# Documentation for Flask API and React Integration with Machine Learning Models

## Overview

This project integrates machine learning models for disease prediction (diabetes, heart disease, and other diseases) into a web-based tool using Flask for the backend API and React for the frontend UI. The system allows users to interact with predictive models through an intuitive interface.

---

## Project Components

### 1. Machine Learning Models

The following Jupyter Notebooks were used to develop and train the machine learning models:

#### a. Diabetes Prediction Model

- **File:** `Diabetes.ipynb`
- **Purpose:** Predict the likelihood of diabetes based on user-provided medical data.
- **Steps:**
    1. Data preprocessing (handling missing values, feature scaling, etc.).
    2. Model training using algorithms such as Logistic Regression, Decision Trees, or Random Forest.
    3. Evaluation using metrics like accuracy, precision, and recall.

#### b. Heart Disease Prediction Model

- **File:** `Heart_Disease.ipynb`
- **Purpose:** Predict the risk of heart disease.
- **Steps:**
    1. Exploratory data analysis and visualization.
    2. Feature selection and model training.
    3. Deployment-ready model export.

#### c. General Disease Prediction Model

- **File:** `Disease_prediction.ipynb`
- **Purpose:** Predict general diseases based on symptoms or diagnostic data.
- **Steps:**
    1. Dataset preparation and cleaning.

2. Model development and validation.
3. Saving the model for API integration.

---

## 2. Backend: Flask API

The Flask API acts as the middleware between the machine learning models and the React frontend.

**Key Features:**

- **Endpoints:**
  - `/predict`: Predicts other diseases.
  - `/heart`: Provides heart disease prediction based on input data.
  - `/ diabetes`: Accepts user input and returns diabetes prediction results.
  -
- **Model Loading:** Pre-trained models are loaded using libraries like `joblib` or `pickle`.
- **Input Validation:** Ensures the data received from the frontend is in the correct format.
- **Error Handling:** Returns meaningful error messages for invalid or incomplete requests.

**Example Code Snippet:**

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import joblib
import pandas as pd
import pickle
import numpy as np

model = joblib.load("models training/savedmodels/generalmodel/best_model.pkl")
scaler = joblib.load("models training/savedmodels/generalmodel/scaler.pkl")
heart_disease_model = pickle.load(open('models training/savedmodels/heartdisease.pkl', 'rb'))
with open('models training/savedmodels/diabetes_model.sav', 'rb') as file:
    data = pickle.load(file)
    diabetes_model = data['model']
    diabetes_scaler = data['scaler']
app = Flask(__name__)
CORS(app)

@app.route('/predict', methods=['POST'])
def predict_disease_with_saved_model():
    try:
        data = request.json
        temp_f = float(data['temperature'])
```

```
    pulse_rate_bpm = float(data['pulse_rate'])
    vomiting = int(data['vomiting'])
    yellowish_urine = int(data['yellowish_urine'])
    indigestion = int(data['indigestion'])

    user_input = pd.DataFrame({
        'Temp': [temp_f],
        'Pulserate': [pulse_rate_bpm],
        'Vomiting': [vomiting],
        'YellowishUrine': [yellowish_urine],
        'Indigestion': [indigestion]
    })

    user_input_scaled = scaler.transform(user_input)

    predicted_disease = model.predict(user_input_scaled)[0]
    disease_names = {
        0: 'Heart Disease',
        1: 'Viral Fever/Cold',
        2: 'Jaundice',
        3: 'Food Poisoning',
        4: 'Normal'
    }

    return jsonify({
        'prediction': disease_names[predicted_disease],
        'message': 'Disease prediction result'
    })
  except Exception as e:
    return jsonify({'error': str(e)}), 500
```

---

## 3. Frontend: React UI

The React-based user interface allows users to input data and view predictions.

**Key Features:**

- **Forms for Data Input:**
    - Separate forms for each prediction type (diabetes, heart disease, general).
    - Input validation and dynamic feedback.
- **Results Display:**
    - Displays prediction results in a user-friendly format.

    - Provides additional insights like confidence scores or health tips.
- **Integration with Flask API:**
    - Uses `axios` or `fetch` to communicate with the Flask backend.
    - Handles loading states and error messages gracefully.

**Example Code Snippet:**

```
import React, { useState } from 'react';
import axios from 'axios';

function DiabetesForm() {
  const [features, setFeatures] = useState({});
  const [result, setResult] = useState(null);

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await axios.post('/predict, { features });
      setResult(response.data.prediction);
    } catch (error) {
      console.error('Error predicting diabetes:', error);
    }
  };

  return (
    <div>
      <h1>Diabetes Prediction</h1>
      <form onSubmit={handleSubmit}>
        {/* Input fields for features */}
        <button type="submit">Predict</button>
      </form>
      {result && <p>Prediction: {result}</p>}
    </div>
  );
}

export default DiabetesForm;
```

---

# Deployment

## 1. Backend Deployment

- **Environment:** Python 3.11+, Flask.
- **Steps:**
    1. Install dependencies: `pip install flask joblib`.
    2. Run the Flask app: `python app.py`.
    3. Use a production server like Gunicorn for deployment.

## 2. Frontend Deployment

- **Environment:** Node.js, React.
- **Steps:**
    1. Install dependencies: `npm install`.
    2. Start the development server: `npm start`.
    3. Build for production: `npm run build`.

---

# Conclusion

This project demonstrates a complete pipeline from machine learning model development to deployment with a user-friendly interface. By combining Flask and React, the tool offers a scalable and interactive solution for disease prediction.