Introduction:

The Million Song dataset contains information on each song such as artist, track title, timestamp of when the song was added to the database, a list of tags, a list of similar songs, and an Echo Nest track id.

Reconciling Data with Million Playlist Dataset:

Unfortunately the Echo Nest API has been shutdown, so the main challenge in dealing with this dataset is to find a way to correlate Echo Nest track id’s with Spotify id’s. We have tried a few methods, but so far have been unsuccessful. The first method was to attempt to search the Spotify Web API for each artist and track name then populate the Spotify ID. While this did work in a few cases, the API frequently returned no results even when there should be a match.

The next attempt was to match the song and artist names in each dataset in order to build a map between them. This method was slow and did not provide a high number of matches.

We then moved on to starting with the Spotify song data that we already downloaded and obtaining the data from the last.fm API for that song directly. This method had a very high success rate, but it was very time consuming. The last.fm API would not match the cleaned song and artist names, so we had to redownload the song name and artist name from the Spotify API using the Spotify ID. This process would have taken several weeks to complete.

Since we were unable to match the Million Song data with the Spotify data, we were not able to incorporate the information into our model.

In [1]:

```
import numpy as np
import pandas as pd

import math
from scipy.special import gamma

import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
sns.set()

from IPython.display import display

import os
import re
import json
```

In [2]:

```
def process_file(file_name, data_dict):
    with open(file_name) as json_data:
        data = json.load(json_data)
    tag_length = len(data['tags'])
    similars_length = len(data['similars'])
    key_str = file_name.split('/')[-1][:-5]
    data_dict[key_str] = data
    return data_dict
```

In [3]:

```
def generate_file_list(directory):
    file_list = []
    #this loop properly gets all files in the directory
    for directory, sub_dirs, files in os.walk(directory):
        for name in files:
            if name[-4:] == 'json':
                file_list.append(directory + '/' + name)
    return file_list
```

In [5]:

```
def parse_data(data):
    similars_list = []
    tags_list = []
    songs_list = []
    for key, song in data.items():
        songs_list.append([song['artist'], song['timestamp'], song['track_id'],
song['title']])
        for similar_list in song['similars']:
            similars_list.append([key]+similar_list)
        for tag_list in song['tags']:
            tags_list.append([key, re.sub('[^a-z0-9 ]+','',tag_list[0].lower())
, tag_list[1]])

    similars_df = pd.DataFrame(similars_list, columns=['track_id1', 'track_id2'
, 'similarity'])
    tags_df = pd.DataFrame(tags_list, columns=['track_id', 'tag', 'strength'])
    songs_df = pd.DataFrame(songs_list, columns=['artist', 'timestamp', 'track_
id', 'title'])

    return similars_df, tags_df, songs_df
```

In [6]:

```
def process_data(directory_in, save_to_disk=True,
                 similars_file_out='data/similars_df.json',
                 tags_file_out='data/tags_df.json',
                 songs_file_out='data/songs_df.json'):
    #retrieve list of all json files in directory and subdirectories
    print('Generating File List')
    file_list = generate_file_list(directory_in)
    #extract data from json files into dict
    data= {}
    print('Reading data from files')
    for name in file_list:
        data = process_file(name, data)
    #parse data into three separate dataframes
    print('Putting Data into dataframes')
    similars_df, tags_df, songs_df = parse_data(data)

    if save_to_disk == True:
        #save dataframes for later use
        print('Saving data to disk')
        similars_df.to_json(similars_file_out)
        tags_df.to_json(tags_file_out)
        songs_df.to_json(songs_file_out)
    return similars_df, tags_df, songs_df
```

In [46]:

```
similars_df, tags_df, songs_df = \
process_data('data/lastfm_train/', True, 'data/similars_train.json',\
             'data/tags_train.json', 'data/songs_train.json')
```

```
Generating File List
Reading data from files
Putting Data into dataframes
```

In [2]:

```
tags_df = pd.read_json('data/tags_train.json')
```

In [10]:

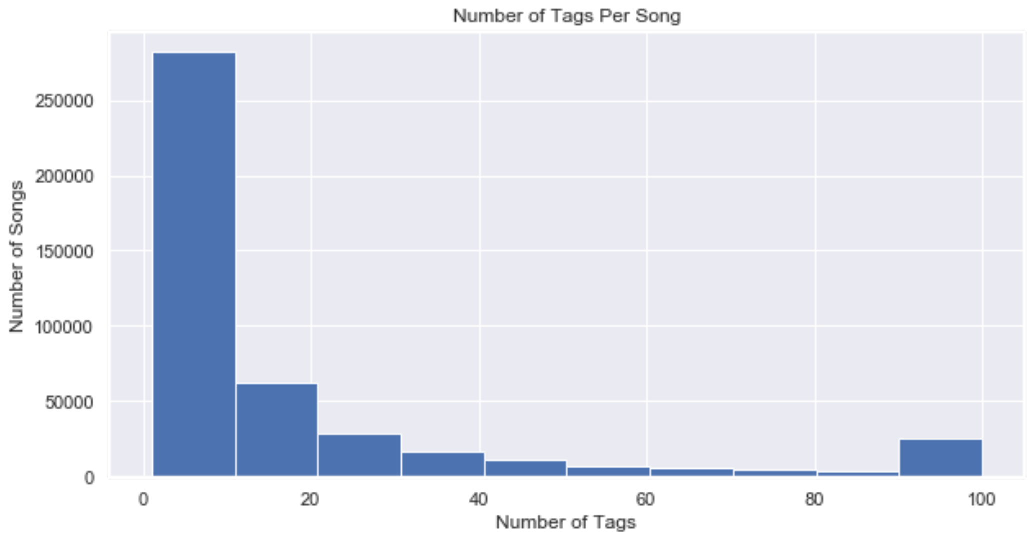
```
display(tags_df.head())
display(tags_df.shape)
```

	track_id	tag	strength
0	TRAAAAK128F9318786	alternative rock	100
1	TRAAAAK128F9318786	rock	60
10	TRAAAAW128F429D538	hieroglyiphics	100
100	TRAAAED128E0783FAB	jazz vocal 2	1
1000	TRAABVM128F92CA9DC	love	22

```
(7671133, 3)
```

In [4]:

```
tags_per_song = tags_df['track_id'].value_counts()
fig, ax = plt.subplots(figsize=(10,5))
ax.hist(tags_per_song)
ax.set_title('Number of Tags Per Song')
ax.set_ylabel('Number of Songs')
ax.set_xlabel('Number of Tags')
plt.show()
```



In [ ]:

```
songs_df = pd.read_json('data/songs_train.json')
```

In [9]:

```
display(songs_df.head())
display(songs_df.shape)
```

	artist	timestamp	track_id	title
0	Adelitas Way	2011-08-15 09:59:32.436152	TRAAAAK128F9318786	Scream
1	Western Addiction	2011-08-12 13:00:44.771968	TRAAAAV128F421A322	A Poor Recipe For Civic Cohesion
10	Son Kite	2011-08-10 19:36:13.851544	TRAAAEM128F93347B9	Game & Watch
100	Lost Immigrants	2011-08-02 09:37:00.958971	TRAACEI128F930C60E	Memories & Rust
1000	The Irish Tenors	2011-08-03 03:38:35.708526	TRAATLC12903D0172B	Mountains Of Mourne

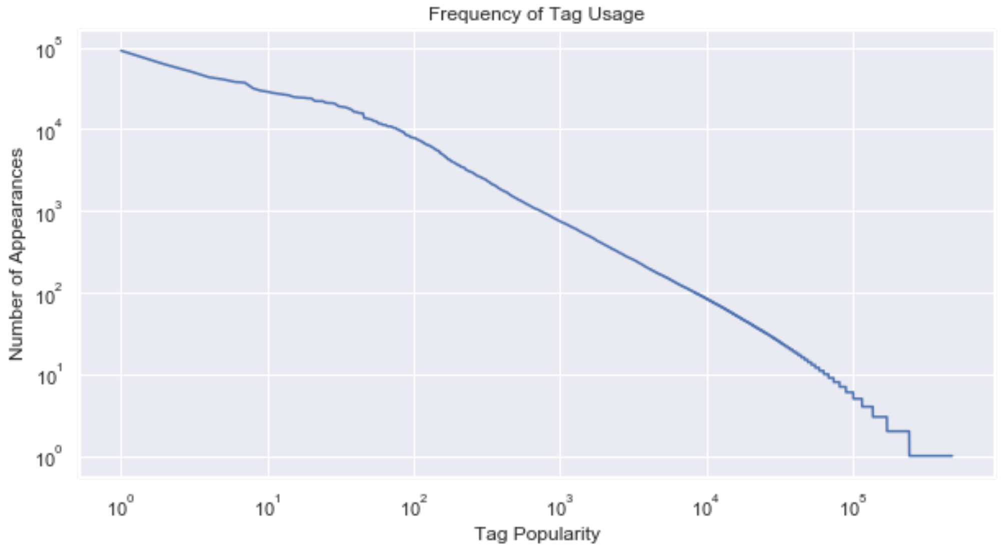
```
(839122, 4)
```

In [ ]:

```
#not enough ram to execute, need to find alternate solution.
#Dataframe is loaded after running the process_data function.
similars_df = pd.read_json('data/similars_train.json')
display(similars_df.head())
```

In [3]:

```
fig, ax = plt.subplots(figsize=(10,5))
tag_counts = tags_df['tag'].value_counts()
tag_counts_gt5 = tag_counts[tag_counts>5]
ax.plot(np.arange(tag_counts.shape[0])+1, tag_counts)
ax.set_yscale('log')
ax.set_xscale('log')
ax.set_title('Frequency of Tag Usage')
ax.set_xlabel('Tag Popularity')
ax.set_ylabel('Number of Appearances')
# ax[0].plot(np.arange(tag_counts.shape[0])+1, tag_counts)
# ax[0].set_yscale('log')
# ax[0].set_xscale('log')
# ax[0].set_title('Frequency of Tag Usage')
# ax[0].set_xlabel('Tag Popularity')
# ax[0].set_ylabel('Number of Appearances')
# ax[1].plot(np.arange(tag_counts_gt5.shape[0]), tag_counts_gt5)
# ax[1].set_yscale('log')
# ax[1].set_xscale('log')
# ax[1].set_title('Frequency of Tag Usage')
# ax[1].set_xlabel('Tag Popularity')
# ax[1].set_ylabel('Number of Appearances')
plt.show()
```



In [5]:

```
data = [[i, w] for i, w in tag_counts[0:10].items()]
```

```
In [7]: pd.DataFrame(data, columns=['Tag', 'Usage Count'], index=np.arange(1,11))
```

Out[7]:

	Tag	Usage Count
1	rock	91222
2	pop	61775
3	alternative	50568
4	indie	43037
5	electronic	40525
6	female vocalists	37804
7	favorites	37029
8	love	31497
9	dance	29495
10	00s	28699

Below is the code that was used to scrape the Spotify Song information from the last.fm API. It makes use of the pylast and spotipy libraries.

In [ ]:

```
tags_list = []
songs_list = []
similars_list = []
missed_count = 0
track_not_found = []
with gzip.open('data/lastfm/songs.csv.gz', 'wt') as fs:
    writer_s = csv.writer(fs, delimiter=',')
    writer_s.writerow(df_songs.columns.tolist()+['Listeners'])
    with gzip.open('data/lastfm/tags.csv.gz', 'wt') as ft:
        writer_t = csv.writer(ft, delimiter=',')
        writer_t.writerow(['id', 'tag', 'weight'])
        with gzip.open('data/lastfm/similars.csv.gz', 'wt') as f:
            writer = csv.writer(f, delimiter=',')
            writer.writerow(['id', 'id_similar', 'similarity'])
            for pidpos_id in df_key['pidpos_id'].values:
            #     artist = df_artists['name'].iloc[df_key['artist_id'].iloc[pid
pos_id]]
            #     song = df2_songs['name'].iloc[df_key['track_id'].iloc[pidpos_
id]]
                spotify_id = df2_songs['uri'].iloc[df_key['track_id'].iloc[pidp
os_id]]
                song_id = df2_songs['id'].iloc[df_key['track_id'].iloc[pidpos_i
d]]
                try:
                    sp_track = sp.track(spotify_id)
                except:
                    token = util.prompt_for_user_token(sp_user,'user-library-re
ad', client_id, client_secret, callback_url)
                    sp = spotipy.Spotify(auth=token)
                    sp_track = sp.track(spotify_id)
                artist = sp_track['album']['artists'][0]['name']
                song = sp_track['name']
                track = last.get_track(artist, song)
                try:
                    top_tags = track.get_top_tags()
                    for top_tag in top_tags:
                        if np.int(top_tag.weight) >= 50:
                            #tags_list.append([spotify_id, top_tag.item.get_nam
e(), top_tag.weight])
                            writer_t.writerow([song_id, top_tag.item.get_name()
, top_tag.weight])
                    #print(top_tag.item.get_name(), top_tag.weight, track.get_l
istener_count())
                    #songs_list.append([spotify_id, artist, song, track.get_lis
tener_count()])
                    writer_s.writerow(df_songs.iloc[song_id].values.tolist()+[t
rack.get_listener_count()])
                    similars = track.get_similar()
                    for similar in similars:
                        try:
                            if similar.match >= .5:
                                match_name = cleanString(similar.item.get_name(
))
                                match_id = df2_songs[df2_songs['name']==match_n
ame].id.values[0]
                                #similars_list.append([spotify_id, match_id, si
milar.match])
                                writer.writerow([song_id, match_id, similar.mat
ch])
                        except:
                            missed_count += 1
                except Exception as e:
    #                 print('Track '+song+' by '+artist+' not found')
```