Beckett Dumo

Dr. Jason Zietz

INFO 3510

12/18/2023

The Holiday Blues: An exploration of popular Christmas songs

Background

As the days grow shorter and the weather colder, excitement for the Holiday season grows. The holidays are a time of joy and family. People worldwide celebrate the holiday season with their traditions and observances. Whether you practice Christmas, Hanukkah, Kwanzaa, or any other celebration, the season can bring beautiful childhood and family memories. It can also bring depression. Feelings of sadness and isolation are not uncommon. These symptoms are nicknamed the Holiday Blues or Holiday depression. The problem comes from expectations. As you grow older and your free time shrinks, many working people feel the mystery and whimsicality of the Holidays fade.

Moreover, those with one to celebrate with can feel lonely because the holidays are synonymous with togetherness. Christmas music, which is many people's favorite (or least favorite) part of the holidays, has accounted for the mixed emotions in the holiday season. After talking with friends and family, everyone has unique Christmas music tastes. My Mother loves *River* by Joni Mitchell. I like Wham!'s *Last Christmas*, and when I looked at my friends' tastes, I found that Christmas songs can be broken down into happy and melancholic. While they are both equally respectable styles of Christmas music, I was interested in whether people want a Happy or Sad Christmas. Moreover, I was interested in which era of artists were most popular because I noticed that very few 2000s and above covers of Christmas music songs became popular. Lastly, in each decade, was happy or melancholic music more popular?

Data and Content Types

The Change

Initially, my project was proposed to be very different. I had planned to import and compile sad and happy Christmas song lists and use Genius and VADER to analyze the lyrics of these songs. With the data, I wanted to examine whether sad songs had sad sentiments. Ultimately, this idea was shallow and wouldn't give me the required two unique content types. Moreover, I felt it wasn't doing anything interesting, so I removed it and repositioned my project's central idea. Furthermore, the lists of happy and sad songs rated by various websites had so much variety and biases that I felt they wouldn't work for my project. Moreover, I needed not to choose the songs because my own bias would come into my analysis. For this reason, I chose the 2022 Billboard staff picks of their 100 favorite Christmas songs.

The Christmas Classics problem

As mentioned, Christmas music is tightly connected with childhood memory. Memories of what your parents or guardians played heavily influence your present-day tastes. Many popular Christmas songs are the 40s-60s versions because of how fondly they were listened to and passed down. The new releases are good, but they often fail to out shadow the past: a sparse few break this barrier, Michael Buble, for example.

Billboard Top 100 Christmas Song List

The 2022 Billboard staff picks 100 favorite Christmas songs and is attractive as a dataset for multiple reasons. Mainly, it had diversity in release year and artists, which was necessary for a comprehensive investigation. Billboard has a chart of the greatest holiday songs of all time, but the problem is they are all the classics. On the other hand, it is a diverse list of personal favorites, which was appropriate for the investigation.

Content

The content I explored was release years, listener data, and audio analysis. The release years were collected from the list itself. Listener data was pulled using the Last FM API, and the

audio analysis data, not to be confused with audio features, was pulled using the Spotify API. I developed the spreadsheet of comma-separated values from the staff list. In the previous project, I found a Billboard API that could pull any chart from the Billboard on any date. I couldn't use that for this project because the staff list is stored as a list and blog, not as a chart. I set the headers to create the CSV as Ranking, Artist, Song, and Year. Within these columns, I filled in the data from the Billboard list in order of the ranking given to them by the staff. During this process, I listened to each song on the list, and a problem was revealed to me before analysis. Ranking 34 on the list was Saturday Night Live Cast's "I Wish It Was Christmas Today." The song was originally performed in a skit by Horatio Sanz, Jimmy Fallon, Chris Kattan, and Tracy Morgan. The problem was that the song wasn't on Spotify, and Last FM had limited information. To remedy this issue, I replaced the song with a cover of "I Wish It Was Christmas Today" by Julian Casablancas of the Strokes. While it isn't the same, it seemed equally popular after research.

Spotify Audio Analysis

I began the investigation by importing the necessary libraries and modules. I also used python-dotenv to read key-value pairs in an env file containing my API secrets, keys, and IDs. I followed by importing the CSV itself and performed an Exploratory Data Analysis of the spreadsheet. I checked that the keys/columns and the names had been imported correctly. I used metadata I noticed in the audio analysis search in Spotify's API to decide whether a song can be considered happy or melancholy. The metadata reports data about the track itself, like the pitch and timbre of sections of the song. For my investigation, I used Key and Tempo as indicators of Positive and Negative. In music theory, the key is a scale of pitches around which the song is centered. Within the notation of the API, this value is reported between 0 and 11, with -1 being no key found. The 0 to 11 scale refers to Pitch Class Notation that assigns a specific integer to a pitch class. For example, C is 0, C sharp/D flat (enharmonically equivalent notes) is 1, D is 2, and so on. Another piece of metadata recorded is the mode. Modality is recorded as 1 being Major key and 2 being Minor key. When combined, we can see the scale on which the song is centered, C sharp Major, for example. We can classify a Major key song as happy and a Minor key song as sad.

To use this metadata, I needed to look through the Spotify API. The Spotify API requires a lot of authorization and parameters. Using requests, I created the API URL with all my authentication information filled in. With the URL in hand, I created a for-loop function that iterates through the index and rows of my CSV-turned data frame. It assigns the data at each index within the columns Song and Artist to a variable name equivalent. Using Requests, I query the API with the URL and the variables within. With the metadata recorded, I iterate through the list indices for the Spotify-generated ID and Spotify-generated popularity score. These values are appended by the empty lists I created outside the loop to ensure I'm not modifying something I'm iterating over. I store all the song data from my original df in a list at the same index as its Spotify metadata. This is done so I can later organize and ensure all the data corresponds with its song. In the case of an index error, which happened frequently on the previous project, I used an except clause to catch and report the song name that failed. In the cells following, I search for the songs individually outside the for-loop. I tried taking and rerunning the names within the loop, but this caused problems. I found searching songs by name only fixed this problem. With all 100 songs added and checked by looking at the length of the list, I moved on to getting the audio analysis.

Using the Spotify IDs stored, I searched for each song's audio analysis metadata. Since their IDs are unique, they are far more accurate when searching the API. Example of search:

```
trackanalysis = requests.get(rootURL+"audio-analysis/" + song,headers=headers).json()
```

The analysis metadata is stored in an empty list, which is turned into a data frame. The organizational data is stored alongside it for when I merge data frames later. This was done because the songs' order changed when four failed and were added at the end. The organizational data proved invaluable for ensuring the order stays true for visualizations.

Popularity through Last FM and Spotify

When taking the IDs, I also took the Spotify popularity score of each song. This score is created by aggregating the total streams of a song, how recently a song has been played, and the frequency of a track being played. The score is from 0 to 100, with 100 being the most popular and 0 being the least.

I used the Last FM API data on song listeners and total plays to complement the Spotify score. I followed a setup similar to the Spotify API for-loop to acquire the metadata from the API. I created several empty lists to store the meta and organizational data. I iterate and assign the song name and artist to a variable within the for-loop. Also, within the for-loop, I placed the base URL for the API call within the loop, unlike how I did with Spotify. Strangely, this was the only way to get the data consistently. The data itself is stored in an if-else check where, within the loop, the data is checked to ensure it isn't empty. If the data is present, the play count and listener data are stored as an integer, not an object under a new variable. These variables are appended to the empty lists along with the organizational data. One song repeatedly failed, 7 O'Clock News/Silent Night by Simon & Garfunkel. When I searched for the song outside the loop, there was no data on the track; I believe this is because of the varied symbols in the song title. So, I took the data from the website and entered it manually as the else portion of the check. I employed the same try-except clause with Spotify to ensure the program didn't stop. With the listener and organizational data, I created a new data frame to store it.

```
alldfs = pd.merge(
    left = analysis,
    right = lastFMDF,
    left_on = ['name','ranking'],
    right_on = ['fmnames','rankingfm'],
    how = 'inner'
)
```

I merged the analysis data frame and the Last FM data frame on an inner merge using the name and ranking columns I had stored as organizational data. Here is the merge function

Compiling Data

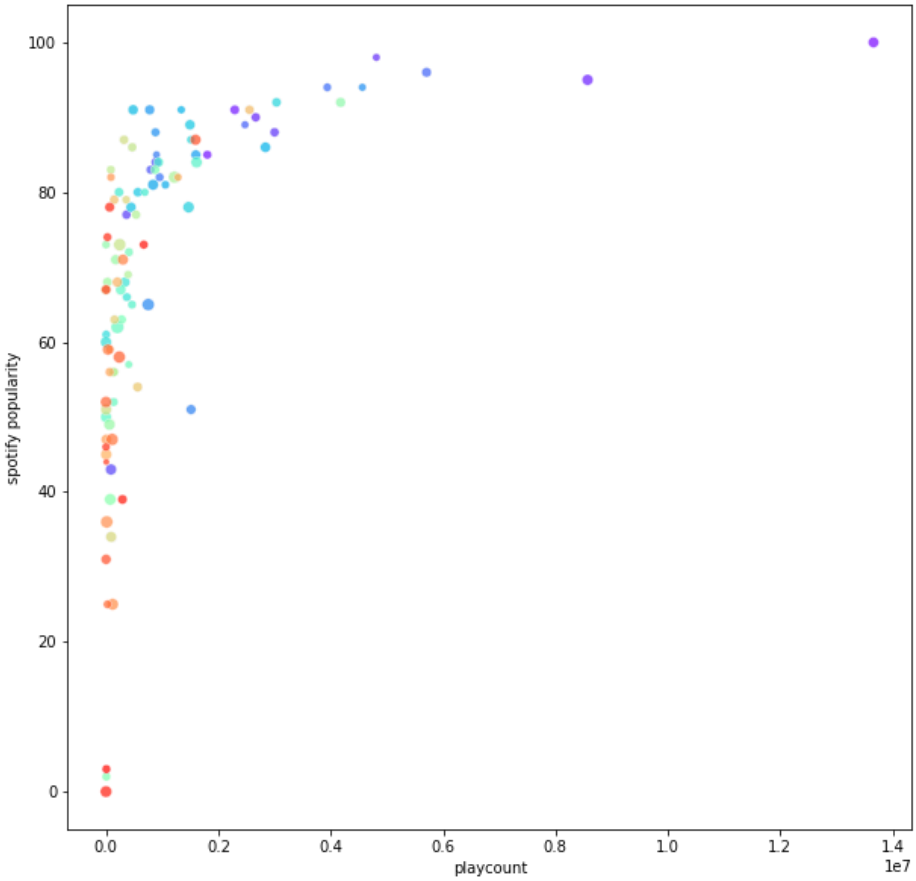
Since the Spotify data was stored as integers and for my analysis, I wanted the names I created for-loops with if statements to check for specific numbers and assign a new name. These functions are long, but I have attached photos below.

```
scale = []
for index,row in analysis.iterrows():
    key = row["key"]
    mode = row["mode"]
    if key != -1:
        if key == 0 and mode == 1:
            scale.append("C Major")
        if key == 0 and mode == 0:
            scale.append("C Minor")
        if key == 1 and mode == 1:
            scale.append("C# or D flat Major")
        if key == 1 and mode == 0:
            scale.append("C# or D flat Minor")
        if key == 2 and mode == 1:
            scale.append("D Major")
        if key == 2 and mode == 0:
            scale.append("D Minor")
        if key == 3 and mode == 1:
            scale.append("D# or E flat Major")
        if key == 3 and mode == 0:
            scale.append("D# or E flat Minor")
        if key == 4 and mode == 1:
            scale.append("E Major")
        if key == 4 and mode == 0:
            scale.append("E Minor")
        if key == 5 and mode == 1:
            scale.append("F Major")
        if key == 5 and mode == 0:
            scale.append("F Minor")
        if key == 6 and mode == 1:
            scale.append("F# or G flat Major")
        if key == 6 and mode == 0:
            scale.append("F# or G flat Minor")
        if key == 7 and mode == 1:
            scale.append("G Major")
        if key == 7 and mode == 0:
            scale.append("G Minor")
        if key == 8 and mode == 1:
            scale.append("G# or A flat Major")
        if key == 8 and mode == 0:
            scale.append("G# or A flat Minor")
        if key == 9 and mode == 1:
            scale.append("A Major")
        if key == 9 and mode == 0:
            scale.append("A Minor")
        if key == 10 and mode == 1:
            scale.append("A# or B flat Major")
        if key == 10 and mode == 0:
            scale.append("A# or B flat Minor")
        if key == 11 and mode == 1:
            scale.append("B Major")
        if key == 11 and mode == 0:
            scale.append("B Minor")
    else:
        pass
```

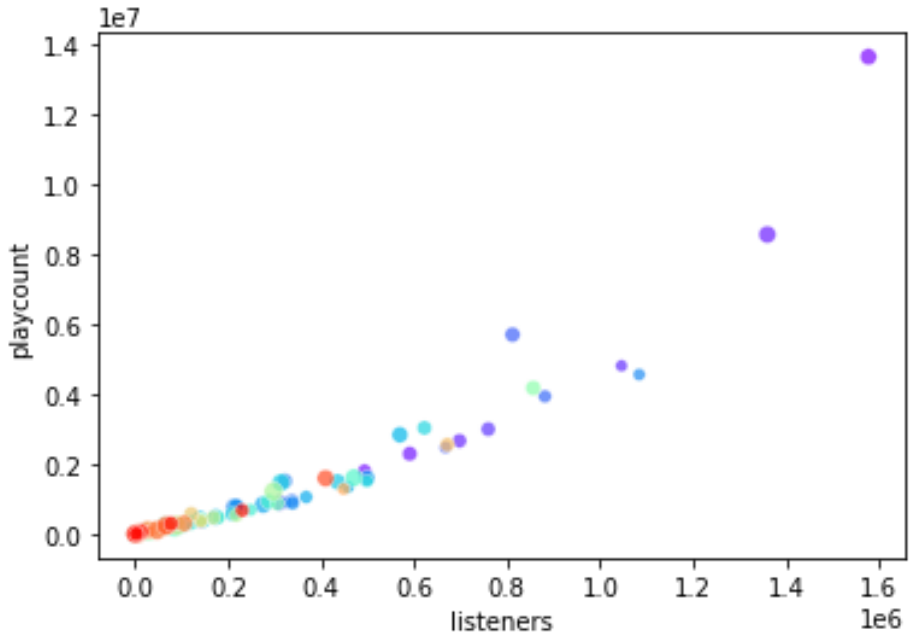
```
decade = []
for index,row in alldfs.iterrows():
    year = row["Year"]
    if year >= 1900 and year <= 1909:
        decade.append("1900s")
    if year >= 1910 and year <= 1919:
        decade.append("1910s")
    if year >= 1920 and year <= 1929:
        decade.append("1920s")
    if year >= 1930 and year <= 1939:
        decade.append("1930s")
    if year >= 1940 and year <= 1949:
        decade.append("1940s")
    if year >= 1950 and year <= 1959:
        decade.append("1950s")
    if year >= 1960 and year <= 1969:
        decade.append("1960s")
    if year >= 1970 and year <= 1979:
        decade.append("1970s")
    if year >= 1980 and year <= 1989:
        decade.append("1980s")
    if year >= 1990 and year <= 1999:
        decade.append("1990s")
    if year >= 2000 and year <= 2009:
        decade.append("2000s")
    if year >= 2010 and year <= 2019:
        decade.append("2010s")
    if year >= 2020 and year <= 2029:
        decade.append("2020s")
```

It was a very disgusting code, but it worked well for investigation.

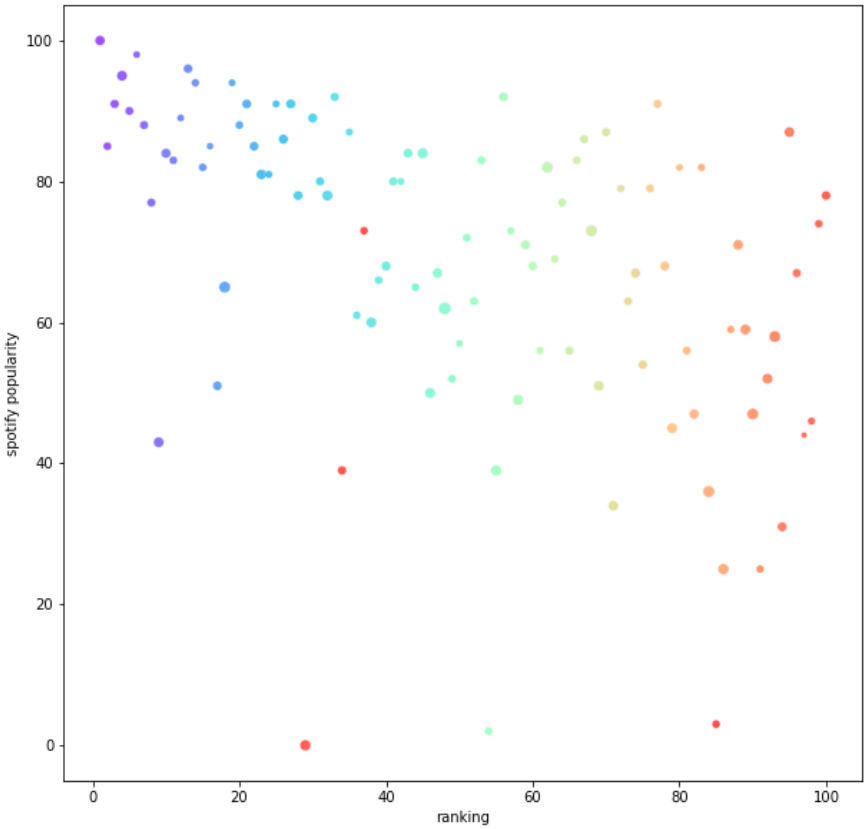
Visualizations



Here is a scatterplot with the Spotify popularity score on the y-axis and the total play count from Last FM. This visualization simply checked how heavily play count influenced the Spotify score.

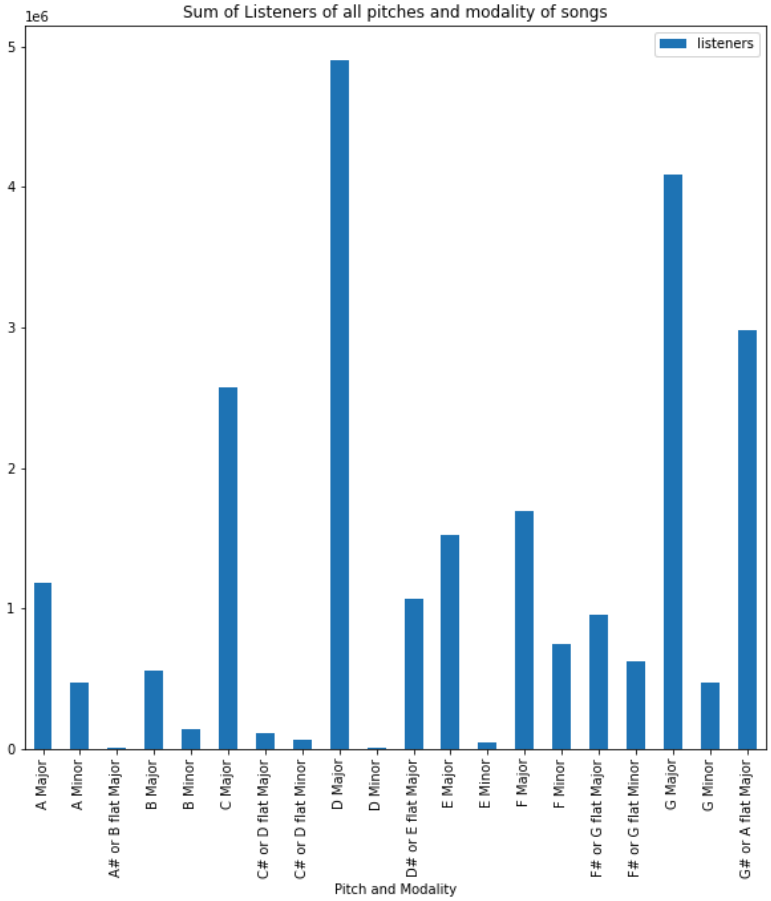


Here is a similar check using only the Last FM data. The total play count is on the y-axis, and the active listeners are recorded on the x-axis. They are obviously correlated but nonetheless interesting. Mariah Carey's "All I Want for Christmas is You" takes the top spot, as expected.

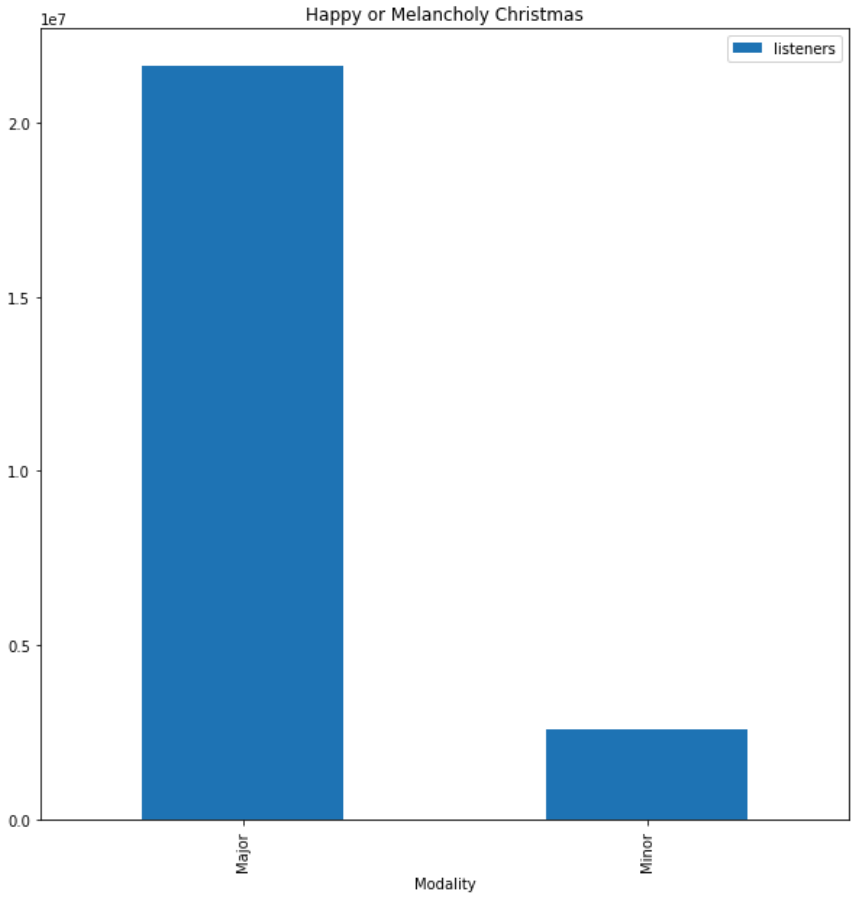


Here, I am looking to see if the ranking given by the Billboard crew was biased by the song's popularity as a whole. The ranking goes from 1-100, and Spotify 0-100. So, in this case, the rough semi-circle suggests that the less popular songs are, on average, lower in the top 100 ranking.

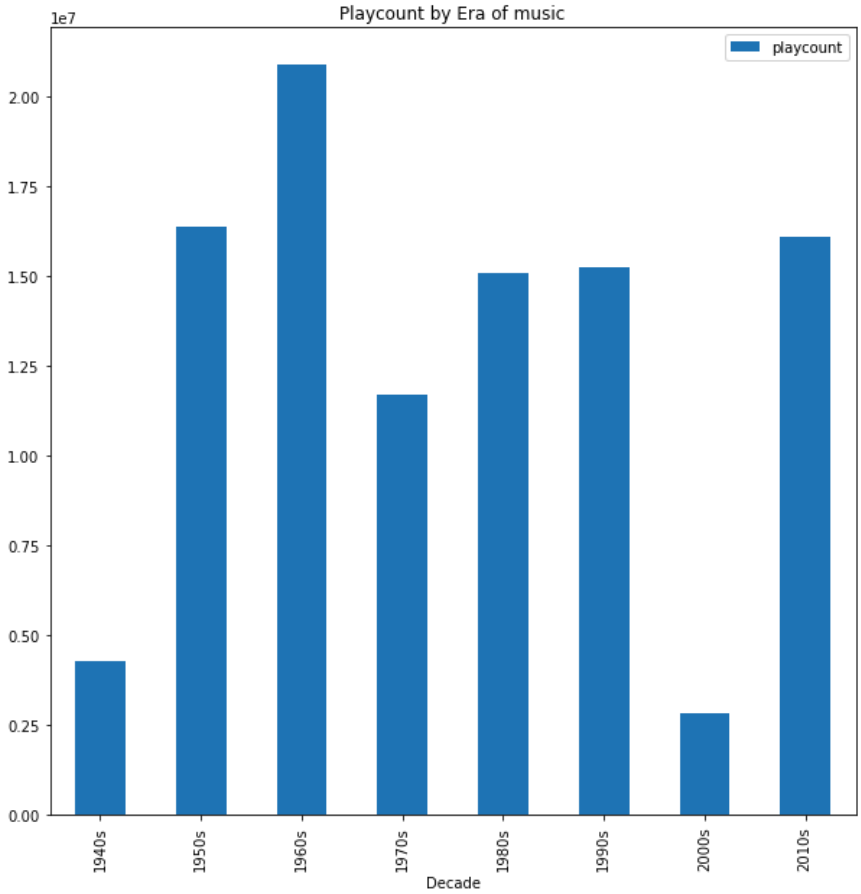
Investigation-appropriate visualizations



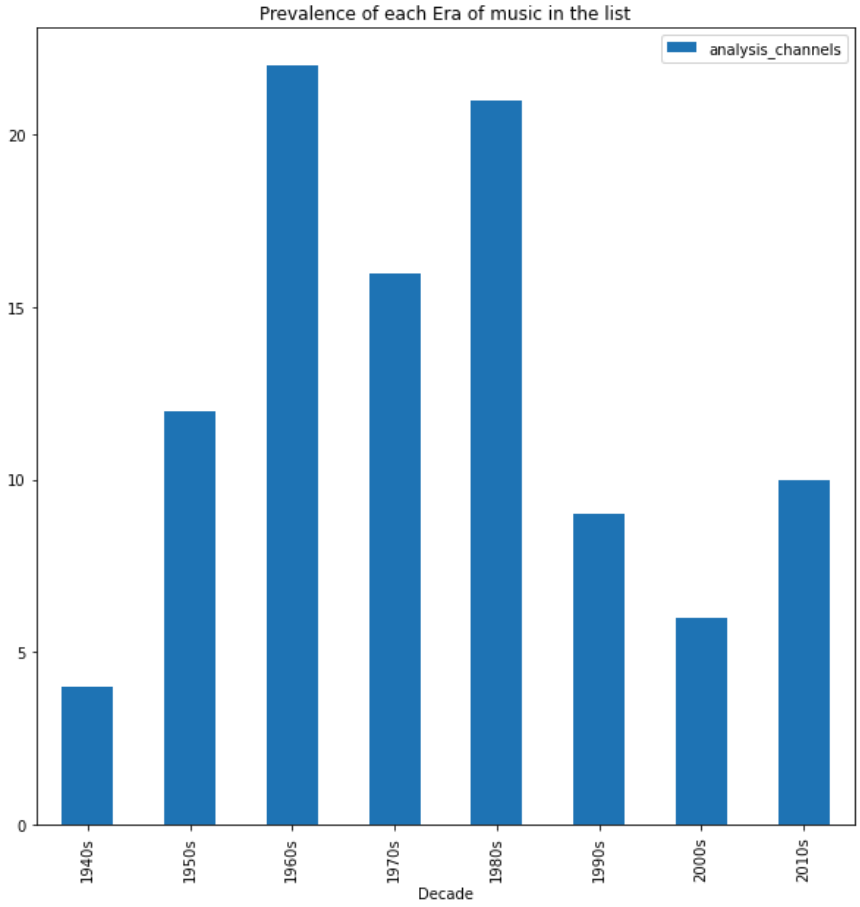
Here is a visualization of the sum of listeners of songs at each key and mode. D-Major is the best key for Christmas songs, with about 5 million listeners. G major comes at a close second. The most popular key for sad songs is F minor.



This visualization explores the number of listeners for Christmas songs with either a major or minor key. As you can see, the majority of listeners enjoy happier keyed songs.



This visualization aggregates the play count of the songs and sorts it by decade of release. Staying true to the Christmas classics, 50s and 60s are the most popular. 80s, 90s, and 2010s closely follow.



This visualization compares the prevalence of each decade on the Billboard itself. Compared with the chart above, the 50s are disproportionately popular compared to the number of songs on the list. Moreover, the 80s are overrepresented but less popular.

Analysis

The main takeaway from this investigation is that, while melancholic Christmas songs are present, their jolly counterparts are more popular. The '60s are the golden age of Christmas songs, with their popularity in the rankings, prevalence in the list, and total listeners keeping them a step above their counterparts.

Work for the Future

If I were to continue working on this project, I would expand to thousands of songs and look into songs in general, not under a happy or sad focus. Looking at the prevalence of words in Christmas songs could be interesting. It would probably be Christmas and Merry at the top of the list.

Resources

Billboard Staff. “The 100 Best Christmas Songs of All Time: Staff List.” *Billboard*, Billboard, 30 Nov. 2022, www.billboard.com/lists/best-christmas-songs/.

Python

Spotify API: https://developer.spotify.com/

Last FM: https://www.last.fm/api

Matplotlib: https://matplotlib.org/

Seaborn: https://seaborn.pydata.org/

Numpy: https://numpy.org/

Regex: https://regexr.com/

Word Count for Peer Reviewer: 2242 Words