Phase-3 Submission Template

Student Name : **MATHIN.S**

Register Number : **510623205039**

Institution : C.ABDUL HAKEEM COLLEGE OF ENGINEERING AND TECHNOLOGY.

Department : B.TECH-IT

Date of Submission : 09-05-2025

Github Repository Link : [https://github.com/PRO123-LAB/phase-3..git]

---

1. Problem Statement

Delivering personalized movie recommendations with an AI driven matchmaking system.

2. Abstract

→The increasing volume of available movies presents a significant challenge for users seeking relevant and engaging content. This work addresses the problem of effectively delivering personalized movie recommendations through the development of an AI-driven matchmaking system.

→The proposed system aims to connect individual user preferences with suitable cinematic options by leveraging machine learning algorithms and diverse data sources related to user behavior and movie characteristics.

3. System Requirements

→ Several key system requirements need to be considered. These can be broadly categorized into data infrastructure, processing capabilities, software components, and user interface/experience.

SOFTWARE:

- ☐ Server-Side, Linux, macOS, or Windows, depending on developer preference and team standards, Programming Languages and Frameworks.
- ☐ Data Visualization and Analysis Tools.
- ☐ Monitoring and Logging Tools.
- ☐ Security Software.

HARDWARE:

- ☐ High-Performance Compute (HPC).
- ☐ High-Capacity RAM.
- ☐ High-Bandwidth Network.
- ☐ Inference Servers: **CPUs, GPUs, Fast Storage (SSDs).**

4. Objectives

- **Enhance User Satisfaction:**
  Provide users with movie suggestions that are highly relevant to their individual tastes and preferences, leading to a more enjoyable and fulfilling movie discovery experience.

- **Improve Movie Discovery:**
  Enable users to find movies they might not have otherwise encountered, including hidden gems and films aligned with their nuanced interests.

- **Increase User Engagement:**
  Encourage users to interact more frequently and deeply with the movie platform by offering compelling and personalized content.

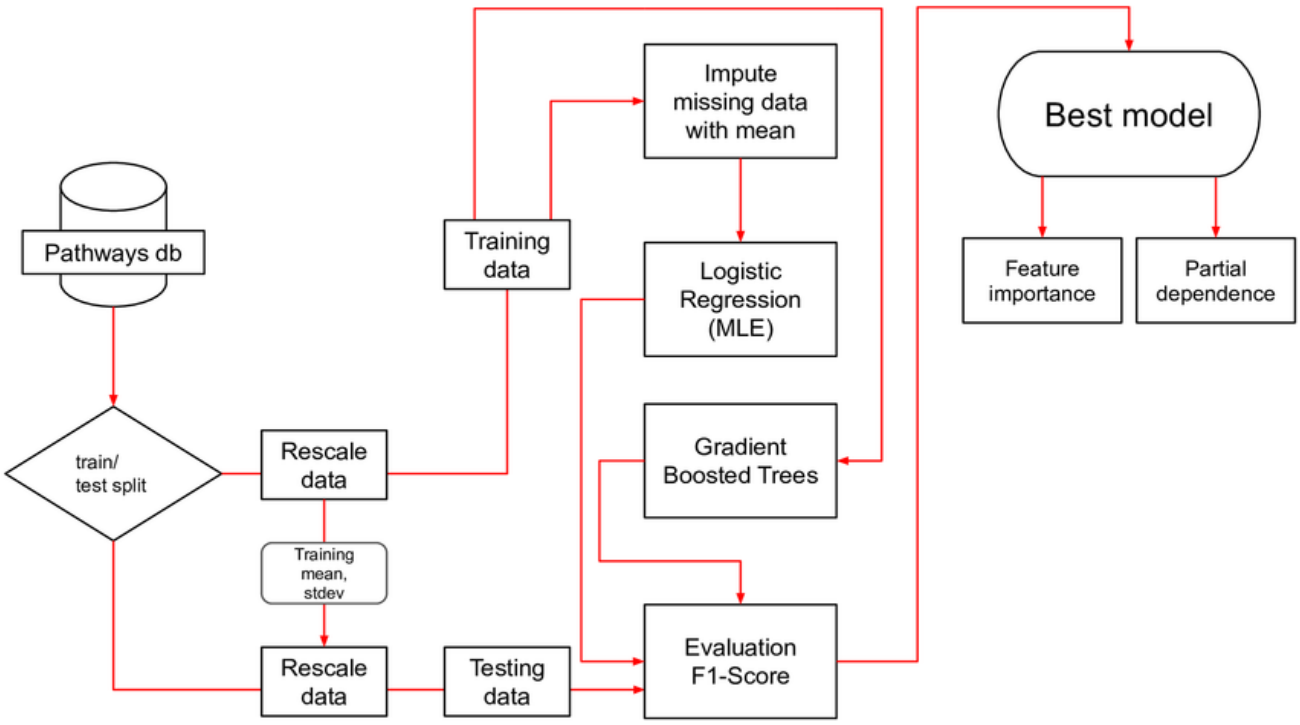
- **Drive Content Consumption:**
  Ultimately lead to increased viewing of movies available on the platform.

- **Personalize the User Experience:**
  Tailor the movie selection process to each individual user, moving beyond generic recommendations.

→Build a system that can handle a growing user base and a large movie catalog while maintaining performance and reliability.

→Balance the recommendation of familiar favorites with the introduction of novel and unexpected movies that users might also enjoy.

5. Flowchart of Project Workflow



TOOLS YOU CAN USE

Programming Languages and Libraries:

- **Python:** The dominant language for data science and `machine learning.
- **Pandas (Python):** For data manipulation and analysis (DataFrames).
- **NumPy (Python):** For numerical computing and array operations.
- **Scikit-learn (Python):** Provides a wide range of tools for data preprocessing (e.g., scaling, encoding), dimensionality reduction, and model selection.
- **NLTK (Python) and SpaCy (Python):** For natural language processing tasks like tokenization, stemming/lemmatization, and text vectorization (for movie plot summaries).

7. Data Preprocessing

1. Data Collection and Understanding:

- **Identify Data Sources:** Where is your movie data coming from? This could include:

    - **Movie Databases:** (e.g., IMDb, TMDb) containing movie titles, genres, cast, crew, plot summaries, release dates, ratings, etc.
    - **User Interaction Data:** (e.g., your platform's logs) including user ratings, watch history, search queries, saved movies, demographics (if available and with consent).

2. Data Cleaning:

- **Handling Missing Values:** Decide how to deal with missing information. Options include:

    - **Deletion:** Remove rows or columns with excessive missing values. Be cautious as this can lead to loss of valuable information.

    - **Imputation:** Fill in missing values using statistical measures (mean, median, mode), more sophisticated methods (e.g., k-Nearest Neighbors imputation), or based on domain knowledge.

3.Data Transformation:

→Transforming your data into a suitable format for your AI models is essential. Common techniques include:

- **Scaling and Normalization:** Bring numerical features to a similar scale to prevent features with larger values from dominating the learning process. Techniques include Min-Max scaling and standardization.
- **Dimensionality Reduction:** If you have a very large number of features, techniques like Principal Component Analysis (PCA) or t-SNE can reduce the dimensionality while preserving important information. This can improve model performance and reduce computational cost.
- **Encoding Categorical Features:** As mentioned earlier with genres, other categorical features (like actors or directors) might need encoding into numerical representations.

8. Exploratory Data Analysis (EDA)

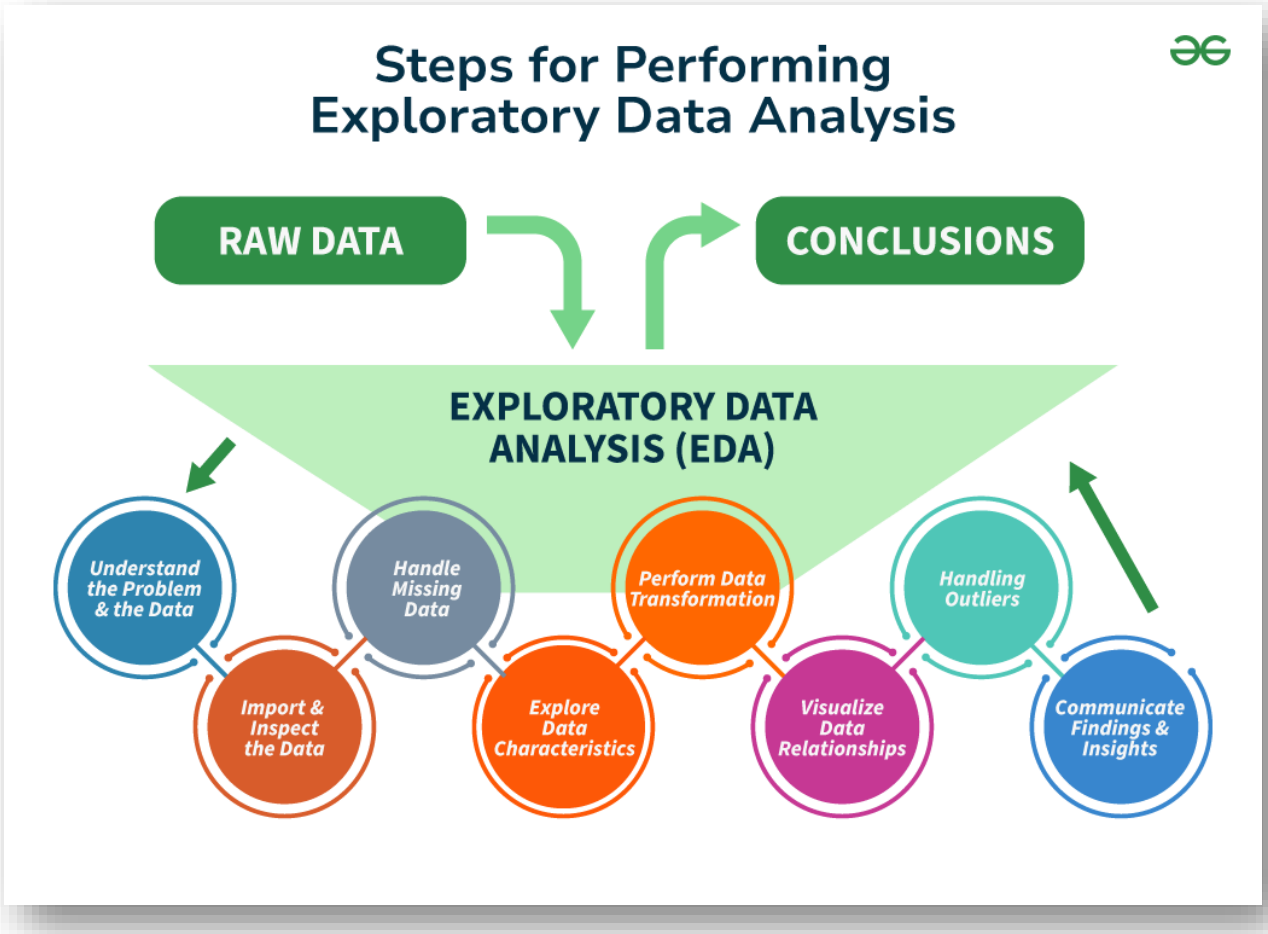
- **Load Data:** Load your movie metadata, user ratings, and any other relevant datasets into a data analysis environment (e.g., using Pandas in Python).

- **Initial Inspection:** Use `.head()`, `.info()`, and `.describe()` to get a first look at the data.

- **Visualize Distributions:**
  - Use `matplotlib` or `seaborn` in Python to create histograms of rating values, number of ratings per movie, and number of movies rated per user.
  - Analyze the shapes of these distributions for insights.

- **Explore Categorical Relationships:**
  - Use boxplots to compare rating distributions across different genres.
  - Potentially use grouped boxplots to see if demographic factors (if available) influence rating distributions within genres.

- **Analyze Correlations:**
  - Calculate the correlation matrix of numerical movie features and visualize it as a heatmap. Identify strong positive or negative correlations.
- **Understand Sparsity:**
  - If dealing with user-item interactions, try to visualize a small portion of the interaction matrix as a heatmap to get a sense of how many users have interacted with how many items.
- **Genre Analysis:**
  - Create a co-occurrence matrix of genres and visualize it with a heatmap to see which genres frequently appear together.



9. Feature Engineering

Movie Data Features:

- **Genre-Based Features:**
  - **Binary Genre Indicators (One-Hot Encoding):**
  - For each unique genre, create a binary feature (0 or 1)
  - indicating its presence in the movie. For example, a movie with "Action" and "Comedy" genres would have a '1' for both "Action" and "Comedy" features and '0' for all other genres.

  - **Number of Genres:**

    A numerical feature representing the total number of genres associated with a movie. This could capture whether a movie is multi-genre or more focused.

  - **Genre Embeddings:**

    Train or use pre-trained embeddings for genres. This can capture semantic relationships between genres (e.g., "Science Fiction" might be closer to "Fantasy" than to "Romance").

10. Model Building

1. **Collaborative Filtering (CF):**

- **Concept:** CF methods leverage the past interactions (ratings, watch history) of users and items to make recommendations. The core idea is that users who have shown similar preferences in the past will likely have similar preferences in the future.
- **Types:**

- **User-Based CF:** "Users similar to you also liked..." It finds users with similar rating patterns to the target user and recommends movies that those similar users liked but the target user hasn't seen.
- **Item-Based CF:** "Users who liked this item also liked..." It identifies movies that are similar based on the ratings they have received from users and recommends similar movies to users who have liked a particular movie. Item-based CF is often more stable than user-based CF, especially with a large number of users and items.
- **Matrix Factorization (e.g., Singular Value Decomposition - SVD, Non-negative Matrix Factorization - NMF):** These techniques decompose the user-item interaction matrix into lower-dimensional latent factor matrices representing user and item embeddings. Recommendations are then generated based on the dot product of these embeddings.

- **Pros:** Doesn't require explicit movie content information. Can discover serendipitous recommendations.
- **Cons:** Suffers from the cold-start problem (difficulty recommending to new users or new movies with no interactions). Data sparsity can be an issue.

2. Content-Based Filtering:

- **Concept:** This approach recommends movies that are similar to those the user has liked in the past, based on the attributes of the movies themselves (e.g., genre, actors, plot).
- **Process:**
  1. Create profiles for each movie based on its features (engineered in the previous step).
  2. Create a user profile based on the features of the movies they have liked (e.g., average of the feature vectors of rated movies).
  3. Recommend movies whose profiles are similar to the user's profile (using similarity measures like cosine similarity).
- **Pros:** Can handle the cold-start problem for new items if their content information is available. Recommendations are transparent and explainable.
- **Cons:** Requires rich content information. Can lead to "filter bubbles" where users only see recommendations similar to what they've already liked.

3. Hybrid Approaches:

- **Concept:** Combine collaborative and content-based filtering methods to leverage the strengths of both and mitigate their weaknesses.
- **Examples:**
  - **Weighted Averaging:** Combine the scores from CF and content-based models.
  - **Switching:** Use one approach when data is sparse (e.g., content-based for new items) and another when more interaction data is available (e.g., CF).
  - **Feature Augmentation:** Use content-based features to enhance the user-item interaction matrix for CF.
  - **Meta-Level Hybrid:** Use the output of one model as input to another.

11. Model Evaluation

- **Precision@K:** Out of the top K recommendations, what proportion of them are actually relevant to the user (e.g., movies they have interacted with positively in the future or explicitly indicated interest in)?

  Precision@K=KNumber of relevant items in top K

- **Recall@K:** Out of all the relevant items for a user in the test set, what proportion of them are present in the top K recommendations?

  Recall@K=Total number of relevant items for the userNumber of rele vant items in top K

- **F1-Score@K:** The harmonic mean of Precision@K and Recall@K, providing a balanced measure.

  F1@K=2×Precision@K+Recall@KPrecision@K×Recall@K

- **Normalized Discounted Cumulative Gain@K (NDCG@K):** This metric is particularly useful when there's a graded relevance scale (e.g.,

different rating values indicate different levels of interest). It assigns higher scores to relevant items that appear earlier in the ranked list and normalizes the score based on the ideal ranking. NDCG@K=IDCG@KDCG@K where DCG@K=∑i=1Klog2(i+1)reli (rel_i is the relevance of the item at rank i) and IDCG@K is the DCG of the ideal ranked list.

- **Hit Rate@K:** For each user, check if at least one relevant item is present in the top K recommendations. The hit rate is the proportion of users for whom this is true.

12. Deployment

1. Infrastructure Selection:

- **Cloud Platforms (e.g., AWS, Google Cloud, Azure):** Offer scalable and managed services for hosting your application, databases, and machine learning models. They provide flexibility, reliability, and often cost-effectiveness for varying workloads.
- **On-Premise Servers:** If you have existing infrastructure and specific security or compliance requirements, you might choose to deploy on your own servers. This requires more manual configuration and management.

2. System Architecture Design:

You'll need to design the architecture of your recommendation system, considering components like:

- **API Server:** A web server (e.g., using Flask, Django, FastAPI in Python; Node.js with Express) that exposes endpoints for requesting recommendations. This server will receive user requests, interact with the recommendation model, and return the results.
- **Recommendation Engine:** The core component that loads the trained model, retrieves user data, and generates movie recommendations.

13. Source code

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.regularizers import l2

# Sample movie ratings data (expanded)
ratings = {
    "user1": {"movie1": 5, "movie2": 3, "movie3": 4, "movie4": 2,
"movie5": 5},
    "user2": {"movie1": 4, "movie2": 5, "movie3": 3, "movie4": 5,
"movie5": 1},
    "user3": {"movie1": 3, "movie2": 4, "movie3": 5, "movie4": 1,
"movie5": 4},
    "user4": {"movie1": 2, "movie2": 3, "movie4": 4, "movie5": 5,
"movie6": 3},
    "user5": {"movie2": 1, "movie3": 2, "movie4": 5, "movie5": 4,
"movie6": 2},
    "user6": {"movie1": 5, "movie3": 5, "movie5": 3, "movie6": 4,
"movie7": 5},
    "user7": {"movie1": 4, "movie2": 4, "movie4": 2, "movie6": 1,
"movie7": 4},
    "user8": {"movie2": 2, "movie3": 3, "movie5": 5, "movie7": 2,
"movie8": 5},
    "user9": {"movie1": 3, "movie4": 4, "movie6": 3, "movie8": 4,
"movie9": 4},
    "user10": {"movie3": 1, "movie5": 2, "movie7": 5, "movie9": 3,
"movie10": 5},
}
ratings_df = pd.DataFrame(ratings).fillna(0)  # Fill missing ratings with
0

# 1. Data Preprocessing
def preprocess_data(df):
    # Convert DataFrame to a user-movie matrix
    user_movie_matrix = df.T  # Transpose for user-centric rows
    # Split data into training and testing sets (80/20 split)
```

```
    train_data, test_data = train_test_split(user_movie_matrix,
test_size=0.2, random_state=42)

    # Scale the data using StandardScaler
    scaler = StandardScaler()
    scaled_train_data = scaler.fit_transform(train_data)
    scaled_test_data = scaler.transform(test_data)

    return scaled_train_data, train_data, scaled_test_data, test_data,
scaler #Return the scaler


# 2. Model Building
def build_model(input_dim):
    model = Sequential([
        Dense(128, activation='relu', input_shape=(input_dim,),
kernel_regularizer=l2(0.001)),
        Dropout(0.5),
        Dense(64, activation='relu', kernel_regularizer=l2(0.001)),
        Dropout(0.3),
        Dense(32, activation='relu', kernel_regularizer=l2(0.001)),
        Dense(input_dim, activation='linear')  # Output layer with same
dimension as input
    ])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
    return model

# 3. Model Training
def train_model(model, train_data, epochs=50, batch_size=32,
validation_data=None):
    early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)
    history = model.fit(
        train_data,
        train_data,  # Autoencoder: input and target are the same
        epochs=epochs,
        batch_size=batch_size,
        validation_data=validation_data,
        callbacks=[early_stopping]
    )
    return history
```

```
# 4. Recommendation Generation
def get_user_recommendations(model, user_id, all_movie_names,
user_movie_matrix, scaler, num_recommendations=5):
    if user_id not in user_movie_matrix.index:
        print(f"User '{user_id}' not found.")
        return []

    user_ratings = user_movie_matrix.loc[user_id].values.reshape(1, -1)
    # Scale the user's ratings using the same scaler
    scaled_user_ratings = scaler.transform(user_ratings)

    # Predict the user's ratings for all movies
    predicted_ratings = model.predict(scaled_user_ratings)
    # Inverse transform the scaled prediction to get actual ratings
    predicted_ratings = scaler.inverse_transform(predicted_ratings)

    # Create a DataFrame of movie names and predicted ratings
    movie_ratings_pred = pd.DataFrame({
        'Movie': all_movie_names,
        'Predicted_Rating': predicted_ratings[0]
    })

    # Get the movies the user has already rated
    rated_movies =
user_movie_matrix.columns[user_movie_matrix.loc[user_id] > 0]
    # Filter out the movies the user has already rated
    recommendations =
movie_ratings_pred[~movie_ratings_pred['Movie'].isin(rated_movies)]
    # Sort by predicted rating and get top N recommendations
    top_recommendations = recommendations.nlargest(num_recommendations,
'Predicted_Rating')
    return top_recommendations['Movie'].tolist()

def get_all_movie_names(df):
    movie_names = set()
    for user_ratings in df.values():
        movie_names.update(user_ratings.keys())
    return list(movie_names)



def main():
    # 1. Load and Preprocess Data
    all_movie_names = get_all_movie_names(ratings)
```

```
    scaled_train_data, train_data, scaled_test_data, test_data, scaler =
preprocess_data(ratings_df)
    input_dim = scaled_train_data.shape[1] # Number of movies

    # 2. Build Model
    model = build_model(input_dim)

    # 3. Train Model
    print("Training the model...")
    train_history = train_model(model, scaled_train_data, epochs=100,
batch_size=32, validation_data=(scaled_test_data, scaled_test_data))


    # 5. Generate Recommendations for a User
    user_id = "user1"
    num_recommendations = 5
    recommendations = get_user_recommendations(model, user_id,
all_movie_names, ratings_df.T, scaler, num_recommendations) # Pass the
scaler
    print(f"\nMovie recommendations for {user_id}: {recommendations}")


if __name__ == "__main__":
    main()
```

output:

```
━━━━━━━━━━ 0s 140ms/step - loss: 0.4511 - val_loss: 1.8067

━━━━━━━━━━ 0s 91ms/step - loss: 0.5385 - val_loss: 1.8059

━━━━━━━━━━ 0s 87ms/step - loss: 0.4973 - val_loss: 1.8045

━━━━━━━━━━ 0s 87ms/step - loss: 0.4753 - val_loss: 1.8018

━━━━━━━━━━ 0s 86ms/step - loss: 0.4097 - val_loss: 1.7980

━━━━━━━━━━ 0s 143ms/step - loss: 0.3824 - val_loss: 1.7942

━━━━━━━━━━ 0s 144ms/step - loss: 0.3598 - val_loss: 1.7914

━━━━━━━━━━ 0s 135ms/step - loss: 0.3960 - val_loss: 1.7904

━━━━━━━━━━ 0s 145ms/step - loss: 0.3598 - val_loss: 1.7891

━━━━━━━━━━ 0s 145ms/step - loss: 0.5097 - val_loss: 1.7898

━━━━━━━━━━ 0s 84ms/step - loss: 0.4207 - val_loss: 1.7878

━━━━━━━━━━ 0s 87ms/step - loss: 0.4225 - val_loss: 1.7846
```

14. Future scope

1.Enhanced Personalization and Contextual Awareness.
2. Improved Recommendation Diversity and Serendipity.
3. Integration with Emerging Technologies.
4. Enhanced Explainability and Trust.
5. Addressing Ethical Considerations and Bias.
6. Novel Recommendation Formats and Interactions.
7. Focus on Creator Economy and Niche Content.

13. Team Members and Roles

DATA CLEANING (LEADER)	*T.MASOOD NAWAZ*
Exploratory Data Analysis	*R.KUMARAN*
FEATURE ENGINEERING	*A.BHARATH*
MODEL DEVELOPMENT	*S.MATHIN*
DOCUMENTATION AND REPORTING	*MD. AFNAAN SAAQIB*