Documentation:
Spotify Music Data Collection & Analysis

Daniella Tahchi - 19/10/25

Outline:

Introduction

- Brief overview of the project
- Purpose of the notebook

Data Collection: Obtaining an Access Token

- Explanation of the Spotify API and the need for an access token
- Steps involved in obtaining the client ID and secret
- Code for encoding credentials and requesting the token

Data Extraction: Gathering Track Information

- Description of the function to extract playlist data
- Details about the information collected for each track (name, artist, album, popularity, etc.)
- How the Spotify API is used to get this information
- Code for the `get_trending_playlist_data` function
- Example usage of the function with a specific playlist ID

Exploratory Data Analysis (EDA)

- Overview of the EDA process
- Initial inspection of the data (e.g., using `df.info()`)
- Highlighting high-popularity tracks
- Visualizations created:
    - Bar plot of tracks per album
    - Pie chart of album popularity percentages
    - Bar plot of average popularity by artist
    - Word cloud of track names
- Interpretation of the visualizations and insights gained

Introduction

This notebook serves as a comprehensive guide to collecting and analyzing music data from the Spotify API. The primary goal is to extract detailed information about tracks from a specified playlist, focusing on attributes like popularity, release date, and explicit content. Subsequently, the notebook performs exploratory data analysis (EDA) to uncover insights and visualize trends within the collected dataset.

Data Collection: Obtaining an Access Token

Interacting with the Spotify API to retrieve data requires authentication in the form of an access token. This token acts as a key, granting permission to access Spotify's resources on behalf of an application.

The first step in obtaining an access token is to have a Spotify Developer account and create an application to get the CLIENT_ID and CLIENT_SECRET. These credentials uniquely identify your application to the Spotify API.

The following code snippet stores the obtained CLIENT_ID and CLIENT_SECRET:

```
# save ID and secret
CLIENT_ID = "49a7bfa69cc343018f1c5362804f804c"
CLIENT_SECRET = "d881bac129b545a9b30bb4395b19c0c1"
```

To request an access token, these client credentials need to be encoded using Base64. This is a standard encoding scheme that converts binary data into an ASCII string format, suitable for transmission over the internet.

```
# encode credentials using base64
client_credentials = f"{CLIENT_ID}:{CLIENT_SECRET}"
client_credentials_base64 = base64.b64encode(client_credentials.encode())
```

The next code block demonstrates how the client credentials are encoded and then used to make a POST request to the Spotify accounts service token endpoint (https://accounts.spotify.com/api/token). The grant_type parameter is set to client_credentials to indicate the type of authorization flow being used.

```
# request the access token
token_url = 'https://accounts.spotify.com/api/token'
headers = {
    'Authorization': f'Basic {client_credentials_base64.decode()}'
}
data = {
    'grant_type': 'client_credentials'
}
response = requests.post(token_url, data=data, headers=headers)
```

After sending the request, the Spotify API responds with a JSON object containing the access token and its expiration time, among other details, if the request is successful. The code then checks the response status code. If it's 200 (OK), it extracts the access_token from the JSON response. Otherwise, it prints an error message and exits.

```
if response.status_code == 200:
    access_token = response.json()['access_token']
    print("Access token obtained successfully.")
else:
    print("Error obtaining access token.")
    exit()
```

This access token is then used in subsequent API calls to fetch data like playlist tracks and their features.

Data Extraction: Gathering Track Information

With the access token successfully obtained, the next step is to extract detailed information about the tracks within a specific Spotify playlist. A Python function, get_trending_playlist_data, is defined to handle this process.

This function takes two input parameters:

- playlist_id: A string representing the unique identifier of the Spotify playlist from which to extract data.
- access_token: The obtained access token string, used for authenticating API requests.

Inside the function, the spotipy library, a lightweight Python library for the Spotify Web API, is utilized. An instance of the spotipy.Spotify class is created, authenticated with the provided access_token.

The function then retrieves the tracks from the specified playlist using sp.playlist_tracks. It specifically requests the items field, structured to include the track's ID, name, artists, and album information (ID and name).

For each track in the playlist, the function extracts the following information:

- **Track Name:** The title of the song.
- **Artists:** A comma-separated string of the artists' names.
- **Album Name:** The name of the album the track belongs to.
- **Album ID:** The unique identifier of the album.
- **Track ID:** The unique identifier of the track.
- **Popularity:** A numerical score indicating the track's popularity on Spotify (0-100).
- **Release Date:** The release date of the album.
- **Explicit:** A boolean indicating whether the track contains explicit content.
- **External URLs:** The Spotify URL for the track.

The popularity and release date are fetched by making additional API calls for each track and album, respectively. The extracted information for each track is stored as a dictionary, and these dictionaries are appended to a list. Finally, this list of dictionaries is converted into a pandas DataFrame.

Here is the code for the get_trending_playlist_data function:

```
# define the function
def get_trending_playlist_data(playlist_id, access_token):
    # set up spotipy with the access token
    sp = spotipy.Spotify(auth=access_token)
    # get the tracks from the playlist
    playlist_tracks = sp.playlist_tracks(playlist_id, fields='items(track(id, name, artists, album(id, name)))')

    # extract relevant information and store in a list of dictionaries
    music_data = []
    for track_info in playlist_tracks['items']:
        track = track_info['track']
        track_name = track['name']
        artists = ', '.join([artist['name'] for artist in track['artists']])
        album_name = track['album']['name']
        album_id = track['album']['id']
        track_id = track['id']
```

```
        # get release date of the album
        try:
            album_info = sp.album(album_id) if album_id != 'Not available' else None
            release_date = album_info['release_date'] if album_info else None
        except:
            release_date = None

        # get popularity of the track
        try:
            track_info = sp.track(track_id) if track_id != 'Not available' else None
            popularity = track_info['popularity'] if track_info else None
        except:
            popularity = None
```

```
            # add additional track information to the track data
            track_data = {
                'Track Name': track_name,
                'Artists': artists,
                'Album Name': album_name,
                'Album ID': album_id,
                'Track ID': track_id,
                'Popularity': popularity,
                'Release Date': release_date,
                'Explicit': track_info.get('explicit', None),
                'External URLs': track_info.get('external_urls', {}).get('spotify', None),

            }

            music_data.append(track_data)

        # create a pandas dataframe from the list of dictionaries
        df = pd.DataFrame(music_data)

        return df
```

Below is an example of how the get_trending_playlist_data function is used with a specific playlist ID and the resulting DataFrame is displayed:

```
# the id of "And, baby, that's show business for you" playlist
playlist_id = '65uAjFTt4N8sEJeonhNOBL'

# call the function to get the music data from the playlist and store it in a DataFrame
music_df = get_trending_playlist_data(playlist_id, access_token)
```

This DataFrame music_df contains the extracted information for each track in the specified playlist and will be used for subsequent exploratory data analysis.

Exploratory Data Analysis (EDA)

After successfully collecting and extracting the music data, the next crucial step is to perform Exploratory Data Analysis (EDA). The goal of EDA is to understand the structure, content, and characteristics of the dataset, identify patterns, detect anomalies, and gain insights that can inform further analysis or modeling.

Initial Data Inspection

The first step in EDA is to get a general overview of the DataFrame. The info() method provides a concise summary of the DataFrame, including the index dtype and column dtypes, non-null values, and memory usage.

```
music_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34 entries, 0 to 33
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Track Name     34 non-null     object
 1   Artists        34 non-null     object
 2   Album Name     34 non-null     object
 3   Album ID       34 non-null     object
 4   Track ID       34 non-null     object
 5   Popularity     34 non-null     int64 
 6   Release Date   34 non-null     object
 7   Explicit       34 non-null     bool  
 8   External URLs  34 non-null     object
dtypes: bool(1), int64(1), object(7)
memory usage: 2.3+ KB
```

This output confirms that our DataFrame music_df contains 34 entries and provides details about each column, such as the data type and the number of non-null entries. We can see that all columns have 34 non-null entries, indicating no missing values in this dataset. The data types are appropriate for each column (e.g., int64 for Popularity, object for strings, bool for Explicit).

Highlighting High-Popularity Tracks

To quickly identify the most popular tracks in the playlist, we can use the DataFrame.style.apply method to visually highlight rows where the 'Popularity' is above a certain threshold (in this case, greater than 90).

```
# highlight top 5 most popular tracks in the dataframe
def highlight_high_popularity(s):
    """
    Highlights high popularity 90 threshold across the row.
    """
    is_high = s['Popularity'] > 90
    return ['background-color: lightgreen' if is_high else '' for v in s]

music_df.style.apply(highlight_high_popularity, axis=1, subset=['Popularity'])
```

This styling helps in visually distinguishing the tracks that are performing exceptionally well based on their popularity score.

Visualizations

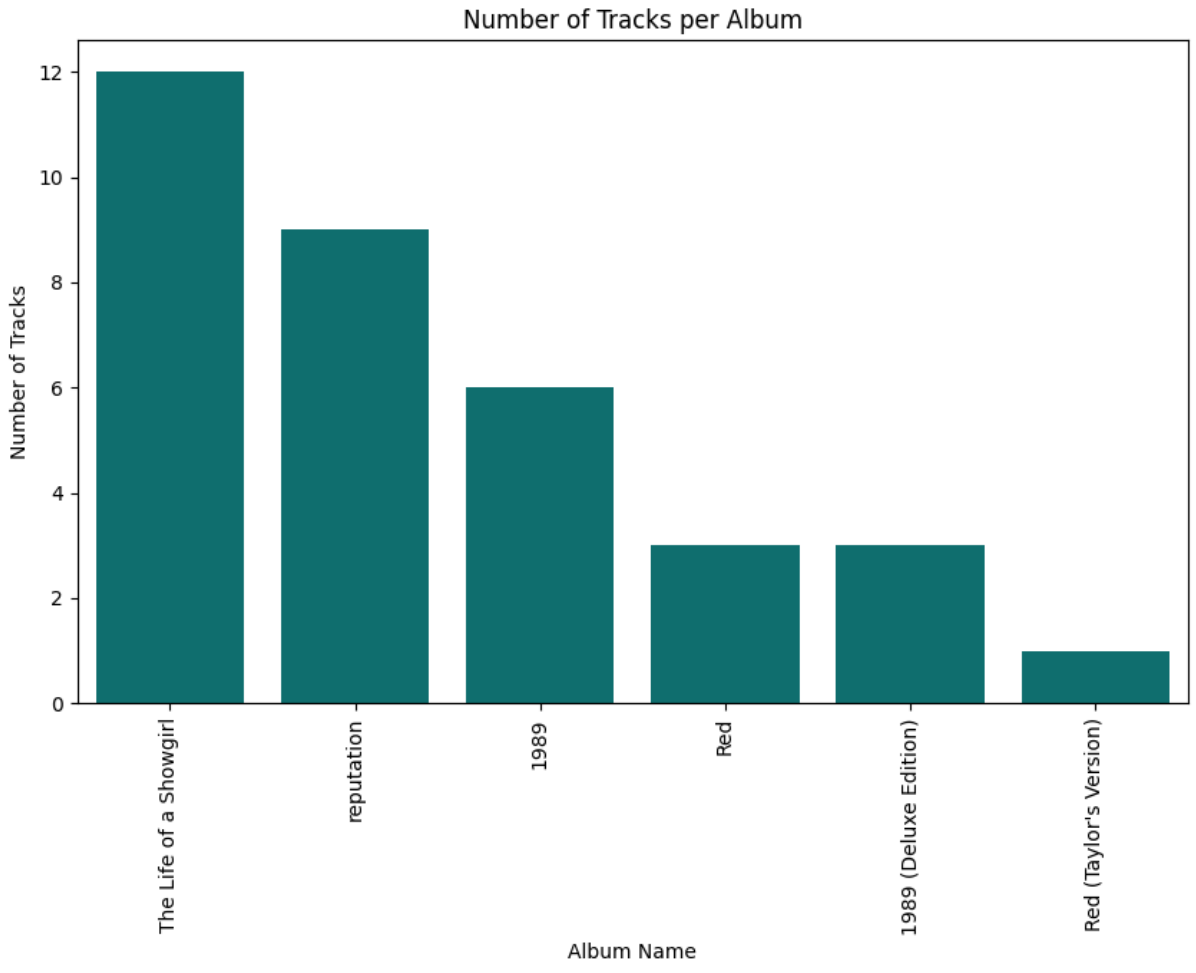
Visualizations are powerful tools in EDA to uncover patterns and trends that might not be obvious from raw data. We will create several plots to gain insights into the music data.

1- Number of Tracks per Album

This bar plot shows the distribution of tracks across different albums within the playlist. It helps us understand which albums are most represented in this particular playlist.

```
# Create some visualizations
import matplotlib.pyplot as plt
import seaborn as sns

# barplot highlighting the number of tracks in this playlist from each album
plt.figure(figsize=(10, 6))
sns.countplot(data=music_df, x='Album Name', order=music_df['Album Name'].value_counts().index, color='teal')
plt.xticks(rotation=90)
plt.xlabel('Album Name')
plt.ylabel('Number of Tracks')
plt.title('Number of Tracks per Album')
plt.show()
```



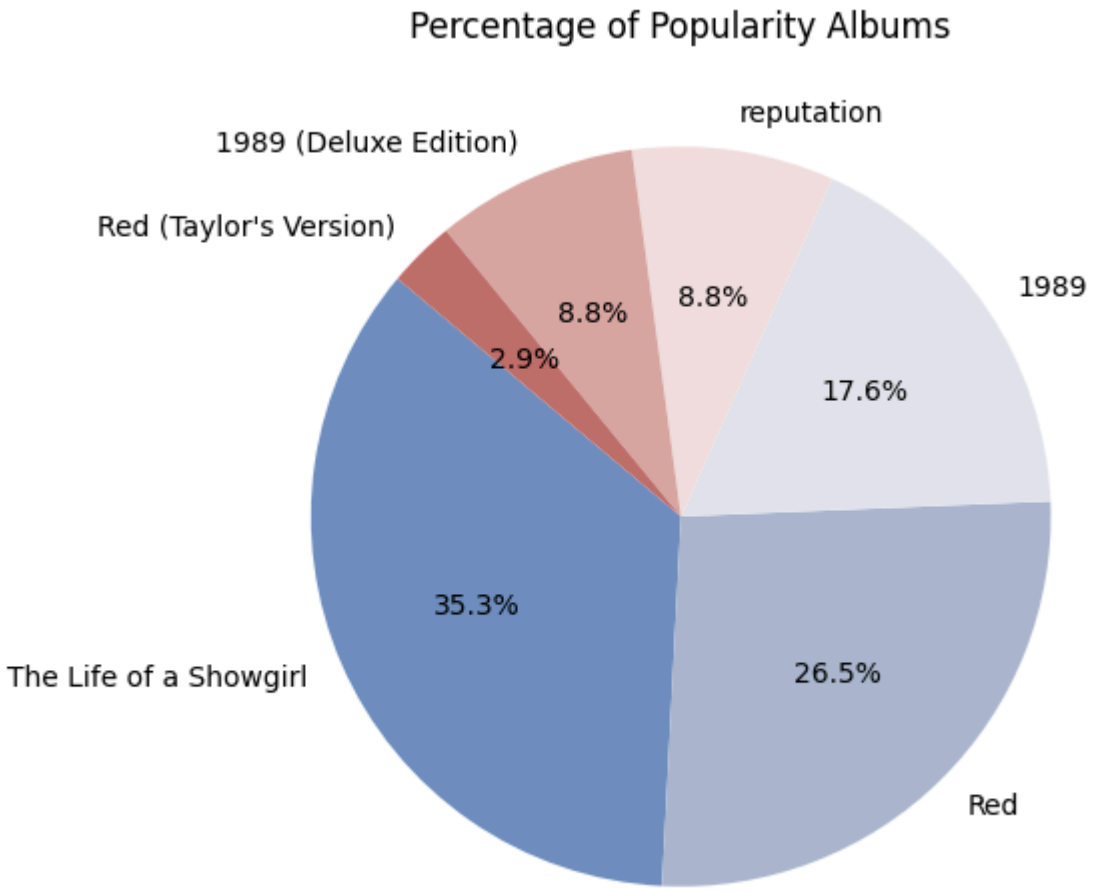
Insight: The plot clearly shows that 'The Life of a Showgirl' album has the highest number of tracks in this playlist, followed by 'reputation' and '1989'. This suggests a strong focus on these albums within the playlist's selection.

2- Percentage of Popularity by Album

This pie chart illustrates the proportion of tracks from each album in the playlist, giving a sense of the overall album representation. While titled "Percentage of Popularity Albums," it actually represents the percentage of tracks contributed by each album to the playlist.

```
# Pie chart showcasing the percentage of popularity of albums
album_popularity_counts = music_df['Album Name'].value_counts()

plt.figure(figsize=(10, 6))
plt.pie(album_popularity_counts, labels= music_df['Album Name'].unique(), autopct='%1.1f%%',
        startangle=140,colors=sns.color_palette("vlag"))
plt.title('Percentage of Popularity Albums')
plt.show()
```



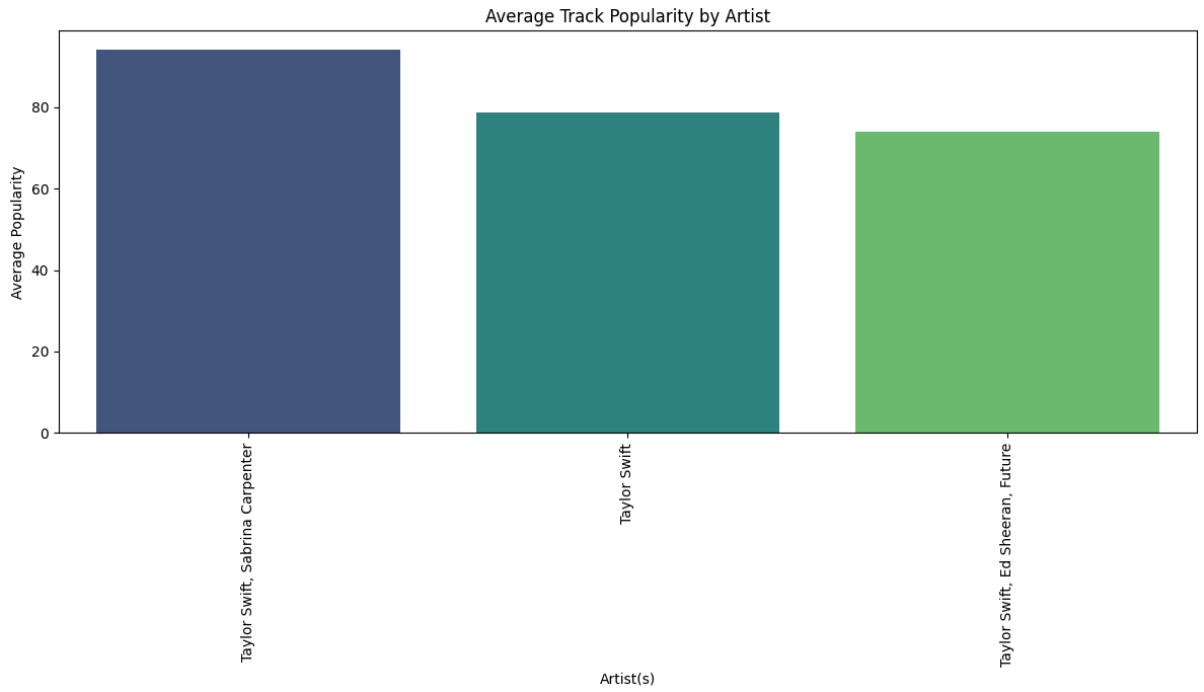
Insight: The pie chart visually reinforces the dominance of 'The Life of a Showgirl' in the playlist by showing its large slice of the pie. It also allows for quick comparison of the representation of other albums.

3- Average Track Popularity by Artist

This bar plot displays the average popularity score for tracks by each artist (or group of artists) in the playlist. It helps identify which artists have the most popular tracks on average within this specific collection.

```
# Calculate the average popularity for each artist
avg_popularity_by_artist = music_df.groupby('Artists')['Popularity'].mean().sort_values(ascending=False)

# Create a bar plot of average popularity by artist
plt.figure(figsize=(12, 7))
sns.barplot(x=avg_popularity_by_artist.index, y=avg_popularity_by_artist.values, palette='viridis')
plt.xticks(rotation=90)
plt.xlabel('Artist(s)')
plt.ylabel('Average Popularity')
plt.title('Average Track Popularity by Artist')
plt.tight_layout()
plt.show()
```



Insight: The plot shows that the collaboration between Taylor Swift and Sabrina Carpenter has the highest average popularity in this playlist, followed by Taylor Swift alone, and then the collaboration with Ed Sheeran and Future. This indicates that tracks featuring Sabrina Carpenter are particularly popular in this playlist.

4- Word Cloud of Track Names

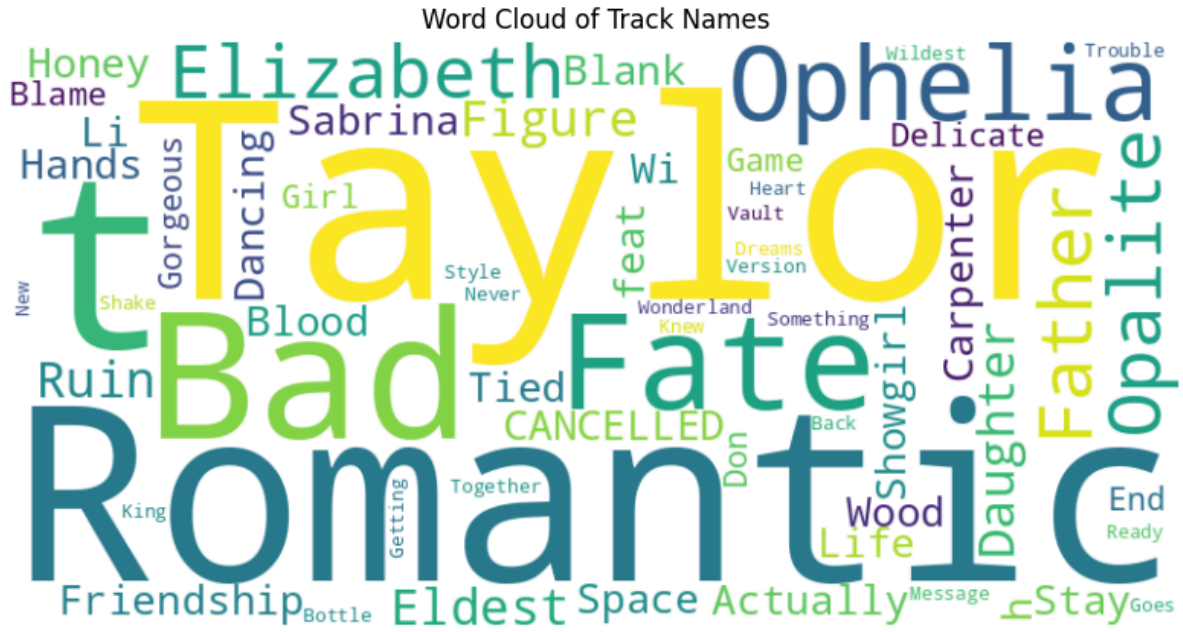
A word cloud provides a visual representation of the most frequent words in the track names. Larger words indicate higher frequency.

```
# Create a word cloud of track names
from wordcloud import WordCloud

# Combine all track names into a single string
all_track_names = ' '.join(music_df['Track Name'].tolist())

# Generate the word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(all_track_names)

# Display the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Track Names')
plt.show()
```



Insight: The word cloud highlights the most common words appearing in the track names, giving a quick sense of recurring themes or popular terms used in the titles within this playlist.

Summary:

- ➔ The documentation article provides a comprehensive guide covering data collection from the Spotify API, data extraction of track information from a playlist, and exploratory data analysis.
- ➔ The data collection section explains obtaining an access token using client credentials and the Base64 encoding method for authentication with the Spotify API's token endpoint.
- ➔ The data extraction section details a Python function (get_trending_playlist_data) that utilizes the spotipy library to retrieve track information (name, artists, album, popularity, release date, explicit content, external URLs) from a specified playlist ID.
- ➔ The EDA section describes the initial inspection of the data using df.info(), highlighting high-popularity tracks, and various visualizations including:
    - ◆ A bar plot showing that 'The Life of a Showgirl' album has the highest number of tracks in the playlist.
    - ◆ A pie chart reinforcing the dominance of 'The Life of a Showgirl' album in terms of track contribution.
    - ◆ A bar plot indicating that the collaboration between Taylor Swift and Sabrina Carpenter has the highest average popularity in this playlist.
    - ◆ A word cloud visualizing the most frequent words appearing in the track names.