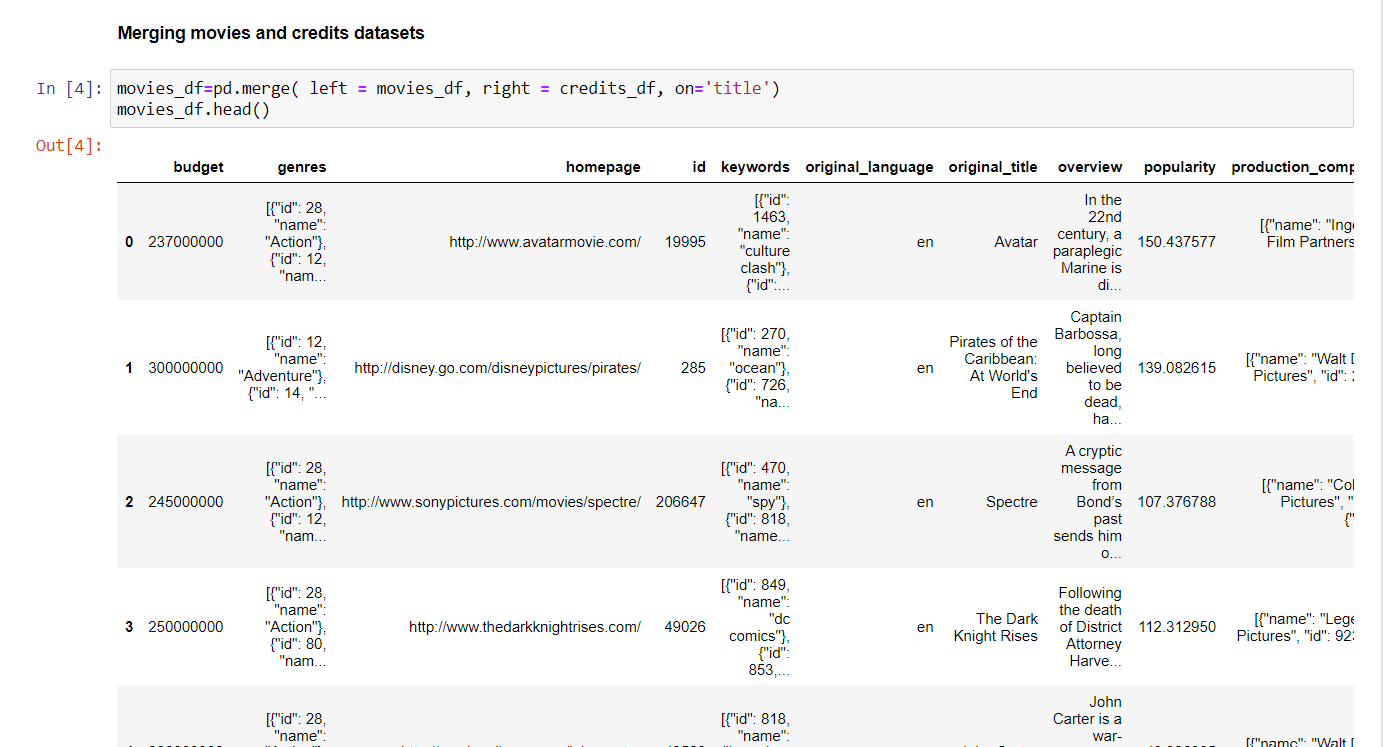
* Step 1: Importing the necessary modules



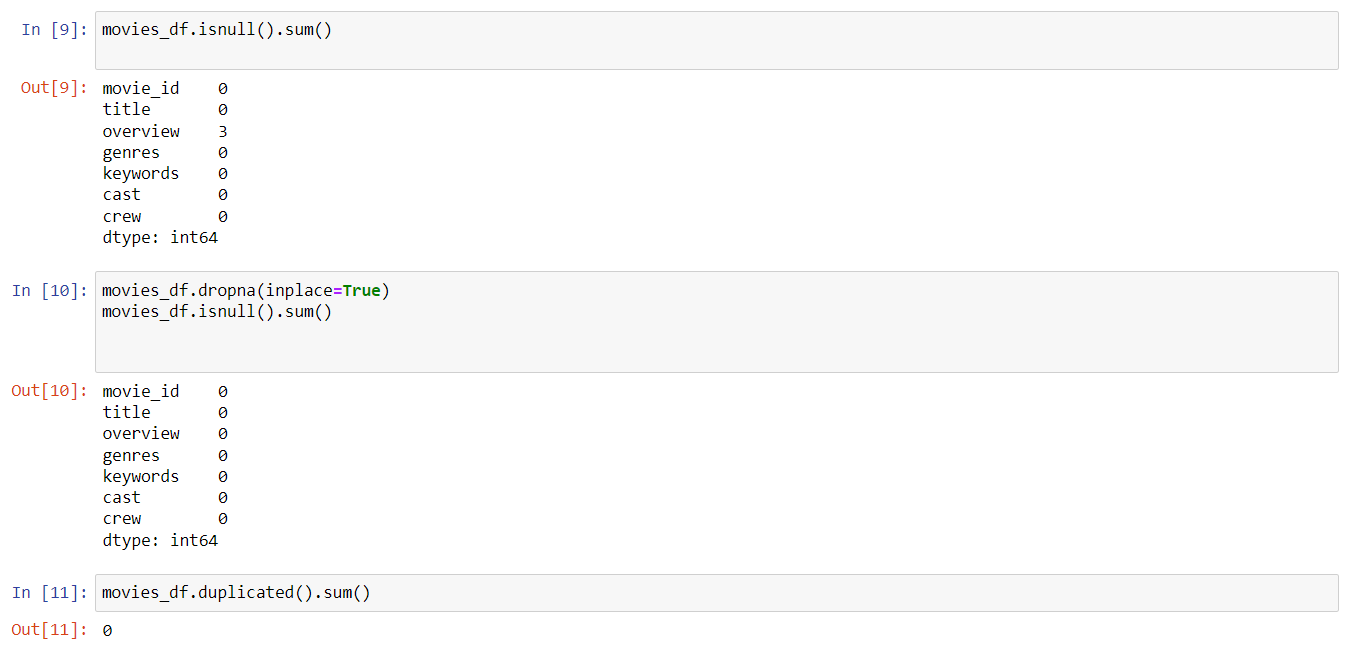
* Step 2: Reading the dataset.



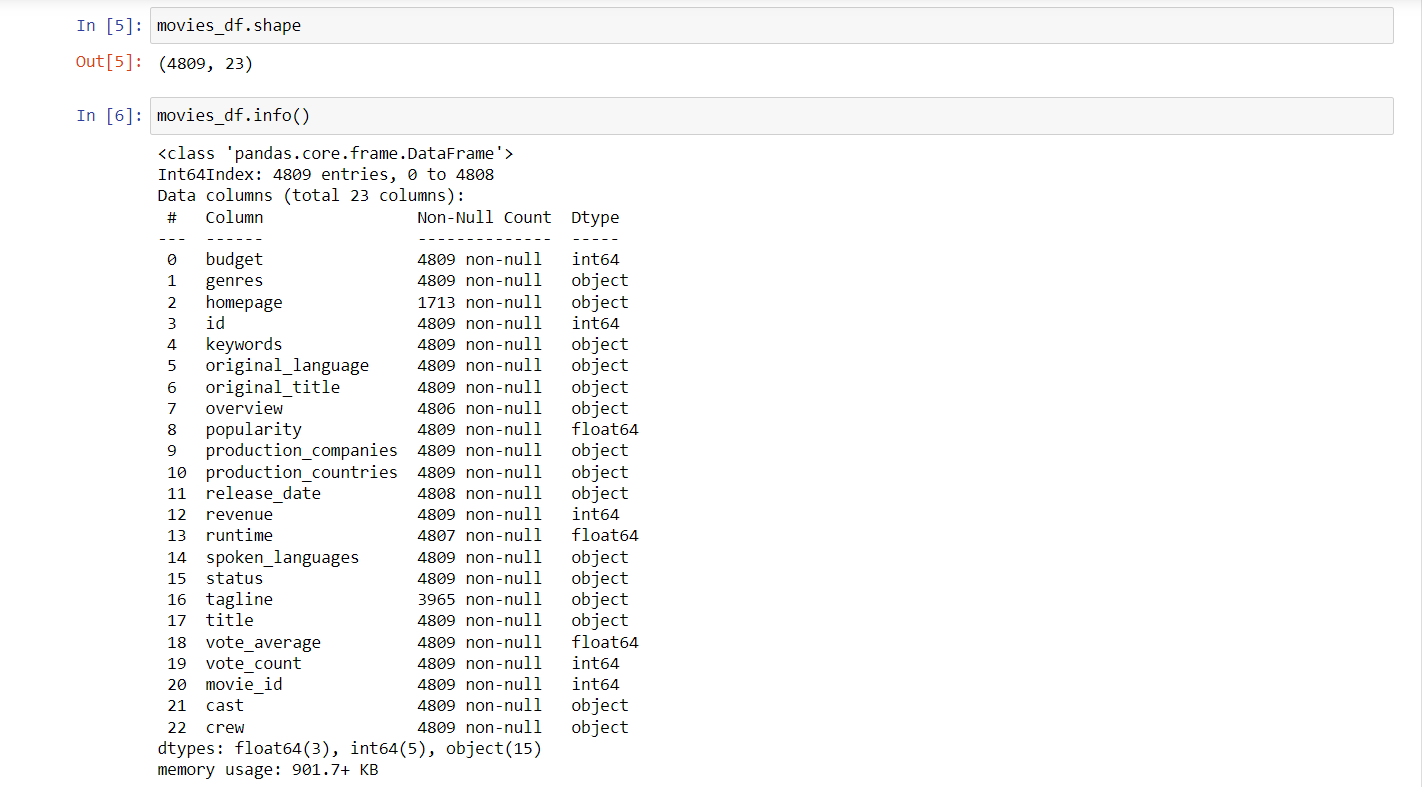
* Step 3: Merging movies and credits dataset.

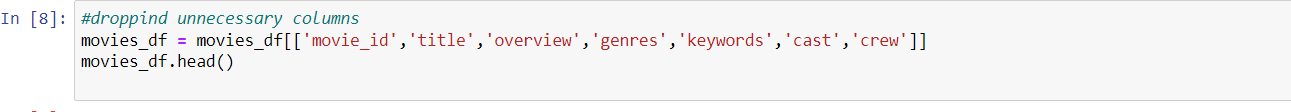


* Step 4: Data Cleaning



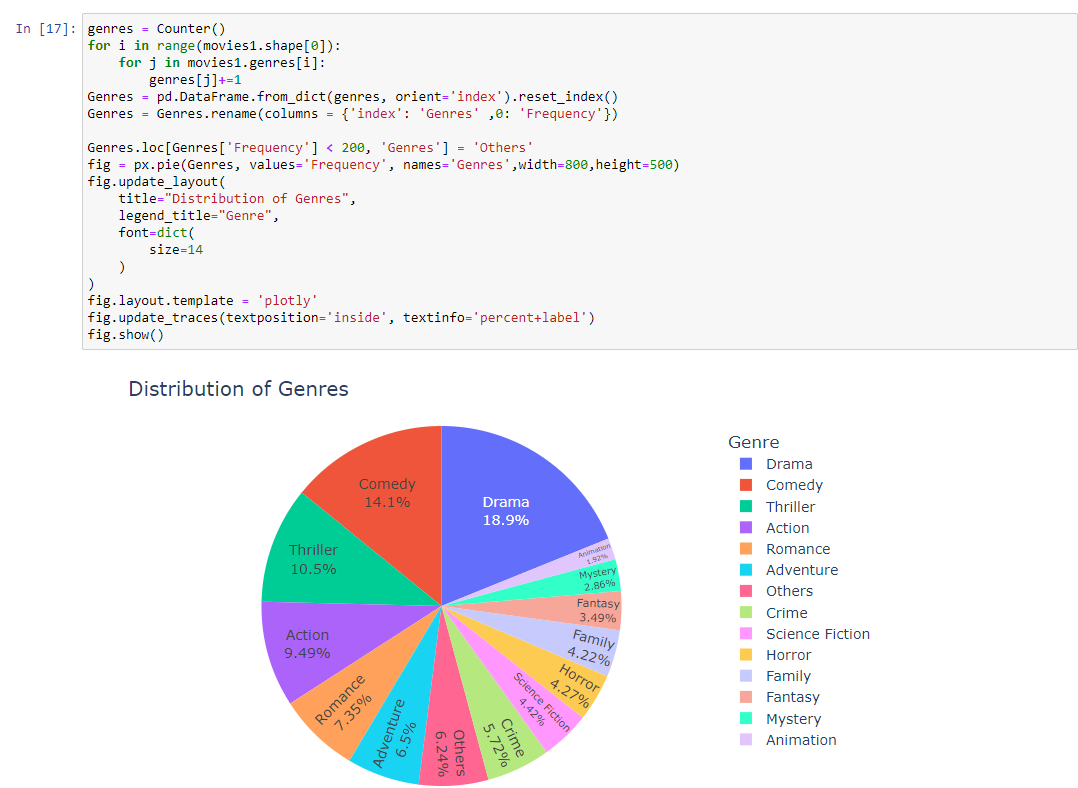
* Step 5: Data Preprocessing

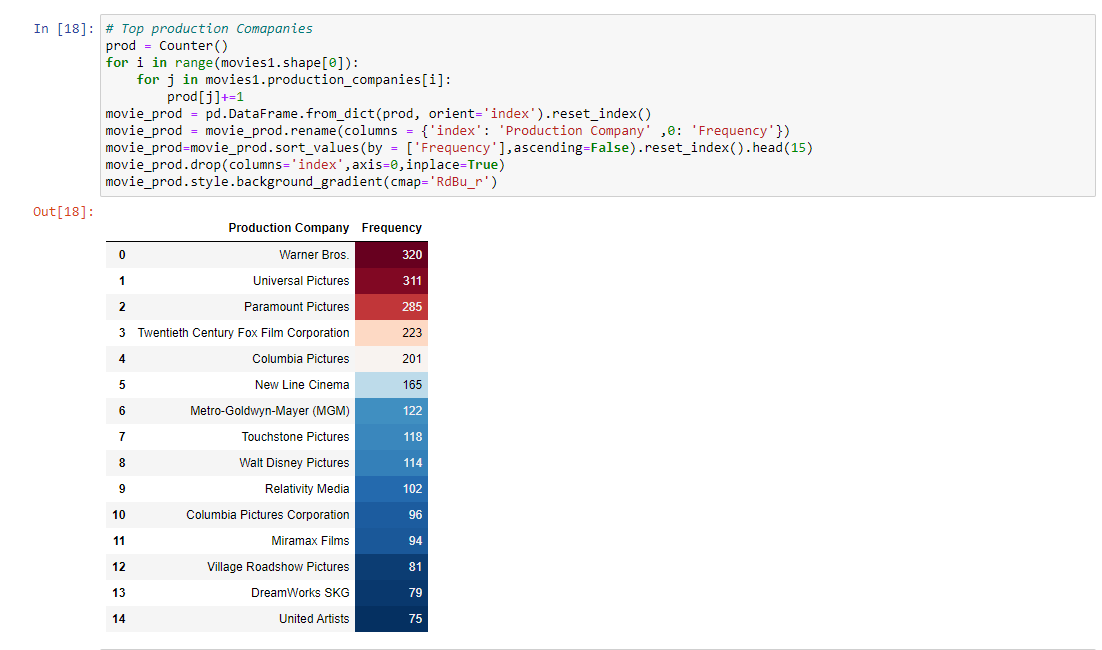


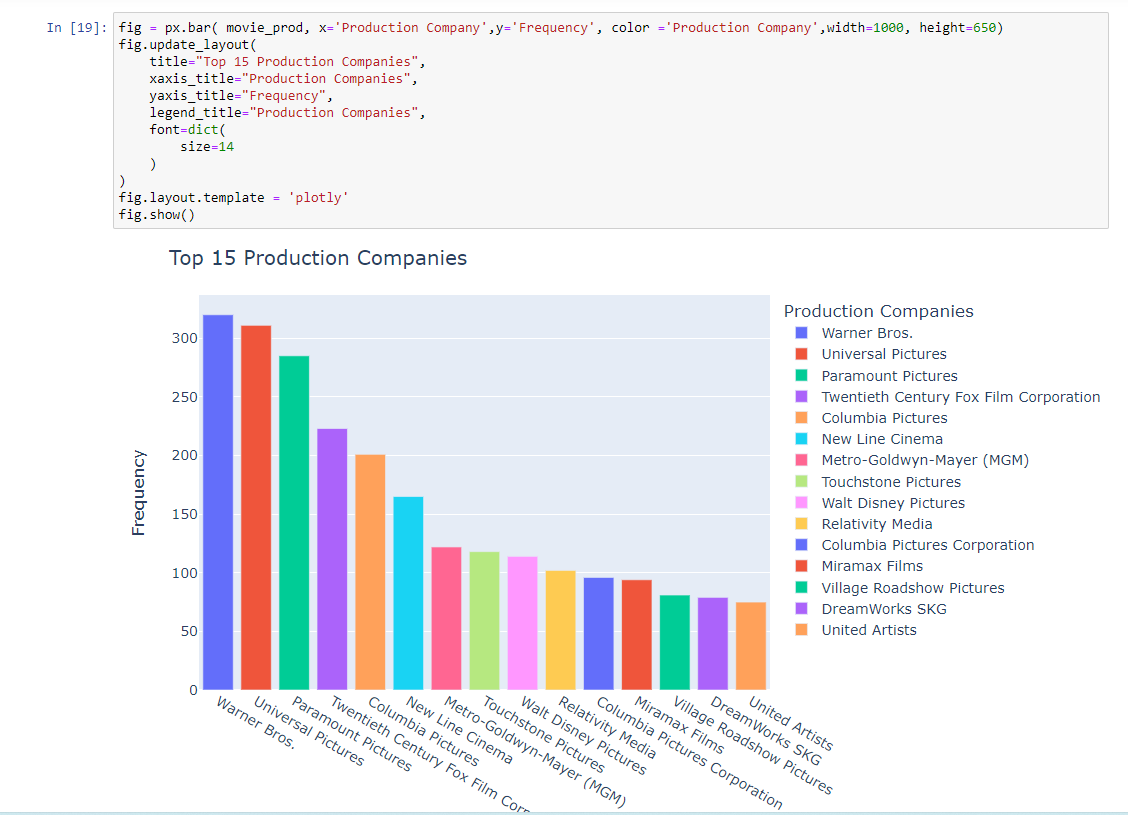


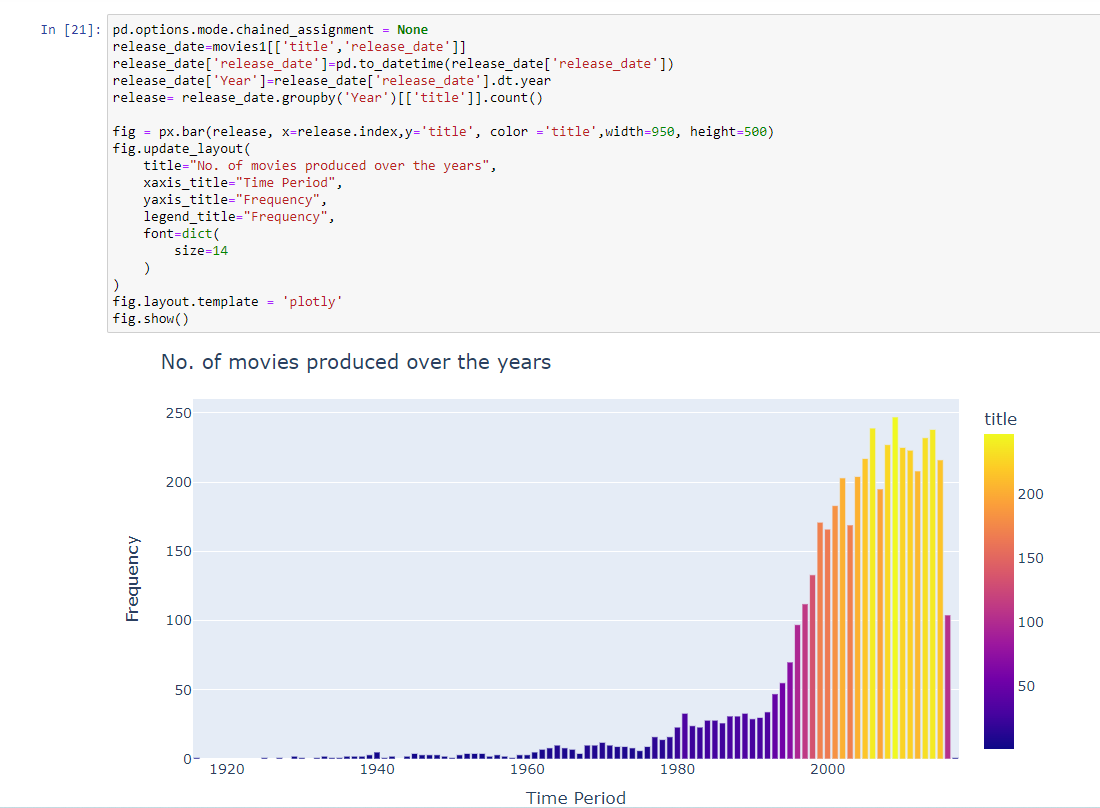


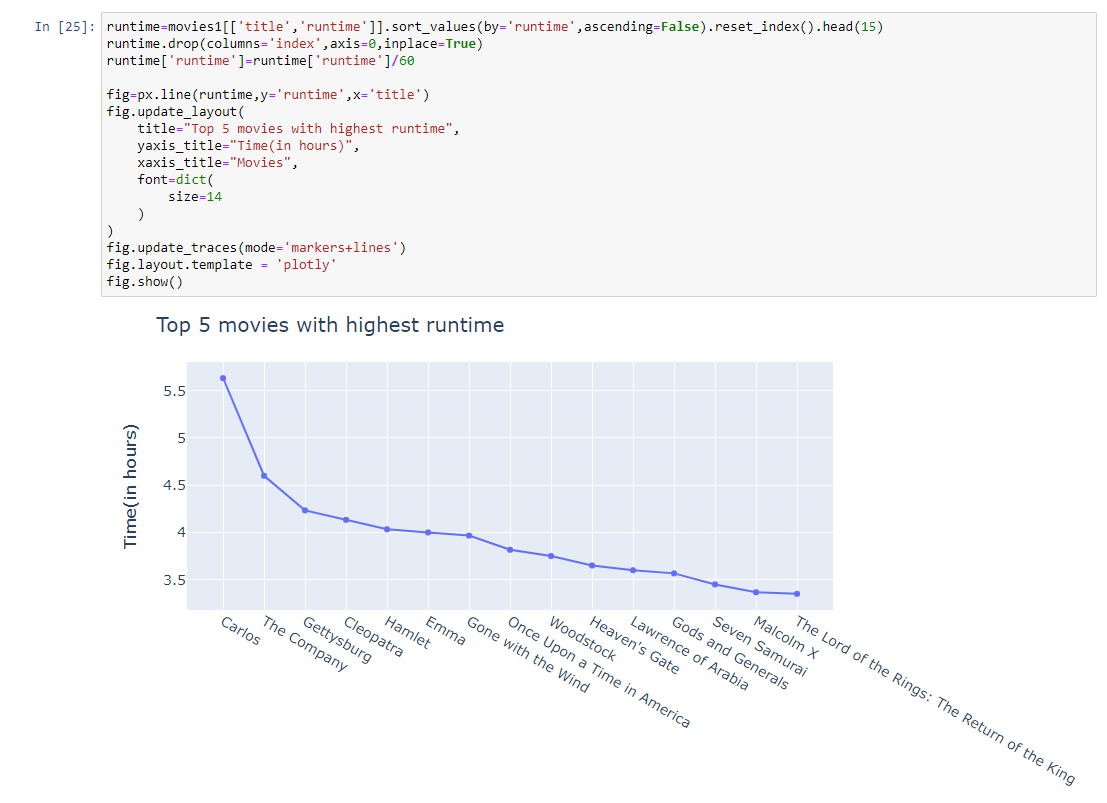
* Step 6: EDA





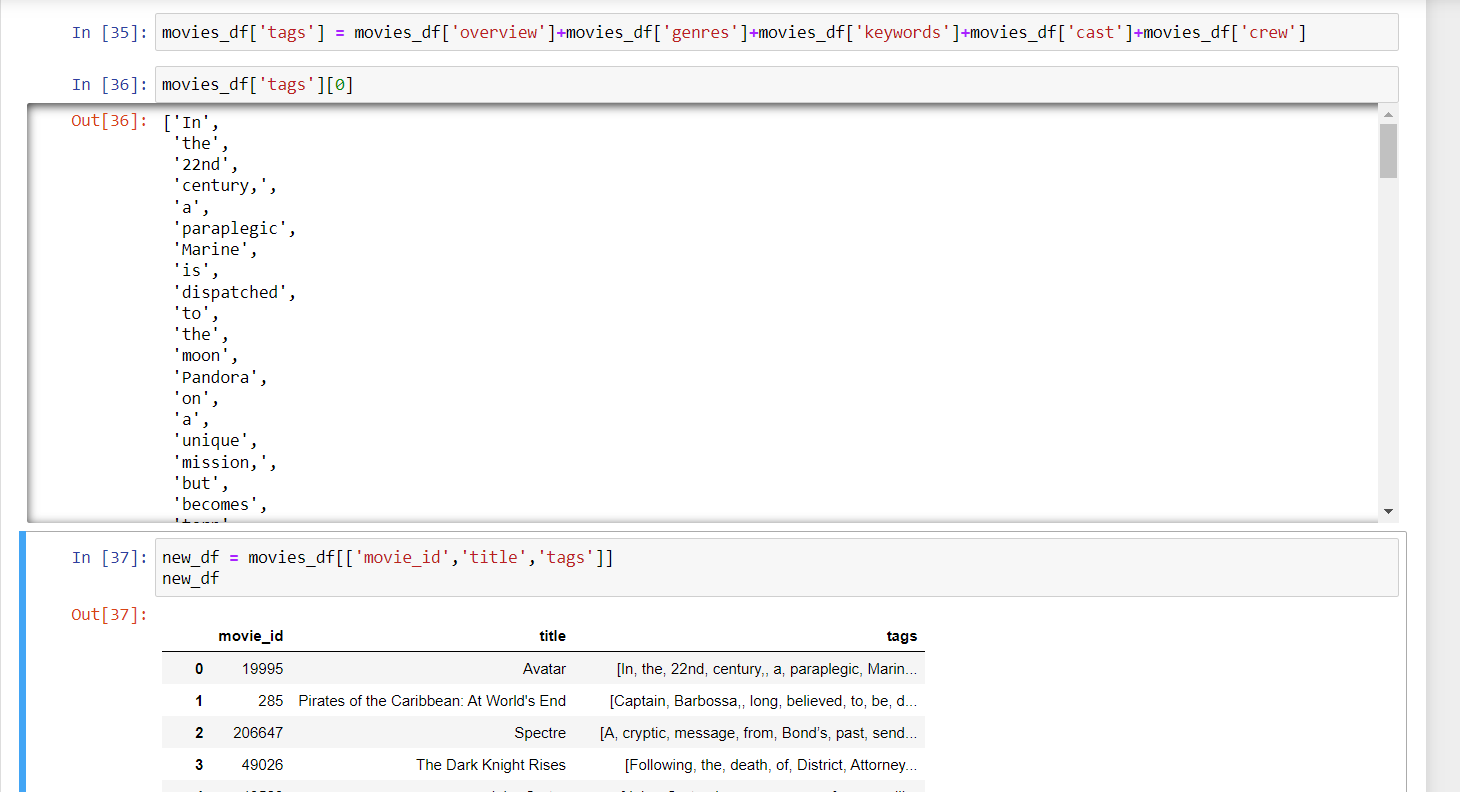




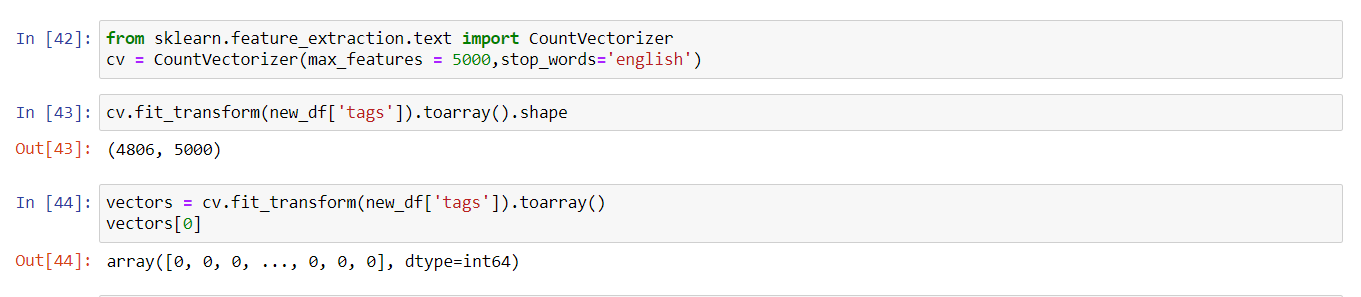


* Step 7: A function named *convert* is created which is used to convert the columns from JSON format to string.



* Step 8: A new column named **tags** is created which contains each and every word from the other columns.
* Step 9: Feature Extraction is done by importing the **CountVectorizer** tool from scikit learn library

This tool basically converts a given text into a Vector based on the frequency of each word that occurs.



All the words in the column ‘*tags’* are converted into vectors.

The fit\_transform function transforms each token to a specific position in the output function.

Then obtained vectors are converted into array using the toarray() function.

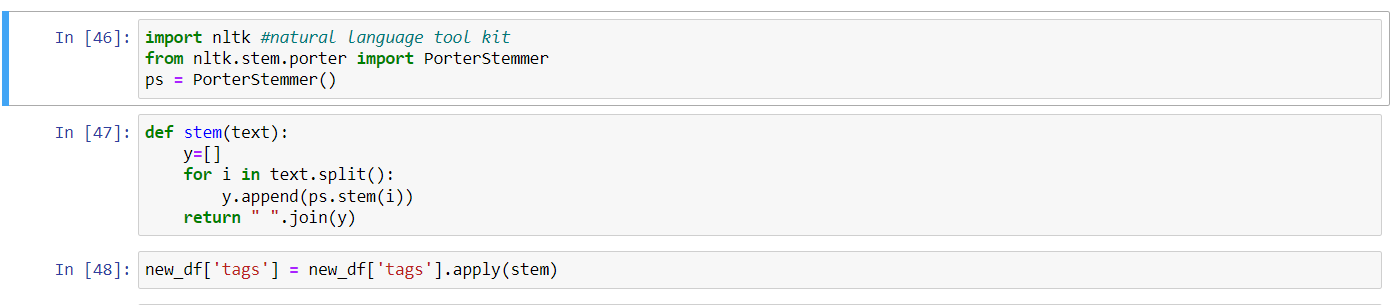
* Step 10: **nltk(natural language tool kit)** library is imported for text processing. From the nltk library **PorterStemmer**  imported which is used for stemming.

Stemming is **the process of reducing a word to its word stem that affixes to suffixes and prefixes or to the roots of words known as a lemma.**



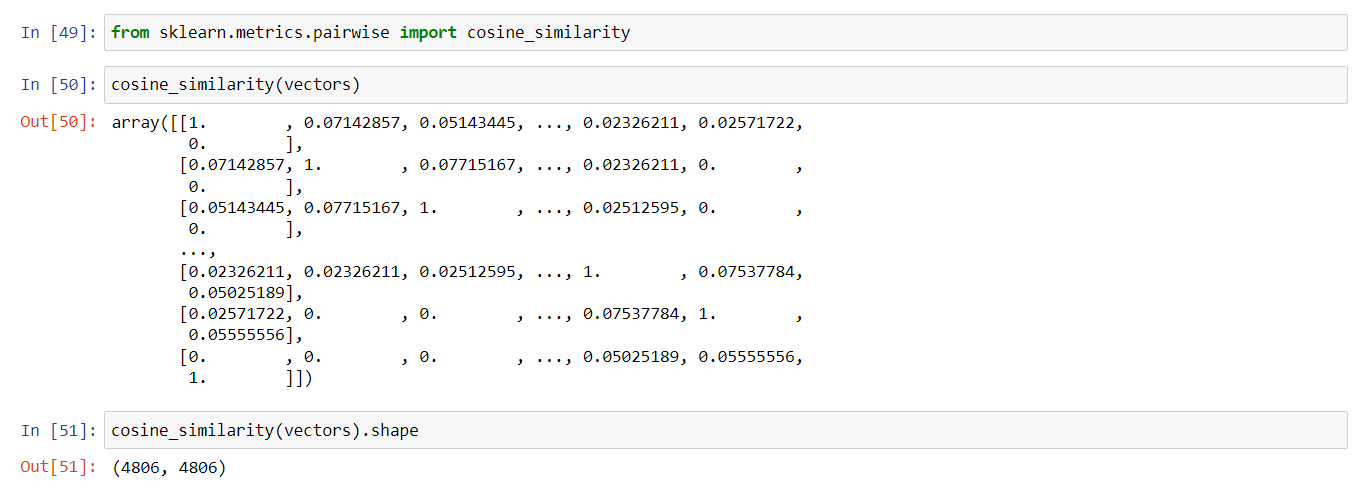
* Step 11: Now we need to apply stemming to each and every word in the tag column. For this we have created a function named ***stem***.

This function is applied to each and every word in the stem.



* Step 12: Cosine similarity tool is imported from sklearn.

***Cosine Similarity:*** measures the similarity between two vectors. It calculates the dot product of the vectors and depending on the values it determines whether the two vectors are pointing roughly in the same direction.



Step 13: The cosine similarity function is applied to all the vectors. This function measures the similarity between all the vectors pairwise.



Now the recommend function is created which applies the cosine similarity function to all the vectors (i.e the vectors which was created from the tags column).

The similarity values are converted into a list and sorted in descending order. From the sorted list we will be selecting only the top 6 values.



CONCLUSION:

**The model doesn't need any data about other users, since the recommendations are specific to this user. This makes it easier to scale to a large number of users.The model can capture the specific interests of a user, and can recommend niche items that very few other users are interested in.**

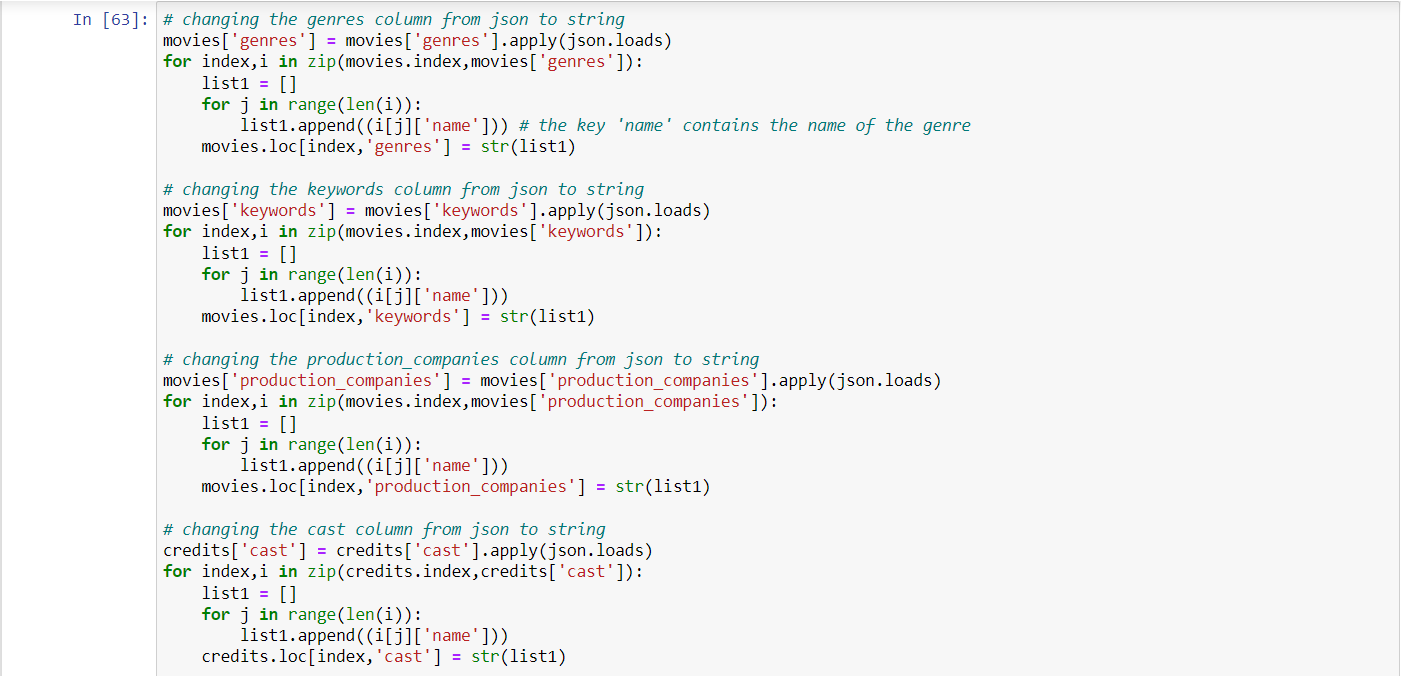
1. **Collaborative Filtering-**Using KNN (K-Nearest Neighbour)

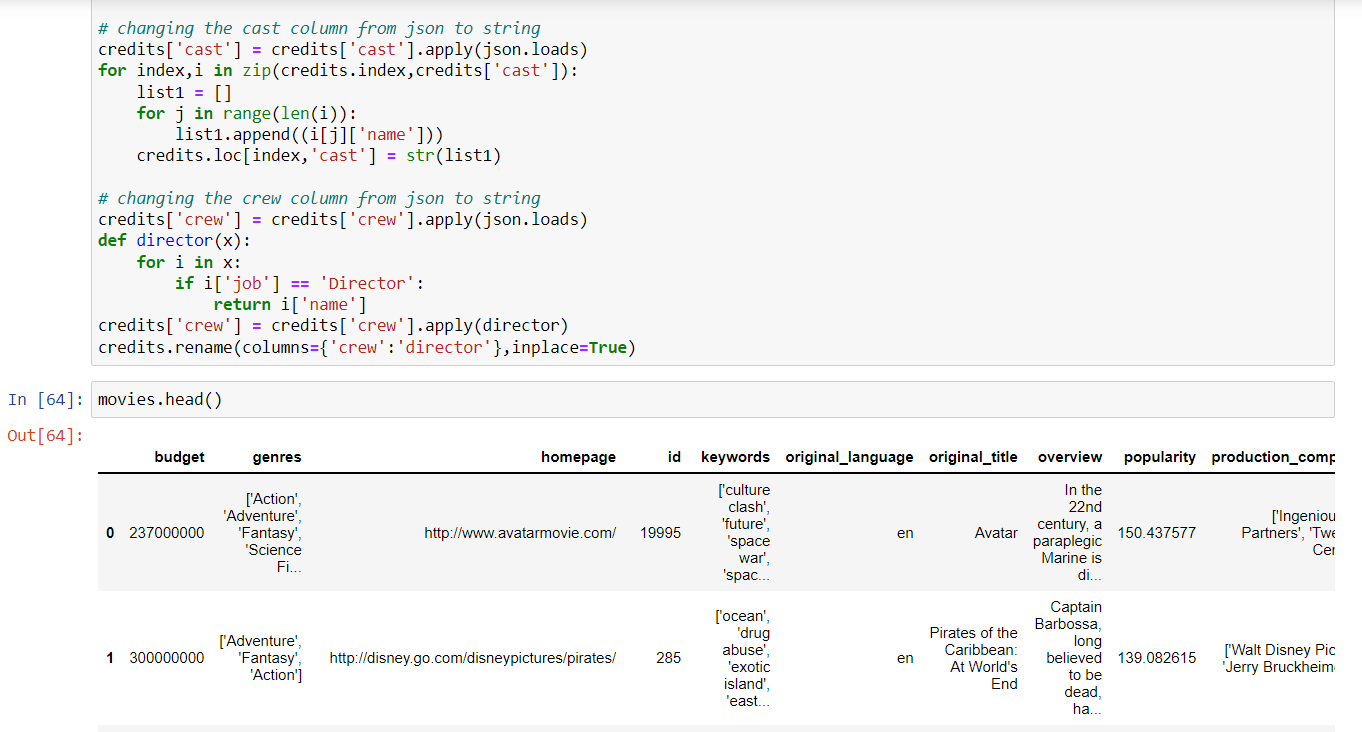
**Item-Item Collaborative Filtering**

 Here, we explore the relationship between the pair of items (the user who bought Y, also bought Z). We find the missing rating with the help of the ratings given to the other items by the user.

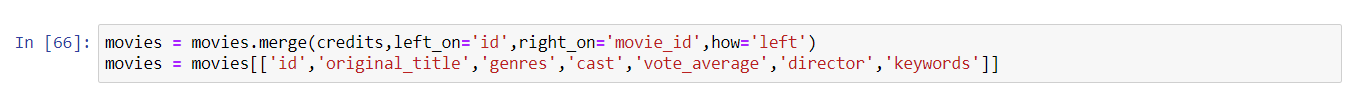
* Step 1: **Data conversion from JSON to String**

 We can see that genres, keywords, production\_companies, production\_countries, spoken\_languages are in the JSON format. Similarly in the other CSV file, cast and crew are in the JSON format. Now let’s convert these columns into a format that can be easily read and interpreted. We will convert them into strings and later convert them into lists for easier interpretation.

****

****

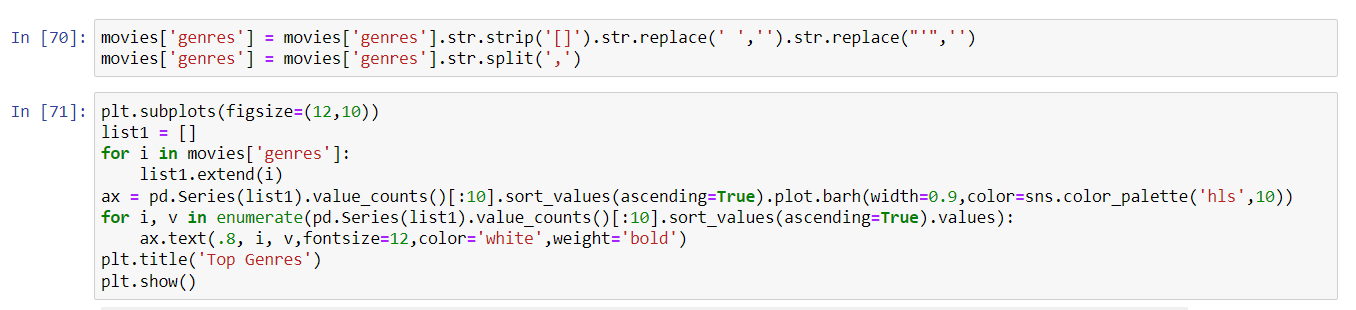
* Step 2: **Merging the two csv files**

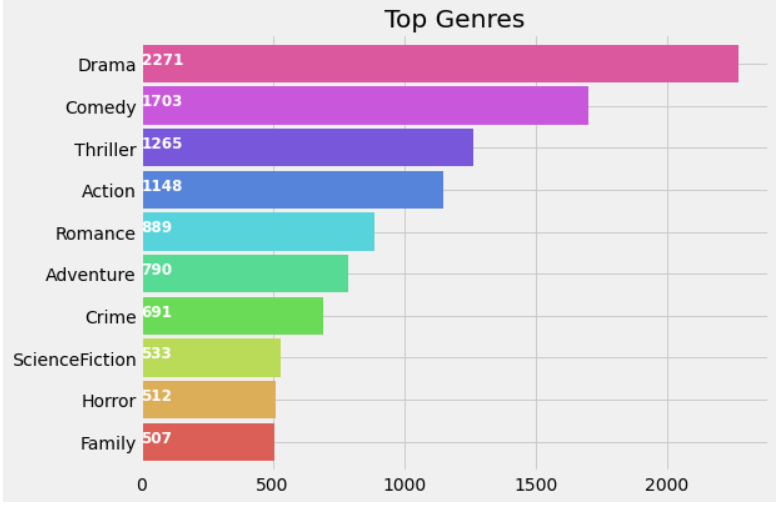
****

* Step 3:**Working with Genres column**

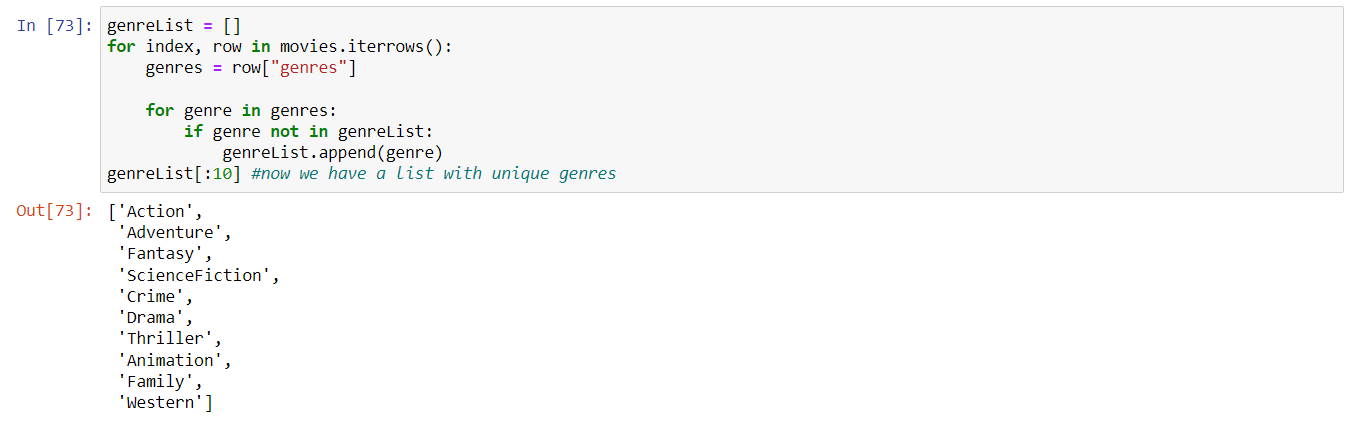
We will clean the genre column to find the genre\_list and plot them.

Genre\_list consists of all the unique genres.



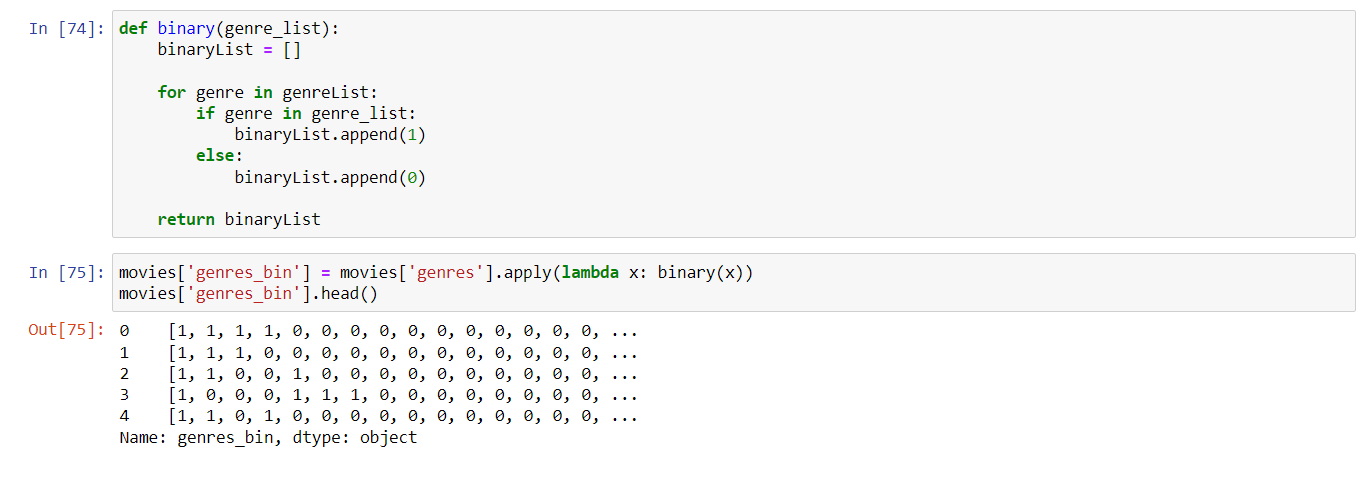


A new list is created which has all the unique genre names.



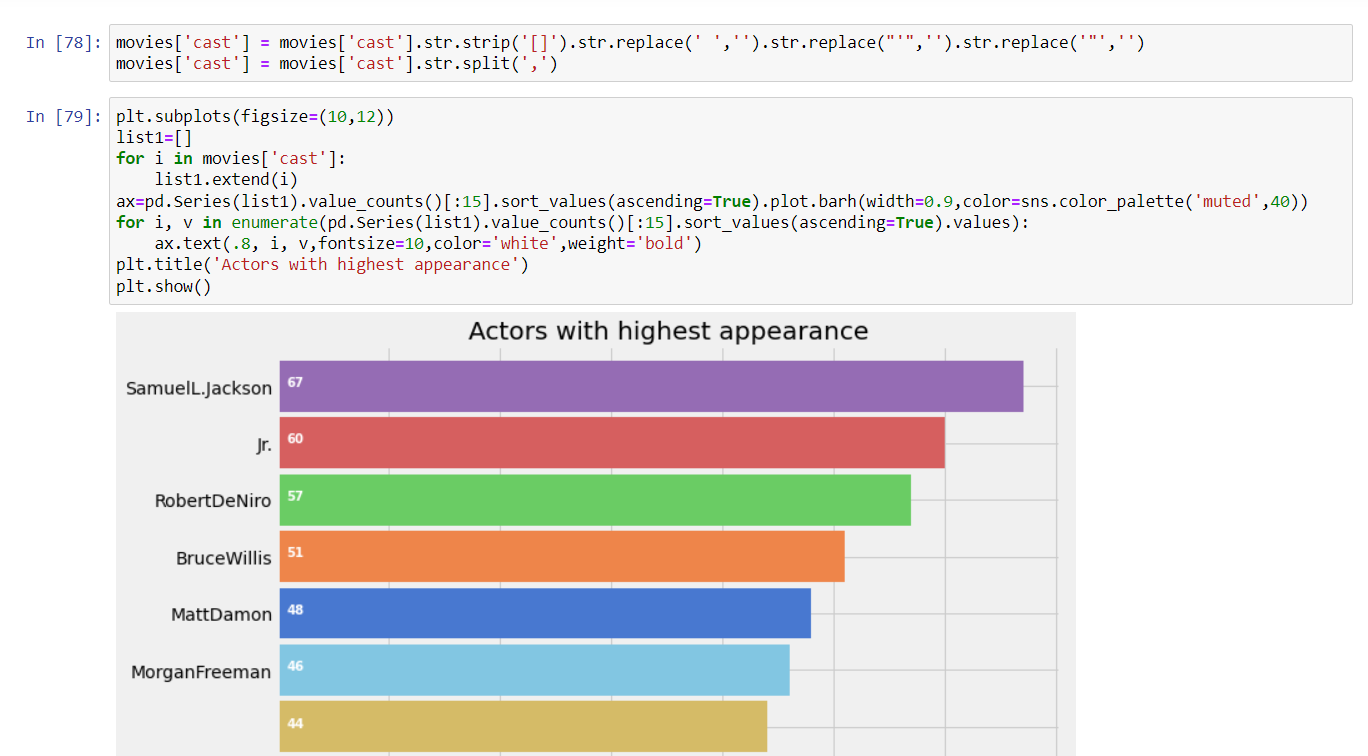
‘genreList’ will now hold all the genres. But how do we come to know about the genres each movie falls into. Now some movies will be ‘Action’, some will be ‘Action, Adventure’, etc. We need to classify the movies according to their genres. We have created a new column in the dataframe that will hold the binary values whether a genre is present or not in it. First, let’s create a method that will return back a list of binary values for the genres of each movie. The ‘genreList’ will be useful now to compare against the values.

**The same procedure will be applied to the cast,director and the keywords column.**



* Step 4: **Working with Cast column**

Let’s plot a graph of Actors with Highest Appearances

****

 I have selected the main 4 actors from each movie.

* Step 5: **Working with director’s column**

Let’s plot Directors with maximum movies



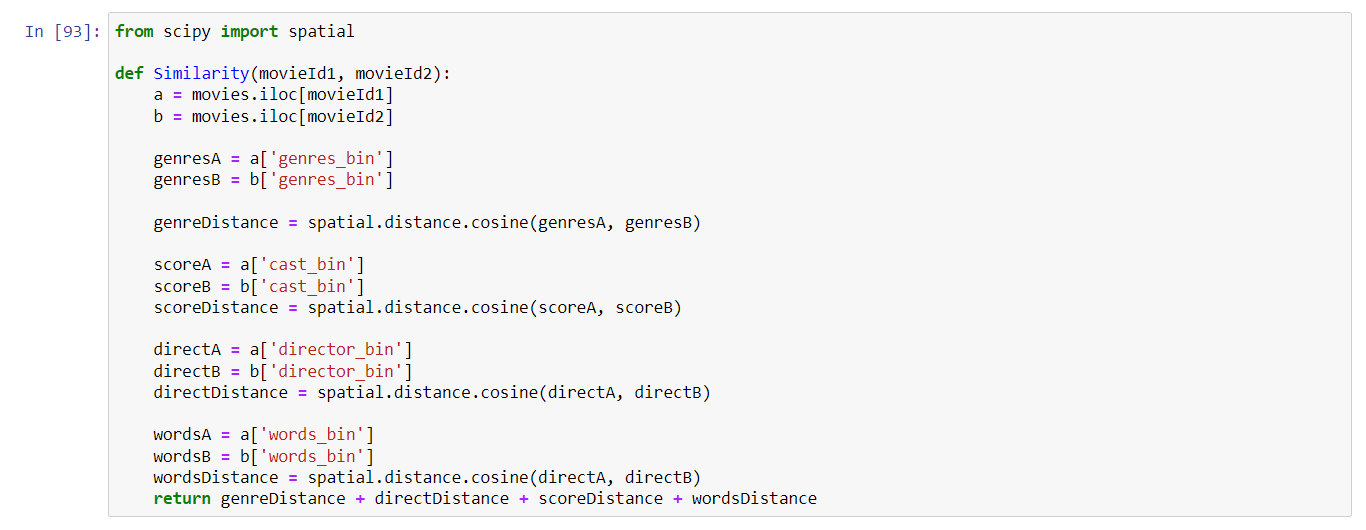
* Step 6: **Working with the keywords column**



* Step 7: **Similarity Between movies**

We will be using **Cosine Similarity** for finding the similarity between 2 movies.

I have defined a function Similarity, which will check the similarity between the movies.



* Step 8: **Score Predictor**

The main function working under the hood will be the **Similarity()** function, which will calculate the similarity between movies, and will find 10 most similar movies. These 10 movies will help in predicting the score for our desired movie. We will take the average of the scores of similar movies and find the score for the desired movie.

Here, We have arbitrarily chosen the value K=10.

A small value of K means that noise will have a higher influence on the result and a large value make it computationally expensive.



Now when the ***predict\_score*** function is called with a movie name as a parameter it will display 10 other similar movies and their predicted ratings.

Predicted ratings will be nothing but the average of the ratings of the 10 movies which are recommended.



**CONCLUSION:**

**The KNN algorithm is implemented in the collaborative based model along with the principle of cosine similarity as it gives more accuracy than the other distance metrics with additional low complexity advantage. KNN makes use of multiple attributes to filter the similar results and increase accuracy. There is a scope for improvement in recommendation.**