Comparing Song Audio Features to Rankings on The Billboard Hot 100

Kevin Carr 501150122
Supervisor: Ceni Babaoglu, PhD
Date: October, 2022

Ryerson University

Table of Contents

Abstract

The central problem in this project is to utilise publicly available audio feature data to predict whether or not a song is likely to appear on the Billboard Hot 100 charts. Music streaming services employ data models to characterise audio features for songs. This data is provided and publicly available for multiple streaming services, notably Spotify. The Billboard Hot 100 has been a music industry standard for approximately 70 years. The Billboard Hot 100 contains weekly rankings for songs which are based on sales, plays, and surveys.

Audio feature data and Billboard Hot 100 chart data for this project have been obtained from multiple sources and combined. Billboard Hot 100 charts were obtained for the entire history of the Billboard Hot 100, from 1958 to 2021. Two additional databases were obtained consisting of approximately 10 million songs with audio feature data. Were possible, this audio feature data was merged with songs from the Billboard Hot 100 charts. In cases where audio feature data for Billboard Hot 100 songs were not in available datasets, missing audio features were obtained using the Spotify API, where available. Audio features were obtained for approximately 75% of songs in the Billboard Hot 100 charts, as well as an approximately 10 million additional songs.

For this project, two main techniques will be employed, namely data mining and predictive analytics. First, data mining and knowledge discovery will be used to explore the data, cluster audio features, and determine correlations between audio features. Second, predictive analytics will be used to attempt to build a predictive model using the data. By utilising knowledge discovered during the data mining phase of the project, predictive analysis will be broken into

sets of clustered genres or songs with similar audio features. It is anticipated that this segmentation will improve predictive accuracy and lead to further insights.

Introduction

Music streaming services employ data models to characterise audio features for songs, and use this data to recommend songs and playlist to their listeners. This data is provided and publicly available for multiple streaming services, notably Spotify (Spotify, n.d.).

The Billboard Hot 100 has been a music industry standard for approximately 70 years (Wikipedia, 2022). The Billboard Hot 100 contains weekly rankings for songs which are based on sales, plays, and surveys.

For this project, data mining and predictive analytics will be employed. Data mining and knowledge discovery will be used to explore the data, cluster audio features, and determine correlations between audio features. Predictive analytics will be used to attempt to build a predictive model.

This analysis has the potential to predict future trends in music or the performance of an individual song. These predictions could be useful to musicians or producers attempting to optimise success, or listeners looking for something new.

Literature Review

The primary focus of this literature review was to gather understanding on previous research related to music clustering techniques and the prediction of popularity for songs, especially in

cases where song audio features were used for clustering or prediction. Google Scholar (Google Scholar, n.d.), PaperDigest.org (Paper Digest, n.d.), and Elicit.org (Elicit, n.d.) were used to investigate and gather research materials.

Popularity can be defined in numerous ways. In this study, popularity will simply be considered to be an appearance on the Billboard Hot 100 charts. The magnitude of this popularity may be further be defined using total weeks on the chart, or top rank on the chart (Lee et. al., 2018). Using appearance on the Billboard Hot 100 as a metric for popularity has been used in other similar studies (Reiman et.al., 2018). Another commonly used popularity metric is 'popularity' as defined in the Spotify API (Kim, 2021; Gao, 2021). Since the Spotify 'popularity' metric does not include historical popularity data and considers only Spotify streaming, this study focusses on the Billboard Hot 100 charts in order to get a longer-term, and wider perspective of music popularity.

Audio features used in music classification have evolved through various stages. MIDI (Musical Instrument Digital Interface) format musical notation has been used to cluster music into categories (Cilibrasi et. al., 2004; Cataltepe et. al., 2007). Low-level audio features such as mel-frequency cepstral coefficients, spectral flatness, and number of zero crossings have been used to predict steaming popularity (Yang et. al., 2017; Lee et. al., 2018; Araujo et. al., 2019), improve music recommendation systems (Li et. al., 2007; Schedl, 2013), and to classify emotion in music (Jia, 2022). High-level audio features such as danceability, instrumentalness, and speechiness are included in track information available from the Spotify API. These high-level audio features have been used to identify song attributes (Febirautami et. al., 2018), predict popularity (Reiman

et.al., 2018; Martín-Gutiérrez et. al., 2020; Kim, 2021; Gao, 2021), to classify music into genres (Setiadi et. al., 2020), and to classify music into moods (Chen et. al., 2021).

Clustering music has been used for recommender systems (Li et. al., 2004; Li et. al., 2007; Huo, 2021), as well as to categorise music (Honingh et. al., 2011). In this study, overall trends, genres, and audio feature clusters will be considered to attempt to improve predictive analytics. It was hypothesised that the accuracy of predictions of song popularity using audio features could be improved by separating predictions by genre (Reiman et.al., 2018). Music genres have been used in combination with high-level audio features to predict popularity (Kim, 2021). It has been noted that audio features within the Billboard Hot 100 within genres are relatively consistent over time (O'Toole et. al., 2022).

In this study, clustering and classification will be used. Similar studies have had success with a variety of techniques and models. Neural networks have been used to predict popularity (Yang et. al., 2017; Gao, 2021), improve music recommendation systems (Li, 2021; Shi, 2021), or classify music (Jia, 2022; Li et. al., 2022). K-Means Clustering has been used in a number of studies to cluster data (Li et. al., 2007; Xu et. al., 2021; Kim et. al., 2021). A variety of classification models have been used to classify music, notably Support Vector Machines (Laurier et. al., 2009; Lee et. al., 2018; Reiman et.al., 2018; Araujo et. al., 2019; Setiadi et. al., 2020; Wilkes et. al., 2021), K-Nearest Neighbours (Cataltepe et. al., 2007; Reiman et.al., 2018; Kim, 2021), Decision Trees / Random Forests / Boosted Trees (West, 2008; Febirautami et. al., 2018; Chen et. al., 2021; Gao, 2021), and Logistic Regression (Reiman et.al., 2018; Chen et. al., 2021; Gao, 2021). In addition, Principle Component Analysis has been used to reduce dimensionality and improve predictive results (Gao, 2021).

Predicting popularity on the Billboard Hot 100 has been investigated in other studies (Lee et. al., 2018; Reiman et.al., 2018). However, no study has successfully used high-level audio features to predict popularity as defined above. Although high-level audio features were used, Reiman et.al. were not able to accurately predict song popularity. This was potentially due to an overly diverse dataset for non-hit songs. It was hypothesised that the accuracy of predictions of song popularity using audio features could be improved by separating predictions by genre (Reiman et.al., 2018). Additionally, although it has been demonstrated that popularity can be predicted using audio features alone, this was demonstrated using low-level audio features and different statistical descriptions for popularity (Lee et. al., 2018). This study aims to accurately predict song popularity, defined as appearance on the Billboard Hot 100 charts, using high-level audio features and genre data available from the Spotify API.

Data Description

Data Sources

The data gathered for this project have been taken from multiple sources and combined. Data was found using the Google dataset search engine (Google Dataset Search, n.d.).

Three of the relevant sources were found on Kaggle.com, a popular online data science community where users can share datasets (Dhruvil Dave, 2021; Malte Grosse, 2022; Rodolfo Figueroa, 2020). Audio features from the large datasets were matched with the list of songs from the Billboard Hot 100. Missing data were obtained, where available, from the Spotify API.

Data Pre-Processing and Organisation

Data Importing and preprocessing was completed in 2 stages, before and after the literature review. The two phases of dataset preprocessing are included as **Attachment 1** and **Attachment 2**, respectively.

The "Billboard 'The Hot 100' Songs" dataset (Dhruvil Dave, 2021) was available in CSV format, and includes date, rank, song title, artist, last-week, peak-rank, and weeks-on-board. This data did not include Spotify song ids or audio features, so the Spotify API was used to gather this data in cases where the audio feature data was unavailable from the other sources. This CSV was imported into Python as a Pandas dataframe.

The SQLite dataset, "8+ M. Spotify Tracks, Genre, Audio Features" (Malte Grosse, 2022) included 9 tables totalling 44 columns. The database was queried to combine song title, artist, Spotify id, release date, and audio features. The queried data were exported to CSV and imported into Python as a Pandas dataframe.

The "Spotify 1.2M+ Songs" dataset (Rodolfo Figueroa, 2020) was available in CSV format. It included combine song title, artist, Spotify id, release date, and audio features. This CSV was imported into Python as a Pandas dataframe.

Based on findings from the literature review portion of this study, genre data was gathered for easily available songs. Since genre data was included in the SQLite database for many of the songs (Malte Grosse, 2022), this data was queried, exported as CSV, and imported into Python. Since multiple genres were often available for a given artist, all genres were populated, then the results were sorted by most common genre, then duplicate song entries were dropped. This

results in songs being categorised with only one genre. It should be noted that this is the most common genre, and not necessarily the most applicable genre. It should also be noted that not all songs have genre data associated with them. Approximately 69% of songs included genre data (approximately 6.6M entries).

Songs from the Billboard Hot 100 were found in the SQLite query data, were available. Songs still missing genre data, which included Spotify id were queried using the Spotify API to gather genre data were available. Since multiple genres were often available, all genres were obtained from the API, then the result corresponding to the most common genres from the SQLite query data were populated as the song's genre. Similar to above, it should be noted that this is the most common genre, and not necessarily the most applicable genre, and not all songs have genre data associated with them.

Since Get Requests from the Spotify API were time consuming, and the utilised workflow results in timeouts after 1 hour, undefined genres in the "Spotify 1.2M+ Songs" dataset (Rodolfo Figueroa, 2020) were left undefined due to time constraints.

The data from each dataset was combined to form 4 non-distinct working datasets:

- All songs including audio features (approximately 10M entries)
- The Billboard Hot 100 historical charts (approximately 300k entries)
- All Songs from The Billboard Hot 100 historical charts (approximately 30k entries)
- Songs from The Billboard Hot 100 that include audio features and genre (approximately 20k entries)

Since the dataframes are not distinct, in cases where mutually exclusive groups were necessary, Pandas dataframe query functions and vectorized formula may be implemented to segment data as required.

Once working datasets were completed, they were exported as Parquet format. Parquet has a number of advantages over CSV format, most notably file size, the retention of data type formats, and the amount of time required to re-load the file into Python Pandas dataframe.

Quality Assurance

In order to check the accuracy of the data and consistency between datasets, a Quality Assurance (QA) check was performed on the final dataset. Audio features from a random sample of 100 songs were gathered from the Spotify API and compared to the datasets listed above.

There were 3 inconsistencies noted in 2 of the 100 songs (approximately 0.2% of audio features checked). All other inconsistencies were extremely small and appeared to be standard rounding errors. One song appeared to be remastered and reuploaded, as the majority of audio features were consistent, but the newer audio features for this track were louder (approximately 7 dB), and indicated the song was now approximately 1 second shorter in length. The other inconsistency involved inaccurate classification of the key of a song. This inconsistency was likely due to the atonal characteristics in that particular song, making the key of the song ambiguous for the purposes of audio feature classification.

Overall, there is a large degree of consistency between datasets. Furthermore, all inconsistencies are all explainable and reasonable. The data was therefore assumed to be consistent and accurate for the purposes of this study.

File and Calculation Locations

Datasets and calculations used in this study can be found at the following URL:

https://github.com/KevinCarr42/Billboard-100-Audio-Feature-Analytics

Files too large to upload to GitHub been uploaded to a shared Google Drive folder (shared with Toronto Metropolitan University Google Drive accounts):

https://drive.google.com/drive/folders/10wpORzZURV11VAUPKmCDDKxHFCvwjloR

Descriptive Statistics

Descriptive statistics for the datasets are included in the following tables. More detailed descriptive calculations are included in **Attachment 3**.

Table 1. Descriptive Statistics - All Songs With Audio Features

Audio Feature	Mean	Standard Deviation	Minimum	Quartiles			Maximum
				25%	50%	75%	
acousticness	0.42	0.37	0.00	0.03	0.34	0.82	1.00
danceability	0.53	0.19	0.00	0.40	0.55	0.68	1.00
duration_ms	238,210	159,342	0	169,600	216,933	275,080	19,672,058
energy	0.55	0.28	0.00	0.31	0.57	0.79	1.00
instrumentalness	0.26	0.37	0.00	0.00	0.00	0.65	1.00
key	5.2	3.5	0	2	5	8	11
liveness	0.21	0.18	0.00	0.10	0.13	0.26	1.00
loudness	-11.0	6.3	-60.0	-13.7	-9.2	-6.4	7.2
mode	0.66	0.47	0.00	0.00	1.00	1.00	1.00
speechiness	0.10	0.14	0.00	0.04	0.05	0.08	0.97
tempo	119	31	0	95	119	137	250
time_signature	3.84	0.57	0.00	4.00	4.00	4.00	5.00
valence	0.47	0.28	0.00	0.23	0.47	0.71	1.00

Table 2. Descriptive Statistics - Billboard Hot 100

Audio Feature	Mean	Standard Deviation	Minimum	Quartiles			Maximum
				25%	50%	75%	
rank	51	29	1	26	51	76	100
last-week	48	28	1	23	47	72	100
peak-rank	41	29	1	13	38	65	100
weeks-on-board	9	8	1	4	7	13	90
acousticness	0.28	0.28	0.00	0.04	0.18	0.47	1.00
danceability	0.60	0.15	0.00	0.51	0.61	0.71	0.99
duration_ms	226,880	66,552	30,213	183,360	221,306	258,399	1,561,133
energy	0.63	0.20	0.01	0.48	0.64	0.79	1.00
instrumentalness	0.03	0.14	0.00	0.00	0.00	0.00	0.99
key	5.2	3.6	0	2	5	8	11
liveness	0.19	0.16	0.01	0.09	0.13	0.24	1.00
loudness	-8.6	3.6	-30.3	-11.0	-8.2	-5.8	2.3
mode	0.73	0.44	0.00	0.00	1.00	1.00	1.00
speechiness	0.06	0.07	0.00	0.03	0.04	0.06	0.95
tempo	120	28	0	100	119	136	241
time_signature	3.94	0.30	0.00	4.00	4.00	4.00	5.00
valence	0.61	0.24	0.00	0.42	0.63	0.81	0.99

Table 3. Descriptive Statistics - All Songs From Billboard Hot 100 With Audio Features

Audio Feature	Mean	Standard Deviation	Minimum	Quartiles			Maximum
				25%	50%	75%	
acousticness	0.32	0.29	0.00	0.05	0.22	0.56	1.00
danceability	0.59	0.15	0.00	0.49	0.60	0.70	0.99
duration_ms	217,638	68,403	30,213	169,533	210,426	251,333	1,561,133
energy	0.61	0.20	0.01	0.46	0.62	0.78	1.00
instrumentalness	0.04	0.15	0.00	0.00	0.00	0.00	0.99
key	5.2	3.6	0	2	5	8	11
liveness	0.20	0.17	0.01	0.09	0.13	0.26	1.00
loudness	-8.9	3.6	-30.3	-11.3	-8.6	-6.1	2.3
mode	0.74	0.44	0.00	0.00	1.00	1.00	1.00
speechiness	0.07	0.08	0.00	0.03	0.04	0.06	0.95
tempo	120	28	0	100	119	136	241
time_signature	3.93	0.33	0.00	4.00	4.00	4.00	5.00
valence	0.61	0.24	0.00	0.43	0.64	0.81	0.99

Table 4. Descriptive Statistics - Songs From Billboard Hot 100 With Audio Features And Genre

Audio Feature	Mean	Standard Deviation	Minimum	Quartiles			Maximum
				25%	50%	75%	
acousticness	0.32	0.29	0.00	0.05	0.22	0.56	1.00
danceability	0.59	0.15	0.00	0.49	0.60	0.70	0.99
duration_ms	217,638	68,403	30,213	169,533	210,426	251,333	1,561,133
energy	0.61	0.20	0.01	0.46	0.62	0.78	1.00
instrumentalness	0.04	0.15	0.00	0.00	0.00	0.00	0.99
key	5.2	3.6	0	2	5	8	11
liveness	0.20	0.17	0.01	0.09	0.13	0.26	1.00
loudness	-8.9	3.6	-30.3	-11.3	-8.6	-6.1	2.3
mode	0.74	0.44	0.00	0.00	1.00	1.00	1.00
speechiness	0.07	0.08	0.00	0.03	0.04	0.06	0.95
tempo	120	28	0	100	119	136	241
time_signature	3.93	0.33	0.00	4.00	4.00	4.00	5.00
valence	0.61	0.24	0.00	0.43	0.64	0.81	0.99

Audio Feature Descriptions

Audio features available from the Spotify API used in this study are described in detail in the below table (Spotify, n.d.).

Table 5. Description of Audio Features From Spotify API

Audio Feature	Type	Description	Min	Max
acousticness	number <float>	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
danceability	number <float>	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. A value of 0.0 is least danceable and 1.0 is most danceable.	0	1
duration_ms	integer	The duration of the track in milliseconds.	0	N/A
energy	number <float>	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.	0	1
instrumentalness	number <float>	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.	0	1
key	integer	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

Audio Feature	Type	Description	Min	Max
liveness	number <float>	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.	0	1
loudness	number <float>	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). Values typically range between -60 and 0 db.	-60	0
mode	integer	Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.	0	1
speechiness	number <float>	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.	0	1
tempo	number <float>	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.	0	N/A
time_signature	integer	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
valence	number <float>	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

Histograms

Histograms for each of the audio features are shown in the below figures, comparing songs on the Billboard Hot 100 with all songs in this study.

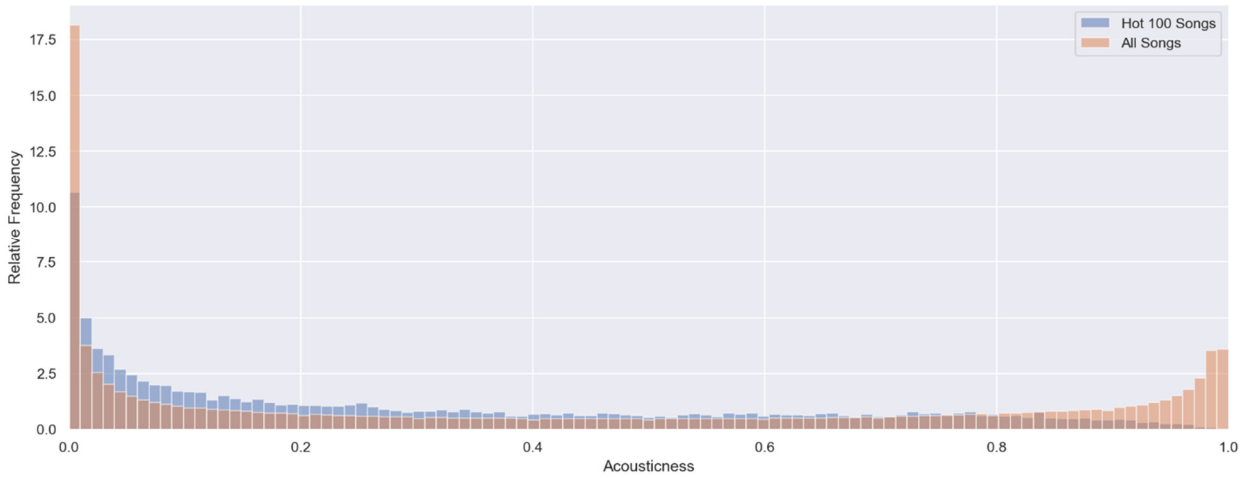


Figure 1. Acousticness Histogram

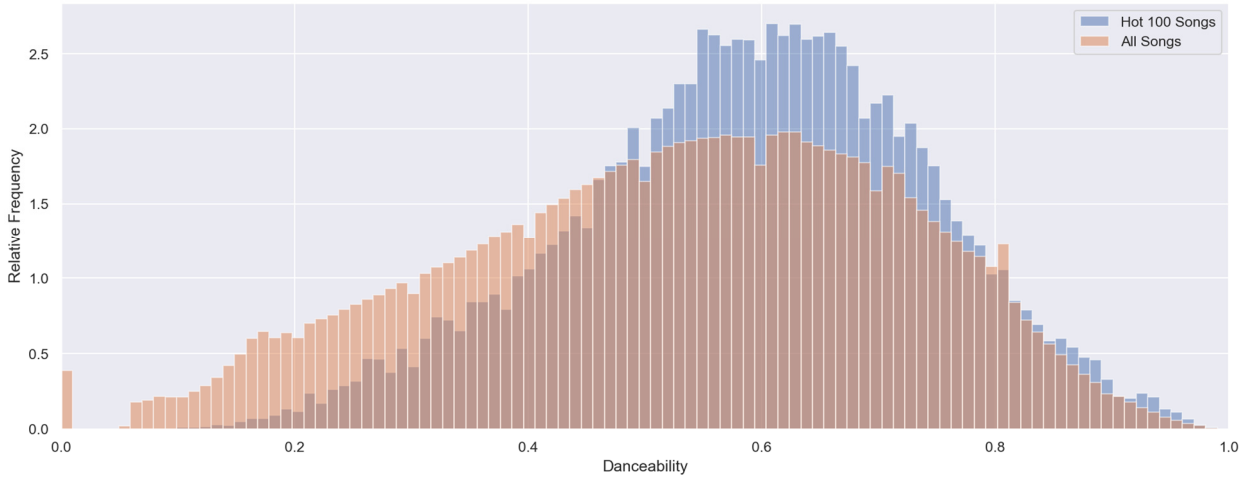


Figure 2. Danceability Histogram

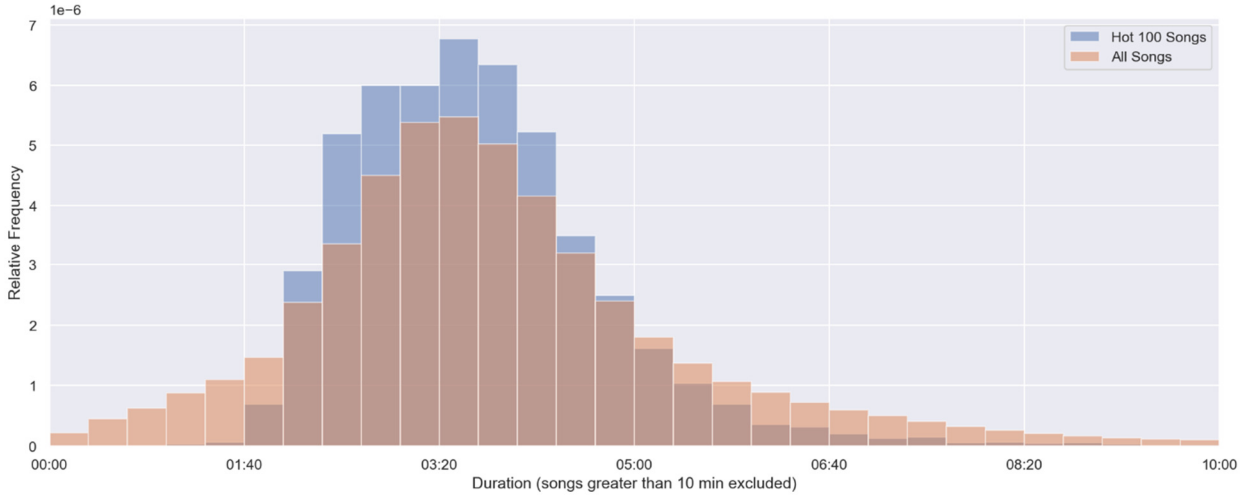


Figure 3. Duration Histogram

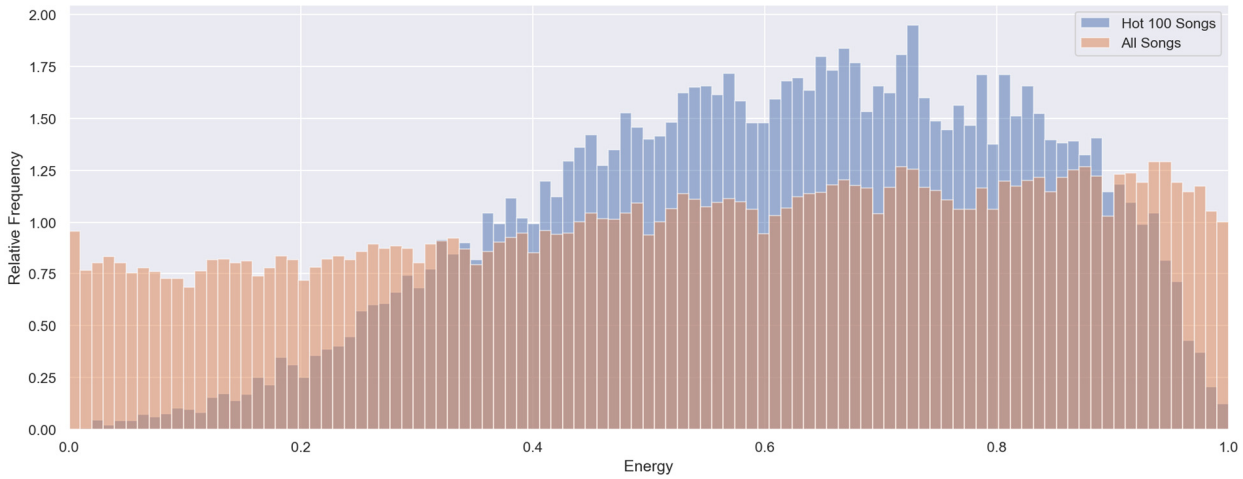


Figure 4. Energy Histogram

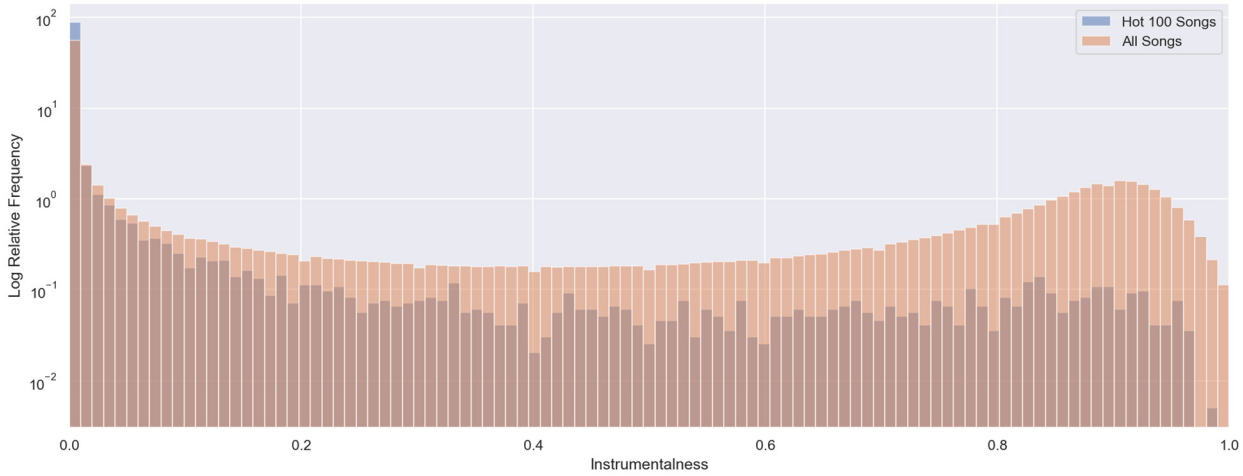


Figure 5. Instrumentalness Histogram

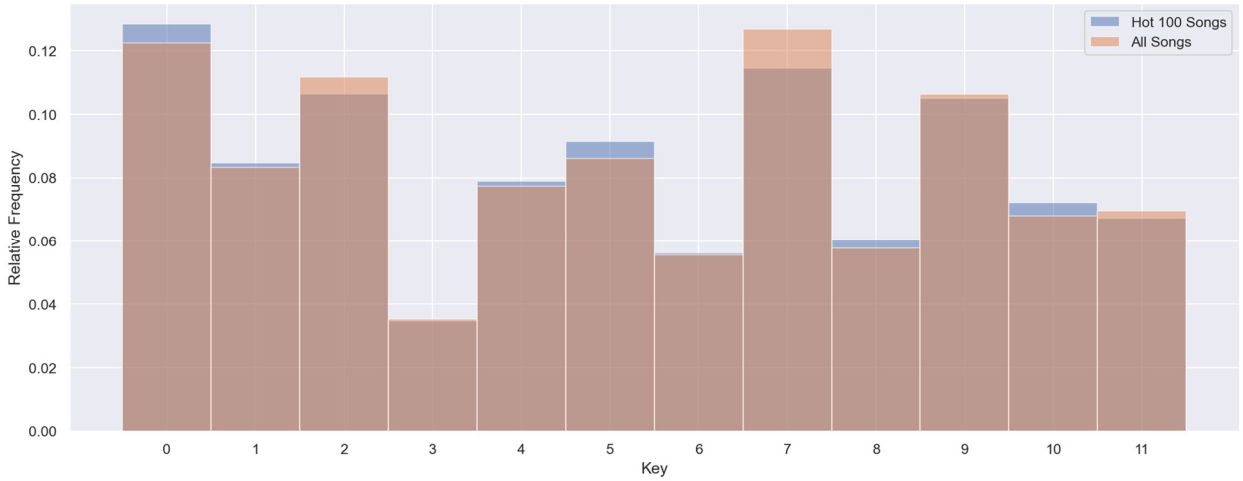


Figure 6. Key Histogram

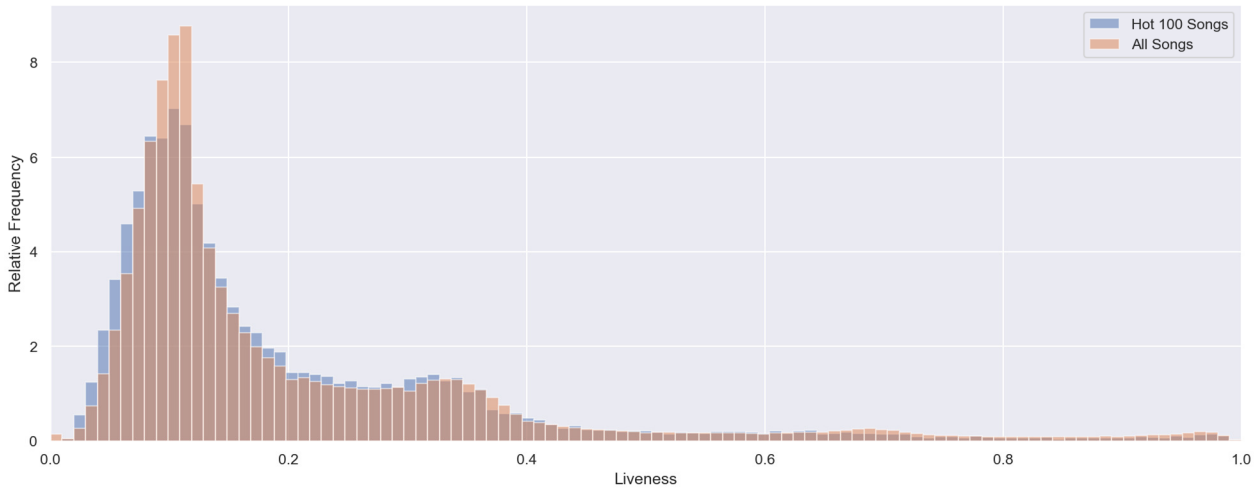


Figure 7. Liveness Histogram

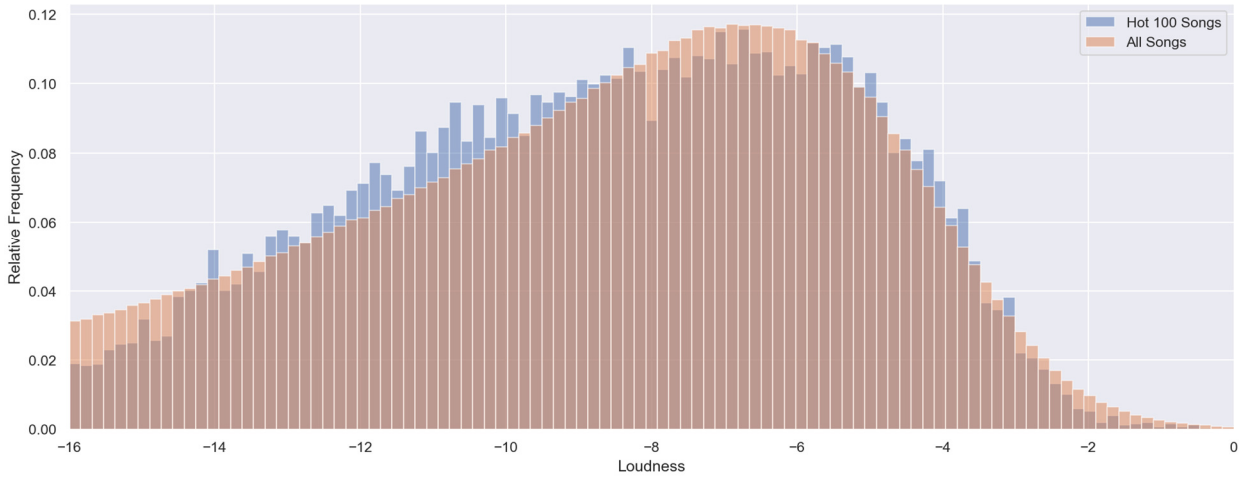


Figure 8. Loudness Histogram

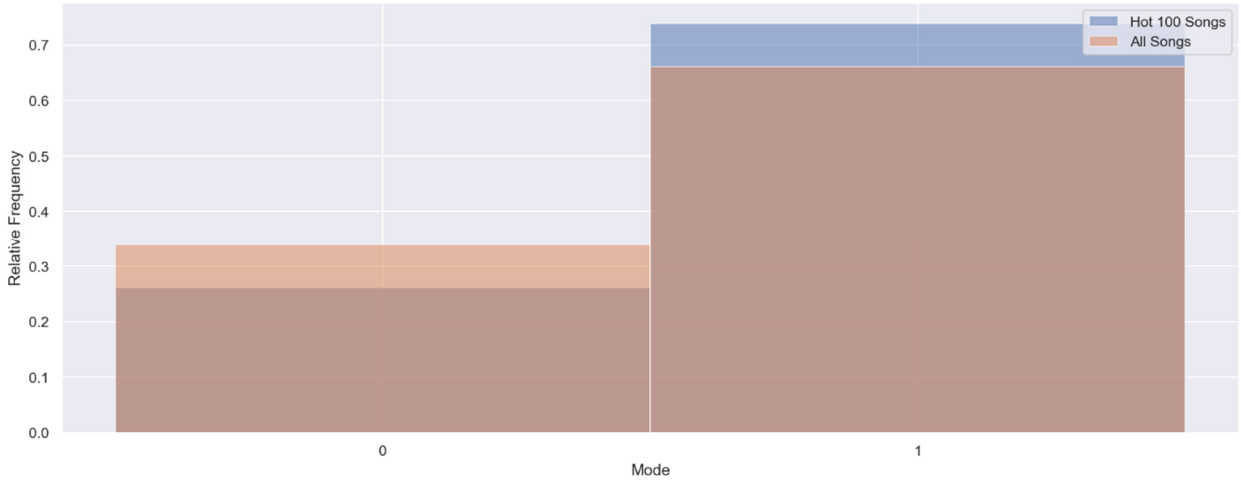


Figure 9. Mode Histogram

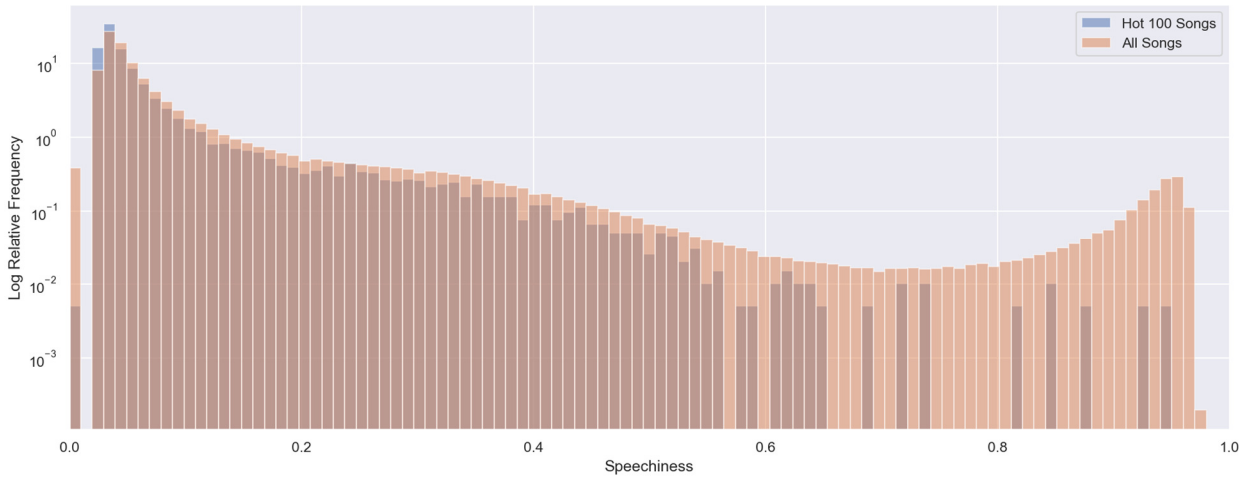


Figure 10. Speechiness Histogram

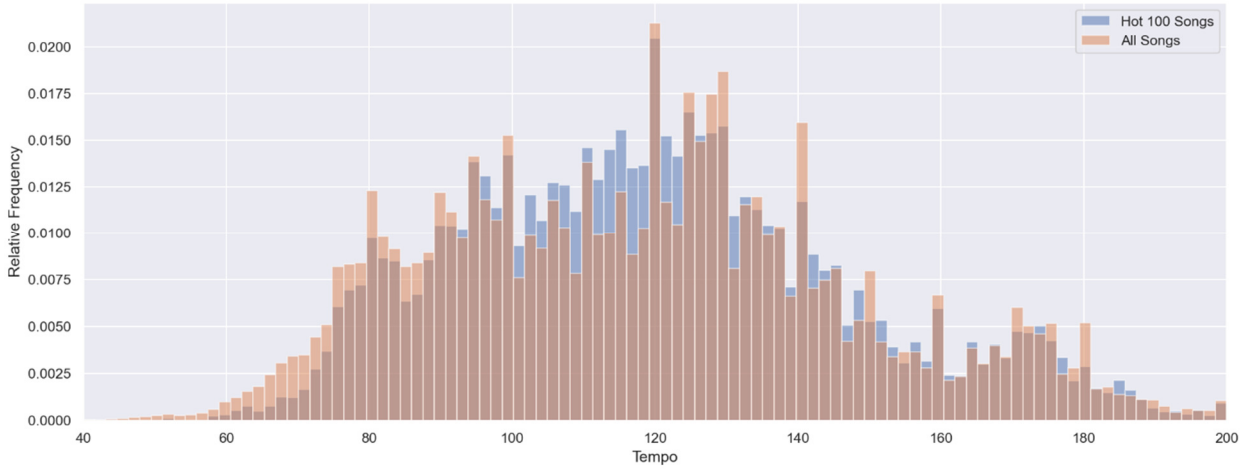


Figure 11. Tempo Histogram

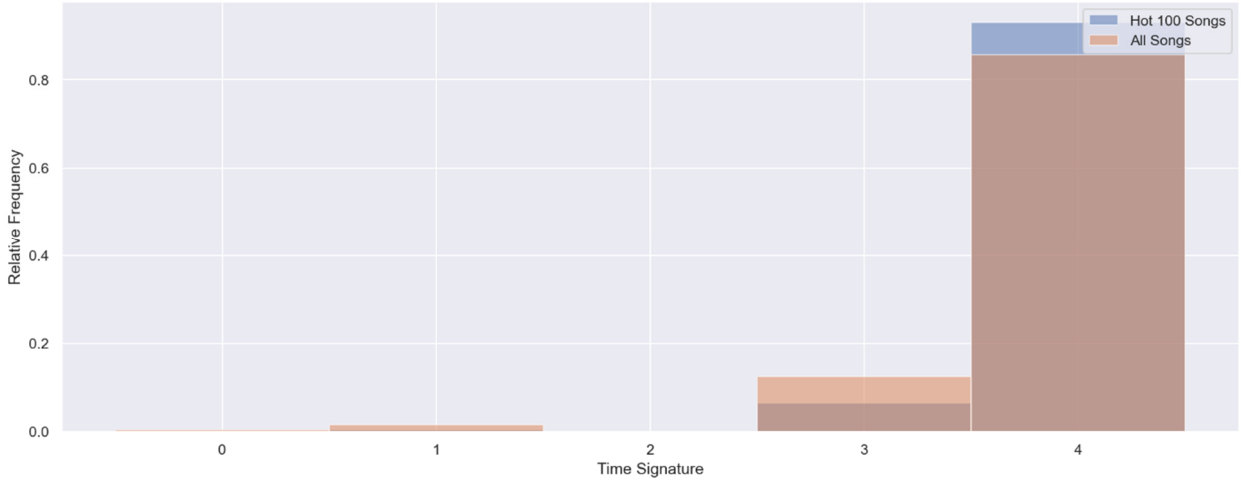


Figure 12. Time Signature Histogram

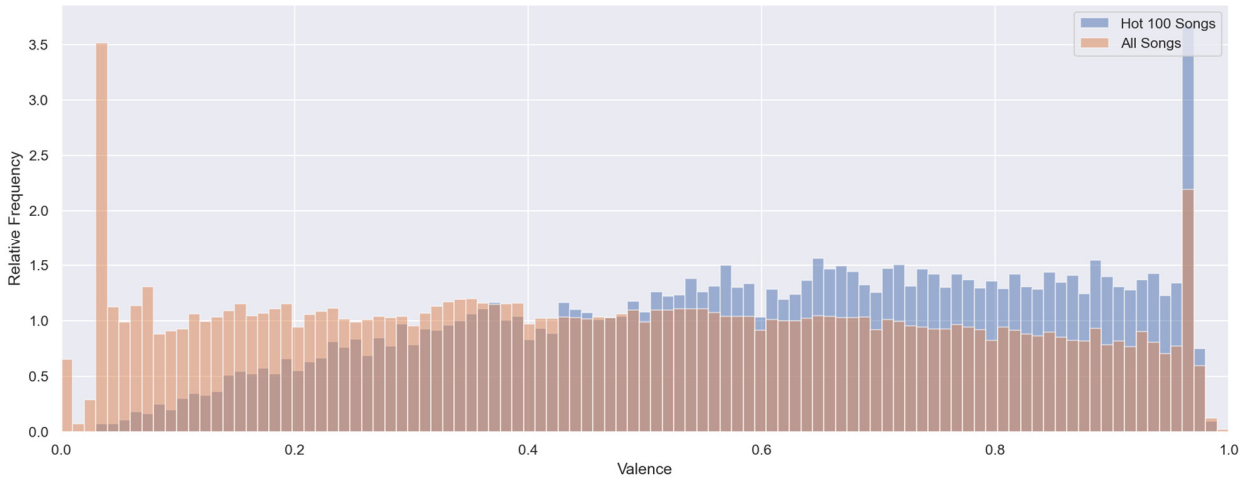


Figure 13. Valence Histogram

Historical Changes in Audio Features

Historical line plots for each of the audio features are shown in the below figures, comparing audio feature averaged by release year for songs on the Billboard Hot 100 compared with all songs in this study.

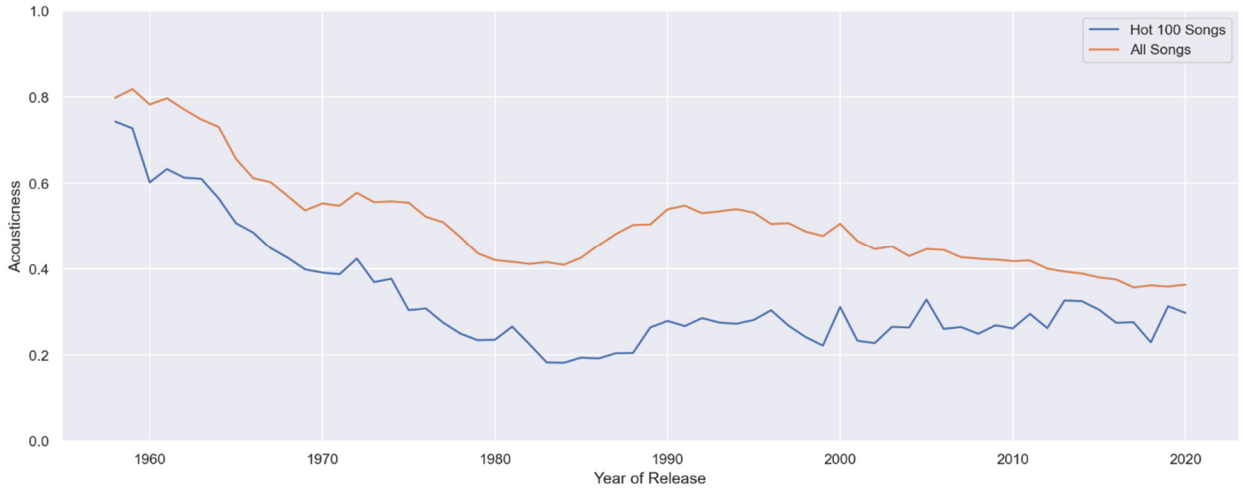


Figure 14. Acousticness History Comparing the Billboard Hot 100 with All Songs

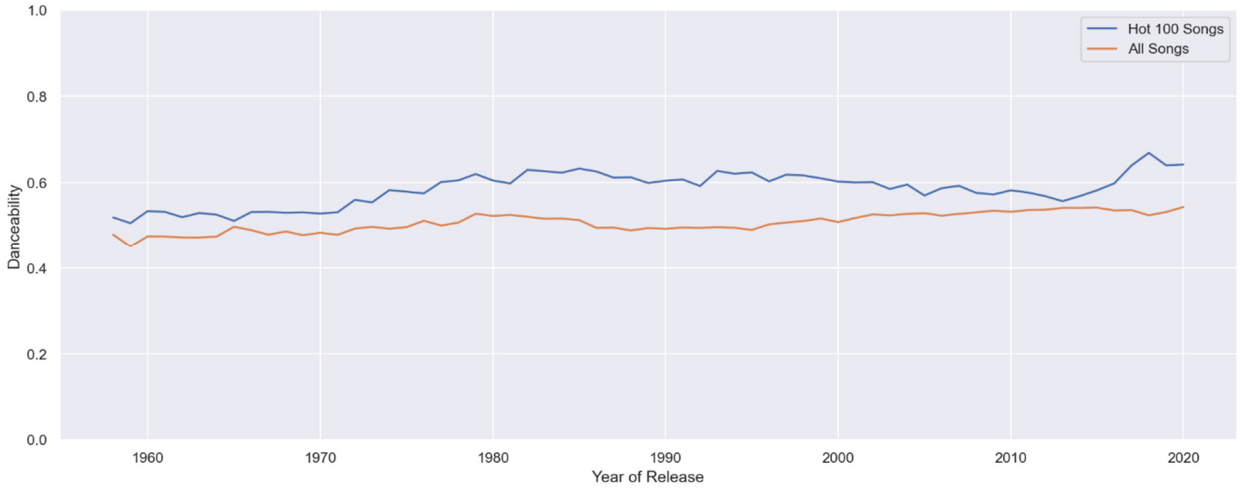


Figure 15. Danceability History Comparing the Billboard Hot 100 with All Songs

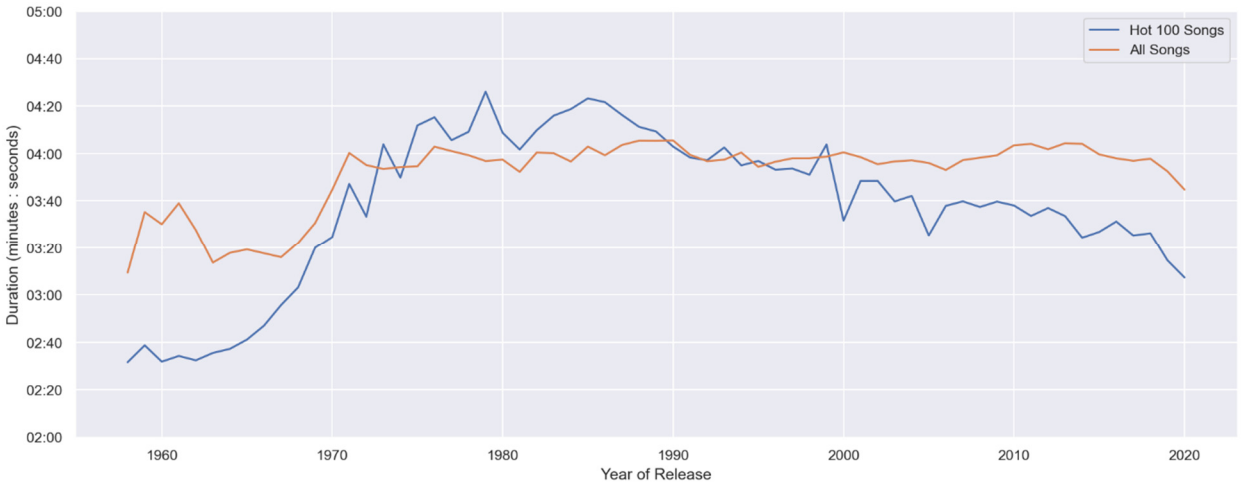


Figure 16. Duration History Comparing the Billboard Hot 100 with All Songs

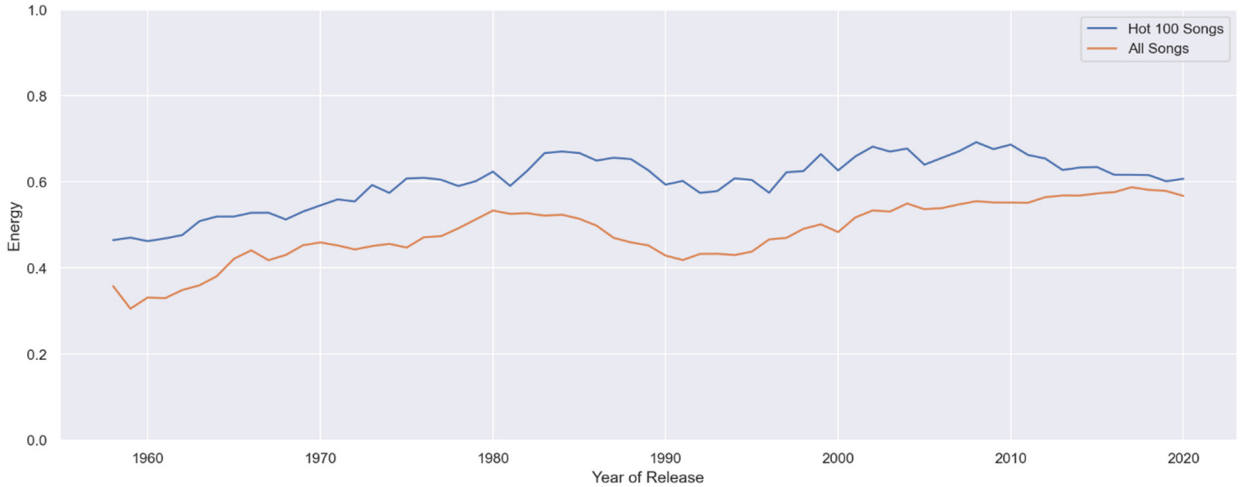


Figure 17. Energy History Comparing the Billboard Hot 100 with All Songs

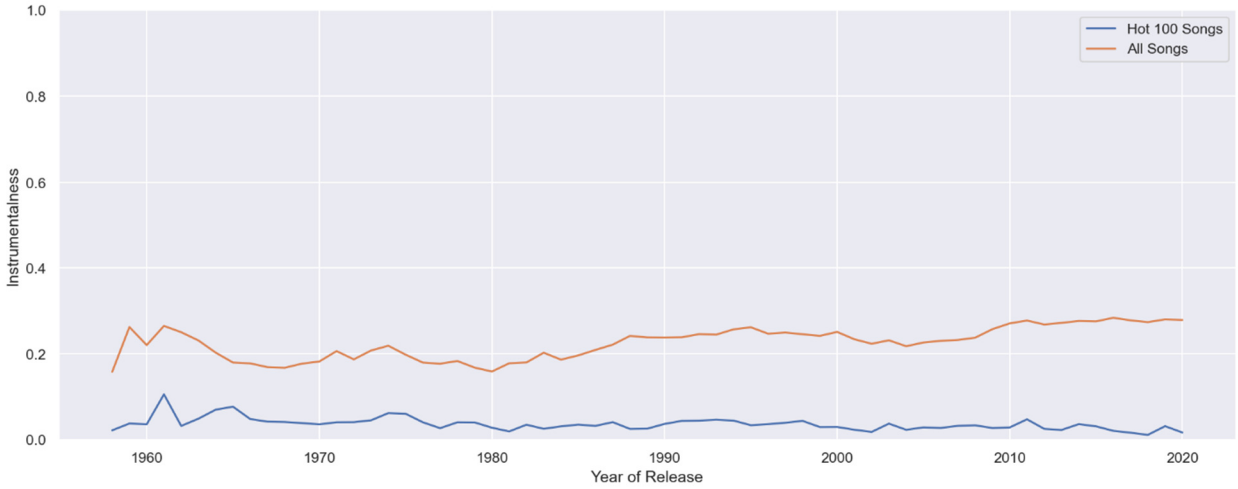


Figure 18. Instrumentalness History Comparing the Billboard Hot 100 with All Songs

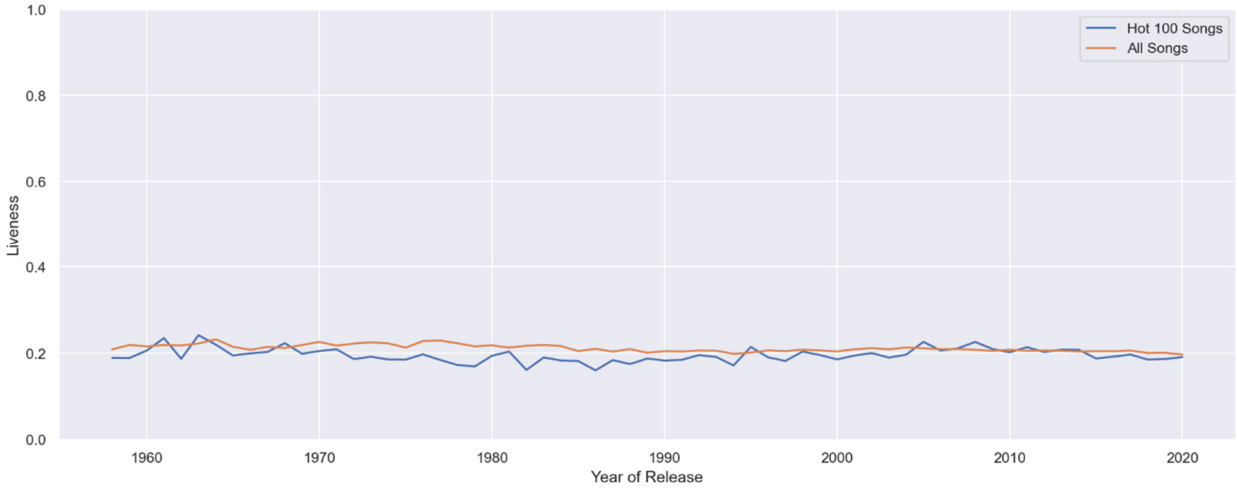


Figure 19. Liveness History Comparing the Billboard Hot 100 with All Songs

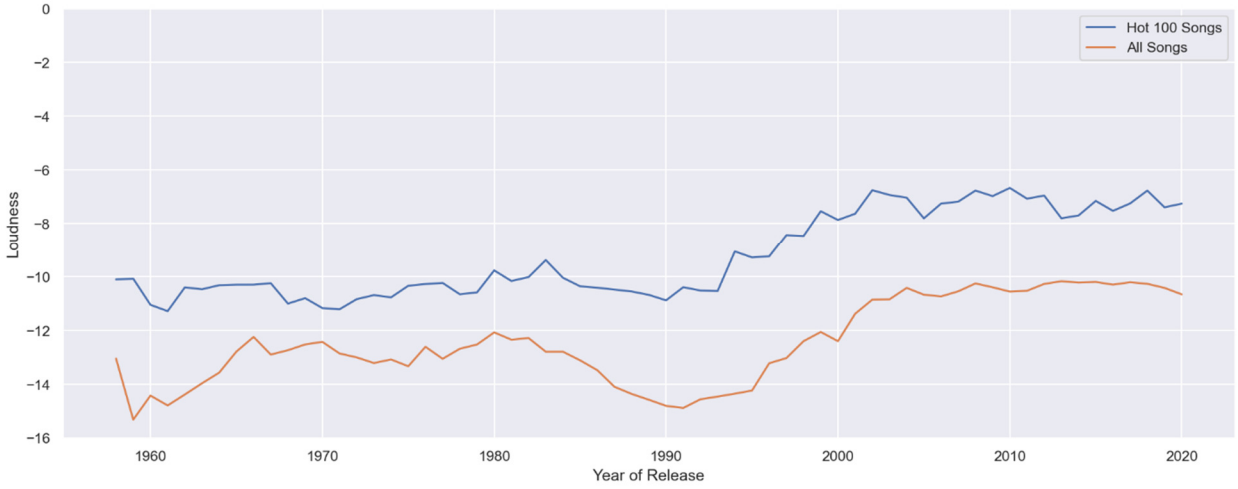


Figure 20. Loudness History Comparing the Billboard Hot 100 with All Songs

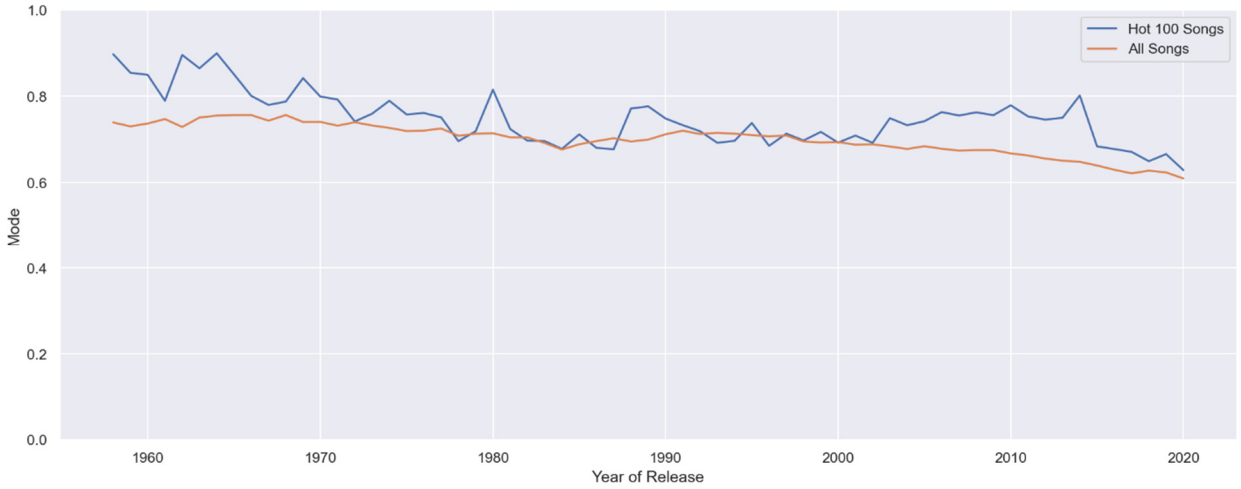


Figure 21. Mode History Comparing the Billboard Hot 100 with All Songs

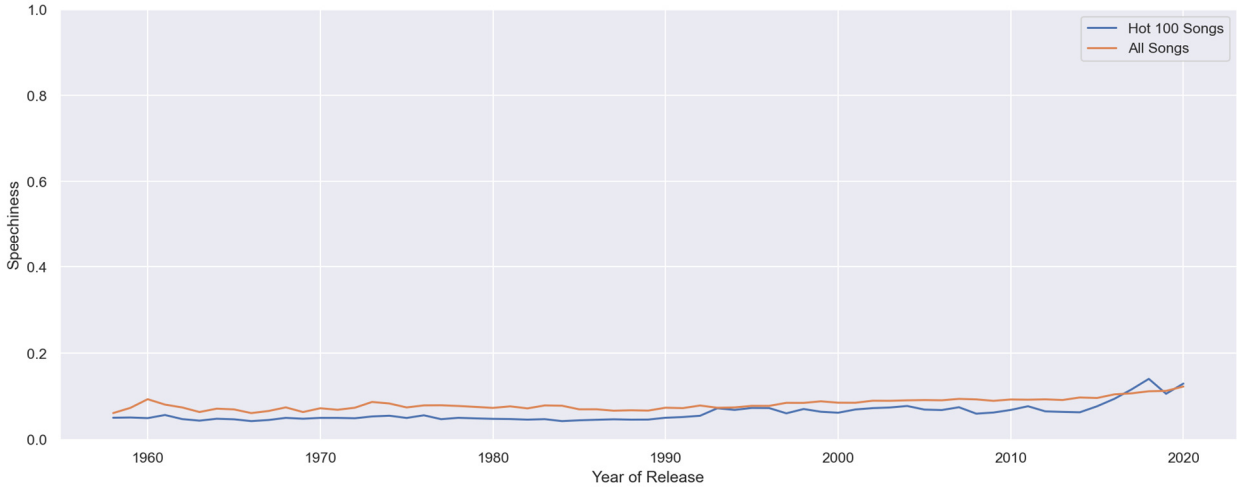


Figure 22. Speechiness History Comparing the Billboard Hot 100 with All Songs

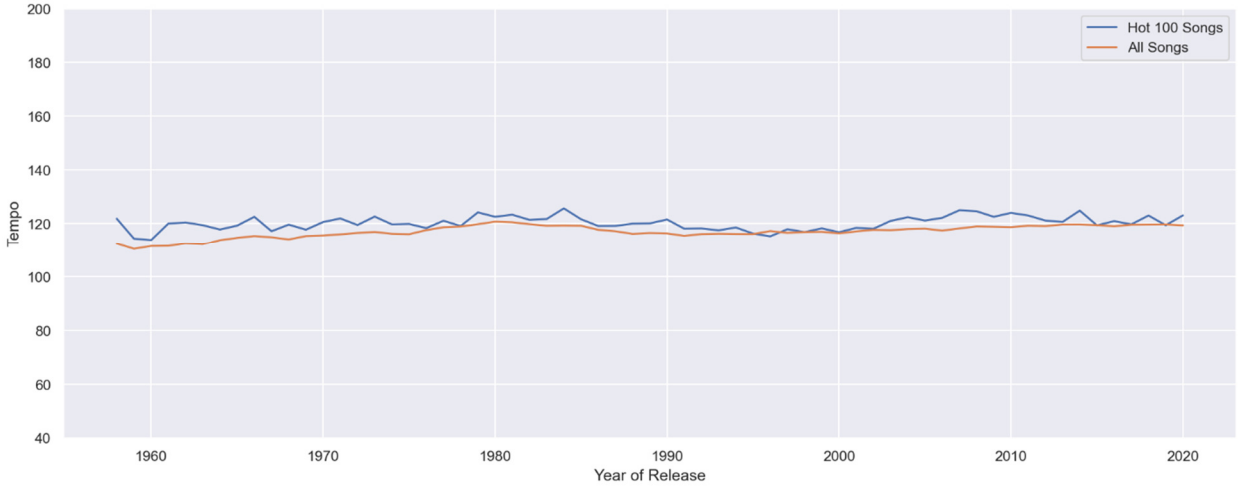


Figure 23. Tempo History Comparing the Billboard Hot 100 with All Songs

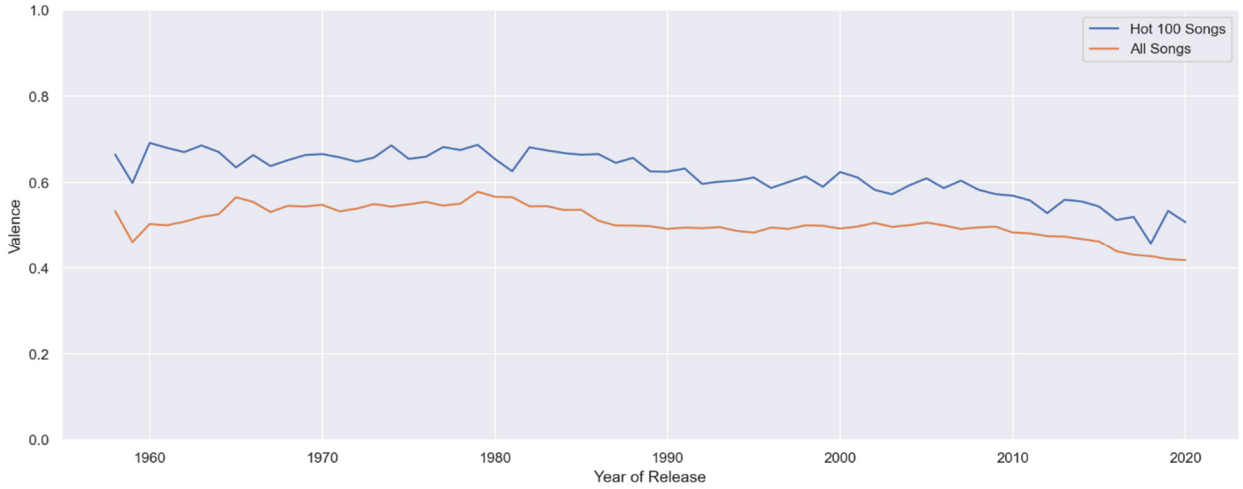


Figure 24. Valence History Comparing the Billboard Hot 100 with All Songs

The following figure shows the Billboard Hot 100 chart yearly audio feature averages over time. In contrast to the above figures, this figure is grouped by appearance on the charts in a given year, and does not necessarily correspond to the release year for any given song. Also of note, all audio features were normalised to range from 0 to 1 in order to create a consistent plot.

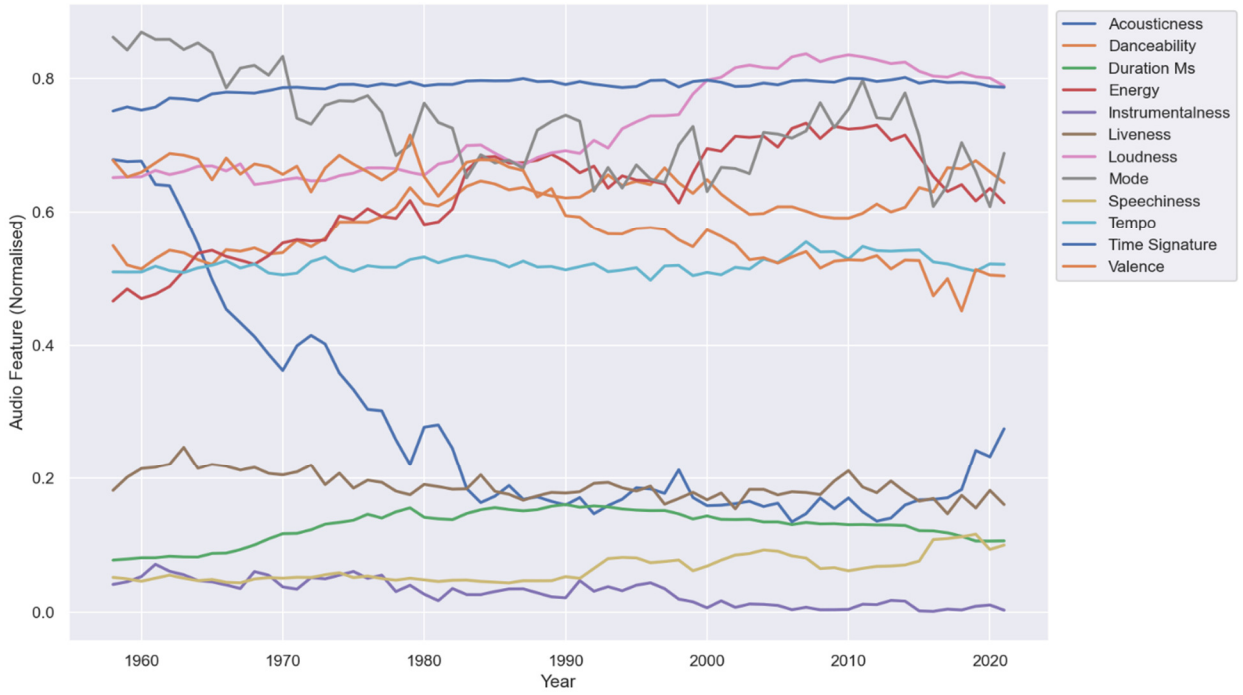


Figure 25. Billboard Hot 100 Historical Charts

A more detailed analysis of each audio feature over time is included in **Attachment 3**.

Correlation Analysis

Correlation analysis is outlined in the below figure. A more detailed correlation analysis in included in **Attachment 3**. The analysis investigated which audio features correlate with each other, differences between Billboard Hot 100 songs and songs not on the list, as well as which features correlate with popularity (as defined above).

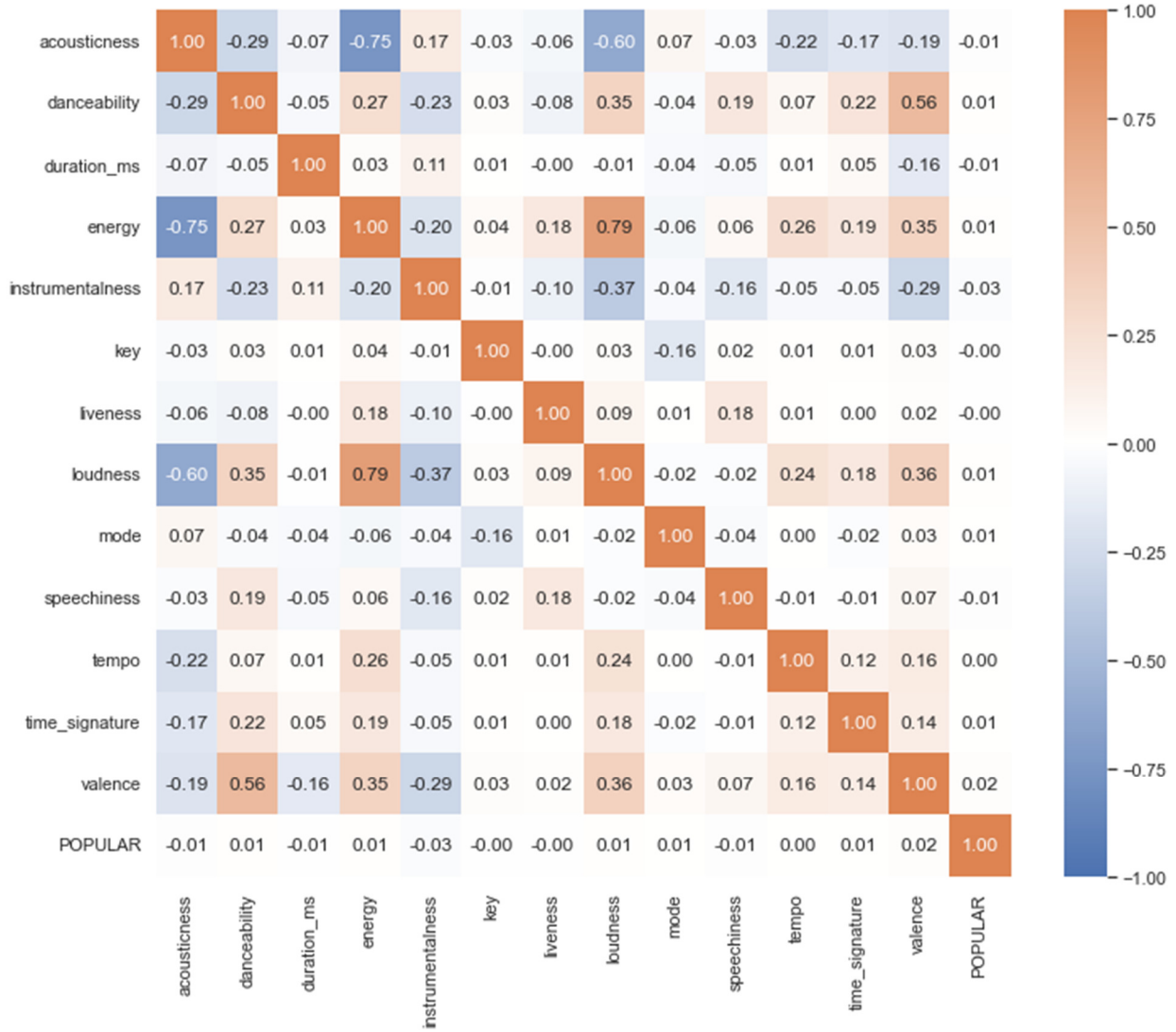


Figure 26. Correlation Analysis Summary

In the above figure, the "POPULAR" feature denotes whether or not the song appeared on the Billboard Hot 100 charts. With a few exceptions, the analysis showed a relatively weak correlation between audio features. This is unsurprising because audio features have been created to represent music simply, accurately, and without redundancies. In later stages of this study, these low correlation values imply that all audio features have the potential to be relevant for predictive analytics, and none of these features can simply be dropped as redundant.

Also of note, every audio feature correlated very weakly with popularity. As noted in the literature review, this is likely due to large varieties of musical styles all being included in the calculations. Separating analysis into genres yielded stronger correlations for some of the audio features, although correlations were still relatively weak. Genres are discussed in the next section, and detailed analysis is included in **Attachment 3**.

Correlation analysis was also conducted strictly on the Billboard Hot 100 dataset. The below figure shows the correlation between audio features and Billboard Hot 100 performance, both peak-rank and weeks-on-board.

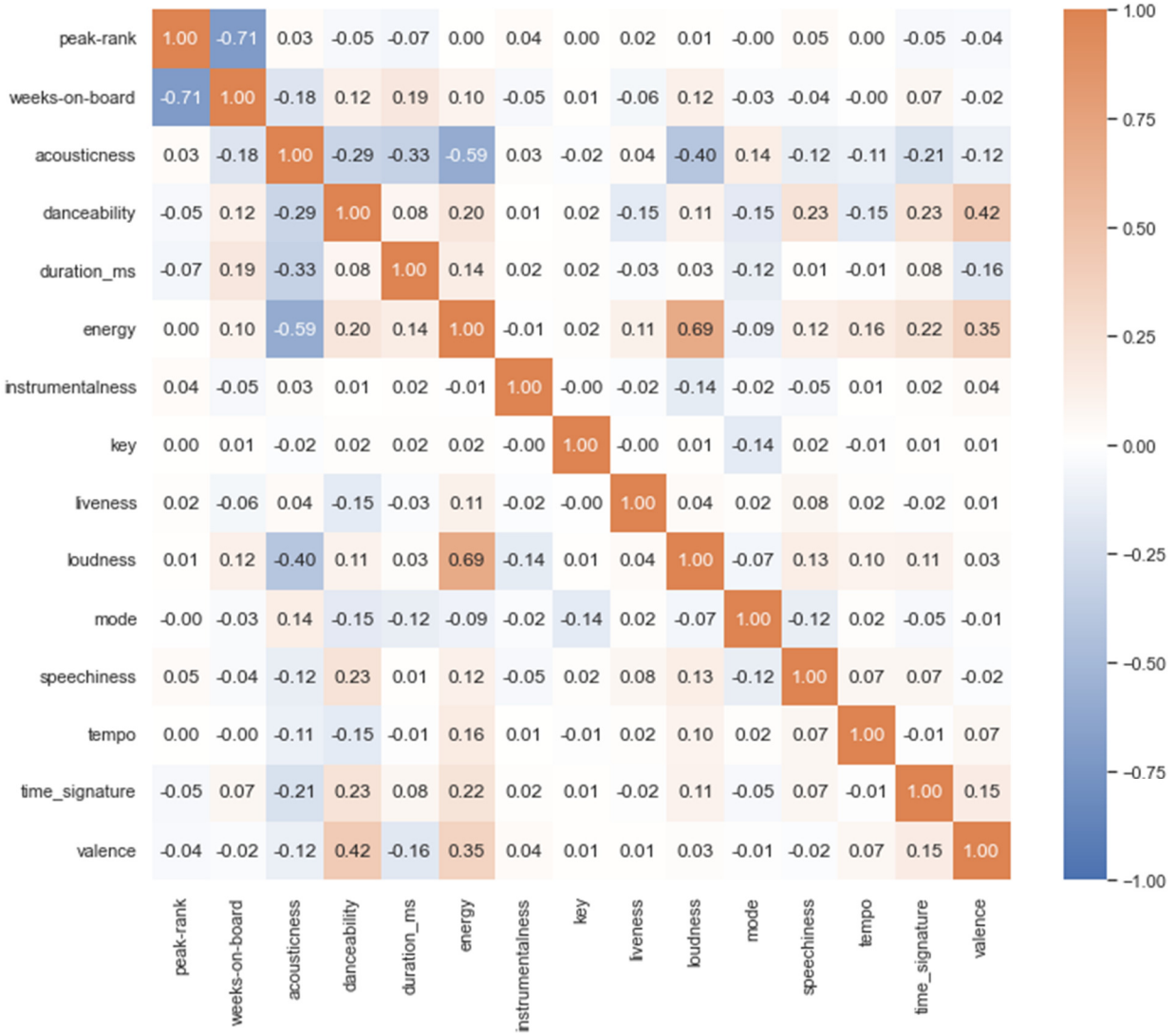


Figure 27. Billboard Hot 100 Correlation Analysis Summary

As shown in the above figure, a number of audio features have a small correlation with performance on the Billboard Hot 100. The following figures show the sorted correlation coefficients corresponding to weeks-on-board and peak-rank, respectively. Note that the order of the charts is reversed because lower peak rank is optimal, whereas higher weeks-on-board is optimal. More detailed analysis is included in **Attachment 3**.

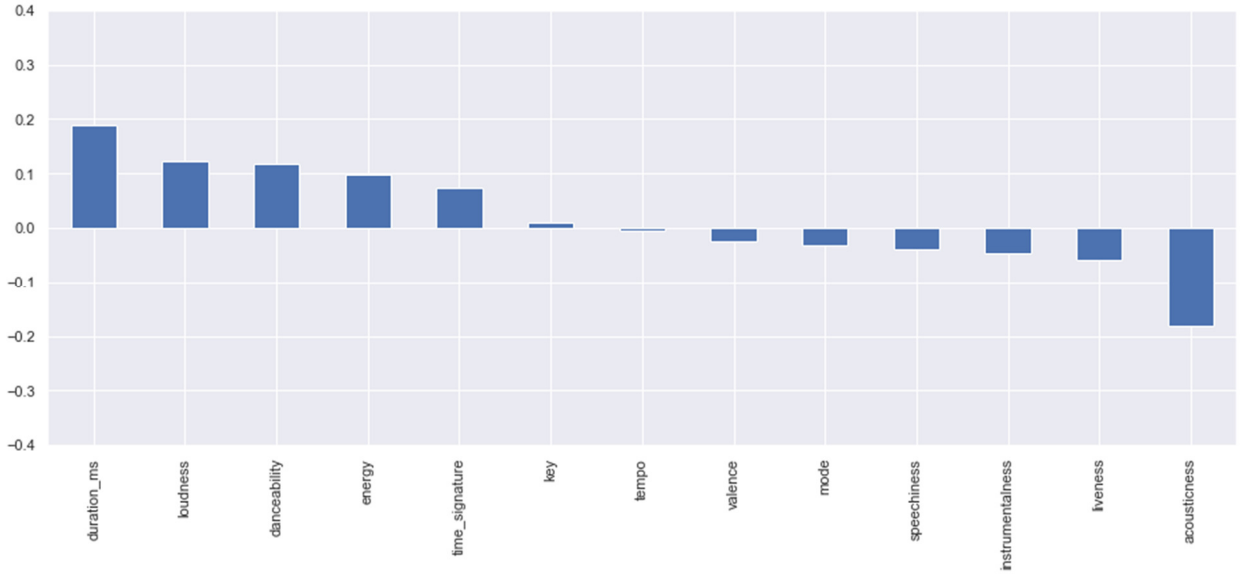


Figure 28. Ranked Audio Feature Correlations With Weeks On Billboard Hot 100 Charts

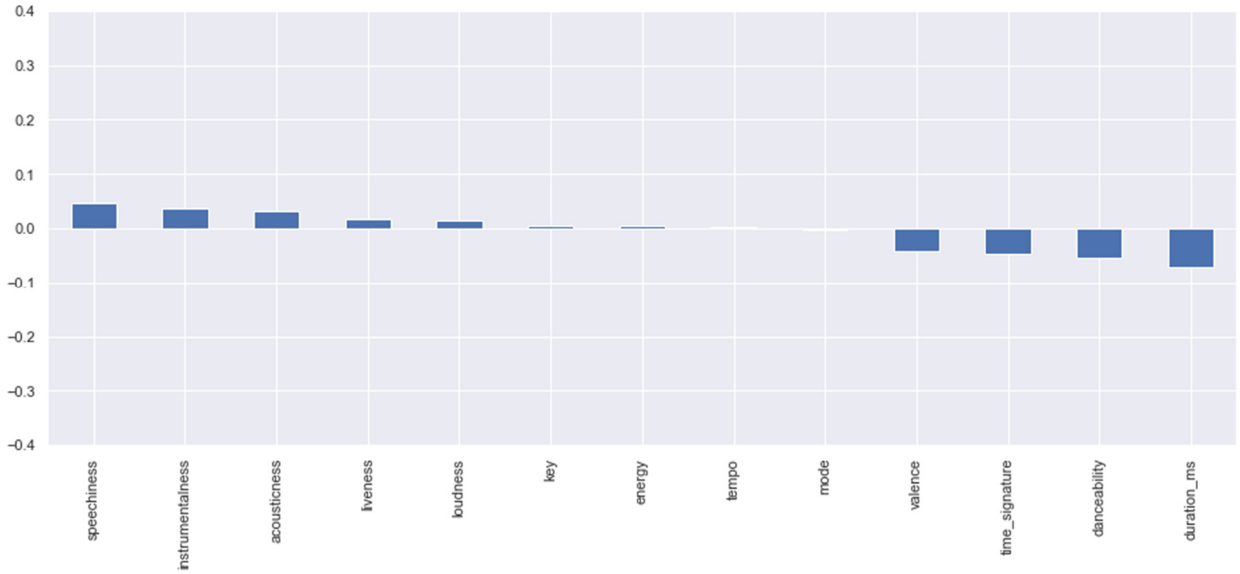


Figure 29. Ranked Audio Feature Correlations With Peak Rank On Billboard Hot 100

Analysis of Genres

In order to improve correlations and future predictions, genre information has been investigated. Correlation analysis showed stronger correlation between audio features and popularity when restricting analysis to specific genres. Detailed genre correlation analysis is included in **Attachment 3**.

Even after excluding all but the most popular genre for each artist, there are still over 5,000 genres in the dataset. Many genres are obscure, and most could more accurately be referred to as sub-genres. For this reason, bins of genres were grouped together to approximate more general genres. Regular Expressions were used to group songs into categories by matching patterns within genre info from the Spotify API. These genre groups were able to reduce variation in audio features to an extent, as shown in the below figure. Details of this genre analysis are included in **Attachment 3**.

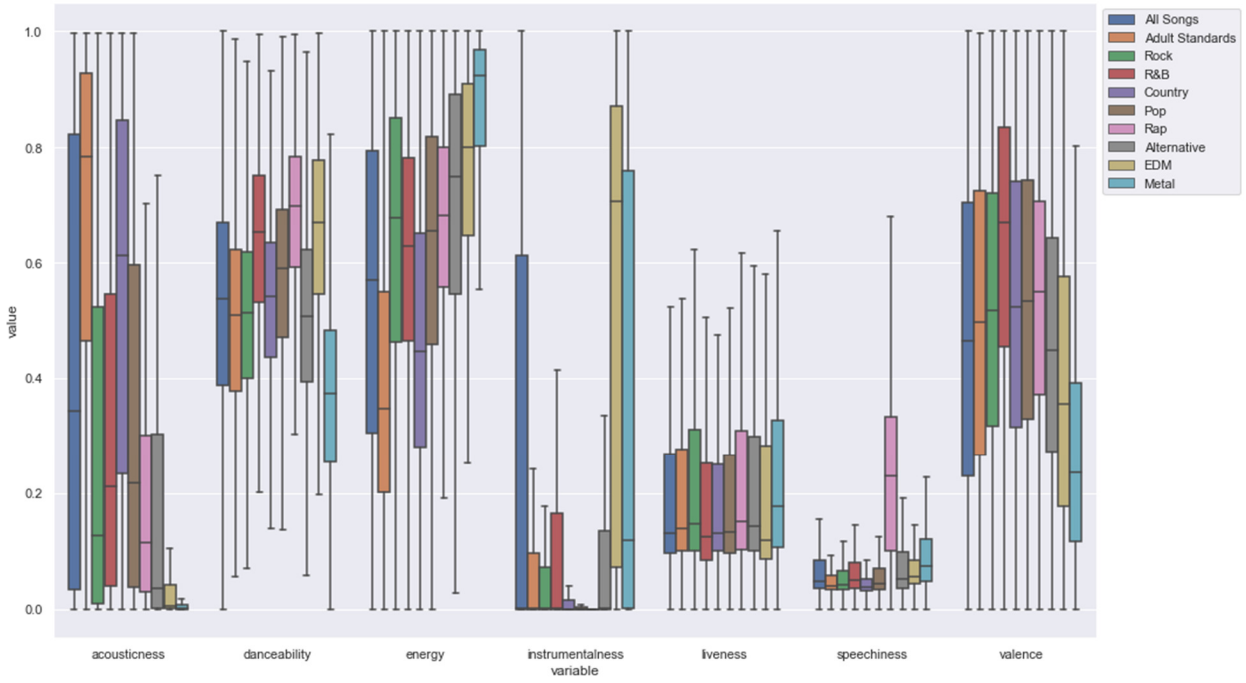


Figure 30. Comparison of Audio Feature Variations Between Genres

It is important to note that these categories are based on assumptions about which genres belong together, and some of these grouping decisions have been made arbitrarily. Future phases on this study will investigate whether clustering analysis will create more useful clusters of songs by avoiding the need to label songs using genre.

Project Approach

The central problem in this project is to utilise publicly available audio feature data to predict the popularity of a song. For this project, two main techniques will be employed, namely data mining and predictive analytics.

Data Mining

First, data mining and knowledge discovery will be used to explore the data, cluster audio features, and determine correlations between audio features.

Genres and audio feature correlation have been explored in detail; details are included in the **Data Description** section above, and detailed calculations are included in **Attachment 3**.

In future phases of this study, this investigation will go more in depth and compare segmentation by genre to clustering performed using machine learning techniques. Ultimately, these clusters will be used in the predictive analytics described below.

For this analysis unsupervised machine learning techniques will employed. K-Means Clustering will most likely be employed, as it was the most common clustering algorithm noted in the Literature Review. Other machine learning models will be considered as the study progresses.

Predictive Analytics

Secondly, predictive analytics will be used to attempt to build a predictive model using the data. We will attempt to predict whether or not a song will be make it onto the Billboard Hot 100 charts using audio features for that song. This prediction will likely incorporated the clusters

from the previous phase of the study, as the correlation between audio features and popularity are weak, as described above in the correlation analysis.

This analysis will investigate a variety of supervised classification algorithms, as well as ensemble methods. Based on the literature review, the most common algorithms for predicting popularity in music are Neural Networks, Support Vector Machines, and K-Nearest Neighbours, as well as some simpler models such as Logistic Regression. Statistical significance and predictive power will be compared between the utilised algorithms.

Outline of Methodology

The overall project methodology and timeline is outlined in the following chart.

Task	week 1	week 2	week 3	week 4	week 5	week 6	week 7	week 8	week 9	week 10	week 11	week 12	week 13
Choose a Theme			Due										
Choose a Dataset			Due										
Write an Abstract			Due										
Literature Review							Due						
Correlation Analysis							Due						
Descriptive Statistics							Due						
Data Description							Due						
Project Approach							Due						
Cluster Analysis											Due		
Predictive Analysis											Due		
Model Validation											Due		
Statistical Testing											Due		
Project Limitations													Due
Final Report													Due
Project Presentation													Due

Figure 31. Project Methodology Timeline

References

Araujo, C. V. S., Cristo, M. A. P., & Giusti, R. (2007). Predicting Music Popularity on Streaming Platforms. ANAIS DO SIMPÓSIO BRASILEIRO DE COMPUTAÇÃO MUSICAL (SBCM 2019).

Billboard Hot 100. (2022, September 6). Wikipedia. https://en.wikipedia.org/w/index.php?title=Billboard_Hot_100&oldid=1108834581

Cataltepe, Z., Yaslan, Y., & Sonmez, A. (2007). Music Genre Classification Using MIDI and Audio Features. EURASIP JOURNAL ON ADVANCES IN SIGNAL PROCESSING.

Chen, Y. C., Chen, Z. C., & Hsia, C. H. (2021). Music Mood Classification System for Streaming Platform Analysis Via Deep Learning Based Feature Extraction. 2021 IEEE INTERNATIONAL CONFERENCE ON CONSUMER ELECTRONICS-TAIWAN (ICCE-TW).

Cilibrasi, R. L., Vitányi, P., & Wolf, R. D. (2004). Algorithmic clustering of music. Proceedings of the Fourth International Conference onWeb Delivering of Music, 2004. EDELMUSIC 2004..

Dhruvil Dave. (2021, November 9). Billboard "The Hot 100" Songs [Data set]. Kaggle. https://doi.org/10.34740/KAGGLE/DS/1211465

Elicit. (n.d.). https://elicit.org/

Febirautami, L. R., Surjandari, I., & Laoh, E. (2018). Determining Characteristics of Popular Local Songs in Indonesia's Music Market. 2018 5TH INTERNATIONAL CONFERENCE ON INFORMATION SCIENCE AND CONTROL ENGINEERING (ICISCE).

Gao, A. (2021). Catching the Earworm: Understanding Streaming Music Popularity Using Machine Learning Models. E3S Web of Conferences 253, 03024

Google Dataset Search. (n.d.). https://datasetsearch.research.google.com/

Google Scholar. (n.d.). https://scholar.google.com/

Honingh, A. K., & Bod, R. (2011). Clustering and Classification of Music by Interval Categories. MCM.

Huo, Y. (2021). Music Personalized Label Clustering and Recommendation Visualization. Complex..

Jia, X. (2022). A Music Emotion Classification Model Based on The Improved Convolutional Neural Network. COMPUTATIONAL INTELLIGENCE AND NEUROSCIENCE.

Kim, J. H. (2021). Music Popularity Prediction Through Data analysis of Music's Characteristics. International Journal of Science, Technology and Society.

Kim, S., Park, J., Seong, K., Cho, N., Min, J., & Hong, H. (2021). Music-Circles: Can Music Be Represented With Numbers?. ARXIV.

Laurier, C. , Lartillot, O. , Eerola, T. , & Toiviainen, P. (2009). Exploring relationships between audio features and emotion in music.

Lee, J. , & Lee, J. S. (2018). Music Popularity: Metrics, Characteristics, and Audio-Based Prediction. IEEE Transactions on Multimedia.

Li, L. (2021). Learning Recommendation Algorithm Based on Improved BP Neural Network in Music Marketing Strategy. COMPUTATIONAL INTELLIGENCE AND NEUROSCIENCE.

Li, Q., Kim, B. M., Guan, D. H., & Oh, D. W. (2004). A Music Recommender Based On Audio Features. SIGIR.

Li, Q., Myaeng, S. H., & Kim, B. M. (2007). A Probabilistic Music Recommender Considering User Opinions and Audio Features. INF. PROCESS. MANAG..

Li, X., & Li, J (2022). Music Classification Method Using Big Data Feature Extraction and Neural Networks. JOURNAL OF ENVIRONMENTAL AND PUBLIC HEALTH.

Malte Grosse. (2022, March 23). 8+ M. Spotify Tracks, Genre, Audio Features [Data set]. Kaggle. https://www.kaggle.com/datasets/maltegrosse/8-m-spotify-tracks-genre-audio-features/

Martín-Gutiérrez, D. , Peñaloza, G. H., Belmonte-Hernández, A. , & García, F Á. (2020). A Multimodal End-to-End Deep Learning Architecture for Music Popularity Prediction. IEEE ACCESS.

O'Toole, K., & Horvát, E. Á. (2022). Novelty and Cultural Evolution in Modern Popular Music. ARXIV.

Paper Digest. (n.d.). https://www.paperdigest.org/

Reiman, M., & Örnell, P. (2018). Predicting Hit Songs with Machine Learning. EXAMENSARBETE INOM TEKNIK, GRUNDNIVÅ, 15 HP.

Rodolfo Figueroa. (2020, December 22). Spotify 1.2M+ Songs [Data set]. Kaggle. https://www.kaggle.com/datasets/rodolfofigueroa/spotify-12m-songs

Schedl, M. (2013). Ameliorating Music Recommendation: Integrating Music Content, Music Context, and User Context for Improved Music Retrieval and Recommendation.

Setiadi, D. R. I. M., Rahardwika, D. S. , Rachmawanto, E. H. , Sari, C. A. , Susanto, A., Mulyono, I. U. W. , Astuti, E. Z. , & Fahmi, A. (2020). Effect of Feature Selection on The Accuracy of Music Genre Classification Using SVM Classifier. 2020 INTERNATIONAL SEMINAR ON APPLICATION FOR TECHNOLOGY OF INFORMATION AND COMMUNICATION (ISEMANTIC).

Shi, J. (2021). Music Recommendation Algorithm Based on Multidimensional Time-Series Model Analysis. COMPLEX..

Spotify for Developers. (n.d.). https://developer.spotify.com/documentation/web-api/

West, K. (2008). Novel Techniques for Audio Music Classification and Search. ACM SIGMULTIMEDIA RECORDS.

Wilkes, B., Vatolkin, I., & Müller, H. (2021). Statistical and Visual Analysis of Audio, Text, and Image Features for Multi-Modal Music Genre Recognition. ENTROPY (BASEL, SWITZERLAND).

Xu, Y., & Xu, S. (2021). A Clustering Analysis Method for Massive Music Data.

Yang, L. C., Chou, S. Y., Liu, J. Y., Yang, Y. H., & Chen, Y. (2017). Revisiting the problem of audio-based hit song prediction using convolutional neural networks. 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).

Attachments

Attachment 1

Import Modules

In [2]:
```
# import modules
import pandas as pd
import numpy as np
import spotipy

# jupyter notebook full-width display
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))

# pandas formatting
pd.set_option('display.float_format', '{:_.2f}'.format)
# NOTE: underscore separaters ('_') are better than commas (',') because
# numbers with underscores work in Python without any extra effort.
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 200)
```

Import Starting Data

The Billboard 100

https://www.kaggle.com/datasets/dhruvildave/billboard-the-hot-100-songs

In [ ]:
```
# Billboard Top 100 Historical Data
# via:  https://toolbox.google.com/datasetsearch
url_billboard = r'D:\RYERSON\820\Datasets\Billboard The Hot 100 Songs\charts.csv'

df_billboard = pd.read_csv(url_billboard)
df_billboard['date'] = pd.to_datetime(df_billboard['date'])

# Unique Songs from The Billboard 100 Dataset

# just the songs on the billboard 100, once per song
df_billboard_songs = df_billboard[['song', 'artist']].drop_duplicates().sort_values(['artist', 'song']).reset_index(drop=True)

# add a blank id column and a blank MISSING column
df_billboard_songs['id'] = ''
df_billboard_songs['MISSING'] = ''

df_billboard.shape, df_billboard_songs.shape
```

1.2M Songs with Metadata (csv)

https://www.kaggle.com/datasets/rodolfofigueroa/spotify-12m-songs

In [ ]:
```
# Spotify 1.2M+ Songs
# via:  https://www.kaggle.com/datasets/
url_1M_songs = r'D:\RYERSON\820\Datasets\Spotify 1.2M+ Songs\tracks_features.csv'

# create the dataframe with the large number of songs metadata
df_1M_songs = pd.read_csv(url_1M_songs)

# make a list of song ids from the 1M dataset
metadata_ids_csv = df_1M_songs.id.to_list()
```

8.7M Songs with Metadata (SQL)

In [ ]:
```
# all ids from the SQLite database
metadata_ids_SQL = pd.read_csv('all_ids_sql.csv', header=None, names=['id'])
metadata_ids_SQL = metadata_ids_SQL.id.to_list()

# audio feature data not imported yet (very large)
```

Combine Ids For Datasets with Metadata

In [ ]:
```
# list of ids for all of our known metadata
all_metadata_ids = set(metadata_ids_csv + metadata_ids_SQL) # set() faster to search, and no duplicates
len(all_metadata_ids)
```

Get Track IDs using API

TEMPORARY TOKEN WORKFLOW

get a temporary authorization token from: https://developer.spotify.com/console/get-search-item

In [ ]:
```
# input the temporary token
TEMP_TOKEN = input('Enter token: ')

# create a spotify object
spotify = spotipy.Spotify(auth=TEMP_TOKEN)
```

In [ ]:
```
# helper function
def find_id(track_title, artist_name, metadata_ids):
    """
    for searches with multiple results, all id were identical for the test cases I ran
    some searches return no results, in this case the song is not on spotify
        confirmed by spot checks in the spotify music player
    some tracks give a 404 error,
        these seem to exist in Spotify but 404 anyway
        not sure why the API does this but
    """
    track_info = spotify.search(q='artist:' + artist_name + ' track:' + track_title, type='track')

    if track_info['tracks']['items'] == []:  # if track doesn't exist on Spotify
        return '', 'MISSING'
    else:
        # default to 0th id
        track_id = track_info['tracks']['items'][0]['id']

        number_of_results = len(track_info['tracks']['items'])

        # check if there is a better match
        for i in range(number_of_results):
            current_id = track_info['tracks']['items'][i]['id']
            if current_id in metadata_ids:
                return current_id, 'matched'  # immediately return it if it's found

        # if we made it through the loop without returning, note 'MISSING' and return the 0th id
        return track_id, 'MISSING'
```

In [ ]:
```
%%time
# TEST
find_id("You Can't Turn Me Off (In The Middle Of Turning Me On)", 'High Inergy', all_metadata_ids)
```

In [ ]:
```
# TEST
find_id("You Can't Find this (SONG)", 'Low Unergy', all_metadata_ids)
```

Add Spotify IDs to billboard songs matched in the datasets

In [ ]:
```
# load saved csv if required
df_billboard_songs = pd.read_csv('df_billboard_songs.csv', keep_default_na=False)
df_billboard_songs.id.nunique()
```

In [ ]:
```
# start over at
start_over_at = 30000

# populate df_billboard_songs with ids, where available
for i, row in df_billboard_songs.iterrows():

    # start over at
    if i < start_over_at-1:
        continue

    # show status update
    if i%10 == 0:
        print(i, end='  ')
    if i%100 == 0:
        print()

    # start over where we finished (don't overwrite known ids)
    if df_billboard_songs['id'].iloc[i] != '':
        continue
    # append id, NONE, or 'ERROR'
    else:
        artist = row[1]
        song = row[0]
```

```
        try:
            # unless there is an error, will append the id or None
            df_billboard_songs['id'].iloc[i], df_billboard_songs['MISSING'].iloc[i] = find_id(song, artist, all_metadata_ids)
        except:  # all errors treated the same
            # if there is an error, change id to 'ERROR'
            print('ERROR:  ', artist, song)
            df_billboard_songs['MISSING'].iloc[i] = 'ERROR'  # leave id blank

        # save every 1000 rows, if new
        if i%1000 == 0:
            df_billboard_songs.to_csv('df_billboard_songs_TEMP.csv', index=False)

# save final dataframe
df_billboard_songs.to_csv('df_billboard_songs.csv', index=False)
```

After Gathering all IDs

In [ ]:
```
# reload df_billboard_songs if required
df_billboard_songs = pd.read_csv('df_billboard_songs.csv', keep_default_na=False)
```

In [ ]:
```
# how many id have we added
df_billboard_songs.id.nunique(), sum(df_billboard_songs.id != "")
# 83 duplicated ids
```

Remove Duplicates / Errors

In [ ]:
```
duplicates = df_billboard_songs[df_billboard_songs.id!='']
duplicates = duplicates[duplicates.id.duplicated(False)]
duplicates.to_csv('duplicated_ids.csv', index=True)
```

In [ ]:
```
# set of duplicated ids
duplicated_ids = set(duplicates.id)
```

In [ ]:
```
sum(df_billboard_songs['id'].isin(duplicated_ids))
```

In [ ]:
```
# drop from billboard list of known ids
for i, row in df_billboard_songs.iterrows():
    if df_billboard_songs.iloc[i]['id'] in duplicated_ids:
        df_billboard_songs['MISSING'].iloc[i] = 'DUPLICATED'
        df_billboard_songs['id'].iloc[i] = ''
```

In [ ]:
```
sum(df_billboard_songs['id'].isin(duplicated_ids))
```

In [ ]:
```
df_billboard_songs.to_csv('df_billboard_songs - duplicates removed.csv', index=False)
```

what songs are still missing audio feature data?

In [ ]:
```
# reload the dataframe if required
df_billboard_songs = pd.read_csv('df_billboard_songs - duplicates removed.csv', keep_default_na=False)
```

In [ ]:
```
# how many id have we added
df_billboard_songs.id.nunique(), sum(df_billboard_songs.id != '')
# off by one because '' counts as a unique id
```

In [ ]:
```
# check how many id match the metadata_ids
sum(df_billboard_songs.id.isin(all_metadata_ids))
```

In [ ]:
```
# songs on spotify that we don't have audio features for yet
need_audio_features = df_billboard_songs[(~df_billboard_songs.id.isin(all_metadata_ids) & (df_billboard_songs.id != ''))]
need_audio_features.to_csv('need_audio_features.csv', index=False)
```

In [ ]:
```
len(need_audio_features.id)
```

Use API again to get missing audio features

https://developer.spotify.com/console/get-audio-features-track/

In [ ]:
```
# input the temporary token
TEMP_TOKEN = input('Enter token: ')

# create a spotify object
spotify = spotipy.Spotify(auth=TEMP_TOKEN)
```

In [ ]:
```
# initialise dataframe

with_audio_features = need_audio_features.copy().reset_index()

list_of_features = [
    'danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness',
    'instrumentalness', 'liveness', 'valence', 'tempo', 'duration_ms', 'time_signature'
]

for feature in list_of_features:
    with_audio_features[feature] = ''

with_audio_features.head()
```

In [ ]:
```
start_over_at = 30000

list_of_features = [
    'danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness',
    'instrumentalness', 'liveness', 'valence', 'tempo', 'duration_ms', 'time_signature'
]

for i, row in with_audio_features.iterrows():

    # start over at
    if i < start_over_at-1:
        continue

    # show status update
    if i%10 == 0:
        print(i, end='  ')
    if i%100 == 0:
        print()

    track_id = with_audio_features['id'].iloc[i]
    temp_audio_features = spotify.audio_features(track_id)

    for key in list_of_features:
        with_audio_features[key].iloc[i] = temp_audio_features[0][key]

    if i%100 == 0:
        with_audio_features.to_csv('audio_features_TEMP.csv', index=True)

# save final df
with_audio_features.to_csv('audio_features_FINAL.csv', index=True)
```

QA SPOTCHECKS (used later to verify data integrity)

In [ ]:
```
QA_DATAFRAME = df_billboard_songs[df_billboard_songs.MISSING == 'matched'].sample(100).reset_index()

list_of_features = [
    'danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness',
    'instrumentalness', 'liveness', 'valence', 'tempo', 'duration_ms', 'time_signature'
]

for feature in list_of_features:
    QA_DATAFRAME[feature] = ''

QA_DATAFRAME.head()
```

In [ ]:
```
start_over_at = 0

list_of_features = [
    'danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness',
    'instrumentalness', 'liveness', 'valence', 'tempo', 'duration_ms', 'time_signature'
]

for i, row in QA_DATAFRAME.iterrows():

    # start over at
    if i < start_over_at-1:
        continue
```

```
    # show status update
    if i%10 == 0:
        print(i, end='  ')
    if i%100 == 0:
        print()

    track_id = QA_DATAFRAME['id'].iloc[i]
    temp_audio_features = spotify.audio_features(track_id)

    for key in list_of_features:
        QA_DATAFRAME[key].iloc[i] = temp_audio_features[0][key]

# save final df
QA_DATAFRAME.to_csv('QA_DATAFRAME.csv', index=True)
```

Finalize Dataset

Reimport data from Billboard 100 and 1.2M Songs Dataset

In [2]:

```
# billboard songs
df_billboard_songs = pd.read_csv('df_billboard_songs - duplicates removed.csv', keep_default_na=False)

# billboard songs with audio features from API (missing from other datasets)
df_api_features = pd.read_csv('audio_features_FINAL.csv', keep_default_na=False)
df_api_features = df_api_features[[
    'id', 'song', 'artist',
    'acousticness', 'danceability', 'duration_ms', 'energy', 'instrumentalness',
    'key', 'liveness', 'loudness', 'mode', 'speechiness', 'tempo', 'time_signature', 'valence'
]]

# billboard time series
url_billboard = r'D:\RYERSON\820\Datasets\Billboard The Hot 100 Songs\charts.csv'
df_billboard = pd.read_csv(url_billboard)
df_billboard['date'] = pd.to_datetime(df_billboard['date'])

# 1.2M songs with metadata
url_1M_songs = r'D:\RYERSON\820\Datasets\Spotify 1.2M+ Songs\tracks_features.csv'
df_1M_songs = pd.read_csv(url_1M_songs)
df_1M_songs.rename(columns={'name': 'song', 'artists': 'artist', 'duration': 'duration_ms'}, inplace=True)
df_1M_songs = df_1M_songs[[
    'id', 'song', 'artist',
    'acousticness', 'danceability', 'duration_ms', 'energy', 'instrumentalness',
    'key', 'liveness', 'loudness', 'mode', 'speechiness', 'tempo', 'time_signature', 'valence'
]]
```

8+ M. Spotify Tracks, Genre, Audio Features (SQLite)

https://www.kaggle.com/datasets/maltegrosse/8-m-spotify-tracks-genre-audio-features

In [3]:

```
url_8M_sql = 'D:\RYERSON\820\Datasets\8+ M. Spotify Tracks, Genre, Audio Features\spotify.sqlite'
url_8M_csv = 'all_audio_features_sql.csv'  # .gitignore (very big)

df_8M_songs = pd.read_csv(url_8M_csv, on_bad_lines='skip')
df_8M_songs.rename(columns={'name': 'artist', 'name:1': 'song', 'duration': 'duration_ms'}, inplace=True)
# already in alphabetical order

df_8M_songs.drop_duplicates(inplace=True)
```

Merge Billboard 100 Songs with Metadata

In [4]:

```
# audio features
list_of_features = [
    'acousticness', 'danceability', 'duration_ms', 'energy', 'instrumentalness',
    'key', 'liveness', 'loudness', 'mode', 'speechiness', 'tempo', 'time_signature', 'valence'
]

# songs in the billboard 100 list
set_billboard = set(df_billboard_songs.id)
set_billboard.discard('')

# df_api_features does not need to be filtered or sorted

# 8M songs from SQL
df1 = df_8M_songs[df_8M_songs.id.isin(set_billboard)]
# 930 duplicate ids, all appear to be collaborations (based on Google spotcheck ~ 10 songs)
# delete duplicates, keep first, should work fine, but secondary artist may be listed first
df1 = df1.drop_duplicates(subset='id', keep='first')
```

```
set1 = set_billboard - set(df1.id)

# 1M songs from SQL
df2 = df_1M_songs[df_1M_songs.id.isin(set1)]
set2 = set1 - set(df2.id) - set(df_api_features.id)
set2.add('')

# missing songs get np.nan
df3a = df_billboard_songs[df_billboard_songs.id.isin(set2)].reset_index(drop=True)
df3b = pd.DataFrame(data=np.nan, index=[x for x in range(df3a.shape[0])], columns=list_of_features)
df3 = pd.concat([df3a, df3b], axis=1)
df3 = df3[[
    'id', 'song', 'artist',
    'acousticness', 'danceability', 'duration_ms', 'energy', 'instrumentalness',
    'key', 'liveness', 'loudness', 'mode', 'speechiness', 'tempo', 'time_signature', 'valence'
]]

# merge all into total dataframe
all_audio_features = pd.concat([df_api_features, df1, df2, df3]).reset_index(drop=True)

# 3 duplicated ids, look like errors (closely named tracks)
duplicate_errors = set(all_audio_features[(all_audio_features.id != '') & (all_audio_features.id.duplicated(keep=False))].id)
# output for reference:  {'4kqOuBMioKeLNLkPpmxduf', '4oR2cCQGs0Yt0Mgr2diV6V', '5FVbvttjEvQ8r2BgUcJgNg'}

set_nan = ['id', 'acousticness', 'danceability', 'duration_ms', 'energy', 'instrumentalness',
    'key', 'liveness', 'loudness', 'mode', 'speechiness', 'tempo', 'time_signature', 'valence'
]
all_audio_features = all_audio_features[~all_audio_features.id.isin(duplicate_errors)].sort_values(['artist', 'song']).reset_index(drop=True)
```

In [ ]:

```
# replace song and artist with the exact text from df_billboard_songs

billboard_ids = set(df_billboard_songs.id)
billboard_ids.discard('')
billboard_ids = list(billboard_ids)  # faster to iterate over list

counter = 0

for idx in billboard_ids:

    # show status update
    counter += 1
    if counter%10 == 0:
        print(counter, end='  ')
    if counter%100 == 0:
        print()

    all_audio_features.loc[all_audio_features.id==idx, 'song'] = df_billboard_songs.loc[df_billboard_songs.id==idx, 'song'].values[0]
    all_audio_features.loc[all_audio_features.id==idx, 'artist'] = df_billboard_songs.loc[df_billboard_songs.id==idx, 'artist'].values[0]
```

In [10]:

```
# re-sort, re-index, and save the dataframe to csv
# some of the tracks re-sort because the names contain odd characters
# eg,  Milord   Ã‰dith Piaf should be Milord    Edith Piaf
# eg2, Safaera  Ã‘engo Flow should be Safaera   Bad Bunny, Jowell & Randy & Nengo Flow

all_audio_features = all_audio_features.sort_values(['artist', 'song']).reset_index(drop=True)
all_audio_features.to_csv('all_audio_features_billboard_100_songs.csv', index=False)
```

Merge Billboard 100 Timeseries with Songs + Metadata

In [21]:

```
# reload merged song dataset
all_audio_features = pd.read_csv('all_audio_features_billboard_100_songs.csv')

# merge with all_audio_features
# use ids from df_billboard_songs or match(song, artist)
all_audio_features_billboard_100 = pd.merge(df_billboard, all_audio_features, on=['song', 'artist'])
all_audio_features_billboard_100 = all_audio_features_billboard_100.sort_values(['date', 'artist', 'song']).reset_index(drop=True)

all_audio_features_billboard_100.to_csv('all_audio_features_billboard_100.csv')
```

In [22]:

```
all_audio_features_billboard_100.head(20)
```

Out[22]:

	date	rank	song	artist	last-week	peak-rank	weeks-on-board	id	acousticness	danceability	duration_ms	energy	instrumentalness	key	live
0	1958-08-04	31	Chantilly Lace	Big Bopper	NaN	31	1	07GtDOCxmye5KDWsTSACPk	0.84	0.49	145_266.00	0.81	0.00	3.00	

	date	rank	song	artist	last-week	peak-rank	weeks-on-board	id	acousticness	danceability	duration_ms	energy	instrumentalness	key	live
1	1958-08-04	82	Blip Blop	Bill Doggett	NaN	82	1	328wGzwVquTqX5m3t1czL0	0.46	0.70	166_826.00	0.73	0.47	10.00	
2	1958-08-04	99	I'll Get By (As Long As I Have You)	Billy Williams	NaN	99	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
3	1958-08-04	3	Splish Splash	Bobby Darin	NaN	3	1	40fD7ct05FvQHLdQTgJelG	0.38	0.64	131_719.00	0.94	0.00	0.00	
4	1958-08-04	60	Over And Over	Bobby Day	NaN	60	1	3ixHQiAUk6F6ZU1tipromq	0.70	0.64	143_320.00	0.55	0.00	0.00	
5	1958-08-04	35	Rock-in Robin	Bobby Day	NaN	35	1	2DAgYTzfPYqKJu0uItsNMd	0.58	0.53	154_627.00	0.79	0.00	7.00	
6	1958-08-04	78	Betty Lou Got A New Pair Of Shoes	Bobby Freeman	NaN	78	1	7h7U3OYppI7HpgFqCc2VZ3	0.22	0.43	149_200.00	0.62	0.00	2.00	
7	1958-08-04	20	Do You Want To Dance	Bobby Freeman	NaN	20	1	4wXPZBafMKbTtdOB7BVGcp	0.52	0.62	165_693.00	0.44	0.00	0.00	
8	1958-08-04	40	Crazy Eyes For You	Bobby Hamilton	NaN	40	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
9	1958-08-04	61	Itchy Twitchy Feeling	Bobby Hendricks	NaN	61	1	36CvDUCDMDbp3dZidKxTds	0.81	0.54	149_200.00	0.71	0.00	2.00	
10	1958-08-04	41	Early In The Morning	Buddy Holly	NaN	41	1	3kKDCw2O8pmEQzhocdkelU	0.72	0.73	126_040.00	0.67	0.00	5.00	
11	1958-08-04	38	Somebody Touched Me	Buddy Knox with the Rhythm Orchids	NaN	38	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
12	1958-08-04	81	Beautiful Delilah	Chuck Berry	NaN	81	1	7nQ757lcDRUlX4F64yXmEg	0.78	0.80	127_600.00	0.67	0.00	1.00	
13	1958-08-04	80	Johnny B. Goode	Chuck Berry	NaN	80	1	2QfiRTz5Yc8DdShCxG1tB2	0.74	0.53	161_560.00	0.80	0.00	10.00	
14	1958-08-04	25	What Am I Living For	Chuck Willis	NaN	25	1	6eIEXINZt8rtgqEtOxP9ur	0.73	0.67	146_800.00	0.39	0.00	10.00	
15	1958-08-04	43	Come What May	Clyde McPhatter	NaN	43	1	18BLPLHPRpxcXUr7rxnHpn	0.84	0.69	101_946.00	0.75	0.00	0.00	
16	1958-08-04	72	Stupid Cupid	Connie Francis	NaN	72	1	0KM3fzzUBVW0vW7RmFh28v	0.83	0.61	134_560.00	0.73	0.00	3.00	
17	1958-08-04	79	The Bird On My Head	David Seville	NaN	79	1	5glQMF4eW0zD77F3dsWscF	0.85	0.56	137_196.00	0.57	0.00	3.00	
18	1958-08-04	30	Angel Baby	Dean Martin	NaN	30	1	5DX3P3b7kv8266XyuA7fsS	0.94	0.60	166_160.00	0.23	0.00	6.00	
19	1958-08-04	48	Return To Me	Dean Martin	NaN	48	1	6YAgmqaeo8Fm3pne8OJEBo	0.97	0.27	144_800.00	0.20	0.00	2.00	

Merge All Songs With Metadata

In [41]:

```
# songs in the billboard 100 list
set_billboard_all = set(all_audio_features.id)
set_billboard_all.discard('')

# 8M songs not in the billboard 100 songs
df1 = df_8M_songs[~df_8M_songs.id.isin(set_billboard_all)]
set1 = set_billboard_all | set(df1.id)

# 1M songs from SQL
df2 = df_1M_songs[~df_1M_songs.id.isin(set1)]

# only billboard songs with known spotify ids
df0 = all_audio_features[~all_audio_features.id.isnull()]
```

```
# concat
every_song_with_data = pd.concat([df0, df1, df2]).reset_index(drop=True)
every_song_with_data = every_song_with_data.drop_duplicates(subset='id', keep='first')
```

In [44]:
```
every_song_with_data.to_csv('every_song_with_data.csv')
```

In [45]:
```
# save
every_song_with_data.shape
```

Out[45]:
```
(9595992, 16)
```

Check How Many Songs Are Accounted For: 75%

In [48]:
```
# check the percentage of songs accounted for
# from songs list:

(
    all_audio_features[all_audio_features.id.notnull()].shape[0],
    all_audio_features.shape[0],
    all_audio_features[all_audio_features.id.notnull()].shape[0] / all_audio_features.shape[0]
)
```

Out[48]:
```
(22189, 29681, 0.7475826286176341)
```

In [47]:
```
# check the percentage of songs accounted for
# from billboard timeseries: all_audio_features_billboard_100

(
    all_audio_features_billboard_100[all_audio_features_billboard_100.id.notnull()].shape[0],
    all_audio_features_billboard_100.shape[0],
    all_audio_features_billboard_100[all_audio_features_billboard_100.id.notnull()].shape[0] / all_audio_features_billboard_100.shape[0]
)
```

Out[47]:
```
(253254, 329930, 0.7675991877064832)
```

QA check vs QA dataframe

In [5]:
```
check_vs = pd.read_csv('all_audio_features_billboard_100_songs.csv')
QA_check = pd.read_csv('QA_DATAFRAME.csv')
QA_check = QA_check[[
    'id', 'song', 'artist',
    'acousticness', 'danceability', 'duration_ms', 'energy', 'instrumentalness',
    'key', 'liveness', 'loudness', 'mode', 'speechiness', 'tempo', 'time_signature', 'valence'
]]
```

In [15]:
```
idx = '7tJm0Kl0FBobXB011e82sN'
value1 = float(check_vs[check_vs.id == idx][feature])
value2 = float(qa_check[qa_check.id == idx][feature])
squared_error = (value1 - value2) * (value1 - value2)
squared_error
```

Out[15]:
```
2.0463631939738127e-18
```

In [16]:
```
QA_results = QA_check.copy()

audio_features = [
    'acousticness', 'danceability', 'duration_ms', 'energy', 'instrumentalness',
    'key', 'liveness', 'loudness', 'mode', 'speechiness', 'tempo', 'time_signature', 'valence'
]

for i, row in QA_results.iterrows():
    idx = QA_results['id'].iloc[i]
    for feature in audio_features:
        value1 = float(check_vs[check_vs.id == idx][feature])
        value2 = float(qa_check[qa_check.id == idx][feature])
        squared_error = (value1 - value2) * (value1 - value2)
        QA_results[feature].iloc[i] = squared_error
```

In [18]:
```
QA_results.describe()
# major errors only in duration, loudness, and mode
```

Out[18]:

	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	tempo	time_signature	valence
count	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
mean	0.00	0.00	11_664.31	0.00	0.00	0.09	0.00	0.55	0.01	0.00	0.00	0.00	0.00
std	0.00	0.00	116_639.97	0.00	0.00	0.90	0.00	5.48	0.10	0.00	0.01	0.00	0.00
min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
75%	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
max	0.02	0.00	1_166_400.00	0.04	0.00	9.00	0.00	54.82	1.00	0.00	0.10	0.00	0.00

2 Songs out of 100 have errors, let's investigate

In [20]:
```
problem_songs = ['095MMFhB9qxPx2VsmvjnUs', '3BUWNzPWz2mDbptZmGEXpB']

QA_results[QA_results.id.isin(problem_songs)]
```

Out[20]:

	id	song	artist	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	temp
16	095MMFhB9qxPx2VsmvjnUs	Live To Tell	Madonna	0.02	0.00	1166400	0.04	0.00	0	0.00	54.82	0	0.00	0.0
45	3BUWNzPWz2mDbptZmGEXpB	Let It All Work Out	Lil Wayne	0.00	0.00	0	0.00	0.00	9	0.00	0.00	1	0.00	0.0

In [24]:
```
# info gathered from other sources
check_vs[check_vs.id.isin(problem_songs)]
```

Out[24]:

	id	song	artist	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	t
15579	3BUWNzPWz2mDbptZmGEXpB	Let It All Work Out	Lil Wayne	0.02	0.55	316192.00	0.56	0.00	8.00	0.30	-6.86	0.00	0.16	1
16559	095MMFhB9qxPx2VsmvjnUs	Live To Tell	Madonna	0.41	0.67	353000.00	0.31	0.03	2.00	0.11	-14.16	0.00	0.03	1

In [25]:
```
# info I gathered from Spotify API
QA_check[QA_check.id.isin(problem_songs)].sort_values('artist')
```

Out[25]:

	id	song	artist	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	temp
45	3BUWNzPWz2mDbptZmGEXpB	Let It All Work Out	Lil Wayne	0.02	0.55	316192	0.56	0.00	11	0.30	-6.86	1	0.16	135.8
16	095MMFhB9qxPx2VsmvjnUs	Live To Tell	Madonna	0.26	0.65	351920	0.52	0.00	2	0.11	-6.75	0	0.03	109.9

Madonna Duration Error

In [28]:
```
(351920 - 353000.00) / 1000

# off by one second, square error looks big because it's squared ms
# odd that this would register incorrectly, but does not seem significant
```

Out[28]:
```
-1.08
```

Madonna Loudness Error

- The older data (from the database) is very quiet (-14.16 dB)
  - Maybe Madonna was allowed to remaster the track but keep the id for the track
  - https://artists.spotify.com/help/article/re-uploading-music

In [30]:
```
QA_check['loudness'].describe()
```

Out[30]:
```
count   100.00
mean     -8.97
std       3.09
min     -16.84
25%     -11.48
50%      -8.87
75%      -6.52
max      -2.27
Name: loudness, dtype: float64
```

- all of the other characteristics of the song remained the same
  - just 1s was trimmed, and the loudness was boosted
  - definitely sounds like a remaster, but I haven't been able to verify
- conclusions:
  - this does not seem like it will affect the results
  - louder music is known to be more appealing
    - however, this bias is baked into Madonna's music library
    - arguably, she was at a disadvantage to begin with
  - in addition, this only happened in 1 of the 100 sampled songs

lil Wayne Key Signature Error

- the data say the key is either D or B
  - mode is also wrong, which corresponds to this
- D is bm (not B)
- it is in B (confirmed) - CORRECT IS key = 11
- checked the SQL database as well, and it has key = 8 (G#?)
  - mode also incorrect
  - all other attributes are the same
- conclusion:
  - this song is very atonal / detuned, vaguely keyed
  - it seems reasonable to me that an AI (or human) would guess the key incorrectly
  - I was not expecting to need to learn to play a lil wayne song for this project

Overall QA Conclusions

- There is a large degree of consistency when checking 100 randomly sampled songs.
- Along with many spot checks, the dataset seems to have high enough integrity to perform analytics.
- Future QA may be considered as well, potentially with statistical tests to determine confidence levels.

In [ ]:
```
```

Attachment 2

Import Modules

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

# jupyter notebook full-width display
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))

# pandas formatting
pd.set_option('display.float_format', '{:_.3f}'.format)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 200)
```

Import Data and Optimise

from: Kevin 820 Data Import and Clean.ipynb convert to parquet formating to save space, time, and preserve formatting

In [2]:
```
desired_formatting = [
    'id', 'song', 'artist',
    'acousticness', 'danceability', 'duration_ms', 'energy',
    'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
    'speechiness', 'tempo', 'time_signature', 'valence'
]

desired_formatting_timeseries = [
    'date',
    'id', 'song', 'artist',
    'rank', 'last-week', 'peak-rank', 'weeks-on-board',
    'acousticness', 'danceability', 'duration_ms', 'energy',
    'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
    'speechiness', 'tempo', 'time_signature', 'valence'
]

# datatypes to reduce file sizes and increase calculation speed
# Int8 and float16 is enough to store data, but error occur with descriptive stats
# Try Int16 and float32
dtypes = {
    'key': 'Int16', 'mode': 'Int16', 'time_signature': 'Int16', 'tempo': 'float32',
    'acousticness': 'float32', 'danceability': 'float32', 'duration_ms': 'Int64',
    # duration_ms needs 64bit to avoid overflow errors
    'energy': 'float32', 'instrumentalness': 'float32', 'liveness': 'float32',
    'loudness': 'float32', 'speechiness': 'float32', 'valence': 'float32'
}

dtypes_timeseries = {
    'rank': 'Int16', 'last-week': 'Int16', 'peak-rank': 'Int16', 'weeks-on-board': 'Int16'
}

# all songs with audio features (combined from 3 sources)
df_10M = pd.read_csv('every_song_with_data.csv', dtype=dtypes)
df_10M = df_10M[desired_formatting]

# all Billboard 100 lists, audio features included where possible
df_B100 = pd.read_csv(
    'all_audio_features_billboard_100.csv',
    dtype={**dtypes, **dtypes_timeseries}
)
df_B100 = df_B100[desired_formatting_timeseries]
df_B100['date'] = pd.to_datetime(df_B100['date'])

# all unique songs from the Billboard 100 lists, audio features included where possible
df_B100_songs = pd.read_csv('all_audio_features_billboard_100_songs.csv', dtype=dtypes)
df_B100_songs = df_B100_songs[desired_formatting]
```

In [3]:
```
# check size
df_10M.shape
```

Out[3]:
```
(9595992, 16)
```

In [4]:
```
# No errors with Int16 and float32, other than duration_ms (64bit)

df_10M.describe().loc['mean':'max'].T
```

Out[4]:

	mean	std	min	25%	50%	75%	max
acousticness	0.421	0.374	0.000	0.034	0.336	0.817	0.996
danceability	0.528	0.190	0.000	0.396	0.545	0.676	1.000
duration_ms	238_209.591	159_341.591	0.000	169_600.000	216_933.000	275_080.000	19_672_058.000
energy	0.545	0.282	0.000	0.310	0.567	0.789	1.000
instrumentalness	0.258	0.374	0.000	0.000	0.002	0.645	1.000
key	5.237	3.542	0.000	2.000	5.000	8.000	11.000
liveness	0.210	0.180	0.000	0.096	0.129	0.262	1.000
loudness	-10.967	6.318	-60.000	-13.675	-9.196	-6.398	7.234
mode	0.661	0.473	0.000	0.000	1.000	1.000	1.000
speechiness	0.098	0.135	0.000	0.036	0.047	0.082	0.974
tempo	118.785	30.832	0.000	95.080	118.950	137.450	249.987
time_signature	3.840	0.567	0.000	4.000	4.000	4.000	5.000
valence	0.474	0.278	0.000	0.234	0.468	0.708	1.000

SIGNIFICANT FIGURES CHECK

to check data types

In [6]:
```
# 3 sig figs of accuracy is adequate for audio features
# everything beyond that appears to be rounding error, and shouldn't affect calculations
# float16 is adequate to store data (but float32 is required for statistical calculations)

df_10M.query('id=="33ZXjLCpiINn8eQIDYEPTD"')
```

Out[6]:

	id	song	artist	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	spe
22205	33ZXjLCpiINn8eQIDYEPTD	Shook Ones, Pt. II	Mobb Deep	0.0146000003	0.7630000114	325506	0.7860000134	0.0114000002	10	0.0816999972	-6.4720001221	0	0.229

In [7]:
```
df_10M.query('id=="2jKoVlU7VAmExKJ1Jh3w9P"')
```

Out[7]:

	id	song	artist	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	
392848	2jKoVlU7VAmExKJ1Jh3w9P	Alkaholik (feat. Erik Sermon, J Ro & Tash)	Tash	0.1800000072	0.8930000067	219160	0.5139999986	0.0000000000	11	0.0595999993	-5.0799999237	1	(

In [8]:
```
df_10M.query('id=="7iL6o9tox1zgHpKUfh9vuC"')
```

Out[8]:

	id	song	artist	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	spee
22198	7iL6o9tox1zgHpKUfh9vuC	In Da Club	50 Cent	0.2549999952	0.8989999890	193466	0.7129999995	0.0000000000	6	0.0707999989	-2.7520000935	0	0.3659

Grid view | Form view

Total rows loaded: 10

	id	acousticness	analysis_url	danceability
1	2jKoVlU7VAmExKJ1Jh3w9P	0.18000000715256	https://api.spotify.com/v1/audio-analysis/2jKoVlU7VAmExKJ1Jh3w9P	0.89300000667572
2	4JYUDRtPZuVNi7FAnbHyux	0.27200001478195	https://api.spotify.com/v1/audio-analysis/4JYUDRtPZuVNi7FAnbHyux	0.51999998092651
3	6YjKAkDYmlasMqYw73iB0w	0.07829999923706	https://api.spotify.com/v1/audio-analysis/6YjKAkDYmlasMqYw73iB0w	0.91799998283386
4	2YlvHjDb4Tyxl4A1IcDhAe	0.58399999141693	https://api.spotify.com/v1/audio-analysis/2YlvHjDb4Tyxl4A1IcDhAe	0.87699997425079
5	3UOuBNEin5peSRqdzvlnWM	0.17000000178814	https://api.spotify.com/v1/audio-analysis/3UOuBNEin5peSRqdzvlnWM	0.81400001049042
6	2g8HN35AnVGIk7B8yMucww	0.43000000715256	https://api.spotify.com/v1/audio-analysis/2g8HN35AnVGIk7B8yMucww	0.77999997138977
7	7iL6o9tox1zgHpKUfh9vuC	0.25499999523163	https://api.spotify.com/v1/audio-analysis/7iL6o9tox1zgHpKUfh9vuC	0.89899998903275
8	6KIKRz9eSTXdNsGUnomdtW	0.04610000178218	https://api.spotify.com/v1/audio-analysis/6KIKRz9eSTXdNsGUnomdtW	0.83399999141693
9	33ZXjLCpiINn8eQIDYEPTD	0.01460000034422	https://api.spotify.com/v1/audio-analysis/33ZXjLCpiINn8eQIDYEPTD	0.76300001144409
10	7N1Vjtzr1lmmCW9iasQ8YO	0.125	https://api.spotify.com/v1/audio-analysis/7N1Vjtzr1lmmCW9iasQ8YO	0.81300002336502

Save Optimised Files as Parquet Format

In [ ]:

```
# delay - don't write over if rerun

# # Save Data
# df_10M.to_parquet('df_10M.parquet')
# df_B100.to_parquet('df_B100.parquet')
# df_B100_songs.to_parquet('df_B100_songs.parquet')
```

Based on Lit Review:

I should be including:

RELEASE DATE & GENRE

Import Release Date

Spotify API (Hot 100 songs)

get a temporary authorization token from: https://developer.spotify.com/console/get-search-item

In [121…

```
# input the temporary token
TEMP_TOKEN = input('Enter token: ')

# create a spotify object
spotify = spotipy.Spotify(auth=TEMP_TOKEN)
```

```
Enter token: BQCoJoa0yB7VAVrWrWQs7pnv8cWO2bb8XevJRVCPKAkPD6eGKANwzZWQKnT2UzXPApJbSpIJpx9rVPD0Y8ICr7i4Uihy6mCL6nQ78lsMKtMHr24psF5ZPYdZfBxbt8A33
lN_748f7xuLAwBjy64_VNy2UKXa6HQ8L2FtFmp8oV38
```

In [118…

```
def get_release_date(track_id):
    track_info = spotify.track(track_id)
    return track_info['album']['release_date']
```

In [119…

```
try: # only do this once, if row exists, just show head()
    display(df_B100_songs['release_date'].head())
except:
    df_B100_songs['release_date'] = ''
```

```
0          1965
1    1965-01-01
2          1965
3           NaN
4    1996-12-31
Name: release_date, dtype: object
```

In [122…

```
# start over at
start_over_at = 30000

# populate df_billboard_songs with ids, where available
for i, row in df_B100_songs.iterrows():

    # start over at
    if i < start_over_at-1:
```

```
        continue

    # show status update
    if i%100 == 0:
        print(i, end='  ')
    if i%1000 == 0:
        print()

    # start over where we finished (don't overwrite known release_dates)
    # np.nan != '' should work
    if row['release_date'] != '':
        continue

    # if id nan, release_date is unknown
    if pd.isna(row['id']):  # if id is not known
        df_B100_songs.loc[i, 'release_date'] = np.nan
    else:
        df_B100_songs.loc[i, 'release_date'] = get_release_date(row['id'])

        # save every 1000 rows, if new
        if i%1000 == 0:
            df_B100_songs.to_parquet('df_B100_songs_TEMP.parquet')

# save to finished dataframe
df_B100_songs.to_parquet('df_B100_songs_RELEASEDATE.parquet')
```

```
27800  27900  28000
28100  28200  28300  28400  28500  28600  28700  28800  28900  29000
29100  29200  29300  29400  29500  29600
```

In [129…

```
df_B100_songs.head(20)
```

Out[129…

	id	song	artist	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	t
0	7DZsH0df0GuULl0FGwXMfd	Misty	"Groove" Holmes	0.806	0.581	360986	0.556	0.925	8	0.218	-10.422	1	0.054	
1	11Aldbvo6UCcVhBzv4oUdw	What Now My Love	"Groove" Holmes	0.648	0.562	327560	0.573	0.314	7	0.056	-10.972	1	0.034	
2	4KRLWRl1bFjnXhY5MgZWrM	May The Bird Of Paradise Fly Up Your Nose	"Little" Jimmy Dickens	0.738	0.660	151693	0.801	0.000	4	0.627	-8.446	1	0.115	1
3	None	I Know I Know	"Pookie" Hudson	NaN	NaN	<NA>	NaN	NaN	<NA>	NaN	NaN	<NA>	NaN	
4	5r96TaQquRrlo3Ym3ZlSL2	Amish Paradise	"Weird Al" Yankovic	0.103	0.728	202920	0.448	0.000	8	0.267	-10.540	1	0.172	
5	1gloYGAZI6eHp6MEPjLuL3	Canadian Idiot	"Weird Al" Yankovic	0.002	0.543	143040	0.697	0.000	8	0.343	-9.211	1	0.061	1
6	0WuYuWhLws8VahMy2zLLRJ	Eat It	"Weird Al" Yankovic	0.087	0.767	200627	0.811	0.000	7	0.068	-8.548	1	0.077	1
7	4is3oF4FlWmedh3TK6Ke7z	Fat	"Weird Al" Yankovic	0.166	0.870	216066	0.551	0.000	6	0.064	-10.563	0	0.060	1
8	022XE9RZVU3a9nULENXfnm	I Lost On Jeopardy	"Weird Al" Yankovic	0.347	0.887	208933	0.769	0.000	0	0.051	-9.150	0	0.050	1
9	0uTSpsec5kXkNQQjzt6dIB	King Of Suede	"Weird Al" Yankovic	0.271	0.780	255373	0.808	0.000	11	0.096	-10.056	0	0.038	1
10	3XhyafaTVD6ViaRFIL4bNo	Like A Surgeon	"Weird Al" Yankovic	0.252	0.838	210627	0.671	0.000	3	0.056	-8.328	0	0.035	1
11	76QlOaKjVbxqduVcwcQZWx	Ricky	"Weird Al" Yankovic	0.066	0.521	157066	0.814	0.000	4	0.087	-8.270	1	0.108	1
12	3rREGSHjY1RmsTj81wL5X4	Smells Like Nirvana	"Weird Al" Yankovic	0.143	0.596	225133	0.760	0.002	6	0.263	-8.217	1	0.072	1

	id	song	artist	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	t
13	60R2v9lheAu3lwZwAFxMZK	White & Nerdy	"Weird Al" Yankovic	0.099	0.791	170640	0.613	0.000	1	0.076	-11.628	0	0.076	1
14	4JqQWAr47pGEoaMArpA7Z3	Word Crimes	"Weird Al" Yankovic	0.012	0.897	223120	0.430	0.000	7	0.047	-12.759	1	0.055	1
15	08zfRiERqpy7J0x8akk7zt	(God Must Have Spent) A Little More Time On You	'N Sync	0.449	0.375	241493	0.527	0.000	10	0.305	-8.422	1	0.051	1
16	62bOmKYxYg7dhrC6gH9vFn	Bye Bye Bye	'N Sync	0.031	0.610	200400	0.926	0.001	8	0.082	-4.843	0	0.048	1
17	4CCUjYJPbSXLL23BFeBVbI	Gone	'N Sync	0.430	0.704	292000	0.409	0.000	11	0.109	-8.581	0	0.060	1
18	6aw26tMh7aY9S7rqQDDaIQ	I Drive Myself Crazy	'N Sync	0.043	0.562	240733	0.718	0.000	9	0.174	-6.551	1	0.027	
19	221LRlPHPuevgE1tuUlof9	I Want You Back	'N Sync	0.052	0.751	200440	0.939	0.001	8	0.477	-3.305	0	0.047	1

In [130…

```
# convert to datetime
df_B100_songs['release_date'] = pd.to_datetime(df_B100_songs['release_date'], errors = 'coerce')
```

In [131…

```
df_B100_songs.head(20)
```

Out[131…

	id	song	artist	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	t
0	7DZsH0df0GuULl0FGwXMfd	Misty	"Groove" Holmes	0.806	0.581	360986	0.556	0.925	8	0.218	-10.422	1	0.054	
1	11Aldbvo6UCcVhBzv4oUdw	What Now My Love	"Groove" Holmes	0.648	0.562	327560	0.573	0.314	7	0.056	-10.972	1	0.034	
2	4KRLWRl1bFjnXhY5MgZWrM	May The Bird Of Paradise Fly Up Your Nose	"Little" Jimmy Dickens	0.738	0.660	151693	0.801	0.000	4	0.627	-8.446	1	0.115	1
3	None	I Know I Know	"Pookie" Hudson	NaN	NaN	<NA>	NaN	NaN	<NA>	NaN	NaN	<NA>	NaN	
4	5r96TaQquRrlo3Ym3ZlSL2	Amish Paradise	"Weird Al" Yankovic	0.103	0.728	202920	0.448	0.000	8	0.267	-10.540	1	0.172	
5	1gloYGAZl6eHp6MEPjLuL3	Canadian Idiot	"Weird Al" Yankovic	0.002	0.543	143040	0.697	0.000	8	0.343	-9.211	1	0.061	1
6	0WuYuWhLws8VahMy2zLLRJ	Eat It	"Weird Al" Yankovic	0.087	0.767	200627	0.811	0.000	7	0.068	-8.548	1	0.077	1
7	4is3oF4FlWmedh3TK6Ke7z	Fat	"Weird Al" Yankovic	0.166	0.870	216066	0.551	0.000	6	0.064	-10.563	0	0.060	1
8	022XE9RZVU3a9nULENXfnm	I Lost On Jeopardy	"Weird Al" Yankovic	0.347	0.887	208933	0.769	0.000	0	0.051	-9.150	0	0.050	1
9	0uTSpsec5kXkNQQjzt6dIB	King Of Suede	"Weird Al" Yankovic	0.271	0.780	255373	0.808	0.000	11	0.096	-10.056	0	0.038	1
10	3XhyafaTVD6ViaRFIL4bNo	Like A Surgeon	"Weird Al" Yankovic	0.252	0.838	210627	0.671	0.000	3	0.056	-8.328	0	0.035	1

	id	song	artist	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	t
11	76QlOaKjVbxqduVcwcQZWx	Ricky	"Weird Al" Yankovic	0.066	0.521	157066	0.814	0.000	4	0.087	-8.270	1	0.108	1
12	3rREGSHjY1RmsTj81wL5X4	Smells Like Nirvana	"Weird Al" Yankovic	0.143	0.596	225133	0.760	0.002	6	0.263	-8.217	1	0.072	1.
13	60R2v9lheAu3lwZwAFxMZK	White & Nerdy	"Weird Al" Yankovic	0.099	0.791	170640	0.613	0.000	1	0.076	-11.628	0	0.076	1.
14	4JqQWAr47pGEoaMArpA7Z3	Word Crimes	"Weird Al" Yankovic	0.012	0.897	223120	0.430	0.000	7	0.047	-12.759	1	0.055	1.
15	08zfRiERqpy7J0x8akk7zt	(God Must Have Spent) A Little More Time On You	'N Sync	0.449	0.375	241493	0.527	0.000	10	0.305	-8.422	1	0.051	1(
16	62bOmKYxYg7dhrC6gH9vFn	Bye Bye Bye	'N Sync	0.031	0.610	200400	0.926	0.001	8	0.082	-4.843	0	0.048	1'
17	4CCUjYJPbSXLL23BFeBVbI	Gone	'N Sync	0.430	0.704	292000	0.409	0.000	11	0.109	-8.581	0	0.060	1
18	6aw26tMh7aY9S7rqQDDaIQ	I Drive Myself Crazy	'N Sync	0.043	0.562	240733	0.718	0.000	9	0.174	-6.551	1	0.027	:
19	221LRlPHPuevgE1tuUlof9	I Want You Back	'N Sync	0.052	0.751	200440	0.939	0.001	8	0.477	-3.305	0	0.047	1

In [132…
```
# save to dataframe
df_B100_songs.to_parquet('df_B100_songs_RELEASEDATE.parquet')
```

From SQL (10M songs)

In [4]:
```
# import sql release dates
sql_release_dates = pd.read_csv('release_dates_from_sql.csv')
```

In [5]:
```
# replace anything older than billboard with arbitrary date, to remove out of bound error
# min
min_date, replace_date = -360200000000, -2208988800000
df_B100.date.min(), pd.to_datetime(min_date, unit='ms', origin='unix'), pd.to_datetime(replace_date, unit='ms', origin='unix')
```

Out[5]:
```
(Timestamp('1958-08-04 00:00:00'),
 Timestamp('1958-08-03 00:26:40'),
 Timestamp('1900-01-01 00:00:00'))
```

In [6]:
```
# replace dates
sql_release_dates.loc[sql_release_dates.release_date < min_date, 'release_date'] = replace_date

# change to datetime format
sql_release_dates['release_date'] = pd.to_datetime(sql_release_dates['release_date'], unit='ms', origin='unix')
```

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

Out[7]:

	id	release_date
0	2g8HN35AnVGIk7B8yMucww	1994-09-13
1	4E5IFAXCob6QqZaJMTw5YN	2003-01-01
2	1gSt2UlC7mtRtJIc5zqKWn	2007-01-01
3	67IvfvAMYQzJEeHopvwMMW	1999-10-19
4	2I9foKseoFQh07p6sD2voE	2003-02-06

also need 1.2M file release dates

In [8]:

```
# import csv release dates
csv_release_dates = pd.read_csv(r'D:\RYERSON\820\Datasets\Spotify 1.2M+ Songs\tracks_features.csv')

# remove useless columns
csv_release_dates = csv_release_dates[['id', 'release_date']]

# change to datetime format
# https://stackoverflow.com/questions/32888124/pandas-out-of-bounds-nanosecond-timestamp-after-offset-rollforward-plus-adding-a
csv_release_dates['release_date'] = pd.to_datetime(csv_release_dates['release_date'], errors = 'coerce')
```

merge release date files

In [24]:
```
csv_release_dates[~csv_release_dates.id.isin(list(sql_release_dates.id))].shape[0], csv_release_dates.shape[0]
```

Out[24]:
```
(857486, 1204025)
```

In [31]:
```
all_release_dates = pd.concat(
    # only add csv data that is missing from sql data, to reduce duplicates
    [sql_release_dates, csv_release_dates[~csv_release_dates.id.isin(list(sql_release_dates.id))]]
# any full duplicates are consistent, keep the first
# completely drop any inconsistent release dates with second drop_duplicates() call
).drop_duplicates().drop_duplicates(subset=['id'], keep=False).reset_index(drop=True)
```

In [32]:
```
all_release_dates.shape, pd.concat([sql_release_dates, csv_release_dates]).shape, sql_release_dates.shape
```

Out[32]:
```
((9449487, 2), (11104198, 2), (9900173, 2))
```

In [33]:
```
all_release_dates[all_release_dates.duplicated(subset=['id'], keep=False)].sort_values('id')
```

Out[33]:

	id	release_date

In [11]:
```
# no longer duplicated because of .drop_duplicates(subset=['id'], keep=False)

# track_id = '7zqguRvtjpcu37IzN3nCVd'

# track_info = spotify.track(track_id)
# artist_id = track_info['artists'][0]['id']

# track_info
```

now merge with dataset

In [37]:
```
df_10M.shape
```

Out[37]:
```
(9595992, 16)
```

In [38]:
```
df_10M = pd.merge(df_10M, all_release_dates, how='left', on='id')
df_10M.shape
```

Out[38]:
```
(9595992, 17)
```

In [39]:
```
# 9% missings release dates with sql data only (missing the 1.2M dataset)
# 1.5% missings release dates (after adding the 1.2M dataset)
df_10M.loc[pd.isnull(df_10M.release_date), 'release_date'].shape[0] / df_10M.release_date.shape[0]
```

Out[39]:
```
0.015326919822359168
```

In [40]:
```
# missing
df_10M.loc[pd.isnull(df_10M.release_date), 'release_date'].shape[0]
```

Out[40]:
```
147077
```

In [12]:
```
# replace null dates with replace_date
df_10M.loc[pd.isnull(df_10M.release_date), 'release_date'] = pd.to_datetime(replace_date, unit='ms', origin='unix')
```

In [43]:
```
df_10M.head()
```

Out[43]:

	id	song	artist	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	temp
0	7DZsH0df0GuULl0FGwXMfd	Misty	"Groove" Holmes	0.806	0.581	360986	0.556	0.925	8	0.218	-10.422	1	0.054	93.4
1	11Aldbvo6UCcVhBzv4oUdw	What Now My Love	"Groove" Holmes	0.648	0.562	327560	0.573	0.314	7	0.056	-10.972	1	0.034	91.8
2	4KRLWRl1bFjnXhY5MgZWrM	May The Bird Of Paradise Fly Up Your Nose	"Little" Jimmy Dickens	0.738	0.660	151693	0.801	0.000	4	0.627	-8.446	1	0.115	104.3
3	5r96TaQquRrlo3Ym3ZlSL2	Amish Paradise	"Weird Al" Yankovic	0.103	0.728	202920	0.448	0.000	8	0.267	-10.540	1	0.172	80.9
4	1gloYGAZI6eHp6MEPjLuL3	Canadian Idiot	"Weird Al" Yankovic	0.002	0.543	143040	0.697	0.000	8	0.343	-9.211	1	0.061	185.9

In [41]:
```
# check shape
df_10M.shape
```

Out[41]:
```
(9595992, 17)
```

In [42]:
```
# Save Data
df_10M.to_parquet('df_10M_RELEASEDATE.parquet')
```

Import Genre

ALGORITHM

1. create a most popular genre list for step 3.
2. check df_track_genres_SQL for track match
   - match most popular genre first
3. check for an artist match in df_artist_genres_SQL
   - match most popular genre first
4. check Spotify API and get remaining genres
   - match most popular genre first
   - then use alphabetical if not found

NOTES

- there are a lot of very specific genres
- may need to either exclude rare genres, of figure out how to group into genre families

Reimport Data First

In [44]:
```
# all songs with audio features (combined from 3 sources)
df_10M = pd.read_parquet('df_10M_RELEASEDATE.parquet')

# all unique songs from the Billboard 100 lists, audio features included where possible
df_B100_songs = pd.read_parquet('df_B100_songs_RELEASEDATE.parquet')
```

In [45]:
```
# import csv again to check for genre data
csv_12M = pd.read_csv(r'D:\RYERSON\820\Datasets\Spotify 1.2M+ Songs\tracks_features.csv')
csv_12M.columns
# csv doesn't have genre data
```

Out[45]:
```
Index(['id', 'name', 'album', 'album_id', 'artists', 'artist_ids',
       'track_number', 'disc_number', 'explicit', 'danceability', 'energy',
       'key', 'loudness', 'mode', 'speechiness', 'acousticness',
       'instrumentalness', 'liveness', 'valence', 'tempo', 'duration_ms',
       'time_signature', 'year', 'release_date'],
      dtype='object')
```

import genre queries from SQL

In [46]:
```
# all genres for each track
df_track_genres_SQL = pd.read_csv('track_genre_sql.csv')

# all genres for all artists from SQL
df_artist_genres_SQL = pd.read_csv('artist_genre_sql.csv')
```

In [47]:
```
df_artist_genres_SQL.head()
```

Out[47]:

	artist_id	name	genre_id
0	4tujQJicOnuZRLiBFdp3Ou	Xzibit	detroit hip hop
1	4tujQJicOnuZRLiBFdp3Ou	Xzibit	g funk
2	4tujQJicOnuZRLiBFdp3Ou	Xzibit	gangster rap
3	4tujQJicOnuZRLiBFdp3Ou	Xzibit	hardcore hip hop
4	4tujQJicOnuZRLiBFdp3Ou	Xzibit	hip hop

STEP 0

create an ordered list of genres

In [48]:
```
# create a sorted list of genres
# only use the most common genre when multiple are listed

genre_counts = (
    df_track_genres_SQL[['track_id','genre_id']]
    .groupby(['genre_id'])
    .count()
    .sort_values('track_id', ascending=False)
)

genre_counts.columns = ['count']

# top 10 genres
genre_counts.head(10)
```

Out[48]:

	count
genre_id	
classical	495467
classical performance	287065
adult standards	191882
rock	136731
early music	127095
orchestral performance	119693
vocal jazz	117062
orchestra	113422
classic rock	109292
lounge	108578

In [49]:
```
list_of_ordered_genres = list(genre_counts.index)
```

In [50]:
```
list_of_ordered_genres[0:10]
```

Out[50]:
```
['classical',
 'classical performance',
 'adult standards',
 'rock',
 'early music',
 'orchestral performance',
 'vocal jazz',
 'orchestra',
 'classic rock',
 'lounge']
```

STEP 0.1

sort df_track_genres_SQL by most popular genre, and delete duplicates

In [51]:
```
# merge with count data
df_track_genres_counts = (
    df_track_genres_SQL
    .merge(genre_counts, left_on='genre_id', right_index=True, how='left')
)
```

In [52]:
```
# sort and delete duplicates
df_track_genres = (
    df_track_genres_counts
    .sort_values('count', ascending=False)
    .drop_duplicates(subset=['track_id'])
    .sort_values('track_id')
    .reset_index(drop=True)
)
```

In [53]:
```
df_track_genres.head(10)
```

Out[53]:

	track_id	track_name	artist_id	genre_id	count
0	0000QBRGPosiFRXKmMYnsO	Guajira Cubana - Original Mix	05JwisTou63l1k92FQEyhT	cha-cha-cha	4088
1	0000gBWfr2zIFzE5tDzxca	Sella Stercoraria	0Tgtl5beIMahbtIzV5jBXw	ukrainian metal	949
2	0000uJA4xCdxThagdLkkLR	Heart As Cold As Stone	5kEVfWQGTw0rIDO2Jqq1ww	progressive bluegrass	24408
3	0001Lyv0YTjkZSqzT4WkLy	Eye Of The Hurricane	1YPYaJyUobMi0eABhZo92N	experimental	24952
4	0001QZSdENvrMx6cZXZJdo	Let the Loser Melt	0pNaVvqSvldpJI7pHpNoM9	alternative rock	52941
5	0003Z98F6hUq7XxqSRM87H	Меня больше тут нет	08RxfNkJpjJ4dJb4xASWzj	russian hip hop	5986
6	0003q2V7hAiIYyzXV4sNyQ	Das wilde Pack, Teil 1: Das wilde Pack, Kapitel 5	7vYT04Nb7z9QUTf4F8oG2c	kleine hoerspiel	8627
7	000490QTLqT1ftnfwj3kGF	Disco Guitar	0cKrM2XKF7wxyfEQDauvyQ	electro	12183
8	00053lDuLvN8Q8voGT3GCt	Soki	6HAIuUS4d8W4zjDTl6rsaU	makossa	3383
9	0005ZRV0QChWUWheZkocOW	Eu Corro Pros Teus Braços	1sCtt8DOdGFdxfRI3tPAr4	adoracao	4376

In [54]:
```
# check what is missing
# tracks with genres VS all track ids
df_track_genres.shape[0], df_10M.drop_duplicates(subset=['id']).shape[0]
```

Out[54]:
```
(6558503, 9595992)
```

STEP 0.2

sort df_artist_genres_SQL by most popular genre, and delete duplicates

In [55]:
```
# merge with count data
df_artist_genre_counts = (
    df_artist_genres_SQL
    .merge(genre_counts, left_on='genre_id', right_index=True, how='left')
)
```

In [56]:
```
# sort and delete duplicates
df_artist_genres = (
    df_artist_genre_counts
    .sort_values('count', ascending=False)
    .drop_duplicates(subset=['artist_id'])
    .sort_values('artist_id')
    .reset_index(drop=True)
)
```

In [57]:
```
# how many artists have a genre (most popular, not necessarily most relevant)
df_artist_genres.shape[0]
```

Out[57]:
```
326123
```

In [58]:
```
######## PREVENT DUPLICATES IN STEP 2 ########
# dropping duplicates still leaves 97% of the data
df_artist_genres.drop_duplicates(subset=['name'], keep=False).shape[0] / df_artist_genres.shape[0]
```

Out[58]:
```
0.9687633193610999
```

In [59]:
```
# drop duplicate artists
df_artist_genres = df_artist_genres.drop_duplicates(subset=['name'], keep=False).reset_index(drop=True)
```

In [60]:
```
# save genre data in reusable format
genre_counts.to_parquet('df_genre_counts.parquet')  # ordered list is just list(genre_counts.index)
df_track_genres.to_parquet('df_track_genres.parquet')
df_artist_genres.to_parquet('df_artist_genres.parquet')
```

STEP 1

check fora **track match** for genre in df_track_genres_SQL

In [61]:
```
# how many matches are there?

df_10M.id.isin(df_track_genres.track_id).sum()
```

Out[61]:
```
6558081
```

In [62]:
```
df_B100_songs.id.isin(df_track_genres.track_id).sum()
```

Out[62]:
```
17157
```

In [63]:
```
df_track_genres.head()
```

Out[63]:

	track_id	track_name	artist_id	genre_id	count
0	0000QBRGPosiFRXKmMYnsO	Guajira Cubana - Original Mix	05JwisTou63I1k92FQEyhT	cha-cha-cha	4088
1	0000gBWfr2zIFzE5tDzxca	Sella Stercoraria	0Tgtl5beIMahbtIzV5jBXw	ukrainian metal	949
2	0000uJA4xCdxThagdLkkLR	Heart As Cold As Stone	5kEVfWQGTw0rIDO2Jqq1ww	progressive bluegrass	24408
3	0001Lyv0YTjkZSqzT4WkLy	Eye Of The Hurricane	1YPYaJyUobMi0eABhZo92N	experimental	24952
4	0001QZSdENvrMx6cZXZJdo	Let the Loser Melt	0pNaVvqSvldpJl7pHpNoM9	alternative rock	52941

In [64]:
```
df_10M.head()
```

Out[64]:

	id	song	artist	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	tem
0	7DZsH0df0GuULl0FGwXMfd	Misty	"Groove" Holmes	0.806	0.581	360986	0.556	0.925	8	0.218	-10.422	1	0.054	93.4
1	11Aldbvo6UCcVhBzv4oUdw	What Now My Love	"Groove" Holmes	0.648	0.562	327560	0.573	0.314	7	0.056	-10.972	1	0.034	91.8
2	4KRLWRl1bFjnXhY5MgZWrM	May The Bird Of Paradise Fly Up Your Nose	"Little" Jimmy Dickens	0.738	0.660	151693	0.801	0.000	4	0.627	-8.446	1	0.115	104.3
3	5r96TaQquRrlo3Ym3ZlSL2	Amish Paradise	"Weird Al" Yankovic	0.103	0.728	202920	0.448	0.000	8	0.267	-10.540	1	0.172	80.9
4	1gloYGAZI6eHp6MEPjLuL3	Canadian Idiot	"Weird Al" Yankovic	0.002	0.543	143040	0.697	0.000	8	0.343	-9.211	1	0.061	185.9

In [65]:
```
df_track_genres.head()
```

Out[65]:

	track_id	track_name	artist_id	genre_id	count
0	0000QBRGPosiFRXKmMYnsO	Guajira Cubana - Original Mix	05JwisTou63I1k92FQEyhT	cha-cha-cha	4088
1	0000gBWfr2zIFzE5tDzxca	Sella Stercoraria	0Tgtl5beIMahbtIzV5jBXw	ukrainian metal	949
2	0000uJA4xCdxThagdLkkLR	Heart As Cold As Stone	5kEVfWQGTw0rIDO2Jqq1ww	progressive bluegrass	24408
3	0001Lyv0YTjkZSqzT4WkLy	Eye Of The Hurricane	1YPYaJyUobMi0eABhZo92N	experimental	24952
4	0001QZSdENvrMx6cZXZJdo	Let the Loser Melt	0pNaVvqSvldpJl7pHpNoM9	alternative rock	52941

In [66]:
```
# merge genres into df_10M
df_10M = df_10M.merge(df_track_genres[['track_id', 'genre_id']], left_on='id', right_on='track_id', how='left')

# check id column
df_10M.genre_id.describe()
```

Out[66]:
```
count       6558081
unique         5440
top       classical
freq         417373
Name: genre_id, dtype: object
```

In [67]:
```
# top genres in the SQL database
df_10M.groupby('genre_id').count()['id'].sort_values(ascending=False)[0:10]
```

Out[67]:
```
genre_id
classical            417373
adult standards      184934
rock                 131574
hip hop               80899
tropical              67295
sleep                 60185
progressive house     51791
soul                  48461
soundtrack            45188
country rock          43596
Name: id, dtype: int64
```

In [68]:
```
# merge genres into df_B100_songs
df_B100_songs = df_B100_songs.merge(df_track_genres[['track_id', 'genre_id']], left_on='id', right_on='track_id', how='left')

# check id column
df_B100_songs.genre_id.describe()
```

Out[68]:
```
count               17157
unique                344
top       adult standards
freq                 2812
Name: genre_id, dtype: object
```

In [69]:
```
# top genres in the Billboard Hot 100
df_B100_songs.groupby('genre_id').count()['id'].sort_values(ascending=False)[0:10]
```

Out[69]:
```
genre_id
adult standards    2812
rock               2614
soul               1439
dance pop          1062
hip hop            1039
country             813
country rock        472
pop                 427
classic rock        406
pop rock            379
Name: id, dtype: int64
```

In [70]:
```
# drop useless columns
df_10M = df_10M.drop('track_id', axis=1)
df_B100_songs = df_B100_songs.drop('track_id', axis=1)
```

In [71]:
```
# confirm no added rows
df_B100_songs.shape, df_10M.shape
```

Out[71]:
```
((29681, 18), (9595992, 18))
```

In [73]:
```
# INTERMEDIATE SAVE

df_B100_songs.to_parquet('df_B100_songs_STEP1COMPLETE.parquet')
df_10M.to_parquet('df_10M_STEP1COMPLETE.parquet')
```

STEP 2

check for an **artist match** for genre in df_artist_genres_SQL

NOTE:

- matching on artist name creates duplicates
- artist genres fixed to drop duplicated names in this dataframe

- now 97% of original dataframe is still included

In [74]:
```
# how many matches are there, excluding currently categorised tracks?

df_10M[df_10M.genre_id.isnull()].artist.isin(df_artist_genres.name).sum()
```

Out[74]:
```
77957
```

In [75]:
```
df_B100_songs[df_B100_songs.genre_id.isnull()].artist.isin(df_artist_genres.name).sum()
```

Out[75]:
```
5078
```

In [76]:
```
def choose_genre(row):
    genre1, genre2 = row['genre_id_x'], row['genre_id_y']
    if pd.notna(genre1):
        return genre1
    elif pd.notna(genre2):
        return genre2
    else:
        return np.nan

df_10M = df_10M.merge(df_artist_genres[['name', 'genre_id']], left_on='artist', right_on='name', how='left')
df_10M['genre_id'] = df_10M.apply(choose_genre, axis=1)
df_10M = df_10M.drop(['name', 'genre_id_x', 'genre_id_y'], axis=1)
```

In [77]:
```
df_10M.head()
```

Out[77]:

	id	song	artist	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	temp
0	7DZsH0df0GuULl0FGwXMfd	Misty	"Groove" Holmes	0.806	0.581	360986	0.556	0.925	8	0.218	-10.422	1	0.054	93.4
1	11Aldbvo6UCcVhBzv4oUdw	What Now My Love	"Groove" Holmes	0.648	0.562	327560	0.573	0.314	7	0.056	-10.972	1	0.034	91.8
2	4KRLWRl1bFjnXhY5MgZWrM	May The Bird Of Paradise Fly Up Your Nose	"Little" Jimmy Dickens	0.738	0.660	151693	0.801	0.000	4	0.627	-8.446	1	0.115	104.3
3	5r96TaQquRrlo3Ym3ZlSL2	Amish Paradise	"Weird Al" Yankovic	0.103	0.728	202920	0.448	0.000	8	0.267	-10.540	1	0.172	80.9
4	1gloYGAZI6eHp6MEPjLuL3	Canadian Idiot	"Weird Al" Yankovic	0.002	0.543	143040	0.697	0.000	8	0.343	-9.211	1	0.061	185.9

In [78]:
```
df_10M.genre_id.count()
```

Out[78]:
```
6636234
```

In [79]:
```
df_B100_songs = df_B100_songs.merge(df_artist_genres[['name', 'genre_id']], left_on='artist', right_on='name', how='left')
df_B100_songs['genre_id'] = df_B100_songs.apply(choose_genre, axis=1)
df_B100_songs = df_B100_songs.drop(['name', 'genre_id_x', 'genre_id_y'], axis=1)
```

In [80]:
```
df_B100_songs.genre_id.count()
```

Out[80]:
```
22235
```

Let's look at what's missing

In [267…
```
df_missing_big = df_10M[pd.isnull(df_10M.genre_id)]
```

In [268…
```
# i ran this dozens of times and didn't recognise any bands, could ignore finding these
df_missing_big.sample(10)
```

Out[268…

	id	song	artist	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode

	id	song	artist	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode
8880921	0N6Pkx7D8EHURMn9A2FsOL	Hilnaric	['TenHornedBeast']	0.529	0.307	185853	0.404	0.879	1	0.158	-14.519	1
8428072	0COeKHqRE2H8bMJLlivDgx	Talismaanit	Kaupungin Valot	0.219	0.423	355947	0.523	0.015	7	0.069	-9.260	1
6908589	3EXwUz0ppyITiBiM837qD9	Countdown (As Made Famous by Beyonce)	Déjà Vu	0.055	0.699	218991	0.721	0.045	5	0.345	-9.648	1
6184592	5Iavrc74BNDBpj89ewI3yJ	Both Worlds	C-Mon & Kypski	0.030	0.631	469493	0.413	0.611	2	0.128	-7.836	0
6068781	2R0EVqGOIg8IsU9mJyH2Zz	Hate That I Love You	R&B Divas United	0.518	0.601	219932	0.756	0.000	8	0.104	-9.756	1
9145039	5r9BuM0fWq4BJJd2mc3dc9	I've Never Been In Love Before	['Dena DeRose', 'Dwayne Burno', 'Matt Wilson']	0.778	0.595	263200	0.416	0.159	7	0.165	-13.142	0
6577711	4ki2NZRhrmcPkDf761zYPt	Inc Ways	Midnight Spaghetti & The Chocolate G-Strings	0.032	0.665	465000	0.696	0.000	1	0.368	-12.352	0
6894313	2AdIKzIYpkeDK5mq86vXq5	The Fight	Gypsys Gift	0.011	0.623	221564	0.557	0.000	0	0.094	-5.800	1
5683392	5OFbhYaIB6c8tPlSnybCuK	En la Palma de la Mano	Danny Frank	0.558	0.817	172386	0.282	0.000	2	0.326	-11.118	0
4376180	4jCo9S9B7X6toajqXnlLuH	El 38	Martín Patrón	0.049	0.749	158306	0.564	0.000	10	0.143	-13.426	1

In [269…
```
df_missing_100 = df_B100_songs[pd.isnull(df_B100_songs.genre_id)]
```

In [270…
```
# the majority of missing genres are also missing audio features
1 - df_missing_100['acousticness'].count() / df_missing_100['artist'].count()
```

Out[270…
```
0.6909750201450443
```

In [271…
```
# for missing genres with audio features
# 1553 artists out of 2192 total tracks
df_missing_100[~pd.isnull(df_missing_100['acousticness'])].artist.describe()
```

Out[271…
```
count         2301
unique        1619
top       R. Kelly
freq            21
Name: artist, dtype: object
```

In [81]:
```
# confirm no added rows
df_B100_songs.shape, df_10M.shape
```

Out[81]:
```
((29681, 18), (9595992, 18))
```

In [85]:
```
# INTERMEDIATE SAVE

df_B100_songs.to_parquet('df_B100_songs_STEP2COMPLETE.parquet')
df_10M.to_parquet('df_10M_STEP2COMPLETE.parquet')
```

STEP 3

get missing data from Spotify API

In [82]:
```
# how many genres are still missing?
# df_10M, df_B100, df_B100 with audio features
pd.isnull(df_10M.genre_id).sum(), pd.isnull(df_B100_songs.genre_id).sum(), pd.isnull(df_B100_songs[~pd.isnull(df_B100_songs['acousticness'])].
```

Out[82]:
```
(2959758, 7446, 2301)
```

In [84]:
```
# we can use the Spotify API to find the df_B100 with audio features but without genre
df_B100_songs[(~pd.isnull(df_B100_songs['acousticness'])) & (pd.isnull(df_B100_songs.genre_id))].head()
```

Out[84]:

	id	song	artist	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	te

	id	song	artist	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	te
1	11Aldbvo6UCcVhBzv4oUdw	What Now My Love	"Groove" Holmes	0.648	0.562	327560	0.573	0.314	7	0.056	-10.972	1	0.034	91
24	5smPCwVq1pp9b9mnhKRxQj	Music Of My Heart	'N Sync & Gloria Estefan	0.752	0.671	261975	0.292	0.006	1	0.179	-13.687	0	0.029	114
94	0VFJiOs6UOzmFCYFaTumKo	Doo Doo Brown	2 Hyped Brothers & A Dog	0.077	0.905	338533	0.383	0.000	9	0.113	-13.631	0	0.281	126
97	6byQuxXOHoos4pB8AnzuM3	Walk Tall	2 Of Clubs	0.012	0.481	148023	0.496	0.000	11	0.174	-10.215	1	0.028	139
136	1hVUDzhWm9Uok4iKvzaNLx	The Hardest Part Of Breaking Up (Is Getting Ba...	2Gether	0.106	0.830	197627	0.948	0.000	9	0.105	-3.973	0	0.045	111

Spotify API: get a temporary authorization token from: https://developer.spotify.com/console/get-search-item

In [27]:
```
# input the temporary token
TEMP_TOKEN = input('Enter token: ')

# create a spotify object
spotify = spotipy.Spotify(auth=TEMP_TOKEN)
```

```
Enter token: BQCcp69da1vgpP4jsGDOA9KyesL6HoiDXZSdCtTmiyuxyu_SVu4jttcnEGInw9I83H3E8ZcNvcmFGXLnRkT4WLckpeHBv-1eT1CyjJSMg-rDCy3EBVkP2JpmhhpCGmG6Q
ObNpgOhvTUs__kmn_XY4YRQN_yCFsZzqK_Mi9fuJe2L
```

In [29]:
```
# HELPER FUNCTIONS

def get_artist_id_by_track(track_id):
    track_info = spotify.track(track_id)
    return track_info['artists'][0]['id']

def get_genre_by_artist(artist_id):
    """this picks the most common genre, NOT the most relevant genre"""
    artist_info = spotify.artist(artist_id)
    list_of_artist_genres = artist_info['genres']

    # default to alphabetically first genre from spotify
    most_common_genre = list_of_artist_genres[0]
    if len(list_of_artist_genres) == 1:
        pass
    else:
        for genre in list_of_ordered_genres:
            if genre in list_of_artist_genres:
                return genre
    return most_common_genre

# GENRE FUNCTION

def get_genre(track_id):
    """combines 2 helper functions, makes 2 queries to Spotify API"""
    artist_id = get_artist_id_by_track(track_id)
    return get_genre_by_artist(artist_id)
```

In [216…
```
# test
track_id = '7DZsH0df0GuULl0FGwXMfd'
get_genre(track_id)
```

Out[216…
```
'soul jazz'
```

In [217…
```
# loop
how_many_passes = 0

for i, row in df_B100_songs[(~pd.isnull(df_B100_songs['acousticness'])) & (pd.isnull(df_B100_songs.genre_id))].iterrows():
    track_id = row.id

    # show status update
    if i%10 == 0:
        print(i, end='  ')
    if i%100 == 0:
```

```
        print()

    try:
        df_B100_songs['genre_id'].iloc[i] = get_genre(track_id)
    except:
        how_many_passes += 1

how_many_passes
```

```
C:\Users\Kevin\anaconda3\lib\site-packages\pandas\core\indexing.py:1732: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
150  280  890  1110  1370  1390  1430  1620  1810  1850  1920  1940  2160  2300
2390  2400
2490  2880  2930  3220  3320  3430  3450  3570  3600
3730  3740  4160  4610  4640  4650  4910  5070  5160  5200
5210  5510  5760  5860  5920  5930  6000
6020  6310  6360  6410  6430  6610  6690  6770  6860  7000
7070  7090  7200
7250  7560  7580  7660  7700
7750  7920  8480  8620  8630  8660  8690  8780  8850  8930  9070  9210  9420  9620  9690  9760  9880  10040  10350  11000
11020  11350  11400
11480  11580  11600
12210  12400
12520  12610  12670  12690  12700
12750  12850  13010  13020  13040  13100
13210  13330  13410  13950  14040  14170  14520  14700
14890  14960  15040  15050  15060  15150  15360  15690  15860  16100
16470  16590  16600
16670  16790  16890  17090  17140  17150  17210  17300
17570  17740  17970  17990  18000
18270  18690  19110  19120  19150  19200
19210  19430  19620  19860  19870  19880  20130  20190  20260  20270  20350  20360  20390  20440  20660  20670  20700
20750  20760  20780  20820  20830  20900
20910  20940  21190  21200
21280  21300
21500
21580  21710  21900
21920  22100
22430  22460  22490  22590  22600
22650  22830  22850  22870  23020  23090  23110  23370  23690  23900
23960  24000
24080  24140  24160  24420  24430  24600
24790  24820  24930  25020  25110  25150  25240  25260  25330  25520  25530  25580  25730  25990  26050  26060  26100
26180  26220  26230  26490  26510  26560  26630  26790  26970  27170  27230  27280  27320  27530  27730  28000
28160  28420  28720  28760  29100
29210  29530  29670
1354
```

Out[217…

In [218…

```
pd.isnull(df_B100_songs[~pd.isnull(df_B100_songs['acousticness'])].genre_id).sum()
```

```
1354
```

Out[218…

In [219…

```
df_B100_songs[(~pd.isnull(df_B100_songs['acousticness'])) & (pd.isnull(df_B100_songs.genre_id))].head(10)
```

Out[219…

	id	song	artist	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness
24	5smPCwVq1pp9b9mnhKRxQj	Music Of My Heart	'N Sync & Gloria Estefan	0.752	0.671	261975	0.292	0.006	1	0.179	-13.687	0	0.029
97	6byQuxXOHoos4pB8AnzuM3	Walk Tall	2 Of Clubs	0.012	0.481	148023	0.496	0.000	11	0.174	-10.215	1	0.028
155	6S4bhrjrlJhS6eQX07VqlO	This Is Ponderous	2nu	0.375	0.852	217467	0.481	0.076	9	0.160	-17.248	1	0.110
194	7sZrgx3rQTRjLHsGT8B9Yu	Have A Little Mercy	4	0.350	0.628	282827	0.575	0.000	0	0.089	-8.511	1	0.031
197	7oqvT7zwySTA2BjDVyXZFD	Stand By Me	4 The Cause	0.002	0.779	226173	0.730	0.001	6	0.255	-10.444	0	0.032
355	50jzaB4NVS3mO6rRIE7mmm	Tequila	A.L.T. And The Lost Civilization	0.006	0.782	243800	0.704	0.000	10	0.944	-10.342	0	0.158
445	0HM6keREJ3h6Hg0hhAkETW	Blues (Stay Away From Me)	Ace Cannon	0.817	0.786	135600	0.260	0.467	10	0.099	-13.510	1	0.073
446	1Mm00HcfXLI6u1f0YfNiVH	Cottonfields	Ace Cannon	0.868	0.654	112867	0.537	0.798	0	0.182	-11.292	1	0.054

	id	song	artist	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness
447	3h25PQrDynfVylukbN5ceN	Searchin'	Ace Cannon	0.759	0.783	149667	0.338	0.907	8	0.078	-13.097	1	0.033
448	6b6zxAESIJZQr7g6dHIwa3	Sugar Blues	Ace Cannon	0.656	0.677	113227	0.364	0.779	5	0.343	-9.894	1	0.035

In [220…
```
# why are these still missing? let's check a few
track_info = spotify.track('5smPCwVq1pp9b9mnhKRxQj')
artist_id = track_info['artists'][0]['id']
```

In [221…
```
artist_info = spotify.artist(artist_id)
```

In [222…
```
# no artist genre
artist_info
```

Out[222…
```
{'external_urls': {'spotify': 'https://open.spotify.com/artist/6NiolfPLVjtGrOqFxPIR3n'},
 'followers': {'href': None, 'total': 134},
 'genres': [],
 'href': 'https://api.spotify.com/v1/artists/6NiolfPLVjtGrOqFxPIR3n',
 'id': '6NiolfPLVjtGrOqFxPIR3n',
 'images': [{'height': 640,
   'url': 'https://i.scdn.co/image/ab67616d0000b273fd1bca4b865b3b8503f0621d',
   'width': 640},
  {'height': 300,
   'url': 'https://i.scdn.co/image/ab67616d00001e02fd1bca4b865b3b8503f0621d',
   'width': 300},
  {'height': 64,
   'url': 'https://i.scdn.co/image/ab67616d00004851fd1bca4b865b3b8503f0621d',
   'width': 64}],
 'name': 'A Nice Vibe',
 'popularity': 5,
 'type': 'artist',
 'uri': 'spotify:artist:6NiolfPLVjtGrOqFxPIR3n'}
```

In [223…
```
album_id = track_info['album']['id']
```

In [224…
```
album_info = spotify.album(album_id)
```

In [225…
```
# no genre for album either
album_info['genres']
```

Out[225…
```
[]
```

In [226…
```
# let's confirm with one more example
track_id = '7oqvT7zwySTA2BjDVyXZFD'

track_info = spotify.track(track_id)
artist_id = track_info['artists'][0]['id']

artist_info
```

Out[226…
```
{'external_urls': {'spotify': 'https://open.spotify.com/artist/6NiolfPLVjtGrOqFxPIR3n'},
 'followers': {'href': None, 'total': 134},
 'genres': [],
 'href': 'https://api.spotify.com/v1/artists/6NiolfPLVjtGrOqFxPIR3n',
 'id': '6NiolfPLVjtGrOqFxPIR3n',
 'images': [{'height': 640,
   'url': 'https://i.scdn.co/image/ab67616d0000b273fd1bca4b865b3b8503f0621d',
   'width': 640},
  {'height': 300,
   'url': 'https://i.scdn.co/image/ab67616d00001e02fd1bca4b865b3b8503f0621d',
   'width': 300},
  {'height': 64,
   'url': 'https://i.scdn.co/image/ab67616d00004851fd1bca4b865b3b8503f0621d',
   'width': 64}],
 'name': 'A Nice Vibe',
 'popularity': 5,
 'type': 'artist',
 'uri': 'spotify:artist:6NiolfPLVjtGrOqFxPIR3n'}
```

In [227…
```
album_id = track_info['album']['id']
album_info = spotify.album(album_id)
```

```
album_info['genres']
# CONCLUSION: these tracks do not seem to have any genre data
```

Out[227…
```
[]
```

In [277…
```
# confirm no added rows
df_B100_songs.shape, df_10M.shape
```

Out[277…
```
((29681, 18), (9601658, 18))
```

In [279…
```
# confirm no added rows
df_B100_songs.shape, df_10M.shape
```

Out[279…
```
((29681, 18), (9601658, 18))
```

In [280…
```
df_B100_songs.head(10)
```

Out[280…

	id	song	artist	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	te
0	7DZsH0df0GuULl0FGwXMfd	Misty	"Groove" Holmes	0.806	0.581	360986	0.556	0.925	8	0.218	-10.422	1	0.054	9
1	11Aldbvo6UCcVhBzv4oUdw	What Now My Love	"Groove" Holmes	0.648	0.562	327560	0.573	0.314	7	0.056	-10.972	1	0.034	9
2	4KRLWRl1bFjnXhY5MgZWrM	May The Bird Of Paradise Fly Up Your Nose	"Little" Jimmy Dickens	0.738	0.660	151693	0.801	0.000	4	0.627	-8.446	1	0.115	10
3	None	I Know I Know	"Pookie" Hudson	NaN	NaN	<NA>	NaN	NaN	<NA>	NaN	NaN	<NA>	NaN	
4	5r96TaQquRrlo3Ym3ZlSL2	Amish Paradise	"Weird Al" Yankovic	0.103	0.728	202920	0.448	0.000	8	0.267	-10.540	1	0.172	8
5	1gloYGAZI6eHp6MEPjLuL3	Canadian Idiot	"Weird Al" Yankovic	0.002	0.543	143040	0.697	0.000	8	0.343	-9.211	1	0.061	18
6	0WuYuWhLws8VahMy2zLLRJ	Eat It	"Weird Al" Yankovic	0.087	0.767	200627	0.811	0.000	7	0.068	-8.548	1	0.077	14
7	4is3oF4FlWmedh3TK6Ke7z	Fat	"Weird Al" Yankovic	0.166	0.870	216066	0.551	0.000	6	0.064	-10.563	0	0.060	11
8	022XE9RZVU3a9nULENXfnm	I Lost On Jeopardy	"Weird Al" Yankovic	0.347	0.887	208933	0.769	0.000	0	0.051	-9.150	0	0.050	11
9	0uTSpsec5kXkNQQjzt6dIB	King Of Suede	"Weird Al" Yankovic	0.271	0.780	255373	0.808	0.000	11	0.096	-10.056	0	0.038	12

In [229…
```
# INTERMEDIATE SAVE

df_B100_songs.to_parquet('df_B100_songs_STEP3COMPLETE.parquet')
# df_10M.to_parquet('df_10M_STEP3COMPLETE.parquet')  # df_10M is not involved in step3
```

OPTIONAL: group genre_id into broader categories for genre

ie, treat genre_id as a subgenre

In [148…
```
genre_set = set(df_10M.genre_id) | set(df_B100_songs.genre_id)
genre_set.remove(np.nan)
```

In [149…
```
len(genre_set)
```

Out[149…
```
5447
```

In [150…
```
import random
random.sample(list(genre_set), 10)
# these categories are insane
# we are going to need better genres than spotify can provide
# otherwise clustering won't work well at all
```

Out[150…
```
['indie triste',
 'hoerspiel',
 'technical thrash',
 'dutch edm',
 'french psychedelic',
 'funk melody',
 'australian hardcore',
 'faroese pop',
 'eurovision',
 'jangle pop']
```

not sure how to group these into supergenres....

MAYBE ANOTHER REFERENCE: https://www.kaylinpavlik.com/classifying-songs-genres/ Could group manually based on word matches: https://en.wikipedia.org/wiki/Music_genre

forget about this for now, could be a lot of work and may not improve anything

reorder dataframes and resave them

In [87]:
```
df_10M = df_10M.rename({'genre_id':'genre'}, axis=1)
df_B100_songs = df_B100_songs.rename({'genre_id':'genre'}, axis=1)

desired_formatting = [
    'id', 'song', 'artist', 'genre', 'release_date',
    'acousticness', 'danceability', 'duration_ms', 'energy',
    'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
    'speechiness', 'tempo', 'time_signature', 'valence'
]

# reorder
df_10M = df_10M[desired_formatting]
df_B100_songs = df_B100_songs[desired_formatting]

# save
df_B100_songs.to_parquet('df_B100_songs.parquet')
df_10M.to_parquet('df_10M.parquet')
```

In [89]:
```
# check shape (good)
df_B100_songs.shape, df_10M.shape
```

Out[89]:
```
((29681, 18), (9595992, 18))
```

merge df_B100_songs into df_B100

In [116…
```
dict_genres = dict(zip(df_B100_songs.id, df_B100_songs.genre))

def lookup_genre(row):
    song_id = row['id']
    return dict_genres[song_id]

df_B100['genre'] = df_B100.apply(lookup_genre, axis=1)
```

In [119…
```
# reorder dataframe, consistent with other df

desired_formatting_timeseries = [
    'date', 'last-week', 'peak-rank', 'weeks-on-board',
    'id', 'song', 'artist', 'genre',
    'acousticness', 'danceability', 'duration_ms', 'energy',
    'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
    'speechiness', 'tempo', 'time_signature', 'valence'
]

# df_B100 =
df_B100[desired_formatting_timeseries]
```

Out[119…

	date	last-week	peak-rank	weeks-on-board	id	song	artist	genre	acousticness	danceability	duration_ms	energy	instrumentalness	k

	date	last-week	peak-rank	weeks-on-board	id	song	artist	genre	acousticness	danceability	duration_ms	energy	instrumentalness	l
0	1958-08-04	<NA>	31	1	07GtDOCxmye5KDWsTSACPk	Chantilly Lace	Big Bopper	rock-and-roll	0.841	0.489	145266	0.806	0.000	
1	1958-08-04	<NA>	82	1	328wGzwVquTqX5m3t1czL0	Blip Blop	Bill Doggett	soul jazz	0.461	0.705	166826	0.729	0.469	
2	1958-08-04	<NA>	99	1	None	I'll Get By (As Long As I Have You)	Billy Williams	NaN	NaN	NaN	<NA>	NaN	NaN	<N
3	1958-08-04	<NA>	3	1	40fD7ct05FvQHLdQTgJelG	Splish Splash	Bobby Darin	adult standards	0.385	0.645	131719	0.943	0.000	
4	1958-08-04	<NA>	60	1	3ixHQiAUk6F6ZU1tipromq	Over And Over	Bobby Day	rock-and-roll	0.697	0.638	143320	0.550	0.000	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
329925	2021-11-06	12	9	17	None	Essence	Wizkid Featuring Justin Bieber & Tems	NaN	NaN	NaN	<NA>	NaN	NaN	<N
329926	2021-11-06	20	20	2	4KDNRh9Oor80z3XlxdWlui	Bubbly	Young Thug With Drake & Travis Scott	NaN	0.054	0.910	165067	0.585	0.000	
329927	2021-11-06	<NA>	58	4	2cEnKRR4dYBB2VA1mjlb1z	Nevada	YoungBoy Never Broke Again	trap	0.397	0.777	155738	0.590	0.000	
329928	2021-11-06	59	56	13	None	Baddest	Yung Bleu, Chris Brown & 2 Chainz	NaN	NaN	NaN	<NA>	NaN	NaN	<N
329929	2021-11-06	55	53	6	4HD9SLK4s9rwRyuFt3n8N7	Same Boat	Zac Brown Band	country	0.052	0.486	190452	0.838	0.000	

329930 rows × 21 columns

In [126…
```
# check null values (<NA> and nan)
# they look fine, but may want to revisit later or impute for consistency

df_B100.loc[2].duration_ms, pd.isnull(df_B100.loc[2].duration_ms)
```

Out[126…
```
(<NA>, True)
```

In [127…
```
df_B100.loc[2].danceability, pd.isnull(df_B100.loc[2].danceability)
```

Out[127…
```
(nan, True)
```

In [131…
```
df_B100['last-week'].loc[0], pd.isnull(df_B100['last-week'].loc[0])
```

Out[131…
```
(<NA>, True)
```

In [ ]:
```
# https://stackoverflow.com/questions/60115806/pd-na-vs-np-nan-for-pandas
# nan and <NA> are different, not sure if it's ok to have both, or if I should impute...
```

In [132…
```
df_B100.to_parquet('df_B100.parquet')
```

In [ ]:
```

```

Attachment 3

DATA

Import

Import Modules

In [1]:
```
# import modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import time
import seaborn as sns
sns.set_theme()

# jupyter notebook full-width display
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))

# pandas formatting
pd.set_option('display.float_format', '{:.3f}'.format)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 200)
```

Import Data

from: Kevin 820 Data Optimisation.ipynb

In [2]:
```
# all songs with audio features (combined from 3 sources)
df_10M = pd.read_parquet('df_10M.parquet')

# all Billboard 100 lists, audio features included where possible
df_B100 = pd.read_parquet('df_B100.parquet')

# all unique songs from the Billboard 100 lists, audio features included where possible
df_B100_songs = pd.read_parquet('df_B100_songs.parquet')

# all unique songs from the Billboard 100 lists, only songs with audio features included
df_B100_songs_AF = df_B100_songs.dropna().copy().reset_index(drop=True)
```

In [3]:
```
# Popularity (as defined as "has been on the Billboard Hot 100" list)
# now combine, categorise, and correlate datasets

set_B100_id = set(df_B100_songs_AF.id)

temp_not_popular = df_10M[~df_10M.id.isin(set_B100_id)].copy().reset_index(drop=True)
temp_not_popular['POPULAR'] = False

temp_popular = df_B100_songs_AF.copy()
temp_popular['POPULAR'] = True

df_popularity = pd.concat([temp_not_popular, temp_popular]).reset_index(drop=True)

del temp_not_popular
del temp_popular
```

In [4]:
```
# import genre count data
df_genre_counts = pd.read_parquet('df_genre_counts.parquet')
```

Data Description

Data Sources

The Billboard 100

https://en.wikipedia.org/wiki/Billboard_Hot_100

https://www.kaggle.com/datasets/dhruvildave/billboard-the-hot-100-songs

1.2M Songs with Metadata (csv)

https://www.kaggle.com/datasets/rodolfofigueroa/spotify-12m-songs

8+ M. Spotify Tracks, Genre, Audio Features (SQL)

https://www.kaggle.com/datasets/maltegrosse/8-m-spotify-tracks-genre-audio-features

Spotify API

https://developer.spotify.com/documentation/web-api/

https://developer.spotify.com/console/get-search-item

https://developer.spotify.com/console/get-audio-features-track/

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

Spotipy Library: https://spotipy.readthedocs.io/en/master/

Data Description and Discussion

- The Billboard 100 data did not include audio features. It was combined with audio features from the following sources:
  - 1.2M Songs with Metadata (csv format)
  - 8+ M. Spotify Tracks, Genre, Audio Features (SQLite format)
  - Spotify API data gathered via the library Spotipy
- Overall, audio features was gathered for approximately 75% of songs from the Billboard 100.
  - Some songs were excluded based on data repetition issues
    - Typically this was only hard to find songs with very similar names
    - For example searching for 'Metallica The Unforgiven' and 'Metallica The Unforgiven Part 2' yielded the same Spotify id
      - It was determined that excluding these songs was less error-prone than manually fixing the issues
      - Alternatively, we could have kept 1 song. In this case, there is up to a 50% chance that the song is mislabelled, so this option appeared less favourable than dropping both repeat instances.
- A Quality Assurance (QA) check was performed on the final dataset.
  - Audio features from 100 songs were gathered from the Spotify API and compared to the datasets listed above.
  - There were 3 non-trivial issues noted in 2 of the 100 songs:
    - Madonna Live To Tell
      - A significant increase in loudness (~7 dB)
      - Approximately 1 second difference in length
      - All other audio features consistent between data sources
      - Both of these changes appear to result from remastering and re-uploading the track
        - https://artists.spotify.com/help/article/re-uploading-music
    - Lil Wayne Let It All Work Out
      - The key signature was not consistent between the 2 sources
      - The newer source (the API request from Sept 11, 2022) was correct (B major)
      - The SQL database was also different
      - My supposition is that these errors are due to the characteristics of the song:
        - atonal (most notably the singing)
        - detuned (bass pitch automation, and low-fi detuning effects)
  - Overall, there is a large degree of consistency between datasets. Furthermore, inconsistencies are all explainable with reasonable suppositions.

Spotify API Audio Feature Descriptions

from: https://developer.spotify.com/documentation/web-api/reference/#/operations/get-several-audio-features

acousticness

number \

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
```

analysis_url

string

A URL to access the full audio analysis of this track. An access token is required to access this data.

danceability

number \

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. A value of 0.0 is least danceable and 1.0 is most danceable.

duration_ms

integer

The duration of the track in milliseconds.

energy

number \

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.

id

string

The Spotify ID for the track.

instrumentalness

number \

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.

key

integer 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
```

liveness

number \

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.

loudness

number \

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). Values typically range between -60 and 0 db.

mode

integer

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

speechiness

number \

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.

tempo

number \

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.

time_signature

integer

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
```

track_href

string

A link to the Web API endpoint providing full details of the track.

type

string

The object type.

Allowed value: "audio_features"

uri

string

The Spotify URI for the track.

valence

number \

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
```

Descriptive Statistics and Data Features

In [5]:
```
# sizes of the datasets
df_10M.shape, df_B100.shape, df_B100_songs.shape, df_B100_songs_AF.shape
```

Out[5]:
```
((9595992, 18), (329930, 22), (29681, 18), (19888, 18))
```

In [6]:
```
# are any keys unknown (key == -1)
sorted(df_B100_songs_AF['key'].unique()), sorted(df_10M['key'].unique())
# no
```

Out[6]:
```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])
```

In [7]:
```
# data types
pd.concat(
    [df_10M.dtypes, df_B100.dtypes, df_B100_songs.dtypes],
    keys=['df_10M.dtypes', 'df_B100.dtypes', 'df_B100_songs.dtypes'],
    axis=1
)
```

Out[7]:

	df_10M.dtypes	df_B100.dtypes	df_B100_songs.dtypes
id	object	object	object
song	object	object	object
artist	object	object	object
genre	object	object	object
release_date	datetime64[ns]	NaN	datetime64[ns]
acousticness	float32	float32	float32
danceability	float32	float32	float32
duration_ms	Int64	Int64	Int64

	df_10M.dtypes	df_B100.dtypes	df_B100_songs.dtypes
energy	float32	float32	float32
instrumentalness	float32	float32	float32
key	Int16	Int16	Int16
liveness	float32	float32	float32
loudness	float32	float32	float32
mode	Int16	Int16	Int16
speechiness	float32	float32	float32
tempo	float32	float32	float32
time_signature	Int16	Int16	Int16
valence	float32	float32	float32
date	NaN	datetime64[ns]	NaN
rank	NaN	Int16	NaN
last-week	NaN	Int16	NaN
peak-rank	NaN	Int16	NaN
weeks-on-board	NaN	Int16	NaN

In [8]:
```
# Date Range for Billboard Hot 100

df_B100.date.min(), df_B100.date.max()
```

Out[8]:
```
(Timestamp('1958-08-04 00:00:00'), Timestamp('2021-11-06 00:00:00'))
```

In [10]:
```
df_10M.describe().loc['mean':'max'].T
```

Out[10]:

	mean	std	min	25%	50%	75%	max
acousticness	0.421	0.374	0.000	0.034	0.336	0.817	0.996
danceability	0.528	0.190	0.000	0.396	0.545	0.676	1.000
duration_ms	238209.591	159341.591	0.000	169600.000	216933.000	275080.000	19672058.000
energy	0.545	0.282	0.000	0.310	0.567	0.789	1.000
instrumentalness	0.258	0.374	0.000	0.000	0.002	0.645	1.000
key	5.237	3.542	0.000	2.000	5.000	8.000	11.000
liveness	0.210	0.180	0.000	0.096	0.129	0.262	1.000
loudness	-10.967	6.318	-60.000	-13.675	-9.196	-6.398	7.234
mode	0.661	0.473	0.000	0.000	1.000	1.000	1.000
speechiness	0.098	0.135	0.000	0.036	0.047	0.082	0.974
tempo	118.785	30.832	0.000	95.080	118.950	137.450	249.987
time_signature	3.840	0.567	0.000	4.000	4.000	4.000	5.000
valence	0.474	0.278	0.000	0.234	0.468	0.708	1.000

In [11]:
```
df_B100.describe().loc['mean':'max'].T
```

Out[11]:

	mean	std	min	25%	50%	75%	max
rank	50.502	28.866	1.000	26.000	51.000	76.000	100.000
last-week	47.593	28.055	1.000	23.000	47.000	72.000	100.000
peak-rank	40.973	29.348	1.000	13.000	38.000	65.000	100.000
weeks-on-board	9.162	7.619	1.000	4.000	7.000	13.000	90.000
acousticness	0.278	0.275	0.000	0.041	0.178	0.468	0.995
danceability	0.603	0.149	0.000	0.507	0.611	0.708	0.988
duration_ms	226879.648	66552.146	30213.000	183360.000	221306.000	258399.000	1561133.000
energy	0.625	0.199	0.007	0.479	0.643	0.786	0.999
instrumentalness	0.034	0.139	0.000	0.000	0.000	0.001	0.985

	mean	std	min	25%	50%	75%	max
key	5.218	3.564	0.000	2.000	5.000	8.000	11.000
liveness	0.190	0.163	0.012	0.089	0.128	0.243	0.999
loudness	-8.609	3.585	-30.346	-10.972	-8.153	-5.790	2.291
mode	0.731	0.443	0.000	0.000	1.000	1.000	1.000
speechiness	0.064	0.071	0.000	0.032	0.040	0.060	0.951
tempo	120.403	27.788	0.000	99.935	119.002	136.003	241.009
time_signature	3.942	0.297	0.000	4.000	4.000	4.000	5.000
valence	0.607	0.238	0.000	0.423	0.629	0.807	0.991

In [7]:
```
df_B100_songs.describe().loc['mean':'max'].T
```

Out[7]:

	mean	std	min	25%	50%	75%	max
acousticness	0.316	0.290	0.000	0.052	0.224	0.556	0.995
danceability	0.590	0.152	0.000	0.491	0.598	0.697	0.988
duration_ms	217638.340	68403.261	30213.000	169533.000	210426.000	251333.000	1561133.000
energy	0.611	0.203	0.007	0.463	0.624	0.775	0.999
instrumentalness	0.039	0.151	0.000	0.000	0.000	0.001	0.985
key	5.196	3.556	0.000	2.000	5.000	8.000	11.000
liveness	0.197	0.168	0.012	0.091	0.132	0.255	0.999
loudness	-8.927	3.622	-30.346	-11.314	-8.554	-6.111	2.291
mode	0.741	0.438	0.000	0.000	1.000	1.000	1.000
speechiness	0.067	0.076	0.000	0.032	0.040	0.061	0.951
tempo	120.482	28.092	0.000	99.790	119.067	136.345	241.009
time_signature	3.925	0.333	0.000	4.000	4.000	4.000	5.000
valence	0.611	0.240	0.000	0.425	0.637	0.814	0.991

In [8]:
```
df_B100_songs_AF.describe().loc['mean':'max'].T
```

Out[8]:

	mean	std	min	25%	50%	75%	max
acousticness	0.307	0.287	0.000	0.049	0.211	0.537	0.995
danceability	0.590	0.151	0.000	0.492	0.598	0.697	0.988
duration_ms	219696.491	68100.808	30213.000	172302.750	212526.500	252883.250	1561133.000
energy	0.613	0.203	0.018	0.466	0.627	0.777	0.997
instrumentalness	0.034	0.139	0.000	0.000	0.000	0.001	0.985
key	5.203	3.558	0.000	2.000	5.000	8.000	11.000
liveness	0.196	0.167	0.012	0.091	0.132	0.253	0.999
loudness	-8.858	3.612	-29.658	-11.253	-8.463	-6.028	-0.463
mode	0.739	0.439	0.000	0.000	1.000	1.000	1.000
speechiness	0.067	0.075	0.000	0.032	0.040	0.062	0.941
tempo	120.529	28.047	0.000	99.739	119.061	136.793	231.028
time_signature	3.928	0.326	0.000	4.000	4.000	4.000	5.000
valence	0.606	0.240	0.000	0.419	0.629	0.808	0.989

In [13]:
```
# Genre Counts

# number of Billboard Hot 100 songs with and without audio features / genre
df_B100_songs_AF.genre.count(), df_B100_songs.shape[0]
```

Out[13]:
```
(19888, 29681)
```

In [14]:
```
# percentage of songs with audio features that also have genre data
# NOTE: not all songs on Spotify have an associated genre
```

```
df_B100_songs_AF.genre.count() / df_B100_songs.valence.count()
```

Out[14]:
```
0.896299968452837
```

In [15]:
```
# number of total songs which include audio features with and without genre
df_10M.genre.count(), df_10M.shape[0], df_10M.genre.count() / df_10M.shape[0]
```

Out[15]:
```
(6636234, 9595992, 0.6915631025953336)
```

Proportion of Songs With Audio Feature Data:

~75% of songs on the Billboard list are available on Spotify, and weren't removed for data errors

In [16]:
```
# All Billboard 100 lists
# number not null, total, proportion not null
(
    df_B100[df_B100.id.notnull()].shape[0],
    df_B100.shape[0],
    df_B100[df_B100.id.notnull()].shape[0] / df_B100.shape[0]
)
```

Out[16]:
```
(253254, 329930, 0.7675991877064832)
```

In [17]:
```
# All songs from Billboard 100 lists
# number not null, total, proportion not null
(
    df_B100_songs[df_B100_songs.id.notnull()].shape[0],
    df_B100_songs.shape[0],
    df_B100_songs[df_B100_songs.id.notnull()].shape[0] / df_B100_songs.shape[0]
)
```

Out[17]:
```
(22189, 29681, 0.7475826286176341)
```

In [18]:
```
df_B100_songs_AF.columns
```

Out[18]:
```
Index(['id', 'song', 'artist', 'genre', 'release_date', 'acousticness',
       'danceability', 'duration_ms', 'energy', 'instrumentalness', 'key',
       'liveness', 'loudness', 'mode', 'speechiness', 'tempo',
       'time_signature', 'valence'],
      dtype='object')
```

EXPLORATORY DATA ANALYSIS

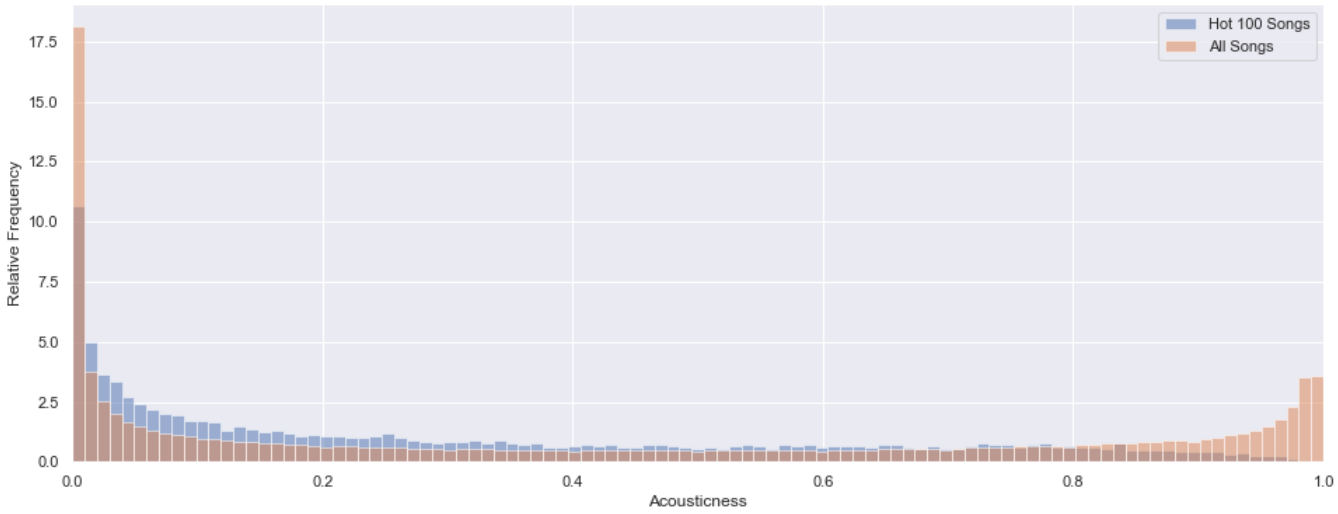
Histograms

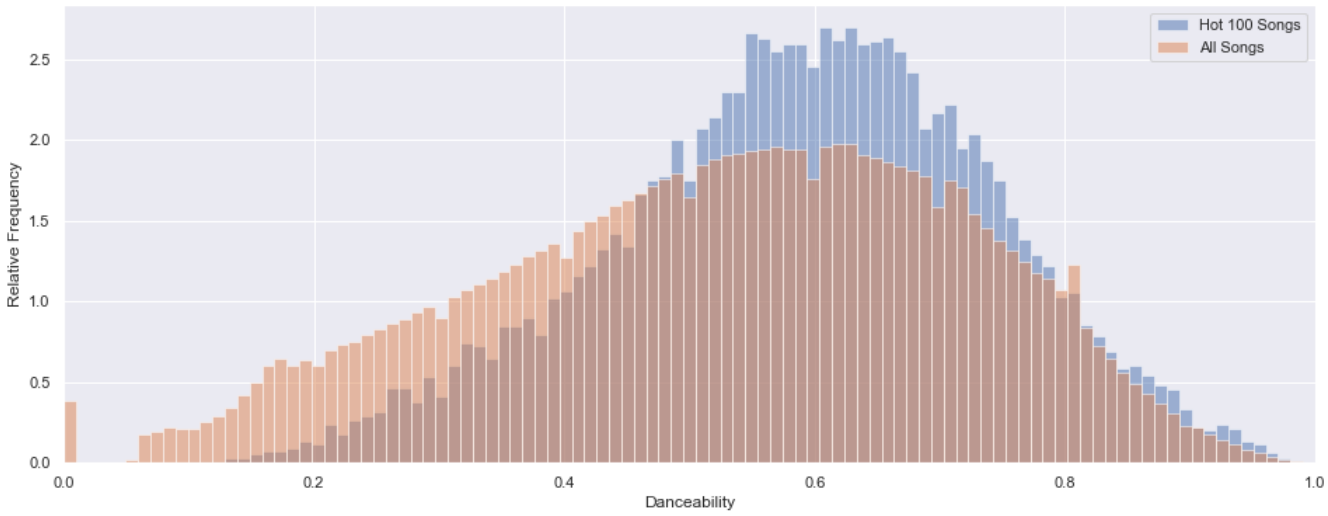
In [19]:
```
# main features
features = ['acousticness', 'danceability', 'energy', 'instrumentalness',
            'liveness', 'loudness', 'speechiness', 'tempo', 'valence']

def compare_histograms(feature, bins=50, logy=False, figsize=(16, 6)):

    plt.figure(figsize=figsize)

    # bins don't line up unless you do this
    if feature == 'loudness':
        xmin, xmax = -16, 0
        plt.xlim(xmin, xmax)  # reasonable range for loudness in music
        bins_plot = [(x/(bins+1))*(xmax-xmin)+xmin for x in range(bins+2)]
    elif feature == 'tempo':
        xmin, xmax = 40, 200
        plt.xlim(xmin, xmax)  # reasonable range for tempo in music
        bins_plot = [(x/(bins+1))*(xmax-xmin)+xmin for x in range(bins+2)]
    else:
        xmin, xmax = 0, 1
        plt.xlim(xmin, xmax)  # most audio features vary between 0 and 1
        bins_plot = [(x/(bins+1))*(xmax-xmin)+xmin for x in range(bins+2)]
        # NOTE: mode is an int, either 0 or 1 (minor/major),
        # but the average shows how commonly a song is in either mode

    # plots
    plt.hist(df_B100_songs_AF[feature], bins_plot, alpha=0.5, label='Hot 100 Songs', density=True)
    plt.hist(df_10M[feature], bins_plot, alpha=0.5, label='All Songs', density=True)

    title = f'{feature.title()} Histogram'
#     plt.title(title)
    plt.xlabel(feature.title().replace('_', ' '))
```

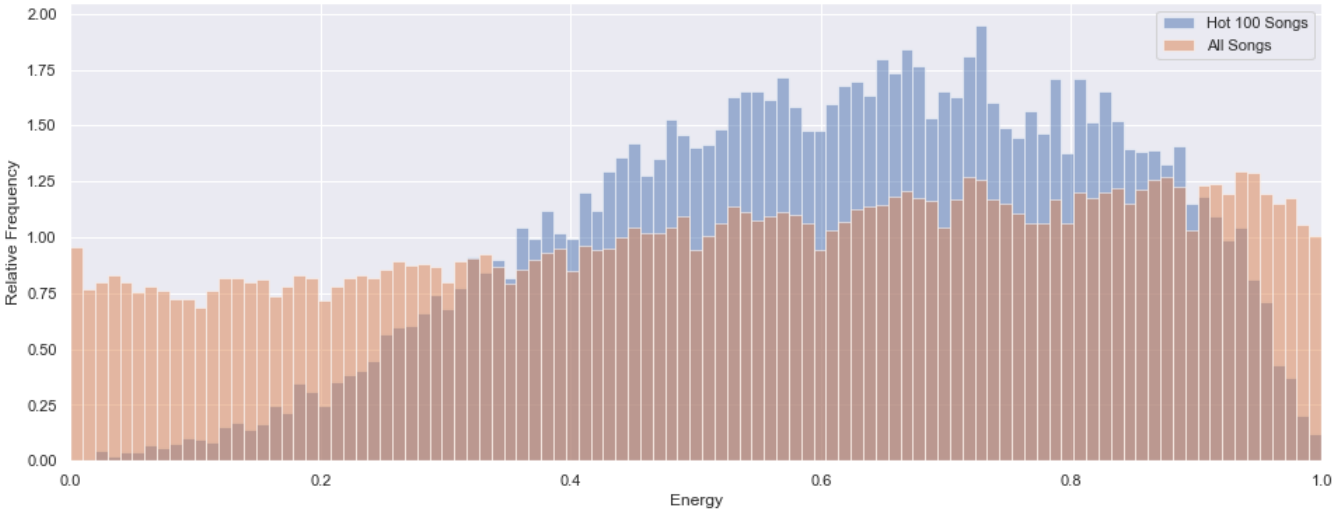
```
    plt.legend(loc='upper right')
    if logy:
        plt.yscale('log')
        plt.ylabel('Log Relative Frequency')
    else:
        plt.ylabel('Relative Frequency')

    # save the image
    plt.savefig(f'figures/histograms/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')

    plt.show()
```

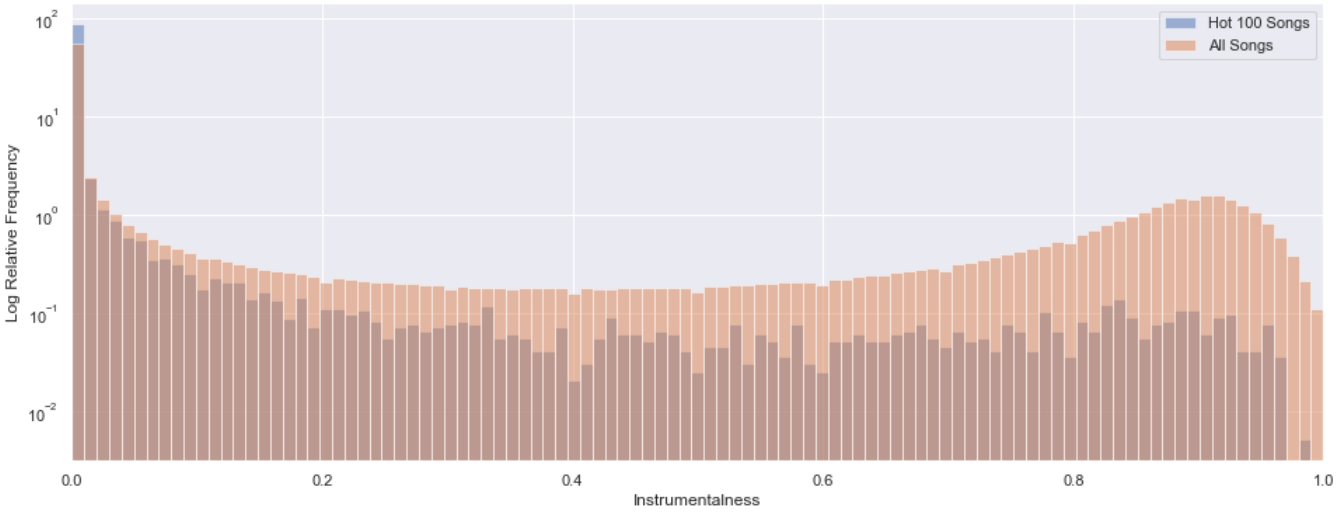
In [20]:

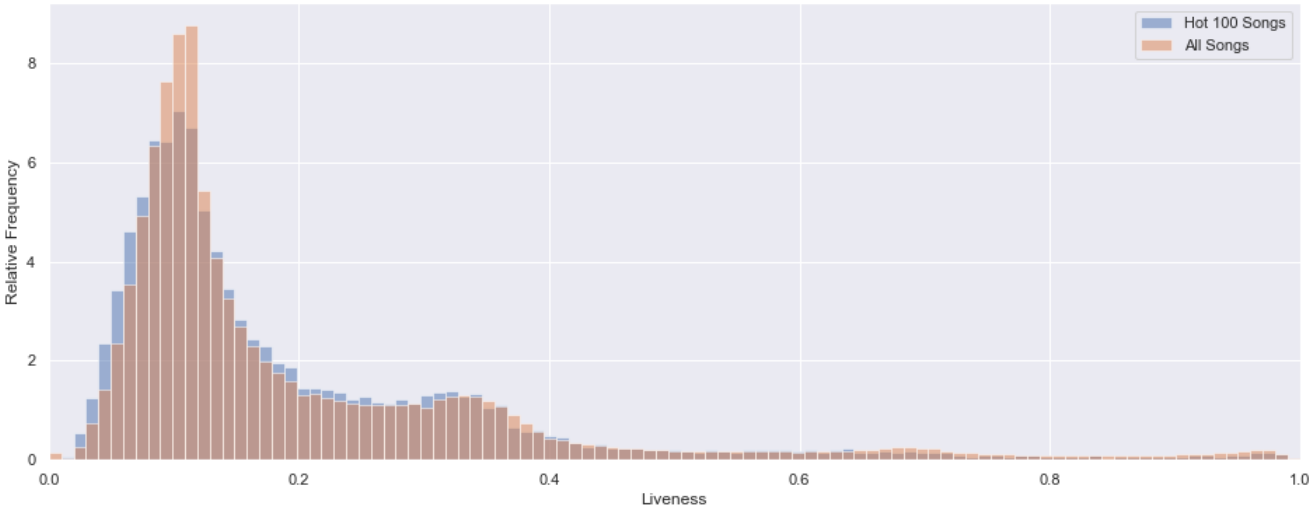
```
# Histograms of Features
for feature in features:
    compare_histograms(feature, 100, logy=(feature in ['duration_ms', 'instrumentalness', 'speechiness']))
```

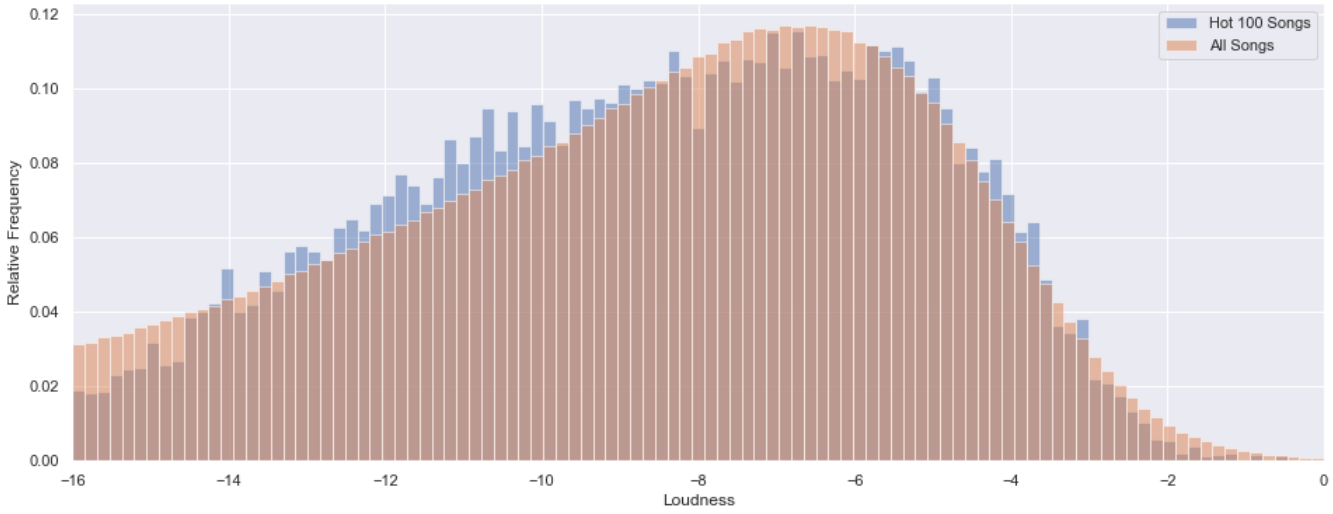


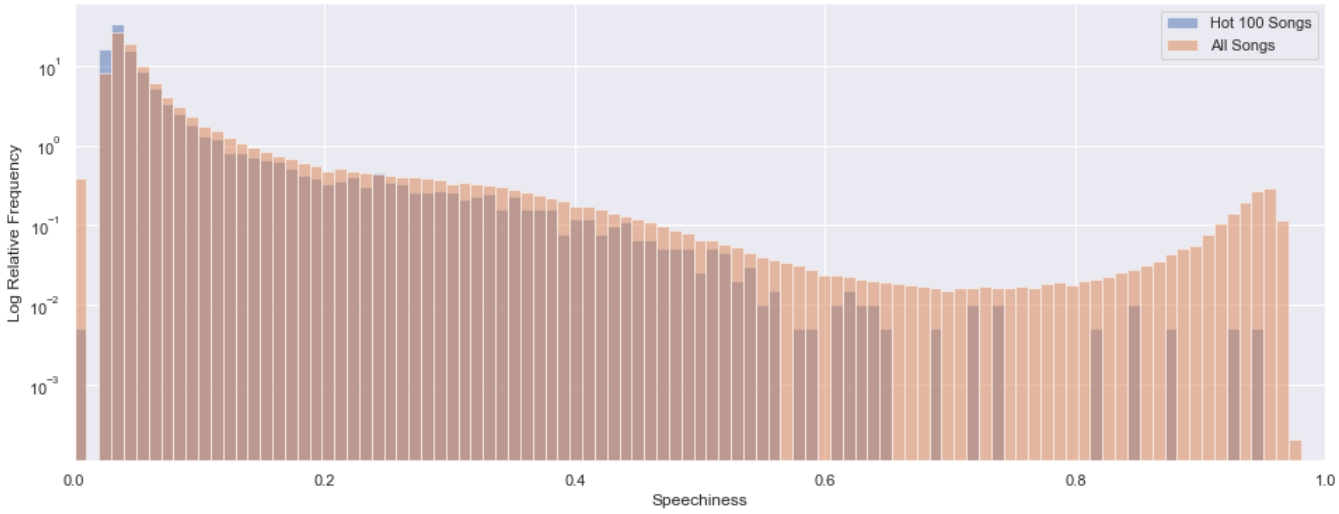


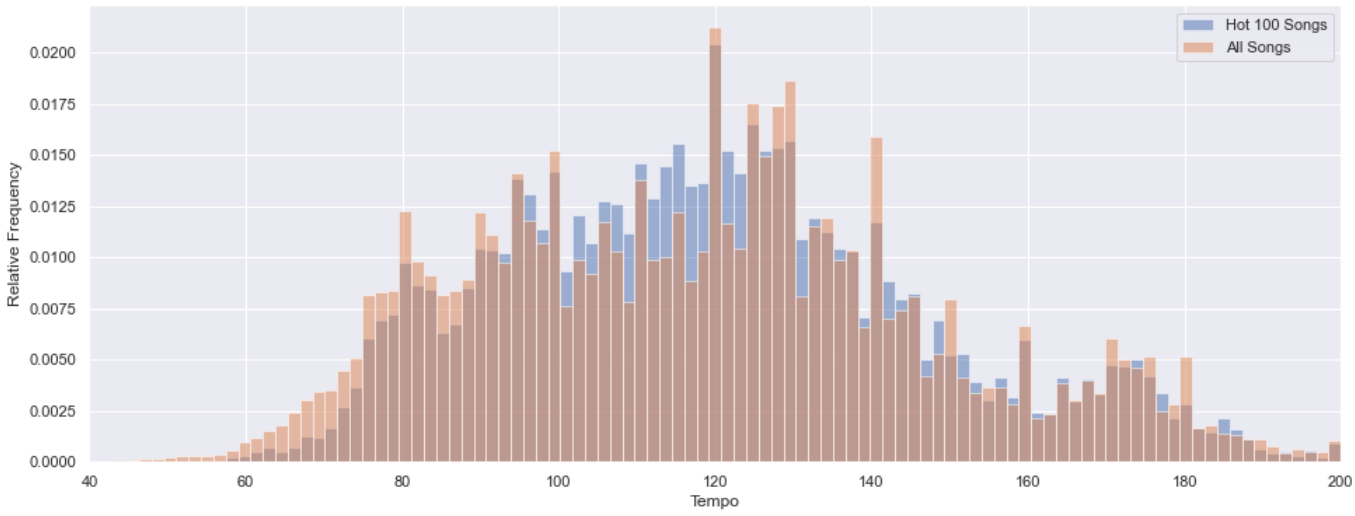

Hot 100 Songs
All Songs
Relative Frequency
Energy

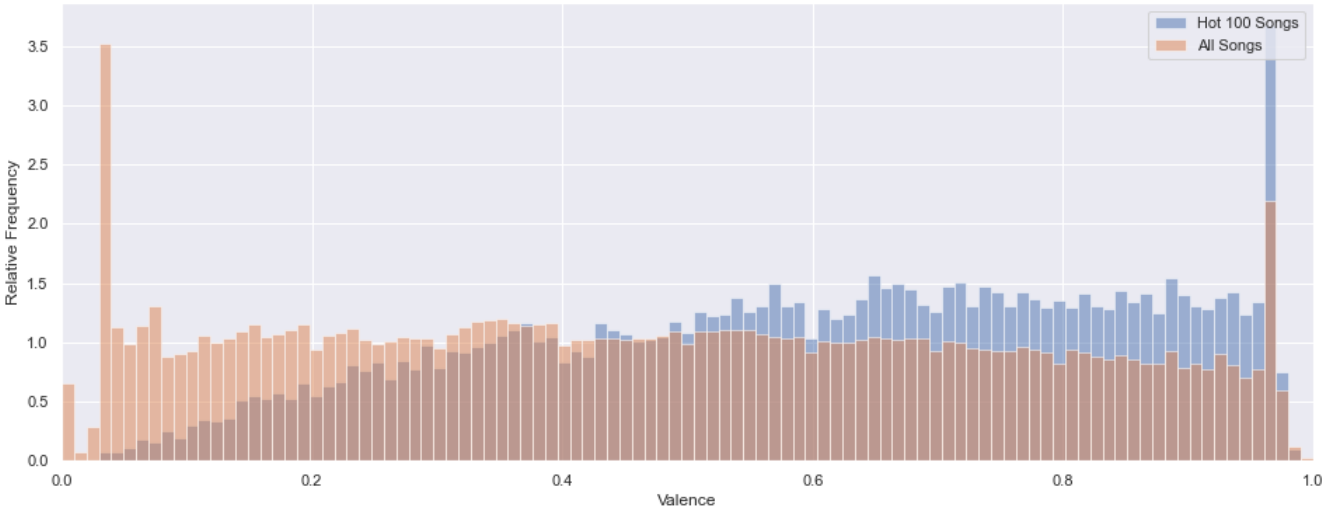

Hot 100 Songs
All Songs
Log Relative Frequency
Instrumentalness


Hot 100 Songs
All Songs
Relative Frequency
Liveness


Hot 100 Songs
All Songs
Relative Frequency
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−16
−14
−12
−10
−8
−6
−4
−2
0
Loudness


Hot 100 Songs
All Songs
Log Relative Frequency
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
0.0
0.2
0.4
0.6
0.8
1.0
Speechiness


Hot 100 Songs
All Songs
Relative Frequency
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
40
60
80
100
120
140
160
180
200
Tempo

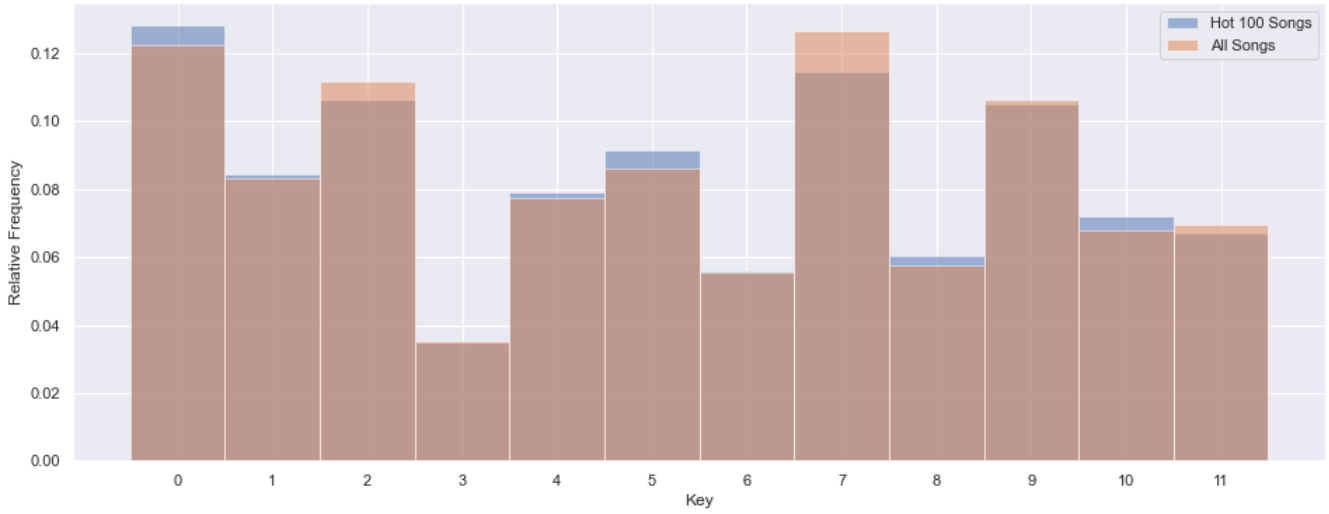


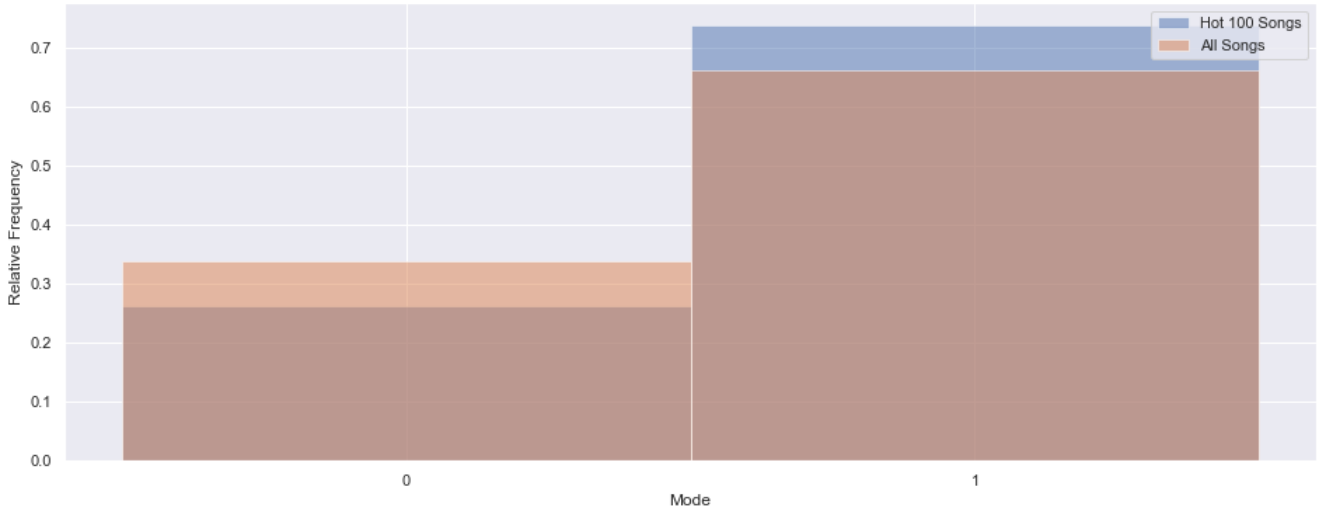
In [21]:

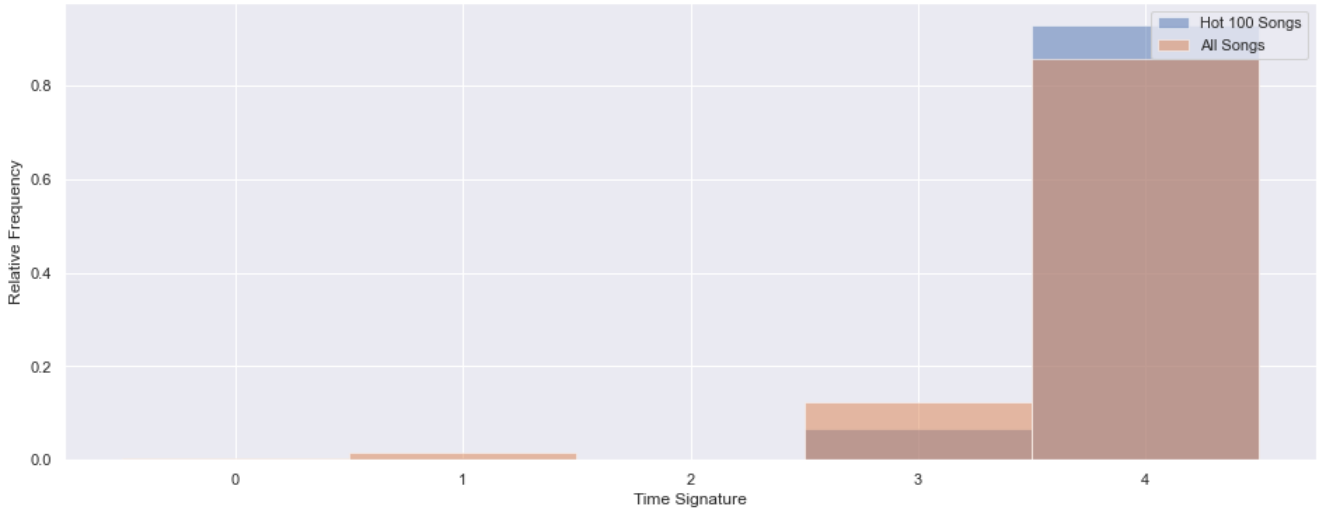
```
# features requiring specific bin sizes
other_features = ['key', 'mode', 'time_signature']

# Histograms of Features with Specific Bin Sizes
other_bins = {'key': 12, 'mode': 2, 'time_signature': 5}

def compare_histograms_discrete(feature, bins=50, logy=False, figsize=(16, 6)):

    fig, ax = plt.subplots(figsize=figsize)

    # use range(bins+1) to offset tick labels and line up with appropriate labels
    plt.hist(df_B100_songs_AF[feature], [x for x in range(bins+1)], alpha=0.5, label='Hot 100 Songs', density=True, align='left')
    plt.hist(df_10M[feature], [x for x in range(bins+1)], alpha=0.5, label='All Songs', density=True, align='left')

    title = f'{feature.title()} Histogram'
#     plt.title(title)
    plt.xlabel(feature.title().replace('_', ' '))
    plt.legend(loc='upper right')
    if logy:
        plt.yscale('log')
        plt.ylabel('Log Relative Frequency')
    else:
        plt.ylabel('Relative Frequency')

    ax.set(xticks=[x for x in range(bins)])

    # save the image
    plt.savefig(f'figures/histograms/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')

    plt.show()
```

In [22]:

```
# Histograms of Features with Specific Bin Sizes

# Histograms of Features
for feature in other_features:
    compare_histograms_discrete(feature, other_bins[feature])
```



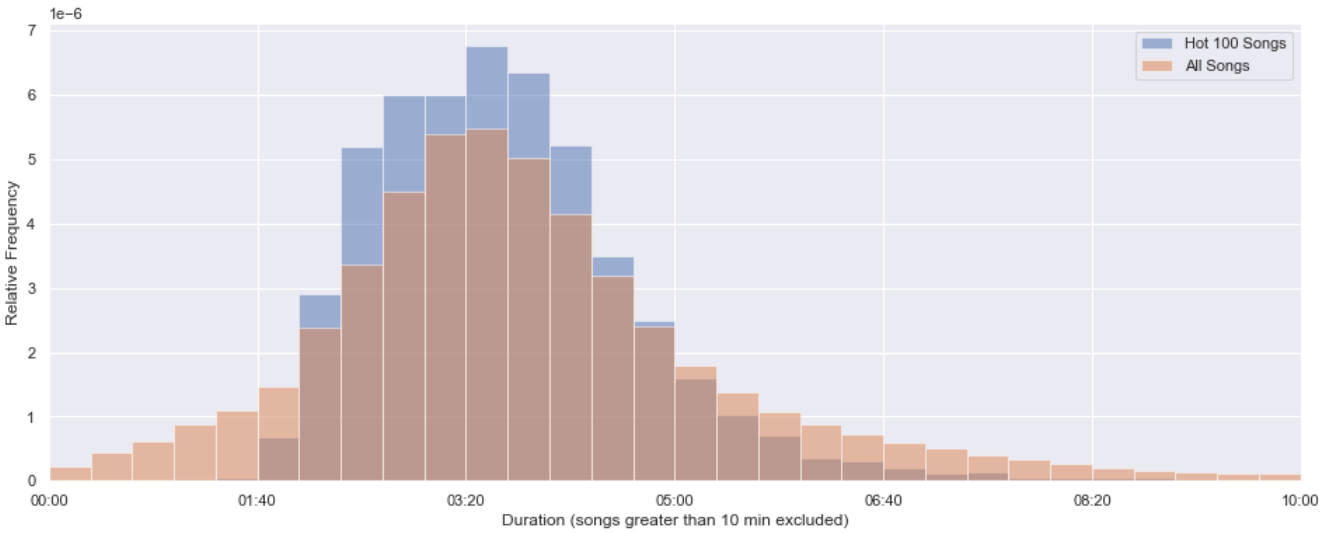




In [23]:

```
# duration_ms needs to share the x-axis

fig, ax = plt.subplots(figsize=(16, 6))

bins = 1000
maxx = df_10M['duration_ms'].max()
maxx = round(maxx, ndigits=-7)
increment = maxx / bins  # need to do this first or get integer wrap around error
bins = [x for x in range(bins) * increment]

plt.hist(df_B100_songs_AF['duration_ms'], bins, alpha=0.5, label='Hot 100 Songs', density=True)
plt.hist(df_10M['duration_ms'], bins, alpha=0.5, label='All Songs', density=True)

title = 'Duration Histogram'
```

```
# plt.title(title)
plt.xlabel('Duration (songs greater than 10 min excluded)')
plt.legend(loc='upper right')
plt.ylabel('Relative Frequency')
ax.set_xlim((0, 600000))

# https://stackoverflow.com/questions/40395227/minute-and-second-format-for-x-label-of-matplotlib
formatter = mpl.ticker.FuncFormatter(lambda ms, x: time.strftime('%M:%S', time.gmtime(ms // 1000)))
ax.xaxis.set_major_formatter(formatter)

# save the image
plt.savefig(f'figures/histograms/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')

plt.show(feature.title())
```



Time Series

In [26]:
```
# audio features
main_features = [
    'acousticness', 'danceability', 'energy', 'instrumentalness',
    'liveness', 'loudness', 'mode', 'speechiness', 'tempo', 'valence'
]

# date filter: this affects all time series plotting functions below
# date_filter = 'release_date >= 1958'  # original filter, revised based on later section (see note below)
date_filter = '1958 <= release_date < 2021'
```

NOTE:

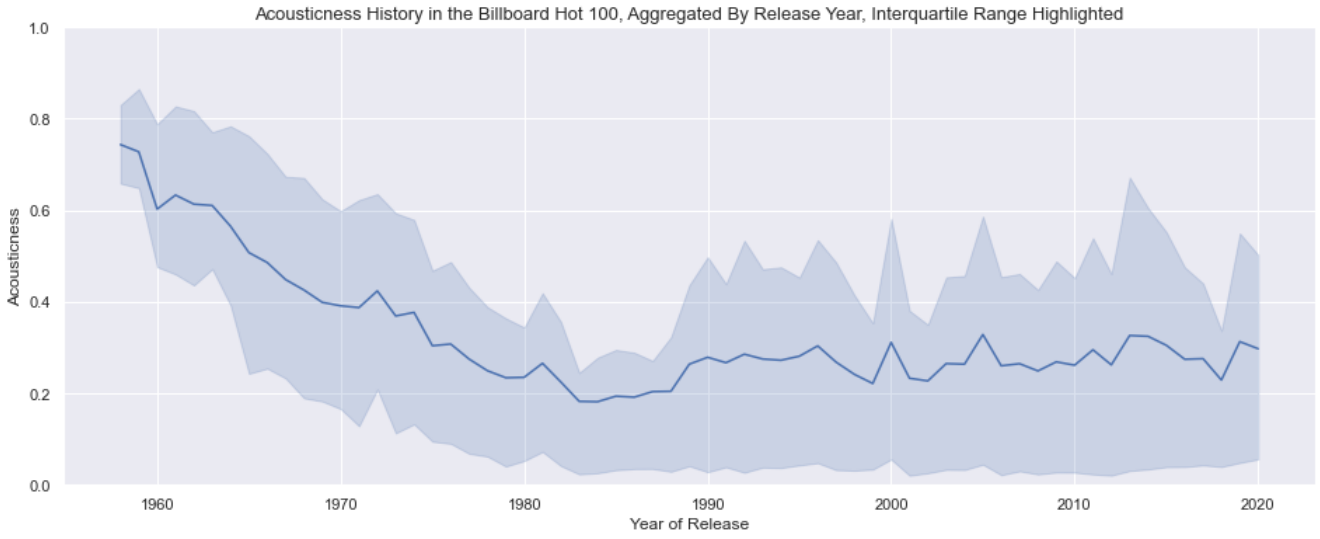
- The revised date_filter is based on data quality past 2020
- this is based on analysis later in this notebook: Time Series Counts - Preferred Date Filter
  - this analysis is later in the notebook because it was conducted to explain abberations in time series plots

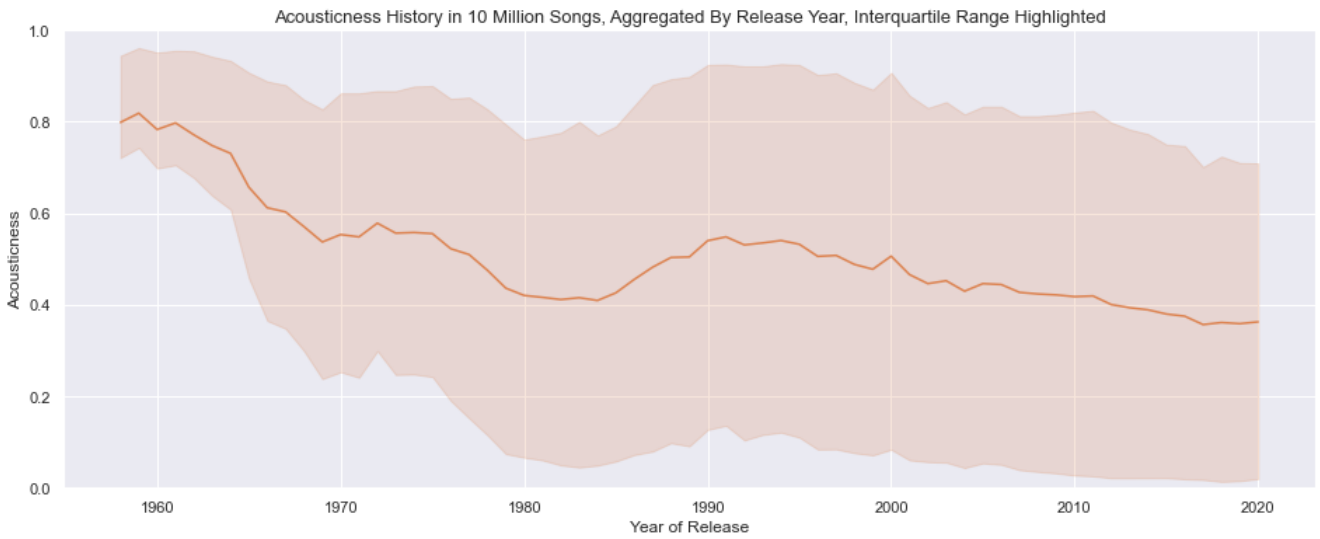
In [55]:
```
def plot_attribute_history(feature, dataframe=df_B100_songs_AF, title=None, colour=0):
    """
    Uses songs df_B100_songs_AF by default
    Plots Billboard Hot 100 audio features over time
    """
    # drop values before billboard 100 (lower quality data)
    temp_df = dataframe.query(date_filter)

    plt.figure(figsize=(16, 6))
    ax = sns.lineplot(
        x=temp_df.release_date.dt.year,
        y=temp_df[feature],
        color=sns.color_palette()[colour],
        errorbar=('pi', 50)  # middle 50% or IQR
    )

    if title:
        title=title
    else:
        title=f'{feature.capitalize()} History in the Billboard Hot 100, Aggregated By Release Year, Interquartile Range Highlighted'

    if feature == 'loudness':
        plt.ylim(-16, 0)  # reasonable range for loudness in music
```

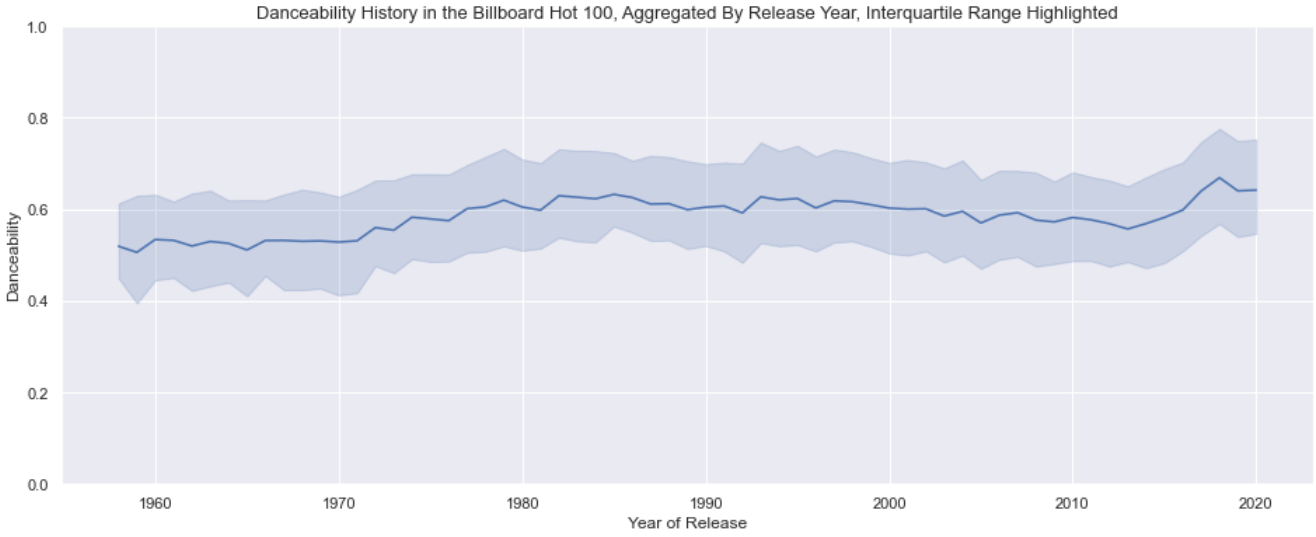
```
    elif feature == 'tempo':
        plt.ylim(40, 200)  # reasonable range for tempo in music
    else:
        plt.ylim(0, 1)  # most audio features vary between 0 and 1
        # NOTE: mode is an int, either 0 or 1 (minor/major),
        # but the average shows how commonly a song is in either mode

    plt.title(title, fontsize=13)
    plt.xlabel('Year of Release')
    plt.ylabel(feature.capitalize().replace('_', ' '))

    # save the image
    plt.savefig(f'figures/timeseries/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')

    plt.show()

# # test
# plot_attribute_history('acousticness')
```

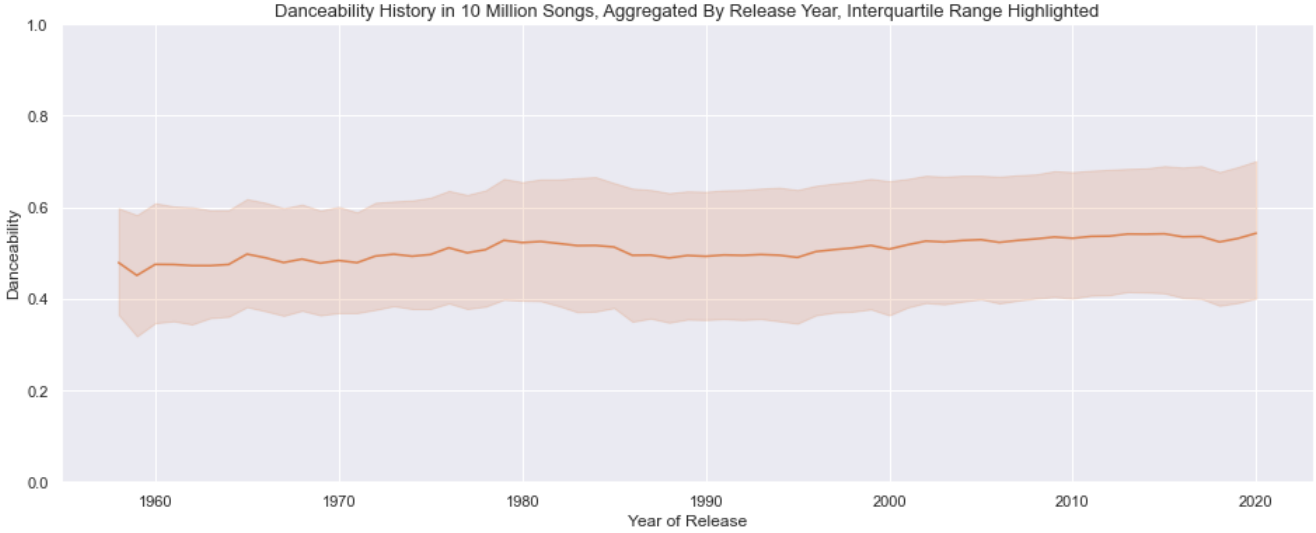
In [57]:

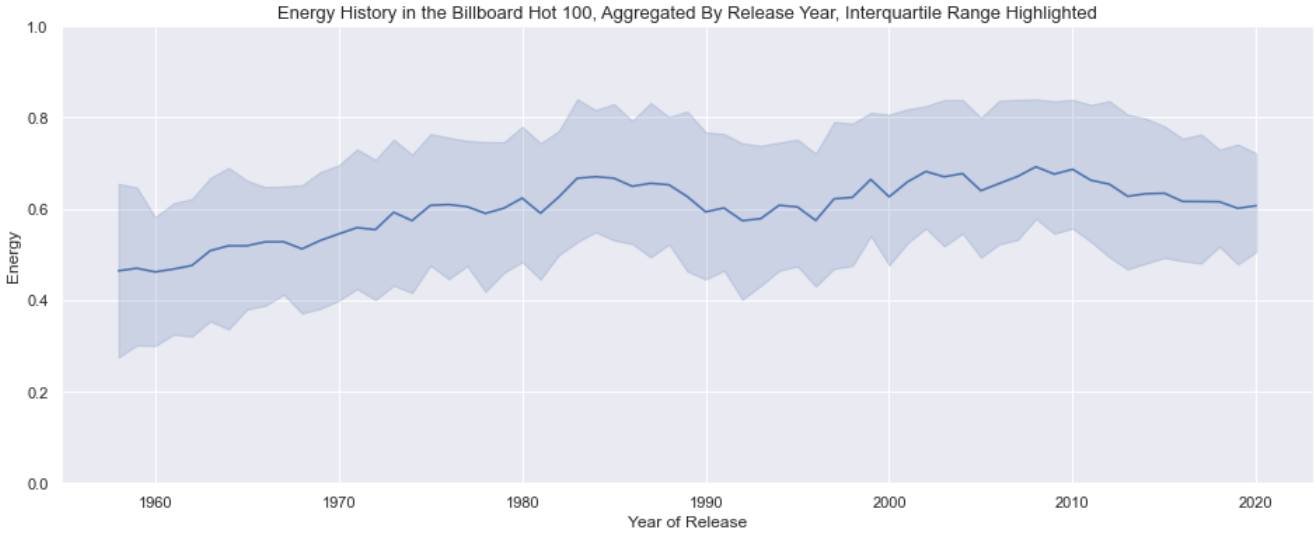
```
%%time
# plot features for Billboard Hot 100 and All Songs
for feature in main_features:
    # first plot the Billboard Hot 100
    plot_attribute_history(feature)
    # then plot the same feature from the large dataset
    title = f'{feature.capitalize()} History in 10 Million Songs, Aggregated By Release Year, Interquartile Range Highlighted'
    plot_attribute_history(feature, dataframe=df_10M, title=title, colour=1)
```

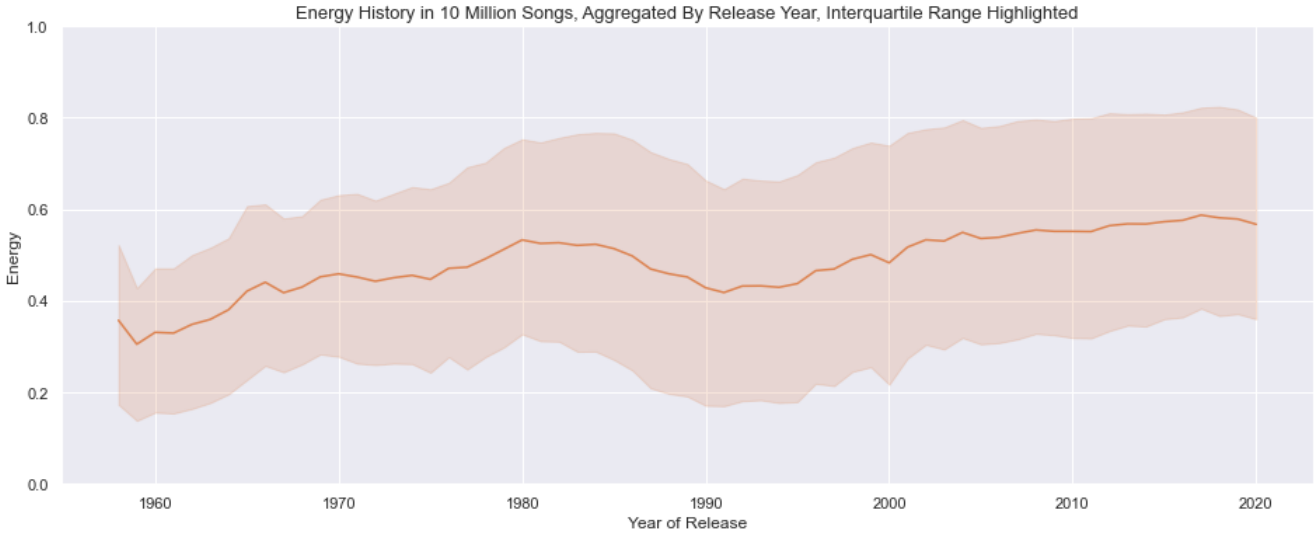


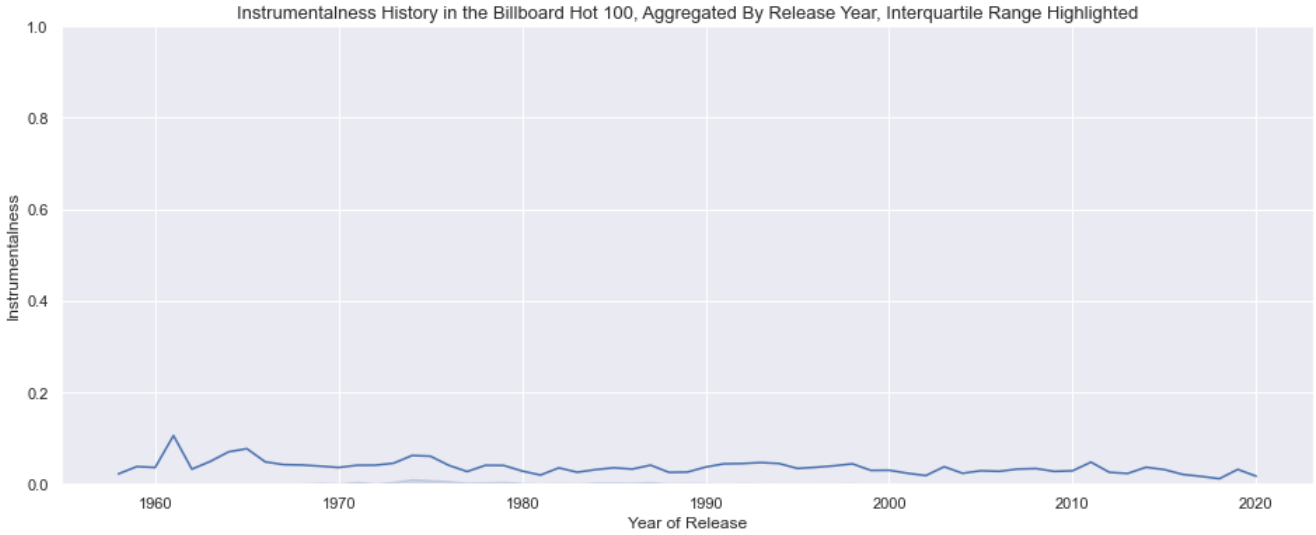


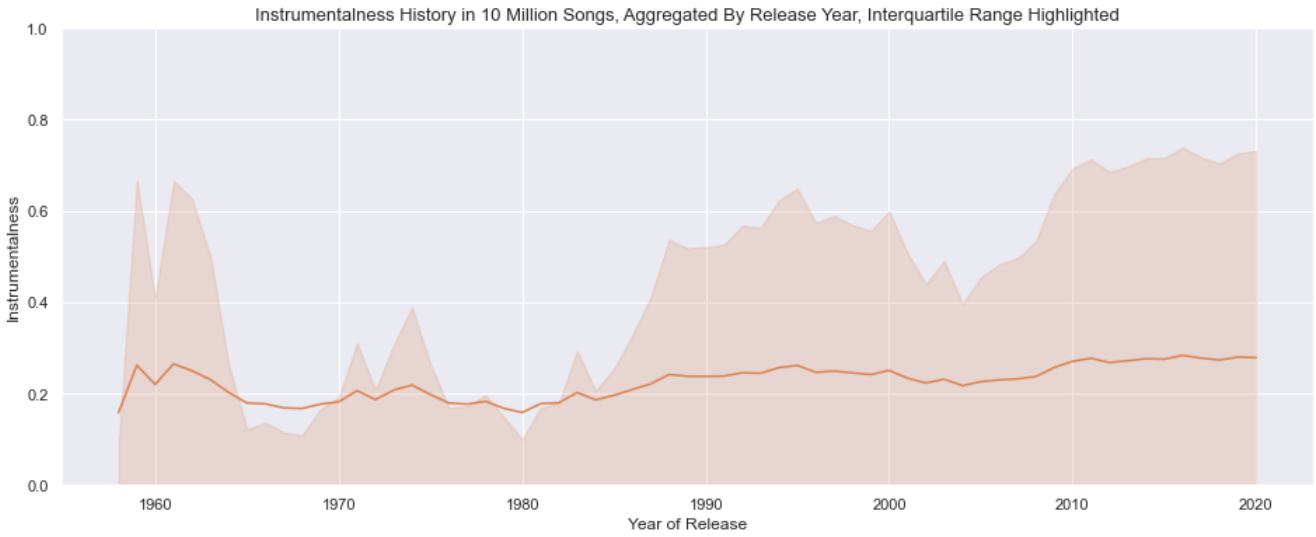

Danceability History in the Billboard Hot 100, Aggregated By Release Year, Interquartile Range Highlighted
Danceability
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year of Release

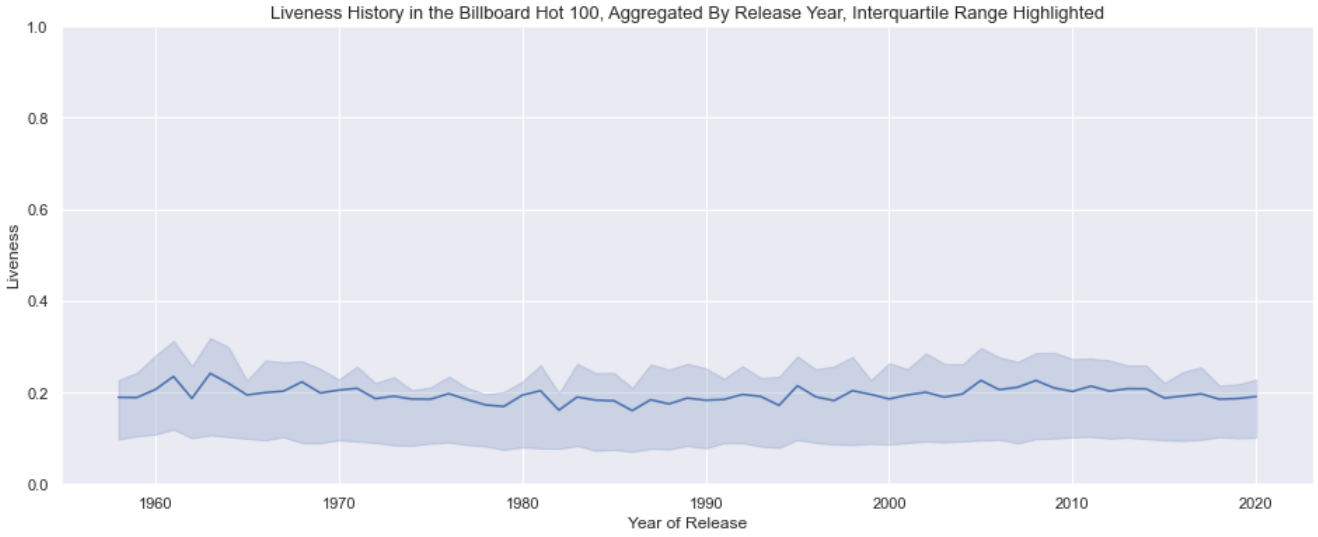

Danceability History in 10 Million Songs, Aggregated By Release Year, Interquartile Range Highlighted
Danceability
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year of Release

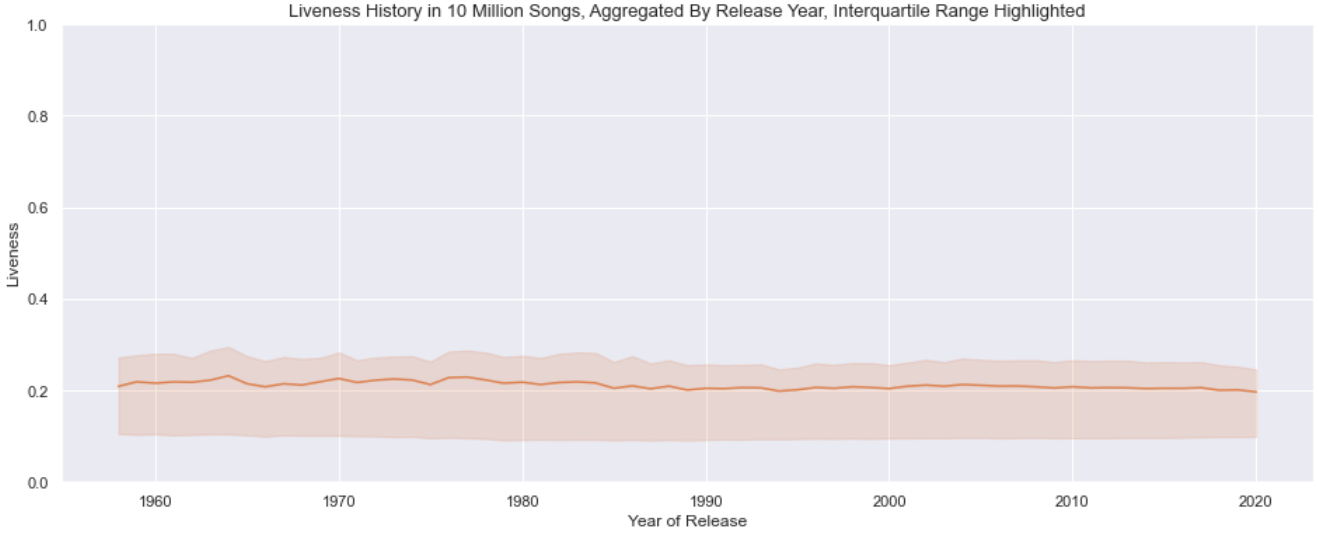

Energy History in the Billboard Hot 100, Aggregated By Release Year, Interquartile Range Highlighted
Energy
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year of Release

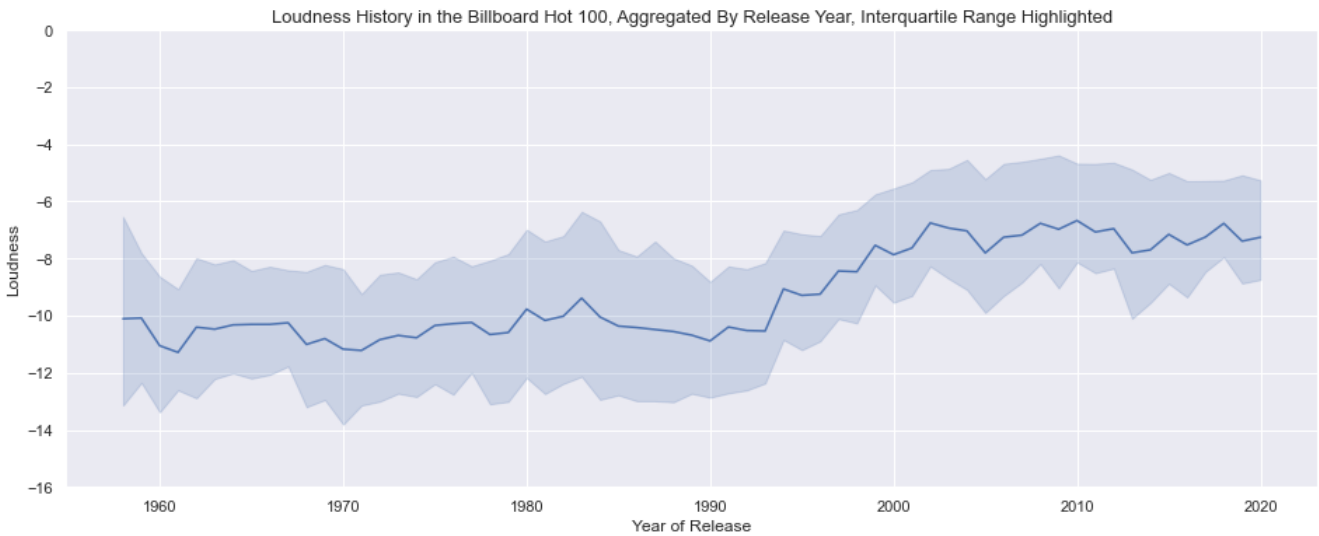

Energy History in 10 Million Songs, Aggregated By Release Year, Interquartile Range Highlighted
Energy
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year of Release

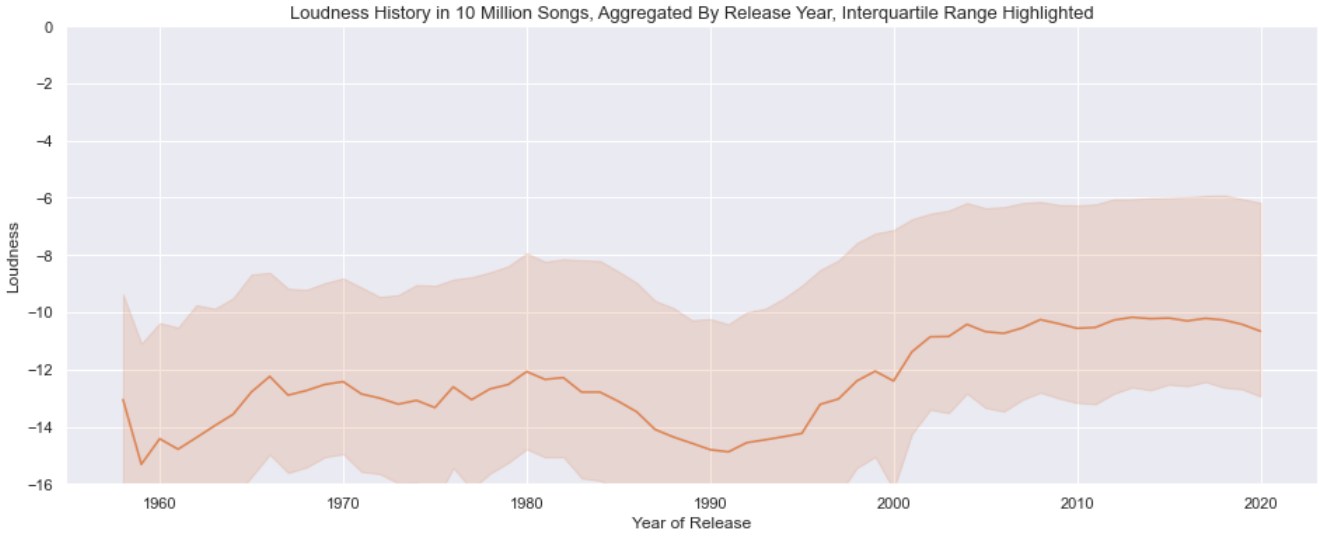

Instrumentalness History in the Billboard Hot 100, Aggregated By Release Year, Interquartile Range Highlighted
Instrumentalness
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year of Release

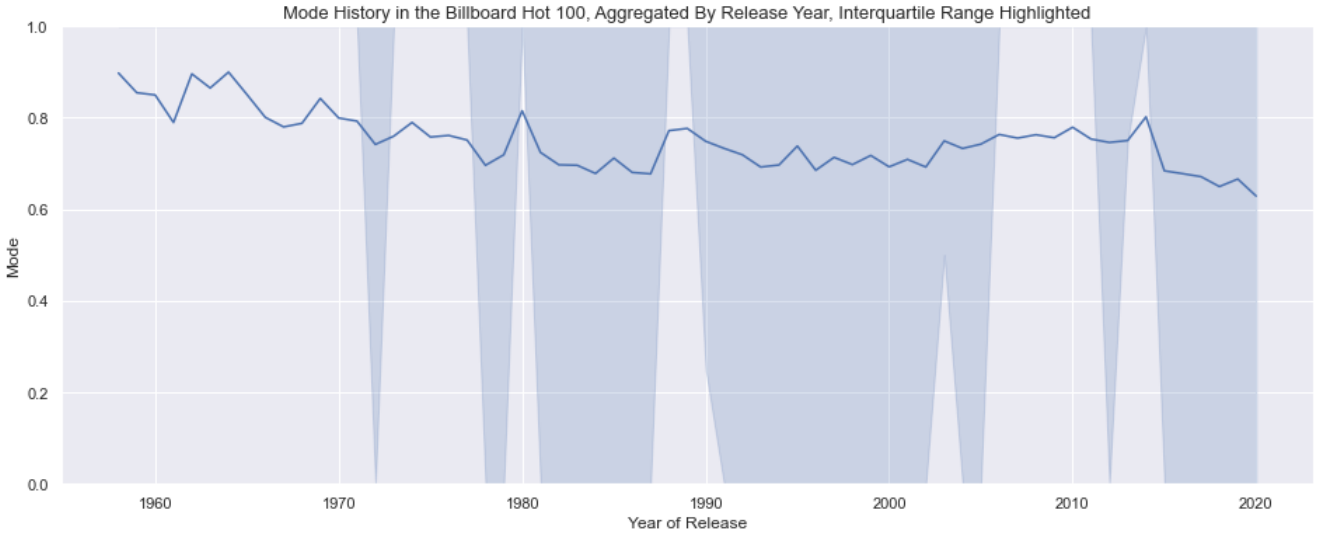

Instrumentalness History in 10 Million Songs, Aggregated By Release Year, Interquartile Range Highlighted
Instrumentalness
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year of Release

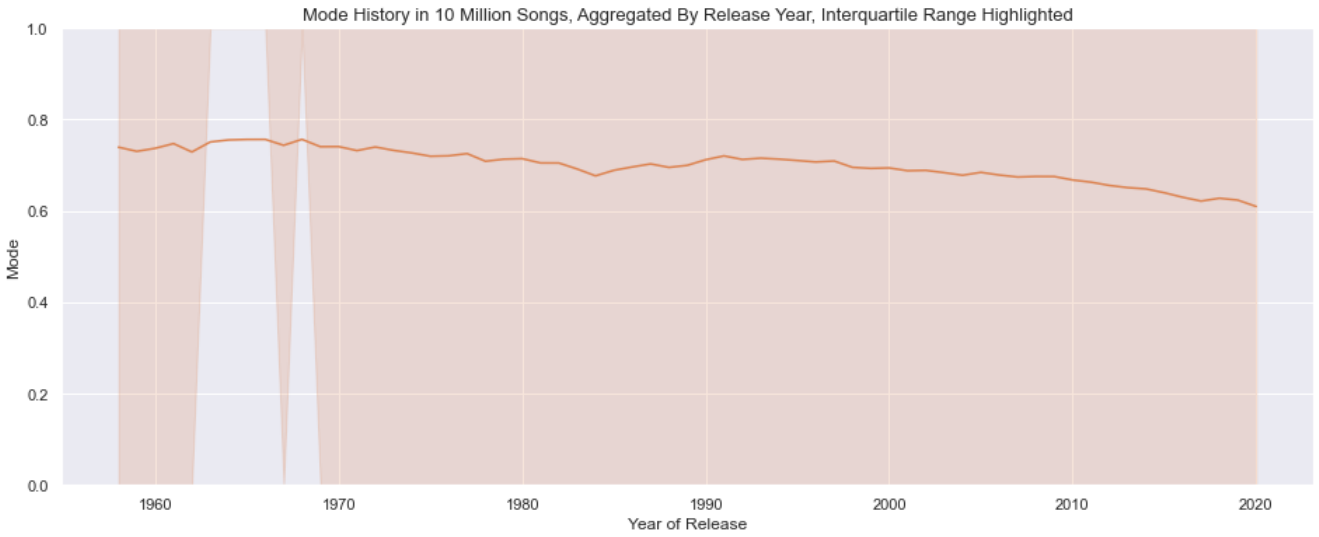

Liveness History in the Billboard Hot 100, Aggregated By Release Year, Interquartile Range Highlighted
Liveness
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year of Release

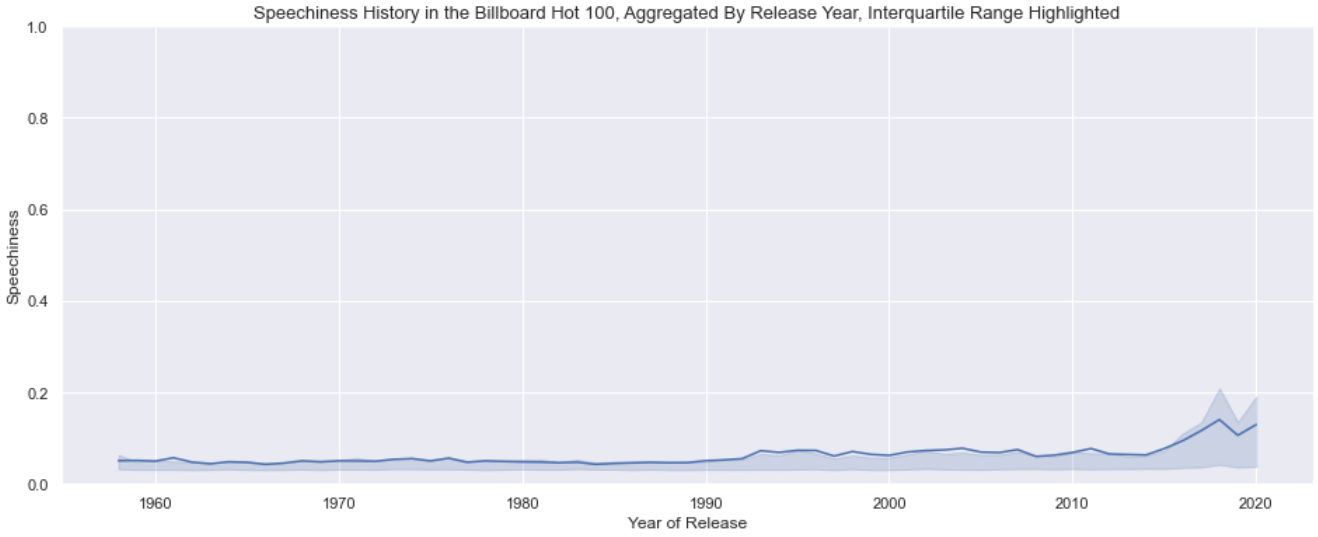

Liveness History in 10 Million Songs, Aggregated By Release Year, Interquartile Range Highlighted
Liveness
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year of Release

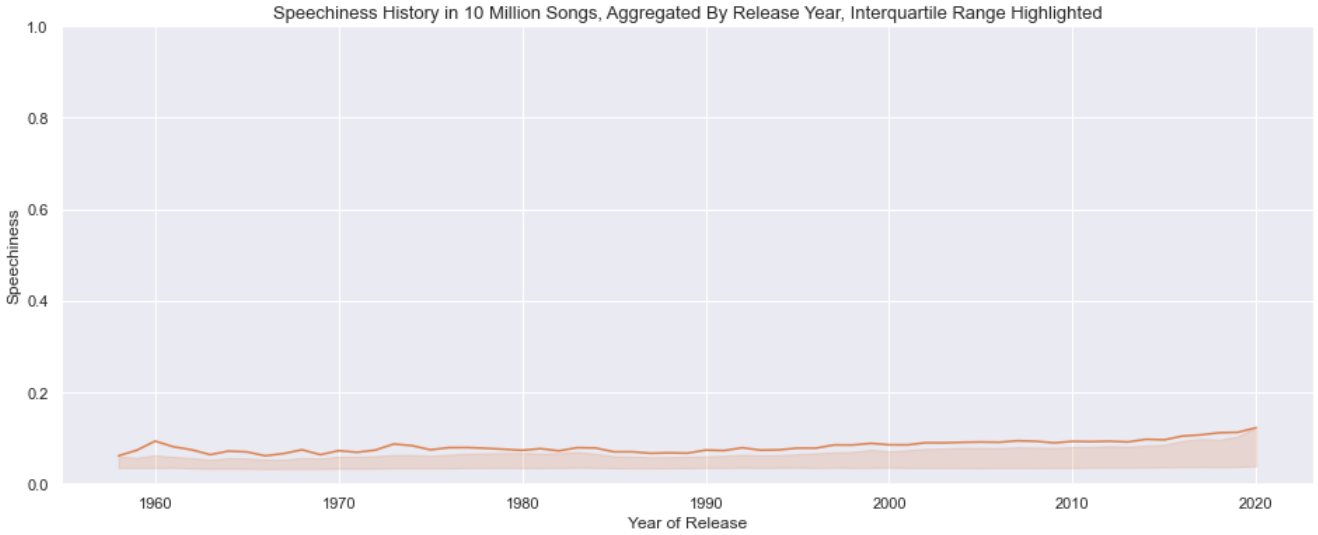

Loudness History in the Billboard Hot 100, Aggregated By Release Year, Interquartile Range Highlighted
Loudness
0
−2
−4
−6
−8
−10
−12
−14
−16
1960
1970
1980
1990
2000
2010
2020
Year of Release

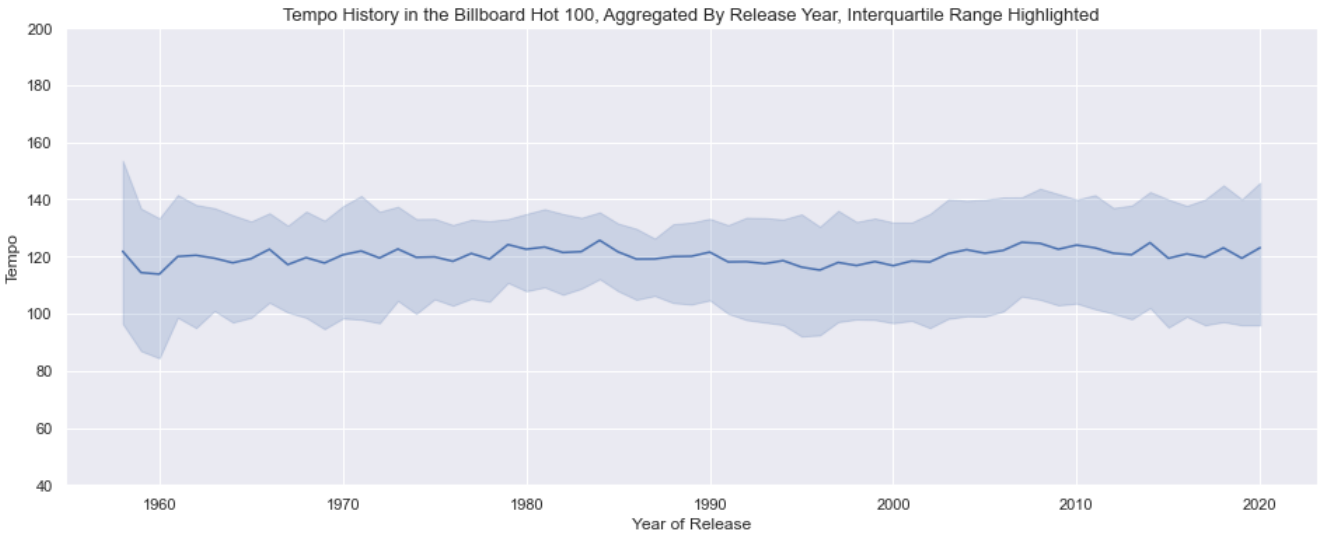

Loudness History in 10 Million Songs, Aggregated By Release Year, Interquartile Range Highlighted
Loudness
0
−2
−4
−6
−8
−10
−12
−14
−16
1960
1970
1980
1990
2000
2010
2020
Year of Release

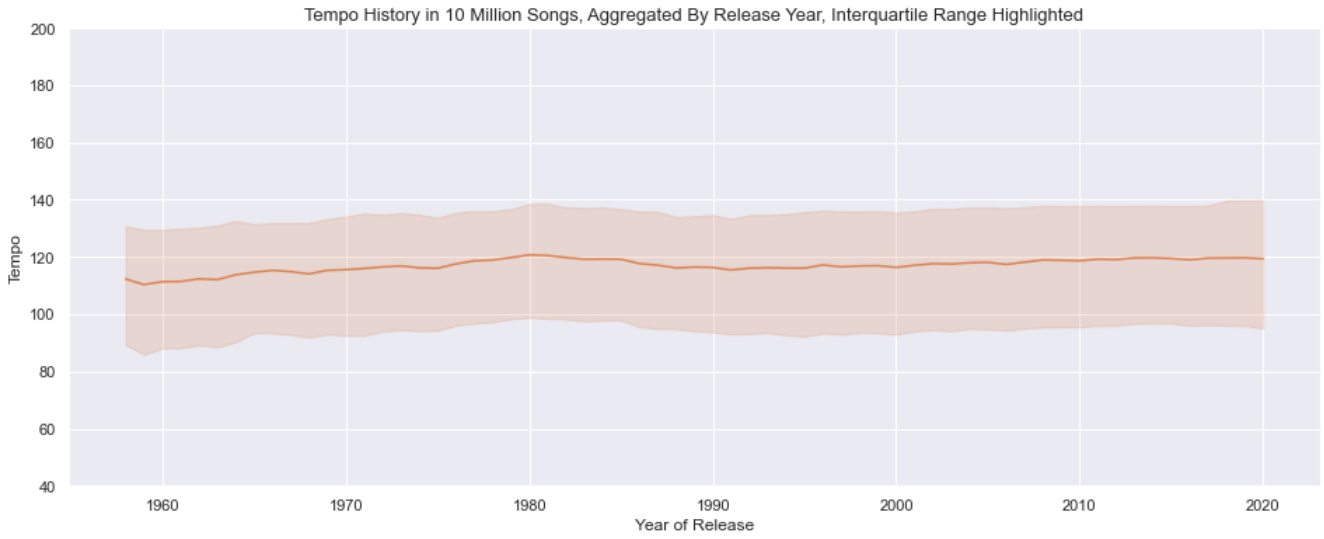

Mode History in the Billboard Hot 100, Aggregated By Release Year, Interquartile Range Highlighted
Mode
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year of Release

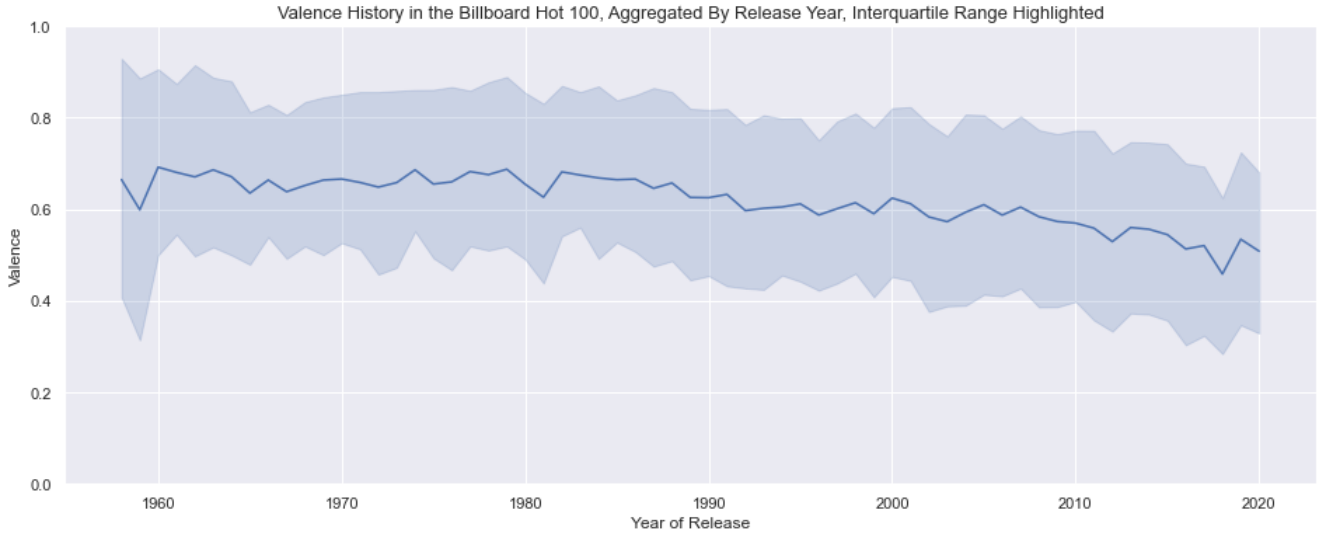

Mode History in 10 Million Songs, Aggregated By Release Year, Interquartile Range Highlighted
Mode
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year of Release

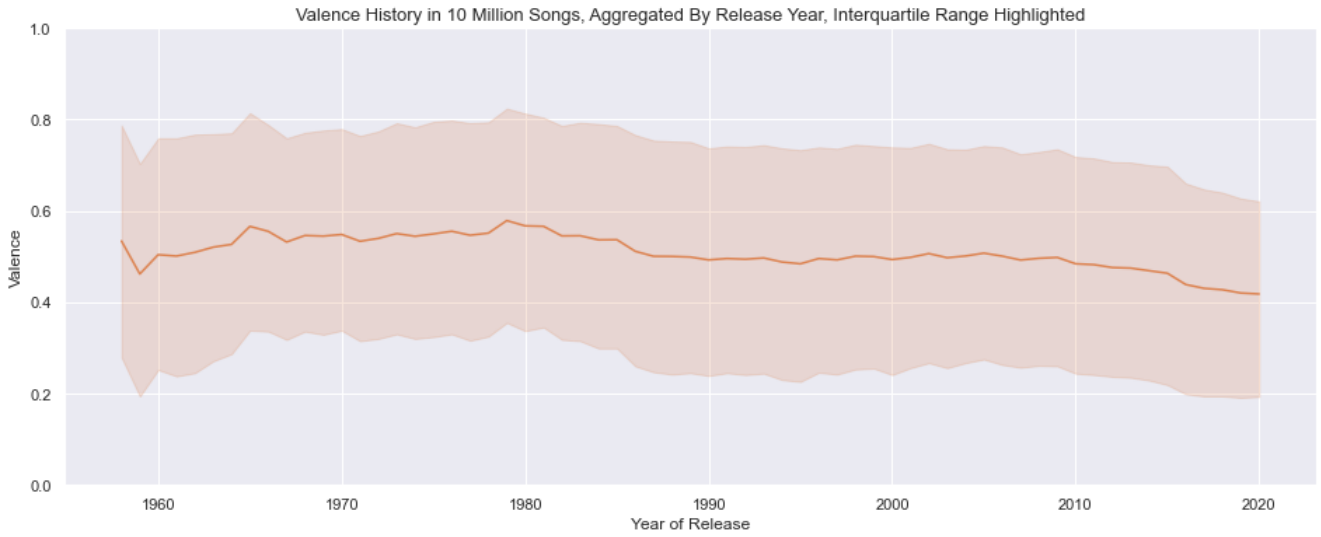

Speechiness History in the Billboard Hot 100, Aggregated By Release Year, Interquartile Range Highlighted
Speechiness
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year of Release


Speechiness History in 10 Million Songs, Aggregated By Release Year, Interquartile Range Highlighted
Speechiness
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year of Release


Tempo History in the Billboard Hot 100, Aggregated By Release Year, Interquartile Range Highlighted
Tempo
200
180
160
140
120
100
80
60
40
1960
1970
1980
1990
2000
2010
2020
Year of Release







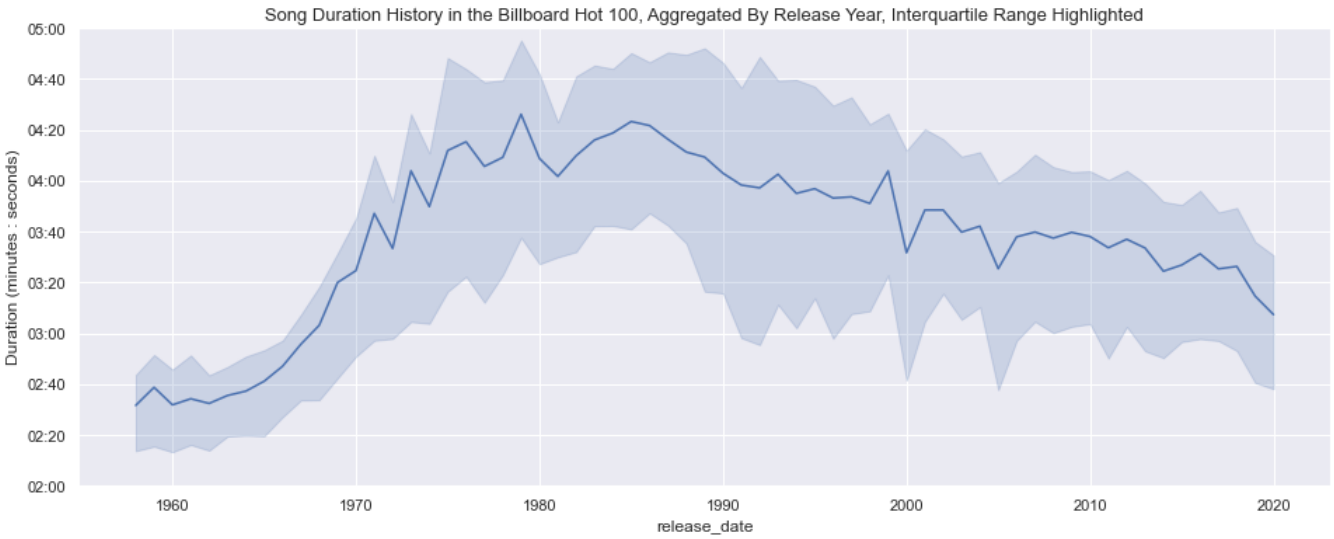

```
Wall time: 1min 18s
```

In [58]:

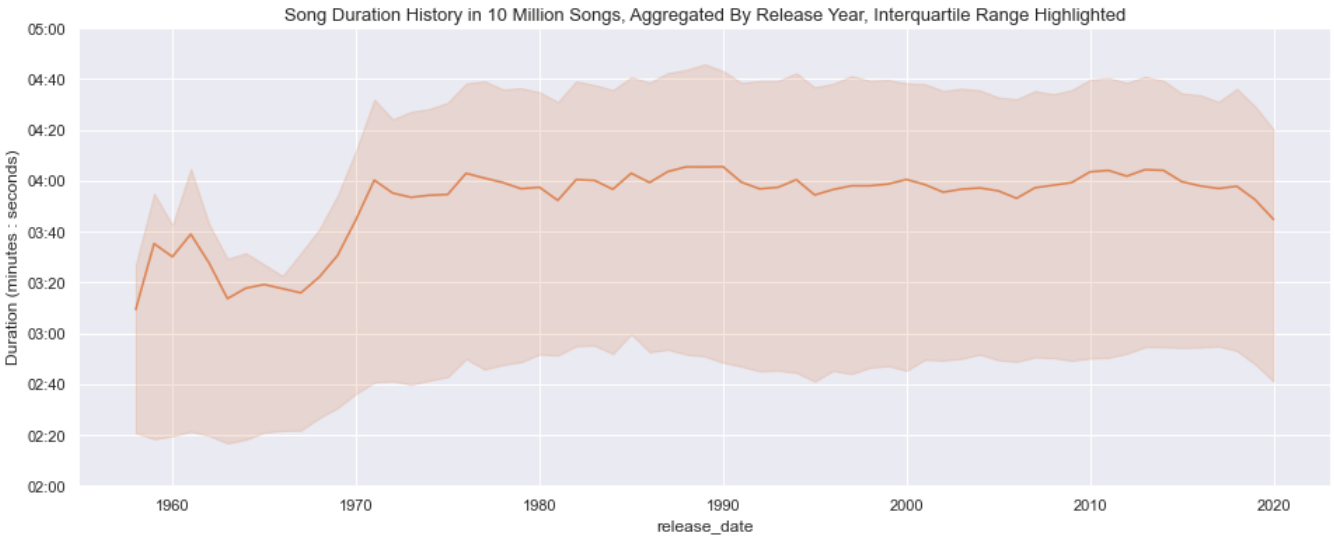
```
# audio features
other_features = [
    'key', 'time_signature', # integers don't make sense in timeseries lineplot (maybe heatplot?)
    'duration_ms',  # irregular feature, plotting requires some extra work
]

def plot_duration(feature='duration_ms', dataframe=df_B100_songs_AF, title=None, colour=0):
    """
    Uses songs df_B100_songs_AF
    Plots Billboard Hot 100 audio features over time
```

```
    """
    # drop values before billboard 100 (lower quality data)
    temp_df = dataframe.query(date_filter)

    plt.figure(figsize=(16, 6))
    ax = sns.lineplot(
        x=temp_df.release_date.dt.year,
        y=temp_df[feature],
        color=sns.color_palette()[colour],
        errorbar=('pi', 50)  # middle 50% or IQR
    )

    formatter = mpl.ticker.FuncFormatter(lambda ms, x: time.strftime('%M:%S', time.gmtime(ms // 1000)))
    ax.yaxis.set_major_formatter(formatter)
    plt.ylabel('Duration (minutes : seconds)')

    if title:
        title=title
    else:
        title=f'Song Duration History in the Billboard Hot 100, Aggregated By Release Year, Interquartile Range Highlighted'

    plt.ylim(120000, 300000)  # reasonable range for duration of music

    plt.title(title, fontsize=13)

    # save the image
    plt.savefig(f'figures/timeseries/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')

    plt.show()
```

In [59]:
```
plot_duration(feature='duration_ms')
```



In [60]:
```
%%time
title = 'Song Duration History in 10 Million Songs, Aggregated By Release Year, Interquartile Range Highlighted'
plot_duration(feature='duration_ms', dataframe=df_10M, title=title, colour=1)
```




```
Wall time: 12.2 s
```

Comparative Time Series

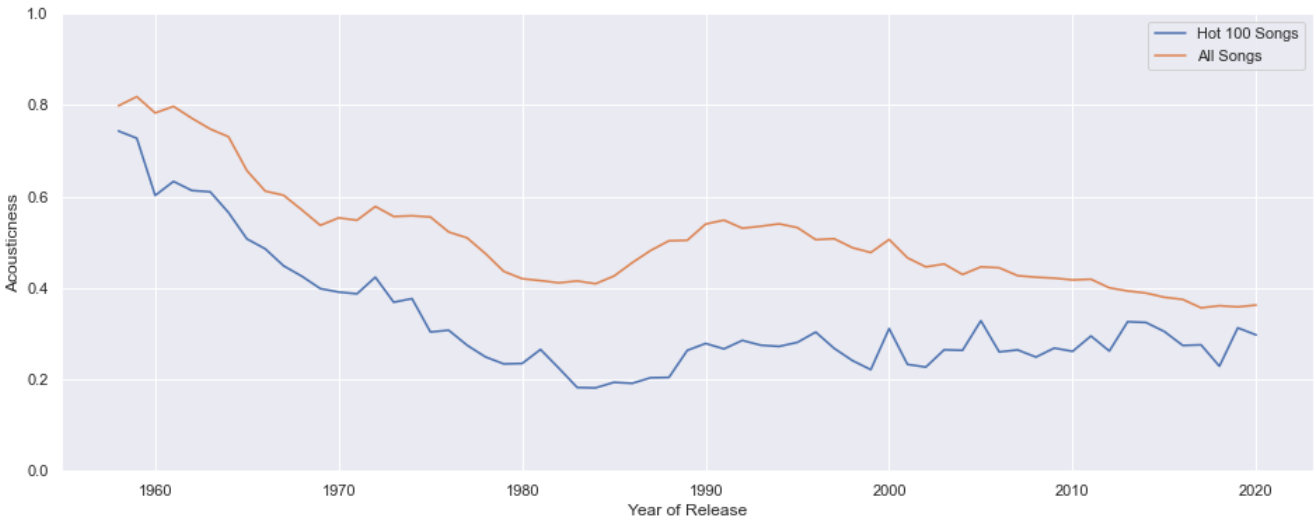
In [27]:

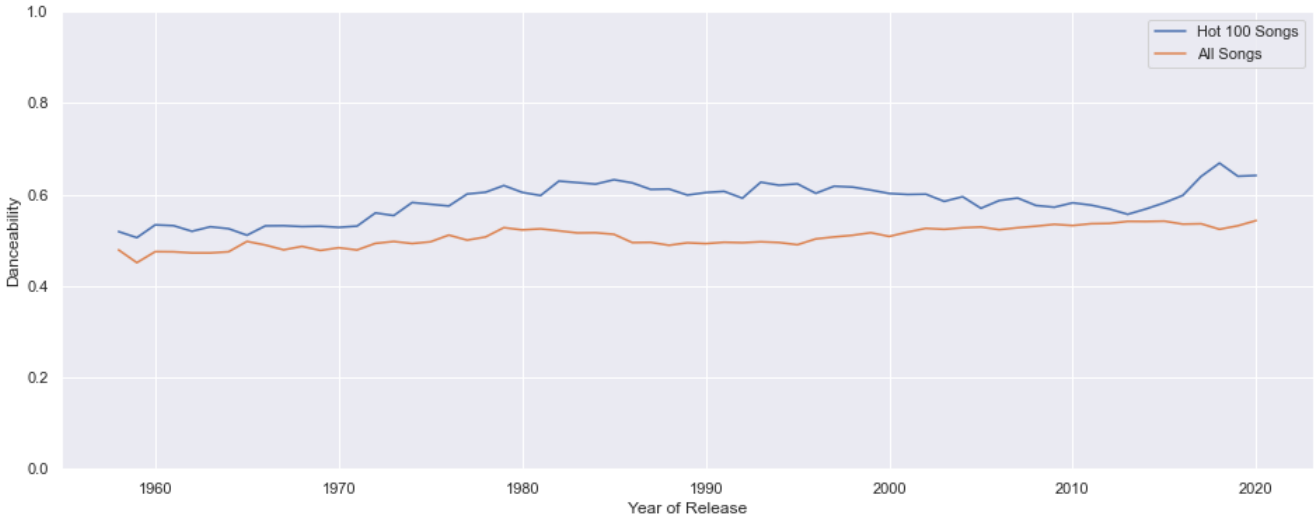
```
def plot_feature_comparison(feature):
    """
    Uses songs df_B100_songs_AF
    Plots Billboard Hot 100 audio features over time
    """

    fig, ax = plt.subplots(figsize=(16, 6))

    df1 = df_B100_songs_AF.query(date_filter)
    ax = sns.lineplot(
        x=df1.release_date.dt.year,
        y=df1[feature],
        color=sns.color_palette()[0],
        errorbar=None,
        label='Hot 100 Songs'
    )

    df2 = df_10M.query(date_filter)
    ax = sns.lineplot(
        x=df2.release_date.dt.year,
        y=df2[feature],
        color=sns.color_palette()[1],
        errorbar=None,
        label='All Songs'
    )

    title=f'{feature.capitalize()} History Comparing the Billboard Hot 100 with All Songs'

    if feature == 'loudness':
        plt.ylim(-16, 0)  # reasonable range for loudness in music
    elif feature == 'tempo':
        plt.ylim(40, 200)  # reasonable range for tempo in music
    else:
        plt.ylim(0, 1)  # most audio features vary between 0 and 1
        # NOTE: mode is an int, either 0 or 1 (minor/major),
        # but the average shows how commonly a song is in either mode

#     plt.title(title, fontsize=13)
    plt.xlabel('Year of Release')
    plt.ylabel(feature.capitalize().replace('_', ' '))
    plt.legend(loc='upper right')

    # save the image
    plt.savefig(f'figures/timeseries/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')

    plt.show()


def plot_duration_comparison(feature='duration_ms'):
    """
    Uses songs df_B100_songs_AF
    Plots Billboard Hot 100 audio features over time
    """

    fig, ax = plt.subplots(figsize=(16, 6))
```

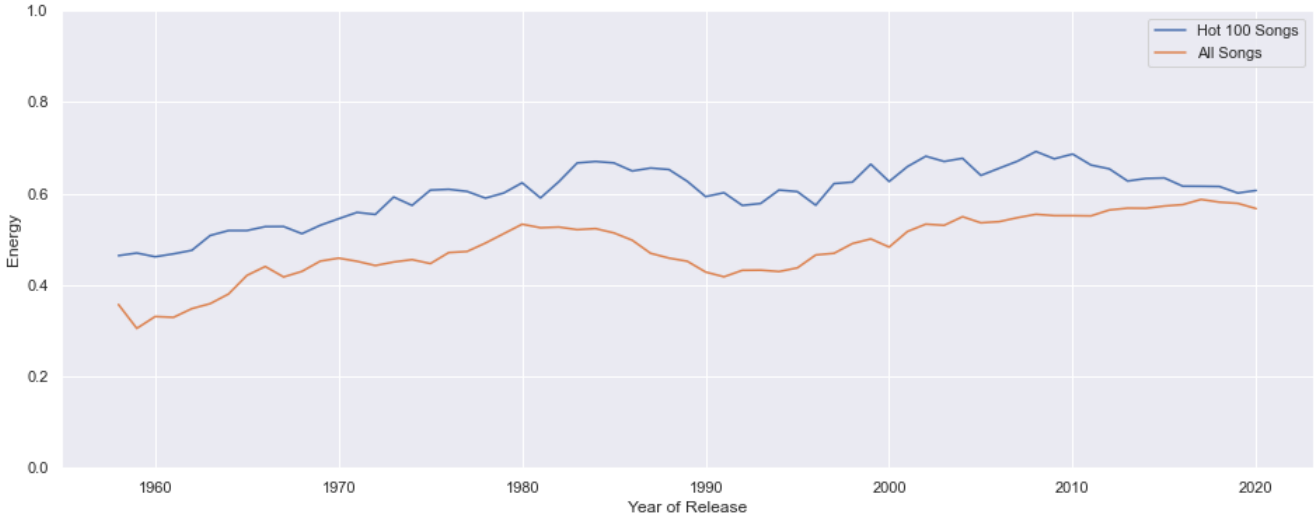
```
    df1 = df_B100_songs_AF.query(date_filter)
    ax = sns.lineplot(
        x=df1.release_date.dt.year,
        y=df1[feature],
        color=sns.color_palette()[0],
        errorbar=None,
        label='Hot 100 Songs'
    )

    df2 = df_10M.query(date_filter)
    ax = sns.lineplot(
        x=df2.release_date.dt.year,
        y=df2[feature],
        color=sns.color_palette()[1],
        errorbar=None,
        label='All Songs'
    )

    formatter = mpl.ticker.FuncFormatter(lambda ms, x: time.strftime('%M:%S', time.gmtime(ms // 1000)))
    ax.yaxis.set_major_formatter(formatter)
    plt.ylabel('Duration (minutes : seconds)')

    title='Song Duration History Comparing the Billboard Hot 100 with All Songs'

    plt.ylim(120000, 300000)  # reasonable range for duration of music

#     plt.title(title, fontsize=13)
    plt.xlabel('Year of Release')
    plt.legend(loc='upper right')

    # save the image
    plt.savefig(f'figures/timeseries/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')

    plt.show()
```

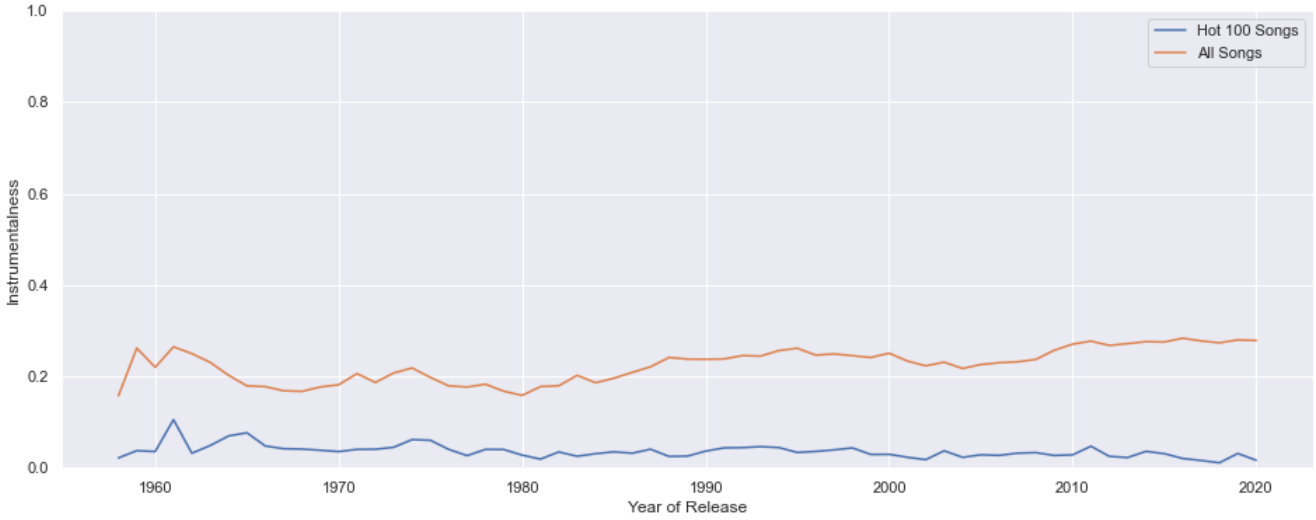
In [28]:

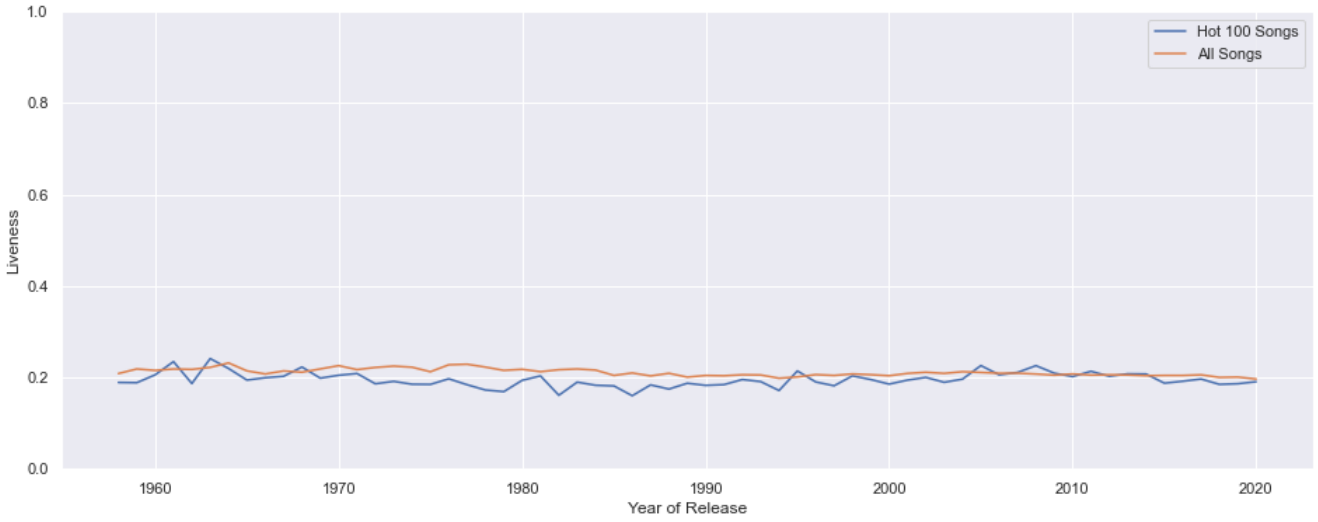
```
%%time
# audio features
main_features = [
    'acousticness', 'danceability', 'energy', 'instrumentalness',
    'liveness', 'loudness', 'mode', 'speechiness', 'tempo', 'valence'
]

for feature in main_features:
    plot_feature_comparison(feature)
plot_duration_comparison()
```

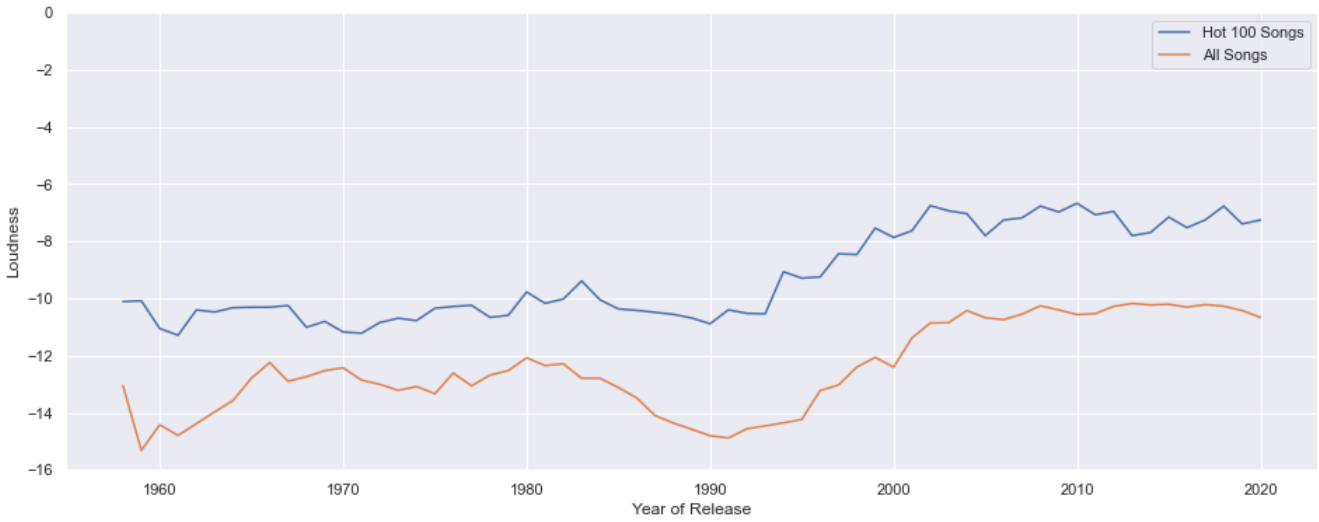


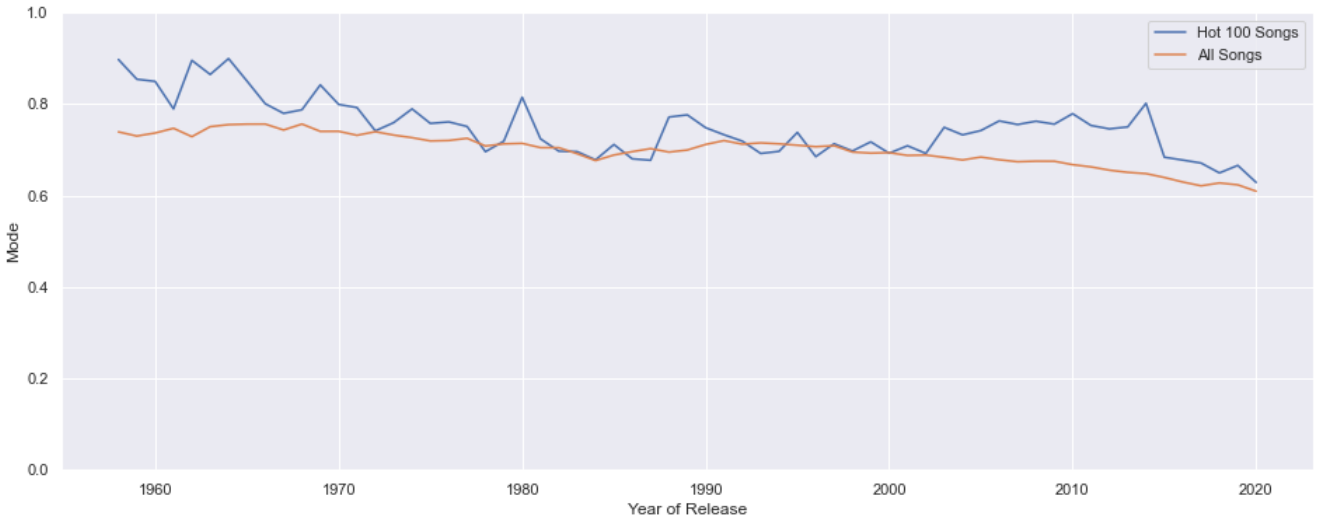

Hot 100 Songs
All Songs
Danceability
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year of Release

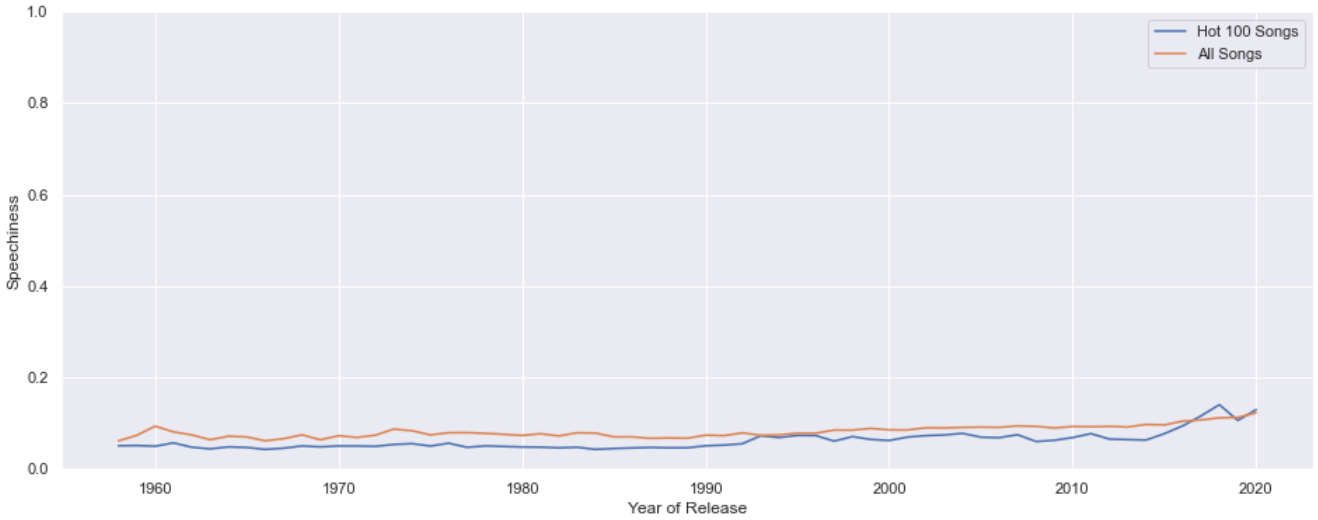

Hot 100 Songs
All Songs
Energy
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year of Release

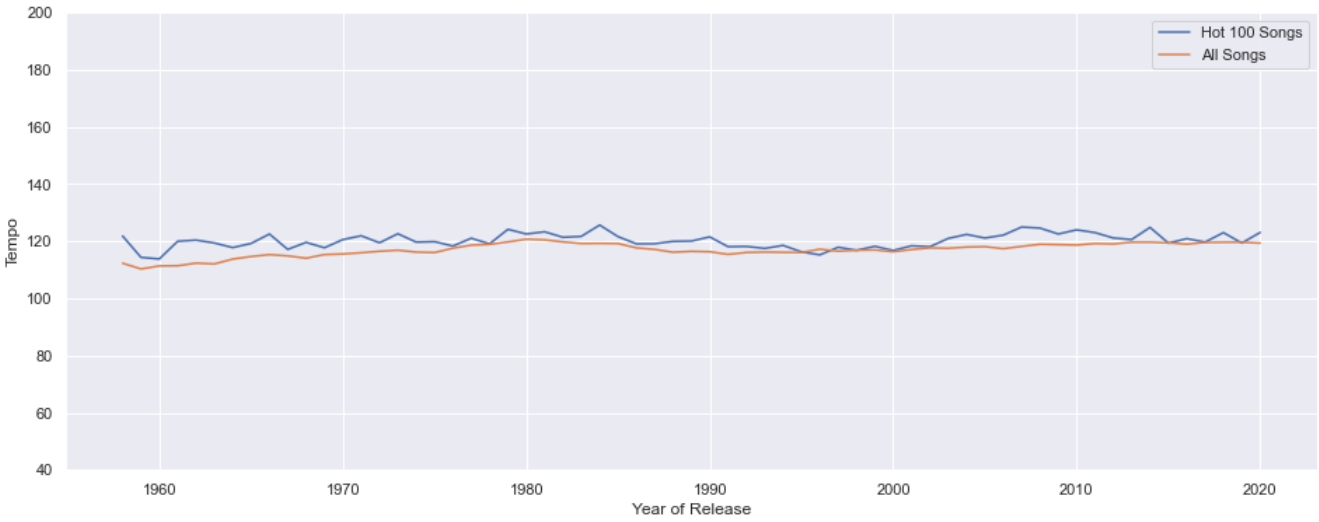

Hot 100 Songs
All Songs
Instrumentalness
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year of Release

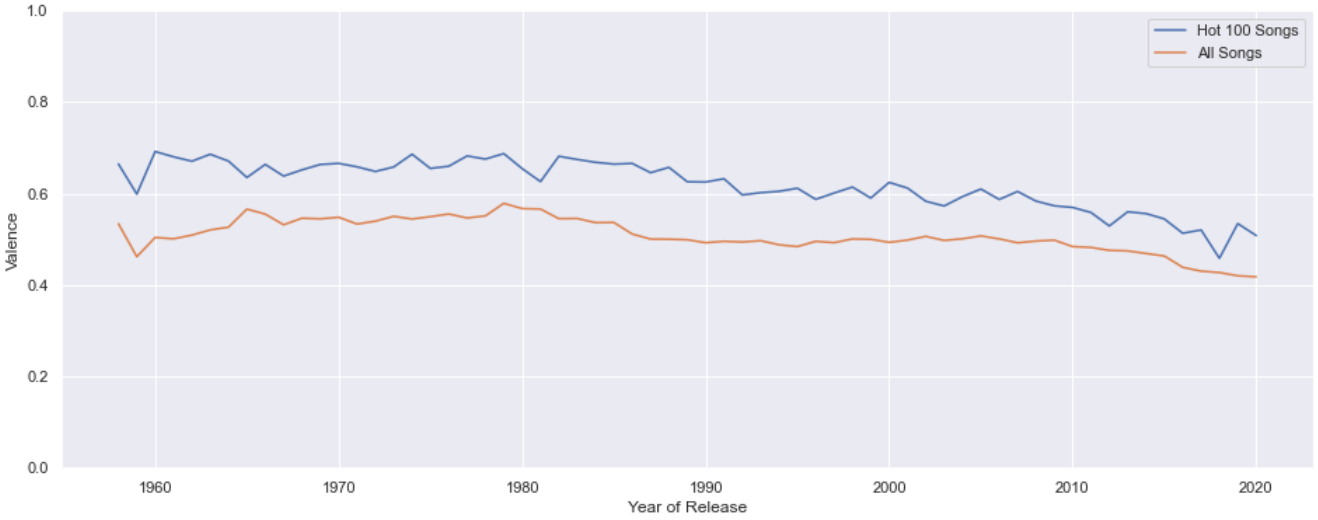

Hot 100 Songs
All Songs
Liveness
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year of Release

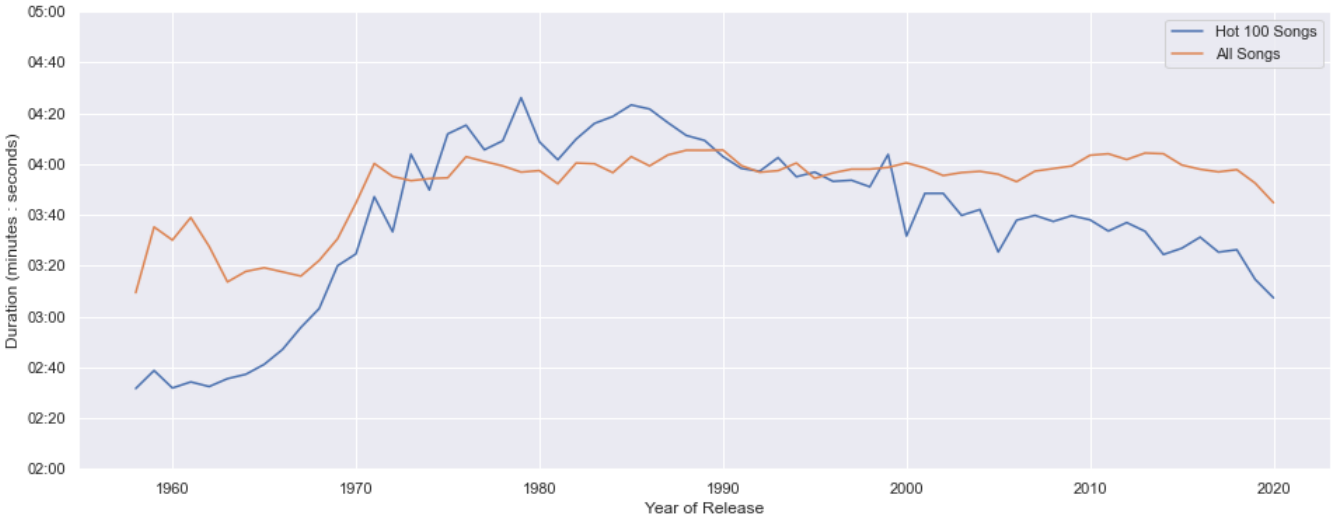

Hot 100 Songs
All Songs
Loudness
0
−2
−4
−6
−8
−10
−12
−14
−16
1960
1970
1980
1990
2000
2010
2020
Year of Release


Hot 100 Songs
All Songs
Mode
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year of Release


Hot 100 Songs
All Songs
Speechiness
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year of Release


Hot 100 Songs
All Songs
Tempo
200
180
160
140
120
100
80
60
40
1960
1970
1980
1990
2000
2010
2020
Year of Release


Hot 100 Songs
All Songs
Valence
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year of Release




```
Wall time: 1min 22s
```

Time Series Counts - Preferred Date Filter

- data seems to deteriorate after 2020, with large movements in music trends (CORRECT)
- alternatively, this could be due to significant societal changes and preferences (NOT VERIFIABLE) ##### Let's find out!

In [30]:
```
df_10M.release_date.dt.year.value_counts().reset_index().sort_values('index', ascending=False).head()
```

Out[30]:

	index	release_date
67	2022.000	68
27	2021.000	84478
9	2020.000	359973
11	2019.000	329738
12	2018.000	321232

In [31]:
```
df_B100_songs.release_date.dt.year.value_counts().reset_index().sort_values('index', ascending=False).head()
```

Out[31]:

	index	release_date
63	2022.000	112
17	2021.000	419
0	2020.000	648
7	2019.000	494
8	2018.000	493

In [32]:
```
# find max dates for each dataset

# SQL
max_date_sql = 1644537600000
pd.to_datetime(max_date_sql, unit='ms', origin='unix')
```

Out[32]:
```
Timestamp('2022-02-11 00:00:00')
```

In [36]:
```
# Billboard Hot 100
df_B100.sort_values('date', ascending=False).head(1)['date']
```

Out[36]:
```
329929   2021-11-06
Name: date, dtype: datetime64[ns]
```

In [42]:
```
# CSV
pd.read_csv('D:/RYERSON/820/Datasets/Spotify 1.2M+ Songs/tracks_features.csv').sort_values('release_date', ascending=False).head(1)['release_d
```

Out[42]:
```
1134062    2020-12-18
Name: release_date, dtype: object
```

In [48]:
```
# Latest Release Dates by Dataset
```

```
latest_sql = pd.to_datetime('2022-02-11 00:00:00')
latest_B100 = pd.to_datetime('2021-11-06 00:00:00')
latest_csv = pd.to_datetime('2020-12-18 00:00:00')

# max date option 1 (all datasets consistent):
maxdate_1 = min(latest_sql, latest_B100, latest_csv)

# max date option 2 (all Billboard 100 songs):
maxdate_2 = min(latest_sql, latest_B100)

# options for filtering
maxdate_1, maxdate_2
```

Out[48]:
```
(Timestamp('2020-12-18 00:00:00'), Timestamp('2021-11-06 00:00:00'))
```

In [51]:
```
# let's check the quantiy (not quality) of 2021 (with SQL, without CSV) vs 2020 (both)

df_10M.query('2021 <= release_date < 2022').shape[0], df_10M.query('2020 <= release_date < 2021').shape[0]
```

Out[51]:
```
(84478, 359973)
```

In [52]:
```
# only 23% of the songs during 2021 vs 2020: looks like maxdate_1 is a better choice
df_10M.query('2021 <= release_date < 2022').shape[0] / df_10M.query('2020 <= release_date < 2021').shape[0]
```

Out[52]:
```
0.2346787120145122
```

CONCLUSION: We should only include data up to, but not including 2021

In [14]:
```
if []:
    print('hi')
```

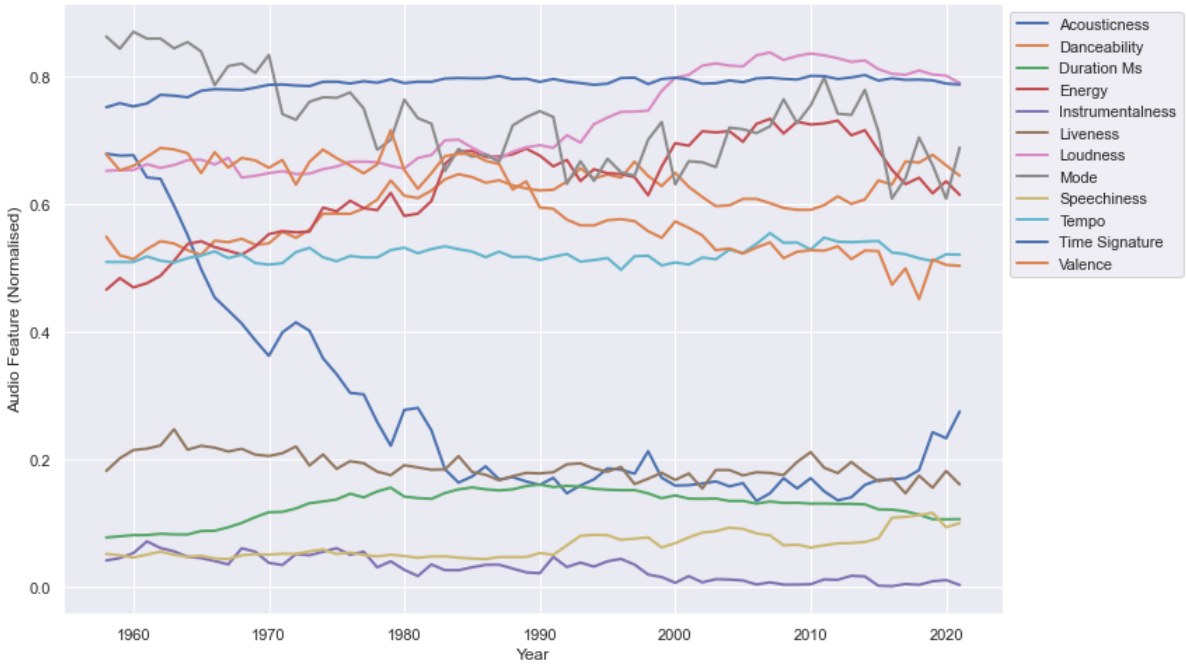
Billboard Charts Historical Plots

In [34]:
```
# audio features
normalised_features = [
    'acousticness', 'danceability', 'energy', 'instrumentalness',
    'liveness', 'mode', 'speechiness', 'valence'
]

# features to normalise
features_to_normalise = [
    'loudness', 'tempo', 'duration_ms', 'time_signature'
]

def plot_billboard_history(normalised_features, features_to_normalise):
    """
    Uses songs df_B100
    Plots Billboard Hot 100 audio features over time
    """
    if features_to_normalise:  # if there are features to normalise
        # combine lists of features
        list_of_features = sorted([*normalised_features, *features_to_normalise])

        # create the normalised dataframe
        df = df_B100.dropna().copy()
        n = features_to_normalise  # just to shorten the next line of code
        df[[*n]] = (df[[*n]]-df[[*n]].min()) / (df[[*n]].max()-df[[*n]].min())
    else:
        list_of_features = normalised_features
        df = df_B100.dropna().copy()

    # create the plot
    fig, ax = plt.subplots(figsize=(12, 8))

    for feature in list_of_features:
        ax = sns.lineplot(
            x=df.date.dt.year,
            y=df[feature],
            errorbar=None,
            linewidth=2,
            label=feature.replace('_', ' ').title()
        )

    title='Billboard Hot 100 Historical Charts - All Features Normalised'

#     plt.title(title, fontsize=13)
    plt.xlabel('Year')
    plt.ylabel('Audio Feature (Normalised)')
    plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
```

```
    # save the image
    plt.savefig(f'figures/timeseries/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')

    plt.show()
```

In [35]:

```
# plot all features historically
plot_billboard_history(normalised_features, features_to_normalise)
```



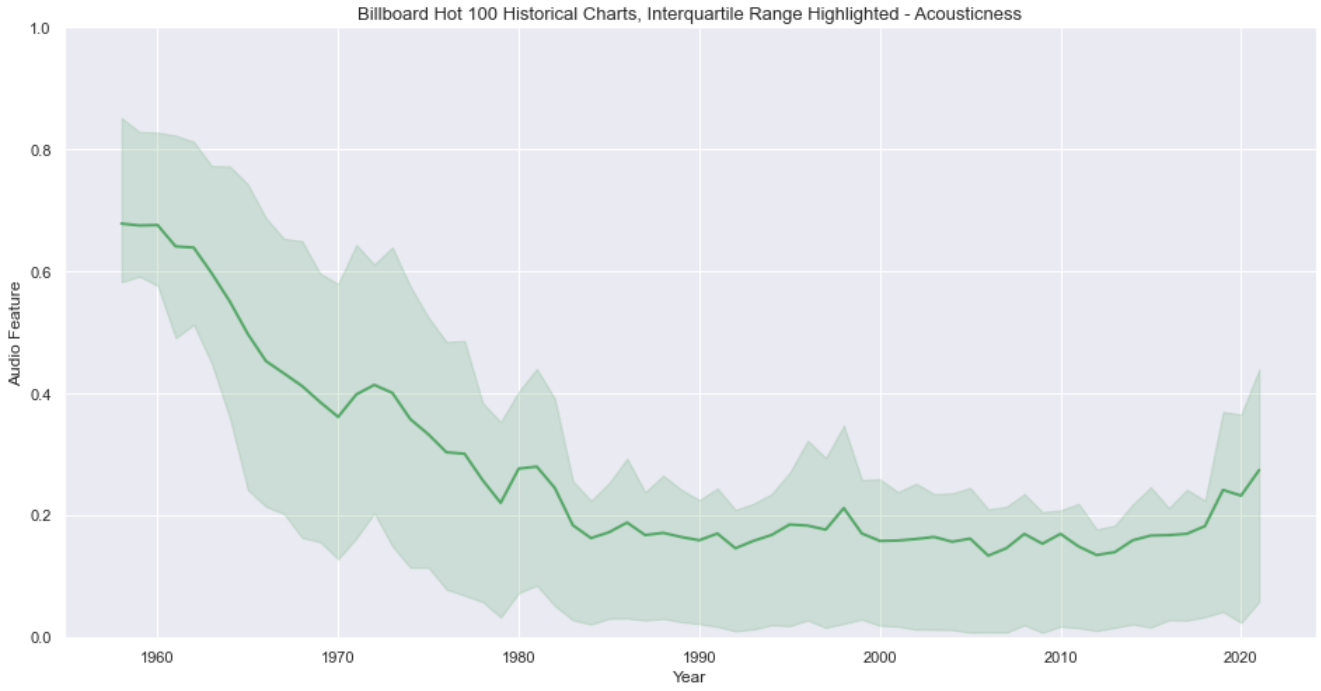
In [61]:

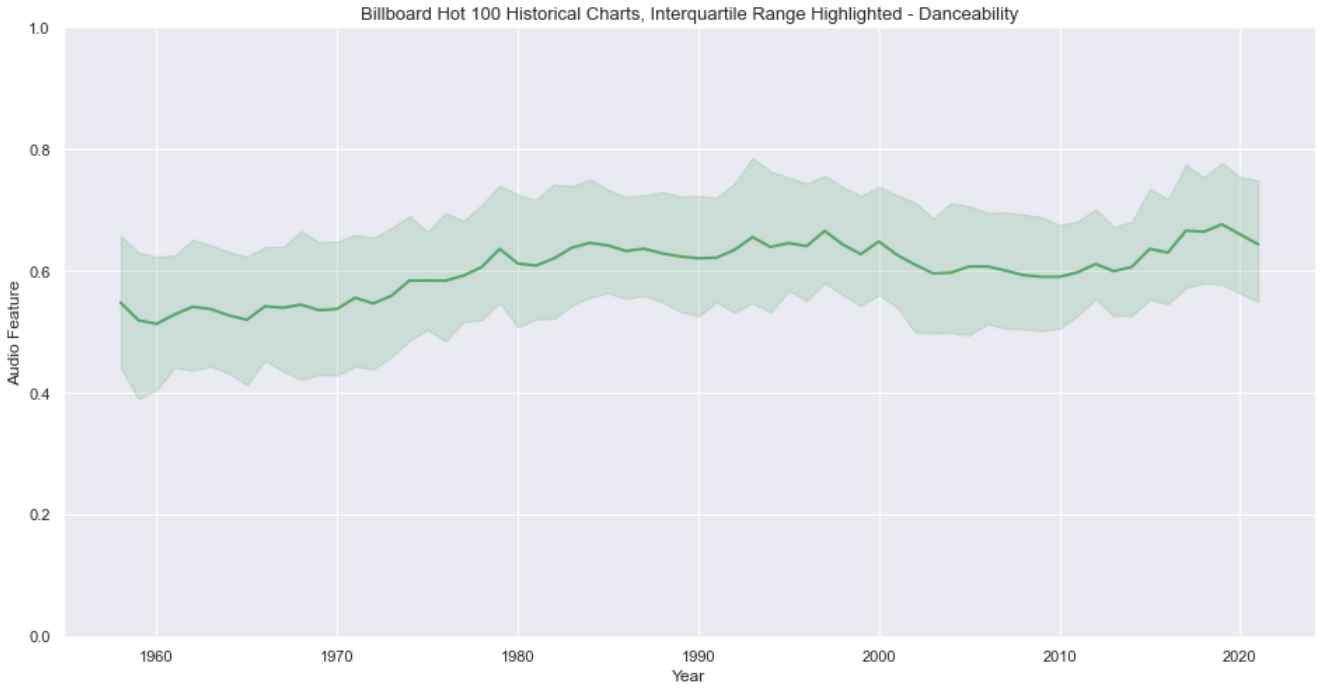
```
# plot individual features

def plot_billboard_history_by_feature(feature):
    """
    Uses songs df_B100
    Plots Billboard Hot 100 audio features over time
    """
    df = df_B100.dropna().copy()
    figsize=(16, 8)

    plt.figure(figsize=figsize)

    ax = sns.lineplot(
        x=df.date.dt.year,
        y=df[feature],
        errorbar=('pi', 50),  # IQR
        linewidth=2,
        color=sns.color_palette()[2]  # different colour to differentiate with other plots
    )

    if feature == 'loudness':
        plt.ylim(-16, 0)  # reasonable range for loudness in music
    elif feature == 'tempo':
        plt.ylim(40, 200)  # reasonable range for tempo in music
    elif feature == 'time_signature':
        plt.ylim(0, 5)
    else:
        plt.ylim(0, 1)  # most audio features vary between 0 and 1
        # NOTE: mode is an int, either 0 or 1 (minor/major),
        # but the average shows how commonly a song is in either mode

    title = 'Billboard Hot 100 Historical Charts, Interquartile Range Highlighted - {}'.format(feature.replace('_', ' ').title())

    plt.title(title, fontsize=13)
    plt.xlabel('Year')
    plt.ylabel('Audio Feature')

    # save the image
    plt.savefig(f'figures/timeseries/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')

    plt.show()



def plot_billboard_history_duration(feature='duration_ms'):
```

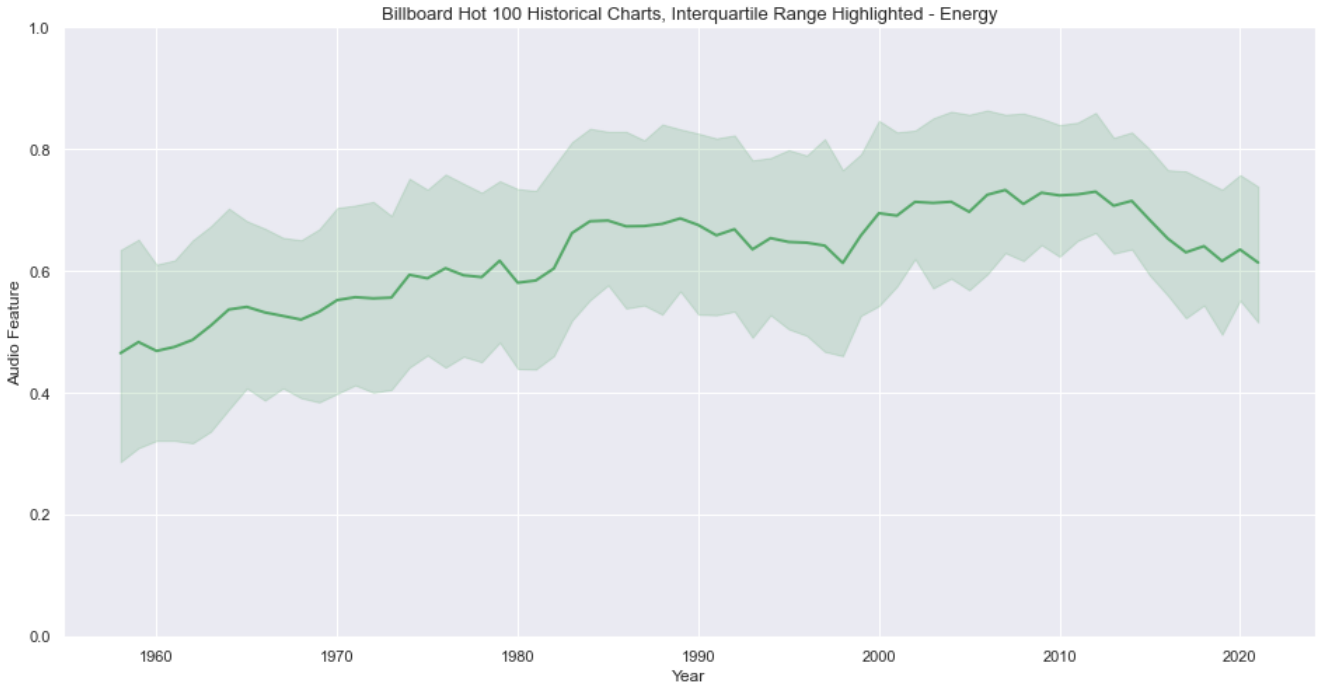
```
    df = df_B100.dropna().copy()
    figsize=(16, 8)
    plt.figure(figsize=figsize)

    ax = sns.lineplot(
        x=df.date.dt.year,
        y=df[feature],
        errorbar=('pi', 50),  # IQR
        linewidth=2,
        color=sns.color_palette()[2]  # different colour to differentiate with other plots
    )

    formatter = mpl.ticker.FuncFormatter(lambda ms, x: time.strftime('%M:%S', time.gmtime(ms // 1000)))
    ax.yaxis.set_major_formatter(formatter)
    plt.ylabel('Duration (minutes : seconds)')

    plt.ylim(120000, 300000)  # reasonable range for duration of music

    title = 'Billboard Hot 100 Historical Charts, Interquartile Range Highlighted - Song Duration'

    plt.title(title, fontsize=13)
    plt.xlabel('Year')
    plt.ylabel('Audio Feature')

    # save the image
    plt.savefig(f'figures/timeseries/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')

    plt.show()
```

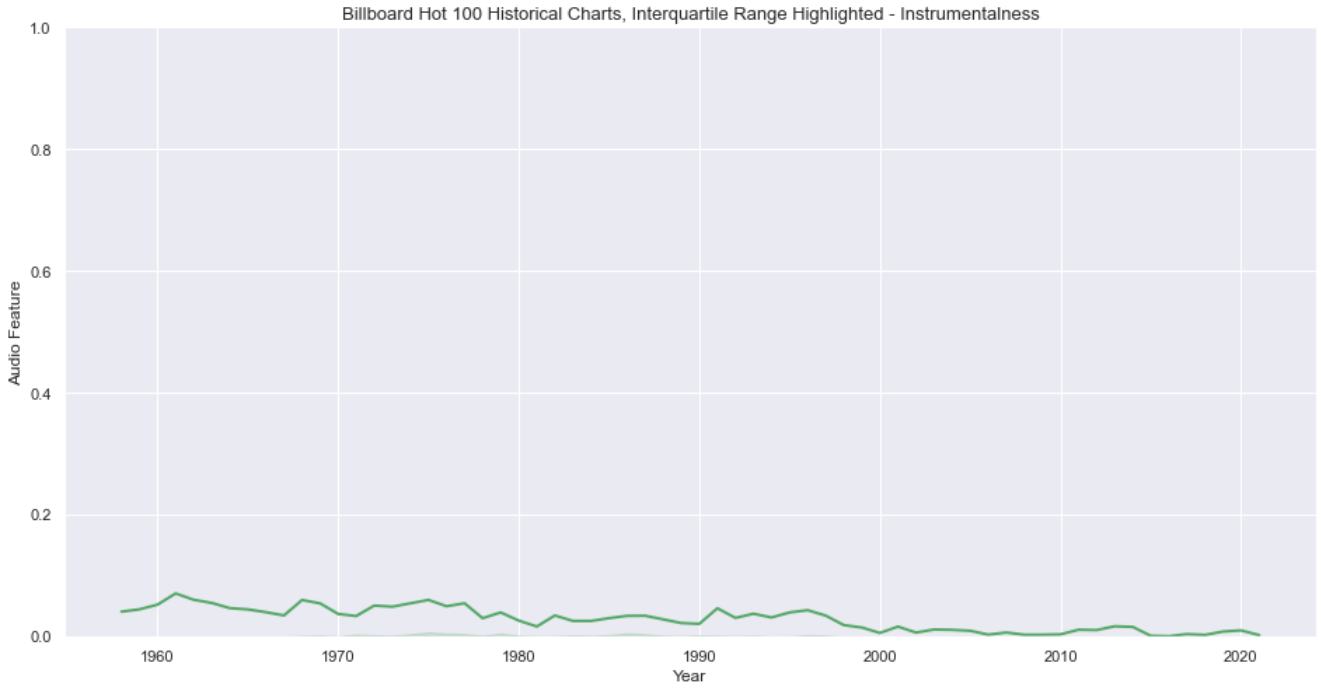
In [62]:

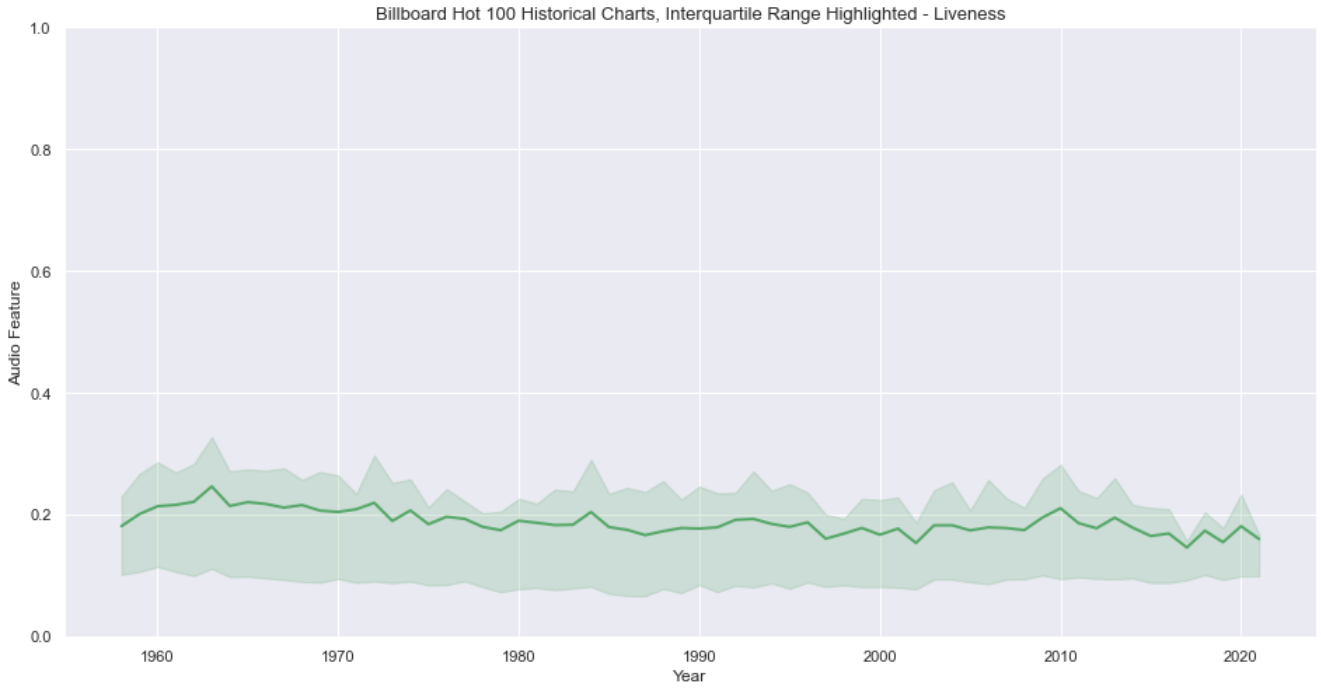
```
# make the plots
list_of_features = ['acousticness',  'danceability', 'energy', 'instrumentalness',
                    'liveness', 'loudness', 'mode', 'speechiness', 'tempo', 'time_signature', 'valence']

for feature in list_of_features:
    plot_billboard_history_by_feature(feature)

plot_billboard_history_duration()
```

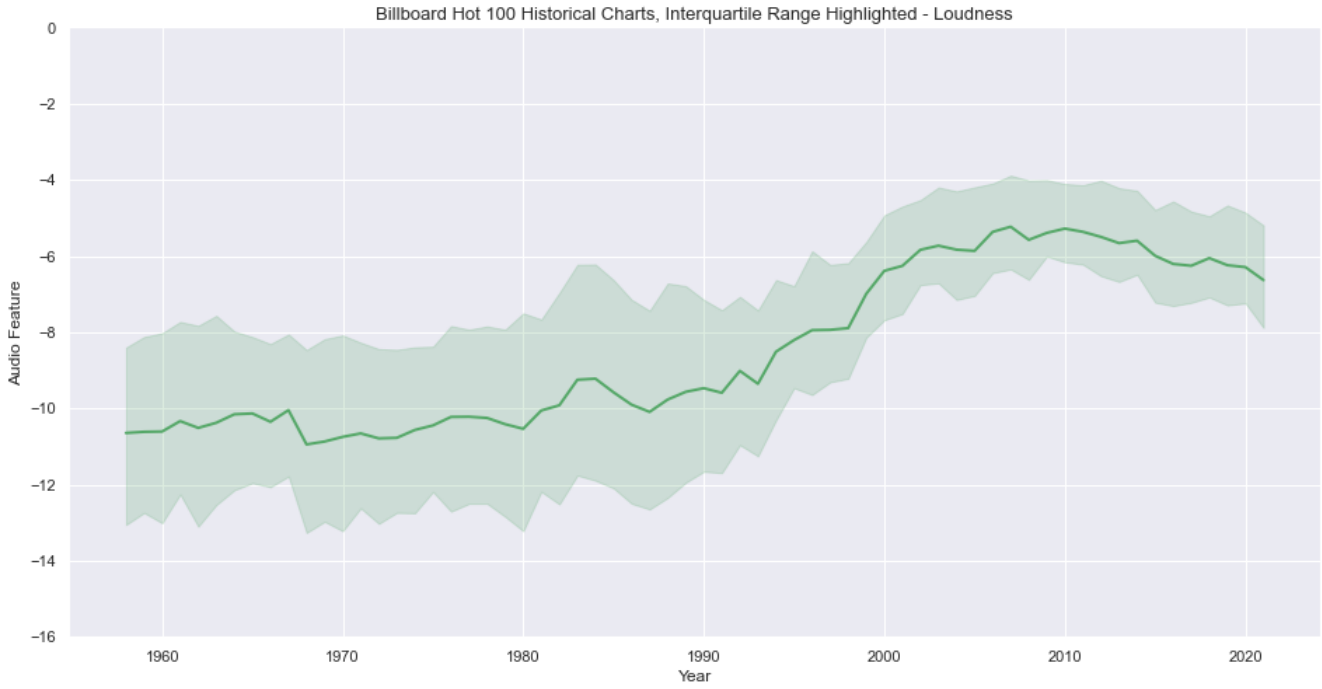


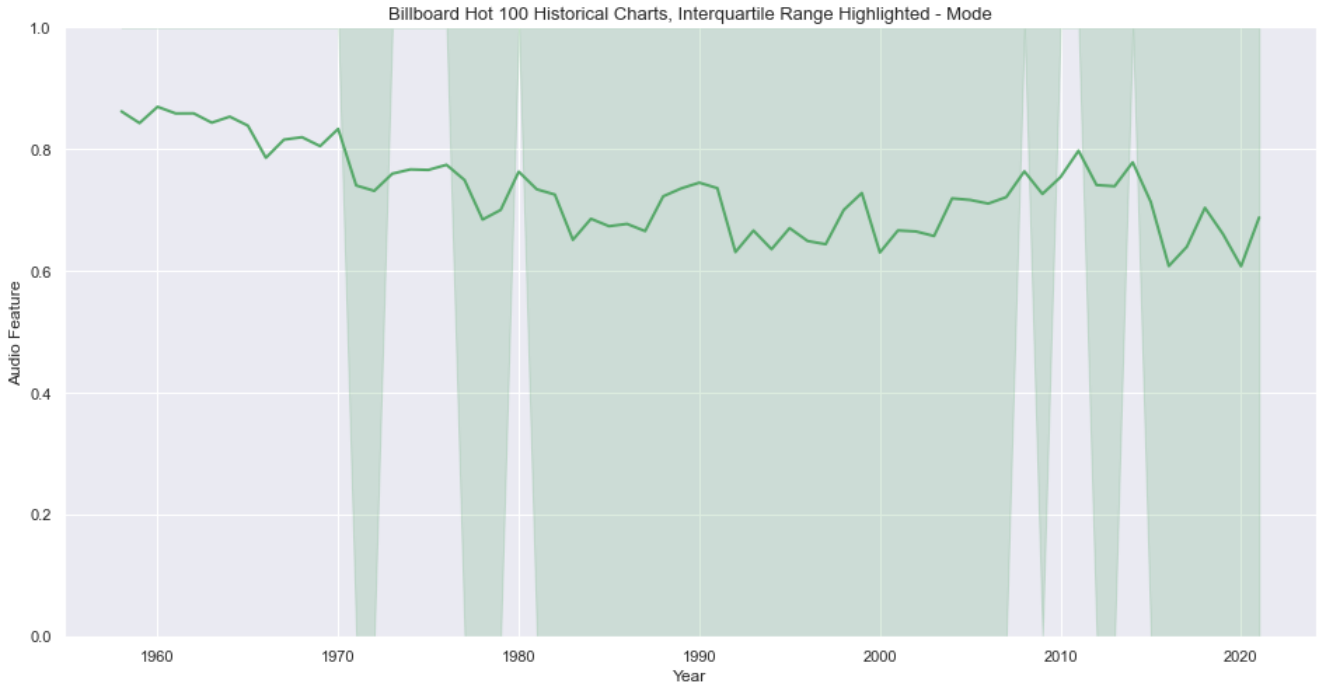

Billboard Hot 100 Historical Charts, Interquartile Range Highlighted - Danceability
Audio Feature
Year
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020

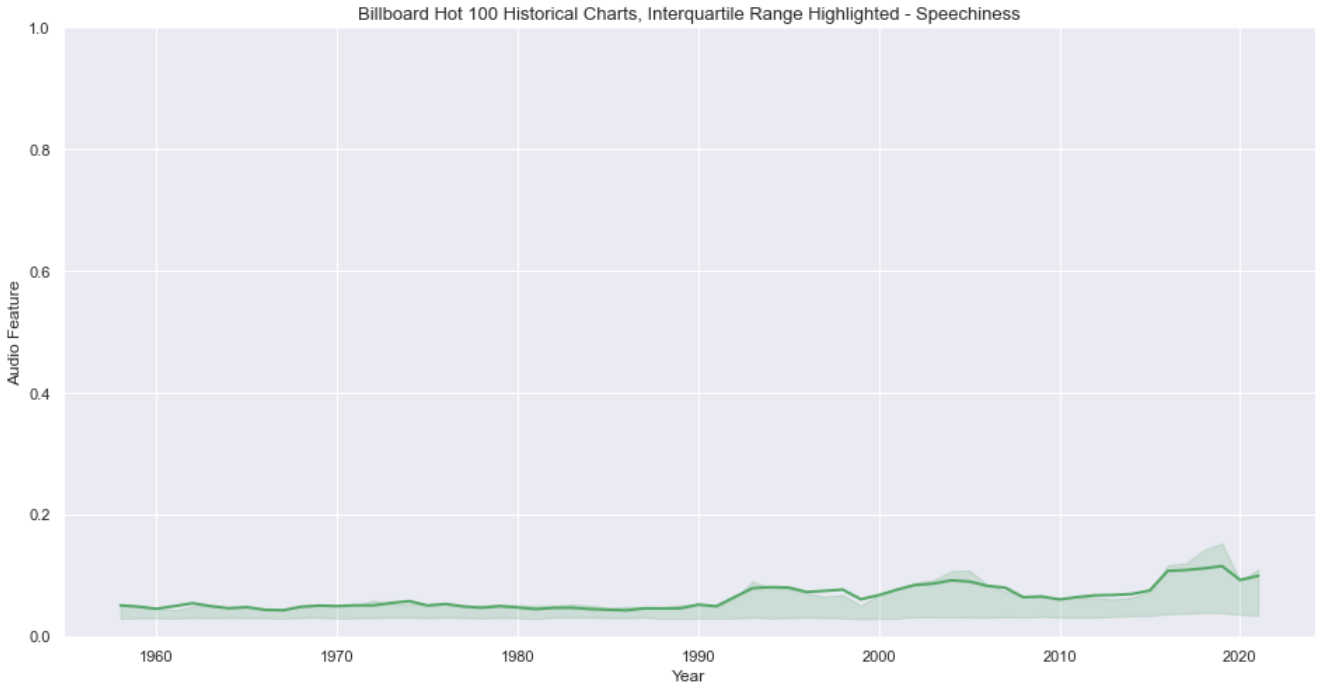

Billboard Hot 100 Historical Charts, Interquartile Range Highlighted - Energy
Audio Feature
Year
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020

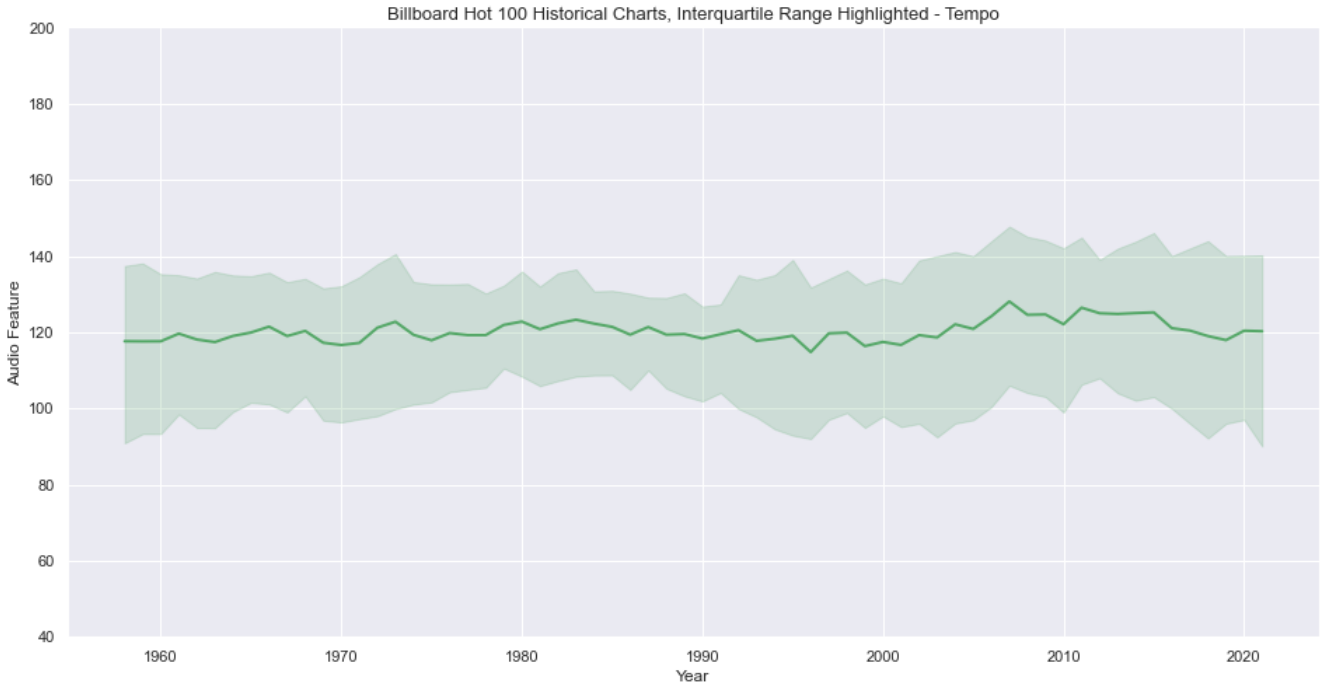

Billboard Hot 100 Historical Charts, Interquartile Range Highlighted - Instrumentalness
Audio Feature
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year

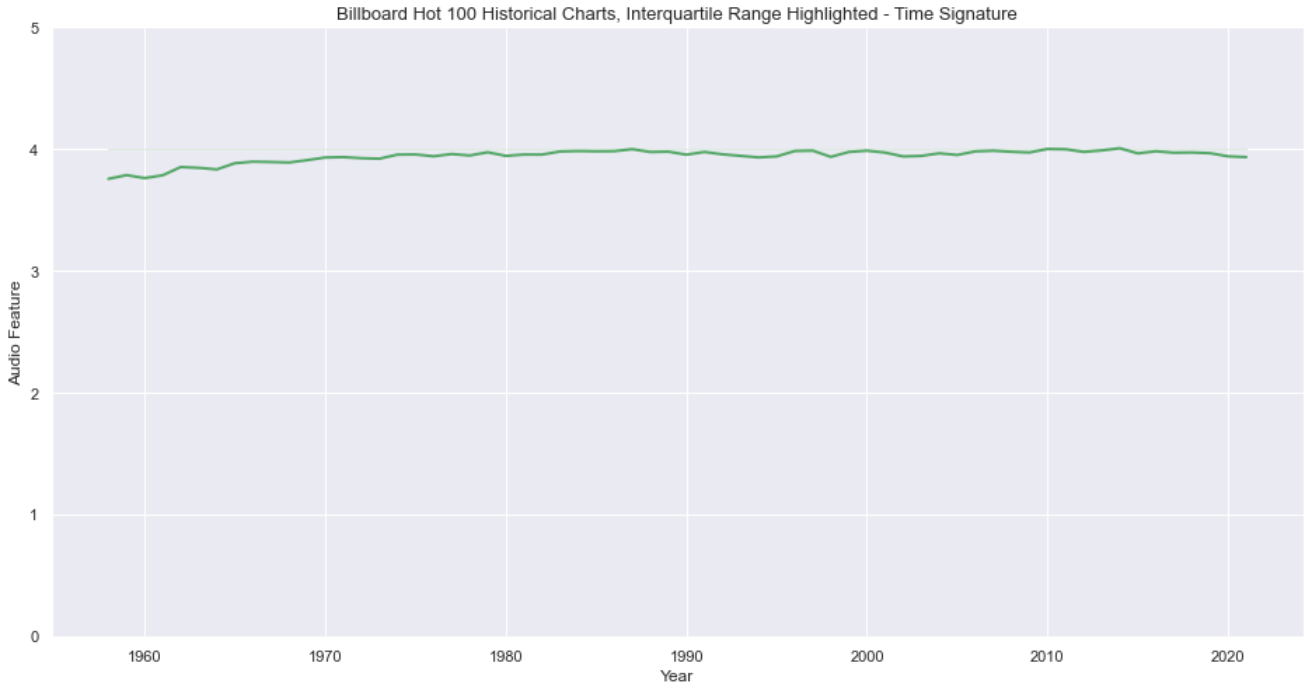

Billboard Hot 100 Historical Charts, Interquartile Range Highlighted - Liveness
Audio Feature
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year

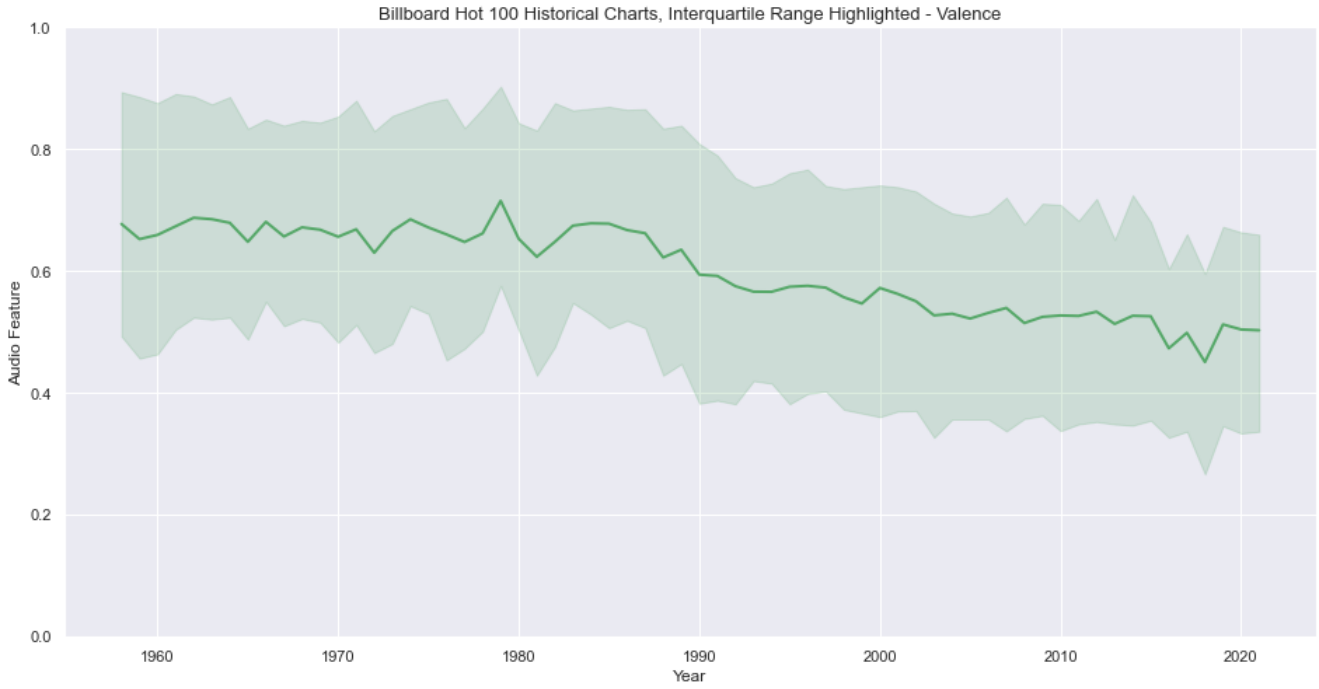

Billboard Hot 100 Historical Charts, Interquartile Range Highlighted - Loudness
0
−2
−4
−6
−8
−10
−12
−14
−16
Audio Feature
1960
1970
1980
1990
2000
2010
2020
Year

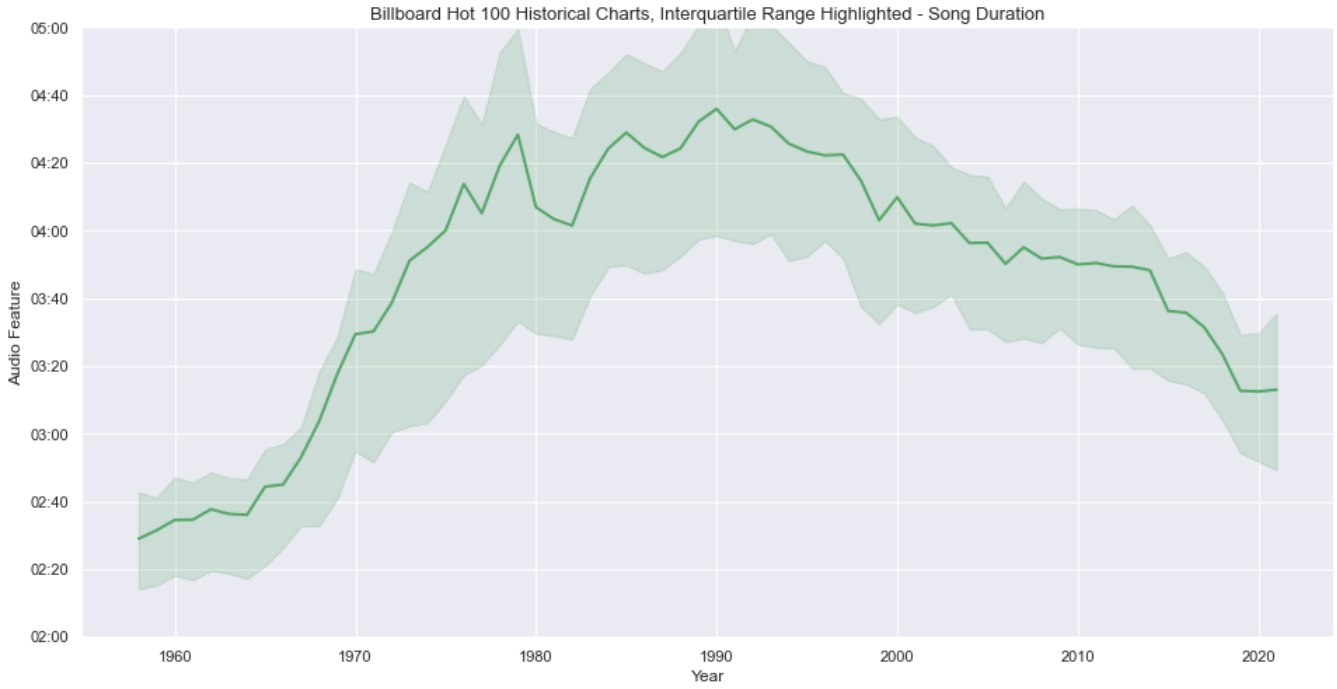

Billboard Hot 100 Historical Charts, Interquartile Range Highlighted - Mode
1.0
0.8
0.6
0.4
0.2
0.0
Audio Feature
1960
1970
1980
1990
2000
2010
2020
Year


Billboard Hot 100 Historical Charts, Interquartile Range Highlighted - Speechiness
Audio Feature
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Year


Billboard Hot 100 Historical Charts, Interquartile Range Highlighted - Tempo
Audio Feature
200
180
160
140
120
100
80
60
40
1960
1970
1980
1990
2000
2010
2020
Year


Billboard Hot 100 Historical Charts, Interquartile Range Highlighted - Time Signature
Audio Feature
Year
0
1
2
3
4
5
1960
1970
1980
1990
2000
2010
2020

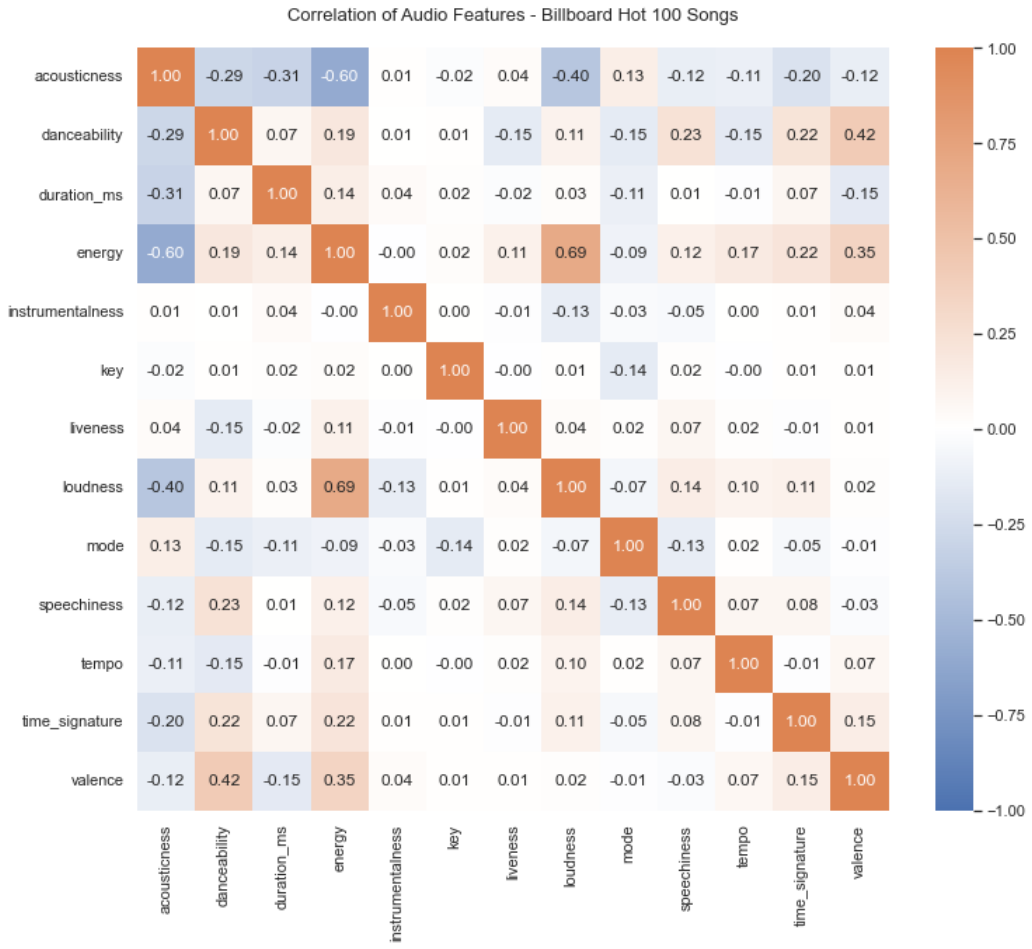

Billboard Hot 100 Historical Charts, Interquartile Range Highlighted - Valence
Audio Feature
Year
0.0
0.2
0.4
0.6
0.8
1.0
1960
1970
1980
1990
2000
2010
2020



Correlation Analysis

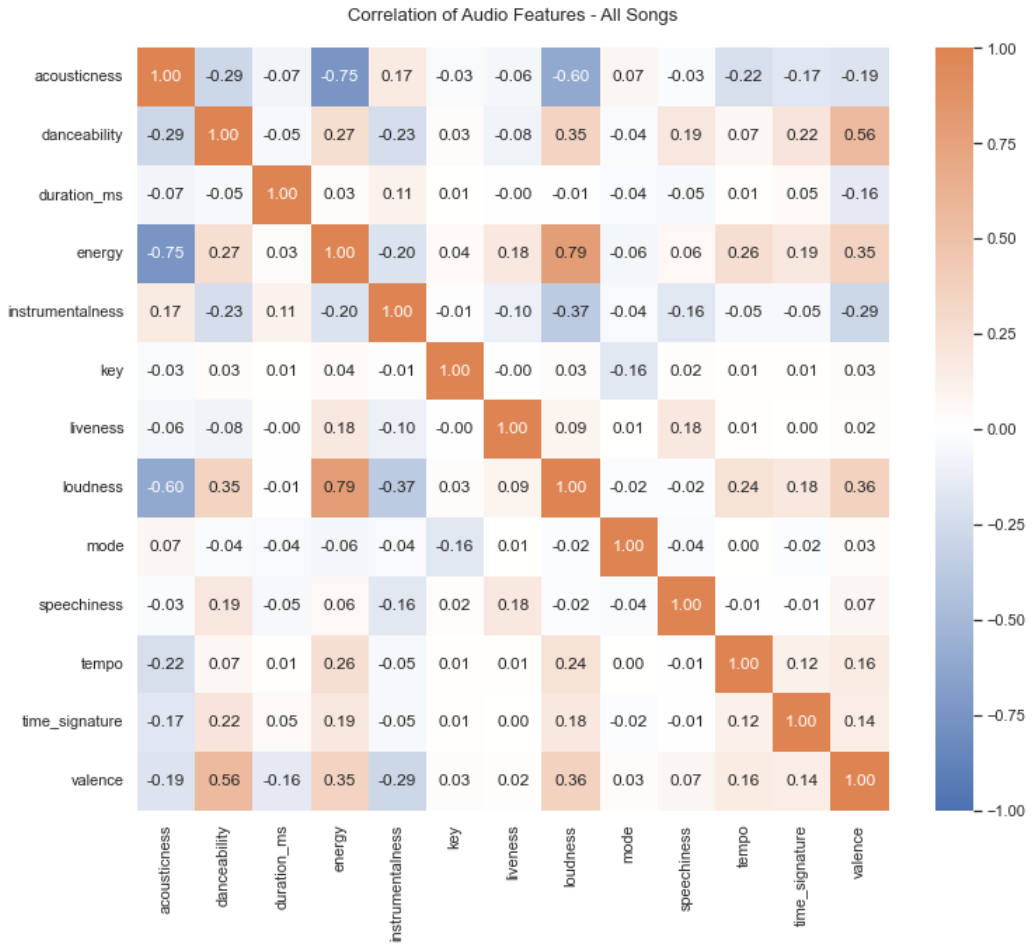
In [11]:

```
# correlation matrix - Billboard Hot 100
corr_B100 = df_B100_songs_AF.corr()

# plot
palette = sns.blend_palette([sns.color_palette()[0], '#ffffff', sns.color_palette()[1]], 3, as_cmap=True)
plt.subplots(figsize=(12, 10))
sns.heatmap(corr_B100, cmap=palette, center=0, vmin=-1, vmax=1, annot=True, fmt='.2f')
title='Correlation of Audio Features - Billboard Hot 100 Songs'
plt.title(title, fontsize=13, pad=20)

# save the image
plt.savefig(f'figures/correlation/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')

plt.show()
```



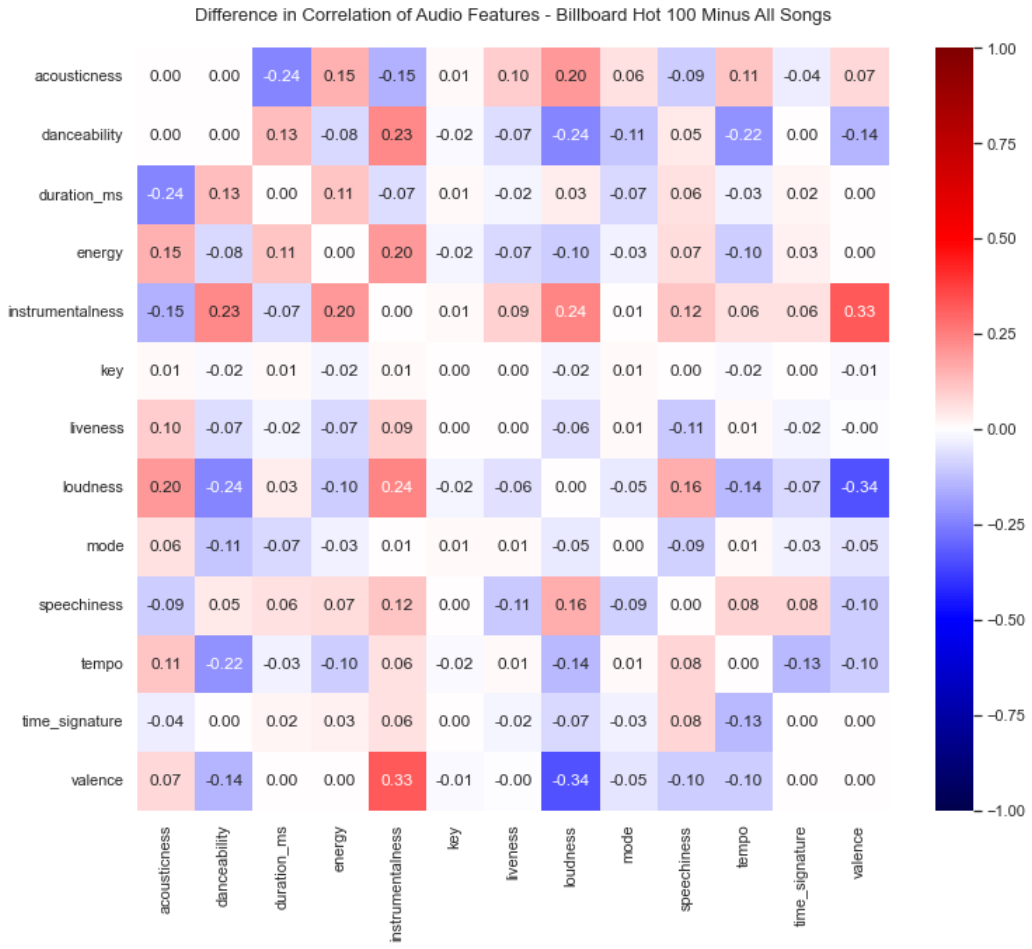
In [12]:

```
# correlation matrix - All Songs
corr_10M = df_10M.corr()

# plot
palette = sns.blend_palette([sns.color_palette()[0], '#ffffff', sns.color_palette()[1]], 3, as_cmap=True)
plt.subplots(figsize=(12, 10))
sns.heatmap(corr_10M, cmap=palette, center=0, vmin=-1, vmax=1, annot=True, fmt='.2f')
title='Correlation of Audio Features - All Songs'
plt.title(title, fontsize=13, pad=20)

# save the image
plt.savefig(f'figures/correlation/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')

plt.show()
```



In [13]:

```
# differences between Billboard Hot 100 and All Songs
corr_diff = corr_B100 - corr_10M

# plot
palette = 'seismic'  # to differentiate from standard colour palette
plt.subplots(figsize=(12, 10))
sns.heatmap(corr_diff, cmap=palette, center=0, vmin=-1, vmax=1, annot=True, fmt='.2f')
title='Difference in Correlation of Audio Features - Billboard Hot 100 Minus All Songs'
plt.title(title, fontsize=13, pad=20)

# save the image
plt.savefig(f'figures/correlation/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')

plt.show()
```



In [11]:

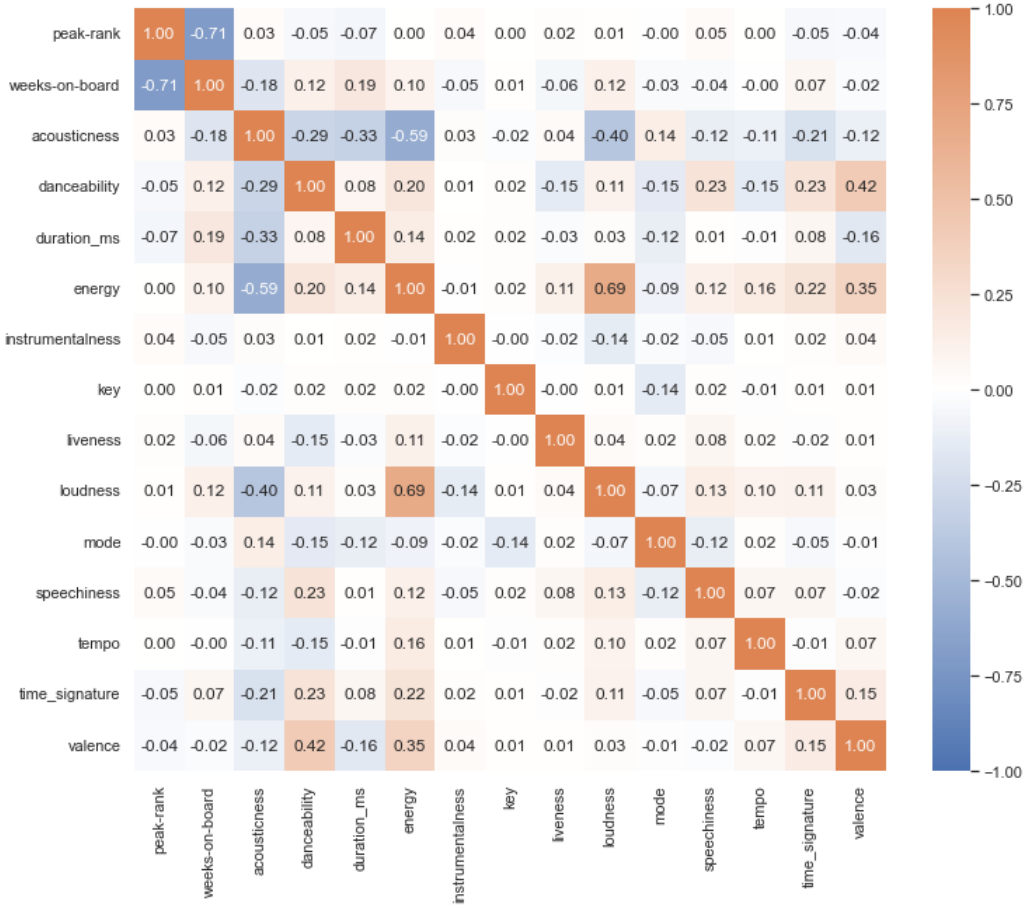
```
# Billboard 100 List - correlation with max weeks on board
df_B100_SORTED = df_B100.sort_values('weeks-on-board', ascending=False).drop_duplicates(subset=['song', 'artist']).drop(['date', 'id', 'rank',
df_B100_SORTED.head(10)
```

Out[11]:

	song	artist	peak-rank	weeks-on-board	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	tempo
0	Blinding Lights	The Weeknd	1	90	0.001	0.514	200040	0.730	0.000	1	0.090	-5.934	1	0.060	171.005
1	Radioactive	Imagine Dragons	3	87	0.106	0.448	186813	0.784	0.000	9	0.668	-3.686	1	0.063	136.245
2	Sail	AWOLNATION	17	79	0.441	0.826	259093	0.436	0.615	1	0.096	-9.583	1	0.056	119.051
3	I'm Yours	Jason Mraz	6	76	0.957	0.549	219053	0.107	0.000	0	0.111	-16.147	1	0.084	74.630
4	How Do I Live	LeAnn Rimes	2	69	0.128	0.577	266973	0.462	0.000	2	0.082	-7.989	1	0.028	128.303
5	Counting Stars	OneRepublic	2	68	0.065	0.664	257839	0.705	0.000	1	0.115	-4.972	0	0.038	122.017
6	Party Rock Anthem	LMFAO Featuring Lauren Bennett & GoonRock	1	68	NaN	NaN	<NA>	NaN	NaN	<NA>	NaN	NaN	<NA>	NaN	NaN
7	Foolish Games/You Were Meant For Me	Jewel	2	65	NaN	NaN	<NA>	NaN	NaN	<NA>	NaN	NaN	<NA>	NaN	NaN
8	Rolling In The Deep	Adele	1	65	0.138	0.730	228093	0.769	0.000	8	0.047	-5.114	1	0.030	104.948
9	Before He Cheats	Carrie Underwood	8	64	0.271	0.519	199946	0.749	0.000	6	0.119	-3.318	0	0.041	147.905

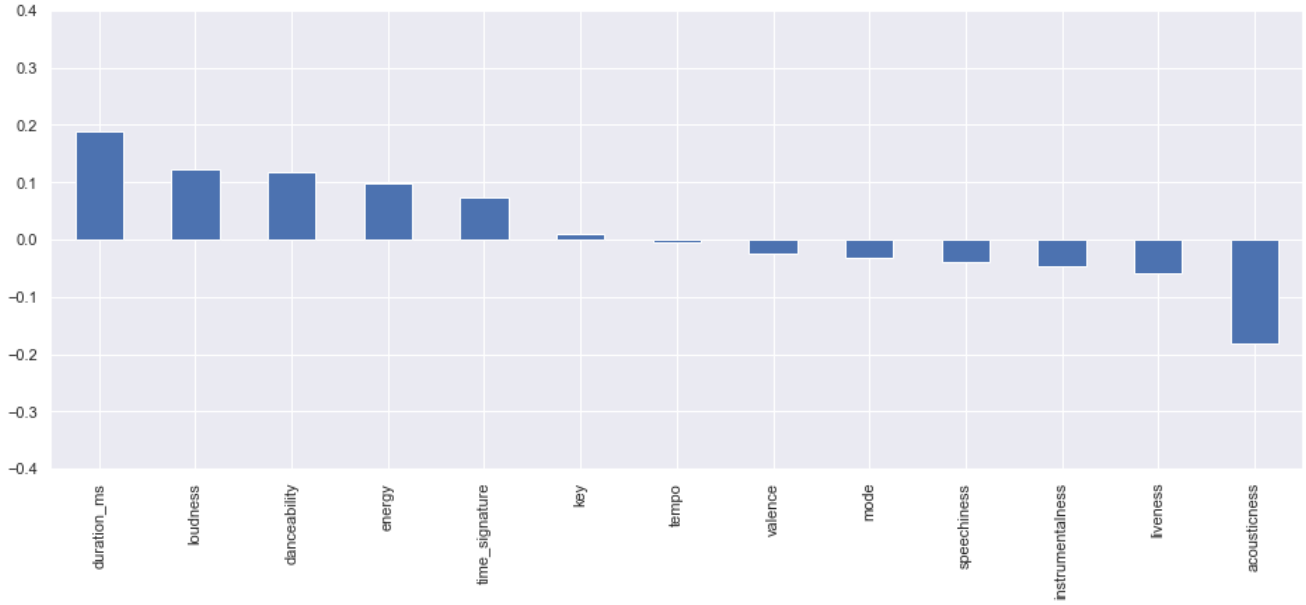
In [12]:

```
# correlation matrix - All Songs
corr_billboard_weeks = df_B100_SORTED.corr()

# plot
palette = sns.blend_palette([sns.color_palette()[0], '#ffffff', sns.color_palette()[1]], 3, as_cmap=True)
plt.subplots(figsize=(12, 10))
sns.heatmap(corr_billboard_weeks, cmap=palette, center=0, vmin=-1, vmax=1, annot=True, fmt='.2f')
title='Correlation of Billboard Rankings to Audio Features'
# plt.title(title, fontsize=13, pad=20)

# save the image
plt.savefig(f'figures/correlation/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')

plt.show()
```



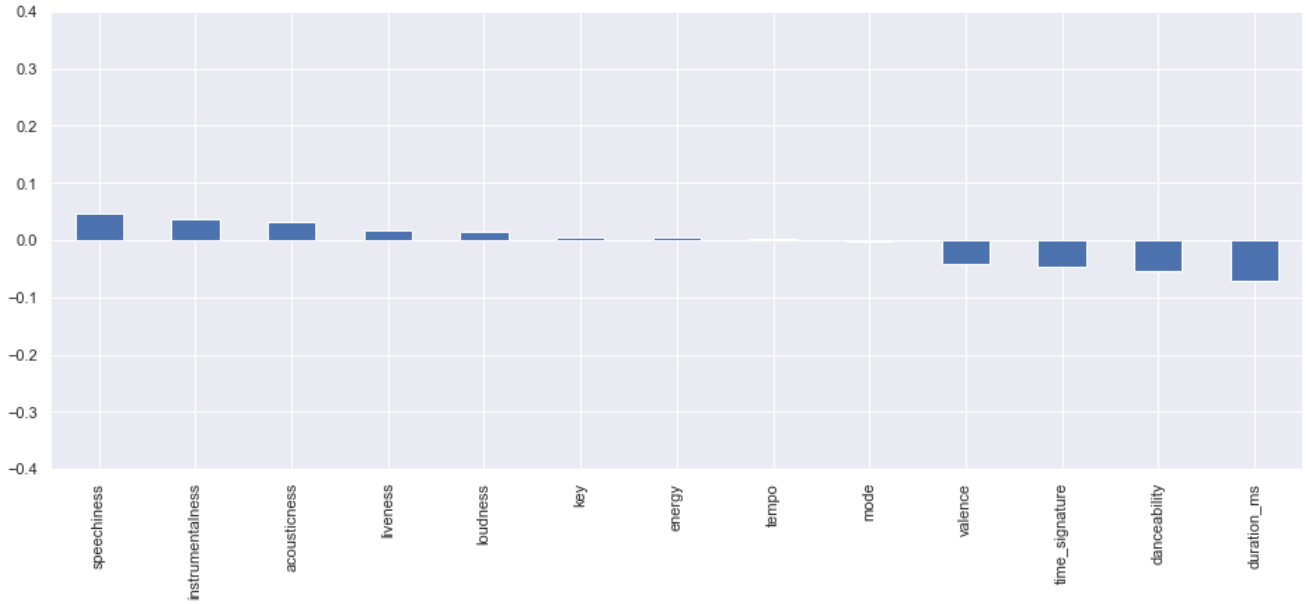
In [13]:

```
# correlated with weeks on board in Billboard Hot 100
corr_billboard_weeks['weeks-on-board'].sort_values(ascending=False)[1:-1].plot(kind='bar', figsize=(16,6))
title='Correlation of Weeks on Billboard Hot 100 vs Audio Features'
# plt.title(title, fontsize=13, pad=20)
plt.ylim(-0.4, 0.4)

# save the image
plt.savefig(f'figures/correlation/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')

plt.show()
```



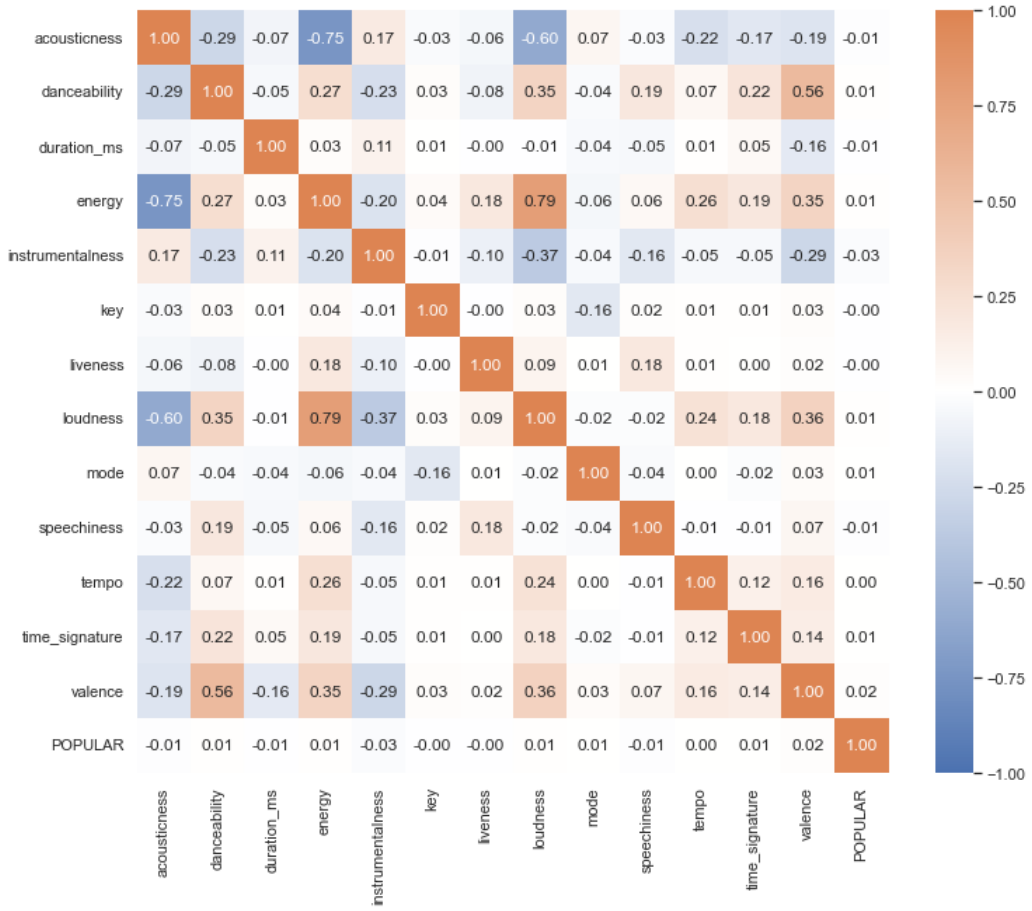
In [14]:

```
# correlated with peak rank in Billboard Hot 100
corr_billboard_weeks['peak-rank'].sort_values(ascending=False)[1:-1].plot(kind='bar', figsize=(16,6))
title='Correlation of Peak Rank on Billboard Hot 100 vs Audio Features'
# plt.title(title, fontsize=13, pad=20)
plt.ylim(-0.4, 0.4)

# save the image
plt.savefig(f'figures/correlation/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')

plt.show()
```



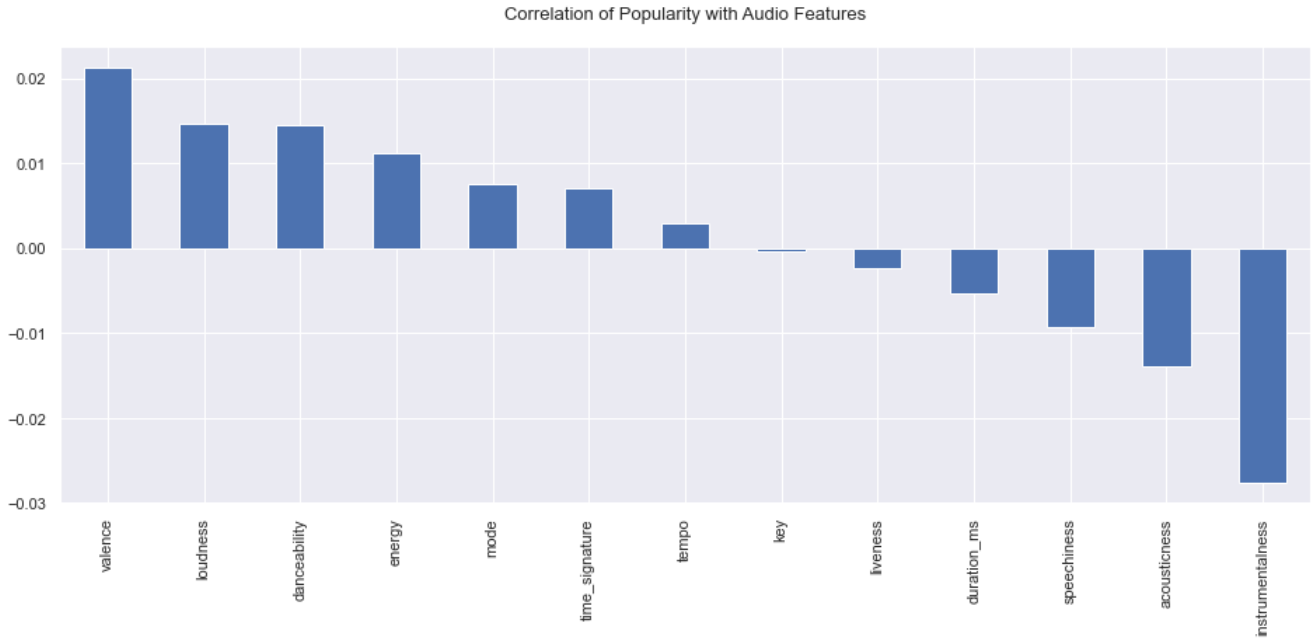
In [9]:

```
# correlation matrix - popular or not
# Note: "POPULAR" simply denotes whether or not a song appeared on the Billboard Hot 100
corr_popular = df_popularity.corr()

# plot
palette = sns.blend_palette([sns.color_palette()[0], '#ffffff', sns.color_palette()[1]], 3, as_cmap=True)
plt.subplots(figsize=(12, 10))
sns.heatmap(corr_popular, cmap=palette, center=0, vmin=-1, vmax=1, annot=True, fmt='.2f')
title='Correlation of Popularity with Audio Features'
# plt.title(title, fontsize=13, pad=20)

# save the image
plt.savefig(f'figures/correlation/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')

plt.show()
```



In [19]:

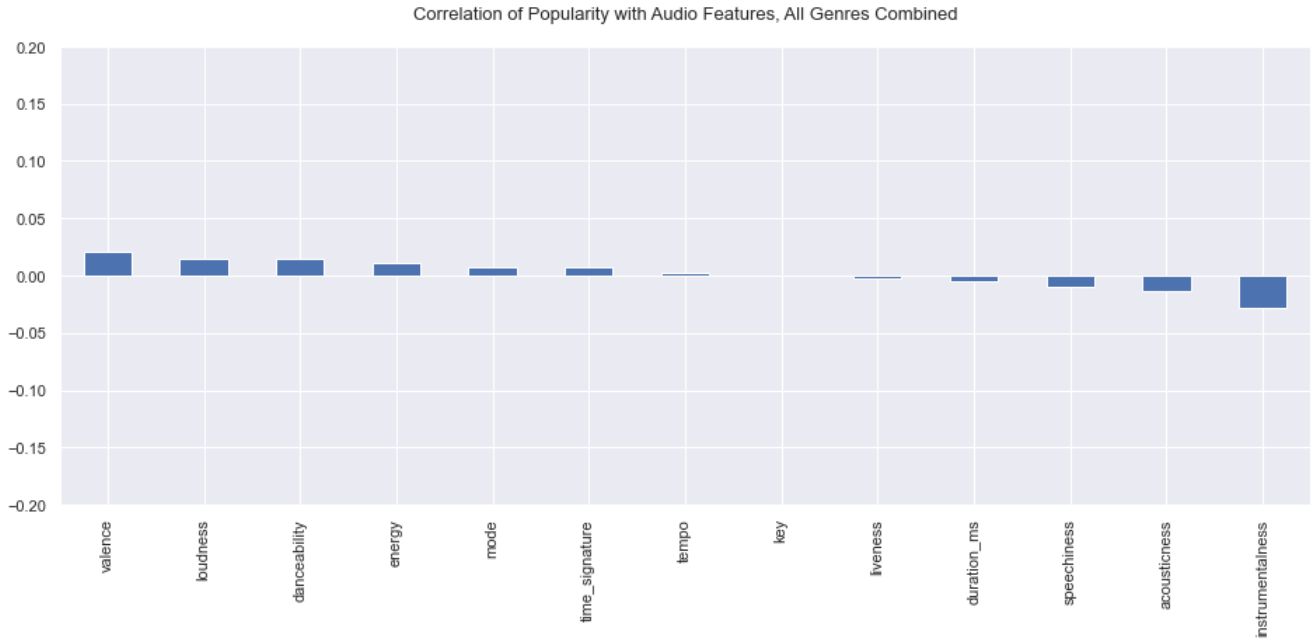
```
# correlated with peak rank in Billboard Hot 100
corr_popular['POPULAR'].sort_values(ascending=False)[1:].plot(kind='bar', figsize=(16,6))
title='Correlation of Popularity with Audio Features'
plt.title(title, fontsize=13, pad=20)

# save the image
plt.savefig(f'figures/correlation/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')

plt.show()

corr_popular['POPULAR'].sort_values(ascending=False)[1:].plot(kind='bar', figsize=(16,6))
title='Correlation of Popularity with Audio Features (Full Scale)'
plt.title(title, fontsize=13, pad=20)
plt.ylim(-1, 1)

# save the image
plt.savefig(f'figures/correlation/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')

plt.show()
```

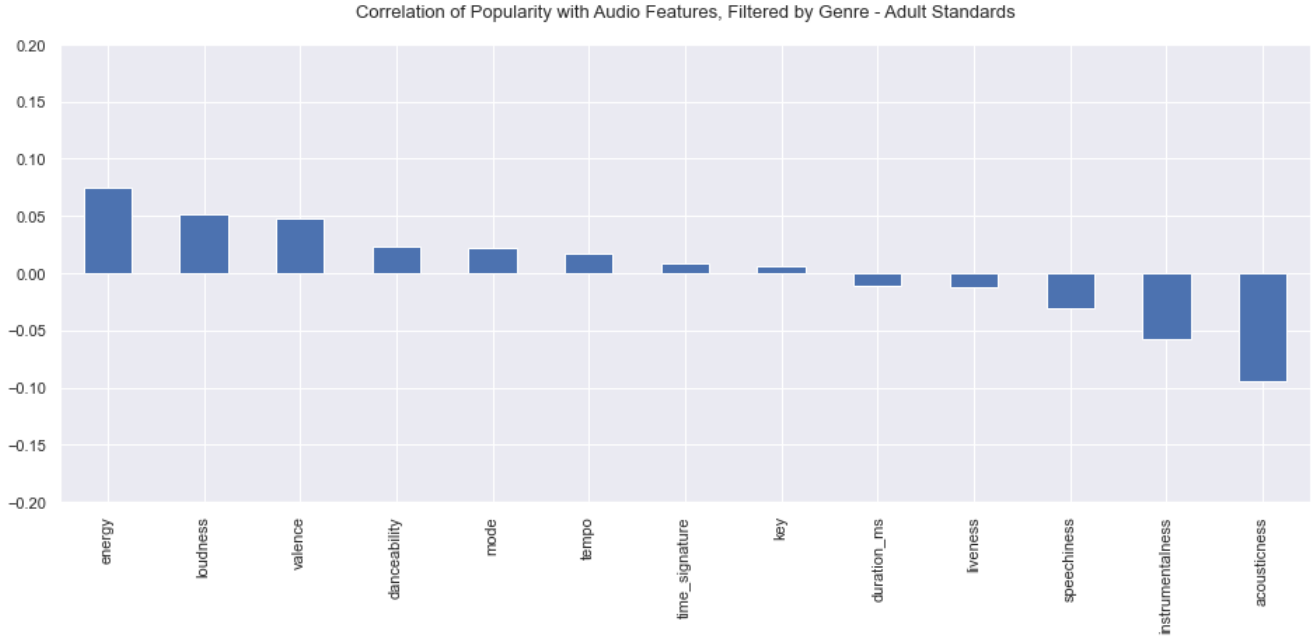


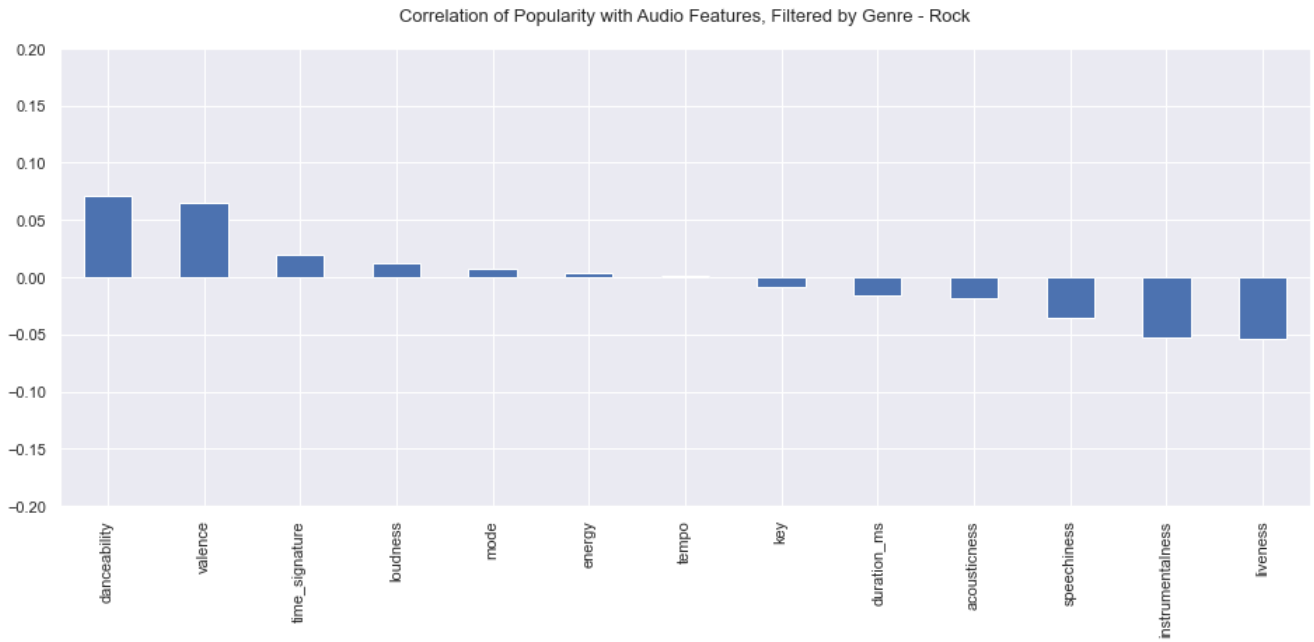
In [20]:

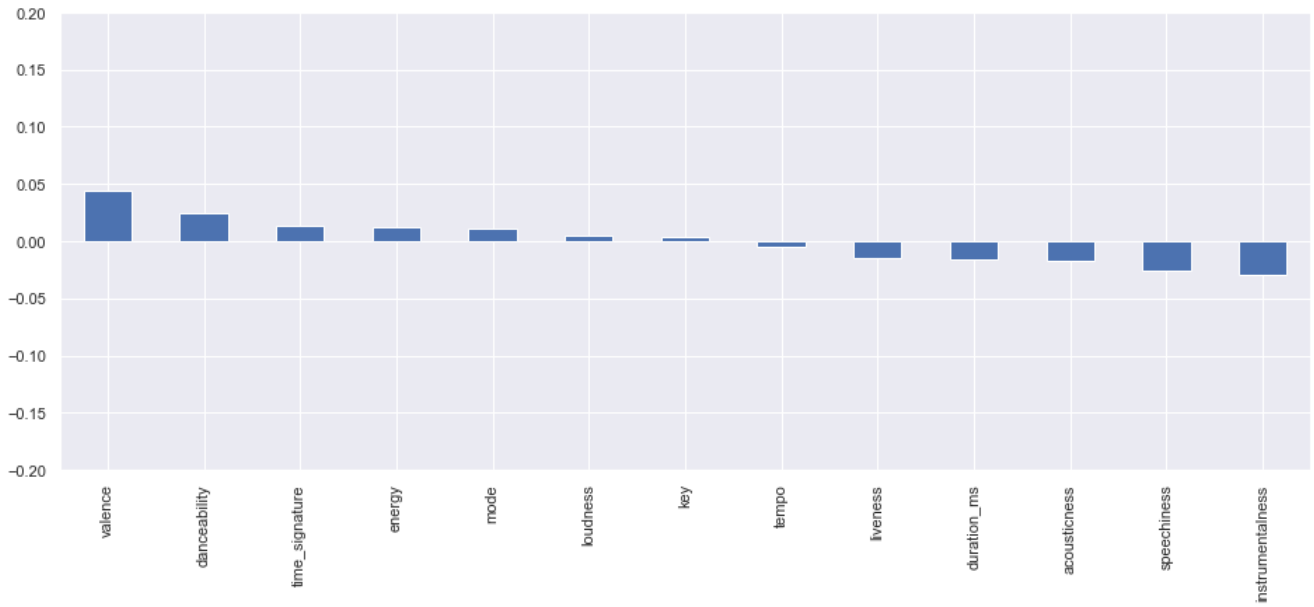
```
# top 10 genres in Billboard Hot 100
genres = list(df_B100_songs_AF.genre.value_counts().head(10).index)

# shared y limits
ylim = -0.2, 0.2

# all genres
df_popularity.corr()['POPULAR'].sort_values(ascending=False)[1:].plot(kind='bar', figsize=(16,6))
title='Correlation of Popularity with Audio Features, All Genres Combined'
plt.title(title, fontsize=13, pad=20)
plt.ylim(*ylim)

# save the image
plt.savefig(f'figures/correlation/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')

plt.show()

# separated genres
for genre in genres:
    df_popularity.query(f'genre == "{genre}"').corr()['POPULAR'].sort_values(ascending=False)[1:].plot(kind='bar', figsize=(16,6))
    title = f'Correlation of Popularity with Audio Features, Filtered by Genre - {genre.title()}'
    plt.title(title, fontsize=13, pad=20)
    plt.ylim(*ylim)

    # save the image
    plt.savefig(f'figures/correlation/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')
```

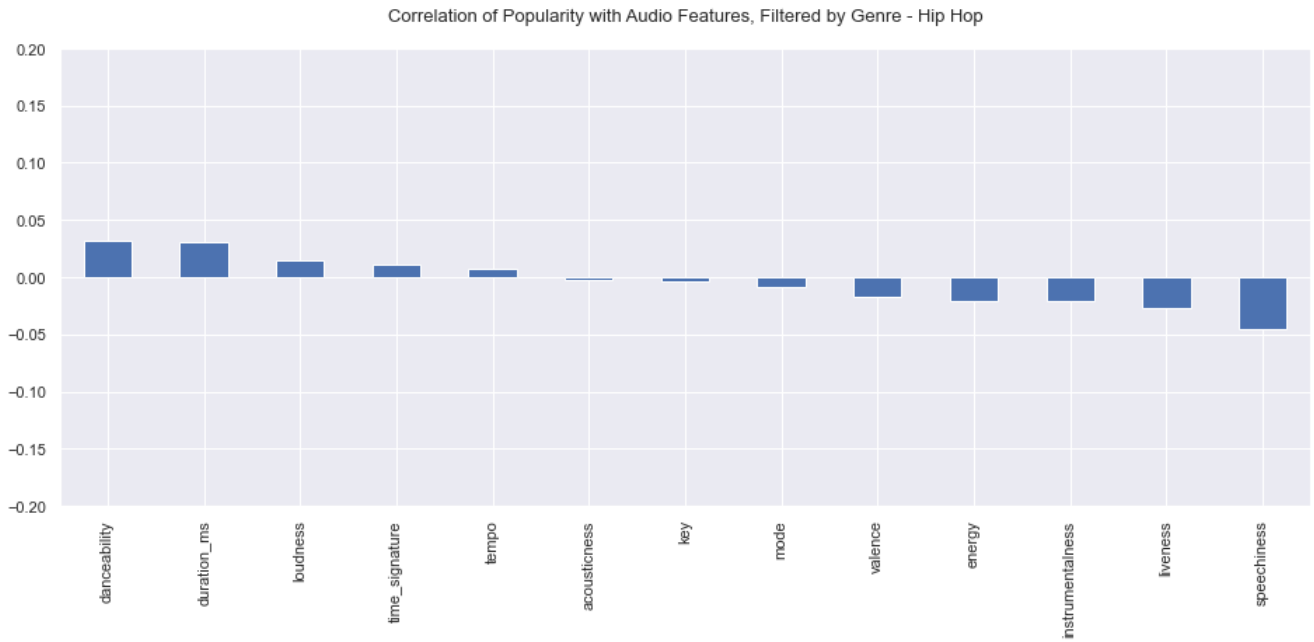
```
plt.show()
```

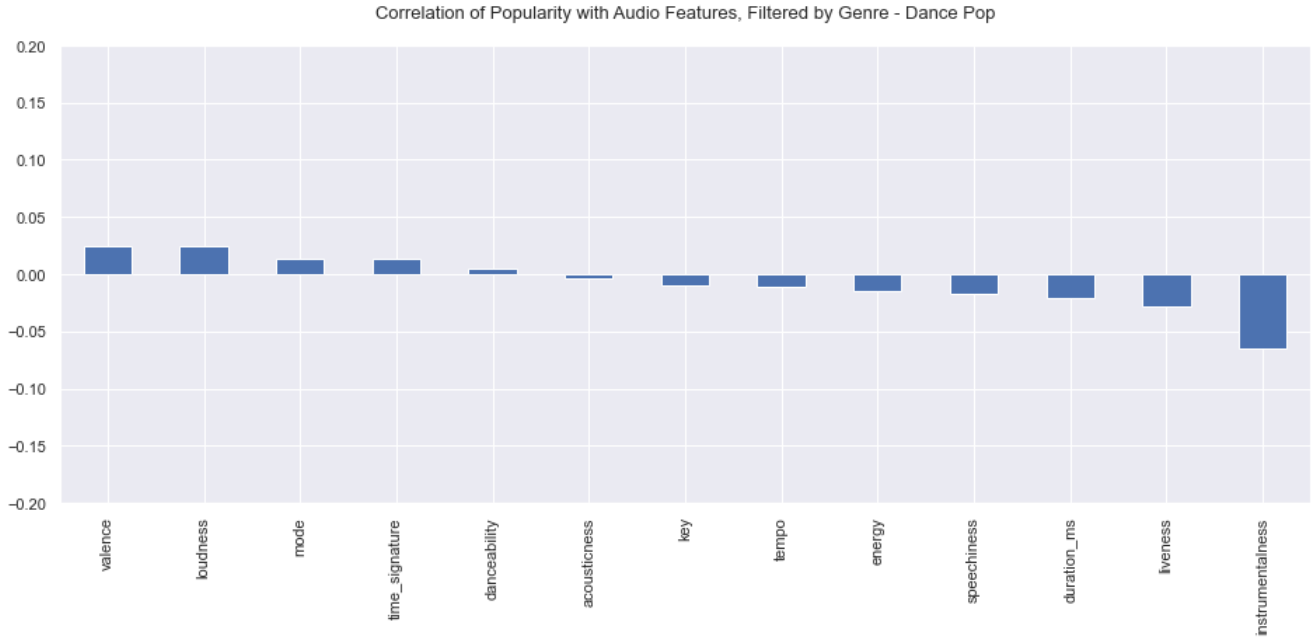

Correlation of Popularity with Audio Features, All Genres Combined
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
valence
loudness
danceability
energy
mode
time_signature
tempo
key
liveness
duration_ms
speechiness
acousticness
instrumentalness

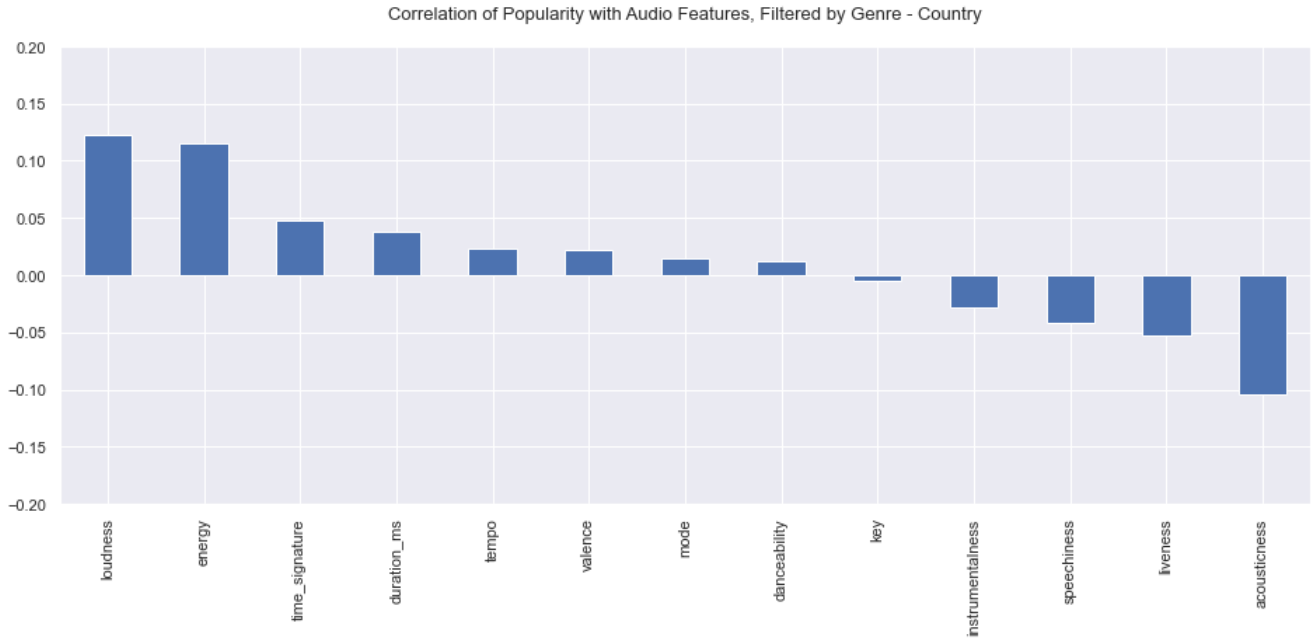

Correlation of Popularity with Audio Features, Filtered by Genre - Adult Standards
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
energy
loudness
valence
danceability
mode
tempo
time_signature
key
duration_ms
liveness
speechiness
instrumentalness
acousticness

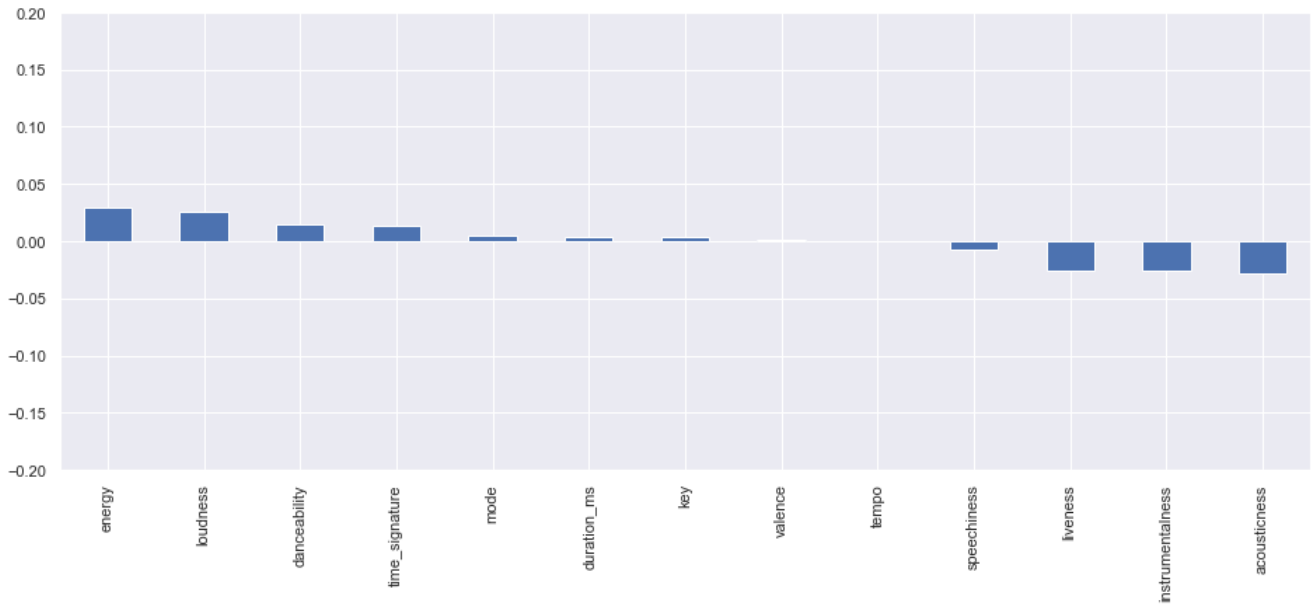

Correlation of Popularity with Audio Features, Filtered by Genre - Rock
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
danceability
valence
time_signature
loudness
mode
energy
tempo
key
duration_ms
acousticness
speechiness
instrumentalness
liveness

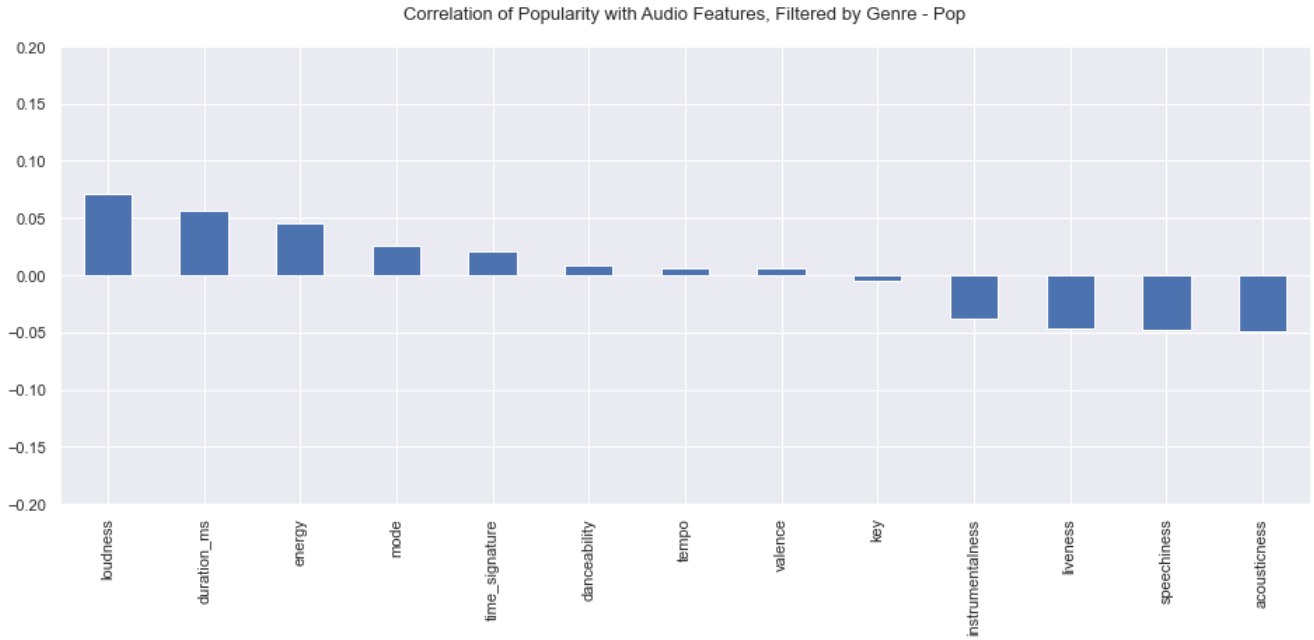

Correlation of Popularity with Audio Features, Filtered by Genre - Soul
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
valence
danceability
time_signature
energy
mode
loudness
key
tempo
liveness
duration_ms
acousticness
speechiness
instrumentalness

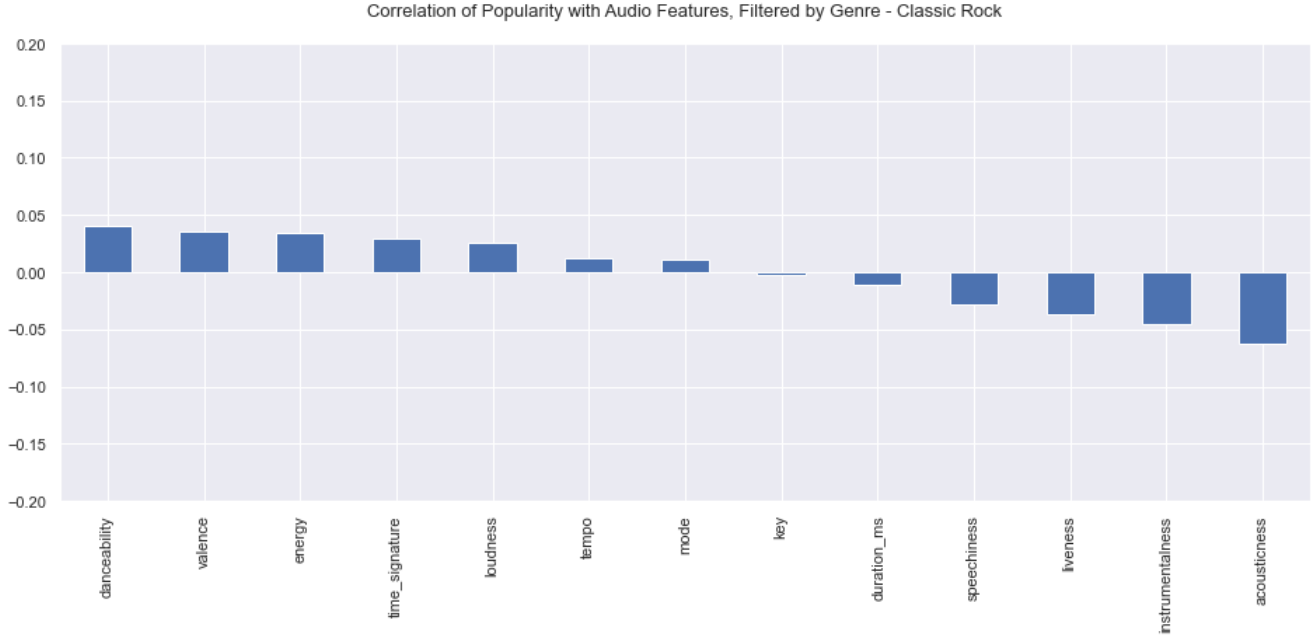

Correlation of Popularity with Audio Features, Filtered by Genre - Hip Hop
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
danceability
duration_ms
loudness
time_signature
tempo
acousticness
key
mode
valence
energy
instrumentalness
liveness
speechiness

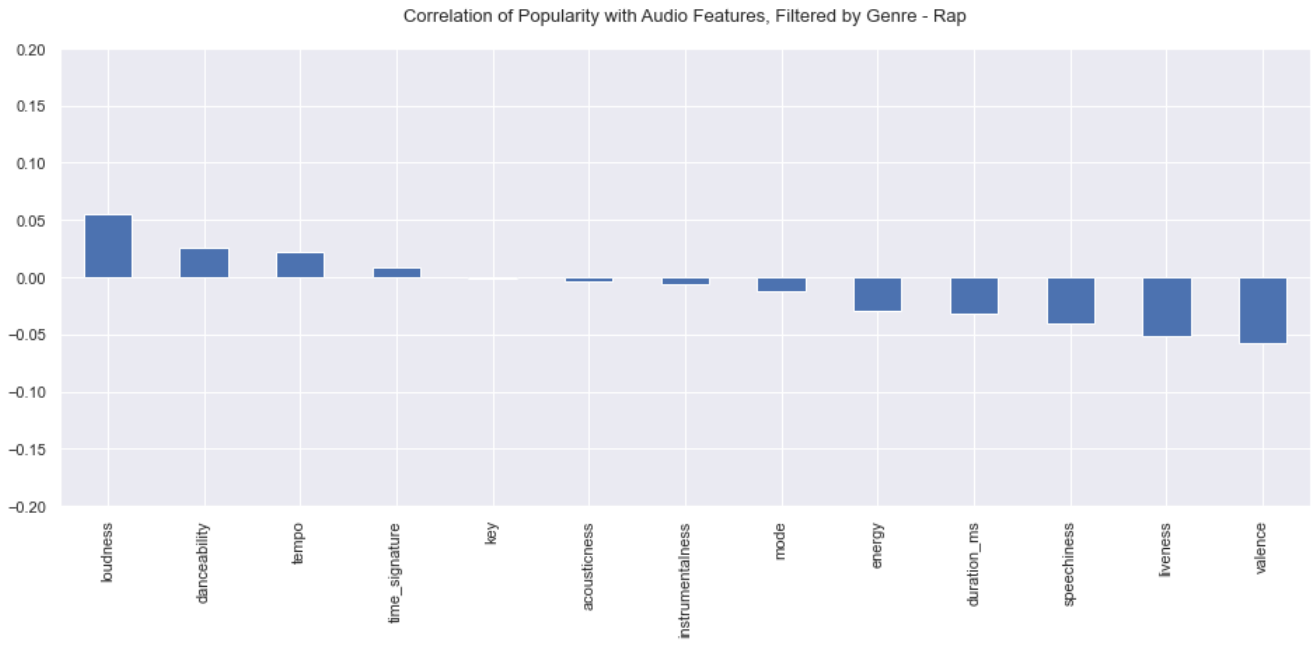

Correlation of Popularity with Audio Features, Filtered by Genre - Dance Pop
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
valence
loudness
mode
time_signature
danceability
acousticness
key
tempo
energy
speechiness
duration_ms
liveness
instrumentalness


Correlation of Popularity with Audio Features, Filtered by Genre - Country
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
loudness
energy
time_signature
duration_ms
tempo
valence
mode
danceability
key
instrumentalness
speechiness
liveness
acousticness


Correlation of Popularity with Audio Features, Filtered by Genre - Country Rock
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
energy
loudness
danceability
time_signature
mode
duration_ms
key
valence
tempo
speechiness
liveness
instrumentalness
acousticness

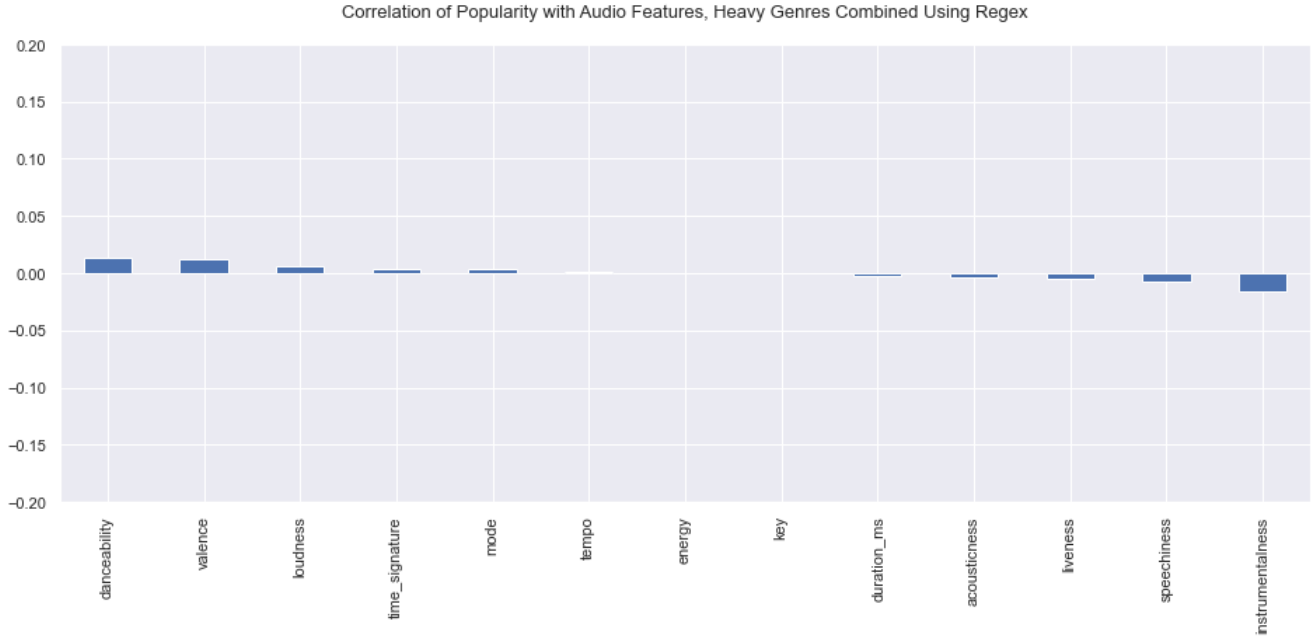

Correlation of Popularity with Audio Features, Filtered by Genre - Pop
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
loudness
duration_ms
energy
mode
time_signature
danceability
tempo
valence
key
instrumentalness
liveness
speechiness
acousticness


Correlation of Popularity with Audio Features, Filtered by Genre - Classic Rock
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
danceability
valence
energy
time_signature
loudness
tempo
mode
key
duration_ms
speechiness
liveness
instrumentalness
acousticness



In [21]:
```
# testing with regex to look for ability to categorise more broadly

heavy_regex = df_popularity[df_popularity.genre.str.contains(r'(?:metal|heavy|hard|alternative)').fillna(False)]
```

In [22]:
```
# shared y limits
ylim = -0.2, 0.2

# all genres
heavy_regex.corr()['POPULAR'].sort_values(ascending=False)[1:].plot(kind='bar', figsize=(16,6))
title='Correlation of Popularity with Audio Features, Heavy Genres Combined Using Regex'
plt.title(title, fontsize=13, pad=20)
plt.ylim(*ylim)
plt.show()
```



In [23]:
```
# there is some interesting stuff here!
heavy_regex.genre.sample(10)
```

Out[23]:
```
5142850                hardcore
2797925         gothenburg metal
1817958            spanish metal
1974334        metal pernambucano
2652048          mexican hardcore
3123181       neo classical metal
```

```
1466484    brazilian death metal
3080051    nederlandse hardstyle
5429164         industrial metal
284091              metal gaucho
Name: genre, dtype: object
```

Genres

Import Genre Data

In [15]:
```
df_genre_counts.shape[0]
```

Out[15]:
```
5489
```

In [78]:
```
# top 10 genres
df_genre_counts.head(10)
```

Out[78]:

genre_id	count
classical	495467
classical performance	287065
adult standards	191882
rock	136731
early music	127095
orchestral performance	119693
vocal jazz	117062
orchestra	113422
classic rock	109292
lounge	108578

In [27]:
```
# 10 random genres
list(df_genre_counts.sample(10).index)
```

Out[27]:
```
['rock alternativo brasileiro',
 'rap chretien',
 'marathi devotional',
 'power blues-rock',
 'swedish melodic rock',
 'deep smooth r&b',
 'flamenco urbano',
 'art rock',
 'psicodelia brasileira',
 'gothic americana']
```

Too Many Genres!!!

- genre_id seems more like a subgenre than an overarching genre
- it may be worth grouping these subgenres into broader genres

Regex Filtering to Create Subgenre Groups

Test Case 1: Heavy Music (not popular on Billboard Charts)

In [34]:
```
regex_heavy = r'(?:metal|heavy|hard|alternative)'

heavy_genre_counts = df_popularity[df_popularity.genre.str.contains(regex_heavy).fillna(False)].groupby('genre').count()['id'].sort_values(asc
```

In [41]:
```
# what about on the Billboard Hot 100?
df_B100_songs_AF[df_B100_songs_AF.genre.str.contains(regex_heavy).fillna(False)].groupby('genre').count()['id'].sort_values(ascending=False).h
```

Out[41]:
```
genre
alternative metal      129
hard rock               30
alternative rock         8
hardcore hip hop         8
alternative hip hop      4
Name: id, dtype: int64
```

In [186…
```
# alternative metal is most popular, let's take a look at the top 10 alt metal bands in the Billboard Hot 100
df_B100_songs_AF[df_B100_songs_AF.genre.str.contains('^alternative metal?')].groupby('artist').count()['id'].sort_values(ascending=False).head
```

Out[186…
```
artist
Linkin Park          19
Nickelback           18
Three Days Grace      7
Shinedown             7
Breaking Benjamin     6
Seether               6
Disturbed             6
Papa Roach            6
Good Charlotte        6
Limp Bizkit           5
Name: id, dtype: int64
```

In [56]:
```
# let's compare to the heavy regex category
df_B100_songs_AF[df_B100_songs_AF.genre.str.contains(regex_heavy)].groupby('artist').count()['id'].sort_values(ascending=False).head(10)

# same bands, slightly different sorting
```

Out[56]:
```
artist
Linkin Park          19
Nickelback           18
Three Days Grace      7
Shinedown             7
Seether               6
Breaking Benjamin     6
Papa Roach            6
Good Charlotte        6
Disturbed             6
Limp Bizkit           5
Name: id, dtype: int64
```

In [55]:
```
# even though the top 10 is the same, many less popular bands are missed
(df_B100_songs_AF[df_B100_songs_AF.genre.str.contains('alternative metal')].groupby('artist').count()['id'].shape[0],
df_B100_songs_AF[df_B100_songs_AF.genre.str.contains(regex_heavy)].groupby('artist').count()['id'].shape[0])
```

Out[55]:
```
(35, 86)
```

In [67]:
```
# let's take a look - every heavy song on the Billboard charts that isn't "alternative metal"
df_B100_songs_AF[
    (df_B100_songs_AF.genre.str.contains(regex_heavy)) & (df_B100_songs_AF.genre != 'alternative metal')
].sort_values('release_date')[['artist', 'song', 'release_date', 'genre']]

# not a lot of data to work with, probably shouldn't use this genre
# lets see if this improves specificity for audio features
```

Out[67]:

	artist	song	release_date	genre
17765	The Raiders	Just Seventeen	1970-03-14	classic hardstyle
6584	Gene Simmons	Radioactive	1978-01-01	hard rock
11954	Mistress	Mistrusted Love	1979-01-01	birmingham metal
6759	Giuffria	Call To The Heart	1984-01-01	hard rock
6761	Giuffria	Lonely In Love	1984-01-01	hard rock
6760	Giuffria	I Must Be Dreaming	1986-01-01	hard rock
15566	Stryper	Always There For You	1988-01-01	hard rock
2570	Britny Fox	Long Way To Love	1988-01-01	hard rock
19339	Vixen	Cryin'	1988-01-01	hard rock
19340	Vixen	Edge Of A Broken Heart	1988-01-01	hard rock
15568	Stryper	I Believe In You	1988-01-01	hard rock
16542	The Church	Under The Milky Way	1988-02-16	alternative rock
2774	BulletBoys	For The Love Of Money	1988-09-16	hard rock
2775	BulletBoys	Smooth Up	1988-09-16	hard rock
15827	Tangier	On The Line	1989-01-01	glam metal
14704	Saraya	Back To The Bullet	1989-01-01	hard rock
14705	Saraya	Love Has Taken Its Toll	1989-01-01	hard rock
18883	Tora Tora	Walkin' Shoes	1989-01-01	hard rock
5662	Enuff Z'Nuff	New Thing	1989-08-18	hard rock

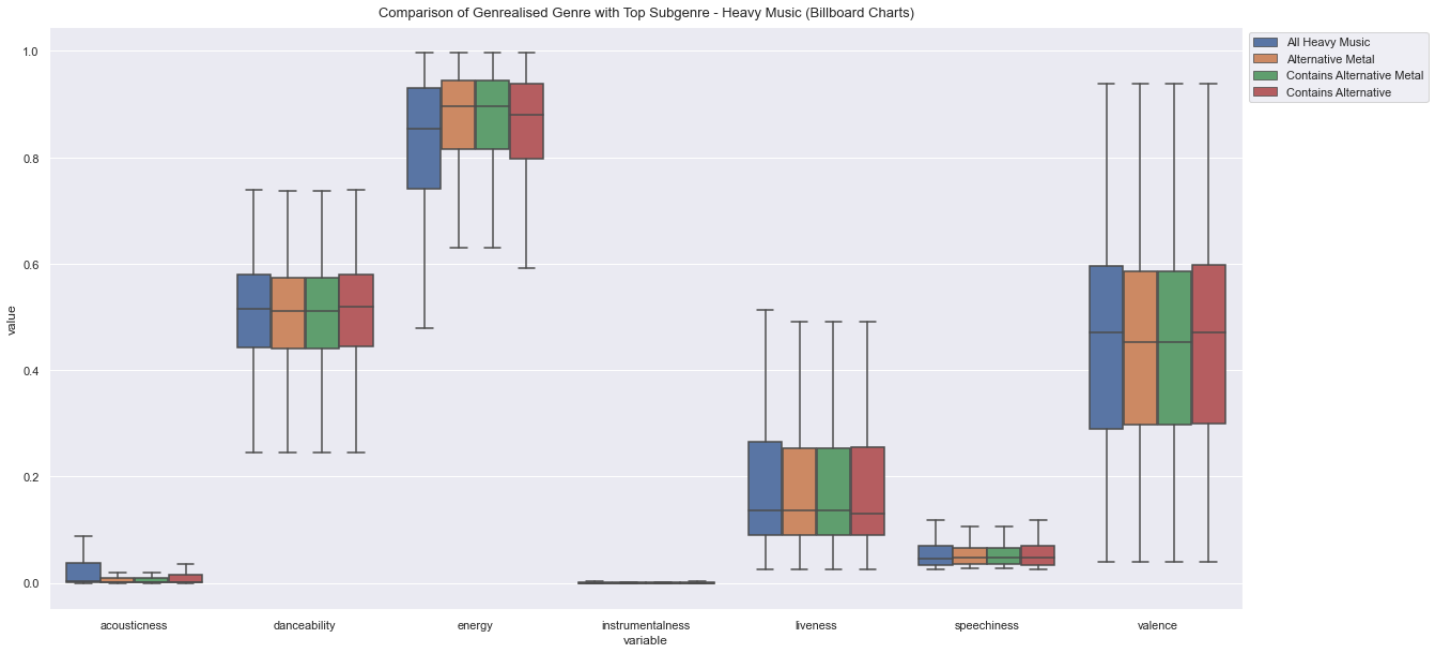
	artist	song	release_date	genre
11549	McAuley Schenker Group	Anytime	1989-10-10	hard rock
19342	Vixen	Love Is A Killer	1990-01-01	hard rock
10686	Little Caesar	Chain Of Fools	1990-01-01	hard rock
19077	Trixter	One In A Million	1990-01-01	hard rock
10687	Little Caesar	In Your Arms	1990-01-01	hard rock
19078	Trixter	Surrender	1990-01-01	hard rock
19076	Trixter	Give It To Me Good	1990-01-01	hard rock
19341	Vixen	How Much Love	1990-01-01	hard rock
12382	Nelson	More Than Ever	1990-06-26	hard rock
12383	Nelson	Only Time Will Tell	1990-06-26	hard rock
12381	Nelson	After The Rain	1990-06-26	hard rock
9519	Kane Roberts	Does Anybody Really Fall In Love Anymore?	1991-07-03	glam metal
11956	Mitch Malloy	Nobody Wins In This War	1992-01-01	melodic hard rock
11067	MC Serch	Here It Comes	1992-01-01	hardcore hip hop
11955	Mitch Malloy	Anything At All	1992-01-01	melodic hard rock
14526	Saigon Kick	Love Is On The Way	1992-01-01	hard rock
1437	Belly	Feed The Tree	1993-01-29	alternative rock
7146	Green Jelly	Three Little Pigs	1993-02-25	funk metal
11548	Mazzy Star	Fade Into You	1993-10-05	alternative rock
3972	Da Youngsta's	Hip Hop Ride	1994-01-01	hardcore hip hop
6378	Freedy Johnston	Bad Reputation	1994-06-07	alternative country
5365	Elastica	Stutter	1995-01-01	alternative rock
3112	Channel Live	Mad Izm	1995-01-01	hardcore hip hop
9348	Juliana Hatfield	Universal Heart-Beat	1995-01-01	alternative rock
5364	Elastica	Connection	1995-01-01	alternative rock
17800	The Rentals	Friends Of P.	1995-10-20	alternative rock
7342	Heather B.	Do You	1996-01-01	hardcore hip hop
14509	Sadat X	Hang 'Em High	1996-03-01	hardcore hip hop
7614	Ideal	Get Gone	1999-01-01	brazilian hardcore
5847	FRENTE!	Bizarre Love Triangle	1999-05-14	australian alternative rock
7084	Gorillaz	Clint Eastwood	2001-01-01	alternative hip hop
19642	X-Ecutioners	It's Goin' Down	2002-02-26	alternative hip hop
15567	Stryper	Honestly	2004-05-18	hard rock
13926	Rhinoceros	Apricot Brandy	2005-02-08	buffalo ny metal
7085	Gorillaz	Feel Good Inc	2005-05-23	alternative hip hop
7219	HIM	Wings Of A Butterfly	2006-02-24	gothic metal
4944	DragonForce	Through The Fire And Flames	2006-05-23	metal
5661	Enuff Z'Nuff	Fly High Michelle	2006-09-26	hard rock
13472	Prelude	For A Dancer	2006-10-30	epic black metal
17975	The Shins	Phantom Limb	2007-01-23	alternative rock
11121	Mad Lion	Take It Easy	2009-06-22	hardcore hip hop
14962	Shiny Toy Guns	Major Tom	2010-01-01	alternative dance
504	Alex Clare	Too Close	2011-01-01	modern alternative rock
14985	Shyheim	On And On	2011-05-17	hardcore hip hop
13471	Prelude	After The Goldrush	2012-03-05	epic black metal
12833	PMD	I Saw It Cummin'	2013-02-05	hardcore hip hop
5968	Felony	The Fanatic	2014-02-03	swiss metal
1207	Bad Wolves	Zombie	2018-01-19	metal

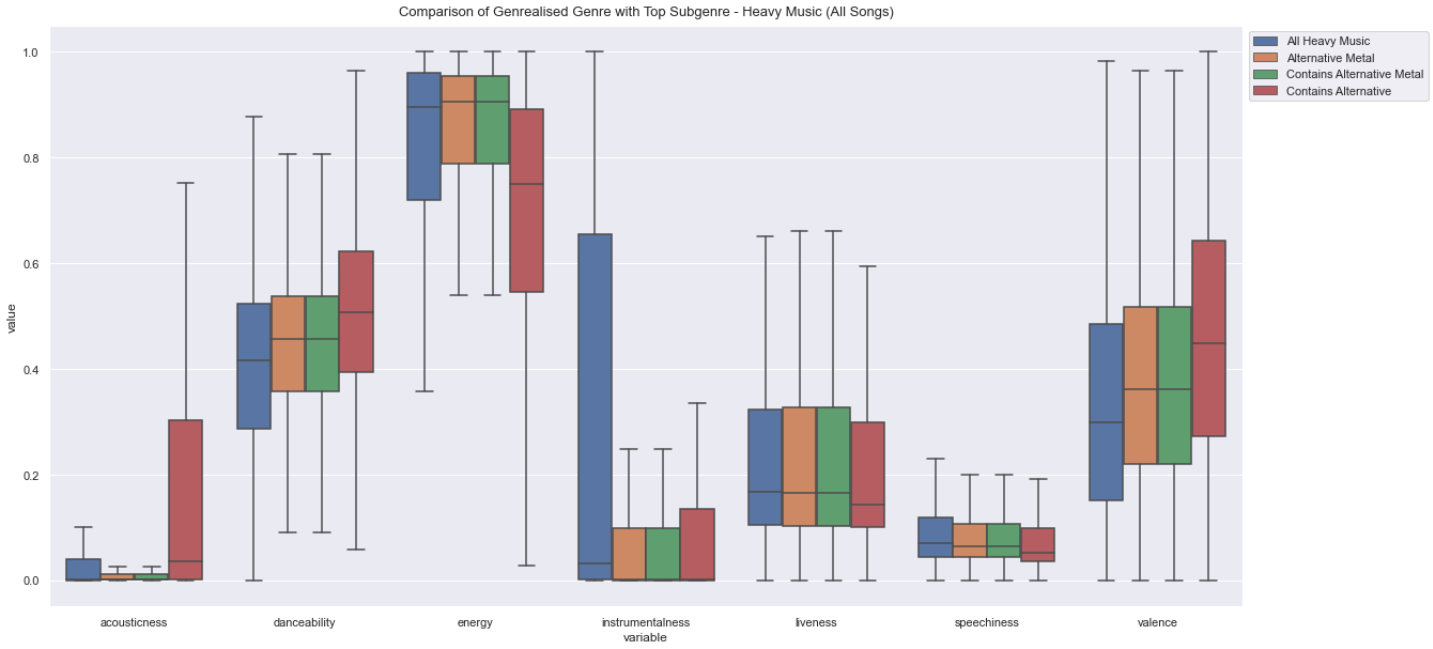
	artist	song	release_date	genre
9145	Joji	Slow Dancing In The Dark	2018-10-26	alternative r&b
5366	Electric Boys	All Lips N' Hips	2019-04-12	hard rock
9144	Joji	Sanctuary	2020-09-24	alternative r&b
9143	Joji	Run	2020-09-25	alternative r&b
12642	Oliver Tree	Life Goes On	2021-05-28	alternative hip hop

Boxplots

In [16]:
```
# helper function with variable number of dataframes would be useful to compare more data

def boxplot_genre_comparison(dataframe, genre_dict, title, figsize=(20,10), showfliers=False):

    boxplot_features = ['acousticness', 'danceability', 'energy', 'instrumentalness', 'liveness', 'speechiness', 'valence']
    plt.figure(figsize=figsize)

    list_of_genre_df = []

    for genre in genre_dict:
        list_of_genre_df.append(dataframe[dataframe.genre.str.contains(genre_dict[genre]).fillna(False)][boxplot_features].assign(genre_name=g

    sns.boxplot(pd.melt(pd.concat(list_of_genre_df), id_vars=['genre_name']), x='variable', y='value', hue='genre_name', showfliers=showfliers
#     plt.title(title, fontsize=13, pad=10)
    plt.legend(loc='upper left', bbox_to_anchor=(1, 1))

    # save the figure
    plt.savefig(f'figures/genres/{title}.png', facecolor='w', dpi=150, bbox_inches='tight')

    plt.show()
```

In [58]:
```
genre_dict = {
    'All Heavy Music': r'(?:metal|heavy|hard|alternative)',
    'Alternative Metal': '^alternative metal?',
    'Contains Alternative Metal': 'alternative metal',
    'Contains Alternative': 'alternative',
}

title = 'Comparison of Genrealised Genre with Top Subgenre - Heavy Music ({})'

boxplot_genre_comparison(df_B100_songs_AF, genre_dict, title.format('Billboard Charts'))
boxplot_genre_comparison(df_10M, genre_dict, title.format('All Songs'))
```

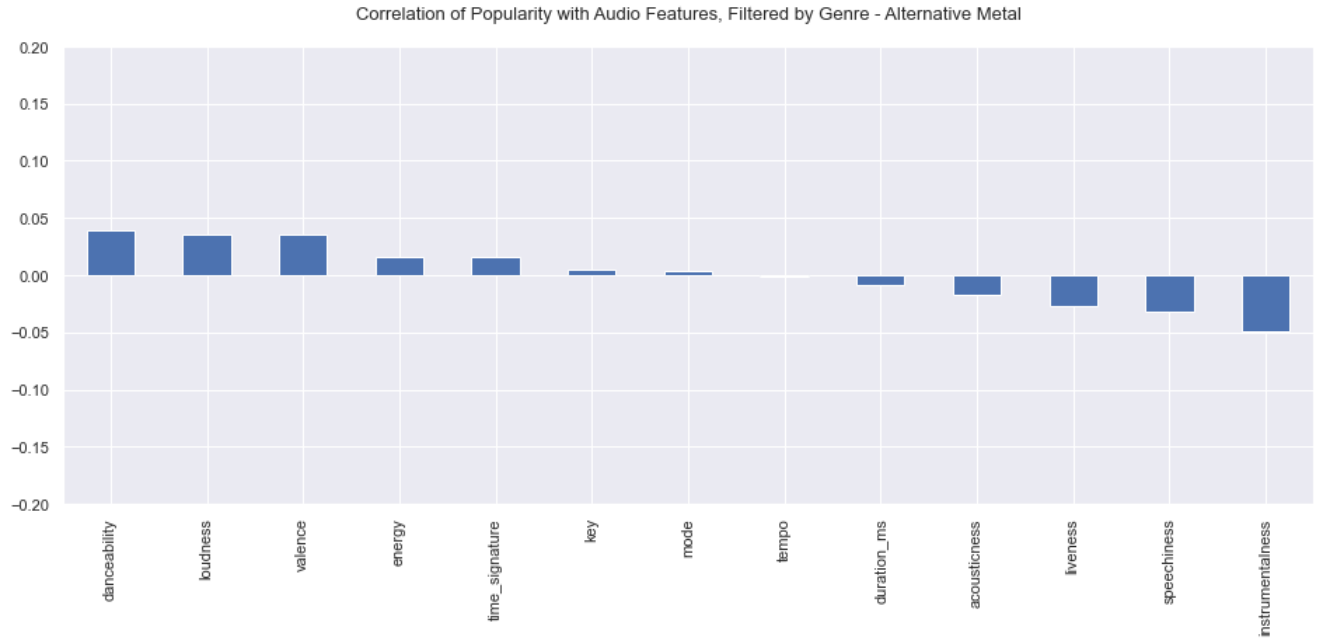




Conclusion: It looks like adding more genrealised genre data increases variability

In [162…

```
# let's see if "alternative metal is any better behaved than heavy music in general"

genre = 'alternative metal'

df_popularity.query(f'genre == "{genre}"').corr()['POPULAR'].sort_values(ascending=False)[1:].plot(kind='bar', figsize=(16,6))
title = f'Correlation of Popularity with Audio Features, Filtered by Genre - {genre.title()}'
plt.title(title, fontsize=13, pad=20)
plt.ylim(*ylim)

plt.show()
```



- This looks a little better, not as strong as some genres, and weak correlation overall
- It may be difficult to make any popularity predictions using these features for this genre
- Interesting non-intuitive results. Popular Alternative Metal is:
  - Billboard charts have more consistent songs (not surprising)
  - danceable
  - loud (not surprising)
  - happier than typically in the genre (positively correlated with higher valence)
  - lower instrumentalness correlated with higher "popularity"

Test Case 2: Pop Music

In [135…
```
regex_pop = r'(?:pop|adult standards|contemporary)'

# top 10 regex_pop genres (all songs)
df_popularity[df_popularity.genre.str.contains(regex_pop).fillna(False)].groupby('genre').count()['id'].sort_values(ascending=False).head(10)
```

Out[135…
```
genre
adult standards      185851
dance pop             36147
c-pop                 19507
pop rock              19077
pop rap               17152
afropop               16285
italian adult pop     14738
europop               14027
new wave pop          13324
russian pop           12319
Name: id, dtype: int64
```

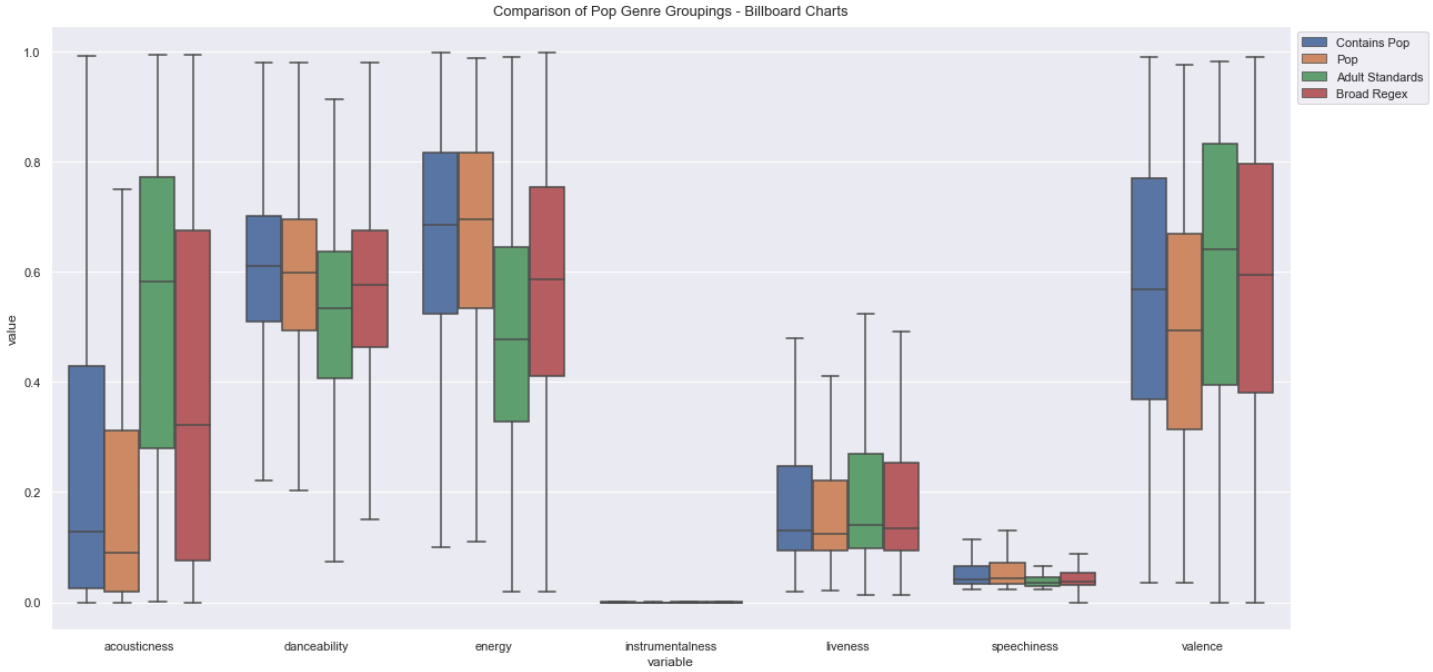
In [138…
```
# top 10 regex_pop genres (billboard)
df_B100_songs_AF[df_B100_songs_AF.genre.str.contains(regex_pop).fillna(False)].groupby('genre').count()['id'].sort_values(ascending=False).hea
```

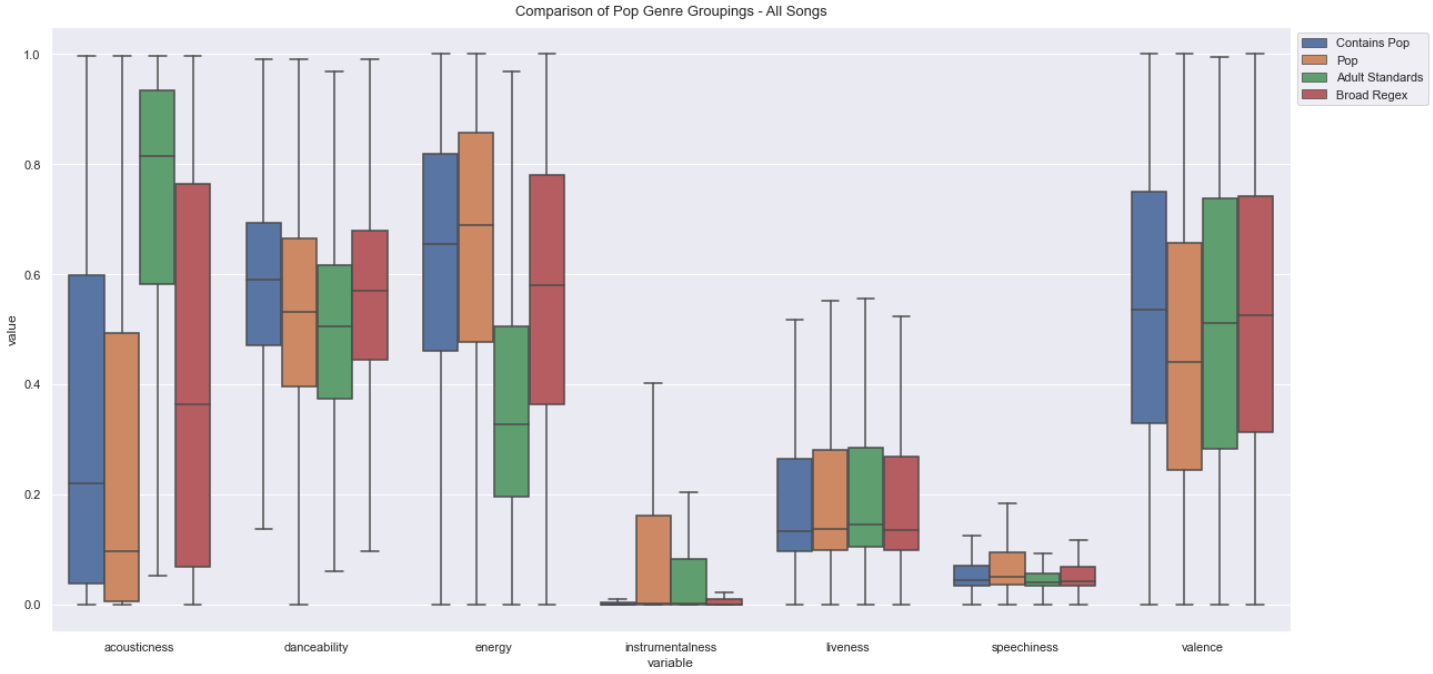
Out[138…
```
genre
adult standards         3339
dance pop               1172
pop                      481
pop rock                 415
pop rap                  389
brill building pop       384
new wave pop             227
post-teen pop            187
bubblegum pop            126
deep adult standards     102
Name: id, dtype: int64
```

In [134…
```
# confirming that "adult standards" are indeed Pop Music - looks like it to me!
df_B100_songs_AF.query('genre == "adult standards"').artist.sample(10)
```

Out[134…
```
2357           Boz Scaggs
14564           Sam Cooke
17070         The Hollies
1074          B.J. Thomas
4286          Dean Martin
1294     Barbra Streisand
14760      Seals & Crofts
15520       Stevie Wonder
9049        Johnny Mathis
7875        Jackie Wilson
Name: artist, dtype: object
```

In [59]:
```
# boxplot comparison of Pop genres

genre_dict = {
    'Contains Pop': r'pop',
    'Pop': r'^pop?',
    'Adult Standards': r'^adult standards?',
    'Broad Regex':  r'(?:pop|adult standards|contemporary)'
}

title = 'Comparison of Pop Genre Groupings - {}'

boxplot_genre_comparison(df_B100_songs_AF, genre_dict, title.format('Billboard Charts'))
boxplot_genre_comparison(df_10M, genre_dict, title.format('All Songs'))
```





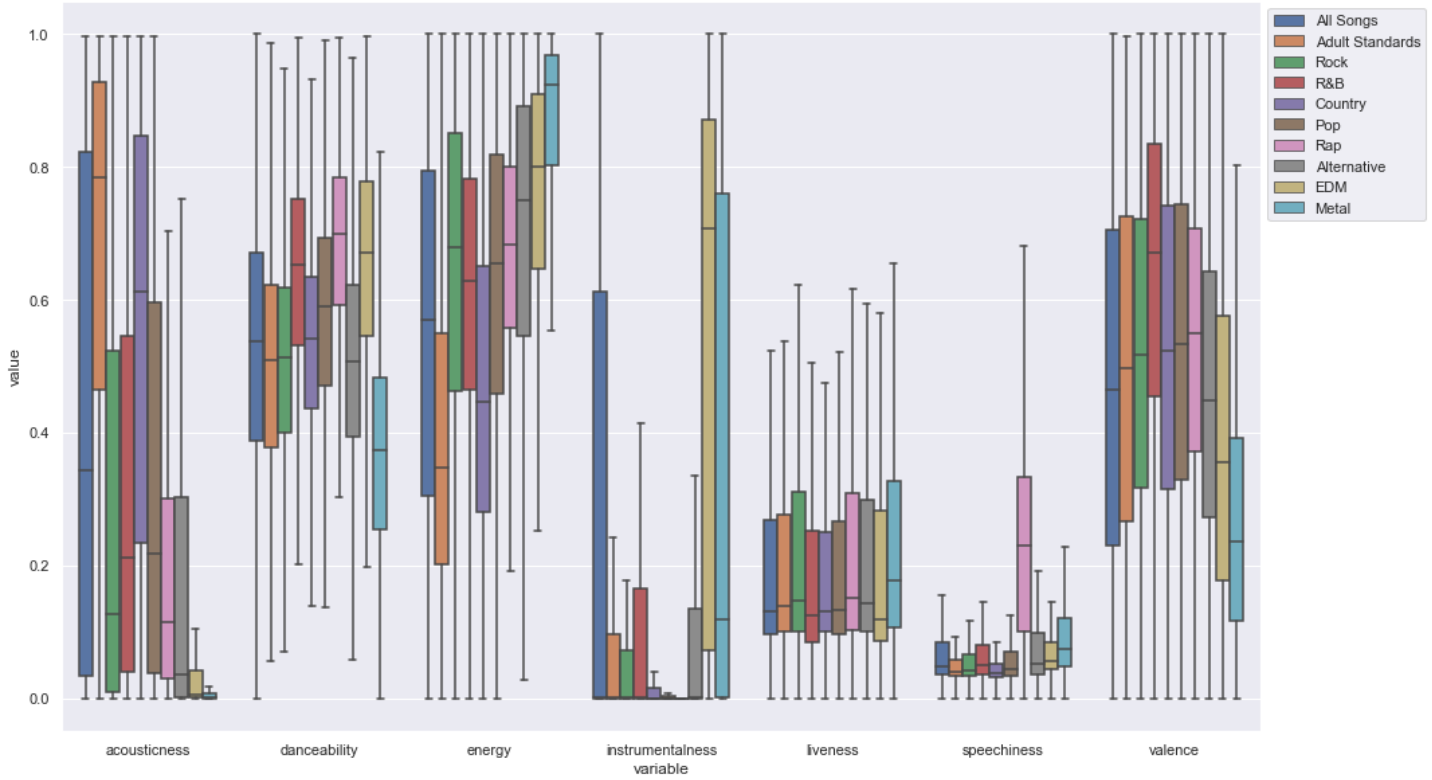
Boxplots for Large Genre Groups

In [52]:
```
# df_B100_songs_AF.groupby('genre').count()['id'].sort_values(ascending=False).head(50)
```

In [18]:
```
# approximating genre categories
genre_dict = {
    'All Songs': r'.+',
    'Adult Standards': r'(?:adult standards|mellow gold|soft rock|contemporary)',
    'Rock': r'(?:rock|^country|^soft)',
    'R&B':  r'(?:soul|rhythm and blues|motown|funk|disco|quiet storm)',  # this one is a stretch
    'Country': r'(?:country|folk|nashville sound)',
    'Pop': r'\b(?:pop|europop)\b',  # whole word only
    'Rap': r'\b(?:rap|trap|hip hop)\b',  # whole word only
    'Alternative': r'(?:alternative)',  # not sure if grunge or punk should be here
    'EDM': r'(?:house|trance|techno|drum and bass|dubstep|synth|edm|breakbeat)', # not very popular on Bilboard
    'Metal': r'(?:heavy|metal|thrash|djent|deathcore|mathcore|grindcore)' # not very popular on Bilboard
}
```

In [20]:
```
# plot it
title = 'Comparison of Audio Featuers for Large Genre Groupings (as defined by Kevin) - {}'
figsize = (16, 10)
```

```
# boxplot_genre_comparison(df_B100_songs_AF, genre_dict, title.format('Billboard Charts'), figsize=figsize)
boxplot_genre_comparison(df_10M, genre_dict, title.format('All Songs'), figsize=figsize)
```



In [125…
```
# genre statistics vs relative popularity

df_genres = pd.DataFrame()

for genre in list(genre_dict.keys()):
    df_genres.loc[genre, 'billboard_counts'] = df_B100_songs_AF[(df_B100_songs_AF.genre.str.contains(genre_dict[genre])).fillna(False)].count(
    df_genres.loc[genre, 'all_counts'] = df_10M[(df_10M.genre.str.contains(genre_dict[genre])).fillna(False)].count()['id']
```

In [147…
```
# new columns
df_genres['proportion_billboard'] = df_genres['billboard_counts'] / df_genres['billboard_counts'].sum()
df_genres['proportion_allsongs'] = df_genres['all_counts'] / df_genres['all_counts'].sum()
df_genres['relative_popularity'] = df_genres['proportion_billboard'] / df_genres['proportion_allsongs']
df_genres['songs_per_popular_song'] = df_genres['all_counts'] / df_genres['billboard_counts']

# transpose and format
dft = df_genres.T
dft.iloc[0:2] = dft.iloc[0:2].applymap('{0:,.0f}'.format)
dft.iloc[2:-1] = dft.iloc[2:-1].applymap('{0:,.3f}'.format)
dft.iloc[-1:] = dft.iloc[-1:].applymap('{0:,.0f}'.format)
dft
```

Out[147…

	Adult Standards	Rock	R&B	Country	Pop	Rap	Alternative	EDM	Metal
billboard_counts	4,313	6,457	2,594	2,268	3,784	2,269	148	101	140
all_counts	234,547	727,276	157,511	283,317	600,693	454,944	89,355	271,306	268,320
proportion_billboard	0.195	0.293	0.118	0.103	0.171	0.103	0.007	0.005	0.006
proportion_allsongs	0.076	0.236	0.051	0.092	0.195	0.147	0.029	0.088	0.087
relative_popularity	2.572	1.242	2.303	1.120	0.881	0.698	0.232	0.052	0.073
songs_per_popular_song	54	113	61	125	159	201	604	2,686	1,917

In [154…
```
# best genres to write music in if you want onto the Billboad Hot 100
print('Total Songs in each Genre per Song on the Billboard Hot 100')
print(df_genres['songs_per_popular_song'].astype('int32').sort_values())
```

```
Total Songs in each Genre per Song on the Billboard Hot 100
Adult Standards      54
R&B                  60
Rock                112
Country             124
```

```
Pop                 158
Rap                 200
Alternative         603
Metal              1916
EDM                2686
Name: songs_per_popular_song, dtype: int32
```

In [ ]: