# MDSC-102 (P) End Sem Assessment

**Name**: Vidyasager G R
**Roll No**: 23914

**Assessment Tasks:** Take a Dataset, and perform the following operations.

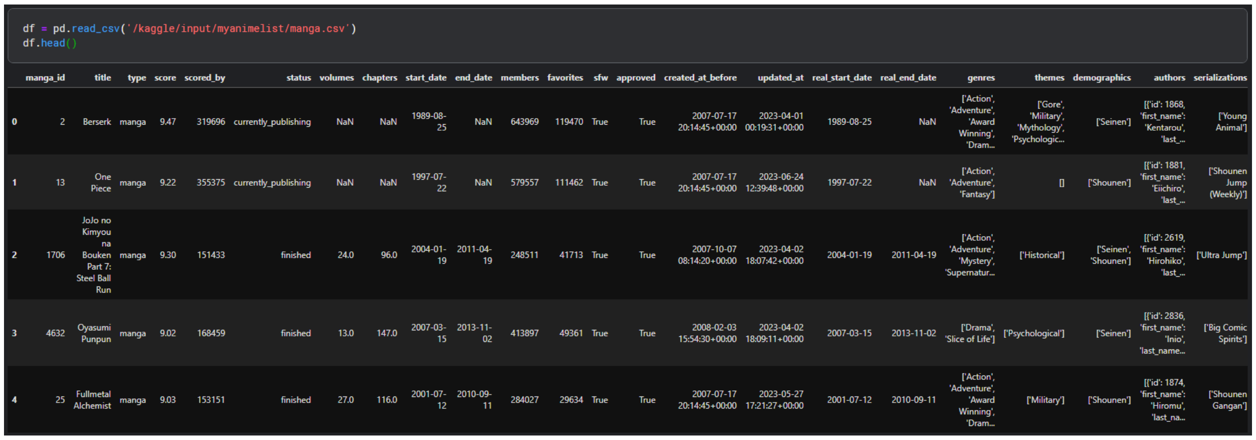
1. Data Cleaning.
2. Summary Statistics (Mean, Mean, Mode, etc)
3. Inferences through Plots.
4. Analyze Features and their Correlation.
5. Take one or more continuous features and check their distribution.
6. Take a parameter of one feature and estimate its confidence interval (for example., Mean of a Feature.)
7. Present an hypothesis for the same parameter and test it using statistical tests that fits it the most.

## Loading Data

First we make a dataframe from the dataset using Pandas. The dataset that I have taken for this task is the Manga Database which is scraped from a website called the 'MyAnimeList'. 'MyAnimeList' holds a huge database of both Anime and Manga (Japanese Books / Comics). It is like iMDB for Anime and Manga.

NOTE: This dataset was scraped and uploaded in Kaggle by a third-person.

pd.read_csv() is used to read the data into a Pandas Dataframe.



```
df = pd.read_csv('/kaggle/input/myanimelist/manga.csv')
df.head()
```

| | manga_id | title | type | score | scored_by | status | volumes | chapters | start_date | end_date | members | favorites | sfw | approved |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | Berserk | manga | 9.47 | 319696 | currently_publishing | NaN | NaN | 1989-08-25 | NaN | 643969 | 119470 | True | True |
| 1 | 13 | One Piece | manga | 9.22 | 355375 | currently_publishing | NaN | NaN | 1997-07-22 | NaN | 579557 | 111462 | True | True |
| 2 | 1706 | JoJo no Kimyou na Bouken Part 7: Steel Ball Run | manga | 9.30 | 151433 | finished | 24.0 | 96.0 | 2004-01-19 | 2011-04-19 | 248511 | 41713 | True | True |
| 3 | 4632 | Oyasumi Punpun | manga | 9.02 | 168459 | finished | 13.0 | 147.0 | 2007-03-15 | 2013-11-02 | 413897 | 49361 | True | True |
| 4 | 25 | Fullmetal Alchemist | manga | 9.03 | 153151 | finished | 27.0 | 116.0 | 2001-07-12 | 2010-09-11 | 284027 | 29634 | True | True |

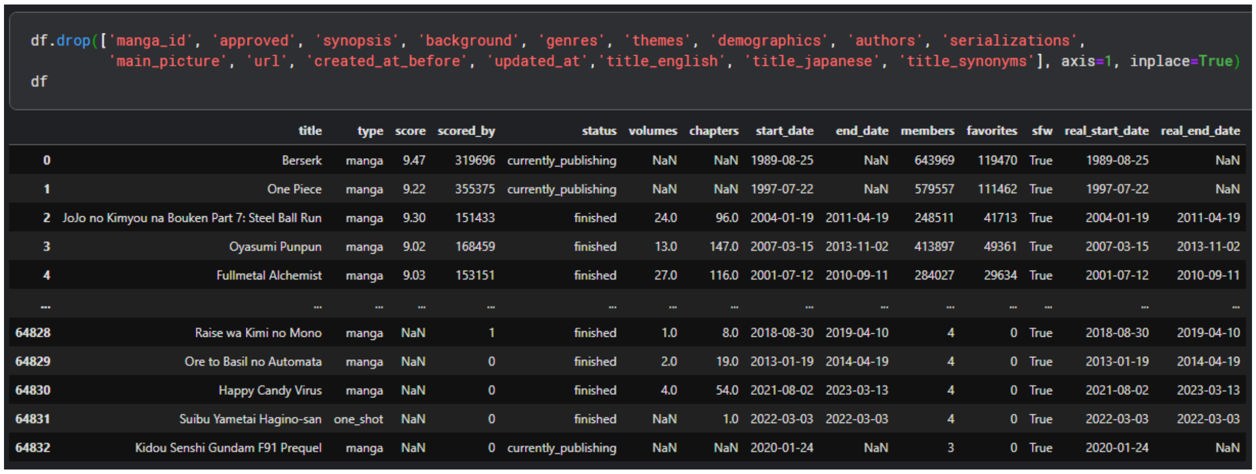
| | created_at_before | updated_at | real_start_date | real_end_date | genres | themes | demographics | authors | serializations |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2007-07-17 20:14:45+00:00 | 2023-04-01 00:19:31+00:00 | 1989-08-25 | NaN | ['Action', 'Adventure', 'Award Winning', 'Dram... | ['Gore', 'Military', 'Mythology', 'Psychologic... | ['Seinen'] | [{'id': 1868, 'first_name': 'Kentarou', 'last_... | ['Young Animal'] |
| 1 | 2007-07-17 20:14:45+00:00 | 2023-06-24 12:39:48+00:00 | 1997-07-22 | NaN | ['Action', 'Adventure', 'Fantasy'] | [] | ['Shounen'] | [{'id': 1881, 'first_name': 'Eiichiro', 'last_... | ['Shounen Jump (Weekly)'] |
| 2 | 2007-10-07 08:14:20+00:00 | 2023-04-02 18:07:42+00:00 | 2004-01-19 | 2011-04-19 | ['Action', 'Adventure', 'Mystery', 'Supernatur... | ['Historical'] | ['Seinen', 'Shounen'] | [{'id': 2619, 'first_name': 'Hirohiko', 'last_... | ['Ultra Jump'] |
| 3 | 2008-02-03 15:54:30+00:00 | 2023-04-02 18:09:11+00:00 | 2007-03-15 | 2013-11-02 | ['Drama', 'Slice of Life'] | ['Psychological'] | ['Seinen'] | [{'id': 2836, 'first_name': 'Inio', 'last_name... | ['Big Comic Spirits'] |
| 4 | 2007-07-17 20:14:45+00:00 | 2023-05-27 17:21:27+00:00 | 2001-07-12 | 2010-09-11 | ['Action', 'Adventure', 'Award Winning', 'Dram... | ['Military'] | ['Shounen'] | [{'id': 1874, 'first_name': 'Hiromu', 'last_na... | ['Shounen Gangan'] |

A Brief Description of the Dataset and its Features:

**manga_id** - MyAnimeList Manga ID.
**title** - Title (rōmaji or english).
**type** - Manga media type.
**score** - MAL Weighted Score.
**scored_by** - Number of users who scored this manga.
**status** - Publishing status.
**volumes** - Number of Volumes.
**chapters** - Number of Chapters.
**start_date** - Publishing Start Date.
**end_date** - Publishing End Date.
**members** - Number of users with this manga in their list.
**favorites** - Number of users who favorited this manga.
**sfw** - Whether it's Safe For Work or it's R18+.
**approved** - Whether it's approved or is yet 'pending approval'.
**created_at_before** - MAL Manga entry created before this date.
**updated_at** - MAL Manga entry latest update date.
**real_start_date** - Some entries only have year or year-month.
**real_end_date** - Some entries only have year or year-month.
**genres** - List of Genres.
**themes** - List of Themes.
**demographics** - List of Demographics.
**authors** - List of Authors (id, first and last name, role).
**serializations** - List of Magazines where it's been published.
**synopsis** - Manga description.
**background** - Manga additional information.
**main_picture** - MAL main picture url.
**url** - MyAnimeList url.
**title_english** - Title in English.
**title_japanese** - Title in Japanese.
**title_synonyms** - List of title synonyms.

From this, I could understand what each feature in the dataset meant and so I could decide which columns/features to drop. Dropping unnecessary features will make analysis a lot easier. So, I decided to drop the following features: **'manga_id', 'approved', 'synopsis', 'background', 'genres', 'themes', 'demographics', 'authors', 'serializations', 'main_picture', 'url', 'created_at_before', 'updated_at','title_english', 'title_japanese', 'title_synonyms'.**

This was done using the **drop()** function provided by Pandas where a list of rows (indices) /columns to be dropped are passed as parameters to the function along with another parameter **'axis'** that specifies whether the function should search entries to be removed by columns or by rows. **(axis=1 is columns and axis=0 is rows)**



```
df.drop(['manga_id', 'approved', 'synopsis', 'background', 'genres', 'themes', 'demographics', 'authors', 'serializations',
         'main_picture', 'url', 'created_at_before', 'updated_at','title_english', 'title_japanese', 'title_synonyms'], axis=1, inplace=True)
df
```

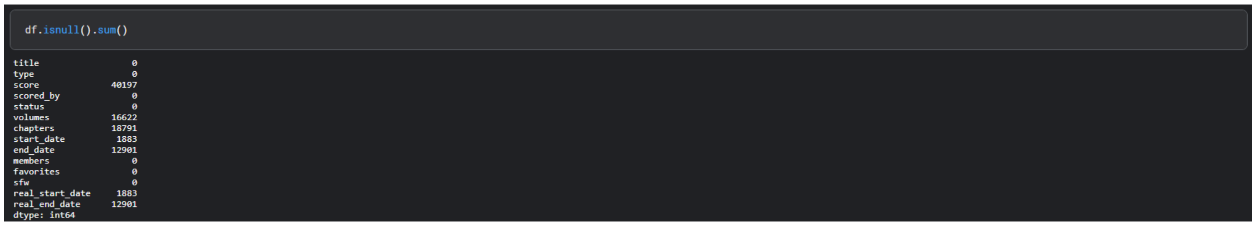
| | title | type | score | scored_by | status | volumes | chapters | start_date | end_date | members | favorites | sfw | real_start_date | real_end_date |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Berserk | manga | 9.47 | 319696 | currently_publishing | NaN | NaN | 1989-08-25 | NaN | 643969 | 119470 | True | 1989-08-25 | NaN |
| 1 | One Piece | manga | 9.22 | 355375 | currently_publishing | NaN | NaN | 1997-07-22 | NaN | 579557 | 111462 | True | 1997-07-22 | NaN |
| 2 | JoJo no Kimyou na Bouken Part 7: Steel Ball Run | manga | 9.30 | 151433 | finished | 24.0 | 96.0 | 2004-01-19 | 2011-04-19 | 248511 | 41713 | True | 2004-01-19 | 2011-04-19 |
| 3 | Oyasumi Punpun | manga | 9.02 | 168459 | finished | 13.0 | 147.0 | 2007-03-15 | 2013-11-02 | 413897 | 49361 | True | 2007-03-15 | 2013-11-02 |
| 4 | Fullmetal Alchemist | manga | 9.03 | 153151 | finished | 27.0 | 116.0 | 2001-07-12 | 2010-09-11 | 284027 | 29634 | True | 2001-07-12 | 2010-09-11 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 64828 | Raise wa Kimi no Mono | manga | NaN | 1 | finished | 1.0 | 8.0 | 2018-08-30 | 2019-04-10 | 4 | 0 | True | 2018-08-30 | 2019-04-10 |
| 64829 | Ore to Basil no Automata | manga | NaN | 0 | finished | 2.0 | 19.0 | 2013-01-19 | 2014-04-19 | 4 | 0 | True | 2013-01-19 | 2014-04-19 |
| 64830 | Happy Candy Virus | manga | NaN | 0 | finished | 4.0 | 54.0 | 2021-08-02 | 2023-03-13 | 4 | 0 | True | 2021-08-02 | 2023-03-13 |
| 64831 | Suibu Yametai Hagino-san | one_shot | NaN | 0 | finished | NaN | 1.0 | 2022-03-03 | 2022-03-03 | 4 | 0 | True | 2022-03-03 | 2022-03-03 |
| 64832 | Kidou Senshi Gundam F91 Prequel | manga | NaN | 0 | currently_publishing | NaN | NaN | 2020-01-24 | NaN | 3 | 0 | True | 2020-01-24 | NaN |

These features were dropped as there was no relevance with the subject at hand. So, now the subset of features that I have to work with are: **'title', 'type', 'score', 'scored_by', 'status', 'volumes', 'status', 'volumes', 'chapters', 'start_date', 'end_date', 'members', 'favorites', 'sfw', 'real_start_date', 'real_end_date'.**

So now, the data frame went from having 30 columns to 14 columns.

## Data Cleaning

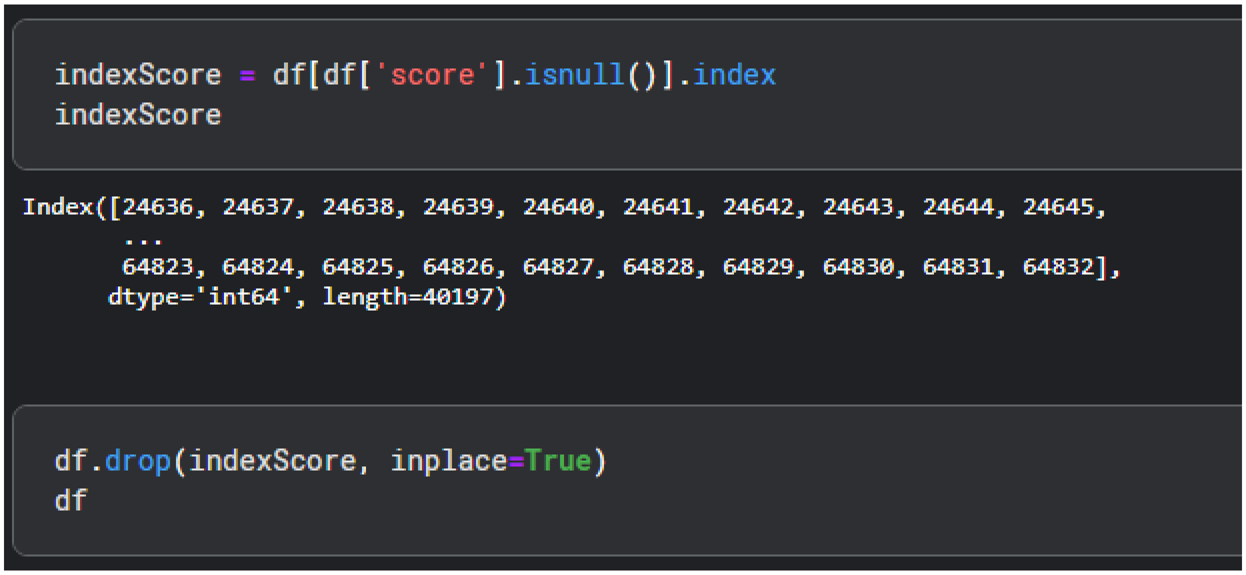
The dataset was checked for unexpected values and missing values. Although there were no unexpected values, there were a lot of missing values and I checked how many values were missing in each column using a simple piece of code.



```
df.isnull().sum()
```

```
title                  0
type                   0
score              40197
scored_by              0
status                 0
volumes            16622
chapters           18791
start_date          1883
end_date           12901
members                0
favorites              0
sfw                    0
real_start_date     1883
real_end_date      12901
dtype: int64
```

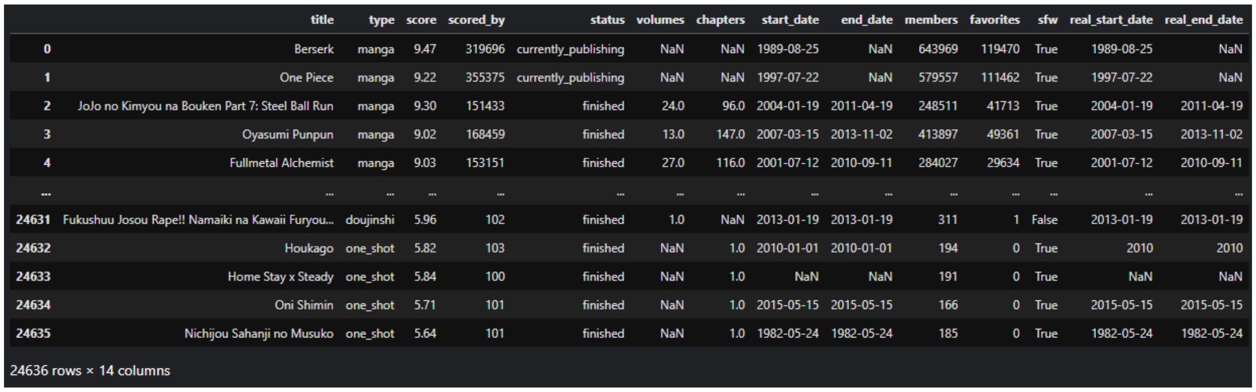
The dataset was pretty clean as it is and did not require a lot of cleaning or pre-processing from my end. Although, we could still see that there were a lot of missing values especially more than 50% of the entries were missing for the feature 'score'. So, I dropped all the rows that had missing values for the feature 'score'.



```
indexScore = df[df['score'].isnull()].index
indexScore
```

```
Index([24636, 24637, 24638, 24639, 24640, 24641, 24642, 24643, 24644, 24645,
       ...
       64823, 64824, 64825, 64826, 64827, 64828, 64829, 64830, 64831, 64832],
      dtype='int64', length=40197)
```

```
df.drop(indexScore, inplace=True)
df
```

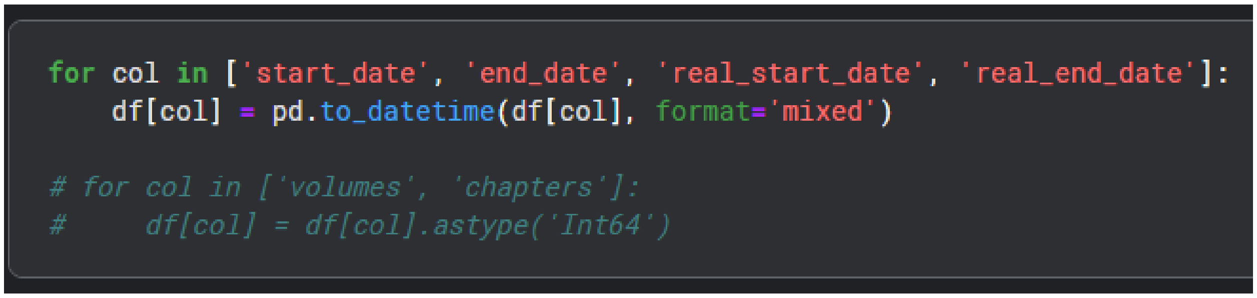


| | title | type | score | scored_by | status | volumes | chapters | start_date | end_date | members | favorites | sfw | real_start_date | real_end_date |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | Berserk | manga | 9.47 | 319696 | currently_publishing | NaN | NaN | 1989-08-25 | NaN | 643969 | 119470 | True | 1989-08-25 | NaN |
| **1** | One Piece | manga | 9.22 | 355375 | currently_publishing | NaN | NaN | 1997-07-22 | NaN | 579557 | 111462 | True | 1997-07-22 | NaN |
| **2** | JoJo no Kimyou na Bouken Part 7: Steel Ball Run | manga | 9.30 | 151433 | finished | 24.0 | 96.0 | 2004-01-19 | 2011-04-19 | 248511 | 41713 | True | 2004-01-19 | 2011-04-19 |
| **3** | Oyasumi Punpun | manga | 9.02 | 168459 | finished | 13.0 | 147.0 | 2007-03-15 | 2013-11-02 | 413897 | 49361 | True | 2007-03-15 | 2013-11-02 |
| **4** | Fullmetal Alchemist | manga | 9.03 | 153151 | finished | 27.0 | 116.0 | 2001-07-12 | 2010-09-11 | 284027 | 29634 | True | 2001-07-12 | 2010-09-11 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **24631** | Fukushuu Josou Rape!! Namaiki na Kawaii Furyou... | doujinshi | 5.96 | 102 | finished | 1.0 | NaN | 2013-01-19 | 2013-01-19 | 311 | 1 | False | 2013-01-19 | 2013-01-19 |
| **24632** | Houkago | one_shot | 5.82 | 103 | finished | NaN | 1.0 | 2010-01-01 | 2010-01-01 | 194 | 0 | True | 2010 | 2010 |
| **24633** | Home Stay x Steady | one_shot | 5.84 | 100 | finished | NaN | 1.0 | NaN | NaN | 191 | 0 | True | NaN | NaN |
| **24634** | Oni Shimin | one_shot | 5.71 | 101 | finished | NaN | 1.0 | 2015-05-15 | 2015-05-15 | 166 | 0 | True | 2015-05-15 | 2015-05-15 |
| **24635** | Nichijou Sahanji no Musuko | one_shot | 5.64 | 101 | finished | NaN | 1.0 | 1982-05-24 | 1982-05-24 | 185 | 0 | True | 1982-05-24 | 1982-05-24 |

24636 rows × 14 columns

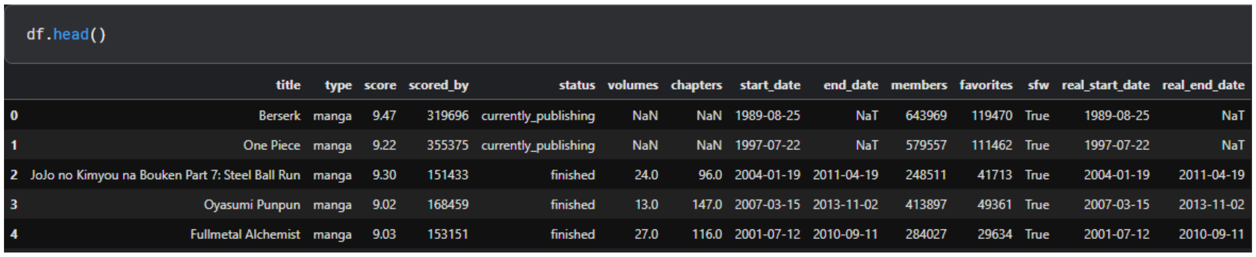
The other features were not imputed or cleaned in someway because it actually made sense for them to be missing values, for example. The currently publishing mangas will definitely not have an end_date. The Volumes and Chapters would obviously not be shown for the currently publishing mangas. So, those were not worked with.

I although worked with the date columns by first converting them into date-time format and then just extracting the years and keeping them in their respective columns.



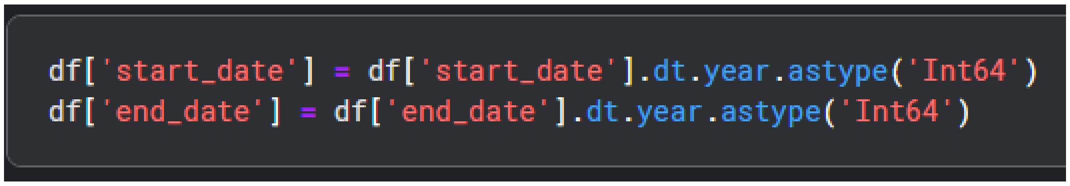
```
for col in ['start_date', 'end_date', 'real_start_date', 'real_end_date']:
    df[col] = pd.to_datetime(df[col], format='mixed')

# for col in ['volumes', 'chapters']:
#     df[col] = df[col].astype('Int64')
```

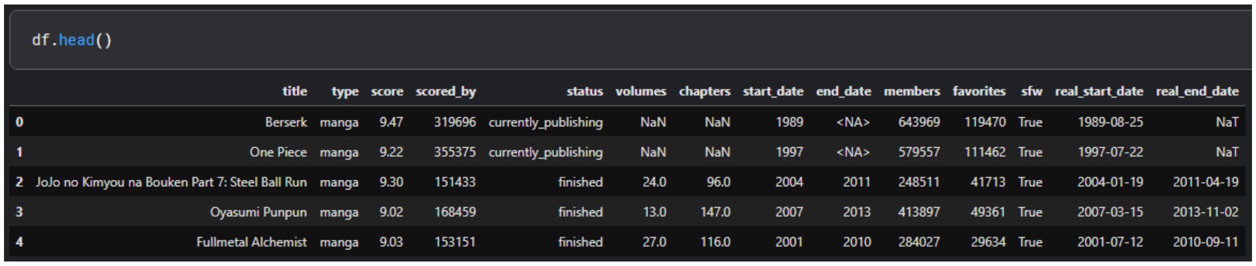


```python
df.head()
```

| | title | type | score | scored_by | status | volumes | chapters | start_date | end_date | members | favorites | sfw | real_start_date | real_end_date |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Berserk | manga | 9.47 | 319696 | currently_publishing | NaN | NaN | 1989-08-25 | NaT | 643969 | 119470 | True | 1989-08-25 | NaT |
| 1 | One Piece | manga | 9.22 | 355375 | currently_publishing | NaN | NaN | 1997-07-22 | NaT | 579557 | 111462 | True | 1997-07-22 | NaT |
| 2 | JoJo no Kimyou na Bouken Part 7: Steel Ball Run | manga | 9.30 | 151433 | finished | 24.0 | 96.0 | 2004-01-19 | 2011-04-19 | 248511 | 41713 | True | 2004-01-19 | 2011-04-19 |
| 3 | Oyasumi Punpun | manga | 9.02 | 168459 | finished | 13.0 | 147.0 | 2007-03-15 | 2013-11-02 | 413897 | 49361 | True | 2007-03-15 | 2013-11-02 |
| 4 | Fullmetal Alchemist | manga | 9.03 | 153151 | finished | 27.0 | 116.0 | 2001-07-12 | 2010-09-11 | 284027 | 29634 | True | 2001-07-12 | 2010-09-11 |



```python
df['start_date'] = df['start_date'].dt.year.astype('Int64')
df['end_date'] = df['end_date'].dt.year.astype('Int64')
```

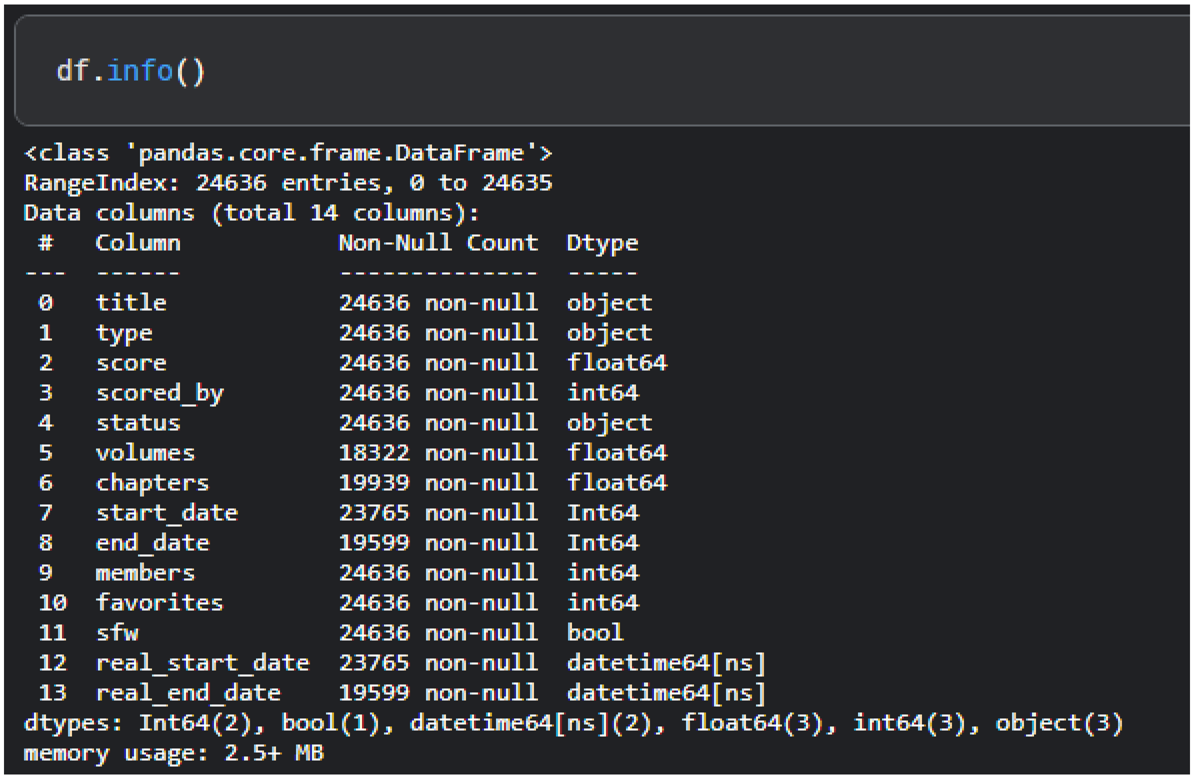


```python
df.head()
```

| | title | type | score | scored_by | status | volumes | chapters | start_date | end_date | members | favorites | sfw | real_start_date | real_end_date |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Berserk | manga | 9.47 | 319696 | currently_publishing | NaN | NaN | 1989 | <NA> | 643969 | 119470 | True | 1989-08-25 | NaT |
| 1 | One Piece | manga | 9.22 | 355375 | currently_publishing | NaN | NaN | 1997 | <NA> | 579557 | 111462 | True | 1997-07-22 | NaT |
| 2 | JoJo no Kimyou na Bouken Part 7: Steel Ball Run | manga | 9.30 | 151433 | finished | 24.0 | 96.0 | 2004 | 2011 | 248511 | 41713 | True | 2004-01-19 | 2011-04-19 |
| 3 | Oyasumi Punpun | manga | 9.02 | 168459 | finished | 13.0 | 147.0 | 2007 | 2013 | 413897 | 49361 | True | 2007-03-15 | 2013-11-02 |
| 4 | Fullmetal Alchemist | manga | 9.03 | 153151 | finished | 27.0 | 116.0 | 2001 | 2010 | 284027 | 29634 | True | 2001-07-12 | 2010-09-11 |

Now, that the data cleaning is done, I now check the data and try to summarize.

## Summary Statistics

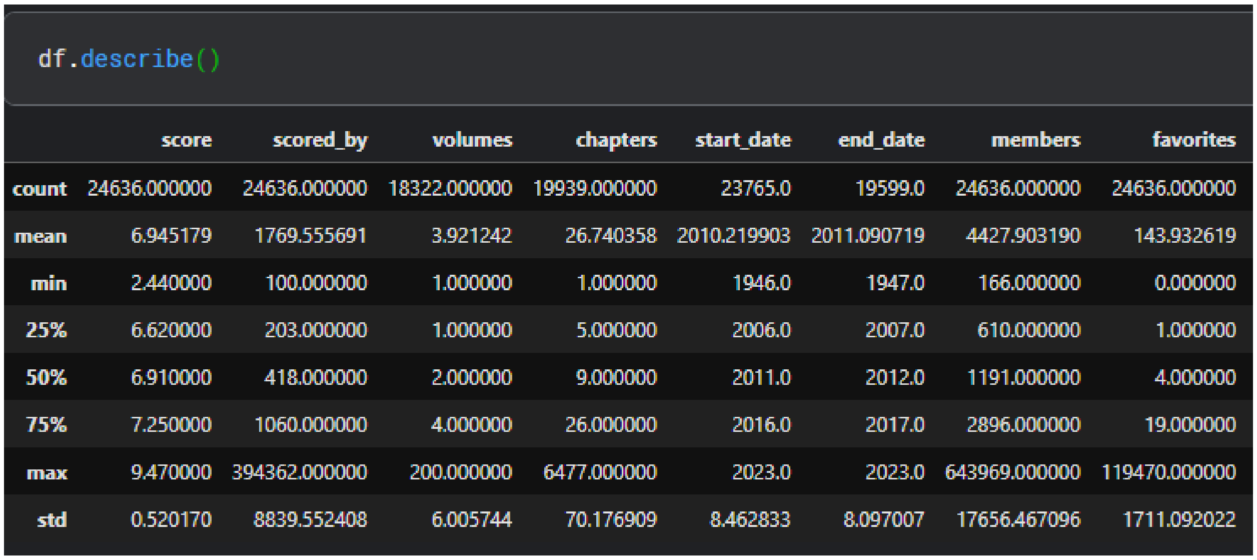


```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24636 entries, 0 to 24635
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   title            24636 non-null  object
 1   type             24636 non-null  object
 2   score            24636 non-null  float64
 3   scored_by        24636 non-null  int64
 4   status           24636 non-null  object
 5   volumes          18322 non-null  float64
 6   chapters         19939 non-null  float64
 7   start_date       23765 non-null  Int64
 8   end_date         19599 non-null  Int64
 9   members          24636 non-null  int64
 10  favorites        24636 non-null  int64
 11  sfw              24636 non-null  bool
 12  real_start_date  23765 non-null  datetime64[ns]
 13  real_end_date    19599 non-null  datetime64[ns]
dtypes: Int64(2), bool(1), datetime64[ns](2), float64(3), int64(3), object(3)
memory usage: 2.5+ MB
```

The data frame has 24636 rows/entries and 14 columns/features after cleaning that data. The **info()** function returns a few details about the data frame such as the amount of non-null values in each column and data type of the data stored in each column.

The describe() function gives us the descriptive statistics of the data such as mean, mode, median, etc and it is really useful for our purpose of making inferences.



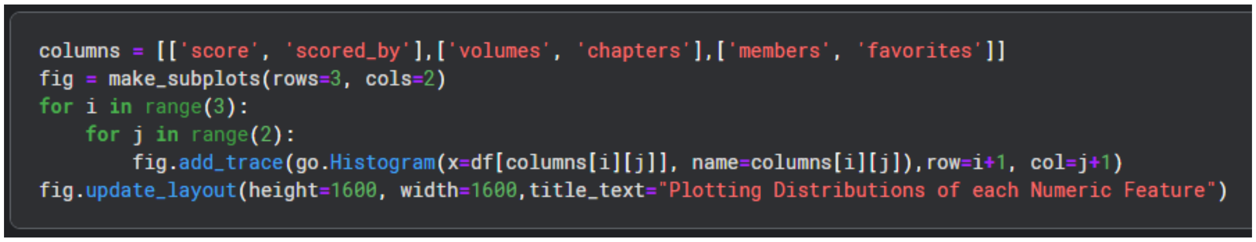
```
df.describe()
```

| | score | scored_by | volumes | chapters | start_date | end_date | members | favorites |
|---|---|---|---|---|---|---|---|---|
| count | 24636.000000 | 24636.000000 | 18322.000000 | 19939.000000 | 23765.0 | 19599.0 | 24636.000000 | 24636.000000 |
| mean | 6.945179 | 1769.555691 | 3.921242 | 26.740358 | 2010.219903 | 2011.090719 | 4427.903190 | 143.932619 |
| min | 2.440000 | 100.000000 | 1.000000 | 1.000000 | 1946.0 | 1947.0 | 166.000000 | 0.000000 |
| 25% | 6.620000 | 203.000000 | 1.000000 | 5.000000 | 2006.0 | 2007.0 | 610.000000 | 1.000000 |
| 50% | 6.910000 | 418.000000 | 2.000000 | 9.000000 | 2011.0 | 2012.0 | 1191.000000 | 4.000000 |
| 75% | 7.250000 | 1060.000000 | 4.000000 | 26.000000 | 2016.0 | 2017.0 | 2896.000000 | 19.000000 |
| max | 9.470000 | 394362.000000 | 200.000000 | 6477.000000 | 2023.0 | 2023.0 | 643969.000000 | 119470.000000 |
| std | 0.520170 | 8839.552408 | 6.005744 | 70.176909 | 8.462833 | 8.097007 | 17656.467096 | 1711.092022 |

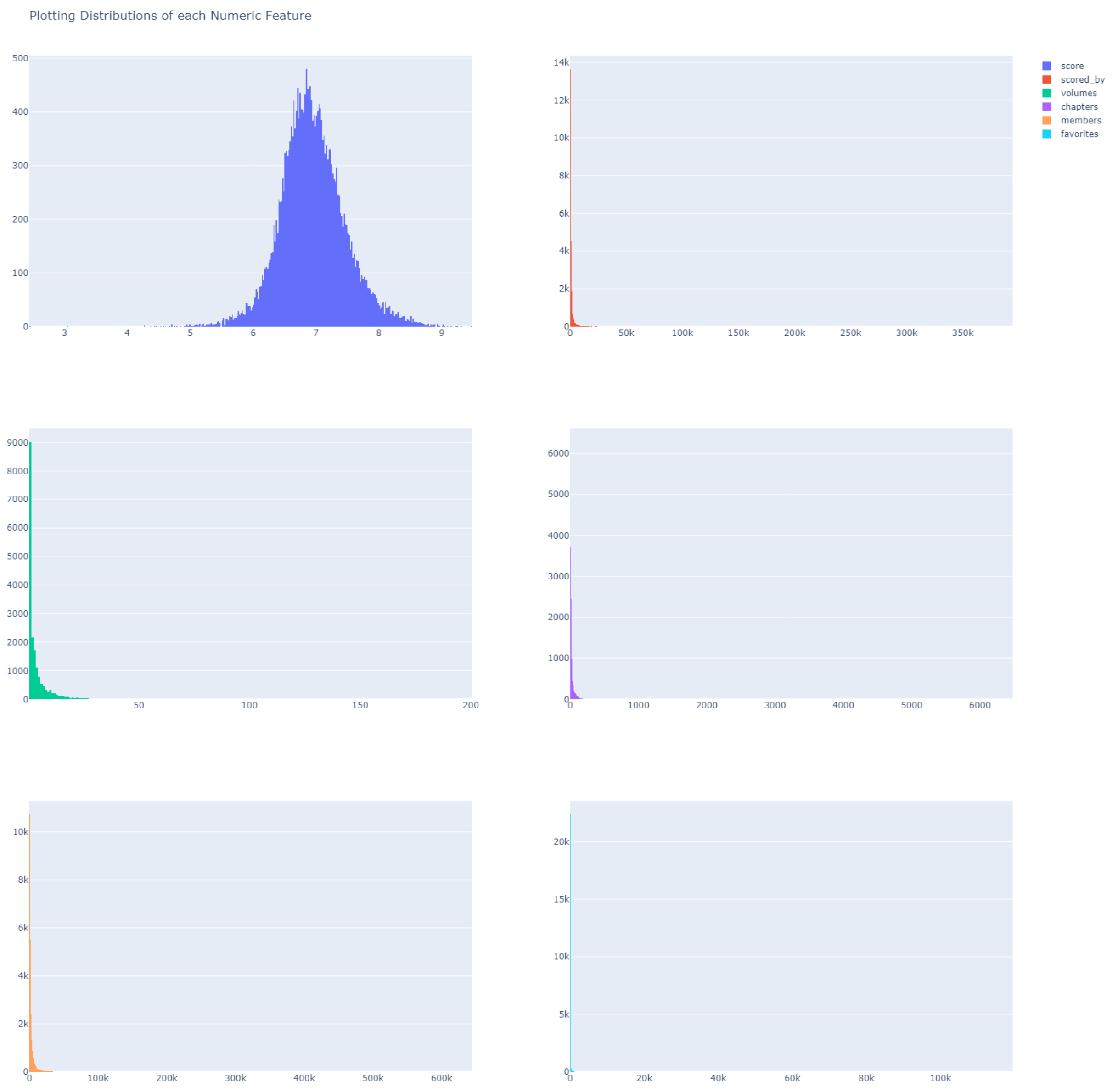
These values will be used later for applications such as Construction of Confidence Intervals and Testing Hypothesis.

## Visualizing Continuous Features

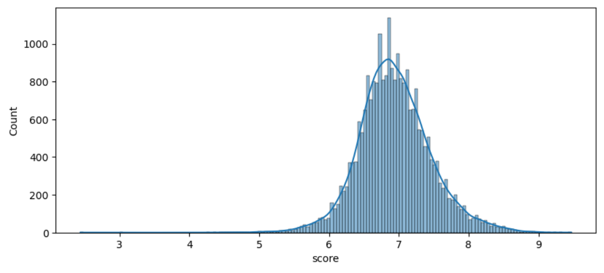
There are totally 6 continuous features in the dataset, namely ‘score’, ‘scored_by’, ‘volumes’, ‘chapters’, ‘members’ and ‘favorites’. I used Plotly, a plotting library in python to plot the graphs.

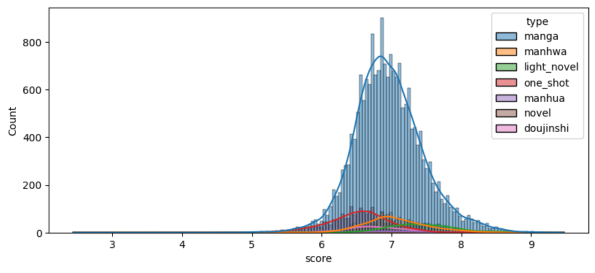


```
columns = [['score', 'scored_by'],['volumes', 'chapters'],['members', 'favorites']]
fig = make_subplots(rows=3, cols=2)
for i in range(3):
    for j in range(2):
        fig.add_trace(go.Histogram(x=df[columns[i][j]], name=columns[i][j]),row=i+1, col=j+1)
fig.update_layout(height=1600, width=1600,title_text="Plotting Distributions of each Numeric Feature")
```

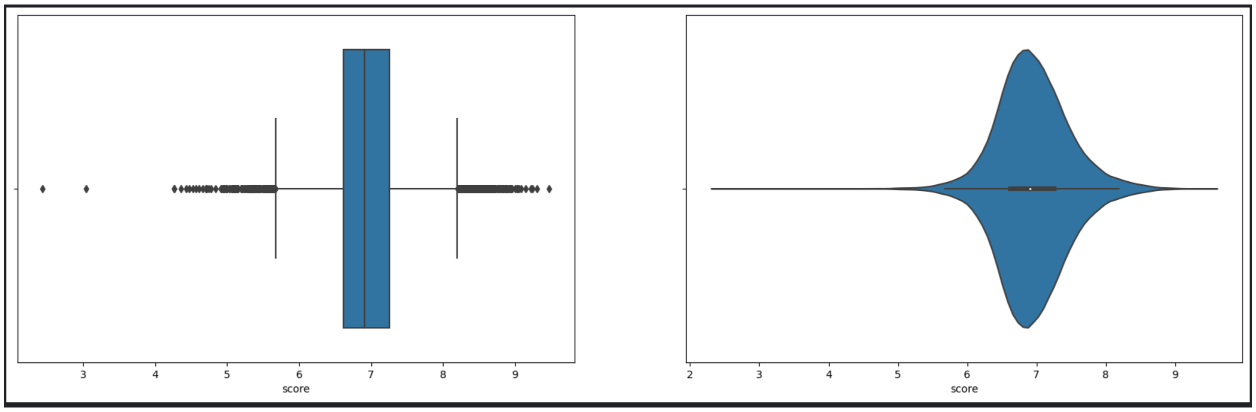


The ‘score’ feature was the one that made sense when plotted. I could see the bell shape in the graph, hence I could atleast just by looking, say that the data in the feature follow a normal distribution.





Plotting it using KDE helped a lot. Now the only step left was to plot a few more different types of plots on the same feature to concretize the notion that it followed a normal distribution. Checking the skewness also helped.

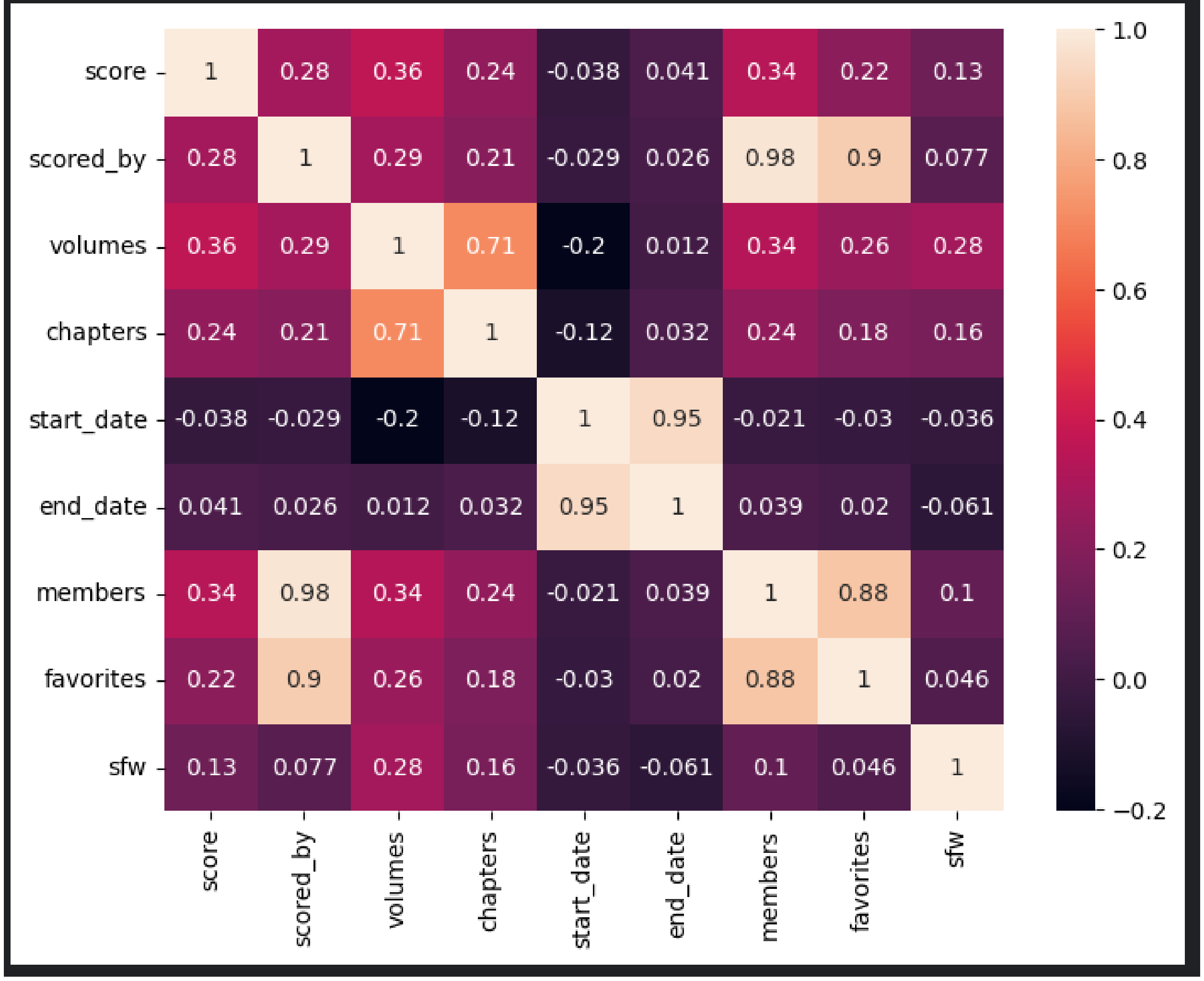


I plotted the ‘score’ feature with boxplot and a violinplot using seaborn, a yet another python library used for plotting. The boxplot and violinplot definitely helped in being sure that the feature followed a normal distribution, one additional aspect discovered was the amount of outliers that were there in the feature was a lot.

The skewness could be checked using the skew() function that Pandas offers, and the skewness value that I got was **0.18206351648076116**. But 0.18 felt a little bit higher for a feature that was normal, so I decided to normalize that feature even further.

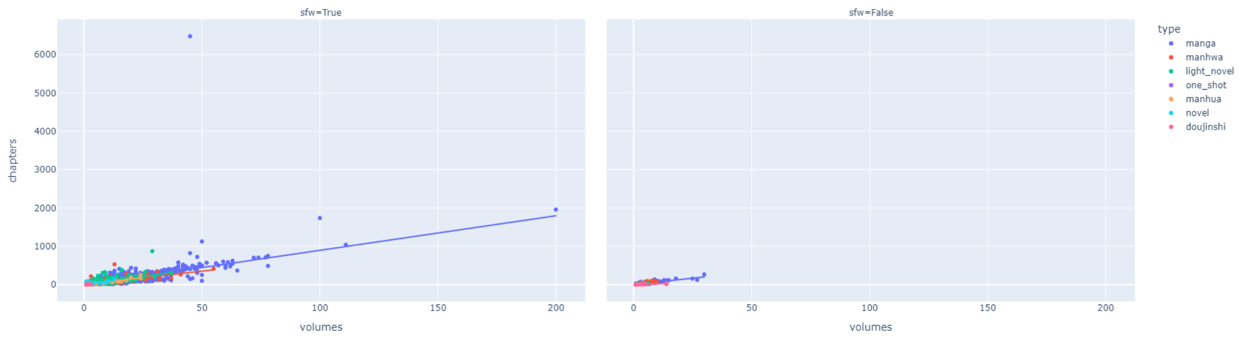
Before normalizing, I tried plotting and finding the correlation between the ‘score’ feature and other features starting with the heatmap.

```
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(numeric_only=True), annot=True)
plt.show()
```

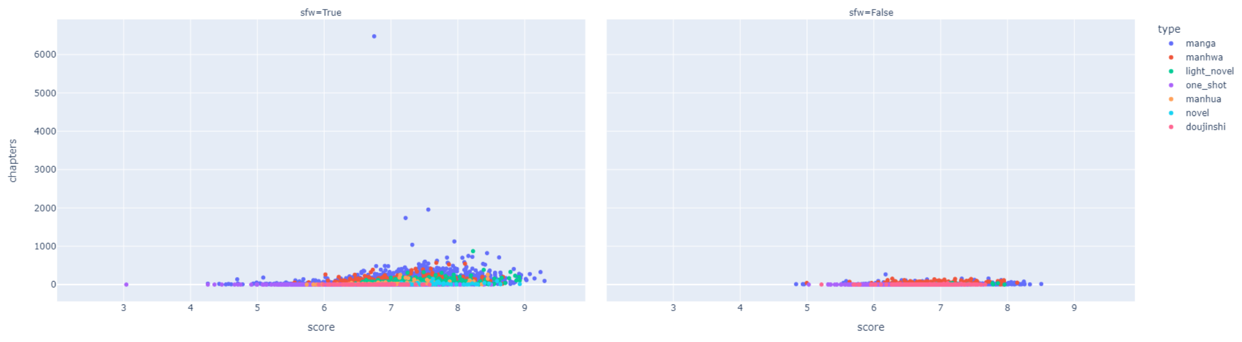


| | score | scored_by | volumes | chapters | start_date | end_date | members | favorites | sfw |
|---|---|---|---|---|---|---|---|---|---|
| score | 1 | 0.28 | 0.36 | 0.24 | -0.038 | 0.041 | 0.34 | 0.22 | 0.13 |
| scored_by | 0.28 | 1 | 0.29 | 0.21 | -0.029 | 0.026 | 0.98 | 0.9 | 0.077 |
| volumes | 0.36 | 0.29 | 1 | 0.71 | -0.2 | 0.012 | 0.34 | 0.26 | 0.28 |
| chapters | 0.24 | 0.21 | 0.71 | 1 | -0.12 | 0.032 | 0.24 | 0.18 | 0.16 |
| start_date | -0.038 | -0.029 | -0.2 | -0.12 | 1 | 0.95 | -0.021 | -0.03 | -0.036 |
| end_date | 0.041 | 0.026 | 0.012 | 0.032 | 0.95 | 1 | 0.039 | 0.02 | -0.061 |
| members | 0.34 | 0.98 | 0.34 | 0.24 | -0.021 | 0.039 | 1 | 0.88 | 0.1 |
| favorites | 0.22 | 0.9 | 0.26 | 0.18 | -0.03 | 0.02 | 0.88 | 1 | 0.046 |
| sfw | 0.13 | 0.077 | 0.28 | 0.16 | -0.036 | -0.061 | 0.1 | 0.046 | 1 |

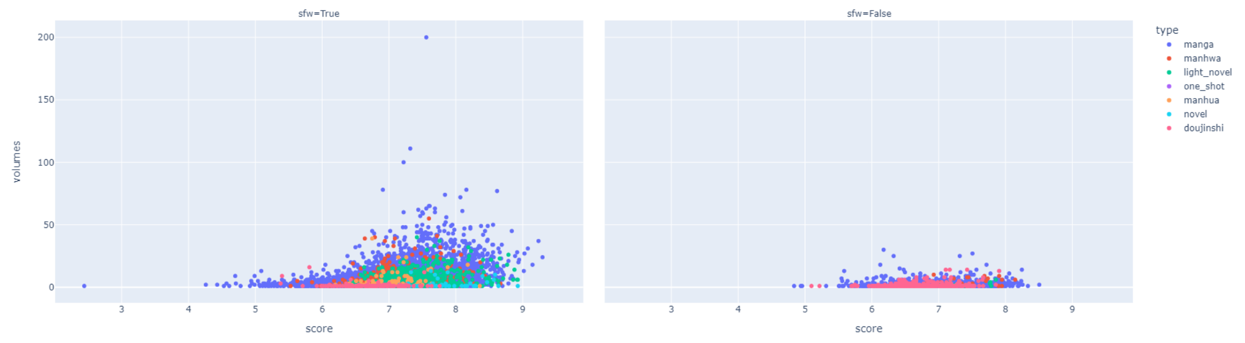
This might be the most obvious inference that can be made. We could see from the correlation heatmap that the features 'volumes' and 'chapters' are positively correlated and to a good extent which makes sense because it is inherent that if the number of volumes are high, the book should have more chapters. A lot of the other correlation coeffecients also only state the obvious for this dataset. The scatter plot shown below shows the positive correlation between volumes and chapters. All of the upcoming plots will be split into two mostly on feature ‘sfw’.



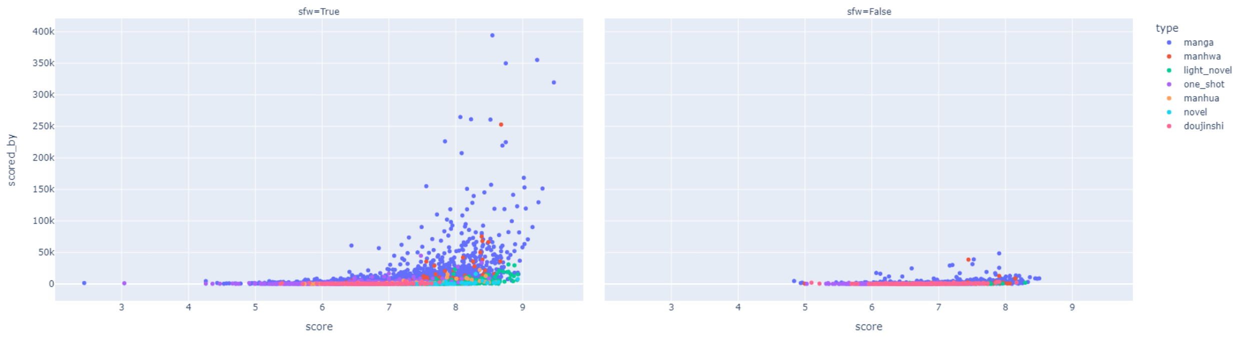
There are a few more obvious inferences we can do about correlations between features that makes a lot of sense, like the number of 'members' who have added that book to their library and the number of people who have 'favourited' them; 'scored_by' and 'favourites' also makes a lot of sense to have high correlation and is obvious, as well as 'members' and 'scored_by'. So their plots are useless for us now. Let us look at more interesting correlations.
The 'score' and 'chapters' have a good inference that can be drawn from the scatterplot. Mostly, people have given good ratings for Mangas ans Books that are having less than 1000 chapters, as we can see that a good number of entries that have good scores are accumulated near the score ranges of 7 to 9 and all those entries' chapter counts are well below 1000. Although, there are a few outliers who are scored considerably well and they have more than 1000 chapters, so the inference is that as the number of chapters increases, people definitely don't like the Manga but that is not all, there are a few more attributes that are also considered like how interesting the story is, for example.



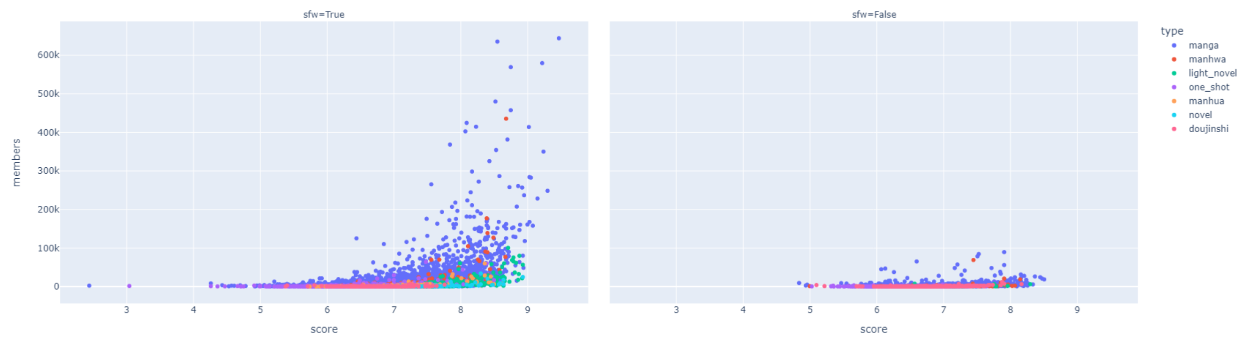
The 'scores' and 'volumes' also will have the same inference. People do not give good scores for Mangas and Books that exceed 50 chapters but still we have outliers which again means that the number of volumes is alone not the criteria. But based on most of the entries, we can infer that 50 and below is a good number of volumes for a Manga or a Book.



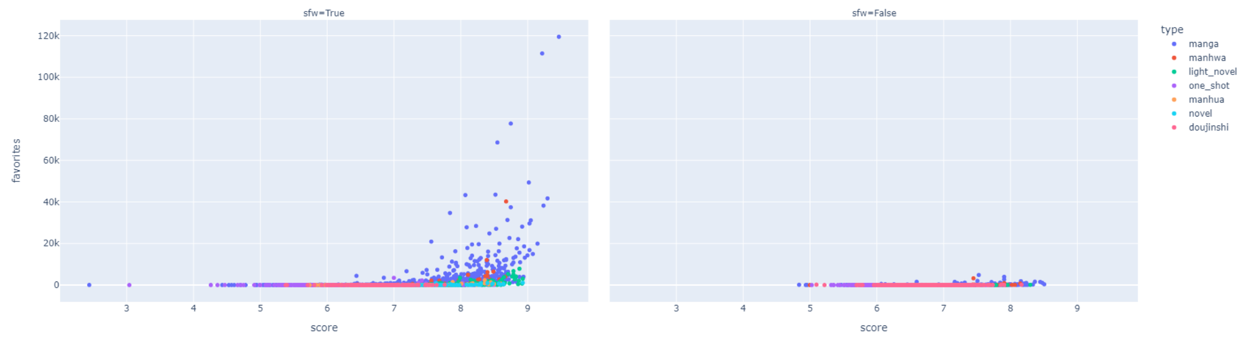
The next scatterplot shows that the score is increasing as the scored_by is also increasing but this is not always an obvious behavior as we know that a product with really good average score that has only very few people who have scored them is not a good judge of the quality of the product. Here, we can see that there are many entries that have good overall scores even after having so many people rating them as those mangas and books are actually nice and people have liked them. Still, this also has some outliers.



The 'score' and 'members' features' correlation makes a good observation. As the manga score is better, more people have added it to their database which means that they have either planned to read it or have already read it. This can mean the other way around too. People who have added it to their database could have also rated it, so the previous plot is a subset of this plot. We can clearly see that the member count increases as the score increases too.



This is also a subset of the previous plot, the number of favorites is increasing as the score is increasing. This is a very obvious relation, because when people like it, they will add it to their favorites list and also rate it if they are interested in doing so. So it perfectly makes sense that the mangas and books with the highest scores also are favorited by more people.



**Inferences from the Above Scatter Plots**: (We are only concerned with Manga here)

1. **Sazae-san** is the SFW manga with the **highest chapter count** having **6477** chapters in total while **Infection** is the NSFW manga with the highest chapter count having **267** chapters in total.
2. **Jojo's Bizarre Adventures** is the **highest rated** SFW manga with a score of **9.30** and **Sotsugyousei** is the highest rated NSFW manga with a score of **8.51**
3. **Kochira...** is the SFW manga with the **highest volume count** with **200** volumes in total and **Infection** is the NSFW manga with the highest volume count with **30** volumes in total.
4. **Berserk** (SFW) and **Nozoki Ana** (NSFW) are the manga with the **highest number of people having added them to their database**. They are also the mangas with the **highest favorites count**.

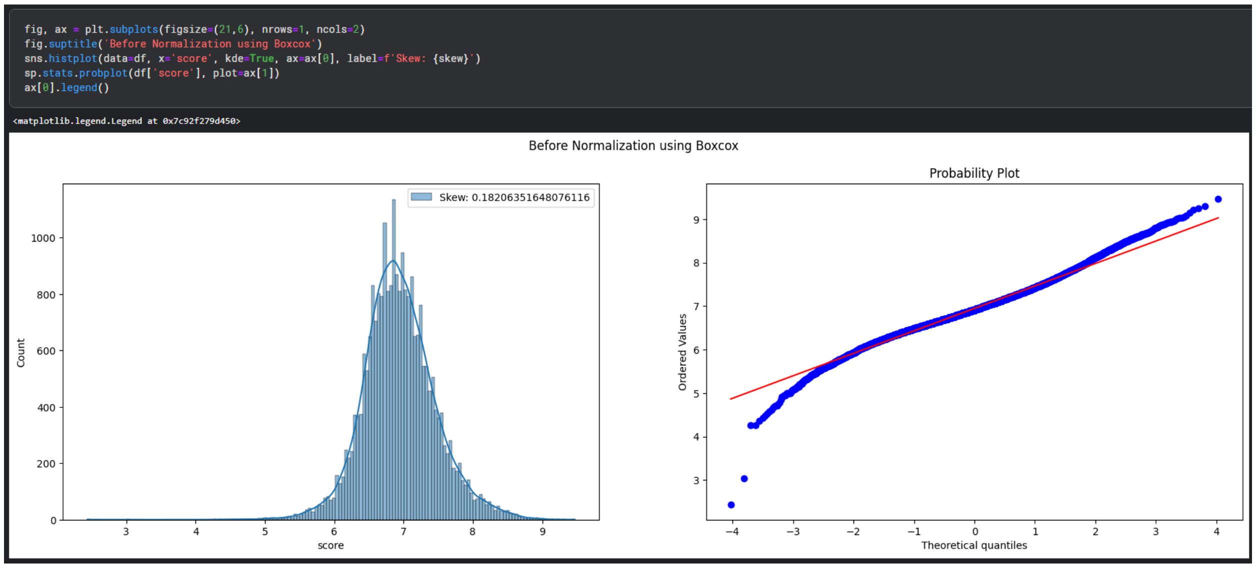
Now, time to normalize the ‘score’ feature

I used the boxcox technique given in the scipy library. The boxcox transform is given as:

$$y = \frac{x^{\lambda - 1}}{\lambda}, \lambda \neq 0$$

$$log(x), \lambda = 0$$

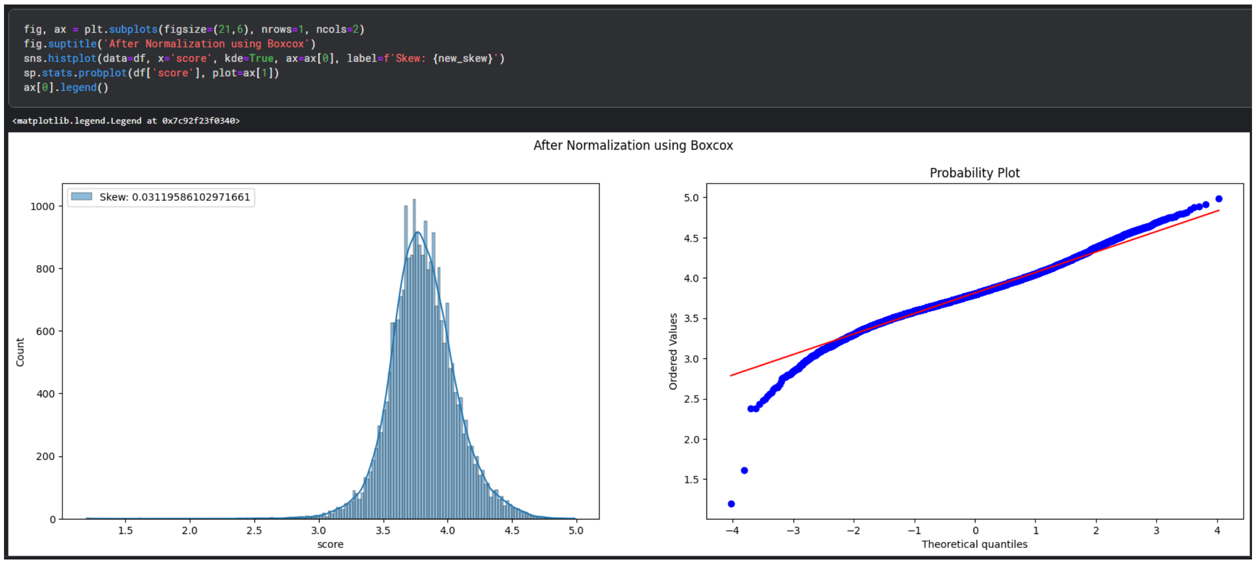
This is how the plot is before the ‘score’ feature is normalized



```
fig, ax = plt.subplots(figsize=(21,6), nrows=1, ncols=2)
fig.suptitle('Before Normalization using Boxcox')
sns.histplot(data=df, x='score', kde=True, ax=ax[0], label=f'Skew: {skew}')
sp.stats.probplot(df['score'], plot=ax[1])
ax[0].legend()
```

```
<matplotlib.legend.Legend at 0x7c92f279d450>
```

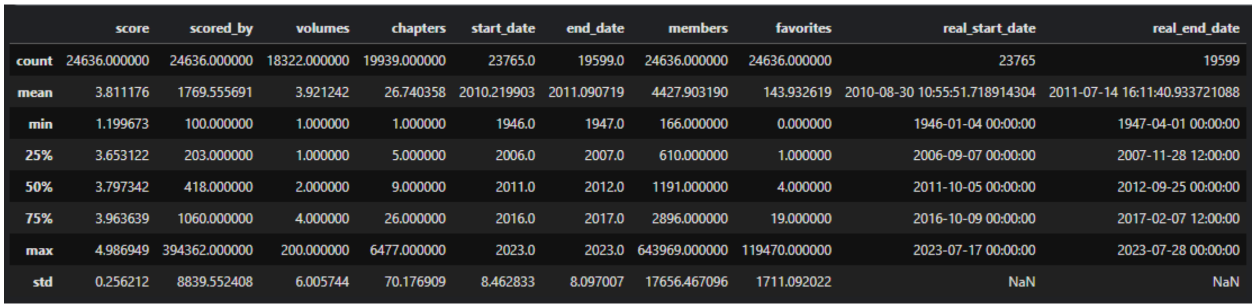
This is how the plot is after the ‘score’ feature is normalized using Boxcox



```
fig, ax = plt.subplots(figsize=(21,6), nrows=1, ncols=2)
fig.suptitle('After Normalization using Boxcox')
sns.histplot(data=df, x='score', kde=True, ax=ax[0], label=f'Skew: {new_skew}')
sp.stats.probplot(df['score'], plot=ax[1])
ax[0].legend()
```

```
<matplotlib.legend.Legend at 0x7c92f23f0340>
```

The data is normalized even further and we can see that the skew is 0.03. But now that we have normalized the score feature, let us check how it change the summary statistics.



|  | score | scored_by | volumes | chapters | start_date | end_date | members | favorites | real_start_date | real_end_date |
|---|---|---|---|---|---|---|---|---|---|---|
| count | 24636.000000 | 24636.000000 | 18322.000000 | 19939.000000 | 23765.0 | 19599.0 | 24636.000000 | 24636.000000 | 23765 | 19599 |
| mean | 3.811176 | 1769.555691 | 3.921242 | 26.740358 | 2010.219903 | 2011.090719 | 4427.903190 | 143.932619 | 2010-08-30 10:55:51.718914304 | 2011-07-14 16:11:40.933721088 |
| min | 1.199673 | 100.000000 | 1.000000 | 1.000000 | 1946.0 | 1947.0 | 166.000000 | 0.000000 | 1946-01-04 00:00:00 | 1947-04-01 00:00:00 |
| 25% | 3.653122 | 203.000000 | 1.000000 | 5.000000 | 2006.0 | 2007.0 | 610.000000 | 1.000000 | 2006-09-07 00:00:00 | 2007-11-28 12:00:00 |
| 50% | 3.797342 | 418.000000 | 2.000000 | 9.000000 | 2011.0 | 2012.0 | 1191.000000 | 4.000000 | 2011-10-05 00:00:00 | 2012-09-25 00:00:00 |
| 75% | 3.963639 | 1060.000000 | 4.000000 | 26.000000 | 2016.0 | 2017.0 | 2896.000000 | 19.000000 | 2016-10-09 00:00:00 | 2017-02-07 12:00:00 |
| max | 4.986949 | 394362.000000 | 200.000000 | 6477.000000 | 2023.0 | 2023.0 | 643969.000000 | 119470.000000 | 2023-07-17 00:00:00 | 2023-07-28 00:00:00 |
| std | 0.256212 | 8839.552408 | 6.005744 | 70.176909 | 8.462833 | 8.097007 | 17656.467096 | 1711.092022 | NaN | NaN |

We can see that now the range of scores is not from 0 to 10 anymore, it is 0 to 5 now. Now that we have done these inferences, let us construct a confidence interval for the mean of scores and also let us put forward an hypothesis and test it.

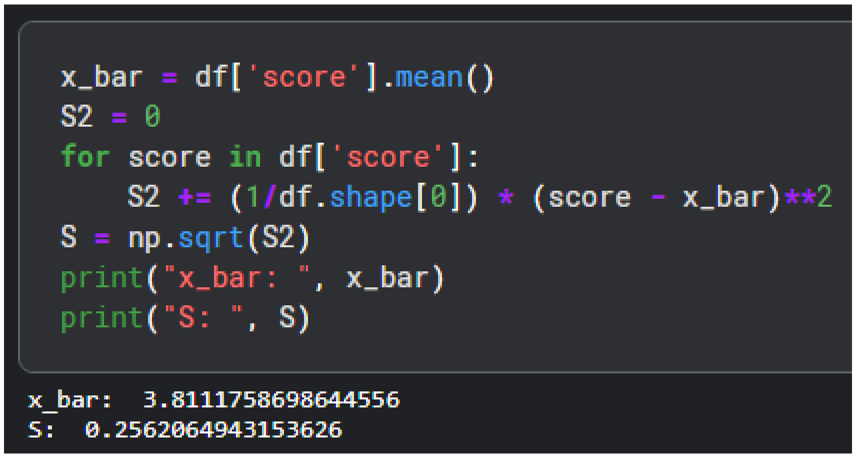
## Construction of confidence interval for mean of scores

We see that the population size is large and it follows a normal distribution, we construct a confidence interval using S (since the population standard deviation is unknown). The population mean is given as 3.811176. So we use the formula

$$\bar{x} \pm Z_{\frac{\alpha}{2}} \frac{S}{\sqrt{n}}$$

to construct the 95% confidence interval for the sample mean for the first 100 samples from the population.

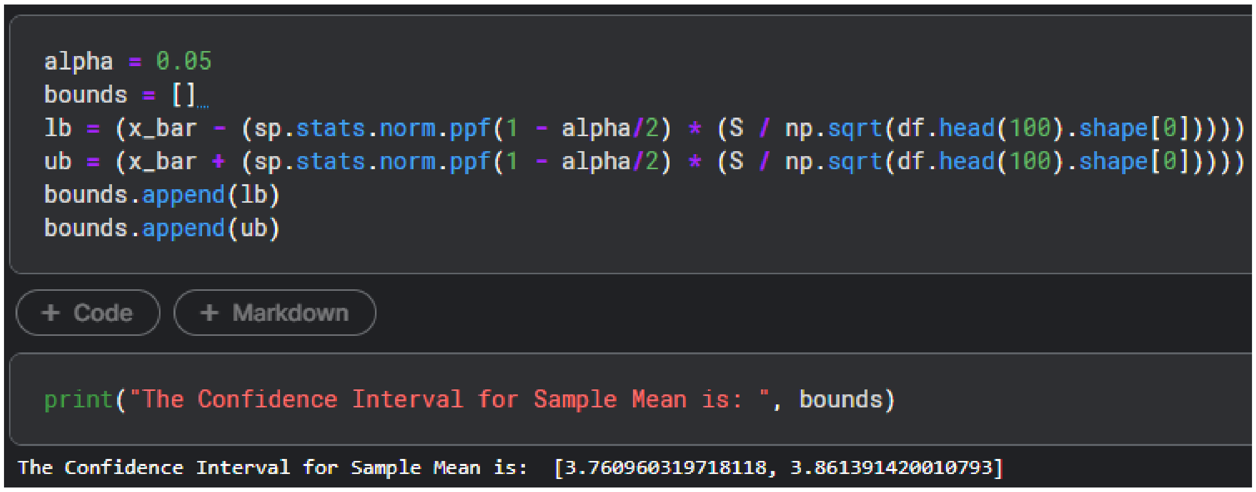
Calculating the population mean and the standard deviation S is done as follows:



```
x_bar = df['score'].mean()
S2 = 0
for score in df['score']:
    S2 += (1/df.shape[0]) * (score - x_bar)**2
S = np.sqrt(S2)
print("x_bar: ", x_bar)
print("S: ", S)
```

```
x_bar:  3.8111758698644556
S:  0.2562064943153626
```

The confidence interval is constructed as follows:



```
alpha = 0.05
bounds = []
lb = (x_bar - (sp.stats.norm.ppf(1 - alpha/2) * (S / np.sqrt(df.head(100).shape[0]))))
ub = (x_bar + (sp.stats.norm.ppf(1 - alpha/2) * (S / np.sqrt(df.head(100).shape[0]))))
bounds.append(lb)
bounds.append(ub)
```

```
print("The Confidence Interval for Sample Mean is: ", bounds)
```

```
The Confidence Interval for Sample Mean is:  [3.760960319718118, 3.861391420010793]
```

This is the required 95% confidence interval for the sample mean : [3.760960319718118, 3.861391420010793].

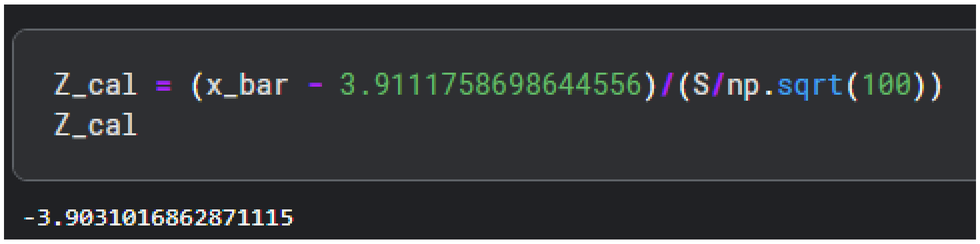
## Testing Hypothesis

I propose a hypothesis for the mean of the data ('score' feature)

$$H_0 : \mu = 3.911175 \text{ vs } H_1 : \mu \neq 3.911175$$

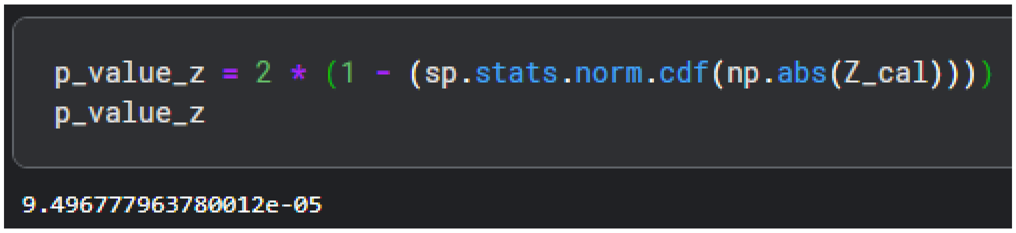
With alpha = 0.5, and sigma unkown. So we do a Z-test with S as our population standard deviation as our sample size is 100. Z_cal is calculated as

$$Z_{cal} = \frac{\bar{x} - \mu_0}{\frac{S}{\sqrt{n}}}$$



```
Z_cal = (x_bar - 3.9111758698644556)/(S/np.sqrt(100))
Z_cal
```

```
-3.9031016862871115
```



```
p_value_z = 2 * (1 - (sp.stats.norm.cdf(np.abs(Z_cal))))
p_value_z
```

```
9.496777963780012e-05
```

Since the p-value is less than 0.5 (alpha), we reject the null hypothesis.

This concludes the task with the Manga Dataset.