

cf-01-rzepinski-ratings_analysis

January 4, 2019

1 Important

make data has to be run before running any notebook cell

2 Imports

```
In [1]: import pandas as pd
import seaborn as sns
```

```
In [2]: book_df = pd.read_csv('../data/raw/book.csv')
ratings_df = pd.read_csv('../data/raw/ratings.csv')
```

```
In [3]: ratings_df.head(1)
```

```
Out[3]:   user_id  book_id  rating
0         1      258         5
```

Make sure there are no duplicates in ratings.

```
In [4]: ratings_df[ratings_df.duplicated(['user_id', 'book_id'], keep=False)]
```

```
Out[4]: Empty DataFrame
Columns: [user_id, book_id, rating]
Index: []
```

3 Ratings user and book coverage

```
In [5]: ratings_df.groupby('user_id')['book_id'].count().describe()
```

```
Out[5]: count      53424.000000
mean         111.868804
std           26.071224
min           19.000000
25%           96.000000
50%          111.000000
75%          128.000000
max           200.000000
Name: book_id, dtype: float64
```

All users rated at least 19 books. Such situation is rarely encountered in similar datasets.

```
In [6]: ratings_df.groupby('book_id')['user_id'].count().describe()
```

```
Out[6]: count      10000.000000
        mean        597.647900
        std         1267.289788
        min          8.000000
        25%         155.000000
        50%         248.000000
        75%         503.000000
        max        22806.000000
        Name: user_id, dtype: float64
```

All books have been rated at least 8 times.

4 How users rate books?

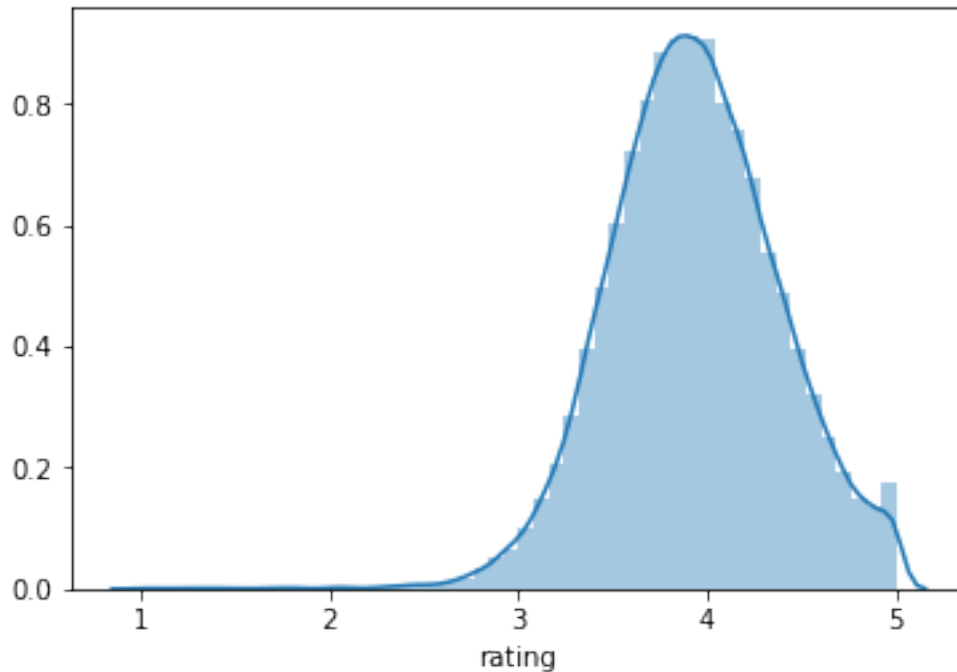
```
In [7]: ratings_df['rating'].describe()
```

```
Out[7]: count      5.976479e+06
        mean      3.919866e+00
        std       9.910868e-01
        min       1.000000e+00
        25%       3.000000e+00
        50%       4.000000e+00
        75%       5.000000e+00
        max       5.000000e+00
        Name: rating, dtype: float64
```

```
In [8]: sns.distplot(ratings_df.groupby('user_id')['rating'].mean())
```

```
/home/rzepinski/Documents/Inzynierka/Recommendation-system/rs-
venv/lib/python3.7/site-packages/scipy/stats/stats.py:1713: FutureWarning: Using a
non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]`
instead of `arr[seq]`. In the future this will be interpreted as an array index,
`arr[np.array(seq)]`, which will result either in an error or a different result.
    return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

```
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0f3fbd0f28>
```



People rate differently - some give only 5 stars reviews, some are more harsh than others, for some only perfect book should get 5 star rating and so on. Generally, people tend to use only the upper part of the scale. Such tendencies can be observed on mean user rating distribution plot.

To correct for biases caused by varying mean ratings of different users and items(i.e. long or hard-to-watch movies can also be rated far lower than others) special factors are introduced in the form of user bias, item bias or baseline. [Section 5.2.1 Recommender Systems Handbook, Ricci]

5 Train and test split

```
In [9]: from sklearn.model_selection import train_test_split
```

```
In [10]: train_df, test_df = train_test_split(ratings_df, test_size=0.1, random_state=44)
```

Some used methods do not generalize well for new(unseen) users and items, so we have to make sure that training test contains all users and items.

```
In [11]: set(train_df['user_id'].unique()) == set(ratings_df['user_id'].unique())
```

```
Out[11]: True
```

```
In [12]: set(train_df['book_id'].unique()) == set(ratings_df['book_id'].unique())
```

```
Out[12]: True
```

```
In [13]: train_df.groupby('user_id')['book_id'].count().describe()
```

```
Out[13]: count    53424.000000
         mean      100.681922
         std       23.671726
         min       17.000000
         25%       86.000000
         50%      100.000000
         75%      115.000000
         max       182.000000
         Name: book_id, dtype: float64
```

```
In [14]: train_df.groupby('book_id')['user_id'].count().describe()
```

```
Out[14]: count    10000.000000
         mean      537.883100
         std      1140.646885
         min       8.000000
         25%      140.000000
         50%      223.000000
         75%      454.250000
         max     20508.000000
         Name: user_id, dtype: float64
```