Assignment

Course Code	CSC402A
Course Name	Data Mining
Programme	B.Tech
Department	CSE
Faculty	FET

Name of the Student	Satyajit Ghana
Reg. No.	17ETCS002159
Semester/Year	07/2020
Course Leader(s)	Prof. Mohan Kumar

Declaration Sheet

Student Name	Satyajit Ghana		
Reg. No	17ETCS002159		
Programme	B.Tech	Semester/Year	07/2020
Course Code	CSC402A		
Course Title	Data Mining		
Course Date		to	
Course Leader	Prof. Mohan Kumar		

Declaration

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Signature of the Student		Date	
Submission date stamp (by Examination & Assessment Section)			

Signature of the Course Leader and date	Signature of the Reviewer and date

Contents

List of Figures

1 Question 1

Solution to Question No. 1 Part A

This contains a brief summary of the data, and its preprocessing, refer to the Jupyter Notebook output at the end of this for a complete study of data.

1.1 Data Cleaning: Redundant and Inconsistent Data

Column	Mean	Std	Min	Max	Skewness	Kurtosis
age	36.23	10.41	5	100	0.83	1.34
book_rating	2.83	3.85	0	10	0.75	-1.21

1.1.1 Inconsistent Data

Figure 1-1 Book Ratings, before and after removing 0 ratings

Since 0 rated books done make sense they were removed, after removing our skewness and kurtosis values have changed a lot.

Column	Mean	Std	Min	Max	Skewness	Kurtosis
age	36.23	10.36	5	100	0.85	1.64
book_rating	2.83	3.85	1	10	-0.66	-0.12

1.1.2 Univariate Analysis

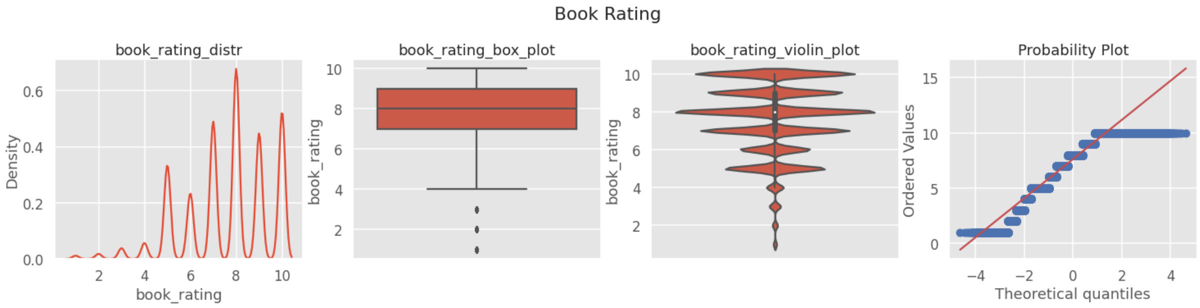


Figure 1-2 Book Rating Univariate Analysis

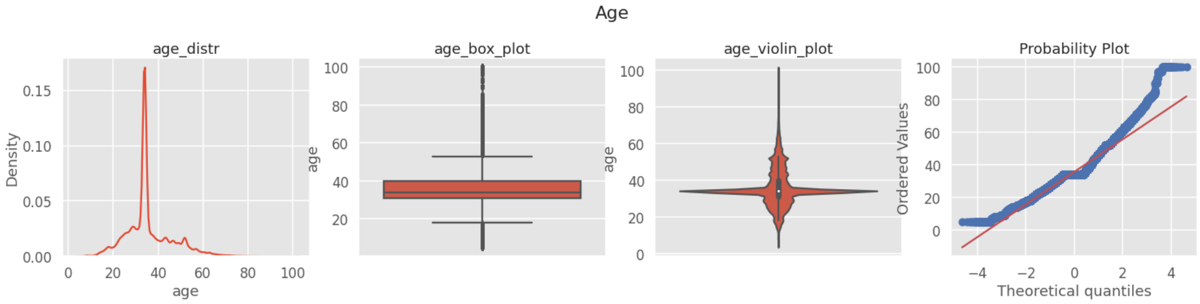


Figure 1-3 Age univariate analysis

1.2 Data Cleaning: Missing Values and Outliers

Refer Jupyter Notebook for Cleaning up Missing Values

1.2.1 Outlier Analysis

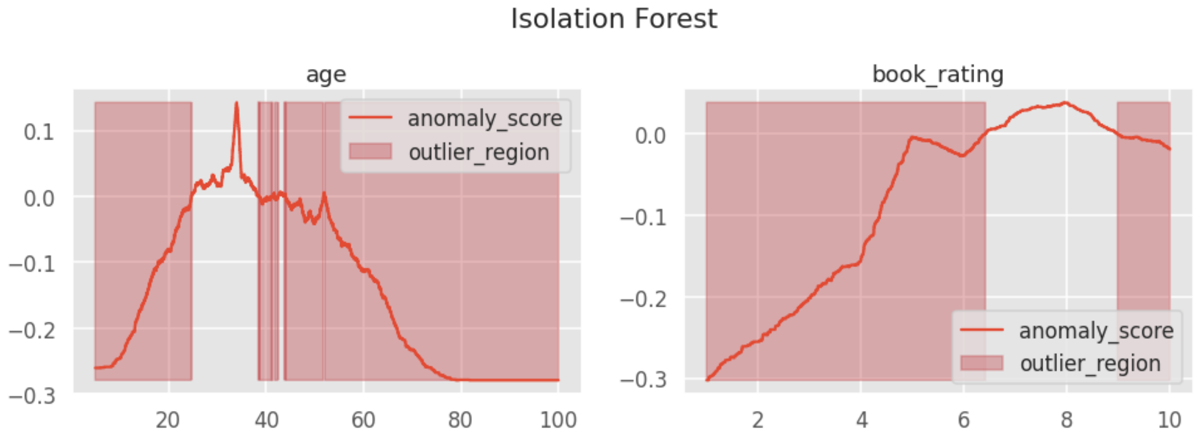


Figure 1-4 Isolation Forest of Original Data

After Dropping Outliers using IQR

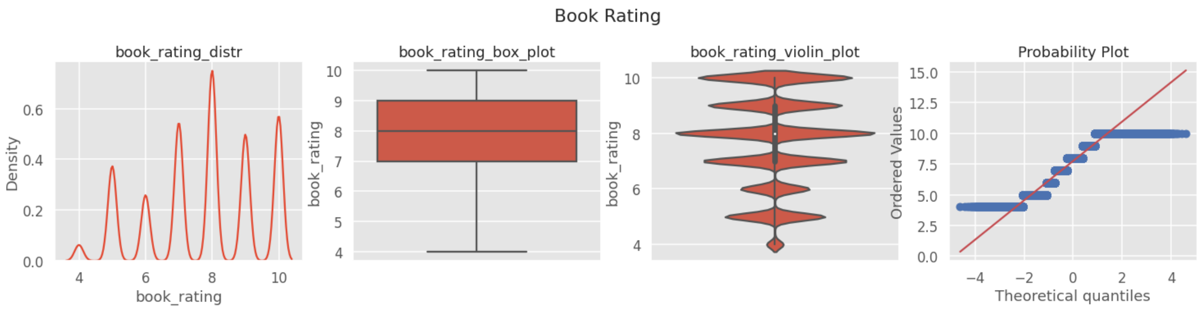


Figure 1-5 Dropping book_rating using IQR

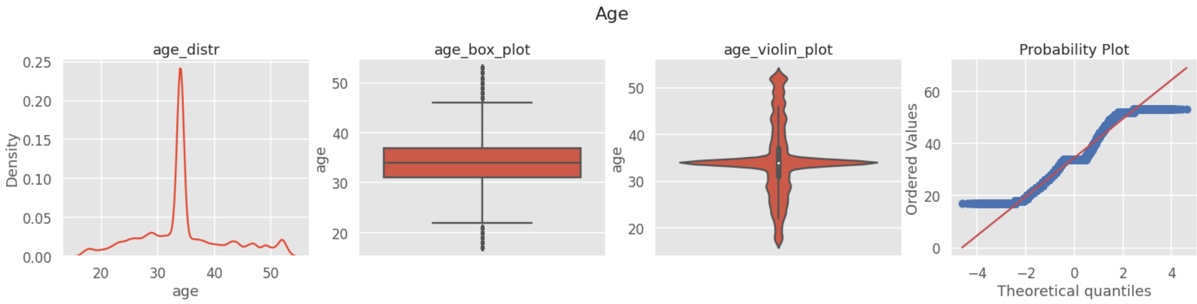


Figure 1-6 Dropping Age using IQR

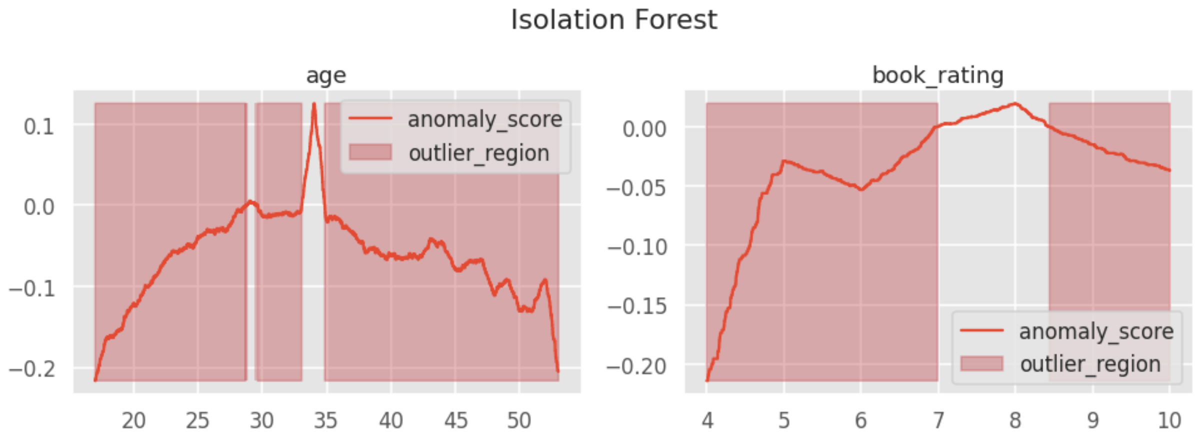


Figure 1-7 Isolation forest after dropping outliers with IQR

Column	Mean	Std	Min	Max	Skewness	Kurtosis
age	36.53	7.69	17	53	0.35	0.16
book_rating	7.74	1.66	4	10	-0.34	-0.80

Removing Outliers with BoxCox

Figure 1-8 Dropping book_rating with BoxCox

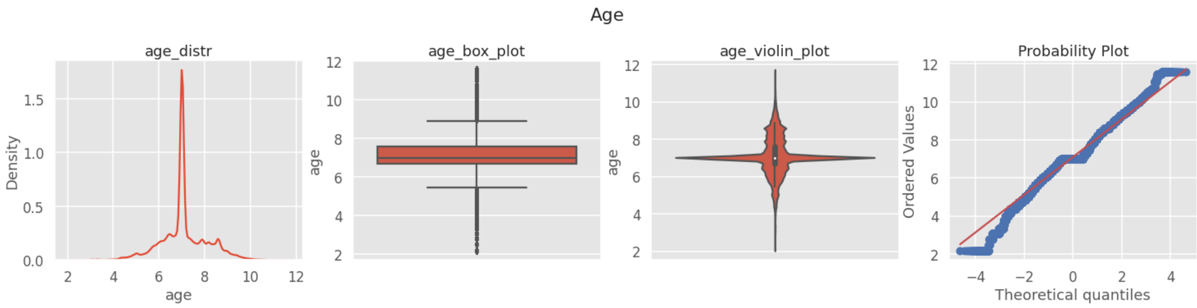


Figure 1-9 Dropping age with BoxCox

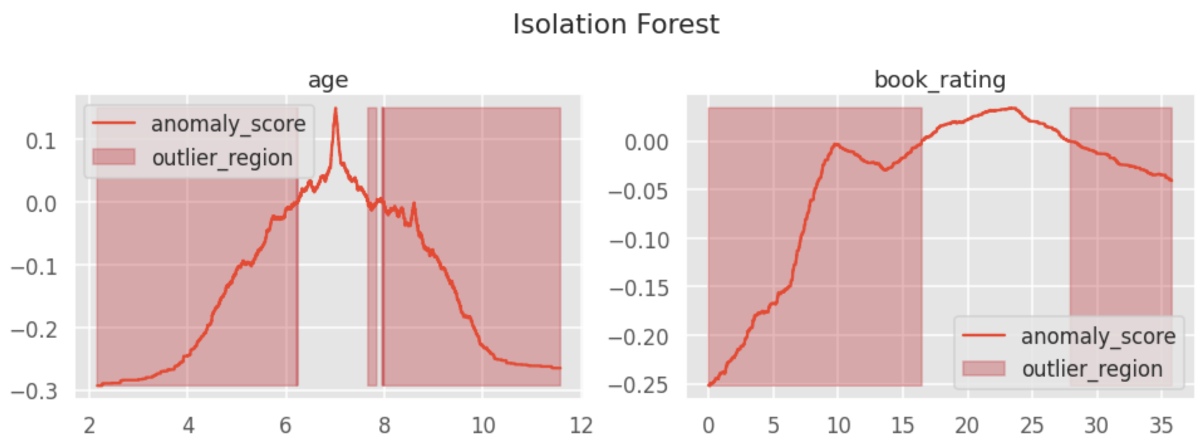


Figure 1-10 Isolation Forest after BoxCox

Column	Mean	Std	Min	Max	Skewness	Kurtosis
age	7.09	1.01	2.16	11.57	0.04	1.14
book_rating	22.62	9.14	0	35	-0.17	-0.83

Removing Outliers with Imputation

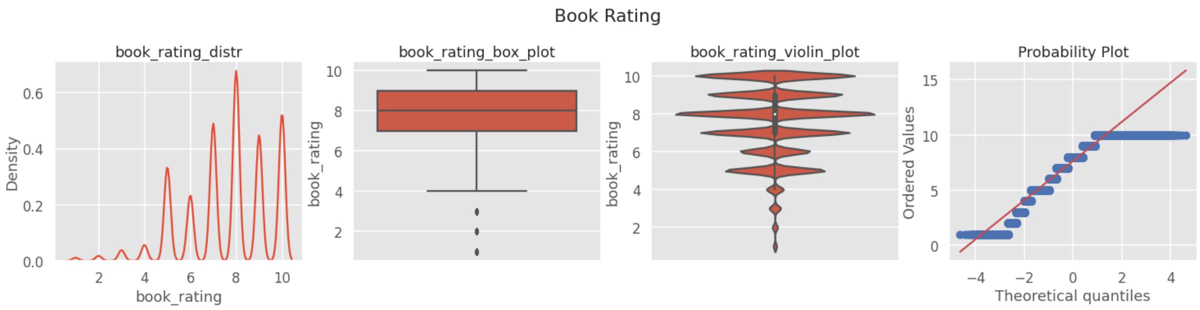


Figure 1-11 Dropping book_rating with imputation

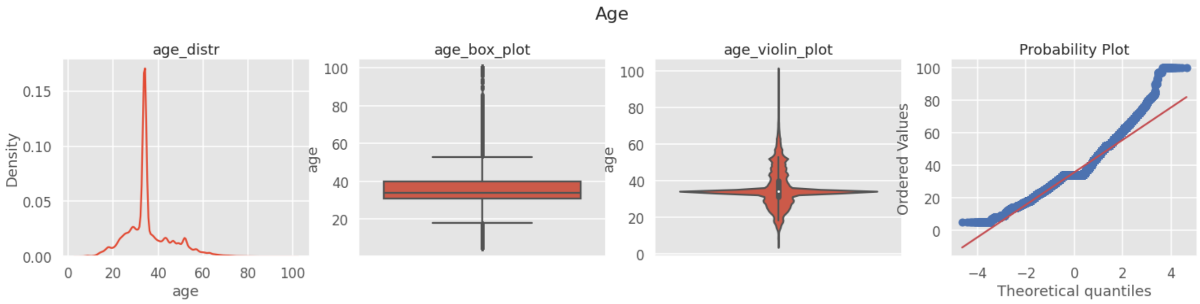


Figure 1-12 Dropping age with imputation

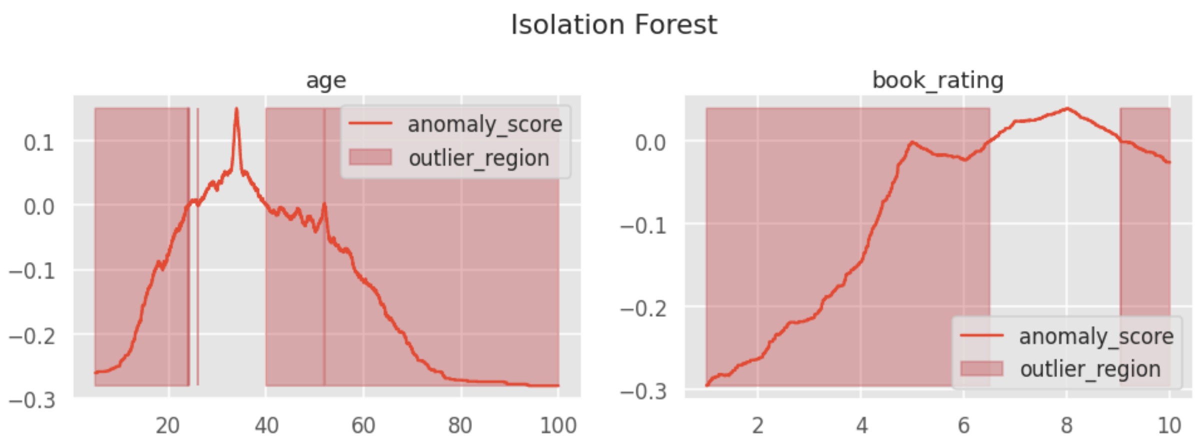
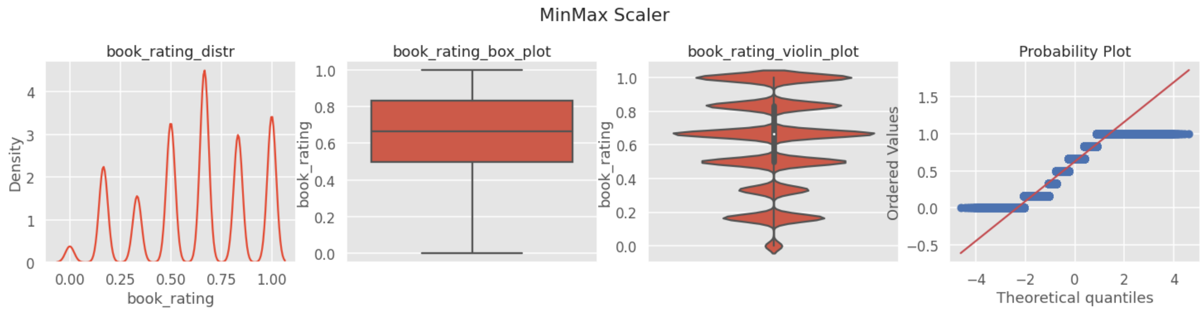


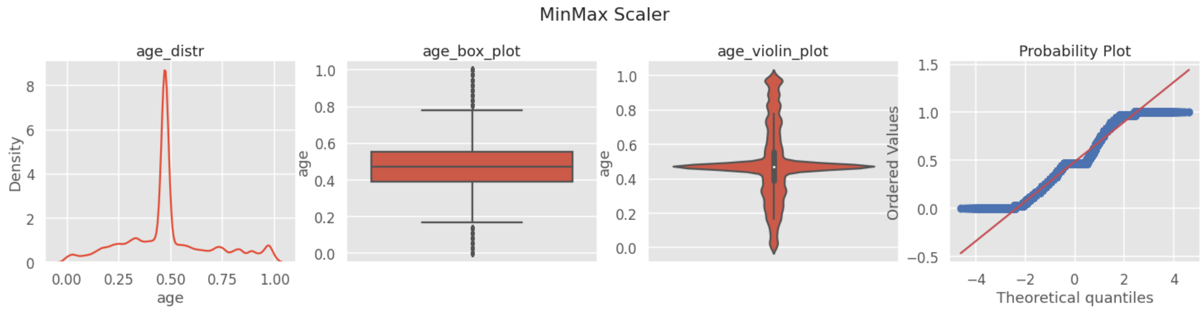
Figure 1-13 Isolation Forest after imputation

Column	Mean	Std	Min	Max	Skewness	Kurtosis
age	35.85	10.36	5	100	0.86	1.64
book_rating	7.62	1.83	1	10	-0.66	-0.12

1.3 Data Normalization

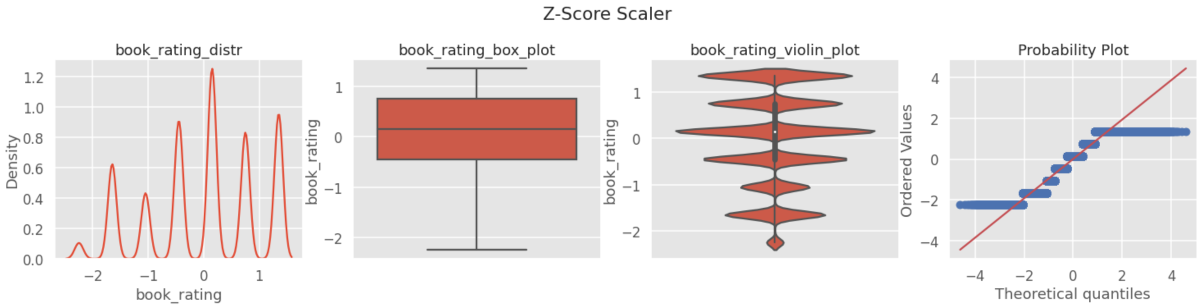
1.3.1 Min-Max Scaling

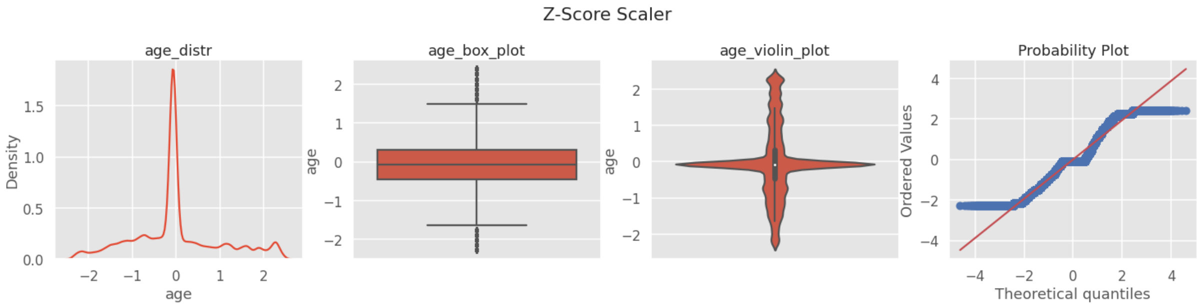




	Mean	Std	Min	Max	Skewness	Kurtosis
book_rating	0.62	0.27	0	1	-0.34	-0.80
age	0.48	0.21	0	1	0.35	0.17

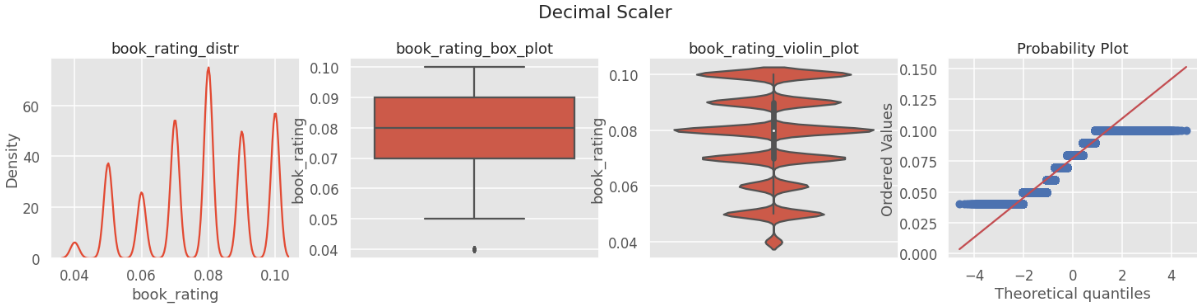
1.3.2 Z-Score Standardization

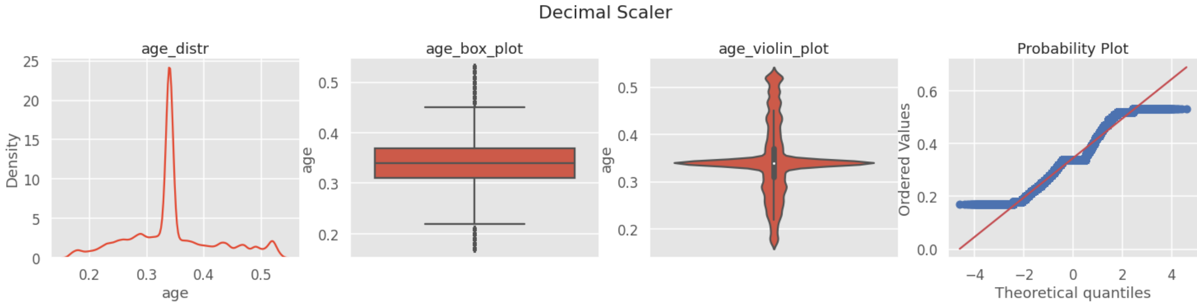




	Mean	Std	Min	Max	Skewness	Kurtosis
book_rating	~0	~1	-2.24	1.35	-0.34	-0.80
age	~0	~1	-2.27	2.39	0.35	0.17

1.3.3 Decimal Scaling

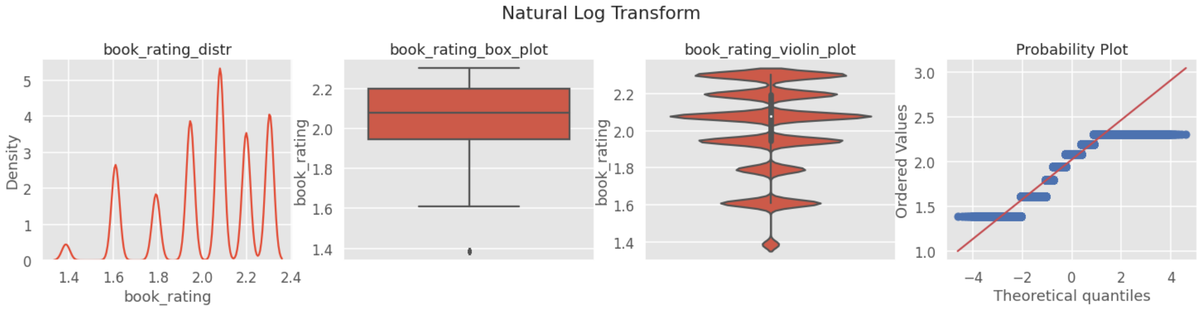


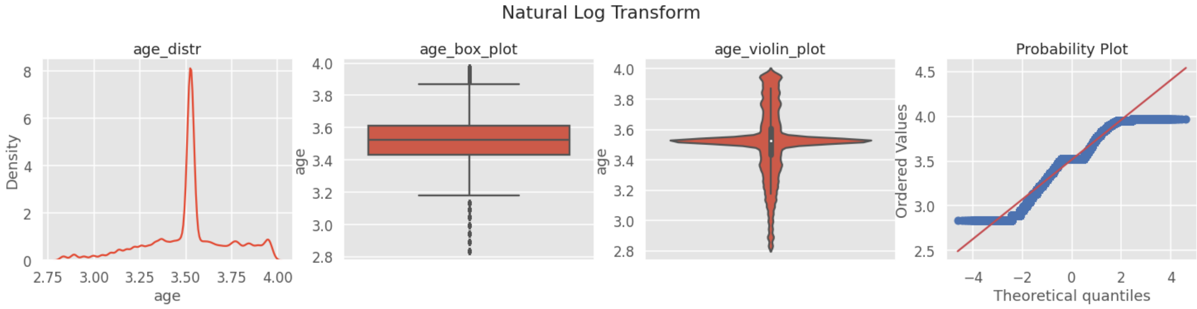


	Mean	Std	Min	Max	Skewness	Kurtosis
book_rating	0.077	0.01	0.04	0.1	-0.34	-0.80
age	0.34	0.07	0.17	0.53	0.35	0.17

1.4 Data Transformation

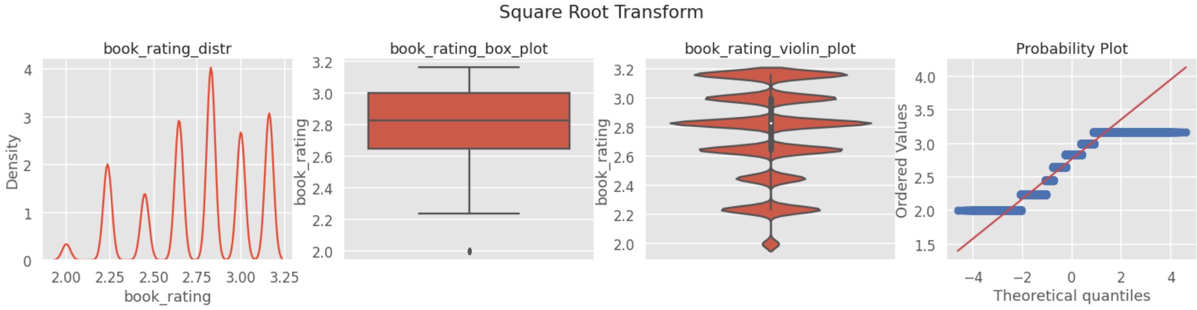
1.4.1 Natural Log Transform

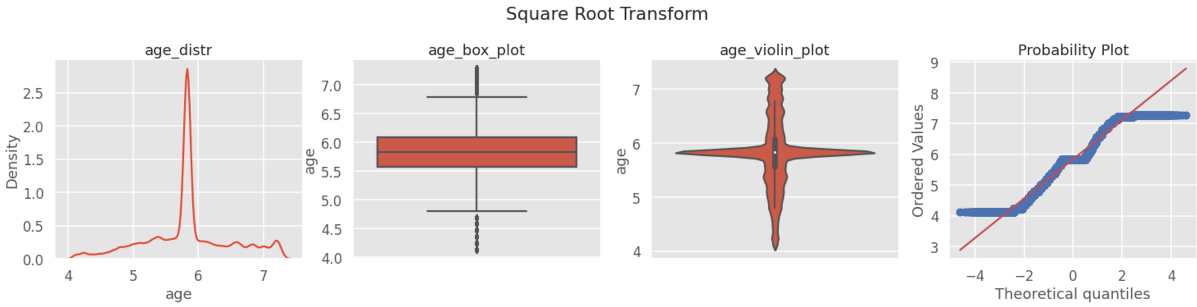




	Mean	Std	Min	Max	Skewness	Kurtosis
book_rating	2.02	0.23	1.38	2.3	-0.74	-0.23
age	3.511	0.229	2.83	3.97	-0.405	0.544

1.4.2 Square Root Transform

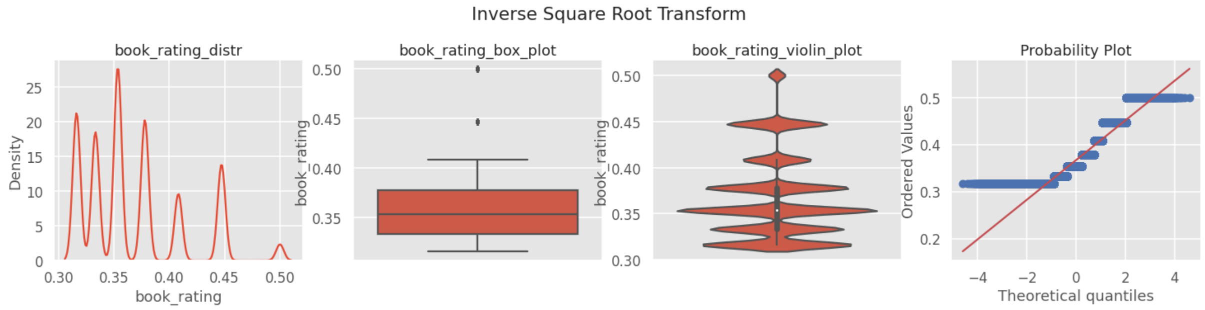


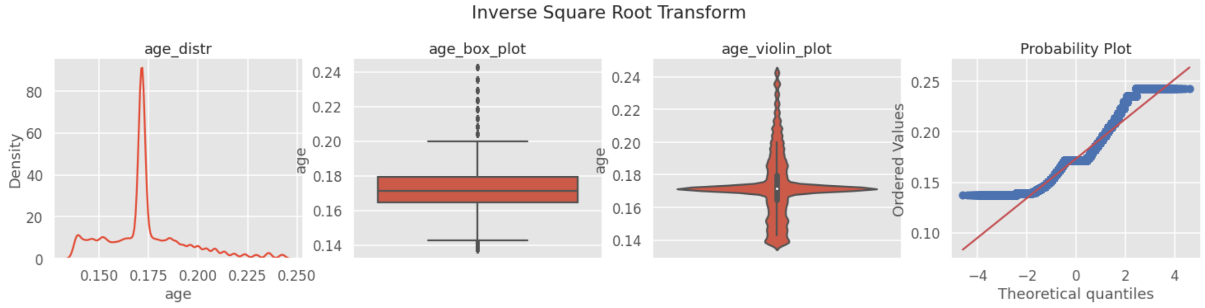


	Mean	Std	Min	Max	Skewness	Kurtosis
book_rating	2.76	0.31	2	3.16	-0.54	-0.57
age	5.84	0.65	4.123	7.28	**-0.0089**	-0.57

NOTE: age has a skewness of 0 using Square Root Transform !

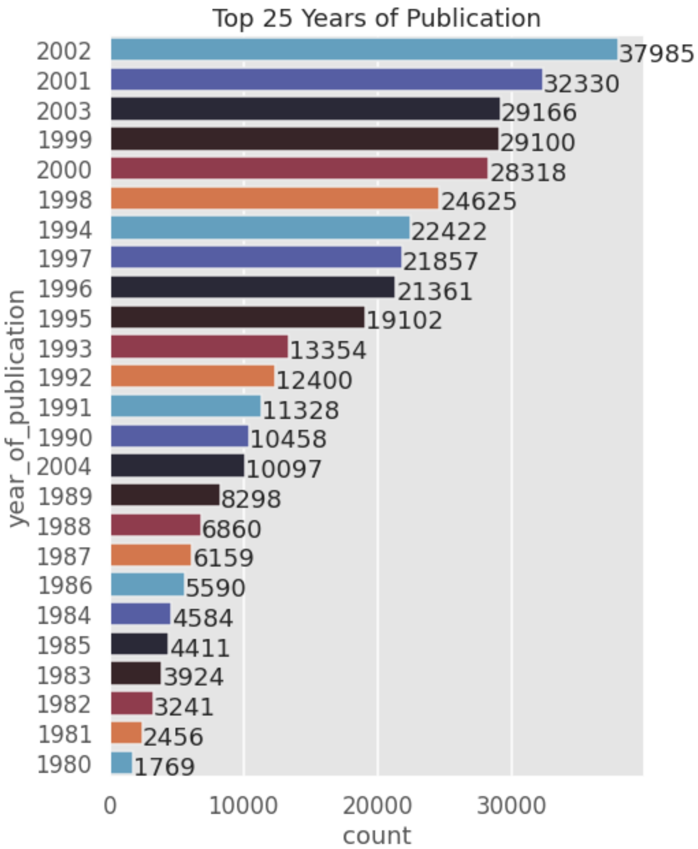
1.4.3 Inverse Square Root Transform

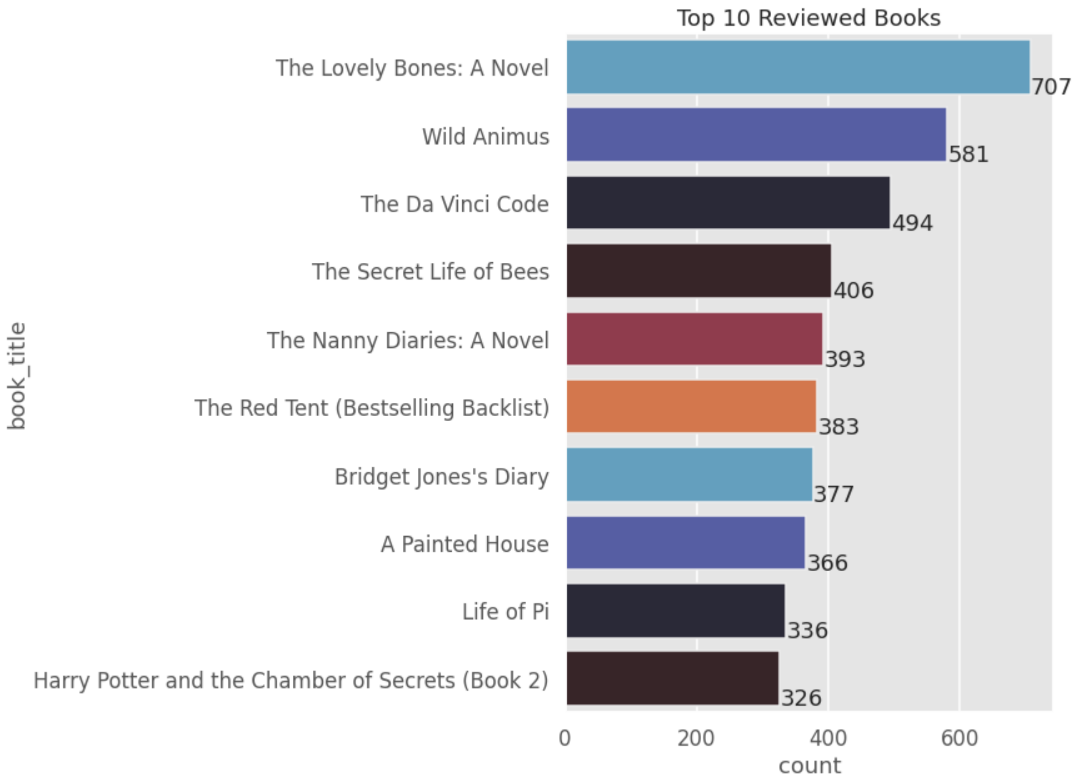


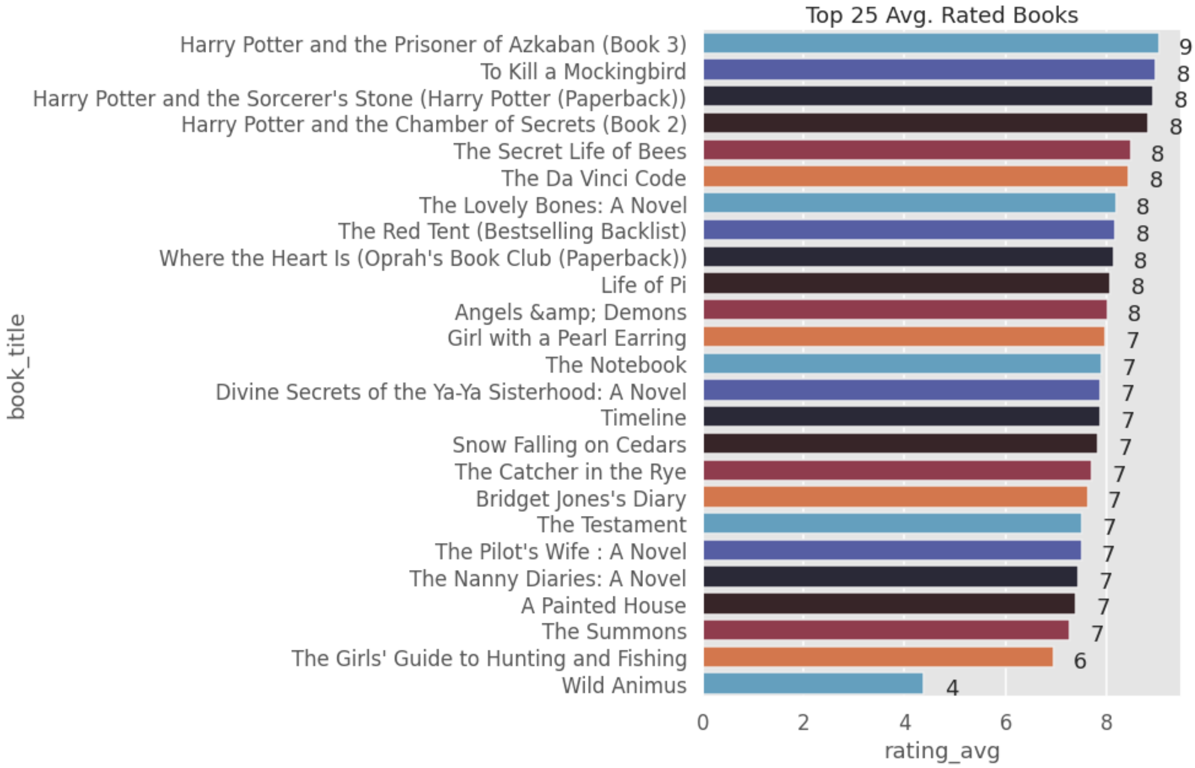


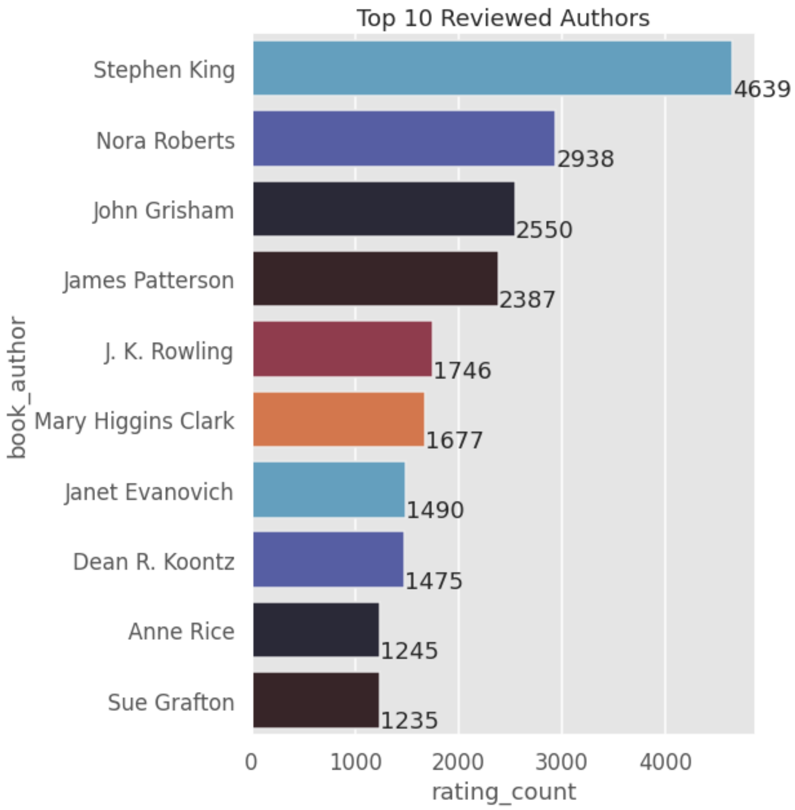
	Mean	Std	Min	Max	Skewness	Kurtosis
book_rating	0.366	0.044	0.31	0.5	0.96	0.24
age	0.17	0.02	0.13	0.24	0.82	1.21

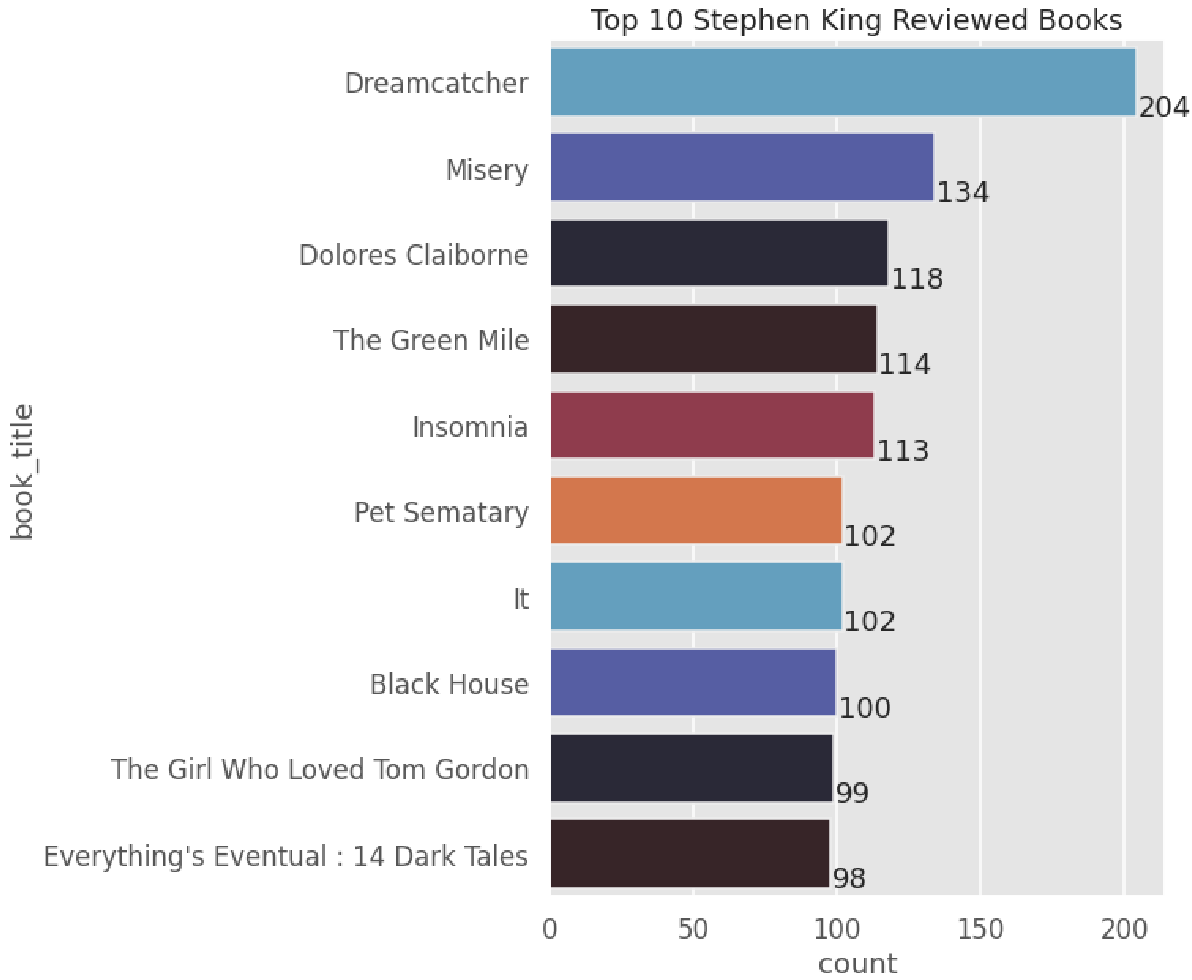
1.5 EDA and Interpretation of Results

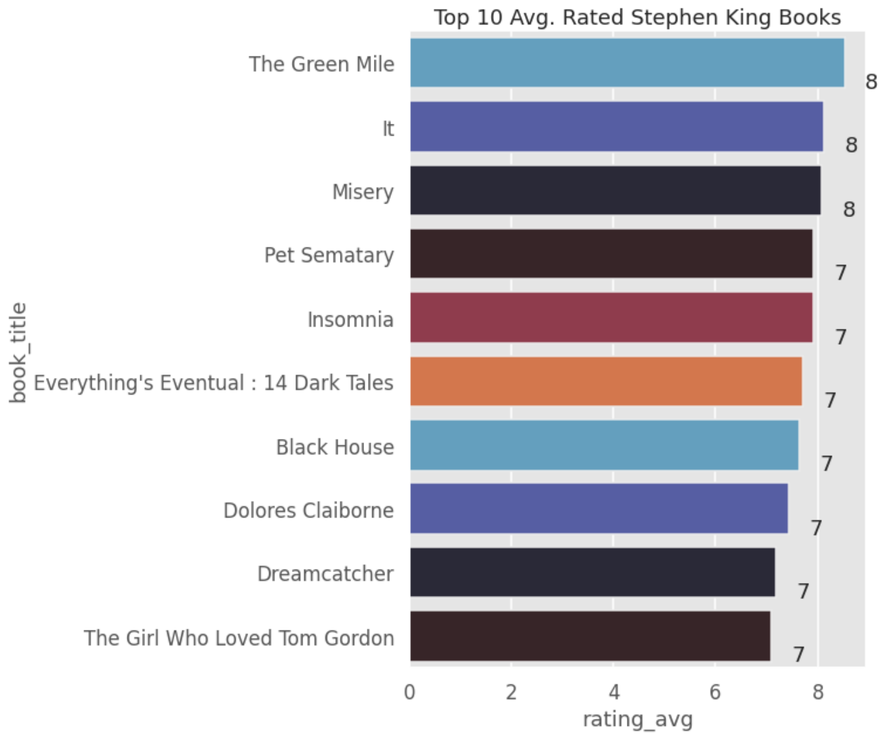





Top 25 Avg. Rated Books
Harry Potter and the Prisoner of Azkaban (Book 3)
To Kill a Mockingbird
Harry Potter and the Sorcerer's Stone (Harry Potter (Paperback))
Harry Potter and the Chamber of Secrets (Book 2)
The Secret Life of Bees
The Da Vinci Code
The Lovely Bones: A Novel
The Red Tent (Bestselling Backlist)
Where the Heart Is (Oprah's Book Club (Paperback))
Life of Pi
Angels & Demons
Girl with a Pearl Earring
The Notebook
Divine Secrets of the Ya-Ya Sisterhood: A Novel
Timeline
Snow Falling on Cedars
The Catcher in the Rye
Bridget Jones's Diary
The Testament
The Pilot's Wife : A Novel
The Nanny Diaries: A Novel
A Painted House
The Summons
The Girls' Guide to Hunting and Fishing
Wild Animus
9
8
8
8
8
8
8
8
8
8
8
7
7
7
7
7
7
7
7
7
7
7
7
6
4
book_title
0
2
4
6
8
rating_avg


Top 10 Reviewed Authors
Stephen King
Nora Roberts
John Grisham
James Patterson
J. K. Rowling
Mary Higgins Clark
Janet Evanovich
Dean R. Koontz
Anne Rice
Sue Grafton
4639
2938
2550
2387
1746
1677
1490
1475
1245
1235
book_author
0
1000
2000
3000
4000
rating_count


Top 10 Stephen King Reviewed Books
Dreamcatcher
Misery
Dolores Claiborne
The Green Mile
Insomnia
Pet Sematary
It
Black House
The Girl Who Loved Tom Gordon
Everything's Eventual : 14 Dark Tales
204
134
118
114
113
102
102
100
99
98
book_title
0
50
100
150
200
count


Top 10 Avg. Rated Stephen King Books
The Green Mile
It
Misery
Pet Sematary
Insomnia
Everything's Eventual : 14 Dark Tales
Black House
Dolores Claiborne
Dreamcatcher
The Girl Who Loved Tom Gordon
8
8
8
7
7
7
7
7
7
7
book_title
0
2
4
6
8
rating_avg

Bibliography

BookReview_EDA_Satyajit_Ghana

December 11, 2020

Book Review - An Exploratory Data Analysis and Data Transformation Notebook

Author: Satyajit Ghana

---

[1]:
```
import gdown

url = 'https://drive.google.com/uc?id=1UPZiTughL3iDtPwreoUs_SX-LfVktrI3'
output = 'BX-CSV-Dump.zip'
gdown.download(url, output, quiet=False)
```

```
Downloading…
From: https://drive.google.com/uc?id=1UPZiTughL3iDtPwreoUs_SX-LfVktrI3
To: /content/BX-CSV-Dump.zip
26.1MB [00:01, 23.9MB/s]
```

[1]: 'BX-CSV-Dump.zip'

[2]:
```
! unzip BX-CSV-Dump.zip
```

```
Archive:  BX-CSV-Dump.zip
  inflating: BX-Book-Ratings.csv
  inflating: BX-Books.csv
  inflating: BX-Users.csv
```

---

1 Book Crossing EDA

[67]:
```
%matplotlib inline

import scipy
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

```
from sklearn.ensemble import IsolationForest
from sklearn.impute import SimpleImputer

sns.set()
palette = sns.color_palette("icefire")

plt.style.use('ggplot')

sns.set_context("talk")
```

1.1 Loading, Cleaning and Merging the Dataset

Reading the csv files

The dataset is semi-colon separated instead of semi-colon separated, and als the incoding is `ISO-8859-1`

[4]:
```
users = pd.read_csv(
    '/content/BX-Users.csv',
    names=['user_id', 'location', 'age'],
    sep=';',
    skiprows=1,
    encoding='ISO-8859-1',
    low_memory=False,
    error_bad_lines=False
)
users
```

[4]:
```
        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
…           …                                    …    …
278853   278854                  portland, oregon, usa   NaN
278854   278855   tacoma, washington, united kingdom  50.0
278855   278856             brampton, ontario, canada   NaN
278856   278857             knoxville, tennessee, usa   NaN
278857   278858                   dublin, n/a, ireland   NaN

[278858 rows x 3 columns]
```

parse the datatypes properly

[5]:
```
users.dtypes
```

[5]:
```
user_id       int64
location     object
age         float64
dtype: object
```

A quick look at the numeric attributes of the users table

[6]:
```
users.describe().T
```

[6]:
```
            count           mean           std  …      50%        75%
max
user_id  278858.0  139429.500000  80499.515020  …  139429.5  209143.75
278858.0
age      168096.0      34.751434     14.428097  …      32.0      44.00
244.0

[2 rows x 8 columns]
```

Data Inconsistency

We notice that the max age is 244, and min age is 0, age cannot be 244 ! so let's fix that, also minimum age cannot be 0, this is likely a mistake during data collection, and missing age values were probably just replaced by 0, so we'll have to fix that

one way is to simply replace the inconsitent values with the mean of the data

[212]:
```
users.loc[(users.age > 100) | (users.age < 5), 'age'] = np.nan
users.age = users.age.fillna(users.age.mean())
```

[8]:
```
users['age'] = users['age'].astype(np.uint8)
```

[9]:
```
users['age'].describe()
```

[9]:
```
count    278858.000000
mean         34.446733
std          10.551712
min           5.000000
25%          29.000000
50%          34.000000
75%          35.000000
max         100.000000
Name: age, dtype: float64
```

checking for any `NA` values

[10]:
```
users.isna().sum()
```

[10]:
```
user_id     0
location    0
```

```
age          0
dtype: int64
```

Now we'll read the books data, same way as before

[11]:
```
books = pd.read_csv(
    '/content/BX-Books.csv',
    names=['isbn', 'book_title', 'book_author', 'year_of_publication',␣
 ↪'publisher', 'img_s', 'img_m', 'img_l'],
    sep=';',
    skiprows=1,
    encoding='ISO-8859-1',
    low_memory=False,
    error_bad_lines=False
)
books
```

[11]:
```
              isbn  …                                            img_l
0       0195153448  …  http://images.amazon.com/images/P/0195153448.0…
1       0002005018  …  http://images.amazon.com/images/P/0002005018.0…
2       0060973129  …  http://images.amazon.com/images/P/0060973129.0…
3       0374157065  …  http://images.amazon.com/images/P/0374157065.0…
4       0393045218  …  http://images.amazon.com/images/P/0393045218.0…
…              …   …                                               …
271374  0440400988  …  http://images.amazon.com/images/P/0440400988.0…
271375  0525447644  …  http://images.amazon.com/images/P/0525447644.0…
271376  006008667X  …  http://images.amazon.com/images/P/006008667X.0…
271377  0192126040  …  http://images.amazon.com/images/P/0192126040.0…
271378  0767409752  …  http://images.amazon.com/images/P/0767409752.0…

[271379 rows x 8 columns]
```

parse the data types properly

[12]:
```
books.dtypes
```

[12]:
```
isbn                   object
book_title             object
book_author            object
year_of_publication    object
publisher              object
img_s                  object
img_m                  object
img_l                  object
dtype: object
```

Dropping Unnecessary Values

drop `['img_s', 'img_m', 'img_l']` since they are not useful for us

[13]:
```
books = books.drop(['img_s', 'img_m', 'img_l'], axis=1)
```

year_of_publication should be a integer

[14]:
```
books['year_of_publication'] = pd.to_numeric(books['year_of_publication'],␣
↪errors='coerce')
```

[15]:
```
books.loc[(books['year_of_publication'] == 0) | (books['year_of_publication'] >␣
↪2008), 'year_of_publication' ] = np.nan
books['year_of_publication'] = books['year_of_publication'].
↪fillna(round(books['year_of_publication'].mean()))
books['year_of_publication'] = pd.to_numeric(books['year_of_publication'],␣
↪downcast='unsigned')
```

Checking for any NA values

[16]:
```
books.isna().sum()
```

[16]:
```
isbn                   0
book_title             0
book_author            1
year_of_publication    0
publisher              2
dtype: int64
```

Since the NA rows are very few, we'll simply drop them

[17]:
```
books = books.dropna()
```

[18]:
```
books.describe().T
```

[18]:
```
                        count         mean       std  …     50%     75%
max
year_of_publication  271376.0  1993.692427  8.248715  …  1995.0  2000.0
2008.0

[1 rows x 8 columns]
```

Read the ratings dataset as usual

[19]:
```
ratings = pd.read_csv(
    '/content/BX-Book-Ratings.csv',
    names=['user_id', 'isbn', 'book_rating'],
    sep=';',
    skiprows=1,
    encoding='ISO-8859-1',
    low_memory=False,
    error_bad_lines=False
```

```
)
ratings
```

[19]:

```
         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
…            …            …          …
1149775   276704   1563526298            9
1149776   276706   0679447156            0
1149777   276709   0515107662           10
1149778   276721   0590442449           10
1149779   276723  05162443314            8

[1149780 rows x 3 columns]
```

[20]:

```
ratings['book_rating'] = ratings['book_rating'].astype(np.uint8)
```

[21]:

```
ratings.dtypes
```

[21]:

```
user_id         int64
isbn           object
book_rating     uint8
dtype: object
```

check for any `NA` values

[22]:

```
ratings.isna().sum()
```

[22]:

```
user_id        0
isbn           0
book_rating    0
dtype: int64
```

There are some inconsistent values, which we will fix later

[23]:

```
ratings.describe().T.astype(np.int32)
```

[23]:

	count	mean	std	min	25%	50%	75%	max
user_id	1149780	140386	80562	2	70345	141010	211028	278854
book_rating	1149780	2	3	0	0	0	7	10

Join the three datasets based on `user_id` and `isbn` as the key

[24]:

```
temp = pd.merge(users, ratings, on='user_id')
temp = pd.merge(temp, books, on='isbn')
```

```
dataset = temp.copy()
```

[25]:
```
dataset
```

[25]:
```
          user_id  …                   publisher
0               2  …   Oxford University Press
1               8  …     HarperFlamingo Canada
2           11400  …     HarperFlamingo Canada
3           11676  …     HarperFlamingo Canada
4           41385  …     HarperFlamingo Canada
…             … …                          …
1031167    278851  …      Simon &amp; Schuster
1031168    278851  …              Broadway Books
1031169    278851  …             Lone Star Books
1031170    278851  …                  Kqed Books
1031171    278851  …  American Map Corporation

[1031172 rows x 9 columns]
```

Split the location into `city`, `state` and `country` and replacing missing location details with just `n/a`

We might use these attributes at a later stage of the assignment in collborative filtering, for now it seems useful so we'll keep them.

[26]:
```
location = dataset['location'].str.split(', ', n=2, expand=True)
location.columns = ['city', 'state', 'country']
location = location.fillna('n/a')
```

[27]:
```
dataset['city'] = location['city'] ; dataset['state'] = location['state'] ;␣
↪dataset['country'] = location['country']
```

[28]:
```
dataset = dataset.drop(['location'], axis=1)
```

[29]:
```
dataset.describe().T.astype(np.int32)
```

[29]:
```
                       count    mean    std  …     50%     75%     max
user_id              1031172  140594  80524  …  141210  211426  278854
age                  1031172      36     10  …      34      41     100
book_rating          1031172       2      3  …       0       7      10
year_of_publication  1031172    1995      7  …    1997    2001    2008

[4 rows x 8 columns]
```

checking for `NA` values

[30]:
```
dataset.isna().sum()
```

[30]:
```
user_id                0
age                    0
isbn                   0
book_rating            0
book_title             0
book_author            0
year_of_publication    0
publisher              0
city                   0
state                  0
country                0
dtype: int64
```

So we have 1,031,172 values in total, that's a lot !

[31]:
```
dataset.shape
```

[31]:
```
(1031172, 11)
```

[32]:
```
dataset.dtypes
```

[32]:
```
user_id                 int64
age                     uint8
isbn                   object
book_rating             uint8
book_title             object
book_author            object
year_of_publication    uint16
publisher              object
city                   object
state                  object
country                object
dtype: object
```

This will be the final dataset we will be working with !

[33]:
```
dataset.head(5)
```

[33]:
```
   user_id  age        isbn  …      city       state country
0        2   18  0195153448  …  stockton  california     usa
1        8   34  0002005018  …   timmins     ontario  canada
2    11400   49  0002005018  …    ottawa     ontario  canada
3    11676   34  0002005018  …       n/a         n/a     n/a
4    41385   34  0002005018  …   sudbury     ontario  canada

[5 rows x 11 columns]
```

[34]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1031172 entries, 0 to 1031171
Data columns (total 11 columns):
 #   Column               Non-Null Count    Dtype
---  ------               --------------    -----
 0   user_id              1031172 non-null  int64
 1   age                  1031172 non-null  uint8
 2   isbn                 1031172 non-null  object
 3   book_rating          1031172 non-null  uint8
 4   book_title           1031172 non-null  object
 5   book_author          1031172 non-null  object
 6   year_of_publication  1031172 non-null  uint16
 7   publisher            1031172 non-null  object
 8   city                 1031172 non-null  object
 9   state                1031172 non-null  object
 10  country              1031172 non-null  object
dtypes: int64(1), object(7), uint16(1), uint8(2)
memory usage: 74.7+ MB
```

[35]:
```
cleaned_data = dataset.copy()
```

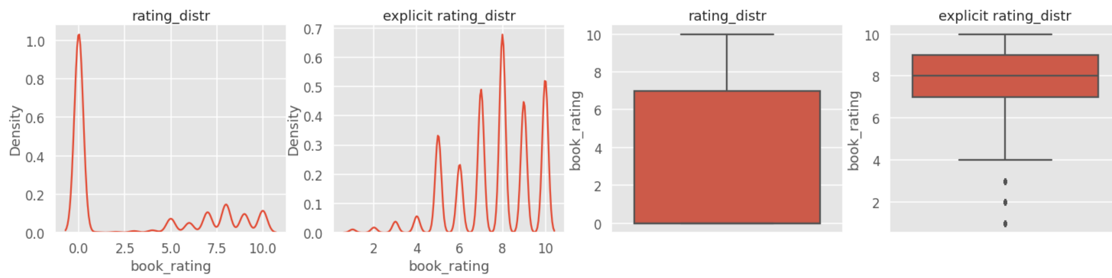
[204]:
```
# dataset = cleaned_data.copy()
```

1.2 Analyzing the Feature Space

[213]:
```
def get_skewness(data, columns):
    """returns the skewness and kurtosis of the specified attributes"""
    skewness = data[columns].skew()
    kurtosis = data[columns].kurtosis()

    df = {
        'skewness': skewness.values,
        'kurtosis': kurtosis.values
    }

    dataframe = pd.DataFrame(data=df, index=columns)

    return dataframe
```

[206]:
```
f, axes = plt.subplots(ncols = 4, figsize=(25, 5))
sns.kdeplot(x="book_rating", data=dataset, ax=axes[0]).set_title('rating_distr')
sns.kdeplot(x="book_rating", data=dataset[dataset['book_rating'] != 0],
↪ax=axes[1]).set_title('explicit rating_distr')
sns.boxplot(y="book_rating", data=dataset, orient='v', ax=axes[2]).
↪set_title('rating_distr')
```

```
sns.boxplot(y="book_rating", data=dataset[dataset['book_rating'] != 0],␣
↪orient='v', ax=axes[3]).set_title('explicit rating_distr')
plt.show()
```



Something we can clearly notice in **rating_distr** is that 0 ratings are a lot ! and when we drop them to get **explicit_rating_distr**, the distribution is lot better, even the box plot is better, so it makes sense to remove the 0 rated values

Why would someone rate a book 0 ? this likely are the values for user that forgot to rate the book, people are lazy right ?

For normally distributed data, the skewness should be about zero. For unimodal continuous distributions, a skewness value greater than zero means that there is more weight in the right tail of the distribution. The function skewtest can be used to determine if the skewness value is close enough to zero, statistically speaking.

[207]:
```
get_skewness(dataset, ['age', 'book_rating'])
```

[207]:

	skewness	kurtosis
age	0.832497	1.343472
book_rating	0.752445	-1.214994

[208]:
```
dataset[['age', 'book_rating']].describe().T
```

[208]:

	count	mean	std	min	25%	50%	75%	max
age	1031172.0	36.232554	10.413914	5.0	31.0	34.0	41.0	100.0
book_rating	1031172.0	2.839005	3.854142	0.0	0.0	0.0	7.0	10.0

We can remove `0` ratings, since these are unrated, and why would someone rate a book as `0` ?

[209]:
```
dataset = dataset[dataset['book_rating'] != 0]
```

We notice that skewness of the **book_rating** changes from a strong positive, to negative !

[210]:
```
get_skewness(dataset, ['age', 'book_rating'])
```

[210]:

	skewness	kurtosis
age	0.859061	1.644862

```
book_rating -0.661283  0.120288
```

[211]:
```
dataset[['age', 'book_rating']].describe().T
```

[211]:

	count	mean	std	min	25%	50%	75%	max
age	383849.0	35.856535	10.363322	5.0	31.0	34.0	40.0	100.0
book_rating	383849.0	7.626702	1.841335	1.0	7.0	8.0	9.0	10.0

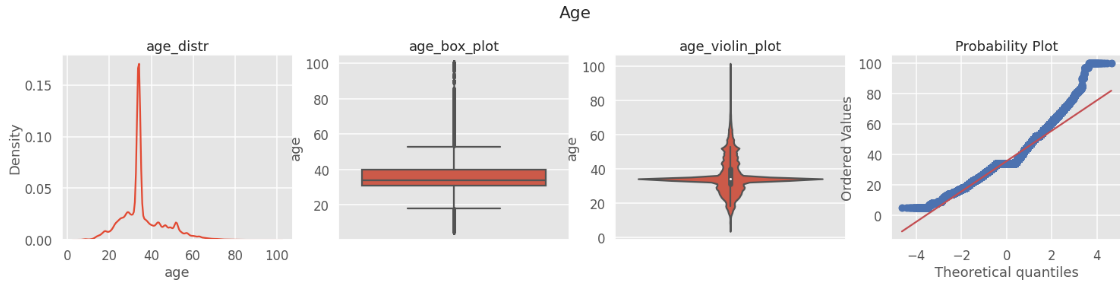
1.2.1 Univariate Analysis of Attributes

Here we plot the Distribution, Box Plot, Violin Plot and QQ Plot for the numeric attributes `age` and `book_rating`

[133]:
```
def plot_univariate(dataset, column_name, suptitle = None, kde_only = True):
    f, axes = plt.subplots(ncols = 4, figsize=(25, 5))
    if kde_only:
        sns.kdeplot(x=column_name, data=dataset, ax=axes[0]).
↪set_title(f'{column_name}_distr')
    else:
        sns.histplot(x=column_name, data=dataset, kde=True, ax=axes[0]).
↪set_title(f'{column_name}_distr')
    sns.boxplot(y=column_name, data=dataset, orient='v', ax=axes[1]).
↪set_title(f'{column_name}_box_plot')
    sns.violinplot(y=column_name, data=dataset, orient='v', ax=axes[2]).
↪set_title(f'{column_name}_violin_plot')
    scipy.stats.probplot(dataset[column_name], dist="norm", plot=axes[3])
    if suptitle:
        plt.suptitle(suptitle)
        plt.subplots_adjust(top=0.80)
    plt.show()
```

[134]:
```
plot_univariate(dataset=dataset, column_name='book_rating', suptitle='Book␣
↪Rating')
```

[135]:
```
plot_univariate(dataset=dataset, column_name='age', suptitle='Age')
```



Inference

- `book_rating`

  We see that the distribution of books is not normal, also the box plot shows outliers in the lower region, ratings from 1 to 4, the probability plot also confirms this, this is something we deal with outlier analysis later

- `age`

  Age has a peak density about 30-34, the box-plot shows a lot of outlisers, and the qq plot also

1.3 Outlier Analysis and removing them

We'll try to infer the outliers using the `IsolationForest` algorithm,

The IsolationForest 'isolates' observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature.

Since recursive partitioning can be represented by a tree structure, the number of splittings required to isolate a sample is equivalent to the path length from the root node to the terminating node.

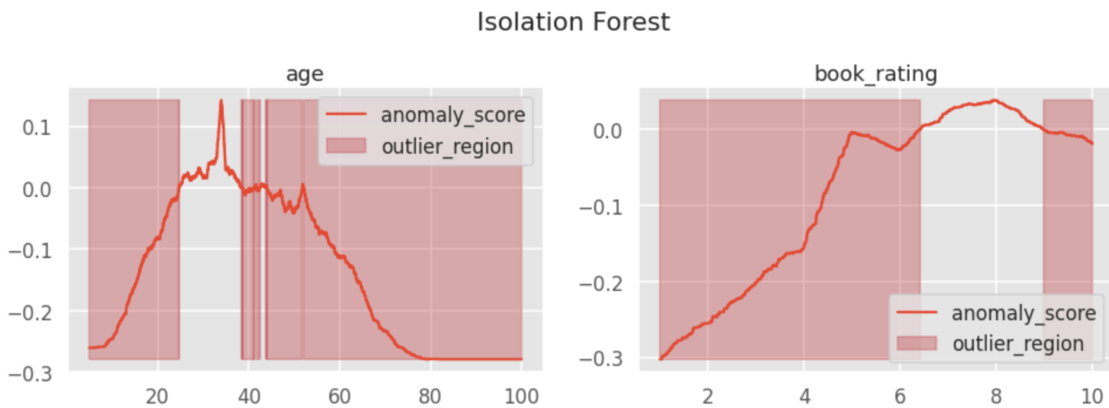
This path length, averaged over a forest of such random trees, is a measure of normality and our decision function.

Random partitioning produces noticeably shorter paths for anomalies. Hence, when a forest of random trees collectively produce shorter path lengths for particular samples, they are highly likely to be anomalies. (Scikit-learn)

1. It classify the data point to outlier and not outliers and works great with very high dimensional data.
2. It works based on decision tree and it isolate the outliers.
3. If the result is -1, it means that this specific data point is an outlier. If the result is 1, then it means that the data point is not an outlier.

[136]:
```
def plot_isolation_forest(data, columns, suptitle = None):
    ncols = len(columns)
    fig, axes = plt.subplots(nrows=1, ncols=ncols, figsize=(len(columns) * 8,␣
↪5))
    isolation_forest = IsolationForest(contamination='auto')
    for i, column in enumerate(columns):
        isolation_forest.fit(data[column].values.reshape(-1, 1))

        xx = np.linspace(data[column].min(), data[column].max(),␣
↪len(data[column])).reshape(-1, 1)
        anomaly_score = isolation_forest.decision_function(xx)
        outlier = isolation_forest.predict(xx)

        axes[i].plot(xx, anomaly_score, label='anomaly_score')
        axes[i].fill_between(xx.T[0], np.min(anomaly_score), np.
↪max(anomaly_score),
                             where=outlier == -1, color='r',
                             alpha=.4, label='outlier_region')
        axes[i].legend()
        axes[i].set_title(column)

    if suptitle:
        plt.suptitle(suptitle)
        plt.subplots_adjust(top=0.80)
```

[137]:
```
plot_isolation_forest(data=dataset, columns=['age', 'book_rating'],␣
↪suptitle='Isolation Forest')
```



Isolation Forest shows us that a lot of data in our range contains anomalies

---

1.3.1 Drop Outliers with IQR

In this method by using Inter Quartile Range (IQR), we detect outliers. IQR tells us the variation in the data set. Any value, which is beyond the range of $-1.5 \times IQR$ to $1.5 \times IQR$ treated as outliers

[138]:
```
def drop_outliers(data, columns):
    data_new = data.copy()

    for column in columns:
        iqr = 1.5 * (np.percentile(data_new[column], 75) - np.
 ↪percentile(data_new[column], 25))
        data_new.drop(data_new[data_new[column] > (iqr + np.
 ↪percentile(data_new[column], 75))].index, inplace=True)
        data_new.drop(data_new[data_new[column] < (np.
 ↪percentile(data_new[column], 25) - iqr)].index, inplace=True)

    return data_new
```

[139]:
```
dataset_wo_outliers = drop_outliers(dataset, ['age', 'book_rating'])
```

[140]:
```
dataset_wo_outliers[['age', 'book_rating']].describe().T
```

[140]:

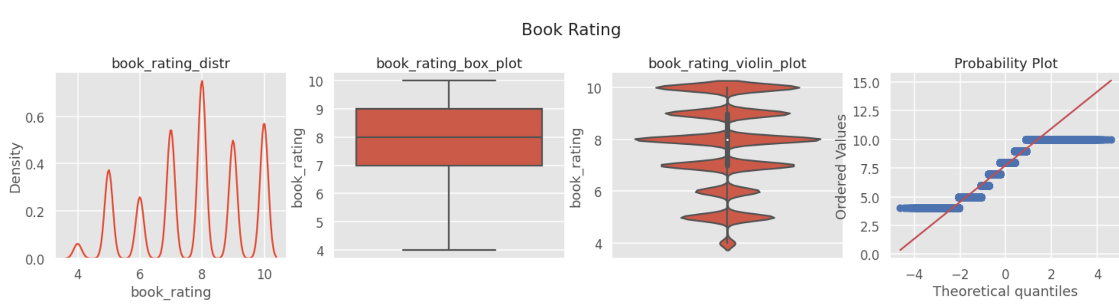
	count	mean	std	min	25%	50%	75%	max
age	343213.0	34.534607	7.697323	17.0	31.0	34.0	37.0	53.0
book_rating	343213.0	7.742178	1.668787	4.0	7.0	8.0	9.0	10.0

[141]:
```
get_skewness(dataset_wo_outliers, ['age', 'book_rating'])
```

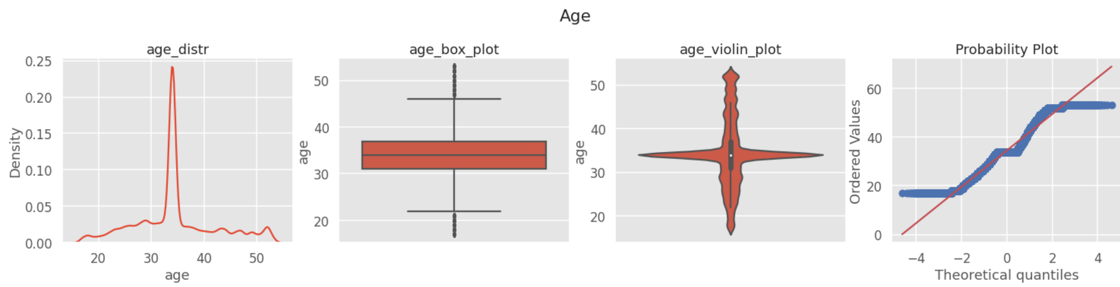
[141]:

	skewness	kurtosis
age	0.356242	0.169093
book_rating	-0.349030	-0.808479

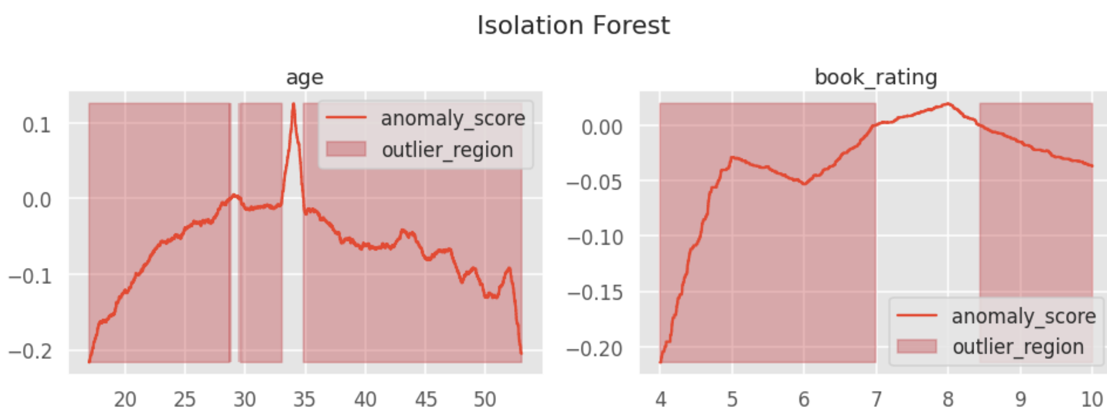
[142]:
```
plot_univariate(dataset=dataset_wo_outliers, column_name='book_rating',␣
 ↪suptitle='Book Rating')
```



[143]:
```
plot_univariate(dataset=dataset_wo_outliers, column_name='age', suptitle='Age')
```



[144]:
```
plot_isolation_forest(data=dataset_wo_outliers, columns=['age', 'book_rating'],␣
↪suptitle='Isolation Forest')
```



1.3.2 BoxCox

A Box Cox transformation is a transformation of a non-normal dependent variables into a normal shape. Normality is an important assumption for many statistical techniques; if your data isn't normal, applying a Box-Cox means that you are able to run a broader number of tests.

The BoxCox transformation is given by

```
y = (x**lmbda - 1) / lmbda,  for lmbda > 0
    log(x),                  for lmbda = 0
```

The confidence limits returned when alpha is provided give the interval where:

$$llf(\hat{\lambda}) - llf(\lambda) < \frac{1}{2}\chi^2(1-\alpha, 1)$$

with `llf` the log-likelihood function and the chi-squared function.

[145]:
```python
def apply_boxcox(data, columns):
    data_new = data.copy()

    for column in columns:
        data_new[column], _ = scipy.stats.boxcox(data_new[column].astype(np.
↪float32), lmbda=None)

    return data_new
```

[146]:
```python
dataset_boxcox = apply_boxcox(dataset, ['age', 'book_rating'])
```

[147]:
```python
dataset_boxcox[['age', 'book_rating']].describe().T
```

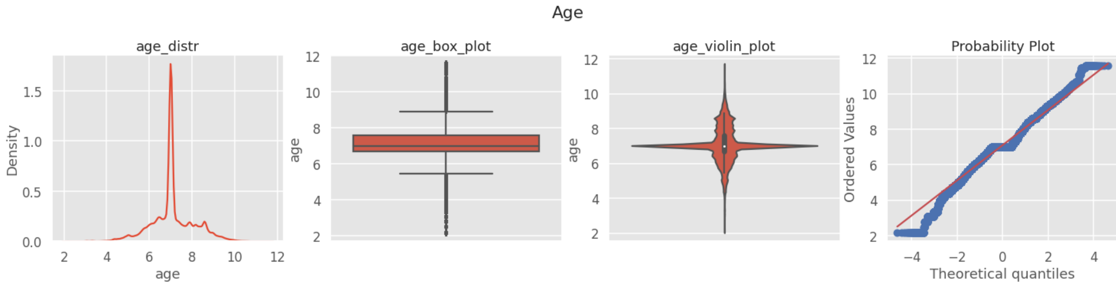
[147]:
```
                count       mean       std  …        50%        75%        max
age          383849.0   7.096895  1.012567  …   7.009771   7.591456  11.576612
book_rating  383849.0  22.629185  9.147474  …  23.642124  29.426302  35.763058

[2 rows x 8 columns]
```

[148]:
```python
get_skewness(dataset_boxcox, ['age', 'book_rating'])
```

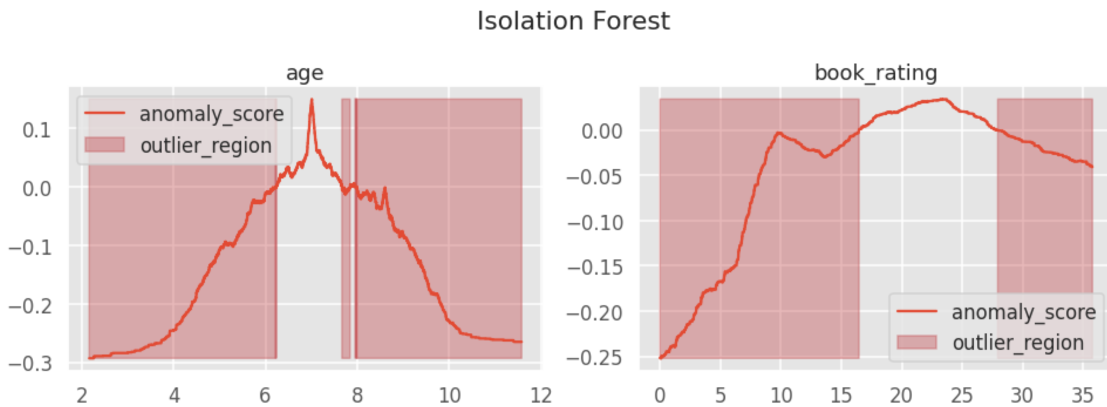
[148]:
```
             skewness  kurtosis
age          0.042891  1.143758
book_rating -0.170664 -0.837900
```

[149]:
```python
plot_univariate(dataset=dataset_boxcox, column_name='book_rating',␣
↪suptitle='Book Rating')
```

[150]:
```python
plot_univariate(dataset=dataset_boxcox, column_name='age', suptitle='Age')
```



[151]:
```
plot_isolation_forest(data=dataset_boxcox, columns=['age', 'book_rating'],␣
↪suptitle='Isolation Forest')
```



1.3.3 Imputation

Another types of outliers are caused by missing values, for which one type of imputation algorithm is univariate, which imputes values in the i-th feature dimension using only non-missing values in that feature dimension (e.g. `impute.SimpleImputer`). (Scikit-learn)

[152]:
```
def apply_imputation(data, columns):
    data_new = data.copy()

    imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

    for column in columns:
        data_new[column] = imputer.fit_transform(data_new[column].astype(np.
↪float32).values.reshape(-1, 1))

    return data_new
```

[153]:
```
dataset_imputed = apply_imputation(data=dataset, columns=['age', 'book_rating'])
```

[154]:
```
dataset_imputed[['age', 'book_rating']].describe().T
```

[154]:

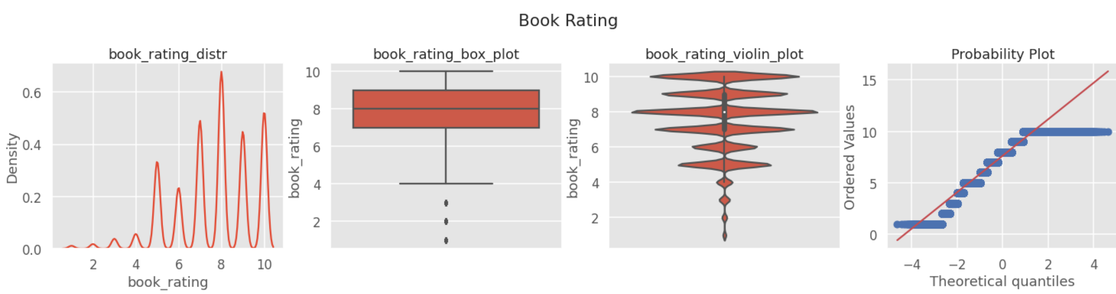
	count	mean	std	min	25%	50%	75%	max
age	383849.0	35.856533	10.366258	5.0	31.0	34.0	40.0	100.0
book_rating	383849.0	7.626702	1.838844	1.0	7.0	8.0	9.0	10.0

[155]:
```
get_skewness(dataset_imputed, ['age', 'book_rating'])
```

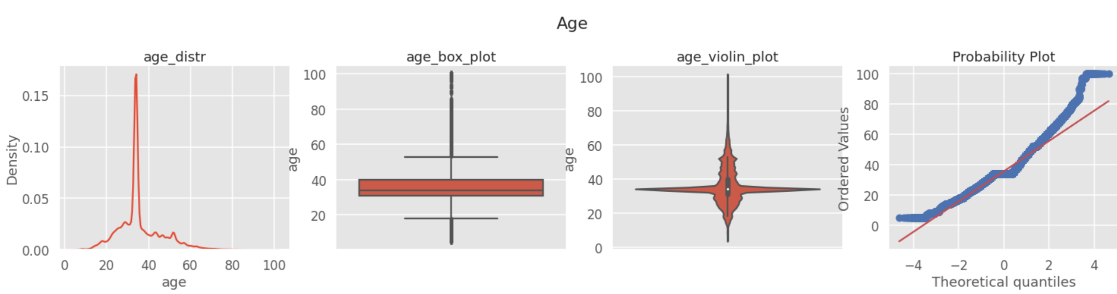
[155]:

	skewness	kurtosis
age	0.859061	1.644862
book_rating	-0.661283	0.120288

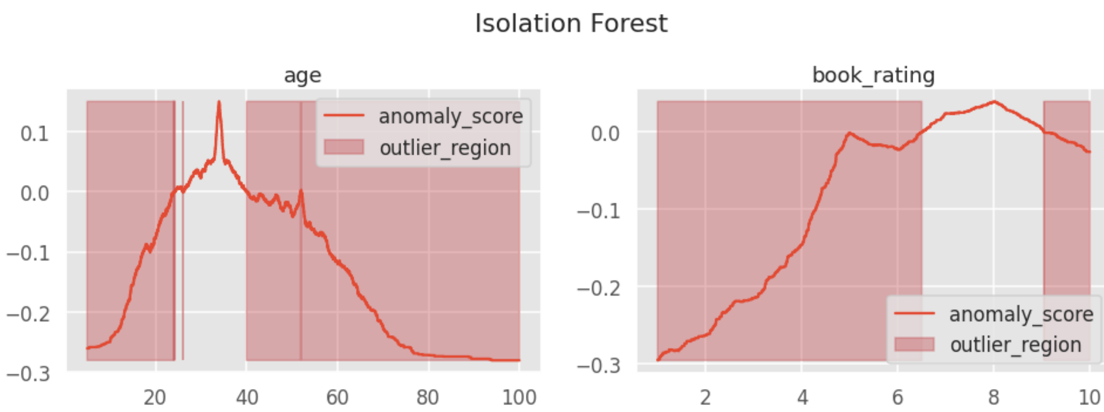
[156]:
```
plot_univariate(dataset=dataset_imputed, column_name='book_rating',␣
↪suptitle='Book Rating')
```



[157]:
```
plot_univariate(dataset=dataset_imputed, column_name='age', suptitle='Age')
```



[158]:
```
plot_isolation_forest(data=dataset_imputed, columns=['age', 'book_rating'],␣
↪suptitle='Isolation Forest')
```



You'll observe that imputation didn't help us much, since our data does not have `nan` values, but this is a demonstration that imputation can also be used

---

1.4 Data Transformation

[223]:
```
dataset_trans = dataset_wo_outliers.copy()
```

[259]:
```
def apply_function(data, columns, function):
    data_new = data.copy()

    for column in columns:
        data_new[column] = function(data_new[column])

    return data_new
```

1.4.1 Min-Max Normalization

This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one.

The transformation is given by:

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min
```

where `min, max = feature_range`.

This transformation is often used as an alternative to zero mean, unit variance scaling.

The mathematical formula being:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

[266]:
```
def min_max_scaling(series):
    scaler = MinMaxScaler()
    scaled_vals = scaler.fit_transform(series.values.reshape(-1, 1)).reshape(-1)
    return scaled_vals
```

[267]:
```
scaled_dset = apply_function(dataset_trans, ['age', 'book_rating'],␣
 ↪min_max_scaling)
scaled_dset[['age', 'book_rating']]
```

[267]:
```
              age  book_rating
1        0.472222     0.166667
3        0.472222     0.666667
5        0.361111     0.666667
8        0.472222     0.833333
9        0.472222     0.833333
…             …            …
1031166  0.444444     0.500000
1031168  0.444444     0.166667
1031169  0.444444     0.500000
1031170  0.444444     0.500000
1031171  0.444444     1.000000

[343213 rows x 2 columns]
```

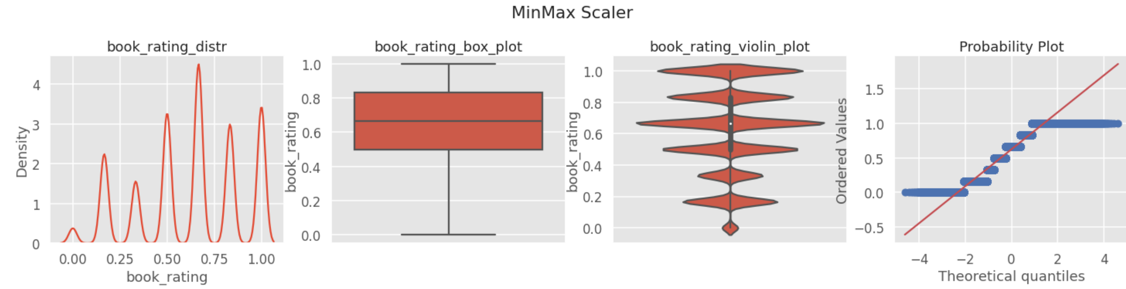
[268]:
```
scaled_dset[['age', 'book_rating']].describe().T
```

[268]:
```
                count      mean       std  …       50%       75%  max
age          343213.0  0.487072  0.213815  …  0.472222  0.555556  1.0
book_rating  343213.0  0.623696  0.278131  …  0.666667  0.833333  1.0

[2 rows x 8 columns]
```

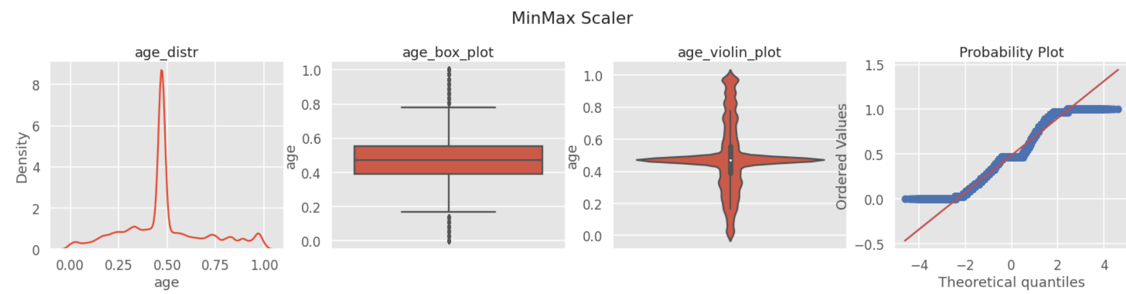
[227]:
```
get_skewness(scaled_dset, ['book_rating', 'age'])
```

[227]:
```
             skewness  kurtosis
book_rating -0.349030 -0.808479
age          0.356242  0.169093
```

[229]:
```
plot_univariate(dataset=scaled_dset, column_name='book_rating',␣
 ↪suptitle='MinMax Scaler')
```



[230]:
```
plot_univariate(dataset=scaled_dset, column_name='age', suptitle='MinMax␣
↪Scaler')
```



Inference

- The skewness has now becomes `-0.34` for book_rating and `0.35` for age

1.4.2 Z-Score Standardization

Standardize features by removing the mean and scaling to unit variance

The standard score of a sample `x` is calculated as:

$$z = \frac{x - \mu}{\sigma}$$

where u is the mean of the training samples or zero if `with_mean=False`, and s is the standard deviation of the training samples or one if `with_std=False`.

Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using transform.

Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data (e.g. Gaussian with 0 mean and unit variance).

For instance many elements used in the objective function of a learning algorithm (such as the RBF kernel of Support Vector Machines or the L1 and L2 regularizers of linear models) assume that all features are centered around 0 and have variance in the same order. If a feature has a variance that is orders of magnitude larger that others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected. (Scikit-learn)

[269]:
```
def z_score_scaling(series):
    scaler = StandardScaler()
    scaled_vals = scaler.fit_transform(series.values.reshape(-1, 1)).reshape(-1)
    return scaled_vals
```

[270]:
```
scaled_dset = apply_function(dataset_trans, ['age', 'book_rating'],␣
↪z_score_scaling)
scaled_dset[['age', 'book_rating']]
```

[270]:
```
              age  book_rating
1       -0.069454    -1.643219
3       -0.069454     0.154497
5       -0.589116     0.154497
8       -0.069454     0.753736
9       -0.069454     0.753736
…             …            …
1031166 -0.199369    -0.444741
1031168 -0.199369    -1.643219
1031169 -0.199369    -0.444741
1031170 -0.199369    -0.444741
1031171 -0.199369     1.352974

[343213 rows x 2 columns]
```

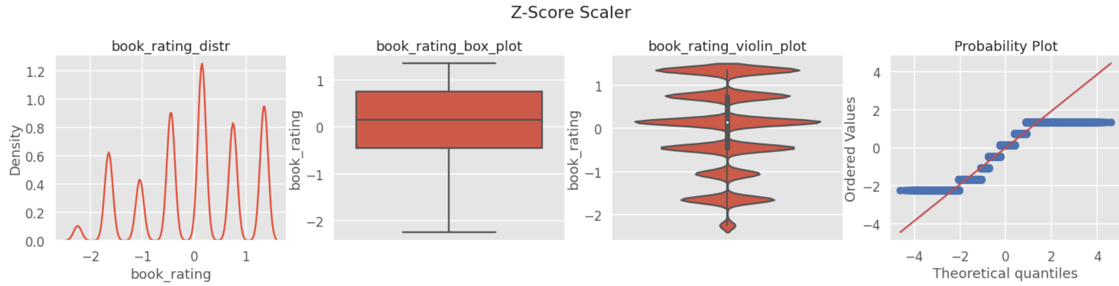
[271]:
```
scaled_dset[['age', 'book_rating']].describe().T
```

[271]:
```
                count          mean       std  …       50%       75%       max
age          343213.0 -1.057090e-13  1.000001  … -0.069454  0.320293  2.398940
book_rating  343213.0  9.450267e-16  1.000001  …  0.154497  0.753736  1.352974

[2 rows x 8 columns]
```

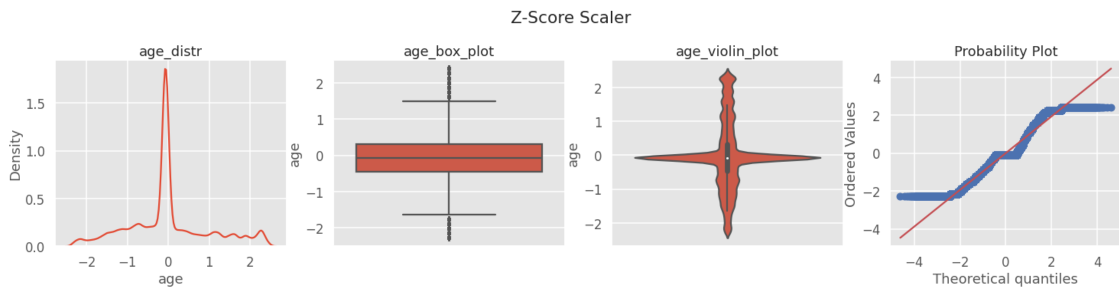
[234]:
```
get_skewness(scaled_dset, ['book_rating', 'age'])
```

[234]:
```
             skewness  kurtosis
book_rating -0.349030 -0.808479
age          0.356242  0.169093
```

[235]:
```
plot_univariate(dataset=scaled_dset, column_name='book_rating',␣
↪suptitle='Z-Score Scaler')
```



[236]:
```python
plot_univariate(dataset=scaled_dset, column_name='age', suptitle='Z-Score Scaler')
```



1.4.3 Decimal Scaling

Decimal scaling is a data normalization technique like Z score, Min-Max, and normalization with standard deviation. Decimal scaling is a data normalization technique. In this technique, we move the decimal point of values of the attribute. This movement of decimal points totally depends on the maximum value among all values in the attribute.

The formula is given by:

$$v_i^{'} = \frac{v_i}{10^j}$$

[272]:
```python
def decimal_scaling(series):
    p = series.max()
    q = len(str(abs(p)))
    scaled_vals = series.values / 10 ** q

    return scaled_vals
```

[273]:
```python
scaled_dset = apply_function(dataset_trans, ['age', 'book_rating'], decimal_scaling)
scaled_dset[['age', 'book_rating']]
```

[273]:
```
             age  book_rating
1           0.34         0.05
3           0.34         0.08
5           0.30         0.08
8           0.34         0.09
9           0.34         0.09
…         …          …
1031166  0.33         0.07
1031168  0.33         0.05
1031169  0.33         0.07
1031170  0.33         0.07
1031171  0.33         0.10

[343213 rows x 2 columns]
```

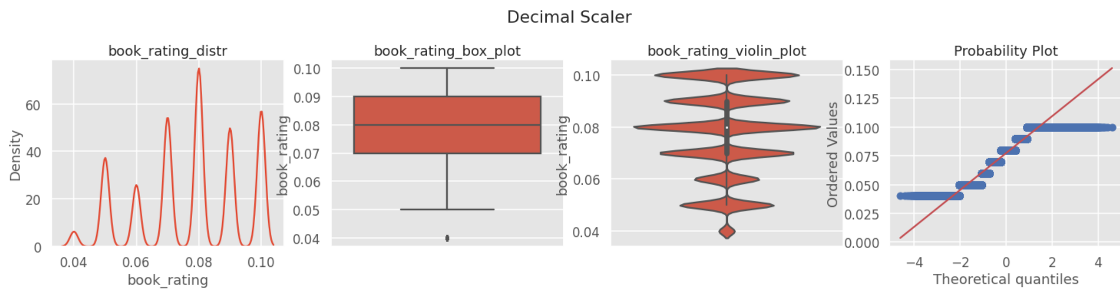
[274]:
```
scaled_dset[['age', 'book_rating']].describe().T
```

[274]:
```
                count      mean       std   min   25%   50%   75%   max
age          343213.0  0.345346  0.076973  0.17  0.31  0.34  0.37  0.53
book_rating  343213.0  0.077422  0.016688  0.04  0.07  0.08  0.09  0.10
```

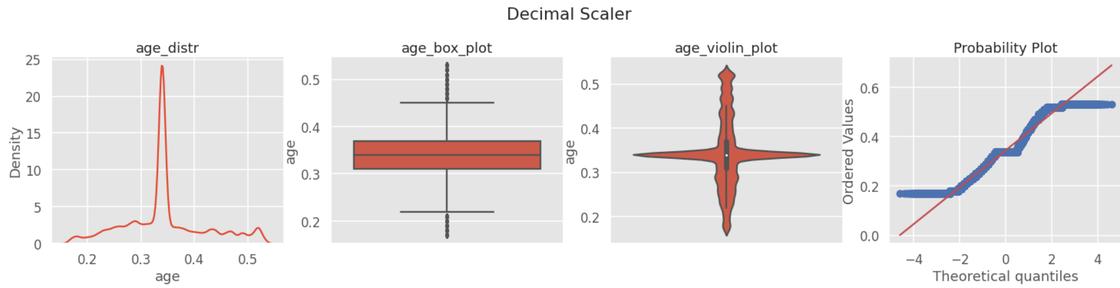
[240]:
```
get_skewness(scaled_dset, ['book_rating', 'age'])
```

[240]:
```
             skewness  kurtosis
book_rating -0.349030 -0.808479
age          0.356242  0.169093
```

[241]:
```
plot_univariate(dataset=scaled_dset, column_name='book_rating',␣
↪suptitle='Decimal Scaler')
```



[242]:
```
plot_univariate(dataset=scaled_dset, column_name='age', suptitle='Decimal␣
↪Scaler')
```



1.5 Data Normality

When a metric variable fails to satisfy the assumption of normality, homogeneity of variance, or linearity, we may be able to crrect the deficiency by using a transformation

- If the distribution of a variable is negatively skewed the adjustment of the values reverses, or reflects, the distribution so that it becomes positively skeweed. The transformations are then computed on the values in the positively skewed distribution.
- Reflection is computed by subtracting all of the values for a variable from one plus the absolute value of maximum value for the variable. This results in a positively skewed distribution with all values larger than zero.

Which transformation?

The main criterion in choosing a transformation is: what works with the data? As examples above indicate, it is important to consider as well two questions.

What makes physical (biological, economic, whatever) sense, for example in terms of limiting behaviour as values get very small or very large? This question often leads to the use of logarithms.

Can we keep dimensions and units simple and convenient? If possible, we prefer measurement scales that are easy to think about. The cube root of a volume and the square root of an area both have the dimensions of length, so far from complicating matters, such transformations may simplify them. Reciprocals usually have simple units, as mentioned earlier. Often, however, somewhat complicated units are a sacrifice that has to be made.

It is not always necessary or desirable to transform a data set to resemble a normal distribution. However, if symmetry or normality are desired, they can often be induced through one of the power transformations.;

1.5.1 Natural Log Transform

Log transformation or $\log_e x$ is a strong transformation with a major effect on distribution shape, it is commonly used for reducing **right skewness** and is often appropriate for measured values. It cannot be applied to zero or negative values. One unit on a logarithm scale means a multiplication by the base of logarithms being used. Exponential growth or decline.

$$x_{new} = \ln x$$

[275]:
```
def natural_log_transform(series):
    trans_vals = np.log(series.astype(np.float32))

    return trans_vals
```

[276]:
```
trans_dset = apply_function(dataset_trans, ['age', 'book_rating'],␣
 ↪natural_log_transform)
trans_dset[['age', 'book_rating']]
```

[276]:
```
              age  book_rating
1        3.526361     1.609438
3        3.526361     2.079442
5        3.401197     2.079442
8        3.526361     2.197225
9        3.526361     2.197225
…             …          …
1031166  3.496508     1.945910
1031168  3.496508     1.609438
1031169  3.496508     1.945910
1031170  3.496508     1.945910
1031171  3.496508     2.302585

[343213 rows x 2 columns]
```

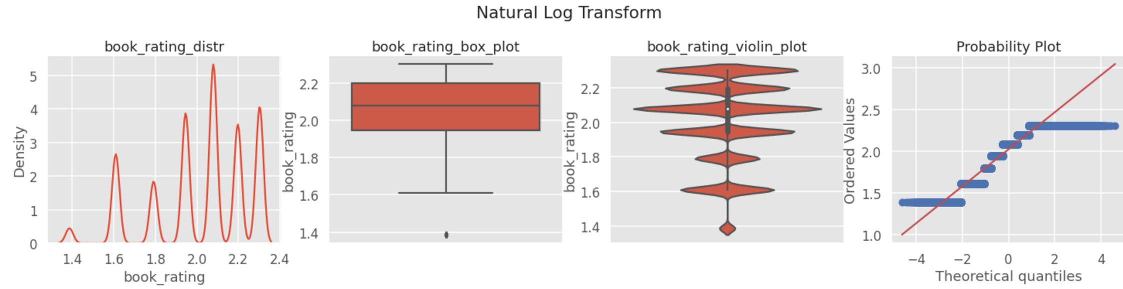
[277]:
```
trans_dset[['age', 'book_rating']].describe().T
```

[277]:
```
                count      mean       std  …       50%       75%       max
age          343213.0  3.511600  0.229122  …  3.526361  3.610918  3.970292
book_rating  343213.0  2.021872  0.234512  …  2.079442  2.197225  2.302585

[2 rows x 8 columns]
```

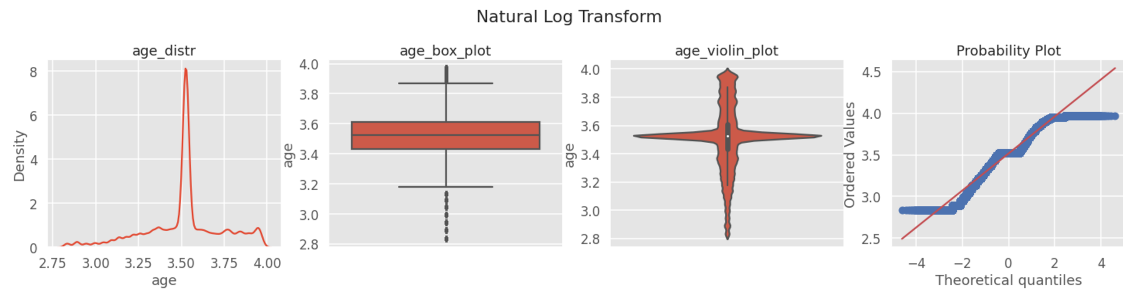
[247]:
```
get_skewness(trans_dset, ['book_rating', 'age'])
```

[247]:
```
             skewness  kurtosis
book_rating -0.747712 -0.236441
age         -0.405089  0.544089
```

[248]:
```
plot_univariate(dataset=trans_dset, column_name='book_rating',␣
 ↪suptitle='Natural Log Transform')
```




```
[249]: plot_univariate(dataset=trans_dset, column_name='age', suptitle='Natural Log␣
        ↪Transform')
```



1.5.2 Square Root Transform

The square root, x to x^(1/2) = sqrt(x), is a transformation with a moderate effect on distribution shape: it is weaker than the logarithm and the cube root. It is also used for reducing right skewness, and also has the advantage that it can be applied to zero values. Note that the square root of an area has the units of a length. It is commonly applied to counted data, especially if the values are mostly rather small.

$$x_{new} = \sqrt{x}$$

```
[278]: def sqrt_transform(series):
           trans_vals = np.sqrt(series.astype(np.float32))

           return trans_vals
```

```
[279]: trans_dset = apply_function(dataset_trans, ['age', 'book_rating'],␣
        ↪sqrt_transform)
       trans_dset[['age', 'book_rating']]
```

```
[279]:            age  book_rating
       1      5.830952     2.236068
```

```
3        5.830952     2.828427
5        5.477226     2.828427
8        5.830952     3.000000
9        5.830952     3.000000
…           …            …
1031166  5.744563     2.645751
1031168  5.744563     2.236068
1031169  5.744563     2.645751
1031170  5.744563     2.645751
1031171  5.744563     3.162278

[343213 rows x 2 columns]
```

[280]:
```
trans_dset[['age', 'book_rating']].describe().T
```

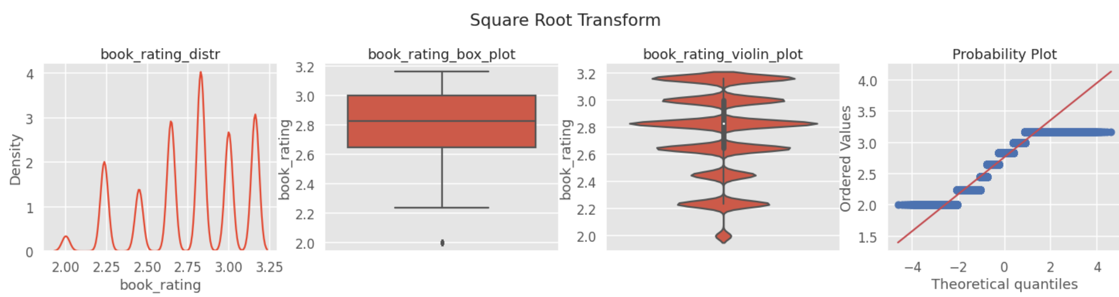
[280]:
```
                count      mean       std  …       50%       75%       max
age          343213.0  5.846551  0.656963  …  5.830952  6.082763  7.280110
book_rating  343213.0  2.765608  0.310675  …  2.828427  3.000000  3.162278

[2 rows x 8 columns]
```

[253]:
```
get_skewness(trans_dset, ['book_rating', 'age'])
```

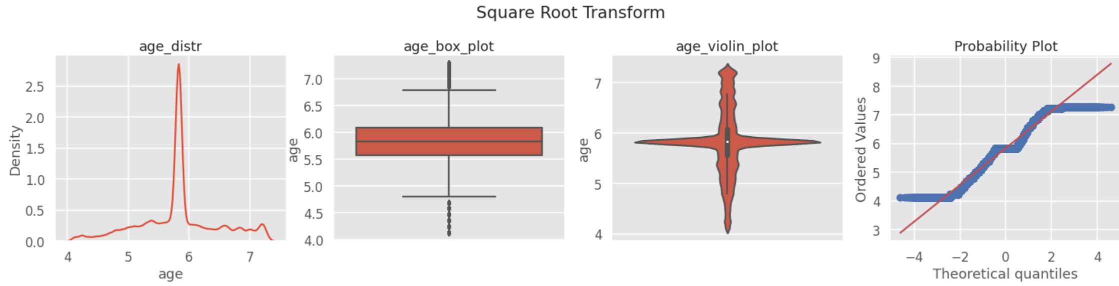
[253]:
```
             skewness  kurtosis
book_rating -0.542093 -0.576979
age         -0.008925  0.214740
```

NOTE: Skewness has a value of ~0 for age using Square Root Transform

[254]:
```
plot_univariate(dataset=trans_dset, column_name='book_rating', suptitle='Square␣
↪Root Transform')
```



[255]:
```
plot_univariate(dataset=trans_dset, column_name='age', suptitle='Square Root␣
↪Transform')
```



1.5.3 Inverse Square Root Transformation

The inverse square root, 1/sqrt(x) is a very strong transformation with a drastic effect on distribution shape. It can not be applied to zero values or negative values, it is not useful unless all values are positive.

$$x_{new} = \frac{1}{\sqrt{x}}$$

[281]:
```
def inv_sqrt_transform(series):
    trans_vals = np.power(series.astype(np.float32), -1/2)

    return trans_vals
```

[285]:
```
trans_dset = apply_function(dataset_trans, ['age', 'book_rating'],␣
 ↪inv_sqrt_transform)
trans_dset[['age', 'book_rating']]
```

[285]:
```
              age  book_rating
1        0.171499     0.447214
3        0.171499     0.353553
5        0.182574     0.353553
8        0.171499     0.333333
9        0.171499     0.333333
…             …           …
1031166  0.174078     0.377964
1031168  0.174078     0.447214
1031169  0.174078     0.377964
1031170  0.174078     0.377964
1031171  0.174078     0.316228

[343213 rows x 2 columns]
```

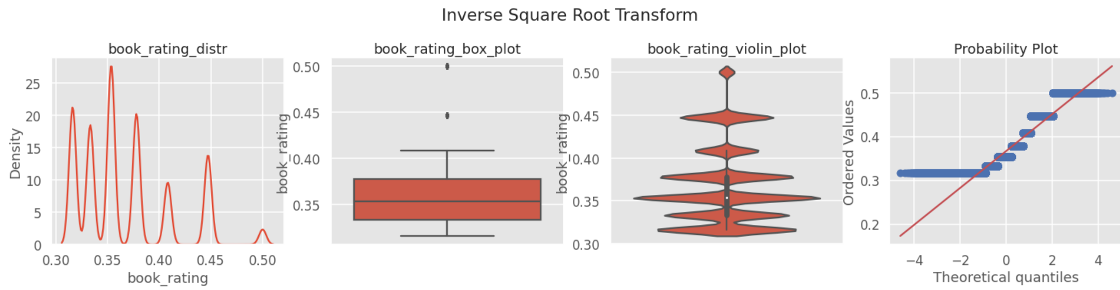
[286]:
```
trans_dset[['age', 'book_rating']].describe().T
```

[286]:
```
                 count      mean       std  ...       50%       75%       max
age           343213.0  0.173505  0.020453  ...  0.171499  0.179605  0.242536
book_rating   343213.0  0.366663  0.044961  ...  0.353553  0.377964  0.500000

[2 rows x 8 columns]
```

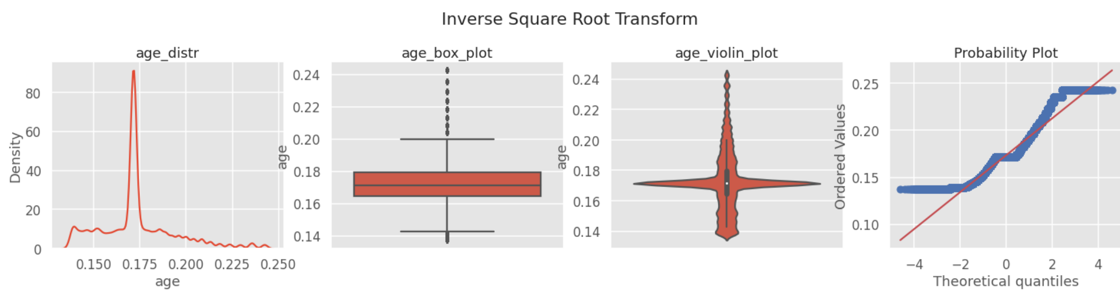
[288]:
```
get_skewness(trans_dset, ['book_rating', 'age'])
```

[288]:
```
             skewness  kurtosis
book_rating  0.966190  0.242251
age          0.824243  1.218674
```

[289]:
```
plot_univariate(dataset=trans_dset, column_name='book_rating',␣
↪suptitle='Inverse Square Root Transform')
```



[290]:
```
plot_univariate(dataset=trans_dset, column_name='age', suptitle='Inverse Square␣
↪Root Transform')
```



1.6 Exploratory Data Analysis

A lot of implicit EDA has already been done above, for univariate variables, now we will go on with trying to make some meaning of other attributes of the dataset, such as `book_title`, `book_author` and `year_of_publication`

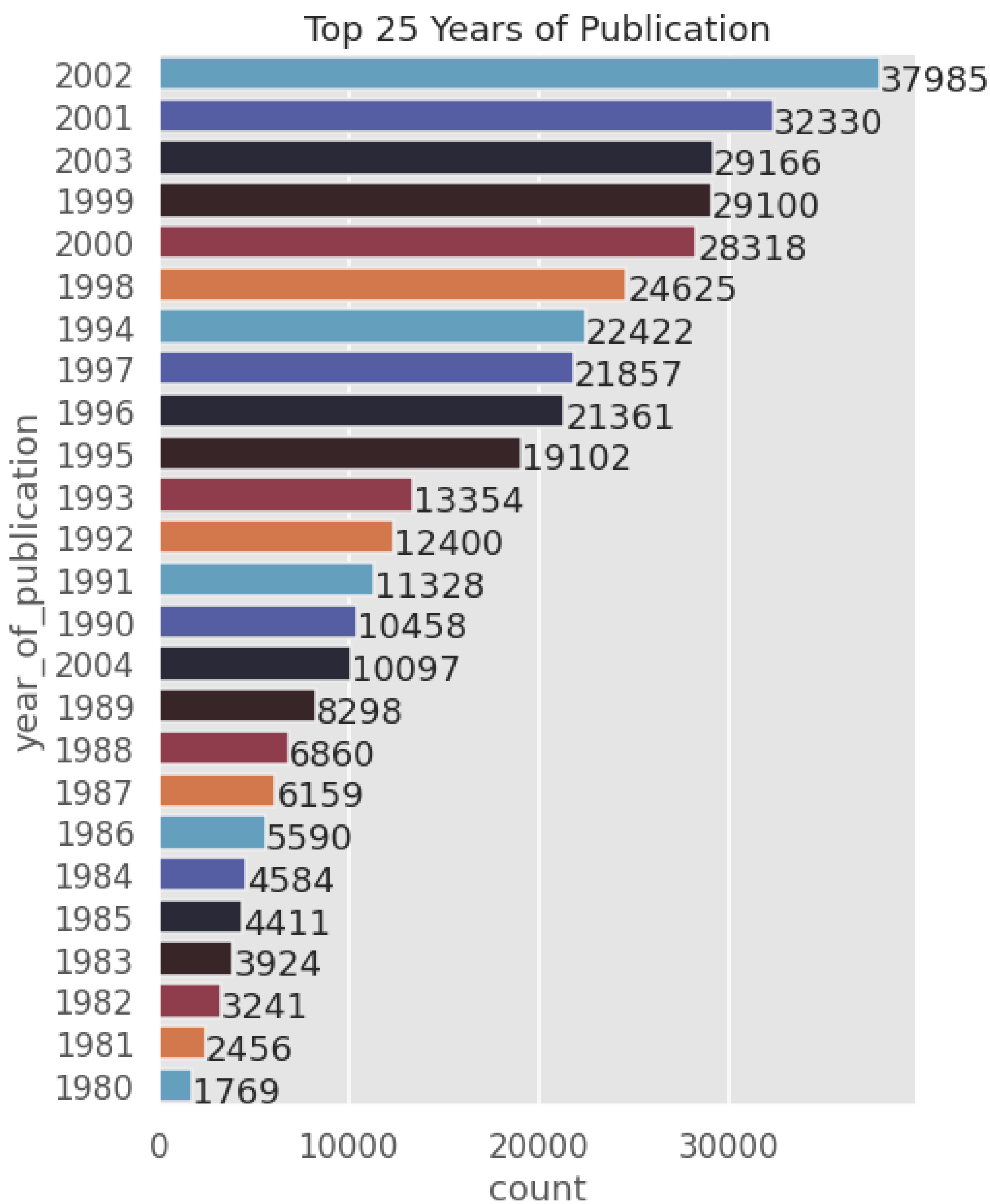
[ ]:
```
# to plot values in barplot, https://stackoverflow.com/a/56780852
def show_values_on_bars(axs, h_v="v", space=0.4):
    def _show_on_single_plot(ax):
        if h_v == "v":
            for p in ax.patches:
                _x = p.get_x() + p.get_width() / 2
                _y = p.get_y() + p.get_height()
                value = int(p.get_height())
                ax.text(_x, _y, value, ha="center")
        elif h_v == "h":
            for p in ax.patches:
                _x = p.get_x() + p.get_width() + float(space)
                _y = p.get_y() + p.get_height()
                value = int(p.get_width())
                ax.text(_x, _y, value, ha="left")

    if isinstance(axs, np.ndarray):
        for idx, ax in np.ndenumerate(axs):
            _show_on_single_plot(ax)
    else:
        _show_on_single_plot(axs)
```

1.6.1 Top 25 Years of Publication

here we try to explore what were the years with huge amount of book publication, this is done by simply counting the number of book in a given year

[ ]:
```
eda = dataset['year_of_publication'].copy().value_counts().head(25).
↪reset_index()
eda.columns = ['year_of_publication', 'count']
eda = eda.sort_values(by=['count'], ascending=False)
```

[ ]:
```
plt.figure(figsize=(7, 10))
splot = sns.barplot(x='count', y='year_of_publication', data=eda,␣
↪order=eda['year_of_publication'], orient='h', palette=palette)
show_values_on_bars(splot, h_v="h")
plt.title('Top 25 Years of Publication')
plt.show()
```



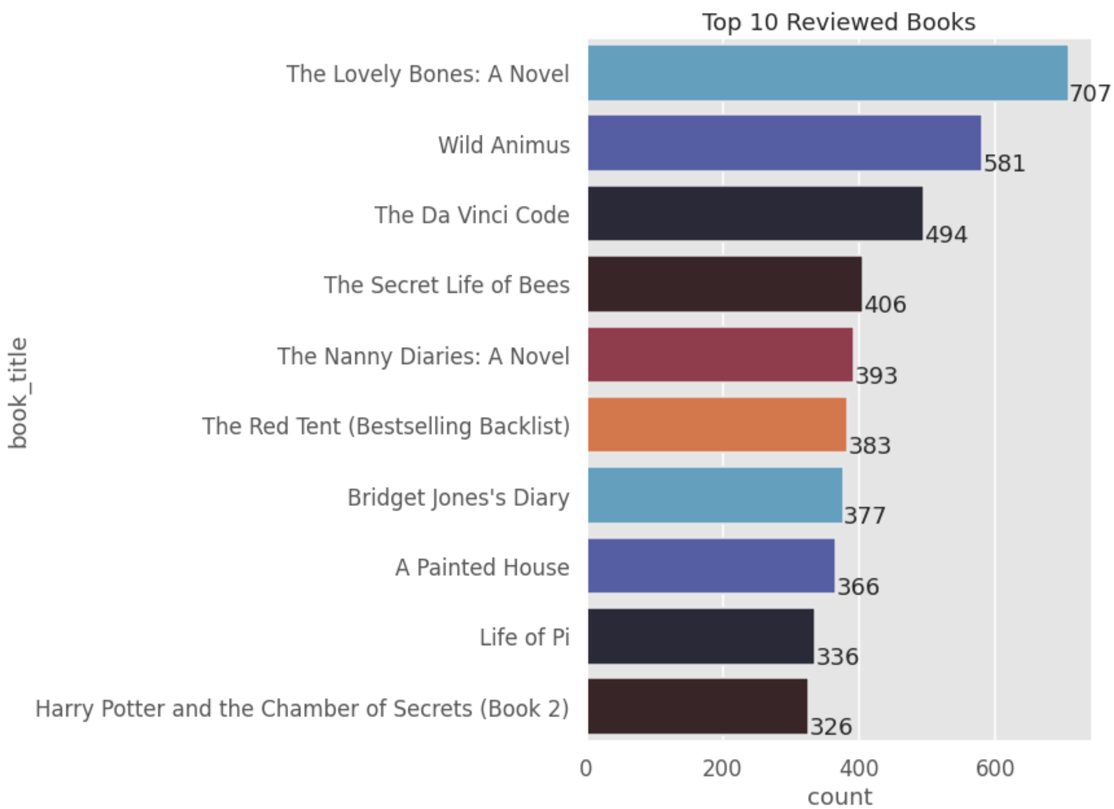
Turns out 2002 was the when highest number of books were published, i.e. 37985

1.6.2 Top 10 Revewed Books

Here we try to find the Books which were reviewed the most !

```
[ ]: eda = dataset['book_title'].value_counts().head(10).reset_index()
     eda.columns = ['book_title', 'count']
```

```
[ ]: plt.figure(figsize=(7, 10))
     splot = sns.barplot(x='count', y='book_title', data=eda,␣
      ↪order=eda['book_title'], orient='h', palette=palette)
     show_values_on_bars(splot, h_v="h")
     plt.title('Top 10 Reviewed Books')
     plt.show()
```



`The Lovely Bone: A Novel` turns out to be most reviewed !

1.6.3 Top 25 Avg. Rated Books

If i want to buy something, one thing i try is to sort the values based on average rating, we try to do similar things for our books dataset, we find the highest reviewed books and then calculate their average rating and sort them again

```
[ ]: rating_count = dataset['book_title'].value_counts().head(25).reset_index().
      ↪sort_values(by='book_title').reset_index(drop=True)
     rating_count.columns = ['book_title', 'rating_count']
     rating_count.head(5)
```

```
[ ]:                             book_title  rating_count
     0  The Girls' Guide to Hunting and Fishing           259
     1                             The Testament           261
     2                                  Timeline           263
     3                    The Catcher in the Rye           265
     4                     To Kill a Mockingbird           267
```

```
[ ]: rating_sum = dataset[dataset['book_title'].isin(rating_count['book_title'])].
      ↪groupby(['book_title'])['book_rating'].sum().reset_index().
      ↪sort_values(by=['book_title'])
     rating_sum.columns = ['book_title', 'rating_sum']
     rating_sum.head(5)
```

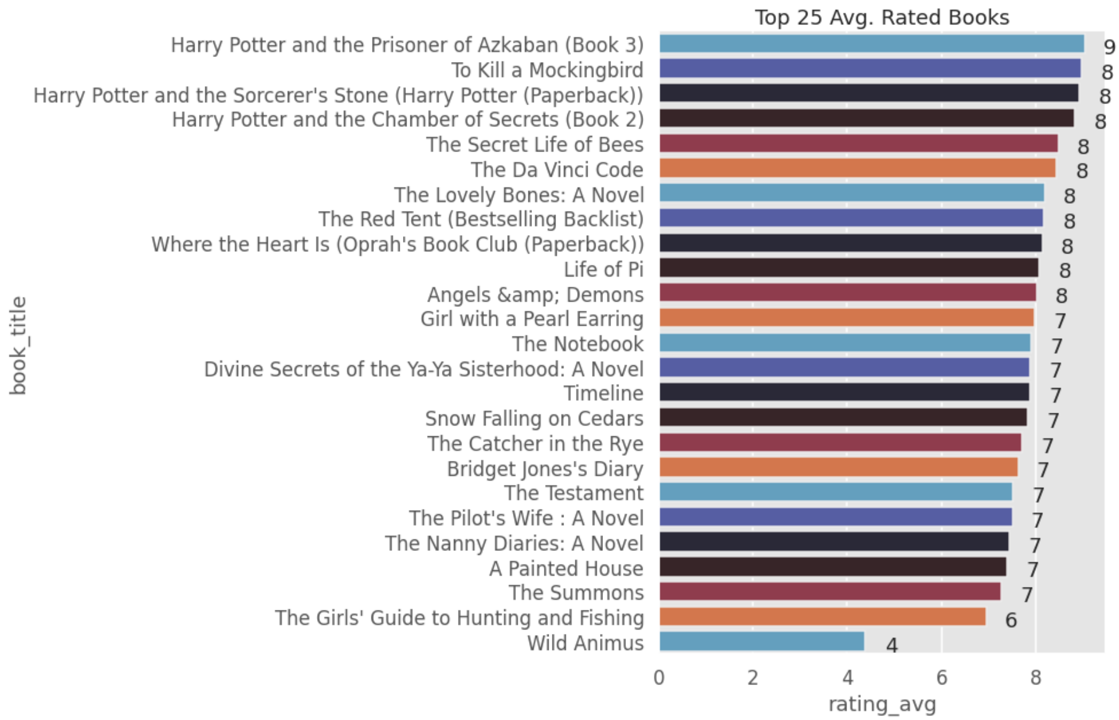
```
[ ]:                                     book_title  rating_sum
     0                             A Painted House      2708.0
     1                         Angels &amp; Demons      2485.0
     2                       Bridget Jones's Diary      2875.0
     3  Divine Secrets of the Ya-Ya Sisterhood: A Novel      2544.0
     4                     Girl with a Pearl Earring      2219.0
```

```
[ ]: avg_rating = pd.merge(rating_count, rating_sum, on='book_title')
     avg_rating['rating_avg'] = avg_rating['rating_sum'] / avg_rating['rating_count']
     avg_rating = avg_rating.sort_values(by='rating_avg', ascending=False).
      ↪reset_index(drop=True)
```

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

```
[ ]:                                         book_title  …  rating_avg
     0  Harry Potter and the Prisoner of Azkaban (Book 3)  …    9.043321
     1                              To Kill a Mockingbird  …    8.977528
     2  Harry Potter and the Sorcerer's Stone (Harry P…  …    8.936508
     3   Harry Potter and the Chamber of Secrets (Book 2)  …    8.840491
     4                            The Secret Life of Bees  …    8.477833

     [5 rows x 4 columns]
```

```
[ ]: plt.figure(figsize=(7, 10))
     splot = sns.barplot(x='rating_avg', y='book_title', data=avg_rating,␣
      ↪order=avg_rating['book_title'], orient='h', palette=palette)
     show_values_on_bars(splot, h_v="h")
     plt.title('Top 25 Avg. Rated Books')
     plt.show()
```



And it turns out `Harry Potter and the Prisoner of Azkaban (Book 3)` is average `Rated 9` !

1.6.4 Top 10 Reviewed Authors

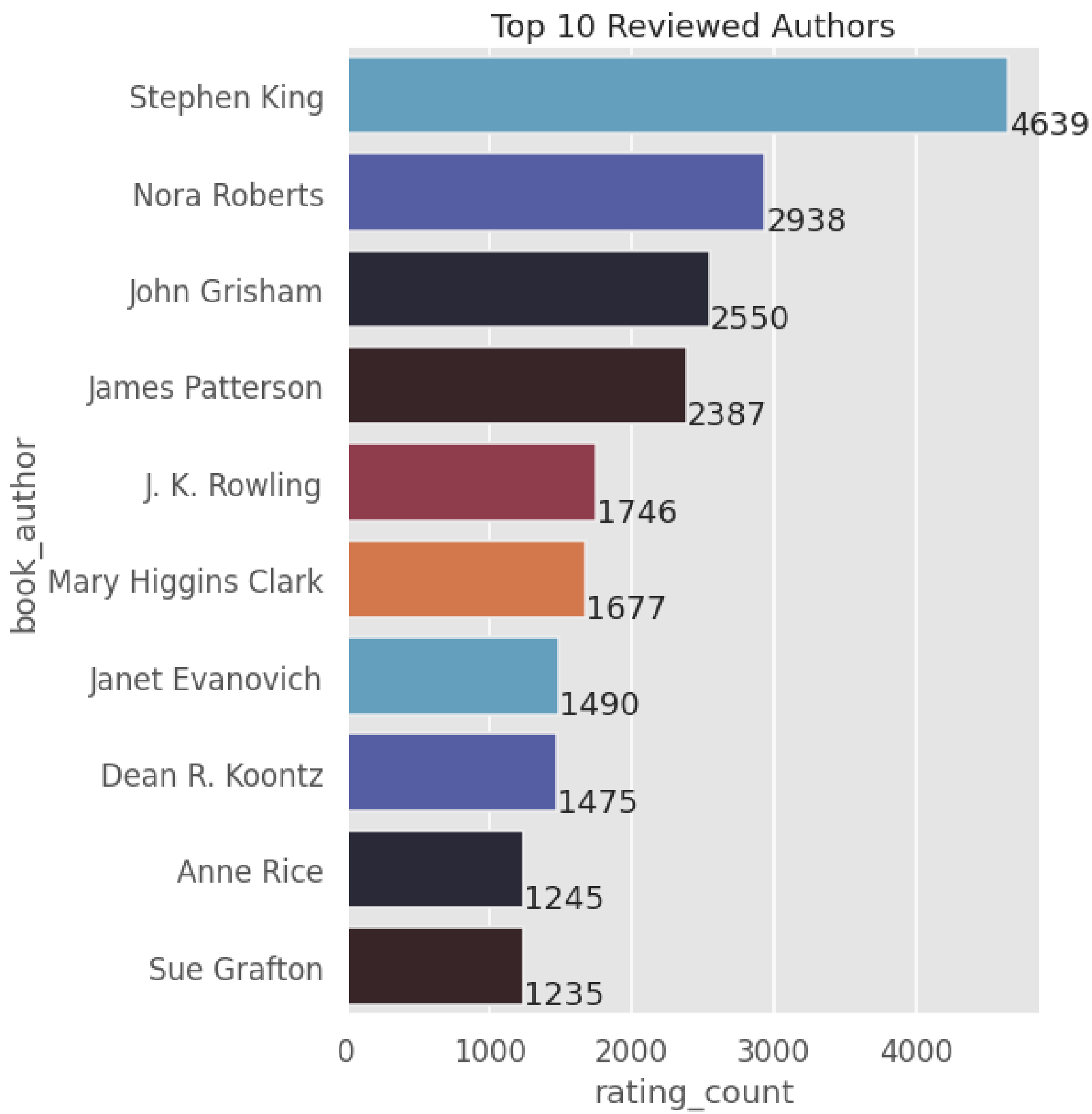
When buying books, author are a major selling point as well, a famous author has more likelihood to be bought again if he publishes a new book

[ ]:
```
author_count = dataset['book_author'].value_counts().head(10).reset_index()
author_count.columns = ['book_author', 'rating_count']
author_count.head(5)
```

[ ]:

	book_author	rating_count
0	Stephen King	4639
1	Nora Roberts	2938
2	John Grisham	2550
3	James Patterson	2387
4	J. K. Rowling	1746

[ ]:
```
plt.figure(figsize=(7, 10))
splot = sns.barplot(x='rating_count', y='book_author', data=author_count,␣
 ↪order=author_count['book_author'], orient='h', palette=palette)
show_values_on_bars(splot, h_v="h")
plt.title('Top 10 Reviewed Authors')
plt.show()
```



`Stephen King` books are the most reviewed !

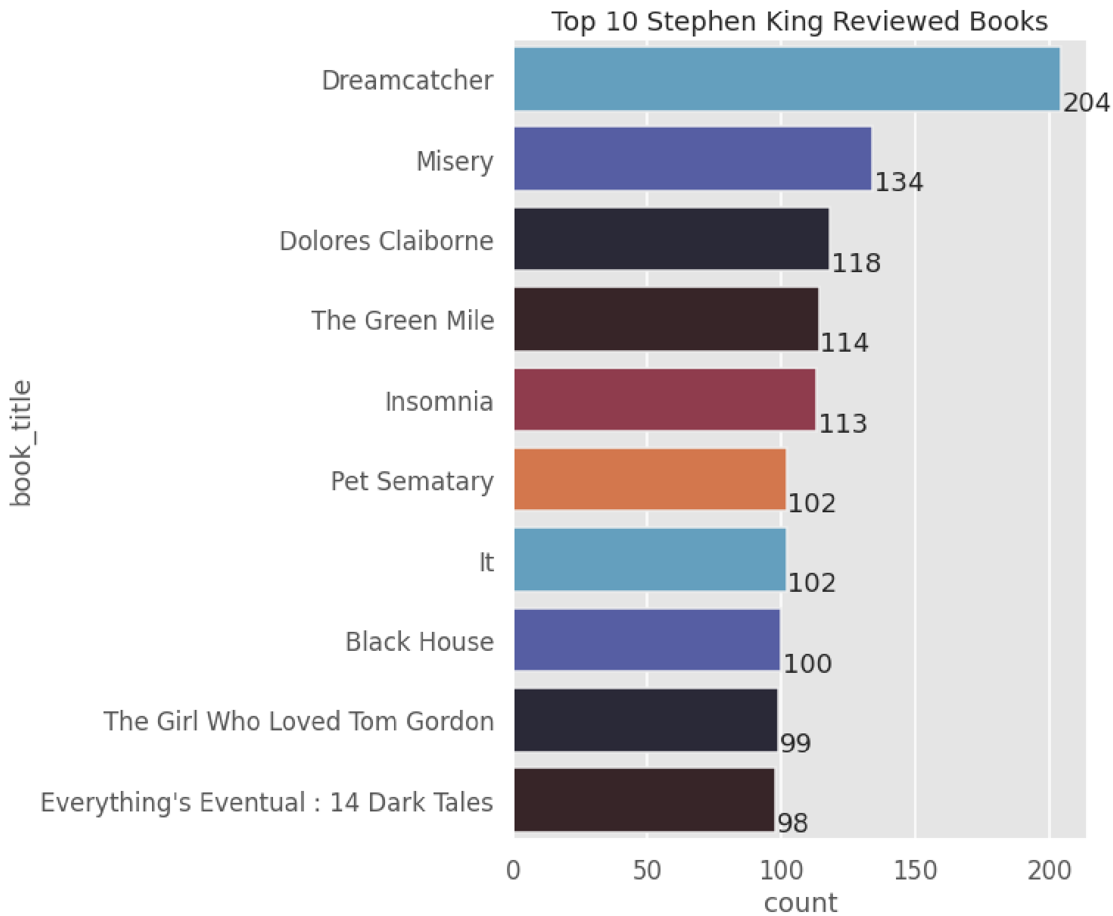
1.6.5 Analysing `Stephen King` Books

Since `Stephen King` is one of the popular authors, we can dig more about it, starting with his most reviewed books !

[ ]:
```
stephen_king = dataset[dataset['book_author'] == 'Stephen King']
```

[ ]:
```
eda = stephen_king['book_title'].value_counts().head(10).reset_index()
eda.columns = ['book_title', 'count']
```

[ ]:
```
plt.figure(figsize=(7, 10))
splot = sns.barplot(x='count', y='book_title', data=eda,␣
 ↪order=eda['book_title'], orient='h', palette=palette)
show_values_on_bars(splot, h_v="h")
plt.title('Top 10 Stephen King Reviewed Books')
plt.show()
```



It turns out Stepen Kings'a book `Dreamcatcher` is the most reviewed book of his ! which was reviewed 204 times !

Now we can do the same for his books as we did before, i.e. calculate the average ratings of his books and sort them to see which book of Stephen King was the most average rated

[ ]:
```
rating_count = stephen_king['book_title'].value_counts().head(10).reset_index().
 ↪sort_values(by='book_title').reset_index(drop=True)
rating_count.columns = ['book_title', 'rating_count']
rating_sum = stephen_king[stephen_king['book_title'].
 ↪isin(rating_count['book_title'])].groupby(['book_title'])['book_rating'].
 ↪sum().reset_index().sort_values(by=['book_title'])
```

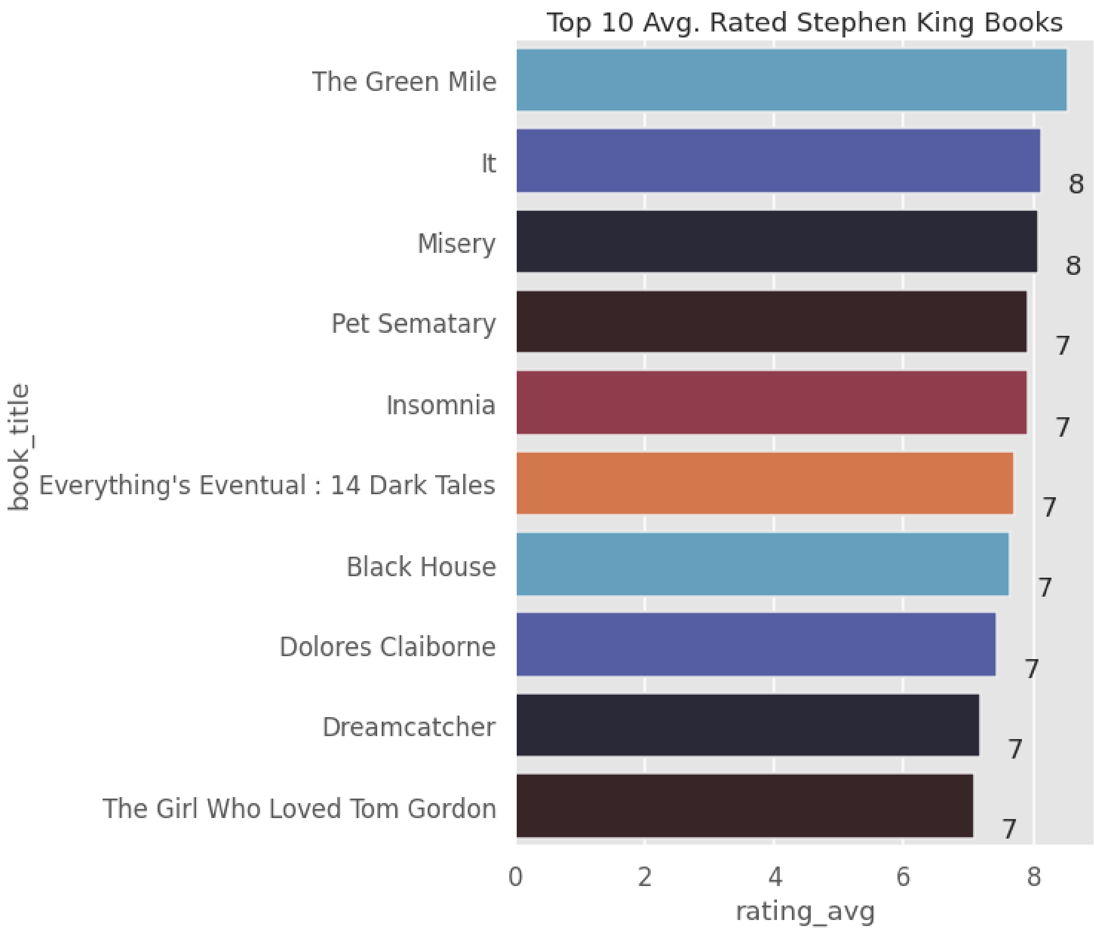
```
rating_sum.columns = ['book_title', 'rating_sum']
avg_rating = pd.merge(rating_count, rating_sum, on='book_title')
avg_rating['rating_avg'] = avg_rating['rating_sum'] / avg_rating['rating_count']
avg_rating = avg_rating.sort_values(by='rating_avg', ascending=False).
 ↪reset_index(drop=True)
```

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

[ ]:

	book_title	rating_count	rating_sum	rating_avg
0	The Green Mile	114	972.0	8.526316
1	It	102	829.0	8.127451
2	Misery	134	1082.0	8.074627
3	Pet Sematary	102	808.0	7.921569
4	Insomnia	113	895.0	7.920354

[ ]:
```
plt.figure(figsize=(7, 10))
splot = sns.barplot(x='rating_avg', y='book_title', data=avg_rating,␣
 ↪order=avg_rating['book_title'], orient='h', palette=palette)
show_values_on_bars(splot, h_v="h")
plt.title('Top 10 Avg. Rated Stephen King Books')
plt.show()
```



Green Mile was Stephen King's top avg. rated book with a avg. rating of 8

2 Bibliography

1. Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
2. G.E.P. Box and D.R. Cox, "An Analysis of Transformations", Journal of the Royal Statistical Society B, 26, 211-252 (1964).
3. http://fmwww.bc.edu/repec/bocode/t/transint.html

3 Convert this Notebook to PDF

[214]:
```
%%capture
! apt update
! apt install texlive-xetex texlive-fonts-recommended␣
↪texlive-generic-recommended
```

[215]:
```
import subprocess
import shlex
```

Convert to PDF

[ ]:
```
s = subprocess.Popen(shlex.split(
    f'jupyter nbconvert /content/BookReview_EDA_Satyajit_Ghana.ipynb --to pdf'
    ), shell = False, stdout = subprocess.PIPE, stderr = subprocess.PIPE)
s.wait()
s.stdout.read()
```

Convert to LaTeX

[219]:
```
s = subprocess.Popen(shlex.split(
    f'jupyter nbconvert /content/BookReview_EDA_Satyajit_Ghana.ipynb --to latex'
    ), shell = False, stdout = subprocess.PIPE, stderr = subprocess.PIPE)
s.wait()
s.stdout.read()
```

[219]:
```
b''
```

[220]:
```
! zip -r BookReview_EDA_Satyajit_Ghana.zip BookReview_EDA_Satyajit_Ghana_files␣
↪BookReview_EDA_Satyajit_Ghana.tex
```

```
  adding: BookReview_EDA_Satyajit_Ghana_files/ (stored 0%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_95_0.png
(deflated 4%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_77_0.png
(deflated 7%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_195_0.png
(deflated 13%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_126_0.png
(deflated 6%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_104_0.png
(deflated 7%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_114_0.png
(deflated 4%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_162_0.png
(deflated 6%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_182_0.png
```

```
(deflated 14%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_155_0.png
(deflated 5%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_169_0.png
(deflated 6%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_105_0.png
(deflated 4%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_63_0.png
(deflated 6%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_134_0.png
(deflated 5%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_83_0.png
(deflated 4%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_177_0.png
(deflated 13%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_76_0.png
(deflated 5%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_206_0.png
(deflated 14%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_113_0.png
(deflated 7%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_94_0.png
(deflated 7%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_93_0.png
(deflated 5%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_141_0.png
(deflated 6%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_201_0.png
(deflated 13%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_190_0.png
(deflated 12%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_112_0.png
```

```
(deflated 5%)
  adding:
BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_103_0.png
(deflated 5%)
  adding: BookReview_EDA_Satyajit_Ghana.tex (deflated 43%)
```

[ ]:

```

```