01-Exploring-data

January 24, 2021

0.1 Problem statement

0.2 Business Objectives

- Learn from data and and build a recommender that recommends best TV shows to users, based on self & others behaviour
- Predict the rating that a user would give to a movie that he has not yet rated.
- Minimize the difference between predicted and actual rating (RMSE and MAPE).

1 Data Selection

The dataset used in this project comes directly from Netflix. The data was used in the Netflix Prize open competition for the best algorithm to predict user ratings for films.

Firstly,the data was loaded into the notebook and the scope, type and properties of the data was examined.

The documentation attached to the data includes the following descriptions of the data files:

Type of Data: * There are 17770 unique movie IDs. * There are 480189 unique user IDs. * There are ratings. Ratings are on a five star (integral) scale from 1 to 5.

Data Overview

Training data:

- combined_data_1.txt
- combined_data_2.txt
- combined_data_3.txt
- combined_data_4.txt

The first line of each file contains the movie id followed by a colon. Each subsequent line in the file corresponds to a rating from a customer and its date in the following format: **<CustomerID,Rating,Date>**

Movies file description: * movie_titles.csv contains movie information in the following format: **<MovieID,YearOfRelease,Title>** * **MovieID** do not correspond to actual Netflix movie id or IMDB movie id * **YearOfRelease** can range from 1890 to 2005 and may correspond to the release of corresponding DVD, not necessarily its theaterical release. * **Title** is the Netflix movie title and may not correspond to titles used on other sites. * **Titles** are in English.

2 Data Preprocessing and Transformation

2.0.1 1. Reading and storing the data

[1]:
```
# To store the data
import pandas as pd

# To do linear algebra
import numpy as np

# Libraries for visualisations in the notebook
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import math
import re
import os
from datetime import datetime

from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
import math

# utils import
from fuzzywuzzy import fuzz

import matplotlib.dates as dates
import matplotlib.dates as mdates
```

3 Loading Rating Data

In the following projekt the Kaggle Netflix Prize data is used. [1] There are multiple data files for different purposes. The following data files are used in this project:

Training data:

- combined_data_1.txt
- combined_data_2.txt
- combined_data_3.txt
- combined_data_4.txt

Movie titles data file: * movie_titles.csv

[2]:
```
def readFile(file_path):
    data_dict = {'Cust_ID' : [], 'Movie_ID' : [], 'Rating' : [], 'Date' : []}
    f = open(file_path, "r")
    count = 0
    for line in f:
```

```
        count += 1
        #if count > rows:
         #    break

        if ':' in line:
            movidId = line[:-2] # remove the last character ':'
            movieId = int(movidId)
        else:
            customerID, rating, date = line.split(',')
            data_dict['Cust_ID'].append(customerID)
            data_dict['Movie_ID'].append(movieId)
            data_dict['Rating'].append(rating)
            data_dict['Date'].append(date.rstrip("\n"))
    f.close()

    return pd.DataFrame(data_dict)
```

[3]:
```
# Records from each training data file is loaded into memory as Panda DataFrame␣
↪as follows:

df1 = readFile('./data/Netflix_prize_data/combined_data_1.txt')
df2 = readFile('./data/Netflix_prize_data/combined_data_2.txt')
df3 = readFile('./data/Netflix_prize_data/combined_data_3.txt')
df4 = readFile('./data/Netflix_prize_data/combined_data_4.txt')
```

[4]:
```
# Combine the different DataFrames of training data into one as follows:

df_ratings=df1.copy()
df_ratings=df_ratings.append(df2)
df_ratings=df_ratings.append(df3)
df_ratings=df_ratings.append(df4)
```

[5]:
```
# Check the columns types
df_ratings.dtypes
```

[5]:
```
Cust_ID     object
Movie_ID     int64
Rating      object
Date        object
dtype: object
```

[6]:
```
# Change type of some columns
df_ratings['Rating'] = df_ratings['Rating'].astype(float)

df_ratings["Date"] = pd.to_datetime(df_ratings["Date"])
#df.sort_values(by = "Date", inplace = True)
```

[7]:
```
# Basic statistics of Rating  column
df_ratings['Rating'].describe()
```

[7]:
```
count    1.004805e+08
mean     3.604290e+00
std      1.085219e+00
min      1.000000e+00
25%      3.000000e+00
50%      4.000000e+00
75%      4.000000e+00
max      5.000000e+00
Name: Rating, dtype: float64
```

[8]:
```
# Check the dimension of the DataFrame
print ("Dimension: "+ str (df_ratings.shape))
print ("Number of rows: "+ str (df_ratings.shape[0]) )
print ("Number of columns: "+ str (df_ratings.shape[1]) )
```

```
Dimension: (100480507, 4)
Number of rows: 100480507
Number of columns: 4
```

3.0.1 Data cleaning

In this step, the primary focus is on handling missing data, noisy data, detection, and removal of outliers, minimizing duplication and computed biases within the data.

Checking for missings

[9]:
```
print("Number of NaN values = "+str(df_ratings.isnull().sum()))
```

```
Number of NaN values = Cust_ID     0
Movie_ID    0
Rating      0
Date        0
dtype: int64
```

[10]:
```
# Check for missings
print(len(df_ratings.columns[df_ratings.isna().any()])/len(df_ratings.columns))
print(df_ratings.isnull().sum().sum()/np.product(df_ratings.shape))
```

```
0.0
0.0
```

Checking for duplicates

[11]:
```
# Check for duplicates
duplicates = df_ratings.duplicated(["Cust_ID", "Movie_ID", "Rating"])
print("Number of duplicate rows = "+str(duplicates.sum()))
```

```
Number of duplicate rows = 0
```

[12]:
```
# Splitting the Date column into year and month
df_ratings['year'] = pd.DatetimeIndex(df_ratings['Date']).year
df_ratings['month'] = pd.DatetimeIndex(df_ratings['Date']).month
```

[13]:
```
df_ratings.head()
```

[13]:

	Cust_ID	Movie_ID	Rating	Date	year	month
0	1488844	1	3.0	2005-09-06	2005	9
1	822109	1	5.0	2005-05-13	2005	5
2	885013	1	4.0	2005-10-19	2005	10
3	30878	1	4.0	2005-12-26	2005	12
4	823519	1	3.0	2004-05-03	2004	5

4 Loading Movie Data

To read the file correctly, you should pass the encoding that the file was written. We also want to give the columns names:

[14]:
```
df_movies = pd.read_csv('./data/Netflix_prize_data/movie_titles.csv',
                        encoding = "ISO-8859-1",   # movie_titles.csv:␣
 ↪text/plain; charset=iso-8859-1
                        header = None,
                        names= ['Movie_ID', 'YearOfRelease','Movie'])
df_movies.head()
```

[14]:

	Movie_ID	YearOfRelease	Movie
0	1	2003.0	Dinosaur Planet
1	2	2004.0	Isle of Man TT 2004 Review
2	3	1997.0	Character
3	4	1994.0	Paula Abdul's Get Up & Dance
4	5	2004.0	The Rise and Fall of ECW

[15]:
```
# Checking the data type
df_movies.dtypes
```

[15]:
```
Movie_ID          int64
YearOfRelease   float64
Movie            object
dtype: object
```

[16]:
```
df_movies.shape
```

[16]:
```
(17770, 3)
```

[17]:
```
# in order to convert non-finite values (NA or inf) to int
df_movies = df_movies.fillna(0)
df_movies['YearOfRelease']=df_movies['YearOfRelease'].astype(int)
df_movies.head ()
```

[17]:

	Movie_ID	YearOfRelease	Movie
0	1	2003	Dinosaur Planet
1	2	2004	Isle of Man TT 2004 Review
2	3	1997	Character
3	4	1994	Paula Abdul's Get Up & Dance
4	5	2004	The Rise and Fall of ECW

[18]:
```
movies_per_year = df_movies.groupby('YearOfRelease')['Movie'].count().
↪sort_values()
movies_per_year
```

[18]:
```
YearOfRelease
1896       1
1909       1
1914       2
1918       2
1923       2
          …
2001    1184
2000    1234
2003    1271
2002    1310
2004    1436
Name: Movie, Length: 95, dtype: int64
```

4.1 Data cleaning

Checking for missings

[19]:
```
print("Number of NaN values = "+str(df_movies.isnull().sum()))
```

```
Number of NaN values = Movie_ID         0
YearOfRelease    0
Movie            0
dtype: int64
```

[20]:
```
print(len(df_movies.columns[df_movies.isna().any()])/len(df_movies.columns))
print(df_movies.isnull().sum().sum()/np.product(df_movies.shape))
```

```
0.0
0.0
```

Checking for duplicates

[21]:
```
# Check for duplicates
duplicates = df_movies.duplicated(["Movie_ID", "Movie"])
print("Number of duplicate rows = "+str(duplicates.sum()))
```

```
Number of duplicate rows = 0
```

[22]:
```
# Combine all dataframes
df_final = df_ratings.join(df_movies.set_index('Movie_ID'),on="Movie_ID")
df_final
```

[22]:

	Cust_ID	Movie_ID	Rating	Date	year	month	YearOfRelease	Movie
0	1488844	1	3.0	2005-09-06	2005	9	2003	Dinosaur Planet
1	822109	1	5.0	2005-05-13	2005	5	2003	Dinosaur Planet
2	885013	1	4.0	2005-10-19	2005	10	2003	Dinosaur Planet
3	30878	1	4.0	2005-12-26	2005	12	2003	Dinosaur Planet
4	823519	1	3.0	2004-05-03	2004	5	2003	Dinosaur Planet
…	…	…	…	…	…	…	…	…
26847518	1790158	17770	4.0	2005-11-01	2005	11	2003	Alien Hunter
26847519	1608708	17770	3.0	2005-07-19	2005	7	2003	Alien Hunter
26847520	234275	17770	1.0	2004-08-07	2004	8	2003	Alien Hunter
26847521	255278	17770	4.0	2004-05-28	2004	5	2003	Alien Hunter
26847522	453585	17770	2.0	2005-03-10	2005	3	2003	Alien Hunter

```
[100480507 rows x 8 columns]
```

[23]:
```
df_final.to_csv (r'./data/Netflix_prize_data/df_final.csv', index = False,␣
↪header=True)
```

[24]:
```
#df_final.index = np.arange(0,len(df_final))
#df_final.head(10)
```

5 Exploring Data

5.1 Basic statistic

[25]:
```
rating_data=str(df_ratings.shape[0])
num_users=str(len(np.unique(df_ratings["Cust_ID"])))
num_mov = str(len(np.unique(df_ratings["Movie_ID"])))
print('There are {} unique users and {} unique movies in this data set'.
 ↪format(num_users, num_mov))
```

```
There are 480189 unique users and 17770 unique movies in this data set
```

[26]:
```
print ("Ratings per year")
# Calculate the ratings per year
rating_per_year = df_final.groupby (by = 'year')["Rating"].count().
 ↪sort_values(ascending = True)
#rating_per_year = df_ratings.groupby (by = 'year')["Rating"].count().
 ↪sort_values(ascending = True)
rating_per_year
```

```
Ratings per year
```

[26]:
```
year
1999        2178
2000      924443
2001     1769031
2002     4342871
2003     9985337
2004    30206574
2005    53250073
Name: Rating, dtype: int64
```

[27]:
```
# Calculate the avarage rating per year
avg_rating_per_year= df_ratings.groupby (by= 'year')['Rating'].mean()
avg_rating_per_year
```

[27]:
```
year
1999    3.337006
2000    3.365216
2001    3.390736
2002    3.381816
2003    3.406285
2004    3.594968
2005    3.676107
Name: Rating, dtype: float64
```

[28]:
```
df_movies.head()
```

[28]:

	Movie_ID	YearOfRelease	Movie
0	1	2003	Dinosaur Planet
1	2	2004	Isle of Man TT 2004 Review
2	3	1997	Character
3	4	1994	Paula Abdul's Get Up & Dance
4	5	2004	The Rise and Fall of ECW

[29]:
```
print ("Rated movies per year")
# Calculate the rated movies per year
movie_rated_per_year = df_final.groupby(by='YearOfRelease')['Movie_ID'].
 ↪nunique().sort_values()
movie_rated_per_year
```

```
Rated movies per year
```

[29]:
```
YearOfRelease
1896       1
1909       1
1914       2
1918       2
1923       2
         …
2001    1184
2000    1234
2003    1271
2002    1310
2004    1436
Name: Movie_ID, Length: 95, dtype: int64
```

[30]:
```
movie_rated_per_month = df_final.groupby(by='Date')['Movie'].count().
 ↪sort_values()
movie_rated_per_month.head()
```

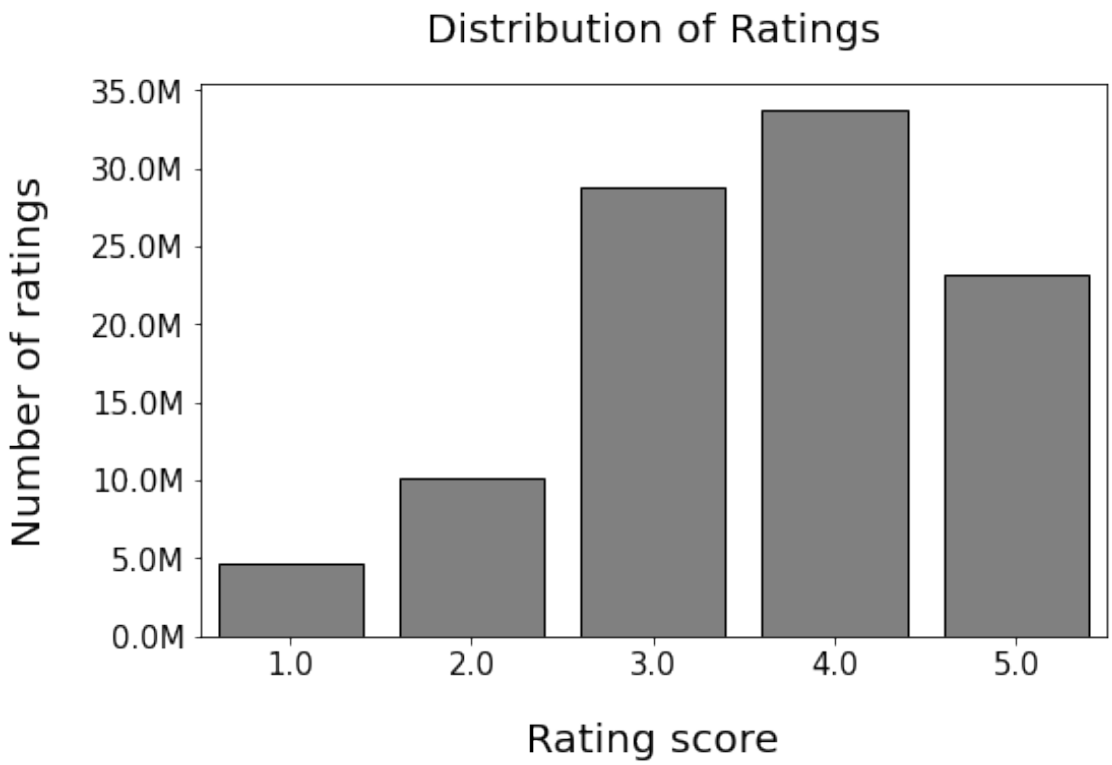
[30]:
```
Date
1999-12-15     5
1999-12-20     6
1999-12-06    12
1999-12-14    15
1999-12-26    20
Name: Movie, dtype: int64
```

[31]:
```
# for showing the data in Millions
def changingLabels(number):
    return str(number/10**6) + "M"
```

5.2 Distribution of ratings

[32]:
```
# Check the distribution of the ratings
plt.figure(figsize = (8, 5))
#sns.set_palette("mako")
ax = sns.countplot(x="Rating", data=df_final,color='grey',  edgecolor='black')

# for showing the data in Millions
ax.set_yticklabels([changingLabels(num) for num in ax.get_yticks()])

plt.tick_params(labelsize = 15)
plt.title("Distribution of Ratings", fontsize = 20, pad =20)
plt.xlabel("Rating score", fontsize = 20, labelpad = 20)
plt.ylabel("Number of ratings", fontsize = 20, labelpad = 20)
plt.show()
```

```
<ipython-input-32-19c5e15c4289>:7: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_yticklabels([changingLabels(num) for num in ax.get_yticks()])
```



[33]:
```
df_final["Date"] = pd.to_datetime(df_final["Date"])
```

6 Analysis of Ratings given by user

[34]:
```
df_final.head()
```

[34]:

	Cust_ID	Movie_ID	Rating	Date	year	month	YearOfRelease	Movie
0	1488844	1	3.0	2005-09-06	2005	9	2003	Dinosaur Planet
1	822109	1	5.0	2005-05-13	2005	5	2003	Dinosaur Planet
2	885013	1	4.0	2005-10-19	2005	10	2003	Dinosaur Planet
3	30878	1	4.0	2005-12-26	2005	12	2003	Dinosaur Planet
4	823519	1	3.0	2004-05-03	2004	5	2003	Dinosaur Planet

[35]:
```
print("Ratings per user")
# Get the ratings given by user
cnt_rating_per_user = df_final.groupby(by = "Cust_ID")["Movie_ID"].count().
 ↪sort_values(ascending = False)
cnt_rating_per_user.head()
```

```
Ratings per user
```

[35]:
```
Cust_ID
305344     17653
387418     17436
2439493    16565
1664010    15813
2118461    14831
Name: Movie_ID, dtype: int64
```

[36]:
```
cnt_rating_per_user.describe()
```

[36]:
```
count    480189.000000
mean        209.251997
std         302.339155
min           1.000000
25%          39.000000
50%          96.000000
75%         259.000000
max       17653.000000
Name: Movie_ID, dtype: float64
```

[37]:
```
print("Users with highest number of ratings")
cnt_rating_per_user_20 = cnt_rating_per_user[:20]
cnt_rating_per_user_20
```

```
Users with highest number of ratings
```

[37]:
```
Cust_ID
305344     17653
387418     17436
2439493    16565
1664010    15813
2118461    14831
1461435     9822
1639792     9767
1314869     9740
2606799     9064
1932594     8880
2056022     8387
1114324     8322
752642      7481
491531      7257
1663888     7080
727242      6997
1403217     6844
1473980     6790
798296      6740
716173      6736
Name: Movie_ID, dtype: int64
```

[38]:
```
print("Users with lowest number of ratings")
cnt_rating_per_user_last_20 = cnt_rating_per_user[-20:]
cnt_rating_per_user_last_20
```

```
Users with lowest number of ratings
```

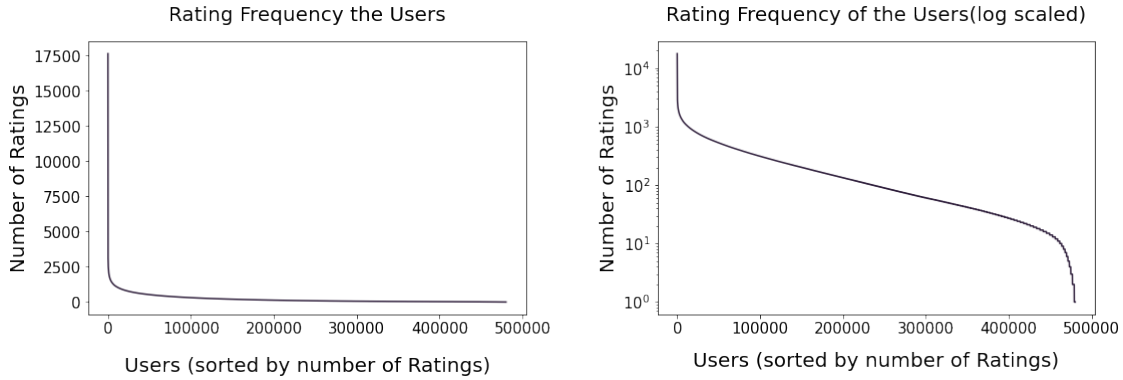
[38]:
```
Cust_ID
2605198    1
2595528    1
1133804    1
108085     1
1885331    1
2346389    1
2382502    1
1626036    1
2130262    1
1636338    1
2086661    1
```

```
2236089    1
1246198    1
932870     1
1763216    1
1560179    1
339212     1
134227     1
1245917    1
103906     1
Name: Movie_ID, dtype: int64
```

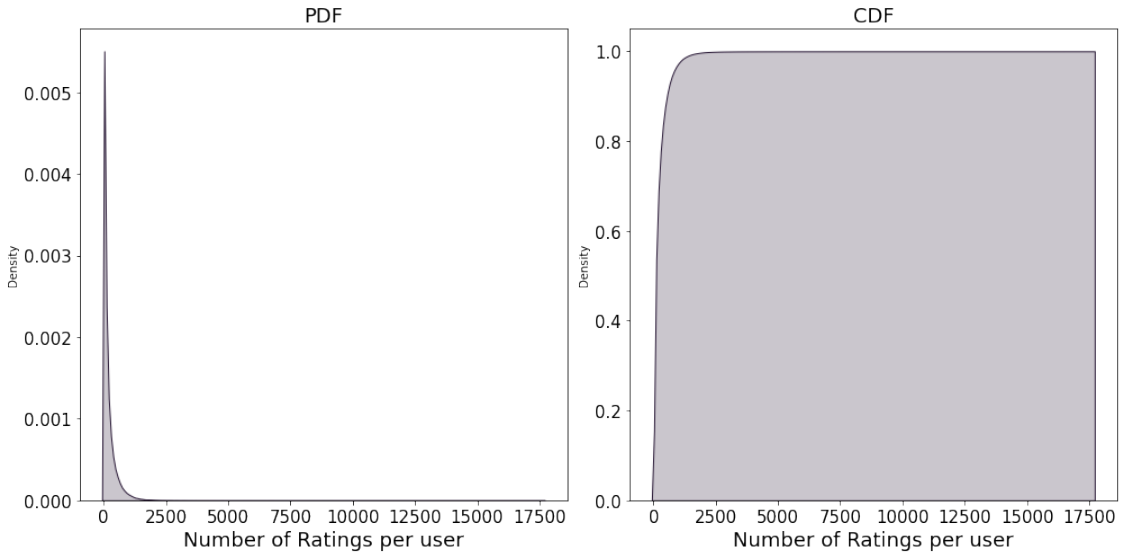
[39]:
```
sns.set_palette("mako")
```

[40]:
```
# Make subplots that are next to each other
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(18, 5))

sns.set_palette("mako")

# plot rating frequency of all movies
cnt_rating_per_user.reset_index(drop=True).plot(ax=ax1); #␣
 ↪reset_index(drop=True)

# Add title and labels
ax1.set_title("Rating Frequency the Users",fontsize = 20,pad =20); #distance␣
 ↪between title
ax1.set_xlabel("Users (sorted by number of Ratings)",fontsize = 20, labelpad=20)
ax1.set_ylabel("Number of Ratings",fontsize = 20)
#ax1.set_yticklabels([changingLabels(num) for num in ax.get_yticks()])

# plot rating frequency of all movies in log scale
cnt_rating_per_user.reset_index(drop=True).plot(logy = True, ax=ax2);

# Add title and labels
ax2.set_title("Rating Frequency of the Users(log scaled)",fontsize = 20, pad␣
 ↪=20);
ax2.set_xlabel("Users (sorted by number of Ratings)",fontsize = 20,␣
 ↪labelpad=15) #fontdict=dict(weight='bold')
ax2.set_ylabel("Number of Ratings",fontsize = 20)

fig.subplots_adjust(wspace=0.3) #the amount of width reserved for space between␣
 ↪subplots,
# Remove empty white space around the plot

# change the fontsize of minor ticks label
ax1.tick_params(axis='both', which='major', labelsize=15)
ax1.tick_params(axis='both', which='minor', labelsize=15)
ax2.tick_params(axis='both', which='major', labelsize=15)
```

```
ax2.tick_params(axis='both', which='minor', labelsize=15)

plt.show()
```



[41]:
```
fig, axes = plt.subplots(nrows = 1, ncols = 2, figsize=(14,7))

# Generate the probability density function (PDF)
sns.kdeplot(cnt_rating_per_user.values, shade = True, ax = axes[0])
axes[0].set_title("PDF", fontsize = 18)
axes[0].set_xlabel("Number of Ratings per user", fontsize = 18)
axes[0].tick_params(labelsize = 15)

# Generate the cumulative distribution (CDF)
sns.kdeplot(cnt_rating_per_user.values, shade = True, cumulative = True, ax =␣
 ↪axes[1])
axes[1].set_title("CDF", fontsize = 18)
axes[1].set_xlabel("Number of Ratings per user", fontsize = 18)
axes[1].tick_params(labelsize = 15)

fig.subplots_adjust(wspace=2)
plt.tight_layout()
plt.show()
```



- Above PDF graph shows that almost all of the users give very few ratings. There are very few users who's ratings count is high.
- Similarly, above CDF graph shows that almost 99% of users give very few ratings.

[42]:
```
df_final.head()
```

[42]:

	Cust_ID	Movie_ID	Rating	Date	year	month	YearOfRelease	Movie
0	1488844	1	3.0	2005-09-06	2005	9	2003	Dinosaur Planet
1	822109	1	5.0	2005-05-13	2005	5	2003	Dinosaur Planet
2	885013	1	4.0	2005-10-19	2005	10	2003	Dinosaur Planet
3	30878	1	4.0	2005-12-26	2005	12	2003	Dinosaur Planet
4	823519	1	3.0	2004-05-03	2004	5	2003	Dinosaur Planet

[43]:
```
cnt_user_ratings_per_year = df_final.groupby(by = "year")["Rating"].count().
↪sort_values(ascending = False)
```

[44]:
```
print("User ratings per year:")
cnt_user_ratings_per_year
```

```
User ratings per year:
```

[44]:
```
year
2005    53250073
2004    30206574
2003     9985337
2002     4342871
2001     1769031
2000      924443
1999        2178
Name: Rating, dtype: int64
```

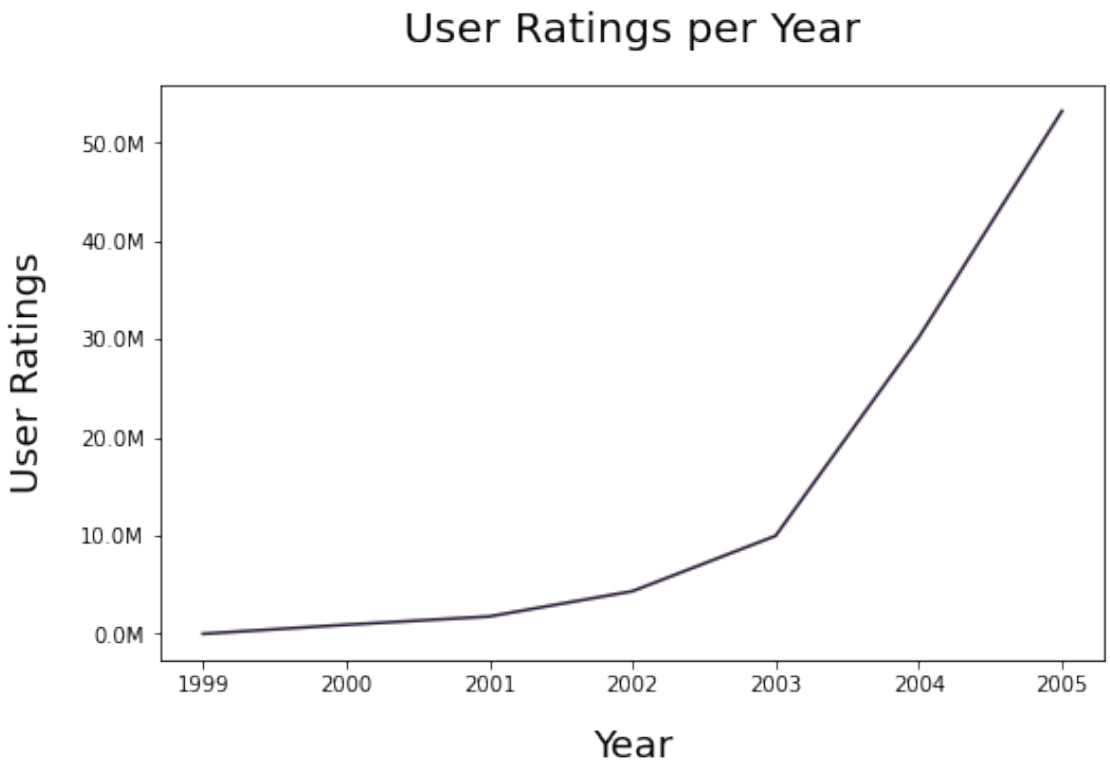
[45]:
```
cnt_user_ratings_per_year.describe()
```

[45]:
```
count    7.000000e+00
mean     1.435436e+07
std      2.011726e+07
min      2.178000e+03
25%      1.346737e+06
50%      4.342871e+06
75%      2.009596e+07
max      5.325007e+07
Name: Rating, dtype: float64
```

[46]:
```
plt.figure(figsize = (8,5))
ax=cnt_user_ratings_per_year.plot()
ax.set_title("User Ratings per Year",fontsize = 20,pad =20)
ax.set_xlabel("Year",fontsize = 18, labelpad=15)
ax.set_ylabel("User Ratings",fontsize = 18, labelpad=15)
ax.set_yticklabels([changingLabels(num) for num in ax.get_yticks()])
```

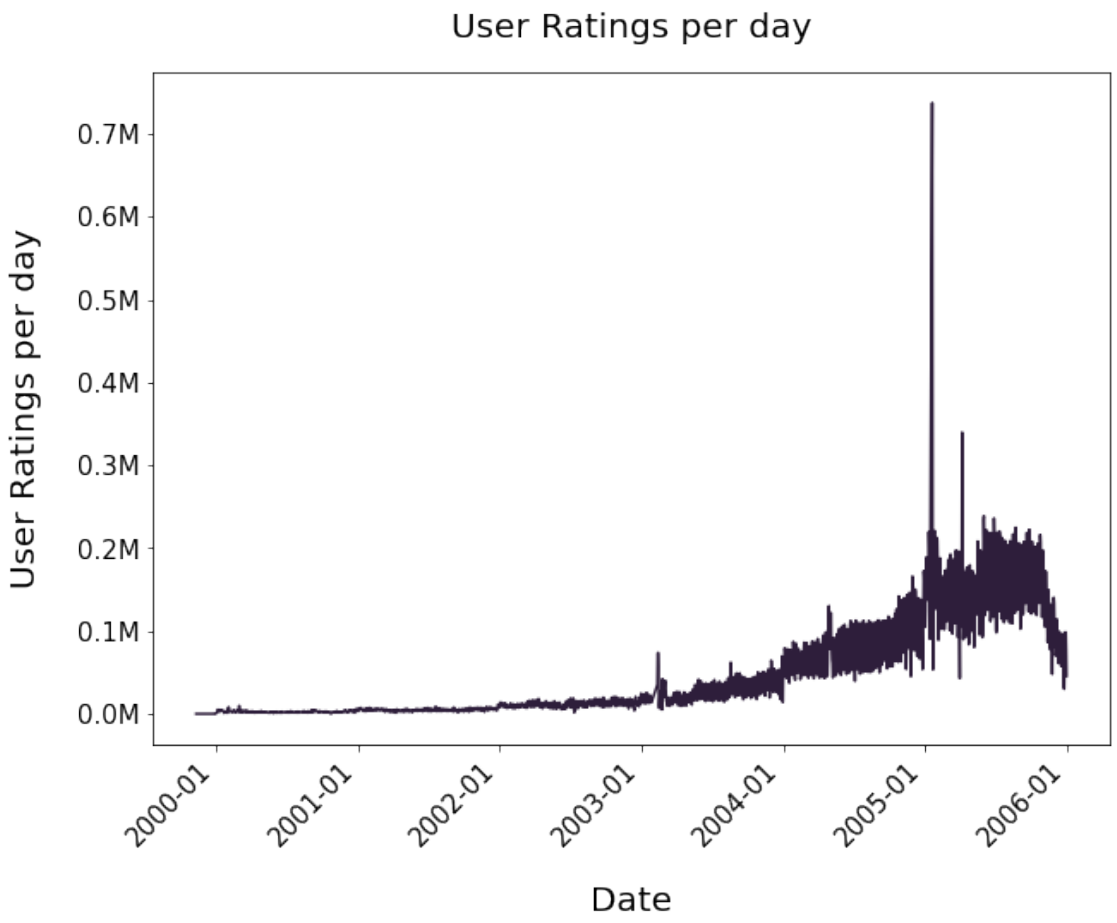
```
<ipython-input-46-6f7dcc70d416>:6: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_yticklabels([changingLabels(num) for num in ax.get_yticks()])
```

[46]:
```
[Text(0, -10000000.0, '-10.0M'),
 Text(0, 0.0, '0.0M'),
 Text(0, 10000000.0, '10.0M'),
 Text(0, 20000000.0, '20.0M'),
 Text(0, 30000000.0, '30.0M'),
 Text(0, 40000000.0, '40.0M'),
 Text(0, 50000000.0, '50.0M'),
 Text(0, 60000000.0, '60.0M')]
```



[47]:
```
plt.figure(figsize = (10,8))
ax = df_final.groupby(by = "Date")["Rating"].count().plot()
ax.set_title("User Ratings per day", fontsize = 20, pad =20)
ax.set_xlabel("Date", fontsize = 20,labelpad=20)
ax.set_ylabel("User Ratings per day", fontsize = 20,labelpad=20)
ax.set_yticklabels([changingLabels(num) for num in ax.get_yticks()])
plt.tick_params(labelsize = 15)
ax.xaxis.set_major_formatter(mdates.DateFormatter ("%Y-%m"))
ax.xaxis.set_minor_formatter(mdates.DateFormatter ("%Y-%m"))
_=plt.xticks(rotation=45)
plt.show()
```

```
<ipython-input-47-e58f6fa2299f>:6: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_yticklabels([changingLabels(num) for num in ax.get_yticks()])
```



[48]:
```
print ("Avarage User rating per year")
cnt_user_ratings_per_year = df_final.groupby(by = "year")["Rating"].mean().
 ↪sort_values(ascending = False)
cnt_user_ratings_per_year
```

```
Avarage User rating per year
```

[48]:
```
year
2005    3.676107
2004    3.594968
2003    3.406285
2001    3.390736
2002    3.381816
2000    3.365216
1999    3.337006
Name: Rating, dtype: float64
```

[49]:
```
cnt_user_ratings_per_year.describe()
```

[49]:
```
count    7.000000
mean     3.450305
std      0.130491
min      3.337006
25%      3.373516
50%      3.390736
75%      3.500627
max      3.676107
Name: Rating, dtype: float64
```

[50]:
```
df_final.head()
```

[50]:
```
   Cust_ID  Movie_ID  Rating       Date  year  month  YearOfRelease  \
0  1488844         1     3.0 2005-09-06  2005      9           2003
1   822109         1     5.0 2005-05-13  2005      5           2003
2   885013         1     4.0 2005-10-19  2005     10           2003
3    30878         1     4.0 2005-12-26  2005     12           2003
4   823519         1     3.0 2004-05-03  2004      5           2003

             Movie
0  Dinosaur Planet
1  Dinosaur Planet
2  Dinosaur Planet
3  Dinosaur Planet
4  Dinosaur Planet
```

[51]:
```
print("Raters per year:")
cnt_raters_per_year = df_final.groupby(by='year')['Cust_ID'].nunique().
 ↪sort_values()
cnt_raters_per_year
```

```
Raters per year:
```

[51]:
```
year
1999        77
2000      8227
2001     19801
2002     51732
2003    117500
2004    259407
2005    451435
Name: Cust_ID, dtype: int64
```

[52]:
```
cnt_raters_per_year.describe()
```

[52]:
```
count         7.000000
mean     129739.857143
```

```
std      168424.442029
min          77.000000
25%       14014.000000
50%       51732.000000
75%      188453.500000
max      451435.000000
Name: Cust_ID, dtype: float64
```

7 Analysis of ratings pro Movie

7.1 Plot rating frequency of all movies

[53]:
```
df_movies.head()
```

[53]:

	Movie_ID	YearOfRelease	Movie
0	1	2003	Dinosaur Planet
1	2	2004	Isle of Man TT 2004 Review
2	3	1997	Character
3	4	1994	Paula Abdul's Get Up & Dance
4	5	2004	The Rise and Fall of ECW

[54]:
```
print("Released movies per year")
cnt_movies_rel_year =df_movies.groupby('YearOfRelease')['Movie_ID'].count().
 ↪sort_values(ascending =False)
cnt_movies_rel_year
```

```
Released movies per year
```

[54]:
```
YearOfRelease
2004    1436
2002    1310
2003    1271
2000    1234
2001    1184
        …
1923       2
1918       2
1914       2
1909       1
1896       1
Name: Movie_ID, Length: 95, dtype: int64
```

[55]:
```
print("Ratings per Movie:")
# get rating frequency
cnt_ratings_per_movie = pd.DataFrame(df_final.groupby('Movie_ID')['Rating'].
 ↪count()
```

```
                                        .sort_values(ascending =False))

cnt_ratings_per_movie.sort_values(by='Movie_ID').head()
```

Ratings per Movie:

[55]:

Movie_ID	Rating
1	547
2	145
3	2012
4	142
5	1140

[56]:
```
print("Movies with highest number of ratings")
cnt_ratings_per_movie_20 = cnt_ratings_per_movie[:20]
cnt_ratings_per_movie_20
```

Movies with highest number of ratings

[56]:

Movie_ID	Rating
5317	232944
15124	216596
14313	200832
15205	196397
1905	193941
6287	193295
11283	181508
16377	181426
16242	178068
12470	177556
15582	176539
9340	173596
6972	171991
12317	164792
2152	162597
3860	160454
15107	160326
6037	158601
4432	156183
5496	155714

[57]:
```
print("Movies with lowest number of ratings")
cnt_ratings_per_movie_last_20 = cnt_ratings_per_movie[-20:]
cnt_ratings_per_movie_last_20
```

Movies with lowest number of ratings

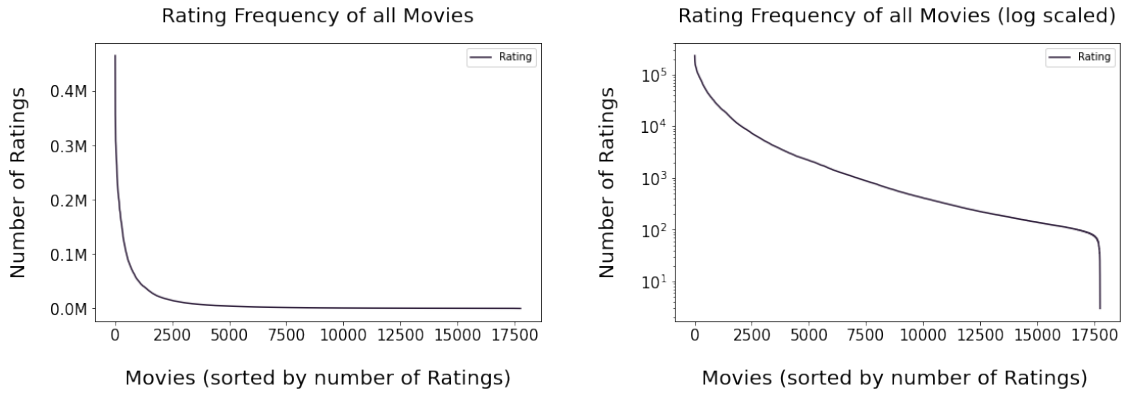
[57]:

Movie_ID	Rating
8858	36
16155	36
7717	35
9507	34
4614	33
10581	31
13405	29
9124	27
11936	27
10597	26
8964	25
16875	23
12418	22
4711	22
8146	14
4806	13
6256	10
11344	10
11148	5
13755	3

[58]:
```
# Make subplots that are next to each other
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(18, 5))

# plot rating frequency of all movies
cnt_ratings_per_movie.reset_index(drop=True).plot(ax=ax1);
# Add title and labels
ax1.set_title("Rating Frequency of all Movies",fontsize = 20,pad =20);␣
 ↪#distance between title
ax1.set_xlabel("Movies (sorted by number of Ratings)",fontsize = 20, labelpad␣
 ↪=25)
ax1.set_ylabel("Number of Ratings",fontsize = 20, labelpad =20)
ax1.set_yticklabels([changingLabels(num) for num in ax.get_yticks()])

# plot rating frequency of all movies in log scale
cnt_ratings_per_movie.reset_index(drop=True).plot(logy = True, ax=ax2);

# Add title and labels
ax2.set_title("Rating Frequency of all Movies (log scaled)",fontsize = 20, pad␣
 ↪=20);
ax2.set_xlabel("Movies (sorted by number of Ratings) ",fontsize = 20,␣
 ↪labelpad=25) #fontdict=dict(weight='bold')
ax2.set_ylabel("Number of Ratings",fontsize = 20,labelpad=20)
```

```
fig.subplots_adjust(wspace=0.3) #the amount of width reserved for space between␣
 ↪subplots,
# Remove empty white space around the plot
#plt.tight_layout()

# change the fontsize of minor ticks label
ax1.tick_params(axis='both', which='major', labelsize=15)
ax1.tick_params(axis='both', which='minor', labelsize=15)
ax2.tick_params(axis='both', which='major', labelsize=15)
ax2.tick_params(axis='both', which='minor', labelsize=15)
plt.show()
```

```
<ipython-input-58-2908e1f7a721>:10: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax1.set_yticklabels([changingLabels(num) for num in ax.get_yticks()])
```



The distribution of ratings among movies often satisfies a property in real-world settings, which is referred to as the long-tail property. According to this property, only a small fraction of the items are rated frequently. Such items are referred to as popular items. The vast majority of items are rated rarely. This results in a highly skewed distribution of the underlying ratings.

We can see that roughly X out of Y movies are rated more than 100 times. More interestingly, roughly 20,000 out of 53,889 movies are rated less than only 10 times. Let's look closer by displaying top quantiles of rating counts

[59]:
```
quantiles=cnt_ratings_per_movie['Rating'].quantile(np.arange(0,1.01,0.01))
```

[60]:
```
quantiles[::5]
```

[60]:
```
0.00      3.00
0.05     98.00
0.10    117.00
0.15    137.00
0.20    161.00
```

```
0.25       192.00
0.30       228.00
0.35       281.00
0.40       349.60
0.45       439.00
0.50       561.00
0.55       754.00
0.60      1006.40
0.65      1360.00
0.70      1948.30
0.75      2667.75
0.80      4040.20
0.85      6638.85
0.90     12303.80
0.95     29199.60
1.00    232944.00
Name: Rating, dtype: float64
```

[61]:
```
df_final["WeekDay"] = df_final.Date.dt.day_name()
df_final["Month"] = df_final.Date.dt.month_name()
```

[62]:
```
df_final.head()
```

[62]:

	Cust_ID	Movie_ID	Rating	Date	year	month	YearOfRelease	Movie	WeekDay	Month
0	1488844	1	3.0	2005-09-06	2005	9	2003	Dinosaur Planet	Tuesday	September
1	822109	1	5.0	2005-05-13	2005	5	2003	Dinosaur Planet	Friday	May
2	885013	1	4.0	2005-10-19	2005	10	2003	Dinosaur Planet	Wednesday	October
3	30878	1	4.0	2005-12-26	2005	12	2003	Dinosaur Planet	Monday	December
4	823519	1	3.0	2004-05-03	2004	5	2003	Dinosaur Planet	Monday	May

[63]:
```
print("Cumulative User Ratings for days of the week")
cnt_ratings_per_day = df_final.groupby('WeekDay')['Rating'].count().
 ↪sort_values()
cnt_ratings_per_day
```

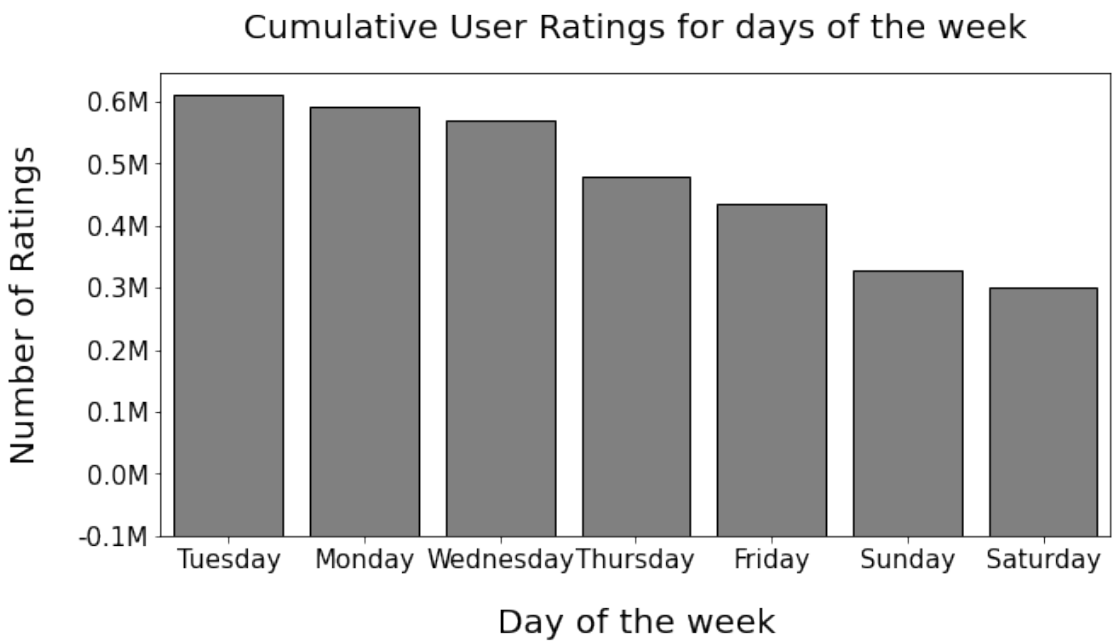
```
Cumulative User Ratings for days of the week
```

[63]:
```
WeekDay
Saturday     10027687
Sunday       10730350
```

```
Friday        13404388
Thursday      14476074
Wednesday     16738311
Monday        17318845
Tuesday       17784852
Name: Rating, dtype: int64
```

[64]:
```
fig = plt.figure(figsize = (10, 5))

axes = sns.countplot(x = "WeekDay", data = df_final,order=df_final['WeekDay'].
 ↪value_counts().index, color='grey',  edgecolor='black')
axes.set_title("Cumulative User Ratings for days of the week", fontsize = 20,␣
 ↪pad =20)
axes.set_xlabel("Day of the week", fontsize = 20, labelpad=20)
axes.set_ylabel("Number of Ratings", fontsize = 20, labelpad=20)
axes.set_yticklabels([changingLabels(num) for num in ax.get_yticks()])
axes.tick_params(labelsize = 15)

plt.show()
```

```
<ipython-input-64-e8f0f2955eb0>:7: UserWarning: FixedFormatter should only be
used together with FixedLocator
  axes.set_yticklabels([changingLabels(num) for num in ax.get_yticks()])
```



[65]:
```
average_ratings_dayofweek = df_final.groupby(by = "WeekDay")["Rating"].mean()
print("Average Ratings on Day of Weeks")
```

```
print(average_ratings_dayofweek)
```

```
Average Ratings on Day of Weeks
WeekDay
Friday       3.605892
Monday       3.597735
Saturday     3.614754
Sunday       3.616449
Thursday     3.604305
Tuesday      3.595808
Wednesday    3.604725
Name: Rating, dtype: float64
```

8 Reference

[1] Netflix Prize data retrieved from: https://www.kaggle.com/netflix-inc/netflix-prize-data