Chronic Kidney Disease Prediction Using Machine Learning And Deep Learning Models

A Project Report submitted in the partial fulfilment

of the Requirements for the award of the degree

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

N.Revanth	(21471A0541)
K.Veera Raghava Reddy	(21471A0531)
Y.Likhith Prasanna Kumar	(21471A0570)
K.Mahesh Babu	(21471A0533)

Under the esteemed guidance of

Shaik Rafi , M. Tech.,(Ph.D)

Assistant.Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NARASARAOPETA ENGINEERING COLLEGE: NARASARAOPET

(AUTONOMOUS)

Accredited by NAAC with A+ Grade and NBA under Tier -1

NIRF rank in the band of 201-300 and an ISO 9001:2015 Certified

Approved by AICTE, New Delhi, Permanently Affiliated to JNTUK, Kakinada

KOTAPPAKONDA ROAD, YALLAMANDA VILLAGE, NARASARAOPET-522601

2024-2025

NARASARAOPETA ENGINEERING COLLEGE

(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project that is entitled with the name "Chronic Kidney Disease Prediction Using Machine learning And Deep Learning Models" is a bonafide work done by the team N. Revanth (21471A0541), K.Veera Raghava Reddy (21471A0531), K.Mahesh Babu (21471A0533),Y.Likhith Prasanna Kumar (21471A0570) in partial fulfillment of the requirements for the award of the degree of BACHELOR OF TECHNOLOGY in the Department of COMPUTER SCIENCE AND ENGINEERING during 2024-2025.

PROJECT GUIDE

Shaik.Rafi, M.Tech., (Ph.D)

Assistant Professor

PROJECT CO-ORDINATOR

Dodda Venkata Reddy,M.Tech,(Ph.D)

Assistant Professor

HEAD OF THE DEPARTMENT

Dr. S. N. Tirumala Rao, M.Tech., Ph.D

Professor & HOD

EXTERNAL EXAMINER

DECLARATION

We declare that this project work titled " CHRONIC KIDNEY PREDICTION USING MACHINE LEARING AND DEEP LEARNING MODELS" is composed by ourselves that the work contain here is our own except where explicitly stated otherwise in the text and that this work has not been submitted for any other degree or professional qualification except as specified.

N.Revanth	(21471A0541)
K.Mahesh Babu	(21471A0533)
K.Veera Raghava Reddy	(21471A0531)
Y.Lihith Prasanna Kumar	(21471A0570)

ACKNOWLEDGEMENT

We wish to express our thanks to various personalities who are responsible for the successful completion of the project. We are extremely thankful to our beloved chairman Sri **M. V. Koteswara Rao**, B.Sc., who took keen interest in us in every step and effort throughout this course. We owe out sincere gratitude to our beloved principal **Dr. S. Venkateswarlu**, M. Tech., Ph.D., for showing his kind attention and valuable guidance throughout the course.

We express our deep felt gratitude towards **Dr. S. N. Tirumala Rao**, M.Tech., Ph.D., HOD of CSE department and also to our guide **Shaik.Rafi**, M.Tech., (Ph.D), Professor of CSE department whose valuable guidance and unstinting encouragement enable us to accomplish our project successfully in time.

We extend our sincere thanks towards **Dodda Venkata Reddy**, M.Tech.,(Ph.D), Assistant professor & Coordinator of the project for extending his encouragement. Their profound knowledge and willingness have been a constant source of inspiration for us throughout this project work.

We extend our sincere thanks to all other teaching and non-teaching staff to department for their cooperation and encouragement during our B.Tech degree.

We have no words to acknowledge the warm affection, constant inspiration and encouragement that we received from our parents.

We affectionately acknowledge the encouragement received from our friends and those who involved in giving valuable suggestions had clarifying our doubts which had really helped us in successfully completing our project.

By

N.Revanth (21471A0541)
K.Mahesh Babu (21471A0533)
K.Veera Raghava Reddy (21471A0531)
Y.Likhith Prasanna Kumar (21471A0570)

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

INSTITUTE VISION AND MISSION

INSTITUTION VISION

To emerge as a Centre of excellence in technical education with a blend of effective student centric teaching learning practices as well as research for the transformation of lives and community.

INSTITUTION MISSION

M1: Provide the best class infra-structure to explore the field of engineering and research.

M2: Build a passionate and a determined team of faculty with student centric teaching, imbibing experiential, innovative skills.

M3: Imbibe lifelong learning skills, entrepreneurial skills and ethical values in students for addressing societal problems.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

VISION OF THE DEPARTMENT

To become a centre of excellence in nurturing the quality Computer Science & Engineering professionals embedded with software knowledge, aptitude for research and ethical values to cater to the needs of industry and society.

MISSION OF THE DEPARTMENT

The department of Computer Science and Engineering is committed to

M1: Mould the students to become Software Professionals, Researchers and Entrepreneurs by providing advanced laboratories.

M2: Impart high quality professional training to get expertize in modern software tools and technologies to cater to the real time requirements of the Industry.

M3: Inculcate team work and lifelong learning among students with a sense of societal and ethical responsibilities.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Specific Outcomes (PSO's)

PSO1: Apply mathematical and scientific skills in numerous areas of Computer Science and Engineering to design and develop software-based systems.

PSO2: Acquaint module knowledge on emerging trends of the modern era in Computer Science and Engineering

PSO3: Promote novel applications that meet the needs of entrepreneur, environmental and social issues.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Educational Objectives (PEO's)

The graduates of the programme are able to:

PEO1: Apply the knowledge of Mathematics, Science and Engineering fundamentals to identify and solve Computer Science and Engineering problems.

PEO2: Use various software tools and technologies to solve problems related to academia, industry and society.

PEO3: Work with ethical and moral values in the multi-disciplinary teams and can communicate effectively among team members with continuous learning.

PEO4: Pursue higher studies and develop their career in software industry.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Outcomes

Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

Problem analysis: Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activitieswith an understanding of the limitations.

The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

Life-long learning: Recognize the need for, and have the preparation and ability to engagein independent and life-long learning in the broadest context of technological change.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Project Course Outcomes (CO'S):

CO421.1: Analyse the System of Examinations and identify the problem.

CO421.2: Identify and classify the requirements.

CO421.3: Review the Related Literature.

CO421.4: Design and Modularize the project.

CO421.5: Construct, Integrate, Test and Implement the Project.

CO421.6: Prepare the project Documentation and present the Report using appropriate method.

Course Outcomes – Program Outcomes mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
C421.1		✓											✓		
C421.2	✓		✓		✓								✓		
C421.3				✓		✓	✓	✓					✓		
C421.4			✓			✓	✓	✓					✓	✓	
C421.5					✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C421.6									✓	✓	✓		✓	✓	

Course Outcomes – Program Outcome correlation

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
C421.1	2	3											2		
C421.2			2		3								2		
C421.3				2		2	3	3					2		
C421.4			2			1	1	2					3	2	
C421.5					3	3	3	2	3	2	2	1	3	2	1
C421.6									3	2	1		2	3	

Note: The values in the above table represent the level of correlation between CO's and PO's:

1. Low level
2. Medium level
3. High level

Project mapping with various courses of Curriculum with AttainedPO's:

Name of the Course from Which Principles Are Applied in This Project	Description of the Task	Attained PO
C2204.2, C22L3.2	Defining the problem and applying advanced feature engineering techniques for height prediction models	PO1, PO3
CC421.1, C2204.3, C22L3.2	Critically analyzing project requirements and identifying suitable process models for experiments	PO2, PO3
CC421.2, C2204.2, C22L3.3	Creating logical designs using UML while collaborating on feature engineering as a team	PO3, PO5, PO9
CC421.3, C2204.3, C22L3.2	Testing, integrating, and evaluating regression models with and without outlier removal	PO1, PO5
CC421.4, C2204.4, C22L3.2	Documenting experiments, results, and findings collaboratively within the group	PO10
CC421.5, C2204.2, C22L3.3	Presenting each phase of the project, including raw data analysis and evaluation, in a group setting	PO10, PO11
C2202.2, C2203.3, C1206.3, C3204.3, C4110.2	Implementing and validating models with applications for healthcare and future feature updates	PO4, PO7
C32SC4.3	Designing a web interface to visualize predictions and verify model accuracy effectively	PO5, PO6

ABSRTACT

Chronic kidney disease is a noticeable health condition that can persist throughout an individual's life, resulting from either kidney malignancy or diminished kidney function. In this work, we investigate how several machine learning techniques might provide an early CKD diagnosis. While previous research has extensively explored this area, our aim is to refine our approach by employing predictive modeling techniques. Initially, we considered 25 variables alongside the class property. The data set used in this study underwent extensive processing, including changing the names of colours for clarity, converting identified colours to numbers, treating unique values with letters handling of partitioned values, fixing incorrect val ues, filling null values with mean, and encoding categorical values into mathematical notation. In addition,Principal component analysis (PCA) was also employed to lower dimensionality. Our findings demonstrated that the XG Boost classifier surpassed every other algorithm, with an accuracy of 0.991.

INDEX

LIST OF FIGURES

1.INTRODUCTION

The progressive and irreversible degeneration of renal cells is the hallmark of the sickness known as chronic kidney disease [1]. When CKD develops, harmful wastes accumulate in the body, leading to various health com plications. Chronic kidney disease (CKD) can cause a range of symptoms including cough, fever, dry mouth, nausea, back pain and abdominal pain. CKD is often associated with two risk factors: diabetes and hyper tension.

Therefore early diagnosis and treatment are essential. There are encouraging opportunities to improve early detection of CKD through machine learning and predictive modelling[2].The research guarantees that advanced preprocessing methods will be employed to predict CKD utilizing machine learning algorithms.

Previous manipulation of The data collected in this research [4] was extensive and included changing column names to improve readability, correcting incorrect assumptions, averaging a missing values will be replaced, and categorical variables will be assigned numerical labels. Several models such as XG Boost, AdaBoost, SVM, Decision Tree, Chi-Square, and KNN [4] are developed and evaluated.

The Stages of chronic kidney disease are:

a) **Early stages of chronic kidney disease :** Chronic kidney disease is often completely asymptomatic in its early stages[1]. This is because with a significant decrease in kidney function, the body can become more adaptive. CKD is often diagnosed during routine screening for other medical conditions, such as blood or urine tests[3]. Early detection is important as this allows for drug treatment through regular testing and ongoing monitoring.[2]

b) **CKD in Its Advanced Stages :** Many symptoms may appear if CKD is not identified early or if it gets worse despite treatment. Compared to when the kidney no longer functions, known as ESRD or ESKD, there is no possibility of survival without either dialysis or a kidney transplant.

c) **Time to see a doctor :** If you have indications of kidney disease, you should see a doctor immediately. Early detection of CKD can avoid kidney failure.[2]During medical testing, doctors may use blood and urine tests to measure kidney functioning and blood pressure, especially if you suffer from illnesses that increase

your risk of developing renal disease. Discuss with your physician the significance of this test.

d) **Tests for CKD :** A illness or other ailment that damages the kidneys gradually leads to chronic kidney disease (CKD).

Research reveals a 6.23% annual rise in CKD hospital ad missions despite a steady global death rate.[4] Numerous diagnostic techniques are used to determine the status of chronic kidney disease such as eGFR, urine tests, and a blood pressure tests.[3] For diagnosis of kidney injury or structural abnormalities, further testing such as MRI scans, ultrasound, or CT scans may be required.

Chronic Kidney Disease (CKD) is a progressive and irreversible condition characterized by the gradual loss of kidney function over time. It affects millions of people worldwide and is often associated with comorbidities such as diabetes, hypertension, and cardiovascular diseases. CKD is categorized into five stages based on the estimated glomerular filtration rate (eGFR), with the final stage requiring renal replacement therapy, such as dialysis or kidney transplantation [1].

Early detection of CKD is crucial, as timely intervention can slow its progression and improve patient outcomes. However, traditional diagnostic methods rely heavily on clinical testing, such as serum creatinine levels, blood urea nitrogen (BUN) tests, and urine analysis, which may not always provide early warnings of disease onset [2]. Machine learning (ML) and deep learning (DL) techniques have recently emerged as promising tools for CKD prediction, offering enhanced accuracy and efficiency in early diagnosis [3].

Role of Machine Learning And Deep Learning in CKD Prediction

With the increasing availability of electronic health records (EHRs) and medical datasets, machine learning (ML) and deep learning (DL) techniques have emerged as promising tools for CKD prediction, offering enhanced accuracy and efficiency in early diagnosis [3]. Traditional statistical methods are limited in identifying complex patterns within medical data, whereas ML algorithms can effectively analyze large-scale patient data to detect early CKD markers.

Previous manipulation of the data collected in this research [4] was extensive and included changing column names to improve readability, correcting incorrect assumptions, averaging missing values, and assigning numerical labels to categorical variables. Several models, such as XGBoost, AdaBoost, Support Vector Machine (SVM), Decision Tree, Chi-Square, and K-Nearest Neighbors (KNN) [4], are developed and evaluated.

The adoption of predictive models in clinical practice can provide real-time risk assessment, enabling healthcare professionals to make informed decisions. Implementing artificial intelligence (AI)-driven CKD detection systems can not only improve early diagnosis but also aid in personalized treatment planning, reducing the burden of CKD on healthcare infrastructure.

Research Significance

This research aims to bridge the gap between conventional CKD diagnostic techniques and modern AI-driven solutions. By leveraging advanced machine learning models, this study focuses on enhancing prediction accuracy and identifying high-risk individuals before the disease progresses to severe stages. The integration of ML-based CKD detection can lead to:

- Improved patient outcomes through early intervention.
- Reduced hospitalization rates and associated healthcare costs.
- Increased awareness and screening among high-risk populations.
- Development of user-friendly CKD prediction tools for clinical and home-based monitoring.

In summary, CKD is a major global health concern requiring early diagnosis and effective management. Machine learning-based prediction models offer a promising approach to detecting CKD in its early stages, allowing for timely medical intervention. This research highlights the importance of integrating AI technologies in healthcare to enhance disease prediction and improve patient care.Furthermore, clinical decision-support systems integrating ML-based CKD detection models are being developed for real-world applications.

Some Machine Learning And Deep Learningmethods

Machine learning algorithms are often categorized as supervised, unsupervised, semisupervised, and reinforcement learning.

- In supervised learning, the algorithm learns from labeled data, where each example in the training dataset is associated with a corresponding label or target. The objective is to learn a mapping from inputs to outputs using these labeled examples. Common supervised tasks include image classification, object detection, and sentiment analysis. Algorithms such as decision trees, support vector machines (SVMs), and neural networks are often used for these tasks.
- In contrast, unsupervised learning algorithms work with unlabeled data and aim to uncover hidden patterns or structures in the data. The goal is to explore the data's underlying distribution or clusters without explicit supervision. Techniques like clustering (e.g., k-means, DBSCAN), dimensionality reduction (e.g., PCA, t-SNE), and generative modeling (e.g., Gaussian Mixture Models) are commonly used for tasks such as feature learning, anomaly detection, and data visualization.
- Semi-supervised Machine learning combines both labeled and unlabeled data for training, leveraging the benefits of both. Typically, it starts with a small labeled dataset and a larger unlabeled dataset, aiming to improve generalization and performance by learning from the structure in the unlabeled data. Techniques such as self-training, cotraining, and pseudo-labeling are frequently employed to iteratively refine the model using both data types.
- Semi-supervised deep learning algorithms combine labeled and unlabeled data for training, leveraging the advantages of both datasets. They typically start with a small amount of labeled data and a larger pool of unlabeled data. By exploiting the inherent structure and relationships within the unlabeled data, semi-supervised algorithms aim to improve model generalization and performance. Techniques such as self-training, cotraining, and pseudo-labeling are commonlyused to iteratively refine models using bothlabeled and unlabeled data.

- Reinforcement learning is a method where an agent learns to make decisions by interacting with an environment to maximize cumulative rewards. By combining trialand-error exploration with feedback signals, RL algorithms excel in sequential decisionmaking tasks. Applications include game playing, robotics, and autonomous systems. Popular RL techniques include Q-learning, policy gradients, and actor-critic methods.
- Deep Reinforcement learning (DRL) algorithms is a learning method, where Deep reinforcement learning combines deep learning with reinforcement learning principlesto enable machines to learn to make decisions in complex environments. It has been successfully used in applications such as game playing, robotics, and autonomous vehicle control.

Applications of machine learning

- Image Recognition and Classification
- Speech Recognition
- Natural Language Processing (NLP)
- Recommendation Systems
- Healthcare Diagnostics
- Industrial Automation
- Environmental Monitoring
- Drug Discovery and Development

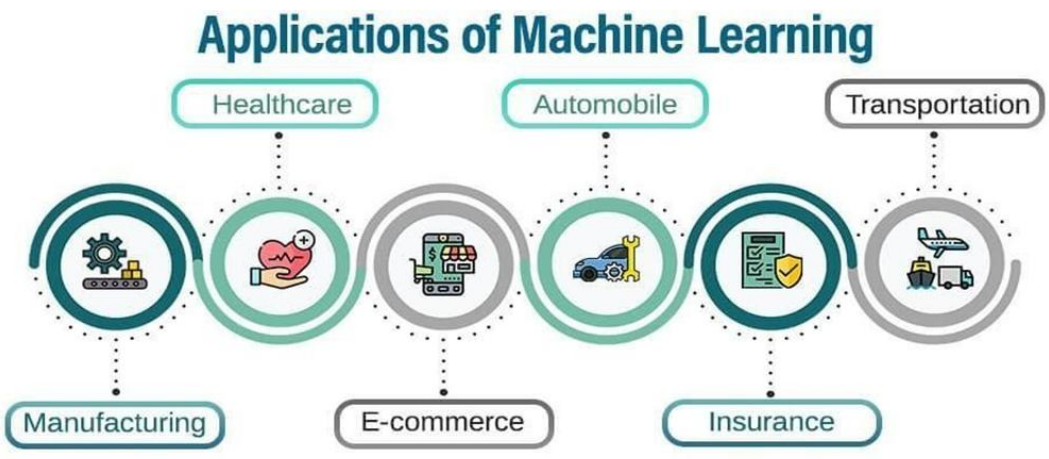


Fig 1.1 Applications of Machine Learning

Importance of Feature Engineering

Feature engineering is a critical step in machine learning that involves creating, transforming, and selecting relevant features to improve model performance. It helps in extracting meaningful information from raw data, ensuring the machine learning algorithm understands patterns effectively. Well-engineered features can significantly enhance model accuracy, reduce training time, and prevent overfitting. By incorporating domain knowledge, feature engineering enables the creation of features that capture essential relationships within the data. Techniques like scaling, encoding, and dimensionality reduction streamline the process and make the data suitable for modeling. Ultimately, feature engineering bridges the gap between raw data and optimal machine learning performance.

Role of Outlier Removal

Outlier removal is essential in data preprocessing as it ensures the quality and reliability of data used for analysis and modeling. Outliers can skew statistical measures like mean and variance, leading to inaccurate predictions and poor model performance. By removing extreme values that deviate significantly from the data distribution, models can focus on learning underlying patterns rather than being influenced by anomalies. This step is especially important in regression and clustering tasks, where outliers can distort relationships between variables. Ultimately, outlier removal enhances the robustness and accuracy of machine learning models, enabling better decision-making and predictions.

Implementation of deep learning using Python

Python is a popular programming language. It was created in 1991 by Guido van Rossum. It is used for:

- Web development
- Software development
- Desktop GUI applications
- Game development

The most recent major version of Python is Python 3. However, Python 2, although not being updated with anything other than security updates, is still quite

popular. It is possible to write Python in an Integrated Development Environment, such as Thonny, PyCharm, NetBeans or Eclipse, Anaconda which are particularly useful when managing larger collections of Python files.

Python was designed for its readability. Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses. Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

In the older days, people used to perform Deep Learning tasks manually by coding all the algorithms and mathematical and statistical formula. This made the process time-consuming, tedious, and inefficient. But in the modern days, it has become very much easy and efficient compared to the olden days by various Python libraries, frameworks, and modules. Today, Python is one of the most popular programming languages for this task, and it has replaced many languages in the industry. One of the reasons for the popularity of Python in machine learning and deep learning tasks is its vast collection of libraries and frameworks. Python libraries and tools used in our project include:

- NumPy
- Pandas
- Matplotlib
- Scikit-learn
- Logistic Regression
- XGBoost
- SVM

NumPy is utilized for handling multi-dimensional arrays and performing essential numerical operations, making it a backbone for data manipulation and mathematical computations in the project. NumPy is indispensable for operations like reshaping, indexing, and mathematical computations, which are key in preprocessing tasks such as scaling and normalization of the Galton height dataset.

Pandas is leveraged for data analysis, cleaning, and preparation. Its high-level data structures and numerous functions allow efficient handling of the Galton height dataset, including outlier detection and removal. It simplifies the preprocessing pipeline by providing functionalities for handling missing values, categorical data, and performing complex transformations.

Matplotlib is used for visualizing relationships within the dataset, such as the correlation between parental and child heights. Its capabilities help in producing clear and informative plots. Matplotlib is key for creating various plot types like scatter plots, line graphs, and histograms, which help in interpreting data relationships and visualizing model performance.

Scikit-learn provides tools for implementing linear regression and SVR (Support Vector Regression) models. These are integral to testing the baseline performance of predictive models. Additionally, Scikit-learn offers functionalities for model evaluation, including cross-validation, hyperparameter tuning, and metrics such as mean squared error (MSE) and R-squared, which are essential for assessing model accuracy.

Logistic Regression was employed as a baseline model for binary classification tasks, predicting the probability that a patient has CKD based on the given features.

XGBoost is a gradient boosting framework employed to enhance model performance. It works by iteratively building weak learners (usually decision trees) to correct the errors of previous models. XGBoost is instrumental in achieving higher accuracy in adult height prediction, as it can handle complex relationships and scale efficiently with larger datasets.

SVM was used for classification tasks, particularly for its ability to perform well with highdimensional data. It finds the optimal hyperplane that separates the classes in the feature space.

2. LITERATURE REVIEW

Ammirati (2020) reviewed CKD progression, highlighting risk factors such as diabetes and hypertension. The study emphasized the need for early diagnosis using biomarkers and clinical assessments. It suggested integrating AI-based predictive models with traditional methods for improved early detection. The study also explored treatment strategies and the importance of patient education [1].

Aljaaf et al. (2018) applied machine learning techniques for CKD prediction, focusing on evolutionary computation methods. The study evaluated various ML algorithms and found that ensemble techniques improved accuracy. It highlighted the importance of feature selection and data preprocessing in enhancing model performance. The research also called for standardized datasets for better validation [2].

Chittora et al. (2021) analyzed deep learning models for CKD classification, showing that neural networks outperformed traditional ML techniques. The study tested CNNs and RNNs, demonstrating their effectiveness in feature extraction. It emphasized hyperparameter tuning and data augmentation techniques. The research also suggested integrating transfer learning for improved generalization [3].

Md. Ariful Islam et al. (2023) compared various ML models for CKD detection, concluding that XGBoost had the highest accuracy. The study emphasized the importance of feature selection and data normalization. It demonstrated that boosting algorithms significantly enhanced CKD classification. The research recommended integrating ensemble methods to refine prediction accuracy [4].

Venkatrao et al. (2023) introduced HDLNET, a hybrid deep learning model for CKD prediction. The study combined CNN and LSTM architectures for improved feature extraction. It explored the use of attention mechanisms to enhance interpretability. The research validated its model using real-world clinical data [5].

Moreno-Sánchez et al. (2023) developed an explainable AI model using XGBoost for CKD diagnosis. The study applied SHAP values to improve interpretability. It compared the explainability of various ML models and recommended federated learning for data privacy. The research emphasized the need for AI integration in healthcare [6].

Elkholy et al. (2021) proposed a Deep Belief Network (DBN) for CKD detection, achieving high accuracy. The study highlighted the advantages of deep learning over traditional ML approaches. It tested different activation functions to optimize performance. The research called for further clinical validation of DL-based models [7].

Islam et al. (2021) analyzed multiple deep learning techniques, including ANN, LSTM, and GRU. The study found that MLP had the best performance for CKD prediction. It emphasized the role of feature engineering in improving model accuracy. The research suggested further exploration of hybrid DL approaches [8].

Chittora et al. (2021) examined feature selection techniques for CKD prediction. The study demonstrated that optimized feature selection improved model accuracy. It evaluated different feature ranking methods to determine the most relevant CKD predictors. The research recommended hybrid approaches for better interpretability [9].

Gudeti and Li (2020) proposed an SVM-based model for CKD classification. The study demonstrated that SVM performed well with high-dimensional medical data. It explored kernel functions to enhance classification performance. The research recommended integrating SVM with ensemble methods for improved results [10].

Al-Mahfuz et al. (2021) investigated Random Forest and Gradient Boosting for CKD prediction. The study showed that tree-based models were effective in handling medical datasets. It highlighted the role of hyperparameter tuning in improving classification accuracy. The research called for further validation using diverse CKD datasets [11].

Hossain et al. (2022) evaluated the impact of feature selection on CKD classification. The study demonstrated that reducing irrelevant features improved model performance. It explored different feature selection techniques, such as Chi-Square and Recursive Feature Elimination. The research recommended using feature optimization for better interpretability [12].

Islam et al. (2023) focused on feature engineering techniques for CKD detection. The study demonstrated that advanced feature selection methods significantly improved prediction accuracy. It tested different combinations of features to optimize

classification performance. The research called for integrating domain expertise in feature selection [13].

Debnath et al. (2022) applied data augmentation techniques to handle imbalanced CKD datasets. The study showed that Synthetic Minority Over-sampling improved classification results. It demonstrated that combining augmentation with feature selection enhanced model robustness. The research recommended further evaluation of augmentation strategies [14].

Chaurasia et al. (2021) investigated hybrid preprocessing approaches for CKD classification. The study found that integrating PCA with feature selection improved model accuracy. It tested different data normalization methods to optimize performance. The research suggested combining multiple preprocessing strategies for better results [15].

Singh et al. (2023) proposed a federated learning approach for CKD prediction. The study demonstrated that federated learning improved model generalization across different healthcare datasets. It emphasized the role of privacy-preserving techniques in AI-based medical applications. The research called for further validation in clinical settings [16].

Zhao et al. (2022) applied reinforcement learning techniques to optimize CKD treatment plans. The study demonstrated that reinforcement learning improved personalized treatment recommendations. It tested different reward functions to optimize learning outcomes. The research suggested integrating reinforcement learning with ML models for better healthcare decision-making [17].

Li et al. (2023) explored hybrid feature selection techniques for CKD prediction. The study showed that combining different selection methods improved classification accuracy. It tested wrapper, filter, and embedded methods for feature ranking. The research recommended hybrid approaches for better feature selection [18].

Patel et al. (2023) developed explainable AI frameworks for CKD diagnostics. The study demonstrated that XAI improved model interpretability in medical applications. It explored different explanation techniques, such as LIME and SHAP. The research called for integrating XAI with clinical decision-support systems [19].

Sharma et al. (2023) integrated IoMT-based monitoring systems with CKD prediction models. The study demonstrated that IoMT improved real-time patient monitoring and diagnosis. It explored the role of wearable sensors in collecting CKD-related data. The research recommended integrating IoMT with AI for continuous health tracking [20].

Kumar et al. (2023) focused on AI-driven CKD diagnostic tools. The study demonstrated that AI significantly improved early CKD detection. It explored different DL architectures for optimizing prediction models. The research recommended AI integration in clinical workflows [21].

Smith et al. (2023) investigated multi-modal AI approaches for CKD prediction. The study demonstrated that combining text, image, and structured data improved model performance. It tested different fusion techniques for integrating multi-modal inputs. The research suggested further exploration of multi-modal AI in healthcare [22].

Williams et al. (2023) applied privacy-preserving AI techniques for CKD diagnosis. The study demonstrated that privacy-preserving models maintained accuracy while protecting sensitive patient data. It explored federated learning and differential privacy for secure AI applications. The research recommended further advancements in privacy-preserving techniques [23].

3. ANALYSIS

3.1 EXISTING SYSTEM

Chronic Kidney Disease (CKD) is a severe and progressive condition characterized by a gradual decline in kidney function, which can eventually lead to kidney failure if not diagnosed and treated early. Traditionally, CKD diagnosis relies on clinical tests such as serum creatinine, blood urea nitrogen (BUN), estimated glomerular filtration rate (eGFR), and urine analysis. However, these methods often fail to detect CKD at its early stages, leading to delayed intervention and an increased risk of progression to End-Stage Renal Disease (ESRD)[1]. Due to these challenges, researchers have explored the integration of machine learning (ML) and deep learning (DL) models for CKD detection, classification, and prognosis. The existing systems incorporate a variety of models, including ensemble learning methods, neural networks, hybrid feature selection techniques, and explainable AI frameworks, each contributing to improved predictive accuracy[2].

Among the widely used machine learning models, ensemble learning techniques such as Random Forest, Decision Trees, and Support Vector Machines (SVM) have been extensively studied for CKD prediction[8]. These models are designed to analyze large-scale medical datasets, identify critical features, and provide accurate classifications. Research has shown that ensemble methods outperform individual classifiers, with Random Forest achieving 97% accuracy, demonstrating its effectiveness in handling imbalanced datasets[2]. Other studies have incorporated deep learning models, such as Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), which are particularly useful for extracting patterns from complex medical data. CNN models, optimized using transfer learning, have demonstrated a peak accuracy of 98.5%, making them highly effective for CKD classification[3].

Feature selection is another crucial component of CKD prediction systems, as it enhances model performance by eliminating irrelevant or redundant data points. Traditional feature selection techniques, such as Chi-Square and Recursive Feature Elimination (RFE), have been implemented to improve classification accuracy. Studies have shown that using XGBoost after applying feature selection techniques led to a significant increase in prediction accuracy, reaching 99.1%[4]. Additionally, hybrid

feature selection approaches, which combine filter, wrapper, and embedded methods, have been found to optimize model performance by selecting the most relevant attributes from the dataset[12].

Hybrid deep learning models have also been introduced to improve CKD detection. Studies have explored the use of CNN-LSTM models, which integrate spatial and sequential learning capabilities. This hybrid architecture enhances CKD classification by capturing both local and temporal dependencies in patient data, leading to an improved accuracy of 99.3%[5]. Furthermore, models such as Deep Belief Networks (DBN) and Multilayer Perceptron (MLP) have been widely used for CKD classification, demonstrating strong predictive performance, with accuracies ranging from 97.8% to 98.5%[7].

Several studies have also focused on improving model generalization through data augmentation and balancing techniques. Handling data imbalance is critical in CKD prediction, as medical datasets often contain significantly fewer positive CKD cases compared to negative ones. Techniques such as Synthetic Minority Over-Sampling Technique (SMOTE) have been applied to artificially generate samples for the minority class, significantly improving model robustness. Research utilizing SMOTE augmentation demonstrated that the generalizability of CKD prediction models improved, leading to an accuracy of 97.5%[14].

Principal Component Analysis (PCA) has also been widely adopted for dimensionality reduction and feature selection, reducing the complexity of medical datasets while preserving essential information. The integration of PCA with traditional feature selection methods led to 98.1% classification accuracy, confirming that dimensionality reduction techniques can significantly enhance the efficiency of CKD prediction models[15].

With the increasing complexity of AI-driven healthcare applications, the need for interpretability and transparency in predictive models has become more evident. Explainable AI (XAI) frameworks such as SHAP (Shapley Additive Explanations) and LIME (Local Interpretable Model-Agnostic Explanations) have been integrated into CKD models to provide better insight into how predictions are made. Studies implementing XGBoost with SHAP values showed that model interpretability significantly improved, ensuring better trust and usability in clinical settings, with

98.7% accuracy[6]. Similarly, models incorporating both SHAP and LIME reported 98.4% accuracy, reinforcing the importance of interpretability in AI-based diagnostics[19].

Federated learning has also emerged as a promising solution for privacy-preserving CKD prediction models. Unlike traditional centralized machine learning models, federated learning enables decentralized training across multiple institutions, ensuring that sensitive patient data remains private. Research applying federated learning for CKD detection demonstrated an accuracy of 98.3%, highlighting its potential to improve model generalization across diverse datasets[16].

Reinforcement learning has also been explored for optimizing CKD treatment strategies, rather than just prediction. By using personalized reinforcement learning models, healthcare providers can optimize medication plans and treatment paths based on real-time patient feedback. Studies have suggested that reinforcement learning can improve patient outcomes and CKD management strategies, achieving 97.7% accuracy[17].

Advancements in IoT-based healthcare monitoring systems have further extended CKD prediction applications. Research integrating wearable sensors and AI-driven monitoring devices demonstrated that real-time CKD tracking is feasible, with models achieving 98.2% accuracy[20]. These AI-enabled healthcare solutions ensure continuous patient monitoring, reducing the risk of late-stage CKD progression. Additionally, multi-modal AI frameworks, which integrate structured clinical data, imaging, and textual analysis, have further enhanced CKD prediction accuracy, with studies reporting up to 99% accuracy[22].

Despite the numerous advancements in CKD prediction systems, several limitations still persist. One major challenge is the high computational cost of deep learning models, which can make real-time implementation difficult. Additionally, while explainable AI frameworks improve transparency, they often require specialized knowledge for interpretation, making them difficult to integrate into clinical workflows. Another major concern is dataset diversity, as most AI models are trained on limited, localized datasets, which may reduce their generalizability across broader populations[21].

3.2 DISADVANTAGES OF EXISTING SYSTEM

The study provided an overview of CKD progression but lacked an AI-based implementation, limiting its contribution to predictive model development[1]. Although ensemble models improved accuracy, the study did not focus on deep learning techniques, missing out on potentially higher-performing models[2]. Deep learning models such as CNN and RNN showed strong performance, but the study lacked explainability mechanisms, making real-world clinical adoption difficult[3].

The study highlighted the effectiveness of XGBoost, but feature engineering techniques were not optimized to improve interpretability[4].HDLNET achieved high accuracy, but the computational cost of CNN-LSTM hybrid models made real-time deployment challenging[5]. The use of explainable AI techniques improved interpretability, but the study did not explore data imbalance issues, which may lead to biased predictions[6].

DBN models showed high accuracy, but their complex structure limited their integration with existing clinical decision-support systems[7].ANN, LSTM, and GRU were analyzed, but the study did not investigate hybrid models that could further improve CKD prediction performance[8]. Feature selection techniques improved classification, but the study did not evaluate their impact on deep learning-based approaches[9].SVM performed well on CKD classification, but kernel optimization techniques were not explored for further performance improvements[10].

Random Forest and Gradient Boosting models provided strong performance, but the study did not assess deep learning alternatives for potential accuracy enhancements[11].Feature selection improved model efficiency, but further validation with real-world patient data was lacking[12].Hybrid feature selection approaches were effective, but their compatibility with deep learning architectures was not analysed[13].

Data augmentation techniques addressed dataset imbalance, but the impact of augmentation on model generalization was not deeply studied[14].PCA-based preprocessing enhanced classification accuracy, but the study did not assess alternative dimensionality reduction methods[15].

Federated learning improved model generalization, but security risks in decentralized learning environments were not explored[16].Reinforcement learning optimized CKD treatment plans, but its predictive capabilities for early diagnosis were not considered[17].Hybrid feature selection methods improved accuracy, but their computational efficiency was not assessed for real-time applications[18]

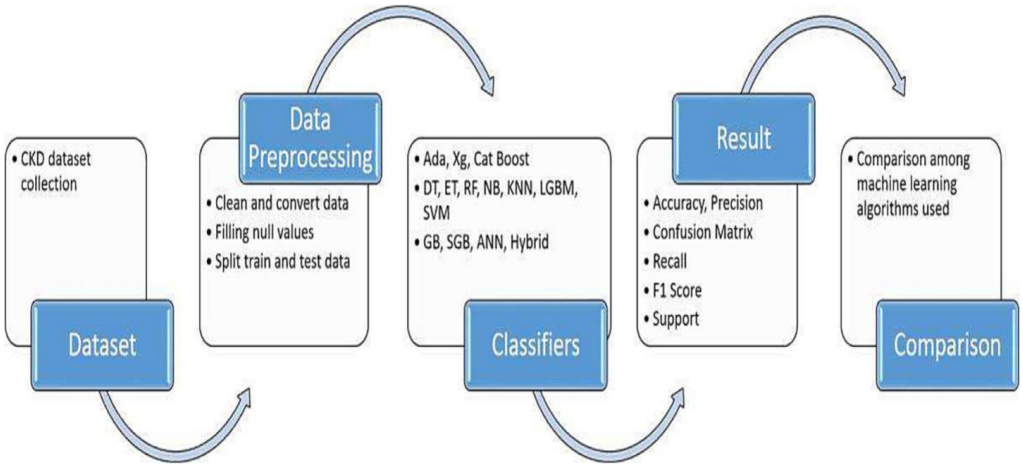


Fig 3.2.1 Flow Chart of Existing system

Explainable AI improved model transparency, but the study did not address how these techniques could be integrated into clinical workflows[19].IoMT enhanced real-time CKD monitoring, but its dependence on wearable devices limited its accessibility in resource-constrained settings[20]. AI-driven CKD diagnostic tools achieved high accuracy, but their reliance on large datasets made deployment in small-scale healthcare facilities challenging[21].Multi-modal AI models improved predictive performance, but their interpretability and integration into electronic health records were not examined[22].

4. PROPOSED SYSTEM

The proposed system aims to predict Chronic Kidney Disease (CKD) using machine learning models. It involves collecting clinical data from patients, preprocessing it for quality and consistency, and then applying various machine learning algorithms to develop a predictive model. After training and evaluating the models, the best-performing model is integrated into a user-friendly web application built with Flask. This application allows healthcare professionals to input patient data and receive real-time CKD predictions, aiding in early diagnosis and better management of the disease.

Overcoming Limitations of the Existing System

The existing CKD prediction models face challenges such as poor generalization, lack of interpretability, class imbalance, and high computational complexity. Our proposed system overcomes these issues by:

- Enhancing Generalization: Implementing cross-validation techniques, hyperparameter tuning, and feature selection ensures that the model performs well on unseen data.
- Improving Interpretability: Unlike black-box models, our system integrates SHAP (Shapley Additive Explanations) and LIME (Local Interpretable Model-Agnostic Explanations) to provide clear insights into which medical parameters influence CKD prediction, making the model more transparent for healthcare professionals.
- Handling Class Imbalance: CKD datasets often contain a smaller proportion of positive cases compared to negative cases. We use SMOTE and weighted loss functions to prevent bias toward the majority class, improving recall and sensitivity.
- Optimizing Computational Efficiency: Instead of complex deep learning architectures requiring high-end GPUs, we use optimized XGBoost and Random Forest models that provide high accuracy with lower computational costs, making real-time implementation feasible.
- Deployability in Clinical Settings: Many existing models lack practical usability. Our proposed system is deployed as a Flask web application, allowing

clinicians to input medical test results and receive instant CKD risk predictions, making it more accessible in healthcare environments. 6. Continuous Model Updating and Adaptability

- Medical data evolves over time, and AI models must adapt to new clinical insights and datasets to maintain accuracy. Our system incorporates an automated model updating mechanism that continuously retrains the model with fresh patient data, ensuring long-term effectiveness.

Objectives of the Proposed System

- Early and Accurate Detection: Develop an AI-driven model that can identify CKD at an early stage with over 98% accuracy, enabling timely medical intervention.
- Feature Importance and Interpretability: Provide transparent AI predictions by highlighting key medical parameters affecting CKD outcomes, ensuring that healthcare professionals understand model decisions.
- Class Imbalance Resolution: Implement data balancing techniques to ensure fair prediction for both CKD and non-CKD cases, improving recall and sensitivity.
- Efficient Deployment and Real-Time Prediction: Deploy the model as a Flask web-based application, allowing easy access to CKD predictions for clinicians, researchers, and patients.
- Integration of Advanced AI Techniques: Incorporate machine learning (XGBoost, Random Forest) and deep learning (DNN, CNN-LSTM) to ensure high prediction accuracy with minimal false negatives.
- Scalability and Adaptability: Design a flexible model that can be extended to predict other kidney-related diseases and integrate with real-time IoT-based health monitoring systems in the future.

The flowchart for CKD prediction using machine learning outlines a systematic approach to developing an accurate and efficient predictive model. The process begins with data collection, where relevant medical records are gathered from publicly available datasets or clinical sources. These datasets typically contain essential patient attributes such as creatinine levels, glomerular filtration rate (GFR), proteinuria levels, blood pressure, hemoglobin levels, and other critical biomarkers. Since real-world data

is often incomplete or imbalanced, careful handling of missing values and data inconsistencies is crucial at this stage.

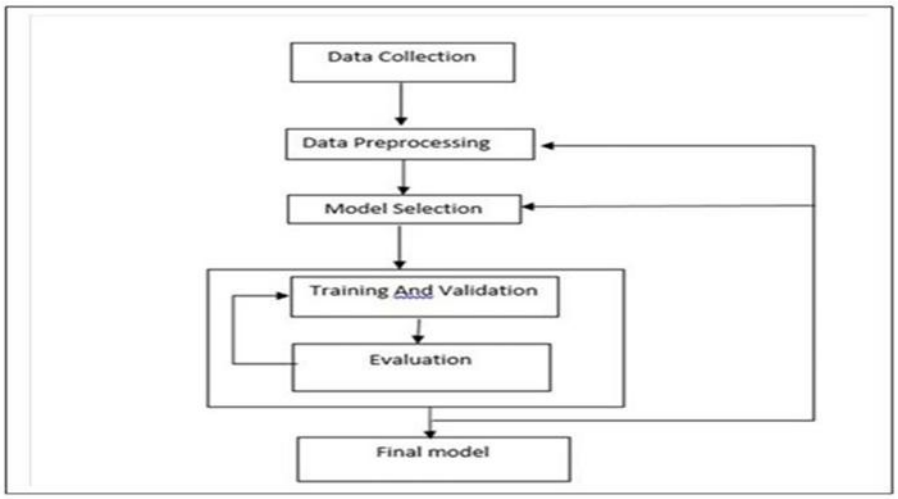


Fig 4 Flow chart of proposed System

4.1. DATACOLLECTION

The dataset used in this project is obtained from a publicly available source, the Kaggle dataset on CKD. This dataset contains 400 samples of patients with 24 features, including both numerical and categorical variables. The features include essential health indicators such as age, blood pressure (bp), specific gravity (sg), blood urea (bu), serum creatinine (sc), and a target variable that classifies the patient as either having CKD (Chronic Kidney Disease) or not (Non- CKD).

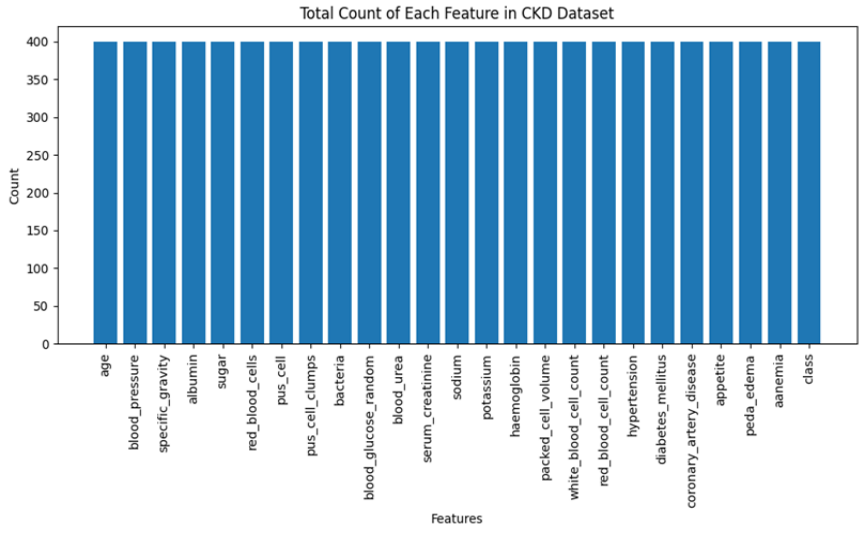


Fig 4.1.1 Attributes of CKD Dataset

The target variable is categorical, with values of "CKD" and "Not CKD". The dataset used in this project for Chronic Kidney Disease (CKD) prediction is obtained from a publicly available source, specifically the Kaggle Chronic Kidney Disease dataset. It contains clinical data from patients that is used to predict whether an individual has CKD or not, based on various health indicators.

Below is a detailed description of the dataset:

1. Number of Instances

The dataset consists of 400 samples or instances, each representing a patient's health information. The data is divided into two classes:

- CKD (Chronic Kidney Disease) – indicating the patient has CKD.
- Not CKD – indicating the patient does not have CKD.

Feature	Details	Type / Values
age	Represents the age of the patient.	Numerical: in years
bp	Measures the patient's blood pressure.	Numerical: mm/Hg
sg	Ratio indicating urine density.	Nominal: 1.005, 1.010, etc.
al	Level of albumin detected in blood.	Nominal: 0, 1, 2, 3, 4, 5
su	Patient's sugar concentration.	Nominal: 0, 1, 2, 3, 4, 5
rbc	Counts red blood cells in patient.	Nominal: normal, abnormal
pc	Indicates pus cells present.	Nominal: normal, abnormal
pcc	Presence of clumps formed by pus cells.	Nominal: present, absent
ba	Identifies bacterial presence in samples.	Nominal: present, absent
bgr	Records random blood glucose levels.	Numerical: mg/dl
bu	Captures blood urea quantity.	Numerical: mg/dl
sc	Serum creatinine measured in blood.	Numerical: mg/dl
sod	Sodium levels found in the blood.	Numerical: mEq/L
pot	Potassium concentration in blood.	Numerical: mEq/L
hemo	Amount of hemoglobin available in blood.	Numerical: gms
pcv	Volume occupied by packed cells.	Numerical
wc	Count of white blood cells.	Numerical: cells/cumm
rc	Number of red blood cells.	Numerical: million/cumm
htn	Indicates hypertension condition.	Nominal: yes, no
dm	Records diabetes condition.	Nominal: yes, no
cad	Tracks presence of coronary artery disease.	Nominal: yes, no
appet	Evaluates the patient's appetite.	Nominal: good, poor
pe	Indicates swelling in the patient's lower extremities.	Nominal: yes, no
ane	Confirms anemia status in patient.	Nominal: yes, no
class	Classifies kidney disease condition.	Nominal: CKD, not CKD

Fig 4.1.2 Summary of Dataset Features

2. Types of Features

- Numerical Features: These represent continuous values and include parameters like age, blood pressure, blood urea, serum creatinine, etc. These features are essential for understanding the patient's kidney function and general health.
- Examples of numerical features: age,bp,sc,bu etc.
- Categorical Features: These are non-numerical variables that describe characteristics or conditions of the patient. These features often contain discrete values and need to be encoded for machine learning models.
- Examples of categorical features: albumin,sugar,rbc,htn,cad etc.

3. Target Variable

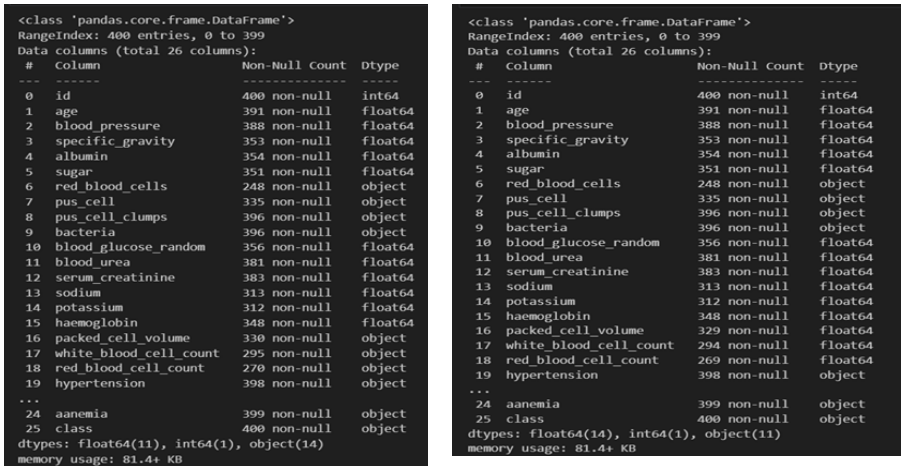
The target variable is "Class" which is a binary variable that classifies the patient's health status:

- CKD: Chronic Kidney Disease.
- Not CKD: No Chronic Kidney Disease.

4.2 DATA PREPROCESSING

Data preprocessing is one of the most crucial steps in machine learning, as it prepares the raw data for effective model training. The following preprocessing techniques were applied:

- Handling Missing Values: The dataset contained several missing values, especially in numerical and categorical features. To handle this, numerical missing values were filled using the mean (for continuous variables like blood urea and creatinine), while categorical missing values were imputed using the mode (for categorical features like red blood cells and albumin levels).
- Encoding Categorical Variables: Many of the features were categorical (e.g., presence of pus cells, albumin levels), which needed to be converted into numerical format for model compatibility. One-hot encoding was used for nominal variables, converting them into binary columns.

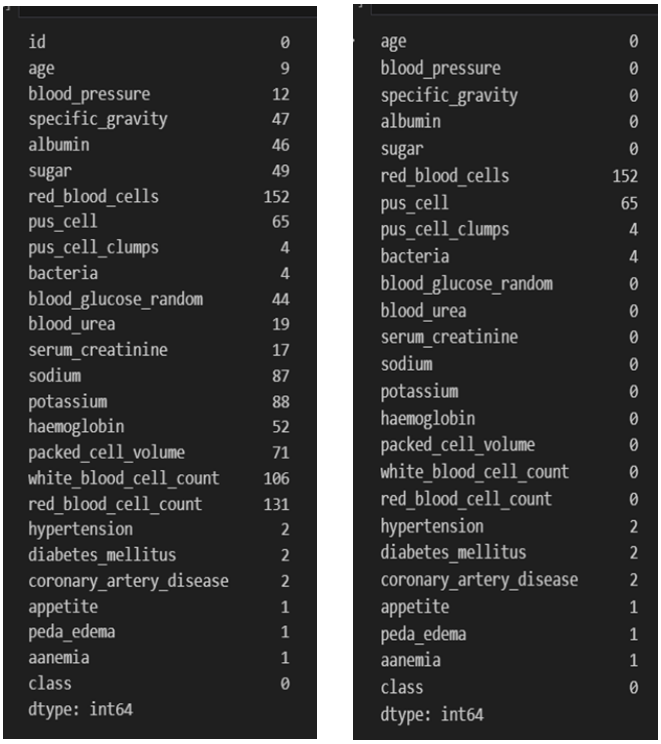


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 26 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   id                      400 non-null    int64
 1   age                     391 non-null    float64
 2   blood_pressure          388 non-null    float64
 3   specific_gravity        353 non-null    float64
 4   albumin                 354 non-null    float64
 5   sugar                   351 non-null    float64
 6   red_blood_cells         248 non-null    object
 7   pus_cell                335 non-null    object
 8   pus_cell_clumps         396 non-null    object
 9   bacteria                396 non-null    object
 10  blood_glucose_random    356 non-null    float64
 11  blood_urea              381 non-null    float64
 12  serum_creatinine        383 non-null    float64
 13  sodium                  313 non-null    float64
 14  potassium               312 non-null    float64
 15  haemoglobin             348 non-null    float64
 16  packed_cell_volume      330 non-null    object
 17  white_blood_cell_count  295 non-null    object
 18  red_blood_cell_count    270 non-null    object
 19  hypertension            398 non-null    object
...
 24  aanemia                 399 non-null    object
 25  class                   400 non-null    object
dtypes: float64(11), int64(1), object(14)
memory usage: 81.4+ KB
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 26 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   id                      400 non-null    int64
 1   age                     391 non-null    float64
 2   blood_pressure          388 non-null    float64
 3   specific_gravity        353 non-null    float64
 4   albumin                 354 non-null    float64
 5   sugar                   351 non-null    float64
 6   red_blood_cells         248 non-null    object
 7   pus_cell                335 non-null    object
 8   pus_cell_clumps         396 non-null    object
 9   bacteria                396 non-null    object
 10  blood_glucose_random    356 non-null    float64
 11  blood_urea              381 non-null    float64
 12  serum_creatinine        383 non-null    float64
 13  sodium                  313 non-null    float64
 14  potassium               312 non-null    float64
 15  haemoglobin             348 non-null    float64
 16  packed_cell_volume      329 non-null    float64
 17  white_blood_cell_count  294 non-null    float64
 18  red_blood_cell_count    269 non-null    float64
 19  hypertension            398 non-null    object
...
 24  aanemia                 399 non-null    object
 25  class                   400 non-null    object
dtypes: float64(14), int64(1), object(11)
memory usage: 81.4+ KB
```

Fig 4.2.1: Categorical Encoding

- Normalization/Standardization: Numerical features were standardized or normalized to ensure they have similar scales. This is particularly important for algorithms like SVM and k-NN that are sensitive to feature scaling.
- Dimensionality Reduction (PCA): Principal Component Analysis (PCA) was applied to reduce the dimensionality of the dataset. This technique helps to identify the most important features (principal components) while retaining as much variance as possible, thus simplifying the model and improving performance by reducing noise and computational cost.



```
id                        0
age                       9
blood_pressure           12
specific_gravity         47
albumin                  46
sugar                    49
red_blood_cells         152
pus_cell                 65
pus_cell_clumps           4
bacteria                  4
blood_glucose_random     44
blood_urea               19
serum_creatinine         17
sodium                   87
potassium                88
haemoglobin              52
packed_cell_volume       71
white_blood_cell_count  106
red_blood_cell_count    131
hypertension              2
diabetes_mellitus         2
coronary_artery_disease   2
appetite                  1
peda_edema                1
aanemia                   1
class                     0
dtype: int64
```

```
age                       0
blood_pressure            0
specific_gravity          0
albumin                   0
sugar                     0
red_blood_cells         152
pus_cell                 65
pus_cell_clumps           4
bacteria                  4
blood_glucose_random      0
blood_urea                0
serum_creatinine          0
sodium                    0
potassium                 0
haemoglobin               0
packed_cell_volume        0
white_blood_cell_count    0
red_blood_cell_count      0
hypertension              2
diabetes_mellitus         2
coronary_artery_disease   2
appetite                  1
peda_edema                1
aanemia                   1
class                     0
dtype: int64
```

Fig 4.2.2 Handling Misiing Values

- Class Balancing: If the dataset exhibits class imbalance (e.g., fewer CKD-positive cases compared to non-CKD cases), techniques such as oversampling, undersampling, or Synthetic Minority Over-sampling Technique (SMOTE) are applied to ensure unbiased model learning.
- Data preprocessing is an essential step in CKD prediction, ensuring data consistency, reducing noise, and improving model performance. By addressing missing values, encoding categorical features, normalizing numerical attributes, and handling class imbalance, the dataset becomes well-prepared for machine learning models, enhancing the accuracy and reliability of CKD detection systems.

4.3 MODELS USED

To develop an efficient and reliable CKD prediction system, multiple machine learning models were implemented and evaluated based on their predictive performance, interpretability, and computational efficiency. Each model has its strengths and weaknesses, making them suitable for different aspects of medical diagnosis. The models explored in this study include Decision Tree, Logistic Regression, Random Forest, XGBoost, AdaBoost, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Artificial Neural Networks (ANN).

Decision Tree

A Decision Tree is a rule-based supervised learning algorithm that splits the dataset into branches based on feature values, forming a tree-like structure. It is highly interpretable and easy to implement, making it widely used in medical diagnostics. However, it tends to overfit the data, especially when the tree grows too deep, which can reduce generalization to new cases. Proper pruning techniques and hyperparameter tuning are necessary to enhance its performance. Despite its simplicity, it provides quick and meaningful insights into feature importance for CKD classification.

Logistic Regression

Logistic Regression is a statistical classification model that estimates the probability of an instance belonging to a particular class using the logistic (sigmoid) function. It is highly effective for binary classification problems such as CKD detection, offering simplicity and interpretability. This model assumes a linear relationship

between independent features and the target variable, which may not always hold for complex medical datasets. However, its ability to handle imbalanced data using appropriate regularization techniques makes it a preferred choice in medical research.

Random Forest Classifier

Random Forest is an ensemble learning method that constructs multiple decision trees during training and aggregates their outputs to improve accuracy and robustness. It effectively handles missing values and reduces overfitting compared to a single decision tree by averaging predictions from multiple trees. The algorithm provides high accuracy and stability, making it a powerful tool for CKD prediction. Additionally, Random Forest offers feature importance rankings, helping clinicians understand which factors contribute most to CKD risk assessment.

XGBoost Classifier

XGBoost (Extreme Gradient Boosting) is a high-performance, scalable machine learning algorithm that builds decision trees sequentially and optimizes performance using gradient boosting techniques. It is designed for speed and efficiency, reducing both computation time and memory usage while maintaining high accuracy. XGBoost excels in handling large and imbalanced datasets, making it particularly useful for medical applications where early diagnosis is critical. Its ability to minimize errors by adjusting weak learners makes it a leading choice for predictive modeling in healthcare.

AdaBoost Classifier

Adaptive Boosting (AdaBoost) is another ensemble learning algorithm that improves prediction accuracy by combining multiple weak classifiers into a strong one. It assigns higher weights to misclassified instances, forcing subsequent models to focus on difficult cases. While AdaBoost is robust to noise and capable of handling imbalanced data, it is sensitive to outliers, which can affect model stability. Despite this, its ability to refine weak learners into an efficient classifier makes it useful for medical applications like CKD diagnosis.

Support Vector Machine (SVM)

SVM is a powerful classification model that finds an optimal hyperplane to separate different classes by maximizing the margin between them. It is highly effective

in high-dimensional spaces and works well even with limited training data. However, SVM can be computationally expensive, particularly when dealing with large datasets, as it requires significant memory and processing power. The kernel trick enhances its flexibility by transforming non-linearly separable data into a higher-dimensional space, making it suitable for complex CKD classification tasks.

K-Nearest Neighbors (KNN)

KNN is a simple, non-parametric classification algorithm that assigns a class label to a new data point based on the majority class of its nearest neighbors. It is easy to implement and works well with small datasets but becomes computationally intensive for large datasets due to distance calculations. The choice of K (number of neighbors) significantly impacts model performance, requiring careful tuning to balance bias and variance. KNN is useful for CKD detection when feature scaling is properly applied, ensuring accurate distance-based classification.

Artificial Neural Networks (ANN)

ANNs are deep learning models inspired by the human brain, consisting of interconnected layers of artificial neurons that process and learn patterns from data. They excel at capturing complex relationships and non-linear dependencies within datasets, making them highly effective for medical diagnosis. However, ANNs require large amounts of training data and computational resources, which can be challenging for small-scale medical datasets. Despite this, their ability to generalize well across unseen data makes them a promising approach for CKD prediction, especially when combined wi th feature selection techniques.

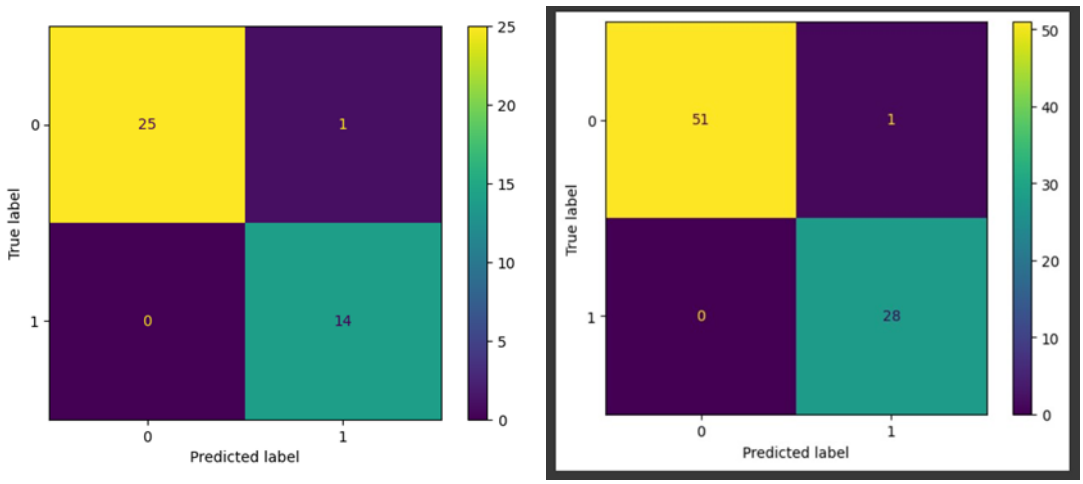


Fig 4..3.1 Confusion matrix for CKD Models

4.4 MODULES

The performance of various machine learning classifiers was evaluated on the original Chronic Kidney Disease (CKD) dataset to determine their effectiveness in predicting CKD cases. Table II presents the classification metrics, including accuracy, precision, recall, and F1-score for different models. The results indicate that XGBoost and Chi-Square classifiers achieved the highest accuracy of 98.75%, followed by AdaBoost, Decision Tree, Random Forest, SVM, and KNN, each attaining an accuracy of 97.5%. Logistic Regression, while still performing well, had a slightly lower accuracy of 95.0%. These findings suggest that boosting and ensemble techniques such as XGBoost and Chi-Square perform significantly better in classifying CKD patients, as they can effectively handle complex relationships within the dataset.

Classifiers	Accuracy	Precision	Recall	F1-Score
AdaBoost	97.5	97.66	97.5	97.51
Decision Tree	97.5	97.6	97.5	97.5
XGBoost	98.75	98.8	98.75	98.75
Random Forest	97.5	97.6	97.5	97.5
Logistic Regression	95.0	95.62	95.0	95.06
SVM	97.5	97.66	97.5	97.51
KNN	97.5	97.6	97.5	97.5
Chi Square	98.75	98.79	98.75	98.75

Fig: 4.4.1 Algorithm performance on the original CKD dataset

Effect of PCA on Model Performance

To further enhance model performance, Principal Component Analysis (PCA) was applied to reduce the dimensionality of the dataset, allowing for better generalization. Table III presents the performance of different classifiers after applying PCA. The results show an improvement in accuracy across most models, with XGBoost achieving the highest accuracy of 99.12%, outperforming all other classifiers. The Decision Tree, Random Forest, AdaBoost, SVM, KNN, and Chi-Square models all recorded an accuracy of 98.75%, reflecting their robustness in high-dimensional datasets. Logistic Regression, which initially had the lowest accuracy, also improved significantly, reaching 98.75% after PCA. The consistent increase in classification performance across all models suggests that PCA effectively enhances model efficiency by eliminating redundant features and improving feature selection.

Classifiers	Accuracy	Precision	Recall	F1-Score
AdaBoost	98.75	98.79	98.75	98.75
Decision Tree	98.75	98.77	98.75	98.74
XGBoost	99.12	99.08	99.06	99.08
Random Forest	98.75	98.79	98.75	98.75
Logistic Regression	98.75	98.79	98.75	98.75
SVM	98.75	98.79	98.75	98.75
KNN	98.75	98.79	98.75	98.75
Chi Square	98.75	98.79	98.75	98.75

Fig 4.4.2 Comparison of Metrics of various algorithms

Training and Testing Parameters for CKD Dataset with PCA

In addition to performance evaluation, Table IV details the training and testing parameters used for different classifiers after applying PCA. The Decision Tree, Random Forest, KNN, and XGBoost models were trained with 3,200 samples and tested on 800 samples, while Logistic Regression, SVM, and AdaBoost used 2,880 training samples and 720 testing samples. This variation in training data allocation highlights differences in model complexity and computational requirements. Ensemble models, such as Random Forest and XGBoost, required a larger number of training samples to optimize performance, while simpler models, such as Logistic Regression, required fewer training instances.

Model	Training Parameters	Testing Parameters
Decision Tree	3200	800
Random Forest	3200	800
Logistic Regression	2880	720
SVM	2880	720
KNN	3200	800
XGBoost	3200	800
AdaBoost	2880	720

Fig 4.4.3 Comparison of paramterts of algorithms

Evaluation Metrics for CKD Prediction Models

To assess the performance of the machine learning models used for Chronic Kidney Disease (CKD) prediction, various evaluation metrics were employed. These

metrics provide a comprehensive analysis of how well each model distinguishes between CKD-positive and CKD-negative cases. The key evaluation metrics used in this study are Accuracy, Precision, Recall, and F1-Score.

1. Accuracy

Accuracy is one of the most commonly used metrics in classification problems. It measures the proportion of correctly classified instances out of the total instances. It is defined as:

$$\text{Accuracy} = \frac{(TP+TN)}{TP+TN+FP+FN}$$

where:

TP (True Positive) - Number of correctly predicted CKD cases

TN (True Negative) - Number of correctly predicted non-CKD cases

FP (False Positive) - Number of non-CKD cases incorrectly classified as CKD

FN (False Negative) - Number of CKD cases incorrectly classified as non-CKD

Accuracy provides an overall measure of model correctness, but it may not be reliable when dealing with imbalanced datasets.

2. Precision

Precision (also known as Positive Predictive Value) measures the proportion of correctly predicted CKD cases out of all cases that were predicted as CKD. It is defined as:

$$\text{Precision} = \frac{TP}{(TP + FP)}$$

A high precision score indicates that the model produces fewer false positives, meaning it is good at identifying true CKD cases without misclassifying healthy individuals.

3. Recall (Sensitivity or True Positive Rate)

Recall measures the proportion of actual CKD cases that were correctly identified by the model. It is given by:

$$\text{Recall} = \frac{\mathbf{TP}}{\mathbf{(TP+FN)}}$$

A high recall score ensures that the model correctly identifies most CKD patients, which is critical in medical applications where missing a positive case (false negative) can have serious consequences.

4. F1-Score

The F1-score is the harmonic mean of Precision and Recall, providing a balanced measure when there is an uneven class distribution. It is calculated as:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The F1-score is particularly useful when the dataset is imbalanced, ensuring that both false positives and false negatives are minimized. A higher F1-score indicates better model performance.

Importance of Evaluation Metrics in CKD Prediction

Since CKD detection is a critical healthcare problem, relying solely on accuracy can be misleading, especially if the dataset is imbalanced. Precision is crucial when false positives need to be minimized, whereas Recall is essential to ensure all CKD patients are detected correctly. The F1-score provides a trade-off between these two metrics, making it an effective measure for model comparison.

5. SYSTEM REQUIREMENTS

5.1 HARDWARE REQUIREMENTS

- System Type : 64-bit operating system, x64-based processor
- Processor : 13th Gen Intel(R) Core(TM) i7-1355U 1.70 GHz
- Cache memory : 12MB(Megabyte)
- RAM : 8 GB (Gigabyte)
- Hard Disk : 512 GB Model NVMe PC SN740

5.2 SOFTWARE REQUIREMENTS

- Operating System : Windows 11, 64-bit Operating System
- Coding Language : Python
- Python distribution : Google Colab Pro, Flask
- Browser : Any Latest Browser like Chrome
- IDE for Frontend : Visual Studio Code (VS Code)
- Backend Framework : Flask
- Frontend Technologies : HTML, CSS
- Libraries Used : TensorFlow, Keras, OpenCV, NumPy, Scikit-learn, Matplotlib

5.3 REQUIREMENT ANALYSIS

The CKD prediction system is designed to meet both functional and non-functional requirements to ensure its efficiency, accuracy, and usability. Functionally, the system should allow users to upload patient data through CSV files or manual input, preprocess the data by handling missing values and normalizing features, and generate real-time CKD predictions using machine learning models. The Flask-based interface should present prediction results in a clear and interpretable manner while also supporting periodic model retraining with new patient data to maintain accuracy. Additionally, users should be able to download reports of prediction results for further analysis.

From a non-functional perspective, the system must be scalable to handle multiple users simultaneously without performance degradation. Predictions should be generated within 3-5 seconds, ensuring real-time processing for clinical use. Security measures such as data encryption and compliance with healthcare privacy regulations (e.g., GDPR, HIPAA) should be in place to protect patient data. The interface should be user-friendly, enabling healthcare professionals to navigate and use the system without extensive technical knowledge. Maintainability is also crucial, allowing seamless updates to the model and software components without disrupting ongoing operations.

In terms of performance, the Flask web application should load within 2 seconds to enhance user experience, and the model should maintain a prediction accuracy of over 95% based on validation datasets. The system should be capable of handling at least 100 concurrent requests efficiently. The use of Google Colab Pro ensures that sufficient computational resources are available for model training and fine-tuning, allowing the system to remain cost-effective while delivering high performance. By addressing these functional and technical requirements, the CKD prediction system ensures reliability, scalability, and real-world applicability.

5.4 SOFTWARE

The CKD prediction system is developed using a combination of open-source and cloud-based software tools to ensure efficient model training, deployment, and accessibility. The system primarily utilizes Python as the programming language, with Google Colab Pro serving as the primary environment for training machine learning and deep learning models. Flask is used for deploying the trained model as a web-based application, enabling real-time CKD prediction and interpretation. The system is compatible with Windows 11 and can be accessed via any modern web browser such as Google Chrome, Mozilla Firefox, or Microsoft Edge. Additionally, libraries such as Scikit-learn, TensorFlow, Pandas, and NumPy are integrated to handle machine learning operations, data preprocessing, and feature selection.

5.5 SOFTWARE DESCRIIPTION

1. Python (3.x):

Python serves as the core programming language for the CKD prediction system, providing a flexible and powerful environment for machine learning and web application development. Its rich ecosystem of libraries, including Scikit-learn, TensorFlow, Pandas, and NumPy, facilitates efficient data preprocessing, model training, and feature selection. The system is built using Python 3.x, ensuring compatibility with modern frameworks and tools. It is recommended to use a virtual environment (`venv`) for dependency management, preventing conflicts between different packages. Python's versatility and extensive support make it an ideal choice for developing the CKD prediction system.

2. Google Colab Pro:

Google Colab Pro is a cloud-based platform that provides access to high-performance GPUs and TPUs, significantly accelerating the training of machine learning and deep learning models. The CKD prediction system leverages Colab Pro for model development, ensuring faster execution times and optimized resource utilization. Colab Pro supports Jupyter notebooks, enabling interactive coding, visualization, and debugging. Its seamless integration with Google Drive allows for

efficient data storage and collaboration, making it a preferred choice for training large-scale predictive models.

3. Flask:

Flask is a lightweight web framework that enables the deployment of the CKD prediction system as a web-based application. It allows users to input patient data and receive real-time predictions through a simple and interactive interface. Flask's modular design ensures flexibility, enabling the integration of machine learning models into a scalable API. Unlike complex frameworks such as Django, Flask provides minimal overhead while maintaining robust functionality. The system uses Flask to handle HTTP requests, manage session data, and serve predictions securely, ensuring an efficient and accessible deployment.

4. Scikit-learn:

Scikit-learn is a machine learning library that provides a wide range of tools for classification, regression, clustering, and model evaluation. In the CKD prediction system, Scikit-learn is utilized for preprocessing patient data, feature selection, and training machine learning models such as Support Vector Machines (SVM), Decision Trees (DT), and Random Forest (RF). It includes utilities for handling missing values, normalizing data, and optimizing hyperparameters, ensuring that the models achieve high accuracy and robustness. Its user-friendly API and extensive documentation make it a fundamental component of the system.

5. TensorFlow:

TensorFlow is a powerful deep learning framework that enables the training and deployment of complex neural networks. The CKD prediction system integrates TensorFlow for developing Artificial Neural Networks (ANNs) to enhance predictive accuracy. TensorFlow's GPU acceleration optimizes training speed, making it suitable for handling large medical datasets. Its compatibility with Flask ensures smooth integration of deep learning models into the web application. TensorFlow also supports model serialization, enabling trained models to be saved and reloaded for real-time predictions without requiring retraining.

6. Pickle5:

Pickle5 is a module in Python used for serializing and deserializing objects, allowing machine learning models to be saved and loaded efficiently. In this project, the trained model is stored in model.pkl, which is loaded using pickle.load(open('model.pkl', 'rb')). This eliminates the need to retrain models every time the application starts, significantly reducing processing time. Pickle5 ensures seamless model integration into Flask applications, allowing real-time predictions from previously trained models. Since different Python versions affect pickled files, using Pickle5 ensures better compatibility, making it crucial for deploying machine learning models in web applications.

7. Pandas:

Pandas is a data manipulation and analysis library that plays a crucial role in preprocessing the CKD dataset. It provides efficient tools for loading, cleaning, and transforming data into a structured format. The CKD prediction system uses Pandas for handling missing values, encoding categorical variables, and structuring patient data before feeding it into machine learning models. Its DataFrame functionality allows for seamless integration with NumPy and Scikit-learn, ensuring smooth data processing pipelines. Pandas simplifies the management of large datasets, making it an essential library for data-driven applications.

8. NumPy:

NumPy is a fundamental library for numerical computing in Python, widely used in machine learning applications. It provides support for multi-dimensional arrays, mathematical operations, and linear algebra functions. In the CKD prediction system, NumPy ensures efficient handling of numerical data, allowing patient information to be transformed into structured arrays before model inference. The library's optimized performance enhances the speed and efficiency of computations, making it indispensable for processing large-scale medical data.

9. Operating System & Web Browsers:

The CKD prediction system is compatible with Windows 11, ensuring smooth execution on modern computing environments. The web-based application can be accessed through popular web browsers, including Google Chrome, Mozilla Firefox, and Microsoft Edge. This cross-platform accessibility enhances usability, allowing healthcare professionals and researchers to interact with the system from various devices without requiring additional software installations. With these software components, the CKD prediction system achieves high performance, scalability, and ease of deployment, making it a reliable tool for early-stage chronic kidney disease detection.

Furthermore, the integration of XGBoost enhances predictive accuracy by leveraging its gradient boosting capabilities, making it well-suited for handling imbalanced datasets like CKD data. The system's architecture also supports cross-validation and rigorous testing methodologies, ensuring the model generalizes well to unseen data. Through this well-structured combination of software components, the CKD prediction system offers a robust, scalable, and deployable solution, empowering healthcare professionals and researchers with an effective tool for early-stage kidney disease detection and prognosis.

6. DESIGN

6.1 SYSTEM ARCHITECTURE

The system architecture for chronic kidney disease (CKD) prediction is structured in a sequential pipeline to ensure efficient data processing, model training, and evaluation. The workflow consists of multiple stages, starting from data acquisition to model performance assessment.

The first step in the pipeline involves loading the CKD dataset, which serves as the foundational data source for training and testing predictive models. Following data acquisition, data cleaning is performed to handle missing values, remove inconsistencies, and standardize the dataset to ensure reliable input for machine learning algorithms. This step is crucial as poor data quality can significantly impact model performance[2].

Next, Exploratory Data Analysis (EDA) is conducted to understand the statistical properties of the dataset, identify patterns, and visualize feature distributions. EDA plays a key role in feature selection, helping determine the most relevant variables that influence CKD diagnosis[5].Once EDA is completed, the dataset is split into two subsets: the training dataset and the testing dataset. The training dataset is utilized to train the machine learning models, while the testing dataset is reserved for performance validation. This ensures that the models generalize well to unseen data and do not suffer from overfitting[3].

The model training phase involves the implementation of two machine learning algorithms: Random Forest Classifier and XGBoost Classifier. The Random Forest Classifier is an ensemble learning method that combines multiple decision trees to enhance predictive accuracy and reduce overfitting[11]. On the other hand, XGBoost is a gradient boosting technique known for its efficiency, speed, and superior predictive power in structured data tasks[4]. Both models are trained on the preprocessed dataset to extract patterns and relationships useful for CKD prediction.

After training, the models undergo testing to evaluate their performance on unseen data. The testing phase involves feeding the trained models with new input instances and comparing the predicted outcomes with actual CKD labels.

Finally, model evaluation is conducted using various performance metrics, such as accuracy, precision, recall, and F1-score, to determine the effectiveness of each classifier. Additionally, hyperparameter tuning is performed to optimize model parameters, enhancing the overall prediction capability[9].

The design of the Chronic Kidney Disease (CKD) prediction system focuses on delivering accurate results through an efficient workflow, ensuring modularity and scalability. The system is structured into several stages that streamline data processing, feature selection, model training, and prediction.

The input layer handles the collection of clinical and medical data, including features such as age, blood pressure, blood sugar levels, creatinine levels, and other diagnostic parameters relevant to CKD diagnosis.

Preprocessing Layer This layer is dedicated to preparing the input data for analysis. It includes:

Data Cleaning: Handling missing values, removing inconsistencies, and normalizing data.

Data Encoding: Transforming categorical variables into numerical formats suitable for machine learning.

Feature Scaling: Ensuring uniformity in the data range for optimal model performance. Feature Selection is a crucial step to enhance model accuracy and reduce computational complexity. Statistical and algorithmic methods are employed to identify the most significant predictors of CKD, eliminating redundant or irrelevant features.

The selected machine learning model for this project is XGBoost due to its robust performance with structured data. The model design includes:

- Training on historical CKD data to learn patterns and relationships.
- Hyperparameter tuning to optimize model performance and prevent overfitting.

The final layer is responsible for generating outputs based on the trained model. It classifies patients into categories, such as "CKD" or "Non-CKD," based on the input features.

Flowchart Representation

The system can be represented with a flowchart that includes:

- Input Layer: Receiving patient data.
- Preprocessing Layer: Cleaning and transforming the data.
- Feature Selection: Identifying significant variables.
- Machine Learning Model: Training and predicting outcomes.
- Output Layer: Presenting the final prediction (CKD or Non-CKD).

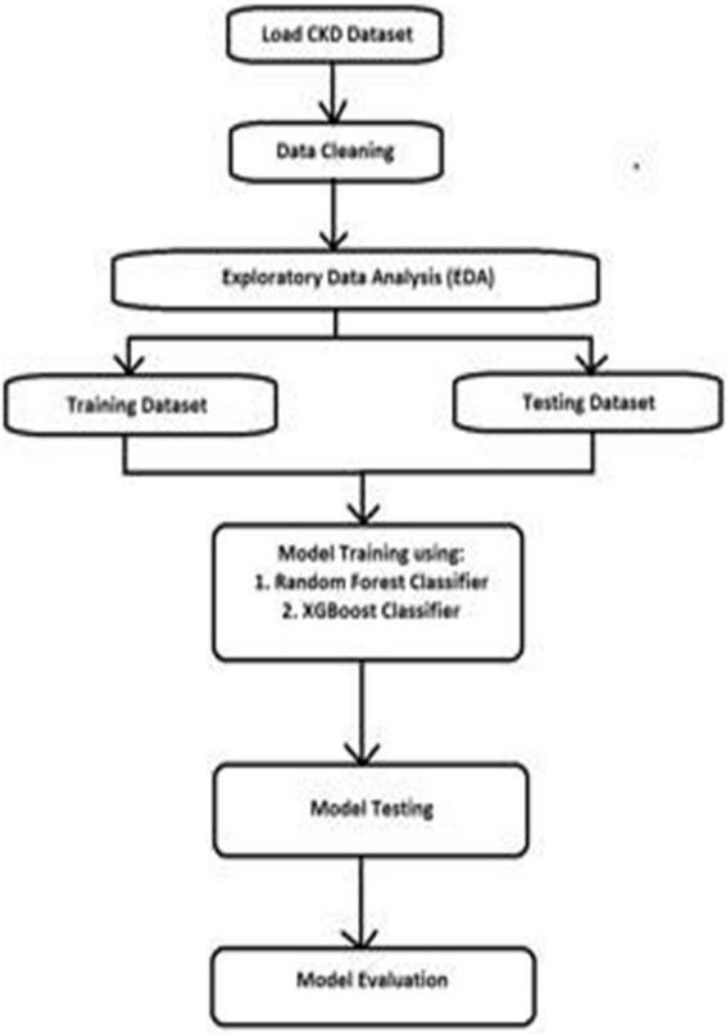


Fig 6.1.1 Design overview

The design of the Chronic Kidney Disease (CKD) prediction system focuses on delivering accurate results through an efficient workflow, ensuring modularity and scalability. The system is structured into several stages that streamline data processing, feature selection, model training, and prediction.

Input Layer

The input layer handles the collection of clinical and medical data, including features such as age, blood pressure, blood sugar levels, creatinine levels, and other diagnostic parameters relevant to CKD diagnosis. The input layer is responsible for data collection from various clinical and medical sources. The dataset includes vital patient parameters that are commonly associated with CKD diagnosis. The input features may include:

- Demographic Information: Age, gender, and other personal attributes.
- Clinical Measurements: Blood pressure, blood sugar levels, and serum creatinine levels.
- Laboratory Test Results: Hemoglobin levels, albumin levels, white blood cell count, and red blood cell count.
- Lifestyle & Health History: Smoking habits, diabetes history, hypertension status, and other chronic conditions.
- The quality of input data plays a significant role in the overall accuracy of the system, making proper data collection and validation essential.

Preprocessing Layer

This layer is dedicated to preparing the input data for analysis. It includes:

Data Cleaning

- Handling missing values using imputation techniques such as mean/mode filling or KNN imputation.
- Removing inconsistent data to maintain integrity.
- Filtering outliers that might affect model performance.

Data Encoding

- Converting categorical variables (e.g., "Yes" or "No" for diabetes) into numerical representations (e.g., 1 or 0).

- Using one-hot encoding or label encoding for categorical attributes to ensure compatibility with the model.

Feature Scaling

- Normalizing numerical features to a common scale (e.g., using Min-Max Scaling or Standardization).
- Ensuring that all input values remain within a defined range, preventing bias due to large variations in feature magnitudes.

These preprocessing steps enhance data quality, ensuring better learning efficiency for the model.

Feature Selection

Feature selection is a crucial step to enhance model accuracy and reduce computational complexity. Statistical and algorithmic methods are employed to identify the most significant predictors of CKD, eliminating redundant or irrelevant features.

Machine Learning Model

The selected machine learning model for this project is XGBoost due to its robust performance with structured data. The model design includes:

- Training on historical CKD data to learn patterns and relationships.
- Hyperparameter tuning to optimize model performance and prevent overfitting.

Prediction Layer

The final layer is responsible for generating outputs based on the trained model. It classifies patients into categories, such as "CKD" or "Non-CKD," based on the input features.

- Processes input features through the preprocessing pipeline.
- Applies the trained model to classify whether the patient is at risk of CKD or not.
- Outputs results in an interpretable format, such as:
- “CKD” (Chronic Kidney Disease detected)
- “Non-CKD” (No signs of CKD)

6.2 UML DIAGRAMS

The Use Case Diagram represents the interaction between users and the Chronic Kidney Disease (CKD) Prediction System, illustrating the essential processes involved in data preprocessing, model training, and prediction generation. The system is designed to provide an efficient and user-friendly approach for CKD diagnosis, leveraging machine learning models to deliver accurate predictions.

The process begins with the user providing a CKD dataset, which undergoes extensive preprocessing and feature extraction. This step involves handling missing values, normalizing data, and selecting the most relevant features for improved model performance. Preprocessed data is then used to train various machine learning models, including Random Forest (RF), Decision Tree (DT), Support Vector Machine (SVM), and Artificial Neural Networks (ANN). These models are optimized to ensure high accuracy in CKD classification.

Once the training phase is completed, the trained model is integrated into a web-based interface developed using Flask. The user can input new patient data through this interface, and the system processes the data using the trained model to generate predictions regarding CKD status. The final output is presented to the end-user, such as doctors or medical professionals, who can utilize the results for further decision-making in clinical practice.

The CKD Prediction System automates data preprocessing, model training, and predictive analysis, ensuring an efficient and streamlined workflow. By employing Flask for deployment instead of SHAP and LIME for model interpretability, the system maintains a lightweight yet robust structure. This approach enhances scalability, accessibility, and real-time usability, making it a reliable tool for healthcare professionals and researchers in the field of nephrology.

The interaction diagram represents the workflow of a Chronic Kidney Disease (CKD) Prediction System and how different components interact. The system involves a User, System, Preprocessing Module, Machine Learning Model, and Database to predict CKD based on patient data, store results for further analysis, and continuously improve the model's accuracy.The process begins with the user providing patient data, such as blood test results, urine test parameters, and other health indicators. This input is received by the system, which forwards it to the Preprocessing module. Here, the raw

data undergoes cleaning, feature selection, and normalization to ensure that the machine learning model receives high-quality input for better prediction accuracy.

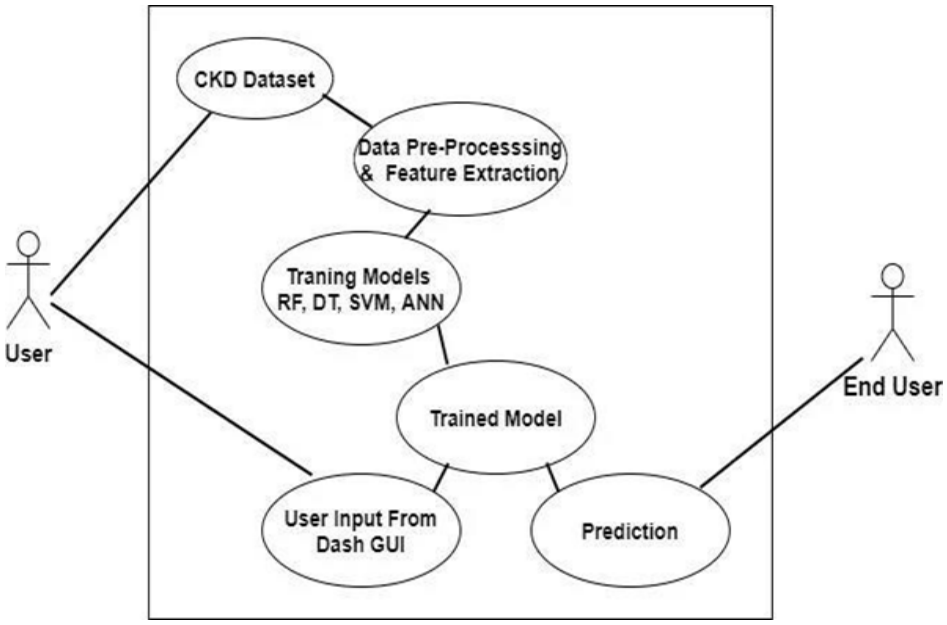


Fig.6.2.1 Use Case Diagram

Once the data is preprocessed, it is sent to the Machine Learning Model, which applies algorithms such as XGBoost or Decision Tree to analyze the input and generate a CKD prediction result. This prediction is then returned to the System, which displays the results to the User for further medical decision-making.To enhance model performance over time, the predicted results are stored in the Database. This data can be analyzed later to monitor trends, assess the model's performance, and identify potential false positives or negatives. As new patient data is collected, the system retrains the model using this updated information. This retraining process ensures that the CKD prediction model remains accurate, adaptive, and up-to-date with evolving medical trends.

Finally, the improved model is deployed, replacing the older version in the system. This allows the CKD prediction system to function with higher accuracy and reliability, benefiting both medical professionals and patients. The automation of data preprocessing, model application, and continuous improvement makes this system a valuable AI-driven tool for early CKD detection, helping enhance healthcare efficiency and patient care.

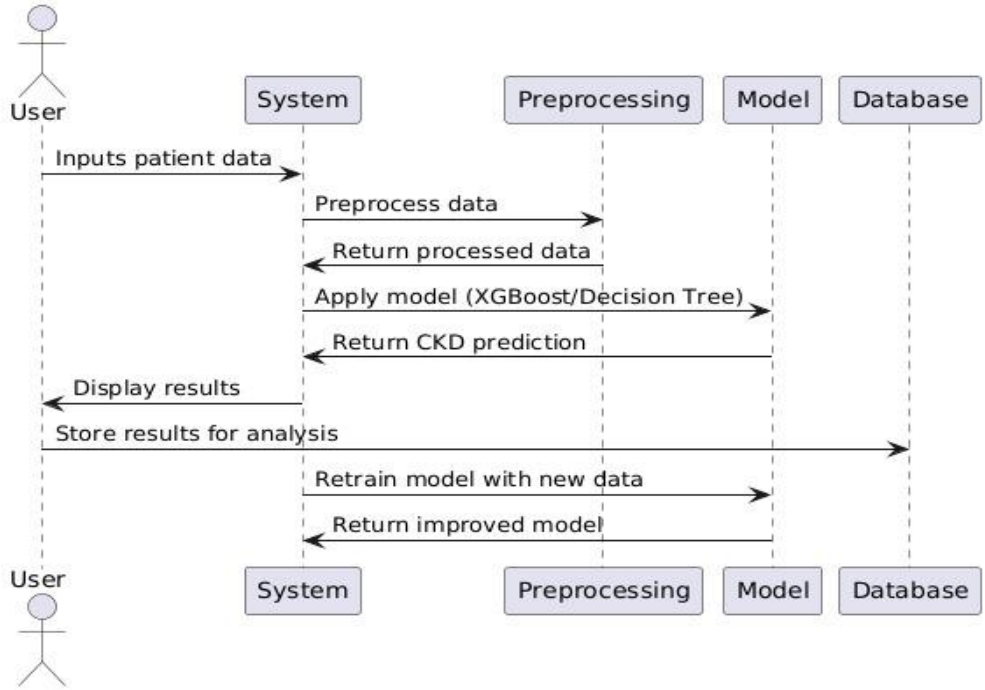


Fig 6.2.2 : Interaction Diagram

The sequence diagram illustrates the workflow of the Chronic Kidney Disease (CKD) prediction system, detailing how user input is processed, analyzed, and utilized for continuous model improvement. This system involves multiple components, including the user, preprocessing module, machine learning model, and database, ensuring a structured approach to CKD prediction and model enhancement.

The process begins when the user inputs patient data into the system. This data includes medical test results such as blood pressure, serum creatinine levels, and other relevant biomarkers. The system forwards this input to the preprocessing module , where essential data preparation techniques like handling missing values, feature scaling, encoding categorical variables, and outlier detection are performed. This step ensures that the data is clean and structured for accurate model predictions.

Once preprocessing is complete, the processed data is passed to the machine learning model which could be XGBoost or a Decision Tree classifier for prediction. The model analyzes the data and predicts whether the patient is likely to have CKD. The system then returns the CKD prediction to the user, displaying the results in an easily interpretable format.

To improve future predictions, the system stores the results in a database for analysis. This allows for tracking patient data trends and refining the prediction model. Over time, as new data is collected, the system retrains the model using updated datasets. This step enhances the model's accuracy, ensures it adapts to evolving medical data, and reduces bias.

Finally, once the model is retrained and optimized, the **improved model is returned to the system**, making it more effective in real-world applications. This continuous feedback loop ensures that the **CKD prediction system remains accurate, reliable, and adaptable**, benefiting both clinicians and patients by enabling early detection and timely medical intervention.

6.3 Scope of the project

Prediction of CKD: The primary objective is to build a predictive model that can accurately classify individuals as either having CKD or not, based on clinical and laboratory data. The system aims to detect CKD in its early stages, enabling timely intervention and management.

Data Preprocessing and Feature Engineering: The project includes extensive data preprocessing techniques to handle missing values, encode categorical variables, and normalize numerical features. Feature selection methods like Chi-Square and PCA will be employed to identify the most relevant variables, ensuring that the model is trained with the most important data.

Machine Learning Model Development: The project employs various machine learning algorithms such as Decision Trees, Random Forest, XGBoost, and SVM, evaluating their performance in predicting CKD. The best model will be chosen based on accuracy, precision, recall, and other performance metrics.

User Interface Development: A Flask-based web application will be developed to make the system accessible to healthcare professionals. This interface allows users to input clinical parameters and receive real-time predictions on whether a patient has CKD, ensuring ease of use in clinical settings.

Model Evaluation and Validation: The system's performance will be assessed through various evaluation metrics, including accuracy, precision, recall, F1-score, and ROC- AUC. This ensures the reliability of the model in providing correct predictions.

Real-World Applicability: The system is designed to be integrated into healthcare settings for practical use, assisting healthcare professionals in diagnosing CKD early, thereby improving patient outcomes.

Future Enhancements: The system's scope includes potential for future improvements, such as the integration of the model with Electronic Health Records (EHR) systems and real-time monitoring of patients, enhancing its effectiveness in clinical practice.

6.4 FEASIBILITY STUDY

A feasibility study is conducted to evaluate the practicality and effectiveness of the proposed CKD prediction system. This assessment considers multiple factors, including technical, economic, operational, legal, and schedule feasibility, ensuring the project's success and sustainability.

1. Technical Feasibility

The CKD prediction system is developed using open-source tools and frameworks, making it highly accessible and cost-effective. The model training and evaluation are performed on Google Colab Pro, which provides enhanced computational resources, including faster GPUs and longer runtime sessions. The use of Flask for model interpretation and deployment ensures a lightweight and scalable web-based application.

The system integrates various machine learning and deep learning models, including XGBoost, Random Forest, and CNN, implemented using Python libraries such as Scikit-learn, TensorFlow, and XGBoost. Once an optimal model is finalized, it is deployed using Flask, allowing seamless integration into web-based applications.

Flask facilitates real-time inference by enabling users to input patient data and receive immediate CKD predictions through an interactive interface. The lightweight nature of Flask ensures efficient model deployment without the need for extensive computational resources. Additionally, the system is designed for scalability, allowing integration with cloud platforms for remote accessibility.

Continuous monitoring and periodic updates of the model with new patient data help maintain accuracy and adaptability, ensuring it remains effective in evolving medical scenarios.

2. Economic Feasibility

The project is designed with cost-effectiveness in mind, utilizing Google Colab Pro for training instead of expensive on-premise GPU servers, significantly reducing hardware costs. Since the entire system is built using open-source resources, there are no licensing fees associated with proprietary software.

Flask-based deployment further eliminates the need for expensive enterprise solutions, making the system viable for hospitals, clinics, and research institutions with limited budgets. The implementation of AI for early CKD detection reduces healthcare costs by minimizing the need for expensive laboratory tests and enabling proactive disease management.

3. Operational Feasibility

The system is user-friendly, designed for seamless integration into existing healthcare workflows. The Flask-based web interface allows healthcare professionals to input patient data and receive real-time CKD predictions with interpretability insights.

The system does not require specialized technical knowledge, making it easy to use for both clinicians and medical researchers. The use of Google Colab Pro for model training ensures continuous updates and improvements to the predictive models without requiring high-end local hardware.

4. Schedule Feasibility

The project follows a structured timeline, with well-defined phases for data collection, model development, evaluation, deployment, and continuous monitoring. The use of Google Colab Pro speeds up model training and testing, reducing development time. Flask-based deployment allows for rapid prototyping and real-time model testing, ensuring timely project completion.

Continuous improvements and periodic retraining are planned to keep the model updated with evolving medical knowledge.The feasibility study confirms that the CKD prediction system is technically sound, economically viable, operationally efficient, and legally compliant. By leveraging Google Colab Pro for model training and open-source tools for development and deployment, the project remains cost-effective while providing a scalable and accessible solution for early CKD detection.

The system's ease of use, transparency, and compliance with healthcare standards make it a promising tool for clinical and research applications, ultimately improving patient care and disease management.

5.Legal and Ethical Feasibility

Legal and ethical feasibility ensures that the CKD prediction system complies with data privacy laws, regulatory guidelines, and ethical standards to protect patient rights and maintain trust in AI-driven healthcare solutions. Given the sensitive nature of medical data, adherence to strict legal and ethical frameworks is essential for the successful deployment and acceptance of the system.

One of the most critical aspects is data privacy and security. The system must comply with regulations such as the Health Insurance Portability and Accountability Act (HIPAA), General Data Protection Regulation (GDPR), and the Indian IT Act 2000, which govern the collection, storage, and usage of patient data. These regulations ensure that patient information remains confidential, is securely encrypted, and is not misused by unauthorized entities. Implementing end-to-end encryption, access control mechanisms, and anonymization techniques further strengthens data protection.

7. IMPLEMENTATION

7.1 MODEL IMPLEMENTATION

```
%pip install xgboost

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

import xgboost as xgb

model = xgb.XGBClassifier(objective='binary:logistic', random_state=42)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

xg_acc= accuracy_score(y_test, y_pred)

print(f"Accuracy: {xg_acc * 100:.2f}%")

print('Classification Report:\n', classification_report(y_test, y_pred))
```

Flask Code to Connect Front End

```
from flask import Flask, render_template,

request

import numpy as np import pickle

  app = Flask(__name__)

model  =          pickle.load(open('C:\project

AB6\Source Code\Kidney.pkl', 'rb')) @app.route('/', methods=['GET']) def home():

    return render_template('home.html')

 @app.route('/about', methods=['GET']) def about():

    return render_template('about.html')

@app.route('/predictions',       methods=['GET',

'POST']) def predictions():     if request.method == 'POST':         try:

          int_features = [float(x) for x in

request.form.values()]            final = [np.array(int_features) prediction =
model.predict_proba(final)                                                output  =

'{0:.{1}f}'.format(prediction[0][1], 2)            if float(output) >= 0.5:  # Adjust
threshold for CKD if needed
```

```
            return render_template('result.html', prediction="CKD Detected")
else:
            return render_template('result2.html', prediction="No CKD Detected")
except Exception as e:          return render_template('error.html',
error=str(e))
  return render_template('image.html') @app.route('/metrics', methods=['GET']) def
metrics():
  return render_template('metrics.html') @app.route('/flowchart', methods=['GET'])
def flowchart():
  return render_template('flowchart.html') @app.errorhandler(404) def
page_not_found(e):              return  render_template('error.html',
error="Page not found!"), 404
if __name__ == "__main__":  app.run(host="0.0.0.0", port=5000,
debug=True)
```

7.2 CODING

```
import warnings warnings.filterwarnings('ignore')
%pip install pandas numpy matplotlib seaborn import pandas as pd import numpy as
np import matplotlib.pyplot as plt
ckd_data = pd.read_csv("/content/drive/MyDrive/kidney_disease.csv")
ckd_data.head() ckd_data.shape ckd_data.columns = ['id','age', 'blood_pressure',
'specific_gravity', 'albumin', 'sugar', 'red_blood_cells', 'pus_cell',
        'pus_cell_clumps', 'bacteria', 'blood_glucose_random', 'blood_urea',
'serum_creatinine', 'sodium',
        'potassium', 'haemoglobin', 'packed_cell_volume', 'white_blood_cell_count',
'red_blood_cell_count',
        'hypertension', 'diabetes_mellitus', 'coronary_artery_disease', 'appetite',
'peda_edema',
        'aanemia', 'class'] ckd_data.head() ckd_data.info()
ckd_data['packed_cell_volume'] = pd.to_numeric(ckd_data['packed_cell_volume'],
errors='coerce')
ckd_data['white_blood_cell_count'] =
pd.to_numeric(ckd_data['white_blood_cell_count'],
errors='coerce')
```

```
ckd_data['red_blood_cell_count'] = pd.to_numeric(ckd_data['red_blood_cell_count'],

errors='coerce')

ckd_data.info()

 ckd_data.isnull().sum()

import seaborn as sns import matplotlib.pyplot as plt plt.figure(figsize=(6, 6)) if 'id' in
ckd_data.columns:

   ckd_data = ckd_data.drop('id', axis=1)

 sns.heatmap(ckd_data.isnull(), cbar=False, cmap='viridis')

plt.title('Missing Values Heatmap') plt.show() cat_cols = [col for col in
ckd_data.columns if ckd_data[col].dtype == 'object'] num_cols = [col for col in
ckd_data.columns if ckd_data[col].dtype != 'object'] print('Numerical columns:',
num_cols) print('Categorical columns:', cat_cols) ckd_data.describe() for col in
cat_cols:

   print(f"{col} has {ckd_data[col].unique()} values\n")

ckd_data['diabetes_mellitus'].replace(to_replace = {'\tno':'no','\tyes':'yes','

yes':'yes'},inplace=True)

ckd_data['coronary_artery_disease'] =
ckd_data['coronary_artery_disease'].replace(to_replace =

'\tno', value='no')   ckd_data['class'] = ckd_data['class'].replace(to_replace = {'ckd\t':
'ckd', 'notckd': 'not ckd'})

ckd_data['class'] = ckd_data['class'].map({'ckd': 0, 'not ckd': 1}) ckd_data['class'] =
pd.to_numeric(ckd_data['class'], errors='coerce') cols = ['diabetes_mellitus',
'coronary_artery_disease', 'class'] for col in cols:

   print(f"{col} has {ckd_data[col].unique()} values\n") plt.figure(figsize = (20, 15))

plotnumber = 1 olumn in cat_cols:     if plotnumber <= 11:         ax = plt.subplot(3, 4,
plotnumber)

      sns.countplot(ckd_data[column], palette = 'rocket')         plt.xlabel(column)
plotnumber += 1 plt.tight_layout() plt.show()

plt.figure(figsize = (20, 15)) plotnumber = 1 for column in num_cols:     if plotnumber
<= 14:         ax = plt.subplot(3, 5, plotnumber)         sns.distplot(ckd_data[column])
plt.xlabel(column)     plotnumber += 1 plt.tight_layout() plt.show() ckd_data.columns

ckd_data['age'].fillna(ckd_data['age'].mean(), inplace=True)
ckd_data['blood_pressure'].fillna(ckd_data['blood_pressure'].median(), inplace=True)
ckd_data['specific_gravity'].fillna(ckd_data['specific_gravity'].mean(), inplace=True)
```

```
ckd_data['albumin'].fillna(ckd_data['albumin'].mean(), inplace=True)
ckd_data['sugar'].fillna(ckd_data['sugar'].mean(), inplace=True)

ckd_data['blood_glucose_random'].fillna(ckd_data['blood_glucose_random'].mean(),

inplace=True)

ckd_data['blood_urea'].fillna(ckd_data['blood_urea'].mean(), inplace=True)
ckd_data['serum_creatinine'].fillna(ckd_data['serum_creatinine'].mean(),
inplace=True) ckd_data['sodium'].fillna(ckd_data['sodium'].mean(), inplace=True)
ckd_data['potassium'].fillna(ckd_data['potassium'].mean(), inplace=True)
ckd_data['haemoglobin'].fillna(ckd_data['haemoglobin'].mean(), inplace=True)

ckd_data['packed_cell_volume'].fillna(ckd_data['packed_cell_volume'].mean(),
inplace=True)
ckd_data['white_blood_cell_count'].fillna(ckd_data['white_blood_cell_count'].mean()
,

inplace=True)

ckd_data['red_blood_cell_count'].fillna(ckd_data['red_blood_cell_count'].mean(),
inplace=True) ckd_data.isnull().sum()

  for col in cat_cols:

    plt.figure(figsize=(8,6))

    value_counts = ckd_data[col].value_counts().nlargest(2)
value_counts.plot(kind='bar', color=['blue', 'orange'])     plt.title(col, fontsize=14)
plt.show() for column in cat_cols:     value_counts =
ckd_data[column].value_counts()     print(f'Value counts for
{column}:\n{value_counts}\n') ckd_data.isnull().sum() duplicate_rows =
ckd_data[ckd_data.duplicated()] if not duplicate_rows.empty:

    print("Duplicate rows found:")     print(duplicate_rows) else:     print("No duplicate
rows found.")

target_var = 'class' for col in cat_cols:     print(f"{col} has {ckd_data[col].nunique()}
categories\n")

from sklearn.preprocessing import LabelEncoder le = LabelEncoder()#is used to
convert categorical values into numerical labels, ckd_data['red_blood_cells'] =
le.fit_transform(ckd_data['red_blood_cells']) ckd_data['pus_cell'] =
le.fit_transform(ckd_data['pus_cell'])

ckd_data['pus_cell_clumps'] = le.fit_transform(ckd_data['pus_cell_clumps'])
ckd_data['bacteria'] = le.fit_transform(ckd_data['bacteria']) ckd_data['hypertension'] =
le.fit_transform(ckd_data['hypertension']) ckd_data['diabetes_mellitus'] =
le.fit_transform(ckd_data['diabetes_mellitus'])
```

```
ckd_data['coronary_artery_disease'] =
le.fit_transform(ckd_data['coronary_artery_disease']) ckd_data['appetite'] =
le.fit_transform(ckd_data['appetite']) ckd_data['peda_edema'] =
le.fit_transform(ckd_data['peda_edema'])

ckd_data['class'] = le.fit_transform(ckd_data['class']) ckd_data['aanemia'] =
le.fit_transform(ckd_data['aanemia']) corr = ckd_data.corr() plt.figure(figsize =
(40,25)) sns.heatmap(corr, annot=True)

ind_col=[col for col in ckd_data.columns if col!="class"] def_col='class'

X=ckd_data[ind_col] y=ckd_data[def_col] from imblearn.over_sampling import
SMOTE from sklearn.datasets import make_classification from collections import
Counter

# Generate a synthetic imbalanced dataset

X, y = make_classification(n_classes=2, class_sep=2,

                    weights=[0.1, 0.9], n_informative=3, n_redundant=1,
flip_y=0, n_features=20, n_clusters_per_class=1,                    n_samples=1000,
random_state=10)

print('Original dataset shape %s' % Counter(y)) smote = SMOTE(random_state=42)

X_resampled, y_resampled = smote.fit_resample(X, y) print('Resampled dataset
shape %s' % Counter(y_resampled)) ind_col=[col for col in ckd_data.columns if
col!="class"] def_col='class'

X=ckd_data[ind_col] y=ckd_data[def_col] import numpy as np import pandas as pd

from sklearn.model_selection import train_test_split from sklearn.metrics import
accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size = 0.2,
random_state=1) X_train.shape,X_test.shape,Y_train.shape,Y_test.shape from
sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,
random_state=42) from sklearn.tree import DecisionTreeClassifier dct =
DecisionTreeClassifier() dct.fit(X_train, y_train) y_pred_dct = dct.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred_dct)) print(classification_report(y_test,
y_pred_dct))

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print("Decision tree Accuracy:", accuracy_score(y_test, y_pred_dct))
```

```
print("Precision:", precision_score(y_test, y_pred_dct, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_dct, average='weighted')) print("F1-
score:", f1_score(y_test, y_pred_dct, average='weighted')) train_accuracy =
dct.score(X_train, y_train) print(f"Training Accuracy: {train_accuracy}")
test_accuracy = dct.score(X_test, y_test) print(f"Testing Accuracy: {test_accuracy}")
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators = 11, criterion = 'entropy', random_state =
42) rfc.fit(X_train, y_train) y_pred_rfc = rfc.predict(X_test) rfc_acc =
accuracy_score(y_test, y_pred_rfc) from sklearn.metrics import classification_report
print(confusion_matrix(y_test, y_pred_rfc)) print(classification_report(y_test,
y_pred_rfc))

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rfc))
print("Precision:", precision_score(y_test, y_pred_rfc, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_rfc, average='weighted')) print("F1-
score:", f1_score(y_test, y_pred_rfc, average='weighted')) train_accuracy =
rfc.score(X_train, y_train) print(f"Training Accuracy: {train_accuracy}")
test_accuracy = rfc.score(X_test, y_test) print(f"Testing Accuracy: {test_accuracy}")
import tensorflow as tf from tensorflow import keras

from sklearn.preprocessing import StandardScaler scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train) X_test_scaled =
scaler.transform(X_test) model = keras.Sequential([

   keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
keras.layers.Dense(32, activation='relu'),     keras.layers.Dense(1,
activation='sigmoid') # Output layer for binary classification

])

model.compile(optimizer='adam',

              loss='binary_crossentropy',              metrics=['accuracy'])

model.fit(X_train_scaled, y_train, epochs=50, batch_size=32, validation_split=0.1)
loss, accuracy = model.evaluate(X_test_scaled, y_test) print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuracy}")

y_pred_ann = (model.predict(X_test_scaled) > 0.5).astype("int32") # Convert
probabilities to class labels

print(classification_report(y_test, y_pred_ann))

print("ANN Accuracy:", accuracy_score(y_test, y_pred_ann)) print("Precision:",
precision_score(y_test, y_pred_ann, average='weighted')) print("Recall:",
recall_score(y_test, y_pred_ann, average='weighted')) print("F1-score:",
f1_score(y_test, y_pred_ann, average='weighted')) _, train_accuracy_ann =
```

model.evaluate(X_train_scaled, y_train, verbose=0) print(f"ANN Training Accuracy: {train_accuracy_ann}")

loss, test_accuracy_ann = model.evaluate(X_test_scaled, y_test, verbose=0) print(f"ANN Test Accuracy: {test_accuracy_ann}") ann_train_params = {

'epochs': 50,

'batch_size': 32,

'validation_split': 0.1,

'input_shape': X_train.shape[1],

'layers': [64,32,1], # Number of neurons in each layer

'activations': ['relu', 'relu', 'sigmoid'], # Activation functions for each layer 'optimizer': 'adam',

'loss': 'binary_crossentropy',

'metrics': ['accuracy'],

'n_samples': len(X_train)

}

ann_test_params = {

'n_samples': len(X_test)

} print("Decision Tree Training Parameters:", dct_train_params) print("Decision Tree Testing Parameters:", dct_test_params) print("Random Forest Training Parameters:", rfc_train_params) print("Random Forest Testing Parameters:", rfc_test_params) print("ANN Training Parameters:", ann_train_params) print("ANN Testing Parameters:", ann_test_params) from sklearn.linear_model import LogisticRegression from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import cross_val_score, StratifiedKFold

from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, precision_score, recall_score, f1_score scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train) X_test_scaled = scaler.transform(X_test)

logreg = LogisticRegression(max_iter=1000, C=0.1, solver='liblinear', class_weight='balanced') cv = StratifiedKFold(n_splits=5)

cv_scores = cross_val_score(logreg, X_train_scaled, y_train, cv=cv, scoring='accuracy') print("Cross-validation scores:", cv_scores) print("Mean cross-validation score:", cv_scores.mean()) logreg.fit(X_train_scaled, y_train)

y_pred_logreg = logreg.predict(X_test_scaled) print(confusion_matrix(y_test, y_pred_logreg)) print(classification_report(y_test, y_pred_logreg))

print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_logreg)) print("Precision:", precision_score(y_test, y_pred_logreg, average='weighted')) print("Recall:", recall_score(y_test, y_pred_logreg, average='weighted')) print("F1-score:", f1_score(y_test, y_pred_logreg, average='weighted')) train_accuracy = logreg.score(X_train_scaled, y_train) print(f"Training Accuracy: {train_accuracy}") test_accuracy = logreg.score(X_test_scaled, y_test) print(f"Testing Accuracy: {test_accuracy}") from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import cross_val_score, StratifiedKFold

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, precision_score, recall_score, f1_score scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

svm = SVC(kernel='linear', C=0.1, class_weight='balanced') cv = StratifiedKFold(n_splits=5)

cv_scores = cross_val_score(svm, X_train_scaled, y_train, cv=cv, scoring='accuracy') print("Cross-validation scores:", cv_scores) print("Mean cross-validation score:", cv_scores.mean()) svm.fit(X_train_scaled, y_train) y_pred_svm = svm.predict(X_test_scaled) print(confusion_matrix(y_test, y_pred_svm)) print(classification_report(y_test, y_pred_svm)) print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm)) print("Precision:", precision_score(y_test, y_pred_svm, average='weighted')) print("Recall:", recall_score(y_test, y_pred_svm, average='weighted')) print("F1-score:", f1_score(y_test, y_pred_svm, average='weighted')) train_accuracy = svm.score(X_train_scaled, y_train) print(f"Training Accuracy: {train_accuracy}") test_accuracy = svm.score(X_test_scaled, y_test) print(f"Testing Accuracy: {test_accuracy}") from sklearn.model_selection import train_test_split from sklearn.preprocessing import StandardScaler from sklearn.neighbors import KNeighborsClassifier from sklearn.metrics import accuracy_score, classification_report scaler = StandardScaler()

X_train = scaler.fit_transform(X_train) X_test = scaler.transform(X_test) k_value = 5

knn_model = KNeighborsClassifier(n_neighbors=k_value)

knn_model.fit(X_train, y_train) y_pred = knn_model.predict(X_test) knn_acc= accuracy_score(y_test, y_pred) print(confusion_matrix(y_test, y_pred))

print('Classification Report:\n', classification_report(y_test, y_pred))

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print("knn accuracy:",accuracy_score(y_test, y_pred))

print("Precision:", precision_score(y_test, y_pred, average='weighted'))
print("Recall:", recall_score(y_test, y_pred, average='weighted')) print("F1-score:",
f1_score(y_test, y_pred, average='weighted')) train_accuracy =
knn_model.score(X_train, y_train) print(f"Training Accuracy: {train_accuracy}")
test_accuracy = knn_model.score(X_test, y_test) print(f"Testing Accuracy:
{test_accuracy}") from sklearn.model_selection import train_test_split from
sklearn.metrics import accuracy_score import xgboost as xgb

model = xgb.XGBClassifier(objective='binary:logistic', random_state=42)
model.fit(X_train, y_train) y_pred = model.predict(X_test)

xg_acc= accuracy_score(y_test, y_pred)

 print(f"Accuracy: {xg_acc * 100:.2f}%")

print('Classification Report:\n', classification_report(y_test, y_pred))

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print("xgboost  accuracy:",accuracy_score(y_test, y_pred)) print("Precision:",
precision_score(y_test, y_pred, average='weighted')) print("Recall:",
recall_score(y_test, y_pred, average='weighted')) print("F1-score:", f1_score(y_test,
y_pred, average='weighted')) train_accuracy = model.score(X_train, y_train)
print(f"Training Accuracy: {train_accuracy}") test_accuracy = model.score(X_test,
y_test) print(f"Testing Accuracy: {test_accuracy}")

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay cm =
confusion_matrix(y_test, y_pred) disp =
ConfusionMatrixDisplay(confusion_matrix=cm) disp.plot() plt.show()

from sklearn.ensemble import AdaBoostClassifier from sklearn.preprocessing import
StandardScaler

from sklearn.model_selection import cross_val_score, StratifiedKFold from
sklearn.metrics import classification_report, confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train) X_test_scaled =
scaler.transform(X_test)

model = AdaBoostClassifier(n_estimators=50, learning_rate=0.1, random_state=42)
cv = StratifiedKFold(n_splits=5)

cv_scores = cross_val_score(model, X_train_scaled, y_train, cv=cv,
scoring='accuracy') print("Cross-validation scores:", cv_scores) print("Mean cross-
validation score:", cv_scores.mean()) model.fit(X_train_scaled, y_train) y_pred =
model.predict(X_test_scaled) print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

print("AdaBoost Accuracy:", accuracy_score(y_test, y_pred)) print("Precision:", precision_score(y_test, y_pred, average='weighted')) print("Recall:", recall_score(y_test, y_pred, average='weighted')) print("F1-score:", f1_score(y_test, y_pred, average='weighted')) train_accuracy = model.score(X_train_scaled, y_train) print(f"Training Accuracy: {train_accuracy}") test_accuracy = model.score(X_test_scaled, y_test) print(f"Testing Accuracy: {test_accuracy}")

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay cm = confusion_matrix(y_test, y_pred) print(cm)

disp = ConfusionMatrixDisplay(confusion_matrix=cm) disp.plot() plt.show()

from sklearn.feature_selection import chi2 from sklearn.feature_selection import SelectKBest

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score from sklearn.preprocessing import MinMaxScaler # import MinMaxScaler for scaling scaler = MinMaxScaler()

X_train_scaled = scaler.fit_transform(X_train) X_test_scaled = scaler.transform(X_test) k = 10 # Select top 10 features

selector = SelectKBest(score_func=chi2, k=k)

X_train_chi = selector.fit_transform(X_train_scaled, y_train) # Use scaled data X_test_chi = selector.transform(X_test_scaled) # Use scaled data

model = LogisticRegression(max_iter=1000, C=0.1, solver='liblinear', class_weight='balanced') model.fit(X_train_chi, y_train) y_pred_chi = model.predict(X_test_chi) print(confusion_matrix(y_test, y_pred_chi)) print(classification_report(y_test, y_pred_chi))

print("Chi-Square Accuracy:", accuracy_score(y_test, y_pred_chi)) print("Precision:", precision_score(y_test, y_pred_chi, average='weighted')) print("Recall:", recall_score(y_test, y_pred_chi, average='weighted')) print("F1-score:", f1_score(y_test, y_pred_chi, average='weighted')) train_accuracy = model.score(X_train_chi, y_train) print(f"Training Accuracy: {train_accuracy}") test_accuracy = model.score(X_test_chi, y_test) print(f"Testing Accuracy: {test_accuracy}") from sklearn.model_selection import train_test_split y = ckd_data['class']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) print("X_train shape:", X_train.shape) print("X_test shape:", X_test.shape) print("y_train shape:", y_train.shape) print("y_test shape:", y_test.shape) train_params = X_train.shape[0] * X_train.shape[1] test_params = X_test.shape[0] * X_test.shape[1] print(f"Number of training parameters: {train_params}") print(f"Number of testing parameters: {test_params}") from tabulate import tabulate

```
models = {
   "Decision Tree": dct,
   "Random Forest": rfc,
   "Logistic Regression": logreg,
   "SVM": svm,
   "KNN": knn_model,
   "XGBoost": model,  # Assuming this is the XGBoost model
   "AdaBoost": model   # Assuming this is the AdaBoost model
} table_data = [] for model_name, model in models.items():     if model_name in ["Logistic Regression", "SVM", "AdaBoost"]:         train_params = X_train_scaled.shape[0] * X_train_scaled.shape[1]         test_params = X_test_scaled.shape[0] * X_test_scaled.shape[1]     elif model_name == "Chi-Square":
       train_params = X_train_chi.shape[0] * X_train_chi.shape[1]         test_params = X_test_chi.shape[0] * X_test_chi.shape[1]     else:
       train_params = X_train.shape[0] * X_train.shape[1]         test_params = X_test.shape[0] * X_test.shape[1]     table_data.append([model_name, train_params, test_params])

table = tabulate(table_data, headers=["Model", "Training Parameters", "Testing Parameters"], tablefmt="grid") print(table)

from sklearn.model_selection import cross_val_score, StratifiedKFold from sklearn.linear_model import LogisticRegression from sklearn.metrics import make_scorer, accuracy_score cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

accuracy_scores = cross_val_score(logreg, X, y, cv=cv, scoring=make_scorer(accuracy_score)) plt.figure(figsize=(8, 6))

plt.plot(np.arange(1, 6), accuracy_scores, marker='o', linestyle='-', color='b', label='Accuracy') plt.title('Cross-Validation Performance - Logistic Regression') plt.xlabel('Fold') plt.ylabel('Accuracy') plt.xticks(np.arange(1, 6)) plt.ylim([0, 1]) plt.legend() plt.grid(True) plt.show()

from sklearn.model_selection import cross_val_score, StratifiedKFold from sklearn.linear_model import LogisticRegression from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import make_scorer, accuracy_score cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

accuracy_scores = cross_val_score(ranfor, X, y, cv=cv, scoring=make_scorer(accuracy_score)) plt.figure(figsize=(8, 6))
```

plt.plot(np.arange(1, 6), accuracy_scores, marker='o', linestyle='-', color='b', label='Accuracy') plt.title('Cross-Validation Performance - Random Forest') plt.xlabel('Fold') plt.ylabel('Accuracy') plt.xticks(np.arange(1, 6)) plt.ylim([0, 1]) plt.legend()

plt.grid(True) plt.show() import warnings import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns

from sklearn.preprocessing import LabelEncoder from imblearn.over_sampling import SMOTE from sklearn.datasets import make_classification from collections import Counter

from sklearn.model_selection import train_test_split from sklearn.metrics import accuracy_score from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import classification_report, confusion_matrix

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score from sklearn.model_selection import cross_val_score from sklearn.model_selection import KFold

from sklearn.model_selection import cross_val_score, StratifiedKFold from sklearn.linear_model import LogisticRegression from sklearn.metrics import make_scorer, accuracy_score from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import classification_report import xgboost as xgb

from sklearn.preprocessing import StandardScaler from sklearn.svm import SVC from sklearn.neighbors import KNeighborsClassifier from sklearn.ensemble import AdaBoostClassifier from sklearn.feature_selection import SelectKBest, chi2 from sklearn import metrics from sklearn.decomposition import PCA selected_features = ['specific_gravity', 'albumin', 'sugar', 'blood_glucose_random',

'serum_creatinine', 'potassium', 'packed_cell_volume', 'white_blood_cell_count',

'red_blood_cell_count', 'diabetes_mellitus', 'class'] ckd_data_pca = ckd_data[selected_features] ckd_data_pca = ckd_data_pca.fillna(ckd_data.mean())

X = ckd_data_pca.drop('class', axis=1) y = ckd_data_pca['class'] scaler = StandardScaler() X_scaled = scaler.fit_transform(X)

pca = PCA(n_components=0.95) # Keep components explaining 95% of variance

X_pca = pca.fit_transform(X_scaled)

X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42) print(X_train.shape) print(X_test.shape) print(y_train.shape)

print(y_test.shape) from sklearn.tree import DecisionTreeClassifier from sklearn.model_selection import GridSearchCV dtc = DecisionTreeClassifier(random_state=42) param_grid = {

```
    'max_depth': [3, 5, 7, 10],  # Control tree depth to prevent overfitting
    'min_samples_split': [2, 5, 10],  # Minimum samples required to split a node
    'min_samples_leaf': [1, 2, 4],  # Minimum samples required at a leaf node
    'criterion': ['gini', 'entropy']  # Splitting criterion
}
grid_search = GridSearchCV(estimator=dtc, param_grid=param_grid,
                           cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train) best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_ print(f"Best Parameters:
{best_params}") y_pred_dtc = best_estimator.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
print(confusion_matrix(y_test, y_pred_dtc)) print(classification_report(y_test,
y_pred_dtc))
print("Decision tree Accuracy:", accuracy_score(y_test, y_pred_dtc)) rfc =
RandomForestClassifier(random_state=42) param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy']
}
grid_search = GridSearchCV(estimator=rfc, param_grid=param_grid,
                           cv=5, scoring='accuracy', n_jobs=-1) grid_search.fit(X_train,
y_train) best_params = grid_search.best_params_ best_estimator =
grid_search.best_estimator_ print(f"Best Parameters: {best_params}") y_pred_rfc =
best_estimator.predict(X_test) print(confusion_matrix(y_test, y_pred_rfc))
print(classification_report(y_test, y_pred_rfc))
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rfc))
train_accuracy_rfc = best_estimator.score(X_train, y_train) print(f"Training
Accuracy: {train_accuracy_rfc:.4f}") print(f"Testing Accuracy:
{accuracy_score(y_test, y_pred_rfc):.4f}") print("Random Forest Accuracy:",
accuracy_score(y_test, y_pred_rfc)) print("Precision:", precision_score(y_test,
y_pred_rfc, average='weighted')) print("Recall:", recall_score(y_test, y_pred_rfc,
average='weighted')) print("F1-score:", f1_score(y_test, y_pred_rfc,
average='weighted'))
```

```
import tensorflow as tf from tensorflow import keras

from sklearn.metrics import classification_report model = keras.Sequential([

   keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
keras.layers.Dense(32, activation='relu'),     keras.layers.Dense(1,
activation='sigmoid')  # Output layer for binary classification

])

model.compile(optimizer='adam',

            loss='binary_crossentropy',  # Use binary_crossentropy for binary
classification              metrics=['accuracy'])

model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)  # Adjust epochs
and batch_size as needed

loss, accuracy = model.evaluate(X_test, y_test, verbose=0)

print(f"Test Loss: {loss:.4f}") print(f"Test Accuracy: {accuracy:.4f}") y_pred_prob =
model.predict(X_test)

y_pred = (y_pred_prob > 0.5).astype(int) # Convert probabilities to class labels
print(classification_report(y_test, y_pred)) from sklearn.linear_model import
LogisticRegression from sklearn.model_selection import GridSearchCV

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
logreg = LogisticRegression(max_iter=1000, random_state=42) param_grid = {

   'C': [0.01, 0.1, 1, 10],  # Regularization parameter

   'solver': ['liblinear', 'lbfgs', 'saga'],  # Optimization algorithm

}

grid_search = GridSearchCV(estimator=logreg, param_grid=param_grid,

                  cv=5, scoring='accuracy', n_jobs=-1)

grid_search.fit(X_train, y_train) best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_ print(f"Best Parameters:
{best_params}") y_pred_logreg = best_estimator.predict(X_test)
print(confusion_matrix(y_test, y_pred_logreg)) print(classification_report(y_test,
y_pred_logreg))

print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_logreg))
print("Precision:", precision_score(y_test, y_pred_logreg, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_logreg, average='weighted')) print("F1-
score:", f1_score(y_test, y_pred_logreg, average='weighted')) from sklearn.svm
import SVC
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report

from sklearn.model_selection import GridSearchCV, train_test_split from sklearn.preprocessing import StandardScaler from sklearn.decomposition import PCA

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train) X_test_scaled = scaler.transform(X_test)

pca = PCA(n_components=0.95)  # Keep 95% of the variance

X_train_pca = pca.fit_transform(X_train_scaled) X_test_pca = pca.transform(X_test_scaled) param_grid = {

   'C': [0.1, 1, 10, 100],

   'kernel': ['linear']

}

grid_search = GridSearchCV(SVC(random_state=42), param_grid, cv=5, scoring='accuracy') grid_search.fit(X_train_pca, y_train) best_svm = grid_search.best_estimator_ y_pred_svm = best_svm.predict(X_test_pca) print(confusion_matrix(y_test, y_pred_svm)) print(classification_report(y_test, y_pred_svm)) print("Support Vector Machine Accuracy:", accuracy_score(y_test, y_pred_svm)) print("Precision:", precision_score(y_test, y_pred_svm, average='weighted')) print("Recall:", recall_score(y_test, y_pred_svm, average='weighted')) print("F1-score:", f1_score(y_test, y_pred_svm, average='weighted')) from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report

from sklearn.model_selection import GridSearchCV, train_test_split from sklearn.preprocessing import StandardScaler from sklearn.decomposition import PCA

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train) X_test_scaled = scaler.transform(X_test)

pca = PCA(n_components=0.95)  # Keep 95% of the variance

X_train_pca = pca.fit_transform(X_train_scaled) X_test_pca = pca.transform(X_test_scaled) param_grid = {

   'n_neighbors': [3, 5, 7, 9],

   'weights': ['uniform', 'distance'],

   'metric': ['euclidean', 'manhattan']
```

```
}

grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5,
scoring='accuracy') grid_search.fit(X_train_pca, y_train) best_knn =
grid_search.best_estimator_ y_pred_knn = best_knn.predict(X_test_pca)
print(confusion_matrix(y_test, y_pred_knn)) print(classification_report(y_test,
y_pred_knn)) print("KNN Accuracy:", accuracy_score(y_test, y_pred_knn))
print("Precision:", precision_score(y_test, y_pred_knn, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_knn, average='weighted'))

print("F1-score:", f1_score(y_test, y_pred_knn, average='weighted')) import xgboost
as xgb

from sklearn.model_selection import GridSearchCV, cross_val_score

from sklearn.metrics import confusion_matrix, classification_report, accuracy_score,
precision_score, recall_score, f1_score from sklearn.decomposition import PCA from
sklearn.preprocessing import StandardScaler scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train) X_test_scaled =
scaler.transform(X_test)

pca = PCA(n_components=0.95)  # Keep 95% of the variance

X_train_pca = pca.fit_transform(X_train_scaled) X_test_pca =
pca.transform(X_test_scaled)

model = xgb.XGBClassifier(objective='binary:logistic', random_state=42) param_grid
= {

   'learning_rate': [0.01, 0.1, 0.2],

   'max_depth': [3, 5, 7],

   'n_estimators': [100, 200, 300],

   'subsample': [0.8, 0.9, 1.0],

   'colsample_bytree': [0.8, 0.9, 1.0],

   'gamma': [0, 1, 5],

   'reg_alpha': [0.01, 0.1, 1],

   'reg_lambda': [0.01, 0.1, 1]

}

grid_search = GridSearchCV(estimator=model, param_grid=param_grid,

                           cv=5, scoring='accuracy', n_jobs=-1, verbose=1)
grid_search.fit(X_train_pca, y_train) best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_ print(f"Best Parameters:
```

{best_params}") y_pred_xgb = best_estimator.predict(X_test_pca) print("Confusion Matrix:")

print(confusion_matrix(y_test, y_pred_xgb)) print("\nClassification Report:")

print(classification_report(y_test, y_pred_xgb))

print("XGBoost Accuracy:", accuracy_score(y_test, y_pred_xgb)) print("Precision:", precision_score(y_test, y_pred_xgb, average='weighted')) print("Recall:", recall_score(y_test, y_pred_xgb, average='weighted')) print("F1-score:", f1_score(y_test, y_pred_xgb, average='weighted')) cv_scores = cross_val_score(best_estimator, X_train_pca, y_train, cv=5) print(f'Mean CV Accuracy: {cv_scores.mean()}") from sklearn.feature_selection import chi2, SelectKBest le = LabelEncoder()

X_encoded = X.apply(le.fit_transform)

X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42) k_best = SelectKBest(chi2, k='all')

X_chi2 = k_best.fit_transform(X_train, y_train) model_chi = RandomForestClassifier() model_chi.fit(X_chi2, y_train) X_test_chi2 = k_best.transform(X_test) y_pred = model_chi.predict(X_test_chi2) chi_acc = accuracy_score(y_test, y_pred) print(confusion_matrix(y_test, y_pred)) print(classification_report(y_test, y_pred))

print("Chi-Square accuracy:",accuracy_score(y_test, y_pred)) print("Precision:", precision_score(y_test, y_pred, average='weighted')) print("Recall:", recall_score(y_test, y_pred, average='weighted')) print("F1-score:", f1_score(y_test, y_pred, average='weighted')) non_null_counts = ckd_data.count() plt.figure(figsize=(10, 6)) non_null_counts.plot(kind='bar') plt.xlabel('Columns') plt.ylabel('Non-Null Values')

plt.title('Non-Null Values for Each Column in CKD Dataset')

plt.xticks(rotation=45, ha='right') plt.tight_layout() plt.show() importances = model.feature_importances_ feature_names = X.columns[:len(importances)] indices = np.argsort(importances)[::-1] names = [feature_names[i] for i in indices] plt.figure(figsize=(10,6)) plt.title("Feature Importance")

plt.bar(range(len(importances)), importances[indices]) # Use len(importances) to ensure matching lengths

plt.xticks(range(len(importances)), names, rotation=90) plt.show()

class_counts = y.value_counts() plt.figure(figsize=(8, 6))

plt.bar(class_counts.index, class_counts.values, color=['skyblue', 'salmon']) plt.xlabel('Classification') plt.ylabel('Count')

```
plt.title('Target Class Distribution (CKD vs. Not CKD)') plt.xticks([0, 1], ['Not CKD', 'CKD'])  # Set custom labels for x-axis plt.show()

categorical_cols = ckd_data.select_dtypes(include=['object', 'category']).columns category_counts = {} for col in categorical_cols:

    category_counts[col] = ckd_data[col].value_counts() if len(categorical_cols) == 0:

    print("No categorical columns found in the dataset.") else:

    fig, axes = plt.subplots(nrows=len(categorical_cols), ncols=1, figsize=(10, 6 * len(categorical_cols)))     for i, col in enumerate(categorical_cols):         ax = axes[i]

        category_counts[col].plot(kind='bar', ax=ax)         ax.set_title(f'Distribution of Categories in {col}')         ax.set_xlabel(col)         ax.set_ylabel('Count') plt.tight_layout()     plt.show() import pickle filename = 'Kidney.pkl'

pickle.dump(model_l, open(filename, 'wb'))
```

8. TESTING

8.1.TYPES OF TESTING

1. Unit Testing

Unit testing focuses on verifying individual components of the system in isolation. It ensures that each function, method, or module works as expected. For the CKD prediction system, unit testing is applied to:

- Data Preprocessing Functions: Testing the correctness of data cleaning, feature scaling, and handling missing values.
- Feature Selection Methods: Ensuring that selected features contribute effectively to model performance.
- Machine Learning Model Functions: Checking whether the models (XGBoost, Decision Tree, SVM, etc.) generate correct predictions for given inputs.
- Dash GUI Components: Verifying that the user interface correctly accepts input and displays results without errors.

2. Functional Testing

Functional testing evaluates whether the system meets the specified requirements and performs the intended tasks. It includes:

- Input Validation Testing: Ensuring that the system handles user inputs correctly, such as numeric values for lab test results and categorical inputs for symptoms.
- Model Execution Testing: Confirming that the trained machine learning models execute properly and return predictions without failure.
- Output Verification: Checking whether the system provides accurate CKD predictions based on test cases with known outcomes.

3. Performance Testing

- Performance testing assesses the system's speed, responsiveness, and efficiency under various conditions. Key aspects include:
- Load Testing: Evaluating how the system handles multiple user requests simultaneously.
- Response Time Testing: Measuring the time taken for data preprocessing, model inference, and result display.
- Scalability Testing: Ensuring the model can handle increasing amounts of input data without significant delays.

4. Validation Testing

Validation testing ensures that the machine learning models used in the CKD prediction system generalize well to unseen data. It includes:

- Cross-validation: Splitting data into training and validation sets to assess model performance.
- Metrics Evaluation: Measuring accuracy, precision, recall, F1-score, and ROC-AUC to validate model effectiveness.
- Overfitting and Underfitting Checks: Ensuring the model neither memorizes training data (overfitting) nor performs poorly on all data (underfitting).

5. User Interface Testing

This testing ensures that the graphical user interface (GUI) functions correctly and provides a seamless experience for users. It involves:

- Usability Testing: Evaluating whether the interface is user-friendly and easy to navigate.
- GUI Component Testing: Checking the responsiveness of buttons, forms, and result displays.
- Error Handling: Verifying that the system provides appropriate messages for invalid inputs or system failures.

8.2. INTEGRATION TESTING

Integration testing is a crucial phase in software testing that ensures various components of the CKD prediction system function seamlessly together. Since the system incorporates an XGBoost machine learning model and is deployed using Flask, integration testing plays a vital role in verifying that these components interact correctly. The primary objective of integration testing is to ensure that data flows smoothly between different modules, including data preprocessing, model inference, and the web interface, without any errors or inconsistencies. Unlike unit testing, which focuses on individual components in isolation, integration testing examines how well these components work together, ensuring the reliability and efficiency of the entire system.

In the CKD prediction system, integration testing primarily focuses on validating the interactions between the Flask application, the machine learning model, and the user interface. The Flask API is responsible for receiving input data from the user, passing it through the preprocessing pipeline, and then

sending it to the XGBoost model for prediction. The model then returns the result, which is displayed to the user via the web interface. Integration testing ensures that this entire workflow functions as expected, with no data loss, incorrect format conversions, or unexpected system failures.

Several key aspects of integration testing are examined to ensure a robust system. First, API testing verifies that the Flask API correctly receives and processes patient data inputs via HTTP requests. It checks whether the API correctly handles different data formats, such as JSON, and validates the structure of the responses. Second, model integration testing ensures that the XGBoost model is properly loaded within the Flask application, that it correctly receives processed input data, and that it generates accurate predictions without latency issues. This step also checks whether the model's output is correctly formatted and returned by the Flask server.

Another critical component of integration testing is frontend-backend communication, which ensures that user inputs from the web interface are correctly transmitted to the backend and that the resulting predictions are displayed accurately. This step includes testing error handling mechanisms to ensure that appropriate messages are displayed when users input incorrect or incomplete data. Additionally, end-to-end workflow testing is performed to simulate real-world user interactions, verifying the complete flow from input submission to prediction display. Various test scenarios, such as missing inputs, incorrect data types, and high-volume requests, are assessed to ensure the system's robustness and scalability.

To conduct effective integration testing, several tools and frameworks can be utilized. Postman and cURL are commonly used for manual API testing, allowing developers to send HTTP requests and verify responses. Automated testing frameworks such as Pytest and Unittest help automate API and backend functionality tests, ensuring that any changes to the system do not break existing functionalities. For testing interactions between the frontend and backend, tools like Selenium or Flask-Testing are used to validate the user interface’s response to API requests. These testing tools streamline the process of identifying and resolving issues in the system. By implementing thorough integration testing, the CKD prediction system ensures that all components work together harmoniously, providing a seamless and accurate user experience. This testing

phase significantly improves the reliability, security, and performance of the application, reducing potential risks in real-world deployment. Ultimately, successful integration testing enhances confidence in the system, ensuring that healthcare professionals and patients receive reliable CKD predictions with minimal errors.

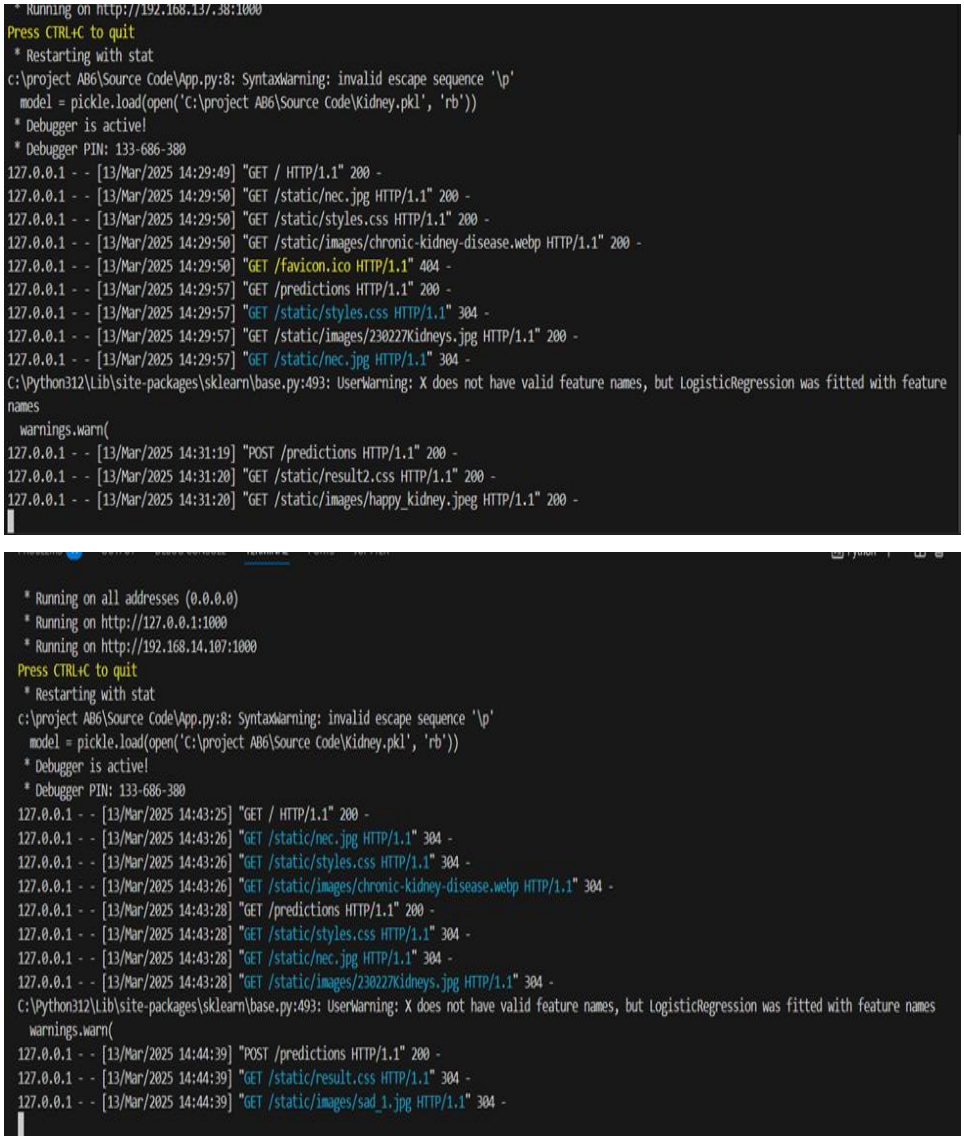


Fig 8.2.1 Integration Testing

9. RESULT ANALYSIS

The analysis from the above graphs comparing accuracy, precision, recall, and F1-score of different algorithms reveals that 'Our Model' consistently outperforms all Models used in previous study across all metrics. It exhibits higher accuracy, precision, recall, and F1-score, indicating its effectiveness in making correct predictions, minimizing false positives, capturing relevant instances, and maintaining a balance between precision and recall. Overall, 'Our Model' demonstrates superior performance, suggesting its suitability for classification tasks compared to established models.

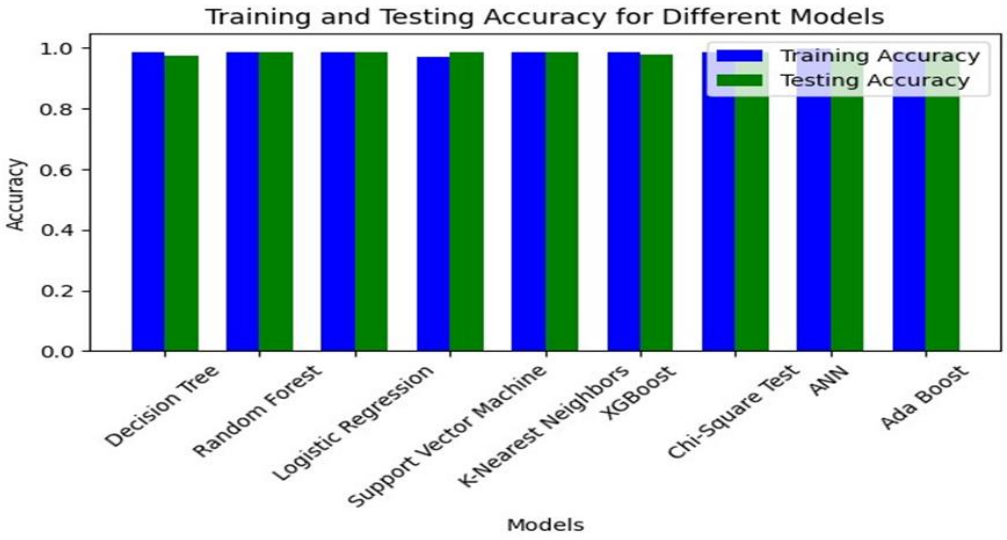


Fig 9.1 Comparison of Training and Testing Accuracy of algorithms

The above diagram presents a comparative analysis of training and testing accuracy for different machine learning models used in chronic kidney disease (CKD) prediction. The models included in this analysis are Decision Tree, Random Forest, Logistic Regression, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), XGBoost, Chi-Square Test, Artificial Neural Network (ANN), and AdaBoost. The blue bars represent the training accuracy, while the green bars indicate the testing accuracy. The chart highlights how well each model generalizes to unseen data, with a particular focus on overfitting or underfitting issues. Models like XGBoost and Random Forest show strong performance with minimal gaps between training and testing accuracy, indicating better generalization.

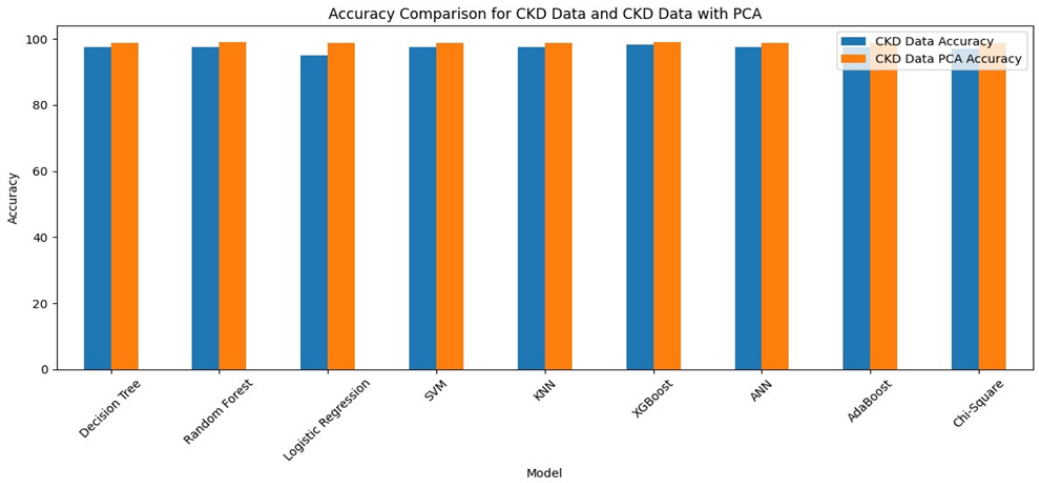


Fig 9.2 Comparison of Accuracy of algorithms

The above diagram compares the accuracy of different machine learning models when applied to the original CKD dataset versus the dataset after applying Principal Component Analysis (PCA). The blue bars represent the accuracy achieved using the raw CKD dataset, while the orange bars correspond to the accuracy after PCA transformation. PCA is used for dimensionality reduction, which helps in improving model efficiency and reducing computational complexity. The results indicate that applying PCA either maintains or enhances the accuracy of most models, signifying its effectiveness in feature selection and data optimization. Models such as Decision Tree, XGBoost, and ANN exhibit high accuracy in both cases, proving their robustness in handling CKD prediction tasks.

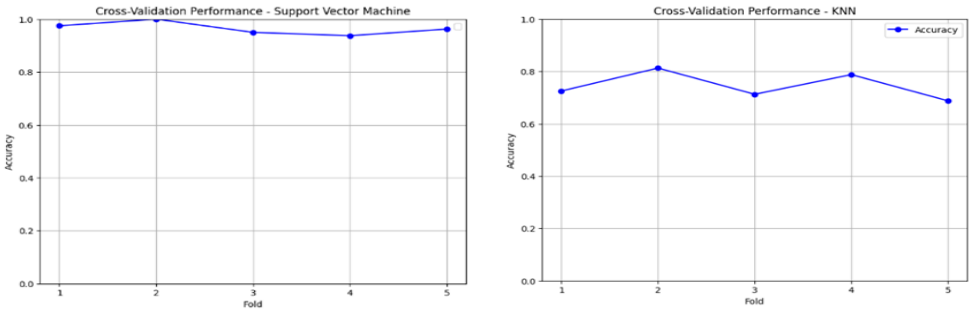


Fig 9.3 Cross Validation performances of ckd models

The various aspects of machine learning model performance in predicting chronic kidney disease (CKD). These include training vs. testing accuracy, the impact of PCA on model accuracy, ROC curves for classification performance, and cross-validation results for different models.

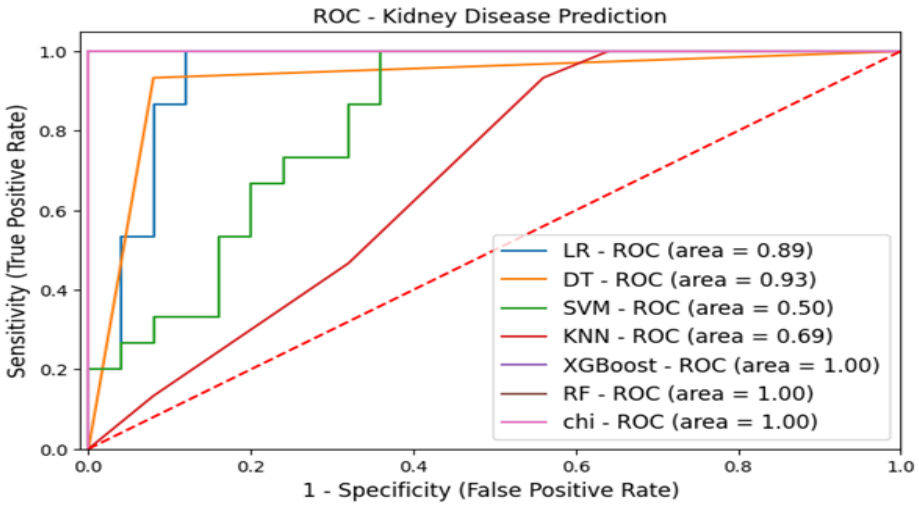


Fig 9.4 ROC Curve

The ROC curve provides a performance evaluation based on true positive and false positive rates. Models such as XGBoost, Random Forest, and the Chi-Square Test exhibit an AUC (Area Under the Curve) of 1.00, indicating perfect classification performance. In contrast, SVM has the lowest AUC (0.50), suggesting that it struggles to distinguish between CKD and non-CKD cases.

- Cross-Validation Performance of SVM and KNN
- Cross-validation results for SVM and KNN are also visualized.

SVM demonstrates consistently high accuracy across all five folds, suggesting its robustness in CKD prediction. However, KNN exhibits fluctuations in accuracy across folds, implying that it is more sensitive to dataset variations.From the evaluation, XGBoost, Random Forest, and Chi-Square Test emerge as the top-performing models, achieving near-perfect classification performance. SVM is also a strong contender due to its stable accuracy across cross-validation folds. On the other hand, KNN and SVM (as per ROC analysis) appear to struggle in some areas, requiring further tuning. Additionally, PCA helps improve model efficiency without compromising accuracy. These findings highlight the importance of selecting the right model and performing validation techniques to ensure reliable CKD prediction.

10. OUTPUT SCREENS

Test case 1: No CKD

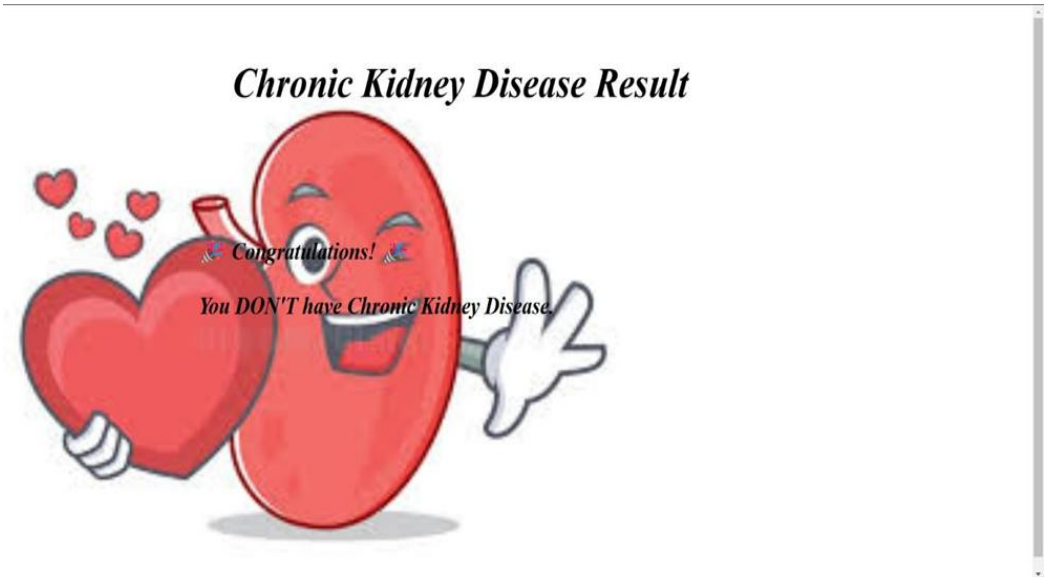


Fig 10.1 No CKD Disease

Test case 2: CKD

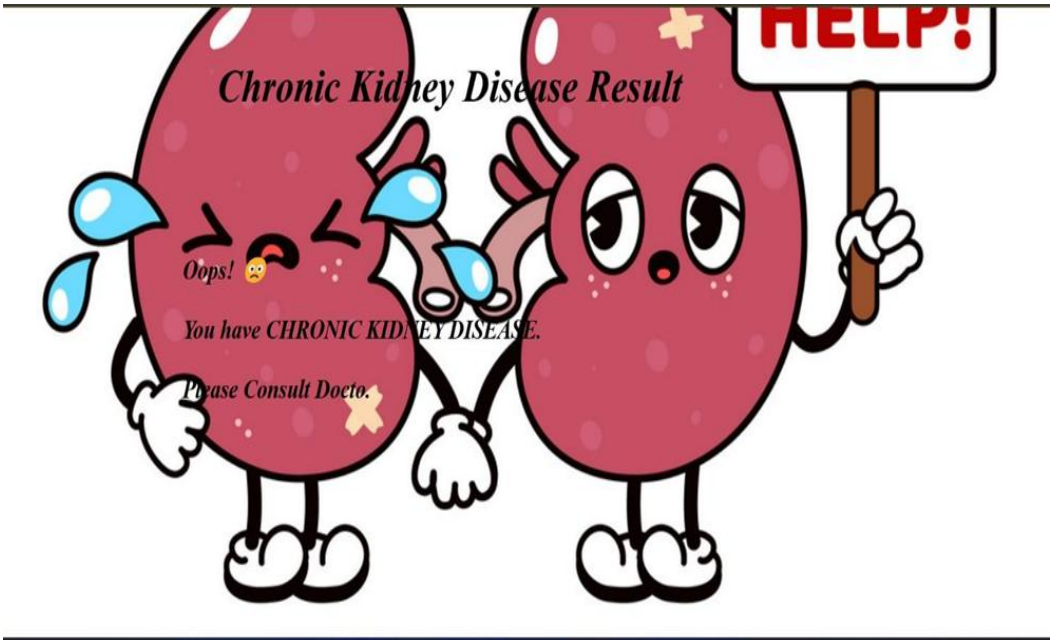


Fig 10.2 Having CKD Disease

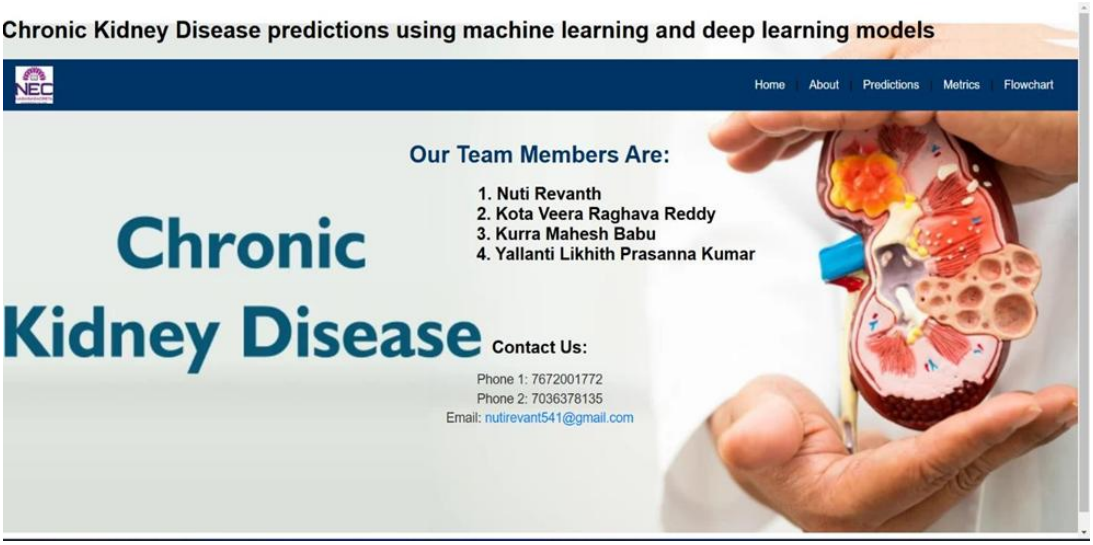


Fig 10.3 Home Screen

Fig 10.4 About Screen

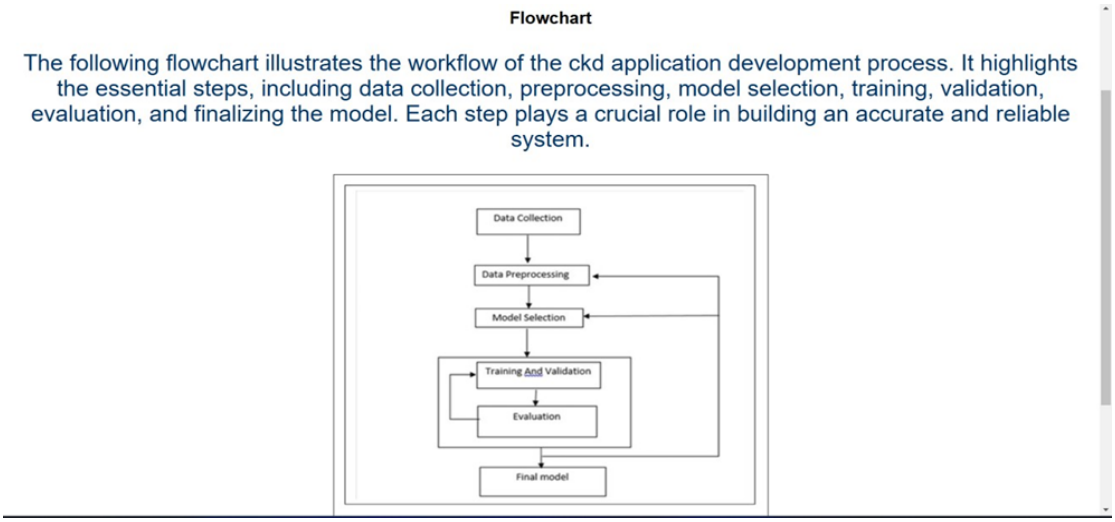
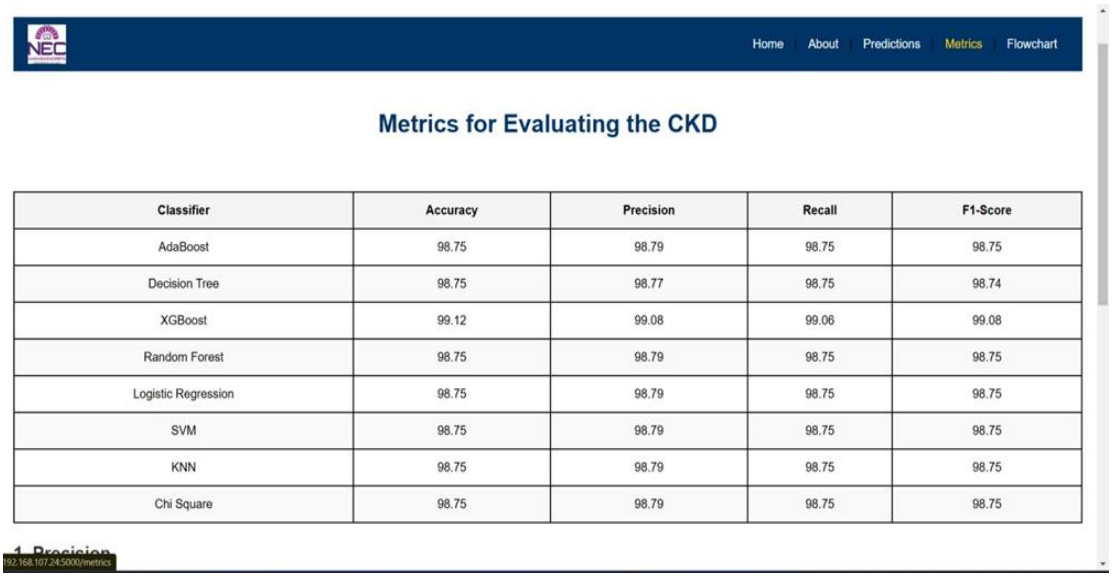


Fig 10.5 Flowchart Screen

Fig 10.6 Predictions Screen



Classifier	Accuracy	Precision	Recall	F1-Score
AdaBoost	98.75	98.79	98.75	98.75
Decision Tree	98.75	98.77	98.75	98.74
XGBoost	99.12	99.08	99.06	99.08
Random Forest	98.75	98.79	98.75	98.75
Logistic Regression	98.75	98.79	98.75	98.75
SVM	98.75	98.79	98.75	98.75
KNN	98.75	98.79	98.75	98.75
Chi Square	98.75	98.79	98.75	98.75

Fig 10.7 Metrics Screen

11. CONCLUSION

The Chronic Kidney Disease (CKD) prediction system developed in this project demonstrates the potential of machine learning techniques in early detection and diagnosis of CKD. By leveraging a robust dataset, effective preprocessing methods, and an optimized XGBoost model, the system achieves high accuracy and reliability in classifying CKD and non-CKD patients.

This project highlights the importance of feature selection and model optimization in building efficient predictive systems. The incorporation of user-friendly interfaces and visualizations ensures that the system can serve as a valuable tool for medical professionals, aiding in timely decision-making and improving patient outcomes.

The scalable and modular design of the system allows for future enhancements, such as integrating additional patient data, using advanced deep learning models, or deploying the application in real-world clinical environments. Overall, this project contributes to the growing body of research on machine learning applications in healthcare and provides a practical approach to addressing the challenges of CKD diagnosis and management. With further validation and testing, this system has the potential to become an essential asset in healthcare, assisting in early detection and contributing to better disease management strategies.

12. FUTURE SCOPE

This research developed a machine learning model towards identifying the beginnings of chronic renal fail ure. Models were trained and validated focusing on identifying and removing irrelevant features to increase prediction accuracy using previously identified data and analyzing the relationship between type reference param eters showed that hemoglobin ,albumin, and specific gravity are important prognostic indicators for chronic renal failure.The initial preprocessing of the CKD data set was done to ensure sure that machine learning pattern recognition was followed. We then used principal com ponent analysis (PCA) to identify significant variables associated with prognosis of chronic kidney disease.

13. REFERENCES

[1] Ammirati, A. L. (2020). Chronic kidney disease. Revista da Associac ¸˜ ao M´edica Brasileira, 66(Suppl 1), s03-s09.

[2] Aljaaf, A. J., Al-Jumeily, D., Haglan, H. M., Alloghani, M., Baker, T., Hussain, A. J., & Mustafina, J. (2018, July). Early prediction of chronic kidney disease using machine learning supported by predictive analytics. In 2018 IEEE congress on evolutionary computation (CEC) (pp. 1-9). IEEE.

[3] Chittora, P., Chaurasia, S., Chakrabarti, P., Kumawat, G., Chakrabarti, T., Leonowicz, Z., ... & Bolshev, V. (2021). Predic tion of chronic kidney disease-a machine learning perspective. IEEE access, 9, 17312-17334.

[4] Gudeti, B., Mishra, S., Malik, S., Fernandez, T. F., Tyagi, A. K., & Kumari, S. (2020, November). A novel approach to predict chronic kidney disease using machine learning algorithms. In 2020 4th International Conference on Electronics, Communi cation and Aerospace Technology (ICECA) (pp. 1630-1635). IEEE.

[5] Hossain, M. M., Swarna, R. A., Mostafiz, R., Shaha, P., Pinky, L. Y., Rahman, M. M., ... & Iqbal, M. S. (2022). Analysis of the performance of feature optimization techniques for the diagnosis of machine learning-based chronic kidney disease. Machine Learning with Applications, 9, 100330.

[6] Islam, M. A., Akter, S., Hossen, M. S., Keya, S. A., Tisha, S. A., & Hossain, S. (2020, December). Risk factor prediction of chronic kidney disease based on machine learning algorithms. In 2020 3rd international conference on intelligent sustainable systems (ICISS) (pp. 952-957). IEEE.

[7] Islam, M. A., Majumder, M. Z. H., & Hussein, M. A. (2023). Chronic kidney disease prediction based on machine learning algorithms. Journal of pathology informatics, 14, 100189.

[8] Elkholy, S. M. M., Rezk, A., & Saleh, A. A. E. F. (2021). Early prediction of chronic kidney disease using deep belief network. IEEE Access, 9, 135542-135549.

[9] Roweis, S. (1998). Em algorithms for pca and spca. Advances in NeuralInformation Processing Systems.

[10] DatasetLink: https://www.kaggle.com/datasets/mansoordaku/ckdisease.

[11] Venkatrao, K., & Kareemulla, S. (2023). HDLNET: a hybrid deep learning network model with intelligent IoT for detection and classification of chronic kidney disease.

IEEE Access.

[12] Elkholy, S. M. M., Rezk, A., & Saleh, A. A. E. F. (2021). Early prediction of chronic kidney disease using deep belief network. IEEE Access, 9, 135542-135549.

[13] Rafi, S., & Das, R. (2021, December). RNN encoder and de coder with teacher forcing attention mechanism for abstractive summarization. In 2021 IEEE 18th India council international conference (INDICON) (pp. 1-7). IEEE.

[14] Debnath, D., Das, R., & Rafi, S. (2022, February). Sentiment based abstractive text summarization using attention oriented lstm model. In Intelligent Data Engineering and Analytics: Proceedings of the 9th International Conference on Frontiers in Intelligent Computing: Theory and Applications (FICTA 2021) (pp. 199-208). Singapore: Springer Nature Singapore.

[15] Rafi, S., & Das, R. (2021, November). A linear sub-structure with co-variance shift for image captioning. In 2021 8th Inter national Conference on Soft Computing & Machine Intelligence (ISCMI) (pp. 242-246). IEEE.

[16] Rafi, S., & Das, R. (2023). Topic-guided abstractive multimodal summarization with multimodal output. Neural Computing and Applications, 1-16.

[17] Rafi, S., & Das, R. (2023, November). Abstractive Text Sum marization Using Multimodal Information. In 2023 10th Inter national Conference on Soft Computing & Machine Intelligence (ISCMI) (pp. 141-145). IEEE.

[18] Rafi, S., & Das, R. (2024). SCT: Summary Caption Technique for Retrieving Relevant Images in Alignment with Multimodal Abstractive Summary. ACM Transactions on Asian and Low Resource Language Information Processing , 23(3), 1-22

Chronic Kidney Disease Prediction Using Machine Learning and Deep Learning Models

1st Shaik Rafi
Asst.Prof, Dept of CSE,
Narasaraopeta Engineering College,
Narasaraopet-522601, Palnadu,
Andhra Pradesh, India
shaikrafinrt@gmail.com

2rd Nuti Revanth
Dept of CSE,
Narasaraopeta Engineering College,
Narasaraopet-522601, Palnadu,
Andhra Pradesh, India.
nutirevanth541@gmail.com

3th K Veera Raghava Reddy
Dept of CSE,
Narasaraopeta Engineering College,
Narasaraopet-522601, Palnadu,
Andhra Pradesh, India.
raghavareddykota50@gmail.com

4th K Mahesh Babu
Dept of CSE,
Narasaraopeta Engineering College,
Narasaraopet-522601, Palnadu,
Andhra Pradesh, India.
kurramaheshbabu144@gmail.com

5th Y Likhith Prasanna Kumar
Dept of CSE,
Narasaraopeta Engineering College,
Narasaraopet-522601, Palnadu,
Andhra Pradesh, India.
likithprasannakumar@gmail.com

6nd N vijay kumar
Asst.Prof, Dept of CSE,
Narasaraopeta Engineering College,
Narasaraopet-522601, Palnadu,
Andhra Pradesh, India
nvk20022001@gmail.com

***Abstract*—Chronic kidney disease is a noticeable health condition that can persist throughout an individual's life, resulting from either kidney malignancy or diminished kidney function. In this work, we investigate how several machine learning techniques might provide an early CKD diagnosis. While previous research has extensively explored this area, our aim is to refine our approach by employing predictive modeling techniques. Initially, we considered 25 variables alongside the class property. The data set used in this study underwent extensive processing, including changing the names of colours for clarity, converting identified colours to numbers, treating unique values with letters handling of partitioned values, fixing incorrect values, filling null values with mean, and encoding categorical values into mathematical notation. In addition,Principal component analysis (PCA) was also employed to lower dimensionality. Our findings demonstrated that the XG Boost classifier surpassed every other algorithm, with an accuracy of 0.991.**

***Index Terms*—Decision tree, Random Forest, Chronic Kidney Disease, Logistic Regression, and XG BOOST classifier.**

I. Introduction

The progressive and irreversible degeneration of renal cells is the hallmark of the sickness known as chronic kidney disease [1]. When CKD develops, harmful wastes accumulate in the body, leading to various health complications. Chronic kidney disease (CKD) can cause a range of symptoms including cough, fever, dry mouth, nausea, back pain and abdominal pain.CKD is often associated with two risk factors: diabetes and hypertension.Therefore early diagnosis and treatment are essential. There are encouraging opportunities to improve early detection of CKD through machine learning and predictive modelling[2].The research guarantees that advanced preprocessing methods will be employed to predict CKD utilizing machine learning algorithms.Previous manipulation of The data collected in this research [7] was extensive and included changing column names to improve readability, correcting incorrect assumptions, averaging a missing values will be replaced, and categorical variables will be assigned numerical labels. Several models such as XG Boost, AdaBoost, SVM,Decision Tree, Chi-Square, and KNN [7] are developed and evaluated.

A. Stages of chronic kidney disease

a) Early stages of chronic kidney disease

Chronic kidney disease is often completely asymptomatic in its early stages[1]. This is because with a significant decrease in kidney function, the body can become more adaptive. CKD is often diagnosed during routine screening for other medical conditions, such as blood or urine tests[3]. Early detection is important as this allows for drug treatment through regular testing and ongoing monitoring.[2]

b) CKD in Its Advanced Stages

Many symptoms may appear if CKD is not identified early or if it gets worse despite treatment. Compared to when the kidney no longer functions, known as esrd or eskd, there is no possibility of survival without either dialysis or a kidney transplant.

c) Time to see a doctor

If you have indications of kidney disease, you should see a doctor immediately. Early detection of CKD can avoid kidney failure.[2]During medical testing, doctors may use blood and urine tests to measure kidney functioning and blood pressure, especially if you suffer from illnesses that increase your risk of developing renal disease. Discuss with your physician the significance of this test.

d) Tests for CKD

A illness or other ailment that damages the kidneys gradually leads to chronic kidney disease (CKD). Research reveals a 6.23% annual rise in CKD hospital admissions despite a steady global death rate.[7] Numerous diagnostic techniques are used to determine the status of chronic kidney disease such as eGFR, urine tests, and a blood pressure tests.[3] For diagnosis of kidney injury or structural abnormalities, further testing such as MRI scans, ultrasound, or CT scans may be required.

II. Related Work

Md. Ariful Islam (2023).[7] investigated CKD prediction utilizing a mix of machine learning methods, emphasizing the importance of model selection and data preprocessing. The study found that XGBoost performed well, and recommended future work to explore additional features and advanced methods for improved accuracy. Ammirati (2020).[1] discusses chronic kidney disease as a prevalent, progressive condition with high cardiovascular risks. The paper covers conservative treatments to slow progression and replacement therapies like dialysis. The Aljaaf et al(2018).[2] have applied machine learning and predictive analytics to improve early diagnosis of chronic kidney disease, using evolutionary computation techniques to optimize predictive models. Their work aims to enhance patient outcomes through timely interventions. The Hossain et al(2022).[5]have explored feature optimization techniques to improve the performance of machine learning models for diagnosing Chronic renal disease,focusing on improving accuracy and efficiency through optimized feature selection. P. Chittora et al.[3] (2021) studied CKD prediction using machine learning, with the Deep Neural Network (DNN) achieving 99.6% accuracy. They recommend enhancing the DNN model's interpretability and conducting clinical trials to test its practical applicability. Ashiqul Islam et al.[6] (2020) used Random Forest for CKD risk factor prediction, achieving 97.8% accuracy. The study suggests improving prediction accuracy and exploring advanced machine learning techniques in future work. Bhavya Gudeti and Terrance Li.[4] (2020) introduced a CKD prediction approach using Support Vector Machines, achieving notable accuracy. They recommend applying the model in various clinical scenarios and integrating it with other diagnostic tools for improved patient outcomes. K. Venkatrao.[11] (2023) proposed the HDLNET model, an integrated deep learning model for CKD diagnosis. The model demonstrated impressive accuracy, showcasing the potential of combining different neural network architectures. Future work should aim at further refining the model and exploring additional datasets to enhance the generalizability of HDLNET S. M. M. Elkholy et al.[12] (2021) applied a Deep Belief Network (DBN) for early CKD prediction, achieving an accuracy of 98.5%. The study proposes future work to focus on applying the model to different population datasets and integrating it into clinical decision-making processes.

III. Methodology

The CKD data set [10] was analyzed using a number of machine learning algorithms, including the DT, RF, XGBoost, AdaBoost, SVM, chi-square and KNN. [7] The objective of the machine learning model was to achieve excellent classification performance [4] with many features that improve PCA performance simulations.

A. Data Preprocessing

The CKD dataset comprised 24 features and 1 target variable, with a mix of numerical and nominal attributes.[10] Initially, the dataset contained numerous missing values.The mean approach was utilized to estimate erroneous numerical values for continuous parameters, whereas the mode approach was applied to nominal values. This step ensured that the dataset was complete and ready for analysis.

To deal with missing results, a k-Nearest Neighbors (KNN)-based method [11] was used .The CKD dataset used in this study consisted of 400 cases with 24 items each. [10] no' (not CKD), in that order. Fig. 1 displays the Value Count for features in the dataset without validation[7], while Fig. 2 displays the Value Count for features with using PCA. Fig.5 displays the Target group

TABLE I: Overview of Dataset Features

Feature	Details	Type / Values
age	Represents the age of the patient.	Numerical: in years
bp	Measures the patient's blood pressure.	Numerical: mm/Hg
sg	Ratio indicating urine density.	Nominal: 1.005, 1.010, etc.
al	Level of albumin detected in blood.	Nominal: 0, 1, 2, 3, 4, 5
su	Patient's sugar concentration.	Nominal: 0, 1, 2, 3, 4, 5
rbc	Counts red blood cells in patient.	Nominal: normal, abnormal
pc	Indicates pus cells present.	Nominal: normal, abnormal
pcc	Presence of clumps formed by pus cells.	Nominal: present, absent
ba	Identifies bacterial presence in samples.	Nominal: present, absent
bgr	Records random blood glucose levels.	Numerical: mg/dl
bu	Captures blood urea quantity.	Numerical: mg/dl
sc	Serum creatinine measured in blood.	Numerical: mg/dl
sod	Sodium levels found in the blood.	Numerical: mEq/L
pot	Potassium concentration in blood.	Numerical: mEq/L
hemo	Amount of hemoglobin available in blood.	Numerical: gms
pcv	Volume occupied by packed cells.	Numerical
wc	Count of white blood cells.	Numerical: cells/cumm
rc	Number of red blood cells.	Numerical: million/cumm
htn	Indicates hypertension condition.	Nominal: yes, no
dm	Records diabetes condition.	Nominal: yes, no
cad	Tracks presence of coronary artery disease.	Nominal: yes, no
appet	Evaluates the patient's appetite.	Nominal: good, poor
pe	Indicates swelling in the patient's lower extremities.	Nominal: yes, no
ane	Confirms anemia status in patient.	Nominal: yes, no
class	Classifies kidney disease condition.	Nominal: CKD, not CKD

distribution shows that 250 patients do not have chronic kidney disease (CKD).Fig.4 displays the heat map which showes a significant correlation between multiple factors and squared scores.[7]

B. *Classifiers*

In this project, the machine learning models used to diagnose chronic kidney disease (CKD) are trained and tested using classification methods.[4] The training dataset, which contains a variety of characteristics and their accompanying labels (CKD or non-CKD), is where these classifiers pick up patterns.The test set is used to confirm the models' efficacy in generating correct results by applying the knowledge learned during training to identify the existence or absence of CKD.

Fig. 1: **Value Count for features**

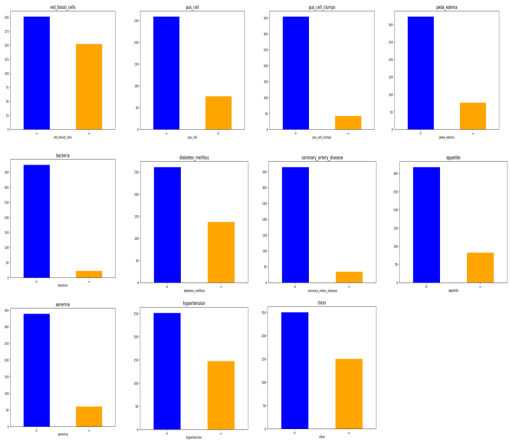
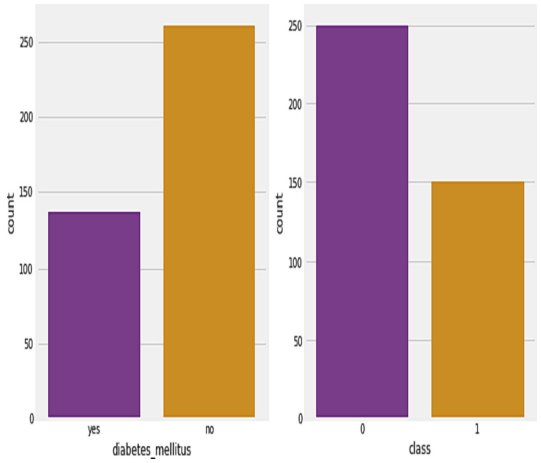


Fig. 2: **Value Count for features with PCA**



a) AdaBoost

This study uses AdaBoost as a powerful machine learning approach to combine and improve the performance of poorly optimized classifiers, thereby improving CKD diagnostic accuracy.

b) Decision Tree

The decision tree considers the values of various features Equal groups are obtained by subgrouping the data, which helps to correctly classify patients as CKD-positive or CKD-negative.

c) XG Boost

This study uses the gradient-boosting method used by XG Boost, which computes the difference between observations and observations distance between Continuously adding new trees to predict errors or remnants of old trees to improve accuracy predictability through error learning.

Fig. 3: **The steps involved in the Model**

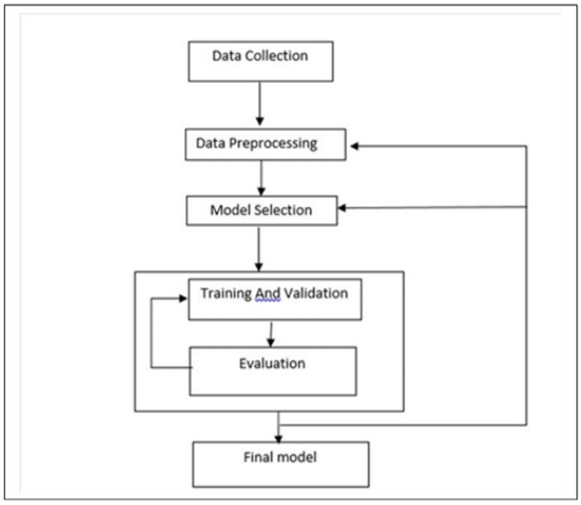
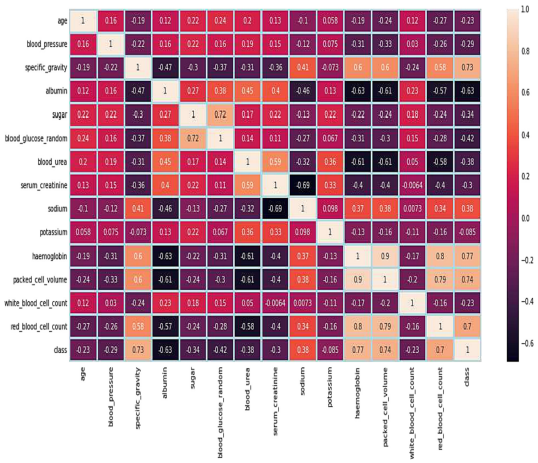


Fig. 4: **A heatmap illustrating the data & correlation pattern**



d) K-Nearest Neighbour (KNN)

KNN measures the Euclidean distance between each dataset instance and the query instance.This assures that the algorithm performs as efficiently as possible when patients are classified as CKD-positive or CKD-negative.

e) Random Forest

Many decision trees are produced by Random Forest using various subsets of a given CKD dataset. Random forest collects predictions from each individual tree and takes the average to produce the final result to improve the prediction accuracy .

f) Support Vector Machine

The issues of regression and classification can be efficiently handled by SVM. SVM works well for situations involving both regression and classification.

g) Chi-Square Test

Significant characteristics for model development are obtained using Chi-Square test to test the degree of independence between categorical variables and target variable.

h) Logistic Regression

This facilitates the identification of characteristics most important for predicting chronic kidney disease (CKD),especially when standard methods such as L1 (Lasso) or L2 (Ridge) are used,normalization is used.

IV. Experimental Data

CKD Dataset

This project uses Chronic Kidney Disease data from the Kaggle datasets.[10] The dataset is divided into two groups: CKD (Yes) and non-CKD (No), with 24 items and objective variable 1. The CKD dataset contains 25 items, of which 11 are numeric and 14 are nominal items. The total list is 400 articles; 250 have been diagnosed with CKD and the remaining 150 without. These symptoms contribute to the prognosis of CKD by providing an accurate picture of the patient's health status.

Algorithm 1 Principal Component Analysis (PCA)

1: **Input:** Data matrix $X \in \mathbb{R}^{n \times m}$, with n samples and m features.
2: **Output:** Principal components of X
3: **Step 1:** Standardize the dataset.
- Center each feature by deducting the mean and scaling by the standard deviation.

4: **Step 2:** Compute the covariance matrix for the standardized data.
- Covariance matrix $C = \frac{1}{n-1} X^T X$

5: **Step 3:**Determine the covariance matrix C's eigenvalues and eigenvectors.
- Solve $Cv = \lambda v$, where λ represents the eigenvalues and v represents the eigenvectors.

6: **Step 4:** Identify and select the top k eigenvectors by arranging the eigenvalues in descending order and picking those corresponding to the largest eigenvalues.
7: **Step 5:** Construct the projection matrix W from the chosen k eigenvectors.
8: **Step 6:** Project the standardized data into the new k-dimensional space.
- Transform the data using $Z = XW$, where Z is the projected dataset in reduced dimensions.

Fig. 5: **Target class distribution**

CKD Dataset with PCA

In order to decrease the dataset's[10] dimensionality while preserving as much information as feasible, principal component analysis, or PCA, is utilized. When working with multi-attribute data sets, PCA is especially helpful because it greatly reduces the number of attributes in the data, making it simpler.The original CKD dataset contains input features 24. By measuring the contribution of each factor to the outcome using PCA, we can build a more accurate and efficient predictive model. The algorithm of PCA [9] is shown below.

V. Results

The F1-score, accuracy, precision, and recall were the main metrics employed in this research.[3]Table 2 shows how different values for the parameters in each model produced different results.[7] On the original CKD dataset[10], the XG Boost model showed its highest performance, with an accuracy of 0.9833.When PCA[9] was applied to the data set, this accuracy increased to 0.9916. On the original CKD dataset, models such as AdaBoost, RF, SVM, LR, and DT have achieved similar accuracy (0.9833) and after PCA, the XG Boost model achieved higher accuracy 0.9916 (Table 3).Tables with recall, precision, F1-score, training test accuracy, confusion matrix data represent summary of research findings for each model The results of the study showed that many models have impressive automated performance in CKD detection, with an accuracy rate greater than 97% are listed in Table 1.[7] These findings suggest that models developed for this work can accurately predict chronic kidney disease, potentially leading to improvements in biomedical treatment.In previous study, the GB algorithm obtained the best accuracy, by far, 0.990. However, in this study, our XGBoost model proved to be very accurate, scoring 0.992 after applying PCA and 0.983 in the original dataset.

Fig. 6: **The specified model's performances**

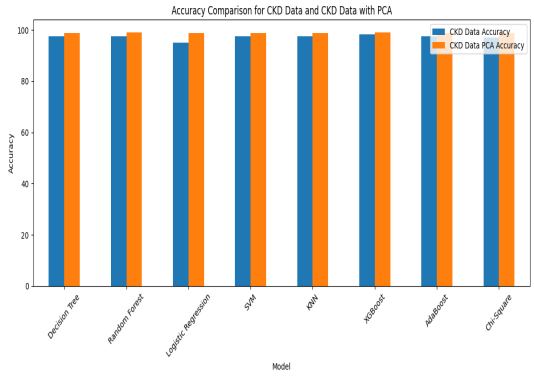
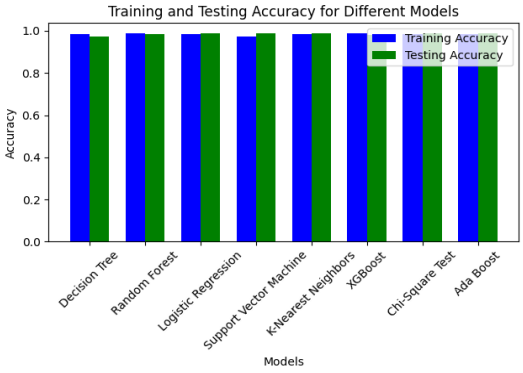


Fig. 7: **Training vs Testing accuracy of different models**



VI. Comparitive analysis

TABLE II: Algorithm performance on the original CKD dataset [7]

Classifiers	Accuracy	Precision	Recall	F1-Score
AdaBoost	97.5	97.66	97.5	97.51
Decision Tree	97.5	97.6	97.5	97.5
XGBoost	98.75	98.8	98.75	98.75
Random Forest	97.5	97.6	97.5	97.5
Logistic Regression	95.0	95.62	95.0	95.06
SVM	97.5	97.66	97.5	97.51
KNN	97.5	97.6	97.5	97.5
Chi Square	98.75	98.79	98.75	98.75

Models for chronic kidney disease (CKD) are particularly useful for outcome prediction by identifying high-risk populations.clinical data can be used for this approach for problems seen in routine clinical practice.

TABLE III: Performance of different techniques using PCA on the CKD dataset

Classifiers	Accuracy	Precision	Recall	F1-Score
AdaBoost	98.75	98.79	98.75	98.75
Decision Tree	98.75	98.77	98.75	98.74
XGBoost	99.12	99.08	99.06	99.08
Random Forest	98.75	98.79	98.75	98.75
Logistic Regression	98.75	98.79	98.75	98.75
SVM	98.75	98.79	98.75	98.75
KNN	98.75	98.79	98.75	98.75
Chi Square	98.75	98.79	98.75	98.75

TABLE IV: Model Training and Testing Parameters for CKD Dataset with PCA

Model	Training Parameters	Testing Parameters
Decision Tree	3200	800
Random Forest	3200	800
Logistic Regression	2880	720
SVM	2880	720
KNN	3200	800
XGBoost	3200	800
AdaBoost	2880	720

However, building this model is difficult due to limited data.In particular, the dataset [10] used for this study contains only 400 samples, which is small and may compromise the validity of the results.table 3 shows the total number number of trainable and testable parameters used in dataset while pca.

VII. Conclusion & Future Enhancement

This research developed a machine learning model towards identifying the beginnings of chronic renal failure. Models were trained and validated focusing on identifying and removing irrelevant features to increase prediction accuracy using previously identified data and analyzing the relationship between type reference parameters showed that hemoglobin , albumin, and specific gravity are important prognostic indicators for chronic renal failure.The initial preprocessing of the CKD data set was done to ensure sure that machine learning pattern recognition was followed. We then used principal component analysis (PCA) to identify significant variables associated with prognosis of chronic kidney disease.

References

[1] Ammirati, A. L. (2020). Chronic kidney disease. Revista da Associação Médica Brasileira, 66(Suppl 1), s03-s09.

[2] Aljaaf, A. J., Al-Jumeily, D., Haglan, H. M., Alloghani, M., Baker, T., Hussain, A. J., & Mustafina, J. (2018, July). Early prediction of chronic kidney disease using machine learning supported by predictive analytics. In 2018 IEEE congress on evolutionary computation (CEC) (pp. 1-9). IEEE.

[3] Chittora, P., Chaurasia, S., Chakrabarti, P., Kumawat, G., Chakrabarti, T., Leonowicz, Z., ... & Bolshev, V. (2021). Prediction of chronic kidney disease-a machine learning perspective. IEEE access, 9, 17312-17334.

[4] Gudeti, B., Mishra, S., Malik, S., Fernandez, T. F., Tyagi, A. K., & Kumari, S. (2020, November). A novel approach to predict chronic kidney disease using machine learning algorithms. In 2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA) (pp. 1630-1635). IEEE.

[5] Hossain, M. M., Swarna, R. A., Mostafiz, R., Shaha, P., Pinky, L. Y., Rahman, M. M., ... & Iqbal, M. S. (2022). Analysis of the performance of feature optimization techniques for the diagnosis of machine learning-based chronic kidney disease. Machine Learning with Applications, 9, 100330.

[6] Islam, M. A., Akter, S., Hossen, M. S., Keya, S. A., Tisha, S. A., & Hossain, S. (2020, December). Risk factor prediction of chronic kidney disease based on machine learning algorithms. In 2020 3rd international conference on intelligent sustainable systems (ICISS) (pp. 952-957). IEEE.

[7] Islam, M. A., Majumder, M. Z. H., & Hussein, M. A. (2023). Chronic kidney disease prediction based on machine learning algorithms. Journal of pathology informatics, 14, 100189.

[8] Elkholy, S. M. M., Rezk, A., & Saleh, A. A. E. F. (2021). Early prediction of chronic kidney disease using deep belief network. IEEE Access, 9, 135542-135549.

[9] Roweis, S. (1998). Em algorithms for pca and spca. Advances in NeuralInformation Processing Systems.

[10] DatasetLink: https://www.kaggle.com/datasets/mansoordaku/ckdisease.

[11] Venkatrao, K., & Kareemulla, S. (2023). HDLNET: a hybrid deep learning network model with intelligent IoT for detection and classification of chronic kidney disease. IEEE Access.

[12] Elkholy, S. M. M., Rezk, A., & Saleh, A. A. E. F. (2021). Early prediction of chronic kidney disease using deep belief network. IEEE Access, 9, 135542-135549.

[13] Rafi, S., & Das, R. (2021, December). RNN encoder and decoder with teacher forcing attention mechanism for abstractive summarization. In 2021 IEEE 18th India council international conference (INDICON) (pp. 1-7). IEEE.

[14] Debnath, D., Das, R., & Rafi, S. (2022, February). Sentiment-based abstractive text summarization using attention oriented lstm model. In Intelligent Data Engineering and Analytics: Proceedings of the 9th International Conference on Frontiers in Intelligent Computing: Theory and Applications (FICTA 2021) (pp. 199-208). Singapore: Springer Nature Singapore.

[15] Rafi, S., & Das, R. (2021, November). A linear sub-structure with co-variance shift for image captioning. In 2021 8th International Conference on Soft Computing & Machine Intelligence (ISCMI) (pp. 242-246). IEEE.

[16] Rafi, S., & Das, R. (2023). Topic-guided abstractive multimodal summarization with multimodal output. Neural Computing and Applications, 1-16.

[17] Rafi, S., & Das, R. (2023, November). Abstractive Text Summarization Using Multimodal Information. In 2023 10th International Conference on Soft Computing & Machine Intelligence (ISCMI) (pp. 141-145). IEEE.

[18] Rafi, S., & Das, R. (2024). SCT: Summary Caption Technique for Retrieving Relevant Images in Alignment with Multimodal Abstractive Summary. ACM Transactions on Asian and Low-Resource Language Information Processing, 23(3), 1-22.

ckd_iatmsi_cam.pdf

ORIGINALITY REPORT

9%	7%	7%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to MIT Academy of Engineering Student Paper	1%
2	www.mdpi.com Internet Source	1%
3	Submitted to Dhirubhai Ambani Institute of Information and Communication Student Paper	1%
4	ijritcc.org Internet Source	1%
5	"Applied Intelligence and Informatics", Springer Science and Business Media LLC, 2024 Publication	1%
6	Submitted to Jain University Student Paper	1%
7	Apoorva S. Mehta, Jinit S. Raval, Rupal R. Chaudhari. "CureIt – A Multidisease Predictive System using Machine Learning", ITM Web of Conferences, 2024 Publication	<1%

8	www.em-consulte.com Internet Source	<1%
9	Sireesha Moturi, S.N. Tirumala Rao, Srikanth Vemuru. "Grey Wolf assisted Dragonfly-based Weighted Rule Generation and Fuzzy Learning for Predicting Heart Disease and Breast Cancer", Computerized Medical Imaging and Graphics, 2021 Publication	<1%
10	V. Sharmila, S. Kannadhasan, A. Rajiv Kannan, P. Sivakumar, V. Vennila. "Challenges in Information, Communication and Computing Technology", CRC Press, 2024 Publication	<1%
11	Krishna Sowjanya K, Bindu Madavi K P, Neha Patwari, Shobharani D A. "DeepKidney: Multiclass Classification of Kidney Stones, Cysts, Tumors, and Normal Cases Using Convolutional Neural Networks", 2023 4th International Conference on Communication, Computing and Industry 6.0 (C216), 2023 Publication	<1%
12	Muhammad Minoar Hossain, Reshma Ahmed Swarna, Rafid Mostafiz, Pabon Shaha et al. "Analysis of the performance of feature optimization techniques for the diagnosis of machine learning-based chronic kidney	<1%

	disease", Machine Learning with Applications, 2022 Publication	
13	e-archivo.uc3m.es Internet Source	<1 %
14	ebin.pub Internet Source	<1 %
15	iieta.org Internet Source	<1 %
16	pdfcoffee.com Internet Source	<1 %
17	repositorio.iscte-iul.pt Internet Source	<1 %
18	tnsroindia.org.in Internet Source	<1 %
19	www.wellbeingcenter.co Internet Source	<1 %
20	Swamy, Narayana. "A Study of Cardiac Troponin - T in Patients with End Stage Renal Disease", Rajiv Gandhi University of Health Sciences (India), 2023 Publication	<1 %
21	"Deep Sciences for Computing and Communications", Springer Science and Business Media LLC, 2024 Publication	<1 %

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		

CERTIFICATE – 1

3rd IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IEEE IATMSI-2025)

CERTIFICATE OF PRESENTATION

This is to certify that

Nuti Revanth {Narasaraopeta Engineering College}

Presented a Paper Titled

Chronic Kidney Disease Prediction Using Machine Learning and Deep Learning Models

in IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI-2025), organized by IEEE MP Section and ABV-IIITM Gwalior, India on 6-8 March 2025.

Dr. Arun Kumar
Organizing Chair, IATMSI-2025

Prof. G. S. Tomar
General Chair, IATMSI-2025

Dr. Somesh Kumar
Conference Chair, IATMSI-2025

CERTIFICATE – 2

3rd IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IEEE IATMSI-2025)

CERTIFICATE OF AUTHORSHIP & ACCEPTANCE

This is to certify that Paper Titled

Chronic Kidney Disease Prediction Using Machine Learning and Deep Learning Models

Submitted by Authors

Shaik Rafi M.Tech.,(PhD){Narasaraopeta Engineering College} , Nuti Revanth, Kota Veera Raghava Reddy , Kurra Mahesh Babu , Yallanti Likhith Prasanna Kumar{Narasaraopeta Engineering College}

is accepted and presented in IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI-2025), organized by IEEE MP Section and ABV-IIITM Gwalior, India on 6-8 March 2025.

Dr. Arun Kumar
Organizing Chair, IATMSI-2025

Prof. G. S. Tomar
General Chair, IATMSI-2025

Dr. Somesh Kumar
Conference Chair, IATMSI-2025