DATA VISUALIZATION
PROCESS BOOK

E P F L - COM 480

SPOTIFY PLAYLIST EXPLORER

DAVID CIAN
PIERRE SCHUTZ
NATHAN KAMMOUN

INTRODUCTION

There are many ways one can listen to music. You can search for a specific track or listen to an album, but one of the most popular ways of doing so is by listening to a playlist, the digital descendant of the mixtape.

In fact, playlists are central to the Spotify experience, as all playlists created by Spotify users start out public and can be composed collaboratively. Even Spotify itself has been lauded for its recommendation engine, which can on top of recommending individual songs generate playlists for users.

Beneath the surface, playlists are a stranger beast than one might expect. It is quite common for a user-made playlist to include tracks from different albums, different artists, even different genres!

Furthermore, playlists represent a window into users' most minute idiosyncrasies. For this reason, playlists represent a perfect pocket of complexity to be explored visually. The goal of our visualizations is to allow Spotify users to gain insight into the patterns hiding in their playlists.

The user becomes a biologist exploring the anatomical details of their playlists, in all of their beautiful diversity. The tracks in my workout playlist come from many artists, but what if their danceability or their upbeat nature is what they have in common? What about my commute playlist, what do those songs have in common?

One could argue that every single playlist is bound together by some common factor hidden deep inside a user's preferences, perhaps the goal of eliciting a certain cocktail of feelings.

Our hope is that our visualization allows users to find that common factor.

THE DATA

The data comes directly from the Spotify API, which provides a wide range of data about songs, users, artists and playlist data. We could have used the large spotify dataset available on Kaggle, but we really wanted to provide a tool that let the user explore his own playlists. The API also enables you to get real-time data about the current top playlist.

Two authentication tokens for our application are generated: the first to access the top playlists, audio features, and related artists, the other to load the user's playlists. The data provided by the API is clean and ready-to-use.

The most interesting features from this API are called the Audio Features, and describe with details the musicality of a song with a few parameters. Figure 1 from our data exploration shows these features and compares them for two different playlists.

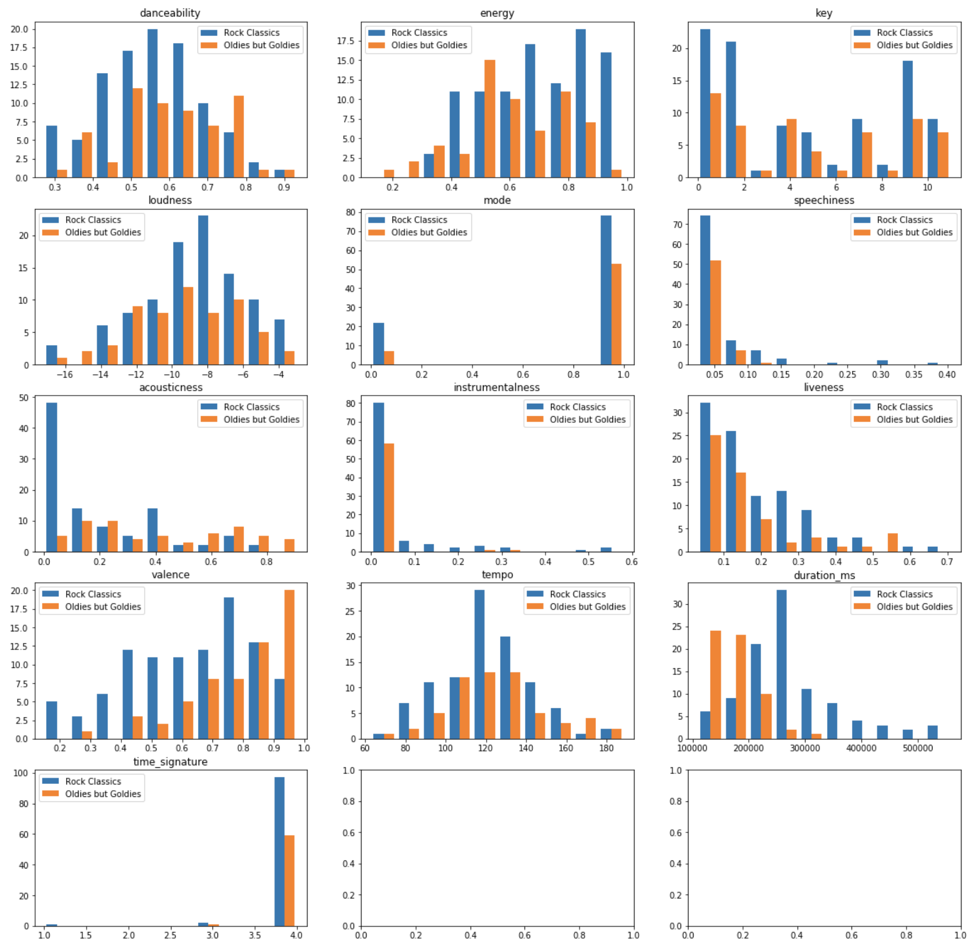


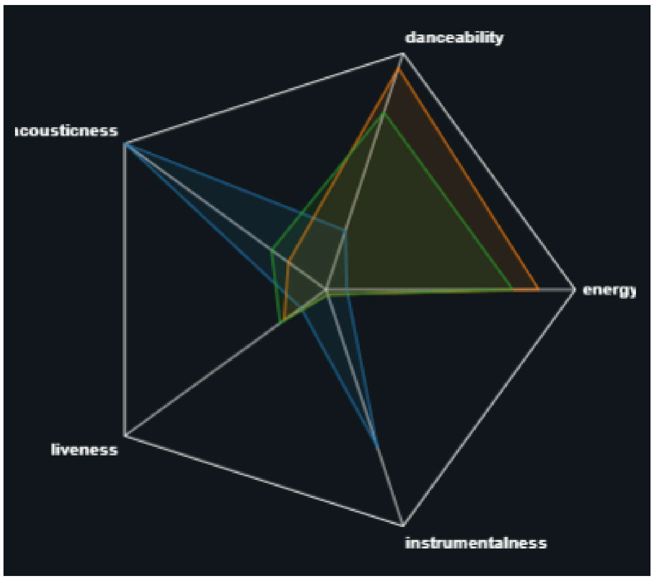
Figure 1. Rock Classics vs Oldies but Goodies audio feature

T H E   V I S U A L I Z A T I O N S

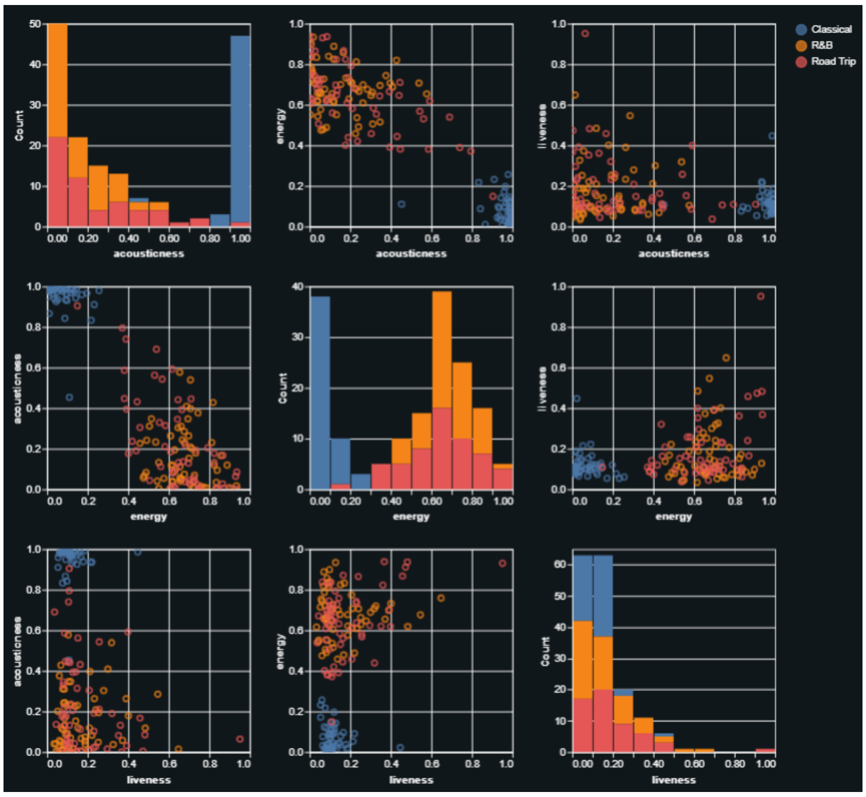
1. Audio features visualization

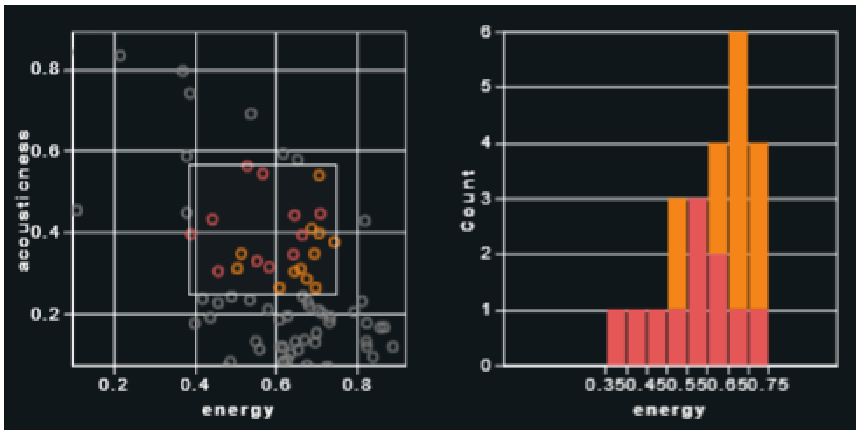
The goal of the first visualization is to look at playlists through the lens of their audio features. The user can select which features he wants to include in the analysis. Based on this, two visualizations are generated, following the principle of going from general to detailed information.

First, a radar chart displays a summary of playlist-wide aggregated features, allowing the user to compare them at a glance.



Next, a scatterplot matrix is displayed, with histograms on the diagonal (for each individual feature) and scatterplots everywhere else, for pairs of features. This displays detailed track-level data. To really drill down into it, you can zoom and pan the scatter plots, and even brush data in a linked manner across plots.





Design choices and challenges

Although we initially used d3.js directly, we decided our requirements were simple enough to move to Vega-Lite, which vastly sped up our development process. We had to fall back to Vega for the radar chart, since Vega-Lite doesn't support it. We are also aware of the shortcomings of radar charts, especially when dealing with unordered categories (such as here), but they remain a popular and engaging type of visualization for most audiences.

2. Graph visualization

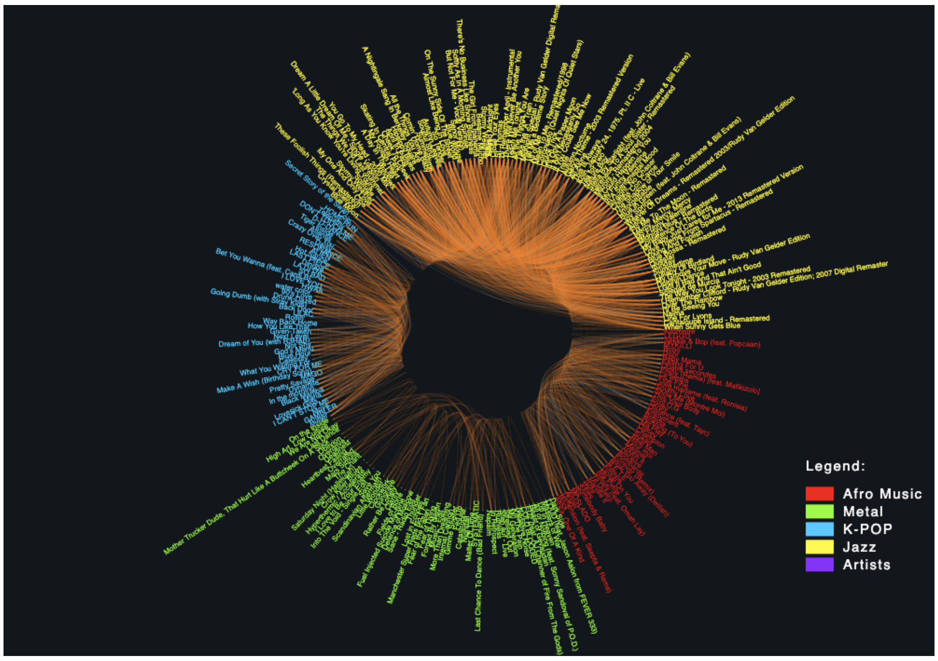


Figure 2. Disconnected playlists arcs visualisation

The second visualization presents the relations between the artists of each song in a playlist. The user can observe similarities between songs and playlists, and search for related songs or artists.

To find new tracks, a user can select his favorite playlist, and a playlist she would like to explore. Using the graph's "Arcs" visualization, she can search for songs from the related artists. In Figure 3, we can observe a number of songs in the "Rock" playlist that are related to the "Jump" track.

We can also visualize how homogeneous is a playlist. The "Jazz" playlist in Figure 3 is more interconnected than "Metal". We can also see how disconnected are the tracks from different playlists in Figure 2.

The visualization contains 3 main displays :

- Default visualisation: graph of tracks, artists, and related artists in circular shape.
- "Force links" visualization: Simulation of force where links between tracks and artists apply strength to group the linked nodes together.
- "Arcs" visualization: grach of links of length at most 3 between artists. The color of the links becomes darker if the link is longer.

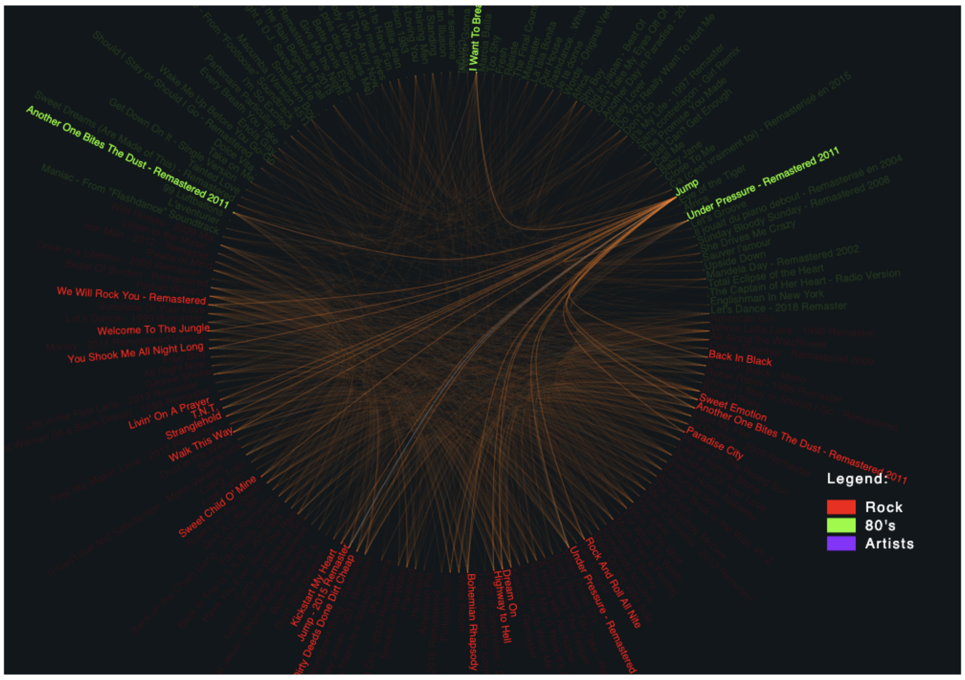


Figure 2 - Jump track links in "Rock" and "80's" playlists

Design choices and challenges

We started with a simple graph that displays the track nodes on a circle. We added the artist nodes, and links between tracks and artists. Then, the tracks and artists labels, the tooltip (displaying the name and type of a node), the legend, and the zoom. We finally added a third circle of nodes with related artists for our prototype version (Simpler version of the "Related Artists" setting).

The main issue we noticed with our initial approach was the exponential increase of the number of related artists (and relationships). This made the graph quickly hard to read. Another issue we faced was to place the artists as close as possible to their track while remaining on their respective circle. We tried to use the d3 force with constraints to remain on a circle but this was not successfully completed. Finally, we also had to bound the number of tracks and the maximum distance between artists in "Arcs" visualization to avoid long loading time.

To fix these issues and improve the user experience, we decided to propose multiple settings helping to visualize the graph differently ("Arcs", "Related Artists", "Force links"), and propose as an options the details that are heavy on the visualization ("Artists Names", "Legend", "Tooltip"). We finally added a display button to load the graph independently and improve the page loading time.

CONCLUSION

The visualizations implemented and the interactions with Spotify API via playlists data and widgets enable the user to visualize complex aspects of his tracks and playlist data in an interaction and precise manner. The various settings help him navigate in the data and highlight different aspects. Using these visualizations, a Spotify user can enhance his experience by improving or tuning his playlists using tracks, audio features and artists relationships.

We faced some challenges with the graph visualization scale, visibility, and computational time that we balanced with display options and multiple more or less detailed graphs.

PEERS ASSESSMENT

Nathan made the screencast, took care of the user interface of the page as well as the data importation from the Spotify servers through the API. He also helped create the first version of the scatter plot.

David designed the interactive scatterplot matrix visualization (brushing, linking, panning, zooming) and took care of the hosting (including not revealing the Spotify secret by using environments).

Pierre performed the exploratory data analysis, implemented the network graph visualization, and worked on refactoring the html components and Spotify API requests.