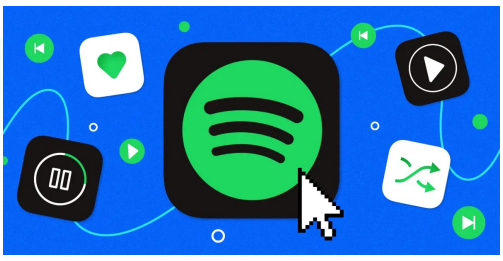
# Informatics Practices (065)
# Project File

# Spotify Song Features Analysis

**NAME: ARNAV MANGLA**
**CLASS: 12-C**



# Certificate

This is to certify that the content of this project "**Spotify Song Features Analysis**" by "**Arnav Mangla**" of class 12-C is the bonafide work of him submitted to "**Amity International School, Mayur Vihar**", for consideration in the partial accomplishment of the provision of CBSE, Delhi.

The original research and analysis work was carried out by him under my supervision in the academic year 2021-2022. On the basis of declaration made by him, I recommend the project for evaluation

.....
(Signature of **Teacher**)

.....
(Signature of **External Examiner**)

.....
(Signature of **Principal**)

# Acknowledgement

I would like to express my sincere gratitude to my teacher for the subject of Informatics Practices, **Ms. Pooja Thakur**, for igniting my interest in the subject of Informatics Practices and in the field of Computer Science allowing me to push myself, learn newer languages, build projects, and develop my problem-solving ability. I would like to thank her also for the constructive advice and guidance provided to me by her allowing me to successfully complete this project.

I would also like to thank my **parents** for their motivation and their support.

Thank you to **everyone** who helped.

# Synopsis

*"Music is the one incorporeal entrance into the higher world of knowledge which comprehends mankind but which mankind cannot comprehend."*

*~ Ludwig van Beethoven*

While surfing through *kaggle.com*, looking for ideas for this very project, I came across a dataset (*kaggle.com/aeryan/spotify-music-analysis/data*) which had a few songs and very interesting features of these songs. However, the dataset wasn't to my liking as it had very few songs that I personally liked. I tried to find other datasets like this, but found none, which gave me an idea to create such a dataset myself.

Upon some research, I came across Spotify's Web API (*developer.spotify.com/documentation/web-api/*). After a few Google searches and help from Stack Overflow and the official docummentaion of the API, I was able to learn how to use it and created my own dataset.
The data thus generated was stored in a csv file for quick access and the csv file was then converted to a pandas Dataframe for data analysis.

The project was created using the language **Python** in JetBrains' IDE **PyCharm** (Edu). Google's cloud-based Python IDE **Colaboratory** was used as a learning ground for the Spotify API. **Github** was used for version control and for better access to the project files.

**Link** to repository: *github.com/Arnav3094/IP-Project*

The **libraries** used in the project are:

- pandas,
- matplotlib.pyplot,
- Matplotlib.ticker,
- numpy,
- requests,
- base64,
- datetime,
- urllib.parse,
- PIL.Image,
- difflib, and
- pprint

# Features of the Project

**Files** in the project:

- Base_SpotifyAPI_Class.py
- FeaturesSpotifyAPI_Class.py
- create.py
- main.py

An object of the class SpotifyAPI (Base Class) in the Base_SpotifyAPI_Class.py file enables us only to authenticate into the API and then is built upon by the FeaturesSpotifyAPI (Derived Class) in the FeaturesSpotifyAPI_Class.py file. An object of this, created in the create.py file in the main() function, accepts the lists of Artists, client_id and client_secret, and fetches all the data used in the project. This is first taken into a pandas Dataframe, which is converted to a CSV, which is later on extracted back into a DataFrame for quick access.

The dataset was generated using Spotify's Web API (*developer.spotify.com*). A list of 100 artists was created manually by going through my personal library of songs. This list was firstly used in the **search endpoint** of the API to get the IDs for the artists. A list of these IDs was then used in the **artists endpoint** of the API to get the top 10 ten tracks of each of these artists. This query not only returns the IDs of the tracks but also a few basic datapoints about the songs like duration and popularity. Simultaneously, a list of the IDs of the songs was also created, which was then used in the **tracks endpoint** of the API to get the interesting features of the songs like "danceability", "energy", "acousticness", etc.

List of the artists: (sorted alphabetically)

```
['20syl', '2pac', '50 Cent', '6LACK', 'A$AP Rocky', 'ABBA', 'AC/DC', 'Adele', 'Alec Benjamin',
'Alicia Keys', 'Anson Seabra', 'Ariana Grande', 'AURORA', 'Avicii', 'Backstreet Boys',
'Bastille', 'Beyonce', 'Billie Eilish', 'Black Eyed Peas', 'Bob Marley', 'Bon Jovi', 'Britney
Spears', 'Bruno Mars', 'Camila Cabello', 'Cardi B', 'Christian French', 'Closed on Sunday',
'Coldplay', 'David Bowie', 'Dean Lewis', 'Dempsey Hope', 'Denzel Curry', 'Dj Snake', 'Dr.
Dre', 'Drake', 'Dua Lipa', 'Duncan Laurence', 'Ed Sheeran', 'Ellie Goulding', 'Elton John',
'Elvis Presley', 'Eminem', 'Frank Ocean', 'Glass Animals', 'Green Day', 'Halsey', 'Hopsin',
'Hozier', 'Imagine Dragons', 'J. Cole', 'JAY-Z', 'Joey Bada$$', 'John Lennon', 'Jon Bellion',
'Joyner Lucas', 'JP Saxe', 'Juice WRLD', 'Justin Bieber', 'Justin Timberlake', 'Kanye West',
'Katy Perry', 'Kendrick Lamar', 'Khalid', 'Kid Cudi', 'Kygo', 'Lady Gaga', 'Lauv', 'Linkin
Park', 'Logic', 'Lukas Graham', 'Mac Miller', 'Machine Gun Kelly', 'Maroon 5', 'Martin
Garrix', 'Meek Mill', 'Michael Jackson', 'Miley Cyrus', 'Nicki Minaj', 'One Direction', 'Paul
McCartney', 'Post Malone', 'Powfu', 'Pusha-T', 'Queen', 'Rihanna', 'Sam Smith', 'Selena
Gomez', 'Shawn Mendes', 'Skrillex', 'Sting', 'SZA', 'Taylor Swift', 'The Beatles', 'The
Chainsmokers', 'The Notorious B.I.G.', 'The Weeknd', 'Travis Scott', 'Whitney Houston', 'X
Ambassadors', 'XXXTENTACION']
```

The main.py file comes into play for the analysis portion. Where a menu driven is used to lead the user to different menus for different datapoints where each datapoint is analysed and visualised using graphs created using pyplot module of the library Matplotlib.

# Dataset Generation

## Base_SpotifyAPI_Class.py

```
import requests
import base64
import datetime as dt
from urllib.parse import urlencode
from pprint import pprint as pp
import pandas as pd
from pandas import DataFrame as DF
from pandas import Series as S
import json
import time


def get_token_data():
    return {'grant_type': 'client_credentials'}

class SpotifyAPI:
    access_token = None
    access_token_expires = None
    client_id = None
    client_secret = None
    token_url = 'https://accounts.spotify.com/api/token'

    def __init__(self, client_id, client_secret):
        self.client_id = client_id
        self.client_secret = client_secret
        self.token_url = 'https://accounts.spotify.com/api/token'
        self.client_creds = None
        self.access_token_did_expire = None

    def get_client_creds(self):
        """
        Returns base64 encoded client_creds
        """
        assert self.client_secret, self.client_id is not None
        self.client_creds = base64.b64encode(f"{self.client_id}:
{self.client_secret}".encode()).decode()
        return self.client_creds

    def get_token_headers(self):
        client_creds_b64 = self.get_client_creds()
        return {'Authorization': f'Basic {client_creds_b64}'}

    def auth(self):
        r = requests.post(self.token_url, data=get_token_data(),
headers=self.get_token_headers())
        if r.status_code not in range(200, 299):
            print(f"Something went wrong.\n[Status Code] {r.status_code}")
            return False
        data = r.json()
        now = dt.datetime.now()
        self.access_token = data['access_token']
        expires_in = data['expires_in']
        self.access_token_expires = now + dt.timedelta(seconds=expires_in)
        self.access_token_did_expire = self.access_token_expires <= now
       return self.access_token
```

## FeaturesSpotifyAPI_Class.py

```
from Base_SpotifyAPI_Class import *

class FeaturesSpotifyAPI(SpotifyAPI):
    def __init__(self, client_id, client_secret, artists):
        super().__init__(client_id, client_secret)
        self.country_code = 'US'
        self.artists = artists
```

```
        self.artist_ids = []
        self.track_ids = []
        self.headers = None
        self.top_tracks = []
        self.top_track_features = []
        self.top_track_details = []
        self.df = []

    def auth(self):
        r = requests.post(self.token_url, data=get_token_data(),
headers=self.get_token_headers())
        if r.status_code not in range(200, 299):
            raise Exception("Something went wrong.\n[Status Code] {r.status_code}")
        else:
            data = r.json()
            now = dt.datetime.now()
            access_token = data['access_token']
            expires_in = data['expires_in']
            self.access_token = access_token
            self.access_token_expires = now + dt.timedelta(seconds=expires_in)
            self.access_token_did_expire = self.access_token_expires <= now
            self.headers = {'Authorization': f'Bearer {access_token}'}
            print('Authorised\n')
            return access_token

    def get_artist_ids(self):
        print("Getting Artist IDs")
        artist_ids = []
        for artist in self.artists:
            data = urlencode({'q': artist, 'type': 'artist'})
            endpoint = f'https://api.spotify.com/v1/search?{data}'
            r = requests.get(endpoint, headers=self.headers)
            id_ = r.json()['artists']['items'][0]['id']
            artist_ids.append(id_)
            self.artist_ids = artist_ids
        print('Got Artist IDs\n')
        return self.artist_ids

    def get_top_tracks(self):
        print("Getting Top Tracks")
        for index, artist_id in enumerate(self.artist_ids):
            endpoint = f'https://api.spotify.com/v1/artists/{artist_id}/top-tracks?
market={self.country_code}'
            r = requests.get(endpoint, headers=self.headers)
            for d in r.json()['tracks']:
                d['album_name'] = d['album']['name']
                d['album_release_date'] = d['album']['release_date']
                d['album_type'] = d['album']['album_type']
                d['duration_sec'] = round(d['duration_ms'] / 1000, 1)
                d['explicit'] = str(d['explicit'])
                to_del = map(str,
                             'type track_number artists album duration_ms disc_number
external_ids external_urls uri is_local is_playable href preview_url'.split())
                for x in to_del:
                    del d[x]
                d['artist'] = self.artists[index]
                self.track_ids.append(d['id'])
                self.top_tracks.append(d)
        print("Got Top Tracks\n")
        return self.top_tracks

    def get_top_track_features(self):
        print("Getting Features for Top Tracks")
        counter = 0
        print('[1]')
        for index, id_ in enumerate(self.track_ids):
            if counter == 60:
                time.sleep(10)
                counter = -20
            endpoint = f"https://api.spotify.com/v1/audio-features/{id_}"
            r = requests.get(endpoint, headers=self.headers)
            d = r.json()
            try:
                to_del = map(str, 'analysis_url uri duration_ms track_href type'.split())
                for x in to_del:
                    del d[x]
            except Exception as e:
```

```
                print(str(e))

            counter += 1
            self.top_track_features.append(d)

            if (index + 1) % 50 == 0:
                print(f'[{index + 1}]')
        print("Got Features for Top Tracks\n")
        return self.top_track_features

    def get_combined(self):
        top_track_features_copy = self.top_track_features.copy()
        top_track_details = self.top_tracks.copy()
        for i in range(len(top_track_details)):
            top_track_details[i].update(top_track_features_copy[i])
        self.top_track_details = top_track_details
        return self.top_track_details

    def get_df(self):
        self.df = DF(self.top_track_details)
        print("DataFrame Created\n")
        return self.df
```

# Create.py

```
from Base_SpotifyAPI_Class import SpotifyAPI
from FeaturesSpotifyAPI_Class import *
import requests
import base64
import datetime as dt
from urllib.parse import urlencode
from pprint import pprint as pp
import pandas as pd
from pandas import DataFrame as DF
from pandas import Series as S
import numpy as np
import json
import time
from matplotlib import pyplot as py
from matplotlib.ticker import MultipleLocator, AutoMinorLocator, ScalarFormatter,
NullFormatter, FormatStrFormatter
from functools import reduce
import operator

client_id = '602ac06174344597a70ddd9949e97f50'
client_secret = '286a7572b3e944ff9708690eaaa4b8f2'

artists = []
artists += list(map(str, 'The Weeknd,Kanye West,Drake,Joyner Lucas,Eminem,J. Cole,Kendrick
Lamar,Logic,Meek Mill,50 Cent,Rihanna,Nicki Minaj,Cardi B,Travis Scott,Juice WRLD,Post
Malone,Machine Gun Kelly,Closed on Sunday,Powfu,Kid Cudi,A$AP Rocky,Dr. Dre,2pac,The Notorious
B.I.G.,JAY-Z,Pusha-T,Joey Bada$$,Hopsin,XXXTENTACION'.split(',')))
artists += list(map(str, 'Queen,The Beatles,Michael Jackson,Glass Animals,Taylor Swift,Imagine
Dragons,Coldplay,Halsey,X Ambassadors,Hozier,The Chainsmokers,Avicii,Dj Snake,Martin
Garrix,Skrillex,20syl,Kygo'.split(',')))
artists += list(map(str, 'Ed Sheeran,Bastille,Billie Eilish,Duncan Laurence,Dempsey
Hope,Christian French,Miley Cyrus,Frank Ocean,Ellie Goulding,Camila Cabello,Alec
Benjamin,Bruno Mars,Lauv,Selena Gomez,Maroon 5,Jon Bellion,JP Saxe,Ariana Grande,Justin
Bieber,Anson Seabra,Justin Timberlake,Khalid,6LACK,Dean Lewis'.split(',')))
artists += list(map(str, 'One Direction,Katy Perry,Lady Gaga,Beyonce,Shawn Mendes,Sam
Smith,Mac Miller,Linkin Park,Britney Spears,Alicia Keys,Black Eyed Peas,Green Day,Denzel
Curry,AURORA,SZA,Lukas Graham'.split(',')))
artists += list(map(str, 'Bon Jovi,Sting,Backstreet Boys,Whitney Houston,David Bowie,AC/
DC,Elvis Presley,Elton John,John Lennon,Bob Marley,ABBA,Paul McCartney,Adele,Dua
Lipa'.split(',')))

def main_():
    spotify = FeaturesSpotifyAPI(client_id, client_secret, artists)
    spotify.auth()
    spotify.get_artist_ids()
    spotify.get_top_tracks()
    spotify.get_top_track_features()
    spotify.get_combined()
    df = spotify.get_df()
    df.to_csv('music_data.csv')
```

```
if __name__ == '__main__':
    main_()

df = pd.read_csv('music_data.csv')
mean_df = df.groupby("artist").mean()
mean_df.drop('Unnamed: 0', 1, inplace=True)
mean_df.to_csv('music_data_by_artist.csv')
```

## Screenshot of music_data.csv (opened in Excel/Numbers)

music_data

| | name | popularity | album_name | album_release_date | album_type | duration_sec | artist | danceability | energy | key | loudness | mode | speechiness | acousticness | instrumentalness | liveness | valence | tempo | time_signature |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Blinding Lights | 94 | After Hours | 2020-03-20 | album | 200.0 | The Weeknd | 0.514 | 0.73 | 1 | -5.934 | 1 | 0.0598 | 0.00146 | 9.54E-05 | 0.0897 | 0.334 | 171.005 | 4 |
| J | Sacrifice | 92 | Dawn FM | 2022-01-06 | album | 188.9 | The Weeknd | 0.735 | 0.795 | 11 | -6.523 | 0 | 0.113 | 0.0296 | 3.18E-05 | 0.0678 | 0.905 | 122.0 | 4 |
| ' | Save Your Tears (with Ariana Grande) (Remix) | 92 | Save Your Tears (Remix) | 2021-04-23 | single | 191.0 | The Weeknd | 0.65 | 0.825 | 0 | -4.645 | 1 | 0.0325 | 0.0215 | 2.44E-05 | 0.0936 | 0.593 | 118.091 | 4 |
| | One Right Now (with The Weeknd) | 92 | One Right Now | 2021-11-05 | single | 193.5 | The Weeknd | 0.687 | 0.781 | 1 | -4.806 | 1 | 0.053 | 0.0361 | 0.0 | 0.0755 | 0.688 | 97.014 | 4 |
| | Moth To A Flame (with The Weeknd) | 92 | Moth To A Flame | 2021-10-22 | single | 234.0 | The Weeknd | 0.542 | 0.659 | 8 | -7.289 | 1 | 0.0389 | 0.00279 | 0.0 | 0.105 | 0.109 | 120.122 | 4 |
| g | Save Your Tears | 90 | After Hours | 2020-03-20 | album | 215.6 | The Weeknd | 0.68 | 0.826 | 0 | -5.487 | 1 | 0.0309 | 0.0212 | 1.24E-05 | 0.543 | 0.644 | 118.051 | 4 |
| | You Right | 89 | Planet Her | 2021-06-25 | album | 186.2 | The Weeknd | 0.828 | 0.621 | 8 | -6.414 | 1 | 0.0565 | 0.0164 | 0.00233 | 0.0845 | 0.436 | 128.986 | 4 |
| ; | Lost in the Fire (feat. The Weeknd) | 89 | Hyperion | 2019-03-08 | album | 202.1 | The Weeknd | 0.658 | 0.671 | 2 | -12.21 | 1 | 0.0363 | 0.0933 | 0.000927 | 0.115 | 0.166 | 100.966 | 4 |
| IK | LA FAMA (with The Weeknd) | 88 | LA FAMA (with The Weeknd) | 2021-11-11 | single | 188.1 | The Weeknd | 0.77 | 0.302 | 0 | -7.911 | 0 | 0.0466 | 0.946 | 3.38E-05 | 0.125 | 0.821 | 136.01 | 4 |
| B | The Hills | 87 | Beauty Behind The Madness | 2015-08-28 | album | 242.3 | The Weeknd | 0.585 | 0.564 | 0 | -7.063 | 0 | 0.0515 | 0.0671 | 0.0 | 0.135 | 0.137 | 113.003 | 4 |
| ' | Praise God | 88 | Donda | 2021-08-29 | album | 226.7 | Kanye West | 0.798 | 0.545 | 1 | -6.466 | 1 | 0.168 | 0.00904 | 9.48E-05 | 0.258 | 0.212 | 118.029 | 4 |
| : | City of Gods | 86 | City of Gods | 2022-02-11 | single | 256.0 | Kanye West | 0.474 | 0.801 | 8 | -5.978 | 0 | 0.375 | 0.102 | 0.0 | 0.322 | 0.497 | 147.356 | 4 |
| 7 | Ni**as In Paris | 86 | Watch The Throne (Deluxe) | 2011-08-08 | album | 219.3 | Kanye West | 0.789 | 0.858 | 1 | -5.542 | 1 | 0.311 | 0.127 | 0.0 | 0.349 | 0.775 | 140.022 | 4 |
| A5 | Heartless | 85 | 808s & Heartbreak | 2008-11-24 | album | 211.0 | Kanye West | 0.79 | 0.647 | 10 | -5.983 | 0 | 0.136 | 0.0515 | 0.0 | 0.248 | 0.654 | 87.999 | 4 |
| l | Eazy | 85 | Eazy | 2022-01-15 | single | 234.3 | Kanye West | 0.716 | 0.551 | 1 | -6.987 | 1 | 0.47 | 0.167 | 0.0 | 0.197 | 0.305 | 82.393 | 4 |
| | Bound 2 | 84 | Yeezus | 2013-06-18 | album | 229.1 | Kanye West | 0.367 | 0.665 | 1 | -2.821 | 1 | 0.0465 | 0.145 | 0.0 | 0.113 | 0.31 | 148.913 | 4 |
| | Hurricane | 83 | Donda | 2021-08-29 | album | 243.2 | Kanye West | 0.587 | 0.561 | 0 | -6.569 | 1 | 0.146 | 0.0466 | 0.0 | 0.109 | 0.351 | 80.009 | 4 |
| | Father Stretch My Hands Pt. 1 | 83 | The Life Of Pablo | 2016-06-10 | album | 135.9 | Kanye West | 0.724 | 0.573 | 7 | -4.113 | 1 | 0.0549 | 0.118 | 0.0 | 0.538 | 0.438 | 113.088 | 4 |
| n | Moon | 82 | Donda | 2021-08-29 | album | 156.1 | Kanye West | 0.319 | 0.236 | 5 | -9.644 | 1 | 0.0318 | 0.898 | 1.62E-05 | 0.272 | 0.254 | 162.351 | 4 |
| c | POWER | 82 | My Beautiful Dark Twisted Fantasy | 2010-11-22 | album | 292.1 | Kanye West | 0.542 | 0.914 | 0 | -4.747 | 0 | 0.113 | 0.0161 | 0.0 | 0.744 | 0.576 | 153.993 | 4 |
| q6 | Knife Talk (with 21 Savage ft. Project Pat) | 89 | Certified Lover Boy | 2021-09-03 | album | 243.0 | Drake | 0.849 | 0.424 | 5 | -9.579 | 0 | 0.324 | 0.0635 | 0.0 | 0.0834 | 0.153 | 145.887 | 4 |
| G | Way 2 Sexy (with Future & Young Thug) | 88 | Certified Lover Boy | 2021-09-03 | album | 257.6 | Drake | 0.803 | 0.597 | 11 | -6.035 | 0 | 0.141 | 0.000619 | 4.5E-06 | 0.323 | 0.331 | 136.008 | 4 |
| | Fair Trade (with Travis Scott) | 88 | Certified Lover Boy | 2021-09-03 | album | 291.2 | Drake | 0.666 | 0.465 | 1 | -8.545 | 1 | 0.26 | 0.0503 | 0.0 | 0.215 | 0.292 | 167.937 | 4 |
| | One Dance | 87 | Views | 2016-05-06 | album | 174.0 | Drake | 0.792 | 0.625 | 1 | -5.609 | 1 | 0.0536 | 0.00776 | 0.0018 | 0.329 | 0.37 | 103.967 | 4 |
| '5b | Wants and Needs (feat. Lil Baby) | 86 | Scary Hours 2 | 2021-03-05 | single | 193.0 | Drake | 0.578 | 0.449 | 1 | -6.349 | 1 | 0.286 | 0.0618 | 2.17E-06 | 0.119 | 0.1 | 136.006 | 4 |

# Data Analysis

The datapoints extracted have been defined and explained:

1. **release_date**: string
   The date the album was first released.

2. **popularity**: integer
   The popularity of the track.
   The popularity of a track is a value between 0 and 100, with 100 being the most popular.
   The popularity is calculated by algorithm and is based, in the most part, on the total number of plays the track has had and how recent those plays are.
   Generally speaking, songs that are being played a lot now will have a higher popularity than songs that were played a lot in the past. Duplicate tracks (e.g. the same track from a single and an album) are rated independently. Artist and album popularity is derived mathematically from track popularity. **Note**: the popularity value may lag actual popularity by a few days: the value is not updated in real time.

3. **duration**: integer
   The track length in seconds

4. **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.

5. **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

6. **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 <= key <= 11

7. **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.

8. **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.

9. **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.

10. **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.
    Acoustic music is music that solely or primarily uses instruments that produce sound through acoustic means, as opposed to electric or electronic means.

11. **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.

12. **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.

13. **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

14. **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.

** The datapoint has been multiplied by 100 for increased readability. The value now ranges between 0 and 100.*

# Main.py

```
import create
from create import *
from difflib import get_close_matches
from PIL import Image as img
import danceability

df = create.df
mean_df = create.mean_df
main_menu = """Main Menu:
1. DataFrame              2. Search for Artists in DataFrame
3. Graph for Mode         4. Graph for Release Date
5. Graph for Key          6. Popularity Menu
7. Duration Menu          8. Danceability Menu
9. Energy Menu            10. Loudness Menu
11. Valence Menu          12. Speechiness Menu
13. Acousticness Menu     14. Instrumentalness Menu
15. Liveness Menu         16. Tempo Menu
To exit any menu press 'q' or 'Q' """

popularity_menu = "\nPopularity Menu:\n1. Show Track Popularity Distribution\t\t2. Show Artist Popularity
Distribution\n3. Show Artists with Highest Popularity\t\t4. Show Songs with Highest Popularity"
duration_menu = "\nDuration Menu:\n1. Show Duration Distribution Histogram\t\t2. Show Artists with
Highest Average Duration\n"
loudness_menu = "\nLoudness Menu:\n1. Show Loudness Distribution Histogram\t\t2. Show Loudest Songs\n3.
Show Loudest Artists\n"

def menu(dpnt: str): return f"\n{dpnt} Menu:\n1. Show {dpnt} Distribution Histogram\t\t2. Show Songs with
the Highest {dpnt}\n3. Show Artists with the Highest {dpnt}\n>>>"

def show_df(): pp(df.loc[:, 'name':])

def show_artist():
    inp_ = 0
    while inp_ not in ['q', 'Q']:
        artists_ = df.artist.unique()
        inp_ = input("Enter Artist Name or press 'i' to list of Artists: ")
        if inp_ in ['q', 'Q']: break
        if inp_ in ['i', 'I']:
            pp(", ".join(create.artists))
            inp_ = input("Enter Artist Name: ")
        if inp_ in artists_: pp(df[df['artist'] == inp_.lower()].loc[:, 'name':])
        else:
            close_matches = get_close_matches(inp_, artists_, 3, 0.1)
            if len(close_matches):
                inp__ = input(f"Did you mean any of {close_matches}?\n(Enter no or 1, 2, or 3 to select)
\n>>>")
                try:
                    inp_ = close_matches[int(inp__)-1]
                    pp(df[df['artist'] == inp_].loc[:, 'name':])
                except ValueError or IndexError: print("Artist data unavailable")
                except Exception as e_: print(e_)
            else: print("Artist data unavailable")
def show_mode():
    df_sorted_mode = df.sort_values(by=['mode'], ascending=True)
    py.figure(dpi=300)
    x = range(0, 2)
    mode_counts = [list(df_sorted_mode['mode']).count(i) for i in x]
    py.pie(mode_counts, labels=x, autopct='%1.0f%%')
    py.legend(title='Mode Value:')
    py.title("Distribution of Modes")
    py.savefig('mode_pie')
    py.close()
    print("Graph saved in enclosing folder and opened in new window")
    img.open('mode_pie.png').show()
def show_release_date():
    df_album_release_date_sorted = df.sort_values(by=['album_release_date'])
    py.figure(dpi=300)
    py.hist(df_album_release_date_sorted.album_release_date.str.slice(0, 4).astype(int),
            bins=[x for x in range(1955, 2030, 5)], rwidth=0.9, color='#997BFF')
    py.xticks(range(1955, 2030, 5), rotation=25)
    py.ylabel('Number of Songs')
    py.xlabel('Year of Release')
    py.grid()
    py.savefig('release_date_hist_year')
    py.close()
    print("Graph saved in enclosing folder and opened in new window")
    img.open('release_date_hist_year.png').show()

def show_key():
    df_sorted_key = df.sort_values(by=['key'], ascending=True)
    py.figure(dpi=450)
```

```
    x = range(-1, 12)
    key_counts = [list(df_sorted_key.key).count(i) for i in x]
    py.bar(x, key_counts, color='#8F4A45')
    py.plot(x, key_counts, linewidth=0.75, marker='.', color='#8F8F8F')
    py.axis(xmax=11.5, xmin=-1.5)
    ax = py.gca()
    ax.yaxis.set_ticks_position('both')
    ax.yaxis.set_minor_locator(AutoMinorLocator())
    py.grid(axis='y')
    py.xticks(x)
    py.title("Distribution of Keys")
    py.xlabel("Key")
    py.ylabel("Number of Tracks")
    py.savefig('key_bar')
    py.close()
    print("Graph saved in enclosing folder and opened in new window")
    img.open('key_bar.png').show()
def show_duration():
    inp_ = 0
    df_sorted_duration = df.sort_values(by=['duration_sec'], ascending=False)
    mean_df_sorted_duration = mean_df.sort_values(by=['duration_sec'], ascending=False)
    while inp_ not in ['Q', 'q']:
        inp_ = input(duration_menu)
        if inp_ in ['Q', 'q']: break
        if inp_ == '1':
            py.figure(dpi=300)
            py.xticks(range(60, 660, 30), fontsize=8)
            py.yticks(range(0, 325, 25))
            py.hist(df_sorted_duration.duration_sec, bins=range(60, 660, 30), rwidth=0.8,
color='#FF8A82')
            py.axis(xmin=50, xmax=640, ymax=300)
            py.gca().yaxis.set_minor_locator(AutoMinorLocator())
            py.xlabel("Duration in Seconds")
            py.ylabel("No. of Songs")
            py.title("Distribution of Duration")
            py.grid(which='major', color='#777777')
            py.grid(which='minor', color='#ACACAC', linestyle='dashed', linewidth=0.3)
            py.savefig('duration')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('duration_hist.png').show()
        if inp_ == '2':
            x = range(5, 55, 5)
            py.figure(dpi=300)
            py.bar(x, height=mean_df_sorted_duration.duration_sec.iloc[0:10], width=2, color="#3CBCFF")
            py.plot(x, mean_df_sorted_duration.duration_sec.iloc[0:10], color='#3CBCFF', marker=".",
markersize=10)
            py.axis(ymin=260, ymax=320)
            ax = py.gca()
            ax.yaxis.set_minor_locator(AutoMinorLocator())
            py.grid()
            py.title("Artists with Highest Average Duration")
            py.ylabel("Duration (in s)")
            ticks = [x if x != "The Notorious B.I.G." else "Biggie" for x in
                     mean_df_sorted_duration.duration_sec.iloc[0:10].index]
            py.xticks(x, ticks, fontsize=6, rotation=-15)
            py.savefig('duration_bar_avg')
            print("Graph saved in enclosing folder and opened in new window")
            img.open('duration_bar_artists_top10.png').show()
        else: print("Invalid Input")
def show_tempo():
    inp_ = 0
    df_sorted_tempo = df.sort_values(by=['tempo'], ascending=False)
    while inp_ not in ['Q', 'q']:
        inp_ = input(menu("Tempo"))
        if inp_ in ['Q', 'q']: break
        if inp_ == '1':
            py.figure(dpi=450)
            py.hist(df_sorted_tempo.tempo, bins=range(60, 220, 10), rwidth=0.7, color='#6EB972')
            py.axis(xmin=60, xmax=210, ymax=150)
            ax = py.gca()
            py.xticks(range(60, 220, 10), fontsize=8)
            py.yticks(range(0, 160, 10))
            ax.yaxis.set_ticks_position('both')
            ax.xaxis.set_ticks_position('both')
            ax.yaxis.set_minor_locator(AutoMinorLocator())
            py.grid()
            py.title("Tempo Distribution")
            py.xlabel("Tempo")
            py.ylabel("Number of Tracks")
            py.savefig('tempo_hist')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('tempo_hist.png').show()
```

```
        elif inp_ == '2':
            py.figure(dpi=450, figsize=(9, 6))
            data = pd.Series(df_sorted_tempo.tempo.unique())
            names = pd.Series(df_sorted_tempo.name.unique())
            py.bar(x1, data.iloc[:10], width=5, color='#599BA7')
            names = list(names.iloc[:10])
            py.axis(ymin=185, ymax=210)
            names_new = []
            for name in names:
                words = name.split(" ")
                names_new.append([" ".join(words[i:i + 1]) for i in range(0, len(words), 1)])
            for name in names_new: names_new[names_new.index(name)] = str.join("\n", name)
            py.xticks(x1, names_new, fontsize=6.5)
            py.title("Songs with the Highest Tempo")
            py.ylabel("Tempo")
            py.grid(axis='y')
            ax = py.gca()
            ax.yaxis.set_ticks_position('both')
            ax.yaxis.set_minor_locator(AutoMinorLocator())
            py.savefig('tempo_bar_top10')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('tempo_bar_top10.png').show()
        elif inp_ == '3':
            py.figure(dpi=450, figsize=(9, 6))
            data = pd.Series(df_sorted_tempo.tempo.unique())
            names = pd.Series(df_sorted_tempo.name.unique())
            py.bar(x1, data.iloc[-10:][::-1], width=5, color='#6BA708')
            names = list(names.iloc[-10:])[::-1]
            py.axis(ymin=60, ymax=75)
            names_new = []
            for name in names:
                words = name.split(" ")
                names_new.append([" ".join(words[i:i + 1]) for i in range(0, len(words), 1)])
            for name in names_new:
                names_new[names_new.index(name)] = str.join("\n", name)
            py.xticks(x1, names_new, fontsize=6.5)
            py.title("Songs with the Lowest Tempo")
            py.ylabel("Tempo")
            py.grid(axis='y')
            ax = py.gca()
            ax.yaxis.set_ticks_position('both')
            ax.yaxis.set_minor_locator(AutoMinorLocator())
            py.savefig('tempo_bar_bottom10')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('tempo_bar_bottom10.png').show()
        else: print("Invalid Input")
def show_danceability():
    inp_ = 0
    while inp_ not in ['Q', 'q']:
        inp_ = input(menu("Danceability"))
        if inp_ in ['Q', 'q']: break
        elif inp_ == '1':
            danceability.danceability_hist()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('danceability_hist.png').show()
        elif inp_ == '2':
            danceability.danceability_bar_top10()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('danceability_bar_top10.png').show()
        elif inp_ == '3':
            danceability.danceability_bar_artists_top10()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('danceability_bar_artists_top10.png').show()
        else: print("Invalid Input")
def show_energy():
    inp_ = 0
    df_sorted_energy = df.sort_values(by=['energy'], ascending=False)
    mean_df_sorted_energy = mean_df.sort_values(by=['energy'], ascending=False)
    while inp_ not in ['Q', 'q']:
        inp_ = input((menu("Energy"))
        if inp_ in ['Q', 'q']: break
        elif inp_ == '1':
            py.figure(dpi=450)
            py.hist(df_sorted_energy.energy, bins=pd.Series(range(0, 21))/20)
            py.yticks(range(0, 130, 10))
            ax = py.gca()
            ax.yaxis.set_minor_locator(AutoMinorLocator())
            py.grid()
            py.axis(xmin=0, xmax=1, ymax=120)
            py.title("Distribution of Energy")
            py.xlabel("Energy")
            py.ylabel("Number of Tracks")
```

```
            py.xticks(pd.Series(range(0, 21))/20, fontsize=7)
            py.savefig('energy_hist')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('energy_hist.png').show()
        elif inp_ == '2':
            py.figure(dpi=450, figsize=(10, 5))
            py.bar(x1, df_sorted_energy.energy.iloc[0:10], width=5, color='#FFE695')
            py.axis(ymin=0.95, ymax=1)
            names = list(df_sorted_energy.name.iloc[0:10])
            names_new = []
            for name in names:
                words = name.split(" ")
                names_new.append([" ".join(words[i:i+2]) for i in range(0, len(words), 2)])
            for name in names_new:
                names_new[names_new.index(name)] = str.join("\n", name)
            py.xticks(x1, names_new, fontsize=6.5)
            py.title("Most Energetic Songs")
            py.ylabel("Energy")
            py.grid(axis='y')
            py.savefig('energy_bar_top10')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('energy_bar_top10.png').show()
        elif inp_ == '3':
            py.figure(dpi=450, figsize=(7, 4))
            py.bar(x1, mean_df_sorted_energy.energy.iloc[0:10], width=5, color='#FF005D')
            artists_ = mean_df_sorted_energy.iloc[0:10, 1:2].index
            artists_new = []
            for artist in artists_:
                words = artist.split(" ")
                artists_new.append([" ".join(words[i:i + 1]) for i in range(0, len(words), 1)])
            for artist in artists_new:
                artists_new[artists_new.index(artist)] = str.join("\n", artist)
            py.xticks(x1, artists_new, fontsize=7)
            py.title("Most Energetic Artists")
            py.ylabel("Energy")
            py.axis(ymin=0.75, ymax=0.9)
            py.grid(axis='y')
            py.gca().yaxis.set_minor_locator(AutoMinorLocator())
            py.gca().yaxis.set_ticks_position('both')
            py.savefig('energy_bar_artists_top10')
            print("Graph saved in enclosing folder and opened in new window")
            img.open('energy_bar_artists_top10.png').show()
        else: print("Invalid Input")
    def show_loudness():
        inp_ = 0
        df_sorted_loudness = df.sort_values(by=['loudness'], ascending=False)
        mean_df_sorted_loudness = mean_df.sort_values(by=['loudness'], ascending=False)
        while inp_ not in ['Q', 'q']:
            inp_ = input(loudness_menu)
            if inp_ in ['Q', 'q']: break
            elif inp_ == '1':
                py.figure(dpi=450)
                py.hist(df_sorted_loudness.loudness, bins=range(-23, 0, 1), rwidth=0.7, color='#51A063')
                py.axis(xmax=0, xmin=-23)
                ax = py.gca()
                ax.yaxis.tick_right()
                ax.yaxis.set_ticks_position('both')
                ax.yaxis.set_minor_locator(AutoMinorLocator())
                ax.xaxis.set_minor_locator(AutoMinorLocator())
                py.grid()
                py.title("Distribution of Loudness")
                py.xlabel("Loudness")
                py.ylabel("Number of Tracks")
                py.savefig('loudness_hist')
                py.close()
                print("Graph saved in enclosing folder and opened in new window")
                img.open('loudness_hist.png').show()
            elif inp_ == '2':
                py.figure(dpi=450, figsize=(9, 5))
                py.bar(x1, df_sorted_loudness.loudness.iloc[0:10], width=5, )
                names = list(df_sorted_loudness.name.iloc[0:10])
                py.yticks(np.array(range(-10, 1)) / 4)
                py.axis(ymax=0, ymin=-2.5)
                names_new = []
                for name in names:
                    words = name.split(" ")
                    names_new.append([" ".join(words[i:i + 2]) for i in range(0, len(words), 2)])
                for name in names_new:
                    names_new[names_new.index(name)] = str.join("\n", name)
                py.xticks(x1, names_new, fontsize=6.5)
                py.title("Loudest Songs")
                py.ylabel("Loudness")
```

```
            py.grid(axis='y')
            py.savefig('loudness_bar_top10')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('loudness_bar_top10.png').show()
        elif inp_ == '3':
            py.figure(dpi=450, figsize=(8, 5))
            py.bar(x1, mean_df_sorted_loudness.loudness.iloc[0:10], width=5, color='#FF005D')
            artists_ = mean_df_sorted_loudness.iloc[0:10, 1:2].index
            py.xticks(x1, artists_, fontsize=6.5)
            py.title("Loudest Artists")
            py.ylabel("Loudness")
            py.yticks(np.array(range(-20, -13)) / 4)
            py.grid(axis='y')
            py.axis(ymin=-4.75, ymax=-3.5)
            ax = py.gca()
            ax.yaxis.set_minor_locator(AutoMinorLocator())
            ax.yaxis.set_ticks_position('both')
            py.savefig('loudness_bar_artists_top10')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('loudness_bar_artists_top10.png').show()
        else: print("Invalid Input")
def show_valence():
    inp_ = 0
    df_sorted_valence = df.sort_values(by=['valence'], ascending=False)
    mean_df_sorted_valence = mean_df.sort_values(by=['valence'], ascending=False)
    df_sorted_valence.valence *= 100
    mean_df_sorted_valence *= 100
    while inp_ not in ['Q', 'q']:
        inp_ = input(menu("Valence"))
        if inp_ in ['Q', 'q']: break
        elif inp_ == '1':
            py.figure(dpi=450)
            py.hist(df_sorted_valence.valence, bins=range(0, 105, 5), rwidth=0.7, color='#C86A72')
            py.axis(xmin=-1, xmax=101, ymax=90)
            ax = py.gca()
            py.xticks(range(0, 105, 5), fontsize=8)
            ax.yaxis.set_ticks_position('both')
            ax.yaxis.set_minor_locator(AutoMinorLocator())
            py.grid()
            py.title("Distribution of Valence")
            py.xlabel("Valence (out of 100)")
            py.ylabel("Number of Tracks")
            py.savefig('valence_hist')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('valence_hist.png').show()
        elif inp_ == '2':
            py.figure(dpi=450)
            py.bar(x1, df_sorted_valence.valence.iloc[0:10], width=5, color='#C80045')
            names = list(df_sorted_valence.name.iloc[0:10])
            py.axis(ymin=96, ymax=97.5)
            py.yticks([96, 96.25, 96.5, 96.75, 97, 97.25, 97.5])
            names_new = []
            for name in names:
                words = name.split(" ")
                names_new.append([" ".join(words[i:i + 1]) for i in range(0, len(words), 1)])
            for name in names_new:
                names_new[names_new.index(name)] = str.join("\n", name)
            py.xticks(x1, names_new, fontsize=6)
            py.title("Songs with the Highest Valence")
            py.ylabel("Valence (out of 100)")
            py.grid(axis='y')
            ax = py.gca()
            ax.yaxis.set_ticks_position('both')
            ax.yaxis.set_minor_locator(AutoMinorLocator())
            py.savefig('valence_bar_top10')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('valence_bar_top10.png').show()
        elif inp_ == '3':
            py.figure(dpi=450)
            py.bar(x1, mean_df_sorted_valence.valence.iloc[0:10], width=5, color='#00A995')
            artists_ = list(mean_df_sorted_valence.iloc[0:10, 1:2].index)
            try: artists_[artists_.index("Joey Bada$$")] = "Joey Badass"
            except ValueError: pass
            except Exception as e: print(e)
            artists_new = []
            for artist in artists_:
                words = artist.split(" ")
                artists_new.append([" ".join(words[i:i + 1]) for i in range(0, len(words), 1)])
            for artist in artists_new: artists_new[artists_new.index(artist)] = str.join("\n", artist)
            py.xticks(x1, artists_new, fontsize=7)
```

```
            py.title("Artists with the Highest Valence")
            py.ylabel("Valence (out of 100)")
            py.axis(ymin=65, ymax=77)
            py.yticks(range(65, 78))
            py.grid(axis='y', which='major')
            ax = py.gca()
            ax.yaxis.set_minor_locator(AutoMinorLocator())
            ax.yaxis.set_ticks_position('both')
            py.savefig('valence_bar_artists_top10')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('valence_bar_artists_top10.png').show()
        else: print("Invalid Input")
def show_speechiness():
    inp_ = 0
    df_sorted_speechiness = df.sort_values(by=['speechiness'], ascending=False)
    mean_df_sorted_speechiness = mean_df.sort_values(by=['speechiness'], ascending=False)
    df_sorted_speechiness.speechiness *= 100
    mean_df_sorted_speechiness *= 100
    while inp_ not in ['Q', 'q']:
        inp_ = input(menu("Speechiness"))
        if inp_ in ['Q', 'q']: break
        elif inp_ == '1':
            py.figure(dpi=450)
            py.hist(df_sorted_speechiness.speechiness, bins=range(0, 56, 2), rwidth=0.7, color='#51A063')
            py.axis(xmin=0, xmax=55, ymax=275)
            ax = py.gca()
            py.xticks(range(0, 56, 2), fontsize=8)
            py.yticks(range(0, 300, 25))
            ax.yaxis.set_ticks_position('both')
            ax.yaxis.set_minor_locator(AutoMinorLocator())
            py.grid()
            py.title("Distribution of Speechiness")
            py.xlabel("Speechiness (out of 100)")
            py.ylabel("Number of Tracks")
            py.savefig('speechiness_hist')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('speechiness_hist.png').show()
        elif inp_ == '2':
            py.figure(dpi=450, figsize=(7, 5))
            py.bar(x1, df_sorted_speechiness.speechiness.iloc[0:10], width=5, color='#5928A9')
            names = list(df_sorted_speechiness.name.iloc[0:10])
            py.axis(ymin=40, ymax=55)
            py.yticks(range(40, 56))
            names_new = []
            for name in names:
                words = name.split(" ")
                names_new.append([" ".join(words[i:i + 2]) for i in range(0, len(words), 2)])
            for name in names_new:
                names_new[names_new.index(name)] = str.join("\n", name)
            py.xticks(x1, names_new, fontsize=6)
            py.title("Songs with the Highest Speechiness")
            py.ylabel("Speechiness (out of 100)")
            py.grid(axis='y')
            ax = py.gca()
            ax.yaxis.set_ticks_position('both')
            ax.yaxis.set_minor_locator(AutoMinorLocator())
            py.savefig('speechiness_bar_top10')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('speechiness_bar_top10.png').show()
        elif inp_ == '3':
            py.figure(dpi=450)
            py.bar(x1, mean_df_sorted_speechiness.speechiness.iloc[0:10], width=5, color='#00A995')
            artists_ = list(mean_df_sorted_speechiness.iloc[0:10, 1:2].index)
            try: artists_[artists_.index("Joey Bada$$")] = "Joey Badass"
            except ValueError: pass
            except Exception as e: print(e)
            artists_new = []
            for artist in artists_:
                words = artist.split(" ")
                artists_new.append([" ".join(words[i:i + 1]) for i in range(0, len(words), 1)])
            for artist in artists_new: artists_new[artists_new.index(artist)] = str.join("\n", artist)
            py.xticks(x1, artists_new, fontsize=7)
            py.yticks(range(20, 33))
            py.title("Artists with the Highest Speechiness")
            py.ylabel("Speechiness (out of 100)")
            py.axis(ymin=20, ymax=32)
            py.grid(axis='y')
            ax = py.gca()
            ax.yaxis.set_minor_locator(AutoMinorLocator())
            ax.yaxis.set_ticks_position('both')
            py.savefig('speechiness_bar_artists_top10')
```

```
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('speechiness_bar_artists_top10.png').show()
        else: print("Invalid Input")
def show_acousticness():
    inp_ = 0
    df_sorted_acousticness = df.sort_values(by=['acousticness'], ascending=False)
    mean_df_sorted_acousticness = mean_df.sort_values(by=['acousticness'], ascending=False)
    df_sorted_acousticness.acousticness *= 100
    mean_df_sorted_acousticness *= 100
    while inp_ not in ['Q', 'q']:
        inp_ = input(menu("Acousticness"))
        if inp_ in ['Q', 'q']: break
        elif inp_ == '1':
            py.figure(dpi=450)
            py.hist(df_sorted_acousticness.acousticness, bins=range(0, 105, 5), rwidth=0.7,
color='#A90036')
            py.axis(xmin=-1, xmax=101, ymax=300)
            ax = py.gca()
            py.xticks(range(0, 105, 5), fontsize=8)
            py.yticks(range(0, 325, 25))
            ax.yaxis.set_ticks_position('both')
            ax.xaxis.set_ticks_position('both')
            ax.yaxis.set_minor_locator(AutoMinorLocator())
            py.grid()
            py.title("Distribution of Acousticness")
            py.xlabel("Acousticness (out of 100)")
            py.ylabel("Number of Tracks")
            py.savefig('acousticness_hist')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('acousticness_hist.png').show()
        elif inp_ == '2':
            py.figure(dpi=450, figsize=(7, 5))
            py.bar(x1, df_sorted_acousticness.acousticness.iloc[0:10], width=5, color='#CB522E')
            names = list(df_sorted_acousticness.name.iloc[0:10])
            py.axis(ymin=95, ymax=98)
            py.yticks(np.array(range(190, 197))/2)
            names_new = []
            for name in names:
                words = name.split(" ")
                names_new.append([" ".join(words[i:i + 2]) for i in range(0, len(words), 2)])
            for name in names_new:
                names_new[names_new.index(name)] = str.join("\n", name)
            py.xticks(x1, names_new, fontsize=6)
            py.title("Songs with the Highest Acousticness")
            py.ylabel("Acousticness (out of 100)")
            py.grid(axis='y')
            ax = py.gca()
            ax.yaxis.set_ticks_position('both')
            ax.yaxis.set_minor_locator(AutoMinorLocator())
            py.savefig('acousticness_bar_top10')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('acousticness_bar_top10.png').show()
        elif inp_ == '3':
            py.figure(dpi=450)
            py.bar(x1, mean_df_sorted_acousticness.acousticness.iloc[0:10], width=5, color='#00A995')
            artists = list(mean_df_sorted_acousticness.iloc[0:10, 1:2].index)
            try: artists[artists.index("Joey Bada$$")] = "Joey Badass"
            except ValueError:
                pass
            except Exception as e:
                print(e)
            artists_new = []
            for artist in artists:
                words = artist.split(" ")
                artists_new.append([" ".join(words[i:i + 1]) for i in range(0, len(words), 1)])
            for artist in artists_new:
                artists_new[artists_new.index(artist)] = str.join("\n", artist)
            py.xticks(x1, artists_new, fontsize=7)
            py.title("Artists with the Highest Acousticness")
            py.ylabel("Acousticness (out of 100)")
            py.axis(ymin=50, ymax=95)
            py.grid(axis='y')
            ax = py.gca()
            ax.yaxis.set_minor_locator(AutoMinorLocator())
            ax.yaxis.set_ticks_position('both')
            py.savefig('acousticness_bar_artists_top10')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('acousticness_bar_artists_top10.png').show()
        else: print("Invalid Input")
```

```
def show_instrumentalness():
    inp_ = 0
    df_sorted_instrumentalness = df.sort_values(by=['instrumentalness'], ascending=False)
    mean_df_sorted_instrumentalness = mean_df.sort_values(by=['instrumentalness'], ascending=False)
    df_sorted_instrumentalness.instrumentalness *= 100
    mean_df_sorted_instrumentalness *= 100
    while inp_ not in ['Q', 'q']:
        inp_ = input(menu("Instrumentalness"))
        if inp_ in ['Q', 'q']: break
        elif inp_ == '1':
            py.figure(dpi=450, figsize=(6, 7))
            py.hist(df_sorted_instrumentalness.instrumentalness, bins=range(0, 105, 5), rwidth=0.7,
color='#A90036')
            py.axis(xmin=-1, xmax=101)
            ax = py.gca()
            py.yscale('log')
            ax.yaxis.set_ticks_position('both')
            ax.xaxis.set_ticks_position('both')
            py.grid()
            py.title("(Logarithmic) Distribution of Instrumentalness")
            py.xlabel("Instrumentalness")
            py.ylabel("Number of Tracks")
            py.savefig('instrumentalness_log_hist')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('instrumentalness_log_hist.png').show()
        elif inp_ == '2':
            py.figure(dpi=450, figsize=(7, 5))
            py.bar(x1, df_sorted_instrumentalness.instrumentalness.iloc[0:10], width=5, color='#C4C82F')
            names = list(df_sorted_instrumentalness.name.iloc[0:10])
            py.axis(ymin=80, ymax=95)
            py.yticks(range(80, 96))
            names_new = []
            for name in names:
                words = name.split(" ")
                names_new.append([" ".join(words[i:i + 2]) for i in range(0, len(words), 2)])
            for name in names_new:
                names_new[names_new.index(name)] = str.join("\n", name)
            py.xticks(x1, names_new, fontsize=6)
            py.title("Songs with the Highest Instrumentalness")
            py.ylabel("Instrumentalness (out of 100)")
            py.grid(axis='y')
            ax = py.gca()
            ax.yaxis.set_ticks_position('both')
            ax.yaxis.set_minor_locator(AutoMinorLocator())
            py.savefig('instrumentalness_bar_top10')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('instrumentalness_bar_top10.png').show()
        elif inp_ == '3':
            py.figure(dpi=450)
            py.bar(x1, mean_df_sorted_instrumentalness.instrumentalness.iloc[0:10], width=5,
color='#75C883')
            artists = list(mean_df_sorted_instrumentalness.iloc[0:10, 1:2].index)
            py.yscale('log')
            try: artists[artists.index("Joey Bada$$")] = "Joey Badass"
            except ValueError: pass
            except Exception as e: print(e)
            artists_new = []
            for artist in artists:
                words = artist.split(" ")
                artists_new.append([" ".join(words[i:i + 1]) for i in range(0, len(words), 1)])
            for artist in artists_new: artists_new[artists_new.index(artist)] = str.join("\n", artist)
            py.xticks(x1, artists_new, fontsize=7)
            py.title("Artists with the Highest Instrumentalness (Logarithmic)")
            py.ylabel("Instrumentalness (out of 100)")
            py.axis(ymin=5, ymax=81)
            py.grid(axis='y', which='major')
            py.grid(axis='y', which='minor', linewidth=0.3)
            ax = py.gca()
            ax.yaxis.set_minor_formatter(ScalarFormatter())
            ax.yaxis.set_major_formatter(ScalarFormatter())
            ax.yaxis.set_ticks_position('both')
            py.savefig('instrumentalness_log_bar_artists_top10')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('instrumentalness_log_bar_artists_top10.png').show()
        else: print("Invalid Input")
def show_liveness():
    inp_ = 0
    df_sorted_liveness = df.sort_values(by=['liveness'], ascending=False)
    mean_df_sorted_liveness = mean_df.sort_values(by=['liveness'], ascending=False)
    df_sorted_liveness.liveness *= 100
    mean_df_sorted_liveness *= 100
```

```
    while inp_ not in ['Q', 'q']:
        inp_ = input(menu("Liveness"))
        if inp_ in ['Q', 'q']: break
        elif inp_ == '1':
            py.figure(dpi=450)
            py.hist(df_sorted_liveness.liveness, bins=range(0, 105, 5), rwidth=0.7, color='#C86A72')
            py.axis(xmin=-1, xmax=101, ymax=325)
            ax = py.gca()
            py.xticks(range(0, 105, 5), fontsize=8)
            py.yticks(range(0, 350, 25))
            ax.yaxis.set_ticks_position('both')
            ax.xaxis.set_ticks_position('both')
            ax.yaxis.set_minor_locator(AutoMinorLocator())
            py.grid()
            py.title("Distribution of Liveness")
            py.xlabel("Liveness (out of 100)")
            py.ylabel("Number of Tracks")
            py.savefig('liveness_hist')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('liveness_hist.png').show()
        elif inp_ == '2':
            py.figure(dpi=450)
            py.bar(x1, df_sorted_liveness.liveness.iloc[0:10], width=5, color='#CB522E')
            names = list(df_sorted_liveness.name.iloc[0:10])
            py.axis(ymin=75, ymax=95)
            names_new = []
            for name in names:
                words = name.split(" ")
                names_new.append([" ".join(words[i:i + 1]) for i in range(0, len(words), 1)])
            for name in names_new:
                names_new[names_new.index(name)] = str.join("\n", name)
            py.xticks(x1, names_new, fontsize=6)
            py.title("Songs with the Highest Liveness")
            py.ylabel("Liveness (out of 100)")
            py.grid(axis='y')
            ax = py.gca()
            ax.yaxis.set_ticks_position('both')
            ax.yaxis.set_minor_locator(AutoMinorLocator())
            py.savefig('liveness_bar_top10')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('liveness_bar_top10.png').show()
        elif inp_ == '3':
            py.figure(dpi=450)
            py.bar(x1, mean_df_sorted_liveness.liveness.iloc[0:10], width=5, color='#00A995')
            artists_ = list(mean_df_sorted_liveness.iloc[0:10, 1:2].index)
            try: artists_[artists_.index("Joey Bada$$")] = "Joey Badass"
            except ValueError: pass
            except Exception as e: print(e)
            artists_new = []
            for artist in artists_:
                words = artist.split(" ")
                artists_new.append([" ".join(words[i:i + 1]) for i in range(0, len(words), 1)])
            for artist in artists_new: artists_new[artists_new.index(artist)] = str.join("\n", artist)
            py.xticks(x1, artists_new, fontsize=7)
            py.title("Artists with the Highest Liveness")
            py.ylabel("Liveness (out of 100)")
            py.axis(ymin=27, ymax=37.5)
            py.yticks(np.array(range(55, 76, 5))/2)
            py.grid(axis='y', which='major')
            py.grid(axis='y', which='minor', linewidth=0.3)
            ax = py.gca()
            ax.yaxis.set_minor_locator(AutoMinorLocator())
            ax.yaxis.set_ticks_position('both')
            ax.yaxis.set_minor_formatter(ScalarFormatter())
            ax.tick_params(which='minor', axis='y', labelsize=7)
            py.savefig('liveness_bar_artists_top10')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('liveness_bar_artists_top10.png').show()
        else: print("Invalid Input")
def show_popularity():
    inp_ = 0
    df_sorted_popularity = df.sort_values(by=['popularity'], ascending=False)
    mean_df_sorted_popularity = mean_df.sort_values(by=['popularity'], ascending=False)
    while inp_ not in ['Q', 'q']:
        inp_ = input(popularity_menu)
        if inp_ in ['Q', 'q']: break
        elif inp_ == '1':
            py.figure(dpi=300)
            py.hist(df.popularity, bins=range(40, 105, 5), color='#FFBA71')
            py.xticks(range(40, 105, 5))
            py.yticks(range(0, 275, 25))
```

```
            py.title("Distribution of Popularity of Tracks")
            py.grid()
            py.ylabel("Number of Tracks")
            py.xlabel("Popularity")
            py.savefig('popularity_hist')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('popularity_hist.png').show()
        elif inp_ == '2':
            py.figure(dpi=300)
            py.hist(mean_df.popularity, bins=range(35, 100, 5))
            py.title("Distribution of Popularity of Artists (by top 10 tracks)")
            py.xticks(range(35, 100, 5))
            py.axis(ymax=35)
            py.ylabel("Number of artists")
            py.xlabel("Popularity")
            py.grid()
            py.savefig('popularity_artists_hist')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('popularity_artists_hist.png').show()
        elif inp_ == '3':
            py.figure(dpi=300)
            py.plot(mean_df_sorted_popularity.iloc[:10, 1], marker='.', color='orange', markersize=10)
            py.title("Popularity of top 10 Artists (by top 10 tracks)")
            py.xticks(fontsize=7, rotation=-20)
            py.yticks(range(84, 92))
            py.grid()
            py.savefig('popularity_line_artists_top10')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('popularity_line_artists_top10.png').show()
        elif inp_ == '4':
            py.figure(dpi=450, figsize=(7, 5))
            py.bar(x1, df_sorted_popularity.popularity.iloc[0:10], width=5, color='#CB522E')
            names = list(df_sorted_popularity.name.iloc[0:10])
            py.axis(ymin=90, ymax=98)
            names_new = []
            for name in names:
                words = name.split(" ")
                names_new.append([" ".join(words[i:i + 1]) for i in range(0, len(words), 1)])
            for name in names_new: names_new[names_new.index(name)] = str.join("\n", name)
            py.xticks(x1, names_new, fontsize=6)
            py.title("Most Popular Songs")
            py.ylabel("Popularity (out of 100)")
            py.grid(axis='y')
            ax = py.gca()
            ax.yaxis.set_ticks_position('both')
            py.savefig('popularity_bar_top10')
            py.close()
            print("Graph saved in enclosing folder and opened in new window")
            img.open('popularity_bar_top10.png').show()
        else: print("Invalid Input")

x1 = range(5, 105, 10)
dispatcher = {1: show_df, 2: show_artist, 3: show_mode, 4: show_release_date, 5: show_key, 6:
show_popularity, 7: show_duration,
     8: show_danceability, 9: show_energy, 10: show_loudness, 11: show_valence, 12: show_speechiness,
     13: show_acousticness, 14: show_instrumentalness, 15: show_liveness, 16: show_tempo}

choices = ['Q', 'q'] + list(range(1, 17)) + list(map(str, range(1, 17)))
choice = 0
if __name__ == '__main__':
    while choice not in ['Q', 'q']:
        print(main_menu)
        inp = input(">>> ")
        if inp in ['Q', 'q']: break
        if inp not in choices:
            print("Invalid Choice.\nRETRY")
            continue
        try: choice = int(inp)
        except ValueError: choice = inp
        except Exception as e:
            print(f"[ERROR] {e}")
            continue
        dispatcher[choice]()

    print("[END] Goodbye!")
```

# Running Main.py

*(Repeated Menus not shown)*

```
arnavmangla@Arnavs-MacBook-Pro IP Project % python3 main.py
Main Menu:
1. DataFrame                2. Search for Artists in DataFrame
3. Graph for Mode           4. Graph for Release Date
5. Graph for Key            6. Popularity Menu
7. Duration Menu            8. Danceability Menu
9. Energy Menu              10. Loudness Menu
11. Valence Menu            12. Speechiness Menu
13. Acousticness Menu       14. Instrumentalness Menu
15. Liveness Menu           16. Tempo Menu
To exit any menu press 'q' or 'Q'
```

## 1. DataFrame

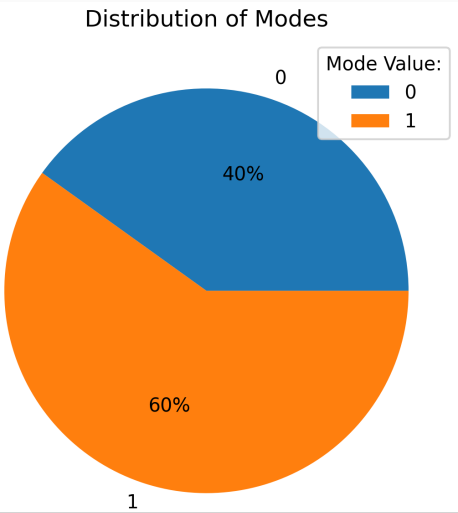
```
>>> 1
                                    name  popularity                          album_name album_release_date album_type  ...  instrumentalness liveness  valence
0                        Blinding Lights          94                      After Hours         2020-03-20      album  ...          0.000095   0.0897    0.334
1                             Sacrifice          92                          Dawn FM         2022-01-06      album  ...          0.000032   0.0678    0.905
2    Save Your Tears (with Ariana Grande) (Remix)          92  Save Your Tears (Remix)         2021-04-23     single  ...          0.000024   0.0936    0.593
3                One Right Now (with The Weeknd)          92                One Right Now         2021-11-05     single  ...          0.000000   0.0755    0.688
4              Moth To A Flame (with The Weeknd)          92             Moth To A Flame         2021-10-22     single  ...          0.000000   0.1050    0.109
..                                           ...         ...                         ...                ...        ...  ...               ...      ...      ...
995                                New Rules          83              Dua Lipa (Deluxe)         2017-06-02      album  ...          0.000016   0.1530    0.608
996                               We're Good          82  Future Nostalgia (The Moonlight Edition)         2021-02-11      album  ...          0.000000   0.1830    0.590
997                             Break My Heart          82                Future Nostalgia         2020-03-27      album  ...          0.000001   0.3490    0.467
998                                    IDGAF          82              Dua Lipa (Deluxe)         2017-06-02      album  ...          0.000000   0.0824    0.510
999                         UN DIA (ONE DAY)          72                           JOSE         2021-09-10      album  ...          0.000000   0.1740    0.402

[1000 rows x 19 columns]
```

## 2. Search for Artist in DataFrame

```
>>> 2
Enter Artist Name or press 'i' to list of Artists: i
('The Weeknd, Kanye West, Drake, Joyner Lucas, Eminem, J. Cole, Kendrick '
 'Lamar, Logic, Meek Mill, 50 Cent, Rihanna, Nicki Minaj, Cardi B, Travis '
 'Scott, Juice WRLD, Post Malone, Machine Gun Kelly, Closed on Sunday, Powfu, '
 'Kid Cudi, A$AP Rocky, Dr. Dre, 2pac, The Notorious B.I.G., JAY-Z, Pusha-T, '
 'Joey Bada$$, Hopsin, XXXTENTACION, Queen, The Beatles, Michael Jackson, '
 'Glass Animals, Taylor Swift, Imagine Dragons, Coldplay, Halsey, X '
 'Ambassadors, Hozier, The Chainsmokers, Avicii, Dj Snake, Martin Garrix, '
 'Skrillex, 20syl, Kygo, Ed Sheeran, Bastille, Billie Eilish, Duncan Laurence, '
 'Dempsey Hope, Christian French, Miley Cyrus, Frank Ocean, Ellie Goulding, '
 'Camila Cabello, Alec Benjamin, Bruno Mars, Lauv, Selena Gomez, Maroon 5, Jon '
 'Bellion, JP Saxe, Ariana Grande, Justin Bieber, Anson Seabra, Justin '
 'Timberlake, Khalid, 6LACK, Dean Lewis, One Direction, Katy Perry, Lady Gaga, '
 'Beyonce, Shawn Mendes, Sam Smith, Mac Miller, Linkin Park, Britney Spears, '
 'Alicia Keys, Black Eyed Peas, Green Day, Denzel Curry, AURORA, SZA, Lukas '
 'Graham, Bon Jovi, Sting, Backstreet Boys, Whitney Houston, David Bowie, '
 'AC/DC, Elvis Presley, Elton John, John Lennon, Bob Marley, ABBA, Paul '
 'McCartney, Adele, Dua Lipa')
Enter Artist Name: kante west
Did you mean any of ['Kanye West', 'Dean Lewis', 'One Direction']?
(Enter no or 1, 2, or 3 to select)
>>>1
                          name  popularity                 album_name album_release_date album_type  duration_sec  ... acousticness  instrumentalness
10                  Praise God          88                      Donda         2021-08-29      album         226.7  ...      0.00904          0.000095
11                City of Gods          87              City of Gods         2022-02-11     single         256.0  ...      0.10200          0.000000
12              Ni**as In Paris          86  Watch The Throne (Deluxe)         2011-08-08      album         219.3  ...      0.12700          0.000000
13                   Heartless          86          808s & Heartbreak         2008-11-24      album         211.0  ...      0.05150          0.000000
14                     Bound 2          85                      Yeezus         2013-06-18      album         229.1  ...      0.14500          0.000000
15                        Eazy          85                        Eazy         2022-01-15     single         234.3  ...      0.16700          0.000000
16  Father Stretch My Hands Pt. 1          84           The Life Of Pablo         2016-06-10      album         135.9  ...      0.11800          0.000000
17                   Hurricane          83                      Donda         2021-08-29      album         243.2  ...      0.04660          0.000000
18                All Falls Down          83         The College Dropout         2004-02-10      album         223.5  ...      0.14900          0.000000
19                      Stronger          83                 Graduation         2007-09-11      album         311.9  ...      0.00564          0.000000

[10 rows x 19 columns]
```

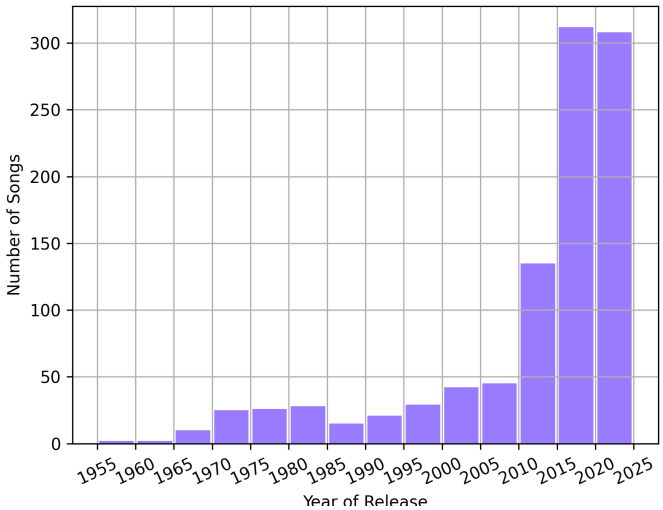
## 3. Graph for Mode

```
>>> 3
Graph saved in enclosing folder and opened in new window
```

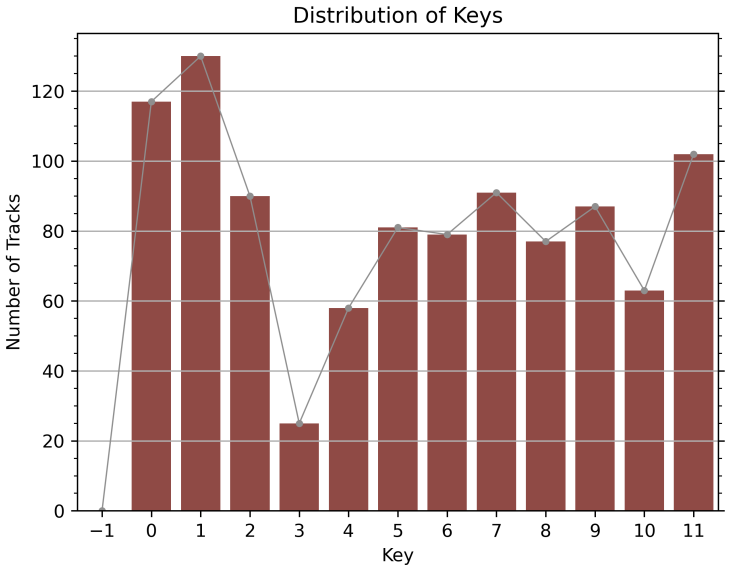


## 4. Graph for Release Date

```
>>> 4
Graph saved in enclosing folder and opened in new window
```



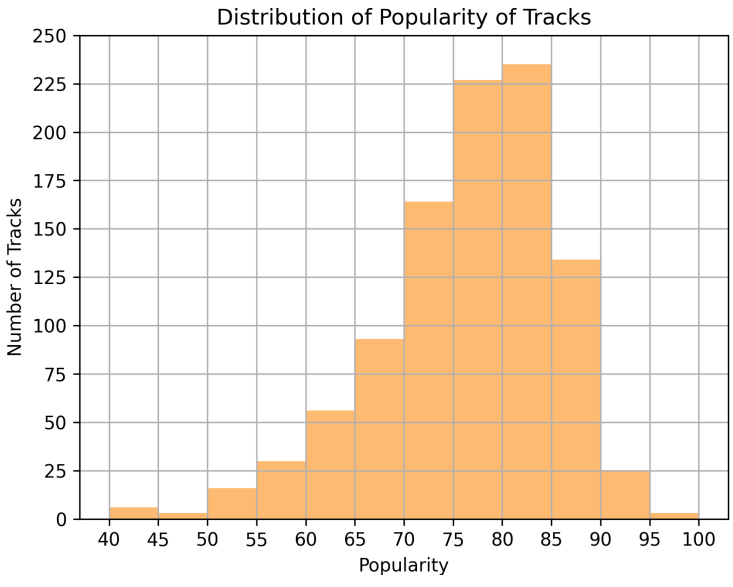

## 5. Graph for Key



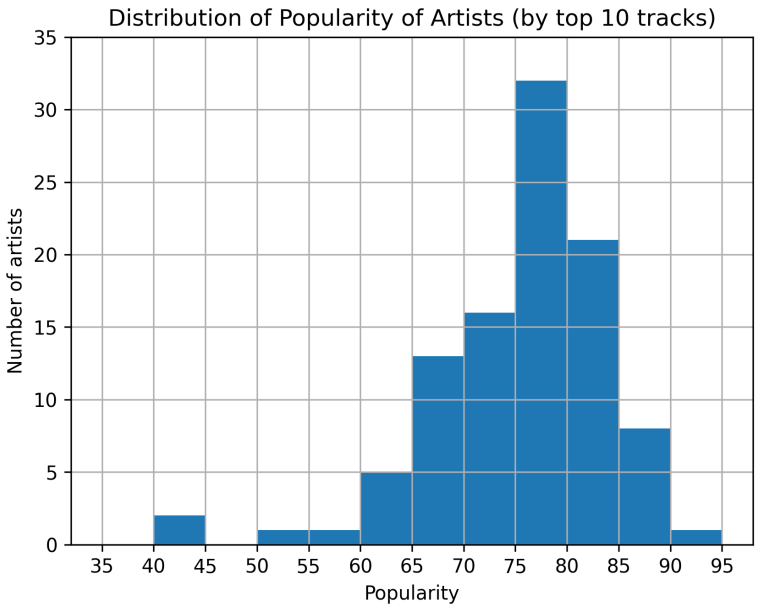

```
>>> 5
Graph saved in enclosing folder and opened in new window
```

## 6. Popularity Menu

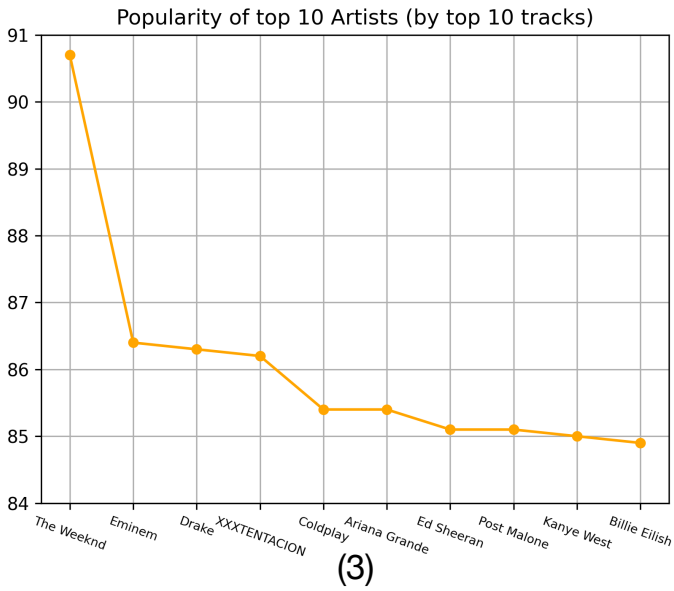
```
>>> 6

Popularity Menu:
1. Show Track Popularity Distribution          2. Show Artist Popularity Distribution
3. Show Artists with Highest Popularity        4. Show Songs with Highest Popularity
```



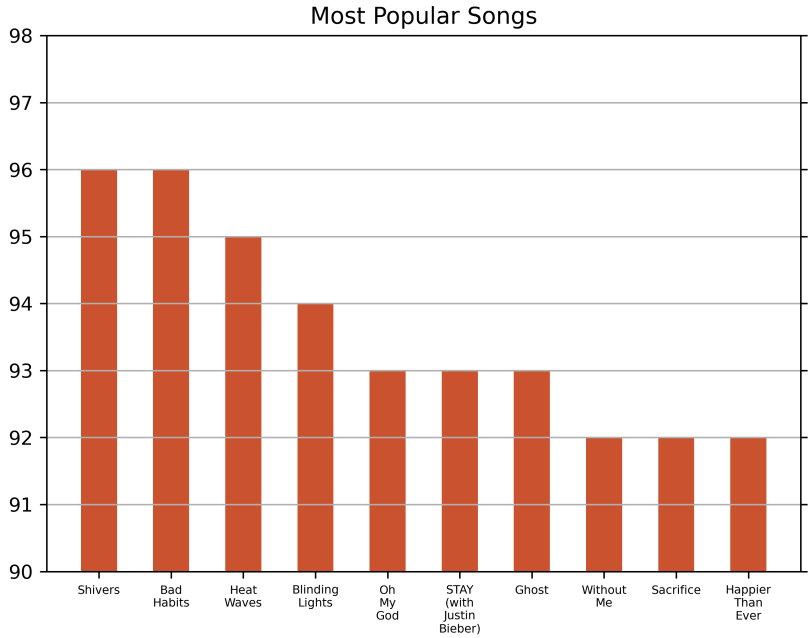

(1)




(2)



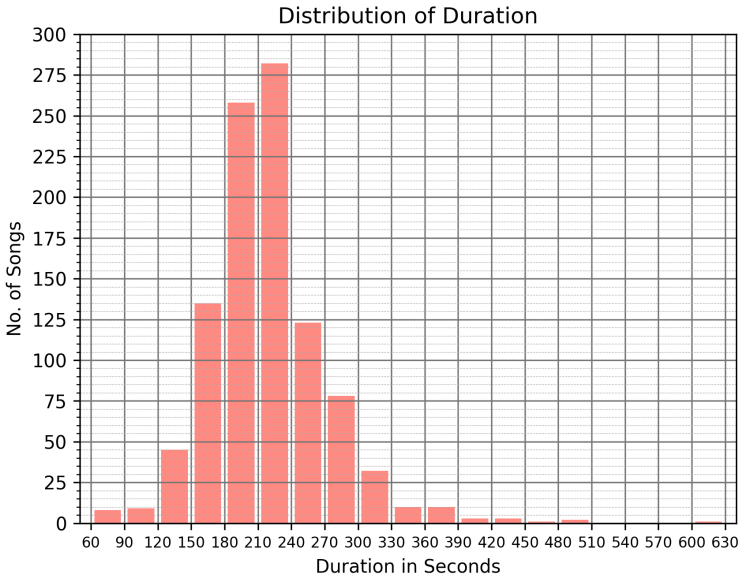

(3)



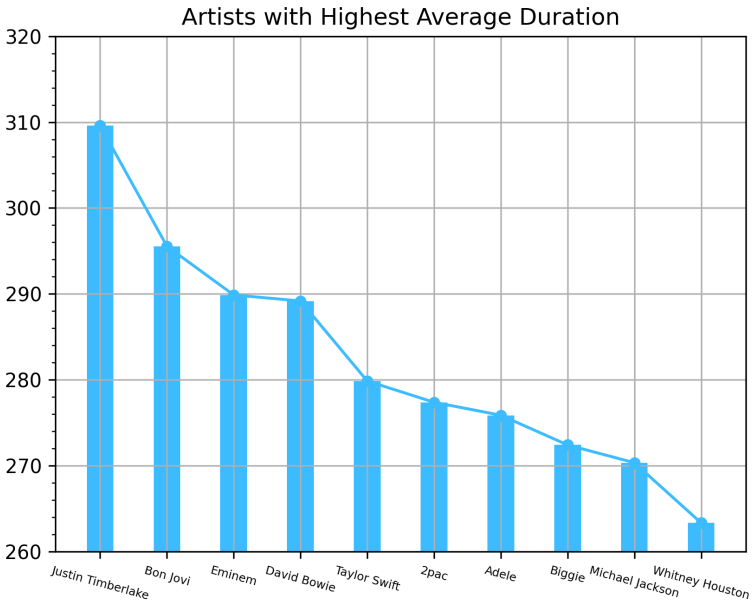

(4)

## 7. Duration Menu

```
>>> 7

Duration Menu:
1. Show Duration Distribution Histogram
2. Show Artists with Highest Average Duration
```



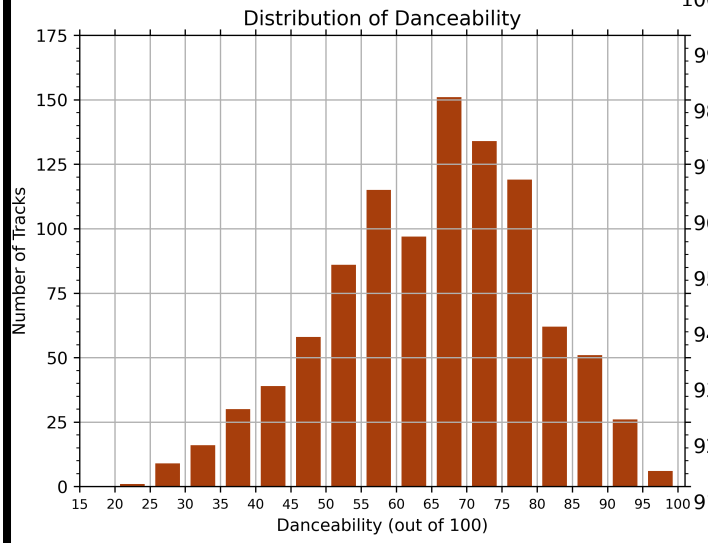

(1)



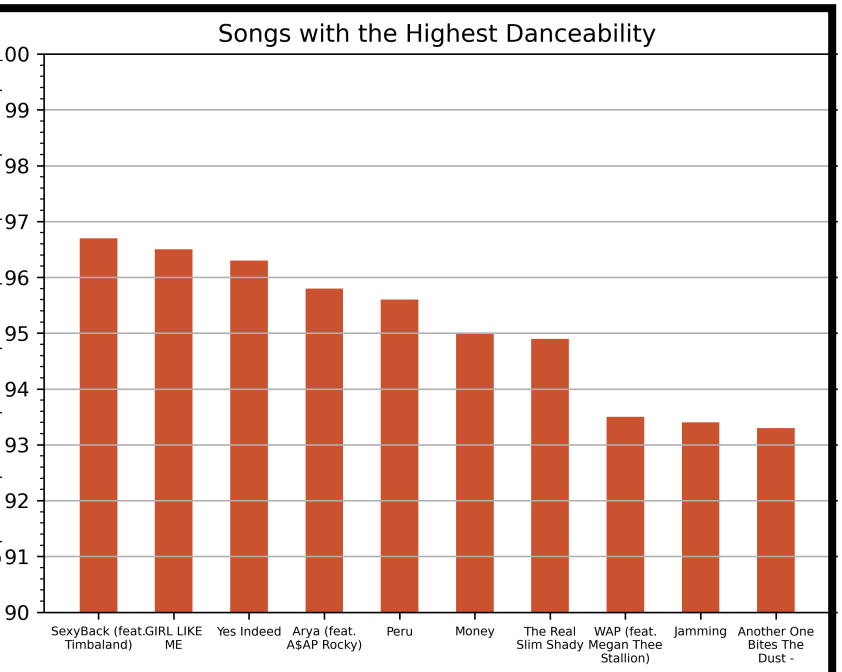

(2)

## 8. Danceability Menu

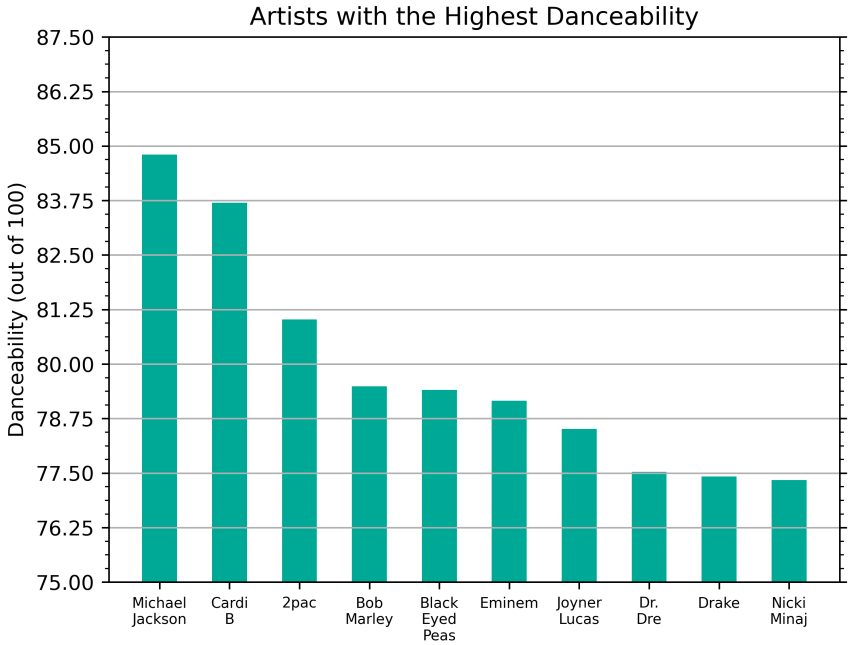
```
>>> 8

Danceability Menu:
1. Show Danceability Distribution Histogram
2. Show Songs with the Highest Danceability
3. Show Artists with the Highest Danceability
```




(1)




(2)

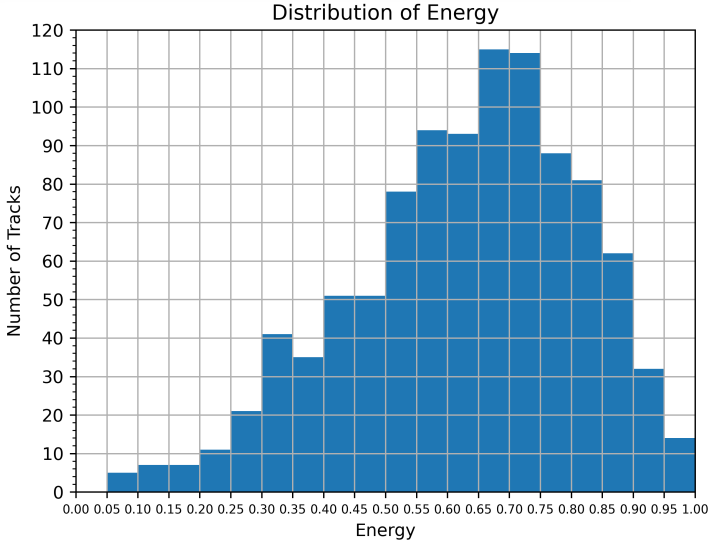


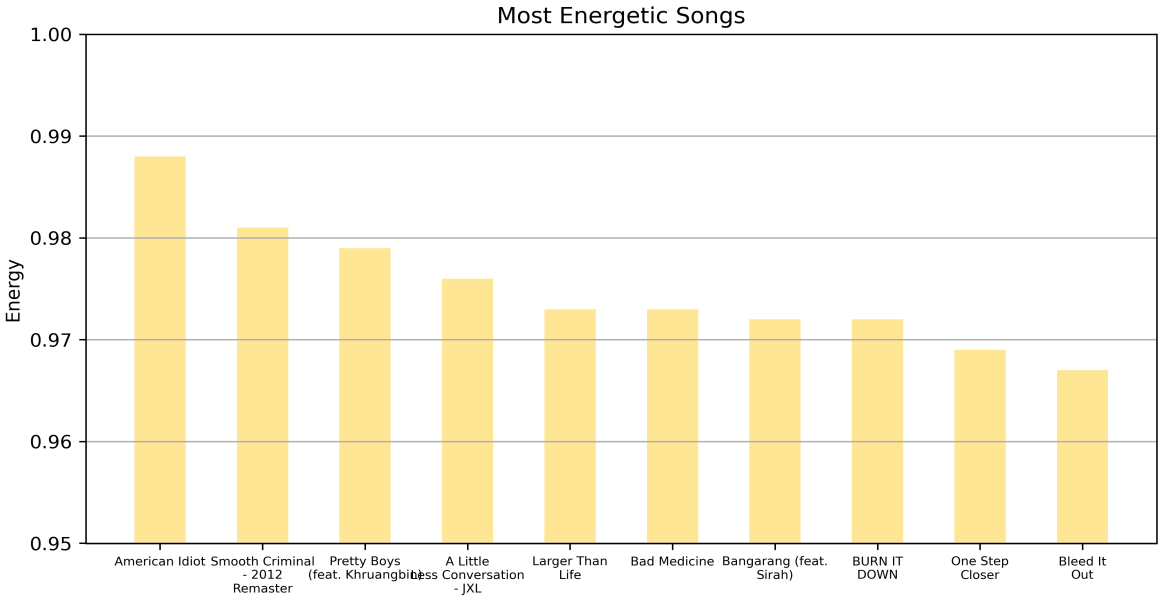

(3)

## 9. Energy Menu

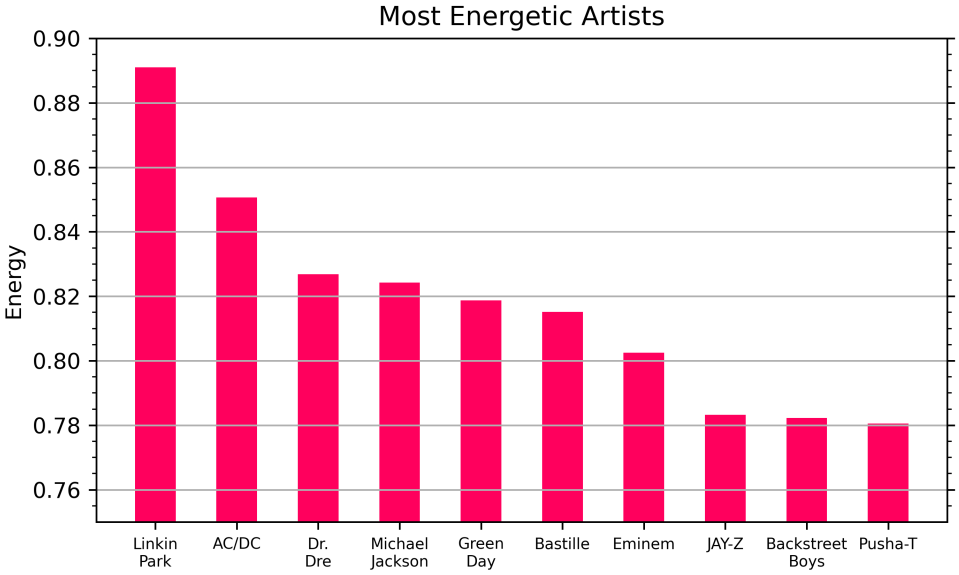
```
>>> 9

Energy Menu:
1. Show Energy Distribution Histogram
2. Show Songs with the Highest Energy
3. Show Artists with the Highest Energy
```

(1)





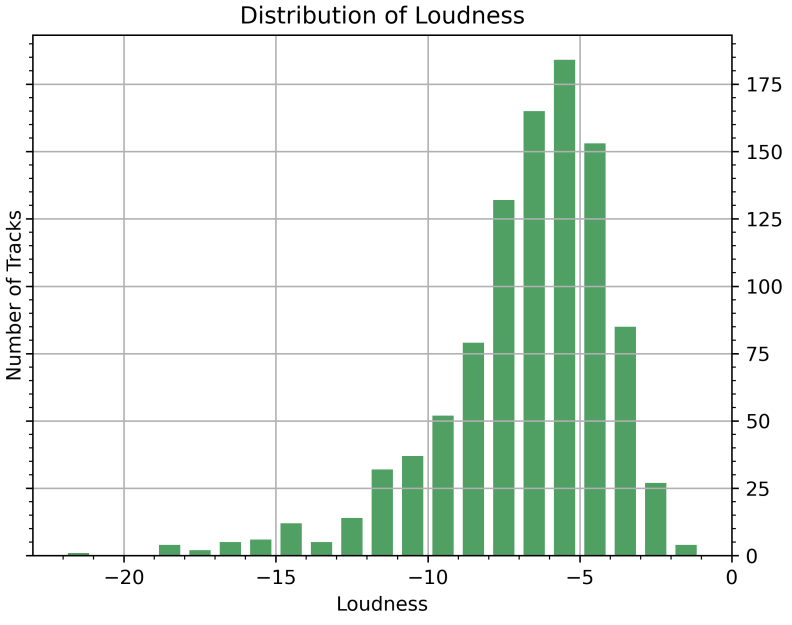

(2)



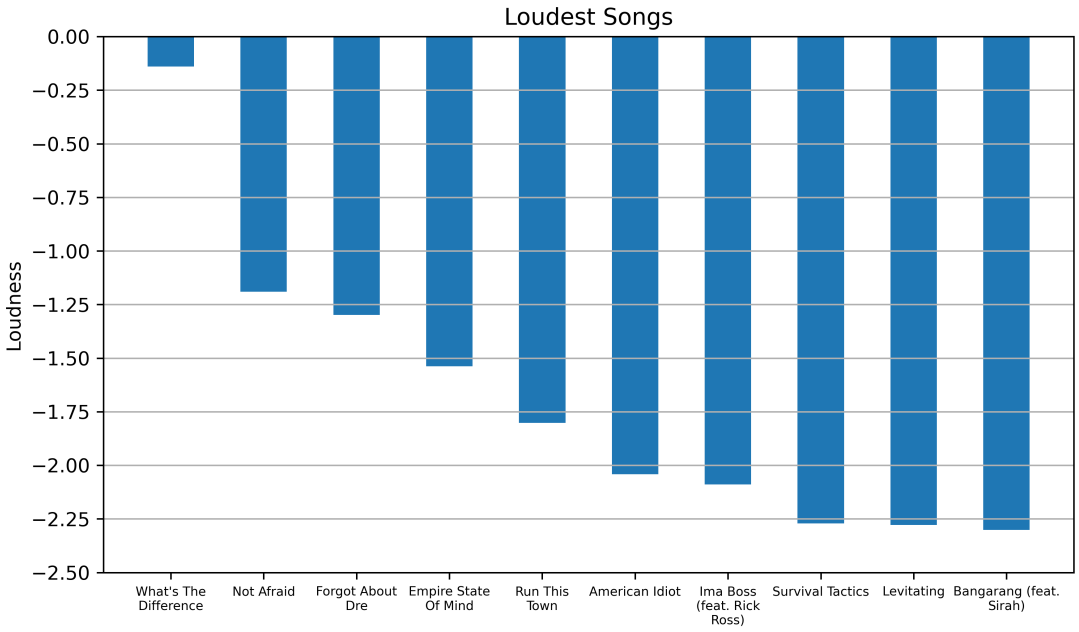

(3)

## 10. Loudness Menu

```
>>> 10

Loudness Menu:
1. Show Loudness Distribution Histogram        2. Show Loudest Songs
3. Show Loudest Artists
```

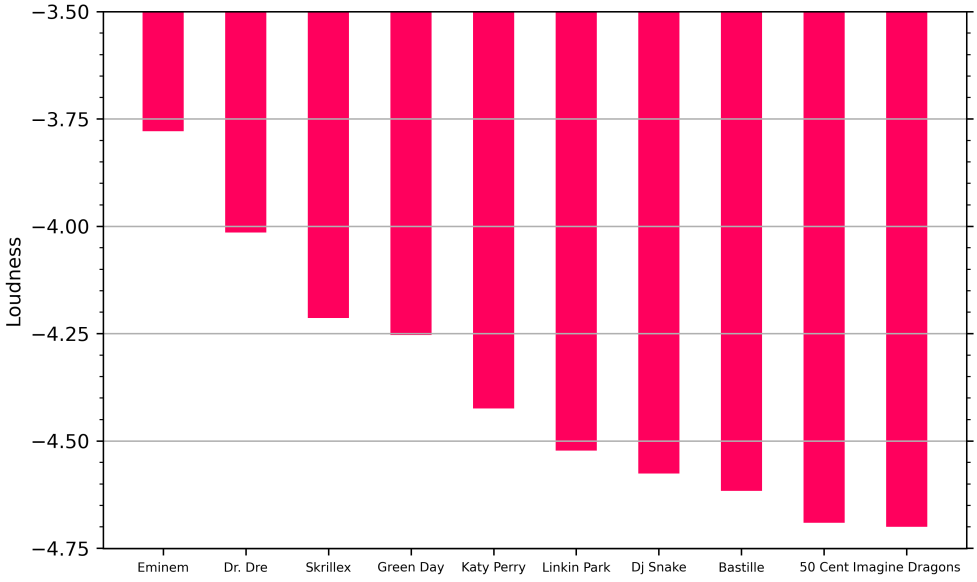



(1)



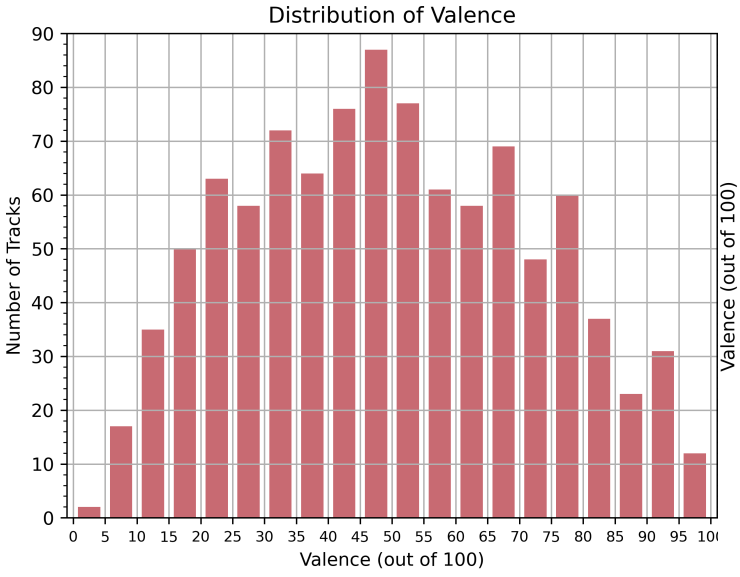

(2)

(3)

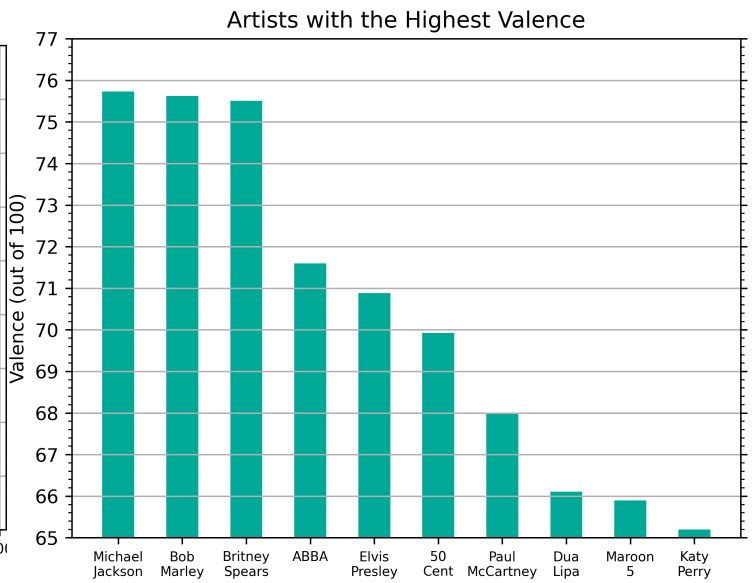



## 11. Valence Menu

```
>>> 11

Valence Menu:
1. Show Valence Distribution Histogram
2. Show Songs with the Highest Valence
3. Show Artists with the Highest Valence
```

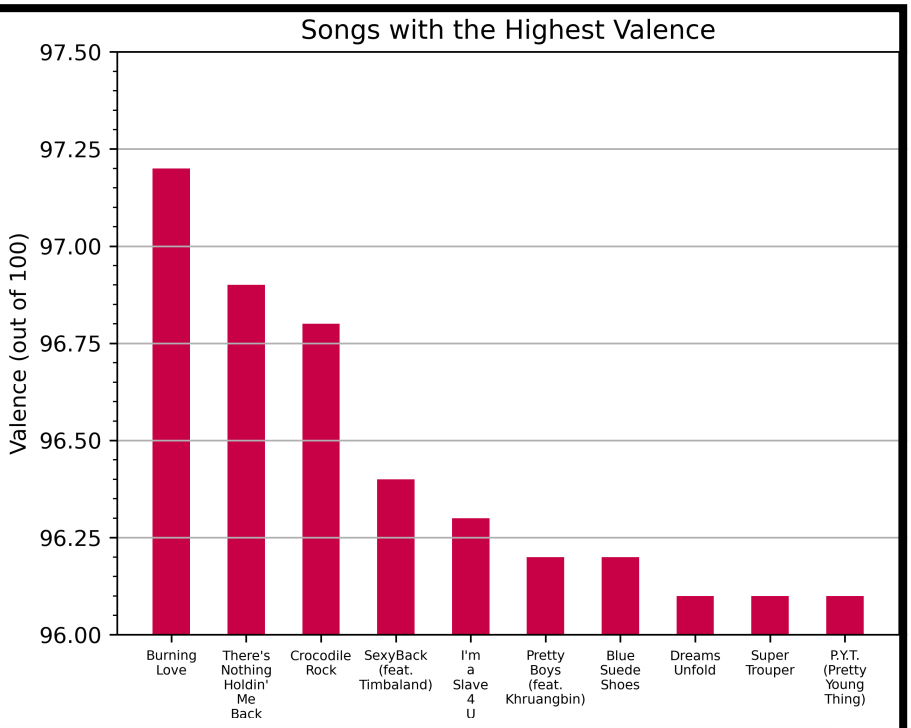



(1)




(3)

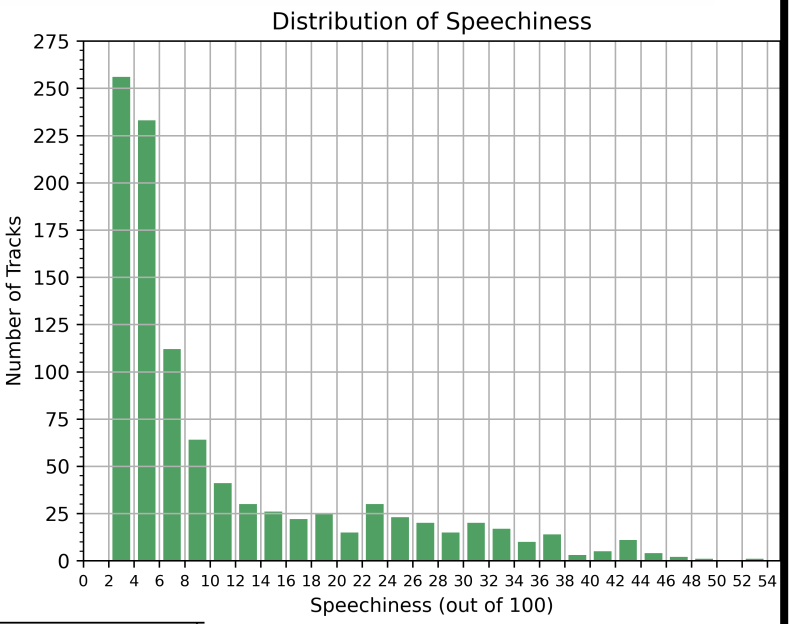
(2)



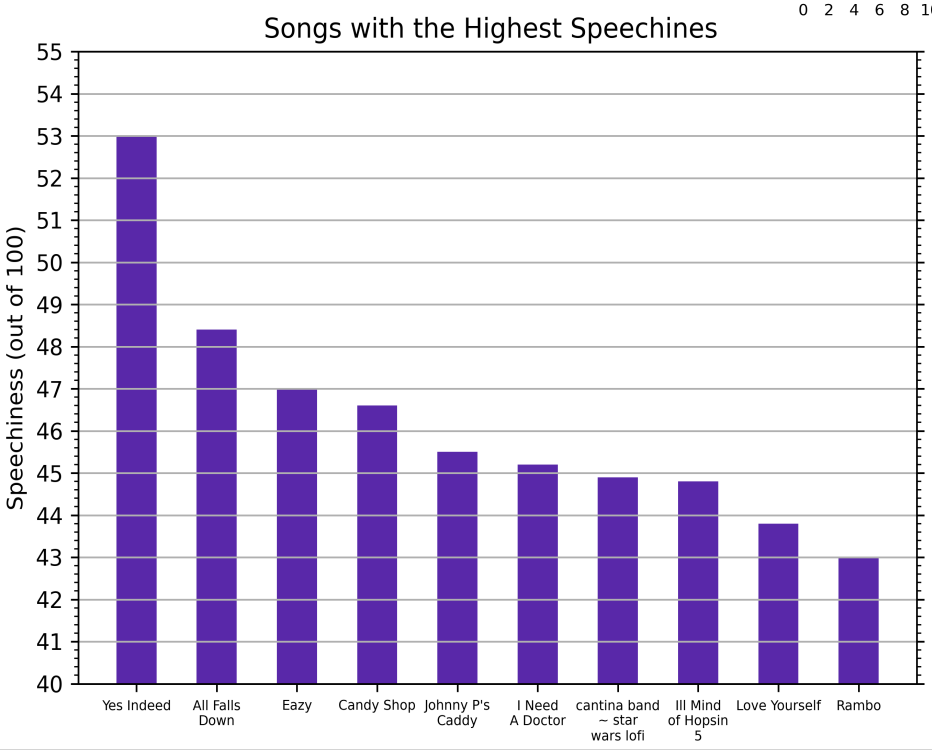

## 12. Speechiness Menu

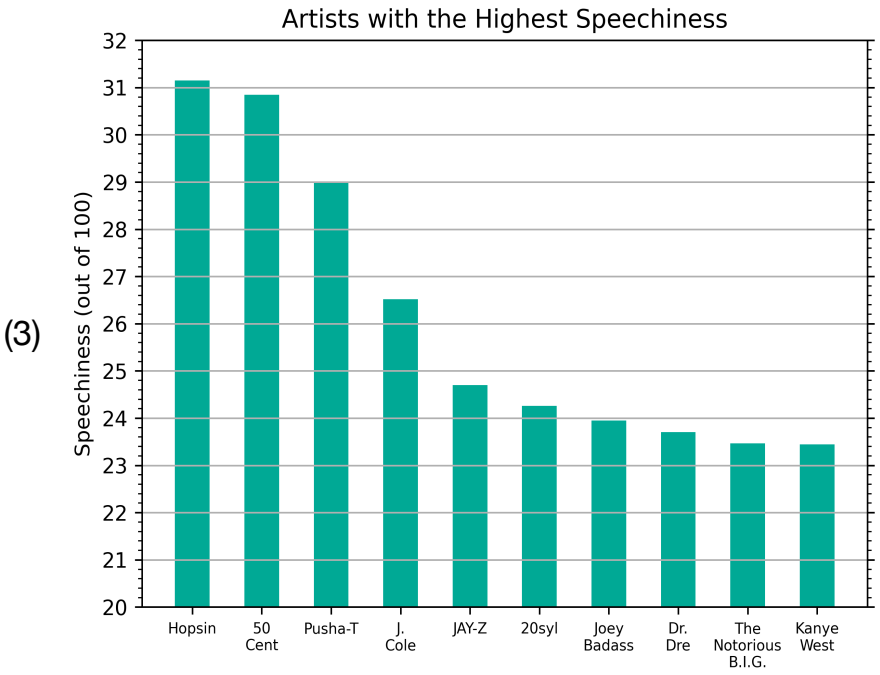
```
>>> 12

Speechiness Menu:
1. Show Speechiness Distribution Histogram
2. Show Songs with the Highest Speechiness
3. Show Artists with the Highest Speechiness
```

(1)




(2)

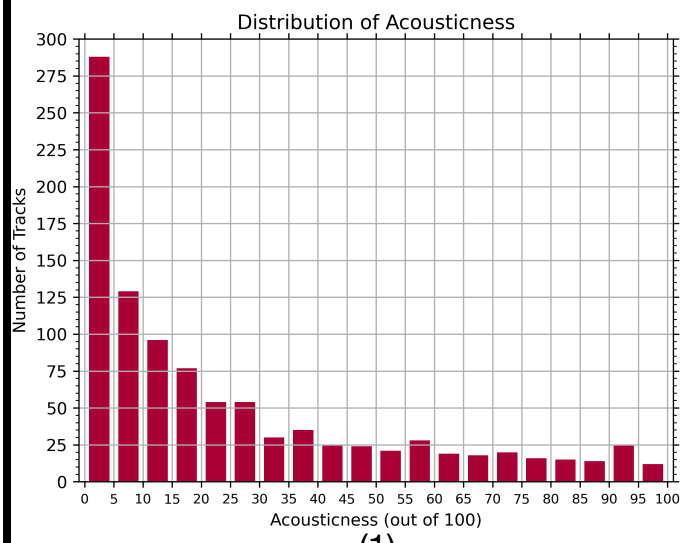




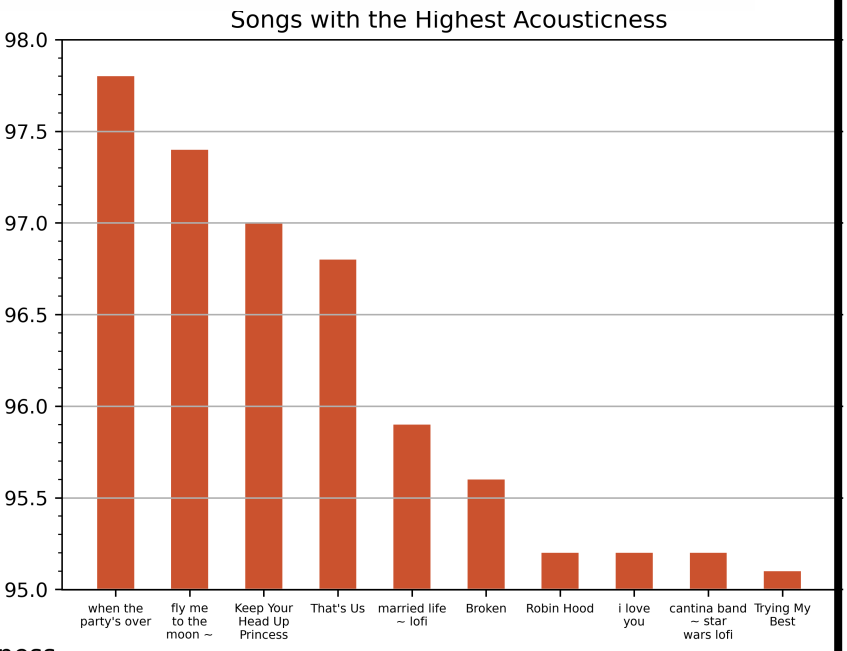

(3)

## 13. Acousticness Menu

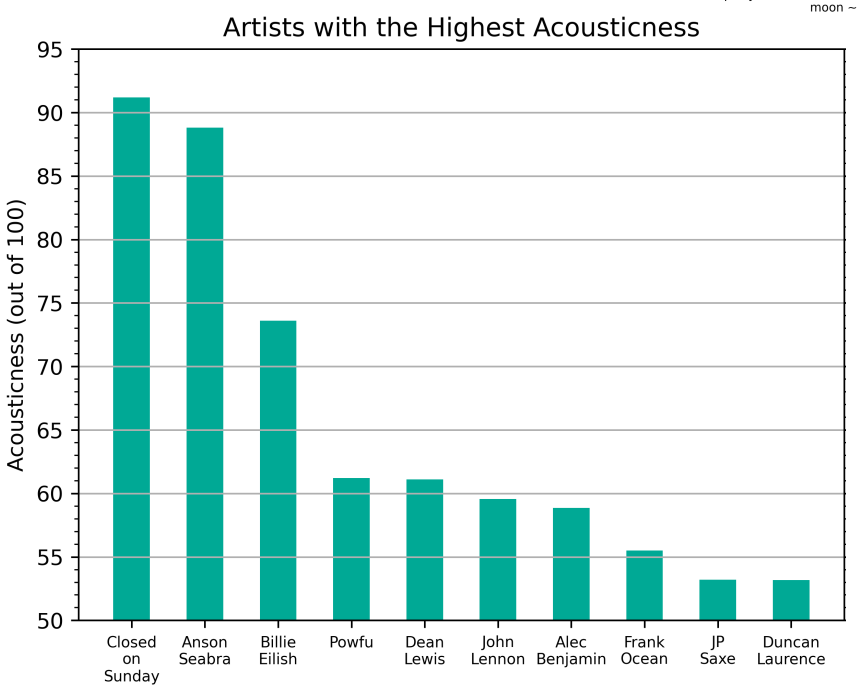
```
>>> 13

Acousticness Menu:
1. Show Acousticness Distribution Histogram
2. Show Songs with the Highest Acousticness
3. Show Artists with the Highest Acousticness
```




(1)



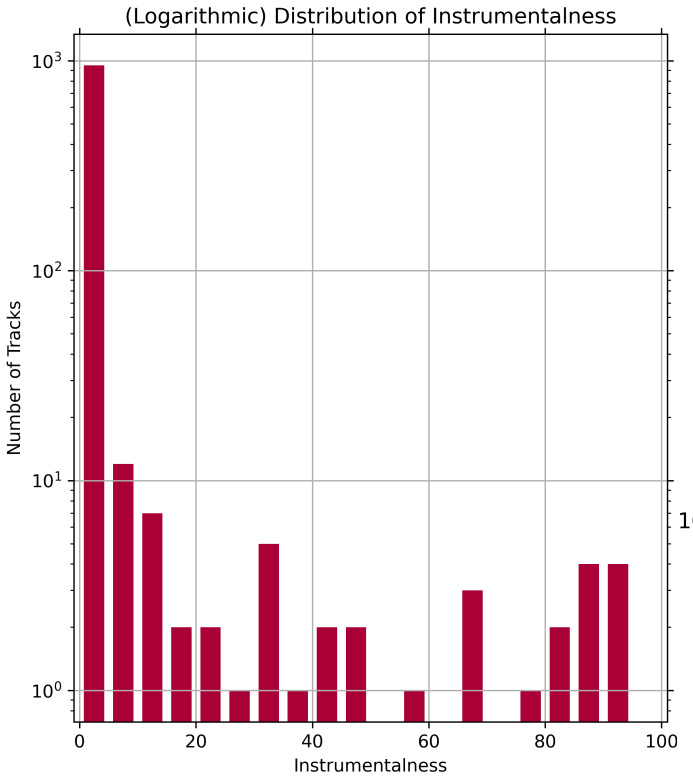

(2)



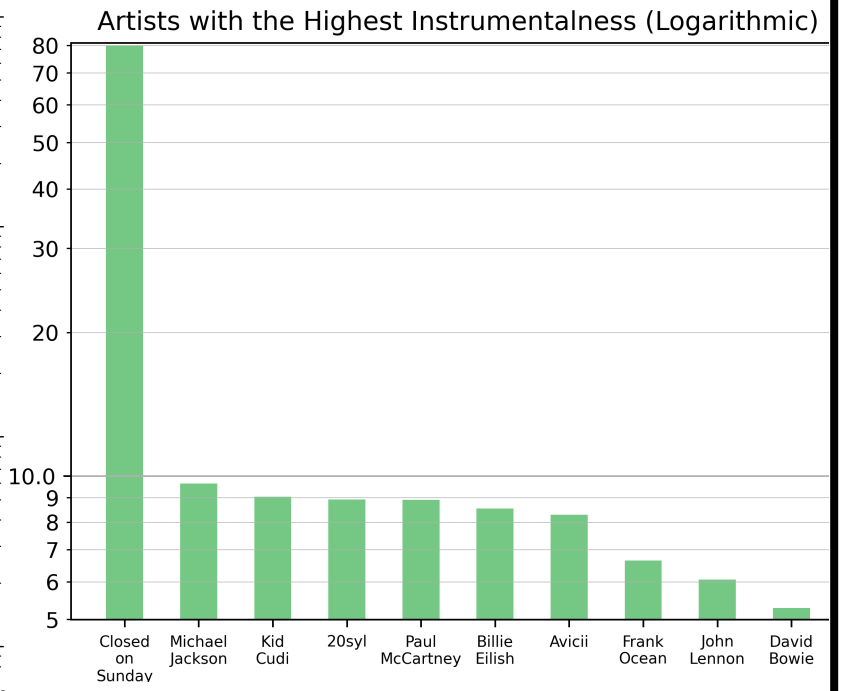

(3)

# 14. Instrumentalness Menu

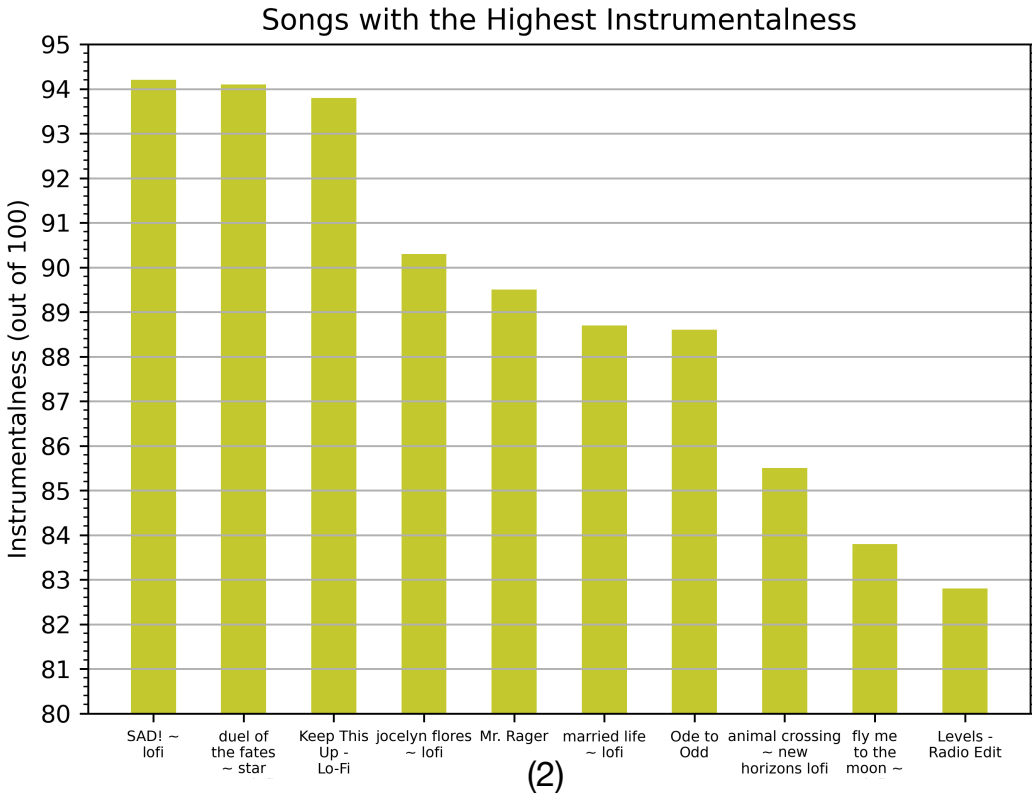
```
>>> 14

Instrumentalness Menu:
1. Show Instrumentalness Distribution Histogram          2. Show Songs with the Highest Instrumentalness
3. Show Artists with the Highest Instrumentalness
```




(1)



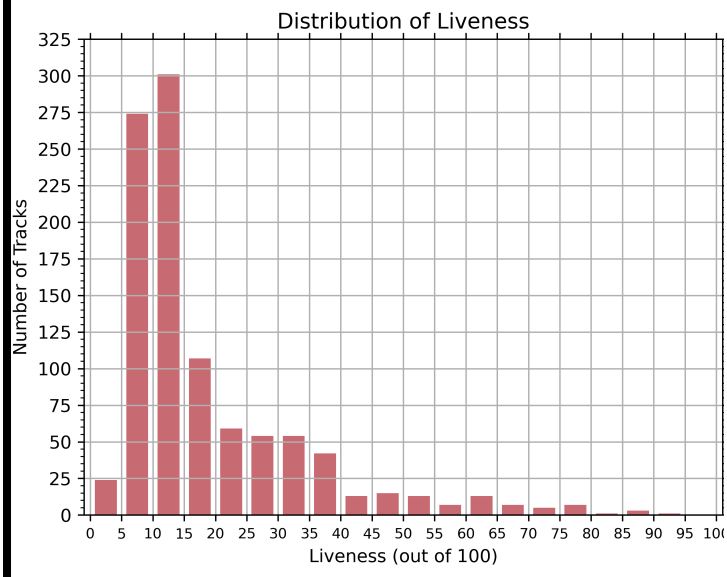

(3)



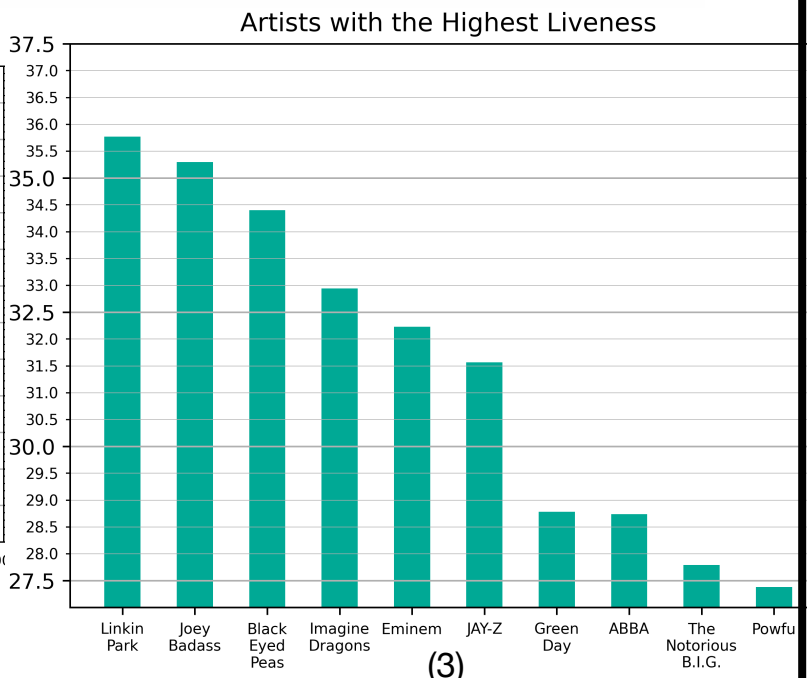

(2)

# 15. Liveness Menu

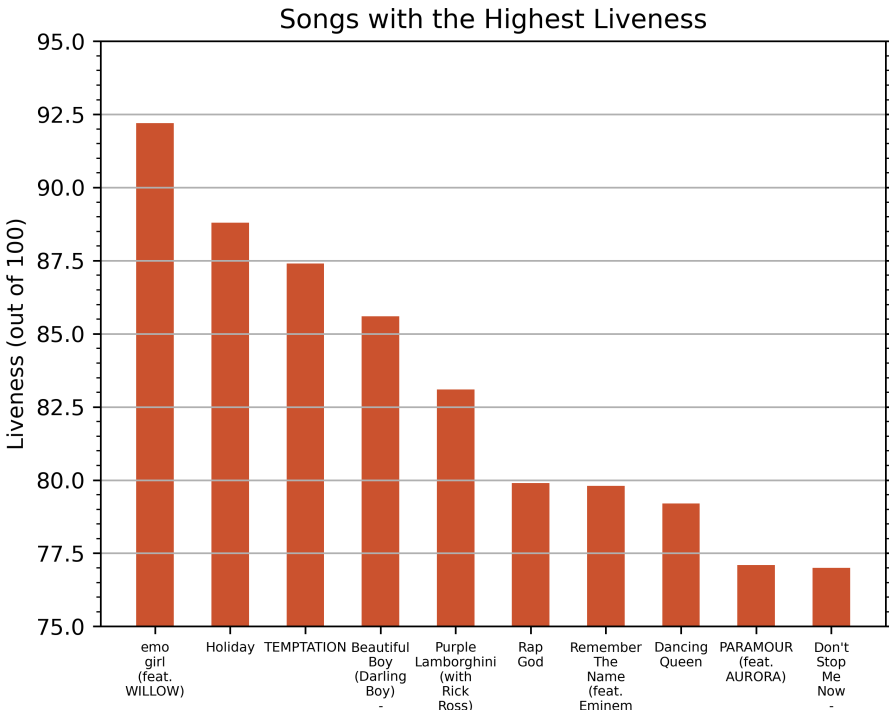
```
>>> 15

Liveness Menu:
1. Show Liveness Distribution Histogram          2. Show Songs with the Highest Liveness
3. Show Artists with the Highest Liveness
```




(1)



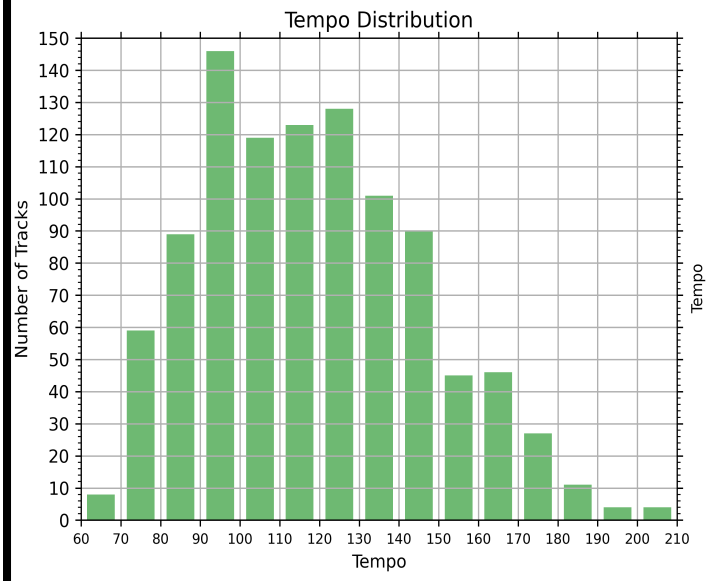

(3)



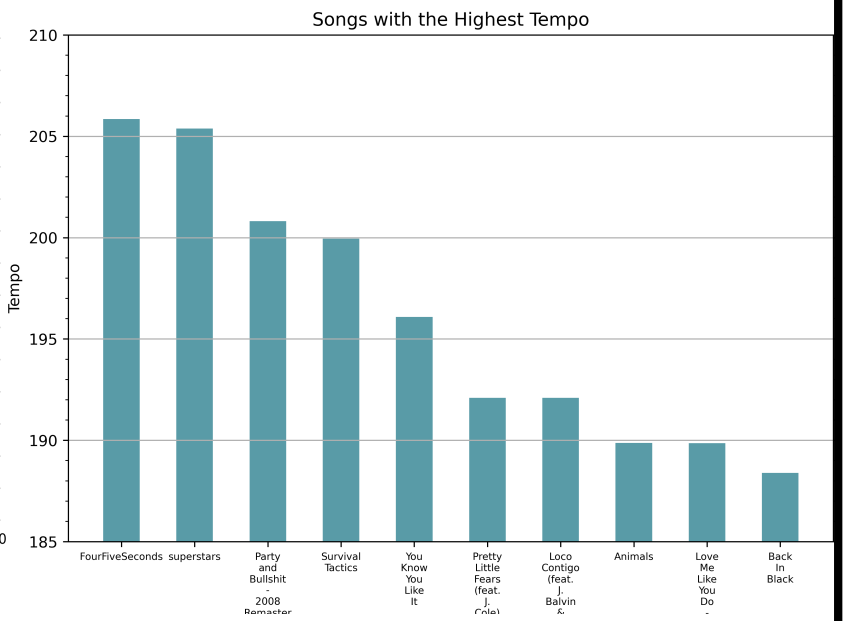

(2)

## 16. Tempo Menu

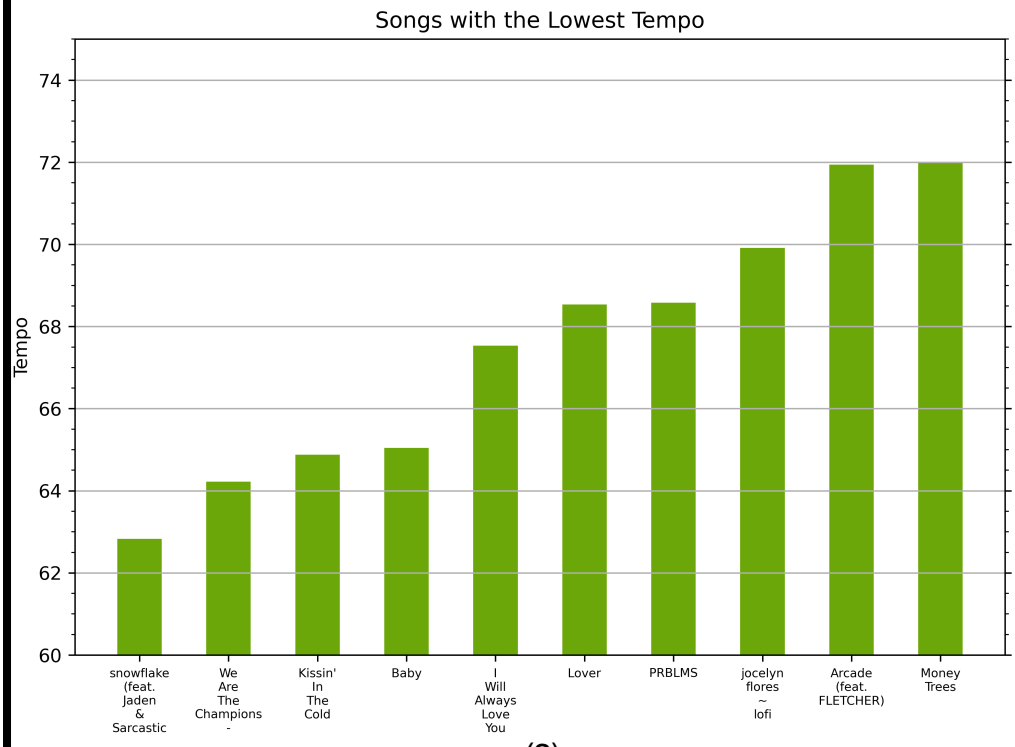
```
>>> 16

Tempo Menu:
1. Show Tempo Distribution Histogram                    2. Show Fastest Songs
3. Show Slowest Songs
```




(1)




(2)




(3)

# Conclusion

Through this project I have acquired extensive knowledge in:

- Working with APIs
- Working with Classes
- Working with Pandas Series and Dataframes
- Drawing graphs and charts using Pyplot (Matplotlib)

This project has also taught me about the importance of data analysis and visualisation.

# Bibliography


- stackoverflow.com
- geeksforgeeks.org
- programiz.com
- https://en.wikipedia.org/wiki/Pitch_class
- https://developer.spotify.com/documentation/web-api/reference/#/
- https://matplotlib.org/stable/api/pyplot_summary.html
- https://youtu.be/xdq6Gz33khQ