# 0. Configuração e preparação dos dados

### Configuração padrão do Notebook

In [1]:

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import seaborn as sns
import time, sys
from IPython.display import clear_output
plt.rcParams['image.cmap'] = 'Greys'
#cmap = ['Greys', 'Purples', 'Blues', 'Greens', 'Oranges', 'Reds', 'YlOrBr', 'YlOrRd', 'OrRd',
#        'PuRd', 'RdPu','BuPu','GnBu', 'PuBu', 'YlGnBu', 'PuBuGn', 'BuGn', 'YlGn']
plt.rcParams['figure.figsize'] = [4, 3]
plt.rcParams['figure.dpi'] = 140

def update_progress(progress):
    bar_length = 20
    if isinstance(progress, int):
        progress = float(progress)
    if not isinstance(progress, float):
        progress = 0
    if progress < 0:
        progress = 0
    if progress >= 1:
        progress = 1

    block = int(round(bar_length * progress))

    clear_output(wait = True)
    text = "Progress: [{0}] {1:.1f}%".format( "#" * block + "-" * (bar_length - block), progress * 100)
    print(text)
```

### Lendo arquivo 'movies.csv'

In [2]:

```
filmes = pd.read_csv('movies.csv')
filmes.head()
```

Out[2]:

| | 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 |

### Alterando o nome das colunas para facilitar a compreensão

In [3]:

```
filmes.columns = ['filmeId', 'titulo', 'generos']
filmes = filmes.set_index('filmeId')
filmes.head()
```

Out[3]:

| filmeId | titulo | generos |
|---|---|---|
| 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 |
| 5 | Father of the Bride Part II (1995) | Comedy |

### Lendo arquivo 'ratings.csv'

In [4]:

```
notas = pd.read_csv('ratings.csv')
notas.head()
```

Out[4]:

| | userId | movieId | rating | timestamp |
|---|---|---|---|---|
| 0 | 1 | 307 | 3.5 | 1256677221 |
| 1 | 1 | 481 | 3.5 | 1256677456 |
| 2 | 1 | 1091 | 1.5 | 1256677471 |
| 3 | 1 | 1257 | 4.5 | 1256677460 |
| 4 | 1 | 1449 | 4.5 | 1256677264 |

### Alterando o nome das colunas para facilitar a compreensão

In [5]:

```
notas.columns = ['usuarioId', 'filmeId', 'nota', 'momento']
notas.head()
```

Out[5]:

| | usuarioId | filmeId | nota | momento |
|---|---|---|---|---|
| 0 | 1 | 307 | 3.5 | 1256677221 |
| 1 | 1 | 481 | 3.5 | 1256677456 |
| 2 | 1 | 1091 | 1.5 | 1256677471 |
| 3 | 1 | 1257 | 4.5 | 1256677460 |
| 4 | 1 | 1449 | 4.5 | 1256677264 |

### Algumas estatísticas sobre as notas

In [6]:

```
notas.describe().round(3)
```

Out[6]:

| | usuarioId | filmeId | nota | momento |
|---|---|---|---|---|
| count | 2.775344e+07 | 2.775344e+07 | 2.775344e+07 | 2.775344e+07 |
| mean | 1.419420e+05 | 1.848800e+04 | 3.530000e+00 | 1.193122e+09 |
| std | 8.170740e+04 | 3.510262e+04 | 1.066000e+00 | 2.160482e+08 |
| min | 1.000000e+00 | 1.000000e+00 | 5.000000e-01 | 7.896520e+08 |
| 25% | 7.117600e+04 | 1.097000e+03 | 3.000000e+00 | 9.986053e+08 |
| 50% | 1.420220e+05 | 2.716000e+03 | 3.500000e+00 | 1.174256e+09 |
| 75% | 2.124590e+05 | 7.150000e+03 | 4.000000e+00 | 1.422744e+09 |
| max | 2.832280e+05 | 1.938860e+05 | 5.000000e+00 | 1.537945e+09 |

### Quantidade de filmes

In [7]:

```
filmes.shape[0]
```

Out[7]:

```
58098
```

### Quantidade de avaliações

In [8]:

```
notas.shape[0]
```

Out[8]:

```
27753444
```

# 1. Heurística baseada no total de votos

### Essa forma de recomendação é usada quando não se conhece nenhuma informação acerca do usuário, desta forma, o sistema de recomendação seleciona aqueles com maior número de avaliações

### Total de votos recebido por filme

In [9]:

```
total_de_votos = pd.DataFrame(notas["filmeId"].value_counts())
total_de_votos.columns = ['total_de_votos']
total_de_votos.head()
```

Out[9]:

| | total_de_votos |
|---|---|
| **318** | 97999 |
| **356** | 97040 |
| **296** | 92406 |
| **593** | 87899 |
| **2571** | 84545 |

### Adicionando a coluna total_de_votos ao dataFrame de filmes

In [10]:

```
filmes['total_de_votos'] = total_de_votos
filmes.head()
```

Out[10]:

| | titulo | generos | total_de_votos |
|---|---|---|---|
| **filmeId** | | | |
| **1** | Toy Story (1995) | Adventure\|Animation\|Children\|Comedy\|Fantasy | 68469.0 |
| **2** | Jumanji (1995) | Adventure\|Children\|Fantasy | 27143.0 |
| **3** | Grumpier Old Men (1995) | Comedy\|Romance | 15585.0 |
| **4** | Waiting to Exhale (1995) | Comedy\|Drama\|Romance | 2989.0 |
| **5** | Father of the Bride Part II (1995) | Comedy | 15474.0 |

### Estes seriam os filmes indicados pela primeira heurística:

In [11]:

```
filmes_indicados = filmes.sort_values("total_de_votos", ascending = False)
genero_alvo = ''
filmes_indicados[filmes_indicados['generos'].str.contains(genero_alvo)][['titulo']].head(10)
```

Out[11]:

| filmeId | titulo |
|---|---|
| 318 | Shawshank Redemption, The (1994) |
| 356 | Forrest Gump (1994) |
| 296 | Pulp Fiction (1994) |
| 593 | Silence of the Lambs, The (1991) |
| 2571 | Matrix, The (1999) |
| 260 | Star Wars: Episode IV - A New Hope (1977) |
| 480 | Jurassic Park (1993) |
| 527 | Schindler's List (1993) |
| 110 | Braveheart (1995) |
| 1 | Toy Story (1995) |

**Ao mudar a variavel genero_alvo, podemos descobrir a recomendação para alguma categoria específica, estes seriam os filmes indicados pela primeira heurística para a categoria infantil:**

In [12]:

```
genero_alvo = 'Children'
filmes_indicados[filmes_indicados['generos'].str.contains(genero_alvo)][['titulo']].head(10)
```

Out[12]:

| filmeId | titulo |
|---|---|
| 1 | Toy Story (1995) |
| 588 | Aladdin (1992) |
| 364 | Lion King, The (1994) |
| 4306 | Shrek (2001) |
| 595 | Beauty and the Beast (1991) |
| 1097 | E.T. the Extra-Terrestrial (1982) |
| 4886 | Monsters, Inc. (2001) |
| 6377 | Finding Nemo (2003) |
| 34 | Babe (1995) |
| 1073 | Willy Wonka & the Chocolate Factory (1971) |

# 2. Heurística baseada na nota média e na filtragem de votos

**Ainda considerando que nada se sabe sobre o usuário, temos outra forma de recomendação. Na primeira heurística, não é levado em conta a nota média dos filmes, sendo assim, levaremos em conta esta nota, e, definiremos um número mínimo de notas que um filme deve ter para ser recomendável.**

### Média da nota dos votos recebido por filme

In [13]:

```
notas_medias = pd.DataFrame(notas.groupby("filmeId").mean()["nota"])
notas_medias.columns = ['nota_media']
notas_medias.head()
```

Out[13]:

| filmeId | nota_media |
|---|---|
| 1 | 3.886649 |
| 2 | 3.246583 |
| 3 | 3.173981 |
| 4 | 2.874540 |
| 5 | 3.077291 |

### Adicionando a coluna nota_media ao dataFrame de filmes

In [14]:

```
filmes['nota_media'] = notas_medias
filmes.head()
```

Out[14]:

| filmeId | titulo | generos | total_de_votos | nota_media |
|---|---|---|---|---|
| 1 | Toy Story (1995) | Adventure\|Animation\|Children\|Comedy\|Fantasy | 68469.0 | 3.886649 |
| 2 | Jumanji (1995) | Adventure\|Children\|Fantasy | 27143.0 | 3.246583 |
| 3 | Grumpier Old Men (1995) | Comedy\|Romance | 15585.0 | 3.173981 |
| 4 | Waiting to Exhale (1995) | Comedy\|Drama\|Romance | 2989.0 | 2.874540 |
| 5 | Father of the Bride Part II (1995) | Comedy | 15474.0 | 3.077291 |

### A filtragem é necessária, caso contrário, os filmes com nota média mais alta seriam filmes de nicho com pouquíssimas avalições

In [15]:

```
filmes.sort_values("nota_media", ascending = False).head(10)
```

Out[15]:

| filmeId | titulo | generos | total_de_votos | nota_media |
|---|---|---|---|---|
| 169338 | Brad Williams: Daddy Issues (2016) | Comedy | 2.0 | 5.0 |
| 187729 | Ab-normal Beauty (2004) | Horror | 1.0 | 5.0 |
| 172149 | Back to You and Me (2005) | Drama\|Romance | 1.0 | 5.0 |
| 160966 | You're Human Like the Rest of Them (1967) | (no genres listed) | 1.0 | 5.0 |
| 134387 | At Ellen's Age (2011) | Comedy\|Drama | 1.0 | 5.0 |
| 98437 | Bed of Roses (1933) | Comedy\|Drama\|Romance | 1.0 | 5.0 |
| 134433 | Le nuove comiche (1994) | (no genres listed) | 1.0 | 5.0 |
| 172151 | The Shocking Miss Pilgrim (1947) | Comedy\|Romance | 1.0 | 5.0 |
| 134605 | Men Don't Cry (1968) | Comedy | 1.0 | 5.0 |
| 134633 | Tony 10 (2012) | Children | 1.0 | 5.0 |

### Estes seriam os filmes indicados pela segunda heurística:

In [16]:

```
filmes_indicados = filmes.query("total_de_votos >= 100").sort_values("nota_media", ascending = False)
genero_alvo = ''
filmes_indicados[filmes_indicados['generos'].str.contains(genero_alvo)][['titulo']].head(10)
```

Out[16]:

| filmeId | titulo |
|---|---|
| 171011 | Planet Earth II (2016) |
| 159817 | Planet Earth (2006) |
| 318 | Shawshank Redemption, The (1994) |
| 170705 | Band of Brothers (2001) |
| 174053 | Black Mirror: White Christmas (2014) |
| 171495 | Cosmos |
| 172591 | The Godfather Trilogy: 1972-1990 (1992) |
| 858 | Godfather, The (1972) |
| 50 | Usual Suspects, The (1995) |
| 176601 | Black Mirror |

**Ao mudar a variavel genero_alvo, podemos descobrir a recomendação para alguma categoria específica, estes seriam os filmes indicados pela segunda heurística para a categoria infantil:**

In [17]:

```
genero_alvo = 'Children'
filmes_indicados[filmes_indicados['generos'].str.contains(genero_alvo)][['titulo']].head(10)
```

Out[17]:

| filmeId | titulo |
|---|---|
| 172577 | Last Year's Snow Was Falling (1983) |
| 170777 | There Once Was a Dog (1982) |
| 5971 | My Neighbor Totoro (Tonari no Totoro) (1988) |
| 1148 | Wallace & Gromit: The Wrong Trousers (1993) |
| 745 | Wallace & Gromit: A Close Shave (1995) |
| 6350 | Laputa: Castle in the Sky (Tenkû no shiro Rapy... |
| 177765 | Coco (2017) |
| 953 | It's a Wonderful Life (1946) |
| 163072 | Winnie Pooh (1969) |
| 60069 | WALL·E (2008) |

# 3. Heurística baseada na fórmula de rankeamento do IMDB

**Esta fórmula corresponde à que é utilizada pelo IMDB. Partindo do mesmo princípio de que o novo usuário ainda não assistiu nenhum filme, os filmes recomendados seriam os mais bem ranqueados de acordo com a fórmula**

**IMDB = (((v / (v + m)) *M)* + *((m / (v + M))* mM))**

- v é o numero de votos recebidos pelo filme.
- m é o número mínimo de votos necessários para o filme ser recomendável.
- M é a nota média do filme.
- mM é a média das notas médias de todos os filmes.

**O número mínimo de votos necessários para o filme ser recomendável é:**

In [18]:

```
m = filmes['total_de_votos'].quantile(.95)
m
```

Out[18]:

```
1855.0
```

### A média das notas médias de todos os filmes é:

In [19]:

```
mM = filmes['nota_media'].mean()
mM
```

Out[19]:

```
3.0685927253973193
```

### Função que calcula a pontuação IMDB

In [20]:

```
def IMDB(filme):
    v = filme['total_de_votos']
    M = filme['nota_media']
    if(v >= m):
        return (v/(v+m) * M) + (m/(m+v) * mM)
    else:
        return 0
```

### Estes seriam os filmes indicados pela terceira heurística:

In [21]:

```
filmes['pontuacao_IMDB'] = filmes.apply(IMDB, axis=1)
filmes_indicados = filmes.sort_values('pontuacao_IMDB', ascending = False)
genero_alvo = ''
filmes_indicados[filmes_indicados['generos'].str.contains(genero_alvo)][['titulo']].head(10)
```

Out[21]:

| filmeId | titulo |
|---|---|
| 318 | Shawshank Redemption, The (1994) |
| 858 | Godfather, The (1972) |
| 50 | Usual Suspects, The (1995) |
| 527 | Schindler's List (1993) |
| 1221 | Godfather: Part II, The (1974) |
| 2959 | Fight Club (1999) |
| 1193 | One Flew Over the Cuckoo's Nest (1975) |
| 296 | Pulp Fiction (1994) |
| 912 | Casablanca (1942) |
| 904 | Rear Window (1954) |

### Ao mudar a variavel genero_alvo, podemos descobrir a recomendação para alguma categoria específica, estes seriam os filmes indicados pela terceira heurística para a categoria infantil:

In [22]:

```
genero_alvo = 'Children'
filmes_indicados[filmes_indicados['generos'].str.contains(genero_alvo)][['titulo']].head(10)
```

Out[22]:

| filmeId | titulo |
|---|---|
| 1148 | Wallace & Gromit: The Wrong Trousers (1993) |
| 745 | Wallace & Gromit: A Close Shave (1995) |
| 5971 | My Neighbor Totoro (Tonari no Totoro) (1988) |
| 60069 | WALL·E (2008) |
| 953 | It's a Wonderful Life (1946) |
| 68954 | Up (2009) |
| 919 | Wizard of Oz, The (1939) |
| 2804 | Christmas Story, A (1983) |
| 1 | Toy Story (1995) |
| 134853 | Inside Out (2015) |

# 4. Heurística baseada na correlação de Pearson

**Partindo de um filme assistido, a heurística pode recomendar filmes usando como ponto de partida, o coeficiente de correlação de Pearson. O coeficiente indica uma correlação positiva, negativa ou nula entre os elementos.**

### Valores de referência para p:

- p = 0.9 para mais ou para menos indica uma correlação muito forte.
- p = 0.7 a 0.9 positivo ou negativo indica uma correlação forte.
- p = 0.5 a 0.7 positivo ou negativo indica uma correlação moderada.
- p = 0.3 a 0.5 positivo ou negativo indica uma correlação fraca.
- p = 0 a 0.3 positivo ou negativo indica uma correlação desprezível.

### Tabela única, que será usada para pivotar os dados

In [23]:

```
tabela_pearson = pd.merge(notas, filmes, on='filmeId')
tabela_pearson.head()
```

Out[23]:

| | usuarioId | filmeId | nota | momento | titulo | generos | total_de_votos | nota_media | pontuacao_IMDB |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 307 | 3.5 | 1256677221 | Three Colors: Blue (Trois couleurs: Bleu) (1993) | Drama | 7958.0 | 3.971727 | 3.801003 |
| 1 | 6 | 307 | 4.0 | 832059248 | Three Colors: Blue (Trois couleurs: Bleu) (1993) | Drama | 7958.0 | 3.971727 | 3.801003 |
| 2 | 56 | 307 | 4.0 | 1383625728 | Three Colors: Blue (Trois couleurs: Bleu) (1993) | Drama | 7958.0 | 3.971727 | 3.801003 |
| 3 | 71 | 307 | 5.0 | 1257795414 | Three Colors: Blue (Trois couleurs: Bleu) (1993) | Drama | 7958.0 | 3.971727 | 3.801003 |
| 4 | 84 | 307 | 3.0 | 999055519 | Three Colors: Blue (Trois couleurs: Bleu) (1993) | Drama | 7958.0 | 3.971727 | 3.801003 |

### Utilizaremos apenas as 10000000 primeiras avaliações para evitar um overflow no sistema

In [24]:

```
tabela_pearson = tabela_pearson.iloc[:10000000,:]
```

### Pivotagem dos dados, onde será aplicada a função de correlação

In [25]:

```
tabela_pearson = tabela_pearson.pivot_table(index='usuarioId', columns='titulo', values='nota')
tabela_pearson.head()
```

Out[25]:

| titulo | 12 Angry Men (1957) | 15 Minutes (2001) | 2001: A Space Odyssey (1968) | 2010: The Year We Make Contact (1984) | 48 Hrs. (1982) | 8MM (1999) | Abyss, The (1989) | Ace Ventura: Pet Detective (1994) | Ace Ventura: When Nature Calls (1995) | Addams Family Values (1993) | ... | Wing Commander (1999) | Wizard of Oz, The (1939) | Wolf (1994) | World Is Not Enough, The (1999) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **usuarioId** | | | | | | | | | | | | | | | |
| **1** | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN |
| **2** | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN |
| **3** | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN |
| **4** | 1.5 | 3.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.5 | 3.0 | 2.0 | 1.0 | ... | 3.0 | 4.0 | 3.0 | 4.0 |
| **5** | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN |

5 rows × 612 columns

### Função de cálculo do coeficiente de correlação de Pearson

In [26]:

```
def coeficiente_de_correlação_de_Pearson(vetor_1, vetor_2):
    covariancia_1 = vetor_1 - vetor_1.mean()
    covariancia_2 = vetor_2 - vetor_2.mean()
    p = np.sum(covariancia_1 * covariancia_2) / np.sqrt(np.sum(covariancia_1**2) * np.sum(covariancia_2**
    return p
```

### Função de cálculo do coeficiente de correlação de pearson entre um filme e os demais, retornando os 10 filmes mais próximos

In [27]:

```
def recomenda_coeficiente_de_correlação_de_Pearson(filme, tabela_pivotada, n):
    recomendacoes = []
    for titulo in tabela_pivotada.columns:
        if titulo == filme:
            continue
        p = coeficiente_de_correlação_de_Pearson(tabela_pivotada[filme], tabela_pivotada[titulo])
        if np.isnan(p):
            continue
        else:
            filme_recomendado = (titulo, p)
            recomendacoes.append(filme_recomendado)

    recomendacoes.sort(key= lambda filme_recomendado: filme_recomendado[1], reverse=True)
    recomendacoes = recomendacoes[:n]
    recomendacoes = pd.DataFrame(recomendacoes)
    recomendacoes.columns = ['titulo', 'p']
    return recomendacoes
```

### Estes seriam os filmes indicados pela quarta heurística para quem assistiu o filme "Toy Story (1995)":

In [28]:

```
recomendacoes = recomenda_coeficiente_de_correlação_de_Pearson('Toy Story (1995)', tabela_pivotada=tabela
recomendacoes
```

Out[28]:

| | titulo | p |
|---|---|---|
| 0 | Toy Story 2 (1999) | 0.334316 |
| 1 | Monsters, Inc. (2001) | 0.235924 |
| 2 | Lion King, The (1994) | 0.212170 |
| 3 | Aladdin (1992) | 0.211972 |
| 4 | Bug's Life, A (1998) | 0.206260 |
| 5 | Back to the Future (1985) | 0.195335 |
| 6 | Jurassic Park (1993) | 0.192194 |
| 7 | Shrek (2001) | 0.186615 |
| 8 | Beauty and the Beast (1991) | 0.180192 |
| 9 | Babe (1995) | 0.162321 |

# 5. Heurística baseada na similaridade entre usuários

**Considerando que os votos de uma pessoa, é um vetor, é possível calcular a "distância" entre duas pessoas. Sendo assim, calcularemos a distância entre um usuário, usando a geometria euclidiana para tal.**

### Lista de filmes assistidas pelo novo usuário

In [29]:

```
lista_de_filmes = [
    ["Monty Python and the Holy Grail (1975)", 4.5],
    ["Fight Club (1999)", 5],
    ["Matrix, The (1999)", 3.5],
    ["Lion King, The (1994)", 5],
    ["Pulp Fiction (1994)", 2.5]
]
```

```
dados = []
for filme in lista_de_filmes:
    dados.append([filmes.query('titulo == @filme[0]').index[0], filme[1], int(0)])

lista_de_filmes = pd.DataFrame(lista_de_filmes)
lista_de_filmes.columns = ['titulo', 'nota']
lista_de_filmes
```

Out[29]:

| | titulo | nota |
|---|---|---|
| 0 | Monty Python and the Holy Grail (1975) | 4.5 |
| 1 | Fight Club (1999) | 5.0 |
| 2 | Matrix, The (1999) | 3.5 |
| 3 | Lion King, The (1994) | 5.0 |
| 4 | Pulp Fiction (1994) | 2.5 |

### Função para criar um novo usuário

In [30]:

```
def novo_usuario(dados):
    usuario = notas['usuarioId'].max()+1
    notas_do_usuario_novo = pd.DataFrame(dados, columns=['filmeId', 'nota', 'momento'])
    notas_do_usuario_novo['usuarioId'] = usuario
    return pd.concat([notas, notas_do_usuario_novo])
```

### Adicionar o novo usuário a tabela de notas

In [31]:

```
notas = novo_usuario(dados)
notas.tail(len(dados) + 5)
```

Out[31]:

| | usuarioId | filmeId | nota | momento |
|---|---|---|---|---|
| **27753439** | 283228 | 8542 | 4.5 | 1379882795 |
| **27753440** | 283228 | 8712 | 4.5 | 1379882751 |
| **27753441** | 283228 | 34405 | 4.5 | 1379882889 |
| **27753442** | 283228 | 44761 | 4.5 | 1354159524 |
| **27753443** | 283228 | 54286 | 4.5 | 1354159718 |
| **0** | 283229 | 1136 | 4.5 | 0 |
| **1** | 283229 | 2959 | 5.0 | 0 |
| **2** | 283229 | 2571 | 3.5 | 0 |
| **3** | 283229 | 364 | 5.0 | 0 |
| **4** | 283229 | 296 | 2.5 | 0 |

### Função para exibir as notas de um usuário

In [32]:

```
def notas_do_usuario(usuario):
    notas_do_usuario = notas.query("usuarioId == @usuario")
    notas_do_usuario = notas_do_usuario[["filmeId", "nota"]].set_index("filmeId")
    return notas_do_usuario
```

### Estas são as notas do novo usuário

In [33]:

```
notas_do_usuario(283229)
```

Out[33]:

| | nota |
|---|---|
| **filmeId** | |
| **1136** | 4.5 |
| **2959** | 5.0 |
| **2571** | 3.5 |
| **364** | 5.0 |
| **296** | 2.5 |

### Função que calcula a "distância" entre dois usuários

In [34]:

```
def distancia_de_usuarios(usuario_id1, usuario_id2, minimo = 5):
    notas1 = notas_do_usuario(usuario_id1)
    notas2 = notas_do_usuario(usuario_id2)
    diferencas = notas1.join(notas2, lsuffix="_esquerda", rsuffix="_direita").dropna()

    if(len(diferencas) < minimo):
        return None

    distancia =  np.linalg.norm(diferencas['nota_esquerda'] - diferencas['nota_direita'])
    return [usuario_id1, usuario_id2, distancia]
```

### Distância entre o novo usuário e o quarto usuário

In [35]:

```
distancia_de_usuarios(283229, 4)[2]
```

Out[35]:

```
2.9154759474226504
```

### Função que calcula a "distância" entre um usuário e os demais

In [36]:

```
def distancia_de_todos(voce_id, numero_de_usuarios_a_analisar = None):
    todos_os_usuarios = notas['usuarioId'].unique()
    if numero_de_usuarios_a_analisar:
        todos_os_usuarios = todos_os_usuarios[:numero_de_usuarios_a_analisar]
    distancias = []
    number_of_elements = len(todos_os_usuarios)
    for i in range(number_of_elements):
        distancias.append(distancia_de_usuarios(voce_id, todos_os_usuarios[i]))
        update_progress(i / number_of_elements)
    update_progress(1)
    distancias = list(filter(None, distancias))
    distancias = pd.DataFrame(distancias, columns = ["voce", "outra_pessoa", "distancia"])
    return distancias
```

### Distâncias do novo usuário para os 100 primeiro usuários

In [37]:

```
distancia_de_todos(283229, numero_de_usuarios_a_analisar=100).head()
```

```
Progress: [####################] 100.0%
```

Out[37]:

| | voce | outra_pessoa | distancia |
|---|---|---|---|
| **0** | 283229 | 4 | 2.915476 |
| **1** | 283229 | 42 | 3.082207 |
| **2** | 283229 | 56 | 3.570714 |
| **3** | 283229 | 67 | 3.570714 |
| **4** | 283229 | 71 | 2.692582 |

### Função que retorna a lista de usuários ordenada pela "distância"

In [38]:

```
def mais_proximos_de(voce_id, n_mais_proximos = 10, numero_de_usuarios_a_analisar=None):
    distancias = distancia_de_todos(voce_id, numero_de_usuarios_a_analisar=numero_de_usuarios_a_analisar)
    distancias = distancias.sort_values("distancia")
    distancias = distancias.set_index("outra_pessoa")
    try:
        distancias = distancias.drop(voce_id)
    finally:
        return distancias.head(n_mais_proximos)
```

### As 10 distâncias mais próximas entre o novo usuário e os 100 primeiro usuários

In [39]:

```
mais_proximos_de(283229, n_mais_proximos = 10, numero_de_usuarios_a_analisar=100)
```

```
Progress: [####################] 100.0%
```

Out[39]:

| outra_pessoa | voce | distancia |
|---|---|---|
| 100 | 283229 | 2.345208 |
| 71 | 283229 | 2.692582 |
| 4 | 283229 | 2.915476 |
| 42 | 283229 | 3.082207 |
| 81 | 283229 | 3.240370 |
| 56 | 283229 | 3.570714 |
| 67 | 283229 | 3.570714 |

### Função que retorna a lista de filmes recomendados baseado na similaridade entre os usuários

In [42]:

```
def sugere_para(voce, n_mais_proximos = 10, numero_de_usuarios_a_analisar = None):
    notas_de_voce = notas_do_usuario(voce)
    filmes_que_ja_viu = notas_de_voce.index
    similares = mais_proximos_de(voce, n_mais_proximos = n_mais_proximos ,
                                 numero_de_usuarios_a_analisar = numero_de_usuarios_a_analisar)
    usuarios_similares = similares.index
    notas_dos_similares = notas.set_index('usuarioId').loc[usuarios_similares]
    recomendacoes = notas_dos_similares.groupby('filmeId').mean()[['nota']]
    recomendacoes = recomendacoes.sort_values('nota', ascending = False)
    recomendacoes = recomendacoes.join(filmes)
    recomendacoes = recomendacoes.drop(filmes_que_ja_viu)
    return recomendacoes.sort_values('total_de_votos', ascending = False).head(10)[['titulo']]
```

### Estes seriam os filmes indicados pela quinta heurística para o novo usuário:

In [43]:

```
sugere_para(283229, numero_de_usuarios_a_analisar = 100)
```

```
Progress: [####################] 100.0%
```

Out[43]:

| filmeId | titulo |
|---|---|
| 318 | Shawshank Redemption, The (1994) |
| 356 | Forrest Gump (1994) |
| 593 | Silence of the Lambs, The (1991) |
| 260 | Star Wars: Episode IV - A New Hope (1977) |
| 480 | Jurassic Park (1993) |
| 527 | Schindler's List (1993) |
| 110 | Braveheart (1995) |
| 1 | Toy Story (1995) |
| 1210 | Star Wars: Episode VI - Return of the Jedi (1983) |
| 1196 | Star Wars: Episode V - The Empire Strikes Back... |