Hit Music Prediction Model

DATA SCIENCE & ADVANCED ANALYTICS

AVILA, HENRIQUE

HIGAKI, JOSEPH

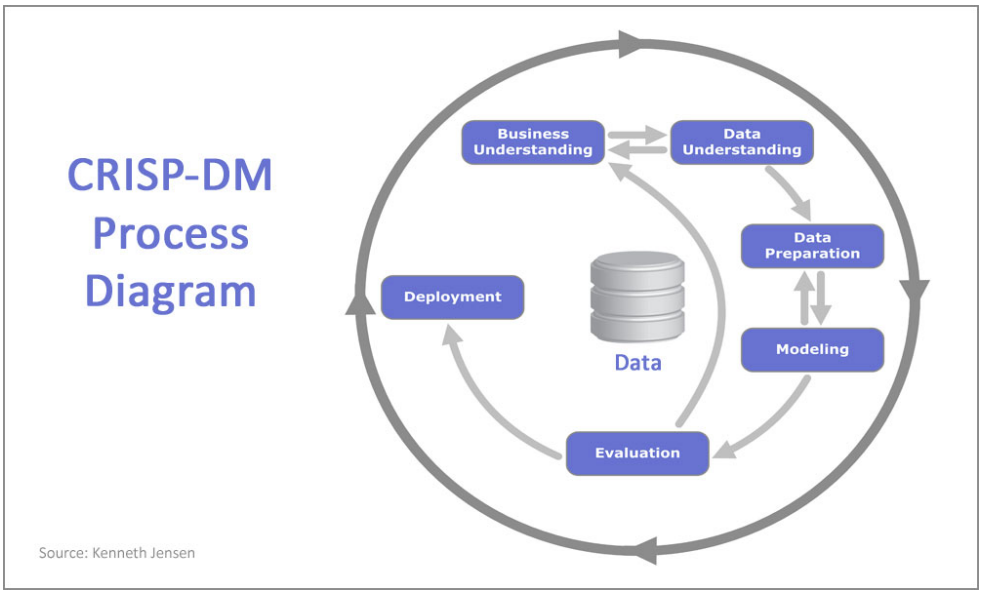
ASHUKRI, ZIYAD

BALEYNAUD, ROMAIN

GANUZA CATALÁN, RAQUEL

Contents

IMAGE 1: CRISP-DM PROCESS DIAGRAM



Business Understanding

Definition of the project objectives

The music industry generates $21.5 Billion per year. Other than the revenue it generates, music touches each and everyone of us on a personal level. More importantly, the music industry is a crowded industry, for example according to Spotify reporting, there are 60,000 songs uploaded on their platform each day. Combining other streaming services, such as Apple music, Napster and Dezzer the number of uploaded songs each day reaches hundreds of thousands.

With the massive number of songs uploaded each day, depending on which chart we are looking at, only a super small fraction of those songs are considered to be hit songs. The questions that present themselves in this case are, what constitutes a hit song? What makes a song a hit song? With that being said, our project objective will be to ***build a Machine Learning model that can predict if a song is a hit or not***?

It is indisputable that music is part of our lives on a personal level. People relate moments, places and even people to songs. But the music industry is a business and as such seeks to make a profit.

In the last decades many platforms have emerged where you can play music and one of the most prominent in the music industry is Spotify. Spotify pays based on the number of plays. First the rights holders (which can be record labels, publishers or composers) are paid, then the distributors (representatives) and finally the artist.

So, the question is: how can you increase the revenue per playback on Spotify?

According to an article published by Business insider, there is a pro-rata system that "pays incumbents according to the level of their transmissions against popular songs over a given period of time. More popular artists get more of their music streamed, which means they earn more." This article also points out that "it is key to be on a Spotify playlist [...] Once an artist manages to be here, the possibility of growing and getting more plays increases" As we can see, in order to earn more money, it is key to have more plays. (BusinessInsider, 2020)

Nowadays, the internet and streaming give many advantages to musicians who want to integrate into the music industry but at the same time it also means disadvantages when it comes to entering the music business. One of the options that artists have is to develop their music career through a record company and they get profits from both record sales and direct earnings from the artists (advertising, sponsorship, ticket sales for their concerts...). This is why record companies look not only for musical talent in the artist but also for the artist to perform concerts that people want to pay to attend. One of the factors that these companies take into consideration when signing a contract with an artist are the numbers in social networks (followers, likes...). They want to make sure that the artist has an audience before signing the contract.

Record companies are not the only way to get started in music. Today there are many ways to produce, distribute and market music. But regardless of whether the artist starts a music career with a record label or individually, the possibility offered by social networks to exploit it to the fullest is very important. (Audioproduccion, 2020)

So it is not only the artists who are interested in increasing the number of plays but also all intermediaries: record labels, publishers, representatives and composers.

It is to all the companies and individuals mentioned above that our work is directed. We want to help them increase their Spotify revenue and plays by providing them with a Data Analytics solution.

Requirement Understanding

To be able to achieve the project objective, we had to develop a deep understanding of the requirements. For example, one of the first questions we had to objectively define was what is a hit song & a non-hit song? In the end, music taste is highly subjective. Would a top 40 or top 100 chart be enough to determine what is a hit song with popular consciences? Or should the streaming count be our main variable? More importantly, by reviewing the empirical work that has been done before, we have observed a couple of limitations that hampered their models, which are the sample size being small (ranging from 300 to 1000 data points) & the attributes used.

Keeping the above in mind, to an extent we knew how to start our understanding and information gathering process. To start with, Kworb.com provides a list of roughly 5000 songs that represents the most streamed songs from 2013- to present. Also, using the Spotify API we were able to obtain 15 audio features to each song in that list. Taking some of the conclusions from previous pulplish papers, we decided to add social attributes such as singer and song information. Which will be detailed in the coming sections.

Describe the business context in detail

Our analysis will be sold to record labels, publishers, reps, songwriters and artists who are interested in moving up Spotify playlists. This work is an analysis of the behavior of the 5000 most played songs on Spotify from 2017 to 2021 but we can adjust the data to clients' demands and perform specific analysis for their needs or to predict how the release of a song on Spotify will behave: Will it be a hit or not?

Use case examples of this:

For Record Labels.

Record Label companies often need to make business decisions that include:

- Deciding which artist to sign-up
- Deciding the marketing / production allocation of funds, from the artist they have.

These decisions are heavily based on sales track record or even expert judgement. Our proposal is that this hit or not prediction becomes an additional tool that contributes to the decision making.

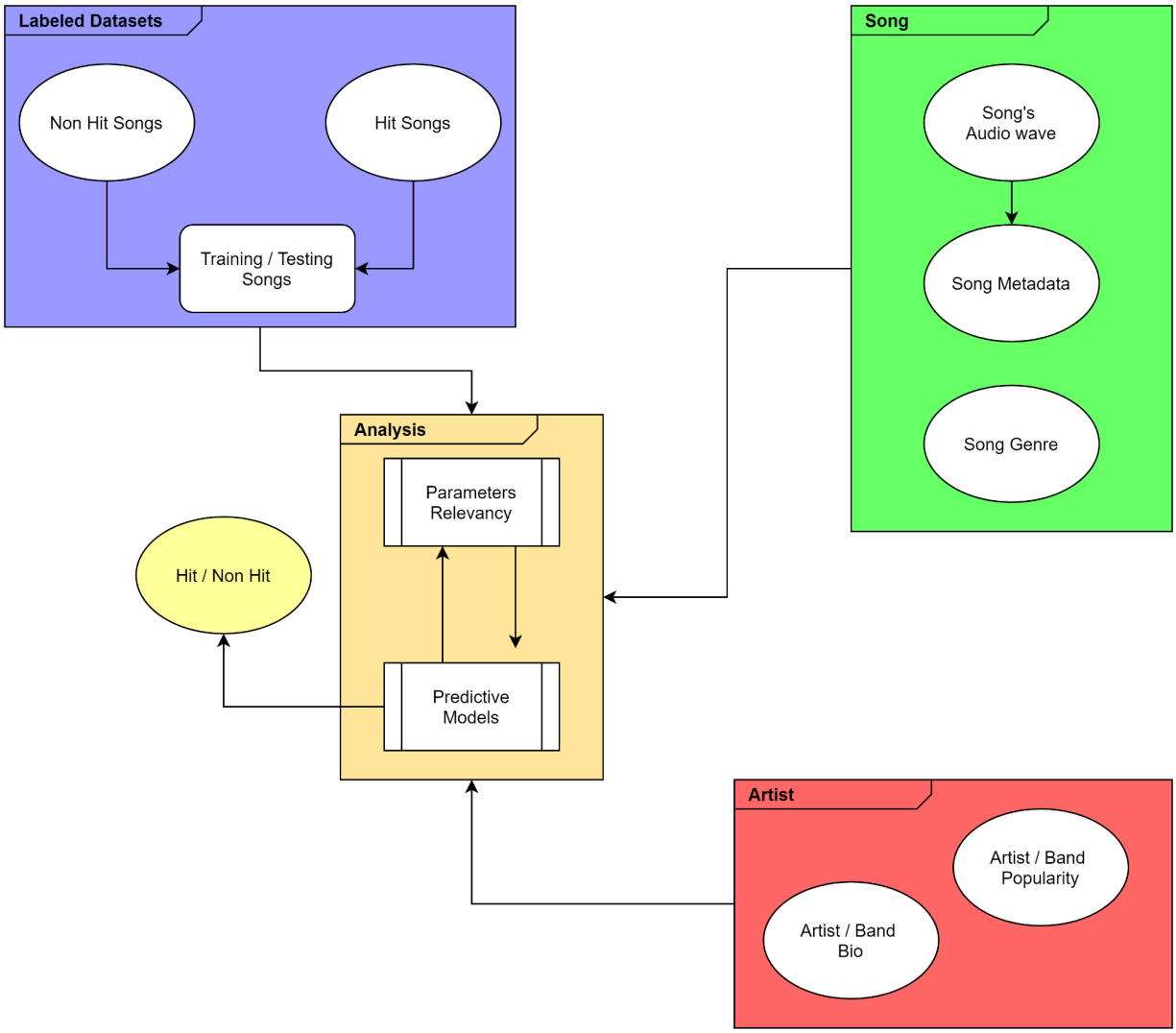
For Streaming Platforms.

The platforms have the capability to influence new music to be played. If the user preference matches a number of new released songs. One tactic can be where the platform use prioritize these songs that have been predicted as a hit. This way artists with predicted hit songs, can get the hit song revenue faster.

Exploratory Analysis and Data Understanding

The first rough outline on how we could predict a song's success, looked like this

IMAGE 2: CONCEPT-LEVEL PROCESS



Source: Own elaboration

Data collected Description

Song Analysis

Being our primary objective to analyze and predict based on songs, we need song metadata that we can interpret in a quantifiable manner. We're leveraging the great work of signal processing analysis of Echo Nest / Spotify.

Spotify API

For obtaining song metadata, we're using Spotify's Audio Features API (DeveloperSpotify, 2021)

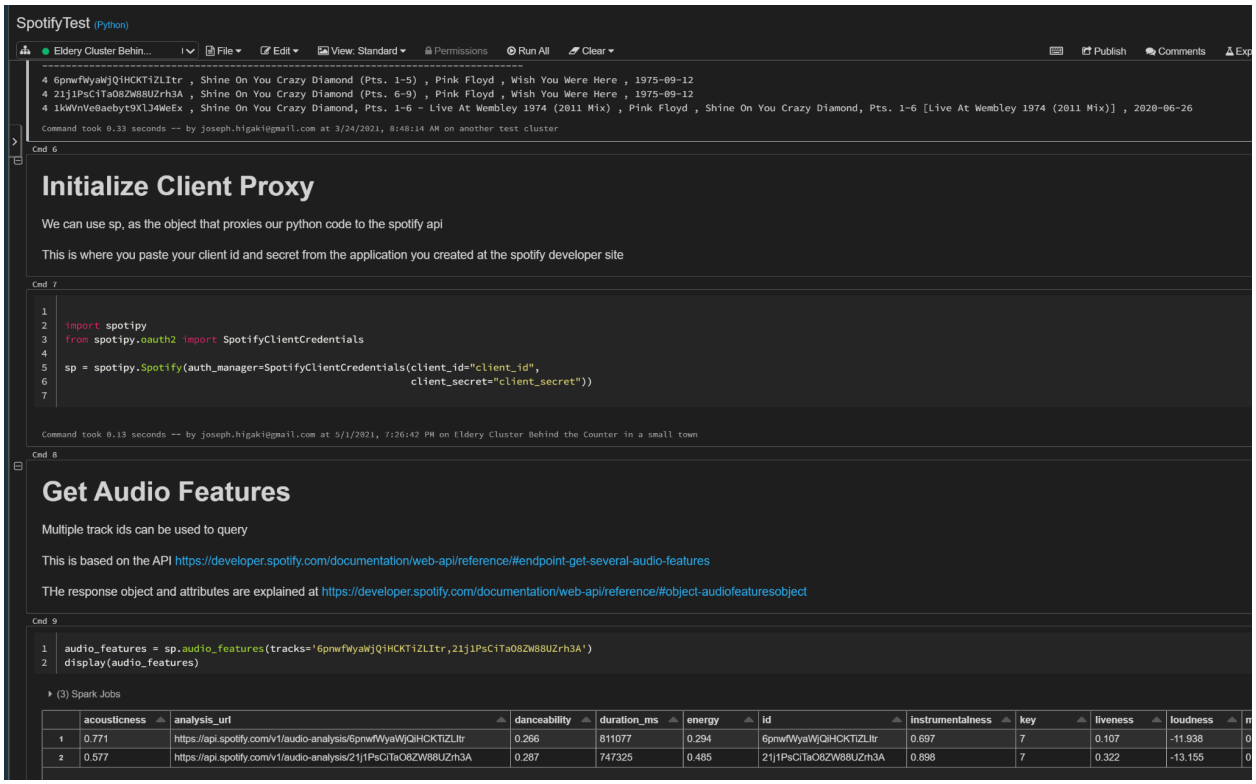
Spotify's Audio Features is the cornerstone of Spotify's recommendation algorithms.

We will use Spotify's Track ID, as the main identifier for a song. Using Track ID as input parameter at the Audio Features API, will get us 14 attributes that describe a song's musical characteristics. (Developer.Spotify, 2021)

The best way we found to extract information was using:

- Python, as programming language
- Databricks, as platform to run extraction and dataset generation routines
- Spotipy, as a Python client library to easily query the APIs. (Spotipy, 2021)

IMAGE 3: GET AUDIO FEATURES



Source: Own elaboration based on Spotipy, 2021

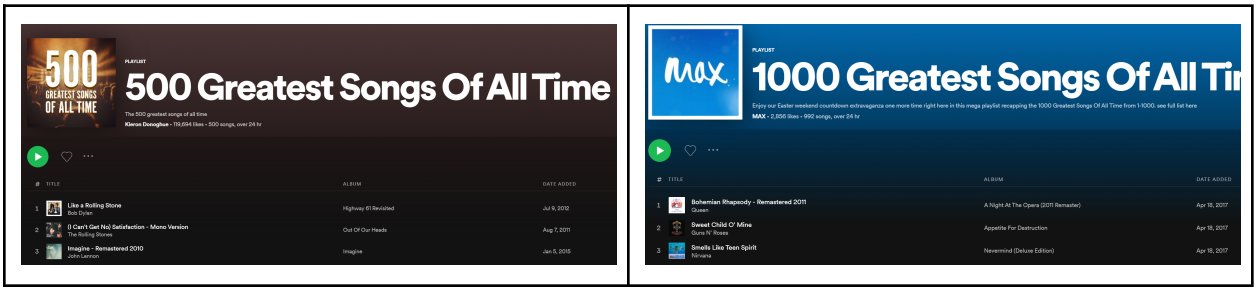
Hit/Non-Hit Dataset

Knowing that we have to go into a classification type of problem, determining whether a song is a hit or not, became our first problem to solve.

Spotify Playlists

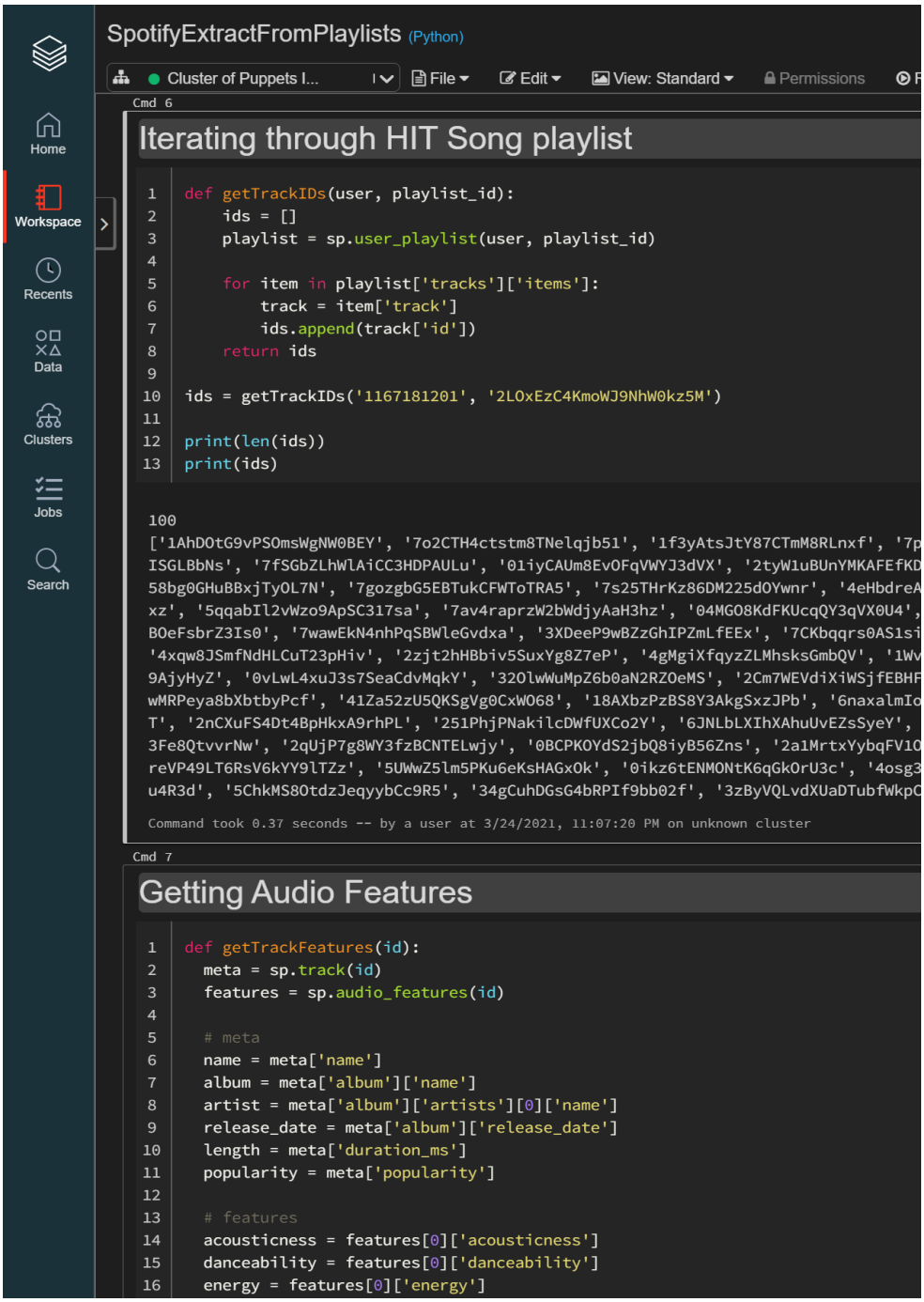
This was our MVP approach to try to have a HIT Song dataset ASAP so that we could start playing-around the audio features. We grabbed some playlists, downloaded its tracks through Spotify's API, and labeled them as our HIT Song Dataset

IMAGE 4: SPOTIFY PLAYLISTS



Source: Own elaboration based on spotify playlists

IMAGE 5: SPOTIFY EXTRACT FROM PLAYLISTS



Source: Own elaboration

This was great to get us started, but we found two major drawbacks to this approach:

1. Limited capability to determine NON-HIT Songs.
2. Greatest hits playlists are actually made by a user from the community, thus highly subjective.

Billboard Charts

Trying to overcome the highly subjective side from TOP x hits at Spotify playlist we got to Billboard charts. Billboard magazine has been arguably the most representative publisher of music popularity rankings, since 200. It reflects sales, airplay and since 2005 it has begun including digital platforms, so this would be a very good way to label HIT data. (Billboard Charts, 2021)

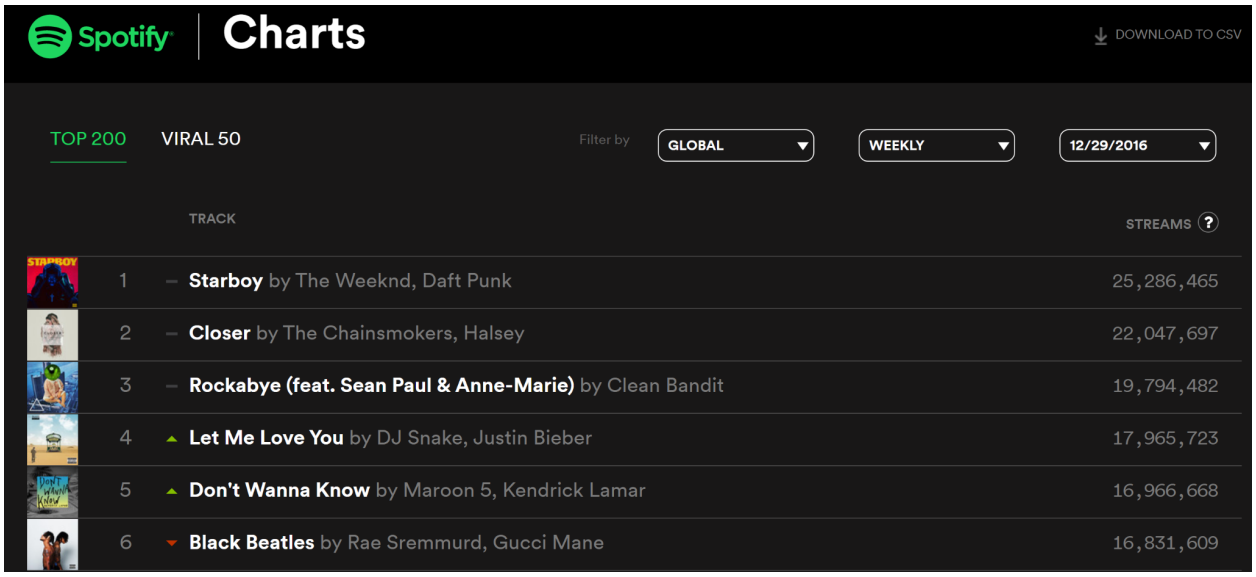
After spending some time on the website for Billboard Charts, and analyzing ways to create datasets we can use, we decided not to go with this option because:

1. Limited capability to determine NON-HIT Songs.
2. No Billboard API available
3. Articles/projects we found about using scraping libraries to extract the charts, weren't as easy to use as we would've wanted to.

Spotify Charts

We naturally went to the Billboard-like approach of Spotify Charts. This website would get us popularity ranking on a daily or weekly basis. It has a .csv download option and has a Stream Count snapshot column for the time queried.

IMAGE 6: SPOTIFY CHARTS



Source: Own elaboration based on Spotify Charts, 2021

One curious thing about Stream Count is that it is just not available at the Spotify API when you query a song. This aspect alone gave great relevance to this data source.

Spotify does not have an API to service their Charts data, so for us to get a dataset that would span several weeks, we would need to download week by week, and aggregate the sets.

This wasn't that challenging to automate, as we could've easily iterated through week-start dates and built a GET request that would've started downloading the .csv's.

We even went the code-less approach, where we just used a spreadsheet to generate URLs like this:

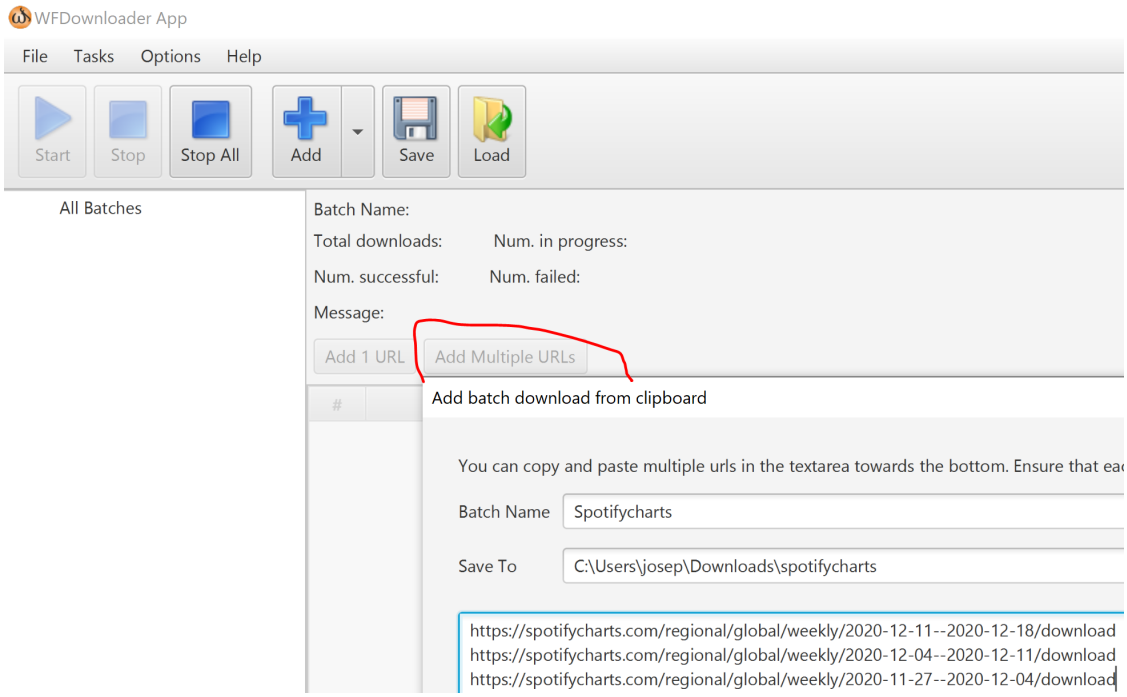
https://spotifycharts.com/regional/global/weekly/2020-12-11--2020-12-18/download

https://spotifycharts.com/regional/global/weekly/2020-12-04--2020-12-11/download

https://spotifycharts.com/regional/global/weekly/2020-11-27--2020-12-04/download

And feed them through a text file, to any download manager tool that would use an input text file as URL input. We used but do not recommend: https://www.wfdownloader.xyz/download

IMAGE 7: WS DOWLOADER APP



Source: Own elaboration based on WS DOWLOADER APP

By the time we started reviewing a sample of files and began designing how to aggregate this information, we found another source of information that was also based on Spotifycharts, but it would save us a lot of processing and aggregation work.

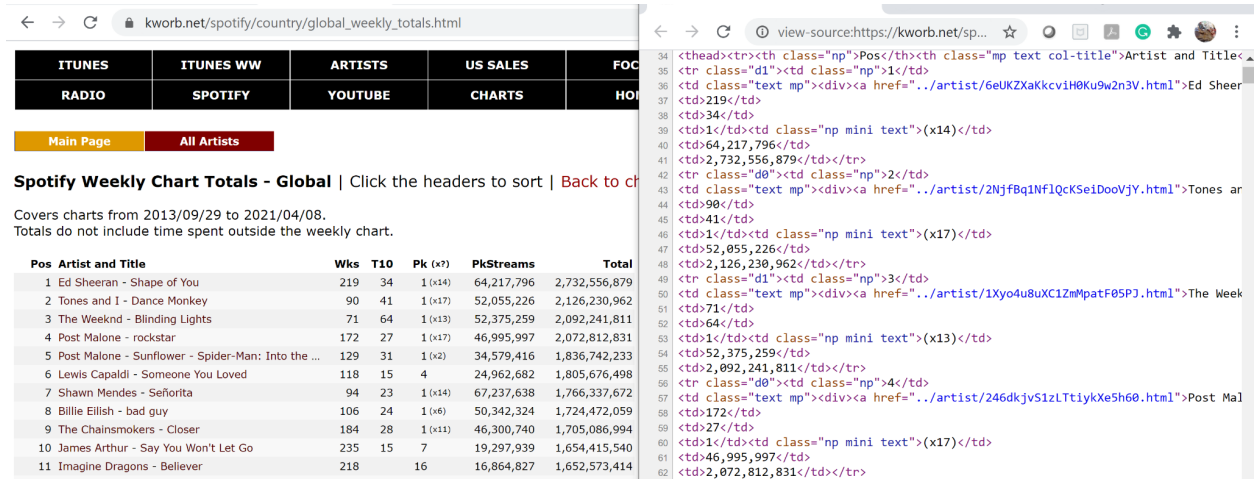
Kworb

https://kworb.net/

This is a website/pet project/hobby of a guy from the Netherlands. It is a website that has song rankings from multiple streaming platforms, one of them being from Spotify.

Even though there is no API, the minimalist yet rich of information static design made it very easy to scrap it.

IMAGE 8: KWORB



Source: Own elaboration based on Kworb

It was surprisingly refreshing to find a site with recent information that would render the information in plain html. No dynamic data loading, no pagination. The global ranking from KWORB had almost 5K songs representing charts since September 2013 and we could parse it using plain MS Excel functions and features. We ran a sample of records through a verification, just checking if the counters and songs looked consistent with what we had been getting at Spotify Charts, and it was impressively up-to-date and accurate.

And just like that, we had our comprehensive train/test dataset that had enough information for us to make assumptions of top/bottom outliers to label hit/non-hit songs

Social Media Artist information

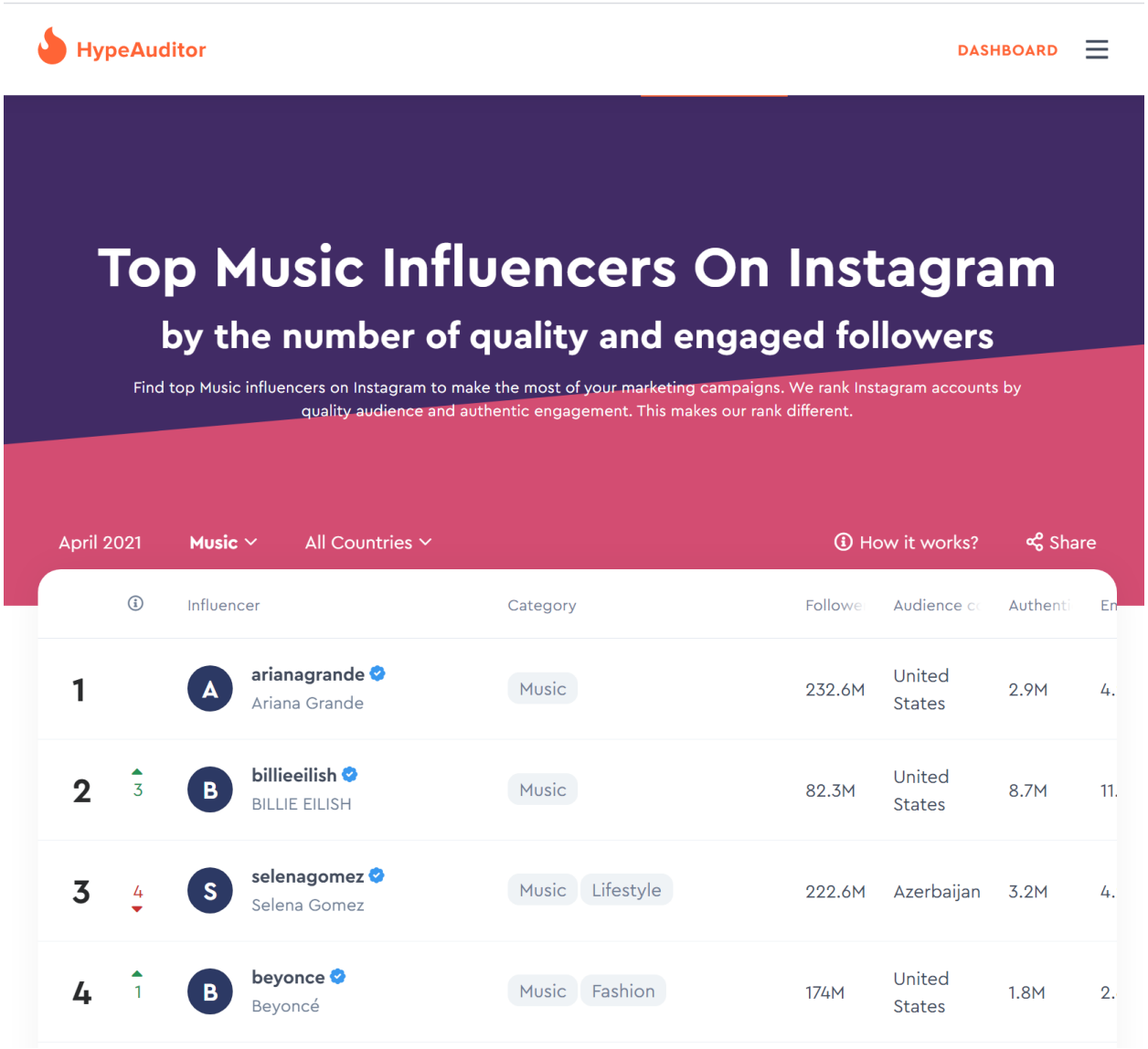
To have additional information for analyzing hit/non-hit songs, we wanted to dive into the artists' information.

Hypeauditor

Hypeauditor is a known service that publishes popularity rankings of people of interest, these rankings are based on social media followers and it has a dedicated section for musicians. (Hypeauditor, 2021)

The major problem was that it lacked an API, and it charged for full report/ranking.

IMAGE 9: HYPEAUDITOR



Source. Own elaboration based on hypeauditor, 2021

Music Brainz

We found Music Brainz, an open music encyclopedia that collects metadata and makes it available.

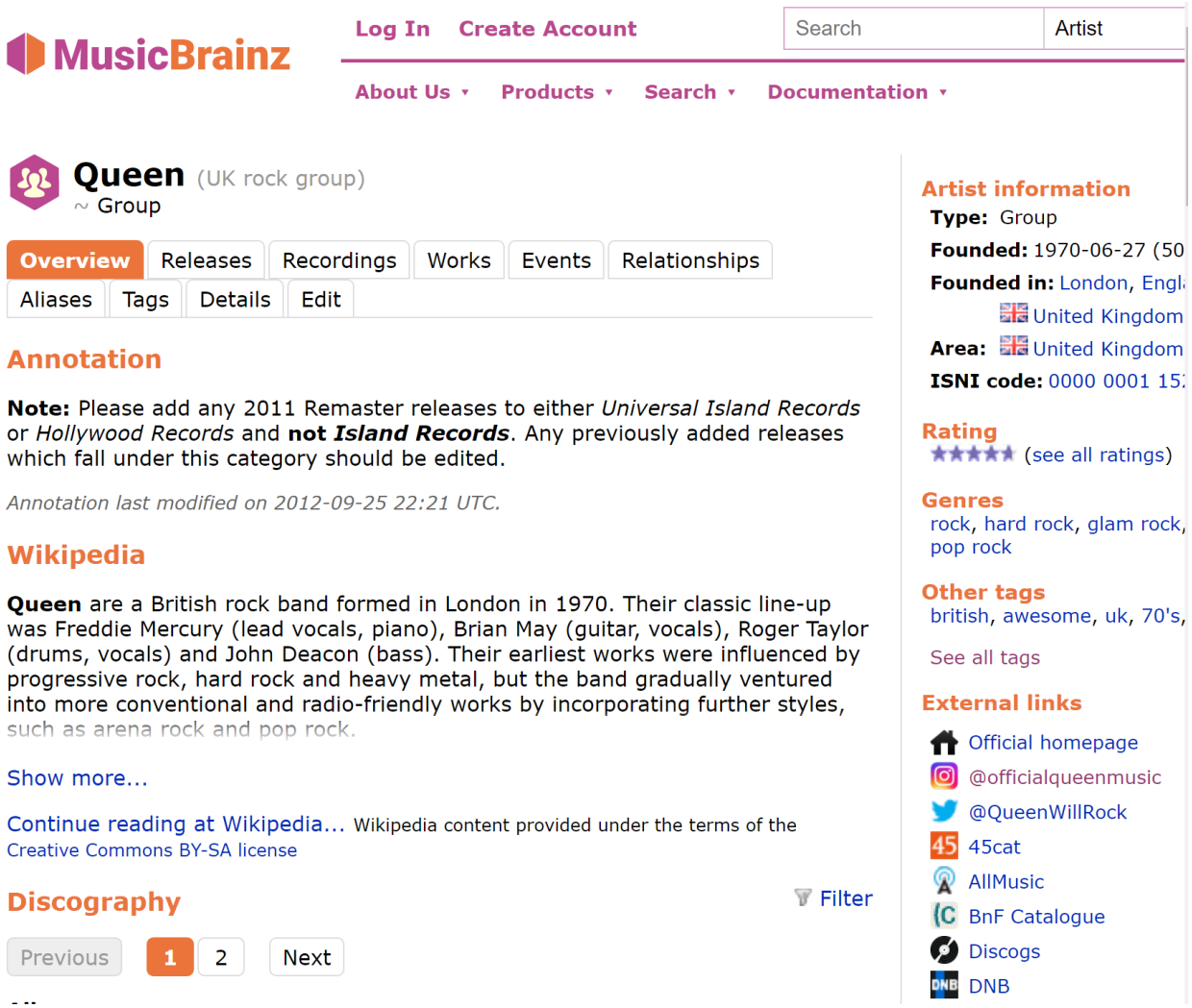
This website, hence the underlying database, was rich with external references to Facebook, Wikipedia, Instagram, Spotify, Twitter, Soundcloud, etc.

The major drawback was its interoperability.

We struggled for the API to get us what we wanted, from our point of view the semantics on the REST API could be improved. Even though the metadata is based on entity relationships, a more hierarchical approach representing the resources as part of the path could improve readability.

Eventually, it was taking too much time to run a small proof of concept on top of MusicBrainz, so in parallel, we needed to have an alternative.

IMAGE 10: MUSIC BRAINZ

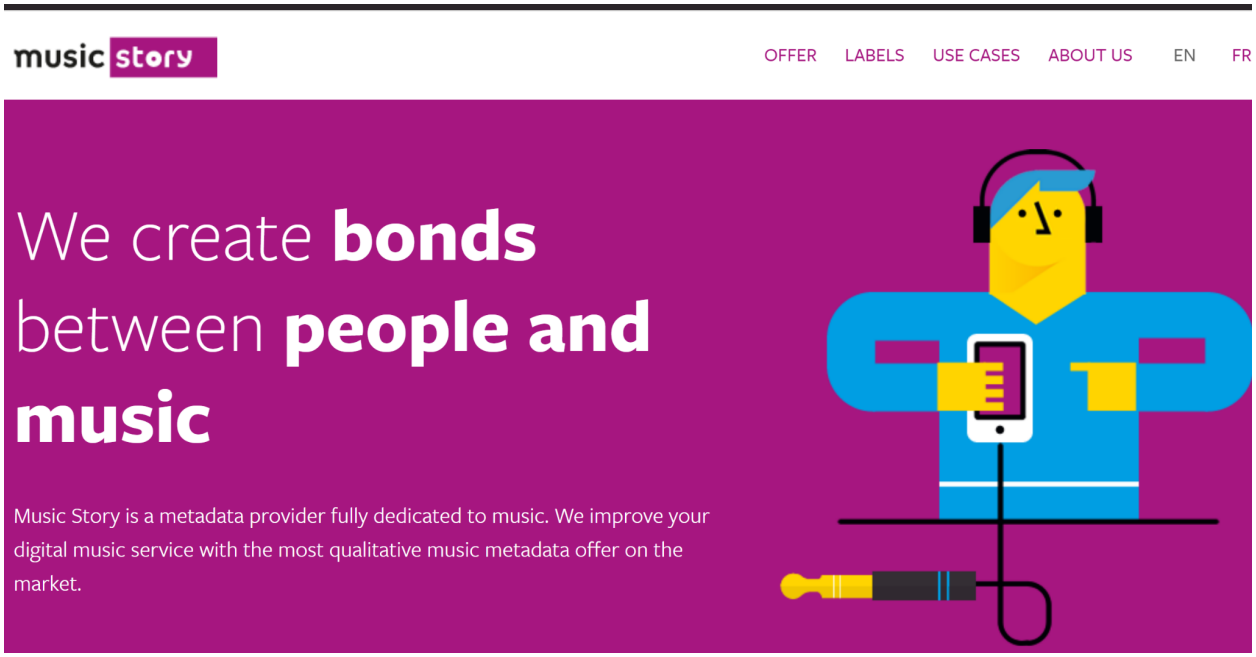


Source: Music Brainz, 2021

Music Story

This is another music metadata database. Its approach is one where they hold the artist as the main entity, from there you can navigate to its biography, different social media channels, and even news articles.

IMAGE 11: MUSIC STORY



Source: Music history, 2021

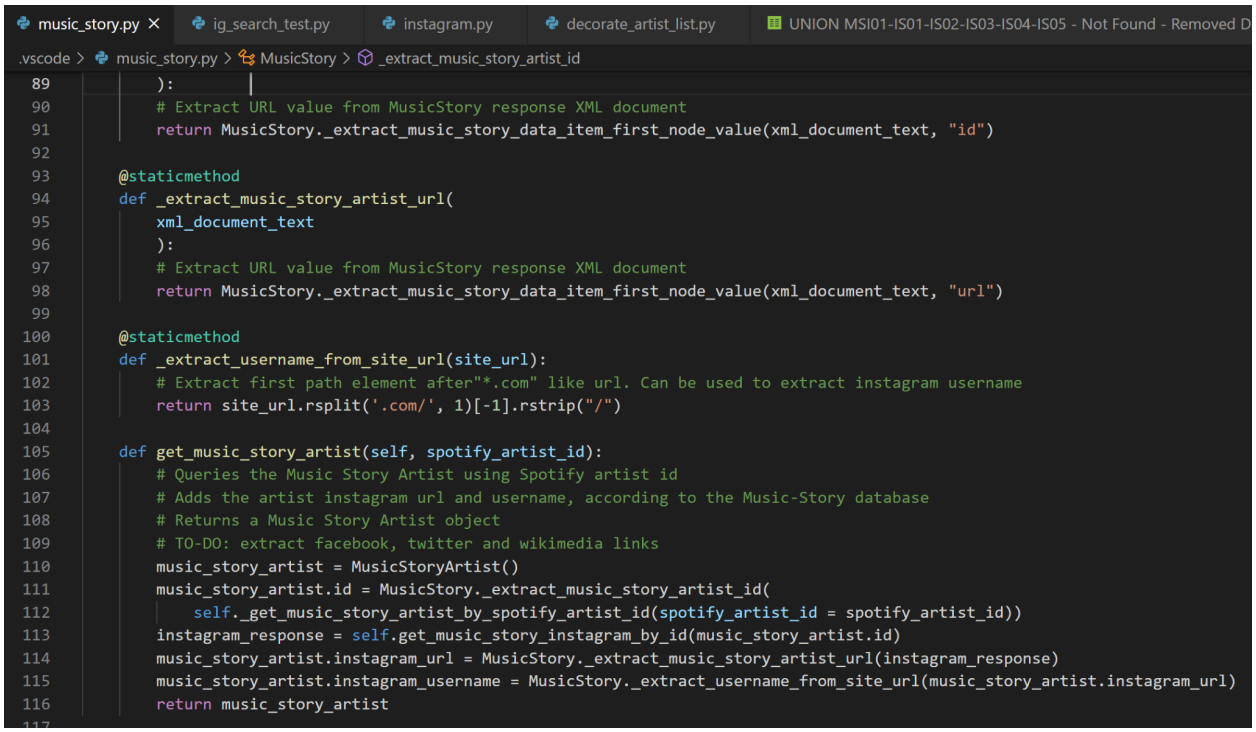
At this point in time, we were against the clock and even though we were aiming to get Instagram follower count, we liked the potential of this artist information hub.

The API design was simple and intuitive, the documentation was great. The only thing we could complain about is the fact that it retrieves an XML document regardless of the Accept header you send. But well, we're all old enough to know how to run simple XPath queries.

We were getting successful API responses in no time, and running a few queries using a small sample from our artist training set, the information looked very promising.

What was great about music-story is that it has an endpoint where you can query their artist entity using the Spotify artist id as the input parameter. Once we have the Music-Story artist entity, we can easily query the artist's Instagram URL and include it in our dataset.

IMAGE 12: MUSIC STORY CLIENT PYTHON



Source: Own elaboration

Did we mention that we ran a small sample of artists from our training set and it looked promising?

Well, after running the entire list through the music-story API, we only gathered a rough ~ 41% of valid Instagram profiles. Nice try to get us started, but we were aiming for the entire list.

Instagram Name Search

As we had 59% of our artist list without an Instagram username, we decided to run through that list and launch Instagram name searches.

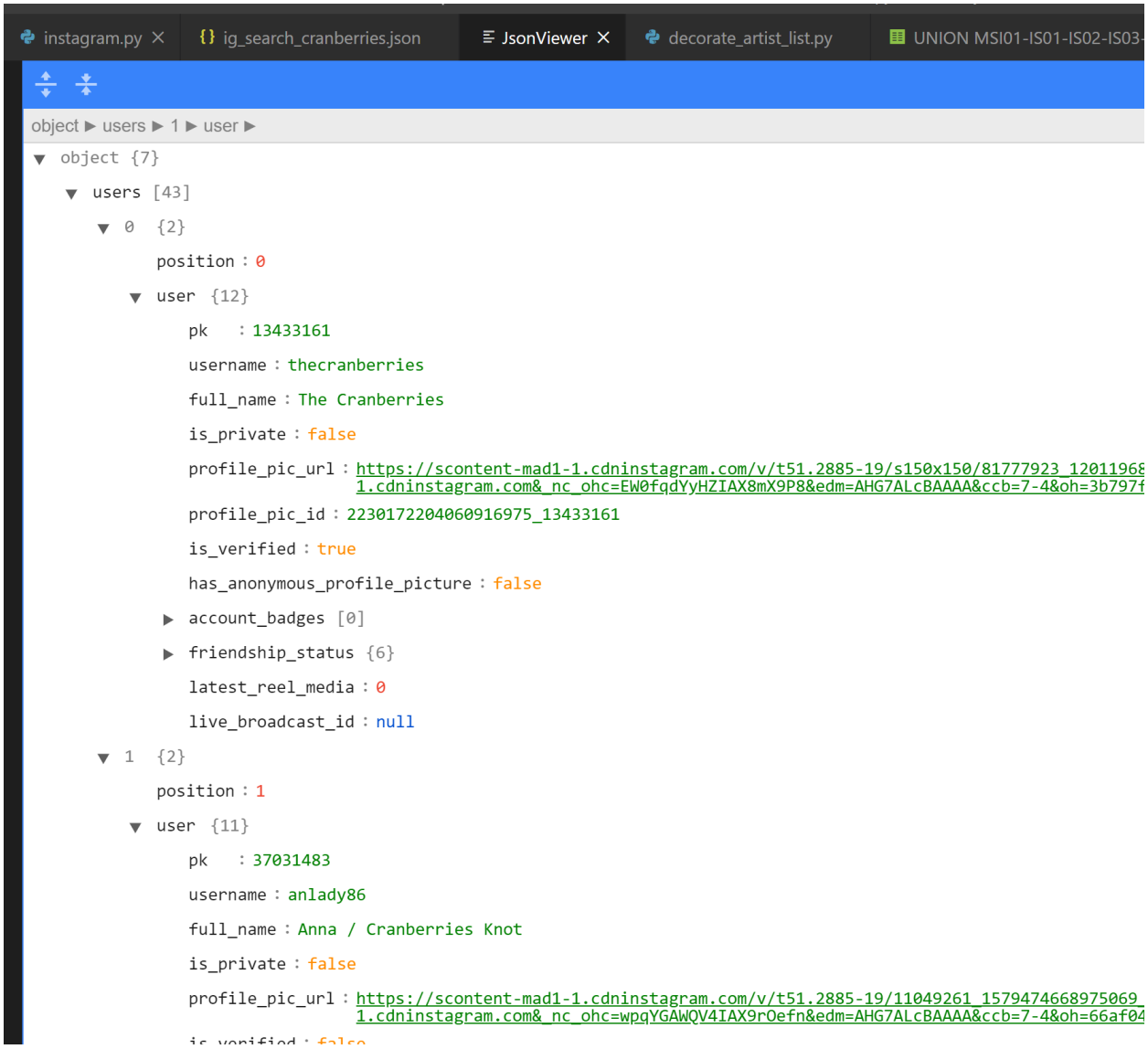
This is being written as of April 2021, by this time the legacy Instagram API had been decommissioned for almost a year. APIs are now based on the Facebook platform, providing a Basic Display and Graph API platforms, none of which were very easy to use.

Fortunately, the Instagram website has a couple of endpoints returning JSON documents that you can invoke with your regular web/app user credentials. Even more fortunate is that we found Instaloader, a Python client module that takes advantage of those Instagram endpoints and tackles the complexity of establishing a user session: https://instaloader.github.io/

An Instagram search would look like this:

https://www.instagram.com/web/search/topsearch/?query=cranberries&__a=1

IMAGE 13: INSTAGRAM SEARCH RESULT: CRANBERRIES



Source: Own elaboration

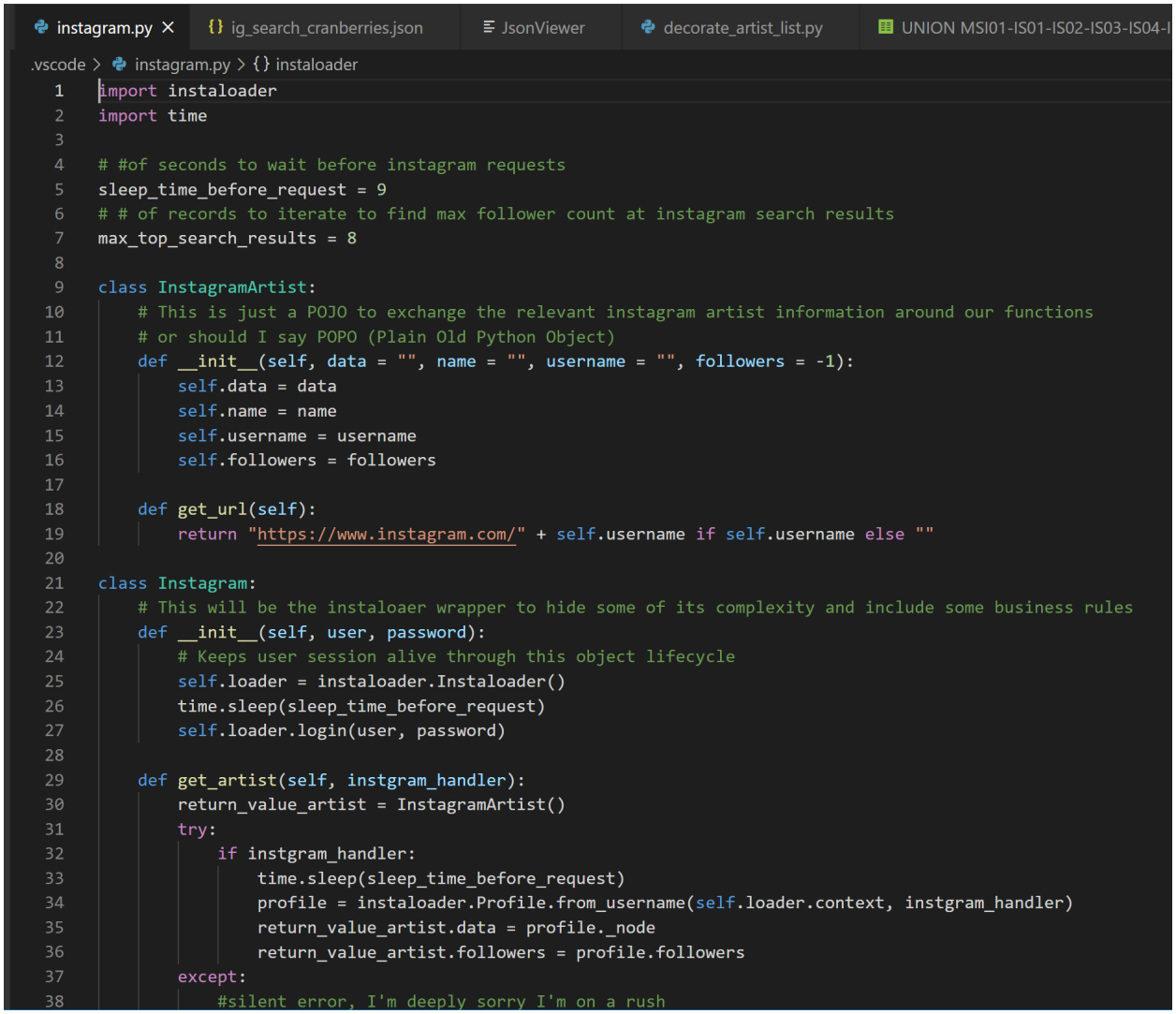
Ideally, and most of the time, the official profile would come up as the first result.

There were other times where we had no idea which item from the result list would be the official one, so we took the assumption that we would consider the one with the most followers.

Making many requests per second is something that Instagram identifies as malicious behavior, wisely probably to prevent DDoS attacks, so we took the assumption that:

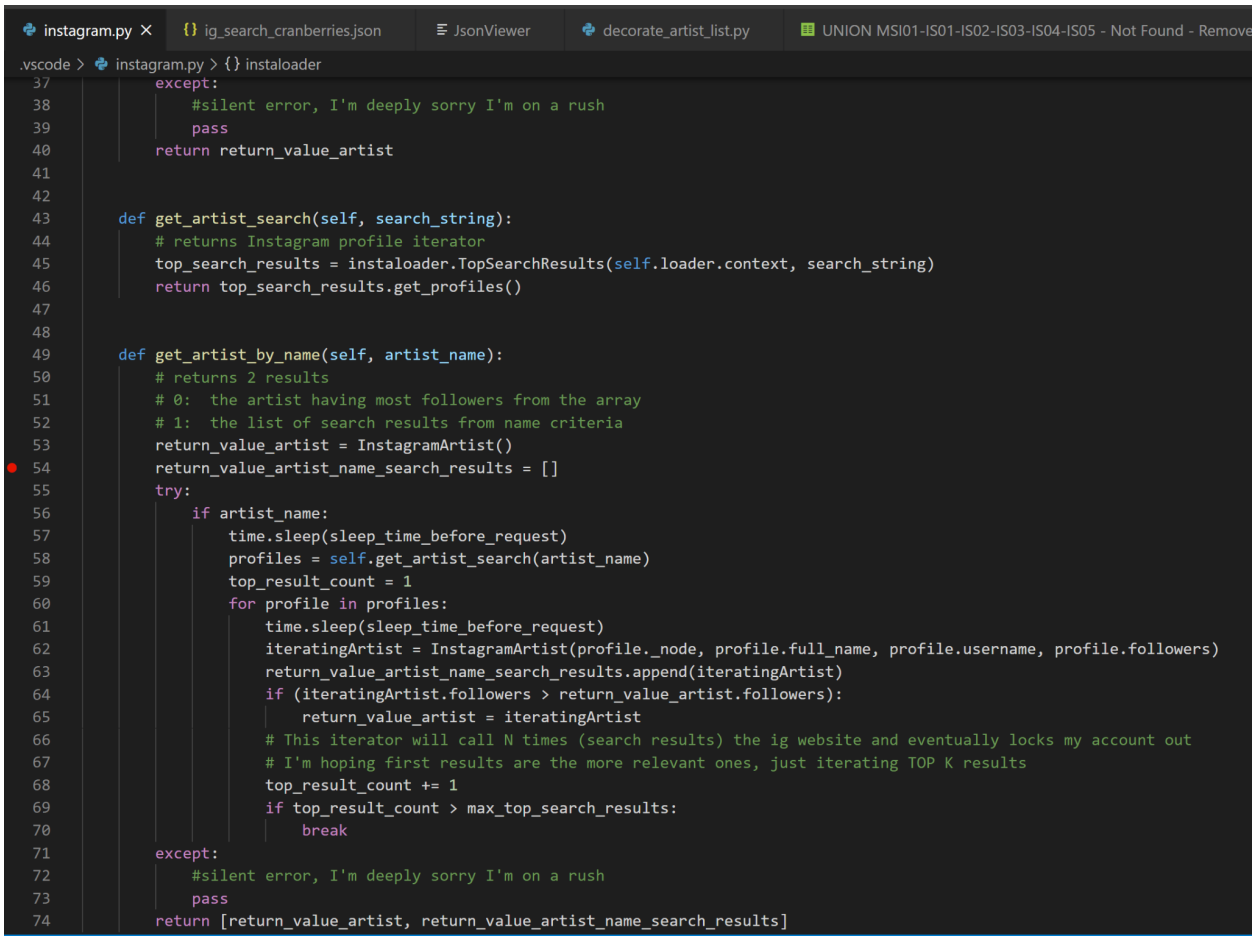
1. We will only iterate over the first 8 result profiles, and grab the item with the largest follower count
2. We will wait 9 seconds before every Instagram request.

IMAGE 14: INSTAGRAM CLIENT PYTHON



Source: Own elaboration

IMAGE 15: INSTAGRAM CLIENT PYTHON (2)



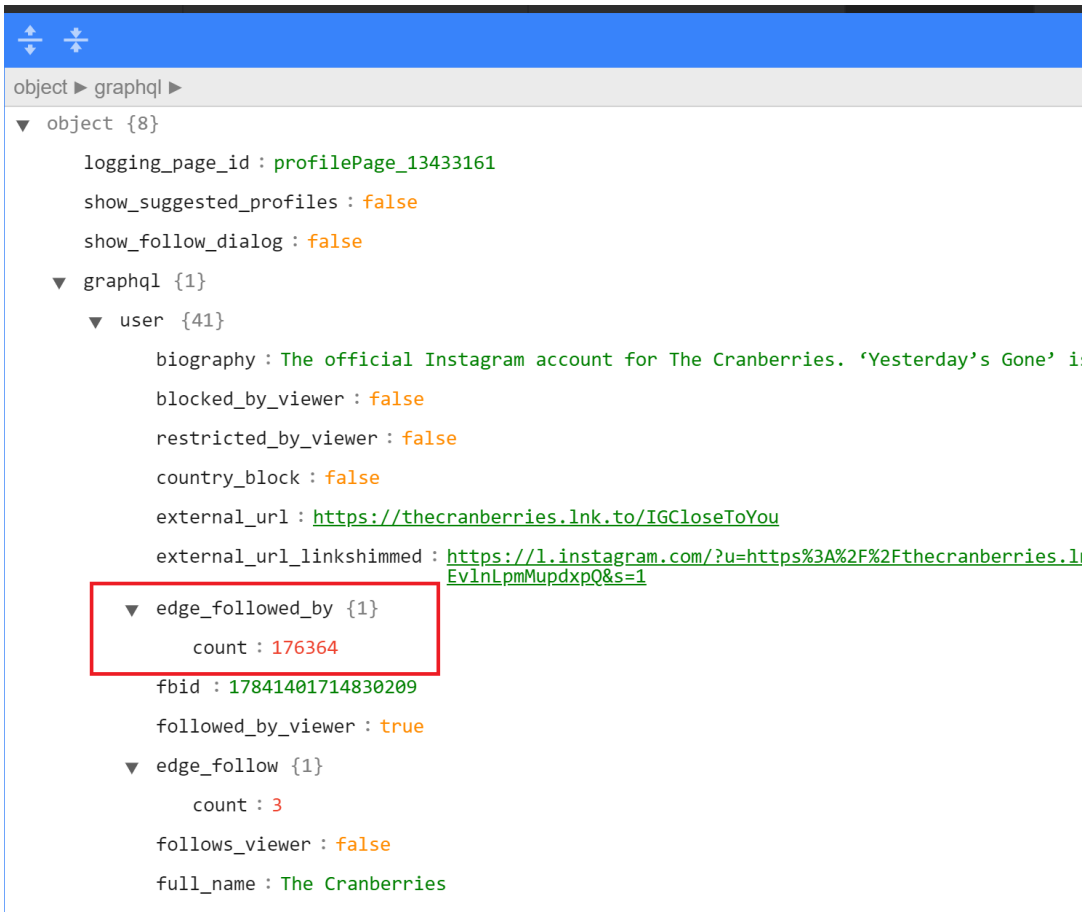
Source: Own elaboration

Instagram Followers Count

if we were 41% lucky at the music-search results, the only thing we needed to do is to grab follower count. Again, instaloader helped us with the user session, as Instagram provides a pretty helpful JSON document when you send the query parameter: __a=1

Example:

IMAGE 16: INSTAGRAM PROFILE DATA: THE CRANBERRIES

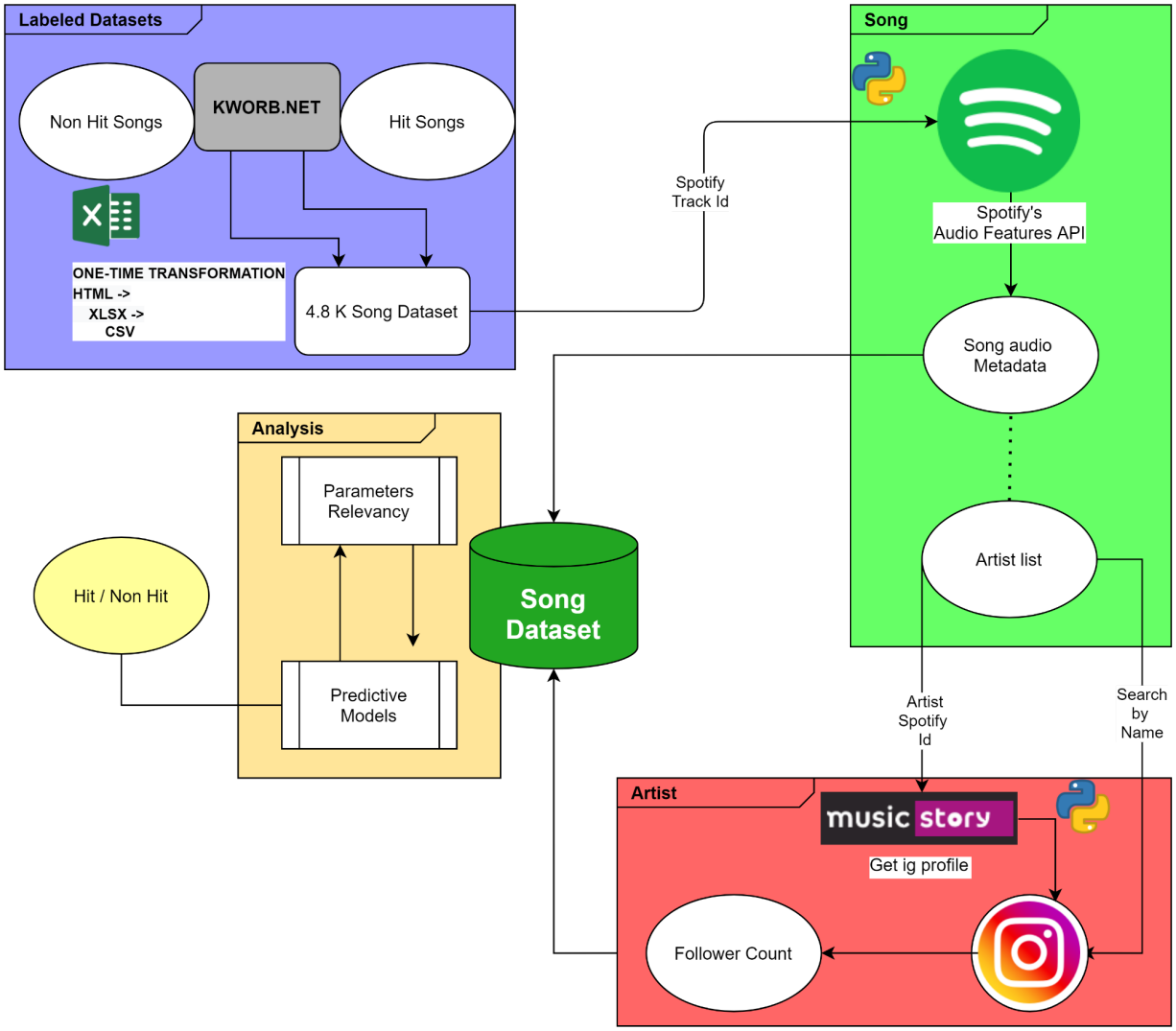


Source: Own elaboration based on https://www.instagram.com/thecranberries/?__a=1

Diagram of Data Sources

The outline of how we obtain the data ended up looking like this

IMAGE 17: DIAGRAM OF DATA SOURCES



Source: Own elaboration

Song Dataset

From the three sources of information from which we have extracted our data, we have obtained a total of 4286 songs and 31 variables, which we are going to name and describe in the following table:

TABLE 1: VARIABLES DESCRIPTION

	VARIABLE	SOURCE	VARIABLE DESCRIPTION	VARIABLE TYPE
Song Information	Pos	KWORB	Position of the song in the table	Quantitative: Numerical
	artist_and_title	KWORB	Name of the artist and name of the song	Categorical: Nominal
	song_name	KWORB	Name of the song	Categorical: Nominal
	name	SPOTIFY API	Name of the song	Categorical: Nominal
	album	SPOTIFY API	Name of the album	Categorical: Nominal
	weeks	KWORB	How many times the song has been in the list	Quantitative: Numerical
	T10	KWORB	How many weeks the song has been in the top 10	Quantitative: Numerical
	Pk	KWORB	Top position of the song in the table	Quantitative: Numerical
	X	KWORB	How many weeks the song has been in the Pk position: (x3)	Quantitative: Numerical
	pk_streams	KWORB	Highest number of plays of the song in one week	Quantitative: Numerical
	total	KWORB	Total number of plays on spotify	Quantitative: Numerical
	artist_url	KWORB	URL of the artist on Spotify	Categorical: nominal
	track	KWORB	URL of the artist on Spotify	Categorical: Nominal
	release_date	SPOTIFY API	date the song was released	Quantitative: Date
	release_month_date	SPOTIFY API	Month the song was released	Categorical: Ordinal
	Popularity	SPOTIFY API	The popularity of the track. The value will be between 0.0 and 1.0, with 1.0 being the most popular. Popularity is track-based and a measure of how many plays a track received and how recent those plays are. Then artist popularity is derived from that.	Quantitative: numerical
Audio Features	Acousticness	SPOTIFY API	A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.	Quantitative: numerical

	Danceability	SPOTIFY API	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.	Quantitative: numerical
	length	SPOTIFY API	The duration of the track in milliseconds.	Quantitative: numerical
	Energy	SPOTIFY API	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.	Quantitative: numerical
	Instrumentalness	SPOTIFY API	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 instrumentals 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.	Quantitative: numerical
	Liveness	SPOTIFY API	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.	Quantitative: numerica
	Loudness	SPOTIFY API	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 typical range between -60 and 0 db.	Quantitative: numerical
	speechiness	SPOTIFY API	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 0.666 describe tracks that are probably made entirely of spoken words. Values between 0.333 and 0.666 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.333 most likely represent music and other non-speech-like tracks.	Quantitative: numerical
	tempo	SPOTIFY API	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.	Quantitative: numerical
	time_signature	SPOTIFY API	An estimated overall time signature of a track. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).	Quantitative: numerical
	mode	SPOTIFY API	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.	Categorical: binary (Major/minor)
	valence	SPOTIFY API	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).	Quantitative: numerical

	key	SPOTIFY API	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.	Categorical: Ordinal
Artist Information	artist_name	KWORB	Name of the artist	Categorical: Nominal
	artist	SPOTIFY API	Name of the artist	Categorical: nominal
	instagram_followers		Number of followers the artist has on instagram	Quantitative: Numerical

Source: Own elaboration based on Spotify for Developers

Additional, non-gathered data points

In this section we list other data that would've been interesting to gather, but each of them posed certain challenges that would've taken more time than the one allotted to finish this analysis.

TABLE 2: NON GATHERED DATA POINTS

VARIABLE	SOURCE	VARIABLE DESCRIPTION	CHALLENGES
Song Genre	Wikipedia, Spotify, Audio Signal processing	Song's genre	Spotify provides the list of genres an artist includes in it's albums. But that doesn't necessarily provide a given song's genre. Another option is to analyze by the Spotify audio_analysis API. This is highly complex. Another option is to extract it from wikipedia.
Artist_age	Wikipedia	Age of the artist. Or the array of a band's members age	MusicStory or MusicBrainz opened up this capability for us. Because we could navigate effectively from band to artists and from artists to wikipedia articles. But, even if we would've been able to do that navigation, wikipedia articles or wikimedia documents do not provide exact metadata. We would've needed to do some text analysis.
Season	Spotify	Season the song was released. For example Summer or Winter	We need to navigate from song to album and find the release date. From there, we could've set Northern / Southern Hemisphere seasons. But, we wanted also to have the artists/band main country, which takes us also to the navigate/scrap wikipedia challenge.
Country	Wikipedia	Country of the artist	Navigate/scrap wikipedia challenge.
Key words	Lyrics provider	The frequency of the words from the song lyrics.	We could go from song to lyrics using lyrics provider. Some of these are not free. The ones that are popular and have an API available are: https://www.musixmatch.com/ https://www.lyrics.com/

Chorus frequency	Lyrics provider	number of times the chorus of the song is repeated	Hag
Lyrics Sentiment	Lyrics provider	The sentiment analysis from the song lyrics	Having the lyrics we could've analyze: ● Polarity from the song title nouns. Meaning if it is Negative, Positive, or Neutral to that noun. ● Emotional states: anger, disgust, joy, fear.
language of the song	Wikipedia / Lyrics provider	In which language(s) the song is.	We don't know whether it would've been easier to do lyrics analysis or the Wikipedia scrapping. Probably the latter.
chord progression	song sheet music	What chord progressions does the song have	Time and we need more people who know how to read music. It is difficult to convert data into numbers
Featuring artists	Spotify	if the song is sung by one or more artists (one or more musical groups)	We needed to spend more time with the Spotify APIs, and we decided this piece of information was not amongst our top priorities.
Band	Spotify	Whether the song is performed by a band or a solo artist.	We needed to spend more time with the Spotify APIs, and we decided this piece of information was not amongst our top priorities.

Source: Own elaboration

Identification of problems due to data quality

Once the data have been obtained and the variables have been described, the problems we may face due to the quality of the data may be the following:

- heteroscedasticity
- missing values
- outliers
- duplicates

We will deal with these problems in the data cleansing part.

Attribute analysis and data statistics

First, to describe the sample that participated in our study, we will carry out a univariate analysis using descriptive statistics. We will focus on two of them: measures of central tendency (mean, median, mode...) and measures of dispersion (standard deviation). For qualitative or categorical variables, we will use a frequency table and for quantitative or continuous variables we will perform descriptive analysis.

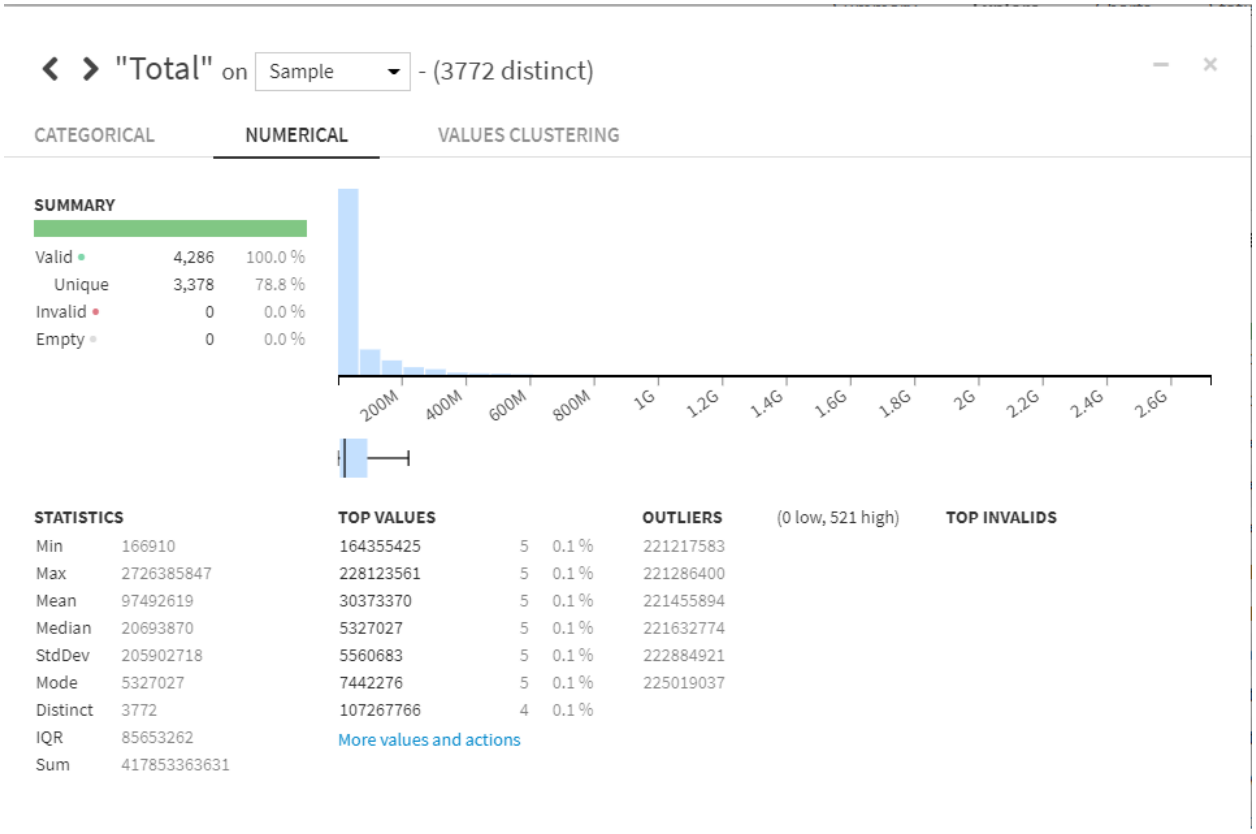
Secondly, in order to discover the possible relationship between two variables, we will perform a bivariate analysis. Within this, we will distinguish the types of variables that we wish to relate in order to subsequently apply the appropriate statistics. In the correlation table, we will use Spearman's ranks

because Spearman's correlation evaluates the monotonic relationship between two continuous or ordinal variables. That the relationship is monotonic means that the variables tend to change at the same time, but not necessarily at a constant rate.

We will also test the Spearman coefficients to determine whether or not there is a relationship between the variables and the total number of Streams, our objective being to reject the H0: the variable X does not affect the number of Total Streams. We will use Dataiku to show the results.

UNIVARIATE ANALYSIS

FIGURE 1: Total Streams

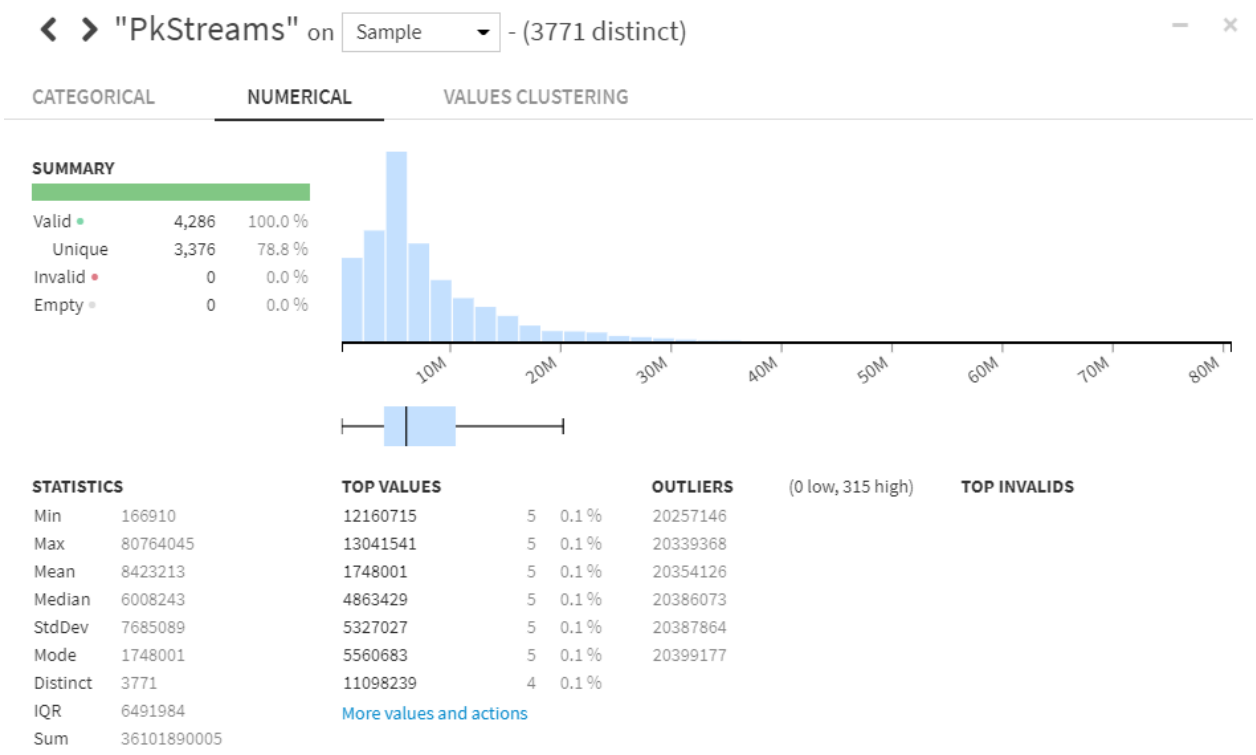


Source: Own elaboration

The variable "Total Streams" shows the total number of plays of a song on Spotify.

We can observe that we have 4286 valid values. The song that has the minimum of accumulated reproductions has 166.910 reproductions and that most reproductions have 2.726.385.847. The range of this variable is very large as well as the standard deviation. It also stands out that half of the songs have more than 21,159,476 plays. We also have to observe the outliers. In this case, outliers do not indicate measurement error. We consider these outliers as part of the population.

FIGURE 2: pk_streams

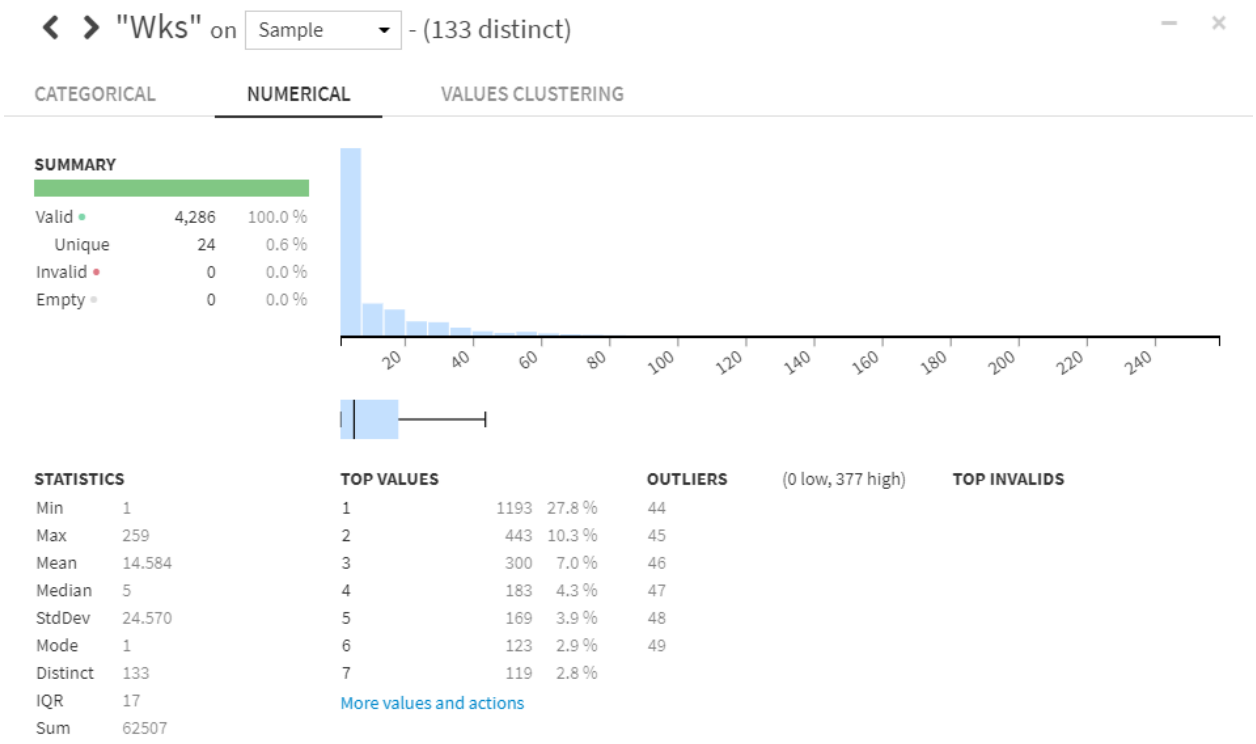


Source: Own elaboration

The variable "Pk_stream" shows the highest number of times a song has been played in a week.

This variable highlights that the song with the most plays in a week had 80,764,045 and the least 166,910 and that half of the songs, in their peak week, obtained more than 6,008,243 plays. This variable depends on the “Total Streams” variable.

FIGURE 3: Weeks

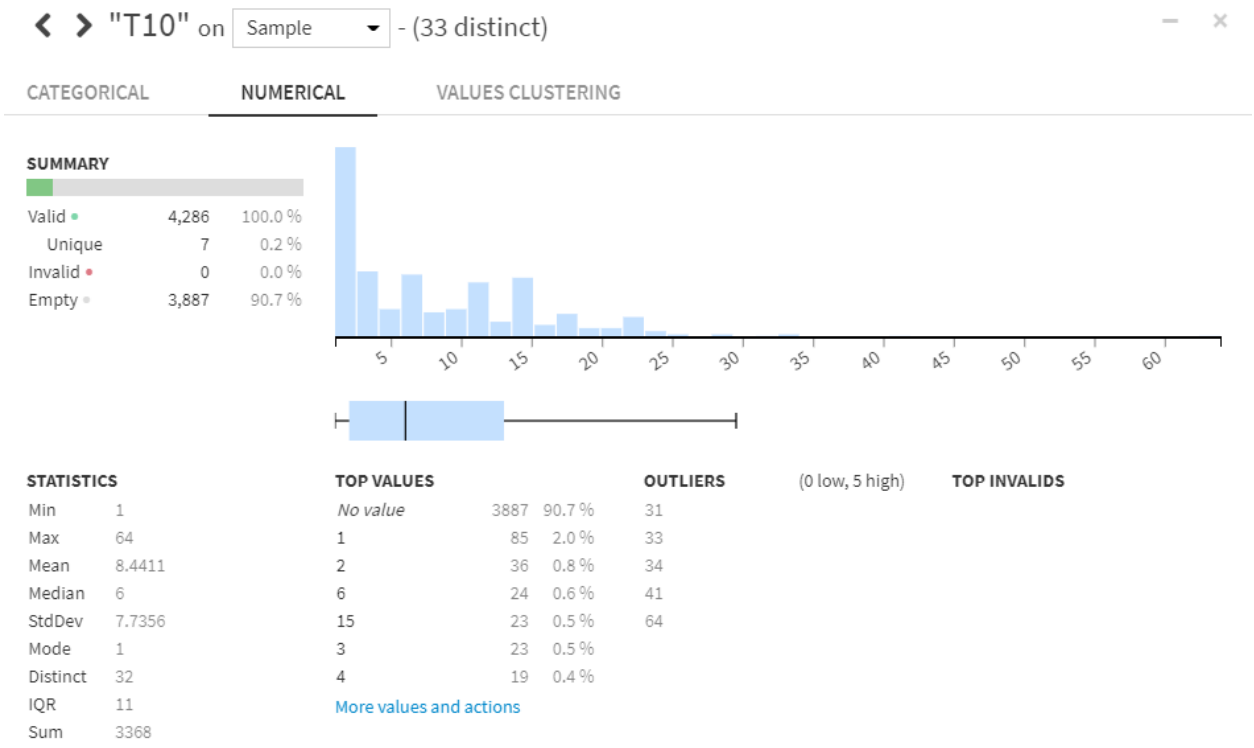


Source: Own elaboration

The variable "Weeks" shows the number of weeks a song has been on the list.

We can see that the most weeks have been 259 and that more than half of them have been more than 5 weeks being the average of weeks 14.58. In this case, the outliers are not interesting for our analysis since we do not consider them to be errors. This variable depends on “Total Streams”

FIGURE 4: T10

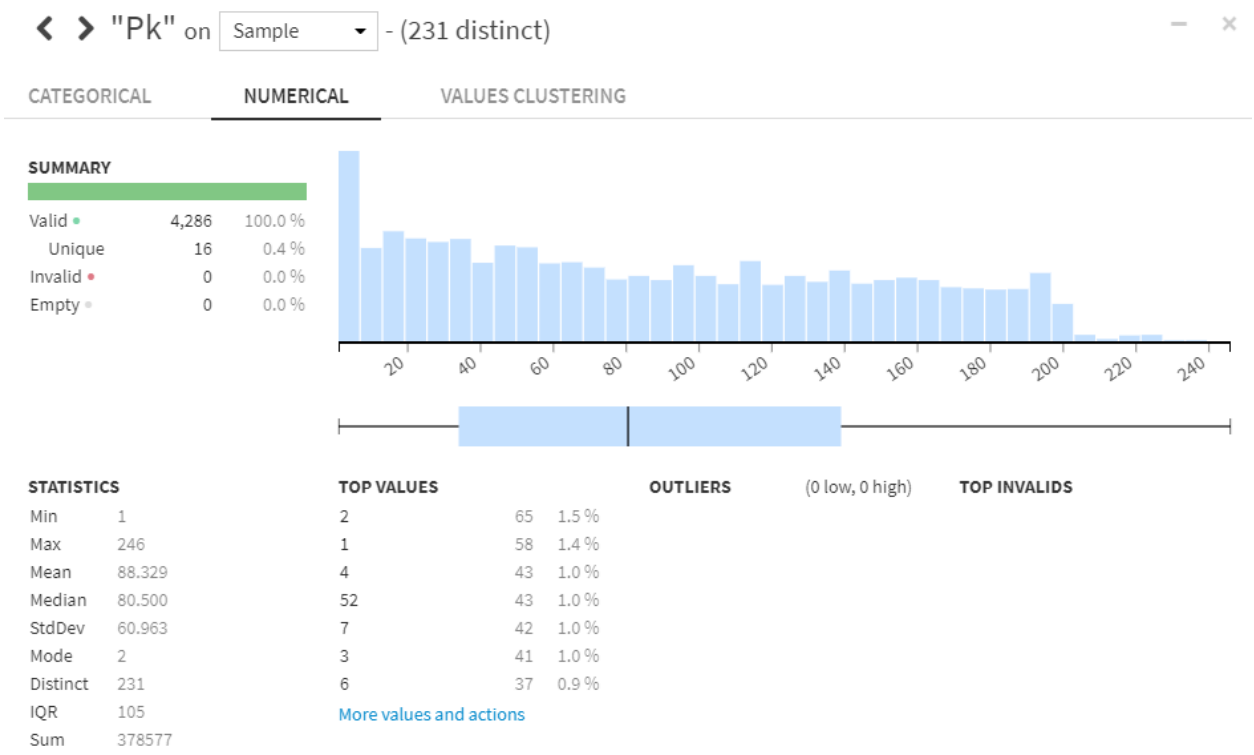


Source: Own elaboration

The "T10" variable shows how many weeks a song has been in the top 10.

In this case we have 3887 empty cells, which represents 90.7% of our sample. We will have to eliminate this variable from our analysis.

FIGURE 5: Pk



Source: Own elaboration

The variable “Pk” shows the top position of the song in the table.

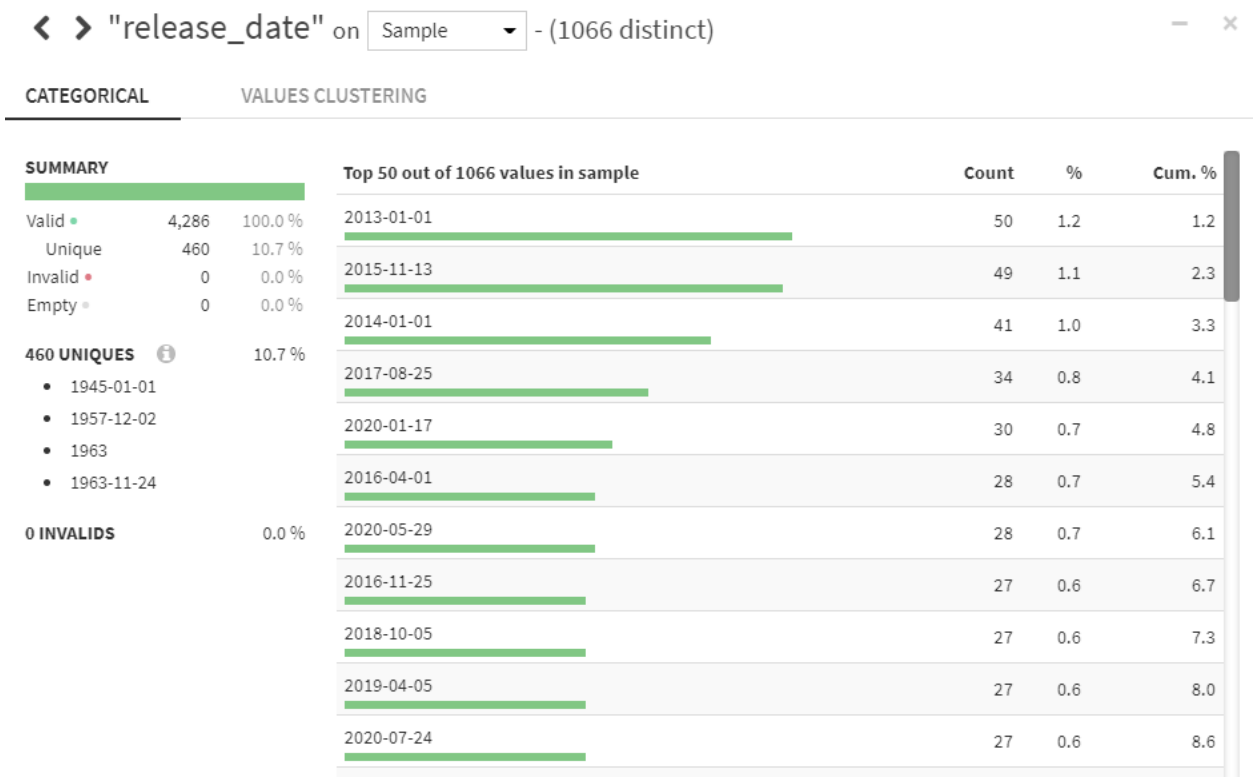
If we analyze the maximum position that a song has acquired in the list, we see that 58 songs have been in position 1, 65 have reached position 2 and 41 in position 3. This means that only 4% of the songs reach the TOP 3 of this list.

X

The variable “X” shows how many weeks the song has been in the Pk position.

It does not make sense to analyze this variable individually since its value is related to the variable Pk. This variable shows us how many times a given song has been in the highest position achieved.

FIGURE 6: release_date

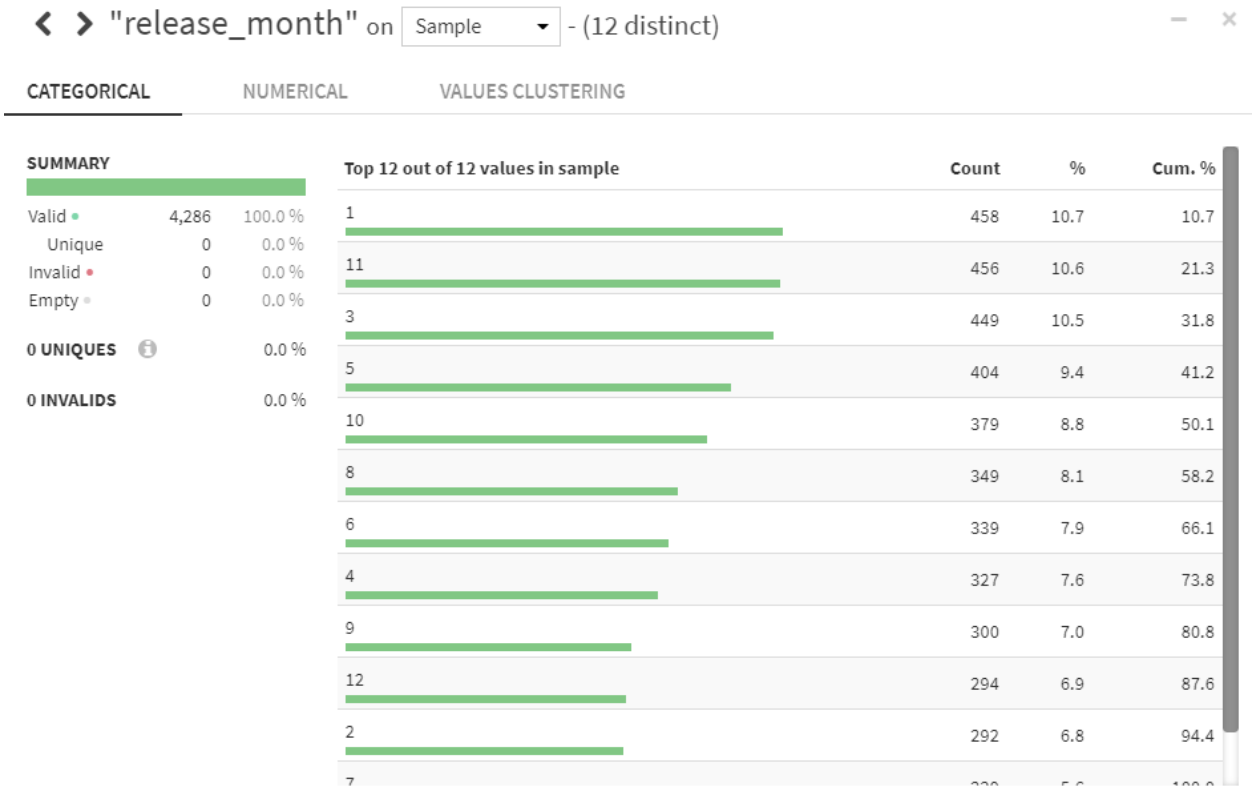


Source: Own elaboration

The variable "release_date" shows the date the song was released.

It looks like 2013 was a year where the database was not capturing release month/day. That is why there are so many records on January 1st. The outliers in this scenario may need to be discarded in case we run a release date / popularity analysis. The songs that are, for example, more than 30 years old may have appeared in modern charts because they were included in a modern movie soundtrack.

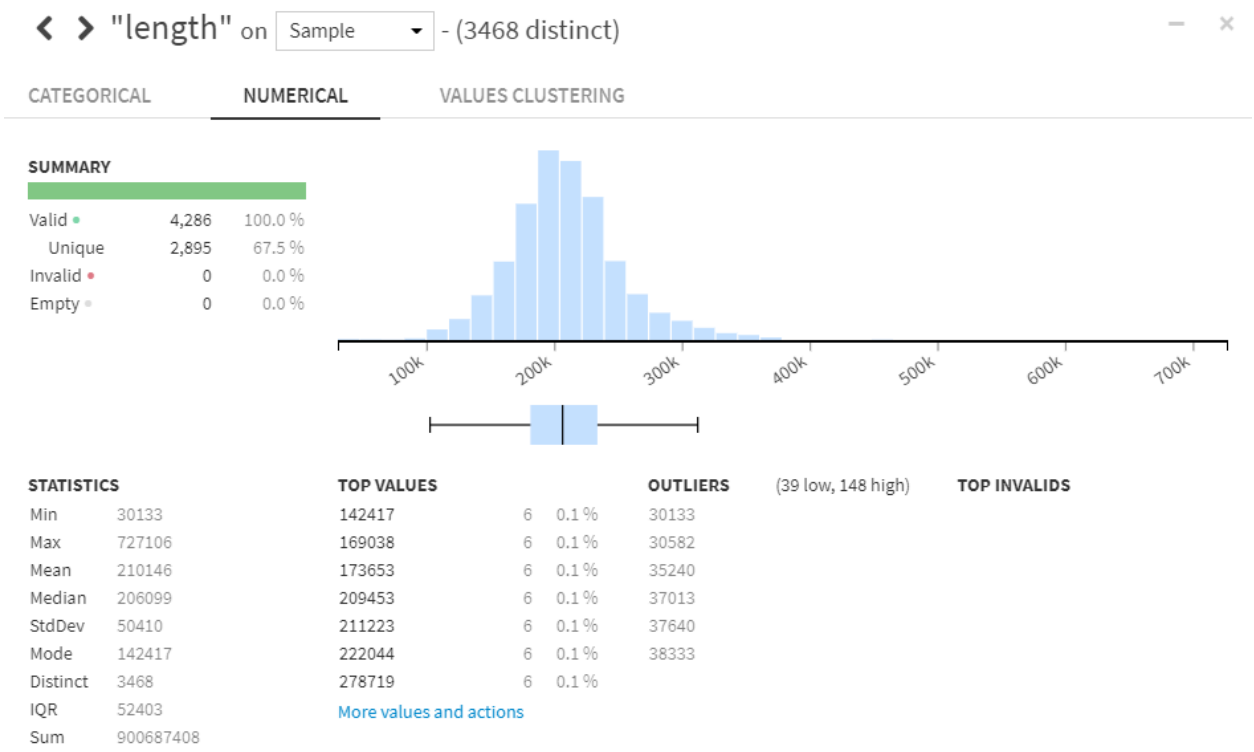
FIGURE 7: release_month_date



Source: Own elaboration

Referring to the month in which the song was released, we first note that 458 songs were released in January, 456 in November, 449 in March and 404 in May. These are the months previous to Christmas and summer, being the summer months when less songs are released.

FIGURE 8: length



Source: Own elaboration

The variable "length" has a normal distribution.

The duration of the 4286 most played songs on Spotify between 2013 and 2021 varies between 30133 milliseconds (half a minute) and 727106 milliseconds(approximately 12 minutes). Regarding outliers and to rule out that this data is wrong, let's filter the data and check the duration of each of these songs on Spotify.

FIGURE 9: LENGTH (2)

name	album	artist	release_date	length
string Text	string Natural lang.	string Text	string Date (unparsed)	string Integer
Mortal Man	To Pimp A Butterfly	Kendrick Lamar	2015-03-16	727106
Not a Bad Thing	The 20/20 Experience - 2 of 2	Justin Timberlake	2013-09-27	688466
Blackstar	Blackstar	David Bowie	2016-01-08	597933
Venice Bitch	Norman Fucking Rockwell!	Lana Del Rey	2019-08-30	577199
4 Your Eyez Only	4 Your Eyez Only	J. Cole	2016-12-09	530253
Give Me Love	+	Ed Sheeran	2011-09-09	526386
untitled 07 | 2014 - 2016	untitled unmastered.	Kendrick Lamar	2016-03-04	496173
Wu Tang Forever (ft. Ghostface Killah, …	YSIV	Logic	2018-09-28	487960
Mirrors	The 20/20 Experience (Deluxe Version)	Justin Timberlake	2013-03-15	484146
FEAR.	DAMN.	Kendrick Lamar	2017-04-14	460573
René	René	Residente	2020-02-27	457591

name	album	artist	release_date	length
string Text	string Natural lang.	string Text	string Date (unparsed)	string Integer
Alfred - Interlude	Music To Be Murdered By	Eminem	2020-01-17	30133
Juice WRLD Speaks From Heaven - Outro	Legends Never Die	Juice WRLD	2020-07-10	30582
Paul - Skit	Kamikaze	Eminem	2018-08-31	35240
Beautiful Trip	Man On The Moon III: The Chosen	Kid Cudi	2020-12-11	37013
raindrops (an angel cried)	Sweetener	Ariana Grande	2018-08-17	37640
Frank's Track	The Life Of Pablo	Kanye West	2016-04-01	38333
Chromatica II	Chromatica	Lady Gaga	2020-05-29	41866
I Love Kanye	The Life Of Pablo	Kanye West	2016-04-01	44826
JACKBOYS	JACKBOYS	JACKBOYS	2019-12-27	46836
Bad Bitch From Tokyo (Intro)	Shoot For The Stars Aim For The Moon	Pop Smoke	2020-07-03	48000
love yourself (interlude)	?	XXXTENTACION	2018-03-16	48422

Source: Own elaboration

After checking that the data is correct, we will assume that those outliers are part of our data due that outliers may be indicative of data that belong to a different population from the rest of the established samples, but in this case the data are correct.

FIGURE 10: LENGTH QUANTILE

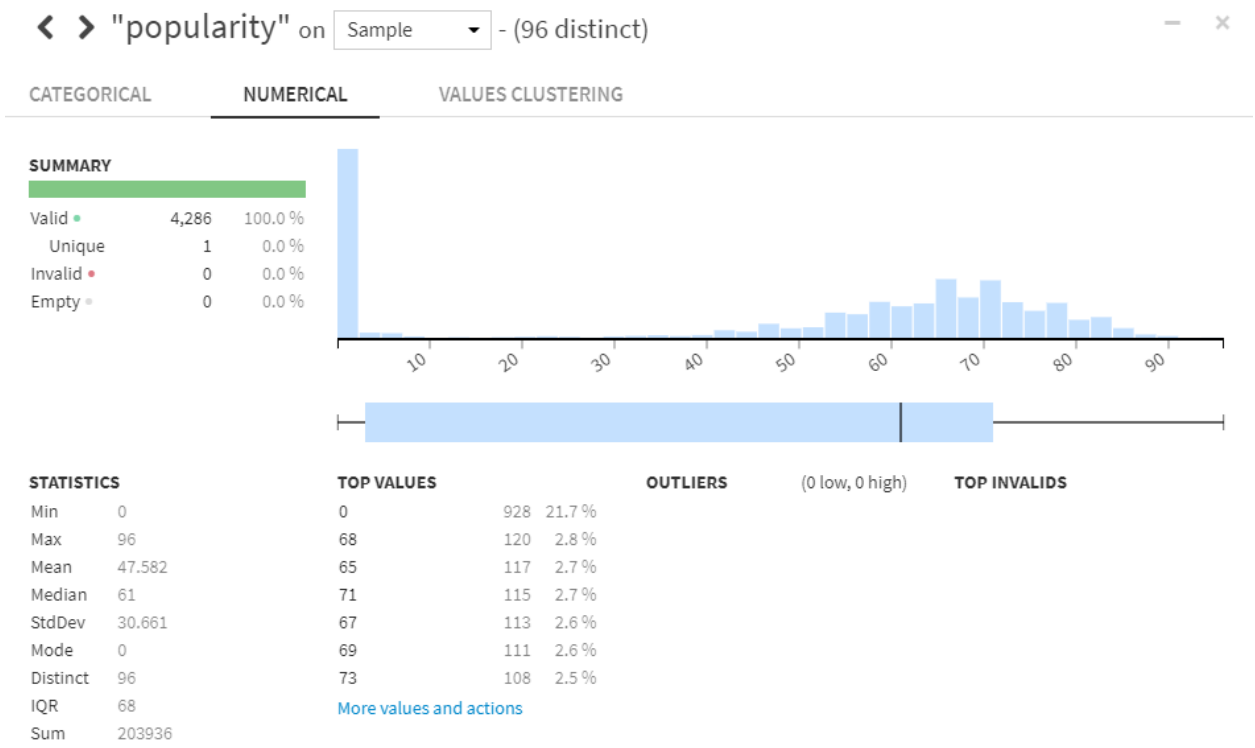
Quantile table

%	Value
1%	105088.25
5%	139449.75
25%	180902
50%	206099
75%	233305
95%	295493
99%	359736

Source: Own elaboration

The mode (the most repeated duration among the songs) is 142417 milliseconds (2 minutes 24 seconds) and only 25% of the songs are longer than 3 minutes 48 seconds (233305 milliseconds). The standard deviation, the dispersion of the data around the mean, is 50 seconds.

FIGURE 11: Popularity

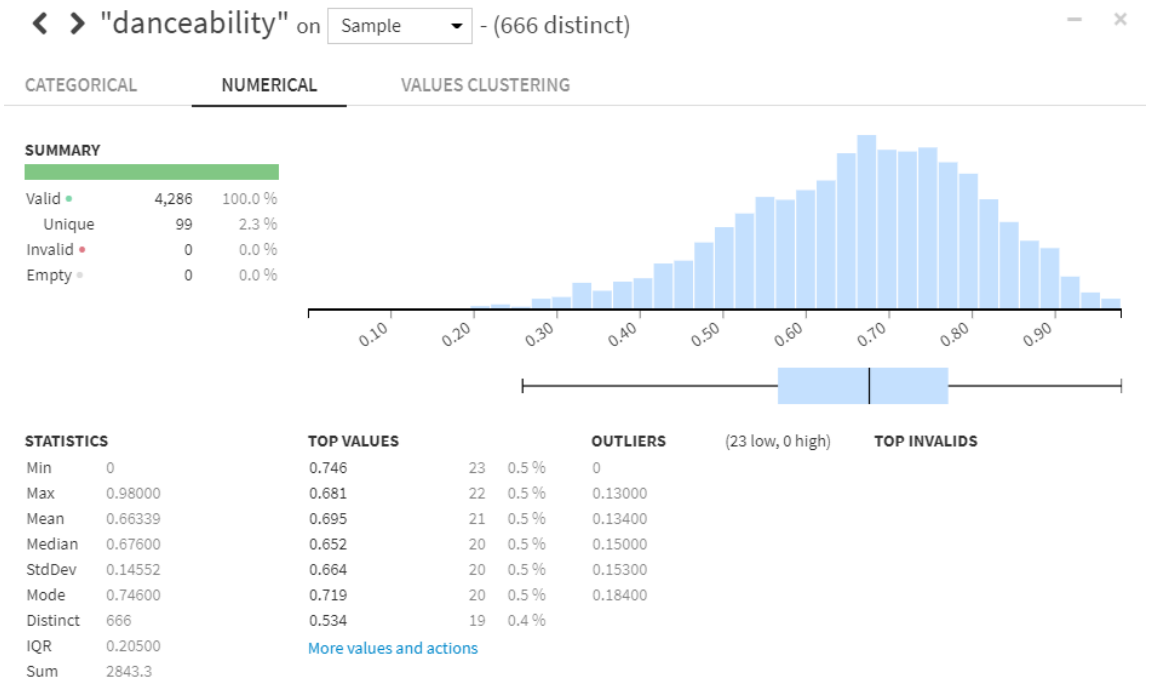


Source: Own elaboration

The "popularity" variable shows the popularity of the track.

there are 928 songs, which is 21.7% of the sample, with popularity 0 (songs that are not popular). This is because popularity is track-based and a measure of how many plays a track received and how recent those plays are. As a result, the song loses popularity over time.

FIGURE 12: Danceability



Source: Own elaboration

FIGURE 13: DANCEABILITY QUANTILE

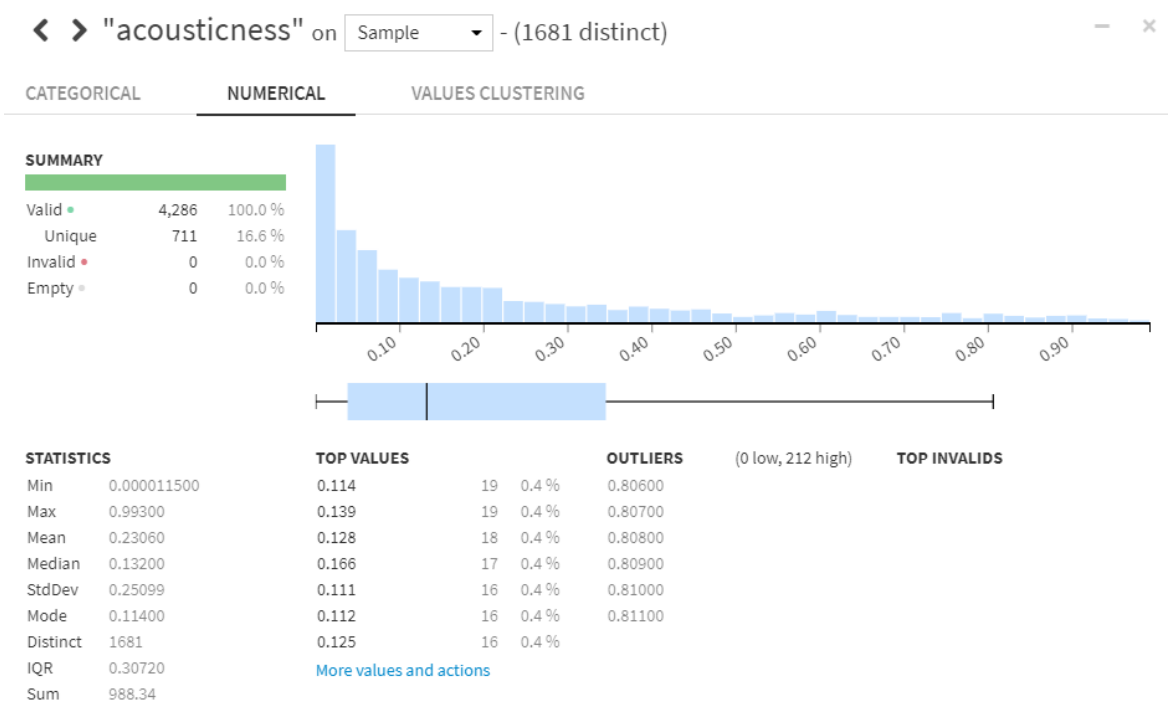
Quantile table

%	Value
1%	0.292
5%	0.399
25%	0.56625
50%	0.676
75%	0.771
95%	0.88275
99%	0.934

Source: Own elaboration

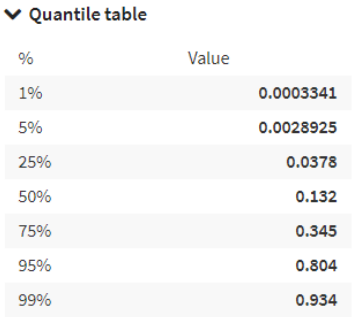
"Danceability" describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable. We can observe a normal distribution around the value 0.66. 75% of the songs have a value bigger than 0.566, which means that they are suitable for dancing. In this case, the outliers are songs that are not danceable but are not errors in the sample, so we are not going to eliminate them.

FIGURE 14: Acousticness



Source: Own elaboration

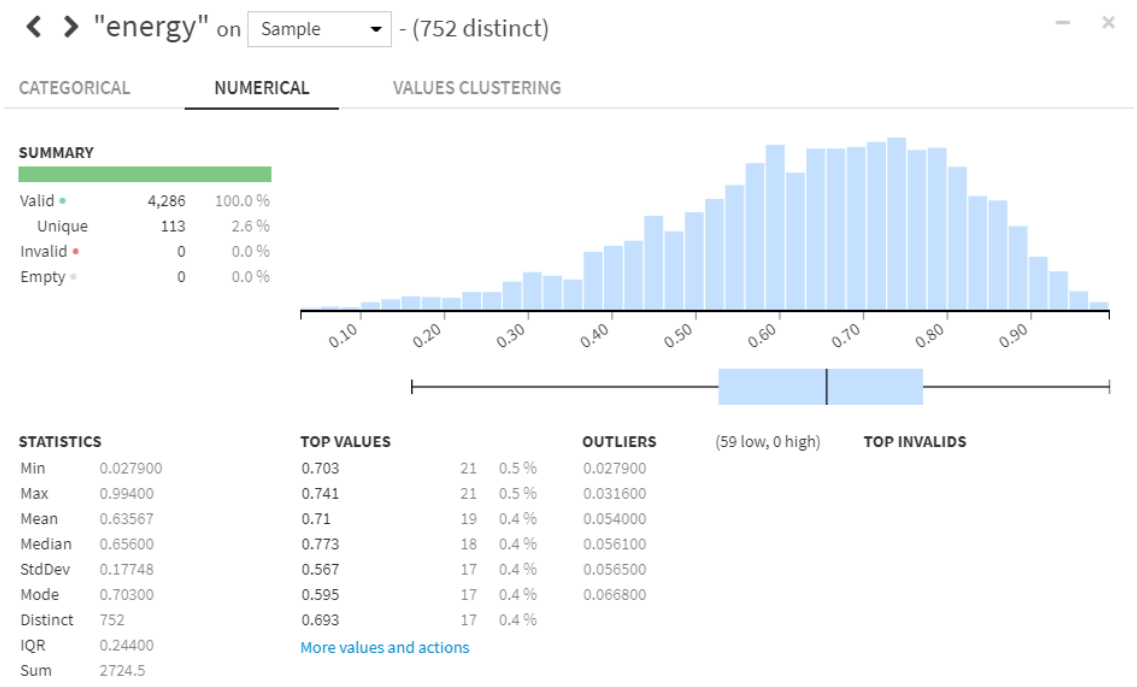
FIGURE 15: ACOUSTICNESS QUANTILE



Source: Own elaboration

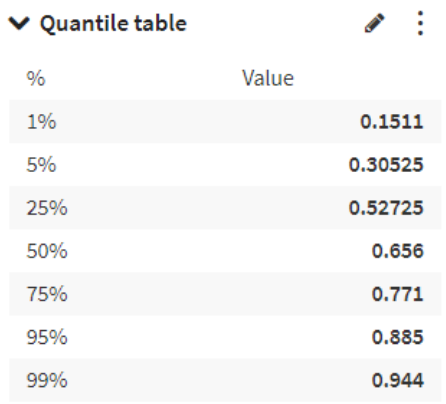
This value describes how acoustic a song is. A score of 1.0 means the song is most likely to be an acoustic one. Acoustic music is music that uses instruments to produce acoustic sounds. The opposite of acoustic music would be electric or electronic music. 75% of our sample has an accoustiness lower than 0.345 (they are not acoustic songs) and only 5% have an accousticness higher than 0.804 (they are acoustic songs).

FIGURE 16: energy



Source: Own elaboration

FIGURE 17: ENERGY QUANTILE

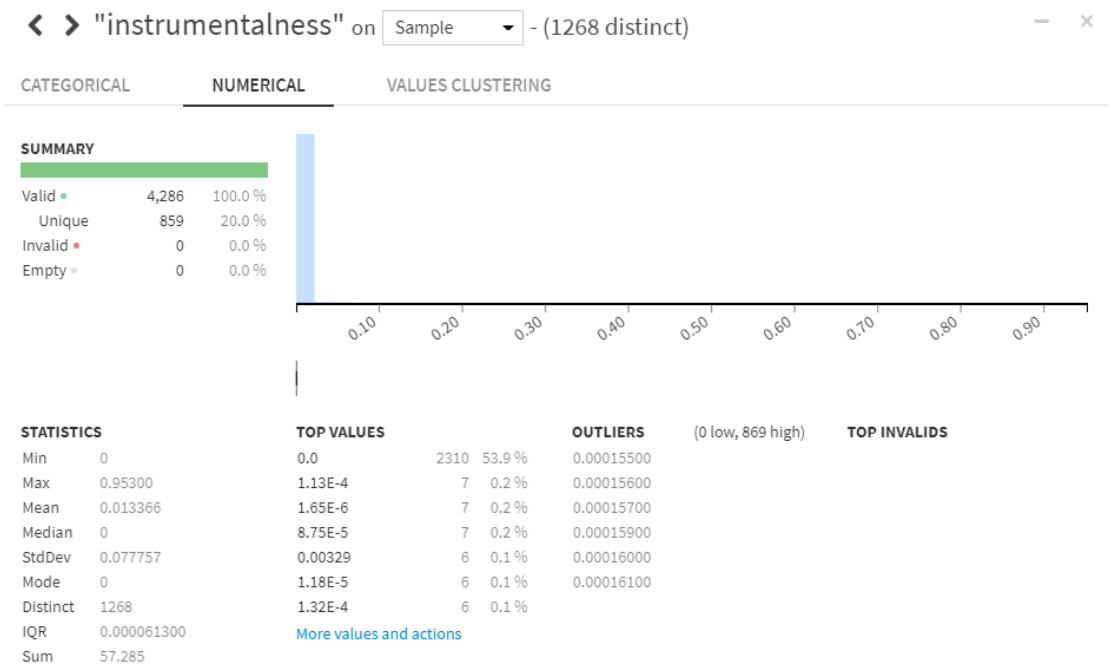


%	Value
1%	0.1511
5%	0.30525
25%	0.52725
50%	0.656
75%	0.771
95%	0.885
99%	0.944

Source: Own elaboration

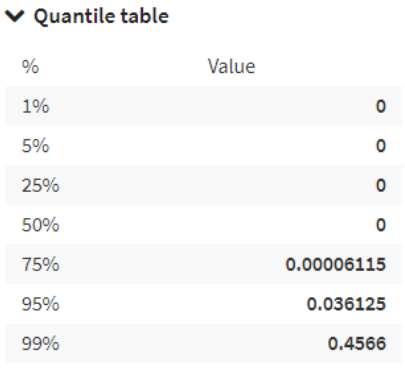
Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. 75% of these songs have an energy above 0.527, which means that 75% of the songs on our list are fast, loud and noisy. The outliers column, in the next image, shows us that there are also quiet songs in our sample, but they are not errors and we will not remove them from our sample.

FIGURE 18: instrumentalness



Source: Own elaboration

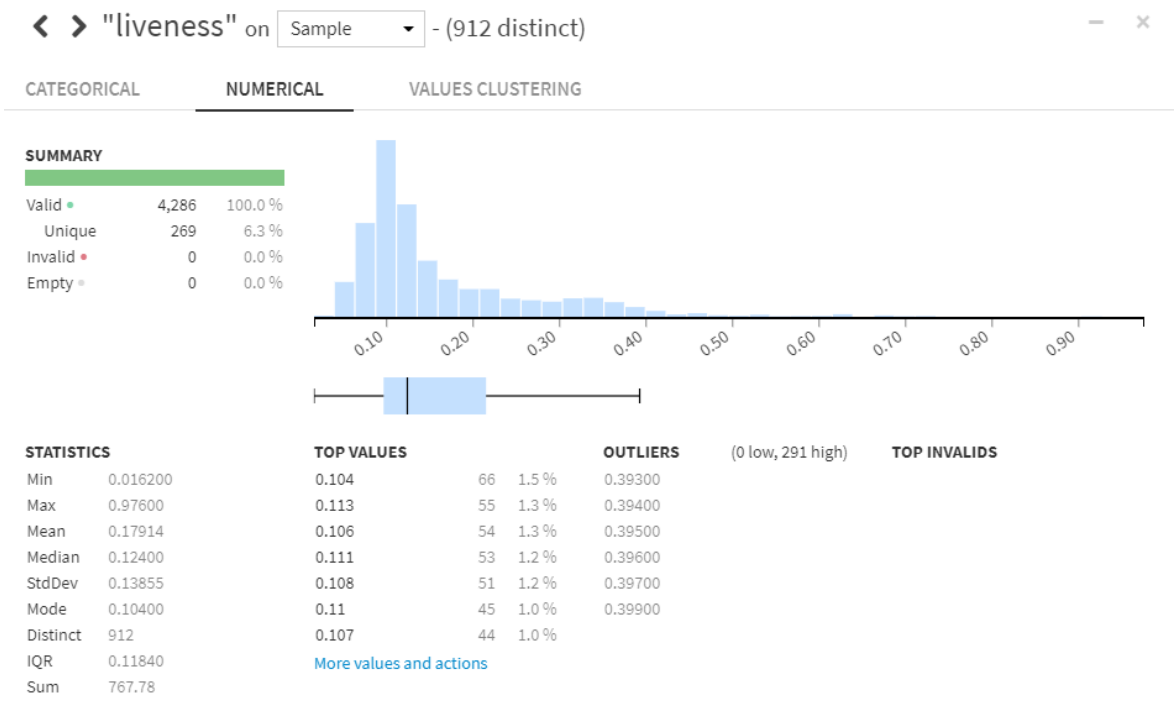
FIGURE 19: INSTRUMENTALNESS QUANTILE



Source: Own elaboration

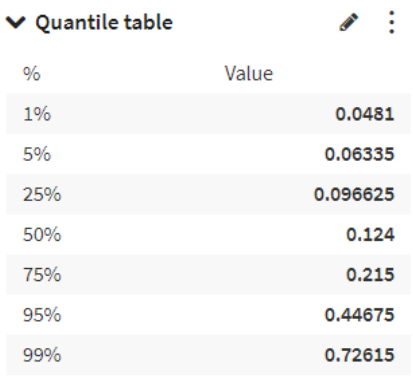
This value predicts whether a track contains no vocals. “Ooh” and “aah” sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly “vocal”. The closer the instrumentals value is to 1.0 , the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.

FIGURE 20: liveness



Source: Own elaboration

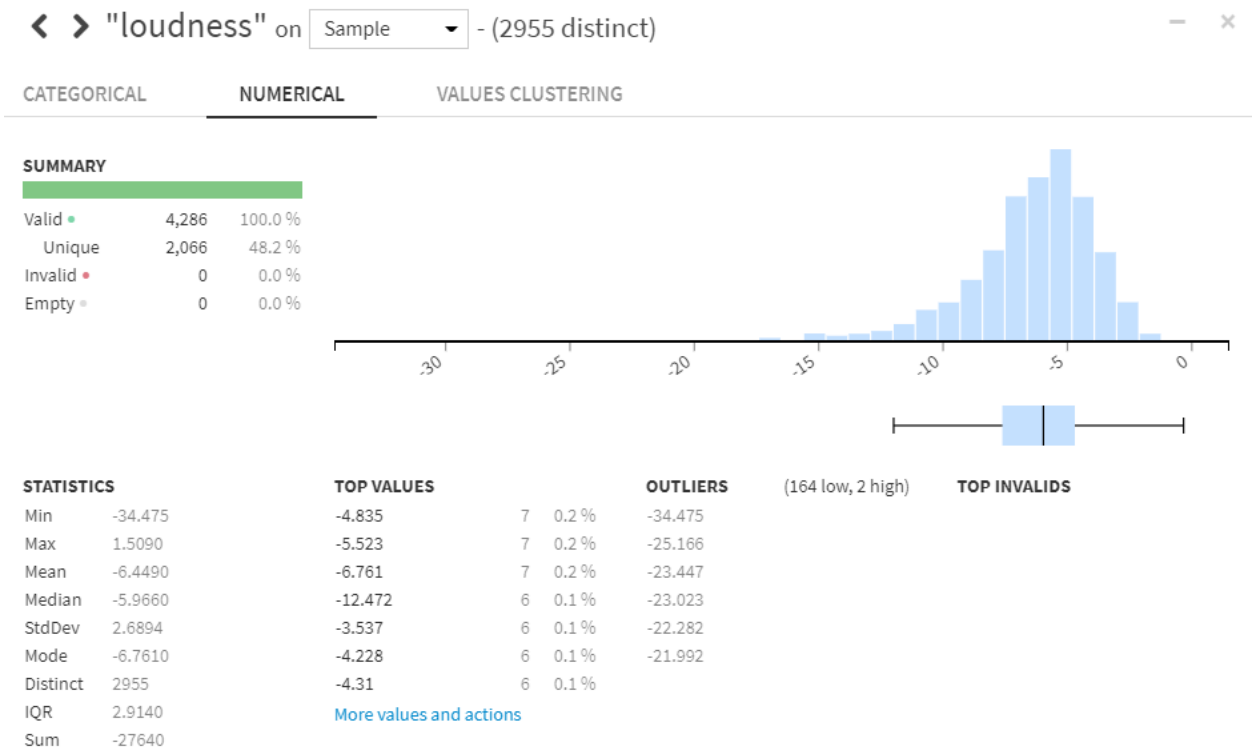
FIGURE 21: LIVENESS QUANTILE



Source: Own elaboration

Liveness detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. As we can see in the following image, half of the songs have a liviness of less than 0.124 and the average is 0.179. Only 1% has a liveness higher than 0.7, which means that, in general, the songs in our sample are not performed live.

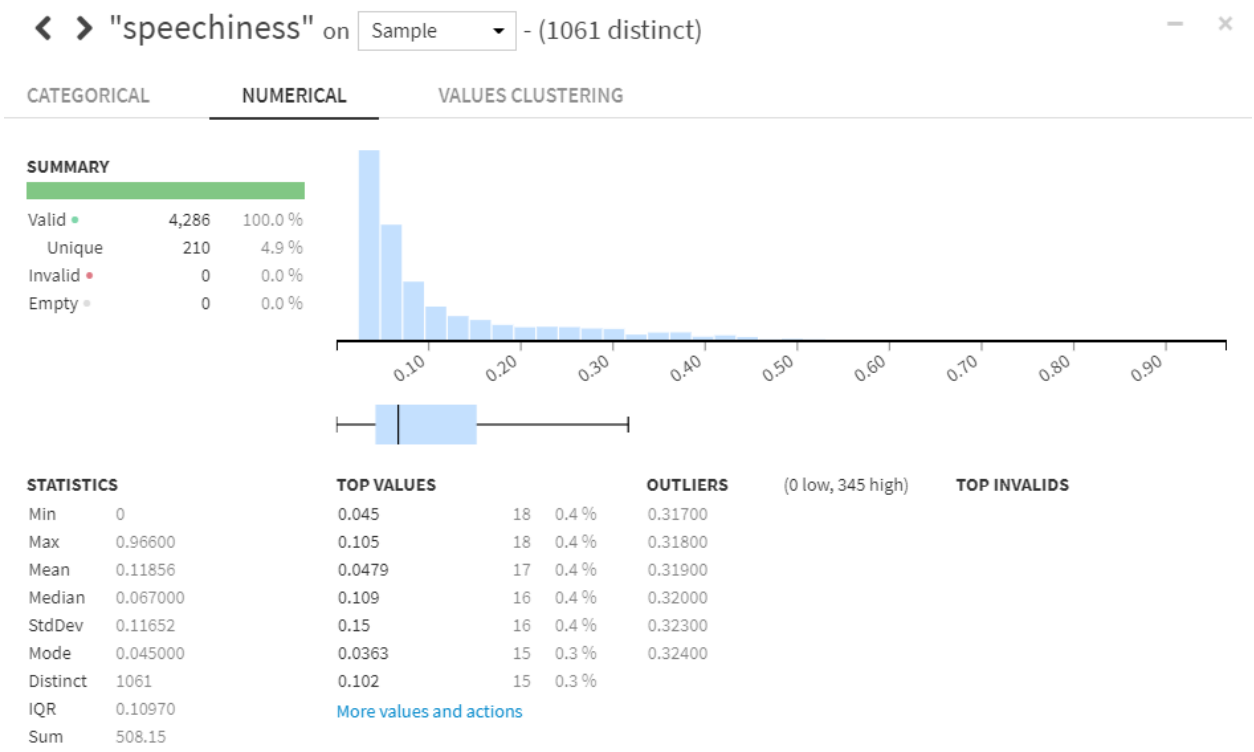
FIGURE 22: loudness



Source: Own elaboration

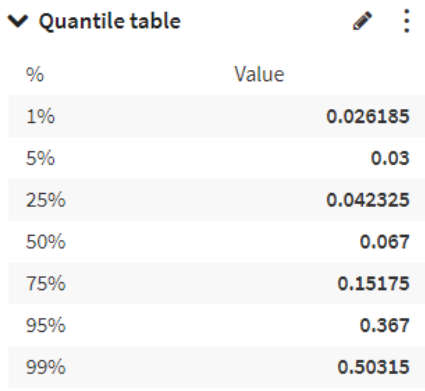
The mean decibel per song is -6.449 and the median -5.966. The mean and median are quite similar so this variable has a distribution very similar to a normal distribution.According to the description of the variables provided by Spotify, it is normal for a song to have between -6000 and 0.0 decibels. Looking at the histogram, we are going to eliminate from the sample those songs that have less than -20 and more than 0 decibels.

FIGURE 23: Speechiness



Source: Own elaboration

FIGURE 24: SPEECHINESS QUANTILE

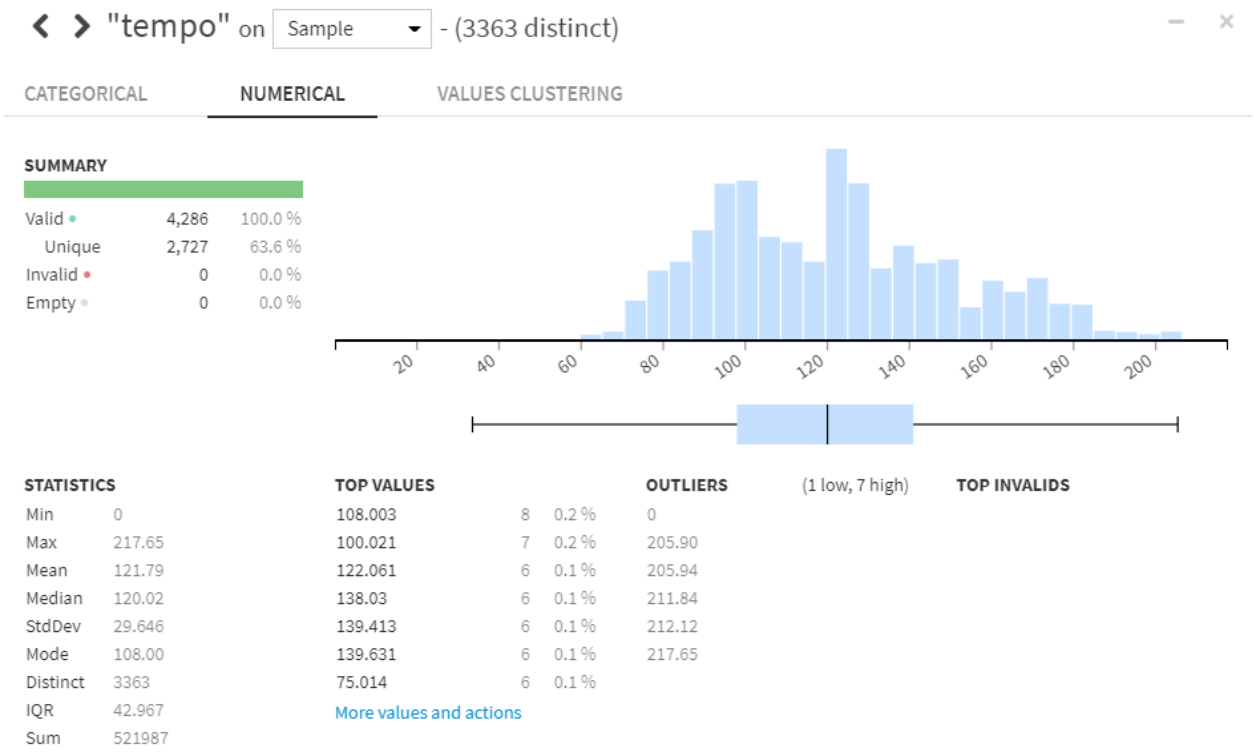


Source: Own elaboration

The variable “Speechiness” detects the presence of spoken words in a track, from 0.0 to 1.0.

The songs on average have a speechiness of 0.11. Values below 0.333 most likely represent music and other non-speech-like tracks. Only 0.3% of the tracks are probably made entirely of spoken words.

FIGURE 25: tempo

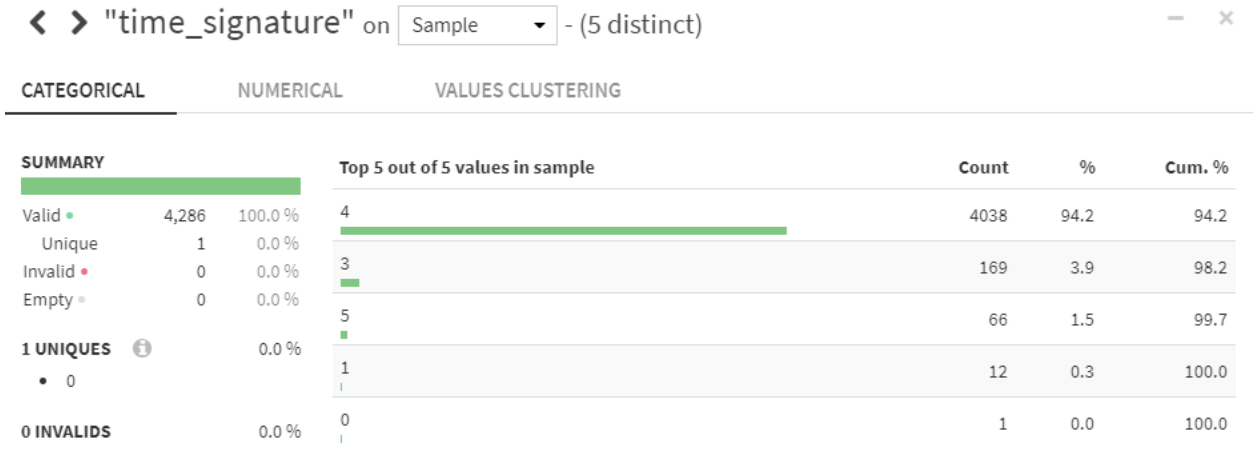


Source: Own elaboration

The variable “tempo” shows the overall estimated tempo of a track in beats per minute (BPM).

The average BPM of the songs in this list is 121.79. according to wikipedia and considering that the music genre that has less BPM is Chill Out (70-100 BPM) and the one that has more is hardcore Techno (165-220 BPM), we are going to eliminate the values that are below 70 BPM.

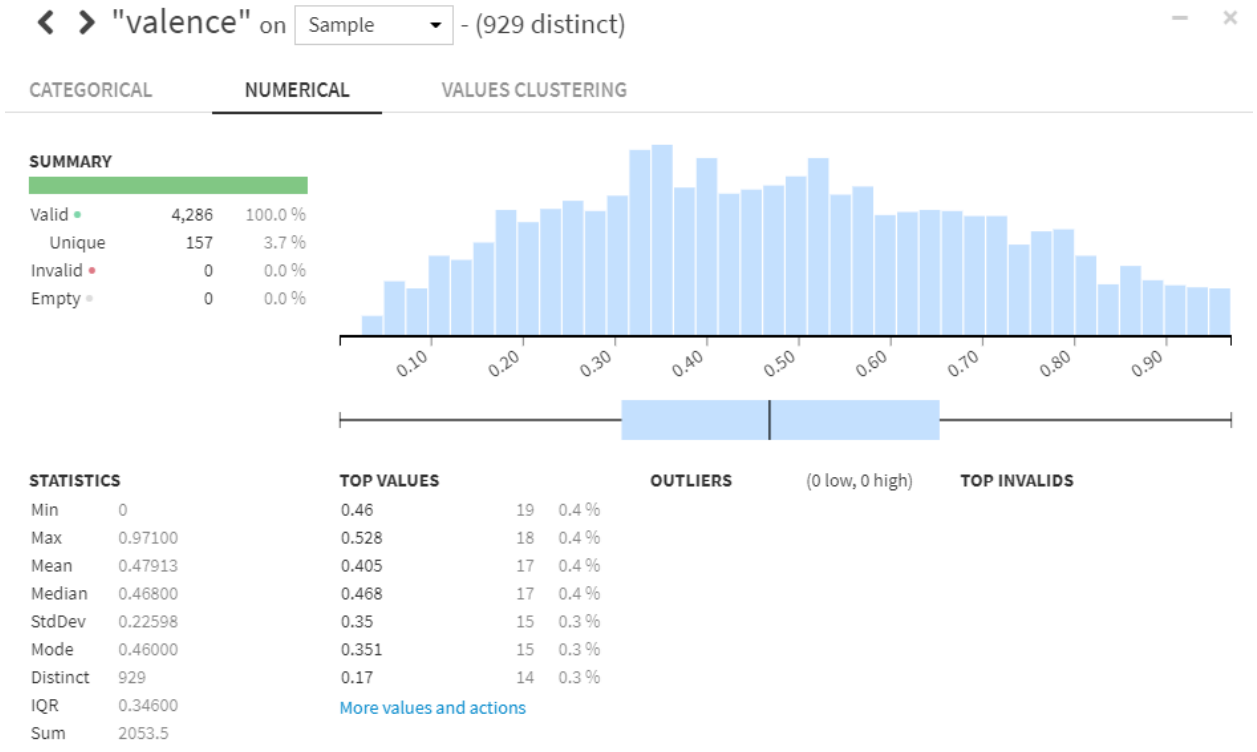
FIGURE 26: time_signature



Source: Own elaboration

"time_signature" is an estimated overall time signature of a track. 94.2% of the songs are written in four-beat measures. Due to the nature of this variable, we do not consider it to have outliers.

FIGURE 27: valence



Source: Own elaboration

FIGURE 28: VALENCE QUANTILE

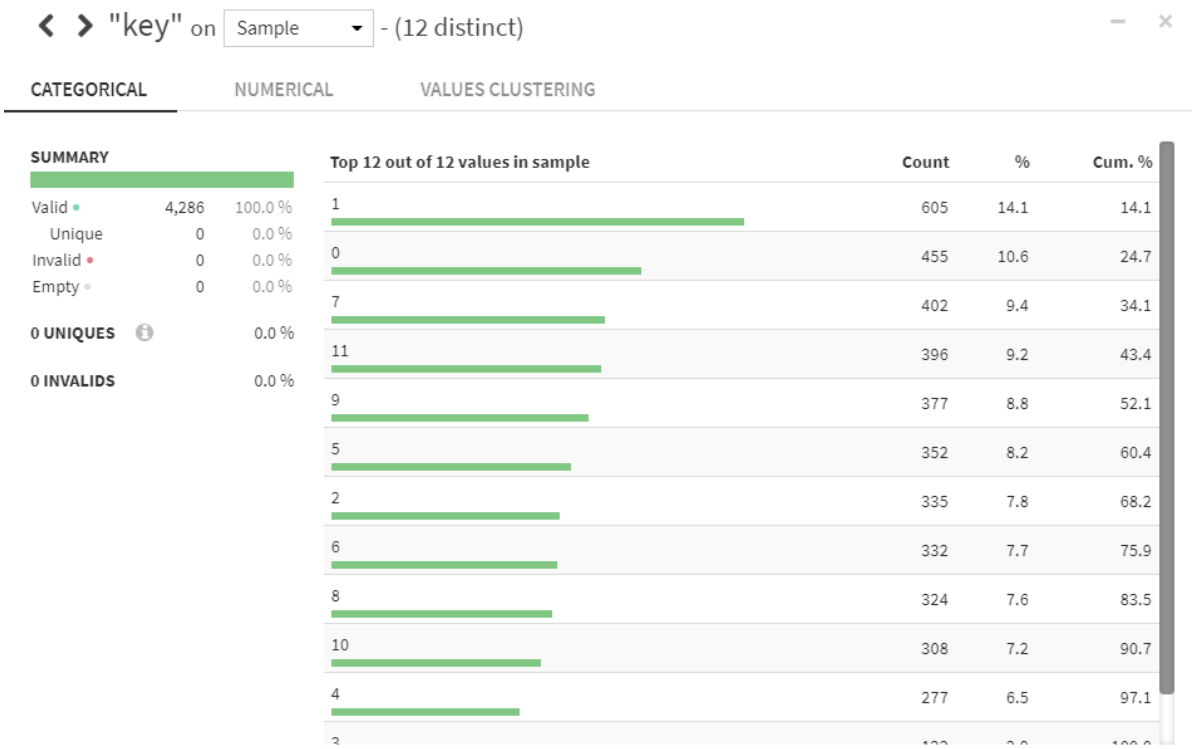
Quantile table

%	Value
1%	0.05838
5%	0.124
25%	0.307
50%	0.468
75%	0.65275
95%	0.866
99%	0.9483

Source: Own elaboration

The valence describes the musical positiveness conveyed by a track. As we can see, in this sample we find very positive songs and very negative songs, with an average of 0.479 (neither positive nor negative).

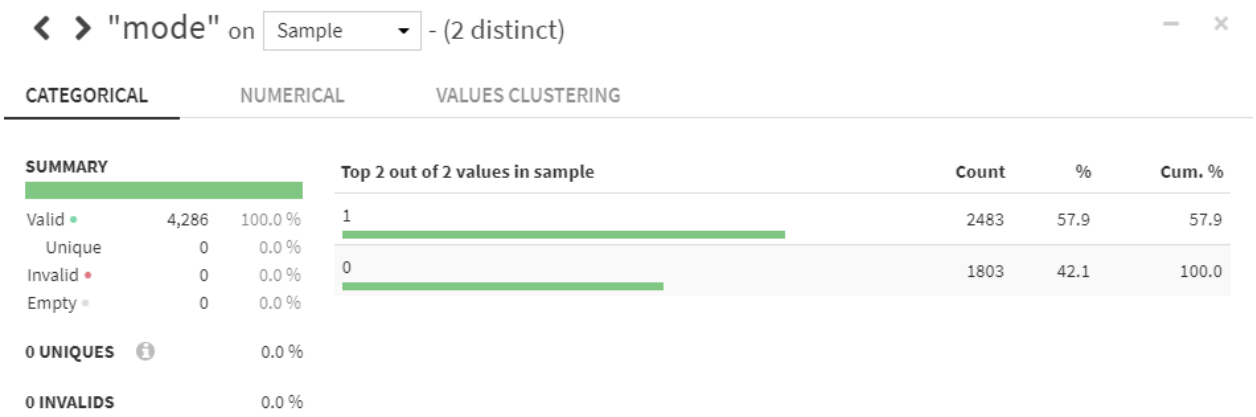
FIGURE 29: key



Source: Own elaboration

52% of the songs are written in the key of C, C#/Db, F, A and G.

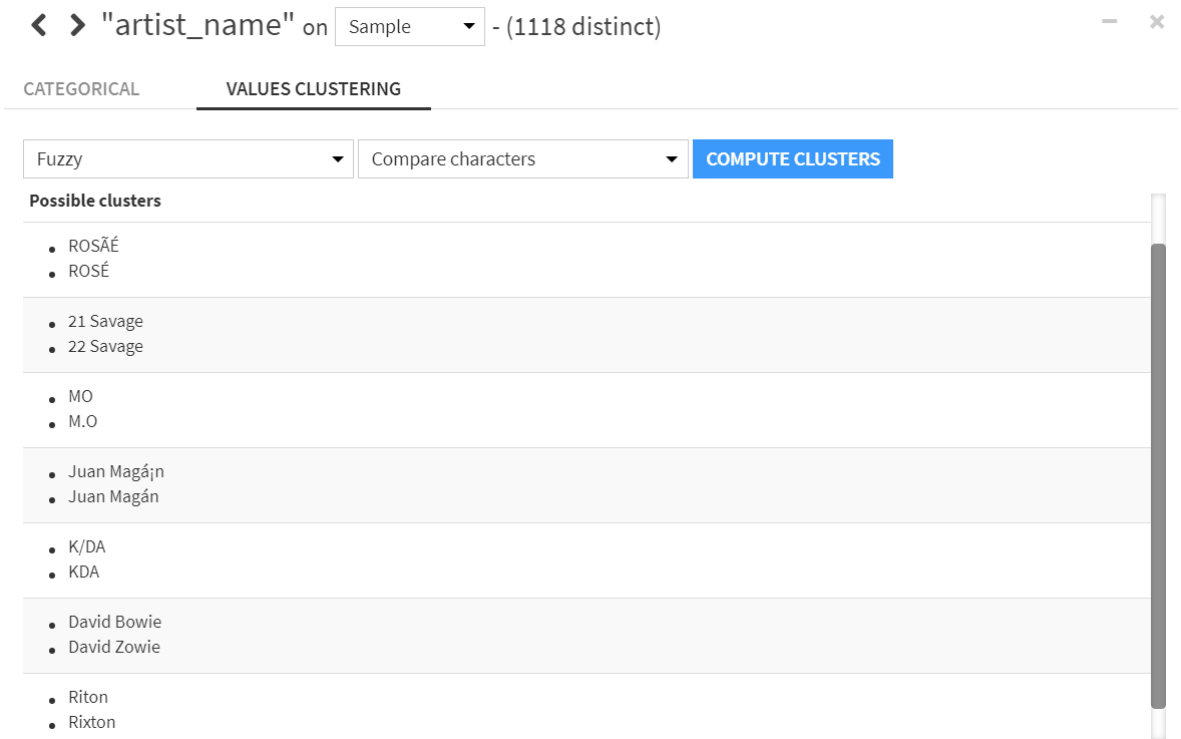
FIGURE 30: mode



Source: Own elaboration

57.9% of the songs are written in a major key and 42.1% in a minor key.

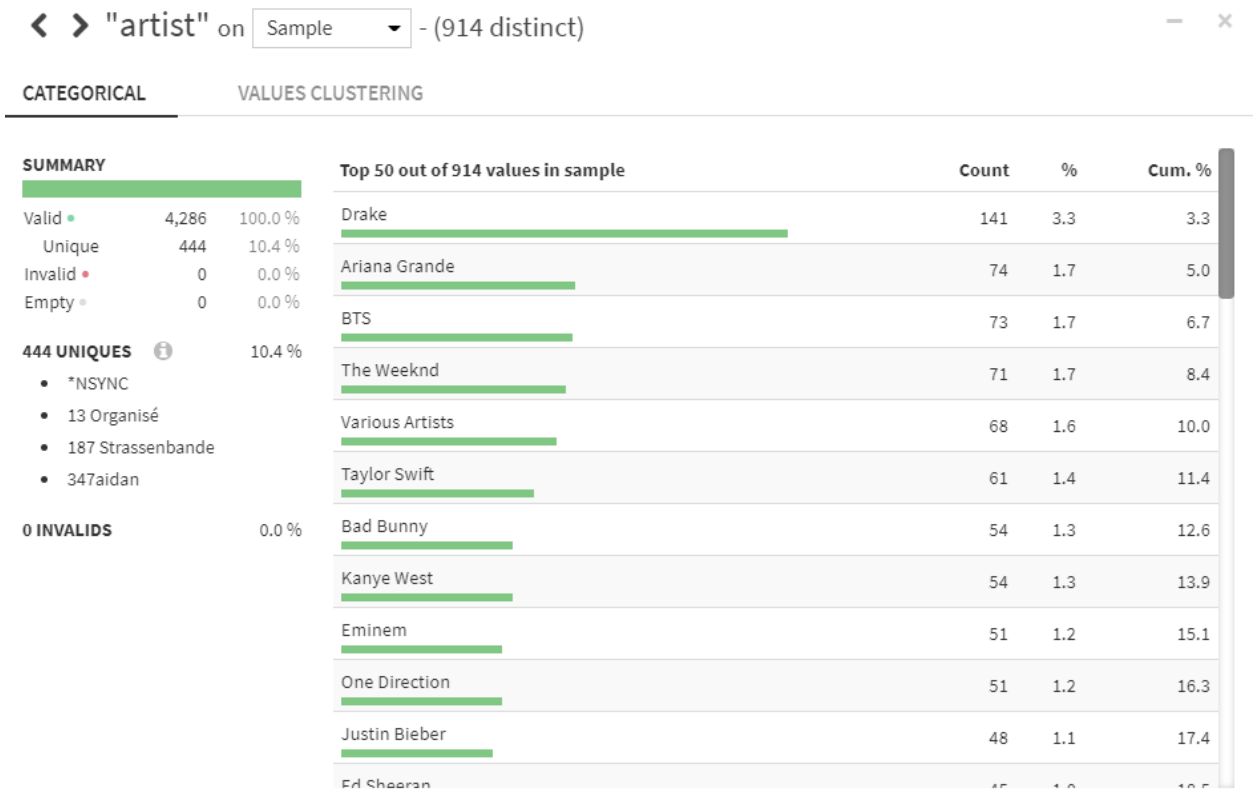
FIGURE 31: artist_name



Source: Own elaboration

To check that the quality of the data is good, we created clusters with the values of the artist_name column and found data that is not clean: the same name written in two different ways. So we have cleaned those names that were incorrect.

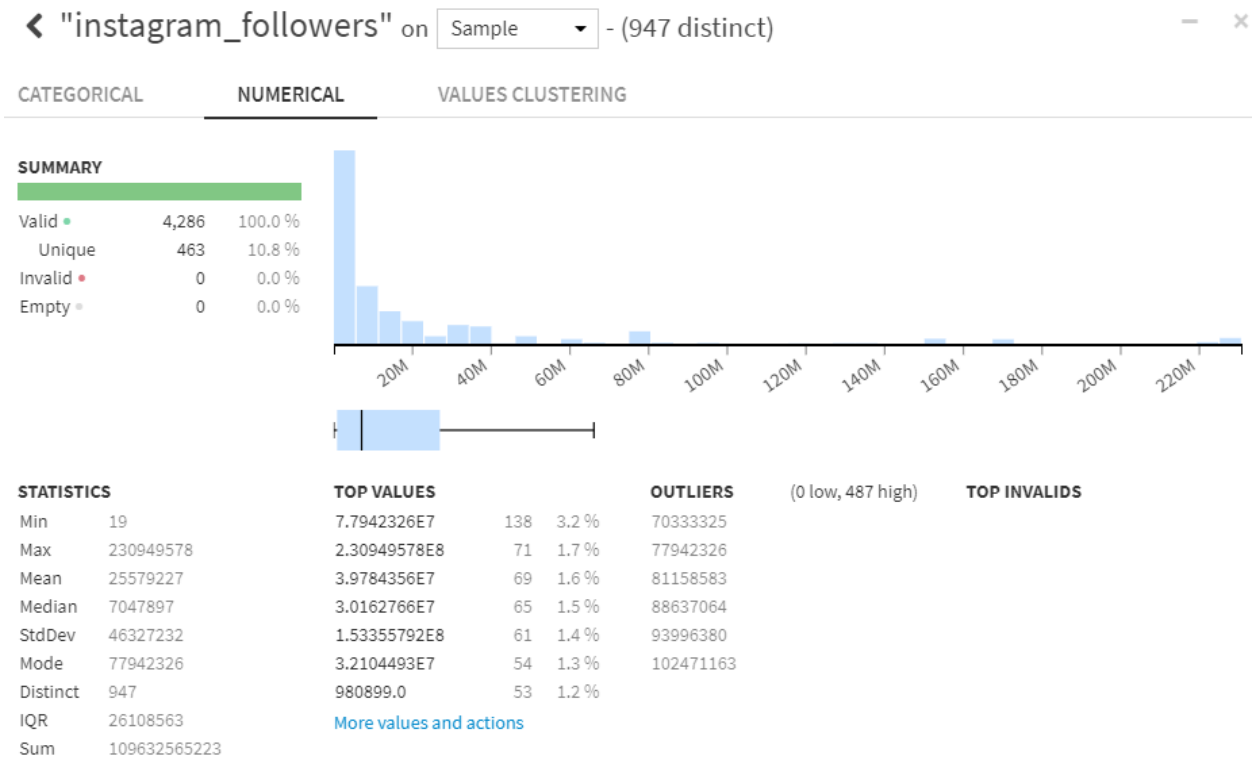
FIGURE 32: ARTIST



Source: Own elaboration

With the names cleared, let's look at their frequency. It stands out that the 4286 songs with the most plays on Spotify belong to only 444 artists. It highlights the artist Drake who has 141 songs, Ariana Grande with 74 and BTS with 73. We have to look at the data to see if the number of times Drake appears in the list is correct or there are duplicate values.

FIGURE 33: Instagram_followers



Source: Own elaboration

We have a total of 4282 data coming from instagram. We checked the outliers to verify those numbers were correct and due that the outliers are real numbers, we are not going to remove them from our sample.

UNIVARIATE ANALYSIS SUMMARY

After examining the data obtained and applying the corresponding methodological analyses to each variable and set of variables, we will proceed to show the results derived from this process. In the first sub-section we will highlight the most significant characteristics of the set of songs belonging to the study and we will group these characteristics according to artist information, song information and audio features.

Regarding the artist's information, It stands out that the 4286 songs with the most plays on Spotify belong to only 444 artists. It highlights the artist Drake who has 141 songs, Ariana Grande with 74 and BTS with 73.

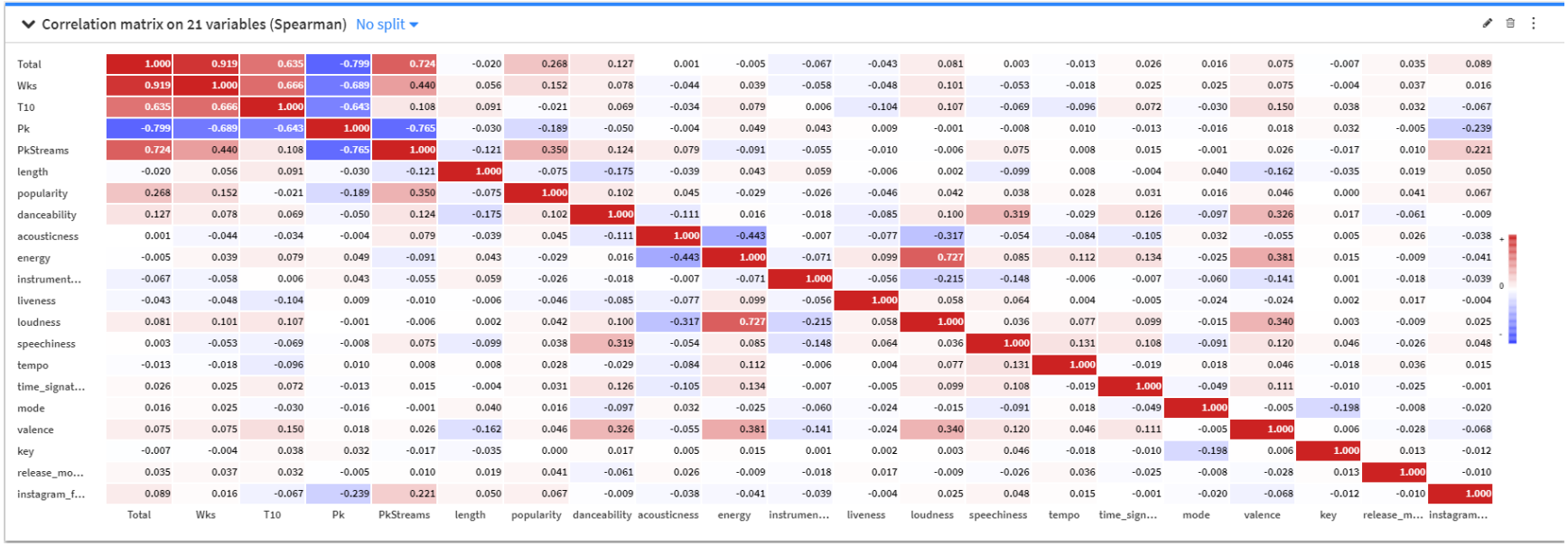
Regarding the information on the songs and as a result of the univariate analysis carried out and taking into account that the results shown are considered on the valid percentage (excluding blanks), we can observe that the difference between the song with the most and least plays is very large, almost 50% of the songs have been on the list more than 5 weeks and only 4% have reached the TOP 3 of the list. It is

also worth noting that the months prior to summer and Christmas are the months when the most songs are released and the summer months when the least number of songs are released.

In relation to the information of the songs, we see that only 25% of the songs are longer than 3 minutes 48 seconds minutes and that the mean is 3 minutes 30 seconds with a standard deviation of 50 seconds. It also stands out that 75% of the songs in our sample are danceable, with an average of 121.79 BPM, -5.966 decibels and generally the songs in our sample have no audience in the recording. Regarding energy, 75% of the songs are fast, loud and noisy and are neither positive nor negative. We also see that 94.2% are written in four-beat measure and in the key of C, D, F, A and G.

BIVARIATE ANALYSIS

FIGURE 34: Correlation matrix

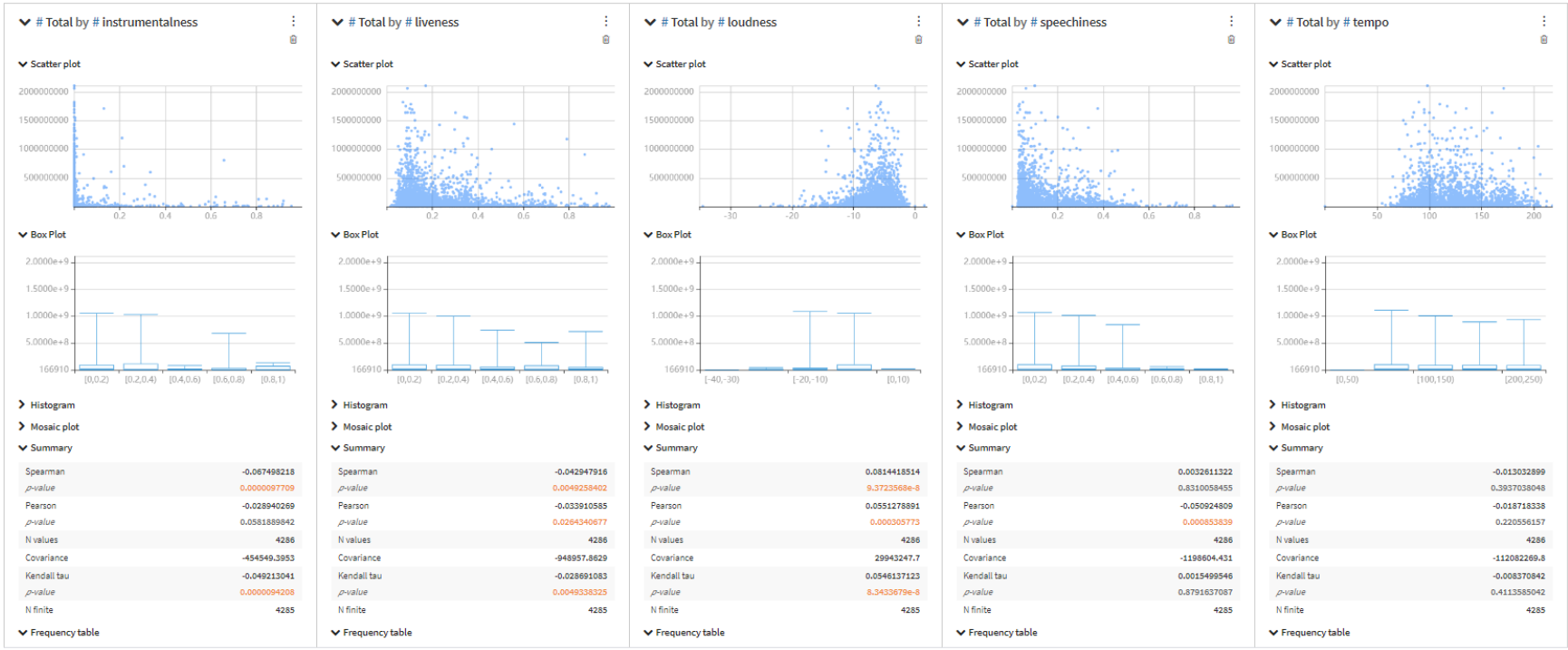


Source: Own elaboration

FIGURE 35: CORRELATION 1

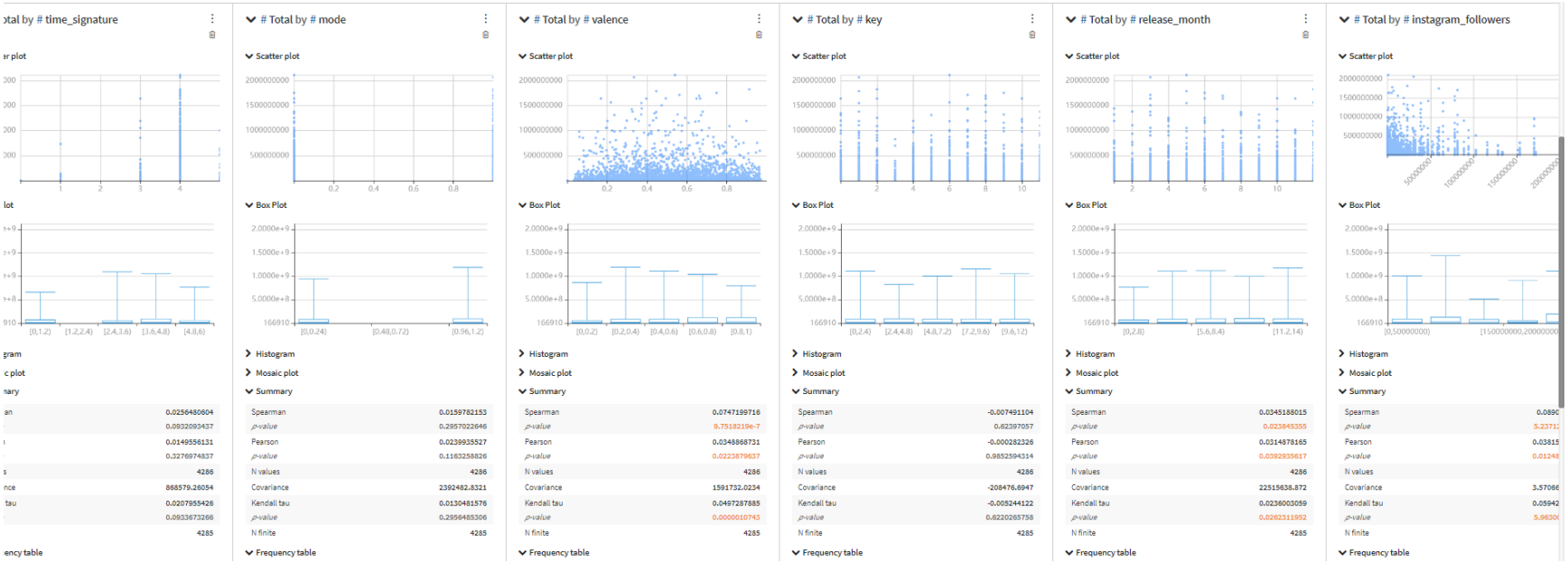
Source: Own elaboration

FIGURE 36: CORRELATION 2



Source: Own elaboration

FIGURE 37: CORRELATION 3



Source: Own elaboration

BIVARIATE ANALYSIS SUMMARY

From the Spearman's correlation table we can conclude that the variables Wks, T10, Pk, PKStreams and Total have a high level of correlation with each other. We can also observe that the energy variable has a correlation of 0.727 with the loudness variable. This means that there exists a high monotone correlation between those two variables.

On the other hand, with the data obtained from the Spearman coefficients we can reject that:

H0: The variable "danceability" does not affect the total number of Streams.

H0: The variable "popularity" does not affect the total number of Streams.

H0: The variable "instrumentalness" does not affect the total number of Streams.

H0: The variable "liveness" does not affect the total number of Streams.

H0: The variable "loudness" does not affect the total number of Streams.

H0: The variable "Speechiness" does not affect the total number of Streams.

H0: The variable "Valence" does not affect the total number of Streams.

H0: The variable "release month" does not affect the total number of Streams.

H0: The variable "instagram-followers" does not affect the total number of Streams.

But the correlation coefficients are very small which shows that the relationship is very small (almost non-existent).

Outliers analysis and strategy

Proposed strategy to treat missing and erroneous data

To deal with missing data, erroneous data, outliers and those variables that can cause problems in our models, we will follow the following strategy:

We will reject from our study those variables that have more than 20% of empty cells, those with more than 20% of invalid values, duplicate variables, variables whose value depends on the total number of Streams and variables that cause heteroscedasticity.

Data Preparation

Data Cleansing process description

With the data obtained from the SWOBN table, we have carried out a data cleaning and transformation process. First, we treated the special characters, converted the release_date column to date and extracted the month and converted the variable X, which was a string, to numeric. After creating clusters with the artist names, we realized that some of the names were misspelled and cleaned up the artist

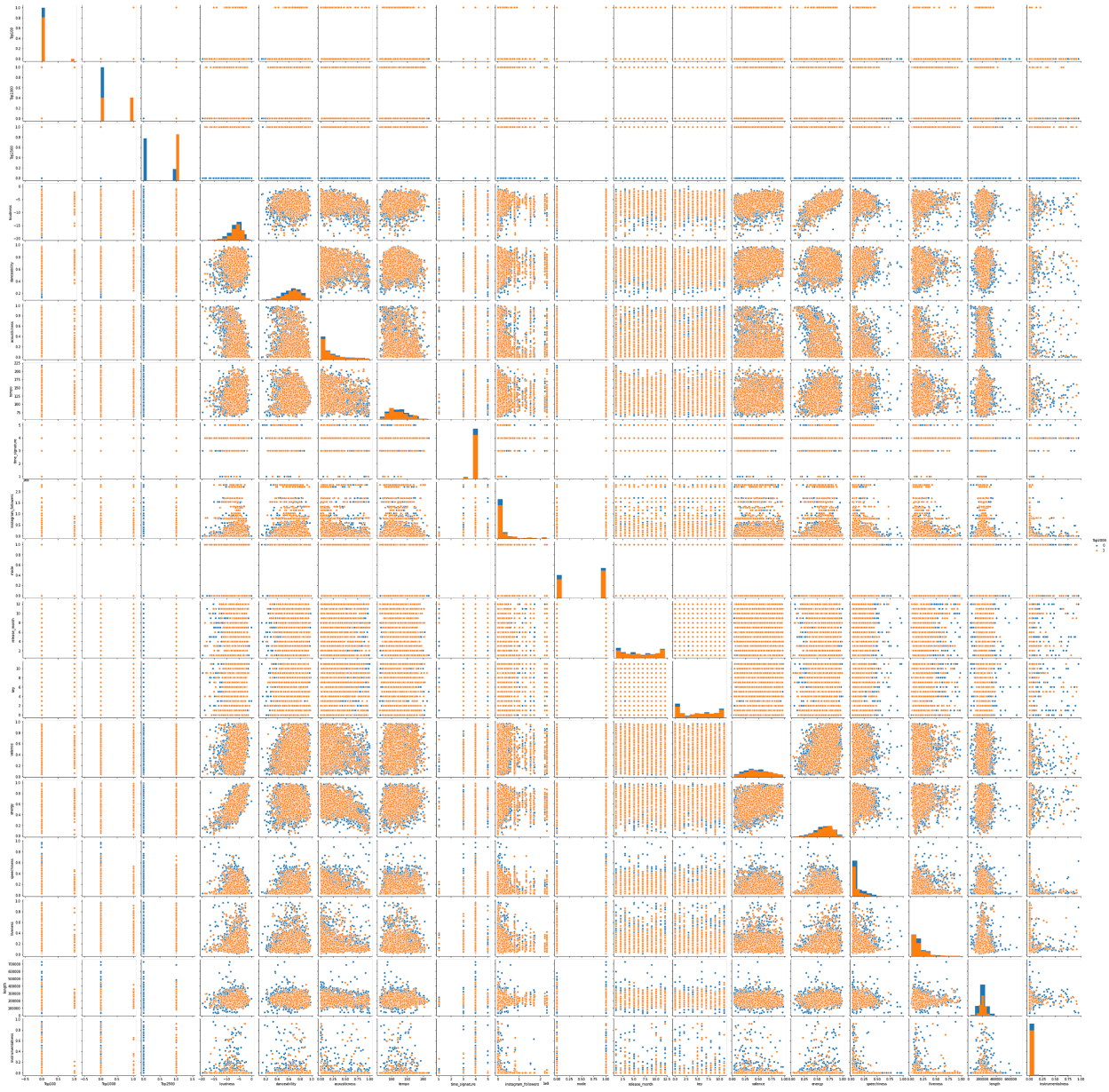
names column. Next, we proceeded to analyze each of the variables and performed the following cleaning processes.

- Elimination of variables:
  - Variables with more than 20% of empty cells are rejected from the study
    - T10: 90.7% are empty cells
    - X: 96% are empty cells
  - Duplicate variables are rejected from the study
    - Artist's name
    - Song's name
    - artist_and_title

  - Variables directly related to the total number of Streams (variables whose output depends on the total number of Streams) are rejected from the study to not cause a data leaking distorting our predictions .
    - Pk_streams
    - Weeks
    - Pk
    - Popularity

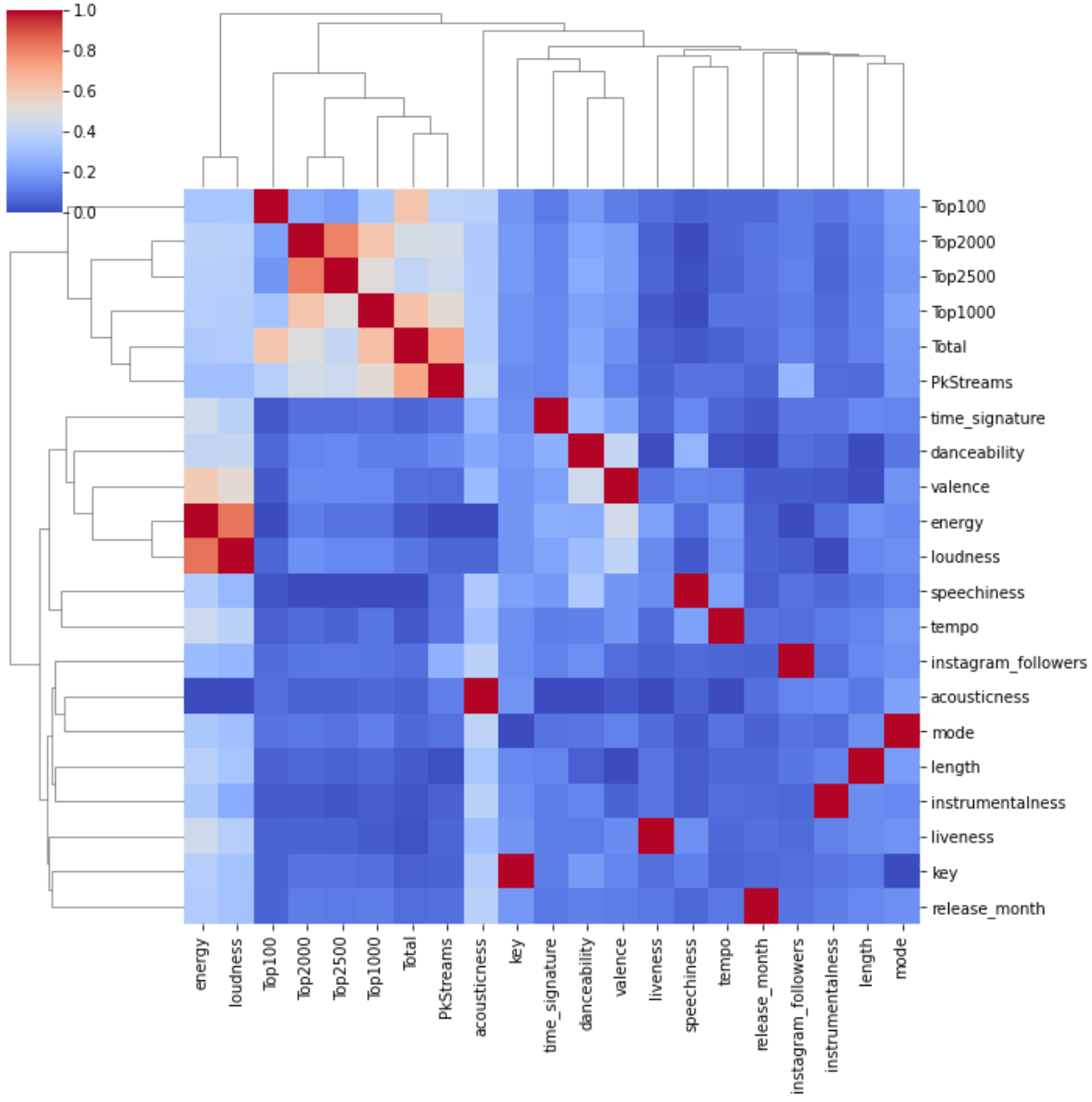
  - Variables causing heteroscedasticity which can cause an overfit of our models are rejected from the study. This analysis was made by observing all the correlations between each variable in the form of a matrix.
    - release_date: It is related to the release_month_date
    - energy: correlated with loudness

FIGURE 38: CORRELATION 4



Source: Own elaboration

FIGURE 39: CORRELATION 5



Source: Own elaboration

- Modification and elimination of rows
  - Removing one of the two "danceability" columns generated when we have extracted data from Spotify.
  - Removing duplicate rows from different sources with the same data but different designations.
    - 'Artist and Title'

    - 'SONG NAME'
    - 'ARTIST NAME'
- Abnormal values were found in the "Tempo" and "Loudness" columns. A mask was created to keep only the values considered as normal as well as the outliers necessary to calculate our output value.
    - Loudness - between (-20 and 0)
    - Tempo - between (60 and 220)
- Changed column designation to improve the clarity of our tables as well as to eliminate characters that are supported by our data manipulation.
    - 'name' -> 'song name',
    - '(x?)' -> 'X?'

After cleaning and filtering our data, our training sample consists of 4273 songs.

Feature Selection: Feature engineering

We asked two music experts what qualities they believe influence a song to be a hit, what characteristics they take into account when composing a song, and what social characteristics they think are interesting to add to our study. These were their answers:

Argentinian music teacher:

"Es algo que se dice. Creo que es al 99% de efectividad, pero sí que hay una fórmula del hit que tiene que ver con ciertas progresiones de acordes que en general son amables para el oído humano. Por ejemplo, hay ciertas melodías con los mismos tonos que en general son progresiones en escala de DO que están casi en el 90% de las canciones hit que escuchamos. Después tiene mucho que ver con el género. Pero más allá de la melodía, se que hay dos cosas que se tienen en cuenta para esta cuestión del del hit y una son los BPM (beats per minute) que son la rítmica en la que está compuesta la canción. Es lo que te hace "sacudir la cabeza". Esto ha ido variando dependiendo de la época. No siempre la misma rítmica ha tenido el mismo impacto. Lo que yo vi en la facultad era que han ido subiendo el speed de las canciones porque generaban cada vez más impacto en esto de "mover la cabeza" que es lo que tienen, por ejemplo, las canciones de los Beatles. Estas canciones están compuestas generalmente en BPM que te lleva a seguir el ritmo aunque sea con la cabeza. Por lo general están compuestas en cuatro cuartos porque son las más fáciles de entender para el oído amateur (el que no estudia música) y después hay otros elementos pero tiene que ver con esto: con melodías amables, fáciles de digerir y letras fáciles de digerir."

American composer:

"When I compose a song, it needs to be a song that makes you want to dance. I take into consideration the tempo (BPM) and then I look for the chords and how many different chords the song has and see if it is in a Major or minor key. The language of the song and the words I use are also important. How direct is the message you want to transmit. So you can identify words that are commonly used (trend words or

expressions). You want to be new but familiar to the listener. Genre of the song is also important. The duration of the song has been changing over time. In the 90s it used to be 3 minutes 30 seconds, that was the hot spot, but now is getting even less because everyone's attention is shrinking. So, I compose 2 minutes 15 seconds, 2 minutes 45 seconds songs and I try not to pass 3 minutes 10 seconds. Usually I use the chorus 2 to 3 times in a song. Season definitely has a factor. October, November, December is when you do slow songs but if you have a dance song, I personally will release that in December or January just because you need time for it to get into a Spotify playlist and people listen to it. About how long the song has been in the top 10 the thing is that people who are really into music know that all these numbers are skewed. You have these big labels that have millions and millions of dollars so they can pay a DJ to play it for 3 months and that increases the numbers. In the free accounts in spotify there are songs that appear in the first one and it does not matter what you choose that those songs are going to be in the first position. Instruments are important and also the base of the song. Does not matter anymore in which studio the song was recorded or released. The number of songs in the album used to be 15 or 20. But in the past 10 years they have been putting as much music as possible in the market. Because before, there were people dictating what people would like the most but now since the internet and streaming the listeners dictates what they like. there are no more taste makers. The artist's age is a big factor. 16 to 23 is the sweet spot but there are outliers."

OUTPUT

After carrying out the univariate and bivariate analyses and taking into account the experts' insights, we selected the following variables to run our models.

- Length
- Danceability
- Accousticness
- Loudness
- Liveness
- Instrumentalness
- Speechiness
- Tempo
- Time_signature
- Mode
- Valence
- Key
- Released month
- Instagram followers

Modeling

Selection and application of the most appropriate modeling techniques

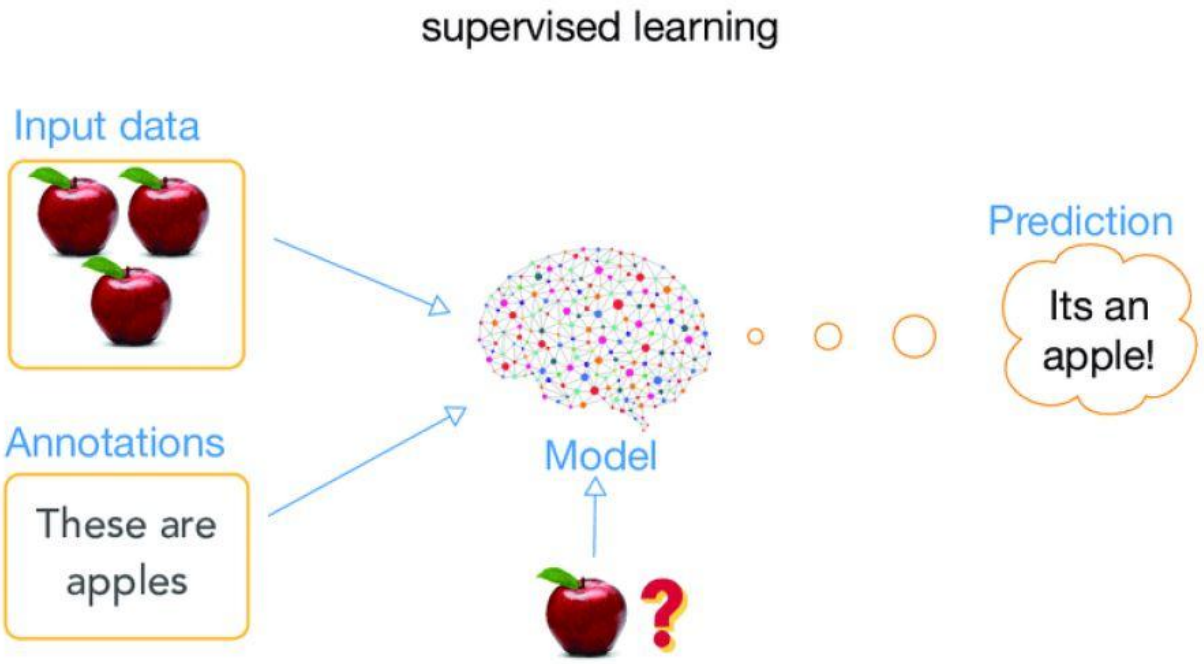
As part of our project, we are faced with a case of classification supervised learning, because we have labeled data that we can base our training models on.

We will assign the label "Top X" for songs in our datasets, which will define which characteristics could make a song be a hit.

We are going to be testing 4 different approaches to set our labels to the data. We will label a song as being a hit, using a Top X from our ranking, considering X as 100, 1000, 2000, 2500.

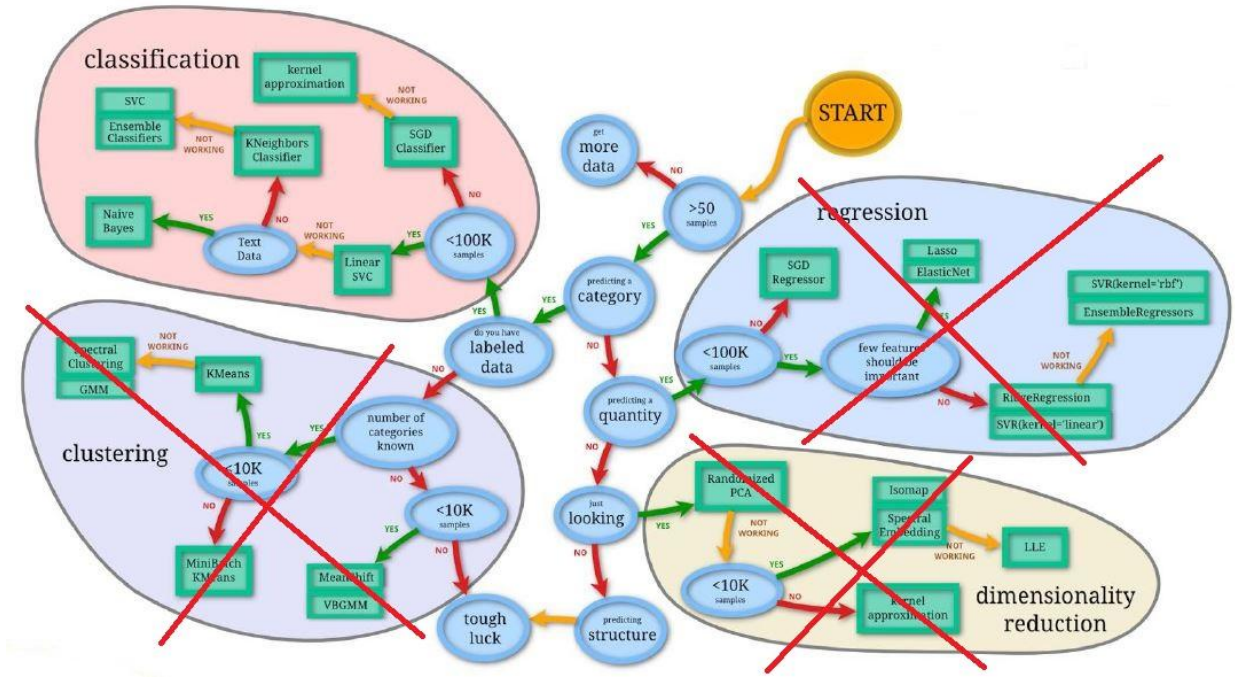
An alternative solution was to use a regression model to generate the total streams but that would have been less relevant according to our objectives.

FIGURE 40: SUPERVISED LEARNING



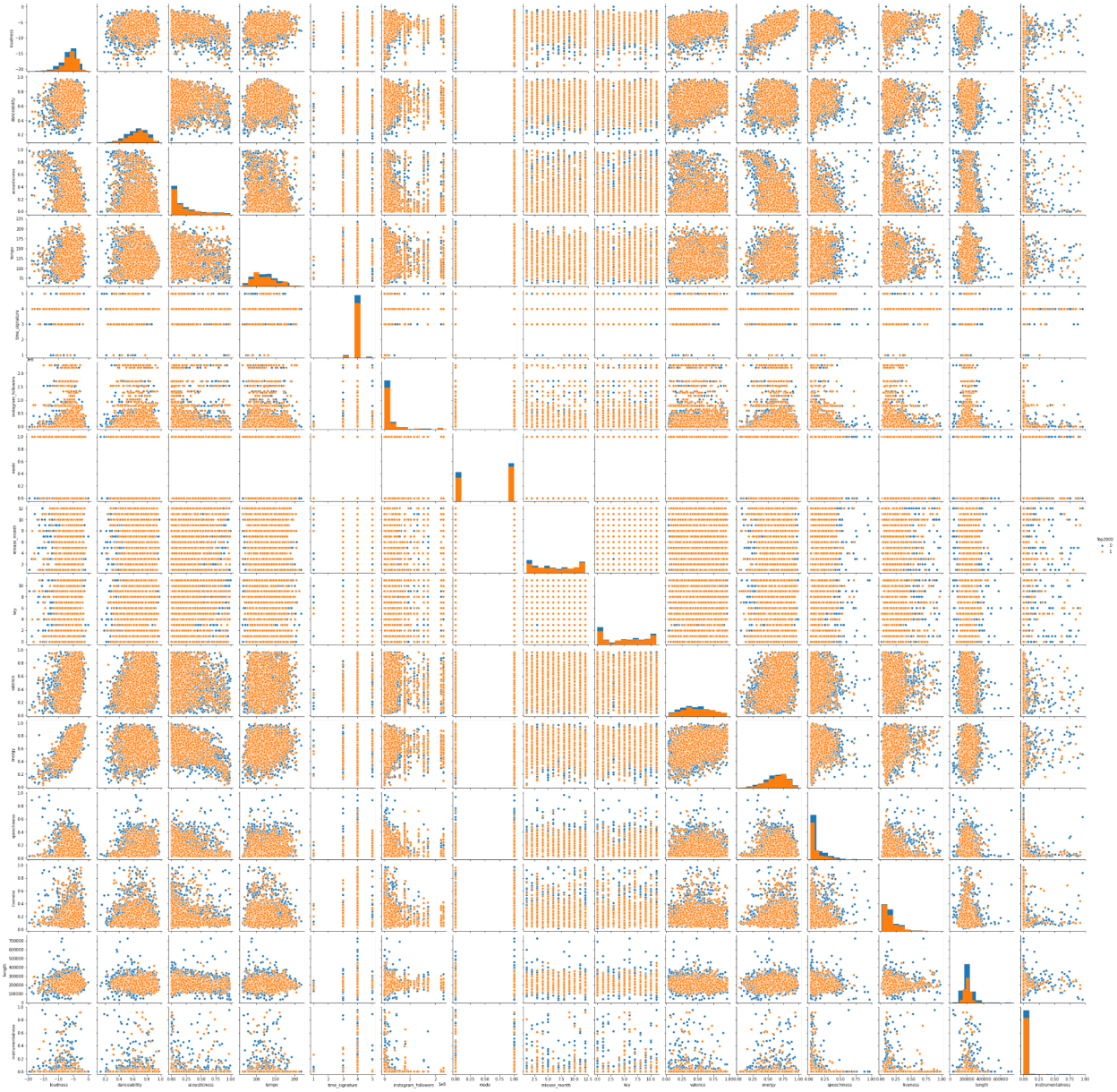
There are a multitude of algorithms that can be used for our classification task. But it is necessary to understand which ones will be most relevant to our case.

FIGURE 40: ML ALGORITHMS



Data representation:

FIGURE 41: DATA REPRESENTATION



In order to choose our model, we must already reduce the spectrum of possibilities to the classification algorithms. The philosophy is to gradually use the simplest to the most complex models according to the characteristics of our datasets.

To begin with, we counted the amount of data we have. Our largest dataset only has **4475** rows; it is not enough and it is advisable in this case to test the SVC model. But when put to the test, it brought bad results at model evaluation.

Furthermore, because we don't need to do text classification with our input data; it is better to use the **KNeighborsClassifier (KNN)** model instead of the Naive Bayes model . KNN made sense too with the layout of our data because it has outliers for each parameter we want to use. Also because they are grouped in different clusters, which is perfect for the KNN model. Finally, it brought very encouraging evaluation results during our first test, compared to other models, deciding us to keep it in our program.

We also decided to use a **Decision Tree** model because it is popular and applicable to a classification problem. The first advantage of this algorithm is that easy to understand. In addition, it is easy to interpret the results and be presented to non-technical people. Finally, the last advantage is that the Decision Tree is a simple algorithm that is not costly in terms of computation time. The main drawback is the risk of over-learning or overfitting. This is when the algorithm learns training data so accurately that it fails to generalize a satisfactory result to new data. In a Decision Tree, it also becomes easy to understand what are the audio characteristics that we must pay special attention to for creating hit songs.

Finally, the last model we want to use is the **Random Forest**, it is a method where we combine our prediction results together and use them for classification voting. This model combines multiple random decision trees outputs and generates an outcome based on the majority voted results for our problem (Hit or Non Hit). Thus this model is a direct improvement of the Decision Tree which corrects the lack of robustness and the overfitting risk of the latter.

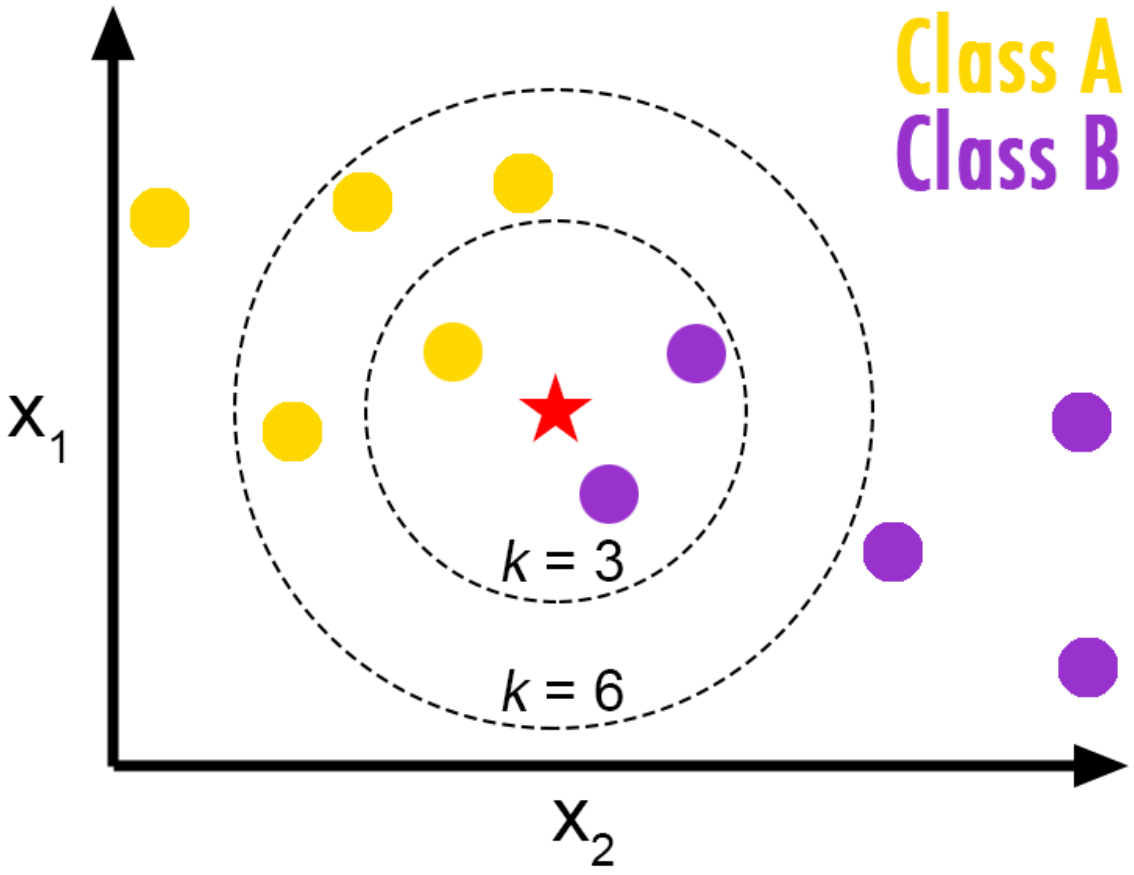
K-Nearest Neighbors (KNN) Model

Model introduction:

The K-Nearest Neighbors model is a popular algorithm that provides functionality for unsupervised and supervised neighbors-based learning methods.

To make a prediction, the KNN algorithm will be based on the entire dataset; for an observation, which is not part of the dataset that we want to predict, the algorithm will look for the K instances of the dataset closest to our observation. Then for these K neighbors, the algorithm will be based on their output variables "y" to calculate the value of the variable "y" of the observation that we want to predict. In our classification problem, the KNN model is used for the classification, it is the mode of the "y" variables of the "K" nearest observations that will be used for the prediction

FIGURE 42: KNN MODEL



Hyperparameters details:

FIGURE 43: HYPERPARAMETERS

K-Nearest Neighbors hyperparameters
n_neighbors : int, default=5
Number of neighbors to use by default for kneighbors queries.
weights : {'uniform', 'distance'} or callable, default='uniform'
weight function used in prediction. Possible values:
'uniform' : uniform weights. All points in each neighborhood are weighted equally.
'distance' : weight points by the inverse of their distance.
[callable] : a user-defined function which accepts an array of distances.
algorithm : {'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'
Algorithm used to compute the nearest neighbors:
'ball_tree' will use BallTree
'kd_tree' will use KDTree
'brute' will use a brute-force search.
'auto' will attempt to decide the most appropriate algorithm.
leaf_size : int, default=30
Leaf size passed to BallTree or KDTree.
p : int, default=2
Power parameter for the Minkowski metric.
metrics : str or callable, default='minkowski'
the distance metric to use for the tree.
metric_params : dict, default=None
Additional keyword arguments for the metric function.
n_jobs : int, default=None
The number of parallel jobs to run for neighbors search.

Bibliographic:

https://scikit-learn.org/stable/modules/neighbors.html

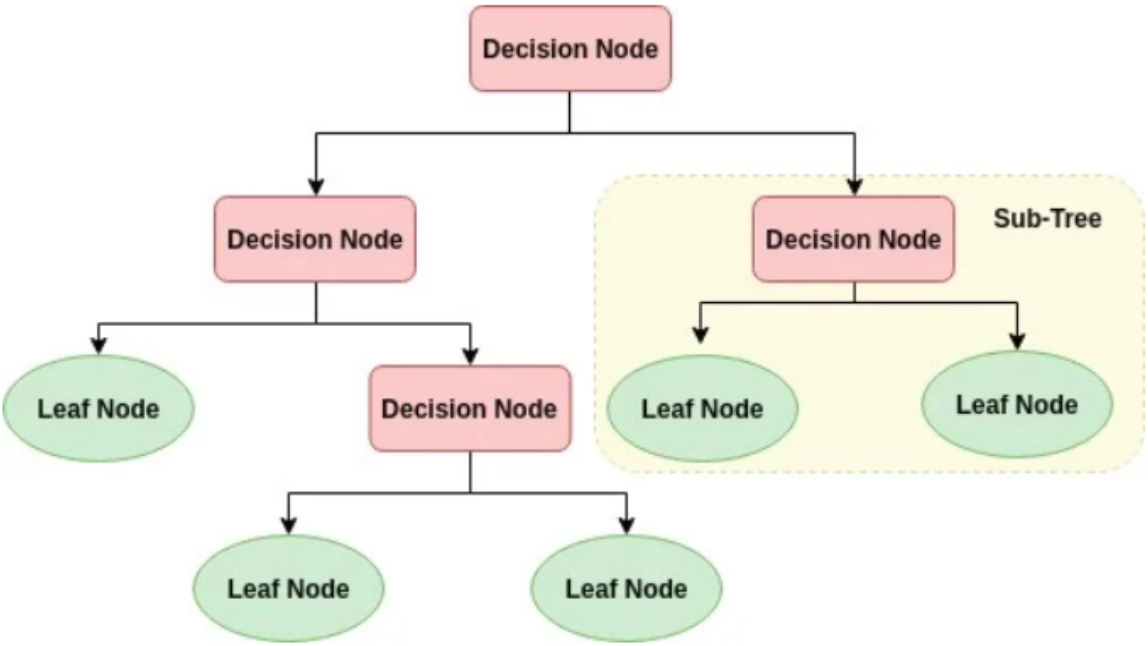
Decision Tree Model

Model introduction:

Decision trees are supervised learning algorithms, used in both regression and classification.

Each rectangle in the tree is called a node. The more nodes there are, the more precise the decision tree will be but at the risk of having an overfitted model. The last nodes of the decision tree are called the "leaves" of the tree. Decision trees are intuitive and easy to build, but they are a bit lacking when it comes to precision or accuracy in addition to being not very robust.

FIGURE 44: DECISION TREE



Hyperparameters details:

max_features*int, float or {"auto", "sqrt", "log2"}, default=None*
The number of features to consider when looking for the best split:
If int, then consider max_features features at each split.
*If float, then max_features is a fraction and int(max_features * n_features) features are considered at each split.*
If "auto", then max_features=sqrt(n_features).
If "sqrt", then max_features=sqrt(n_features).
If "log2", then max_features=log2(n_features).
If None, then max_features=n_features.
random_state*int, RandomState instance or None, default=None*
Controls the randomness of the estimator.
The features are always randomly permuted at each split, even if splitter is set to "best".
max_leaf_nodes*int, default=None*
Grow a tree with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.
min_impurity_decrease*float, default=0.0*
A node will be split if this split induces a decrease of the impurity greater than or equal to this value.
min_impurity_split*float, default=0*
Threshold for early stopping in tree growth. A node will split if its impurity is above the threshold, otherwise it is a leaf.
class_weight*dict, list of dict or "balanced", default=None*
Weights associated with classes in the form {class_label: weight}. If None, all classes are supposed to have weight one. For multi-output problems, a list of dicts can be provided in the same order as the columns of y.
cp_alpha*non-negative float, default=0.0*
Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than ccp_alpha will be chosen.

Bibliographic:

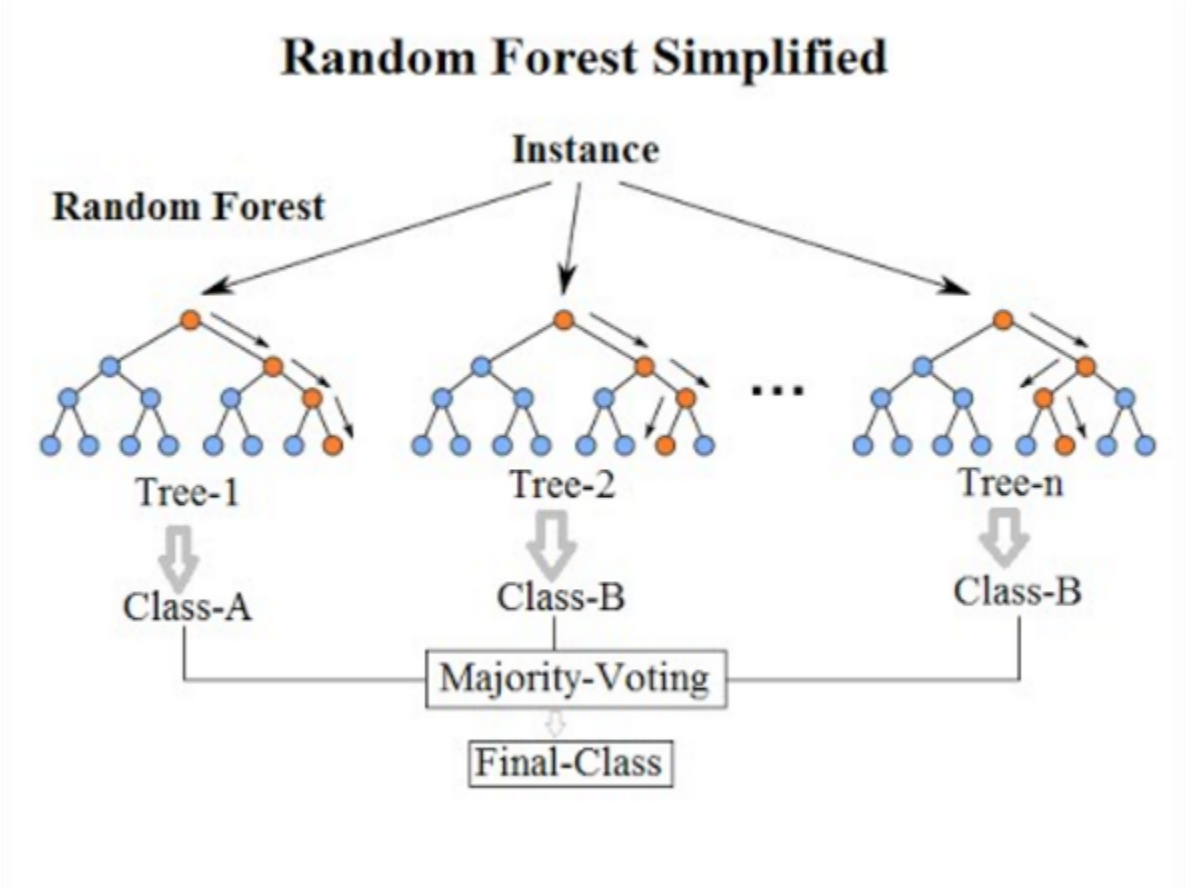
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Random Forest Model

Model introduction:

Random Forest algorithm is a set learning technique that relies on decision trees. The random forest model involves the creation of multiple decision trees using split data sets from the original data. And by randomly selecting a subset of variables at each step of the decision tree. The model then selects the mode of all the predictions of each decision tree.

The advantage over the Decision Tree algorithm is based on a majority prevalence model (where the majority wins), it reduces the risk of error of an individual tree and is therefore much more robust.



Hyperparameters details:

Random Forest hyperparameters
Decision Tree Hyperparameters
n_estimators*int, default=100*
The number of trees in the forest.

Bibliographic:

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Hyperparameter adjustment

Data representation :

Hyperparameters selection :

As seen previously, there is a multitude of hyperparameters for each algorithm. To choose the right ones for each model and configure them according to our data we will use the GridSearch algorithm to accurately select the optimal hyperparameters, but we can already estimate the order of magnitude of the latter.

K-Nearest Neighbors:

'n_neighbors': Between 5 and 500

Following our strategy of testing different configurations of the proportion of outputs and the desire to integrate new data in the future. It is not possible to provide a fixed estimate of the number of neighbors required for k-neighbors queries. So we will use the Grid-Search algorithm on a wide spectrum of solutions in order to retrieve the optimal configuration for each configuration.

'weights': 'distance'

We have two choices of weighting methods which are uniform and distance. Uniform does not take into account new data points and its k nearest neighbors. While the distance weighting solution assigns more points to the closer training points. So far the distance configuration has brought better results but it is still important to test uniform again once our data has evolved.

'algorithm': 'auto'

We have decided to leave the KNN model on its proposed default algorithm model to compute the nearest neighbor, which is auto. In order to improve our model, new data will be submitted to our model changing their configuration. Letting our model look for the best calculation solution seems at this stage of our project to be the best solution.

Decision Tree:

'criterion': 'entropy'

The choice between gini and entropy was decided based on the still small number of our data observations. Entropy will allow us to collect as much information as possible but we will be more penalized in the face of fewer small impurities. When we get more data improving our metrics we can then consider switching to gini.

'max_depth':Between 3 and 6

The maximum depth allows us to improve our accuracy. But using too large of a value may overfit our model and make it unusable when it is applied to another data set. This is why with our limited number of parameters it was decided to limit the depth to 6 layers.

‘max_leaf_nodes': Between 10 and 50

As for the max_depth the max_leaf _nod will be chosen sparingly. The best nodes configuration is defined with a balance between a relative reduction in impurity and an improvement of the accuracy. Depending on our limited number of parameters, we are therefore going to frame the maximum number of sheets between 10 and 50.

'min_samples_split': Between 1 and 5

The min_samples_split specifies the minimum number of samples required to split an internal node. In this case, with our short amount of data observations, we must not set it too high. This is why an estimate between 1 and 5 seems ideal in our case.

'min_samples_leaf': Between 1 and 5

The min_samples_leaf specifies the minimum number of samples required to be at a leaf node. Like min_samples_split, we don't have a lot *of data; and so we must not set it too high. This is why an estimate between 1 and 5 seems ideal in our case.*

Random Forest:

'n_estimators': 100

The number of trees in our model construction allows us to ensure the robustness of our model. The default value of 100 trees seems a good compromise between performance and robustness.

'max_leaf_nodes': Between 10 and 50

Same reason as the Decision Tree model

'min_samples_split': Between 1 and 5

Same reason as the Decision Tree model

'min_samples_leaf': Between 1 and 5

Same reason as the Decision Tree model

'criterion': 'entropy'

Same reason as the Decision Tree model

'max_depth':Between 3 and 6

Same reason as the Decision Tree model

Grid Search methodology :

We will use the GridSearch method, which consists of trying a multitude of values for each hyperparameter and then selecting which minimizes the error on the basis of the model test.

To do this, we create before creating for each model a dictionary comprising all possible hyperparameters values we want to test. The algorithm will then select the optimal combination and allow us to generate the ideal model.

In addition, the Grid Search method optimizes the hyperparameters while doing a cross-validation. Thus, all the data are used to optimize a parameter and the cross-validation checks the robustness of the learning on numerous divisions.

```
##TOP1000##
#KNN MODEL#
#Creation of the dictionnary in preparation of GridSearch
param_knn = {'n_neighbors': [5, 10, 50, 100, 175, 200, 215, 225, 235, 250], 'weights': ['uniform', 'distance'], 'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']}
#fit Gridsearch on data
model_knn1000 = GridSearchCV(KNeighborsClassifier(), param_knn, refit=True, verbose=2)
model_knn1000.fit(X_train_1000,y_train_1000)
#DECISION TREE MODEL#
#Creation of the dictionnary in preparation of GridSearch
param_dct = {'max_leaf_nodes': [2, 5, 7, 9, 10, 15, 20], 'min_samples_split': [2, 3, 4, 5], 'criterion':['gini','entropy'], 'max_depth':[3,4,5,6,7,8,10]}
#fit Gridsearch on data
model_dct1000 = GridSearchCV(DecisionTreeClassifier(), param_dct, refit=True, verbose=2)
model_dct1000.fit(X_train_1000,y_train_1000)
#RANDOM FOREST MODEL#
#Creation of the dictionnary in preparation of GridSearch
param_rdf = {'n_estimators': [50], 'max_leaf_nodes': [2, 5, 7, 9, 10, 15, 20], 'min_samples_split': [2, 3, 4, 5], 'criterion':['gini','entropy'], 'max_depth':[3,4,5,6,7,8,10]}
#fit Gridsearch on data
model_rdf1000 = GridSearchCV(RandomForestClassifier(), param_rdf, refit=True, verbose=2)
model_rdf1000.fit(X_train_1000,y_train_1000)
```

It is possible to find below the hyperparameters configurations retained for each of our models according to the two most interesting Top label configurations. The Top750 and the Top1000. It is then possible to observe a constant shared also between all the other configurations which therefore allows us to precisely define the optimal setting for our current data set.

GridSearch results for the TOP750 configuration:

```
KNN Model
KNeighborsClassifier(n_neighbors=50, weights='distance')

Decision Tree
DecisionTreeClassifier(criterion='entropy', max_depth=4, max_leaf_nodes=15)

Random Forest Model
RandomForestClassifier(max_depth=10, max_leaf_nodes=20, min_samples_split=5,
                       n_estimators=50)
```

GridSearch results for the TOP1000 configuration:

```
KNN Model
KNeighborsClassifier(n_neighbors=50, weights='distance')

Decision Tree
DecisionTreeClassifier(criterion='entropy', max_depth=4, max_leaf_nodes=15)

Random Forest Model
RandomForestClassifier(max_depth=10, max_leaf_nodes=20, min_samples_split=5,
                       n_estimators=50)
```

GridSearch results for the TOP2000 configuration:

```
KNN Model
KNeighborsClassifier(n_neighbors=50, weights='distance')

Decision Tree
DecisionTreeClassifier(criterion='entropy', max_depth=6, max_leaf_nodes=30,
                       min_samples_split=4)

Random Forest Model
RandomForestClassifier(criterion='entropy', max_depth=6, max_leaf_nodes=25,
                       min_samples_leaf=3, min_samples_split=3)
```

GridSearch results for the TOP2500 configuration:

```
KNN Model
KNeighborsClassifier(n_neighbors=100, weights='distance')

Decision Tree
DecisionTreeClassifier(criterion='entropy', max_depth=3, max_leaf_nodes=5,
                       min_samples_split=3)

Random Forest Model
RandomForestClassifier(criterion='entropy', max_depth=6, max_leaf_nodes=30,
                       min_samples_leaf=2, min_samples_split=5)
```

Retro Data Engineering strategy

In our situation, it is very difficult to know what a Top Song is. Even if we have the musical characteristics and the total score of the most streamed and popular music over a long period of time; it is not possible to classify this target with certainty. In addition, a classification of Top songs that is too demanding, risk of rendering our models unusable for the next datasets.

Once our models have been developed, the strategy employed will therefore be to test them on different scenarios. These are generated by gradually increasing the requirement of our classification over the total of our total dataset. For example, we will use to build our models a dataset where the target only includes the 100 most popular songs. Then the 500 first, the 750, 1000, 1500 and finally the first 2000.

```
TOP 100:
0    4375
1     100
Name: Top100, dtype: int64
TOP 500:
0    3992
1     483
Name: Top500, dtype: int64
TOP 750:
0    3520
1     955
Name: Top1000, dtype: int64
TOP 1000:
0    3520
1     955
Name: Top1000, dtype: int64
TOP 1500:
0    3047
1    1428
Name: Top1500, dtype: int64
TOP 2000:
```

Command took 0.05 seconds -- by rombale

```
X_train_100 = X_train_100[predictors].values
X_test_100 = X_test_100[predictors].values
X_train_500 = X_train_500[predictors].values
X_test_500 = X_test_500[predictors].values
X_train_750 = X_train_750[predictors].values
X_test_750 = X_test_750[predictors].values
X_train_1000 = X_train_1000[predictors].values
X_test_1000 = X_test_1000[predictors].values
X_train_1500 = X_train_1500[predictors].values
X_test_1500 = X_test_1500[predictors].values
X_train_2000 = X_train_2000[predictors].values
X_test_2000 = X_test_2000[predictors].values
```

Command took 0.04 seconds -- by rombaleynaud@student.eae.

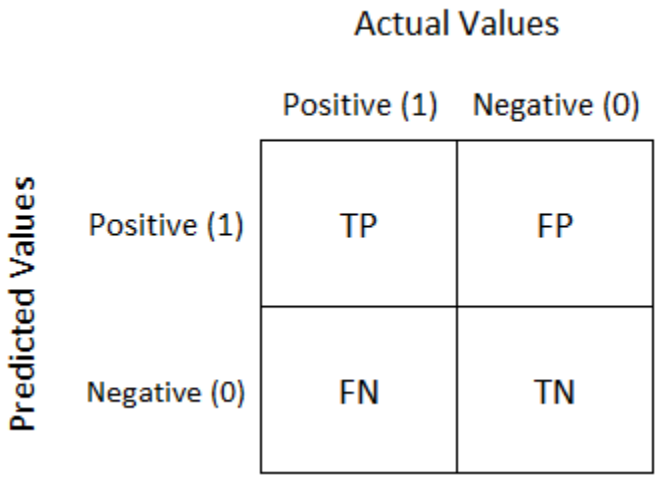
Thus it will be possible for us to find the ideal balance between the requirement of the Top music and the robustness and precision of our models.

Evaluation

Results evaluation

In order to obtain a vision of the performance of our classification models, we decided to use a confusion matrix. In order to illustrate the performance of the classifier based on the values True Positive, True Negative, False Positive, False Negative. It is then possible to generate the Precision, Recall, and F1-Score metrics.

In our study, the positive class corresponds to a song classification being a hit.



Accuracy - Accuracy is a ratio of correctly predicted observations to the total observations. Accuracy is a great indicator but only when you have symmetric datasets where values of false positives and false negatives are almost the same. Therefore, it's important to look at other parameters to evaluate the performance of your model.

Accuracy = TP+TN/TP+FP+FN+TN

Precision - Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The question that this metric answer is of all songs labeled as Top song, how many actually are really in the top class?

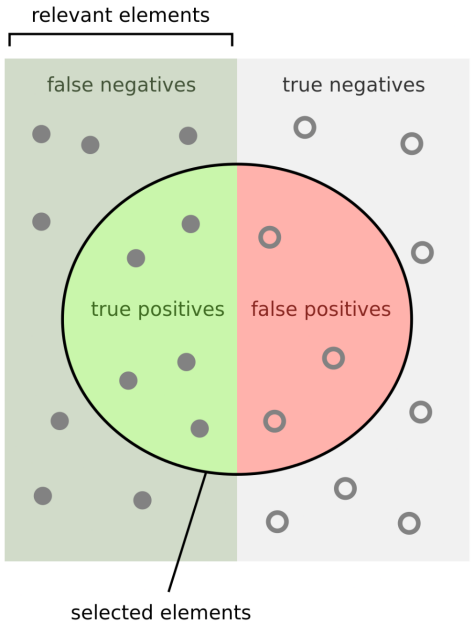
Precision = TP/TP+FP

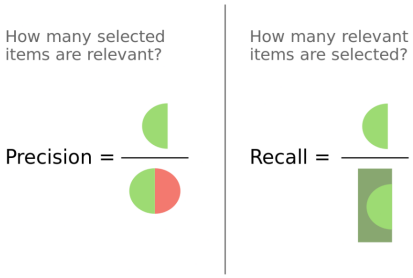
Recall (Sensitivity) - Recall is the ratio of correctly predicted positive observations of all observations in the classification, meaning True positives and False Positives. The question recall answers is: Of all the songs that are truly in the top class, how many did we label?

Recall = TP/TP+FN

F1 score - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account.

In addition, we use the cross-validation procedure which allows resampling to evaluate our models on a limited data sample. It suffices to set the parameter "k" which represents the number of groups into which the sample is to be divided.





Confusion matrix and metrics: TOP2000

```
KNN Model
KNeighborsClassifier(n_neighbors=50, weights='distance')
[[392 180]
 [218 278]]
Precision: 0.607
Recall: 0.560
F1 Score: 0.583
              precision    recall  f1-score   support

           0       0.64      0.69      0.66       572
           1       0.61      0.56      0.58       496

    accuracy                           0.63      1068
   macro avg       0.62      0.62      0.62      1068
weighted avg       0.63      0.63      0.63      1068


Decision Tree
DecisionTreeClassifier(criterion='entropy', max_depth=6, max_leaf_nodes=30,
                       min_samples_split=4)
[[258 314]
 [155 341]]
Precision: 0.521
Recall: 0.688
F1 Score: 0.593
              precision    recall  f1-score   support

           0       0.62      0.45      0.52       572
           1       0.52      0.69      0.59       496

    accuracy                           0.56      1068
   macro avg       0.57      0.57      0.56      1068
weighted avg       0.58      0.56      0.56      1068


Random Forest Model
RandomForestClassifier(criterion='entropy', max_depth=6, max_leaf_nodes=25,
                       min_samples_leaf=3, min_samples_split=3)
[[453 119]
 [293 203]]
Precision: 0.630
Recall: 0.409
F1 Score: 0.496
              precision    recall  f1-score   support

           0       0.61      0.79      0.69       572
           1       0.63      0.41      0.50       496

    accuracy                           0.61      1068
   macro avg       0.62      0.60      0.59      1068
weighted avg       0.62      0.61      0.60      1068
```

Confusion matrix and metrics: TOP2500

```
KNN Model
KNeighborsClassifier(n_neighbors=100, weights='distance')
[[163 300]
 [119 486]]
Precision: 0.618
Recall: 0.803
F1 Score: 0.699
              precision    recall  f1-score   support

           0       0.58      0.35      0.44       463
           1       0.62      0.80      0.70       605

    accuracy                           0.61      1068
   macro avg       0.60      0.58      0.57      1068
weighted avg       0.60      0.61      0.59      1068
```

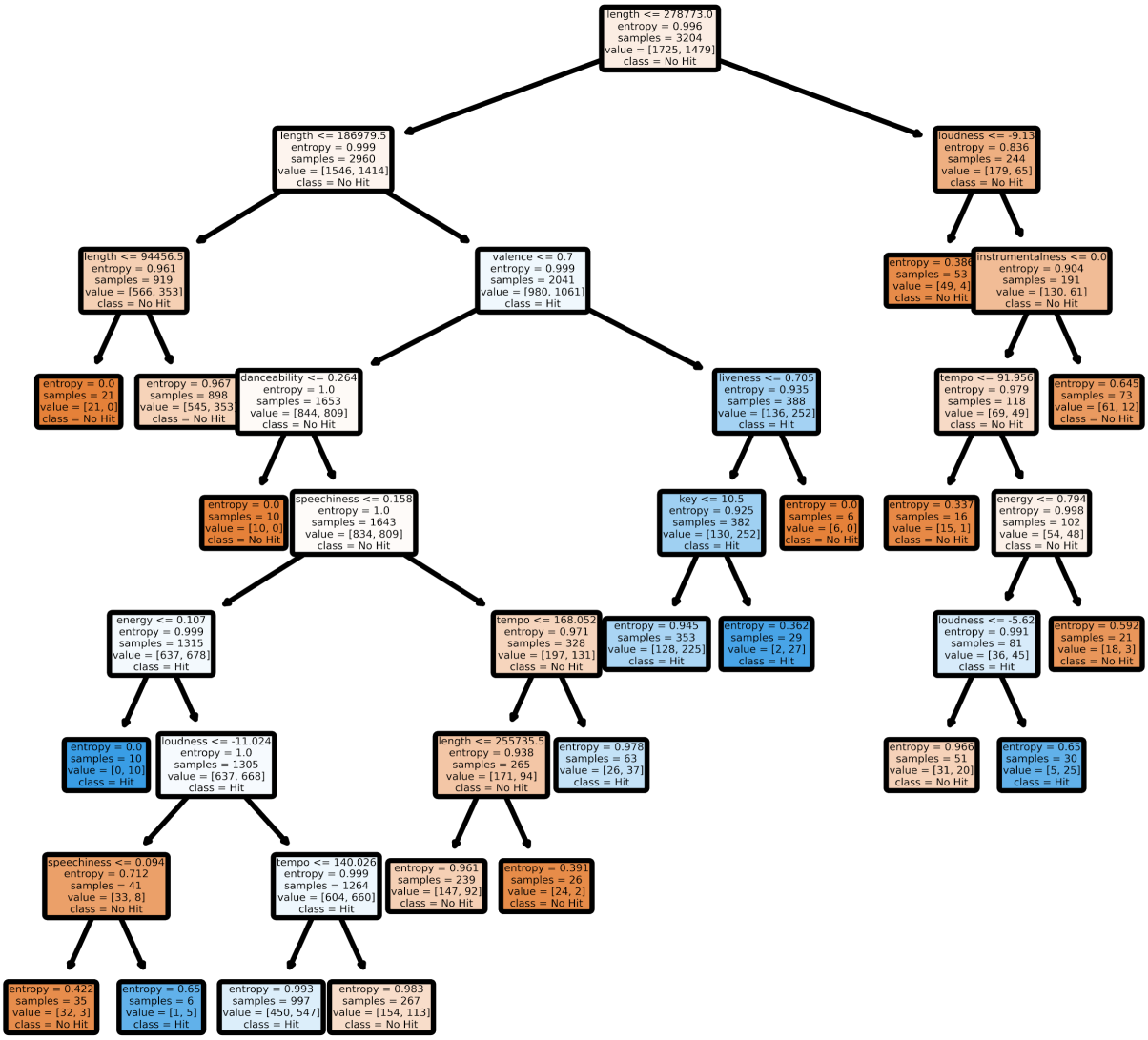
```
Decision Tree
DecisionTreeClassifier(criterion='entropy', max_depth=3, max_leaf_nodes=5,
                       min_samples_split=3)
[[ 38 425]
 [ 15 590]]
Precision: 0.581
Recall: 0.975
F1 Score: 0.728
              precision    recall  f1-score   support

           0       0.72      0.08      0.15       463
           1       0.58      0.98      0.73       605

    accuracy                           0.59      1068
   macro avg       0.65      0.53      0.44      1068
weighted avg       0.64      0.59      0.48      1068
```

```
Random Forest Model
RandomForestClassifier(criterion='entropy', max_depth=6, max_leaf_nodes=30,
                       min_samples_leaf=2, min_samples_split=5)
[[ 87 376]
 [ 25 580]]
Precision: 0.607
Recall: 0.959
F1 Score: 0.743
              precision    recall  f1-score   support

           0       0.78      0.19      0.30       463
           1       0.61      0.96      0.74       605

    accuracy                           0.62      1068
   macro avg       0.69      0.57      0.52      1068
weighted avg       0.68      0.62      0.55      1068
```

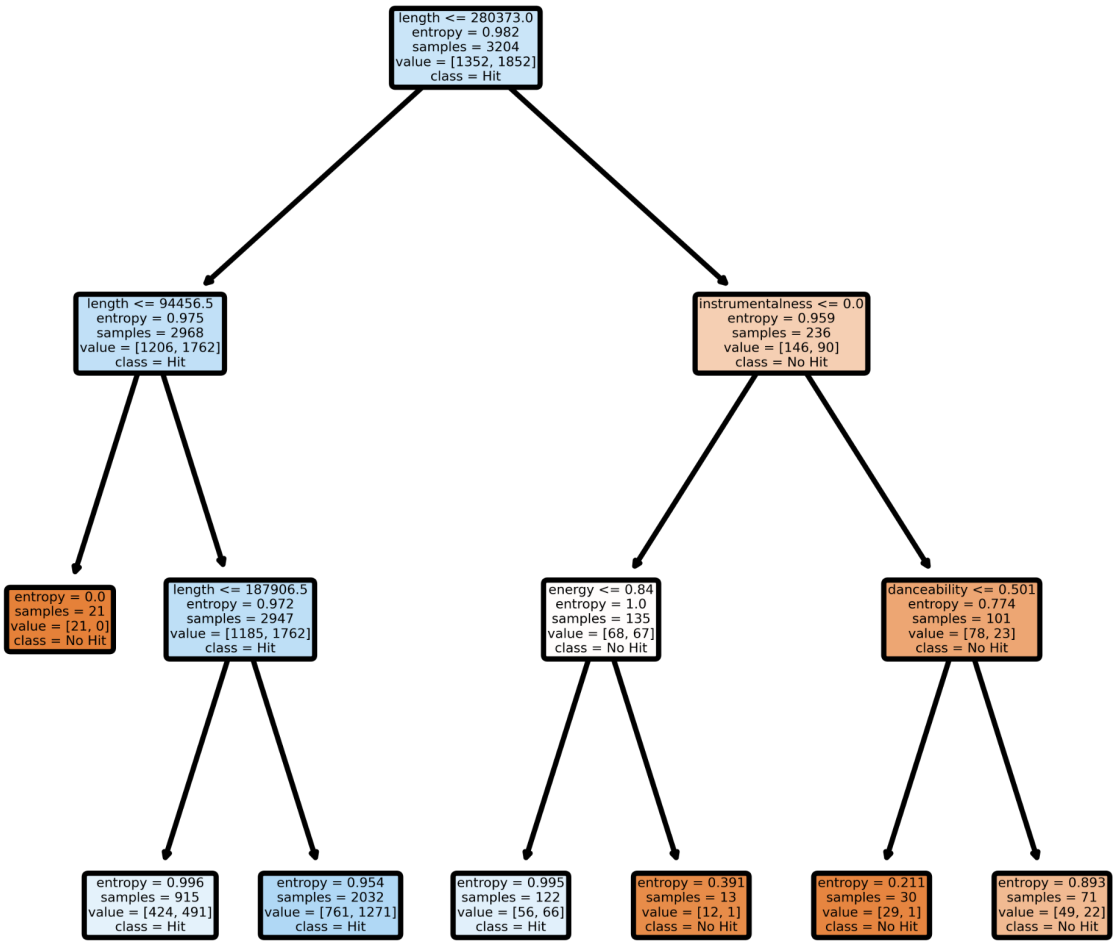
Metric results through the different Top outputs configuration :

Models		Top 100	Top 1000	Top 2000	Top 2500
KNN	Accuracy	0,98	0,77	0,64	0,62
	Precision	0,75	0,52	0,59	0,64
	Recall	0,13	0,178	0,54	0,78
	F1 Score	0,22	0,26	0,56	0,7
Decision Tree	Accuracy	0,98	0,76	0,56	0,6
	Precision	0	0	0,61	0,59
	Recall	0	0	0,15	0,98
	F1 Score	0	0	0,24	0,74
Random Forest	Accuracy	0,98	0,76	0,57	0,62
	Precision	0	0,33	0,58	0,61
	Recall	0	0,04	0,34	0,95
	F1 Score	0	0,08	0,43	0,74

Decision Tree plot: Top 2000



Decision Tree plot: Top 2500



Conclusion on the strategy of using different Top outputs configuration:

Following our strategy of using different Top configurations to see the ideal balance between the restriction of the Top hits and the performance of our models. We can see that today we are only starting to have a reliable model from the TOP 2500, which represents 73% of our training set. This result is not yet satisfactory but we can see that we are on the right track and that with a larger amount of data and new parameters we will undoubtedly be able to make our model more precise on the Top Songs restriction.

Validation against business objectives

Interpretation and contextualization

- Understand the cost associated with errors: The best model is the one that makes errors with the lowest associated cost and not the one that makes the fewest errors

The metric accuracy works at its best when the false positives and false negatives have similar costs. Otherwise, if the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall metrics depending on where the scale tilts.

In the case of our project, a False Positive will induce a marketing investment to make the potential Hit more visible, thus inducing a significant financial cost and a loss if the music does not turn out to be a Hit.

On the other hand, the False Negative does not induce a cost but an opportunity miss and an economic problem. This is why we have decided to prioritize the False Positives which will be more penalizing for the users of our algorithm and to prioritize the model with the greatest precision; which here is actually the Random Forest model.

In a real scenario where we are helping a specific record label, consultant, or streaming platform we would ask the customer to provide a cost weight between these two cost approaches and fine-tune our evaluation techniques.

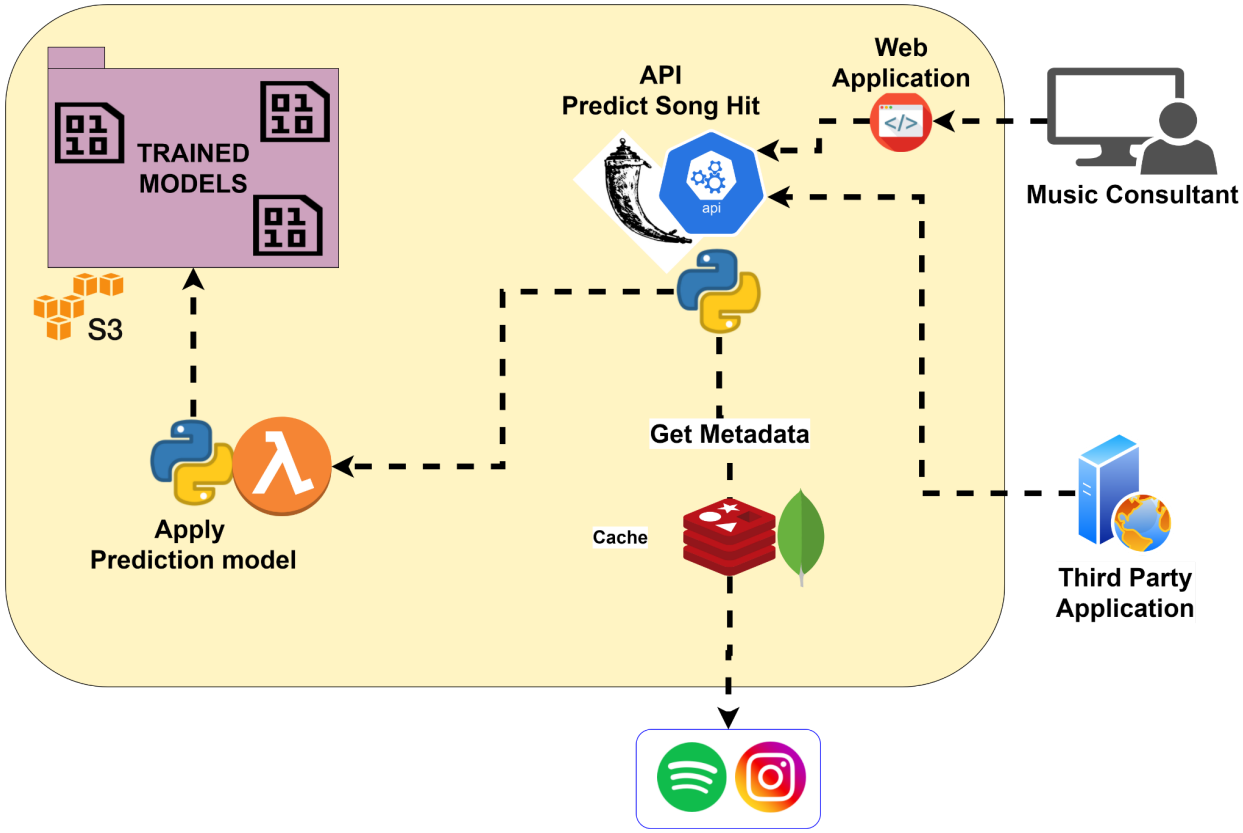
Deployment

Given the fact that we have a deep understanding of the business, we have drafted the outlines on how these models should be deployed for production use.

Ideally, a seasoned data engineer should take on:

1. Code review our artifacts for performance and industrialization
2. Review deployment strategy and recommend any ML Ops platform

FIGURE 52: Deployment Model



Source: Own elaboration

API Predict hit song(s)

The API will serve the prediction from a song being a hit or not, to:

- A third party application
- A simple web application where a music consultant can select Spotify's song or playlist to evaluate

We will use Flask as a web development framework to build the REST APIs in Python.

The API will query the metadata needed for prediction from Instagram and Spotify, and send the prediction request to the Prediction model service.

Metadata Cache

To boost the performance of the solution, and to prevent calls that would otherwise lock out our accounts from Instagram and Spotify.

We will cache metadata retrieved values for:

- Spotify audio features. Recommended cache store will be MongoDB. Audio features for a given song are pretty static. So, once we have queried Spotify for these values, we can safely assume that they will not change for a given song.

  Cache invalidation strategy will be based on a large expiration time, or be based on an active rule that could check optimal query performance vs the amount of documents stored at the cache collection.

- Instagram follower count. Recommended cache store is Redis. As incoming requests may contain several songs of the same artist, we can benefit from an in-memory store, especially when having a large volume of requests.

  Cache invalidation strategy will be based on a very short expiration time, as information is volatile.

Applying Prediction Model

This application layer will be deployed on a serverless service or FaaS (Function as a Service), such as AWS Lambda or Azure Functions.

It will receive the prediction request from the public-facing API and load the corresponding model from an object storage service, such as AWS S3.

Once deserialized the fit model, it will predict according to the input values and get the response back to the API.

Trained Models

These are binary objects from fit models that have been serialized through Pickle.

Model Evolution

Our dataset labels are based on song popularity, which at this moment we are capturing through weekly stream count charts from Spotify.

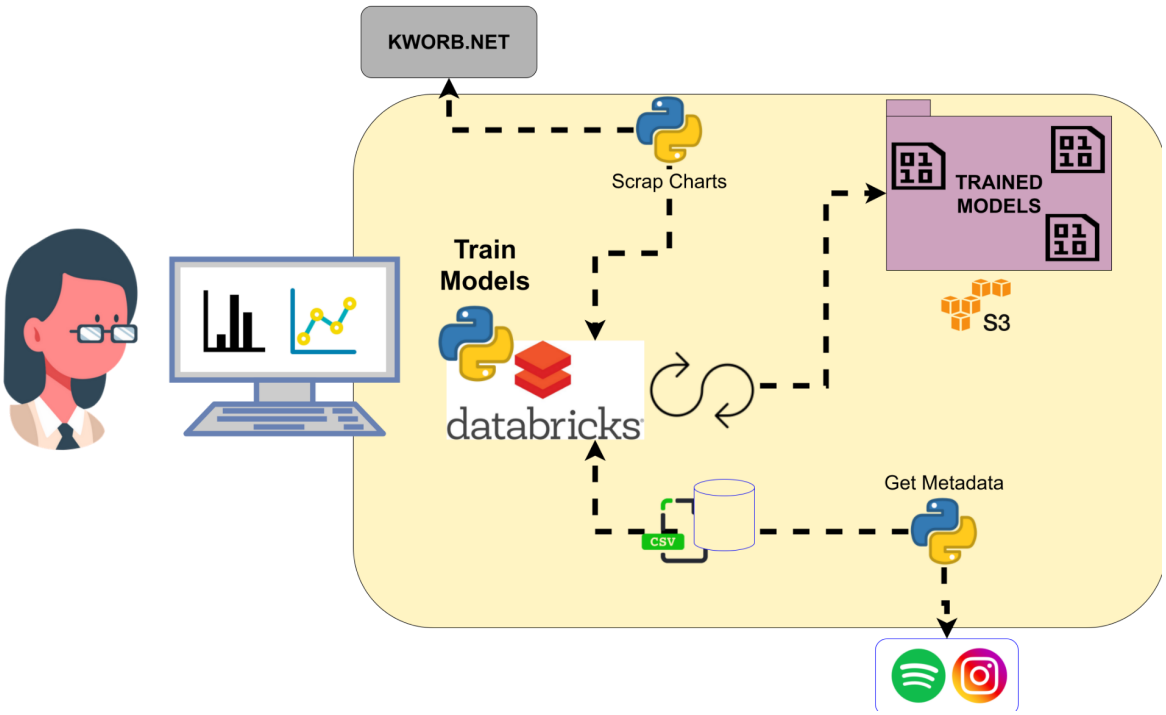
In our first review and iteration of the models, we will not automate the model evolution because we believe it is important to do a manual review of the datasets we're feeding for re-train.

At this first stage, the manual model review will be done between 5 to 15 weeks. This frequency has been determined because:

- 5 weeks is the Median from the weeks variable from our initial dataset. This means that more than 50% of our initial sample will be 5 weeks or less on a top chart.
- 15 weeks is the Average from the weeks variable from our initial dataset.

We acknowledge that the weeks variable does not mean consecutive weeks, which could've been a better source of information to determine model re-evaluation frequency, but it is the best-informed value we are willing to take to determine re-evaluation frequency.

FIGURE 53: Model Evolution



Source: Own elaboration

Scrap Charts

In our MVP (Minimum Viable Product) at the DSAA course deliverable, we have parsed KWORB charts, using excel and a text editor.

We will use python web-scrapping libraries to ease this task. That way we can upload the output directly to our databricks instance for later analysis.

Train Models

Model Training will be done in python using Apache Spark in Databricks.

In this first stage, Databricks notebooks will not be compiled into Jobs for automation. Manual intervention from the Data Scientist will be needed to act on any manual data cleaning if needed.

Trained Models

Once the models are fit and tested with multiple hyperparameter combinations. The Data scientist will determine which ones are eligible for production deployment. Fit models will be serialized using Pickle, and objects will be uploaded into AWS S3.

Get Metadata for Datasets

Python console programs that decorate songs and artists' datasets are running in the local machine of the Data Scientist.

As this is not recommended, we would need to industrialize the Python console programs, so that they can run in the cloud.

As these programs are not performance-intensive, we could deploy them on top of:

1. AWS Lambda, so that we're only charged on-demand.
2. AWS Elastic Container Services (ECS). Furthermore, since these console programs have stop/resume processing time, we could add AWS Spot Instances to the ECS cluster to minimize costs.

BIBLIOGRAPHY

Audioproduccion, 2020. https://www.audioproduccion.com/compania-discografica/

Billboard Charts, 2021. https://www.billboard.com/charts

BusinessInsider, 2020. https://www.businessinsider.com/how-much-does-spotify-pay-per-stream

DeveloperSpotify, 2021. https://developer.spotify.com/documentation/web-api/reference/#endpoint-get-audio-features

Developer.Spotify, 2021. https://developer.spotify.com/documentation/web-api/reference/#object-audiofeaturesobject

Hypeauditor, 2021. https://hypeauditor.com/

Music Brainz, 2021. https://musicbrainz.org/

Music Story, 2021. https://www.music-story.com/

Spotify for Developers, 2021. https://developer.spotify.com/

Spotipy, 2021. https://spotipy.readthedocs.io/en/2.17.1/?highlight=audio%20features#welcome-to-spotipy

Spotify Charts, 2021. https://spotifycharts.com/

Flask Web Development Framework, 2021. https://flask.palletsprojects.com/en/2.0.x/

Mongo DB, 2021. https://www.mongodb.com/1

Redis In-Memory Data Store, 2021. https://redis.io/

AWS Lambda, 2021. https://docs.aws.amazon.com/lambda/index.html

Azure Functions, 2021. https://docs.microsoft.com/en-us/azure/azure-functions/

AWS Simple Storage Service S3, 2021. https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

Pickle, object serialization for Python, 2021. https://docs.python.org/3/library/pickle.html

AWS Elastic Container Service, 2021. https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html

AWS Spot Instances, 2021. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-instances.html

Adding AWS Spot instances to an ECS cluster to minimize costs, 2017. https://aws.amazon.com/blogs/compute/powering-your-amazon-ecs-cluster-with-amazon-ec2-spot-instances/#:~:text=The%20ECS%20console%20uses%20Spot,the%20best%20prices%20for%20you.