Delivering personalized movie recommendations with an AI driven matchmaking system

```
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
```