# Import Required Libraries

In [1]:
```
import pandas as pd
import numpy as np
```

# Load Data

In [2]:
```
movies_credits = pd.read_csv('./data/tmdb_5000_credits.csv')

movies = pd.read_csv('./data/tmdb_5000_movies.csv')
```

# 1.0 Initial Data Analysis

In [3]:
```
# Top 10 movies ranked by the number of votes

movies.sort_values('vote_count', ascending=False).head(10)['title']
```

Out[3]:
```
96                   Inception
65             The Dark Knight
0                       Avatar
16               The Avengers
788                   Deadpool
95                Interstellar
287          Django Unchained
94     Guardians of the Galaxy
426           The Hunger Games
127         Mad Max: Fury Road
Name: title, dtype: object
```

In [4]:
```
# Top 10 movies ranked by the revenue

movies.sort_values('revenue', ascending=False).head(10)['title']
```

Out[4]:
```
0                         Avatar
25                       Titanic
16                  The Avengers
28                Jurassic World
44                     Furious 7
7        Avengers: Age of Ultron
124                       Frozen
31                    Iron Man 3
546                      Minions
26     Captain America: Civil War
Name: title, dtype: object
```

# 2.0 Recommendation Engines

## 2.1 Initial Data Cleaning

In [5]:
```
movies_credits.drop('title', axis = 1, inplace = True)

movies_credits.head()
```

Out[5]:

| | movie_id | cast | crew |
|---|---|---|---|
| **0** | 19995 | [{"cast_id": 242, "character": "Jake Sully", "... | [{"credit_id": "52fe48009251416c750aca23", "de... |
| **1** | 285 | [{"cast_id": 4, "character": "Captain Jack Spa... | [{"credit_id": "52fe4232c3a36847f800b579", "de... |
| **2** | 206647 | [{"cast_id": 1, "character": "James Bond", "cr... | [{"credit_id": "54805967c3a36829b5002c41", "de... |
| **3** | 49026 | [{"cast_id": 2, "character": "Bruce Wayne / Ba... | [{"credit_id": "52fe4781c3a36847f81398c3", "de... |
| **4** | 49529 | [{"cast_id": 5, "character": "John Carter", "c... | [{"credit_id": "52fe479ac3a36847f813eaa3", "de... |

In [6]:
```
movies_credits.columns = ['id', 'cast', 'crew']
movies = movies.merge(movies_credits, on='id')

movies.head()
```

Out[6]:

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

5 rows × 22 columns

## 2.2 Demographic Filtering - Trending now for All Users

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

C
```

Out[7]: 6.092171559442011

In [8]:
```
m = movies['vote_count'].quantile(0.8)

m
```

Out[8]: 957.6000000000004

In [9]:
```
qualified_movies_demographic = movies.copy().loc[movies['vote_count'] >= m]

qualified_movies_demographic.shape
```

Out[9]: (961, 22)

In [10]:
```
def weighted_rating_score(data, m=m, C=C):
    v = data['vote_count']
    R = data['vote_average']
    # weighted rating score calculation formula
    return (v/(v+m) * R) + (m/(m+v) * C)
```

In [11]:
```
# calculate weighted rating score for each movie we have

qualified_movies_demographic['weighted_score'] = qualified_movies_demographic.appl
y(weighted_rating_score, axis=1)
```

In [12]:
```
# descending sort movies based on the weighted rating score

qualified_movies_demographic = qualified_movies_demographic.sort_values('weighted_
score', ascending=False)
```

In [13]:
```
# print out the top 20 movies based on the weighted_score

qualified_movies_demographic[['title', 'vote_count', 'vote_average', 'weighted_sco
re']].head(20)
```

Out[13]:

| | title | vote_count | vote_average | weighted_score |
|---|---|---|---|---|
| **1881** | The Shawshank Redemption | 8205 | 8.5 | 8.248353 |
| **662** | Fight Club | 9413 | 8.3 | 8.096134 |
| **3337** | The Godfather | 5893 | 8.4 | 8.077404 |
| **3232** | Pulp Fiction | 8428 | 8.3 | 8.074738 |
| **65** | The Dark Knight | 12002 | 8.2 | 8.044250 |
| **809** | Forrest Gump | 7927 | 8.2 | 7.972814 |
| **96** | Inception | 13752 | 8.1 | 7.969290 |
| **95** | Interstellar | 10867 | 8.1 | 7.937399 |
| **1990** | The Empire Strikes Back | 5879 | 8.2 | 7.904757 |
| **1818** | Schindler's List | 4329 | 8.3 | 7.900080 |
| **3865** | Whiplash | 4254 | 8.3 | 7.894325 |
| **329** | The Lord of the Rings: The Return of the King | 8064 | 8.1 | 7.886879 |
| **2294** | Spirited Away | 3840 | 8.3 | 7.859318 |
| **2912** | Star Wars | 6624 | 8.1 | 7.846400 |
| **1553** | Se7en | 5765 | 8.1 | 7.813995 |
| **262** | The Lord of the Rings: The Fellowship of the Ring | 8705 | 8.0 | 7.810927 |
| **2731** | The Godfather: Part II | 3338 | 8.3 | 7.807818 |
| **690** | The Green Mile | 4048 | 8.2 | 7.796760 |
| **330** | The Lord of the Rings: The Two Towers | 7487 | 8.0 | 7.783656 |
| **77** | Inside Out | 6560 | 8.0 | 7.756979 |

## 2.3 Content Based Filtering - using Movie Overviews

In [14]:
```
# use scikit-learn's TfIdfVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

# remove english stop words like 'the', 'a' and define a TF-IDF Vectorizer Object.
tfidf = TfidfVectorizer(stop_words='english')

# change NaN into an empty string
movies['overview'] = movies['overview'].fillna('')

# build the TF-IDF matrix
tfidf_matrix = tfidf.fit_transform(movies['overview'])
```

In [15]:
```
# import linear_kernel
from sklearn.metrics.pairwise import linear_kernel

# build the cosine similarity matrix
cosine_sim_overviews = linear_kernel(tfidf_matrix, tfidf_matrix)
```

In [16]:
```
# build a reverse map of movie titles and indices
indices = pd.Series(movies.index, index=movies['title']).drop_duplicates()
```

In [17]:
```
# recommend similar movies to users given a movie title input

def get_recommendation_movies(title, cosine_sim):
    # get the movie index
    idx = indices[title]

    # calculate the pairwsie similarity scores 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 10 most similar movies scores
    sim_scores = sim_scores[1:11]

    # get movie indices
    movie_indices = [i[0] for i in sim_scores]

    # return the recommended similar movies
    return movies['title'].iloc[movie_indices]
```

In [18]:
```
get_recommendation_movies('The Dark Knight Rises', cosine_sim_overviews)
```

Out[18]:
```
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 [19]:
```
get_recommendation_movies('The Avengers', cosine_sim_overviews)
```

Out[19]:
```
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
```

## 2.4 Content Based Filtering - using Movie Metadata ('genres', 'keywords', 'cast', 'crew', 'production_companies')

In [20]:
```
# import function to parse the stringified features into python objects

from ast import literal_eval
```

In [21]:
```
features = ['genres', 'keywords', 'cast', 'crew', 'production_companies']

for feature in features:
    movies[feature] = movies[feature].apply(literal_eval)
```

In [22]:
```
movies.head(3)
```

Out[22]:

| | budget | genres | homepage | id | keywords | original_language | origina |
|---|---|---|---|---|---|---|---|
| **0** | 237000000 | [{'id': 28, 'name': 'Action'}, {'id': 12, 'nam... | http://www.avatarmovie.com/ | 19995 | [{'id': 1463, 'name': 'culture clash'}, {'id':... | en | Avatar |
| **1** | 300000000 | [{'id': 12, 'name': 'Adventure'}, {'id': 14, '... | http://disney.go.com/disneypictures/pirates/ | 285 | [{'id': 270, 'name': 'ocean'}, {'id': 726, 'na... | en | Pirates the Caribb At Wor End |
| **2** | 245000000 | [{'id': 28, 'name': 'Action'}, {'id': 12, 'nam... | http://www.sonypictures.com/movies/spectre/ | 206647 | [{'id': 470, 'name': 'spy'}, {'id': 818, 'name... | en | Spectr |

3 rows × 22 columns

In [23]:
```
# helper function: extract director name

def get_director(crew_data):
    for i in crew_data:
        if i['job'] == 'Director':
            return i['name']
    return np.nan
```

In [24]:
```
# helper function: transfer into list data objects

def get_list(data):
    if isinstance(data, list):
        names = [i['name'] for i in data]
        # return only the first three items
        if len(names) > 3:
            names = names[:3]
        return names

    return []
```

In [25]:
```
# use helper functions to parse features

movies['director'] = movies['crew'].apply(get_director)

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

In [26]:
```
# print the new features of the first 3 films

movies[['title', 'genres', 'keywords', 'cast', 'production_companies', 'director']
].head(3)
```

Out[26]:

| | title | genres | keywords | cast | production_companies | director |
|---|---|---|---|---|---|---|
| **0** | Avatar | [Action, Adventure, Fantasy] | [culture clash, future, space war] | [Sam Worthington, Zoe Saldana, Sigourney Weaver] | [Ingenious Film Partners, Twentieth Century Fo... | James Cameron |
| **1** | Pirates of the Caribbean: At World's End | [Adventure, Fantasy, Action] | [ocean, drug abuse, exotic island] | [Johnny Depp, Orlando Bloom, Keira Knightley] | [Walt Disney Pictures, Jerry Bruckheimer Films... | Gore Verbinski |
| **2** | Spectre | [Action, Adventure, Crime] | [spy, based on novel, secret agent] | [Daniel Craig, Christoph Waltz, Léa Seydoux] | [Columbia Pictures, Danjaq, B24] | Sam Mendes |

In [27]:
```
# convert all strings into lower case and
# strip the spaces between names so as to be specific

def clean_data(data):
    if isinstance(data, list):
        return [str.lower(i.replace(" ", "")) for i in data]
    else:
        if isinstance(data, str):
            return str.lower(data.replace(" ", ""))
        else:
            return ''
```

In [28]:
```
features = ['genres', 'keywords', 'cast', 'production_companies', 'director']

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

In [29]:
```
def create_data_soup(x):
    return ' '.join(x['keywords']) + ' ' + ' '.join(x['cast']) + ' ' + x['director
'] + ' ' + ' '.join(x['genres']) + ' ' + ' '.join(x['production_companies'])
```

In [30]:
```
movies['data_soup'] = movies.apply(create_data_soup, axis=1)
```

In [31]:
```
# import CountVectorizer, create count matrix
from sklearn.feature_extraction.text import CountVectorizer

count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(movies['data_soup'])
```

In [32]:
```
# compute the Cosine Similarity matrix
from sklearn.metrics.pairwise import cosine_similarity

cosine_sim_metadata = cosine_similarity(count_matrix, count_matrix)
```

In [33]:
```
# reset the indices of our main DataFrame, construct reverse mapping

movies = movies.reset_index()
indices = pd.Series(movies.index, index=movies['title'])
```

In [34]:
```
get_recommendation_movies('The Dark Knight Rises', cosine_sim_metadata)
```

Out[34]:
```
65               The Dark Knight
119                Batman Begins
14                  Man of Steel
1196                The Prestige
4638    Amidst the Devil's Wings
10              Superman Returns
1035                   Jonah Hex
299               Batman Forever
303                     Catwoman
747                Gangster Squad
Name: title, dtype: object
```

In [35]:
```
get_recommendation_movies('The Avengers', cosine_sim_metadata)
```

Out[35]:
```
7                   Avengers: Age of Ultron
26               Captain America: Civil War
79                               Iron Man 2
169     Captain America: The First Avenger
85     Captain America: The Winter Soldier
174                     The Incredible Hulk
31                               Iron Man 3
68                                 Iron Man
182                                 Ant-Man
94                  Guardians of the Galaxy
Name: title, dtype: object
```

## 2.5 Content Based Filtering - using Movie Metadata ('genres', 'keywords', 'cast', 'crew', 'production_companies') and voting scores

In [39]:
```
def improved_recommendations_metadata_votes(title, cosine_sim):
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:36]
    movie_indices = [i[0] for i in sim_scores]

    movies_improved_recomm = movies.copy().iloc[movie_indices][['title', 'vote_cou
nt', 'vote_average']]
    vote_counts_improved = movies_improved_recomm[movies_improved_recomm['vote_cou
nt'].notnull()]['vote_count'].astype('int')
    vote_averages_improved = movies_improved_recomm[movies_improved_recomm['vote_a
verage'].notnull()]['vote_average'].astype('int')
    C_improved = vote_counts_improved.mean()
    m_improved = vote_averages_improved.quantile(0.60)

    qualified_improve = movies_improved_recomm[(movies_improved_recomm['vote_count
'] >= m_improved) & (movies_improved_recomm['vote_count'].notnull()) & (movies_imp
roved_recomm['vote_average'].notnull())]
    qualified_improve['wr'] = qualified_improve.apply(weighted_rating_score, axis=
1)
    qualified_improve = qualified_improve.sort_values('wr', ascending=False).head(
10)
    return qualified_improve
```

In [40]:
```
improved_recommendations_metadata_votes('The Dark Knight Rises', cosine_sim_metada
ta)
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:15: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stabl
e/indexing.html#indexing-view-versus-copy
  from ipykernel import kernelapp as app
```

Out[40]:

| | title | vote_count | vote_average | wr |
|---|---|---|---|---|
| **65** | The Dark Knight | 12002 | 8.2 | 8.044250 |
| **96** | Inception | 13752 | 8.1 | 7.969290 |
| **95** | Interstellar | 10867 | 8.1 | 7.937399 |
| **1196** | The Prestige | 4391 | 8.0 | 7.658427 |
| **119** | Batman Begins | 7359 | 7.5 | 7.337898 |
| **1663** | Once Upon a Time in America | 1069 | 8.2 | 7.204018 |
| **1052** | Training Day | 1634 | 7.3 | 6.853706 |
| **3854** | Batman: The Dark Knight Returns, Part 2 | 419 | 7.9 | 6.642426 |
| **14** | Man of Steel | 6359 | 6.5 | 6.446623 |
| **1456** | Bound by Honor | 115 | 7.7 | 6.264557 |

In [41]:
```
improved_recommendations_metadata_votes('The Avengers', cosine_sim_metadata)
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:15: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stabl
e/indexing.html#indexing-view-versus-copy
  from ipykernel import kernelapp as app
```

Out[41]:

| | title | vote_count | vote_average | wr |
|---|---|---|---|---|
| **94** | Guardians of the Galaxy | 9742 | 7.9 | 7.738202 |
| **85** | Captain America: The Winter Soldier | 5764 | 7.6 | 7.385186 |
| **68** | Iron Man | 8776 | 7.4 | 7.271335 |
| **158** | Star Trek | 4518 | 7.4 | 7.171280 |
| **47** | Star Trek Into Darkness | 4418 | 7.4 | 7.167026 |
| **7** | Avengers: Age of Ultron | 6767 | 7.3 | 7.150268 |
| **26** | Captain America: Civil War | 7241 | 7.1 | 6.982285 |
| **182** | Ant-Man | 5880 | 7.0 | 6.872859 |
| **1294** | Serenity | 1264 | 7.4 | 6.836273 |
| **31** | Iron Man 3 | 8806 | 6.8 | 6.730577 |