



JUST REPEAT- HIT

Measuring nowadays
music **repetitiveness**
using textual compression



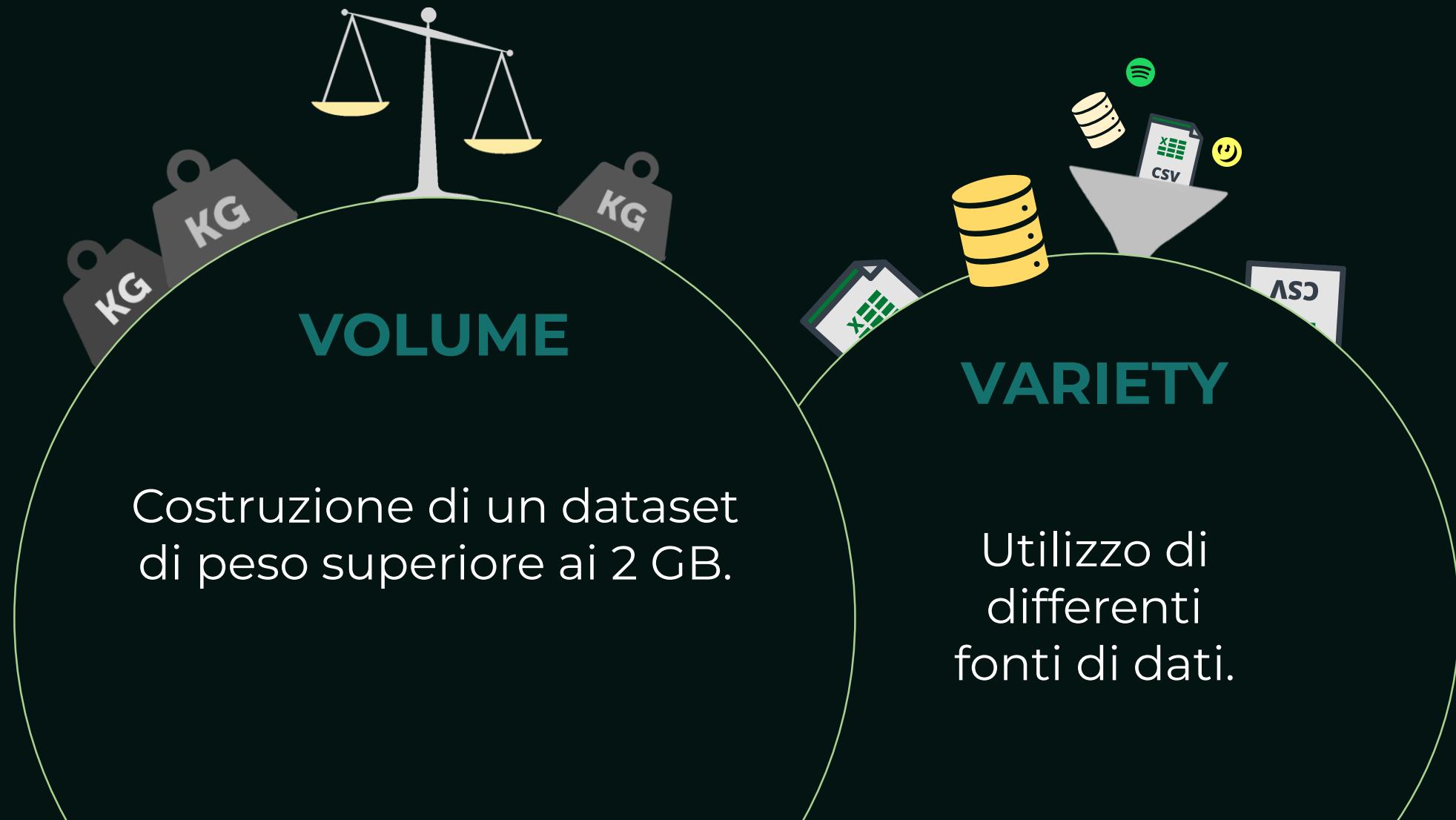
*«Stavamo molto meglio quando vi erano i vecchi col sigaro, invece che con questi giovani produttori esperti che **decidono cosa le persone devono vedere e ascoltare sul mercato**. I giovani sono più conservatori e più pericolosi per l'arte che i vecchi.»*

OBIETTIVI

Ci siamo posti **due** quesiti:

1. La musica sta effettivamente diventando *sempre più ripetitiva*?
2. In caso affermativo, *sono soprattutto i nuovi generi*, come ad esempio la Trap, *a trainare* questa tendenza?

OBIETTIVI DIDATTICI



APPROCCIO METODOLOGICO



Billie Eilish



CALCOLO INDICE
di RIPETITIVITÀ



CALCOLO INDICE DI RIPETITIVITÀ

$$\text{indice di ripetitività} = 1 - \frac{\text{parole uniche}}{\text{parole totali}}$$

Attraverso la libreria Collections, dando in input alla sua funzione Counter un testo, era possibile ricavare un dizionario che avesse come chiave una parola del testo e come valore il numero di volte che la stessa si ripeteva.

A partire da questo dizionario, è stato possibile ricavare il valore dell'**Indice di Ripetitività** dato dal complementare del rapporto tra il numero di parole uniche all'interno di un testo di una canzone e le parole totali del testo stesso.



CALCOLO INDICE DI RIPETITIVITÀ

Conteggio **parole totali**

```
def contaparoletot(text):
    words = re.findall(r'\w+',text)
    parole = Counter(words)
    return sum(parole.values())
```

Attraverso l'input alla sua funzione Counter un testo, era possibile ricavare un dizionario che avesse come chiave una parola del testo Conteggio **parole uniche** come valore il numero di volte che la

parole uniche
parole totali

A partire da questo valore dell'**Indice di ripetitività**, si calcola il rapporto tra le parole uniche e le parole totali di un testo di una canzone e le parole totali del testo stesso.

```
def contaparolemin(text):
    words = re.findall(r'\w+',text)
    parole = Counter(words)
    return len(parole)
```



CALCOLO INDICE DI RIPETITIVITÀ

Conteggio **parole totali**

```
def contaparoletot(text):
    words = re.findall(r'\w+',text)
    parole = Counter(words)

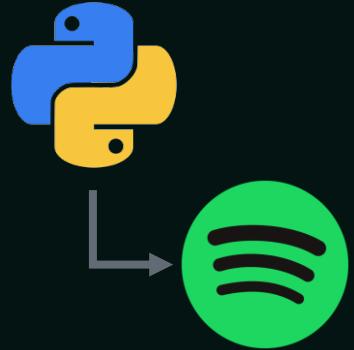
    d_final["indice"] = (1-(d_final["parolemin"]/d_final["paroletot"]))
```

$$\frac{\text{parole uniche}}{\text{parole totali}}$$

Calcolo effettivo dell'**Indice**

```
def contaparolemin(text):
    words = re.findall(r'\w+',text)
    parole = Counter(words)
    return len(parole)
```

A partire da
valore dell'**Indice**
del rapporto
un testo di una canzone e le parole totali del testo stesso.

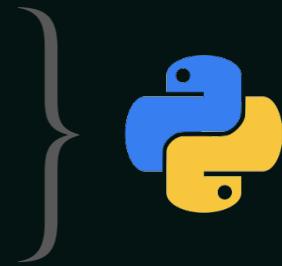


API SPOTIFY



PRIMA FASE DI RACCOLTA DATI (*SPOTIFY*)

La richiesta alla **API di Spotify** è stata effettuata iterando su tutto l'alfabeto, richiedendo tutti gli artisti che iniziassero con una determinata lettera



Il processo si è rivelato molto lungo (~ 8'000'000 di artisti ottenuti)

È stato necessario usufruire delle **Macchine Virtuali**



PRIMA FASE DI RACCOLTA DATI (SPOTIFY)

```
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
import json
import pandas as pd
import pymongo
from pymongo import MongoClient

cid ="f2edcce358934598a27da7bf4ba92082"
secret = "5c7c7158908a48f79ad2bba8faa6f2c2"

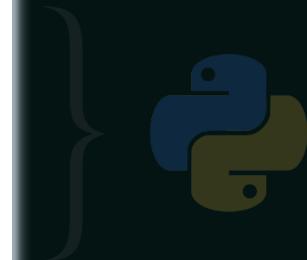
client_credentials_manager = SpotifyClientCredentials(client_id=cid, client_secret=secret)
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)

client=MongoClient('localhost',27017)

db=client.Progetto
singer=db.singer

alphabet = ["a*", "b*", "c*", "d*", "e*", "f*", "g*", "h*", "i*", "j*", "k*", "l*", "m*", "n*", "o*", "p*", "q*", "r*", "s*", "t*", "u*", "v*", "w*", "x*", "y*", "z*"]

conta = 0
import pprint
for x in alphabet:
    print("Lettera: " + x)
    artist_res = sp.search(q=x ,type = 'artist')
    for i in range(0, artist_res["artists"]["total"] // 50):
        artist_res = sp.search(q=x ,type = 'artist', limit = 50,offset=i)
        for i, t in enumerate(artist_res['artists']['items']):
            conta = conta + 1
            id = conta
            #print(id)
            name = t['name'].encode(encoding = 'UTF-8',errors = 'replace')
            #print(name)
            url = (t['external_urls']['spotify']).encode(encoding = 'UTF-8',errors = 'replace')
            #print(url)
            follower = t['followers']['total']
            #print(follower)
            genres = (t['genres'])
            #print(genres)
            popularity = t['popularity']
            #print(popularity)
            n_art = {'id': id, 'name': name,'follower': follower, 'genres':genres, 'popularity': popularity, 'url':url}
            singer.insert_one(n_art)
```



molto lungo
i ottenuti)

riuire delle



PRIMA FASE DI RACCOLTA DATI (SPOTIFY)

Risultati ottenuti:

- Nome
- Numero di follower
- Genere musicale
- Popolarità al momento della richiesta
- URL alla pagina Spotify dell'artista





STORAGE + MANIPULATION

STORAGE



L'idea iniziale era quella di agire direttamente su MongoDB, memorizzando sullo stesso grazie alla libreria Pymongo

Dopo un'attenta
esplorazione

The diagram shows a white rectangular box containing a green icon of a broom sweeping over a grid, with the word 'CSV' written below it. A grey arrow points from this box to the right, leading into a large rounded rectangle. Inside this rectangle, the text reads: "Ci siamo accorti che il Dataset necessitava di un'ingente pre-processing, quindi abbiamo dovuto estrarre un file CSV per poterlo lavorare su Python".

STORAGE

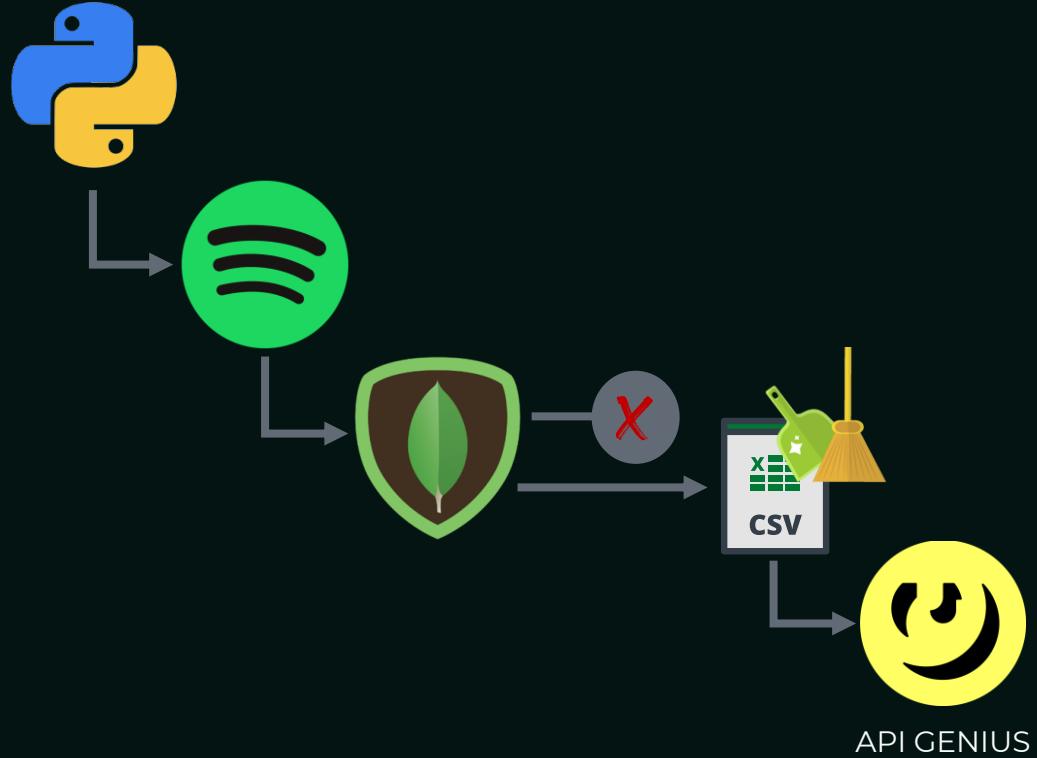


- Eliminazione duplicati
- Gestione NaN value
- Gestione caratteri speciali

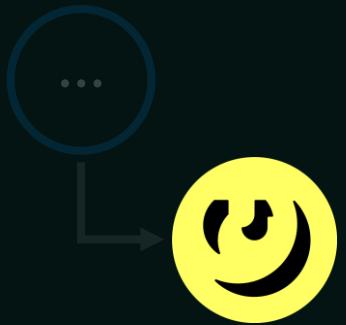
{}

Da 8.000.000 di artisti
si passa a circa 255.000

Questi *255.000* artisti risultano tutti **significativi** e pronti per essere iterati su Genius al fine di richiedere i testi di dieci loro canzoni



API GENIUS



SECONDA FASE DI RACCOLTA DATI (*GENIUS*)

Iteriamo gli Artisti precedentemente ottenuti, avvalendoci della API di Genius per ottenere **dieci** loro canzoni

- In questo caso abbiamo dovuto gestire alcune eccezioni:
1. Artista non trovato
 2. Meno di dieci canzoni per artista



artist
title
lyrics

L'API in parte risolveva questo problema attraverso la funzionalità "*changing name*" che trovava gli artisti anche in caso di differenze minime (*ex*: accenti o maiuscole) ma in seguito si è rivelata un problema

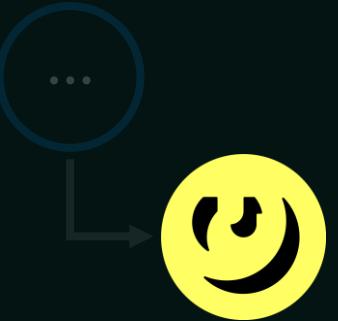
SECONDA FASE DI RACCOLTA DATI

Iteriamo gli Artisti
precedentemente otte-
nuti avvalendoci della AP
Genius per ottenere
le loro canzoni

artist

title

lyric



```
import pandas as pd
import lyricsgenius as genius

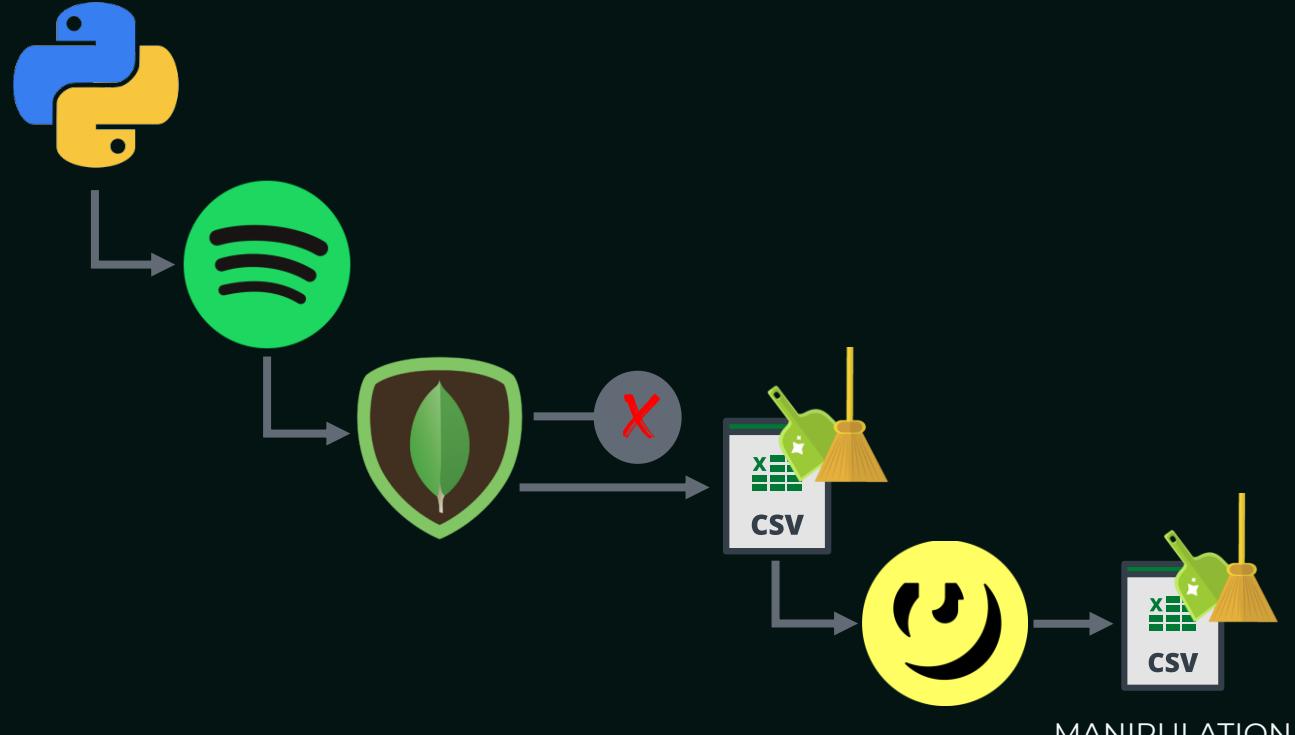
geniusCreds = "1JWlkyRPDtsPsd4ooQam5qrNzyKe7kh-G9-DMZ59BsVMaK23JoI6zAphdWQvSDiE"

df = pd.read_csv('Artisti.csv', sep = ',')

a = df["name"]
a = list(a)
artist = []
title = []
lyrics = []
conta = 0
n_song = 10
for count, c in enumerate (a):
    if (count % 1000 == 0):
        try:
            api = genius.Genius(geniusCreds)
        except Exception as e:
            print("error in client", e)
            time.sleep(10)
            api = genius.Genius(geniusCreds)
    try:
        res = api.search_artist(c, max_songs = n_song)
    except Exception as e:
        print("Error in client", e)
        api = genius.Genius(geniusCreds)
    if(res != None):
        for i in range(0,len(res.songs)):
            if (len(res.songs) != 0):
                artist.append(res.songs[i].artist)
                title.append(res.songs[i].title)
                lyrics.append(res.songs[i].lyrics)
            else:
                artist.append(c)
                title.append("?")
                lyrics.append("?")
    else:
        artist.append(c)
        title.append("?")
        lyrics.append("?")

canzoni = pd.DataFrame({"Artist": artist, "Title": title, "Lyrics": lyrics})
```

esto problema
changing name
che in caso di
capi (ogni spazio o maiuscole)
è un problema





DATA MANIPULATION

[Intro: Kyla]\r\r\r\r\nBaby, I like your style\r\r\r\r\n[Verse 1: Drake]\r\r\r\r\nGrips on your waist, front way, back way\r\r\r\r\nYou know that I don't play\r\r\r\r\nStreets not safe but I never run away\r\r\r\r\nEven when I'm away\r\r\r\r\nOtii, otii\r\r\r\r\nThere's never much love when we go OT\r\r\r\r\nI pray to make it back in one piece\r\r\r\r\nI pray, I pray\r\r\r\r\nI pray\r\r\r\r\n[Chorus: Drake]\r\r\r\r\nThat's why I need a one dance\r\r\r\r\nGot a Hennessy in my hand\r\r\r\r\nOne more time 'fore I go\r\r\r\r\nHigher powers taking a hold on me\r\r\r\r\nI need a one dance\r\r\r\r\nGot a Hennessy in my hand\r\r\r\r\nOne more time 'fore I go\r\r\r\r\nHigher powers taking a hold on me\r\r\r\r\n[Refrain: Kyla]\r\r\r\r\nBaby, I like your style\r\r\r\r\n[Verse 2: Drake]\r\r\r\r\nStrength and guidance\r\r\r\r\nAll that I'm wishing for my friends\r\r\r\r\nNobody makes it from my ends\r\r\r\r\nI had to bust up the silence\r\r\r\r\nYou know you gotta stick by me\r\r\r\r\nSoon as you see the text, reply me\r\r\r\r\nI don't wanna spend time fighting\r\r\r\r\nWe've got no time\r\r\r\r\n[Chorus: Drake]\r\r\r\r\nAnd that's why I need a one dance\r\r\r\r\nGot a Hennessy in my hand\r\r\r\r\nOne more time 'fore I go\r\r\r\r\nHigher powers taking a hold on me\r\r\r\r\nI need a one dance\r\r\r\r\nGot a Hennessy in my hand\r\r\r\r\nOne more time 'fore I go\r\r\r\r\nHigher powers taking a hold on me\r\r\r\r\n[Bridge: Wizkid]\r\r\r\r\nGot a pretty girl and she love me long time\r\r\r\r\nWine it, wine it, very long time\r\r\r\r\nOh yeah, very long time\r\r\r\r\nBack up, back up, back up and wine it\r\r\r\r\nBack up, back up and wine it, girl\r\r\r\r\nBack up, back up, back up and wine it\r\r\r\r\nOh yeah, very long time\r\r\r\r\nBack, up, back up and wine it, girl\r\r\r\r\n[Refrain: Kyla + Wizkid]\r\r\r\r\nTell me\r\r\r\r\nI need to know, where do you wanna go?\r\r\r\r\nCause if you're down, I'll take it slow\r\r\r\r\nMake you lose control\r\r\r\r\nWhere, where, where\r\r\r\r\nWhere, where, where\r\r\r\r\nOh yeah, very long time\r\r\r\r\nWhere, where, where\r\r\r\r\nBack, up, back up and wine it, girl\r\r\r\r\nWhere, where, where\r\r\r\r\nCause if you're down\r\r\r\r\nBack up, back up and\r\r\r\r\nCause if you're down\r\r\r\r\nBack up, back up and\r\r\r\r\n[Chorus: Drake]\r\r\r\r\nI need a one dance\r\r\r\r\nGot a Hennessy in my hand\r\r\r\r\nOne more time 'fore I go\r\r\r\r\nHigher powers taking a hold on me\r\r\r\r\nI need a one dance\r\r\r\r\nGot a Hennessy in my hand\r\r\r\r\nOne more time 'fore I go\r\r\r\r\nHigher powers taking a hold on me"

- Eliminazione caratteri di *escape* (ex: /n, /r, ...)
 - Eliminazione parole quali [chorus], [intro], ...

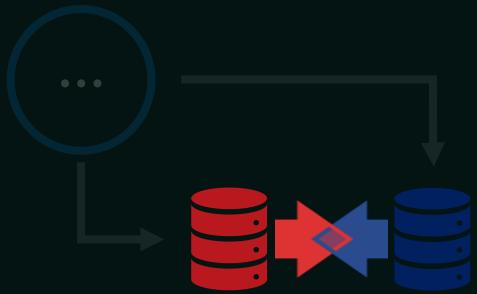
DATA MANIPULATION



- Eliminazione caratteri di *escape* (ex: /n, /r, ...)
 - Eliminazione parole quali [chorus], [intro], ...



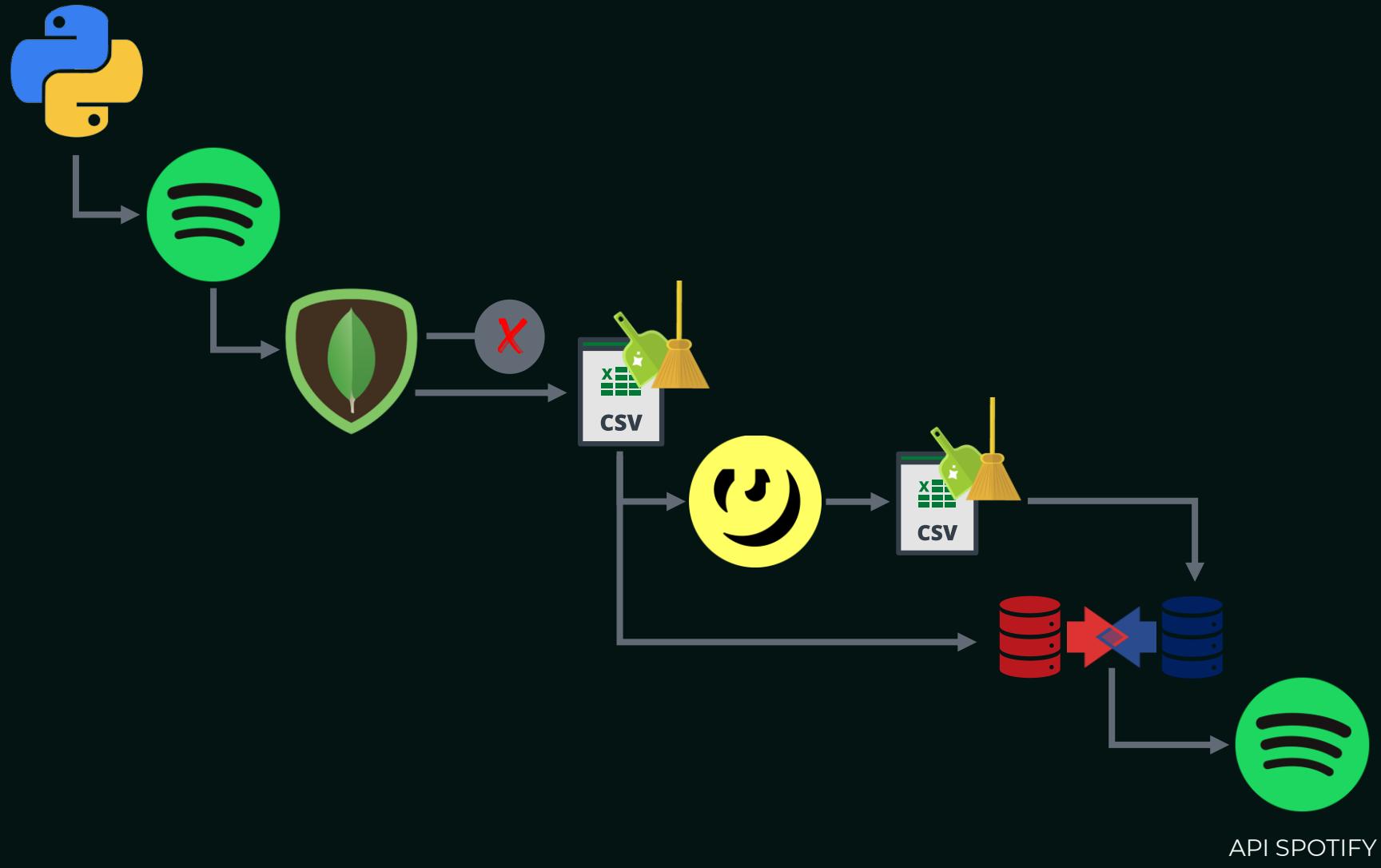
DATA INTEGRATION



Matching sul campo
Artista con
“FuzzyWuzzy” accettando
solo i match al di sopra di
una soglia di similarità da
noi empiricamente
stabilita



Matching sul campo
Artista dopo aver
accuratamente pulito il
campo da spazi
accenti, simboli et similia
per ottenere
il maggior numero di
match possibili

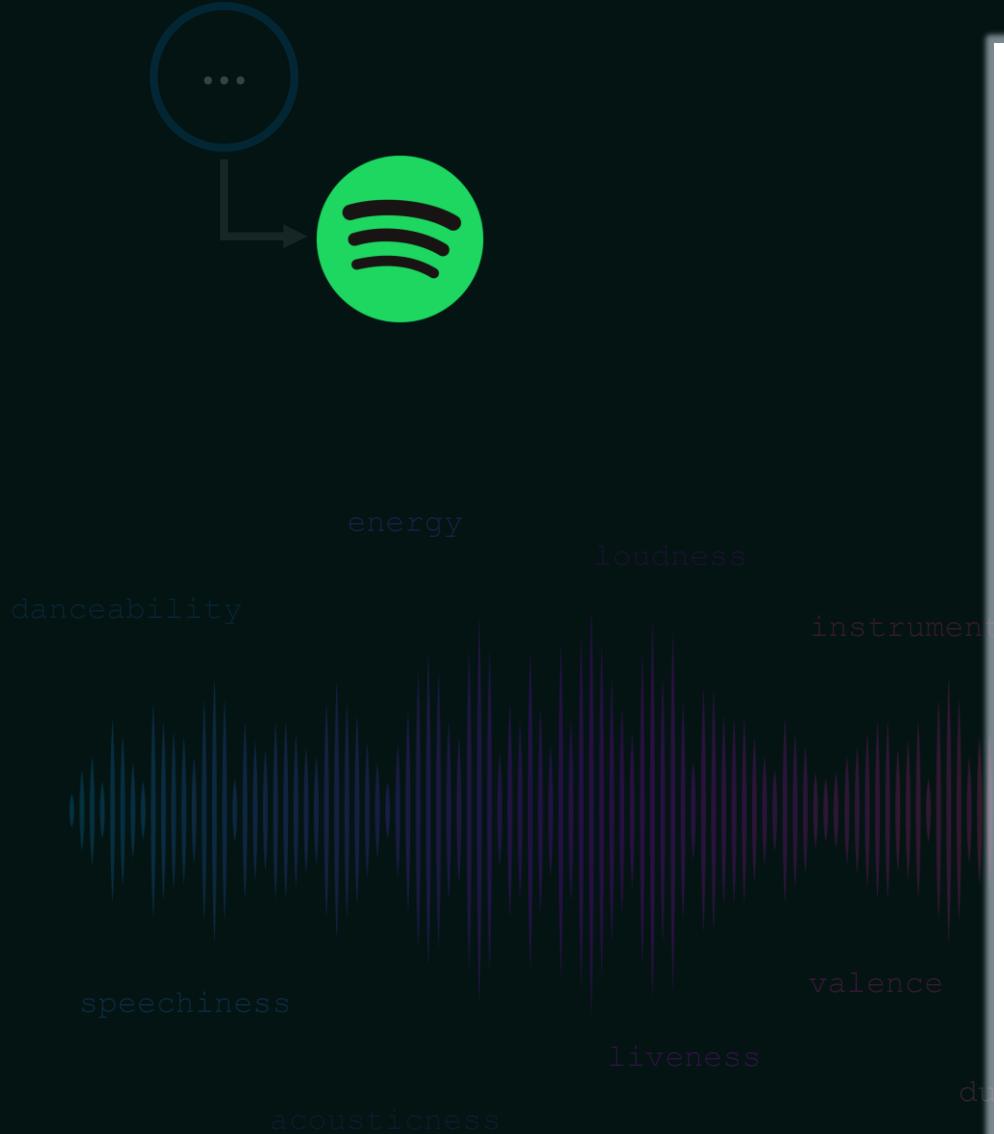




TERZA FASE DI RACCOLTA DATI (SPOTIFY)

Ottenute le canzoni e calcolato l'indice, abbiamo effettuato una seconda richiesta all'API di Spotify per raccogliere le **"Audio Feature"** di ogni singola canzone

Queste caratteristiche non verranno utilizzate poiché il nostro focus è stato posto su altro, non si esclude però che in futuro possano rivelarsi altrettanto interessanti



TERZA FASE DI

```

def find_track_ID(artist, track,i):
    try:
        track_id = sp.search(q='artist:' + artist + ' track:' + track, type='track')
        #pprint.pprint(track_id)
        ID_TRACK = track_id['tracks']['items'][0]['id']
        #print(ID_TRACK)
        POPULARITY = track_id['tracks']['items'][0]['popularity']
        #print(POPULARITY)
        RELEASE_DATE = track_id['tracks']['items'][0]['album']['release_date']
        EXTERNAL_URLS = track_id['tracks']['items'][0]['album']['external_urls']['spotify']
        AVAILABLE_MARKETS = len(track_id['tracks']['items'][0]['album']['available_markets'])

        df["ID_TRACK"][i] = ID_TRACK
        df["POPULARITY"][i] = POPULARITY
        df["RELEASE_DATE"][i] = RELEASE_DATE
        df["EXTERNAL_URLS"][i] = EXTERNAL_URLS
        df["AVAILABLE_MARKETS"][i] = AVAILABLE_MARKETS
        #audio_features
        af = sp.audio_features(str(ID_TRACK))[0]
        df["danceability"][i]= af["danceability"]
        df["energy"] [i] = af["energy"]
        df["loudness"] [i] = af["loudness"]
        df["speechiness"] [i] = af["speechiness"]
        df["acousticness"] [i] = af["acousticness"]
        df["instrumentalness"] [i] = af["instrumentalness"]
        df["liveness"] [i] = af["liveness"]
        df["valence"] [i] = af["valence"]
        df["duration_ms"] [i] = af["duration_ms"]
    except:
        print("Errore")
        df["ID_TRACK"][i] = None
        df["POPULARITY"][i] = None
        df["RELEASE_DATE"][i] = None
        df["EXTERNAL_URLS"][i] = None
        df["AVAILABLE_MARKETS"][i] = None
        df["danceability"] [i]= None
        df["energy"] [i] = None
        df["loudness"] [i] = None
        df["speechiness"] [i] = None
        df["acousticness"] [i] = None
        df["instrumentalness"] [i] = None
        df["liveness"] [i] = None
        df["valence"] [i] = None
        df["duration_ms"] [i] = None

```

PULIZIA FINALE E PREPARAZIONE

Nelle fasi finali di ottenimento dei dati sono emerse due **criticità**:



All'interno del dataset finale vi erano scrittori registi, autori di podcast e altre categorie di autori che non producono testi musicali

Abbiamo risolto il problema tenendo in considerazione solo i testi con un numero di parole maggiore di 20 e minore di 2000



Era necessario sistemare i generi al fine di poterli aggregare con facilità

Artista	Genere
Drake	Rap, Pop

Artista	Genere
Drake	Rap
Drake	Pop

PULIZIA FINALE E PREPARAZIONE

Nelle fasi finali di ottenimento dei dati sono emerse due **criticità**:



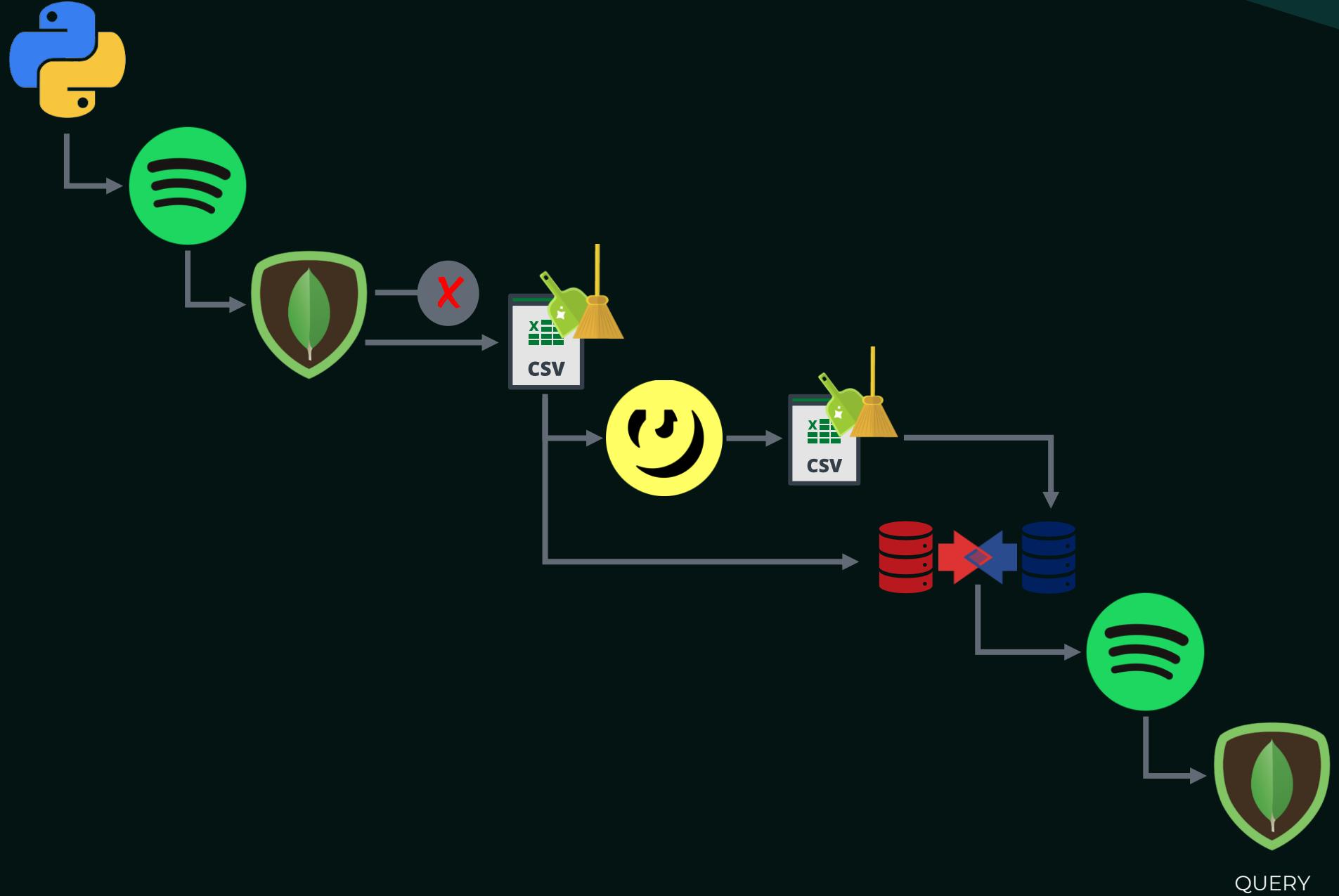
All'interno della pulizia finale erano stati registrati errori e altri dati che non erano musiche. Il problema era il fatto di non riuscire a distinguere solo i numeri di canzoni con un numero maggiore di 2000 ore di 2000.



Era necessario sistemare i generi al fine di poterli aggregare con facilità

Artista	Genere
Drake	Rap, Pop

Artista	Genere
Drake	Rap
Drake	Pop





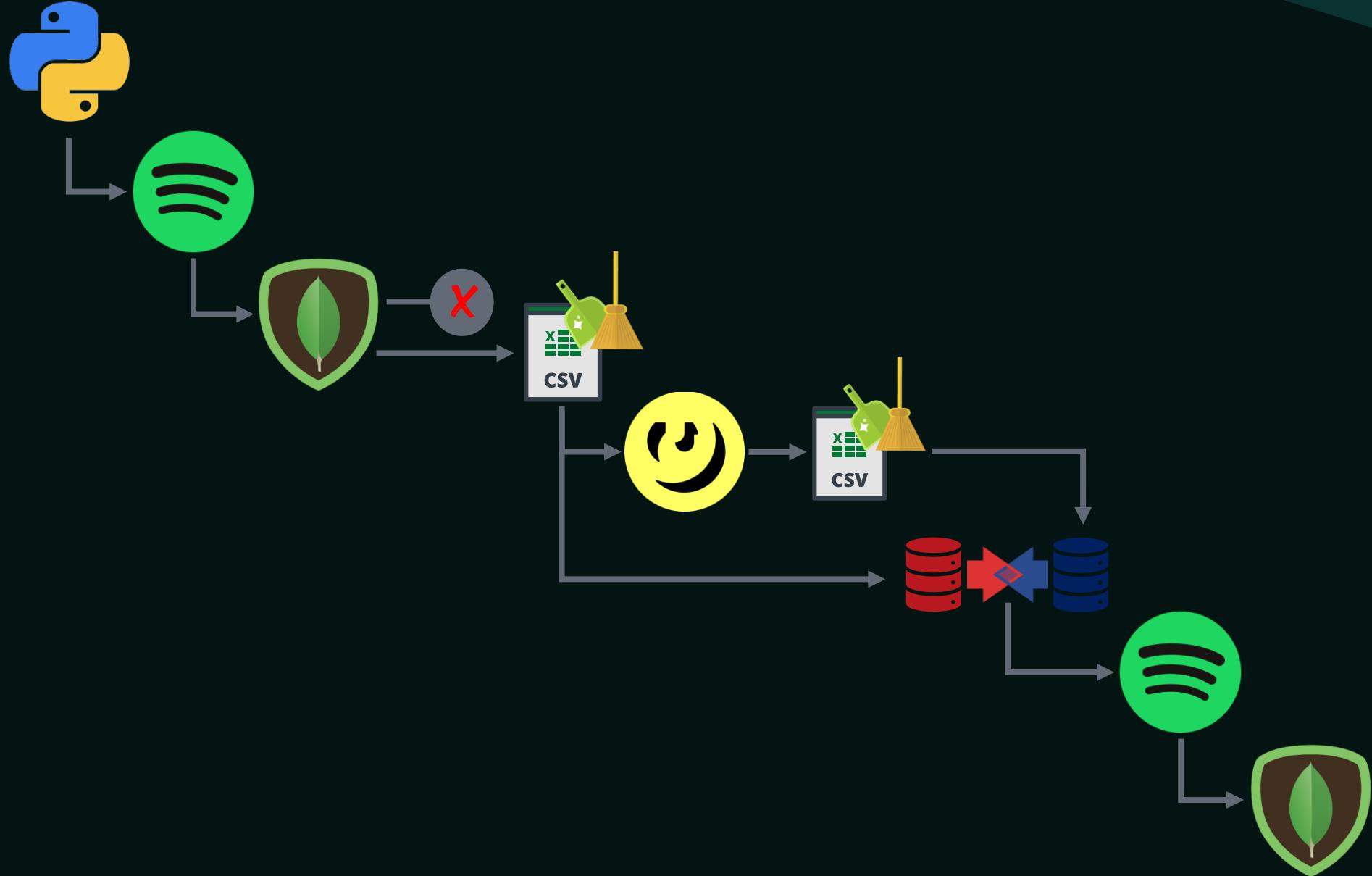
QUERY SU MONGODB

1. AGGREGAZIONE **PER ANNO** E CALCOLO DELL'INDICE DI RIPETITIVITÀ MEDIO

```
db.songs1.aggregate([{$group:{_id: "$Date", indice_avg: {avg: "$indice"} }},  
{$sort:{_id:1}}])
```

2. AGGREGAZIONE **PER GENERE** E CALCOLO DELL'INDICE DI RIPETITIVITÀ MEDIO

```
db.genres.aggregate([{$group:{_id: "$GENERI_NOSPAZIO",  
indice_avg: {avg:"$indice"} } }])
```

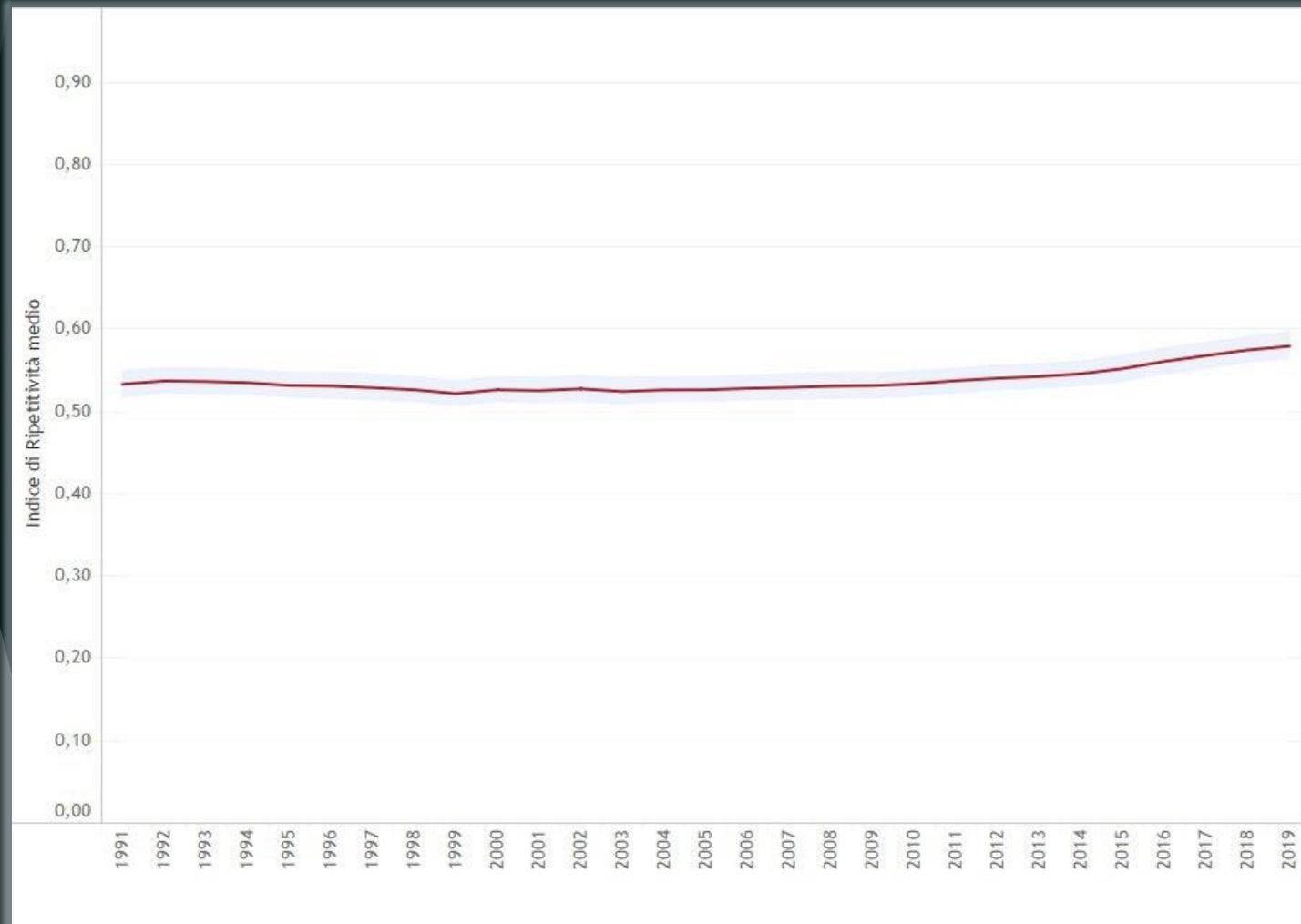


DISCUSSIONE DEI RISULTATI



Khalid

RISULTATO E COMMENTO QUERY n°1

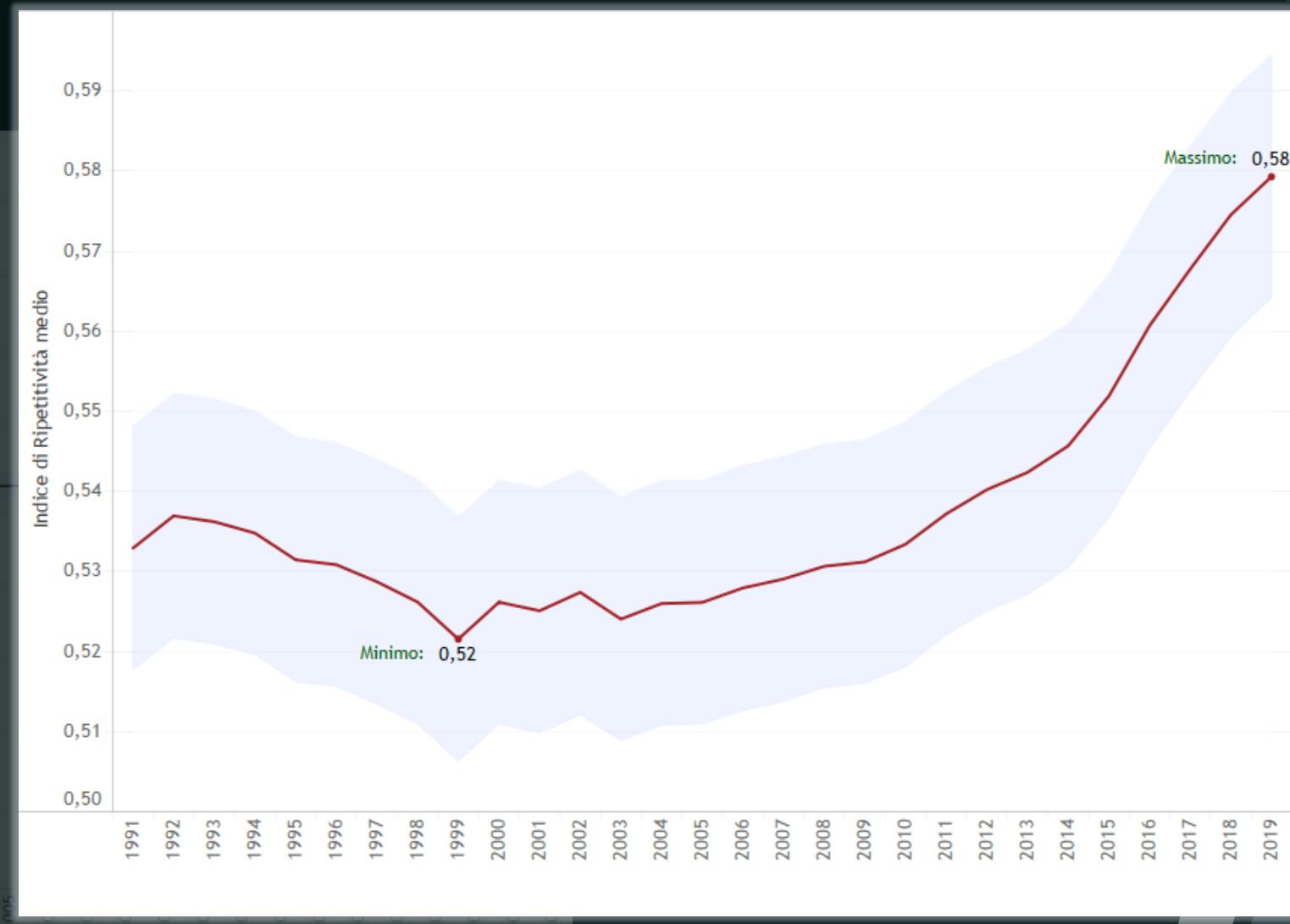


Aggregando i risultati per anno, è stato possibile notare un leggero ma chiaro **trend in salita**.

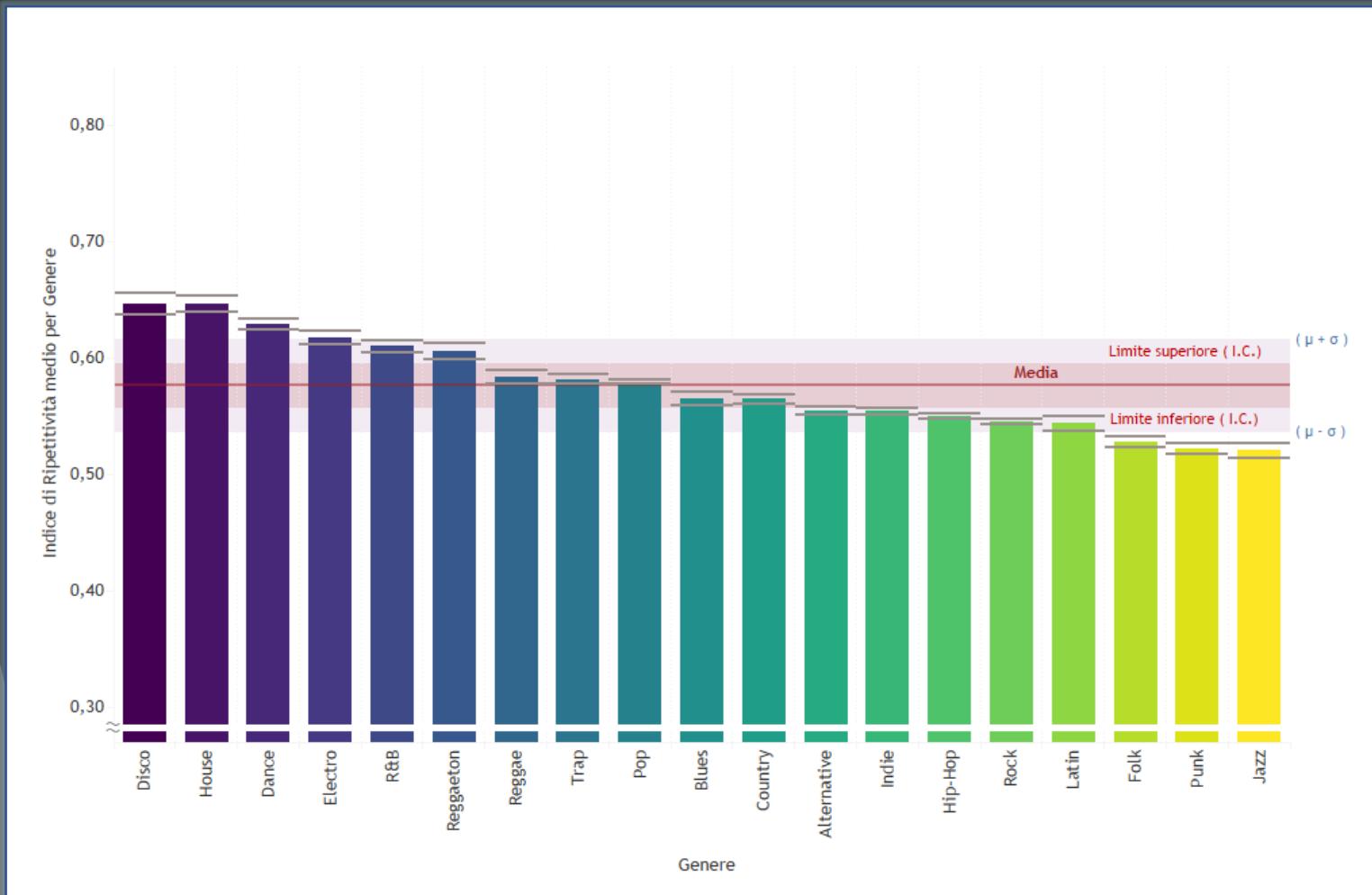
Quest'ultimo conferma che gli artisti più recenti tendano a scrivere testi sempre più ripetitivi.

RISULTATO E COMMENTO

Focalizzandosi su un range dell'Indice che va da 0,50 a 0,60, il trend viene **confermato** con ancora più evidenza.



RISULTATO E COMMENTO QUERY n°2



Contrariamente a quanto ipotizzato inizialmente, non sono la musica **Trap** e gli **altri generi** in voga attualmente a trainare questa tendenza alla ripetitività, bensì generi come la musica **Disco**, quella **Dance** e quella **House** tendono ad alzare il valore medio dell'Indice di Ripetitività.

MIGLIORAMENTI e SVILUPPI FUTURI



Shawn Mendes

MIGLIORAMENTI: SHARDING

Lo sharding è un metodo adottato al fine di distribuire i dati su più macchine. MongoDB sviluppa lo sharding per supportare implementazioni su grandi dataset e ad alto carico computazionale.



La nostra **strategia**:

- Shard Key applicata al campo **"Anno"**;
- Shard Key implementata tramite un criterio ***range-based***;
- **30 nodi** contenenti ognuno uno shard (bilanciamento dei chunk);
- Uno shard per ogni anno compreso nel **range** (1991, 2019);
- Anni precedenti al 1991 **aggregati** in un unico shard.

MIGLIORAMENTI: SHARDING

Artista		Anno
Nilla Pizzi		1950
Gianni Morandi		1970
Gorillaz		2001
Britney Spears		2001
Justin Bieber		2019
Billie Eilish		2019



Lo sharding è un metodo per suddividere i dati su più macchine. Consente di gestire implementazioni su grandi

La nostra **strategia**:

- Shard Key applicata al campo **"Anno"**;
- Shard Key implementata tramite un criterio **range-based**;
- **30 nodi** contenenti ognuno uno shard (bilanciamento dei chunk);
- Uno shard per ogni anno compreso nel **range** (1991, 2019);
- Anni precedenti al 1991 **aggregati** in un unico shard.

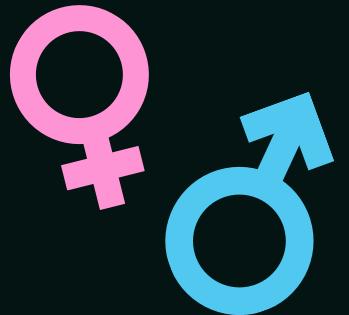
SVILUPPI FUTURI

1. Poiché *Spotify* e *Genius* mettono a disposizione non solo canzoni, ma anche libri, podcast o copioni, in futuro si potrebbe estendere la valutazione dell'Indice di Ripetitività anche a queste categorie di testi e **non solo alle canzoni.**



SVILUPPI FUTURI

2. Studiando altri dati, non si esclude di ottenere risultati significativi utilizzando altri tipi di informazioni, quali le **caratteristiche della traccia** audio (già presenti nel dataset e inutilizzate) o le **informazioni anagrafiche** riguardanti gli artisti.



SVILUPPI FUTURI



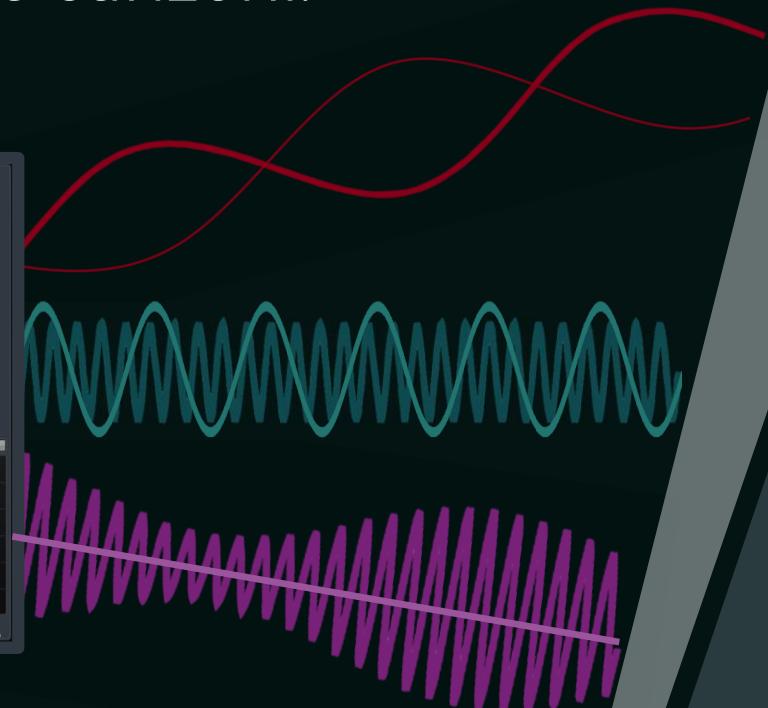
3. Attraverso tecniche di analisi tipiche del mondo del **Text Mining** si terrebbe conto anche della semantica delle parole nel calcolo dell'Indice di Ripetitività, il quale potrebbe subire cambiamenti degni di nota.

SVILUPPI FUTURI

- 
- 
4. Potrebbe essere interessante approfondire in che misura questa ripetitività dipenda dagli **autori** delle canzoni, dai **produttori** o se sia imputabile anche alla **casa discografica**, la quale stipula contratti solo con artisti che garantiscono un alto **ritorno economico** spesso a discapito dell'originalità dei testi.

SVILUPPI FUTURI

5. Avendo a disposizione dati relativi alla base musicale di ogni canzone, sarebbe interessante raccoglierne di nuovi e analizzare la canzone nella sua globalità, cercando anche similarità con altre canzoni.



A photograph of a person from behind, wearing dark headphones and working on a laptop. The screen shows a software interface with multiple windows. The scene is set against a warm, orange sunset sky. In the foreground, there's a small glowing candle and some blurred objects.

**GRAZIE
PER L'ATTENZIONE!**