Book Review - An Exploratory Data Analysis and Data Transformation Notebook

Author: Satyajit Ghana

In [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]
```

Out[1]:

```
'BX-CSV-Dump.zip'
```

In [2]:

```
!! unzip BX-CSV-Dump.zip
```

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

Book Crossing EDA

In [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")
```

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`

In [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
```

Out[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 × 3 columns

parse the datatypes properly

In [5]:

```
users.dtypes
```

Out[5]:

```
user_id       int64
location     object
age         float64
dtype: object
```

A quick look at the numeric attributes of the users table

In [6]:

```
users.describe().T
```

Out[6]:

	count	mean	std	min	25%	50%	75%	max
user_id	278858.0	139429.500000	80499.515020	1.0	69715.25	139429.5	209143.75	278858.0
age	168096.0	34.751434	14.428097	0.0	24.00	32.0	44.00	244.0

Data Inconsistency

We notice that the max age is 244, and min age is 0, age cannot be 244 I 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

In [212]:

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

In [8]:

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

In [9]:

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

Out[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

In [10]:

```
users.isna().sum()
```

Out[10]:

```
user_id     0
location    0
age         0
dtype: int64
```

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

In [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
```

Out[11]:

	isbn	book_title	book_author	year_of_publication	publisher	img_s	
0	0195153448	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press	http://images.amazon.com/images/P/0195153448.0...	http://images.amazon.com
1	0002005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	http://images.amazon.com/images/P/0002005018.0...	http://images.amazon.com
2	0060973129	Decision in Normandy	Carlo D'Este	1991	HarperPerennial	http://images.amazon.com/images/P/0060973129.0...	http://images.amazon.com
3	0374157065	Flu: The Story of the Great Influenza Pandemic...	Gina Bari Kolata	1999	Farrar Straus Giroux	http://images.amazon.com/images/P/0374157065.0...	http://images.amazon.com
4	0393045218	The Mummies of Urumchi	E. J. W. Barber	1999	W. W. Norton &amp; Company	http://images.amazon.com/images/P/0393045218.0...	http://images.amazon.com
...	...	...	...	...	...	...	...
271374	0440400988	There's a Bat in Bunk Five	Paula Danziger	1988	Random House Childrens Pub (Mm)	http://images.amazon.com/images/P/0440400988.0...	http://images.amazon.com
271375	0525447644	From One to One Hundred	Teri Sloat	1991	Dutton Books	http://images.amazon.com/images/P/0525447644.0...	http://images.amazon.com
271376	006008667X	Lily Dale : The True Story of the Town that Ta...	Christine Wicker	2004	HarperSanFrancisco	http://images.amazon.com/images/P/006008667X.0...	http://images.amazon.com
271377	0192126040	Republic (World's Classics)	Plato	1996	Oxford University Press	http://images.amazon.com/images/P/0192126040.0...	http://images.amazon.com
271378	0767409752	A Guided Tour of Rene Descartes' Meditations o...	Christopher Biffle	2000	McGraw-Hill Humanities/Social Sciences/Languages	http://images.amazon.com/images/P/0767409752.0...	http://images.amazon.com

271379 rows × 8 columns

parse the data types properly

In [12]:

```
books.dtypes
```

Out[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

In [13]:

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

`year_of_publication` should be a integer

In [14]:

```
books['year_of_publication'] = pd.to_numeric(books['year_of_publication'], errors='coerce')
```

In [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

In [16]:

```
books.isna().sum()
```

Out[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

In [17]:

```
books = books.dropna()
```

In [18]:

```
books.describe().T
```

Out[18]:

	count	mean	std	min	25%	50%	75%	max
year_of_publication	271376.0	1993.692427	8.248715	1376.0	1989.0	1995.0	2000.0	2008.0

Read the ratings dataset as usual

In [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
```

Out[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 × 3 columns

In [20]:

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

In [21]:

```
ratings.dtypes
```

Out[21]:

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

check for any `NA` values

In [22]:

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

Out[22]:

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

There are some inconsistent values, which we will fix later

In [23]:

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

Out[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

In [24]:

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

In [25]:

```
dataset
```

Out[25]:

	user_id	location	age	isbn	book_rating	book_title	book_author	year_of_publication	publisher
0	2	stockton, california, usa	18	0195153448	0	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press
1	8	timmins, ontario, canada	34	0002005018	5	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada
2	11400	ottawa, ontario, canada	49	0002005018	0	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada
3	11676	n/a, n/a, n/a	34	0002005018	8	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada
4	41385	sudbury, ontario, canada	34	0002005018	0	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada
...	...	...	...	...	...	...	...	...	...
1031167	276851	dallas, texas, usa	33	0743203763	0	As Hogan Said . . . : The 389 Best Things Anyo...	Randy Voorhees	2000	Simon &amp; Schuster
1031168	276851	dallas, texas, usa	33	0767907566	5	All Elevations Unknown: An Adventure in the He...	Sam Lightner	2001	Broadway Books
1031169	276851	dallas, texas, usa	33	0884159221	7	Why stop?: A guide to Texas historical roadsid...	Claude Dooley	1985	Lone Star Books
1031170	276851	dallas, texas, usa	33	0912333022	7	The Are You Being Served? Stories: 'Camping In...	Jeremy Lloyd	1997	Kqed Books
1031171	276851	dallas, texas, usa	33	1569661057	10	Dallas Street Map Guide and Directory, 2000 Ed...	Mapsco	1999	American Map Corporation

1031172 rows × 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.

In [26]:

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

In [27]:

```
dataset['city'] = location['city'] ; dataset['state'] = location['state'] ; dataset['country'] = location['country']
```

In [28]:

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

In [29]:

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

Out[29]:

	count	mean	std	min	25%	50%	75%	max
user_id	1031172	140594	80524	2	70415	141210	211426	278854
age	1031172	36	10	5	31	34	41	100
book_rating	1031172	2	3	0	0	0	7	10
year_of_publication	1031172	1995	7	1376	1992	1997	2001	2008

checking for `NA` values

In [30]:

```
dataset.isna().sum()
```

Out[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 !

In [31]:

```
dataset.shape
```

Out[31]:

```
(1031172, 11)
```

In [32]:

```
dataset.dtypes
```

Out[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 !

In [33]:

```
dataset.head(5)
```

Out[33]:

	user_id	age	isbn	book_rating	book_title	book_author	year_of_publication	publisher	city	state	country
0	2	18	0195153448	0	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press	stockton	california	usa
1	8	34	0002005018	5	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	timmins	ontario	canada
2	11400	49	0002005018	0	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	ottawa	ontario	canada
3	11676	34	0002005018	8	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	n/a	n/a	n/a
4	41385	34	0002005018	0	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	sudbury	ontario	canada

In [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
```

In [35]:

```
cleaned_data = dataset.copy()
```

In [204]:

```
# dataset = cleaned_data.copy()
```

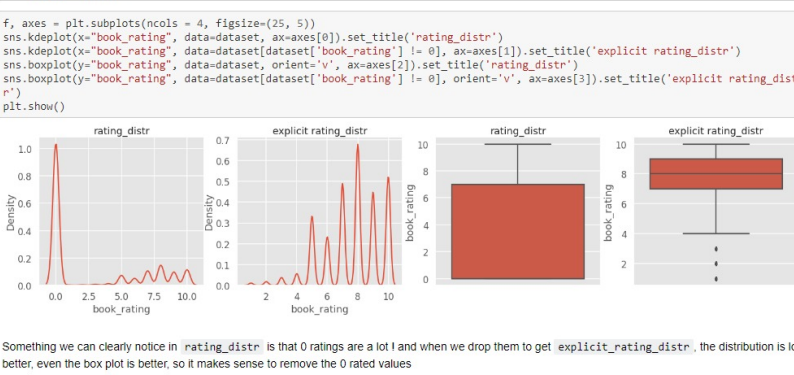
Analyzing the Feature Space

In [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
```

In [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_dist
r')
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 ?

In [207]:

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

Out[207]:

	skewness	kurtosis
age	0.832497	1.343472
book_rating	0.752445	-1.214894

In [208]:

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

Out[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` ?

In [209]:

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

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

In [210]:

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

Out[210]:

	skewness	kurtosis
age	0.859061	1.644862
book_rating	-0.661263	0.120288

In [211]:

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

Out[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

Univariate Analysis of Attributes

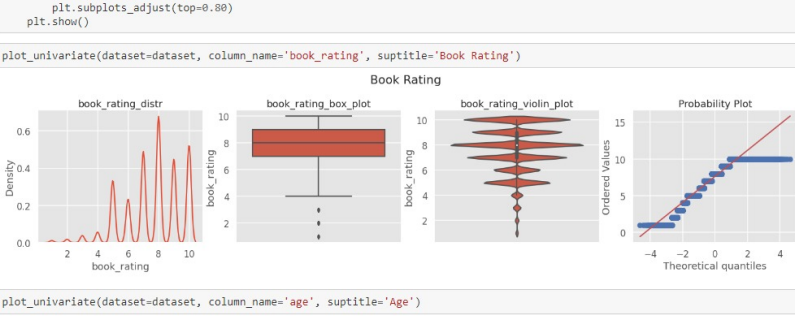
Here we plot the Distribution, Box Plot, Violin Plot and QQ Plot for the numeric attributes `age` and `book_rating`

In [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()
```

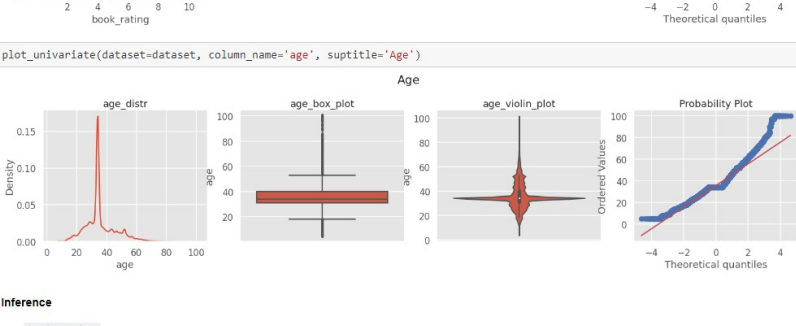
In [134]:

```
plot_univariate(dataset=dataset, column_name='book_rating', suptitle='Book Rating')
```



In [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 outliers, and the qq plot also

Outlier Analysis and removing them

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

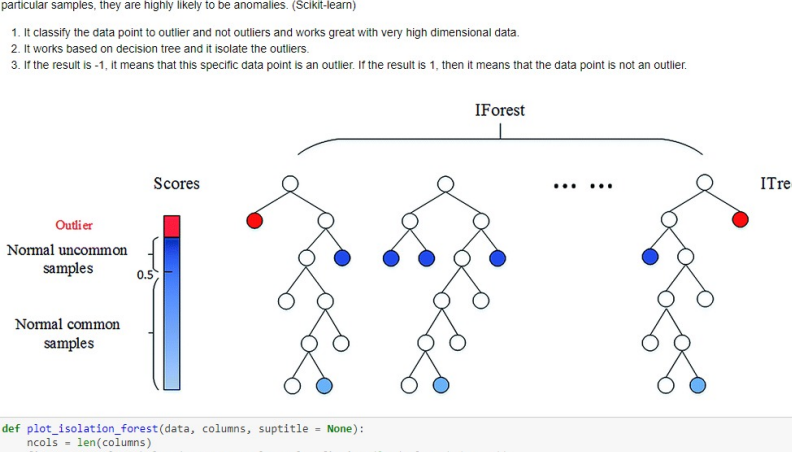
The IsolationForest 'isolates' observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature.

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.

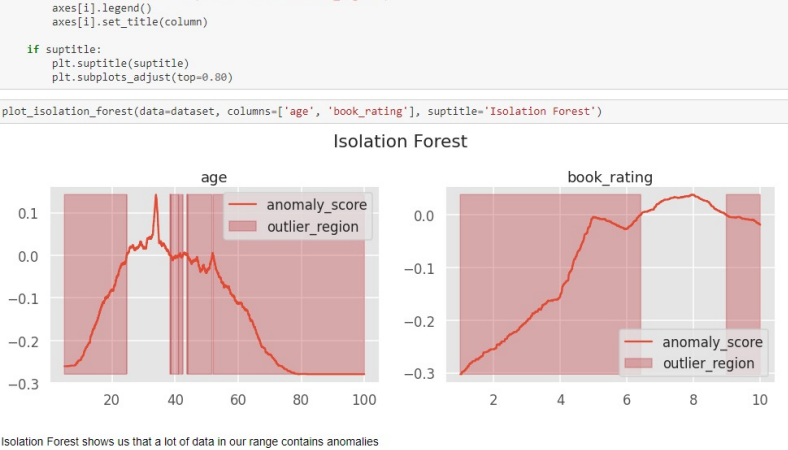


In [136]:

```
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)
```

In [137]:

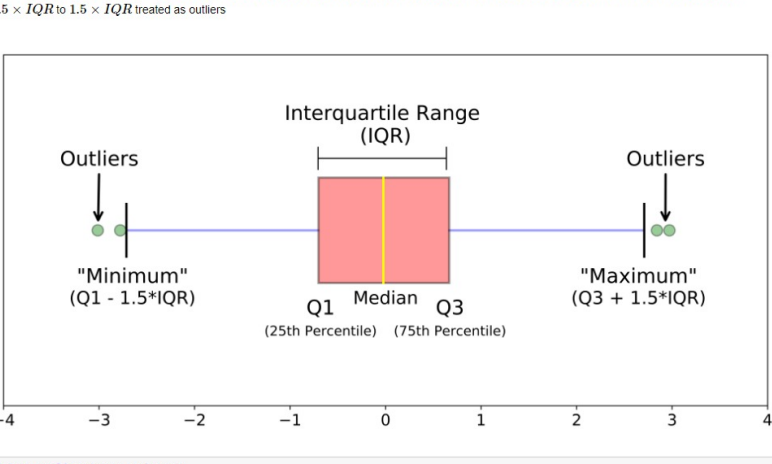
```
plot_isolation_forest(data=dataset, columns=['age', 'book_rating'], suptitle='Isolation Forest')
```



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

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



In [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
```

In [139]:

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

In [140]:

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

Out[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

In [141]:

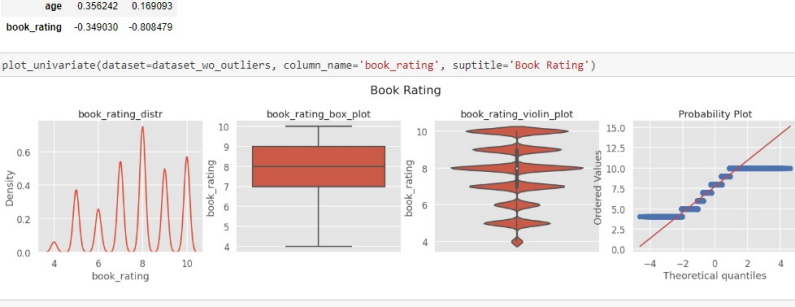
```
get_skewness(dataset_wo_outliers, ['age', 'book_rating'])
```

Out[141]:

	skewness	kurtosis
age	0.356242	0.169093
book_rating	-0.349030	-0.808479

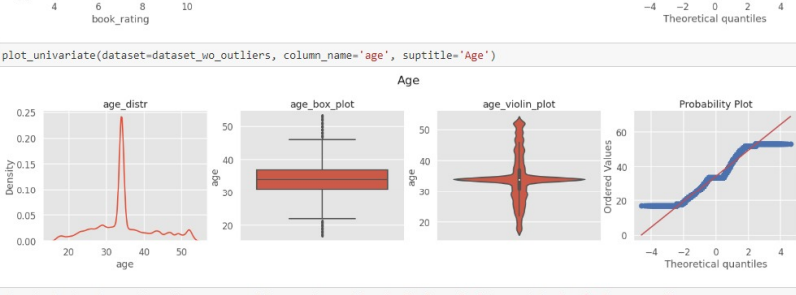
In [142]:

```
plot_univariate(dataset=dataset_wo_outliers, column_name='book_rating', suptitle='Book Rating')
```



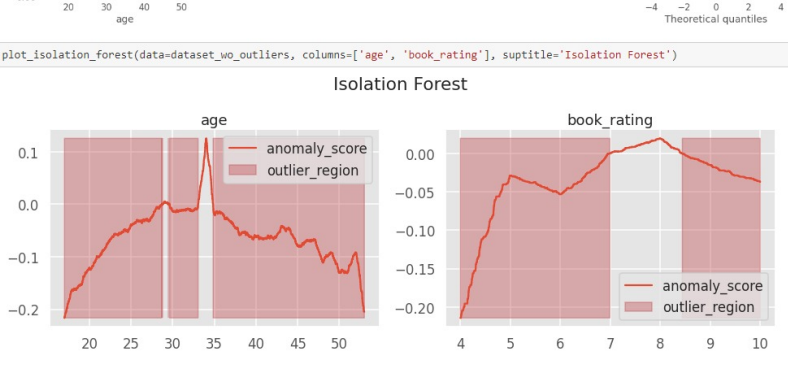
In [143]:

```
plot_univariate(dataset=dataset_wo_outliers, column_name='age', suptitle='Age')
```



In [144]:

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



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.

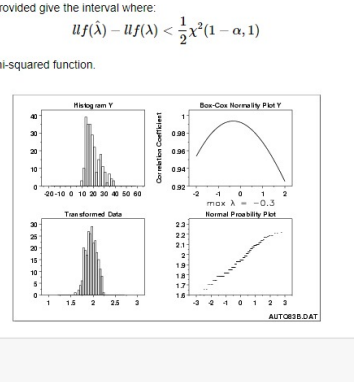
The BoxCox transformation is given by

```
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.



In [145]:

```
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
```

In [146]:

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

In [147]:

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

Out[147]:

	count	mean	std	min	25%	50%	75%	max
age	383849.0	7.096895	1.012567	2.19784	6.693741	7.099771	7.591456	11.579612
book_rating	383849.0	22.629185	9.147474	0.00000	18.422366	23.642124	29.426302	35.763058

In [148]:

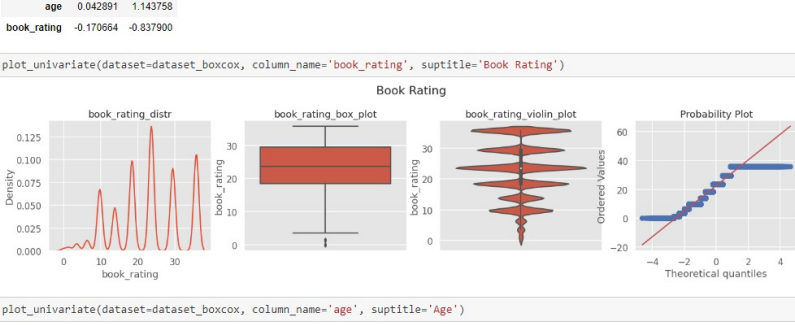
```
get_skewness(dataset_boxcox, ['age', 'book_rating'])
```

Out[148]:

	skewness	kurtosis
age	0.042881	1.143758
book_rating	-0.170664	-0.837900

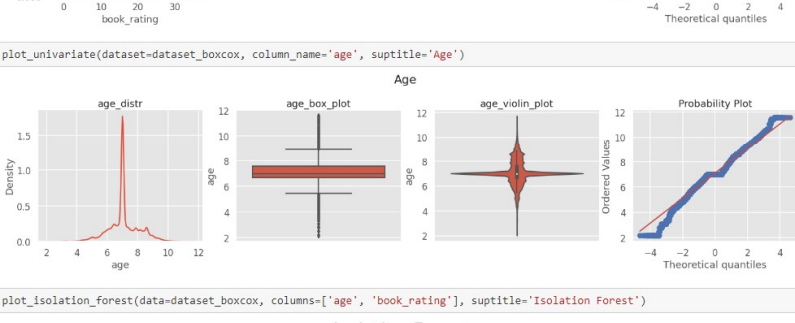
In [149]:

```
plot_univariate(dataset=dataset_boxcox, column_name='book_rating', suptitle='Book Rating')
```



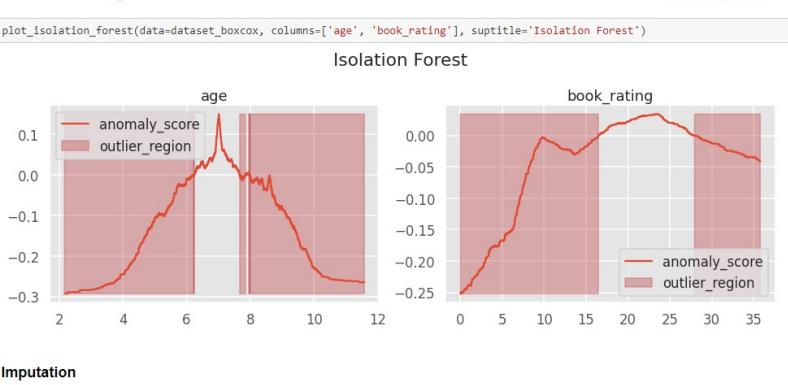
In [150]:

```
plot_univariate(dataset=dataset_boxcox, column_name='age', suptitle='Age')
```



In [151]:

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



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)

In [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
```

In [153]:

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

In [154]:

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

Out[154]:

	count	mean	std	min	25%	50%	75%	max
age	383849.0	35.856535	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

In [155]:

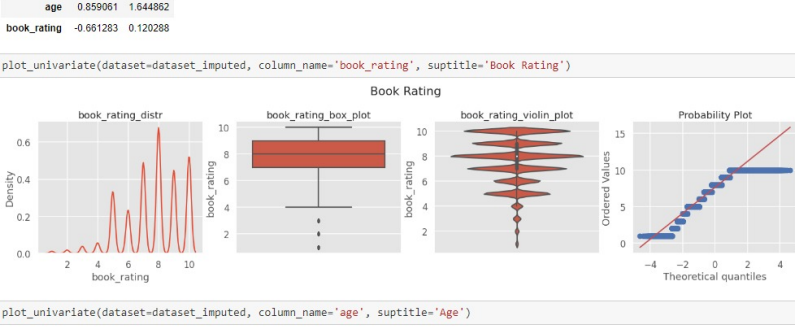
```
get_skewness(dataset_imputed, ['age', 'book_rating'])
```

Out[155]:

	skewness	kurtosis
age	0.859061	1.644862
book_rating	-0.661263	0.120288

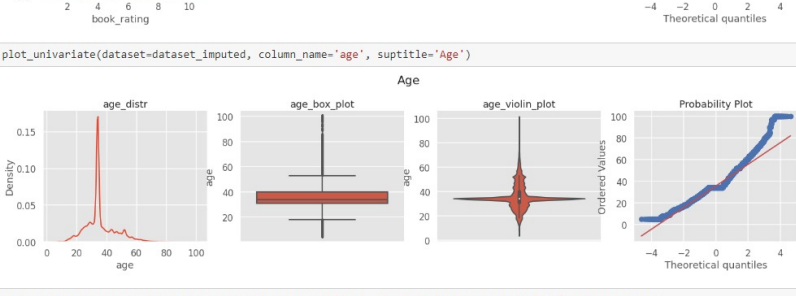
In [156]:

```
plot_univariate(dataset=dataset_imputed, column_name='book_rating', suptitle='Book Rating')
```



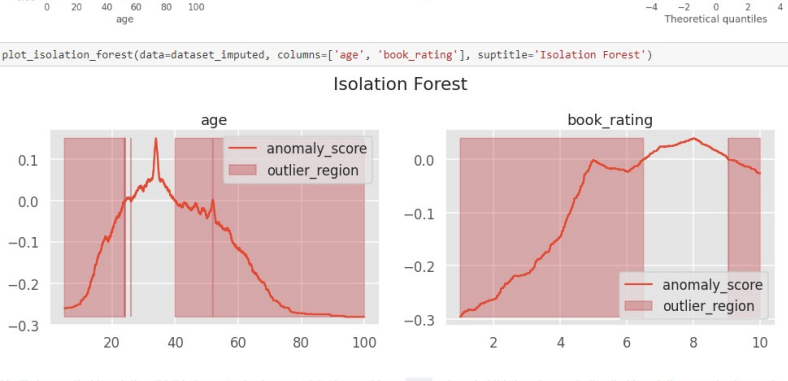
In [157]:

```
plot_univariate(dataset=dataset_imputed, column_name='age', suptitle='Age')
```



In [158]:

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

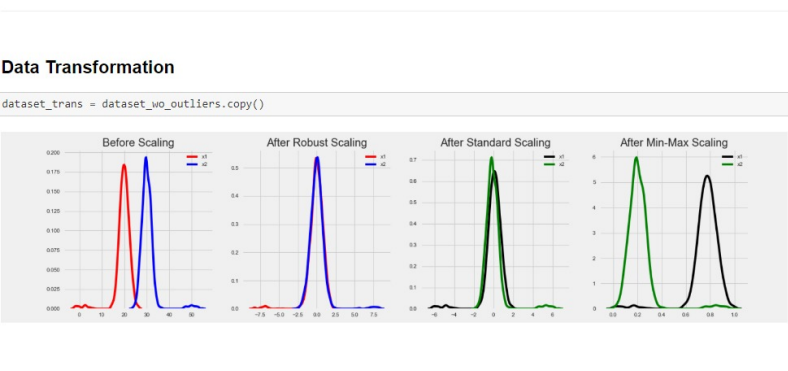


You'll observe that imputation didn't help us much, since our data does not have `nan` values, but this is a demonstration that imputation can also be used

Data Transformation

In [159]:

```
dataset_trans = dataset_wo_outliers.copy()
```



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

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

In [160]:

```
scaler = MinMaxScaler()
scaled = scaler.fit_transform(dataset_trans['book_rating'].values.reshape(-1, 1)).reshape(-1)
```

In [161]:

```
scaled_rating = pd.DataFrame(data=scaled, columns=['book_rating'])
scaled_rating
```

Out[161]:

	book_rating
0	0.166667
1	0.666667
2	0.666667
3	0.833333
4	0.833333
...	...
343208	0.500000
343209	0.166667
343210	0.500000
343211	0.500000
343212	1.000000

343213 rows × 1 columns

In [162]:

```
scaled_rating.describe().T
```

Out[162]:

	count	mean	std	min	25%	50%	75%	max
book_rating	343213.0	0.623696	0.278131	0.0	0.5	0.666667	0.833333	1.0

In [163]:

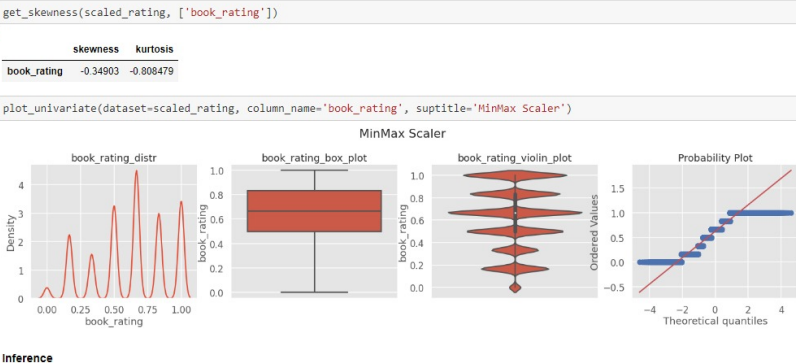
```
get_skewness(scaled_rating, ['book_rating'])
```

Out[163]:

	skewness	kurtosis
book_rating	-0.34903	-0.808479

In [90]:

```
plot_univariate(dataset=scaled_rating, column_name='book_rating', suptitle='MinMax Scaler')
```



Inference

- The skewness has now becomes `-0.34`

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)

In [164]:

```
scaler = StandardScaler()
scaled = scaler.fit_transform(dataset_trans['book_rating'].values.reshape(-1, 1)).reshape(-1)
```

In [165]:

```
scaled_rating = pd.DataFrame(data=scaled, columns=['book_rating'])
scaled_rating
```

Out[165]:

	book_rating
0	-1.643219
1	0.154487
2	0.154487
3	0.753736
4	0.753736
...	...
343208	-0.444741
343209	-1.643219
343210	-0.444741
343211	-0.444741
343212	1.352974

343213 rows × 1 columns

In [166]:

```
scaled_rating.describe().T
```

Out[166]:

	count	mean	std	min	25%	50%	75%	max
book_rating	343213.0	9.450267e-16	1.000001	-2.242457	-0.444741	0.154487	0.753736	1.352974

In [167]:

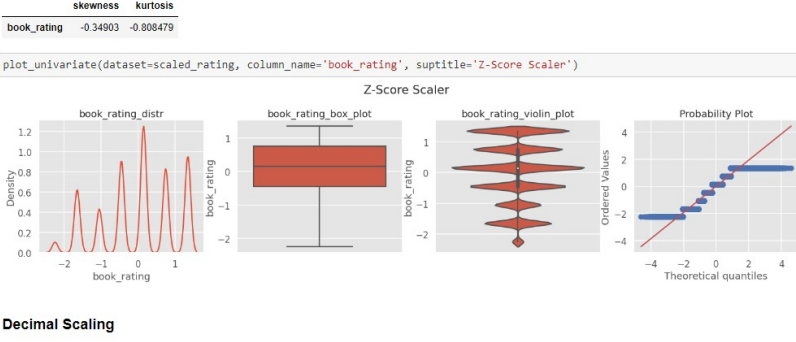
```
get_skewness(scaled_rating, ['book_rating'])
```

Out[167]:

	skewness	kurtosis
book_rating	-0.34903	-0.808479

In [168]:

```
plot_univariate(dataset=scaled_rating, column_name='book_rating', suptitle='Z-Score Scaler')
```



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:

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

In [169]:

```
p = dataset_trans['book_rating'].max()
q = len(str(abs(p)))
scaled = dataset_trans['book_rating'].values / 10 ** q
```

In [170]:

```
scaled_rating = pd.DataFrame(data=scaled, columns=['book_rating'])
scaled_rating
```

Out[170]:

	book_rating
0	0.05
1	0.08
2	0.08
3	0.09
4	0.09
...	...
343208	0.07
343209	0.05
343210	0.07
343211	0.07
343212	0.10

343213 rows × 1 columns

In [171]:

```
scaled_rating.describe().T
```

Out[171]:

	count	mean	std	min	25%	50%	75%	max
book_rating	343213.0	0.077422	0.016688	0.04	0.07	0.08	0.09	0.1

In [172]:

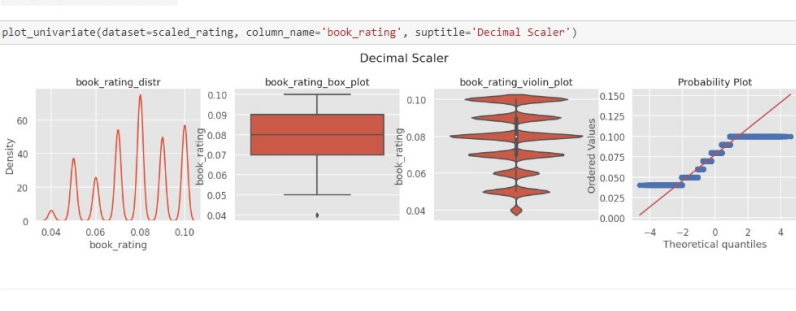
```
get_skewness(scaled_rating, ['book_rating'])
```

Out[172]:

	skewness	kurtosis
book_rating	-0.34903	-0.808479

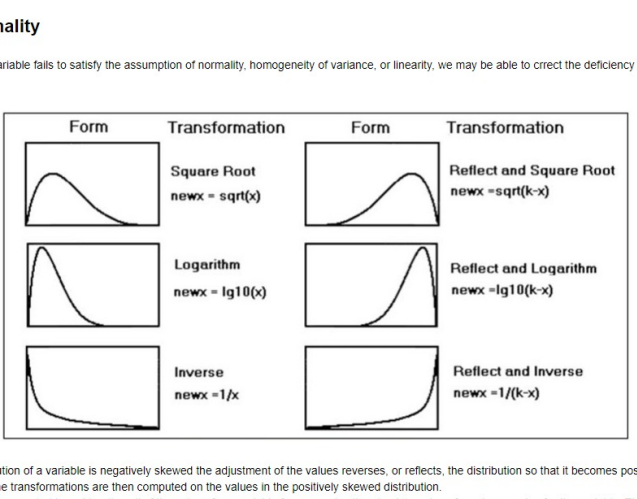
In [173]:

```
plot_univariate(dataset=scaled_rating, column_name='book_rating', suptitle='Decimal Scaler')
```



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 skewed. 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.;

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.

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

In [183]:

```
transformed = np.log(dataset_trans['book_rating'].astype(np.float32))
```

In [184]:

```
trans_rating = pd.DataFrame(data=transformed, columns=['book_rating'])
trans_rating
```

Out[184]:

	book_rating
1	1.609438
3	2.079442
5	2.079442
8	2.197225
9	2.197225
...	...
1031166	1.945910
1031168	1.609438
1031169	1.945910
1031170	1.945910
1031171	2.302585

343213 rows × 1 columns

In [185]:

```
trans_rating.describe().T
```

Out[185]:

	count	mean	std	min	25%	50%	75%	max
book_rating	343213.0	2.021872	0.234512	1.386294	1.94591	2.079442	2.197225	2.302585

In [186]:

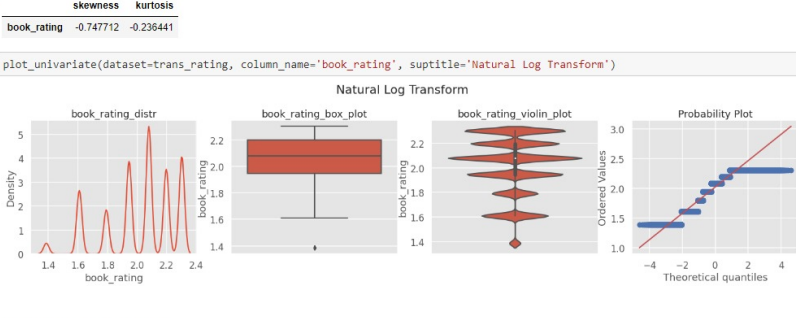
```
get_skewness(trans_rating, ['book_rating'])
```

Out[186]:

	skewness	kurtosis
book_rating	-0.747712	-0.236441

In [187]:

```
plot_univariate(dataset=trans_rating, column_name='book_rating', suptitle='Natural Log Transform')
```



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.

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

In [188]:

```
transformed = np.sqrt(dataset_trans['book_rating'].astype(np.float32))
```

In [189]:

```
trans_rating = pd.DataFrame(data=transformed, columns=['book_rating'])
trans_rating
```

Out[189]:

	book_rating
1	2.236068
3	2.828427
5	2.828427
8	3.000000
9	3.000000
...	...
1031166	2.645751
1031168	2.236068
1031169	2.645751
1031170	2.645751
1031171	3.162278

343213 rows × 1 columns

In [190]:

```
trans_rating.describe().T
```

Out[190]:

	count	mean	std	min	25%	50%	75%	max
book_rating	343213.0	2.765608	0.310675	2.0	2.645751	2.828427	3.0	3.162278

In [191]:

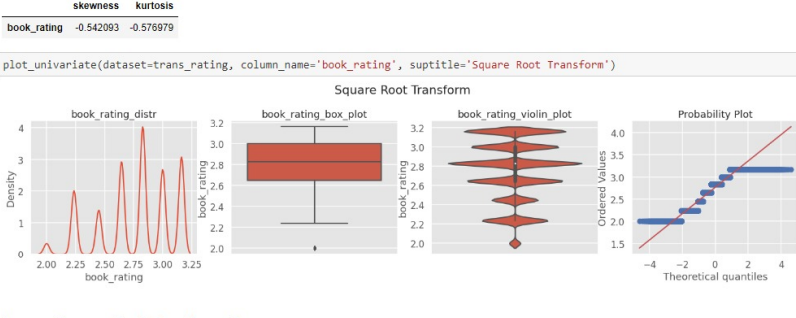
```
get_skewness(trans_rating, ['book_rating'])
```

Out[191]:

	skewness	kurtosis
book_rating	-0.542083	-0.575979

In [192]:

```
plot_univariate(dataset=trans_rating, column_name='book_rating', suptitle='Square Root Transform')
```



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.

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

In [193]:

```
transformed = np.power(dataset_trans['book_rating'].astype(np.float32), -1/2)
```

In [194]:

```
trans_rating = pd.DataFrame(data=transformed, columns=['book_rating'])
trans_rating
```

Out[194]:

	book_rating
1	0.447214
3	0.353553
5	0.353553
8	0.333333
9	0.333333
...	...
1031166	0.377964
1031168	0.447214
1031169	0.377964
1031170	0.377964
1031171	0.316228

343213 rows × 1 columns

In [195]:

```
trans_rating.describe().T
```

Out[195]:

	count	mean	std	min	25%	50%	75%	max
book_rating	343213.0	0.366663	0.044961	0.316228	0.333333	0.353553	0.377964	0.5

In [196]:

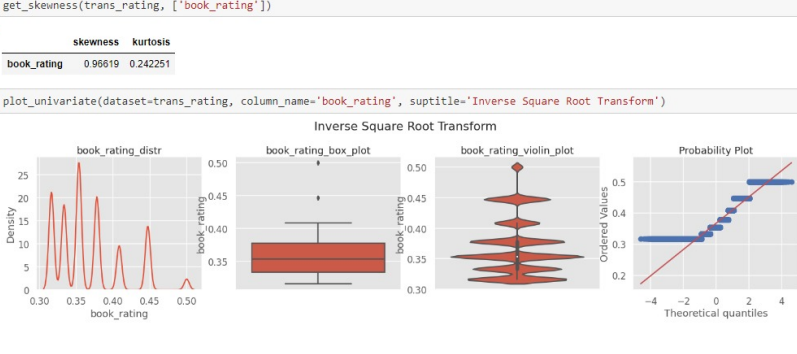
```
get_skewness(trans_rating, ['book_rating'])
```

Out[196]:

	skewness	kurtosis
book_rating	0.96619	0.242251

In [197]:

```
plot_univariate(dataset=trans_rating, column_name='book_rating', suptitle='Inverse Square Root Transform')
```



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`

In [ ]:

```
# to plot values in barplot, https://stackoverflow.com/q/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)
```

Top 25 Years of Publication

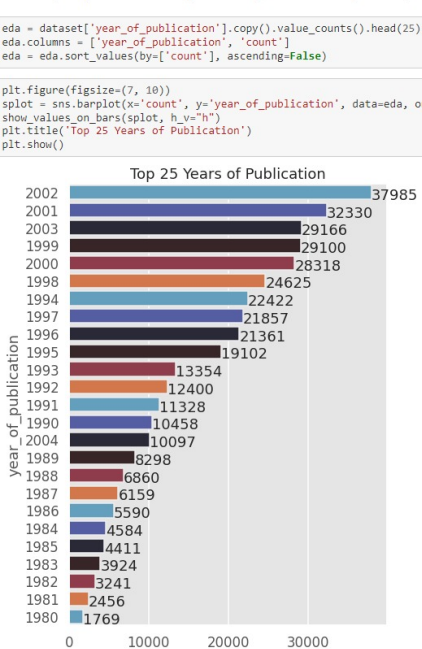
here we try to explore what were the years with huge amount of book publication, this is done by simply counting the number of book in a given year

In [ ]:

```
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)
```

In [ ]:

```
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`

Top 10 Revewed Books

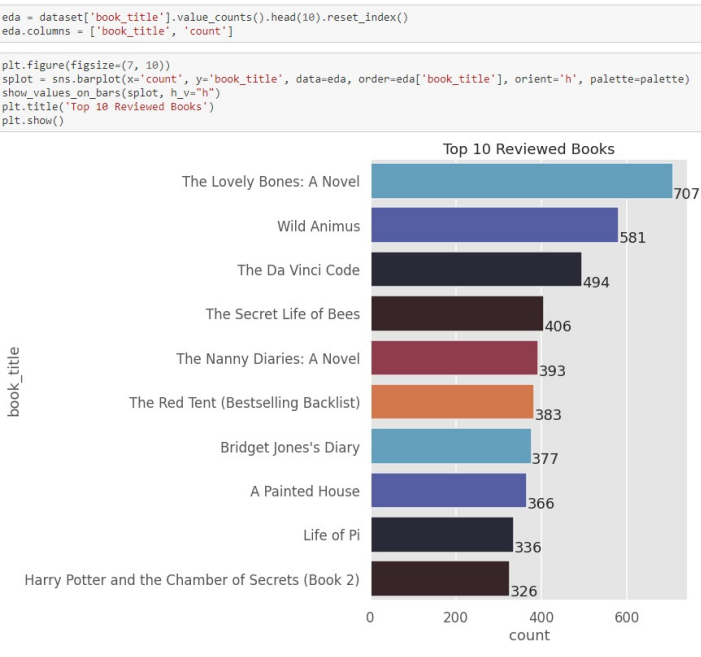
Here we try to find the Books which were reviewed the most !

In [ ]:

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

In [ ]:

```
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 !

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

In [ ]:

```
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)
```

Out[ ]:

	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

In [ ]:

```
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)
```

Out[ ]:

	book_title	rating_sum
0	A Painted House	2798.0
1	Angels &amp; Demons	2465.0
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

In [ ]:

```
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)
```

In [ ]:

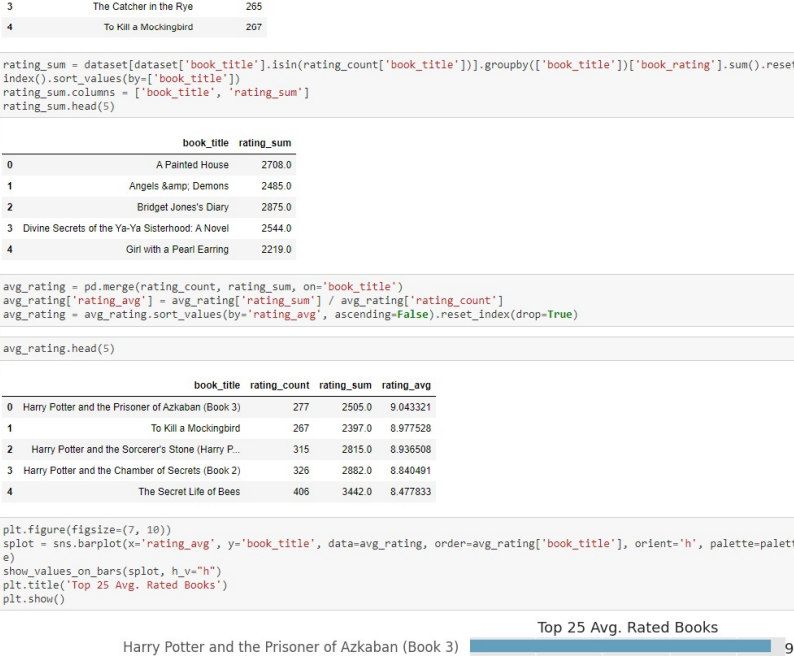
```
avg_rating.head(5)
```

Out[ ]:

	book_title	rating_count	rating_sum	rating_avg
0	Harry Potter and the Prisoner of Azkaban (Book 3)	277	2505.0	9.043321
1	To Kill a Mockingbird	267	2397.0	8.977528
2	Harry Potter and the Sorcerer's Stone (Harry P...	315	2815.0	8.936508
3	Harry Potter and the Chamber of Secrets (Book 2)	326	2882.0	8.840491
4	The Secret Life of Bees	406	3442.0	8.477833

In [ ]:

```
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=palett
e)
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` !

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

In [ ]:

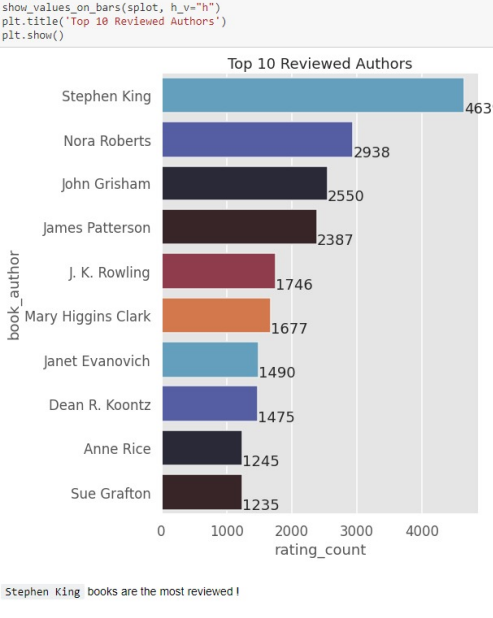
```
author_count = dataset['book_author'].value_counts().head(10).reset_index()
author_count.columns = ['book_author', 'rating_count']
author_count.head(5)
```

Out[ ]:

	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

In [ ]:

```
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 !

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 !

In [ ]:

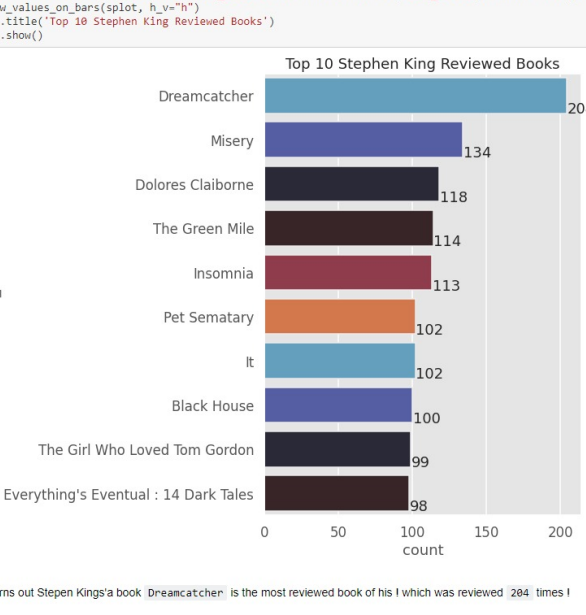
```
stephen_king = dataset[dataset['book_author'] == 'Stephen King']
```

In [ ]:

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

In [ ]:

```
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

In [ ]:

```
rating_count = stephen_king['book_title'].value_counts().head(10).reset_index().sort_values(by='book_title').reset_index(drop=Tr
ue)
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'].su
m().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)
```

In [ ]:

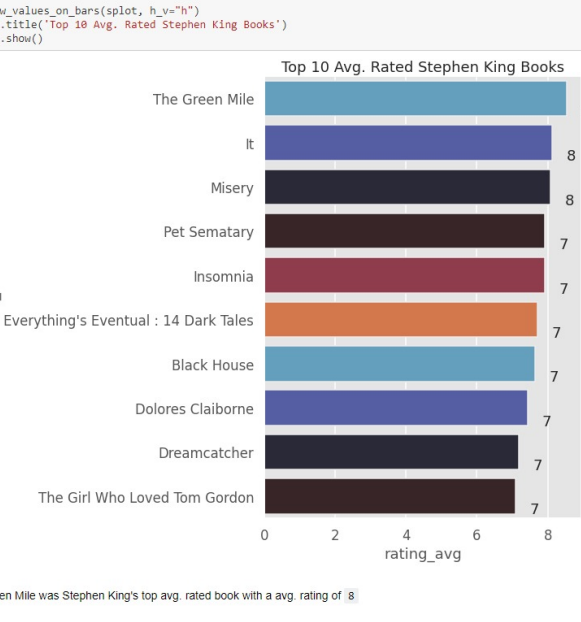
```
avg_rating.head(5)
```

Out[ ]:

	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

In [ ]:

```
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=palett
e)
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`

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

Convert this Notebook to PDF

In [214]:

```
%%capture
!! apt update
!! apt install texlive-xetex texlive-fonts-recommended texlive-generic-recommended
```

In [ ]:

```
import subprocess
import shlex
```

Convert to PDF

In [ ]:

```
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()
```

Out[ ]:

```
b''
```

Convert to $\LaTeX$

In [ ]:

```
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()
```

Out[ ]:

```
b''
```

In [ ]:

```
!! 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 14%)
  adding: BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_73_0.png (deflated 5%)
  adding: BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_67_0.png (deflated 5%)
  adding: BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_49_0.png (deflated 6%)
  adding: BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_106_0.png (deflated 13%)
  adding: BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_61_0.png (deflated 5%)
  adding: BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_84_0.png (deflated 6%)
  adding: BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_54_0.png (deflated 7%)
  adding: BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_89_0.png (deflated 8%)
  adding: BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_93_0.png (deflated 13%)
  adding: BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_79_0.png (deflated 6%)
  adding: BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_102_0.png (deflated 13%)
  adding: BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_100_0.png (deflated 12%)
  adding: BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_53_0.png (deflated 5%)
  adding: BookReview_EDA_Satyajit_Ghana_files/BookReview_EDA_Satyajit_Ghana_109_0.png (deflated 14%)
  adding: BookReview_EDA_Satyajit_Ghana.tex (deflated 88%)
```

In [ ]: