

Music recommendation system using Spotify API



Music Adviser

by Raphaël CANIN & Melisa KOCKAN

Normal search



Submit

Raphaël CANIN, Melisa KOCKAN
E4 AIC

Introduction	3
Construction of the database	4
Perfecting the dataset	5
Data visualization	6
Recommender System	10
Dimensional reduction	10
Finding the hyper parameter k	11
Clustering	12
Music recommendation	12
Web application	16
Deployment on AWS	17

Introduction

This project suggested 3 different subjects, we decided to choose the subject 3 "Build a recommendation system".

In order to implement a recommendation system, we created a music recommendation system using Spotify. To do, we created a database using Spotipy for developers (<https://developer.spotify.com/>) with Melisa's account since she had a subscription.

To create this database, we used all the titles provided by an existing database on Kaggle : <https://www.kaggle.com/dhruvildave/spotify-charts>.

Once it was done, we created a web application using Flask and deployed it on AWS.

Construction of the database

We connected onto the spotify platform in order to access the client ID and the secret key.

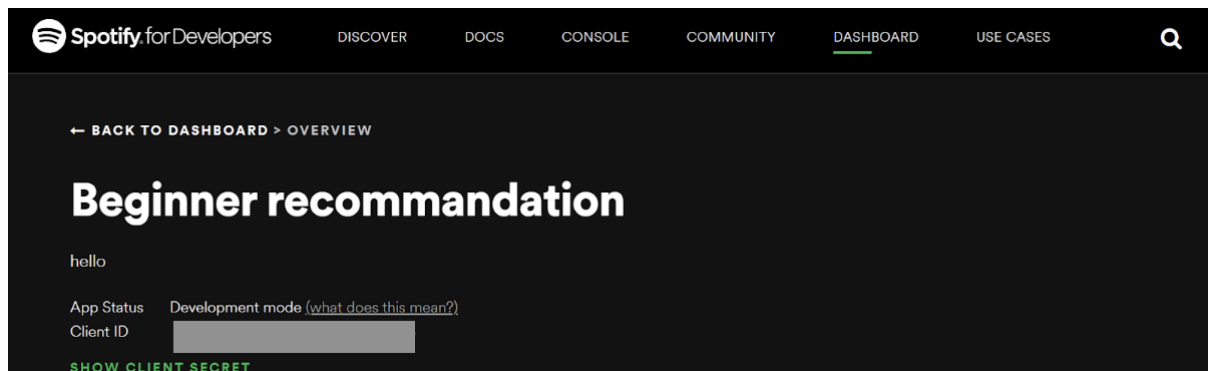


Fig. 1 - Spotify API Dashboard

Therefore, we connected onto spotify using jupyter thanks to the *spotipy* module on Python. Here is a brief overview of what we can do on Jupyter using the spotify module :

```
1 title = "bellyache"

1 track = sp.search(title, limit=1)["tracks"]["items"][0]
2
3 print('track : ', track['name'])
4 print('audio : ', track['preview_url'])
5 print('cover art : ', track['album']['images'][0]['url'])
6 print()

track : bellyache
audio : None
cover art: https://i.scdn.co/image/ab67616d0000b2734adbeb26299adca766cec2c5
```

Fig. 2 - Data on a track

From the title of a song, we can access the url enabling to access to the song.

```
1 sp.audio_features(track['id'])

[{'danceability': 0.695,
 'energy': 0.573,
 'key': 4,
 'loudness': -8.597,
 'mode': 0,
 'speechiness': 0.106,
 'acousticness': 0.46,
 'instrumentalness': 0.0734,
 'liveness': 0.482,
 'valence': 0.408,
 'tempo': 99.939,
 'type': 'audio_features',
 'id': '51NFxnQvaosfDDutk0tams',
 'uri': 'spotify:track:51NFxnQvaosfDDutk0tams',
 'track_href': 'https://api.spotify.com/v1/tracks/51NFxnQvaosfDDutk0tams',
 'analysis_url': 'https://api.spotify.com/v1/audio-analysis/51NFxnQvaosfDDutk0tams',
 'duration_ms': 179172,
 'time_signature': 4}]
```

Fig. 3 - Fetching useful features from the API

Using the id associated with the title of a song, we can access the characteristics of a song. Hence, the only thing we needed in order to build a dataset for our recommender system was to possess a list of titles. The dataset we chose to access to a list of titles is the following one : <https://www.kaggle.com/datasets/dhruvildave/spotify-charts>.

We opened this dataset using a Dataframe on Pandas; it contains 26173513 lines each corresponding to a title. We used it in order to collect characteristics for each songs ; however, we got the following error : `httpsconnectionpool(host='api.spotify.com', port=443): read timed out`

In order to bypass this error, we firstly tried to increase the timeout ; it helped but wasn't enough yet to fix the problem. Therefore, we decided to use the try / except keywords in order to definitely bypass these errors.

As we have seen in the paragraph above, the information given for a title is provided inside a dictionary. We stored all the information in a list, which contained the title of the song, the name of the artist, and a dictionary of all the characteristics we collected.

We transformed this list into a DataFrame, which contains the 3 columns we mentioned above. We estimated that it would be easier for the rest of this project to have all the information in a panda dataframe in which each characteristic corresponds to a single column.

To obtain this, we created a sub data frame containing only the 3rd column. We looped over each line using the keys of the dictionary, stored it into a list, and used `pd.DataFrame.from_dict` to get a Dataframe from this list. Finally, we merged this dataframe with the other dataframe of 2 columns.

Perfecting the dataset

This dataframe contained many imperfections. As an instance, since we found it strange that the dataset contained an amount of music as huge as this, we used the function `values.count` in order to see how many songs each artist composed according to our dataset.

According to the dataset, Ed Sheeran composed thousands of songs ; which isn't true. Hence we cleaned the dataset by removing all the duplicata having the same title of songs thanks to the command `drop_duplicates`.

Owing to this procedure, we went from a dataset of 26173513 lines to a dataset of only 197535 lines. It was important to reset the index of the dataframe in order to know the correct amount of lines left, since the `drop_duplicates` doesn't update the index automatically. Since the dataset was still huge, we only selected 10000 titles of songs.

Another issue that we faced was the absence of some values. We replaced each nan value with the mean of its column for the numerical columns, and with ' ' for each non

numerical column. This was very easy to do thanks to the command `fillna` onto our DataFrame.

Last but not least, we found it relevant to have the type of music and to keep only the titles of songs for each the url of the extract was available. To do so, we used the same methods as in the first part - for more information, check the jupyter notebook provided.

Data visualization

The name of the characteristics collected is already explicit, but we found a website explaining precisely the description of each of them :

<https://towardsdatascience.com/is-my-spotify-music-boring-an-analysis-involving-music-data-and-machine-learning-47550ae931de>

Here is a brief description of each of them :

Instrumentalness - vocals in the song

Acousticness - Describes if the song is acoustic or not

Liveness - probability that the song was recorded during a concert on live

Speechiness - presence of spoken words

Energy - enables to measure the intensity and the loudness of a song

Danceability - tells if the music is adapted to dance or not, according to other characteristics such as tempo, rhythm stability, beat

Valence - grades the positiveness of a track

There are 2040 different types of music in our dataset ; according to the pie chart we created, here are the 15 most present types of music :

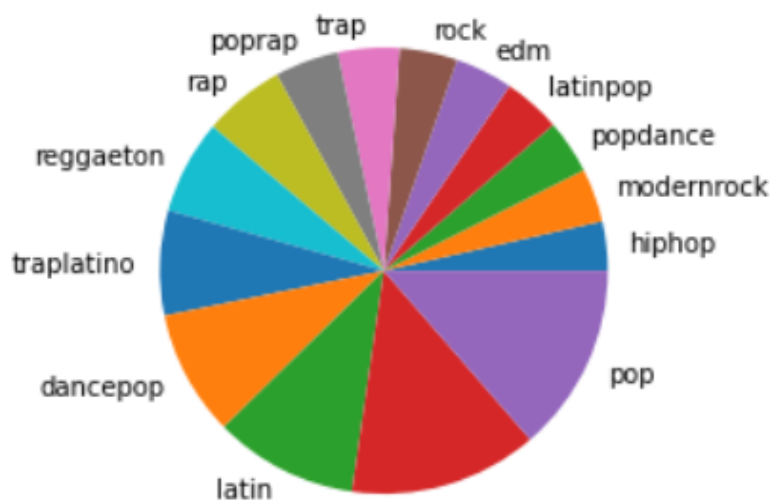


Fig. 4 - 15 most present types of music in our dataset

We created a sub dataset for 3 different types of music : dancepop, latino, and boyband. As expected, dancepop and latino are more danceable than the boyband genre. However, they all have a similar high energy. Dance Pop and latino genres are louder than the boy band genre. All three genres were probably recorded in the studio with a fairly low liveness.

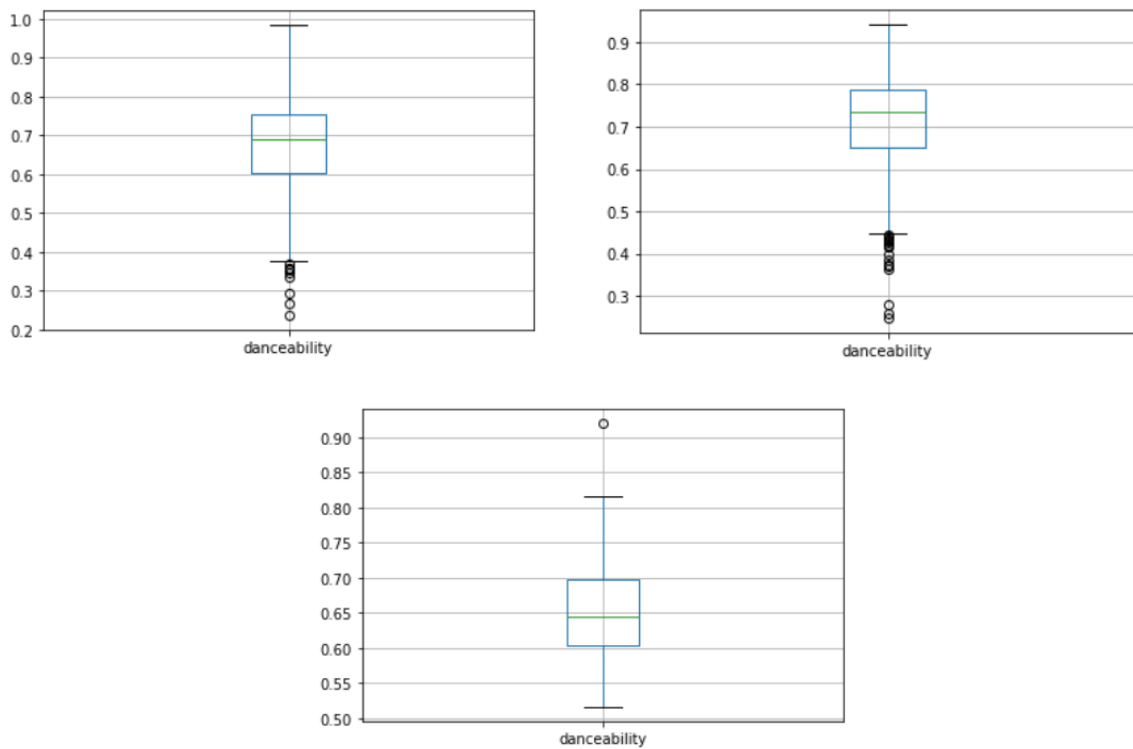
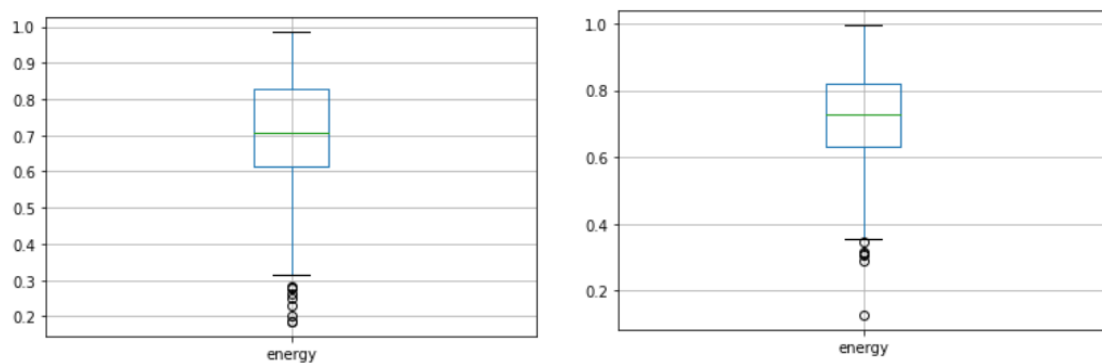


Fig. 5 - Danceability for dancepop, latino, and boyband



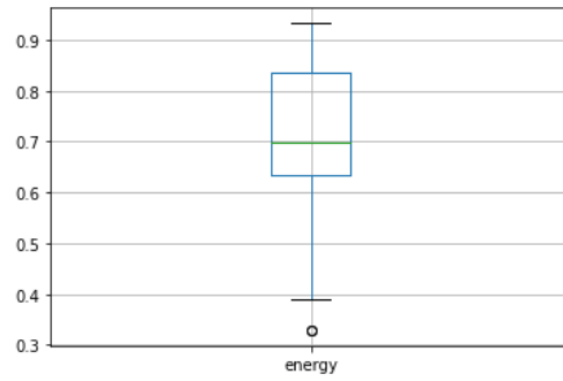


Fig. 6 - Energy for dancepop, latino, and boyband

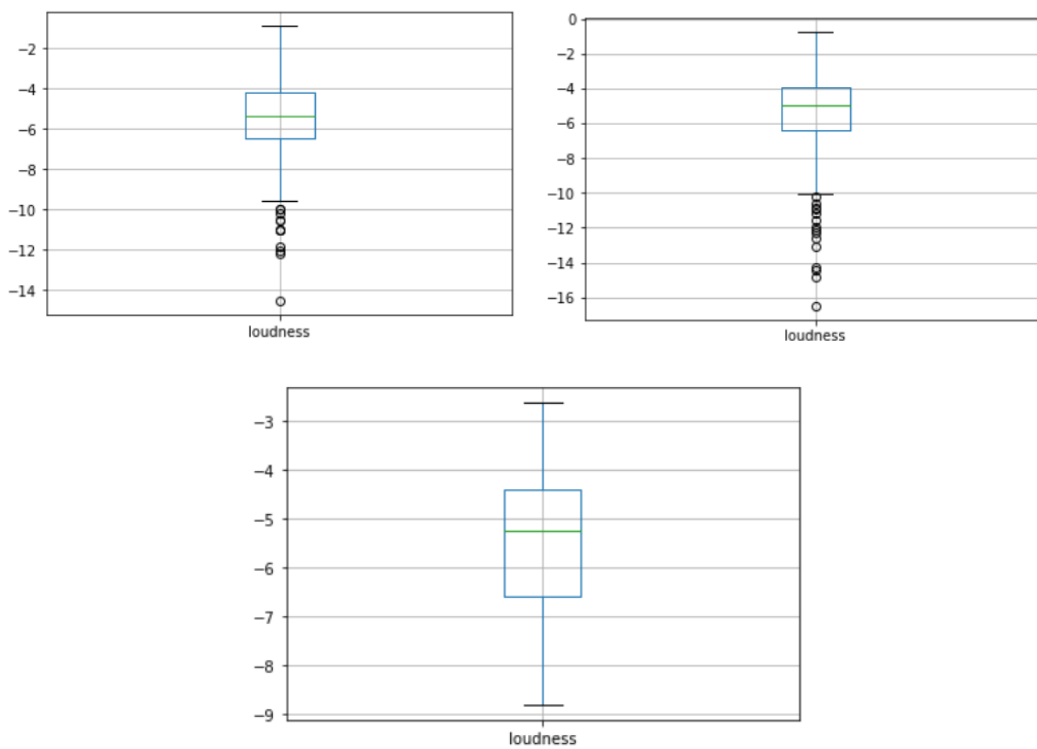


Fig. 7 - Loudness for dancepop, latino, and boyband

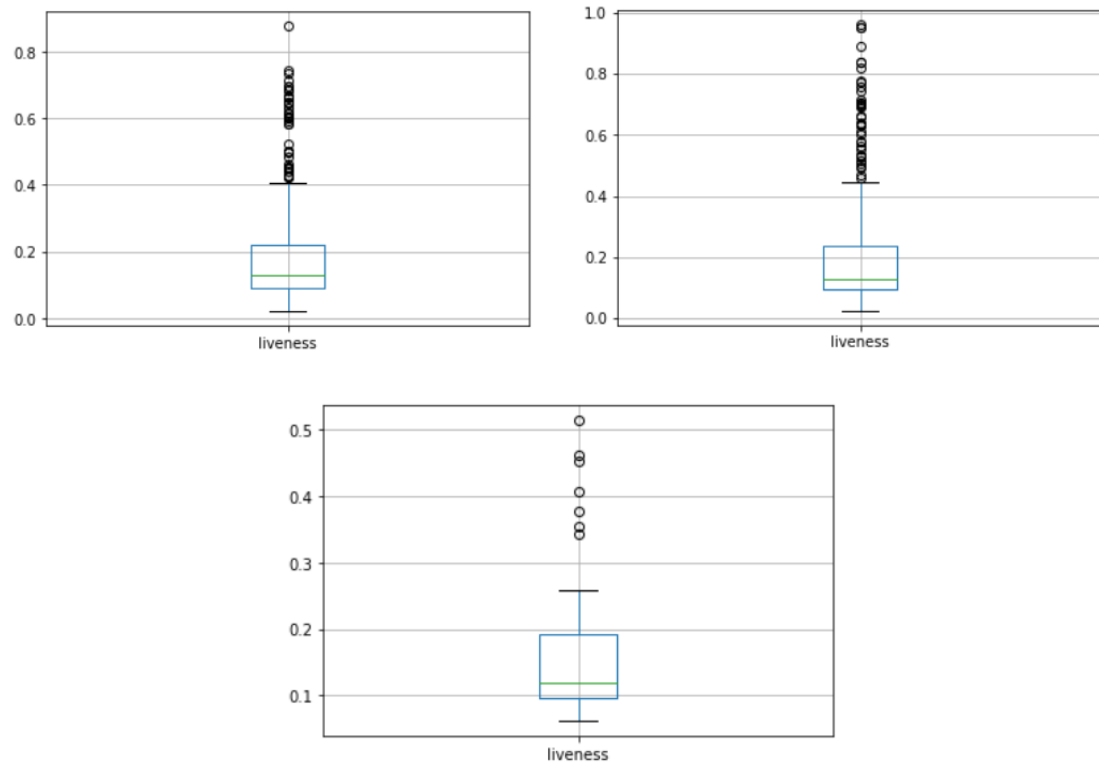


Fig. 8 - Liveness for dancepop, latino, and boyband

Recommender System

Now that we have our dataset, we can start building our recommendation system. After importing our dataset of about 10000 musics, we recover all the numerical columns in order to perform a clustering afterwards.

	title	artist	genre	demo	danceability	energy	key	loudness	mode	speechiness	...	instrumentalness	liveness	va
0	Vente Pa' Ca (feat. Maluma)	Ricky Martin	['dance pop', 'latin', 'latin pop', 'mexican p...]	https://p.scdn.co/mp3-preview/21e38a8983daf1c3...	0.663	0.920	11.0	-4.070	0.0	0.2260	...	0.000017	0.1010	
1	Reggaetón Lento (Bailemos)	CNCO	['boy band', 'latin', 'latin pop', 'reggaeton']	https://p.scdn.co/mp3-preview/ced5c17cadb43603...	0.761	0.838	4.0	-3.073	0.0	0.0502	...	0.000000	0.1760	
2	SAFARI	Tyler, The Creator	['hip hop', 'rap']	https://p.scdn.co/mp3-preview/46545a950dc150f8...	0.471	0.710	1.0	-6.462	0.0	0.2130	...	0.000002	0.3380	
3	Cuando Se Pone a Bailar	Rombai	['cumbia pop']	https://p.scdn.co/mp3-preview/02730a7c430c379f...	0.588	0.682	11.0	-7.169	0.0	0.1730	...	0.000027	0.0840	
4	Otra vez (feat. J Balvin)	Zion & Lennox	['latin', 'latin hip hop', 'reggaeton', 'trap ...]	https://p.scdn.co/mp3-preview/80f292da16e796af...	0.832	0.772	10.0	-5.429	1.0	0.1000	...	0.000486	0.4400	
...
9314	Sweet Scar	Weird Genius	['indonesian pop']	https://p.scdn.co/mp3-preview/4e8d36d748dfa5b8...	0.529	0.857	0.0	-1.183	1.0	0.0385	...	0.000000	0.0989	

Fig. 9 - Our dataset

num_only													
	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature
0	0.663	0.920	11.0	-4.070	0.0	0.2260	0.004310	0.000017	0.1010	0.533	99.935	259196.0	4.0
1	0.761	0.838	4.0	-3.073	0.0	0.0502	0.400000	0.000000	0.1760	0.710	93.974	222560.0	4.0
2	0.471	0.710	1.0	-6.462	0.0	0.2130	0.061800	0.000002	0.3380	0.599	80.453	177743.0	4.0
3	0.588	0.682	11.0	-7.169	0.0	0.1730	0.085100	0.000027	0.0840	0.937	205.643	195274.0	4.0
4	0.832	0.772	10.0	-5.429	1.0	0.1000	0.055900	0.000486	0.4400	0.704	96.016	209453.0	4.0
...

Fig. 10 - Numerical-only dataset

In order to normalize the data as much as possible, we use the standard scaler of sklearn, which removes the mean and divides by the variance.

Dimensional reduction

As we can see, our final dataset is composed of 13 columns. We will use the unsupervised learning clustering algorithm called k means. Kmeans allows us to divide the input data into k clusters, with k a hyper parameter of the model.

The drawback of k means is that it performs less well on high dimensional problems. It is therefore necessary to keep 2 dimensions at most. However, we are not going to select just 2 columns out of the 13, there would be too much loss of information.

Fortunately, there are already very powerful dimension reduction algorithms. We will use the PCA (Principal component analysis) algorithm which uses the singular value decomposition

```
|: from sklearn.decomposition import PCA

pca = PCA(n_components=2).fit(X)
pca_data = pca.transform(X)
X_2D = pd.DataFrame(pca_data)
```

```
|: X_2D
```

	0	1
0	-1.810710	-0.426535
1	-1.395301	-0.573977
2	-0.701171	-0.170683
3	-1.457461	-0.309121
4	-1.696674	-0.306883
...
9314	-0.941482	1.662510
9315	-0.280887	2.822782
9316	4.047180	-0.380125
9317	-0.245043	-0.423390
9318	-0.083256	1.661571

9319 rows × 2 columns

Fig. 11 - Applying the PCA algorithm

Finding the hyper parameter k

By definition, it is up to us to choose the value of a hyper parameter. Instead of randomly choosing the number of clusters, we will use the elbow method.

To do this, we train several models with a number of clusters ranging from 0 to 50. Then we plot the inertia of the set of models. The inertia is a metric that uses the distance of each point to its centroid to evaluate the performance of a model.

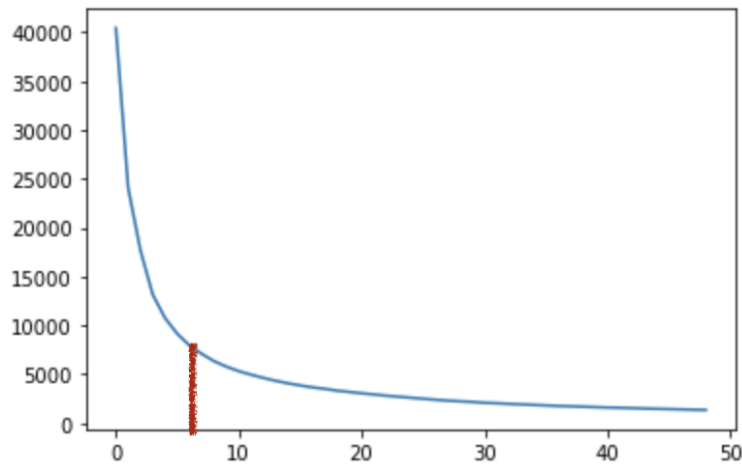


Fig 12 - Elbow method

As can be seen, a number of clusters equal to 7 seems to be relevant. This is what we will choose.

Clustering

With a number of clusters equal to 7, we can now train our model on our data and plot the results.

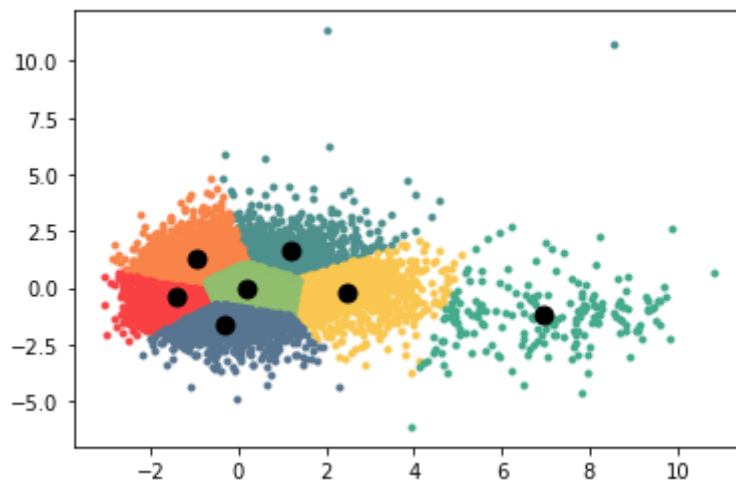


Fig. 13 - K means performing on our set

The result is quite satisfactory, there are few errors and the boundaries are consistent. The model seems to work perfectly for the moment.

Music recommendation

It's all well and good to have a powerful model but how to use it to correctly recommend titles to the user ? After implementing functions to retrieve features from the title of any song using the Spotify API, we can move on to the interesting part.

```
: get_features("bts dynamite")[1]
```

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature
0	0.746	0.765	6.0	-4.41	0.0	0.0993	0.0112	0.0	0.0936	0.737	114.044	199054.0	4.0

Fig. 14 - Example of song retrieved from Spotify API

As we can see below, we apply the same transformations to the features of our song as we did when training the model (scaling, dimension reduction).

We can then predict the cluster to which the music belongs and select all the music from the same cluster. From there, we use the Euclidean distance to compute the distance to each song in that cluster.

```
def predict(song, mode='normal'):
    song_data, features = get_features(song) # retrieving features from Spotify API

    X0 = scaler.transform(features) # Scaling
    X0_pca = pca.transform(X0) # Dimension reduction

    pred=kmeans.predict(X0_pca) # Cluster prediction

    # Selecting all the songs of the same cluster
    data_cluster = data[data['clusters']==pred[0]].copy().reset_index()
    data_cluster_2 = data_cluster.drop(labels=['clusters', 'index'], axis=1).copy()

    X_cluster = scaler.transform(data_cluster_2.select_dtypes(np.number).copy())

    # We use euclidean distance to classify the songs from the closest to the farthest from the original music.
    cdist_X0 = euclidean_distances(X0, X_cluster)
    songs = data_cluster.loc[np.argsort(cdist_X0)[0]]

    # If the original song is in the dataset, we remove it
    songs = songs.drop(songs[songs['title'] == song_data[0]].index)
```

Fig. 15 - Ordering songs by distance

Let's try our algorithm on a random song :

```
: song_data, songs = predict("Black Swan")
```

```
: from IPython.display import HTML
```

```
HTML(songs.to_html(render_links=True, escape=False))
```

	title	artist	demo	genre
1581	Abu Mecca	Marwan Pablo	https://p.scdn.co/mp3-preview/c07e5914e21dc2e3c8e46df0bc45e8096bedd03d?cid=64f00e0d6c1b4a4f9c37e331d24323c6	['egyptian hip hop', 'egyptian trap']
515	RGF Island	Fetty Wap	https://p.scdn.co/mp3-preview/d9c741fee97263caee8e48088fba0f46128111f?cid=64f00e0d6c1b4a4f9c37e331d24323c6	['new jersey rap', 'pop rap', 'rap', 'southern hip hop', 'trap']
2103	İki Cihan	Turac Berkay	https://p.scdn.co/mp3-preview/7a1c15c38498b336a4b3baae07d2a146176abee9?cid=64f00e0d6c1b4a4f9c37e331d24323c6	[]
1811	Come Milano	Ghali	https://p.scdn.co/mp3-preview/bba3b293b48bb11579d7736098b12da362435afd?cid=64f00e0d6c1b4a4f9c37e331d24323c6	['italian hip hop', 'italian pop', 'milan indie', 'rap tunisien']
866	25/8	Bad Bunny	https://p.scdn.co/mp3-preview/fd38e03acdb167a1320afb098549e5a00b2fc20?cid=64f00e0d6c1b4a4f9c37e331d24323c6	['latin', 'reggaeton', 'trap latino']
1264	Don't Call the Po-Po	Rose Villain	https://p.scdn.co/mp3-preview/281275dc8b75d08ebe091075b332f903cea1b214?cid=64f00e0d6c1b4a4f9c37e331d24323c6	['milan indie']
743	Sinä ansaitset kulta (Vain elämää kausi 7)	Cheek	https://p.scdn.co/mp3-preview/35e51a021e916af1df0598b86f6b088b829057a9?cid=64f00e0d6c1b4a4f9c37e331d24323c6	['finnish dance pop', 'finnish hip hop', 'finnish pop']
946	Shuffla	Samir & Viktor	https://p.scdn.co/mp3-preview/3fda220d433f74a521a3eae36eefb6c0b15a5f70?cid=64f00e0d6c1b4a4f9c37e331d24323c6	['europop', 'swedish pop']
1692	UnFazed (feat. The Weeknd)	Lil Uzi Vert	https://p.scdn.co/mp3-preview/9613900ed83c74ff9895b88b9426af564ec2a9a?cid=64f00e0d6c1b4a4f9c37e331d24323c6	['melodic rap', 'philly rap', 'rap', 'trap']
1415	H <3 T E L	Dardan	https://p.scdn.co/mp3-preview/10ee4fd6abcd769968f238722ffbc5f39435d464?cid=64f00e0d6c1b4a4f9c37e331d24323c6	['albanian hip hop', 'german hip hop']

Fig. 16 - Prediction on Black Swan by BTS

If you listen to the music, you realize that the mood and style is really very close to Black Swan. So, our model is functional and recommends music that resembles the input song.

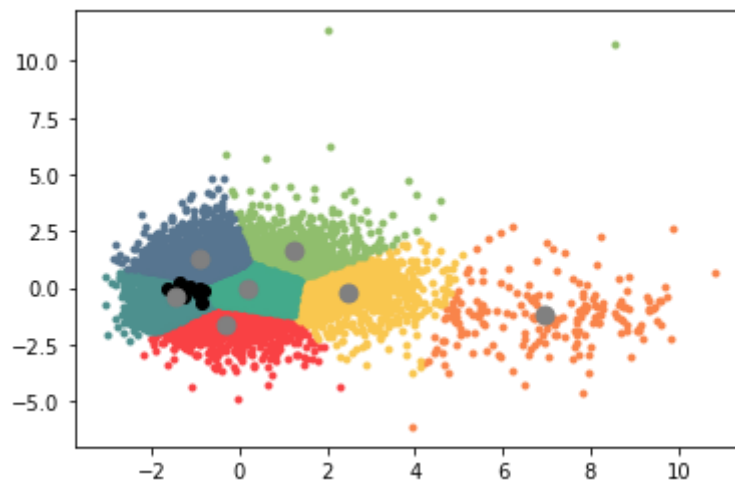


Fig. 17 - Predictions in black

Below, we can see in black that the predictions are coherent and close to each other: the sounds are musically very close according to the parameters given in input.

However, when we talk about music recommendations, we tend to think of music from similar artists and genres. For example, when you enter Black Swan from BTS, you would expect to get other BTS music or at least k-pop music.

Here, the genres of the predicted music are quite random. In order to improve this, we decided to add an option to improve the prediction by using the genres of the input music. Thus, only music that has at least one genre in common with the input music is kept.

```
if mode == 'genre': # if genre mode is activated
    final = []
    for row in songs.genre:
        new_row = False
        for el in ast.literal_eval(row): # we convert the string list row to a real python list
            if el in song_data[2]:
                new_row = True
        final.append(new_row) # True only If the sound has at least one genre in common with the input music
    songs = songs[final] # Selecting the titles with the correct genres
```

Fig. 18 - Genre enhanced search

```

: song_data, songs = predict("Black Swan", mode='genre')

: from IPython.display import HTML
HTML(songs.to_html(render_links=True, escape=False))

```

	title	artist	demo	genre
1169	Truth Hurts	Lizzo	https://p.scdn.co/mp3-preview/fac34b23f9462b22b3e63486adb8697edc9812?cid=64f00e0d6c1b4a4f9c37e331d24323c6	['dance pop', 'escape room', 'minnesota hip hop', 'pop', 'trap queen']
1413	Kings & Queens	Ava Max	https://p.scdn.co/mp3-preview/3d1b7c2dd0fa6109d374344a82e204e2df14da62?cid=64f00e0d6c1b4a4f9c37e331d24323c6	['dance pop', 'pop']
1787	BAZOOKA!	GWSN	https://p.scdn.co/mp3-preview/27e609d5cd86c36fca3b0205625eef18f026c3a?cid=64f00e0d6c1b4a4f9c37e331d24323c6	['k-pop']
675	Nuh Ready Nuh Ready (feat. PARTYNEXTDOOR)	Calvin Harris	https://p.scdn.co/mp3-preview/39487585fc7a5b183214e1c68f1fb8343cd3069b?cid=64f00e0d6c1b4a4f9c37e331d24323c6	['dance pop', 'edm', 'electro house', 'house', 'pop', 'pop dance', 'pop rap', 'progressive house', 'uk dance']
447	Straight up & Down	Bruno Mars	https://p.scdn.co/mp3-preview/0da92beb3cc0396e01a245d96ff35e31fe9d09af?cid=64f00e0d6c1b4a4f9c37e331d24323c6	['dance pop', 'pop']
1894	Stay Tonight	CHUNG HA	https://p.scdn.co/mp3-preview/87002a8ddf2843aee780edf5775c73f546e8a1fb?cid=64f00e0d6c1b4a4f9c37e331d24323c6	['k-pop']
92	Blow Your Mind (Mwah)	Dua Lipa	https://p.scdn.co/mp3-preview/01400c3daa98dafd23e7241df0f48e1fc48c8ef7?cid=64f00e0d6c1b4a4f9c37e331d24323c6	['dance pop', 'pop', 'uk pop']
828	Out of My Head (feat. Tove Lo)	Charli XCX	https://p.scdn.co/mp3-preview/92df8e6c3604cb9b179c3f942ae4b05214b7b91a?cid=64f00e0d6c1b4a4f9c37e331d24323c6	['art pop', 'candy pop', 'dance pop', 'electropop', 'experimental', 'hyperpop', 'indie pop', 'pop', 'synth pop']

Fig. 19 - Prediction on Black Swan by BTS with Genre mode activated

And, when we make a prediction on Black Swan with this new mode, we get even better results. Listening to them, you can hear that they are musically close. And in terms of genres and artists, we are also more consistent. We tried our model on a lot of songs and we even discovered new songs that we liked. As can be seen below, the predictions are consistent and close.

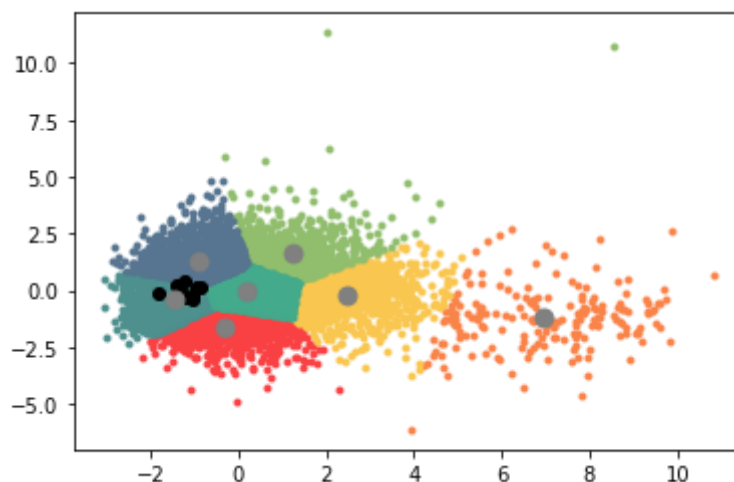
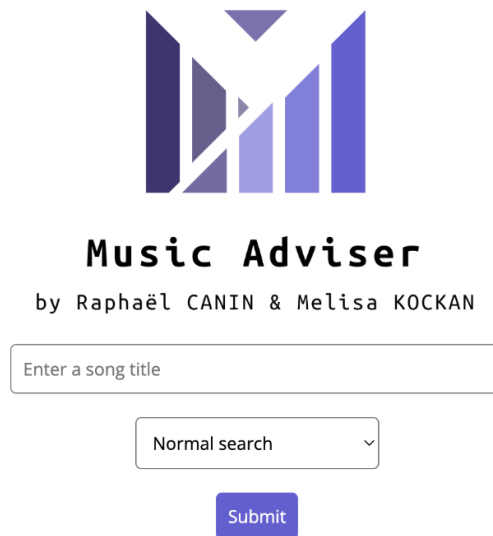


Fig. 20 - Predictions in black with genre mode

Web application

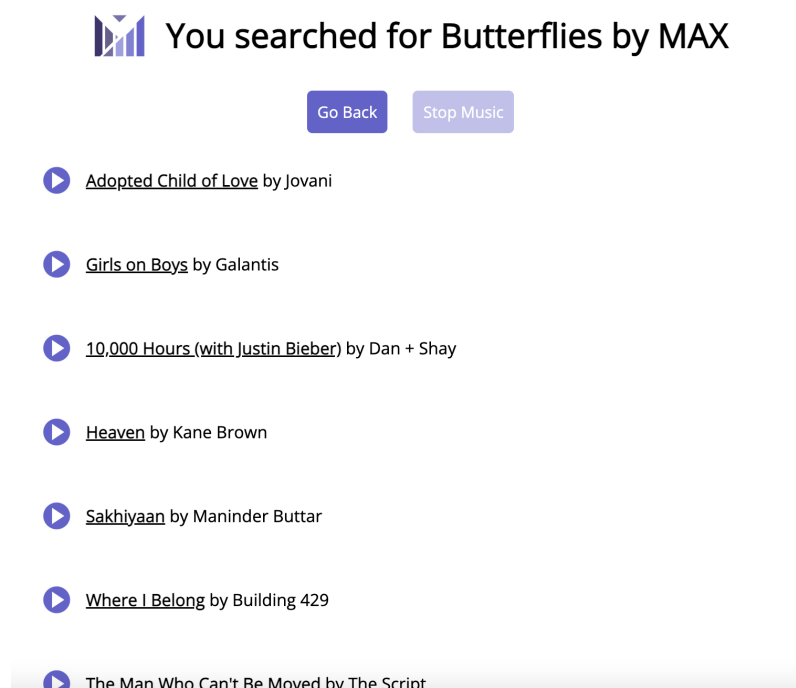
Now that our recommendation system is performing well, we will use Flask to create a user-friendly interface. To do this, we create two pages: a search page and a results page. We will use the post method to pass the form data from one page to another.

We have implemented at length two very aesthetic templates in HTML and CSS. We used CSS Grid and CSS Flex in order to have a regular layout of the elements. On the results page, we also used Javascript to play the audio demos of the sounds. We let you judge the result below:



The image shows a web form for 'Music Adviser'. At the top is a logo consisting of several vertical bars of varying heights in shades of purple and blue. Below the logo, the text 'Music Adviser' is centered in a bold, black, sans-serif font. Underneath that, in a smaller font, is 'by Raphaël CANIN & Melisa KOCKAN'. The form itself consists of a text input field with the placeholder 'Enter a song title', a dropdown menu currently showing 'Normal search', and a blue 'Submit' button.

Fig. 21 - Search page template



The image shows a web page for search results. At the top, there is a small version of the 'Music Adviser' logo followed by the text 'You searched for Butterflies by MAX'. Below this are two buttons: 'Go Back' (blue) and 'Stop Music' (light blue). A list of seven song recommendations follows, each with a play button icon and the song title and artist. The first six items are: 'Adopted Child of Love by Jovani', 'Girls on Boys by Galantis', '10,000 Hours (with Justin Bieber) by Dan + Shay', 'Heaven by Kane Brown', 'Sakhiyaan by Maninder Buttar', and 'Where I Belong by Building 429'. The seventh item, 'The Man Who Can't Be Moved by The Script', is partially obscured by a grey bar at the bottom of the list.

Fig. 22 - Results page

To make the prediction, we simply used the previous functions.

```
if request.method=='POST':
    try:
        mysong=request.form["song"].title()

        pred = predict(mysong, mode = request.form["mode"])
        res = pred[1].to_numpy()
        song = pred[0].to_numpy()[0]

        return render_template('tables.html', tables=res, song=song)
    except:
        return "An error has occurred"
```

Fig. 23 - Flask code

To display the arrays given by python, we used the Jinja2 syntax to write in the HTML templates.

```
{% for table in tables %}

<span><strong style='text-decoration:underline;'>{{ table[0] }}</strong> by {{ table[1] }}</span>

{% endfor %}
```

Fig. 24 - Jinja2 syntax

Deployment on AWS

In order to make our application accessible to everyone, we used an AWS EC2 virtual machine running Ubuntu. After connecting to the VM via SSH, we can transfer our files using SCP. We installed pip and the requirements.txt. Then we just need to add a rule to accept incoming TCP connections on port 5000 that Flask uses.

The screenshot displays the 'Résumé de l'instance pour' (Instance Summary) page for an EC2 instance named 'music_adviser'. The page is organized into a grid of information blocks. Key details include:

- Instance ID:** [redacted] (music_adviser)
- Public IPv4 Address:** 72.44.42.252 | [adresse ouverte](#)
- Instance State:** ✔ En cours d'exécution (Running)
- Private IPv4 Addresses:** [redacted]
- Public DNS:** [redacted].compute-1.amazonaws.com | [adresse ouverte](#)
- Private DNS:** [redacted].internal
- Elastic IP:** -
- Search for AWS Compute Optimizer:** [Inscrivez-vous à AWS Compute Optimizer pour obtenir des recommandations. | En savoir plus](#)
- Auto Scaling Group name:** -
- Role IAM:** -

Other visible details include the instance type 't2.micro', VPC ID, and Subnet ID.

Fig. 25 - EC2 VM on AWS

Fig. 26 - TCP rule to accept external traffic

Flask then launches without problem and we can access the site at the URL : <http://72.44.42.252:5000/>

```
[ubuntu@ip-172-31-16-57:~/FINAL$ python3 V1.py
Let's go
9319
* Serving Flask app 'V1' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on all addresses (0.0.0.0)
  WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://127.0.0.1:5000
* Running on http://172.31.16.57:5000 (Press CTRL+C to quit)
```

Fig. 27 - Flask logs

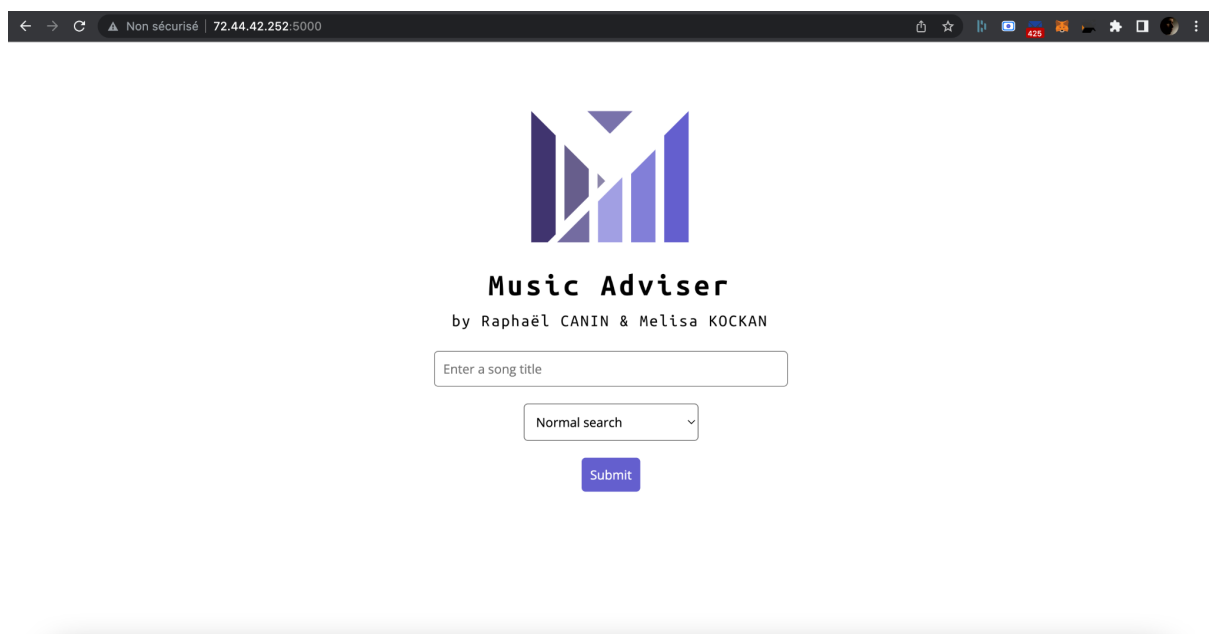


Fig. 28 - External access on our web application

We have demonstrated our application in a video attached to the report called demo.mp4. Thanks for reading, we hope it was informative.