Componentes da equipe

- Maria Eduarda Bonnel - RA00318891
- Vinícius Ferreira - RA00319760
- Vitória Teixeira - RA00320578

Spotify Hits

Tema e objetivos do projeto

- Área de interesse: música e entretenimento.
- **OBJETIVO DO NEGÓCIO**: Produzir músicas de sucesso.
- **OBJETIVO DO PROJETO**: Prever a popularidade de uma música a partir de suas características como dançabilidade, energia, acústica, ao vivo, instrumentalidade, etc.

Referências.

- Documentação oficial da API do Spotify
- Documentação oficial do Spotipy, uma biblioteca python para acessar a API
  - Tutorial de primeiros passos com o Spotipy

Primeiros passos (ambiente linux)

1. Crie um ambiente virtual. Na pasta atual, rode no terminal: `make venv`
2. Ative o ambiente virtual: `source spot_env/bin/activate`
3. Instale as dependências: `pip install -r requirements.txt`
4. Crie uma pasta com as variáveis de ambiente:
   - `touch .env`

- Configure as variáveis `SPOTIPY_CLIENT_ID` e `SPOTIPY_CLIENT_SECRET` com os valores disponíveis no app criado no site do spotify para desenvolvedores. Não as compartilhe nem use-as diretamente em código público.

In [ ]:
```
# carrega as variáveis de ambiente
from dotenv import load_dotenv
load_dotenv()

import os
import pandas as pd
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
import numpy as np
```

Extração dos dados com API

Credenciais da API

In [ ]:
```
client_credentials_manager = SpotifyClientCredentials(
    client_id=os.environ.get('SPOTIPY_CLIENT_ID'),
    client_secret=os.environ.get('SPOTIPY_CLIENT_SECRET')
)
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)
```

In [ ]:
```
playlists = {
    'Global': '37i9dQZEVXbNG2KDcFcKOF',
    'Brazil': '37i9dQZEVXbKzoK95AbRy9',
    'Australia': '37i9dQZEVXbK4fwx2r07XW',
    'Austria': '37i9dQZEVXbM1EaZ0igDlz',
    'Argentina': '37i9dQZEVXbKPTKrnFPD0G',
    'Chile': '37i9dQZEVXbLJ0paT1JkgZ',
    'Colombia': '37i9dQZEVXbL1Fl8vdBUba',
    'Mexico': '37i9dQZEVXbKUoIkUXteF6',
    'USA': '37i9dQZEVXbLp5XoPON0wI'
}
```

In [ ]:
```
df = pd.read_csv('../top_songs.csv')
set(df['playlist_id'])
```

Out[ ]:
```
{'37i9dQZEVXbK4fwx2r07XW',
 '37i9dQZEVXbKPTKrnFPD0G',
 '37i9dQZEVXbKUoIkUXteF6',
 '37i9dQZEVXbKzoK95AbRy9',
 '37i9dQZEVXbLp5XoPON0wI',
 '37i9dQZEVXbM1EaZ0igDlz',
 '37i9dQZEVXbNG2KDcFcKOF'}
```

Criando o CSV das playlists

In [ ]:
```
'''
dict: dicionário com o país e o id da sua respectiva playlist.
df: dataframe, caso pré-existente.
refresh: refaz as requisições para todas as playlists. False por padrão.
'''
def playlists_into_df(playlists_dict: dict, df: pd.DataFrame, refresh: bool = False) -> pd.DataFrame:
    country_list = []
    playlist_id_list = []
    artist_name = []
    track_name = []
    release_date = []
    popularity = []
    track_id = []
    for country, playlist_id in playlists_dict.items():
        # Pula requisição se ela já foi feita
        if (not refresh) and (playlist_id in set(df['playlist_id'])):
            continue
        # Faz uma requisição na API
        playlist = sp.playlist_items(
            playlist_id=playlist_id,
            fields='items.track(album.release_date, artists.name, name, popularity, id)'
        )
        tracks = playlist['items']
        for track in tracks:
            track = track['track']
            country_list.append(country)
            playlist_id_list.append(playlist_id)
            artist_name.append(track['artists'][0]['name'])
            track_name.append(track['name'])
            release_date.append(track['album']['release_date'])
            popularity.append(track['popularity'])
```

```
            track_id.append(track['id'])
        print(f'Added playlist from {country} with id {playlist_id}')
    new_df = pd.DataFrame({
        'playlist_country': country_list,
        'artist_name': artist_name,
        'track_name': track_name,
        'release_date': release_date,
        'popularity': popularity,
        'track_id': track_id,
        'playlist_id': playlist_id_list
    })
    if refresh:
        return new_df
    return pd.concat([df, new_df], ignore_index=True)
```

In [ ]:
```
top_songs = playlists_into_df(playlists, df, False)
```

```
Added playlist from Chile with id 37i9dQZEVXbLJ0paT1JkgZ
Added playlist from Colombia with id 37i9dQZEVXbL1Fl8vdBUba
```

In [ ]:
```
top_songs
```

Out[ ]:

	playlist_country	artist_name	track_name	release_date	popularity	track_id	pl
0	Global	Artemas	i like the way you kiss me	2024-03-19	97	2GxrNKugF82CnoRFbQfzPf	37i9dQZEVXbNG2K
1	Global	Hozier	Too Sweet	2024-03-22	86	0AjmK0Eai4zGrLaJwPvrDp	37i9dQZEVXbNG2K
2	Global	Benson Boone	Beautiful Things	2024-04-05	82	3xkHsmpQCBMytMJNiDf3li	37i9dQZEVXbNG2K
3	Global	Djo	End of Beginning	2022-09-16	99	3qhlB30KknSejmIvZZLjOD	37i9dQZEVXbNG2K
4	Global	Ariana Grande	we can't be friends (wait for your love)	2024-03-08	95	51ZQ1vr10ffzbwIjDCwqm4	37i9dQZEVXbNG2K
...	...	...	...	...	...	...	
445	Colombia	Tony Dize	Solos	2009-11-17	59	0WKd91LoIHCFIhDmgewjhy	37i9dQZEVXbL1FI
446	Colombia	Dei V	Narcotics (with Bryant Myers)	2023-09-14	81	0p0cOpBujR114Wirv5AM7W	37i9dQZEVXbL1FI
447	Colombia	De La Ghetto	Sensacion Del Bloque	2006-01-01	78	5clFSlfkCRlhnH1cAQjSBi	37i9dQZEVXbL1FI
448	Colombia	The Academy: Segunda Misión	QUÍTENME EL TELÉFONO (feat. Yandel, Jay Wheeler)	2024-03-28	75	41XmmKJHx1ZAH0lykgjxfx	37i9dQZEVXbL1FI
449	Colombia	Myke Towers	LALA	2023-03-23	87	7ABLbnD53cQK00mhcaOUVG	37i9dQZEVXbL1FI

450 rows × 7 columns

In [ ]:
```
# Salva o df em um csv
top_songs.to_csv('../top_songs.csv', index=False)
```

Criando o CSV com as features de cada música

Como a API só suporta requisições de 50 músicas, dividimo-as em batches.

In [ ]:
```
def save_track_features_to_df(songs_df: pd.DataFrame) -> pd.DataFrame:
    df_list = []
    # Separa o df com todas as músicas em batches de 50 músicas
    batches = np.array_split(songs_df, len(songs_df) // 50)
    for batch in batches:
        track_features_list = sp.audio_features(tracks=batch['track_id']) # retorna uma lista de dicionário
        for track_dict in track_features_list:
            df_list.append(pd.DataFrame([track_dict]))
    return pd.concat(df_list, ignore_index=True)

df = save_track_features_to_df(songs_df=pd.read_csv('top_songs.csv'))
df.to_csv('top_songs_features.csv', index=False)
```

```
/home/vinifm/spotify_hits_2/spot_venv/lib/python3.11/site-packages/numpy/core/fromnumeric.py:59: FutureWarn
ing: 'DataFrame.swapaxes' is deprecated and will be removed in a future version. Please use 'DataFrame.tran
spose' instead.
  return bound(*args, **kwds)
```

In [ ]:
```
playlist_df = pd.read_csv('../top_songs.csv')
playlist_df.head(10)
```

Out[ ]:

	playlist_country	artist_name	track_name	release_date	popularity	track_id	playli
0	Global	Artemas	i like the way you kiss me	2024-03-19	97	2GxrNKugF82CnoRFbQfzPf	37i9dQZEVXbNG2KDcF
1	Global	Hozier	Too Sweet	2024-03-22	86	0AjmK0Eai4zGrLaJwPvrDp	37i9dQZEVXbNG2KDcF
2	Global	Benson Boone	Beautiful Things	2024-04-05	82	3xkHsmpQCBMytMJNiDf3li	37i9dQZEVXbNG2KDcF
3	Global	Djo	End of Beginning	2022-09-16	99	3qhlB30KknSejmIvZZLjOD	37i9dQZEVXbNG2KDcF
4	Global	Ariana Grande	we can't be friends (wait for your love)	2024-03-08	95	51ZQ1vr10ffzbwIjDCwqm4	37i9dQZEVXbNG2KDcF
5	Global	FloyyMenor	Gata Only	2024-02-02	96	6XjDF6nds4DE2BBbagZol6	37i9dQZEVXbNG2KDcF
6	Global	Future	Like That	2024-03-22	96	2tudvzsrR56uom6smgOcSf	37i9dQZEVXbNG2KDcF
7	Global	Tate McRae	greedy	2023-09-15	97	3rUGC1vUpkDG9CZFHMur1t	37i9dQZEVXbNG2KDcF
8	Global	Teddy Swims	Lose Control	2023-09-15	93	17phhZDn6oGtzMe56NuWvj	37i9dQZEVXbNG2KDcF
9	Global	Beyoncé	TEXAS HOLD 'EM	2024-03-29	87	7wLShogStyDeZvL0a6daN5	37i9dQZEVXbNG2KDcF

In [ ]:
```
features_df = pd.read_csv('../top_songs_features.csv')
features_df.head(10)
```

Out[ ]:

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo
0	0.599	0.946	11	-4.263	1	0.0447	0.000938	0.010600	0.0826	0.747	151.647
1	0.741	0.620	10	-5.505	1	0.0412	0.029500	0.000809	0.0398	0.934	117.038
2	0.472	0.471	10	-5.692	1	0.0603	0.151000	0.000000	0.1400	0.219	105.029
3	0.689	0.454	2	-7.643	1	0.0584	0.035100	0.002590	0.0707	0.912	159.982
4	0.645	0.663	5	-8.305	1	0.0447	0.065700	0.000032	0.0751	0.287	115.830
5	0.791	0.499	8	-8.472	0	0.0509	0.446000	0.000024	0.0899	0.669	99.986
6	0.814	0.676	11	-4.670	0	0.2310	0.007090	0.000013	0.1190	0.312	162.012
7	0.750	0.733	6	-3.180	0	0.0319	0.256000	0.000000	0.1140	0.844	111.018

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo
8	0.561	0.604	9	-4.409	1	0.0337	0.199000	0.000019	0.1040	0.242	159.920
9	0.727	0.711	2	-6.549	1	0.0780	0.582000	0.000000	0.1580	0.375	110.012

Criando do banco de dados

1. Instalar sqlite:
    - a. `sudo apt update`
    - b. `sudo apt install sqlite3`

In [ ]:
```
import sqlite3
import csv
```

In [ ]:
```
def create_db_from_csv(csv_files, db_name):
    # Conecta com o banco de dados. Se inexistente, cria-o
    conn = sqlite3.connect(db_name)
    cur = conn.cursor()

    # Iteração nos CSVs
    for csv_file in csv_files:
        table_name = csv_file.split('.')[0]  # extrai o nome do arquivo
        with open(csv_file, 'r', newline='', encoding='utf-8') as f:
            reader = csv.reader(f) # transforma cada linha em uma lista
            headers = next(reader)  # Retorna a 1a linha e itera para a 2a
            column_names = ', '.join(headers)
            placeholders = ', '.join('?' * len(headers))
            cur.execute(f'CREATE TABLE IF NOT EXISTS {table_name} ({column_names})')
            cur.executemany(f'INSERT INTO {table_name} VALUES ({placeholders})', reader)

    conn.commit()
    conn.close()

# Example usage:
csv_files = ['top_songs.csv', 'top_songs_features.csv']
db_name = 'database.db'

create_db_from_csv(csv_files, db_name)
```

In [ ]:
```
conn = sqlite3.connect('../database.db')
cur = conn.cursor()
```

```
cur.execute('PRAGMA table_info(top_songs)')
columns = cur.fetchall()
print("Colunas da tabela top_songs:")
for column_name in columns:
    print(column_name[1])

cur.execute('PRAGMA table_info(top_songs_features)')
columns = cur.fetchall()
print("\nColunas da tabela top_songs_features:")
for column_name in columns:
    print(column_name[1])
conn.close()
```

```
Colunas da tabela top_songs:
playlist_country
artist_name
track_name
release_date
popularity
track_id
playlist_id

Colunas da tabela top_songs_features:
danceability
energy
key
loudness
mode
speechiness
acousticness
instrumentalness
liveness
valence
tempo
type
id
uri
track_href
analysis_url
duration_ms
time_signature
```

Descrição de cada audio feature:

- Acousticness (acusticidade)
  - feature: number [float]
  - Medida de confiança de 0,0 a1.0
  - 1,0 representa alta confiança de que a faixa é acústica
- Analysis_url (análise_url)
  - feature: string
  - É a URL para acessar a análise completa de áudio da faixa.
- Danceability (dançabilidade)
  - feature: number [float]
  - Descreve o quão adequada uma faixa é para dançar com base em um combinação.
  - 0,0 significa menos dançante.
  - 1,0 significa mais dançante.
- Duration_ms (duração_ms)
  - feature : integer
  - Representa a duração da faixa em milissegundos.
- Energy (energia)
  - feature: number [float]
  - É uma medida de 0,0 a 1,0 e representa uma medida perceptiva de intensidade e atividade.
- Id (id)
  - feature : string
  - É o ID do spotify para a faixa.
- Instrumentalness (instrumentalidade)
  - feature : number [float]
  - Prevê se uma faixa não contém vocais.
  - 1,0 : valores proximos significa maior probabilidade da faixa não conter conteúdo vocal.

    - 0,5 : valores acinma pretendem representar faixas instrumentais.
- Key (chave)
    - feature : integer
    - Representa a tonalidade em que a faixa está.
    - Os números inteiros são mapeados para as notas usando a notação padrão de classe de nota
    - https://en.wikipedia.org/wiki/Pitch_class
    - Se nenhuma chave foir detectada, o valor será: -1
- Liveness (vivacidade)
    - feature : numer [float]
    - Detecta a presença de público na gravação.
    - Valores altos representam uma maior probabilidade que a faixa tenha sido tocada ao vivo.
    - 0,8 : tal valor representa uma forte probabilidade de que a pista esteja ativa.
- Loudness (intensidade)
    - feature : number [float]
    - Representa o volume geral de uma faixa em decibéis (dB)
    - Os valorem normalmente variam entre -60 e 0 dB.
    - Os valores são calculados em média em toda faixa e são uteo para comparar a intensidade relativa das faixas.
- Mode (modo)
    - feature : integger
    - Indica a modalidade (maior ou menos) de uma faixa, o tipo de escala da qual deriva seu conteúdo melódico.
    - Maior é representado por 1 e menor é 0.
- Speechiness (fala)
    - feature : number [float]
    - Detecta a presença de palavras faladas em uma faixa
    - Quanto mais falada for a gravação, mais próximo de 1,0 será o valor atribuido.
    - Valores acima de 0,66 descrevem faixas que provavelmente são composta inteiramente de palavras faladas.
    - Valores entre 0,33 e 0,66 descrevem faixas que podem contem música e fala
    - Valores abaixo de 0,33 provavelmente representam música e outras faixas não faladas.
- Tempo

    - feature: number [float]
    - Represrnta o andamento geral estimado de uma faixa em batidas por minutos (BPM)
- Time_signature (assinatura_hora)
    - feature: integger
    - Uma formula de compasso estimada.
    - A fórmula de compasso varia de 3 a 7 indicando fórmulas de compasso de “3/4” a “7/4”
- Track_href
    - feature : string
    - Um link para endpoint da API Web fornecendo detalhes completos da faixa.
- Type (tipo)
    - feature : string
    - Representa o tipo de objeto.
    - Valores permitidos: `"audio_features"`
- URI
    - feature : string
    - O URI do Spotify para a faixa
- Valence (valência)
    - feature: number [float]
    - Representa uma medida de 0,0 a 1,0 que descreve a positivdade musical transmitida por uma faixa.
    - Faixas com valência alta soam mais positivas (por exemplo, feliz, alegre, eufórica).
    - faixas com valência baixa soam mais negativas (por exemplo, triste, deprimida, irritada).

Criação da API com flask

In [ ]:
```
from flask import Flask, jsonify, url_for
from flask import request
import sqlite3
import os

def execute_query(query):
    conn = sqlite3.connect('database.db')
    cur = conn.cursor()
    cur.execute(query)
    results = cur.fetchall()
    conn.close()
    return results

app = Flask(__name__)

# Lista as urls disponíveis
# ex.: http://127.0.0.1:5000/
@app.route("/")
def index():
    links = []
    for rule in app.url_map.iter_rules():
        # Filter out rules we can't navigate to in a browser
        # and rules that require parameters
        if "GET" in rule.methods and not rule.endpoint.startswith('static'):
            url = url_for(rule.endpoint)
            links.append((url, rule.endpoint))
    return '\n'.join([f'<div><a href="{url}">{endpoint}</a></div>' for url, endpoint in links])

# Retorna todas os países no banco de dados
# ex.: http://127.0.0.1:5000/all_playlists
@app.route("/all_playlists",methods = ['GET'])
def all_playlists():
    return jsonify(execute_query("SELECT DISTINCT playlist_country FROM top_songs"))

# Retorna todas as músicas de um país
# ex.: http://127.0.0.1:5000/playlist?country=Brazil
```

```python
@app.route("/playlist",methods = ['GET'])
def country():
    country = request.args.get('country')
    return jsonify(execute_query(f"SELECT * FROM top_songs WHERE playlist_country='{country}'"))

# Retorna as músicas mais populares
# ex.: http://127.0.0.1:5000/popular
@app.route("/popular",methods = ['GET'])
def popular():
    return jsonify(execute_query(f"SELECT * FROM top_songs ORDER BY popularity DESC"))

# Retorna as músicas menos populares
# ex.: http://127.0.0.1:5000/unpopular
@app.route("/unpopular",methods = ['GET'])
def unpopular():
    return jsonify(execute_query(f"SELECT * FROM top_songs ORDER BY popularity ASC"))

# Retorna as músicas mais antigas
# ex.: http://127.0.0.1:5000/old_songs
@app.route("/old_songs",methods = ['GET'])
def old_songs():
    return jsonify(execute_query(f"SELECT * FROM top_songs ORDER BY release_date ASC"))

# Retorna as músicas mais recentes
# ex.: http://127.0.0.1:5000/new_songs
@app.route("/new_songs",methods = ['GET'])
def new_songs():
    return jsonify(execute_query(f"SELECT * FROM top_songs ORDER BY release_date DESC"))

# Retorna os artistas que mais aparecem
# ex.: http://127.0.0.1:5000/frequent_artists
@app.route("/frequent_artists",methods = ['GET'])
def frequent_artists():
    return jsonify(execute_query(f"SELECT artist_name, COUNT(artist_name) AS frequency \
                                 FROM top_songs \
                                 GROUP BY artist_name \
                                 HAVING frequency > 1 \
                                 ORDER BY frequency DESC"))

if __name__=="__main__":
    app.run(
        port=5000,
```

```
        debug=True)
```

Treinamento do modelo

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

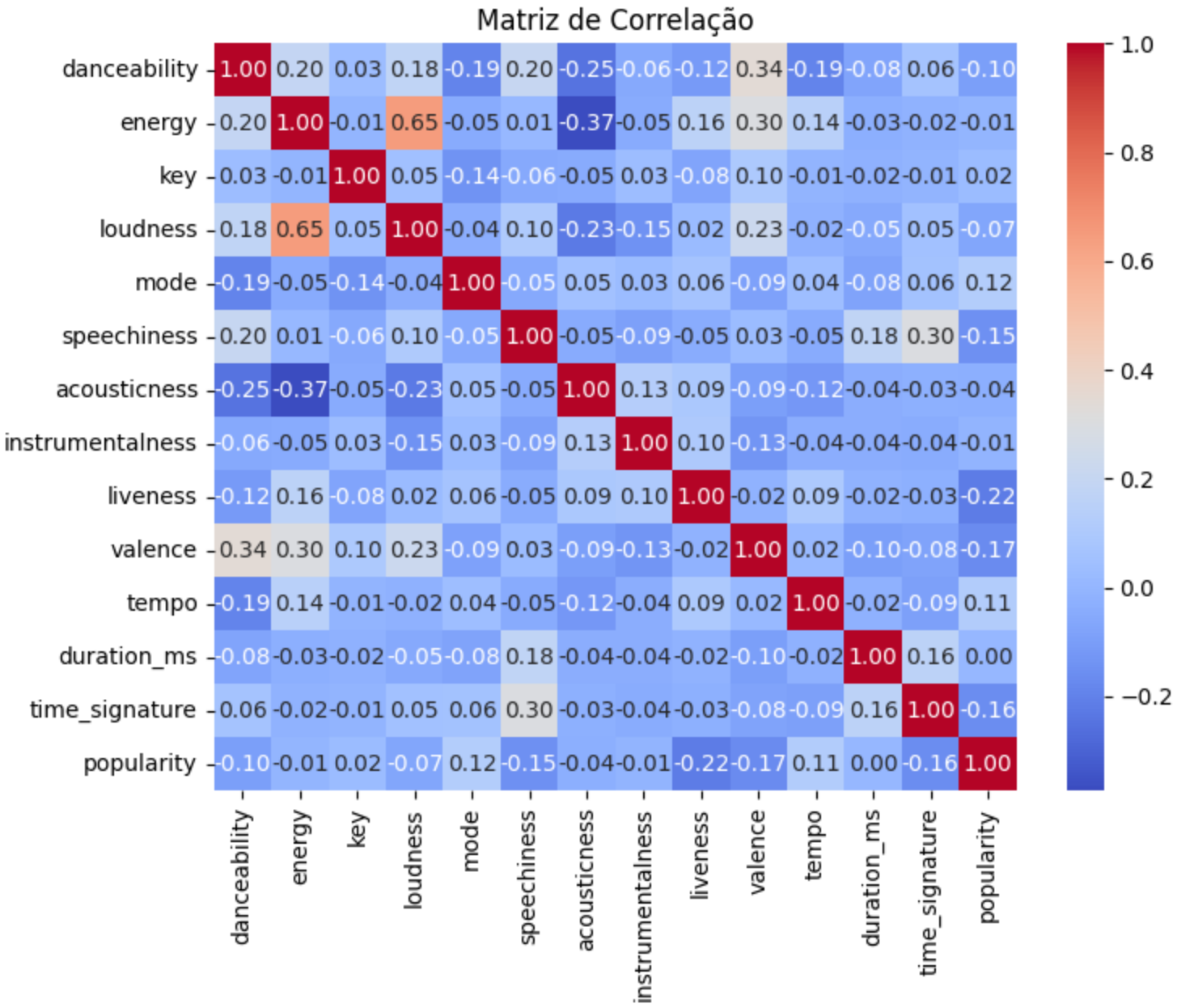
In [ ]:
```
df = pd.read_csv('top_songs_features.csv')
top_songs = pd.read_csv('top_songs.csv')

# mantém somente colunas numéricas
for column in df.select_dtypes(exclude=['number']).columns:
    df = df.drop(columns=[column])
df = pd.concat([df, top_songs['popularity']], axis=1)
df.dtypes
```

Out[ ]:
```
danceability        float64
energy              float64
key                   int64
loudness            float64
mode                  int64
speechiness         float64
acousticness        float64
instrumentalness    float64
liveness            float64
valence             float64
tempo               float64
duration_ms           int64
time_signature        int64
popularity            int64
dtype: object
```

Matriz de correlação

In [ ]:
```
# Calcular a matriz de correlação
matriz_correlacao = df.corr()
```

```
# Visualizar a matriz de correlação como um mapa de calor
plt.figure(figsize=(8, 6))
sns.heatmap(matriz_correlacao, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Matriz de Correlação')
plt.show()
```



01. Random forest

O Random Forest é composto por um conjunto de árvores de decisão, onde cada árvore é construída de forma independente usando uma amostra aleatória do conjunto de dados de treinamento e um subconjunto aleatório das características. Isso introduz aleatoriedade e diversidade no processo de treinamento.

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

x = df.drop(columns='popularity')
y = df['popularity']

# Dividir o conjunto de dados em conjunto de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# Criar o modelo de Random Forest
# Você pode ajustar os hiperparâmetros conforme necessário (por exemplo, n_estimators, max_depth, etc.)
rf_model = RandomForestRegressor(n_estimators=100,
                                 random_state=42,
                                 bootstrap=True,
                                 criterion='squared_error')

# Treinar o modelo
rf_model.fit(X_train, y_train)

# Fazer previsões no conjunto de teste
predictions = rf_model.predict(X_test)
```

Métricas

- Mean Squared Error (MSE): O MSE é a média dos quadrados das diferenças entre as previsões do modelo e os valores reais. Ele fornece uma medida da qualidade geral do modelo, onde valores menores indicam um melhor ajuste aos dados.
- Mean Absolute Error (MAE): O MAE é a média das diferenças absolutas entre as previsões do modelo e os valores reais. Ele mede a magnitude média dos erros do modelo, sem considerar sua direção.
- R^2 (R-squared): O R^2 é uma medida da proporção da variância nos valores de resposta que é explicada pelo modelo. Ele varia de 0 a 1, onde valores mais próximos de 1 indicam um melhor ajuste do modelo aos dados.
- Root Mean Squared Error (RMSE): O RMSE é a raiz quadrada do MSE e fornece uma interpretação na mesma unidade dos valores de destino. É uma medida comum de erro que penaliza mais fortemente grandes erros.

In [ ]:
```
# Avaliar o desempenho do modelo usando as métricas
mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)
r2 = r2_score(y_test, predictions)
rmse = np.sqrt(mse)

# Imprimir as métricas com 4 números após a vírgula usando f-strings
print("Métricas de Avaliação:")
print(f" - Mean Squared Error (MSE): {mse:.4f}")
print(f" - Mean Absolute Error (MAE): {mae:.4f}")
print(f" - R² (R-squared): {r2:.4f}")
print(f" - Root Mean Squared Error (RMSE): {rmse:.4f}")
```

```
Métricas de Avaliação:
 - Mean Squared Error (MSE): 22.1355
 - Mean Absolute Error (MAE): 3.1534
 - R² (R-squared): 0.6678
 - Root Mean Squared Error (RMSE): 4.7048
```

02. Naive Bayes (Gaussian)

Naive Bayes é uma técnica de classificação estatística baseada no Teorema de Bayes. É um dos algoritmos de aprendizagem supervisionada.

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np
import pandas as pd


x = df.drop(columns='popularity')
y = df['popularity']

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

nb_model = GaussianNB()

nb_model.fit(X_train, y_train)
```

```
predictions = nb_model.predict(X_test)

mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)
r2 = r2_score(y_test, predictions)
rmse = np.sqrt(mse)

print("Métricas de Avaliação:")
print(f" - Mean Squared Error (MSE): {mse:.4f}")
print(f" - Mean Absolute Error (MAE): {mae:.4f}")
print(f" - R² (R-squared): {r2:.4f}")
print(f" - Root Mean Squared Error (RMSE): {rmse:.4f}")
```

```
Métricas de Avaliação:
 - Mean Squared Error (MSE): 116.9667
 - Mean Absolute Error (MAE): 8.7000
 - R² (R-squared): -0.7551
 - Root Mean Squared Error (RMSE): 10.8151
```

03. KNN (K-nearest neighbor)

KNN é um algoritmo de aprendizado supervisionado que se baseia na proximidade dos exemplos de treinamento para tomar decisões de classificação ou regressão.

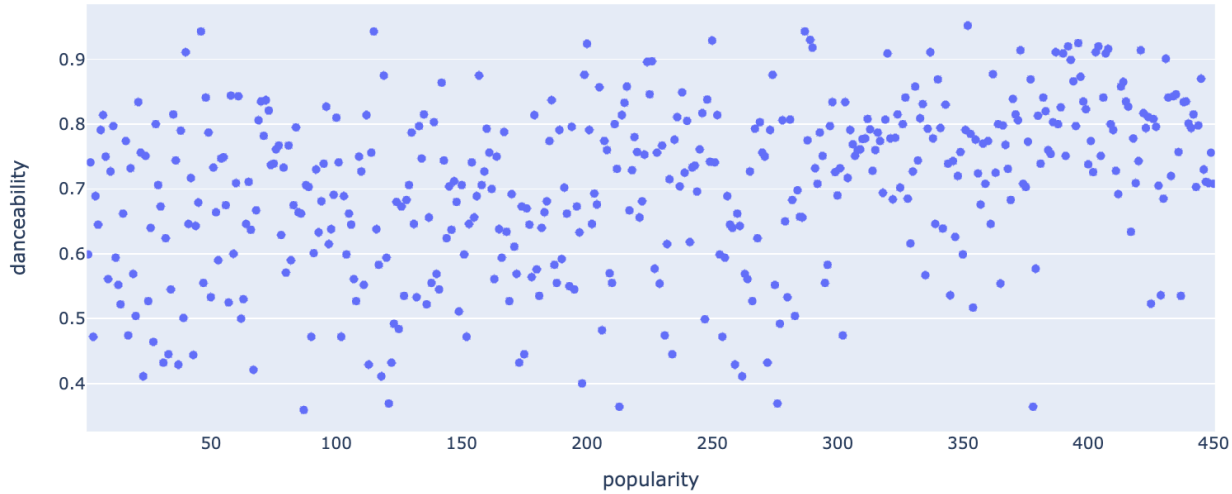
In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

x = df.drop(columns='popularity')
y = df['popularity']

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

k = 5
knn_model = KNeighborsRegressor(n_neighbors=k)

knn_model.fit(X_train, y_train)

predictions = knn_model.predict(X_test)
```

```
mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)
r2 = r2_score(y_test, predictions)
rmse = np.sqrt(mse)

print("Métricas de Avaliação:")
print(f" - Mean Squared Error (MSE): {mse:.4f}")
print(f" - Mean Absolute Error (MAE): {mae:.4f}")
print(f" - R² (R-squared): {r2:.4f}")
print(f" - Root Mean Squared Error (RMSE): {rmse:.4f}")
```

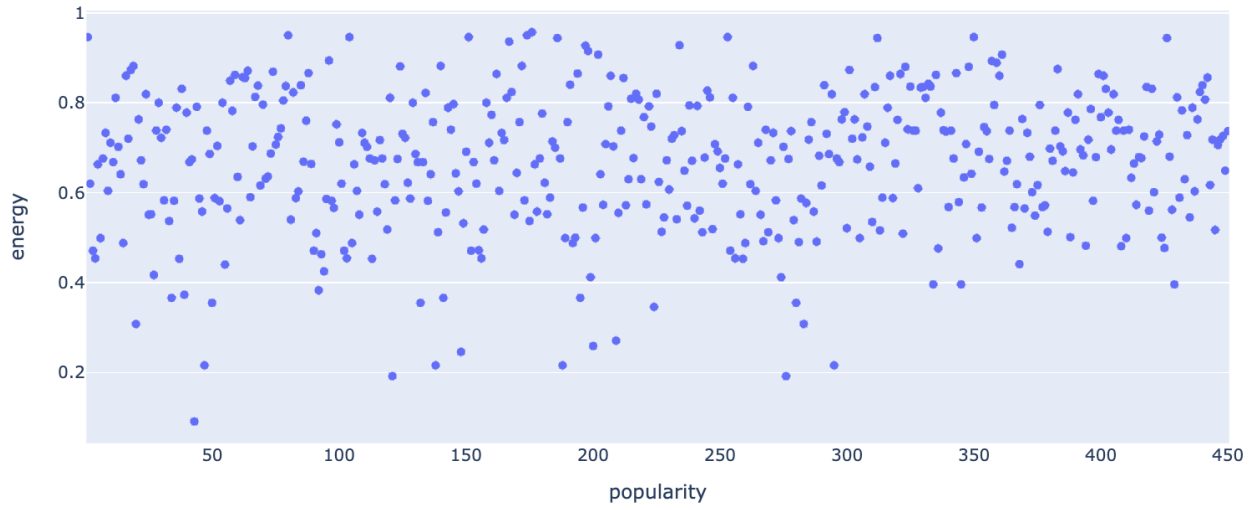
```
Métricas de Avaliação:
 - Mean Squared Error (MSE): 62.3538
 - Mean Absolute Error (MAE): 6.3378
 - R² (R-squared): 0.0643
 - Root Mean Squared Error (RMSE): 7.8964
```

Análise Exploratória

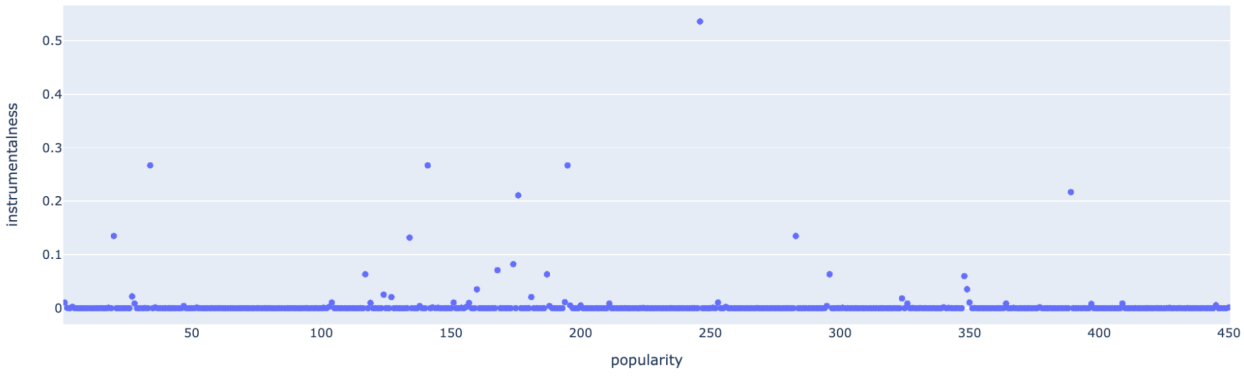
Dançabilidade X Popularidade



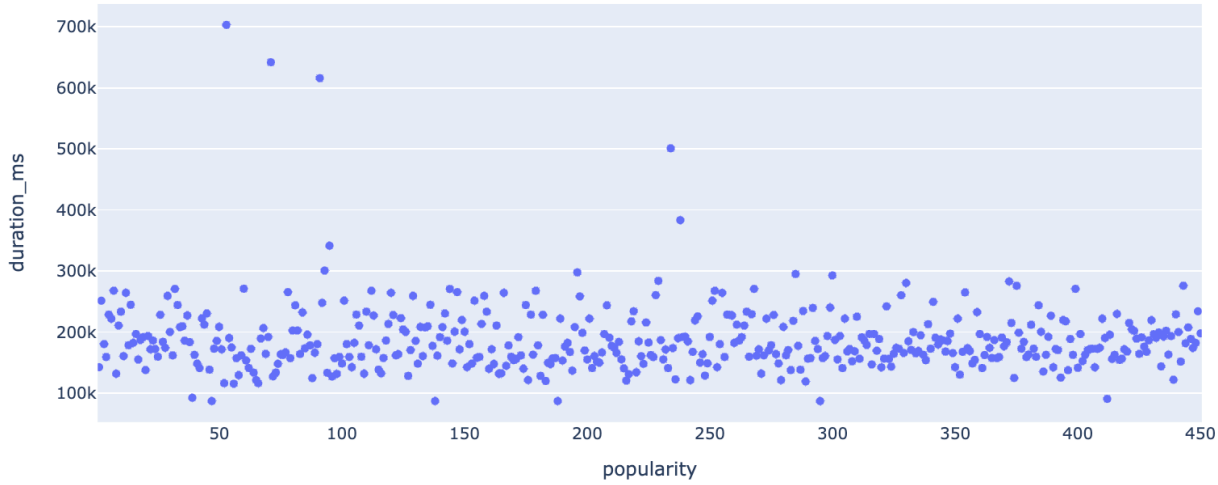
Energia x popularidade



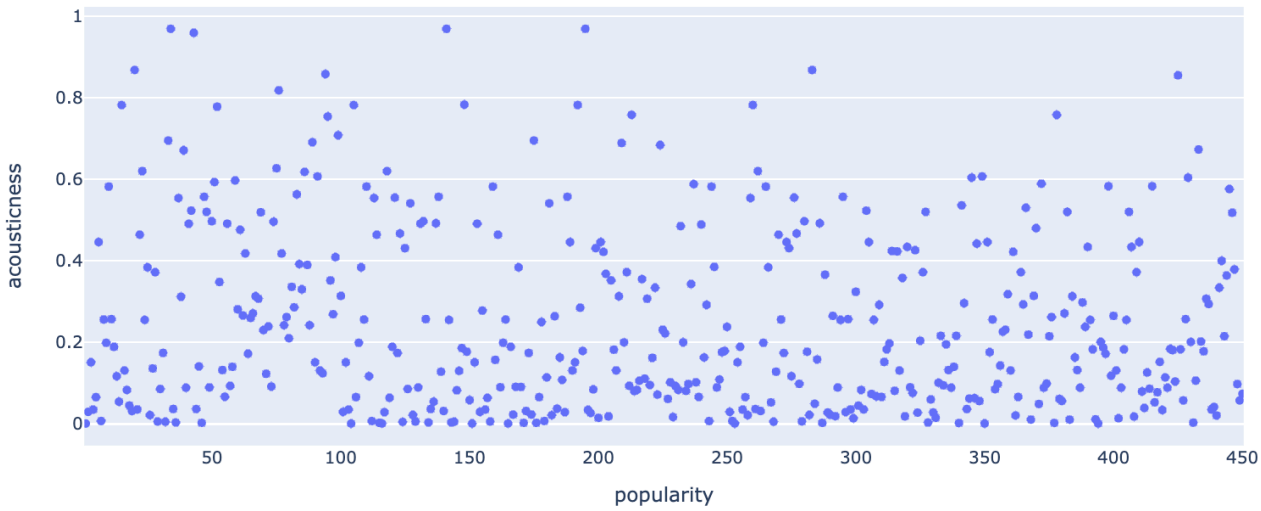
Instrumental x popularidade



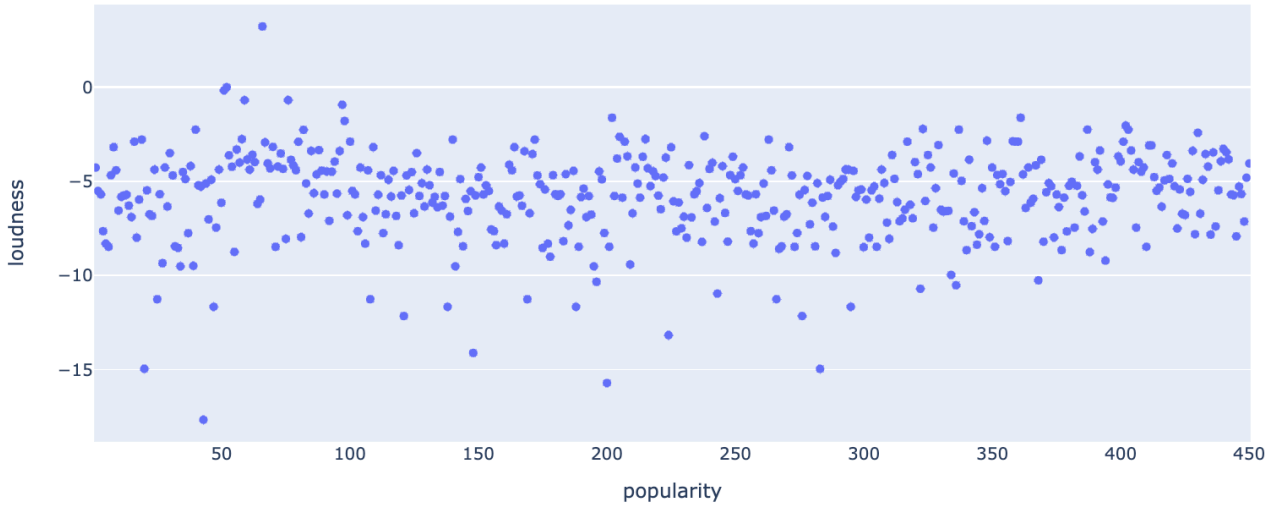
Duração da música x popularidade



Acusticidade x popularidade



Intensidade x popularidade



Valência x popularidade