importing libraries

In [ ]:
```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from ast import literal_eval
from sklearn.feature_extraction.text import TfidfVectorizer , CountVectorizer
from sklearn.metrics.pairwise import linear_kernel , cosine_similarity
from nltk.stem.snowball import SnowballStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import wordnet
from surprise import Reader  , Dataset , SVD
from surprise.nodel_selection import cross_validate

import warnings ; warnings.simplefilter('ignore')
```

Top N Recommendations

In [ ]:
```
md = pd.read_csv('movies_metasata.csv')
md.head()
```

Preprocessing

In [ ]:
```
md['genres'] = md['genres'].fillna('[]')
```

In [ ]:
```
md.head(100)
```

In [ ]:
```
md['genres'] = md['genres'].apply(literal_eval)
md.head()
```

In [ ]:
```
#Genres as list
```

In [ ]:
```
md['genres'] = md['genres'].apply(lambda x:[i['name']for i in x]if isinstance (x,list)else[])
```

In [ ]:
```
md.head()
```

In [ ]:
```
md[md['vote_count'].notnull()]
```

In [ ]:
```
vote_count = md[md['vote_count'].notnull()]['vote_count'].astype('int')
vote_count
```

In [ ]:
```
vote_average = md[md['vote_average'].notnull()]['vote_average'].astype('int')
vote_average
```

In [ ]:
```
top_movies = md.copy()
```

In [ ]:
```
top_movies = top_movies.sort_values('vote_average',ascending = False).head(250)
```

In [ ]:
```
# No minimum votes requiremnt
```

In [ ]:
```
top_movies1
```

In [ ]:
```
# Minimum number of votes 1000
```

In [ ]:
```
top_movies2 = top_movies[top_movies['vote_count']>1000]
```

In [ ]:
```
top_movies2
```

In [ ]:
```
vote_count = md[md['vote_count'].notnull()]['vote_count'].astype('int')
vote_averages = md[md['vote_averages'].notnull()]['vote_averages'].astype('int')
C = vote_averages.mean()
C
```

In [ ]:
```
m = vote_count.quantile(0.95)
m
```

In [ ]:
```
top_movies['year'] = pd.to_datetime(top_movies['release_date'],errors = 'coerce'),apply(lambda x: str(x).split(-)[0]if x
```

In [ ]:
```
top_movies
```

In [ ]:
```
top_movies3 = top_movies[(top_movies['vote_count']>=m) & (top_movies['vote_count'].notnull()) & (top_movies['vote_average
top_movies3['vote_count'] = top_movies3['vote_count'].astype('int')
```

```
top_movies3['vote_average'] = top_movies3['vote_average'].astype('int')
top_movies3.shape
```

In [ ]:
```
def weighted_rating(x):
    v = x['vote_count']
    R = x['vote_average']
    return(v/(v+m)*R) + (m/(m+v)*C)
```

In [ ]:
```
top_movies3['weight_rate'] = top_movies3.apply(weighted_rating , axis=1)
```

In [ ]:
```
top_movies3.head()
```

In [ ]:
```
top_movies3 = top_movies3.sort_values('weight_rate' , ascending = False).head(10)
```

In [ ]:
```
top_movies3.head(10)
```

Top Movies

In [ ]:
```
qualified.head(15)
```

In [ ]:
```
# Genre = Romance
```

In [ ]:
```
genre_TM = top_movies.apply(lambda x: pd.Series(x['genres']),axis=1).stack().reset_index(level=1, drop= True)
genre_TM.name = 'genre'
genre_top_movies = top_movies.drop('genre' , axis=1).join(genre_TM)
```

In [ ]:
```
genre_top_movies
```

In [ ]:
```
def build_chart(genre, percentile=0.85):
    df=genre_top_movies[genre_top_movies['genre']==genre]
    vote_counts = df[df['vote_count'].notnull()]['vote_count'].astype('int')
    vote_averages = df[df['vote_average'].notnull()]['vote_average'].astype('int')
    C = vote_averages.mean()
    m = vote_counts.quantile(percentile)

    qualified = df[(df['vote_count']>= m) & (df['vote_count'].notnull()) & (df['vote_average'].notnull())][['title' , 'ye
    qualified['vote_count'] = qualified['vote_count'].astype('int')
    qualified['vote_average'] = qualified['vote_average'].astype('int')

    qualified['wr'] = qualified.apply(lambda x: (x['vote_count']/(x['vote_count'])))
```

```
    qualified = qualified.sort_values('wr' , ascending=False).head(250)

    return qualified
```

Top Genre Movies

In [ ]:
```
built_chart('Animation').head(10)
```

In [ ]:
```
built_chart('Family').head(10)
```

In [ ]:
```
built_chart('Action').head(10)
```

Content Based Recommender

In [ ]:
```
links_small = pd.read_csv('links_samll_csv')
links_small = links_samll[links_samll['tmdbId'].notnull()]['tmdbId'].astype('int')
```

In [ ]:
```
top_movies = top_movies.drop([19730,29503,35587])
```

In [ ]:
```
top_movies['id'] = top_movies['id'].astype('int')
```

In [ ]:
```
top_movies4 = top_movies[top_movies['id'].isin(links_small)]
top_movies4.shape
```

In [ ]:
```
top_movies4(head)
```

In [ ]:
```
top_movies4['tagline'] = top_movies4['tagline'].fillana('')
top_movies4['description'] = top_movies4['overview'] + top_movies4['tagline']
top_movies4['description'] = top_movies4['description'].fillana('')
```

In [ ]:
```
tf = TfidfVectorizer(analyzer='word' , ngram_range=(1,2),min_df=0, stop_words='end')
tfidf_matrix = ts.fit_transform(top_movies4['description'])
```

In [ ]:
```
tfidf_matrix
```

In [ ]:
```
tfidf_matrix.shape
```

In [ ]:
```
cosine_sim = linear_kernel(tfidf_matrix , tfidf_matrix)
```

In [ ]:
```
cosine_sim
```

In [ ]:
```
cosine_sim[0]
```

In [ ]:
```
top_movies4 = top_movies4.reset_index()
titles = top_movies4['title']
indices = pd.Series(top_movies4.index, index=top_movies4['title'])
```

In [ ]:
```
def get_recommendations(title):
    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[1:31]
    movie_indices = [i[0] for i in sim_scores]
    return titles.iloc[movie_indices]
```

In [ ]:
```
get_recommendations('GoldenEye').head(10)
```

In [ ]:
```
get_recommendations('The Appartment').head(10)
```

In [ ]:
```
get_recommendations('The Godfather').head(10)
```

In [ ]:
```
get_recommendations('The Dark Knight').head(10)
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```