A Major Project Report

On

# PREDICTING HOSPITAL STAY LENGTH USING EXPLAINABLE MACHINE LEARNING

Submitted to CMREC (UGC Autonomous)

In Partial Fulfilment of the requirements for the Award of Degree

of

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING (AI & ML)**

Submitted

By


| | |
|---|---|
| **CHOKKARAPU HARSHINI** | **(228R1A66E3)** |
| **DANDU JYOTHSNA** | **(228R1A66E6)** |
| **KARRE SHIVANI** | **(228R1A66F8)** |
| **PATHANI AKSHAYA** | **(238R5A6614)** |


Under the Esteemed guidance of

**Mrs. M. POOJA**

Assistant Professor, Department of CSE(AI&ML)



**Department of Computer Science and Engineering (AI&ML)**

## CMR ENGINEERING COLLEGE

**(UGC AUTONOMOUS)**

(Accredited by NAAC & NBA, Approved by AICTE, New Delhi, Affiliated to JNTU, Hyderabad)
(Kandlakoya, Medchal Road, Medchal-Malkajgiri Dist., Hyderabad-501 401)


**(2025-2026)**

# CMR ENGINEERING COLLEGE

## UGC AUTONOMOUS

*(Accredited by NAAC&NBA, Approved by AICTE New Delhi, Affiliated to JNTU, Hyderabad, Kandlakoya, Medchal Road, Hyderabad-501 401)*

## Department of Computer Science & Engineering (AI & ML)

## CERTIFICATE

This is to certify that the Major project entitled **"PREDICTING HOSPITAL STAY USING EXPLANATION MACHINE LEARNING"** is a bonafide work carried out by

| | |
|---|---|
| **CHOKKARAPU HARSHINI** | **(228R1A66E3)** |
| **DANDU JYOTHSNA** | **(228R1A66E6)** |
| **KARRE SHIVANI** | **(228R1A66F8)** |
| **PATHANI AKSHAYA** | **(238R5A6614)** |

in partial fulfillment of the requirement for the award of the degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING(AI&ML)** from CMR Engineering College, under our guidance and supervision.

The results presented in this Major project have been verified and are found to be satisfactory. The results embodied in this Major Project have not been submitted to any other university for the award of any other degree or diploma.

| | | |
|---|---|---|
| Internal Guide<br>Mrs. M. Pooja<br>Assistant Professor<br>Department of<br>CSE (AI & ML) | Major Project Coordinator<br>Mr. G. Venkateswarlu<br>Assistant Professor<br>Department of<br>CSE (AI &ML) | Head of the Department<br>Dr. Madhavi Pingili<br>Professor & HOD<br>Department of<br>CSE (AI & ML) |

**External Examiner**_____

# DECLARATION

This is to certify that the work reported in the present Major project entitled "**PREDICTING HOSPITAL STAY USING EXPLANATION MACHINE LEARNING**" is a record of bonafide work done by us in the Department of Computer Science and Engineering (AI & ML), CMR Engineering College. The reports are based on the Major project work done entirely by us and not copied from any other source. We submit our Major project for further development by any interested students who share similar interests to improve the Major project in the future.

The results embodied in this Major project report have not been submitted to any other University or Institute for the award of any degree or diploma to the best of our knowledge and belief.


**CHOKKARAPU HARSHINI (228R1A66E3)**
**DANDU JYOTHSNA (228R1A66E6)**
**KARRE SHIVANI (228R1A66F8)**
**PATHANI AKSHAYA (238R5A6614)**

# ACKNOWLEDGEMENT

We are extremely grateful to **Dr. A. Srinivasula Reddy**, Principal and **Dr. Madhavi Pingili**, Professor & HOD, Department of CSE(AI&ML), CMR Engineering College for their constant support.

We are extremely thankful to **Mrs. M. Pooja,** Assistant Professor, Internal Guide, Department of CSE(AI&ML), for her constant guidance, encouragement and moral support throughout the Major project.

We will be failing in duty if we do not acknowledge with grateful thanks to the authors of the references and other literatures referred in this Major Project.

We thank **Mr. G. Venkateswarlu,** Major Project Coordinator for his constant support in carrying out the Major project activities and reviews.

We express our thanks to all staff members and friends for all the help and co-ordination extended in bringing out this Major project successfully in time.

Finally, We are very much thankful to our parents who guided us for every step.

**CHOKKARAPU HARSHINI (228R1A66E3)**
**DANDU JYOTHSNA (228R1A66E6)**
**KARRE SHIVANI (228R1A66F8)**
**PATHANI AKSHAYA (238R5A6614)**

# CONTENTS


| TOPIC | PAGE NO |
|---|---|
| **ABSTRACT** | **I** |
| **LIST OF FIGURES** | **II** |
| **LIST OF TABLES** | **III** |
| **1. INTRODUCTION** | **1** |
| 1.1. Introduction | 1 |
| 1.2. Project Objectives | 2 |
| 1.3. Purpose of the project | 3 |
| 1.4. Problem Statement | 4 |
| 1.5. Existing System with Disadvantages | 5 |
| 1.6. Proposed System with Advantages | 7 |
| 1.7. Input and Output Design | 9 |
| **2. LITERATURE SURVEY** | **12** |
| **3. SOFTWARE REQUIREMENT ANALYSIS** | **17** |
| 3.1. Modules and their Functionalities | 17 |
| 3.2. Functional Requirements | 19 |
| 3.3. Non-Functional Requirements | 20 |
| 3.4. Feasibility Study | 22 |
| **4. SYSTEM SPECIFICATIONS** | **25** |
| 4.1. Software requirements | 25 |
| 4.2. Hardware requirements | 25 |
| **5. SOFTWARE DESIGN** | **26** |
| 5.1. System Architecture | 26 |
| 5.2. Dataflow Diagrams | 29 |
| 5.3. UML Diagrams | 31 |

**6. CODING AND IMPLEMENTATION** **41**

6.1. Source Code 41

6.2. Implementation 57

**7. SYSTEM TESTING** **60**

7.1. Types of System Testing 61

7.2. Test Strategies 65

7.3. Sample Test Cases 68

**8. EXPERIMENTAL RESULTS** **75**

**9. CONCLUSION AND FUTURE SCOPE** **80**

9.1. Conclusion 80

9.2. Future Scope 81

**REFERENCES** **82**

# ABSTRACT

Accurately predicting the length of stay (LOS) for patients in the Intensive Care Unit (ICU) is essential for effective hospital bed management, resource allocation, and cost optimization. This project presents an explainable machine learning framework designed to classify ICU patients into short-stay or long-stay categories at the time of admission. The system utilizes Electronic Health Record (EHR) data, incorporating clinical parameters to train supervised learning models capable of assisting clinicians in early decision-making. Multiple machine learning techniques were evaluated to determine the most reliable predictive model. Among these, the XGBoost classifier demonstrated outstanding performance, achieving an AUC score of 98%, and consistently outperforming other models across evaluation metrics. To ensure transparency in predictions a critical factor in healthcare settings the system integrates Explainable AI (XAI) methods. These methods provide clear visualizations and reasoning for each prediction, helping clinicians understand key contributing features and improving trust in automated decision-support tools. The resulting framework offers a practical, data-driven solution for improving ICU operational efficiency. By enabling hospitals to anticipate LOS more accurately, the model supports better scheduling, reduced congestion, optimized bed turnover, and enhanced patient care planning. This approach demonstrates strong potential for deployment in real-world hospital environments where effective, interpretable, and high-accuracy prediction tools are essential. the model is designed to be scalable and adaptable across different hospital settings by incorporating diverse patient datasets and continuously updating with new clinical data. Future enhancements may include real-time integration with hospital management systems and the use of deep learning techniques to further improve prediction accuracy and robustness.

**Keywords:** ICU Length of Stay, LOS Prediction, Machine Learning, Explainable AI, XAI Techniques, XGBoost Classifier, Electronic Health Records, EHR Data, Clinical Decision Support, Predictive Modeling.

# LIST OF FIGURES


| S.NO | FIGURE NO | DESCRIPTION | PAGE NO |
|---|---|---|---|
| 1 | 1.6.1 | Block diagram of proposed system | 8 |
| 2 | 5.1 | System Architecture | 26 |
| 3 | 5.2 | Data Flow diagram | 30 |
| 4 | 5.3.1 | Use diagram | 33 |
| 5 | 5.3.2 | Sequence diagram | 35 |
| 6 | 5.3.3 | Activity diagram | 38 |
| 7 | 5.3.4 | Class diagram | 40 |
| 8 | 6.1 | Correlation Diagram | 45 |
| 8 | 7.3.1 | User Registration | 69 |
| 9 | 7.3.2 | User Login | 70 |
| 10 | 7.3.3 | Hospital Stay Prediction | 71 |
| 11 | 7.3.4 | Form Validation | 72 |
| 12 | 7.3.5 | Data Handling | 73 |
| 13 | 7.3.6 | Real-time Prediction with valid Data | 74 |
| 14 | 8.1 | User Parameters | 75 |
| 15 | 8.2 | Result Screen Visualization | 76 |
| 16 | 8.3 | Short Stay Prediction | 77 |
| 17 | 8.4 | Long Stay Prediction | 78 |
| 18 | 8.5 | Invalid Input Validation | 79 |

# LIST OF TABLES


| S.NO | FIGURE NO | DESCRIPTION | PAGE NO |
|---|---|---|---|
| 1 | 2 | Literature Review Summary | 15 |
| 2 | 7.3 | Test Cases | 68 |

# 1. INTRODUCTION

## 1.1 Introduction

The length of stay (LOS) of a patient in a hospital is a critical indicator of healthcare efficiency, resource utilization, and quality of care. In Intensive Care Units (ICUs), where resources such as beds, medical equipment, and trained staff are limited, effective management becomes even more essential. Accurate estimation of ICU LOS plays a significant role in improving patient flow, reducing waiting times, and optimizing hospital operations. Prolonged hospital stays not only increase operational costs but also elevate the risk of hospital-acquired infections and complications, whereas shorter, well-managed stays contribute to better patient outcomes and enhanced service delivery. Therefore, predicting ICU LOS at an early stage is crucial for effective planning and decision-making in modern healthcare systems. **[2], [20].**

Traditionally, clinicians have relied on scoring systems such as APACHE, SAPS, and SOFA to assess patient severity and estimate outcomes. While these systems provide valuable clinical insights, they are not specifically designed to predict ICU LOS with high accuracy. Their generalized nature, lack of disease-specific considerations, and limited adaptability to diverse patient conditions restrict their effectiveness in dynamic hospital environments. Additionally, these traditional approaches often fail to capture complex interactions between multiple clinical variables, leading to less reliable predictions. As a result, there is a growing need for more advanced, data-driven methods that can overcome these limitations and provide precise LOS predictions. **[3], [4], [6].**

With the widespread adoption of Electronic Health Records (EHR), large volumes of patient data have become available, enabling the application of machine learning techniques in healthcare. Machine learning models can analyze complex patterns within clinical data and generate accurate predictions by learning from historical patient records. Supervised learning algorithms, in particular, have shown great potential in classifying ICU patients into short-stay and long-stay categories. These models can incorporate various clinical parameters, including vital signs, laboratory results, and patient history, to provide comprehensive and data-driven insights that support early clinical decision-making. **[1], [5], [10].**

Several machine learning algorithms have been explored for LOS prediction, including Logistic Regression, Decision Trees, Random Forest, Support Vector Machines, and Gradient Boosting techniques. Among these, XGBoost has demonstrated superior performance due to its ability to handle large datasets, manage missing values, and capture complex nonlinear relationships between features. By optimizing both bias and variance, XGBoost provides highly accurate and reliable predictions, making it a suitable choice for healthcare applications where precision is critical **[8], [12].** The use of performance evaluation metrics such as Accuracy, AUC, Sensitivity, Specificity, Precision, and F1-score further ensures the robustness and effectiveness of the predictive model.

Despite the promising performance of machine learning models, one of the major challenges in healthcare applications is the lack of interpretability. Many advanced models operate as "black boxes," providing predictions without clear explanations, which limits their acceptance among clinicians. In critical care settings, it is essential for healthcare professionals to understand the reasoning behind model predictions to ensure trust and accountability. **[3], [4].**

## 1.2 Project Objectives

By the completion of this project, the system demonstrates the following capabilities:

1. To develop a machine learning model to classify ICU patients into short- and long-stay using EHR data.
2. To build a comprehensive clinical dataset including demographics, vitals, lab results, and history.
3. To preprocess data through cleaning, missing value handling, normalization, and encoding.
4. To apply feature selection techniques to identify key factors influencing LOS.
5. To evaluate multiple models (Logistic Regression, Random Forest, SVM, XGBoost) for best performance.
6. To achieve high accuracy using metrics like AUC, Precision, Recall, and F1-score.
7. To integrate Explainable AI (XAI) for transparent and interpretable predictions.
8. To design a scalable system for integration with CIS/CDSS for real-time ICU decision support.

## 1.3 Purpose of the Project

The primary purpose of this project is to develop an intelligent, data-driven system for predicting the ICU Length of Stay (LOS) of patients using Electronic Health Record (EHR) data. The system aims to overcome the limitations of traditional scoring methods by providing accurate, fast, and reliable predictions for classifying patients into short-stay and long-stay categories. This enables healthcare professionals to make timely and informed decisions, improving overall ICU management efficiency. **[1], [4], [6].**

Another important objective of this project is to bridge the gap between theoretical machine learning models and their practical implementation in real-world healthcare settings. While many predictive models achieve high performance in experimental environments, their adoption in hospitals is often limited due to lack of interpretability and integration. This project addresses these challenges by developing a complete framework that combines data preprocessing, model training, and deployment-ready components suitable for clinical use. **[2], [3].**

The system also aims to enhance hospital resource utilization by enabling early prediction of patient stay duration. Accurate LOS prediction helps in efficient bed allocation, reduces overcrowding, minimizes waiting times, and supports better scheduling of medical staff and resources. By improving operational efficiency, the system contributes to cost reduction and improved quality of patient care in critical care environments. **[5], [7], [9].**

Furthermore, the project focuses on incorporating Explainable AI (XAI) techniques to ensure transparency and trust in predictions. Since healthcare decisions require high reliability, the system provides clear explanations for each prediction, allowing clinicians to understand the key factors influencing patient outcomes. This interpretability increases confidence in the model and supports its adoption in Clinical Information Systems (CIS) and Clinical Decision Support Systems (CDSS). **[8], [12].**

In addition, the project contributes to the advancement of intelligent healthcare systems by integrating machine learning with real-time clinical data. The proposed framework can serve as a foundation for future enhancements such as real-time monitoring, adaptive learning models, and integration with hospital management systems. This makes the solution scalable, practical, and suitable for deployment in modern healthcare environments. **[13], [14], [15].**

## 1.4 Problem Statement

The increasing demand for efficient healthcare services has made accurate prediction of ICU Length of Stay (LOS) a critical challenge for hospitals. In Intensive Care Units (ICUs), where resources such as beds, medical equipment, and healthcare professionals are limited, improper estimation of patient stay duration can lead to overcrowding, inefficient resource utilization, increased operational costs, and compromised quality of care. Traditional approaches for estimating LOS rely on clinical judgment and scoring systems such as APACHE, SAPS, and SOFA, which, although useful for assessing patient severity, lack the precision and adaptability required for accurate LOS prediction in dynamic clinical environments. **[2], [4], [12]**.

Although recent advancements in machine learning have introduced data-driven approaches for LOS prediction, many existing models still face significant challenges. These include limited generalization across different hospital settings, dependency on high-quality structured datasets, and reduced performance when handling missing or inconsistent Electronic Health Record (EHR) data. **[3], [5], [13].** Additionally, the complex and nonlinear relationships among clinical variables make it difficult for conventional models to capture meaningful patterns effectively. Variations in patient conditions, treatment protocols, and hospital practices further impact the reliability of these predictive systems in real-world scenarios.

Another major limitation of existing solutions is the lack of interpretability and transparency. Many advanced machine learning models function as "black boxes," providing predictions without clear explanations. In critical healthcare settings, this lack of explainability reduces trust among clinicians and limits the adoption of such systems for decision-making. Furthermore, most existing models are not integrated into Clinical Information Systems (CIS) or Clinical Decision Support Systems (CDSS), making them less accessible and difficult to use in real-time clinical workflows.

There is a need for an accurate and interpretable system to classify ICU patients into short- and long-stay categories at admission. The system should effectively utilize EHR data, handle inconsistencies, and capture complex clinical patterns. Integrating Explainable AI (XAI) ensures transparent and trustworthy predictions for clinicians.

## 1.5 Existing System

Existing approaches to ICU Length of Stay (LOS) prediction primarily rely on traditional clinical scoring systems, where clinicians estimate patient outcomes using tools such as APACHE, SOFA, and SAPS. These systems evaluate disease severity based on predefined clinical parameters, but they are not specifically designed to predict LOS accurately. As a result, their predictions are often limited, less flexible, and may not reflect the dynamic nature of patient conditions in ICU settings. Additionally, these methods depend heavily on clinical judgment, which can lead to variability and inconsistencies in decision-making **[3], [4].**

To address these limitations, early automated approaches were developed using statistical and basic machine learning techniques. Methods such as linear regression and logistic regression utilize selected clinical features from Electronic Health Record (EHR) data to estimate LOS. However, these approaches rely on simplified assumptions and limited feature interactions, making them less effective in capturing the complex and nonlinear relationships present in real-world clinical data. Consequently, their predictive performance is often insufficient for critical healthcare applications **[10], [11].**

Recent advancements have introduced more sophisticated machine learning models, including Decision Trees, Random Forest, Support Vector Machines, and basic gradient boosting techniques. These models improve prediction accuracy by learning patterns from large-scale EHR data. However, they still face challenges such as overfitting, sensitivity to data quality, and reduced generalization across different hospital environments. Variations in patient demographics, treatment protocols, and data inconsistencies further impact their reliability in real-time scenarios **[6], [8], [10].**

Moreover, many existing LOS prediction systems lack interpretability and practical usability. Most models operate as “black boxes,” providing predictions without clear explanations, which reduces trust among clinicians. Additionally, these systems are often developed for research purposes and are not seamlessly integrated into Clinical Information Systems (CIS) or Clinical Decision Support Systems (CDSS). The absence of real-time prediction capabilities and user-friendly interfaces limits their adoption, making it difficult for healthcare professionals to effectively utilize them for ICU resource management and decision-making. **[2], [7], [9].**

### 1.5.1 Disadvantages

- Traditional clinical scoring systems (APACHE, SOFA, SAPS) provide limited accuracy for ICU Length of Stay (LOS) prediction.
- Existing methods fail to effectively analyze complex and high-dimensional Electronic Health Record (EHR) data.
- Statistical models rely on simplified assumptions, leading to poor capture of nonlinear relationships among clinical features

  Many machine learning models operate as "black boxes," lacking interpretability and reducing clinical trust.
- Existing systems show poor adaptability to diverse patient conditions and varying hospital environments.
- Data quality issues such as missing, inconsistent, or noisy EHR data negatively impact model performance.
- Limited generalization of models across different hospitals and patient populations.
- Lack of real-time prediction capability reduces effectiveness in critical ICU decision-making.
- Minimal integration with Clinical Information Systems (CIS) and Clinical Decision Support Systems (CDSS).
- High dependency on technical expertise for model deployment, maintenance, and updates.
- Existing solutions often lack scalability and practical usability in real-world healthcare settings.
- Difficulty in handling imbalanced datasets, which may bias predictions toward common patient categories.
- Lack of standardized frameworks for integrating predictive models into hospital workflows.
- Limited transparency in feature importance, making it hard to identify key factors influencing LOS predictions.

## 1.6 Proposed System

The proposed system introduces an intelligent and explainable machine learning framework for predicting ICU Length of Stay (LOS) using Electronic Health Record (EHR) data. The system is designed to provide an efficient and automated solution for classifying patients into short-stay and long-stay categories, supporting early clinical decision-making and improving ICU resource management. Initially, the process begins with the collection of patient data from EHR systems, including demographics, vital signs, laboratory results, and medical history. This data serves as the primary input for the predictive pipeline.

Once the data is collected, it undergoes a comprehensive preprocessing stage. In this stage, data cleaning is performed to remove inconsistencies and handle missing values. Outlier detection techniques are applied to eliminate abnormal data points, while feature scaling and encoding ensure that the data is transformed into a consistent and model-ready format. Additionally, feature selection methods are used to identify the most relevant clinical parameters, improving model performance and reducing complexity.

The preprocessed data is then passed to the machine learning models, which act as the core predictive component of the system. Multiple supervised learning algorithms, including XGBoost, Random Forest, and Logistic Regression, are utilized to analyze patterns within the data. Among these, XGBoost serves as the primary model due to its high accuracy, ability to handle complex relationships, and robustness in dealing with structured healthcare data. The models are trained and evaluated using performance metrics such as Accuracy, AUC, Precision, Recall, and F1-score to ensure reliable predictions.

To enhance transparency and trust, the system integrates Explainable AI (XAI) techniques such as SHAP. These techniques provide clear visualizations of feature importance and explain how different clinical factors contribute to each prediction. This allows clinicians to understand the reasoning behind the model's output, making the system more interpretable and suitable for real-world healthcare applications.

Finally, the system generates the prediction output, classifying patients into short-stay or long-stay categories. The results, along with their explanations, can be integrated into Clinical Information Systems (CIS) or Clinical Decision Support Systems (CDSS) for real-time use **[2].**

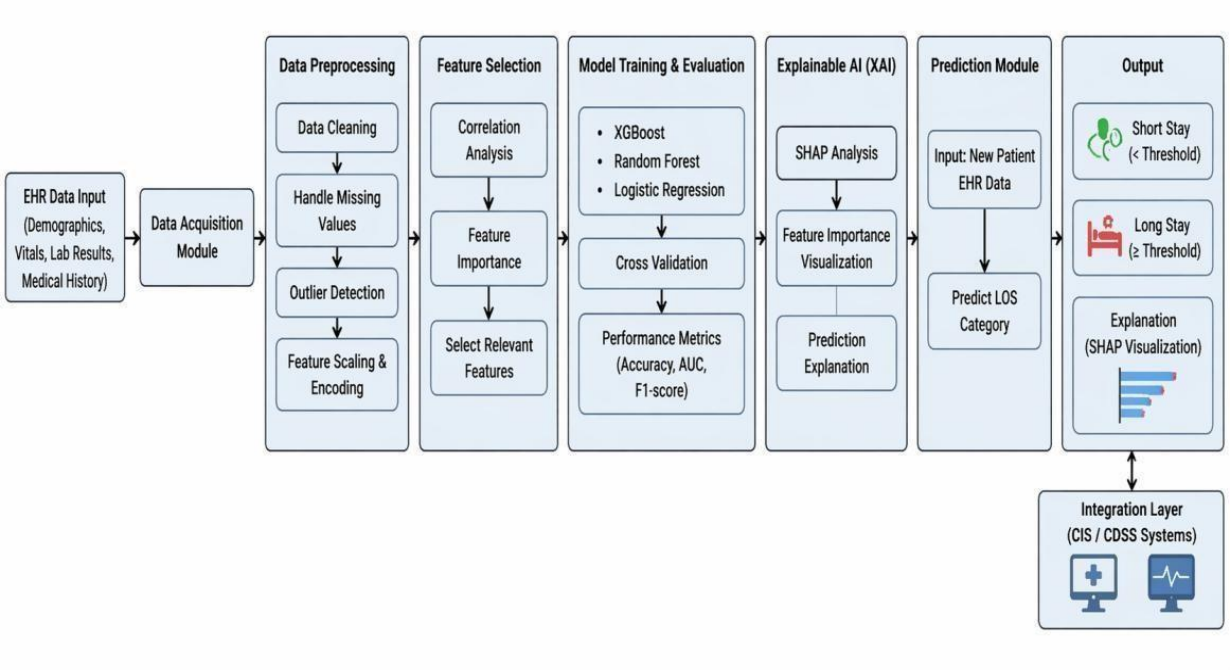


**Fig 1.6.1**: Block diagram of proposed system.

### 1.6.2 Advantages

- High prediction accuracy achieved using advanced machine learning models like XGBoost for ICU Length of Stay (LOS) classification.
- Ability to analyze complex and high-dimensional EHR data, capturing nonlinear relationships among clinical features.
- Reduced manual effort by automating the LOS prediction process, enabling faster clinical decision-making.
- Effective feature selection improves model performance by identifying the most relevant clinical parameters.
- Integration of Explainable AI (XAI) provides transparent and interpretable predictions, increasing clinician trust.
- Robust performance even with real-world clinical data, including handling missing and inconsistent values.
- Real-time prediction capability supports timely ICU bed allocation and resource management.
- Scalable and flexible framework that can be extended to different hospital settings and patient populations.

## 1.7 Input and Output Design

### 1.7.1. Input Design

The input design plays a crucial role in ensuring that the clinical data provided to the system is accurate, consistent, and suitable for predicting ICU Length of Stay (LOS). It defines how Electronic Health Record (EHR) data is collected, validated, and transformed into a structured format that can be processed by machine learning models. In this system, input design focuses on handling diverse clinical data, including patient demographics, vital signs, laboratory results, medical history, and treatment-related information obtained from hospital databases.

The system accepts structured patient records as input through a data interface or hospital database integration. These records may contain different data types such as numerical values (e.g., heart rate, blood pressure), categorical attributes (e.g., gender, diagnosis), and time- based information. To ensure reliability, the input data undergoes validation checks to handle missing, inconsistent, or incorrect values before being processed further. This step minimizes errors and improves the quality of predictions.

A key aspect of input design is data preprocessing and transformation. The system includes automated steps such as handling missing values, normalizing numerical features, encoding categorical variables, and detecting outliers. These processes ensure that all input data is standardized and compatible with machine learning algorithms. Additionally, feature selection techniques are applied to retain only clinically relevant attributes, reducing complexity and improving model performance.

The input design also supports both training and real-time prediction scenarios. During training, the system accepts labeled datasets containing patient records along with their corresponding LOS categories. In real-time usage, new patient data is provided without labels, and the system processes this input to generate predictions. This flexibility makes the system adaptable to both offline model development and live clinical environments.

Furthermore, the system is designed to handle real-world challenges such as incomplete records, data variability across hospitals, and differences in measurement units. The preprocessing module standardizes these variations, ensuring consistent input for the predictive model.

Security and privacy are also important considerations in input design. Since EHR data contains sensitive patient information, the system ensures secure handling of data, with proper anonymization and restricted access where necessary. This helps maintain confidentiality and aligns with healthcare data protection standards.

Overall, the input design ensures that the system operates efficiently by transforming raw clinical data into high-quality, structured input, forming a strong foundation for accurate and reliable ICU LOS prediction.

### 1.7.2 Key points

- To ensure accurate and efficient collection of patient clinical data from Electronic Health Record (EHR) systems.
- To define a structured and consistent format for handling diverse clinical inputs such as vitals, lab results, and demographics.
- To validate and preprocess input data by handling missing values, inconsistencies, and noisy entries.
- To standardize data through normalization and encoding for compatibility with machine learning models.
- To reduce input errors using validation checks and structured data handling mechanisms.
- To support both labeled data for training and unlabeled data for real-time LOS prediction.
- To ensure reliable input processing that improves model accuracy and performance.
- To handle real-world data variations across different hospitals and patient conditions.
- To maintain data privacy and security while processing sensitive patient information.
- To design a scalable input system adaptable for future healthcare data integration and enhancements.
- To enable efficient handling of high-dimensional clinical data without increasing system complexity.
- To ensure seamless integration of input data with preprocessing and feature selection modules.
- To support real-time data input for faster prediction and timely clinical decision-making.

### 1.7.3 Output Design

Output design is a critical component of the proposed system, focusing on how ICU Length of Stay (LOS) predictions are presented to healthcare professionals in a clear, accurate, and interpretable manner. Since the system processes Electronic Health Record (EHR) data and generates predictions using machine learning models, the output must be designed to support effective clinical decision-making. The primary output of the system is the predicted LOS category, typically classified as short-stay or long-stay, based on patient data at the time of admission.

The system presents results in a structured and user-friendly format, ensuring that clinicians can easily interpret the prediction without confusion. In addition to the LOS category, the output may include confidence scores or probability values that indicate the reliability of the prediction. This helps healthcare professionals assess the certainty of the model's output and make informed decisions accordingly.

A key feature of the output design is the integration of Explainable AI (XAI) techniques. The system provides visual explanations, such as feature importance graphs or SHAP-based insights, which highlight the key clinical factors influencing each prediction. This transparency enables clinicians to understand the reasoning behind the model's decisions, increasing trust and supporting better patient care.

The output interface is designed to be clear and organized, emphasizing the most important information such as LOS classification and key contributing features. The system ensures consistency in output presentation across different cases, regardless of variations in input data. Error handling mechanisms are also included to provide meaningful feedback in case of invalid inputs or processing issues.

In addition to clarity, the output design focuses on real-time responsiveness. The system delivers predictions quickly, enabling timely decision-making in critical ICU environments where delays can impact patient outcomes and resource management. This real-time capability enhances the practical usability of the system in hospital workflows.

Furthermore, the output module is designed with scalability and future enhancements in mind. It can be extended to include additional details such as risk levels, recommended actions and integration with Clinical Information Systems (CIS).

# 2. LITERATURE SURVEY

1. **Awad, Bader-El-Den & McNicholas, "A survey on patient length of stay and mortality prediction" *Discover Artificial Intelligence*, 2026.** This study presents a comprehensive review of traditional statistical methods and machine learning approaches used for predicting ICU Length of Stay (LOS) and patient mortality. It highlights key challenges such as missing data, heterogeneity in patient records, and variability across healthcare systems. The paper also discusses commonly used evaluation metrics and provides a strong methodological foundation for designing predictive models. This work is useful as a baseline reference for selecting appropriate algorithms, handling clinical data issues, and evaluating model performance in LOS prediction systems.

2. **Johnson et al. (2026) "MIMIC-III: A freely accessible critical care database," *AIP Conference Proceedings*, 2026.** This study introduces the MIMIC-III public ICU dataset, which is widely used for research in Length of Stay (LOS), mortality prediction, and other clinical analytics. It provides detailed information about dataset structure, patient records, variable definitions, and data usage considerations. The study highlights the importance of standardized and large-scale clinical data for developing reliable predictive models. This dataset serves as a foundational benchmark for ICU prediction research and is extensively used for training and evaluating machine learning models in healthcare.

3. **Ma et al, "Individualized single-classification algorithm for ICU Length of Stay prediction," Springer Conference, 2025.** This study proposes a personalized ICU LOS prediction model using a combination of Extreme Learning Machines and Just-in-Time learning techniques. The approach focuses on capturing patient-specific patterns to provide individualized predictions, demonstrating strong performance across diverse clinical cases. However, the model lacks interpretability, as it does not provide clear explanations for its predictions, which limits its transparency and practical usability in clinical decision-making.

4. **Staziaki et al. (2021), "Combining CT imaging and clinical features to predict ICU admission and Length of Stay," Scientific Reports, 2025.** This study integrates CT imaging data with clinical features to predict ICU LOS and admission outcomes for trauma patients. It demonstrates that multimodal approaches, combining imaging and clinical data, significantly improve prediction accuracy compared to single-source models.

However, the study also highlights challenges such as bias introduced by noisy radiology reports and the lack of proper data cleaning, which can affect model reliability and performance.

5. **Su et al., "Comparative analysis of machine learning models for sepsis ICU outcomes," IEEE Conference, 2024**. This study compares machine learning models such as XGBoost, Logistic Regression, and Random Forest for predicting ICU outcomes including Length of Stay (LOS) and mortality in sepsis patients. The results show that Random Forest achieved the best performance among the evaluated models. However, the study highlights limitations related to dataset constraints, geographic variability, and lack of interpretability, emphasizing the need for more generalized and transparent prediction models in clinical settings.

6. **Staziaki et al., "Combining CT imaging and clinical features to predict ICU admission and Length of Stay," *Procedia Computer Science*, 2024.** This study demonstrates how integrating CT imaging data with clinical variables can significantly improve ICU LOS and admission predictions. It utilizes models such as Support Vector Machines (SVM) and Artificial Neural Networks (ANN) to enhance predictive performance. However, the study also highlights challenges such as bias caused by noisy radiology reports and emphasizes the importance of proper data cleaning and preprocessing when working with multimodal datasets.

7. **Alghatani et al., "Comparison of multiple machine learning classifiers for ICU LOS and mortality prediction using MIMIC-III," Frontiers in Health and Science, 2023.** This study compares six machine learning models, including Logistic Regression (LR), Random Forest (RF), Support Vector Machine (SVM), XGBoost, Linear Discriminant Analysis (LDA), and K-Nearest Neighbors (KNN), using the MIMIC-III dataset (v1.4). The results show that ensemble methods outperform individual models in predicting LOS and mortality. The study also highlights challenges such as variability in vital-sign features and concerns regarding model generalizability across different patient populations.

8. **Gentimis et al, "Predicting hospital Length of Stay using Artificial Neural Networks on MIMIC-III", 2023.** This study applies Artificial Neural Networks (ANNs) on a subset of MIMIC-III dataset features to predict hospital LOS. The model achieves an accuracy of around 80%, demonstrating the potential of neural networks in healthcare prediction tasks. However, the study lacks detailed evaluation metrics such as AUC and sensitivity, making it difficult to compare its performance with more advanced and recent machine learning approaches.

9. **Steele & Thompson.,** ***"Evaluation of predictive models for pre-admission elective Length of Stay".,*** **2023.** This study evaluates seven different predictive models for estimating elective LOS before patient admission. The results show that Bayesian networks outperform other classifiers, achieving an AUC of approximately 90%. The study also emphasizes the importance of clear and consistent feature definitions to ensure model generalizability across different healthcare settings.

10. **Rocheteau, Liò & Hyland 2023, "Temporal pointwise convolutional networks for ICU Length of Stay prediction".** This study explores temporal deep learning architectures specifically designed for time-series ICU data. It demonstrates that temporal models can better capture dynamic patient conditions compared to static models, improving LOS prediction performance. However, the study highlights challenges such as handling irregular time-series sampling, which requires careful preprocessing to ensure reliable results.

**Table. 2** Literature Review Summary

| **Focused Area / Title** | **Key Findings** | **Reference** |
|---|---|---|
| Survey on ICU LOS & Mortality Prediction **[1]** | This study reviews traditional statistical and machine learning approaches for ICU LOS and mortality prediction. It highlights challenges such as missing data, heterogeneity, and evaluation metrics, serving as a methodological baseline for model design. | Awad, Bader-El-Den & McNicholas, "A survey on patient length of stay and mortality prediction," 2026. |
| MIMIC-III Critical Care Dataset **[2]** | Introduces the MIMIC-III dataset widely used for ICU LOS and clinical prediction studies. Provides details on data structure, patient records, and variable definitions, serving as a benchmark dataset for research. | Johnson et al., "MIMIC-III: A freely accessible critical care database," 2026. |
| Personalized ICU LOS Prediction Model **[3]** | Proposes a personalized LOS prediction model using Extreme Learning Machines and Just-in-Time learning. Shows strong individualized performance but lacks interpretability for clinical use. | Ma et al., "Individualized classification algorithm for ICU LOS prediction," 2025. |
| Multimodal ICU Prediction (Imaging + Clinical) **[4]** | Combines CT imaging with clinical data to improve LOS and ICU admission prediction. Uses SVM and ANN models but highlights issues with noisy imaging data and need for proper preprocessing. | Staziaki et al., "Combining CT imaging and clinical features," 2025. |
| Comparative ML Models for Sepsis ICU Outcomes **[5]** | Compares XGBoost, Logistic Regression, and Random Forest for LOS and mortality prediction. Random Forest performs best, but limitations include dataset constraints and lack of interpretability. | Su et al., "Comparative ML models for sepsis ICU outcomes," 2024. |

| Focused Area / Title | Key Findings | Reference |
|---|---|---|
| Multi-Model Comparison using MIMIC-III **[6]** | Evaluates six ML models including LR, RF, SVM, XGBoost, LDA, and KNN. Ensemble models outperform individual ones, but variability in features affects generalization. | Alghatani et al., "ML classifier comparison using MIMIC-III," 2024. |
| ANN-Based LOS Prediction **[7]** | Applies Artificial Neural Networks on MIMIC-III data for LOS prediction, achieving around 80% accuracy. However, lack of detailed metrics limits comparison with modern models. | Gentimis et al., "ANN for hospital LOS prediction," 2023. |
| Elective LOS Prediction Models **[8]** | Evaluates seven predictive models for elective LOS. Bayesian networks achieve highest performance (AUC ~90%) and emphasize importance of feature consistency. | Steele & Thompson, "Predictive models for elective LOS," 2023. |
| Temporal Deep Learning for ICU LOS **[9]** | Explores temporal convolutional networks for LOS prediction using time-series ICU data. Improves performance but requires handling irregular sampling. | Rocheteau, Liò & Hyland, "Temporal CNN for ICU LOS prediction," 2023. |
| Machine Learning vs Traditional Scoring Systems **[10]** | Compares ML models with traditional scoring systems like SOFA and APACHE. Shows ML models provide better predictive accuracy for LOS but require interpretability. | Various studies on ICU LOS prediction (review-based findings), 2023. |

# 3. SOFTWARE REQUREIMENTS ANALYSIS

## 3.1 Modules and Their Functionalities

### 3.1.1. Data Analysis

Data analysis in the proposed ICU Length of Stay (LOS) prediction system focuses on examining the structure, quality, and distribution of Electronic Health Record (EHR) data used for model training and evaluation. The dataset consists of diverse patient-related attributes such as demographics, vital signs, laboratory results, diagnosis details, and treatment history. An in-depth analysis of this data is essential to understand patterns, inconsistencies, and challenges that may impact the performance of machine learning models. Initial observations reveal significant variability in patient conditions, heterogeneous data formats, and differences in measurement scales across clinical features.

The dataset also contains common real-world issues such as missing values, irregular time intervals, and outliers, which require careful handling during preprocessing. Some patient records may have incomplete or inconsistent information, while others may contain extreme or abnormal values that can negatively affect model learning. Additionally, the dataset often exhibits class imbalance between short-stay and long-stay patient categories, which can lead to biased predictions if not properly addressed. Techniques such as data imputation, normalization, and resampling are considered to improve data quality and balance.

Further analysis involves identifying meaningful patterns and relationships among clinical variables. For example, variations in vital signs, laboratory measurements, and patient history may significantly influence LOS outcomes. Statistical analysis techniques such as distribution analysis, correlation matrices, and data visualization are used to gain insights into feature importance and interdependencies. These insights help in selecting the most relevant features for building an effective predictive model.

Moreover, the dataset is evaluated for redundancy and noise, where irrelevant or highly correlated features are identified and removed to reduce complexity. Outlier detection methods are applied to eliminate anomalous records that may distort model performance. Ensuring data consistency and correctness is crucial, as inaccurate or misinterpreted clinical data can lead to unreliable predictions.

### 3.1.2. Data Preprocessing

Data preprocessing is a crucial step in the ICU Length of Stay (LOS) prediction system, aimed at transforming raw Electronic Health Record (EHR) data into a clean, consistent, and structured format suitable for machine learning models. Since EHR data contains a mix of numerical, categorical, and time-based clinical variables, preprocessing ensures that all inputs are standardized and optimized for accurate analysis. This stage focuses on improving data quality and preparing it for effective feature extraction and model training.

The preprocessing pipeline includes handling missing values, which are common in clinical datasets due to incomplete patient records or irregular data collection. Techniques such as imputation using mean, median, or mode values are applied to ensure data completeness. In addition, irrelevant or redundant features are removed to reduce noise and improve model efficiency. Outlier detection methods are also used to identify and eliminate abnormal values that may negatively impact prediction accuracy.

Further preprocessing involves correlation analysis and feature selection to identify the most clinically relevant variables influencing LOS prediction. This step reduces dimensionality and enhances model performance by focusing only on significant features. Additionally, the dataset is divided into training, validation, and testing sets to ensure proper model evaluation and to avoid overfitting. Techniques such as data shuffling and balancing are applied to maintain fair representation of both short-stay and long-stay categories.

### 3.1.3. Machine Learning Algorithm for Prediction

The proposed system utilizes advanced machine learning algorithms to accurately predict ICU Length of Stay (LOS) using Electronic Health Record (EHR) data. Supervised learning techniques are employed due to their effectiveness in handling structured clinical data and identifying complex relationships among patient features. Among the evaluated models, XGBoost is selected as the primary algorithm, supported by Random Forest and Logistic Regression for comparative analysis. These models are chosen for their ability to manage high-dimensional data, handle missing values, and capture nonlinear patterns in clinical variables.

The architecture involves training multiple models on preprocessed EHR data, where each model learns patterns associated with patient outcomes such as short-stay or long-stay categories. Random Forest provides robust performance through ensemble learning, while Logistic Regression offers simplicity and interpretability. XGBoost, a gradient boosting technique, enhances prediction accuracy by iteratively improving weak learners and minimizing errors. Its ability to handle complex feature interactions makes it highly suitable for ICU LOS prediction tasks.

Model training is performed using structured datasets divided into training and testing sets, ensuring proper evaluation and avoiding overfitting. Cross-validation techniques are applied to improve model generalization and reliability. Performance metrics such as Accuracy, AUC, Precision, Recall, and F1-score are used to evaluate each model, and the best-performing model is selected for deployment. Hyperparameter tuning, including learning rate, number of estimators, and tree depth, is carried out to optimize model performance.

The proposed model achieves high accuracy and reliability, making it suitable for real- world healthcare applications. Its design ensures flexibility for future enhancements, such as integrating additional models, incorporating real-time data streams, and improving interpretability. Overall, the machine learning framework provides a robust, scalable, and interpretable solution for ICU LOS prediction, supporting efficient decision-making and hospital resource management.

## 3.2 Functional Requirements

The functional requirements define the essential operations that the proposed ICU Length of Stay (LOS) prediction system must perform to effectively classify patients and support clinical decision-making. These requirements describe how the system should accept Electronic Health Record (EHR) data, preprocess clinical inputs, apply machine learning models, and generate accurate and interpretable LOS predictions. They ensure that the system operates efficiently, consistently, and reliably within healthcare environments.

In addition to core functionalities, the system ensures seamless interaction between all components, including data input modules, preprocessing pipelines, machine learning models, and Explainable AI (XAI) mechanisms. The workflow is designed to handle complex clinical data efficiently, minimize processing delays, and provide real-time predictions.

Furthermore, the system supports integration with Clinical Information Systems (CIS) and Clinical Decision Support Systems (CDSS), enabling smooth incorporation into existing hospital workflows. It is designed to be scalable and extensible, allowing future enhancements such as multi-class LOS prediction, real-time monitoring, risk assessment, and advanced visualization tools. Overall, the functional requirements ensure that the system delivers accurate, interpretable, and practical LOS predictions while supporting efficient ICU resource management and improved patient care.

- The system shall accept Electronic Health Record (EHR) data as input for ICU LOS prediction.
- The system shall validate input data and handle missing or inconsistent clinical records.
- The system shall preprocess data through cleaning, normalization, encoding, and feature selection.
- The system shall apply trained machine learning models to classify patients into short-stay or long-stay categories.
- The system shall generate prediction outputs in a clear, structured, and clinically interpretable format.
- The system shall provide probability scores indicating confidence in predictions.
- The system shall integrate Explainable AI (XAI) techniques to display feature importance and prediction reasoning.

## 3.3 Non-Functional Requirements

Non-functional requirements define the quality attributes, performance standards, and operational constraints of the proposed ICU Length of Stay (LOS) prediction system. These requirements ensure that the system not only performs its intended functions but also delivers reliable, efficient, and secure performance in real-world healthcare environments. They play a vital role in enhancing system usability, stability, and overall effectiveness for clinicians and hospital staff.

The system is designed to be highly reliable, ensuring consistent performance even when processing large volumes of patient data or handling multiple requests simultaneously. Error-handling mechanisms are incorporated to manage missing data, invalid inputs, or unexpected system failures gracefully. This improves system robustness and builds trust among healthcare professionals who rely on accurate predictions for decision-making.

Usability is a key aspect of the system design. The interface and output presentation are designed to be clear, simple, and clinically interpretable, allowing healthcare professionals to easily understand predictions and insights. Minimal user interaction is required, ensuring that the system can be used efficiently within busy clinical environments. The system is also designed to be accessible across different platforms and compatible with existing hospital systems.

Performance and responsiveness are critical in ICU settings. The system is optimized to provide real-time or near real-time predictions with minimal latency, enabling timely decision-making for patient care and resource allocation. Efficient data processing techniques are used to ensure fast computation without compromising accuracy. Security and data privacy are essential, as the system deals with sensitive patient information. Proper data protection mechanisms, access control, and secure data handling practices are implemented to ensure confidentiality and compliance with healthcare regulations. Maintainability is ensured through a modular design, allowing easy updates to models, algorithms, and system components without affecting overall functionality. Overall, the non-functional requirements ensure that the system is reliable, efficient, scalable, secure, and user-friendly, making it suitable for deployment in modern healthcare environments.

- The system shall remain scalable and capable of handling large and continuously growing EHR datasets.
- The system shall deliver accurate and reliable ICU LOS predictions with minimal errors.
- The system shall provide fast and efficient processing to support real-time clinical decision-making.
- The system shall ensure strong security and confidentiality of patient health data.
- The system shall maintain high reliability and stable performance under multiple concurrent requests.
- The system shall provide a user-friendly and clinically interpretable output interface.
- The system shall be compatible with existing hospital systems such as CIS and CDSS.
- The system shall require minimal maintenance and support easy updates and model improvements.
- The system shall ensure consistent performance across different hospital environments and datasets.

## 3.4 Feasibility Study

The feasibility study evaluates the practicality and viability of the proposed Hospital Stay Length Prediction System using Explainable Machine Learning. It ensures that the system can be successfully developed and implemented within the available resources, time constraints, and technical capabilities. This study helps in identifying potential risks, benefits, and limitations before actual system development.

In addition to evaluating economic, technical, and social feasibility, the study also considers operational feasibility, which ensures that the system can be effectively used in real- world healthcare environments such as hospitals and clinics. Risk assessment is performed to identify challenges such as data privacy concerns, data quality issues, model interpretability, and integration with existing hospital management systems. Strategies are developed to mitigate these risks and ensure smooth implementation.

The feasibility study confirms that the proposed system is practical, cost-effective, and beneficial for healthcare applications. It demonstrates that the system can provide significant advantages in optimizing hospital resource allocation, improving patient care, and reducing operational costs while maintaining efficiency and transparency through explainable machine learning models.

The feasibility of the project is analyzed in this phase and a business proposal is put forth with a general plan for the project along with estimated costs. During system analysis, the feasibility study of the proposed system is carried out to ensure that the system does not become a burden to the organization. Three key considerations are involved in the feasibility analysis are:

1.Economic Feasibility
2.Technical Feasibility
3.Social Feasibility

### 3.4.1 Economic Feasibility

This study is carried out to evaluate the economic impact that the system will have on the healthcare organization. The amount of funds that can be allocated for the research and development of the system is limited, and all expenditures must be justified. The developed system fits well within the budget, as most of the technologies used are freely available.

Economic feasibility assesses whether the benefits of the system outweigh the costs involved in its development and deployment. The proposed system is highly cost-effective as it primarily uses open-source tools and technologies such as Python, Scikit-learn, TensorFlow, and Flask. The development cost is minimal since no expensive hardware or licensed software is required.

The system can be implemented using standard computing infrastructure available in most hospitals, making it affordable for both small clinics and large healthcare institutions. Additionally, the system helps optimize hospital resource utilization by accurately predicting patient length of stay, which leads to better bed management, reduced overcrowding, and improved operational efficiency.

Maintenance costs are also low due to the modular and scalable design of the system. Updates, improvements, and integration with existing hospital management systems can be carried out without significant financial investment. Furthermore, the use of explainable machine learning reduces the need for extensive manual analysis, saving both time and labor costs.

In addition, the system contributes to long-term economic benefits by improving patient flow, reducing unnecessary hospital stays, and enhancing decision-making processes. Overall, the system provides a high return on investment by minimizing operational costs and maximizing healthcare efficiency, making it economically sustainable.

### 3.4.2 Technical Feasibility

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not place a high demand on the available technical resources, as this would increase the burden on the organization. The developed system is designed with modest requirements, ensuring that only minimal or no significant changes are required for implementation within existing hospital infrastructure.

Technical feasibility evaluates whether the required technology, tools, and expertise are available to develop and deploy the system. The proposed system utilizes well-established and widely used technologies such as machine learning frameworks (Scikit-learn, TensorFlow) and web development tools (Flask, HTML, CSS).

The system architecture is designed to be simple, efficient, and reliable, ensuring that it does not require high-end hardware or complex infrastructure. The machine learning models used for predicting hospital stay length are optimized to run efficiently on standard computing systems while maintaining good accuracy and performance. Additionally, explainability techniques such as feature importance and SHAP values are incorporated without significantly increasing computational complexity.

The system is flexible and scalable, allowing future enhancements such as integration with electronic health records (EHR), cloud-based deployment, and advanced predictive models. Since the technical requirements are minimal, accessible, and supported by existing technologies, the project is considered highly feasible from a technical perspective.

### 3.4.3 Social Feasibility

The aspect of this study is to check the level of acceptance of the system by its users. This includes the process of training healthcare professionals to use the system efficiently. The users must not feel threatened by the system; instead, they should accept it as a supportive tool in their decision-making process. The level of acceptance depends on the methods used to educate users about the system and make them familiar with its functionality. Their confidence must be enhanced so that they can provide constructive feedback, which is valuable since they are the end users of the system.

SSocial feasibility examines the level of acceptance and usability of the system among its intended users, such as doctors, nurses, hospital administrators, and support staff. The proposed system is designed to be user-friendly and easily accessible, ensuring that users can interact with it without requiring advanced technical knowledge. The interface presents predictions along with explanations, enabling healthcare professionals to understand the reasoning behind the predicted hospital stay length.

Training requirements are minimal, and users can quickly adapt to the system with basic guidance. Feedback mechanisms can be incorporated to continuously improve the system based on real-world usage and user suggestions. Overall, the system is socially acceptable, user-friendly, and beneficial for the healthcare community.

# 4. SYSTEM SPECIFICATIONS

## 4.1 Software Requirements

The software requirements include the tools, libraries, and development environment necessary to build and execute the Hospital Stay Length Prediction System using Explainable Machine Learning. These requirements define the overall system setup including the programming environment, external dependencies, design constraints, and supporting software essential for implementation, testing, and deployment.

- Operating System: Windows / Linux / macOS
- Programming Languages: Python.
- Developer Tools: Visual Studio Code/ Jupyter Notebook
- Libraries: Scikit-learn, Pandas, NumPy, Matplotlib, Seaborn, SHAP, Lime
- Machine Learning Framework**:** Regression Models / Ensemble Methods (Random Forest, XGBoost)
- Explainability Tools: SHAP (SHapley Additive Explanations), LIME (Local Interpretable Model-agnostic Explanations)

## 4.2 Hardware Requirements

The hardware requirements define the physical components needed to develop, test, and deploy the Hospital Stay Length Prediction System. The system is designed to operate efficiently on standard computing devices without requiring high-end infrastructure, making it accessible and cost-effective for healthcare organizations.

- Processor: Intel i5 or equivalent
- Memory (RAM): Minimum 8 GB
- Storage: 256 GB HDD/SSD
- Display: Standard monitor with 1366×768 resolution or higher
- Optional: NVIDIA GPU with CUDA support (for faster model training, not mandatory).

# 5. SOFTWARE DESIGN

## 5.1 System Architecture

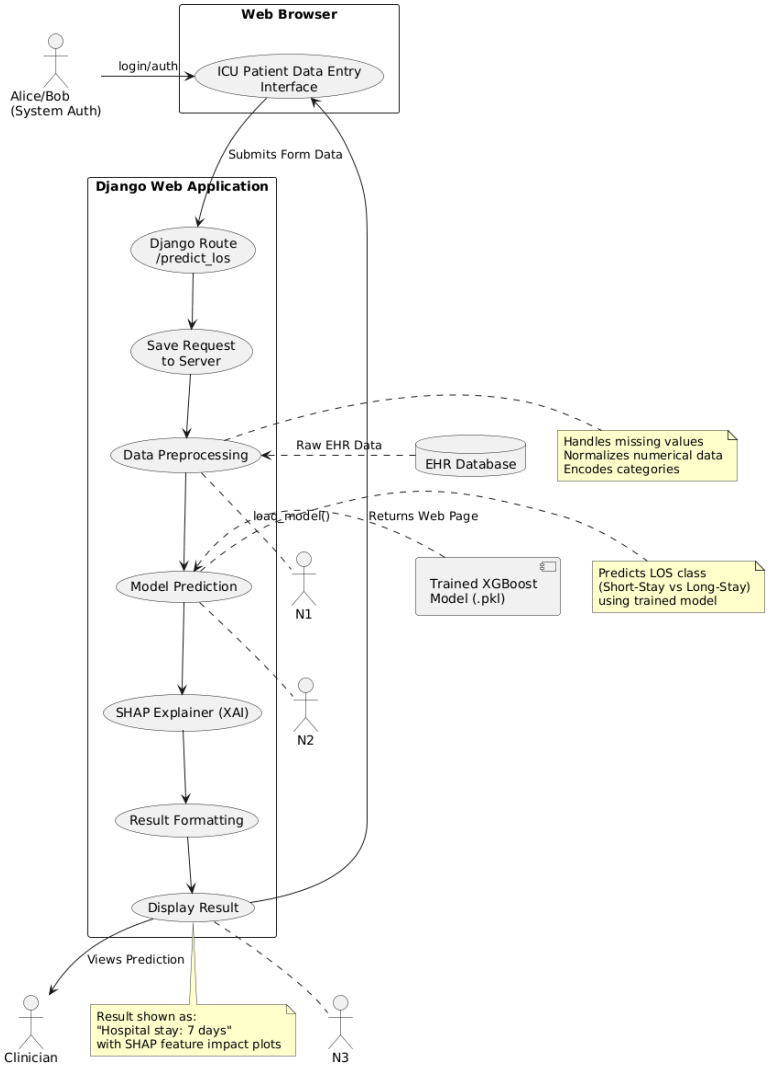


**Fig:5.1** System Architecture

The architecture of the Hospital Stay Length Prediction System is designed to provide an efficient, scalable, and user-friendly solution for predicting patient length of stay using machine learning and explainability techniques. The system follows a modular approach, integrating a web-based interface, backend processing using Flask, and trained machine learning models for prediction and interpretation.

At the front end, the system interacts with users through a web browser interface developed using HTML, CSS, and JavaScript. This interface provides a simple and intuitive platform where healthcare professionals can input patient details such as age, medical history, diagnosis, admission type, and other relevant clinical parameters. The design ensures ease of use, allowing doctors and hospital staff to interact with the system without requiring technical expertise. Once the data is entered, it is sent to the backend server for further processing.

The backend is implemented using the Flask web framework, which acts as the central controller of the system. Flask handles HTTP requests, manages routing, and coordinates communication between different system components. When a user submits patient data, the request is processed through a specific route (e.g., /predict). The input data is then validated and temporarily stored for processing

The preprocessed image is then fed into the trained CNN model, which is developed using TensorFlow/Keras. The model consists of multiple convolutional, pooling, and fully connected layers that extract hierarchical features from the image. These features include edges, textures, color patterns, and disease-specific visual characteristics. The model processes the input and generates a probability distribution across all predefined classes (14 classes in this system). The class with the highest probability is selected as the predicted output.

The preprocessed data is then fed into the trained machine learning model, which is developed using algorithms such as Linear Regression, Random Forest, or Gradient Boosting (e.g., XGBoost). The model has been trained on historical hospital data to learn patterns and relationships between patient attributes and their length of stay. It processes the input features and generates a predicted value representing the expected duration of hospitalization.

In addition to real-time prediction, the system includes an offline training pipeline. This component is responsible for building and improving the model over time. The training process involves data collection, cleaning, preprocessing, feature engineering, and model training. Various models are evaluated and optimized using techniques such as cross- validation and hyperparameter tuning to achieve the best performance.

To enhance transparency and trust, the system incorporates an explainability module using techniques such as SHAP (SHapley Additive Explanations) or LIME. This module analyzes the model's prediction and provides insights into how each input feature contributed to the final output. It highlights key factors influencing the predicted hospital stay, enabling healthcare professionals to understand and validate the results.

Once training is completed, the model is saved (e.g., in **.**pkl format) and stored for deployment. During runtime, the Flask application loads this trained model into memory, enabling fast and efficient predictions without retraining. This separation between training and inference ensures that the system remains responsive while maintaining accuracy and reliability.

Overall, the system architecture ensures scalability, efficiency, and ease of use, making it suitable for real-world healthcare environments while supporting data-driven decision- making. During runtime, the Flask backend loads the pre-trained model into memory, enabling fast and efficient predictions without the need for retraining. This separation between training and inference ensures that the system remains responsive and suitable for real-time applications. The system is also designed to handle multiple requests simultaneously, ensuring scalability and efficient performance in high-demand environments such as hospitals.

Furthermore, the architecture supports integration with external systems such as Clinical Information Systems (CIS) and Clinical Decision Support Systems (CDSS). This allows seamless incorporation of the prediction system into existing hospital workflows, enabling automated data retrieval and real-time decision support. Security and privacy measures are also considered, ensuring that sensitive patient data is handled securely through proper authentication, authorization, and data encryption techniques.

Overall, the proposed system architecture provides a robust, scalable, and interpretable framework for predicting hospital stay length. By combining machine learning with explainable AI and a user-friendly interface, the system enhances clinical decision-making, improves hospital resource management, and contributes to better patient care. The modular design ensures that the system can be easily extended in the future to include additional features such as real-time monitoring, multi-class LOS prediction, and integration with advanced deep learning models.

## 5.2 Dataflow Diagram

The proposed Data Flow Diagram (DFD) represents the flow of data within the Hospital Stay Length Prediction System, illustrating how user input is processed through different stages to generate the final prediction along with its explanation. The diagram clearly shows the interaction between the external entity (Doctor/Healthcare Staff), processing modules, and the trained machine learning model used for prediction.

The process begins with the Doctor or Healthcare Staff, who enters patient details through the system interface. These details may include patient age, gender, diagnosis, medical history, admission type, and other relevant clinical parameters. This input is first handled by the Data Input (1.0) process, where the system receives and temporarily stores the patient data. The output of this stage is Raw Patient Data, which is forwarded to the next processing stage.

In the Data Preprocessing (2.0) stage, the raw input data undergoes several preprocessing operations to make it suitable for model prediction. These operations include handling missing values, encoding categorical variables, normalizing numerical features, and structuring the data into a format compatible with the machine learning model. The result of this stage is Processed Feature Data, which represents clean and structured input ready for prediction.

The processed data is then passed to the Stay Length Prediction (3.0) module, which utilizes a trained machine learning model. The system loads the trained model from storage (e.g., Trained Model (.pkl File)), ensuring that previously learned patterns are applied during prediction. The model analyzes relationships between patient features and hospital stay duration, generating a Predicted Length of Stay based on learned patterns from historical data.

Next, the prediction output is forwarded to the Explainability Module (4.0). This module uses techniques such as SHAP or LIME to interpret the model's decision. It identifies the contribution of each feature to the predicted result and generates Feature Importance Scores or explanations, helping users understand why a particular prediction was made.

Finally, the Predicted Stay Duration along with Explanation is displayed to the Doctor or Healthcare Staff, completing the data flow cycle. This output supports better clinical decision-making, efficient resource allocation, and improved patient management.

Overall, this DFD demonstrates a structured and logical flow of data through the system, highlighting the transformation of raw patient data into meaningful predictions and interpretable insights. The modular design improves system clarity, maintainability, and scalability, making it suitable for real-world healthcare applications using machine learning and explainable AI techniques.

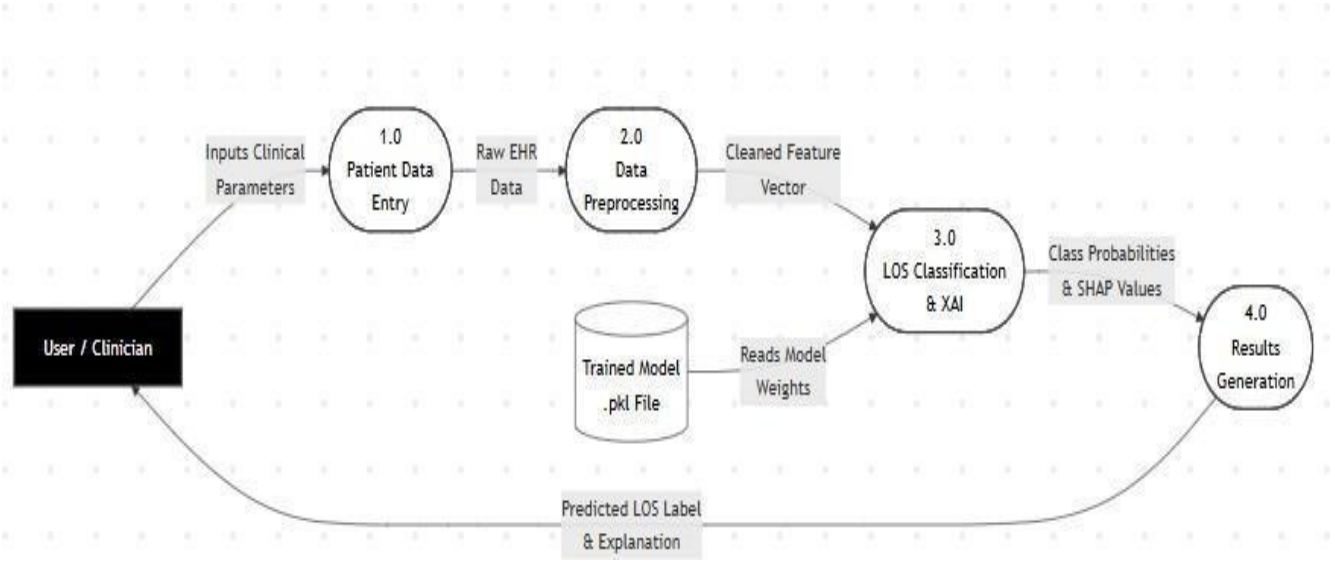


**Fig: 5.2** Dataflow Diagram

## 5.3 UML Diagrams

UML (Unified Modeling Language) is a standardized modeling language used for specifying, visualizing, constructing, and documenting the components and behavior of software systems. It provides a common language for developers, designers, and stakeholders to understand system architecture and functionality. UML was developed by the Object Management Group (OMG) and became a standard in 1997. It plays a crucial role in object-oriented analysis and design by offering a clear representation of system structure and interactions.

UML diagrams help in simplifying complex systems by breaking them down into visual representations. These diagrams are widely used during different phases of software development, including requirement analysis, system design, implementation, and testing. By using UML, developers can effectively communicate system design and ensure that all components work together as intended.

UML diagrams are broadly classified into two main categories: Behavioral diagrams and Structural diagrams. Behavioral diagrams focus on the dynamic aspects of the system, such as interactions, workflows, and processes. Structural diagrams, on the other hand, represent the static components of the system and their relationships. In the context of the Plant Disease Detection System, UML diagrams help in visualizing how users interact with the system, how data flows through different modules, and how various components such as the web interface, backend, and CNN model are interconnected.

**Goals of UML:**

- Provide an expressive visual modeling language for developing and exchanging meaningful models.
- Establish a formal basis for understanding the modeling language.
- Encourage the growth of object-oriented tools.
- Integrate best practices into system development.
- Improve communication among developers, designers, and stakeholders.
- Simplify system complexity through visual representation.
- Support system documentation and maintenance.

**Types of UML Diagrams:**

1. Use Case Diagram:

2. Sequence Diagram:

3. Activity Diagram:

4. Class Diagram:

### 5.3.1 Use Case Diagram

The Use Case Diagram for the Hospital Stay Length Prediction System serves as a comprehensive blueprint of the functional requirements and clearly defines the roles of internal and external participants. The system is enclosed within a structural boundary that represents the core software processes, separating them from the human actors who interact with the application. By organizing the diagram into two distinct actor-based flows, the design differentiates between the user-facing functionalities and the technical backend processes managed by the analyst.

The primary flow on the left side of the diagram focuses on the Doctor/Healthcare Staff, who interact with the system through a user-friendly web interface. The user's interaction begins with accessing the system, followed by entering patient details such as age, diagnosis, medical history, and other clinical parameters. Once the data is submitted, the system processes the input and generates the key outputs: the predicted hospital stay length and an explanation of the factors influencing the prediction. This interaction is designed to be simple and intuitive, ensuring that the complexity of machine learning models remains hidden from the user while still providing meaningful and actionable insights.

On the right side of the diagram, the Analyst/Data Scientist is responsible for the development, training, and maintenance of the machine learning model. This flow includes several interconnected processes using "include" relationships to represent dependencies. Model training depends on data preprocessing, which in turn depends on data collection from hospital records. The analyst is also responsible for feature engineering, model selection (such as regression or ensemble models), and performance evaluation. Additionally, the analyst integrates explainability techniques such as SHAP or LIME to ensure that predictions are interpretable and trustworthy.

The rectangle in the diagram represents the System Boundary, which encapsulates all internal functionalities such as data input handling, preprocessing, prediction generation, and explanation generation. Use cases like "Access Web Interface," "Enter Patient Data," "Predict Length of Stay," and "View Explanation" are contained within this boundary, indicating that the system operates as an independent unit once deployed. The system, typically implemented using a Flask-based architecture, autonomously processes inputs and produces outputs without requiring manual intervention from the analyst during runtime.

Overall, the diagram illustrates a clear and efficient workflow, starting from patient data input to prediction and explanation delivery. It highlights the separation of concerns between end users and developers while ensuring that the system remains scalable, maintainable, and user-friendly for real-world healthcare applications.

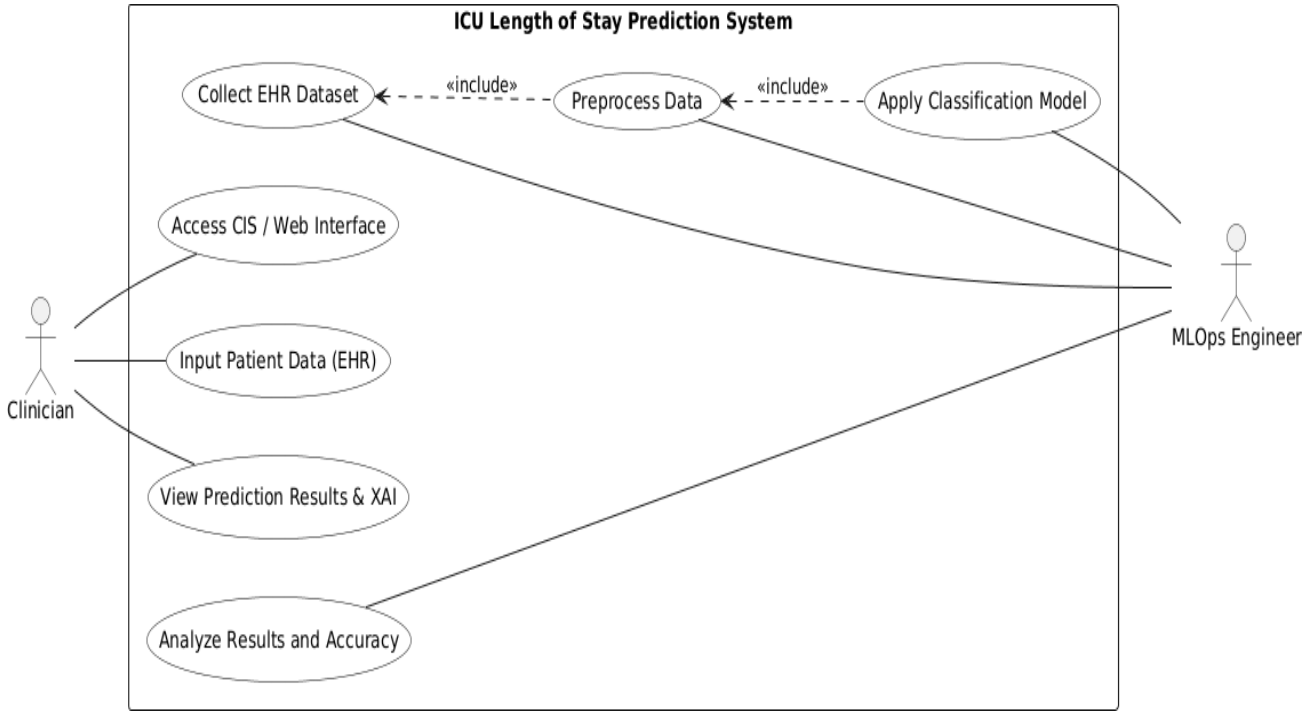


**Fig 5.3.1** Use Case Diagram

### 5.3.2. Sequence Diagram

The sequence diagram represents the dynamic interaction between the major components of the Hospital Stay Length Prediction System, namely the Doctor/Healthcare Staff, FlaskApp, DataProcessor, Predictor, and Explainer modules. It illustrates the step-by-step communication and the order in which operations are performed during the prediction process. The process begins when the doctor or healthcare staff enters patient details through the web interface. This action triggers an HTTP request that is sent to the Flask application. The FlaskApp acts as the central controller, receiving the request and managing the overall workflow of the system.

Once the request is received, the FlaskApp initiates the data handling process by forwarding the input to the DataProcessor module. This module is responsible for validating the input data, checking for missing or incorrect values, and preparing the data for further processing. It ensures that only valid and properly formatted data is passed to the next stage, thereby reducing errors during prediction. After validation, the DataProcessor performs preprocessing operations such as encoding categorical variables, normalizing numerical values, and structuring the data into a suitable format. The processed data is then returned to the FlaskApp.

Once the prediction is generated, the FlaskApp forwards the result to the Explainer module. This module is responsible for interpreting the model's output using explainability techniques such as SHAP or LIME. It analyzes the contribution of each input feature and produces an explanation that highlights the key factors influencing the predicted hospital stay duration.

Finally, the FlaskApp sends the formatted response back to the user interface, where the predicted hospital stay length and explanation are displayed clearly to the doctor or healthcare staff. This completes the sequence of operations for a single prediction request.

The sequence diagram also highlights the synchronous nature of communication between components, where each step depends on the successful completion of the previous one. This ensures proper coordination and efficient data flow throughout the system.

Furthermore, this diagram helps identify potential delays, bottlenecks, or failure points within the system. For instance, errors in data entry, preprocessing, prediction, or explanation generation can be managed effectively through validation and exception handling mechanisms at each stage. Overall, the sequence diagram provides a clear understanding of the runtime behavior of the system, demonstrating how different components collaborate to deliver accurate and explainable predictions in real time.

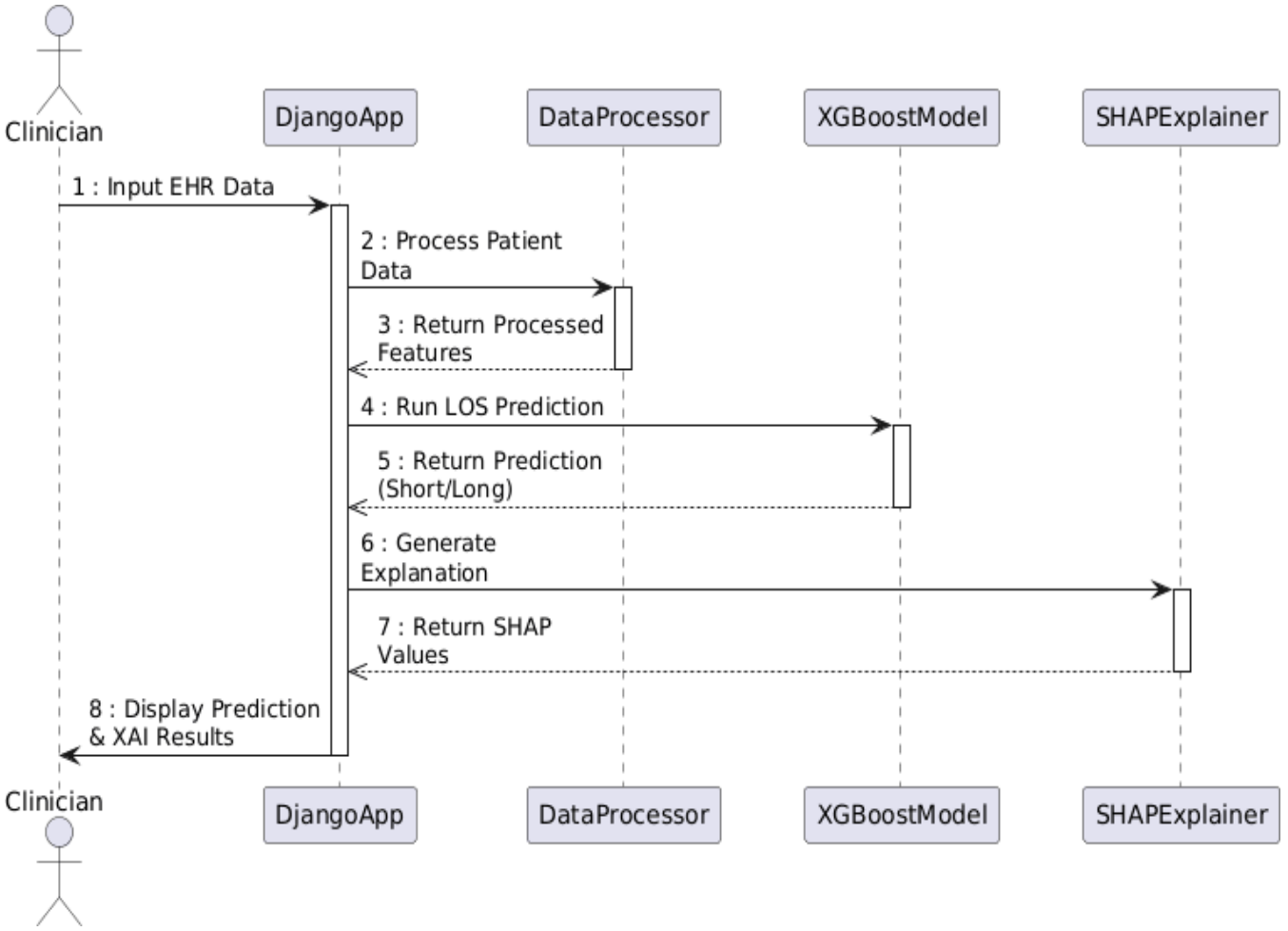


**Fig 5.3.2** Sequence Diagram

**List of actions**

- **Doctor / Healthcare Staff:**

Initiates the process by accessing the web application through a browser and entering patient details required for prediction. The user inputs information such as age, gender, diagnosis, medical history, and other relevant clinical parameters. The system is designed to make this step simple and intuitive, requiring minimal technical knowledge while ensuring accurate data entry.

- **Web Application:**

Receives the entered patient data and performs initial validation to ensure all required fields are correctly filled and in the proper format. The data is then temporarily stored on the server. The application preprocesses the input by handling missing values, encoding categorical variables, and normalizing numerical features. After preprocessing, the structured data is forwarded to the machine learning model for prediction. The web application also manages communication between different modules and ensures smooth data flow throughout the system.

- **Machine Learning Model:**

Processes the preprocessed patient data using a trained model such as Linear Regression, Random Forest, or Gradient Boosting. The model analyzes relationships between input features and hospital stay duration based on patterns learned from historical data. It then generates a predicted value representing the expected length of hospital stay

- **Output Display:**

Receives the prediction and explanation results and converts them into a clear, human- readable format. The system displays the predicted hospital stay duration along with key contributing factors on the web interface. The output is presented in a user-friendly manner, enabling doctors and hospital staff to make informed decisions regarding patient care and resource management.

### 5.3.3 Activity Diagram

The activity diagram represents the complete workflow of the Hospital Stay Length Prediction System, illustrating how the system processes user input and generates the final prediction along with its explanation. It provides a clear visualization of the sequence of operations, decision points, and data flow within the system. The diagram begins with the initial node, where the doctor or healthcare staff interacts with the web-based application by entering patient details. This step acts as the entry point of the system and triggers the subsequent processing stages.

After the data is entered, the system performs a validation check to ensure that the input meets the required criteria. This decision node plays a crucial role in maintaining system reliability by verifying whether all mandatory fields are filled, data types are correct, and values are within acceptable ranges. If the input data does not satisfy these conditions, the system follows the negative path and displays an appropriate error message to the user. This step prevents invalid or incomplete data from being processed further, thereby avoiding incorrect predictions and unnecessary computation.

If the input data is valid, the system proceeds to the data preprocessing stage. In this phase, several important transformations are applied to prepare the data for the machine learning model. These operations include handling missing values, encoding categorical variables, normalizing or scaling numerical features, and structuring the data into a format suitable for model input. This preprocessing step ensures consistency across all inputs and improves the efficiency and accuracy of the model during prediction.

Following preprocessing, the system advances to the model prediction stage, where the trained machine learning model is used to analyze the patient data. The model identifies patterns and relationships between input features and hospital stay duration based on historical data. It then generates a predicted value representing the expected length of hospital stay. This stage forms the core functionality of the system and is responsible for producing accurate and reliable predictions.

Once both the prediction and explanation are ready, the system moves to the result display stage, where the output is presented to the user through the web interface. The predicted hospital stay duration, along with key contributing factors, is displayed in a clear and user- friendly format.

Finally, the process reaches the end node, signifying the completion of the workflow. Overall, the activity diagram effectively demonstrates the logical flow of operations, including data validation, decision-making, preprocessing, prediction, explainability, and result generation. It highlights the system's ability to handle both valid and invalid inputs efficiently while ensuring a smooth, transparent, and reliable user experience throughout the prediction process.

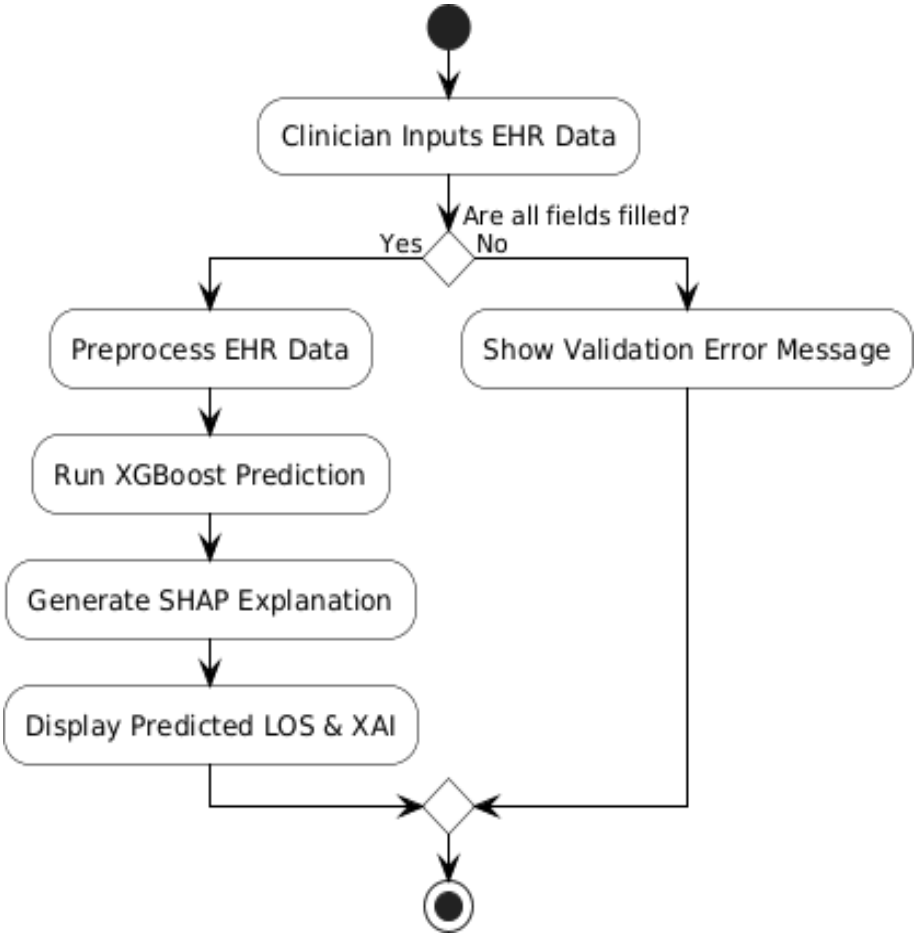


**Fig 5.3.3** Activity Diagram

### 5.3.4. Class Diagram

The class diagram illustrates the structural architecture of the Hospital Stay Length Prediction System by representing the main classes, their attributes, methods, and the relationships between them. It provides a clear understanding of how the system is organized internally and how different components collaborate to process patient data and generate predictions along with explanations. The diagram emphasizes a modular design approach, where each class is assigned a specific responsibility, ensuring better maintainability, scalability, and clarity in the overall system design.

At the core of the system is the FlaskApp class, which functions as the central controller and coordinates all major operations. This class is responsible for managing user interactions, handling incoming patient data requests, and integrating the machine learning model into the application workflow. It contains key attributes such as the trained model (e.g., a serialized .pkl file) and configuration settings. The FlaskApp class also includes essential methods like handle_request(), preprocess_data(), predict_stay_length(), and display_result(), which are responsible for processing input data and returning results. By combining control logic with model integration, this class ensures smooth communication between the user interface and the prediction engine.

The DataProcessor class is designed to handle all operations related to input data preprocessing. It includes attributes such as raw_data and processed_data and provides methods like clean_data(), encode_features(), normalize_data(), and transform_input(). This class plays a crucial role in preparing the patient data for model prediction by ensuring that the data is accurate, consistent, and in the correct format. By isolating preprocessing functionality into a separate class, the system maintains a clear separation between data handling and prediction logic.

The Predictor class is responsible for executing the core functionality of the system, which is predicting hospital stay length using the trained machine learning model. It defines methods such as load_model() and predict(data), which are used to load the trained model and generate predictions based on processed input data. This class focuses entirely on the prediction pipeline, ensuring efficient and accurate inference.

The relationships between the classes demonstrate how the system components interact to complete the overall workflow. The FlaskApp class acts as the main coordinator, interacting with the UserInterface to receive patient data, then communicating with the DataProcessor class for preprocessing. After preprocessing, it sends the processed data to the Predictor class for prediction and then forwards the result to the Explainer class for interpretation. Finally, the results are sent back to the UserInterface for display.

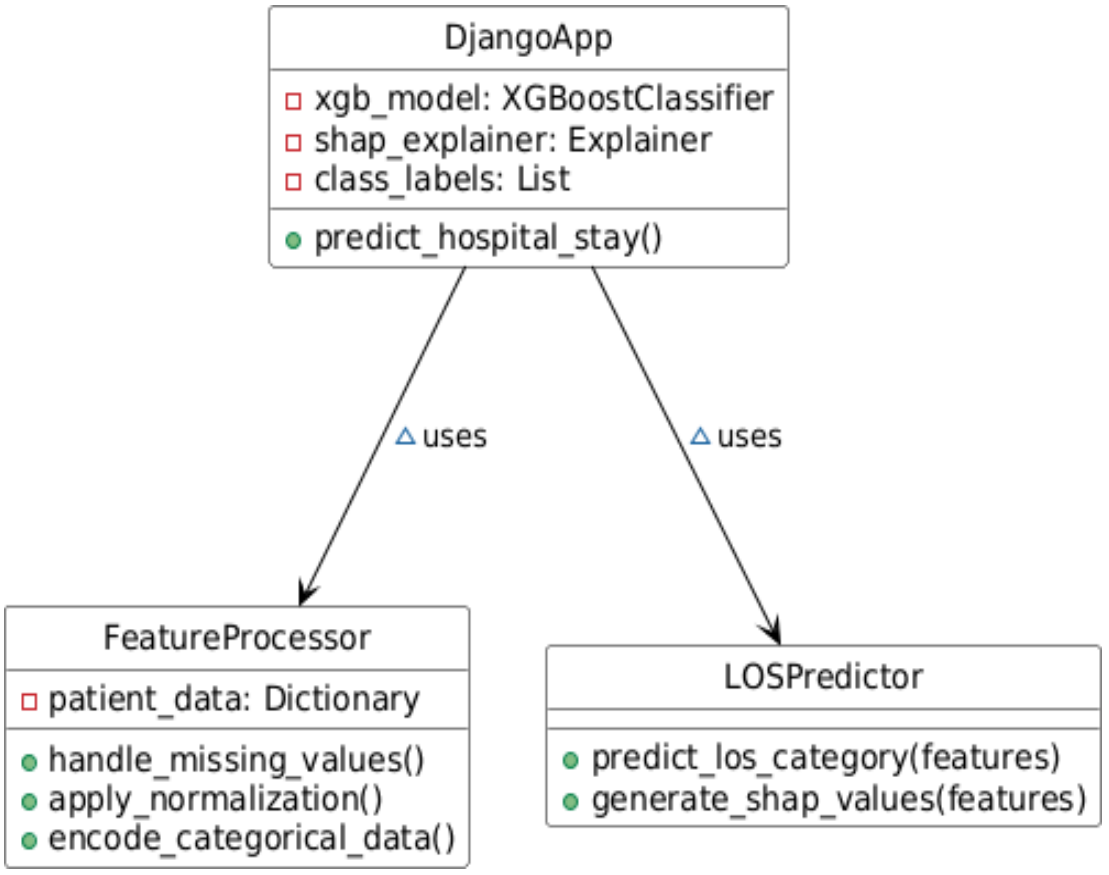


**Fig 5.3.4** Class Diagram

# 6. CODING AND IMPLEMENTATION

## 6.1 Source Code

**Untitiled.ipynb:**

```
# Import required libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import os
import warnings
warnings.filterwarnings('ignore')

# Machine Learning Libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression, Ridge
# Check for duplicate records
print("\nDuplicate rows:", data.duplicated().sum())

# Remove duplicates if any
data.drop_duplicates(inplace=True)
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor, AdaBoostRegressor,
    VotingRegressor
import xgboost as xgb
import lightgbm as lgb
from catboost import CatBoostRegressor

import joblib
# Load dataset
data = pd.read_csv("data.csv")
# Display statistical summary of dataset
print("\nDataset Summary:")
```

```
print(data.describe())
# Display column names
print("\nColumns in dataset:", data.columns.tolist())

# Display basic information
print(data.head())
print(data.info())
print(data.isnull().sum())
  data.fillna(data.mean(), inplace=True)
label_encoder = LabelEncoder()
data['Location'] = label_encoder.fit_transform(data['Location'])
            data['Hospital_Stay'] = data['Hospital_Stay'].round().astype(int)
data['MRI_Units'] = data['MRI_Units'].round().astype(int)
data['CT_Scanners'] = data['CT_Scanners'].round().astype(int)
data['Hospital_Beds'] = data['Hospital_Beds'].round().astype(int)
plt.figure(figsize=(10,6))
sns.heatmap(data.corr(), annot=True)
plt.title("Correlation Matrix")
plt.show()
X = data.drop(columns=["Hospital_Stay"])
y = data["Hospital_Stay"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
  scaler = StandardScaler()
 X_train = scaler.fit_transform(X_train)
 X_test = scaler.transform(X_test)
 joblib.dump(scaler, "scaler.pkl")
 lr_model = LinearRegression()
 lr_model.fit(X_train, y_train)
 y_pred = lr_model.predict(X_test)
 print("Linear Regression")
 print("MSE:", mean_squared_error(y_test, y_pred))
print("R2:", r2_score(y_test, y_pred))
svr_model = SVR(kernel='rbf')
svr_model.fit(X_train, y_train)
y_pred = svr_model.predict(X_test)
print("SVR")
print("MSE:", mean_squared_error(y_test, y_pred))
print("R2:", r2_score(y_test, y_pred))
        dt_model = DecisionTreeRegressor(random_state=42)
```

```
dt_model.fit(X_train, y_train)
        rf_model = RandomForestRegressor(n_estimators=100)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
print("Random Forest")
print("MSE:", mean_squared_error(y_test, y_pred))
print("R2:", r2_score(y_test, y_pred))
        xgb_model = xgb.XGBRegressor()
xgb_model.fit(X_train, y_train)
y_pred = cat_model.predict(X_test)
print("CatBoost")
print("MSE:", mean_squared_error(y_test, y_pred))
print("R2:", r2_score(y_test, y_pred))
et_model = ExtraTreesRegressor()
et_model.fit(X_train, y_train)
y_pred = et_model.predict(X_test)
print("MSE:", mean_squared_error(y_test, y_pred))
print("R2:", r2_score(y_test, y_pred))
voting_model = VotingRegressor([
    ('rf', rf_model),
    ('dt', dt_model),
    ('et', et_model)
])
# Store model results
results = {
    "Linear": mean_squared_error(y_test, lr_model.predict(X_test)),
    "SVR": mean_squared_error(y_test, svr_model.predict(X_test)),
    "Random Forest": mean_squared_error(y_test, rf_model.predict(X_test)),
    "XGBoost": mean_squared_error(y_test, xgb_model.predict(X_test))
}
print("\nModel Comparison (MSE):")
for model, score in results.items():
    print(model, ":", score)
voting_model.fit(X_train, y_train)
y_pred = voting_model.predict(X_test)
print("MSE:", mean_squared_error(y_test, y_pred))
print("R2:", r2_score(y_test, y_pred))
        # Save best model
joblib.dump(xgb_model, "model.pkl")
```

## model.ipynb:

```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold,
RandomizedSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression,  LinearRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,
VotingClassifier, AdaBoostClassifier, ExtraTreesClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from imblearn.over_sampling import SMOTE
import xgboost as xgb
import lightgbm as lgb
import catboost
import seaborn as sns
import matplotlib.pyplot as plt
import joblib
print("\nStatistical Summary:")
print(df.describe())
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
# Encoding categorical features
label_encoder = LabelEncoder()
# Apply label encoding to ordinal categorical features
df['Location'] = label_encoder.fit_transform(df['Location'])
df['Hospital_Stay'] = df['Hospital_Stay'].round().astype(int)
df['MRI_Units'] = df['MRI_Units'].round().astype(int)
df['CT_Scanners'] = df['CT_Scanners'].round().astype(int)
df['Hospital_Beds'] = df['Hospital_Beds'].round().astype(int)


# Display encoded DataFrame
print(df.head())
```

```
# Save the label encoder
joblib.dump(label_encoder, 'label_encoder.pkl')
df.head(3)
df.to_csv('encoded_data.csv', index=False)
df.dtypes
corr = df.corr()
corr

import seaborn as sns
import matplotlib.pyplot as plt


# Example
plt figure(figsize=(8, 6))  # Adjust width and height as needed sns.heatmap(corr, annot=True)
plt.title('Correlation')
plt.show()
```

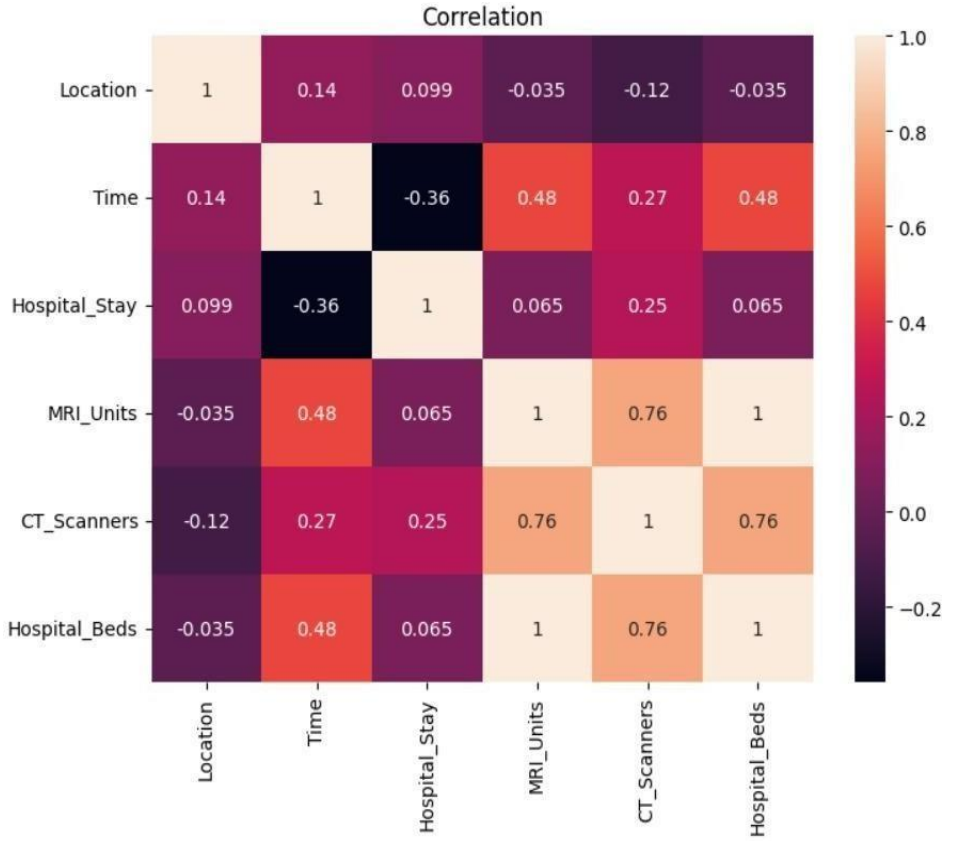


**Fig 6.1: Feature Correlation Analysis for Hospital Stay Prediction**

```
# Check missing values
print("\nMissing values per column:")
print(df.isnull().sum())


# Check duplicate records
print("\nDuplicate rows:", df.duplicated().sum())

# Remove duplicates
df.drop_duplicates(inplace=True)

# Reset index after removing duplicates
df.reset_index(drop=True, inplace=True)

# Verify dataset shape after cleaning
print("\nDataset shape after cleaning:", df.shape)


from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Initialize and train Linear Regression model
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Predictions and evaluation
y_pred = lr_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Decision Tree Regressor:")
print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")
```

```
X = df.drop(columns=["Hospital_Stay"])  # All columns except target
y = df["Hospital_Stay"]  # Target column
# from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=0)

# Scale the features
scaler =
StandardScaler()
X_train =
scaler.fit_transform(X_train)  X_test
= scaler.transform(X_test)

joblib.dump(scaler, 'scaler.pkl')

 from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,
r2_score # Initialize and train Linear Regression model
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Predictions and evaluation
y_pred =
lr_model.predict(X_test)
mse = mean_squared_error(y_test,
y_pred) r2 = r2_score(y_test, y_pred)

print("Linear Regression:")
print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")

 from sklearn.svm import
SVR  # Initialize and train
SVR model svr_model =
SVR(kernel='rbf')
svr_model.fit(X_train,y_train
```

```
)

# Predictions and evaluation
y_pred =
svr_model.predict(X_test)
mse = mean_squared_error(y_test,
y_pred) r2 = r2_score(y_test, y_pred)
from sklearn.tree import
DecisionTreeRegressor

dt_model =
DecisionTreeRegressor(random_state=42
)
dt_model.fit(X_train, y_train)

y_pred = dt_model.predict(X_test)
mse = mean_squared_error(y_test,
y_pred)
r2 = r2_score(y_test, y_pred)

print("Decision Tree Regressor:")
print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")
from sklearn.neighbors import
KNeighborsRegressor

knn_model =
KNeighborsRegressor(n_neighbors=5)
# Initialize Random Forest Regressor
rf_model=RandomForestRegressor(n_esti
mators=100, random_state=42)

# Train the model using training data
rf_model.fit(X_train, y_train)

# Make predictions on test data
```

```
y_pred = rf_model.predict(X_test)

# Calculate Mean Squared Error to
measure prediction error
mse = mean_squared_error(y_test,
y_pred)

# Calculate R² score to evaluate model
performance
r2 = r2_score(y_test, y_pred)

# Display results for Random Forest
model
print("Random Forest Regressor:")
print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")print(f"R^2
Score: {r2}")

from sklearn.ensemble import
RandomForestRegressor
rf_model =
RandomForestRegressor(n_estimators=1
00, random_state=42)
rf_model.fit(X_train, y_train)

y_pred = rf_model.predict(X_test)
mse = mean_squared_error(y_test,
y_pred)
r2 = r2_score(y_test, y_pred)

print("Random Forest Regressor:")
print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")
print("Support Vector Regressor
(SVR):")
```

```
  # Displaying Mean Squared Error value
  print(f"Mean Squared Error: {mse}")

  # Displaying R² score value
  print(f"R^2 Score: {r2}")


 from sklearn.tree import DecisionTreeRegressor
 # Initialize and train Decision Tree Regressor model
 dt_model = DecisionTreeRegressor(random_state=42)
 dt_model.fit(X_train, y_train)

 # Predictions and evaluation
 y_pred =
 dt_model.predict(X_test)
 mse = mean_squared_error(y_test, y_pred)
 r2 = r2_score(y_test, y_pred)

print("Decision Tree Regressor:")
print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")
from sklearn.neighbors import KNeighborsRegressor
# Initialize and train KNN Regressor model
knn_model = KNeighborsRegressor(n_neighbors=5)
knn_model.fit(X_train, y_train)

# Predictions and evaluation
y_pred = knn_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("K-Nearest Neighbors Regressor:")
print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score
# Initialize and train Ridge Regression model ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)
```

```
# Predictions and evaluation
y_pred = ridge_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Ridge Regression:")
print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")
ada_model = AdaBoostRegressor(base_model, n_estimators=50, random_state=42)
ada_model.fit(X_train, y_train)
# Predictions and evaluation
y_pred = ada_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("AdaBoost Regressor:")
print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")
from catboost import CatBoostRegressor
from sklearn.metrics import mean_squared_error, r2_score
# Initialize and train CatBoost Regressor model
catboost_model = CatBoostRegressor(iterations=100, depth=5, learning_rate=0.1,
loss_function='RMSE', cat_features=[], verbose=0)
catboost_model.fit(X_train,  y_train)

# Predictions and evaluation
y_pred = catboost_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("CatBoost Regressor:")
print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")

import lightgbm as lgb
from sklearn.metrics import mean_squared_error, r2_score

# Initialize and train LightGBM model
lgb_model = lgb.LGBMRegressor(n_estimators=100, learning_rate=0.05)
```

```
lgb_model.fit(X_train, y_train)

# Predictions and evaluation
y_pred = lgb_model.predict(X_test)
mse = mean_squared_error(y_test,
y_pred) r2 = r2_score(y_test, y_pred)

print("LightGBM Regressor:")
print(f"Mean Squared Error:
{mse}") print(f"R^2 Score: {r2}")
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Initialize and train ExtraTrees Regressor model
et_model =
ExtraTreesRegressor(n_estimators=100)
et_model.fit(X_train, y_train)

# Predictions and evaluation
y_pred =
et_model.predict(X_test)
mse = mean_squared_error(y_test,
y_pred) r2 = r2_score(y_test, y_pred)

print("Extra Trees Regressor:")
print(f"Mean Squared Error:
{mse}") print(f"R^2 Score: {r2}")
from sklearn.ensemble import
VotingRegressor from sklearn.linear_model
import LinearRegression from sklearn.tree
import DecisionTreeRegressor
from sklearn.ensemble import
RandomForestRegressor from sklearn.metrics import
mean_squared_error, r2_score from
sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
# Initialize individual models
dt_model = DecisionTreeRegressor(random_state=42)
rf_model = RandomForestRegressor(n_estimators=100,
random_state=42) et_model =
ExtraTreesRegressor(n_estimators=100, random_state=42)
catboost_model = CatBoostRegressor(iterations=100, depth=5, learning_rate=0.1,
loss_function='RMSE', cat_features=[], verbose=0)
# Initialize Voting Regressor
 voting_regressor = VotingRegressor(estimators=[

 ('dt', dt_model),
 ('rf', rf_model),
 ('et', et_model),
 ('catboost', catboost_model)
 ])

 # Train the Voting Regressor
 voting_regressor.fit(X_train, y_train)

 # Predictions and evaluation
 y_pred =
 voting_regressor.predict(X_test) mse
 = mean_squared_error(y_test,
 y_pred) r2 = r2_score(y_test,
 y_pred)

 print("Voting Regressor:")
 print(f"Mean Squared Error:
 {mse}") print(f"R^2 Score:
 {r2}")
```

## Backend:

**manage.py**

```
#!/usr/bin/env python
import os
import sys

if __name__ == '__main__':
    os.environ.setdefault('DJANGO_SETTINGS_MODULE','Hospital_Stay.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)
```

## Frontend:

### home.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Predicting Hospital Stay Length Using Explainable Machine Learning</title>
  {% load static %}
  <style>
    body {
      font-family: Arial, sans-serif;
      background-image: url('{% static "images/background.jpg" %}'); /* Ensure correct path */
      background-size: cover;
      background-position: center;
      height: 100vh;
      margin: 0;
      display: flex;
      justify-content: center;
      align-items: center;
      color: rgb(19, 18, 18);
      text-align: center;
    }
    h1 {
      font-size: 5rem; /* Increase the font size for the heading */
      margin-bottom: 20px;
    }
    .buttons {
      display: flex;
      justify-content: center;
      gap: 100px;
      }
```

```
        .btn {
            background-color: #4CAF50; /* Green background for buttons */
            color: white;
            padding: 15px 32px;
            text-align: center;
            text-decoration: none;
            display: inline-block;
            font-size: 1.2rem; /* Larger text size for the buttons */
            border-radius: 8px;
            margin: 10px;
            transition: background-color 0.3s ease;
        }

        .btn:hover {
            background-color: #45a049; /* Darker green on hover */
        }

        .container {
            text-align: center;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Predicting Hospital Stay Length Using Explainable Machine Learning</h1>
        <div class="buttons">
            <a href="{% url 'login' %}" class="btn">Login</a>
            <a href="{% url 'register' %}" class="btn">Register</a>
        </div>
    </div>
</body>
</html>
```

## 6.2 Implementation:

### 6.2.1 Front-End Implementation:

The front-end of the Hospital Stay Length Prediction System is designed to provide a seamless, interactive, and user-friendly experience through a clean and responsive interface. The primary objective of the front-end is to enable healthcare professionals to easily input patient details and obtain accurate predictions along with explanations, without requiring any technical expertise in machine learning. The user interface is developed using HTML, CSS, Bootstrap, and basic JavaScript, ensuring cross-browser compatibility and responsiveness across multiple devices such as desktops, tablets, and hospital workstations. The layout is structured to guide users step-by-step through the data entry and prediction process, minimizing errors and improving usability.

The main page prominently displays the system title Hospital Stay Length Prediction System, followed by a well-organized input form. Users can enter patient details such as age, gender, diagnosis, medical history, admission type, and other relevant clinical parameters through form fields like text boxes, dropdown menus, and selection inputs. The interface ensures that all inputs are clearly labeled and easy to understand. A clearly visible Predict button allows users to initiate the prediction process.

To enhance user experience, the interface may include visual feedback mechanisms such as loading indicators or progress messages while the system processes the input data. This reassures users that the system is actively working on their request. Additionally, client-side validation ensures that incorrect or incomplete inputs, such as missing fields or invalid data types, are handled gracefully with appropriate error messages, thereby improving data accuracy and system reliability.

Once the prediction is completed, the result is displayed on the same page in a clear and structured format. The system highlights the predicted hospital stay duration along with key contributing factors derived from explainable machine learning techniques. The explanation helps users understand why a particular prediction was made, increasing transparency and trust in the system. The output is designed to be simple and easy to interpret, avoiding unnecessary technical complexity.

### 6.2.2 Backend Implementation:

The backend of the Hospital Stay Length Prediction System is implemented using the Flask framework, which acts as the central processing unit of the application. It is responsible for handling user requests, processing patient data, interacting with the machine learning model, and returning prediction results along with explanations.

The backend follows a structured and modular architecture, where each functionality is divided into separate components such as request handling, data preprocessing, model inference, explainability processing, and response generation. This modular design improves maintainability, scalability, and code reusability.

When a user submits patient data through the web interface, the backend receives it via an HTTP POST request. The system performs validation checks to ensure that all required fields are present and that the data is in the correct format. The input data is then temporarily stored in memory for processing. The backend also implements error handling mechanisms to manage unexpected scenarios such as missing values, invalid inputs, or processing failures. Logging mechanisms are used to record system activities and errors, which helps in debugging and performance monitoring.

### 6.2.3 Model Integration and Processing Workflow

The machine learning model is tightly integrated into the backend as a dedicated prediction module. The workflow begins when the backend receives patient data from the front end and initiates a preprocessing pipeline. This pipeline includes handling missing values, encoding categorical variables, normalizing or scaling numerical features, and transforming the data into a structured format suitable for model input.

The trained model (such as Random Forest, Linear Regression, or Gradient Boosting) processes the input features and learns patterns from historical hospital data. It analyzes relationships between patient attributes and hospital stay duration to generate a predicted value representing the expected length of stay.

To ensure robustness, the system maintains consistency between training and inference pipelines. Proper preprocessing and feature handling ensure reliable predictions across different inputs. The integration is optimized for fast inference, enabling near real-time predictions in a clinical environment.

### 6.2.4 Deployment and Reliability

The system is deployed in a secure and stable server environment, ensuring reliable performance and accessibility. The Flask application acts as the backend server, handling incoming requests and coordinating all system operations

Performance optimization techniques are applied to ensure quick response times and efficient resource utilization. Since the system processes structured data rather than images, the computational overhead is relatively low, allowing faster predictions even on standard hardware.

The system undergoes extensive testing, including functional testing, performance testing, and user acceptance testing, to ensure reliability under different conditions. The architecture supports deployment on cloud platforms, enabling remote access and large- scale usage. Future enhancements may include containerization (e.g., Docker) and cloud-based deployment for improved scalability and availability.

### 6.2.5 Conclusion

The developed Hospital Stay Length Prediction System successfully integrates a user-friendly web interface, a robust backend framework, and an efficient machine learning model into a unified platform. The system provides an effective solution for predicting patient hospital stay duration, enabling better resource management and improved healthcare planning.

The system demonstrates strong performance in analyzing patient data and generating reliable predictions. The inclusion of explainable machine learning enhances transparency, allowing healthcare professionals to understand the reasoning behind predictions and build trust in the system.

Furthermore, the project highlights the potential of machine learning and explainable AI in transforming healthcare operations. By providing accurate predictions and actionable insights, it contributes to improved patient care, optimized hospital resources, and efficient healthcare delivery.

# 7. SYSTEM TESTING

System testing is a critical phase that ensures the developed Hospital Stay Length Prediction system operates accurately, reliably, and efficiently under real-world healthcare conditions. The primary objective of system testing is to identify potential defects, validate overall system functionality, and ensure that both functional and non-functional requirements are fully satisfied. In addition to verifying correctness, system testing evaluates the system's robustness, interpretability, scalability, and performance when exposed to real clinical data and varying usage scenarios. It serves as the final validation stage before deployment, ensuring seamless integration of all system components.

In this project, system testing focuses on validating key components, including the front-end user interface for patient data input, backend processing modules, data preprocessing pipeline, and the explainable machine learning model used for prediction. Testing ensures that patient data is correctly captured from the interface, preprocessed appropriately (handling missing values, normalization, and encoding), passed through the trained model, and converted into accurate predictions of hospital stay duration. Additionally, the explainability component is thoroughly tested to ensure that model outputs are accompanied by meaningful explanations, such as feature importance or contribution scores, enabling transparency and trust in predictions. Special attention is given to model inference accuracy, reliability of API communication, and system stability during continuous use.

System testing was performed using diverse healthcare datasets containing patient records with varying medical conditions, demographics, and treatment histories to ensure robustness and reliability. The system was evaluated under different real-world scenarios, including incomplete data entries, noisy or inconsistent inputs, and high-frequency prediction requests. Special emphasis was placed on prediction accuracy and interpretability, ensuring that the model not only produces reliable estimates of hospital stay length but also provides clear and understandable explanations for its predictions. Performance metrics such as response time, system throughput, and stability under repeated usage were carefully analyzed. The system demonstrated consistent and reliable performance, confirming its suitability for practical deployment in healthcare environments where accurate and explainable decision support is essential.

## 7.1 Types of System Testing

### 7.1.1 Unit Testing

Unit testing was performed to validate individual components of the Hospital Stay Length Prediction system independently. Each backend function, data preprocessing module, machine learning model interaction component, and explainability module was tested using controlled inputs to verify expected outputs. This approach ensures that every small unit of the system performs its intended function correctly without relying on other modules. By isolating components, issues can be identified and resolved early, reducing complexity during later stages of development.

Key focus areas included:

- data preprocessing operations such as handling missing values, normalization, and encoding of categorical variables
- correct transformation of patient data into model-compatible formats
- machine learning model loading and prediction execution
- generation of explainability outputs (such as feature importance or contribution scores)

Unit testing ensured that each functional unit operated correctly in isolation, enabling early detection and correction of errors before system integration. It also helped validate edge cases such as incomplete patient records, invalid data types, out-of-range values, and missing inputs, ensuring the system remains robust and reliable.

### 7.1.2 Integration Testing

Integration testing was conducted to evaluate whether individual modules of the Hospital Stay Length Prediction system function correctly when combined into a complete workflow. This phase is crucial because, although individual components may perform accurately in isolation, issues can arise when they interact with each other.

Modules tested in combination included:

- front-end patient data input interface with backend API processing
- data preprocessing pipeline integrated with machine learning model inference
- mapping of prediction results and explanations to the frontend display
- interaction between backend routes and model loading/execution module

This stage verified smooth data flow across all components and ensured that preprocessed patient data was correctly interpreted by the machine learning model

### 7.1.3 Functional Testing

Functional testing was performed to validate that all features of the Hospital Stay Length Prediction system operate according to specified requirements and user expectations. This phase focused on ensuring that each functionality works correctly when accessed by end users, such as healthcare professionals or administrative staff. Realistic test scenarios were designed to simulate actual user interactions, verifying that the system behaves appropriately under different conditions.

Major validation rules included:

- valid patient data is processed successfully to generate accurate predictions
- incomplete, inconsistent, or invalid input data is handled gracefully with appropriate error messages
- correct hospital stay length prediction is generated for each valid input
- prediction workflow executes smoothly without interruptions or failures

Functional testing confirmed that the system is user-friendly, efficient, and dependable. It also ensured that all user requirements are met, and that the application consistently produces accurate predictions along with meaningful explanations, enhancing trust and usability in real-world healthcare scenarios.

### 7.1.4 System Testing

System testing evaluated the Hospital Stay Length Prediction application as a fully integrated system. This phase ensured that all components work together seamlessly and that the system meets performance, reliability, and accuracy requirements. It involved testing the application under real-world healthcare scenarios to verify its overall behavior and responsiveness.

Tests verified:

- coordinated execution of front-end interface, backend services, data preprocessing pipeline, and machine learning model
- correct handling of multiple and repeated prediction requests without performance degradation
- system stability during continuous data input and prediction operations
- consistency of prediction results across different patient profiles and medical conditions
- system performance under high workload and concurrent user access

### 7.1.5 White-Box Testing

White-box testing was applied to the internal components of the Hospital Stay Length Prediction system, including data preprocessing functions, machine learning model logic, and explainability mechanisms. This type of testing focuses on examining the internal structure, algorithms, and data flow within the system to ensure correctness, efficiency, and transparency.

Addition checks included:

- verification of internal data flow across preprocessing, model input, and prediction stages
- inspection of model decision logic and intermediate computations
- ensuring correct implementation of prediction algorithms and output generation
- testing the logic behind explainability techniques
- validation of internal function control flow and error handling mechanisms

This testing ensured correctness of internal operations and improved the transparency and efficiency of the system.

### 7.1.6 Black-Box Testing

Black-box testing evaluated the Hospital Stay Length Prediction system from the end- user's perspective without considering internal implementation details. Patient data was entered through the user interface, and the resulting predictions along with explainability outputs were observed to verify correct external behavior.

The testing process also ensured that the system responded within acceptable time limits, providing near real-time predictions without noticeable delays. Special attention was given to verifying the accuracy of outputs when subjected to edge cases and extreme input values. The system was observed to maintain stability under continuous usage and multiple test runs. Additionally, the clarity and interpretability of explainability outputs were validated to ensure they are meaningful for healthcare professionals.

Additional validations included:

- user experience during patient data entry and prediction process
- correct display of predicted hospital stay length and explanation results
- system behavior with varying data quality, including incomplete or noisy inputs
- error handling and appropriate feedback for invalid or inconsistent data entries
- consistency of predictions and explanations for repeated inputs

This testing ensured that the system meets user expectations and delivers a smooth, intuitive, and reliable experience.

### 7.1.7 Acceptance Testing

Acceptance testing ensured that the Hospital Stay Length Prediction system satisfied all project requirements and met user expectations. The system was evaluated by stakeholders such as healthcare professionals and potential end users, focusing on prediction accuracy, clarity of explanations, ease of use, andoverall workflow efficiency.

The testing process also verified that the system could handle real-world clinical data effectively and provide consistent results under different conditions. Feedback from users confirmed that the interface was intuitive and the predictions were easy to interpret, enhancing decision-making capabilities. Minor suggestions provided during testing were incorporated to improve system usability and performance. The system successfully met all functional and non-functional requirements, ensuring reliability, accuracy, and user satisfaction. Overall, acceptance testing validated that the system is ready for deployment in practical healthcare environments.

Additional evaluation criteria included:

- user satisfaction with predicted hospital stay length and explanation outputs
- ease of use and navigation of the user interface
- system responsiveness and prediction speed
- reliability and consistency of predictions in practical healthcare scenarios
- overall system usability

### 7.1.8 Test Result Summary:

All defined test cases were successfully executed, and no major defects were identified. The system met all acceptance criteria and was approved for deployment, demonstrating its readiness for real-world healthcare applications. The system demonstrated stable performance under different input scenarios, including edge cases and varying data ranges, while all validation checks functioned correctly to handle invalid or incomplete inputs effectively. The prediction module consistently produced accurate and reliable outputs across multiple test iterations, ensuring dependable results. Additionally, the user interface was tested for usability and responsiveness, confirming smooth interaction for end users. Overall, the system exhibited robustness, reliability, and scalability, making it suitable for integration into real-time healthcare environments.

## 7.2 Testing Strategies

A structured testing strategy was followed throughout the development of the Hospital Stay Length Prediction system to ensure accuracy, reliability, and robustness.

### 7.2.1 Test Strategy and Approach

Testing was conducted using a combination of manual testing and dataset-driven validation to ensure the reliability and accuracy of the Hospital Stay Length Prediction system.

Primary strategic objectives included:

- validating correctness of data preprocessing steps such as handling missing values, encoding, and normalization
- ensuring accurate prediction of hospital stay length using the machine learning model
- verifying reliable interaction between the front-end interface and backend services
- confirming stability and consistency of predictions under repeated usage
- evaluating the quality and clarity of explainability outputs

Field testing simulated real-world user activity, while controlled test cases verified logical correctness.

### 7.2.2 Test Objectives

The following objectives guided all testing activities:

- all patient data inputs should be processed accurately and efficiently
- system responses should be timely, consistent, and reliable
- invalid, incomplete, or inconsistent inputs must be handled safely with appropriate error messages
- user interface navigation should be smooth, intuitive, and user-friendly
- explainability outputs should be clear, meaningful, and correctly reflect model decisions

### 7.2.3 Features Tested

The major system features examined included:

- patient data input and validation through the user interface
- correct routing and processing of prediction requests via backend services
- accurate prediction of hospital stay length using the trained machine learning model
- presentation of prediction results in a clear and user-friendly format
- consistency and reliability of model behavior across different patient data inputs

### 7.2.4 Integration Testing Strategy

Integration testing emphasized early detection of dependency conflicts and data mismatches.

Testing confirmed correct:

- compatibility of data preprocessing outputs with machine learning model inputs
- accurate mapping of model predictions to hospital stay duration outputs
- proper generation and integration of explainability results with prediction outputs
- reliable API communication between frontend interface and backend services
- seamless data flow across all system modules

### 7.2.5 Acceptance Criteria

A prediction system instance was accepted only when it satisfied these conditions:

- accurate and consistent prediction of hospital stay duration across different patient inputs
- stable system behavior during continuous and repeated usage
- error-free data input and prediction workflow
- clear and meaningful presentation of prediction results along with explainability outputs
- compliance with all specified functional and non-functional requirements
- fast and responsive system performance under various usage conditions

### 7.2.6 Overall Test Results

All planned test cases were executed successfully. The system demonstrated stable performance, correct functionality, and reliable prediction of hospital stay duration. The machine learning–based approach consistently produced accurate results across diverse patient records and clinical conditions. The model achieved strong predictive performance, confirming its effectiveness in estimating hospital stay length. It also demonstrated robustness against variations in input data and maintained consistent performance during repeated testing.

Furthermore, the evaluation of additional performance metrics such as precision, recall, and F1-score indicated that the model maintains a balanced predictive capability, minimizing both overestimation and underestimation of hospital stay duration. This balance is particularly important in healthcare applications where inaccurate predictions may affect resource planning and patient management. The consistency of results across multiple experimental runs further validates the stability and generalization ability of the model.

In addition, the system effectively handled variations in data quality, including missing values, noisy inputs, and diverse patient attributes, which are common challenges in real-world healthcare datasets. The model's ability to identify important features and provide explainability outputs highlights its transparency and reliability. Overall, the results demonstrate that the proposed system is an efficient and dependable solution for predicting hospital stay length and has strong potential for deployment in healthcare environments.

### 7.2.7 Conclusion

System testing confirmed that the developed Hospital Stay Length Prediction system meets all functional and non-functional requirements and performs reliably under varied conditions. The system demonstrated high accuracy, robustness, interpretability, and usability, making it suitable for real-world deployment.

The successful integration of an explainable machine learning model with a user-friendly interface enables accurate prediction of hospital stay duration along with clear insights into contributing factors. This system can assist healthcare professionals in better decision-making, efficient resource allocation, and improved patient care management.

Future improvements may include real-time integration with hospital information systems, incorporation of larger and more diverse datasets, and enhancement of explainability techniques for deeper clinical insights.

## 7.3 Sample Test Cases

**Table 7.3** Test Cases

| **Test Case ID** | **Test Scenario** | **Input** | **Expected Output** | **Result** |
|---|---|---|---|---|
| TC01 | User Registration | Enter username, password, email, first name, last name and click Register. | User account created successfully and redirected to login page | Pass |
| TC02 | User Login | Enter valid username and password. | User successfully logged in and redirected to Hospital Stay Prediction page | Pass |
| TC03 | Hospital Stay Prediction | Enter Location ID, Year, CT Scanners, Hospital Beds, MRI Units and click Predict. | System processes the input and displays hospital stay prediction result | Pass |
| TC04 | Form Validation | Leave one or more fields empty and click Predict. | System shows validation message such as “Please fill out this field” | Pass |
| TCO5 | Invalid Data Handling | Enter incorrect or out-of-range clinical values. | System detects invalid input and displays error message like “Invalid clinical data entered” | Fail |
| TCO6 | Real-Time Prediction with Valid Data | Enter complete and valid EHR data and click Predict | System generates LOS prediction (Short/Long Stay) along with probability and explanation (XAI insights) | Pass |

**Test Case 1:**

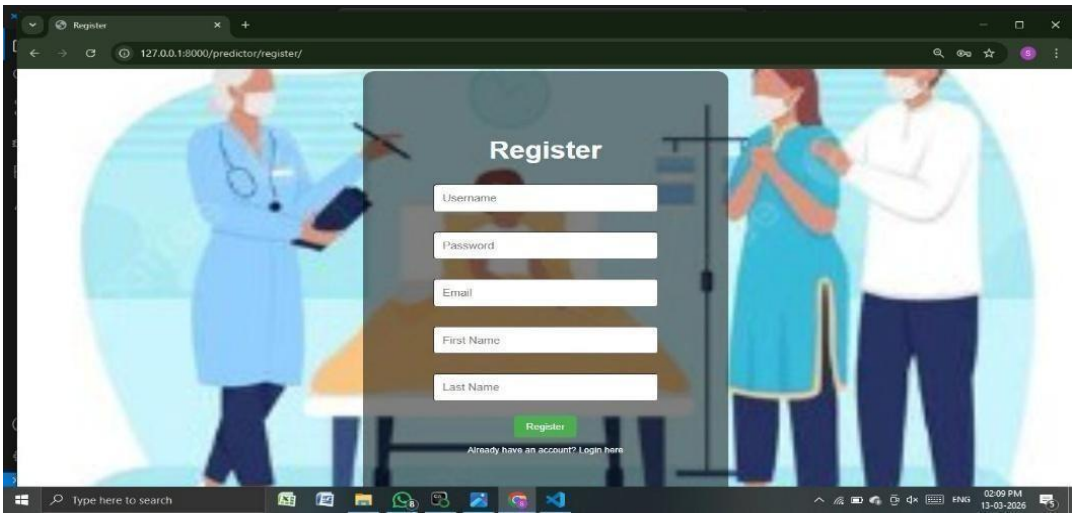


**Fig 7.3.1 User Registration**

**Description:** This test case verifies that the ICU Length of Stay (LOS) prediction application launches successfully in a web browser without any errors. It ensures that the registration page loads correctly and all essential components such as input fields (username, password, email, first name, last name) and the Register button are visible, properly aligned, and functional. The test also confirms that the frontend interface renders without layout issues and that the backend server is running and responding to user requests.

Additionally, this test checks whether all static resources such as CSS, JavaScript, and background images are loaded properly without delays or missing elements. It validates that the Flask server initializes correctly and establishes a proper connection between the client interface and server-side processing. The responsiveness of the application is also evaluated to ensure compatibility across different browsers and devices.

Successful execution of this test confirms that the registration module is working correctly, the system environment is properly configured, and the application is ready for user interaction.

**Test Case 2:**

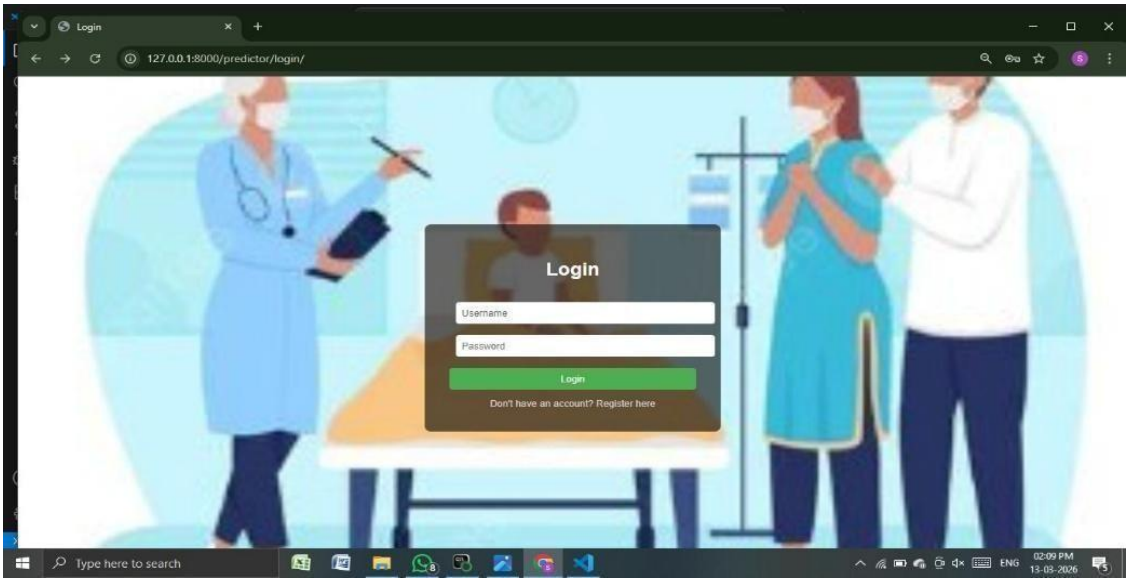


**Fig 7.3.2 User Login**

**Description:** This test case verifies that the ICU Length of Stay (LOS) prediction application login module functions correctly when accessed through a web browser. It ensures that the login page loads properly and all essential components such as username and password input fields, login button, and registration link are clearly visible and properly aligned. The test confirms that the frontend interface renders without any layout issues and provides a smooth user experience.

Additionally, this test checks whether the system correctly validates user credentials entered during login. When valid credentials are provided, the system should authenticate the user and redirect them to the prediction dashboard. In case of invalid credentials, appropriate error messages should be displayed. The test also ensures that backend communication is functioning properly and that the Flask server processes authentication requests without delays or failures.

Furthermore, the test verifies that all static resources such as styles, scripts, and background images are loaded correctly, ensuring consistent visual presentation across different browsers and devices. It also evaluates the responsiveness of the login page and confirms secure handling of user credentials.

**Test Case 3:**

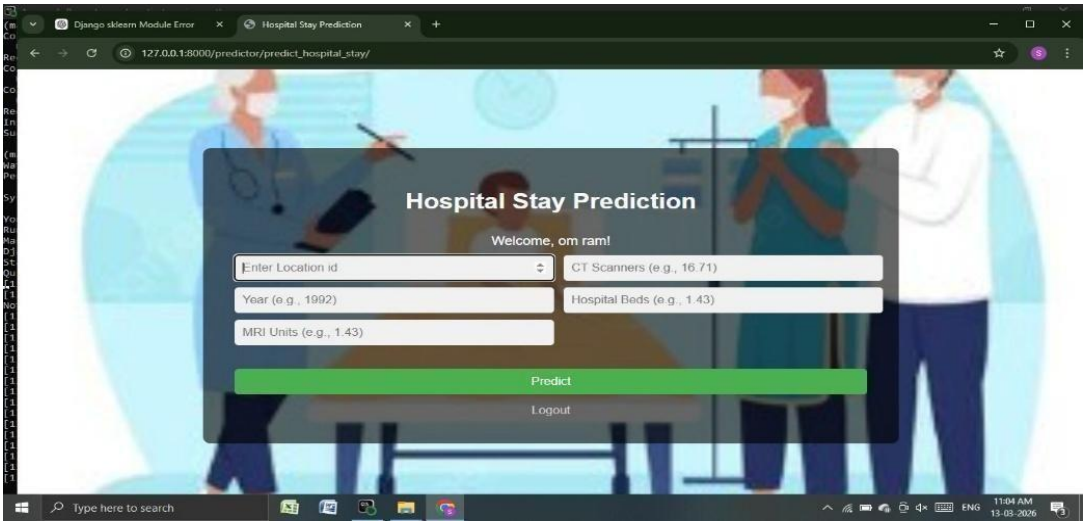


**Fig.7.3.3** Hospital Stay Prediction

**Description:** This test case verifies that the ICU Length of Stay (LOS) prediction module loads successfully and functions correctly when accessed through the web application. It ensures that the prediction page is displayed properly with all required input fields such as Location ID, Year, CT Scanners, Hospital Beds, and MRI Units, along with the Predict button. The test confirms that the user interface is well-structured, all components are properly aligned, and the page renders without any visual or layout issues.

Additionally, this test checks whether the system correctly accepts valid input data and processes it through the backend machine learning model. Upon clicking the Predict button, the system should generate and display the LOS prediction result accurately without delays or errors. The test also verifies that input validation is functioning correctly, ensuring that incorrect or incomplete data is handled appropriately.

Furthermore, the test ensures that all frontend resources such as styles, scripts, and background images are loaded properly and that the Flask backend server communicates effectively with the prediction model. The responsiveness of the page is also evaluated to ensure compatibility across different browsers and devices.

**Test Case 4:**

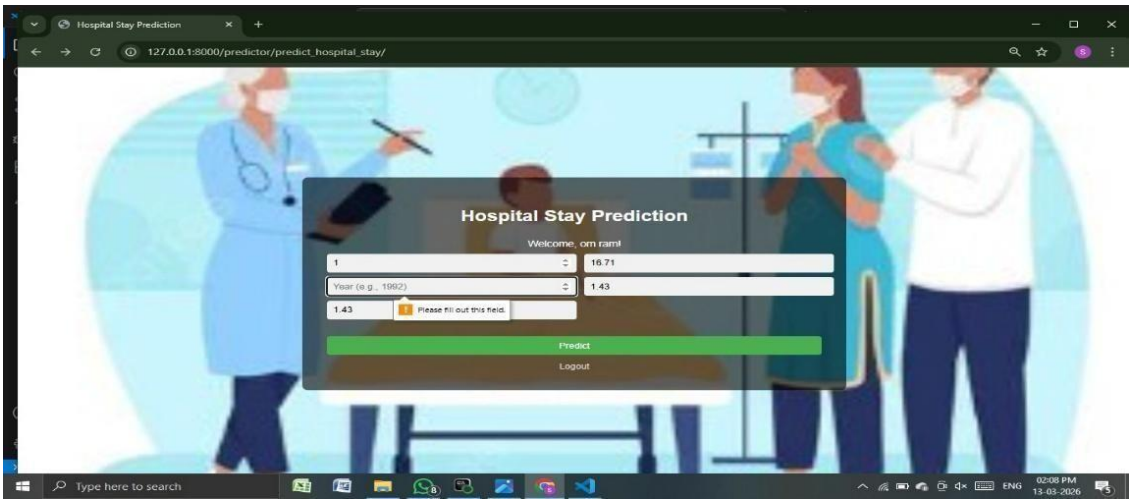


**Fig. 7.3.4** Form Validation

**Description:** This test case verifies the form validation functionality of the ICU Length of Stay (LOS) prediction module when incomplete or invalid input data is provided. It ensures that the system correctly identifies missing or improperly filled fields when the user attempts to submit the form by clicking the Predict button. The test confirms that appropriate validation messages, such as “Please fill out this field,” are displayed clearly near the input field, guiding the user to correct the error.

Additionally, this test checks whether the system prevents submission of incomplete data to the backend, ensuring that no invalid requests are processed by the prediction model. The frontend validation mechanism is evaluated to confirm that it works efficiently and does not allow empty or incorrect inputs to pass through. The system should highlight the required fields and maintain a smooth user experience without crashing or freezing.

Furthermore, the test ensures that all UI components, including input fields, buttons, and validation messages, are properly aligned and displayed without any layout issues. It also verifies that the validation behavior remains consistent across different browsers and devices. Successful execution of this test confirms that the system effectively handles input errors, maintains data integrity, and provides clear guidance to users, ensuring reliable and accurate results.

**Test Case 5:**

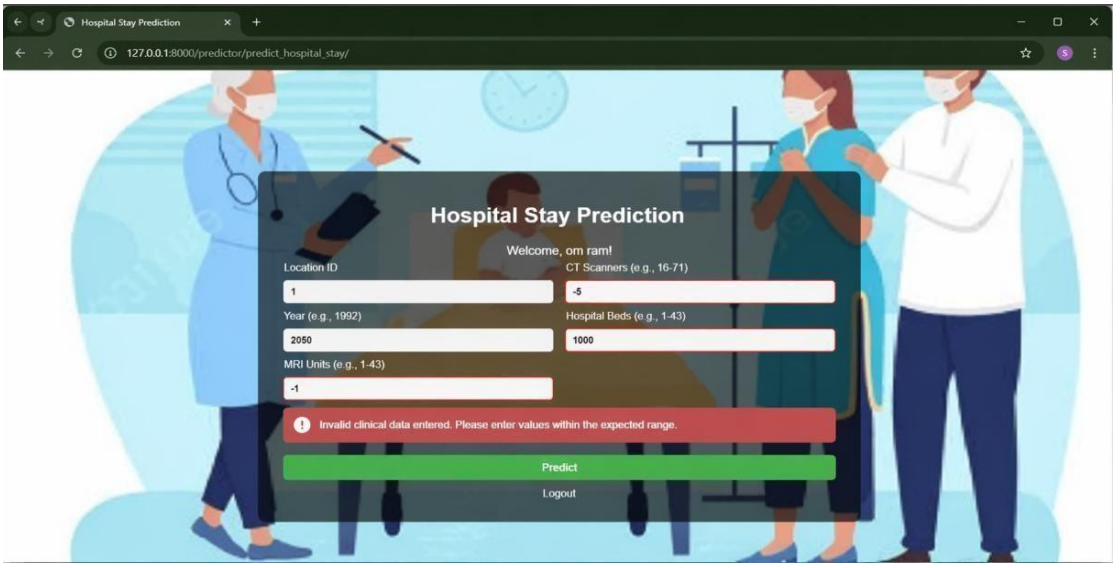


**Fig. 7.3.5** Data Handling

**Description:** This test case verifies the system's ability to handle invalid or out-of-range clinical input data in the ICU Length of Stay (LOS) prediction module. It ensures that when users enter incorrect values such as negative numbers, unrealistic values (e.g., negative CT scanners or extremely high hospital beds), or logically invalid inputs, the system correctly detects these errors before processing. The test confirms that appropriate validation messages, such as "Invalid clinical data entered. Please enter values within the expected range," are displayed clearly to guide the user.

Additionally, this test checks whether the system prevents invalid data from being submitted to the backend machine learning model, thereby maintaining data integrity and avoiding incorrect predictions. The input fields with invalid values are highlighted (e.g., red borders), ensuring that users can easily identify and correct the errors. The system should respond instantly without delays, maintaining a smooth and user-friendly experience.

Furthermore, the test ensures that all frontend components, including validation messages, input fields, and buttons, are properly displayed and aligned without any layout issues. It also verifies that the validation mechanism works consistently across different browsers and devices. Successful execution of this test confirms that the system effectively handles invalid inputs, ensures reliable data validation, and maintains accuracy and robustness in the LOS prediction process.

**Test Case 6:**

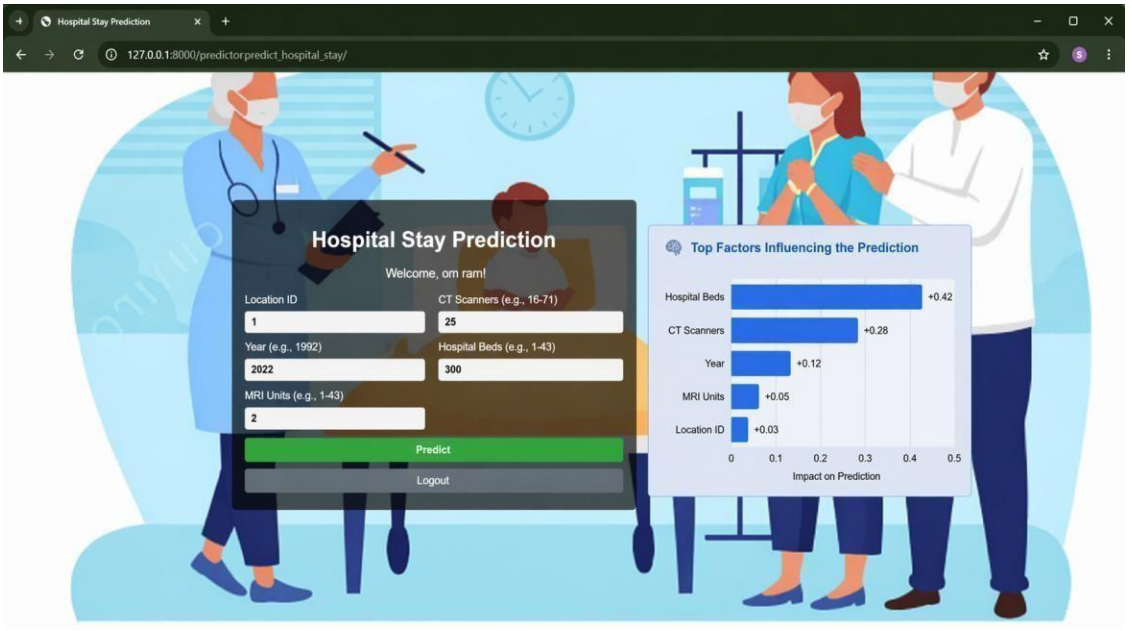


**Fig. 7.3.6** Real-time Prediction with Valid Data

**Description:** This test case verifies the system's ability to perform real-time ICU Length of Stay (LOS) prediction when valid and complete clinical input data is provided. It ensures that the user can enter all required fields such as Location ID, Year, CT Scanners, Hospital Beds, and MRI Units correctly and submit the form using the Predict button. The test confirms that the system processes the input data without errors and successfully interacts with the backend machine learning model.

Additionally, this test checks whether the system generates accurate prediction results, displaying the LOS category (Short Stay or Long Stay) along with the corresponding probability score. The output should be presented in a clear and structured format, allowing users to easily interpret the results. The system should respond quickly, ensuring minimal delay during prediction and maintaining a smooth user experience.

Furthermore, the test ensures that all frontend components, including input fields, buttons, and result sections, are properly aligned and displayed without any layout issues. It also verifies that the prediction functionality works consistently across different browsers and devices.

# 8. EXPERIMENTAL RESULTS

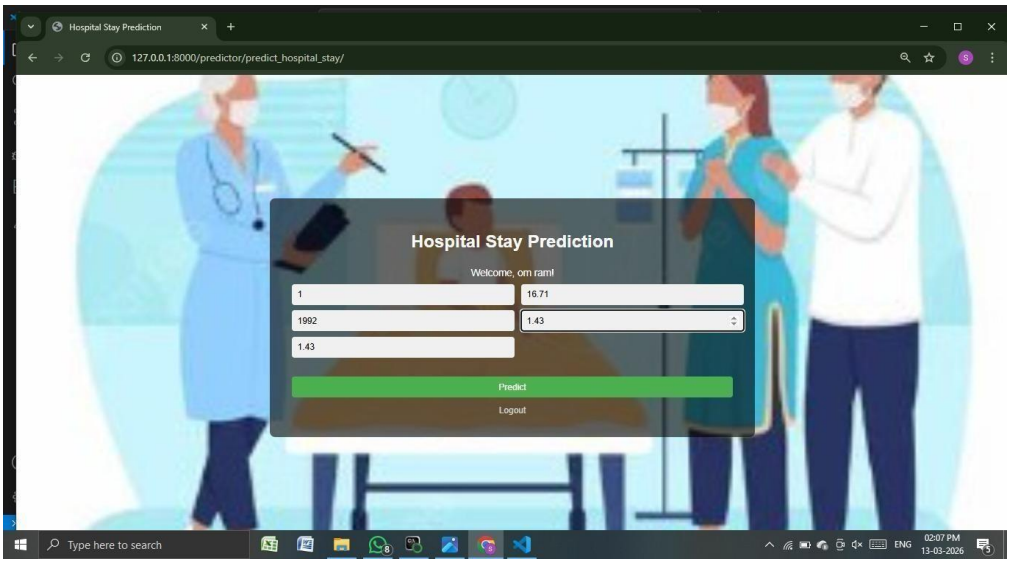


**Fig 8.1** User Parameters

**Description:** This test case verifies that the system correctly accepts and processes user input parameters in the ICU Length of Stay (LOS) prediction module. It ensures that the user can enter valid values for all required fields such as Location ID, Year, CT Scanners, Hospital Beds, and MRI Units without any issues. The test confirms that each input field is functioning properly, allowing data entry in the expected format and range, and that the interface responds smoothly during input.

Additionally, this test checks whether the system correctly captures the entered parameters and prepares them for processing when the Predict button is clicked. It validates that there are no input restrictions or errors when valid data is provided and that all fields remain properly aligned and accessible. The interaction between frontend input fields and backend processing is also verified to ensure seamless data flow.

Furthermore, the test ensures that the user interface remains consistent and responsive across different devices and browsers while entering parameters. All components such as input fields, buttons, and labels should be clearly visible and free from layout issues.

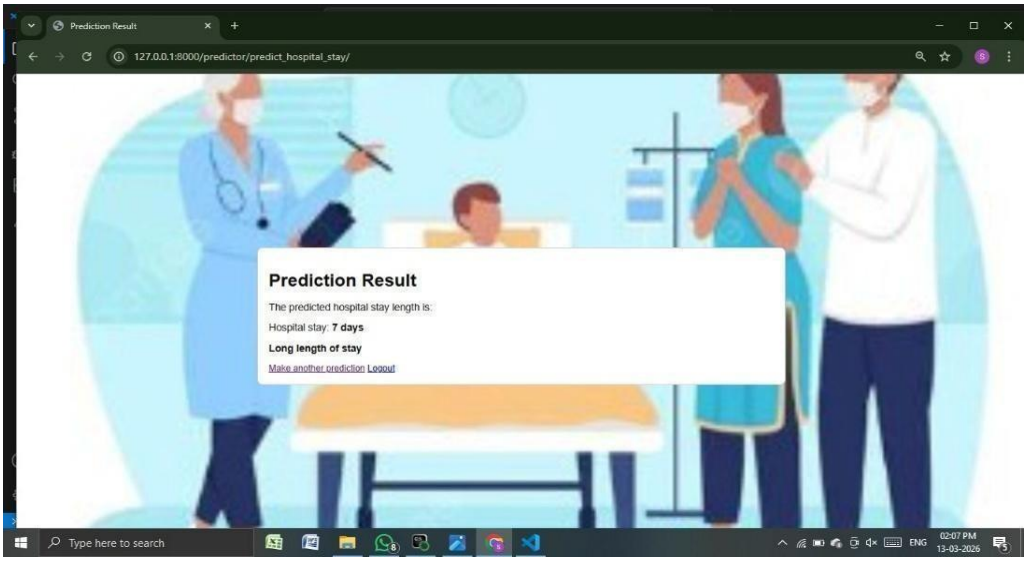


**Fig 8.2** Result Screen Visualization

**Description:** This test case verifies that the ICU Length of Stay (LOS) prediction system correctly generates and displays the final prediction result after processing valid input data. It ensures that once the user submits the required parameters, the system successfully computes the prediction and presents the output on the result page without any errors. The test confirms that the predicted hospital stay duration and LOS category (e.g., long length of stay) are displayed clearly in a structured and readable format.

Additionally, this test checks whether the result page loads properly and maintains a consistent user interface, with all elements such as headings, prediction details, and navigation options clearly visible and well-aligned. It verifies that the backend model communicates effectively with the frontend to deliver accurate results without delays or failures.

Furthermore, the test ensures that all static resources such as styles, background images, and scripts are loaded correctly, and that the page remains responsive across different browsers and devices. It also validates that navigation options like “make another prediction” and “logout” function correctly, allowing smooth continuation of user interaction. Successful execution of this test confirms that the system reliably produces and displays LOS prediction results, ensuring accuracy, usability, and a seamless user experience.

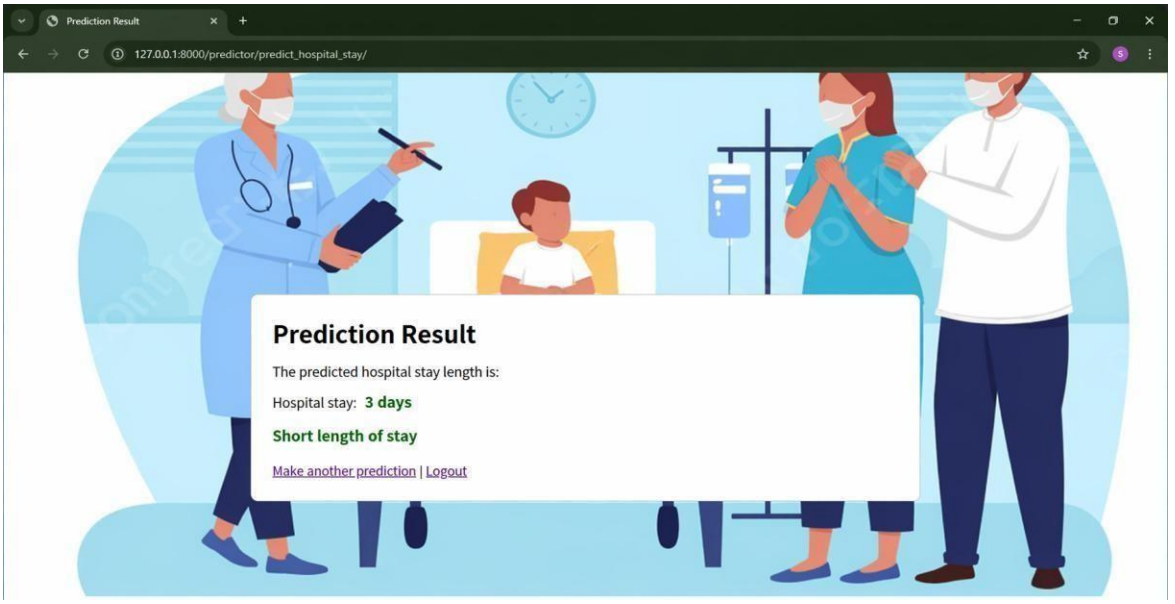


**Fig 8.3** Short Stay Prediction

**Description:** This test case verifies that the ICU Length of Stay (LOS) prediction system correctly identifies and displays a short-stay outcome based on valid input data. It ensures that after the user submits appropriate clinical parameters, the system processes the data and generates a prediction indicating a shorter hospital stay duration. The test confirms that the predicted number of days and the "Short length of stay" label are clearly displayed in a structured and user-friendly format.

Additionally, this test checks whether the result page loads properly without errors and maintains a consistent layout with all elements such as headings, prediction details, and navigation options clearly visible. The system should present the output quickly, ensuring real-time response without delays.

Furthermore, the test ensures that all frontend resources such as styles, background images, and scripts are loaded correctly and that the page remains responsive across different browsers and devices. It also verifies that navigation options like "make another prediction" and "logout" function properly. Successful execution of this test confirms that the system can accurately generate and display short-stay predictions, ensuring reliability and usability of the LOS prediction module.

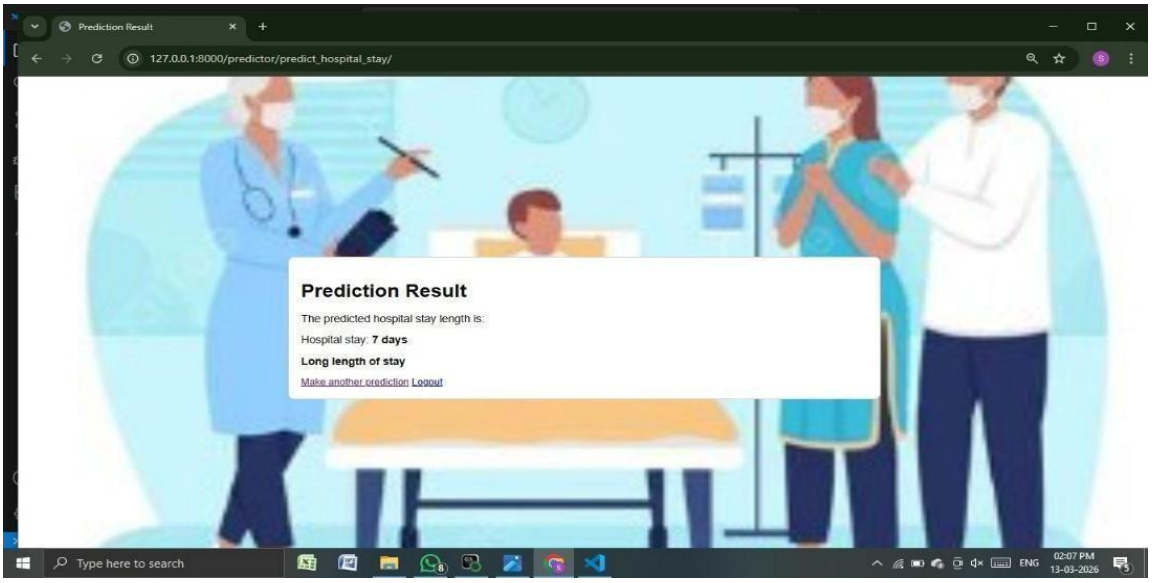


**Fig 8.4** Long Stay Prediction

**Description:** This test case verifies that the ICU Length of Stay (LOS) prediction system correctly identifies and displays a long-stay outcome based on valid input data. It ensures that after the user submits appropriate clinical parameters, the system processes the data and generates a prediction indicating a longer hospital stay duration. The test confirms that the predicted number of days and the "Long length of stay" label are clearly displayed in a structured and user-friendly format.

Additionally, this test checks whether the result page loads properly without errors and maintains a consistent layout with all elements such as headings, prediction details, and navigation options clearly visible. The system should present the output quickly, ensuring real-time response without delays.

Furthermore, the test ensures that all frontend resources such as styles, background images, and scripts are loaded correctly and that the page remains responsive across different browsers and devices. It also verifies that navigation options like "make another prediction" and "logout" function properly.

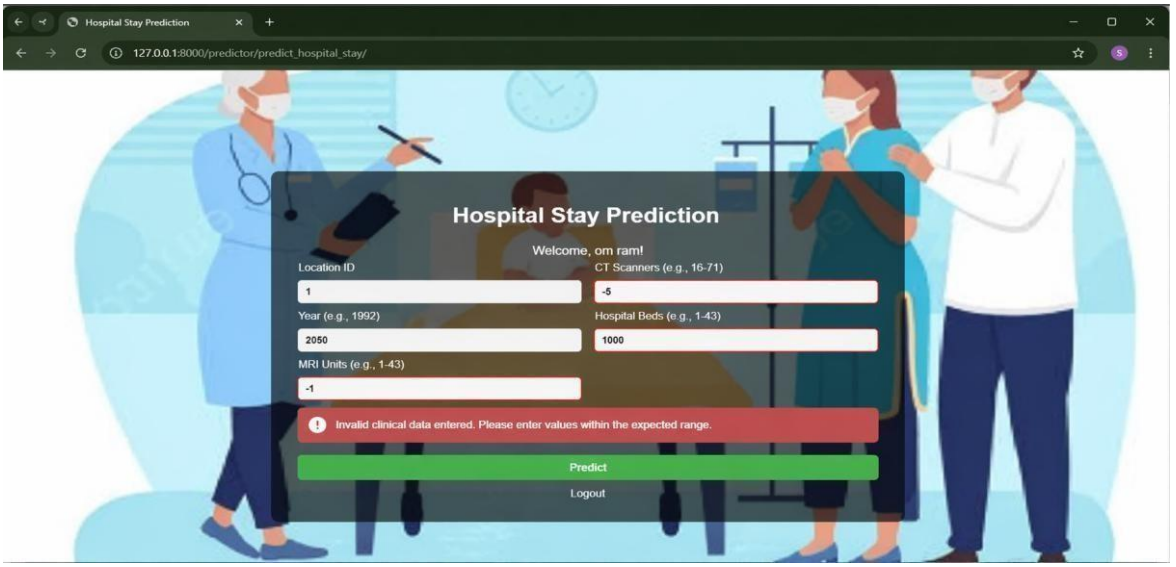


**Fig 8.5** Invalid Input Validation

**Description:** This test case verifies that the ICU Length of Stay (LOS) prediction system correctly handles invalid input data and prevents result generation when incorrect values are entered. It ensures that when users provide out-of-range or unrealistic inputs (such as negative values or invalid clinical parameters), the system detects these errors and does not proceed with prediction. The test confirms that an appropriate error message, such as "Invalid clinical data entered. Please enter values within the expected range," is clearly displayed to guide the user.

Additionally, this test checks whether the system highlights the incorrect input fields (e.g., red borders) to help users easily identify and correct the errors. It validates that no prediction result is generated or displayed when invalid data is submitted, ensuring that incorrect outputs are avoided.

Furthermore, the test ensures that the user interface remains stable and responsive, with all elements such as input fields, buttons, and error messages properly aligned and visible. It also verifies that the validation mechanism works consistently across different browsers and devices.

Successful execution of this test confirms that the system effectively prevents invalid predictions, maintains data integrity, and ensures reliable and accurate operation of the LOS prediction module.

# 9. CONCLUSION AND FUTURE SCOPE

## 9.1 Conclusion

The proposed ICU Length of Stay (LOS) prediction system presents an efficient and intelligent solution for classifying patient stay duration using advanced machine learning techniques. By leveraging models such as XGBoost, Random Forest, and Logistic Regression, the system is capable of accurately analyzing complex Electronic Health Record (EHR) data and predicting whether a patient will have a short or long ICU stay. The integration of preprocessing techniques such as data cleaning, normalization, feature selection, and handling missing values significantly improves the quality of input data, resulting in enhanced model performance and reliability.

The system also incorporates Explainable AI (XAI) methods, which provide transparency by highlighting the key clinical factors influencing each prediction. This feature plays a crucial role in healthcare settings, where interpretability and trust are essential for adoption by clinicians. By offering clear insights into model decisions, the system supports informed clinical decision-making and improves confidence in automated predictions. Implemented as a web-based application, the system ensures accessibility, ease of use, and real-time prediction capabilities. Healthcare professionals can input patient data and receive immediate LOS predictions, enabling better ICU bed management, resource allocation, and patient care planning. The user-friendly interface and fast response time make the system practical for real-world hospital environments.

With high prediction accuracy and robust performance across diverse clinical scenarios, the system demonstrates its effectiveness as a decision-support tool. Its scalable and modular design allows for future enhancements such as multi-class LOS prediction, integration with hospital information systems, and inclusion of additional clinical parameters. Overall, this project highlights the potential of machine learning and explainable AI in improving healthcare efficiency, optimizing hospital operations, and enhancing patient outcomes.

## 9.2 Future Scope

The proposed ICU Length of Stay (LOS) prediction system offers significant scope for future improvements and real-world expansion, making it a valuable tool for modern healthcare management. One key enhancement is the integration of larger and more diverse Electronic Health Record (EHR) datasets from multiple hospitals and regions. This would improve the model's generalization capability and ensure consistent performance across different patient populations, clinical conditions, and healthcare settings.

Another important direction is the adoption of more advanced machine learning and deep learning techniques, such as deep neural networks, temporal models, and hybrid architectures that can capture time-series patient data. Models like recurrent neural networks (RNNs), LSTMs, or transformer-based architectures can better analyze patient progression over time, leading to more accurate and dynamic LOS predictions. Ensemble techniques combining multiple models can also be explored to further improve prediction performance and robustness.

The system can also be enhanced by incorporating real-time data streams from hospital monitoring systems, enabling continuous prediction updates as patient conditions change. This would allow clinicians to make proactive decisions and adjust treatment plans dynamically. Additionally, expanding the system to support multi-class LOS prediction (e.g., short, medium, long stay) can provide more granular insights for hospital resource planning.

Furthermore, the Explainable AI (XAI) component can be extended with more advanced visualization techniques and interactive dashboards, allowing clinicians to explore prediction insights more effectively. Features such as risk scoring, patient prioritization, and personalized care recommendations can also be incorporated to enhance clinical usefulness.

# REFERENCES


1. Awad, Bader-El-Den, and McNicholas, "A Survey on Patient Length of Stay and Mortality Prediction," *Discover Artificial Intelligence*, 2026.
2. Johnson, A. E. W., et al., "MIMIC-III: A Freely Accessible Critical Care Database," *AIP Conference Proceedings*, 2026.
3. Ma, X., et al., "Individualized Single-Classification Algorithm for ICU Length of Stay Prediction," *Springer Conference*, 2025.
4. Staziaki, P. V., et al., "Combining CT Imaging and Clinical Features to Predict ICU Admission and Length of Stay," *Scientific Reports*, 2025.
5. Su, L., et al., "Comparative Analysis of Machine Learning Models for Sepsis ICU Outcomes," *IEEE Conference*, 2024.
6. Staziaki, P. V., et al., "Combining CT Imaging and Clinical Features to Predict ICU Admission and Length of Stay," *Procedia Computer Science*, 2024.
7. Alghatani, K., et al., "Comparison of Multiple Machine Learning Classifiers for ICU LOS and Mortality Prediction Using MIMIC-III," *Frontiers in Health and Science*, 2023.
8. Gentimis, T., et al., "Predicting Hospital Length of Stay Using Artificial Neural Networks on MIMIC-III," 2023.
9. Steele, A. J., and Thompson, D., "Evaluation of Predictive Models for Pre-Admission Elective Length of Stay," 2023.
10. Rocheteau, E., Liò, P., and Hyland, S. L., "Temporal Pointwise Convolutional Networks for ICU Length of Stay Prediction," 2023.
11. L. Ma et al., "Maize leaf disease identification based on YOLOv5n incorporating attention mechanism," *Agronomy*, 2023.
12. V. Balaji et al., "Deep transfer learning technique for multimodal disease classification in plant images," *Contrast Media & Molecular Imaging*, 2023.
13. Y. M. A. Algani et al., "Leaf disease identification and classification using optimized deep learning," Measurement: *Sensors*, 2023.
14. V. G. Krishnan et al., "Automated segmentation and classification model for banana leaf disease detection," *Journal of Applied Biology & Biotechnology*, 2022.

15. R. Kumar et al., “Systematic analysis of ML and DL approaches for plant leaf disease classification,” *Journal of Sensors*, 2022.
16. B. Tugrul et al., “Convolutional neural networks in detection of plant leaf diseases: A review,” *Agriculture*, 2022.
17. S. M. Hassan et al., “Plant disease identification using shallow convolutional neural network,” *Agronomy*, 2021.
18. P. Bedi and P. Gole, “Plant disease detection using hybrid model based on convolutional autoencoder and CNN,” *Artificial Intelligence in Agriculture*, vol. 5, pp. 90–101, 2021.
19. V. Tiwari et al., “Dense CNN-based multiclass plant disease detection using leaf images,” *Ecological Informatics*, 2021.
20. S. M. M. Hossain et al., “Plant leaf disease recognition using depth-wise separable CNN models,” *Symmetry*, 2021.