Song Recommender

Steve Bachmeier
2019-01-22

```
# Code to run for report

#---------------------------------------------------------------
# Run the following two lines to hide the In[] and Out[] margin.
# Doing so will not allow headings to be collapsed.

from IPython.core.display import display,HTML
display(HTML('<style>.prompt{width: 0px; min-width: 0px; visibility: collapse}</style>'))

#---------------------------------------------------------------
# Run the following lines

# Import libraries
import dill
import pandas as pd

# Load
df_playlists_train = dill.load(open("df_playlists_train.pkl", "rb"))
playlist_song_mat_train = dill.load(open("playlist_song_mat_train.pkl", "rb"))
```

1 Synopsis

A basic song recommender system was built using a 1.8+ million row dataset that was scraped in late 2010 / early 2011. The user inputs a specific song and the program suggestions three songs. These recommendations are the songs that show up the most in all playlists that include the song of interest.

A working example of the recommender is below:

```
# ---- EXAMPLE ----

# Ask for song or band
print('\n')
song_band_inquiry = input('Name a song or band: ').lower()

# Show options
options = pd.DataFrame([col for col in playlist_song_mat_train.columns if song_band_inqui
ry in col])[0]
print('\n')
print('Suggestions:')
print(options)

# Obtain user input
while True:
    print('\n')
    options
    try:
        song_band_number = int(input('What number? '))
    except:
        print('\n')
        print('*** Must input an integer. ***')
        continue
    else:
        if (song_band_number in range(0,len(options))):
            song_band_choice = options[song_band_number]
            break
        else:
            print('\n')
            print('*** Choose a number from the table. ***')
            continue

#=============================================================================
#
# RECOMMEND
#
#=============================================================================
print('\n')
print('Top 3 suggested songs: ')
for x_song_band, x_count in \
    playlist_song_mat_train[playlist_song_mat_train[song_band_choice]==1]\
        .sum().sort_values(ascending=False)[1:4,].iteritems():
    print(' * ', x_song_band)
```

```
Name a song or band: Maroon 5


Suggestions:
0                          give a little more [maroon 5]
1              happy christmas (war is over) [maroon 5]
2                           harder to breathe [maroon 5]
3     if i never see your face again [maroon 5 & rih...
4                             makes me wonder [maroon 5]
5                                     misery [maroon 5]
6                 never gonna leave this bed [maroon 5]
7                           she will be loved [maroon 5]
8                             sunday morning [maroon 5]
9                                  this love [maroon 5]
10                              wake up call [maroon 5]
11                 won't go home without you [maroon 5]
Name: 0, dtype: object


What number? 6


Top 3 suggested songs:
 *  for the first time [the script]
 *  rolling in the deep [adele]
 *  grenade [bruno mars]
```

2 Overview

2.1 Background

Recommender systems are one of the most common applications I try and describe when asked just what a data scientist does. From Netflix's movie suggestions to targeted advertising, it often seems like technology knows what we want even before we do. I thought it would be fun to try and build such a system. For this project, I used playlist data to offer song suggestions to a user-input song.

2.2 Data

I was surprised to find that there is not a lot of widely available and open song/playlist data. I initially thought I could use Spotify's Million Song Playlist competition data, but it unfortunately is not yet available for non-competitors. Kaggle has a very large 90+ million row set that I started using, but it is primarily Russian and I do not have the computer resources to translate such a large dataset (although Google's translator package worked admirably during my brief testing).

I settled on a moderately large (~11000 playlists consisting of ~1.88 million songs) dataset that was scraped in late 2010/early 2011. The raw data was collected by Shuo Chen from Cornell University's Department of Computer Science. Specifically, playlist and tag data was scraped from Yes.com and Last.fm, respectively; Yes.com and Last.fm thus owns the data.

Dataset location: https://www.cs.cornell.edu/~shuochen/lme/data_page.html (https://www.cs.cornell.edu/~shuochen/lme/data_page.html)

References

[1] Shuo Chen, Joshua L. Moore, Douglas Turnbull, Thorsten Joachims, Playlist Prediction via Metric Embedding, ACM Conference on Knowledge Discovery and Data Mining (KDD), 2012.

[2] Joshua L. Moore, Shuo Chen, Thorsten Joachims, Douglas Turnbull, Learning to Embed Songs and Tags for Playlists Prediction, International Society for Music Information Retrieval (ISMIR), 2012.

[3] Shuo Chen, Jiexun Xu, Thorsten Joachims, Multi-space Probabilistic Sequence Modeling, ACM Conference on Knowledge Discovery and Data Mining (KDD), 2013.

2.3 Goal

The goal of this project is to make song recommendations based on a single user-input song.

Note that the approach taken is very simple and can certainly be improved upon; I am more interested in the thought process that goes into such applications rather than a super robust and accurate model.

3 Model creation

This section outlines the analysis completed. Refer to Appendix A1 for relevant code.

3.1 Data preparation

The raw dataset is a simple .csv file where the first two rows are useless for this analysis, the third consists of unique playlist IDs, and the rest include song IDs for each playlist. These song IDs are tied to their respective song/band information via the song_hash.txt file.

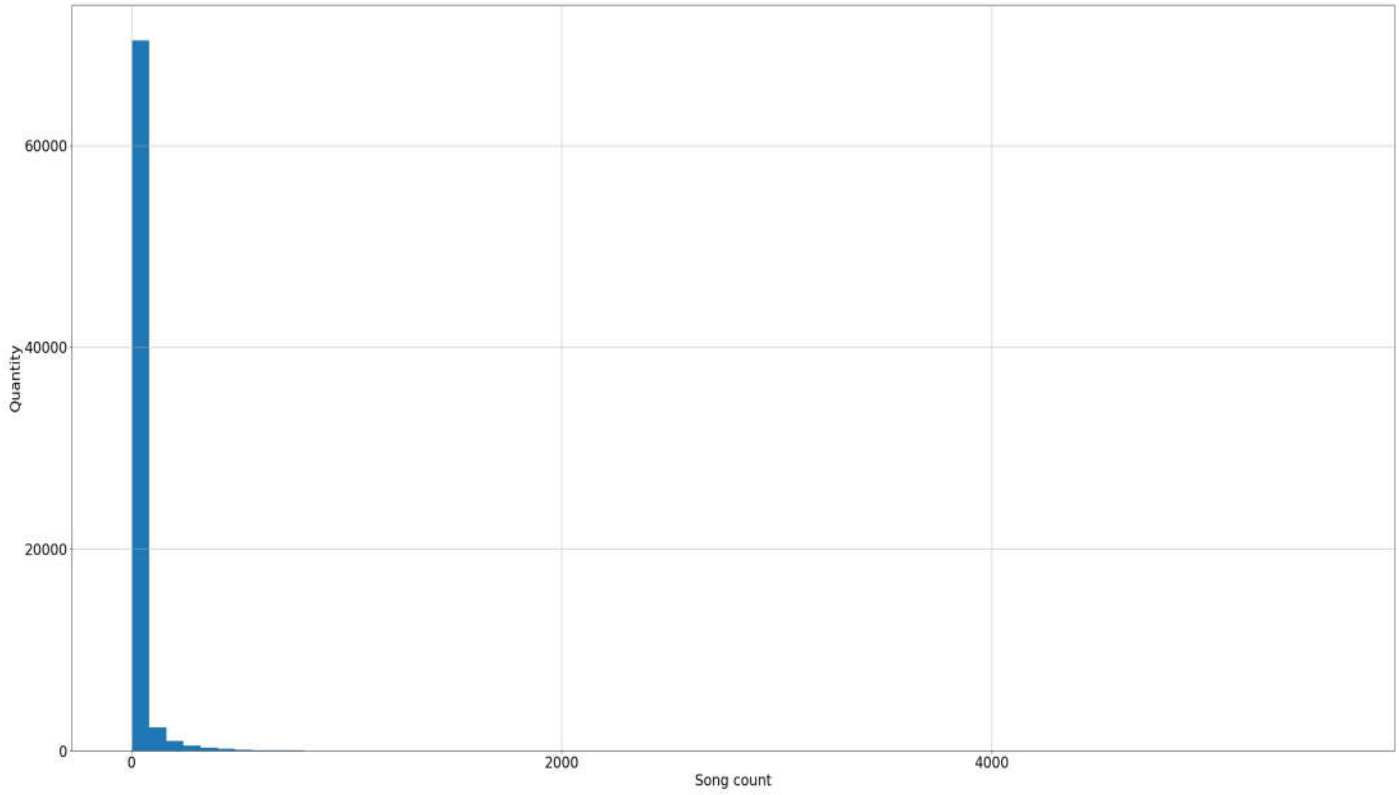
A summary of the data preparation process is below:

1. Read in the raw data.
2. Split the each row's values, convert to integer, and compile into a list.
3. Flatmap the playlists list and convert to a dataframe.
4. Insert a primary key column of index numbers.
5. Merge the song_hash.txt file onto the dataframe.
6. Reset the index (since the merge method changes the order of things).

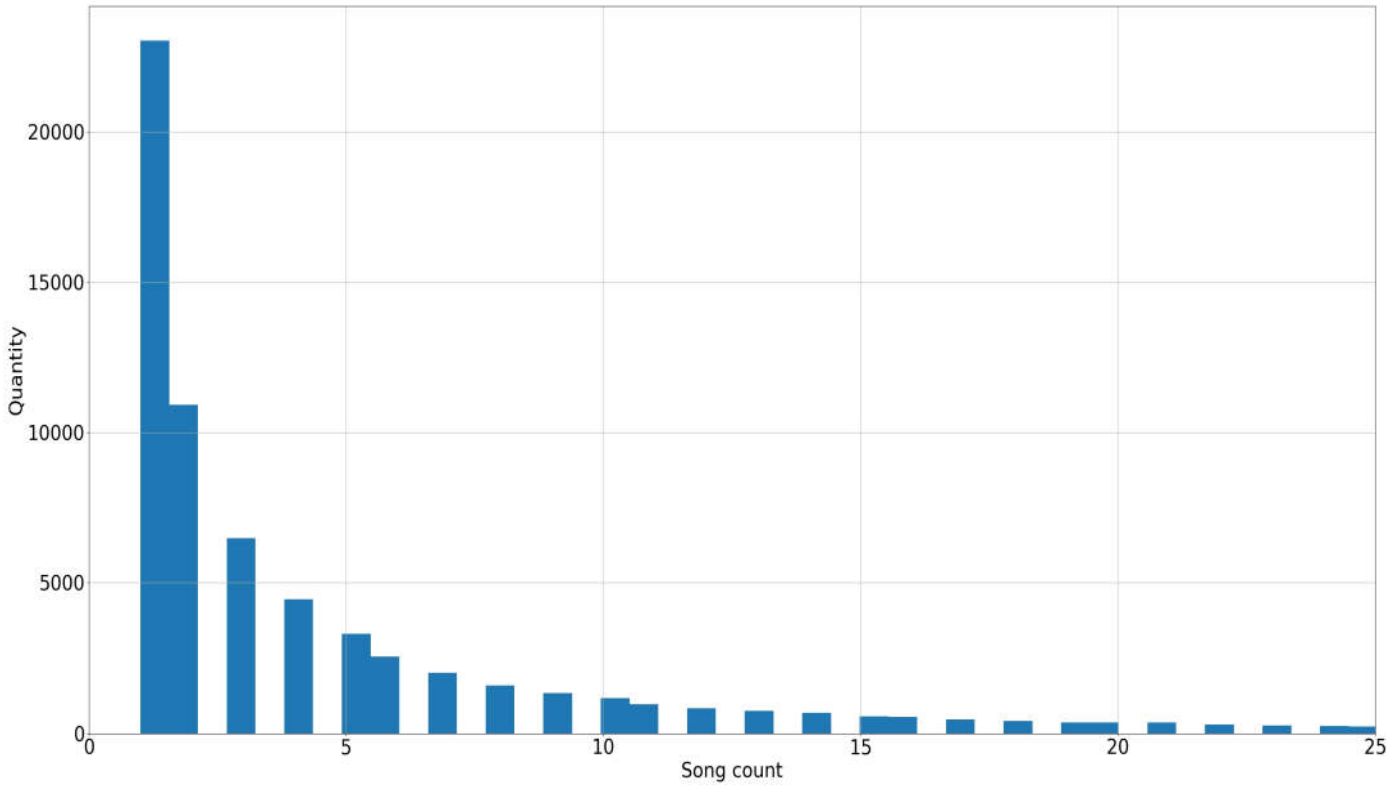
At this point we have a tidy dataframe consisting of 1,887,938 rows and 5 columns (index, playlist, song_id, song, and band).

3.2 Exploratory data analysis

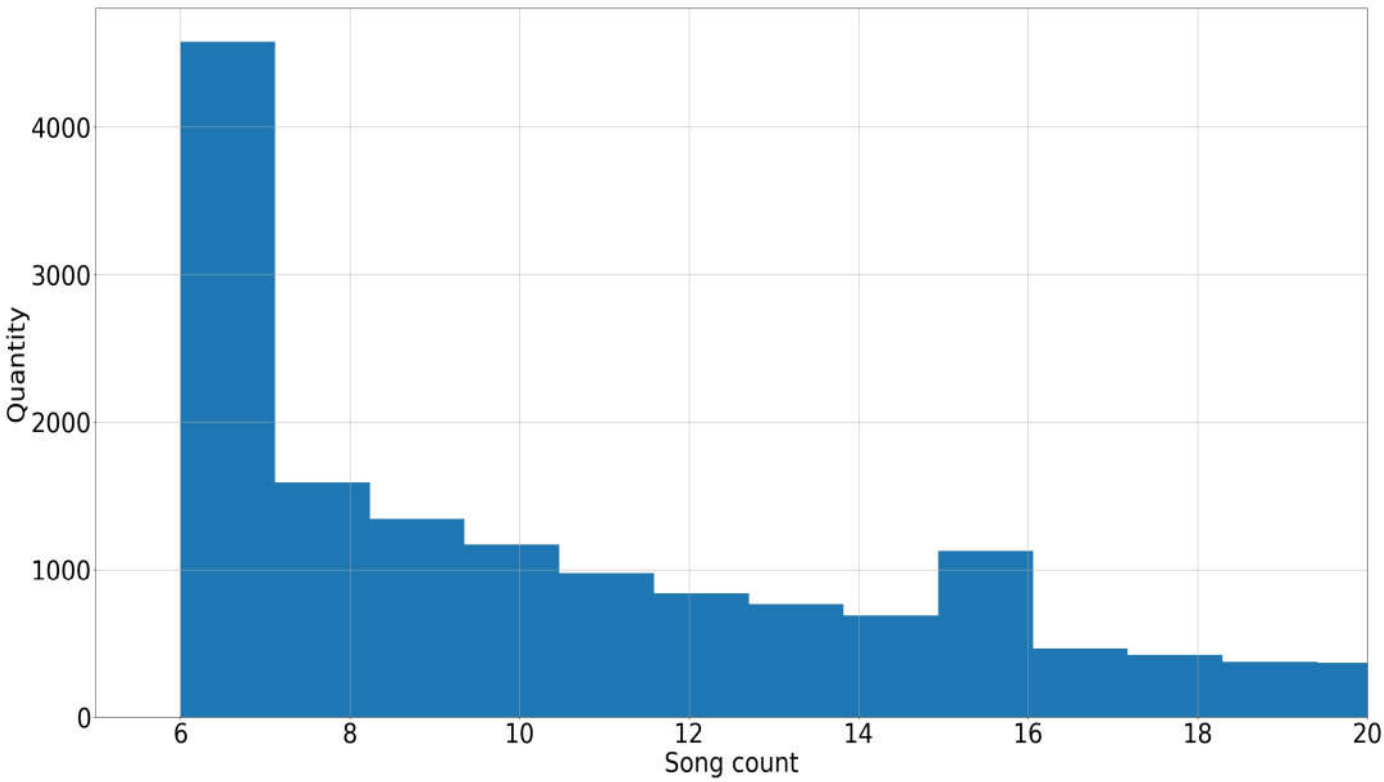
It is always a good idea to do at least a bit of exploratory data analysis; creating visualizations can uncover interesting trends and also help guide further analysis. We can start by looking at the distribution of songs as shown by the histogram below.



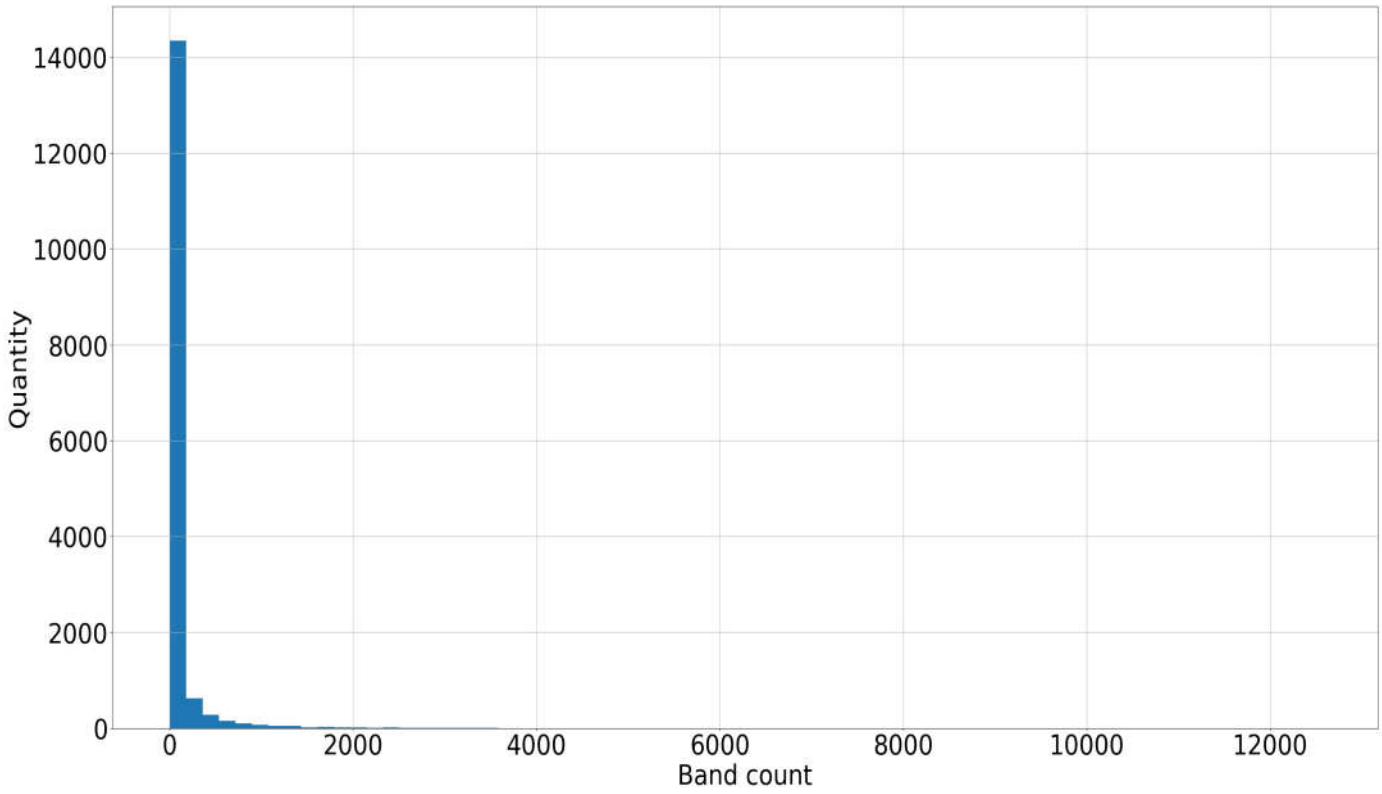
There is a huge spike at lower song counts which means that there are many unique songs that show up a small number of times in the dataset. The histogram below shows the same data but zoomed into song counts between 1-25.

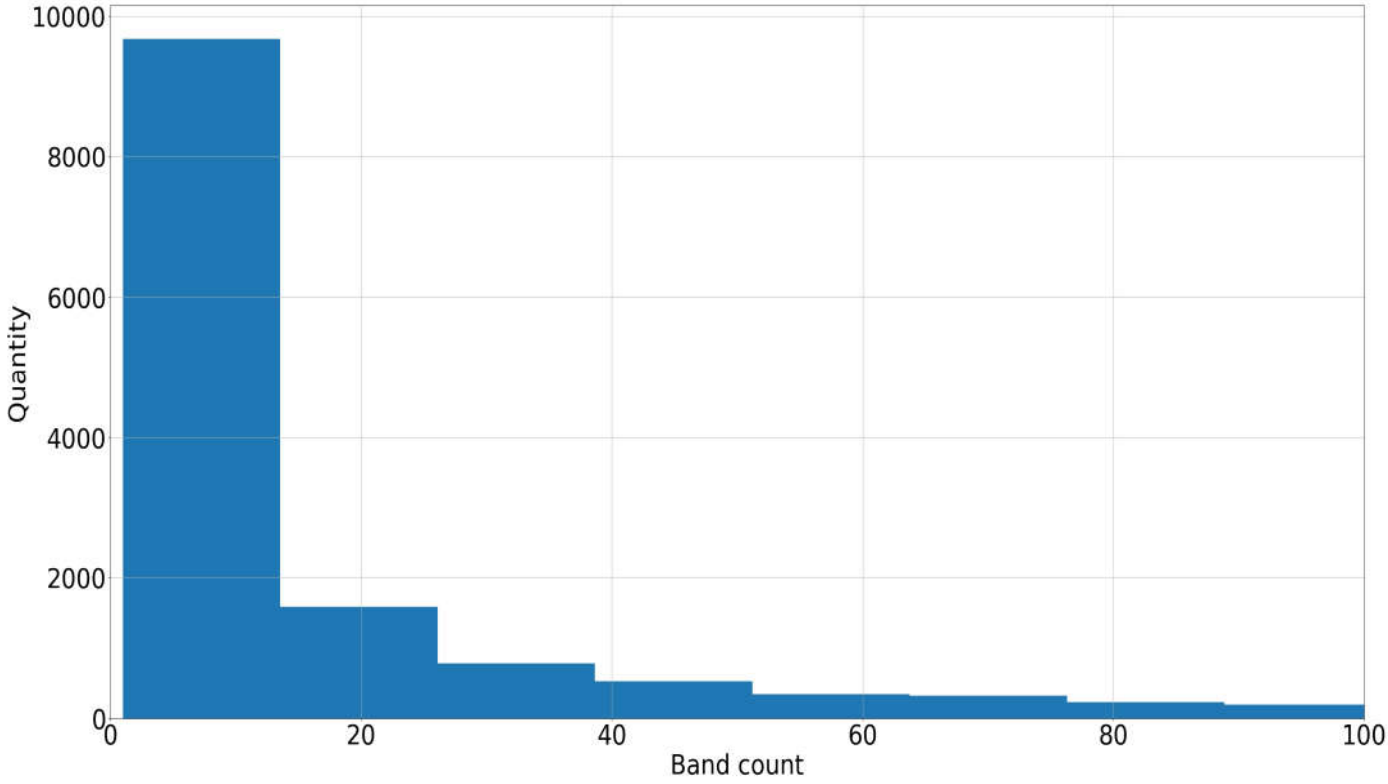


The plot above shows that there are a particularly large amount of songs (quantity) that show up five or less times in the dataset. A final song count histogram but zoomed into song counts between 5-20 is shown below.



A similar set of histograms can be created for unique bands instead of songs. These are shown below.


14000
12000
10000
8000
6000
4000
2000
0
Quantity
0
2000
4000
6000
8000
10000
12000
Band count



3.3 Data cleaning

The following steps are taken to clean the data:

1. Create and merge a song count vector onto the dataframe.
2. Pare the data by removing any unique songs that show up less than six times in the dataset.
3. Check for any null and NA values (there are none).
4. Check for and remove any songs and/or bands labeled "-".
5. Check for and remove any songs and/or bands labeled "" or " ".
6. Remove any escape characters "\".
7. Replace any square brackets "[ ]" with parentheses "( )".
8. Lower the case of song and band labels.
9. Add a new column "song_band" and concatenate the song and band strings like "song [band]".

At this point, the tidy and clean dataset looks like:

```
df_playlists_train.head()
```

	index	playlist	song_id	song	band	song_band	count
0	0	0	0	gucci time (w/ swizz beatz)	gucci mane	gucci time (w/ swizz beatz) [gucci mane]	138
1	1	0	1	aston martin music (w/ drake & chrisette miche...	rick ross	aston martin music (w/ drake & chrisette miche...	2833
2	2	0	2	get back up (w/ chris brown)	t.i.	get back up (w/ chris brown) [t.i.]	297
3	3	0	3	hot toddy (w/ jay-z & ester dean)	usher	hot toddy (w/ jay-z & ester dean) [usher]	502
4	4	0	4	whip my hair	willow	whip my hair [willow]	700

3.4 Build the recommender system

The first step in building the actual song recommender is to create a playlist-song matrix where each row represents a specific playlist and each column is a dummy variable for each unique song. The values of each cell is then binary (either 0 or 1) and represents whether or not a specific song shows up in a specific playlist. Note that this is a sparse matrix.

First, the song dummy variables are created using Pandas' *get_dummies()* method. Then, index and playlist number columns are inserted. Once this matrix is created, it is converted to a dask dataframe which allows for parallel computing. Doing this, in addition to deleting dataframes that are no longer required, clears up memory errors I was initially getting on my personal laptop.

The song-playlist dataframe is then grouped by playlist with the cells consisting of the aggregate *sum()* function. All non-zero sums are in turn converted to ones (I didn't want to over-weight songs that mistakenly are included multiple times in the same playlist(s)).

The final step is then to take a user-input song and filter the song-playlist dataframe to only return playlists that include that song. The *sum()* function is finally applied to this filtered dataset across every column to provide sums of every song in that reduced dataframe (note that most are still zero). The recommendations are thus the songs that show up the most among all playlists that contain the song of interest!

4 Recommendation / final validation

As explained in the previous section, the song recommendations are simply the songs that show up the most among all playlists that contain the song of interest. No attempt has been made to increase accuracy of recommendations by, say, analyzing the accompanying tags datasets.

Indeed, at this point in time, no attempt has been made to validate the recommender on the available test dataset. An outline of one approach to validate the program is outlined here:

1. Use all or a sample of playlists from the test set.
2. From the chosen test set data, extract all or a sample of unique songs.
3. Run the selected songs through the recommender.
4. For each selected song, compare the recommendations to the actual songs that exist in the test playlists.

    - As an example, the accuracy of a single recommendation to *song i* might be:

$$accuracy_i = \frac{count(song_i)}{count(songs)}\bigg|_{playlist_i}$$

Even without having validated it, the recommender does seem to work reasonably well. If I look for suggestions based on a rock and roll song such as, for example, "Sweet Child o' Mine" by Guns n' Roses, I indeed get rock and roll suggestions:

```
# ---- EXAMPLE ----

# Ask for song or band
print('\n')
song_band_inquiry = input('Name a song or band: ').lower()

# Show options
options = pd.DataFrame([col for col in playlist_song_mat_train.columns if song_band_inqui
ry in col])[0]
print('\n')
print('Suggestions:')
print(options)

# Obtain user input
while True:
    print('\n')
    options
    try:
        song_band_number = int(input('What number? '))
    except:
        print('\n')
        print('*** Must input an integer. ***')
        continue
    else:
        if (song_band_number in range(0,len(options))):
            song_band_choice = options[song_band_number]
            break
        else:
            print('\n')
            print('*** Choose a number from the table. ***')
            continue

#=============================================================================
#
# RECOMMEND
#
#=============================================================================
print('\n')
print('Top 3 suggested songs: ')
for x_song_band, x_count in \
    playlist_song_mat_train[playlist_song_mat_train[song_band_choice]==1]\
        .sum().sort_values(ascending=False)[1:4,].iteritems():
    print(' * ', x_song_band)
```

```
Name a song or band: sweet child o' mine


Suggestions:
0    sweet child o' mine [guns n' roses]
Name: 0, dtype: object


What number? 0


Top 3 suggested songs:
 *  welcome to the jungle [guns n' roses]
 *  sweet emotion [aerosmith]
 *  dream on [aerosmith]
```

On the other hand, if I look for recommendations based on a rap song like "Boyz-n-the-Hood" By Eazy-E the program offers raps songs:

```
# ---- EXAMPLE ----

# Ask for song or band
print('\n')
song_band_inquiry = input('Name a song or band: ').lower()

# Show options
options = pd.DataFrame([col for col in playlist_song_mat_train.columns if song_band_inqui
ry in col])[0]
print('\n')
print('Suggestions:')
print(options)

# Obtain user input
while True:
    print('\n')
    options
    try:
        song_band_number = int(input('What number? '))
    except:
        print('\n')
        print('*** Must input an integer. ***')
        continue
    else:
        if (song_band_number in range(0,len(options))):
            song_band_choice = options[song_band_number]
            break
        else:
            print('\n')
            print('*** Choose a number from the table. ***')
            continue

#==============================================================================
#
# RECOMMEND
#
#==============================================================================
print('\n')
print('Top 3 suggested songs: ')
for x_song_band, x_count in \
    playlist_song_mat_train[playlist_song_mat_train[song_band_choice]==1]\
        .sum().sort_values(ascending=False)[1:4,].iteritems():
    print(' * ', x_song_band)
```

```
Name a song or band: boyz-n


Suggestions:
0    boyz-n-the hood [dynamite hack]
1           boyz-n-the hood [eazy-e]
Name: 0, dtype: object


What number? 1


Top 3 suggested songs:
 *  deuces [chris brown & tyga]
 *  black and yellow [wiz khalifa]
 *  what's my name (w/ drake) [rihanna]
```

5 Next steps

As mentioned above, this analysis is very simple and there is ample room for improvement should it be desired. Some recommendations to improve this analysis include:

- Validate the model. Refer to the above section for an outline of how this might be done.
- Analyze the effects of filtering out different song count values.
- Analyze and use the accompanying tags datasets to increase recommendation accuracy.
- Further clean the text data to better deal with mis-spellings, special characters, etc.
- Find and incorporate genre data to more heavily weight songs of the same genre.
- Find and incorporate ratings data to more heavily weight songs with higher ratings.

Appendix

A1 Code

```
#=============================================================================
#
# IMPORT LIBRARIES
#
#=============================================================================
import time
import pandas as pd
#pd.options.display.max_columns = None # Shows all columns
import string as str
import numpy as np
import matplotlib.pyplot as plt
import dill
import dask
import dask.dataframe as dd
import re
from pprint import pprint

#=============================================================================
#
# READ/EXTRACT RAW DATA
#
#=============================================================================

file_train = 'data/dataset/yes_complete/train.txt'

with open(file_train) as f:
    train_raw = f.readlines()

playlists_train_raw = train_raw[2:]

# How many unique playlists?
print('There are ', len(playlists_train_raw), ' playlists.', sep='')

# Split and convert to integers
playlists_train_list = []
for i in range(0, len(playlists_train_raw)):
    playlists_train_list.append(list(map(int, playlists_train_raw[i].split())))

# Flatmap the playlists list
df_playlists_train = []
for i in range(0, len(playlists_train_list)):
    for j in range(0, len(playlists_train_list[i])):
        df_playlists_train.append([i, playlists_train_list[i][j]])

df_playlists_train = pd.DataFrame(df_playlists_train, columns=['playlist', 'song_i
d'])
```

```
# Create index primary key
df_playlists_train.insert(loc=0, column='index', value=df_playlists_train.index)

# ---- Merge songs on song_id ----
df_song_hash = pd.read_table('data/dataset/yes_complete/song_hash.txt', header=Non
e, names=['song_id','song','band'])

df_playlists_train = df_playlists_train.merge(df_song_hash, on='song_id')
df_playlists_train.sort_values('index', inplace=True)
df_playlists_train.reset_index(drop=True, inplace=True)

#==============================================================================
#
# EDA
#
#==============================================================================

song_count = df_playlists_train.groupby(['song_id','song','band']).count().reset_in
dex().drop(columns='index')
song_count.columns = ['song_id','song','band','count']
song_count = song_count.sort_values('count', ascending=False)
song_count.reset_index(inplace=True)
song_count = song_count[['song_id','song','band','count']]

%matplotlib qt
plt.figure(figsize=(10,4))
song_count['count'].hist(bins=70)
plt.xlabel('Song count')
plt.ylabel('Quantity')
plt.title('')
plt.rcParams.update({'font.size': 120})

%matplotlib qt
plt.figure(figsize=(10,4))
song_count['count'].hist(bins=10000)
plt.xlabel('Song count')
plt.ylabel('Quantity')
plt.title('')
plt.xlim(0,25)
plt.rcParams.update({'font.size': 10})

%matplotlib qt
plt.figure(figsize=(10,4))
song_count[song_count['count'] > 5]['count'].hist(bins=5000)
plt.xlabel('Song count')
plt.ylabel('Quantity')
plt.title('')
plt.xlim(5,20)
plt.rcParams.update({'font.size': 40})
```

```
band_count = pd.DataFrame(df_playlists_train.groupby('band')['playlist'].count().so
rt_values(ascending=False))
band_count.reset_index(inplace=True)
band_count.columns = ['band','count']

%matplotlib qt
plt.figure(figsize=(10,4))
band_count['count'].hist(bins=70)
plt.xlabel('Band count')
plt.ylabel('Quantity')
plt.title('')
plt.rcParams.update({'font.size': 40})

%matplotlib qt
plt.figure(figsize=(10,4))
band_count['count'].hist(bins=1000)
plt.xlabel('Band count')
plt.ylabel('Quantity')
plt.title('')
plt.xlim(0,100)
plt.rcParams.update({'font.size': 40})

#==============================================================================
#
# CLEAN DATA
#
#==============================================================================

# Add count column
df_playlists_train = pd.merge(df_playlists_train, song_count)
df_playlists_train.sort_values(by='index', inplace=True)
df_playlists_train.reset_index(drop=True, inplace=True)

# ---- PARE DATA ----
# Remove songs that show up less than 6 times
df_playlists_train = df_playlists_train[df_playlists_train['count']>=6]

# ---- CHECK FOR NULLS ----
df_playlists_train.isnull().sum().sum()
df_playlists_train.isna().sum().sum()

# ---- REMOVE "", " ", "-" ----
df_playlists_train[df_playlists_train['song'] == '-'].head()
df_playlists_train.drop(df_playlists_train[df_playlists_train['song']=='-'].index,
 inplace=True)
df_playlists_train[df_playlists_train['song'] == '-']

df_playlists_train[df_playlists_train['band'] == '-'].head()
```

```
df_playlists_train.drop(df_playlists_train[df_playlists_train['band']=='-'].index,
 inplace=True)
df_playlists_train[df_playlists_train['band']=='-']

df_playlists_train[df_playlists_train['song']=='']
df_playlists_train[df_playlists_train['band']=='']
df_playlists_train[df_playlists_train['song']==' ']
df_playlists_train[df_playlists_train['band']==' ']

# ---- REMOVE ESCAPE CHARACTER '\' ----
df_playlists_train['song'] = [x.replace("\\","") for x in df_playlists_train['son
g']]
df_playlists_train['band'] = [x.replace("\\","") for x in df_playlists_train['ban
d']]

# REPLACE [] WITH ()
df_playlists_train['song'] = [x.replace("[","(") for x in df_playlists_train['son
g']]
df_playlists_train['song'] = [x.replace("]",")") for x in df_playlists_train['son
g']]
df_playlists_train['band'] = [x.replace("[","(") for x in df_playlists_train['ban
d']]
df_playlists_train['band'] = [x.replace("]",")") for x in df_playlists_train['ban
d']]

# LOWER SONG AND BAND
df_playlists_train['song'] = [x.lower() for x in df_playlists_train['song']]
df_playlists_train['band'] = [x.lower() for x in df_playlists_train['band']]

# ---- CONCAT SONG AND BAND ----
df_playlists_train['song_band'] = df_playlists_train['song'] + " [" + df_playlists_
train['band'] + "]"

# Rearrange columns
df_playlists_train = df_playlists_train[['index','playlist','song_id','song','ban
d','song_band','count']]

#==============================================================================
#
# BUILD RECOMMENDER
#
#==============================================================================

# ---- CREATE PLAYLIST-SONG MATRIX ----
start_clock = time.clock()

# Create song_band dummies
#playlist_song_mat_train = pd.get_dummies(df_playlists_train['song_band'], sparse=T
rue)
```

```
playlist_song_mat_train = pd.get_dummies(df_playlists_train['song_band'])

# Insert index and playlist columns
playlist_song_mat_train.insert(loc=0, column='playlist', value=df_playlists_train
['playlist'])
playlist_song_mat_train.insert(loc=0, column='index', value=df_playlists_train['ind
ex'])

end_clock = time.clock()

print('\n')
print('Runtime: ', round((end_clock - start_clock)/60, 2), ' min', sep='')

# CONVERT TO DASK DATAFRAME
start_clock = time.clock()
dd_playlist_song_mat_train = dd.from_pandas(data=playlist_song_mat_train, npartitio
ns=1000)
end_clock = time.clock()

print('Runtime: ', round((end_clock - start_clock)/60, 2), ' min')

# CLEAR MEMORY
del(band_count, df_song_hash, playlist_song_mat_train,
    playlists_train_list, playlists_train_raw, song_count, train_raw)

# GROUP BY PLAYLIST
start_clock = time.clock()
playlist_song_mat_train = dd_playlist_song_mat_train.groupby('playlist').sum().comp
ute()
playlist_song_mat_train.drop(columns='index', inplace=True)
end_clock = time.clock()

print('Runtime: ', round((end_clock - start_clock)/60, 2), ' min')

# CLEAR MEMORY
del(dd_playlist_song_mat_train)

# Convert every non-zero count to just 1, ie each playlist should count each unique
 song as appearing only once
# regardless of how often it actually appears.
playlist_song_mat_train = playlist_song_mat_train.astype(bool).astype(int)

# ---- SAVE/LOAD DATAFRAMES ----
import dill
import pandas as pd

# Save out dataframes
# Dump
#dill.dump(df_playlists_train, open("df_playlists_train.pkl", "wb"))
```

```
#dill.dump(playlist_song_mat_train, open("playlist_song_mat_train.pkl", "wb"))

# Load
#df_playlists_train = dill.load(open("df_playlists_train.pkl", "rb"))
#playlist_song_mat_train = dill.load(open("playlist_song_mat_train.pkl", "rb"))

#==============================================================================
#
# RUN
#
#==============================================================================

# ---- SONG OR BAND INQUIRY ----
song_band_inquiry = input('What song or band do you like? ').lower()

options = pd.DataFrame([col for col in playlist_song_mat_train.columns if song_band
_inquiry in col])[0]

while True:
    print('\n')
    options
    try:
        song_band_number = int(input('What number? '))
    except:
        print('\n')
        print('*** Must input an integer. ***')
        continue
    else:
        if (song_band_number in range(0,len(options))):
            song_band_choice = options[song_band_number]
            break
        else:
            print('\n')
            print('*** Choose a number from the table. ***')
            continue

print('Top 3 suggested songs: ')
for x_song_band, x_count in \
    playlist_song_mat_train[playlist_song_mat_train[song_band_choice]==1]\
        .sum().sort_values(ascending=False)[1:4,].iteritems():
    print(' * ', x_song_band)
```