PREDICTION OF CHRONIC KIDNEY DISEASE USING MACHINE LEARNING

Dissertation submitted

in partial fulfilment of requirements for the award of the degree of

Master of Computer Applications (M.C.A)

Submitted By

ARAVA ANANTHA RAO

(Regd. No: 21131F0001)

Under the esteemed guidance of

Mr. P.V.V.R CHANDRA SEKHAR

Assistant Professor

Department of Computer Applications

Department of Computer Applications

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Affiliated to JNTU-K Kakinada)

VISAKHAPATNAM-530048

2021-2023

PREDICTION OF CHRONIC KIDNEY DISEASE USING MACHINE LEARNING

Dissertation submitted

in partial fulfilment of requirements for the award of the degree of

Master of Computer Applications (M.C.A)

Submitted By

ARAVA ANANTHA RAO

(Regd. No: 21131F0001)

Under the esteemed guidance of

Mr. P.V.V.R CHANDRA SEKHAR

Assistant Professor

Department of Computer Applications

Department of Computer Applications

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Affiliated to JNTU-K Kakinada)

VISAKHAPATNAM-530048

2021-2023

CERTIFICATE

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING
(AUTONOMOUS)

This is to certify that, the dissertation titled **"PREDICTION OF CHRONIC KIDNEY DISEASE USING MACHINE LEARNING"** is submitted by **Mr. ARAVA ANANTHA RAO** in partial fulfillment of the requirement for award of the Degree of **M.C.A** in **GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING (AUTONOMOUS)** affiliated to Jawaharlal Nehru Technological University, Kakinada is a bonafide record of project carried out by him under my guidance and supervision.

The contents of the project report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree.

Internal Guide
Mr. P. V. V. R Chandra Sekhar
Assistant Professor
Dept. of Computer Applications

Head of the Department
Dr. Y. Anuradha
Associate Professor
Dept. of Computer Applications

External Examiner

CERTIFICATE OF PLAGIARISM CHECK

This is to certify the Master of Computer Applications (MCA) dissertation submitted by ARAVA ANANTHA RAO (21131F0001) of Department of Computer Applications under the supervision of Mr. P.V.V.R CHANDRA SEKHAR, Assistant professor of Department of Computer Applications, has undergone plagiarism check and found to have similarity index less than 40%. The details of the plagiarism check are as under:

File Name	: **PREDICTION_OF_CHRONIC_KIDNEY_ DISEASE_USING_MACHINE_LEARNING.p df (1.76M)**
Dissertation Title	: **PREDICTION OF CHRONIC KIDNEY DISEASE USING MACHINE LEARNING.**
Date and Time of Submission	**: 10-Jul-2023 12:30PM (UTC+0530)**
Submission ID	: **2128993803**
Similarity Index	**: 8%**

Dean- Academic Programs (PG)

DECLARATION

I hereby declare the dissertation titled **"PREDICTION OF CHRONIC KIDNEY DISEASE USING MACHINE LEARNING"** is submitted to the Department of Computer Applications, **Gayatri Vidya Parishad College of Engineering (Autonomous)** affiliated to Jawaharlal Nehru Technological University, Kakinada in partial fulfillment of the requirements for the award of the degree of **Master of Computer Applications (MCA).** This work is done by me and authentic to the best of my knowledge under the direction and valuable guidance of **Mr. P.V.V.R Chandra Sekhar,** Assistant Professor in the department of Computer Applications.

ARAVA ANANTHA RAO

(21131F0001)

ACKNOWLEDGEMENT

I take the opportunity to thank everyone who has contributed to making the project possible. I am thankful to Gayatri Vidya Parishad College of Engineering (Autonomous) for giving opportunity me to work on a project as part of the curriculum.

I offer my copious thanks to **Dr. A. B. Koteswara Rao**, the principal of **Gayatri Vidya Parishad College of Engineering (Autonomous)** for providing the best faculty and lab facilities throughout the completion of Master of Computer Applications course.

I offer my copious thanks to Prof. **Dr. K. Narasimha Rao** M.Tech, Ph.D. Professor & Dean, Academics (PG) for his valuable suggestions and constant motivation that greatly helped me to complete project successfully.

I offer my copious thanks to **Dr. Y. Anuradha**, Associate Professor, Head of the Department of Computer Applications for his valuable suggestions and motivation that greatly helped me to complete the project successfully.

My sincere thanks to **Mr. P. V. V. R Chandra Sekhar**, Assistant Professor,Department of Computer Applications, my project guide, for his constant support, encouragement, and guidance. I am very much grateful for his valuable suggestions.

ARAVA ANANTHA RAO
(Regd. No. 21131F0001)

ABSTRACT

Through a detailed study of several chronic medical conditions, technological innovation has had an immense impact on health. The detection of such disorders at early stages is particularly crucial. It is challenging for clinicians to manually diagnose the disorders precisely. Mainly chronic kidney disease is caused by diabetes and high blood pressure. People with chronic kidney disease are more likely to die early.

Therefore, the system aims to develop an intelligent model that classifies patients into three groups: 'Severe CKD', 'Mild-Mod CKD', and 'End Stage of Renal Disease (ESRD)'. This model facilitates doctor's efficient management of multiple patients and faster diagnoses.

In this study, the prediction of CKD, Mild-Mod CKD and ESRD conditions is performed using 'K-Nearest Neighbors (KNN)' and 'Decision Tree' algorithms. Through experimentation, the decision tree algorithm achieved highest accuracy.

INDEX

CONTENTS

List of Figures

List of Tables

List of Symbols / Acronyms / Abbreviations

Symbol/ Acronym/ Abbreviation	DESCRIPTION
CKD	CHRONIC KIDNEY DISEASE
ESRD	END STAGE OF RENAL DISEASE
GUI	GRAPHICAL USER INTERFACE
UML	UNIFIED MODELING LANGUAGE

CHAPTER 1
INTRODUCTION

INTRODUCTION

Humans having chronic kidney disease have been reported to have damaged kidneys that are unable to function properly. Globally, chronic kidney disease is a severe public health issue that affects millions of people. Early diagnosis and treatment of (CKD) may greatly enhance patient outcomes, avoid complications, and save medical expenses.

Kidney disease, including CKD, is found to impact more than 800 million individuals worldwide. For someone to be identified as having CKD, two sets of samples must be collected at least 90 days apart. CKD can worsen over time, and both kidneys may eventually quit working. CKD is frequently associated with other diseases.

The prediction of chronic kidney disease is of crucial relevance in healthcare, since early detection and intervention can dramatically improve patient outcomes and minimize the cost on healthcare systems. CKD is a frequent and dangerous disorder that affects millions of individuals worldwide. It is distinguished by the kidneys' progressive deterioration, which lowers their ability to filter waste materials and to regulate the equilibrium of fluids in the body.

The system intends to construct a predictive model for CKD, leveraging powerful machine learning algorithms and a rich dataset of patient information. By examining numerous clinical, demographic, and laboratory data, we strive to discover important predictors that can accurately indicate the risk of CKD onset or progression in patients.

The predictive model will utilize the power of data mining to assess large volumes of data, including characteristics such as age, gender, medical history, lifestyle decisions, and laboratory test results. By training the model on a varied range of patient characteristics and outcomes, we intend to produce a trustworthy tool that can assist healthcare practitioners in detecting high-risk individuals and initiating timely interventions.

1.1 Overview:

Chronic Kidney Disease (CKD) is a pervasive and life-altering medical condition

that affects millions of people worldwide. It is characterized by the gradual deterioration of kidney function over time, leading to a reduced ability to filter waste products from the blood and maintain fluid balance in the body. CKD is a severe public health issue that poses significant challenges to both patients and healthcare systems. Globally, it ranks as one of the leading causes of morbidity and mortality.

The early detection and management of CKD are paramount to improving patient outcomes and reducing the burden on healthcare resources. Unfortunately, CKD often remains undiagnosed until it reaches an advanced stage, where treatment options are limited and costly. As a result, there is a critical need for innovative approaches that can facilitate early identification and risk stratification of individuals at higher risk of developing CKD or experiencing disease progression.

In recent years, the field of healthcare has witnessed tremendous advancements in artificial intelligence and machine learning. These technologies have demonstrated exceptional potential in revolutionizing various medical domains, including disease prediction and personalized treatment. Machine learning algorithms, in particular, have the ability to analyze vast and complex datasets, extracting patterns and insights that might be difficult for human experts to discern.

In light of this, our system aims to harness the power of machine learning to develop a predictive model for CKD. By leveraging a diverse and comprehensive dataset comprising patient demographics, medical history, laboratory test results, lifestyle factors, and genetic information, our model endeavors to identify significant risk factors and indicators associated with CKD.

The predictive model will be designed to assess large volumes of data efficiently and provide accurate risk assessments for individuals. This will enable healthcare practitioners to intervene proactively and tailor treatment plans based on each patient's unique profile, potentially slowing down the progression of CKD and reducing the incidence of adverse outcomes.

In the process of model development, we will explore a range of machine learning algorithms, including decision trees and KNN. By subjecting the model to rigorous experimentation and validation, we seek to fine-tune its performance to achieve the highest possible accuracy, sensitivity, and specificity in predicting CKD occurrence and

progression.

Moreover, while technological advancements present immense opportunities, they also come with ethical considerations that we must address responsibly. Patient privacy and data security will be strictly adhered to, ensuring that all data usage complies with relevant regulations and guidelines. Additionally, transparent communication with patients and healthcare providers about the purpose and implications of the predictive model will be an essential aspect of successful implementation.

By combining cutting-edge machine learning techniques with a wealth of patient information, our project aspires to revolutionize CKD diagnosis and management. The development of an efficient and reliable predictive model for CKD holds the potential to save lives, improve patient outcomes, and alleviate the global burden of this debilitating disease. Through collaborative efforts with the medical community, we aim to pave the way for a more proactive and personalized approach to combat CKD and enhance the overall well-being of affected individuals worldwide. Ultimately, this study represents a significant step forward in the fight against CKD, offering hope for a brighter and healthier future.

The development of a predictive model for Chronic Kidney Disease holds immense promise in transforming the landscape of healthcare. By providing early detection and risk assessment, this model can offer a paradigm shift in CKD management, leading to improved patient outcomes, reduced healthcare costs, and enhanced quality of life for those affected.

1.2 Objective:

The impact of CKD goes beyond the physiological aspects, as it significantly affects the emotional well-being of patients and their families. The uncertainty surrounding disease progression and the need for prolonged medical care can create significant stress and anxiety. Therefore, an accurate and reliable predictive model can alleviate these concerns by empowering patients and healthcare providers with timely information for informed decision-making and proactive interventions.

Furthermore, the insights derived from the predictive model can be leveraged for public health initiatives and resource allocation. Identifying high-risk populations and

geographical regions with a higher prevalence of CKD can help policymakers implement targeted preventive measures and awareness campaigns, thereby potentially reducing the overall burden of the disease on healthcare systems.

Collaboration among interdisciplinary teams, including nephrologists, data scientists, biostatisticians, and healthcare policymakers, will be crucial for the successful implementation and adoption of the predictive model. This partnership will ensure that the model's development aligns with the practical needs of the healthcare sector and enhances its usability and impact.

Additionally, as the model is developed and deployed, it will continuously learn and improve with additional data, thereby refining its accuracy and adaptability. Regular updates and maintenance will be essential to ensure that the model remains up-to-date with the latest medical advancements and epidemiological trends.

While the potential benefits of the predictive model for CKD are compelling, it is essential to acknowledge the challenges that lie ahead. Data quality, data integration, and interpretability of the model's results will be critical aspects to address. Moreover, the model's implementation should prioritize equity and fairness, ensuring that it does not exacerbate existing healthcare disparities or biases.

Finally, the prediction of Chronic Kidney Disease using machine learning is a promising endeavor that has the potential to revolutionize healthcare and enhance the lives of millions affected by this debilitating condition. By harnessing the power of data and cutting-edge technology, we aim to develop a robust and accurate predictive model that can empower healthcare practitioners, inform patients, and drive informed decisions for a healthier and more resilient global population. Through a collective effort of researchers, clinicians, policymakers, and patients, we strive to pave the way for a future where CKD can be detected early, managed effectively, and its impact minimized on a global scale.

CHAPTER 2

LITERATURE SURVEY

LITERATURE SURVEY

[1] M. P. N. Wickramasingh suggests a strategy in 2020 to regulate the disease by following a healthy diet.

A few of the methods employed in this research to develop classifiers are Multiclass Decision Jungle, Multiclass Neural Network, and Multiclass Logistic Regression. The anticipated permissible potassium zone depends on the patient's blood potassium levels. The categorization algorithms based on the expected potassium zone give a diet location.

[2] Using data analytics, W. Gunarathne completed a performance evaluation of machine learning classification techniques for chronic kidney disease forecasting and classification in 2017.

Predicting CKD and Non-CKD in a patient is their area of research. Multiclass - Decision Forests, Jungle, Logistic Regression, and NN are some of the classification techniques that have been employed. Utilising Microsoft Azure Machine Learning Studio, the results were attained.

[3] A system developed by J. Snegha in 2020 makes use of the Random Forest algorithm and Back Propagation Neural Network, among other data mining approaches.

As they evaluate the two algorithms, they find that the Back Propagation approach produces the best results since it makes use of the Feed Forward Neural Network, a supervised learning network.

[4] J. Van Eiyck, J. Remon, F. Giza, G. Mefroidt, M. Bruynighe, G. Van den Berghe, K. U. Leuven, and others used data mining methods in 2017.

Using machine learning (classification and regression tasks) and the Gaussian process for predicting renal damage after elective heart surgery.

[5] Aljaaf, A.J. 2018 Early Prediction of Chronic Kidney Disease Using Machine Learning Supported by Predictive Analytics. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC). Wellington. New Zealand.

A combination of estimated glomerular filtration rate (GFR), age, diet, existing medical conditions, and albuminuria can be used to assess the severity of kidney disease, but requires more accurate information about the risk to the kidney is required to make

clinical decisions about diagnosis, treatment, and referral.

[6] A. Nishanth, T. Thiruvaran, Identifying important attributes for early detection of chronic kidney disease. IEEE Rev. Biomed. Eng. 11, 208–216 (2018).

The purpose of this model is to develop and validate predictive models for chronic kidney disease. The main goal will be to evaluate kidney failure, which means the need for kidney dialysis or kidney transplant first.

[7] F. Aqlan, R. Markle, A. Shamsan, "Data mining for chronic kidney disease prediction." in IIE Annual Conference. Proceedings, Institute of Industrial and Systems Engineers, (IISE 2017), pp. 1789–1794.

The purpose of the proposed model is to predict whether the patient will suffer or develop chronic kidney disease in the future if he continues their lifestyle. This information can be used to determine whether the kidney disease is using eGFR (glomerular filtration rate), which helps the doctor plan the appropriate treatment. Estimated glomerular filtration rate (eGFR) defines the degree of kidney disease and measures kidney function.

[8] N. Borisagar, D. Barad, P. Raval, Chronic kidney disease prediction using back propagation neural network algorithm. Proce. Int. Confe. Commun. Netw. 19–20, 295–303 (2017).

The main function of the kidney is to filter the blood in the body. Kidney disease is a silent killer because it can cause kidney failure without causing any symptoms or concern. Chronic kidney disease is defined as a decline in kidney function over a period of months or years. Not getting the right treatment for chronic kidney disease can have serious consequences, affecting people who can't afford it. Glomerular filtration rate (GFR) is the most accurate test to determine your kidney function and the degree of chronic kidney disease. Blood creatinine level, age, gender, and other characteristics can be used to calculate it. In most cases it is better to get sick early. Therefore, it is possible to prevent serious illness.

[9] R.S. Walse, G.D. Kurundkar, S.D. Khamitkar, A.A. Muley, P.U. Bhalchandra, S.N. Lokhande, Effective use of naïve bayes, decision tree, and random forest techniques for analysis of chronic kidney disease, in International Conference on

Information and Communication Technology for Intelligent Systems. ed. by T. Senjyu, P.N. Mahalle, T. Perumal, A. Joshi (Springer, Singpore, 2020).

Wasle et al. devised a statistical analysis technique. For the examination of the Chronic Kidney Disease dataset, the authors employed a variety of machine learning approaches. They used Nave Bayes, Decision Trees, and Random Forest to improve prediction, and they discovered that Random Forest computed greater classification accuracy than the other algorithms.

[10] A. Nithya, A. Appathurai, N. Venkatadri, D.R. Ramji, C.A. Palagan, Kidney disease detection and segmentation using artificial neural network and multi-kernel k-means clustering for ultrasound images. Measurement (2020).

On the Kidney disease dataset, Nithya et al. developed a method for categorization and cluster-based analysis. On diverse sets of photos, the authors utilized the K-Means clustering technique to collect the closest familiar images. They calculated 99.61 percent classification accuracy using Artificial Neural Networks for Kidney Disease Image Prediction.

CHAPTER 3
SYSTEM ANALYSIS

3.1 EXISTING METHOD:

Currently, the diagnosis and prediction of chronic kidney disease (CKD) rely on traditional diagnostic methods such as blood tests, urine tests, and imaging tests. These tests are often costly and time-consuming, making them impractical for widespread use. Additionally, these tests may not always detect CKD in its early stages.

Lack of nephrologists or general doctors makes it difficult for many local hospitals and clinics to diagnose symptoms. There are various machine learning-based models for predicting CKD now available, however they are often constrained by the quantity and quality of the training data.. Moreover, these models may use complex algorithms that are difficult to interpret and may not be suitable for clinical environments.

Disadvantages of Existing Method:

- Time efficiency is high.
- Don’t know which algorithm can offer better results.

3.2 PROPOSED METHOD:

In the proposed system, we identify the best combinations that can execute, compare the performance and accuracy of the algorithm, and provide a better accuracy and detection rate. Finally, we demonstrate that the 'Decision Tree Algorithm' successfully separates the clinical data of CKD from non-CKD with an overall accuracy of 86%.

Advantages of Proposed Method:

- Provides fast operation.
- Low cost expense than old system.
- Provides user friendly interface.
- Reducing Time efficiency.

CHAPTER 4

SYSTEM REQUIREMENTS AND SPECIFICATION

4.1 FUNCTIONAL REQUIREMENTS:

- **Data Collection:**

  Collect a large dataset of clinical data from kidney disease patients

- **Data Pre-Processing:**

  Preprocess the collected data by removing missing values and outliers

- **Model Training and Evolution:**

  The system should allow the training of the models using a labeled dataset, providing options to adjust the parameters

- **Usability:**

  The system should have a user-friendly interface, allowing easy navigation

- **Security and Privacy:**

  The system should integrate adequate security mechanisms to secure patient's data

4.2 NON FUNCTIONAL REQUIREMENTS:

- **Performance:**

  The system should be capable of handling big datasets effectively, ensuring minimum processing time for training and prediction.

- **Reliability:**

  The system should provide a reliable environment for the user to ensure that predictions are consistent

- **Availability:**

  The system will be available by 24/7.

4.3 SYSTEM REQUIREMENTS

4.3.1 MINIMUM HARDWARE REQUIREMENTS:

Processor	:	Intel I3
RAM	:	8GB
Hard Disk	:	128 GB

4.3.2 MINIMUM SOFTWARE REQUIREMENTS:

Front End	:	Html, CSS, Flask
Programming Language	:	Python 3.7
Platform	:	Visual Studio Code
OS	:	Windows 7 (64bit)

4.4 DATASET

AGE	WEIGHT	SG	Alb	eGFR	Wbc	CLASS
57	89	1.02	4	80	7800	0
78	87	1.02	3	78	6300	0
76	90	1.01	4	90	7500	0
70	92	1.005	2	75	6700	0
65	94	1.01	4	89	4300	0
80	74	1.015	3	87	10000	0
72	86	1.01	2	75	4500	0
75	84	1.005	1	60	12000	2
89	82	1.015	1	35	11000	2
72	73	1.01	2	31	11500	2
64	74	1.015	4	95	7900	0
69	81	1.02	1	37	12000	1
67	63	1.01	2	27	12500	1
75	76	1.01	1	29	10000	1
63	89	1.01	2	68	7900	0
69	64	1.015	2	67	5700	0
59	89	1.005	3	98	4500	0
78	87	1.01	4	78	8500	0
87	88	1.015	2	50	10000	2
73	74	1.02	1	43	11000	2
71	75	1.01	2	100	5500	0
70	75	1.01	3	110	6500	0
79	72	1.015	1	26	12000	1
73	78	1.015	2	97	10000	0
68	79	1.015	3	115	7800	0
67	85	1.015	1	64	9800	2
63	82	1.015	4	97	4200	0
65	80	1.015	2	100	8300	0
68	84	1.001	3	110	3800	0
63	82	1.015	4	97	4200	0

Figure 4.4 Dataset

CHAPTER 5
SYSTEM DESIGN

5.1 ARCHITECTURE DIAGRAM:

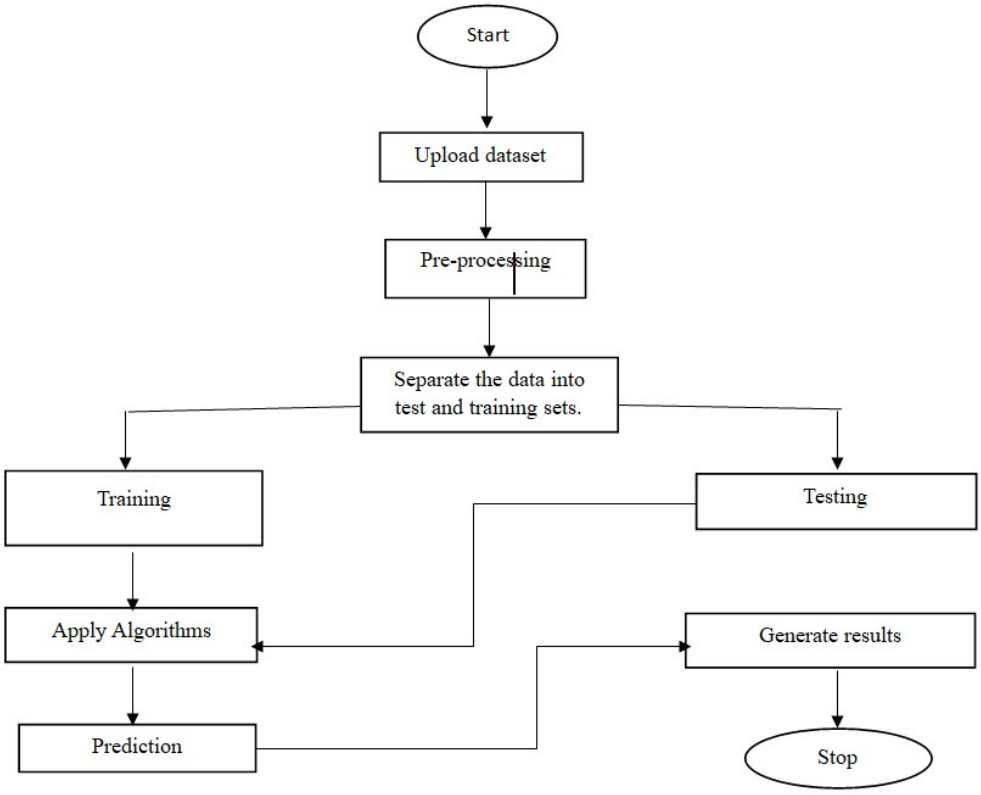


Figure 5.1 Architecture Diagram

5.2 UML DIAGRAMS:

UML (Unified Modeling Language) diagrams are visual depictions used in software engineering and system development to show the structure, behavior, and relationships of a system.

UML diagrams play a significant role in the software development lifecycle, assisting in analysis, design, implementation, and documentation, thereby boosting the clarity and comprehension of complex systems.

Features:

- It is a language used for broad modelling.
- It differs from other programming languages like Python, C++, and others.
- It is related to analysis and design that is object-oriented.
- It is utilised to visualise the system's workflow.

5.2.1 USE CASE DIAGRAM:

A use case diagram is used to graphically displays the functional needs of a system or software application from the viewpoint of its users or actors. It gives a high-level picture of the system's activity and interactions with its external components.

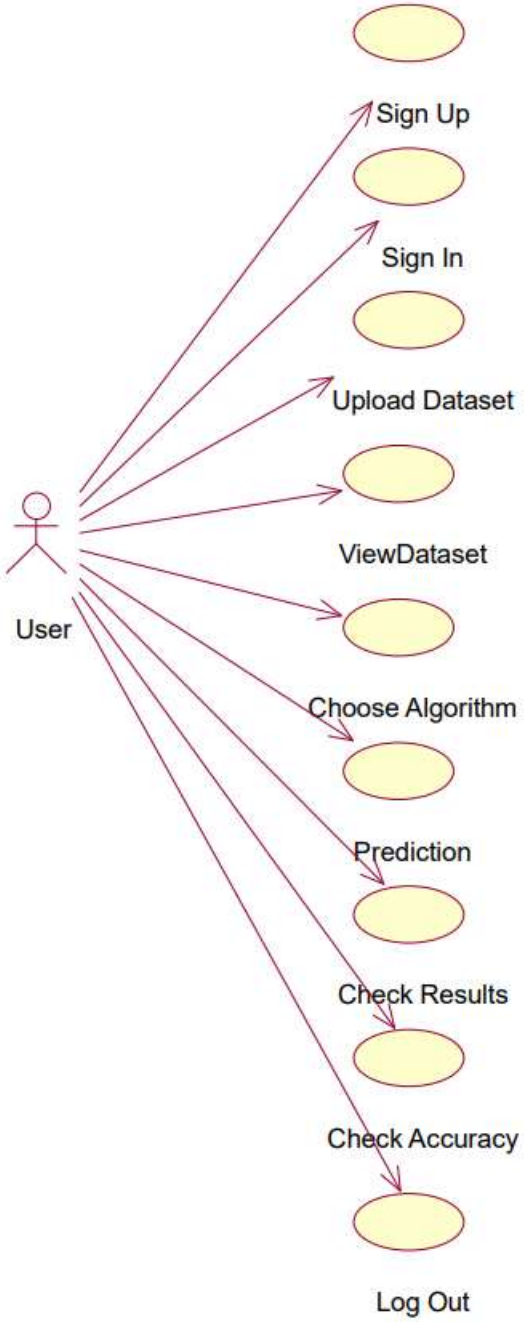


Figure 5.2.1 Use Case Diagram

5.2.2 SEQUENCE DIAGRAM:

Sequence diagrams consist of lifelines, messages, and activation boxes. Lifelines indicate the items or components participating in the interaction, each portrayed as a vertical line. Messages are represented by arrows that link the lifelines, signifying the communication or connection between items.

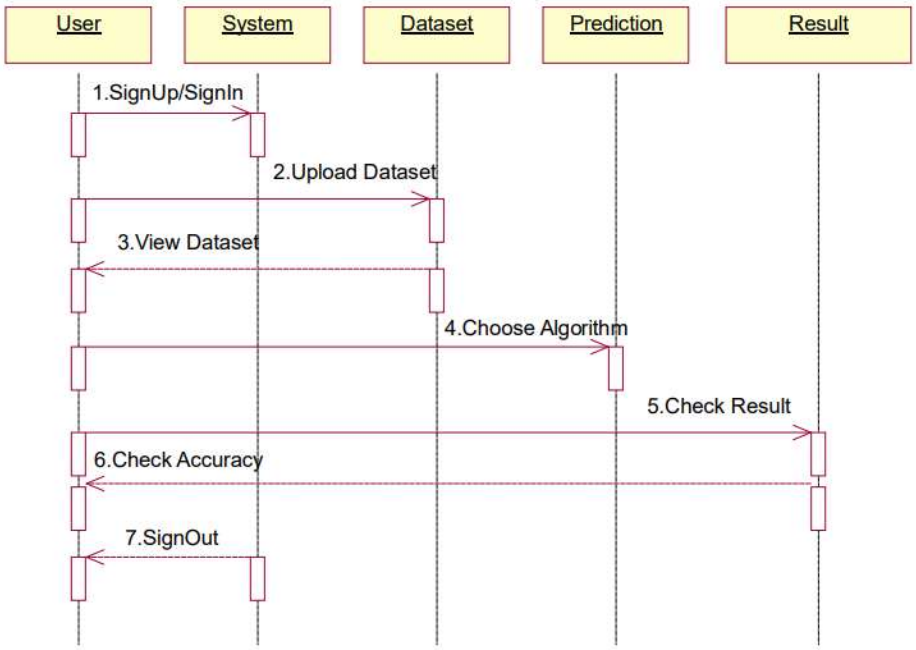


Figure 5.2.2 Sequence Diagram

5.2.3 ACTIVITY DIAGRAM:

An activity diagram is used to visualize the flow of activities or processes inside a system or business process. It emphasises on the workflow or behavioral elements of a system, capturing the sequence of events, choices, and concurrency involved. Activity diagrams consist of nodes, edges, actions, and control components.

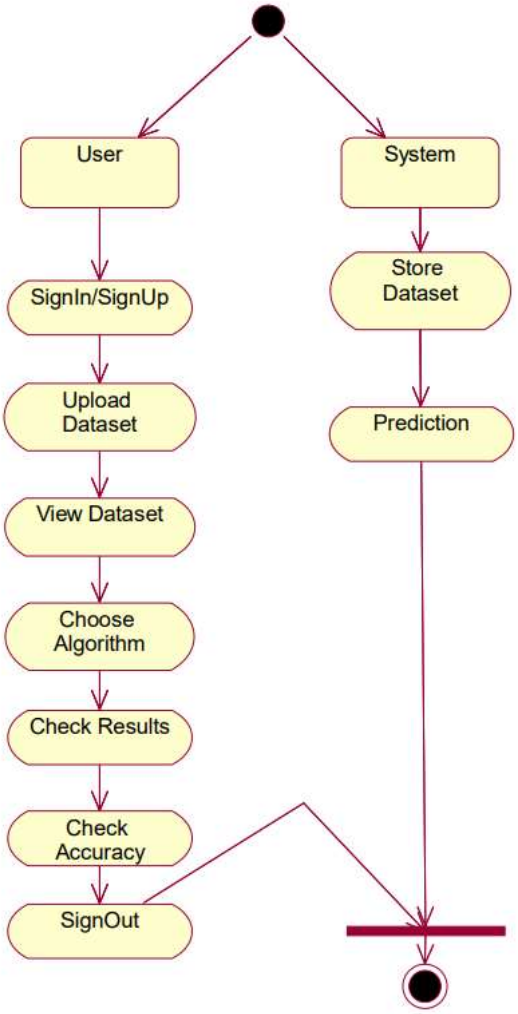


Figure 5.2.3 Activity Diagram

CHAPTER 6

IMPLEMENTATION

6.1 IMPLEMENTATION

6.1.1 SOFTWARE ENVIRONMENT :

Python:

- Python is a high-level programming language noted for its simplicity and clarity.
- Python has a simple and legible syntax, which makes it easy to learn and comprehend. Its use of indentation as a technique to designate code chunks increases code uniformity and readability.
- Python has a huge and active community of developers that contribute to its open-source environment. This community-driven approach assures continual progress, regular updates, and substantial documentation and support resources.
- Google and Yahoo! both utilise Python a lot to make their Web crawlers and search engines work.

Flask:

- Flask offers a rapid debugger and an integrated development server.
- Its basis is Unicode.
- The documentation for Python Flask is vast.
- Additionally, it includes the integrated support essential for unit testing.
- Flask can also be installed in a virtual environment on Windows, Mac, and Linux.

Installation:

- pip install flask

6.2 LIBRARIES USED

Pandas:

A well-liked data analysis and manipulation toolfor the python programming language is the panda's library. Pandas has inbuilt tools for dealing with inconsistent data, duplicate values, and missing data. It also works very well with tabular and time-series data.

Matplotlib:

Matplotlib is a library in python which is used for visualization. It allows us to plot graphs and pie charts and that can be used for comparison of several metrics in the format of a diagram.

NumPy:

NumPy is a powerful Python library widely used for numerical computing and data analysis. Short for "Numerical Python," NumPy provides an extensive collection of multi-dimensional array objects, along with functions to manipulate these arrays efficiently.

Seaborn:

Seaborn is a popular Python data visualization library built on top of matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics. Seaborn is designed to work seamlessly with data frames from the Pandas library, making it an excellent choice for data visualization in data science and statistical analysis projects.

Scikit-learn (Sklearn):

Scikit-learn, often abbreviated as sklearn, is one of the most widely used and well-known machine learning libraries in Python. It is built on top of NumPy, SciPy, and matplotlib and provides a comprehensive set of tools for machine learning, data mining, and data analysis tasks. Scikit-learn is open-source and actively developed by a vibrant community of contributors.

6.3 K-NEAREST NEIGHBORS (KNN) ALGORITHM:

- K-Nearest Neighbors (KNN) is a machine learning technique used for classification and regression tasks.
- It works by identifying the k nearest points in the training set that are closest to a given input sample.
- The method predicts the class or value of the input based on the majority vote (for classification) or the average (for regression) of the neighbors.
- KNN is a non-parametric method, which means it does not make any assumptions about the underlying data distribution and may function effectively with various datasets.

Steps:

- Choose the k-number of neighbors.
- Find the neighbors euclidian distances using k.
- Determine the k nearest neighbors in accordance with the calculation.
- Count the number of data points in each category among the k neighbors.
- The category with the greatest number of neighbors should receive the additional data points.
- Stop the operation.

6.4 DECISION TREE ALGORITHM:

The supervised learning approach known as a decision tree may be applied to classify and regression issues, although it is most typically utilised to deal with classification challenges. It is a tree-structured classifier, with internal nodes standing in for dataset characteristics, branches for decision-making procedures, and leaf nodes for conclusions.

Steps:

- Identify the tree's root.
- Split the data based on select feature.
- Repeat the splitting process until a stopping condition is met.
- Set labels to every single leaf node.
- Prune the tree.
- Make predictions by traversing the tree from root to leaf node.
- Evaluate the model.

6.5 SOURCE CODE

- **app.py:**

```
from flask import Flask, render_template, request, redirect, url_for
import pandas as pd
import numpy as np
import random
import string
import secrets
import csv
import math
from knn import k,predict,train_features,train_target
from DT import DecisionTree
from knnacc import acc
from DTACC import d_accuracy
app = Flask(__name__)
users=[]
@app.route('/')
def home():
    return render_template('index.html')
@app.route('/second')
def second():
return render_template('sign.html')
```

```
@app.route('/third')
def third():
    return render_template('aboutus.html')
@app.route('/fourth')
def fourth():

    df = pd.read_csv(file)
 # render the view.html template with the dataframe
 return render_template('view.html', table=df.to_html(index=False))
    return redirect(request.url)
@app.route('/accuracy')
def make_accurate():
   knn_acc=acc
   dec_acc=d_accuracy
   return
render_template('accuracy.html',knn_prediction=knn_acc,dt_prediction=dec_acc)
@app.route('/prediction')
def prediction():
    return render_template('prediction.html')
@app.route('/result', methods=['POST'])
def make_prediction():
    patient_details=[]
    # Get user input from the HTML form
```

```
    name = request.form['name']
    age = int(request.form['age'])
    weight = float(request.form['weight'])
    sg = float(request.form['sg'])
    alb = float(request.form['alb'])
    egfr = float(request.form['egfr'])
    wbc = float(request.form['wbc'])
    algorithm = request.form['algorithm']
    # Create a list from the user input
    patient_details.append([age, weight, sg, alb, egfr, wbc])

    if algorithm == 'knn':
        prediction = predict(train_features, train_target, patient_details, k)[0]
        if prediction == 0:
            prediction_text = "Mild-Mod CKD"
        elif prediction == 1:
            prediction_text = "ESRD"
        elif prediction == 2:
  prediction_text = "Severe CKD"
  return render_template('result.html', name=name, age=age, weight=weight,
sg=sg, alb=alb, egfr=egfr, wbc=wbc, prediction=prediction_text)
    elif algorithm == 'decisiontree':
        patient_data = np.array([[age, weight, sg, alb, egfr, wbc]])
```

```
        df = pd.read_csv("kidney.csv")
        X = df.drop(columns=["CLASS"]).values
        y = df["CLASS"].values
        train_size = int(0.7 * len(df))
        X_train, X_test = X[:train_size], X[train_size:]
        y_train, y_test = y[:train_size], y[train_size:]
        dt = DecisionTree(max_depth=5)
        dt.fit(X_train, y_train)
        prediction2 = dt.predict1(patient_data)[0]
        if prediction2 == 0:
            prediction_text2 = "Mild-Mod CKD"
        elif prediction2 == 1:
            prediction_text2 = "ESRD"
        elif prediction2 == 2:
            prediction_text2 = "Severe CKD"
        return render_template('resultdec.html', name=name, age=age,
weight=weight, sg=sg, alb=alb, egfr=egfr, wbc=wbc,
prediction2=prediction_text2)
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        # Get the form data
        name = request.form['name']
```

```
        email = request.form['email']
        password = request.form['password']
        confirm_password = request.form.get('confirm-password', '') # Get the confirm_password field if it exists, otherwise use an empty string
        # Validate the form data
if not name or not email or not password or not confirm_password:
            return "Please fill out all fields."
        elif password != confirm_password:
            return "Passwords do not match."
        elif email in [user['email'] for user in users]:
            return "An account with that email already exists."
        else:
            # Add the user data to the list
            users.append({
                'name': name,
                'email': email,
                'password': password
            })
            # Redirect to the main page
            return redirect('/main')
    return render_template('signup.html')
@app.route('/signin', methods=['GET', 'POST'])
def signin():
```

```
    if request.method == 'POST':
        # Get the form data
        email = request.form['email']
        password = request.form['password']
        # Validate the form data
        if not email or not password:
            return "Please enter your email and password."
        elif email not in [user['email'] for user in users]:
            return "No account with that email exists."
        else:
            # Check if the password is correct
            user = next(user for user in users if user['email'] == email)
if password != user['password']:
                return “Incorrect password.”
else:
                # Redirect to the main page
                return redirect('/main')
    return render_template('sign.html')
@app.route('/main')
def main():
    return render_template('main.html')
@app.route('/forgot', methods=['GET', 'POST'])
def forgot_password():
```

```
if request.method == 'POST':
    email = request.form['email']

    # Validate the form data
    if not email:
        return "Please enter your email."
    user = next((user for user in users if user['email'] == email), None)
    if not user:
        return "No account with that email exists."
    # Generate a new random password for the user
            new_password = generate_random_password()
    # Update the user's password in the users list or database
    user['password'] = new_password
    # Send an email to the user with the new password
    # Render the forgotpass.html template with the new password
        return render_template('newpass.html', new_password=new_password)
           return render_template('forgot.html')
        def generate_random_password(length=12):
alphabet = string.ascii_letters + string.digits + string.punctuation
```

```
    password = ''.join(secrets.choice(alphabet) for i in range(length))
    return password
if __name__ == '__main__':
    app.run(debug=True)
```

- **Sample Html Code:**

```
<html>
  <head>
    <title>
      HOMEPAGE
    </title>
    <link rel="stylesheet" href="{{url_for('static',filename='css/style.css')}}">
  </head>
  <body>
    <div class="container">
     <header>
          <h3><b>CHRONIC KIDNEY DISEASE
PREDICTION</b></h3>
          <ul>

               <li><a href="#one">Home</a></li>
```

```
                        <li><a href="/seven">Upload Dataset</a></li>
    <li><a href="/prediction">Prediction</a></li>
                        <li><a href="/accuracy">Accuracy</a></li>
    <li><a href="/metrices">Metrices</a></li>
    <li><a href="/graph">Comparison</a></li>
                        <li><a href="/fourth">ContactUS</a></li>
    <li><a href="/">Log Out</a></li>
                    </ul>
            </header>
<div class="content" >



<center>
<h1><span style="border:7px solid #fff; padding:20px;">Prediction Using
Machine Learning Algorithms!!</span></h1>
</center>
            </div>
</body>
</html>
```

- **Sample CSS Code:**

```
*
{
   margin:0;
   padding:0;
   font-family:'poppins',sans-serif;
   box-sizing:border-box;
}
.container
{
   width:100%;
   height:100vh;
   background-image:linear-
gradient(rgba(0,0,50,0.8),rgba(0,0,50,0.8)),url(index.jpg);
   background-position: center;
   background-size:cover;
   position: relative;
}
.container h1{
   color:whitesmoke;
   padding-top: px;
   text-align: center;
```

font-family: Cambria, Cochin, Georgia, Times, 'Times New Roman', serif;}

- **KNN Algorithm Code:**

```
import pandas as pd
# Load the dataset using pandas
data = pd.read_csv('kidney.csv')
# Split the data into features and target
features = data.iloc[:, :-1].values
target = data.iloc[:, -1].values
# Split the dataset into training and test sets
split_ratio = 0.7
split_index = int(split_ratio * len(features))
train_features = features[:split_index]
train_target = target[:split_index]
test_features = features[split_index:]
test_target = target[split_index:]
# Define the number of nearest neighbors to consider
k =3
# Define a function to calculate the Euclidean distance between two data points
def euclidean_distance(x1, x2):
    distance = 0
    for i in range(len(x1)):
        distance += (x1[i] - x2[i])**2
    return distance**0.5
# Define a function to find the k nearest neighbors of a given data point
def get_neighbors(train_data, train_labels, test_instance, k):
    distances = []
    for i in range(len(train_data)):
dist = euclidean_distance(test_instance, train_data[i])
  distances.append((train_labels[i], dist))
distances.sort(key=lambda x: x[1])
```

```
neighbors = [distances[i][0] for i in range(k)]
return neighbors
    # Define a function to make predictions for a given test dataset
    def predict(train_data, train_labels, test_data, k):
      predictions = []
      for i in range(len(test_data)):
        neighbors = get_neighbors(train_data, train_labels, test_data[i], k)
        counts = {}
        for j in range(len(neighbors)):
          response = neighbors[j]
          if response in counts:
            counts[response] += 1
          else:
            counts[response] = 1
        sorted_counts = sorted(counts.items(), key=lambda x: x[1], reverse=True)
        predictions.append(sorted_counts[0][0])
      return predictions


    def accuracy(predictions, actual):
      correct = 0
      for i in range(len(predictions)):
        if predictions[i] == actual[i]:
          correct += 1
      return (correct / len(predictions)) * 100
    predictions = predict(train_features, train_target, test_features, k)
```

- **Decision Tree Algorithm Code:**

```
    import pandas as pd

    import numpy as np

    class DecisionNode:

      def __init__(self, feature=None, threshold=None, left=None, right=None,
```

```
value=None):
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value
    def is_leaf_node(self):
        return self.value is not None
class DecisionTree:
    def __init__(self, max_depth=None):
        self.max_depth = max_depth
    def fit(self, X, y):
        self.n_classes_ = len(set(y))
        self.tree_ = self._grow_tree(X, y)
    def predict1(self, X):
        return [self._predict(inputs) for inputs in X]
    def _best_split(self, X, y):
        m = y.size
        if m <= 1:
            return None, None
        num_parent = [np.sum(y == c) for c in range(self.n_classes_)]
        best_gini = 1.0 - sum((n / m) ** 2 for n in num_parent)
        best_idx, best_thr = None, None
```

```
        for idx in range(X.shape[1]):
thresholds, classes = zip(*sorted(zip(X[:, idx], y)))
            num_left = [0] * self.n_classes_
            num_right = num_parent.copy()
            for i in range(1, m):
                c = classes[i - 1]
                num_left[c] += 1
                num_right[c] -= 1
                gini_left = 1.0 - sum((num_left[x] / i) ** 2 for x in
range(self.n_classes_))
                gini_right = 1.0 - sum((num_right[x] / (m - i)) ** 2 for x in
range(self.n_classes_))
                gini = (i * gini_left + (m - i) * gini_right) / m
                if thresholds[i] == thresholds[i - 1]:
                    continue
                if gini < best_gini:
                    best_gini = gini
                    best_idx = idx
                    best_thr = (thresholds[i] + thresholds[i - 1]) / 2
            return best_idx, best_thr
        def _grow_tree(self, X, y, depth=0):
        num_samples_per_class = [np.sum(y == i) for i in range(self.n_classes_)]
```

```
            predicted_class = np.argmax(num_samples_per_class)
            node = DecisionNode(value=predicted_class)
                if depth < self.max_depth:
               idx, thr = self._best_split(X, y)
               if idx is not None:
                  indices_left = X[:, idx] <= thr
                  X_left, y_left = X[indices_left], y[indices_left]
          X_right, y_right = X[~indices_left], y[~indices_left]
          node = DecisionNode(feature=idx, threshold=thr, left=self._grow_tree(X_left,
y_left, depth + 1)
                              right=self._grow_tree(X_right, y_right, depth + 1))
    return node


  def _predict(self, inputs):
    node = self.tree_
    while not node.is_leaf_node():
      if inputs[node.feature] <= node.threshold:
        node = node.left
      else:
        node = node.right
    return node.value
```

- **Decision Tree Algorithm Accuracy Code:**

```
import pandas as pd
import numpy as np

class DecisionNode:
    def __init__(self, feature=None, threshold=None, left=None, right=None, value=None):
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value

    def is_leaf_node(self):
        return self.value is not None

class DecisionTree:
    def __init__(self, max_depth=None):
        self.max_depth = max_depth

    def fit(self, X, y):
        self.n_classes_ = len(set(y))
        self.tree_ = self._grow_tree(X, y)

    def predict(self, X):
        return [self._predict(inputs) for inputs in X]

    def _best_split(self, X, y):
        m = y.size
        if m <= 1:
```

```
            return None, None

        num_parent = [np.sum(y == c) for c in range(self.n_classes_)]
        best_gini = 1.0 - sum((n / m) ** 2 for n in num_parent)
        best_idx, best_thr = None, None

        for idx in range(X.shape[1]):
            thresholds, classes = zip(*sorted(zip(X[:, idx], y)))
            num_left = [0] * self.n_classes_
            num_right = num_parent.copy()

            for i in range(1, m):
                c = classes[i - 1]
                num_left[c] += 1
                num_right[c] -= 1
                gini_left = 1.0 - sum((num_left[x] / i) ** 2 for x in range(self.n_classes_))
                gini_right = 1.0 - sum((num_right[x] / (m - i)) ** 2 for x in
range(self.n_classes_))
                gini = (i * gini_left + (m - i) * gini_right) / m

                if thresholds[i] == thresholds[i - 1]:
                    continue

                if gini < best_gini:
                    best_gini = gini
                    best_idx = idx
                    best_thr = (thresholds[i] + thresholds[i - 1]) / 2

        return best_idx, best_thr

    def _grow_tree(self, X, y, depth=0):
```

```
        num_samples_per_class = [np.sum(y == i) for i in range(self.n_classes_)]
        predicted_class = np.argmax(num_samples_per_class)
        node = DecisionNode(value=predicted_class)

        if depth < self.max_depth:
            idx, thr = self._best_split(X, y)
            if idx is not None:
                indices_left = X[:, idx] <= thr
                X_left, y_left = X[indices_left], y[indices_left]
                X_right, y_right = X[~indices_left], y[~indices_left]
                node = DecisionNode(feature=idx, threshold=thr,
left=self._grow_tree(X_left, y_left, depth + 1),
                                    right=self._grow_tree(X_right, y_right, depth + 1))

        return node

    def _predict(self, inputs):
        node = self.tree_
        while not node.is_leaf_node():
            if inputs[node.feature] <= node.threshold:
                node = node.left
            else:
                node = node.right
        return node.value

df = pd.read_csv("kidney.csv")
X = df.drop(columns=["CLASS"]).values
y = df["CLASS"].values

train_size = int(0.7 * len(df))
X_train, X_test = X[:train_size], X[train_size:]
```

```
y_train, y_test = y[:train_size], y[train_size:]

# Create and train the decision tree model
dt = DecisionTree(max_depth=5)
dt.fit(X_train, y_train)

# Make prediction on the test set
y_pred = dt.predict(X_test)
d_accuracy = sum(y_pred == y_test) / len(y_test) *100
print(d_accuracy)
```

- **KNN Algorithm Accuracy Code:**

```
import pandas as pd

# Load the dataset using pandas
data = pd.read_csv('kidney.csv')

# Split the data into features and target
features = data.iloc[:, :-1].values
target = data.iloc[:, -1].values

# Split the dataset into training and test sets
split_ratio = 0.7
split_index = int(split_ratio * len(features))
train_features = features[:split_index]
train_target = target[:split_index]
test_features = features[split_index:]
test_target = target[split_index:]

# Define the number of nearest neighbors to consider
k =3

# Define a function to calculate the Euclidean distance between two data points
def euclidean_distance(x1, x2):
    distance = 0
    for i in range(len(x1)):
        distance += (x1[i] - x2[i])**2
    return distance**0.5

# Define a function to find the k nearest neighbors of a given data point
```

```
def get_neighbors(train_data, train_labels, test_instance, k):
    distances = []
    for i in range(len(train_data)):
        dist = euclidean_distance(test_instance, train_data[i])
        distances.append((train_labels[i], dist))
    distances.sort(key=lambda x: x[1])
    neighbors = [distances[i][0] for i in range(k)]
    return neighbors

# Define a function to make predictions for a given test dataset
def predict(train_data, train_labels, test_data, k):
    predictions = []
    for i in range(len(test_data)):
        neighbors = get_neighbors(train_data, train_labels, test_data[i], k)
        counts = {}
        for j in range(len(neighbors)):
            response = neighbors[j]
            if response in counts:
                counts[response] += 1
            else:
                counts[response] = 1
        sorted_counts = sorted(counts.items(), key=lambda x: x[1], reverse=True)
        predictions.append(sorted_counts[0][0])
    return predictions

def accuracy(predictions, actual):
    correct = 0
    for i in range(len(predictions)):
        if predictions[i] == actual[i]:
            correct += 1
    return (correct / len(predictions)) * 100
```

```
# Make predictions for the test dataset
predictions = predict(train_features, train_target, test_features, k)

# Calculate the accuracy of the predictions
acc = accuracy(predictions, test_target)
print(acc)
```

CHAPTER 7

OUTPUT SCREENS

HOME PAGE:

Figure 7.1.1 Home Page

MAIN PAGE:

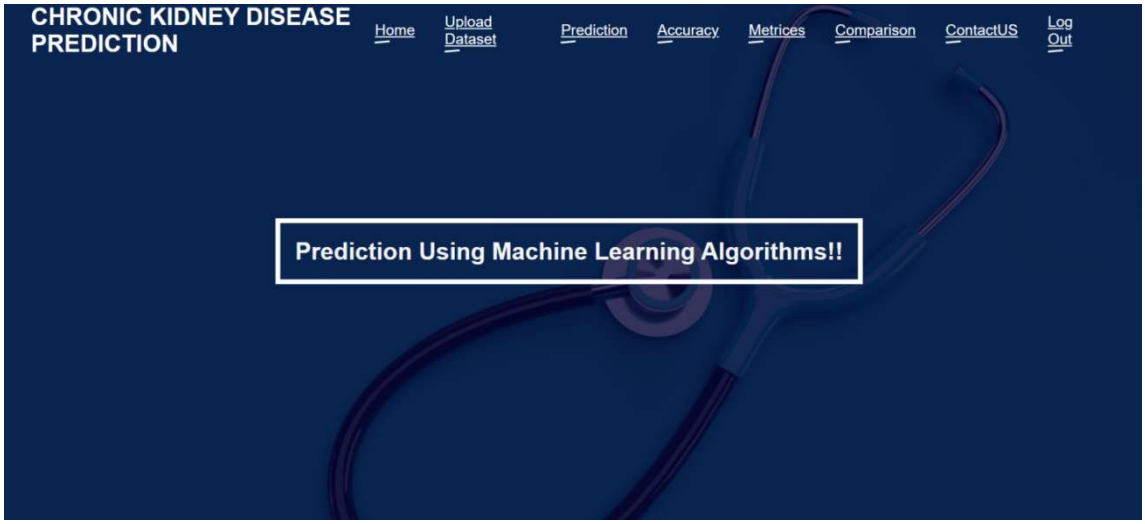


Figure 7.1.2 Main Page

DATASET UPLOAD PAGE:

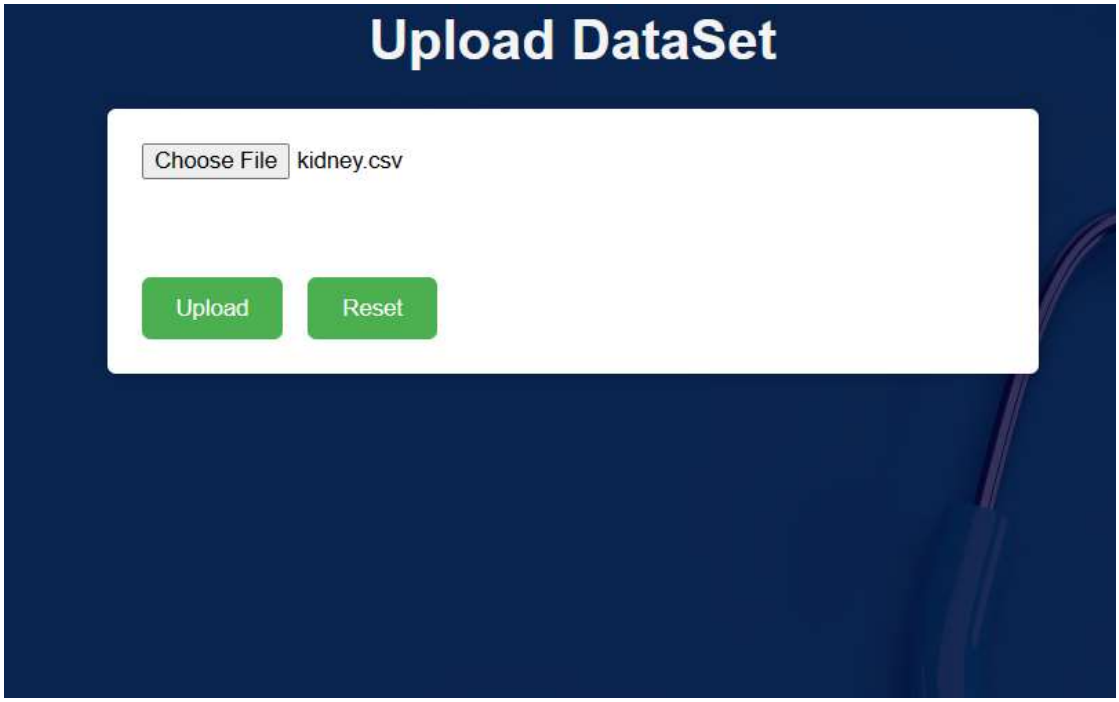


Figure 7.1.3 Upload Dataset Page

DATASET VIEW PAGE:

AGE	WEIGHT	SG	Alb	eGFR	Wbc	CLASS
57.000000	89.000000	1.020000	4.000000	80.00	7800.000000	0
78.000000	87.000000	1.020000	3.000000	78.00	6300.000000	0
76.000000	90.000000	1.010000	4.000000	90.00	7500.000000	0
70.000000	92.000000	1.005000	2.000000	75.00	6700.000000	0
65.000000	94.000000	1.010000	4.000000	89.00	4300.000000	0
80.000000	74.000000	1.015000	3.000000	87.00	10000.000000	0
72.000000	86.000000	1.010000	2.000000	75.00	4500.000000	0
75.000000	84.000000	1.005000	1.000000	60.00	12000.000000	2
89.000000	82.000000	1.015000	1.000000	35.00	11000.000000	2
72.000000	73.000000	1.010000	2.000000	31.00	11500.000000	2
64.000000	74.000000	1.015000	4.000000	95.00	7900.000000	0
69.000000	81.000000	1.020000	1.000000	37.00	12000.000000	1
67.000000	63.000000	1.010000	0.000000	27.00	12500.000000	1
75.000000	76.000000	1.010000	1.000000	29.00	10000.000000	1
63.000000	89.000000	1.010000	2.000000	68.00	7900.000000	0

Figure 7.1.4 View Data Page

PREDICTION PAGE:

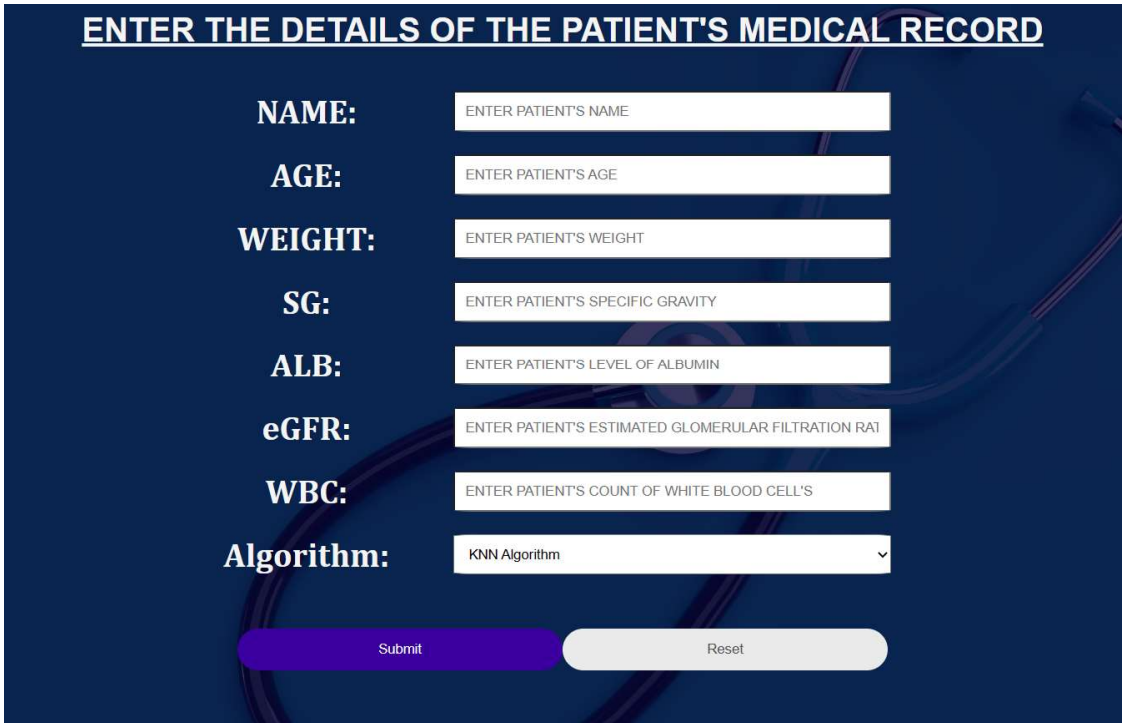


Figure 7.1.5 Prediction Page

RESULT PAGE:

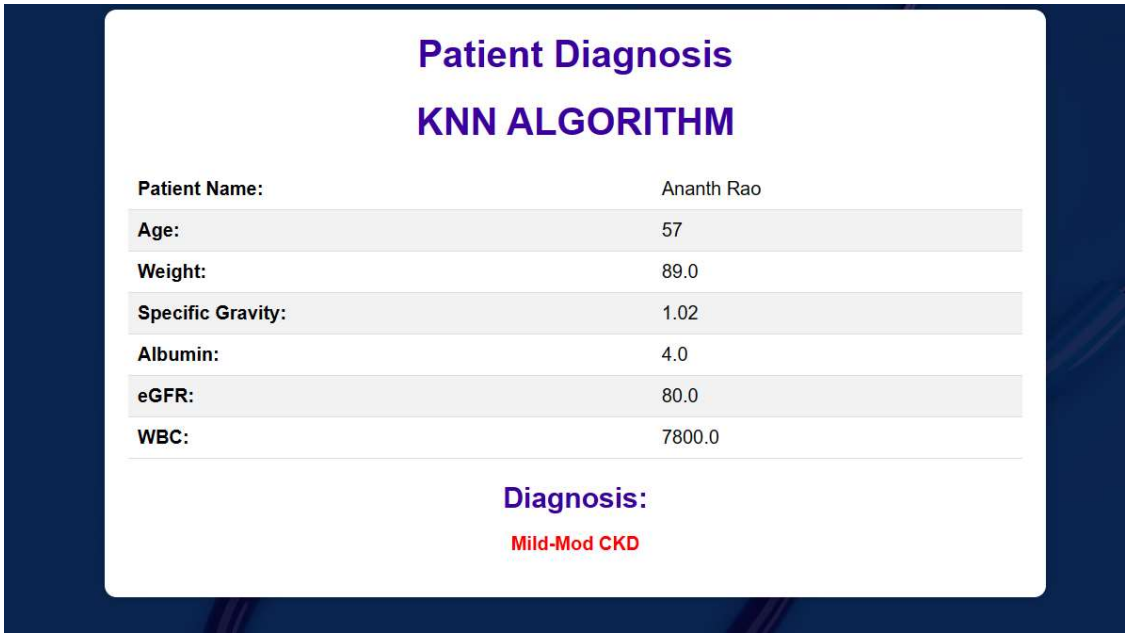


Figure 7.1.6 Algorithm-1 Result Page

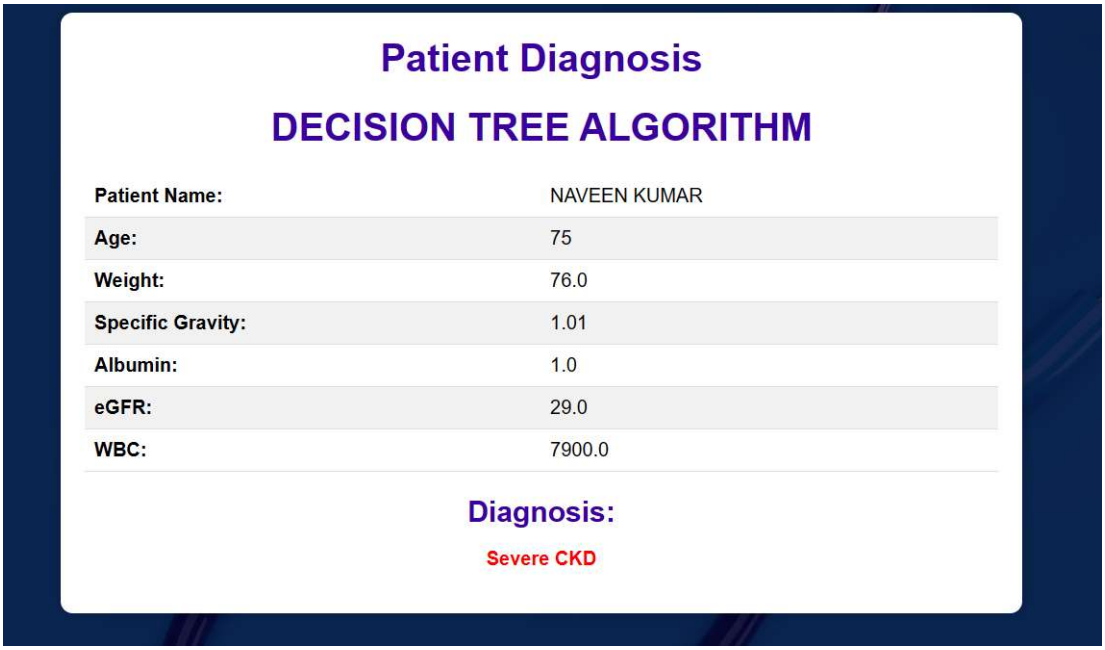
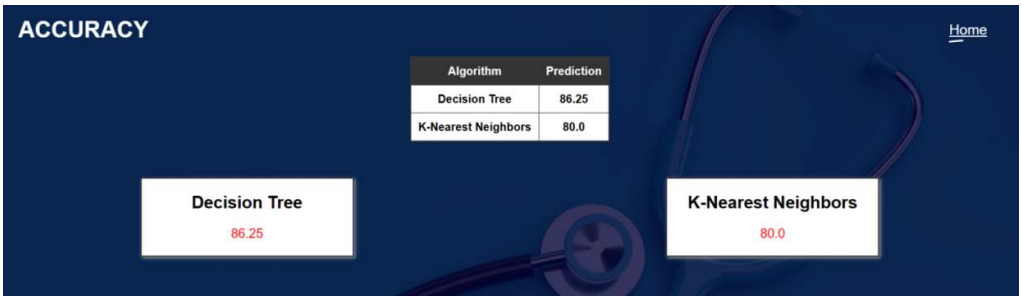


Figure 7.1.7 Algorithm-2 Result Page

ACCURACY PAGE:



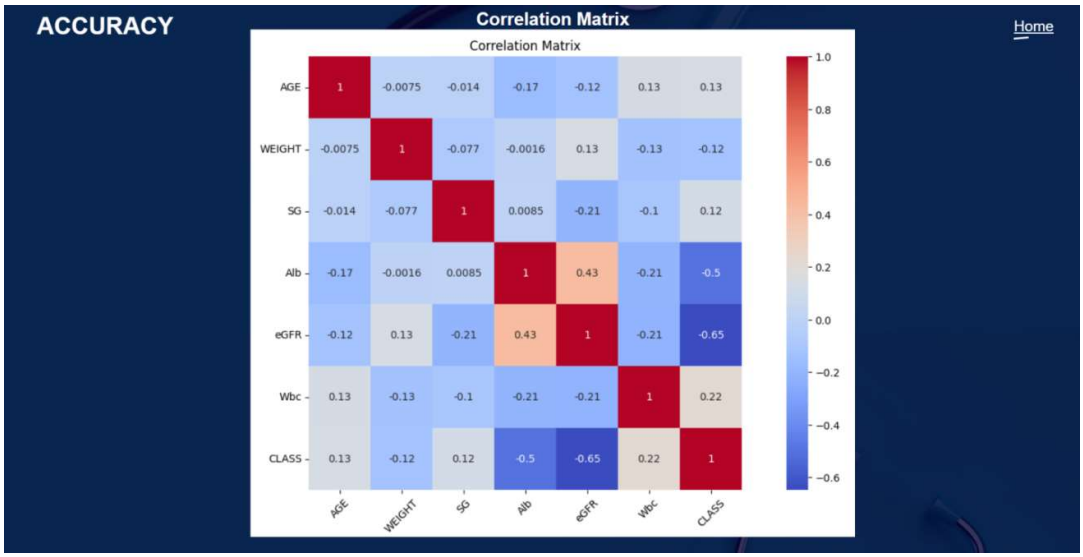


Figure 7.1.8 Accuracy Page

METRICS PAGE:

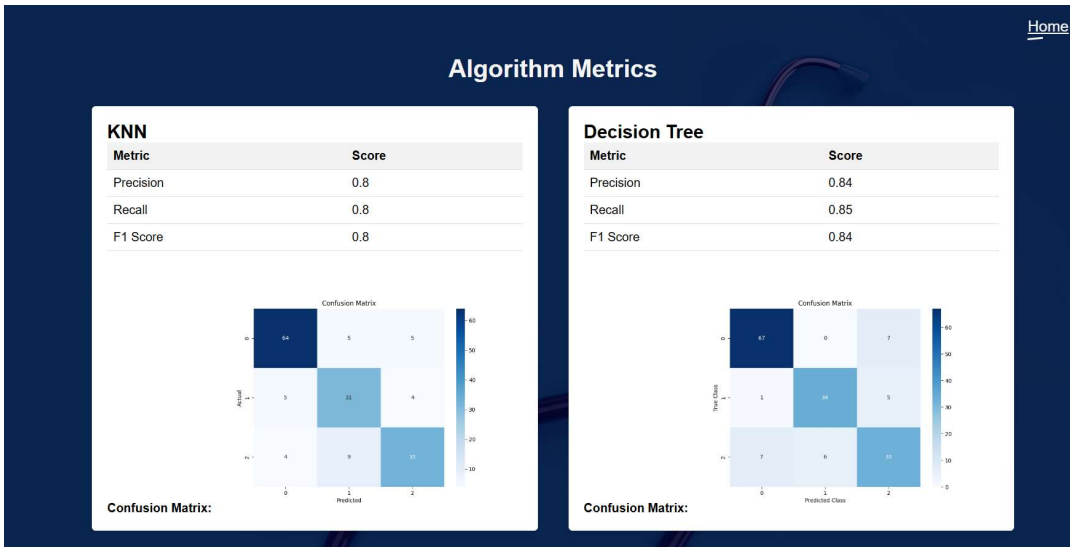


Figure 7.1.9 Metrics Page

METRICS GRAPH PAGE:

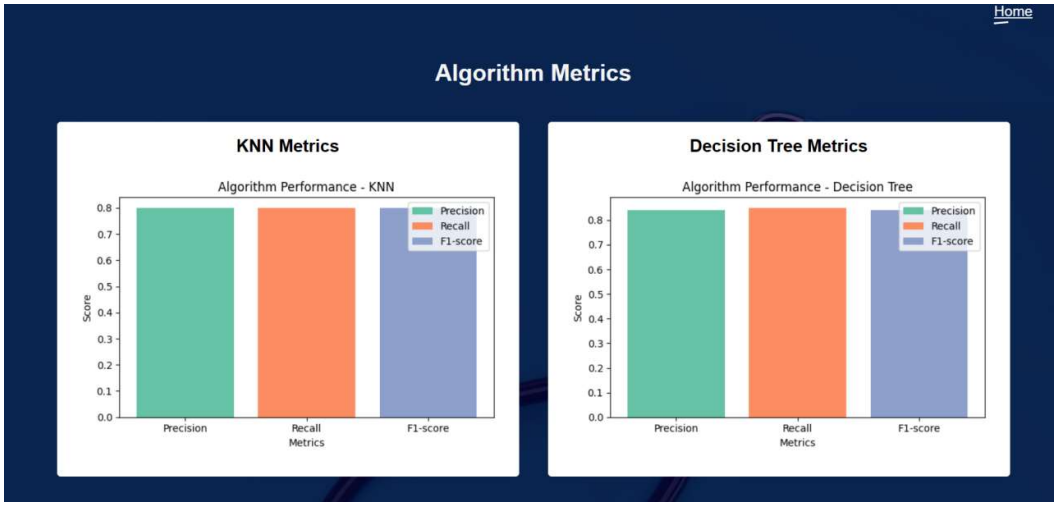


Figure 7.2.0 Metrics Graph Page

ALGORITHMS COMPARISION GRAPH PAGE:

Figure 7.2.1 Algorithms Comparison Graph Page

CHAPTER 8

SYSTEM TESTING

8.1 SYSTEM TESTING:

System testing makes sure that the total software satisfies the requirements. It evaluates a configuration to ensure that the consequences are understood and foreseeable. An illustration of a system test is the configuration-based system integration test. System testing is based on process flows and models, with a focus on links and interaction points that have previously been established.

8.2 UNIT TESTING:

Making test cases as part of unit testing entails confirming that the program's basic logic is solid and that genuine inputs result in valid outputs. The program's decision paths and code flow should both be reviewed. Every piece of software that goes into a program is tested in this approach. It is finished after the completion of each component but before they are joined. This is an invasive test of the structure that depends on knowing how it was created. Unit tests execute rapid tests at the level of individual pieces and assess a single business method, software program, or system configuration. Unit tests guarantee that each individual route of a business process runs exactly as stated in the documentation and has explicit inputs.

8.3 INTEGRATION TESTING:

Integration tests are intended to check whether a program's component pieces perform as a single program. Testing is focused on events and is generally concerned with the functioning of screens or fields. Even though each component functioned brilliantly on its own, as evidenced by competent unit testing, integration tests showed that the components are coherent and consistent. Integration testing is aimed to uncover problems that appear when numerous components are integrated together.

8.4 BLACK BOX TESTING:

Black box testing includes analysing software without having any knowledge of how it performs, how it is constructed, or what language it was written in. Black box tests need to be built from a clear source document, like a blueprint or standards document, much like the majority of other kinds of tests.

8.5 TEST CASES:

S.NO	Test Cases	Input	Expected Output	Actual Output	Pass/Fail
1	If Login details are not filled.	A warning statement is to be displayed.	Users cannot get access to the next page.	User cannot get access to the next page.	Pass
2	If user enter the exisiting email while signup	Passing user email id.	The output should be already email id exists	Already email id exists	Pass.
3	If user choose KNN algorithm	Passing patient's details.	To predict the type of chronic kidney disease	It predicts kidney disease type.	Pass.
4	Dataset is uploaded or not.	Upload dataset.	Dataset is uploaded.	Dataset is uploaded.	Pass

CHAPTER 9

CONCLUSION AND FUTURE SCOPE

9.1 CONCLUSION:

In conclusion, the system is aimed to predict 'CHRONIC KIDNEY DISEASE' from the kidney patient's dataset using machine learning algorithms namely 'KNN' and 'Decision Tree'. Each of this algorithm demonstrated promising results in prediction. However the performance varies among the algorithms. The system utilized the concept of 'Machine Learning Algorithms' in prediction of chronic kidney disease and paved a way for advancement of healthcare analytics. In further deep learning algorithm can also be utilised for enhancement of the system.

9.2 FUTURE SCOPE:

With potential improvements in a number of domains, the prediction of chronic kidney disease using machine learning has a promising future. Here are some noteworthy details to investigate in the future:

1. Enhanced Accuracy
2. Feature Selection and Extraction
3. Real-time Monitoring
4. Personalized Treatment
5. Integration of Multiple Data Sources
6. Collaborative Research and Data Sharing

Overall, the potential for improving diagnosis, treatment, and patient outcomes from the prediction of chronic renal disease using machine learning is enormous. Future healthcare interventions may be more individualized and successful as a result of ongoing study and innovation in this area.

REFERENCES

[1] **A. Ogunleye** and **A. Nishanth**, ‘‘*XGBoost model for chronic kidney disease diagnosis*,’’ IEEE/ACM Trans. Comput. Biol. Bioinf., vol. 17, no. 6, pp. 2131–2140, Nov. 2020.

[2] **A. U. Haq**, **J. P. Li**, **J. Khan**, **M. H. Memon**, **S. Nazir**, **M. P. N. Wickramasingh, G. A. Khan**, and **A. Aliss**, ‘‘*Intelligent machine learning approach for effective recognition of diabetes in E-healthcare using clinical data*,’’ Sensors, vol. 20, no. 9, p. 2649, May 2020.

[3] **B. Khan**, **R.S. Walse, G.D. Kurundkar, S.D. Khamitkar**, and **S. Kim**, ‘‘*An empirical evaluation of machine learning techniques for chronic kidney disease prophecy*,’’ IEEE Access, vol. 8, pp. 55012–55022, 2020.

[4] **J. Snegha**, ‘‘*Early prediction of chronic kidney disease by using machine learning techniques*,’’ Amer. J. Comput. Sci. Eng. Survey, vol. 8, no. 2, p. 7, 2020.

[5] **J. Van Eiyck, J. Remon, X. Feng**, **T. Sun**, **S. Zhu**, and **Z. Ye**, ‘‘*Comparison and development of machine learning tools in the prediction of chronic kidney disease progression*,’’ J. Transl. Med., vol. 17, p. 119, Dec. 2019.

[6] **N. Tangri**, **N. Borisagar, D. Barad** et al., ‘‘*Multinational assessment of accuracy of equations for predicting risk of kidney failure: A meta-analysis*,’’ J. Amer. Med. Assoc., vol. 315, no. 2, pp. 164–174, 2016.

[7] **O. Viktorsdottir, Aljaaf, A.J.** 2018 Early Prediction of “*Chronic Kidney Disease Using Machine Learning Supported by Predictive Analytics*”. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC). Wellington. New Zealand.

[8] **S. Drall**, **G. S. Drall**, **S. Singh**, **D. Barad** and **B. B. Naib**, ‘‘*Chronic kidney disease prediction using machine learning: A new approach*,’’ Int. J. Manage., Technol. Eng., vol. 8, pp. 278–287, May 2018.

[9] **S. Shankar**, **S. Verma**, **S. Elavarthy**, **T. Kiran**, and **P. Ghuli**, ‘‘*Analysis and prediction of chronic kidney disease*,’’ Int. Res. J. Eng. Technol., vol. 7, no. 5, May 2020, pp. 4536–4541.

[10] **W. Gunarathne, K. D. M Perera**, and **K. A. D. C. P Kahandawaarachchi**, "*Performance evaluation on machine learning classification techniques for disease classification and forecasting through data analytics for chronic kidney disease (CKD)*,"in Oct. 2017