**Ans :**

**Python Code :**

## Recommendation System

########## Books Data Set #########

######### Recommendation based on Author ############

import pandas as pd

#import Dataset

book = pd.read\_csv("file:\\D:\\Data Science\\Excelr\\Assignments\\Assignment\\Recommendation System\\books1.csv")

book.shape #shape

book.columns

book.Author#ratings column

from sklearn.feature\_extraction.text import TfidfVectorizer #term frequencey- inverse document frequncy is a numerical statistic that is intended to reflect how important a word is to document in a collecion or corpus

# Creating a Tfidf Vectorizer to remove all stop words

tfidf = TfidfVectorizer(stop\_words="english") #taking stop words from tfid vectorizer

# replacing the NaN values in overview column with

# empty string

book["Author"].isnull().sum()

book["Author"] = book["Author"].fillna(" ")

# Preparing the Tfidf matrix by fitting and transforming

tfidf\_matrix = tfidf.fit\_transform(book.Author) #Transform a count matrix to a normalized tf or tf-idf representation

tfidf\_matrix.shape #12294,46

# with the above matrix we need to find the

# similarity score

# There are several metrics for this

# such as the euclidean, the Pearson and

# the cosine similarity scores

# For now we will be using cosine similarity matrix

# A numeric quantity to represent the similarity

# between 2 movies

# Cosine similarity - metric is independent of

# magnitude and easy to calculate

# cosine(x,y)= (x.y⊺)/(||x||.||y||)

from sklearn.metrics.pairwise import linear\_kernel

# Computing the cosine similarity on Tfidf matrix

cosine\_sim\_matrix = linear\_kernel(tfidf\_matrix,tfidf\_matrix)

# creating a mapping of book name to index number

book\_index = pd.Series(book.index,index=book['Title']).drop\_duplicates()

book\_index["Clara Callan"]

def get\_book\_recommendations(Title,topN):

#topN = 10

# Getting the movie index using its title

book\_id = book\_index[Title]

# Getting the pair wise similarity score for all the book's with that

# book

cosine\_scores = list(enumerate(cosine\_sim\_matrix[book\_id]))

# Sorting the cosine\_similarity scores based on scores

cosine\_scores = sorted(cosine\_scores,key=lambda x:x[1],reverse = True)

# Get the scores of top 10 most similar book's

cosine\_scores\_10 = cosine\_scores[0:topN+1]

# Getting the book index

book\_idx = [i[0] for i in cosine\_scores\_10]

book\_scores = [i[1] for i in cosine\_scores\_10]

# Similar movies and scores

book\_similar\_show = pd.DataFrame(columns=["Title","Score"])

book\_similar\_show["Title"] = book.loc[book\_idx,"Title"]

book\_similar\_show["Score"] = book\_scores

book\_similar\_show.reset\_index(inplace=True)

book\_similar\_show.drop(["index"],axis=1,inplace=True)

print (book\_similar\_show)

#return (book\_similar\_show)

# Enter your book and number of book's to be recommended

get\_book\_recommendations("Classical Mythology",topN=20)

######### Recommendation based on Publisher ############

import pandas as pd

#import Dataset

book = pd.read\_csv("file:\\D:\\Data Science\\Excelr\\Assignments\\Assignment\\Recommendation System\\books1.csv")

book.shape #shape

book.columns

book.Publisher#ratings column

from sklearn.feature\_extraction.text import TfidfVectorizer #term frequencey- inverse document frequncy is a numerical statistic that is intended to reflect how important a word is to document in a collecion or corpus

# Creating a Tfidf Vectorizer to remove all stop words

tfidf = TfidfVectorizer(stop\_words="english") #taking stop words from tfid vectorizer

# replacing the NaN values in overview column with

# empty string

book["Publisher"].isnull().sum()

book["Publisher"] = book["Publisher"].fillna(" ")

# Preparing the Tfidf matrix by fitting and transforming

tfidf\_matrix = tfidf.fit\_transform(book.Publisher) #Transform a count matrix to a normalized tf or tf-idf representation

tfidf\_matrix.shape #12294,46

# with the above matrix we need to find the

# similarity score

# There are several metrics for this

# such as the euclidean, the Pearson and

# the cosine similarity scores

# For now we will be using cosine similarity matrix

# A numeric quantity to represent the similarity

# between 2 movies

# Cosine similarity - metric is independent of

# magnitude and easy to calculate

# cosine(x,y)= (x.y⊺)/(||x||.||y||)

from sklearn.metrics.pairwise import linear\_kernel

# Computing the cosine similarity on Tfidf matrix

cosine\_sim\_matrix = linear\_kernel(tfidf\_matrix,tfidf\_matrix)

# creating a mapping of book name to index number

book\_index = pd.Series(book.index,index=book['Title']).drop\_duplicates()

book\_index["Clara Callan"]

def get\_book\_recommendations(Title,topN):

#topN = 10

# Getting the movie index using its title

book\_id = book\_index[Title]

# Getting the pair wise similarity score for all the book's with that

# book

cosine\_scores = list(enumerate(cosine\_sim\_matrix[book\_id]))

# Sorting the cosine\_similarity scores based on scores

cosine\_scores = sorted(cosine\_scores,key=lambda x:x[1],reverse = True)

# Get the scores of top 10 most similar book's

cosine\_scores\_10 = cosine\_scores[0:topN+1]

# Getting the book index

book\_idx = [i[0] for i in cosine\_scores\_10]

book\_scores = [i[1] for i in cosine\_scores\_10]

# Similar movies and scores

book\_similar\_show = pd.DataFrame(columns=["Title","Score"])

book\_similar\_show["Title"] = book.loc[book\_idx,"Title"]

book\_similar\_show["Score"] = book\_scores

book\_similar\_show.reset\_index(inplace=True)

book\_similar\_show.drop(["index"],axis=1,inplace=True)

print (book\_similar\_show)

#return (book\_similar\_show)

# Enter your book and number of book's to be recommended

get\_book\_recommendations("Classical Mythology",topN=10)

**Results :**

**######### Recommendation Based on Author ##########**

import pandas as pd

#import Dataset

book = pd.read\_csv("file:\\D:\\Data Science\\Excelr\\Assignments\\Assignment\\Recommendation System\\books1.csv")

book.shape #shape

Out[34]: (5000, 5)

book.columns

Out[35]: Index(['users[, 1]', 'Title', 'Author', 'Publisher', 'ratings'], dtype='object')

book.Author#ratings column

Out[36]:

0 Mark P. O. Morford

1 Richard Bruce Wright

2 Carlo D'Este

3 Gina Bari Kolata

4 E. J. W. Barber

4995 Daniella Chace

4996 Bryce Courtenay

4997 David Smith

4998 Carole Bloom

4999 Salman Rushdie

Name: Author, Length: 5000, dtype: object

from sklearn.feature\_extraction.text import TfidfVectorizer #term frequencey- inverse document frequncy is a numerical statistic that is intended to reflect how important a word is to document in a collecion or corpus

# Creating a Tfidf Vectorizer to remove all stop words

tfidf = TfidfVectorizer(stop\_words="english") #taking stop words from tfid vectorizer

# replacing the NaN values in overview column with

# empty string

book["Author"].isnull().sum()

Out[42]: 0

book["Author"] = book["Author"].fillna(" ")

# Preparing the Tfidf matrix by fitting and transforming

tfidf\_matrix = tfidf.fit\_transform(book.Author) #Transform a count matrix to a normalized tf or tf-idf representation

tfidf\_matrix.shape #12294,46

Out[46]: (5000, 8438)

# with the above matrix we need to find the

# similarity score

# There are several metrics for this

# such as the euclidean, the Pearson and

# the cosine similarity scores

# For now we will be using cosine similarity matrix

# A numeric quantity to represent the similarity

# between 2 movies

# Cosine similarity - metric is independent of

# magnitude and easy to calculate

# cosine(x,y)= (x.y⊺)/(||x||.||y||)

from sklearn.metrics.pairwise import linear\_kernel

# Computing the cosine similarity on Tfidf matrix

cosine\_sim\_matrix = linear\_kernel(tfidf\_matrix,tfidf\_matrix)

# creating a mapping of book name to index number

book\_index = pd.Series(book.index,index=book['Title']).drop\_duplicates()

book\_index["Clara Callan"]

Out[63]: 1

def get\_book\_recommendations(Title,topN):

#topN = 10

# Getting the movie index using its title

book\_id = book\_index[Title]

# Getting the pair wise similarity score for all the book's with that

# book

cosine\_scores = list(enumerate(cosine\_sim\_matrix[book\_id]))

# Sorting the cosine\_similarity scores based on scores

cosine\_scores = sorted(cosine\_scores,key=lambda x:x[1],reverse = True)

# Get the scores of top 10 most similar book's

cosine\_scores\_10 = cosine\_scores[0:topN+1]

# Getting the book index

book\_idx = [i[0] for i in cosine\_scores\_10]

book\_scores = [i[1] for i in cosine\_scores\_10]

# Similar movies and scores

book\_similar\_show = pd.DataFrame(columns=["Title","Score"])

book\_similar\_show["Title"] = book.loc[book\_idx,"Title"]

book\_similar\_show["Score"] = book\_scores

book\_similar\_show.reset\_index(inplace=True)

book\_similar\_show.drop(["index"],axis=1,inplace=True)

print (book\_similar\_show)

#return (book\_similar\_show)

get\_book\_recommendations("Classical Mythology",topN=20)

Title Score

0 Classical Mythology 1.000000

1 Fishboy: A Ghost's Story 0.404020

2 The Adventures of Tom Sawyer 0.346620

3 Adventures of Huckleberry Finn (Signet Classic... 0.346620

4 Adventures of Huckleberry Finn (Dover Thrift E... 0.346620

5 A Connecticut Yankee in King Arthur's Court 0.346620

6 Adventures of Huckleberry Finn 0.346620

7 The Adventures of Tom Sawyer 0.346620

8 A Connecticut Yankee in King Arthur's Court (D... 0.346620

9 The Adventures of Tom Sawyer (Penguin Popular ... 0.346620

10 Adventures of Tom Sawyer 0.346620

11 The Diaries of Adam and Eve 0.346620

12 The Diaries of Adam and Eve (Literary Classics) 0.346620

13 Living on the Borders: What the Church Can Lea... 0.338950

14 COPS 0.335935

15 The Fractal Murders (Pepper Keane Mysteries) 0.324675

16 Lying Awake 0.319788

17 Lost in Place: Growing Up Absurd in Suburbia 0.319788

18 True Notebooks (Alex Awards (Awards)) 0.319788

19 Lying Awake 0.319788

20 A Soldier of the Great War 0.313953

**######### Recommendation Based on Publisher ##########**

import pandas as pd

#import Dataset

book = pd.read\_csv("file:\\D:\\Data Science\\Excelr\\Assignments\\Assignment\\Recommendation System\\books1.csv")

book.shape #shape

Out[70]: (5000, 5)

book.columns

Out[71]: Index(['users[, 1]', 'Title', 'Author', 'Publisher', 'ratings'], dtype='object')

book.Publisher#ratings column

Out[72]:

0 Oxford University Press

1 HarperFlamingo Canada

2 HarperPerennial

3 Farrar Straus Giroux

4 W. W. Norton &amp; Company

4995 Prima Lifestyles

4996 Ballantine Books

4997 Disney Editions

4998 Harpercollins

4999 Penguin USA

Name: Publisher, Length: 5000, dtype: object

from sklearn.feature\_extraction.text import TfidfVectorizer #term frequencey- inverse document frequncy is a numerical statistic that is intended to reflect how important a word is to document in a collecion or corpus

# Creating a Tfidf Vectorizer to remove all stop words

tfidf = TfidfVectorizer(stop\_words="english") #taking stop words from tfid vectorizer

# replacing the NaN values in overview column with

# empty string

book["Publisher"].isnull().sum()

Out[78]: 0

book["Publisher"] = book["Publisher"].fillna(" ")

# Preparing the Tfidf matrix by fitting and transforming

tfidf\_matrix = tfidf.fit\_transform(book.Publisher) #Transform a count matrix to a normalized tf or tf-idf representation

Out[82]: (5000, 5822)

tfidf\_matrix.shape #12294,46

# with the above matrix we need to find the

# similarity score

# There are several metrics for this

# such as the euclidean, the Pearson and

# the cosine similarity scores

# For now we will be using cosine similarity matrix

# A numeric quantity to represent the similarity

# between 2 movies

# Cosine similarity - metric is independent of

# magnitude and easy to calculate

# cosine(x,y)= (x.y⊺)/(||x||.||y||)

from sklearn.metrics.pairwise import linear\_kernel

# Computing the cosine similarity on Tfidf matrix

cosine\_sim\_matrix = linear\_kernel(tfidf\_matrix,tfidf\_matrix)

# creating a mapping of book name to index number

book\_index = pd.Series(book.index,index=book['Title']).drop\_duplicates()

book\_index["Clara Callan"]

Out[99]: 1

def get\_book\_recommendations(Title,topN):

#topN = 10

# Getting the movie index using its title

book\_id = book\_index[Title]

# Getting the pair wise similarity score for all the book's with that

# book

cosine\_scores = list(enumerate(cosine\_sim\_matrix[book\_id]))

# Sorting the cosine\_similarity scores based on scores

cosine\_scores = sorted(cosine\_scores,key=lambda x:x[1],reverse = True)

# Get the scores of top 10 most similar book's

cosine\_scores\_10 = cosine\_scores[0:topN+1]

# Getting the book index

book\_idx = [i[0] for i in cosine\_scores\_10]

book\_scores = [i[1] for i in cosine\_scores\_10]

# Similar movies and scores

book\_similar\_show = pd.DataFrame(columns=["Title","Score"])

book\_similar\_show["Title"] = book.loc[book\_idx,"Title"]

book\_similar\_show["Score"] = book\_scores

book\_similar\_show.reset\_index(inplace=True)

book\_similar\_show.drop(["index"],axis=1,inplace=True)

print (book\_similar\_show)

#return (book\_similar\_show)

get\_book\_recommendations("Classical Mythology",topN=10)

Title Score

0 Classical Mythology 1.0

1 What a Wonderful World: A Lifetime of Recordings 1.0

2 Julius Caesar (Oxford School Shakespeare) 1.0

3 Cranford (The World's Classics) 1.0

4 How Not to Say What You Mean: A Dictionary of ... 1.0

5 Kidnapped and Catriona (World Classics) 1.0

6 Metaphysical Lyrics Poems 17 Cen 1.0

7 The Selfish Gene 1.0

8 Dada and Surrealism: A Very Short Introduction... 1.0

9 No Name (World's Classics) 1.0

10 Tibetan Book of the Dead 1.0

**Inference :**

Successfully created recommender system.