# Anna's Blog

Actualizaciones sobre el Archivo de Anna, la biblioteca verdaderamente abierta más grande en la historia de la humanidad.

## Backing up Spotify

*annas-archive.li/blog, 2025-12-20, Discuss on Hacker News*

We backed up Spotify (metadata and music files). It's distributed in bulk torrents (~300TB), grouped by popularity.

This release includes the largest publicly available music metadata database with 256 million tracks and 186 million unique ISRCs.

It's the world's first "preservation archive" for music which is fully open (meaning it can easily be mirrored by anyone with enough disk space), with 86 million music files, representing around 99.6% of listens.

Anna's Archive normally focuses on text (e.g. books and papers). We explained in "The critical window of shadow libraries" that we do this because text has the highest information density. But our mission (preserving humanity's knowledge and culture) doesn't distinguish among media types. Sometimes an opportunity comes along outside of text. This is such a case.

A while ago, we discovered a way to scrape Spotify at scale. We saw a role for us here to build a music archive primarily aimed at preservation.

Generally speaking, music is already fairly well preserved. There are many music enthusiasts in the world who digitized their CD and LP collections, shared them through torrents or other digital means, and meticulously catalogued them.

However, these existing efforts have some major issues:

1. **Over-focus on the most popular artists.** There is a long tail of music which only gets preserved when a single person cares enough to share it. And such files are often poorly seeded.
2. **Over-focus on the highest possible quality.** Since these are created by audiophiles with high end equipment and fans of a particular artist, they chase the highest possible file quality (e.g. lossless FLAC). This inflates the file size and makes it hard to keep a full archive of all music that humanity has ever produced.
3. **No authoritative list of torrents aiming to represent all music ever produced.** An equivalent of our book torrent list (which aggregate torrents from LibGen, Sci-Hub, Z-Lib,

and many more) does not exist for music.

This Spotify scrape is our humble attempt to start such a "preservation archive" for music. Of course Spotify doesn't have all the music in the world, but it's a great start.

Before we dive into the details of this collection, here is a quick overview:

- Spotify has around 256 million tracks. This collection contains metadata for an estimated 99.9% of tracks.
- We archived around 86 million music files, representing around 99.6% of listens. It's a little under 300TB in total size.
- We primarily used Spotify's "popularity" metric to prioritize tracks. View the top 10,000 most popular songs in this HTML file (13.8MB gzipped).
- For `popularity>0`, we got close to all tracks on the platform. The quality is the original OGG Vorbis at 160kbit/s. Metadata was added without reencoding the audio (and an archive of diff files is available to reconstruct the original files from Spotify, as well as a metadata file with original hashes and checksums).
- For `popularity=0`, we got files representing about half the number of listens (either original or a copy with the same ISRC). The audio is reencoded to OGG Opus at 75kbit/s — sounding the same to most people, but noticeable to an expert.
- The cutoff is 2025-07, anything released after that date may not be present (though in some cases it is).
- This is by far the largest music metadata database that is publicly available. For comparison, we have 256 million tracks, while others have 50-150 million. Our data is well-annotated: MusicBrainz has 5 million unique ISRCs, while our database has 186 million.
- This is the world's first "preservation archive" for music which is fully open (meaning it can easily be mirrored by anyone with enough disk space).

The data will be released in different stages on our Torrents page:

- [X] Metadata (Dec 2025)
- [ ] Music files (releasing in order of popularity)
- [ ] Additional file metadata (torrent paths and checksums)
- [ ] Album art
- [ ] .zstdpatch files (to reconstruct original files before we added embedded metadata)

For now this is a torrents-only archive aimed at preservation, but if there is enough interest, we could add downloading of individual files to Anna's Archive. Please let us know if you'd like this.

**Please help preserve these files:**

1. Donate to Anna's Archive. Any amount helps!
2. Seed these torrents (on the Torrents page of Anna's Archive). Even a seeding a few torrents helps!

With your help, humanity's musical heritage will be forever protected from destruction by natural disasters, wars, budget cuts, and other catastrophes.

In this blog we will analyze the data and look at details of the release. We hope you enjoy.

— Volunteer "ez" of Anna's Archive team

♫ ♫ ♫ ♫ ♫

# Data Exploration

Let's dive into the data! Here's some high-level statistics pulled from the metadata:

## 📊 Spotify Collection Overview

| Artists | Albums | Songs |
|---------|--------|-------|
| 15.43M | 58.6M | 256M |

### Songs / Tracks

Spotify has around 256 million tracks.

The most convenient available way to sort songs on Spotify is using the popularity metric, defined as follows:

> 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.
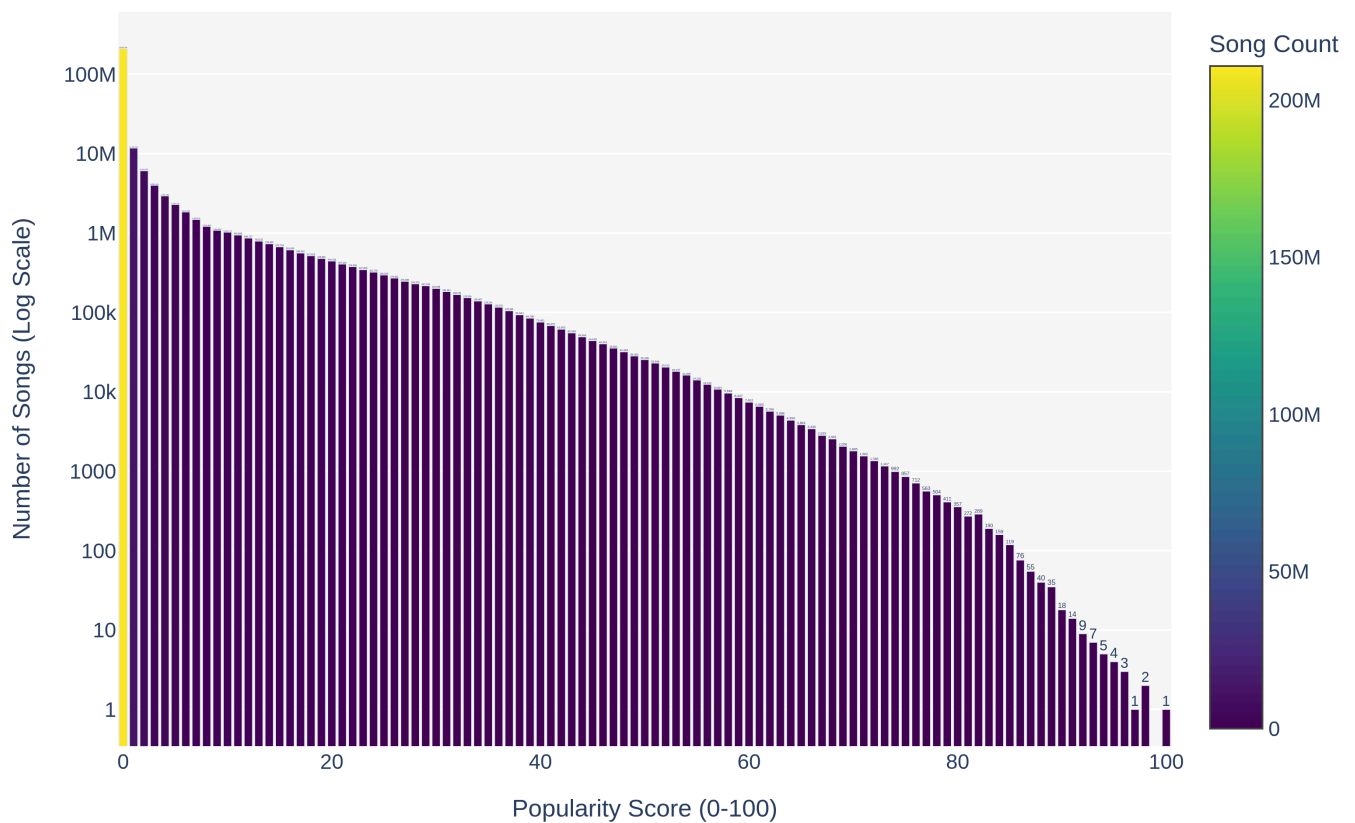
If we group songs by popularity, we see that there is an extremely large tail end:

# ⭐ Songs by Popularity Score
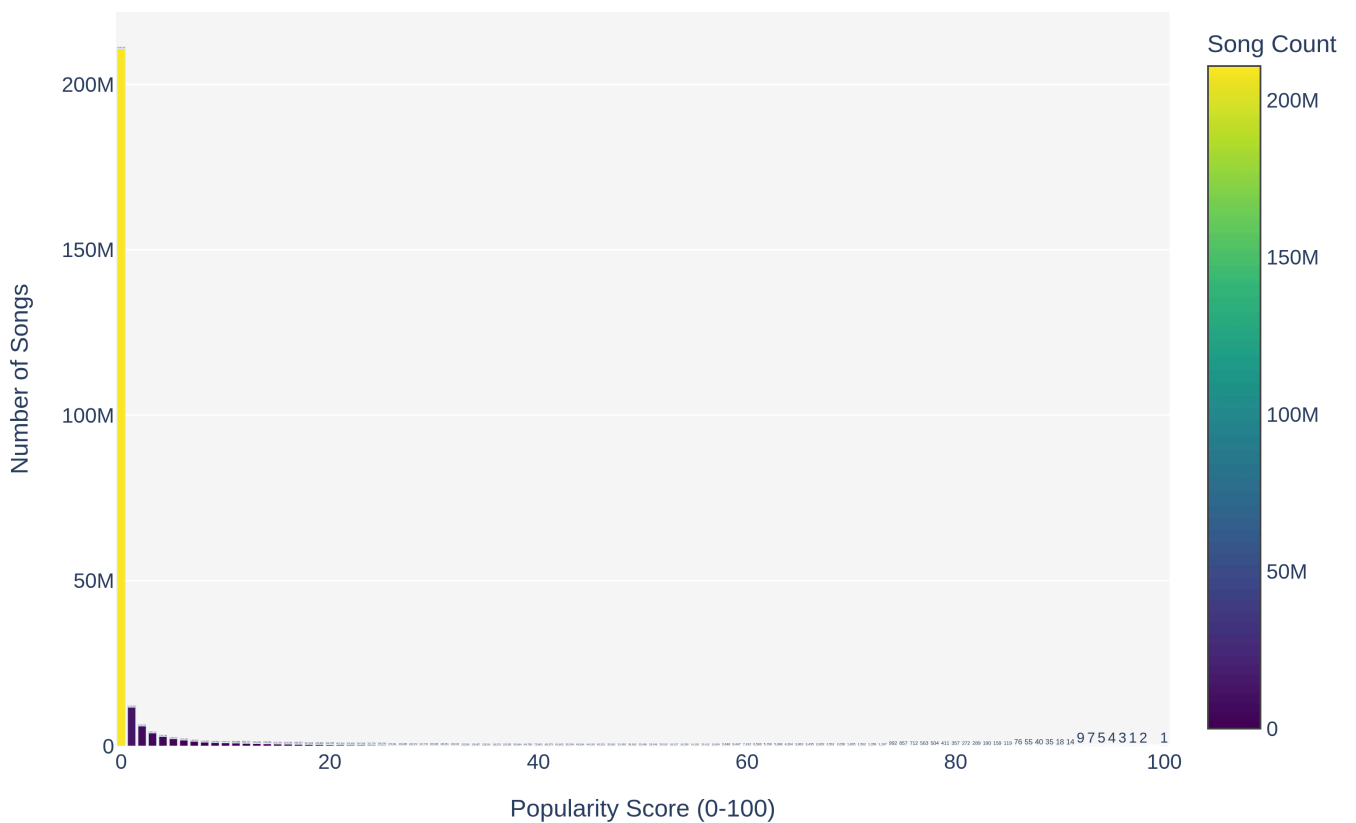


≥70% of songs are ones almost no one ever listens to (stream count < 1000). To see some detail, we can plot this on a logarithmic scale:

# ⭐ Songs by Popularity Score (Log Scale)



The top 10,000 songs span popularities 70-100. You can view them all in this HTML file (13.8MB gzipped).

Additionally, we can estimate the number of listens per track and total number per popularity. The stream count data is estimated since it is difficult to fetch at scale, so we sampled it randomly.

# 📊 Stream Count Analysis by Popularity

Total streams (bars) vs. Average per song (line). Stream counts are rough estimates.
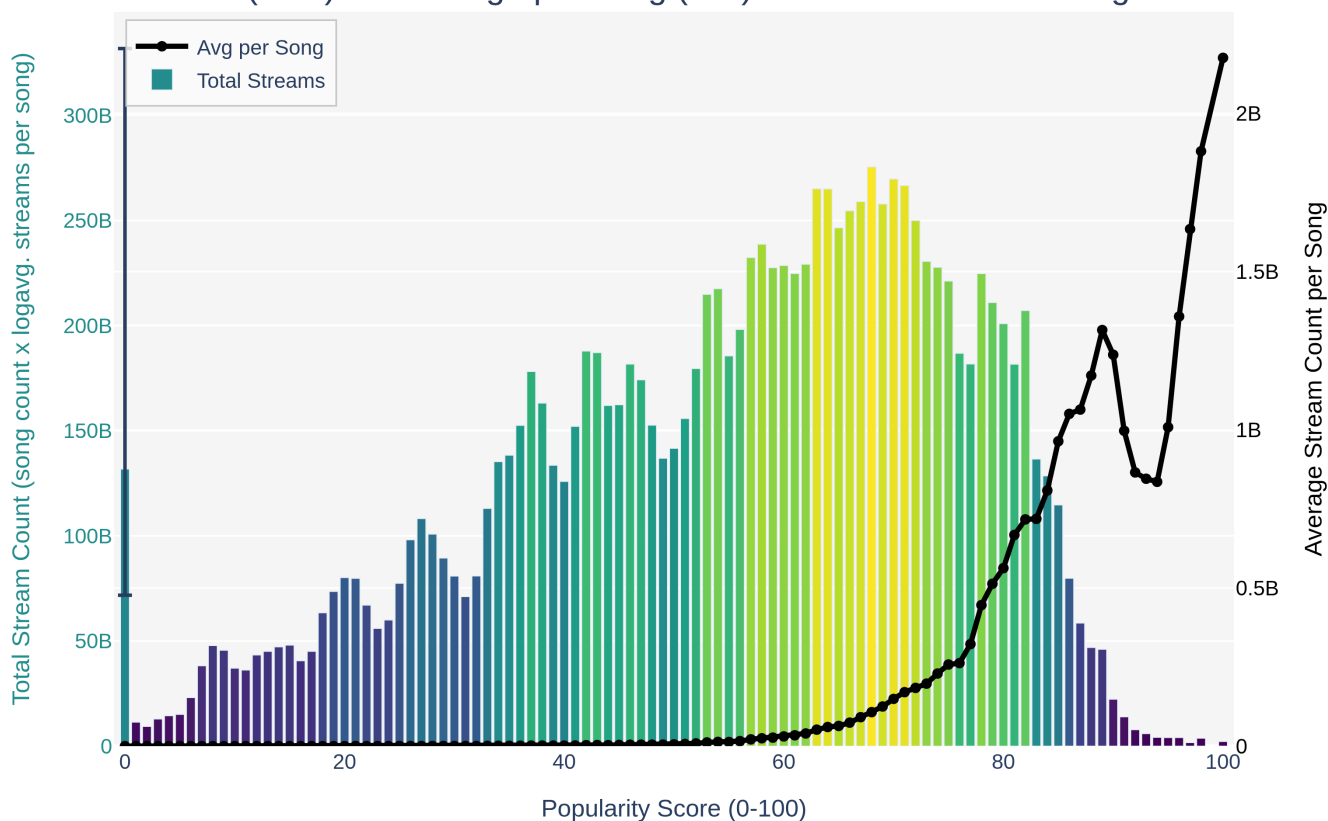


As we can see, most of the listens come from songs with a popularity between 50 and 80, even though there's only 210.000 songs with popularity ≥50, around 0.1% of songs. Note the huge (subjectively estimated) error bar on pop=0 — the reason for this is that Spotify does not publish stream counts for songs with < 1000 streams.

We can also estimate that the top three songs (as of writing) have a higher total stream count than the bottom **20-100 million songs combined**:

| Artists | Name | Popularity | Stream Count |
|---|---|---|---|
| Lady Gaga, Bruno Mars | Die With A Smile | 100 | 3.075 Billion |
| Billie Eilish | BIRDS OF A FEATHER | 98 | 3.137 Billion |
| Bad Bunny | DtMF | 98 | 1.124 Billion |

▶ SQLite Query

Note that the popularity is very time-dependent and not directly translatable into stream counts, so these top songs are basically arbitrary.

## Songs

We have archived around 86 million songs from Spotify, ordering by popularity descending. While this only represents 37% of songs, it represents around 99.6% of listens:

# Archived Tracks by Estimated Total Stream Count
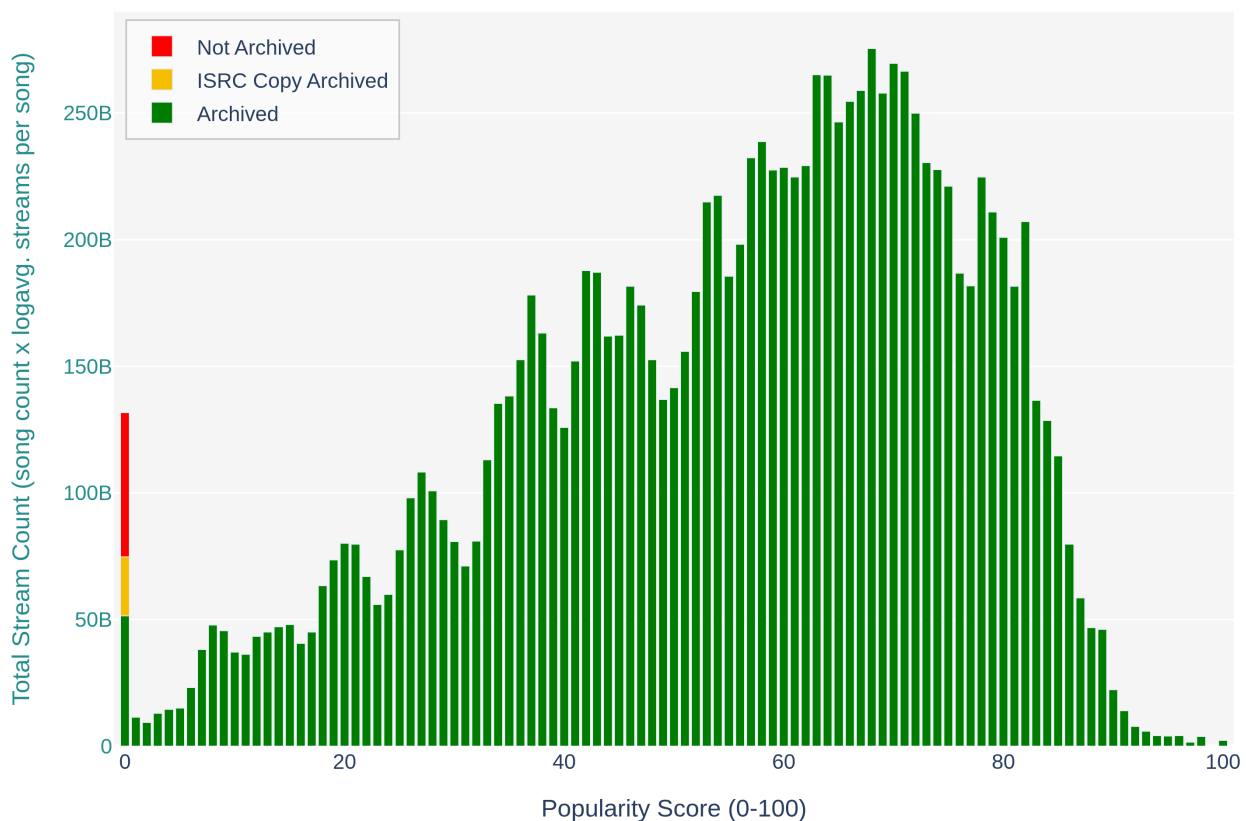


Put another way, for any random song a person listens to, there is a 99.6% likelihood that it is part of the archive. We expect this number to be higher if you filter to only human-created songs. Do remember though that the error bar on listens for popularity 0 is large.

For `popularity=0`, we ordered tracks by a secondary importance metric based on artist followers and album popularity, and fetched in descending order.

We have stopped here due to the long tail end with diminishing returns (700TB+ additional storage for minor benefit), as well as the bad quality of songs with `popularity=0` (many AI generated, hard to filter).

# Torrents

Before diving into more fun stats, let's look at how the collection itself is structured. It's in two parts: **metadata** and **music files**, both of which are distributed through torrents.

## Metadata

The metadata torrents contain, based on statistical analysis, around 99.9% of artists, albums, tracks. The metadata is published as compact queryable SQLite databases. Care was taken, by doing API response reconstruction, that there is (almost) no data loss in the conversion from the API JSON.

The metadata for artists, albums, tracks is less than 200 GB compressed. The secondary metadata of audio analysis is 4TB compressed.

We look at more detail at the structure of the metadata at the end of this blog post.

## Music Files

The data itself is distributed in the Anna's Archive Containers (AAC) format. This is a standard which we created a few years ago for distributing files across multiple torrents. It is not to be confused with the Advanced Audio Coding (AAC) encoding format.

Since the original files contain zero metadata, as much metadata as possible was added to the OGG files, including title, url, ISRC, UPC, album art, replaygain information, etc. The invalid OGG data packet Spotify prepends to every track file was stripped — it is present in the `track_files` db.

For `popularity>0`, the quality is the original OGG Vorbis at 160kbit/s. Metadata was added without reencoding the audio (and an archive of diff files is available to reconstruct the original files from Spotify).

For `popularity=0`, the audio is reencoded to OGG Opus at 75kbit/s — sounding the same to most people, but noticeable to an expert.

There is a known bug where the `REPLAYGAIN_ALBUM_PEAK` vorbiscomment tag value is a copy-paste of `REPLAYGAIN_ALBUM_GAIN` instead of the correct value for many files.

## The True Shuffle

Many people complain about how Spotify shuffles tracks. Since we have metadata for 99.9+% of tracks on Spotify, we can create a **true** shuffle across all songs on Spotify!

▶ Example True Shuffle Playlist
▶ Or, filtering to only somewhat popular songs

# More Stats

Here's some more statistics:

## Tracks

# ⏱ Tracks by Duration



Duration (hh:mm:ss)
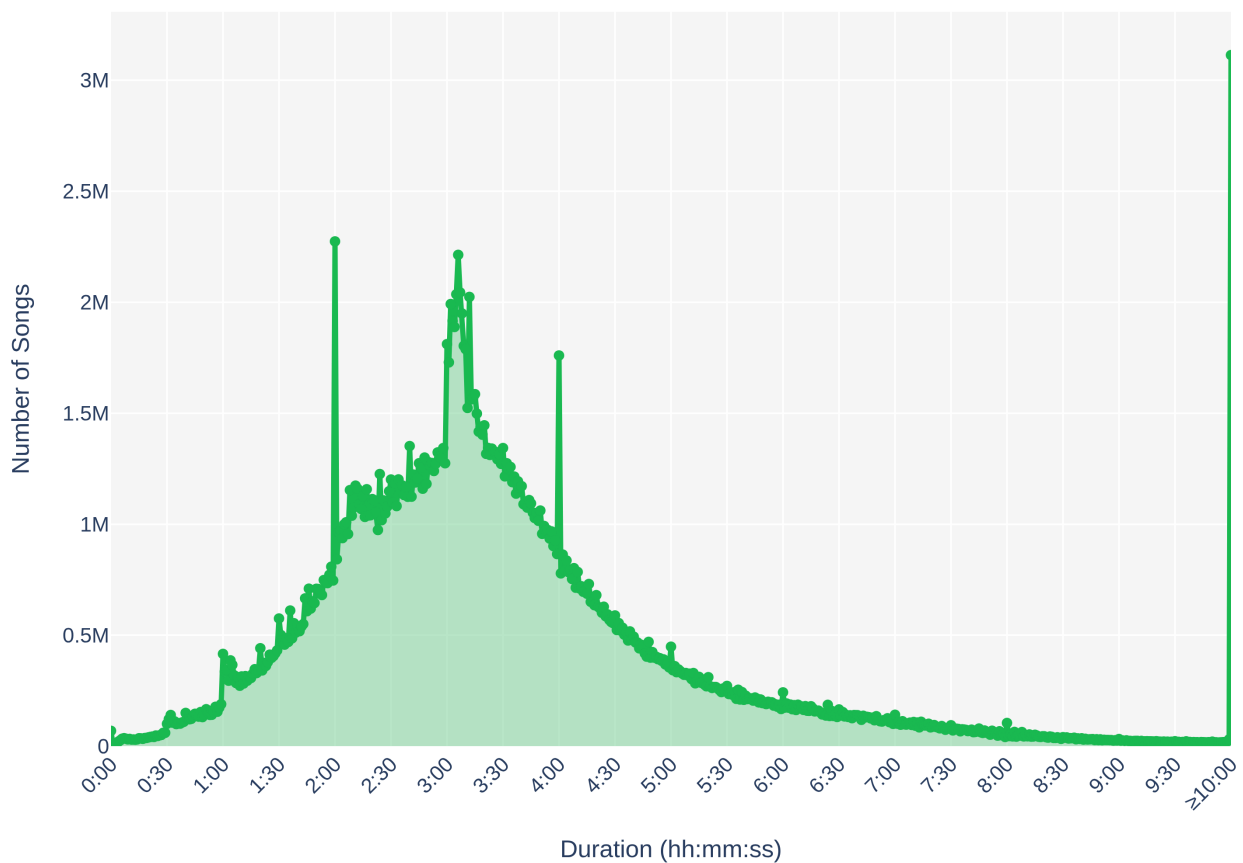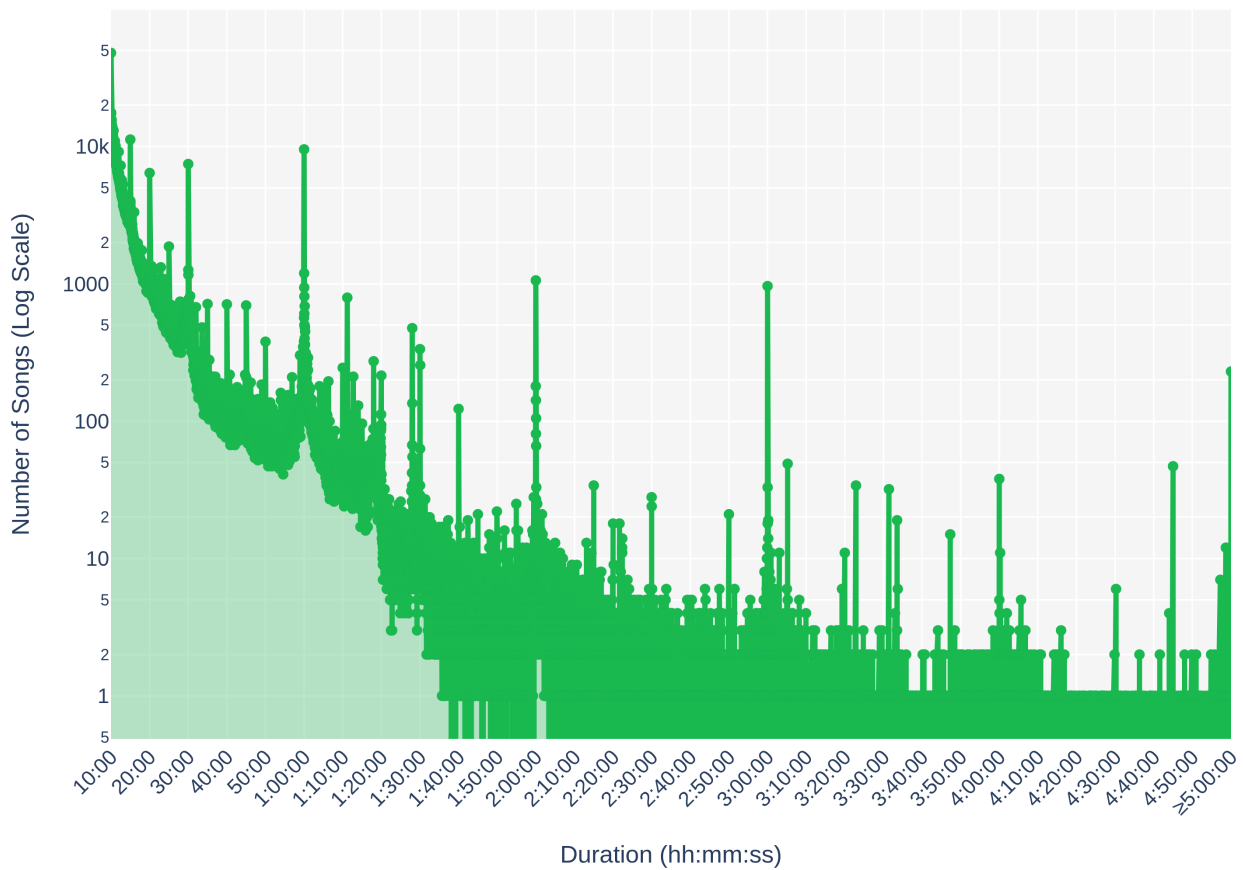
We're curious about the peaks at whole minutes (particularly 2:00, 3:00, 4:00). If you know why this is, please let us know!

# ⏱ Long Tracks by Duration (≥10 min, Log Scale)

# 🔞 Songs by Content Rating
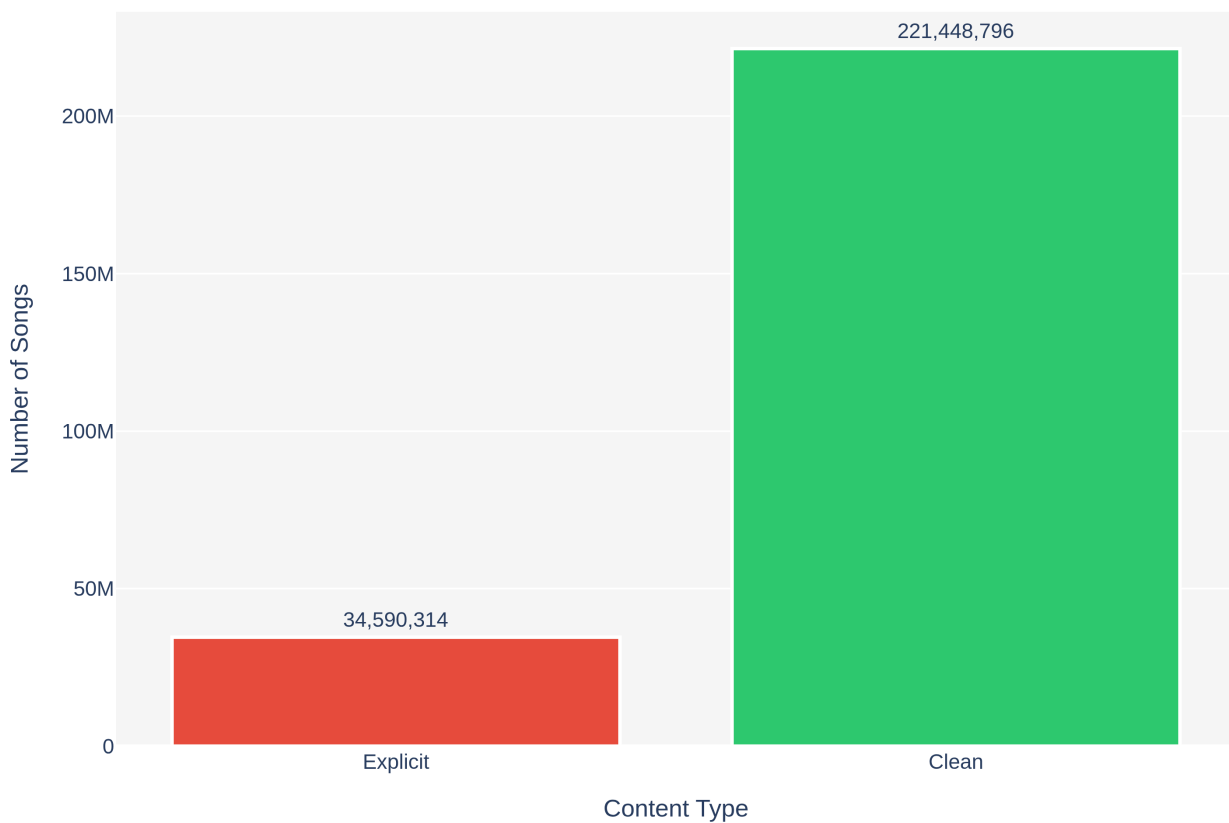


Some songs, especially popular songs, have 2, 3, or even 20 different versions. We can quantify this by counting number of songs per ISRC:

# 🆔 Duplicates per ISRC (Log Scale)

## Shows how many ISRCs have N track copies (same recording, different licensing/version)
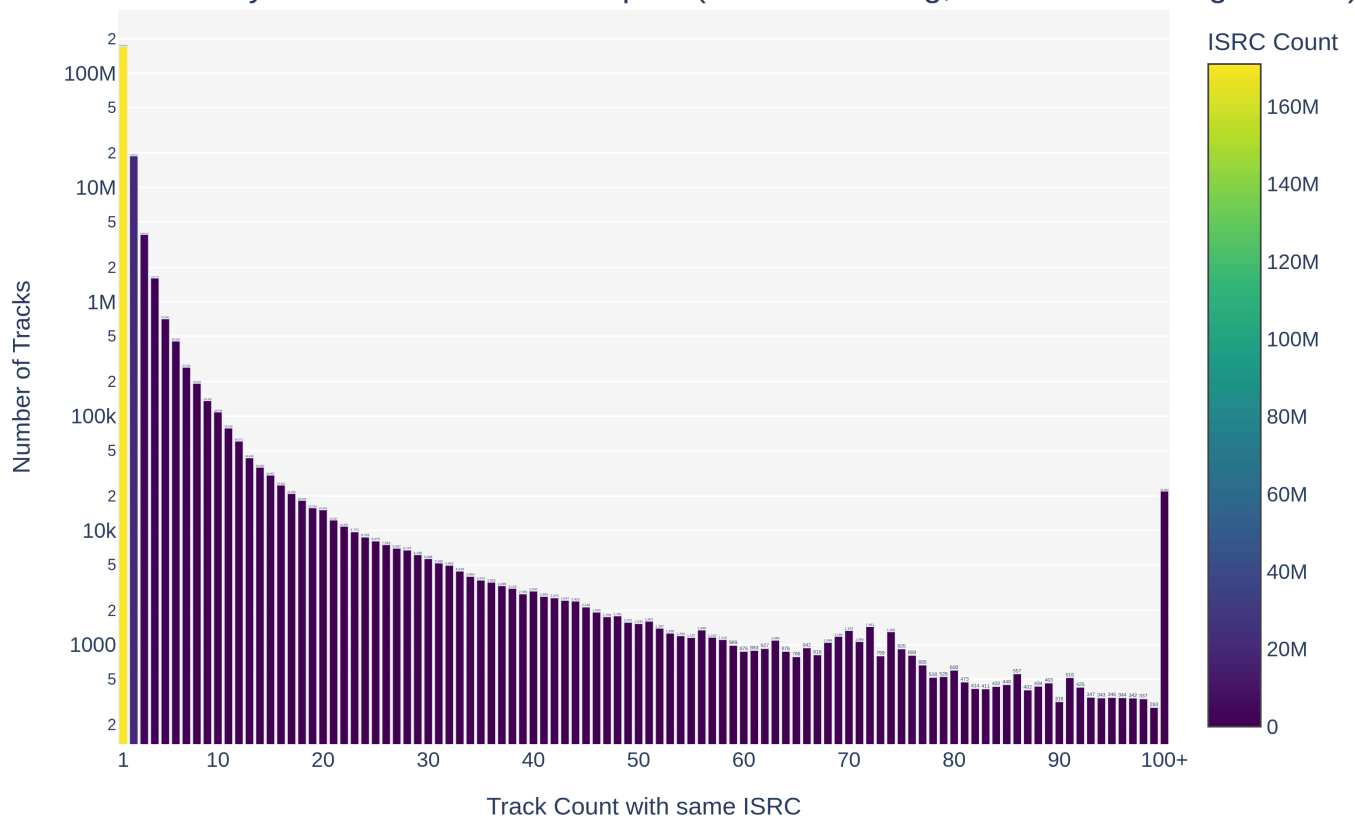


Each song on Spotify is only available on a specific set of markets. Most songs are available on most markets, but if we filter to popular songs, you can see a difference in availability (without filtering, the graph is almost flat):

# 🌍 Top 20 vs Bottom 20 Markets by Song Availability (Pop ≥ 10)
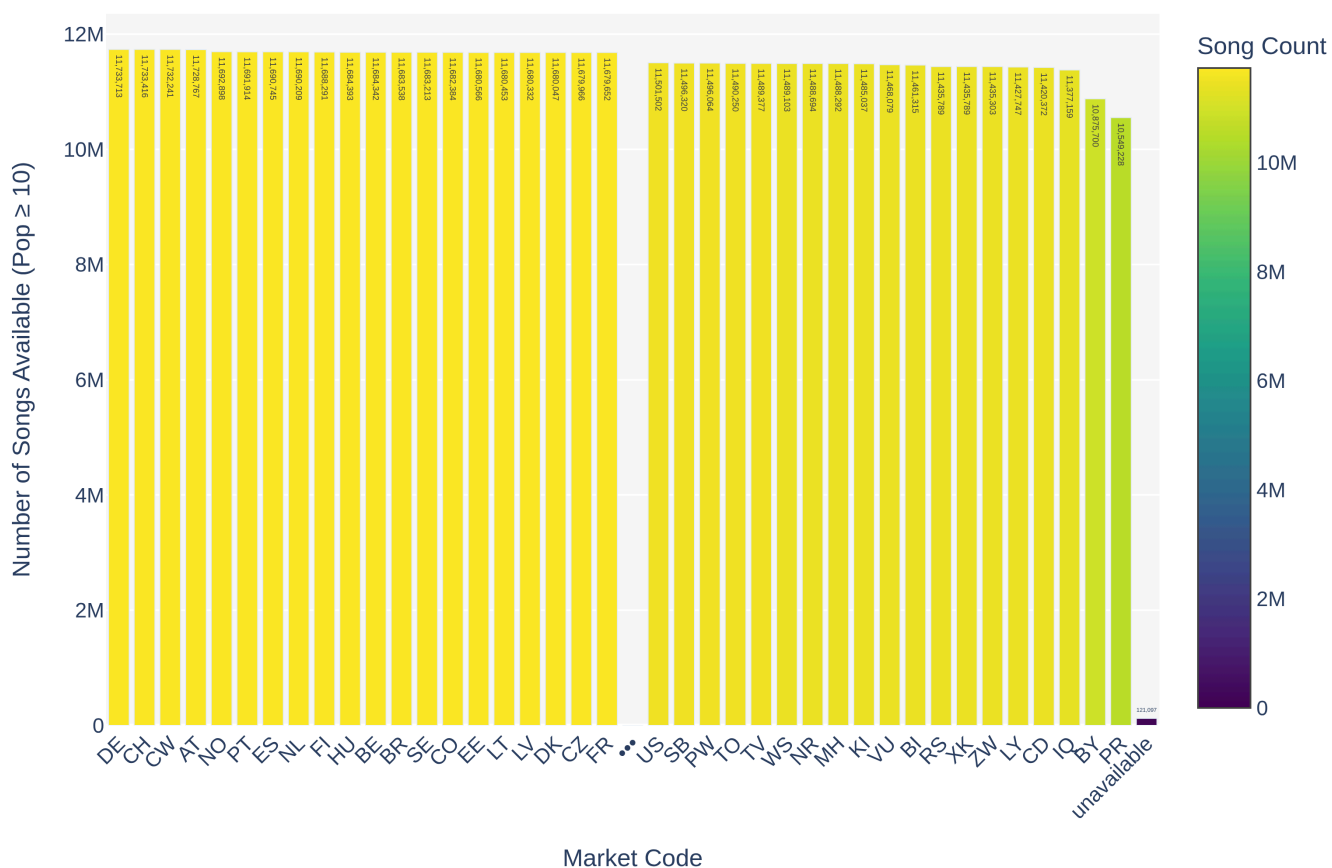


## Artists

Spotify provides a list of genres per artist (not per song). If we count the artists for each genre, we get this result:

# 🎵 Top 30 Music Genres by Artist Count

| Genre | Number of Artists |
|---|---|
| opera | 22822 |
| choral | 21188 |
| chamber music | 20581 |
| psytrance | 18352 |
| rockabilly | 14496 |
| black metal | 14376 |
| avant-garde | 13802 |
| bhajan | 13652 |
| traditional music | 13493 |
| dancehall | 12724 |
| riddim | 12289 |
| progressive house | 11866 |
| devotional | 11525 |
| afro house | 11444 |
| drum and bass | 11411 |
| breakbeat | 11326 |
| hardcore punk | 11259 |
| celtic | 10988 |
| experimental | 10977 |
| christian hip hop | 10835 |
| tech house | 10799 |
| amapiano | 10744 |
| hard techno | 10686 |
| death metal | 10604 |
| hardcore | 10585 |
| tango | 10472 |
| trance | 10307 |
| minimal techno | 10009 |
| hyperpop | 9766 |
| underground hip hop | 9704 |

Since each genre is very specific, we can also group genres and count the results:

# 🌟 Genre Hierarchy (Artists > 500)
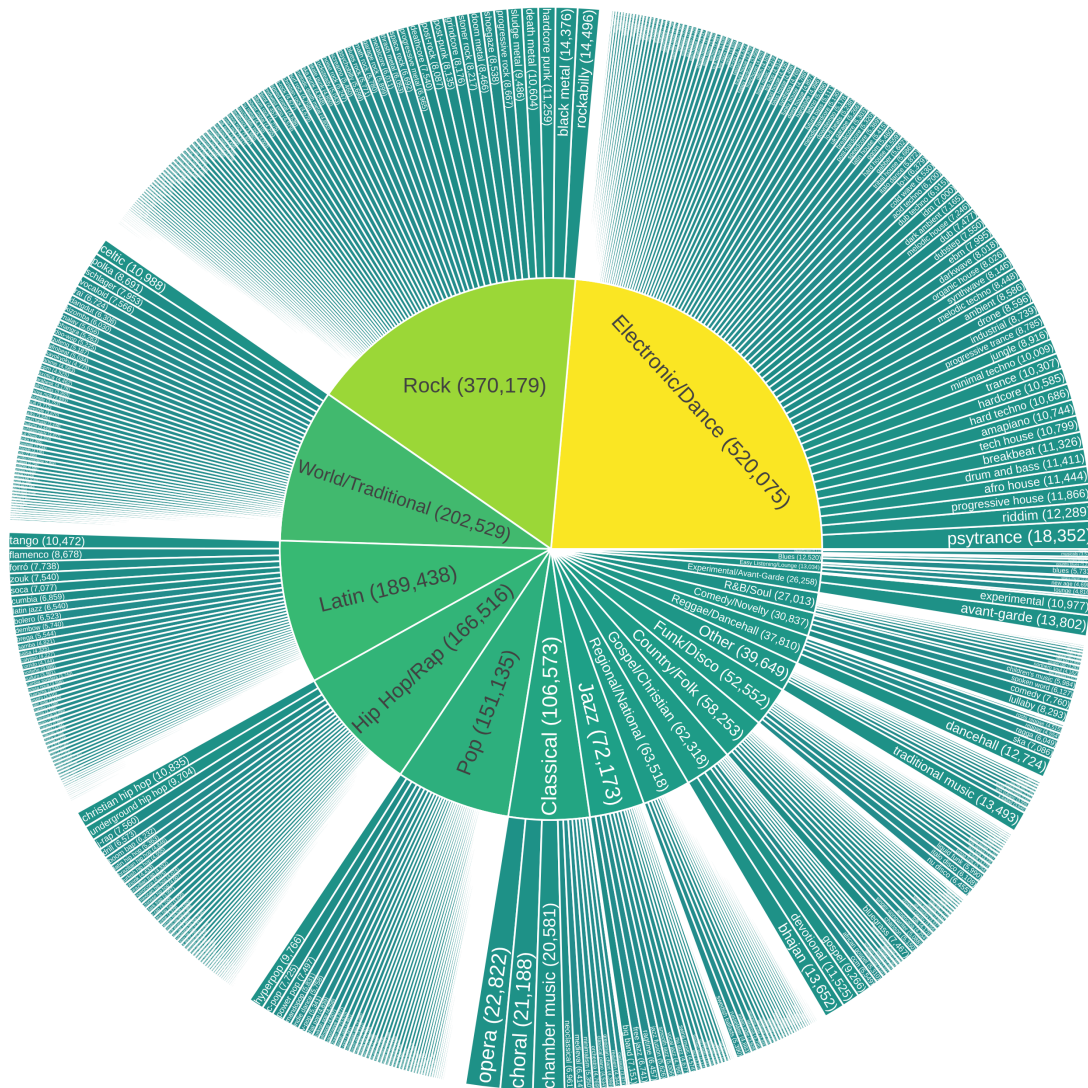


We can also group artists by popularity. The resulting graph looks very similar to the tracks popularity graph:

# ⭐ Artists by Popularity Score (Log Scale)
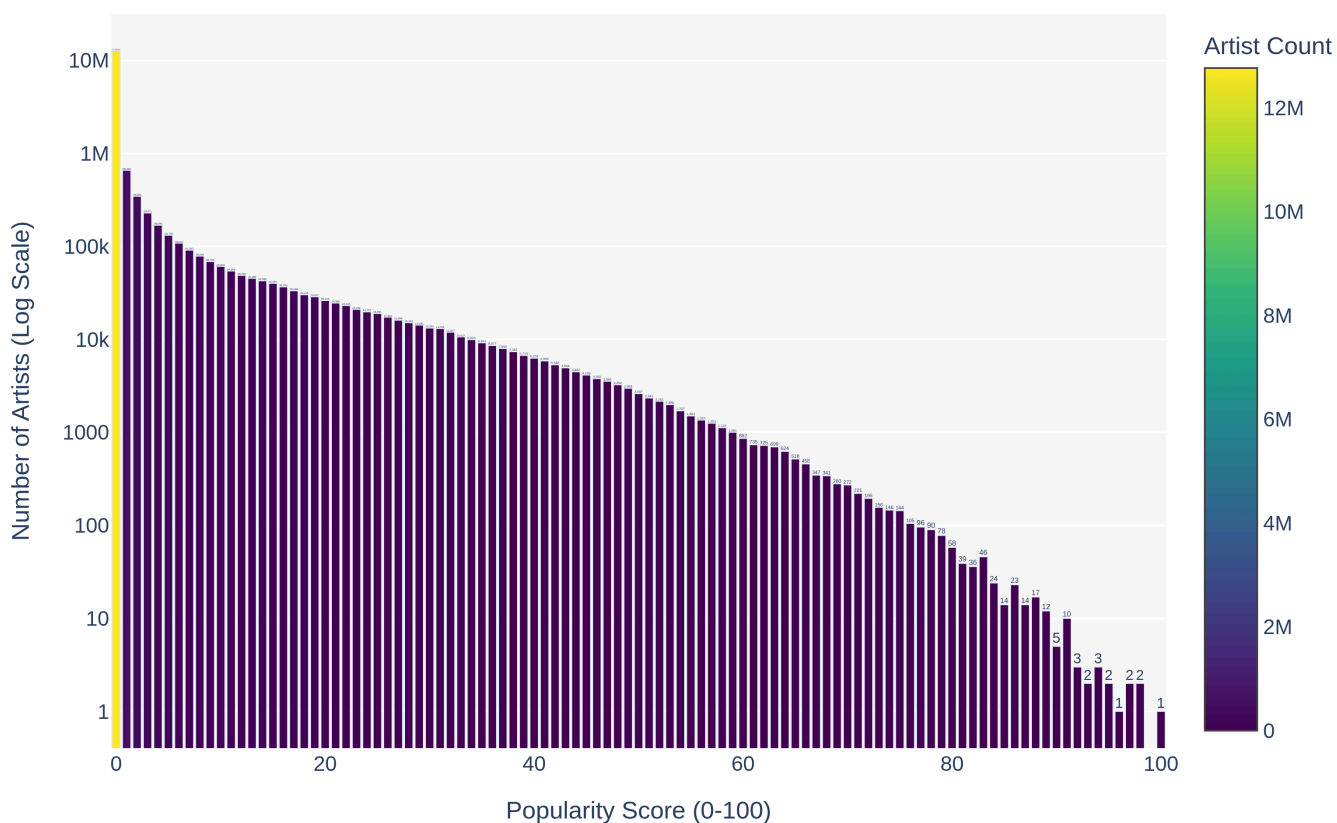


The same graph for albums also looks the same.

## Albums

If we group albums by release year, we see that more and more new music is added to Spotify, a lot of it likely automatically generated:

# 📅 Album Releases by Year
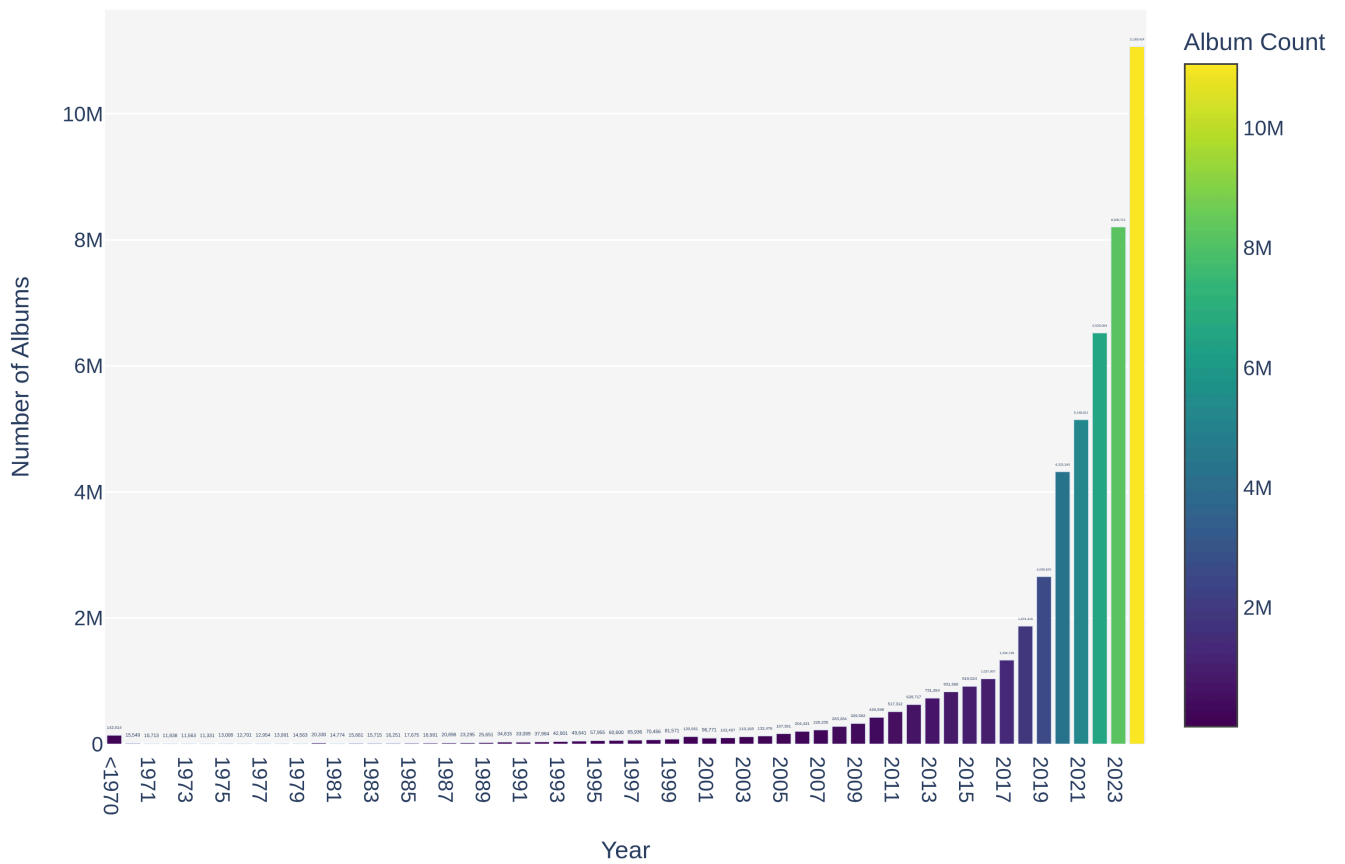


The amount of procedurally and AI generated content makes it hard to find what is actually valuable.

# 💿 Albums by Track Count (Log Scale)
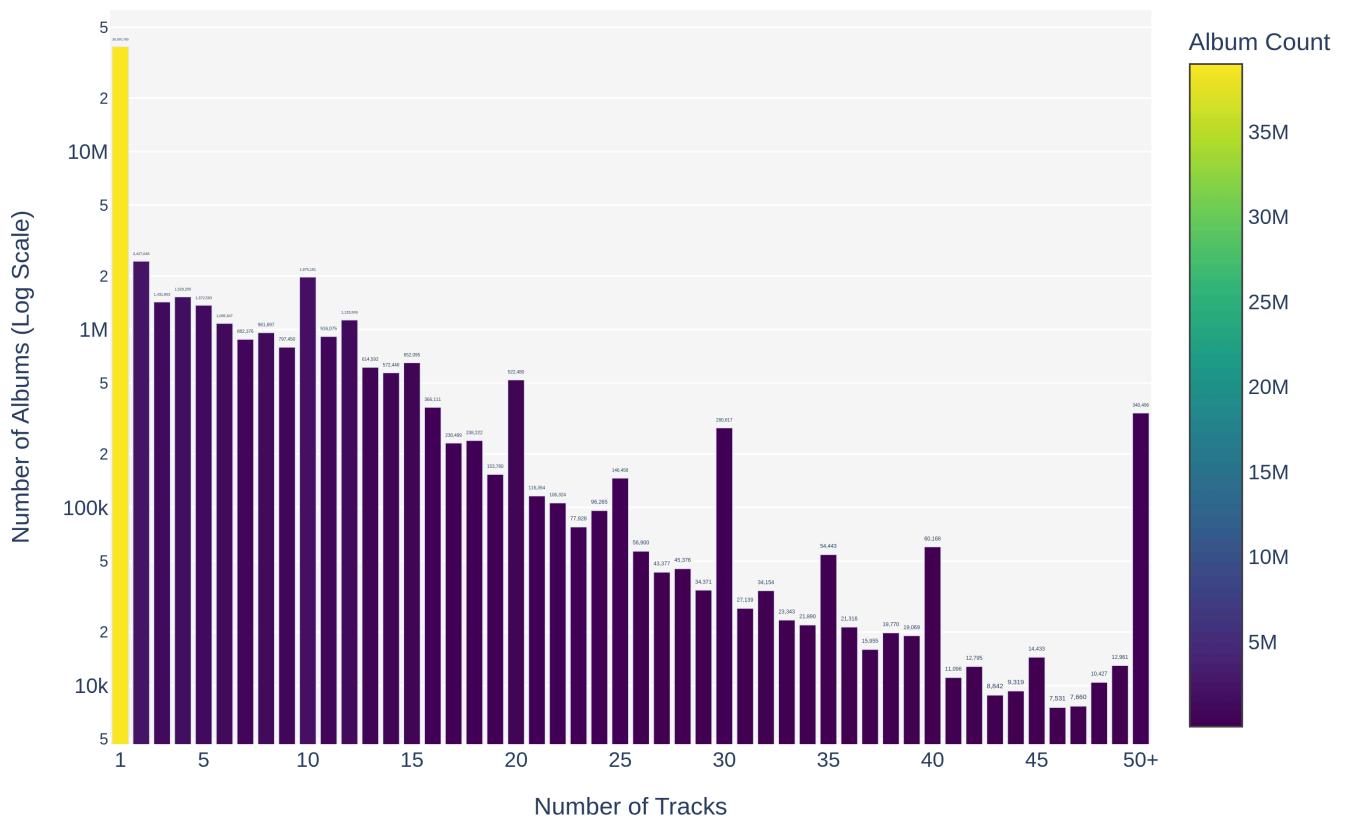


You can see that most songs on Spotify are singles, not part of an album.

# 🔢 Duplicate Albums per UPC (Log Scale)

## Shows how many UPCs have N duplicate albums (same album, different licensing/version)
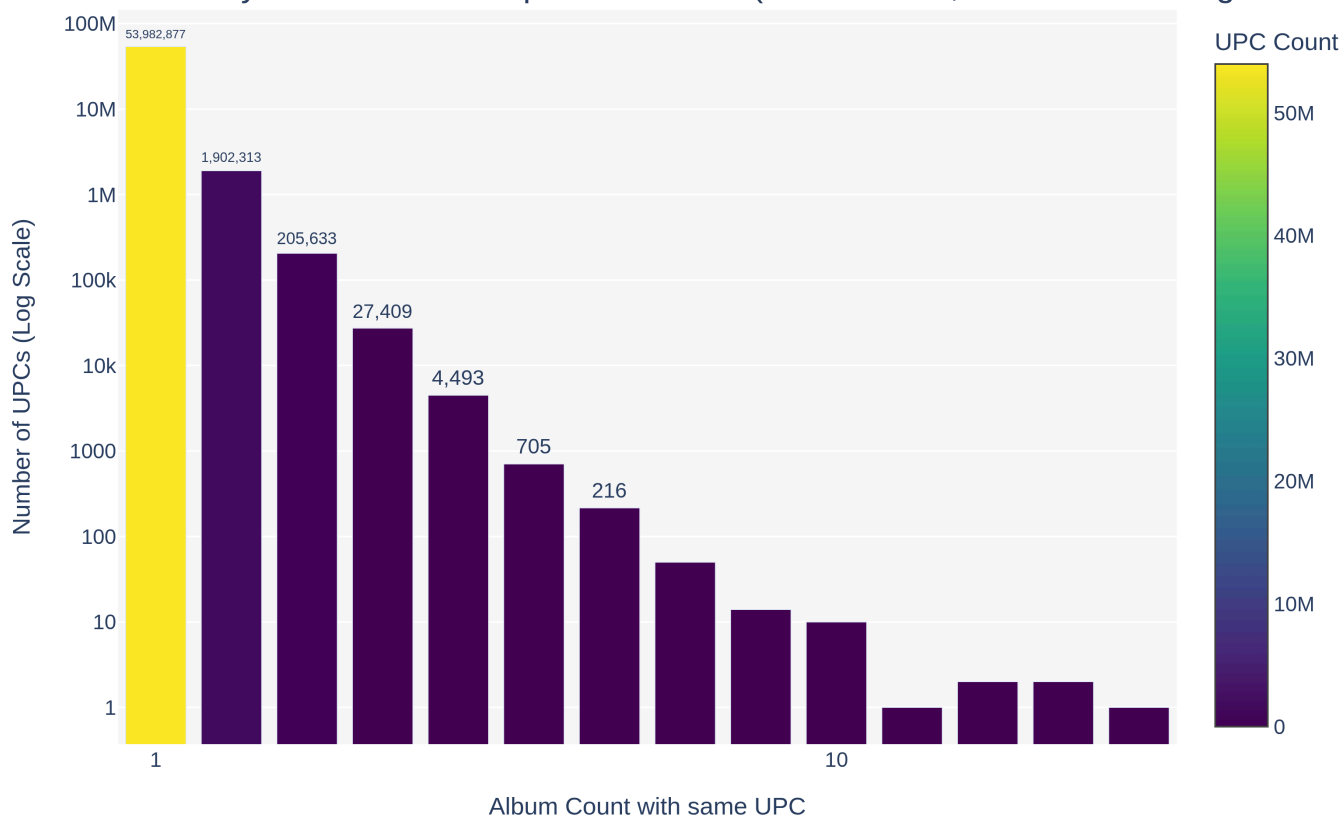


## Audio Features

We also scraped audio features generated by Spotify. They can be analyzed to find interesting trends.

# 🎼🎹 Mode Distribution by Musical Key
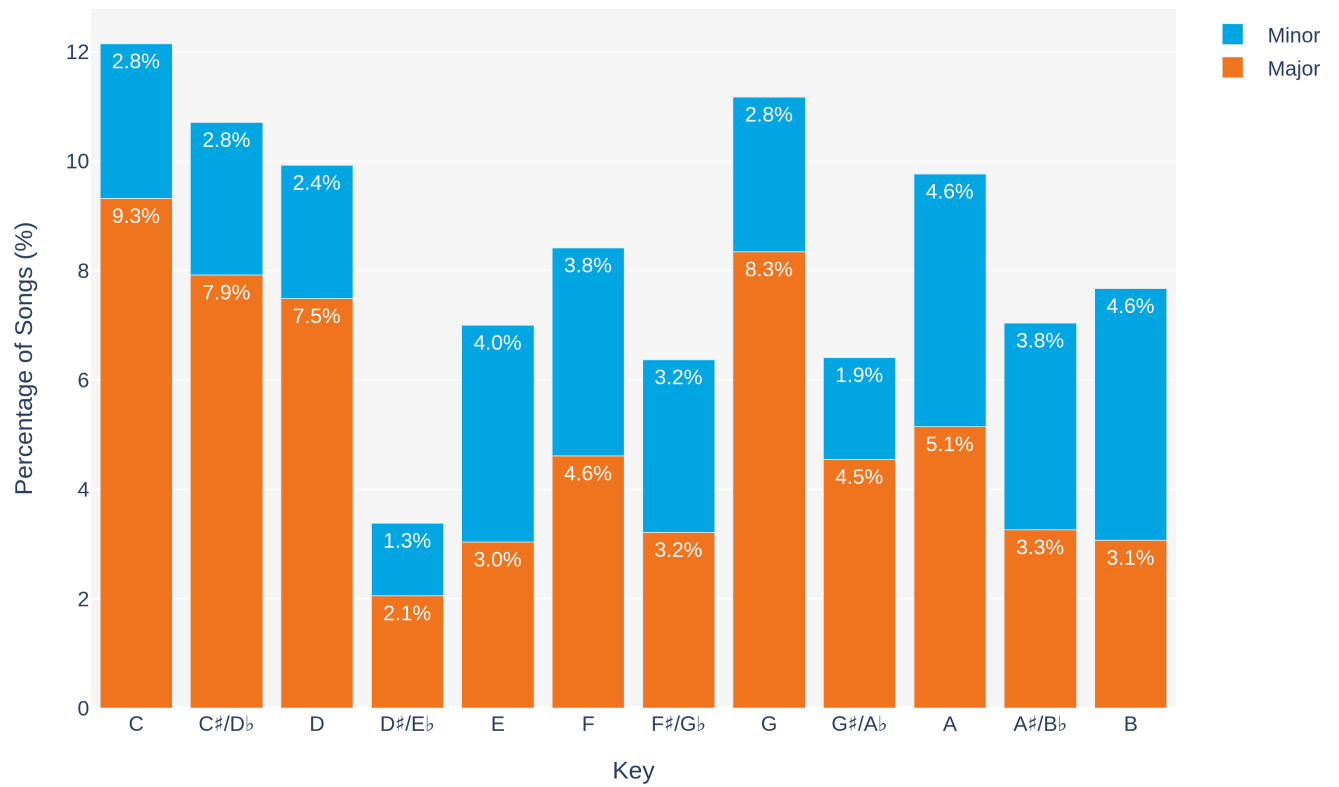
## 🖼️ Audio Features Correlation Heatmaps

2D density distributions showing the correlation between each audio feature. Heatmaps use log scale.

The diagonals show a histogram of the feature.
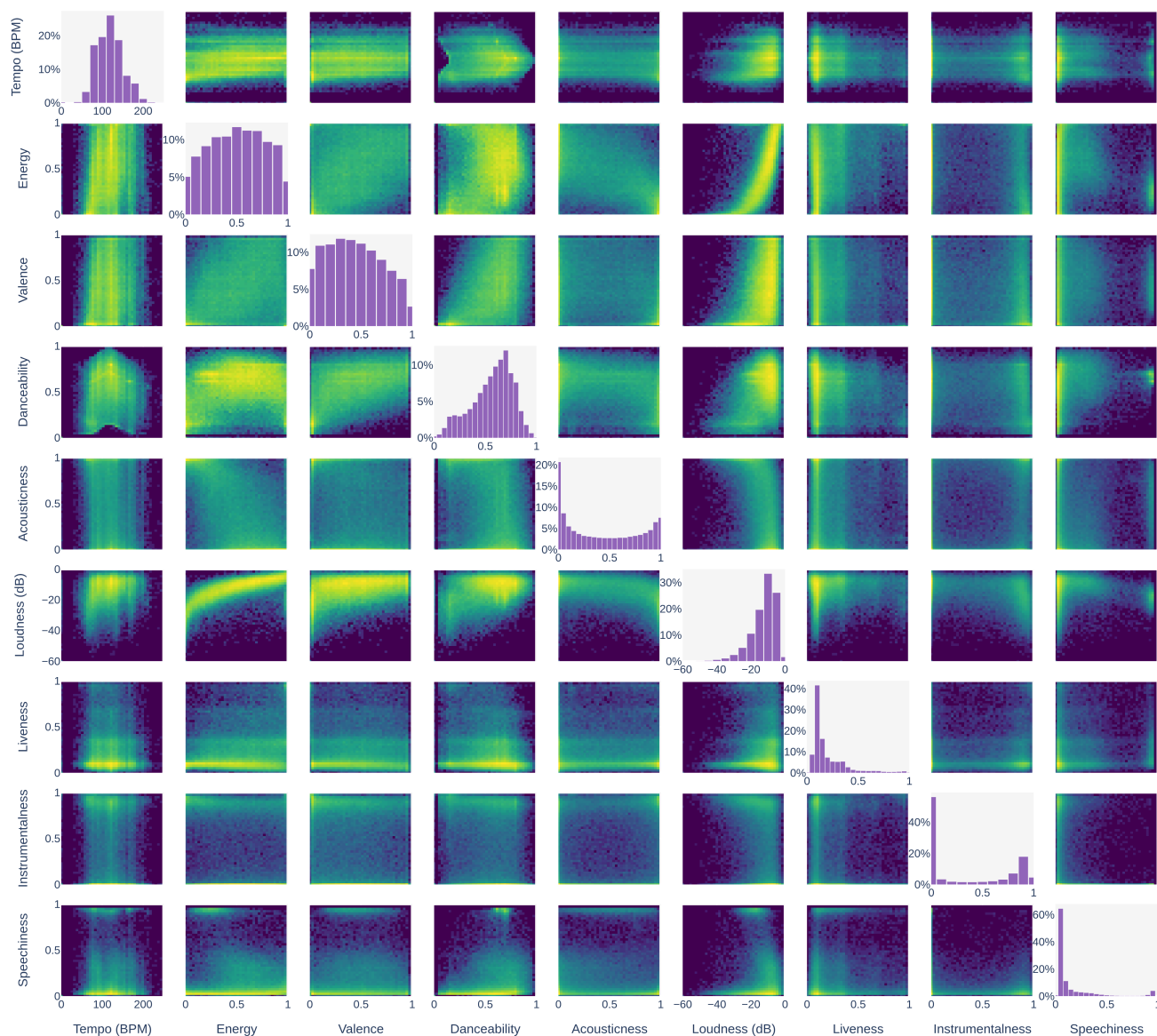


This chart contains a lot of information. You can for example see that loudness correlates with energy and that BPM is normally distributed with a mean around 120.

# Metadata Files

`annas_archive_spotify_2025_07_metadata.torrent/spotify_clean.sqlite3`

A scrape of the three main APIs of Spotify: artists, albums, tracks. Each track exists in exactly one album, but each track and each album can have multiple artists.

The tables are an almost lossless representation of Spotify API JSON responses, care was taken during creation by always reconstructing the original JSON based on each inserted row, with minor exceptions.

```
CREATE TABLE `artists` (
        `rowid` integer PRIMARY KEY NOT NULL,
        /* The original Spotify base62 ID. */
        `id` text NOT NULL,
        /* When the item was fetched (unixepoch ms). */
        `fetched_at` integer NOT NULL,
        /* "The name of the artist."*/
        `name` text NOT NULL,
        /* followers.total - "The total number of followers." */
        `followers_total` integer NOT NULL,
        /* "The popularity of the artist. The value will be between 0 and 100, with
100 being the most popular. The artist's popularity is calculated from the popularity
of all the artist's tracks." */
        `popularity` integer NOT NULL
);
/* "A list of the genres the artist is associated with. If not yet classified, the
array is empty." */
CREATE TABLE `artist_genres` (
        `artist_rowid` integer NOT NULL,
        `genre` text NOT NULL,
        FOREIGN KEY (`artist_rowid`) REFERENCES `artists`(`rowid`)
);
/* Images of the artist in various sizes, widest first. */
CREATE TABLE `artist_images` (
        `artist_rowid` integer NOT NULL,
        `width` integer NOT NULL,
        `height` integer NOT NULL,
        `url` text NOT NULL,
        FOREIGN KEY (`artist_rowid`) REFERENCES `artists`(`rowid`)
);
/*
 * Information about artist-albums relationships from /artists/{id}/albums,
album.artists[] and album.tracks[].artists[].
 * The relationships "album", "single", "compilation" are left out because they can
be reconstructed from `album.type`.
 */
CREATE TABLE "artist_albums" (
        `artist_rowid` integer NOT NULL,
        `album_rowid` integer NOT NULL,
        /* True if this link was retrieved from /artists/{id}/albums with an
"album_group" response of "appears_on". */
        `is_appears_on` integer NOT NULL,
        /* True if this link is based on the actual artists of each track in the
album. Only exists if the link is not explicit (above). */
        `is_implicit_appears_on` integer NOT NULL,
        /* If neither is_appears_on or is_implicit_appears_on, the index of
album.data.artists[] this was retrieved from. */
        `index_in_album` integer,
        FOREIGN KEY (`artist_rowid`) REFERENCES `artists`(`rowid`),
        FOREIGN KEY (`album_rowid`) REFERENCES `albums`(`rowid`)
);
/* combinations of available markets in a separate table to save space */
```

```
    CREATE TABLE `available_markets` (
            `rowid` integer PRIMARY KEY NOT NULL,
            /* comma separated ISO 3166-1 alpha-2 country codes. */
            `available_markets` text NOT NULL
    );
    /* /albums/{id} - "Get Spotify catalog information for a single album." */
    CREATE TABLE `albums` (
            `rowid` integer PRIMARY KEY NOT NULL,
            /* The original Spotify base62 ID. */
            `id` text NOT NULL,
            /* When the item was fetched (unixepoch ms). */
            `fetched_at` integer NOT NULL,
            /* "The name of the album. In case of an album takedown, the value may be an
empty string." */
            `name` text NOT NULL,
            /* 'The type of the album. Allowed values: "album", "single", "compilation"'
*/
            `album_type` text NOT NULL,
            /* available markets as an index into the available_markets table to save
space. - "The markets in which the album is available: ISO 3166-1 alpha-2 country
codes. NOTE: an album is considered available in a market when at least 1 of its tracks
is available in that market." */
            `available_markets_rowid` integer NOT NULL,
            /* external_id.upc - Universal Product Code */
            `external_id_upc` text,
            /* external_id.amgid - undocumented - AMG MUSIC GROUP Internal ID */
            "external_id_amgid" text,
            /* "The copyright" */
            `copyright_c` text,
            /* "The sound recording (performance) copyright." */
            `copyright_p` text,
            /* "The label associated with the album." */
            `label` text NOT NULL,
            /* "The popularity of the album. The value will be between 0 and 100, with
100 being the most popular." */
            `popularity` integer NOT NULL,
            /* "The date the album was first released." */
            `release_date` text NOT NULL,
            /* 'The precision with which release_date value is known. Allowed values:
"year", "month", "day"' */
            `release_date_precision` text NOT NULL,
            /* tracks.total */
            `total_tracks` integer NOT NULL,
            FOREIGN KEY (`available_markets_rowid`) REFERENCES
`available_markets`(`rowid`)
    );

    /* album.images[] - "The cover art for the album in various sizes, widest first." */
    CREATE TABLE "album_images" (
            `album_rowid` integer NOT NULL,
            `width` integer NOT NULL,
```

```
        height  integer NOT NULL,
        `url` text NOT NULL,
        FOREIGN KEY (`album_rowid`) REFERENCES `albums`(`rowid`)
    );
    /* /tracks/{id} */
    CREATE TABLE `tracks` (
        `rowid` integer PRIMARY KEY NOT NULL,
        /* The original Spotify base62 ID. */
        `id` text NOT NULL,
        /* When the item was fetched (unixepoch ms). */
        `fetched_at` integer NOT NULL,
        /* "The name of the track." */
        `name` text NOT NULL,
        /* "A link to a 30 second preview (MP3 format) of the track. Can be null" */
        `preview_url` text,
        `album_rowid` integer NOT NULL,
        /* "The number of the track. If an album has several discs, the track number
is the number on the specified disc." */
        `track_number` integer NOT NULL,
        /* http://en.wikipedia.org/wiki/International_Standard_Recording_Code */
        `external_id_isrc` text,
        `external_id_ean` text,
        `external_id_upc` text,
        /* "The popularity of the track. The value will be between 0 and 100, with
100 being the most popular.

  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." */
        `popularity` integer NOT NULL,
        /* A reference into the available_markets table. - "A list of the countries
in which the track can be played, identified by their ISO 3166-1 alpha-2 code." */
        `available_markets_rowid` integer NOT NULL,
        /* "The disc number (usually 1 unless the album consists of more than one
disc)." */
        `disc_number` integer NOT NULL,
        /* "The track length in milliseconds." */
        `duration_ms` integer NOT NULL,
        /* "Whether or not the track has explicit lyrics ( true = yes it does; false
= no it does not OR unknown)." */
        `explicit` integer NOT NULL,
        FOREIGN KEY (`available_markets_rowid`) REFERENCES
`available_markets`(`rowid`)
    );
    /* "The artists who performed the track." */
    CREATE TABLE `track_artists` (
        `track_rowid` integer NOT NULL,
        `artist_rowid` integer NOT NULL,
```

```
        FOREIGN KEY (`track_rowid`) REFERENCES `tracks`(`rowid`),
        FOREIGN KEY (`artist_rowid`) REFERENCES `artists`(`rowid`)
    );

    CREATE INDEX `artist_genres_artist_id` ON `artist_genres` (`artist_rowid`);
    CREATE INDEX `artist_genres_genre` ON `artist_genres` (`genre`);
    CREATE INDEX `artist_images_artist_id` ON `artist_images` (`artist_rowid`);
    CREATE UNIQUE INDEX `artists_id_unique` ON `artists` (`id`);
    CREATE INDEX `artists_name` ON `artists` (`name`);
    CREATE INDEX `artists_popularity` ON `artists` (`popularity`);
    CREATE INDEX `artists_followers` ON `artists` (`followers_total`);
    CREATE INDEX `artist_album_artist_id` ON "artist_albums" (`artist_rowid`);
    CREATE INDEX `artist_album_album_id` ON "artist_albums" (`album_rowid`);
    CREATE UNIQUE INDEX `albums_id_unique` ON `albums` (`id`);
    CREATE INDEX `album_name` ON `albums` (`name`);
    CREATE INDEX `album_popularity` ON `albums` (`popularity`);
    CREATE UNIQUE INDEX `available_markets_available_markets_unique` ON
`available_markets` (`available_markets`);
    CREATE INDEX `track_artists_artist_id` ON `track_artists` (`artist_rowid`);
    CREATE INDEX `track_artists_track_id` ON `track_artists` (`track_rowid`);
    CREATE UNIQUE INDEX `tracks_id_unique` ON `tracks` (`id`);
    CREATE INDEX `tracks_popularity` ON `tracks` (`popularity`);
    CREATE INDEX `tracks_album` ON `tracks` (`album_rowid`);
    CREATE INDEX `album_images_album_id` ON `album_images` (`album_rowid`);
    CREATE INDEX tracks_isrc on tracks(external_id_isrc);
```
**annas_archive_spotify_2025_07_metadata.torrent/spotify_clean_audio_features.sqlite3**

A scrape of `AudioFeatures` objects from the Spotify API. One row per track.

```
/* /audio-features/{id} "Get audio feature information for a single track identified by
its unique Spotify ID." */
  CREATE TABLE `track_audio_features` (
          `rowid` integer PRIMARY KEY NOT NULL,
          `track_id` text NOT NULL,
          `fetched_at` integer NOT NULL,
          /* true if the API returned null for the whole request */
          `null_response` integer NOT NULL,
          /* "The duration of the track in milliseconds." */
          `duration_ms` integer,
          /* 'An estimated time signature. The time signature (meter) is a notational
convention to specify how many beats are in each bar (or measure). The time signature
ranges from 3 to 7 indicating time signatures of "3/4", to "7/4".' */
          `time_signature` integer,
          /* "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." */
          `tempo` 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." */
          `key` 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." */
          `mode` integer,
          /* "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." */
          `danceability` real,
          /* "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." */
          `energy` real,
          /* "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." */
          `loudness` real,
          /* "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."
*/
          `speechiness` real,
          /* "A confidence measure from 0.0 to 1.0 of whether the track is acoustic.
1.0 represents high confidence the track is acoustic." */
          `acousticness` real,
```

```
          /* "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." */
          `instrumentalness` real,
          /* "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." */
          `liveness` real,
          /* "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)." */
          `valence` real
  );
  CREATE UNIQUE INDEX `track_audio_features_track_id_unique` ON `track_audio_features`
(`track_id`);
```

**annas_archive_spotify_2025_07_metadata.torrent/spotify_clean_playlists.sqlite3**

A scrape of `Playlist` objects from the Spotify API. Requires the `spotify_clean.sqlite3` database to map track_rowid to track_id.

Most playlists with < 1000 followers were excluded. Completeness unknown.

Row count: 6.6 million playlists with 1.7 billion playlist tracks.

```sql
/* /get-playlist/{id} - "Get a playlist owned by a Spotify user." */
  CREATE TABLE "playlists" (
          `rowid` integer PRIMARY KEY NOT NULL,
          /* the original spotify base62 ID */
          `id` text NOT NULL,
          /* the spotify snapshot ID that was fetched (we only store one copy of each
playlist) - "The version identifier for the current playlist. Can be supplied in other
requests to target a specific playlist version" */
          `snapshot_id` text NOT NULL,
          /* When the playlist was fetched (unixepoch ms). */
          `fetched_at` integer NOT NULL,
          /* "The name of the playlist." */
          `name` text NOT NULL,
          /* "The playlist description. Only returned for modified, verified playlists,
otherwise null. */
          `description` text,
          /* "true if the owner allows other users to modify the playlist." */
          `collaborative` integer NOT NULL,
          -- "The playlist's public/private status (if it is added to the user's
profile): true the playlist is public, false the playlist is private, null the playlist
status is not relevant. For more about public/private status, see Working with
Playlists"
          `public` integer NOT NULL,
          `primary_color` text,
          /* owner.id - "The unique string identifying the Spotify user that you can
find at the end of the Spotify URI for the user. The ID of the current user can be
obtained via the Get Current User's Profile endpoint." */
          `owner_id` text,
          /* owner.display_name - "The name displayed on the user's profile. null if
not available." */
          `owner_display_name` text,
          /* followers.total */
          `followers_total` integer,
          `tracks_total` integer NOT NULL
  );
  CREATE UNIQUE INDEX `playlists_id_unique` ON `playlists` (`id`);
  CREATE INDEX `playlists_name` ON `playlists` (`name`);
  CREATE INDEX `playlists_owner_id` ON `playlists` (`owner_id`);
  CREATE INDEX `playlists_snapshot_id` ON `playlists` (`snapshot_id`);
  CREATE INDEX `playlists_followers` ON `playlists` (`followers_total`);


  /* "Images for the playlist. The array may be empty or contain up to three images.
The images are returned by size in descending order. See Working with Playlists. Note:
If returned, the source URL for the image (url) is temporary and will expire in less
than a day." */
  CREATE TABLE `playlist_images` (
          `playlist_rowid` integer NOT NULL,
          `width` integer,
          `height` integer,
          `url` text NOT NULL,
          FOREIGN KEY (`playlist_rowid`) REFERENCES `playlists`(`rowid`)
```

```
    );
    CREATE INDEX `playlist_images_playlist_id` ON `playlist_images` (`playlist_rowid`);


    /* The tracks of the playlist. Merged from all pages of getPlaylistItems. */
    CREATE TABLE "playlist_tracks" (
            `playlist_rowid` integer NOT NULL,
            /* 0-based integer position within the playlists response
(playlist.tracks.items[i]) */
            `position` integer NOT NULL,
            /* true if track.type == "episode". false if track.type == "track" */
            `is_episode` integer NOT NULL,
            /* The rowid of this track in the `tracks` table. */
            `track_rowid` integer,
            /* If the rowid is null, this is the spotify base62 ID instead. */
            `id_if_not_in_tracks_table` text,
            /* (unixepoch seconds) - "The date and time the track or episode was added.
Note: some very old playlists may return null in this field." */
            `added_at` integer NOT NULL,
            /* added_by.id - "The Spotify user who added the track or episode. Note: some
very old playlists may return null in this field." */
            `added_by_id` text,
            /* undocumented */
            `primary_color` text,
            /* video_thumbnail.url - undocumented */
            `video_thumbnail_url` text,
            /* "Whether this track or episode is a local file or not."
https://developer.spotify.com/documentation/web-api/concepts/playlists#local-files */
            `is_local` integer NOT NULL,
            /* track.name if is_local is true. For non-local tracks, name can be
retrieved from the `tracks` table. */
            `name_if_is_local` text,
            /* track.uri if is_local is true */
            `uri_if_is_local` text,
            /* track.album if is_local is true. Spotify constructs a fake album object
for local tracks. For non-local tracks, album can be retrieved from the `albums` table.
*/
            `album_name_if_is_local` text,
            /* track.artists[0].name if is_local is true. Spotify constructs a single
fake artists object for local tracks. For non-local tracks, artist name can be
retrieved from the `artists` table. */
            `artists_name_if_is_local` text,
            /* track.duration_ms if is_local is true. For non-local tracks, duration_ms
can be retrieved from the `tracks` table. */
            `duration_ms_if_is_local` integer,
            PRIMARY KEY(`playlist_rowid`, `position`),
            FOREIGN KEY (`playlist_rowid`) REFERENCES `playlists`(`rowid`)
    ) WITHOUT ROWID;
```

This is an almost lossless representation of the original Spotify API response. JSON
reconstruction was tested during creation of all tables, with minor exceptions. For example, the
original response JSON for playlists can be reconstructed with code similar to the following:

▶ Playlist Reconstruction Code (Example)

**annas_archive_spotify_2025_07_metadata.torrent/spotify_clean_track_files.sqlite3**

The link between the tracks table in `spotify_clean` and the actual files we have.

Row count:

```
CREATE TABLE IF NOT EXISTS "track_files" (
        `rowid` integer PRIMARY KEY NOT NULL,
        /* spotify base62 id of the track */
        `track_id` text NOT NULL,
        /* the filename of this track within the Spotify Archive 2025 */
        `filename` text,
        /* The status of archiving this track. The nullability of most other fields
depends on the status.
            - success: Track file present in archive
            - error_not_available: Track no longer available in any country (likely
license expired).
            - error_has_replacement: Track not available, but replacement is. This
means Spotify itself replaces transparently (without showing to the user) replace this
track with a different track.
            - skipped_isrc_has_download: For tracks with low popularity, we only
fetch one copy of a song per ISRC - the one with the highest popularity.
            - skipped_low_priority: Archiving was skipped because track_popularity =
0 and secondary_priority < 0.351
            - todo_unk: Archiving status unknown
            - error_no_ogg160: Spotify reports no file in the highest quality is
available
            - error_zero_duration: Spotify reports a duration of 0ms on the track
file
        */
        `status` text NOT NULL,
        /* if this track has `popularity=0`, this shows the Ogg Opus kbit/s it was
reencoded with. if null, the quality is original. */
        `reencoded_kbit_vbr` integer,
        /* When the item was fetched (unixepoch ms). */
        `fetched_at` integer,
        /* The country the track was fetched from */
        `session_country` text,
        /* The SHA256 of the original raw file - different track ids may be 100%
identical */
        `sha256_original` text,
        /* The SHA256 of the actual file with metadata added */
        `sha256_with_embedded_meta` text,
        /* True if this ISRC has an archived file with higher popularity */
        `isrc_has_download` integer,
        /* The popularity of the track at the time of archiving */
        `track_popularity` integer,
        /* secondary archiving priority.
          = (max_artist.popularity from track.artists) / 100
            + album.popularity / 100
            + log10(min(100e6, max_artist.followers.total + 1)) / 8)
            - 10 * isrc_has_download.

          Max = 3. Negative for duplicates.
        */
        `secondary_priority` real,
        /* The actual raw bytes of the invalid Ogg packet stripped from the Ogg
Vorbis file.
```

```
              Contains unknown data plus the replaygain information:
              offset 144
                + 0: float32le track_gain_db
                + 4: float32le track_peak
                + 8: float32le album_gain_db
                +12: float32le album_peak
          */
          `prefixed_ogg_packet` blob,
          /* JSON array of track ids of transparently-replaceable alternatives Spotify
reports for this track. Likely similar to ISRC-equality. */
          `alternatives` text,
          /* The internal unique file ID of the various qualities available for this
track. */
          `file_id_ogg_vorbis_96` text,
          /* The archive always contains the file for this quality (vorbis160) */
          `file_id_ogg_vorbis_160` text,
          `file_id_ogg_vorbis_320` text,
          `file_id_aac_24` text,
          `file_id_mp3_96` text,
          /* json array of language codes present in the song */
          `language_of_performance` text,
          /* json array of the role of each artist (e.g. main_artist, composer,
featurd_artist) */
          `artist_roles` text,
          /* whether or not Spotify serves lyrics for this track */
          `has_lyrics` integer,
          /* internal id of the entity licensing this track */
          `licensor` text,
          /* The title without suffixes (e.g. feat. X) */
          `original_title` text,
          /* The name of this version of this track */
          `version_title` text,
          /* json array of country-specific content ratings, usually null */
          `content_ratings` text
   );
```

**annas_archive_spotify_2025_07_metadata.torrent/spotify_audiobooks.jsonl.zst**

Raw JSON API responses for audiobooks on Spotify. Contains around 700 thousand rows.
Incomplete.

**annas_archive_spotify_2025_07_metadata.torrent/spotify_audiobook_chapters.jsonl.zst**

Raw JSON API responses for audiobook chapters on Spotify. Contains around 20 million rows.
Incomplete.

**annas_archive_spotify_2025_07_metadata.torrent/spotify_shows.jsonl.zst**

Raw JSON API responses for shows (podcasts) on Spotify. Contains around 5 million rows.
Incomplete.

`annas_archive_spotify_2025_07_metadata.torrent/spotify_show_episodes.jsonl.zst`

Raw JSON API responses for (podcast) episodes on Spotify. Contains around 54 million rows.
Incomplete.

`annas_archive_spotify_2025_07_metadata.torrent/spotify_artist_redirects.json`

Redirect responses for the artist endpoint.

`annas_archive_spotify_2025_07_audio_analysis.torrent/##.json.zst`

Raw JSON API responses for audio analysis on Spotify. Contains around 40 million rows, fetched
in descending priority order. Many songs do not have an audio analysis (404). Incomplete.

`annas_archive_spotify_2025_07_coverart.tar.torrent`

Album art files. Correspond to last part of `url` from `album_images`. Indexed into directories with
filename prefixes (after stripping the first 16 chars from the filename).