Data Mining and Data Warehousing

COURSE CODE: CSE4005

Book Recommendation System

Under Guidance of :

Dr. Nagaraju Devarakonda

The Team:

Arnav Panigrahi - 18BCD7063

Chintala Sakshith Reddy - 18BCD7074

CODE:-

In [8]:
```
#Making necesarry imports
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.metrics as metrics
import numpy as np
from sklearn.neighbors import NearestNeighbors
from scipy.spatial.distance import correlation
from sklearn.metrics.pairwise import pairwise_distances
import ipywidgets as widgets
from IPython.display import display, clear_output
from contextlib import contextmanager
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import os, sys
import re
import seaborn as sns
```

In [10]:
```
from IPython.display import HTML
HTML('''<script>
code_show_err=false;
function code_toggle_err() {
 if (code_show_err){
 $('div.output_stderr').hide();
 } else {
 $('div.output_stderr').show();
 }
 code_show_err = !code_show_err
}
$( document ).ready(code_toggle_err);
</script>
To toggle on/off output_stderr, click <a href="javascript:code_toggle_err()">h
```

In [13]:
```
#Loading data
books = pd.read_csv('books.csv', sep=';', error_bad_lines=False, encoding="lat
books.columns = ['ISBN', 'bookTitle', 'bookAuthor', 'yearOfPublication', 'publ
users = pd.read_csv('users.csv', sep=';', error_bad_lines=False, encoding="lat
users.columns = ['userID', 'Location', 'Age']
ratings = pd.read_csv('ratings.csv', sep=';', error_bad_lines=False, encoding=
ratings.columns = ['userID', 'ISBN', 'bookRating']
```

```
b'Skipping line 6452: expected 8 fields, saw 9\nSkipping line 43667: expected
8 fields, saw 10\nSkipping line 51751: expected 8 fields, saw 9\n'
b'Skipping line 92038: expected 8 fields, saw 9\nSkipping line 104319: expect
ed 8 fields, saw 9\nSkipping line 121768: expected 8 fields, saw 9\n'
b'Skipping line 144058: expected 8 fields, saw 9\nSkipping line 150789: expec
ted 8 fields, saw 9\nSkipping line 157128: expected 8 fields, saw 9\nSkipping
line 180189: expected 8 fields, saw 9\nSkipping line 185738: expected 8 field
s, saw 9\n'
b'Skipping line 209388: expected 8 fields, saw 9\nSkipping line 220626: expec
ted 8 fields, saw 9\nSkipping line 227933: expected 8 fields, saw 11\nSkippin
g line 228957: expected 8 fields, saw 10\nSkipping line 245933: expected 8 fi
elds, saw 9\nSkipping line 251296: expected 8 fields, saw 9\nSkipping line 25
9941: expected 8 fields, saw 9\nSkipping line 261529: expected 8 fields, saw
9\n'
```

In [18]:
```
#checking shapes of the datasets
print (books.shape)
print (users.shape)
print (ratings.shape)
```

```
(271360, 5)
(278858, 3)
(1149780, 3)
```

In [19]:
```
#Exploring books dataset
books.head()
```

Out[19]:

	ISBN	bookTitle	bookAuthor	yearOfPublication	publisher
0	0195153448	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press
1	0002005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada
2	0060973129	Decision in Normandy	Carlo D'Este	1991	HarperPerennial
3	0374157065	Flu: The Story of the Great Influenza Pandemic...	Gina Bari Kolata	1999	Farrar Straus Giroux
4	0393045218	The Mummies of Urumchi	E. J. W. Barber	1999	W. W. Norton & Company

In [20]:
```
#Now the books datasets looks like....
books.head()
```

Out[20]:

	ISBN	bookTitle	bookAuthor	yearOfPublication	publisher
0	0195153448	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press
1	0002005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada
2	0060973129	Decision in Normandy	Carlo D'Este	1991	HarperPerennial
3	0374157065	Flu: The Story of the Great Influenza Pandemic...	Gina Bari Kolata	1999	Farrar Straus Giroux
4	0393045218	The Mummies of Urumchi	E. J. W. Barber	1999	W. W. Norton & Company

In [21]:
```
#checking data types of columns
books.dtypes
```

Out[21]:
```
ISBN                 object
bookTitle            object
bookAuthor           object
yearOfPublication    object
publisher            object
dtype: object
```

In [22]:
```
#making this setting to display full text in columns
pd.set_option('display.max_colwidth', -1)
```

In [22]:
```
#making this setting to display full text in columns
pd.set_option('display.max_colwidth', -1)
```

yearOfPublication

In [23]:
```
#yearOfPublication should be set as having dtype as int
#checking the unique values of yearOfPublication
books.yearOfPublication.unique()

#as it can be seen from below that there are some incorrect entries in this fi
#'DK Publishing Inc' and 'Gallimard' have been incorrectly loaded as yearOfPub
#Also some of the entries are strings and same years have been entered as numb
```

Out[23]:
```
array([2002, 2001, 1991, 1999, 2000, 1993, 1996, 1988, 2004, 1998, 1994,
       2003, 1997, 1983, 1979, 1995, 1982, 1985, 1992, 1986, 1978, 1980,
       1952, 1987, 1990, 1981, 1989, 1984, 0, 1968, 1961, 1958, 1974,
       1976, 1971, 1977, 1975, 1965, 1941, 1970, 1962, 1973, 1972, 1960,
       1966, 1920, 1956, 1959, 1953, 1951, 1942, 1963, 1964, 1969, 1954,
       1950, 1967, 2005, 1957, 1940, 1937, 1955, 1946, 1936, 1930, 2011,
       1925, 1948, 1943, 1947, 1945, 1923, 2020, 1939, 1926, 1938, 2030,
       1911, 1904, 1949, 1932, 1928, 1929, 1927, 1931, 1914, 2050, 1934,
       1910, 1933, 1902, 1924, 1921, 1900, 2038, 2026, 1944, 1917, 1901,
       2010, 1908, 1906, 1935, 1806, 2021, '2000', '1995', '1999', '2004',
       '2003', '1990', '1994', '1986', '1989', '2002', '1981', '1993',
       '1983', '1982', '1976', '1991', '1977', '1998', '1992', '1996',
       '0', '1997', '2001', '1974', '1968', '1987', '1984', '1988',
       '1963', '1956', '1970', '1985', '1978', '1973', '1980', '1979',
       '1975', '1969', '1961', '1965', '1939', '1958', '1950', '1953',
       '1966', '1971', '1959', '1972', '1955', '1957', '1945', '1960',
       '1967', '1932', '1924', '1964', '2012', '1911', '1927', '1948',
       '1962', '2006', '1952', '1940', '1951', '1931', '1954', '2005',
       '1930', '1941', '1944', 'DK Publishing Inc', '1943', '1938',
       '1900', '1942', '1923', '1920', '1933', 'Gallimard', '1909',
       '1946', '2008', '1378', '2030', '1936', '1947', '2011', '2020',
       '1919', '1949', '1922', '1897', '2024', '1376', '1926', '2037'],
      dtype=object)
```

In [24]:
```
#investigating the rows having 'DK Publishing Inc' as yearOfPublication
books.loc[books.yearOfPublication == 'DK Publishing Inc',:]
```

Out[24]:

	ISBN	bookTitle	bookAuthor	yearOfPublication	
209538	078946697X	DK Readers: Creating the X-Men, How It All Began (Level 4: Proficient Readers)\";Michael Teitelbaum"	2000	DK Publishing Inc	http://images.amazon.com/ir
221678	0789466953	DK Readers: Creating the X-Men, How Comic Books Come to Life (Level 4: Proficient Readers)\";James Buckley"	2000	DK Publishing Inc	http://images.amazon.com/ir

In [25]:
```
#From above, it is seen that bookAuthor is incorrectly loaded with bookTitle,
#ISBN '0789466953'
books.loc[books.ISBN == '0789466953','yearOfPublication'] = 2000
books.loc[books.ISBN == '0789466953','bookAuthor'] = "James Buckley"
books.loc[books.ISBN == '0789466953','publisher'] = "DK Publishing Inc"
books.loc[books.ISBN == '0789466953','bookTitle'] = "DK Readers: Creating the
```

In [26]:
```
#ISBN '078946697X'
books.loc[books.ISBN == '078946697X','yearOfPublication'] = 2000
books.loc[books.ISBN == '078946697X','bookAuthor'] = "Michael Teitelbaum"
books.loc[books.ISBN == '078946697X','publisher'] = "DK Publishing Inc"
books.loc[books.ISBN == '078946697X','bookTitle'] = "DK Readers: Creating the
```

In [27]:
```
#rechecking
books.loc[(books.ISBN == '0789466953') | (books.ISBN == '078946697X'),:]
#corrections done
```

Out[27]:

	ISBN	bookTitle	bookAuthor	yearOfPublication	publisher
209538	078946697X	DK Readers: Creating the X-Men, How It All Began (Level 4: Proficient Readers)	Michael Teitelbaum	2000	DK Publishing Inc
221678	0789466953	DK Readers: Creating the X-Men, How Comic Books Come to Life (Level 4: Proficient Readers)	James Buckley	2000	DK Publishing Inc

In [28]:
```
#investigating the rows having 'Gallimard' as yearOfPublication
books.loc[books.yearOfPublication == 'Gallimard',:]
```

Out[28]:

	ISBN	bookTitle	bookAuthor	yearOfPublication	
220731	2070426769	Peuple du ciel, suivi de 'Les Bergers\";Jean-Marie Gustave Le ClÃ?Â©zio"	2003	Gallimard	http://images.amazon.com/imag

In [29]:
```
#making required corrections as above, keeping other fields intact
books.loc[books.ISBN == '2070426769','yearOfPublication'] = 2003
books.loc[books.ISBN == '2070426769','bookAuthor'] = "Jean-Marie Gustave Le Cl
books.loc[books.ISBN == '2070426769','publisher'] = "Gallimard"
books.loc[books.ISBN == '2070426769','bookTitle'] = "Peuple du ciel, suivi de
```

In [31]:
```
#Correcting the dtypes of yearOfPublication
books.yearOfPublication=pd.to_numeric(books.yearOfPublication, errors='coerce'
```

In [32]:
```
print (sorted(books['yearOfPublication'].unique()))
#Now it can be seen that yearOfPublication has all values as integers
```

```
[0, 1376, 1378, 1806, 1897, 1900, 1901, 1902, 1904, 1906, 1908, 1909, 1910, 1
911, 1914, 1917, 1919, 1920, 1921, 1922, 1923, 1924, 1925, 1926, 1927, 1928,
1929, 1930, 1931, 1932, 1933, 1934, 1935, 1936, 1937, 1938, 1939, 1940, 1941,
1942, 1943, 1944, 1945, 1946, 1947, 1948, 1949, 1950, 1951, 1952, 1953, 1954,
1955, 1956, 1957, 1958, 1959, 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967,
1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980,
1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993,
1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006,
2008, 2010, 2011, 2012, 2020, 2021, 2024, 2026, 2030, 2037, 2038, 2050]
```

In [33]:
```
#However, the value 0 is invalid and as this dataset was published in 2004, I
#invalid keeping some margin in case dataset was updated thereafer
#setting invalid years as NaN
books.loc[(books.yearOfPublication > 2006) | (books.yearOfPublication == 0),'y
```

In [34]:
```
#replacing NaNs with mean value of yearOfPublication
books.yearOfPublication.fillna(round(books.yearOfPublication.mean()), inplace=
```

In [35]:
```
#rechecking
books.yearOfPublication.isnull().sum()
#No NaNs
```

Out[35]:
```
0
```

In [36]:
```
#resetting the dtype as int32
books.yearOfPublication = books.yearOfPublication.astype(np.int32)
```

publisher

In [37]:
```
#exploring 'publisher' column
books.loc[books.publisher.isnull(),:]
#two NaNs
```

Out[37]:

	ISBN	bookTitle	bookAuthor	yearOfPublication	publisher
128890	193169656X	Tyrant Moon	Elaine Corvidae	2002	NaN
129037	1931696993	Finders Keepers	Linnea Sinclair	2001	NaN

In [38]:
```
#investigating rows having NaNs
#Checking with rows having bookTitle as Tyrant Moon to see if we can get any c
books.loc[(books.bookTitle == 'Tyrant Moon'),:]
#no clues
```

Out[38]:

	ISBN	bookTitle	bookAuthor	yearOfPublication	publisher
128890	193169656X	Tyrant Moon	Elaine Corvidae	2002	NaN

In [39]:
```
#Checking with rows having bookTitle as Finder Keepers to see if we can get an
books.loc[(books.bookTitle == 'Finders Keepers'),:]
#all rows with different publisher and bookAuthor
```

Out[39]:

	ISBN	bookTitle	bookAuthor	yearOfPublication	publisher
10799	082177364X	Finders Keepers	Fern Michaels	2002	Zebra Books
42019	0070465037	Finders Keepers	Barbara Nickolae	1989	McGraw-Hill Companies
58264	0688118461	Finders Keepers	Emily Rodda	1993	Harpercollins Juvenile Books
66678	1575663236	Finders Keepers	Fern Michaels	1998	Kensington Publishing Corporation
129037	1931696993	Finders Keepers	Linnea Sinclair	2001	NaN
134309	0156309505	Finders Keepers	Will	1989	Voyager Books
173473	0973146907	Finders Keepers	Sean M. Costello	2002	Red Tower Publications
195885	0061083909	Finders Keepers	Sharon Sala	2003	HarperTorch
211874	0373261160	Finders Keepers	Elizabeth Travis	1993	Worldwide Library

In [40]:
```
#checking by bookAuthor to find patterns
books.loc[(books.bookAuthor == 'Elaine Corvidae'),:]
#all having different publisher...no clues here
```

Out[40]:

	ISBN	bookTitle	bookAuthor	yearOfPublication	publisher
126762	1931696934	Winter's Orphans	Elaine Corvidae	2001	Novelbooks
128890	193169656X	Tyrant Moon	Elaine Corvidae	2002	NaN
129001	0759901880	Wolfkin	Elaine Corvidae	2001	Hard Shell Word Factory

In [41]:
```
#checking by bookAuthor to find patterns
books.loc[(books.bookAuthor == 'Linnea Sinclair'),:]
```

Out[41]:

	ISBN	bookTitle	bookAuthor	yearOfPublication	publisher
129037	1931696993	Finders Keepers	Linnea Sinclair	2001	NaN

In [42]:
```
#since there is nothing in common to infer publisher for NaNs, replacing these
books.loc[(books.ISBN == '193169656X'),'publisher'] = 'other'
books.loc[(books.ISBN == '1931696993'),'publisher'] = 'other'
```

Users

In [43]:
```
print (users.shape)
users.head()
```

```
(278858, 3)
```

Out[43]:

	userID	Location	Age
0	1	nyc, new york, usa	NaN
1	2	stockton, california, usa	18.0
2	3	moscow, yukon territory, russia	NaN
3	4	porto, v.n.gaia, portugal	17.0
4	5	farnborough, hants, united kingdom	NaN

In [44]:
```
users.dtypes
```

Out[44]:
```
userID      int64
Location    object
Age         float64
dtype: object
```

userID

In [45]:
```
users.userID.values
#it can be seen that these are unique
```

Out[45]:
```
array([     1,      2,      3, ..., 278856, 278857, 278858], dtype=int64)
```

In [46]:
```
print (sorted(users.Age.unique()))
#Age column has some invalid entries like nan, 0 and very high values like 100
```

```
[nan, 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.
0, 14.0, 15.0, 16.0, 17.0, 18.0, 19.0, 20.0, 21.0, 22.0, 23.0, 24.0, 25.0, 2
6.0, 27.0, 28.0, 29.0, 30.0, 31.0, 32.0, 33.0, 34.0, 35.0, 36.0, 37.0, 38.0,
39.0, 40.0, 41.0, 42.0, 43.0, 44.0, 45.0, 46.0, 47.0, 48.0, 49.0, 50.0, 51.0,
52.0, 53.0, 54.0, 55.0, 56.0, 57.0, 58.0, 59.0, 60.0, 61.0, 62.0, 63.0, 64.0,
65.0, 66.0, 67.0, 68.0, 69.0, 70.0, 71.0, 72.0, 73.0, 74.0, 75.0, 76.0, 77.0,
78.0, 79.0, 80.0, 81.0, 82.0, 83.0, 84.0, 85.0, 86.0, 87.0, 88.0, 89.0, 90.0,
91.0, 92.0, 93.0, 94.0, 95.0, 96.0, 97.0, 98.0, 99.0, 100.0, 101.0, 102.0, 10
3.0, 104.0, 105.0, 106.0, 107.0, 108.0, 109.0, 110.0, 111.0, 113.0, 114.0, 11
5.0, 116.0, 118.0, 119.0, 123.0, 124.0, 127.0, 128.0, 132.0, 133.0, 136.0, 13
7.0, 138.0, 140.0, 141.0, 143.0, 146.0, 147.0, 148.0, 151.0, 152.0, 156.0, 15
7.0, 159.0, 162.0, 168.0, 172.0, 175.0, 183.0, 186.0, 189.0, 199.0, 200.0, 20
1.0, 204.0, 207.0, 208.0, 209.0, 210.0, 212.0, 219.0, 220.0, 223.0, 226.0, 22
8.0, 229.0, 230.0, 231.0, 237.0, 239.0, 244.0]
```

In [47]:
```
#In my view values below 5 and above 90 do not make much sense for our book ra
users.loc[(users.Age > 90) | (users.Age < 5), 'Age'] = np.nan
```

In [48]:
```
#replacing NaNs with mean
users.Age = users.Age.fillna(users.Age.mean())
```

In [49]:
```
#setting the data type as int
users.Age = users.Age.astype(np.int32)
```

In [50]:
```
#rechecking
print (sorted(users.Age.unique()))
#looks good now
```

```
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 2
5, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 4
4, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 6
3, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 8
2, 83, 84, 85, 86, 87, 88, 89, 90]
```

Ratings Dataset

In [51]:
```
#checking shape
ratings.shape
```

Out[51]:
```
(1149780, 3)
```

In [52]:
```
#ratings dataset will have n_users*n_books entries if every user rated every i
n_users = users.shape[0]
n_books = books.shape[0]
print (n_users * n_books)
```

```
75670906880
```

In [53]:
```
#checking first few rows...
ratings.head(5)
```

Out[53]:

	userID	ISBN	bookRating
0	276725	034545104X	0
1	276726	0155061224	5
2	276727	0446520802	0
3	276729	052165615X	3
4	276729	0521795028	6

In [54]:
```
ratings.bookRating.unique()
```

Out[54]:
```
array([ 0,  5,  3,  6,  8,  7, 10,  9,  4,  1,  2], dtype=int64)
```

In [55]:
```
#ratings dataset should have books only which exist in our books dataset, unle
ratings_new = ratings[ratings.ISBN.isin(books.ISBN)]
```

In [56]:
```
print (ratings.shape)
print (ratings_new.shape)
#it can be seen that many rows having book ISBN not part of books dataset got
```

```
(1149780, 3)
(1031136, 3)
```

In [57]:
```
#ratings dataset should have ratings from users which exist in users dataset,
ratings = ratings[ratings.userID.isin(users.userID)]
```

In [58]:
```
print (ratings.shape)
print (ratings_new.shape)
#no new users added, hence we will go with above dataset ratings_new (1031136,
```

```
(1149780, 3)
(1031136, 3)
```

In [59]:
```
print ("number of users: " + str(n_users))
print ("number of books: " + str(n_books))
```

```
number of users: 278858
number of books: 271360
```

In [60]:
```
#Sparsity of dataset in %
sparsity=1.0-len(ratings_new)/float(n_users*n_books)
print ('The sparsity level of Book Crossing dataset is ' +  str(sparsity*100)
```

```
The sparsity level of Book Crossing dataset is 99.99863734155898 %
```

In [61]:
```
#As quoted in the description of the dataset -
#BX-Book-Ratings contains the book rating information. Ratings are either expl
#higher values denoting higher appreciation, or implicit, expressed by 0
ratings.bookRating.unique()
```

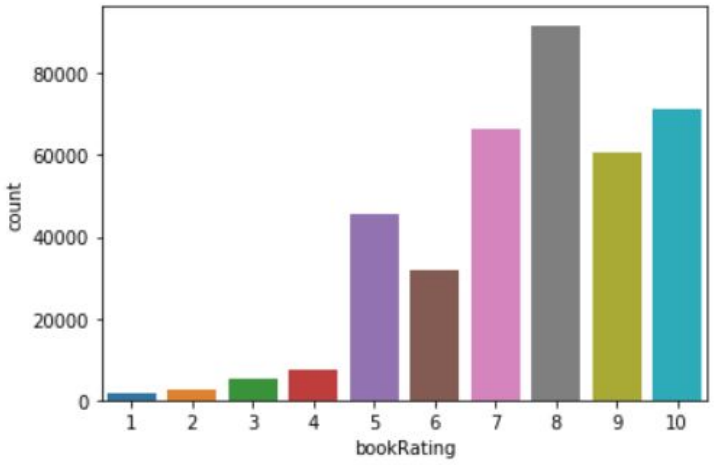
Out[61]:
```
array([ 0,  5,  3,  6,  8,  7, 10,  9,  4,  1,  2], dtype=int64)
```

In [62]:
```
#Hence segragating implicit and explict ratings datasets
ratings_explicit = ratings_new[ratings_new.bookRating != 0]
ratings_implicit = ratings_new[ratings_new.bookRating == 0]
```

In [63]:
```
#checking shapes
print (ratings_new.shape)
print (ratings_explicit.shape)
print (ratings_implicit.shape)
```

```
(1031136, 3)
(383842, 3)
(647294, 3)
```

In [64]:
```
#plotting count of bookRating
sns.countplot(data=ratings_explicit , x='bookRating')
plt.show()
#It can be seen that higher ratings are more common amongst users and rating 8
```



```
#At this point , a simple popularity based recommendation system can be built
ratings_count = pd.DataFrame(ratings_explicit.groupby(['ISBN'])['bookRating'].
top10 = ratings_count.sort_values('bookRating', ascending = False).head(10)
print ("Following books are recommended")
top10.merge(books, left_index = True, right_on = 'ISBN')

#Given below are top 10 recommendations based on popularity. It is evident tha
```

```
Following books are recommended
```

	bookRating	ISBN	bookTitle	bookAuthor	yearOfPublication	publisher
408	5787	0316666343	The Lovely Bones: A Novel	Alice Sebold	2002	Little, Brown
748	4108	0385504209	The Da Vinci Code	Dan Brown	2003	Doubleday
522	3134	0312195516	The Red Tent (Bestselling Backlist)	Anita Diamant	1998	Picador USA
2143	2798	059035342X	Harry Potter and the Sorcerer's Stone (Harry Potter (Paperback))	J. K. Rowling	1999	Arthur A. Levine Books
356	2595	0142001740	The Secret Life of Bees	Sue Monk Kidd	2003	Penguin Books
26	2551	0971880107	Wild Animus	Rich Shapero	2004	Too Far
1105	2524	0060928336	Divine Secrets of the Ya-Ya Sisterhood: A Novel	Rebecca Wells	1997	Perennial
706	2402	0446672211	Where the Heart Is (Oprah's Book Club (Paperback))	Billie Letts	1998	Warner Books
231	2219	0452282152	Girl with a Pearl Earring	Tracy Chevalier	2001	Plume Books
118	2179	0671027360	Angels & Demons	Dan Brown	2001	Pocket Star

```
#Similarly segregating users who have given explicit ratings from 1-10 and tho
users_exp_ratings = users[users.userID.isin(ratings_explicit.userID)]
users_imp_ratings = users[users.userID.isin(ratings_implicit.userID)]
```

```
#checking shapes
print (users.shape)
print (users_exp_ratings.shape)
print (users_imp_ratings.shape)
```

```
(278858, 3)
(68091, 3)
(52451, 3)
```

Collaborative Filtering Based Recommendation Systems

```
#To cope up with computing power I have and to reduce the dataset size, I am c
#and books which have atleast 100 ratings
counts1 = ratings_explicit['userID'].value_counts()
ratings_explicit = ratings_explicit[ratings_explicit['userID'].isin(counts1[co
counts = ratings_explicit['bookRating'].value_counts()
ratings_explicit = ratings_explicit[ratings_explicit['bookRating'].isin(counts
```

```
#Generating ratings matrix from explicit ratings table
ratings_matrix = ratings_explicit.pivot(index='userID', columns='ISBN', values
userID = ratings_matrix.index
ISBN = ratings_matrix.columns
print(ratings_matrix.shape)
ratings_matrix.head()
#Notice that most of the values are NaN (undefined) implying absence of rating
```

```
(449, 66574)
```

Out[207]:

ISBN	0000913154	0001046438	000104687X	0001047213	0001047973	000104799X	0001048082
userID							
2033	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2110	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2276	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4017	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4385	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 66574 columns

In [208]:
```
n_users = ratings_matrix.shape[0] #considering only those users who gave expli
n_books = ratings_matrix.shape[1]
print (n_users, n_books)
```

```
449 66574
```

In [209]:
```
#since NaNs cannot be handled by training algorithms, replacing these by 0, wh
#setting data type
ratings_matrix.fillna(0, inplace = True)
ratings_matrix = ratings_matrix.astype(np.int32)
```

In [210]:
```
#checking first few rows
ratings_matrix.head(5)
```

Out[210]:

ISBN	0000913154	0001046438	000104687X	0001047213	0001047973	000104799X	0001048082
userID							
2033	0	0	0	0	0	0	0
2110	0	0	0	0	0	0	0
2276	0	0	0	0	0	0	0
4017	0	0	0	0	0	0	0
4385	0	0	0	0	0	0	0

5 rows × 66574 columns

In [211]:
```
#rechecking the sparsity
sparsity=1.0-len(ratings_explicit)/float(users_exp_ratings.shape[0]*n_books)
print ('The sparsity level of Book Crossing dataset is ' +  str(sparsity*100)
```

```
The sparsity level of Book Crossing dataset is 99.99772184106935 %
```

Training our recommendation system

In [212]:
```
#setting global variables
global metric,k
k=10
metric='cosine'
```

User-based Recommendation System

In [213]:
```
#This function finds k similar users given the user_id and ratings matrix
#These similarities are same as obtained via using pairwise_distances
def findksimilarusers(user_id, ratings, metric = metric, k=k):
    similarities=[]
    indices=[]
    model_knn = NearestNeighbors(metric = metric, algorithm = 'brute')
    model_knn.fit(ratings)
    loc = ratings.index.get_loc(user_id)
    distances, indices = model_knn.kneighbors(ratings.iloc[loc, :].values.resh
    similarities = 1-distances.flatten()

    return similarities,indices
```

```
#This function predicts rating for specified user-item combination based on us
def predict_userbased(user_id, item_id, ratings, metric = metric, k=k):
    prediction=0
    user_loc = ratings.index.get_loc(user_id)
    item_loc = ratings.columns.get_loc(item_id)
    similarities, indices=findksimilarusers(user_id, ratings,metric, k) #simil
    mean_rating = ratings.iloc[user_loc,:].mean() #to adjust for zero based in
    sum_wt = np.sum(similarities)-1
    product=1
    wtd_sum = 0

    for i in range(0, len(indices.flatten())):
        if indices.flatten()[i] == user_loc:
            continue;
        else:
            ratings_diff = ratings.iloc[indices.flatten()[i],item_loc]-np.mean
            product = ratings_diff * (similarities[i])
            wtd_sum = wtd_sum + product

    #in case of very sparse datasets, using correlation metric for collaborati
    #which are handled here as below
    if prediction <= 0:
        prediction = 1
    elif prediction >10:
        prediction = 10

    prediction = int(round(mean_rating + (wtd_sum/sum_wt)))
    print ('\nPredicted rating for user {0} -> item {1}: {2}'.format(user_id,i

    return prediction
```

```
predict_userbased(11676,'0001056107',ratings_matrix);
```

```
Predicted rating for user 11676 -> item 0001056107: 2
```

Item-based Recommendation Systems

```
#This function finds k similar items given the item_id and ratings matrix

def findksimilaritems(item_id, ratings, metric=metric, k=k):
    similarities=[]
    indices=[]
    ratings=ratings.T
    loc = ratings.index.get_loc(item_id)
    model_knn = NearestNeighbors(metric = metric, algorithm = 'brute')
    model_knn.fit(ratings)

    distances, indices = model_knn.kneighbors(ratings.iloc[loc, :].values.resh
    similarities = 1-distances.flatten()

    return similarities,indices
```

```
similarities,indices=findksimilaritems('0001056107',ratings_matrix)
```

```
#This function predicts the rating for specified user-item combination based c
def predict_itembased(user_id, item_id, ratings, metric = metric, k=k):
    prediction= wtd_sum =0
    user_loc = ratings.index.get_loc(user_id)
    item_loc = ratings.columns.get_loc(item_id)
    similarities, indices=findksimilaritems(item_id, ratings) #similar users b
    sum_wt = np.sum(similarities)-1
    product=1
    for i in range(0, len(indices.flatten())):
        if indices.flatten()[i] == item_loc:
            continue;
        else:
            product = ratings.iloc[user_loc,indices.flatten()[i]] * (similarit
            wtd_sum = wtd_sum + product
    prediction = int(round(wtd_sum/sum_wt))

    #in case of very sparse datasets, using correlation metric for collaborati
    #which are handled here as below //code has been validated without the coa
    #predictions which might arise in case of very sparse datasets when using
    if prediction <= 0:
        prediction = 1
    elif prediction >10:
        prediction = 10

    print ('\nPredicted rating for user {0} -> item {1}: {2}'.format(user_id,i

    return prediction
```

```
prediction = predict_itembased(11676,'0001056107',ratings_matrix)
```

```
Predicted rating for user 11676 -> item 0001056107: 1
```

```
@contextmanager
def suppress_stdout():
    with open(os.devnull, "w") as devnull:
        old_stdout = sys.stdout
        sys.stdout = devnull
        try:
            yield
        finally:
            sys.stdout = old_stdout
```

```
#This function utilizes above functions to recommend items for item/user basea
#Recommendations are made if the predicted rating for an item is >= to 6,and t
def recommendItem(user_id, ratings, metric=metric):
    if (user_id not in ratings.index.values) or type(user_id) is not int:
        print ("User id should be a valid integer from this list :\n\n {} ".fo
    else:
        ids = ['Item-based (correlation)','Item-based (cosine)','User-based (c
        select = widgets.Dropdown(options=ids, value=ids[0],description='Selec
        def on_change(change):
            clear_output(wait=True)
            prediction = []
            if change['type'] == 'change' and change['name'] == 'value':
                if (select.value == 'Item-based (correlation)') | (select.valu
                    metric = 'correlation'
                else:
                    metric = 'cosine'
                with suppress_stdout():
                    if (select.value == 'Item-based (correlation)') | (select.
                        for i in range(ratings.shape[1]):
                            if (ratings[str(ratings.columns[i])][user_id] !=0)
                                prediction.append(predict_itembased(user_id, s
                            else:
                                prediction.append(-1) #for already rated items
                    else:
                        for i in range(ratings.shape[1]):
                            if (ratings[str(ratings.columns[i])][user_id] !=0)
                                prediction.append(predict_userbased(user_id, s
                            else:
                                prediction.append(-1) #for already rated items
                prediction = pd.Series(prediction)
                prediction = prediction.sort_values(ascending=False)
                recommended = prediction[:10]
                print ("As per {0} approach....Following books are recommended
                for i in range(len(recommended)):
                     print ("{0}. {1}".format(i+1,books.bookTitle[recommended.
        select.observe(on_change)
        display(select)
```

```
#checking for incorrect entries
recommendItem(999999,ratings_matrix)
```

```
User id should be a valid integer from this list :

   2033   2110   2276   4017   4385   5582   6242   6251   6543   6575
   7286   7346   8067   8245   8681   8890  10560  11676  11993  12538
  12824  12982  13552  13850  14422  15408  15418  16634  16795  16966
  17950  19085  21014  23768  23872  23902  25409  25601  25981  26535
  26544  26583  28591  28634  29259  30276  30511  30711  30735  30810
  31315  31556  31826  32773  33145  35433  35836  35857  35859  36299
  36554  36606  36609  36836  36907  37644  37712  37950  38023  38273
  38281  39281  39467  40889  40943  43246  43910  46398  47316  48025
  48494  49144  49889  51883  52199  52350  52584  52614  52917  53220
  55187  55490  55492  56271  56399  56447  56554  56959  59172  60244
  60337  60707  63714  63956  65258  66942  67840  68555  69078  69389
  69697  70415  70594  70666  72352  73681  75591  75819  76151  76223
  76499  76626  78553  78783  78834  78973  79441  81492  81560  83287
  83637  83671  85526  85656  86189  86947  87141  87555  88283  88677
  88693  88733  89602  91113  92652  92810  93047  93363  93629  94242
  94347  94853  94951  95010  95359  95902  95932  96448  97754  97874
  98391  98758 100459 100906 101209 101606 101851 102359 102647 102702
 102967 104399 104636 105028 105517 105979 106007 107784 107951 109574
 109901 109955 110483 110912 110934 110973 112001 113270 113519 114368
 114868 114988 115002 115003 116599 117384 120565 122429 122793 123094
 123608 123883 123981 125519 125774 126492 126736 127200 127359 128835
 129074 129716 129851 130554 130571 132492 132836 133747 134434 135149
 135265 136010 136139 136348 136382 138578 138844 140000 140358 141902
 142524 143175 143253 143415 145449 146113 146348 147847 148199 148258
 148744 148966 149907 149908 150979 153662 156150 156269 156300 156467
 157247 157273 158226 158295 158433 159506 160295 162052 162639 162738
 163759 163761 163804 163973 164096 164323 164533 164828 164905 165308
 165319 165758 166123 166596 168047 168245 169682 170513 170634 171118
 172030 172742 172888 173291 173415 174304 174892 177072 177432 177458
 178522 179718 179978 180378 180651 181176 182085 182086 182993 183958
 183995 184299 184532 185233 185384 187145 187256 187517 189139 189334
 189835 189973 190708 190925 193458 193560 193898 194600 196077 196160
 196502 197659 199416 200226 201290 203240 204864 205735 205943 206534
 207782 208406 208671 209516 210485 211426 211919 212965 214786 216012
```

```
recommendItem(4385, ratings_matrix)
```

Select approach Item-based (cosine)

```
As per Item-based (cosine) approach....Following books are recommended...
1. My Wicked Wicked Ways
2. Fair Peril
3. Wolfpointe
4. A Nest of Ninnies
5. A Bitter Legacy
6. A Hymn Before Battle
7. Thomas the Rhymer
8. Gatherer of Clouds (Initiate Brother Duology)
9. Wege zum Ruhm: 13 Hilfestellungen fÃ¼r junge KÃ¼nstler und 1 Warnung
10. Love In Bloom's
```

Conclusion:

The Book Recommendation System to provide the best suggestion to the user by analyzing the buyer's interest was achieved. The quality and the content were taken into consideration by employing content filtering, association rule mining and collaborative filtering.