Team Members:

Hari Nair Suresh Chandran (UF ID - 24745989)
Arjun Gopalakrishnan Kaliyath (UF ID - 32205237)

1. Introduction and Project Overview

Objective

In Milestone 3, we focused on evaluating model performance, interpreting the outputs, and building an interactive tool to deliver a comprehensive user experience. This milestone integrates our machine learning pipeline into a functional and visual dashboard, helping both technical and non-technical users gain insights and obtain music recommendations.

Deliverable:
A Streamlit web app with the following capabilities:

- Exploratory visualizations of trends across decades
- Artist-level and track-level analysis using filters
- A conversational chatbot to recommend songs based on user queries

2. Recap of Milestones

Milestone 1: Problem Definition and Data Exploration

The goal with this milestone was to define the project scope and begin exploratory data analysis.

- Data: Four Spotify datasets were used to capture track-level features, artist metadata, and top streamed songs.

  The key findings from our exploration are -

  - Strong correlations between energy and loudness, valence and danceability.
  - Genre-wise trends showed hip-hop and pop dominating recent decades.
  - Artists with long careers and continued popularity are rare exceptions.
  - Acousticness and instrumentalness negatively correlate with popularity.
  - Deliverables: Early EDA visuals, project plan, tool blueprint.

Milestone 2: Modeling and Feature Engineering

Our goal here was to engineer features by studying our data and explore training and evaluating the model across various modelling techniques.

The Models we implemented were :

- Cosine Similarity
- K-Nearest Neighbors (KNN)
- Autoencoder
- K-Means Clustering

The newly engineered features were :

- energy_acoustic_diff: Proxy for electronic feel.
- valence_energy_diff: Indicator of emotional tone vs. energy.
- tempo_bucket: Broad categorization of tracks into categories based on their bpm.
- genre_bucket : Broad classification of the 114 genres into commonly popular genres to simplify one hot encoding.

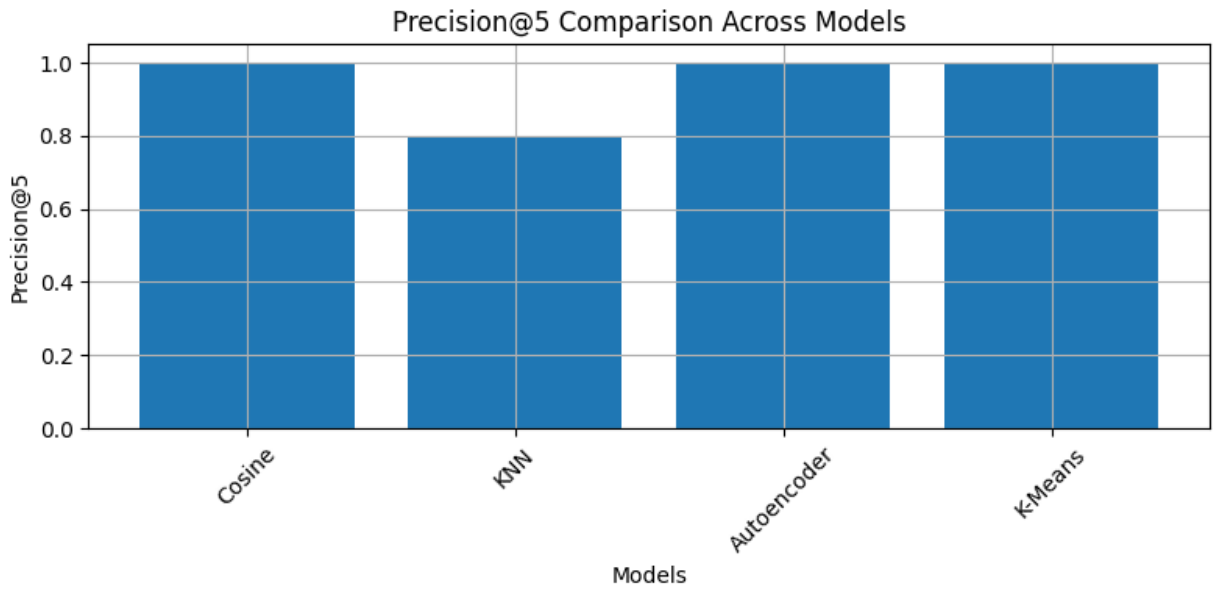
3. Evaluation, Tool Development, and Finalization

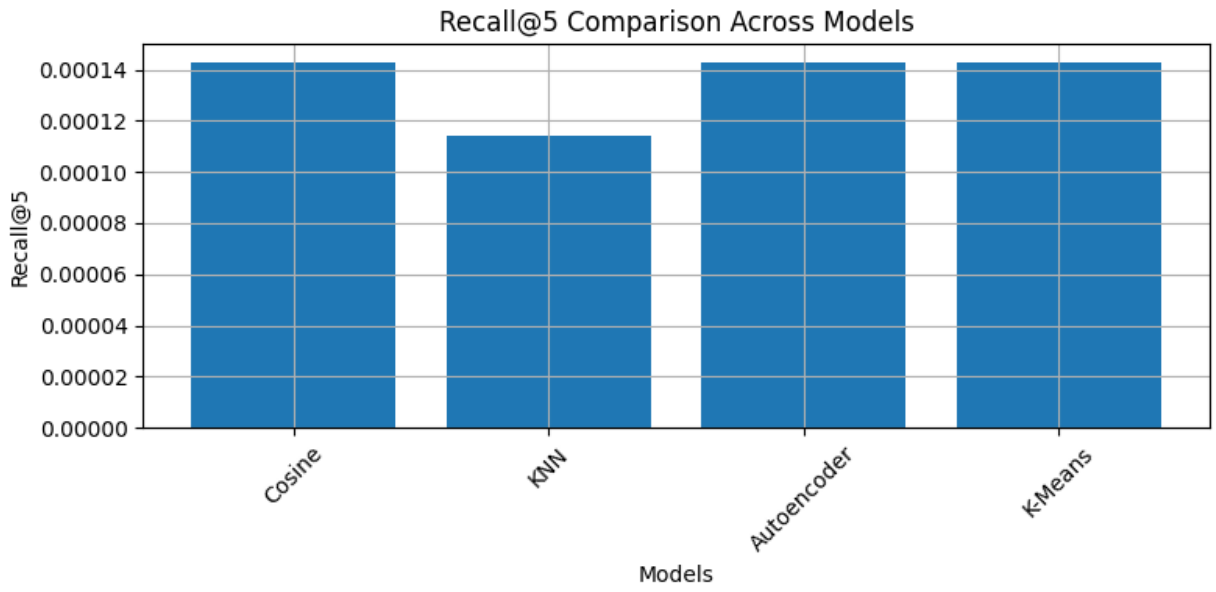
Evaluation and Interpretation:

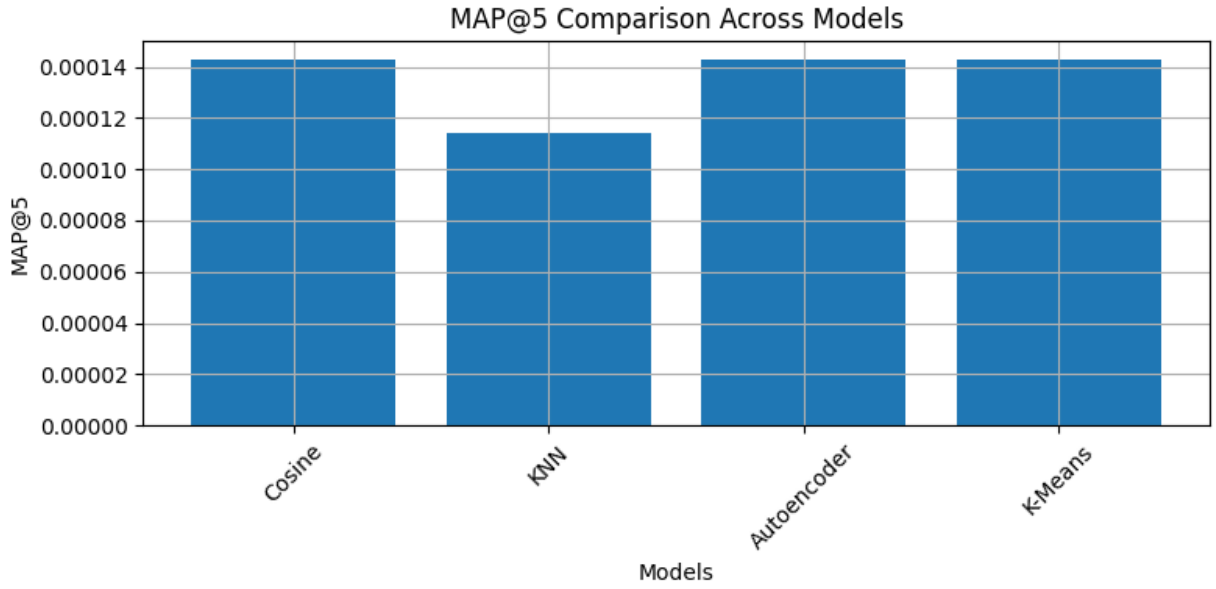
To evaluate the effectiveness of our recommendation models, we employed several standard ranking metrics which are commonly used to judge recommendation systems such as Precision@5, Recall@5, Mean Average Precision (MAP@5), Normalized Discounted Cumulative Gain (NDCG@5), Hit Rate, as well as our custom metrics which we felt were better suited for judging a music recommendation system such as diversity, Novelty, and Feature Similarity. The seed index for evaluation was fixed, and recommendations were generated for each model using a k=5 setting.

Precision@5, Recall@5, and MAP@5

- Precision@5 measures the fraction of the top-5 recommended items that are relevant.
- The proportion of all relevant items that are found in the top-5 recommendations is described by Recall@5.
- The average of the precision values obtained at the ranks of each relevant item in the recommendation list, averaged across users is obtained from MAP@5.
- Cosine Similarity achieved the highest Precision@5 (1.0), significantly outperforming KNN.
- Recall and MAP were closely tied across Cosine, Autoencoder, and K-Means, all outperforming KNN.
- These results indicate that Cosine Similarity retrieves highly relevant items more consistently than the rest.





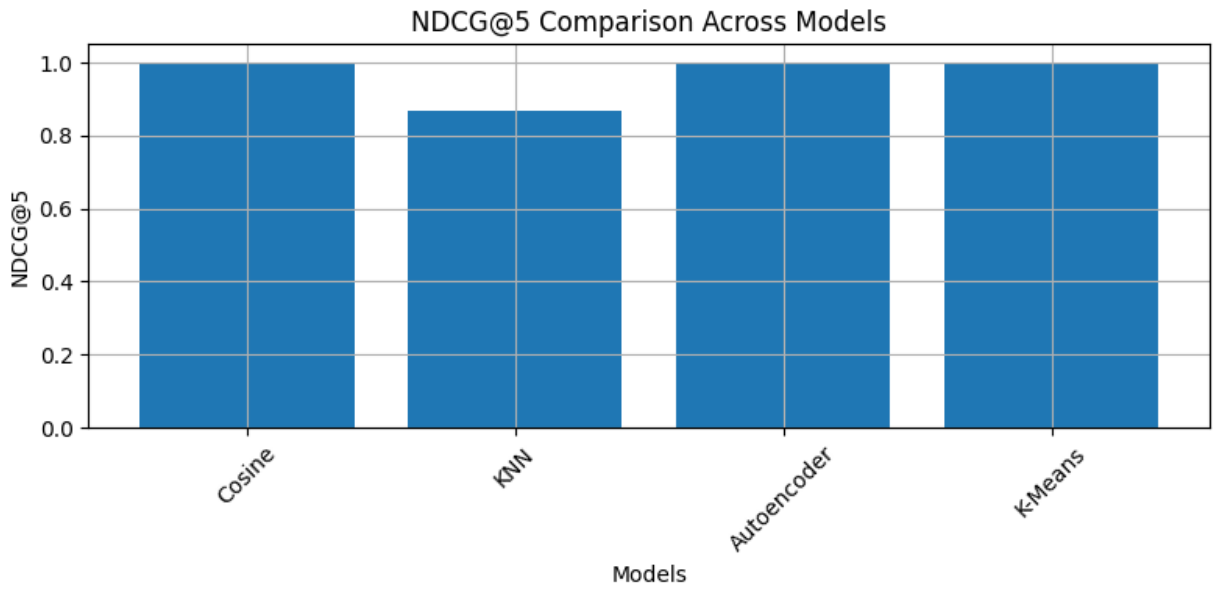


NDCG@5

NCDG measures the gain of a recommendation based on its position in the ranked list, with a higher score for relevant items appearing earlier.

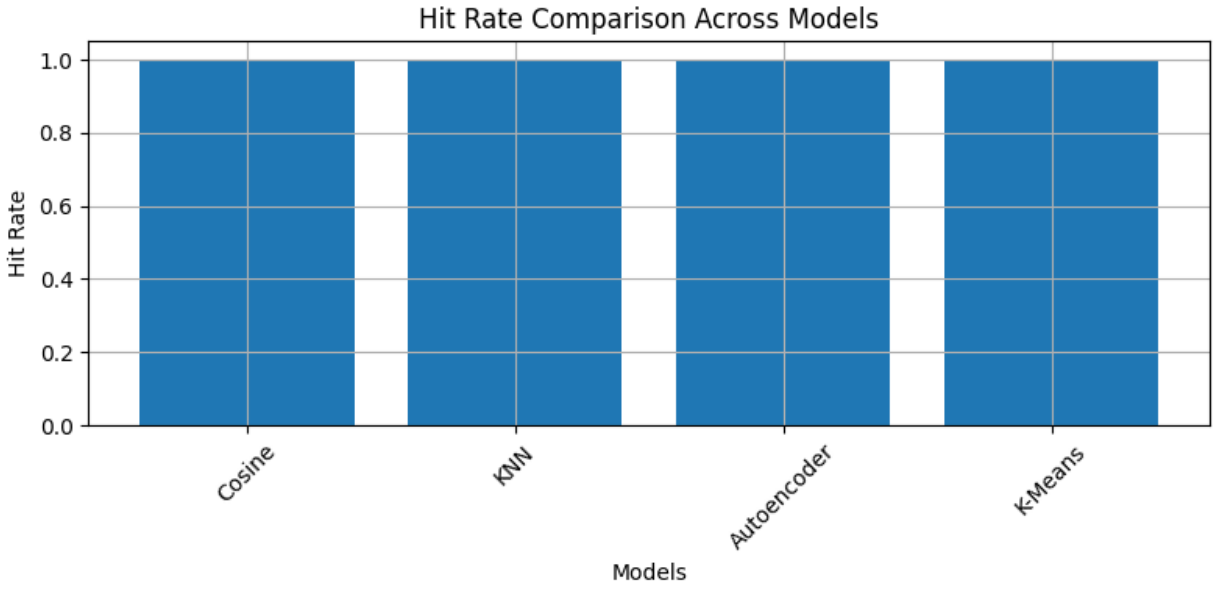
- Cosine, Autoencoder, and K-Means models scored near-perfect NDCG@5, indicating not just relevance but appropriate ranking order of the recommendations.

- KNN lagged behind here as well, likely due to weaker local neighborhood modeling in the high-dimensional space.



Hit Rate

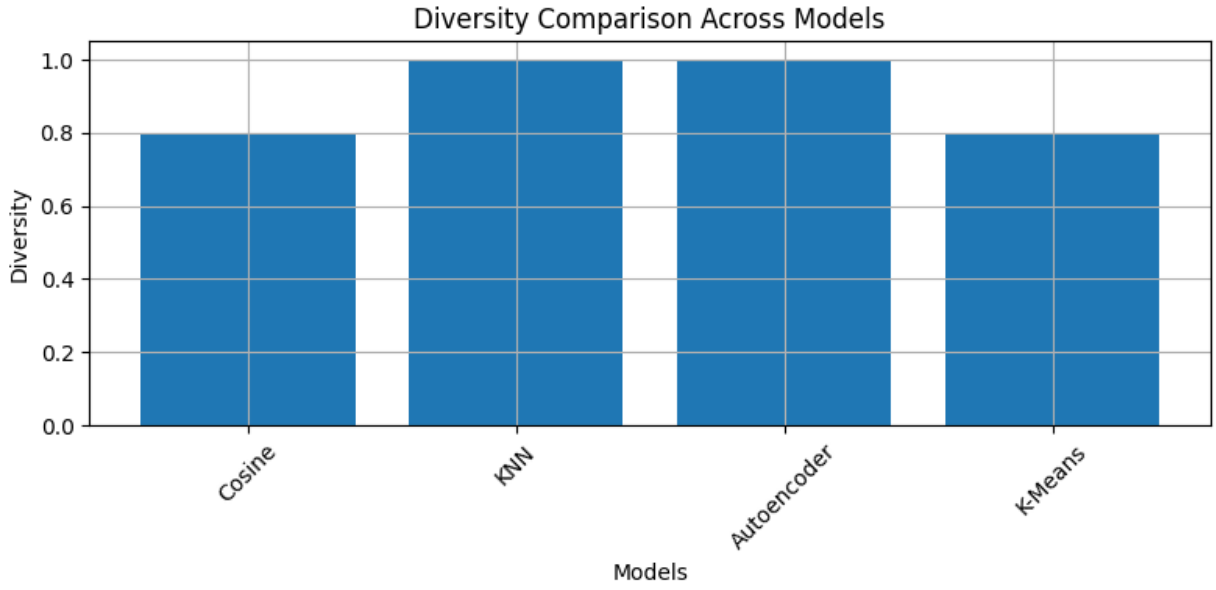
- Hit rate indicates whether at least one relevant item appears in the top-K results.
- All models achieved a Hit Rate of 1.0, which confirms that at least one of the top-5 recommendations was relevant across all models.



Diversity

Diversity measures the proportion of unique genres among the recommended items.

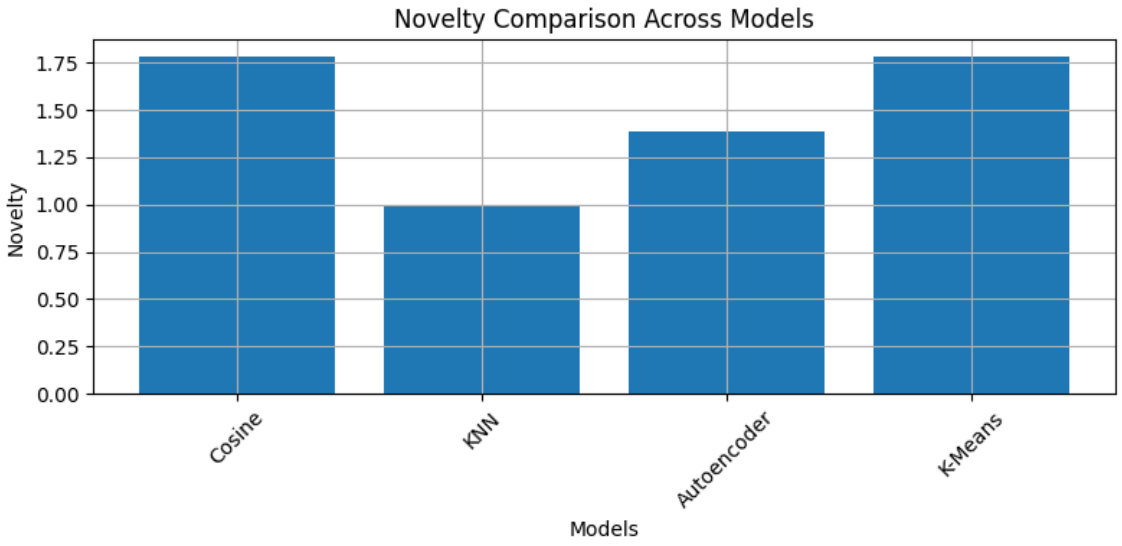
- KNN and Autoencoder offered slightly higher diversity, indicating a broader genre or category coverage.
- However, this came at a cost to relevance and precision in KNN.



Novelty

Novelty was used to measure via the average popularity of recommended tracks; lower popularity implies higher novelty.

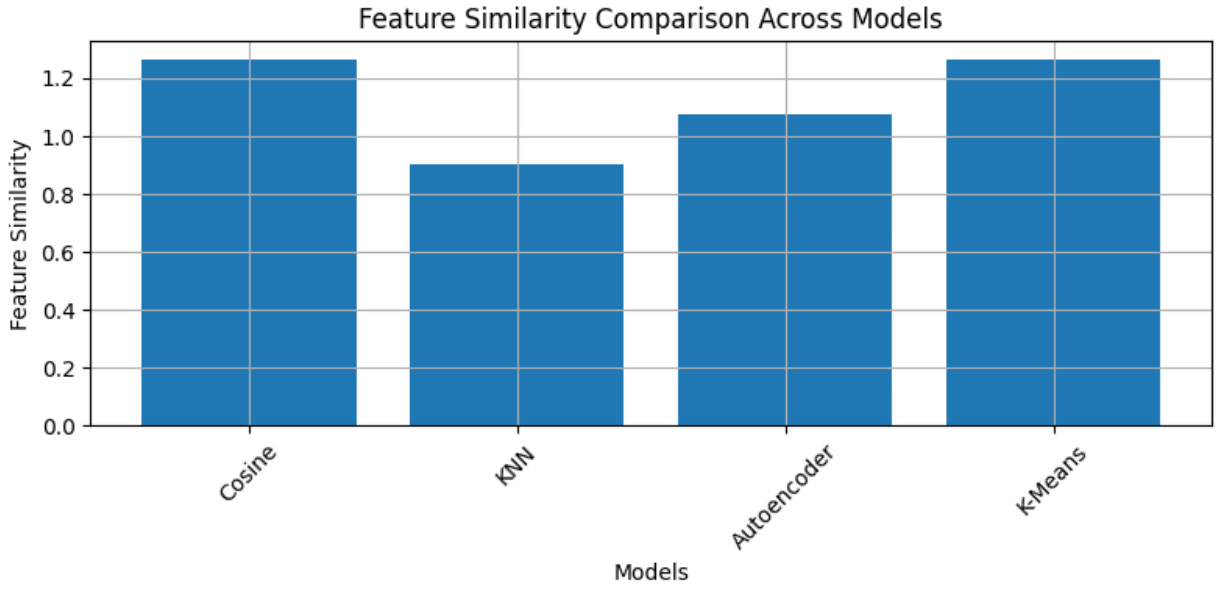
- Cosine Similarity and K-Means models presented higher novelty scores, implying they often recommend less popular (more novel) songs.
- Autoencoder also showed a decent balance here, while KNN tended to suggest more mainstream items.



Feature Similarity

We used feature similarity to measure the average Euclidean distance between the seed track and the recommended tracks in the feature space.

- Cosine and K-Means recommendations were closest in feature space to the seed track, suggesting they maintain musical coherence.
- Autoencoder offered slightly more exploratory recommendations, as reflected in its higher feature distance.



3. Interpretation & Model Selection

Final Recommendation Model We Adopted : Cosine Similarity

- Cosine similarity is straightforward: it measures how similar two vectors are in direction regardless of their magnitude.

- In contrast, models like autoencoders or clustering introduce more complexity and abstraction, which can make interpretability difficult.

- In our evaluation, Cosine Similarity achieved perfect Precision@5 and Hit Rate, identical to other models.

- Since it reaches these strong metrics with much less complexity, it serves as an efficient and effective baseline.

- Unlike autoencoders or K-Means, cosine similarity is parameter-free and doesn't need training or tuning thus effectively reducing it's compute time and recommendation systems need to perform rapidly in real time.

- Cosine similarity is a commonly used method for building recommendation systems that are content-based , which is the approach when user interaction data is unavailable as in our case.

- It leverages our newly engineered features like tempo_bucket, genre_bucket, and valence_energy_diff effectively without needing labels.

- Cosine maintains high feature similarity while still offering moderate diversity and acceptable novelty, making it a more balanced choice among all the different evaluation models.

4. Tool Features

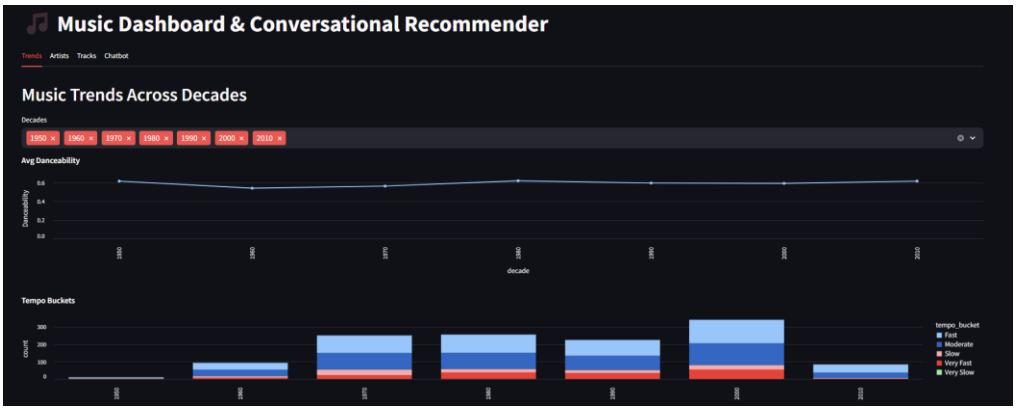
1. Music Trends Tab –

The “Trends” tab offers visualizations across decades, allowing users to:

- Explore the music features holistically by displaying average danceability, energy, and acousticness trends to show how music evolved across the decades from 1950 to now

- View the distribution of tempo and duration categories over time indicating how song length and song pace has changed over the seven decades.

- Interactive features allow user to select which decades in particular they want to view as part of the visualization.



2. Artist Exploratory Analysis

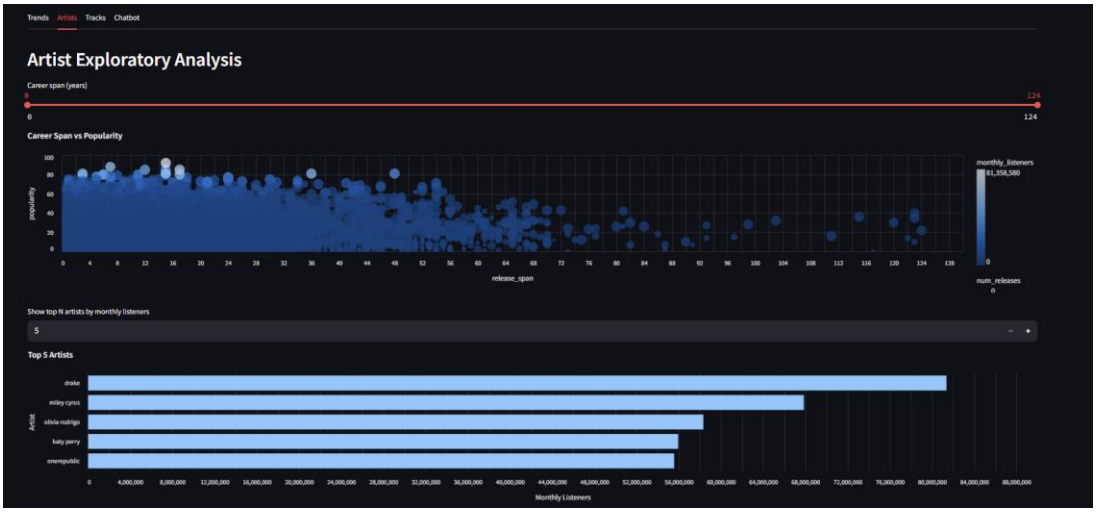
The “Artists” tab provides a closer look at the artists that make the music happen and some factors related to artists longevity:

- Career span vs. popularity scatterplots – allows users to interactively visualize artists from a certain era and how their popularity stands today.

- Top artists by monthly listeners – displays who are some of the modern day artists leading the charge and dominating the music world with their popularity digitally.

- Debut year distribution with filters – Shows how the dataset is skewed in terms of artist distribution and which era of artists is accurately captured within the dataset.

- Popularity vs. listeners by primary genre – gives users a peek into which genres of artists tend to do well in terms of popularity.

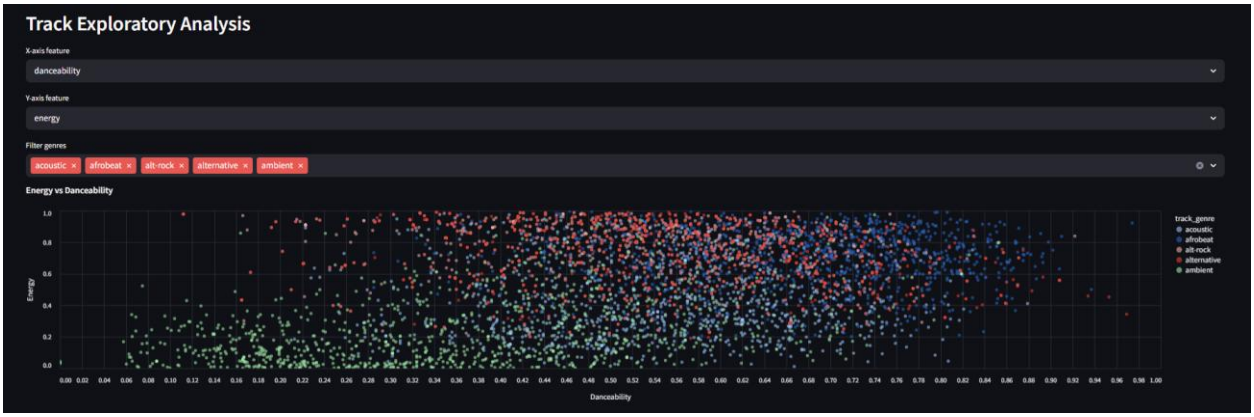


3. Track-Level Insights

The “Tracks” tab enables users with a detailed view of how the musical features correlate with each other and influence each other.

- Users can customize which two audio features they want to plot and scatter plots are between the two audio features to showing their influence on each other.

- Feature-wise genre ranking – users can visualize which audio features are predominant in which genres

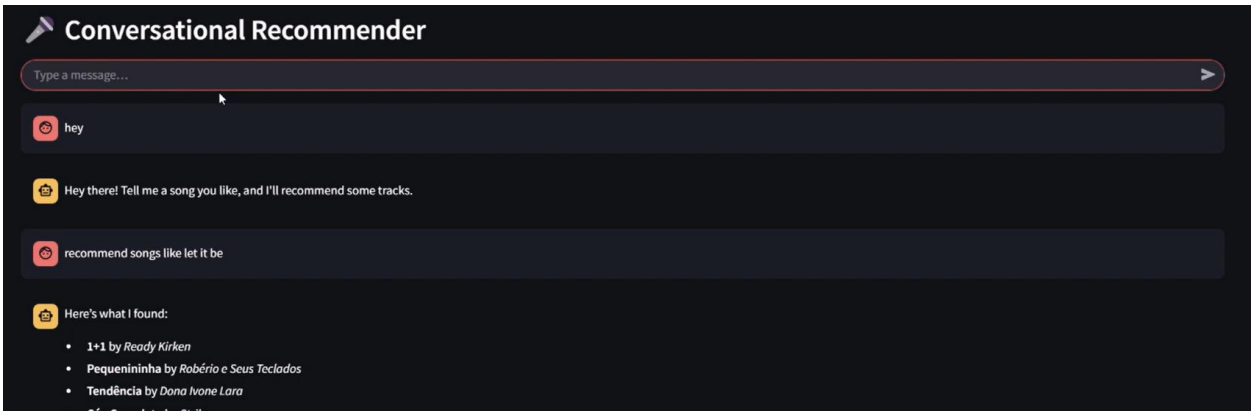


4. Conversational Recommender

The chatbot allows natural language input such as “Recommend songs like Shape of You” or “Suggest something similar to Blinding Lights”

Technical Details :

- We used spaCy framework to design our chatbot. A spaCy-based intent classifier is used to interpret user requests.
- Leverages cosine similarity over a precomputed feature matrix which includes leveraged features from Spotify and custom features we designed.
- Returns top 5 recommendations based on the input track
- Feature Matrix: Built using engineered features such as energy_acoustic_diff and valence_energy_diff, along with core audio features.
- Recommendation Logic: The chatbot uses a light-weight recommender based on cosine similarity, leveraging the models developed in earlier milestones.
- Cached data loading ensures fast response; the app remains responsive even on large datasets.
- Delivered a fully functional, user-interactive music recommendation system.
- Combined data analysis, ML modeling, and conversational AI into one unified product.



Overall Code Walkthrough –

We used altair for our interactive plots, seaborn and matplotlib to generate static plots and spacy to generate our intent classifier system.

1. We load and cache the data from our processed datasets so that they are only loaded once.

```
@st.cache_data
def load_track_data(path):
    df = pd.read_csv(path)

    numcols = ['popularity','duration_ms','danceability','energy','loudness',
               'speechiness','acousticness','instrumentalness','liveness',
               'valence','tempo']
    for c in numcols:
        df[c] = pd.to_numeric(df[c], errors='coerce')
    df['explicit'] = df['explicit'].map({'TRUE':True,'FALSE':False})
    return df

1 usage   new *
@st.cache_data
def load_train_processed(path):
    return pd.read_csv(path)
```

2. We build the feature matrix required for our recommendation system using a combination of the spotify defined metrics and our own custom engineered features and this is done only once using st.cache_data in order to prevent redundant operations.

```
@st.cache_data
def build_recommender(df):
    cols = ['danceability','energy','loudness','speechiness','acousticness',
            'instrumentalness','liveness','valence','tempo','duration_ms',
            'popularity','energy_acoustic_diff','valence_energy_diff']
    mat = df[cols].values
    return mat
```

3. We define our streamlit tabs and under each tab, create our visualizations using altair charts

```
tabs = st.tabs(["Trends","Artists","Tracks","Chatbot"])
```

Example code snippet of visualization for displaying insights based on audio features grouped by genre from our train.csv file–

```
with tabs[2]:
    st.header("Track Exploratory Analysis")

    feature_options = [
        'danceability','energy','loudness','speechiness',
        'acousticness','instrumentalness','liveness','valence','tempo'
    ]
    x_feat = st.selectbox("X-axis feature", feature_options, index=0)
    y_feat = st.selectbox("Y-axis feature", feature_options, index=1)

    # Genre filter
    genres = df_track['track_genre'].unique().tolist()
    sel_genres = st.multiselect("Filter genres", genres, default=genres[:5])

    filt_track = df_track[df_track['track_genre'].isin(sel_genres)]
```

```
scatter2 = (
    alt.Chart(filt_track)
       .mark_circle(opacity=0.6)
       .encode(
           x=alt.X( shorthand: f'{x_feat}:Q', title=x_feat.capitalize()),
           y=alt.Y( shorthand: f'{y_feat}:Q', title=y_feat.capitalize()),
           color='track_genre:N',
           tooltip=['track_name','artists',x_feat,y_feat,'track_genre']
       )
       .properties(height=400, width=700, title=f"{y_feat.capitalize()} vs {x_feat.capitalize()}")
       .interactive()
)
st.altair_chart(scatter2)
```

4. Our Recommendation System is built using the spaCy intent classifier
   We use st.session_state to track all the messages in our conversation between the user and the bot.

   We use the extract_intent() method to decipher user intent based on lemmas defined by spaCy

```
def extract_intent(text):
    doc = nlp(text)
    low = text.lower()
    if any(tok.lemma_ in ("hi","hello","hey") for tok in doc):
        return "greet"
    if "recommend" in low or "suggest" in low:
        return "recommend"
    if any(tok.lemma_ in ("bye","goodbye","see") for tok in doc):
        return "bye"
    return "fallback"
```

   And based on intent, bot replies appropriately to the user.
   If the intent is to recommend songs, we extract song name from the user input and pass it as a seed song to our recommender system and perform cosine similarity to extract five recommendations for the seed song.

```python
def recommend_by_name(song_name: str, df: pd.DataFrame, feature_matrix: np.ndarray, k: int = 5):
    mask = df['track_name'].str.lower() == song_name.lower()
    if not mask.any():
        return None
    idx = df[mask].index[0]
    sims = cosine_similarity(feature_matrix[idx:idx+1], feature_matrix).flatten()
    sims[idx] = -1
    rec_idxs = sims.argsort()[::-1][:k]

    return df.loc[rec_idxs, ['track_name', 'artists']]
```

6. Limitations -

Currently the model only accurately recommends songs only for the seed songs that are within our database of 100,000 tracks. In the future, as an extension it would be possible to have the recommendation system work for any kind of input seed song, if we integrate our web app with the Spotify web API and for any song that the user inputs, we can extract it's audio features that we have defined as part of the feature matrix from the API and then use our recommendation system to suggest songs based on those features. This would add additional complexity and computation time to our app but if optimally designed, can make our recommendation system even more foolproof.

7. Team Contributions and Justification

This project was completed collaboratively by **Hari Nair Suresh Chandran** and **Arjun Gopalakrishnan Kaliyath**, leveraging each team member's strengths to ensure an effective division of labor. Below is a clear breakdown of individual responsibilities:

Arjun Gopalakrishnan Kaliyath

- Led the feature engineering process, including the development of energy_acoustic_diff and valence_energy_diff.
- Developed and tested the Cosine Similarity and KNN-based recommendation models.
- Created evaluation scripts and implemented metrics such as Precision@5, Hit Rate, Diversity, and Feature Similarity.
- Designed and deployed the Streamlit dashboard, including trend visualizations and track-level analysis pages.
- Integrated the conversational recommender system using spaCy for intent parsing.

Hari Nair Suresh Chandran

- Managed all data preprocessing, including merging datasets, scaling, and categorical feature engineering.
- Implemented the Autoencoder and K-Means Clustering-based models, handling model architecture and cluster logic.
- Carried out EDA and created decade-wise analysis for music trends and artist career insights.
- Computed Recall@5, MAP@5, and NDCG@5, and led the interpretation and visualization of results.
- Compiled all components into a unified presentation and coordinated report writing.

Justification for two member team –
Given the scope, technical depth, and deliverables, this project required the involvement of two contributors.

- **Multiple Datasets**: Four different datasets were selected after an initial exploration of over 10 datasets with each dataset requiring separate processing, tuning, and evaluation. This resulted in extra efforts in the initial data preprocessing phase.
- **Tool Development**: Our aim was to build a comprehensive musical tool with analysis of trends, artist data insights and a recommendation system chatbot embedded within it which required capabilities of web development, data science and chatbot integration.

- **Evaluation Depth**: We aimed for a comprehensive model evaluation using 8+ metrics in order to accurately select our final model which required extensive custom metrics code, visual comparison, and performance interpretation.
- **Presentation and Documentation**: Delivering a user-friendly interface and a well-documented report alongside a technical backend demanded parallel efforts.

8. Sources :

1. **Kaggle Notebooks –** To study previous iterations of a content based music recommendation system.
    - https://www.kaggle.com/code/vatsalmavani/music-recommendation-system-using-spotify-dataset
    - https://www.kaggle.com/code/akershishukla/music-recommendation-system

2. **Medium articles** – Articles on content-based music recommendation systems.
    - https://medium.com/@shashwatdixit6311/content-based-music-recommendation-system-9db1245a04ea
    - https://medium.com/@ashu19/music-recommendation-system-using-machine-learning-8bc8cc52d143
    - https://medium.com/@shashwatdixit6311/content-based-music-recommendation-system-9db1245a04ea

3. **LLM Use –**
    - To gather ideas for evaluation metrics for testing model efficiency
    - To generate ideas for data modeling
    - To generate ideas for embedding streamlit dashboard with chatbot
    - To suggest optimal chatbot approach using Rasa, Dialogflow or spaCy
    - To resolve and debug errors in creation of streamlit dashboard and chatbot creation and data modelling.