import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import os

import ast

movies = pd.read\_csv(r'tmdb\_5000\_movies.csv')

credits = pd.read\_csv(r'tmdb\_5000\_credits.csv')

movies.head()

credits.head()

movies.shape

credits.shape

# Merging datasets

movies = movies.merge(credits, on = 'title')

movies.head(3)

movies['original\_language'].value\_counts()

movies.columns

# Choosing columns which I think would be relevant

movies = movies[['movie\_id', 'title', 'overview','genres', 'keywords', 'cast', 'crew','vote\_average' ]]

movies.head()

# missing values

movies.isnull().sum()

# Duplicated

movies.duplicated().sum()

# Removing rows with missing values

movies.dropna(inplace = True)

movies.shape

type(movies.iloc[0]['genres'])

# integer-location based indexing and returns a pandas series

movies.iloc[0]['genres']

# Convert str to list

# ast is used here ( abstract syntax tree, evaluates strings )

def convert(text):

l = []

for i in ast.literal\_eval(text): # converts a stringified list into an actual list of dicts

l.append(i['name']) # extracts the name from each dictionary

return l

movies['genres'] = movies['genres'].apply(convert)

movies.iloc[0]['keywords']

def convert(text):

l = []

for i in ast.literal\_eval(text):

l.append(i['name'])

return l

movies['keywords'] = movies['keywords'].apply(convert)

movies.iloc[0]['cast']

# We only took three casts to reduce dimensionality and keeping the most important casts

def convert(text):

l = []

counter = 0

for i in ast.literal\_eval(text):

if counter < 3:

l.append(i['name'])

counter += 1

return l

movies['cast'] = movies['cast'].apply(convert)

movies.iloc[0]['crew']

def convert(text):

l = []

for i in ast.literal\_eval(text):

if i['job'] == 'Director':

l.append(i['name'])

return l

movies['crew'] = movies['crew'].apply(convert)

# Splits the overview column into a list of words

# see the commas are added to each word, this helps in NLP tasks

# And this should not be done to the cast and crew column

movies['overview'] = movies['overview'].apply(lambda x:x.split())

movies['overview'].head()

def remove\_space(word):

l1 = []

for i in word:

l1.append(i.replace(" ","")) # from " " to ""

return l1

# Another method that checks if it is list before replacing

'''def remove\_space(word):

if isinstance(word, list): # Check if it's a list

return [i.replace(" ", "") for i in word] # Remove spaces from each element

else:

return word

movies['genres'] = movies['genres'].apply(remove\_space)

movies['keywords'] = movies['keywords'].apply(remove\_space)

movies['cast'] = movies['cast'].apply(remove\_space)

movies['crew'] = movies['crew'].apply(remove\_space)

movies.head()

'''

movies['genres'] = movies['genres'].apply(remove\_space)

movies['keywords'] = movies['keywords'].apply(remove\_space)

movies['cast'] = movies['cast'].apply(remove\_space)

movies['crew'] = movies['crew'].apply(remove\_space)

movies.head()

# Creating tags

movies['tags'] = movies['overview'] + movies["genres"] + movies["keywords"] + movies["cast"] + movies["crew"]

movies.iloc[0]['tags']

# Creating another df

movies\_new = movies[['movie\_id', 'title', 'tags', 'vote\_average']]

movies\_new.head()

#

movies\_new['tags'] = movies\_new['tags'].apply(lambda x: " ".join(x))

movies\_new.iloc[0]['tags']

# Lowercase

movies\_new['tags'] = movies\_new['tags'].apply(lambda x: x.lower())

movies\_new.iloc[0]['tags']

# nltk library

# Preparing text for counter vectorizer

import nltk

from nltk.stem import PorterStemmer

ps = PorterStemmer()

def stems(text):

l1 = []

for i in text.split():

l1.append(ps.stem(i))

return " ".join(l1)

movies\_new['tags'] = movies\_new['tags'].apply(stems)

movies\_new.iloc[0]['tags']

# COUNTER VECTORIZER

from sklearn.feature\_extraction.text import CountVectorizer

cv = CountVectorizer(max\_features = 5000, stop\_words = 'english')

vector = cv.fit\_transform(movies\_new['tags']).toarray()

vector

vector.shape

# COSINE SIMILARITY

from sklearn.metrics.pairwise import cosine\_similarity

similarity = cosine\_similarity(vector)

similarity

similarity.shape

movies\_new[movies\_new['title'] == "Spider-Man"].index[0]

# CREATING RECOMMEND FUNCTION

def recommend(movie):

index = movies\_new[movies\_new['title'] == movie].index[0]

distances = sorted(list(enumerate(similarity[index])), reverse = True, key = lambda x: x[1])

print("Top 5 Recommended Movies:")

for i in distances[1:6]: # Skip the first one since it's the input movie itself

print(movies\_new.iloc[i[0]].title)

import pickle

pickle.dump(movies\_new, open('artifacts/movie\_list.pkl', 'wb'))

pickle.dump(similarity, open('artifacts/similarity.pkl','wb'))