A PROJECT REPORT

on

"SMART BLOOD TEST INTERPRETER"

Submitted to
KIIT Deemed to be University

In Partial Fulfilment of the Requirement for the Award of

BACHELOR'S DEGREE IN
COMPUTER SCIENCE ENGINEERING

BY

PRAKHAR CHOYAL	22051867
PRANJAL AGRAWAL	22051868
SAYAN DAS	22051885
TUSHAR AGARWAL	22051905
SUYASH PANDEY	22052075
VINAYAK PURANIK	22052083

UNDER THE GUIDANCE OF
HIMANSHU RANJAN

SCHOOL OF COMPUTER ENGINEERING
KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY
BHUBANESWAR, ODISHA - 751024

April 2025

KIIT Deemed to be University
School of Computer Engineering
Bhubaneswar, ODISHA 751024

CERTIFICATE

This is certify that the project entitled
"SMART BLOOD TEST INTERPRETER"
submitted by

PRAKHAR CHOYAL	22051867
PRANJAL AGRAWAL	22051868
SAYAN DAS	22051885
TUSHAR AGARWAL	22051905
SUYASH PANDEY	22052075
VINAYAK PURANIK	22052083

is a record of bonafide work carried out by them, in the partial fulfilment of the requirement for the award of Degree of Bachelor of Engineering (Computer Science & Engineering) at KIIT Deemed to be university, Bhubaneswar. This work is done during year 2024-2025, under our guidance.

Date: 07/04/2025

(HIMANSHU RANJAN)
Project Guide

Acknowledgements

We are profoundly grateful to Prof. Himanshu Ranjan of KIIT University for his invaluable guidance, unwavering support, and continuous encouragement throughout this project. His insightful feedback, constructive suggestions, and dedicated mentorship have played a pivotal role in steering this endeavor from its inception to its successful completion. We are deeply appreciative of his time, expertise, and commitment, which have greatly enriched our learning experience and contributed significantly to the realization of our objectives.

PRAKHAR CHOYAL
PRANJAL AGRAWAL
SAYAN DAS
TUHSAR AGARWAL
SUYASH PANDEY
VINAYAK PURANIK

ABSTRACT

At With technology transforming the healthcare space, we've built a smart system that takes your regular blood test results and turns them into meaningful health insights using machine learning. Our project, the Smart Blood Test Interpreter, is designed to help both patients and medical professionals by predicting possible diseases based on numerical blood parameters—offering a quick, supportive diagnostic suggestion before a full medical consultation.

The system is trained on a real-world dataset of around 90,000 patient records, covering a wide range of blood test values and associated diseases. We carefully handled missing values using a two-step imputation approach to ensure the data remained reliable. The core models powering our predictions are Random Forest and XGBoost—both delivering impressive accuracy of over 90%. These models also help identify which test values contribute most to specific disease predictions, adding a layer of interpretability.

To make things simple and interactive, we've deployed the model using Streamlit, allowing users to input their own test values and instantly get predictions. By turning raw blood data into health insights, our system aims to offer a helpful, accessible tool for early detection and awareness.

Keywords: Blood Test Analysis, Machine Learning, Health Prediction, Random Forest, XGBoost, Medical AI, Streamlit App, Smart Healthcare, Disease Risk Detection, Data Imputation

Contents

Chapter 1

Introduction

In today's fast-paced healthcare environment, early and accurate disease detection plays a crucial role in improving patient outcomes. With the continuous surge in medical data, especially diagnostic information like blood test results, there's an increasing demand for intelligent systems that can assist medical professionals in interpreting these reports efficiently. This project, *Smart Blood Test Interpreter*, aims to bridge that gap by leveraging the power of machine learning to analyze numerical blood test data and predict potential diseases, thus offering a supportive tool for both clinicians and patients.

Blood tests are one of the most commonly used diagnostic tools in the medical field, providing a detailed view of a person's physiological state. However, interpreting these results manually can be time-consuming and prone to human error, particularly when dealing with large volumes of patient data. Moreover, slight variations in blood components that may indicate early symptoms of illness often go unnoticed in traditional methods. Our system addresses this issue by using machine learning algorithms—specifically Random Forest and XGBoost—to recognize patterns and correlations in the test data that may point toward specific diseases.

To make the system user-friendly and accessible, the model has been integrated into a web application built using Streamlit. Users can input custom blood test values and receive real-time predictions about possible health conditions. The model has been trained on a comprehensive dataset containing nearly 90,000 anonymized patient records, which include a wide range of blood parameters and corresponding diagnoses. The uniqueness of this system lies in its robust preprocessing pipeline—where missing values are handled through a combination of statistical imputation and K-nearest neighbours—and its careful feature selection tailored to each model. This enhances the predictive accuracy, with both models achieving over 90% on testing data. The end goal is not to replace professional medical advice but to provide an assistive diagnostic layer that enhances decision-making and speeds up the diagnostic process.

In essence, this project represents a confluence of data science and healthcare, demonstrating how technology can be used to derive meaningful insights from routine medical data and potentially improve early diagnosis and treatment planning.

Chapter 2
Basic Concepts/ Literature Review

This section outlines the key ideas and foundational technologies that form the backbone of the Smart Blood Test Interpreter. The project draws from multiple domains including machine learning, medical diagnostics, data preprocessing, and web-based deployment. A strong grasp of these concepts is crucial to understanding how the system predicts potential health conditions based solely on numerical blood test findings.

2.1 Medical Diagnostics through Blood Tests

Blood tests are a vital diagnostic tool in modern medicine. They provide a snapshot of the body's internal state and help detect infections, deficiencies, and chronic conditions. Specific biomarkers, such as white blood cell counts, hemoglobin levels, and platelet counts, serve as indicators for various diseases. Traditionally, interpreting these results requires medical expertise. However, with large volumes of digitized patient data now available, machine learning offers an opportunity to support or automate diagnostic suggestions based on blood profiles.

2.2 Machine Learning in Healthcare

The Machine learning has increasingly found applications in healthcare—ranging from image-based diagnosis to pattern recognition in lab results. In this project, we adopt supervised learning techniques to predict disease classes from structured blood test data. Specifically, **Random Forest** and **XGBoost** are utilized due to their high accuracy, feature importance capabilities, and robustness to noise. These models work by identifying relationships between test results and historical diagnoses across thousands of records, allowing the system to generalize to new, unseen inputs.

2.3.1 Multi-Label Classification in Disease Prediction

Since a single patient can have multiple conditions simultaneously, the problem is framed as a **multi-label classification** task rather than a single-label prediction. This means the model can output several possible diseases per record. Evaluation metrics like **Hamming Loss**, **F1 Score (micro/macro)**, **Precision**, and **Recall** are used to assess the performance, ensuring balanced accuracy even in cases with overlapping conditions.

2.4.1 Model Interpretation and Feature Importance

Understanding *why* a model makes a certain prediction is crucial in healthcare. The project compares **feature importance techniques** from Random Forest (based on impurity reduction) and XGBoost (based on loss reduction/gain). These help highlight which blood test results contribute the most to specific disease predictions—enhancing transparency and trust in the model's decisions.

2.5 **Web-Based Deployment for Accessibility**

To make the system user-friendly, a **Streamlit web application** is developed. Users can manually enter blood test values or upload a CSV for batch predictions. Although the model is pre-trained, users receive instant results alongside a breakdown of probable diseases. The frontend is simple, accessible, and designed for potential integration with clinical interfaces in the future.

Chapter 3
Problem Statement/Requirement Specificiations

The Accurate diagnosis is key to effective treatment, and blood tests are essential in assessing a patient's health. However, interpreting complex test results often requires expert analysis, which can lead to missed patterns or inconsistencies. With the rise of machine learning, there's an opportunity to automate and enhance this process. This project addresses the gap by building a user-friendly web application that uses ML to predict possible diseases from numerical blood test results, making diagnostics more accessible and intelligent.

Problem statement

To design and implement a machine learning-driven system that can interpret blood test parameters and predict probable diseases, making healthcare diagnostics more accessible, intelligent, and efficient. The system processes blood test values, handles missing data intelligently, and outputs disease predictions based on learned patterns from a large dataset, while maintaining usability through a web-based interface.

The system supports single or batch inputs, providing predictions using trained models (Random Forest and XGBoost). Its key focus is accuracy, interpretability, and user-friendliness for both technical and non-technical users.

3.1 Project Planning

The project was executed in structured phases to ensure effective development and deployment:

- **Requirement Gathering:** Identified the core functionalities—data input, preprocessing, disease prediction, model evaluation, and an easy-to-use web interface.
- **Data Preprocessing:** Collected a real-world medical dataset with approximately 90,000 patient records. Addressed missing values using a two-step imputation strategy: median imputation followed by K-Nearest Neighbors (KNN) imputation.
- **Model Selection:** Chose **Random Forest** and **XGBoost** for their high accuracy, robustness to noise, and interpretability in tabular, high-dimensional healthcare data.
- **Model Development:** Implemented the selected algorithms, tuned hyperparameters, and handled multi-label classification. Feature importance was analyzed to understand model decisions.
- **Evaluation:** Measured model performance using accuracy, Hamming loss, precision, recall, and F1 scores (micro/macro). Ensured that evaluation was done on a well-split test set to avoid overfitting.

- **Interface Design:** Utilized **Streamlit** to develop an intuitive and interactive frontend that allows users to enter custom inputs and view model predictions in real time.

- **Documentation:** Maintained clear, structured documentation of each phase, including preprocessing steps, model architecture, evaluation metrics, and usage instructions to ensure reproducibility and ease of understanding.

3.2 **Project Evaluation**

To ensure the system's feasibility and reliability, the following checks and validations were implemented:

- Verified prediction consistency through multiple runs on test data with slight perturbations.
- Ensured imputation logic (median, KNN, mode) preserved the original data distribution and did not introduce bias.
- Analyzed feature importance outputs from both models to validate medical relevance of key test parameters.
- Compared evaluation metrics like Hamming Loss, F1 Score (Micro and Macro), Precision, and Recall to ensure balanced performance across all labels.

Recognized Challenges:

- **Variability in Test Data Distribution:** The dataset exhibited significant variance in numerical ranges across features, making normalization and consistent interpretation essential.
- **Missing Data Patterns:** The dataset had missing values in both random and structured patterns, requiring a robust multi-step imputation process to avoid bias.
- **Class Imbalance:** Some diseases were underrepresented compared to others, leading to skewed learning and necessitating the use of techniques like weighted metrics and sampling strategies.
- **Overlapping Symptoms and Feature Ambiguity:** Several diseases shared similar blood test indicators, making multi-label classification difficult and increasing false positives.
- **Feature Redundancy and Noise:** Certain features had low relevance or were highly correlated, potentially introducing noise that affected model generalization.
- **Model Interpretability:** Despite high accuracy, tree-based ensemble models can be difficult to interpret for non-technical stakeholders, requiring additional tools to explain predictions.

Mitigations included stratified splitting, multi-label metrics, and model tuning.

3.3 System Design

3.3.1 Design Constraints and Requirements

Software Requirements:

- **Programming Language**: Python
- **Framework**: Streamlit for frontend; scikit-learn, XGBoost for ML logic
- **Libraries**: pandas, numpy, matplotlib, seaborn, scikit-learn, xgboost
- **Frontend**: Web-based interface with Streamlit

Hardware Requirements:

- Minimum: 4 GB RAM, dual-core processor
- Recommended: 8 GB RAM or higher for smoother model execution

Development Environment:

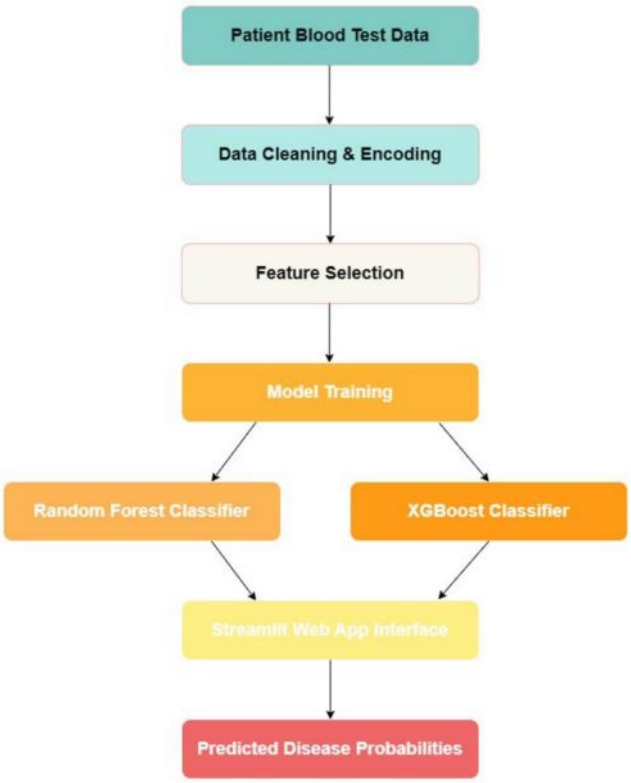
- **OS**: Windows / Linux
- **IDE**: Visual Studio Code / Jupyter Notebook
- **Browser**: Chrome, Firefox, or any modern browser for frontend use

3.3.2 System Architecture

The system follows a modular pipeline design with the following components:

1. **Input Module**: Accepts manual input or CSV file upload containing blood test values.
2. **Preprocessing Module**: Cleans the data, imputes missing values, and prepares it for prediction.
3. **Prediction Module**: Applies the trained Random Forest and XGBoost models to generate multi-label disease predictions.
4. **Output Module**: Displays predicted conditions along with interpretation, using a simple and user-friendly Streamlit interface.
5. **Evaluation Module** (internal): Calculates performance metrics on test data to validate and compare model reliability.

Project Flow Diagram



Chapter 4
Implementation

This chapter outlines the key steps taken during the implementation of the blood test interpretation system. It describes the methodology, testing approach, results, and quality assurance techniques applied throughout the development cycle.

4.1 Procedure

The development process followed a logical and structured approach:

Step 1: Dataset Loading and Preprocessing

- A medical dataset with ~90k blood test records was loaded.
- Initial preprocessing handled inconsistent formatting, missing values, and outliers.
- Two-step imputation was applied: **Median imputation** for numerical fields, followed by **KNN imputation** for improved accuracy.

Step 2: Feature Engineering and Label Transformation

- Irrelevant or highly correlated features were removed.
- Labels were multi-label encoded to support prediction of multiple diseases per patient.
- Features like **Gender**, **Absolute Eosinophil Count**, etc., were selectively dropped per model behavior.

Step 3: Model Development

- Two models were implemented: **Random Forest** (for interpretability) and **XGBoost** (for performance).
- Each model was trained on the processed data and evaluated using cross-validation techniques.
- Feature importance was computed to understand which inputs most influenced predictions.

Step 4: Streamlit Frontend Integration

- A web interface was created using **Streamlit**, allowing users to enter custom blood test values.
- On submit, the system passes values to the ML model and displays predicted diseases.

Step 5: Results and Metrics Display

- Static model evaluation metrics (accuracy, F1 score, Hamming Loss) were calculated on test data.
- Displayed alongside predictions to improve user transparency.

4.2 Validation / Confirmation plan

The system was tested under multiple conditions to ensure its robustness across scenarios such as missing data, skewed samples, and inconsistent values.

ID	Test Case Title	Test Condition	System Behaviour	Expected Result
T01	Valid Input	Numerical input with complete values	Predicts relevant diseases based on trained models	Display list of possible diseases with good accuracy
T02	Missing Values Present	Inputs with some values left blank	Performs imputation and predicts diseases	Handles missing data without crash; accurate prediction
T03	Outlier Detection	Extremely high/low blood test values	Adjusts prediction or flags as outlier	Provides prediction or warns about unrealistic values
T04	Multiple Disease Prediction	Input that maps to overlapping symptoms	Returns multiple disease predictions	Shows all likely conditions with confidence
T05	Rare Disease Class	Input pointing to underrepresented disease	Evaluates using weighted model techniques	Still predicts minor class with reduced false positives
T06	Invalid Data Type	Non-numeric input provided (e.g., text)	System skips or flags incorrect inputs	Prompts user to correct values
T07	Model Comparison (Internal)	Same input passed through Random Forest and XGBoost	Displays both model results side by side	Offers insights into prediction consistency

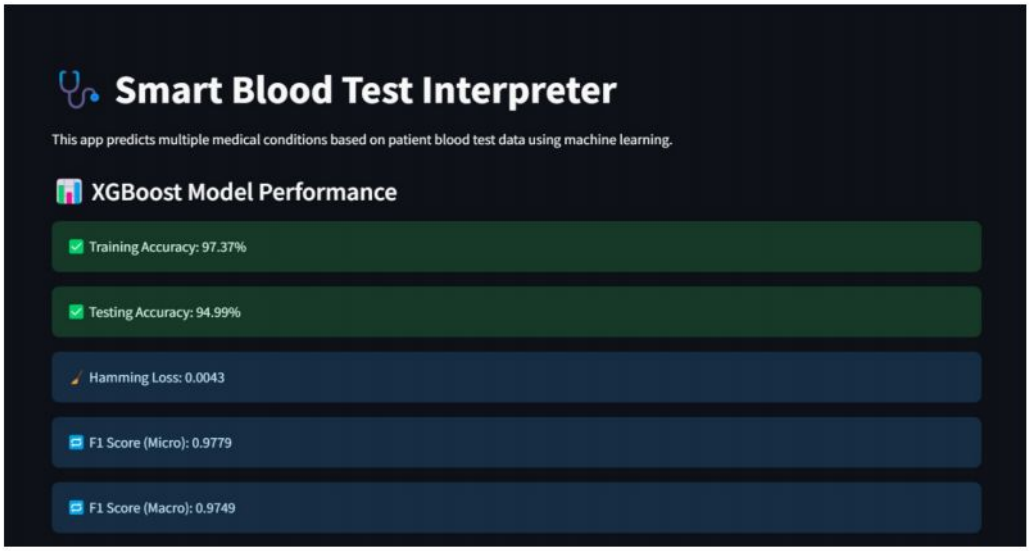
4.3 **Result Analysis / Screenshots**

- Results were visualized in the frontend using clear labels and styling.
- Evaluation metrics were pre-computed and shown to users to reflect model reliability.
- Model outputs were compared using F1 score (micro & macro), accuracy, precision, and recall.

Screenshots

Below are some screesshots from the project :

Screenshot 1: Depicting the dashboard user dashboard showing model metrics

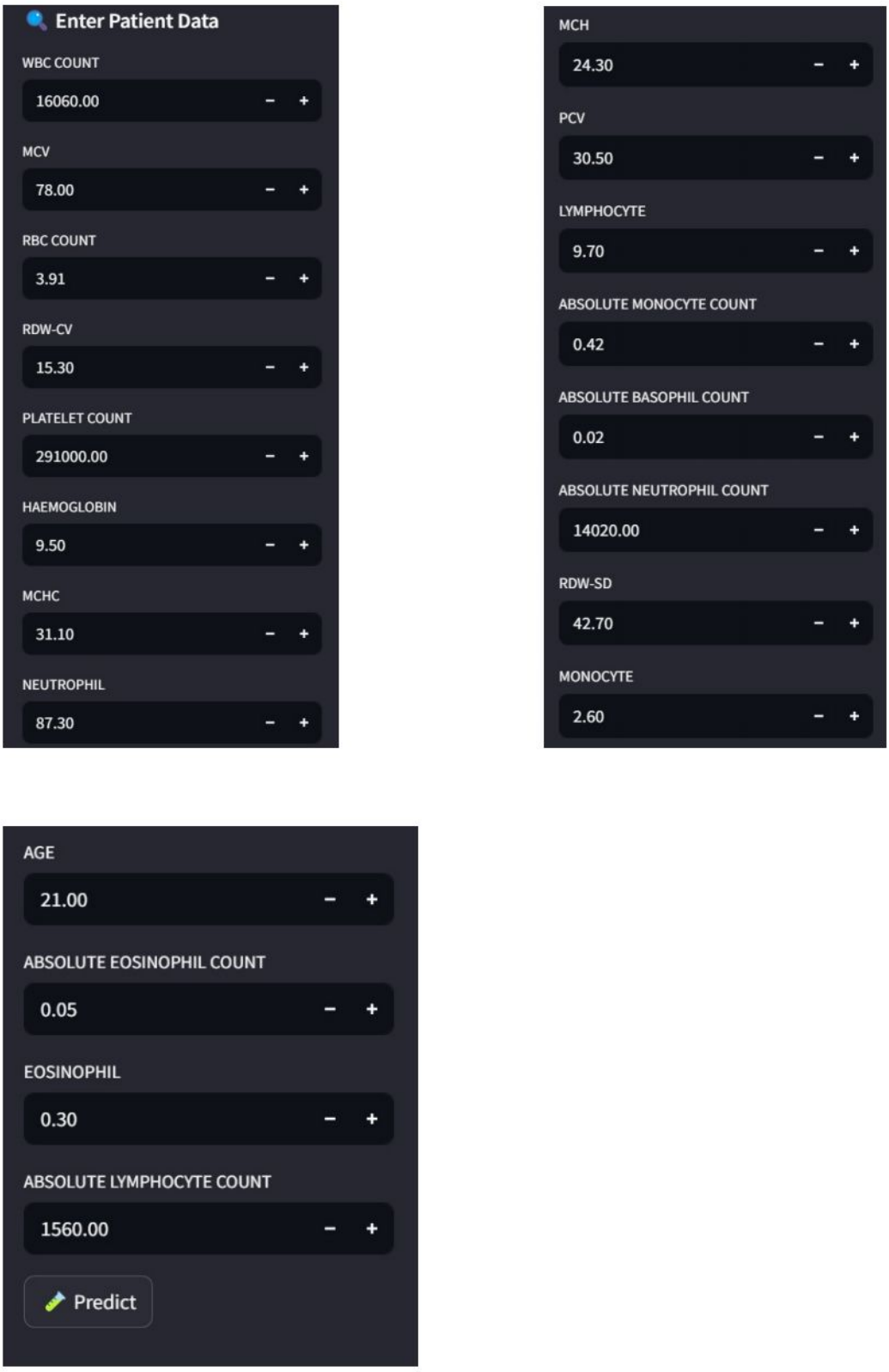


Screenshot 2: Table showing metrics regarding different output parameters

	precision	recall	f1-score	support
No Major Condition Detected	0.96	0.97	0.96	1097
Iron Deficiency Anemia	0.94	0.94	0.94	205
Hemolytic Anemia	0.97	0.96	0.97	951
Vitamin B12 & Folate Deficiency	0.96	0.96	0.96	27
Chronic Kidney Disease	0.99	0.99	0.99	277
Thalassemia	0.91	0.95	0.93	42
Sepsis	1.00	0.99	0.99	83
Liver Disease	1.00	0.99	0.99	287
Dengue	1.00	0.98	0.99	45
Malaria	1.00	0.99	0.99	287
Aplastic Anemia	0.97	0.97	0.97	29
Leukemia	1.00	0.94	0.97	69
Multiple Myeloma	1.00	0.97	0.98	60
Myelodysplastic Syndrome	0.98	0.96	0.97	50
Pernicious Anemia	0.97	0.99	0.98	69
General Infection	1.00	1.00	1.00	1092
Hypothyroidism	0.98	0.97	0.98	546
Possible Autoimmune Disease	1.00	0.97	0.98	64
micro avg	0.98	0.98	0.98	5280
macro avg	0.98	0.97	0.97	5280
weighted avg	0.98	0.98	0.98	5280
samples avg	0.97	0.98	0.97	5280

Screenshot 3: Showing graphical representation of probabilities of predicted disease

Screenshot 4: Showing the input labels for different blood tests from the user



All the above screenshots are with respect to the dashboard for XGBoost model (Similar dashboard exists for Random forest model)

Chapter 5

To ensure quality, maintainability, and performance, the project followed standard software engineering practices in terms of design, coding, and testing. These practices helped improve development efficiency, reduced debugging time, and increased overall system reliability.

5.1 Design Standards

The design phase followed a structured and modular approach to maintain clarity and extensibility:.

- **System Diagrams**: Flow diagrams and block architectures were created to visualize data flow between modules such as preprocessing, model inference, and the user interface.
- **IEEE 1016 Standard** was followed for documenting the software architecture, describing components like the data pipeline, model selection logic, and frontend-backend communication.
- **Modular Architecture**: Functionalities such as data imputation, model loading, prediction logic, and result formatting were encapsulated into independent modules for easier maintenance and reuse.
- **Separation of Concerns**: Streamlit was used strictly for UI rendering, while all core logic was handled in backend scripts, ensuring clean separation between layers.
- **Minimalist Frontend Design**: The web interface followed modern UI design principles for ease of use, with responsive layout optimized for desktops and laptops.

5.2 Coding Standards

Code development followed best practices to enhance readability, modularity, and performance:

- **Descriptive Naming Conventions**: All variables, functions, and classes were named to reflect their responsibilities clearly (e.g., impute_missing_values(), predict_disease()).
- **Modularity**: Each function or script performed a single, specific task—data cleaning, feature selection, model evaluation, etc.—to ensure easy debugging and future enhancements.
- **Consistent Code Style**: Python's PEP 8 guidelines were followed for consistent indentation, spacing, and organization.
- **Inline and Block Comments**: Relevant code blocks were annotated to explain logic, especially in complex parts like data imputation and evaluation metric computation. **Efficient Loops and Data Structures**: Loop operations and condition checks were optimized to reduce time complexity wherever possible.
- **Library Best Practices**: Libraries like pandas, scikit-learn, and xgboost were used per their official documentation to ensure proper implementation and performance.

5.3 **Testing Standards**

Rigorous testing was performed throughout the development cycle to verify system robustness and correctness:

- **Structured Test Case Documentation**: Each test case included IDs, input conditions, expected outputs, and actual observations to maintain traceability and thorough coverage.
- **Functional Testing**: Each module (data input, imputation, model prediction, metrics calculation) was individually tested to ensure correctness.
- **Model Evaluation**: Comprehensive evaluation using metrics like Accuracy, Hamming Loss, F1 Score (micro and macro), Precision, and Recall ensured model reliability on real-world data.
- **Edge Case Testing**: Inputs with missing values, extreme values, and overlapping disease indicators were tested to assess model resilience.
- **User Interaction Testing**: Streamlit interface was tested across different browsers to confirm smooth experience, correct prediction flow, and responsive behavior.

These standards ensured a dependable, transparent, and maintainable system throughout the lifecycle of the Smart Blood Test Interpreter project.

Chapter 6

6.1. Conclusion

The *Smart Blood Test Interpreter* demonstrates how machine learning can be effectively applied to assist in healthcare diagnostics by predicting potential diseases based on numerical blood test results. Through the integration of real-world patient data, advanced preprocessing techniques, and robust classification models like Random Forest and XGBoost, the system successfully bridges the gap between raw medical data and accessible clinical insights.

The web-based platform, developed using Streamlit, offers a clean and user-friendly interface where predictions are generated instantly based on user input. The project follows standard design, coding, and testing practices to ensure reliability, maintainability, and performance across various test scenarios.

Despite its lightweight architecture, the system delivers significant value by making preliminary diagnostic predictions more accessible. It showcases the potential of AI to complement clinical judgment and opens the door to future data-driven healthcare tools that can enhance early detection, reduce diagnostic delays, and support medical professionals.

6.2 Future Scope

While the *Smart Blood Test Interpreter* achieves its core objective, there are several areas where it can be expanded and improved. One major enhancement could involve incorporating more advanced models, such as neural networks or ensemble stacking methods, to better capture complex patterns in blood test data and improve prediction accuracy.

Increasing the diversity and volume of the dataset—by including data from multiple demographics, regions, or additional test parameters—can help the model generalize better and handle more varied patient profiles. This would also support the system in predicting a broader set of diseases.

A potential upgrade could involve integrating the platform into electronic health record (EHR) systems, making it easier for medical professionals to use predictions in real-time clinical settings. Additionally, deploying the system as a mobile application would improve accessibility and allow users to get quick insights anywhere.

Lastly, introducing a feedback system where users or doctors can confirm or reject predictions would help the model learn and evolve over time, increasing both reliability and trust.

References

[1] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). *Scikit-learn: Machine learning in Python.* Journal of machine learning research, 12, 2825-2830.

[2] Chen, T., & Guestrin, C. (2016). *XGBoost: A scalable tree boosting system.* In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785-794). ACM.

[3] Breiman, L. (2001). *Random forests.* Machine learning, 45(1), 5-32..

[4] Streamlit Documentation. *Streamlit: Turn data scripts into shareable web apps.* https://docs.streamlit.io/

[5] Scikit-learn Documentation. *Scikit-learn: Machine Learning in Python.* https://scikit-learn.org/.

[6] Suresh, H., & Guttag, J. V. (2021). *A Framework for Understanding Unintended Consequences of Machine Learning.* Communications of the ACM, 64(7), 62–71.

[7] National Institutes of Health (NIH). *Understanding Blood Tests.* https://medlineplus.gov/lab-tests/ (Accessed 2024)

[8] XGBoost Documentation. *XGBoost Python Package.* https://xgboost.readthedocs.io/