Customer Engagement Dashboard Documentation

1. Introduction

The **Customer Engagement Dashboard** provides insights into user activity, predicts churn likelihood, and helps businesses understand customer retention. The system consists of a **frontend (React)** for visualization and a **backend (Node.js with Python for AI models)** for processing engagement and churn prediction.

2. Technologies Used

- **Frontend**: React, CSS
- **Backend**: Node.js, Express
- **Database**: JSON-based mock data (can be replaced with MongoDB or SQL)
- **Machine Learning**: Python (scikit-learn, pandas)

3. Frontend

1. UserTable.js

- Displays user activity in a table.
- Includes search and filtering options.
- Allows triggering churn prediction & insight generation.

2. Dashboard.js

- Main component housing user analytics.
- Integrates user table and AI insights.

3. AIInsights.js

- Displays AI-generated engagement insights.
- Dynamically updates based on selected user.

4. App.js

- Entry point for the React frontend.
- Manages routing and state.

4. Backend

server.js

- Sets up an Express server to handle API requests.
- Endpoints:
    - /api/predict → Predicts churn based on user data.
    - /api/insights → Generates insights on user engagement.

predict_churn.py

- Implements a machine learning model for churn prediction.
- Uses user activity data (logins, feature usage, session duration, etc.).

generate_insights.py

- Processes user activity data to generate insights.
- Returns recommendations on improving user engagement.

5. Features

- **User Search & Filtering**: Search users by name or email and filter by date range, engagement score, and retention category.
- **Churn Prediction**: Uses machine learning to predict whether a user is likely to churn.
- **AI Insights**: Generates insights into user behavior and engagement trends.
- **Dynamic UI**: Built with React for an interactive user experience.

6. Engagement Score Formula

The **Engagement Score** is calculated based on multiple factors such as logins, feature usage, session duration, and interactions. The formula used is:

$$\text{EngagementScore} = (\text{Logins} \times 0.3) + (\text{FeatureUsage} \times 0.4) + (\text{SessionDuration} \times 0.2) + (\text{Interactions} \times 0.1)$$

Where:

- **Logins**: Number of times a user logs in.
- **Feature Usage**: Frequency of feature interactions.
- **Session Duration**: Total time spent using the platform.
- **Interactions**: Messages, comments, or actions taken.

Each parameter is normalized before being used in the calculation to ensure a balanced score. The **higher** the engagement score, the **more active** a user is.

7. Churn Prediction Logic

- Churn prediction is handled by a **machine learning model** in **predict_churn.py**. It uses a logistic regression classifier trained on:
- **User behavior data** (logins, interactions, last login days, session duration).
- **Threshold-based classification** (if last login > 30 days & engagement score < 50 → likely to churn).

Prediction Categories:

- **"Likely to Churn" (1)** → Users showing disengagement.
- **"Not Likely to Churn" (0)** → Active users.

8. API Endpoints

1. Predict Churn

- **Endpoint**: POST /api/predict
- **Request Body**:

```
{
  "logins": 10,
  "feature_usage": 5,
  "session_duration": 20,
  "interactions": 15,
  "last_login_days": 7
}
```

- **Response**:

```
{
  "churnPrediction": "Likely to Churn"
}
```

2. Generate AI Insights

- **Endpoint**: POST /api/insights
- **Request Body**: Similar to predict churn.
  Response:

```
{
  "insights": "User engagement is declining. Recommend sending a re-engagement email."
}
```

9. Research Findings & Design Choices

Findings from Research

- **Churn models**: Many models rely on **logistic regression** or **random forests** for churn classification.
- **Engagement tracking**: Best practices suggest using **weight-based engagement scores**.
- **UI simplicity**: Users prefer dashboards with **search, filters, and insights**.

Design Choices

- **Dark theme UI** for better visibility and professional look.
- **AI-based predictions** with REST API to enable **scalability**.
- **Filterable tables** for better user exploration.

10. Challenges & Potential Improvements

Challenges Faced

- **API Requests**: Faced difficulties in sending and receiving API requests.
- **Latency in AI requests**: API calls for churn predictions had slight delays.
- **Frontend filtering**: Ensuring filters work in sync with search was tricky.

Future Improvements

- **Optimize AI API responses** with caching for faster results.
- **Improve visualization** using graphs (e.g., engagement trends).
- **Enhance model accuracy** by incorporating **user feedback loops**.