DJ with Data - Spotify Music Analysis in Python

July 9, 2023

1 Introduction

There is something so powerful about music. It can makes us laugh or it can make us cry. It can fill us with energy or it can relax us to sleep. It can remind us of the past or inspire us for the future. There are limitless ways to experience music, and that's what makes it so special to me. I've always had a huge passion for music (my 80,000+ listening minutes last year on Spotify can attest to that), but it's not always easy to know if others will share the same liked songs. As the guy who loves creating new playlists (and has a massive boombox I bring everywhere I go), it's usually my job to pick the tunes at gatherings. Music is truly subjective when considering what's "good" or "bad", but I often rely on intuition for picking the right songs. While I *think* I do a decent job, I've always asked myself if there's a better way. What if I could use data to find the right songs? Now I could always rely on the experts to tell me what's popular, but what's the fun in that! Let's explore my music to see what insights we can find.

1.1 Objective:

Identify trends in songs from a preferred music playlist to help identify songs that match the same trends.

1.2 Guiding Questions:

- Is a high popularity important?
- What trends do we see as songs get older in our dataset?
- What trends do we see in technical song characteristics?
- Do the identified song trends help us develop criteria for future songs?

2 Data Overview

2.1 About the Data

For my analysis, I will be using one of my favorite playlists I've created on Spotify called "Suns Out Guns Out" that I tried to make as a catch-all summer playlist for popular songs over the years. The playlist has 730 songs, a total listening time of 42.33 hours, and houses a variety of different genres.

2.2 Tools for the Job

Tools needed include:

- Spotify API
- Python
- Python Libraries:
  - Numpy
  - Pandas
  - Matplotlib
  - Seaborn
  - Scipy
  - Spotipy

In order to access the data, I will need to use Spotify's provided API coupled with `spotipy` for Python. This will allow me to pull all the necessary data from any playlist under my account. To conduct the analysis, I will utilize `pandas` for my dataframe, `matplotlib` and `seaborn` for visualizations, and `scipy` for statistical analysis. I will start by importing all my libraries and initiating a new token for access through the Spotify API.

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from seaborn_qqplot import pplot
import matplotlib.pyplot as plt
import scipy.stats as sci
import os
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
from spotipy.oauth2 import SpotifyOAuth

#left align all markdown tables for consistency
from IPython.core.display import HTML
table_css = 'table {align:left;display:block} '
HTML('<style>{}</style>'.format(table_css))
```

[1]:
```
<IPython.core.display.HTML object>
```

[3]:
```
#This sets the environmental variables during this session. Token strings are␣
↪hidden for confidentiality.
os.environ['SPOTIPY_CLIENT_ID'] = client_ID
os.environ['SPOTIPY_CLIENT_SECRET'] = client_secret
os.environ['SPOTIPY_REDIRECT_URI'] = redirect_uri

sp = spotipy.Spotify(auth_manager=SpotifyOAuth(scope='user-library-read'))
```

3 Data Extraction, Cleaning, and Transformation

Now that I have access to my data, I need to pull my data from Spotify into a dataframe before I can start cleaning or transforming. I will then display my dataframe to start learning about my data.

[4]:
```
remaining_songs = sp.playlist_tracks('https://open.spotify.com/playlist/
↪4I8DT9gh83aBpiJRSEZ9Jb?si=d5aa22b7400b49d9')['total'] #Determine the total␣
↪number of songs
offset=0
limit=100
saved_tracks = []

while True:

    #pulls a list of songs from the playlist based on the starting position␣
↪(offset) and number of songs to pull (limit)
    tracks = sp.playlist_tracks('https://open.spotify.com/playlist/
↪4I8DT9gh83aBpiJRSEZ9Jb?
↪si=d5aa22b7400b49d9',offset=offset,limit=limit)['items']

    for track in tracks:

        #pulls the id, song name, album release date, first artist, first␣
↪artist ID, and popularity and saves them in the saved_tracks list
        saved_tracks.append((track['track']['id'],
                             track['track']['name'],
                             track['track']['album']['release_date'],
                             track['track']['artists'][0]['name'],
                             track['track']['artists'][0]['id'],
                             track['track']['popularity']))

    remaining_songs-=100

    #if we've reached the end of the playlist, exit the loop. Otherwise, change␣
↪the starting position in the playlist and continue.
    if remaining_songs <=0:
        break
    else:
        offset+=limit

df = pd.
↪DataFrame(data=saved_tracks,columns=['track_ID','name','release_date','artist','artist_ID',
df
```

[4]:
```
                 track_ID  \
0    4QNpBfC0zvjKqPJcyqBy9W
```

```
1    2bJvI42r8EF3wxjOuDav4r
2    4kWO6O1BUXcZmaxitpVUwp
3    07nH4ifBxUB4lZcsf44Brn
4    24LS4lQShWyixJ0ZrJXfJ5
..                      …
725  0JbSghVDghtFEurrSO8JrC
726  57DJaoHdeeRrg7MWthNnee
727  0zKbDrEXKpnExhGQRe9dxt
728  3g5OlVimHO0rK6qmRiwokX
729  0qXGuBm0tmBLjC7InLM3EK

                                                 name release_date  \
0    Give Me Everything (feat. Ne-Yo, Afrojack & Na…   2011-06-17
1                                Time of Our Lives   2014-11-21
2                                      Jackie Chan   2018-05-18
3                        Blame (feat. John Newman)   2014-10-31
4                Sweet Nothing (feat. Florence Welch)   2012-10-29
..                                               …          …
725                  Country Girl (Shake It For Me)   2011-01-01
726                   Body Back (feat. Maia Wright)   2019-10-24
727                                         Lay Low   2023-01-06
728                                      Glad U Came   2023-04-27
729                                   Don't Say Love   2023-06-16

            artist               artist_ID  popularity
0          Pitbull  0TnOYISbd1XYRBk9myaseg          86
1          Pitbull  0TnOYISbd1XYRBk9myaseg          85
2           Tiësto  2o5jDhtHVPhrJdv3cEQ99Z          75
3    Calvin Harris  7CajNmpbOovFoOoasH2HaY          81
4    Calvin Harris  7CajNmpbOovFoOoasH2HaY          78
..             …                     …         …
725     Luke Bryan  0BvkDsjIUla7X0k6CSWh1I          82
726        Gryffin  2ZRQcIgzPCVaT9XKhXZIzh          62
727         Tiësto  2o5jDhtHVPhrJdv3cEQ99Z          88
728   Jason Derulo  07YZf4WDAMNwqr4jfgOZ8y          73
729     Leigh-Anne  79QUtAVxGAAoiWNlqBz9iy          76

[730 rows x 6 columns]
```

We now have a dataframe filled with songs from my playlist! Let's use this opportunity to make some quick observations about my data. I pulled the full 730 songs from my playlist, and there are a number of columns that were generated for each song. I pulled the unique track ID, song title (name), release date, artist, unique artist ID, and popularity score.

What is a popularity score?

Spotify describes it as the following:

> "The popularity of a track is a value between 0 and 100, with 100 being the most popular. The popularity is calculated by algorithm and is based, in the most part, on the total number of plays the track has had and how recent those plays are" (Spotify, n.d. -a).

It's also important to highlight that every unique song has it's own calculated popularity score. The same song could have very different scores (e.g. explicit vs non-explicit).

Another observation worth noting is the current format of my release date column. It is currently in yyyy/mm/dd format, but for my analysis I'll want to see only the year. I'll create a function to extract just the year from the date, apply it across the entire column, and generate a new column for the year.

[5]:
```
def release_year(date):
    return int(str(date)[:4])
```

[6]:
```
df['release_year'] = df['release_date'].apply(release_year)
df.head()
```

[6]:
```
                 track_ID                                               name  \
0  4QNpBfC0zvjKqPJcyqBy9W  Give Me Everything (feat. Ne-Yo, Afrojack & Na…
1  2bJvI42r8EF3wxjOuDav4r                                  Time of Our Lives
2  4kWO6O1BUXcZmaxitpVUwp                                        Jackie Chan
3  07nH4ifBxUB4lZcsf44Brn                       Blame (feat. John Newman)
4  24LS4lQShWyixJ0ZrJXfJ5              Sweet Nothing (feat. Florence Welch)

  release_date         artist               artist_ID  popularity  \
0   2011-06-17        Pitbull  0TnOYISbd1XYRBk9myaseg          86
1   2014-11-21        Pitbull  0TnOYISbd1XYRBk9myaseg          85
2   2018-05-18         Tiësto  2o5jDhtHVPhrJdv3cEQ99Z          75
3   2014-10-31  Calvin Harris  7CajNmpbOovFoOoasH2HaY          81
4   2012-10-29  Calvin Harris  7CajNmpbOovFoOoasH2HaY          78

   release_year
0          2011
1          2014
2          2018
3          2014
4          2012
```

Nice! However, I need more data before I can continue with my analysis. Now that I have unique ID's for each individual track in the playlist, I can look up the associated unique track audio features per song. I will be pulling different audio features that are scored on a scale between 0 - 1. Where a score lies on the scale expresses information about the song. Table 1 below highlights the track features I'll be investigating along with the scale boundaries.

Table 1

Track Audio Feature Boundaries and Definitions

Audio Feature	Lower Boundary (0)	Upper Boundary (1)
Acousticness	low confidence acoustic	high confidence acoustic
Danceability	least danceable	most danceable
Energy	low energy	high energy (fast, loud, noisy)
Instrumentalness	not instrumental	instrumental (no vocals)
Liveness	not live	live (audience present)
speechiness	low vocal level (no vocals)	high vocal level (audio book)
Valence	negative sound (sad, depressed)	positive sound (happy, cheerful)

Note. This table illustrates upper and lower boundary examples for specific track audio features where a score of 0 represents the lower boundary and a score of 1 represents the upper boundary. Adapted from *Get Track's Audio Features*, by Spotify (n.d. -b). Copyright 2023 by Spotify AB.

Just like I pulled my data from the playlist, I'll look up each unique track ID and pull the track audio features into a new dataframe. Once I have two dataframes with the same unique track ID's (unique primary keys), I can utilize a merge operation to combine the dataframes. In this case, I will conduct an inner join to produce my new dataframe with all the data integrated together. In order to ensure I have a clean merge, I will first clean my primary keys on each dataframe to remove any potential duplicates. Once that is complete I can move forward with the merge. The very last thing to complete the cleaning process is checking for any empty values (set as "DEFAULT" in my script) before I move on to the analysis phase.

[7]:
```
remaining_songs = sp.playlist_tracks('https://open.spotify.com/playlist/
↪4I8DT9gh83aBpiJRSEZ9Jb?si=d5aa22b7400b49d9')['total'] #Determine the total␣
↪number of songs
track_ids = list(df['track_ID'])
track_subset = []
features_list = []
offset = 0
limit = 100

while True:

    #pulls a subset of tracks from the track list
    track_subset = track_ids[offset:offset+limit]

    #pulls track audio features for the subset of tracks as a Spotipy object
    audio_features_results = sp.audio_features(track_subset)

    #For each location and track in the Spotipy object, create a new tuple that␣
↪is appended to a features list.
    #If there is no data available for a track, populate the values with a␣
↪generic string "DEFAULT".
```

```
    for index,track in enumerate(audio_features_results):

        try:
            feature_tuple = (track['id'],
                             track['acousticness'],
                             track['danceability'],
                             track['energy'],
                             track['instrumentalness'],
                             track['liveness'],
                             track['speechiness'],
                             track['valence'])

            features_list.append(feature_tuple)

        except TypeError:
            feature_tuple = (track_subset[index],
                             "DEFAULT",
                             "DEFAULT",
                             "DEFAULT",
                             "DEFAULT",
                             "DEFAULT",
                             "DEFAULT",
                             "DEFAULT",)

            features_list.append(feature_tuple)


    remaining_songs-=limit

    #if we've reached the end of the playlist, exit the loop. Otherwise, change␣
 ↪the starting position in the playlist and continue.
    if remaining_songs <=0:
        break
    else:
        offset+=limit

features_df = pd.DataFrame(data=features_list,columns=['track_ID',
                                                       'acousticness',
                                                       'danceability',
                                                       'energy',
                                                       'instrumentalness',
                                                       'liveness',
                                                       'speechiness',
                                                       'valence'])
```

[8]:

```
#remove duplicates in both dataframes then generate a new dataframe by merging␣
 ↪both dataframes on 'track_ID'
```

```
df = df.drop_duplicates(subset=['track_ID'])
features_df = features_df.drop_duplicates(subset=['track_ID'])
playlist_df = pd.merge(df,features_df,how='inner',on='track_ID')

#loop through the new dataframe playlist_df and delete any rows containing␣
↪'DEFAULT'. If one column is empty, all columns are empty.
index = []
empty_tracks_df = playlist_df[playlist_df['acousticness'] == 'DEFAULT']

for val in empty_tracks_df.index:
    playlist_df = playlist_df.drop(val,axis=0)

playlist_df
```

[8]:
```
                   track_ID  \
0    4QNpBfC0zvjKqPJcyqBy9W
1    2bJvI42r8EF3wxjOuDav4r
2    4kW0601BUXcZmaxitpVUwp
3    07nH4ifBxUB4lZcsf44Brn
4    24LS4lQShWyixJ0ZrJXfJ5
..                      …
721  0JbSghVDghtFEurrS08JrC
722  57DJaoHdeeRrg7MWthNnee
723  0zKbDrEXKpnExhGQRe9dxt
724  3g5OlVimHO0rK6qmRiwokX
725  0qXGuBm0tmBLjC7InLM3EK

                                                  name release_date  \
0    Give Me Everything (feat. Ne-Yo, Afrojack & Na…   2011-06-17
1                                     Time of Our Lives   2014-11-21
2                                           Jackie Chan   2018-05-18
3                            Blame (feat. John Newman)   2014-10-31
4                   Sweet Nothing (feat. Florence Welch)   2012-10-29
..                                                   …          …
721                       Country Girl (Shake It For Me)   2011-01-01
722                        Body Back (feat. Maia Wright)   2019-10-24
723                                              Lay Low   2023-01-06
724                                           Glad U Came   2023-04-27
725                                        Don't Say Love   2023-06-16

            artist               artist_ID  popularity  release_year  \
0          Pitbull  0TnOYISbd1XYRBk9myaseg          86          2011
1          Pitbull  0TnOYISbd1XYRBk9myaseg          85          2014
2           Tiësto  2o5jDhtHVPhrJdv3cEQ99Z          75          2018
3    Calvin Harris  7CajNmpbOovFoOoasH2HaY          81          2014
4    Calvin Harris  7CajNmpbOovFoOoasH2HaY          78          2012
..             …                       …         …           …
```

```
721     Luke Bryan  0BvkDsjIUla7X0k6CSWh1I         82         2011
722        Gryffin  2ZRQcIgzPCVaT9XKhXZIzh         62         2019
723         Tiësto  2o5jDhtHVPhrJdv3cEQ99Z         88         2023
724   Jason Derulo  07YZf4WDAMNwqr4jfgOZ8y         73         2023
725     Leigh-Anne  79QUtAVxGAAoiWNlqBz9iy         76         2023

     acousticness  danceability  energy  instrumentalness  liveness  \
0         0.19100         0.671   0.939          0.000000    0.2980
1         0.09210         0.721   0.802          0.000000    0.6940
2         0.37400         0.747   0.834          0.000000    0.0586
3         0.02870         0.414   0.857          0.005740    0.3430
4         0.19700         0.573   0.929          0.000112    0.0567
..            …             …      …               …         …
721       0.02930         0.645   0.904          0.000000    0.0834
722       0.09310         0.687   0.832          0.000001    0.1830
723       0.06070         0.534   0.855          0.000263    0.3460
724       0.00627         0.675   0.801          0.000000    0.3710
725       0.01170         0.703   0.883          0.000000    0.2550

     speechiness  valence
0         0.1610    0.530
1         0.0583    0.724
2         0.0450    0.687
3         0.0808    0.348
4         0.1090    0.582
..           …        …
721       0.0462    0.671
722       0.0366    0.448
723       0.1830    0.420
724       0.0579    0.521
725       0.1840    0.812

[726 rows x 14 columns]
```

We now have a complete dataframe with all the data I need from my playlist. The next step is to conduct my analysis.

4 Data Analysis

For my data analysis, I'll start with simply getting to know my data. I'll ulitize python's `describe` function to get statistical information regarding my dataset.

[9]:
```
playlist_df.describe()
```

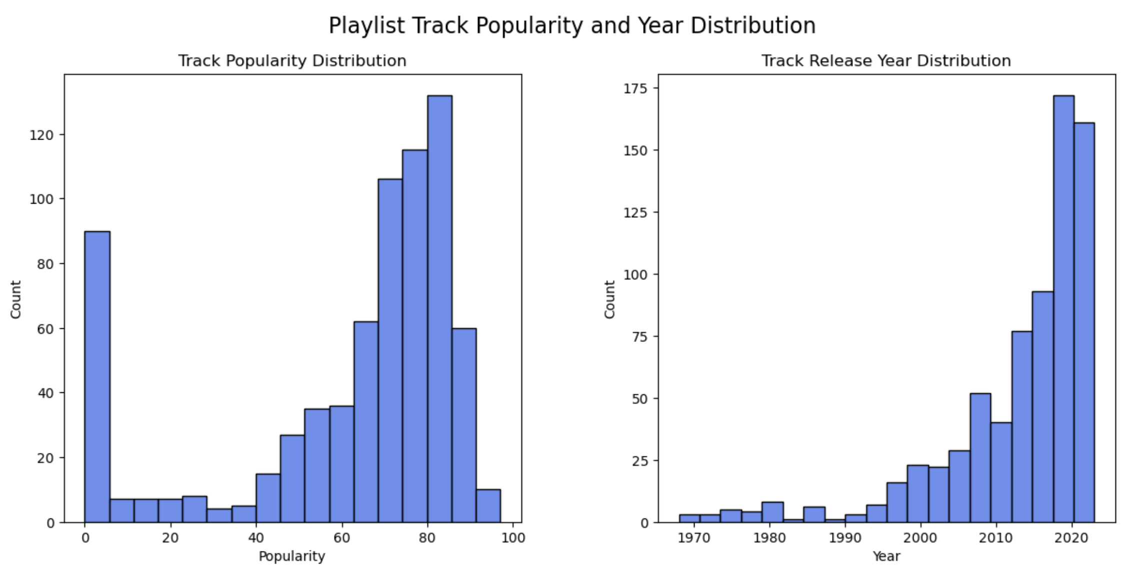
[9]:
```
       popularity  release_year  acousticness  danceability      energy  \
count  726.000000    726.000000    726.000000    726.000000  726.000000
mean    61.279614   2013.151515      0.108295      0.689017    0.731063
std     27.767762     10.242800      0.128209      0.122136    0.133644
min      0.000000   1968.000000      0.000018      0.276000    0.127000
25%     54.000000   2009.000000      0.017900      0.618000    0.647000
50%     72.000000   2017.000000      0.058250      0.696500    0.741500
75%     80.000000   2020.000000      0.162000      0.763750    0.833000
max     97.000000   2023.000000      0.699000      0.967000    0.978000

       instrumentalness    liveness  speechiness     valence
count        726.000000  726.000000   726.000000  726.000000
mean           0.008038    0.185606     0.087175    0.588983
std            0.056529    0.141560     0.077953    0.207926
min            0.000000    0.022600     0.025200    0.057900
25%            0.000000    0.090900     0.040700    0.446500
50%            0.000000    0.128000     0.057450    0.594000
75%            0.000033    0.255000     0.100750    0.754750
max            0.837000    0.826000     0.592000    0.979000
```

There's a lot to unpack with this data, but some initial observations can be made: - Popularity scores fluctate a lot. The mean score is around 61 out of 100, but standard deviation is quite high at around 27.8. - Songs generally have high danceability, energy, and valence based on their mean values (scale 0-1). - Songs generally have low instrumentalness, speechiness, acoustics, and liveness based on their mean values (scale 0-1). - Songs are fairly modern with the mean year being around 2013 and a standard deviation of around 10.

The release year data shows that the majority of the songs tend to be more modern, but songs go back as far as 1968! Popularity, on the other hand, is a lot more merky. 50% of the data is between 54 and 80 and a mean of 61 suggests that popularity is generally high, but that's not the full story. There is quite a large standard deviation at 27.8 and at least one 0 score present in the dataset. I'll plot the popularity and release year variables on a histogram to get a better idea of the distribution.

[10]:
```
#define the size of the figure
fig,ax = plt.subplots(1,2,figsize=(14,6))

#histogram plot generation
sns.histplot(data=df,x='popularity',color='royalblue',ax=ax[0])
sns.histplot(data=df,x='release_year',bins=20,color='royalblue',ax=ax[1])

#title and label generation
```

```python
plt.suptitle("Playlist Track Popularity and Year Distribution",fontsize=16)
ax[0].set_title("Track Popularity Distribution")
ax[1].set_title("Track Release Year Distribution")
ax[0].set_xlabel("Popularity")
ax[1].set_xlabel("Year")

#tweak proximity of plots to each other
fig.subplots_adjust(left=None,
    bottom=None,
    right=None,
    top=None,
    wspace=0.3,
    hspace=0.1,)

plt.show()
```



That's a lot of songs with a popularity score close to 0! It's important to consider that popularity scores are derived from a spotify algorithm that considers the number of times a song is played and how recent those plays are (Spotify, n.d. -a). It's also important to consider that every song on spotify is ranked independently, so duplicates of the same song can have different scores. One version of the song on Spotify could have a signifcantly different score compared to another version (e.g. explicit vs non-explicit). It's possible that the songs ranked at 0 in the playlist are a mix of songs I like that aren't popular and songs that are not from the original album. Further analysis would be required to determine the maximum popularity of a given song from that subset of data. Looking at the rest of the popularity data shows generally the songs in the playlist tend toward a high popularity.

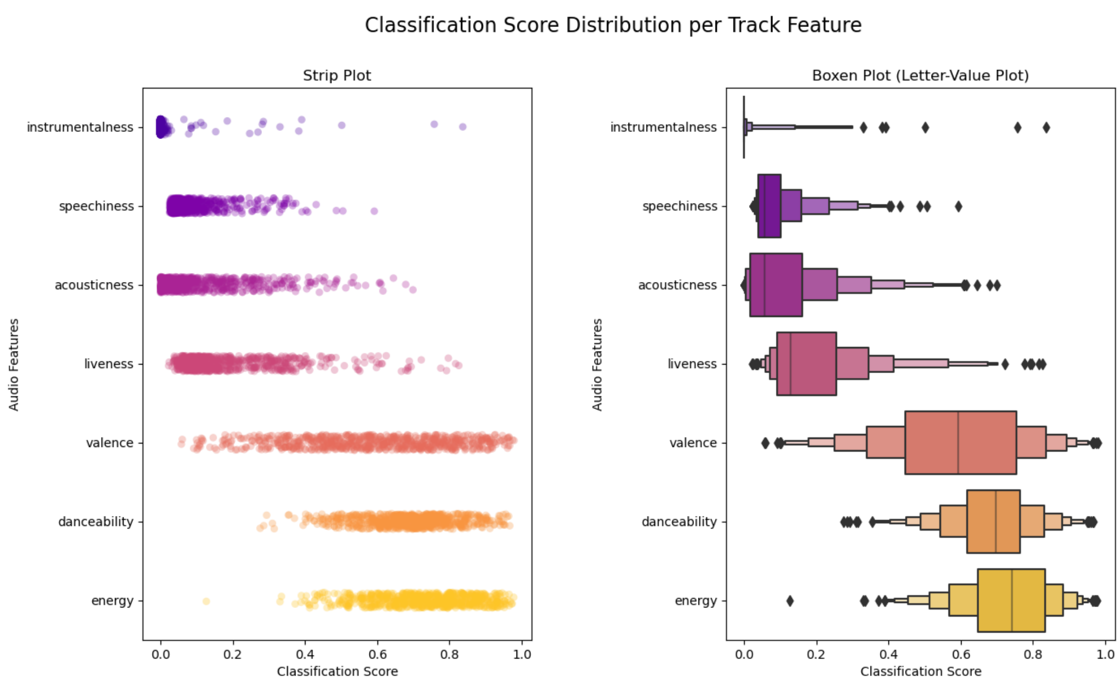
Looking at the release years show an expotential trend of increasing track counts as the year increases. There are a handful of songs that are added from 1968 to around the mid 1990's, but the quantity quickly increases afterwards. This data shows that the songs in the playlist tend toward more modern songs.

What about the song audio features? I'll plot those as well to get a better idea of their behavior. I'll first transform the data into something easier to plot. Next, I'll generate two plots. The first plot will be a strip plot with all data points plotted for each audio feature. The second plot will be a boxen plot (letter-value plot) for each audio feature.

[11]:
```
#Transform dataframe for all audio features to be in one column with their␣
↪respective values in a second column
features_df_transformed = pd.melt(features_df,value_vars=['acousticness',
                                                          'danceability',
                                                          'energy',
                                                          'instrumentalness',
                                                          'liveness',
                                                          'speechiness',
                                                          'valence'])
```

[12]:
```
#define the size of the figure
fig,ax = plt.subplots(1,2,figsize=(14,8))

#boxen plot generation
sns.boxenplot(data=features_df_transformed,
              y='variable',
              x='value',
              palette='plasma',
              ax=ax[1],
              order=['instrumentalness',
                     'speechiness',
                     'acousticness',
                     'liveness',
                     'valence',
                     'danceability',
                     'energy'])

#strip plot generation
sns.stripplot(data=features_df_transformed,
              y='variable',
              x='value',
              size=6,
              alpha=0.3,
              order=['instrumentalness',
                     'speechiness',
                     'acousticness',
                     'liveness',
                     'valence',
                     'danceability',
                     'energy'],
```

```
                palette='plasma',
                ax=ax[0])

#title and label generation
plt.suptitle("Classification Score Distribution per Track Feature",fontsize=16)
ax[0].set_title("Strip Plot")
ax[1].set_title("Boxen Plot (Letter-Value Plot)")
ax[0].set_ylabel("Audio Features")
ax[1].set_ylabel("Audio Features")
ax[0].set_xlabel("Classification Score")
ax[1].set_xlabel("Classification Score")

#tweak proximity of plots to each other
fig.subplots_adjust(left=None,
    bottom=None,
    right=None,
    top=None,
    wspace=0.5,
    hspace=0.1,)

plt.show()
```



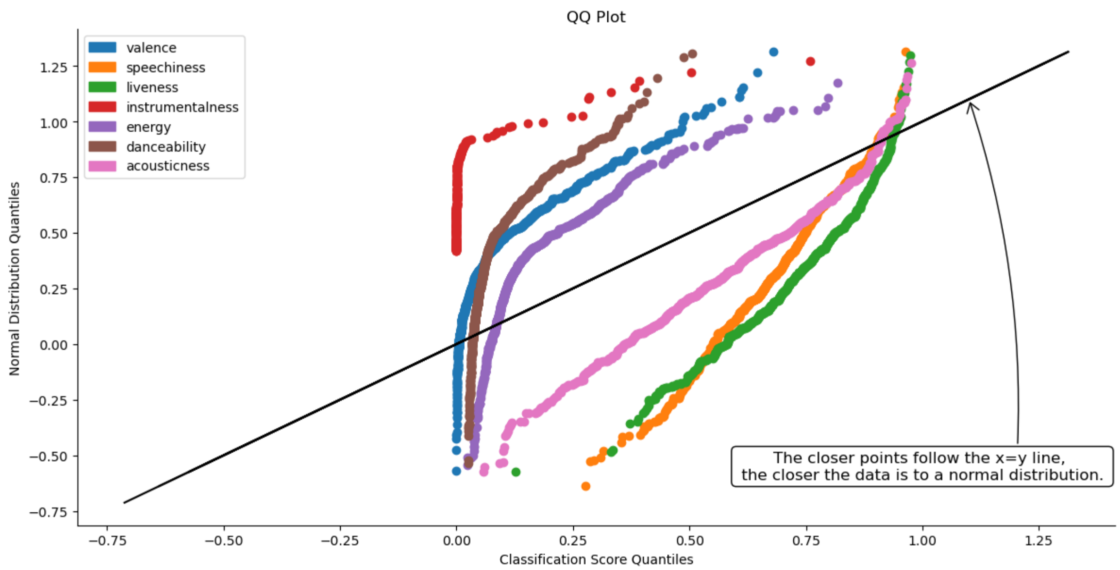
Investigating the strip plot first shows the overall density of points across the classification score scale from 0 to 1. An alpha value (translucency) of 0.3 was used to better highlight overlapping values. The audio features have been sorted in descending order from smallest mean to largest mean for the dataset. Reviewing the strip plot and correlating back to Table 1 helps to visualize the general concentration of points thus the general characteristics of my playlist. A very low instrumentalness, speechiness, acousticness, and liveness can be observed. A relatively high danceability and energy can also be seen. This suggests that the playlist has a lot of songs that are danceable; are fast, loud, and noisy; are not live recordings; and have enough vocals to not be instrumental while also not being a podcast or audio book. Valence is fairly evenly distributed suggesting that the playlist has a mix of positive and negative sounding songs. There are also many potential outliers in the data, but a strip plot doesn't give a lot of descriptive information. That's where a boxen plot comes in.

The second plot is a boxen plot (letter-value plot) that gives a bit more information on the data distribution. The largest two boxes in the middle represent 50% of the data. The next two boxes (one on each side) together represent another 25% of the data. Each series of boxes cuts the percentage in half from the previous (12.5%, 6.25%, etc.). The boxen plot helps confirm where the largest concentrations of data points are as well as the outliers in the data (represented by the black diamonds). The majority of the data appears to be fairly concentrated for each audio feature, but there are significant tails present as well as outliers for each audio feature.

Can this data help determine which songs would be a good or bad fit for my playlist? Ideally, a mathematical approach should be used to guarantee the best possible accuracy in our predictions. The first step in my quest for picking the best mathematical solution is to determine if I am dealing with a normal distribution or not. Normal distributions can help make our lives a lot easier, but unfortunately my boxen plot helps visualize that my data may not fit a normal distribution too well. I can double check this by utilizing a more specific tool called a QQ plot. While this won't tell me what the distribution is, it can tell me what it's NOT. I'll plot the data on the QQ plot to determine if any audio features align with a normal distribution.

[13]:
```
#pplot generation
pplot(features_df_transformed,y=sci.
 ↪norm,x='value',hue='variable',kind='qq',height=6,aspect=2,display_kws={"identity":
 ↪True})

#title, label, annotation generation
plt.title("QQ Plot")
plt.ylabel("Normal Distribution Quantiles")
plt.xlabel("Classification Score Quantiles")
plt.annotate('The closer points follow the x=y line, \n the closer the data is␣
 ↪to a normal distribution. ',
             xy=(1.1, 1.1),
             xycoords='data',
            xytext=(1, -0.55),
             bbox=dict(boxstyle="round", fc="w"),
             va="center",
             ha="center",
             size=12,
            arrowprops=dict(arrowstyle="->,head_width=0.4,head_length=0.
 ↪6",facecolor='black',connectionstyle="arc3,rad=0.1",fc="w",relpos=(0.75, 0.
 ↪)))
```

```
plt.show()
```



What I am looking for with a QQ plot is whether the points closely follow the x=y line for the given distribution, but unfortunately it seems none of my audio features are well defined by a normal distribution. I could use this opportunity to determine the best possible distribution model for each individual audio feature with tools such as the `fitter` python library, but for simplicity's sake, there is an alternative option I can employ. I can use Chebyshev's inequality.

Chebyshev's inequality (also called Bienaymé–Chebyshev inequality) states that:

> "For a wide class of probability distributions, no more than a certain fraction of values can be more than a certain distance from the mean. Specifically, no more than $\frac{1}{k^2}$ of the distribution's values can be k or more standard deviations away from the mean" ("Chebyshev's inequality", 2023).

$$Pr(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2}$$

In a normal distribution, I could confidently say that 95% of the data would fit within 2 standard deviations from the mean. Using Chebyshev's inequality, I can conclude that at least 75% of the data will fit within 2 standard deviations of the mean for a wide range of different distributions. Using these bounds does not give me the same level of accuracy I can expect with a normal distribution, but it does allow us to better identify values in our data which would correlate to a song better fitting our playlist.

5 Putting the Data into Action

Now that I've identified the general characteristics of my playlist data, I can use this information to help predict which songs may fit better into my playlist. I will use a range of values 2 standard deviations from the mean for each audio feature to help develop a filter for the type of songs I want. I will upload a new dataframe of over 6,000 songs (my liked songs on Spotify) to identify which would best fit my new criteria.

For my analysis, I will start by calculating the upper and lower bounds of my target values based on 2 standard deviations from the mean.

[14]:
```
pop_min = df['popularity'].mean() - 2*df['popularity'].std()
pop_max = df['popularity'].mean() + 2*df['popularity'].std()
year_min = df['release_year'].mean() - 2*df['release_year'].std()
year_max = df['release_year'].mean() + 2*df['release_year'].std()
acoust_min = features_df['acousticness'].mean() - 2*features_df['acousticness'].
 ↪std()
acoust_max = features_df['acousticness'].mean() + 2*features_df['acousticness'].
 ↪std()
dance_min = features_df['danceability'].mean() - 2*features_df['danceability'].
 ↪std()
dance_max = features_df['danceability'].mean() + 2*features_df['danceability'].
 ↪std()
en_min = features_df['energy'].mean() - 2*features_df['energy'].std()
en_max = features_df['energy'].mean() + 2*features_df['energy'].std()
instru_min = features_df['instrumentalness'].mean() -␣
 ↪2*features_df['instrumentalness'].std()
instru_max = features_df['instrumentalness'].mean() +␣
 ↪2*features_df['instrumentalness'].std()
live_min = features_df['liveness'].mean() - 2*features_df['liveness'].std()
live_max = features_df['liveness'].mean() + 2*features_df['liveness'].std()
speech_min = features_df['speechiness'].mean() - 2*features_df['speechiness'].
 ↪std()
speech_max = features_df['speechiness'].mean() + 2*features_df['speechiness'].
 ↪std()
val_min = features_df['valence'].mean() - 2*features_df['valence'].std()
val_max = features_df['valence'].mean() + 2*features_df['valence'].std()
```

I will now pull the full dataset of over 6000 songs into a new dataframe, pull the associated audio features for each song into a different dataframe, and merge the dataframes together as I did in the initial exercise.

[15]:
```
remaining_songs = sp.current_user_saved_tracks()['total']
offset=0
limit=50
saved_tracks = []

while True:
```

```python
    spotify_liked_songs = sp.current_user_saved_tracks(limit,offset)
    tracks = spotify_liked_songs['items']

    for track in tracks:

        saved_tracks.append((track['track']['id'],
                             track['track']['name'],
                             track['track']['album']['release_date'],
                             track['track']['artists'][0]['name'],
                             track['track']['artists'][0]['id'],
                             track['track']['popularity']))

    remaining_songs-=limit

    if remaining_songs <=0:
        break
    else:
        offset+=limit

liked_songs_df = pd.
 ↪DataFrame(data=saved_tracks,columns=['track_ID','name','release_date','artist','artist_ID',
```

[16]:
```python
remaining_songs = sp.current_user_saved_tracks()['total']
track_ids = list(liked_songs_df['track_ID'])
track_subset = []
features_list = []
offset = 0
limit = 50

while True:

    track_subset = track_ids[offset:offset+limit]

    audio_features_results = sp.audio_features(track_subset)

    for index,track in enumerate(audio_features_results):

        try:
            feature_tuple = (track['id'],
                             track['acousticness'],
                             track['danceability'],
                             track['energy'],
                             track['instrumentalness'],
                             track['liveness'],
                             track['speechiness'],
                             track['valence'])
```

```
            features_list.append(feature_tuple)

        except TypeError:
            feature_tuple = (track_subset[index],
                             "DEFAULT",
                             "DEFAULT",
                             "DEFAULT",
                             "DEFAULT",
                             "DEFAULT",
                             "DEFAULT",
                             "DEFAULT",)

            features_list.append(feature_tuple)


    remaining_songs-=limit

    if remaining_songs <=0:
        break
    else:
        offset+=limit

liked_songs_features_df = pd.DataFrame(data=features_list,columns=['track_ID',
                                                                    ␣
 ↪'acousticness',
                                                                    ␣
 ↪'danceability',
                                                                     'energy',
                                                                    ␣
 ↪'instrumentalness',
                                                                     'liveness',
                                                                    ␣
 ↪'speechiness',
                                                                     'valence'])
```

[17]:
```
#remove duplicates in both dataframes then generate a new dataframe by merging␣
 ↪both dataframes on 'track_ID. At the end of the merge, create a new␣
 ↪release_year column.'
liked_songs_df = liked_songs_df.drop_duplicates(subset=['track_ID'])
liked_songs_features_df = liked_songs_features_df.
 ↪drop_duplicates(subset=['track_ID'])
merged_df = pd.
 ↪merge(liked_songs_df,liked_songs_features_df,how='inner',on='track_ID')
merged_df['release_year'] = merged_df['release_date'].apply(release_year)
```

```python
#loop through the new dataframe playlist_df and delete any rows containing␣
↪'DEFAULT'. If one column is empty, all columns are empty.
index = []
empty_tracks_df = merged_df[merged_df['acousticness'] == 'DEFAULT']

for val in empty_tracks_df.index:
    merged_df = merged_df.drop(val,axis=0)

merged_df
```

[17]:

```
                    track_ID  \
0     0AAMnNeIc6CdnfNU85GwCH
1     7njDhlprmHJ1I9pMOrxMON
2     0LRh8jRfWbP0KPkXzfJUXg
3     0O85RmVuOTvfjOoHIVrZlv
4     1Mn12q06NlGF3EdDkugUAi
…                        …
6134  6FxMzrdk9dmjUMLdHeL5tl
6135  6HOcAwQAAKaj1GHntfmSoI
6136  6aKWBLAFHXHUjsJJri2ota
6137  75CgwX1tMoPfE8C4ZR14D8
6138  6HFbq7cewJ7rPiffV0ciil

                                                   name release_date  \
0     Self Love (Spider-Man: Across the Spider-Verse…   2023-06-02
1     Barbie Dreams (feat. Kaliii) [From Barbie The …   2023-07-06
2                                  We Don't Need Malibu   2023-06-23
3     Ring Ring (feat. Travis Scott, Don Toliver, Qu…   2023-07-05
4                                              Deep End   2023-07-07
…                                                   …          …
6134                                  Miss Atomic Bomb   2013-01-01
6135          When You Were Young - Calvin Harris Remix   2013-01-01
6136                                When You Were Young   2013-01-01
6137                                For Reasons Unknown   2013-01-01
6138                                A Sky Full of Stars   2014-05-02

              artist               artist_ID  popularity acousticness  \
0       Metro Boomin  0iEtIxbK0KxaSlF7G42ZOp          91        0.211
1        FIFTY FIFTY  4GJ6xDCF5jaUqD6avOuQT6          65        0.378
2             Valley  7blXVKBSxdFZsIqlhdViKc          52        0.654
3            CHASE B  2cMVIRpseAO7fJAxNfg6rD          60        0.014
4     Landon Conrath  2PJO6l59DomDd440az768u          40        0.398
…                …                       …         …          …
6134     The Killers  0C0XlULifJtAgn6ZNCW2eu          42       0.0234
6135     The Killers  0C0XlULifJtAgn6ZNCW2eu          45     0.000162
6136     The Killers  0C0XlULifJtAgn6ZNCW2eu          41     0.000335
6137     The Killers  0C0XlULifJtAgn6ZNCW2eu          43     0.000433
```

```
6138        Coldplay  4gzpq5DPGxSnKTe4SA8HAU         60      0.00617

     danceability energy instrumentalness liveness speechiness valence  \
0           0.775  0.298         0.000233    0.132      0.0517  0.0472
1            0.72  0.794                0    0.122      0.0862   0.856
2           0.675  0.616          0.00349    0.335       0.132   0.714
3           0.689  0.713         0.000061    0.189      0.0312   0.231
4           0.726  0.661                0    0.414      0.0319   0.715
…             …    …                 …      …           …      …
6134         0.56  0.722         0.000027    0.257      0.0315   0.348
6135        0.586  0.918           0.0543    0.199      0.0942   0.459
6136        0.441  0.976           0.0475    0.298       0.147   0.248
6137        0.496  0.889           0.0415    0.122      0.0372   0.519
6138        0.545  0.675          0.00197    0.209      0.0279   0.162

      release_year
0             2023
1             2023
2             2023
3             2023
4             2023
…            …
6134          2013
6135          2013
6136          2013
6137          2013
6138          2014

[6138 rows x 14 columns]
```

We can see that I successfully pulled 6138 songs into my dataframe. Now, I'll finally update the data type of the columns and create a new dataframe by applying a filter for only tracks that meet my criteria. After the filter is applied, I need to make sure I only include unique songs in this new dataframe that don't already exist in my playlist. I'll do this by merging my playlist dataframe to my new tracks dataframe and use an indicator to find when a value exists in both dataframes. I'll build a new dataframe to only include values that are not present in both the previous dataframes.

[18]:
```python
#change all columns from objects to numeric values
merged_df['acousticness'] = pd.to_numeric(merged_df['acousticness'])
merged_df['energy'] = pd.to_numeric(merged_df['energy'])
merged_df['danceability'] = pd.to_numeric(merged_df['danceability'])
merged_df['valence'] = pd.to_numeric(merged_df['valence'])
merged_df['liveness'] = pd.to_numeric(merged_df['liveness'])
merged_df['speechiness'] = pd.to_numeric(merged_df['speechiness'])
merged_df['instrumentalness'] = pd.to_numeric(merged_df['instrumentalness'])

#generate a new dataframe based on our filter targets
```

```
new_tracks_df = merged_df[(merged_df['popularity']>pop_min) &␣
↪(merged_df['popularity']<pop_max) &
                  (merged_df['energy']>en_min) & (merged_df['energy']<en_max) &
                  (merged_df['danceability']>dance_min) &␣
↪(merged_df['danceability']<dance_max) &
                  (merged_df['valence']>val_min) &␣
↪(merged_df['valence']<val_max) &
                  (merged_df['liveness']>live_min) &␣
↪(merged_df['liveness']<live_max) &
                  (merged_df['acousticness']>acoust_min) &␣
↪(merged_df['acousticness']<acoust_max) &
                  (merged_df['speechiness']>speech_min) &␣
↪(merged_df['speechiness']<speech_max) &
                  (merged_df['instrumentalness']>instru_min) &␣
↪(merged_df['instrumentalness']<instru_max) &
                  (merged_df['release_year']>year_min) &␣
↪(merged_df['release_year']<year_max)]
```

[19]:
```
df_all = pd.merge(new_tracks_df, df, on=['name','name'],␣
↪how='left',indicator=True)
unique_tracks_df = df_all[df_all['_merge'] == 'left_only']
print('There are ' + str(unique_tracks_df['_merge'].count()) + ' unique songs␣
↪that meet my playlist criteria')
```

```
There are 2255 unique songs that meet my playlist criteria
```

Out of 6138 available tracks, I found 2255 that could potentially fit into my playlist. I reduced the total number of songs by about 63.3%!

The last thing I'll do is export the potential songs out to an excel sheet and I'm done!

[20]:
```
unique_tracks_df.to_excel('new_tracks.xlsx',index=False)
```

6 Conclusion

6.1 Summary

Based on the analysis of the playlist "Suns Out Guns Out", we can conclude the following about the playlist:

- Songs increase in quantity as release year increases
- Songs tend to be more popular, but a significant number of songs appear closer to a score of 0
- Songs are generally danceable
- Songs are generally fast, loud, and noisy
- Songs are not live
- Songs are vocal enough to not be instrumental
- Songs are not vocal enough to be considered an audio book or podcast
- The playlist is a fairly even mix of positive and negative sounding songs with a slight positive bias
- Distributions of the audio features do not follow a normal distribution, thus we can conclude at least 75% of data points are within 2 standard deviations from the mean (Chebyshev's inequality)
- We can use Chebyshev's inequality to narrow down compatible songs for our playlist

6.2 Next Steps

- Analysis on songs with lower popularity scores within the "Suns Out Guns Out" playlist to determine potential causes
- Train and deploy a machine learning model for advanced recommendations on songs
- Compare audio features between our chosen playlist and that of a different type of playlist (e.g. studying/relaxation playlist)
- Identify trends in types of genres that fit with my playlist's audio features

7 References

Chebyshev's inequality. (2023, April 9). In *Wikipedia.*

https://en.wikipedia.org/wiki/Chebyshev%27s_inequality

Spotify (n.d. -a). *Get Playlist Items.* Spotify for Developers.

https://developer.spotify.com/documentation/web-api/reference/get-playlists-tracks

Spotify (n.d. -b). *Get Track's Audio Features.* Spotify for Developers.

https://developer.spotify.com/documentation/web-api/reference/get-audio-features