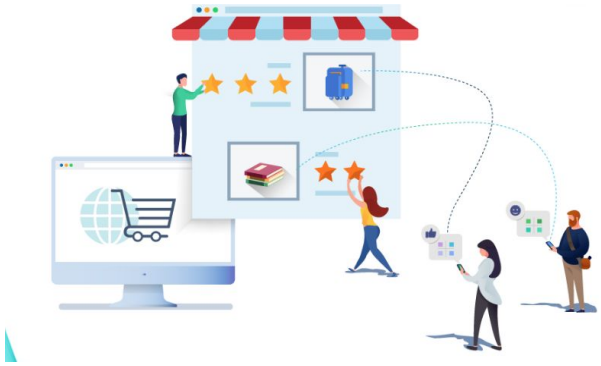
1.Executive summary

Goodreads is a social cataloging website where the users can search its database to find book reviews, give ratings and suggestions along with creating their own reading list. A recommender system is an algorithm which uses gradient descent to calculate the ratings which users would have highly rated. My goal for this capstone is to use the ratings and book descriptions for books using unsupervised learning and Natural language processing techniques in order to build a recommender system.

Pairwise distance and cosine similarities were used to measure the distance between each book against the remaining. The purpose of this is to find a book which has the least distance i.e closer to 0.

Data collection tools used were goodreads API, Selenium and Beautiful Soup, which scraped data from the web page. User based and content based recommender system was able to predict similar books based on user ratings and book description. The limitations for this project was data collection. It is a very slow process and uses a lot of RAM. Thus for the purpose of this capstone I considered a part of my dataset to run the recommender system.



2.Building a scraper for data collection

2.1.Data Collection for ratings

Goodreads API key was not very useful for data collection. Web scraping tools like Selenium and Beautiful Soup had to be used for data collection. The libraries used for scraping were:

```
import requests, json, time
from bs4 import BeautifulSoup
import pandas as pd
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
import os, sys
```

Selenium is a web browser automation tool which helps click, fill out information and so on. Sometimes websites ban web scrapers if the requests are very frequent. Thus time was added at regular intervals in the function. ChromeDriver is a separate executable that Selenium WebDriver uses to control Chrome. Once the ChromeDriver is setup we add our goodreads url to the driver in order to get the webpage. Beautiful Soup is a python package for parsing HTML and XMLthat can be used to extract data. The

function was created based on this basic concept of selenium in order to get ratings from all the pages. The maximum pages are 10. The scraper breaks when there is no web element and moves on to the next book. This continues until it reached the last book mentioned in the range. Once the data was collected it was saved as a csv file as a dataframe.

```
driver = webdriver.Chrome(executable_path="./chromedriver/macos/chromedriver")
driver.get('https://www.goodreads.com/book/show/1')
```

```
soup = BeautifulSoup(driver.page_source, 'lxml')
```

```
first_url = 8000        # Book number - start point
last_url = 10000       # Book number - end point (Last book is 8,630,000)
ratings = []          # Empty container to store new columns
for book_reference_number in range(first_url, last_url):
    print(book_reference_number)  # This prints which book the scrapper is currently scrapping
    driver.get("https://www.goodreads.com/book/show/"+str(book_reference_number)) # This gets the url for the book
    time.sleep(2)
    soup = BeautifulSoup(driver.page_source, 'lxml') #create a soup object

    # some pages do not have title, adding a logical condition to pass those pages so that the code does not break
    no_title = soup.find('title').text
    if no_title == 'Page not found':
        continue

    # This portion of the code finds the weblink to click the next page, it stops as soon as there is no next page
    # referred this portion from stack overflow
    last_page_source = ''
```

```
    last_page_source = ''

    while True:
        page_changed = False # It's useful to declare whether the page has changed or not
        attempts = 0
        while(not page_changed):
            if last_page_source != driver.page_source:
                page_changed = True
            else:
                if attempts > 5: # Decide on some point when you want to give up.
                    break;
                else:
                    time.sleep(3) # Give time to load new page. Interval could be shorter.
                    attempts += 1
        if page_changed:
            soup_1 = BeautifulSoup(driver.page_source, 'lxml')
            user = soup_1.find('div', {'id': 'bookReviews'})
            user = user.find_all('div',{'class':'friendReviews elementListBrown'})
            review_list = soup_1.find_all('div', class_ = 'reviewHeader uitext stacked')
            for row in review_list: # finding for each separate rating
                # create an empty dict to add columns
                rating = {}

                try:
                    book_title = soup.select('.gr-h1.gr-h1--serif')[0].text.strip()
                except:
                    book_title = ''
```

```
            try:
                # try and except is needed because not all the users have a rating
                rating['user_id'] = row.find('a', class_ = 'user', href = True)['href'].split('/')[3].split('-')[0] #
                rating['rating'] = row.find('span',{'class':'staticStars'})['title'] # grabbing user rating out of 5
                rating['book_name']= book_title
                rating['book_id'] = book_reference_number
                ratings.append(rating)
            except:
                pass
        # if it reaches last page the code breaks and moves on to the next book
        last_page_source = driver.page_source
        try:
            next_page_element = driver.find_element_by_class_name('next_page')
            driver.execute_script("arguments[0].click();", next_page_element) # clicking on the next button to scrape
            time.sleep(2)
        except:
            break
    else:
        df_rev = pd.DataFrame(ratings) # merging all the results to build a data frame
        #print(df_rev.drop_duplicates())
        break;
```

2.2.Data Collection for description and book names

Selenium was used to collect book description and goodreads API key was used to get the name of the books in order to increase the efficiency. Once the two data sets were collected they were merged on the book_id into one dataset. The scrapper used to scrape the book description is similar to the one above.

```
first_url = 1000
last_url = 3000
container = []
for book_reference_number in range(first_url, last_url):
    print(book_reference_number)
    time.sleep(5)
    driver.get("https://www.goodreads.com/book/show/"+str(book_reference_number))
    time.sleep(2)
    soup = BeautifulSoup(driver.page_source, 'lxml')
    desc_dict = {}

    try:
        book_title = soup.select('.gr-h1.gr-h1--serif')[0].text.strip()
    except:
        book_title = ''
        time.sleep(1)
    try:
        description_list = soup.find( 'div', class_= 'readable stacked').text
        time.sleep(3)
    except:
        continue
    desc_dict['description'] = description_list
    desc_dict['book_id'] = book_reference_number
    container.append(desc_dict)

    df_desc = pd.DataFrame(container)
```

```
# df = pd.concat(map(pd.read_csv, glob.glob('book_description_dataset/*.csv')))
df_description = pd.read_csv('./book_description_dataset/combined_desc.csv')
df_title = pd.read_csv('./book_description_dataset/title.csv')
```

```
df = pd.merge(df_title, df_description, on = 'book_id')
df.head()
```

2.3.Data collection for hybrid recommender

For the hybrid recommender dataset, the ratings and description datasets were already collected. The next step was to match the shape of the datasets since one dataset had more books than the other. Once the shape matched, merged the two dataframes on book_id.

Reading the ratings dataset and only considering 2999 books

```
df_ratings = pd.read_csv('./book_rating_dataset/ratings.csv')
df_ratings.head()
```

```
# only take book ids upto 3000 to match the description dataframe
df_ratings = df_ratings[df_ratings['book_id'] <3000]
df_ratings.drop_duplicates(keep='first', inplace = True)
```

Reading the book description dataset

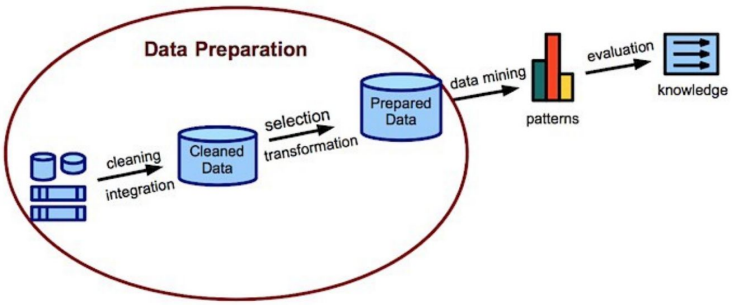
```
# take the original datasets for ratings with 3000 books
df_description = pd.read_csv('./book_description_dataset/description.csv')
df_description.head()
```

Merged the two data frames on book_id

```
hybrid_df = pd.merge(df_description,df_ratings, on=['book_id'], how='left')
```

```
hybrid_df.to_csv('./book_hybrid_dataset/hybrid.csv')
```

3.Exploratory Data Analysis



Data pre-processing... | Source: img_credit

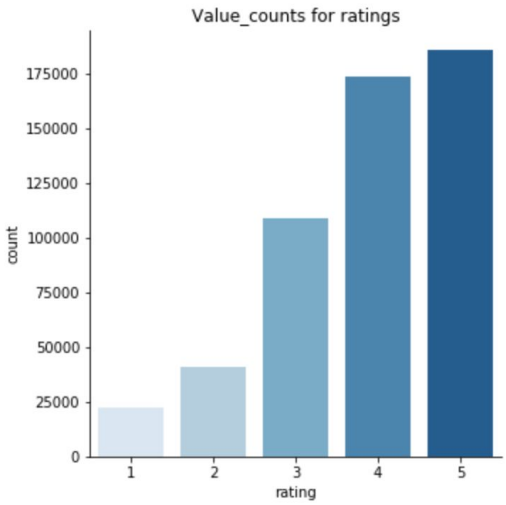
3.1.EDA for book ratings

Since Selenium was used to scrape data, only required data was collected, thus there was very little data cleaning required. There was a chance of duplicate entries and unnamed columns since multiple csv files were collected. Duplicates and unnamed columns were dropped and clean dataset was created. The main feature of this recommender were ratings. Goodreads rates 1 - 5 stars as follows :

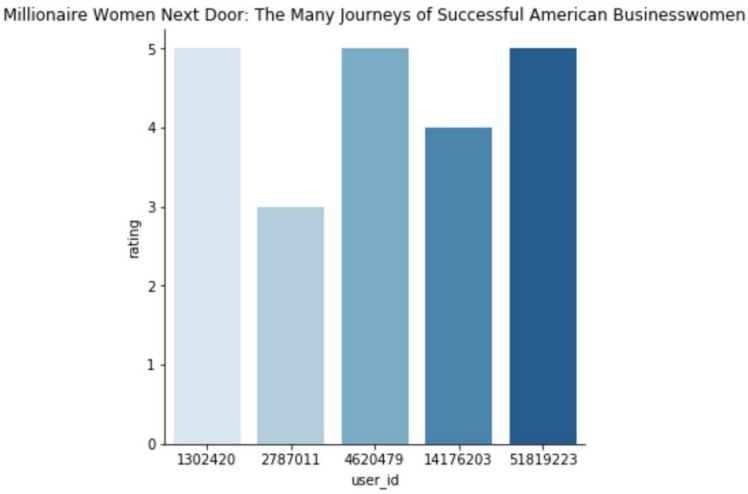
```
it was amazing     185740
really liked it    173860
liked it           108993
it was ok           40834
did not like it     22602
Name: rating, dtype: int64
```

Ratings were than mapped in order to make it numeric and easy to interpret and calculate. Once the data frame is mapped the graph below shows us the value counts

```
#mapped rating from 1-5
df['rating']=df['rating'].map({'it was amazing': 5,
                               'really liked it':4,
                               'liked it':3,
                               'it was ok':2,
                               'did not like it': 1})
```



First five user ratings for book id = 1000. As we see from the visual user_id first, third and fourth have rated similarly for this book. These three users may have similar interests. This is just an idea how similarities will be calculated across all the books and users.



3.2.EDA for book description

Natural Language Processing techniques were used to clean data for the description column. In order to remove special characters, punctuations and HTML formats from the description column, used Regular Expression library. Unnamed columns were dropped along with setting the index to book_name in order to run smoothly through building a recommender system.

```
# removed speacial characters
# removed punctutions
# all lowercase
df = df.replace('\n',' ', regex=True)
df["description"] = df['description'].str.replace('[^\w\s]',' ')
df['description'] = df['description'].str.lower()
```

book_name	description
Harry Potter and the Half-Blood Prince (Harry Potter, #6)	when harry potter and the half blood prince o...
Harry Potter and the Order of the Phoenix (Harry Potter, #5)	there is a door at the end of a silent corrid...
Harry Potter and the Sorcerer's Stone (Harry Potter, #1)	harry potter s life is miserable his parents...
Harry Potter and the Chamber of Secrets (Harry Potter, #2)	the dursleys were so mean and hideous that su...
Harry Potter and the Prisoner of Azkaban (Harry Potter, #3)	harry potter s third year at hogwarts is full...
Harry Potter and the Goblet of Fire (Harry Potter, #4)	harry potter is midway through his training a...
The Harry Potter Collection (Harry Potter, #1-6)	six years of magic adventure and mystery ma...
Harry Potter Boxed Set, Books 1-5 (Harry Potter, #1-5)	box set containing harry potter and the sorce...
Unauthorized Harry Potter Book Seven News: "Half-Blood Prince" Analysis and Speculation	through the magic of print on demand technolo...
Harry Potter Collection (Harry Potter, #1-6)	six years of magic adventure and mystery ma...
The Hitchhiker's Guide to the Galaxy (Hitchhiker's Guide to the Galaxy, #1)	seconds before the earth is demolished to mak...
The Ultimate Hitchhiker's Guide: Five Complete Novels and One Story (Hitchhiker's Guide to the Galaxy, #1-5)	at last in paperback in one complete volume ...

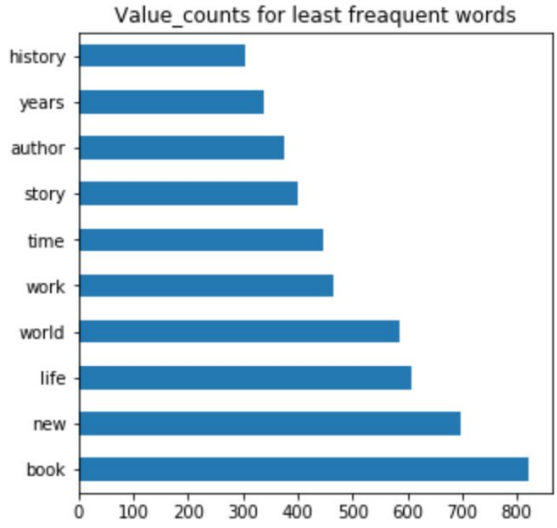
Once the description column was cleaned, it was Count Vectorized with hyperparameters.

```
cvec = CountVectorizer(stop_words = ENGLISH_STOP_WORDS, max_df = 0.95, min_df = 3, binary = True)
matrix = cvec.fit_transform(df['description'])
df_matrix = pd.DataFrame(matrix.todense(), columns = cvec.get_feature_names())
```

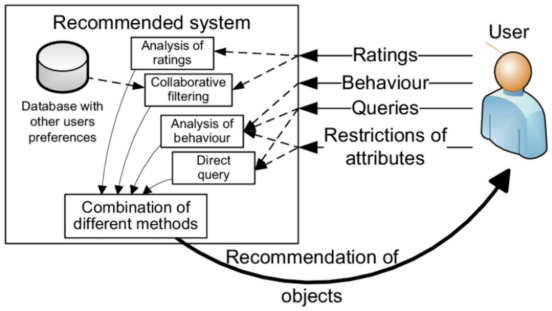
We observe that there are a lot of numbers and foreign language as column names, which looks unnecessary. We use regular expressions to get rid of them. We want to make our matrix as clean as possible.

```
#removed numbers and foreign language
english_columns = []
non_english_columns = []
for word in list(df_matrix.columns):
    if re.sub("[^a-zA-Z]", " ", word) == word:
        english_columns.append(word)
    else:
        non_english_columns.append(word)
```

Value counts for the least frequent words, since they are important in determining the book it came from

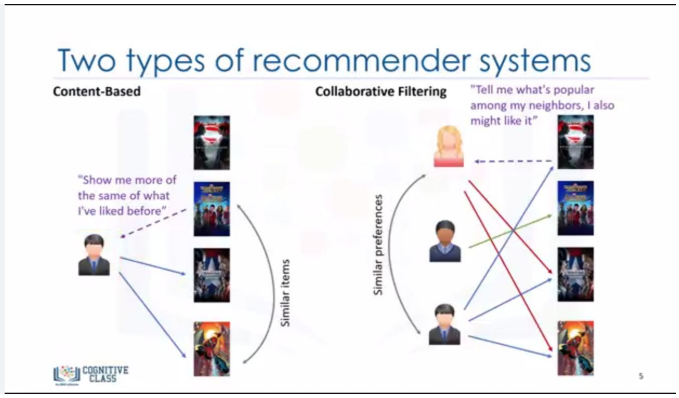


4. Recommender Systems



4.1.Item-based collaborative recommender system

Collaborative filtering methods to collect and analyze information on user behavior i.e. ratings and then predicts what the users will like based on their similarity to other users. For an item based collaborative recommender we take the weighted sum of ratings of other books.



source:https://medium.com/towards-artificial-intelligence/building-a-recommender-system-with-pandas-1ca0bb03fdce

Create a pivot table

- Pivot table will convert an array into matrix form
- We need to specify our index, columns and values for the matrix
- Index is the book's name and the values for the index is the ratings. Columns are the users
- Checking the shape of the matrix

```
pivot = df.pivot_table(index = 'book_name', columns = 'user_id', values = 'rating')
pivot.head()
```

user_id	1	3	5	6	8	9	14	18	21	26	...	105738922	105740054	105757466	10575
book_name															
"A" Is for Zebra	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	
"Fabulae Novae" Di Fedro	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	
"The Earth is Flat" and Other Great Mistakes	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	
"The Pleasure Dome": Amerikansk Experimentfilm 1939 1979: [Moderna Museet, Stockholm], 16	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	

Create sparse matrix

- Created a sparse matrix since the size of the pivot dataset is too large
- Sparse matrix performs gradient descent under the hood in order to decrease the size of the matrix
- The shape is not affected, only the size of the matrix is reduced
- Sparse matrix shows the index and the column name which has value(rating), since everything else is 0 except one rating per user
- It also shows the rating value (code shown below in the print statement)

```
print(pivot_sparse[:5, :])
```

```
(0, 5943)     2.0
(0, 6644)     4.0
(0, 11990)    3.0
(0, 14450)    3.0
(0, 22327)    2.0
(0, 22404)    4.0
(0, 23381)    5.0
(0, 23716)    4.0
(0, 25418)    3.0
```

```
# we want a metrix with a smaller file size for easier computation
pivot_sparse = sparse.csr_matrix(pivot.fillna(0))
```

Calculate cosine similarities

- Pairwise distances is an sklearn function which will calculate the row wise difference between books
- It will return a square matrix with the comparisons

```
# cos sim: -1 and +1 where +1 is good
# pairwise: 0 and 1 where 0 is good
recommender = pairwise_distances(pivot_sparse, metric='cosine')
```

Create distances dataframe

- We create a pandas dataframe from the array above since it is easier to visualize
- The values are the distances calculated between books

```
# previous pivot table has the name of the indices, we can put them togeather and create a pandas dataframe
recommender_df = pd.DataFrame(recommender, index = pivot.index, columns = pivot.index)
```

Evaluate recommender performance

- Create a filter to find the name of the books
- Copy paste the name of the book to the recommender df
- Sort values from highest to lowest
- Started from 1 since 0 is the book against itself

```
1 book_name = "Love the Life You Live: 3 Secrets to Feeling Good--Deep Down in Your Soul"
2 recommender_df[book_name].sort_values()[1:20]
```

```
book_name
Finding Contentment: When Momentary Happiness Just Isn't Enough                                  0.953688
How to Know If Someone is Worth Pursuing in Two Dates or Less                                    0.963274
The Feeling Good Handbook                                                                        0.977409
Power of an Hour: Business and Life Mastery in One Hour a Week                                   0.979855
Betty Crocker's Diabetes Cookbook: Everyday Meals Easy as 1-2-3                                  0.984637
Cinco Lenguajes del Amor Para Solteros, Los: Five Love Languages for Singles                     0.985847
Why Zebras Don't Get Ulcers                                                                      0.986295
The Dream Giver                                                                                  0.986705
The Broker                                                                                       0.987316
The Long Tail: Why the Future of Business Is Selling Less of More                                0.988020
The Long Tail:  Why The Future Is Selling Less Of More                                           0.988089
The Omnivore's Dilemma: A Natural History of Four Meals                                          0.989169
CSS Cookbook                                                                                     0.990002
The Game: Penetrating the Secret Society of Pickup Artists                                       0.990693
Hidden Persuaders                                                                                0.991917
The Hidden Persuaders                                                                            0.991917
God Said It, Don't Sweat It: Sound Encouragement to Keep the Little Things from Overwhelming You 0.992826
Democracy in America                                                                             0.993466
The Power Broker: Robert Moses and the Fall of New York                                          0.995597
Name: Love the Life You Live: 3 Secrets to Feeling Good--Deep Down in Your Soul, dtype: float64
```

4.2.Content based recommender system

Content-based recommender system uses characteristics such as the description of the books. We will figure out what kind of books a user likes based on his history. The system groups similar products based on their features. Content-based systems rely on machine learning techniques to calculate the probabilities of user liking. Machine learning techniques include neural networks, decision trees classifiers etc.

Same process as the item based recommender system with the exception of column names and the index values.

```
recommender_df = pd.DataFrame(recommender, index = df.index, columns = df.index)
recommender_df
```

book_name	Harry Potter and the Half-Blood Prince (Harry Potter, #6)	Harry Potter and the Order of the Phoenix (Harry Potter, #5)	Harry Potter and the Sorcerer's Stone (Harry Potter, #1)	Harry Potter and the Chamber of Secrets (Harry Potter, #2)	Harry Potter and the Prisoner of Azkaban (Harry Potter, #3)	Harry Potter and the Goblet of Fire (Harry Potter, #4)	The Harry Potter Collection (Harry Potter, #1-6)	Harry Potter Boxed Set, Books 1-5 (Harry Potter, #1-5)	Unauthorized Harry Potter Book Seven News: "Half-Blood Prince" Analysis and Speculation	Co
book_name										
Harry Potter and the Half-Blood Prince (Harry Potter, #6)	0.000000	0.818868	0.833819	0.830337	0.841765	0.850104	0.863164	0.910874	0.889621	0

```
recommender_df[["Harry Potter and the Half-Blood Prince (Harry Potter, #6)"]]
```

book_name	Harry Potter and the Half-Blood Prince (Harry Potter, #6)
book_name	
Harry Potter and the Half-Blood Prince (Harry Potter, #6)	0.000000
Harry Potter and the Order of the Phoenix (Harry Potter, #5)	0.818868
Harry Potter and the Sorcerer's Stone (Harry Potter, #1)	0.833819
Harry Potter and the Chamber of Secrets (Harry Potter, #2)	0.830337
Harry Potter and the Prisoner of Azkaban (Harry Potter, #3)	0.841765
...	...
Louisa May Alcott: Her Life, Letters and Journals	0.964606
The Journals of Louisa May Alcott	0.953870
Louisa May Alcott and "Little Women": Biography, Critique, Publications, Poems, Songs, and Contemporary Relevance	0.967156
The Secret Garden	0.929338
My Secret Garden: Women's Sexual Fantasies	0.984421

2345 rows × 1 columns

4.3.Hybrid recommender system

This approach takes calculates content based recommender and item based recommender separately at first, then collaborates the outputs to produce a system that considers both the systems. The steps to build this is by following the previous to recommenders first then:

- Defining a function which would merge the two recommender systems together
- Concat the output of both the lists
- Takes an average of the books that exist in both the recommender systems
- Displays the top five books

```python
def average_similar_books(book_name):
    list_1 = list(recommender_df[book_name].sort_values()[1:6].index)
    list_2 = list(recommender_df_2[book_name].sort_values()[1:6].index)

    intersection = list(set(list_1).intersection(set(list_2)))
    intersection_recommender_1 = recommender_df.loc[list_1,:][book_name][intersection]
    intersection_recommender_2 = recommender_df_2.loc[list_2,:][book_name][intersection]

    average_scores = (intersection_recommender_1 + intersection_recommender_2) /2
    first_recommender_scores = recommender_df.loc[list_1,:][book_name].drop(labels = intersection)
    second_recommender_scores = recommender_df_2.loc[list_2,:][book_name].drop(labels = intersection)

    book_results = pd.concat([average_scores,
                              first_recommender_scores
                              ,second_recommender_scores],axis = 0).sort_values(ascending = True)
    combined_scores = list(book_results.index)

    return book_results
```

- The distance should be as close to 0 as possible for better result
- In this case due to RAM issues we only considered 2999 books

```python
average_similar_books("Stylin' with CSS: A Designer's Guide")
```

```
book_name
HTML Utopia                                                                  0.725749
CSS Mastery: Advanced Web Standards Solutions                                0.745110
HTML, XHTML, and CSS (Visual Quickstart Guide)                               0.751931
Head First HTML with CSS & XHTML                                             0.772356
Agile Web Development with Rails: A Pragmatic Guide                          0.779279
Willem de Kooning: The Late Paintings, the 1980s                             0.901753
Competing for Customers and Capital                                          0.906795
The Language of New Media Design: Theory and Practice                        0.921402
Doing Things with Things: The Design and Use of Everyday Objects             0.931589
Eric Meyer on CSS: Mastering the Language of Web Design with Cascading Style Sheets    0.950244
Name: Stylin' with CSS: A Designer's Guide, dtype: float64
```

5.Flask app

The function shown above is used to build a flask app:

```python
# imports
import pandas as pd
import pickle
import numpy as np
import os
from flask import Flask, request, Response, render_template, jsonify

# initialize the flask app
app = Flask('myApp')
### route 4: show a form to the user
# define the route
@app.route('/form')
# create the controller
def form():
    # return the view
    return render_template('form.html', combined_scores1= "", combined_scores2=
    "", combined_scores3= "")
# define the route
@app.route('/submit')
# create the controller
def submit():
    user_input = request.args
    book_name = user_input['book-title']
    data = str(book_name)

    recommender_df = pd.read_csv(os.getcwd() + '/assets/recommender_1.csv',
    index_col = 'book_name')
    recommender_df_2 = pd.read_csv(os.getcwd() + '/assets/recommender_2.csv',
    index_col = 'book_name')
```

```
# recommender function
    def average_similar_books(book_name):
        list_1 = list(recommender_df[book_name].sort_values()[1:6].index)
        list_2 = list(recommender_df_2[book_name].sort_values()[1:6].index)
        intersection = list(set(list_1).intersection(set(list_2)))
        intersection_recommender_1 =
        recommender_df.loc[list_1,:][book_name][intersection]
        intersection_recommender_2 =
        recommender_df_2.loc[list_2,:][book_name][intersection]
        average_scores = (intersection_recommender_1 +
        intersection_recommender_2) /2
        first_recommender_scores =
        recommender_df.loc[list_1,:][book_name].drop(labels = intersection)
        second_recommender_scores =
        recommender_df_2.loc[list_2,:][book_name].drop(labels = intersection)
        book_results =
        pd.concat([average_scores,first_recommender_scores,second_recommender_sc
        ores], axis = 0).sort_values(ascending = True)
        combined_scores = list(book_results.index)[:3]
        return combined_scores
    combined_scores1 = average_similar_books(data)[0]
    combined_scores2 = average_similar_books(data)[1]
    combined_scores3 = average_similar_books(data)[2]
    # return the view
    return render_template('form.html', combined_scores1= combined_scores1,
    combined_scores2= combined_scores2, combined_scores3= combined_scores3)
# run the app
if __name__ == '__main__':
    app.run(debug=True)
```

OUTPUT