

Recommendation System

aineshsootha.pythonanywhere.com (live version)

ENGR-133 Final Project

Ainesh Sootha

This project is a **book/movie recommendation system** written in **Python** (Flask). It uses the CMU book summaries dataset for books and the Kaggle movies dataset for movies.* The program uses Pandas (Python data analysis library) to work with the datasets. The program uses an algorithm called '**cosine similarity**' to find similar books to recommend. The algorithm is explained in later paragraphs.

*The datasets are originally in csv but are converted using the csvtohdf.py file to hdf5 files since hdf5 is faster to access than csv.

Books

The program allows the user to enter the title of the book they have read and choose what they liked about the book. They may choose **Genre** or **Writing Style**. Each of these options prompts the program to call a different function. If the user chooses Genre, the program would return a list of books with similar genres **sorted according to the similarity of their summaries**. Similarly, if the user chooses Writing Style, the program would return a list of books from the same author **sorted according to the similarity of their summaries**.

The similarity is computed using the cosine similarity algorithm which is explained in later paragraphs.

Movies

The program allows the user to enter the title of the movie they have watched and choose what they liked about the movie. They may choose **Genre**, **Cast** or **Collection**. Each of these options prompts the program to call a different function. If the user chooses Genre, the program would return a list of movies with similar genres **sorted according to the similarity of their summaries**.

*Collection refers to a franchise or series like Toy Story or Harry Potter

The similarity is computed using the cosine similarity algorithm (A ML algorithm) which is explained in later paragraphs.

Other details

The UI of the program is created using Flask + HTML/CSS/JS. I used flask primarily because it is very lightweight, minimalist, whereas Django is a heavy, batteries-included, full-stack framework. This would have added too many features I wouldn't need.

The design is completely responsive (designed in bootstrap) and can be used on any device (iPhone, iPad etc.)

I chose to host the project on Pythonanywhere.com since it is easier to go from development straight to production without the requirement of virtual environments and without the requirement of my laptop being present at the time of use.

The program uses the Goodreads API (for books) which generates an XML for the title, which is converted to a multi-level dictionary. This is used to get a link to the poster of the book. This is then displayed in the result page.

Similarly, the program uses the OMDb API (for movies) which generates a json for the title, which is converted to a dictionary. Then the link of the poster for the movie is obtained from this dictionary and the poster (from the link) is displayed on the result page.

If a book/movie title is entered such that it doesn't exist in the dataset(s), the user is shown an error screen and the title entered is appended to either missingbook.csv or missingmovie.csv (depending on whether user chose book or movie). This is done to create reference files for me to add new titles into the dataset.

Cosine Similarity

Similarity, for shorter strings, can be measured by simply counting the number of times each word appears in the string.

Eg:

Mark and Andy play with the balls.

Andy and Mark play with a ball.

If we computed the similarity of these 2 strings by counting the repeating words, they would be almost 100% similar. But, there is an inherent flaw with this method, especially when used with larger files. Since words may be repeated more frequently in a large string, the comparison using this method becomes less accurate. Thus, I used the cosine similarity method, a **Machine Learning algorithm used to compute the similarity of large strings/documents**.

This algorithm, as the name suggests, is based on the cosine of vectors.

$$\mathbf{A} \cdot \mathbf{B} = \|\mathbf{A}\| \|\mathbf{B}\| \cos \theta$$

Where A,B are 2 vectors.

Thus, $\cos\theta$ is given by

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}},$$

To put it simply, here the vectors are the strings to compare. The 2 strings are converted to vectors, or sets of words (dictionaries in Python, with each number representing the number of times the word is repeated)

“When plotted on a multi-dimensional space, where each dimension corresponds to a word in the document, the cosine similarity captures the orientation (the angle) of the documents and not the magnitude.” – (From machinelearningplus.com)

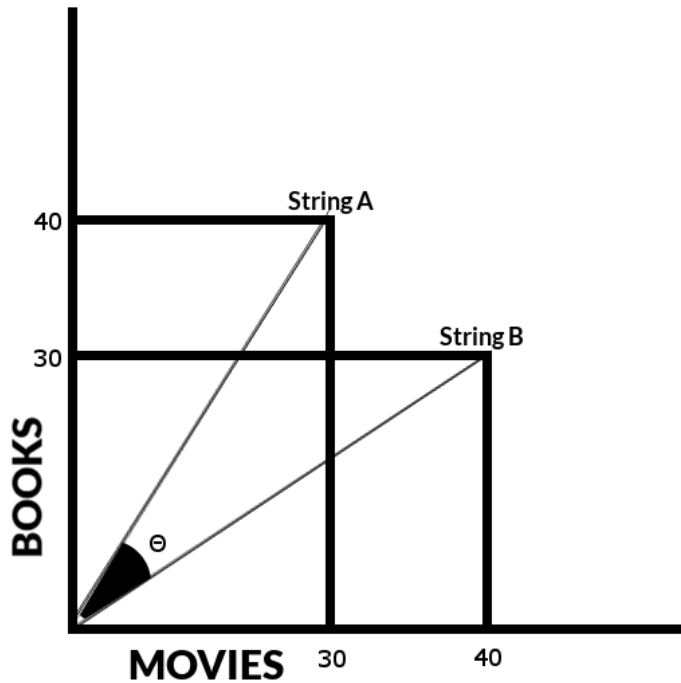
Basically, each word creates an axis in a multi dimensional space, and the vector of each string (computed using the number of times each word is repeated) can be plotted on this multi-dimensional space. If the 2 strings have vectors pointing in the same direction, it represents that these 2 strings are very similar in meaning. Thus, they have a high cosine similarity value. (Lower angle = Higher cosine).

To make it easier to understand, here is an example:

If there are 2 strings, consisting of the words 'books' and 'movies' only, with:

- String A having 40 repetitions of the word 'books' and 20 repetitions of 'movies'
- String B having 30 repetitions of the word 'books' and 30 repetitions of 'movies'

If we plot these on a 2D plot with one axis being 'books' and one being 'movies', the graph would look like this:



It can be observed here that the angle theta is used to measure the cosine similarity of the 2 strings. Once more words are added, more axes are created and thus the accuracy of the similarity algorithm improves. For the similarity to be higher, theta is smaller, thus the cosine of the angle is closer to 1.

This is very useful in computing the similarity of books and movies because the summaries of books and movies tell us a lot about the books/movies themselves. For example, a fantasy book would have words like "wizard", "dragons" etc, so other books with similar words (with the same meaning) would be closer (in terms of vector in the multidimensional space).

Thus, I used this method to compute similar books/ movies.

Overview

The program consists of 3 main files:

- **Project.py**
 - This is the main file, the flask app. It renders the HTML templates (index.html, result.html, error.html) as required and calls the functions findbook() and findmovie() from the books and movies modules respectively.
 - It gets the user input from index.html (from the main form), where the user chooses whether they want to enter book or movie, the title of the book/movie and finally what they liked about the book/movie. This is received using the flask request module and then the required function (findbook() or findmovie()) is called depending on the user input for the choice of book/movie.
 - It also calls findposter() for either book or movie as required.
 - It renders the error.html template if the list of books is empty (suggesting that the dataset is empty) or if the movie is not a part of any collection/series. It also calls writemissing() from missing.py if an error is encountered.
 - If no errors are encountered, it renders the result.html template and populates it with data returned from the functions.
- **Books.py**
 - This is the file for finding similar books.
 - The main function of the file is findbooks(), which requires the title of the main book, and it returns sorted lists of book titles, one list with similar genres, and one with the same author as the main book.
 - It uses pandas to read the hdf5 file which contains the dataset of books (converted from csv) and find the author/genres as required and then creates a list of books with the same author/genre as required, and finally sorts the lists based on the cosine similarity of the book summaries.
 - The dataset used is the CMU book summaries dataset.
 - It also uses the goodreads API to find the poster link to the book being searched. It uses the requests module to work with the API, which returns XML data about the book. This data is converted to a dictionary using the function xmltodict() (from PyPI) and then the dict is used to locate the poster image.
- **Movies.py**
 - This is the file for finding similar movies
 - The main function of the file is findmovies(), which requires the title of the main movie, and it returns sorted lists of movie titles, one list with similar genres, one with similar cast, and one from the same series/collection.
 - It uses pandas to read the hdf5 file which contains the dataset of movies (converted from csv) and find the genres/cast/collection as required and then creates a list of movies with the same genres/cast/collection as required, and finally sorts the list based on the cosine similarity of the book summaries.
 - The dataset used is the Kaggle movies dataset.
 - It also uses the OMDb API to find the poster link to the movie being searched. It uses the requests module to work with the API, which returns JSON data about the book. This

data is converted to a dictionary using the `.json()` method and then the dict is used to locate the poster image.

Other files:

- **Missing.py**
 - The main function of this module (`writemissing()`) is used to append the title of the book/movie to the appropriate file if an error is observed.
- **Index.html**
 - This is the main template file.
 - It contains the main form and sends the data to flask from this form using the POST method.
- **Result.html**
 - This is the result template file.
 - It gets the values from the `Project.py` file and displays the details as required.
- **Error.html**
 - This is the error template file.
 - It is rendered if an error is encountered. The message displayed depends on the error.
 - Possible errors are:
 - Book/Movie not present in dataset
 - Movie is not part of any collection/series
- **Main.css**
 - This is the css file for all the templates
- **Csvto hdf.py**
 - This is a file I created to convert the csv dataset files to hdf once I learned that hdf5 files are faster to access
- **Datasetbooks.csv / datasetbook.h5**
 - These are the CMU book datasets I am using for the books module
- **Datasetmovies.csv / datasetmovies.h5**
 - These are the Kaggle movie datasets I am using for the movies module
- **Datasetcast.csv / datasetcast.h5**
 - These are the Kaggle movie datasets I am using for finding the cast in the movies module

Detailed Description (Functions)

`Project.py`:

- **home()**
 - This is a flask function that renders the template for the home page of the site.
- **result()**
 - This is a function that calls all the required functions from the other files (`movies.py`, `books.py` etc) and generates the required list of books and movies. It then renders the result template (`result.html`) and prints the required items.
 - It can also render the error template (`error.html`) which is rendered if the function is unable to find the book/movie or if the movie doesn't belong to any collection.

Movies.py:

- `get_cosine()`
 - This is the function that calculates the cosine similarities of the 2 vectors (as explained in the cosine similarity paragraph).
 - It returns the cosine similarity of the 2 vectors.
- `text_to_vector()`
 - This converts the strings to vectors (dictionaries) and removes any special characters etc.
- `findGenre()`
 - This finds the genre of the movie using pandas commands.
 - It processes the genre string to create a readable list.
- `findCollection()`
 - This finds the collections that the movie belongs to.
 - It processes the collections string and creates a readable list.
- `findCast()`
 - It uses datacast.h5 dataset to find the appropriate cast of the movie.
 - It processes and creates a readable list of cast members.
- `findMoviesGenre()`
 - It finds all the movies with the same genre as the main movie.
 - It then sorts the list of movies according to the similarity of the summary to the summary of the main movie (using cosine similarity)
- `findMoviesCast()`
 - It finds all the movies with the same cast as the main movie.
 - It then sorts the list of movies according to the similarity of the summary to the summary of the main movie (using cosine similarity)
- `findPoster()`
 - It uses the OMDb API to find the poster link for the movie. It gets the data as a json, and converts it to a dict and then uses that data to find the poster image of the movie.
- `findMoviesCollection()`
 - It finds all the movies from the collection.
 - It then sorts the list of movies according to the similarity of the summary to the summary of the main movie (using cosine similarity)
- `findMovies()`
 - It calls all the required functions.
 - This is the main driving function of the movies.py file.
 - It returns the lists from all the functions, allowing the main program to print these lists.

Books.py

- `get_cosine()`
 - This is the function that calculates the cosine similarities of the 2 vectors (as explained in the cosine similarity paragraph).
 - It returns the cosine similarity of the 2 vectors.
- `text_to_vector()`

- This converts the strings to vectors (dictionaries) and removes any special characters etc.
- findGenre()
 - This finds the genre of the book using pandas commands.
 - It processes the genre string to create a readable list.
- findAuthor()
 - This finds the author of the book using pandas.
 - It processes the author as a string and creates a readable list.
- findBooksGenre()
 - It finds all the books with the same genre as the main book.
 - It then sorts the list of books according to the similarity of the summary to the summary of the main book (using cosine similarity)
- findPoster()
 - It uses the GoodReads API to find the poster link for the book. It gets the data in XML format, and converts it to a dict and then uses that data to find the poster image of the book.
- findMoviesAuthor()
 - It finds all the movies from the author.
 - It then sorts the list of books according to the similarity of the summary to the summary of the main book (using cosine similarity)
- findBooks()
 - It calls all the required functions.
 - This is the main driving function of the books.py file.
 - It returns the lists from all the functions, which can be used by the main file to print.

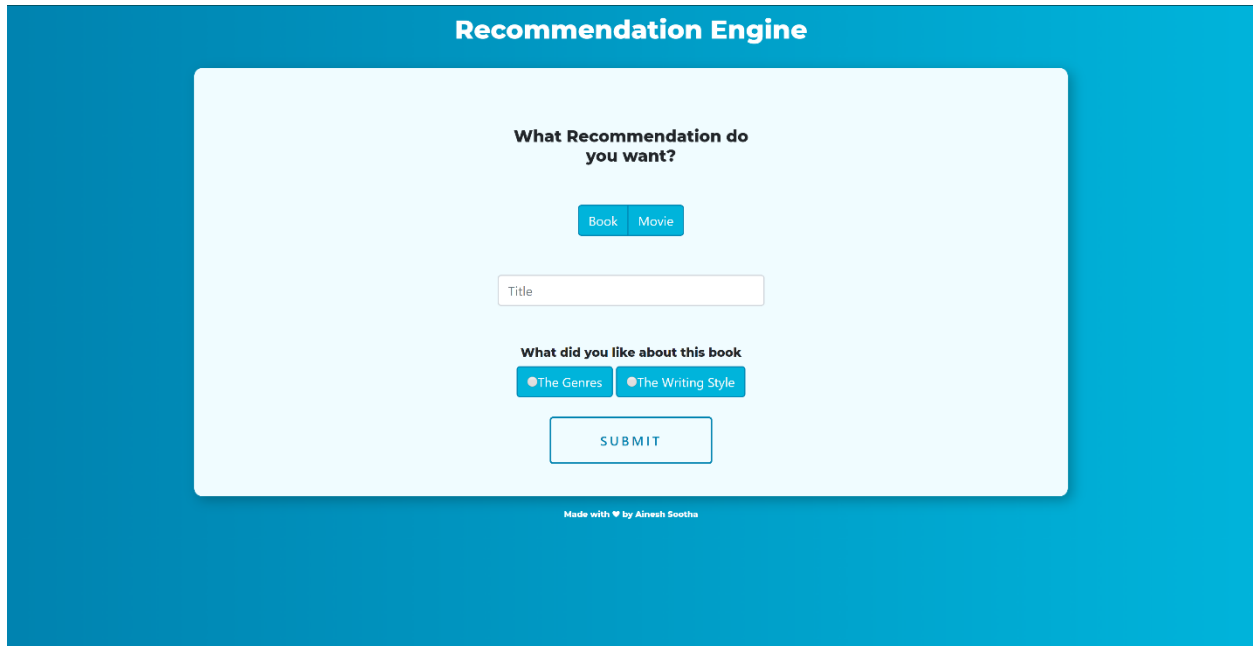
Missing.py

- writemissing()
 - It is called by the main file (Project.py) when the title entered is not found. It checks if the user picked movie or book, and then writes the title of the movie/book into the appropriate file (missingbook.csv OR missingmovie.csv) (append mode). This allows me to go and check which book/movie should be added to the dataset.

USER MANUAL

Steps for using the system:

1. Open <https://aineshsootha.pythonanywhere.com/>



The screenshot shows a web application titled "Recommendation Engine" on a blue background. The main content area is a light blue box with the following elements:

- What Recommendation do you want?**: A heading followed by two buttons: "Book" and "Movie".
- Title**: A text input field.
- What did you like about this book**: A heading followed by two radio buttons: "The Genres" (selected) and "The Writing Style".
- SUBMIT**: A button.
- Made with ♥ by Ainesh Sootha: A footer note.

2. Select the option you wish to use for the recommendation system (book/movie)
3. Enter the title of the book/movie
4. Select what you liked about the book/movie

5. Hit Submit

Recommendation Engine

What Recommendation do you want?

What did you like about this book

☐ The Genres ☒ The Writing Style


Made with ♥ by Ainesh Sootha

6. (a) Your recommendations appear!

Your Recommendations

[Go Back](#)

I think that this is the book you entered:



harry potter and the goblet of fire

You might like some of these books:

- deathstalker destiny
- deathstalker
- created by
- angel light
- harry potter and the philosopher's stone

Made with ♥ by Ainesh Sootha

6. (b) Error screen appears

Error!

[Go Back](#)

This book is not in my dataset currently! Sorry!

Made with ♥ by Ainesh Sootha

*In this case the title is appended to the missingbook / missingmovie .csv file. As shown:

(When I entered 'abcde')



7. You can hit 'Go Back' at the top and get more recommendations.

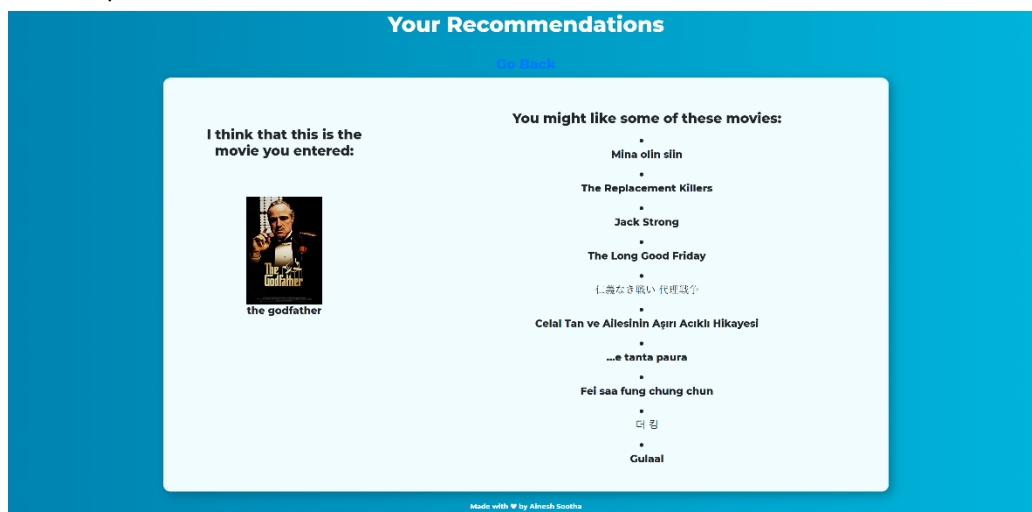
*Some recommendations that I have tried (are present the dataset):

- And then there were none (book)
- Any harry potter book / movie
- Oliver twist (book)
- Jumanji (movie)
- Toy story (movie)
- Things fall apart (book)

*There are 1million + books in the dataset and 400k+ movies in the dataset.

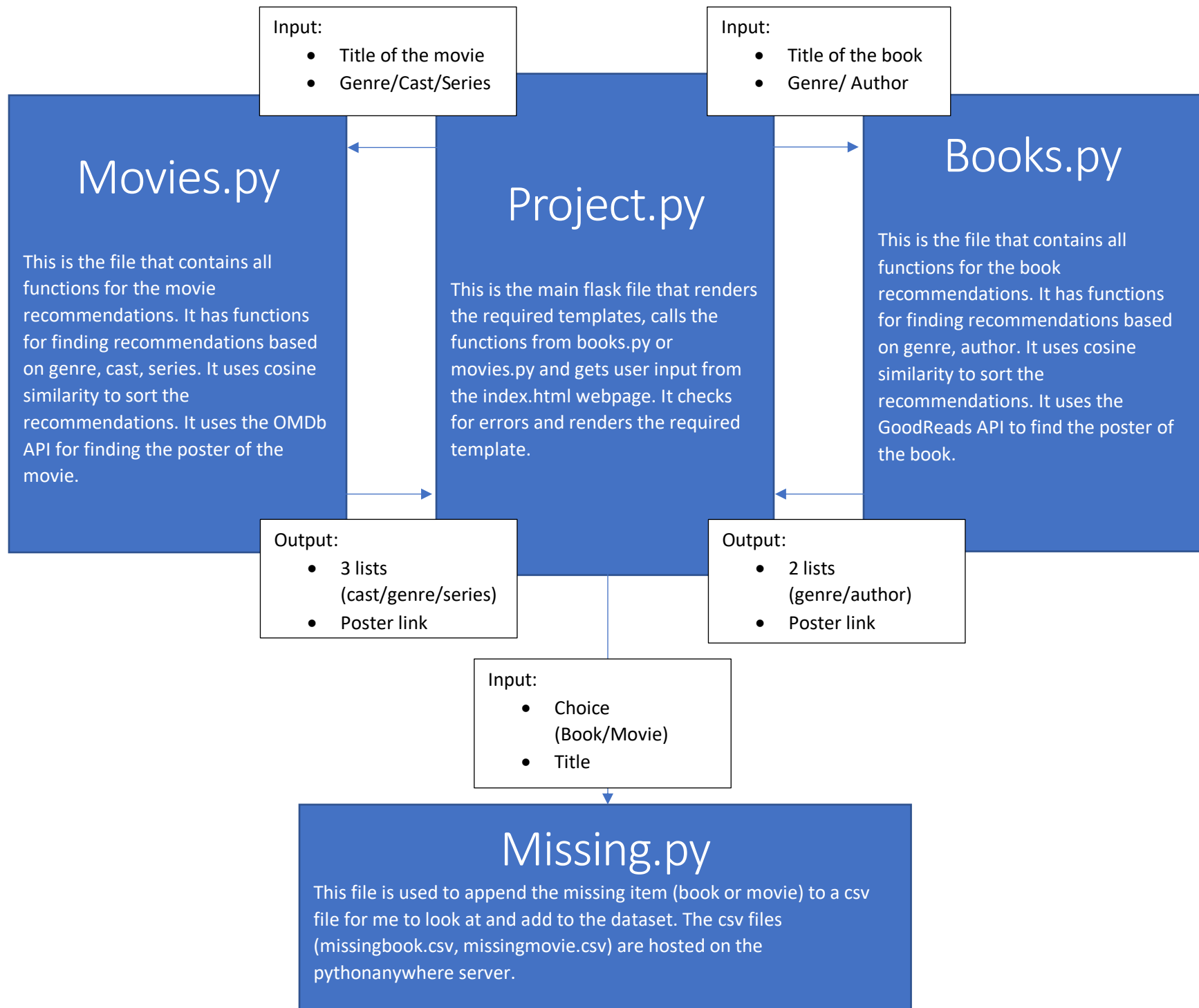
To view the error screen, you can enter a random string like 'abc' or a book that is not present in the dataset / movie that is not present in the dataset, like "the glass castle" (book).

*Keep in mind that the datasets have books/movies of various languages, so you may get recommendations in other languages, of books/movies that are very similar (in terms of the summaries) to the main book/movie.



*You can also use this on your phone/ iPad etc!

Interaction Diagram on Next Page



APPENDIX

(For code you may also visit <https://github.com/AineshSootha/Recommender>)

missing.py

```
'''
=====
ENGR 133 Program Description
    This file appends the missing titles to the appropriate file
    (missingmovie.csv or missingbook.csv)

Assignment Information
    Assignment:      Final Project
    Author:          Ainesh Sootha, asootha@purdue.edu
    Team ID:         001-14

Contributor:

    My contributor(s) helped me:
    [ ] understand the assignment expectations without
        telling me how they will approach it.
    [ ] understand different ways to think about a solution
        without helping me plan my solution.
    [ ] think through the meaning of a specific error or
        bug present in my code without looking at my code.
    Note that if you helped somebody else with their code, you
    have to list that person as a contributor here as well.
=====
'''

#This function appends the missing title to the appropriate csv file (missingmovi
e.csv or missingbook.csv)
#It requires the title and the choice (book/movie)
def writemissing(title,bm):
    if(bm=='movie'):
        with open('missingmovie.csv','a') as f:
            f.write(title)
    else:
        with open('missingbook.csv','a') as f:
            f.write(title)

'''
=====
ACADEMIC INTEGRITY STATEMENT
```

I have not used source code obtained from any other unauthorized source, either modified or unmodified. Neither have I provided access to my code to another. The project I am submitting is my own original work.

=====

'''

books.py

```
'''
=====
ENGR 133 Program Description
    This is the file that finds the book titles similar to the title entered based on the option picked.
    It then computes the cosine similarity and sorts the list of titles according to the similarity.

Assignment Information
    Assignment:      Final Project
    Author:          Ainesh Sootha, asootha@purdue.edu
    Team ID:         001-14

Contributor:

    My contributor(s) helped me:
    [ ] understand the assignment expectations without
        telling me how they will approach it.
    [ ] understand different ways to think about a solution
        without helping me plan my solution.
    [ ] think through the meaning of a specific error or
        bug present in my code without looking at my code.
    Note that if you helped somebody else with their code, you
    have to list that person as a contributor here as well.
=====
'''

import pandas as pd
import numpy as np
import re, math
from collections import Counter
import requests
import xmltodict

#This reads the hdf5 file (dataset) as a pandas dataframe. I am using hdf5 instead of csv because it is faster to access
data1 = pd.read_hdf("datasetBooks.h5")
#This basically breaks lines into words (excluding symbols, numerals etc)
WORD = re.compile(r'\w+')
#This function computes the cosine similarity for 2 vectors
#It requires 2 arguments (2 vectors computed using text_to_vector())
def get_cosine(vec1, vec2):
```



```

    #This finds similarity for each word and creates a list
    intersection = set(vec1.keys()) & set(vec2.keys())
    #The rest of this function basically computes the "dot product" of the 2 vec
tors and divides it with the magnitude of the list
    numerator = sum([vec1[x] * vec2[x] for x in intersection])
    sum1 = sum([vec1[x]**2 for x in vec1.keys()])
    sum2 = sum([vec2[x]**2 for x in vec2.keys()])
    denominator = math.sqrt(sum1) * math.sqrt(sum2)
    if not denominator:
        return 0.0
    else:
        return float(numerator) / denominator

#This function converts the string(text) to vectors (Dictionary of all words with
their respective count)
def text_to_vector(text):
    words = WORD.findall(text)
    return Counter(words)

#This function finds the author of the book from the dataset
#It requires the title of the book as a parameter
def findAuthor(title):
    #This creates a pandas series with all the rows with the same title as the pa
rameter
    rows = data1[data1['title'] == title.lower()]
    #This then matches the author column of the title and stores the author as a
list into author
    author = list(rows['Author'].values)
    #Finally the author(list) is returned
    return author

#This fuction finds the list of genres of the book from the dataset
#It requires the title of the book as a parameter
def findGenre(title):
    #This creates a pandas series with all the rows with the same title as the pa
rameter
    rows = data1[data1['title'] == title.lower()]
    #This then matches the author column of the title and stores the genres as a
list into genrelist
    genreset = list(rows['Genre'].values)
    l=[]
    linit=[]
    #Since in the dataset, the genres are separated using ", this loop is checks
if the list is empty, if not, it separates the strings
    for i in genreset:
        if(str(i).lower()=='nan'):

```

```

        return None
    linit=i.split('')
    #Finally, each genre is appended to the final list (l) after being checked for
    #random values that may occur
    for i in linit:
        if(': ' not in i and '/m/' not in i and ', ' not in i):
            l.append(i)
    l=list(filter(None,l))
    return l

```

#This function finds books with the same genre(s) as the book that is entered. It returns a sorted list of titles(based on cosine similarity)
 #It requires 2 arguments, the list of genre(s) of the book and the title of the book

```

def findBooksGenre(genrelist,title):
    #This creates a pandas series from all the genres in the list
    for i in genrelist:
        rows=(data1[data1['Genre'].str.contains(i,na=False)])
    l=[]
    #This creates a list of 0s (same number of 0s as rows from the series)
    for index,i in rows.iterrows():
        l.append(0)
        for j in genrelist:
            #This conditional adds 1 for every genre matched in the list of genres
            #from the book and the series created
            if(j in i['Genre']):
                l[-1]=l[-1]+1
    titleset=[]
    summaryset=[]
    j=-1
    #This iterates through the rows and creates 2 new lists (titles and summaries)
    for index,i in rows.iterrows():
        j=j+1
        #This conditional checks if the number of genres matching the main book is
        #close to the maximum in the list
        if(l[j]>0 and (l[j]==max(l) or l[j]==max(l)-1)):
            titleset.append(i['title'])
            summaryset.append(i['Summary'])
        #Here I created the mainrow which contains all the values of the main book from
        #the dataset
        mainrow=(data1[data1['title'].str.contains(title.lower(),na=False)])
        summary=mainrow['Summary'].values
        for i in summary:
            summary=i

```

```

#This converts the summary of the book into a vector
v1=text_to_vector(summary)
cosinelist=[]
#This loop computes the cosine similarity of the summaries and the summary of
the main book
for i in summaryset:
    v2= text_to_vector(i)
    cosinelist.append(get_cosine(v1,v2))
maintitles=[]
#This creates a sorted list of titles using the cosine similarities(minimum t
o maximum similarity)
maintitles=[x for y,x in sorted(zip(cosinelist,titleset))]
#Since the most similar summary would be itself, the last(maximum) similar ti
tle is popped from the list and the list is returned.
maintitles.pop(-1)
return maintitles
#This function finds the books with the same author and returns the list of title
s, sorted from least similar book to most similar book.
#It requires the name of the author and the title of the book
def findBooksAuthor(author,title):
    #This creates a pandas series of all rows with matching authors to the main b
ook
    rows=data1[data1['Author']==str(author)]
    #This creates a list of titles from the pandas series
    titleset = list(rows['title'].values)
    #This removes the title if the title matches the title of the main book
    for i in range(0,len(titleset)):
        if(str(titleset[i])==title.lower()):
            titleset.pop(i)
            break

    summaryset=[]
    #This loop creates a list of summaries of all books from the titleset
    for index,i in rows.iterrows():
        summaryset.append(i['Summary'])
    #This is the row of the main book
    mainrow=(data1[data1['title'].str.contains(title.lower(),na=False)])
    summary=mainrow['Summary'].values
    for i in summary:
        summary=i
    #This converts the summary of the main book to a vector
    v1=text_to_vector(summary)
    cosinelist=[]
    #This computes the cosine similarity of each book with the main book summary
    for i in summaryset:

```

```

        v2= text_to_vector(i)
        cosinelist.append(get_cosine(v1,v2))
maintitles=[]
#This creates a list of titles sorted using the list of cosine similarities
maintitles=[x for y,x in sorted(zip(cosinelist,titleset))]

return maintitles

#This is the main function of this file, it calls all the required functions and
returns the final lists
#It requires the title of the main book
def findbook(title):
    #This finds the author of the book using the findAuthor() function
    author = findAuthor(title)
    #If the author is not found, it is because the book doesn't exist in the data
    set. Thus the function returns None,None, so that the error can be handled
    if(not author):
        return None,None
    #This creates a list of genres for the book
    genrelist = findGenre(title)
    #Some books are missing genres in the dataset. Thus, the function returns Non
    e,None to handle the error
    if not genrelist:
        return None,None
    #Since Novel is a very common 'genre', I deleted Novel from the genrelist to
    find more efficient results
    for i in range(0,len(genrelist)):
        if('Novel' in genrelist[i]):
            genrelist.pop(i)
            break
    #This rechecks if the genrelist doesn't have elements. If so, it returns None
    ,None to handle the error
    if not genrelist:
        return None,None
    #Finally, the sorted list of books with same genre is created
    titlesFromGenre = findBooksGenre(genrelist,title)
    if(len(author)>1):
        author=author[0]
    else:
        for i in author:
            author=i
    #This finds the books with the same author
    titlesFromAuthor = findBooksAuthor(author,title)
    finallistGenre=[]
    finallistAuthor=[]

```

```

maxn=5
maxn2=5
#Since I don't want to display more than 5 books, this limits the number in the list to 5 or total (whichever is less)
if len(titlesFromGenre)<5:
    maxn=len(titlesFromGenre)
if len(titlesFromAuthor)<5:
    maxn2=len(titlesFromAuthor)
for i in range(0,maxn):
    finallistGenre.append(titlesFromGenre[i-1])
for i in range(0,maxn2):
    finallistAuthor.append(titlesFromAuthor[i-1])
return finallistAuthor,finallistGenre

```

#This function uses the Goodreads API to return a link to the poster of the book
 #It requires the author and the title

```

def findPoster(title,author):
    #First, the title is modified to work with the API
    title_edited=title.replace(' ','+')
    author_edited=author.replace(' ','')
    #Using the requests module, the goodreads API returns an XML file with all the data required
    r=requests.get(f"https://www.goodreads.com/search/index.xml?key=ToxHhDKy3FxFWYdcRmhUg&q={title_edited}")
    #This converts the xml to a readable dict which can be parsed to find the required data
    books = xmltodict.parse(r.content)
    #This loop finds the required book from the API dict and then returns the posterlink
    for i in books['GoodreadsResponse']['search']['results']['work']:
        x= i['best_book']['author']['name'].lower()
        x=x.replace(' ','')
        if(x==author_edited.lower()):
            return i['best_book']['image_url']

```

'''

=====

ACADEMIC INTEGRITY STATEMENT

I have not used source code obtained from any other unauthorized source, either modified or unmodified. Neither have I provided access to my code to another. The project I am submitting is my own original work.

=====

'''

movies.py

```
'''
=====
ENGR 133 Program Description
    This is the file that finds the movie titles similar to the title entered based on the option picked.
    It then computes the cosine similarity and sorts the list of titles according to the similarity.

Assignment Information
    Assignment:      Final Project
    Author:          Ainesh Sootha, asootha@purdue.edu
    Team ID:         001-14

Contributor:

    My contributor(s) helped me:
    [ ] understand the assignment expectations without
        telling me how they will approach it.
    [ ] understand different ways to think about a solution
        without helping me plan my solution.
    [ ] think through the meaning of a specific error or
        bug present in my code without looking at my code.
    Note that if you helped somebody else with their code, you
    have to list that person as a contributor here as well.
=====
'''

import pandas as pd
import numpy as np
from operator import add
import re, math
from collections import Counter
import requests
from PIL import Image
import io
import urllib.request
import functools

#This reads the hdf5 files (datasets) as pandas dataframe. I am using hdf5 instead of csv because it is faster to access
data1 = pd.read_hdf("datasetmovies.h5", usecols=['original_title', 'genres', 'overview', 'belongs_to_collection'])
datacast=pd.read_hdf("datasetcast.h5", usecols=['cast'])
```

```

#This basically breaks lines into words (excluding symbols, numerals etc)
WORD = re.compile(r'\w+')

#This function computes the cosine similarity for 2 vectors
#It requires 2 arguments (2 vectors computed using text_to_vector())
def get_cosine(vec1, vec2):
    #This finds similarity for each word and creates a list
    intersection = set(vec1.keys()) & set(vec2.keys())
    #The rest of this function basically computes the "dot product" of the 2 vec
    tors and divides it with the magnitude of the list
    numerator = sum([vec1[x] * vec2[x] for x in intersection])
    sum1 = sum([vec1[x]**2 for x in vec1.keys()])
    sum2 = sum([vec2[x]**2 for x in vec2.keys()])
    denominator = math.sqrt(sum1) * math.sqrt(sum2)
    if not denominator:
        return 0.0
    else:
        return float(numerator) / denominator

#This function converts the string(text) to vectors (Dictionary of all words with
    their respective count)
def text_to_vector(text):
    words = WORD.findall(text)
    return Counter(words)

#This fuction finds the list of genres of the movie from the dataset
#It requires the title of the movie as a parameter
def findGenre(title):
    #This creates a pandas series with all the rows with the same title as the pa
    rameter
    rows = data1[data1['original_title'].str.lower() == title.lower()]
    #This then matches the author column of the title and stores the genres as a
    list into genreset
    genreset = rows['genres'].values
    #The list is modified as a string to create a list of genres
    genreset = str(genreset)
    genreset = genreset.strip('[')
    genreset = genreset.strip(']')
    genreset = genreset.strip('"')

    genreset = genreset.strip('[')
    genreset = genreset.strip(']')
    genreset = genreset.strip('{')
    genreset = genreset.strip('}')
    genreset = genreset.split('}, {')

```

```

for i in range(0,len(genreset)):
    if genreset[i]=='':
        genreset.pop(i)

return genreset

```

```

#This function finds the list of collections for the movie from the dataset
#It requires the title of the movie as a parameter
def findCollection(title):
    #This creates a pandas series with all the rows with the same title as the parameter
    rows = data1[data1['original_title'].str.lower() == title.lower()]
    #This then matches the author column of the title and stores the author as a list into author
    collectionSet = list(rows['belongs_to_collection'].values)
    #Clean the list of any 'nan' values
    cleanedList = [x for x in collectionSet if str(x) != 'nan']
    #If the list is empty, return None as error
    if not cleanedList:
        return None
    #Modify list to make it easier to use
    for i in collectionSet:
        collectionSet=i
    collectionSet=str(collectionSet)
    collectionSet=collectionSet.strip('{')
    collectionSet=collectionSet.strip('}')
    collectionSet=collectionSet.split(', ')
    for i in range(0,len(collectionSet)):
        if("name" in collectionSet[i]):
            mainCollection=collectionSet[i]
    return mainCollection

```

```

#This function finds the cast of the movie from the 2 datasets
#It requires the title of the movie
def findCast(title):
    #Since the datasets have matching indices (First movie in main dataset -
    > First movie in Cast dataset)
    #Get the value of the index of the title from the main pandas dataframe
    index=data1.loc[data1['original_title'].str.lower()==title.lower()].index[0]
    #Locate the same index in the cast dataframe
    rows=datacast.iloc[index,0]
    #Modify the string to remove unneeded characters
    rows=rows.strip('[')

```



```

rows=rows.strip(' ')
rows=rows.split(', ')
l=[]
#Create and return list of cast
for i in rows:
    if("name" in i):
        l.append(i)
return l

#This function finds the list of genres of the movie from the dataset
#It requires the list of genres and the title of the movie as parameters
def findMoviesGenre(genrelist,title):
    #Find the rows containing any genre from the list
    for i in genrelist:
        rows=(data1[data1['genres'].str.contains(i,na=False)])
    l=[]
    #Iterate through the pandas series and create a list of 0s
    #Then 1 is added for every matching genre
    for index,i in rows.iterrows():
        l.append(0)
        for j in genrelist:
            if(j in i['genres']):
                l[-1]=l[-1]+1
    titleset=[]
    summaryset=[]
    j=-1
    #Save only the titles with max number of genres matching the main movie (or c
lose to the max)
    #Create a list of titles and summaries to these books
    for index,i in rows.iterrows():
        j=j+1
        if(l[j]>0 and (l[j]==max(l) or l[j]==max(l)-1)):
            titleset.append(i['original_title'])
            summaryset.append(i['overview'])
    #This creates a pandas row for the main movie with all the details from the d
ataframe
    mainrow=(data1[data1['original_title'].str.lower()==title.lower()])
    summary=mainrow['overview']

    for i in summary:
        summary=i
    #This converts the summary to 'vectors'
    v1=text_to_vector(summary)
    cosinelist=[]

```

```

#This computes the cosine similarity of the summaries to the main summary
for i in summaryset:
    v2= text_to_vector(str(i))
    cosinelist.append(get_cosine(v1,v2))
maintitles=[]
finaltitles=[]
#This creates a SORTED list of the titles based on the cosine similarities
maintitles=[x for y,x in sorted(zip(cosinelist,titleset))]
#Since the most similar sumamry would be the main movie itself, the last(max
similarity) movie title is deleted from the list
maintitles.pop(-1)
#The last 10 titles are returned as a list (10 Most similar)
for i in range(len(maintitles)-10,len(maintitles)):
    finaltitles.append(maintitles[i])
return finaltitles

#This finds the movies with same cast members as the movie entered
#It requires the castlist and the title of the main movie as parameters
def findMoviesCast(castlist,title):
    indexlist=[]
    for i in range(0,5):
        #Find all rows with the same cast members (top 5) and save the indices of
the rows as a list
        rows = datacast.index[datacast['cast'].str.contains(castlist[i])].tolist(
)
        #Append the list to indexlist
        indexlist.append(rows)
    #Since the 'rows' created would be multi dimensional, reduce() reduces the li
st to a single dimension
    indexlist=functools.reduce(add ,indexlist)

    #Remove any repeated elements by converting the list to a set and the back to
a list
    indexlist=list(set(indexlist))
    titleset=[]
    l=[]
    summaryset=[]
    #Locate all the rows with indices from the indexlist
    row=data1.iloc[indexlist]
    #Create a list of titles and summaries from the rows
    titleset=list(row['original_title'])
    summaryset=list(row['overview'])
    #Get the data for the main title
    mainrow=(data1[data1['original_title'].str.lower()==title.lower()])
    #Get the summary of the main title

```

```

summary=mainrow['overview']
for i in summary:
    summary=i
#Convert the summary to a 'vector'
v1=text_to_vector(summary)
cosinelist=[]
#Compute the cosine similarirty for each summary to the main summary
for i in summaryset:
    v2= text_to_vector(str(i))
    cosinelist.append(get_cosine(v1,v2))
maintitles=[]
#This creates a SORTED list of titles using the cosine similarities (min to m
ax similarity)
maintitles=[x for y,x in sorted(zip(cosinelist,titleset))]
#Since the most similar summary is of itself, the last (most similar summary)
title is deleted
maintitles.pop(-1)
#6 titles are returned (6 most similar summaries)
return maintitles[-6:]

#This function usins the OMDb API and returns the link to the poster of a movie
#It requires the title of the movie
def findPoster(title):
    #This uses the requests module to get the JSON data from the OMDb API
    detailslist = requests.get(url=f'http://www.omdbapi.com/?apikey=477f0f4c&t={t
itle}')
    data=detailsList.json()
    #The JSON data is converted to a dict and the the poster link is returned
    posterlink=data['Poster']
    return posterlink

#This function finds movies from the same collection as the main movie
#It requires the collection and the title of the main movie as parameters
def findMoviesCollection(collectionlist,title):
    #All rows that contain movies from the same collection are stored as a pandas
series
    rows=(data1[data1['belongs_to_collection'].str.contains(collectionlist,na=Fa
lse)])
    l=[]
    #Iterate through the pandas series and create a list of 0s
    #Then 1 is added for every matching collection
    for index,i in rows.iterrows():
        l.append(0)
        for j in collectionlist:
            if(j in i['belongs_to_collection']):

```

```

        l[-1]=l[-1]+1
titleset=[]
summaryset=[]
j=-1
#The rest of the function is similar to other functions
for index,i in rows.iterrows():
    j=j+1
    if(l[j]>0 and (l[j]==max(l) or l[j]==max(l)-1)):
        titleset.append(i['original_title'])
        summaryset.append(i['overview'])
mainrow=(data1[data1['original_title'].str.lower()==title.lower()])
summary=mainrow['overview']

```

```

for i in summary:
    summary=i
v1=text_to_vector(summary)
cosinelist=[]
for i in summaryset:
    v2= text_to_vector(str(i))
    cosinelist.append(get_cosine(v1,v2))
maintitles=[]
maintitles=[x for y,x in sorted(zip(cosinelist,titleset))]
maintitles.pop(-1)
return maintitles

```

#This is the main function of the file. It calls the other functions and returns the required lists

#It requires the title of the movie as a parameter

```

def findMovies(title):
    #Store genres in list
    genrelist=findGenre(title)
    #If genrelist is empty, return None,None,None (Error)
    if not genrelist:
        return None,None,None
    #Find castlis and store as list
    castlist=findCast(title)
    #Store collection of the movie(If exists (else empty))
    collectionlist=findCollection(title)
    #Get a sorted list of titles with same genres and similar summaries
    moviesByGenre=findMoviesGenre(genrelist,title)
    #Get a sorted list of titles with same actor(s) and similar summaries
    moviesByCast=findMoviesCast(castlist,title)
    #If collectionlist is empty, store none to return
    #Else, find movies from same collection
    if not collectionlist:

```

```

        moviesByCollection=None
    else:
        moviesByCollection=findMoviesCollection(collectionlist,title)
    return moviesByGenre,moviesByCast,moviesByCollection

'''
=====
ACADEMIC INTEGRITY STATEMENT
    I have not used source code obtained from any other unauthorized
    source, either modified or unmodified. Neither have I provided
    access to my code to another. The project I am submitting
    is my own original work.
=====
'''

```

csvTohdf.py

```

import pandas as pd
df = pd.read_csv("datasetbooks.csv")
df.to_hdf('./datasetbooks.h5', 'test')

```

Project.py

```
'''
=====
ENGR 133 Program Description
    This is the main project file, ie-
the flask app. It renders the HTML templates
    and displays the GUI that is seen by the user.

Assignment Information
    Assignment:      Final Project
    Author:          Ainesh Sootha, asootha@purdue.edu
    Team ID:         001-14

Contributor:

    My contributor(s) helped me:
    [ ] understand the assignment expectations without
        telling me how they will approach it.
    [ ] understand different ways to think about a solution
        without helping me plan my solution.
    [ ] think through the meaning of a specific error or
        bug present in my code without looking at my code.
    Note that if you helped somebody else with their code, you
    have to list that person as a contributor here as well.
=====
'''

from flask import Flask, render_template,request
import books
import movies
import missing
app = Flask(__name__)

@app.route("/")
def home():
    return render_template('index.html')

#This allows flask to receive information from the form and display appropriate i
nfo on the results page
@app.route('/result',methods = ['POST', 'GET'])
#This is the main function to call the books module or the movies module
def result():
    titleMain=request.form['title']
    valueOption=request.form['options']
    selectOption=request.form['bmselect']
```

```

#If the user selects books. get the lists using the title
if(valueOption=="book"):
    #If user selects genre, make listgenre the mainlist
    if selectOption=="genre":
        listsecondary,listmain=books.findbook(titleMain)
    #Else make the other list the main list
    else:
        listmain,listsecondary=books.findbook(titleMain)
    author=books.findAuthor(titleMain)
    for i in author:
        author=i
    #If list does not exist, display error screen
    if not listmain:
        missing.writemissing(titleMain,'book')
        return render_template("error.html",error="list",option=valueOption)
    #Otherwise, display the result screen and find the poster of the book
    else:
        posterlink=books.findPoster(titleMain,author)
        return render_template("result.html", titleMain=titleMain, listmain=listmain,listsecondary=listsecondary,valueOption=valueOption,posterlink=posterlink
)

#The same as above for movies
else:
    if selectOption=="genre":
        listmain,listsecondary,listtertiary=movies.findMovies(titleMain)
    elif selectOption=="cast":
        listsecondary,listmain,listtertiary=movies.findMovies(titleMain)
    elif selectOption=="collection":
        listsecondary,listtertiary,listmain=movies.findMovies(titleMain)
        if listmain==None:
            return render_template("error.html",error="collection",option=valueOption, selectOption=selectOption)

    if not listmain:
        missing.writemissing(titleMain,'movie')
        return render_template("error.html",error="list",option=valueOption,
selectOption=selectOption)
    else:
        posterlink=movies.findPoster(titleMain)
        return render_template("result.html", titleMain=titleMain, listmain=listmain,listsecondary=listsecondary,valueOption=valueOption,posterlink=posterlink
)

```

```
if __name__ == "__main__":  
    app.run(debug=True)
```

```
'''
```

```
=====
```

ACADEMIC INTEGRITY STATEMENT

```
I have not used source code obtained from any other unauthorized  
source, either modified or unmodified. Neither have I provided  
access to my code to another. The project I am submitting  
is my own original work.
```

```
=====
```

```
'''
```