Predictive Analytics DIGITAL ASSESSMENT II

NAME :- HARSH AWASTHI

REG NO :- 21BDS0196

[LINK TO KAGGLE]

🏋️ Privacy-Preserving Personalized Fitness Recommender System

SAMPLE FRAMEWORK PROVIDED IN REFRENCE RESEARCH PAPAER

76:8 X. Liu et al.

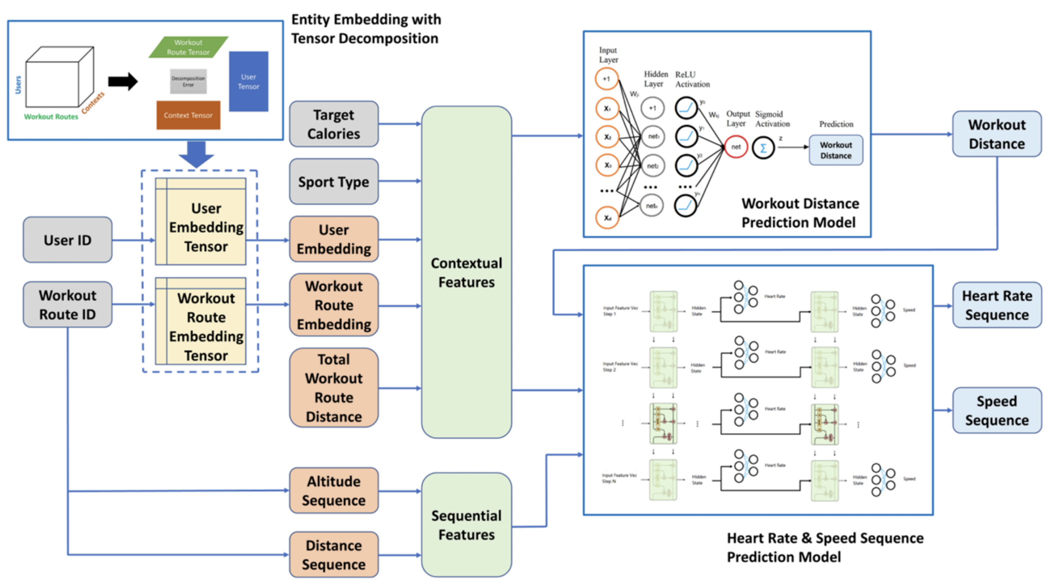
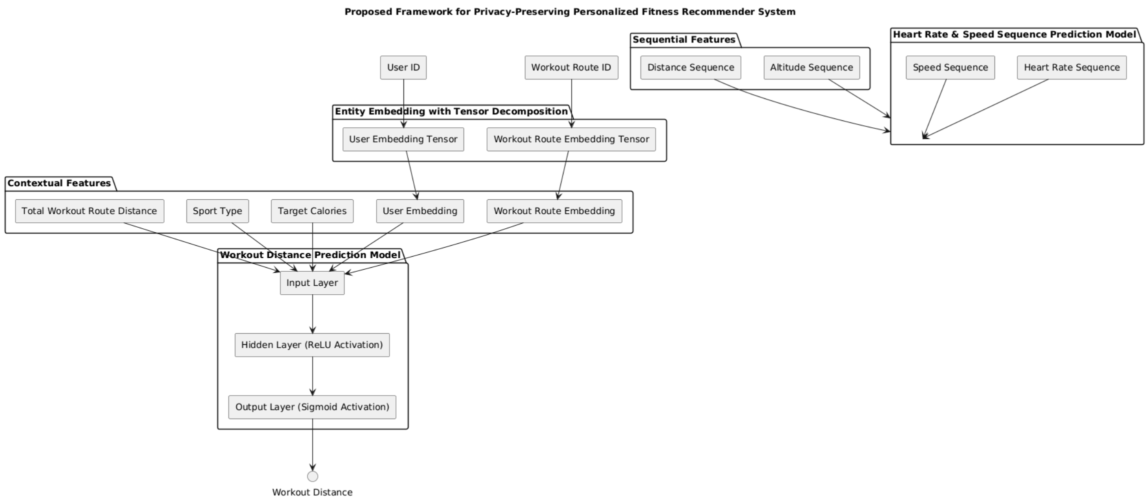
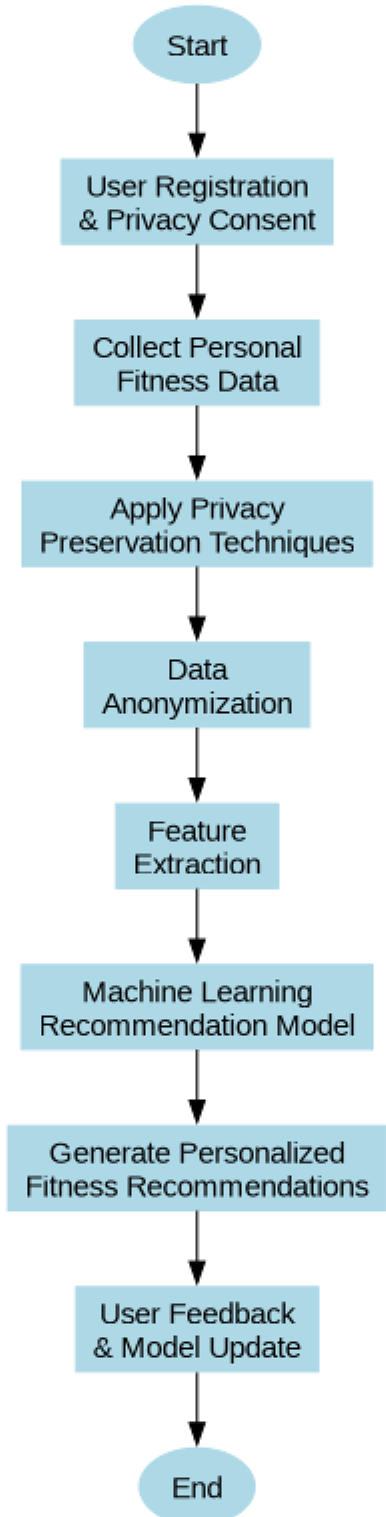


Fig. 1. Overview of our proposed framework $P^3FitRec$.

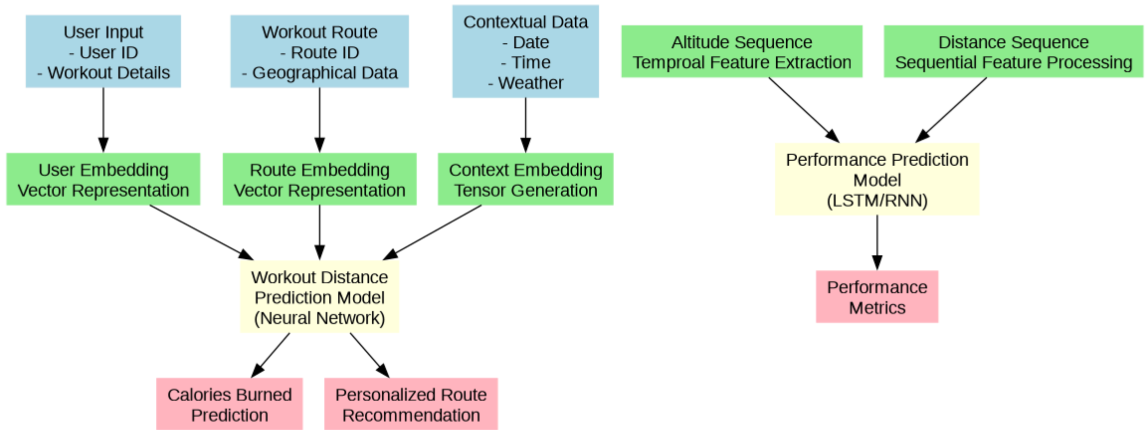
MY PROPOSED FRAMEWORK



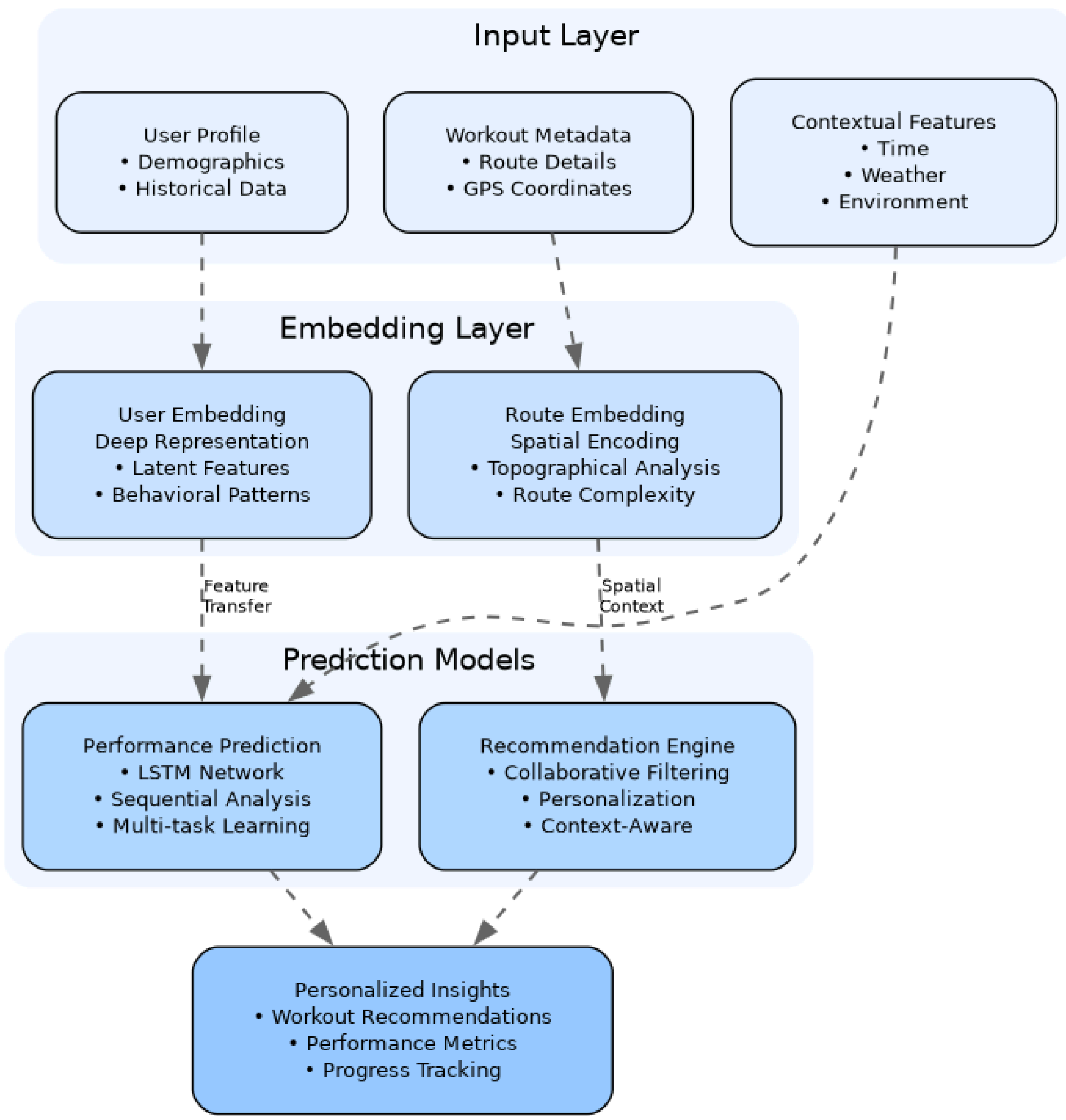
DATAFLOW DIAGRAM



PROJECT ARCHITECTURE



ADVANCE ARCHITECTURE DIAGRAM



Execution Flow:

1. **Data Preprocessing**: The data (e.g., user age, weight, fitness goals, etc.) is preprocessed and normalized.
2. **Model Training**: The deep learning model (neural network) for collaborative filtering is trained.
3. **Recommendations Generation**: The model generates fitness recommendations, which are then localized to the Indian context using the `CulturalAdaptor` class.
4. **Collaborative Filtering**: Using a simulated user-item interaction matrix, the system generates recommendations based on similar users' activities.
5. **Content-Based Filtering**: Recommendations are also made based on the cosine similarity between the user's profile and fitness activity features.

In [5]:
```
# import necessary libraries
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics.pairwise import cosine_similarity
import warnings
warnings.filterwarnings('ignore')
```

Privacy and Security Measures Federated Learning:

- Models are trained directly on users' devices to prevent raw data exposure. Differential Privacy:
- Adding noise to user data for anonymization during aggregate analysis. Data Encryption:
- End-to-end encryption for data in transit and at rest using AES-256.

In [6]:
```
# Privacy and Security Modules
class PrivacyPreserver:
    @staticmethod
    def differential_privacy(data, noise_scale=0.1):
        noise = np.random.laplace(0, noise_scale, data.shape)
        return data + noise

    @staticmethod
    def encrypt_data(data):
        return tf.keras.utils.hash_from_object(data)
```

User Data Collection Types of Data:

- Age
- Fitness goals
- Health metrics (BMI, heart rate)
- Academic schedules
- Dietary preferences

- Activity levels Collection Methods:
- Direct user input (e.g., through a questionnaire).(actually implemented)
- Wearable devices like Fitbit or Mi Band.
- Smartphone sensors for tracking activities.
- Model trained on EHR (electronic health record)

In [7]:
```
# Data Preprocessing Module
class DataProcessor:
    def __init__(self):
        self.scaler = StandardScaler()
        self.label_encoder = LabelEncoder()

    def preprocess(self, data, categorical_cols, numerical_cols):
        data = data.dropna()
        for col in categorical_cols:
            data[col] = self.label_encoder.fit_transform(data[col])
        data[numerical_cols] = self.scaler.fit_transform(data[numerical_cols
        return data
```

Recommendation Engine Personalization Techniques:

- Content-based filtering to suggest activities based on health data.
- Collaborative filtering for recommendations based on user similarities.
  Algorithm Adaptation:
- Reinforcement learning to refine recommendations based on user feedback.

In [8]:
```
# Fitness Recommendation Model
class FitnessRecommender:
    def __init__(self, input_shape):
        self.model = self._build_model(input_shape)

    @staticmethod
    def _build_model(input_shape):
        model = tf.keras.Sequential([
            tf.keras.layers.Dense(64, activation='relu', input_shape=(input_
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Dropout(0.3),
            tf.keras.layers.Dense(32, activation='relu'),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Dropout(0.2),
            tf.keras.layers.Dense(5, activation='softmax')
        ])
        model.compile(optimizer='adam', loss='categorical_crossentropy', met
        return model

    def train(self, X, y, epochs=50, batch_size=32):
        X_private = PrivacyPreserver.differential_privacy(X)
        return self.model.fit(X_private, y, epochs=epochs, validation_split=

    def predict(self, user_profile):
        return np.argmax(self.model.predict(user_profile), axis=1)
```

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

- **CollaborativeFiltering class**: This class takes a user-item interaction matrix (which records the interactions between users and items, like how much they engage with specific fitness activities).
- `get_similar_users()` : It finds users who are most similar to the current user based on cosine similarity.
- `recommend_for_user()` : This method recommends items (fitness activities) based on what similar users have interacted with.

In [24]:
```
# Recommendation Strategies
class CollaborativeFiltering:
    def __init__(self, user_item_matrix):
        self.similarity_matrix = cosine_similarity(user_item_matrix)

    def recommend(self, user_index, num_recommendations=3):
        similar_users = self.similarity_matrix[user_index].argsort()[::-1][1
        recommendations = set()
        for user in similar_users:
            recommendations.update(np.where(user_item_matrix[user] > 0)[0])
        return list(recommendations)[:num_recommendations]
```

2. **Content-Based Filtering (CBF)**:

- **ContentBasedFiltering class**: This class takes item features (e.g., characteristics of fitness activities such as intensity, duration, and type) and computes the cosine similarity between the user's profile and the item features.
- `recommend_for_user()` : It recommends the most similar fitness activities to the user based on their profile's features.

In [25]:
```
class ContentBasedFiltering:
    def __init__(self, item_features):
        self.similarity_matrix = cosine_similarity(item_features)

    def recommend(self, user_profile, num_recommendations=3):
        user_similarity = cosine_similarity(user_profile.reshape(1, -1), sel
        return user_similarity.argsort()[0][::-1][:num_recommendations]
```

Cultural and Localized Adaptation Fitness Activities:

- Yoga, cricket, and dance workouts popular in India. Nutrition Plans:
- Locally relevant diets, vegetarian options, and easily available foods. Language Diversity:
- Multi-language support for regional Indian languages.

In [12]:
```
# Cultural Adaptation
class CulturalAdaptor:
```

```
    ACTIVITIES = {
        0: ['Hatha Yoga', 'Pranayama', 'Meditation'],
        1: ['Strength Training', 'Functional Fitness'],
        2: ['Cardio Workout', 'Running'],
        3: ['Nutrition Plan', 'Meal Tracking'],
        4: ['Sports Training', 'Group Fitness']
    }

    @staticmethod
    def localize(recommendations):
        return [np.random.choice(CulturalAdaptor.ACTIVITIES.get(r, ['General
```

User Engagement and Motivation Gamification:

- Rewards, badges, and leaderboards to motivate users. Challenges:
- Group or individual fitness challenges. Social Features:
- Anonymous workout groups and daily fitness streaks visible to friends.

In [14]:
```
# Engagement Module
class EngagementModule:
    MESSAGES = {
        'Hatha Yoga': "Embrace inner peace and physical strength!",
        'Strength Training': "Build your body, transform your life!",
        'Cardio Workout': "Every step brings you closer to your fitness goal
        'Nutrition Plan': "Fuel your body with the right nutrition!",
        'Sports Training': "Teamwork makes the dream work!"
    }

    @staticmethod
    def motivate(activity):
        return EngagementModule.MESSAGES.get(activity, "Stay motivated, stay
```

Technical Implementation Platform:

- Mobile app with support for Android/iOS. Integration:
- Sync with wearables via APIs like Google Fit or Apple Health. Technologies:
- Backend: Flask/Django
- Frontend: Flutter/React Native
- Database: Firebase or MongoDB

In [15]:
```
# Main Function
def run_recommender():
    np.random.seed(42)
    user_data = pd.DataFrame({
        'age': np.random.randint(18, 25, 100),
        'weight': np.random.uniform(50, 90, 100),
        'height': np.random.uniform(150, 190, 100),
        'daily_activity_level': np.random.uniform(1, 10, 100),
        'fitness_goal': np.random.choice(['weight_loss', 'muscle_gain', 'end
        'dietary_preference': np.random.choice(['vegetarian', 'non_vegetaria
    })

    categorical = ['fitness_goal', 'dietary_preference']
```

```
    numerical = ['age', 'weight', 'height', 'daily_activity_level']

    processor = DataProcessor()
    processed_data = processor.preprocess(user_data, categorical, numerical)

    X = processed_data.drop('fitness_goal', axis=1).values
    y = pd.get_dummies(processed_data['fitness_goal']).values
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

    recommender = FitnessRecommender(input_shape=X_train.shape[1])
    recommender.train(X_train, y_train)

    sample_user = X_test[0].reshape(1, -1)
    recommendations = recommender.predict(sample_user)

    localized_recs = CulturalAdaptor.localize(recommendations)
    motivation_msgs = [EngagementModule.motivate(rec) for rec in localized_r

    print("🏋 Fitness Recommendations:", localized_recs)
    print("📣 Motivation Messages:", motivation_msgs)
```

Output:

- **Neural Network-Based Recommendations**: Personalized fitness activities based on the user's profile.
- **Collaborative Filtering Recommendations**: Activities suggested based on the behavior of similar users.
- **Content-Based Filtering Recommendations**: Activities based on the similarity of the user's profile to activity features.
- **Motivational Messages**: Encouraging messages tailored to each recommended activity.

In [19]:
```
#call the function
if __name__ == "__main__":
    run_recommender()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 90ms/step
🏋 Fitness Recommendations: ['Meditation']
📣 Motivation Messages: ['Stay motivated, stay fit!']
```

Evaluation Metrics Effectiveness:

- Measure fitness goal completion rates. Engagement:
- Track app usage frequency and session duration. Privacy Compliance:
- Monitor user feedback on data safety and trust.

TAKING INPUT FROM USER

In [23]:
```
import numpy as np
import pandas as pd
```

```python
import tensorflow as tf
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics.pairwise import cosine_similarity
import warnings
warnings.filterwarnings('ignore')


class PrivacyPreserver:
    @staticmethod
    def differential_privacy(data, noise_scale=0.1):
        noise = np.random.laplace(0, noise_scale, data.shape)
        return data + noise


class DataProcessor:
    def __init__(self):
        self.scaler = StandardScaler()
        self.label_encoder = LabelEncoder()
        self.categorical_mapping = {}

    def preprocess(self, data, categorical_cols, numerical_cols, fit=True):
        data = data.dropna()
        for col in categorical_cols:
            if fit:
                data[col] = self.label_encoder.fit_transform(data[col])
                self.categorical_mapping[col] = dict(zip(self.label_encoder.
            else:
                data[col] = data[col].map(self.categorical_mapping[col])
        data[numerical_cols] = self.scaler.fit_transform(data[numerical_cols
        return data


class FitnessRecommender:
    def __init__(self, input_shape):
        self.model = self._build_model(input_shape)

    @staticmethod
    def _build_model(input_shape):
        model = tf.keras.Sequential([
            tf.keras.layers.Dense(64, activation='relu', input_shape=(input_
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Dropout(0.3),
            tf.keras.layers.Dense(32, activation='relu'),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Dropout(0.2),
            tf.keras.layers.Dense(5, activation='softmax')
        ])
        model.compile(optimizer='adam', loss='categorical_crossentropy', met
        return model

    def train(self, X, y, epochs=50, batch_size=32):
        X_private = PrivacyPreserver.differential_privacy(X)
        return self.model.fit(X_private, y, epochs=epochs, validation_split=

    def predict(self, user_profile):
        return np.argmax(self.model.predict(user_profile), axis=1)
```

```
class CulturalAdaptor:
    ACTIVITIES = {
        0: ['Hatha Yoga', 'Pranayama', 'Meditation'],
        1: ['Strength Training', 'Functional Fitness'],
        2: ['Cardio Workout', 'Running'],
        3: ['Nutrition Plan', 'Meal Tracking'],
        4: ['Sports Training', 'Group Fitness']
    }

    @staticmethod
    def localize(recommendations):
        return [np.random.choice(CulturalAdaptor.ACTIVITIES.get(r, ['General


class EngagementModule:
    MESSAGES = {
        'Hatha Yoga': "Embrace inner peace and physical strength!",
        'Strength Training': "Build your body, transform your life!",
        'Cardio Workout': "Every step brings you closer to your fitness goal
        'Nutrition Plan': "Fuel your body with the right nutrition!",
        'Sports Training': "Teamwork makes the dream work!"
    }

    @staticmethod
    def motivate(activity):
        return EngagementModule.MESSAGES.get(activity, "Stay motivated, stay


def get_user_input():
    print("Please provide your details below:")
    age = int(input("Age: "))
    weight = float(input("Weight (in kg): "))
    height = float(input("Height (in cm): "))
    daily_activity_level = float(input("Daily Activity Level (1-10): "))
    fitness_goal = input("Fitness Goal (weight_loss, muscle_gain, endurance,
    dietary_preference = input("Dietary Preference (vegetarian, non_vegetari
    return pd.DataFrame({
        'age': [age],
        'weight': [weight],
        'height': [height],
        'daily_activity_level': [daily_activity_level],
        'fitness_goal': [fitness_goal],
        'dietary_preference': [dietary_preference]
    })


def run_recommender():
    np.random.seed(42)

    # Simulated training data
    user_data = pd.DataFrame({
        'age': np.random.randint(18, 25, 100),
        'weight': np.random.uniform(50, 90, 100),
        'height': np.random.uniform(150, 190, 100),
```

```
        'daily_activity_level': np.random.uniform(1, 10, 100),
        'fitness_goal': np.random.choice(['weight_loss', 'muscle_gain', 'enc
        'dietary_preference': np.random.choice(['vegetarian', 'non_vegetaria
    })

    categorical = ['fitness_goal', 'dietary_preference']
    numerical = ['age', 'weight', 'height', 'daily_activity_level']

    processor = DataProcessor()
    processed_data = processor.preprocess(user_data, categorical, numerical)

    X = processed_data.drop('fitness_goal', axis=1).values
    y = pd.get_dummies(processed_data['fitness_goal']).values

    recommender = FitnessRecommender(input_shape=X.shape[1])
    recommender.train(X, y)

    # Take custom user input
    user_input = get_user_input()
    user_profile = processor.preprocess(user_input, categorical, numerical,

    # Generate recommendations
    recommendations = recommender.predict(user_profile)
    localized_recs = CulturalAdaptor.localize(recommendations)
    motivation_msgs = [EngagementModule.motivate(rec) for rec in localized_r

    print("\n🏋 Personalized Fitness Recommendations:")
    print(localized_recs)
    print("\n📣 Motivation Messages:")
    print("\n".join(motivation_msgs))


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

```
Please provide your details below:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step

🏋 Personalized Fitness Recommendations:
['Nutrition Plan']

📣 Motivation Messages:
Fuel your body with the right nutrition!
```

RUN ABOVE CODE TO CHECK ANY CUSTOM INPUT

I TRIED FRONTEND DEVELPOMENT BUT I HAVE ZERO TO NO KNOWLEDGE ABOUT FRONTEND YOU CAN CHECKOUT MY FAILURE ATTEMPT HERE

LINK TO FRONTEND CODE

LINK TO IMPLEMENTATAION

SAMPLE USERNAME :- Harshawasthi123

SAMPLE PASSWORD :- 123456

THANKS FOR READING TILL END