Explorify: A Personalized Interactive Visualization Tool for Spotify Listening History

Inna Ivanova
innai@cs.ubc.ca
University of British Columbia

Jonatan Engstad
jonatane@stud.ntnu.no
Norwegian University of Science and Technology

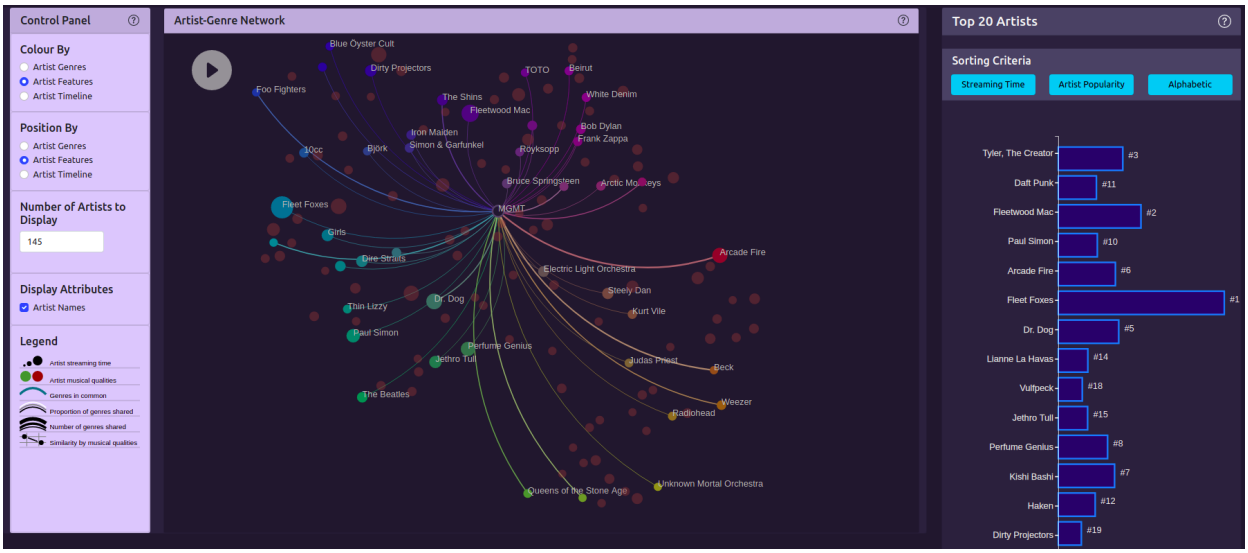

Fig. 1. Explorify: Opening page of the Explorify dashboard for personal data exploration

Abstract—There are many music streaming platforms but Spotify has dominated the market over the last few year and currently is the largest one with millions of users. Spotify collects huge amounts of data from its users which presents a great amount of research opportunities in the data science community. However, at the moment there are very limited attempts that have been published on the topic of visualizing Spotify streaming history. The common curiosity of people about there own listening habits is the user story which inspired this project. We implement an interactive dashboard for Spotify users with any level of expertise to perform exploratory, consumption, and analysis tasks on their personal Spotify streaming data. The interactive and playful nature of the dashboard aids Spotify users to explore their own music taste, find listening patterns and engage with their data.

Index Terms—Explorify, Spotify, dashboard, streaming history

1 Introduction

Spotify[1] is without a doubt the largest music streaming platform which servers millions of users on a daily basis. Spotify has revolutionized music discovery by collecting streaming data from its users and applying modern neural network algorithms to provide personalized music recommendations. In 2017 Spotify launched a marketing campaign called 'Spotify Wrapped' which lets Spotify users relive their musical discoveries and check their total time spent on the platform in the form of a short report card. The campaign has proved to be very successful in engaging more user and for longer times in addition to becoming one of the signature features of the platform. While the report cards are experiencing great design style they still have a major shortcoming in supplying Spotify users with a detailed and interactive view of their streaming history and listening patterns.

Luckily, Spotify users can obtain their streaming data through the platform if they wish to. In addition, Spotify provides information and audio features for each track and artist on the platform which is easily retrievable through their API[2]. Users are even able to request details including streaming history, playlists, top artists and others[3]. While there are a lot of visualization attempts for Spotify data on the Internet they are relatively simple and do not have interactive components.

With this gap in the field and easy access to streaming history data, we have embraced the opportunity to develop our own platform. We called it Explorify and it targets the current visualization limitations with Spotify streaming histories. Explorify requires no expert knowledge from its user to perform exploratory, consumption and analysis tasks on their own personal streaming data.

This paper introduces Explorify as a novel visualization platform for personal music streaming history provided by Spotify. It also illustrates how the built-in tools and visualisations allow users to understand their personal streaming history. By creating an interactive and customizable solution, we hope that users will be able to explore their streaming histories in a manner that they find interesting and rewarding and that they are able to discover patterns and trends that they are not aware of.

Explorify does not require programming knowledge and we believe that its simple and intuitive interface makes it easy to use. Through the main view and the three supporting views, users will be able to select

[1] https://spotify.com

[2] https://developer.spotify.com/documentation/web-api/

[3] https://support.spotify.com/us/article/gdpr-article-15-information/

and examine their streaming activity for specific days and times of day, as well as the relationships between the artists that they streamed and the genres of the artists. Through a control panel, users can select which attributes of the artists they wish to inspect and/or compare. Explorify's views are highly integrated so that complex results can by achieved through simple intuitive interactions. We believe that this level of integration has made the user experience more intuitive and interaction simpler for the end user.

Explorify makes use of standard visualization techniques, but also provides more sophisticated visualizations such as edgemaps [6], that encourage users to actively interact with their own data. While these sophisticated visualizations may be unfamiliar to the user, they are powerful enough that their choice is warranted. Interaction with Explorify cannot make any permanent changes to the user's data, and reloading the page will instantly reset all visualization back to their starting configurations. Thus the user will be able to safely familiarize themselves with Explorify and learn to exploit the interface to address their needs.

In section 2 we look at previously published visualisation of streaming history specifically and also of other connected multidimensional datasets. In section 3 we describe the reference datasets which have been used to develop the platform and their various parameters which are also available in a typical user data. In section 4 we define several typical tasks which can be accomplished using the dataset and associated questions to answer. In section 5 we explain the various elements of Explorify and how they are used, while in section 6 we discuss implementation architecture and details. Section 7 shows the historical timeline of the project. In section 8 we showcase Explorify with the reference dataset. Section 9 contains a discussion of the strengths and limitations of the platform, as well as next steps and a reflection on the project. Finally section 10 concludes the paper.

2 Related Work

2.1 Visualizing Music Streaming History

So far only a few publications have attempted to visualize listening history provided by the Spotify platform. Paradoxically, there exists more work on the topic of visualizing the listening history of Last.fm[4] users: a platform with very similar service offering to that of Spotify but only a fraction of users. There are some minor differences in the available data provided by the two platforms and the semantics remain largely the same. In discussing these works, we will usually make no distinction between which platform the work relies on since it generally does not affect the result.

The published works for this particular area can be divided into two groups based on their main goal. The goal of the first group is to help user discover new music. Papers targeting this goal seek to create tools that reveal new tracks, artists, and genres previously unknown but strongly believed to be enjoyable to the user. Usually, this is achieved by providing the user with tools to explore and traverse multiple networks of similar artists in search of undiscovered ones [4, 7]. Several tools that use this approach are available on the Internet. To name a few: LivePlasma[5], MusicRoamer[6], Music-graph[7], Music-map[8]. As examples we have included figures **??** and 3, which show the Liveplasma and Music-graph websites, respectively.

The goal of the second group is to reveal trends and facts about the user's listening history while also support the user in introspection about their own listening habits. The tools built with this goal in mind all visualize the user's listening history with a greater focus on the temporal aspect of the data than those of the first group. Furthermore, they support various means of interaction such as adjusting the granularity of the data being shown [1, 5, 10], or allowing for the comparison of two datasets [10]. Another tool, LastHistory, attempts to make this presentation more meaningful to the user by augmenting the listening history with contextual information from other data sources such as personal photos and calendar entries [2].

In a recently published work, Wirfs-Brock et al. explore how these goals can be combined in order to enhance music exploration and give the user deeper insight into their own listening history and listening patterns [11]. While the goal of this work was to research how a future voice assistant might use a persons listening history to offer better help, several participants expressed joy and engagement when presented with, and while exploring, summaries of their personal Spotify data.

While Explorify falls into the second group, we hope that our solution will facilitate discovery by piquing the interest of the user and deepening their insight into their own music preferences in a way that will inspire exploration.

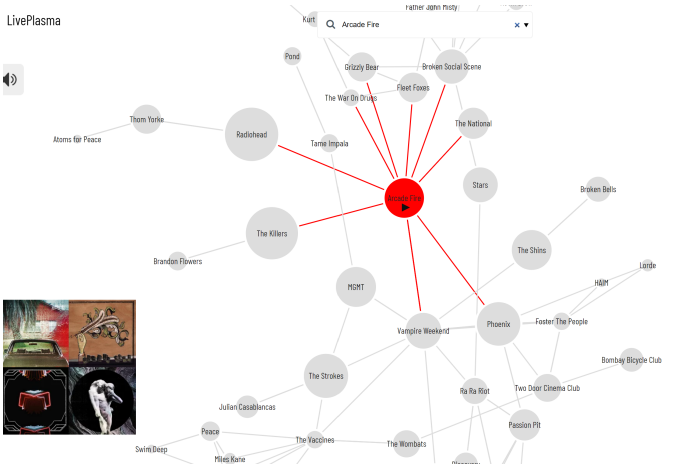

Fig. 2. The liveplasma website after a query has been made for the band "Arcade fire"

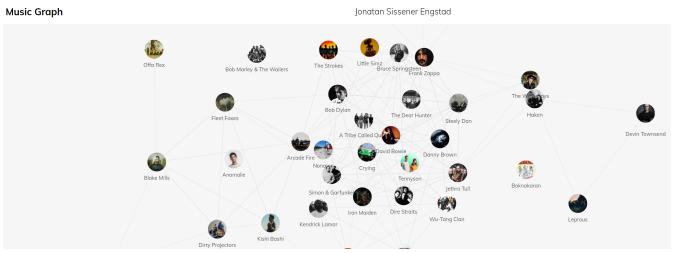

Fig. 3. Music-graph.xyz as it looks for one of the authors when logged in with their personal Spotify account.

2.2 Visualizing Explicit and Implicit Connections in Multidimensional Datasets

Dimensionality reduction (DR) is a common attribute aggregation technique for multidimensional datasets [8]. Generally speaking, DR allows significant reduction in the number of attributes needed to represent a dataset while still retaining semantic meaning. One of the most popular algorithms that achieve good DR results is t-Distributed Stochastic Neighbor Embedding (t-SNE). It is an unsupervised machine learning algorithm for nonlinear dimensionality reduction that is based on the distribution of the datapoints. The algorithm computes the probability distribution of pairs of datapoints in the original high dimensional space. Another probability distribution is then defined in a way that similar (closely related) objects have high probability of being displayed close to each other in the low dimensional map. This results in a low dimensional embedding that produces a similar distribution in the high dimensional data making it prefect for discovering clusters of similar artists and tracks solely based on their features.

[4]https://last.fm
[5]https://liveplasma.com/
[6]https://musicroamer.com/
[7]https://music-graph.xyz/
[8]https://music-map.com/

Explorify's main view is node-link diagram showing both explicit and implicit relations. The inspiration for this view comes from the work of Dörk et al.'s on EdgeMaps [6]. The edgemap idiom allows for the visualization of both explicit relations, like specific connections between items, and implicit relations, like items in a multidimensional dataset having a high similarity measure. This is achieved through the synthesis of spatialization techniques and graph-drawing techniques. In example used by Dörk et al., a dataset of philosophers is used. In the interest view, nodes (philosophers) are positioned by similarity, but also connected by edges if they inspired or were inspired by each other. In another view, the philosophers are positioned by their date of birth along a temporal axis, while the edges stay the same. The edgemap idiom also encourages interactive exploration by initially hiding all edges in the node-link diagram, and then displaying only those connected to the node highlighted by the user.

A common idiom for node-link network layouts is force-directed placement [8]. In a common variant of this idiom, nodes are assigned a repelling force, making them push away from each other so that they do not overlap. We utilize this technique to position nodes in Explorify in the cases where nodes end up overlapping in the DR result.

3 Dataset

There are a total of three streaming history datasets **SH_J, SH_D** and **SH_K** obtained from users **J, D** and **K** respectively, that were used during the development of Explorify. Each dataset is a timeseries dataset and comprises the streaming history for a user over one year period, in JSON format. Each item of the datasets contains:

- `trackName`: unique sting for each track, categorical data
- `artistName`: unique artist for the track, categorical data
- `msPlayed`: integer describing for how long the user has listened to the track in milliseconds, quantitative sequential data
- `endTime`: a timestamp describing the exact time at which the user stopped listening to the track, quantitative sequential data

A summary of dataset cardinalities is provided in Table 1.

	Total Tracks		Unique Tracks	Unique Artists
	Skipped	**Not Skipped**		
SH_J	1126	6887	4272	1598
SH_D	1919	3570	1428	375
SH_K	1862	1711	1451	818

Table 1. Datasets overview for each user **J, D** and **K**. Skipped tracks are ones that are played for less than 10 seconds.

The raw datasets are further extended with detailed information for each track and the corresponding artists and album by querying the Spotify API. Data processing libraries such as `numpy`, `pandas` and `sklearn` are used for the computations. Examples of additional information provided by the Spotify API include:

- Track audio features which are all quantitative sequential data with different ranges:
  1. danceability
  2. valence
  3. energy
  4. tempo
  5. loudness
  6. speechiness
  7. instrumentalness
  8. liveness
  9. acousticness
- Genres associated with the artist: categorical data;
- Related artists of an artist: Artists which listeners of the current artists are likely to enjoy as well;
- Popularity score of the artist: ordinal data in the range [0, 100];
- Number of followers the artist has: quantitative sequential data;

Dataset availability is both static and dynamic. The static part is the streaming history provided by the user. The dynamic part is the prepossessing of the raw user data and the additional features fetched through the Spotify API.

4 Data and Task Abstractions

While we do not assume our audience to have specialized knowledge about specific genre categories or audio features we expect them to be familiar with the concept of genres and that music artists are associated with them. In addition, we do not expect our audience to have expert knowledge in interpreting complex visualizaions, however, we anticipate that they will participate in tasks which target some form of exploratory data analysis. Some tasks include observing listening patterns and how they change over time, comparing top artists within single views, locating artists across multiple views, and browsing relationships between artists.

In total, we have identified 4 different use cases that Explorify enables: **Discovering Artist and Genre Relationships**, **Locating and Comparing Top Artists**, **Exploring Daily Listening Patterns** and **Exploring Single Day Streaming Activity**.

4.1 Discovering Artist and Genre Relationships

Spotify users often want to explore the relationships between the artists they listen to the most as well as the relationships between artists and genres in general. The user may want to know how genre connects and differentiates artists. In particular, they may wish to identify which genres connect two artists and how other genres belonging to only one artist make the artists differ. Similarly, we believe that the user will be interested in exploring how artists relate with respect to the auditive qualities of their music. We also believe that the user wishes to explore how their interest in genres and musical qualities has changed through the period for which they have been subscribed to Spotify. The aim of this task is to allow the users to explore these relationships in an interactive fashion. By allowing the user to select which attributes of the artists are used in the visualization, the user will be able to compare and contrast the features that they are interested in. In consuming the solution, the user will be able to spot patterns, outliers and abonrmalities in the data, thus achieving this task.

We also wish to facilitate the user discovering when they first started listening to an artist, and during which times they discovered more new artists than normal.

Both **genre** and **artist** are categorical attributes. The cardinality of the artist data is shown in Table 1. The cardinality of the genre attribute ranges between 249 and 1049 with the different datasets. Each artist has a number of associated genres, linking the two attributes. This makes it possible to construct networks where artists are vertexes and genres are edges or the other way around. In one dataset, SH_J, each artist has 4.5 genres on average, and each genre links 6.5 artists on average. This means that the degree of connectivity in the network is quite high. Because the number of artists in each dataset can be in the order of thousands, we will be performing a filtering reduction, limiting the visualization to displaying only the user's top 100 artists. This will help achieve a meaningful visualization while avoiding the hairball issue [8]. In terms of abstract definition, this task explores the connectivity, and lack thereof, of nodes within a network.

With each recorded stream, there is an associated date and time, marking the exact point at which the stream ended. From this data, we can produce a derived attribute, "firstStream", which denotes, for each artist, the first recorded instance of them being streamed by the user. Using this derived attribute, we can order the artists by the date at which the user first started listening to them (as far as we can tell using the data).

There are a total of 9 audio features attributes for each track (see section 3), meaning that each track is represented in a high dimensional space by its audio feature vector. All these features are quantitative sequential data, though their ranges and cardinalities differ. For our

analysis, all audio feature values were scaled to fit a standard bell curve centered around 0. The cardinally of the track data is described in Table 1. By averaging over the feature vectors of all tracks belonging to each artist, we were able to derive a feature vector describing the mean musical quallities of every artist in the dataset. By employing dimensionality reduction, we mapped these audio feature vectors into a 2D space. Additionally, by encoding the genres of an artist using one-hot-encoding, we obtain a vector in a —genres—-dimensional space for each artist. As with the audio features, we perform DR to map these feature vectors into 2D space.

Thus, we obtain three separate attributes to position every artist in the dataset by: along a timeline axis using "firstStream", in a 2D space using the mean aggregated audio feature vector and in a 2D space using the aggregated genre feature vector.

4.2 Locating and Comparing Top Artists

Spotify users often search their top artists through the Spotify platform. One use case we identify is locating and comparing top artist. This use case will fall into the task abstraction of consuming, identifying and comparing data. Current visualization that Spotify provides in this regard is very limited: it summarizes the top artist of the user but only over the last 4 weeks, 6 months or entire history. On the other hand, Explorify aims to provide its user not only with the most streamed artists for their entire history, but also allow them to look up their top artists for specific days. The data used for this use case consists of:

- `artistName`: categorical data. Cardinality is documented in Table 1.
- `msPlayed`: quantitative sequential data. Cardinality is same as total tracks number and is documented in Table 1.
- `endTime`: quantitative sequential data. Cardinality is same as total tracks number and is documented in Table 1.

To obtain the listening times for each artist we aggregate the listening times (`msPlayed` attribute) for their tracks overall and on specific dates (the `endTime` attribute). Since the cardinality of the artist over all datasets is quite high (in the hundreds to thousands) we will be performing filtering reduction – only the top 20 artists will be displayed at any time. To obtain the actual top 20 artist we will perform data reordering (sorting). In terms of abstract definition, this task explores the distribution (interaction time) for discrete data (the artists) over a period of time.

4.3 Exploring Daily Listening Pattern

The third use case scenario is concerned with the task of exploring the daily Spotify usage of a user. Users may wish to have a summarized view of their entire listening activity and identify patterns or abnormalities. The goal of the task is to let users explore their daily streaming history and discover interesting patterns. Questions this particular task aims to answer are:

- How music streaming routine changes over time?
- Which days observe the most listening hours?
- Is there spike usage for specific dates?

The data used for this task will be:

- `msPlayed`: quantitative sequential data. Cardinality is same as total tracks number and is documented in Table 1.
- `endTime`: quantitative sequential data. Cardinality is same as total tracks number and is documented in Table 1.

To retrieve the data needed for visualization of the task, the listening interaction on a particular day are summed up together (aggregated). This is a time series dataset so it's easy to aggregate tracks times by day. This results in a new attribute `streaming_time` that is ordered quantitative sequential data. The cardinality of the resulting dataset is 365 since the original datasets contain information over a year and we aggregate that information by dates.

4.4 Exploring Single Day Streaming Activity

While exploring daily listening patterns helps the user understand their overall activity on the Spotify platform, users may also want to examine their streaming sessions for specific days. Therefore, we identify another use case scenario where users wish to explore and navigate throughout daily activity sessions. The data used for this task will be:

- `msPlayed`: quantitative sequential data. Cardinality is same as total tracks number and is documented in Table 1.
- `endTime`: quantitative sequential data. Cardinality is same as total tracks number and is documented in Table 1.
- `artistName`: categorical data. Cardinality is documented in Table 1.

To retrieve the data needed for the this use case scenario, the listening interactions on a particular day are summed up together (aggregated). However, the aggregation happens in a manner which groups together consecutive streaming sessions where the user has listened to the same artist.

5 Solution

The Explorify platform consists of two parts: the preprocessing pipeline and the Explorify dashboard. The Explorify dashboard contains all of the visualizations in our solution and it is the focus in this section.

The Explorify dashboard consists of 4 views, with the centerpiece, or the main view, being the artist-genre network. The artist-genre network is intended to be the main vehicle for exploration, while the remaining views complement it by making available supplementary information and interactions. To the right of the artist-genre network is the top artists bar chart, while below the artist-genre network, we have placed the calendar heatmap which shows streaming activity across the recorded period with daily granularity. While there are four views in total, only three are initially visible to the user. Selecting a day in the heatmap displays the fourth view below the calendar heatmap which describes the streaming timeline fr the selected day.

5.1 Artist-Genre Network

The requirements of use case 1 (see section 4.1) are very broad so we considered many visualizations for it. To satisfy them, we prioritized being able to display both artists and genres at once, while also highlighting how the two are connected. For this, we settled on a node-link network visualization [8], where nodes represent artists and edges represent the genres that the incident nodes have in common (no edge exist if the intersection is the empty set). Force-directed placement [8] was used to position the nodes, with nodes being given a repelling force and edges constituting an attracting force. This visualization turned out insufficient on its own: the many shared genres resulted in massive hairballs, and interactivity and support of exploration was limited. We decided to expand on the node-link network, settling on the edgemap idiom [6].

5.1.1 The Artist-Genre Edgemap

The final edgemap-implementation consists of two parts: the main edgemap view, and a control panel for customizing the edgemap parameters. In addition, the edgemap implementation features several connections to the remaining views. This is explained in more detail in the sections of those respective views. The control panel controls four variables relating to the edgemap: which attribute to position the nodes by, which attribute to color the nodes by, how many nodes to display (upper limit) and whether to display node labels. The edgemap view, as well as the edgemap control panel (on the left) can be seen Figure 1.

5.1.2 Nodes and Transitions

Explorify offers three options regarding to positioning of the nodes based on three different attributes we define as: the derived "firstStream" attribute (which we will call the "timeline position"), the result of DR over mean audio features ("feature position") and the result of DR

over artist genres (”genre position”). If the user chooses the position the nodes by their timeline position, a horizontal timeline axis will appear in the middle of the edgemap's viewport, making it easy to tell for the user at which time they first listened to an artist. Upon the user changing the selected position attribute in the control panel, an animated transition will play. The transition shows the nodes moving from their previous positions to their new positions, as dictated by the selected position attribute. This helps the user maintain a sense of context, and relieves them of the cognitive load of having to re-locate the nodes they were tracking [8].

5.1.3 A Partial Force-Directed Layout

Due to the limited space offered by the edgemap viewport, as well as the limitations and inherent properties of the data as well as the DR techniques applied, some nodes end up overlapping in the edgemap. This can cause partial or complete occlusion of nodes, and the visualizations interactivity and clarity suffers as a result. To avoid this issue, force-directed placement is employed, with nodes repelling each other when in close proximity. This prevents the drawn nodes from overlapping, though it does come at some cost. First, the proximity of nodes with similarity gets limited. Two nodes with very high similarity will appear with some minimum distance between them, as enforced by the simulation. The similarity between these two nodes and another pair whose nodes are positioned naturally at this distance due to being less similar will appear identical to user. This effect if worsened by clusters of very similar nodes, where the distance across the cluster may be much greater than what the similarity of the nodes would entail. Another drawback to the force-directed placement solution is that it causes some visual artifacts each time the the user changes the positional attribute of the edgemap. This happens due to the dynamic nature of the solution, and the fact that transitions use as destination the coordinates given by the dimensionality reduction. At the end of a transition, nodes may appear overlapping each other, before the simulation kicks in and re-positions them. While this does cause visual clutter, we also believe that it helps the user understand the nature of the positions and the simulation since the nodes are first displayed in their original position, and only after the simulation moves them to their final positions. We feel that the gains from the visual clarity and interactivity would make up for its drawbacks. In the case of the timeline axis, the force-directed layout simulation will only place nodes further up or down along the vertical axis, while keeping nodes' horizontal position constant. This allows us to avoid nodes overlapping, while still keeping nodes at their correct position with respect to the timeline axis.

5.1.4 Color

In [6], nodes are colored by their polar coordinates in edgemap: saturation is proportional to the node's distance from the centre, while the node's hue is given by its angle relative to the centre. Whereas the referenced edgemap implementation uses static colors for the nodes, Explorify allows the user to customize which attribute to color the nodes by. This makes the nodes' position and color independent from each other, and allows the user to juxtapose two attributes, by assigning node color the first attribute and and node position to the second. This is demonstrated in figure Fig. 4. As Explorify only shows outgoing edges, all edges are given the color of their target node.

5.1.5 Edges

In the work of [6], edges are kept hidden until a selection occurs, at which point all edges incident to the highlighted nodes are shown. Our implementation works in a very similar fashion, with the exception that only outgoing edges of the selected node are shown. Edges in the Explorify's edgemap are directed. An edge represents the genres shared by two artists, a relation which is commutative. Therefore all edges come in pairs, if an edge from node1 to node2 exists, an edge from node2 to node1 will also exist. The difference between these edges lies in the degree of overlap between the genres of the two artists, a relation which is not necessarily commutative. As we experimented with using non-commutative measures of overlap, it was decided to keep edges directed and separate. The appearance of edges in the our edgemap implementation is influenced by two run-time calculated attributes: proportion of genre overlap, and size of genre overlap. The higher the number of genres two artists share, the thicker the edges connecting them will be. In addition, lightness channel of the edge encodes the proportion of genres that the two artists share, with edges that connect artists with the exact same set of genres being completely white.

5.1.6 Highlighting

Explorify's edgemap nodes support direct interactions: Upon hovering the cursor over a node, the a label showing the artist's name will be superimposed on top of the node. When the user selects a node, the node, along with its neighbors, (connected by outgoing edges) is highlighted. This is shown in Fig. 6. When a node is selected it's fill color becomes transparent, and it is given a thick border, making it visually distinct from every other node. The selected node's neighbors keep their colors as they were before any node was selected. The remaining nodes are all given the same anonymous-looking color so they blend in more with the background. They also have their labels hidden, if they were being displayed. All edges originating at the selected node are made visible. The visualization will still respond to hovering as before. The user may click on the background to cancel the highlighting, or on any other node to change which node (and set of neighbors) is highlighted. The user may also hover over visible edges to have a list of all genres in the edge be superimposed on top the visualization.

During the transition that plays when node positional attribute is changed by the user, as well as the ensuing re-positioning by the force-directed placement simulation, edges and labels are hidden from view to reduce visual clutter. Node color highlights are kept the same. This again aids the user in tracking the changes in position, often heavily reducing the number of nodes that the user must keep track of during the transition. Since the nodes stay highlighted after the transition ends, the user can quickly and easily identify their new positions.

5.1.7 Labelling

By default, labels showing their respective artists' names are superimposed on top of the nodes. To avoid labels overlapping and ruining readability, a simplified version of the approach used by [9] is employed. The same approach is used to add labels to edges. Edges are labeled with the genres which their incident artists share, as shown in Fig. 7. Due to the edge labels running along curved paths, most of which are oblique, the edge labels often end up with huge bounding boxes. The result is that the edgemap ends up being very conservative with how many edge labels are made visible, so edge labels are not very prominent in the resulting visualization.

5.1.8 Other features

The edgemap encodes the how much time the user spent streaming the artists' music through the size channel of nodes. The size of a node is linearly proportional to the amount of time they were streamed, on a scale from the least streamed artist to the most streamed artist. Using the control panel, the user may also control the maximum number of nodes drawn by the edgemap. As the user adjusts this number, nodes will pop into and out of view depending on if the number is being adjusted up or down respectively. If a node is highlighted during this process, added nodes and edges will be styled according to the rules of highlighting explained earlier. Which set of artist the nodes are drawn from depends on user input to the calendar heatmap view, but by default it is the set of all artists for which we were able to retrieve data. Because adding and removing nodes is relatively expensive operation, the input is debounced and the sum of changes is applied after a small amount of time has passed. If the user decides to add 20 nodes by repeatedly adding one node at a time in very rapid succession, this technique allows us to add all 20 nodes to the edgemap at once instead of redrawing and re-simulating the graph 20 separate times.

A final feature of the edgemap is what we call the ”autoplay” feature. This feature is simply the edgemap automatically cycling through a

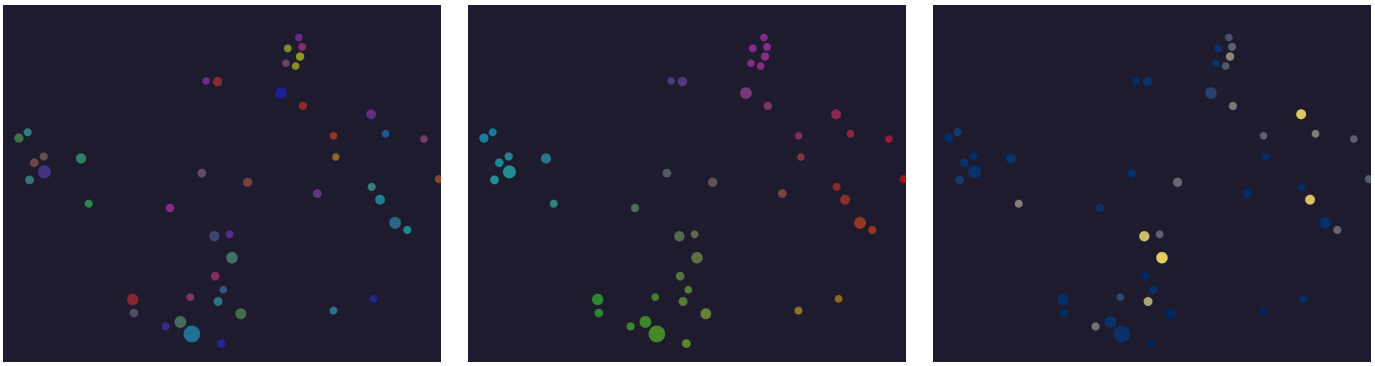

Fig. 4. The edgemap with nodes positioned by genre, but with color attribute varying. The color attributes, from left to right, are: audio feature, genre, timeline.

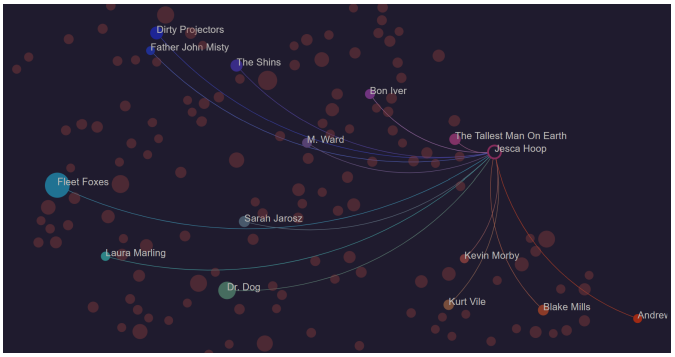

Fig. 5. The edgemap with the artist "Jesca Hoop" highlighted and labels enabled.

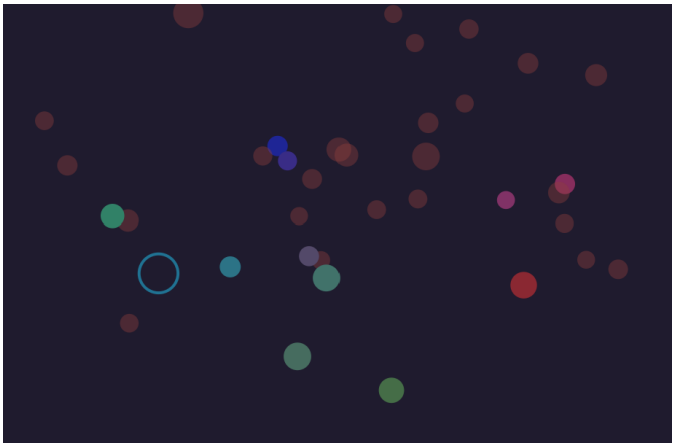

Fig. 6. Edgemap: highlighted nodes in the middle of a transition

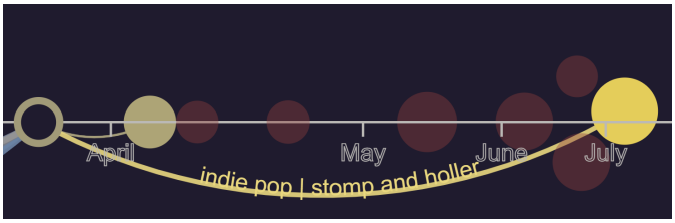

Fig. 7. Two edges in the edgemap, one labelled with the connect artists' shared genres, the other missing labels.

number of states with a short pause in between each change. These states are simply the product of all the options controllable from the control panel, with the number option being either 50, 100 or 150. This feature can be enabled as disabled by the user at any time by pressing the play or pause button (respectively) in the upper left corner of the edgemap view. As the state of the edgemap changes, the edgemap control panel UI widgets change as well, reflecting the new configuration of the edgemap. While this feature offers little additional functionality, we have found it useful in demonstrating the the capabilities of Explorify. We hope that it will also help users the possibilities that the edgemap control panels offers by demonstrating the range of possible configurations.

5.1.9 Top Artists Bar Chart Connectivity

If the artist selected (to be highlighted) by the user is also present in the bar chart, the bar of corresponding to that artist will be highlighted as well. If the user cancels the highlighting, the corresponding bar will also lose its highlighting.

5.2 Locating and Comparing Top Artists: Interactive Bar Chart

To address the task in section 5.2 we are considering an interactive bar chart shown in Fig. 8. We use a horizontal bar chart to fit the alignment of the rest of the views in the Explorify dashboard. We use the position on common scale channel to display the data and the hue channel to highlight specific artist on the bar chart. The bar chart displays the top twenty artists of the user over the period of their entire hist ory or for specific days. This visualization results in a total of twenty bars maximum displayed at once (some days may have less than twenty artist). We complement the bar chart visualization by allowing three distinct actions to be performed on it: sorting, hover and selection.

There are three buttons positioned on top of the bar chart that define the sorting criteria of the displayed artists. The user can perform click operation on each of the buttons which will result in a transition of the barchart. In addition, the user can hover over individual bars which will display a pop-up display with detailed information about the listening times for the artist. The user can further select an artist of interest by clicking on the bar item which in turn will highlight the selected artist by changing the bar colour.

Since the barchart is interconnected to the Artist-Genre Network a selection operation on the bar chart will result in highlighting the node of the selected artist in the Artist-Genre Network visualization (see section 5.1 for details). The bar chart is further interconnected to the Calendar Heatmap: if the user wish to explore their most popular artists for a particular day they can perform a selection operation in the heatmap which will derive a new view of the bar chart with the relevant data.

5.3 Exploring Daily Listening Pattern: Calendar Heatmap

The interactive calendar heatmap shown in Fig. 9 is the solution to the the task described in section 4.3. The visualization for the task of

Fig. 8. All three different views of the interactive barchart. On the right the barchart is sorted by the streaming time of each artist. In the middle, the barchart is sorted by the artitst popularity scores. In addition, a selection action has been performed which highlighted (changed the colour of) the bar item for the selected artist. On the left, the barchart is sorted by the names of the displayed artists. In addition, the hover action has been displayed which triggers a pop-up display with details about the artist.

exploring daily listening patterns was inspired by the famous calendar heatmap which records the activity rates on Github. We have chosen this particular visualization due to its simple and easy-to-interpret design. There are a total of 7 rows in the heatmap and each of them is labeled for the corresponding day of the week. The columns of the heatmap suggest the week number in a year (a total of 52 weeks). However, instead of displaying the week number we have decided to display a month label since it is more descriptive. The dataset used for this visualization contain information for each day over the course of a year which results in a total of 365 cells displayed in the calendar heatmap.

Each cell value holds data for the daily activity time on the Spotify platform converted into hours, minutes and seconds. We use the colour and saturation channels to encode the listening times for each day. A higher saturation value suggest higher interaction times. Since the values for each cell are quantitative sequential data with positive range, a single colour sequential color map is used for the mapping.

By aligning the streaming interactions in this manner the visualization prompts the users to discover and observe patterns over months, weeks or days. To aid the visualization and the exploratory tasks we implement additional functionality features that support user interactions. The user can hover over the day cells which triggers a pop-up display showing a short summarization of the listening times for that day. In addition, the user can perform selection actions by clicking on individual day cells. This results in highlighting the selected day cell by changing its colors, and additionally it triggers multiple changes in the rest of the visualizations. When a day is selected, the interactive bar chart and the artist-genre network are filtered to display only relevant information for that day: the bar chart displays top artists for the selected day; the artist-genre network displays only artists played on the selected day. Finally, additional gantt chart is displayed below the heatmap map which provides a detailed overview of the streaming sessions along with the times and artists on the selected day.

5.4 Exploring Daily Streaming Patterns: Gantt Chart

The visualization that provides solution to the task of exploring daily streamline patterns is displayed in Fig. 10. We have chosen a gantt chart because of its ability to provide an overview for the different activity sessions throughout a day. We use the position on common scale channel to display the different sessions throughout the day and the colour channel to differentiate between different sessions and artists. The gantt chart is using the colour scale from the artist-genre network which allows users to recognize patterns in their listening history. For example, consecutive sessions of similar colours could aid the user to identify a preference for specific genre or artist type for that date. In addition, the gantt chart supports the task of querying the displayed data by implementing user interaction in the form of pop-up display with session summarization when the user hovers over single session items.

6 Implementation

The Explorify platform is implemented in Python and JavaScript with D3 and React . Python is mainly used for data processing while

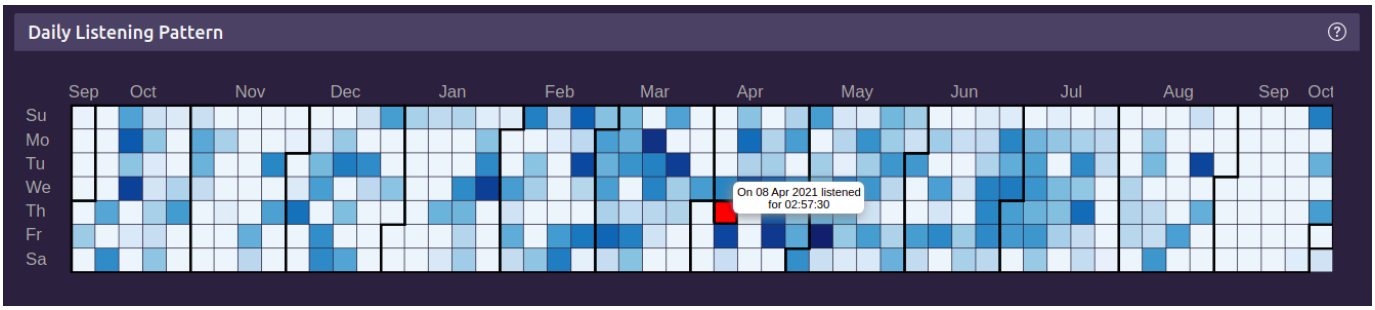

Fig. 9. Calendar Heatmap: the final implementation of the calendar heatmap. The heatmap supports hovering and selection for individual dates.

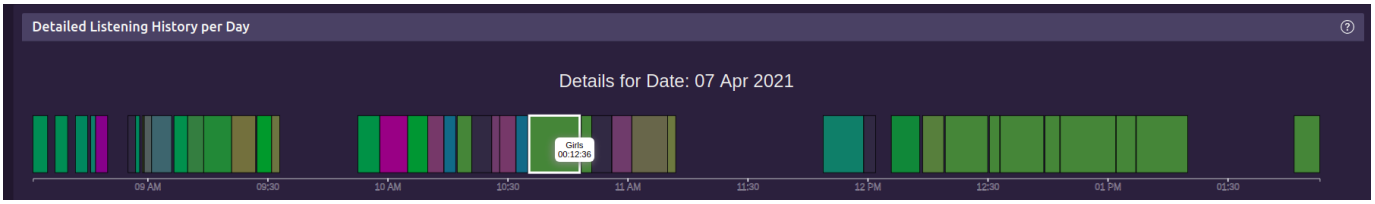

Fig. 10. Gantt chart: the final implementation of the gantt chart view. Individual items on the chart describe different sessions. In addition, the gantt chart supports hovering over individual sessions which in turn displays details about the session.

JavaScript with D3 is used for creating the visualizations.

Explorify is implemented in two parts: a data processing pipeline written in python, and the Explorify application itself, which is written in a mix of Javascript and Typescript using React components for structuring the code and D3 for creating the visualizations. Explorify was built using Parcel[9].

6.1 Preprocessing

Explorify's preprocessing pipeline is built in python. The python libraries pandas and numpy are used to ease and speed up the data processing. The preprocessing pipeline mainly consists of calls to the Spotify web API to retrieve additional data to fill out the user's streaming history as well as joining the retrieved data to the existing data. Some additional processing is performed to create derived attributes and groupings of data items to reduce the need for data processing inside the Explorify application. In addition, some dimensionality reduction is performed in order to calculate the artist positions used in some configurations of the edgemap. For the DR and some of the more advanced calculations, the python libraries Scikit-learn[10] and Scipy[11] were used. To ease the process of authenticating against the Spotify API, and partly to ease the use of the API, the python library Spotipy[12] was used.

The preprocessing pipeline has remained fairly constant throughout the process of developing Explorify, having been developed early in the process and being fairly robust to the change of plans that happened about halfway through development.

6.2 Explorify

The initial plans for Explorify were very different from the final product. As development began, focus was placed on developing a streamgraph [3] and a calendar heatmap (similar to the one found in the final iteration of Explorify). After receiving feedback on our plans, and brainstorming for new ideas, we decided on creating an node-link network using force-directed placement. This iteration of the artist-genre network can be seen in Fig. 11. As development of this view was underway, we encountered the edgemap idiom and were struck by how fitting it was for the task we were trying to solve. After some consultation and additional feedback, we decided to implement an edgemap with several connected supporting views.

While initial development was all done in plain javascript and D3, the decision to implement an edgemap also came with the realization that we would be creating a dashboard with several connected views. We chose to use React to divide the project into independent, but easily connected components. React also carries the advantage of having plugins for it that make it easy to style a webpage, as well as one member of the team having some experience with it.

As implementing the edgemap turned out to be quite a large project with many different objects and types being used throughout, it was decided that it would be implemented using Typescript. This turned out to be very beneficial to the implementation, as the project kept growing in scope. The supporting views were all written in plain javascript. Using React made it simple to compartmentalize code and divide the views into separate files. This helped make the process of connecting the views together quite simple.

7 Execution Timeline

A breakdown of the execution timeline and distribution of work between the group members is summarized in section 10.

8 Results

The final version of the Explorify platform is shown in Fig. 12. There are a total of 4 different interactive and interconnected views corresponding to the solutions described in section 5. Displayed in the top left corner is the main view of the Explorify dashboard, namely the 'Artist-Genre Network'. The view contains additional 'Control Panel' which allows the user to control the positioning, the colouring, and the number of nodes in the 'Artist-Genre Network' edgemap. To the right of the 'Artist-Genre Network' is displayed the 'Top 20 Artist' view containing the interactive bar chart and the sorting criteria buttons explained in section 5.2. Another supporting view 'Daily Listening Pattern' is displayed below main view. At the very bottom is positioned the detailed 'Detailed Listening History per Day' view and it is displayed only when a day is selected in the calendar heatmap matrix.

9 Discussion

In this chapter we discuss the strengths of our final solution and underlying implementation, and we identify the present limitations and future directions of the project. We wrap up with project reflections and some considerations for future work.

[9] https://parceljs.org/
[10] https://scikit-learn.org/stable/
[11] https://scipy.org/
[12] https://spotipy.readthedocs.io/en/2.19.0/

Fig. 11. An early iteration of the artist-genre network, before we decided to make it more like an edgemap.

9.1 Strengths

We are very happy with having created something relatively unique – we have not been able to identify any other tools that generate visualizations similar to that of Explorify for streaming history data. Looking at previous literature, we believe there are unique contributions and ideas contained in this paper, that may inspire others in their future work on visualizing this type of data.

Another strength of Explorify is that it in our opinion is quite pretty. While we created Explorify with functionality foremost in our minds, we were also able to style it in a fashion that we find quite pleasing. As good visualizations have both great function and great form [8], we believe that this is only a good thing, and we believe that this will strengthen the enjoyment that users get out of using Explorify.

Finally, we believe that a great strength of Explorify is the amount of major and minor features we have been able to cram into it, without it feeling over-encumbered and heavy to use. The various views are semantically coherent and their functionality complementary. During personal testing, we have found that exploring our own streaming histories using Explorify has been fun and engaging, without ever feeling confused or that too many options were available to us. Explorify offers good interactivity and we are happy with the connectedness of its views. Features like allowing artists to be selected and deselected in both the the bar chart and the edgemap are quite simple, but we feel they add a lot to the general usability and ”comfortability” of the tool.

9.2 Technical Challenges

One of the major challenges with implementing the edgemap was encountered when making the switch from plain Javascript to Typescript. This was due to the philosophy of D3's interface being incompatible with Typescript. In specific, certain D3 functions, like the ones for creating a force-directed layout simulation mutate the nodes and links that they receive into what Typescript views as completely different types, taking the received object and adding new fields and mutating, in-place, the types existing ones. This was actually part of the reason for the switch to Typescript, as keeping track of when and where nodes and links had been mutated by D3 and when and where they had not was quite hard. But as it turns out, Typescript does not support this kind of ”type mutation” in any way or fashion. In the end the simplest solution was to add nodes and links to the simulation as early as possible, so that they could always be assumed to be mutated. The type annotations of nodes and links correspond to this. This, in turn, has resulted in some type errors that cannot be fixed, where the type inferencer is smart enough to realize that the objects are not of the mutated type as we claim they are.

Another challenge we encountered was debugging our visualizations. When a piece of code did have the expected result, or when the visualization did something that we did not expect, figuring out where the issue was, was at times quite hard. Inspecting the raw svg could be quite time-consuming, especially in the cases where one element out of very many did not appear, and we had to manually locate it in the generated html to inspect its properties. Due to how Javascript works, the code does not necessarily crash when the wrong number of arguments are given, or the arguments or of the wrong type. This led to some subtle bugs where typos, or things like arguments being in the wrong order did not result in a crash, but simply some slightly unexpected behavior in another part of the code. This challenge was exacerbated by D3's heavy reliance on strings as arguments, which cannot be inferred by the IDE's auto-complete and/or language server.

9.3 Limitations

Our results can be no more accurate or detailed than the data we receive from Spotify. For example, genres only being registered to artists and not tracks hurts how granular we can be in any analysis based on genre. In future work, it could be interesting to use external data source such as MusicBrainz or LastFM to find more granular genre data and see if more interesting results could be found.

9.4 Future Work

The Explorify platform could be extended to music streaming histories from other platforms such as YouTube or Last.fm. In addition, the Explorify platform could adopt datasets from two or more users and display their similarities and differences. Several people we have talked with have expressed interest in such a feature. In his work on Last.fm explorer, M. Pretzlav shows how this could be done for the data of two users [10]

In the future we would very much like to perform a user study of Explorify. This was something we would actually have liked to do quite early in the process, but due to time constraints were unable commit to. Many of the questions we have regarding the efficacy of our solution would be answered by such a study.

There are many smaller features which we would have loved to add to Explorify, but sadly did not have time to implement:

- More control over which nodes are shown. Currently, we only show top n most popular nodes, as controlled by the edgemap control panel. We would have liked to have made this widget a sliding scale instead, allowing the user exact control over which range of artist popularity to select from. We would also have liked to support more ordering criteria than personal popularity. For example global popularity, or by the ”firstStream” attribute.

- Support for selecting multiple days in the heatmap calendar, either to contrast or combine the sets of artists for the selected days.

- For days with a lot of listening activity, the timeline view can get very cramped showing periods with lots of switching between artists. We would have liked to help this issue by adding support for zooming in the timeline view.

- More edgemap options. We are excited by the though of what one could discover if more options for edgemap configuration were added. For example, we would have liked to add support for changing between multiple types of edges, having edges for related artists, artists that were streamed in the same session, and more. We would also have liked to explore the possibility of adding more options for positioning and coloring artists.

Fig. 12. Final version of the Explorify platform. There are a total of 4 different interactive and interconnected views. The main view is the 'Artist Genre Network' which is supported by the 'Daily Listening Pattern' and the 'Top 20 Artists' views. Another view, 'Detailed Listening History per Day', is positioned at the very bottom and is displayed only when the user has selected a day in the 'Daily Listening Pattern' view.

- Have the calendar heatmap tiles encode the most dominant hue for their respective days, with saturation encoding how dominant and lightness encoding the total activity as before (only lightness is limited to some range where the hue of very active days is still easily discernible).
- We would also have liked to add more connectivity to all the views. For example, it would have been great if, when highlighting an artist, the calendar heatmap would display on which days the highlighted artist was streamed, and how much they were streamed.

9.5 Project Reflection

Working through this project has been a very challenging but rewarding experience. It has taught us a lot about going from an idea to doing research that supports and develops that idea, and finally, building a pipeline to implement it while iterating the original design and modifying accordingly to the time constraints. While some things did not go as smoothly as we had initially expected, namely compatibility issues with different build tools causing massive headaches when trying to combine code bases, we still managed to work around these constraints while gaining valuable learning experience. It has taught us the importance of being diligent and checking each line of code to ensure that it functions as intended. It has also highlighted the importance of publishing functioning and well-documented code for any of our future research projects. Another challenge that we did not anticipate was the time required to learn D3. While we managed to handle most of these challenges, next time we plan to be more vigilant in identifying them ahead and plan accordingly rather than resolving them if and when they arise.

On the time management side, we would also want to reflect on the challenges of correctly estimating the time needed to deliver individual components which is an important skill for the successful prioritizing of the most promising ideas. For the future, we would aim to have the overall work split into smaller chunks so that it is easier to accurately estimate the delivery times.

10 Conclusion

We introduce a platform which allows Spotify users to process and visualize their personal streaming history. The interactive and playful nature of our final dashboard aids Spotify users to explore their own music taste, find listening patterns and engage with their data. Our dashboard provides multiple interactive views and visualization simulations making it an enjoyable experience especially for people with narrower attention span.

ACKNOWLEDGMENTS

We would like to thank Elizabeth Reid, Mifta Sintaha and Nichole Boufford for providing their detailed and constructive feedback on our initial design which tremendously helped to navigate the project in the right direction. We would also like to express our gratitude to Professor Tamara Munzner for her valuable support and feedback which were paramount in achieving the positive outcome as well as completion of this project.

REFERENCES

[1] D. Baur and A. Butz. Pulling strings from a tangle: Visualizing a personal music listening history. IUI '09, p. 439–444. Association for Computing Machinery, New York, NY, USA, 2009. doi: 10.1145/1502650.1502715

[2] D. Baur, F. Seiffert, M. Sedlmair, and S. Boring. The streams of our lives: Visualizing listening histories in context. IEEE Transactions on Visualization and Computer Graphics, 16(6):1119–1128, 2010. doi: 10.1109/TVCG.2010.206

[3] L. Byron and M. Wattenberg. Stacked graphs – geometry amp; aesthetics. IEEE Transactions on Visualization and Computer Graphics, 14(6):1245–1252, 2008. doi: 10.1109/TVCG.2008.166

[4] T. Dang, A. Anand, and L. Wilkinson. Fmfinder: Search and filter your favorite songs. vol. 7431, pp. 348–358, 07 2012. doi: 10.1007/978-3-642-33179-4_34

[5] R. Dias, M. J. Fonseca, and D. Gonçalves. Interactive exploration of music listening histories. AVI '12, p. 415–422. Association for Computing Machinery, New York, NY, USA, 2012. doi: 10.1145/2254556.2254637

[6] M. Doerk, S. Carpendale, and C. Williamson. Edgemaps: Visualizing explicit and implicit relations. 04 2011. doi: 10.1117/12.872578

[7] M. D. M. C. José Bateira, Fabian Gouyon. Music discovery in spotify with rama. 2014.

[8] T. Munzner. Visualizaiton Analysis and Design. CRC Press, Taylor Francis Group, 2014.

[9] T. Munzner, F. Guimbretière, S. Tasiran, L. Zhang, and Y. Zhou. Treejuxtaposer: Scalable tree comparison using focus+context with guaranteed visibility. 22(3):453–462, jul 2003. doi: 10.1145/882262.882291

[10] M. A. Pretzlav. Last.fm explorer: An interactive visualization of hierarchical time-series data.pdf. 2008.

[11] J. Wirfs-Brock, S. Mennicken, and J. Thom-Santelli. Giving voice to silent data: Designing with personal music listening history. Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, 2020.

Task	People	Target date	Actual date	Target hours	Actual hours
Get to know the dataset	Both	Oct 21	Oct 21	5	5
Write initial report	Both	Oct 20	Oct 21	6	8
Familiarize ourselves with D3	Both	Oct 30	Oct 30	7	10
Create artist streamgraph	Jonatan	Nov 7	-	7	-
Create daily listening pattern heatmap	Inna	Nov 7	Nov 18	5	10
Create top artists over time barchart	Inna	Nov 15	Nov 16	7	7
Add interactivity to heatmap	Inna	Nov 15	Nov 16	5	4
Create artist-genre network	Jonatan	-	Nov 16	-	14
Write updated report	Both	Nov 15	Nov 16	6	4
Add interactivity to artist-genre network	Jonatan	Nov 21	-	5	-
Add interactivity to artist streamgraph	Jonatan	Nov 21	-	5	-
Create track clustering by audio features	Inna	Nov 21	-	6	-
Create top artists barchart	Inna	Nov 12	Nov 15	4	6
Create final UI for Explorify	Inna	Dec 1	Dec 1	5	7
Make aritst-genre network more edgemap-like	Jonatan	-	Dec 1	-	16
Add interactivity to top artists barchart	Jonatan	Dec 1	-	7	-
Add interactivity to track clustering	Inna	Dec 1	-	6	-
Create single-day timeline chart	Inna	Dec 10	Dec 7	6	4
Combine all visualizations together	Both	Dec 7	Dec 13	10	8
Improve supporting views	Inna	Dec 12	Dec 10	2	4
Improve artist-genre network	Jonatan	-	Dec 12	-	16
Create legend for edgemap	Jonatan	-	Dec 14	-	3
Improve responsiveness of dashboard	Both	Dec 14	-	5	-
Prepare presentation and demo	Both	Dec 14	Dec 14	5	5
Write final report	Both	Dec 17	Dec 18	10	10
Sum **Inna** hours:				67	99
Sum **Jonatan** hours:				76	99
Hours **total:**				143	198

Table 2. Target hours are individual. Items missing target hours and dates were not present in the original milestones plan. Items missing actual hours and dates were never implemented.