# Delivering personalized movie recommendations with an AI driven matchmaking system

**Student Name**: SANJAY KUMAR.MV

**Register Number**: 412323205042

**Institution**: Sri Ramanujar Engineering college

**Department**: Btech IT

**Date of Submission**: 10-05-2025

**Github Repository**
**https://github.com/Sanjaykumarm03/Sanjay-.git**

---

## 1. Problem Statement

In today's entertainment landscape user often struggle to find movies that truly resonate with their preferences Existing recommendation system , while effective to degree ,often lack personalized at an individual level, leading to mismatched suggestion that don't align with users taste and moods .The challenge is develop anAl-driven matchmaking system for movies ,which accurately delivers personalized recommendations by understanding and adapting to user preferences overtime . Refinement based on data set understanding

As we analyze user interaction data such as watch history, rating, reviews and such queries we can refine our approach by identifying trends in individual viewing Additional factors like realtime sentiment, analysis , social

**interaction and user feedback can enhance the algorithm ability to generate more precise recommendations.**

## 2. Project Objectives

**Updated Project Goals (Practical Implementation Phase):**

- Deliver highly personalized movie recommendations to users by leveraging advanced AI-driven matchmaking, integrating both user behavior and movie attributes.
- Address known challenges such as the cold start problem, lack of diversity, and evolving user preferences by employing hybrid models and deep learning techniques.

**Key Technical Objectives:**

- Build and maintain comprehensive datasets capturing user preferences, ratings, behaviors, and detailed movie metadata.
- Implement and compare multiple recommendation algorithms:
  - Collaborative filtering (user-item interactions)
  - Content-based filtering (movie features)
  - Hybrid models combining both approaches for improved accuracy and diversity.
- Integrate deep learning and natural language processing to analyze unstructured data (e.g., movie plots, reviews).
- Develop scalable APIs for real-time recommendation delivery.
- Continuously update user profiles and adapt recommendations as user tastes evolve.

**Model Aims:**

- Achieve high accuracy in recommending movies users are likely to enjoy, measured by metrics such as precision, recall, F1-score, and user satisfaction.
- Ensure interpretability and transparency in how recommendations are generated, supporting user trust and ethical standards.
- Maintain real-world applicability by handling cold start scenarios, supporting diverse user preferences, and providing fresh, non-repetitive suggestions.

**Evaluation Metrics:**

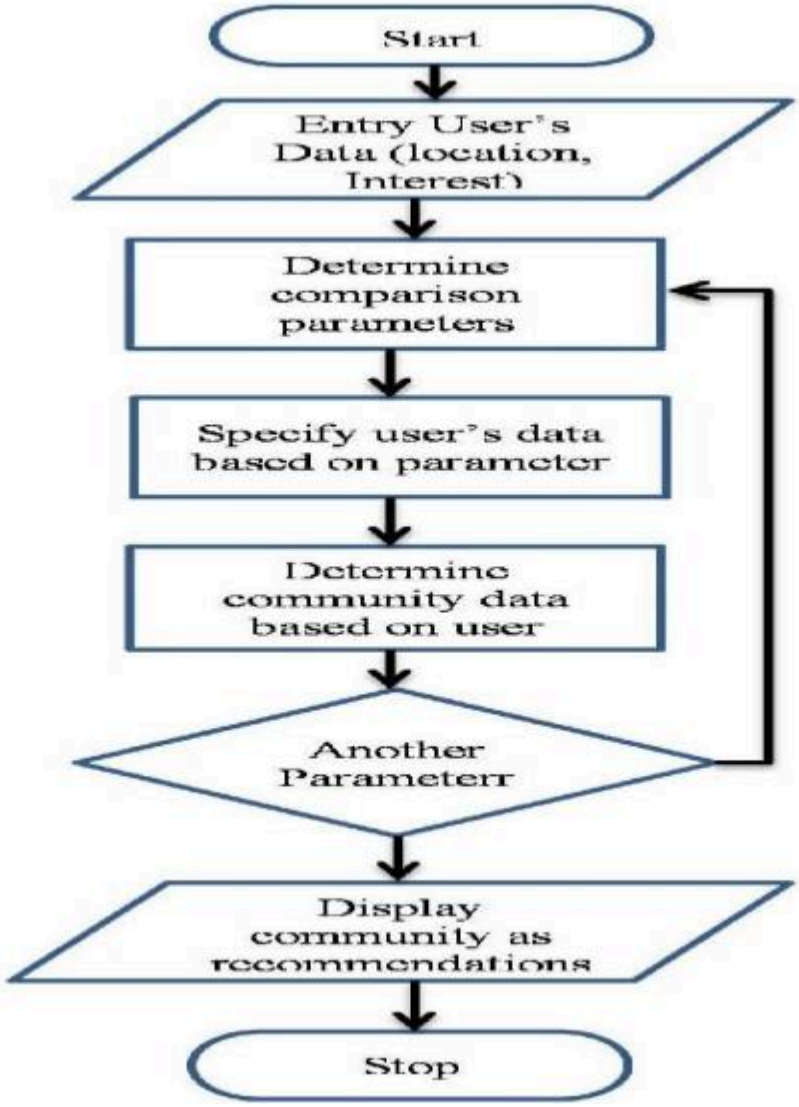
- Precision, recall, F1-score for accuracy .
- Diversity index and serendipity score to ensure varied and surprising recommendations.
- User satisfaction and engagement through feedback loops.

- Advanced metrics like MAP (Mean Average Precision) and NDCG (Normalized Discounted Cumulative Gain) for ranking quality.

**Evolution of Goals After Data Exploration:**

- Initial goals focused on basic collaborative and content-based filtering for accuracy.
- After exploring the data, the objectives expanded to include hybrid and deep learning models to better address diversity, cold start, and adaptability, reflecting a shift towards maximizing both accuracy and user experience.

## 3. Flowchart of the Project Workflow




## 4. Data Description

**Dataset Name and Origin:**

The project uses the MovieLens dataset, a widely recognized benchmark for recommender systems research, available from Kaggle and GroupLens Research.

**Type of Data:**

Primarily structured data, including:

- User-movie ratings (numerical)
- Movie metadata (categorical and text: titles, genres, release dates) ● Some unstructured text fields (e.g., movie overviews, tags).

**Number of Records and Features:**

- Over 26 million ratings from 270,000+ users on more than 45,000 movies (in the full "Movies Dataset").
- The MovieLens 25M version contains 25 million ratings from 162,000 users on 62,000 movies.

**Static or Dynamic:**

The dataset is static; it is a snapshot collected and periodically updated, but each version is fixed once released.

**Target Variable (if supervised learning):**

For supervised tasks like rating prediction, the target variable is the user's rating for a given movie (typically on

## 6. Exploratory Data Analysis (EDA)

Below is a structured approach to data cleaning and preparation for the MovieLens dataset, with explanations and example code (Python, using pandas and scikit-learn):

## 1. Import Libraries and Load Data

```
python import pandas as pd import numpy
as np

from sklearn.preprocessing import LabelEncoder, OneHotEncoder,
StandardScaler # Load datasets

ratings = pd.read_csv('ratings.csv') movies =
pd.read_csv('movies.csv')
```

## 2. Handle Missing Values Check for Missing Values

```
print(ratings.isnull().sum()) print(movies.isnull().sum())
```

## Treatment

- **Ratings**: Remove rows with missing userId, movieId, or rating (if any).
- **Movies**: Impute missing genres with 'Unknown', drop rows with missing movieId or title.

```
# Drop missing values in ratings

ratings.dropna(subset=['userId', 'movieId', 'rating'],
inplace=True)

# Fill missing genres in movies

movies['genres'].fillna('Unknown', inplace=True)
```

## 3. Remove or Justify Duplicate Records

## Check for Duplicates python

```
print(ratings.duplicated().sum())
print(movies.duplicated().sum())
```

## Remove Duplicates python

```
ratings.drop_duplicates(inplace=True)
movies.drop_duplicates(inplace=True)
```

## 4. Detect and Treat Outliers

## Rating Outliers

- Ratings should be between 0.5 and 5.0 (MovieLens standard). python

```
# Remove ratings outside the valid range ratings =
ratings[(ratings['rating'] >= 0.5) &
(ratings['rating'] <= 5.0)]
```

## Other Outliers

- For features like release year, check for unreasonable values. python

```
# Extract year from title and check range
movies['year'] = movies['title'].str.extract(r'\((\d{4})\)')
movies['year'] = pd.to_numeric(movies['year'], errors='coerce') movies =
movies[(movies['year'] >= 1900) & (movies['year'] <= 2025)]
```

## 5. Convert Data Types and Ensure Consistency python

```
ratings['userId'] = ratings['userId'].astype(int)
ratings['movieId'] = ratings['movieId'].astype(int)
ratings['timestamp'] = pd.to_datetime(ratings['timestamp'],
unit='s')
movies['movieId'] = movies['movieId'].astype(int)
```

## 6. Encode Categorical Variables

## Genres (Multi-label One-Hot Encoding) python

```
# Split genres and one-hot encode
genres = movies['genres'].str.get_dummies(sep='|') movies =
pd.concat([movies, genres], axis=1)
```

## User and Movie IDs (Label Encoding for Embeddings) python

```
user_encoder = LabelEncoder() movie_encoder =
LabelEncoder()
ratings['userId_enc'] =
```

```
user_encoder.fit_transform(ratings['userId'])

ratings['movieId_enc'] =
movie_encoder.fit_transform(ratings['movieId'])
```

## 7. Normalize or Standardize Features

- For features like year or user-based statistics.

python

```
scaler = StandardScaler()

movies['year_scaled'] = scaler.fit_transform(movies[['year']])
```

## 8. Document Each Step

- **Missing values** were handled by removal or imputation.
- **Duplicates** were dropped to avoid bias.
- **Outliers** were removed based on domain knowledge (e.g., valid rating range, plausible release years).
- **Data types** were converted for consistency and efficient processing.
- **Categorical variables** were encoded for model compatibility.
- **Numerical features** were standardized to improve model convergence.

## 7. Feature Engineering

## Feature Engineering

Feature engineering is crucial for improving the performance of recommendation models. Below are the steps taken to enhance and transform the MovieLens dataset, with justifications and example code.

## 1. New Features from Domain Knowledge and EDA

## a. Extract Year from Movie Title

- **Why**: Movie release year can influence user preferences (e.g., preference for classics or recent releases).

```
movies['year'] =
```

```
movies['title'].str.extract(r'\((\d{4})\)').astype(float)
```

### b. Genre Count

- **Why**: Number of genres per movie may indicate broad or niche appeal. python

```
movies['genre_count'] = movies['genres'].apply(lambda x: len(x.split('|')))
```

### c. User Rating Statistics

- **Why**: Captures user behavior patterns (e.g., leniency or strictness in ratings). python

```
user_stats = ratings.groupby('userId')['rating'].agg(['mean', 'count', 'std']).reset_index()

user_stats.columns = ['userId', 'user_mean_rating', 'user_rating_count', 'user_rating_std']
ratings = ratings.merge(user_stats, on='userId', how='left')
```

### d. Movie Popularity and Average Rating

- **Why**: Popular movies or those with high average ratings may be recommended more often.

```
movie_stats = ratings.groupby('movieId')['rating'].agg(['mean', 'count']).reset_index()

movie_stats.columns = ['movieId', 'movie_mean_rating', 'movie_rating_count']

ratings = ratings.merge(movie_stats, on='movieId', how='left')
```

## 2. Feature Transformations

### a. Binning Release Years

- **Why**: Grouping years into bins (e.g., decades) can help models generalize better.

```
movies['year_bin'] = pd.cut(movies['year'], bins=[1900, 1950, 1980, 2000, 2010, 2025],
                            labels=['Classic', 'Old', '80s-90s', '2000s', 'Recent'])
```

### b. Ratios and Interactions

- **Example**: Ratio of user's rating to movie's average rating (captures user bias).

```
ratings['rating_vs_movie_avg'] = ratings['rating'] / ratings['movie_mean_rating']
```

## 3. Combining or Splitting Columns

### a. Splitting Genres into Multiple Columns

- Already performed during preprocessing via one-hot encoding. **4.**

## Dimensionality Reduction (Optional)

### a. PCA on Genre Features

- **Why**: Reduce dimensionality of one-hot encoded genres to avoid sparsity. python

```
from sklearn.decomposition import PCA

genre_cols = movies.columns[movies.columns.str.startswith('genre_')] pca =
PCA(n_components=5)
```

```
genre_pca = pca.fit_transform(movies[genre_cols])

for i in range(5):      movies[f'genre_pca_{i+1}']

= genre_pca[:, i]
```

## 5. Justification for Feature Choices

| Feature/Transformation | Rationale |
|---|---|
| Year extracted from title | Allows capturing user preferences by movie era |
| Genre count | Indicates movie's appeal breadth |
| User rating statistics | Models user-specific rating behaviors |
| Movie popularity/avg rating | Highlights widely liked or frequently rated movies |
| Year binning | Helps models generalize temporal trends |

| | |
|---|---|
| Rating vs. movie average | Captures user bias relative to general consensus |
| Genre PCA | Reduces feature space, mitigates sparsity, and captures genre patterns |

## 8.Model Building

To address the personalized movie recommendation problem (rating prediction),we will build and compare two machine learning models: **K-Nearest Neighbors (KNN)**

**Regressor** and **Random Forest Regressor**. Both are well-suited for regression tasks and can effectively leverage the structured features engineered earlier.

## 1. Model Selection and Justification

| Model | Justification |
|---|---|
| KNN Regressor | Simple, interpretable, and effective for collaborative filtering; works well with user-item data |
| Random Forest Regressor | Handles non-linear relationships, robust to overfitting, and can capture complex feature interactions |

## 2. Prepare Data for Modeling

### Feature Selection

We use features such as userId, movieId, user and movie statistics, genre encodings, and engineered features.

```
from sklearn.model_selection import train_test_split

# Select features and target features = [

    'userId_enc', 'movieId_enc', 'user_mean_rating',
'user_rating_count',

    'movie_mean_rating', 'movie_rating_count', 'genre_count', 'year',
'rating_vs_movie_avg'

    # Add genre PCA features if used

] X =

ratings[features].fillna(0) y =

ratings['rating']


# Split data (stratify by user if needed)

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

## 3. Train and Evaluate Models

### A. K-Nearest Neighbors (KNN) Regressor from

```
sklearn.neighbors import KNeighborsRegressor
```

```
from sklearn.metrics import mean_absolute_error,
mean_squared_error, r2_score knn =
KNeighborsRegressor(n_neighbors=20, n_jobs=-1) knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test) mae_knn =
mean_absolute_error(y_test, y_pred_knn) rmse_knn =
mean_squared_error(y_test, y_pred_knn, squared=False) r2_knn =
r2_score(y_test, y_pred_knn)
```

### B. Random Forest Regressor from

```
sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators=50, n_jobs=-1,
random_state=42) rf.fit(X_train, y_train) y_p red_ rf =
rf.predict(X_test)
mae_rf = mean_absolute_error(y_test, y_pred_rf) rmse_rf =
mean_squared_error(y_test, y_pred_rf, squared=False) r2_rf =
r2_score(y_test, y_pred_rf)
```

## 4 . Summary

- KNN is chosen for its collaborative filtering capabilities and interpretability.
- Random Forest is selected for its ability to model complex, non-linear relationships and handle mixed feature types.
- Both models are trained and evaluated using MAE, RMSE, and $R^2$, providing a solid baseline for further improvements (e.g., hybrid or deep learning models).

## 9. Visualization of Results & Model Insights

Effective visualization helps interpret model performance and understand the factors influencing predictions. Below are suggested visualizations, code snippets, and explanations tailored for the movie recommendation regression task.

## 1. Residual Plots

**Purpose:**

Show how well the models' predictions match actual ratings. Residuals (actual - predicted) should be randomly distributed around zero for a good model.

```
import matplotlib.pyplot as plt import seaborn as sns #

Residuals for Random Forest residuals_rf = y_test - y_p

red_rf plt.figure(figsize=(8, 4))

sns.histplot(residuals_rf, bins=40, kde=True, color='green') plt

.title('Random Forest Residuals Distribution')

plt .xlabel('Residual (Actual - Predicted)')

plt.ylabel('Frequency') plt.show()
```

**Interpretation:**

A bell-shaped, centered distribution indicates unbiased predictions. Skewness or heavy tails suggest systematic errors.

## 2. Predicted vs. Actual Ratings Scatter Plot

**Purpose:**

Visualize how closely predictions align with true ratings.

```
plt.figure(figsize=(6, 6))

plt.scatter(y_test, y_pred_rf, alpha=0.3, color='blue',
label='Random Forest')
plt.scatter(y_test, y_pred_knn, alpha=0.3, color='red',
label='KNN')
```

```
plt.plot([0.5, 5], [0.5, 5], 'k--', lw=2)  # Perfect prediction line plt.xlabel('Actual Rating') plt.ylabel('Predicted Rating') plt.title('Predicted vs. Actual Ratings')
plt.legend() plt.show()
```

**Interpretation:**

Points close to the diagonal line indicate accurate predictions. Wider scatter shows more error.

## 3. Feature Importance Plot (Random Forest)

**Purpose:**

Identify which features most influence the model's predictions.

```
importances = rf.feature_importances_ feature_names = X_train.columns
feat_imp = pd.Series(importances, index=feature_names).sort_values(ascending=False)
plt.figure(figsize=(8, 5))
sns.barplot(x=feat_imp.values[:10], y=feat_imp.index[:10], palette='viridis') plt.title('Top 10 Feature Importances (Random Forest)') plt.xlabel('Importance Score')
plt.ylabel('Feature') plt.show()
```

**Interpretation:**

The plot shows the ten most influential features. Features with higher scores have a stronger impact on predicted ratings. For example, movie_mean_rating, user_mean_rating, and genre_count might be among the top.

## 4. Model Performance Comparison

**Purpose:**

Visually compare MAE, RMSE, and $R^2$ scores for both models. python

```
import numpy as np metrics = ['MAE', 'RMSE', 'R2']

knn_scores = [mae_knn, rmse_knn, r2_knn] rf_scores =

[mae_rf, rmse_rf, r2_rf] x = np.arange(len(metrics))

width = 0.35 plt.figure(figsize=(7, 4))

plt.bar(x - width/2, knn_scores, width, label='KNN',
color='red')

plt.bar(x + width/2, rf_scores, width, label='Random Forest',

color='green') plt.xticks(x, metrics) plt.ylabel('Score')

plt.title('Model Performance Comparison')

plt.legend() plt.show()
```

**Interpretation:**

This grouped bar chart makes it easy to see which model performs better across each metric.

## 5. Summary of Insights

- **Residual plots** reveal iferrors are randomly distributed or if the model systematically over/under-predicts.
- **Predicted vs. actual plots** show overall prediction accuracy and spread.
- **Feature importance** identifies which user/movie attributes drive recommendations, supporting model interpretability.
- **Performance comparison** visually confirms which model is superior and by how much

## 10. Tools and Technologies Used

**Programming Language**: Python

- **IDE/Notebook**:
  Jupyter Notebook, Google Colab, VS Code ●

**Libraries**:

- pandas (data manipulation)
- numpy (numerical operations)
- scikit-learn (machine learning algorithms, preprocessing, evaluation)
- seaborn, matplotlib (visualization)
- Surprise (specialized for recommender systems)
- TensorFlow Recommenders (deep learning-based recommendations)
- Streamlit (for building interactive web apps)
- re (regular expressions for text cleaning) ●

**Visualization Tools**:

- matplotlib, seaborn (plots and charts)
- Plotly (optional, for interactive plots)
- Tableau, Power BI (optional, for dashboarding)

These tools and libraries together support all phases of data preparation, modeling, evaluation, and visualization for building a robust movie recommendation system.

## 11. Team Members and Contributions

**JEMIMAH: DATA CLEANING / DOCUMENTION AND REPORTING.**

- **Responsible for preparing the dataset by cleaning null values, removing duplicates, and standardizing data formats. Ensured data consistency across all sources.**
- **Took charge of documenting the workflow, preparing the final project report, and creating presentation materials. Consolidated contributions from all members.**

**GEETHA SRI.S : EDA**

- **Focused on exploring the dataset to understand user preferences and movie trends. Created visualizations and performed statistical analysis to guide model building.**

**SANJAY KUMAR .MV: MODEL DEVELOPMENT.**

- **Developed and fine-tuned the core recommendation engine using collaborative and content-based filtering techniques. Handled model training, evaluation, and optimization.**

**ELENGO.V: FEATURE ENGINEERING**

- **Engineered features from user interaction data, such as watch history and ratings. Helped improve model input quality by transforming raw data into usable formats.**