import pandas as pd

metadata = pd.read\_csv('movies\_metadata.csv', low\_memory=False)

metadata.head(3)

C = metadata['vote\_average'].mean()

print(C)

m = metadata['vote\_count'].quantile(0.90)

print(m)

q\_movies = metadata.copy().loc[metadata['vote\_count'] >= m]

q\_movies.shape

metadata.shape

def weighted\_rating(x, m=m, C=C):

v = x['vote\_count']

R = x['vote\_average']

return (v/(v+m) \* R) + (m/(m+v) \* C)

q\_movies['score'] = q\_movies.apply(weighted\_rating, axis=1)

q\_movies = q\_movies.sort\_values('score', ascending=False)

q\_movies[['title', 'vote\_count', 'vote\_average', 'score']].head(20)

#Print plot overviews of the first 5 movies.

q\_movies['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

q\_movies['overview'] = q\_movies['overview'].fillna('')

#Construct the required TF-IDF matrix by fitting and transforming the data

tfidf\_matrix = tfidf.fit\_transform(q\_movies['overview'])

#Output the shape of tfidf\_matrix

tfidf\_matrix.shape

#Array mapping from feature integer indices to feature name.

tfidf.get\_feature\_names\_out()[5000:5010]

# 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(metadata.index, index=metadata['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 metadata['title'].iloc[movie\_indices]

get\_recommendations('Toy Story')