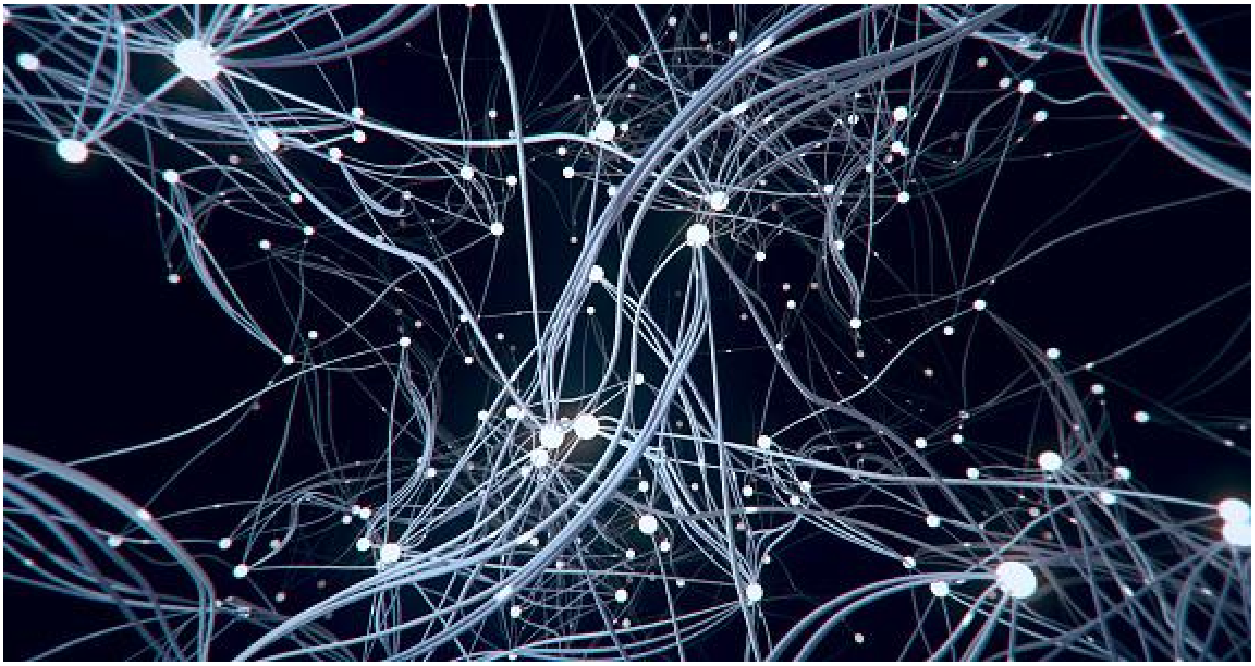


# Spotify Music Analysis Using ML Techniques

15.01.2023

Dilara KIZILKAYA - 1030510143

Mustafa Furkan DEMİRBİLEK - 1030510031

## • Introduction

In this design project, we aimed to do a basic data analysis of the music data and recommendation system using Python's Spotipy library and Machine Learning Techniques.

## • Goals

Our goals in this design project are:

1. Learn how to use Spotipy library.
2. Collect music data using Spotipy library.
3. Preprocessing.
4. Model selection.
5. General analysis.
6. Creating a recommendation system.

# 1. Spotipy

*Spotipy* is a Python library for the Spotipy WEB API. With spotipy one can get a full access to all the music data provided by the Spotify platform.

We used Spotipy for collecting the music data. To be able to do that, we first entered Spotify's WEB API ("Spotify for Developers"). In the WEB API, in the Dashboard tab we log into our Spotify account and create an app. This app provides us with a Client ID and a Client Secret ID. These IDs are necessary to reach the API.

Our music data comes with many features, some of them are:

| KEY | VALUE TYPE | VALUE DESCRIPTION |
|---|---|---|
| acousticness | float | A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic. |
| analysis_url | string | An HTTP URL to access the full audio analysis of this track. An access token is required to access this data. |

| | | |
|---|---|---|
| danceability | float | 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. |
| duration_ms | int | The duration of the track in milliseconds. |
| energy | float | 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. |
| id | string | The Spotify ID for the track. |
| instrumentalness | float | Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0. |
| key | int | 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. |
| liveness | float | 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. |
| loudness | float | 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. |
| mode | int | 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. |
| speechiness | float | Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks. |
| tempo | float | The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration. |

| | | |
|---|---|---|
| tempo | float | 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. |
| time_signature | int | 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). |
| track_href | string | A link to the Web API endpoint providing full details of the track. |
| type | string | The object type: "audio_features" |
| uri | string | The Spotify URI for the track. |
| valence | float | 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). |

## 2. Collecting the data

Before we started to pull the data from Spotify, we created two playlists called “Liked” and “Disliked”. We did that because we needed some previously classified data as “liked” and “disliked” in order to train our model.

After we completed both of the playlists, it was time to use the Spotipy library.

```
In [1]: import pandas as pd
        import spotipy
        from spotipy.oauth2 import SpotifyClientCredentials
        from collections import Counter
```

```
In [2]: sp = spotipy.Spotify()

        cid ="d062190fd93b4e4488e75ae2d9256814"

        secret = "5afbd4a19ee54607be0db6036fb417c2"

        client_credentials_manager = SpotifyClientCredentials(client_id=cid, client_secret=secret)

        sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)

        sp.trace=False
```

```
In [4]: playlist_l = sp.user_playlist("Liked", "3cYqEaPBFyxqlXbViPtzDv?si=3553c6c98f714847")
        playlist_d = sp.user_playlist("Disliked", "4HC1kIY2bof1CHfZqmo8u1?si=080494e8e1bf4f75")
```

As seen in the code, first we uploaded necessary libraries. Then called the Spotify() function. As mentioned in the previous section, there are two ids(Client ID and Client Secret ID) that Spotify provides us. For an easy usage we assigned them to two variables called ‘cid’ and ‘secret’. We will be needing them as parameters for the SpotifyClientCredentials() function.

Other lines are required for us to enter our user space inside the Spotify.

As explained before, we created two playlists called 'liked' and 'disliked'. In the code block 4, we use Spotipy's user_playlist() method and get the 'liked' data and 'disliked' data. The method takes two parameters which are the name of the playlist and the link code of the playlist.

```
In [8]: liked_songs = playlist_l["tracks"]["items"]
```

```
In [21]: disliked_songs = playlist_d["tracks"]["items"]
```

In the code block 8 and 158, we created two lists called 'liked_songs' and 'disliked_songs'. In these lines we fill our lists with tracks and their items.

```
In [22]: disliked_ids = []
         for i in range(len(disliked_songs)):
             disliked_ids.append(disliked_songs[i]["track"]["id"])

In [23]: disliked_features = sp.audio_features(disliked_ids)

         disliked_features

Out[23]: [{'danceability': 0.742,
           'energy': 0.497,
           'key': 0,
           'loudness': -9.381,
           'mode': 0,
           'speechiness': 0.0746,
           'acousticness': 0.249,
           'instrumentalness': 0.00335,
           'liveness': 0.0993,
           'valence': 0.558,
           'tempo': 156.013,
           'type': 'audio_features',
           'id': '3dEtfTtCXWW7N8WQ9FD29z',
           'uri': 'spotify:track:3dEtfTtCXWW7N8WQ9FD29z',
           'track_href': 'https://api.spotify.com/v1/tracks/3dEtfTtCXWW7N8WQ9FD29z',
           'analysis_url': 'https://api.spotify.com/v1/audio-analysis/3dEtfTtCXWW7N8WQ9FD29z',
           'duration_ms': 310860,
           'time_signature': 4},
          {'danceability': 0.699,
```

```
In [16]: liked_ids = []
         for i in range(len(liked_songs)):
             liked_ids.append(liked_songs[i]["track"]["id"])

In [17]: liked_features = sp.audio_features(liked_ids)

         liked_features

Out[17]: [{'danceability': 0.271,
           'energy': 0.165,
           'key': 5,
           'loudness': -20.652,
           'mode': 1,
           'speechiness': 0.0351,
           'acousticness': 0.729,
           'instrumentalness': 1.6e-06,
           'liveness': 0.118,
           'valence': 0.203,
           'tempo': 77.082,
           'type': 'audio_features',
           'id': '29U7stRjqHU6rMiS8BfaI9',
           'uri': 'spotify:track:29U7stRjqHU6rMiS8BfaI9',
           'track_href': 'https://api.spotify.com/v1/tracks/29U7stRjqHU6rMiS8BfaI9',
           'analysis_url': 'https://api.spotify.com/v1/audio-analysis/29U7stRjqHU6rMiS8BfaI9',
           'duration_ms': 139227,
           'time_signature': 4},
          {'danceability': 0.177,
```

In the code blocks 22 and 16, we get the track ids and use them in the code blocks 23 and 17 in order to get the specified audio features. (First audio's features can be seen in the Output[23] and Output[17].)

After we got all the features that specify the specific music's features. We need to turn the lists into the Pandas Dataframe.

[Pandas Dataframe is two-dimensional, size-mutable, potentially heterogeneous tabular data. Data structure also contains labeled axes (rows and columns). Arithmetic operations align on both row and column labels. Can be thought of as a dict-like container for Series objects.]

In [18]:
```
liked_df = pd.DataFrame(liked_features)
liked_df
```

Out[18]:

| | danceability | energy | key | loudness | mode | speechiness | acousticness | instrumentalness | liveness | valence | tempo | type | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.271 | 0.165 | 5 | -20.652 | 1 | 0.0351 | 0.7290 | 0.000002 | 0.1180 | 0.2030 | 77.082 | audio_features | 29U7stRjqHU6rMiS8I |
| **1** | 0.177 | 0.215 | 3 | -9.886 | 1 | 0.0335 | 0.8070 | 0.000125 | 0.0921 | 0.0415 | 75.862 | audio_features | 2MfwT3SKUyF0na2LCa |
| **2** | 0.530 | 0.497 | 0 | -11.348 | 1 | 0.0322 | 0.4980 | 0.000384 | 0.0665 | 0.1000 | 141.977 | audio_features | 5b7wNSn8yuj5ZZ3JA |
| **3** | 0.617 | 0.567 | 0 | -4.188 | 1 | 0.0828 | 0.0584 | 0.000000 | 0.0933 | 0.5050 | 90.246 | audio_features | 1CnPYaKxTVb4LWOtiGO |
| **4** | 0.214 | 0.367 | 6 | -10.858 | 0 | 0.0321 | 0.8440 | 0.018100 | 0.1880 | 0.1750 | 79.854 | audio_features | 1M8OAc4NnD5GFMUiqj |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |

In [24]:
```
disliked_df = pd.DataFrame(disliked_features)
disliked_df
```

Out[24]:

| | danceability | energy | key | loudness | mode | speechiness | acousticness | instrumentalness | liveness | valence | tempo | type | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.742 | 0.4970 | 0 | -9.381 | 0 | 0.0746 | 0.24900 | 0.003350 | 0.0993 | 0.5580 | 156.013 | audio_features | 3dEtfTtCXWW7N8WQ9F |
| **1** | 0.699 | 0.5880 | 9 | -6.533 | 0 | 0.0402 | 0.00807 | 0.000131 | 0.1070 | 0.3500 | 151.971 | audio_features | 2TaWcNAkUM6buVOkL0 |
| **2** | 0.642 | 0.7230 | 1 | -6.149 | 1 | 0.2430 | 0.58000 | 0.000942 | 0.1120 | 0.4870 | 186.043 | audio_features | 1EYeZGTMCjV7uVey5fN |
| **3** | 0.862 | 0.5060 | 10 | -8.651 | 0 | 0.2670 | 0.05420 | 0.000030 | 0.1170 | 0.5120 | 136.052 | audio_features | 5h1Zdr8ghV5RaWjtd |
| **4** | 0.921 | 0.4320 | 0 | -11.109 | 1 | 0.0502 | 0.50600 | 0.002230 | 0.0617 | 0.6930 | 112.117 | audio_features | 4Pv7OY4tfCA8Kldtlk |

After we created these two dataframes, we needed to combine them in order to get a mixed whole dataset.

In [27]:
```
#combining

frames = [liked_df, disliked_df]

result = pd.concat(frames).reset_index()
result
```

Out[27]:

| | index | danceability | energy | key | loudness | mode | speechiness | acousticness | instrumentalness | liveness | valence | tempo | type | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0.271 | 0.1650 | 5 | -20.652 | 1 | 0.0351 | 0.7290 | 0.000002 | 0.1180 | 0.2030 | 77.082 | audio_features | 29U7stRjqH |
| **1** | 1 | 0.177 | 0.2150 | 3 | -9.886 | 1 | 0.0335 | 0.8070 | 0.000125 | 0.0921 | 0.0415 | 75.862 | audio_features | 2MfwT3SKUyF |
| **2** | 2 | 0.530 | 0.4970 | 0 | -11.348 | 1 | 0.0322 | 0.4980 | 0.000384 | 0.0665 | 0.1000 | 141.977 | audio_features | 5b7wNSn8y |
| **3** | 3 | 0.617 | 0.5670 | 0 | -4.188 | 1 | 0.0828 | 0.0584 | 0.000000 | 0.0933 | 0.5050 | 90.246 | audio_features | 1CnPYaKxTVb4 |
| **4** | 4 | 0.214 | 0.3670 | 6 | -10.858 | 0 | 0.0321 | 0.8440 | 0.018100 | 0.1880 | 0.1750 | 79.854 | audio_features | 1M8OAc4NnD5 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **136** | 36 | 0.270 | 0.4740 | 5 | -11.531 | 1 | 0.0337 | 0.6060 | 0.000000 | 0.4230 | 0.5240 | 79.144 | audio_features | 2ydyrhV1UFn |
| **137** | 37 | 0.330 | 0.4390 | 10 | -9.801 | 0 | 0.0410 | 0.6250 | 0.000368 | 0.1770 | 0.2100 | 175.676 | audio_features | 5NPg92vbjaGk4 |
| **138** | 38 | 0.679 | 0.1560 | 2 | -13.920 | 1 | 0.0360 | 0.9080 | 0.000000 | 0.1030 | 0.2650 | 135.846 | audio_features | 6Qu9OC9o7f8C |
| **139** | 39 | 0.435 | 0.1590 | 7 | -17.582 | 0 | 0.0494 | 0.9790 | 0.377000 | 0.1170 | 0.3320 | 109.076 | audio_features | 3m8Zy0Tw6Fl1 |
| **140** | 40 | 0.190 | 0.0672 | 4 | -23.670 | 0 | 0.0360 | 0.9890 | 0.917000 | 0.0700 | 0.0606 | 67.532 | audio_features | 27krluDgmk6 |

141 rows × 20 columns

We combined our dataframes with the pandas's concat() method. After combining we used reset_index() to reset the indexes.

But now the problem was that our dataset wasn't shuffled, it was first ordered as the 'liked' dataset then the 'disliked' dataset. So to fix that we needed to shuffle the dataset as well.

In [28]:
```
#shuffle
result = result.sample(frac = 1).reset_index()
result
```

To achieve that we used the sample() function.

[**sample()** is an inbuilt function of **random module** in Python that returns a particular length list of items chosen from the sequence i.e. list, tuple, string or set. Used for random sampling without replacement.]()

After applying all the methods, our dataset was ready to be analyzed.

Out[28]:

| | level_0 | index | danceability | energy | key | loudness | mode | speechiness | acousticness | instrumentalness | ... | valence | tempo | type | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 25 | 25 | 0.730 | 0.2850 | 11 | -15.103 | 0 | 0.0319 | 0.877 | 0.012600 | ... | 0.2840 | 107.140 | audio_features | 3h9T2wL |
| **1** | 81 | 81 | 0.724 | 0.4650 | 9 | -8.088 | 1 | 0.0361 | 0.642 | 0.039000 | ... | 0.3590 | 115.037 | audio_features | 1TQPsGG43v |
| **2** | 51 | 51 | 0.339 | 0.4300 | 0 | -10.102 | 1 | 0.0440 | 0.667 | 0.031200 | ... | 0.1260 | 105.236 | audio_features | 4FIR1nTr9 |
| **3** | 85 | 85 | 0.728 | 0.4560 | 7 | -7.930 | 1 | 0.0279 | 0.380 | 0.046700 | ... | 0.1360 | 114.008 | audio_features | 60u9Wxwtz |
| **4** | 32 | 32 | 0.553 | 0.1900 | 2 | -14.961 | 1 | 0.0298 | 0.903 | 0.000227 | ... | 0.2990 | 100.002 | audio_features | 5OHbgQb |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **136** | 125 | 25 | 0.200 | 0.4660 | 9 | -12.298 | 1 | 0.0486 | 0.557 | 0.702000 | ... | 0.0572 | 136.913 | audio_features | 07eGxuz8bL |
| **137** | 12 | 12 | 0.653 | 0.5050 | 10 | -7.102 | 0 | 0.0261 | 0.766 | 0.000008 | ... | 0.3880 | 96.109 | audio_features | 7Dq7IjABZf |
| **138** | 50 | 50 | 0.411 | 0.0595 | 3 | -15.542 | 1 | 0.0357 | 0.974 | 0.000013 | ... | 0.1340 | 78.560 | audio_features | 2i7XH68Y |
| **139** | 11 | 11 | 0.535 | 0.0609 | 11 | -17.805 | 1 | 0.0542 | 0.930 | 0.000144 | ... | 0.1720 | 106.261 | audio_features | 7lY3juj1 |
| **140** | 124 | 24 | 0.224 | 0.1390 | 11 | -22.587 | 1 | 0.0354 | 0.961 | 0.913000 | ... | 0.2130 | 111.316 | audio_features | 6Q5uDNuuF |

141 rows × 21 columns

# 3. Preprocessing

First of all, when we look at the data we realized that we had a few features that won't help us while classifying.

These features were "target", "level_0", "index", "type", "id", "uri", "track_href", "analysis_url". So we decided to drop them from the dataset.

In [29]:

```
X = result.drop(["target", "level_0", "index", "type", "id", "uri", "track_href", "analysis_url"], axis=1)
X
```

Out[29]:

| | danceability | energy | key | loudness | mode | speechiness | acousticness | instrumentalness | liveness | valence | tempo | duration_ms | time_signature |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.730 | 0.2850 | 11 | -15.103 | 0 | 0.0319 | 0.877 | 0.012600 | 0.1370 | 0.2840 | 107.140 | 330747 | 4 |
| **1** | 0.724 | 0.4650 | 9 | -8.088 | 1 | 0.0361 | 0.642 | 0.039000 | 0.0637 | 0.3590 | 115.037 | 185107 | 4 |
| **2** | 0.339 | 0.4300 | 0 | -10.102 | 1 | 0.0440 | 0.667 | 0.031200 | 0.2530 | 0.1260 | 105.236 | 293267 | 4 |
| **3** | 0.728 | 0.4560 | 7 | -7.930 | 1 | 0.0279 | 0.380 | 0.046700 | 0.0855 | 0.1360 | 114.008 | 275146 | 4 |
| **4** | 0.553 | 0.1900 | 2 | -14.961 | 1 | 0.0298 | 0.903 | 0.000227 | 0.0993 | 0.2990 | 100.002 | 239653 | 4 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **136** | 0.200 | 0.4660 | 9 | -12.298 | 1 | 0.0486 | 0.557 | 0.702000 | 0.1080 | 0.0572 | 136.913 | 606850 | 5 |
| **137** | 0.653 | 0.5050 | 10 | -7.102 | 0 | 0.0261 | 0.766 | 0.000008 | 0.0701 | 0.3880 | 96.109 | 228907 | 4 |
| **138** | 0.411 | 0.0595 | 3 | -15.542 | 1 | 0.0357 | 0.974 | 0.000013 | 0.1850 | 0.1340 | 78.560 | 272041 | 3 |
| **139** | 0.535 | 0.0609 | 11 | -17.805 | 1 | 0.0542 | 0.930 | 0.000144 | 0.1060 | 0.1720 | 106.261 | 224813 | 3 |
| **140** | 0.224 | 0.1390 | 11 | -22.587 | 1 | 0.0354 | 0.961 | 0.913000 | 0.1080 | 0.2130 | 111.316 | 196893 | 4 |

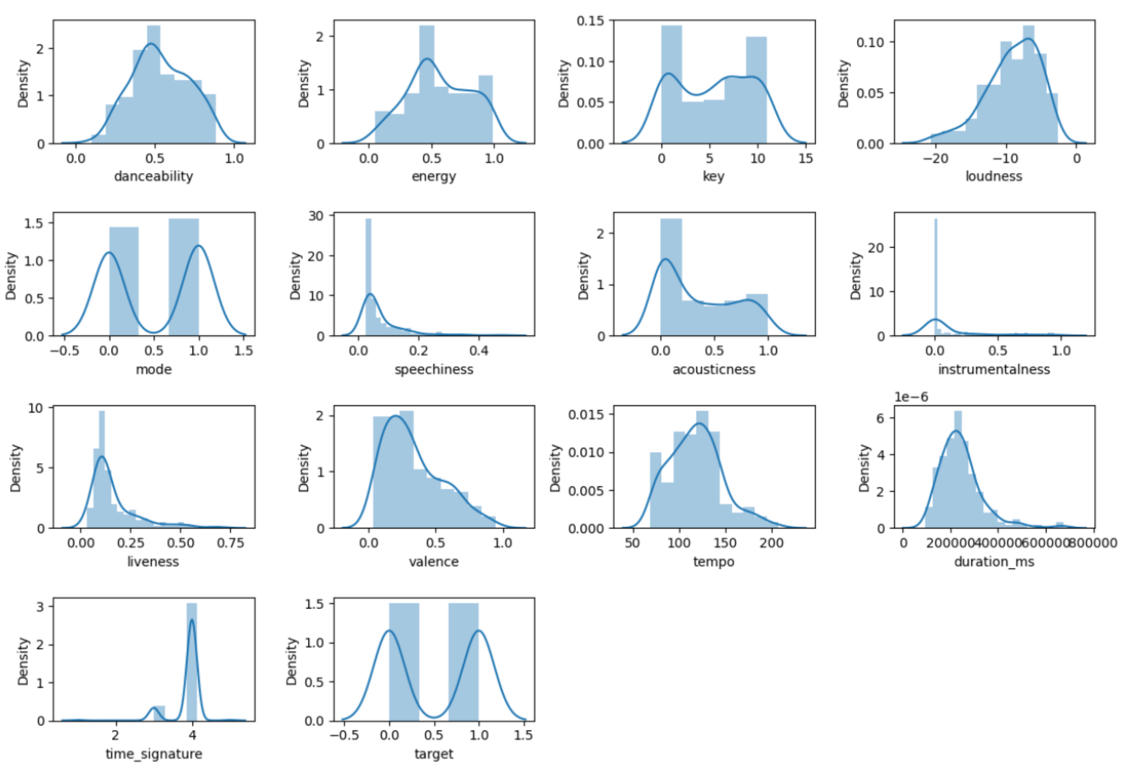
141 rows × 13 columns

As seen in the code block 29, we used the drop() method. We specified the columns that we want to drop (the parameter should be list like or single label), then specified the axis which in here means the columns.

# 4. Visualization

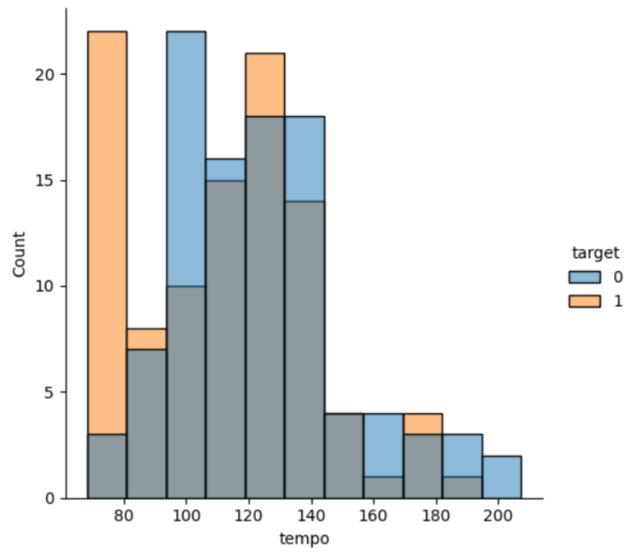
In this phase, we used some visualization methods to understand our dataset better.

```
In [26]: import seaborn as sns

         numerical_features = result.select_dtypes(exclude=['object']).drop(['level_0', 'index'], axis=1).copy()

         fig = plt.figure(figsize=(12,18))
         for i in range(len(numerical_features.columns)):
             fig.add_subplot(9,4,i+1)
             sns.distplot(numerical_features.iloc[:,i].dropna())
             plt.xlabel(numerical_features.columns[i])
         plt.tight_layout()
         plt.show()
```




In this code block we visualize all feature columns and one class column that we had after dropping the useless features. First, we took “numerical features” (we did not take categorical data because they are meaningless features for us) to measure their densities. We wanted to show all features so we implemented a for loop.

```
In [28]: sns.displot(data=numerical_features, x="tempo", hue="target")
Out[28]: <seaborn.axisgrid.FacetGrid at 0x239153d5d30>
```

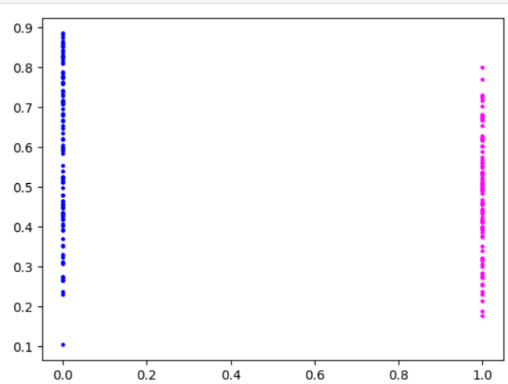



In this bar graph, we can easily see our numeraical_features's distribution according to classes which are liked and disliked. We used the seaborn library to visualize these values.

```
In [30]: import matplotlib.pyplot as plt

         color = []

         for i in range(0, len(y)):
             if y[i] == 0:
                 color.append("blue")
             else:
                 color.append("magenta")
```

```
In [31]: columns = []

         for col in X.columns:
             columns.append(col)
```

```
In [32]: for i in range(0, 200):
             plt.scatter(y[i], X[columns[0]][i], c=color[i], s=10, linewidth = 0)
```



Here, we visualized our columns and their whole samples in a scatter plot. When we did this we used the Scatter() method from the matplotlib library. We implemented a for loop to give different colors to each class.

## 5. Building the Model

This phase includes, splitting the data, scaling phrases, building the machine learning model and examining metrics.

```
In [201]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=8)
```

Here,  we determined our train and test data. We tried “ 0.2, 0.3, 0.25 ” values for test_size and we decided for 0.25 as the best one.

```
In [219]: sc = StandardScaler()

          X_train = sc.fit_transform(X_train)
          X_test = sc.fit_transform(X_test)
```

We also used a scaling method so that one feature does not dominate the others. We used the StandardScaler method from the sklearn library.

```
In [71]: clf = RandomForestClassifier(criterion='entropy')
         clf.fit(X_train, y_train)

         y_pred = clf.predict(X_test)
```

In this code block, we implemented our algorithm. We tried all algorithms which in the environment and Random Forest algorithm has the best performance.

```
In [72]: cm = confusion_matrix(y_test, y_pred)
         cm

Out[72]: array([[22,  4],
                [ 1, 23]], dtype=int64)
```

```
In [73]: acc = accuracy_score(y_test, y_pred)
         print('\n Accuracy Score:', acc)

         f1 = f1_score(y_test, y_pred, average='macro')
         print('\n F1 Score:', f1)

         auc = roc_auc_score(y_test, y_pred)
         print('\n AUC Score:', auc)

          Accuracy Score: 0.9

          F1 Score: 0.8999599839935976

          AUC Score: 0.9022435897435898
```

In this code block we want to show our results by some metrics. Accuracy, F1 score and AUC under ROC curve metrics are some of them. We used the entropy metric for the “criterion” parameter because it is more suitable for our dataset.

```
In [75]: class_1_tpr = cm[0, 0]
         class_2_tpr = cm[1, 1]

         class_1 = np.sum(cm[0, :])
         class_2 = np.sum(cm[1, :])

         sens = class_1_tpr / class_1
         spec = class_2_tpr / class_2
```

```
In [76]: print('Sensitivity:', sens)
         print('Specifity:', spec)

         Sensitivity: 0.8461538461538461
         Specifity: 0.9583333333333334
```

We also analyze sensitivity and specificity values to understand better.

$$\text{Sensitivity:} \ = \frac{correct\ number\ of\ prediction\ of\ the\ first\ class}{total number of elements in the first class}$$
$$\text{Specificity:} \ = \frac{correct\ number\ of\ prediction\ of\ the\ second\ class}{total number of elements in the second class}$$

We implemented that code according to these formulas.

```
In [217]: resp_0 = "This song is not in your preferences!"
          resp_1 = "Wonderful! This song is definitely made for you!"

          y_pred_resp = []

          for i in range(0, len(y_pred)):
              if y_pred[i] == 0:
                  y_pred_resp.append(resp_0)
              else:
                  y_pred_resp.append(resp_1)

                  y_pred_resp
          y_pred_resp
```

```
Out[217]: ['Wonderful! This song is definitely made for you!',
           'This song is not in your preferences!',
           'Wonderful! This song is definitely made for you!',
           'This song is not in your preferences!',
           'This song is not in your preferences!',
           'This song is not in your preferences!',
           'Wonderful! This song is definitely made for you!',
           'This song is not in your preferences!',
           'Wonderful! This song is definitely made for you!',
           'Wonderful! This song is definitely made for you!',
           'This song is not in your preferences!',
           'This song is not in your preferences!',
           'Wonderful! This song is definitely made for you!',
           'This song is not in your preferences!',
           'This song is not in your preferences!',
           'Wonderful! This song is definitely made for you!',
           'This song is not in your preferences!',
           'This song is not in your preferences!',
           'Wonderful! This song is definitely made for you!',
```

```
In [218]: y_pred

Out[218]: array([1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
                 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1,
                 1, 1, 1, 1, 0, 1], dtype=int64)
```

In addition, we implemented a return code block. We aim that true predictions will return as “Wonderful! This song is definitely made for you!” message and false predictions will return as “This song is not in your preferences!” message.

| Algortihm | Confusion Matrix | Accuracy Score | F1 Score | AUC ROC Curve | Sensitivity | Specifity | Cross-validation Scores |
|---|---|---|---|---|---|---|---|
| Gaussian Naive Bayes | [[23, 3]<br>[ 1, 23]] | 0.92 | 0.9199999999999999 | 0.921474358974359 | 0.8846153846153846 | 0.9583333333333334 | [0.6 , 0.75 , 0.75 , 0.6 , 0.675] |
| X-Gboost Classifier | [[22, 4]<br>[ 1, 23]] | 0.9 | 0.8999599839935976 | 0.9022435897435898 | 0.8461538461538461 | 0.9583333333333334 | [0.875, 0.8 , 0.85 , 0.85 , 0.775] |
| Random Forest Classifier | [[23, 3]<br>[ 2, 22]] | 0.9 | 0.8999599839935974 | 0.9006410256410255 | 0.8846153846153846 | 0.9166666666666666 | [0.85 , 0.775, 0.825, 0.875, 0.825] |
| K-Nearest Neighbors Classifier | [[22, 4]<br>[ 2, 22]] | 0.88 | 0.8799999999999999 | 0.8814102564102563 | 0.8461538461538461 | 0.9166666666666666 | [0.6 , 0.725, 0.525, 0.65 , 0.55 ] |
| Support Vector Classifier | [[22, 4]<br>[ 2, 22]] | 0.88 | 0.8799999999999999 | 0.8814102564102563 | 0.8461538461538461 | 0.9166666666666666 | [0.575, 0.7 , 0.55 , 0.6 , 0.575] |
| LogisticRegression | [[20, 6]<br>[ 2, 22]] | 0.84 | 0.8397435897435896 | 0.8429487179487177 | 0.7692307692307693 | 0.9166666666666666 | [0.525, 0.525, 0.725, 0.675, 0.675] |
| DecisionTreeClassifier | [[20, 6]<br>[ 6, 18]] | 0.76 | 0.7596153846153846 | 0.7596153846153846 | 0.7692307692307693 | 0.75 | [0.8 , 0.7 , 0.7 , 0.725, 0.725] |

Finally, we all get together each algorithm's performance according to metrics. As results we can see which algorithm determines how much score from each metric.

## 6. GITHUB

Throughout this project, we used github.

https://github.com/vyperid/Design-Project

https://github.com/Furkan-png/Design-Project

## SOURCES


https://spotipy.readthedocs.io/en/2.22.0/

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

https://www.geeksforgeeks.org/python-random-sample-function/

https://lazypredict.readthedocs.io/en/latest/#

https://scikit-learn.org/stable/

https://matplotlib.org/stable/index.html

https://seaborn.pydata.org/