Detecção de Músicas Anômalas no Top 200

Este notebook implementa métodos de análise de anomalias *(outliers)* com o objetivo de encontrar músicas que chegaram ao Top 200 mesmo apresentando caracterísitcas que fogem do padrão do ranking.

Um exemplo curioso: será que Gangnan Style, quando foi lançada e popularizada, possuia as mesmas características das músicas mais ouvidas daquele ano?

Breve Explicação dos Modelos Utilizados

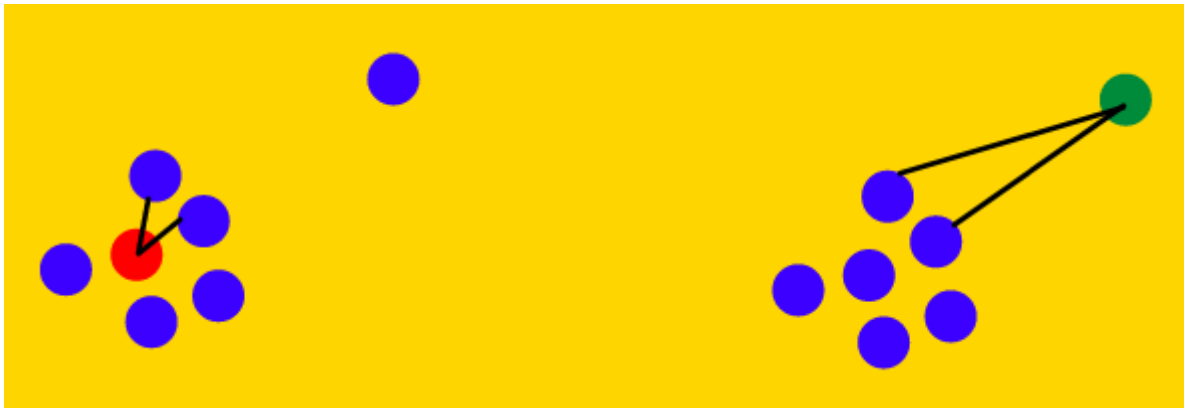
ABOD

O Angle Based Outlier Detection, como o nome sugere, utiliza os ângulos entre trincas de pontos do espaço de variáveis *(feature space)* para detectar outliers.

O algoritmo baseia-se na seguinte propriedade dos ângulos entre os pontos: quanto mais anômalo o ponto **P** em relação aos demais pontos do dataset (**A** e **B** para formar a trinca e podermos calcular um ângulo), menor será a variação do ângulo **APB** ao variamos **A** e **B**. Por outro lado, quanto mais *clusterizados* os 3 pontos, maior será a variação dos ângulos com a escolha de diferentes **A** e **B**.

A imagem abaixo resume bem essa ideia.

O ponto em verde é outlier e o vermelho, não. Note que, ao escolher diferentes pontos azuis, o ângulo obtido vai variar muito mais para o ponto vermelho do que para o verde.



O algoritmo em si, basicamente, calcula esses ângulos θ para cada trinca de pontos (ou para os *k* vizinhos mais próximos) seguindo a seguinte fórmula:

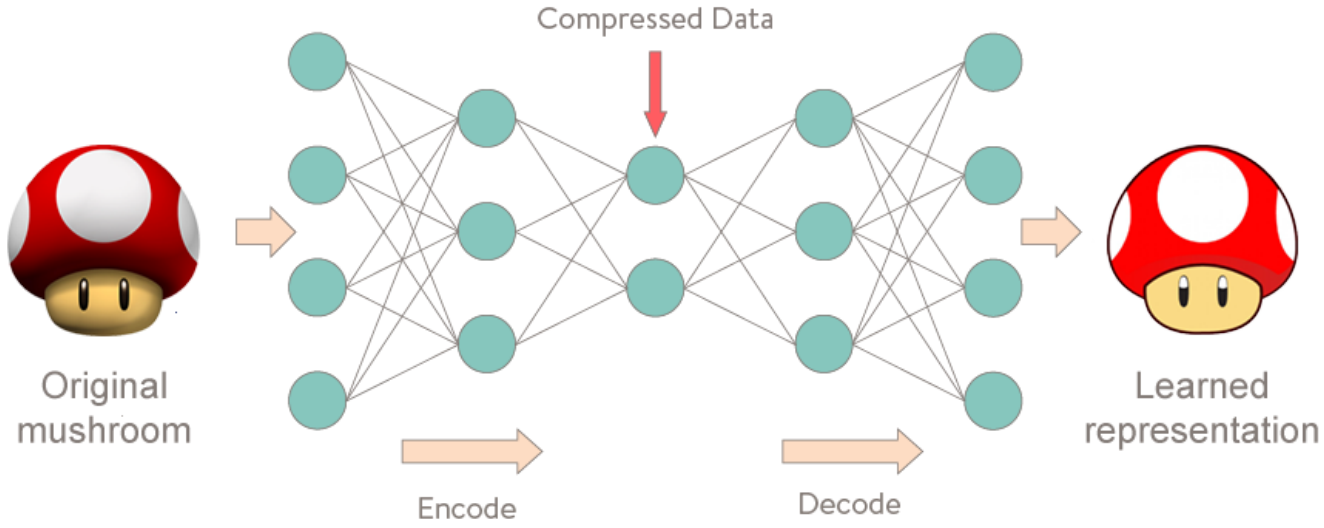
$cos\theta = \frac{\vec{A} \cdot \vec{B}}{A^2 B^2}$ onde $\vec{A} = \overline{\mathrm{AP}}$ e $\vec{B} = \overline{\mathrm{BP}}$, $\vec{A} \cdot \vec{B}$ é o produto interno entre A e B, e A^2B^2 é o produto dos módulos ao quadrado.

A partir dessa expressão, calcula-se a **variância** de θ para todo A e B, dado um P fixo. Se a variância for grande, P é classificado como *inliner* e, caso seja pequena, como $outlier$.

Autoencoder

O funcionamento básico de uma rede neural *autoencoder* é que ela aprende comprime e reconstrói os dados de entrada. Um bom autoencoder é capaz de gerar essa reconstrução extremamente similar aos dados originais que foram utilizados como entrada do algoritmo, ou seja, o erro entre a reconstrução e os dados originais deve ser o menor possível.

A imagem abaixo resume a intuição por detrás do algoritmo.



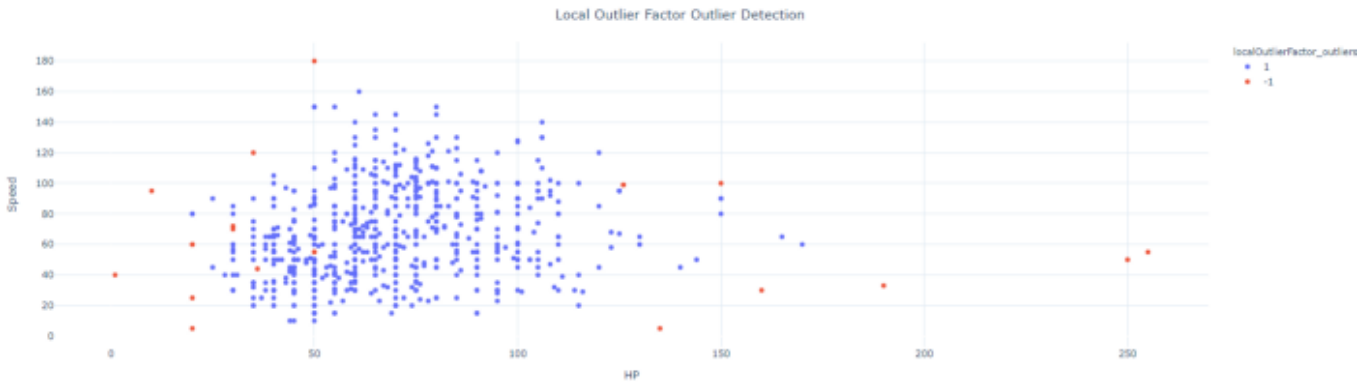
A ideia por detrás de usar essas redes neurais para detecção de anomalias é que, ao apresentar um dado anômalo, o erro ao recontrui-lo será maior que o esperado, uma vez que o algoritmo aprendeu a reconstruir os dados a partir de um leque específico e comum de características e, um dado anômlado, por definição, apresenta alguma irregularidade.

Sendo assim, os pontos com os maiores erros de reconstrução podem ser considerados possíveis *outliers*.

LoOP

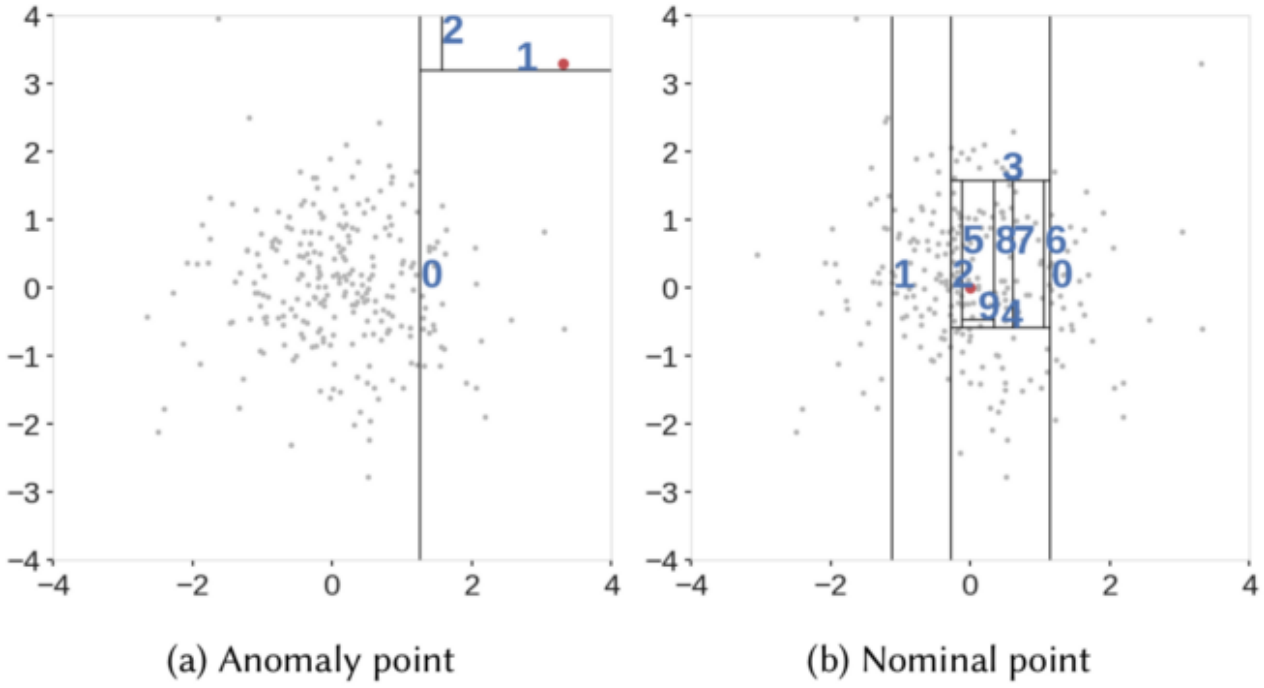
O LoOP é um algoritmo baseado em densidade que calcula e comparada a densidade de cada ponto com seus vizinhos. Pontos anômalos estão, normalmente, mais espalhados pelo espaço de variáveis e, com isso, apresentam menor densidade em sua vizinhança. Se a densidade de um ponto for muito menor que a dos seus vizinhos (LOF>>1), o ponto está longe de áreas de alta densidade e é, portanto, um possível outlier.

O LoOP é uma versão mais robusta do LOF (Local Outlier Factor), que retorna a probabilidade do ponto ser outiler.



Isolation Forest

O modelo de Isolation Forest é construído em cima de Random Forests que, por sua vez, é um ensemble de árvores de decisão. Esse algoritmo utiliza o fato de que observações anômalas são menos frequentes e relevantemente distoantes das demais, de modo que, durante as divisões internas dos ramos da árvore de decisão, essas variáveis normalmente são identificadas mais próximas à raiz da árvore. Uma nota de anomalia é, então, gerada a partir do tamanho médio do caminho até a variável ser escolhida dentre todas as árvores da floresta, sendo que, quando menor o caminho (mais perto da raiz), maior o score de outlier.



Variável booleana que diz se preciso fazer uma nova consulta à API para gerar um novo dataset de musicas

In [ ]:

```
playlist_existente = True
```

```
time: 1.15 ms (started: 2022-06-21 16:24:33 +00:00)
```

In [ ]:

```
!pip install ipython-autotime
%load_ext autotime
!pip install pyod
!pip install spotipy

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm # tqdm mostra barra de processo https://github.com/tqdm/tqdm

import os
import glob

import spotipy
from spotipy.oauth2 import SpotifyClientCredentials

from pyod.models.abod import ABOD
from pyod.models.auto_encoder import AutoEncoder
from pyod.models.iforest import IForest
from pyod.models.lof import LOF


from google.colab import drive
drive.mount('/content/drive')
```

Criando dataset global

In [ ]:

```
# Usuário: fijita8647@karavic.com
# Senha: INF10322803
cid ="17fd5fc78c79479fb19bd36652021b46"
secret = "85ee26c27c4f4601a3c6aef838a80e89"

client_credentials_manager = SpotifyClientCredentials(client_id=cid, client_secret=secr
et)
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)
```

time: 8.53 ms (started: 2022-06-21 16:24:46 +00:00)

O path abaixo é para o Top 200 global

In [ ]:

```
path_global = "/content/drive/MyDrive/INF1032 - Spotify/Dados/Anna/csv/global"
```

time: 1.36 ms (started: 2022-06-21 16:24:46 +00:00)

A funcao abaixo monta um dataframe com as URIs de todas as musicas que entraram em um dado Top 200

In [ ]:

```
def pega_uri_charts(path):

  #Pega todos os arquivos do diretorio
  arquivos = glob.glob(os.path.join(path,"*.csv"))
  dados = pd.DataFrame()

  for file in arquivos:
      dados_temp = pd.read_csv(file)
      #Coloca o nome certo nas colunas
      dados_temp.columns = dados_temp.iloc[0]
      dados_temp = dados_temp.drop(axis=0,index=0)
      #Vai concatenando no dataet final
      dados = pd.concat([dados,dados_temp])

  #Preciso apenas do URI
  dados = dados[["URL","Streams"]]

  #Para analise de anomalias nao preciso da informacao de quantas vezes a musica aparec
eu no top 200, entao removo duplicatas
  dados = dados.drop_duplicates(keep="first")

  #Reset no index, ja que removi linhas
  dados = dados.reset_index(drop=True)

  return dados
```

time: 23 ms (started: 2022-06-21 16:24:46 +00:00)

A função abaixo pega as features de um track via URI do track

In [ ]:

```
def get_features_from_tracks(track_id):
  #Pega as features da musica e outras informações importantes e/ou interessantes

  #Audio features
  features = sp.audio_features(track_id)

  #Informacoes sobre a musica
  data_lancamento = sp.track(track_id)['album']['release_date']
  nome = sp.track(track_id)['name']
  popularidade_musica = sp.track(track_id)['popularity']

  #Informacoes sobre o artista
  nome_artista = sp.track(track_id)['artists'][0]['name']
  uri_artista = sp.track(track_id)["artists"][0]['uri']
  popularidade_artista = sp.artist(uri_artista)['popularity']
  estilo = sp.artist(uri_artista)['genres']


  #Montando o dataframe
  features = pd.DataFrame(features)

  features["nome"] = nome
  features["data_lancamento"] = data_lancamento
  features["data_lancamento"] = pd.to_datetime(features["data_lancamento"])
  features["mes_lancamento"] = features["data_lancamento"].dt.month
  features["dia_lancamento"] = features["data_lancamento"].dt.day
  features["dia_semana_lancamento"] = features["data_lancamento"].dt.day_name()

  features["artista"] = nome_artista
  features["popularidade_musica"] = popularidade_musica
  features["popularidade_artista"] = popularidade_artista
  features["estilo_musical"] = estilo[-1] #sempre é uma lista e eu, arbritariamente, es
colhi pegar o ultimo estilo da lista


  return features
```

```
time: 26.8 ms (started: 2022-06-21 16:24:46 +00:00)
```

A função abaixo pega as audio features de cada um dos tracks e monta um dataframe

In [ ]:

```
def gera_dataset(lista_uris_tracks):
  df_features_final = pd.DataFrame()
  for idx,track_uri in lista_uris_tracks.iterrows():

    #Eu pego as features da musica
    df_features_parcial = get_features_from_tracks(track_uri.URL)

    df_features_parcial["Streams"] = track_uri.Streams
    # E concateno no df final
    df_features_final = pd.concat([df_features_final,df_features_parcial],axis=0)


  if("Unnamed: 0" in df_features_final.columns):
    df_features_final = df_features_final.drop(axis=1,columns=["Unnamed: 0"])


  return df_features_final
```

time: 7.14 ms (started: 2022-06-21 16:24:46 +00:00)

In [ ]:

```
path_top_global = '/content/drive/My Drive/INF1032 - Spotify/Dados/Matheus/MusicasAnomalas/'

if playlist_existente:
  df_musicas_global = pd.read_csv(path_top_global+"top200_global_consolidado.csv")
  df_musicas_boas = pd.read_csv("/content/drive/My Drive/INF1032 - Spotify/Dados/Consolidados/musicas_boas.csv")
  df_musicas_ruins = pd.read_csv("/content/drive/My Drive/INF1032 - Spotify/Dados/Consolidados/musicas_ruins.csv")
else:
  lista_uris_global = pega_uri_charts(path_global)
  df_musicas_global = gera_dataset(lista_uris_tracks=lista_uris_global)
  df_musicas_global.to_csv(path_top_global+"top200_global_consolidado.csv")
```

time: 176 ms (started: 2022-06-21 16:24:46 +00:00)

In [ ]:

```
df_musicas_global.head(10)
```

Out[ ]:

	Unnamed: 0	danceability	energy	key	loudness	mode	speechiness	acousticness	instru
0	0	0.520	0.731	6	-5.338	0	0.0557	0.3420	
1	0	0.905	0.563	8	-6.135	1	0.1020	0.0254	
2	0	0.761	0.525	11	-6.900	1	0.0944	0.4400	
3	0	0.591	0.764	1	-5.484	1	0.0483	0.0383	
4	0	0.756	0.697	8	-6.377	1	0.0401	0.1820	
5	0	0.728	0.783	11	-4.424	0	0.2660	0.2370	
6	0	0.795	0.800	1	-6.320	1	0.0309	0.0354	
7	0	0.741	0.691	10	-7.395	0	0.0672	0.0221	
8	0	0.812	0.736	4	-5.421	0	0.0833	0.1520	
9	0	0.870	0.548	10	-5.253	0	0.0770	0.0924	

```
time: 28.1 ms (started: 2022-06-21 16:24:46 +00:00)
```

Aquisicao dos dados vindo das Top Hits

Esses dados não são adequados para a analise de anomalias por causa da janela temporal grande

In [ ]:

```
df_musicas_boas.columns
```

Out[ ]:

```
Index(['Unnamed: 0', 'danceability', 'energy', 'key', 'loudness', 'mode',
       'speechiness', 'acousticness', 'instrumentalness', 'liveness',
       'valence', 'tempo', 'type', 'id', 'uri', 'track_href', 'analysis_ur
l',
       'duration_ms', 'time_signature', 'nome', 'data_lancamento',
       'Popularidade Musica', 'Artista', 'ano_lancamento', 'mes_lancament
o',
       'dia_semana_lancamento', 'Popularidade Artista', 'Seguidores',
       'Estilos'],
      dtype='object')
```

time: 5.41 ms (started: 2022-06-21 16:24:46 +00:00)

In [ ]:

```
features_de_interesse = ['danceability','energy', 'loudness', 'speechiness', 'acousticn
ess', 'instrumentalness', 'liveness', 'valence','duration_ms','key', 'mode', 'tempo']
features_de_interesse2 = ['danceability','energy', 'loudness', 'speechiness', 'acoustic
ness', 'instrumentalness', 'liveness', 'valence','duration_ms','key', 'mode', 'tempo',
'Popularidade Musica', 'ano_lancamento', 'mes_lancamento',
       'dia_semana_lancamento', 'Popularidade Artista', 'Seguidores']

df_input = df_musicas_global[features_de_interesse]
df_input_boas = df_musicas_boas[features_de_interesse2]
df_input_ruins = df_musicas_ruins[features_de_interesse2]

df_resultados = df_musicas_global.copy(deep=True)
df_resultados = df_resultados.drop(axis=1,columns=['type', 'id', 'uri', 'track_href',
'analysis_url','time_signature'])
df_resultados = df_resultados.reset_index(drop=True)
# df_input = df_input.reset_index(drop=True)

df_resultados_boas = df_musicas_boas.copy(deep=True)
df_resultados_boas = df_resultados_boas.dropna()
df_resultados_boas = df_resultados_boas.reset_index(drop=True)
df_resultados_boas = df_resultados_boas.drop(axis=1,columns=['type', 'id', 'uri', 'trac
k_href', 'analysis_url','time_signature','Estilos','data_lancamento','Artista'])
# df_input_boas = df_input_boas.reset_index(drop=True)

df_resultados_ruins = df_musicas_ruins.copy(deep=True)
df_resultados_ruins = df_resultados_ruins.dropna()
df_resultados_ruins = df_resultados_ruins.reset_index(drop=True)
df_resultados_ruins = df_resultados_ruins.drop(axis=1,columns=['type', 'id', 'uri', 'tr
ack_href', 'analysis_url','time_signature','Estilos','data_lancamento','Artista'])
# df_input_ruins = df_input_ruins.reset_index(drop=True)
```

time: 52 ms (started: 2022-06-21 16:24:46 +00:00)

Analisando os valores existentes para cada uma dessas features

Esse passo é importante pois alguns dos modelos de detecção de anomalias precisam dos valores de input em um dado formato.

O autoencoder, por exemplo, por ser uma rede neural, pode precisar dos valores normalizados entre [0,1] ou [-1,1].

In [ ]:

```
for feature in features_de_interesse:
  max = df_input[[feature]].max()
  min = df_input[[feature]].min()
  print("Coluna %s:\tMax = %.2f\tMin = %.2f"%(feature,max,min))
```

```
Coluna danceability:    Max = 0.98      Min = 0.15
Coluna energy:  Max = 0.99      Min = 0.03
Coluna loudness:        Max = 1.51      Min = -34.48
Coluna speechiness:     Max = 0.97      Min = 0.02
Coluna acousticness:    Max = 0.99      Min = 0.00
Coluna instrumentalness:        Max = 0.95      Min = 0.00
Coluna liveness:        Max = 0.98      Min = 0.02
Coluna valence: Max = 0.98      Min = 0.03
Coluna duration_ms:     Max = 690732.00 Min = 30133.00
Coluna key:     Max = 11.00     Min = 0.00
Coluna mode:    Max = 1.00      Min = 0.00
Coluna tempo:   Max = 212.12    Min = 46.72
time: 53.8 ms (started: 2022-06-21 16:24:46 +00:00)
```

In [ ]:

```
df_input_boas = df_input_boas.dropna()
df_input_ruins = df_input_ruins.dropna()
df_input.isna().sum()
```

Out[ ]:

```
danceability        0
energy              0
loudness            0
speechiness         0
acousticness        0
instrumentalness    0
liveness            0
valence             0
duration_ms         0
key                 0
mode                0
tempo               0
dtype: int64

time: 20.8 ms (started: 2022-06-21 16:24:46 +00:00)
```

A maior parte das colunas está entre [0,1]. Apenas *loudness* e *duration_ms* que, talvez, precisem ser ajustadas, a depender do método utilizado.

Implementação dos métodos de deteção de anomalias

In [ ]:

```
def gera_prob_clssificacao(df,metodo,lista_proba):
  prob_normal = []
  prob_outlier = []
  for prob in lista_proba[0]:
      prob_normal.append(prob[0])
      prob_outlier.append(prob[1])

  prob_classif = []
  for i in range(len(df[metodo])):
      if df[metodo][i]==1:
          prob_classif.append(prob_outlier[i])
      else:
          prob_classif.append(prob_normal[i])

  return prob_classif
```

time: 9.21 ms (started: 2022-06-21 16:24:46 +00:00)

In [ ]:

```
#Para mais de um modelo
contaminacao = 0.05
num_vizinhos = 50

#Autoencoders
num_neuronios = df_input.shape[1]
neuronios = [num_neuronios,num_neuronios//2,num_neuronios//4,num_neuronios//4,num_neuro
nios//2,num_neuronios]

num_neuronios_boas = df_input_boas.shape[1]
neuronios_boas = [num_neuronios_boas,num_neuronios_boas//2,num_neuronios_boas//4,num_ne
uronios_boas//4,num_neuronios_boas//2,num_neuronios_boas]

num_neuronios_ruins = df_input_ruins.shape[1]
neuronios_ruins = [num_neuronios_ruins,num_neuronios_ruins//2,num_neuronios_ruins//4,nu
m_neuronios_ruins//4,num_neuronios_ruins//2,num_neuronios_ruins]
```

time: 7.71 ms (started: 2022-06-21 16:24:46 +00:00)

ABOD

In [ ]:

```
modelo_abod = ABOD(contamination=contaminacao,method='fast',n_neighbors=num_vizinhos).f
it(df_input)
resultado_abod = modelo_abod.predict(df_input, return_confidence=True)
probs_abod = modelo_abod.predict_proba(df_input,method="unify", return_confidence=True)
```

time: 6min 16s (started: 2022-06-21 16:24:46 +00:00)

In [ ]:

```
modelo_abod_boas = ABOD(contamination=contaminacao,method='fast',n_neighbors=num_vizinh
os).fit(df_input_boas)
resultado_abod_boas = modelo_abod_boas.predict(df_input_boas, return_confidence=True)
probs_abod_boas = modelo_abod_boas.predict_proba(df_input_boas,method="unify", return_c
onfidence=True)
```

time: 5min 59s (started: 2022-06-21 16:31:02 +00:00)

In [ ]:

```
modelo_abod_ruins = ABOD(contamination=contaminacao,method='fast',n_neighbors=num_vizin
hos).fit(df_input_ruins)
resultado_abod_ruins = modelo_abod_ruins.predict(df_input_ruins, return_confidence=True
)
probs_abod_ruins = modelo_abod_ruins.predict_proba(df_input_ruins,method="unify", retur
n_confidence=True)
```

time: 4min 9s (started: 2022-06-21 16:37:02 +00:00)

In [ ]:

```
df_resultados["abod"] = resultado_abod[0]
df_resultados["prob abod"] = gera_prob_clssificacao(df_resultados,"abod",probs_abod)

df_resultados_boas["abod"] = resultado_abod_boas[0]
df_resultados_boas["prob abod"] = gera_prob_clssificacao(df_resultados_boas,"abod",prob
s_abod_boas)

df_resultados_ruins["abod"] = resultado_abod_ruins[0]
df_resultados_ruins["prob abod"] = gera_prob_clssificacao(df_resultados_ruins,"abod",pr
obs_abod_ruins)

mask1 = df_resultados["abod"]==1
mask2 = df_resultados["prob abod"]>=0.5
mask = mask1 & mask2
df_resultados[mask]
```

Out[ ]:

Unnamed: 0	danceability	energy	key	loudness	mode	speechiness	acousticness	instrum

time: 118 ms (started: 2022-06-21 16:41:12 +00:00)

Autoencoder

In [ ]:

```
modelo_autoencoder = AutoEncoder(hidden_neurons=neuronios,contamination=contaminacao,verbose=0,random_state=0).fit(df_input)
resultado_autoencoder = modelo_autoencoder.predict(df_input, return_confidence=True)
probs_autoencoder = modelo_autoencoder.predict_proba(df_input,method="unify", return_confidence=True)
```

time: 35 s (started: 2022-06-21 16:41:12 +00:00)

In [ ]:

```
modelo_autoencoder_boas = AutoEncoder(hidden_neurons=neuronios_boas,contamination=contaminacao,verbose=0,random_state=0).fit(df_input_boas)
resultado_autoencoder_boas = modelo_autoencoder_boas.predict(df_input_boas, return_confidence=True)
probs_autoencoder_boas = modelo_autoencoder_boas.predict_proba(df_input_boas,method="unify", return_confidence=True)
```

time: 45.2 s (started: 2022-06-21 16:41:47 +00:00)

In [ ]:

```
modelo_autoencoder_ruins = AutoEncoder(hidden_neurons=neuronios_ruins,contamination=contaminacao,verbose=0,random_state=0).fit(df_input_ruins)
resultado_autoencoder_ruins = modelo_autoencoder_ruins.predict(df_input_ruins, return_confidence=True)
probs_autoencoder_ruins = modelo_autoencoder_ruins.predict_proba(df_input_ruins,method="unify", return_confidence=True)
```

time: 31.6 s (started: 2022-06-21 16:42:32 +00:00)

In [ ]:

```
df_resultados["autoencoder"] = resultado_autoencoder[0]
df_resultados["prob autoencoder"] = gera_prob_clssificacao(df_resultados,"autoencoder",
probs_autoencoder)

df_resultados_boas["autoencoder"] = resultado_autoencoder_boas[0]
df_resultados_boas["prob autoencoder"] = gera_prob_clssificacao(df_resultados_boas,"aut
oencoder",probs_autoencoder_boas)

df_resultados_ruins["autoencoder"] = resultado_autoencoder_ruins[0]
df_resultados_ruins["prob autoencoder"] = gera_prob_clssificacao(df_resultados_ruins,"a
utoencoder",probs_autoencoder_ruins)

mask1 = df_resultados["autoencoder"]==1
mask2 = df_resultados["prob autoencoder"]>=0.9
mask = mask1 & mask2
df_resultados[mask]
```

Out[ ]:

	Unnamed: 0	danceability	energy	key	loudness	mode	speechiness	acousticness	ins
51	0	0.332	0.225	0	-8.697	1	0.0348	0.76700	
71	0	0.684	0.449	3	-9.738	1	0.6110	0.86900	
81	0	0.426	0.225	8	-12.422	1	0.0291	0.82300	
111	0	0.672	0.745	5	-5.269	0	0.3000	0.43800	
121	0	0.746	0.251	11	-16.169	0	0.2590	0.78200	
...	...	...	...	...	...	...	...	...	
5107	0	0.586	0.740	1	-2.997	0	0.4040	0.55500	
5112	0	0.394	0.327	4	-14.291	1	0.1140	0.84900	
5118	0	0.544	0.369	2	-9.514	1	0.0380	0.96900	
5143	0	0.412	0.881	2	-3.502	1	0.0870	0.00039	
5155	0	0.732	0.831	0	-3.791	0	0.0691	0.55400	

258 rows × 18 columns

time: 147 ms (started: 2022-06-21 16:43:04 +00:00)

LOF

In [ ]:

```
modelo_lof = LOF(contamination=contaminacao,n_neighbors=num_vizinhos,algorithm='auto').
fit(df_input)
resultado_lof = modelo_lof.predict(df_input, return_confidence=True)
probs_lof = modelo_lof.predict_proba(df_input,method="unify", return_confidence=True)
```

time: 1.24 s (started: 2022-06-21 16:43:04 +00:00)

In [ ]:

```
modelo_lof_boas = LOF(contamination=contaminacao,n_neighbors=num_vizinhos,algorithm='auto').fit(df_input_boas)
resultado_lof_boas = modelo_lof_boas.predict(df_input_boas, return_confidence=True)
probs_lof_boas = modelo_lof_boas.predict_proba(df_input_boas,method="unify", return_confidence=True)
```

time: 3.85 s (started: 2022-06-21 16:43:05 +00:00)

In [ ]:

```
modelo_lof_ruins = LOF(contamination=contaminacao,n_neighbors=num_vizinhos,algorithm='auto').fit(df_input_ruins)
resultado_lof_ruins = modelo_lof_ruins.predict(df_input_ruins, return_confidence=True)
probs_lof_ruins = modelo_lof_ruins.predict_proba(df_input_ruins,method="unify", return_confidence=True)
```

time: 2.55 s (started: 2022-06-21 16:43:09 +00:00)

In [ ]:

```
df_resultados["lof"] = resultado_lof[0]
df_resultados["prob lof"] = gera_prob_clssificacao(df_resultados,"lof",probs_lof)

df_resultados_boas["lof"] = resultado_lof_boas[0]
df_resultados_boas["prob lof"] = gera_prob_clssificacao(df_resultados_boas,"lof",probs_
lof_boas)

df_resultados_ruins["lof"] = resultado_lof_ruins[0]
df_resultados_ruins["prob lof"] = gera_prob_clssificacao(df_resultados_ruins,"lof",prob
s_lof_ruins)
mask1 = df_resultados["lof"]==1
mask2 = df_resultados["prob lof"]>=0.9
mask = mask1 & mask2
df_resultados[mask]
```

Out[ ]:

	Unnamed: 0	danceability	energy	key	loudness	mode	speechiness	acousticness	ins
111	0	0.672	0.745	5	-5.269	0	0.3000	0.438000	
369	0	0.418	0.106	8	-22.507	0	0.0448	0.994000	
444	0	0.526	0.328	1	-9.864	1	0.0461	0.694000	
641	0	0.588	0.479	1	-7.039	1	0.2810	0.604000	
677	0	0.837	0.840	7	-4.293	1	0.2390	0.062100	
...	...	...	...	...	...	...	...	...	
4938	0	0.269	0.591	10	-5.790	0	0.0416	0.453000	
5050	0	0.574	0.664	11	-6.068	1	0.0409	0.044800	
5072	0	0.408	0.875	8	-1.536	1	0.0546	0.000017	
5093	0	0.780	0.768	6	-4.325	0	0.2380	0.037100	
5107	0	0.586	0.740	1	-2.997	0	0.4040	0.555000	

61 rows × 20 columns

time: 138 ms (started: 2022-06-21 16:43:12 +00:00)

IF

In [ ]:

```
modelo_if = IForest(contamination=contaminacao,random_state=0,).fit(df_input)
resultado_if = modelo_if.predict(df_input, return_confidence=True)
probs_if = modelo_if.predict_proba(df_input,method="unify", return_confidence=True)
```

In [ ]:

```
modelo_if_boas = IForest(contamination=contaminacao,random_state=0,).fit(df_input_boas)
resultado_if_boas = modelo_if_boas.predict(df_input_boas, return_confidence=True)
probs_if_boas = modelo_if_boas.predict_proba(df_input_boas,method="unify", return_confi
dence=True)
```

In [ ]:

```
modelo_if_ruins = IForest(contamination=contaminacao,random_state=0,).fit(df_input_ruin
s)
resultado_if_ruins = modelo_if_ruins.predict(df_input_ruins, return_confidence=True)
probs_if_ruins = modelo_if_ruins.predict_proba(df_input_ruins,method="unify", return_co
nfidence=True)
```

In [ ]:

```
df_resultados["if"] = resultado_if[0]
df_resultados["prob if"] = gera_prob_clssificacao(df_resultados,"if",probs_if)

df_resultados_boas["if"] = resultado_if_boas[0]
df_resultados_boas["prob if"] = gera_prob_clssificacao(df_resultados_boas,"if",probs_if
_boas)

df_resultados_ruins["if"] = resultado_if_ruins[0]
df_resultados_ruins["prob if"] = gera_prob_clssificacao(df_resultados_ruins,"if",probs_
if_ruins)

mask1 = df_resultados["if"]==1
mask2 = df_resultados["prob if"]>=0.9
mask = mask1 & mask2
df_resultados[mask]
```

Out[ ]:

	Unnamed: 0	danceability	energy	key	loudness	mode	speechiness	acousticness	ins
51	0	0.332	0.225	0	-8.697	1	0.0348	0.76700	
71	0	0.684	0.449	3	-9.738	1	0.6110	0.86900	
81	0	0.426	0.225	8	-12.422	1	0.0291	0.82300	
121	0	0.746	0.251	11	-16.169	0	0.2590	0.78200	
124	0	0.414	0.404	0	-9.928	0	0.0499	0.27100	
...	...	...	...	...	...	...	...	...	
5103	0	0.456	0.840	8	-3.637	0	0.2830	0.51900	
5107	0	0.586	0.740	1	-2.997	0	0.4040	0.55500	
5118	0	0.544	0.369	2	-9.514	1	0.0380	0.96900	
5143	0	0.412	0.881	2	-3.502	1	0.0870	0.00039	
5155	0	0.732	0.831	0	-3.791	0	0.0691	0.55400	

258 rows × 22 columns

time: 146 ms (started: 2022-06-21 16:43:19 +00:00)

Juntando resultados

In [ ]:

```
df_resultados["total votos"] = df_resultados["abod"] + df_resultados["autoencoder"] + d
f_resultados["if"] + df_resultados["lof"]
df_resultados["probabilidade"] = df_resultados["abod"]*df_resultados["prob abod"] + df_
resultados["autoencoder"]*df_resultados["prob autoencoder"] + df_resultados["if"]*df_re
sultados["prob if"] + df_resultados["lof"]*df_resultados["prob lof"]
df_resultados["probabilidade"] /= 4

df_resultados_boas["total votos"] = df_resultados_boas["abod"] + df_resultados_boas["au
toencoder"] + df_resultados_boas["if"] + df_resultados_boas["lof"]
df_resultados_boas["probabilidade"] = df_resultados_boas["abod"]*df_resultados_boas["pr
ob abod"] + df_resultados_boas["autoencoder"]*df_resultados_boas["prob autoencoder"] +
df_resultados_boas["if"]*df_resultados_boas["prob if"] + df_resultados_boas["lof"]*df_r
esultados_boas["prob lof"]
df_resultados_boas["probabilidade"] /= 4

df_resultados_ruins["total votos"] = df_resultados_ruins["abod"] + df_resultados_ruins[
"autoencoder"] + df_resultados_ruins["if"] + df_resultados_ruins["lof"]
df_resultados_ruins["probabilidade"] = df_resultados_ruins["abod"]*df_resultados_ruins[
"prob abod"] + df_resultados_ruins["autoencoder"]*df_resultados_ruins["prob autoencode
r"] + df_resultados_ruins["if"]*df_resultados_ruins["prob if"] + df_resultados_ruins["l
of"]*df_resultados_ruins["prob lof"]
df_resultados_ruins["probabilidade"] /= 4
#Exportando resultados
df_resultados.to_csv(path_top_global+"resultado_anomalias.csv")
df_resultados_boas.to_csv("/content/drive/My Drive/INF1032 - Spotify/Dados/Consolidado
s/resultado_anomalias_musicas_boas.csv")
df_resultados_ruins.to_csv("/content/drive/My Drive/INF1032 - Spotify/Dados/Consolidado
s/resultado_anomalias_musicas_ruins.csv")
```

time: 904 ms (started: 2022-06-21 16:43:19 +00:00)

Análise dos Resultados

In [ ]:

```
path_top_global = '/content/drive/My Drive/INF1032 - Spotify/Dados/Matheus/MusicasAnomalas/'
df_resultados = pd.read_csv(path_top_global+"resultado_anomalias.csv",index_col="Unnamed: 0")
df_resultados
```

Out[ ]:

	Unnamed: 0.1	danceability	energy	key	loudness	mode	speechiness	acousticness	ins
0	0	0.520	0.731	6	-5.338	0	0.0557	0.34200	
1	0	0.905	0.563	8	-6.135	1	0.1020	0.02540	
2	0	0.761	0.525	11	-6.900	1	0.0944	0.44000	
3	0	0.591	0.764	1	-5.484	1	0.0483	0.03830	
4	0	0.756	0.697	8	-6.377	1	0.0401	0.18200	
...	...	...	...	...	...	...	...	...	
5151	0	0.714	0.442	6	-5.909	1	0.0605	0.74200	
5152	0	0.601	0.713	4	-3.758	0	0.0449	0.02820	
5153	0	0.578	0.431	2	-7.034	1	0.0269	0.46900	
5154	0	0.340	0.684	11	-6.306	1	0.0440	0.00979	
5155	0	0.732	0.831	0	-3.791	0	0.0691	0.55400	

5156 rows × 24 columns

time: 1.76 s (started: 2022-06-26 23:49:12 +00:00)

Voto Majoritário

In [ ]:

```
mask = df_resultados["total votos"]==4

df_resultados[mask]
```

Out[ ]:

	Unnamed: 0.1	danceability	energy	key	loudness	mode	speechiness	acousticness	ins
369	0	0.418	0.1060	8	-22.507	0	0.0448	0.994000	
391	0	0.722	0.3310	8	-7.789	1	0.0726	0.337000	
641	0	0.588	0.4790	1	-7.039	1	0.2810	0.604000	
901	0	0.461	0.0279	0	-21.992	1	0.0412	0.973000	
911	0	0.570	0.2850	9	-14.125	0	0.0381	0.737000	
918	0	0.597	0.1130	0	-34.475	1	0.9540	0.461000	
1020	0	0.773	0.8590	11	-4.913	1	0.0747	0.085500	
1121	0	0.546	0.8900	7	-3.380	1	0.5130	0.078100	
1351	0	0.610	0.2580	0	-12.758	1	0.0331	0.883000	
1355	0	0.356	0.1430	9	-15.148	1	0.0388	0.976000	
1359	0	0.746	0.3380	11	-13.472	0	0.1090	0.674000	
1454	0	0.512	0.2800	8	-16.708	0	0.0441	0.811000	
1826	0	0.336	0.2310	1	-6.217	1	0.0497	0.942000	
1872	0	0.636	0.3350	11	-13.327	1	0.9660	0.993000	
1875	0	0.707	0.3140	6	-10.115	0	0.7470	0.977000	
2180	0	0.712	0.3740	0	-10.215	0	0.5110	0.795000	
2191	0	0.593	0.0668	11	-15.268	0	0.0722	0.949000	
2901	0	0.368	0.2860	9	-13.031	0	0.0294	0.913000	
3044	0	0.243	0.2600	4	-5.876	0	0.0305	0.828000	
3162	0	0.535	0.5210	0	-5.942	1	0.0272	0.000532	
3198	0	0.413	0.1300	0	-25.166	0	0.0336	0.900000	
3266	0	0.429	0.2310	5	-20.430	1	0.4020	0.878000	
3658	0	0.184	0.2970	2	-14.534	1	0.0359	0.473000	
3660	0	0.231	0.4570	6	-10.773	0	0.0318	0.012600	
3775	0	0.307	0.5280	11	-11.337	0	0.1080	0.972000	
3779	0	0.396	0.3730	9	-14.097	0	0.0989	0.991000	
4168	0	0.331	0.5130	11	-15.392	0	0.6320	0.972000	
4430	0	0.356	0.1760	8	-8.291	1	0.0391	0.843000	
4599	0	0.468	0.6520	0	-4.912	1	0.0722	0.333000	
4917	0	0.702	0.0474	3	-15.198	0	0.5260	0.005160	
4938	0	0.269	0.5910	10	-5.790	0	0.0416	0.453000	
5010	0	0.280	0.4780	9	-8.755	0	0.0370	0.674000	
5107	0	0.586	0.7400	1	-2.997	0	0.4040	0.555000	

33 rows × 24 columns

```
time: 155 ms (started: 2022-06-22 15:43:23 +00:00)
```

Probabilidade de ser anomalia

In [ ]:

```
prob = 0.7
mask1 = df_resultados["total votos"]==4
mask2 = df_resultados["probabilidade"]>=prob
mask = (mask1 & mask2)
print("Qtd musicas anômalas (%.2f%% prob) = %d"%(100*prob,mask.sum()))
df_resultados[mask]
```

```
Qtd musicas anômalas (70.00% prob) = 25
```

Out[ ]:

	Unnamed: 0.1	danceability	energy	key	loudness	mode	speechiness	acousticness	ins
369	0	0.418	0.1060	8	-22.507	0	0.0448	0.99400	
641	0	0.588	0.4790	1	-7.039	1	0.2810	0.60400	
901	0	0.461	0.0279	0	-21.992	1	0.0412	0.97300	
918	0	0.597	0.1130	0	-34.475	1	0.9540	0.46100	
1355	0	0.356	0.1430	9	-15.148	1	0.0388	0.97600	
1359	0	0.746	0.3380	11	-13.472	0	0.1090	0.67400	
1826	0	0.336	0.2310	1	-6.217	1	0.0497	0.94200	
1872	0	0.636	0.3350	11	-13.327	1	0.9660	0.99300	
1875	0	0.707	0.3140	6	-10.115	0	0.7470	0.97700	
2191	0	0.593	0.0668	11	-15.268	0	0.0722	0.94900	
2901	0	0.368	0.2860	9	-13.031	0	0.0294	0.91300	
3044	0	0.243	0.2600	4	-5.876	0	0.0305	0.82800	
3198	0	0.413	0.1300	0	-25.166	0	0.0336	0.90000	
3266	0	0.429	0.2310	5	-20.430	1	0.4020	0.87800	
3658	0	0.184	0.2970	2	-14.534	1	0.0359	0.47300	
3660	0	0.231	0.4570	6	-10.773	0	0.0318	0.01260	
3775	0	0.307	0.5280	11	-11.337	0	0.1080	0.97200	
3779	0	0.396	0.3730	9	-14.097	0	0.0989	0.99100	
4168	0	0.331	0.5130	11	-15.392	0	0.6320	0.97200	
4430	0	0.356	0.1760	8	-8.291	1	0.0391	0.84300	
4599	0	0.468	0.6520	0	-4.912	1	0.0722	0.33300	
4917	0	0.702	0.0474	3	-15.198	0	0.5260	0.00516	
4938	0	0.269	0.5910	10	-5.790	0	0.0416	0.45300	
5010	0	0.280	0.4780	9	-8.755	0	0.0370	0.67400	
5107	0	0.586	0.7400	1	-2.997	0	0.4040	0.55500	

25 rows × 24 columns

```
time: 47.3 ms (started: 2022-06-22 15:43:33 +00:00)
```

In [ ]:

```
df_resultados[mask].nome
```

Out[ ]:

```
369                                    Carol of the Bells
641                                                  FEAR.
901                              Dead Inside (Interlude)
918                                     The Explanation
1355                             before I close my eyes
1359                          love yourself (interlude)
1826                         raindrops (an angel cried)
1872                                         Paul - Skit
1875                               Em Calls Paul - Skit
2191                            difference (interlude)
2901                                      Venice Bitch
3044                                     Jesus Is Lord
3198                                          JACKBOYS
3266                                 Alfred - Interlude
3658                                      Chromatica II
3660                                       Chromatica I
3775                                    Anxiety - Intro
3779                Juice WRLD Speaks From Heaven - Outro
4168                                    Beautiful Trip
4430                                       To Be Loved
4599    SWEET / I THOUGHT YOU WANTED TO DANCE (feat. B...
4917                                       Donda Chant
4938                                    Love Is A Game
5010                                            Dawn FM
5107              Residente: Bzrp Music Sessions, Vol. 49
Name: nome, dtype: object

time: 10.7 ms (started: 2022-06-22 15:43:33 +00:00)
```

In [ ]:

```
df_resultados[mask][['danceability', 'energy', 'key', 'loudness', 'mode',
       'speechiness', 'acousticness', 'instrumentalness', 'liveness',
       'valence', 'tempo', 'duration_ms', 'nome']]
```

Out[ ]:

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalne
369	0.418	0.1060	8	-22.507	0	0.0448	0.99400	0.0292
641	0.588	0.4790	1	-7.039	1	0.2810	0.60400	0.0000
901	0.461	0.0279	0	-21.992	1	0.0412	0.97300	0.0022
918	0.597	0.1130	0	-34.475	1	0.9540	0.46100	0.0003
1355	0.356	0.1430	9	-15.148	1	0.0388	0.97600	0.0000
1359	0.746	0.3380	11	-13.472	0	0.1090	0.67400	0.0003
1826	0.336	0.2310	1	-6.217	1	0.0497	0.94200	0.0000
1872	0.636	0.3350	11	-13.327	1	0.9660	0.99300	0.0000
1875	0.707	0.3140	6	-10.115	0	0.7470	0.97700	0.0000
2191	0.593	0.0668	11	-15.268	0	0.0722	0.94900	0.0000
2901	0.368	0.2860	9	-13.031	0	0.0294	0.91300	0.1770
3044	0.243	0.2600	4	-5.876	0	0.0305	0.82800	0.0000
3198	0.413	0.1300	0	-25.166	0	0.0336	0.90000	0.8200
3266	0.429	0.2310	5	-20.430	1	0.4020	0.87800	0.0000
3658	0.184	0.2970	2	-14.534	1	0.0359	0.47300	0.8930
3660	0.231	0.4570	6	-10.773	0	0.0318	0.01260	0.8750
3775	0.307	0.5280	11	-11.337	0	0.1080	0.97200	0.0005
3779	0.396	0.3730	9	-14.097	0	0.0989	0.99100	0.0978
4168	0.331	0.5130	11	-15.392	0	0.6320	0.97200	0.9530
4430	0.356	0.1760	8	-8.291	1	0.0391	0.84300	0.0000
4599	0.468	0.6520	0	-4.912	1	0.0722	0.33300	0.0003

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalne
4917	0.702	0.0474	3	-15.198	0	0.5260	0.00516	0.0000
4938	0.269	0.5910	10	-5.790	0	0.0416	0.45300	0.0006
5010	0.280	0.4780	9	-8.755	0	0.0370	0.67400	0.0000
5107	0.586	0.7400	1	-2.997	0	0.4040	0.55500	0.0000

time: 50.7 ms (started: 2022-06-22 15:49:17 +00:00)

In [ ]:

```
df_resultados[['danceability', 'energy', 'key', 'loudness', 'mode',
       'speechiness', 'acousticness', 'instrumentalness', 'liveness',
       'valence', 'tempo', 'duration_ms', 'nome']].describe()
```

Out[ ]:

	danceability	energy	key	loudness	mode	speechiness	acou
count	5156.000000	5156.000000	5156.000000	5156.000000	5156.000000	5156.000000	5156
mean	0.684269	0.635878	5.227308	-6.309993	0.576804	0.127005	(
std	0.141697	0.163691	3.648195	2.489318	0.494114	0.116225	(
min	0.150000	0.027900	0.000000	-34.475000	0.000000	0.023200	(
25%	0.598000	0.535000	1.000000	-7.458500	0.000000	0.045000	(
50%	0.701000	0.650000	5.000000	-5.954000	1.000000	0.076800	(
75%	0.786000	0.757000	8.000000	-4.671500	1.000000	0.173000	(
max	0.985000	0.989000	11.000000	1.509000	1.000000	0.966000	(

time: 175 ms (started: 2022-06-22 15:50:25 +00:00)