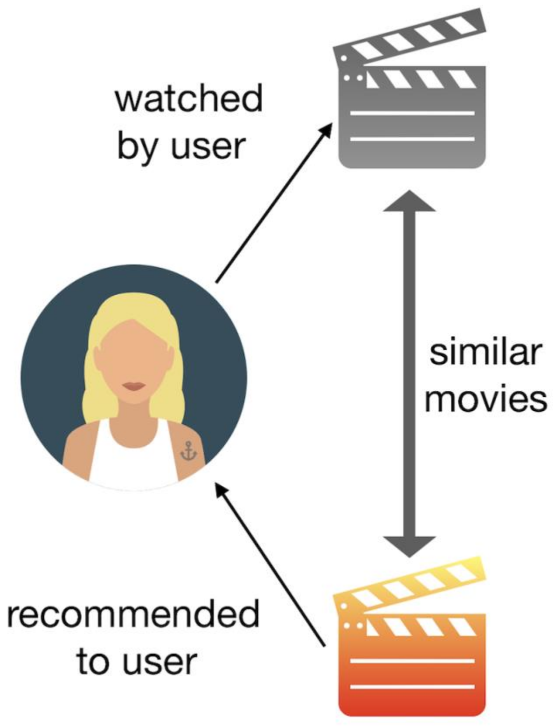
Content Based Movie Recommender System – A Basic Approach

The hot summer and Covid-19 lockdown had brought outdoor life to a standstill and brought families of all household glued to digital world. With the dominance of technology, there were several changes in Entertainment industry, but the growth of online streaming has been redoubtable. The Content Based Movie Recommender System is a step up approach to recommended movies to user based on their interest. It computes by using user's preferred movie as input and analyses its contents like storyline, genre, cast, and director etc. to find other movies of similar content by ranking it using similarity scores.

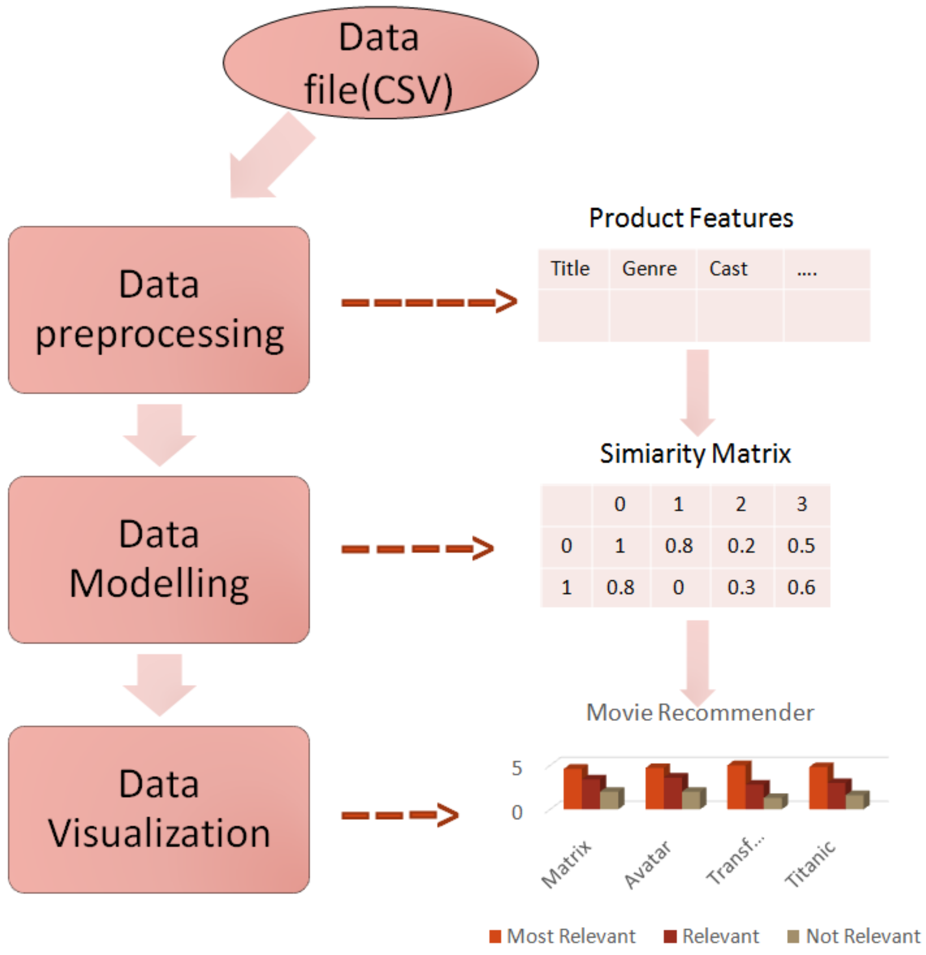


A Sneak-Peak into Recommender System

A Content-based recommendation system tries to recommend items to users based on their profile. The user's profile revolves around that user's preferences and tastes. It is shaped based on user ratings, including the number of times that user has clicked on different items or perhaps even liked those items.

The Implementation

There are different machine learning process and languages to implement the recommender system , but we have considered the **Python language** and **Similarity Matrix modelling process** to implement our system.



The Process

Step 1:

The first step in the process of implementation of this project is gathering of data. Our data is stored in a csv file.

Step 2:

In the next stage we import the required libraries used for the process.

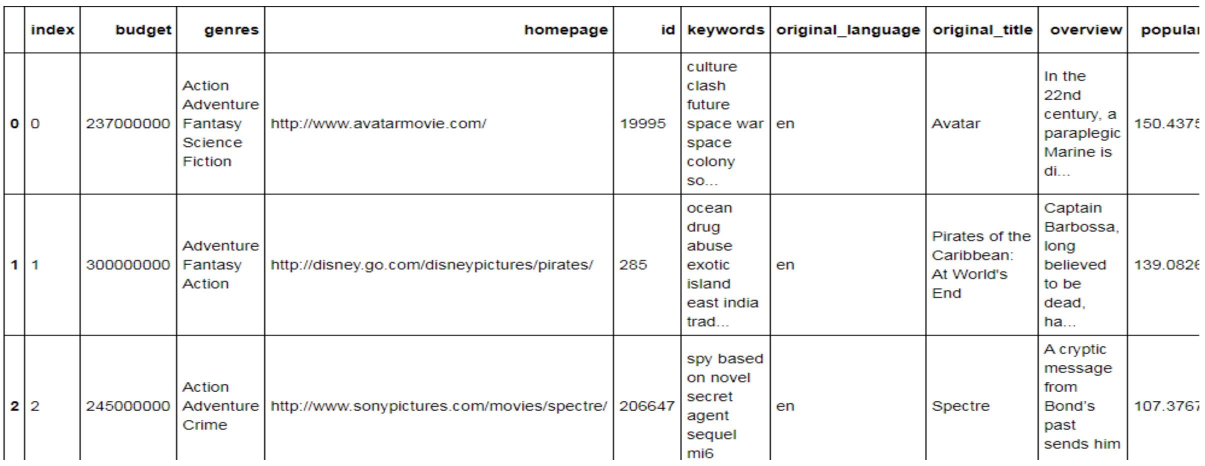
```
import pandas as pd
import numpy as np
```

Step 3:

Read the data file using pandas.

```
df= pd.read_csv('movie_dataset.csv')
df.head()
```

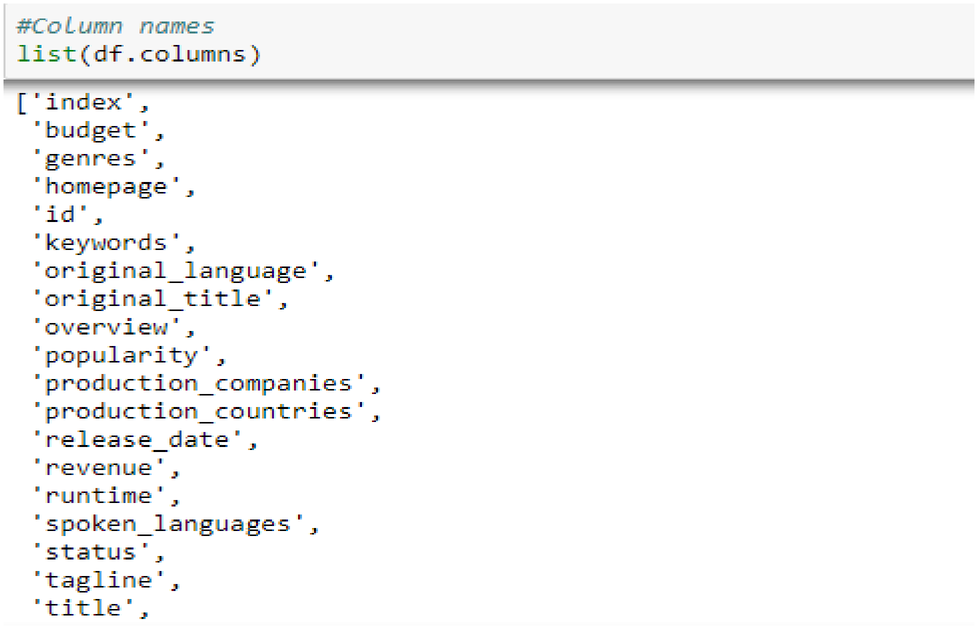
The file is read into the data frame ‘df’ and df.head () displays rows of the dataset.



	index	budget	genres	homepage	id	keywords	original_language	original_title	overview	popular
0	0	237000000	Action Adventure Fantasy Science Fiction	http://www.avatarmovie.com/	19995	culture clash future space war space colony so...	en	Avatar	In the 22nd century, a paraplegic Marine is di...	150.4375
1	1	300000000	Adventure Fantasy Action	http://disney.go.com/disneypictures/pirates/	285	ocean drug abuse exotic island east india trad...	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	139.0826
2	2	245000000	Action Adventure Crime	http://www.sonypictures.com/movies/spectre/	206647	spy based on novel secret agent sequel mi6	en	Spectre	A cryptic message from Bond's past sends him	107.3767

Step 4:

The column names is displayed using list(df.columns).

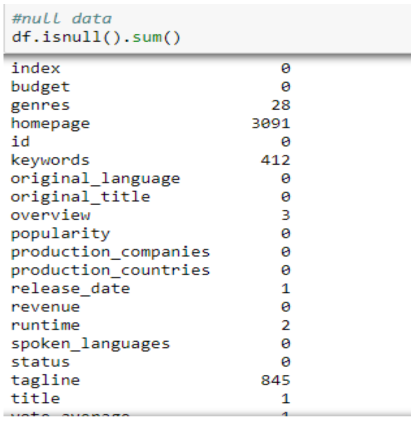


```
#Column names
list(df.columns)
```

```
['index',
 'budget',
 'genres',
 'homepage',
 'id',
 'keywords',
 'original_language',
 'original_title',
 'overview',
 'popularity',
 'production_companies',
 'production_countries',
 'release_date',
 'revenue',
 'runtime',
 'spoken_languages',
 'status',
 'tagline',
 'title',
```

Step 5:

Check for the null values.

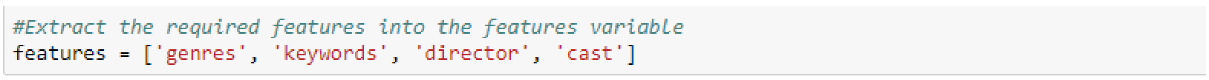


```
#null data
df.isnull().sum()
```

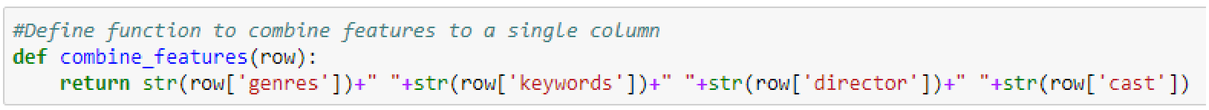
```
index                      0
budget                     0
genres                    28
homepage                3091
id                         0
keywords                 412
original_language          0
original_title             0
overview                   3
popularity                 0
production_companies       0
production_countries       0
release_date               1
revenue                    0
runtime                    2
spoken_languages           0
status                     0
tagline                  845
title                      1
```

Step 6:

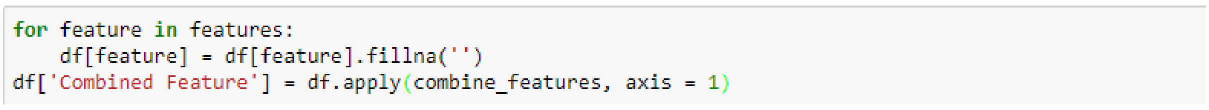
Select Genres, Keywords, Director, Cast and combine them into another column and fill the nan values with blank space.



```
#Extract the required features into the features variable
features = ['genres', 'keywords', 'director', 'cast']
```



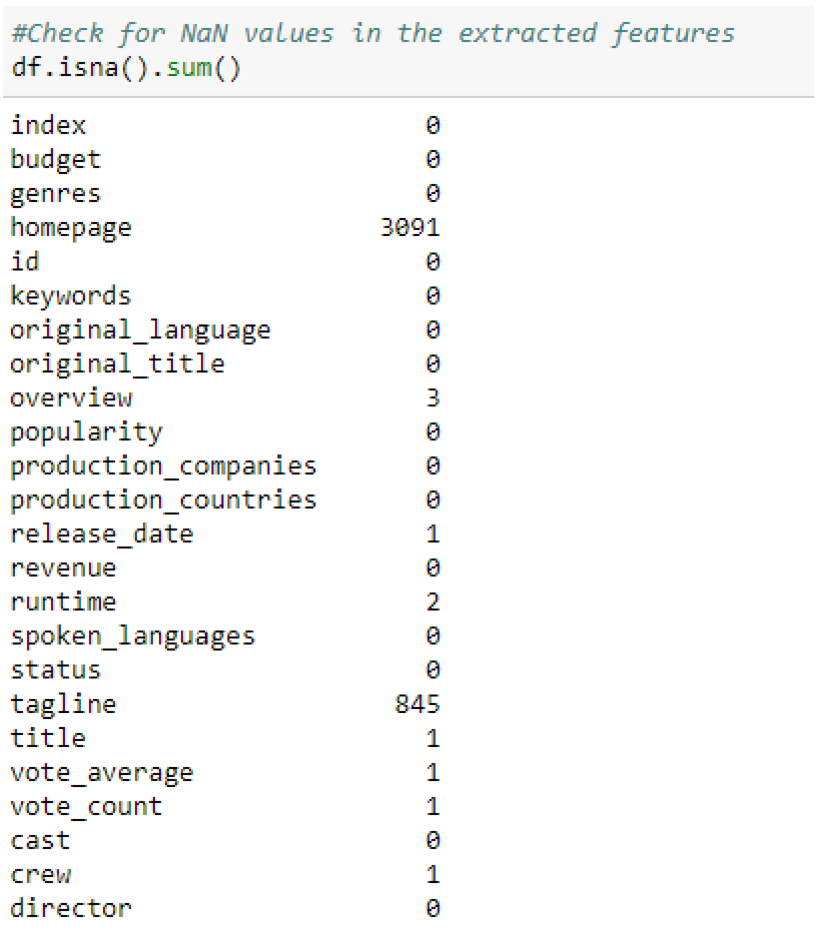
```
#Define function to combine features to a single column
def combine_features(row):
    return str(row['genres'])+" "+str(row['keywords'])+" "+str(row['director'])+" "+str(row['cast'])
```



```
for feature in features:
    df[feature] = df[feature].fillna('')
df['Combined Feature'] = df.apply(combine_features, axis = 1)
```

Step 7:

In the next step we check if all the null values are filled or not in the required fields.



```
#Check for NaN values in the extracted features
df.isna().sum()
```

```
index                      0
budget                     0
genres                     0
homepage                3091
id                         0
keywords                   0
original_language          0
original_title             0
overview                   3
popularity                 0
production_companies       0
production_countries       0
release_date               1
revenue                    0
runtime                    2
spoken_languages           0
status                     0
tagline                  845
title                      1
vote_average               1
vote_count                 1
cast                       0
crew                       1
director                   0
```

Step 8:

In the next step, we import the Count-Vectorizer method from Scikit learn library and store it in an object.

```
#Tokenization and vectorization
from sklearn.feature_extraction.text import CountVectorizer
```

```
cv=CountVectorizer()
```

Step 9:

The combined feature column is fit transformed into X and the X is converted into an array using x.toarray() function.

```
# word counting
x=cv.fit_transform(df['Combined Feature'])#x is a sparse matrix
print(x.toarray())
print(type(x))
```

```
[[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]]
<class 'scipy.sparse.csr.csr_matrix'>
```

Step 10:

From the above output it can be seen that X is an sparse csr_matrix. Hence we import the sparse method and the cosine similarity from scipy and Scikit learn libraries respectively for further processes.

```
from sklearn.metrics.pairwise import cosine_similarity
from scipy import sparse
```

Step 11:

The sparse.csr_matrix method is applied on the X matrix and the output is stored into another matrix A_sparse. The cosine_similarity method is them used on this new matrix.

```
A_sparse=sparse.csr_matrix(x) #convert sparse matrix to cosine Similarity Matrix
similarity_score=cosine_similarity(A_sparse,dense_output=True) #if False then sparse output is returned
```

Step 12:

The similarity score is printed to the similarity between the different columns.

```
print(similarity_score)
```

```
[[1.         0.10540926 0.12038585 ... 0.         0.         0.        ]
 [0.10540926 1.         0.0761387  ... 0.03651484 0.         0.        ]
 [0.12038585 0.0761387  1.         ... 0.         0.11145564 0.        ]
 ...
 [0.         0.03651484 0.         ... 1.         0.         0.04264014]
 [0.         0.         0.11145564 ... 0.         1.         0.        ]
 [0.         0.         0.         ... 0.04264014 0.         1.        ]]
```

Step 13:

In the next step, we define two helper functions to get the values of the title and index as per requirement.

```
def gt_title(tit):
    return df[df.index==tit]['title'].values[0]

def gt_index(ind):
    return df[df.title==ind]['index'].values[0]
```

Step 14:

Enter the movie name of choice.

```
#take value from user and search for similiar movies
input_user=input('\n Enter a movie name \n')
movie_index=int(gt_index(input_user))
similiar_movies=list(enumerate(similarity_score[movie_index]))
```

```
 Enter a movie name 
Avatar
```

Step 15:

Get the similar movies as per the movie name entered in the above step and select the top 5 movies to display in the results.

```
sorted_similiar_movies=sorted(similiar_movies,key=lambda x:x[1],reverse=True)[1:] #Descending values
```

```
i=0
print('Top 5 similiar movies are \n')
for element in sorted_similiar_movies:
    print(gt_title(element[0]))

    i=i+1
    if i>5:
        break
```

```
Top 5 similiar movies are

Man of Steel
X-Men: Days of Future Past
Superman
Superman II
Beastmaster 2: Through the Portal of Time
Suicide Squad
```

We henceforth implement a basic Content Based movie Recommendation System.