Ans- Recommender system

Hide   Copy Code

*# In[1]:*

*#importing libraries*

import pandas as pd

import numpy as np

pandas and numpy are two powerful libraries provided by python for scientific computation, data manipulation and data analysis. numpy; above all; provides high performance, multi-dimensional array along with the tools to manipulate it. Whereas pandas is known for its data structures and operations for manipulating data. We will be using both of these libraries in this article.

Hide   Copy Code

*# In[2]:*

*#reading the files*

data = pd.read\_csv('listing.csv', encoding = 'latin-1')

books = pd.read\_csv('books.csv', encoding = 'latin-1')

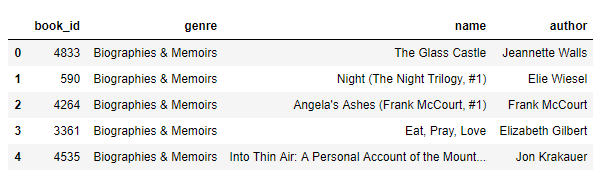
Hide   Copy Code

*# In[3]:*

*#using head() function to view first 5 rows for the object based on position.*

Just to test if we have right data.

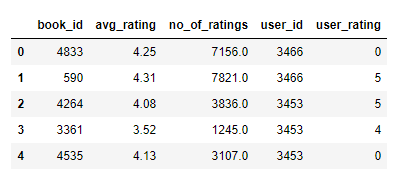
data.head()



Hide   Copy Code

*# In[4]:*

books.head()



Hide   Copy Code

*# In[5]:*

*# Getting recommendation based on No. Of ratings*

rating\_count = pd.DataFrame(books, columns=['book\_id','no\_of\_ratings'])

*# Sorting and dropping the duplicates*

rating\_count.sort\_values('no\_of\_ratings', ascending=False).drop\_duplicates().head(10)



Hide   Copy Code

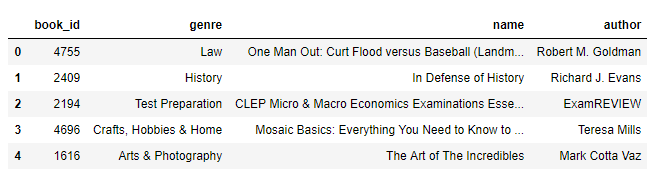
*# In[6]:*

*# getting the detail of 5 most rated books*

most\_rated\_books = pd.DataFrame([4755, 2409, 2194, 4696, 1616], index=np.arange(5), columns=['book\_id'])

detail = pd.merge(most\_rated\_books, data, on='book\_id')

detail



You can also get only the most rated book as follows:

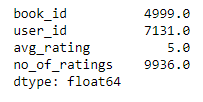
Hide   Copy Code

*# In[7]:*

*# getting the most rated book*

most\_rated\_book = pd.DataFrame(books, columns=['book\_id', 'user\_id', 'avg\_rating', 'no\_of\_ratings'])

most\_rated\_book.max()

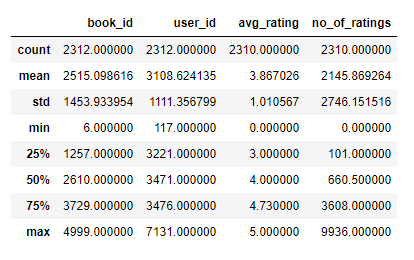


Hide   Copy Code

*# In[8]:*

*#getting description for most rated book*

most\_rated\_book.describe()



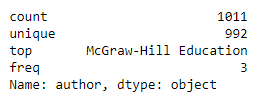
You can also get the description of any column using the same function.

Hide   Copy Code

*# In[9]:*

*# description for author*

data['author'].describe()



Correlation Based Recommender

As this is an age of more ‘personalized’ stuff so, popularity based recommenders are not enough to satisfy the need. Thus, there exist Correlation Based Recommenders which would make the recommendations based on the similarity of items (review similarity we’re talking about). The basic idea behind being that if you like this item, you are most probable to like an item similar to it. Correlation Based Recommenders are a simpler form of **collaborative filtering based recommenders**. They give you more flavor of being personalized as they would recommend the item that is most similar to the item selected before.

We are going to use **Pearson’s correlation** for our recommendation system. This recommendation system would use item based similarity; correlate the items based on user ratings.

Hide   Copy Code

*# In[1]:*

*# importing libraries*

import pandas as pd

import numpy as np

Hide   Copy Code

*# In[2]:*

*# reading files*

data = pd.read\_csv('listing.csv', encoding = 'latin-1')

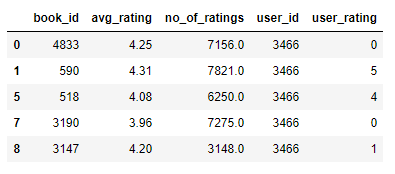
books = pd.read\_csv('books.csv', encoding = 'latin-1')

Hide   Copy Code

*# In[3]:*

*# Checking the data using head function*

books.head()



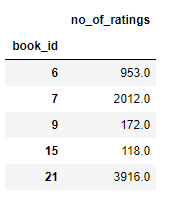
Hide   Copy Code

*# In[4]:*

*# calculating the mean*

rating = pd.DataFrame(books.groupby('book\_id')['no\_of\_ratings'].mean())

rating.head()

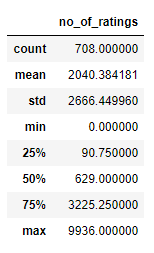


Hide   Copy Code

*# In[5]:*

*# getting the description of rating*

rating.describe()

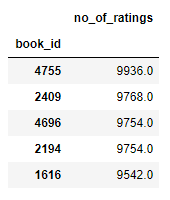


Hide   Copy Code

*# In[6]:*

*# sorting based on no of ratings that each book got*

rating.sort\_values('no\_of\_ratings', ascending=False).head()



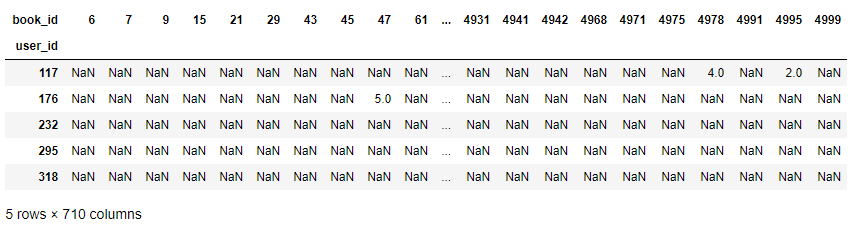
Hide   Copy Code

*# In[7]:*

*# Preparing data table for analysis*

ratings\_pivot = pd.pivot\_table(data=books, values='user\_rating', index='user\_id', columns='book\_id')

ratings\_pivot.head()



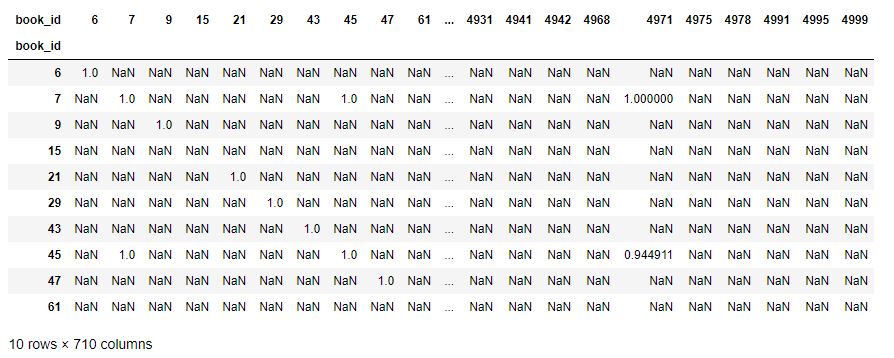
As we are interested in finding correlation between two variables, for that, we are going to use Pearson correlation which would simply measure the linear correlation. In this case, we are interested in knowing the relation between two books based on user rating.

Hide   Copy Code

*# In[8]:*

correlation\_matrix = user\_rating.corr(method='pearson')

correlation\_matrix.head(10)



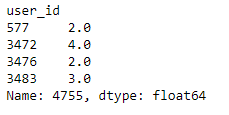
As you can see, now our table contains pearson correlation coefficient values.

Hide   Copy Code

*# getting the users who rated this particular book (most rated) and making sure rating is not zero*

OneManOut\_rating = ratings\_pivot[4755]

OneManOut\_rating[OneManOut\_rating>=0]



Hide   Copy Code

*# In[9]:*

*# finidng similar books to One Man Out book using Pearson correlation*

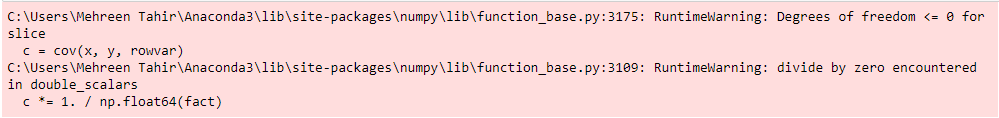
similar\_to\_OneManOut = ratings\_pivot.corrwith(OneManOut\_rating)

corr\_OneManOut = pd.DataFrame(similar\_to\_OneManOut, columns=['PearsonR'])

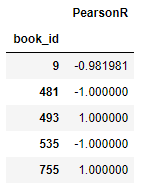
corr\_OneManOut.dropna(inplace=True)

corr\_OneManOut.head()

You’ll encounter a runtime warning because of encountering divide by zero.



But that will not get into our way so it can be ignored. We’ll still get the output as follows:



Hide   Copy Code

*# In[10]:*

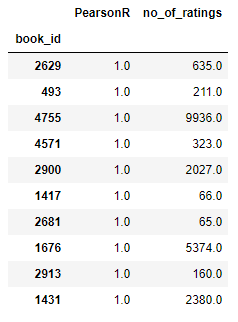
OneManOut\_corr\_summary = corr\_OneManOut.join(rating)

Hide   Copy Code

*# In[11]:*

*# getting the most similar book*

OneManOut\_corr\_summary.sort\_values('PearsonR', ascending=False).head(10)



Hide   Copy Code

*# In[12]:*

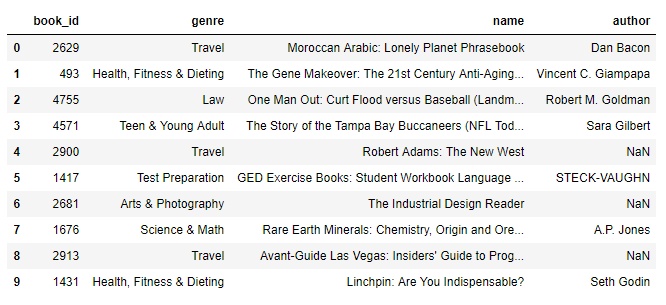
*# getting the details for most similar books*

book\_corr\_OneManOut = pd.DataFrame([2629, 493, 4755, 4571, 2900, 1417, 2681, 1676, 2913, 1431],

  index = np.arange(10), columns=['book\_id'])

summary = pd.merge(book\_corr\_OneManOut, data,on='book\_id')

summary



Now if you see most rated book in our dataset which is **One Man Out: Curt Flood Versus Baseball**is of law genre but our recommendation engine is giving us mixed recommendations including Travel, Law, etc. This is because we are using the relation between ratings to make our recommendation. This book was rated 4 times in our dataset and so was the very first recommended by our recommendation engine. It means our recommender is working.

Content Base Recommender

There exists another type of recommender known as content based recommender. This type of recommender uses the description of the item to recommend next most similar item. Content based recommenders also make the ‘personalized’ recommendation. The main difference between correlation based recommender and content based recommender is that the former considers the ‘user behavior’ while later considers the content for making recommendation. Content based recommender uses the product features or keywords used in description to find the similarity between the items. Let’s see how can we build one.

Hide   Copy Code

*# In[1]:*

*# importing libraries*

import pandas as pd

from sklearn.metrics.pairwise import linear\_kernel

from sklearn.feature\_extraction.text import TfidfVectorizer

linear\_kernel is used to compute the linear kernel between two variables. We would use this function instead of cosine\_similarities() because it is faster and as we are also using TF-IDF vectorization, a simple dot product will give us the same cosine similarity score. Now what is TF-IDF vector? We cannot compute the similarity between the given description in the form it is in our dataset. This is practically impossible. For this purpose, Term Frequency-Inverse Document Frequency (TF-IDF) is calculated for all the documents which would simply return you a matrix with each word representing a column. sklearn’s TfidfVectorizer would do this for us in a couple of lines:

Hide   Copy Code

*# In[2]:*

*# reading file*

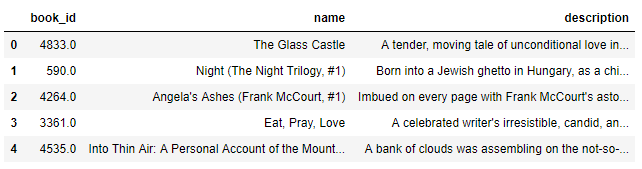
book\_description = pd.read\_csv('description.csv', encoding = 'latin-1')

Hide   Copy Code

*# In[3]:*

*# checking if we have the right data*

book\_description.head()



Hide   Copy Code

*# In[4]:*

*# removing the stop words*

books\_tfidf = TfidfVectorizer(stop\_words='english')

*# filling the missing values with empty string*

book\_description['description'] = book\_description['description'].fillna('')

*# computing TF-IDF matrix required for calculating cosine similarity*

book\_description\_matrix = books\_tfidf.fit\_transform(book\_description['description'])

Hide   Copy Code

*# In[5]:*

*# Let's check the shape of computed matrix*

book\_description\_matrix.shape

Image 21

The above shape means that 4186 words are used to describe 143 books in our dataset.

Hide   Copy Code

*# computing cosine similarity matrix using linear\_kernal of sklearn*

cosine\_similarity = linear\_kernel(book\_description\_matrix, book\_description\_matrix)

Hide   Copy Code

*# In[6]:*

indices = pd.Series(book\_description['name'].index)

Hide   Copy Code

*# In[7]:*

*# Function to get the most similar books*

def recommend(index, cosine\_sim=cosine\_similarity):

id = indices[index]

*# Get the pairwsie similarity scores of all books compared to that book,*

*# sorting them and getting top 5*

similarity\_scores = list(enumerate(cosine\_sim[id]))

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

similarity\_scores = similarity\_scores[1:6]

*# Get the books index*

books\_index = [i[0] for i in similarity\_scores]

*# Return the top 5 most similar books using integer-location based indexing (iloc)*

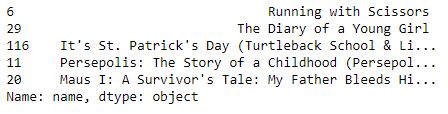
return book\_description['name'].iloc[books\_index]

Hide   Copy Code

*# In[8]:*

*# getting recommendation for book at index 2*

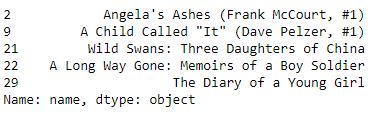
recommend(2)



*# In[9]:*

*# getting recommendation for book at index 6*

recommend(6)



If you notice the results we got; book at the index 2 is similar to book at index 6 according to our recommendation engine. Let’s follow along the description and see if our recommender is working.