1. Introduction

This document provides a detailed explanation of the data analysis, training process, and the chosen model for the **Pratilipi Story Recommendation System**. The goal of this system is to predict which Pratilipis (stories) each user is likely to read in the future, allowing us to recommend five relevant stories based on their past reading behavior.

2. Data Analysis

2.1 Dataset Description

We used two datasets:

1. **user_interaction.csv**
    - **user_id**: Unique identifier of the user.
    - **pratilipi_id**: Unique identifier of the pratilipi (story).
    - **read_percentage**: How much of the pratilipi the user has read (0–100).
    - **updated_at**: Timestamp when the interaction occurred.
2. **metadata.csv**
    - **author_id**: Unique identifier of the pratilipi's author.
    - **pratilipi_id**: Unique identifier of the pratilipi (story).
    - **category_name**: Category of the pratilipi (e.g., Romance, Mystery).
    - **reading_time**: Estimated reading time in seconds.
    - **updated_at**: Timestamp for the last update of the metadata.
    - **published_at**: Timestamp for when the pratilipi was published.

2.2 Key Insights

1. **Sparsity**: Most users read only a small percentage of the total pratilipis available.
2. **Popularity Bias**: A small subset of pratilipis receives the majority of reads.
3. **Dominant Category**: Romance is the most-read category, followed by Mystery and Thriller.
4. **Active Users**: Users who frequently read pratilipis tend to read a larger percentage of each.

These insights guided our decisions on data preprocessing steps and model choice.

3. Training Process

3.1 Data Preprocessing

1. **Cleaning & Missing Values**: Removed rows with missing or invalid fields (e.g., null user IDs) and converted timestamps into proper datetime objects.
2. **Filtering**: Excluded interactions where read_percentage < 10% to minimize noise.
3. **Merging**: Joined user_interaction.csv with metadata.csv if needed for additional features (such as category_name).

3.2 Train–Test Split

- We split the dataset using a **time-based** approach to avoid data leakage:
    - **Training**: The first 75% of the timeline.
    - **Testing**: The last 25% of the timeline.
- This ensures that predictions only use data from prior user interactions to forecast future reads.

4. Chosen Model

4.1 Collaborative Filtering (SVD)

We implemented a **Singular Value Decomposition (SVD)** model using the Surprise library. SVD is well-suited for large, sparse datasets typical of user–item (in this case, user–pratilipi) interactions.

Why SVD?

- **Handles Sparsity**: Real-world recommendation datasets are usually sparse. SVD can manage this by factorizing the user–item rating matrix into lower-dimensional latent factors.
- **Performance**: SVD-based methods often provide high-quality recommendations while remaining relatively efficient.

4.2 Model Evaluation

- **Root Mean Squared Error (RMSE)**: Used to measure the difference between the predicted and actual read percentages (or implicit ratings).
- **Precision@k and Recall@k** (we used k=5 in this assignment):
    - **Precision@5**: Out of the top 5 recommended pratilipis, how many were actually relevant (i.e., the user read them)?
    - **Recall@5**: Out of the pratilipis the user actually read (in the test period), how many appeared in the top 5 recommendations?

Results:

- **Before Optimization**: RMSE ~22.09
- **After Hyperparameter Tuning**: RMSE ~9.21

5. Future Improvements

1. **Hybrid Approaches**: Combine collaborative filtering with content-based methods (e.g., using pratilipi metadata like category, reading time) to handle cold-start scenarios for new pratilipis or new users.
2. **Neural Models**: Explore deep learning architectures (e.g., autoencoders or neural collaborative filtering) that can capture more complex user–item relationships.
3. **Scalability & Deployment**: Host the recommendation service on AWS or GCP to handle real-time recommendation requests at scale.