CS109a Spotify Playlist Generation

Vladimir Yelizarov
Brandon Joye

Original data

1m playlists along with tracks (songs)/albums/artists which are part of particular playlist. Data is in JSON format, split into 1000 files. After download, we adjusted code provided with the dataset to read, clean and reconcile the data.

Problem

Size. Loaded as-is, this dataset wouldn't fit into memory (~66m records). Names of songs / artists / playlists are duplicated which is a waste. Therefore, preprocessing utility was adjusted to split data into tables, adding ids for each entry: songs, artists, albums, playlists, tracks. Names were sanitized: brought to lower case, removed leading/trailing space, removed non-ASCII letter symbols, skipped entry if resulting string is empty (some names were all-hieroglyphs). This also allowed an easier analysis of tables (i.e. group by names). Even after the data normalization, track data wouldn't fit into a notebook with 8Gb of RAM. That, plus early results of EDA (~70% of track data belongs to playlists with just 1 follower), led to removing track data that has a single follower. Resulting dataframe fits into RAM.

Song details

('danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo', 'time_signature') are being downloaded through Spotify's REST API (using Spotify ID from playlist dataset to reconcile data). It's a slow and laborious process because authentication token must be re-taken every hour (manually) plus there's a rate limit (~4 songs per second). At this point, ~90% data has been downloaded.

After cleaning and preprocessing :

- Songs: 1389690. Not all songs are found in Spotify – hence post-join with playlist tracks entries which have N/A are dropped. Since this is a rare occurrence, we don't expect major impact on quality. Spotify URI is used to fetch song details (danceability, loudness, etc)
- Artists: 281893. Names are used to find match in similarity dataset but otherwise only having "same" artist (same id) seems to be important for playlists
- Albums: 547603. Same as artist: use for matching but otherwise track-album id is used
- Playlists: 984547; after dropping single-follower and top-follower (71643) playlists: 241799
- Tracks: 65138632; after dropping single-follower playlists: 19822736; after joining to song details 18660867 (preliminary - song details download is in-progress)

Playlist names

(17384 unique entries) don't look descriptive: although some provide context (e.g. decade-related '00s' repeated 219 times, '10' – 175; mood: 'zone' - 107 'zumba' – 342, 'zombie' - 25), many names look strange ('zzz' – 218 times). Until more promising modelling is done, no lexical matching is done on names.

Number of playlists by followers

Are heavily skewed by playlists with few followers, details below

Engineered features for playlists

Joining back track data with song details and representing playlist by its own data (number of tracks/albums/followers) and aggregating (depending on the field - e.g max mode) of song data per playlist, gives engineered features of the playlist (all fields range 0-1 for modelling) which allows modelling of playlists – e.g. clustering based on these features.

PCA

It was done on these features: 88% of variation is explained by 4 components. Since number of engineered features is not large, PCA is not used in modelling.

In [1]:
```
import sys
import json
import codecs
import datetime
import numpy as np
import pandas as pd
import string
DATA_DIR="./data/data"
```

In [2]:
```
df = pd.read_csv(DATA_DIR + '/preproc.csv.gz', compression='gzip')
dfAugSongs = pd.read_csv(DATA_DIR + '/full_aug_songs.csv.gz', compression='gzip
')
dfPlaylists = pd.read_csv(DATA_DIR + '/playlists.csv.gz', compression='gzip')
```

In [3]:
```
nfol = dfPlaylists.groupby('num_followers').agg(['count'])
```

In [4]:
```
nfol[nfol > 50].dropna()
```

Out[4]:

	id	name	collaborative	num_tracks	num_albums
	count	count	count	count	count
num_followers					
1	742663.0	742659.0	742663.0	742663.0	742663.0
2	147138.0	147137.0	147138.0	147138.0	147138.0
3	46178.0	46178.0	46178.0	46178.0	46178.0
4	19304.0	19304.0	19304.0	19304.0	19304.0
5	9669.0	9669.0	9669.0	9669.0	9669.0
6	5276.0	5276.0	5276.0	5276.0	5276.0
7	3256.0	3256.0	3256.0	3256.0	3256.0
8	2119.0	2119.0	2119.0	2119.0	2119.0
9	1494.0	1494.0	1494.0	1494.0	1494.0
10	998.0	998.0	998.0	998.0	998.0
11	820.0	820.0	820.0	820.0	820.0
12	626.0	626.0	626.0	626.0	626.0
13	468.0	468.0	468.0	468.0	468.0
14	356.0	356.0	356.0	356.0	356.0
15	325.0	325.0	325.0	325.0	325.0
16	285.0	285.0	285.0	285.0	285.0
17	232.0	232.0	232.0	232.0	232.0
18	206.0	206.0	206.0	206.0	206.0
19	161.0	161.0	161.0	161.0	161.0
20	137.0	137.0	137.0	137.0	137.0
21	117.0	117.0	117.0	117.0	117.0
22	124.0	124.0	124.0	124.0	124.0
23	106.0	106.0	106.0	106.0	106.0
24	95.0	95.0	95.0	95.0	95.0
25	66.0	66.0	66.0	66.0	66.0
26	69.0	69.0	69.0	69.0	69.0
27	73.0	73.0	73.0	73.0	73.0
28	70.0	70.0	70.0	70.0	70.0
29	71.0	71.0	71.0	71.0	71.0
31	64.0	64.0	64.0	64.0	64.0
32	60.0	60.0	60.0	60.0	60.0
36	65.0	65.0	65.0	65.0	65.0

In [5]:
```
dfNumFol = dfPlaylists[dfPlaylists.num_followers > 1] # remove min
dfNumFol = dfNumFol[dfNumFol.num_followers < 71643] # remove max as outlier
dfPlNumFol = pd.merge(df, dfNumFol, left_on='playlist_id', right_on='id', how='
left').dropna()
```

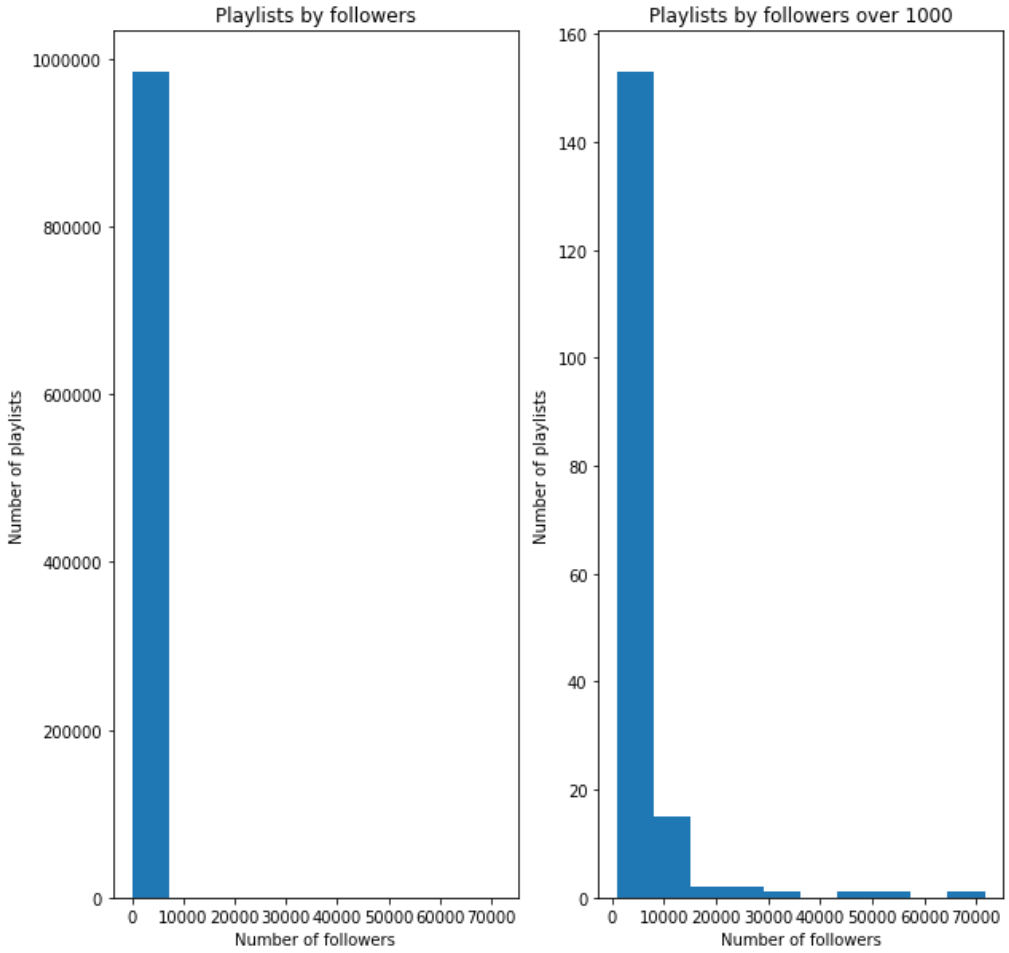
In [6]:
```
dfPlNumFol.drop(['id_y', 'name', 'num_followers', 'collaborative', 'num_tracks'
, 'num_albums'], axis=1, inplace=True)
```

In [7]:
```
dfPlNumFol.rename(columns={'id_x': 'pidpos_id'}, inplace=True)
```

In [8]:
```
dfPlNumFol.to_csv(DATA_DIR + '/pidpos.csv.gz', compression="gzip")
```

In [9]:
```
import matplotlib.pyplot as plt
plt.hist(dfPlaylists.num_followers);
```

In [10]:
```
fig, ax = plt.subplots(1, 2, figsize=(10, 10))
ax[0].hist(dfPlaylists[dfPlaylists.num_followers > 0].num_followers)
ax[0].set_title("Playlists by followers")
ax[0].set_xlabel('Number of followers')
ax[0].set_ylabel('Number of playlists')
ax[1].hist(dfPlaylists[dfPlaylists.num_followers > 1000].num_followers)
ax[1].set_title("Playlists by followers over 1000")
ax[1].set_xlabel('Number of followers')
ax[1].set_ylabel('Number of playlists');
```



In [11]:
```
dfSongs = pd.read_csv(DATA_DIR + '/orig_songs.csv.gz', compression='gzip')
dfSongs.head()
```

Out[11]:

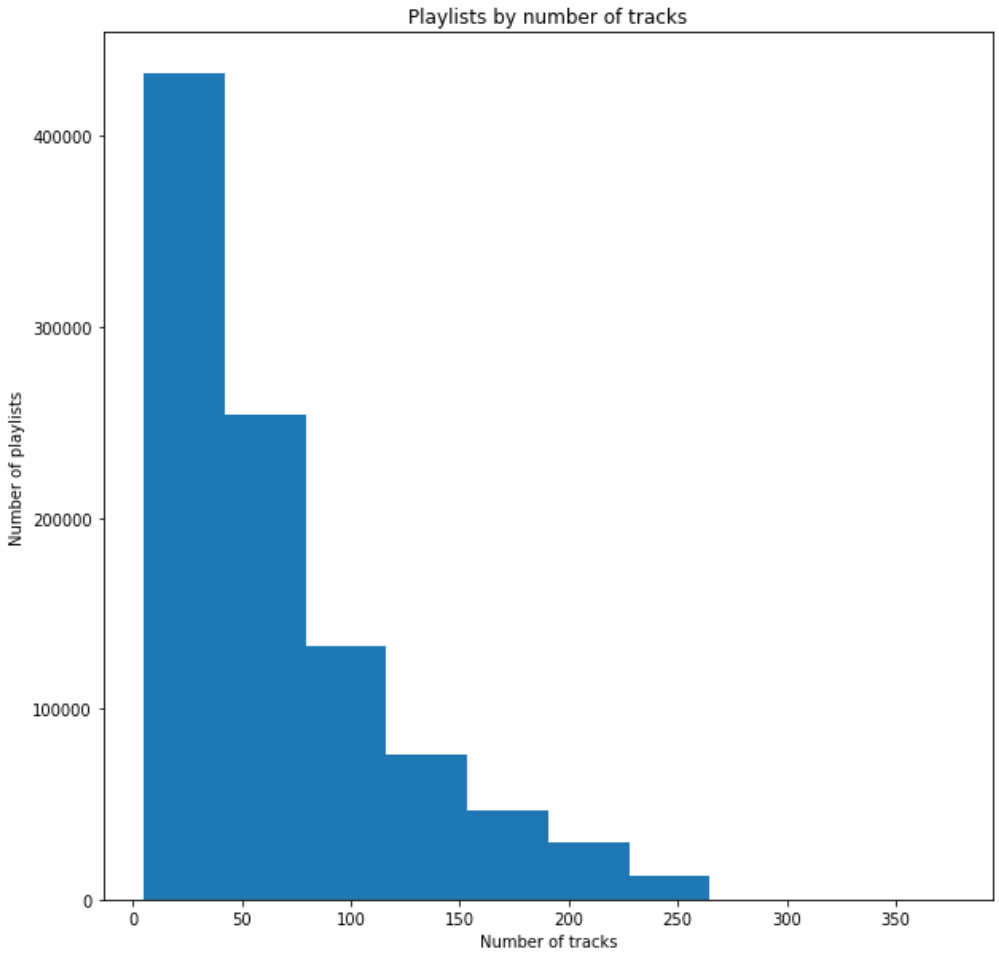
	name	id	uri	duration_ms
0	lose control feat ciara fat man scoop	0	0UaMYEvWZi0ZqiDOoHU3YI	226.863
1	toxic	1	6l9VzXrHxO9rA9A5euc8Ak	198.800
2	crazy in love	2	0WqIKmW4BTrj3eJFmnCKMv	235.933
3	rock your body	3	1AWQoqb9bSvzTjaLralEkT	267.266
4	it wasnt me	4	1lzr43nnXAijIGYnCT8M8H	227.600

In [13]:
```
dfAugSongs.describe()
```

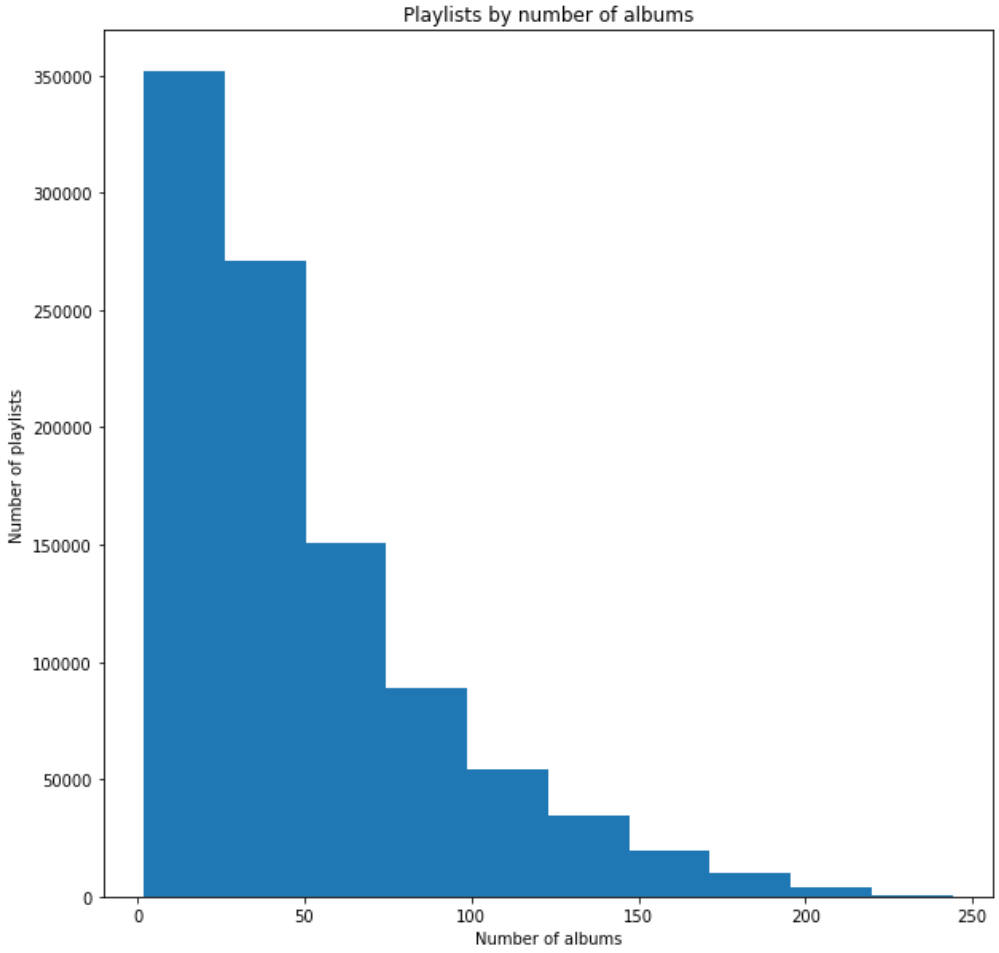
Out[13]:

	id	duration	danceability	energy	key	loudness
count	265512.000000	265512.000000	265512.000000	265512.000000	265512.000000	265512.000000
mean	132317.224099	244.971160	0.564075	0.618639	5.260873	-8.648893
std	76351.805012	128.179798	0.179457	0.249564	3.571320	5.021407
min	0.000000	0.000000	0.000000	0.000000	0.000000	-60.000000
25%	66383.750000	190.862000	0.447000	0.451000	2.000000	-10.486000
50%	132037.500000	227.040000	0.578000	0.660000	5.000000	-7.333500
75%	198417.250000	274.384500	0.697000	0.824000	8.000000	-5.335000
max	264795.000000	5823.660000	0.991000	1.000000	11.000000	3.792000

In [45]:
```
fig, ax = plt.subplots(1, 1, figsize=(10, 10))
ax.hist(dfPlaylists.num_tracks)
ax.set_title("Playlists by number of tracks")
ax.set_xlabel('Number of tracks')
ax.set_ylabel('Number of playlists');
```



In [46]:
```
fig, ax = plt.subplots(1, 1, figsize=(10, 10))
ax.hist(dfPlaylists.num_albums)
ax.set_title("Playlists by number of albums")
ax.set_xlabel('Number of albums')
ax.set_ylabel('Number of playlists');
```



In [16]:
```
dfPlTracks = pd.merge(df, dfNumFol, left_on='playlist_id', right_on='id', how='
left').dropna()
```

In [17]:
```
dfPlTracks.drop(['id_y', 'name', 'collaborative'], axis=1, inplace=True)
dfPlTracks.rename(columns={'id_x': 'pidpos_id'}, inplace=True)
```

In [18]:
```
dfPlTracks.head()
```

Out[18]:

	pidpos	pidpos_id	track_id	album_id	artist_id	playlist_id	num_followers	num_tracks	num_a
281	4-0	281	280	225	175	4	2.0	17.0	16.0
282	4-1	282	281	226	176	4	2.0	17.0	16.0
283	4-2	283	282	227	177	4	2.0	17.0	16.0
284	4-3	284	283	228	178	4	2.0	17.0	16.0
285	4-4	285	284	229	179	4	2.0	17.0	16.0

In [19]:
```
dfPlSongs = pd.merge(dfPlTracks, dfAugSongs, left_on='track_id', right_on='id',
how='left').dropna()
```

In [20]:
```
dfPlSongs.head()
```

Out[20]:

	pidpos	pidpos_id	track_id	album_id	artist_id	playlist_id	num_followers	num_tracks	num_alb
0	4-0	281	280	225	175	4	2.0	17.0	16.0
1	4-1	282	281	226	176	4	2.0	17.0	16.0
2	4-2	283	282	227	177	4	2.0	17.0	16.0
3	4-3	284	283	228	178	4	2.0	17.0	16.0
4	4-4	285	284	229	179	4	2.0	17.0	16.0

5 rows × 25 columns

In [21]:
```
dfPlSongs.columns
```

Out[21]:
```
Index(['pidpos', 'pidpos_id', 'track_id', 'album_id', 'artist_id',
       'playlist_id', 'num_followers', 'num_tracks', 'num_albums', 'name',
       'id', 'uri', 'duration', 'danceability', 'energy', 'key', 'loudness',
       'mode', 'speechiness', 'acousticness', 'instrumentalness', 'liveness',
       'valence', 'tempo', 'time_signature'],
      dtype='object')
```

In [22]:
```
dfPlSongs.drop(['id', 'name', 'uri'], axis=1, inplace=True)
```

In [23]:
```
dfPlGroupped = dfPlSongs.groupby('playlist_id')
```

In [24]:
```
# trust direct count of tracks, not the field - some songs haven't been downloa
ded yet
# key (pitch) is integer hence - max
# loudness, tempo, time_signature: 'max'
# mode: drop (not mean, why max / min ?)
dfPlSongsAgg = dfPlGroupped.agg({'track_id': 'count', 'num_followers' : 'first'
, 'num_tracks' : 'first',
                   'num_albums' : 'first', 'duration': 'mean',
                   'danceability': 'mean', 'energy': 'mean', 'key' : 'max', 'lou
dness': 'max',
       'speechiness': 'mean', 'acousticness': 'mean', 'instrumentalness': 'mean
', 'liveness': 'mean',
       'valence': 'mean', 'tempo': 'max', 'time_signature': 'max'}).rename(
    columns={'track_id': 'sum_num_tracks', 'duration': 'mean_duration', 'dancea
bility': 'mean_danceability', 'energy': 'mean_energy',
             'key' : 'max_key', 'loudness': 'max_loudness',
       'speechiness': 'mean_speechiness', 'acousticness': 'mean_acousticness',
        'instrumentalness': 'mean_instrumentalness', 'liveness': 'mean_liveness
',
       'valence': 'mean_valence', 'tempo': 'max_tempo', 'time_signature': 'max_
time_signature' })
```

In [25]:
```
dfPlSongsAgg.head()
```

Out[25]:

	sum_num_tracks	num_followers	num_tracks	num_albums	mean_duration	mean_dan
playlist_id						
4	17	2.0	17.0	16.0	255.016588	0.576765
8	46	2.0	46.0	37.0	216.280891	0.512370
10	72	2.0	72.0	60.0	229.388917	0.725931
20	14	3.0	14.0	9.0	171.202429	0.824429
22	42	2.0	42.0	39.0	215.148548	0.645238

In [26]:
```
# incomplete song data - still being downloaded
dfPlSongsAgg[dfPlSongsAgg.sum_num_tracks != dfPlSongsAgg.num_tracks].max_key.co
unt()
```

Out[26]: 152227

In [27]:
```
# normalize
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
dfPlSongsAgg[['sum_num_tracks', 'num_tracks', 'num_albums', 'mean_duration', 'm
ax_key',
              'max_loudness', 'max_tempo', 'max_time_signature']] = scaler.fit_
transform(
    dfPlSongsAgg[['sum_num_tracks', 'num_tracks', 'num_albums', 'mean_duration'
,
                                        'max_key', 'max_loudness', 'max_tempo',
'max_time_signature']])
```

In [28]:
```
dfPlSongsAgg.head()
```

Out[28]:

	sum_num_tracks	num_followers	num_tracks	num_albums	mean_duration	mean_dan
playlist_id						
4	0.053872	2.0	0.048980	0.058333	0.043293	0.576765
8	0.151515	2.0	0.167347	0.145833	0.034563	0.512370
10	0.239057	2.0	0.273469	0.241667	0.037517	0.725931
20	0.043771	3.0	0.036735	0.029167	0.024404	0.824429
22	0.138047	2.0	0.151020	0.154167	0.034308	0.645238

In [29]:
```
dfPlSongsAgg.to_csv(DATA_DIR + '/aug_playlists.csv.gz', compression="gzip")
```

In [30]:
```
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(dfPlSongsAgg)
plSongsAgg_pca = pca.transform(dfPlSongsAgg)
```

In [31]:
```
plSongsAgg_pca
```

Out[31]:
```
array([[-5.26570221e+00, -4.00077050e-01, -5.97823891e-02, ...,
        -1.70195438e-02, -6.43652413e-03,  1.99573931e-03],
       [-5.26570176e+00, -2.13333439e-01, -7.34344757e-02, ...,
         1.95680783e-02, -6.91630848e-03, -6.18612150e-03],
       [-5.26570178e+00, -4.07861078e-02, -9.54723793e-02, ...,
         1.60570187e-02, -1.48854729e-02,  1.30105156e-03],
       ...,
       [-5.26569821e+00,  6.07251470e-01,  3.06131730e-01, ...,
        -1.13039556e-02,  3.52395122e-03,  3.75963882e-04],
       [-5.26570271e+00, -3.33009279e-01, -1.32031598e-01, ...,
         2.01985984e-02, -6.51052815e-02, -4.22438075e-03],
       [-5.26570097e+00, -2.69565443e-01,  2.24656246e-01, ...,
        -6.06234178e-02, -2.66207575e-02, -1.07814352e-03]])
```

In [32]:
```
sum_variance, component_count = 0, 0
while sum_variance < 0.85:
    sum_variance += pca.explained_variance_ratio_[component_count]
    component_count += 1

print('Number of Principal Components that explain >=85% of Variance: ', compon
ent_count)
print('Total Variance Explained by '+str(component_count)+' components:', str(s
um_variance*100)+'%')
```

```
Number of Principal Components that explain >=85% of Variance:  1
Total Variance Explained by 1 components: 99.99956072623758%
```

In [33]:
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=10, random_state=0).fit(dfPlSongsAgg)
```

In [34]:
```
kmeans.labels_
```

Out[34]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int32)
```

In [35]:
```
kmeans.cluster_centers_
```

Out[35]:
```
array([[2.56485005e-01, 3.51142654e+00, 3.15235927e-01, 2.51979118e-01,
        3.87335284e-02, 6.12080067e-01, 6.39715216e-01, 9.90645744e-01,
        8.49586489e-01, 9.45291764e-02, 2.34634695e-01, 5.79217407e-02,
        1.86836720e-01, 4.91607804e-01, 7.81176972e-01, 8.92775347e-01],
       [1.24579125e-01, 2.28010000e+04, 1.53061224e-01, 1.62500000e-01,
        3.47872841e-02, 5.90602953e-01, 6.29248811e-01, 1.00000000e+00,
        8.12799885e-01, 9.14648482e-02, 2.27811302e-01, 5.63848588e-02,
        2.03426128e-01, 5.54462961e-01, 7.54142088e-01, 8.00000000e-01],
       [2.27425773e-01, 1.06494545e+04, 3.23191095e-01, 1.33333333e-01,
        4.20223397e-02, 6.10645694e-01, 5.95658106e-01, 1.00000000e+00,
        8.31586737e-01, 9.37783396e-02, 2.64282487e-01, 1.20797576e-01,
        1.90890294e-01, 4.67734203e-01, 7.69939985e-01, 8.90909091e-01],
       [2.99663300e-01, 4.97305000e+04, 4.22448980e-01, 3.81250000e-01,
        4.03486842e-02, 5.50774350e-01, 6.23402677e-01, 1.00000000e+00,
        8.50707789e-01, 6.69327002e-02, 2.63845752e-01, 9.10937069e-02,
        1.77731158e-01, 4.85033631e-01, 7.83157445e-01, 8.00000000e-01],
       [2.37143363e-01, 1.81252632e+03, 3.05649839e-01, 2.24517544e-01,
        3.92306167e-02, 5.88960689e-01, 6.21113691e-01, 9.79904306e-01,
        8.40717451e-01, 8.15314535e-02, 2.81929259e-01, 6.45956108e-02,
        1.87169997e-01, 4.83522168e-01, 7.68919505e-01, 8.98947368e-01],
       [3.53054353e-01, 6.87428571e+03, 4.54227405e-01, 2.90178571e-01,
        4.01591818e-02, 5.77118000e-01, 6.38399155e-01, 9.93506494e-01,
        8.56750683e-01, 9.36443491e-02, 2.38562335e-01, 1.06845725e-01,
        1.92557294e-01, 4.62526088e-01, 8.13007205e-01, 9.42857143e-01],
       [2.01178451e-01, 1.46597500e+04, 2.37755102e-01, 1.45833333e-01,
        3.87751388e-02, 6.35418933e-01, 6.81821100e-01, 9.77272727e-01,
        8.39088348e-01, 1.10400991e-01, 1.73343638e-01, 2.55005509e-02,
        1.84883420e-01, 4.85866106e-01, 8.13477446e-01, 9.00000000e-01],
       [3.15151515e-01, 3.78733333e+03, 4.14557823e-01, 3.14444444e-01,
        3.93125253e-02, 5.85017415e-01, 6.36766049e-01, 9.93939394e-01,
        8.52098701e-01, 8.72829991e-02, 2.50847350e-01, 7.80740039e-02,
        1.87838387e-01, 4.49326894e-01, 7.92581023e-01, 9.06666667e-01],
       [2.52500312e-01, 5.44522222e+02, 3.41360544e-01, 2.37438272e-01,
        3.84172366e-02, 5.90198411e-01, 6.20607206e-01, 9.82828283e-01,
        8.45342189e-01, 9.06914819e-02, 2.68934431e-01, 8.30649808e-02,
        1.86253047e-01, 4.69576657e-01, 7.75703930e-01, 8.94074074e-01],
       [2.03703704e-01, 2.96845000e+04, 3.18367347e-01, 1.66666667e-01,
        2.95165475e-02, 5.94047619e-01, 5.85061596e-01, 1.00000000e+00,
        8.34379700e-01, 7.68080247e-02, 3.44960353e-01, 4.67019029e-02,
        1.88245635e-01, 4.70625309e-01, 7.46683426e-01, 9.00000000e-01]])
```

In [36]:
```
kmeans.predict([[0.053872, 0.000000, 0.048980, 0.058333, 0.444, 0.576765, 0.650
535, 0.818182, 0.812418,
                0.041159, 0.177148, 0.081875, 0.166524, 0.99, 0.723059, 0.8
]])
```

Out[36]:
```
array([0], dtype=int32)
```

In [47]:
```
dfPlSongsAgg.num_tracks.count()
```

Out[47]:
```
241798
```

In [38]:
```
groupedPlNames = dfPlaylists.groupby('name', sort=False).count()
groupedPlNames.sort_index(ascending=False)
# little sense in playlist names
```

Out[38]:

name	id	num_followers	collaborative	num_tracks	num_albums
zzzzzzzzzz	6	6	6	6	6
zzzzzzzzz	13	13	13	13	13
zzzzzzzz	18	18	18	18	18
zzzzzzz	34	34	34	34	34
zzzzzz	55	55	55	55	55
zzzzz	57	57	57	57	57
zzzz	80	80	80	80	80
zzz	218	218	218	218	218
zz top	5	5	5	5	5
zz	24	24	24	24	24
zyzz	5	5	5	5	5
zydeco	54	54	54	54	54
zumba	342	342	342	342	342
zucchero	1	1	1	1	1
zouk	45	45	45	45	45
zoom	14	14	14	14	14
zoo	16	16	16	16	16
zoned	27	27	27	27	27
zone out	76	76	76	76	76
zone	107	107	107	107	107
zona ganjah	1	1	1	1	1
zomboy	4	4	4	4	4
zombies	64	64	64	64	64
zombie	25	25	25	25	25
zoe	56	56	56	56	56
zo	1	1	1	1	1
zion	36	36	36	36	36
zik	1	1	1	1	1
ziggy	12	12	12	12	12
zhu	6	6	6	6	6
...	...	...	...	...	...
10	175	175	175	175	175
1 playlist	29	29	29	29	29
1 5	1	1	1	1	1

In [42]:
```
df.pidpos.count()
```

Out[42]: 65138632

In [43]:
```
dfPlTracks.pidpos.count()
```

Out[43]: 19822697

In [44]:
```
dfPlSongs.pidpos.count()
```

Out[44]: 18660828

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
```

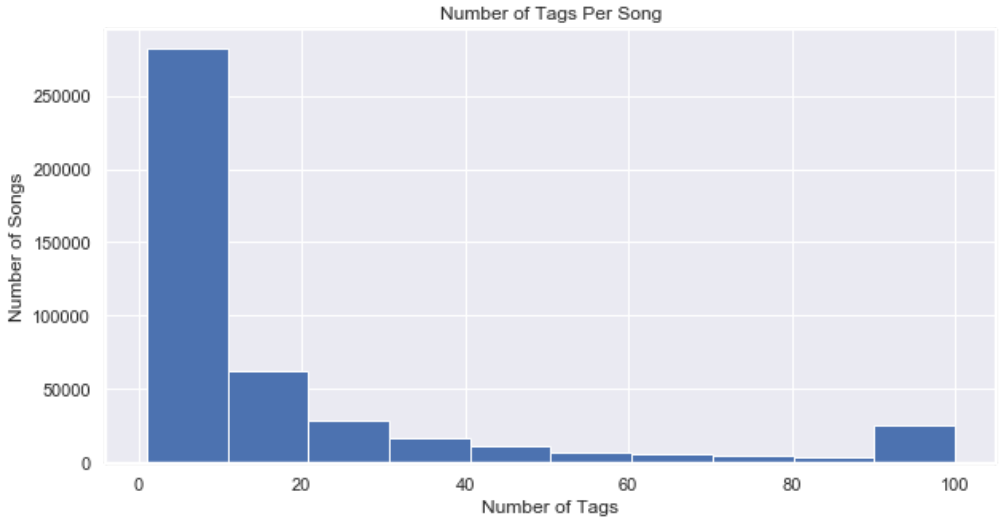
In [2]:
```
tags_df = pd.read_json('data/tags_train.json')
```

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')
```

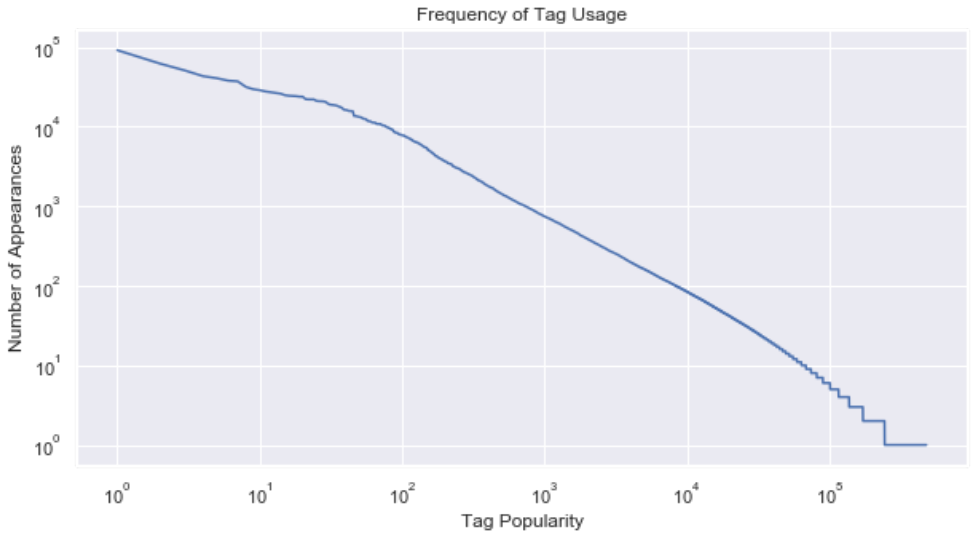
In [9]:
```
display(songs_df.head())
display(songs_df.shape)
```

	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')
```

title: Data Procurement And Processing

nav_include: 3

Data procurement

To get data that describes songs ("audio features" in Spotify), we queried Spotify API for song identifier (Spotify id). We tried to fetch song similarity data from Spotify but that was taking too much time and hence wasn't added to our dataset. Overall, we used compressed csv files to store intermediate and final data sets.

Data processing

- All names (playlists names, song names) where cleaned by removing punctuation, extra spaces, etc
- Songs in playlists processed to find co-occurences of songs in playlists. Our take is that song co-occurence in playlist shows signal for song similarity (of course, Spotify promoted songs and "hits" add noise). If songs are found in the same list - that adds a count and this pair is saved into resulting set.

Sources below. These are Python scripts, not Jupyter notebooks to decrease memory pressure and let scripts run unattended.

In [1]:
```
# Preprocessing:
#!/usr/bin/env python
# coding: utf-8

import sys
import json
import codecs
import datetime
import numpy as np
import pandas as pd
import re
import time
import gzip
import csv

DATA_DIR = "./data/data/"
pretty = True
compact = False
cache = {}

def cleanString(s):
    s = re.sub(r'[^\w\s]','', s) # remove punctuation
    s = re.sub(' +', ' ', s) # remove double spaces
    s = "".join(i for i in s if ord(i)<128) # remove non-ascii letters
    return s.lower().strip()

def getId(dict, str) :
    s = cleanString(str)
    if len(s) == 0 :
        return -1
    if s in dict :
        id = dict[s]
    else :
        id = len(dict)
        dict[s] = id
    return id

def getSongId(dict, track, s_id) :
    s = cleanString(track['track_name'])
    if len(s) == 0 :
        return (-1, s_id)
    if s in dict :
        return (dict[s][0], s_id)
    id = s_id
    s_id += 1
    dict[s] = [ id, track['track_uri'].split(':')[2], float(track['duration_ms'
]) / 1000 ]
    return (id, s_id)

def getPlaylId(dict, playlist) :
    s = cleanString(playlist['name'])
    if len(s) == 0 :
        return -1
    id = len(dict)
    dict[id] = [ s, int(playlist['num_followers']), 1 if bool(playlist['collabo
rative']) else 0,
                 int(playlist['num_tracks']), int(playlist['num_albums']) ]
    return id

def full_playlist(start, end, showOnly=True, namesOnly=False, data=None, playlD
ata=None, lastIndex=0):
    playlists = None
    prevFile = None
```

In [ ]:
```
# Song info fetch. Several files were stored and merged to account for transmis
sion failures, etc.
import urllib
import urllib.request as request
import json
import time
import gzip
import csv
import sys

regSleep = 1.0 / 165;
authH = {'Accept': 'application/json',
    'User-agent': 'Mozilla/5.0',
    "Content-Type": "application/json",
    'Authorization' : 'Bearer {}'.format(
    sys.argv[3]
    ) }

startId = sys.argv[2]
searchFirst = True
DATA_DIR = "./data/data/"
with gzip.open(DATA_DIR + sys.argv[1] + "_aug_songs_" + (startId if startId is
not None else "") + ".csv.gz", 'wt') as fz:
    writer = csv.writer(fz, delimiter=',')
    if startId is None :
        writer.writerow(['name', 'id', 'uri', 'duration', 'danceability', 'ener
gy', 'key', 'loudness', 'mode',
               'speechiness', 'acousticness', 'instrumentalness', 'liveness', '
valence', 'tempo', 'time_signature'])
    with gzip.open(DATA_DIR + "songs.csv.gz", mode="rt") as f:
        print("file", f)
        csvobj = csv.reader(f, delimiter=',', quotechar="'")
        first = True
        fullStop = False
        for line in csvobj:
            if first :
                first = False
                continue
            if len(line) == 0 :
                continue
            id = line[2]
            print("->", id)
            if startId is not None and searchFirst:
               if startId == id :
                   print("found", startId)
                   searchFirst = False
               else:
                   continue
            for retry in range(0, 10):
                code = -1
                try :
                    req = request.Request(url = "https://api.spotify.com/v1/aud
io-features/{}".format(id), headers=authH)
                    resp = request.urlopen(req)
                    content = resp.read()
                    resp = json.loads(content)
                    out = [line[0], line[1], line[2], line[3], resp['danceabili
ty'], resp['energy'], resp['key'], resp['loudness'], resp['mode'],
                           resp['speechiness'], resp['acousticness'], resp['ins
trumentalness'], resp['liveness'], resp['valence'], resp['tempo'],
                           resp['time_signature'] ]
                    writer.writerow(out)
                    time.sleep(regSleep)
```

In [ ]:
```
# Song similarity - done in chunks to fit into memory
import gzip
import csv
import sys
import numpy as np

def calcPlayl(plId, songs) :
    n = len(songs)
    if n == 0 or n == 1:
        return
    print (plId, "len", n)
    r = range(0, n)
    for i in r :
        if matrix[songs[i], 0] is None :
            matrix[songs[i], 0] = {}
        for j in range(i + 1, n) :
            otherId = songs[j]
            if otherId in matrix[songs[i], 0] :
                matrix[songs[i], 0][otherId] += 1
            else :
                matrix[songs[i], 0][otherId] = 1


DATA_DIR = "./data/data/"
numSongs = 1389689 + 1 # max song id , starts from 0
matrix = np.empty([numSongs, 1], dtype=np.dtype('O'))

cnt = 0
MAX = 233000
runId = int(sys.argv[1])
startPl = runId * MAX
endPl = startPl + MAX
with gzip.open(DATA_DIR + "preproc.csv.gz", mode="rt") as f:
    print("file", f)
    csvobj = csv.reader(f, delimiter=',', quotechar="'")
    first = True
    prevPl = -1
    songs = []
    for line in csvobj:
        if first :
            first = False
            continue
        if len(line) == 0 :
            continue
        songId = int(line[2])
        plId = int(line[5])
        if plId < startPl :
            continue
        if plId > endPl :
            break
        if plId != prevPl :
            calcPlayl(prevPl, songs)
            songs = []
        songs.append(songId)
        prevPl = plId

print("done from ", startPl, "to", endPl)
with gzip.open(DATA_DIR + str(runId) + "_simsong_calc_.csv.gz", 'wt') as fz:
    writer = csv.writer(fz, delimiter=',')
    writer.writerow(['songid', 'simsongid', 'count'])
    for i in range(0, matrix.shape[0]) :
        if matrix[i, 0] is None :
            continue
```

In [ ]:
```
# Song similarity merge
import gzip
import csv
import sys

def empty(line1) :
    return line1 is None or len(line1) == 0

def allEmpty(dones) :
    for done in dones :
       if not done:
           return False
    return True

def writeSims(writer, songId, simSongs):
    for key, value in sorted(simSongs.items(), key=lambda kv: kv[1], reverse=Tr
ue) :
        # cutoff ?
        writer.writerow([songId, key, value])

#return false if song ids don't match
def addSongInfo(songId, line, simSongs):
    # print("add", songId, line, simSongs)
    newSongId = int(line[0])
    if songId == newSongId :
        simId = int(line[1])
        cnt = int(line[2])
        if simId in simSongs :
            simSongs[simId] += cnt
        else :
            simSongs[simId] = cnt
        return True
    return False

#return empty list if end of file is reached, next-song line otherwise
def fetchSongInfo(songId, prevLine, curIter, simSongs) :
    # print("fetch", songId, prevLine)
    if not empty(prevLine) :
        match = addSongInfo(songId, prevLine, simSongs)
        if not match :
            return prevLine
    while True :
        line = next(curIter, None)
        if empty(line):
            return []
        match = addSongInfo(songId, line, simSongs)
        if not match :
            return line

DATA_DIR = "./data/data/"

numSongs = 1389689 + 1 # max song id , starts from 0
numFiles = 2
files = [None]*numFiles
csvobj = [None]*numFiles
iters = [None]*numFiles
prevLine  = [None]*numFiles
doneFile = [False]*numFiles
simSongs = {}
fCnt = 0
file0 = sys.argv[1]
file1 = sys.argv[2]
with gzip.open(DATA_DIR + file0 + "_" + file1 + "_simsong_calc_csv.gz", 'wt')
```

title: Splitting Data

nav_include: 4

Stratification Goals

There were two goals when splitting the data, the first was to ensure there was an equal distribution of each playlist type in each data set. To acheive this we grouped the data based on the playlist names.

The secondary goal was to try and evenly split the data by number of followers, to ensure that our model saw playlists with all different levels of popularity for each playlist name. To acheive this we split each name group that had at least 100 playlists in it into 10 groups of follower size and then randomly assigned the playlists in each group based on the weights given to the `split_train_test_validate` function.

In [1]:
```
import numpy as np
import pandas as pd
import gzip
import csv

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

import pylast
```

In [ ]:
```
df_key = pd.read_csv('data' + '/pidpos.csv.gz', compression='gzip', index_col=0
)
```

In [3]:
```
df_key.head()
```

Out[3]:

	pidpos	pidpos_id	track_id	album_id	artist_id	playlist_id
281	4-0	281	280	225	175	4
282	4-1	282	281	226	176	4
283	4-2	283	282	227	177	4
284	4-3	284	283	228	178	4
285	4-4	285	284	229	179	4

In [4]:
```
df_key.playlist_id.unique().shape
```

Out[4]: (241878,)

In [5]:
```
playlist_ids = df_key.playlist_id.unique()
song_ids = df_key.track_id.unique()
album_ids = df_key.album_id.unique()
artist_ids = df_key.artist_id.unique()
```

In [6]:
```
df_songs = pd.read_csv('data' + '/full_aug_songs.csv.gz', compression='gzip')
```

In [7]:
```
df_songs.head()
```

Out[7]:

	name	id	uri	duration	danceability	energy	key	loudness	mode	spee
0	lose control feat ciara fat man scoop	0	0UaMYEvWZi0ZqiDOoHU3YI	226.863	0.904	0.813	4	-7.105	0	0.12
1	toxic	1	6l9VzXrHxO9rA9A5euc8Ak	198.800	0.774	0.838	5	-3.914	0	0.11
2	crazy in love	2	0WqIKmW4BTrj3eJFmnCKMv	235.933	0.664	0.758	2	-6.583	0	0.21
3	rock your body	3	1AWQoqb9bSvzTjaLralEkT	267.266	0.891	0.714	4	-6.055	0	0.14
4	it wasnt me	4	1lzr43nnXAijIGYnCT8M8H	227.600	0.853	0.606	0	-4.596	1	0.07

In [8]:
```
df2_songs = pd.read_csv('data/'+'songs.csv.gz', compression='gzip')
```

In [9]:
```
df2_songs.head()
```

Out[9]:

	name	id	uri	duration_ms
0	lose control feat ciara fat man scoop	0	0UaMYEvWZi0ZqiDOoHU3YI	226.863
1	toxic	1	6l9VzXrHxO9rA9A5euc8Ak	198.800
2	crazy in love	2	0WqIKmW4BTrj3eJFmnCKMv	235.933
3	rock your body	3	1AWQoqb9bSvzTjaLralEkT	267.266
4	it wasnt me	4	1lzr43nnXAijIGYnCT8M8H	227.600

In [10]:
```
df_artists = pd.read_csv('data/'+'artists.csv.gz', compression='gzip')
```

In [11]:
```
df_artists.head()
```

Out[11]:

	name	id
0	missy elliott	0
1	britney spears	1
2	beyonc	2
3	justin timberlake	3
4	shaggy	4

In [12]:
```
df_playlists = pd.read_csv('data/'+'playlists.csv.gz', compression='gzip')
```

In [13]:
```
df_playlists.head()
```

Out[13]:

	id	name	num_followers	collaborative	num_tracks	num_albums
0	0	throwbacks	1	1	52	47
1	1	awesome playlist	1	1	39	23
2	2	korean	1	1	64	51
3	3	mat	1	1	126	107
4	4	90s	2	1	17	16

In [14]:
```
df_playlists.describe()
```

Out[14]:

	id	num_followers	collaborative	num_tracks	num_albums
count	984547.000000	984547.000000	984547.0	984547.000000	984547.000000
mean	492273.000000	2.612579	1.0	66.388307	49.626047
std	284214.382075	129.852211	0.0	53.685494	39.993167
min	0.000000	1.000000	1.0	5.000000	2.000000
25%	246136.500000	1.000000	1.0	26.000000	20.000000
50%	492273.000000	1.000000	1.0	49.000000	37.000000
75%	738409.500000	1.000000	1.0	92.000000	68.000000
max	984546.000000	71643.000000	1.0	376.000000	244.000000

In [15]:
```
df_playlists_trim = df_playlists.iloc[playlist_ids]
```

In [16]:
```
df_playlists_trim.head()
```

Out[16]:

	id	name	num_followers	collaborative	num_tracks	num_albums
4	4	90s	2	1	17	16
8	8	bop	2	1	46	37
10	10	abby	2	1	72	60
20	20	mixtape	3	1	14	9
22	22	fall 17	2	1	42	39

In [17]:
```
df_playlists_trim.describe()
```

Out[17]:

	id	num_followers	collaborative	num_tracks	num_albums
count	241878.000000	241878.000000	241878.0	241878.000000	241878.000000
mean	492351.319314	7.563855	1.0	82.227152	62.454018
std	284354.961297	261.919131	0.0	58.002575	44.234803
min	4.000000	2.000000	1.0	5.000000	2.000000
25%	245372.500000	2.000000	1.0	36.000000	28.000000
50%	492954.500000	2.000000	1.0	67.000000	51.000000
75%	737872.000000	3.000000	1.0	116.000000	87.000000
max	984540.000000	71643.000000	1.0	250.000000	242.000000

In [18]:
```
df_albums = pd.read_csv('data/'+'albums.csv.gz', compression='gzip')
```

In [19]:
```
df_albums.head()
```

Out[19]:

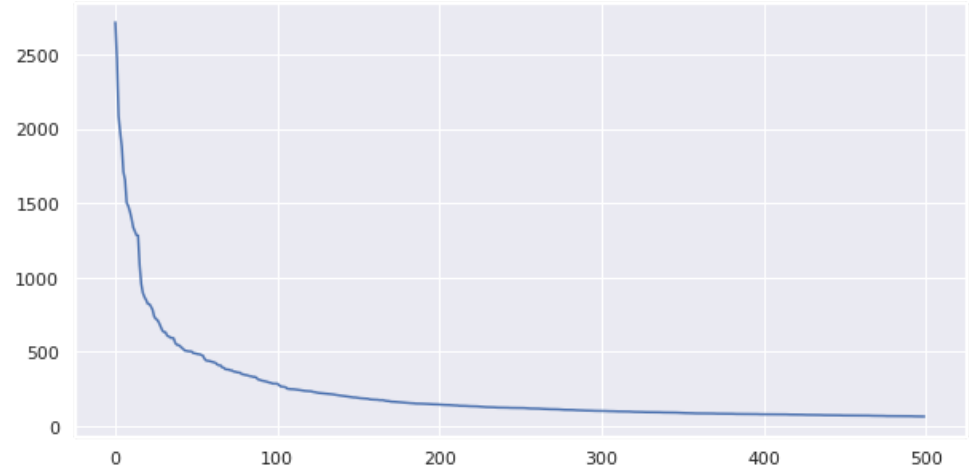
	name	id
0	the cookbook	0
1	in the zone	1
2	dangerously in love alben fr die ewigkeit	2
3	justified	3
4	hot shot	4

In [20]:
```
name_counts = df_playlists_trim.name.value_counts()
```

In [21]:
```
name_counts.shape[0]
```

Out[21]: 14831

In [22]:
```
fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.arange(500), name_counts[0:500])
plt.show()
```



In [76]:
```
name_counts[name_counts>=100].shape
```

Out[76]: (315,)

In [26]:
```
groups = df_playlists_trim.groupby(['name'])
```

In [170]:
```
def divide_df(df, p1, p2, p1_split, p2_split, rest_split):
    split = np.random.randint(1, 11, size=df.shape[0])
    p1_thr = np.int(p1*10)
    p2_thr = np.int((p2+p1)*10)
    p1_split = np.append(p1_split, df.iloc[split <= p1_thr]['id'].values)
    p2_split = np.append(p2_split, df.iloc[(split > p1_thr) & (split <= p2_thr)
][['id'].values)
    rest_split = np.append(rest_split, df.iloc[split > p2_thr]['id'].values)
    return p1_split, p2_split, rest_split
```

In [171]:
```
def split_train_test_validate(df_groupby, test_prop = .1, validate_prop = .1):
    test_id = np.array([])
    train_id = np.array([])
    validate_id = np.array([])
    for name, group in df_groupby:
        if group.shape[0]<100:
            test_id, validate_id, train_id = divide_df(group, test_prop, valida
te_prop, test_id, validate_id, train_id)
        else:
            group2 = group.sort_values(by='num_followers')
            num_groups = np.int(group2.shape[0]/10)
            step_size = np.int(group2.shape[0]/num_groups)
            for step in range(1, num_groups):
                subgroup = group2.iloc[(step-1)*step_size:step*step_size]
                test_id, validate_id, train_id = divide_df(subgroup, test_prop,
validate_prop, test_id, validate_id, train_id)
            subgroup = group2.iloc[step*step_size:]
            test_id, validate_id, train_id = divide_df(subgroup, test_prop, val
idate_prop, test_id, validate_id, train_id)
    return train_id, test_id, validate_id
```

In [172]:
```
train_id, test_id, validate_id = split_train_test_validate(groups)
```

In [175]:
```
train_id.shape[0]/df_playlists_trim.shape[0]
```

Out[175]: 0.8003249572098331

In [176]:
```
test_id.shape[0]/df_playlists_trim.shape[0]
```

Out[176]: 0.10030676622098744

In [177]:
```
validate_id.shape[0]/df_playlists_trim.shape[0]
```

Out[177]: 0.0993682765691795

In [173]:
```
test_id.shape[0]+train_id.shape[0]+validate_id.shape[0], df_playlists_trim.shape[0]
```

Out[173]: (241878, 241878)

In [2]:
```
df_raw_aug = pd.read_csv('data/raw_aug_playlists.csv.gz')
```

In [3]:
```
df_raw_aug.head()
```

Out[3]:

	playlist_id	sum_num_tracks	num_followers	num_tracks	num_albums	mean_duration	mean_da
0	4	17	2.0	17.0	16.0	255.016588	0.57676
1	8	46	2.0	46.0	37.0	216.280891	0.51237
2	10	72	2.0	72.0	60.0	229.388917	0.72593
3	20	14	3.0	14.0	9.0	171.202429	0.82442
4	22	42	2.0	42.0	39.0	215.148548	0.64523

In [30]:
```
df_raw_aug.index = df_raw_aug.playlist_id.values
```

In [31]:
```
df_raw_aug.head()
```

Out[31]:

	playlist_id	sum_num_tracks	num_followers	num_tracks	num_albums	mean_duration	mean_
4	4	17	2.0	17.0	16.0	255.016588	0.5767
8	8	46	2.0	46.0	37.0	216.280891	0.5123
10	10	72	2.0	72.0	60.0	229.388917	0.7259
20	20	14	3.0	14.0	9.0	171.202429	0.8244
22	22	42	2.0	42.0	39.0	215.148548	0.6452

In [6]:
```
train_pl = pd.read_csv('data/train_playlists.csv.gz', index_col=0)
test_pl = pd.read_csv('data/test_playlists.csv.gz', index_col=0)
validate_pl = pd.read_csv('data/validate_playlists.csv.gz', index_col=0)
```

In [8]:
```
train_id = train_pl.id.values
test_id = test_pl.id.values
validate_id = validate_pl.id.values
```

In [ ]:
```
train_aug_pl = df_raw_aug.loc[train_id]
test_aug_pl = df_raw_aug.loc[test_id]
validate_aug_pl = df_raw_aug.loc[validate_id]
```

In [53]:
```
train_aug_pl.to_csv('data/train_aug_playlists.csv.gz', compression='gzip')
test_aug_pl.to_csv('data/test_aug_playlists.csv.gz', compression='gzip')
validate_aug_pl.to_csv('data/validate_aug_playlists.csv.gz', compression='gzip'
)
```

In [26]:
```
df_raw_aug.isnull().values.any()
```

Out[26]:
```
False
```

In [52]:
```
train_aug_pl.isnull().values.any(), test_aug_pl.isnull().values.any(), validate
_aug_pl.isnull().values.any()
```

Out[52]:
```
(False, False, False)
```

In [48]:
```
test_aug_pl[test_aug_pl.isnull().any(axis=1)]
```

Out[48]:

	playlist_id	sum_num_tracks	num_followers	num_tracks	num_albums	mean_duration	me
244514	NaN	NaN	NaN	NaN	NaN	NaN	Na

In [51]:
```
test_aug_pl = test_aug_pl.dropna()
```

In [20]:
```
validate_aug_pl.shape, validate_id.shape
```

Out[20]:
```
((24115, 17), (24115,))
```

In [37]:
```
np.sum(df_raw_aug.playlist_id.isin(test_id).values), test_id.shape
```

Out[37]:
```
(24200, (24201,))
```

title: Modelling

nav_include: 5

Modelling

The goal is to create a playlist, starting from one song, as this could be easily extended to starting from multiple songs. Dataset is split into three parts:

- Training
- Validation
- Test

(please see "Splitting Data" page for details)

We use the following three components for modelling:

- Similar songs lookup (recursive)
- Regression model that estimates number of followers (coefficients calculated based on training dataset)
- KMeans clustering on playlist engineered features to add songs from playlists which belong to the same centroid as the "base" playlist. "Top" similar (belonging to the same cluster, ordered by number of followers, descending) playlists are used to supply songs (chosen at random)

`Metrics`:

- Find which songs generated and original playlists have in common and calculate number of songs which code guessed correctly
- Calculate aggregated (engineered) values from songs to generated playlist and come up with (Euclidean) "distance" between generated playlist and original playlist
- Using regression model that was fit on training data, predict number of followers for generated playlist and calculate how different it is from true num_followers
- Sum these metrics together to find the loss but invert number of song matches to make sure smaller metric is better
- Alternative summary metric("metric2" in Modelling Results): we found out that resulting summary metric has large variance. Majority of this variance comes from number of followers estimation. Metric2 includes only inverted number of song matches plus distance of generated playlist's engineered features to the original playlist. This metric has much lower variance

`Metaparameters`

- Number of clusters. We tried 2 / 10 / 50 / 100 cluster splits
- Number of playlists (5) to choose songs from the same cluster. Refers to taking songs from only 5 playlists which came from closest cluster (ordered by num_followers, descending)
- Number of similar songs (10) to fetch at each step. Similar songs are added recursively (i.e. add 10 songs, go through them to find similar songs for each in order) - until there are enough or none can be taken
- Algorithm fills 50% of playlist from similar songs and 50% from clusters. If there were only a few similar songs for the first phase - similar songs are also taken from the list of songs from clusters

It would be great to run through several metaparam variations but processing time is too high to try that out. Only variation of number of clusters was done. There was not enough time to find out if splitting similar songs / playlists should be done not 50 / 50 but in different proportion. After training and validation, it looks like best number of clusters are 2 and 10. On average, 10 clusters seems to be best.

`Execution`

- It turned out that playlist generation is pretty slow - running through 500 playlists takes an hour. Hence, code splits (after shuffling) the input dataset into "batches" and runs analysis in parallel
- Each run (train / validation / test) saves results into separate compressed csv files - by batch and by cluster size metaparameter

`Programmatically` , code is very similar for train / validation and test. Major differences:

- Regression model for number of followers is trained based on training dataset and applied for all datasets
- Training is limited to 1000 playlists in each of 6 threads calculated in paralllel. This allows validation and training datasets sizes to be roughly equivalent. Below is the script for training.

In [ ]:

```
#!/usr/bin/env python
# coding: utf-

# Training script

import sys
import datetime
import numpy as np
import pandas as pd
import string
from sklearn.cluster import KMeans
from sklearn.linear_model import LinearRegression
import gzip
import csv
from multiprocessing import Process
from sklearn.utils import shuffle

DATA_DIR="./data/data"

df = pd.read_csv(DATA_DIR + '/pidpos.csv.gz', compression='gzip').drop([
'Unnamed: 0'],axis=1)
dfAugSongs = pd.read_csv(DATA_DIR + '/full_aug_songs.csv.gz', compression
='gzip')
dfPlaylists = pd.read_csv(DATA_DIR + '/playlists.csv.gz', compression='gz
ip')
dfTrain = pd.read_csv(DATA_DIR + '/train_aug_playlists.csv.gz', compressi
on='gzip').drop(['Unnamed: 0'],axis=1)
# For validation, added dfValidate = pd.read_csv(DATA_DIR + '/validate_au
g_playlists.csv.gz', compression='gzip').drop(['Unnamed: 0'],axis=1)
# For test: dfTest = pd.read_csv(DATA_DIR + '/test_aug_playlists.csv.gz',
 compression='gzip').drop(['Unnamed: 0'],axis=1)

dfSim = pd.read_csv(DATA_DIR + '/simsong5.csv.gz', compression='gzip').dr
op(['Unnamed: 0'],axis=1)

def addSim(dfSim, cur_set, c_id, num) :
    dfCandidates = dfSim[(dfSim.songid == c_id) | (dfSim.simsongid == c_i
d)]
    t = dfCandidates.sort_values(by='count', ascending=False).values[0:nu
m, :]
    for i in t :
        id = i[1] if i[0] == c_id else i[0]
        if id in cur_set :
            continue
        cur_set.append(id)
    return cur_set

def getPlAgg(candidate_pl) :
     return [ candidate_pl.danceability.mean(), candidate_pl.energy.mean
(),
                    candidate_pl.speechiness.mean(), candidate_pl.acoust
icness.mean(),
                    candidate_pl.instrumentalness.mean(), candidate_pl.l
iveness.mean(),
                    candidate_pl.valence.mean(), candidate_pl.duration.m
ean(), candidate_pl.key.max(),
```

```
                    candidate_pl.loudness.max(),
                    candidate_pl.tempo.max(), candidate_pl.time_signatur
e.max() ]# .iloc[0] -> first

def getNumfXy(dfPlSongsAgg) :
    y = dfPlSongsAgg.num_followers
    X = dfPlSongsAgg[['mean_danceability','mean_energy','mean_speechines
s','mean_acousticness',
                                'mean_instrumentalness', 'mean_liveness',
'mean_valence',
                'mean_duration', 'max_key', 'max_loudness', 'max_tempo',
 'max_time_signature']].values
    return (X, y)

def getNumfRegr(dfPlSongsAgg) :
    (X, y) = getNumfXy(dfPlSongsAgg)
    return LinearRegression().fit(X, y)

def scoreNumfRegr(reg, dfValidate) :
    (X, y) = getNumfXy(dfValidate)
    return reg.score()

def getSongsFrom(dfSongMatch, n) :
    return dfSongMatch.sample(n, random_state=0)['track_id'].values.tolis
t()

def getSimSongs(dfSim, song_set, start_num, from_i, n) :
    size = len(song_set)
    while True :
        if size > n or from_i > (size - 1):
            break
        addSim(dfSim, song_set, song_set[from_i], start_num)
        from_i +=1
    song_set = song_set[0: n]
    assert len(set(song_set)) == len(song_set) # no dups expected
    return song_set

cluster_columns = ['mean_danceability','mean_energy','mean_speechiness',
'mean_acousticness',
                           'mean_instrumentalness', 'mean_liveness', 'mean
_valence',
             'mean_duration', 'max_key', 'max_loudness', 'max_tempo', 'ma
x_time_signature']

dfTrain.dropna(inplace=True)
reg = getNumfRegr(dfTrain) # training
dfTrain=shuffle(dfTrain)
# For validation: dfValidation=shuffle(dfTrain)
# For test: dfTest=shuffle(dfTrain)

# start_num : how many similar songs to fetch on each level
def generate_playlists(dfPlSongsAgg, dfPidPosPl, name, reg, try_clusters,
 try_startNum, song_name) :
    num_top_pl = 5 # choose top 5 playlists
    train_song_id = 0 # first song is used to continue playlist
    toCluster = dfPlSongsAgg[cluster_columns].values
    for n_clusters in try_clusters :
```

```
        kmeans = KMeans(n_clusters=n_clusters, random_state=0, n_jobs = -
1).fit(toCluster)
        print("Found", n_clusters, "clusters")
        train_pl = dfPlSongsAgg.playlist_id.values
        for start_num in try_startNum :
            print("start_num", start_num)
            with gzip.open(DATA_DIR + "/result_" + name + "_" + str(n_clu
sters) + "_" + str(start_num) + ".csv.gz",
                           'wt', newline='') as fz:
                writer = csv.writer(fz, delimiter=',')
                writer.writerow(['playlist_id', 'n_clusters', 'start_num'
, 'metric', 'match', 'distance', 'numf',
                                 'diff'])
                for pl_id in train_pl :
                    print("Running playlist", pl_id)
                    try :
                        train_numf = dfPlSongsAgg[dfPlSongsAgg.playlist_i
d == pl_id].num_followers.values
                        target_n = int(dfPlSongsAgg[dfPlSongsAgg.playlist
_id == pl_id].sum_num_tracks.values)
                        (train_agg_info, _) = getNumfXy(dfPlSongsAgg[dfPl
SongsAgg.playlist_id == pl_id])
                        train_song_set = df[df.playlist_id == pl_id].trac
k_id.values
                        root_name = dfAugSongs[dfAugSongs.id == train_son
g_set[train_song_id]].name.values[0]
                        fromsim_n = int(target_n / 2) # metaparam
                        root_id = dfAugSongs[dfAugSongs.name == root_name
].id.values[0]
                        song_set = [train_song_set[0]]
                        addSim(dfSim, song_set, root_id, start_num)
                        # choose start_num - loop to fromsim_n
                        getSimSongs(dfSim, song_set, start_num, 0, fromsi
m_n)
                        # find song aug data and create playlist
                        candidate_pl = dfAugSongs.iloc[song_set, :]
                        candidate_pl_agg = getPlAgg(candidate_pl)
                        p_label = kmeans.predict([candidate_pl_agg])[0]
                        dfPlaylistMatch = dfPlSongsAgg[kmeans.labels_ ==
p_label]
                        dfPlaylistMatchTop = dfPlaylistMatch.sort_values(
by='num_followers', ascending=False).head(
                            num_top_pl)
                        dfSongMatch = pd.merge(dfPidPosPl, dfPlaylistMatc
hTop, left_on='playlist_id', right_on='playlist_id',
                                               how='left').dropna()
                        from_cluster = target_n - fromsim_n
                        if from_cluster > dfSongMatch.shape[0] : # not en
ough songs
                            song_from_pl = getSongsFrom(dfSongMatch, dfSo
ngMatch.shape[0])
                            getSimSongs(dfSim, song_from_pl, start_num, 0
, from_cluster)
                        else :
                            song_from_pl = getSongsFrom(dfSongMatch, from
_cluster)
                        song_play = [*song_set, *song_from_pl]
```

```
                        song_info = dfAugSongs.iloc[song_play, :]
                        # metrics
                        metric_match = list(set(train_song_set) & set(son
g_play))
                        metric_match_n = len(metric_match)
                        song_agg = np.array(getPlAgg(song_info)) # aggreg
ated to playlist
                        dist = (song_agg - train_agg_info)**2
                        dist = np.sum(dist, axis=1)
                        dist = np.sqrt(dist)[0]
                        song_numf = round(reg.predict(song_agg.reshape(1,
 -1))[0])
                        numf_diff = int(np.abs(train_numf - song_numf)[0
])
                        sum_metric = 1.0 / metric_match_n + dist + numf_d
iff
                        writer.writerow([pl_id, n_clusters, start_num, ro
und(sum_metric, 2), metric_match_n,
                                         round(dist, 2), song_numf, numf_
diff])
                    except  Exception as e:
                        print("Ex playlist", pl_id, str(e))

kT = int(dfTrain.shape[0] / 6) # dfTrain was changed to dfValidation / df
Test for validation / test
# For validation, no limit was used as number of records was roughly the
 same
limit = 1000
n_cl = [2,10,50,100]
def func1() :
    print("starting 1")
    dfT = dfTrain.iloc[0 : kT, :].sample(limit, random_state=0)
    dfP = pd.merge(df, dfT, left_on='playlist_id', right_on='playlist_id'
, how='left').dropna()
    print("running 1")
    generate_playlists(dfT, dfP, "train1", reg, try_clusters=n_cl, try_st
artNum=[10])
    print("done 1")
def func2() :
    print("starting 2")
    dfT = dfTrain.iloc[kT : 2*kT, :].sample(limit, random_state=0)
    dfP = pd.merge(df, dfT, left_on='playlist_id', right_on='playlist_id'
, how='left').dropna()
    print("running 2")
    generate_playlists(dfT, dfP, "train2", reg, try_clusters=n_cl, try_st
artNum=[10])
    print("done 2")
def func3() :
    print("starting 3")
    dfT = dfTrain.iloc[2*kT : 3*kT, :].sample(limit, random_state=0)
    dfP = pd.merge(df, dfT, left_on='playlist_id', right_on='playlist_id'
, how='left').dropna()
    print("running 3")
    generate_playlists(dfT, dfP, "train3", reg, try_clusters=n_cl, try_st
artNum=[10])
    print("done 3")
def func4() :
```

```
    print("starting 4")
    dfT = dfTrain.iloc[3*kT : 4*kT, :].sample(limit, random_state=0)
    dfP = pd.merge(df, dfT, left_on='playlist_id', right_on='playlist_id'
, how='left').dropna()
    print("running 4")
    generate_playlists(dfT, dfP, "train4", reg, try_clusters=n_cl, try_st
artNum=[10])
    print("done 4")
def func5() :
    print("starting 5")
    dfT = dfTrain.iloc[4*kT : 5*kT, :].sample(limit, random_state=0)
    dfP = pd.merge(df, dfT, left_on='playlist_id', right_on='playlist_id'
, how='left').dropna()
    print("running 5")
    generate_playlists(dfT, dfP, "train5", reg, try_clusters=n_cl, try_st
artNum=[10])
    print("done 5")
def func6() :
    print("starting 6")
    dfT = dfTrain.iloc[5*kT :, :].sample(limit, random_state=0)
    dfP = pd.merge(df, dfT, left_on='playlist_id', right_on='playlist_id'
, how='left').dropna()
    print("running 6")
    generate_playlists(dfT, dfP, "train6", reg, try_clusters=n_cl, try_st
artNum=[10])
    print("done 6")

p1 = Process(target=func1)
p1.start()
p2 = Process(target=func2)
p2.start()
p3 = Process(target=func3)
p3.start()
p4 = Process(target=func4)
p4.start()
p5 = Process(target=func5)
p5.start()
p6 = Process(target=func6)
p6.start()
p1.join()
p2.join()
p3.join()
p4.join()
p5.join()
p6.join()

print("Training done")
```

title: Modelling Results

nav_include: 7

Analysis Approach:

Since our model runs took a very long time, we saved all of our metrics into dataframes to be loaded for later analysis. Each metric has six different values for each metric, based on the mean of that metric for the results of that batch.

Initial Approach

Initially we planned on aggregating the results of the batches and then comaring the different model results. However we found that the way we created our metrics gave us results with very high variability. If we were able to run the models again, we would change the metrics to be proportions of change, i.e. %change in followers vs the current method of absolute change in followers. This would normalize our data and allow us to directly compare results on playlists with large differences in number of songs and number of followers.

Compromise Approach

In order to try and compare models with the metrics we already have, we combined the means of each batch along with the standard deviation of the means for each metric and compared those values. This led to some strange results, with the validation and test sets consistently outperforming the train set for each metric. However when you look at the $\pm 2\sigma$ bounds on the scores, you can see that there is significant overlap.

Results summary

- Our model generates playlists, even matches some of the original songs in playlist from which only 1 song is taken! I.e. it addresses both goals set out for the project
- Scoring and analysis shows that clustering and song similarity is a viable approach. Visual inspection shows that playlists seem to make sense (i.e. follow the "theme" of nucleus song)
- There's still variance in results which means there's room for improvement. A lot of variance come from estimated number of followers which is a metric that is subject to Spotify promotion and "hit" phenomena

Future work

Extension would be to test other metaparameters listed in "Modelling". Added song tags and experimenting with words (lyrics, playlist names) would be helpful. An interesting approach would be great to try - TF/IDF and collaborative filtering which were used in https://ieatyanyans.github.io/music-recommender/ (https://ieatyanyans.github.io/music-recommender/)

```
Team Group 71:
```

- Brandon Joye
- Vladimir Yelizarov

In [1]:
```
import sys
import datetime
import numpy as np
import pandas as pd
import string
from sklearn.cluster import KMeans
from sklearn.linear_model import LinearRegression
import gzip
import csv
import matplotlib
import matplotlib.pyplot as plt

DATA_DIR="../../../data"
```

In [2]:
```
def add(r, names, df) :
    for m in names:
        r[m].append(df[m].mean())

def tonp (r, names):
    for m in names:
        r[m] = np.array(r[m])

def readResults(n, shortname, name):
    m_names = ['metric', 'match', 'distance', 'numf', 'diff']
    prefix = DATA_DIR + "/results/" + shortname + str(n) + "/result_" + n
ame
    suffix = "_" + str(n) + "_10.csv.gz"
    r = {'metric':[], 'match' : [], 'distance' : [], 'numf' : [], 'diff'
: [], 'metric2' : [] }
    for i in range(1, 7) :
        fullName = prefix + str(i) + suffix
        df = pd.read_csv(fullName, compression='gzip')#.drop(['Unnamed:
 0'],axis=1)
        add(r, m_names, df)
        r['metric2'].append(((1.0 / df['match']) + df['distance']).mean
())
    tonp(r, m_names)
    tonp(r, ['metric2'])
    return r

def readResults2(n, shortname, name):
    m_names = ['metric', 'match', 'distance', 'numf', 'diff']
    prefix = DATA_DIR + "/results/" + shortname + str(n) + "/result_" + n
ame
    suffix = "_" + str(n) + "_10.csv.gz"
    r = {'metric':[], 'match' : [], 'distance' : [], 'numf' : [], 'diff'
: [], 'metric2' : [] }
    df_full = pd.DataFrame()
    for i in range(1, 7) :
        fullName = prefix + str(i) + suffix
        df = pd.read_csv(fullName, compression='gzip')#.drop(['Unnamed:
 0'],axis=1)
        df_full = df_full.append(df)
        add(r, m_names, df)
        r['metric2'].append(((1.0 / df['match']) + df['distance']).mean
())
    tonp(r, m_names)
    tonp(r, ['metric2'])
    return df_full
```

In [3]:
```
t2 = readResults(2, "t", "train")
t10 = readResults(10, "t", "train")
t50 = readResults(50, "t", "train")
t100 = readResults(100, "t", "train")
```

In [4]:
```
v2 = readResults(2, "v", "validate")
v10 = readResults(10, "v", "validate")
v50 = readResults(50, "v", "validate")
v100 = readResults(100, "v", "validate")
```

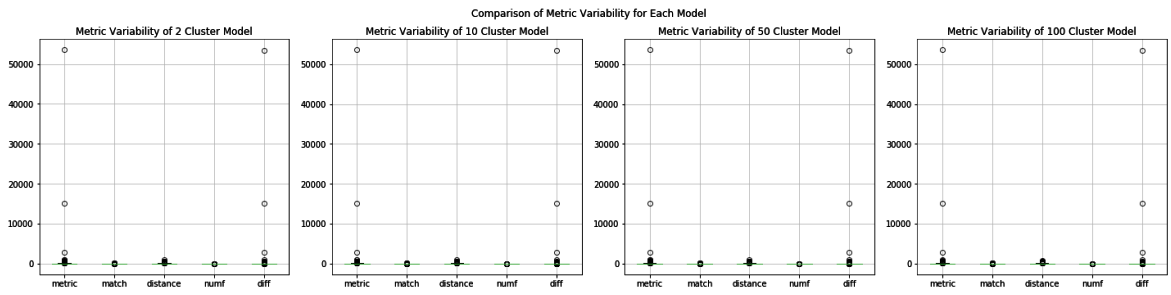
In [5]:
```
t2df = readResults2(2, "t", "train")
t10df = readResults2(10, "t", "train")
t50df = readResults2(50, "t", "train")
t100df = readResults2(100, "t", "train")
```

In [6]:
```
t2df.agg([np.mean, np.std])
```

Out[6]:

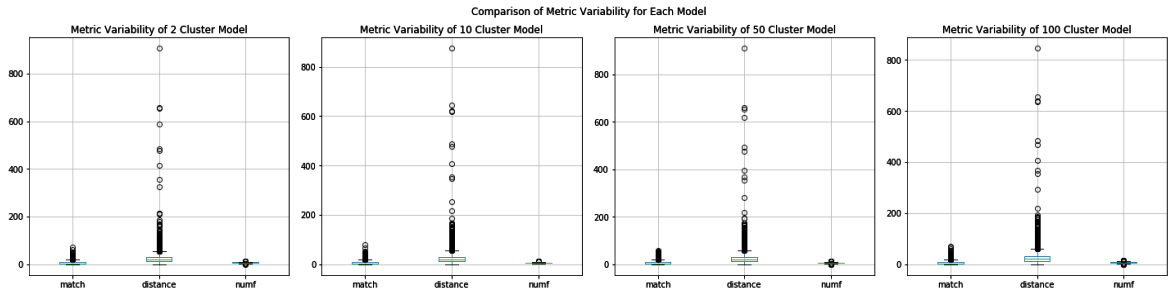
	playlist_id	n_clusters	start_num	metric	match	distance	numf	c
mean	492621.840113	2.0	10.0	41.498213	6.019840	23.742583	6.903134	17.3676
std	284337.026741	0.0	0.0	719.542719	5.571262	27.351196	1.296133	719.1216

In [7]:
```
fig, ax = plt.subplots(1,4,figsize=(20,5))
t2df.drop(['playlist_id', 'n_clusters', 'start_num'], axis=1).boxplot(ax=
ax[0])
ax[0].set_title('Metric Variability of 2 Cluster Model')
t10df.drop(['playlist_id', 'n_clusters', 'start_num'], axis=1).boxplot(ax
=ax[1])
ax[1].set_title('Metric Variability of 10 Cluster Model')
t50df.drop(['playlist_id', 'n_clusters', 'start_num'], axis=1).boxplot(ax
=ax[2])
ax[2].set_title('Metric Variability of 50 Cluster Model')
t100df.drop(['playlist_id', 'n_clusters', 'start_num'], axis=1).boxplot(a
x=ax[3])
ax[3].set_title('Metric Variability of 100 Cluster Model')
fig.suptitle('Comparison of Metric Variability for Each Model')
fig.tight_layout(rect=[0, 0, 1, .95])
plt.show()
```



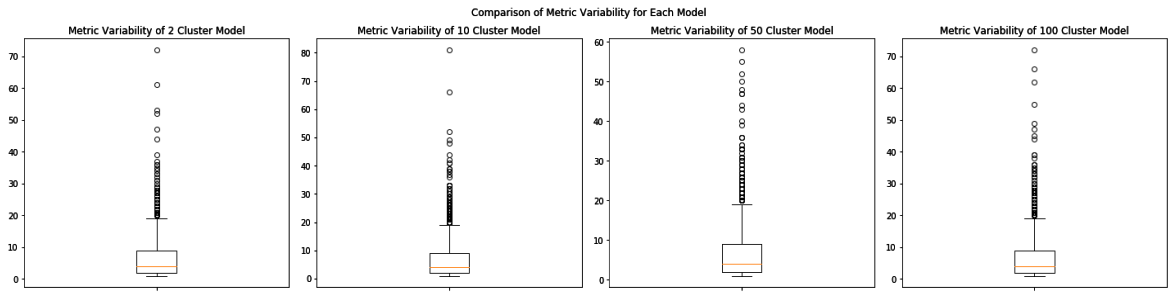
The above plots show that there is significant variability in each metric, this is most pronounced in the difference in predicted followers vs actual followers metric.

In [8]:
```
fig, ax = plt.subplots(1,4,figsize=(20,5))
t2df.drop(['playlist_id', 'n_clusters', 'start_num', 'metric', 'diff'], a
xis=1).boxplot(ax=ax[0])
ax[0].set_title('Metric Variability of 2 Cluster Model')
t10df.drop(['playlist_id', 'n_clusters', 'start_num', 'metric', 'diff'],
axis=1).boxplot(ax=ax[1])
ax[1].set_title('Metric Variability of 10 Cluster Model')
t50df.drop(['playlist_id', 'n_clusters', 'start_num', 'metric', 'diff'],
axis=1).boxplot(ax=ax[2])
ax[2].set_title('Metric Variability of 50 Cluster Model')
t100df.drop(['playlist_id', 'n_clusters', 'start_num', 'metric', 'diff'],
 axis=1).boxplot(ax=ax[3])
ax[3].set_title('Metric Variability of 100 Cluster Model')
fig.suptitle('Comparison of Metric Variability for Each Model')
fig.tight_layout(rect=[0, 0, 1, .95])
plt.show()
```

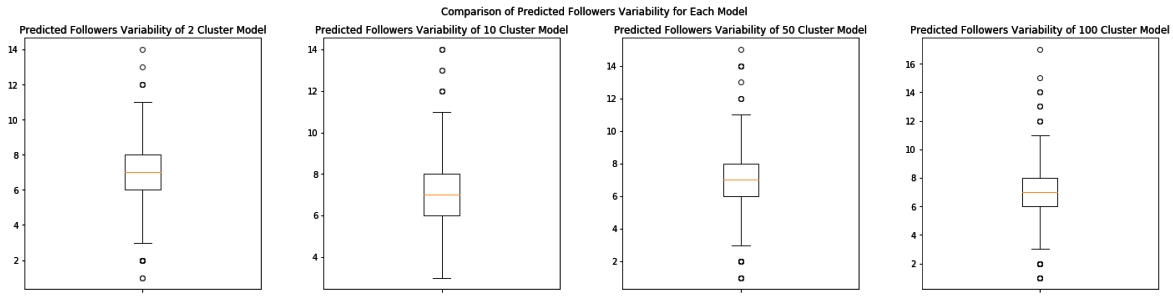


The distance between playlists also has significant variability, one way we could possibly reduce this is to normalize the distance variables so that distance is a number between 0 and 1.

In [8]:
```
fig, ax = plt.subplots(1,4,figsize=(20,5))
ax[0].boxplot(t2df.match)
ax[0].set_title('Metric Variability of 2 Cluster Model')
ax[1].boxplot(t10df.match)
ax[1].set_title('Metric Variability of 10 Cluster Model')
ax[2].boxplot(t50df.match)
ax[2].set_title('Metric Variability of 50 Cluster Model')
ax[3].boxplot(t100df.match)
ax[3].set_title('Metric Variability of 100 Cluster Model')
for axis in ax:
    axis.set_xticklabels('')
fig.suptitle('Comparison of Metric Variability for Each Model')
fig.tight_layout(rect=[0, 0, 1, .95])
plt.show()
```



In [9]:
```
fig, ax = plt.subplots(1,4,figsize=(20,5))
ax[0].boxplot(t2df.numf)
ax[0].set_title('Predicted Followers Variability of 2 Cluster Model')
ax[1].boxplot(t10df.numf)
ax[1].set_title('Predicted Followers Variability of 10 Cluster Model')
ax[2].boxplot(t50df.numf)
ax[2].set_title('Predicted Followers Variability of 50 Cluster Model')
ax[3].boxplot(t100df.numf)
ax[3].set_title('Predicted Followers Variability of 100 Cluster Model')
for axis in ax:
    axis.set_xticklabels('')
fig.suptitle('Comparison of Predicted Followers Variability for Each Mode
l')
fig.tight_layout(rect=[0, 0, 1, .95])
plt.show()
```



In [10]:
```
test2 = readResults(2, 'test', 'test')
test10 = readResults(10, 'test', 'test')
```

In [11]:
```
def make_mean_metrics(models, sn):
    results_dict = {}
    #models = [t2, t10, t50, t100]
    names = [2, 10, 50, 100]
    for model, name in zip(models, names):
        model_params = {}
        for key in model:
            model_params[key] = np.mean(model[key])
            model_params[key+'std'] = np.std(model[key])
        results_dict[sn+str(name)] = model_params
    return results_dict
```

In [12]:
```
models_train = [t2, t10, t50, t100]
train_dict = make_mean_metrics(models_train, 't')
results_train = pd.DataFrame.from_dict(train_dict).T
results_train
```

Out[12]:

	diff	diffstd	distance	distancestd	match	matchstd	metric	metric2	m
t2	17.366214	19.586043	23.742464	1.204582	6.019813	0.117456	41.496689	24.130509	
t10	17.447881	19.232502	24.761268	0.929312	6.129652	0.176668	42.599327	25.151356	
t50	17.313173	19.268623	25.829698	0.902331	6.151795	0.124508	43.530253	26.217108	
t100	17.419482	19.445556	25.663053	0.693858	6.215457	0.099706	43.468382	26.048898	

In [13]:
```
models_validate = [v2, v10, v50, v100]
validate_dict = make_mean_metrics(models_validate, 'v')
results_validate = pd.DataFrame.from_dict(validate_dict).T
results_validate
```

Out[13]:

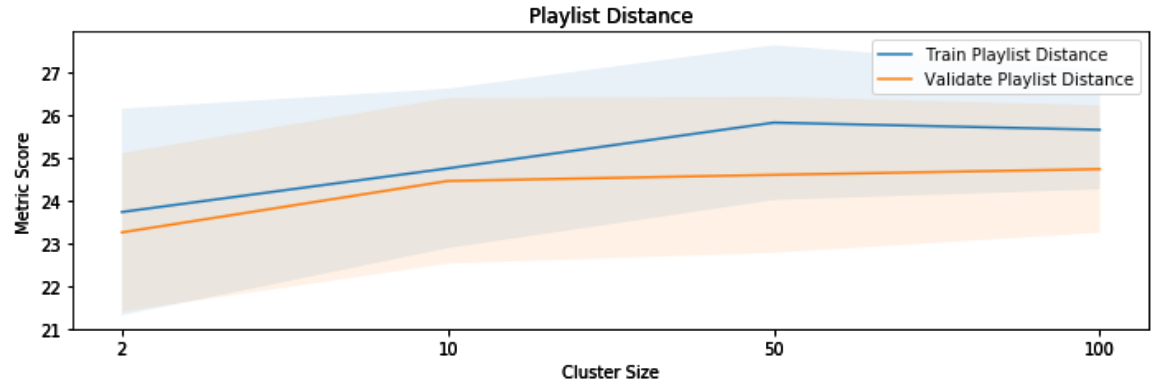
	diff	diffstd	distance	distancestd	match	matchstd	metric	metric2	met
v2	8.383877	3.004734	23.266120	0.923800	6.221482	0.087430	32.019275	23.635325	0.
v10	7.866056	2.634214	24.467220	0.964562	6.322889	0.081322	32.699332	24.833359	0.
v50	7.961983	2.822155	24.610007	0.909844	6.493108	0.070977	32.932138	24.970164	0.
v100	7.884393	2.696971	24.743925	0.742098	6.541668	0.095974	32.985807	25.101392	0.

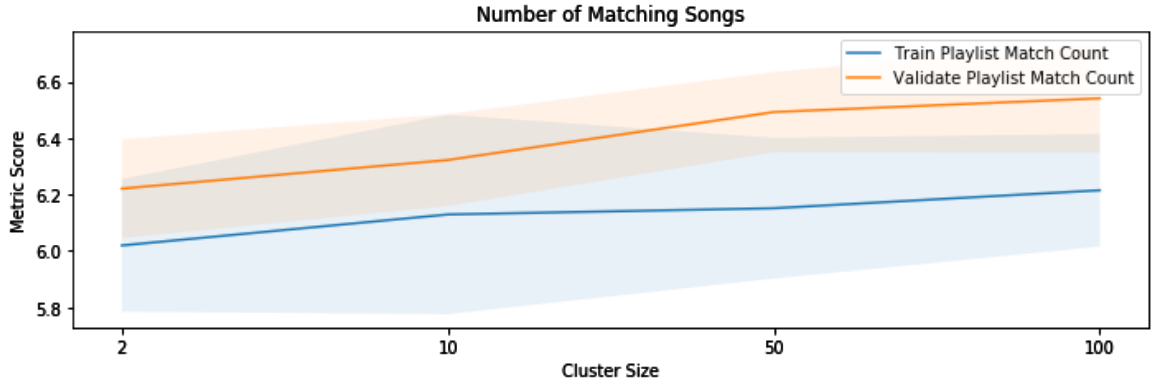
In [14]:
```
models_test = [test2, test10]
test_dict = make_mean_metrics(models_test, 'test')
results_test = pd.DataFrame.from_dict(test_dict).T
results_test
```

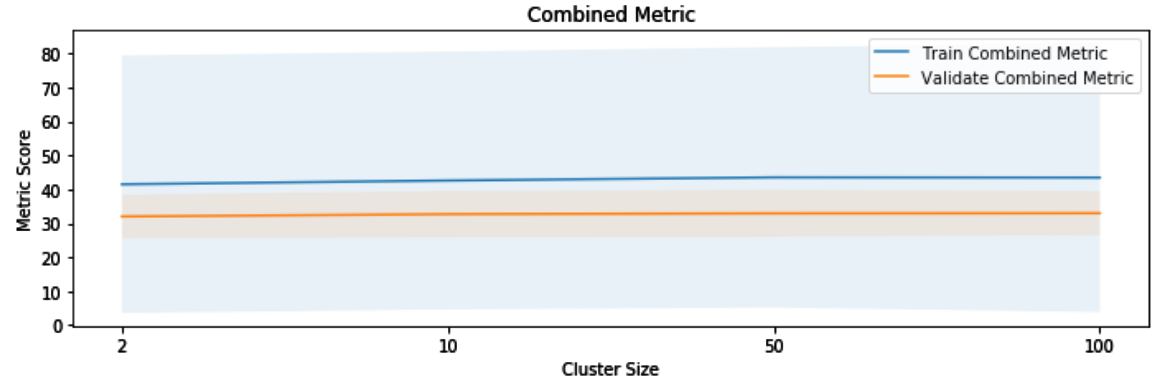
Out[14]:

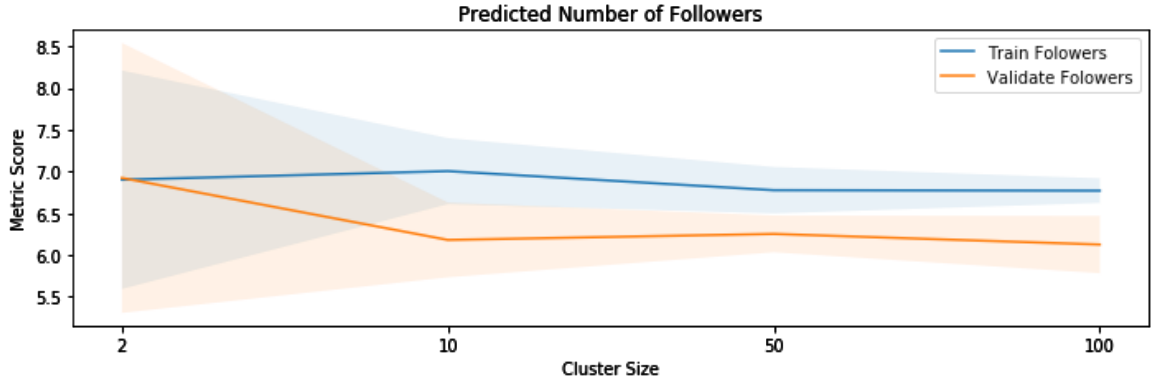
	diff	diffstd	distance	distancestd	match	matchstd	metric	metric2	m
test2	13.347084	7.996984	25.105574	1.956354	6.189874	0.139431	38.832057	25.48498	
test10	12.797532	8.089814	24.755806	0.846285	6.294219	0.155728	37.930501	25.13291	

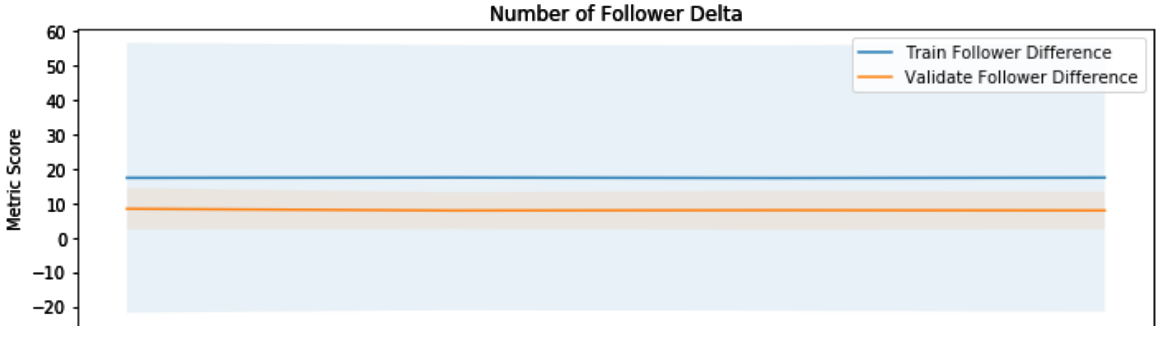
In [15]:
```
plot_alpha = .1
fig, ax = plt.subplots(6,1, figsize=(10,20))
labels_train = ['Train Playlist Distance', 'Train Playlist Match Count',
'Train Combined Metric', 'Train Folowers',
                'Train Follower Difference', 'Train Matching Songs plus D
istance']
labels_validate = ['Validate Playlist Distance', 'Validate Playlist Match
 Count', 'Validate Combined Metric',
                   'Validate Folowers', 'Validate Follower Difference',
'Validate Matching Songs plus Distance']
plot_order = ['distance', 'match', 'metric', 'numf', 'diff', 'metric2']
titles = ['Playlist Distance', 'Number of Matching Songs', 'Combined Metr
ic', 'Predicted Number of Followers',
          'Number of Follower Delta', 'Inverted Matching Songs plus Dista
nce']
names = [2, 10, 50, 100]
# results_train.plot(y='diff', ax=ax[0])
# results_train.plot(y='distance', ax=ax[1])
# results_train.plot(y='match', ax=ax[2])
# results_train.plot(y='metric', ax=ax[3])
for axis, po, labelt, labelv, title in \
zip(ax, plot_order, labels_train, labels_validate, titles):
    results_train.plot(y=po, ax=axis, label=labelt)
    axis.fill_between(np.arange(4), results_train[po] + 2*results_train[p
o+'std'],
                      results_train[po] - 2*results_train[po+'std'], alph
a=plot_alpha)
    results_validate.plot(y=po, ax=axis, label=labelv)
    axis.fill_between(np.arange(4), results_validate[po] + 2*results_vali
date[po+'std'],
                      results_validate[po] - 2*results_validate[po+'std'
], alpha=plot_alpha)
    axis.set_xlabel('Cluster Size')
    axis.set_ylabel('Metric Score')
    axis.set_xticks(np.arange(4))
    axis.set_xticklabels(names)
    axis.set_title(title)
fig.tight_layout()
plt.show()
```

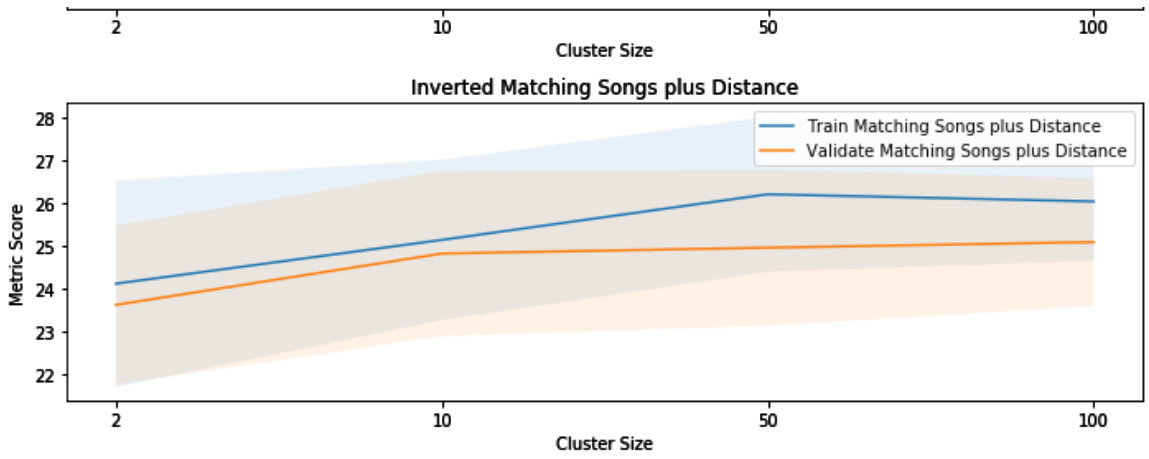

Playlist Distance
Train Playlist Distance
Validate Playlist Distance
Metric Score
Cluster Size
27
26
25
24
23
22
21
2
10
50
100


Number of Matching Songs
Train Playlist Match Count
Validate Playlist Match Count
Metric Score
Cluster Size
6.6
6.4
6.2
6.0
5.8
2
10
50
100


Combined Metric
Train Combined Metric
Validate Combined Metric
Metric Score
Cluster Size
80
70
60
50
40
30
20
10
0
2
10
50
100


Predicted Number of Followers
Train Folowers
Validate Folowers
Metric Score
Cluster Size
8.5
8.0
7.5
7.0
6.5
6.0
5.5
2
10
50
100


Number of Follower Delta
Train Follower Difference
Validate Follower Difference
Metric Score
60
50
40
30
20
10
0
−10
−20

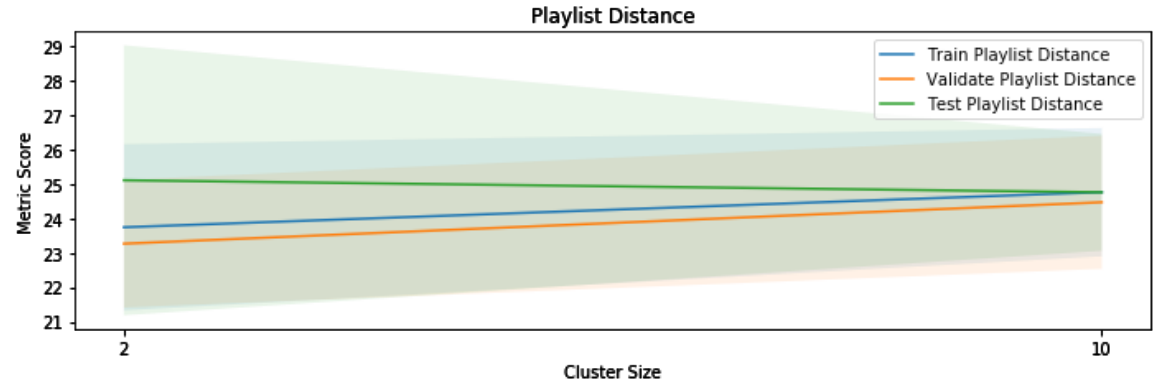


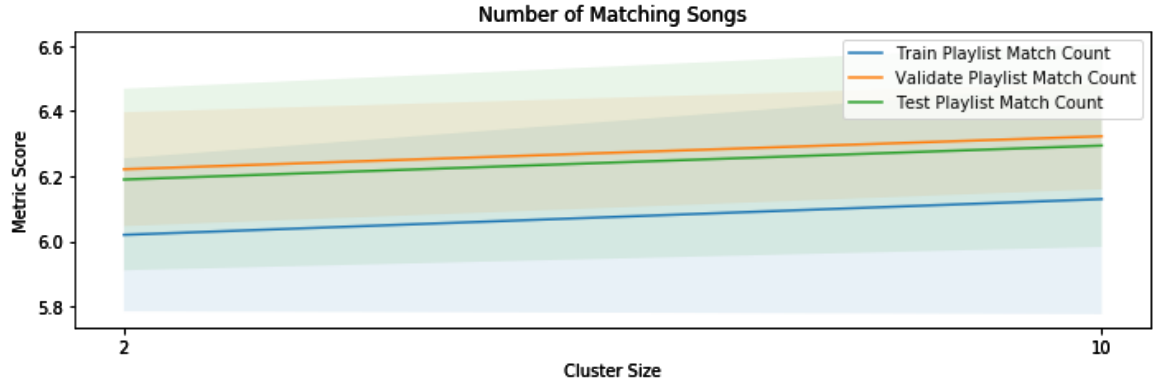
In [16]:
```
results_validate_test = results_validate.loc[['v2', 't10'], :]
results_train_test = results_train.loc[['t2', 't10'], :]
```

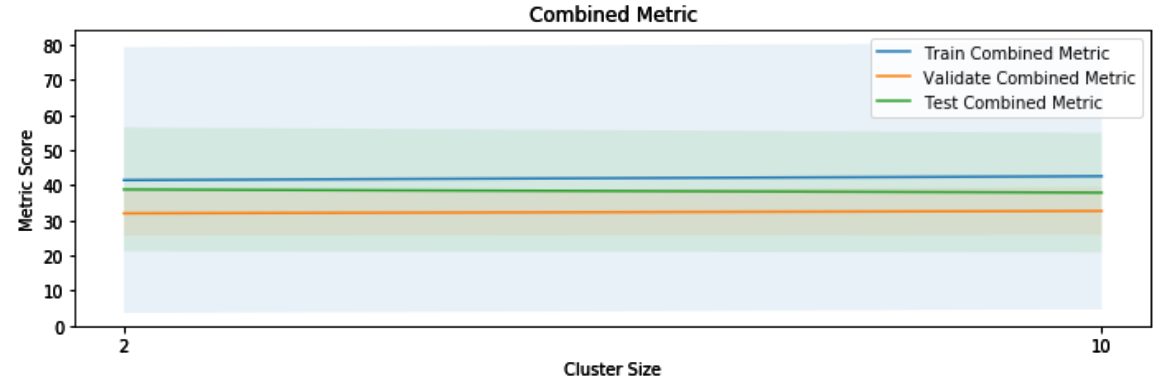
In [17]:
```
po, results_train[po+'std']
```

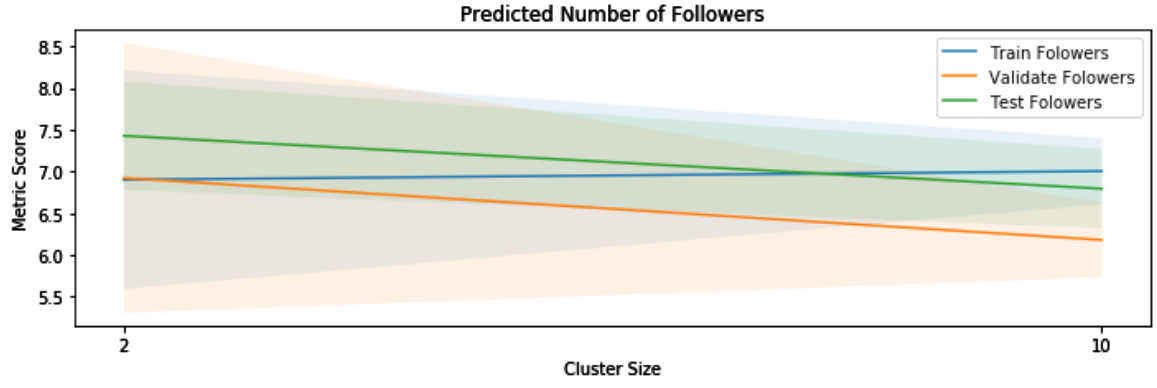
Out[17]:
```
('metric2', t2      1.202147
 t10     0.934653
 t50     0.903429
 t100    0.689577
 Name: metric2std, dtype: float64)
```

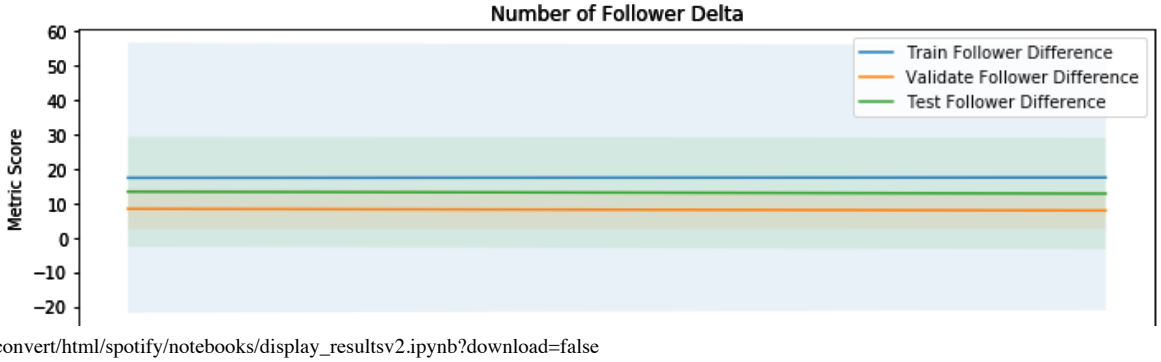
In [18]:
```
fig, ax = plt.subplots(6,1, figsize=(10,20))
labels_train = ['Train Playlist Distance', 'Train Playlist Match Count',
'Train Combined Metric', 'Train Folowers',
                'Train Follower Difference', 'Train Matching Songs plus D
istance']
labels_validate = ['Validate Playlist Distance', 'Validate Playlist Match
 Count', 'Validate Combined Metric',
                   'Validate Folowers', 'Validate Follower Difference',
'Validate Matching Songs plus Distance']
labels_test = ['Test Playlist Distance', 'Test Playlist Match Count', 'Te
st Combined Metric', 'Test Folowers',
               'Test Follower Difference', 'Test Matching Songs plus Dist
ance']
plot_order = ['distance', 'match', 'metric', 'numf', 'diff', 'metric2']
titles = ['Playlist Distance', 'Number of Matching Songs', 'Combined Metr
ic', 'Predicted Number of Followers',
          'Number of Follower Delta', 'Inverted Matching Songs plus Dista
nce']
names = [2, 10]
# results_train.plot(y='diff', ax=ax[0])
# results_train.plot(y='distance', ax=ax[1])
# results_train.plot(y='match', ax=ax[2])
# results_train.plot(y='metric', ax=ax[3])
for axis, po, labelt, labelv, title, labeltest in \
zip(ax, plot_order, labels_train, labels_validate, titles, labels_test):
    results_train_test.plot(y=po, ax=axis, label=labelt)
    axis.fill_between(np.arange(2), results_train_test[po] + 2*results_tr
ain_test[po+'std'],
                      results_train_test[po] - 2*results_train_test[po+'s
td'], alpha=plot_alpha)
    results_validate_test.plot(y=po, ax=axis, label=labelv)
    axis.fill_between(np.arange(2), results_validate_test[po] + 2*results
_validate_test[po+'std'],
                      results_validate_test[po] - 2*results_validate_test
[po+'std'], alpha=plot_alpha)
    results_test.plot(y=po, ax=axis, label=labeltest)
    axis.fill_between(np.arange(2), results_test[po] + 2*results_test[po+
'std'],
                      results_test[po] - 2*results_test[po+'std'], alpha=
plot_alpha)
    axis.set_xlabel('Cluster Size')
    axis.set_ylabel('Metric Score')
    axis.set_xticks(np.arange(2))
    axis.set_xticklabels(names)
    axis.set_title(title)
fig.tight_layout()
plt.show()
```

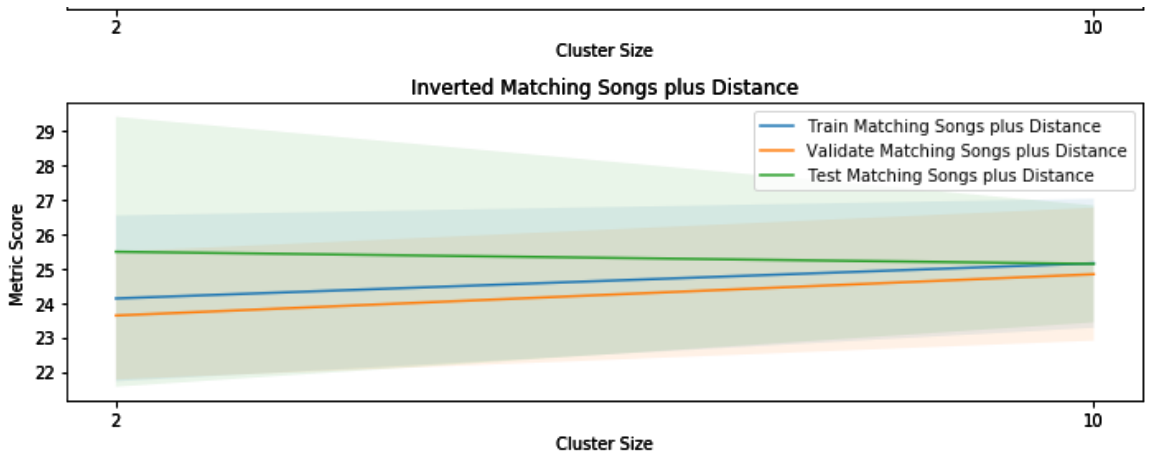

Playlist Distance
Train Playlist Distance
Validate Playlist Distance
Test Playlist Distance
Metric Score
Cluster Size


Number of Matching Songs
Train Playlist Match Count
Validate Playlist Match Count
Test Playlist Match Count
Metric Score
Cluster Size

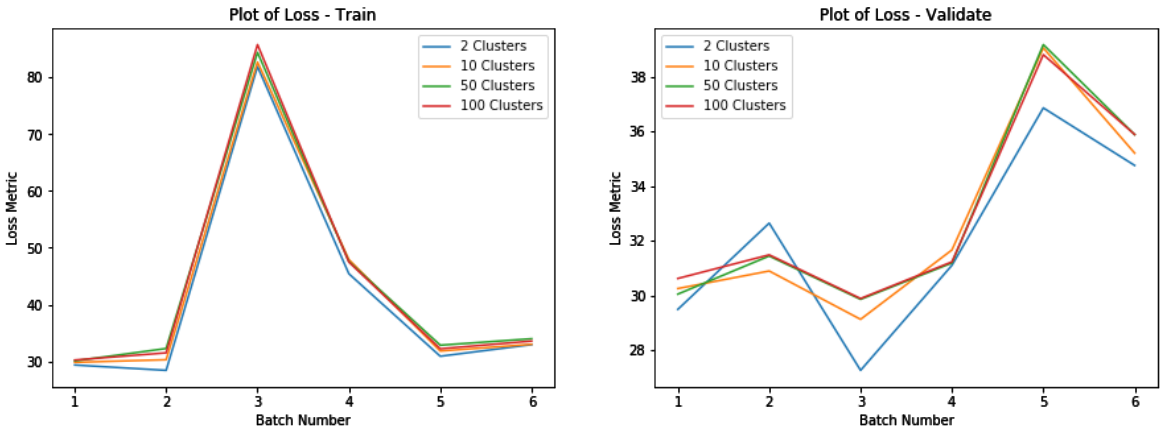

Combined Metric
Train Combined Metric
Validate Combined Metric
Test Combined Metric
Metric Score
Cluster Size


Predicted Number of Followers
Train Folowers
Validate Folowers
Test Folowers
Metric Score
Cluster Size


Number of Follower Delta
Train Follower Difference
Validate Follower Difference
Test Follower Difference
Metric Score

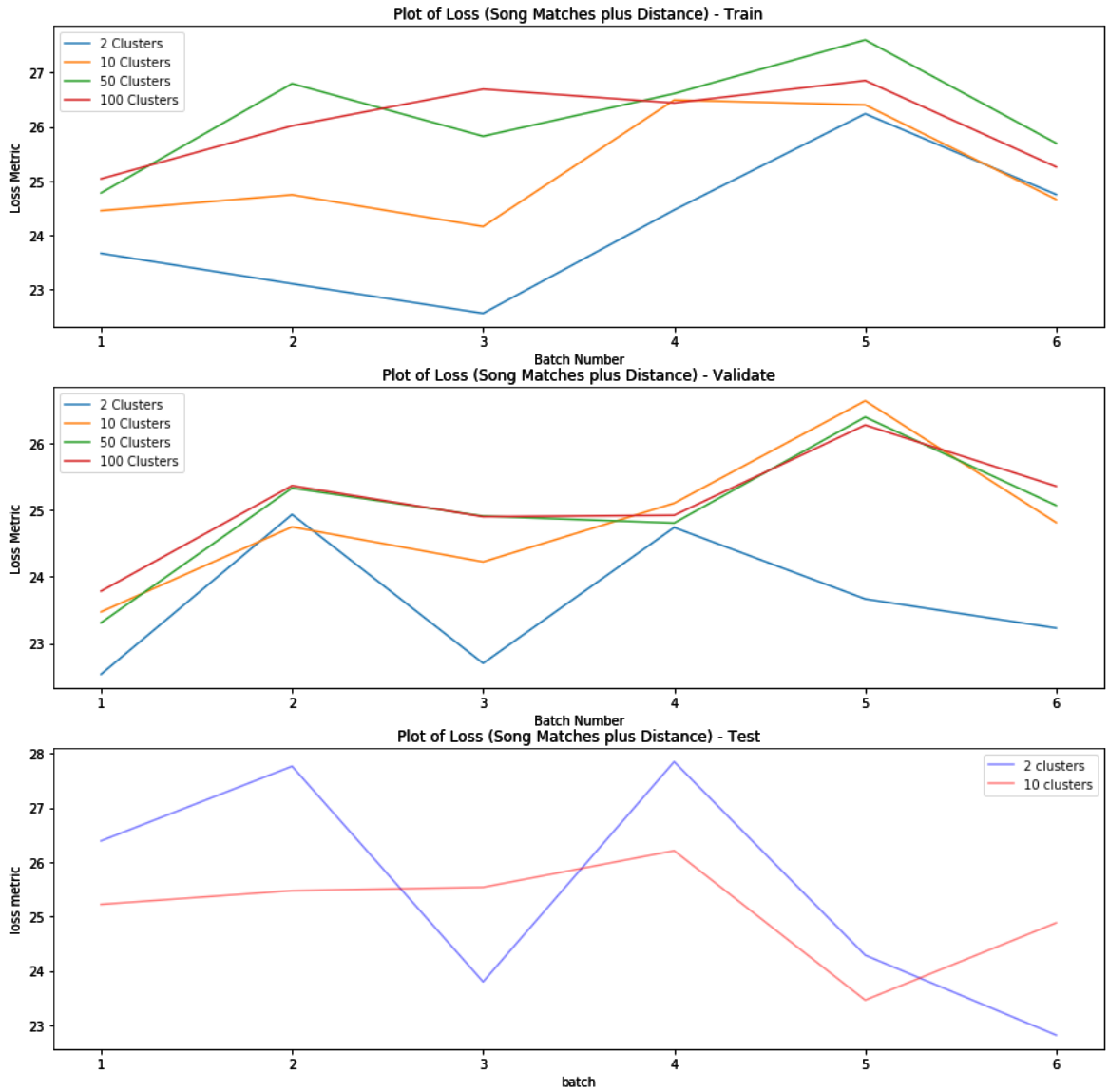


In [19]:
```
x = range(1, 7)
fig, axes = plt.subplots(1,2,figsize=(15,5))
axes[0].set_title('Plot of Loss - Train')
axes[0].plot(x, t2['metric'], label='2 Clusters')
axes[0].plot(x, t10['metric'], label='10 Clusters')
axes[0].plot(x, t50['metric'], label='50 Clusters')
axes[0].plot(x, t100['metric'], label='100 Clusters')
axes[0].set_xlabel('Batch Number')
axes[0].set_ylabel('Loss Metric')
axes[0].legend()
axes[1].set_title('Plot of Loss - Validate')
axes[1].plot(x, v2['metric'], label='2 Clusters')
axes[1].plot(x, v10['metric'], label='10 Clusters')
axes[1].plot(x, v50['metric'], label='50 Clusters')
axes[1].plot(x, v100['metric'], label='100 Clusters')
axes[1].set_xlabel('Batch Number')
axes[1].set_ylabel('Loss Metric')
axes[1].legend()
plt.show()
```



Looking at the combined metric by batch shows that each model reacted to the individual batches in a fairly consistent manner. This is slightly less true for the validation set, but the relationship is still there for at least the 10, 50 and 100 cluster models.

In [20]:
```
x = range(1, 7)
fig, axes = plt.subplots(3,1,figsize=(15,15))
axes = axes.ravel()
axes[0].set_title('Plot of Loss (Song Matches plus Distance) - Train')
axes[0].plot(x, t2['metric2'], label='2 Clusters')
axes[0].plot(x, t10['metric2'], label='10 Clusters')
axes[0].plot(x, t50['metric2'], label='50 Clusters')
axes[0].plot(x, t100['metric2'], label='100 Clusters')
axes[0].set_xlabel('Batch Number')
axes[0].set_ylabel('Loss Metric')
axes[0].legend()
axes[1].set_title('Plot of Loss (Song Matches plus Distance) - Validate')
axes[1].plot(x, v2['metric2'], label='2 Clusters')
axes[1].plot(x, v10['metric2'], label='10 Clusters')
axes[1].plot(x, v50['metric2'], label='50 Clusters')
axes[1].plot(x, v100['metric2'], label='100 Clusters')
axes[1].set_xlabel('Batch Number')
axes[1].set_ylabel('Loss Metric')
axes[1].legend()
axes[2].set_title('Plot of Loss (Song Matches plus Distance) - Test')
axes[2].plot(x, test2['metric2'], alpha=0.5, color='b', label='2 cluster
s')
axes[2].plot(x, test10['metric2'], alpha=0.5, color='r', label='10 cluste
rs')
axes[2].set_xlabel('batch')
axes[2].set_ylabel('loss metric')
axes[2].legend()
plt.show()
```



We created a second combined metric aimed at reducing the variability, this metric has results that are more in line with expectations with the model performing slightly better on the train set than the other sets.