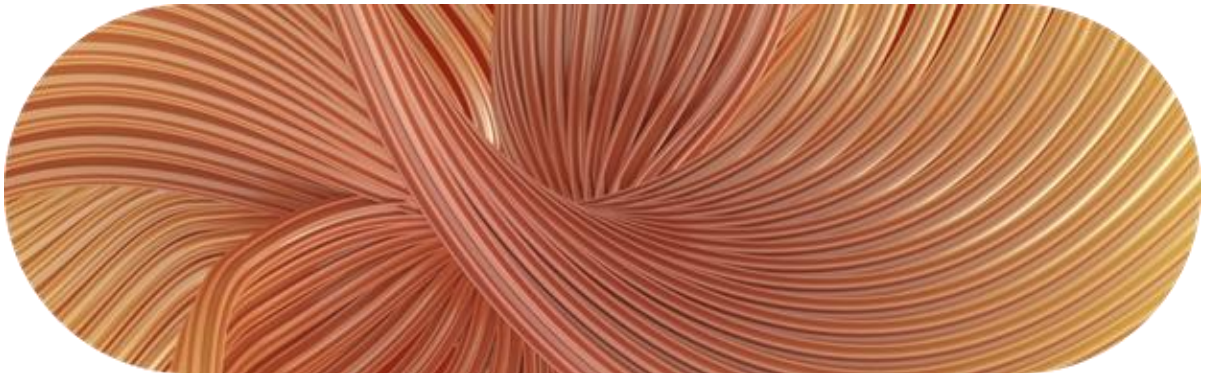
MUSIC
RECOMMENDER



MUSIC RECOMMENDER

This project involves creation of a music recommender system based on one's liking.

Team members: Caroline Muiruri {project maker}

Technologies : python

Creating a music recommender system based on a set of 'liked' songs involves several steps, including data collection, feature extraction, and recommendation algorithms. Here's a simplified outline of how you can build such a system using Python and common libraries:

1. **Data Collection**:

- Obtain a dataset of songs with their metadata (e.g., title, artist, genre) and audio features (e.g., tempo, danceability, energy) if available. You can use APIs like Spotify or manually curate a dataset.

2. **Data Preprocessing**:

- Clean and preprocess the data. Remove duplicates, missing values, and irrelevant columns.

- Normalize or scale the audio features so that they have the same weight during recommendation.

3. **Feature Extraction**:

- Extract relevant features from the audio data, such as tempo, danceability, and energy. You can use libraries like Librosa or Essentia for this.

4. **User Input**:

- Allow the user to input a set of 'liked' songs. This can be done through a user interface or by directly providing song IDs or features.

5. **Similarity Calculation**:

- Calculate the similarity between the 'liked' songs and all other songs in your dataset. Common similarity metrics include Euclidean distance, cosine similarity, or Pearson correlation.

6. **Recommendation Algorithm**:

- Implement a recommendation algorithm based on the calculated similarity scores. Here are a few common methods:

  - **Content-Based Filtering**: Recommend songs that are similar in audio features to the 'liked' songs.

  - **Collaborative Filtering**: Recommend songs based on the preferences of users who liked similar songs.

  - **Hybrid Methods**: Combine content-based and collaborative filtering for more accurate recommendations.

7. **Filtering and Ranking**:

- Filter out songs that the user has already liked to avoid recommending duplicates.

- Rank the recommended songs based on their similarity scores or other relevant criteria (e.g., popularity).

MUSIC
RECOMMENDER

8. **Presentation**:

   - Display the recommended songs to the user, preferably in a user-friendly interface.

9. **Feedback Loop**:

   - Allow users to provide feedback on the recommendations (e.g., thumbs up/down) to continuously improve the system.

10. **Testing and Evaluation**:

    - Evaluate the performance of your recommender system using metrics like precision, recall, or user engagement.

Here's a simplified Python code snippet to get you started with a content-based music recommender system:

```python
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity

# Load your dataset
data = pd.read_csv('music_data.csv')

# User's liked songs (you can replace these with actual data)
liked_songs = ["Song1", "Song2", "Song3"]

# Filter the dataset to include only the liked songs
```

```
liked_songs_data = data[data['Song Title'].isin(liked_songs)]


# Calculate cosine similarity between liked songs and all other songs

similarities = cosine_similarity(liked_songs_data.drop(['Song Title'], axis=1), data.drop(['Song Title'], axis=1))


# Get recommended songs based on similarity scores

recommended_songs = data.iloc[similarities.argmax(axis=1)]


# Display recommended songs to the user

print(recommended_songs)
```

Keep in mind that building a production-ready music recommender system involves more advanced techniques, scalability considerations, and user experience design. You may also want to explore collaborative filtering, matrix factorization, and deep learning methods to improve recommendations further.

Infrastructure

The choice of branching and merging strategy, deployment strategy, data population, and testing tools/processes can vary depending on the specific project and team's preferences. Here, I'll describe a common approach for each of these aspects:

1. Branching and Merging Strategy:

- **GitHub Flow**: GitHub Flow is a popular branching and merging strategy that emphasizes simplicity and continuous delivery. Here's a simplified process:

- Create a new branch for each feature or bug fix.

- Make changes and commit to the branch.

- Open a pull request (PR) for code review.

- Discuss and review code in the PR.

- Merge the PR into the main or master branch once it's approved.

- Deploy the main/master branch to a staging environment for testing.

- If testing is successful, deploy to production.

2. Deployment Strategy:

- **Continuous Integration and Continuous Deployment (CI/CD)**: Implement CI/CD pipelines using tools like Jenkins, Travis CI, or GitHub Actions to automate the deployment process.

- Use containerization (e.g., Docker) and orchestration (e.g., Kubernetes) for scalable and consistent deployments.

- Employ blue-green or canary deployment strategies to minimize downtime and mitigate risks during deployments.

- Use environment-specific configuration files to manage different deployment environments (e.g., development, staging, production).

3. Populating the App with Data:

- For development and testing, generate synthetic data or use a subset of real data.

- For production, you might use ETL (Extract, Transform, Load) processes to import data from various sources (e.g., databases, APIs, CSV files).

- Consider using database migration tools (e.g., Flyway, Alembic) to manage schema changes.

- Regularly back up and archive data to prevent data loss.

4. Testing Tools and Automation:

- **Unit Testing**: Use testing frameworks like PyTest (Python), JUnit (Java), or NUnit (.NET) for unit tests to verify individual components.

- **Integration Testing**: Test interactions between different components or services to ensure they work together correctly.

- **End-to-End Testing**: Implement end-to-end tests using tools like Selenium, Cypress, or Puppeteer to simulate user interactions.

- **Load Testing**: Use tools like Apache JMeter or locust.io to simulate heavy loads and measure system performance.

- **Static Code Analysis**: Employ tools like ESLint, Pylint, or SonarQube to identify code quality issues.

- **Continuous Monitoring**: Set up monitoring and alerting with tools like Prometheus, Grafana, or New Relic to detect issues in production.

- **Automation**: Integrate testing into the CI/CD pipeline to automate the testing process for every code change.

- **Code Reviews**: Implement code review processes to catch issues early and improve code quality.

Remember that the choice of tools and processes may vary depending on your technology stack, project requirements, and team preferences. It's essential to continuously iterate and improve your development and deployment practices based on feedback and evolving project needs.

In the context of developing an image compression algorithm using Transform coding, it's important to consider existing solutions and their similarities and differences. Here's how you can structure the "Existing Solutions" section:

Existing Solutions:

1. **JPEG (Joint Photographic Experts Group) Compression:**

   - **Similarities**: Like Transform coding, JPEG is a lossy compression technique commonly used for image compression. It involves transforming the image data into the frequency domain using the Discrete Cosine Transform (DCT).

- **Differences**: JPEG uses DCT to convert image blocks into frequency components, whereas Transform coding encompasses a broader category of techniques that includes various transformations beyond DCT.

2. **PNG (Portable Network Graphics) Compression:**

- **Similarities**: PNG is a lossless compression format that, like image compression algorithms in general, aims to reduce file size. It uses filtering and entropy coding techniques.

- **Differences**: Unlike Transform coding, PNG is lossless and doesn't transform image data into a frequency domain. It focuses on efficient encoding of pixel values and is better suited for images that require lossless compression, like graphics and logos.

3. **GIF (Graphics Interchange Format) Compression:**

- **Similarities**: GIF is a lossless or lossy compression format that, similar to image compression algorithms, aims to reduce file size. It uses LZW (Lempel-Ziv-Welch) compression.

- **Differences**: GIF is primarily used for animations and supports limited colors. It's less suitable for high-quality image compression compared to Transform coding techniques.

4. **WebP Compression:**

- **Similarities**: WebP is a modern image format that combines both lossless and lossy compression. It uses various encoding methods, including Transform coding.

- **Differences**: While WebP incorporates Transform coding, it also employs other techniques like predictive coding and entropy coding. It's designed to provide a balance between compression efficiency and image quality.

Choice to Reimplement Transform Coding:

Transform coding, as a category of techniques, includes various transformations beyond DCT. These transformations can be tailored to the specific requirements of the project. Choosing to reimplement Transform coding might be based on:

MUSIC
RECOMMENDER

1. **Customization**: Existing solutions like JPEG are standardized and may not allow for fine-grained customization to meet specific project needs. Reimplementing Transform coding allows you to tailor the transformation and encoding process to your exact requirements.

2. **Performance**: Depending on the nature of the images and the target devices, reimplementing Transform coding can optimize compression performance by choosing a specific transformation that suits the image characteristics.

3. **Flexibility**: Transform coding provides flexibility in selecting the transformation method (e.g., DCT, wavelets, or others) and quantization techniques, which can be adjusted based on the trade-off between compression ratio and image quality.

4. **Patent or Licensing Considerations**: Some existing compression algorithms, like JPEG, may have patent or licensing restrictions. Reimplementing Transform coding can help avoid such issues and provide more control over intellectual property.

In summary, while existing image compression solutions like JPEG, PNG, GIF, and WebP offer valuable features, reimplementing Transform coding can provide greater flexibility and customization options to meet specific project requirements and optimize image compression for different use cases.