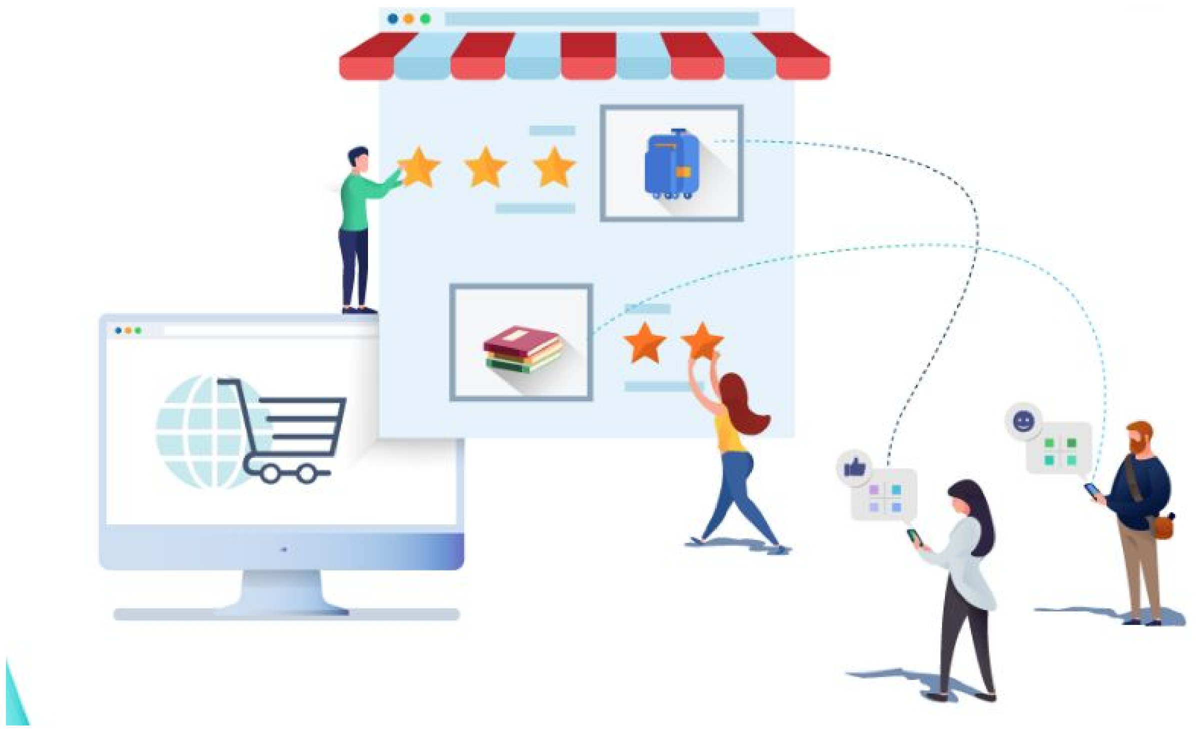
Executive summary

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.



Building a Scraper for data collection

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:

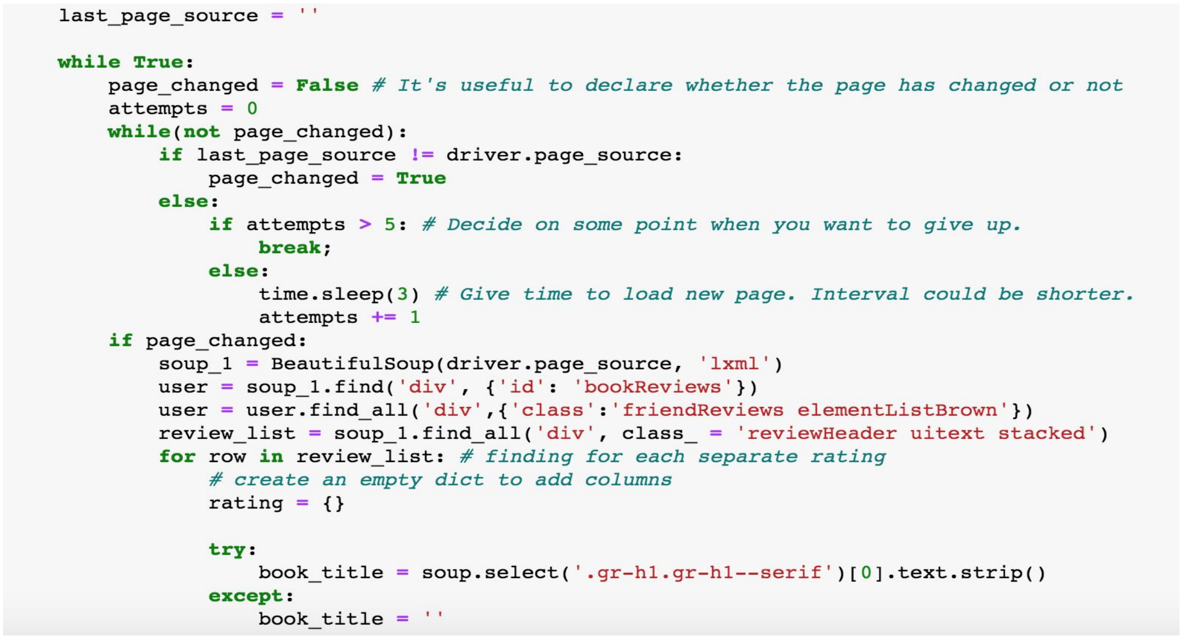
```python
import requests, json, time
from bs4 import BeautifulSoup
import pandas as pd
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
import os, sys
```

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.

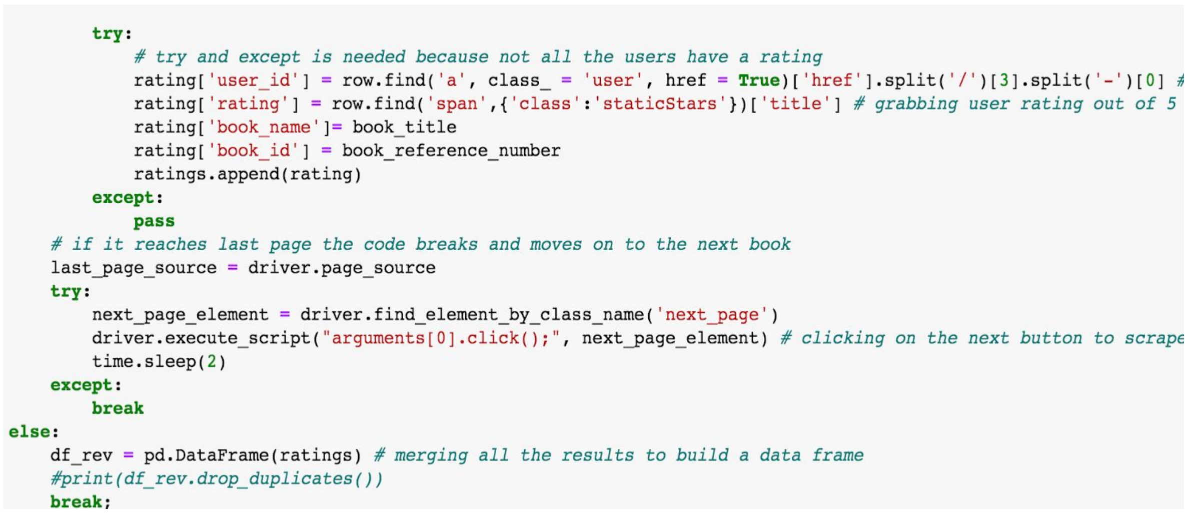
```python
driver = webdriver.Chrome(executable_path="./chromedriver/macos/chromedriver")
driver.get('https://www.goodreads.com/book/show/1')
```

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

```python
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 = ''
```

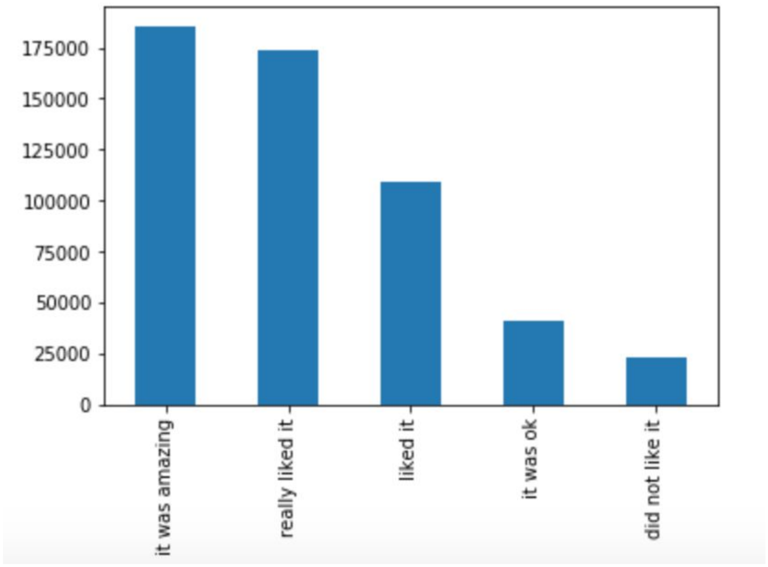


```
        try:
            # try and except is needed because not all the users have a rating
            rating['user_id'] = row.find('a', class_ = 'user', href = True)['href'].split('/')[3].split('-')[0] #
            rating['rating'] = row.find('span',{'class':'staticStars'})['title'] # grabbing 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;
```

Exploratory Data Analysis

1. User based recommender system:
   Since Selenium was used to scrape data, only required data was collected, thus there was very little data cleaning in the dataset. 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:

```
#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})
```