Netflix_Movie_Recommendation_System

April 12, 2023

[1]:
```
# Python Language Version
from platform import python_version
print('Python Language Version Used in This Jupyter Notebook:',␣
↪python_version())
```

```
Python Language Version Used in This Jupyter Notebook: 3.7.4
```

0.1 Netflix's Movie Recommendation System

0.2 Problem Definition

Netflix's goal is to connect people to the movies they love. To help customers find these movies, they have developed a world-class movie recommendation system: CinematchSM. We are going to to predict whether someone will like a movie based on how much they liked or disliked other movies. Netflix uses these predictions to make personal movie recommendations based on each customer's unique tastes. And although Cinematch is doing very well, it can always be improved.

Goals:

1. Predict the rating a user would give to a movie they haven't rated yet.
2. Minimize the difference between predicted and actual assessment (RMSE and MAPE).

0.3 Data source

Netflix provided a lot of anonymized ranking data and a prediction accuracy bar that is 10% better than what Cinematch can do on the same training dataset. Accuracy is a measure of how closely predicted movie ratings match subsequent actual ratings.

Netflix Prize

Dataset

0.4 Loading Packages

[2]:
```
# Imports
import os
import random
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib
```

```python
import matplotlib.pyplot as plt
import scipy
import sklearn
from scipy import sparse
from scipy.sparse import csr_matrix
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics.pairwise import cosine_similarity
from datetime import datetime

# Graphics formatting
matplotlib.use('nbagg')
plt.rcParams.update({'figure.max_open_warning': 0})
sns.set_style('whitegrid')
```

0.5 Loading the Data

To load the data we will perform the following operations:

- 1- Read the lines of all available files.
- 2- Combine all lines from all files into a single file.
- 3- Load the generated file into a pandas dataframe.

[3]:
```python
# Marks the beginning of the file reading execution.
start = datetime.now()
```

[4]:
```python
# We will create a final file called data.csv

# If the file does not exist, we create the file in write mode (w)
if not os.path.isfile('data/data.csv'):

    # Create and open the file for writing
    dataset = open('data/data.csv', mode = 'w')

    # List for lines of files
    linhas = list()

    # File names and paths
    files = ['data/combined_data_1.txt',
             'data/combined_data_2.txt',
             'data/combined_data_3.txt',
             'data/combined_data_4.txt']

    # Loop through each file in the file list
    for file in files:

        # Print
        print("Reading the file {}...".format(file))
```

```
        # With the file open, we extract the lines
        with open(file) as f:

            # Loop through each line of the file
            for row in f:

                # Delete the contents of the list
                del rows[:]

                # Split file lines by end-of-line character
                row = row.strip()

                # If we find a "colon" at the end of the line, we replace it by␣
↪removing the character,
                # because we only want the id of the movie
                if row.endswith(':'):
                    movie_id = row.replace(':', '')

                # If not, create a comprehension list to separate the columns␣
↪by commas
                else:

                    # Separate the columns
                    rows = [x for x in row.split(',')]

                    # Use movie id at zero index position
                    rows.insert(0, movie_id)

                    # Write the result to the new file
                    dataset.write(','.join(linhas))
                    dataset.write('\n')

        print("Finished.\n")

    dataset.close()
```

[5]:
```
# Print the total time
print('Total Time to Load Files:', datetime.now() - start)
```

```
Total Time to Load Files: 0:00:00.024933
```

[6]:
```
print("Creating pandas dataframe from data file.csv...")
df_netflix = pd.read_csv('data/data.csv', sep = ',', names = ['movie', 'user',␣
↪'rating', 'date'])
df_netflix.date = pd.to_datetime(df_netflix.date)
print('Finished.')
```

```
Creating pandas dataframe from data file.csv…
```

```
Finished.
```

```
[7]: # Sorting the dataframe by date
print('Sorting dataframe by date..')
df_netflix.sort_values(by = 'date', inplace = True)
print('Finished.')
```

```
Sorting dataframe by date..
Finished.
```

```
[8]: # Shape
df_netflix.shape
```

```
[8]: (100480507, 4)
```

```
[9]: # Viewing the data
df_netflix.head()
```

[9]:

	movie	user	rating	date
56431994	10341	510180	4	1999-11-11
9056171	1798	510180	5	1999-11-11
58698779	10774	510180	3	1999-11-11
48101611	8651	510180	2	1999-11-11
81893208	14660	510180	2	1999-11-11

0.6 Exploratory Data Analysis

```
[10]: # Data summary
print("Data summary")
print("-"*50)
print("Total Number of Films:", len(np.unique(df_netflix.movie)))
print("Total Number of Users:", len(np.unique(df_netflix.user)))
print("Total Number of Reviews:", df_netflix.shape[0])
```

```
Data summary
--------------------------------------------------
Total Number of Films: 17770
Total Number of Users: 480189
Total Number of Reviews: 100480507
```

```
[11]: # Let's save these two values to use later
total_users = len(np.unique(df_netflix.user))
total_movies = len(np.unique(df_netflix.movie))
```

```
[12]: # Checking the average of the ratings
df_netflix.describe()['rating']
```

```
[12]: 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
```

[13]:
```python
# Checking for missing values
sum(df_netflix.isnull().any())
```

[13]: 0

[14]:
```python
# Checking if we have duplicate values (in this case we don't consider the date)
sum(df_netflix.duplicated(['movie', 'user', 'rating']))
```

[14]: 0

Let's split the data into training and testing before continuing with the exploratory analysis, as some analyzes only make sense for training data. We will use the 80/20 ratio for training/testing.

[15]:
```python
# We will create a dataset on disk with the training data
# That way we don't need to run the whole loading process again each time we
↪run this notebook
if not os.path.isfile('data/training_data.csv'):
    df_netflix.iloc[:int(df_netflix.shape[0] * 0.80)].to_csv("data/
↪training_data.csv", index = False)
```

[16]:
```python
# We will create a dataset on disk with the test data
# That way we don't need to run the whole loading process again each time we
↪run this notebook
if not os.path.isfile('data/test_data.csv'):
    df_netflix.iloc[int(df_netflix.shape[0] * 0.80):].to_csv("data/test_data.
↪csv", index = False)
```

[17]:
```python
# Delete the original dataframe to free memory
del df_netflix
```

[18]:
```python
# Now we load the files into pandas dataframes
df_netflix_train = pd.read_csv("data/training_data.csv", parse_dates = ['date'])
df_netflix_test = pd.read_csv("data/test_data.csv")
```

[19]:
```python
# Training data summary
print("Training data summary")
print("-"*50)
print("Total Number of Films:", len(np.unique(df_netflix_train.movie)))
print("Total Number of Users:", len(np.unique(df_netflix_train.user)))
print("Total Number of Reviews:", df_netflix_train.shape[0])
```

```
Training data summary
--------------------------------------------------
Total Number of Films: 17424
Total Number of Users: 405041
Total Number of Reviews: 80384405
```

[20]:
```python
# Test data summary
print("Test data summary")
print("-"*50)
print("Total Number of Films:", len(np.unique(df_netflix_test.movie)))
print("Total Number of Users:", len(np.unique(df_netflix_test.user)))
print("Total Number of Reviews:", df_netflix_test.shape[0])
```

```
Test data summary
--------------------------------------------------
Total Number of Films: 17757
Total Number of Users: 349312
Total Number of Reviews: 20096102
```

The function below will adjust the measurements in thousands, millions and billions to make the graphs easier to read.

[21]:
```python
# Function for setting the units of measure
def units_setting(num, units = 'M'):
    units = units.lower()
    num = float(num)
    if units == 'k':
        return str(num/10**3) + " K"
    elif units == 'm':
        return str(num/10**6) + " M"
    elif units == 'b':
        return str(num/10**9) +  " B"
```

[22]:
```python
# Supress warnings
import sys
import warnings
if not sys.warnoptions:
    warnings.simplefilter("ignore")
```

Let's check the distribution of ratings.

[23]:
```python
# Plot
fig, ax = plt.subplots()
plt.title('Distribution of Training Data Evaluations', fontsize = 15)
sns.countplot(df_netflix_train.rating)
ax.set_yticklabels([units_setting(item, 'M') for item in ax.get_yticks()])
ax.set_ylabel('Number of Reviews (in Millions)')
plt.show()
```

Does the day of the week influence the user's assessment? Let's add a column with the day of the week and find out.

[24]:
```
# Parameter to avoid warning due to high volume of data
pd.options.mode.chained_assignment = None
```

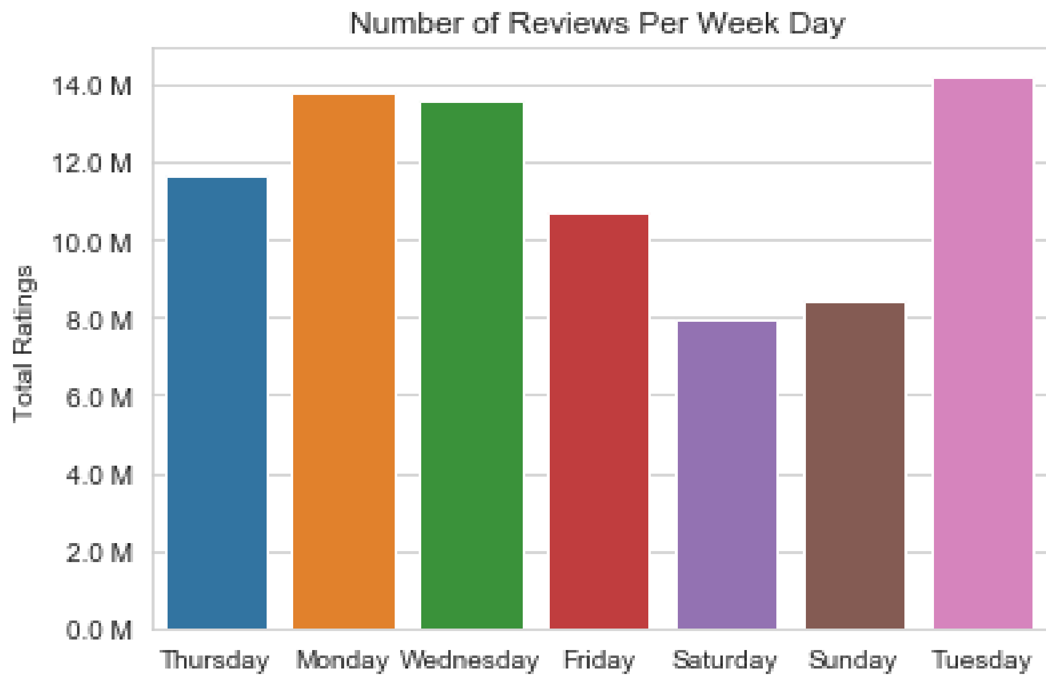
[25]:
```
# Extract the day of the week and write it to a new column
df_netflix_train['week_day'] = df_netflix_train['date'].dt.strftime("%A")
df_netflix_train.head()
```

[25]:

	movie	user	rating	date	week_day
0	10341	510180	4	1999-11-11	Thursday
1	1798	510180	5	1999-11-11	Thursday
2	10774	510180	3	1999-11-11	Thursday
3	8651	510180	2	1999-11-11	Thursday
4	14660	510180	2	1999-11-11	Thursday

[26]:
```
# Plot
fig, ax = plt.subplots()
sns.countplot(x = 'week_day', data = df_netflix_train, ax = ax)
plt.title('Number of Reviews Per Week Day')
plt.ylabel('Total Ratings')
plt.xlabel('')
ax.set_yticklabels([units_setting(item, 'M') for item in ax.get_yticks()])
```

```
plt.show()
```



Let's calculate the average ratings per day of the week.

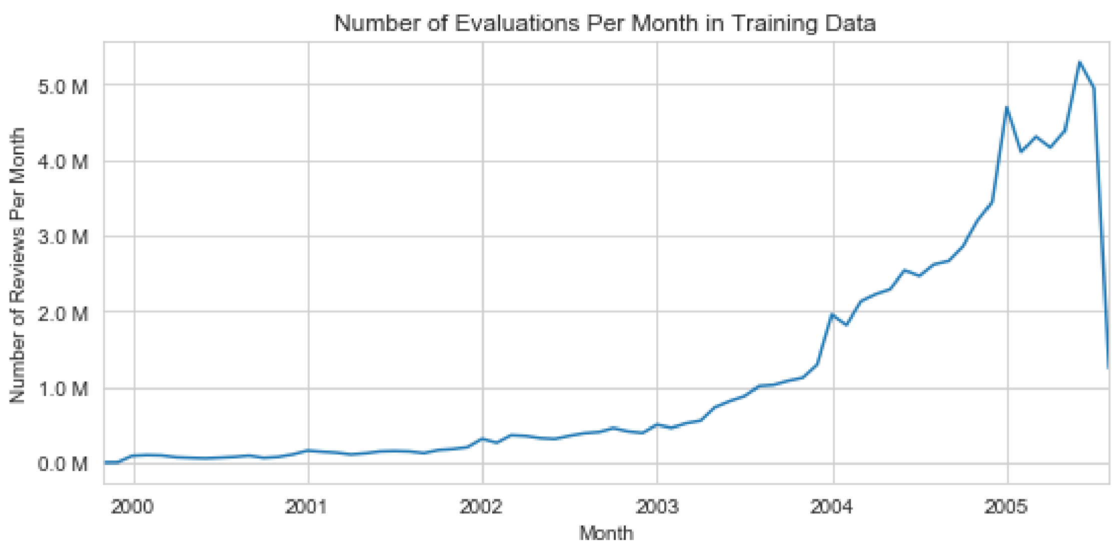
[27]:
```
# Average ratings per week day
average_week_day = df_netflix_train.groupby(by = ['week_day'])['rating'].mean()
print("Average Ratings")
print("-"*30)
print(average_week_day)
print("\n")
```

```
Average Ratings
------------------------------
week_day
Friday       3.585274
Monday       3.577250
Saturday     3.591791
Sunday       3.594144
Thursday     3.582463
Tuesday      3.574438
Wednesday    3.583751
Name: rating, dtype: float64
```

The week days does not seem to have an influence on the user's evaluation.

We will analyze user ratings over time.

[28]:
```
# Plot
fig = plt.figure(figsize = plt.figaspect(.45))
ax = df_netflix_train.resample('m', on = 'date')['rating'].count().plot()
ax.set_title('Number of Evaluations Per Month in Training Data')
plt.xlabel('Month')
plt.ylabel('Number of Reviews Per Month')
ax.set_yticklabels([units_setting(item, 'M') for item in ax.get_yticks()])
plt.show()
```



There is clearly an increase in user ratings over time, either due to more users or because users have learned to use the feature.

Let's check the users who made the most movie ratings.

[29]:
```
# Ratings number per user
ratings_num_per_user = df_netflix_train.groupby(by = 'user')['rating'].count().
↪sort_values(ascending = False)
ratings_num_per_user.head()
```

[29]:
```
user
305344     17112
2439493    15896
387418     15402
1639792     9767
1461435     9447
Name: rating, dtype: int64
```

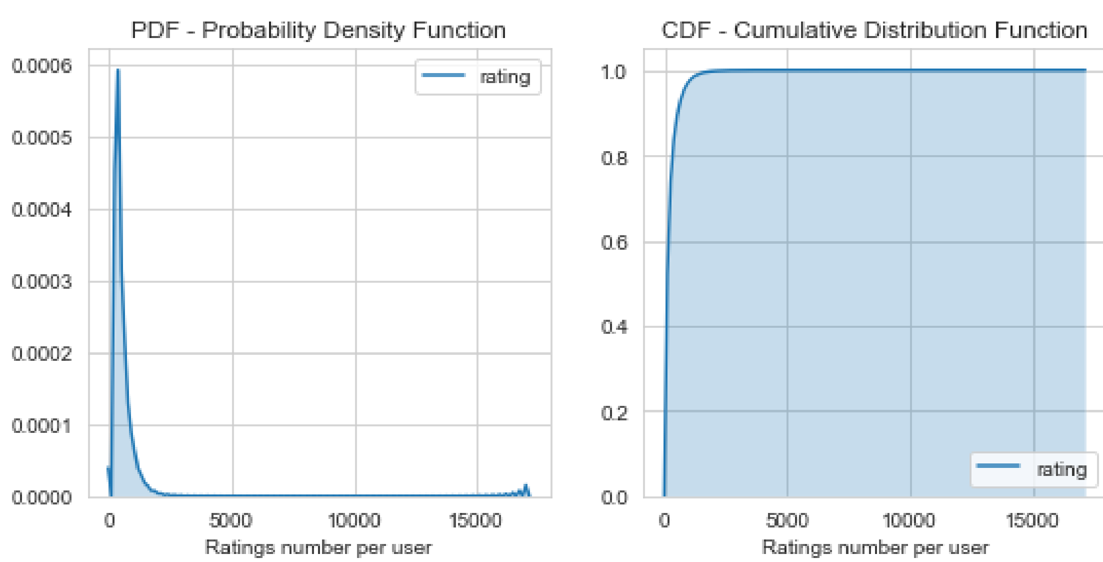
[30]:
```
# Statistical summary
ratings_num_per_user.describe()
```

[30]:
```
count    405041.000000
mean        198.459921
std         290.793238
min           1.000000
25%          34.000000
50%          89.000000
75%         245.000000
max       17112.000000
Name: rating, dtype: float64
```

Let's create a plot of the probability density function function and the cumulative distribution function.

The probability density function (pdf) and cumulative distribution function (cdf) are two of the most important statistical functions in reliability and are closely related. When these functions are known, almost any other reliability measure of interest can be derived or obtained.

[31]:
```
# Plot
fig = plt.figure(figsize = plt.figaspect(.45))
ax1 = plt.subplot(121)
sns.kdeplot(ratings_num_per_user, shade = True, ax = ax1)
plt.xlabel('Ratings number per user')
plt.title("PDF - Probability Density Function")
ax2 = plt.subplot(122)
sns.kdeplot(ratings_num_per_user, shade = True, cumulative = True, ax = ax2)
plt.xlabel('Ratings number per user')
plt.title('CDF - Cumulative Distribution Function')
plt.show()
```



Note that the vast majority of users have less than 1000 reviews.

How many reviews are in the bottom 5% of all reviews?

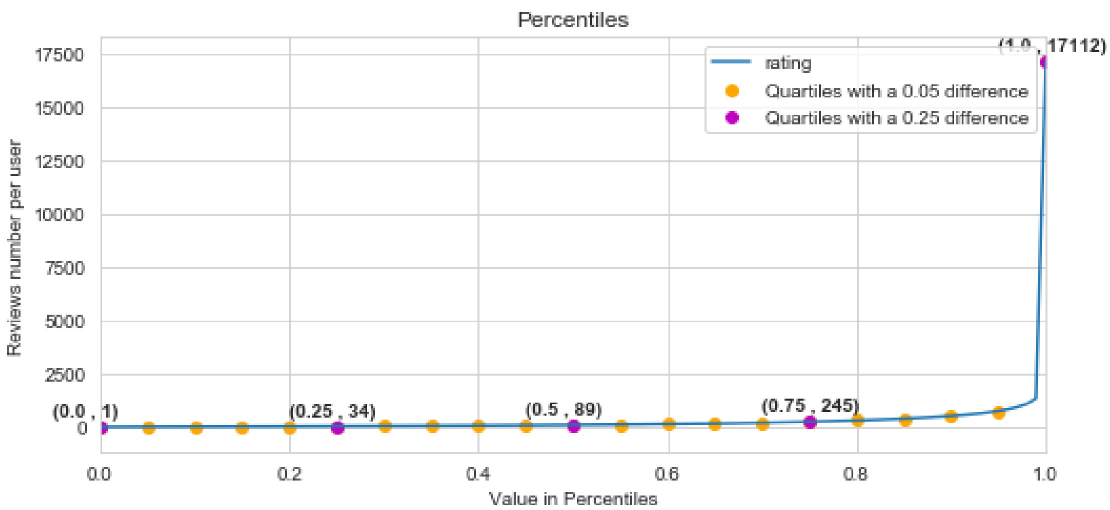
[32]:
```
# Let's extract the percentiles
percentiles = ratings_num_per_user.quantile(np.arange(0,1.01,0.01),␣
↪interpolation = 'higher')
```

[33]:
```
# Viewing by 5
percentiles[::5]
```

[33]:
```
0.00        1
0.05        7
0.10       15
0.15       21
0.20       27
0.25       34
0.30       41
0.35       50
0.40       60
0.45       73
0.50       89
0.55      109
0.60      133
0.65      163
0.70      199
0.75      245
0.80      307
0.85      392
0.90      520
0.95      749
1.00    17112
Name: rating, dtype: int64
```

[34]:
```
# Plot
fig = plt.figure(figsize = plt.figaspect(.45))
plt.title("Percentiles")
percentiles.plot()

# Quartiles with a 0.05 difference
plt.scatter(x = percentiles.index[::5],
            y = percentiles.values[::5],
            c = 'orange',
            label = "Quartiles with a 0.05 difference")

# Quartiles with a 0.25 difference
```

```
plt.scatter(x = percentiles.index[::25],
            y = percentiles.values[::25],
            c = 'm',
            label = "Quartiles with a 0.25 difference")

# Labels and caption
plt.ylabel('Reviews number per user')
plt.xlabel('Value in Percentiles')
plt.legend(loc = 'best')

# Let's mark the 25th, 50th, 75th and 100th percentiles
for x,y in zip(percentiles.index[::25], percentiles[::25]):
    plt.annotate(s = "({} , {})".format(x,y), xy = (x,y), xytext = (x-0.05,␣
 ↪y+500), fontweight = 'bold')

plt.show()
```



- There are some movies (which are very popular) that are rated by a large number of users.
- But most movies (like 90%) have a few hundred ratings.

0.7 Sparse Matrix Creation

Train Sparse Matrix Creation

[35]:
```
# We create the sparse matrix in Numpy format if it doesn't exist
# If it exists, just load it from disk
if os.path.isfile('data/train_sparse_matrix.npz'):
    train_sparse_matrix = sparse.load_npz('data/train_sparse_matrix.npz')
    print("Matrix loaded.")
```

```
else:
    train_sparse_matrix = sparse.csr_matrix((df_netflix_train.rating.values,␣
 ↪(df_netflix_train.user.values,
                                                                            ␣
 ↪ df_netflix_train.movie.values)),)
    print('Matrix created. The shape is: (user, movie): ', train_sparse_matrix.
 ↪shape)
    sparse.save_npz("data/train_sparse_matrix.npz", train_sparse_matrix)
    print('Matrix saved to disk.')
```

```
Matrix loaded.
```

[36]:
```
# Calculate the matrix sparsity
rows, columns = train_sparse_matrix.shape
non_zero_elements = train_sparse_matrix.count_nonzero()
print("Training Matrix Sparsity : {} % ".format(  (1 - (non_zero_elements /␣
 ↪(rows * columns))) * 100) )
```

```
Training Matrix Sparsity : 99.8292709259195 %
```

Test Sparse Matrix Creation

[37]:
```
# We create the sparse matrix in Numpy format if it doesn't exist
# If it exists, just load it from disk
if os.path.isfile('dados/test_sparse_matrix.npz'):
    test_sparse_matrix = sparse.load_npz('data/test_sparse_matrix.npz')
    print("Matrix loaded.")
else:
    test_sparse_matrix = sparse.csr_matrix((df_netflix_test.rating.values,␣
 ↪(df_netflix_test.user.values,
                                                                            ␣
 ↪df_netflix_test.movie.values)))

    print('Matrix created. The shape is: (user, movie): ', test_sparse_matrix.
 ↪shape)
    sparse.save_npz("data/test_sparse_matrix.npz", test_sparse_matrix)
    print('Matrix saved to disk.')
```

```
Matrix created. The shape is: (user, movie):  (2649430, 17771)
Matrix saved to disk.
```

[38]:
```
# Calculate the matrix sparsity
rows, column = test_sparse_matrix.shape
non_zero_elements = test_sparse_matrix.count_nonzero()
print("Training Matrix Sparsity : {} % ".format(  (1 - (non_zero_elements /␣
 ↪(rows * columns))) * 100) )
```

```
Training Matrix Sparsity : 99.95731772988694 %
```

Let's calculate the global average of all movie ratings, average rating per user and

average rating per movie.

[39]:
```
# Global average of all user ratings.
train_avg = dict()
global_train_avg = train_sparse_matrix.sum() / train_sparse_matrix.
↪count_nonzero()
train_avg['global'] = global_train_avg
train_avg
```

[39]:
```
{'global': 3.582890686321557}
```

Let's build a function to calculate the average rating.

[40]:
```
# Averaging function
def calculate_average_ratings(sparse_matrix, of_users):

    # Average user/axis ratings
     # 1 = user axis
     # 0 = film axis
    ax = 1 if of_users else 0

    # Sum
    sum_of_ratings = sparse_matrix.sum(axis=ax).A1

    # Boolean array of ratings (whether a user rated a movie or not)
    is_rated = sparse_matrix!=0

    # Number of ratings for each user or movie
    no_of_ratings = is_rated.sum(axis=ax).A1

    # Maximum user and movie ids in sparse array
    u, m = sparse_matrix.shape

    # We created a dictionary of users and their average ratings.
    avg_ratings = {i:sum_of_ratings[i]/no_of_ratings[i] for i in range(u if␣
↪of_users else m) if no_of_ratings[i]!=0}

    # Returns the rating average dictionary
    return avg_ratings
```

Now we calculate the average of ratings per user.

[41]:
```
# Average User Ratings
train_avg['user'] = calculate_average_ratings(train_sparse_matrix, of_users =␣
↪True)
```

[42]:
```
# View the dictionary
train_avg
```

[42]:
```
{'global': 3.582890686321557,
 'user': {6: 3.5160550458715596,
  7: 3.992957746478873,
  10: 3.3781094527363185,
  25: 3.5,
  33: 3.787878787878788,
  42: 3.9322033898305087,
  59: 3.698717948717949,
  79: 3.5559947299077734,
  83: 4.0,
  87: 3.544642857142857,
  94: 2.8125,
  97: 3.182377049180328,
  131: 4.0,
  134: 4.703081232492997,
  142: 3.45,
  149: 4.25,
  158: 3.625,
  168: 4.208333333333333,
  169: 3.738562091503268,
  178: 3.0,
  183: 3.7096774193548385,
  188: 3.4456066945606696,
  189: 3.0,
  192: 3.522222222222222,
  195: 3.689655172413793,
  199: 3.974747474747475,
  201: 3.605714285714286,
  242: 2.8392857142857144,
  247: 4.019230769230769,
  248: 3.6511627906976742,
  261: 2.769230769230769,
  265: 3.680297397769517,
  266: 4.102222222222222,
  267: 3.325,
  268: 4.008,
  283: 3.4794816414686824,
  291: 3.4745762711864407,
  296: 3.789473684210526,
  298: 3.8052805280528053,
  299: 3.5555555555555554,
  301: 4.05524861878453,
  302: 3.212,
  304: 3.8051948051948052,
  305: 4.096551724137931,
  307: 3.6486486486486487,
  308: 3.4285714285714284,
```

310: 4.1875,
312: 4.071428571428571,
314: 3.6122448979591835,
330: 3.8983050847457625,
331: 2.8181818181818183,
333: 3.7386363636363638,
352: 3.7777777777777777,
363: 3.7714285714285714,
368: 3.6,
369: 3.824561403508772,
379: 4.0,
383: 3.4270833333333335,
384: 3.4275862068965517,
385: 3.0,
392: 2.9333333333333333,
413: 3.4285714285714284,
416: 4.826086956521739,
424: 3.49003984063745,
437: 2.784,
439: 3.8421052631578947,
440: 4.0,
442: 4.127272727272727,
453: 3.375,
462: 3.091116173120729,
470: 3.5714285714285716,
471: 3.3220338983050848,
477: 3.4031620553359683,
478: 3.6333333333333333,
479: 3.0,
481: 4.3090909090909095,
485: 3.5,
490: 5.0,
491: 3.642857142857143,
492: 3.3043478260869565,
495: 4.666666666666667,
508: 4.148936170212766,
515: 3.1176470588235294,
517: 3.5238095238095237,
527: 3.4160919540229884,
529: 4.222222222222222,
536: 4.2825112107623315,
540: 4.6,
544: 3.1839080459770117,
546: 3.247422680412371,
550: 3.792079207920792,
561: 3.88855421686747,
576: 4.240963855421687,

```
585: 3.8135593220338984,
592: 3.585858585858586,
596: 3.99492385786802,
602: 4.092896174863388,
609: 3.7296137339055795,
614: 3.754491017964072,
616: 3.844155844155844,
623: 3.25,
633: 4.0,
657: 3.625,
660: 3.230769230769231,
663: 4.046511627906977,
664: 3.54421768707483,
684: 4.086261980830671,
685: 3.888888888888889,
688: 4.5,
692: 3.2857142857142856,
695: 3.374613003095975,
711: 3.902439024390244,
719: 4.354166666666667,
734: 4.0,
735: 3.950113378684807,
739: 3.696969696969697,
742: 3.937853107344633,
744: 3.1964285714285716,
748: 3.0681818181818183,
750: 2.9431818181818183,
756: 2.32013201320132,
766: 4.056497175141243,
767: 4.028846153846154,
769: 3.2895805142083896,
781: 3.2580645161290325,
784: 4.666666666666667,
785: 3.6560693641618496,
787: 4.1,
788: 3.324110671936759,
793: 3.973684210526316,
798: 3.4035087719298245,
815: 3.4701195219123506,
825: 3.4545454545454546,
829: 3.8142857142857145,
834: 3.2874493927125505,
840: 3.1065573770491803,
857: 3.601503759398496,
870: 4.133333333333334,
873: 3.4545454545454546,
877: 3.5714285714285716,
```

906: 3.712680577849117,
909: 3.4725274725274726,
911: 3.1176470588235294,
915: 4.0,
921: 4.391304347826087,
933: 3.616822429906542,
939: 4.027777777777778,
944: 3.6923076923076925,
952: 4.222222222222222,
955: 3.625,
962: 3.1538461538461537,
967: 3.5161290322580645,
968: 4.086956521739131,
979: 3.111111111111111,
981: 3.3817330210772836,
989: 3.5384615384615383,
997: 2.5142857142857142,
998: 3.7484662576687118,
1007: 4.392857142857143,
1020: 2.5277777777777777,
1024: 3.6153846153846154,
1034: 3.5913978494623655,
1038: 3.6831683168316833,
1044: 3.300751879699248,
1047: 3.76,
1059: 3.3676470588235294,
1067: 3.5084745762711864,
1069: 3.5384615384615383,
1070: 3.7003367003367003,
1079: 3.4,
1082: 3.5330188679245285,
1086: 3.5659574468085107,
1088: 3.595744680851064,
1097: 3.5517241379310347,
1109: 2.8461538461538463,
1135: 3.26027397260274,
1146: 3.4263565891472867,
1179: 3.2275132275132274,
1188: 3.364864864864865,
1206: 3.6610169491525424,
1215: 4.631578947368421,
1221: 3.872413793103448,
1222: 3.5294117647058822,
1236: 3.2903225806451615,
1243: 3.732876712328767,
1248: 3.5223880597014925,
1268: 5.0,

```
1271: 3.840909090909091,
1276: 3.734,
1285: 3.75,
1295: 3.5317460317460316,
1298: 3.0,
1310: 3.5771725032425423,
1315: 3.9272727272727272,
1323: 4.020979020979021,
1326: 4.0606060606060606,
1327: 3.389830508474576,
1328: 3.6320754716981134,
1333: 2.6519033674963395,
1346: 4.72,
1350: 3.3260869565217392,
1353: 3.9391304347826086,
1355: 3.4603174603174605,
1357: 3.2246153846153844,
1358: 3.0851063829787235,
1365: 5.0,
1374: 3.764957264957265,
1375: 3.0,
1378: 4.3125,
1420: 4.059701492537314,
1427: 3.53781512605042,
1442: 4.1661600810536985,
1457: 3.5915678524374175,
1465: 4.196078431372549,
1475: 3.6666666666666665,
1481: 4.153846153846154,
1493: 3.1142857142857143,
1498: 3.5789473684210527,
1500: 3.7146853146853145,
1512: 3.1176470588235294,
1518: 4.242424242424242,
1522: 4.083333333333333,
1523: 3.217391304347826,
1527: 3.615483870967742,
1531: 3.56,
1533: 3.2876106194690267,
1534: 3.824053452115813,
1553: 3.3076923076923075,
1561: 4.105263157894737,
1573: 4.240196078431373,
1576: 3.5851063829787235,
1582: 2.909090909090909,
1588: 3.4545454545454546,
1593: 3.9583333333333335,
```

1611: 3.7387914230019494,
1619: 2.9,
1626: 3.513711151736746,
1636: 3.102766798418972,
1650: 3.2857142857142856,
1658: 2.989847715736041,
1664: 4.115107913669065,
1671: 4.029411764705882,
1689: 3.697674418604651,
1691: 3.408,
1693: 3.936224489795918,
1702: 3.4658823529411764,
1715: 4.245019920318725,
1717: 4.276315789473684,
1722: 3.43,
1723: 3.8173076923076925,
1724: 2.0,
1726: 3.411764705882353,
1730: 5.0,
1734: 3.7547528517110265,
1736: 4.2560975609756095,
1738: 3.4225352112676055,
1746: 3.5555555555555554,
1750: 4.0476190476190474,
1760: 3.6517857142857144,
1768: 3.4545454545454546,
1771: 4.23841059602649,
1779: 4.111111111111111,
1789: 3.238095238095238,
1791: 4.15625,
1798: 2.9615384615384617,
1801: 3.6666666666666665,
1810: 3.5316455696202533,
1811: 3.221978021978022,
1818: 2.9545454545454546,
1819: 4.214285714285714,
1830: 3.8565965583173996,
1831: 3.9692307692307693,
1846: 3.923954372623574,
1856: 4.224489795918367,
1871: 3.548725637181409,
1882: 4.1,
1888: 4.675,
1890: 4.5813953488372094,
1893: 3.3555555555555556,
1894: 3.916417910447761,
1897: 3.8057455540355676,

```
1900: 3.7209302325581395,
1911: 3.5058823529411764,
1912: 4.428571428571429,
1913: 3.3098591549295775,
1918: 2.8728682170542634,
1921: 3.390625,
1922: 3.2085106382978723,
1928: 3.357142857142857,
1933: 2.9705882352941178,
1940: 3.9166666666666665,
1942: 4.125,
1944: 4.0,
1947: 3.564625850340136,
1969: 4.0,
1982: 4.0,
1983: 3.6153846153846154,
1985: 4.125,
1994: 4.666666666666667,
2000: 3.732919254658385,
2003: 3.308641975308642,
2007: 3.628205128205128,
2025: 2.5,
2026: 3.2616822429906542,
2027: 4.434782608695652,
2030: 4.0,
2040: 3.259493670886076,
2050: 4.0,
2057: 3.075268817204301,
2104: 2.6254980079681274,
2110: 2.4,
2114: 3.375,
2122: 4.258064516129032,
2128: 3.3595238095238096,
2133: 3.4239316239316238,
2138: 3.5925925925925926,
2142: 4.25,
2150: 3.9642857142857144,
2157: 3.7142857142857144,
2161: 3.8363636363636364,
2163: 4.108108108108108,
2165: 3.4339622641509435,
2168: 3.533333333333333,
2172: 2.9804878048780488,
2177: 4.8,
2183: 4.068965517241379,
2189: 4.136363636363637,
2190: 3.518796992481203,
```

```
2196: 3.7023809523809526,
2198: 4.014925373134329,
2203: 3.1041666666666665,
2213: 3.8474082702387884,
2214: 3.7522935779816513,
2223: 4.010752688172043,
2225: 3.2633228840125392,
2233: 3.6904024767801857,
2242: 3.5081967213114753,
2250: 3.4917355371900825,
2264: 3.6798245614035086,
2267: 5.0,
2270: 3.9615384615384617,
2273: 3.5185185185185186,
2276: 3.1666666666666665,
2280: 3.7891304347826087,
2282: 3.44,
2295: 3.5,
2297: 2.9323308270676693,
2304: 4.382352941176471,
2305: 3.96,
2307: 3.750326797385621,
2308: 4.555555555555555,
2317: 3.727272727272727,
2333: 3.0675675675675675,
2338: 4.492146596858639,
2347: 4.125,
2363: 4.462776659959759,
2379: 2.7916666666666665,
2385: 4.221902017291066,
2397: 3.6451612903225805,
2404: 4.069767441860465,
2411: 3.68,
2412: 3.138888888888889,
2431: 3.315649867374005,
2442: 3.482573726541555,
2448: 3.1,
2455: 3.492904446546831,
2458: 3.4583333333333335,
2467: 3.814159292035398,
2469: 3.9044652128764277,
2475: 4.261904761904762,
2479: 4.6875,
2492: 4.142857142857143,
2504: 3.5,
2511: 3.1739130434782608,
2514: 2.75,
```

2519: 4.5,
2530: 3.950617283950617,
2534: 3.336283185840708,
2549: 4.609756097560975,
2551: 3.5625,
2555: 3.5294117647058822,
2563: 4.310344827586207,
2583: 3.869565217391304,
2599: 4.0,
2605: 4.048387096774194,
2612: 3.0588235294117645,
2614: 3.25,
2618: 4.0,
2623: 4.852941176470588,
2624: 3.35,
2630: 3.7142857142857144,
2636: 3.873015873015873,
2639: 3.613861386138614,
2662: 3.6267605633802815,
2678: 3.281461434370771,
2680: 4.639344262295082,
2693: 3.6998011928429424,
2697: 3.6451612903225805,
2701: 3.8141025641025643,
2704: 3.183673469387755,
2705: 3.3629032258064515,
2720: 3.895348837209302,
2723: 3.7941176470588234,
2743: 4.3478260869565215,
2746: 4.679245283018868,
2753: 2.7777777777777777,
2754: 3.675,
2757: 2.8185595567867034,
2762: 2.9195402298850577,
2775: 3.7254901960784315,
2781: 2.5873015873015874,
2782: 3.1666666666666665,
2790: 3.4465408805031448,
2794: 3.3819875776397517,
2807: 3.596085409252669,
2820: 4.375,
2831: 3.584,
2844: 3.8260869565217392,
2848: 3.881720430107527,
2862: 4.0,
2869: 2.823529411764706,
2878: 3.9401840490797544,

```
2882: 3.0555555555555554,
2892: 3.477112676056338,
2894: 3.0,
2900: 3.6923076923076925,
2905: 3.4136960600375232,
2909: 4.242424242424242,
2913: 3.75,
2922: 2.4285714285714284,
2945: 4.5,
2947: 3.3820224719101124,
2960: 3.9119318181818183,
2975: 4.0588235294117645,
2976: 3.5388679245283017,
2977: 2.4,
2980: 3.753846153846154,
2994: 5.0,
2996: 4.147186147186147,
3010: 3.6,
3012: 3.937984496124031,
3018: 3.662921348314607,
3031: 3.9545454545454546,
3035: 2.9583333333333335,
3040: 3.7016129032258065,
3046: 4.40625,
3049: 3.6403508771929824,
3053: 3.3333333333333335,
3060: 3.0,
3069: 3.667870036101083,
3094: 3.6808510638297873,
3097: 3.581081081081081,
3098: 4.571428571428571,
3104: 3.3411764705882354,
3132: 3.864864864864865,
3157: 3.272727272727273,
3168: 3.2625,
3172: 3.506849315068493,
3174: 1.0,
3184: 3.0829694323144103,
3186: 3.9647302904564317,
3188: 4.5,
3195: 3.9130434782608696,
3200: 3.7837837837837838,
3204: 4.285714285714286,
3210: 4.0504587155963305,
3221: 3.589041095890411,
3224: 3.4242424242424243,
3232: 3.1666666666666665,
```

3240: 3.764705882352941,
3247: 3.413793103448276,
3260: 3.5801526717557253,
3263: 3.5,
3284: 3.7049180327868854,
3285: 4.071428571428571,
3290: 3.3333333333333335,
3292: 4.011705685618729,
3295: 3.33,
3300: 3.467741935483871,
3303: 4.666666666666667,
3308: 3.9130434782608696,
3321: 2.6910714285714286,
3323: 3.5062111801242235,
3328: 4.503759398496241,
3330: 4.108527131782946,
3335: 3.5454545454545454,
3344: 3.5,
3350: 4.25,
3354: 3.4680851063829787,
3356: 3.7083333333333335,
3358: 3.8518518518518516,
3363: 3.432520325203252,
3373: 3.7217391304347824,
3377: 3.987704918032787,
3378: 3.127659574468085,
3386: 4.508771929824562,
3392: 3.9557522123893807,
3402: 3.8518518518518516,
3404: 3.2470588235294118,
3405: 3.6315789473684212,
3406: 3.2786069651741294,
3419: 3.75,
3423: 2.5594202898550726,
3424: 3.509259259259259,
3432: 4.4088397790055245,
3433: 3.4,
3439: 4.6923076923076925,
3449: 3.526315789473684,
3456: 3.7285714285714286,
3458: 3.20598469032707,
3461: 3.1666666666666665,
3462: 3.611111111111111,
3466: 3.8583333333333334,
3468: 3.6470588235294117,
3469: 3.2083333333333335,
3470: 3.532994923857868,

3487: 3.342857142857143,
3492: 3.3529411764705883,
3493: 3.361111111111111,
3495: 3.9881889763779528,
3498: 4.128712871287129,
3501: 2.8181818181818183,
3502: 3.673611111111111,
3504: 4.2,
3513: 4.0,
3518: 4.363636363636363,
3520: 3.888888888888889,
3521: 3.909090909090909,
3522: 3.6733870967741935,
3530: 3.988826815642458,
3532: 3.5517241379310347,
3549: 3.472,
3561: 2.540983606557377,
3582: 4.4,
3589: 3.875,
3593: 3.581081081081081,
3595: 3.1765151515151517,
3596: 4.689655172413793,
3597: 4.352380952380952,
3599: 3.111111111111111,
3602: 3.5892857142857144,
3604: 3.423432682425488,
3611: 3.0869565217391304,
3612: 3.712230215827338,
3613: 3.6818181818181817,
3621: 3.8312020460358056,
3636: 3.8157894736842106,
3638: 4.589743589743589,
3643: 3.439252336448598,
3660: 3.4705882352941178,
3679: 3.3797468354430378,
3687: 3.395348837209302,
3694: 3.316526610644258,
3695: 3.258687258687259,
3715: 3.1555555555555554,
3718: 3.074858757062147,
3720: 3.5353535353535355,
3723: 3.6065573770491803,
3725: 3.9523809523809526,
3728: 3.187878787878788,
3741: 2.96,
3742: 3.345991561181435,
3743: 3.0761523046092183,

3764: 4.237410071942446,
3767: 3.500805152979066,
3777: 3.65,
3786: 3.5745526838966204,
3788: 3.536082474226804,
3798: 3.306410256410256,
3800: 2.6666666666666665,
3802: 3.515418502202643,
3817: 4.25,
3825: 3.7142857142857144,
3826: 3.7909967845659165,
3830: 3.258426966292135,
3841: 3.8863636363636362,
3870: 3.6196213425129087,
3871: 3.6315789473684212,
3872: 4.467532467532467,
3894: 3.8076923076923075,
3902: 3.542372881355932,
3906: 3.6989247311827955,
3916: 4.1692307692307695,
3920: 4.466666666666667,
3921: 3.686868686868687,
3931: 3.5238095238095237,
3934: 4.558739255014327,
3935: 3.7903225806451615,
3937: 2.740740740740741,
3941: 3.409090909090909,
3949: 3.7049180327868854,
3952: 2.9302325581395348,
3983: 3.04,
3984: 3.662303664921466,
3992: 3.8157894736842106,
3998: 4.699427480916031,
4011: 3.5,
4015: 3.3333333333333335,
4016: 3.9420289855072466,
4034: 4.6,
4044: 3.613821138211382,
4050: 3.477777777777778,
4057: 3.925243770314193,
4059: 3.517123287671233,
4062: 3.183673469387755,
4066: 3.9692307692307693,
4084: 3.728395061728395,
4087: 3.28125,
4089: 3.52,
4101: 3.357142857142857,

4103: 3.8260869565217392,
4106: 2.9473684210526314,
4125: 3.819277108433735,
4127: 3.6863905325443787,
4142: 3.3,
4166: 3.618181818181818,
4167: 3.738095238095238,
4171: 4.178217821782178,
4177: 3.775,
4184: 3.8771929824561404,
4193: 3.635514018691589,
4196: 3.5238095238095237,
4229: 3.4630541871921183,
4230: 4.177570093457944,
4244: 2.8372093023255816,
4247: 3.4922206506364923,
4265: 3.6551724137931036,
4269: 3.85,
4277: 3.784090909090909,
4292: 3.253012048192771,
4294: 3.8727272727272726,
4300: 3.8205128205128207,
4306: 3.0605714285714285,
4314: 3.419642857142857,
4315: 4.187817258883249,
4318: 3.0827586206896553,
4326: 3.8106382978723405,
4330: 3.890625,
4334: 3.1,
4343: 4.0,
4347: 3.8520499108734403,
4350: 3.759259259259259,
4356: 3.7,
4365: 3.6621621621621623,
4368: 3.7017543859649122,
4374: 4.2444444444444445,
4389: 3.926829268292683,
4395: 4.176470588235294,
4398: 3.8843537414965987,
4407: 2.473684210526316,
4409: 4.514915693904021,
4414: 4.280487804878049,
4419: 4.133333333333334,
4421: 3.3930024410089503,
4422: 3.923758865248227,
4424: 3.946236559139785,
4433: 3.7777777777777777,

4436: 4.3076923076923075,
4437: 3.4611111111111112,
4439: 3.3341346153846154,
4451: 3.5853658536585367,
4456: 3.6527777777777777,
4460: 3.22962962962963,
4461: 3.7265917602996255,
4465: 3.327731092436975,
4473: 3.5,
4477: 3.7240875912408757,
4488: 2.8393162393162394,
4490: 2.9390243902439024,
4491: 4.317073170731708,
4504: 4.123076923076923,
4505: 3.5163934426229506,
4518: 3.9767441860465116,
4522: 2.111111111111111,
4525: 4.1875,
4537: 4.217741935483871,
4540: 3.6875,
4545: 4.18595041322314,
4556: 3.439252336448598,
4559: 4.176470588235294,
4574: 3.1176470588235294,
4576: 3.4461152882205512,
4580: 3.6904761904761907,
4581: 4.065420560747664,
4588: 4.531645569620253,
4597: 3.0960874568469503,
4605: 4.5,
4606: 3.3848039215686274,
4607: 2.6296296296296298,
4623: 3.8294573643410854,
4624: 3.5185185185185186,
4628: 2.6881188118811883,
4630: 3.5733333333333333,
4638: 3.7657142857142856,
4639: 2.9047619047619047,
4658: 2.9375,
4666: 3.930622009569378,
4679: 4.760744985673353,
4702: 4.0,
4705: 5.0,
4706: 3.760824742268041,
4728: 4.0,
4730: 3.3863636363636362,
4733: 3.68,

```
4737: 3.2222222222222223,
4738: 3.9069767441860463,
4739: 3.7541371158392436,
4751: 3.857142857142857,
4753: 4.323917137476459,
4764: 3.769230769230769,
4773: 3.5714285714285716,
4783: 3.7022613065326633,
4796: 3.0,
4797: 3.413793103448276,
4804: 2.6391752577319587,
4807: 4.5162790697674415,
4815: 3.4285714285714284,
4832: 3.1161616161616164,
4837: 3.8260869565217392,
4850: 3.6538461538461537,
4853: 3.552380952380952,
4862: 4.0,
4873: 3.860294117647059,
4875: 3.4782608695652173,
4878: 3.588235294117647,
4882: 3.1946902654867255,
4893: 3.7663934426229506,
4895: 3.985074626865672,
4905: 3.2235915492957745,
4906: 3.856294536817102,
4919: 3.3703703703703702,
4932: 2.411764705882353,
4953: 4.285714285714286,
4954: 3.48,
4960: 3.3496240601503757,
4967: 3.8781725888324874,
4973: 3.212121212121212,
4977: 3.5866666666666664,
4983: 3.7804878048780486,
4991: 3.6097560975609757,
4996: 3.3076923076923075,
5011: 4.914285714285715,
5017: 3.6344827586206896,
5022: 3.933333333333333,
5026: 3.0,
5027: 3.5771670190274842,
5029: 1.0,
5031: 3.391304347826087,
5034: 3.304,
5082: 4.3070539419087135,
5087: 3.485981308411215,
```

5088: 4.217391304347826,
5092: 3.3391304347826085,
5094: 3.9473684210526314,
5099: 4.111111111111111,
5105: 3.8923076923076922,
5107: 3.62803738317757,
5109: 3.2666666666666666,
5112: 3.217391304347826,
5117: 3.9692307692307693,
5126: 3.519083969465649,
5127: 2.0,
5145: 3.4642857142857144,
5159: 3.8666666666666667,
5163: 3.1724137931034484,
5167: 3.7777777777777777,
5168: 3.378504672897196,
5169: 3.4814814814814814,
5170: 4.7,
5172: 3.8461538461538463,
5176: 4.265486725663717,
5178: 3.076923076923077,
5202: 3.380281690140845,
5203: 3.310344827586207,
5210: 3.3846153846153846,
5225: 3.9615384615384617,
5228: 4.194805194805195,
5245: 3.6,
5251: 3.5,
5255: 3.736,
5256: 3.207920792079208,
5269: 2.857142857142857,
5275: 3.0,
5277: 3.218543046357616,
5280: 3.5106382978723403,
5281: 4.142857142857143,
5298: 3.875,
5308: 3.537313432835821,
5323: 2.8333333333333335,
5336: 3.1846153846153844,
5339: 3.3863636363636362,
5349: 3.4324324324324325,
5353: 5.0,
5356: 2.6323529411764706,
5362: 4.0344827586206895,
5369: 3.55,
5380: 3.2222222222222223,
5390: 3.5,

5399: 3.8098663926002057,
5404: 3.6011904761904763,
5412: 3.3425925925925926,
5426: 2.0,
5428: 3.769230769230769,
5430: 3.4202020202020202,
5442: 3.25,
5446: 3.276995305164319,
5454: 3.3402061855670104,
5469: 3.5942028985507246,
5470: 4.0,
5474: 4.096774193548387,
5480: 2.909090909090909,
5496: 3.616279069767442,
5498: 3.5714285714285716,
5507: 2.9444444444444446,
5516: 3.15625,
5519: 3.277227722772277,
5525: 3.8846153846153846,
5530: 3.9352773826458036,
5534: 3.426470588235294,
5549: 4.083333333333333,
5552: 3.0727272727272728,
5568: 2.946132596685083,
5569: 3.9445178335535007,
5570: 3.0357142857142856,
5573: 4.115384615384615,
5575: 3.672,
5583: 3.7783018867924527,
5600: 3.644628099173554,
5602: 4.384615384615385,
5605: 4.1875,
5609: 3.6793893129770994,
5612: 3.5511811023622046,
5618: 3.7502392344497606,
5631: 3.0,
5646: 3.1666666666666665,
5650: 3.509433962264151,
5652: 3.571041948579161,
5655: 3.711764705882353,
5658: 3.046153846153846,
5663: 4.15,
5665: 3.505649717514124,
5668: 4.662921348314606,
5671: 3.5789473684210527,
5682: 3.4976076555023923,
5685: 3.2222222222222223,

5686: 4.426778242677824,
5689: 2.888888888888889,
5696: 4.232843137254902,
5710: 4.03125,
5720: 3.4444444444444446,
5722: 3.5636363636363635,
5727: 4.347474747474747,
5734: 2.4358974358974357,
5739: 4.008620689655173,
5742: 3.5,
5749: 3.986111111111111,
5756: 4.096774193548387,
5758: 3.336206896551724,
5762: 3.8524173027989823,
5768: 3.6097560975609757,
5771: 3.799363057324841,
5773: 3.6944444444444446,
5775: 3.507575757575758,
5823: 4.041095890410959,
5827: 4.266666666666667,
5834: 1.0,
5836: 3.3676470588235294,
5841: 3.466403162055336,
5842: 4.277777777777778,
5851: 4.621621621621622,
5856: 2.7044334975369457,
5858: 3.3016528925619837,
5862: 3.6511627906976742,
5871: 3.6608695652173915,
5872: 3.5555555555555554,
5875: 2.9,
5878: 3.1354838709677417,
5880: 4.176470588235294,
5889: 3.628032345013477,
5899: 3.5,
5900: 3.4716981132075473,
5916: 3.978723404255319,
5917: 3.097560975609756,
5918: 3.8727272727272726,
5922: 4.0,
5926: 3.302325581395349,
5927: 4.122641509433962,
5931: 4.130434782608695,
5945: 2.34392523364486,
5948: 3.9646464646464645,
5949: 2.7777777777777777,
5959: 3.25,

5966: 2.5454545454545454,
5977: 3.8684210526315788,
5980: 3.8882521489971347,
5985: 4.111111111111111,
5989: 3.2467532467532467,
5994: 3.5620915032679736,
6013: 4.769736842105263,
6018: 3.6,
6024: 4.285714285714286,
6025: 3.327272727272727,
6037: 3.4556962025316458,
6040: 2.606060606060606,
6042: 4.214285714285714,
6048: 2.8911290322580645,
6056: 3.210221793635487,
6057: 3.597938144329897,
6062: 3.9223880597014924,
6065: 3.4322766570605188,
6071: 4.227513227513228,
6080: 4.393103448275862,
6082: 3.1960132890365447,
6092: 4.209090909090909,
6096: 3.5211267605633805,
6110: 3.130434782608696,
6130: 3.4770992366412212,
6134: 3.7333333333333334,
6141: 3.4,
6154: 3.25,
6161: 3.53072625698324,
6172: 3.4385382059800667,
6177: 4.174603174603175,
6181: 3.6869565217391305,
6184: 4.214285714285714,
6186: 3.6666666666666665,
6190: 3.6923076923076925,
6195: 4.45925925925926,
6206: 3.738430583501006,
6228: 3.141843971631206,
6231: 3.6761363636363638,
6262: 3.9069767441860463,
6265: 4.161137440758294,
6268: 3.929936305732484,
6273: 3.65625,
6303: 4.211267605633803,
6304: 3.0,
6310: 3.5374149659863945,
6311: 4.32,

6312: 3.5,
6319: 3.829525483304042,
6323: 4.5,
6325: 3.585858585858586,
6334: 4.515463917525773,
6336: 3.419047619047619,
6337: 3.0357142857142856,
6339: 3.532258064516129,
6340: 3.7857142857142856,
6360: 3.7755102040816326,
6363: 2.736842105263158,
6369: 3.4388489208633093,
6371: 4.21875,
6373: 4.315436241610739,
6374: 2.9384615384615387,
6378: 3.390625,
6381: 2.7211538461538463,
6384: 3.5172413793103448,
6395: 4.0,
6401: 4.195583596214511,
6408: 3.563063063063063,
6409: 3.5,
6416: 3.6839506172839505,
6418: 4.2439024390243905,
6427: 4.204545454545454,
6435: 4.0,
6438: 4.0,
6440: 3.6363636363636362,
6457: 3.5348837209302326,
6460: 3.820212765957447,
6463: 4.235294117647059,
6464: 5.0,
6483: 3.9632183908045975,
6485: 3.9644128113879002,
6487: 4.204545454545454,
6488: 3.87603305785124,
6496: 3.3442622950819674,
6499: 3.8,
6500: 3.254612546125461,
6504: 3.9868073878627968,
6510: 3.488,
6514: 4.201680672268908,
6517: 4.611111111111111,
6518: 4.105263157894737,
6522: 3.121212121212121,
6527: 3.4,
6529: 3.856230031948882,

```
6534: 3.7866666666666666,
6544: 4.011556240369799,
6567: 3.9125,
6586: 2.880952380952381,
6603: 4.123893805309734,
6608: 3.6327272727272727,
6628: 3.3821656050955413,
6629: 3.158559696778269,
6635: 4.92,
6655: 3.7903225806451615,
6660: 2.5,
6663: 4.120930232558139,
6666: 3.7265917602996255,
6669: 3.109181141439206,
…}}
```

[43]:
```
# Print
print('Average User Rating 149 :', train_avg['user'][149])
```

```
Average User Rating 149 : 4.25
```

Now we calculate the average of ratings per film.

[44]:
```
# Average Movie Ratings
train_avg['movie'] =  calculate_average_ratings(train_sparse_matrix, of_users =␣
↪False)
```

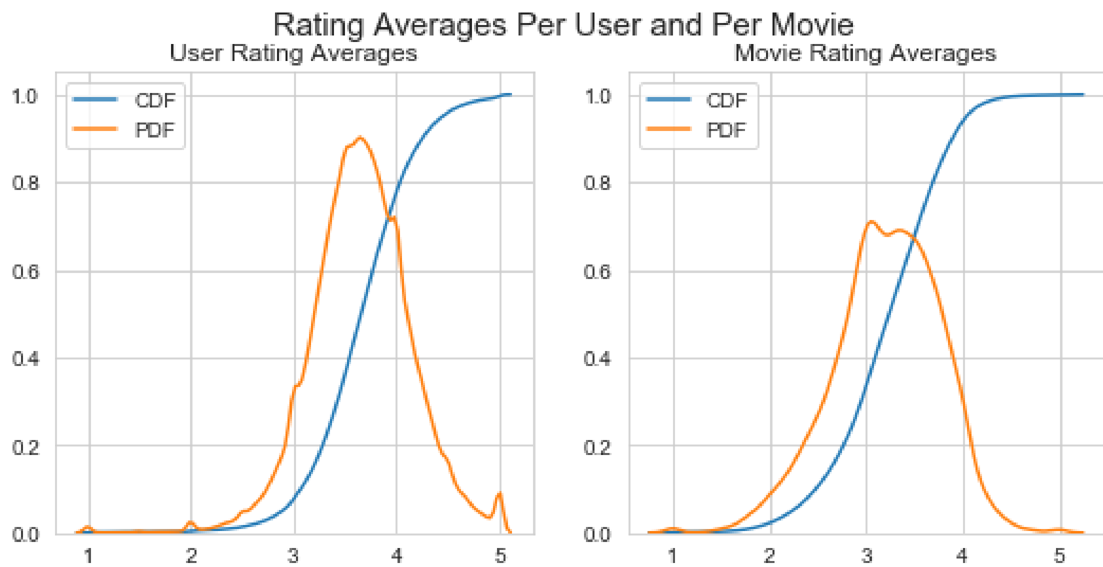
[45]:
```
# Print
print('Average Movie Rating 32 :', train_avg['movie'][32])
```

```
Average Movie Rating 32 : 3.9922680412371134
```

Average PDFs and CDFs. User ratings and movies (training data).

[46]:
```
# Plot
fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, figsize = plt.figaspect(.
↪45))
fig.suptitle('Rating Averages Per User and Per Movie', fontsize = 15)

ax1.set_title('User Rating Averages')

# We get the list of average user ratings from the average dictionary.
users_avg = [rat for rat in train_avg['user'].values()]
sns.distplot(users_avg, ax = ax1, hist = False, kde_kws = dict(cumulative =␣
↪True), label = 'CDF')
sns.distplot(users_avg, ax = ax1, hist = False, label = 'PDF')

ax2.set_title('Movie Rating Averages')
```

```
# We get the list of average movie ratings from the dictionary.
movies_avg = [rat for rat in train_avg['movie'].values()]
sns.distplot(movies_avg, ax = ax2, hist = False, kde_kws = dict(cumulative =␣
↪True), label = 'CDF')
sns.distplot(movies_avg, ax = ax2, hist = False, label = 'PDF')

plt.show()
```



0.8 Cold Start Problem

[47]:
```
# Users Cold start
train_users = len(train_avg['user'])
new_users = total_users - train_users
```

[48]:
```
# Print
print('Grand Total of Users:', total_users)
print('Total Users in Training :', train_users)
print("Total Users Not in Training: {} ({}%)".format(new_users,
                                                                    np.
↪round((new_users / total_users) * 100, 2)))
```

```
Grand Total of Users: 480189
Total Users in Training : 405041
Total Users Not in Training: 75148 (15.65%)
```

75148 users are not part of the training data, that is, we have no way of learning the evaluation pattern of these users! This is the cold start problem.

[49]:
```python
# Movies Cold start
train_movies = len(train_avg['movie'])
new_movies = total_movies - train_movies
```

[50]:
```python
# Print
print('Grand Total of Movies:', total_movies)
print('Total Films in Training:', train_movies)
print("Total Films Not in Training: {} ({}%)".format(new_movies,
                                                                   np.
 ↪round((new_movies/total_movies)*100, 2)))
```

```
Grand Total of Movies: 17770
Total Films in Training: 17424
Total Films Not in Training: 346 (1.95%)
```

346 movies do not appear in the training data. We will have to deal with this when we work especially on the Machine Learning model.

0.9 Calculating the User Similarity Matrix

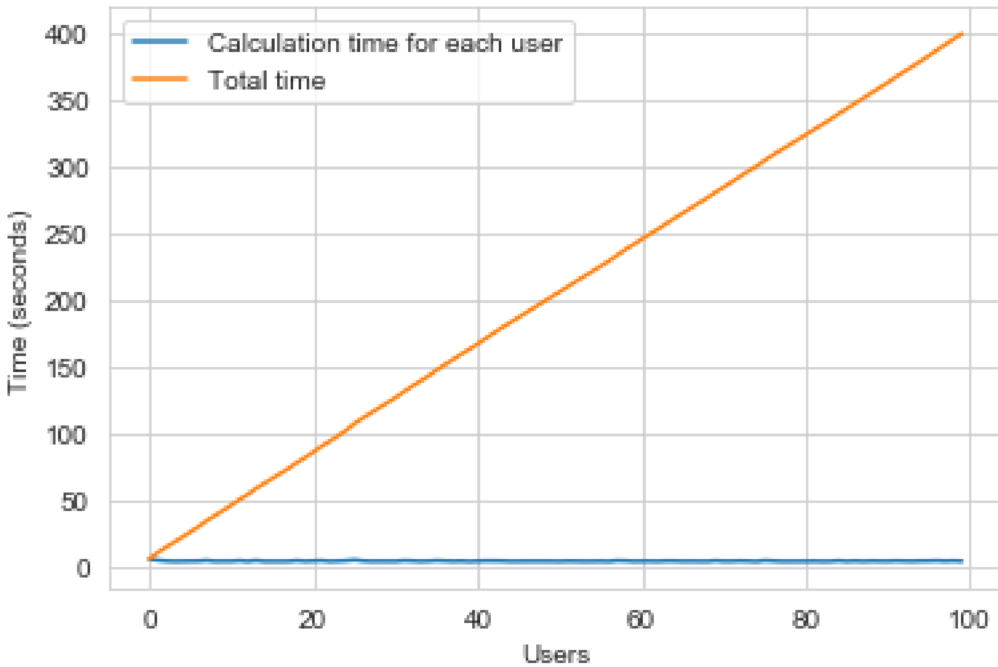
[51]:
```python
# Similarity calculation function
def calculate_user_similarity(sparse_matrix,
                                 compute_for_few = False,
                                 top = 100,
                                 verbose = False,
                                 verb_for_n_rows = 20,
                                 draw_time_taken = True):

    # Control variables
    no_of_users, _ = sparse_matrix.shape
    row_ind, col_ind = sparse_matrix.nonzero()
    row_ind = sorted(set(row_ind))
    time_taken = list()
    rows, cols, data = list(), list(), list()
    if verbose: print("Calculating top", top, "similarities for each user...")
    start = datetime.now()
    temp = 0

    # Matrix Loop
    for row in row_ind[:top] if compute_for_few else row_ind:
        temp = temp + 1
        prev = datetime.now()

        # Calculating the cosine similarity
        sim = cosine_similarity(sparse_matrix.getrow(row), sparse_matrix).
 ↪ravel()
        top_sim_ind = sim.argsort()[-top:]
        top_sim_val = sim[top_sim_ind]
```

```
        rows.extend([row]*top)
        cols.extend(top_sim_ind)
        data.extend(top_sim_val)
        time_taken.append(datetime.now().timestamp() - prev.timestamp())

        if verbose:
            if temp%verb_for_n_rows == 0:
                print("Completed calculation for {} users [  total time : {} ␣
↪]".format(temp, datetime.now()-start))

    if verbose: print('Creation of sparse matrix from computed similarities...')

    if draw_time_taken:
        plt.plot(time_taken, label = 'Calculation time for each user')
        plt.plot(np.cumsum(time_taken), label = 'Total time')
        plt.legend(loc = 'best')
        plt.xlabel('Users')
        plt.ylabel('Time (seconds)')
        plt.show()

    return sparse.csr_matrix((data, (rows, cols)), shape = (no_of_users,␣
↪no_of_users)), time_taken
```

[52]:

```
# Calculate the similarity

# Mark the start
start = datetime.now()

# Calculates the similarity
sparse_matrix_user, _ = calculate_user_similarity(train_sparse_matrix,
                                                  compute_for_few = True,
                                                  top = 100,
                                                  verbose = True)

print("Total Processing Time:", datetime.now() - start)
```

```
Calculating top 100 similarities for each user…
Completed calculation for 20 users [  total time : 0:01:21.823198  ]
Completed calculation for 40 users [  total time : 0:02:42.962442  ]
Completed calculation for 60 users [  total time : 0:04:01.593251  ]
Completed calculation for 80 users [  total time : 0:05:20.324747  ]
Completed calculation for 100 users [  total time : 0:06:39.804223  ]
Creation of sparse matrix from computed similarities…
```




```
Total Processing Time: 0:06:51.595724
```

We have **405,041 users** in our training set and computing similarities between them (**17K dimensional array**) is time consuming.

We will try to reduce the dimensions using SVD, in order to speed up the process.

0.10 Dimensionality Reduction with TruncatedSVD

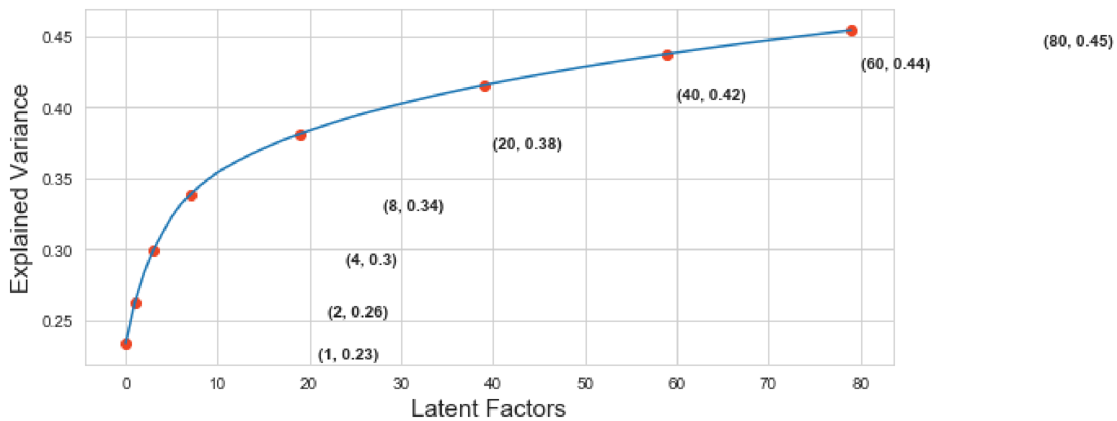
[53]:
```
# Dimensionality reduction

# Mark the start
start = datetime.now()

# Creates TruncatedSVD object by reducing dimensionality to 80 dimensions
netflix_svd = TruncatedSVD(n_components = 80, algorithm = 'randomized',␣
↪random_state = 15)

# Apply TruncatedSVD
trunc_svd = netflix_svd.fit_transform(train_sparse_matrix)

print("Total Processing Time:", datetime.now() - start)
```

```
Total Processing Time: 0:02:29.773465
```

Let's calculate the variance explained by the components.

```
[54]: # Calculates the explained variance
expl_var = np.cumsum(netflix_svd.explained_variance_ratio_)
```

```
[55]: # Plot
fig, (ax1) = plt.subplots(nrows = 1, ncols = 1, figsize = plt.figaspect(.45))

ax1.set_ylabel("Explained Variance", fontsize = 15)
ax1.set_xlabel("Latent Factors", fontsize = 15)
ax1.plot(expl_var)

# Let's mark some combinations of (latent factors, explained variance) to make␣
↪the graph clearer
ind = [1, 2, 4, 8, 20, 40, 60, 80]
ax1.scatter(x = [i-1 for i in ind], y = expl_var[[i-1 for i in ind]], c =␣
↪'#ee4422')

for i in ind:
    ax1.annotate(s ="({}, {})".format(i,  np.round(expl_var[i-1], 2)), xy =␣
↪(i-1, expl_var[i-1]),
                 xytext = ( i+20, expl_var[i-1] - 0.01), fontweight = 'bold')

plt.show()
```



With 80 components we explain approximately 45% of the data variance. This is enough for our example.

```
[56]: # Let's project our array into 80-dimensional space
start = datetime.now()
trunc_matrix = train_sparse_matrix.dot(netflix_svd.components_.T)
print("Total Processing Time:", datetime.now() - start)
```

```
Total Processing Time: 0:00:05.359700
```

[57]:
```
# Shape
trunc_matrix.shape
```

[57]: (2649430, 80)

[58]:
```
# Type
type(trunc_matrix)
```

[58]: numpy.ndarray

[59]:
```
# Let's create and save to disk the matrix with the dimensionality reduced to␣
↪80 dimensions
if not os.path.isfile('data/truncated_sparse_matrix_user.npz'):
    truncated_sparse_matrix_user = sparse.csr_matrix(trunc_matrix)
    sparse.save_npz('data/truncated_sparse_matrix_user',␣
↪truncated_sparse_matrix_user)
else:
    truncated_sparse_matrix_user = sparse.load_npz('data/
↪truncated_sparse_matrix_user.npz')
```

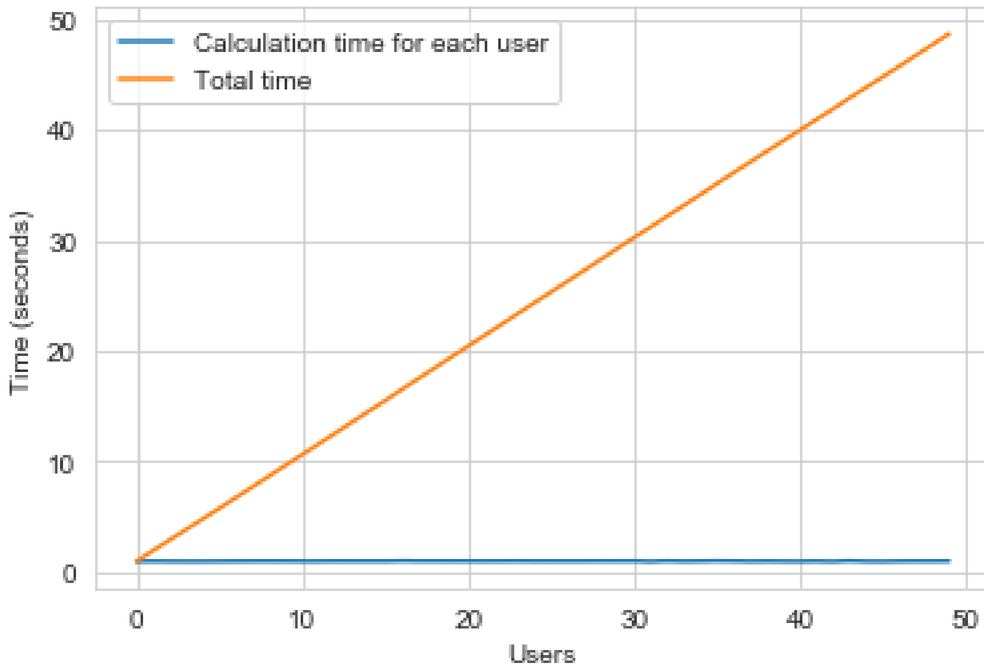
[60]:
```
# Shape
truncated_sparse_matrix_user.shape
```

[60]: (2649430, 80)

Now we recalculate the similarity of users using the truncated matrix.

[61]:
```
# Calculate similarity of users

# Mark the start
start = datetime.now()

# Calculates the similarity
trunc_sim_matrix, _ = calculate_user_similarity(truncated_sparse_matrix_user,
                                                 compute_for_few = True,
                                                 top = 50,
                                                 verbose = True)

print("Total Processing Time:", datetime.now() - start)
```

```
Calculating top 50 similarities for each user…
Completed calculation for 20 users [  total time : 0:00:19.442012  ]
Completed calculation for 40 users [  total time : 0:00:38.986719  ]
Creation of sparse matrix from computed similarities…
```




```
Total Processing Time: 0:00:52.963406
```

0.11 Calculating Movie Similarity Matrix

[62]:
```
# Calculation of movie similarity

# Mark the start
start = datetime.now()

# Create if it doesn't exist
if not os.path.isfile('data/sparse_matrix_movie.npz'):
    sparse_matrix_movie = cosine_similarity(X = train_sparse_matrix.T,␣
 ↪dense_output = False)
    print("Matrix Created.")
    sparse.save_npz("data/sparse_matrix_movie.npz", sparse_matrix_movie)
    print("Matrix Saves in disk")
else:
    sparse_matrix_movie = sparse.load_npz("data/sparse_matrix_movie.npz")
    print("Matrix Loaded.")

print("Total Processing Time:", datetime.now() - start)
```

```
Matrix Created.
Matrix Saves in disk
Total Processing Time: 0:08:01.174500
```

[63]:
```
# Shape
sparse_matrix_movie.shape
```

[63]: (17771, 17771)

[64]:
```
# Extract movie ids
movie_ids = np.unique(sparse_matrix_movie.nonzero()[1])
```

[65]:
```
# Calculate the similarity of movies according to the rating pattern of users

# Mark the start
start = datetime.now()

# Dictionary to store similarities
similar_movies = dict()

# Loop through movie ids
for movie in movie_ids:
    # Get the top similar movies and store them in the dictionary
    sim_movies = sparse_matrix_movie[movie].toarray().ravel().argsort()[::-1][1:
 ↪]
    similar_movies[movie] = sim_movies[:100]

print("Total Processing Time:", datetime.now() - start)
```

```
Total Processing Time: 0:00:29.393400
```

[66]:
```
# Movies similar to id 43's movie
similar_movies[43]
```

[66]:
```
array([ 6938,  3353, 15088, 14888,  7550,  3257,  8115,  1054, 11436,
        8309, 11181, 15275,  2041,  7211, 12321,  2619,  9667,   568,
       14988,  7044, 10062,  9895, 15061,  5916,  1113, 11557, 16935,
        7498,  6926,   512, 14415, 13525,  2466,  9468,  8974, 15157,
        1808, 12396,  1944,  3645,  4222,  9893,  3362, 10777,  7543,
        9883,  4062,  7185,  7107,  9143, 17086, 13000, 16184,  5723,
        8452,  3068,  2943, 16515, 13429, 13885,  9664, 12229,   101,
       17602, 17564, 14189, 15292, 13802,  1737, 12650, 17444, 12712,
       15639, 14024,   603,  6081, 10534, 17717, 14824,  9804, 15438,
       15191,  9794,  7137,  7408, 10584,  6629,  1639, 14614,  1927,
        2202, 17755,  5122, 16804,   887,  1768, 16101, 14037,  5666,
         991], dtype=int64)
```

Now let's find the most similar movies using the similarity matrix.

[68]:
```
# Let's load the movie titles from the csv file provided by Netflix
movie_titles = pd.read_csv("data/movie_titles.csv",
                           sep = ',',
```

```
                              header = None,
                              names = ['ID_Movie', 'Year_Launch', 'Title'],
                              verbose = True,
                              index_col = 'ID_Movie',
                              encoding = "ISO-8859-1")
```

```
Tokenization took: 4.00 ms
Type conversion took: 9.99 ms
Parser memory cleanup took: 0.00 ms
```

[69]:
```
# Visualize the data
movie_titles.head()
```

[69]:
```
          Year_Launch                         Title
ID_Movie
1              2003.0                Dinosaur Planet
2              2004.0     Isle of Man TT 2004 Review
3              1997.0                      Character
4              1994.0  Paula Abdul's Get Up & Dance
5              2004.0       The Rise and Fall of ECW
```

Let's see which films are similar to the ID 43 film.

[70]:
```
# Movie ID
id_movie = 43
```

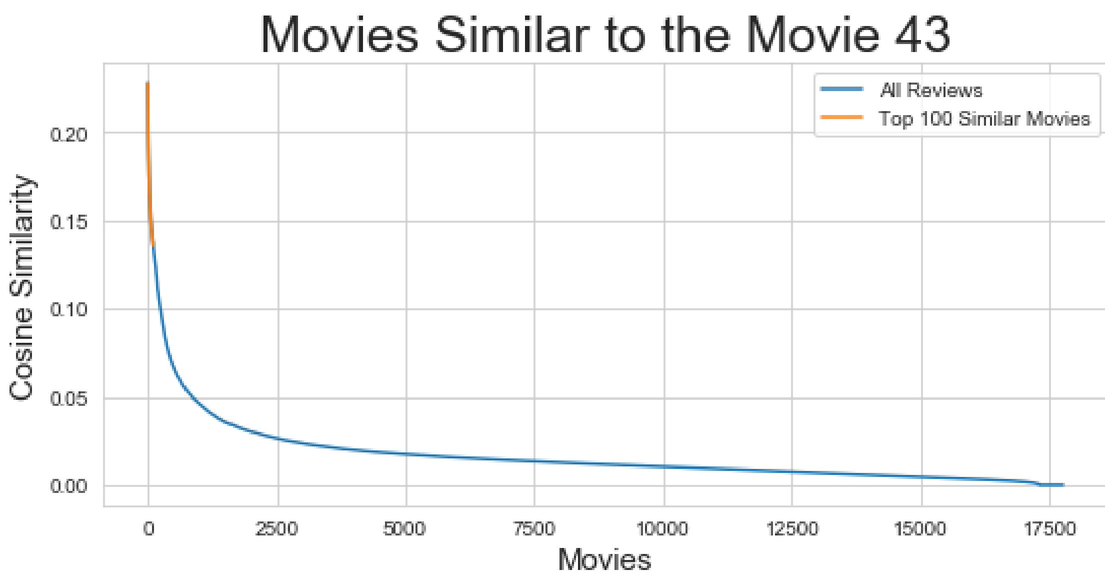
[71]:
```
# Print
print("Movie:", movie_titles.loc[id_movie].values[1])
print("Total User Ratings = {}.".format(train_sparse_matrix[:,id_movie].
 ↪getnnz()))
print("We found {} movies that are similar to this one and let's print the most␣
 ↪similar ones.".format(sparse_matrix_movie[:,id_movie].getnnz()))
```

```
Movie: Silent Service
Total User Ratings = 102.
We found 17300 movies that are similar to this one and let's print the most
similar ones.
```

[72]:
```
# Finding all similarities
similarities = sparse_matrix_movie[id_movie].toarray().ravel()
similar_indexes = similarities.argsort()[::-1][1:]
similarities[similar_indexes]
sim_indexes = similarities.argsort()[::-1][1:]
```

[73]:
```
# Plot
fig = plt.figure(figsize = plt.figaspect(.45))
plt.plot(similarities[sim_indexes], label = 'All Reviews')
plt.plot(similarities[sim_indexes[:100]], label = 'Top 100 Similar Movies')
plt.title("Movies Similar to the Movie {}".format(id_movie), fontsize = 25)
```

```python
plt.xlabel("Movies", fontsize = 15)
plt.ylabel("Cosine Similarity", fontsize = 15)
plt.legend()
plt.show()
```



[74]:
```python
# Here are the top 10 movies most similar to movie 43
movie_titles.loc[sim_indexes[:10]]
```

[74]:

ID_Movie	Year_Launch	Title
6938	1995.0	Battle Skipper
3353	1999.0	Midnight Panther
15088	1996.0	Yamamoto Yohko: Starship Girl
14888	1996.0	Ayane's High Kick
7550	1996.0	Big Wars
3257	1990.0	Takegami: Guardian of Darkness: War God
8115	2000.0	Virgin Fleet
1054	1986.0	Odin: Photon Space Sailer Starlight
11436	1995.0	Super Atragon
8309	2000.0	Go Shogun: The Time Etranger

1 End