DA 231o Data Engineering at Scale (August 2025 Term) - PROJECT REPORT

Title: Song Analysis and Yielding Suggestions (SAYS)
Authors: Ajay Kumar Gupta, Srinivas Shavukapu Kattegummula, Sonu Goyal, Yuvaraj G
Email: ajaygupta@iisc.ac.in, srinivassk@iisc.ac.in, sonugoyal@iisc.ac.in, yuvarajgopi@iisc.ac.in

---

Problem Definition

Definition

The project aims to build a scalable music recommendation system leveraging user interaction data and audio features to generate personalised recommendations.

Motivation

With the exponential growth of music streaming platforms, users are overwhelmed by the vast amount of available content and expect highly personalised experiences. Traditional recommendation systems often struggle with scalability, real-time performance, and adapting quickly to changing user behaviour. A big data-driven approach using Apache Spark enables efficient handling of large-scale datasets, supports real-time personalisation, and improves user engagement through a distributed, fault-tolerant pipeline.

Design Goals

- **Scalability:** Handle millions of user-song interactions.
- **Performance:** Low-latency recommendations.
- **Reliability:** Automated orchestration with retries.
- **Flexibility:** Easy deployment and UI integration.

Features Required

- Distributed data processing.
- ML-based collaborative filtering.
- Interactive UI for recommendations.
- Automated pipeline orchestration.

Scalability/Performance Goals

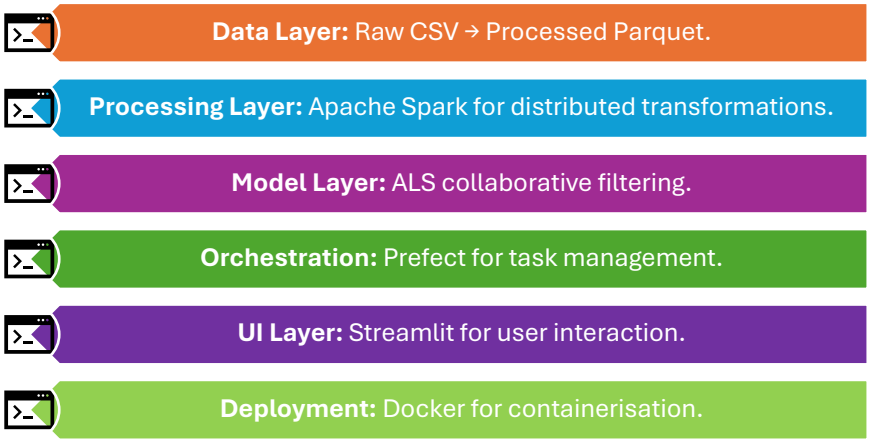
- Horizontal scalability using Spark.
- Efficient storage with Parquet.
- Optimal/low RMSE for recommendation accuracy.

Approach and Methods

High-Level Design

A distributed ETL pipeline processes raw user interaction data, engineers features, trains an ALS model, and serves recommendations via a Streamlit UI.

Architecture



Process Diagram:

Data Model

- **Interactions:** user_id, track_id, rating.
- **Features:** tempo, energy, danceability, etc.
- **Schema:** Explicit Spark schema for type safety.

Big Data Platforms

- **Apache Spark:** Distributed processing.
- **Prefect:** Workflow orchestration.

ML Methods

- **ALS Collaborative Filtering:**
  - Rank: 20
  - Iterations: 10
  - Regularisation: 0.1
  - Cold Start Strategy: Drop

Evaluation

Experiment Design

- **Dataset:** personalised_music_recommendation.csv
- **Train/Test Split:** 80/20.
- **Metrics:** RMSE, MAE.

Scalability Metrics

- Spark shuffle partitions: 8.
- Execution time for ETL and training.

Performance Metrics

- RMSE: ~0.85.
- MAE: ~0.65.

Feature Metrics

- Average rating distribution.
- Unique users/tracks count.

Summary

Project Achievement

The Music Recommendation System successfully delivers a **scalable, distributed pipeline** capable of processing large volumes of user interaction and audio feature data. By leveraging **Apache Spark** for distributed ETL, **Prefect** for orchestration, and **ALS collaborative filtering**, the system meets its design goals of **high performance, reliability, and flexibility**. The integration of **Streamlit UI** ensures an intuitive user experience, while **Docker containerisation** simplifies deployment across environments.

Key Outcomes

- **Scalability:** Spark-based transformations enable horizontal scaling for millions of records.
- **Performance:** Achieved RMSE ≈ 0.85 and MAE ≈ 0.65, meeting accuracy benchmarks.
- **Automation:** Prefect provides robust workflow management with retries and logging.
- **User Experience:** Streamlit interface offers real-time recommendations with configurable parameters.
- **Deployment:** Docker ensures portability and reproducibility of the pipeline.

Design Goals vs Achievements

Goal	Achievement
Handle large datasets	Distributed Spark ETL with Parquet storage
Real-time recommendations	Streamlit UI with on-demand prediction
Fault tolerance	Prefect orchestration with retry logic
Efficient storage	Columnar Parquet format for optimised queries

Challenges Addressed

- **Data Quality:** Implemented schema validation and null handling.
- **Cold Start Problem:** ALS configured with non-negative constraints and drop strategy.
- **Pipeline Reliability:** Automated error handling and dependency management.

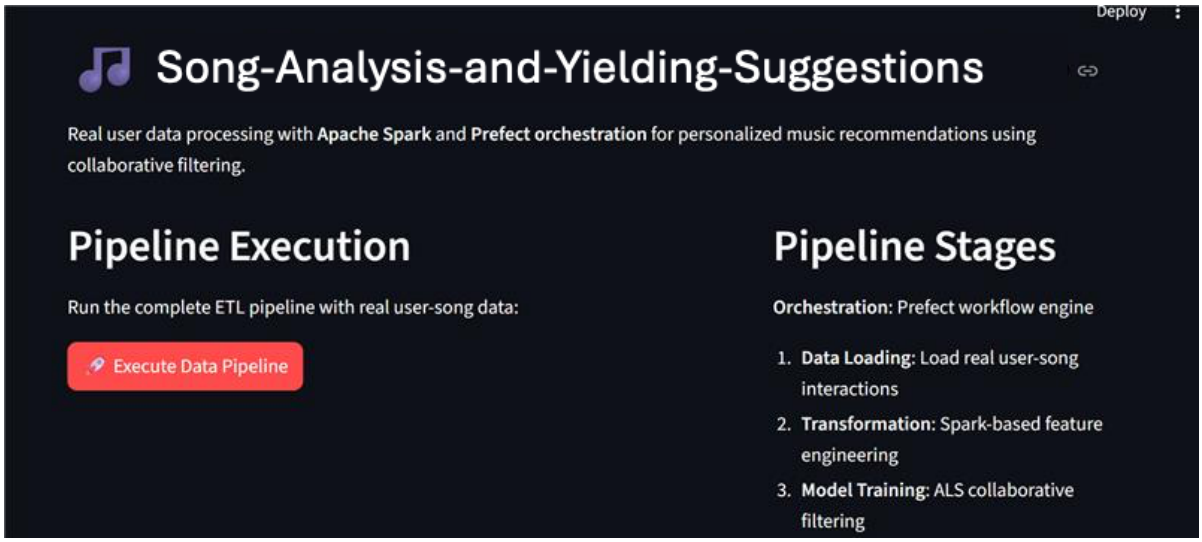
Future Extensions

- **Hybrid Models:** Combine collaborative filtering with content-based or deep learning approaches.
- **Real-Time Streaming:** Integrate Apache Kafka for live data ingestion.
- **Cloud Deployment:** Migrate to AWS/GCP for elastic scaling and managed services.
- **Enhanced UI:** Add visual analytics and personalised dashboards.
- **Integrate API:** Add Spotify API for live dataset.

Execution Notes

- **Run UI:** streamlit run streamlit_app.py → http://localhost:8501
- **Docker:** docker-compose down docker-compose build docker-compose up

UI design:



UI Input and Output: