From Chaos to Cohesion: Structuring Playlists with User-Tuned Audio Feature Preferences

Roshen Chatwal

2025

1 Abstract

The impact of playlist ordering on music listening experiences is often overlooked. Few tools exist that automate track sequencing while accounting for dynamic individual preferences across a wide range of musical features. In this project, I develop a model that embeds songs into a multidimensional feature space and constructs a graph representation, using user-defined preferences to compute an ordering that maximizes adjacent track similarity throughout a listening session. I find that while no ordering can compensate for poor song selection, incorporating individual preferences can produce smoother transitions and a more cohesive listening experience across musical dimensions.

2 Background

2.1 Purpose

Playlists take essential roles in supporting incredible parties, long study sessions, and everything in-between demanding a consistent stream of music. Most of the work behind a playlist lies in the song selection. It's incredibly important to pick the right tracks for the right vibe, as music reigns king at the end of the day. But, perhaps the most underrated detail that makes or breaks a listening session is the song ordering within a playlist. I think we've all been there– vibing to a tune we love before rolling our eyes and hitting "skip" instantly after hearing the next track's intro (even though both are hits on their own). What a mood kill. Professional DJs consistently spend HOURS organizing their playlists and scheming up cool transitions to avert these undesirable flow breaks.

Incoming message from Captain Obvious: not everyone has the time or energy to manually order every song. Popular methods of automating song sequencing are surprisingly mediocre, too. Streaming giants including Spotify and Apple Music currently offer features like "shuffle" and the newer "smart shuffle". Respectively, these transitions randomly queue songs and use AI to queue songs matching the original playlist's vibe (often importing tracks from outside the list altogether).

I think we can create much better than random orderings without manually organizing songs, wasting precious time that should be spent on enjoying the moment.

I also think we can capture individuals' dynamic song transition preferences without resorting to opaque, catch-all "AI strategies" that ordinary people don't understand under the hood. *In this project, I aim to create an intuitive and customizable framework for ordering tracks within a playlist by maximizing the total similarity between successive songs. In doing so, I hope people can enjoy pressing "play" on all their playlists without a worry, trusting the music to flow as they would've wanted it.*

2.2 Key Assumptions

- Transitions between similar songs are smoother and most preferrable.
- Songs are representable as embeddings in Euclidean space, with similar songs being "closer" in terms of Euclidean distance.
- People using my model demand direct agency over the ordering of at most one song in their playlist.

3 The Data

I used the "Spotify Tracks Dataset" created by Maharshi Pandya. Pandya used the Spotify API to scrape data on 114,000 songs across 100+ different genres. I believe the dataset is fairly reliable, given my domain knowledge as a musician and its perfect Kaggle "Usability" score. For a detailed overview of the features, refer to my Jupyter notebook or Pandya's Kaggle webpage: Spotify Tracks Dataset on Kaggle.

4 Model Pipeline

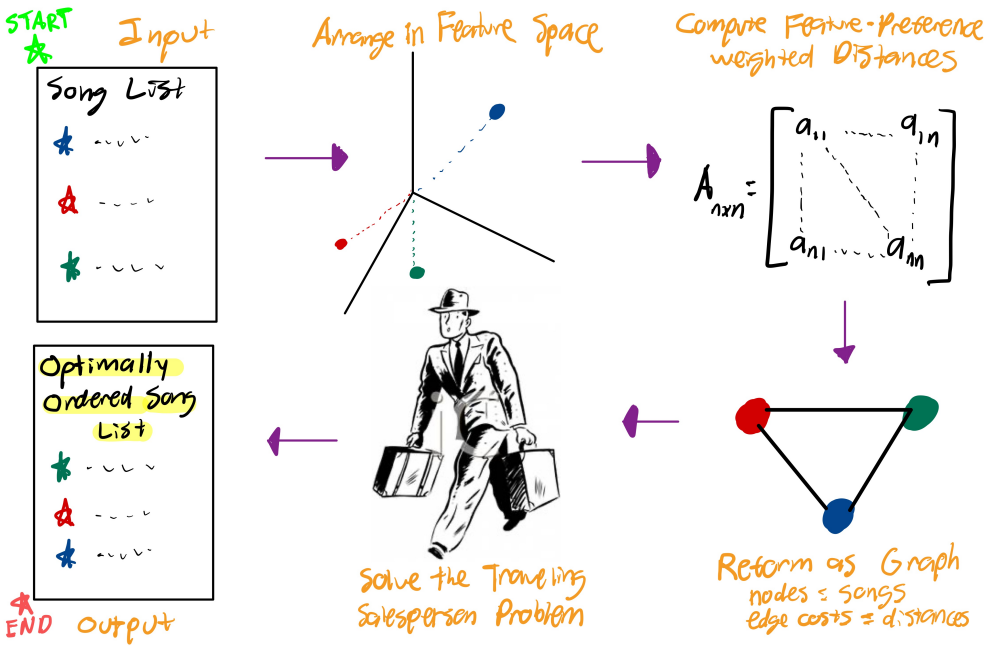

Figure 1: Diagram of playlist ordering process

I chose to model songs and their transitions through an undirected graph because graphs capture the concept of movement between states over time well. Songs being nodes and edge costs being a function of "similarity" (for which squared weighted Euclidean distance is the heuristic) follows intuitively. Minimizing total edge costs on a traversal of all nodes is mathematically the same as optimizing total similarity between successive songs– a main goal of my model. I will now explain the pipeline in depth, step by step.

4.1 Input: Picking my Playlists

An arbitrarily ordered/unordered list of songs is the input to my model. From experience, one of the most fundamental distinctions in tonal music is the presence or absence of lyrics. To explore this, I create two inputs playlists with some of my favorite songs in the dataset– one consisting of 12 tracks with vocals (e.g., pop, rap, country, EDM), and the other featuring 12 instrumental tracks (primarily classical music). I strictly include songs in different albums by different artists as to avoid special situations where the "smoothest" listening experience contradicts a composer's intentions for some standard ordering of tracks (more common in classical music).

Vocal Tracks	**Instrumental Tracks**
Lucky – Jason Mraz; Colbie Caillat	The Truman Show: Truman Sleeps – Philip Glass
You Proof – Morgan Wallen	Horn Concerto No. 4 in E-flat, K.495: 3. Rondo – Mozart
Swalla – Jason Derulo; Nicki Minaj; Ty Dolla $ign	Violin Concerto in D Major, Op. 61: II. Larghetto – Beethoven
Hot in It – Tiësto; Charli XCX	Le carnaval des animaux: IV–VI – Saint-Saëns
Just the Way You Are – Bruno Mars	Peer Gynt Suite No. 1: In the Hall of the Mountain King – Grieg
Stay – Zedd; Alessia Cara	Hungarian Dances No. 5 – Brahms
Centuries – Fall Out Boy	Piano Sonata No. 2 in B-flat Minor – Rachmaninoff
Drip Too Hard – Lil Baby; Gunna	Cello Suite No. 1 in G Major: I. Prélude – Bach
Ten Feet Tall – Afrojack; Wrabel	Carmen: "Habanera" – Bizet
Gasolina – Daddy Yankee	Swan Lake, Op. 20, Act II: No. 10 – Tchaikovsky
Hey Ma – J Balvin; Pitbull; Camila Cabello	Das Lied von der Erde: No. 2 – Mahler
Memories – Maroon 5	Die Meistersinger von Nürnberg: Overture – Wagner

Table 1: Unordered 12-song input playlists with artist names

4.2 Embedding Songs in Feature Space

4.2.1 Choosing THE Features

I want to consider enough features to facilitate orderings catering the preferences of partiers, locked-in students, and music listeners in between. Maybe some people love when successive songs in their listening session are similarly danceable, long, or fast. Others may prefer to maintain the emotional mood or stay within a genre for as long as possible. Most of us likely have preferences tuned across a wide range of parameters. But at the same time, offering too many custom controls may make the interface too busy and dissuade usership.

To balance simplicity in my model presentation while supporting high preferential expressiveness, I prioritize what I believe to be the $N = 8$ most relevant song features (in the dataset) that people consider while listening to music:

- `danceability`
- `duration`
- `mode`
- `acousticness`
- `instrumentalness`
- `valence`
- `tempo`
- `track_genre`

More detail on my feature selection process can be found in the Jupyter notebook.

4.2.2 Introducing Hyperparameters to Express Preferences

I establish new hyperparameters, or weights, to serve as intuitive "sliders" or "dials" enabling users to control the extent of each of the N features' influences on transitions. These weights are directly proportional to their respective feature's contribution in the overall squared weighted Euclidean distance.

$$\text{Hyperparameters are weights } w_i \text{ with } i \in \{0, 1, \ldots, N\} \text{ and } w_i \in [0, 5]$$

$$\text{Base case (neutral preferences):} \quad w_i = 2.5 \quad \forall i \in \{0, 1, \ldots, N\}$$

I constrain these hyperparameters to the range $[0, 5]$, which provides flexibility while allowing certain features to dominate the distance calculation (and thus the transition cost) if desired. Intuitively, features with larger weights contribute more to the total distance metric, which encourages the TSP solver to find a song ordering that minimizes large jumps in those specific characteristics across adjacent songs. Thus, *higher weights indicate a stronger preference for similarity in their respective features* between adjacent tracks.

4.3 Song Distance Matrix

Let

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1s} \\ a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ a_{s1} & a_{s2} & \cdots & a_{ss} \end{bmatrix} \in \mathbb{R}^{s \times s}$$

represent the song distance matrix, where s is the number of songs in the playlist.

Each entry a_{ij} equals the squared weighted Euclidean distance between songs i and j. I use squared weighted Euclidean distance so the *preference weights proportionally impact their respective features' contributions to cost metric.* This is necessary since we break the square-root in conventional Euclidean distance to account for the genre contribution through a separate embedding process.

I use Euclidean distance over other norms because it provides the straight-line distance between points in feature space, aligning better with our intuitive understanding of similarity in multi-dimensional spaces.

4.3.1 Categorical Feature Handling + Normalization

`track_genre` is a categorical feature, meaning its values lack cardinality and cannot be directly compared to one other using numerical interpretations. To address this issue, I represent each genre by its normalized aggregated feature profile—the rescaled means of all numerical variables across tracks within that genre. This process yields virtually continuous embeddings for each category and enables meaningful comparisons of genre similarity based on their cardinal, numerical audio characteristics. I then calculate the pairwise Euclidean distances between these normalized genre-level vectors, storing them as entries $d_{\text{genre(song i), genre song(j)}}$ in a *Genre Distance Matrix* D.

Normalization is a critical step before all distance calculations to make sure features with larger units don't have disproportionate impacts on the cost/distance metric. It rescales all variables to the same range, allowing for apples-to-apples comparisons and meaningful preference weighting across features. As mentioned above, I normalize the means of all numerical variables across tracks within each genre before making D. I also normalize the values of all remaining $N-1$ numerical features before proceeding to the overall transition cost calculation.

4.3.2 Calculating Preference-Weighted Distances

I calculate the **squared weighted Euclidean distance** using the normalized numerical feature values $x_n^{(i)}$ for songs i, the user-chosen weights w_n for features n, and the Euclidean `track_genre` distance matrix D entries $d_{\text{genre(song i), genre song(j)}}$. As mentioned earlier, the elegance of this distance variant is that *our weights proportionally scale the contribution of each feature's difference to the overall distance.* My formula uses a general N to account for potential future changes in the selected features, and will work so long as w_N remains the hyperparameter representing preference for genre similarity among successive songs.

$$D^2_{\text{Euclidean}}(\text{song}_i, \text{song}_j) = \left(\sum_{n=1}^{N-1} w_n \cdot \left(x_n^{(i)} - x_n^{(j)}\right)^2 \right) + w_N \cdot \left(d_{\text{genre(song i), genre song(j)}}\right)^2$$

$$a_{ij} = D^2_{\text{Euclidean}}(\text{song}_i, \text{song}_j)$$

Since I have two input playlists, I make matrices A_{vocal} and $A_{not\ vocal}$. Please refer to the Jupyter notebook to see all matrices, as they're quite large.

4.4 Representing Song Transitions via a Graph

It's quite straightforward to take the distance matrix and process it into an undirected, fully-connected graph $G = (V, E)$. V is the set of songs in our input playlist and E is the set of edges connecting $u, v \in V$ (with $u \neq v$), with each edge $e_{u,v}$ containing a cost attribute equivalent to a_{uv}.

Again, the key connection between the embeddings in multidimensional space and graphical representation is that our **edge cost equals the squared weighted Euclidean distance** between the pair of nodes sharing the edge. One of my model's key assumptions is that a high distance *under a consistent preference regime* reflects

less similarity between songs along the desired parameters. Intuitively, it follows those edges/transitions are "expensive" relative to others between more similar songs.

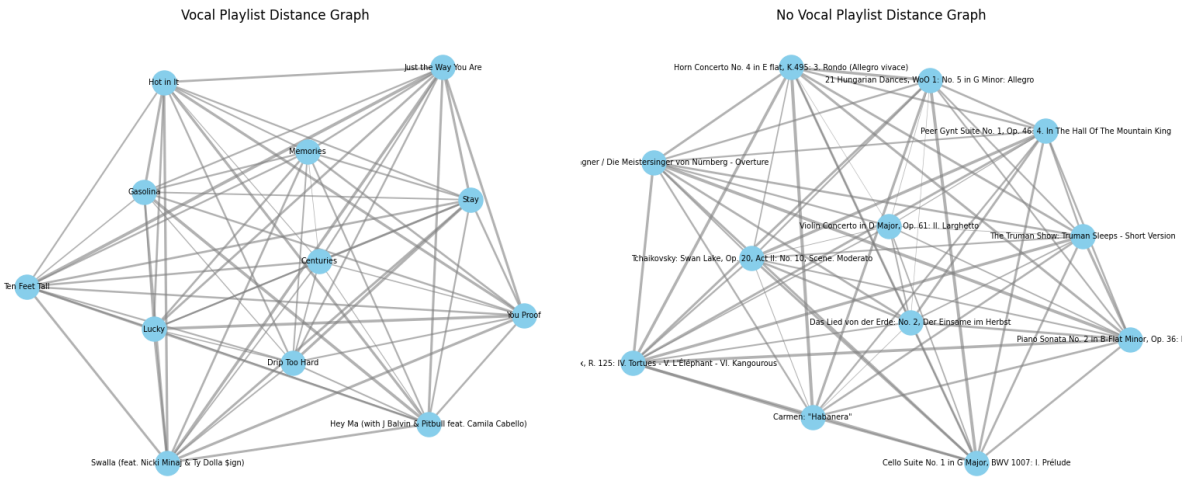

Figure 2: Playlist graphical representations (edges thicker between similar songs)

4.5 Solve the Traveling Salesperson Problem

The classic Traveling Salesman Problem (TSP) seeks the most cost-effective path through a graph G that visits every node exactly once and returns to the starting point— similar to how a playlist completely loops over time without skipping or repeating interior songs.

I use NetworkX's approximation algorithm in Python to compute a near-optimal ordering of songs by solving the TSP. By default, NetworkX applies a greedy heuristic starting from an arbitrary node (i.e. the first given) and repeatedly selects the "nearest" unvisited neighbor to construct a complete, near-optimal tour. Since the TSP is an NP-hard problem, I avoid using an exact solver with exponential time complexity, as it scales poorly on larger inputs (bigger playlists).

I solve the TSP with the cycle assumption to balances smoothness with practical usability. Setting `cycle=True` ensures that the playlist forms a closed loop, returning to the starting song— similar to how most streaming platforms behave when looping is enabled. This structure allows users to begin playback on *any* song in the cycle while maintaining smooth transitions in either direction, making the order flexible and user-centric. While setting `cycle=False` might yield slightly smoother transitions in a linear sense, it restricts the user to a specific starting and ending song. The lack of a return-to-start edge means the loop is open, reducing flexibility if users want to start playback on a song different than the first.

I also calculate the total path cost, which provides a useful cardinal measure of how smooth/cohesive the "optimal" ordering is per the user's entered preferences. Note that *path costs are only comparable under a consistent parameter regime and playlist size,* since changing either would alter the basic structure of the squared weighted Euclidean distance formula and TSP solution.

4.6 Output

I simply print out the "optimal" ordering and path cost for each playlist as determined by the TSP solver results.

Vocal Playlist (TSP Order)	**Instrumental Playlist (TSP Order)**
1. Lucky	1. The Truman Show: Truman Sleeps
2. Centuries	2. Le carnaval des animaux: IV–VI
3. Stay	3. Piano Sonata No. 2 in B-Flat Minor
4. Drip Too Hard	4. Die Meistersinger von Nürnberg
5. Just the Way You Are	5. Violin Concerto in D Major: II. Larghetto
6. Ten Feet Tall	6. Das Lied von der Erde: No. 2
7. Swalla (feat. Nicki Minaj & Ty Dolla $ign)	7. Peer Gynt Suite No. 1: In The Hall Of The Mountain King
8. You Proof	8. Swan Lake: Act II: No. 10
9. Memories	9. Cello Suite No. 1 in G Major: I. Prélude
10. Gasolina	10. 21 Hungarian Dances: No. 5 in G Minor
11. Hey Ma (with J Balvin & Pitbull feat. Camila Cabello)	11. Horn Concerto No. 4: 3. Rondo
12. Hot in It	12. Carmen: "Habanera"
Total distance: 41.8684	**Total distance: 31.1489**

Table 2: TSP-Optimized playlist orderings and total path costs (base param case)

As mentioned earlier, these orderings can be shifted (without increasing total path cost) to cater a listener's preferred starting point so long as all adjacent songs remain adjacent. The last and first song are considered adjacent due to my `cycle=True` setup.

I find it interesting that the total distance was about 25% less for the instrumental playlist. More conjectures and analysis on this to come!

5 Sensitivity Analysis

5.1 Playlist Content

This section explores the model's sensitivity to changes in the input, holding the base parameter regime constant.

5.1.1 Song Swaps

I want to examine my model's behavior following a few songs being randomly swapped between the vocal and instrumental playlists. This scenario intends to represent accidental or quirky little changes we may make to our playlists in real life.

Vocal-ish Playlist (TSP Order)	**Instrumental-ish Playlist (TSP Order)**
1. Lucky	1. The Truman Show: Truman Sleeps
2. Carmen: "Habanera"	2. Centuries
3. Peer Gynt Suite No. 1: In The Hall Of The Mountain King	3. Swalla (feat. Nicki Minaj & Ty Dolla $ign)
4. Stay	4. Horn Concerto No. 4: 3. Rondo
5. Drip Too Hard	5. 21 Hungarian Dances: No. 5 in G Minor
6. Just the Way You Are	6. Cello Suite No. 1 in G Major: I. Prélude
7. Ten Feet Tall	7. Swan Lake: Act II: No. 10
8. Hey Ma (with J Balvin & Pitbull feat. Camila Cabello)	8. Das Lied von der Erde: No. 2
9. Memories	9. Violin Concerto in D Major: II. Larghetto
10. Gasolina	10. Die Meistersinger von Nürnberg
11. Hot in It	11. Piano Sonata No. 2 in B-Flat Minor
12. You Proof	12. Le carnaval des animaux: IV–VI
Total distance: 48.4901	**Total distance: 50.8534**

Table 3: TSP-Optimized orderings and path costs for randomly modified playlists

Both path costs increased, but the jump was MUCH higher in the playlist originally containing instrumental songs (approximately +61% vs +18%). This suggests vocal tracks may be more out of place in playlists with instrumental songs than instrumental tracks are aloof in playlists with vocal songs. More rigorous analysis and trials may be necessary to conclude this, since the result shown is just from one random swap of two songs between the original playlists.

Perhaps as expected, the songs that got swapped remained neighbors in their new playlist. This likely comes a combination of their sonic similarities and my base/neutral parameter regime. The stationary songs' orderings moved a bit more than I expected, but most that shifted maintained at least one neighbor from before. It makes sense that completely swapping swapping nodes can change the "best" available neighbor. My model overall seems moderately robust to small changes like contrasting song swaps. I bet it would be more robust to pure song additions rather than swaps.

5.1.2 Song Additions

I also want to see how my model responds to a large additive disruption in the original playlists. I thought forming a mega playlist of vocal and instrumental tracks would be a decent way of representing the potential extreme heterogeneity and bigger sizes of real-life playlists.

Combined Playlist (TSP Order)
1. Lucky
2. Horn Concerto No. 4 in E flat, K.495: 3. Rondo (Allegro vivace)
3. Carmen: "Habanera"
4. Le carnaval des animaux: IV. Tortues – V. L'Éléphant – VI. Kangourous
5. The Truman Show: Truman Sleeps
6. Piano Sonata No. 2 in B-Flat Minor: I. Allegro agitato
7. Die Meistersinger von Nürnberg – Overture
8. Violin Concerto in D Major: II. Larghetto
9. Das Lied von der Erde: No. 2
10. Peer Gynt Suite No. 1: In The Hall Of The Mountain King
11. Swan Lake: Act II: No. 10, Scene. Moderato
12. Cello Suite No. 1 in G Major: I. Prélude
13. 21 Hungarian Dances: No. 5 in G Minor: Allegro
14. Memories
15. Gasolina
16. Hey Ma (with J Balvin & Pitbull feat. Camila Cabello)
17. Just the Way You Are
18. Stay
19. Drip Too Hard
20. Centuries
21. Ten Feet Tall
22. Hot in It
23. Swalla (feat. Nicki Minaj & Ty Dolla $ign)
24. You Proof
Total distance: 55.2709

Table 4: TSP-Optimized combined playlist and total path cost

The optimal combined playlist ordering clusters pretty cleanly into contiguous blocks of vocal and non-vocal tracks. This outcome is somewhat expected and mirrors the behavior seen in the swap perturbation case at a large scale. Notably, the relative orderings of vocal and non-vocal tracks remain similar to those in the base case with two separate playlists, even under the large additive perturbation. This suggests that my model becomes *less sensitive to individual changes* once many consistent, similar changes are introduced— a sign of *robust behavior under structured variation.* At least, in the constructive sense where we add more songs (as is what typically happens to playlists over time).

Even more interestingly, *the total path cost of the optimal combined playlist is much lower than the sum of the path costs of the individual playlists* (approximately 44% less). This is a fantastic result from the song-additive sensitivity analysis, as it suggests that the model may perform better (under a consistent preference regime) when applied to larger and more diverse sets of songs. Rather than simply aggregating inefficiencies, the model appears to find smoother transitions and more efficient pathways when given a broader search space. I didn't expect this, but it's good for potential users so I'm happy.

5.1.3 Noisy Playlist

Following the previous analysis, I'm curious how my model performs on randomly generated playlists of 24 unique songs under the same parameter regime. Sometimes we're too busy to actively curate songs and just want need to put something on.

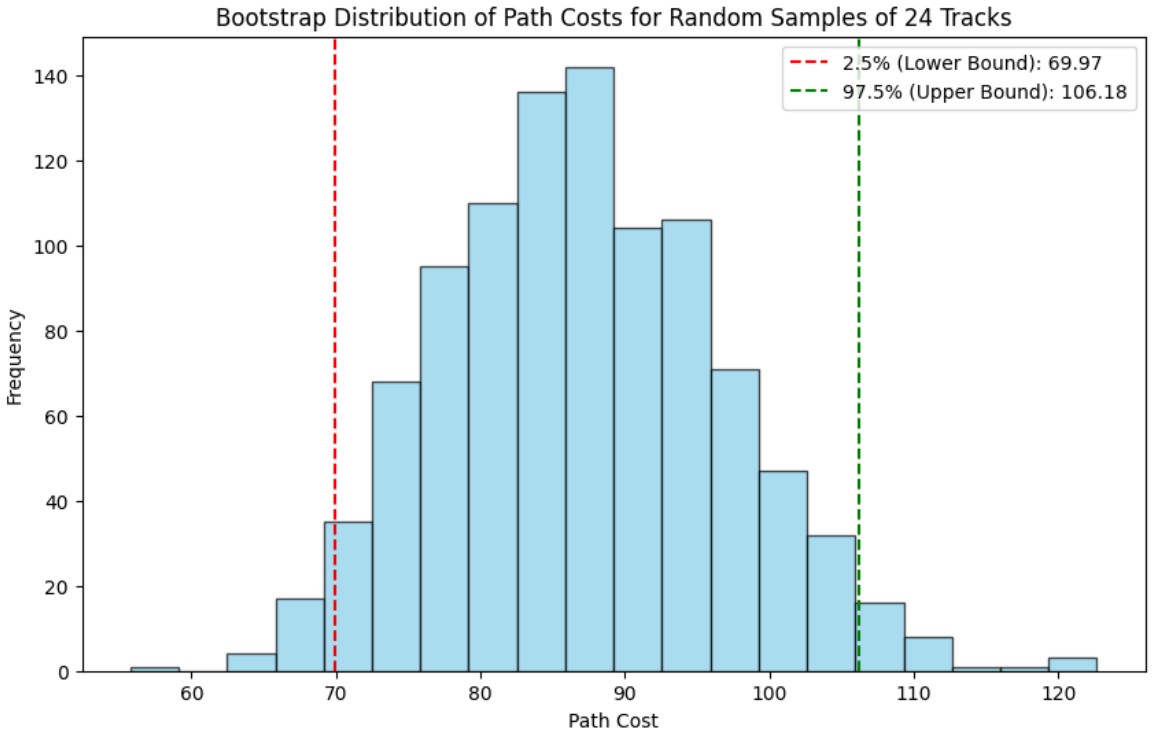

Figure 3: Simulated Optimal Total Path Costs on Random Playlists of 24 songs

The confidence interval for random performance under the same weights/preference regime suggests that curating a playlist can drastically improve the optimal listening experience. My combined playlist in 5.1.2 had a statistically significant path cost relative to that of identically sized random samples. This indicates that music selection really does matter for creating cohesive playlists, as we all know. It's unlikely to get an amazing listening experience (in terms of smooth transitions) with tons of disjointed songs, so users of my model should make sure to stay diligent about the tracks they're adding to the playlist! Perhaps keeping a general theme or vibe in mind while adding songs is enough to enjoy smoother song transitions (relative to an optimally ordered playlist containing randomly picked songs).

5.2 Hyperparameters

Up till now, I've presented results and sensitivities under the base hyperparameter setting indicating neutral preference (with each $w_i = 2.5$ to lie in the middle of its permissible range, though there are multiple possible base settings so long as all the w_i's are equal). There are $N = 8$ hyperparameters, and I think it would be futile to exhaustively start adjusting them. I'll analyze the sensitivity to hyperparameters simply by returning to the motivation for the project: examining how different sets of preferences can translate into "ideal" playlist orderings.

Let's say I'm on a road trip with my three friends. Here's some background on each of us:

- **Jake** is in his feels. He cares a lot about maintaining the emotional mood and `track_genre`, and much less about everything else.

- **Stacy** has OCD. She cares a lot about smoothing the timbre (`acousticness`) and tempo changes across successive songs, and less about everything else.
- **I'm** a party animal. I deeply and exclusively care about matching adjacent songs based on their `danceability` and, to a lesser extent, their emotional mood.
- **Riya** thinks elevator music is cool. She cares decently about maintaining modality and a lot about keeping successive pieces at similar `duration`, and moderately about everything else.

Since we're all friends, our road trip parameter settings will be the average of our personal settings. I solely examine the discrepancy between the group and individual orderings on my original playlist with vocals, as we like singing together on road trips.

`params`	[`dance`, `duration`, `mode`, `acoustic`, `instrumental`, `valence`, `tempo`, `genre`]
`jake_params`	[1.5, 1.5, 1.5, 1.5, 1.5, 5, 1.5, 5]
`stacy_params`	[2.5, 2.5, 2.5, 5, 2.5, 2.5, 5, 2.5]
`rosh_params`	[5, 0, 0, 0, 0, 3.5, 0, 0]
`riya_params`	[3, 5, 4, 3, 3, 3, 3, 3]
`trip_params (avg)`	[3.0, 2.25, 2.0, 2.375, 1.75, 3.5, 2.375, 2.625]

Table 5: Individual and Group Hyperparameter Preferences

I used the model to produce ideal orderings of the same playlist to cater all the individual preferences and the group preference. I also produced ordered song feature heatmaps like the ones below in an attempt to gain insight into any patterns present.

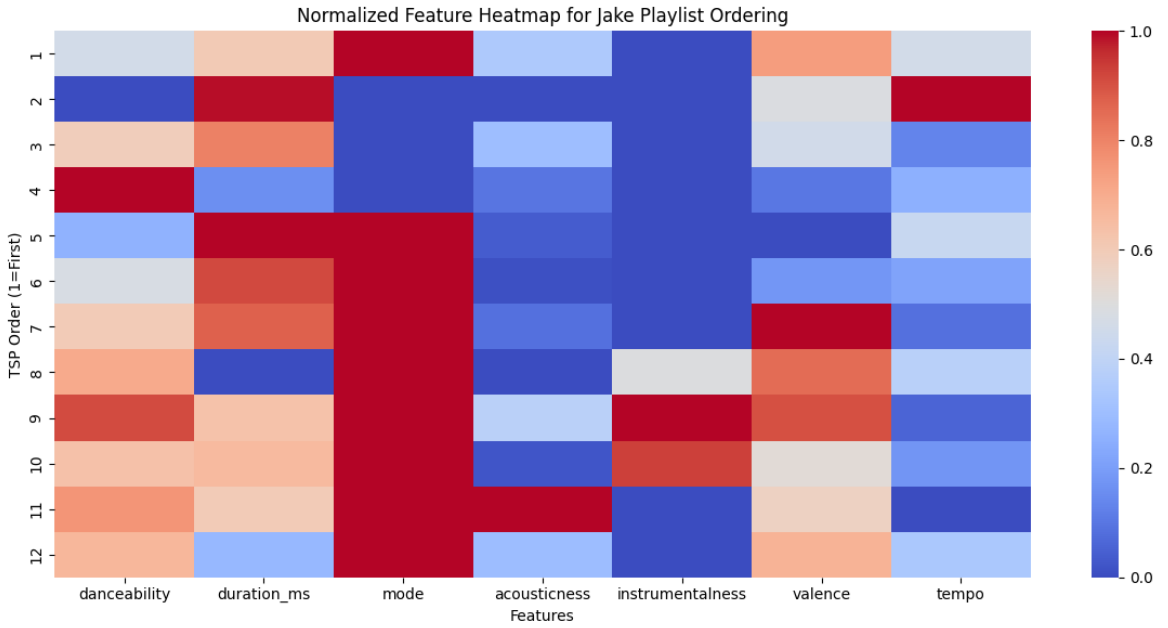

Figure 4: Individual case: ideally ordered song feature map for Jake

Let's look at Jake's results as a representative of the individual cases, in which there are more likely to exist dominant preferences for smoothness across a few features. It's pretty clear that the ideal adjacencies largely are between songs of similar mood (i.e. "Memories" and "You Proof"). The `valence` gradient appears much smoother

than the others, although I'll note that there's generally just lesser variance in this vocal playlist among the songs' `mode` and `instrumentalness`. Based on Jake's actual ordering, which I didn't list here for brevity, the songs are also pretty contiguous based on `track_genre` (i.e. the dance songs from 7-11). I wasn't able to place `track_genre` into the heatmap since it's categorical, so perhaps I'm missing some intuition from lacking an interpretable visualization of this feature Jake cares about.

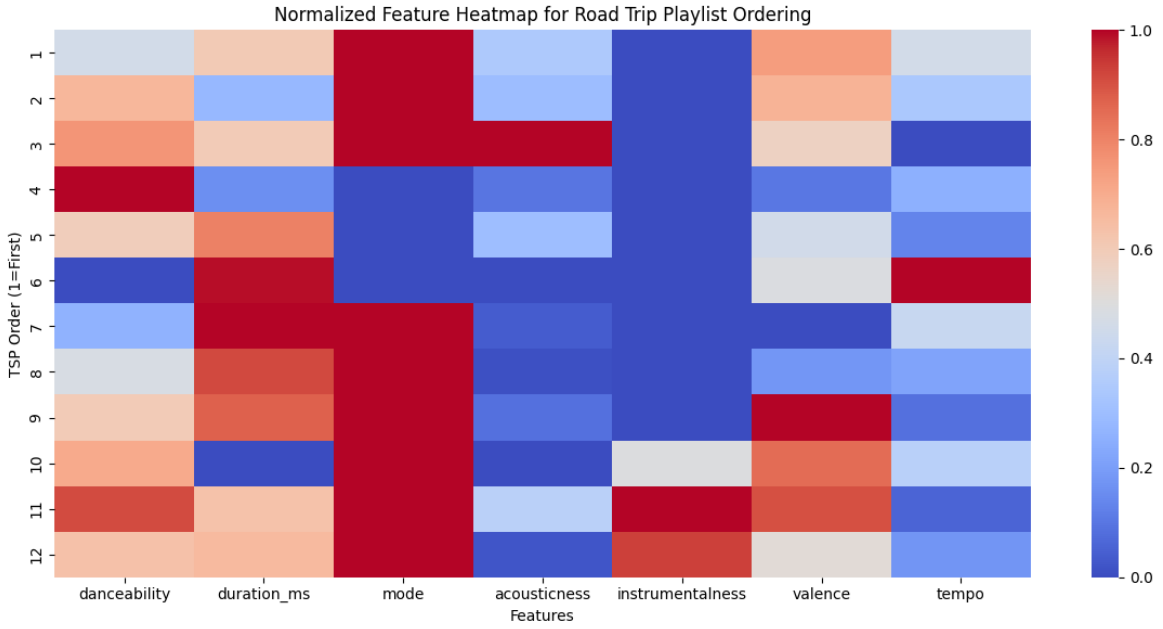

Figure 5: Aggregate case: ideally ordered song feature map for the whole car

As expected, the aggregated preferences are more all over the place. No single feature dominates organizationally as much as in the individual cases, and thus we don't observe a smooth gradient. The group weights themselves reflect this in how they're more smoothed out relative to the spikes across a few features in the individual parameters. The `danceability` and `valence` gradients look relatively smooth here compared to the other features, based on the visuals and intuition from their relative importance within the group preferences. But, clearly the other parameters have some rounding effects. There exist more transitions like those between tracks 6-7, which appears surprisingly distant on the `danceability` front, and between tracks 3-4, which seems a bit wacky on the `valence` front relative to other songs that could be smoother neighbors.

Overall, it seems like my model parameters can be adjusted to either make individuals or groups of similar people happy. It's much harder to appease a group of people with diverse preferences, as the aggregation usually smoothens out individual preferences with a more middle-middle ordering that few in the group truly resonate with. This result is consistent with what we observe in life, especially made evident by recent politics. Therefore, I especially recommend the use of my model for individuals or similar groups and a bit less so for diverse groups who'll on average get less utility from the ordering (even though that's likely to happen no matter the ordering method used).

5.3 TSP Path Type

Perhaps one of the biggest decisions I made is that the playlist should be presented in a way where starting point is arbitrary. By setting `cycle=True` while solving the TSP with no repeats (besides the start/terminal node), I effectively provide an "optimal" playlist order that is robust to variance in the starting point. As mentioned earlier, following a full loop in the cycle (stopping once the first song repeats) yields the same path cost regardless of where playback begins, since every edge cost in the cycle is incurred exactly once.

Although the `cycle` case was helpful for intuition and user flexibility, it is certainly a strong representational choice on my part. Who am I to say that users actually want to pick their first song? And realistically, how many music streamers actually make it all the way through their playlist loop during a listening session?

I now examine my model's performance under a `cycle=False` TSP solution, analyzing output differences under the stricter linear ordering regime that stops after the last song. In this case, no edge cost will be incurred for looping track number s (e.g., 12 in the example above) back to track 1, since that loop simply does not occur based on our start at song 1.

Vocal Playlist (TSP Order)	**Instrumental Playlist (TSP Order)**
1. Centuries	1. Das Lied von der Erde: No. 2
2. Stay	2. Peer Gynt Suite No. 1: In The Hall Of The Mountain King
3. Drip Too Hard	3. Swan Lake: Act II: No. 10, Scene. Moderato
4. Just the Way You Are	4. Cello Suite No. 1 in G Major: I. Prélude
5. Ten Feet Tall	5. 21 Hungarian Dances: No. 5 in G Minor: Allegro
6. Swalla (feat. Nicki Minaj & Ty Dolla $ign)	6. Horn Concerto No. 4: 3. Rondo (Allegro vivace)
7. You Proof	7. Carmen: "Habanera"
8. Memories	8. Le carnaval des animaux: IV–VI
9. Gasolina	9. The Truman Show: Truman Sleeps
10. Hey Ma (with J Balvin & Pitbull feat. Camila Cabello)	10. Piano Sonata No. 2 in B-Flat Minor: I. Allegro agitato
11. Hot in It	11. Die Meistersinger von Nürnberg – Overture
12. Lucky	12. Violin Concerto in D Major: II. Larghetto
Total distance: 35.8814	**Total distance: 23.4523**

Table 6: TSP-Optimized playlists and total path cost (no cycle)

The optimal playlist orderings did change quite a bit visually from the earlier `cycle=True` case. However, it seems that the new ordered lists are simply translated versions of the old ones, with tracks maintaining the same neighboring tracks as before (except for one neighbor swap in the Instrumental Playlist). This is a good result, as it demonstrates that the difference in optimal orderings between the `cycle` and `non-cycle` cases hinges on the starting song as expected. The fact that track neighborhoods remained largely intact between the `cycle=True` and `cycle=False` cases indicates that my model's local transition logic is stable and not overly dependent on arbitrary starting points.

The total distance traveled is slightly lower among both of the `non-cycle` orderings. That small drop is largely expected, since there is one fewer song-to-song transition in the path without a loop back to the top of the list.

6 Discussion

6.1 Playlist Curation: Music is King

One of the biggest findings from my model is that optimal ordering can do very little to fix bad/disjoint song selection. This was made apparent from my bootstrap playlist sampling, which almost always resulted in less cohesive listening sessions almost than using the same-sized (and still diverse) playlist I curated under the same parameter regime. My model did perform much better with more songs and heterogeneity though, demonstrating cost synergies with the combined playlist.

In light of these observations, I suggest that users of my model still bring their A-game when it comes to making input playlists. I also suggest that you live adventurously, and consider adding songs for which like but aren't sure about on a fit basis (the model can mostly help take care of that according to your preferences). Overall the model is fairly robust to song additions, and benefits from potentially exploiting closer connections between previously and newly added songs. I recommend to reduce the number of songs users remove because the orderings are much more sensitive to those. But, in light of how people barely remove songs from playlists anyways in real-life relative to the numbers they add on, I don't think this will be a big issue or consideration for most people.

6.2 Smoother Feature Gradients with Stronger Preferences

The model's parameter sensitivity captures the idea that you're more likely to get out what you put in. This isn't a shame on people with mellow/neutral preferences for the features along which their song transitions are most cohesiveness. In fact, it demonstrates a key strength of the model to adapt to preferences via the feature weighting on Euclidean distance. We saw through the smoother color gradients in the individual case that having a few spikes in weight values for one's preferred features can genuinely lead to smoother transitions than when the weights are more homogeneous. The true value proposition of my model compared to others is its expressiveness of orderings on par with individual preference, so I encourage users to be bold and ponder more about what features they *truly* care about across adjacent songs.

At the same time, I encourage users not to put all their eggs in one (or a couple) baskets. Setting too many weights to zero and caring overly much about smoothness in one or two features can lead to problems in the orderings– especially playlists with songs that are outliers in those prioritized features. It's good practice to at least assign some small positive weight to each feature's contribution as to not completely regularize them LASSO-style. Check out the Jupyer Notebook analysis of Roshen's individual playlist for a better idea of the downside to assigning weights like his.

6.3 Vocal vs Instrumental: Music is Diverse

The results of my model and its sensitivities reveal a surprising amount of tendencies of vocal and instrumental music.

First off, the path costs suggest that instrumental music flows much more smoothly than vocal music in a vacuum. Perhaps it's the type of instrumental music I used, but my conjecture is that instrumental music tends to be longer, with more ebbs and

flows, and contains less sonic extremes than vocal music. Maybe some discrepancy could stem from differing studio technologies. But I think it's primarily the length of the instrumental tracks leading to song-aggregated features mellowing out and having less variance across instrumental tracks (thus giving smaller path costs relative to their vocal counterparts).

It follows that vocal tracks creeping into instrumental playlists are more disruptive to flow than the reverse. The higher variance in vocal tracks' audio features and higher consistency of extremes within those tracks (which don't get smoothed out as much due to their shorter durations) intuitively explode path cost when subbed into a instrumental playlist that probably has much less similar songs. Thus, I informally conclude that instrumental music is "closer" to vocal music than vocal music is to instrumental music. It kind of makes sense in the context of squares and rectangles, as most vocal tracks include instruments but few instrumental tracks have vocals.

6.4 Shortcomings

6.4.1 TSP Difficulty

The TSP is a hard problem to solve both exactly and efficiently, as it's NP-Hard. I already conceded that the "optimal" orderings my model generates via NetworkX's solution aren't without a doubt ideal. But it's hard to know how far off they are when the true optimum is unknown. Even the greedy heuristic methods don't scale particularly well with graph size (especially in a fully-connected, where there's many edges). I'm slightly worried about the speed at which my algorithm would work for potentially gigantic playlists as a result.

6.4.2 Key Assumptions

My key assumptions make the model much easier to formulate. However, they certainly don't capture some of the most significant ways people like to organize their music listening sessions. "Smoothness" certainly isn't what people always want. Often, I observe people order playlists to fit some narrative arc within the music. My model can't do that. Sometimes, although rarely, people prefer choppier transitions. My model could do that if I optimized the TSP to take the highest cost path, but I didn't include an option for that. Most importantly, it's undeniable that many people have very specific orderings in mind that capture unmeasureable stochasticity in their personal desires or patterns that I can't draw without additional features or user input. My model is not very flexible at all to changes from hard coded transitions, and if I had more time I'd find a way to introduce more adaptability to in-the-moment changes in preference beyond a general idea of feature prioritization

7 Summary

My project in large part achieves its goal of automating song sequencing withing playlist while creating maximum cohesion, according to individual preferences. It utilizes data on songs and user inputs to optimize the total similarity between adjacent tracks, using that similarity as it's key metric for transition "smoothness." It does this by representing songs as embeddings in multidimensional space, calculating distances,

and reforming the scenario as a Traveling Salesperson Problem on a fully connected, undirected graph.

The model, its sensitivities, and contextualization in the broader area of music and playlist curation are emphasized throughout this paper. Some notable findings include a confirmation of the importance of cohesive music selection for playlists, the strength of the model at satisfying individuals demonstrating their preferences, and a true sonic distinction between vocal and instrumental music relating to their flow and playlist inclusion. Some model shortcomings include the difficulty of solving the Traveling Salesperson Problem itself and its limiting key assumptions.

8 Sources

- Spotify Tracks Dataset Discussion on Kaggle
- ChatGPT