Book_Crossing_recommender_system

February 9, 2021

1 Book- Crossing recomender system

1.1 Abstract:

The objective of this project is to develop a recommender application using the BookCrossing dataset collected by Cai-Nicolas Ziegler. The outcome of this system makes predictions utilizing the dataset that is up for cleaning and analysis. This accurately foresees the users' preferences of the books they read to that of others based on ratings

1.2 Ethical ML Framework

The purpose of this framework is to establish a set of standards prior to developing a book recommender system and undertake any scenario or dataset values. However, since the BookCrossing dataset has an open platform and collected with demographic information, many of the ethical ML framework principles do not apply. The website carrying this dataset does state the following message: "Freely available for research use when acknowledged with the following reference." This implies that Cai-Nicolas Ziegler wants its users to acknowledge the research group one is partaking in and the publications that may result from using the BookCrossing dataset. Therefore depending on the scenario and impact of the application, there would be more stricter measures in appropriating the techniques utilized for this project. The only data that were part of the model genration were the User ID, Books ISBN and the Rating of the users

1.2.1 Importing the Lib. and the datasets

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle
import warnings
import seaborn as sns
#from sklearn import model_selection
from surprise import Reader, Dataset
from surprise import model_selection, accuracy
from surprise import NMF
from surprise import SVD
from surprise import SVDpp
from surprise import CoClustering
```

```
from surprise import Dataset, Reader
from surprise import SVD, NMF
from surprise.model_selection import cross_validate, train_test_split,␣
 ↪GridSearchCV
from surprise import KNNBasic, KNNWithMeans, KNNWithZScore, KNNBaseline
import nbconvert



warnings.filterwarnings('ignore') # Turn off warnings
plt.style.use('seaborn-white') # Use seaborn-style plots
plt.rcParams.update({'font.size': 15}) # Set larger default plot font size
%matplotlib inline
```

The data set was downlaoded from "http://www2.informatik.uni-freiburg.de/~cziegler/BX/" and it is consisted of 3 files 1- Users Data 2- Books Data 3- Books Ratings Below is the data imported

[2]:
```
users = pd.read_csv ("C:/Users/16472/OneDrive/Documents/Data analytics/Advanced␣
 ↪course Big data/Course 2/Project 1/books/users.csv", sep=';',␣
 ↪encoding='ansi')
books = pd.read_csv ("C:/Users/16472/OneDrive/Documents/Data analytics/Advanced␣
 ↪course Big data/Course 2/Project 1/books/Books.csv", sep=';',␣
 ↪encoding='ansi', escapechar='\\')
ratings = pd.read_csv ("C:/Users/16472/OneDrive/Documents/Data analytics/
 ↪Advanced course Big data/Course 2/Project 1/books/Book-Ratings.csv", sep=';
 ↪', encoding='ansi')
```

[3]:
```
users.head()
```

[3]:

	User-ID	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

[4]:
```
ratings.head()
```

[4]:

	User-ID	ISBN	Book-Rating
0	276725	034545104X	0
1	276726	0155061224	5
2	276727	0446520802	0
3	276729	052165615X	3
4	276729	0521795028	6

[5]:
```
books.head()
```

[5]:

	ISBN	Book-Title	\
0	0195153448	Classical Mythology	
1	0002005018	Clara Callan	
2	0060973129	Decision in Normandy	

```
3  0374157065  Flu: The Story of the Great Influenza Pandemic...
4  0393045218                              The Mummies of Urumchi

            Book-Author  Year-Of-Publication                   Publisher  \
0    Mark P. O. Morford                 2002     Oxford University Press
1  Richard Bruce Wright                 2001       HarperFlamingo Canada
2          Carlo D'Este                 1991             HarperPerennial
3      Gina Bari Kolata                 1999        Farrar Straus Giroux
4       E. J. W. Barber                 1999  W. W. Norton &amp;amp; Company

                                          Image-URL-S  \
0  http://images.amazon.com/images/P/0195153448.0...
1  http://images.amazon.com/images/P/0002005018.0...
2  http://images.amazon.com/images/P/0060973129.0...
3  http://images.amazon.com/images/P/0374157065.0...
4  http://images.amazon.com/images/P/0393045218.0...

                                          Image-URL-M  \
0  http://images.amazon.com/images/P/0195153448.0...
1  http://images.amazon.com/images/P/0002005018.0...
2  http://images.amazon.com/images/P/0060973129.0...
3  http://images.amazon.com/images/P/0374157065.0...
4  http://images.amazon.com/images/P/0393045218.0...

                                          Image-URL-L
0  http://images.amazon.com/images/P/0195153448.0...
1  http://images.amazon.com/images/P/0002005018.0...
2  http://images.amazon.com/images/P/0060973129.0...
3  http://images.amazon.com/images/P/0374157065.0...
4  http://images.amazon.com/images/P/0393045218.0...
```

1.3 Exploring the data sets

[6]:
```
#checking the types of data in each data set
dtBooks = books.dtypes
dtratings = ratings.dtypes
dtusers = users.dtypes

print(dtBooks)
```

```
ISBN                  object
Book-Title            object
Book-Author           object
Year-Of-Publication    int64
Publisher             object
Image-URL-S           object
Image-URL-M           object
```

```
Image-URL-L            object
dtype: object
```

[7]:
```
print(dtratings)
```

```
User-ID         int64
ISBN           object
Book-Rating     int64
dtype: object
```

[8]:
```
print(dtusers)
```

```
User-ID       int64
Location     object
Age         float64
dtype: object
```

[9]:
```
books.info()
books.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271379 entries, 0 to 271378
Data columns (total 8 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   ISBN                 271379 non-null  object
 1   Book-Title           271379 non-null  object
 2   Book-Author          271378 non-null  object
 3   Year-Of-Publication  271379 non-null  int64
 4   Publisher            271377 non-null  object
 5   Image-URL-S          271379 non-null  object
 6   Image-URL-M          271379 non-null  object
 7   Image-URL-L          271379 non-null  object
dtypes: int64(1), object(7)
memory usage: 16.6+ MB
```

[9]:
```
       Year-Of-Publication
count        271379.000000
mean           1959.756050
std             258.011363
min               0.000000
25%            1989.000000
50%            1995.000000
75%            2000.000000
max            2050.000000
```

[10]:
```
ratings.info()
ratings.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1149780 entries, 0 to 1149779
Data columns (total 3 columns):
 #   Column       Non-Null Count    Dtype
---  ------       --------------    -----
 0   User-ID      1149780 non-null  int64
 1   ISBN         1149780 non-null  object
 2   Book-Rating  1149780 non-null  int64
dtypes: int64(2), object(1)
memory usage: 26.3+ MB
```

[10]:

	User-ID	Book-Rating
count	1.149780e+06	1.149780e+06
mean	1.403864e+05	2.866950e+00
std	8.056228e+04	3.854184e+00
min	2.000000e+00	0.000000e+00
25%	7.034500e+04	0.000000e+00
50%	1.410100e+05	0.000000e+00
75%	2.110280e+05	7.000000e+00
max	2.788540e+05	1.000000e+01

[11]:
```
users.columns = ['User_id',"Location","Age"]
users.info()
users.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 278858 entries, 0 to 278857
Data columns (total 3 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   User_id   278858 non-null  int64
 1   Location  278858 non-null  object
 2   Age       168096 non-null  float64
dtypes: float64(1), int64(1), object(1)
memory usage: 6.4+ MB
```

[11]:

	User_id	Age
count	278858.00000	168096.000000
mean	139429.50000	34.751434
std	80499.51502	14.428097
min	1.00000	0.000000
25%	69715.25000	24.000000
50%	139429.50000	32.000000
75%	209143.75000	44.000000

```
max    278858.00000     244.000000
```

[12]:
```
users.isna().sum()
users.isnull().sum()
```

[12]:
```
User_id          0
Location         0
Age         110762
dtype: int64
```

[13]:
```
ratings.isna().sum()
ratings.isnull().sum()
```

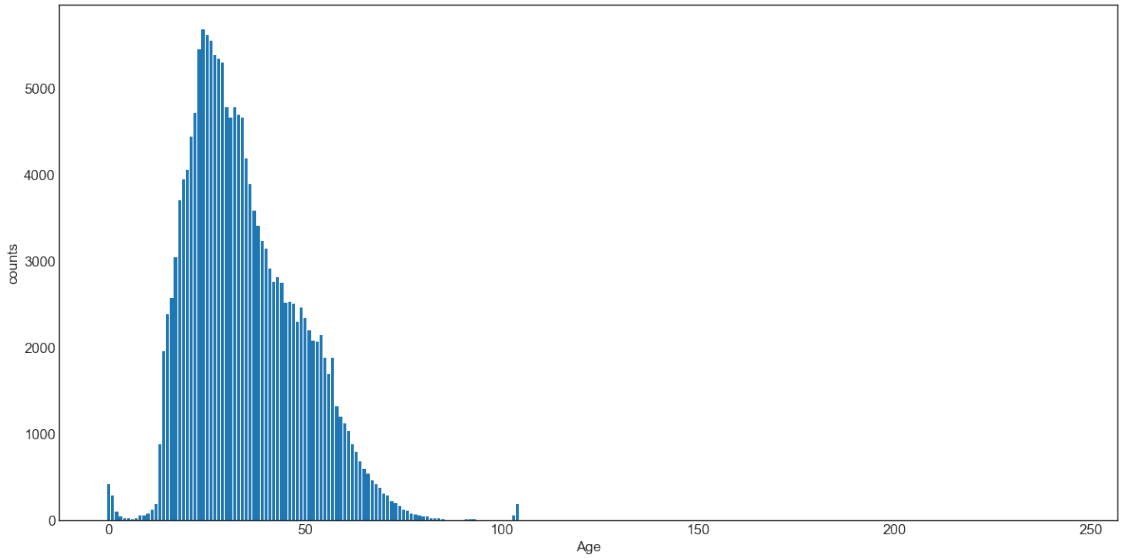
[13]:
```
User-ID        0
ISBN           0
Book-Rating    0
dtype: int64
```

[14]:
```
books.isna().sum()
#books.isnull().sum()
```

[14]:
```
ISBN                   0
Book-Title             0
Book-Author            1
Year-Of-Publication    0
Publisher              2
Image-URL-S            0
Image-URL-M            0
Image-URL-L            0
dtype: int64
```

1.4 Discovery of the users data set

[15]:
```
u = users.Age.value_counts().sort_index()
plt.figure(figsize=(20, 10))
plt.rcParams.update({'font.size': 15}) # Set larger plot font size
plt.bar(u.index, u.values)
plt.xlabel('Age')
plt.ylabel('counts')
plt.show()
```



From the above graph it seems that there are some oult liares as some users age is more than 100 and less than 10 years old So We will be cleaning users data set by changing all the values less tha more than 100 and less than 5 years old to na

[16]:
```
users.loc[(users.Age<5) | (users.Age>100), 'Age'] = np.nan
users1 = pd.DataFrame(users)
```

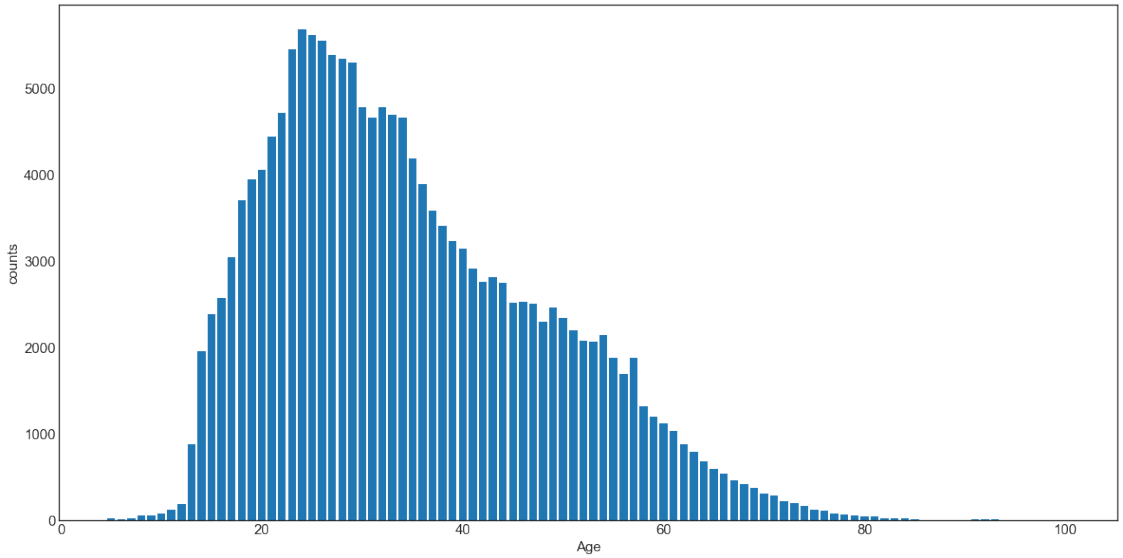
[17]:
```
print(sorted(users1.Age.unique()))
```

```
[nan, 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, 26.0, 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]
```

[18]:
```
#40% of the users does not have age value which is a big number so we will try␣
 ↪to fill all the na values with the same distribution of age
users1.Age.isnull().sum() / len(users)
```

[18]:
```
0.4016739702644357
```

[19]:
```
u = users1.Age.value_counts().sort_index()
plt.figure(figsize=(20, 10))
plt.rcParams.update({'font.size': 15}) # Set larger plot font size
plt.bar(u.index, u.values)
plt.xlabel('Age')
plt.ylabel('counts')
plt.show()
```



[20]:
```
users1.Age.fillna(users1.Age.mean()).describe()
```

[20]:
```
count    278858.000000
mean         34.746638
std          10.545361
min           5.000000
25%          29.000000
50%          34.746638
75%          35.000000
max         100.000000
Name: Age, dtype: float64
```

[21]:
```
# create a normal disrtribution pd.Series to fill Nan values with
temp_age_series = pd.Series(np.random.normal(loc=users1.Age.mean(),␣
 ↪scale=users1.Age.std(), size=users1.User_id[users1.Age.isna()].count()))
print("Describtion of the user1 dataset age")
print(users1.Age.describe())
print("Describtion of the Age series")
print(temp_age_series.describe())
print("checking the negative values in the distribution series")
print(temp_age_series[temp_age_series<0].count())


#As we can see the destribution doesnt change a lot. There are some negative␣
 ↪values which we will take only the absulate values for the age distribution
```

```
Describtion of the user1 dataset age
count    166848.000000
mean         34.746638
std          13.633051
```

```
min           5.000000
25%          24.000000
50%          32.000000
75%          44.000000
max         100.000000
Name: Age, dtype: float64
Describtion of the Age series
count    112010.000000
mean         34.743332
std          13.605077
min         -24.461690
25%          25.586021
50%          34.713044
75%          43.915006
max          93.426564
dtype: float64
checking the negative values in the distribution series
622
```

[22]:
```
# take the abs value of temp_Age_series
pos_age_series=np.abs(temp_age_series)

# sort users1 Df so as NaN values in Age to be first and reset index to match␣
 ↪with index of pos_Age_series. Then use fillna()
users1 = users1.sort_values('Age',na_position='first').reset_index(drop=True)
users1.Age.fillna(pos_age_series, inplace = True)
users1.loc[users1.Age<5, 'Age'] = users1.Age.mean()
users1.Age = users1.Age.round().astype(int)

#Sort users1 based on User-ID so as to be the same as before
users1 = users1.sort_values('User_id').reset_index(drop=True)
print(users1.Age.describe(),"\n")
users1.head()
```

```
count    278858.000000
mean         34.937907
std          13.369662
min           5.000000
25%          25.000000
50%          33.000000
75%          44.000000
max         100.000000
Name: Age, dtype: float64
```

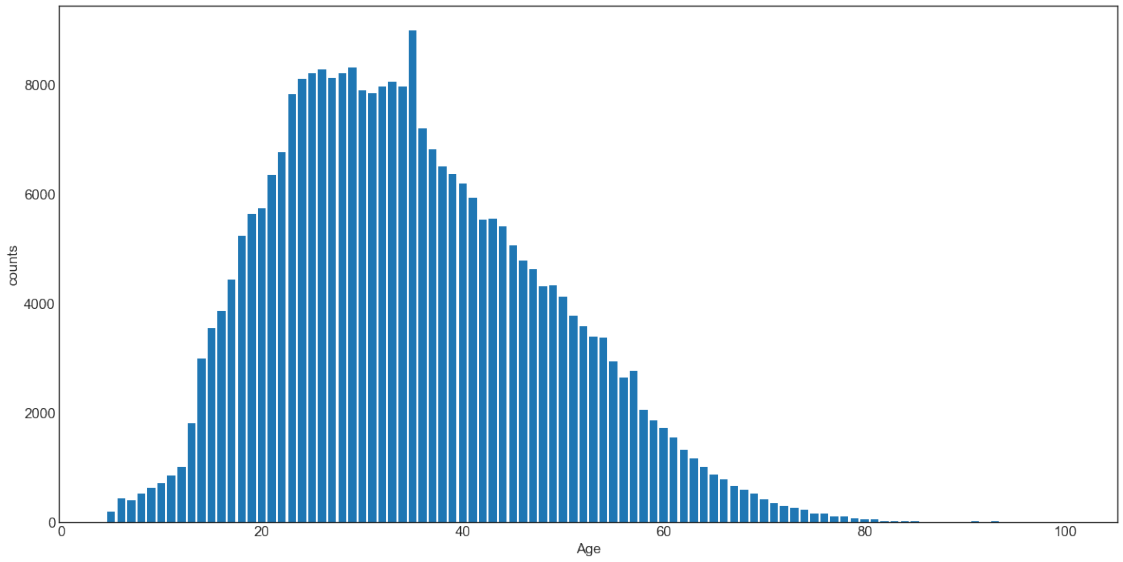
[22]:

	User_id	Location	Age
0	1	nyc, new york, usa	12

```
1        2           stockton, california, usa   18
2        3     moscow, yukon territory, russia   25
3        4           porto, v.n.gaia, portugal   17
4        5  farnborough, hants, united kingdom   22
```

As we can see the destribution doesnt change a lot. There are some negative values which we will take only the absulate values for the age distribution

[23]:
```
u = users1.Age.value_counts().sort_index()
plt.figure(figsize=(20, 10))
plt.rcParams.update({'font.size': 15}) # Set larger plot font size
plt.bar(u.index, u.values)
plt.xlabel('Age')
plt.ylabel('counts')
plt.show()

users1.Age.isnull().sum() / len(users)
```



[23]:
```
0.0
```

1.4.1 Exploring the countries of users

[24]:
```
#users.Location.str.split(",")
#users[["City", "State", "Country"]] = users.Location.str.split(",",␣
 ↪expand=True)

split_data = users1["Location"].str.split(",",2,expand=True)
split_data.columns = ["City", "State", "Country"]
users2 = users1.join(split_data)
del users2["Location"]
```

```
print(users2)
```

```
        User_id  Age         City             State          Country
0             1   12          nyc          new york              usa
1             2   18     stockton        california              usa
2             3   25       moscow   yukon territory           russia
3             4   17        porto          v.n.gaia         portugal
4             5   22  farnborough             hants   united kingdom
...         ...  ...          ...               ...              ...
278853   278854   23     portland            oregon              usa
278854   278855   50       tacoma        washington   united kingdom
278855   278856   37     brampton           ontario           canada
278856   278857   29    knoxville         tennessee              usa
278857   278858   35       dublin               n/a          ireland

[278858 rows x 5 columns]
```

[25]:
```
users2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 278858 entries, 0 to 278857
Data columns (total 5 columns):
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   User_id  278858 non-null  int64
 1   Age      278858 non-null  int32
 2   City     278858 non-null  object
 3   State    278857 non-null  object
 4   Country  278856 non-null  object
dtypes: int32(1), int64(1), object(3)
memory usage: 9.6+ MB
```

[26]:
```
users2['City'].value_counts().sort_values(ascending=False)
```

[26]:
```
london                    4105
barcelona                 2664
toronto                   2342
madrid                    1933
sydney                    1884
                          ...
corozal town                 1
castelnuovo magra            1
luberon                      1
san nicandro garganico       1
as                           1
Name: City, Length: 32770, dtype: int64
```

[27]:
```
users2['Country'].value_counts()
```

[27]:
```
usa                          139183
canada                        21556
united kingdom                18286
germany                       17021
spain                         13088
                              ...
livingston                        1
bosnia                            1
canterbury, new zealand           1
hernando                          1
camden                            1
Name: Country, Length: 1276, dtype: int64
```

Based on the values missing We will use the City for the analysis of the users and we will drop the Country and the State as most of the countries does not have states.

[28]:
```
print((users2["City"].values == '').sum())
print((users2["City"].isnull().sum()))
```

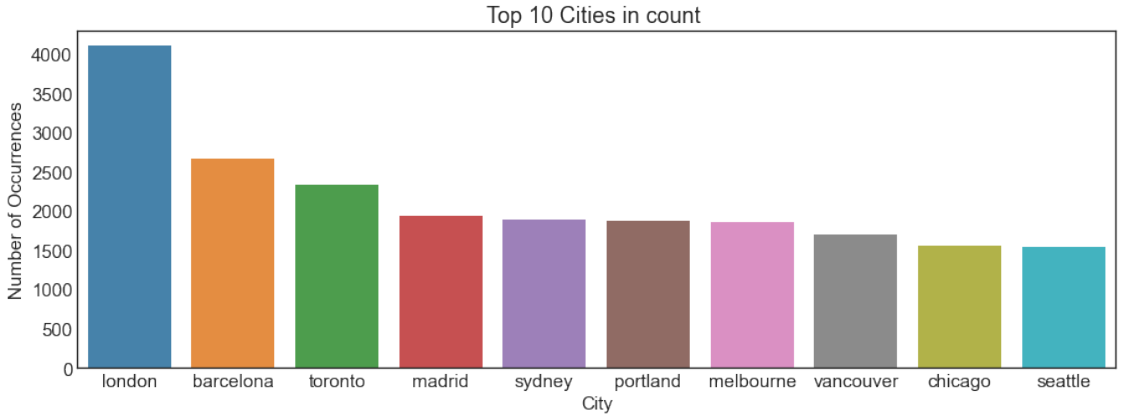
```
122
0
```

[29]:
```
nan_value = float("NaN")
users2.replace("", nan_value, inplace=True)
users2.dropna(subset = ["City"], inplace=True)
```

[30]:
```
print((users2["City"].values == '').sum())
print((users2["City"].isnull().sum()))
```

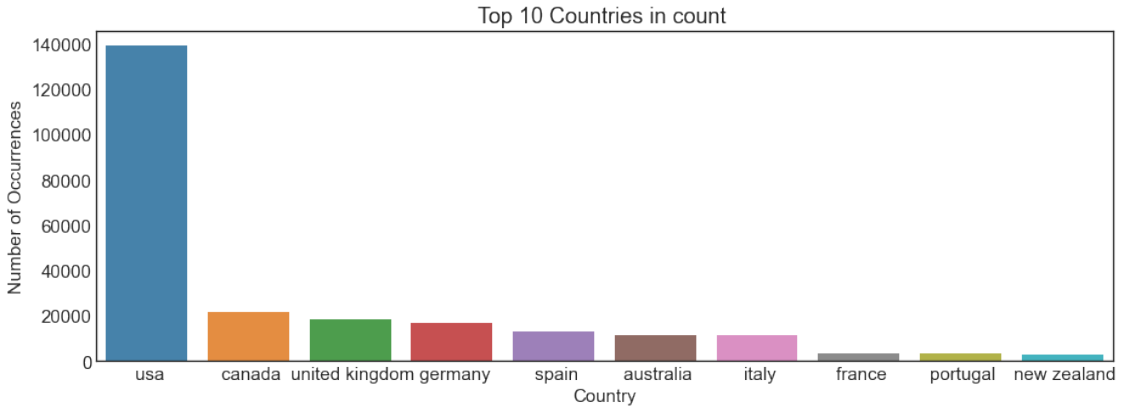
```
0
0
```

[31]:
```
city_count  = users2["City"].value_counts()
city_count = city_count[:10,]
plt.figure(figsize=(15,5))
sns.barplot(city_count.index, city_count.values, alpha=0.9)
plt.title('Top 10 Cities in count ')
plt.ylabel('Number of Occurrences', fontsize=15)
plt.xlabel('City', fontsize=15)
plt.show()
```



The chart above highlights the total number of readers per each city; where London, England ranks topmost in terms of majority of readers, followed by Barcelona, Spain and Toronto, Canada

[32]:
```
country_count  = users2["Country"].value_counts()
country_count = country_count[:10,]
plt.figure(figsize=(15,5))
sns.barplot(country_count.index, country_count.values, alpha=0.9)
plt.title('Top 10 Countries in count ')
plt.ylabel('Number of Occurrences', fontsize=15)
plt.xlabel('Country', fontsize=15)
plt.show()
```



The chart below highlights the countries from whom the majority of the responses from readers were recorded for the data purposes where United States racks up the majority count, followed by Canada and the United States

1.5 Exploring and Cleaning Books data set

[33]:
```
# first drop the image columns (Image-URL-S,Image-URL-M, and Image-URL-L) from
 ↪the data set
books.head()
```

[33]:
```
         ISBN                                        Book-Title  \
0  0195153448                                Classical Mythology
1  0002005018                                       Clara Callan
2  0060973129                              Decision in Normandy
3  0374157065  Flu: The Story of the Great Influenza Pandemic...
4  0393045218                            The Mummies of Urumchi

            Book-Author  Year-Of-Publication                   Publisher  \
0    Mark P. O. Morford                 2002     Oxford University Press
1  Richard Bruce Wright                 2001       HarperFlamingo Canada
2          Carlo D'Este                 1991             HarperPerennial
3     Gina Bari Kolata                 1999        Farrar Straus Giroux
4      E. J. W. Barber                 1999  W. W. Norton &amp;amp; Company

                                          Image-URL-S  \
0  http://images.amazon.com/images/P/0195153448.0...
1  http://images.amazon.com/images/P/0002005018.0...
2  http://images.amazon.com/images/P/0060973129.0...
3  http://images.amazon.com/images/P/0374157065.0...
4  http://images.amazon.com/images/P/0393045218.0...

                                          Image-URL-M  \
0  http://images.amazon.com/images/P/0195153448.0...
1  http://images.amazon.com/images/P/0002005018.0...
2  http://images.amazon.com/images/P/0060973129.0...
3  http://images.amazon.com/images/P/0374157065.0...
4  http://images.amazon.com/images/P/0393045218.0...

                                          Image-URL-L
0  http://images.amazon.com/images/P/0195153448.0...
1  http://images.amazon.com/images/P/0002005018.0...
2  http://images.amazon.com/images/P/0060973129.0...
3  http://images.amazon.com/images/P/0374157065.0...
4  http://images.amazon.com/images/P/0393045218.0...
```

[34]:
```
books.drop(['Image-URL-S','Image-URL-M', 'Image-URL-L'], axis=1, inplace=True)
```

[35]:
```
print(books.dtypes)
books.columns =
 ↪['ISBN',"Book_title","Book_Author","Year_of_Publication","Publisher"]
print(books.info())
print(books.describe())
```

```
ISBN                  object
Book-Title            object
Book-Author           object
Year-Of-Publication    int64
Publisher             object
dtype: object
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271379 entries, 0 to 271378
Data columns (total 5 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   ISBN                 271379 non-null  object
 1   Book_title           271379 non-null  object
 2   Book_Author          271378 non-null  object
 3   Year_of_Publication  271379 non-null  int64
 4   Publisher            271377 non-null  object
dtypes: int64(1), object(4)
memory usage: 10.4+ MB
None
       Year_of_Publication
count        271379.000000
mean           1959.756050
std             258.011363
min               0.000000
25%            1989.000000
50%            1995.000000
75%            2000.000000
max            2050.000000
```

[36]:
```
# check for na and empty with the columns of the books dataset
books_na = books.isna().sum()
books_null = books.isnull().sum()

title_empty = books[books.Book_title == ""].Book_title .count()
Author_empty = books[books.Book_Author == ""].Book_Author.count()
year_empty = books[books.Year_of_Publication == 0].Year_of_Publication.count()
Publisher_empty = books[books.Publisher == ""].Publisher.count()

print(f'count of na in the books:\\ {books_na}')
print(f'count of null in the books:\\ {books_null}')
print(f'count of  cells in the title:\\ {100*title_empty/books.Book_title.
 ↪count()} %')
print(f'count of empty cells in the Author:\\ {100*Author_empty/books.
 ↪Book_Author.count()} %')
print(f'count of cells in the years with "0":\\ {100*year_empty/books.
 ↪Year_of_Publication.count()} %')
```

```
print(f'count of empty cells in the Publisher:\\ {100*Publisher_empty/books.
 ↪Publisher.count()} %')
```

```
count of na in the books:\ ISBN                   0
Book_title             0
Book_Author            1
Year_of_Publication    0
Publisher              2
dtype: int64
count of null in the books:\ ISBN                   0
Book_title             0
Book_Author            1
Year_of_Publication    0
Publisher              2
dtype: int64
count of  cells in the title:\ 0.0 %
count of empty cells in the Author:\ 0.0 %
count of cells in the years with "0":\ 1.7020476897622883 %
count of empty cells in the Publisher:\ 0.0 %
```

[37]:
```
# Replace all years of zero with NaN
books.Year_of_Publication.replace(0, np.nan, inplace=True)

nan_value = float("NaN")
books.replace("", nan_value, inplace=True)
books.dropna(subset = ["Book_title"], inplace=True)
books.dropna(subset = ["Year_of_Publication"], inplace=True)
books.dropna(subset = ["Book_Author"], inplace=True)
books.dropna(subset = ["Publisher"], inplace=True)
```

[38]:
```
# check for na and empty with the columns of the books dataset
books_na = books.isna().sum()
books_null = books.isnull().sum()

title_empty = books[books.Book_title == ""].Book_title .count()
Author_empty = books[books.Book_Author == ""].Book_Author.count()
year_empty = books[books.Year_of_Publication == 0].Year_of_Publication.count()
Publisher_empty = books[books.Publisher == ""].Publisher.count()

print(f'count of na in the books:\\ {books_na}')
print(f'count of null in the books:\\ {books_null}')
print(f'count of  cells in the title:\\ {100*title_empty/books.Book_title.
 ↪count()} %')
print(f'count of empty cells in the Author:\\ {100*Author_empty/books.
 ↪Book_Author.count()} %')
print(f'count of cells in the years with "0":\\ {100*year_empty/books.
 ↪Year_of_Publication.count()} %')
```

```
print(f'count of empty cells in the Publisher:\\ {100*Publisher_empty/books.
 ↪Publisher.count()} %')
```

```
count of na in the books:\ ISBN                   0
Book_title             0
Book_Author            0
Year_of_Publication    0
Publisher              0
dtype: int64
count of null in the books:\ ISBN                   0
Book_title             0
Book_Author            0
Year_of_Publication    0
Publisher              0
dtype: int64
count of  cells in the title:\ 0.0 %
count of empty cells in the Author:\ 0.0 %
count of cells in the years with "0":\ 0.0 %
count of empty cells in the Publisher:\ 0.0 %
```

[39]:
```
#Checking the descriptive stats of the range of years the books were published␣
 ↪in the dataset
print(books.describe())
```

```
       Year_of_Publication
count        266757.000000
mean           1993.689534
std               8.326039
min            1376.000000
25%            1989.000000
50%            1996.000000
75%            2000.000000
max            2050.000000
```

1.5.1 It seems there are books that has dates of 1376 and books that has future dates. all of this will be cosidered as wrong inputs, so lets check the count of these books

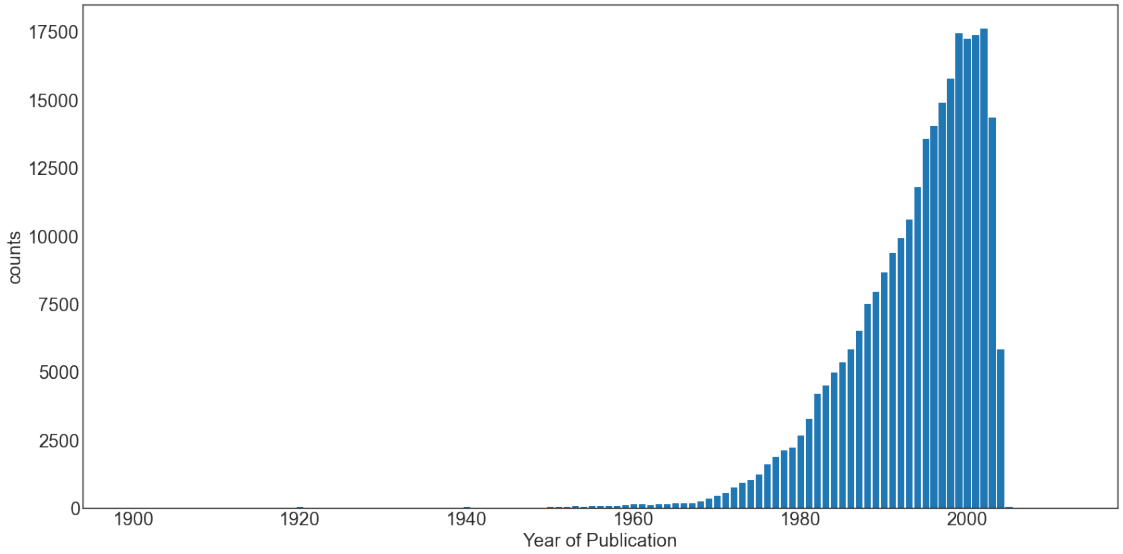
[40]:
```
#print(old_books = books[books.Year_of_Publication<1900])
print(f'outlaiers with old dates: {books[books.Year_of_Publication <1900].
 ↪Year_of_Publication.count()}')
print(f'outlaiers with future dates: {books[books.Year_of_Publication >2019].
 ↪Year_of_Publication.count()}')


print(f'Based on the count of the outlaiers we will remove the the rows for␣
 ↪these books')
```

```
outlaiers with old dates: 4
outlaiers with future dates: 17
Based on the count of the outlaiers we will remove the the rows for these books
```

[41]:
```
books = books[books['Year_of_Publication'] >= 1900 ]
books = books[books['Year_of_Publication'] <= 2019 ]
print(books.describe())
print(books.count())
```

```
       Year_of_Publication
count        266736.000000
mean           1993.692887
std               8.137504
min            1900.000000
25%            1989.000000
50%            1996.000000
75%            2000.000000
max            2012.000000
ISBN                266736
Book_title          266736
Book_Author         266736
Year_of_Publication 266736
Publisher           266736
dtype: int64
```

[42]:
```
# plot to show the years destribution

yr = books.Year_of_Publication.value_counts().sort_index()
plt.figure(figsize=(20, 10))
plt.rcParams.update({'font.size': 20})
plt.bar(yr.index, yr.values)
plt.xlabel('Year of Publication')
plt.ylabel('counts')
plt.show()
```



The graph below highlights the amount of books that were published through out the years with the adjusted dataset The distribution has a negative skewness, meaning as years progressed more books were published by each year (till mid-2000s)

[43]:
```
books.head(20)
```

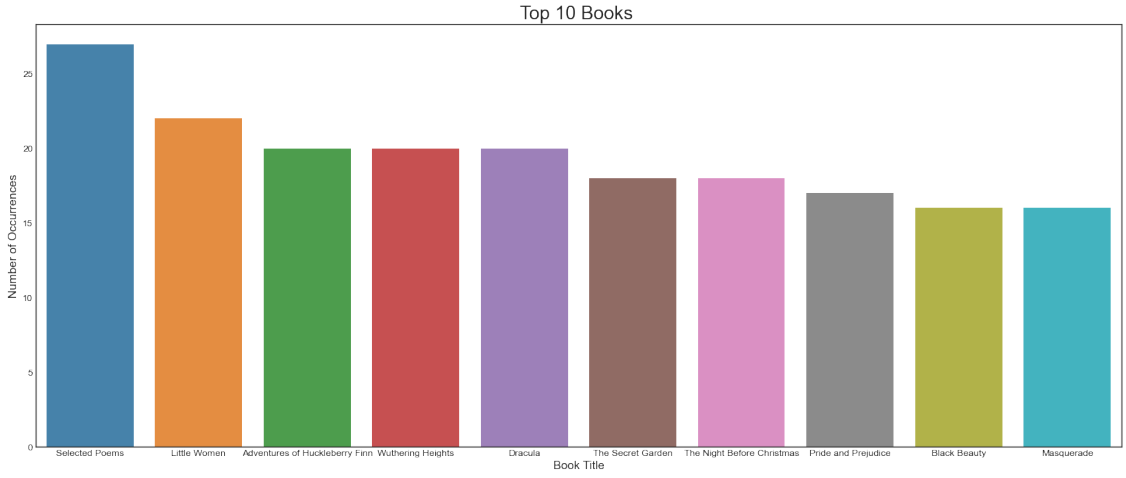
[43]:
```
           ISBN                                          Book_title  \
0    0195153448                                Classical Mythology
1    0002005018                                       Clara Callan
2    0060973129                              Decision in Normandy
3    0374157065  Flu: The Story of the Great Influenza Pandemic...
4    0393045218                           The Mummies of Urumchi
5    0399135782                           The Kitchen God's Wife
6    0425176428  What If?: The World's Foremost Military Histor...
7    0671870432                                PLEADING GUILTY
8    0679425608  Under the Black Flag: The Romance and the Real...
9    074322678X           Where You'll Find Me: And Other Stories
10   0771074670                        Nights Below Station Street
11   080652121X  Hitler's Secret Bankers: The Myth of Swiss Neu...
12   0887841740                              The Middle Stories
13   1552041778                                          Jane Doe
14   1558746218  A Second Chicken Soup for the Woman's Soul (Ch...
15   1567407781      The Witchfinder (Amos Walker Mystery Series)
16   1575663937  More Cunning Than Man: A Social History of Rat...
17   1881320189                    Goodbye to the Buttermilk Sky
18   0440234743                                   The Testament
19   0452264464             Beloved (Plume Contemporary Fiction)

             Book_Author  Year_of_Publication                  Publisher
0     Mark P. O. Morford               2002.0     Oxford University Press
```

1	Richard Bruce Wright	2001.0	HarperFlamingo Canada
2	Carlo D'Este	1991.0	HarperPerennial
3	Gina Bari Kolata	1999.0	Farrar Straus Giroux
4	E. J. W. Barber	1999.0	W. W. Norton & Company
5	Amy Tan	1991.0	Putnam Pub Group
6	Robert Cowley	2000.0	Berkley Publishing Group
7	Scott Turow	1993.0	Audioworks
8	David Cordingly	1996.0	Random House
9	Ann Beattie	2002.0	Scribner
10	David Adams Richards	1988.0	Emblem Editions
11	Adam Lebor	2000.0	Citadel Press
12	Sheila Heti	2004.0	House of Anansi Press
13	R. J. Kaiser	1999.0	Mira Books
14	Jack Canfield	1998.0	Health Communications
15	Loren D. Estleman	1998.0	Brilliance Audio - Trade
16	Robert Hendrickson	1999.0	Kensington Publishing Corp.
17	Julia Oliver	1994.0	River City Pub
18	John Grisham	1999.0	Dell
19	Toni Morrison	1994.0	Plume

It seems there is a problem with the "&" in the publisher name

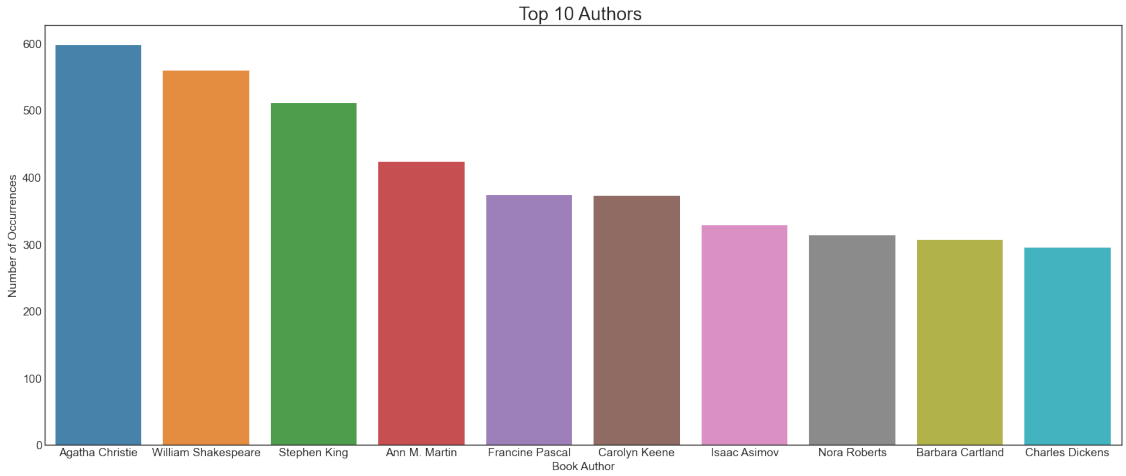
[44]:
```
books.Publisher = books.Publisher.str.replace('&amp', '&', regex=False)
```

[45]:
```
Top_books  = books["Book_title"].value_counts()
Top_books = Top_books[:10,]
plt.figure(figsize=(25,10))
sns.barplot(Top_books.index, Top_books.values, alpha=0.9)
plt.title('Top 10 Books ')
plt.ylabel('Number of Occurrences', fontsize=15)
plt.xlabel('Book Title', fontsize=15)
plt.tick_params(labelsize=12)
plt.show()
```



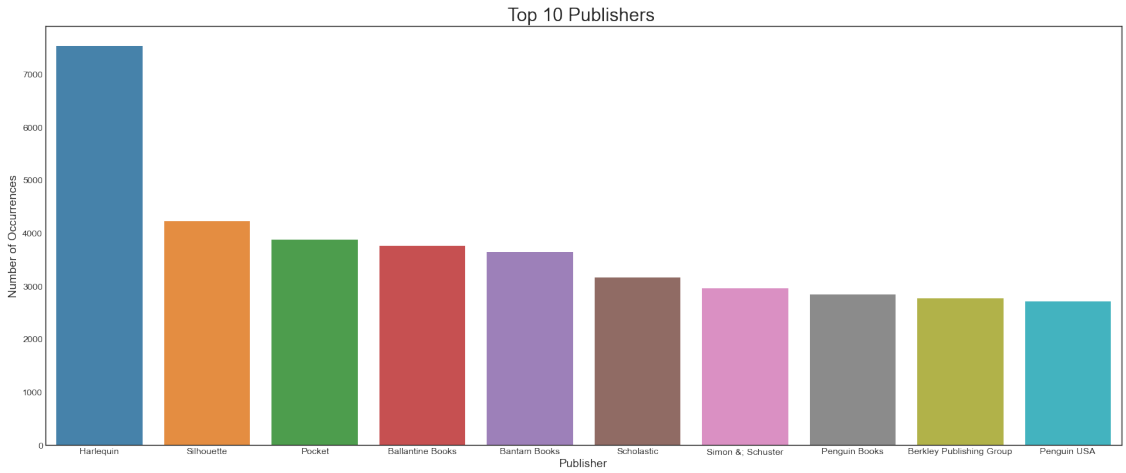
The chart above highlights the top 10 books preferred by readers throughout the years with "Selected Poems" taking the top spot, followed by "Little Women" and "Adventures of Huckleberry Finn"

[46]:
```
Top_Author  = books["Book_Author"].value_counts()
Top_Author = Top_Author[:10,]
plt.figure(figsize=(25,10))
sns.barplot(Top_Author.index, Top_Author.values, alpha=0.9)
plt.title('Top 10 Authors ')
plt.ylabel('Number of Occurrences', fontsize=15)
plt.xlabel('Book Author', fontsize=15)
plt.tick_params(labelsize=15)
plt.show()
```



The chart above highlights the top 10 authors readers have taken preference of throughout the years with Agatha Christie establishing herself as a top author on the chart, followed by William Shakespeare and Stephen King

[47]:
```
Top_Publisher  = books["Publisher"].value_counts()
Top_Publisher = Top_Publisher[:10,]
plt.figure(figsize=(25,10))
sns.barplot(Top_Publisher.index, Top_Publisher.values, alpha=0.9)
plt.title('Top 10 Publishers ')
plt.ylabel('Number of Occurrences', fontsize=15)
plt.xlabel('Publisher', fontsize=15)
plt.tick_params(labelsize=12)
plt.show()
```



The chart below highlights the top 10 publishers/publishing companies for all the books throughout the years with Harlequin taking the majority of the top spot, followed by Silhouette and Packet

[48]:
```
# checking for duplicate books
print(books[(books.duplicated(['Book_title', 'Book_Author'], keep=False))].
 ↪describe(include=[object]))
print(books[(books.duplicated(['Book_title', 'Book_Author'], keep='first'))].
 ↪describe(include=[object]))
print(books[(books.duplicated(['Book_title', 'Book_Author']))].Book_Author.
 ↪value_counts().head())
```

```
              ISBN                      Book_title      Book_Author  \
count        34936                           34936            34936
unique       34936                           15057             7577
top     0767905202  Adventures of Huckleberry Finn  Agatha Christie
freq             1                              20              249

               Publisher
count              34936
unique              2370
top     Ballantine Books
freq                1082
              ISBN                      Book_title   Book_Author  \
count        19524                           19524         19524
unique       19524                           15057          7577
top     1565075676  Adventures of Huckleberry Finn  Stephen King
freq             1                              19           183

               Publisher
count              19524
unique              1912
```

```
top     Ballantine Books
freq                 583
Stephen King         183
Agatha Christie      149
Dick Francis          82
William Shakespeare   69
Charles Dickens       66
Name: Book_Author, dtype: int64
```

[49]:
```
# it seems that the boos are duplicated due to differnet publishers over the␣
 ↪world. so we will use only one version of the books to avoid duplicates in␣
 ↪the recomender model her is an example for the book of "Life of Pi'"
books[books.Book_title=='Life of Pi']
```

[49]:
```
             ISBN  Book_title  Book_Author  Year_of_Publication  \
246    0151008116  Life of Pi  Yann Martel               2002.0
563    0156027321  Life of Pi  Yann Martel               2003.0
8745   1565117794  Life of Pi  Yann Martel               2003.0
62892  184195425X  Life of Pi  Yann Martel               2004.0

              Publisher
246            Harcourt
563       Harvest Books
8745   Highbridge Audio
62892    Pub Group West
```

It appears that the books are duplicated due to different publishers around the world. Hence we will use only one version of the books to avoid duplicates in the recomender model her is an example for the book of "Life of Pi"'

[50]:
```
#Eliminating the duplications of books
books = books.drop_duplicates(['Book_title', 'Book_Author'])
books.describe(include=[object,int])
```

[50]:
```
              ISBN      Book_title          Book_Author  Publisher
count       247212          247212               247212     247212
unique      247212          238512               100667      16224
top     0755109996  Selected Poems  William Shakespeare  Harlequin
freq             1              25                  490       7508
```

[51]:
```
#Checking for the duplications
print(books[(books.duplicated(['Book_title', 'Book_Author'], keep=False))].
 ↪describe(include=[object]))
print(books[(books.duplicated(['Book_title', 'Book_Author'], keep='first'))].
 ↪describe(include=[object]))
print(books[(books.duplicated(['Book_title', 'Book_Author']))].Book_Author.
 ↪value_counts().head())

books[books.Book_title=='Life of Pi']
```

```
       ISBN Book_title Book_Author Publisher
count     0          0           0         0
unique    0          0           0         0
top     NaN        NaN         NaN       NaN
freq    NaN        NaN         NaN       NaN
       ISBN Book_title Book_Author Publisher
count     0          0           0         0
unique    0          0           0         0
top     NaN        NaN         NaN       NaN
freq    NaN        NaN         NaN       NaN
Series([], Name: Book_Author, dtype: int64)
```

[51]:
```
            ISBN  Book_title  Book_Author  Year_of_Publication Publisher
246  0151008116  Life of Pi  Yann Martel               2002.0  Harcourt
```

Output appears to show the duplicates no longer exist and there is only one true cell of book with its title, author, and publisher

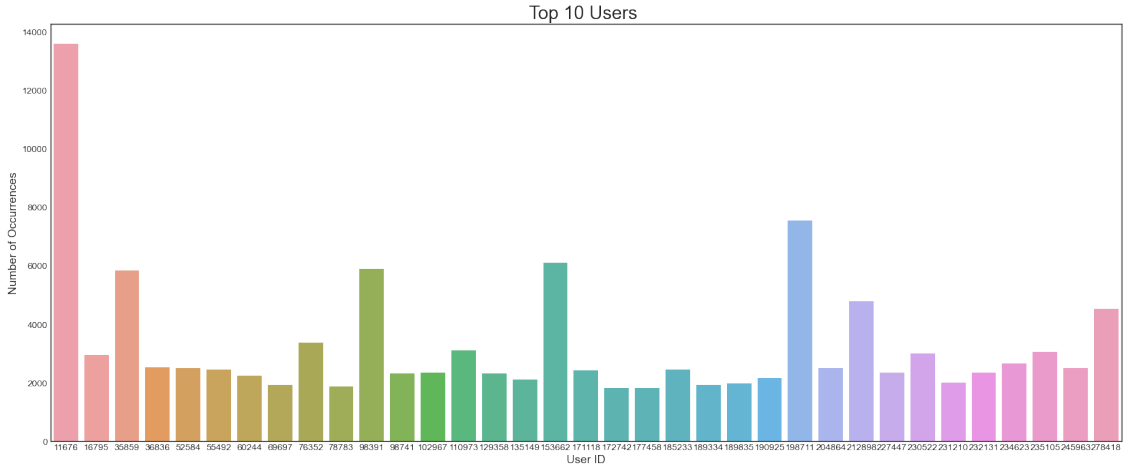
1.6 Exploring and Cleaning Books ratings set

[52]:
```
ratings.columns = ['User_id',"ISBN","Rating"]
ratings.head()
```

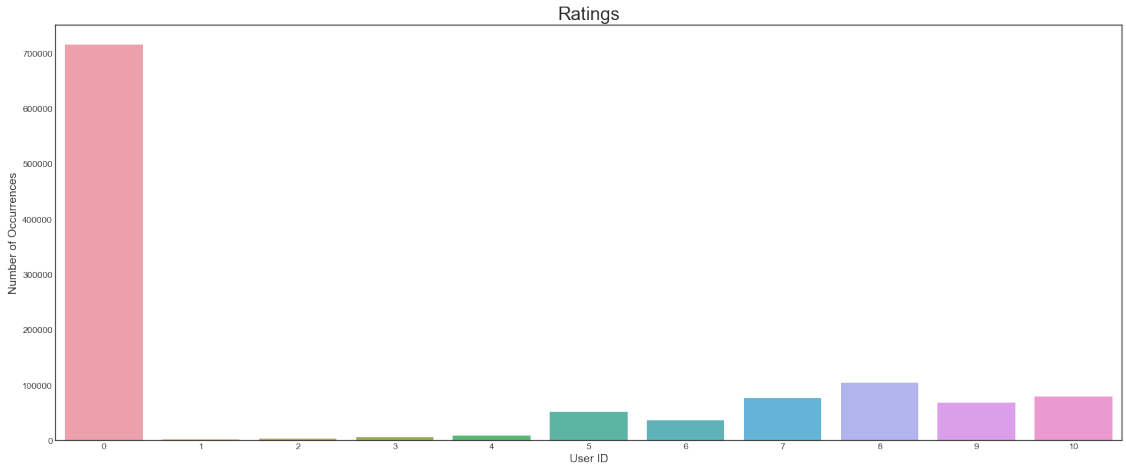
[52]:
```
   User_id        ISBN  Rating
0   276725  034545104X       0
1   276726  0155061224       5
2   276727  0446520802       0
3   276729  052165615X       3
4   276729  0521795028       6
```

print(ratings.dtypes) ratings.describe()

[53]:
```
Top_user  = ratings["User_id"].value_counts()
Top_user = Top_user[:35,]
plt.figure(figsize=(25,10))
sns.barplot(Top_user.index, Top_user.values, alpha=0.9)
plt.title('Top 10 Users ')
plt.ylabel('Number of Occurrences', fontsize=15)
plt.xlabel('User ID', fontsize=15)
plt.tick_params(labelsize=12)
plt.show()
```



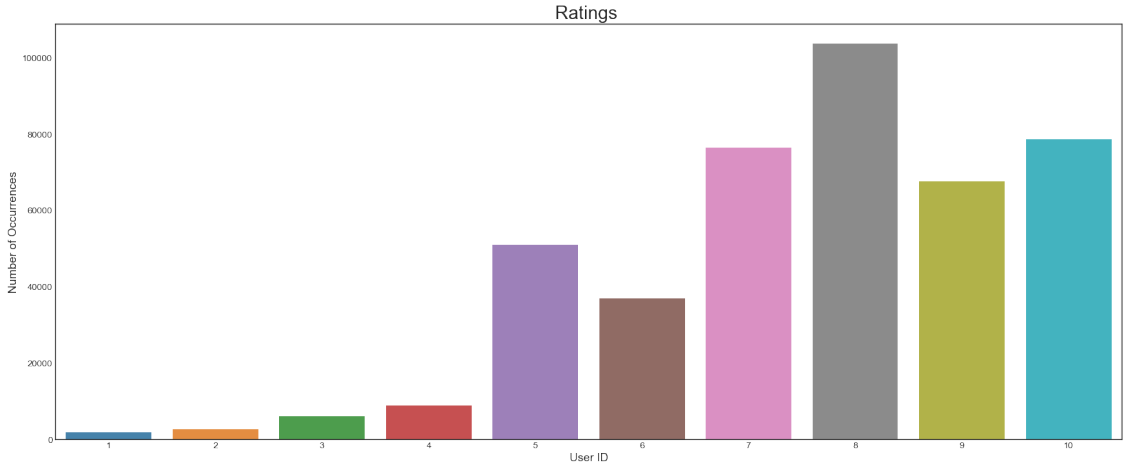
[54]:
```
#Ratings destribution
Rate_distrbt  = ratings["Rating"].value_counts()
plt.figure(figsize=(25,10))
sns.barplot(Rate_distrbt.index, Rate_distrbt.values, alpha=0.9)
plt.title('Ratings')
plt.ylabel('Number of Occurrences', fontsize=15)
plt.xlabel('User ID', fontsize=15)
plt.tick_params(labelsize=12)
plt.show()
```



Based on the describtion, these rating are implicit and explicit. The explicit ratings represented by 1–10 and implicit ratings represented by 0 will have to be segregated now. We will be using only explicit ratings for building our book recommendation system. Similarly, users are also segregated into those who rated explicitly and those whose implicit behavior was recorded.

[55]:
```
ratings = ratings[ratings.Rating != 0]
```

[56]:
```
#Ratings destribution
Rate_distrbt = ratings["Rating"].value_counts()
plt.figure(figsize=(25,10))
sns.barplot(Rate_distrbt.index, Rate_distrbt.values, alpha=0.9)
plt.title('Ratings ')
plt.ylabel('Number of Occurrences', fontsize=15)
plt.xlabel('User ID', fontsize=15)
plt.tick_params(labelsize=12)
plt.show()
```



1.7 Joining the datasets

Before Joining the datasets we will work with the data filtered for users with more than 4 ratings and the top 20% most frequnt rated books due to the computing poer

[57]:
```
user_ratings_threshold = 4

filter_users = ratings['User_id'].value_counts()
filter_users_list = filter_users[filter_users >= user_ratings_threshold].index.
 ↪to_list()

ratings2 = ratings[ratings['User_id'].isin(filter_users_list)]

print('Filter: users with at least %d ratings\nNumber of records: %d' %␣
 ↪(user_ratings_threshold, len(ratings)))
```

```
Filter: users with at least 4 ratings
Number of records: 433671
```

[58]:
```
book_ratings_threshold_perc = 0.20
```

```
book_ratings_threshold = len(ratings2['ISBN'].unique()) *␣
 ↪book_ratings_threshold_perc

filter_books_list = ratings2['ISBN'].value_counts().
 ↪head(int(book_ratings_threshold)).index.to_list()
ratings2 = ratings2[ratings2['ISBN'].isin(filter_books_list)]

print('Filter: top %d%% most frequently rated books\nNumber of records: %d' %␣
 ↪(book_ratings_threshold_perc*100, len(ratings2)))
```

```
Filter: top 20% most frequently rated books
Number of records: 208310
```

[59]:
```
books_ratings = ratings2.join(books.set_index('ISBN'), on='ISBN')
```

[60]:
```
books_ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 208310 entries, 16 to 1149775
Data columns (total 7 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   User_id              208310 non-null  int64
 1   ISBN                 208310 non-null  object
 2   Rating               208310 non-null  int64
 3   Book_title           170824 non-null  object
 4   Book_Author          170824 non-null  object
 5   Year_of_Publication  170824 non-null  float64
 6   Publisher            170824 non-null  object
dtypes: float64(1), int64(2), object(4)
memory usage: 12.7+ MB
```

[61]:
```
books_ratings.head()
```

[61]:

	User_id	ISBN	Rating	Book_title	Book_Author	Year_of_Publication	Publisher
16	276747	0060517794	9	Little Altars Everywhere	Rebecca Wells	2003.0	HarperTorch
19	276747	0671537458	9	Waiting to Exhale	Terry McMillan	1995.0	Pocket
20	276747	0679776818	8	Birdsong: A Novel of Love and War	Sebastian Faulks	1997.0	Vintage Books USA
33	276762	0380711524	5	See Jane Run	Joy Fielding	1992.0	Avon
81	276786	8478442588	6	El Elogio de La Sombra	Tanazaki	1998.0	Siruela

[62]:
```
books_ratings.isnull().sum()
```

[62]:
```
User_id                   0
ISBN                      0
Rating                    0
Book_title            37486
Book_Author           37486
Year_of_Publication   37486
Publisher             37486
dtype: int64
```

As per the above it seems that some ISBN that are rated has no books linked to them. this will be cleaned later

[63]:
```
books_ratings.dropna(subset=['Book_title'], inplace=True)
```

[64]:
```
books_ratings.isnull().sum()
```

[64]:
```
User_id               0
ISBN                  0
Rating                0
Book_title            0
Book_Author           0
Year_of_Publication   0
Publisher             0
dtype: int64
```

[65]:
```
#Join users data sets with books_ratings
ratings_wth_detls = books_ratings.join(users2.set_index('User_id'),␣
 ↪on='User_id')
```

[66]:
```
ratings_wth_detls.head()
```

[66]:

	User_id	ISBN	Rating	Book_title	Book_Author	Year_of_Publication	Publisher	Age	City	State	Country
16	276747	0060517794	9	Little Altars Everywhere	Rebecca Wells	2003.0	HarperTorch	25.0	iowa city	iowa	usa
19	276747	0671537458	9	Waiting to Exhale	Terry McMillan	1995.0	Pocket	25.0	iowa city	iowa	usa
20	276747	0679776818	8	Birdsong: A Novel of Love and War	Sebastian Faulks	1997.0	Vintage Books USA	25.0	iowa city	iowa	usa
33	276762	0380711524	5	See Jane Run	Joy Fielding	1992.0	Avon	25.0	duisburg		
81	276786	8478442588	6	El Elogio de La Sombra	Tanazaki	1998.0	Siruela	34.0	madrid		

```
 33   nordrhein-westfalen   germany
 81                madrid     spain
```

[67]:
```
ratings_wth_detls.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 170824 entries, 16 to 1149775
Data columns (total 11 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   User_id              170824 non-null  int64
 1   ISBN                 170824 non-null  object
 2   Rating               170824 non-null  int64
 3   Book_title           170824 non-null  object
 4   Book_Author          170824 non-null  object
 5   Year_of_Publication  170824 non-null  float64
 6   Publisher            170824 non-null  object
 7   Age                  170779 non-null  float64
 8   City                 170779 non-null  object
 9   State                170588 non-null  object
 10  Country              165774 non-null  object
dtypes: float64(2), int64(2), object(7)
memory usage: 15.6+ MB
```

[68]:
```
ratings_wth_detls.isnull().sum()
```

[68]:
```
User_id                   0
ISBN                      0
Rating                    0
Book_title                0
Book_Author               0
Year_of_Publication       0
Publisher                 0
Age                      45
City                     45
State                   236
Country                5050
dtype: int64
```

[69]:
```
# we will drop all the missing values from "age","city","State" and "Country"

ratings_wth_detls.dropna(subset=['Age','City','State','Country'], inplace=True)
```

[70]:
```
ratings_wth_detls.isnull().sum()
```

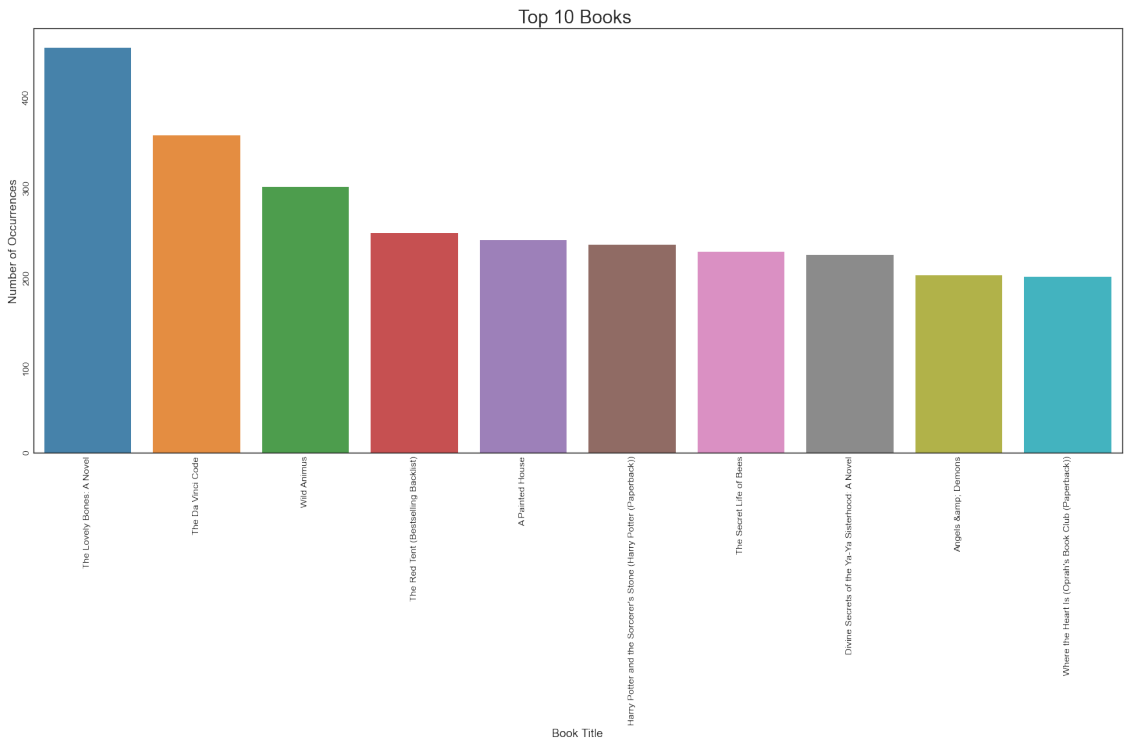
[70]:
```
User_id               0
ISBN                  0
Rating                0
Book_title            0
Book_Author           0
```

```
Year_of_Publication    0
Publisher              0
Age                    0
City                   0
State                  0
Country                0
dtype: int64
```

[71]:
```
ratings_wth_detls.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 165583 entries, 16 to 1149775
Data columns (total 11 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   User_id              165583 non-null  int64
 1   ISBN                 165583 non-null  object
 2   Rating               165583 non-null  int64
 3   Book_title           165583 non-null  object
 4   Book_Author          165583 non-null  object
 5   Year_of_Publication  165583 non-null  float64
 6   Publisher            165583 non-null  object
 7   Age                  165583 non-null  float64
 8   City                 165583 non-null  object
 9   State                165583 non-null  object
 10  Country              165583 non-null  object
dtypes: float64(2), int64(2), object(7)
memory usage: 15.2+ MB
```

[72]:
```
Top_books  = ratings_wth_detls["Book_title"].value_counts()
Top_books = Top_books[:10,]
plt.figure(figsize=(25,10))
sns.barplot(Top_books.index, Top_books.values, alpha=0.9)
plt.title('Top 10 Books ')
plt.ylabel('Number of Occurrences', fontsize=15)
plt.xlabel('Book Title', fontsize=15)
plt.tick_params(labelsize=12, rotation=90)
plt.show()
```



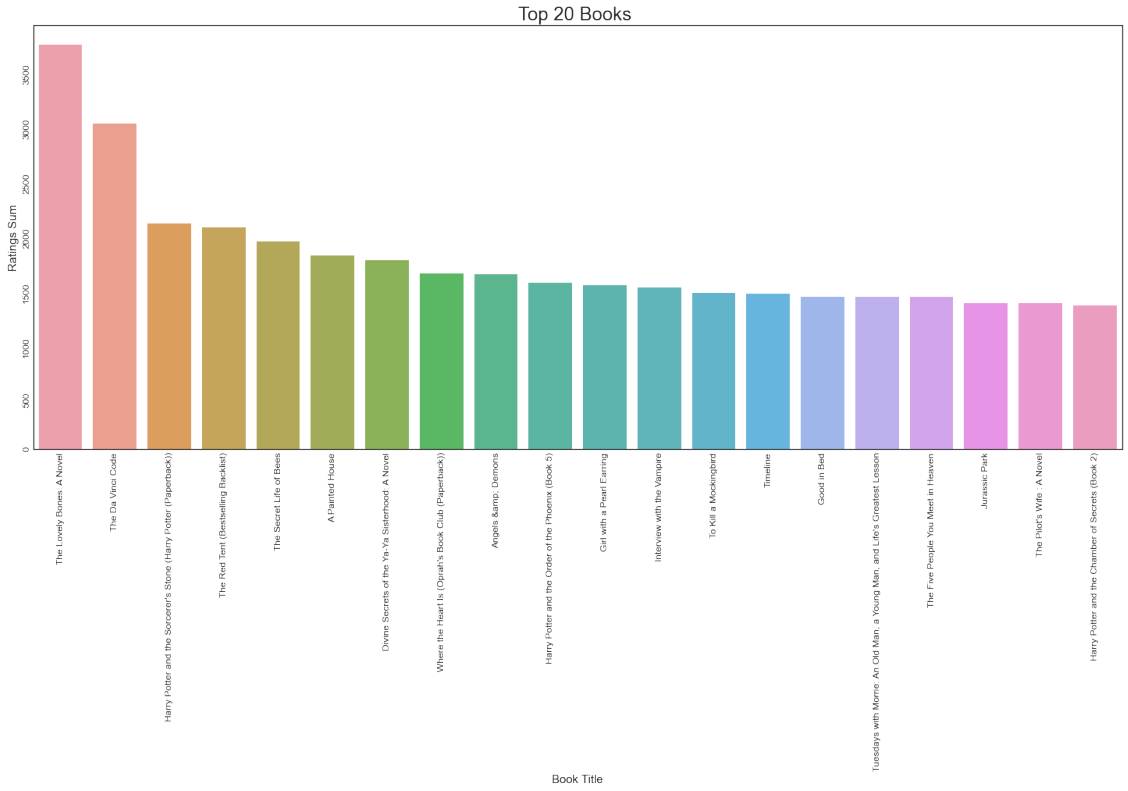
After further cleaning, the chart above lists the top 10 books enlisted by readers based on title throughout the years of the dataset. This highlights "The lovely Bones: A Novel" as the top read novel by readers; followed by "Wild Animus" and "The Da Vinci Code"

[73]:
```
BK_RT = ratings_wth_detls[['Book_title', 'Rating']]

BK_RT.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 165583 entries, 16 to 1149775
Data columns (total 2 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   Book_title  165583 non-null  object
 1   Rating      165583 non-null  int64
dtypes: int64(1), object(1)
memory usage: 3.8+ MB
```

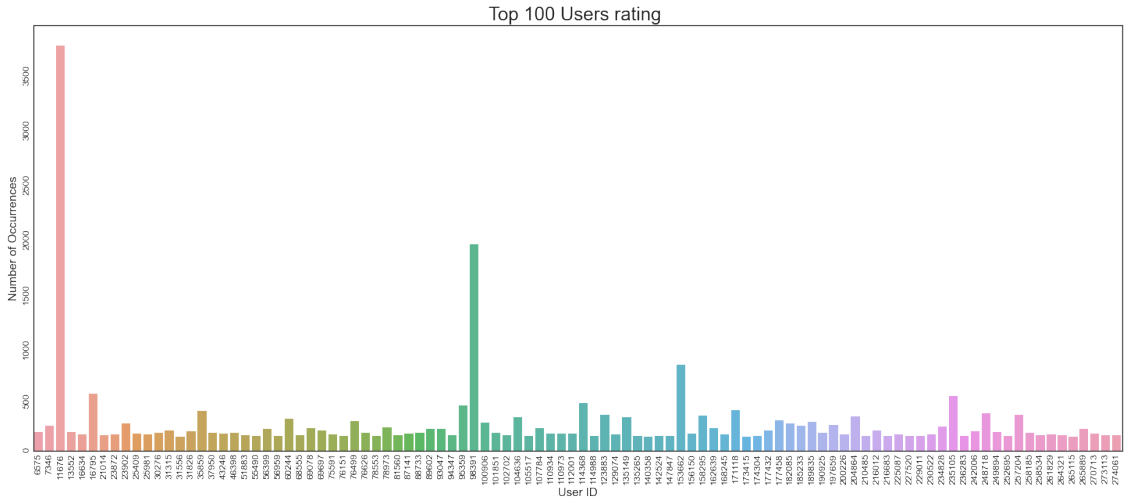
[74]:
```
#Top 20 Books is the Rating totals
BR_sum = BK_RT.groupby('Book_title').Rating.sum()
BR_sum = BR_sum.sort_values(ascending=False)[:20]
plt.figure(figsize=(25,10))
sns.barplot(BR_sum.index, BR_sum.values, alpha=0.9)
plt.title('Top 20 Books ')
plt.ylabel('Ratings Sum', fontsize=15)
```

```
plt.xlabel('Book Title', fontsize=15)
plt.tick_params(labelsize=12, rotation=90 )
plt.show()
```



The chart below highlights the Top 20 Books assessed and rated on readers' preference with "The lovely Bones: A Novel" having the highest rating amongst others; followed by "The Da Vinci Code" and "The Red Tent"

[75]:
```
Top_user_id  = ratings_wth_detls["User_id"].value_counts()
Top_user_id = Top_user_id[:100,].sort_values()
plt.figure(figsize=(25,10))
sns.barplot(Top_user_id.index, Top_user_id.values, alpha=0.9)
plt.title('Top 100 Users rating ')
plt.ylabel('Number of Occurrences', fontsize=15)
plt.xlabel('User ID', fontsize=15)
plt.tick_params(labelsize=12, rotation=90)
plt.show()
```



2 Recommendation systems

2.0.1 Collaborative Filtering Based Recommendation Systems

- We will compar sevral models and look at the RMSE to find the modle with lowest RMSE and try to tune the modle for better performance
- We useing SurPRISE libraray Simple Python RecommendatIon System Engine as it providea various ready-to-use prediction algorithms such as baseline algorithms, neighborhood methods, matrix factorization-based (SVD, PMF, SVD++, NMF), and many others. Also, various similarity measures (cosine, MSD, pearson...) are built-in.
- We will use only the User_ID, ISBN and Ratings as inputs for the algorithms to built the matrix and compare the RMSE
- The for Algorithms used in the notebook are (KNN_with_menas, KNN_Basic, SVD, NMF)

[76]:
```
ratings_wth_detls.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 165583 entries, 16 to 1149775
Data columns (total 11 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   User_id              165583 non-null  int64
 1   ISBN                 165583 non-null  object
 2   Rating               165583 non-null  int64
 3   Book_title           165583 non-null  object
 4   Book_Author          165583 non-null  object
 5   Year_of_Publication  165583 non-null  float64
 6   Publisher            165583 non-null  object
 7   Age                  165583 non-null  float64
```

```
 8   City                 165583 non-null  object
 9   State                165583 non-null  object
 10  Country              165583 non-null  object
dtypes: float64(2), int64(2), object(7)
memory usage: 15.2+ MB
```

[77]:
```
df = pd.DataFrame(ratings_wth_detls)
reader = Reader(rating_scale=(1, 10))
data = Dataset.load_from_df(ratings_wth_detls[['User_id', 'ISBN', 'Rating']],␣
 ↪reader)
```

[78]:
```
df1 = pd.DataFrame(df[['User_id', 'ISBN', 'Rating']])

df2 = pd.DataFrame(df1[df1.groupby('User_id')['User_id'].transform('size') > 3])
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 153461 entries, 133 to 1149746
Data columns (total 3 columns):
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   User_id  153461 non-null  int64
 1   ISBN     153461 non-null  object
 2   Rating   153461 non-null  int64
dtypes: int64(2), object(1)
memory usage: 4.7+ MB
```

[79]:
```
data2 = Dataset.load_from_df(df2[['User_id', 'ISBN', 'Rating']], reader)
```

2.0.2 KNN With Means model

[80]:
```
KNNWithMeansmodel = KNNWithMeans()

cv_results_knn= cross_validate(KNNWithMeansmodel, data,  measures=['RMSE'],␣
 ↪cv=5, verbose=True)
pd.DataFrame(cv_results_knn).mean()
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
```

```
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.7814  1.8019  1.8040  1.8055  1.8104  1.8006  0.0100
Fit time          20.17   16.54   17.60   12.34   13.18   15.97   2.88
Test time         2.23    2.68    1.81    1.74    1.94    2.08    0.34
```

[80]:
```
test_rmse     1.800634
fit_time     15.968263
test_time     2.083414
dtype: float64
```

2.0.3 KNN_Basic

[81]:
```
KNNBasicmodel = KNNBasic()
cv_results_knnBas= cross_validate(KNNBasicmodel, data,  measures=['RMSE'],␣
 ↪cv=5, verbose=True)
pd.DataFrame(cv_results_knnBas).mean()
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.9674  1.9631  1.9744  1.9742  1.9633  1.9685  0.0050
Fit time          10.12   12.32   9.00    8.06    7.33    9.37    1.75
Test time         2.35    1.78    1.43    1.49    1.50    1.71    0.34
```

[81]:
```
test_rmse    1.968482
fit_time     9.366542
test_time    1.712595
dtype: float64
```

2.0.4 SVD

[82]:
```
model_svd = SVD()
cv_results_svd = cross_validate(model_svd, data, cv=5)
pd.DataFrame(cv_results_svd).mean()
```

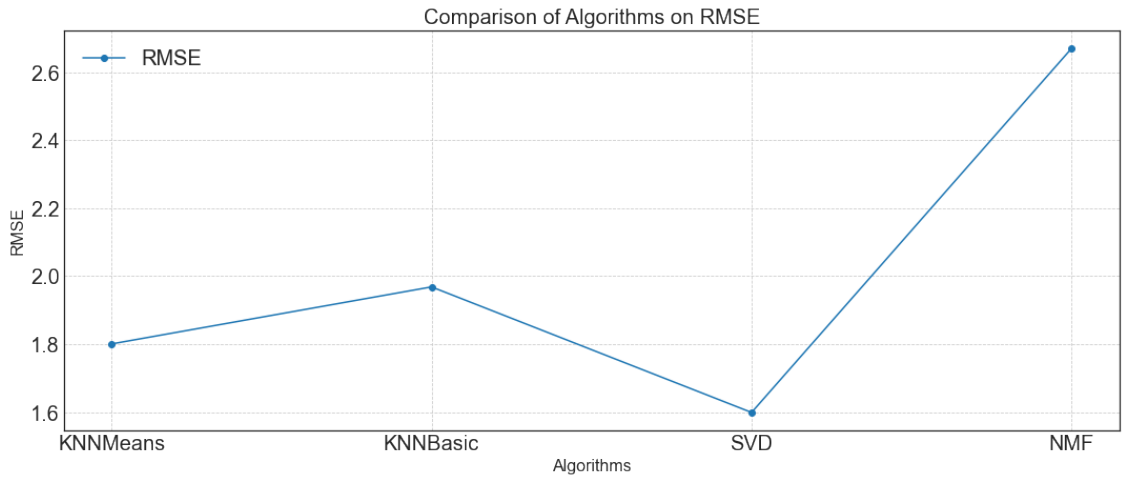
[82]:
```
test_rmse    1.598905
test_mae     1.235027
fit_time     8.056791
test_time    0.290340
dtype: float64
```

2.0.5 NMF

[83]:
```
model_nmf = NMF()
cv_results_nmf = cross_validate(model_nmf, data, cv=5)
pd.DataFrame(cv_results_nmf).mean()
```

[83]:
```
test_rmse     2.670657
test_mae      2.277006
fit_time     12.320202
test_time     0.290177
dtype: float64
```

[101]:
```
Algorithms = ['KNNMeans','KNNBasic','SVD', 'NMF']
#mannullt recode mean RMSE...since re-running takes too much time
rmse_results =[1.800634, 1.968482, 1.598905, 2.670657]

#rmse vs. algorithms
plt.figure(figsize=(18,7))
plt.title('Comparison of Algorithms on RMSE', loc='center', fontsize=20)
plt.plot(Algorithms, rmse_results, label='RMSE', marker='o')
plt.xlabel('Algorithms', fontsize=16)
plt.ylabel('RMSE', fontsize=16)
plt.grid(ls='dashed')
plt.legend()
plt.show()
```



The graph above highlights the four types of Algorithms that were assessed based on RMSE. It appears that SVD has the lowest error of the four and hence will be utilized for the validation of the test dataset

2.1 Tunning the SVD to find the best configuration of parameters.

n_factors - the number of factors. n_epochs - the number of iteration of the SGD procedure.

lr_all - the learning rate for all parameters.

reg_all - the regularization term for all parameters.

As a result, there was a minor improvement obtained with the tunning. $param_grid = {'n_factors' : [50, 60, 80,],' n_epochs' : [20, 30, 40],' lr_all' : [0.005, 0.008],' reg_all' : [0.1, 0.2, 0.4]}$

gs = GridSearchCV(SVD, $param_grid, measures = ['rmse',' mae'], cv = 3)gs.fit(data)$

print(gs.best$_score['rmse'])print(gs.best_params['rmse'])$

1.5991992941148412 'n$_factors' : 50,' n_epochs' : 20,' lr_all' : 0.008,' reg_all' : 0.2$

2.1.1 Testing and analysing the SVD Collaborative Filtering model through the train test and 10% test data

[85]:
```
import random
my_seed = 50
random.seed(my_seed)
np.random.seed(my_seed)
trainset, testset = train_test_split(data, test_size=0.10)

model = SVD(n_factors=50, n_epochs=20, lr_all=0.008, reg_all=0.2)
model.fit(trainset)
model.qi.shape
predictions = model.test(testset)
```

[86]:
```
# checking the matrix shape
print(model.qi.shape)


pd.DataFrame(model.qi).iloc[0].pow(2).sum()
model.qi /= np.linalg.norm(model.qi, ord=2, axis=1).reshape(-1, 1)
pd.DataFrame(model.qi).iloc[0].pow(2).sum()
```

```
(26283, 50)
```

[86]:
```
1.0
```

def display(df: pd.DataFrame): item$_to_rowidx_df = pd.DataFrame(list(item_to_rowidx.items()), columns = ['ISBN',' model.qirowidx'],).set_index('ISBN')return item_to_rowidx_df.head(5)$

Testing the mdoel with inputing some User_id and book ISBN to check the Rating predection

[87]:
```
model.predict('276762','3453092007')
```

[87]:
```
Prediction(uid='276762', iid='3453092007', r_ui=None, est=7.7299763796435474,
details={'was_impossible': False})
```

[88]:
```
model.predict('208492','0312990456')
```

[88]:
```
Prediction(uid='208492', iid='0312990456', r_ui=None, est=8.361346844619057,
details={'was_impossible': False})
```

[89]:
```
# Looking at the RMSE of the model

cv_results_svd1 = cross_validate(model, data, cv=5)
print(cv_results_svd1)
pd.DataFrame(cv_results_svd1).mean()
```

```
{'test_rmse': array([1.5726525 , 1.58102142, 1.59642619, 1.58764673,
1.58750512]), 'test_mae': array([1.22435064, 1.22616708, 1.23533612, 1.22912374,
1.23213248]), 'fit_time': (6.035475730895996, 5.556454181671143,
4.977792024612427, 5.3365478515625, 5.629119873046875), 'test_time':
(0.33211231231689453, 0.47233057022094727, 0.26831960678100586,
0.44010472297668457, 0.5125257968902588)}
```

[89]:
```
test_rmse    1.585050
test_mae     1.229422
fit_time     5.507078
test_time    0.405079
dtype: float64
```

Below joined the matrix with the orginal data set to calaculate the absolut error betwen predection and orginal rating

[90]:
```
df_pred = pd.DataFrame(predictions, columns=['User_id', 'ISBN',␣
 ↪'actual_rating', 'pred_rating', 'details'])
df_pred['impossible'] = df_pred['details'].apply(lambda x: x['was_impossible'])
df_pred['pred_rating_round'] = df_pred['pred_rating'].round()
df_pred['abs_err'] = abs(df_pred['pred_rating'] - df_pred['actual_rating'])
df_pred.drop(['details'], axis=1, inplace=True)
df_pred.sample(20)
```

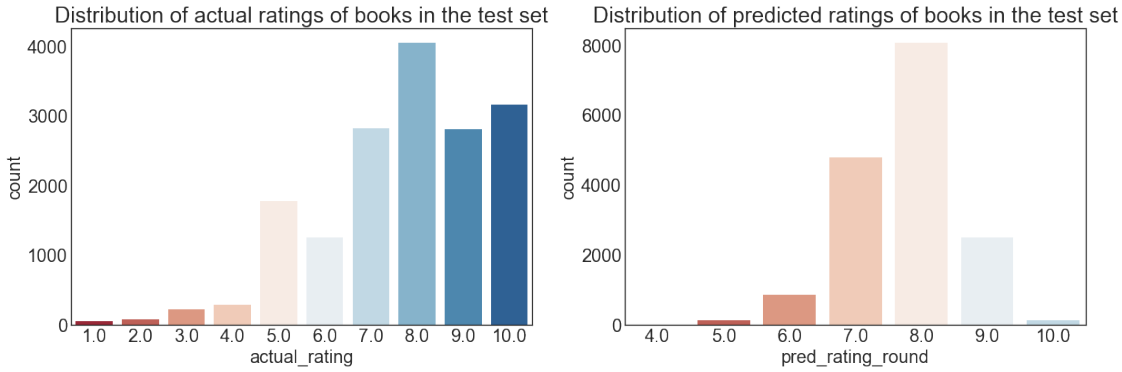
[90]:

	User_id	ISBN	actual_rating	pred_rating	impossible	\
1245	11676	0553574086	7.0	7.658422	False	
3658	49526	0312976275	7.0	8.015455	False	
9988	80453	0886776511	7.0	7.900020	False	
4978	221871	0060930535	7.0	8.049973	False	
3414	60080	0671016784	9.0	7.482408	False	
12673	251394	0553268449	8.0	7.967898	False	
7880	114200	0515118559	10.0	7.822826	False	
1050	209516	0375500510	10.0	9.219908	False	
1407	177432	0140177396	10.0	8.619778	False	
5182	2411	0971880107	3.0	4.585605	False	
7423	81121	0553212281	8.0	8.567692	False	
10385	235282	0345361571	8.0	8.092487	False	
15100	240484	0440222109	5.0	7.171879	False	

```
6578    252820  0913299545           8.0     6.718668       False
7531    257145  0671864416           5.0     6.957667       False
5004     91751  0553106651           5.0     7.405021       False
10923    16795  0440193613           7.0     7.442049       False
2727    123094  0894808249          10.0     8.938473       False
15422   132275  0553277472           8.0     7.633843       False
12332   139742  1853261912           7.0     7.828471       False

       pred_rating_round   abs_err
1245                 8.0  0.658422
3658                 8.0  1.015455
9988                 8.0  0.900020
4978                 8.0  1.049973
3414                 7.0  1.517592
12673                8.0  0.032102
7880                 8.0  2.177174
1050                 9.0  0.780092
1407                 9.0  1.380222
5182                 5.0  1.585605
7423                 9.0  0.567692
10385                8.0  0.092487
15100                7.0  2.171879
6578                 7.0  1.281332
7531                 7.0  1.957667
5004                 7.0  2.405021
10923                7.0  0.442049
2727                 9.0  1.061527
15422                8.0  0.366157
12332                8.0  0.828471
```

2.1.2 Comparison between the destribution of the actual ratings vs the predicted ratings

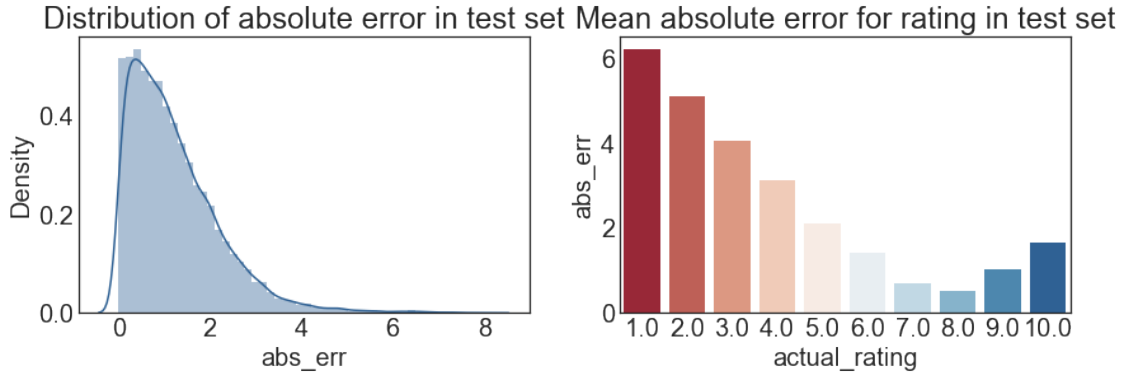
[91]:
```
palette = sns.color_palette("RdBu", 10)
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(20, 6))

sns.countplot(x='actual_rating', data=df_pred, palette=palette, ax=ax1)
ax1.set_title('Distribution of actual ratings of books in the test set')

sns.countplot(x='pred_rating_round', data=df_pred, palette=palette, ax=ax2)
ax2.set_title('Distribution of predicted ratings of books in the test set')

plt.show()
```



When comparing the actual and predicted values destribution of ratingsacross the datasets, the predicted ratings , it appears that the rating 8 has the higest counts over the 2 datasets but while the rest of the destribution between 5 and 10 is not identical show more users to rate 7, 9 and 10 wihile it is prdeicted at 8 by the model. This is offcoursethe reflection of the RMSE error of 1.5. These means the modle is not as perfect but it might give a an idea for the user about the books. probabbly if we could integrate the books genre to the details and the age of the users that would improve the model RMSE and give better insights to the analysis.

This is also shown in the graphs below fro the error destribution

[92]:
```
df_pred_err = df_pred.groupby('actual_rating')['abs_err'].mean().reset_index()

fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(14, 4))

sns.distplot(df_pred['abs_err'], color='#2f6194', ax=ax1)
ax1.set_title('Distribution of absolute error in test set')

sns.barplot(x='actual_rating', y='abs_err', data=df_pred_err, palette=palette,␣
↪ax=ax2)
ax2.set_title('Mean absolute error for rating in test set')

plt.show()
```



[93]:
```
df_ext = df.merge(df_pred[['ISBN', 'User_id','pred_rating', 'abs_err']],␣
↪on=['ISBN', 'User_id'], how='left')
df_ext.head(30)
```

[93]:

	User_id	ISBN	Rating	Book_title	Book_Author
0	276747	0060517794	9	Little Altars Everywhere	Rebecca Wells
1	276747	0671537458	9	Waiting to Exhale	Terry McMillan
2	276747	0679776818	8	Birdsong: A Novel of Love and War	Sebastian Faulks
3	276762	0380711524	5	See Jane Run	Joy Fielding
4	276786	8478442588	6	El Elogio de La Sombra	Tanazaki
5	276813	8423996565	8	La hija del CanÃbal	Rosa Montero
6	276813	8426449476	8	El Diaro De Bridget Jones	Helen Fielding
7	276813	8426449573	6	Bridget Jones:SobrevivirÃ	Helen Fielding
8	276822	0060096195	10	The Boy Next Door	Meggin Cabot
9	276822	0141310340	9	Skin and Other Stories (Now in Speak!)	Roald Dahl
10	276822	0142302198	10		
11	276822	0375821813	9		
12	276822	038076041X	10		
13	276822	0439401399	6		
14	276822	0689804458	8		
15	276822	0786817070	10		
16	276847	3404148576	8		
17	276847	3404921178	7		
18	276847	3423071516	10		
19	276847	3442413508	10		
20	276847	3442422035	9		
21	276847	3442435773	8		
22	276847	3442437717	7		
23	276847	3442441080	10		
24	276847	3442444020	8		
25	276847	3442446414	10		
26	276847	3442448530	7		
27	276847	3453137442	6		
28	276847	347354034X	7		
29	276847	3492231322	9		

10	Growing Wings	Laurel Winter
11	Hoot (Newbery Honor Book)	CARL HIAASEN
12	A Kid's Guide to How to Save the Planet (Camel...	Billy Goodman
13	The Contest	Gordon Korman
14	A String in the Harp	Nancy Bond
15	Artemis Fowl (Artemis Fowl, Book 1)	Eoin Colfer
16	Nordermoor	Arnaldur Indridason
17	Nur der Tod ist ohne Makel.	Ann Granger
18	Der Kleine Hobbit	J. R. R. Tolkien
19	Auf Ehre und Gewissen. Roman.	Elizabeth George
20	Keiner werfe den ersten Stein. Roman.	Elizabeth George
21	Denn keiner ist ohne Schuld.	Elizabeth George
22	Asche zu Asche.	Elizabeth George
23	Im Angesicht des Feindes.	Elizabeth George
24	Denn sie betrÃ?Âgt man nicht.	Elizabeth George
25	Mit dem KÃ?Âhlschrank durch Irland.	Tony Hawks
26	Die HirnkÃ?Âűnigin.	Thea Dorn
27	Der Goldene Kompass / The Golden Compass	Philip Pullman
28	Die Welle	Rhue
29	Der UnterhÃ?Âdndler.	Frederick Forsyth

	Year_of_Publication	Publisher
0	2003.0	HarperTorch
1	1995.0	Pocket
2	1997.0	Vintage Books USA
3	1992.0	Avon
4	1998.0	Siruela
5	1998.0	Espasa Calpe Mexicana, S.A.
6	1996.0	Lumen Espana
7	2000.0	Downtown Book Center Inc
8	2002.0	Avon Trade
9	2002.0	Puffin Books
10	2002.0	Puffin Books
11	2002.0	Knopf Books for Young Readers
12	1990.0	Harpercollins Juvenile Books
13	2002.0	Scholastic
14	1996.0	Aladdin
15	2002.0	Miramax Kids
16	2003.0	LÃ?Âbbe
17	2002.0	LÃ?Âbbe
18	2002.0	Distribooks
19	1992.0	Goldmann
20	1993.0	Goldmann
21	1996.0	Goldmann
22	1997.0	Goldmann
23	1998.0	Goldmann
24	1999.0	Goldmann

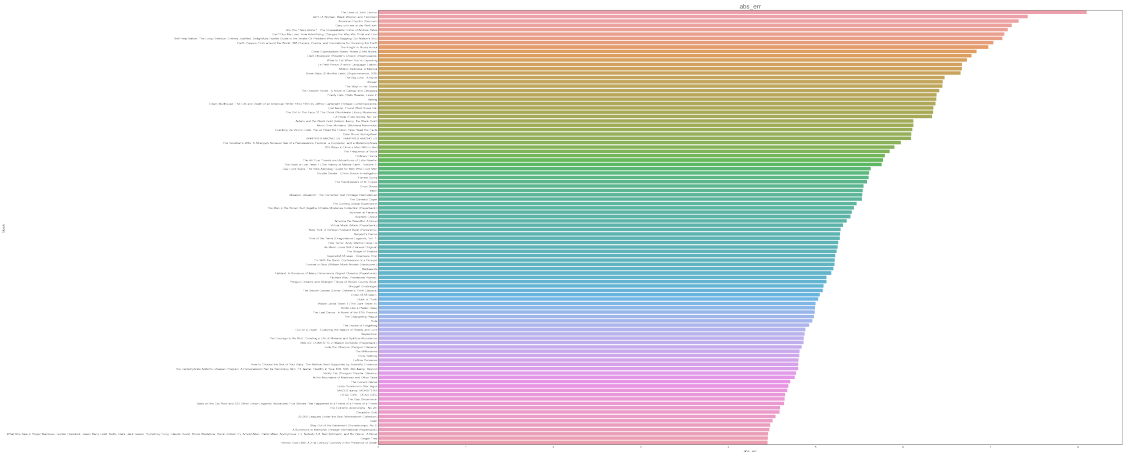
```
25              2000.0                                          Goldmann
26              2001.0                                          Goldmann
27              2002.0                                       Distribooks
28              1998.0  Ullstein-Taschenbuch-Verlag, Zweigniederlassun...
29              2000.0                                           Piper
```

	Age	City	State	Country	pred_rating	abs_err
0	25.0	iowa city	iowa	usa	NaN	NaN
1	25.0	iowa city	iowa	usa	NaN	NaN
2	25.0	iowa city	iowa	usa	NaN	NaN
3	25.0	duisburg	nordrhein-westfalen	germany	NaN	NaN
4	34.0	madrid	madrid	spain	NaN	NaN
5	29.0	sitges	barcelona	spain	NaN	NaN
6	29.0	sitges	barcelona	spain	NaN	NaN
7	29.0	sitges	barcelona	spain	NaN	NaN
8	11.0	calgary	alberta	canada	8.651289	1.348711
9	11.0	calgary	alberta	canada	8.016534	0.983466
10	11.0	calgary	alberta	canada	NaN	NaN
11	11.0	calgary	alberta	canada	NaN	NaN
12	11.0	calgary	alberta	canada	NaN	NaN
13	11.0	calgary	alberta	canada	NaN	NaN
14	11.0	calgary	alberta	canada	8.317067	0.317067
15	11.0	calgary	alberta	canada	NaN	NaN
16	27.0	köln	nordrhein-westfalen	germany	8.303936	0.303936
17	27.0	köln	nordrhein-westfalen	germany	NaN	NaN
18	27.0	köln	nordrhein-westfalen	germany	NaN	NaN
19	27.0	köln	nordrhein-westfalen	germany	NaN	NaN
20	27.0	köln	nordrhein-westfalen	germany	NaN	NaN
21	27.0	köln	nordrhein-westfalen	germany	8.038623	0.038623
22	27.0	köln	nordrhein-westfalen	germany	NaN	NaN
23	27.0	köln	nordrhein-westfalen	germany	NaN	NaN
24	27.0	köln	nordrhein-westfalen	germany	NaN	NaN
25	27.0	köln	nordrhein-westfalen	germany	8.668682	1.331318
26	27.0	köln	nordrhein-westfalen	germany	NaN	NaN
27	27.0	köln	nordrhein-westfalen	germany	NaN	NaN
28	27.0	köln	nordrhein-westfalen	germany	NaN	NaN
29	27.0	köln	nordrhein-westfalen	germany	NaN	NaN

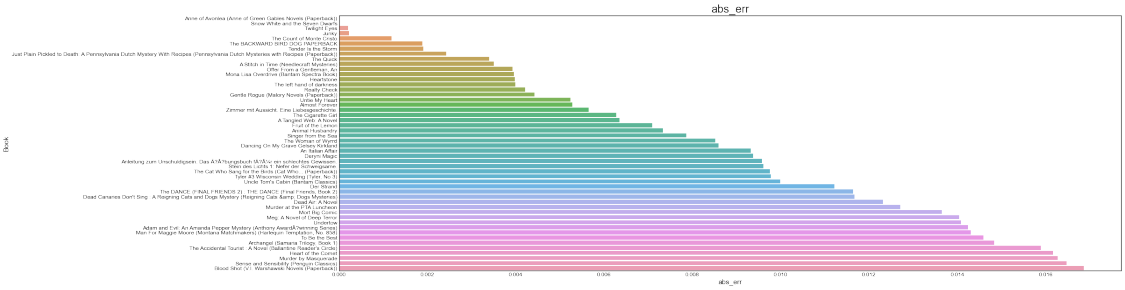
2.1.3 Some Ploting of errors against users and books for explortation

[94]:
```
# 100 Books with the highest mean error in prediction
df_Cou_err = df_ext.groupby('Book_title')['abs_err'].mean().reset_index()
df_Cou_err  = df_Cou_err.sort_values('abs_err', ascending=False)[:100]
plt.figure(figsize=(50,30))
sns.barplot(df_Cou_err['abs_err'], df_Cou_err['Book_title'], alpha=0.9)
plt.title('abs_err')
```

```
plt.ylabel('Book', fontsize=15)
plt.xlabel('abs_err', fontsize=15)
plt.tick_params(labelsize=12, rotation=0 )
plt.show()
```

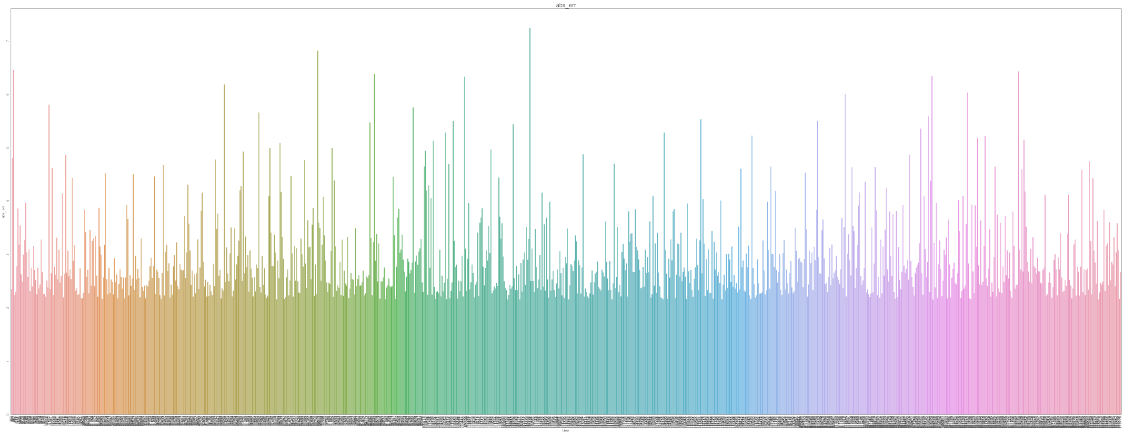


[95]:
```
# 50 Books with the loest mean error in prediction
df_Cou_err = df_ext.groupby('Book_title')['abs_err'].mean().reset_index()
df_Cou_err  = df_Cou_err.sort_values('abs_err', ascending=True)[:50]
plt.figure(figsize=(30,10))
sns.barplot(df_Cou_err['abs_err'], df_Cou_err['Book_title'], alpha=0.9)
plt.title('abs_err')
plt.ylabel('Book', fontsize=15)
plt.xlabel('abs_err', fontsize=15)
plt.tick_params(labelsize=12, rotation=0 )
plt.show()
```



[96]:
```
# 100 Users with the highest mean error in prediction
df_Cou_err = df_ext.groupby('User_id')['abs_err'].mean().reset_index()
df_Cou_err  = df_Cou_err.sort_values('abs_err', ascending=False)[:1000]
plt.figure(figsize=(80,30))
sns.barplot(df_Cou_err['User_id'],df_Cou_err['abs_err'],  alpha=0.9)
```

```
plt.title('abs_err')
plt.ylabel('abs_err', fontsize=15)
plt.xlabel('User', fontsize=15)
plt.tick_params(labelsize=15, rotation=90 )
plt.show()
```



2.2 Analysis and validation for User_ID 227250

We selected a random user to check the recmonede books for him against what was recommended actually by him

[104]:
```
# Building the user profile for ratings more than 8
selected_user_id = 43246
df_user = df_ext[df_ext['User_id']==selected_user_id]

df_user[(df_user['pred_rating'].isna())&(df_user['Rating']>=9)].sample(20)
```

[104]:

	User_id	ISBN	Rating	\
30172	43246	0425103633	10	
30183	43246	0446605239	9	
30119	43246	015600710X	9	
30215	43246	0684854759	9	
30199	43246	0553578693	9	
30150	43246	0380010038	10	
30167	43246	0399128298	9	
30126	43246	0312979770	10	
30233	43246	1400060052	9	
30188	43246	0449210677	9	
30186	43246	0446610038	9	
30240	43246	1577310047	10	
30095	43246	0060391901	9	
30156	43246	0380791978	9	
30133	43246	0316969443	9	

```
30207    43246  0671008978      10
30244    43246  1853261912       9
30097    43246  0060801263       9
30098    43246  0060926317       9
30241    43246  1592400876       9

                                                Book_title  \
30172  Murder in Mesopotamia (Hercule Poirot Mysterie...
30183                                        The Notebook
30119                  Strange Fits of Passion: A Novel
30215                      FLOATING IN MY MOTHER'S PALM
30199                                 The Next Accident
30150                                          The Cay
30167   As a Man Thinketh (Family Inspirational Library)
30126  The Seven Dials Mystery (St. Martin's Minotaur...
30233                           Lucia, Lucia : A Novel
30188                                Resurrection Row
30186                             1st to Die: A Novel
30240                  Papa's Angels: A Christmas Story
30095  Mastering the Zone: The Next Step in Achieving...
30156                                The Family Tree
30133                     Suzanne's Diary for Nicholas
30207                                        Timepiece
30244             Mrs. Dalloway (Wordsworth Collection)
30097                           Tree Grows In Brooklyn
30098                        Mutant Message Down Under
30241  Eats, Shoots &amp;amp; Leaves: The Zero Tolerance ...

              Book_Author  Year_of_Publication                   Publisher  \
30172       Agatha Christie               2003.0     Berkley Publishing Group
30183       Nicholas Sparks               1998.0                 Warner Books
30119          Anita Shreve               1999.0                Harvest Books
30215           Ursula Hegi               1998.0                   Touchstone
30199          LISA GARDNER               2002.0                       Bantam
30150       Theodore Taylor               1995.0                  Avon Books
30167           James Allen               1955.0      Putnam Publishing Group
30126       Agatha Christie               2001.0      St. Martin's Paperbacks
30233      ADRIANA TRIGIANI               2003.0                 Random House
30188            Anne Perry               1989.0                Fawcett Books
30186       James Patterson               2002.0                Warner Vision
30240  Collin Wilcox Paxton               1996.0            New World Library
30095           Barry Sears               1996.0                   ReganBooks
30156       Sheri S. Tepper               1998.0                          Eos
30133       James Patterson               2001.0                Little, Brown
30207    Richard Paul Evans               1997.0                       Pocket
30244         Virginia Woolf               1999.0  NTC/Contemporary Publishing
30097           Betty Smith               1988.0      Harpercollins Publisher
```

```
30098          Marlo Morgan               1995.0                    Perennial
30241           Lynne Truss               2004.0                Gotham Books

        Age     City     State  Country  pred_rating  abs_err
30172  36.0  toronto   ontario   canada          NaN      NaN
30183  36.0  toronto   ontario   canada          NaN      NaN
30119  36.0  toronto   ontario   canada          NaN      NaN
30215  36.0  toronto   ontario   canada          NaN      NaN
30199  36.0  toronto   ontario   canada          NaN      NaN
30150  36.0  toronto   ontario   canada          NaN      NaN
30167  36.0  toronto   ontario   canada          NaN      NaN
30126  36.0  toronto   ontario   canada          NaN      NaN
30233  36.0  toronto   ontario   canada          NaN      NaN
30188  36.0  toronto   ontario   canada          NaN      NaN
30186  36.0  toronto   ontario   canada          NaN      NaN
30240  36.0  toronto   ontario   canada          NaN      NaN
30095  36.0  toronto   ontario   canada          NaN      NaN
30156  36.0  toronto   ontario   canada          NaN      NaN
30133  36.0  toronto   ontario   canada          NaN      NaN
30207  36.0  toronto   ontario   canada          NaN      NaN
30244  36.0  toronto   ontario   canada          NaN      NaN
30097  36.0  toronto   ontario   canada          NaN      NaN
30098  36.0  toronto   ontario   canada          NaN      NaN
30241  36.0  toronto   ontario   canada          NaN      NaN
```

[105]:
```
df_user[df_user['pred_rating'].notna()].sort_values('pred_rating',␣
↪ascending=False).head(12)
```

[105]:
```
       User_id        ISBN  Rating  \
30217    43246  0740704818      10
30160    43246  0385484518      10
30239    43246  1573228737       8
30115    43246  0146000552      10
30154    43246  0380773805       6
30166    43246  0393314367       8
30242    43246  1843330202       8
30116    43246  0151002630       8
30146    43246  0375500316       9
30153    43246  0380723085       8
30123    43246  0312305060       7
30136    43246  0345435168       8

                                            Book_title  \
30217                                 The Blue Day Book
30160  Tuesdays with Morrie: An Old Man, a Young Man,...
30239                                          Affinity
30115  " Lamb to the Slaughter and Other Stories (Pen...
30154  The Class Menagerie (Jane Jeffry Mysteries (Pa...
```

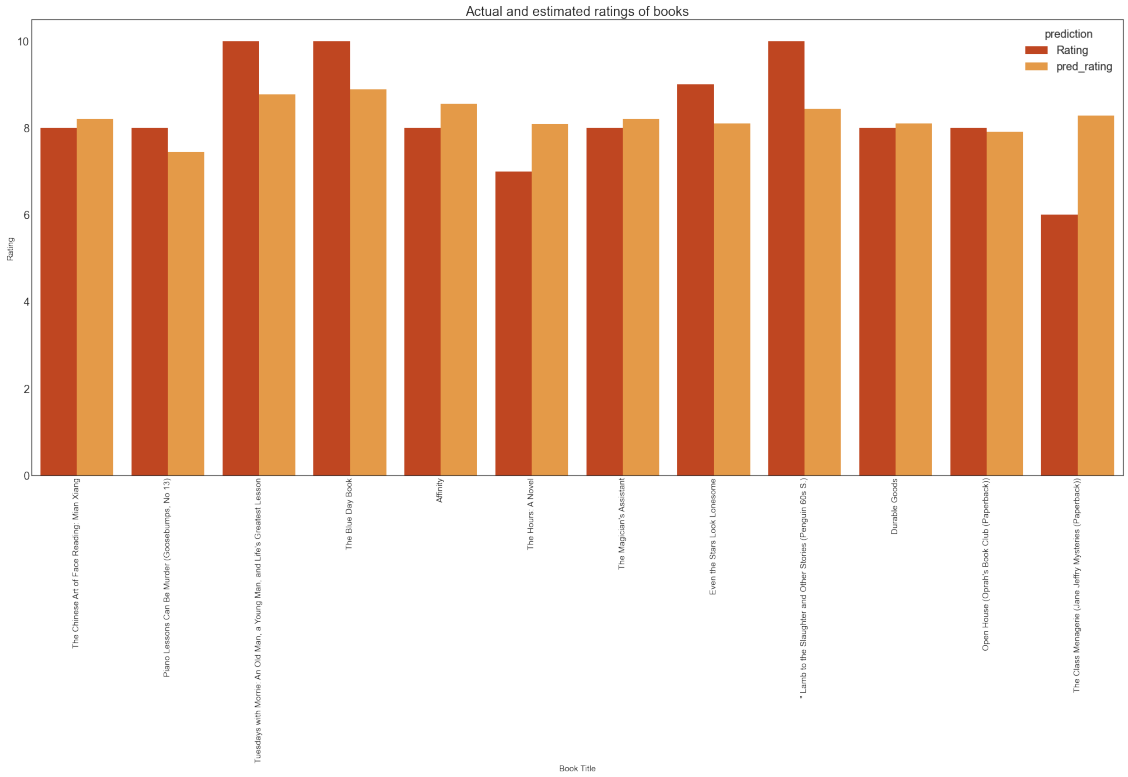
```
30166  Now All We Need Is a Title: Famous Book Titles...
30242        The Chinese Art of Face Reading: Mian Xiang
30116                             The Magician's Assistant
30146                        Even the Stars Look Lonesome
30153                                     Durable Goods
30123                                  The Hours: A Novel
30136          Open House (Oprah's Book Club (Paperback))

                Book_Author  Year_of_Publication                  Publisher  \
30217  Bradley Trevor Greive               2000.0  Andrews McMeel Publishing
30160           MITCH ALBOM               1997.0                 Doubleday
30239          Sarah Waters               2002.0           Riverhead Books
30115            Roald Dahl               1995.0         Penguin Books Ltd
30154        Jill Churchill               1999.0                      Avon
30166         Andre Bernard               1996.0   W. W. Norton &; Company
30242   Hai Lee Yang Henning               2002.0                 Vega Books
30116          Ann Patchett               1997.0                   Harcourt
30146          Maya Angelou               1997.0              Random House
30153        Elizabeth Berg               1994.0  Perennial (HarperCollins)
30123     Michael Cunningham              2002.0                    Picador
30136          Elizabeth Berg              2001.0          Ballantine Books

        Age     City     State  Country  pred_rating   abs_err
30217  36.0  toronto   ontario   canada     8.881208  1.118792
30160  36.0  toronto   ontario   canada     8.769555  1.230445
30239  36.0  toronto   ontario   canada     8.551844  0.551844
30115  36.0  toronto   ontario   canada     8.439233  1.560767
30154  36.0  toronto   ontario   canada     8.287444  2.287444
30166  36.0  toronto   ontario   canada     8.267001  0.267001
30242  36.0  toronto   ontario   canada     8.210923  0.210923
30116  36.0  toronto   ontario   canada     8.199253  0.199253
30146  36.0  toronto   ontario   canada     8.107267  0.892733
30153  36.0  toronto   ontario   canada     8.105218  0.105218
30123  36.0  toronto   ontario   canada     8.092818  1.092818
30136  36.0  toronto   ontario   canada     7.903755  0.096245
```

[106]:
```
df_user[df_user['pred_rating'].notna()].sort_values('Rating', ascending=False).
↪head(12)
```

[106]:
```
       User_id        ISBN  Rating  \
30115    43246  0146000552      10
30160    43246  0385484518      10
30217    43246  0740704818      10
30146    43246  0375500316       9
30116    43246  0151002630       8
30136    43246  0345435168       8
30153    43246  0380723085       8
30166    43246  0393314367       8
```

```
30202    43246  0590494481       8
30239    43246  1573228737       8
30242    43246  1843330202       8
30123    43246  0312305060       7

                                          Book_title  \
30115  " Lamb to the Slaughter and Other Stories (Pen...
30160  Tuesdays with Morrie: An Old Man, a Young Man,...
30217                               The Blue Day Book
30146                    Even the Stars Look Lonesome
30116                        The Magician's Assistant
30136       Open House (Oprah's Book Club (Paperback))
30153                                 Durable Goods
30166  Now All We Need Is a Title: Famous Book Titles...
30202    Piano Lessons Can Be Murder (Goosebumps, No 13)
30239                                       Affinity
30242       The Chinese Art of Face Reading: Mian Xiang
30123                               The Hours: A Novel

               Book_Author  Year_of_Publication                 Publisher  \
30115            Roald Dahl               1995.0         Penguin Books Ltd
30160          MITCH ALBOM                1997.0                 Doubleday
30217  Bradley Trevor Greive              2000.0  Andrews McMeel Publishing
30146          Maya Angelou               1997.0              Random House
30116          Ann Patchett               1997.0                   Harcourt
30136        Elizabeth Berg               2001.0          Ballantine Books
30153        Elizabeth Berg               1994.0  Perennial (HarperCollins)
30166         Andre Bernard               1996.0    W. W. Norton &; Company
30202           R. L. Stine               1995.0                 Scholastic
30239          Sarah Waters               2002.0            Riverhead Books
30242   Hai Lee Yang Henning              2002.0                 Vega Books
30123     Michael Cunningham              2002.0                    Picador

        Age     City     State  Country  pred_rating   abs_err
30115  36.0  toronto   ontario   canada     8.439233  1.560767
30160  36.0  toronto   ontario   canada     8.769555  1.230445
30217  36.0  toronto   ontario   canada     8.881208  1.118792
30146  36.0  toronto   ontario   canada     8.107267  0.892733
30116  36.0  toronto   ontario   canada     8.199253  0.199253
30136  36.0  toronto   ontario   canada     7.903755  0.096245
30153  36.0  toronto   ontario   canada     8.105218  0.105218
30166  36.0  toronto   ontario   canada     8.267001  0.267001
30202  36.0  toronto   ontario   canada     7.443649  0.556351
30239  36.0  toronto   ontario   canada     8.551844  0.551844
30242  36.0  toronto   ontario   canada     8.210923  0.210923
30123  36.0  toronto   ontario   canada     8.092818  1.092818
```

[108]:
```
df_pred_sample = df_user[df_user['pred_rating'].notna()].sample(12)
df_pred_sample = pd.melt(df_pred_sample[['Book_title', 'Rating',␣
 ↪'pred_rating']], id_vars='Book_title', var_name='prediction',␣
 ↪value_name='rating')

fig, ax = plt.subplots(figsize=(35, 15))

sns.barplot(x='Book_title', y='rating', hue='prediction', data=df_pred_sample,␣
 ↪palette=sns.color_palette("turbo_r"))
ax.set_title('Actual and estimated ratings of books')
plt.ylabel('Rating', fontsize=15)
plt.xlabel('Book Title', fontsize=15)
plt.xticks(rotation=90, fontsize=15)

plt.show()
```



The chart below highlights the actual and predicted ratings of the books. There is error between the rating of the books, However the result of the top 10 recomended books are similar for the user 10 books rated

[109]:
```
#Extraction of the model matrix with the data
df_cle = pd.DataFrame(df_ext[['User_id','ISBN','Book_title','pred_rating']])

df_cle.to_csv('Books_clean.csv')
```

2.3 Deployment

The model does not hold much of its use of interactivity considering the lack of accuracy and latency. However, it will provide a plain demonstration for the interested parties. For instance, this model could be used to aid brick and mortar bookstores (e.g. Chapters and Indigo) and digital libraries (e.g. Audible) in recommending books/novels to a specific demographic with specific preferences towards authors and genres. The BookCrossing dataset collected is quite sparse and we believe requires more upkeeping as it was gathered on a span of only four weeks. With people's preferences ever evolving when it comes to genre type and implicit/explicit ratings, it is significant to update every two to three years. This involves running as many backend instances as possible in order to finetune scalability. With the utilization of Python, the Dash app developed would be the appropriate deployment method in gathering new data. Our app is also created to cater to the marketing experts and analysts who would like to use it for their analysis and insight of topics as such as of the dataset.

2.4 Recomendation App

https://bookrecommender1.herokuapp.com/

Recommendation App is built to serve the marketing team to do life anallysis on the recmended books for the users. this can be deployed at a bookstore chain with users who has registed profiles or bookstore advisors who can filter the books and recomend similar books for users who has the same taste of registered users

2.4.1 GitHub link

https://github.com/tamerhanna/Book-Crossing-.git