

# ADVANCE MACHING LEARNING PROJECT

## 1.Introduction

- **Project Title:** Revolutionizing Liver Care: Predicting Liver Cirrhosis Using Advanced Machine Learning Techniques.
- **Team Members:** 1) N. Jayachandra Vedanayakam  
2)N. Ravithreni  
3)N. Mounika  
4) Nayani Spandana

## 2.Project Overview

- **Purpose:** This project aims to develop a machine learning-based predictive model for early detection of liver cirrhosis using clinical and biochemical data. By applying advanced algorithms and performance tuning, the model identifies patients at risk with high accuracy. The best- performing model is deployed using Flask, enabling integration into real-time diagnostic systems. This approach supports faster, data-driven decisions in liver healthcare and improves patient outcomes through early intervention.
- **Features:** The Liver Cirrhosis Prediction project is a comprehensive machine learning solution designed to predict the likelihood of liver cirrhosis in patients based on their medical and lifestyle data. The project involves key steps such as data preprocessing, handling missing values, encoding categorical variables, and normalizing numerical features. Multiple classification algorithms including Logistic Regression, Random Forest, K-Nearest Neighbors (KNN), and XGBoost are trained and evaluated using metrics like accuracy, precision, recall, and F1 score. The best-performing model is selected, saved, and integrated into a Flask-based web application, enabling users to input patient data and receive instant predictions. This project aims to support early diagnosis and assist healthcare professionals in clinical decision-making.

## 3.Architecture

- **Frontend:** The frontend of the Liver Cirrhosis Prediction project is a simple, user-friendly interface built with HTML and CSS. It includes a form where users input

patient details like age, gender, medical history, and test results. Upon submission, the data is sent to the Flask backend, which returns a prediction. The result is then displayed clearly on the same page, making it easy for users to understand whether the patient is at risk of liver cirrhosis.

- **Backend:** The backend of the Liver Cirrhosis Prediction project is built using Flask, a lightweight Python web framework. It handles user input from the frontend, processes the data, and loads a trained machine learning model to make predictions. The backend also includes data preprocessing steps like encoding and scaling to ensure the input matches the model's requirements. Once the prediction is generated, the result is sent back to the frontend for display. The backend ensures smooth integration between the user interface and the predictive model, maintaining accuracy and performance.
- **Database:** The project uses MongoDB to store patient input data and corresponding liver cirrhosis prediction results. Each prediction is saved as a document containing key medical features, the model's outcome and a timestamp. When a user submits data through the frontend, the backend processes the input, predicts the result using a trained model, and stores the full record in the predictions collection using `insert_one()`. MongoDB provides flexible schema design, seamless integration with Flask using PyMongo, and efficient storage of medical records for further analysis.

## 4.Setup Instructions

- **Prerequisites:**
  1. Python 3.x – Core language for development
  2. Pandas – For data manipulation and preprocessing
  3. NumPy – For numerical operations
  4. Scikit-learn – Machine learning models and evaluation metrics
  5. XGBoost – Gradient boosting classifier
  6. Flask – Backend web framework
  7. Flask-PyMongo – To connect Flask with MongoDB
  8. Pickle or Joblib – For saving and loading ML models
  9. MongoDB – NoSQL database to store user inputs and predictions
- **Installation:**
  1. Python 3 installed
  2. Flask installed :`pip install flask`
  3. Clone the repository:  
`https://github.com/Jayachandravedanayakam/revolutionizing-liver-care-predicting-liver-cirrhosis-using-advanced-machine-learning-techniques`
  4. Navigate to project folder:  
`cd liver_cirrhosis`  
`cd project_executable_files`

5. Run the Flask app: `python app.py`

6. Visit: <http://127.0.0.1:5000/>

## 5. Folder Structure

- **Client:** In this project, the client refers to the frontend interface through which users interact with the liver cirrhosis prediction system. The client is built using HTML, CSS, and optionally JavaScript to ensure a responsive and user-friendly experience. Users can input patient data via a structured form, and upon submission, the data is sent to the Flask backend for processing. The client then displays the prediction result (whether the patient is likely suffering from liver cirrhosis or not) clearly on the same or a redirected results page. This interface aims to be intuitive, clean, and accessible for medical professionals or researchers using the system.
- **Server:** The **server** in this project is developed using the Flask web framework, serving as the backend that handles the core logic and communication between the client and the machine learning model. When a user submits input data through the frontend, the Flask server receives the data via HTTP requests, preprocesses it as needed, loads the trained machine learning model, and performs the prediction. It then sends the prediction result back to the client for display. Additionally, the server ensures proper routing, error handling, and integrates seamlessly with the MongoDB database for storing input data and prediction results if required.

## 6. API Documentation

### 1. POST `/predict`

- **Description:** Accepts patient data and returns a prediction indicating whether the patient is likely to suffer from liver cirrhosis.
- **Request Type:** `application/json`
- **Request Body:**

```
{
  "Age": 45,
  "Gender": "male",
  "Duration_of_alcohol_consumption": 10,
  "Hepatitis_B_infection": "positive",
  ...
}
```

- **Response:**

```
{
  "prediction": 1,
  "result": "Patient is likely suffering from liver cirrhosis"
}
```

## 2. GET /

- **Description:** Returns a welcome message or a homepage route.
- **Response:**

```
{  
  "message": "Welcome to the Liver Cirrhosis Prediction API"  
}
```

## 3. POST /store-data *(Optional if using MongoDB)*

- **Description:** Saves patient input and prediction result to the database.
- **Request Body:** Same as /predict
- **Response:**

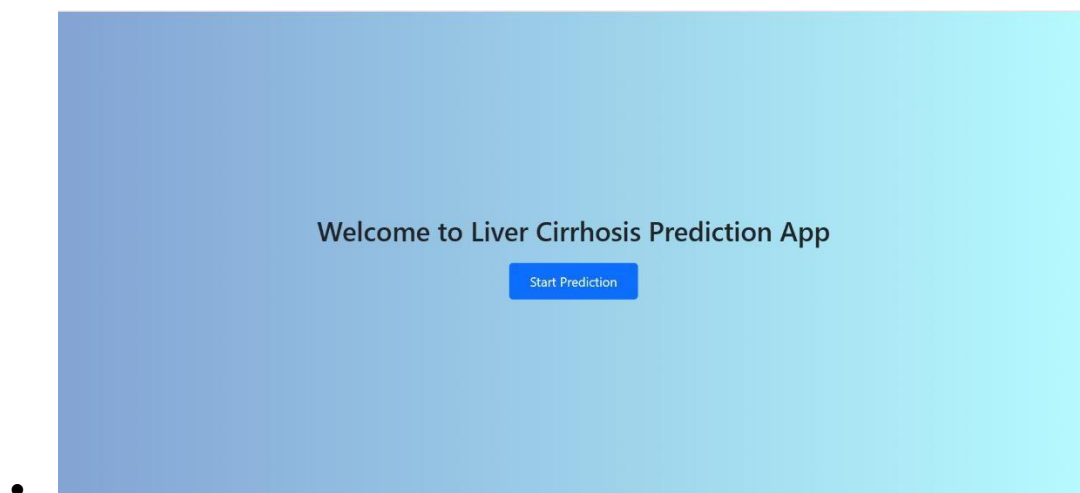
```
{  
  "status": "success",  
  "message": "Data stored successfully"  
}
```

## 8. Authentication

- No specific login or authentication is provided for this project as it can be used by anyone and everyone.

## 9. User Interface

- The User Interface (UI) of the Liver Cirrhosis Prediction project is designed to be clean, user-friendly, and accessible to both medical professionals and non-technical users. It focuses on simplicity and functionality, enabling users to easily input data and receive predictions.



## 9. Testing

- To ensure the accuracy, reliability, and robustness of the Liver Cirrhosis Prediction project, a combination of testing strategies and tools were employed across both machine learning and web application components.

### Testing Strategies:

#### 1. Data Validation Testing:

- Checked for missing values, outliers, and incorrect data types.
- Ensured balanced preprocessing, especially for categorical encoding and numerical normalization.

#### 2. Model Evaluation:

- Used **stratified train-test split** to preserve class distribution.
- Evaluated multiple models (Logistic Regression, Random Forest, KNN, XGBoost) using metrics like **accuracy, precision, recall, and F1-score**.
- Applied **cross-validation** to ensure generalization.

#### 3. API Testing:

- Ensured endpoints respond correctly to valid and invalid inputs.
- Checked for correct HTTP status codes and JSON responses.

#### 4. Integration Testing:

- Validated end-to-end flow from UI to backend to model and back.
- Confirmed correct data handling between frontend forms, Flask server, and MongoDB database.

#### 5. Edge Case Testing:

- Input edge values (e.g., extreme ages, abnormal test results) to observe model and system behavior.

### Tools Used:

- **Scikit-learn** – for model evaluation and metrics.
- **Pytest / Unittest (Python)** – to test backend functions and logic.
- **Postman / cURL** – for manual API testing.
- **Jupyter Notebook** – for exploratory data analysis and testing preprocessing logic.

- **Browser DevTools** – to test and debug frontend interactions.

## 10. Screenshots

**Liver Cirrhosis Prediction Form**

Age  
55

Gender (0=Female, 1=Male)  
1

Eosinophils  
3

Basophils  
1

Platelet Count (in lakhs)  
1.5

Total Bilirubin  
7

AST  
56

56

ASG  
34

Albumin  
3

A/G Ratio  
0.79

Predict

Predict

**Prediction:** Patient likely has Liver Cirrhosis: Yes

## 11. Known Issues

While the Liver Cirrhosis Prediction System functions effectively in most cases, there are several limitations and known issues that users and developers should be aware of:

### 1. Data Imbalance

- The dataset is heavily imbalanced (very few positive cirrhosis cases).
- This may lead to overfitting where models predict the majority class more often, impacting real-world accuracy.
- Mitigation: Stratified splitting, model tuning, and future inclusion of resampling techniques (SMOTE) are recommended.

### 2. Missing or Incomplete Data

- Several features in the original dataset contain missing values, especially in medical metrics like MCH, RBC, TG, and LDL.
- Mitigation: Imputation is performed using mean/mode, but it may affect model reliability. Future data collection should focus on completeness.

### 3. Precision Warning in Classification Report

- In cases where the model predicts only one class, scikit-learn raises a warning: *"Precision is ill-defined and being set to 0.0 in labels with no predicted samples..."*
- Mitigation: Ensuring proper model generalization and handling edge cases helps reduce this issue.

### 4. No Role-Based Access Control

- The current authentication system is basic (login/signup) and does not support role-based access (e.g., doctor vs patient).
- Mitigation: Future versions should implement user roles and permissions for enhanced security.

### 5. Limited Input Validation on Frontend

- The frontend does not fully validate form inputs (e.g., negative numbers, non-numeric entries).
- Mitigation: JavaScript-based and backend validation should be added to prevent malformed data submissions.

## 6. Browser Compatibility

- UI might display inconsistently on older browsers or mobile views.
- Mitigation: Frontend should be tested with responsive design tools and cross-browser testing.

## 7. Model Deployment Limitations

- Model is hosted within the Flask server and may not scale well for high traffic.
- Mitigation: Consider containerizing with Docker and deploying via services like Heroku or AWS for better scalability.

## 13. Future Enhancements

- **Model Explainability**  
Integrate SHAP or LIME to help users understand the reasoning behind predictions.
- **Real-Time API Integration**  
Expose the prediction model via a REST API for real-time use in hospitals or clinics.
- **Mobile-Friendly UI**  
Optimize the frontend for mobile devices to improve accessibility.
- **Role-Based Access Control**  
Add login roles like doctor, admin, or researcher for secure, personalized access.
- **Dashboard and Visualization**  
Add a visual dashboard for tracking patient trends and prediction statistics.