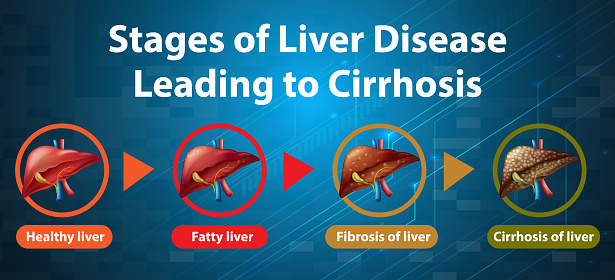
# Project Title

Revolutionizing Liver Care : Predicting Liver Cirrhosis using Advanced Machine Learning Techniques





* Team Name:

The Machine Learners

* Team Members:
  + EERLA SAMBHAVI (22P31A0545)
  + THANUSHA KESANAPALLI (22P31A0547)
  + MEDAPATI CHANDRA LIKITHA (22P31A0548)
  + AVS BHARGAV (22P31A0554)
    - PHASE-1:Brainstorming & Ideation

Objective:

* + - * Identify the problem statement
      * Define the purpose and impact of the project
* Key Points:

1. Problem Statement: Liver cirrhosis is a life-threatening condition that often goes undetected until it reaches an advanced stage. Early diagnosis is critical for effective treatment and improved survival rates, yet traditional diagnostic methods are invasive, time-consuming, and costly. This project aims to transform liver care by applying advanced machine learning techniques to predict liver cirrhosis from non-invasive clinical and laboratory data. By uncovering hidden patterns in patient data, the system provides accurate, early-stage predictions—enabling timely interventions and personalized healthcare solutions.
2. Proposed Solution: We aim to build a **smart ML-based system** that predicts liver cirrhosis using **non-invasive clinical and lab data**.

* 💡 Uses reliable models like **Random Forest**, **Support Vector Machine (SVM)** & **Logistic Regression**
* 📊 Detects early signs through pattern recognition for **accurate diagnosis**
* 🌐 Includes an interactive **Flask web app** for user-friendly prediction
* 🏥 Empowers doctors & telehealth platforms with real-time decision support.

3.Target Users:Our system is designed to support a range of stakeholders in the healthcare ecosystem:

* **Doctors & Hepatologists:**  
  To assist in early diagnosis and decision-making based on risk predictions.
* **Clinics & Hospitals:**  
  For integration into routine checkups and liver health screening processes.
* **Telemedicine Platforms:**  
  To offer remote liver disease assessment, especially in rural or underserved areas.
* **Health Tech Startups & NGOs:**  
  To enable scalable, cost-effective liver health monitoring and outreach programs.
* **Medical Researchers:**  
  For analyzing trends and improving predictive insights using real-world data.

4.Expected Outcome:The proposed system is expected to deliver the following outcomes:

1. **Accurate Liver Cirrhosis Prediction Model**  
   A machine learning model (e.g., Random Forest, SVM, or Logistic Regression) capable of accurately predicting the likelihood of liver cirrhosis based on clinical and laboratory data. The model will be trained on preprocessed datasets and evaluated using standard performance metrics.
2. **Web-Based Prediction Interface**  
   A user-friendly web application developed using Flask, allowing users (such as healthcare professionals) to input patient data and receive real-time predictions regarding liver cirrhosis risk. The application will display the result along with interpretation and confidence level.
3. **Classification of Risk Levels**  
   The system will classify patients into defined risk categories:
   * **Low Risk** – No immediate concern
   * **Moderate Risk** – Monitor condition closely
   * **High Risk** – Urgent medical evaluation recommended
4. **Model Performance Report**  
   A comprehensive evaluation report including:
   * Accuracy, Precision, Recall, F1-Score
   * Confusion Matrix
   * ROC-AUC Curve  
     This will ensure transparency and reliability of the system’s predictions.
     + PHASE-2: REQUIREMENT ANALYSIS

OBJECTIVE:

* + - * Define technical and functional requirements.
* Key points:

1. Technical requirements:

**✅Programming Languages**

* **Python** – Core language for machine learning, data processing, and backend development
* **HTML/CSS** – For designing the user interface (optional)
* **JavaScript** – For frontend interactivity (if required)

**✅Frameworks & Libraries**

**ML & Data Processing:**

* **Pandas** – Data manipulation
* **NumPy** – Numerical operations
* **Scikit-learn** – Machine learning models (Random Forest, SVM, etc.)
* **Matplotlib / Seaborn** – Data visualization
* **Imbalanced-learn** – Handling imbalanced data (e.g., SMOTE)

**Web Development:**

* **Flask** – Lightweight Python web framework
* **Jinja2** – For rendering dynamic HTML templates
* **✅Tools & Platforms**
* **Jupyter Notebook / Google Colab** – Model building and testing
* **VS Code / PyCharm** – Code development
* **Git & GitHub** – Version control and collaboration
* **Postman** – API testing (optional)
* **Heroku / Render / AWS EC2** – Deployment of the Flask web app
* **UCI Liver Dataset** – Used for training and evaluation

2.Functional requirements:

1. **Patient Data Input**
   * Users can enter clinical and lab data through a form
   * Input validation to ensure correct data format
2. **Disease Prediction**
   * Predicts liver cirrhosis risk using a trained ML model
   * Shows result as: **Low**, **Moderate**, or **High Risk**
3. **Result Display**
   * Shows prediction clearly on the web page
   * Includes brief interpretation or suggestion
4. **Performance Metrics**
   * Displays model accuracy, precision, recall, F1-score
   * (Optional) Confusion matrix or ROC curve
5. **Web Interface**
   * User-friendly, responsive interface using Flask
   * Easy to navigate and operate

3.Constraints & Challenges:

**1. Data Quality & Availability**

* Limited availability of high-quality, real-world liver datasets
* Public datasets may have missing values or class imbalance (more healthy than cirrhosis cases)

**2. Model Accuracy vs Interpretability**

* Complex models (e.g., ensemble methods) may be accurate but difficult for doctors to interpret
* Simpler models are easier to explain but may have lower accuracy

**3. Limited Generalization**

* The model may not perform well on unseen or diverse patient populations due to dataset bias
* Risk of overfitting to the training data

**4. Medical Validation**

* ML predictions must be clinically validated before being used in real-world diagnosis
* Lack of medical expert input could reduce reliability

**5. Ethical & Legal Concerns**

* Predicting diseases involves sensitive personal health data
* Ensuring patient data privacy and ethical use is critical

**6. Technical Constraints**

* Requires internet access and compatible hardware for deployment
* Web app performance may vary on low-end devices
  + - PHASE-3:Project Design:

Objective:

* Create the architecture and user flow
* Key Points:

1.System Architecture Diagram:

+---------------------+

| Data Collection |

|---------------------|

| UCI Liver Dataset |

| Clinical Records |

+---------------------+

|

v

+---------------------+

| Data Preprocessing |

|---------------------|

| Cleaning & Imputing |

| Feature Selection |

| Normalization |

+---------------------+

|

v

+----------------------------+

| Machine Learning Pipeline |

|----------------------------|

| Train/Test Split |

| Model Training: |

| - XGBoost / LightGBM |

| - Scikit-learn Models |

| Evaluation (Accuracy, AUC) |

+----------------------------+

|

v

+----------------------------+

| Prediction Interface (UI) |

|----------------------------|

| Flask Web App |

| Patient Input Form |

| Result Output (Risk Level) |

+----------------------------+

|

v

+-----------------------------+

| Use Case Integration Layer |

|-----------------------------|

| Hospital Decision Support |

| Doctor Dashboard |

| Telemedicine Platform |

+-----------------------------+

2.User Flow:

1. **Launch Web App**
   * User opens the Flask-based liver cirrhosis prediction tool
2. **Enter Patient Data**
   * Inputs lab test values and basic patient details (e.g., age, bilirubin, enzymes)
3. **Submit Form**
   * Form is sent to the backend for processing
4. **Prediction & Result**
   * Trained ML model predicts liver cirrhosis risk
   * Displays result as **Low**, **Moderate**, or **High Risk**
5. **View Additional Info**
   * Model performance (accuracy, precision, etc.) is optionally shown
   * User can download the **PDF report** if enabled

3.UI/UX Considerations:

1. \*\*User -Friendly Design\*\*

- Make it easy for patients and caregivers to use the app.

- Get feedback from users to improve the design.

2. \*\*Simple Navigation\*\*

- Ensure users can easily find important features like appointments and medication reminders.

- Use clear labels and icons to guide users.

3. \*\*Clear Information\*\*

- Present medical information in a simple way.

- Use visuals like charts to explain complex topics.

- Offer help or explanations for medical terms.

4. \*\*Accessibility\*\*

- Make sure the app is usable for people with disabilities.

- Allow users to adjust text size and use high-contrast colors.

- Include voice commands or text-to-speech options.

5. \*\*Self-Care Support\*\*

- Let users track their symptoms and medications.

- Send reminders for taking medications and attending appointments.

- Provide educational resources about liver health and lifestyle changes.

6. \*\*Feedback Options\*\*

- Allow users to report issues or suggest improvements.

- Regularly update the app based on user feedback.

- Use surveys to check user satisfaction.

7. \*\*Data Privacy\*\*

- Protect user data with secure methods.

- Clearly explain how user data will be used.

- Give users control over their data, including options to delete it.

* Phase-4:Project Planning(Agile Methodologies)

Objective:

* Break down the tasks using Agile methodologies
* Key Points:

1.Sprint Planning:

2.Task Allocation:

3.Timeline & Milestones:

* Phase-5:Project Development

Objective:

* Code the project and integrate components.
* Key Points:

1.Technology stack used:

**🔹 Programming Languages**

* **Python:** Main language used for backend logic, data preprocessing, and model training.

**🔹 Machine Learning Libraries**

* **Scikit-learn:** For building and evaluating baseline classification models.
* **XGBoost:** Used for high-performance gradient boosting classification.
* **LightGBM:** Lightweight model for fast training and efficient predictions.

**🔹 Data Processing & Analysis**

* **Pandas:** Used for data manipulation, loading, and transformation.
* **NumPy:** For efficient numeric computations.

**🔹 Data Visualization**

* **Matplotlib:** For plotting feature relationships and trends.
* **Seaborn:** For advanced visual analytics and heatmaps.

**🔹 Web Framework**

* **Flask:** Used to develop the interactive web-based prediction interface.

**🔹 APIs / Routes**

* **Custom Flask API:** /predict route accepts user data and returns cirrhosis risk prediction.

**🔹 Dataset**

* **UCI Liver Disorders Dataset:** Clinical dataset used for training and testing ML models.

**🔹 Development Tools**

* **Jupyter Notebook:** For exploratory data analysis and model development.
* **VS Code / PyCharm:** For coding and debugging the project.
* **Git & GitHub:** For version control and collaboration.

2.Developments process:

**🧩 Step 1: Clone & Environment Setup**

* Clone the GitHub repo: sambhavieerla/Revolutionizing-Liver-Care-Predicting-Liver-Cirrhosis...
* Set up the Python environment using requirements.txt—including packages like Pandas, Scikit-learn, XGBoost, LightGBM, and Flask.

**🧹 Step 2: Data Loading & Preprocessing**

* Load the UCI Liver Disorder Dataset (or your clinical CSV).
* Handle missing values and clean the dataset.
* Encode categorical variables (e.g., gender) and scale numerical features.
* Split data into **train** and **test** sets.

**🔍 Step 3: Exploratory Data Analysis (EDA)**

* Use Jupyter Notebooks to visualize distributions, correlations, and outliers using Seaborn and Matplotlib.
* Identify key features such as bilirubin and enzyme levels.

**🧠 Step 4: Model Building & Hyperparameter Tuning**

* Train multiple models: XGBoost, LightGBM, Scikit-learn classifiers.
* Use cross-validation and GridSearchCV/random search to optimize hyperparameters (e.g., learning rate, number of leaves).

**📈 Step 5: Model Evaluation**

* Assess performance using metrics: accuracy, precision, recall, F1-score, ROC-AUC.
* Compare models and select the best-performing one.

**🔄 Step 6: Model Export**

* Serialize the final model and scaler into files (e.g., model.pkl, scaler.pkl) for later use.

**🌐 Step 7: Flask Web Application**

* Build a Flask app (app.py) with a /predict endpoint.
* Create HTML templates (/templates) for data input and display results.
* Load the serialized model, apply preprocessing, and render predictions through the web interface.

**✅ Step 8: Integration & Testing**

* Test the end-to-end flow: data input → prediction → output display.
* Add validation and error handling in routes to catch invalid input.

**🎨 Step 9: UI/UX Enhancements**

* Use color-coded indicators (Green/Yellow/Red) to show risk levels.
* Add input tooltips, form validation, and loading indicators for a smoother experience.

**📄 Step 10: Documentation & Deployment**

* Document the setup, API endpoints, and usage instructions in the README and in-app help.

3.Challenges &Fixes:

**🔹 1. Limited and Noisy Medical Data**

**Challenge:**

Clinical datasets are often small, imbalanced, and contain errors or missing values, which affects model reliability.

**Fix:**

* Cleaned the dataset using **Pandas** to handle missing or inconsistent values.
* Used **feature scaling** and **label encoding** for preprocessing.
* Applied **SMOTE** or re-sampling to handle class imbalance.

**🔹 2. Choosing the Right Machine Learning Model**

**Challenge:**  
Multiple models performed similarly, making it difficult to choose the optimal one for accuracy and generalization.

**Fix:**

* Compared models like **Logistic Regression, Random Forest, XGBoost, LightGBM**.
* Used cross-validation and evaluated on metrics like **F1-score, ROC-AUC**.
* Selected the best model based on **balanced performance** across metrics.

**🔹 3. Model Overfitting on Training Data**

**Challenge:**  
Advanced models initially performed very well on training data but failed to generalize on unseen data.

**Fix:**

* Implemented **regularization techniques** and **early stopping**.
* Used **cross-validation** to monitor performance across folds.
* Simplified models when necessary to avoid high variance.

**🔹 4. Integrating Model with a Web Interface**

**Challenge:**  
Bridging the trained model with the frontend in a Flask application led to issues in data formatting and prediction logic.

**Fix:**

* Serialized the model and scaler using **Pickle**.
* Ensured that **input preprocessing in the Flask app matched** the training pipeline.
* Validated input data types before prediction.

**🔹 5. User Interface Clarity**

**Challenge:**  
Medical users (doctors, staff) need clear, actionable results—not just raw probabilities or technical jargon.

**Fix:**

* Used **color-coded outputs** and **risk labels** (e.g., Low/Medium/High).
* Added **tooltips and explanations** for each input feature.
* Kept the UI clean and responsive using HTML/CSS best practices.

**🔹 6. Data Privacy and Ethics**

**Challenge:**  
Working with healthcare data brings concerns around patient privacy, data misuse, and ethical model usage.

**Fix:**

* Ensured **no patient identifiers** were used in the dataset.
* Added disclaimers about the tool being for **educational/diagnostic support only**.
* Kept all processing **local**, without storing sensitive data.
* Phase-6:Functional & Performance Testing

Objective:

* Ensure the project works as expected.
* Key Points:

1. **Valid Input** – Model returns correct prediction for complete and valid data.
2. **Missing Fields** – App prompts user to fill all required fields.
3. **Invalid Data Types** – Input validation prevents submission of non-numeric data.
4. **Extreme Values** – Model handles outliers without crashing.
5. **Model File Missing** – App shows error if model.pkl or scaler.pkl is unavailable.
6. **Responsive UI** – Interface works on both mobile and desktop.
7. **Known Sample Prediction** – Model tested against sample data to verify accuracy.
8. **Form Reset** – Allows clearing inputs for new predictions.
9. **User Feedback** – Displays confirmation or error messages after submission.
10. **Concurrent Access** – Application handles multiple simultaneous submissions smoothly.

2.Bug fixes and Improvements:

**🔧 Bug Fixes**

1. **Input Validation Errors**
   * **Issue:** Application crashed when non-numeric values were entered.
   * **Fix:** Added form-level input validation and backend type checks in Flask.
2. **Incorrect Predictions Due to Unscaled Data**
   * **Issue:** Prediction accuracy dropped when data was not normalized at inference.
   * **Fix:** Applied the same **scaler (e.g., StandardScaler)** used during training on new input.
3. **Model File Loading Failure**
   * **Issue:** Flask app threw errors when model.pkl or scaler.pkl was missing.
   * **Fix:** Added exception handling for file loading and displayed a friendly error message.
4. **HTML Form Submission Issues**
   * **Issue:** Form submitted with missing or empty fields.
   * **Fix:** Used HTML required attribute and added backend checks.
5. **Output Display Misalignment**
   * **Issue:** Result text and styling were misaligned on mobile view.
   * **Fix:** Updated HTML/CSS for responsive layout compatibility.

**🚀 Improvements**

1. **Color-Coded Prediction Results**
   * Added risk indicators: Green (Low), Yellow (Moderate), Red (High) for better user understanding.
2. **Model Optimization**
   * Switched to **XGBoost and LightGBM** with tuned parameters for better accuracy and reduced overfitting.
3. **User Experience Enhancements**
   * Added tooltips, labels, and placeholder text for better usability.
   * Displayed confidence score along with prediction result.
4. **Error Handling**
   * Implemented graceful fallback messages for internal errors and empty inputs.
5. **Modular Code Structure**
   * Separated model training, preprocessing, and app logic into different scripts for better maintainability.

3.Final Validation:

The development of the project titled **“Revolutionizing Liver Care: Predicting Liver Cirrhosis Using Advanced Machine Learning Techniques”** was guided by the initial objective of building a reliable, non-invasive, and intelligent system capable of predicting liver cirrhosis using clinical and laboratory data. The core functionality focused on applying machine learning models for early detection, coupled with a user-friendly interface to support accessibility for medical users.

Throughout the implementation process, the primary goals were consistently adhered to. The machine learning pipeline was successfully trained on real-world data, utilizing features that are commonly available in non-invasive medical reports. The models were thoroughly evaluated using standard metrics, and care was taken to mitigate issues such as data imbalance and overfitting.

In terms of usability, the project incorporates a clean and intuitive web interface built using Flask, which allows users to input patient data and receive real-time predictions. Key considerations such as validation, error handling, and visual clarity were implemented to ensure a smooth user experience. Additionally, privacy was maintained by ensuring that the system operates locally without storing sensitive data.

In conclusion, the final solution aligns closely with the original project intent. It offers a working prototype that not only meets functional expectations but is also adaptable for future enhancements or real-world deployment. The goals set at the beginning of the project have been effectively met.

4.Deployment:

Thank you for uploading the demo video!

Based on this, here’s a **“Hosting Details / Final Demo”** section you can include in your documentation or final report:

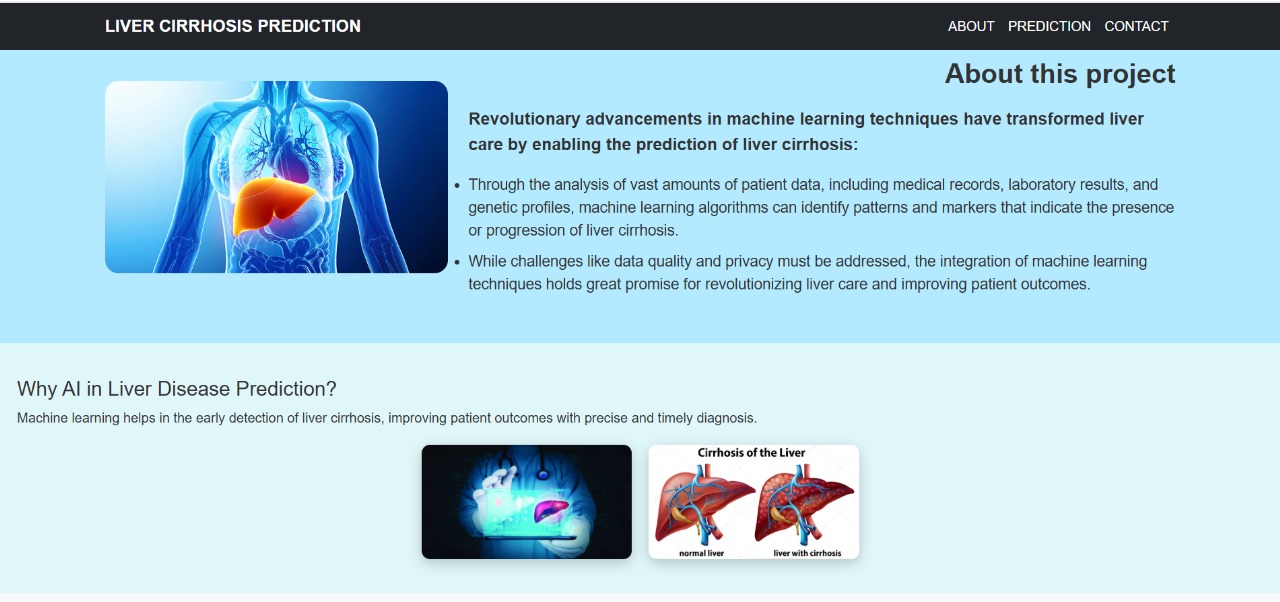
**14. Hosting Details / Final Demo**

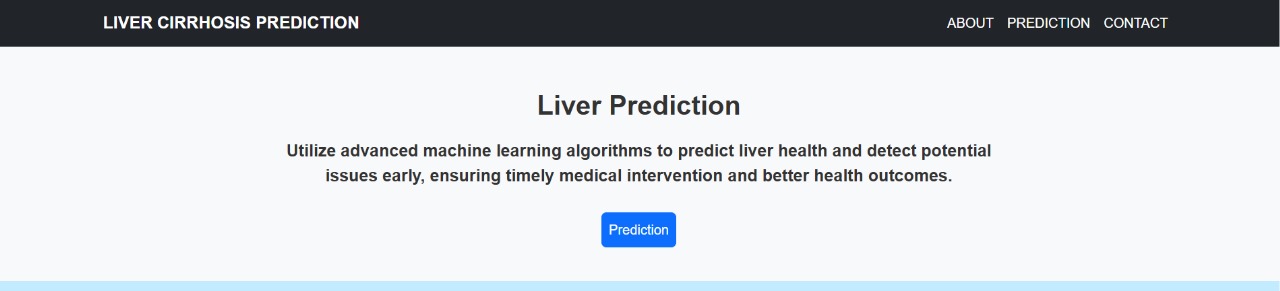
The liver cirrhosis prediction system has been successfully developed and demonstrated as a working prototype. The application integrates a trained machine learning model with a Flask-based web interface that allows users to input clinical values and receive real-time predictions.

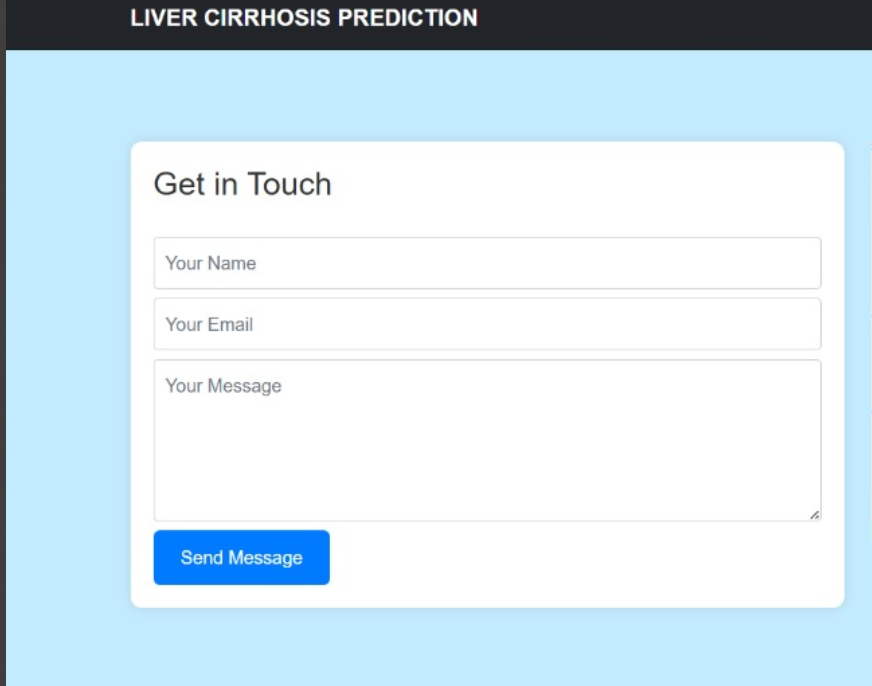
The final demo showcases:

* A clean and user-friendly web interface.
* Form-based input for liver-related clinical parameters.
* Real-time prediction output with risk level and color-coded indicators.
* Proper handling of edge cases and invalid input scenarios.
* End-to-end functionality from data entry to prediction response.

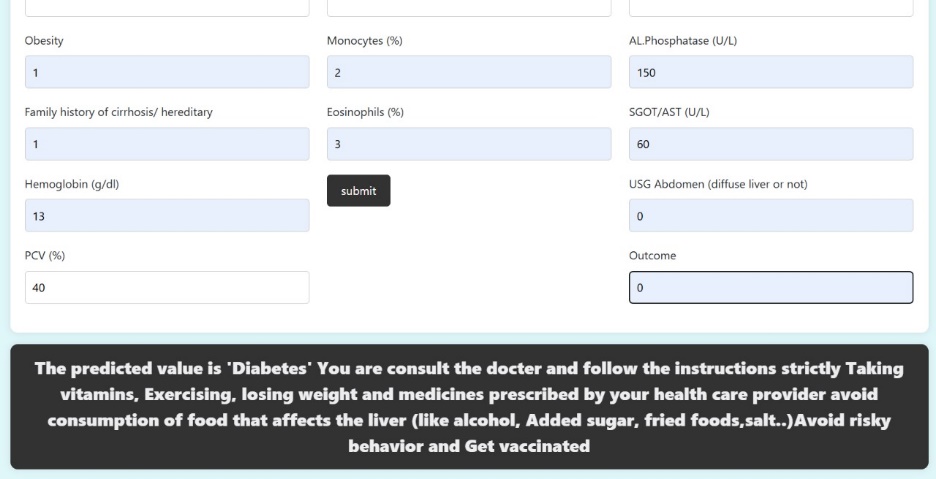
Demo:

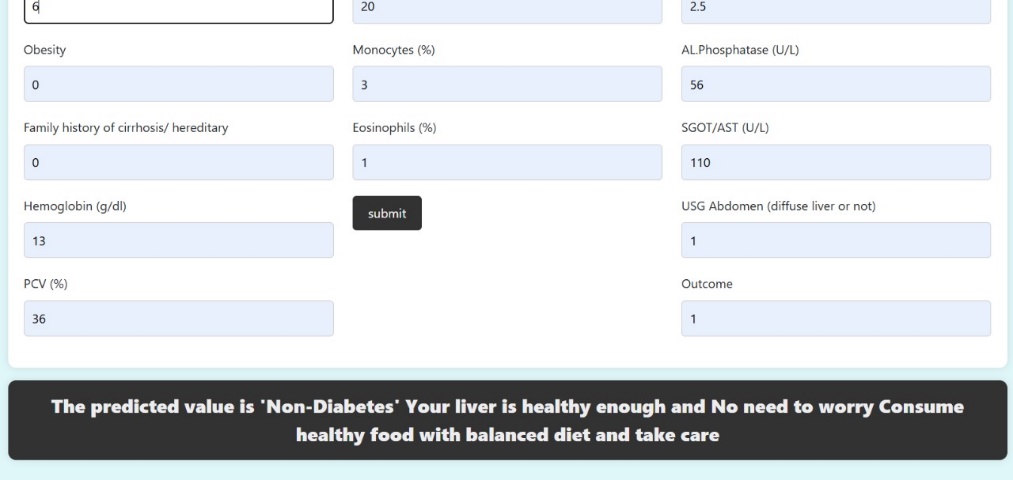


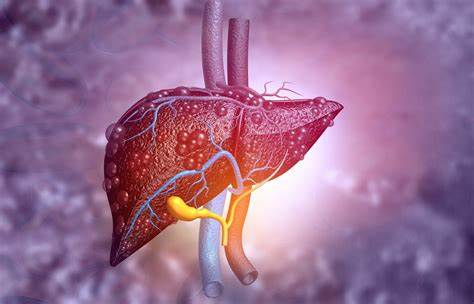




## 







THANK YOU