```
Movie Recommendation System
```

Loading the Dataset

```
import pandas as pd
import numpy as np
df1=pd.read_csv('tmdb_5000_credits.csv')
df2=pd.read_csv('tmdb_5000_movies.csv')
```

```
df1.columns = ['id','tittle','cast','crew']
df2 = df2.merge(df1,on='id')
```

Peak at our Data.

```
df2.head(5)
```

	budget	genres	homepage	id	keywords	o
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	http://www.thedarkknightrises.com/	49026	[{"id": 849, "name": "dc comics"}, {"id": 853,...	
4	260000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://movies.disney.com/john-carter	49529	[{"id": 818, "name": "based on novel"}, {"id":...	

5 rows × 23 columns

Demographic Filtering

```
C= df2['vote_average'].mean()
C
```

```
6.092171559442016
```

```
m= df2['vote_count'].quantile(0.9)
m
```

```
1838.4000000000015
```

```
q_movies = df2.copy().loc[df2['vote_count'] >= m]
q_movies.shape
```

```
(481, 23)
```

```
def weighted_rating(x, m=m, C=C):
    v = x['vote_count']
    R = x['vote_average']
    # Calculation based on the IMDB formula
    return (v/(v+m) * R) + (m/(m+v) * C)
```

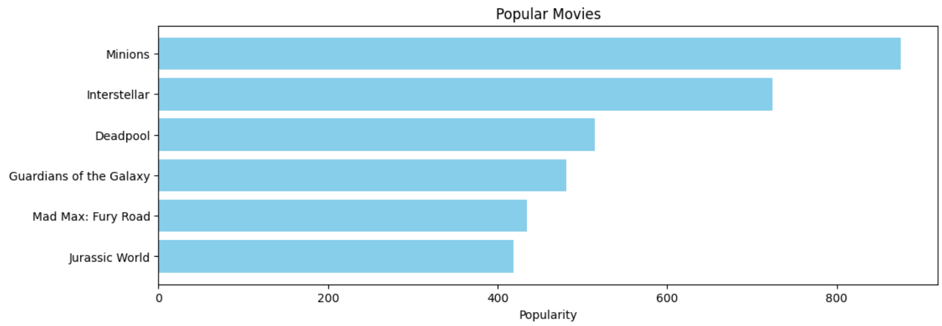
```
# Define a new feature 'score' and calculate its value with `weighted_rating()`
q_movies['score'] = q_movies.apply(weighted_rating, axis=1)
```

```
#Sort movies based on score calculated above
q_movies = q_movies.sort_values('score', ascending=False)

#Print the top 15 movies
q_movies[['title', 'vote_count', 'vote_average', 'score']].head(10)
```

	title	vote_count	vote_average	score
1881	The Shawshank Redemption	8205	8.5	8.059258
662	Fight Club	9413	8.3	7.939256
65	The Dark Knight	12002	8.2	7.920020
3232	Pulp Fiction	8428	8.3	7.904645
96	Inception	13752	8.1	7.863239
3337	The Godfather	5893	8.4	7.851236
95	Interstellar	10867	8.1	7.809479
809	Forrest Gump	7927	8.2	7.803188
329	The Lord of the Rings: The Return of the King	8064	8.1	7.727243
1990	The Empire Strikes Back	5879	8.2	7.697884

```
pop= df2.sort_values('popularity', ascending=False)
import matplotlib.pyplot as plt
plt.figure(figsize=(12,4))

plt.barh(pop['title'].head(6),pop['popularity'].head(6), align='center',
        color='skyblue')
plt.gca().invert_yaxis()
plt.xlabel("Popularity")
plt.title("Popular Movies")
```

```
Text(0.5, 1.0, 'Popular Movies')
```



Content Based Filtering

```
df2['overview'].head(5)
```

```
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
```

Plot description based Recommender

```
#Import TfIdfVectorizer from scikit-learn
from sklearn.feature_extraction.text import TfidfVectorizer

#Define a TF-IDF Vectorizer Object. Remove all english stop words such as 'the', 'a'
tfidf = TfidfVectorizer(stop_words='english')

#Replace NaN with an empty string
df2['overview'] = df2['overview'].fillna('')

#Construct the required TF-IDF matrix by fitting and transforming the data
tfidf_matrix = tfidf.fit_transform(df2['overview'])

#Output the shape of tfidf_matrix
tfidf_matrix.shape
```

```
(4803, 20978)
```

```
# Import linear_kernel
from sklearn.metrics.pairwise import linear_kernel

# Compute the cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

```
#Construct a reverse map of indices and movie titles
indices = pd.Series(df2.index, index=df2['title']).drop_duplicates()
```

```
# 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 df2['title'].iloc[movie_indices]
```

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

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

```
get_recommendations('The Avengers')
```

```
7               Avengers: Age of Ultron
3144                            Plastic
1715                            Timecop
4124                 This Thing of Ours
3311              Thank You for Smoking
3033                      The Corruptor
588     Wall Street: Money Never Sleeps
2136         Team America: World Police
1468                       The Fountain
```

```
1286                      Snowpiercer
Name: title, dtype: object
```

Credits, Genres and Keywords Based Recommender

```
# Parse the stringified features into their corresponding python objects
from ast import literal_eval
features = ['cast', 'crew', 'keywords', 'genres']
for feature in features:
    df2[feature] = df2[feature].apply(literal_eval)
```

```
# Get the director's name from the crew feature. If director is not listed, return NaN
def get_director(x):
    for i in x:
        if i['job'] == 'Director':
            return i['name']
    return np.nan
```

```
# Returns the list top 3 elements or entire list; whichever is more.
def get_list(x):
    if isinstance(x, list):
        names = [i['name'] for i in x]
        #Check if more than 3 elements exist. If yes, return only first three. If no, return entire list.
        if len(names) > 3:
            names = names[:3]
        return names

    #Return empty list in case of missing/malformed data
    return []
```

```
# Define new director, cast, genres and keywords features that are in a suitable form.
df2['director'] = df2['crew'].apply(get_director)

features = ['cast', 'keywords', 'genres']
for feature in features:
    df2[feature] = df2[feature].apply(get_list)
```

```
# Print the new features of the first 3 films
df2[['title', 'cast', 'director', 'keywords', 'genres']].head(3)
```

	title	cast	director	keywords	genres
0	Avatar	[Sam Worthington, Zoe Saldana, Sigourney Weaver]	James Cameron	[culture clash, future, space war]	[Action, Adventure, Fantasy]
1	Pirates of the Caribbean: At World's	[Johnny Depp, Orlando	Gore	[ocean, drug abuse, exotic	[Adventure,

```
# Function to convert all strings to lower case and strip names of spaces
def clean_data(x):
    if isinstance(x, list):
        return [str.lower(i.replace(" ", "")) for i in x]
    else:
        #Check if director exists. If not, return empty string
        if isinstance(x, str):
            return str.lower(x.replace(" ", ""))
        else:
            return ''
```

```
# Apply clean_data function to your features.
features = ['cast', 'keywords', 'director', 'genres']

for feature in features:
    df2[feature] = df2[feature].apply(clean_data)
```

```
def create_soup(x):
    return ' '.join(x['keywords']) + ' ' + ' '.join(x['cast']) + ' ' + x['director'] + ' ' + ' '.join(x['genres'])
df2['soup'] = df2.apply(create_soup, axis=1)
```

```
# Import CountVectorizer and create the count matrix
from sklearn.feature_extraction.text import CountVectorizer

count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(df2['soup'])
```

```
# Compute the Cosine Similarity matrix based on the count_matrix
from sklearn.metrics.pairwise import cosine_similarity

cosine_sim2 = cosine_similarity(count_matrix, count_matrix)
```

```
# Reset index of our main DataFrame and construct reverse mapping as before
df2 = df2.reset_index()
indices = pd.Series(df2.index, index=df2['title'])
```

```
get_recommendations('The Dark Knight Rises', cosine_sim2)
```

```
65              The Dark Knight
119               Batman Begins
4638    Amidst the Devil's Wings
1196               The Prestige
3073          Romeo Is Bleeding
3326             Black November
1503                     Takers
1986                     Faster
303                    Catwoman
747              Gangster Squad
Name: title, dtype: object
```

```
get_recommendations('The Godfather', cosine_sim2)
```

```
867      The Godfather: Part III
2731      The Godfather: Part II
4638    Amidst the Devil's Wings
2649           The Son of No One
1525              Apocalypse Now
1018             The Cotton Club
1170     The Talented Mr. Ripley
1209              The Rainmaker
1394               Donnie Brasco
1850                    Scarface
Name: title, dtype: object
```

Collaborative Filtering

```
pip install scikit-surprise
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting scikit-surprise
  Downloading scikit-surprise-1.1.3.tar.gz (771 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 772.0/772.0 kB 15.7 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.9/dist-packages (from scikit-surprise) (1.2.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.9/dist-packages (from scikit-surprise) (1.22.4)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.9/dist-packages (from scikit-surprise) (1.10.1)
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (setup.py) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.3-cp39-cp39-linux_x86_64.whl size=3195792 sha256=171ccba0c4910f2e
  Stored in directory: /root/.cache/pip/wheels/c6/3a/46/9b17b3512bdf283c6cb84f59929cdd5199d4e754d596d22784
Successfully built scikit-surprise
Installing collected packages: scikit-surprise
Successfully installed scikit-surprise-1.1.3
```

```
from surprise import Reader, Dataset, SVD
from surprise.model_selection import cross_validate
reader = Reader()
ratings = pd.read_csv('ratings_small.csv')
ratings.head()
```

	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

```
data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)
svd = SVD()
cross_validate(svd, data, measures=['RMSE','MAE'], cv=5)
```

```
{'test_rmse': array([0.90074163, 0.88864268, 0.89761464, 0.89946563, 0.89282843]),
 'test_mae': array([0.69126334, 0.68522416, 0.69195747, 0.69266082, 0.68729051]),
 'fit_time': (1.7612783908843994,
  1.5000770092010498,
  1.4106993675231934,
  2.4583699703216553,
  1.8040664196014404),
 'test_time': (0.32048487663269043,
  0.1479783058166504,
  0.15651559829711914,
  0.1526472568511963,
  0.16502070426940918)}
```

```
trainset = data.build_full_trainset()
svd.fit(trainset)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7fdb1cdd4880>
```

```
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=2.810398000928227, details={'was_impossible': False})
```

Conclusion

We create recommenders using demographic , content- based and collaborative filtering. While demographic filtering is very elemantary and cannot be used practically, Hybrid Systems can take advantage of content-based and collaborative filtering as the two approaches are proved to be almost complimentary. This model was very baseline and only provides a fundamental framework to start with.