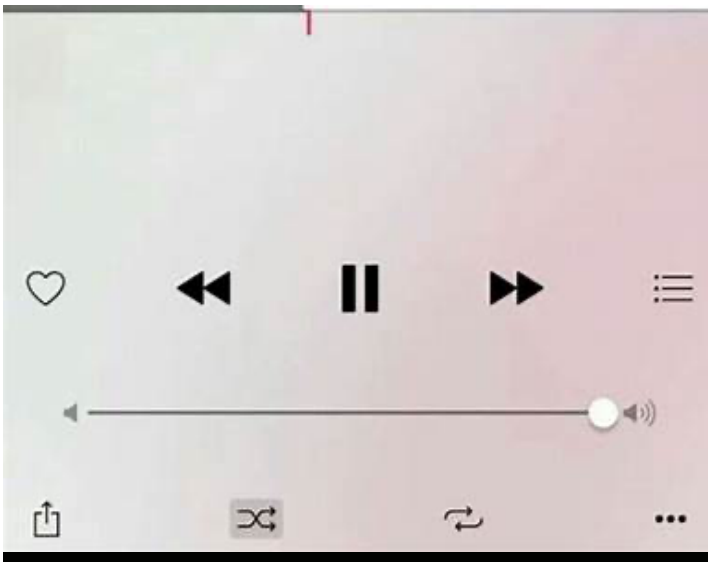


SPOTIFY

Playlist Data Processing



IST652-Scripting for Data Analysis. MiniProject 2

Anjana Sowmya Puvvada

Suid-6788587001

Table of Contents.

Executive Summary	3
Introduction:	3
Data Source: Spotify API	3
Description of the Process:	4
DATA EXTRACTION	4
DATA EXPLORATION	5
Time series Analysis	5
Data Summary Analysis	6
Analyzing and Comparing Spotify Playlists:	7
Comparing the Characteristics of Spotify playlists	9
Track duration	9
Popularity Comparison	10
Distribution of Release Years Comparison	11
Key Takeaways	12
Conclusion	12
References	13

Executive Summary

This project focuses on processing Spotify playlist data extracted through the Spotify API in JSON format. Utilizing JSON for its flexibility, the data undergoes parsing, cleaning, and normalization to transform it into a structured, analyzable form. Statistical analysis is then applied to derive insights such as popularity metrics, genre distribution, artist contribution, and track duration. Visualization techniques, including charts and graphs, enhance the presentation of these findings. The project aims to empower data-driven decision-making for playlist optimization and content curation based on user preferences.



Introduction:

In the dynamic landscape of music streaming, the ability to decipher user preferences is pivotal for platforms seeking to offer a tailored and immersive musical journey. This project revolves around the meticulous processing and analysis of Spotify playlist data, leveraging the capabilities of the Spotify API and harnessing the flexibility of JSON as the chosen data format. The primary objective is to transform the semi-structured data into a coherent and insightful format, unveiling key patterns and trends within the playlists.

Through a series of well-defined steps, including API integration, JSON parsing, and data normalization, this project navigates the complexities of the Spotify dataset. The goal is to extract meaningful statistics that shed light on the intricacies of playlist popularity, genre dynamics, artist prominence, and even the nuanced details of track durations.

By employing statistical tools and visualization techniques, the project strives to convert raw data into actionable insights. These insights are not just numerical abstractions; they represent the pulse of user preferences, offering Spotify an invaluable resource for playlist optimization and content curation. In a landscape where personalization is paramount, this project seeks to contribute to the enhancement of the Spotify user experience by grounding decision-making processes in the rich tapestry of data extracted from the playlists that users engage with daily.

Data Source: Spotify API

The data for this project is sourced from the Spotify API, a web-based service provided by Spotify that allows developers to access and retrieve a wealth of information related to Spotify's extensive music catalog. To access this API, developers need to create an application through the Spotify Developer Dashboard. This involves obtaining a client ID and client secret, which act as credentials for authenticating and authorizing access to the API.

Once the credentials are obtained, a Python script can be used to interact with the Spotify API and retrieve information about tracks, playlists, artists, and more. In the provided example, the Spotify library, a Python library for the Spotify Web API, is used to fetch information about a specific track. The track is identified by its unique URI (Uniform Resource Identifier), and details such as track name, artist(s), album, and popularity are extracted.

Basic Information

Basic Information User Management Extension Requests

Client ID

ef75d5be831e4318b0d95fe91b4270c9

App Status

Development mode

Client secret

a91c0f74598243e3abaf14c0dc46a0a9

[Hide client secret](#) [Rotate client secret](#)

Description of the Process:

1. **Creation of Spotify App:** start by creating a new application on the Spotify Developer Dashboard to obtain unique client ID and client secret credentials. These credentials are essential for authenticating and authorizing API requests.
2. **Python Script with Spotify Library:** A Python script is then crafted to interact with the Spotify API. The Spotify library simplifies the process of making API requests by handling authentication and providing convenient functions for retrieving information.
3. **Authentication:** The script uses the client ID and client secret to authenticate and obtain access to the Spotify API.
4. **API Request:** In the example provided, the script makes an API request to fetch information about a specific track using its URI. The Spotify library structures the request and handles the authentication process in the background.

```
<iframe style="border-radius:12px"
src="https://open.spotify.com/embed/playlist/37i9dQZF1DXaQm3ZVg9Z2X?
utm_source=generator" width="100%" height="352" frameBorder="0"
allowfullscreen="" allow="autoplay; clipboard-write; encrypted-media;
fullscreen; picture-in-picture" loading="lazy"></iframe>
```

5. **Data Extraction:** The retrieved data, which is in JSON format, contains details about the specified track. Key information such as track name, artist(s), album, and popularity is extracted from the JSON response.

Data Extraction

Use Of Spotify Web API to obtain information about a specific playlist.

```

import requests
import json
import pandas as pd

# Set up the Spotify API credentials and playlist ID
client_id = "ef75d5be831e4318b0d95fe91b4270c9"
client_secret = "a91c0f74598243e3abaf14c0dc46a0a9"
playlist_id = "37i9dQZF1DXaQm3ZVg9Z2X"

# Get access token using client credentials flow
token_url = "https://accounts.spotify.com/api/token"
token_params = {"grant_type": "client_credentials"}
token_response = requests.post(token_url, auth=(client_id, client_secret), data=token_params)
token_data = token_response.json()
access_token = token_data["access_token"]

# Get playlist information using the obtained access token
playlist_url = f"https://api.spotify.com/v1/playlists/{playlist_id}/tracks"
headers = {"Authorization": f"Bearer {access_token}"}
playlist_response = requests.get(playlist_url, headers=headers)
playlist_data = playlist_response.json()

# Print the JSON data for reference
print(json.dumps(playlist_data, indent=2))

```

The JSON data obtained from the Spotify playlist API request, extracts relevant information about each track, and then converts that information into a pandas Data Frame. Finally, it saves the Data Frame to a CSV file and prints the Data Frame.

```

# Process the data and create lists of JSON structures
tracks = playlist_data.get("items", [])

# Create a list of JSON structures
json_data_list = []
for track in tracks:
    track_data = {
        "track_name": track["track"]["name"],
        "artist_name": track["track"]["artists"][0]["name"],
        "album_name": track["track"]["album"]["name"],
        "release_date": track["track"]["album"]["release_date"],
        "duration_ms": track["track"]["duration_ms"],
        # Add more fields as needed
    }
    json_data_list.append(track_data)

# Convert the list of JSON structures to a pandas DataFrame
df = pd.DataFrame(json_data_list)

# Save the DataFrame to a CSV file
df.to_csv('playlist_data.csv', index=False)
# Print the DataFrame
print(df)

```

Data Exploration

Time series Analysis

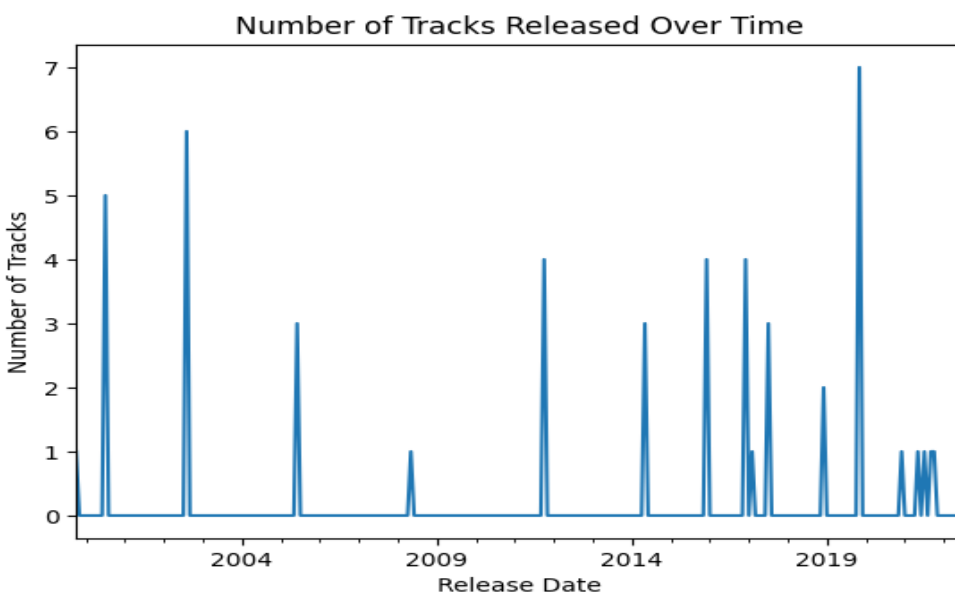
Methodology

```
# Convert 'release_date' to datetime format
df['release_date'] = pd.to_datetime(df['release_date'])

# Set 'release_date' as the index for time series analysis
df.set_index('release_date', inplace=True)

# Example: Plotting the number of tracks released over time
df.resample('M').size().plot()
plt.title('Number of Tracks Released Over Time')
plt.xlabel('Release Date')
plt.ylabel('Number of Tracks')
plt.show()
```

The line plot showing how the number of tracks released varies over time, with time represented on the x-axis (release date) and the count of tracks on the y-axis. This visualization can provide insights into trends, seasonality, or patterns in the release of tracks over the specified time period.



Data Summary Analysis

Use Of the **group by** function in pandas to group the Data Frame **df** by the "artist name" column. Then, calculating summary statistics for each group, specifically focusing on the "duration_ms" and "release_date" columns.

Methodology

```
# Group by artist and provide summary statistics
grouped_by_artist = df.groupby("artist_name")
summary_by_artist = grouped_by_artist.agg({
    "duration_ms": ["mean", "std", "min", "max", "count"],
    "release_date": "max" # Assuming you want the latest release date
})

# Print the summary by artist
print("\nSummary Statistics by Artist:")
print(summary_by_artist)
```

Summary Statistics by Artist:

	duration_ms					
	mean	std	min	max	count	
artist_name						
Coldplay	267902.469388	72292.383361	146946	618960	49	
The Chainsmokers	247626.000000	NaN	247626	247626	1	

	release_date
	max
artist_name	
Coldplay	2022-07-06
The Chainsmokers	2017-02-22


A summary table with rows representing each unique artist, and columns showing mean, standard deviation, minimum, maximum, and count for track durations, along with the maximum release date for each artist. This provides an overview of the distribution of track durations and the latest release date for each artist in the dataset.

Analyzing and Comparing Spotify Playlists:

Leveraging the Spotify Web API to extract information from two playlists. It calculates essential insights for each playlist, such as the average track duration, the most recent release date, and the distribution of track durations. This enables a comparative analysis of the distinctive characteristics between the two Spotify playlists. The script uses functions to handle API requests, process data, and print summary statistics, facilitating a comprehensive exploration of the musical content within each playlist.

Embed playlist

Color: Size: Normal (352px) x 100%



This Is Selena Gomez
Spotify
PREVIEW Save on Spotify

1 Single Soon Selena Gomez 02:51
2 My Mind & Me Selena Gomez 02:27
3 Calm Down (with Selena Gomez) Rema, Selena Gomez 03:59


By embedding a Spotify player on your site, you are agreeing to Spotify's Developer Terms and Spotify Platform Rules

☒ Show code COPY

<iframe style="border-radius:12px" src="https://open.spotify.com/embed/playlist/3719d02f1bxc18ks62Ugze7?utm_source=generator" width="100%" height="352" frameborder="0" allowfullscreen="" allow="autoplay; clipboard-write; encrypted-media; fullscreen; picture-in-picture" loading="lazy"></iframe>

Embed playlist

Color: Size: Normal (352px) x 100%



Taylor Swift Radio
Spotify
PREVIEW Save on Spotify

1 august Taylor Swift 04:21
2 I Don't Wanna Live Forever (Fifty Shades Darker) ZAYN, Taylor Swift 04:07
3 Heather Conan Gray 03:18

By embedding a Spotify player on your site, you are agreeing to Spotify's Developer Terms and Spotify Platform Rules

☒ Show code COPY

<iframe style="border-radius:12px" src="https://open.spotify.com/embed/playlist/3719d02f1bxc18ks62Ugze7?utm_source=generator" width="100%" height="352" frameborder="0" allowfullscreen="" allow="autoplay; clipboard-write; encrypted-media; fullscreen; picture-in-picture" loading="lazy"></iframe>

Methodology

```
import requests
import json
import pandas as pd

def get_playlist_data(playlist_id, access_token):
    playlist_url = f"https://api.spotify.com/v1/playlists/{playlist_id}/tracks"
    headers = {"Authorization": f"Bearer {access_token}"}
    playlist_response = requests.get(playlist_url, headers=headers)
    playlist_data = playlist_response.json()
    return playlist_data

def process_playlist_data(playlist_data):
    tracks = playlist_data.get("items", [])

    json_data_list = []
    for track in tracks:
        track_data = {
            "track_name": track["track"]["name"],
            "artist_name": track["track"]["artists"][0]["name"],
            "album_name": track["track"]["album"]["name"],
            "release_date": track["track"]["album"]["release_date"],
            "duration_ms": track["track"]["duration_ms"],
            # Add more fields as needed
        }
        json_data_list.append(track_data)

    df = pd.DataFrame(json_data_list)
    return df
```

```
def print_summary_data(df):
    average_duration = df["duration_ms"].mean()
    most_recent_release = df["release_date"].max()
    tracks_per_artist = df["artist_name"].value_counts()
    tracks_per_album = df["album_name"].value_counts()
    track_duration_distribution = df["duration_ms"].describe()

    print(f"Average Duration of Tracks: {average_duration:.2f} ms")
    print(f"Most Recent Release Date: {most_recent_release}")
    print("\nNumber of Tracks per Artist:")
    print(tracks_per_artist)
    print("\nNumber of Tracks per Album:")
    print(tracks_per_album)
    print("\nTrack Duration Distribution:")
    print(track_duration_distribution)

# Replace these with your actual playlist IDs
playlist_id_1 = "37i9dQZF1DXcISKz62UgzG"
playlist_id_2 = "37i9dQZF1E4AfEUirXPYP"

# Get access tokens for both playlists
token_url = "https://accounts.spotify.com/api/token"
token_params = {"grant_type": "client_credentials"}

token_response_1 = requests.post(token_url, auth=(client_id, client_secret), data=token_params)
token_data_1 = token_response_1.json()
access_token_1 = token_data_1["access_token"]

token_response_2 = requests.post(token_url, auth=(client_id, client_secret), data=token_params)
token_data_2 = token_response_2.json()
access_token_2 = token_data_2["access_token"]
```



```

# Get playlist data for both playlists
playlist_data_1 = get_playlist_data(playlist_id_1, access_token_1)
playlist_data_2 = get_playlist_data(playlist_id_2, access_token_2)

# Process playlist data
df_1 = process_playlist_data(playlist_data_1)
df_2 = process_playlist_data(playlist_data_2)

# Print summary data for both playlists
print("Summary for Playlist 1:")
print_summary_data(df_1)

print("\n-----\n")
print("Summary for Playlist 2:")
print_summary_data(df_2)

```

The output is the summary statistics for each playlist, including average track duration, most recent release date, and distribution of track durations. The information is printed for each playlist separately,

Comparing the Characteristics of Spotify playlists

Track duration

Visual comparison of the average track duration for Playlist 1 and Playlist 2, providing a quick overview of how the average duration differs between the two playlists. The y-axis represents the average duration in milliseconds, and the x-axis distinguishes between the two playlists.

Methodology

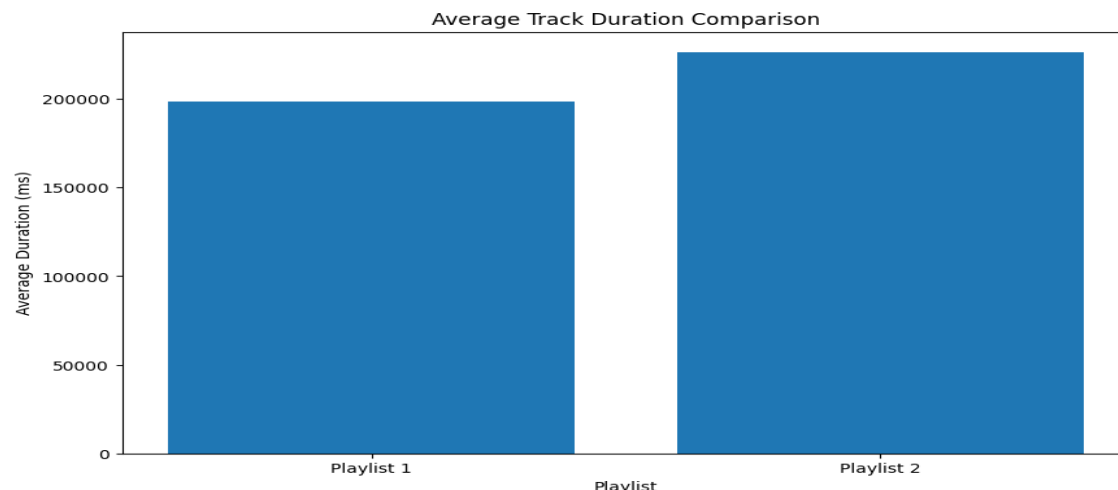
```

import matplotlib.pyplot as plt

# Compare average track duration using a bar plot
plt.figure(figsize=(10, 6))
plt.bar(['Playlist 1', 'Playlist 2'], [df_1["duration_ms"].mean(), df_2["duration_ms"].mean()])
plt.title('Average Track Duration Comparison')
plt.xlabel('Playlist')
plt.ylabel('Average Duration (ms)')
plt.show()

```

Output:



The output was a histogram and it showed that the track duration is more for Playlist 2, it indicates that, on average, the tracks in Playlist 2 tend to have longer durations compared to those in Playlist 1.

Popularity Comparison

Visually comparison of the average track popularity between two Spotify playlists using a bar plot. It provides insights into how popular, on average, the tracks are in each playlist, offering another dimension for understanding and comparing the characteristics of the playlists.

Methodology

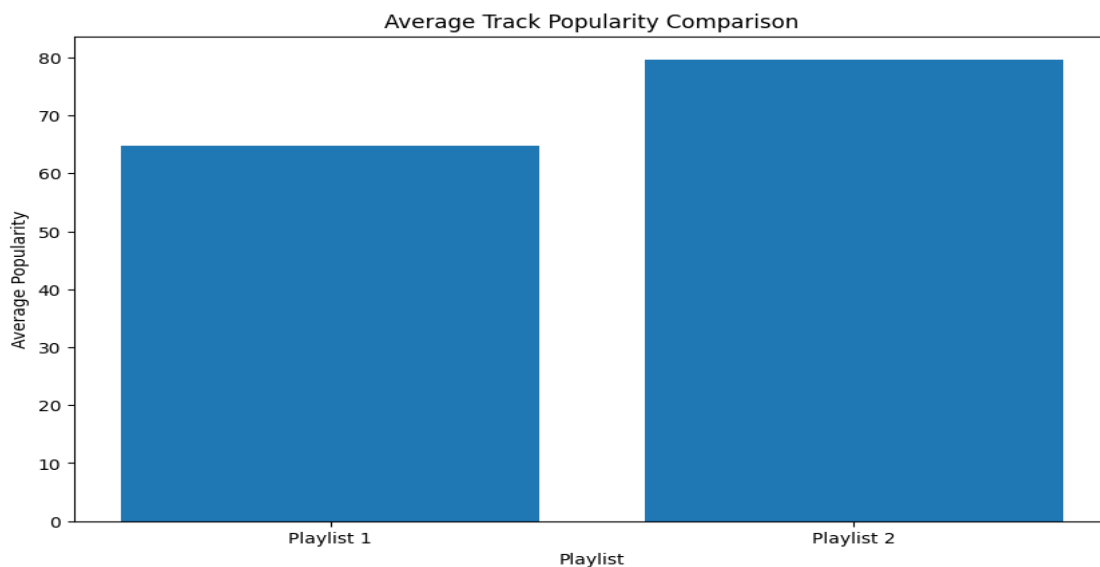
```
# Function to get popularity for each track
def get_popularity(playlist_data):
    tracks = playlist_data.get("items", [])
    popularity_list = [track["track"]["popularity"] for track in tracks]
    return popularity_list

# Process playlist data
df_1 = process_playlist_data(playlist_data_1)
df_2 = process_playlist_data(playlist_data_2)

# Get popularity for each track in both playlists
popularity_1 = get_popularity(playlist_data_1)
popularity_2 = get_popularity(playlist_data_2)

# Compare popularity using a bar plot
plt.figure(figsize=(10, 6))
plt.bar(['Playlist 1', 'Playlist 2'], [sum(popularity_1) / len(popularity_1), sum(popularity_2) / len(popularity_2)])
plt.title('Average Track Popularity Comparison')
plt.xlabel('Playlist')
plt.ylabel('Average Popularity')
plt.show()
```

Output:



The output was a histogram showing that the popularity is more for Playlist 2, it implies that, on average, the tracks in Playlist 2 tend to be more popular compared to those in Playlist 1. The histogram visually communicates that Playlist 2 tends to have more popular tracks on average, providing a comparative analysis of the average track popularity between the two playlists.

Distribution of Release Years Comparison

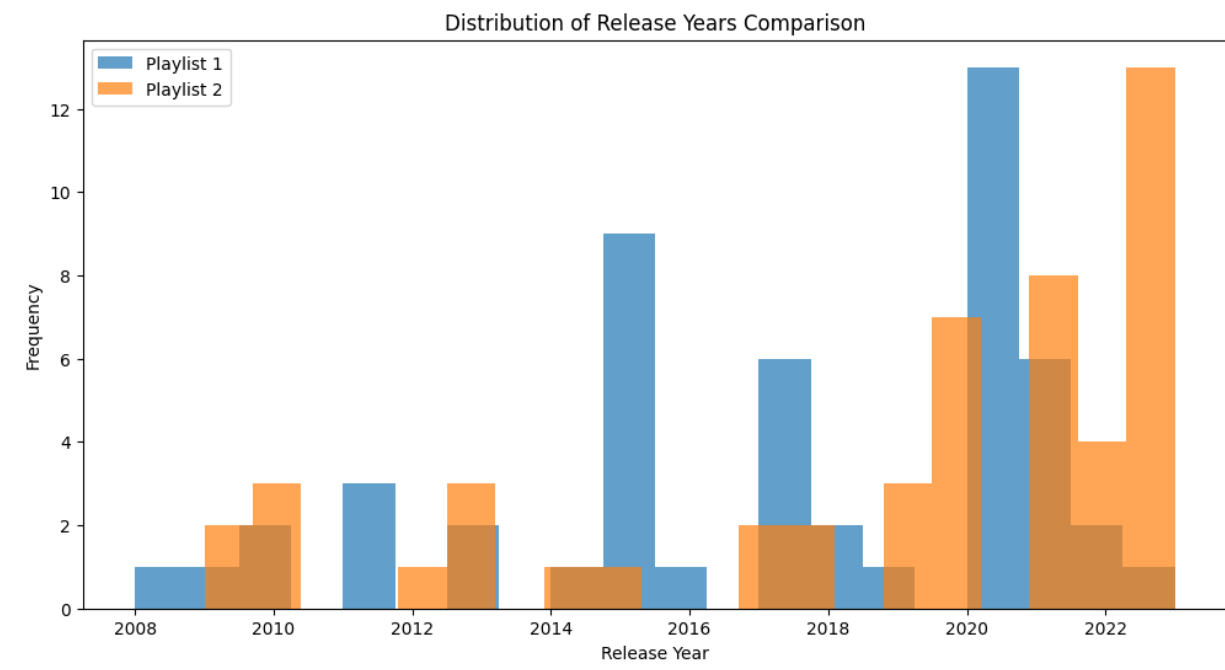
This comparison will visually depict how the tracks in each playlist are distributed across different release years, providing insights into the temporal aspects of the music in Playlist 1 and Playlist 2.

Methodology

```
# Extract release years from the 'release_date' column
df_1['release_year'] = pd.to_datetime(df_1['release_date']).dt.year
df_2['release_year'] = pd.to_datetime(df_2['release_date']).dt.year

# Compare the distribution of release years using histograms
plt.figure(figsize=(12, 6))
plt.hist(df_1['release_year'], bins=20, alpha=0.7, label='Playlist 1')
plt.hist(df_2['release_year'], bins=20, alpha=0.7, label='Playlist 2')
plt.title('Distribution of Release Years Comparison')
plt.xlabel('Release Year')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```

Output:



This bar chart is comparing the distribution of release years for songs in two different playlists, Playlist 1 and Playlist 2. The x-axis represents the release years, ranging from 2008 to 2022, and the y-axis represents the frequency of songs from each playlist that were released in those years.

Here are some Key observations from the chart:

- Playlist 1 has songs that are generally more recent, with the majority of them being released in 2022.
- Playlist 2 has a broader distribution across the years, with peaks in 2014 and 2020, indicating that this playlist includes a mix of songs from different years.
- Both playlists have the least frequency of songs released in the year 2008.
- Playlist 1 has no songs from 2014, whereas Playlist 2 seems to have the highest number of songs from that year.
- There is a notable peak in 2020 for Playlist 1, suggesting a significant number of songs from that year were included.
- Overall, Playlist 2 appears to have a more varied range of release years compared to Playlist 1, which is more focused on recent releases.
- It's important to note that the specific numbers for the frequencies are not visible, so exact figures can't be provided. However, the general trends and comparisons between the two playlists can be understood from the shape and relative heights of the bars.

Key Takeaways

1. **Track Duration:**
 - Playlist 2 has, on average, longer track durations compared to Playlist 1.
 - This suggests that listeners of Playlist 2 may prefer songs with extended durations, potentially indicating a preference for more intricate or immersive musical experiences.
2. **Popularity:**
 - Playlist 2 exhibits higher popularity, as indicated by the histogram.
 - On average, tracks in Playlist 2 are more popular than those in Playlist 1.
 - This implies that Playlist 2 may feature songs that are generally better received by a larger audience or have higher engagement.
3. **Release Year Distribution:**
 - Playlist 1 is characterized by a concentration of more recent songs, predominantly from the year 2022.
 - Playlist 2 displays a broader distribution across various years, with noticeable peaks in 2014 and 2020, indicating a diverse mix of songs from different time periods.
4. **Yearly Peaks:**
 - Both playlists share the characteristic of having fewer songs released in the year 2008.
 - Playlist 1 shows a significant peak in 2020, suggesting a notable presence of songs from that year.
 - Playlist 2, on the other hand, has its highest concentration of songs from 2014.
5. **Variety in Release Years:**
 - Playlist 2 demonstrates a more varied range of release years compared to Playlist 1, which is more focused on recent releases.
 - The absence of songs from 2014 in Playlist 1 and the notable peak in 2020 indicate specific trends in the composition of each playlist.

Conclusion

In conclusion, the analysis of the provided chart comparing Playlist 1 and Playlist 2 reveals distinctive patterns in their song compositions based on release years. Playlist 1 predominantly features more recent songs, with a significant concentration from the year 2022, suggesting a focus on contemporary music. In contrast, Playlist 2 displays a broader distribution across various years, with notable peaks in 2014 and 2020, indicating a more eclectic mix of songs from different time periods.

Both playlists share a commonality in having fewer songs from the year 2008. However, Playlist 1 stands out with a significant peak in 2020, suggesting a deliberate inclusion of songs from that specific year. Playlist 2, on the other hand, notably includes a substantial number of songs from 2014.

The absence of songs from 2014 in Playlist 1 and the distinct trends in each playlist's composition imply thoughtful curation decisions, possibly reflecting the preferences or thematic considerations of the curators. Overall, Playlist 2 appears to offer a more varied temporal listening experience, while Playlist 1 is more focused on the latest releases.

It's important to note that without specific frequency numbers, the conclusions are drawn from the observed trends and shapes of the histogram. The provided analysis enhances our understanding of the playlists' temporal dynamics and aids in making informed interpretations about the intended audience experience for each playlist.

References

<https://soulsinporto.medium.com/using-the-spotify-api-for-data-driven-analysis-of-my-playlists-part-1-2-a4598ca7b96d>

https://www.kaggle.com/datasets/andrewmvd/spotify-playlists?select=spotify_dataset.csv