```
# Importing required librarie
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)


import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))


# This code we have taken the file as .csv and uploaded it
movies = pd.read_csv('tmdb_5000_movies.csv')
credits = pd.read_csv('tmdb_5000_credits.csv')


# This code reads two CSV files, 'tmdb_5000_movies.csv' and 'tmdb_5000_credits.csv', and stores the data in two DataFrames named 'movies' and
movies.head(2)


# This displays the first two rows of the 'movies' DataFrame, providing a glimpse of its contents.
movies.shape


# This shows the dimensions (number of rows and columns) of the 'movies' DataFrame.
credits.head()


# This displays the first few rows of the 'credits' DataFrame, showing some of its contents.
movies = movies.merge(credits,on='title')


# This merges the 'movies' and 'credits' DataFrames based on the 'title' column, combining their data into a new 'movies' DataFrame.
movies.head()


# This displays the first few rows of the 'movies' DataFrame after the merge operation.
movies = movies[['movie_id','title','overview','genres','keywords','cast','crew']]


# This creates a new 'movies' DataFrame containing only specific columns, including 'movie_id', 'title', 'overview', 'genres', 'keywords', 'c
movies.head()


# This displays the first few rows of the 'movies' DataFrame after selecting specific columns.
import ast


# This imports the 'ast' module, which provides functions to handle Abstract Syntax Trees (AST) in Python.
def convert(text):
    L = []
    for i in ast.literal_eval(text):
        L.append(i['name'])
    return L


# This defines a function 'convert' that takes a text input, evaluates it as a Python data structure (a list of dictionaries), and extracts t
movies.dropna(inplace=True)


# This removes any rows with missing values (NaN) from the 'movies' DataFrame, updating it in place.
movies['genres'] = movies['genres'].apply(convert)
movies.head()


# This applies the 'convert' function to each element of the 'genres' column in the 'movies' DataFrame. It extracts the 'name' key from the l
movies['keywords'] = movies['keywords'].apply(convert)
movies.head()


# This applies the 'convert' function to each element of the 'keywords' column in the 'movies' DataFrame. It extracts the 'name' key from the
import ast
ast.literal_eval('[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {"id": 878, "name": "Science


# This is an example of using 'ast.literal_eval' to evaluate a string representation of a list of dictionaries containing genre information.
def convert3(text):
    L = []
    counter = 0
    for i in ast.literal_eval(text):
        if counter < 3:
            L.append(i['name'])
        counter+=1
    return L
```

```
# This defines a function 'convert3' that takes a text input, evaluates it as a Python data structure (a list of dictionaries), and extracts 
movies['cast'] = movies['cast'].apply(convert)
movies.head()
```

```
# This applies the 'convert' function to each element of the 'cast' column in the 'movies' DataFrame. It extracts the 'name' key from the lis
movies['cast'] = movies['cast'].apply(lambda x:x[0:3])
```

```
# This applies a lambda function to each element of the 'cast' column in the 'movies' DataFrame. It keeps only the first three names in the l
def fetch_director(text):
    L = []
    for i in ast.literal_eval(text):
        if i['job'] == 'Director':
            L.append(i['name'])
    return L
```

```
# This defines a function 'fetch_director' that takes a text input, evaluates it as a Python data structure (a list of dictionaries), and fil
movies['crew'] = movies['crew'].apply(fetch_director)
```

```
movies.sample(5)
```

```
def collapse(L):
    L1 = []
    for i in L:
        L1.append(i.replace(" ",""))
    return L1
```

```
movies['cast'] = movies['cast'].apply(collapse)
movies['crew'] = movies['crew'].apply(collapse)
movies['genres'] = movies['genres'].apply(collapse)
movies['keywords'] = movies['keywords'].apply(collapse)
```

```
movies.head()
```

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

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

```
new = movies.drop(columns=['overview','genres','keywords','cast','crew'])
#new.head()
```

```
new['tags'] = new['tags'].apply(lambda x: " ".join(x))
new.head()
```

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=5000,stop_words='english')
```

```
vector = cv.fit_transform(new['tags']).toarray()
```

```
vector.shape
```

```
from sklearn.metrics.pairwise import cosine_similarity
```

```
similarity = cosine_similarity(vector)
```

```
similarity
```

```
new[new['title'] == 'The Lego Movie'].index[0]
```

```
# This code displays the 'similarity' 2D array, showing the pairwise cosine similarity values between different movies.
def recommend(movie):
    index = new[new['title'] == movie].index[0]
    distances = sorted(list(enumerate(similarity[index])),reverse=True,key = lambda x: x[1])
    for i in distances[1:6]:
        print(new.iloc[i[0]].title)
```

```
# This code finds the index of the row in the 'new' DataFrame where the movie title is 'The Lego Movie'.
recommend('Gandhi')
```

```
# This code defines a function 'recommend' that takes a movie title as input. It finds the index of the input movie in the 'new' DataFrame, t
import pickle
```

```
# This code calls the 'recommend' function with the movie title 'Gandhi', which will print the top 5 recommended movies similar to 'Gandhi'.
pickle.dump(new,open('movie_list.pkl','wb'))
pickle.dump(similarity,open('similarity.pkl','wb'))
```

```
# This code saves the 'new' DataFrame and 'similarity' array into separate pickle files named 'movie_list.pkl' and 'similarity.pkl', respecti
recommend("Batman Begins")
```