In [1]:
```
%matplotlib inline

!pip install pandas
!pip install matplotlib
!pip install seaborn

import pandas as pd
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.6/dist-packages (0.24.2)
Requirement already satisfied: pytz>=2011k in /usr/local/lib/python3.6/dist-packages (from pandas) (2018.9)
Requirement already satisfied: python-dateutil>=2.5.0 in /usr/local/lib/python3.6/dist-packages (from pandas) (2.5.3)
Requirement already satisfied: numpy>=1.12.0 in /usr/local/lib/python3.6/dist-packages (from pandas) (1.16.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/dist-packages (from python-dateutil>=2.5.0->pandas) (1.12.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-packages (3.0.3)
Requirement already satisfied: numpy>=1.10.0 in /usr/local/lib/python3.6/dist-packages (from matplotlib) (1.16.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib) (2.5.3)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib) (2.4.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib) (1.1.0)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from cycler>=0.10->matplotlib) (1.12.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (from kiwisolver>=1.0.1->matplotlib) (41.0.1)
Requirement already satisfied: seaborn in /usr/local/lib/python3.6/dist-packages (0.9.0)
Requirement already satisfied: pandas>=0.15.2 in /usr/local/lib/python3.6/dist-packages (from seaborn) (0.24.2)
Requirement already satisfied: scipy>=0.14.0 in /usr/local/lib/python3.6/dist-packages (from seaborn) (1.3.0)
Requirement already satisfied: numpy>=1.9.3 in /usr/local/lib/python3.6/dist-packages (from seaborn) (1.16.3)
Requirement already satisfied: matplotlib>=1.4.3 in /usr/local/lib/python3.6/dist-packages (from seaborn) (3.0.3)
Requirement already satisfied: python-dateutil>=2.5.0 in /usr/local/lib/python3.6/dist-packages (from pandas>=0.15.2->seaborn) (2.5.3)
Requirement already satisfied: pytz>=2011k in /usr/local/lib/python3.6/dist-packages (from pandas>=0.15.2->seaborn) (2018.9)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=1.4.3->seaborn) (1.1.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=1.4.3->seaborn) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=1.4.3->seaborn) (2.4.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/dist-packages (from python-dateutil>=2.5.0->pandas>=0.15.2->seaborn) (1.12.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (from kiwisolver>=1.0.1->matplotlib>=1.4.3->seaborn) (41.0.1)
```

Recomendation System for groups

Dataset

Loading the dataset

Mounting your google drive account

In [2]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=u
rn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fw
ww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/gdrive
```

In [3]:
```
! cd "gdrive/My Drive/datasets/movieLensSmall" && ls -la
```

```
total 3227
-rw------- 1 root root  197979 Oct  7  2018 links.csv
-rw------- 1 root root  494431 Oct  7  2018 movies.csv
-rw------- 1 root root 2483723 Oct  7  2018 ratings.csv
-rw------- 1 root root    8342 Oct  7  2018 README.txt
-rw------- 1 root root  118660 Oct  7  2018 tags.csv
```

In [4]:

```
!fold -w 80 -s "gdrive/My Drive/datasets/movieLensSmall/README.txt"
```

Summary
=======

This dataset (ml-latest-small) describes 5-star rating and free-text tagging activity from [MovieLens](http://movielens.org), a movie recommendation service. It contains 100836 ratings and 3683 tag applications across 9742 movies. These data were created by 610 users between March 29, 1996 and September 24, 2018. This dataset was generated on September 26, 2018.

Users were selected at random for inclusion. All selected users had rated at least 20 movies. No demographic information is included. Each user is represented by an id, and no other information is provided.

The data are contained in the files `links.csv`, `movies.csv`, `ratings.csv` and `tags.csv`. More details about the contents and use of all these files follows.

This is a *development* dataset. As such, it may change over time and is not an appropriate dataset for shared research results. See available *benchmark* datasets if that is your intent.

This and other GroupLens data sets are publicly available for download at <http://grouplens.org/datasets/>.

Usage License
=============

Neither the University of Minnesota nor any of the researchers involved can guarantee the correctness of the data, its suitability for any particular purpose, or the validity of results based on the use of the data set. The data set may be used for any research purposes under the following conditions:

* The user may not state or imply any endorsement from the University of Minnesota or the GroupLens Research Group.
* The user must acknowledge the use of the data set in publications resulting from the use of the data set (see below for citation information).
* The user may redistribute the data set, including transformations, so long as it is distributed under these same license conditions.
* The user may not use this information for any commercial or revenue-bearing purposes without first obtaining permission from a faculty member of the GroupLens Research Project at the University of Minnesota.
* The executable software scripts are provided "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of them is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

In no event shall the University of Minnesota, its affiliates or employees be liable to you for any damages arising out of the use or inability to use these programs (including but not limited to loss of data or data being rendered inaccurate).

If you have any further questions or comments, please email <grouplens-info@umn.edu>

Citation
========

To acknowledge use of the dataset in publications, please cite the following paper:

> F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4: 19:1–19:19. <https://doi.org/10.1145/2827872>

Further Information About GroupLens
===================================

GroupLens is a research group in the Department of Computer Science and Engineering at the University of Minnesota. Since its inception in 1992, GroupLens's research projects have explored a variety of fields including:

* recommender systems
* online communities
* mobile and ubiquitious technologies
* digital libraries
* local geographic information systems

GroupLens Research operates a movie recommender based on collaborative filtering, MovieLens, which is the source of these data. We encourage you to visit <http://movielens.org> to try it out! If you have exciting ideas for experimental work to conduct on MovieLens, send us an email at <grouplens-info@cs.umn.edu> - we are always interested in working with external collaborators.

Content and Use of Files
========================

Formatting and Encoding

The dataset files are written as [comma-separated values](http://en.wikipedia.org/wiki/Comma-separated_values) files with a single header row. Columns that contain commas (`,`) are escaped using double-quotes (`"`). These files are encoded as UTF-8. If accented characters in movie titles or tag values (e.g. Misérables, Les (1995)) display incorrectly, make sure that any program reading the data, such as a text editor, terminal, or script, is configured for UTF-8.

User Ids

MovieLens users were selected at random for inclusion. Their ids have been anonymized. User ids are consistent between `ratings.csv` and `tags.csv` (i.e., the same id refers to the same user across the two files).

Movie Ids

Only movies with at least one rating or tag are included in the dataset. These movie ids are consistent with those used on the MovieLens web site (e.g., id `1` corresponds to the URL <https://movielens.org/movies/1>). Movie ids are consistent between `ratings.csv`, `tags.csv`, `movies.csv`, and `links.csv` (i.e., the same id refers to the same movie across these four data files).

```
Ratings Data File Structure (ratings.csv)
-----------------------------------------

All ratings are contained in the file `ratings.csv`. Each line of this file
after the header row represents one rating of one movie by one user, and has
the following format:

    userId,movieId,rating,timestamp

The lines within this file are ordered first by userId, then, within user, by
movieId.

Ratings are made on a 5-star scale, with half-star increments (0.5 stars - 5.0
stars).

Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of
January 1, 1970.


Tags Data File Structure (tags.csv)
-----------------------------------

All tags are contained in the file `tags.csv`. Each line of this file after the
header row represents one tag applied to one movie by one user, and has the
following format:

    userId,movieId,tag,timestamp

The lines within this file are ordered first by userId, then, within user, by
movieId.

Tags are user-generated metadata about movies. Each tag is typically a single
word or short phrase. The meaning, value, and purpose of a particular tag is
determined by each user.

Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of
January 1, 1970.


Movies Data File Structure (movies.csv)
---------------------------------------

Movie information is contained in the file `movies.csv`. Each line of this file
after the header row represents one movie, and has the following format:

    movieId,title,genres

Movie titles are entered manually or imported from
<https://www.themoviedb.org/>, and include the year of release in parentheses.
Errors and inconsistencies may exist in these titles.

Genres are a pipe-separated list, and are selected from the following:

* Action
* Adventure
* Animation
* Children's
* Comedy
* Crime
* Documentary
* Drama
* Fantasy
* Film-Noir
* Horror
* Musical
* Mystery
* Romance
* Sci-Fi
* Thriller
* War
* Western
* (no genres listed)


Links Data File Structure (links.csv)
---------------------------------------

Identifiers that can be used to link to other sources of movie data are
contained in the file `links.csv`. Each line of this file after the header row
represents one movie, and has the following format:

    movieId,imdbId,tmdbId

movieId is an identifier for movies used by <https://movielens.org>. E.g., the
movie Toy Story has the link <https://movielens.org/movies/1>.

imdbId is an identifier for movies used by <http://www.imdb.com>. E.g., the
movie Toy Story has the link <http://www.imdb.com/title/tt0114709/>.

tmdbId is an identifier for movies used by <https://www.themoviedb.org>. E.g.,
the movie Toy Story has the link <https://www.themoviedb.org/movie/862>.

Use of the resources listed above is subject to the terms of each provider.


Cross-Validation
----------------

Prior versions of the MovieLens dataset included either pre-computed
cross-folds or scripts to perform this computation. We no longer bundle either
of these features with the dataset, since most modern toolkits provide this as
a built-in feature. If you wish to learn about standard approaches to
cross-fold computation in the context of recommender systems evaluation, see
[LensKit](http://lenskit.org) for tools, documentation, and open-source code
examples.
```

In [5]:
```
!wc -l "gdrive/My Drive/datasets/movieLensSmall/links.csv"
!wc -l "gdrive/My Drive/datasets/movieLensSmall/movies.csv"
!wc -l "gdrive/My Drive/datasets/movieLensSmall/ratings.csv"
!wc -l "gdrive/My Drive/datasets/movieLensSmall/tags.csv"
```

```
9743 gdrive/My Drive/datasets/movieLensSmall/links.csv
9743 gdrive/My Drive/datasets/movieLensSmall/movies.csv
100837 gdrive/My Drive/datasets/movieLensSmall/ratings.csv
3684 gdrive/My Drive/datasets/movieLensSmall/tags.csv
```

In [6]:
```
!head -n5 "gdrive/My Drive/datasets/movieLensSmall/links.csv"
```

```
movieId,imdbId,tmdbId

1,0114709,862

2,0113497,8844

3,0113228,15602

4,0114885,31357
```

In [7]:
```
!head -n5 "gdrive/My Drive/datasets/movieLensSmall/movies.csv"
```

```
movieId,title,genres

1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy

2,Jumanji (1995),Adventure|Children|Fantasy

3,Grumpier Old Men (1995),Comedy|Romance

4,Waiting to Exhale (1995),Comedy|Drama|Romance
```

In [8]:
```
!head -n5 "gdrive/My Drive/datasets/movieLensSmall/ratings.csv"
```

```
userId,movieId,rating,timestamp

1,1,4.0,964982703

1,3,4.0,964981247

1,6,4.0,964982224

1,47,5.0,964983815
```

In [9]:
```
!head -n5 "gdrive/My Drive/datasets/movieLensSmall/tags.csv"
```

```
userId,movieId,tag,timestamp

2,60756,funny,1445714994

2,60756,Highly quotable,1445714996

2,60756,will ferrell,1445714992

2,89774,Boxing story,1445715207
```

Dataset exploration

Loading dependencies

In [0]:
```
filepath_links = 'gdrive/My Drive/datasets/movieLensSmall/links.csv'
filepath_movies = 'gdrive/My Drive/datasets/movieLensSmall/movies.csv'
filepath_ratings = 'gdrive/My Drive/datasets/movieLensSmall/ratings.csv'
filepath_tags = 'gdrive/My Drive/datasets/movieLensSmall/tags.csv'
```

In [11]:
```
df_links = pd.read_csv(filepath_links)
df_links.head(10)
df_links.describe()
```

Out[11]:

	movieId	imdbId	tmdbId
count	9742.000000	9.742000e+03	9734.000000
mean	42200.353623	6.771839e+05	55162.123793
std	52160.494854	1.107228e+06	93653.481487
min	1.000000	4.170000e+02	2.000000
25%	3248.250000	9.518075e+04	9665.500000
50%	7300.000000	1.672605e+05	16529.000000
75%	76232.000000	8.055685e+05	44205.750000
max	193609.000000	8.391976e+06	525662.000000

In [12]:
```
df_movies =  pd.read_csv(filepath_movies)
df_movies.head(10)
```

Out[12]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
5	6	Heat (1995)	Action|Crime|Thriller
6	7	Sabrina (1995)	Comedy|Romance
7	8	Tom and Huck (1995)	Adventure|Children
8	9	Sudden Death (1995)	Action
9	10	GoldenEye (1995)	Action|Adventure|Thriller

Se puede observar que la columna que contien el título de la película, también contiene el año en el que esta ha sido lanzada, así que mejor vamos a separar esta información en dos columnas separadas

In [13]:
```
df_movies['has_year'] = df_movies['title'].apply(lambda x: "(" in x)
df_movies['has_year'].value_counts()
```

Out[13]:
```
True     9730
False      12
Name: has_year, dtype: int64
```

Parece que hay películas que no contienen la fecha en el título

In [14]:
```
df_movies[df_movies['has_year'] == False]
```

Out[14]:

	movieId	title	genres	has_year
6059	40697	Babylon 5	Sci-Fi	False
9031	140956	Ready Player One	Action|Sci-Fi|Thriller	False
9091	143410	Hyena Road	(no genres listed)	False
9138	147250	The Adventures of Sherlock Holmes and Doctor W...	(no genres listed)	False
9179	149334	Nocturnal Animals	Drama|Thriller	False
9259	156605	Paterson	(no genres listed)	False
9367	162414	Moonlight	Drama	False
9448	167570	The OA	(no genres listed)	False
9514	171495	Cosmos	(no genres listed)	False
9515	171631	Maria Bamford: Old Baby	(no genres listed)	False
9525	171891	Generation Iron 2	(no genres listed)	False
9611	176601	Black Mirror	(no genres listed)	False

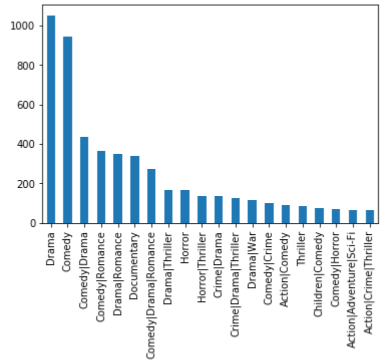
Parece que la columna de géneros siempre tiene los generos ordenados por categoría para cada entrada, así que vamos ver que agrupación son las más frecuentes

In [15]:
```
df_movies['genres'].value_counts()
```

Out[15]:
```
Drama                                          1053
Comedy                                          946
Comedy|Drama                                    435
Comedy|Romance                                  363
Drama|Romance                                   349
Documentary                                     339
Comedy|Drama|Romance                            276
Drama|Thriller                                  168
Horror                                          167
Horror|Thriller                                 135
Crime|Drama                                     134
Crime|Drama|Thriller                            125
Drama|War                                       114
Comedy|Crime                                    101
Action|Comedy                                    92
Thriller                                         84
Children|Comedy                                  74
Comedy|Horror                                    69
Action|Adventure|Sci-Fi                          66
Action|Crime|Thriller                            66
Action|Drama                                     62
Action|Crime|Drama|Thriller                      61
Action                                           60
Action|Thriller                                  60
Horror|Sci-Fi                                    53
Action|Crime|Drama                               50
Crime|Thriller                                   45
Drama|Musical                                    44
Action|Sci-Fi|Thriller                           43
Action|Drama|Thriller                            43
                                               ... 
Action|Adventure|Sci-Fi|War|IMAX                  1
Comedy|Documentary|Drama|Romance                  1
Action|Adventure|Mystery|Romance|Thriller         1
Action|Fantasy|Thriller|IMAX                      1
Crime|Drama|Film-Noir|Romance|Thriller            1
Adventure|Crime|Drama|Thriller                    1
Action|Adventure|Crime|Horror|Thriller            1
Adventure|Romance|Thriller                        1
Crime|Horror|Sci-Fi                               1
Action|Animation|Children|Comedy|IMAX             1
Children|Drama|War                                1
Adventure|Documentary|Western                     1
Action|Comedy|Sci-Fi|Western                      1
Children|Musical|Mystery                          1
Adventure|Animation|Children|Western              1
Adventure|Fantasy|Romance|Sci-Fi|Thriller         1
Adventure|Children|Comedy|Drama|Fantasy|Sci-Fi    1
Action|Animation|Children|Comedy|Sci-Fi|IMAX      1
Adventure|Comedy|Fantasy|Romance                  1
Comedy|Crime|Drama|Fantasy                        1
Animation|Children|Comedy|Musical|Romance         1
Comedy|Crime|Sci-Fi                               1
Adventure|Animation|Fantasy|Romance               1
Comedy|Documentary|Romance                        1
Adventure|Comedy|Crime|Thriller                   1
Animation|Comedy|Fantasy|Musical|Romance          1
Adventure|Animation|Comedy|Fantasy|IMAX           1
Drama|Musical|Mystery                             1
Adventure|Romance|Sci-Fi|IMAX                     1
Animation|Children|Comedy|Fantasy|Musical         1
Name: genres, Length: 951, dtype: int64
```

si probamos a representalo

In [16]:
```
df_movies['genres'].value_counts()[:20].plot(kind='bar')
```
Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa0577121d0>



In [17]:
```
df_movies.describe()
```
Out[17]:

	movieId
count	9742.000000
mean	42200.353623
std	52160.494854
min	1.000000
25%	3248.250000
50%	7300.000000
75%	76232.000000
max	193609.000000

In [18]:
```
df_ratings = pd.read_csv(filepath_ratings)
df_ratings.head(10)
```
Out[18]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931
5	1	70	3.0	964982400
6	1	101	5.0	964980868
7	1	110	4.0	964982176
8	1	151	5.0	964984041
9	1	157	5.0	964984100

In [19]:
```
df_movies.describe()
```
Out[19]:

	movieId
count	9742.000000
mean	42200.353623
std	52160.494854
min	1.000000
25%	3248.250000
50%	7300.000000
75%	76232.000000
max	193609.000000

In [20]:
```
df_tags = pd.read_csv(filepath_tags)
df_tags.head(10)
```
Out[20]:

	userId	movieId	tag	timestamp
0	2	60756	funny	1445714994
1	2	60756	Highly quotable	1445714996
2	2	60756	will ferrell	1445714992
3	2	89774	Boxing story	1445715207
4	2	89774	MMA	1445715200
5	2	89774	Tom Hardy	1445715205
6	2	106782	drugs	1445715054
7	2	106782	Leonardo DiCaprio	1445715051
8	2	106782	Martin Scorsese	1445715056
9	7	48516	way too long	1169687325

In [21]:
```
df_tags.describe()
```

Out[21]:

	userId	movieId	timestamp
count	3683.000000	3683.000000	3.683000e+03
mean	431.149335	27252.013576	1.320032e+09
std	158.472553	43490.558803	1.721025e+08
min	2.000000	1.000000	1.137179e+09
25%	424.000000	1262.500000	1.137521e+09
50%	474.000000	4454.000000	1.269833e+09
75%	477.000000	39263.000000	1.498457e+09
max	610.000000	193565.000000	1.537099e+09

Data wrangling

Extraemos el año del título de la película para dispones de el como un campo separado para el análisis

In [22]:
```
df_movies['year'] = df_movies.title.str.extract("\((\d{4})\)", expand=True)
df_movies.year = pd.to_datetime(df_movies.year, format='%Y')
df_movies.year = df_movies.year.dt.year # As there are some NaN years, resulting type will be float (decimals)
df_movies.title = df_movies.title.str[:-7]

df_movies.head()
```

Out[22]:

	movieId	title	genres	has_year	year
0	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	True	1995.0
1	2	Jumanji	Adventure|Children|Fantasy	True	1995.0
2	3	Grumpier Old Men	Comedy|Romance	True	1995.0
3	4	Waiting to Exhale	Comedy|Drama|Romance	True	1995.0
4	5	Father of the Bride Part II	Comedy	True	1995.0

Transforma los generos asociados a cada categoría como un One Hot Encoding

In [23]:
```
# Categorize movies genres properly. Working later with +20MM rows of strings proved very resource consuming
genres_unique = pd.DataFrame(df_movies.genres.str.split('|').tolist()).stack().unique()
genres_unique = pd.DataFrame(genres_unique, columns=['genre']) # Format into DataFrame to store later
df_movies = df_movies.join(df_movies.genres.str.get_dummies().astype(bool))
df_movies.drop('genres', inplace=True, axis=1)
df_movies.head()
```

Out[23]:

	movieId	title	has_year	year	(no genres listed)	Action	Adventure	Animation	Children	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror	IMAX	Musical	Mystery	Romance	Sci-Fi	Thriller	W
0	1	Toy Story	True	1995.0	False	False	True	True	True	True	False	False	False	True	False	False	False	False	False	False	False	False	Fal
1	2	Jumanji	True	1995.0	False	False	True	False	True	False	False	False	False	True	False	False	False	False	False	False	False	False	Fal
2	3	Grumpier Old Men	True	1995.0	False	False	False	False	False	True	False	False	False	False	False	False	False	False	False	True	False	False	Fal
3	4	Waiting to Exhale	True	1995.0	False	False	False	False	False	True	False	False	True	False	False	False	False	False	False	True	False	False	Fal
4	5	Father of the Bride Part II	True	1995.0	False	False	False	False	False	True	False	False	False	False	False	False	False	False	False	False	False	False	Fal

Otra transformación que vamos a realizar con el fin de comprender mejor el dataset es transformar el timestamp de las calificaciones realizadas por los usuarios a un formato más manejable

In [24]:
```
# Modify rating timestamp format (from seconds to datetime year)
#ratings.timestamp = pd.to_datetime(ratings.timestamp, unit='s')
df_ratings.timestamp = pd.to_datetime(df_ratings.timestamp, infer_datetime_format=True)
df_ratings.timestamp = df_ratings.timestamp.dt.year
df_ratings.head()
```

Out[24]:

	userId	movieId	rating	timestamp
0	1	1	4.0	1970
1	1	3	4.0	1970
2	1	6	4.0	1970
3	1	47	5.0	1970
4	1	50	5.0	1970

Finalmente comprobamos los registros de cada dataframe que pueda contener valores nulos y al tratarse de unos pocos, simplemente prescendiremos de ellos

In [25]:
```
# Check and clean NaN values
print ("Number of movies Null values: ", max(df_movies.isnull().sum()))
print ("Number of ratings Null values: ", max(df_ratings.isnull().sum()))
df_movies.dropna(inplace=True)
df_ratings.dropna(inplace=True)
```

```
Number of movies Null values:  13
Number of ratings Null values:  0
```

Obtenemos todas las valoraciones realizadas para cada película

In [26]:
```
df_movie_rates = df_movies.set_index('movieId').join(df_ratings.set_index('movieId'))
df_movie_rates.head()
```

Out[26]:

	title	has_year	year	(no genres listed)	Action	Adventure	Animation	Children	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror	IMAX	Musical	Mystery	Romance	Sci-Fi	Thriller	War	We
movieId																							
1	Toy Story	True	1995.0	False	False	True	True	True	True	False	False	False	True	False	False	False	False	False	False	False	False	False	
1	Toy Story	True	1995.0	False	False	True	True	True	True	False	False	False	True	False	False	False	False	False	False	False	False	False	
1	Toy Story	True	1995.0	False	False	True	True	True	True	False	False	False	True	False	False	False	False	False	False	False	False	False	
1	Toy Story	True	1995.0	False	False	True	True	True	True	False	False	False	True	False	False	False	False	False	False	False	False	False	
1	Toy Story	True	1995.0	False	False	True	True	True	True	False	False	False	True	False	False	False	False	False	False	False	False	False	

In [27]:
```python
avg_rate = df_movie_rates[['title','rating']].groupby('title').mean().sort_values(by='rating', ascending=False)
avg_rate
```

Out[27]:

title	rating
Formula of Love	5.0
Down Argentine Way	5.0
Mother (Madeo)	5.0
Light Years (Gandahar)	5.0
Tokyo Tribe	5.0
Hunting Elephants	5.0
Big Top Scooby-Doo!	5.0
Into the Forest of Fireflies' Light	5.0
Goodbye Charlie	5.0
Eichmann	5.0
Bill Hicks: Revelations	5.0
Moscow Does Not Believe in Tears (Moskva slezam ne verit)	5.0
All the Vermeers in New York	5.0
Saving Face	5.0
Martin Lawrence Live: Runteldat	5.0
Lady Jane	5.0
Who Killed Chea Vichea?	5.0
All Yours	5.0
Saving Santa	5.0
Snowflake, the White Gorilla	5.0
Bitter Lake	5.0
More	5.0
Investigation Held by Kolobki	5.0
Dylan Moran: Monster	5.0
Scooby-Doo Goes Hollywood	5.0
Tom Segura: Completely Normal	5.0
Into the Abyss	5.0
Man and a Woman, A (Un homme et une femme)	5.0
Empties	5.0
Louis Theroux: Law & Disorder	5.0
...	...
Indestructible Man	0.5
Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie)	0.5
Yongary: Monster from the Deep	0.5
3 Ninjas Knuckle Up	0.5
Don't Look Now	0.5
3 dev adam (Three Giant Men)	0.5
Giant Spider Invasion, The	0.5
In the Name of the King: A Dungeon Siege Tale	0.5
Captain America II: Death Too Soon	0.5
My Bloody Valentine	0.5
Carabineers, The (Carabiniers, Les)	0.5
Carnival Magic	0.5
Glitter	0.5
Idaho Transfer	0.5
Mortal Kombat: The Journey Begins	0.5
Call Northside 777	NaN
Chalet Girl	NaN
Chosen, The	NaN
Color of Paradise, The (Rang-e khoda)	NaN
For All Mankind	NaN
I Know Where I'm Going!	NaN
In the Realms of the Unreal	NaN
Innocents, The	NaN
Niagara	NaN
Parallax View, The	NaN
Road Home, The (Wo de fu qin mu qin)	NaN
Roaring Twenties, The	NaN
Scrooge	NaN
This Gun for Hire	NaN
Twentieth Century	NaN

9448 rows × 1 columns

In [28]:
```python
number_of_rates = df_movie_rates[['title','rating']].groupby('title').size().sort_values(ascending=False).rename('count').to_frame()
number_of_rates
```

Out[28]:

	count
title	
Forrest Gump	329
Shawshank Redemption, The	317
Pulp Fiction	307
Silence of the Lambs, The	279
Matrix, The	278
Star Wars: Episode IV - A New Hope	251
Jurassic Park	238
Braveheart	237
Terminator 2: Judgment Day	224
Schindler's List	220
Fight Club	218
Toy Story	215
Star Wars: Episode V - The Empire Strikes Back	211
American Beauty	204
Usual Suspects, The	204
Seven (a.k.a. Se7en)	203
Independence Day (a.k.a. ID4)	202
Apollo 13	201
Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark)	200
Lord of the Rings: The Fellowship of the Ring, The	198
Star Wars: Episode VI - Return of the Jedi	196
Batman	194
Godfather, The	192
Fugitive, The	191
Lord of the Rings: The Two Towers, The	188
Saving Private Ryan	188
Lord of the Rings: The Return of the King, The	185
Aladdin	183
Fargo	181
Gladiator	179
...	...
Legend of Rita, The (Stille nach dem Schuß, Die)	1
Legionnaire	1
Lemonade	1
Leningrad Cowboys Go America	1
Leprechaun 2	1
Leprechaun 3	1
Leprechaun 4: In Space	1
Let It Be Me	1
Let It Snow	1
Let the Bullets Fly	1
Le Maître d'école	1
Latter Days	1
Last Man on Earth, The (Ultimo uomo della Terra, L')	1
Late Shift, The	1
Last Metro, The (Dernier métro, Le)	1
Last Night	1
Last Orders	1
Last Shift	1
Last Song, The	1
Last Train Home	1
Last Waltz, The	1
Last Wave, The	1
Last Wedding, The (Kivenpyörittäjän kylä)	1
Last Winter, The	1
Last Year's Snow Was Falling	1
Last of the Dogmen	1
Late Marriage (Hatuna Meuheret)	1
Late Night Shopping	1
Late Night with Conan O'Brien: The Best of Triumph the Insult Comic Dog	1
'71	1

9448 rows × 1 columns

In [29]:
```
avg_rate.join(number_of_rates)
```

Out[29]:

title	rating	count
Formula of Love	5.0	1
Down Argentine Way	5.0	1
Mother (Madeo)	5.0	1
Light Years (Gandahar)	5.0	1
Tokyo Tribe	5.0	1
Hunting Elephants	5.0	1
Big Top Scooby-Doo!	5.0	1
Into the Forest of Fireflies' Light	5.0	1
Goodbye Charlie	5.0	1
Eichmann	5.0	1
Bill Hicks: Revelations	5.0	1
Moscow Does Not Believe in Tears (Moskva slezam ne verit)	5.0	1
All the Vermeers in New York	5.0	1
Saving Face	5.0	1
Martin Lawrence Live: Runteldat	5.0	1
Lady Jane	5.0	1
Who Killed Chea Vichea?	5.0	1
All Yours	5.0	1
Saving Santa	5.0	1
Snowflake, the White Gorilla	5.0	1
Bitter Lake	5.0	1
More	5.0	1
Investigation Held by Kolobki	5.0	1
Dylan Moran: Monster	5.0	1
Scooby-Doo Goes Hollywood	5.0	1
Tom Segura: Completely Normal	5.0	1
Into the Abyss	5.0	1
Man and a Woman, A (Un homme et une femme)	5.0	1
Empties	5.0	1
Louis Theroux: Law & Disorder	5.0	1
...	...	...
Indestructible Man	0.5	1
Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie)	0.5	2
Yongary: Monster from the Deep	0.5	1
3 Ninjas Knuckle Up	0.5	1
Don't Look Now	0.5	1
3 dev adam (Three Giant Men)	0.5	1
Giant Spider Invasion, The	0.5	1
In the Name of the King: A Dungeon Siege Tale	0.5	1
Captain America II: Death Too Soon	0.5	1
My Bloody Valentine	0.5	1
Carabineers, The (Carabiniers, Les)	0.5	1
Carnival Magic	0.5	1
Glitter	0.5	1
Idaho Transfer	0.5	1
Mortal Kombat: The Journey Begins	0.5	1
Call Northside 777	NaN	1
Chalet Girl	NaN	1
Chosen, The	NaN	1
Color of Paradise, The (Rang-e khoda)	NaN	1
For All Mankind	NaN	1
I Know Where I'm Going!	NaN	1
In the Realms of the Unreal	NaN	1
Innocents, The	NaN	1
Niagara	NaN	1
Parallax View, The	NaN	1
Road Home, The (Wo de fu qin mu qin)	NaN	1
Roaring Twenties, The	NaN	1
Scrooge	NaN	1
This Gun for Hire	NaN	1
Twentieth Century	NaN	1

9448 rows × 2 columns

Obtenemos todas las valoraciones realizada por cada usuario

In [30]:
```python
df_each_user_ratings = df_ratings \
  .pivot(index="userId", columns="movieId", values="rating") \
  .fillna(0)

df_each_user_ratings.head()
```

Out[30]:

movieId userId	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	34	36	38	39	40	41	42	43	...	185135
1	4.0	0.0	4.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
5	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	4.0	0.0	3.0	0.0	0.0	0.0	0.0	...	0.0

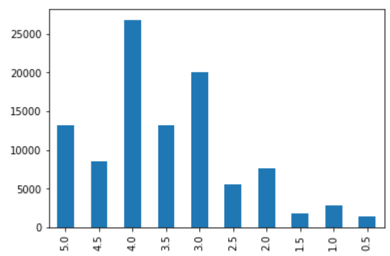
5 rows × 9724 columns

Data visualization

Como se distribuyen las valoraciones de los usuarios entre las películas

In [31]:
```python
df_movie_rates[['title','rating']] \
  .rating.value_counts() \
  .sort_index(ascending=False) \
  .plot(kind='bar')
```

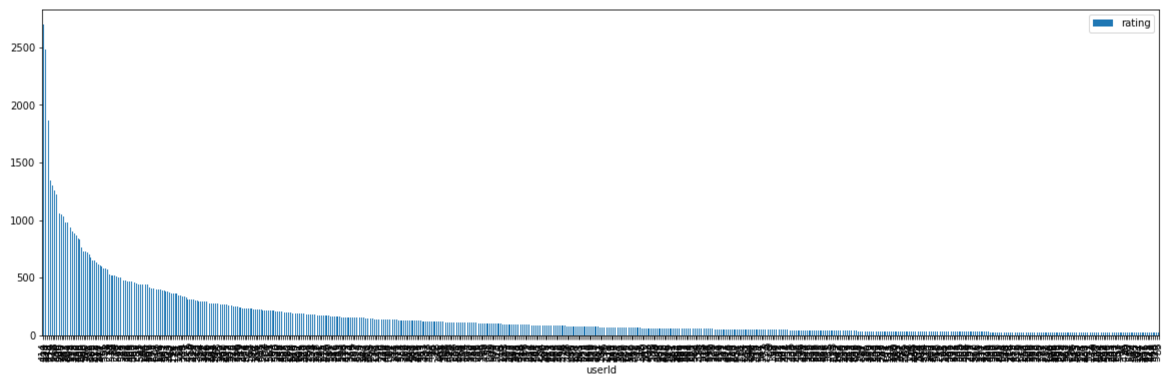
Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa05776f860>



Número de valoraciones por los usuarios

In [32]:
```python
df_ratings[['userId','rating']] \
  .groupby('userId') \
  .count() \
  .sort_values(by='rating',ascending=False) \
  .plot(kind='bar',figsize=(20,6))
```

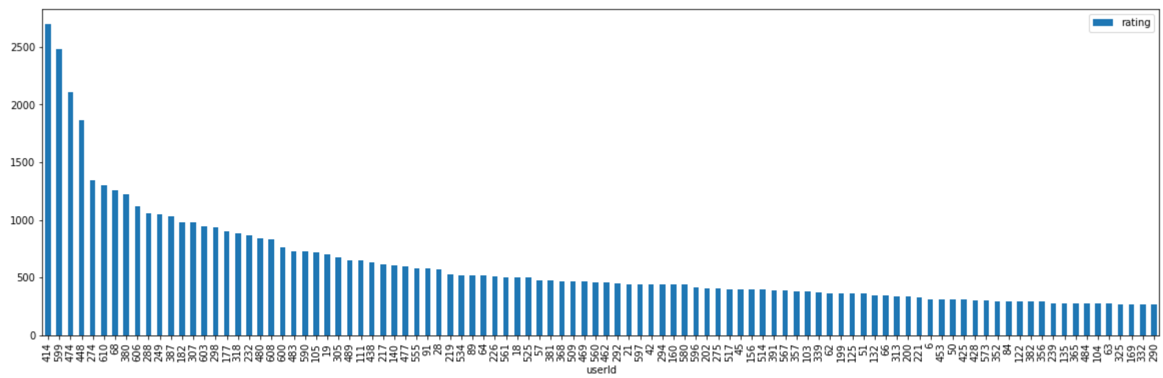
Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa057740160>



Hacemos zoom para mostrar el número de valoraciones de los 100 usuarios que más valoraciones han realizado

In [33]:
```python
df_ratings[['userId','rating']] \
  .groupby('userId') \
  .count() \
  .sort_values(by='rating',ascending=False)[0:100] \
  .plot(kind='bar',figsize=(20,6))
```

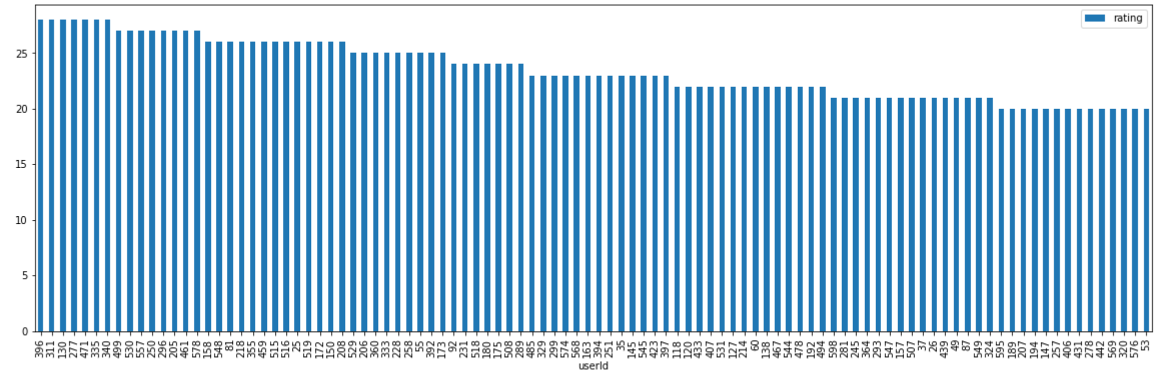
Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa0536ca6d8>



Ahora hacemos zoom, para ver los 100 usuarios del dataset que menos valoraciones han realizado

In [34]:
```
df_ratings[['userId','rating']] \
  .groupby('userId') \
  .count() \
  .sort_values(by='rating',ascending=False)[-100:] \
  .plot(kind='bar',figsize=(20,6))
```

Out[34]: `<matplotlib.axes._subplots.AxesSubplot at 0x7fa052d2bcf8>`



Esta última gráfica resulta interesante, porque se puede ver que el dataset ha sido preparado de forma que todos los usuarios que realizasen menos de 20 valoraciones han sido excluidos, por lo tanto podemos asumir que partimos de un mínimo de información sobre los gustos de cada usuario del dataset

Factorización de Matrices

Aqui procedere a explica la tećnica utilizada para la generación de recomendaciones individuales y a usarla

Singular Value Decomposition

Singular Value Decomposition o *SVD*, es una técnia que factorización de matrices que proclamaa que dara una matriz **A**, esta puede descomponerse de la siguiente forma:

$A_{m \times n} \approx U_{m \times r} \Sigma_{r \times r} V_{n \times r}^{T}$

Donde:

- A: es la matriz con los datos de entrada a factorizar
  - matriz m x n (m documentos, n terminos)
- U: es la matriz izquierda de vectores de valores singulares
  - matriz m x r (m documentos, r conceptos/ratings)
- Σ: valores singulares
  - matriz diagonal r x r (representa el peso de cada concepto)
  - r; rango de la matriz A
- V: es la matriz derecha de vectores de valores singulares
  - matriz n x r (n terminos, r conceptos/ratings)

En la *figura 1* podemos ver un ejemplo de la representación de las matrices aplicado a *NLP* (Natural Language Processing), en el que las columnas de la matriz A representan frases y las filas representan (mediante un índice) la pertenencia de una palabra a las diferentes frases.

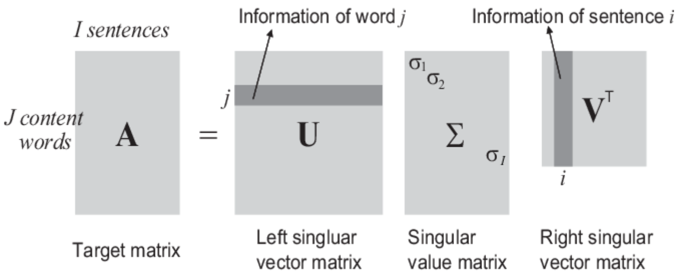


figura1: imagen ilustrando las matrices

Propiedades

SVD establece que **siempre** es posible descomponen la matriz A en $A \approx U\Sigma V$, de tal forma que:

- U, Σ, V: son únicas
- U, V: tienen columnas ortonormales
  - $U^T U = I, V^T V = I$
  - Las columnas son vectores unitarios ortogonales
- Σ: diagonal
  - Las entradas (valores singulares) son positivas y se encuentran ordenadas decrecientemente ($\sigma_1 \geq \sigma_2 \geq ... \geq 0$)

Como realizar la factorización

Una vez explicado en que consiste *SVD*, el siguiente paso es plantearse como realizar el proceso de decomposición de la matriz A. Esto se puede plantear como un proceso de optimización en el que vamos generando matrices U y V a la vez que observamos como la matriz resultante $\hat{A}$ difiere de la matriz original A, donde para ello podemos utilizar una métrica de error como el RMSE.

$error(A, U, V) = RMSE(A.\hat{A}) = RMSE(A.UV)$

$RMSE(A.UV)) = \sqrt{\sum_{u,i}(\hat{r}_{ui} - r_{ui})^2}$ donde $\hat{r} \in UV$ y $r \in A$

Se puede ver que se está obviando la matriz diagonal Σ en este proceso, esto se debe a que en la práctica, podemos considerar que esta matriz ya se encuentra integrada en las matrices U y V, simplicando así las operaciones necesarias para el proceso de obtención de dichas matrices.

Ahora que tenemos una métrica de error y planteado el problema, podemos aplicar un mecanismo de optimización como SGD (Stochastic Gradiand Descend) e ir iterando de forma que, en cada iteración, generemos un nuevo par de matrices U y V guiadas por el descenso del gradiente, que traten de minimizar el error cometido al generar la matriz $\hat{A}$

Implementación simple

In [35]:
```
import numpy as np

np.random.seed(1337)

A = np.random.rand(10, 10)
A = A * A

# prettify print options for matrix
np.set_printoptions(formatter={'float': '{: 0.3f}'.format})
print(A)

# set print options back to normal
np.set_printoptions(edgeitems=3,infstr='inf', linewidth=75, nanstr='nan', precision=8, suppress=False, threshold=1000, formatter=None)
```

```
[[ 0.069  0.025  0.077  0.211  0.103  0.269  0.069  0.953  0.537  0.013]
 [ 0.149  0.395  0.016  0.967  0.196  0.623  0.631  0.131  0.173  0.341]
 [ 0.578  0.035  0.083  0.449  0.250  0.032  0.171  0.040  0.283  0.693]
 [ 0.034  0.917  0.181  0.254  0.261  0.000  0.535  0.987  0.027  0.016]
 [ 0.141  0.481  0.000  0.136  0.003  0.623  0.122  0.494  0.241  0.946]
 [ 0.699  0.372  0.319  0.995  0.065  0.000  0.008  0.882  0.948  0.242]
 [ 0.116  0.523  0.000  0.578  0.451  0.036  0.444  0.830  0.026  0.829]
 [ 0.105  0.490  0.070  0.269  0.031  0.219  0.202  0.157  0.627  0.243]
 [ 0.524  0.629  0.124  0.827  0.508  0.805  0.177  0.219  0.817  0.401]
 [ 0.284  0.056  0.896  0.313  0.574  0.056  0.185  0.155  0.261  0.016]]
```

In [0]:
```
def simple_SGD(data,n_factors = 10, alpha = .01, n_epochs = 10):
  '''Learn the vectors p_u and q_i with SGD.
     data is the user-item matrix
     n_factor is the number of latent factors to use
     alppha is the learning rate of the SGD
     n_epochs is the number of iterations to run the algorithm
  '''
  shape = np.shape(data)
  n_users = shape[0]
  n_items = shape[1]

  # Randomly initialize the user and item factors.
  p = np.random.normal(0, .1, (n_users, n_factors))
  q = np.random.normal(0, .1, (n_items, n_factors))

  # Optimization procedure
  for _ in range(n_epochs):
      for (u, i), r_ui in np.ndenumerate(data):
          err = r_ui - np.dot(p[u], q[i])
          # Update vectors p_u and q_i
          p[u] += alpha * err * q[i]
          q[i] += alpha * err * p[u]

  return p,q

def rmse(U,V):
  errors = U - V
  return np.sqrt(np.sum(errors*errors) / errors.size)



n_factors = 5 # number o latent factors
alpha = .01 # learning rate
n_epochs = 5000 # number of iteration of the SGD procedure

u,v = simple_SGD(A,n_factors,alpha,n_epochs)
```

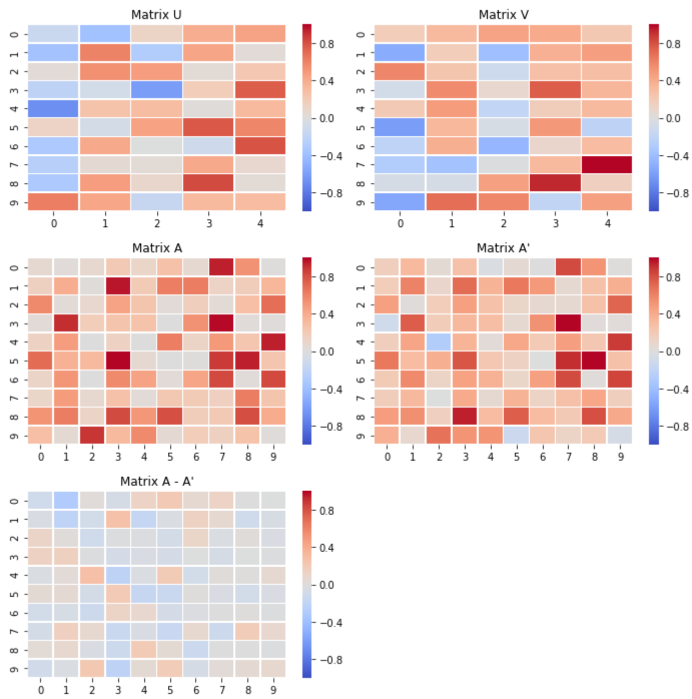
In [37]:
```
import seaborn as sns
import matplotlib.pylab as plt

print("RMSE: {}".format(rmse(A,u.dot(v.T))))

plt.subplots(figsize=(12,12))
plt.subplot(321)
ax = sns.heatmap(u, linewidth=0.5, vmin=-1, vmax=1, cmap="coolwarm").set_title("Matrix U")
plt.subplot(322)
ax = sns.heatmap(v, linewidth=0.5, vmin=-1, vmax=1, cmap="coolwarm").set_title("Matrix V")
plt.subplot(323)
ax = sns.heatmap(A, linewidth=0.5, vmin=-1, vmax=1, cmap="coolwarm").set_title("Matrix A")
plt.subplot(324)
ax = sns.heatmap(u.dot(v.T), linewidth=0.5, vmin=-1, vmax=1, cmap="coolwarm").set_title("Matrix A'")
plt.subplot(325)
ax = sns.heatmap(A - u.dot(v.T), linewidth=0.5, vmin=-1, vmax=1, cmap="coolwarm").set_title("Matrix A - A'")

plt.subplots_adjust(hspace=0.25, wspace=0.07)
plt.show()
```

```
RMSE: 0.10888291026731306
```



Adaptación para matrices dispersas

El algoritmo propuesto cumple con su cometido en el caso de que le proporcionemos una matriz a factorizar que no sea dispersa, pero el problema nos los encontramos cuando tratamos de aplicar dicho método sobre una matriz dispersa, ya que para valoraciones de las que no disponemos en la matriz, no podemos calcular el error que cometemos.

La solución a esta casuística resulta bastante sencilla, simplemente calculamos las matrices U y V teniendo en cuenta únicamente las valoraciones que tenemos para calcular el gradiente y los factores latentes que componen las matrices U y V. Si tenemos suficientes valoraciones, los factores latentes se ajustarán de tal forma que representen los gustos de los usuarios y las películas, lo que dará lugar a que sea capaz de generar recomendaciones adecuadas.

Implementación

In [38]:
```
import numpy as np

import seaborn as sns
import matplotlib.pylab as plt

np.random.seed(1337)

A = np.random.rand(10, 10)
A = A * A

A[0,1:5] = 0
A[1,5:9] = 0

# prettify print options for matrix
np.set_printoptions(formatter={'float': '{: 0.3f}'.format})
print(A)

# set print options back to normal
np.set_printoptions(edgeitems=3,infstr='inf', linewidth=75, nanstr='nan', precision=8, suppress=False, threshold=1000, formatter=None)
```

```
[[ 0.069  0.000  0.000  0.000  0.000  0.269  0.069  0.953  0.537  0.013]
 [ 0.149  0.395  0.016  0.967  0.196  0.000  0.000  0.000  0.000  0.341]
 [ 0.578  0.035  0.083  0.449  0.250  0.032  0.171  0.040  0.283  0.693]
 [ 0.034  0.917  0.181  0.254  0.261  0.000  0.535  0.987  0.027  0.016]
 [ 0.141  0.481  0.000  0.136  0.003  0.623  0.122  0.494  0.241  0.946]
 [ 0.699  0.372  0.319  0.995  0.065  0.000  0.008  0.882  0.948  0.242]
 [ 0.116  0.523  0.000  0.578  0.451  0.036  0.444  0.830  0.026  0.829]
 [ 0.105  0.490  0.070  0.269  0.031  0.219  0.202  0.157  0.627  0.243]
 [ 0.524  0.629  0.124  0.827  0.508  0.805  0.177  0.219  0.817  0.401]
 [ 0.284  0.056  0.896  0.313  0.574  0.056  0.185  0.155  0.261  0.016]]
```

In [39]:
```python
def simple_SGD2(data,n_factors = 10, alpha = .01, n_epochs = 10):
    '''Learn the vectors p_u and q_i with SGD.
       data is the user-item matrix
       n_factor is the number of latent factors to use
       alppha is the learning rate of the SGD
       n_epochs is the number of iterations to run the algorithm
    '''
    print(type(data))

    shape = np.shape(data)
    n_users = shape[0]
    n_items = shape[1]

    # Randomly initialize the user and item factors.
    p = np.random.normal(0, .1, (n_users, n_factors))
    q = np.random.normal(0, .1, (n_items, n_factors))

    # Optimization procedure
    for _ in range(n_epochs):
        for (u, i), r_ui in np.ndenumerate(data):
            if(r_ui > 0):
                err = r_ui - np.dot(p[u], q[i])
                # Update vectors p_u and q_i
                p[u] += alpha * err * q[i]
                q[i] += alpha * err * p[u]

    return p,q

def rmse(U,V):
    errors = U - V
    return np.sqrt(np.sum(errors*errors) / errors.size)



n_factors = 5 # number o latent factors
alpha = .01 # learning rate
n_epochs = 5000 # number of iteration of the SGD procedure

u,v = simple_SGD2(A,n_factors,alpha,n_epochs)
```

```
<class 'numpy.ndarray'>
```

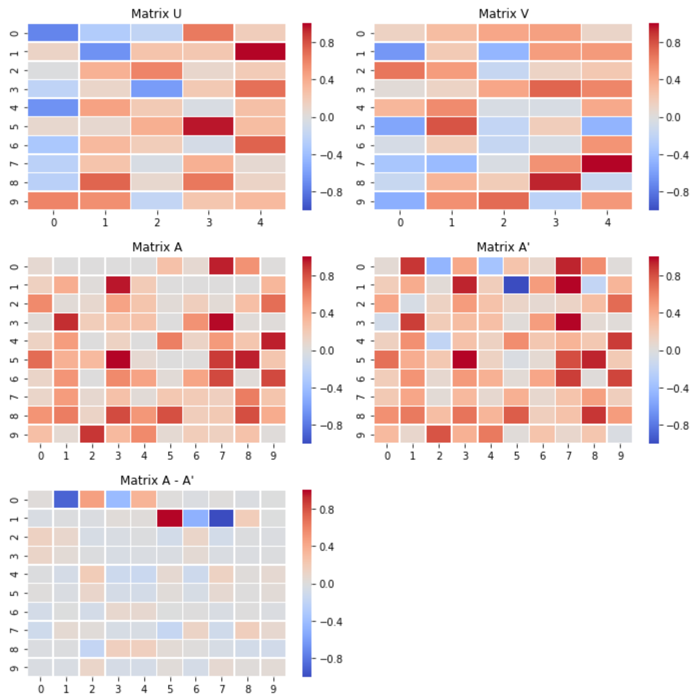
In [40]:
```python
import seaborn as sns
import matplotlib.pylab as plt

print("RMSE: {}".format(rmse(A,u.dot(v.T))))

plt.subplots(figsize=(12,12))
plt.subplot(321)
ax = sns.heatmap(u, linewidth=0.5, vmin=-1, vmax=1, cmap="coolwarm").set_title("Matrix U")
plt.subplot(322)
ax = sns.heatmap(v, linewidth=0.5, vmin=-1, vmax=1, cmap="coolwarm").set_title("Matrix V")
plt.subplot(323)
ax = sns.heatmap(A, linewidth=0.5, vmin=-1, vmax=1, cmap="coolwarm").set_title("Matrix A")
plt.subplot(324)
ax = sns.heatmap(u.dot(v.T), linewidth=0.5, vmin=-1, vmax=1, cmap="coolwarm").set_title("Matrix A'")
plt.subplot(325)
ax = sns.heatmap(A - u.dot(v.T), linewidth=0.5, vmin=-1, vmax=1, cmap="coolwarm").set_title("Matrix A - A'")

plt.subplots_adjust(hspace=0.25, wspace=0.07)
plt.show()
```

```
RMSE: 0.27191355220596636
```



In [41]:
```
# prettify print options for matrix
np.set_printoptions(formatter={'float': '{: 0.3f}'.format})
print("Matriz original de valoraciones")
print(A)
print("\nMatriz aprendida mediante la factorización")
print(u.dot(v.T))
# set print options back to normal
np.set_printoptions(edgeitems=3,infstr='inf', linewidth=75, nanstr='nan', precision=8, suppress=False, threshold=1000, formatter=None)
```

```
Matriz original de valoraciones
[[ 0.069  0.000  0.000  0.000  0.000  0.269  0.069  0.953  0.537  0.013]
 [ 0.149  0.395  0.016  0.967  0.196  0.000  0.000  0.000  0.000  0.341]
 [ 0.578  0.035  0.083  0.449  0.250  0.032  0.171  0.040  0.283  0.693]
 [ 0.034  0.917  0.181  0.254  0.261  0.000  0.535  0.987  0.027  0.016]
 [ 0.141  0.481  0.000  0.136  0.003  0.623  0.122  0.494  0.241  0.946]
 [ 0.699  0.372  0.319  0.995  0.065  0.000  0.008  0.882  0.948  0.242]
 [ 0.116  0.523  0.000  0.578  0.451  0.036  0.444  0.830  0.026  0.829]
 [ 0.105  0.490  0.070  0.269  0.031  0.219  0.202  0.157  0.627  0.243]
 [ 0.524  0.629  0.124  0.827  0.508  0.805  0.177  0.219  0.817  0.401]
 [ 0.284  0.056  0.896  0.313  0.574  0.056  0.185  0.155  0.261  0.016]]

Matriz aprendida mediante la factorización
[[ 0.051  0.891 -0.467  0.427 -0.357  0.255  0.087  0.943  0.556  0.019]
 [ 0.181  0.405  0.034  0.942  0.173 -1.209  0.489  1.963 -0.186  0.347]
 [ 0.436 -0.051  0.138  0.488  0.263  0.110  0.066  0.120  0.296  0.709]
 [-0.083  0.871  0.194  0.298  0.289  0.030  0.483  1.015  0.050  0.026]
 [ 0.151  0.543 -0.195  0.260  0.141  0.560  0.233  0.358  0.221  0.878]
 [ 0.686  0.397  0.223  1.061  0.137 -0.030  0.058  0.818  0.943  0.210]
 [ 0.187  0.525  0.072  0.496  0.375  0.040  0.437  0.868  0.021  0.851]
 [ 0.211  0.432  0.043  0.321  0.077  0.376  0.082  0.253  0.463  0.168]
 [ 0.540  0.641  0.296  0.668  0.351  0.759  0.167  0.265  0.898  0.492]
 [ 0.314  0.089  0.792  0.371  0.641  0.028  0.250  0.087  0.239 -0.026]]
```

Generación de predicciones

Lo primero es realizar la factorización de matrices sobre la matriz de valoraciones de usuarios aplicando SVD

In [42]:
```
n_factors = 50
alpha = 0.01
n_epochs = 100
u,v = simple_SGD2(df_each_user_ratings, n_factors, alpha, n_epochs)
```

```
<class 'pandas.core.frame.DataFrame'>
```

De cara a generar las recomendaciones para un usuario, llega con multiplica la fila correspondiente al usuario en la matriz que contiene los factores latentes de los usuarios($U[usuario,]$), por matriz traspuestas que contiene todos la factores latentes de las películas (V^T),

In [43]:
```
u[0].shape
```

Out[43]:
```
(50,)
```

In [44]:
```
v.T.shape
```

Out[44]:
```
(50, 9724)
```

In [45]:
```
u[0].dot(v.T)
```

Out[45]:
```
array([4.21591503, 3.79842813, 4.12853265, ..., 1.64174382, 2.47908129,
       3.21313069])
```

Si queremos generar todas las recomendaciones, para todos los usuarios, llegaría con multiplicar la matriz U con V^T. Que podemos ver que tiene las mismas dimensiones que la matriz de valoraciones original

In [46]:
```
u.dot(v.T).shape
```

Out[46]:
```
(610, 9724)
```

In [47]:
```
df_each_user_ratings.shape
```

Out[47]:
```
(610, 9724)
```

Tecnicas de recomendación para grupos

Dado que el dataset no contiene grupos como tal y muchos menos existen valoraciones explicitas realizadas por un grupo de usuarios(que permitiría abordar el problema simplemente tratando los grupos como usuarios), para la generación de recomendaciones grupales, se ha preferido optar por la exploración de agregaciones de las recomendaciones individuales de los usuarios que conforman un grupo. Además este tipo de agregaciones se puede aplicar sobre cualquier grupo de usuarios, sin necesidad de tener información previa del propio grupo, es decir, siempre podemos generar recomendaciones ante una nueva combinación de usuarios que se acabe de formar.

Lo primero que necesitamos para explorar estas agregaciones, es formar un grupo de usuarios que solicitan que les recomienden películas

In [48]:
```
total_users = u.shape[0]
group_size = 5

users = np.random.randint(total_users, size=group_size)
users
```

Out[48]:
```
array([383,  95, 366,  98, 477])
```

A continuación podemos ver los factores latentes de cada usuario que conforma el grupo aleatorio que se acaba de crear

In [49]:
```python
group_latent_factors = u[users]
group_latent_factors
```

Out[49]:
```
array([[ 1.12346363e+00,  1.23025842e-01,  2.48649284e-02,
        -5.78537232e-01,  4.70591683e-01,  1.25625866e+00,
         8.13316513e-01,  1.01539615e-01, -8.90614313e-02,
        -2.69570662e-01,  5.73283369e-01, -5.97172664e-02,
        -2.89190149e-01,  1.38697126e-01,  4.79102876e-01,
        -3.63304413e-01, -1.10059738e+00, -1.33656809e-01,
        -7.89061633e-01,  3.87304716e-01,  8.09312336e-03,
         1.80737735e-01, -3.11223671e-01, -2.91140426e-01,
         6.45881121e-01, -4.32242684e-01,  1.68279346e-01,
         6.90765220e-01, -2.16574216e-02, -1.94788958e-01,
        -1.21009076e-01,  1.02591955e+00,  3.83810220e-01,
        -6.48064231e-01,  1.22849215e-01, -7.36432750e-01,
        -6.07146870e-01,  6.69298463e-01, -3.49748528e-02,
        -3.06670989e-02,  1.41724746e-01, -1.87137870e-02,
         4.57159967e-01,  5.04796820e-01,  4.15830459e-02,
         2.10300472e-02, -3.70506348e-01,  1.49655546e+00,
        -3.06984690e-01,  1.21752005e-01],
       [ 7.39219008e-01,  7.00369271e-01,  2.79477468e-01,
         3.83259898e-02, -1.36954877e+00,  4.73883506e-01,
         6.22990037e-01,  3.24954188e-02, -1.28001179e+00,
        -3.19315619e-01, -1.77867587e-01, -3.37670390e-01,
         3.45242526e-01, -6.61859110e-01,  1.24921545e-03,
         4.08273096e-01, -3.23193318e-01,  7.83014811e-01,
        -6.76554880e-01,  1.52311273e-01,  6.63996339e-01,
        -4.91968836e-01, -6.87415146e-01, -1.26020907e-01,
         3.76105002e-01, -2.72936959e-01, -5.56934243e-01,
         5.84408879e-01, -7.98845417e-01, -3.21109982e-01,
         6.15217479e-01,  1.30554477e+00, -5.51668681e-01,
        -1.50265845e+00,  2.46879114e-01, -7.58321102e-01,
        -7.13350169e-02,  7.83272826e-01, -1.09662364e+00,
         2.45686133e-01,  9.93380889e-01,  8.78243336e-01,
         8.58953978e-01, -4.39960819e-01,  4.77764942e-02,
        -4.11871147e-01, -3.91026445e-01,  7.31471409e-01,
         5.81937297e-01, -5.17952335e-01],
       [ 1.11810159e+00, -1.41202417e-01,  6.67921394e-01,
         1.85175780e-01, -1.96680941e-01,  1.75285594e-01,
        -9.39967643e-01,  6.64848698e-01, -4.99058492e-01,
        -2.86282350e-01,  3.55640177e-02, -2.19169414e-01,
        -7.08410599e-01,  6.75682511e-02, -1.98217476e-01,
         1.41376885e+00, -1.13647594e+00,  7.88365608e-01,
        -1.03379440e+00, -8.52429001e-02,  6.88116093e-01,
         1.15901231e+00,  1.44027957e-02,  4.43850768e-01,
         1.04496181e+00,  2.43057985e-01,  6.15304691e-01,
         5.10069630e-01,  2.83705957e-01, -1.18476678e+00,
        -1.43524177e+00,  1.87479157e+00,  2.44819642e-01,
        -8.91283314e-01,  1.02414781e+00, -3.09955469e-01,
         1.28884751e-01,  1.43894259e+00, -8.58145730e-01,
        -1.10245928e-01,  2.45500821e-03,  2.23518199e-01,
        -1.04415686e+00,  3.62814891e-01, -5.74902912e-01,
         1.78198446e-01, -1.53294940e-01,  3.04903382e-01,
         7.54582405e-01,  6.89035838e-01],
       [ 9.24215076e-01, -1.07013092e+00, -4.14588665e-01,
        -8.72111763e-01, -1.58377000e-01,  7.68253259e-01,
        -3.75648829e-01,  2.50813197e-01, -2.37123323e-01,
        -3.39074057e-01,  4.77602939e-01, -5.88833065e-01,
         1.25872882e-01, -8.03707589e-01, -3.22799052e-02,
         5.72959249e-01, -9.19297248e-01,  1.05601266e-01,
        -1.04018172e+00,  3.36215166e-01, -1.62085409e-01,
        -6.23053169e-01, -4.80464016e-02, -3.79107220e-02,
         8.18799631e-02, -8.18906932e-01, -1.71647766e-02,
         1.75706061e-01,  4.98588521e-01, -1.93542651e-01,
         3.55106437e-01,  9.51537160e-01, -6.91495536e-02,
        -9.01817655e-01,  6.38933006e-02, -1.21134409e+00,
        -8.86091607e-02,  8.52731865e-01, -1.07127372e+00,
         2.40365435e-01,  1.02696270e-01,  6.71882412e-01,
        -7.06579620e-01,  2.88326024e-01, -2.73977678e-01,
        -1.88249029e-01, -1.77656904e-01,  1.24518452e+00,
         2.54333226e-01, -8.53716934e-03],
       [ 6.64301028e-01, -4.47177585e-02, -6.88916075e-02,
        -1.04699501e-01,  7.03232001e-02,  7.31827006e-01,
         9.92483463e-02,  3.58465056e-01, -3.56859101e-01,
        -7.58568088e-01, -1.39189371e-01, -2.88705941e-01,
        -2.66434426e-01, -3.90556309e-01,  3.84824613e-01,
        -4.79180836e-01, -5.53124383e-01,  8.60149969e-01,
        -5.45880810e-01,  6.98425615e-01,  7.95584684e-01,
        -2.18777336e-01, -3.80458520e-01, -2.08927222e-01,
         4.41321780e-01, -1.77957648e-01,  1.75392600e-01,
         4.45130242e-01, -1.00340107e-01, -7.88752965e-01,
        -1.61759298e-01,  5.31399607e-01,  3.83461213e-01,
        -3.62583796e-01,  1.16239700e+00, -7.70469325e-02,
         8.20793972e-02,  6.29332309e-01, -1.08026937e-02,
         1.91396369e-01,  4.62263626e-01,  1.15230769e-01,
         3.65408220e-02,  1.02490319e-01,  8.99998727e-02,
        -3.31468327e-01, -2.01140205e-01,  8.89639602e-01,
        -1.75487674e-01,  7.67609155e-02]])
```

Y estas serían las predicciones individuales para cada miembro del grupo

In [50]:
```python
group_individual_recommendations = group_latent_factors.dot(v.T)
group_individual_recommendations
```

Out[50]:
```
array([[5.10239423, 2.22071579, 2.79964582, ..., 1.77469039, 1.53806327,
        2.44905738],
       [4.39270945, 3.27591223, 2.65964213, ..., 2.26642673, 2.44483001,
        2.83610671],
       [4.97569951, 2.57248542, 3.1092161 , ..., 1.51902206, 2.44509515,
        4.37492714],
       [4.50735652, 5.67135937, 2.87275592, ..., 1.00665556, 1.77858396,
        2.81478851],
       [3.49117767, 2.72865084, 2.80715308, ..., 1.47871212, 1.46058451,
        2.75185813]])
```

Si ordenamos las predicciones por usuario, en base a los valores que se acaban de generar, podemos ver que cada usuario tiene diferentes preferencias. Ya que en la columna 0, que contiene la mejor película para cada usuario, todos presentan diferentes películas y lo mismo sucede para el resto de columnas

In [51]:
```python
movies_id_for_each_user_order_by_likehood = np.argsort(-group_individual_recommendations)
movies_id_for_each_user_order_by_likehood
```

Out[51]:
```
array([[2027, 1916, 1230, ..., 1077, 7744, 8875],
       [ 520, 3012,  618, ..., 7990, 8875, 8399],
       [ 485, 3563,  992, ..., 8875, 8913, 2034],
       [  43, 1066, 1795, ..., 7458, 9022, 8694],
       [1796, 3191, 7396, ..., 5649, 5200, 8875]])
```

Ahora que tenemos las recomendaciones individuales para cada uno de los elementos del grupo, procederemos a utilizar diferentes técnicas de agregación de las recomendaciones para la generación de las recomnedaciones finales para el grupo

Media de las recomendaciones individuales

La primera agregación básica que podemos considerar de cara a generar las recomendaciones para el grupo, es el uso de la media de las valoraciones predichas de cada usuario para cada película y acto seguido recomendar las películas que presenten un valor más alto para la media.

In [52]:
```
mean_recommendations = np.mean(group_individual_recommendations, axis=0)
mean_recommendations_indexs = np.argsort(-mean_recommendations)
mean_recommendations_indexs
```

Out[52]:
```
array([ 602, 2077,  982, ..., 8399, 8694, 8875])
```

Recupero el título de las películas que forman parte de las mejores 30 recomendaciones

In [53]:
```
df_movies.loc[mean_recommendations_indexs].title[0:30]
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike
  """Entry point for launching an IPython kernel.
```

Out[53]:
```
602     Dr. Strangelove or: How I Learned to Stop Worr...
2077                                   Iron Giant, The
982                                          High Noon
1066                                       Under Siege
520                                              Fargo
277                          Shawshank Redemption, The
1043                                        Nightwatch
2592                                               Hud
899                                Princess Bride, The
27                                          Persuasion
898        Star Wars: Episode V - The Empire Strikes Back
2979                                    102 Dalmatians
224                     Star Wars: Episode IV - A New Hope
964                                      Groundhog Day
485                                          Tombstone
901                                             Brazil
510                          Silence of the Lambs, The
974                                         Highlander
4131                                  Maid in Manhattan
46                                 Usual Suspects, The
913                                     Third Man, The
1211                         Hunt for Red October, The
905                                       12 Angry Men
1544                                 Lady and the Tramp
4755                                       42nd Street
0                                            Toy Story
1945                                         Following
4791                                       Cooler, The
818                                            Bananas
914                                         Goodfellas
Name: title, dtype: object
```

El problema de esta mecanismo de agregación es que si un usuario tiene gustos muy diferentes comparados con el resto del grupo, sus preferencias quedaran ignoradas con respecto al resto del grupo, lo que poderíamos llegar a considerar como una mala recomendación según el escenario

Multiplicación de las recomendaciones individuales

Otra medida de agregación similar a la media y con un comportamiento similar en este caso, que altera ligeramente las recomendaciones, es la agregación de las recomendaciones indivuduales de cada película mediante la multiplicación de la valoraciones individuales de cada usuario para cada película

In [54]:
```
multiply_recommendations = np.prod(group_individual_recommendations, axis=0)
multiply_recommendations_indexs = np.argsort(-multiply_recommendations)
multiply_recommendations_indexs
```

Out[54]:
```
array([ 982,  602, 2077, ..., 1144,  145, 2034])
```

Recupero el título de las películas que forman parte de las mejores 30 recomendaciones

In [55]:
```
df_movies.loc[multiply_recommendations_indexs].title[0:30]
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike
  """Entry point for launching an IPython kernel.
```

Out[55]:
```
982                                          High Noon
602     Dr. Strangelove or: How I Learned to Stop Worr...
2077                                   Iron Giant, The
1066                                       Under Siege
277                          Shawshank Redemption, The
1043                                        Nightwatch
2592                                               Hud
27                                          Persuasion
898        Star Wars: Episode V - The Empire Strikes Back
899                                  Princess Bride, The
520                                              Fargo
224                  Star Wars: Episode IV - A New Hope
964                                      Groundhog Day
510                          Silence of the Lambs, The
974                                         Highlander
913                                    Third Man, The
46                                 Usual Suspects, The
1211                         Hunt for Red October, The
905                                       12 Angry Men
4131                                 Maid in Manhattan
901                                             Brazil
4755                                       42nd Street
0                                            Toy Story
2979                                    102 Dalmatians
7396              I Killed My Mother (J'ai tué ma mère)
1945                                         Following
123                                          Apollo 13
835                                     Sophie's Choice
107                Chungking Express (Chung Hing sam lam)
7127                             Hunt For Gollum, The
Name: title, dtype: object
```

Al igual que en el caso anterior, el problema de esta técnica de agregación vuelven a ser los usuarios que presentan gustos diferentes a los principales del grupo. A continuación se muestran dos ejemplos de posibles agregaciones, para un grupo formado por 3 usuarios, y no queda claro si realmente sería mejor recomendar la primera película en vez de la segunda

In [83]:
```
1*5*5
```

Out[83]: 25

In [86]:
```
2*3*4
```

Out[86]: 24

Borda Count

Este método de agregacón consiste en asignar puntos a las películas en función de en que posición aparecentro dentro del ranking individual de recomendaciones de cada usuario, recibiendo la primera película un número de puntos igual al número de peliculas en el ranking y recibiendo la última película del ranking 0 puntos. Finalmente se suman los puntos obtenidos por cada película y se ordenan las películas en base a estos

In [56]:
```
movies_id_for_each_user_order_by_likehood
```

Out[56]:
```
array([[2027, 1916, 1230, ..., 1077, 7744, 8875],
       [ 520, 3012,  618, ..., 7990, 8875, 8399],
       [ 485, 3563,  992, ..., 8875, 8913, 2034],
       [  43, 1066, 1795, ..., 7458, 9022, 8694],
       [1796, 3191, 7396, ..., 5649, 5200, 8875]])
```

In [57]:
```
number_of_movies = df_movies.shape[0]
borda_rating = np.arange(1, number_of_movies+1)
borda_rating
```

Out[57]:
```
array([   1,    2,    3, ..., 9727, 9728, 9729])
```

In [58]:
```
with np.nditer(borda_rating, op_flags=['readwrite']) as it:
  for x in it:
    positions_inside_individual_recommendations = np.where(movies_id_for_each_user_order_by_likehood == x)[1]
    value_by_position_inside_each_ranking = number_of_movies - positions_inside_individual_recommendations
    x[...] = value_by_position_inside_each_ranking.sum()

borda_rating
```

Out[58]:
```
array([34111, 32883, 22427, ...,     0,     0,     0])
```

Recupero el título de las películas que forman parte de las mejores 30 recomendaciones

In [59]:
```
borda_recommendations_indexs = np.argsort(-borda_rating)
df_movies.loc[borda_recommendations_indexs].title[0:30]
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike
```

Out[59]:
```
981                                          Fantasia
2076                                             Dick
1042                                Breaking the Waves
1210                             187 (One Eight Seven)
897                        Cheech and Chong's Up in Smoke
601                                     Arrival, The
276                                Santa Clause, The
7126                         Men Who Stare at Goats, The
963                                              Diva
4130                                    Hot Chick, The
4754                         Hunchback of Notre Dame, The
795                                 Secret Agent, The
6630                                      I Am Legend
223                                     Kiss of Death
2591                                  Heart and Souls
106                                         Boomerang
996                              Pink Floyd: The Wall
3632       Spacehunter: Adventures in the Forbidden Zone
1420                                 All the King's Men
509                                            Batman
912            Wings of Desire (Himmel über Berlin, Der)
2450                             White Men Can't Jump
1944                                         Metroland
26                                       Now and Then
8433                                       Maleficent
9299                                        All Yours
834                               Glengarry Glen Ross
6158                                      Leprechaun 2
898      Star Wars: Episode V - The Empire Strikes Back
1659                                       Ring, The
Name: title, dtype: object
```

Copeland Rule

Se calcula la valoración media de cada película y se ordenan estas en un raking, una vez ordenadas, se genera una nueva valoración para cada película en base al número de peliculas que se encuentran por debajo de cada película en el ranking, menos el número de películas que se encuentran por encima de cada película en el ranking

In [60]:
```
add_recommendations = np.mean(group_individual_recommendations, axis=0)
index_sorted_elements = np.argsort(-add_recommendations)
index_sorted_elements
```

Out[60]:
```
array([ 602, 2077,  982, ..., 8399, 8694, 8875])
```

In [61]:
```
copeland_values = np.arange(0, index_sorted_elements.shape[0])
copeland_values
```

Out[61]:
```
array([   0,    1,    2, ..., 9721, 9722, 9723])
```

In [62]:
```
with np.nditer(copeland_values, op_flags=['readwrite']) as it:
  for x in it:
    positive = np.where(index_sorted_elements == x)[0][0]
    x[...] = positive - (number_of_movies - positive)

copeland_values
```

Out[62]:
```
array([-9679, -6307, -3091, ...,  5443,  3421, -4581])
```

Recupero el título de las películas que forman parte de las mejores 30 recomendaciones

In [63]:
```
copeland_recommendations_indexs = np.argsort(-copeland_values)
df_movies.loc[borda_recommendations_indexs].title[0:30]
```


```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike
```


Out[63]:

```
981                                          Fantasia
2076                                             Dick
1042                                Breaking the Waves
1210                             187 (One Eight Seven)
897                        Cheech and Chong's Up in Smoke
601                                     Arrival, The
276                                Santa Clause, The
7126                         Men Who Stare at Goats, The
963                                              Diva
4130                                    Hot Chick, The
4754                         Hunchback of Notre Dame, The
795                                 Secret Agent, The
6630                                      I Am Legend
223                                     Kiss of Death
2591                                  Heart and Souls
106                                         Boomerang
996                              Pink Floyd: The Wall
3632       Spacehunter: Adventures in the Forbidden Zone
1420                                 All the King's Men
509                                            Batman
912            Wings of Desire (Himmel über Berlin, Der)
2450                             White Men Can't Jump
1944                                         Metroland
26                                       Now and Then
8433                                       Maleficent
9299                                        All Yours
834                               Glengarry Glen Ross
6158                                      Leprechaun 2
898      Star Wars: Episode V - The Empire Strikes Back
1659                                       Ring, The
Name: title, dtype: object
```

Least Misery

Genera un raking de películas, tomando para cada película la peor valoración realizada por un miembro del grupo

In [64]:
```
least_misery_recommendations = np.amin(group_individual_recommendations, axis=0)
least_misery_indexs = np.argsort(-least_misery_recommendations)
least_misery_indexs
```

Out[64]:
```
array([ 898,  982, 1066, ..., 8875, 2034, 8399])
```

Recupero el título de las películas que forman parte de las mejores 30 recomendaciones

In [65]:
```
df_movies.loc[least_misery_indexs].title[0:30]
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike
  """Entry point for launching an IPython kernel.
```

Out[65]:
```
898       Star Wars: Episode V - The Empire Strikes Back
982                                            High Noon
1066                                         Under Siege
4755                                         42nd Street
27                                            Persuasion
680                               Philadelphia Story, The
277                              Shawshank Redemption, The
74                               Antonia's Line (Antonia)
107                   Chungking Express (Chung Hing sam lam)
974                                           Highlander
7127                                    Hunt For Gollum, The
965                                           Unforgiven
9300                                        Kill Command
4931         Scenes From a Marriage (Scener ur ett äktenskap)
2077                                       Iron Giant, The
2590                                         Modern Times
913                                        Third Man, The
6810                         Heart of a Dog (Sobachye serdtse)
4134                                              Evelyn
147                                                 Kids
123                                            Apollo 13
2329                                      Babes in Toyland
835                                       Sophie's Choice
46                                     Usual Suspects, The
1660                     Lodger: A Story of the London Fog, The
940                                  Dead Alive (Braindead)
921                                     Blues Brothers, The
7752                                              Lifted
905                                         12 Angry Men
1294                                    Horse Whisperer, The
Name: title, dtype: object
```

Most Pleasure

Genera un raking de películas, tomando para cada película la mejor valoración realizada por un miembro del grupo ure

In [66]:
```
most_pleasure_recommendations = np.amax(group_individual_recommendations, axis=0)
most_pleasure_indexs = np.argsort(-most_pleasure_recommendations)
most_pleasure_indexs
```

Out[66]:
```
array([ 520,  485, 3563, ..., 4656, 7280, 8875])
```

Recupero el título de las películas que forman parte de las mejores 30 recomendaciones

In [67]:
```
df_movies.loc[most_pleasure_indexs].title[0:30]
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike
  """Entry point for launching an IPython kernel.
```

Out[67]:
```
520                                        Fargo
485                                    Tombstone
3563                        High Heels and Low Lifes
3012                                 Dracula 2000
618                         Hunchback of Notre Dame, The
2978                          Rugrats in Paris: The Movie
906                                   Lawrence of Arabia
992                                          Gandhi
15                                           Casino
311                                       Crow, The
1231                                    Chasing Amy
1544                                 Lady and the Tramp
1294                               Horse Whisperer, The
2729                  Puppet Master 5: The Final Chapter
2903                                    Nurse Betty
434                               Much Ado About Nothing
7987                                          V/H/S
6520                                   Evan Almighty
43                                 Seven (a.k.a. Se7en)
5938                                 Wedding Crashers
2077                                  Iron Giant, The
8272        Blue Is the Warmest Color (La vie d'Adèle)
2979                                  102 Dalmatians
8599                              Penguins of Madagascar
1066                                    Under Siege
6579                                  Good Luck Chuck
55                                  Mr. Holland's Opus
964                                   Groundhog Day
8358                                        RoboCop
7626                                         Bernie
Name: title, dtype: object
```

Average without Misery

Consiste en calcular la media de las recomendaciones individuales predichas para cada usuario, ignorando todas aquellas valoraciones inferiores a un umbral que seleccionemos previamente

In [68]:
```
average_without_misery_values = np.arange(0, index_sorted_elements.shape[0])

threshold = 3
columns = group_individual_recommendations.T
idx = 0
for column in columns:
  valid = column > threshold
  filtered = column[valid]
  value = np.mean(filtered)
  if (np.isnan(value)):
    value = 0
  average_without_misery_values[idx] = value
  idx += 1

average_without_misery_indexs = np.argsort(-average_without_misery_values)
average_without_misery_indexs
```

```
/usr/local/lib/python3.6/dist-packages/numpy/core/fromnumeric.py:3118: RuntimeWarning: Mean of empty slice.
  out=out, **kwargs)
/usr/local/lib/python3.6/dist-packages/numpy/core/_methods.py:85: RuntimeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```

Out[68]:
```
array([6522, 1548, 2979, ..., 5850,  480, 5243])
```

Recupero el título de las películas que forman parte de las mejores 30 recomendaciones

In [69]:
```
df_movies.loc[average_without_misery_indexs].title[0:30]
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike
  """Entry point for launching an IPython kernel.
```

Out[69]:
```
6522    Harry Potter and the Order of the Phoenix
1548                                       Newsies
2979                                101 Dalmatians
2034                             Muppets From Space
8532                                   Captive, The
2044                                    Mystery Men
1397              Buffalo '66 (a.k.a. Buffalo 66)
485                                      Tombstone
618                     Hunchback of Notre Dame, The
2823                      The Golden Voyage of Sinbad
3081                              Monster Squad, The
520                                          Fargo
4527                        Once Upon a Time in Mexico
8161                                        Darkon
2603                                     Ladyhawke
1230                                  Ice Storm, The
992                                         Gandhi
7594                                 Mildred Pierce
2476                                    Hanging Up
2341                                       Shampoo
2377                              Anna and the King
2740                       Man with the Golden Arm, The
3979                                       Trapped
5163                                    Soul Plane
15                                          Casino
6632                       Futurama: Bender's Big Score
6640                             My Blueberry Nights
6608                              King of California
1882                                    October Sky
2274                                    Creepshow 2
Name: title, dtype: object
```

Agregación seleccionada

Tras probar las diferentes técnicas para agregar las recomendaciones individuales, resulta complicado decantarse por una en concreto sin disponer de mayor información del contexto en el que se está generando la recomendación. Ya que existen incógnitas de las que no disponemos ningún tipo de recomendación, como pueden ser:

- existen usuarios que puedan influenciar al resto del grupo en su valoración?
- existen usuarios que estean dispuestos a ignorar sus preferencias por el "bien" del grupo?
- que tipo de recomendación consideramos "mejores" para un grupo, en el caso de que sus usuarios presenten opiniones muy diferentes?

Ante incógnitdas como las planteadas, la técnica de agregación **Least Misery**, parece la mejor opción a la hora de mantener contentos a todos los usuarios del grupo con la generación recomendada, pero al tratarse de una técnica que se basa en la peores de las valoraciones individuales, será complicado generar recomendaciones que resulten sorprendentes al grupo de usuarios

Recursos consultados

1. Lecture 47 — Singular Value Decomposition | Stanford University (https://www.youtube.com/watch?v=P5mlg91as1c)
2. Understanding matrix factorization for recommendation (part 4) - algorithm implementation (http://nicolas-hug.com/blog/matrix_facto_4)
3. Takács, G., Pilászy, I., Németh, B., & Tikk, D. (2014). Matrix factorization and neighbor based algorithms for the netflix prize problem Sugeno-Yasukawa qualitative modeling View project Matrix Factorization and Neighbor Based Algorithms for the Netflix Prize Problem General Terms. http://doi.org/10.1145/1454008.1454049 (http://doi.org/10.1145/1454008.1454049)
4. J. Masthoff. Group recommender systems: combining individual models. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, Recommender Systems Handbook, page 677. Springer US, Boston, MA, 2011.