```
import pandas as pd

metadata = pd.read_csv('movies_metadata.csv',low_memory=False)

metadata.head(3)

C = metadata['vote_average'].mean()
print(C)

#number of votes m receive by a movie in the 90th percentile
m=metadata['vote_count'].quantile(0.90)
print(m)

#filter out all qualified movie into net dataset
q_movies = metadata.copy().loc[metadata['vote_count'] >= m]
q_movies.shape

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

#define new feature score and calculate its weighted rate
q_movies['score']= q_movies.apply(weighted_rating, axis = 1)


#sorted movie
q_movies = q_movies.sort_values('score',ascending=False)

q_movies[['title', 'vote_count' , 'vote_average' , 'score']].head(20)
```

—----------

Content-Based Recommender

```
#Print plot overviews of the first 5 movies.
metadata['overview'].head()



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



# Import linear_kernel
from sklearn.metrics.pairwise import linear_kernel
# Compute the cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)

cosine_sim.shape

cosine_sim[1]

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

indices[:10]



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

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

get_recommendations('The Godfather')
```