

# Laboratorio 2 --- Marcelo Cena

## Objetivos

En este práctico tendrán que integrar en la notebook el sistema de recomendación basado en contenido que se propone en [http://nbviewer.jupyter.org/github/khanhnamle1994/movielens/blob/master/Content\\_Based\\_\(http://nbviewer.jupyter.org/github/khanhnamle1994/movielens/blob/master/Content\\_Based\\_\)](http://nbviewer.jupyter.org/github/khanhnamle1994/movielens/blob/master/Content_Based_(http://nbviewer.jupyter.org/github/khanhnamle1994/movielens/blob/master/Content_Based_)) and\_Collaborative\_Filtering\_Models.ipynb.

Entrega Entregar un notebook documentado a través de slack, antes del día 7 de septiembre.

## Datasets

Usaremos un dataset extraido de <https://labrosa.ee.columbia.edu/millionsong/> (<https://labrosa.ee.columbia.edu/millionsong/>) que consiste en datos de canciones extraidos de diferentes sitios y el rating que los usuarios le dan a las mismas. Este dataset contiene dos archivos, uno guardado en formato csv, contiene metadatos de cada canción, como id, título, disco, artista y año, de los cuales vamos a utilizar solamente los datos correspondientes a los titulos y disco para intentar hacer un clustering de las canciones en relación a las palabras en común. Como en la version anterior conseguimos un gran cluster con muchas canciones y todos los otros con no mas de 500, vamos a probar concatenando el nombre de la cancion con el nombre del disco .

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.sparse.linalg import svds
%matplotlib inline

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
import wordcloud
from wordcloud import WordCloud, STOPWORDS
```

## Loading of Data

Cargamos el dataset, extraemos los datos de titulos y albumes y votos de los oyentes, como en el laboratorio anterior. Unimos los dos datasets y le agregamos una columna cancion+album y despues recortamos el dataset (de 2 millones a 40 mil). Despues utilizamos TfidfVectorizer para representarlos.

In [2]:

```
canciones = pd.read_csv('data/song_data.csv')
#display(canciones.head(2))
canciones_df = pd.read_table('data/Songs_10000.txt', header=None)
canciones_df.columns = ['user_id', 'song_id', 'listen_count']
#display(canciones_df.head(2))

song_df1 = pd.merge(canciones_df, canciones.drop_duplicates(['song_id']), on="song_id",
 how="left")

song_df= song_df1.head(35000)
song_df["tag"] = song_df["title"] + " - " + song_df["release"]
display(song_df.head(5))
```

C:\Users\marcelo.cena\Anaconda3\lib\site-packages\ipykernel\_launcher.py:1

0: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy>

# Remove the CWD from sys.path while we load stuff.

	user_id	song_id	listen_coun
0	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOAKIMP12A8C130995	1
1	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBBMDR12A8C13253B	2
2	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBXHDL12A81C204C0	1
3	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBYHAJ12A6701BF1D	1
4	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SODACBL12A8C13C273	1

Para facilitar la implementación de las técnicas de recomendación, a las variables de event\_type se les asigna un valor cuantitativo el cual se agrega como un dato mas en el dataset\_2.

In [3]:

```
display(canciones.head(10))
display(canciones_df.head(100))

display(song_df.head(40))
```

	<b>song_id</b>	<b>title</b>	<b>release</b>	<b>artist_name</b>	<b>year</b>
<b>0</b>	SOQMMHC12AB0180CB8	Silent Night	Monster Ballads X-Mas	Faster Pussy cat	2003
<b>1</b>	SOVFVAK12A8C1350D9	Tanssi vaan	Karkuteillä	Karkkiautomaatti	1995
<b>2</b>	SOGTUKN12AB017F4F1	No One Could Ever	Butter	Hudson Mohawke	2006
<b>3</b>	SOBNYVR12A8C13558C	Si Vos Querés	De Culo	Yerba Brava	2003
<b>4</b>	SOHSBXH12A8C13B0DF	Tangle Of Aspens	Rene Ablaze Presents Winter Sessions	Der Mystic	0
<b>5</b>	SOZVAPQ12A8C13B63C	Symphony No. 1 G minor "Sinfonie Serieuse"/All...	Berwald: Symphonies Nos. 1/2/3/4	David Montgomery	0
<b>6</b>	SOQVRHI12A6D4FB2D7	We Have Got Love	Strictly The Best Vol. 34	Sasha / Turbulence	0
<b>7</b>	SOEYRFT12AB018936C	2 Da Beat Ch'yall	Da Bomb	Kris Kross	1993
<b>8</b>	SOPMIYT12A6D4F851E	Goodbye	Danny Boy	Joseph Locke	0
<b>9</b>	SOJCFMH12A8C13B0C2	Mama_ mama can't you see ?	March to cadence with the US marines	The Sun Harbor's Chorus- Documentary Recordings	0

	user_id	song_id	listen_cc
0	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOAKIMP12A8C130995	1
1	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBBMDR12A8C13253B	2
2	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBXHDL12A81C204C0	1
3	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBYHAJ12A6701BF1D	1
4	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SODACBL12A8C13C273	1
5	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SODDNQT12A6D4F5F7E	5
6	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SODXRTY12AB0180F3B	1
7	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOFGUAY12AB017B0A8	1
8	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOFRQTD12A81C233C0	1
9	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOHQWYZ12A6D4FA701	1
10	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOIYTOA12A6D4F9A23	1
11	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOIZAZL12A6701C53B	5
12	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOJNNUA12A8AE48C7A	1
13	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOJPFQG12A58A7833A	1
14	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOKRIMP12A6D4F5DA3	5
15	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOLLGNU12AF72A4D4F	1
16	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOMGIYR12AB0187973	6
17	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOMLMKI12A81C204BC	1
18	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOMSQJY12A8C138539	1
19	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SONSAEZ12A8C138D7A	1
20	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOOKGRB12A8C13CD66	1
21	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOPCVQE12AC468AF36	1
22	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOQIVUD12AB01821D2	1
23	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOQJLDY12AAF3B456D	1
24	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOQLCKR12A81C22440	1
25	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SORPMYJ12AF729EB90	1
26	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SORQHCG12A58A7EEBA	1
27	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SORUVVF12AB018230B	1
28	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SORWLTW12A670208FA	1
29	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SORZASF12A6D4F8CFA	2
...	...	...	...
70	969cc6fb74e076a68e36a04409cb9d3765757508	SOQBMFK12A8C13835B	1
71	969cc6fb74e076a68e36a04409cb9d3765757508	SORPHNV12A6701CD63	1
72	969cc6fb74e076a68e36a04409cb9d3765757508	SOUFPNI12A8C142D19	1

	<b>user_id</b>	<b>song_id</b>	<b>listen_cc</b>
73	969cc6fb74e076a68e36a04409cb9d3765757508	SOXHIDK12A58A7CFB3	10
74	969cc6fb74e076a68e36a04409cb9d3765757508	SOXIIM12A6D4F66C8	2
75	4bd88bfb25263a75bbdd467e74018f4ae570e5df	SODGVGW12AC9075A8D	6
76	4bd88bfb25263a75bbdd467e74018f4ae570e5df	SOEPZQS12A8C1436C7	13
77	4bd88bfb25263a75bbdd467e74018f4ae570e5df	SOGDDKR12A6701E8FA	6
78	4bd88bfb25263a75bbdd467e74018f4ae570e5df	SOGEKGR12A6D4F81E8	4
79	4bd88bfb25263a75bbdd467e74018f4ae570e5df	SOJNQZO12AF72AAE32	2
80	4bd88bfb25263a75bbdd467e74018f4ae570e5df	SOKOXWU12AF72AD1BC	4
81	4bd88bfb25263a75bbdd467e74018f4ae570e5df	SOMNTMT12A8C1400F6	1
82	4bd88bfb25263a75bbdd467e74018f4ae570e5df	SOUSQCN12A8C133302	3
83	4bd88bfb25263a75bbdd467e74018f4ae570e5df	SOVEUVC12A6310EAF1	5
84	4bd88bfb25263a75bbdd467e74018f4ae570e5df	SOWEHOM12A6BD4E09E	1
85	4bd88bfb25263a75bbdd467e74018f4ae570e5df	SOWGXOP12A6701E93A	8
86	4bd88bfb25263a75bbdd467e74018f4ae570e5df	SOWPAXV12A67ADA046	18
87	4bd88bfb25263a75bbdd467e74018f4ae570e5df	SOXGQEM12AB0181D35	12
88	e006b1a48f466bf59feefed32bec6494495a4436	SOAUWYT12A81C206F1	2
89	e006b1a48f466bf59feefed32bec6494495a4436	SOAXGDH12A8C13F8A1	2
90	e006b1a48f466bf59feefed32bec6494495a4436	SOBFMHC12A6D4F9401	1
91	e006b1a48f466bf59feefed32bec6494495a4436	SOBONKR12A58A7A7E0	2
92	e006b1a48f466bf59feefed32bec6494495a4436	SOBTRCD12A6701E976	2
93	e006b1a48f466bf59feefed32bec6494495a4436	SODEOCO12A6701E922	2
94	e006b1a48f466bf59feefed32bec6494495a4436	SODJWHY12A8C142CCE	1
95	e006b1a48f466bf59feefed32bec6494495a4436	SOEGIYH12A6D4FC0E3	3
96	e006b1a48f466bf59feefed32bec6494495a4436	SOEKGDE12AB0182795	4
97	e006b1a48f466bf59feefed32bec6494495a4436	SOFRQTD12A81C233C0	3
98	e006b1a48f466bf59feefed32bec6494495a4436	SOIICEQ12A6D4F7FE0	1
99	e006b1a48f466bf59feefed32bec6494495a4436	SOIRRMU12A6D4FB0C0	2

100 rows × 3 columns



	user_id	song_id	listen_count
0	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOAKIMP12A8C130995	1
1	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBBMDR12A8C13253B	2
2	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBXHDL12A81C204C0	1
3	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBYHAJ12A6701BF1D	1
4	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SODACBL12A8C13C273	1
5	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SODDNQT12A6D4F5F7E	5
6	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SODXRTY12AB0180F3B	1
7	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOFGUAY12AB017B0A8	1
8	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOFRQTD12A81C233C0	1
9	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOHQWYZ12A6D4FA701	1
10	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOIYTOA12A6D4F9A23	1
11	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOIZAZL12A6701C53B	5
12	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOJNNUA12A8AE48C7A	1
13	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOJPFQG12A58A7833A	1
14	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOKRIMP12A6D4F5DA3	5

	<b>user_id</b>	<b>song_id</b>	<b>listen_col</b>
<b>15</b>	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOLLGNU12AF72A4D4F	1
<b>16</b>	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOMGIYR12AB0187973	6
<b>17</b>	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOMLMKI12A81C204BC	1
<b>18</b>	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOMSQJY12A8C138539	1
<b>19</b>	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SONSAEZ12A8C138D7A	1
<b>20</b>	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOOKGRB12A8C13CD66	1
<b>21</b>	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOPCVQE12AC468AF36	1
<b>22</b>	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOQIVUD12AB01821D2	1
<b>23</b>	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOQJLDY12AAF3B456D	1
<b>24</b>	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOQLCKR12A81C22440	1
<b>25</b>	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SORPMYJ12AF729EB90	1
<b>26</b>	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SORQHCG12A58A7EEBA	1
<b>27</b>	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SORUFVF12AB018230B	1
<b>28</b>	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SORWLTW12A670208FA	1
<b>29</b>	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SORZASF12A6D4F8CFA	2
<b>30</b>	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOSYBEV12AB0182933	1
<b>31</b>	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOTFATN12A6D4FA74D	1

	<b>user_id</b>	<b>song_id</b>	<b>listen_count</b>
32	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOTLVCL12AB0182D22	1
33	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOTRSFZ12A8C142BF6	1
34	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOUKXIN12A8C133C7F	1
35	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOVHRGF12A8C13852F	1
36	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOVQEYZ12A8C1379D8	1
37	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOVYIYI12A8C138D88	1
38	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOWQLXP12AF72A08A2	1
39	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOWSPUS12AC468BEE3	1

◀ ▶

In [4]:

```
import keras
```

```
print(keras.__version__)
```

```
C:\Users\marcelo.cena\Anaconda3\lib\site-packages\h5py\_init_.py:36: FutureWarning: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
```

```
from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

2.2.2

## Análisis exploratorio

Showing the most popular songs in the dataset||

In [5]:

```
song_grouped = song_df.groupby(['title']).agg({'listen_count': 'count'}).reset_index()
grouped_sum = song_grouped['listen_count'].sum()
song_grouped['percentage'] = song_grouped['listen_count'].div(grouped_sum)*100
song_grouped.sort_values(['listen_count', 'title'], ascending = [0,1])
```

Out[5]:

		<b>title</b>	<b>listen_count</b>	<b>percentage</b>
<b>5958</b>	Sehr kosmisch	135	0.385714	
<b>1714</b>	Dog Days Are Over (Radio Edit)	130	0.371429	
<b>8253</b>	You're The One	122	0.348571	
<b>7580</b>	Undo	119	0.340000	
<b>5661</b>	Revelry	118	0.337143	
<b>5949</b>	Secrets	109	0.311429	
<b>2251</b>	Fireflies	90	0.257143	
<b>2990</b>	Horn Concerto No. 4 in E flat K495: II. Romanc...	89	0.254286	
<b>7379</b>	Tive Sim	78	0.222857	
<b>4355</b>	Marry Me	76	0.217143	
<b>2897</b>	Hey_Soul Sister	74	0.211429	
<b>7123</b>	The Scientist	74	0.211429	
<b>7627</b>	Use Somebody	72	0.205714	
<b>1279</b>	Clocks	67	0.191429	
<b>1004</b>	Bulletproof	65	0.185714	
<b>1841</b>	Drop The World	65	0.185714	
<b>4986</b>	OMG	63	0.180000	
<b>1152</b>	Catch You Baby (Steve Pitron & Max Sanna Radio...)	62	0.177143	
<b>3679</b>	Just Dance	62	0.177143	
<b>1104</b>	Canada	61	0.174286	
<b>1438</b>	Creep (Explicit)	60	0.171429	
<b>4194</b>	Love Story	58	0.165714	
<b>751</b>	Billionaire [feat. Bruno Mars] (Explicit Albu...)	57	0.162857	
<b>8144</b>	Yellow	57	0.162857	
<b>237</b>	Alejandro	55	0.157143	
<b>4231</b>	Lucky (Album Version)	54	0.154286	
<b>5645</b>	Représente	53	0.151429	
<b>213</b>	Ain't Misbehavin	52	0.148571	
<b>5490</b>	Pursuit Of Happiness (nightmare)	52	0.148571	
<b>6291</b>	Somebody To Love	52	0.148571	
...	...	...	...	
<b>8172</b>	You Are My Sister	1	0.002857	
<b>8175</b>	You Are So Beautiful	1	0.002857	

		<b>title</b>	<b>listen_count</b>	<b>percentage</b>
8191	You Don't Understand Me	1	0.002857	
8192	You Don't Have A Clue	1	0.002857	
8198	You Dont Know Me	1	0.002857	
8200	You Found Me (Album Version)	1	0.002857	
8212	You Know What You Are?	1	0.002857	
8219	You Never Know	1	0.002857	
8230	You Should Have Killed Me When You Had The Chance	1	0.002857	
8231	You Spin Me Round (Like A Record)	1	0.002857	
8232	You Think I Ain't Worth A Dollar_ But I Feel L...	1	0.002857	
8233	You Told A Lie	1	0.002857	
8237	You Will Leave a Mark	1	0.002857	
8240	You and Me	1	0.002857	
8255	You've Got A Friend	1	0.002857	
8259	You've Passed	1	0.002857	
8262	Young And Wild	1	0.002857	
8264	Younger Than Springtime	1	0.002857	
8265	Your Arms Feel Like home	1	0.002857	
8267	Your English Is Good	1	0.002857	
8277	Your Mother's Here To Stay (LP Version)	1	0.002857	
8279	Your Picture	1	0.002857	
8281	Your Rap Is Sad	1	0.002857	
8282	Your Rocky Spine	1	0.002857	
8283	Your Song	1	0.002857	
8299	Zwitter	1	0.002857	
8303	high fives	1	0.002857	
8304	in white rooms	1	0.002857	
8312	the Love Song	1	0.002857	
8314	¿Lo Ves? [Piano Y Voz]	1	0.002857	

8316 rows × 3 columns

## Armamos una nube de palabras con los títulos de los temas

In [6]:

```
song_df['title'] = song_df['title'].fillna("").astype('str')
title_corpus = ' '.join(song_df['title'])
title_wordcloud = WordCloud(stopwords=STOPWORDS, background_color='black', height=2000,
                             width=4000).generate(title_corpus)

# Plot the wordcloud
plt.figure(figsize=(16,8))
plt.imshow(title_wordcloud)
plt.axis('off')
plt.show()
```

C:\Users\marcelo.cena\Anaconda3\lib\site-packages\ipykernel\_launcher.py:1:

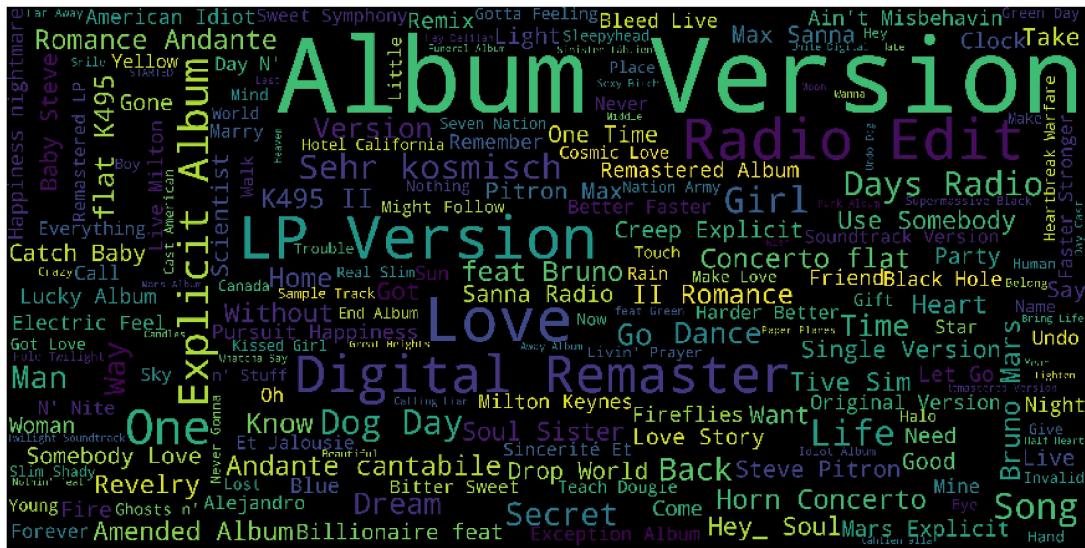
**SettingWithCopyWarning:**

A value is trying to be set on a copy of a slice from a DataFrame.

Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy>

```
"""Entry point for launching an IPython kernel.
```



Se pueden apreciar los títulos más comunes, obviamente aparecen Album y Version además de las palabras Radio, edit, Digital y Remaster

# Ratings

Ahora vamos a ver que hay de los ratings

In [7]:

```
song_df['listen_count'].describe()
```

Out[7]:

count	35000.000000
mean	3.054971
std	7.537485
min	1.000000
25%	1.000000
50%	1.000000
75%	3.000000
max	796.000000
Name:	listen_count, dtype: float64

In [8]:

```
import seaborn as sns
sns.set_style('whitegrid')
sns.set(font_scale=1.5)

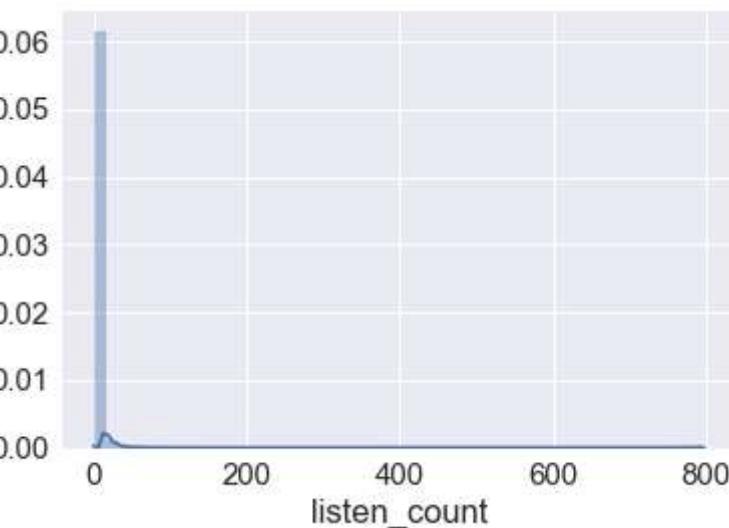
# Display distribution of rating
sns.distplot(song_df['listen_count'].fillna(song_df['listen_count'].median()))
```

C:\Users\marcelo.cena\AppData\Roaming\Python\Python35\site-packages\matplotlib\axes\\_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.

warnings.warn("The 'normed' kwarg is deprecated, and has been "

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x181b7b03358>
```



Podemos observar que la mayoría de las canciones fue escuchada pocas veces con 4, seguidos de 5 y 3. Muy pocas fueron oídas muchas veces. No está distribuido uniformemente.

## Autores

Una suposición básica es que los autores suelen escribir canciones generalmente parecidas. Vamos a mostrar en una tabla cuales son los autores que mas escribieron.

In [9]:

```
song_df['artist_name'] = song_df['artist_name'].fillna("").astype('str')
title_corpus = ' '.join(song_df['artist_name'])
title_wordcloud = WordCloud(stopwords=STOPWORDS, background_color='black', height=2000,
                             width=4000).generate(title_corpus)

# Plot the wordcloud
plt.figure(figsize=(16,8))
plt.imshow(title_wordcloud)
plt.axis('off')
plt.show()
```

C:\Users\marcelo.cena\Anaconda3\lib\site-packages\ipykernel\_launcher.py:1:

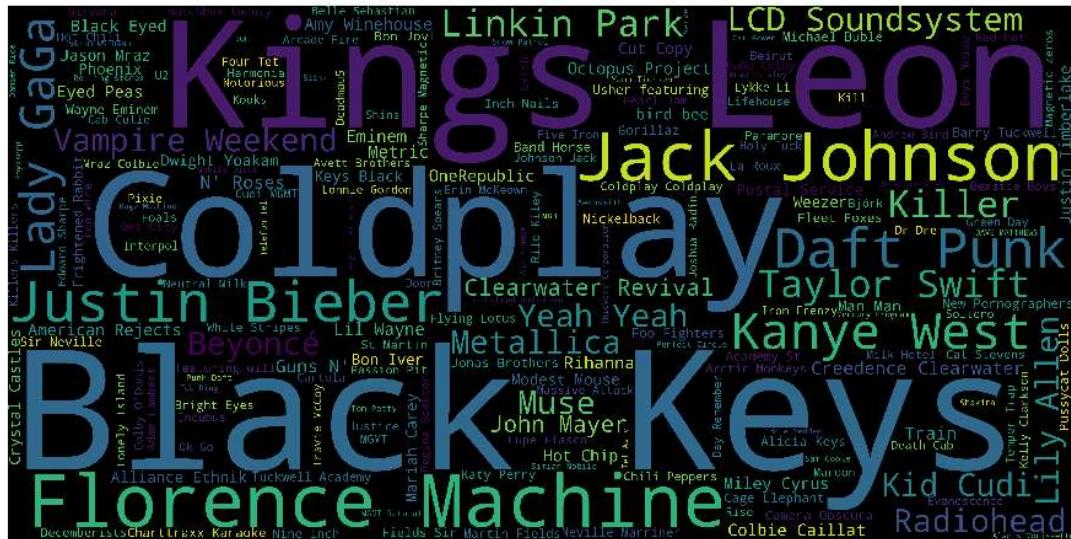
## SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy>

```
"""Entry point for launching an IPython kernel.
```



Aquí, claramente, se observa que artistas como Jack Johnson, Black Keys y Kings Leon son los más escuchados, seguidos de cerca por Florence Machine y Justin Bieber

## Tema + Disco

Otra suposición básica es que las diskeras suelen agrupar canciones relacionadas en los mismos discos. Vamos a mostrar en una tabla si tales relaciones se cumplen.

In [10]:

```
song_df['tag'] = song_df['tag'].fillna("").astype('str')
title_corpus = ' '.join(song_df['tag'])
title_wordcloud = WordCloud(stopwords=STOPWORDS, background_color='black', height=2000,
                             width=4000).generate(title_corpus)

# Plot the wordcloud
plt.figure(figsize=(16,8))
plt.imshow(title_wordcloud)
plt.axis('off')
plt.show()
```

C:\Users\marcelo.cena\Anaconda3\lib\site-packages\ipykernel\_launcher.py:1:

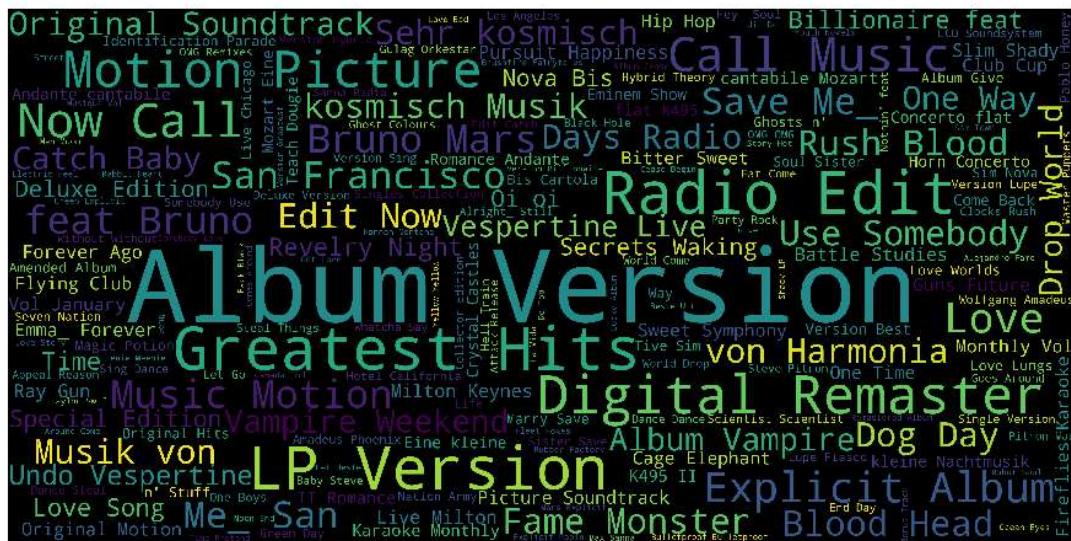
## SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy>

```
"""Entry point for launching an IPython kernel.
```



acá vemos un poco mas de dispersion, las palabras mas usadas son Album y Version seguidos por Greatest Hits, Radio Edit, LP Version

- Vamos a aplicar TfidfVectorizer a los títulos y quitarle las stop words a ver que sucede

# Hay 2 Tipos de motores de recomendación

## 1. Basados en contenido

El recomendador basado en el contenido se basa en la similitud de los elementos que se recomiendan. La idea básica es que si te gusta un artículo, también te gustará un artículo "similar". Por lo general, funciona bien cuando es fácil determinar el contexto / propiedades de cada elemento.

Un recomendador basado en contenido funciona con datos que el usuario proporciona, en este caso las clasificaciones de las películas de MovieLens. En función de esos datos, se genera un perfil de usuario, que luego se utiliza para hacer sugerencias al usuario de nuevas películas. A medida que el usuario proporciona más calificaciones, el motor se vuelve cada vez más preciso.

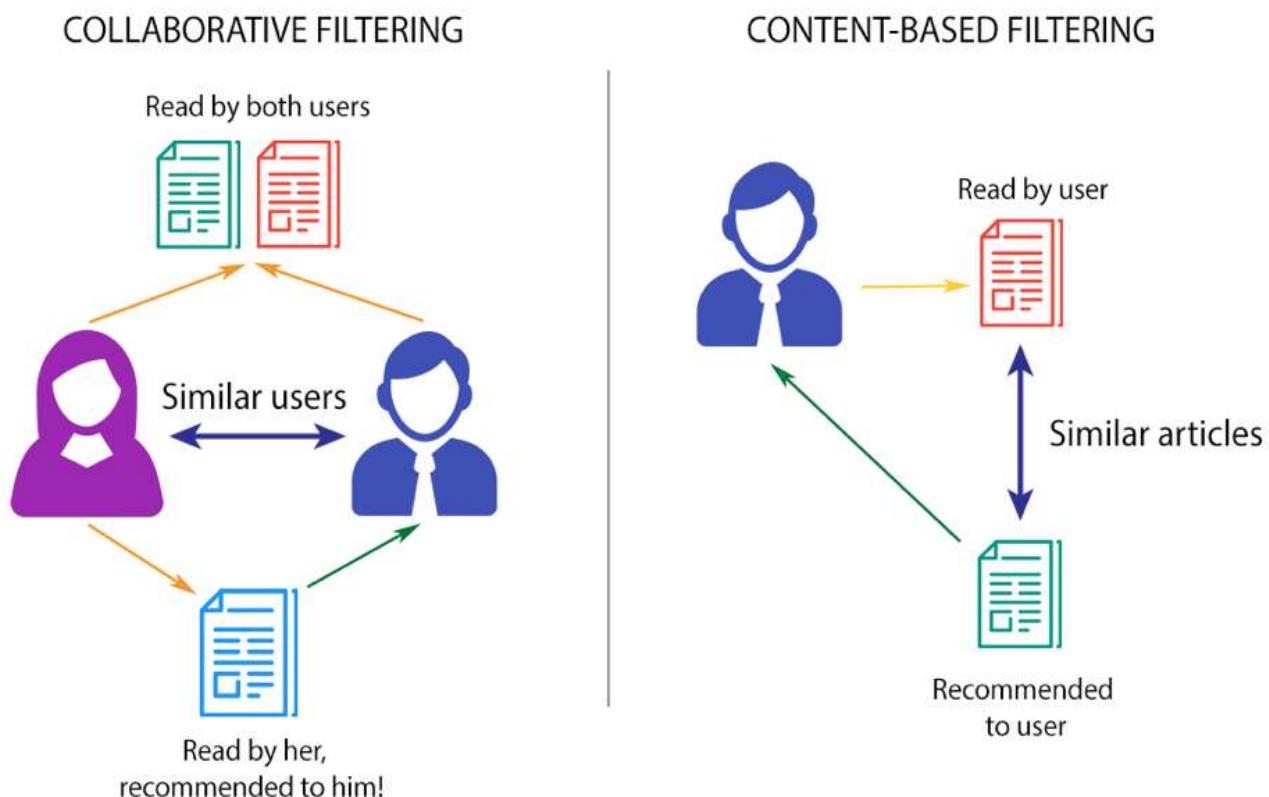
## 2. Filtrado colaborativo

El "Collaborative Filtering Recommender" se basa completamente en el comportamiento pasado y no en el contexto. Más específicamente, se basa en la similitud de preferencias, gustos y elecciones de dos usuarios. Analiza cuán similares son los gustos de un usuario a otro y hace recomendaciones sobre la base de eso.

Por ejemplo, si al usuario A le gustan las películas 1, 2, 3 y al usuario B le gustan las películas 2,3,4, entonces tienen intereses similares y A debería gustarle a la película 4 y B debería gustarle la película 1.

En general, el filtrado colaborativo es el caballo de batalla de los motores de recomendación. El algoritmo tiene una propiedad muy interesante de poder hacer el aprendizaje de características por sí mismo, lo que significa que puede comenzar a aprender por sí mismo qué características usar.

Se puede dividir en Filtrado colaborativo basado en memoria y Filtrado colaborativo basado en modelo.



# Modelo de recomendación basado en el contenido

## Teoría

Los conceptos de Frecuencia de término (TF) y Frecuencia inversa de documento (IDF) se utilizan en sistemas de información y también en mecanismos de filtrado basados en contenido (como un recomendador basado en contenido). Se usan para determinar la importancia relativa de un documento / artículo / noticia / película, etc.

TF es simplemente la frecuencia de una palabra en un documento. IDF es la inversa de la frecuencia del documento entre todo el corpus de documentos. TF-IDF se utiliza principalmente por dos motivos: supongamos que buscamos

"**los resultados de los últimos juegos europeos de Futbol**" en Google.

Es cierto que las palabras como "el", "los", "de" ocurrirán con más frecuencia que "**juegos europeos de Futbol**" pero la importancia relativa de **juegos europeos de Futbol** es más alta que el punto de vista de la búsqueda. En tales casos, la ponderación TF-IDF niega el efecto de las palabras de alta frecuencia para determinar la importancia de un elemento (documento).

## Implementación del un Montor de Recomendacion basado en contenido.

Utilizamos la Similitud coseno para calcular una cantidad numérica que denote la similitud entre dos películas. Ya que hemos utilizado el Vectorizador TF-IDF, al calcular el Producto Punto, nos dará directamente el Puntaje de Similaridad. Utilizaremos kernel lineal de sklearn.

In [11]:

```
tf = TfidfVectorizer(analyzer='word', ngram_range=(1, 2), min_df=0, stop_words='english')
tfidf_matrix = tf.fit_transform(song_df['tag'])
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

```
C:\Users\marcelo.cena\Anaconda3\lib\site-packages\sklearn\feature_extraction\text.py:1089: FutureWarning: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
```

```
if hasattr(X, 'dtype') and np.issubdtype(X.dtype, np.float):
```

In [12]:

```
cosine_sim
```

Out[12]:

```
array([[1., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 0., 1.]])
```

In [13]:

```
print('Veamos por ejemplo cuáles canciones fueron oidas por el usuario con ID= b80344d0  
63b5ccb3212f76538f3d9e43d87dca9e:')

song_df[song_df.user_id=='b80344d063b5ccb3212f76538f3d9e43d87dca9e'][['tag','listen_count']].sort_values(by='listen_count', ascending=False).head(3)
```

Veamos por ejemplo cuáles canciones fueron oidas por el usuario con ID= b80344d063b5ccb3212f76538f3d9e43d87dca9e:

Out[13]:

	tag	listen_count
<b>43</b>	Moonshine - Thicker Than Water	8
<b>16</b>	Behind The Sea [Live In Chicago] - Live In Chi...	6
<b>14</b>	I?m A Steady Rollin? Man - Diggin' Deeper Vol...	5

- Comportamiento de dos usuarios

In [14]:

```
index = song_df.listen_count
user_1 = song_df[song_df.user_id=='b80344d063b5ccb3212f76538f3d9e43d87dca9e']['listen_count']

user_1 = pd.Series(user_1, name='usuario_1')
user_2= song_df[song_df.user_id=='4bd88fb25263a75bbdd467e74018f4ae570e5df']['listen_count']
user_2 = pd.Series(user_2, name='usuario_2')
user_1_2=pd.concat([user_1 ,user_2],axis=1).fillna(0)

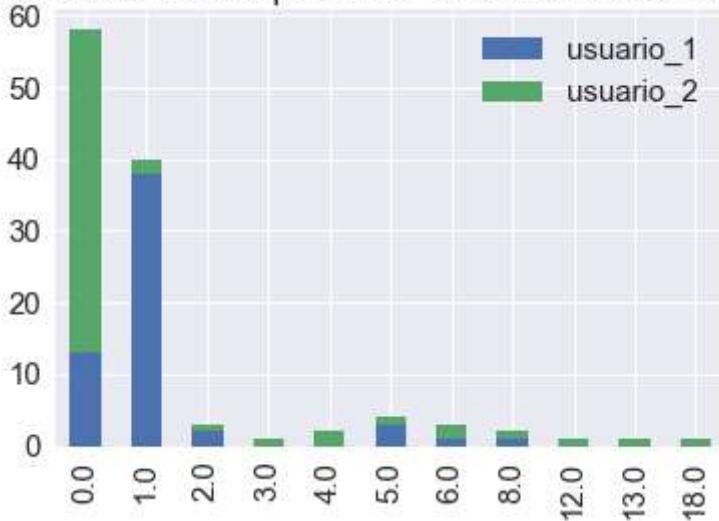
result = user_1_2.apply(pd.value_counts).fillna(0)

result.plot(y=['usuario_1', 'usuario_2'],kind='bar', stacked=True,title='Cantidad de veces que los usuarios escuchan un tema')
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x181c2450f98>
```

Cantidad de veces que los usuarios escuchan un tema



La barra correspondiente a **0.0** muestra que el **usuario\_1** tuvo menor interacción con los temas que el **usuario\_2**, sin embargo se observa que los dos usuarios tienden a tener gustos disimiles teniendo en cuenta la cantidad de veces que coinciden en un tema, es decir, sus gustos musicales no son parecidos. Adicionalmente, vemos que el **usuario\_2** tiende a escuchar muchas veces (12-13-18) algunos temas, mientras que el **usuario\_1** tiende a escuchar una sola vez distintos temas (alrededor de 35).

## Recomendación

- Recomendación Basada en Contenido

Para implementar el recomendador basado en el contenido nos limitaremos en usar el dataset\_1 usando la similitud de los títulos de los artículos y el texto de los mismos, dado que la idea básica es que si te gusta un artículo, también te gustará un artículo "similar".

### Implementación

Como los artículos tienen distintos idiomas, observe como se distribuyen los mismos.

In [15]:

```
edio_en = song_df['tag']
edio_en.head(3)
```

Out[15]:

```
0      The Cove - Thicker Than Water
1      Entre Dos Aguas - Flamenco Para Niños
2      Stronger - Graduation
Name: tag, dtype: object
```

Entonces generamos un dataset que contenga solo temas musicales en idioma inglés.

In [16]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
tf = TfidfVectorizer(analyzer='word', ngram_range=(1, 3), min_df=0, stop_words='english')
tfidf_matrix = tf.fit_transform(edio_en)
tfidf_matrix.shape
```

C:\Users\marcelo.cena\Anaconda3\lib\site-packages\sklearn\feature\_extraction\text.py:1089: FutureWarning: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.

```
    if hasattr(X, 'dtype') and np.issubdtype(X.dtype, np.float):
```

Out[16]:

```
(35000, 48036)
```

La similitud numérica entre los artículos:

In [17]:

```
from sklearn.metrics.pairwise import linear_kernel
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
cosine_sim[:4, :4]
```

Out[17]:

```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

Vemos que existe valores igual a 1 fuera de la diagonal, esto se debe a que los artículos se repiten en el dataset.

Usando la matriz de similitud de coseno pairwise para todos los artículos. Veamos los 10 artículos más similares en función de la puntuación de similitud del coseno.

In [18]:

```
# Build a 1-dimensional array with movie titles
titulos = song_df['tag']
indices = pd.Series(song_df.index, index=song_df['tag'])

# Function that get movie recommendations based on the cosine similarity score of movie genres
def text_recommendations(titulo):
    idx = indices[titulo]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[0], reverse=True)
    sim_scores = sim_scores[1:11]
    canciones_indices = [i[0] for i in sim_scores]
    return titulos.iloc[canciones_indices]
```

Vemos como resulta la recomendación basada en contenido para un artículo en particular

In [19]:

```
display(song_df.tag[4])
text_recommendations(song_df.tag[4])
```

'Learn To Fly - There Is Nothing Left To Lose'

Out[19]:

```
14    I?'m A Steady Rollin? Man - Diggin' Deeper Vol...
13    Clarity - As/Is: Cleveland/Cincinnati_ OH - 8/...
12          Love Shack - Original Hits - Rock
11    I'll Be Missing You (Featuring Faith Evans & 1...
10          Let It Be Sung - If I Had Eyes
9     Heaven's gonna burn your eyes - Hôtel Costes 7...
8          Sehr kosmisch - Musik von Harmonia
7     Stacked Actors - There Is Nothing Left To Lose
6          Paper Gangsta - The Fame Monster
5     Apuesta Por El Rock 'N' Roll - Antología Audio...
Name: tag, dtype: object
```

In [20]:

```
display(song_df.tag[1003])
text_recommendations(song_df.tag[1003])
```

'Fix You - X & Y'

Out[20]:

```
42             Right Back - 40oz To Freedom
41             City Love - Any Given Thursday
40                 Sun Giant - Sun Giant
39     Bigger Isn't Better - Born on the Wrong Planet
38                 Trani - Youth And Young Manhood
37             He Doesn't Know Why - Fleet Foxes
36 Come Back To Bed - As/Is: Philadelphia_PA/Ha...
35             Generator - There Is Nothing Left To Lose
34                 Drive - Make Yourself
33             Are You In? - Monuments And Melodies
Name: tag, dtype: object
```

Como no pudimos utilizar todo el dataset por razones de memoria, la información aquí extraída es absolutamente sesgada ya que estamos tomando las primeras 30000 filas del archivo. Podríamos haber probado haciendo una selección aleatoria de las filas del archivo, pero igual vamos a tener el problema de la pérdida de variabilidad. A pesar de esto, vemos que hay una buena concordancia en la similitud de temas utilizando título de la canción más nombre del álbum. Si hacemos un análisis un poco más profundo podemos llegar a apreciar que más que el nombre en sí, vemos relaciones entre palabras : Fix You con Are you in?/Come Back... pero ahí ya nos estamos metiendo en otro tipo de problemas ...