1 DISCO DUO - Capstone Project

- Student name: Vi Bui
- Student pace: Part-Time
- Scheduled project review date/time: Mon. 05/02/22
- Instructor name: Claude Fried
- Blog post URL: https://datasciish.com/ (https://datasciish.com/)

```
<img src='GitHub_Images/Disco Duo Logo Prototype.jpg' width=70%>
```


```
## Overview

**Client:** Existing or new music streaming services. Existing: Spotify,
Pandora, Amazon Music, etc. New: companies interested in building new
platforms to connect people through music.

**Objective:** Create a platform where listeners of the same song are
connected and able to discover new songs through their “connector song”
by requesting another song based on a musical metric such as:
danceability, loudness, acousticness, valence, etc.
```

```
**Data, Methodology, and Models** <br/>

**Data source**: Spotify
1. Spotify Song Data - https://www.kaggle.com/akiboy96/spotify-dataset
2. Spotify Genre Data - https://www.kaggle.com/code/akiboy96/spotify-song-popularity-genre-exploration/data?select=genre_music.csv

**Methodology:** Pull sample from data; create spectrogram images for songs; train model to predict danceability

**Models:** Sequential Models (Keras)
1. Layers
2. Stochastic
3. Add layers
```

1.1 Overview

Client: Existing or new music streaming services. Existing: Spotify, Pandora, Amazon Music, etc. New: companies interested in building new platforms to connect people through music.

Objective: Create a platform where listeners of the same song are connected and able to discover new songs through their "connector song" by requesting another song based on a musical metric such as: danceability, loudness, acousticness, valence, etc.

Data, Methodology, and Models

Data source: Spotify

1. Spotify Song Data - https://www.kaggle.com/akiboy96/spotify-dataset (https://www.kaggle.com/akiboy96/spotify-dataset)
2. Spotify Genre Data - https://www.kaggle.com/code/akiboy96/spotify-song-popularity-genre-exploration/data?select=genre_music.csv (https://www.kaggle.com/code/akiboy96/spotify-song-popularity-genre-exploration/data?select=genre_music.csv)

Methodology: Pull sample from data; create spectrogram images for songs; train model to predict danceability

Models: Sequential Models (Keras)

1. Layers
2. Stochastic
3. Add layers

2 Data Exploration, Cleansing, and Visualization

Data Exploration
Explore Spotify dataset

Data Cleansing
Check for duplicates; drop duplicate and NaN (missing) values; continuously clean data as necessary

Data Visualization
Use visualizations to explore the data and determine how to further refine the dataset in order to prepare for modeling

Data Preparation
Prepare the data for modeling

2.1 Data Exploration and Cleansing

Import data and all packages needed for data exploration and modeling

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

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import pydub
from pydub import AudioSegment

import librosa
import librosa.display

import tensorflow as tf
import tensorflow_io as tfio
from tensorflow.keras import layers, models
from tensorflow.keras.callbacks import EarlyStopping

from keras.models import Sequential

# Import from keras_preprocessing not from keras.preprocessing
from keras_preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Activation, Flatten, Dropout, BatchNormaliz
from keras.layers import Conv2D, MaxPooling2D
from keras import regularizers, optimizers

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder

from numpy.random import seed
seed(11)

tf.random.set_seed(11)

import os
import warnings
```

executed in 4.00s, finished 17:07:26 2022-05-05

In [2]:
```
# Import song data

songs = pd.read_csv('spotify_data.csv',index_col=0)
```
executed in 119ms, finished 17:07:26 2022-05-05

In [3]:
```
# View song dataframe

songs.head()
```
executed in 16ms, finished 17:07:26 2022-05-05

Out[3]:

	artist	uri	danceability	energy	key	loudness	m
track							
Jealous Kind Of Fella	Garland Green	spotify:track:1dtKN6wwlolkM8XZy2y9C1	0.417	0.620	3	-7.727	
Initials B.B.	Serge Gainsbourg	spotify:track:5hjsmSnUefdUqzsDogisiX	0.498	0.505	3	-12.475	
Melody Twist	Lord Melody	spotify:track:6uk8tI6pwxxdVTNlNOJeJh	0.657	0.649	5	-13.392	
Mi Bomba Sonó	Celia Cruz	spotify:track:7aNjMJ05FvUXACPWZ7yJmv	0.590	0.545	7	-12.058	
Uravu Solla	P. Susheela	spotify:track:1rQ0clvgkzWr001POOPJWx	0.515	0.765	11	-3.515	

In [4]:
```
# Import genre data

genres = pd.read_csv('genre_data.csv',index_col=0)
```
executed in 99ms, finished 17:07:26 2022-05-05

In [5]:
```
# View genre dataframe

genres.head()
```

executed in 14ms, finished 17:07:26 2022-05-05

Out[5]:

track	artist	danceability	energy	key	loudness	mode	speechiness	acousticness	instrur
Jealous Kind Of Fella	Garland Green	0.417	0.620	3	-7.727	1	0.0403	0.490	
Initials B.B.	Serge Gainsbourg	0.498	0.505	3	-12.475	1	0.0337	0.018	
Melody Twist	Lord Melody	0.657	0.649	5	-13.392	1	0.0380	0.846	
Mi Bomba Sonó	Celia Cruz	0.590	0.545	7	-12.058	0	0.1040	0.706	
Uravu Solla	P. Susheela	0.515	0.765	11	-3.515	0	0.1240	0.857	

In [6]:
```
# Merge Song and Genre datasets

df2 = pd.merge(left=songs, right=genres, on='track')
```

executed in 67ms, finished 17:07:26 2022-05-05

In [7]:
```
# Explore new dataset

df2.head()
```
executed in 20ms, finished 17:07:26 2022-05-05

Out[7]:

	artist_x	uri	danceability_x	energy_x	key_x	loudne
track						
Jealous Kind Of Fella	Garland Green	spotify:track:1dtKN6wwlolkM8XZy2y9C1	0.417	0.620	3	-7
Initials B.B.	Serge Gainsbourg	spotify:track:5hjsmSnUefdUqzsDogisiX	0.498	0.505	3	-12
Melody Twist	Lord Melody	spotify:track:6uk8tl6pwxxdVTNlNOJeJh	0.657	0.649	5	-13
Mi Bomba Sonó	Celia Cruz	spotify:track:7aNjMJ05FvUXACPWZ7yJmv	0.590	0.545	7	-12
Uravu Solla	P. Susheela	spotify:track:1rQ0clvgkzWr001POOPJWx	0.515	0.765	11	-3

5 rows × 38 columns

Note: Dataframe does not reflect desired output; create new dataframe with just 'track' and 'genre'

In [8]:
```
# Create new dataframe with just 'track' and 'genre'
# genres[['track','genre']] did not work; use filter method

new_genre = genres.filter(['track','genre'])
```
executed in 3ms, finished 17:07:26 2022-05-05

In [9]:
```
# View new_genre dataframe

new_genre.head()
```
executed in 5ms, finished 17:07:26 2022-05-05

Out[9]:

	genre
track	
Jealous Kind Of Fella	edm
Initials B.B.	pop
Melody Twist	pop
Mi Bomba Sonó	pop
Uravu Solla	r&b

In [10]:
```
# Merge genre dataframe with song dataframe

df = pd.merge(left=songs, right=new_genre, on='track')
```
executed in 48ms, finished 17:07:26 2022-05-05

In [11]:
```
# View new dataframe

df.head()
```
executed in 15ms, finished 17:07:26 2022-05-05

Out[11]:

	artist	uri	danceability	energy	key	loudness	m
track							
Jealous Kind Of Fella	Garland Green	spotify:track:1dtKN6wwlolkM8XZy2y9C1	0.417	0.620	3	-7.727	
Initials B.B.	Serge Gainsbourg	spotify:track:5hjsmSnUefdUqzsDogisiX	0.498	0.505	3	-12.475	
Melody Twist	Lord Melody	spotify:track:6uk8tl6pwxxdVTNlNOJeJh	0.657	0.649	5	-13.392	
Mi Bomba Sonó	Celia Cruz	spotify:track:7aNjMJ05FvUXACPWZ7yJmv	0.590	0.545	7	-12.058	
Uravu Solla	P. Susheela	spotify:track:1rQ0clvgkzWr001POOPJWx	0.515	0.765	11	-3.515	

In [12]:
```
# View info for dataframe

df.info()
```

executed in 26ms, finished 17:07:26 2022-05-05

```
<class 'pandas.core.frame.DataFrame'>
Index: 58472 entries, Jealous Kind Of Fella to Calling My Spirit
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   artist            58472 non-null  object 
 1   uri               58472 non-null  object 
 2   danceability      58472 non-null  float64
 3   energy            58472 non-null  float64
 4   key               58472 non-null  int64  
 5   loudness          58472 non-null  float64
 6   mode              58472 non-null  int64  
 7   speechiness       58472 non-null  float64
 8   acousticness      58472 non-null  float64
 9   instrumentalness  58472 non-null  float64
 10  liveness          58472 non-null  float64
 11  valence           58472 non-null  float64
 12  tempo             58472 non-null  float64
 13  duration_ms       58472 non-null  int64  
 14  time_signature    58472 non-null  int64  
 15  chorus_hit        58472 non-null  float64
 16  sections          58472 non-null  int64  
 17  popularity        58472 non-null  int64  
 18  decade            58472 non-null  object 
 19  genre             58472 non-null  object 
dtypes: float64(10), int64(6), object(4)
memory usage: 9.4+ MB
```

2.1.1 Feature Description Definitions

There are 58,472 rows in the merged dataframe

Features
Source: [https://developer.spotify.com/documentation/web-api/reference/#/operations/get-audio-features (https://developer.spotify.com/documentation/web-api/reference/#/operations/get-audio-features)](https://developer.spotify.com/documentation/web-api/reference/#/operations/get-audio-features)
*** used for current model
single asterisk - will be used for future models

1. **danceability** ***
   A value of 0.0 is least danceable and 1.0 is most danceable. Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity.

2. **energy** *
   Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.

3. **key**
   The key the track is in. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on. If no key was detected, the value is -1. (>= -1, <= 11).

4. **loudness**
   Values typically range between -60 and 0 db. The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude).

5. **mode**
   Major is represented by 1 and minor is 0. Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived.

6. **speechiness** *
   Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.

7. **acousticness** *
   A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic. (>= 0, <= 1).

8. **instrumentalness** *
   Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.

9. **liveness** *
   Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.

10. **valence** *
    A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).(>= 0, <= 1)

11. **tempo**
    The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.

12. **duration_ms**
    The duration of the track in milliseconds.

13. **time_signature**
    An estimated time signature. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure). The time signature ranges from 3 to 7 indicating time signatures of "3/4", to "7/4". (>= 3, <= 7).

14. **id**
    The Spotify ID for the track.

15. **uri**
    The Spotify URI for the track.

2.1.2 Clean Data

In [13]:
```
# Check for duplicates

df.duplicated().sum()
```
executed in 51ms, finished 17:07:26 2022-05-05

Out[13]: 10656

In [14]:
```
# Drop duplicates

df = df.drop_duplicates()
```
executed in 55ms, finished 17:07:26 2022-05-05

In [15]:
```
# Check there are no duplicates remaining

df.duplicated().sum()
```
executed in 42ms, finished 17:07:26 2022-05-05

Out[15]: 0

In [16]:
```
# Check sum of Missing (NaN) values

df.isna().sum()
```
executed in 12ms, finished 17:07:26 2022-05-05

Out[16]:
```
artist              0
uri                 0
danceability        0
energy              0
key                 0
loudness            0
mode                0
speechiness         0
acousticness        0
instrumentalness    0
liveness            0
valence             0
tempo               0
duration_ms         0
time_signature      0
chorus_hit          0
sections            0
popularity          0
decade              0
genre               0
dtype: int64
```

In [17]:
```
# Create formula to observe percentages of the values missing

df_missing = df.isna().sum()
df_missing/len(df)
```
executed in 15ms, finished 17:07:26 2022-05-05

```
uri                 0.0
danceability        0.0
energy              0.0
key                 0.0
loudness            0.0
mode                0.0
speechiness         0.0
acousticness        0.0
instrumentalness    0.0
liveness            0.0
valence             0.0
tempo               0.0
duration_ms         0.0
time_signature      0.0
chorus_hit          0.0
sections            0.0
popularity          0.0
decade              0.0
genre               0.0
dtype: float64
```

In [18]:
```
# Check data types in latest dataframe

df.info()
```
executed in 15ms, finished 17:07:26 2022-05-05

```
 3   energy            47816 non-null  float64
 4   key               47816 non-null  int64
 5   loudness          47816 non-null  float64
 6   mode              47816 non-null  int64
 7   speechiness       47816 non-null  float64
 8   acousticness      47816 non-null  float64
 9   instrumentalness  47816 non-null  float64
 10  liveness          47816 non-null  float64
 11  valence           47816 non-null  float64
 12  tempo             47816 non-null  float64
 13  duration_ms       47816 non-null  int64
 14  time_signature    47816 non-null  int64
 15  chorus_hit        47816 non-null  float64
 16  sections          47816 non-null  int64
 17  popularity        47816 non-null  int64
 18  decade            47816 non-null  object
 19  genre             47816 non-null  object
dtypes: float64(10), int64(6), object(4)
memory usage: 7.7+ MB
```

In [19]:
```
# Explore "Artist" column

df['artist'].value_counts()
```
executed in 12ms, finished 17:07:26 2022-05-05

Out[19]:
```
Traditional                                215
Harry Belafonte                            147
Antônio Carlos Jobim                       130
P. Susheela                                129
Ennio Morricone                            124
                                          ...
Kimara Lovelace                              1
Timbaland & Magoo                            1
Cashmere Cat Featuring Ariana Grande         1
Diamond Platnumz                             1
Don Nix                                      1
Name: artist, Length: 11847, dtype: int64
```

In [20]:
```
# Percentages of Artists' counts

df['artist'].value_counts(normalize=True)
```

executed in 12ms, finished 17:07:26 2022-05-05

Out[20]:
```
Traditional                                0.004496
Harry Belafonte                            0.003074
Antônio Carlos Jobim                       0.002719
P. Susheela                                0.002698
Ennio Morricone                            0.002593
                                             ...
Kimara Lovelace                            0.000021
Timbaland & Magoo                          0.000021
Cashmere Cat Featuring Ariana Grande       0.000021
Diamond Platnumz                           0.000021
Don Nix                                    0.000021
Name: artist, Length: 11847, dtype: float64
```

In [21]:
```
# Explore the value counts of each feature

for col in df.columns:
    print(df[col].value_counts())
```

executed in 72ms, finished 17:07:26 2022-05-05

```
Traditional                                215
Harry Belafonte                            147
Antônio Carlos Jobim                       130
P. Susheela                                129
Ennio Morricone                            124
                                          ...
Kimara Lovelace                              1
Timbaland & Magoo                            1
Cashmere Cat Featuring Ariana Grande         1
Diamond Platnumz                             1
Don Nix                                      1
Name: artist, Length: 11847, dtype: int64
spotify:track:3y4LxiYMgDl4RethdzpmNe     8
spotify:track:0jsANwwkkHyyeNyuTFq2XO     8
spotify:track:756YOXmKh2iUnx33nAdfPf     8
spotify:track:22ML0MuFKfw16WejbxsLOy     8
spotify:track:6HSqyfGnsHYw9MmIpa9zlZ     8
                                        ..
spotify:track:59wdeLZoQ0AY56JkxyTyMF     1
spotify:track:40riOy7x9W7GXjyGp4pjAv     1
spotify:track:2QjOHCTQ1Jl3zawyYOpxh6     1
spotify:track:1Y4ZdPOOgCUhBcKZOrUFiS     1
spotify:track:1OdFAYq591cuGvEu5wSPIA     1
Name: uri, Length: 40160, dtype: int64
0.6200    142
0.6520    133
0.5830    129
0.6570    128
0.6000    128
         ...
0.0983      1
0.0651      1
0.0597      1
0.0991      1
0.0882      1
Name: danceability, Length: 1041, dtype: int64
0.93700    95
0.72700    94
0.64100    91
0.79100    88
0.68100    87
           ..
0.00268     1
0.00696     1
0.06680     1
0.01110     1
0.00383     1
Name: energy, Length: 1762, dtype: int64
0     5918
7     5786
```

```
2     5290
9     5132
5     4464
4     3868
1     3842
11    3313
10    3186
8     2778
6     2582
3     1657
Name: key, dtype: int64
-17.135    36
-8.142     16
-6.215     16
-8.279     16
-6.293     15
           ..
-16.881     1
-28.526     1
-23.839     1
-16.670     1
-20.000     1
Name: loudness, Length: 16012, dtype: int64
1    33205
0    14611
Name: mode, dtype: int64
0.0330    196
0.0295    194
0.0315    192
0.0306    191
0.0298    191
         ...
0.7990      1
0.5760      1
0.5650      1
0.4970      1
0.7580      1
Name: speechiness, Length: 1344, dtype: int64
0.995000    112
0.994000     98
0.993000     90
0.990000     86
0.992000     85
           ...
0.000070      1
0.000057      1
0.008910      1
0.000893      1
0.009060      1
Name: acousticness, Length: 4192, dtype: int64
0.000000    13951
0.893000       49
0.908000       44
0.903000       44
0.553000       44
            ...
0.000009        1
```

```
0.007200        1
0.071400        1
0.008440        1
0.005700        1
Name: instrumentalness, Length: 5118, dtype: int64
0.1110    462
0.1070    439
0.1100    432
0.1140    422
0.1040    404
         ...
0.0167      1
0.0278      1
0.6370      1
0.9990      1
0.0292      1
Name: liveness, Length: 1674, dtype: int64
0.9610    257
0.9620    206
0.9630    199
0.9640    171
0.9600    150
         ...
0.0209      1
0.0272      1
0.0450      1
0.0908      1
0.0269      1
Name: valence, Length: 1599, dtype: int64
142.187    36
119.993    17
119.987    15
119.989    14
94.997     12
           ..
109.516     1
124.133     1
84.714      1
129.777     1
119.228     1
Name: tempo, Length: 31894, dtype: int64
321853    36
228867    19
212933    17
218947    17
164000    16
          ..
180864     1
196302     1
247497     1
277680     1
327680     1
Name: duration_ms, Length: 21347, dtype: int64
4    42441
3     4330
5      643
1      396
```

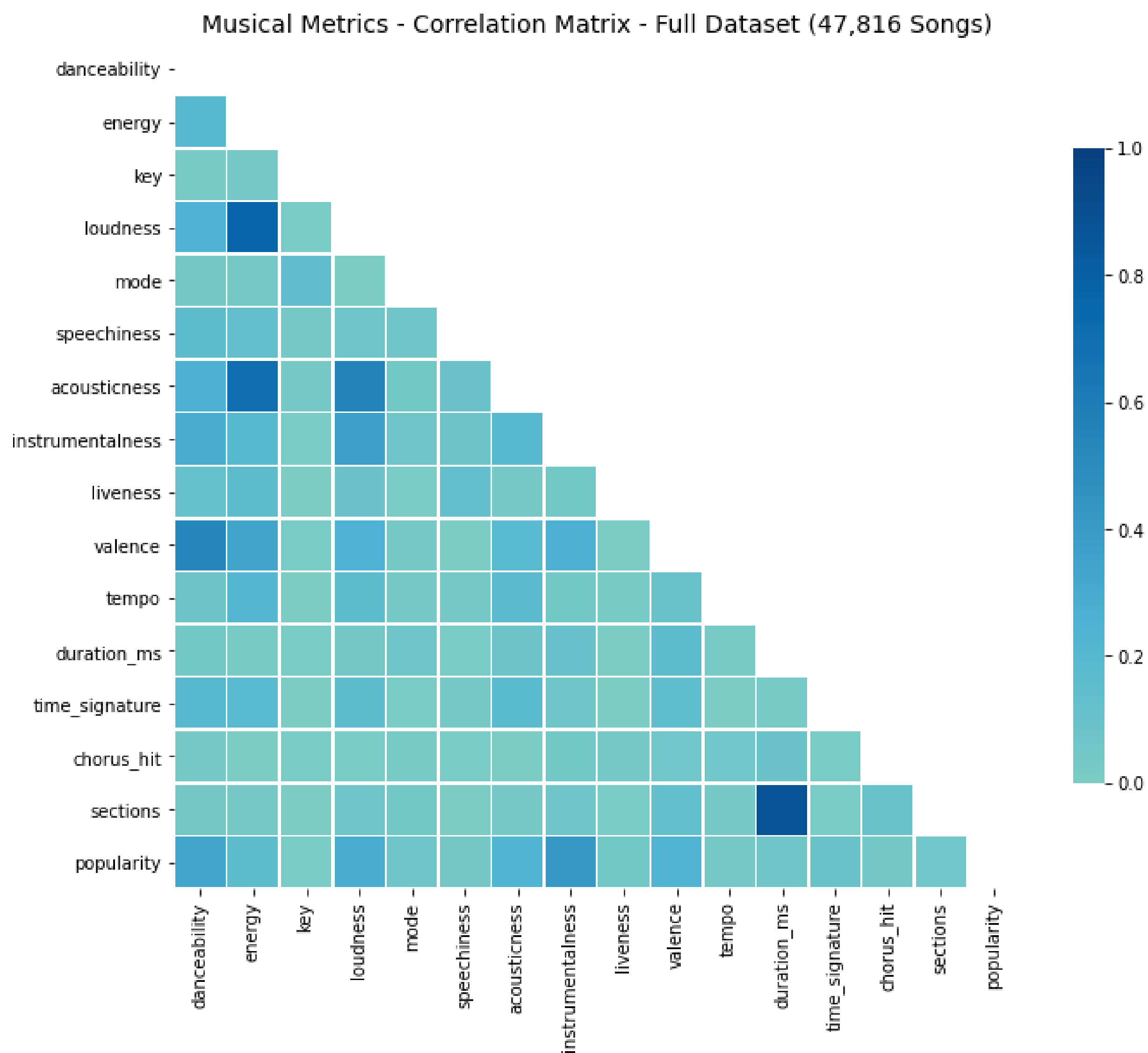
```
0        6
Name: time_signature, dtype: int64
0.00000     169
60.94077     36
41.37868      9
36.66328      8
26.28229      8
           ...
42.52036      1
58.48824      1
42.13211      1
27.50186      1
40.05079      1
Name: chorus_hit, Length: 39563, dtype: int64
9      6596
10     6215
8      5711
11     5440
7      4305
       ...
54        1
76        1
101       1
82        1
159       1
Name: sections, Length: 84, dtype: int64
1    25723
0    22093
Name: popularity, dtype: int64
60s    9717
70s    8835
80s    8140
10s    7664
00s    6929
90s    6531
Name: decade, dtype: int64
pop      18527
r&b      12927
rock      7730
latin     3746
rap       2872
edm       2014
Name: genre, dtype: int64
```

2.1.3 Data Visualization

2.1.3.1 Correlation Matrix of all metrics - Full Dataset (47,816 songs)

In [117]:
```
# Create a correlation matrix for FULL DATASET
corr = df.corr().abs()

# Create mask for upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
# GnBu is your color preference
sns.heatmap(corr, mask=mask, cmap="GnBu", vmin=0, vmax=1.0, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .75})

plt.title("Musical Metrics - Correlation Matrix - Full Dataset (47,816 Song
          fontsize=14);
```

executed in 318ms, finished 05:30:19 2022-05-06

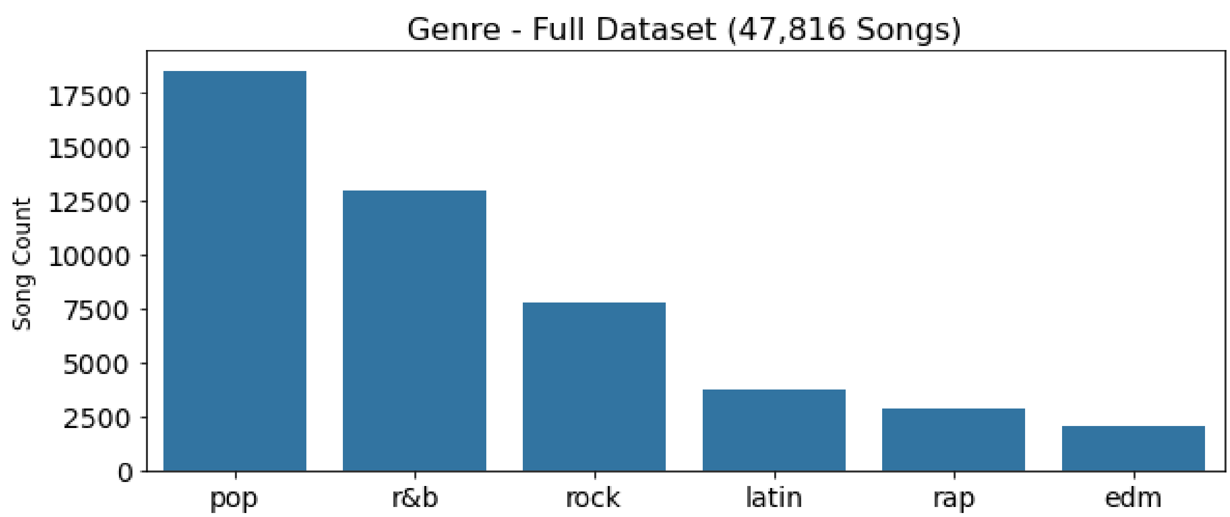


2.1.3.2 Genre Countplot - Full Dataset (47,816 songs)

In [116]:

```
# Genre countplot for FULL DATASET

fig, ax = plt.subplots(figsize=(10,4))
ax.grid(False)

sns.countplot(x='genre',
              data=df,
              order = df['genre'].value_counts().index,
              color='tab:blue')


plt.xlabel(None)
plt.ylabel("Song Count", fontsize=12)
plt.title("Genre - Full Dataset (47,816 Songs)",fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
plt.tight_layout();
```

executed in 145ms, finished 05:29:46 2022-05-06




```
<Figure size 432x288 with 0 Axes>
```

2.1.3.3 HOLD for more visualizations

In [24]:
```
# for col in df.columns:
#     fig,ax=plt.subplots(figsize=(8,4))
#     if col!='artist' or 'track':
#         sns.countplot(x=col,data=df,ax=ax,color='tab:blue')
#     else:
#         sns.histplot(x=col,data=df,ax=ax,color='tab:blue')
#     ax.set(title=col.title())
#     plt.show()
```
executed in 2ms, finished 17:07:27 2022-05-05

In [25]:
```
# for col in df.columns:
#     fig,ax=plt.subplots(figsize=(8,4))
#     sns.countplot(x=col,data=df,ax=ax,color='tab:blue')
#     ax.set(title=col.title())
#     plt.show()
```
executed in 2ms, finished 17:07:27 2022-05-05

In [26]:
```
# for col in df.columns:
#     plt.figure(figsize=(12,8))
#     sns.displot(df[col],bins=20)
#     plt.title(col)
#     plt.show();
```
executed in 2ms, finished 17:07:27 2022-05-05

In [27]:
```
# for col in df.columns:
#      fig,ax=plt.subplots(figsize=(8,4))
#      if col!='uri' or 'artist':
#          sns.countplot(x=col,data=df,ax=ax,color='tab:blue')
#      #else:
#      #    sns.histplot(x=col,data=df,ax=ax,color='tab:blue')
#      ax.set(title=col.title())
#      plt.show()
```
executed in 2ms, finished 17:07:27 2022-05-05

3 Preprocessing Data

Create Clean Sample of Data

3.1 Create Clean Sample of Data

In [28]:
```
# Create sample of 1000 songs from 47,816 songs

sample = df.sample(n=1000,replace=False, random_state=11).reset_index()
```
executed in 6ms, finished 17:07:27 2022-05-05

In [29]:
```
# View sample dataframe

sample.head()
```
executed in 19ms, finished 17:07:27 2022-05-05

Out[29]:

	track	artist	uri	danceability	energy	key	loudness
0	Rock And Roll Dreams Come Through	Jim Steinman	spotify:track:5Y7JlzuX1CtyEl8qf58qeU	0.628	0.6370	0	-13.175
1	Peace Will Come (According To Plan)	Melanie	spotify:track:1lMhE01kAot77D8M17ac3m	0.370	0.2950	8	-7.307
2	Let It Happen	Vangelis	spotify:track:59HzNVTc331SYrl6vQEJJQ	0.349	0.4920	2	-13.886
3	Keeps Gettin' Better	Christina Aguilera	spotify:track:0j0n5CUS1g3QSwDWg8r5qq	0.645	0.6970	5	-4.733
4	Aubrey	Bread	spotify:track:3his1UkcI0rwrniPDR9kTj	0.326	0.0902	7	-20.588

5 rows × 21 columns

In [30]:
```
# Explore sample data (count, datatypes)

sample.info()
```
executed in 8ms, finished 17:07:27 2022-05-05

```
 5   key               1000 non-null   int64
 6   loudness          1000 non-null   float64
 7   mode              1000 non-null   int64
 8   speechiness       1000 non-null   float64
 9   acousticness      1000 non-null   float64
 10  instrumentalness  1000 non-null   float64
 11  liveness          1000 non-null   float64
 12  valence           1000 non-null   float64
 13  tempo             1000 non-null   float64
 14  duration_ms       1000 non-null   int64
 15  time_signature    1000 non-null   int64

 16  chorus_hit        1000 non-null   float64
 17  sections          1000 non-null   int64
 18  popularity        1000 non-null   int64
 19  decade            1000 non-null   object
 20  genre             1000 non-null   object
dtypes: float64(10), int64(6), object(5)
memory usage: 164.2+ KB
```

3.1.1 Convert Sample Dataframe into a csv file for Modeling - run once in intial build

- keep code for reference

In [31]:
```
# Code used to convert Sample dataframe into a csv file for modeling

# sample.to_csv(r'Sample.csv')
```
executed in 2ms, finished 17:07:27 2022-05-05

3.1.2 Create "url" Column from "uri" Column to Retrieve Songs from Spotify

In [32]:
```
# Create "url" column from "uri" column

sample['url'] = sample['uri'].map(lambda x: x.lstrip('spotify:track:'))
```
executed in 3ms, finished 17:07:27 2022-05-05

In [33]:
```
# Check new "url" column

sample.head()
```
executed in 19ms, finished 17:07:27 2022-05-05

Out[33]:

	track	artist	uri	danceability	energy	key	loudness
0	Rock And Roll Dreams Come Through	Jim Steinman	spotify:track:5Y7JlzuX1CtyEl8qf58qeU	0.628	0.6370	0	-13.175
1	Peace Will Come (According To Plan)	Melanie	spotify:track:1lMhE01kAot77D8M17ac3m	0.370	0.2950	8	-7.307
2	Let It Happen	Vangelis	spotify:track:59HzNVTc331SYrl6vQEJJQ	0.349	0.4920	2	-13.886
3	Keeps Gettin' Better	Christina Aguilera	spotify:track:0j0n5CUS1g3QSwDWg8r5qq	0.645	0.6970	5	-4.733
4	Aubrey	Bread	spotify:track:3his1UkcI0rwrniPDR9kTj	0.326	0.0902	7	-20.588

5 rows × 22 columns

In [34]:
```
# Create "url" column with 'https://open.spotify.com/track/' format to retr

sample['url'] = 'https://open.spotify.com/track/' + sample['url']
```

executed in 3ms, finished 17:07:27 2022-05-05

In [35]:
```
# View dataframe with "url" column

sample.head()
```

executed in 19ms, finished 17:07:27 2022-05-05

uri	danceability	energy	key	loudness	mode	speechiness	acousticness	...	valence	temp
Y7JlzuX1CtyEl8qf58qeU	0.628	0.6370	0	-13.175	1	0.0294	0.1510	...	0.755	110.4
ıE01kAot77D8M17ac3m	0.370	0.2950	8	-7.307	1	0.0278	0.5670	...	0.269	132.4
IzNVTc331SYrl6vQEJJQ	0.349	0.4920	2	-13.886	0	0.0465	0.6990	...	0.503	106.0
CUS1g3QSwDWg8r5qq	0.645	0.6970	5	-4.733	0	0.0285	0.0739	...	0.250	130.0
3his1UkclOrwrniPDR9kTi	0.326	0.0902	7	-20.588	1	0.0344	0.6470	...	0.218	137.6

```
### BUILD CHECKLIST & CLEAN DATA TO CREATE USABLE DATASET
There are 652 songs in final dataset to be used for model (653 minus
".ds store" file)
```

3.1.3 BUILD CHECKLIST & CLEAN DATA TO CREATE USABLE DATASET

There are 652 songs in final dataset to be used for model (653 minus ".ds store" file)

In [36]:
```
# Create a "checklist" column from "track" and "artist" columns to cross-ch

sample['checklist'] = sample['artist'] + " - " + sample['track'] + ".mp3"
sample.head()
```
executed in 21ms, finished 17:07:27 2022-05-05

Out[36]:

	track	artist	uri	danceability	energy	key	loudness
0	Rock And Roll Dreams Come Through	Jim Steinman	spotify:track:5Y7JlzuX1CtyEl8qf58qeU	0.628	0.6370	0	-13.175
1	Peace Will Come (According To Plan)	Melanie	spotify:track:1lMhE01kAot77D8M17ac3m	0.370	0.2950	8	-7.307
2	Let It Happen	Vangelis	spotify:track:59HzNVTc331SYrl6vQEJJQ	0.349	0.4920	2	-13.886
3	Keeps Gettin' Better	Christina Aguilera	spotify:track:0j0n5CUS1g3QSwDWg8r5qq	0.645	0.6970	5	-4.733
4	Aubrey	Bread	spotify:track:3his1UkcI0rwrniPDR9kTj	0.326	0.0902	7	-20.588

5 rows × 23 columns

In [37]:
```
# Check object using one song

sample[sample['artist'] == 'Johnny Sea']['checklist'].values
```
executed in 5ms, finished 17:07:27 2022-05-05

Out[37]:
```
array(['Johnny Sea - Day For Decision.mp3'], dtype=object)
```

In [38]:
```
# Check for tracks not in "checklist" column

counter = 0
for track_name in os.listdir('Song_Data'):
    if track_name == '.DS_Store':
        continue
    if track_name not in sample['checklist'].values:
        print(track_name)
        artist, title = track_name.split('.mp3')[0].split('-')[:2]
        artist, title = artist.strip(), title.strip()

        print(f'Artist: {artist}\tTitle: {title}')
        display(sample[sample['artist'] == artist])
        print('-'*40)
        counter += 1
        print(counter)
```

executed in 3.34s, finished 17:07:30 2022-05-05

```
26
Roberta Flack, Donny Hathaway - You've Lost That Lovin' Feelin'.mp3
Artist: Roberta Flack, Donny Hathaway   Title: You've Lost That Lovi
n' Feelin'
```

track	artist	uri	danceability	energy	key	loudness	mode	speechiness	acousticness	...

0 rows × 23 columns

```
----------------------------------------
27
6ix9ine, Nicki Minaj, Murda Beatz - FEFE.mp3
Artist: 6ix9ine, Nicki Minaj, Murda Beatz       Title: FEFE
```

track	artist	uri	danceability	energy	key	loudness	mode	speechiness	acousticness	...

In [40]:
```
# Find songs that do not align with "checklist" (DIRTYDATA)

DIRTYDATA = []
for idx, data in sample.iterrows():
    if data['checklist'] not in os.listdir('Song_Data'):
        DIRTYDATA.append(idx)
DIRTYDATA
```

executed in 1.01s, finished 17:07:31 2022-05-05

```
 743,
 744,
 745,
 749,
 751,
 752,
 759,
 760,
 763,
 765,
 766,
 768,
 770,
 772,
 774,
 777,
 780,
 783,
 784,
 785.
```

In [41]:
```
# Drop DIRTYDATA from data to get USABLE data

USABLE = sample.drop(DIRTYDATA)
USABLE.shape
```

executed in 4ms, finished 17:07:31 2022-05-05

Out[41]:
```
(653, 23)
```

In [42]:
```
# View USABLE dataframe

USABLE
```
executed in 26ms, finished 17:07:31 2022-05-05

Out[42]:

	track	artist	uri	danceability	energy	key	l
2	Let It Happen	Vangelis	spotify:track:59HzNVTc331SYrI6vQEJJQ	0.349	0.4920	2	
3	Keeps Gettin' Better	Christina Aguilera	spotify:track:0j0n5CUS1g3QSwDWg8r5qq	0.645	0.6970	5	
4	Aubrey	Bread	spotify:track:3his1UkcI0rwrniPDR9kTj	0.326	0.0902	7	
5	Most Of All	B.J. Thomas	spotify:track:4GPF6wnqZSBtEBUuSxHivV	0.501	0.3920	9	
6	High Speed GTO	White Wizzard	spotify:track:4AZRFiO74C2HwRVePGrmR2	0.252	0.9410	6	
...	...	...	...	...	...	...	
975	Candy	Mandy Moore	spotify:track:2YhE6xeWN0R9RVwEOG9IR1	0.813	0.8360	7	
993	Arthur Comes to Sophie	Hildur Guðnadóttir	spotify:track:0dvAO2KbsqDZGv8g03JFRy	0.198	0.3300	0	
995	Guantanamera	Joe Dassin	spotify:track:2zo7m7HTcjMuioTTrIt4yF	0.716	0.4410	2	
996	Let Me In	Young Buck	spotify:track:6qkZ6D3ogNyW2YDWsz7e3z	0.685	0.8900	1	
997	Superfly	Curtis Mayfield	spotify:track:4XsH9zBWPOCdXoH9ZDdS8r	0.784	0.7080	2	

653 rows × 23 columns

In [43]:
```
# Explore USABLE info

USABLE.info()
```

executed in 7ms, finished 17:07:31 2022-05-05

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 653 entries, 2 to 997
Data columns (total 23 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   track             653 non-null    object 
 1   artist            653 non-null    object 
 2   uri               653 non-null    object 
 3   danceability      653 non-null    float64
 4   energy            653 non-null    float64
 5   key               653 non-null    int64  
 6   loudness          653 non-null    float64
 7   mode              653 non-null    int64  
 8   speechiness       653 non-null    float64
 9   acousticness      653 non-null    float64
 10  instrumentalness  653 non-null    float64
 11  liveness          653 non-null    float64
 12  valence           653 non-null    float64
 13  tempo             653 non-null    float64
 14  duration_ms       653 non-null    int64  
 15  time_signature    653 non-null    int64  
 16  chorus_hit        653 non-null    float64
 17  sections          653 non-null    int64  
 18  popularity        653 non-null    int64  
 19  decade            653 non-null    object 
 20  genre             653 non-null    object 
 21  url               653 non-null    object 
 22  checklist         653 non-null    object 
dtypes: float64(10), int64(6), object(7)
memory usage: 122.4+ KB
```

3.1.4 Create ".png" Column for Images

In [44]:
```
# Create 'songpng' column for .mp3 files to be connected to .png files in m

USABLE['songpng'] = USABLE['checklist'].apply(lambda x: x.replace('.mp3','.
```

executed in 3ms, finished 17:07:31 2022-05-05

In [45]:

```
# Check

USABLE.head()
```

executed in 18ms, finished 17:07:31 2022-05-05

Out[45]:

	popularity	decade	genre	url	checklist	songpng
	0	70s	rock	https://open.spotify.com/track/59HzNVTc331SYrl...	Vangelis - Let It Happen.mp3	Vangelis - Let It Happen.png
	1	00s	pop	https://open.spotify.com/track/0j0n5CUS1g3QSwD...	Christina Aguilera - Keeps Gettin' Better.mp3	Christina Aguilera - Keeps Gettin' Better.png
	1	70s	rock	https://open.spotify.com/track/3his1UkcI0rwrni...	Bread - Aubrey.mp3	Bread - Aubrey.png
	1	70s	r&b	https://open.spotify.com/track/4GPF6wnqZSBtEBU...	B.J. Thomas - Most Of All.mp3	B.J. Thomas - Most Of All.png
	0	00s	rock	https://open.spotify.com/track/4AZRFiO74C2HwRV...	White Wizzard - High Speed GTO.mp3	White Wizzard - High Speed GTO.png

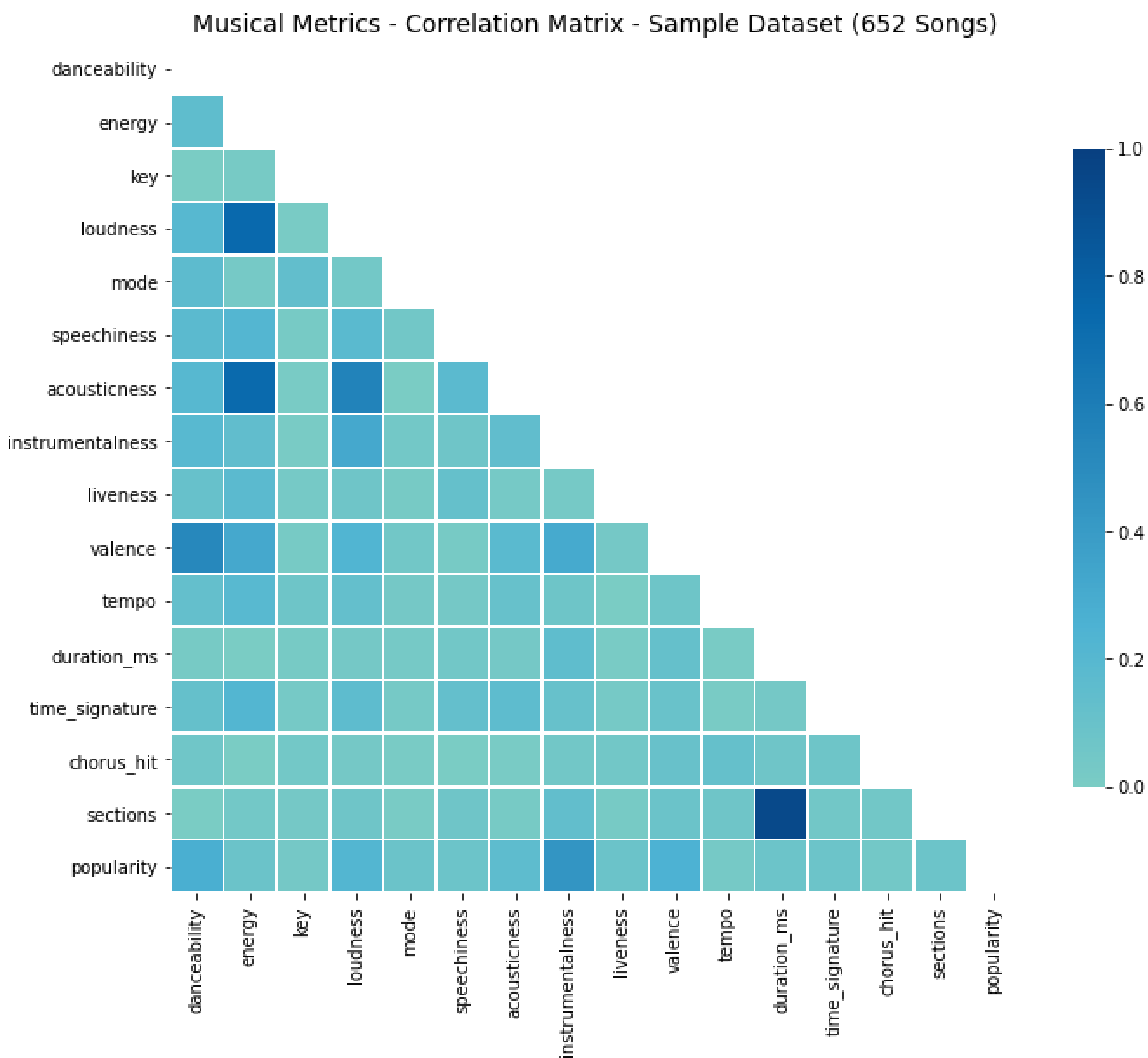
In [46]:
```
# Check datatypes

USABLE.info()
```
executed in 9ms, finished 17:07:31 2022-05-05

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 653 entries, 2 to 997
Data columns (total 24 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   track             653 non-null    object 
 1   artist            653 non-null    object 
 2   uri               653 non-null    object 
 3   danceability      653 non-null    float64
 4   energy            653 non-null    float64
 5   key               653 non-null    int64  
 6   loudness          653 non-null    float64
 7   mode              653 non-null    int64  
 8   speechiness       653 non-null    float64
 9   acousticness      653 non-null    float64
 10  instrumentalness  653 non-null    float64
 11  liveness          653 non-null    float64
 12  valence           653 non-null    float64
 13  tempo             653 non-null    float64
 14  duration_ms       653 non-null    int64  
 15  time_signature    653 non-null    int64  
 16  chorus_hit        653 non-null    float64
 17  sections          653 non-null    int64  
 18  popularity        653 non-null    int64  
 19  decade            653 non-null    object 
 20  genre             653 non-null    object 
 21  url               653 non-null    object 
 22  checklist         653 non-null    object 
 23  songpng           653 non-null    object 
dtypes: float64(10), int64(6), object(8)
memory usage: 127.5+ KB
```

3.1.5 Visualizations for Sample Dataset

In [115]:
```
# Correlation matrix for SAMPLE DATASET
corr = USABLE.corr().abs()

# Create a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
# GnBu is your color preference
sns.heatmap(corr, mask=mask, cmap="GnBu", vmin=0, vmax=1.0, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .75})

# Set title
plt.title("Musical Metrics - Correlation Matrix - Sample Dataset (652 Songs
          fontsize=14);
```

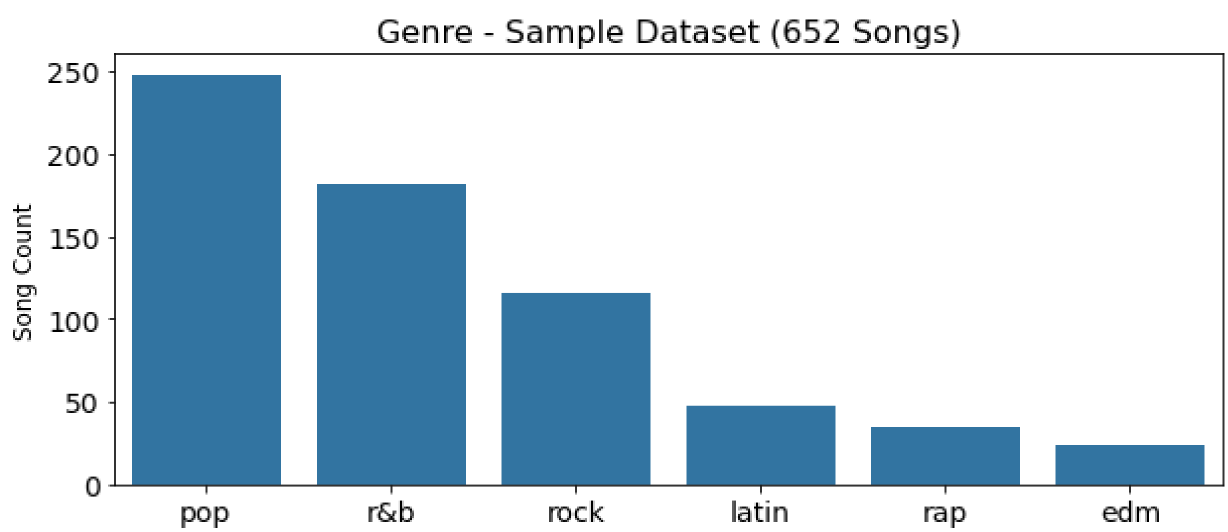
executed in 291ms, finished 04:49:55 2022-05-06



In [114]:

```
# Genre countplot for SAMPLE DATASET

fig, ax = plt.subplots(figsize=(10,4))
ax.grid(False)

sns.countplot(x='genre',
              data=USABLE,
              order = df['genre'].value_counts().index,
              color='tab:blue')

plt.xlabel(None)
plt.ylabel("Song Count", fontsize=12)
plt.title("Genre - Sample Dataset (652 Songs)",fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
plt.tight_layout();
```

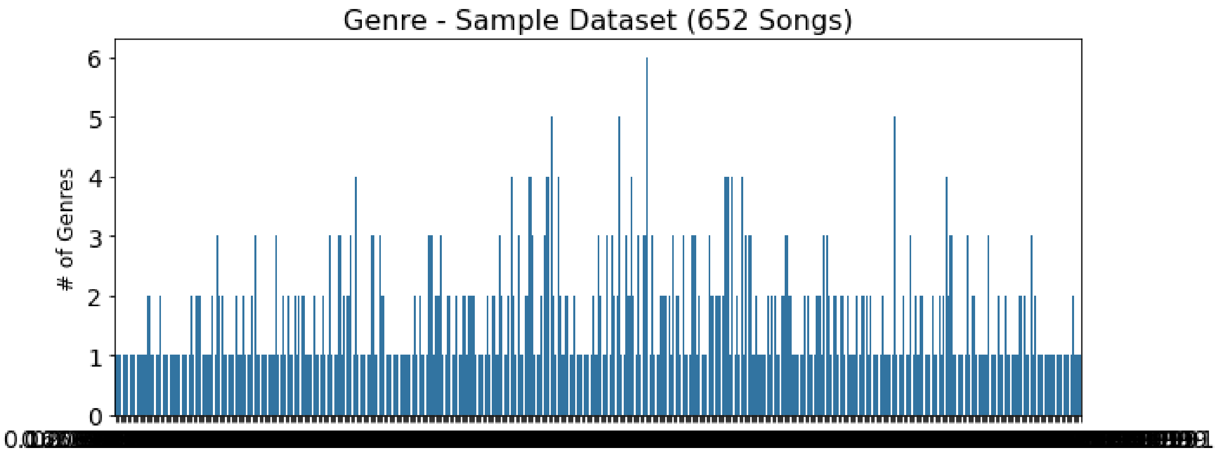
executed in 119ms, finished 04:49:24 2022-05-06




```
<Figure size 432x288 with 0 Axes>
```

In [49]:
```
# Danceability countplot for sample data - need to improve

fig, ax = plt.subplots(figsize=(10,4))
sns.countplot(x='danceability',data=USABLE, color='tab:blue');
ax.grid(False)

plt.xlabel(None)
plt.ylabel("# of Genres", fontsize=12)
plt.title("Genre - Sample Dataset (652 Songs)",fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
plt.tight_layout()
```

executed in 4.50s, finished 17:07:36 2022-05-05




```
<Figure size 432x288 with 0 Axes>
```

4 Data Preparation for Modeling

4.1 Create y (Target: "danceability") and X

In [50]:
```
# Create y (Target: "danceability")
# Create X

y = USABLE['danceability']
X = USABLE.drop(columns=['danceability'])
```

executed in 4ms, finished 17:07:36 2022-05-05

In [51]:
```python
# View X data

X
```

executed in 27ms, finished 17:07:36 2022-05-05

[illegible]	GTO	Wizzard	[illegible]	[illegible]	[illegible]	[illegible]
...	...	...	...	...	...	...
975	Candy	Mandy Moore	spotify:track:2YhE6xeWN0R9RVwEOG9IR1	0.8360	7	-4.230
993	Arthur Comes to Sophie	Hildur Guðnadóttir	spotify:track:0dvAO2KbsqDZGv8g03JFRy	0.3300	0	-15.555
995	Guantanamera	Joe Dassin	spotify:track:2zo7m7HTcjMuioTTrlt4yF	0.4410	2	-9.909
996	Let Me In	Young Buck	spotify:track:6qkZ6D3ogNyW2YDWsz7e3z	0.8900	1	-4.302
997	Superfly	Curtis Mayfield	spotify:track:4XsH9zBWPOCdXoH9ZDdS8r	0.7080	2	-9.141

653 rows × 23 columns

In [52]:
```python
# Gutcheck - check metrics on one song

USABLE[USABLE.track.str.contains('Wherever You Will Go')]
```

executed in 17ms, finished 17:07:36 2022-05-05

Out[52]:

	track	artist	uri	danceability	energy	key	loudness
517	Wherever You Will Go	The Calling	spotify:track:5QpaGzWp0hwB5faV8dkbAz	0.558	0.719	2	-5.113

1 rows × 24 columns

In [53]:
```
# View y data

y
```
executed in 4ms, finished 17:07:36 2022-05-05

Out[53]:
```
2      0.349
3      0.645
4      0.326
5      0.501
6      0.252
       ...
975    0.813
993    0.198
995    0.716
996    0.685
997    0.784
Name: danceability, Length: 653, dtype: float64
```

4.2 Train Test Split

In [54]:
```
# Create Train and Test data subsets using train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, random_state=100)
```
executed in 3ms, finished 17:07:36 2022-05-05

In [55]:
```
# Check shape of each data set

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```
executed in 4ms, finished 17:07:36 2022-05-05

Out[55]:
```
((489, 23), (164, 23), (489,), (164,))
```

In [56]:
```
# Check the shape of the data is the same

X_train.shape[0]+X_test.shape[0]==USABLE.shape[0]
```
executed in 4ms, finished 17:07:36 2022-05-05

Out[56]:
```
True
```

In [57]:
```
# View X_train

X_train
```
executed in 30ms, finished 17:07:36 2022-05-05

Out[57]:

	track	artist	uri	energy	key	loudness	mode
684	Más Allá	Javier Solís	spotify:track:2eZT2Jw3gjv8ZqBUu9oCTE	0.325	0	-11.149	1
619	Footprints - Remastered	Wayne Shorter	spotify:track:2JlTVZu8o6ls9k8SoMRy7w	0.454	7	-11.190	0
904	Rock Of Ages	Jack Jezzro	spotify:track:2U9L4wYRxRgYy42uhvOloy	0.280	7	-14.582	1
855	Do You Believe In Magic	Shaun Cassidy	spotify:track:5LJ93CrqstdBdVmC0xhZbu	0.726	0	-10.154	1
318	People Like You	Eddie Fisher	spotify:track:6cahHUfSQDIB8i0Yx3srwx	0.339	0	-8.351	1
...	...	...	...	...	...	...	...
854	Dangerous	Roxette	spotify:track:756YOXmKh2iUnx33nAdfPf	0.898	4	-4.893	1
72	Barefoot In Baltimore	Strawberry Alarm Clock	spotify:track:7gxeDaqGLT33dkWSTAEOue	0.566	7	-11.186	1
517	Wherever You Will Go	The Calling	spotify:track:5QpaGzWp0hwB5faV8dkbAz	0.719	2	-5.113	1
109	Milagre Brasileiro - Ao Vivo	MPB4	spotify:track:7gluxKYkMdLREvbCrXdGQh	0.675	2	-8.183	1
771	Say You Really Want Me	Kim Wilde	spotify:track:1lemomv6vJ9UcHxMRDINMJ	0.642	10	-13.852	0

489 rows × 23 columns

In [58]:
```
# View y_train

y_train
```
executed in 3ms, finished 17:07:36 2022-05-05

Out[58]:
```
684    0.399
619    0.530
904    0.275
855    0.499
318    0.490
       ...
854    0.712
72     0.682
517    0.558
109    0.368
771    0.699
Name: danceability, Length: 489, dtype: float64
```

4.3 Create Images for Songs to be Modeled

In [59]:
```
# Check directory of songs

os.listdir('Song_Data')
```
executed in 13ms, finished 17:07:36 2022-05-05

```
 'Andrew Bird - Why.mp3',
 'Jake Owen - I Was Jack (You Were Diane).mp3',
 "Frankie Valli & The Four Seasons - C'mon Marianne - 2006 Remaster.m
p3",
 'The Motels - Only The Lonely - Remastered 1999.mp3',
 "Karlheinz Stockhausen - 11'-38'.mp3",
 'Ashra - Ocean Of Tenderness.mp3',
 'Buzzcocks - What Do I Get - 2001 Remastered Version.mp3',
 'Earth, Wind & Fire - Kalimba Story.mp3',
 'The Bubble Puppy - Hot Smoke & Sasafrass (Live Version).mp3',
 'Trans-Siberian Orchestra - Christmas Eve  Sarajevo 1224 - Instrumen
tal.mp3',
 'Job For A Cowboy - Knee Deep.mp3',
 'The Vejtables - I Still Love You.mp3',
 'MPB4 - Milagre Brasileiro - Ao Vivo.mp3',
 'Billy Joel, Ray Charles - Baby Grand (with Ray Charles).mp3',
 'Ben Colder - Almost Persuaded No. 2.mp3',
 "David Banner, Lil' Flip - Like A Pimp.mp3",
 'Jeff Lewis, Mitchell Hope, Disney - Did I Mention.mp3',
 'Eric Burdon & the Animals - See See Rider.mp3',
```

In [60]:
```
# Check number of songs in directory

len(os.listdir('Song_Data'))
```

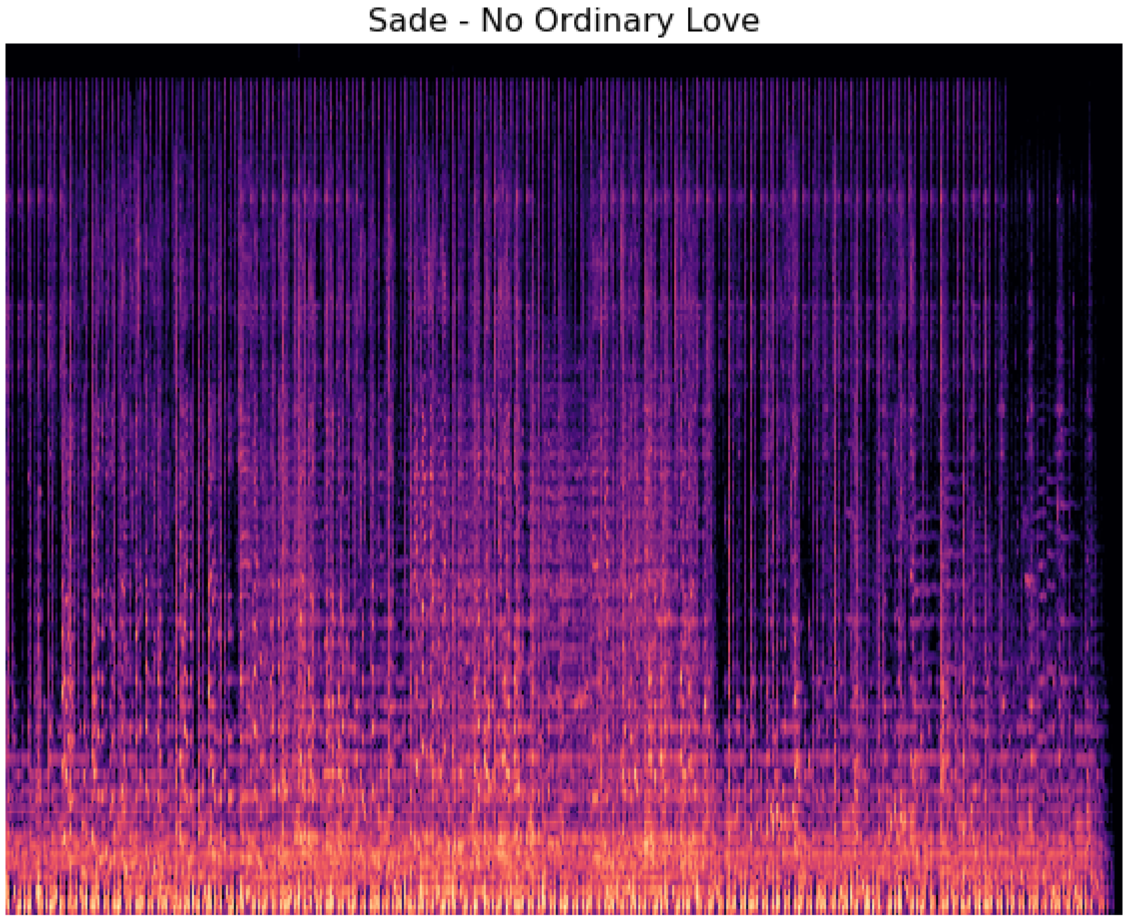
executed in 3ms, finished 17:07:36 2022-05-05

Out[60]: 953

In [61]:
```
# CREATE IMAGE FOR ONE SONG (I.E. TEST WITHOUT FOR LOOP)
# EXAMPLE: Sade - No Ordinary Love

SONG_DATA = os.listdir('Song_Data')

# Instantiate constants (taken from source:)
SAMPLE_RATE = 48000
HOP_LENGTH = 256
N_FFT = 2048
N_MELS = 256
REF = np.max

fpath = 'Song_Data/Sade - No Ordinary Love.mp3'

# Load song into memory
signal, sr = librosa.load(fpath, sr=SAMPLE_RATE)

# Create "mel-spectrogram"
mel_signal = librosa.feature.melspectrogram(
    y=signal, # Created above
    sr=SAMPLE_RATE, # Stuff we decided at the top:
    hop_length=HOP_LENGTH,
    n_fft=N_FFT,
    n_mels=N_MELS)

power_to_db = librosa.power_to_db(mel_signal, ref=REF)

# Create figure
fig = plt.figure(figsize=(10,8))
ax = fig.add_subplot(111)

# Hide axes and image frame
ax.axes.get_xaxis().set_visible(False)
ax.axes.get_yaxis().set_visible(False)
ax.set_frame_on(False)

# Display spectrogram for song
librosa.display.specshow(power_to_db, sr=SAMPLE_RATE, cmap='magma', hop_len

plt.title("Sade - No Ordinary Love",fontsize=16)
plt.show()
```

executed in 11.3s, finished 17:07:48 2022-05-05

```
/Users/v/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/libros
a/util/decorators.py:88: UserWarning: PySoundFile failed. Trying audiorea
d instead.
  return f(*args, **kwargs)
```

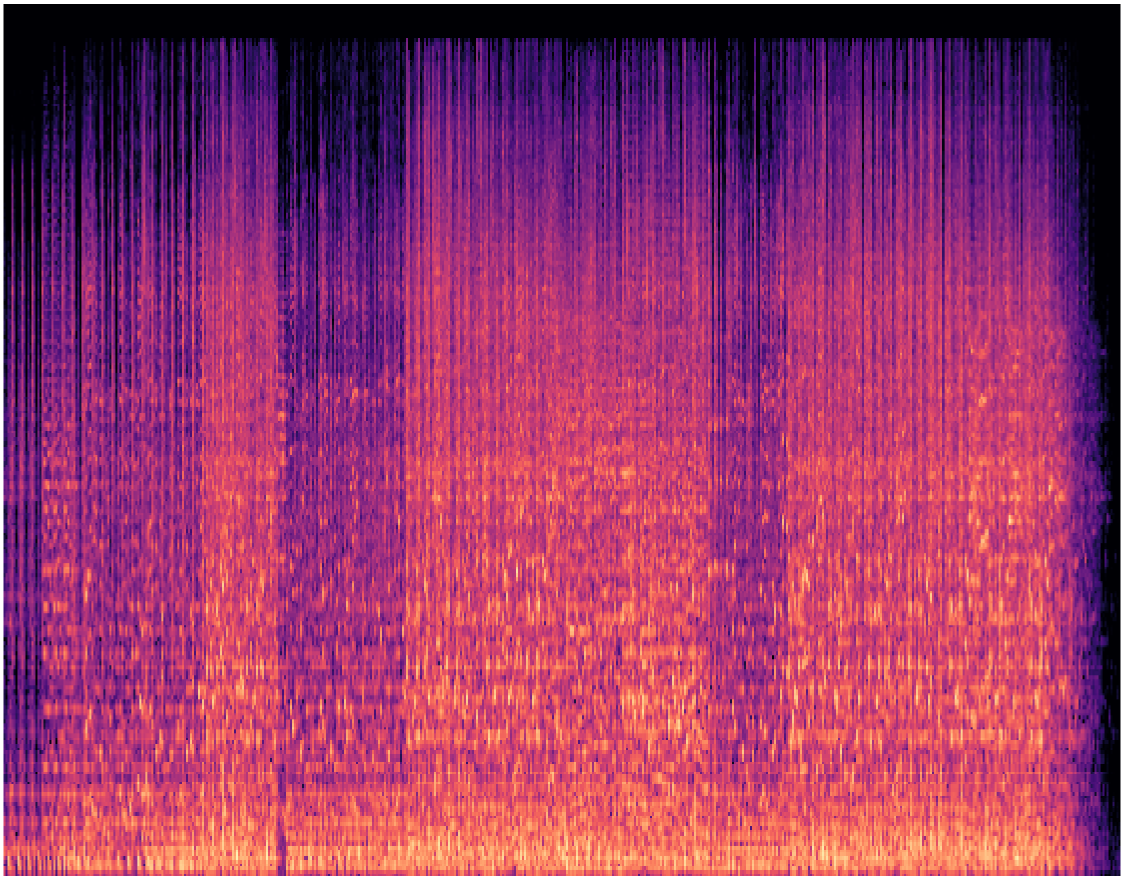

Sade - No Ordinary Love

In [62]:
```
# Check another song
# EXAMPLE: Kansas - Power

SONG_DATA = os.listdir('Song_Data')

# Instantiate constants (taken from source:)
SAMPLE_RATE = 48000
HOP_LENGTH = 256
N_FFT = 2048
N_MELS = 256
REF = np.max

fpath = 'Song_Data/Kansas - Power.mp3'

# Load song into memory
signal, sr = librosa.load(fpath, sr=SAMPLE_RATE)

# Create "mel-spectrogram"
mel_signal = librosa.feature.melspectrogram(
    y=signal, # Created above
    sr=SAMPLE_RATE, # Stuff we decided at the top:
    hop_length=HOP_LENGTH,
    n_fft=N_FFT,
    n_mels=N_MELS)

power_to_db = librosa.power_to_db(mel_signal, ref=REF)

# Create figure
fig = plt.figure(figsize=(10,8))
ax = fig.add_subplot(111)

# Hide axes and image frame
ax.axes.get_xaxis().set_visible(False)
ax.axes.get_yaxis().set_visible(False)
ax.set_frame_on(False)

# Display spectrogram for song
librosa.display.specshow(power_to_db, sr=SAMPLE_RATE, cmap='magma', hop_len

plt.title("Kansas - Power",fontsize=16)
plt.show()
```

executed in 6.80s, finished 17:07:55 2022-05-05

```
/Users/v/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/libros
a/util/decorators.py:88: UserWarning: PySoundFile failed. Trying audiorea
d instead.
  return f(*args, **kwargs)
```

Kansas - Power



CREATE AND SAVE SPECTROGRAMS FOR TRAIN AND TEST SONGS!!

4.4 CREATE AND SAVE SPECTROGRAMS FOR TRAIN AND TEST SONGS!!

Code to Create Spectrograms & Put in Train and Test Image Folders - only needs to be run once in initial build

Code for reference

4.4.1 Code to Create Spectrograms & Put in Train and Test Image Folders - only needs to be run once in initial build

Code for reference

In [63]:
```
# SONG_DATA = X['checklist'].values

# # Instantiate constants (taken from source:)
# SAMPLE_RATE = 48000
# HOP_LENGTH = 256
# N_FFT = 2048
# N_MELS = 256
# REF = np.max


# # Iterate through .mp3 files
# for mp3 in SONG_DATA:
#     #
#     if not mp3.endswith('.mp3'):
#         continue

#     # Create path to mp3.
#     fpath = os.path.join(path, mp3)
#     #fpath = 'Song_Data'

#     # Load song into memory
#     signal, sr = librosa.load(fpath, sr=SAMPLE_RATE)

#     # Create "mel-spectrogram"
#     mel_signal = librosa.feature.melspectrogram(
#         y=signal, # Created above
#         sr=SAMPLE_RATE, # Stuff we decided at the top:
#         hop_length=HOP_LENGTH,
#         n_fft=N_FFT,
#         n_mels=N_MELS
#     )
#     power_to_db = librosa.power_to_db(mel_signal, ref=REF) # Part of the

#     # Creating figure
#     fig = plt.figure(figsize=(10,8))
#     ax = fig.add_subplot(111)
#         # Hiding axes and image frame
#     ax.axes.get_xaxis().set_visible(False)
#     ax.axes.get_yaxis().set_visible(False)
#     ax.set_frame_on(False)

#         # Displaying our spectrograms
#     librosa.display.specshow(power_to_db, sr=SAMPLE_RATE, cmap='magma', h

#       # SAVE THE IMAGES IN RESPECTIVE FOLDERS
#     if mp3 in X_train['checklist'].values:
#         folder = 'Train'
#     else:
#         # Save in Images/Test/...
#         folder = 'Test'

#     plt.savefig(
#         fname=f'Images/{folder}/{mp3.split(".mp3")[0]}.png',
#         dpi=400,
#         bbox_inches='tight',
#         pad_inches=0
```

```
#     )

#     # Cleanup
#     plt.close()
#     fig.clf()
#     plt.close(fig)
#     plt.close('all')
#     print(mp3)
```

executed in 3ms, finished 17:07:55 2022-05-05

4.4.2 Resulting Train and Test Image Folders

4.5 Create Train and Test Datasets for Model

In [64]:
```
# Create Train dataset
# Concatenate X_train, y_train

Train = pd.concat([X_train, y_train], axis=1)
Train
```

executed in 32ms, finished 17:07:55 2022-05-05

Out[64]:

	track	artist	uri	energy	key	loudness	mode
684	Más Allá	Javier Solís	spotify:track:2eZT2Jw3gjv8ZqBUu9oCTE	0.325	0	-11.149	1
619	Footprints - Remastered	Wayne Shorter	spotify:track:2JITVZu8o6ls9k8SoMRy7w	0.454	7	-11.190	0
904	Rock Of Ages	Jack Jezzro	spotify:track:2U9L4wYRxRgYy42uhvOloy	0.280	7	-14.582	1
855	Do You Believe In Magic	Shaun Cassidy	spotify:track:5LJ93CrqstdBdVmC0xhZbu	0.726	0	-10.154	1
318	People Like You	Eddie Fisher	spotify:track:6cahHUfSQDIB8i0Yx3srwx	0.339	0	-8.351	1
...	...	...	...	...	...	...	...
854	Dangerous	Roxette	spotify:track:756YOXmKh2iUnx33nAdfPf	0.898	4	-4.893	1
72	Barefoot In Baltimore	Strawberry Alarm Clock	spotify:track:7gxeDaqGLT33dkWSTAEOue	0.566	7	-11.186	1
517	Wherever You Will Go	The Calling	spotify:track:5QpaGzWp0hwB5faV8dkbAz	0.719	2	-5.113	1
109	Milagre Brasileiro - Ao Vivo	MPB4	spotify:track:7gluxKYkMdLREvbCrXdGQh	0.675	2	-8.183	1
771	Say You Really Want Me	Kim Wilde	spotify:track:1lemomv6vJ9UcHxMRDINMJ	0.642	10	-13.852	0

489 rows × 24 columns

In [65]:
```
# Check shape

Train.shape
```
executed in 3ms, finished 17:07:55 2022-05-05

Out[65]:
```
(489, 24)
```

In [66]:

```
# Create Test dataset
# Concatenate X_test, y_test

Test = pd.concat([X_test, y_test], axis=1)
Test
```

executed in 32ms, finished 17:07:55 2022-05-05

Out[66]:

	track	artist	uri	energy	key	loudness	mode	spe
836	Blues Jumped a Rabbit	Sonny Boy Nelson	spotify:track:7stzHo184FVS9VdGUWIkqi	0.234	4	-14.302	1	
462	Love Will Conquer All	Lionel Richie	spotify:track:6nbi2AJ9hAi2SE8jH6mRKV	0.443	2	-10.078	0	
169	Eight	Sleeping At Last	spotify:track:1lSnBlAErRss6asu9Y5HuA	0.287	7	-10.683	0	
578	Circles	Atlantic Starr	spotify:track:0l3fOjcytIQoqnmRdkjAOx	0.690	4	-6.576	0	
67	Power	Kansas	spotify:track:0a0AgpXGzXslRztBZHyw0j	0.556	9	-12.066	1	
...	...	...	...	...	...	...	...	
181	Ooh La	The Kooks	spotify:track:6qWUtUtexTpwXJhCPFTxTr	0.874	10	-5.129	0	
584	Raga Bhimpalasi - Live	Ravi Shankar	spotify:track:2NAJP1wqoaxbSJLiv2X8tL	0.410	2	-17.458	1	
411	Calibre Rhossi	Pavilhão 9	spotify:track:158GdVwF8WM7jAmdYPcxI9	0.614	1	-7.327	0	
491	(You Drive Me) Crazy	Britney Spears	spotify:track:1DSJNBNhGZCigg9lI5VeZv	0.939	0	-4.288	0	
550	Souvenir Bottles	New Grass Revival	spotify:track:1nXO4SscDhGg8J5kknhiDK	0.446	2	-12.246	0	

164 rows × 24 columns

In [67]:

```
# Check shape

Test.shape
```

executed in 3ms, finished 17:07:55 2022-05-05

Out[67]: (164, 24)

In [69]:

```
# Create Train subset with 'songpng' and 'danceability'

traindf = Train[['songpng','danceability']]
traindf
```

executed in 9ms, finished 17:07:55 2022-05-05

Out[69]:

	songpng	danceability
684	Javier Solís - Más Allá.png	0.399
619	Wayne Shorter - Footprints - Remastered.png	0.530
904	Jack Jezzro - Rock Of Ages.png	0.275
855	Shaun Cassidy - Do You Believe In Magic.png	0.499
318	Eddie Fisher - People Like You.png	0.490
...	...	...
854	Roxette - Dangerous.png	0.712
72	Strawberry Alarm Clock - Barefoot In Baltimore...	0.682
517	The Calling - Wherever You Will Go.png	0.558
109	MPB4 - Milagre Brasileiro - Ao Vivo.png	0.368
771	Kim Wilde - Say You Really Want Me.png	0.699

489 rows × 2 columns

In [70]:
```
# Create Test subset with 'songpng' and 'danceability'

testdf = Test[['songpng','danceability']]
testdf
```

executed in 9ms, finished 17:07:55 2022-05-05

Out[70]:

	songpng	danceability
836	Sonny Boy Nelson - Blues Jumped a Rabbit.png	0.656
462	Lionel Richie - Love Will Conquer All.png	0.790
169	Sleeping At Last - Eight.png	0.341
578	Atlantic Starr - Circles.png	0.779
67	Kansas - Power.png	0.477
...	...	...
181	The Kooks - Ooh La.png	0.544
584	Ravi Shankar - Raga Bhimpalasi - Live.png	0.360
411	Pavilhão 9 - Calibre Rhossi.png	0.704
491	Britney Spears - (You Drive Me) Crazy.png	0.748
550	New Grass Revival - Souvenir Bottles.png	0.569

4.6 Keras flow_from_dataframe (ImageDataGenerator)

In [71]:
```
# Create train_datagen
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        validation_split=0.2)

# Create test_datagen
test_datagen = ImageDataGenerator(rescale=1./255)

# Set target_size (proportional to actual image size)
target_size = (380,245)

# Create train_generator
train_generator=train_datagen.flow_from_dataframe(
        dataframe=traindf,
        directory="Images/Train/",
        x_col="songpng",
        y_col="danceability",
        batch_size=38,
        seed=11,
        shuffle=True,
        class_mode='other',
        target_size=target_size)

# Create test_generator
test_generator=test_datagen.flow_from_dataframe(
        dataframe=testdf,
        directory="Images/Test/",
        x_col="songpng",
        y_col="danceability",
        batch_size=38,
        seed=11,
        shuffle=False,
        class_mode='other',
        target_size=target_size)

# Create validation_generator
validation_generator = train_datagen.flow_from_dataframe(
        dataframe=traindf,
        directory="Images/Train/",
        x_col="songpng",
        y_col="danceability",
        batch_size=38,
        seed=11,
        shuffle=True,
        class_mode='other',
        target_size=target_size,
        subset='validation')
```

executed in 17ms, finished 17:07:55 2022-05-05

```
Found 489 validated image filenames.
Found 163 validated image filenames.
Found 97 validated image filenames.
```

```
/Users/v/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/ker
as_preprocessing/image/dataframe_iterator.py:283: UserWarning: Found
1 invalid image filename(s) in x_col="songpng". These filename(s) wil
l be ignored.
  warnings.warn(
```

5 BUILD MODELS

5.1 Model 1: Layers

- Input layer
- Output layer

In [72]:
```
# Start model construction
# Model 1

model = Sequential()
model
```
executed in 22ms, finished 17:07:55 2022-05-05

Out[72]:
```
<tensorflow.python.keras.engine.sequential.Sequential at 0x7f78be613e50>
```

In [73]:
```
# Add input and output layers

model.add(Conv2D(32, (3, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1))
```
executed in 8ms, finished 17:07:55 2022-05-05

In [74]:
```
# Compile model with optimizer and loss function being 'mean_squared_error'

model.compile(loss='mean_squared_error', optimizer='adam')
```
executed in 7ms, finished 17:07:55 2022-05-05

In [75]:
```
# Fit model

history = model.fit(
    train_generator,
    validation_data=validation_generator,
    batch_size = 38, epochs = 20,
    callbacks=[EarlyStopping(patience=10, restore_best_weights=True, verbos
)
```

executed in 20m 20s, finished 17:28:15 2022-05-05

```
Epoch 1/20
13/13 [==============================] - 59s 5s/step - loss: 4984.924
3 - val_loss: 0.3203
Epoch 2/20
13/13 [==============================] - 55s 4s/step - loss: 0.3202 -
val_loss: 0.3215
Epoch 3/20
13/13 [==============================] - 55s 4s/step - loss: 0.3209 -
val_loss: 0.3217
Epoch 4/20
13/13 [==============================] - 55s 4s/step - loss: 0.3209 -
val_loss: 0.3216
Epoch 5/20
13/13 [==============================] - 57s 4s/step - loss: 2.3208 -
val_loss: 0.3215
Epoch 6/20
13/13 [==============================] - 56s 4s/step - loss: 0.3207 -
val_loss: 0.3213
Epoch 7/20
13/13 [==============================] - 59s 5s/step - loss: 0.3205 -
val_loss: 0.3211
Epoch 8/20
13/13 [==============================] - 57s 4s/step - loss: 0.3202 -
val_loss: 0.3208
Epoch 9/20
13/13 [==============================] - 57s 4s/step - loss: 0.3199 -
val_loss: 0.3204
Epoch 10/20
13/13 [==============================] - 56s 4s/step - loss: 0.3195 -
val_loss: 0.3201
Epoch 11/20
13/13 [==============================] - 56s 4s/step - loss: 0.3192 -
val_loss: 0.3197
Epoch 12/20
13/13 [==============================] - 58s 4s/step - loss: 0.3188 -
val_loss: 0.3193
Epoch 13/20
13/13 [==============================] - 57s 4s/step - loss: 0.3184 -
val_loss: 0.3189
Epoch 14/20
13/13 [==============================] - 57s 4s/step - loss: 0.3180 -
val_loss: 0.3185
Epoch 15/20
13/13 [==============================] - 55s 4s/step - loss: 0.3176 -
val_loss: 0.3181
Epoch 16/20
```

```
13/13 [==============================] - 56s 4s/step - loss: 0.3172 -
val_loss: 0.3176
Epoch 17/20
13/13 [==============================] - 56s 4s/step - loss: 0.3167 -
val_loss: 0.3172
Epoch 18/20
13/13 [==============================] - 57s 4s/step - loss: 0.3162 -
val_loss: 0.3167
Epoch 19/20
13/13 [==============================] - 58s 4s/step - loss: 0.3158 -
val_loss: 0.3162
Epoch 20/20
13/13 [==============================] - 57s 4s/step - loss: 0.3153 -
val_loss: 0.3157
```

In [76]:

```
# Show model summary

model.summary()
```

executed in 4ms, finished 17:28:15 2022-05-05

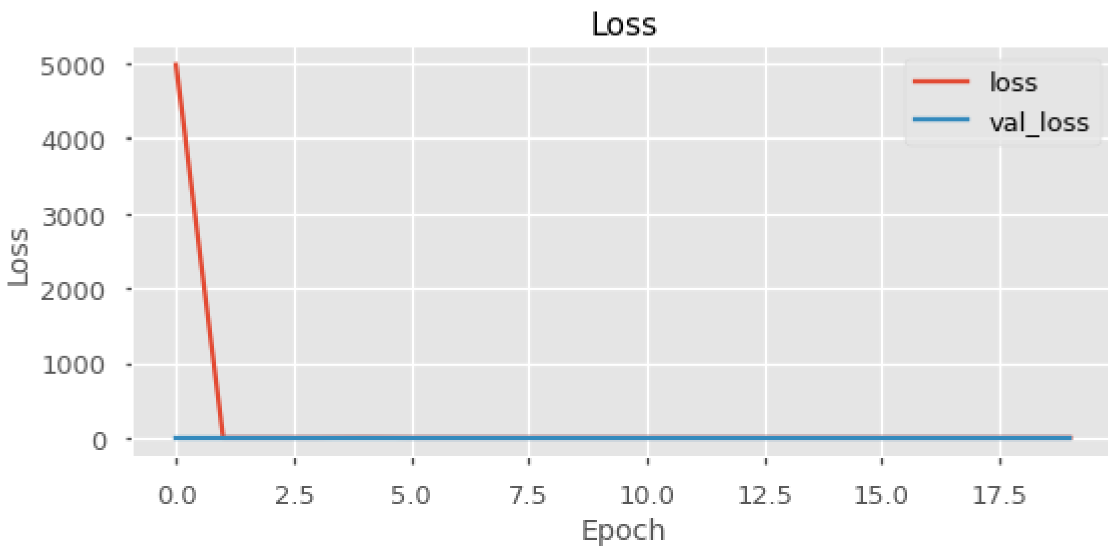
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, None, None, 32)    896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, None, None, 32)    0
_________________________________________________________________
flatten (Flatten)            (None, None)              0
_________________________________________________________________
dense (Dense)                (None, 32)                23417888
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 23,418,817
Trainable params: 23,418,817
Non-trainable params: 0
_________________________________________________________________
```

In [77]:
```
# Function to plot model performance

def plot_history(history, style=['ggplot', 'seaborn-talk']):
    """
    Plot history from History object (or history dict)
    once Tensorflow model is trained.

    Parameters:
    -----------
    history:
        History object returned from a model.fit()
    style: string or list of strings (default: ['ggplot', 'seaborn-talk'])
        Style from matplotlib.
    """

    # Pass in a model history object or a dictionary.
    if not isinstance(history, dict): # We prefer this type of check over `
        history = history.history

    metrics_lst = [m for m in history.keys() if not m.startswith('val')]
    N = len(metrics_lst)
    with plt.style.context(style):
        fig, ax_lst = plt.subplots(nrows=N, figsize=(8, 4*(N)))
        ax_lst = [ax_lst] if N == 1 else ax_lst.flatten() # Flatten ax_lst.
        for metric, ax in zip(metrics_lst, ax_lst):
            val_m = f'val_{metric}'
            ax.plot(history[metric], label=metric)
            ax.plot(history[val_m], label=val_m)
            ax.set(title=metric.title(), xlabel='Epoch', ylabel=metric.titl
            ax.legend()
        fig.tight_layout()
        plt.show()
```

executed in 5ms, finished 17:28:15 2022-05-05

In [78]:
```
# Plot model perfromance

plot_history(history)
```

executed in 171ms, finished 17:28:15 2022-05-05



In [79]:
```
history.history
```
executed in 3ms, finished 17:28:15 2022-05-05

Out[79]:
```
{'loss': [4984.92431640625,
  0.3202227056026459,
  0.3208640217781067,
  0.32090890407562256,
  2.3207967281341553,
  0.32068321108818054,
  0.3204504251480102 5,
  0.3201730847358703 6,
  0.319861501455307,
  0.31952926516532 9,
  0.3191784918308258,
  0.3188054263591766 4,
  0.318421334028244,
  0.3180115818977356,
  0.3175937831401825,
  0.3171612918376922 6,
  0.316709041595459,
  0.3162390589714050 3,
  0.3157650828361511,
  0.3152695000171661 4],
 'val_loss': [0.3202879726886749,
  0.3214782476425171,
  0.3216944336891174 3,
  0.321601212024688 7,
  0.3215090036392212,
  0.3213225901126861 6,
  0.3210610747337341 3,
  0.320758670568466 2,
  0.3204323351383209,
  0.3200866580009460 4,
  0.3197213709354400 6,
  0.319342672824859 6,
  0.3189405202865600 6,
  0.318526953458786,
  0.3180947303771972 7,
  0.3176445066928863 5,
  0.317181825637817 4,
  0.316706150770187 4,
  0.316213935613632 2,
  0.315710097551345 8]}
```

In [80]:
```
# Predict

# test_generator.reset()
# predictions = model.predict(test_generator)
# predictions
```
executed in 14ms, finished 17:28:15 2022-05-05

5.1.0.1 Result: Model 1 performed well with loss: 0.3153 & val_loss: 0.3157 [...]

5.2 Model 2: Stochastic Batching

In [81]:
```
# Model 2: Stochastic Batching

model = Sequential()
model
```
executed in 5ms, finished 17:28:15 2022-05-05

Out[81]:
```
<tensorflow.python.keras.engine.sequential.Sequential at 0x7f78a85794f0>
```

In [82]:
```
# Same input and output layers as first model

model.add(Conv2D(32, (3, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1))
```
executed in 7ms, finished 17:28:15 2022-05-05

In [83]:
```
# Compile model

model.compile(loss='mean_squared_error', optimizer='adam')
```
executed in 8ms, finished 17:28:15 2022-05-05

In [84]:
```
# Fit model
# Stochastic Batching - set batch_size = 1

history = model.fit(
    train_generator,
    validation_data=validation_generator,
    batch_size = 1, epochs = 20,
    callbacks=[EarlyStopping(patience=10, restore_best_weights=True, verbos
)
```

executed in 20m 35s, finished 17:48:50 2022-05-05

```
Epoch 1/20
13/13 [==============================] - 60s 5s/step - loss: 3747.3982 -
val_loss: 0.3204
Epoch 2/20
13/13 [==============================] - 57s 4s/step - loss: 0.3204 - val
_loss: 34.6313
Epoch 3/20
13/13 [==============================] - 59s 5s/step - loss: 3.1542 - val
_loss: 0.3221
Epoch 4/20
13/13 [==============================] - 57s 4s/step - loss: 0.3213 - val
_loss: 0.3220
Epoch 5/20
13/13 [==============================] - 58s 4s/step - loss: 0.3212 - val
_loss: 0.3218
Epoch 6/20
13/13 [==============================] - 58s 4s/step - loss: 0.3209 - val
_loss: 0.3215
Epoch 7/20
13/13 [==============================] - 56s 4s/step - loss: 0.3206 - val
_loss: 0.3212
Epoch 8/20
13/13 [==============================] - 57s 4s/step - loss: 0.3203 - val
_loss: 0.3208
Epoch 9/20
13/13 [==============================] - 56s 4s/step - loss: 0.3199 - val
_loss: 0.3204
Epoch 10/20
13/13 [==============================] - 57s 4s/step - loss: 0.3195 - val
_loss: 0.3201
Epoch 11/20
13/13 [==============================] - 57s 4s/step - loss: 0.3191 - val
_loss: 0.3196
Epoch 12/20
13/13 [==============================] - 57s 4s/step - loss: 0.3187 - val
_loss: 0.3192
Epoch 13/20
13/13 [==============================] - 58s 4s/step - loss: 0.3182 - val
_loss: 0.3187
Epoch 14/20
13/13 [==============================] - 57s 4s/step - loss: 0.3178 - val
_loss: 0.3182
Epoch 15/20
13/13 [==============================] - 57s 4s/step - loss: 0.3173 - val
_loss: 0.3177
```

```
Epoch 16/20
13/13 [==============================] - 57s 4s/step - loss: 0.3168 - val
_loss: 0.3172
Epoch 17/20
13/13 [==============================] - 58s 4s/step - loss: 0.3163 - val
_loss: 0.3167
Epoch 18/20
13/13 [==============================] - 57s 4s/step - loss: 0.3157 - val
_loss: 0.3162
Epoch 19/20
13/13 [==============================] - 56s 4s/step - loss: 0.3152 - val
_loss: 0.3156
Epoch 20/20
13/13 [==============================] - 58s 4s/step - loss: 0.3146 - val
_loss: 0.3150
```

In [85]:

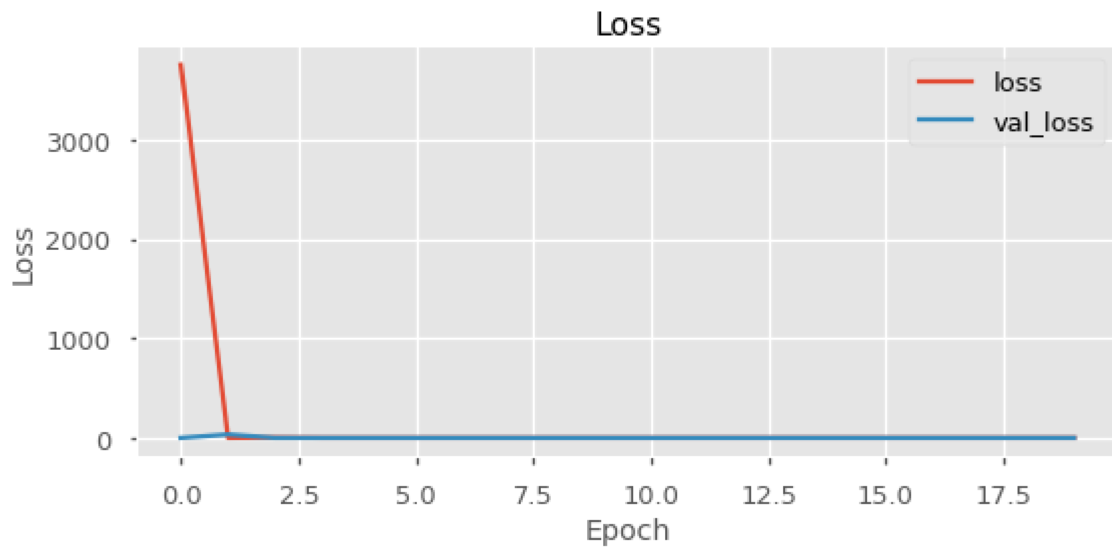
```
# Show model summary

model.summary()
```

executed in 4ms, finished 17:48:50 2022-05-05

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, None, None, 32)    896       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, None, None, 32)    0         
_________________________________________________________________
flatten_1 (Flatten)          (None, None)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 32)                23417888  
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33        
=================================================================
Total params: 23,418,817
Trainable params: 23,418,817
Non-trainable params: 0
_________________________________________________________________
```

In [86]:
```
# Plot model performance

plot_history(history)
```

executed in 157ms, finished 17:48:50 2022-05-05



Result: Model 2 also performed well with loss: 0.3146 & val_loss: 0.3150 (similar to Model 1)

5.2.0.1 Result: Model 2 also performed well with loss: 0.3146 & val_loss: 0.3150 (similar to Model 1)

5.3 Model 3: Add Layers to Model 1

In [87]:
```
# Model 3: Add layers to Model 1

model = Sequential()
model
```
executed in 6ms, finished 17:48:50 2022-05-05

Out[87]:
```
<tensorflow.python.keras.engine.sequential.Sequential at 0x7f78990e89a0>
```

In [88]:
```
# Add layers

model.add(Conv2D(32, (3, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1))
```
executed in 18ms, finished 17:48:50 2022-05-05

In [89]:
```
# Compile model

model.compile(loss='mean_squared_error', optimizer='adam')
```
executed in 9ms, finished 17:48:50 2022-05-05

In [90]:
```
# Fit model

history = model.fit(
    train_generator,
    validation_data=validation_generator,
    batch_size = 38, epochs = 20,
    callbacks=[EarlyStopping(patience=10, restore_best_weights=True, verbos
)
```

executed in 21m 55s, finished 18:10:45 2022-05-05

```
Epoch 1/20
13/13 [==============================] - 66s 5s/step - loss: 246.2116 - v
al_loss: 26.9277
Epoch 2/20
13/13 [==============================] - 61s 5s/step - loss: 8.5070 - val
_loss: 0.5065
Epoch 3/20
13/13 [==============================] - 59s 5s/step - loss: 1.5512 - val
_loss: 1.2820
Epoch 4/20
13/13 [==============================] - 59s 5s/step - loss: 0.5023 - val
_loss: 0.2157
Epoch 5/20
13/13 [==============================] - 59s 5s/step - loss: 0.1194 - val
_loss: 0.0276
Epoch 6/20
13/13 [==============================] - 63s 5s/step - loss: 0.0657 - val
_loss: 0.0578
Epoch 7/20
13/13 [==============================] - 65s 5s/step - loss: 0.0366 - val
_loss: 0.0253
Epoch 8/20
13/13 [==============================] - 64s 5s/step - loss: 0.0246 - val
_loss: 0.0241
Epoch 9/20
13/13 [==============================] - 65s 5s/step - loss: 0.0241 - val
_loss: 0.0248
Epoch 10/20
13/13 [==============================] - 65s 5s/step - loss: 0.0254 - val
_loss: 0.0257
Epoch 11/20
13/13 [==============================] - 61s 5s/step - loss: 0.0225 - val
_loss: 0.0261
Epoch 12/20
13/13 [==============================] - 60s 5s/step - loss: 0.0224 - val
_loss: 0.0234
Epoch 13/20
13/13 [==============================] - 60s 5s/step - loss: 0.0218 - val
_loss: 0.0221
Epoch 14/20
13/13 [==============================] - 59s 5s/step - loss: 0.0213 - val
_loss: 0.0245
Epoch 15/20
13/13 [==============================] - 59s 5s/step - loss: 0.0214 - val
_loss: 0.0219
Epoch 16/20
```

```
13/13 [==============================] - 58s 4s/step - loss: 0.0203 - val
_loss: 0.0374
Epoch 17/20
13/13 [==============================] - 58s 4s/step - loss: 0.0232 - val
_loss: 0.0222
Epoch 18/20
13/13 [==============================] - 59s 5s/step - loss: 0.0213 - val
_loss: 0.0201
Epoch 19/20
13/13 [==============================] - 58s 4s/step - loss: 0.0204 - val
_loss: 0.0200
Epoch 20/20
13/13 [==============================] - 59s 5s/step - loss: 0.0191 - val
_loss: 0.0183
```

In [91]:

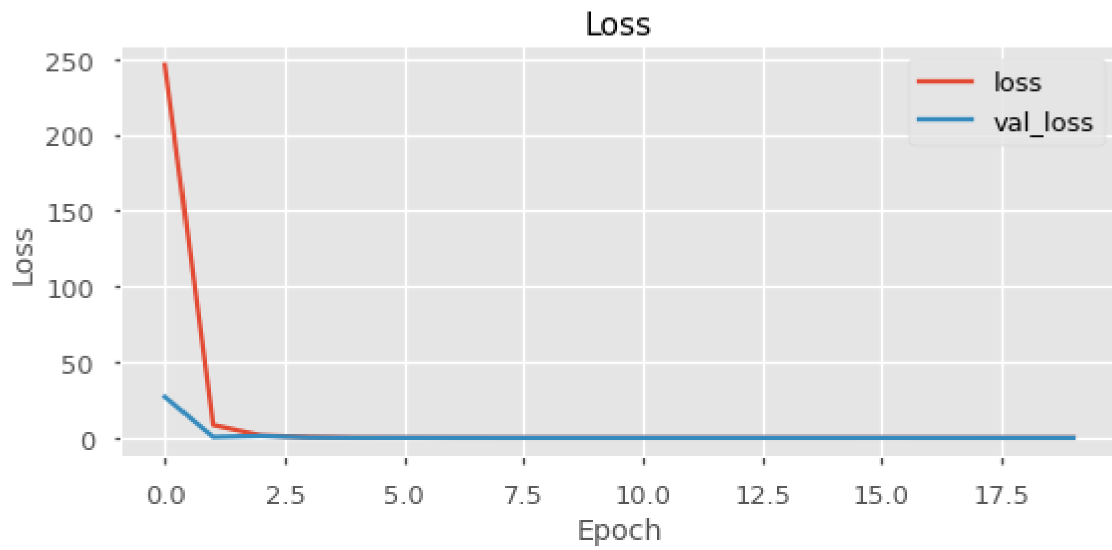
```
# Show model summary

model.summary()
```

executed in 5ms, finished 18:10:45 2022-05-05

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, None, None, 32)    896
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, None, None, 32)    0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, None, None, 64)    18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, None, None, 64)    0
_________________________________________________________________
flatten_2 (Flatten)          (None, None)              0
_________________________________________________________________
dense_4 (Dense)              (None, 32)                11237408
_________________________________________________________________
dense_5 (Dense)              (None, 16)                528
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 17
=================================================================
Total params: 11,257,345
Trainable params: 11,257,345
Non-trainable params: 0
_________________________________________________________________
```

In [92]:
```
# Plot model performance

plot_history(history)
```

executed in 163ms, finished 18:10:45 2022-05-05



5.3.1 Check Model Against Unseen (Test) Data

Show test metrics for validation and evaluate

In [120]:
```
# Predictions on Test set

test_generator.reset()
predictions = model.predict(test_generator)
predictions
```

executed in 17.5s, finished 05:58:25 2022-05-06

Out[120]:
```
array([[0.6918415 ],
       [1.0344843 ],
       [0.84120595],
       [1.087685  ],
       [0.8141254 ],
       [0.56475675],
       [0.8014492 ],
       [0.49327695],
       [0.65290296],
       [0.5474533 ],
       [0.6528648 ],
       [0.70644987],
       [0.6197113 ],
       [0.7553543 ],
       [0.9941238 ],
       [0.61422575],
       [0.77679574],
       [0.7148117 ],
       [0.9610828 ],
```

In [93]:
```
# Evaluate model on Test set

model.evaluate(test_generator)
```

executed in 17.6s, finished 18:11:03 2022-05-05

```
5/5 [==============================] - 11s 2s/step - loss: 0.0755
```

Out[93]:
```
0.07550748437643051
```

Result: Model 3 performed the best with loss: 0.0191 & val_loss: 0.0183

5.3.1.1 Result: Model 3 performed the best with loss: 0.0191 & val_loss: 0.0183

Evaluation and Conclusions

```
* All three Sequential Models performed well, and we feel most confident
with Model 3
* With Model 3's MSE (mean squared error) = loss: 0.0191 & val_loss:
0.0183, our model shows it will be a strong predictor of "danceability"
of songs
* We will use the same approach in our Future Work with other metrics in
the dataset
```

6 Evaluation and Conclusions

6 Evaluation and Conclusions

- All three Sequential Models performed well, and we feel most confident with Model 3
- With Model 3's MSE (mean squared error) = loss: 0.0191 & val_loss: 0.0183, our model shows it will be a strong predictor of "danceability" of songs
- We will use the same approach in our Future Work with other metrics in the dataset

```
# FUTURE WORK

* Run models for all remaining metrics to add dimensionality to final
output
* Remaining metrics:
1. Energy
2. Speechiness
3. Acousticness
4. Instrumentalness
5. Liveness
6. Valence
* Build platform to connect users listening to the same or similar song
and apply Disco Duo
```

7 FUTURE WORK

- Run models for all remaining metrics to add dimensionality to final output
- Remaining metrics:

1. Energy
2. Speechiness
3. Acousticness
4. Instrumentalness
5. Liveness
6. Valence

- Build platform to connect users listening to the same or similar song and apply Disco Duo

7.1 Appendix

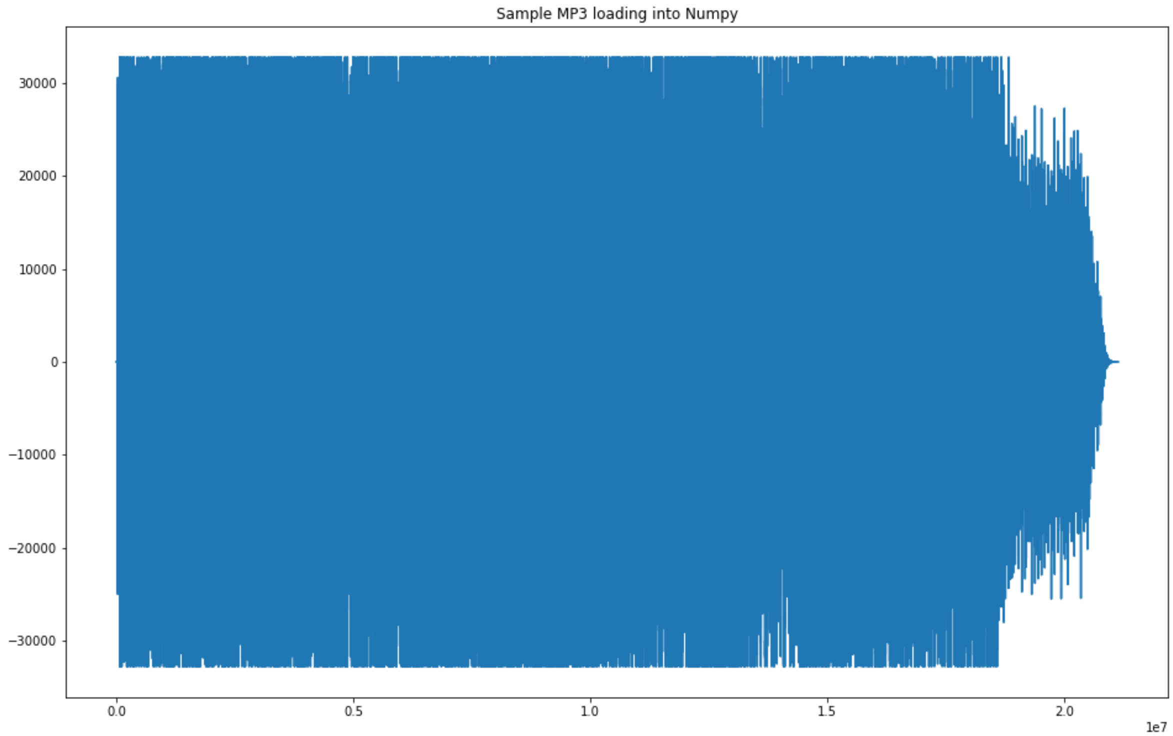
```
## Appendix - Part I
For Future Work
<br>AudioSegment from pydub
```

7.2 Appendix - Part I

For Future Work
AudioSegment from pydub

In [94]:
```
import pydub
import numpy as np

def read(f, normalized=False):
    """MP3 to numpy array"""
    a = pydub.AudioSegment.from_mp3(f)
    y = np.array(a.get_array_of_samples())
    if a.channels == 2:
        y = y.reshape((-1, 2))
    if normalized:
        return a.frame_rate, np.float32(y) / 2**15
    else:
        return a.frame_rate, y

def write(f, sr, x, normalized=False):
    """numpy array to MP3"""
    channels = 2 if (x.ndim == 2 and x.shape[1] == 2) else 1
    if normalized:  # normalized array - each item should be a float in [-1
        y = np.int16(x * 2 ** 15)
    else:
        y = np.int16(x)
    song = pydub.AudioSegment(y.tobytes(), frame_rate=sr, sample_width=2, c
    song.export(f, format="mp3", bitrate="320k")

audio_file = 'Song_Data/Sade - No Ordinary Love.mp3'
sr, x = read(audio_file)

import matplotlib.pyplot as plt
plt.figure(figsize=(16,10))
plt.plot(x, color='tab:blue')
plt.title("Sample MP3 loading into Numpy")
plt.show()
```

executed in 5.87s, finished 18:11:09 2022-05-05



In [95]:
```
sade_song = AudioSegment.from_mp3("Song_Data/Sade - No Ordinary Love.mp3")

sade_song[:100_000]
```
executed in 2.58s, finished 18:11:11 2022-05-05

Out[95]:

1:40 / 1:40

In [96]:
```
kansas_song = AudioSegment.from_mp3("Song_Data/Kansas - Power.mp3")

kansas_song[:100_000]
```
executed in 2.21s, finished 18:11:14 2022-05-05

Out[96]:

0:00 / 1:40

In [97]:
```
kiiara_song = AudioSegment.from_mp3("Song_Data/Kiiara - Gold.mp3")

kiiara_song[:100_000]
```
executed in 2.15s, finished 18:11:16 2022-05-05

Out[97]:

0:00 / 1:40

In [98]:
```
type(sade_song)
```
executed in 3ms, finished 18:11:16 2022-05-05

Out[98]:
```
pydub.audio_segment.AudioSegment
```

In [99]:
```
np.array(sade_song.get_array_of_samples()).reshape((-1,2))
```
executed in 93ms, finished 18:11:16 2022-05-05

Out[99]:
```
array([[0, 0],
       [0, 0],
       [0, 0],
       ...,
       [0, 0],
       [0, 0],
       [0, 0]], dtype=int16)
```

In [100]:
```
sr, x = read('Song_Data/Sade - No Ordinary Love.mp3')
x.shape
```
executed in 937ms, finished 18:11:17 2022-05-05

Out[100]:
```
(21142400, 2)
```

In [101]:
```
x
```
executed in 3ms, finished 18:11:17 2022-05-05

Out[101]:
```
array([[0, 0],
       [0, 0],
       [0, 0],
       ...,
       [0, 0],
       [0, 0],
       [0, 0]], dtype=int16)
```

```
## Appendix - Part II
For Future Work <br>
Reference code for Librosa
```

7.3 Appendix - Part II

For Future Work
Reference code for Librosa

In [102]:
```
# LIBROSA

# 1. Get the file path to an included audio example
# filename = librosa.example('nutcracker')
audio_file = 'Song_Data/Sade - No Ordinary Love.mp3'

# 2. Load the audio as a waveform `y`
#    Store the sampling rate as `sr`
# signal, sr = librosa.load(filename)

signal, sr = librosa.load(audio_file)
```

executed in 14.3s, finished 18:11:31 2022-05-05

```
/Users/v/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/libros
a/util/decorators.py:88: UserWarning: PySoundFile failed. Trying audiorea
d instead.
  return f(*args, **kwargs)
```

In [103]:
```
# audio_file = 'Song_Data/Sade - No Ordinary Love.mp3'
# sr, x = read(audio_file)
```

executed in 2ms, finished 18:11:31 2022-05-05

In [104]:
```
# signal, sr = librosa.load(fpath, sr=SAMPLE_RATE)
```

executed in 2ms, finished 18:11:31 2022-05-05