## Reading the TMDB movie dataset

In [4]:

```
import pandas as pd
import numpy as np
df1=pd.read_csv('C:\\Users\\Dwarkish\\Downloads\\archive (2)\\tmdb_5000_credits.csv')
df2=pd.read_csv('C:\\Users\\Dwarkish\\Downloads\\archive (2)\\tmdb_5000_movies.csv')
```

In [5]:

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

In [6]:

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

Out[6]:

| | budget | genres | homepage | id | keywords | original_langu |
|---|---|---|---|---|---|---|
| **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

In [7]:
```
C= df2['vote_average'].mean()
C
```

Out[7]:
```
6.092171559442011
```

In [9]:
```
m= df2['vote_count'].quantile(0.9)
m
```

Out[9]:
```
1838.4000000000015
```

In [11]:
```
q_movies = df2.copy().loc[df2['vote_count'] >= m]
q_movies.shape
```

Out[11]:
```
(481, 23)
```

In [12]:
```
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)
```

In [13]:
```
# Define a new feature 'score' and calculate its value with `weighted_rating()`
q_movies['score'] = q_movies.apply(weighted_rating, axis=1)
```

In [14]:
```
#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)
```

Out[14]:

| | 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 |

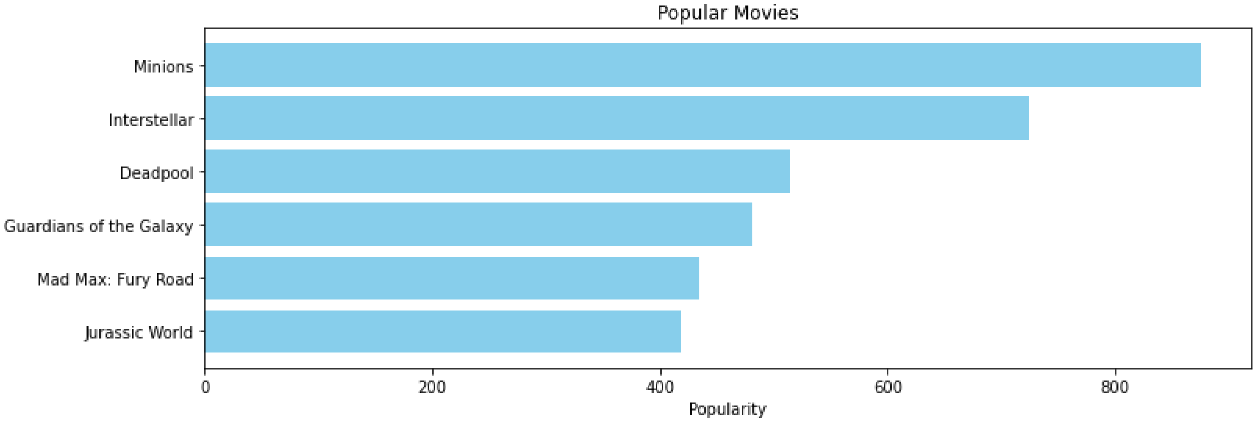
In [15]:

```
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")
```

Out[15]:

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



## Content Based Filtering

## Plot description based Recommender

In [17]:

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

Out[17]:

```
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
```

In [18]:

```
#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
```

Out[18]:

```
(4803, 20978)
```

In [19]:

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

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

In [20]:

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

In [21]:

```
# 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]
```

In [22]:

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

Out[22]:

```
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
```

In [23]:

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

Out[23]:

```
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

In [24]:

```python
# 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)
```

In [25]:

```python
# 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
```

In [26]:

```python
# 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, r
        if len(names) > 3:
            names = names[:3]
        return names

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

In [27]:

```python
# 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)
```

In [28]:

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

Out[28]:

| | 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 End | [Johnny Depp, Orlando Bloom, Keira Knightley] | Gore Verbinski | [ocean, drug abuse, exotic island] | [Adventure, Fantasy, Action] |

| | title | cast | director | keywords | genres |
|---|---|---|---|---|---|
| **2** | Spectre | [Daniel Craig, Christoph Waltz, Léa Seydoux] | Sam Mendes | [spy, based on novel, secret agent] | [Action, Adventure, Crime] |

In [29]:

```python
# 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 ''
```

In [30]:

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

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

In [31]:

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

In [32]:

```python
# 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'])
```

In [33]:

```python
# 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)
```

In [34]:

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

In [35]:

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

Out[35]:

```
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
```

In [36]:

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

Out[36]:

```
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

In [39]:

```
from surprise import Reader, Dataset, SVD
from surprise.model_selection import cross_validate
reader = Reader()
ratings = pd.read_csv('C:\\Users\\Dwarkish\\Downloads\\archive (1)\\ratings_small.csv')
ratings.head()
```

Out[39]:

| | 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 |

In [40]:

```
data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)
```

In [41]:

```
svd = SVD()
cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=5)
```

Out[41]:

```
{'test_rmse': array([0.89608291, 0.90241519, 0.89471592, 0.89519998, 0.89583649]),
 'test_mae': array([0.68773676, 0.69487939, 0.68684834, 0.68994524, 0.69134276]),
 'fit_time': (6.703330039978027,
  6.949191093444824,
  6.659073829650879,
  6.9347639083862305,
  6.675144672393799),
 'test_time': (0.25872802734375,
  0.1831045150756836,
  0.19442415237426758,
  0.20020771026611328,
  0.2141282558441162)}
```

In [42]:

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

Out[42]:

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x21c6ff6bac0>
```

In [43]:

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

Out[43]:

| | 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 |

In [44]:

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

Out[44]:

```
Prediction(uid=1, iid=302, r_ui=3, est=2.7375526346714354, details={'was_impossible': Fa
lse})
```

In [45]:

```
df2.tail(5)
```

Out[45]:

| | index | budget | genres | homepage | id | keywo |
|---|---|---|---|---|---|---|

| | index | budget | genres | homepage | id | keywor |
|---|---|---|---|---|---|---|
| **4798** | 4798 | 220000 | [action, crime, thriller] | NaN | 9367 | [unitedstat mexicobarr legs, arr |
| **4799** | 4799 | 9000 | [comedy, romance] | NaN | 72766 | |
| **4800** | 4800 | 0 | [comedy, drama, romance] | http://www.hallmarkchannel.com/signedsealeddel... | 231617 | [da loveatfirstsig narrati |
| **4801** | 4801 | 0 | [] | http://shanghaicalling.com/ | 126186 | |
| **4802** | 4802 | 0 | [documentary] | NaN | 25975 | [obsessi camcorc cru |

5 rows × 26 columns

In [ ]: