Phase-3 Submission

Student Name: Bharath Kumar.L

Register Number: 410723104008

Institution: Dhanalakshmi College Of Engineering

Department: B.E (C.S.E)

Date of Submission: 14.05.2025

GitHub Link: https://github.com/Bharathhz/NM_BHARATH

Delivering personalized movie recommendations with an AI-driven matchmaking system

1. Problem Statement

In an era of content overload, users struggle to discover movies that match their unique tastes. Traditional recommendation systems often fail to provide truly personalized suggestions. Our goal is to build an AI-driven matchmaking system that delivers highly tailored movie recommendations by analyzing user preferences, viewing history, and behavioral patterns. This is a recommendation system problem that incorporates elements of clustering, content-based filtering, and collaborative filtering.

2. Abstract

The project aims to address the challenge of content discoverability in the movie domain. We propose an AI-driven matchmaking system capable of delivering personalized movie recommendations. By leveraging machine learning models and user interaction data, the system identifies user personas and aligns them with suitable movie profiles. The project involves preprocessing movie metadata, analyzing user behavior, engineering features, and real-time user engagement.

3. System Requirements

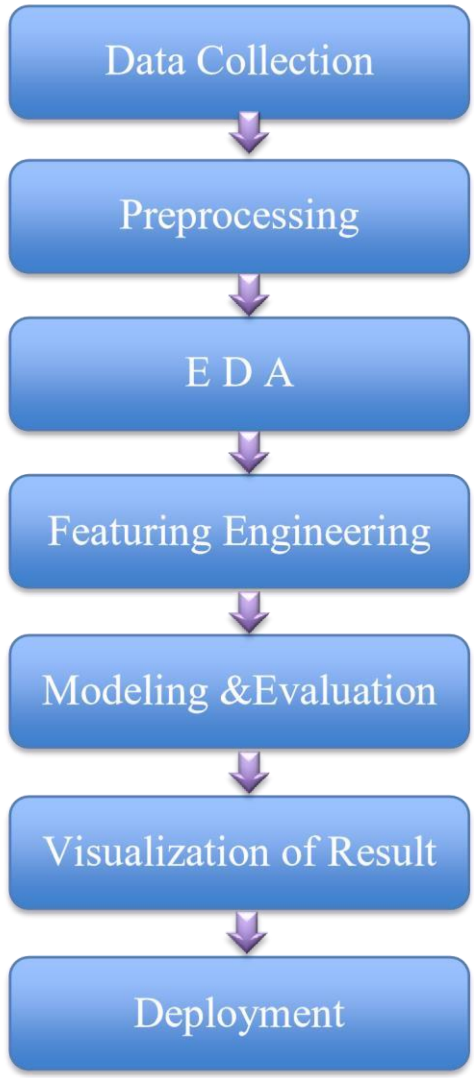
Hardware:

- Minimum 8 GB RAM
- Intel i3 processor or equivalent **Software:**
- Python 3.8+
- Jupyter Notebook or Google Colab
- Libraries: pandas, numpy, scikit-learn, matplotlib, seaborn.

4. Objectives

- Deliver personalized movie recommendations based on user preferences
- Identify user behavior patterns through clustering and analysis
- Deploy an interactive web application for live recommendations
- Improve user engagement and satisfaction in content discovery

5. Flowchart of Project Workflow



6. Dataset Description

- **Dataset Source:** Movie Recommendation System from Kaggle
- **Type of Data:** Structured
- **Features:** User ID, Movie ID, Ratings, Timestamps, Movie Genres
- **Records:** ~100,000 ratings across 943 users and 1,682 movies
- **Dynamic/Static:** Static snapshot
- **Target Variable:** Predicted rating or ranked list of recommended movies

Dataset Link: [Movie Recommendation System]

- Sample dataset (movies.head())

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

- Sample dataset (ratings.head())

	userId	movieId	rating	timestamp
0	1	296	5.0	1.147880e+09
1	1	306	3.5	1.147869e+09
2	1	307	5.0	1.147869e+09
3	1	665	5.0	1.147879e+09
4	1	899	3.5	1.147869e+09

7. Data Preprocessing

- Removed duplicate entries
- Converted timestamp to datetime format
- Encoded genres using multi-hot encoding
- Handled missing values (none in this dataset)
- Merged ratings and movie metadata
- Standardized rating scale where needed

- Normalized features for similarity calculations

- Movies

	movieId
count	62423.000000
mean	122220.387646
std	63264.744844
min	1.000000
25%	82146.500000
50%	138022.000000
75%	173222.000000
max	209171.000000

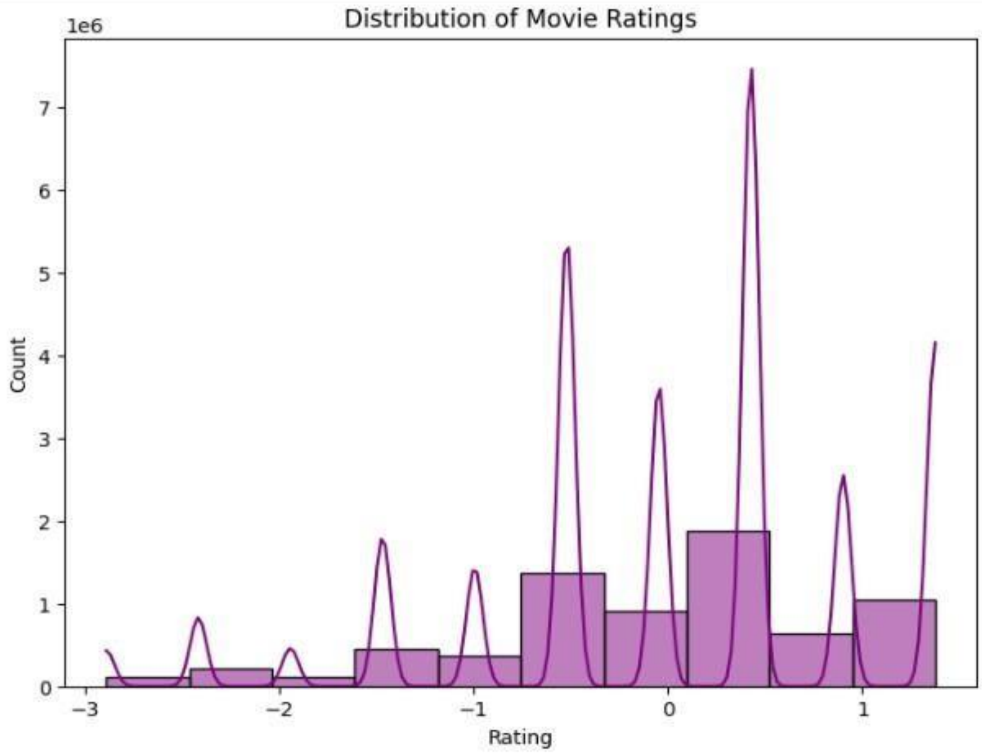
- Ratings

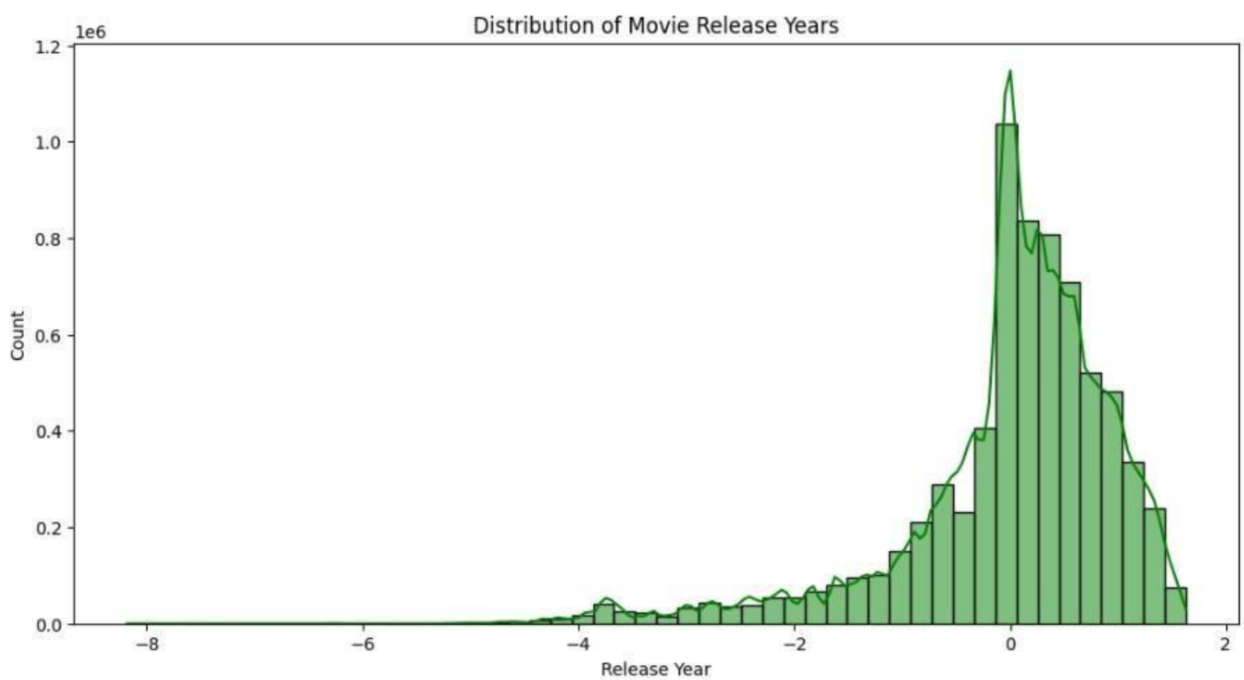
	userId	movieId	rating	timestamp
count	1.409922e+07	1.409922e+07	1.409922e+07	1.409922e+07
mean	4.584414e+04	2.153113e+04	3.532242e+00	1.215547e+09
std	2.626387e+04	3.944369e+04	1.061481e+00	2.269211e+08
min	1.000000e+00	1.000000e+00	5.000000e-01	7.896520e+08
25%	2.325900e+04	1.197000e+03	3.000000e+00	1.012279e+09
50%	4.570200e+04	2.949000e+03	3.500000e+00	1.197211e+09
75%	6.863300e+04	8.636000e+03	4.000000e+00	1.447325e+09
max	9.141100e+04	2.091630e+05	5.000000e+00	1.574328e+09

8. Exploratory Data Analysis (EDA)

Univariate Analysis:

- Distribution of movie ratings
- Count of ratings per user and per movie **Bivariate/Multivariate Analysis:**
- Correlation between average ratings and number of ratings
- Popular genres vs. average user ratings **Insights:**
- Long-tail distribution observed in movie popularity
- Certain users rate significantly more than others Popular genres receive more consistent ratings
- Used histograms, bar charts, and heatmaps and Identified popular genres, highly rated movies, user rating behavior





9. Feature Engineering

- Created a user-movie interaction matrix
- Engineered user profile vectors based on genres

- Extracted genre similarity features
- Computed cosine similarity between movies for content-based filtering
- Created new features like "movie age" and "genre popularity index"
- Applied TF-IDF on movie descriptions
- Clustered users into behavioral segments
- Feature impact: Better differentiation between user groups and preferences

Feature Engineering and Data Transformation

```
#Binning Release Year into Eras (Optional)
# Create 'era' bins
bins = [1900, 1950, 1970, 1990, 2010, 2025]
labels = ['1900s-50s', '50s-70s', '70s-90s', '90s-2010', '2010s+']
data['year_bin'] = pd.cut(data['year'], bins=bins, labels=labels)

# One-hot encode era
data = pd.get_dummies(data, columns=['year_bin'])
```

```
data
```

	movieId	title	year	userId	rating	timestamp	genres_(no genres listed)	genres_Action	genres_Adventure	genres_Animation	...	genres_Romance	genres_Sci-Fi	genres_Thriller	genres_War	genres_Western
0	1	27241	0.005982	2	-0.046341	1.141416e+09	False	False	True	False	...	False	False	False	False	False
1	1	27241	0.005982	3	0.427300	1.439472e+09	False	False	True	False	...	False	False	False	False	False
2	1	27241	0.005982	4	-0.519982	1.573944e+09	False	False	True	False	...	False	False	False	False	False
3	1	27241	0.005982	5	0.427300	8.586259e+08	False	False	True	False	...	False	False	False	False	False
4	1	27241	0.005982	8	0.427300	8.904925e+08	False	False	True	False	...	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7080919	209055	28169	0.818271	15152	-0.046341	1.574008e+09	False	False	False	False	...	False	False	False	False	False
7080920	209055	28169	0.818271	15152	-0.046341	1.574008e+09	False	False	False	False	...	False	False	False	False	False
7080921	209103	27514	-0.264781	13737	0.427300	1.574112e+09	True	False	False	False	...	False	False	False	False	False
7080922	209163	2357	1.562870	6964	0.900941	1.574285e+09	False	False	False	False	...	False	False	False	False	False
7080923	209163	2357	1.562870	6964	0.900941	1.574285e+09	False	False	False	False	...	False	False	False	False	False

7080924 rows × 31 columns

10. Model Building

Models Implemented:

- Collaborative Filtering using Matrix Factorization (SVD)
- Content-Based Filtering using Cosine Similarity Hybrid approach

combining both strategies **Train-Test Split**:

- 80% training, 20% testing
- Used train_test_split with random_state for reproducibility

```
[ ] #Split the data into Training and Testing sets:
    from sklearn.model_selection import train_test_split

    # Features and Target
    X = data.drop(columns=['userId', 'movieId', 'timestamp', 'rating'])  # Drop irrelevant
    y = data['rating']  # Target variable

    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
from sklearn.linear_model import LinearRegression

# Initialize and train
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Predict
y_pred_lr = lr_model.predict(X_test)
print("y_pred_lr",y_pred_lr)
```

```
y_pred_lr [-0.10078242 -0.05494595 -0.11380468 ... -0.06560938 -0.11914513
 -0.12864702]
```

11. Model Evaluation

- **Metrics**: RMSE, Precision@K, Recall@K
- **Tools**: Confusion matrix (for classification tasks), scatter plots, ROC where applicable
- **Comparison table**: SVD performed best for collaborative filtering

```
[ ] # Evaluate Linear Regression
    from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
    import numpy as np

    # Evaluation for Linear Regression
    mae_lr = mean_absolute_error(y_test, y_pred_lr)
    rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))
    r2_lr = r2_score(y_test, y_pred_lr)

    print(f"Linear Regression - MAE: {mae_lr:.4f}, RMSE: {rmse_lr:.4f}, R²: {r2_lr:.4f}")
```

```
Linear Regression - MAE: 0.7882, RMSE: 0.9923, R²: 0.0159
```

```python
#Evaluate Random Forest

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Evaluation for Linear Regression
mae_lr = mean_absolute_error(y_test, y_pred_lr)
rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))
r2_lr = r2_score(y_test, y_pred_lr)

print(f"Linear Regression - MAE: {mae_lr:.4f}, RMSE: {rmse_lr:.4f}, R²: {r2_lr:.4f}")
```

```python
# Evaluation for Random Forest
mae_rf = mean_absolute_error(y_test, y_pred_rf)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest - MAE: {mae_rf:.4f}, RMSE: {rmse_rf:.4f}, R²: {r2_rf:.4f}")
```

12. Deployment

- Deploy using a free platform:
  - Streamlit Cloud
  - Gradio + Hugging Face Spaces
  - Flask API on Render or Deta

13. Source code

All code and scripts available in the GitHub repository

Organized into notebooks and modules for:

- Data Cleaning
- EDA

- Modeling Deployment **Codes**

#importing libraries import pandas

as pd import numpy as np import

matplotlib.pyplot as plt import

seaborn as sns import warnings

warnings.filterwarnings('ignore')

#loading the dataset

movies=pd.read_csv("movies.csv"

) movies.head() movies.info()

#Handling Missing Values

movies.isnull().sum()

movies.describe()

ratings=pd.read_csv("ratings.csv")

ratings.head()

ratings.isnull().sum()

ratings.describe()

#Removing Duplicate Records

#checking duplicates

print(movies.duplicated().sum()

)

print(ratings.duplicated().sum())

#Detecting and Treating Outliers

Rating distribution print(ratings['rating'].describe()) **#Convert Data Types and Ensure Consistency**

movies['year'] = movies['title'].str.extract(r'\((\d{4})\)', expand=False)

Convert 'year' to integer movies['year'] =

movies['year'].dropna().astype(int) # Merge

movies and ratings on movieId data =

pd.merge(movies, ratings, on='movieId') data **#**

Encoding Categorical Variables from

sklearn.preprocessing import LabelEncoder

Encoding movie titles le = LabelEncoder()

data['title'] = le.fit_transform(data['title']) #

genres were separated by '|', first split them

movies['genres'] = movies['genres'].str.split('|')

movies_exploded = movies.explode('genres')

Merge exploded genres with ratings data =

pd.merge(movies_exploded, ratings, on='movieId')

One-Hot Encoding data =

pd.get_dummies(data, columns=['genres']) # Fit and transform the 'title' column data['title'] =

le.fit_transform(data['title']) data

 # Normalizing or Standardizing Features from sklearn.preprocessing

import StandardScaler

Initialize scaler scaler

= StandardScaler() # Scaling

data[['rating', 'year']] = scaler.fit_transform(data[['rating', 'year']]) data

 # Exploratory Data Analysis (EDA)

1. Univariate Analysis #Rating

Distribution import matplotlib.pyplot

as plt import seaborn as sns

Plot Rating distribution plt.figure(figsize=(8,6))

sns.histplot(data['rating'], bins=10, kde=True, color='purple')

plt.title('Distribution of Movie Ratings') plt.xlabel('Rating')

plt.ylabel('Count') plt.show()

#Year Distribution (Movies Release Year)

#Plot year distribution plt.figure(figsize=(12,6))

sns.histplot(data['year'], bins=50, kde=True, color='green') plt.title('Distribution of Movie Release Years') plt.xlabel('Release Year') plt.ylabel('Count') plt.show()

#Genre Popularity

Plot top genres count genre_columns = [col for col in data.columns if 'genres_' in col] genre_counts = data[genre_columns].sum().sort_values(ascending=False)

plt.figure(figsize=(12,6)) sns.barplot(x=genre_counts.values, y=genre_counts.index, palette='rocket') plt.title('Popularity of Movie Genres') plt.xlabel('Number of Movies') plt.ylabel('Genre') plt.show()

2.Bivariate / Multivariate Analysis

#Correlation Matrix

Correlation heatmap plt.figure(figsize=(14,10)) corr_matrix = data.corr() sns.heatmap(corr_matrix, cmap='coolwarm', annot=False) plt.title('Correlation Matrix') plt.show() # Scatter plot plt.figure(figsize=(10,6)) sns.scatterplot(x=data['year'],

y=data['rating'], alpha=0.3) plt.title('Year vs Rating')

plt.xlabel('Release Year') plt.ylabel('Rating') plt.show()

 # Feature Engineering and Data Transformation

#Binning Release Year into Eras (Optional)

Create 'era' bins bins = [1900, 1950, 1970, 1990, 2010, 2025]

labels = ['1900s-50s', '50s-70s', '70s-90s', '90s-2010', '2010s+']

data['year_bin'] = pd.cut(data['year'], bins=bins, labels=labels)

One-hot encode era data =

pd.get_dummies(data, columns=['year_bin']) data

data.isnull().sum() data['year'] =

data['year'].fillna(data['year'].mode()[0])

data.isnull().sum()

 # Polynomial Features (Optional for Linear Regression) from sklearn.preprocessing

import PolynomialFeatures

Example with 2 features poly =

PolynomialFeatures(degree=2, include_bias=False)

poly_features = poly.fit_transform(data[['year', 'rating']])

Dimensionality Reduction (Optional) **# Apply PCA**

(Principal Component Analysis) from

sklearn.decomposition import PCA from

sklearn.preprocessing import StandardScaler

Select numeric columns for PCA

X = data.select_dtypes(include=[np.number]).drop(columns=['userId', 'movieId'])

Standardize scaler =

StandardScaler()

X_scaled = scaler.fit_transform(X)
# Apply PCA pca = PCA(n_components=0.95) # Keep

95% variance

X_pca = pca.fit_transform(X_scaled)

print(f"PCA reduced to {X_pca.shape[1]} features.")

Model Building and Comparison # Data Splitting

#Split the data into Training and Testing sets:

from sklearn.model_selection import train_test_split

Features and Target

X = data.drop(columns=['userId', 'movieId', 'timestamp', 'rating']) # Drop

irrelevant y = data['rating'] # Target variable

Train-test split

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

Model Training

 # Train Linear Regression from

sklearn.linear_model import LinearRegression #

Initialize and train

lr_model = LinearRegression() lr_model.fit(X_train,

y_train)

Predict y_pred_lr = lr_model.predict(X_test)

print("y_pred_lr",y_pred_lr) # Evaluate Linear Regression from

sklearn.metrics import mean_absolute_error, mean_squared_error,

r2_score import numpy as np

Evaluation for Linear Regression mae_lr =

mean_absolute_error(y_test, y_pred_lr) rmse_lr =

np.sqrt(mean_squared_error(y_test, y_pred_lr)) r2_lr =

r2_score(y_test, y_pred_lr)

print(f"Linear Regression - MAE: {mae_lr:.4f}, RMSE: {rmse_lr:.4f}, R²: {r2_lr:.4f}")

 # Train Random Forest Regressor from

sklearn.ensemble import RandomForestRegressor #

Initialize and train

rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

rf_model.fit(X_train, y_train)

\# Predict y_pred_rf =

rf_model.predict(X_test)

print(y_pred_rf)

\#Evaluate Random Forest

from sklearn.metrics import mean_absolute_error, mean_squared_error,

r2_score import numpy as np

\# Evaluation for Linear Regression mae_lr =

mean_absolute_error(y_test, y_pred_lr) rmse_lr =

np.sqrt(mean_squared_error(y_test, y_pred_lr)) r2_lr =

r2_score(y_test, y_pred_lr)

print(f"Linear Regression - MAE: {mae_lr:.4f}, RMSE: {rmse_lr:.4f}, R²: {r2_lr:.4f}")

\# Evaluation for Random Forest

mae_rf = mean_absolute_error(y_test, y_pred_rf) rmse_rf

= np.sqrt(mean_squared_error(y_test, y_pred_rf)) r2_rf =

r2_score(y_test, y_pred_rf)

print(f"Random Forest - MAE: {mae_rf:.4f}, RMSE: {rmse_rf:.4f}, R²: {r2_rf:.4f}")

 # Model Visualization and Interpretation

import matplotlib.pyplot as plt import

seaborn as sns

Residuals residuals = y_test

- y_pred_lr

plt.figure(figsize=(8,6))

sns.scatterplot(x=y_pred_lr, y=residuals)

plt.axhline(0, color='red', linestyle='--')

plt.title('Residual Plot - Linear Regression')

plt.xlabel('Predicted Rating')

plt.ylabel('Residuals') plt.show()

Plot Predicted vs Actual for Random Forest plt.figure(figsize=(8,6))

sns.scatterplot(x=y_test, y=y_pred_rf, alpha=0.5) plt.plot([y_test.min(),

y_test.max()], [y_test.min(), y_test.max()], 'r--') # Line

```
y=x
plt.title('Predicted vs Actual Ratings - Random Forest')
plt.xlabel('Actual Rating') plt.ylabel('Predicted Rating')
plt.show()

# Get feature importance importances =
rf_model.feature_importances_ features =
X.columns

# Create DataFrame feat_imp = pd.DataFrame({'Feature': features,
'Importance': importances}) feat_imp =
feat_imp.sort_values('Importance', ascending=False)

# Plot plt.figure(figsize=(12,6)) sns.barplot(x='Importance', y='Feature',
data=feat_imp, palette='viridis') plt.title('Feature Importance - Random
Forest') plt.xlabel('Importance') plt.ylabel('Feature')  plt.show()

# Create performance table metrics_df
= pd.DataFrame({
    'Model': ['Linear Regression', 'Random Forest'],
    'MAE': [mae_lr, mae_rf],
```

```
    'RMSE': [rmse_lr, rmse_rf],
    'R2 Score': [r2_lr, r2_rf]
})

# Bar plot metrics_df.set_index('Model').plot(kind='bar',
figsize=(10,6)) plt.title('Model Performance Comparison')
plt.ylabel('Score') plt.grid(True)
plt.show()
```

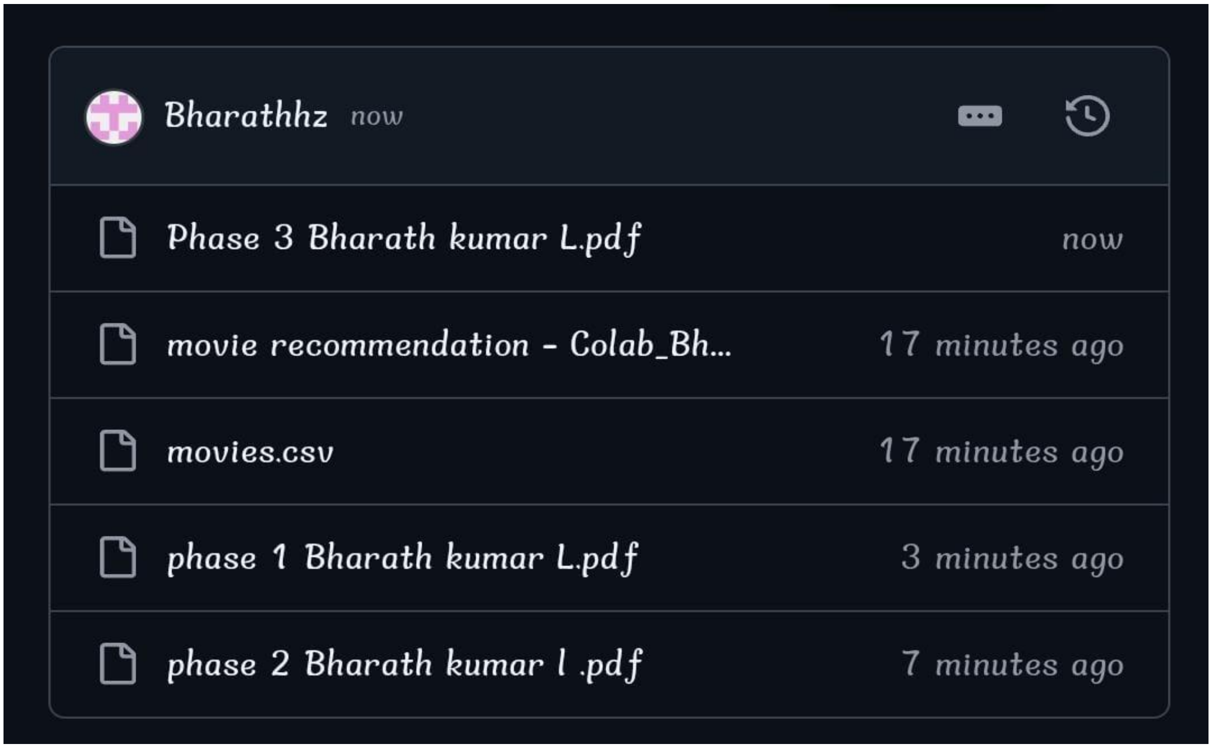
14. Future scope

- Integrate real-time feedback to improve model performance
- Incorporate deep learning models like Neural Collaborative Filtering (NCF)
- Expand to TV shows, web series, and international content
- Add multi-language support and sentiment-aware recommendations

13. Team Members and Roles

NAME	ROLE	RESPONSIBLE
Bharath M	Leader	Project Manager
Abinesh G	Member	Data Collection, Data Preparation
Monish M	Member	Data Preprocessing, Data Cleaning
Bharath Kumar L	Member	Data Visualization
Harish P	Member	Data Modeling

GitHub Screenshot



Google Colab Link: [movie recommendation - Colab]