

CHRONIC KIDNEY DISEASE PREDICTION USING MACHINE LEARNING ALGORITHM

Submitted in partial fulfillment for the award of the degree of

Master of Science

In

Data Science

By

POOJA

21MDT0140

Under the guidance of

Dr. PRAKASH M

School of Advanced Sciences

VIT, Vellore



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

April, 2023


DECLARATION

I hereby declare that the thesis entitled " **CHRONIC KIDNEY DISEASE PREDICTION USING MACHINE LEARNING** " submitted by me, for the award of the degree of *Master of Science in Data Science* to VIT is a record of bonafide work carried out by me under the supervision of **Dr. PRAKASH M.**

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore

Date: 11.04.2023



Signature of the candidate
POOJA K

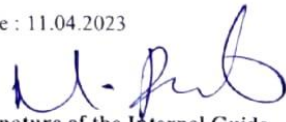
CERTIFICATE

This is to certify that the thesis entitled "**CHRONIC KIDNEY DISEASE PREDICTION USING MACHINE LEARNING**" submitted by **POOJA K (21MDT0140)**, School of Advanced Sciences, Vellore Institute of Technology, Vellore for the award of the degree of *Master of Science in Data Science*, is a record of bonafide work carried out by him/her under my supervision during the period, 01.01.2023 to 11.04.2023, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion, meets the necessary standards for submission.

Place :Vellore

Date : 11.04.2023



Signature of the Internal Guide
Department of Mathematics

SAS, VIT-Vellore



Signature of the External Examiner



Head, Department of Mathematics

SAS, VIT-Vellore

ABSTRACT

Chronic kidney disease (CKD) is a growing public health concern worldwide, and early detection and intervention are crucial for improving patient outcomes. Machine learning algorithms have shown promise in accurately predicting CKD, allowing for early intervention and management. This study aims to develop a predictive model for CKD using machine learning algorithms such as logistic regression, random forest, and support vector machines. The dataset used in this study comprises demographic, clinical, and laboratory variables from patients with CKD. The accuracy of each algorithm will be compared using metrics such as sensitivity, specificity, and area under the curve. The results of this study will help to identify patients at risk of developing CKD and provide a framework for implementing machine learning algorithms in clinical practice.

ACKNOWLEDGEMENTS

With immense pleasure and a deep sense of gratitude, I wish to express my sincere thanks to my guide Dr. PRAKASH M, School of Advanced Sciences, VIT, Vellore without his motivation and continuous encouragement, this research would not have been successfully completed.

I am grateful to the Chancellor of VIT, Vellore, Dr. G. Viswanathan, the Vice Presidents, and the Vice Chancellor for motivating me to carry out research in the Vellore Institute of Technology, Vellore, and also for providing me with infrastructural facilities and many other resources needed for my research

I express my sincere thanks to Dr. N Arunai Nambi Raj, Dean, School of Advanced Sciences, VIT, Vellore, for her kind words of support and encouragement. I like to acknowledge the support rendered by my classmates in several ways throughout my researchwork.

I wish to thank Dr. Jagadeesh Kumar M.S. Head of Department Mathematics, School of Advanced Sciences, VIT, Vellore for his encouragement and support.

I wish to extend my profound sense of gratitude to my parents and friends for all the support they made during my research and also for providing me with encouragement whenever required.

Signature of the student

POOJA K

Table of Contents

Chapter 1	6
INTRODUCTION	6
1.1 BACKGROUND	6
1.2 PROJECT STATEMENT	6
1.3 OBJECTIVES.....	6
1.4 SCOPE OF PROJECT	7
Chapter 2	9
LITERATURE SURVEY	9
2.1 SUMMARY OF THE EXISTING WORKS	9
2.2 CHALLENGES PRESENT IN EXISTING SYSTEM.....	10
Chapter 3	11
REQUIREMENTS.....	11
3.1 HARDWARE REQUIREMENTS	11
3.2 SOFTWARE REQUIREMENTS.....	11
3.3 GANTT CHART.....	11
Chapter 4	13
ANALYSIS AND DESIGN	13
4.1 PROPOSED METHODOLOGY	13
4.2 SYSTEM ARCHITECTURE.....	14
4.3 MODULE DESCRIPTIONS	15
Chapter 5	31
IMPLEMENTATION AND TESTING	31
5.1 DATA SET	31
5.2 SAMPLE CODE	31
5.3 SAMPLE OUTPUT.....	44
Chapter 6	61
RESULTS.....	61
6.1 CONCLUSIONS AND FUTURE WORK	61
6.2 REFERENCES.....	62

Chapter 1

INTRODUCTION

1.1 BACKGROUND

Chronic kidney disease (CKD) is a serious health condition affecting millions of people worldwide. It is a progressive and irreversible disease that can lead to kidney failure and other complications such as cardiovascular disease, anemia, and bone disease. Early detection and intervention are essential to slow the progression of the disease and improve patient outcomes.

Machine learning algorithms have shown great potential in predicting the onset and progression of CKD. These algorithms are able to analyze vast amounts of data and spot trends that are challenging for people to notice. By using machine learning, it may be possible to identify patients at risk of developing CKD at an earlier stage, allowing for earlier intervention and management.

The aim of this study is to develop a predictive model for CKD using machine learning algorithms such as logistic regression, random forest, and support vector machines. The dataset used in this study comprises demographic, clinical, and laboratory variables from patients with CKD. The accuracy of each algorithm will be compared using metrics such as sensitivity, specificity, and area under the curve.

The development of an accurate and reliable predictive model for CKD using machine learning algorithms has the potential to revolutionize the management of this disease. It could allow for earlier detection and intervention, ultimately improving patient outcomes and reducing the burden on healthcare systems.

1.2 PROJECT STATEMENT

CKD frequently results in kidney donations or permanent dialysis. High likelihood of CKD is also increased by a family history of kidney illness. According to published research, CKD affects almost one in three individuals with a diabetes diagnosis.

1.3 OBJECTIVES

The main objective of using machine learning algorithms for predicting chronic kidney disease (CKD) is to develop a reliable and accurate model that can identify individuals at high risk for developing the disease. The specific objectives of using machine learning algorithms for CKD prediction include:

Early detection: Identifying individuals at high risk for developing CKD at an early stage can help prevent or delay the onset of the disease, and enable timely interventions such as lifestyle modifications or medication.

Improved diagnosis: Machine learning algorithms can help improve the accuracy of CKD diagnosis by identifying patterns and relationships between different variables, such as age, sex, blood pressure, and laboratory tests.

Personalized medicine: The predictive model can help identify personalized treatment options for individuals with CKD, based on their risk profile and other clinical characteristics.

Healthcare cost reduction: Early detection and prevention of CKD can help reduce the burden on healthcare systems and lower the cost of treatment, by avoiding costly interventions such as dialysis or kidney transplantation.

Research: The predictive model can also be used for research purposes, such as identifying new risk factors for CKD or understanding the mechanisms underlying the disease.

Overall, the objective of using machine learning algorithms for CKD prediction is to improve the quality of care for individuals with the disease, reduce healthcare costs, and advance our understanding of the disease.

1.4 SCOPE OF THE PROJECT

The scope of this project is to develop a predictive model for Chronic Kidney Disease (CKD) using machine learning algorithms. The project aims to analyze a large dataset of demographic, clinical, and laboratory variables from patients with CKD and use this data to train and test machine learning algorithms such as random forest and logistic regression

The primary goal of this project is to develop a model that accurately predicts CKD, allowing for early detection and intervention. This will involve analyzing the various variables and their relationship with CKD, selecting appropriate features, and developing models that can effectively identify patients at risk of developing the disease.

The project will also involve evaluating the performance of the developed models, comparing their accuracy, sensitivity, specificity, and area under the curve. This evaluation will be critical to identify the best algorithm for the prediction of CKD.

The most difficult job in the medical field is to create a diagnostic method that uses a quick and precise algorithm that runs quickly and produces accurate results. Therefore,

developing a trustworthy and potent medical diagnosis system to assist the current diagnostic procedures has become a challenging task. Due to the complexity of the conventional diagnosis, computing techniques like logistic regression and random forest can be used to create the diagnosis. Due to time and data limitations, even a partial diagnosis of a disease can be very useful.

Based on a variety of variables present in the dataset, we use machine learning algorithms to ascertain whether the patient has chronic kidney disease at an early stage. Therefore, an automated tool that uses machine learning techniques to assess the patient's kidney health will be useful in helping physicians anticipate chronic kidney disease and, consequently, provide better care. The patient information in the collection pertains to people who have CKD already. When a random patient detail is provided as input after the model has been trained, it analyses and outputs the outcome.

Chapter 2

LITERATURE SURVEY

2.1 SUMMARY OF THE EXISTING WORKS

S.NO	Title/Author	Merits	De-Merits
1	Title:Survey on Chronic Kidney Disease Prediction System with Feature Selection and Feature Extraction using Machine Learning Technique. Author: Ajay kumar , Karthik Raja , Jebaz Sherwin , Revathi .	They created a (CNN)-based multimodal illness risk prediction algorithm using both organised and unstructured hospital data.	In the research mentioned above, the accuracy of the trained models is significantly influenced by the missing value filling method.
2	Title:Chronic Kidney Disease Prediction using Data Mining and Machine Learning Author: Adeeba Azmi, Amiksha Hingu, Ruchi Dholaria, Ms. Alvina Alphonso	The accuracy percentage of these models is higher than the accuracy percentage used in earlier research.	KNN and SVM are found to take longer to forecast the chronic kidney disease
3	Title:Chronic kidney disease diagnosis using machine learning. Author: Vijayaprabakaran, Pratheek Reddy, Puthin Kumar Reddy, Munnaf, Reddi Prasad	Compared the output of various versions. The Multiclass Decision forest algorithm, they found, provides greater accuracy than other algorithms.	Precision is Lower

4	<p>Title:Chronic kidney disease prediction using neural networks.</p> <p>Author: S.Priya1, S.Nirmal Kumar, G.Sibi Saravanan, E.Pandiyan, Ashuthosh Kumar Pandey</p>	The mistake rate is extremely low.	Time management is improved.
5	<p>Title:Chronic Kidney Disease Prediction using Machine Learning</p> <p>Author: Reshma S , Salma Shaji , S R Ajina , Vishnu Priya S , Janisha A</p>	According to experimental findings, AdaBoost performed slightly worse than Logit Boost.	The test's outcome fell short of a perfect score.

2.2 CHALLENGES PRESENT IN EXISTING SYSTEM

1. Low precision
2. There are issues with this division.
3. Feature extraction is inaccurate.
4. Low accuracy and heavy computational load are expected.

Chapter 3

REQUIREMENTS

3.1 HARDWARE REQUIREMENT

Since a contract for the system's implementation may be based on the hardware requirements, they should be a comprehensive and consistent specification of the system as a whole. They serve as the foundation for system design for software engineers. It demonstrates what the system does, not how it ought to be used.

PROCESSOR : Intel I5
RAM : 4GB
HARD DISK : 40 GB

3.2 SOFTWARE REQUIREMENTS

The system's specification can be found in the software requirements document. It ought to include both a definition of the requirements and a description of them. Instead of a set of how the system should operate, it is a set of what it should do. The software requirements specification is based on the software requirements. It can be used to estimate costs, plan team activities, complete tasks, and monitor the team's progress throughout the development activity.

PYTHON IDE : Anaconda Jupyter Notebook
PROGRAMMING LANGUAGE : Python

3.3 GANTT CHART

TASK	AUG	SEP	OCT	NOV
TASK 1				

TASK 2				
TASK 3				
TASK 4				
TASK 5				
TASK 6				
TASK 7				
TASK 8				

Chapter 4

ANALYSIS AND DESIGN

4.1 PROPOSED METHODOLOGY

(a) To predict chronic kidney disease using machine learning algorithms, the following methodology could be proposed:

(b) Data Collection: Collect a dataset of patients diagnosed with chronic kidney disease along with their medical history, demographic information, and laboratory test results.

(c) Data Pre processing: Clean the dataset by removing missing or irrelevant data, dealing with outliers, and performing feature selection to identify the most important predictors.

(d) Data Splitting: Divide the dataset into training and testing sets. The training set will be used to train the machine learning models, while the testing set will be used to evaluate the performance of the models.

(e) Model Selection: Select a few machine learning algorithms that are suitable for classification tasks and compare their performance using evaluation metrics such as accuracy, precision, recall, and F1 score.

(f) Hyperparameter Tuning: Optimize the parameters of the best-performing machine learning algorithm using techniques such as grid search or randomized search.

(g) Model Evaluation: Evaluate the performance of the optimized model using the testing set and report the performance metrics.

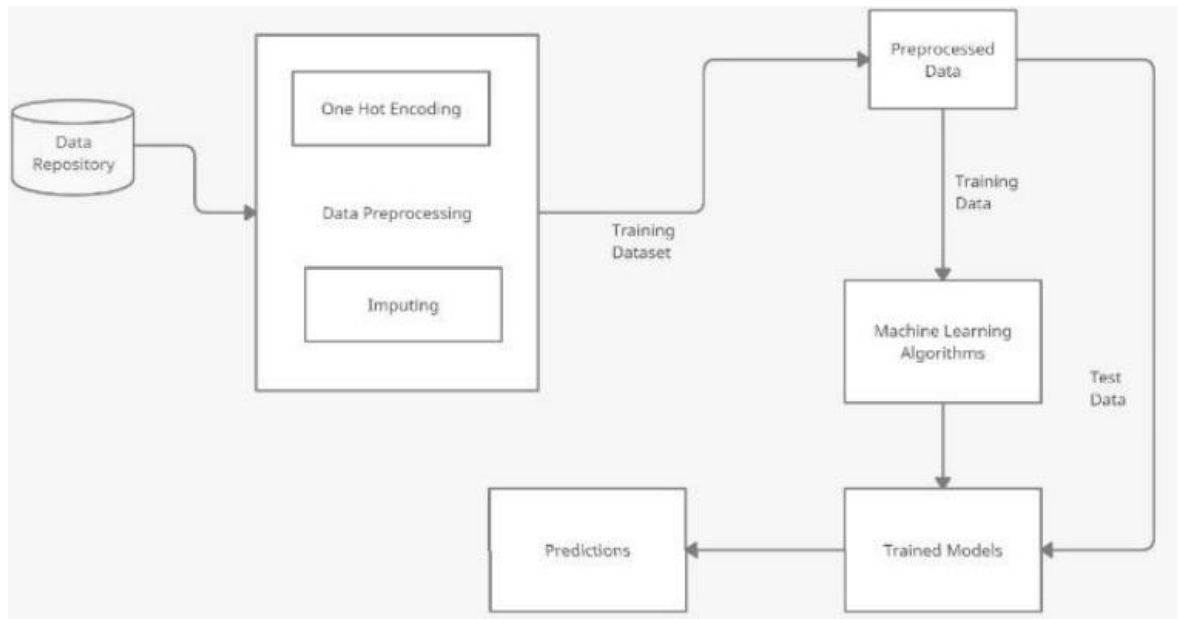
(h) Deployment: Once the model is finalized, deploy it to a web application or integrate it into an electronic medical record system for clinical use.

Some of the machine learning algorithms that could be used for chronic kidney disease prediction include logistic regression, decision trees, random forests, support vector machines, and artificial neural networks. It is important to note that the performance of

these algorithms can vary depending on the size and quality of the dataset, as well as the chosen evaluation metrics.

The CKD Dataset from the UCI repository is the dataset we use in this case. 400 samples from two different classes are included. Out of 25 ascribes, 11 are numeric and 13 are ostensible and one is class characteristic. There are a number of missing values in the data set. The patient's information, such as age, blood pressure, specific gravity, albumin, sugar, and red blood cells, is used in this dataset. Diabetes and high blood pressure are the root causes of CKD. Our numerous organs are impacted by diabetes, which is followed by high blood sugar. Therefore, it is essential to anticipate the disease as soon as possible. Some of the machine learning methods used to predict the disease are improvised in this study. The stage known as data pre-processing is where the encoded or distorted data is brought to a state where the machine can easily analyse it. A collection of data objects is what is known as a dataset. Information objects are marked by various elements, that guarantees the fundamental highlights of an item, for example, the mass of an actual item or the time at which an occasion guaranteed. There may be values missing from the dataset, which can be estimated or eliminated. Filling in missing values with the respective feature's mean, median, or mode value is the most common approach. We must convert the numerical values of type object to float64 because object values cannot be used in the analysis. The most frequently occurring value in that attribute column is used to change the null values in the categorical attributes. By assigning an integer value to each distinct attribute, label encoding converts categorical attributes into numerical attributes. The attributes will now be of the int type as a result of this. Each column's missing values are replaced with the mean value, which is predetermined from that column. We are making use of the imputer function for this function, which finds the mean value for each column. After the supplanting and encoding is finished, the information ought to be prepared, approved and tried. Our algorithms actually learn how to build a model by training on the data. The part of the dataset called validation is used to validate our various model fits or make the model better. Our model hypothesis is tested by testing the data.

4.2 SYSTEM ARCHITECTURE



4.3 MODULE DESCRIPTIONS

A module for chronic kidney disease prediction using machine learning algorithms could include the following components:

Data Collection and Pre processing: This component would handle the data collection and pre processing steps. It would include modules for cleaning the dataset, dealing with missing or irrelevant data, and performing feature selection.

Model Selection: This component would handle the selection of suitable machine learning algorithms for classification tasks. It would include modules for comparing the performance of different algorithms using evaluation metrics such as accuracy, precision, recall, and F1 score.

Hyperparameter Tuning: This component would optimize the parameters of the best-performing machine learning algorithm. It would include modules for performing techniques such as grid search or randomized search.

Model Training and Evaluation: This component would handle the training of the machine learning models on the training set and evaluate the performance of the optimized model using the testing set. It would include modules for reporting performance metrics and generating visualizations.

Deployment: This component would handle the deployment of the finalized model to a web application or integration into an electronic medical record system for clinical use.

The module could also include modules for data visualization, data pre-processing techniques such as normalization or scaling, and data splitting techniques such as stratified sampling. Additionally, the module could provide options for customizing the model parameters and selecting different evaluation metrics. Finally, the module could include documentation and tutorials to help users understand how to use the module effectively.

ALGORITHMS:

- 1.Random Forest Algorithm

- 2.logistic regression

RANDOM FOREST:

Random Forest is a popular machine learning algorithm used for both classification and regression tasks. It belongs to the ensemble learning category of algorithms, which means it combines multiple individual models to improve overall accuracy. In the case of random forest, the algorithm creates multiple decision trees on random subsets of the data and features, and then aggregates the predictions of all the individual trees to make a final prediction. This approach helps to reduce overfitting and increase generalization performance.

The key steps involved in building a random forest model are as follows:

Data Pre-processing: The first step is to pre-process the data by handling missing or irrelevant data, encoding categorical variables, and performing feature scaling.

Dataset splitting: Divide the dataset into training and testing sets.

Random Sampling: For each decision tree, a random subset of the training data is selected to create a bootstrap sample. This sampling is done with replacement, which means that some instances may be selected more than once.

Feature Sampling: For each node in the decision tree, a random subset of the features is selected as candidates for splitting the node. This helps to reduce the correlation between individual trees and improve generalization performance.

Decision Tree Training: For each bootstrap sample and feature subset, a decision tree is trained using a criterion such as Gini Impurity or Information Gain. The decision tree is trained recursively by splitting each node into two child nodes until a stopping criterion is met.

Prediction Aggregation: Once all the decision trees have been trained, they are used to make predictions on the testing set. The final prediction is obtained by

aggregating the predictions of all the individual trees, usually by taking the majority vote.

Model Evaluation: Finally, the performance of the random forest model is evaluated on the testing set using evaluation metrics such as accuracy, precision, recall, and F1 score.

Random Forest is a popular algorithm due to its high accuracy, robustness to outliers, and ability to handle high-dimensional datasets. However, it can be computationally expensive and may require careful tuning of hyperparameters such as the number of trees and the maximum depth of the decision trees.

LOGISTIC REGRESSION:

Logistic Regression is a popular machine learning algorithm used for classification tasks. It is a type of supervised learning algorithm that models the probability of the occurrence of a binary or multi-class outcome based on one or more predictor variables.

The key steps involved in building a logistic regression model are as follows:

Data Pre-processing: The first step is to pre-process the data by handling missing or irrelevant data, encoding categorical variables, and performing feature scaling.

Dataset splitting: Divide the dataset into training and testing sets.

Model Training: For logistic regression, the model is trained by minimizing a cost function such as the binary cross-entropy or the categorical cross-entropy loss. This is typically done using gradient descent or a variant thereof.

Prediction: Once the model is trained, it can be used to make predictions on new data by estimating the probability of the occurrence of the outcome based on the predictor variables. If the probability is above a certain threshold, the model predicts the positive class; otherwise, it predicts the negative class.

Model Evaluation: Finally, the performance of the logistic regression model is evaluated on the testing set using evaluation metrics such as accuracy, precision, recall, and F1 score.

Logistic Regression is a popular algorithm due to its simplicity, interpretability, and efficiency. It can handle both binary and multi-class classification tasks and can also be extended to handle ordinal or multinomial outcomes. However, it assumes a linear

relationship between the predictor variables and the log-odds of the outcome, which may not always be true in practice. In addition, it may not perform well on datasets with non-linear relationships or complex interactions between the predictor variables.

PERFORMANCE MATRICES:

There are several performance metrics that can be used to evaluate the effectiveness of the models. The choice of performance metric depends on the specific objectives of the prediction task and the nature of the data.

Some common performance metrics for binary classification tasks, such as CKD prediction, include:

1. Accuracy: The proportion of correctly classified instances out of all the instances.
2. Precision: The proportion of true positive predictions out of all positive predictions. Precision is useful when the cost of false positives is high.
3. Recall: The proportion of true positive predictions out of all actual positive instances. Recall is useful when the cost of false negatives is high.
4. F1 score: A harmonic mean of precision and recall, calculated as $2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$. F1 score is useful when precision and recall are equally important.
5. ROC curve and AUC: A receiver operating characteristic (ROC) curve is a plot of the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The area under the ROC curve (AUC) is a measure of the overall performance of the classifier, with a value of 1 indicating perfect classification and a value of 0.5 indicating random classification.

For multi-class classification tasks, additional performance metrics may be used, such as:

1. Macro/Micro F1 score: Macro F1 score calculates the F1 score for each class and then takes the average, while micro F1 score calculates the F1 score based on the total number of true positives, false positives, and false negatives across all classes.

2. Confusion matrix: A table that shows the number of true positives, true negatives, false positives, and false negatives for each class.
3. Precision-Recall curve: A curve that plots precision against recall at various threshold settings.

In summary, when evaluating the performance of CKD prediction models, it is important to consider multiple performance metrics to get a comprehensive understanding of their strengths and weaknesses.

CONFUSION MATRIX:

A confusion matrix is a table that summarizes the performance of the model on the testing data. The confusion matrix displays the actual and predicted class labels for each instance in the testing set and is used to calculate various performance metrics, such as accuracy, precision, recall, and F1 score.

A confusion matrix for a binary classification problem typically looks like this:

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

where TP, FN, FP, and TN represent the number of true positives, false negatives, false positives, and true negatives, respectively.

Based on the values in the confusion matrix, we can calculate several performance metrics:

1. Accuracy: $(TP + TN) / (TP + TN + FP + FN)$
2. Precision: $TP / (TP + FP)$
3. Recall: $TP / (TP + FN)$
4. F1 score: $2 * (precision * recall) / (precision + recall)$

A confusion matrix for a multi-class classification problem can be represented as a table with rows and columns corresponding to the actual and predicted classes, respectively. Each entry in the table represents the number of instances that were classified as belonging to a particular combination of actual and predicted classes.

In summary, a confusion matrix is a useful tool for visualizing the performance of a CKD prediction model and calculating performance metrics such as accuracy, precision, recall, and F1 score.

Actual	Negative (0)	True Negative (TN)	False Positive (FP)
	Positive (1)	False Negative (FN)	True Positive (TP)
		Negative (0)	Positive (1)
		Predicted	

True Positive: It is an outcome in which the positive class is correctly predicted by the model. When the system correctly predicts that an incident has occurred, the outcome is deemed true positive.

True Negative: It is an outcome in which the negative class is correctly predicted by the model. When the system correctly predicts that the particular incident has not occurred, the outcome is deemed true negative.

False Positive: When the model predicts the positive class incorrectly, this is called a false positive. When the system is unable to accurately predict that a particular incident has occurred, the result is deemed a False Positive.

False Negative: When the model misidentifies the negative class, the accuracy value False Negative is set. When the system is unable to accurately predict that the particular incident has not occurred, the outcome is deemed False Negative.

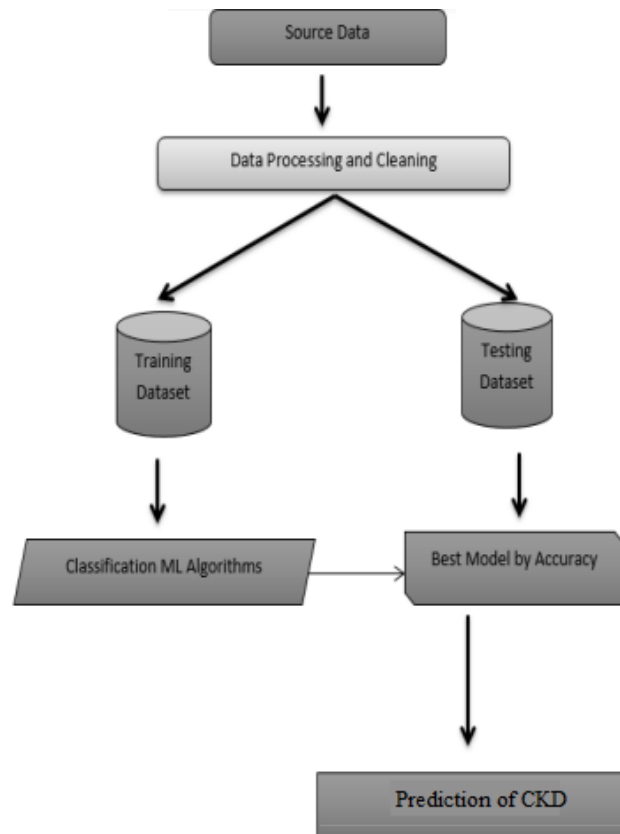
DATA FLOW DIAGRAM:

A data flow diagram (DFD) is a graphical representation of the flow of data through a system. In the context of chronic kidney disease prediction using machine learning algorithms, a DFD can be used to illustrate the various stages involved in the data processing pipeline, from data collection to model evaluation.

The DFD consists of four main stages:

1. **Data Collection:** In this stage, the relevant data is collected from various sources, such as electronic health records or medical databases. The data is pre processed to remove any missing or invalid values.
2. **Feature Engineering:** In this stage, the raw data is transformed into a set of relevant features that can be used as input to the machine learning algorithms. Feature engineering may involve techniques such as normalization, scaling, and dimensionality reduction.
3. **Model Training:** In this stage, the machine learning algorithms are trained on the pre processed data to learn the relationship between the input features and the target variable (i.e., CKD status). Several algorithms may be used, such as logistic regression and random forest.
4. **Model Evaluation:** In this stage, the performance of the trained models is evaluated on a held-out testing set using various performance metrics, such as accuracy, precision, recall, and F1 score. Based on the results, the best-performing model can be selected for deployment in a real-world setting.

In summary, a data flow diagram provides a high-level overview of the CKD prediction pipeline, from data collection to model evaluation, and helps to identify potential areas for improvement or optimization.



USE CASE DIAGRAM: .

A use case diagram is a visual representation of the interactions between actors (users or systems) and a system, highlighting the various use cases or functionalities of the system. In the context of chronic kidney disease prediction using machine learning algorithms, a use case diagram can be used to illustrate the different functionalities of the CKD prediction system.

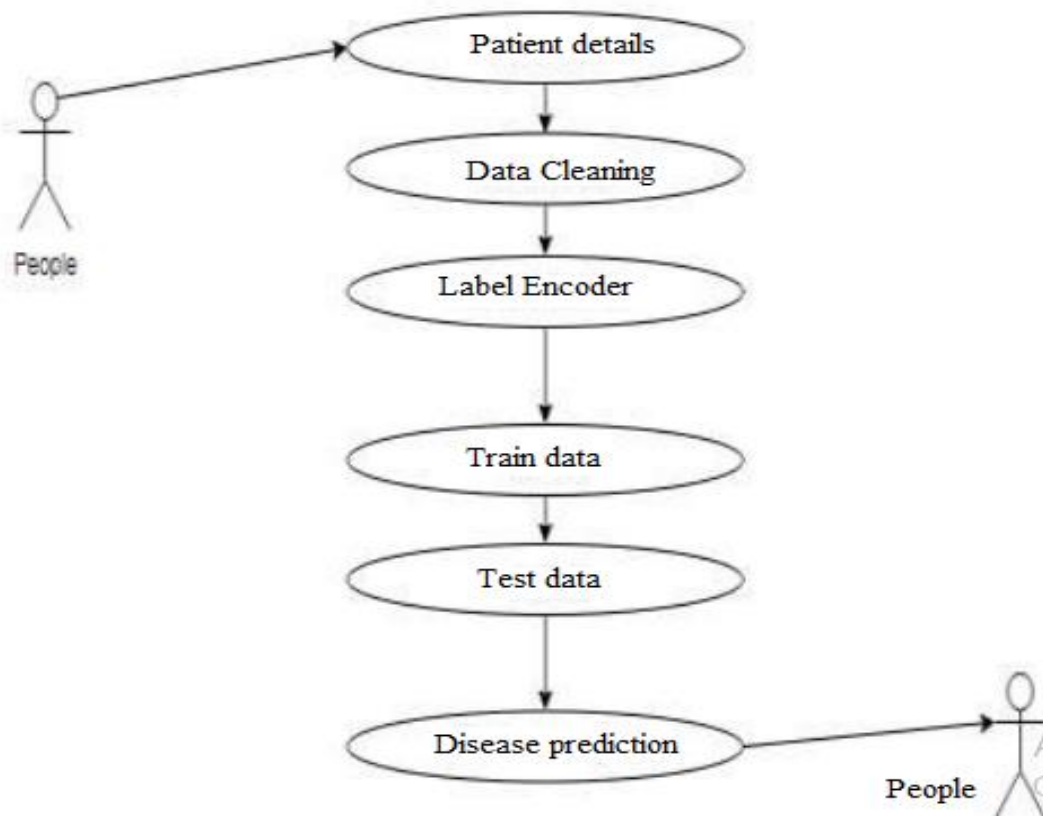
1. Patient: The patient is the user who provides the input data for the CKD prediction system.
2. Healthcare Professional: The healthcare professional is the user who may use the CKD prediction system to assist in the diagnosis and treatment of chronic kidney disease.

The use case diagram includes the following use cases:

1. Data Entry: The patient enters their personal and medical information into the system, including age, gender, blood pressure, and serum creatinine levels.
2. Prediction Request: The patient or healthcare professional requests a CKD prediction based on the input data.

3. Model Selection: The system selects the best-performing machine learning model for the CKD prediction.
4. Prediction Calculation: The system calculates the probability of CKD based on the input data and the selected machine learning model.
5. Prediction Display: The system displays the CKD prediction to the patient or healthcare professional.

In summary, a use case diagram provides a high-level overview of the different functionalities of the CKD prediction system and the interactions between the users and the system. It helps to identify the requirements and specifications for the system and can serve as a basis for further development and testing.



CLASS DIAGRAM:

In this diagram, we have the following classes:

1. Patient: This class represents the patient and includes attributes such as age, gender, blood pressure, and serum creatinine levels.

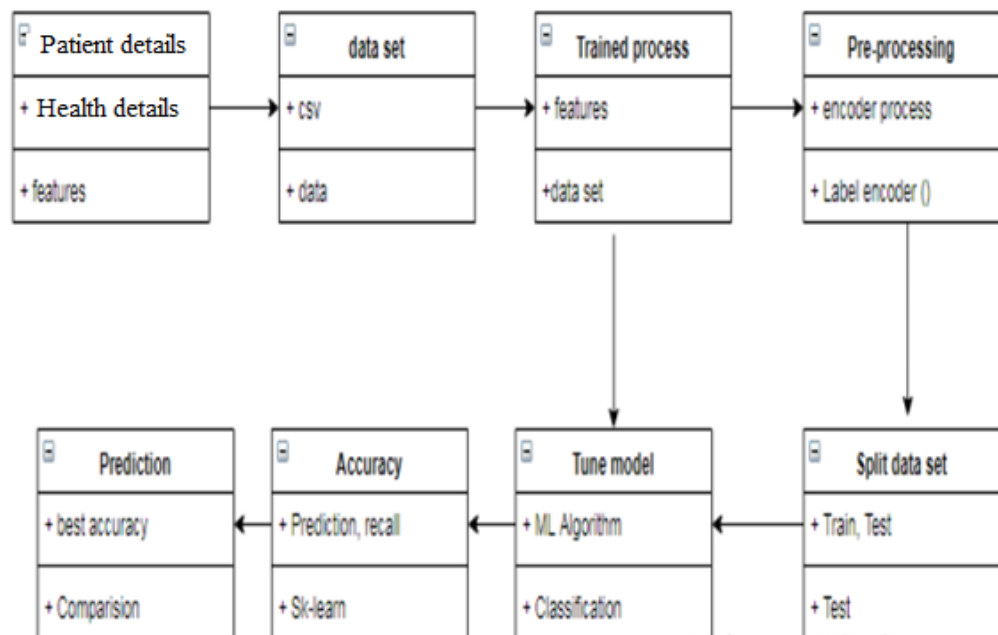
2. **CKD Prediction System:** This class represents the CKD prediction system and includes the methods for data processing, feature engineering, model training, and model evaluation.
3. **Data Pre processing:** This class represents the pre processing stage of the CKD prediction pipeline and includes the methods for data cleaning, imputation, and normalization.
4. **Feature Engineering:** This class represents the feature engineering stage of the CKD prediction pipeline and includes the methods for feature selection, scaling, and dimensionality reduction.
5. **Machine Learning Model:** This class represents the selected machine learning model, such as logistic regression or random forest, and includes the methods for model training, prediction, and evaluation.
6. **Model Evaluation:** This class represents the evaluation stage of the CKD prediction pipeline and includes the methods for calculating the performance metrics, such as accuracy, precision, recall, and F1 score.

The class diagram includes the following relationships:

1. **Patient – CKD Prediction System:** The CKD Prediction System has a dependency on the Patient class, as it requires input data from the patient to perform CKD prediction.
2. **CKD Prediction System – Data Pre processing:** The CKD Prediction System has a composition relationship with the Data Pre processing class, as it owns and controls the Data Pre processing object.
3. **CKD Prediction System – Feature Engineering:** The CKD Prediction System has a composition relationship with the Feature Engineering class, as it owns and controls the Feature Engineering object.
4. **CKD Prediction System – Machine Learning Model:** The CKD Prediction System has a composition relationship with the Machine Learning Model class, as it owns and controls the Machine Learning Model object.
5. **Machine Learning Model – Model Evaluation:** The Machine Learning Model has a composition relationship with the Model Evaluation class, as it owns and controls the Model Evaluation object.

The class diagram illustrates the structure and relationships between the different classes involved in a CKD prediction system using machine learning algorithms. It highlights the dependencies and ownership relationships between objects and the methods associated with each class. This diagram can be used to help design and implement the CKD prediction system and ensure that each object is

responsible for its respective tasks. It can also be used to guide future updates or modifications to the system.



SEQUENCE DIAGRAM:

In this diagram, we have the following objects:

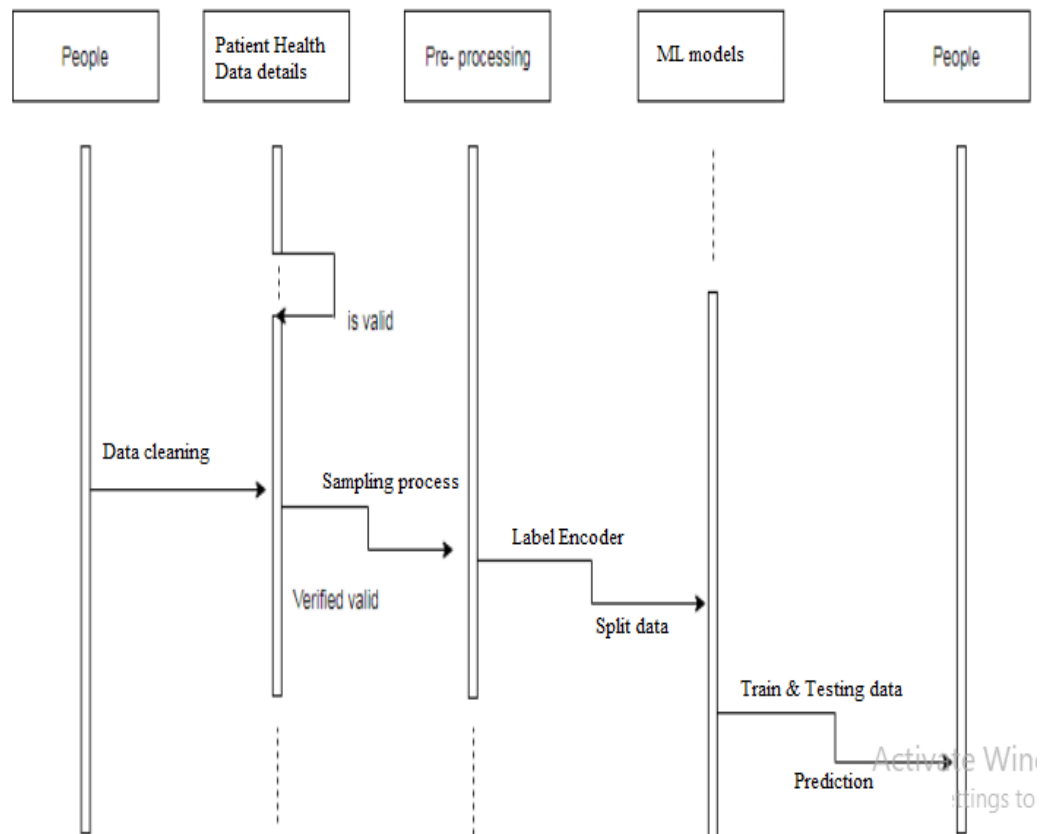
1. Patient: The patient enters their personal and medical information into the system, including age, gender, blood pressure, and serum creatinine levels.
2. CKD Prediction System: This object represents the CKD prediction system and includes the methods for data processing, feature engineering, model training, and model evaluation.
3. Data Pre processing: This object represents the pre processing stage of the CKD prediction pipeline and includes the methods for data cleaning, imputation, and normalization.
4. Feature Engineering: This object represents the feature engineering stage of the CKD prediction pipeline and includes the methods for feature selection, scaling, and dimensionality reduction.

5. Machine Learning Model: This object represents the selected machine learning model, such as logistic regression or random forest.
6. Model Evaluation: This object represents the evaluation stage of the CKD prediction pipeline and includes the methods for calculating the performance metrics, such as accuracy, precision, recall, and F1 score.

The sequence diagram includes the following messages:

1. Data Entry: The patient enters their personal and medical information into the system.
2. Data Pre processing: The CKD Prediction System sends the input data to the Data Pre processing object, which cleans, imputes, and normalizes the data.
3. Feature Engineering: The CKD Prediction System sends the pre processed data to the Feature Engineering object, which selects, scales, and reduces the features.
4. Model Training: The CKD Prediction System sends the engineered features to the Machine Learning Model object, which trains the selected machine learning model.
5. Model Evaluation: The CKD Prediction System sends the trained model to the Model Evaluation object, which evaluates the model's performance using the input data.
6. Prediction Request: The patient requests a CKD prediction based on the input data.
7. Prediction Calculation: The CKD Prediction System sends the pre processed and engineered input data to the Machine Learning Model object, which calculates the probability of CKD based on the selected machine learning model.
8. Prediction Display: The CKD Prediction System displays the CKD prediction result to the patient, including the probability of CKD and the predicted CKD status (positive or negative).

In summary, this sequence diagram illustrates the flow of information and processing steps in a CKD prediction system using machine learning algorithms. It highlights the interactions between objects and the order and timing of messages exchanged between them. This diagram can be used to help identify any potential bottlenecks or inefficiencies in the system and guide future optimization efforts.



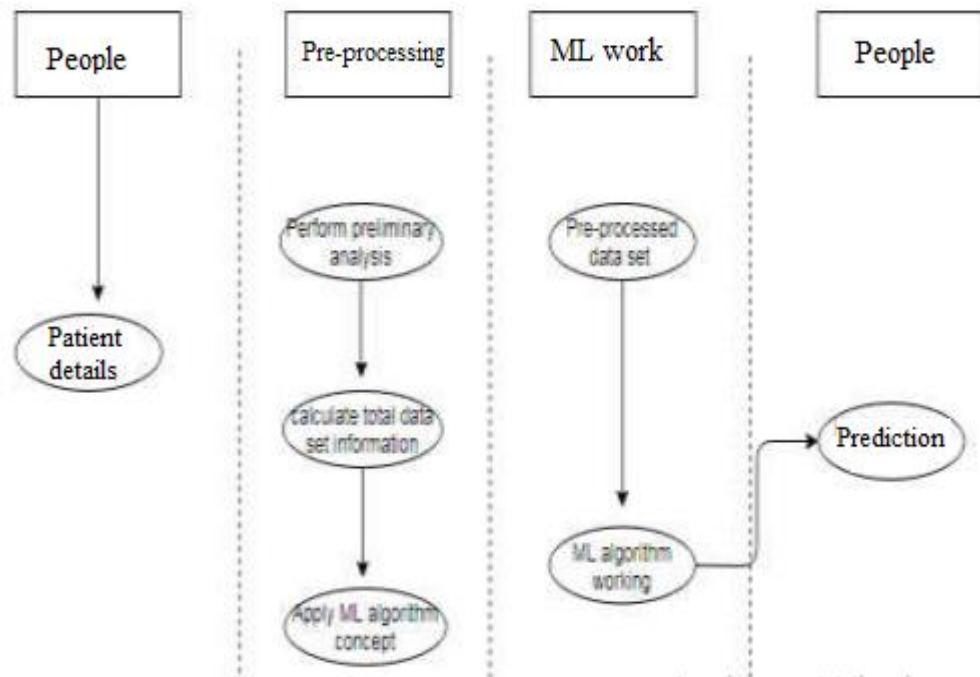
ACTIVITY DIAGRAM:

Chronic kidney disease (CKD) is a significant public health problem worldwide, with increasing prevalence and a high economic burden. Early detection and management of CKD can prevent or delay its progression to end-stage renal disease (ESRD). Machine learning algorithms can be used to develop predictive models for CKD, which can help identify individuals at high risk for developing the disease and enable timely interventions.

To develop a predictive model for CKD using machine learning algorithms, the following steps can be taken:

1. **Data collection:** Collect data on relevant variables, such as age, gender, blood pressure, serum creatinine levels, albuminuria, diabetes, and hypertension status. This data can be obtained from electronic health records or surveys.
2. **Data cleaning and preparation:** Pre-process the data to remove missing values, outliers, and irrelevant variables. Convert categorical variables to numerical values using encoding techniques, such as one-hot encoding or label encoding.
3. **Feature selection:** Identify the most important variables for predicting CKD using feature selection algorithms, such as mutual information, chi-square, or correlation-based feature selection.
4. **Model selection:** Choose a suitable machine learning algorithm for developing the predictive model, such as logistic regression, decision trees, random forests, or support vector machines (SVM).
5. **Model training and validation:** Split the data into training and testing sets, and train the selected model using the training set. Evaluate the performance of the model using metrics such as accuracy, precision, recall, and F1-score on the testing set.
6. **Model optimization:** Tune the hyperparameters of the model to improve its performance, using techniques such as grid search or random search.
7. **Model deployment:** Deploy the final model in a suitable environment, such as a web application or a mobile app, to make it accessible to end-users.

An activity diagram can be used to represent the above steps as a flowchart, showing the sequence of activities involved in developing the CKD predictive model using machine learning algorithms. The activity diagram can help visualize the process and identify potential bottlenecks or areas for improvement.



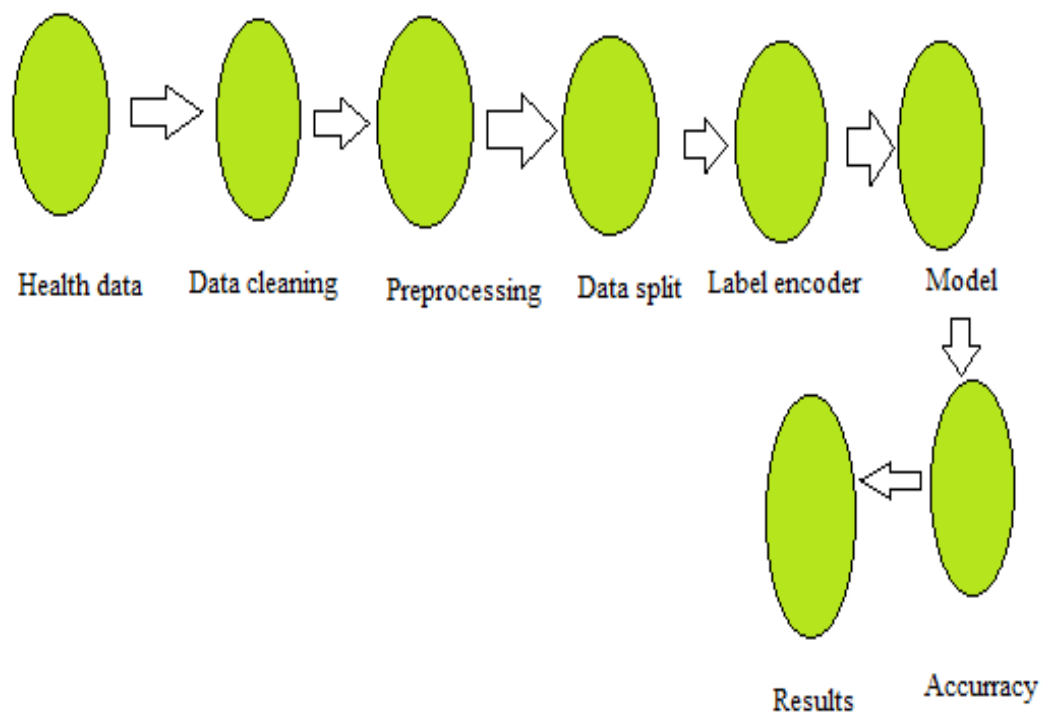
STATE FLOW DIAGRAM:

A state flow diagram can also be used to represent the process of developing a predictive model for chronic kidney disease using machine learning algorithms. The state flow diagram can help illustrate the different states of the system and the transitions between them. Here is an example of a state flow diagram for CKD prediction:

1. Data collection: The system starts in the "Data Collection" state, where data on relevant variables is collected from electronic health records or surveys.
2. Data cleaning and preparation: Once the data is collected, the system transitions to the "Data Cleaning and Preparation" state, where the data is pre processed to remove missing values, outliers, and irrelevant variables.
3. Feature selection: The system then moves to the "Feature Selection" state, where the most important variables for predicting CKD are identified using feature selection algorithms.
4. Model selection: In the "Model Selection" state, a suitable machine learning algorithm is chosen for developing the predictive model, such as logistic regression, decision trees, random forests, or SVM.

5. Model training: The system then moves to the "Model Training" state, where the selected model is trained using the training set.
6. Model validation: Once the model is trained, the system transitions to the "Model Validation" state, where the performance of the model is evaluated using metrics such as accuracy, precision, recall, and F1-score on the testing set.
7. Model optimization: The system may then move to the "Model Optimization" state, where the hyperparameters of the model are tuned to improve its performance.
8. Model deployment: Finally, the system transitions to the "Model Deployment" state, where the final model is deployed in a suitable environment, such as a web application or a mobile app, to make it accessible to end-users.

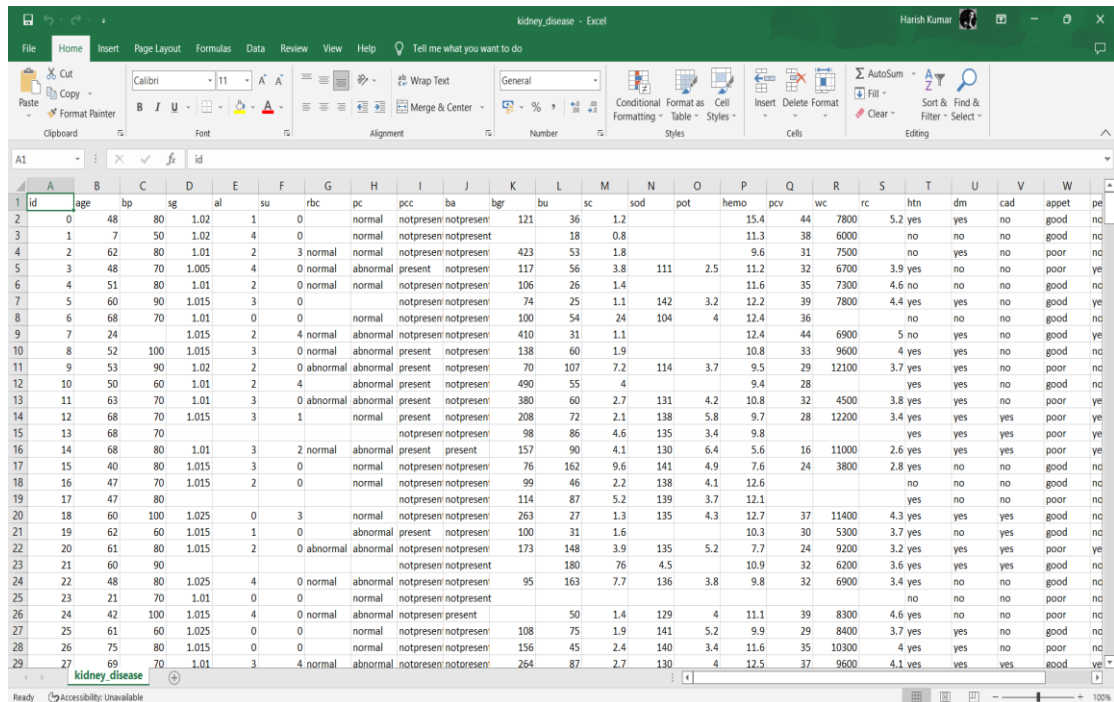
The state flow diagram can help visualize the process and identify potential bottlenecks or areas for improvement in developing a predictive model for chronic kidney disease using machine learning algorithms.



Chapter 5

IMPLEMENTATION AND TESTING

5.1 DATA SET



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
id	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	sod	pot	hemo	pcv	wc	rc	htn	dm	cad	appet	pe
0	48	80	1.02	1	0		normal	notpresen	notpresen	121	36	1.2			15.4	44	7800	5.2	yes	yes	no	good	no
1	7	50	1.02	4	0		normal	notpresen	notpresen		18	0.8			11.3	38	6000		no	no	no	good	no
2	62	80	1.01	2	3	normal	normal	notpresen	notpresen	423	53	1.8			9.6	31	7500		no	yes	no	poor	no
3	48	70	1.005	4	0	normal	abnormal	present	notpresen	117	56	3.8	111	2.5	11.2	32	6700	3.9	yes	no	no	poor	yes
4	51	80	1.01	2	0	normal	normal	notpresen	notpresen	106	26	1.4			11.6	35	7300	4.6	no	no	no	good	no
5	60	90	1.015	3	0					74	25	1.1	142	3.2	12.2	39	7800	4.4	yes	yes	no	good	yes
6	68	70	1.01	0	0		normal	notpresen	notpresen	100	54	24	104		12.4	36			no	no	no	good	no
7	24		1.015	2	4	normal	abnormal	notpresen	notpresen	410	31	1.1			12.4	44	6900	5	no	yes	no	good	yes
8	52	100	1.015	3	0	normal	abnormal	present	notpresen	138	60	1.9			10.8	33	9600	4	yes	yes	no	good	no
9	53	90	1.02	2	0	abnormal	abnormal	present	notpresen	70	107	7.2	114	3.7	9.5	29	12100	3.7	yes	yes	no	poor	no
10	50	60	1.01	2	4		abnormal	present	notpresen	490	55	4			9.4	28			yes	yes	no	good	no
11	63	70	1.01	3	0	abnormal	abnormal	present	notpresen	380	60	2.7	131	4.2	10.8	32	4500	3.8	yes	yes	no	poor	yes
12	68	70	1.015	3	1		normal	present	notpresen	208	72	2.1	138	5.8	9.7	28	12200	3.4	yes	yes	yes	poor	yes
13	68	70						notpresen	notpresen	98	86	4.6	135	3.4	9.8				yes	yes	yes	poor	yes
14	68	80	1.01	3	2	normal	abnormal	present	present	157	90	4.1	130	6.4	5.6	16	11000	2.6	yes	yes	yes	poor	yes
15	40	80	1.015	3	0		normal	notpresen	notpresen	76	162	9.6	141	4.9	7.6	24	3800	2.8	yes	no	no	good	no
16	47	70	1.015	2	0		normal	notpresen	notpresen	99	46	2.2	138	4.1	12.6				no	no	no	good	no
17	47	80						notpresen	notpresen	114	87	5.2	139	3.7	12.1				yes	no	no	poor	no
18	60	100	1.025	0	3		normal	notpresen	notpresen	263	27	1.3	135	4.3	12.7	37	11400	4.3	yes	yes	yes	good	no
19	62	60	1.015	1	0		abnormal	present	notpresen	100	31	1.6			10.3	30	5300	3.7	yes	no	yes	good	no
20	61	80	1.015	2	0	abnormal	abnormal	notpresen	notpresen	173	148	3.9	135	5.2	7.7	24	9200	3.2	yes	yes	yes	poor	yes
21	60	90						notpresen	notpresen	180	76	4.5			10.9	32	6200	3.6	yes	yes	yes	good	no
22	48	80	1.025	4	0	normal	abnormal	notpresen	notpresen	95	163	7.7	136	3.8	9.8	32	6900	3.4	yes	no	no	good	no
23	21	70	1.01	0	0		normal	notpresen	notpresen										no	no	no	poor	no
24	42	100	1.015	4	0	normal	abnormal	notpresen	present		50	1.4	129	4	11.1	39	8300	4.6	yes	no	no	poor	no
25	61	60	1.025	0	0		normal	notpresen	notpresen	108	75	1.9	141	5.2	9.9	29	8400	3.7	yes	yes	no	good	no
26	75	80	1.015	0	0		normal	notpresen	notpresen	156	45	2.4	140	3.4	11.6	35	10300	4	yes	yes	yes	poor	no
27	69	70	1.01	3	4	normal	abnormal	notpresen	notpresen	264	87	2.7	130	4	12.5	37	9600	4.1	yes	yes	yes	good	yes

ROWS:400

COLUMNS:24

We will be using csv dataset with 24 columns and 400 rows from Kaggle on Chronic Kidney Disease.

5.2 SAMPLE CODE

```
import pandas as pd
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
#list of useful imports that I will use
```

```
%matplotlib inline
```



```
import os

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import seaborn as sns

import random

from sklearn.metrics import roc_curve


from sklearn.preprocessing import LabelEncoder


from sklearn.impute import SimpleImputer

#imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

from sklearn_pandas import DataFrameMapper

import warnings

warnings.filterwarnings('ignore')

data =
pd.read_csv(r'C:\Users\yusuf\Videos\KIDNEY\DATASET\kidney_disease.csv')

data.head()

data.columns.drop('id')

data.head()

data = data.drop(columns=['id'])

data.columns

# print number of classes in our dataset
```

```

num_classes = len(np.unique(data['classification']))

num_classes

data['classification'].value_counts()

sns.countplot(x= data['classification'])

data.info()

# Total missing values for each feature

print(data.isnull().sum())

# numerical columns

num_cols = ['age', 'bp', 'sg', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wc', 'rc']

# categorical columns

cate_cols = data.columns.drop('classification').drop(num_cols)

# display categorical columns

cate_cols

# convert numerical data

data[num_cols] = data[num_cols].apply(pd.to_numeric, errors='coerce')

data.info()

# define numerical imputer

num_imputer = SimpleImputer(strategy='mean')

# imputing on numerical data

data[num_cols] = num_imputer.fit_transform(data[num_cols])

data.info()

data.head()

#imputer=CategoricalImputer(strategy='most_frequent', axis=1)

imp = SimpleImputer(strategy='most_frequent')

data[cate_cols] = imp.fit_transform(data[cate_cols])

data.info()

data['classification'].value_counts()

```

```

data['classification'] = data['classification'].replace('ckd\t', 'ckd')

data['classification'].value_counts()

sns.countplot(x= data['classification'])

cate_cols

print(data['rbc'].value_counts())

sns.countplot(x= data['rbc'])

print(data['pc'].value_counts())

sns.countplot(x= data['pc'])

print(data['pcc'].value_counts())

sns.countplot(x= data['pcc'])

print(data['ba'].value_counts())

sns.countplot(x= data['ba'])

print(data['htn'].value_counts())

sns.countplot(x= data['htn'])

k = data[cate_cols].columns

k = list(k)

k

data['rbc'].value_counts()

data['pc'].value_counts()

for i in range(len(k)):

    print(data[k[i]].value_counts())

data['dm'] = data['dm'].replace('\tno', 'no')

data['htn'] = data['htn'].replace((' \tno', '\tyes', ' yes'), ('no', 'yes', 'yes'))

for i in range(len(k)):

    print(data[k[i]].value_counts())

data['htn'].value_counts()

data['htn'].value_counts()

```

```

data.columns.drop('classification')

data.shape

data['classification'].value_counts()

from sklearn.utils import resample

# Separate majority and minority classes
df_majority = data[data.classification=='ckd']
df_minority = data[data.classification=='notckd']

# Downsample majority class
df_minority_upsampled = resample(df_minority,
                                 replace=True, # sample without replacement
                                 n_samples=250, # to match minority class
                                 random_state=123) # reproducible results

# Combine minority class with downsampled majority class
df_upsampled = pd.concat([df_minority_upsampled, df_majority])

# Display new class counts
df_upsampled.classification.value_counts()

# shuffle the DataFrame rows
data = df_upsampled.sample(frac = 1)

sns.countplot(x= data.classification)

data.head(10)

y = data['classification']

# convert categorical target to numerical
y = y.apply(lambda x: 1 if x=='ckd' else 0)

# show the head of df['class']

```

```

y.head()

x = data.drop(columns=['classification'])

x.shape

x.head(5)

from sklearn.model_selection import GridSearchCV, train_test_split, cross_val_score

X_train, X_test, y_train, y_test = train_test_split(x,y,stratify=y,random_state = 42)

print(X_train.shape)

print(X_test.shape)

X_train.head()

le = LabelEncoder()

le.fit(data['rbc'])

X_train.rbc = le.transform(X_train.rbc)

X_test.rbc = le.transform(X_test.rbc)

le.fit(data['pc'])

X_train.pc = le.transform(X_train.pc)

X_test.pc = le.transform(X_test.pc) le.fit(data['pcc'])

X_train.pcc = le.transform(X_train.pcc)

X_test.pcc = le.transform(X_test.pcc)

le.fit(data['ba'])

X_train.ba = le.transform(X_train.ba)

X_test.ba = le.transform(X_test.ba)

le.fit(data['htn'])

X_train.htn = le.transform(X_train.htn)

X_test.htn = le.transform(X_test.htn)

le.fit(data['dm'])

X_train.dm = le.transform(X_train.dm)

X_test.dm = le.transform(X_test.dm)

```

```

le.fit(data['cad'])

X_train.cad = le.transform(X_train.cad)
X_test.cad = le.transform(X_test.cad)

le.fit(data['appet'])

X_train.appet = le.transform(X_train.appet)
X_test.appet = le.transform(X_test.appet)

le.fit(data['pe'])

X_train.pe = le.transform(X_train.pe)
X_test.pe = le.transform(X_test.pe)

le.fit(data['ane'])

X_train.ane = le.transform(X_train.ane)
X_test.ane = le.transform(X_test.ane)

X_train.head()

X_test.head()

X_test.info()

from sklearn.preprocessing import StandardScaler

ss = StandardScaler()

X_train = ss.fit_transform(X_train)
X_test = ss.fit_transform(X_test)

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import roc_auc_score

import math

c = [10000,1000,100,10,1,0.1,0.01,0.001,0.0001,0.00001]

train_auc = []

test_auc = []

```

```

for i in c:

    clf = LogisticRegression(penalty='l2',C=i)

    clf.fit(X_train,y_train)

    prob_test = clf.predict_proba(X_test)[:,1]

    test_auc.append(roc_auc_score(y_test,prob_test))

    prob_train = clf.predict_proba(X_train)[:,1]

    train_auc.append(roc_auc_score(y_train,prob_train))

optimal_c= c[test_auc.index(max(test_auc))]

c = [math.log(x) for x in c]


#plot auc vs alpha
x = plt.subplot( )

x.plot(c, train_auc, label='AUC train')

x.plot(c, test_auc, label='AUC TEST')

plt.title('AUC vs hyperparameter')

plt.xlabel('c')

plt.ylabel('AUC')

x.legend()

plt.show()


print('optimal c for which auc is maximum : ',optimal_c)

#Testing AUC on Test data

log = LogisticRegression(penalty='l2',C=optimal_c)

log.fit(X_train,y_train)

pred_test = log.predict_proba(X_test)[:,1]

fpr1, tpr1, thresholds1 = roc_curve(y_test, pred_test)

```

```

pred_train = log.predict_proba(X_train)[: ,1]

fpr2,tpr2,thresholds2 = roc_curve(y_train,pred_train)


#plot ROC curve
x = plt.subplot( )

x.plot(fpr1, tpr1, label='Test ROC ,auc='+str(roc_auc_score(y_test,pred_test)))
x.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_train)))

plt.title('ROC curve')

plt.xlabel('FPR')

plt.ylabel('TPR')

x.legend()

plt.show()


print("AUC on Test data is " +str(roc_auc_score(y_test,pred_test)))

print("AUC on Train data is " +str(roc_auc_score(y_train,pred_train)))


print("-----")


# Code for drawing seaborn heatmaps

from sklearn.metrics import confusion_matrix

class_names = ['CKD','NOT CKD']

df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_test.round()),
index=class_names, columns=class_names )

fig = plt.figure( )

heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

results=pd.DataFrame(columns=['model','classifier','hyper parameter','Train-
AUC','Test-AUC'])

```



```

new = ['Logistic Regression','LogisticRegression',"c = 1000",1.0,1.0]
results.loc[0] = new

predicted = log.predict_proba(X_test[:20])[:,1]
predicted[predicted>0.5] = 1
predicted[predicted<0.5] = 0

original = ["CKD" if x==1 else "No_CKD" for x in y_test[:20]]

pred = []

for i in predicted:
    if i == 1:
        k = "CKD"
        pred.append(k)
    else:
        k = "No_CKD"
        pred.append(k)

# Creating a data frame
df = pd.DataFrame(list(zip(original, pred)),
                    columns=['original_Classlabel', 'predicted_classlebel'])

df

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV

dept = [1, 5, 10, 50, 100, 500, 1000]
n_estimators = [20, 40, 60, 80, 100, 120]

param_grid={'n_estimators':n_estimators , 'max_depth':dept}
clf = RandomForestClassifier()

```

```

model = GridSearchCV(clf,param_grid,scoring='roc_auc',n_jobs=-1,cv=3)
model.fit(X_train,y_train)
print("optimal n_estimators",model.best_estimator_.n_estimators)
print("optimal max_depth",model.best_estimator_.max_depth)

import seaborn as sns

X = []
Y = []
test_auc = []
train_auc = []

for n in n_estimators:
    for d in dept:
        clf = RandomForestClassifier(max_depth = d,n_estimators = n)
        clf.fit(X_train,y_train)
        pred_cv = clf.predict_proba(X_test)[: ,1]
        pred_train = clf.predict_proba(X_train)[: ,1]
        X.append(n)
        Y.append(d)
        test_auc.append(roc_auc_score(y_test,pred_cv))
        train_auc.append(roc_auc_score(y_train,pred_train))

#Heatmap for cross validation data
data = pd.DataFrame({'n_estimators': X, 'max_depth': Y, 'AUC': test_auc})
data_pivoted = data.pivot("n_estimators", "max_depth", "AUC")
ax = sns.heatmap(data_pivoted,annot=True)
plt.title('Heatmap for cross validation data')
plt.show()

```

```

#Heatmap for training data

data = pd.DataFrame({'n_estimators': X, 'max_depth': Y, 'AUC': train_auc})

data_pivoted = data.pivot("n_estimators", "max_depth", "AUC")

ax = sns.heatmap(data_pivoted,annot=True)

plt.title('Heatmap for training data')

plt.show()

optimal_n_estimators = model.best_estimator_.n_estimators

optimal_max_depth = model.best_estimator_.max_depth

#training our model for max_depth=10,n_estimators = 120

rf = RandomForestClassifier(max_depth = optimal_max_depth,n_estimators =
optimal_n_estimators)

rf.fit(X_train,y_train)

pred_test =rf.predict_proba(X_test)[: ,1]

fpr1, tpr1, thresholds1 = roc_curve(y_test, pred_test)

pred_train = rf.predict_proba(X_train)[: ,1]

fpr2,tpr2,thresholds2=roc_curve(y_train,pred_train)


#ROC curve

x = plt.subplot( )

x.plot(fpr1, tpr1, label='Test ROC ,auc='+str(roc_auc_score(y_test,pred_test)))

x.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_train)))

plt.title('ROC')

plt.xlabel('FPR')

plt.ylabel('TPR')

x.legend()

plt.show()

```

```

print("AUC on Test data is " +str(roc_auc_score(y_test,pred_test)))

print("AUC on Train data is " +str(roc_auc_score(y_train,pred_train)))


print("-----")


# Code for drawing seaborn heatmaps

class_names = ['ckd','notckd']

df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_test.round()),
index=class_names, columns=class_names )

fig = plt.figure( )

heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

new = ['Random Forest','RandomForestClassifier',"n_estimators= 80 ,max_depth=
100 ",1.0,1.0]

results.loc[1] = new

predicted = rf.predict_proba(X_test[:20])[:,1]

predicted[predicted>0.5] = 1

predicted[predicted<0.5] = 0

original = ["CKD" if x==1 else "No_CKD" for x in y_test[:20]]

pred = []

for i in predicted:

    if i == 1:

        k = "CKD"

        pred.append(k)

    else:

        k = "No_CKD"

        pred.append(k)

# Creating a data frame

df = pd.DataFrame(list(zip(original, pred)),

```

```
columns=['original_Classlabel', 'predicted_classlebel'])
```

results

5.3 SAMPLE OUTPUT

DATASET:

```
In [4]: data.head()
```

```
Out[4]:
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd

5 rows × 26 columns

DROPPING THE COLUMN ID:

```
In [5]: data.columns.drop('id')
```

```
Out[5]: Index(['age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba', 'bgr', 'bu',  
              'sc', 'sod', 'pot', 'hemo', 'pcv', 'wc', 'rc', 'htn', 'dm', 'cad',  
              'appet', 'pe', 'ane', 'classification'],  
              dtype='object')
```

DISPLAYING THE HEAD:

```
In [6]: data.head()
```

```
Out[6]:
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd

5 rows × 26 columns

NO OF CLASSES:

```
In [9]: # print number of classes in our dataset  
num_classes = len(np.unique(data['classification']))
```

```
In [10]: num_classes
```

```
Out[10]: 3
```

CLASSIFICATION:

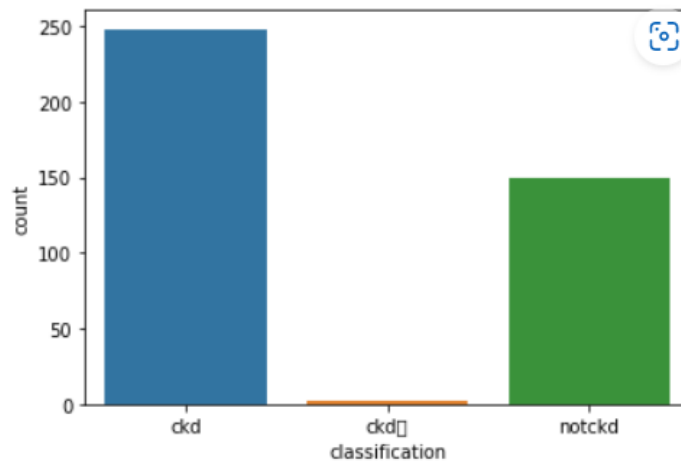
```
In [11]: data['classification'].value_counts()
```

```
Out[11]: ckd      248  
         notckd   150  
         ckd\t      2  
         Name: classification, dtype: int64
```

PLOT TABLE:

```
In [12]: sns.countplot(x= data['classification'])
```

```
Out[12]: <AxesSubplot:xlabel='classification', ylabel='count'>
```



DATA INFO:

```
In [13]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 400 entries, 0 to 399  
Data columns (total 25 columns):  
#   Column                Non-Null Count  Dtype    
---  ---                  
0   age                   391 non-null   float64  
1   bp                    388 non-null   float64  
2   sg                    353 non-null   float64  
3   al                    354 non-null   float64  
4   su                    351 non-null   float64  
5   rbc                   248 non-null   object  
6   pc                    335 non-null   object  
7   pcc                   396 non-null   object  
8   ba                    396 non-null   object  
9   bgr                   356 non-null   float64  
10  bu                    381 non-null   float64  
11  sc                    383 non-null   float64  
12  sod                   313 non-null   float64  
13  pot                   312 non-null   float64  
14  hemo                  348 non-null   float64  
15  pcv                   330 non-null   object  
16  wc                    295 non-null   object  
17  rc                    270 non-null   object  
18  htn                   398 non-null   object  
19  dm                    398 non-null   object  
20  cad                   398 non-null   object  
21  appet                399 non-null   object  
22  pe                   399 non-null   object  
23  ane                  399 non-null   object  
24  classification        400 non-null   object  
dtypes: float64(11), object(14)  
memory usage: 78.2+ KB
```

IMPUTING:

Imputing

```
In [19]: # define numerical imputer
num_imputer = SimpleImputer(strategy='mean')
```

```
In [20]: # imputing on numerical data
data[num_cols] = num_imputer.fit_transform(data[num_cols])
```

AFTER IMPUTING:

```
In [24]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 25 columns):
#   Column                Non-Null Count  Dtype
---  -
0   age                   400 non-null   float64
1   bp                    400 non-null   float64
2   sg                    400 non-null   float64
3   al                    400 non-null   float64
4   su                    400 non-null   float64
5   rbc                   400 non-null   object
6   pc                    400 non-null   object
7   pcc                   400 non-null   object
8   ba                    400 non-null   object
9   bgr                   400 non-null   float64
10  bu                    400 non-null   float64
11  sc                    400 non-null   float64
12  sod                   400 non-null   float64
13  pot                   400 non-null   float64
14  hemo                  400 non-null   float64
15  pcv                   400 non-null   float64
16  wc                    400 non-null   float64
17  rc                    400 non-null   float64
18  htn                   400 non-null   object
19  dm                    400 non-null   object
20  cad                   400 non-null   object
21  appet                 400 non-null   object
22  pe                    400 non-null   object
23  ane                   400 non-null   object
24  classification        400 non-null   object
dtypes: float64(14), object(11)
memory usage: 78.2+ KB
```


DISPLAYING DATA COUNT AND REPLACING:

```
In [25]: data['classification'].value_counts()
```

```
Out[25]: ckd      248  
         notckd   150  
         ckd\t     2  
         Name: classification, dtype: int64
```

```
In [26]: data['classification'] = data['classification'].replace('ckd\t', 'ckd')
```

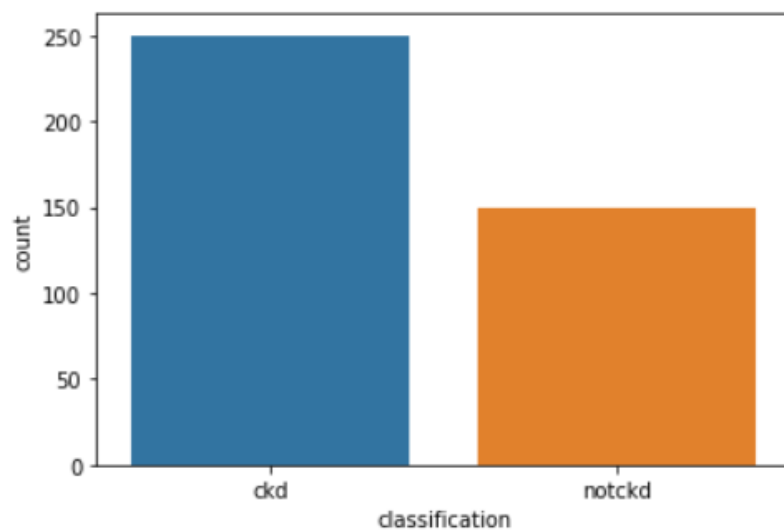
```
In [27]: data['classification'].value_counts()
```

```
Out[27]: ckd      250  
         notckd   150  
         Name: classification, dtype: int64
```

PLOT TABLE:

```
In [28]: sns.countplot(x= data['classification'])
```

```
Out[28]: <AxesSubplot:xlabel='classification', ylabel='count'>
```

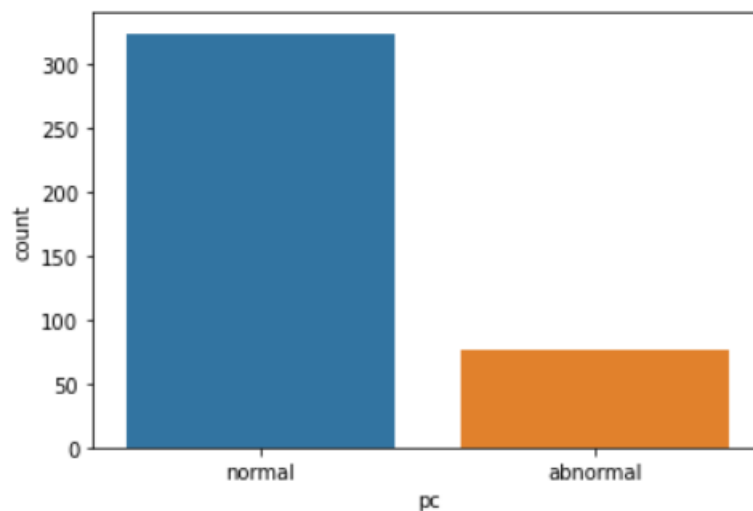


COUNT OF EACH ATTRIBUTES AND PLOT TABLE:

```
In [31]: print(data['pc'].value_counts())  
sns.countplot(x= data['pc'])
```

```
normal    324  
abnormal   76  
Name: pc, dtype: int64
```

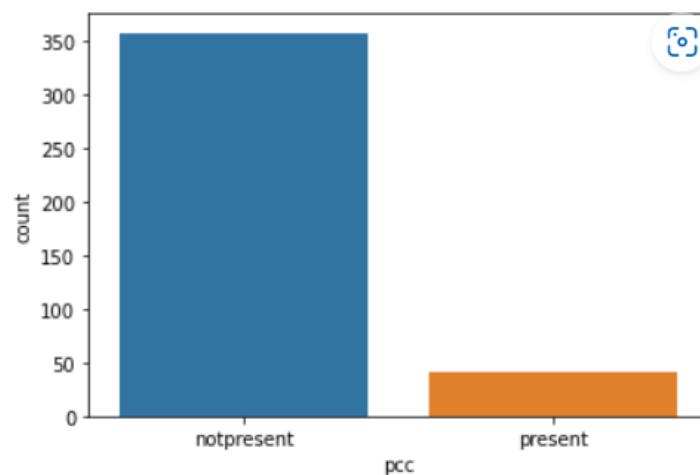
```
Out[31]: <AxesSubplot:xlabel='pc', ylabel='count'>
```



```
In [32]: print(data['pcc'].value_counts())  
sns.countplot(x= data['pcc'])
```

```
notpresent  358  
present     42  
Name: pcc, dtype: int64
```

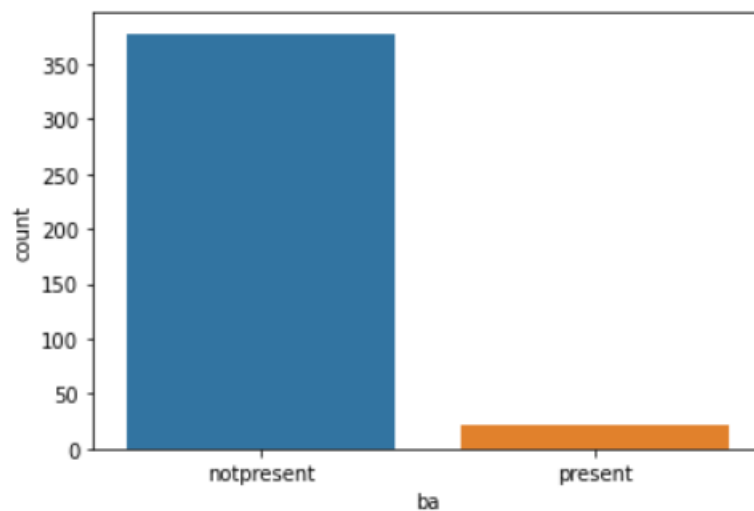
```
Out[32]: <AxesSubplot:xlabel='pcc', ylabel='count'>
```



```
In [33]: print(data['ba'].value_counts())  
sns.countplot(x= data['ba'])
```

```
notpresent    378  
present        22  
Name: ba, dtype: int64
```

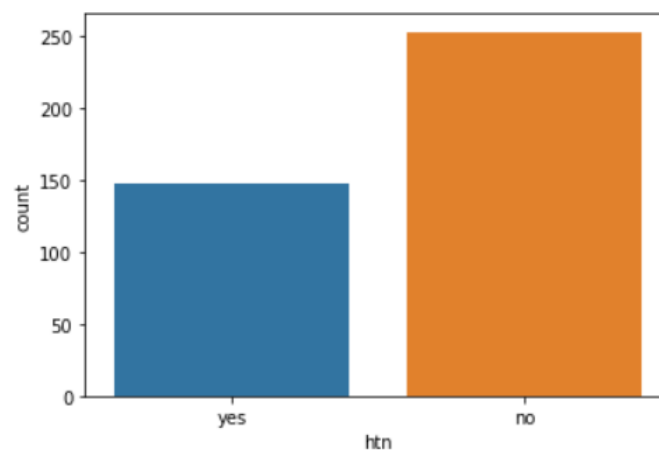
```
Out[33]: <AxesSubplot:xlabel='ba', ylabel='count'>
```



```
In [34]: print(data['htn'].value_counts())  
sns.countplot(x= data['htn'])
```

```
no    253  
yes   147  
Name: htn, dtype: int64
```

```
Out[34]: <AxesSubplot:xlabel='htn', ylabel='count'>
```



RESAMPLING:

```
In [49]: from sklearn.utils import resample
# Separate majority and minority classes
df_majority = data[data.classification=='ckd']
df_minority = data[data.classification=='notckd']

# Downsample majority class
df_minority_upsampled = resample(df_minority,
                                replace=True,      # sample without replacement
                                n_samples=250,     # to match minority class
                                random_state=123)  # reproducible results

# Combine minority class with downsampled majority class
df_upsampled = pd.concat([df_minority_upsampled, df_majority])

# Display new class counts
df_upsampled.classification.value_counts()
```

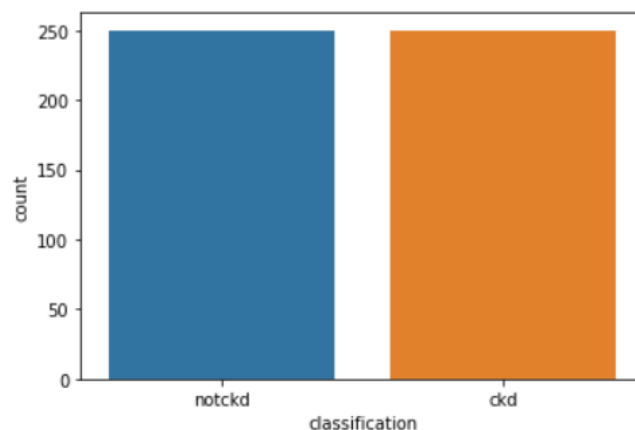
```
Out[49]: notckd    250
         ckd       250
         Name: classification, dtype: int64
```

AFTER SHUFFLING THE DATA FRAMES ROWS:

```
In [50]: # shuffle the DataFrame rows
data = df_upsampled.sample(frac = 1)
```

```
In [51]: sns.countplot(x= data.classification)
```

```
Out[51]: <AxesSubplot:xlabel='classification', ylabel='count'>
```



LABEL ENCODER:

LabelEncoder

```
In [61]: le = LabelEncoder()
le.fit(data['rbc'])
X_train.rbc = le.transform(X_train.rbc)
X_test.rbc = le.transform(X_test.rbc)
```

```
In [62]: le.fit(data['pc'])
X_train.pc = le.transform(X_train.pc)
X_test.pc = le.transform(X_test.pc)
```

```
In [63]: le.fit(data['pcc'])
X_train.pcc = le.transform(X_train.pcc)
X_test.pcc = le.transform(X_test.pcc)
```

```
In [64]: le.fit(data['ba'])
X_train.ba = le.transform(X_train.ba)
X_test.ba = le.transform(X_test.ba)
```

```
In [65]: le.fit(data['htn'])
X_train.htn = le.transform(X_train.htn)
X_test.htn = le.transform(X_test.htn)
```

```
In [66]: le.fit(data['dm'])
X_train.dm = le.transform(X_train.dm)
X_test.dm = le.transform(X_test.dm)
```

```
In [67]: le.fit(data['cad'])
X_train.cad = le.transform(X_train.cad)
X_test.cad = le.transform(X_test.cad)
```

OUTPUT AFTER LABEL ENCODING:

```
In [71]: X_train.head()
```

Out[71]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	hemo	pcv	wc	rc	htn	dm	cad	appet	pe	ane
18	60.0	100.0	1.025	0.0	3.000000	1	1	0	0	263.000000	...	12.7	37.000000	11400.000000	4.300000	1	3	2	0	0	0
85	70.0	70.0	1.015	2.0	0.450142	1	1	0	0	148.036517	...	9.9	38.884498	8406.122449	4.707435	0	3	1	1	1	0
0	48.0	80.0	1.020	1.0	0.000000	1	1	0	0	121.000000	...	15.4	44.000000	7800.000000	5.200000	1	3	1	0	0	0
46	48.0	70.0	1.015	0.0	0.000000	1	1	0	0	124.000000	...	12.4	37.000000	6400.000000	4.700000	0	3	1	0	0	0
308	43.0	80.0	1.025	0.0	0.000000	1	1	0	0	81.000000	...	13.9	48.000000	6900.000000	4.900000	0	2	1	0	0	0

5 rows × 24 columns

LOGISTIC REGRESSION:

```
In [75]: from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import math

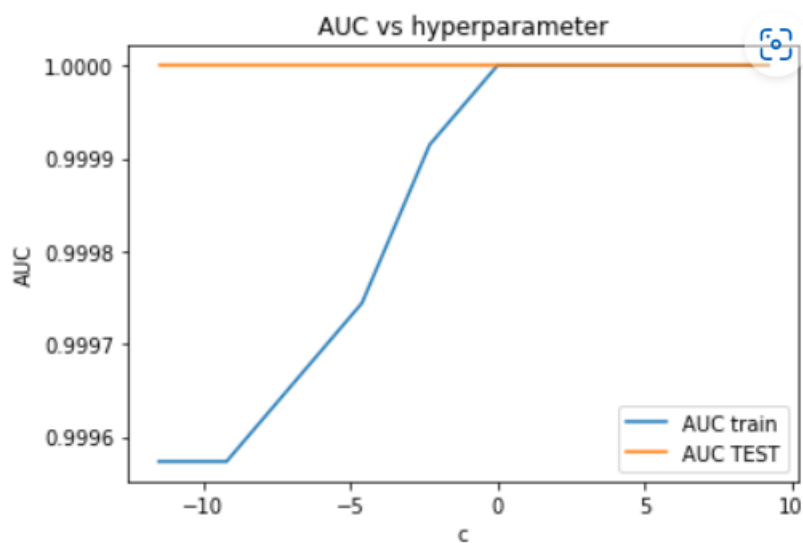
c = [10000,1000,100,10,1,0.1,0.01,0.001,0.0001,0.00001]

train_auc = []
test_auc = []

for i in c:
    clf = LogisticRegression(penalty='l2',C=i)
    clf.fit(X_train,y_train)
    prob_test = clf.predict_proba(X_test)[:,-1]
    test_auc.append(roc_auc_score(y_test,prob_test))
    prob_train = clf.predict_proba(X_train)[:,-1]
    train_auc.append(roc_auc_score(y_train,prob_train))
optimal_c = c[test_auc.index(max(test_auc))]
c = [math.log(x) for x in c]

#plot auc vs alpha
x = plt.subplot( )
x.plot(c, train_auc, label='AUC train')
x.plot(c, test_auc, label='AUC TEST')
plt.title('AUC vs hyperparameter')
plt.xlabel('c')
plt.ylabel('AUC')
x.legend()
plt.show()

print('optimal c for which auc is maximum : ',optimal_c)
```



optimal c for which auc is maximum : 10000

TESTING AUC:

```
In [76]: #Testing AUC on Test data
log = LogisticRegression(penalty='l2',C=optimal_c)
log.fit(X_train,y_train)
pred_test = log.predict_proba(X_test)[:,-1]
fpr1, tpr1, thresholds1 = roc_curve(y_test, pred_test)
pred_train = log.predict_proba(X_train)[:,-1]
fpr2,tpr2,thresholds2 = roc_curve(y_train,pred_train)

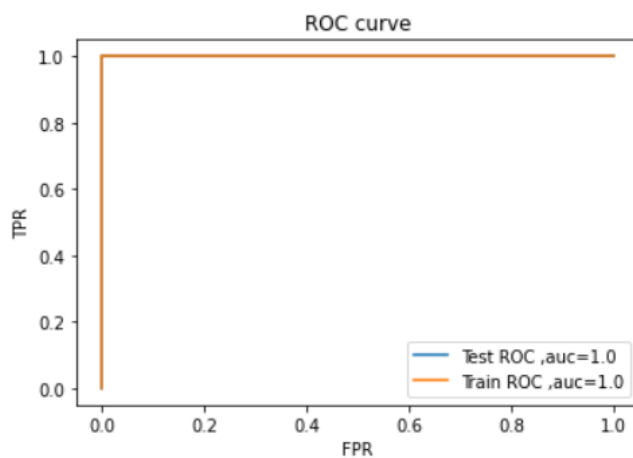
#plot ROC curve
x = plt.subplot( )
x.plot(fpr1, tpr1, label='Test ROC ,auc='+str(roc_auc_score(y_test,pred_test)))
x.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_train)))
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
x.legend()
plt.show()

print("AUC on Test data is " +str(roc_auc_score(y_test,pred_test)))
print("AUC on Train data is " +str(roc_auc_score(y_train,pred_train)))

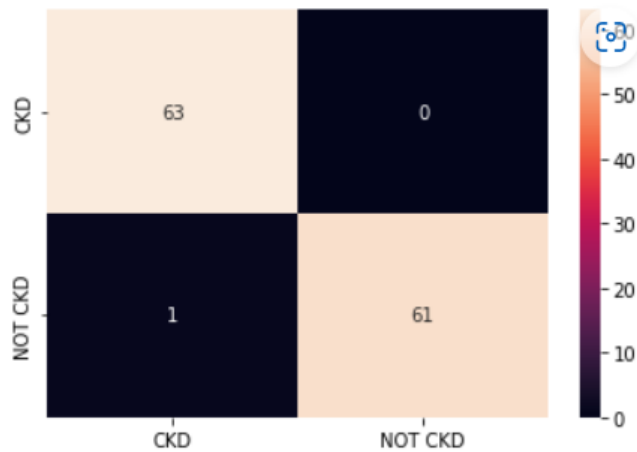
print("-----")

# Code for drawing seaborn heatmaps
from sklearn.metrics import confusion_matrix
class_names = ['CKD','NOT CKD']
df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_test.round()), index=class_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```

ROC CURVE:



AUC on Test data is 1.0
AUC on Train data is 1.0



RANDOM FOREST:

Random Forest

```
[n [79]: from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV

dept = [1, 5, 10, 50, 100, 500, 1000]
n_estimators = [20, 40, 60, 80, 100, 120]

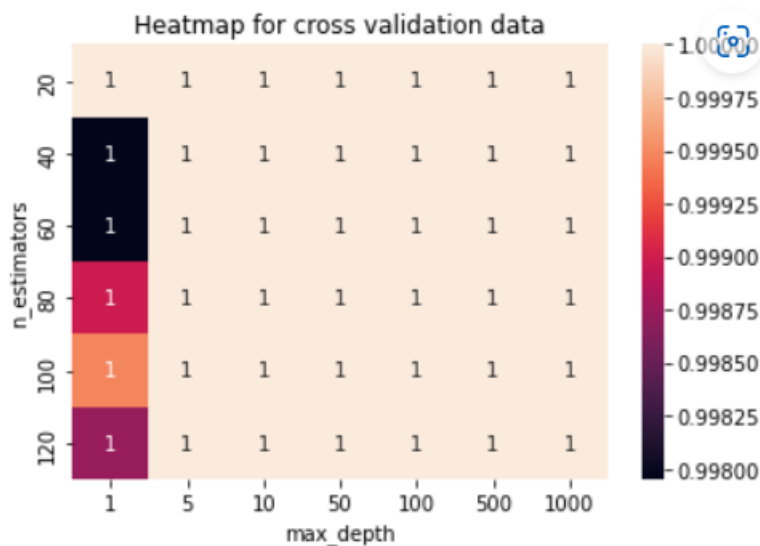
param_grid={'n_estimators':n_estimators , 'max_depth':dept}
clf = RandomForestClassifier()
model = GridSearchCV(clf,param_grid,scoring='roc_auc',n_jobs=-1,cv=3)
model.fit(X_train,y_train)
print("optimal n_estimators",model.best_estimator_.n_estimators)
print("optimal max_depth",model.best_estimator_.max_depth)

optimal n_estimators 60
optimal max_depth 10
```


HEATMAP FOR CROSS VALIDATION DATA:

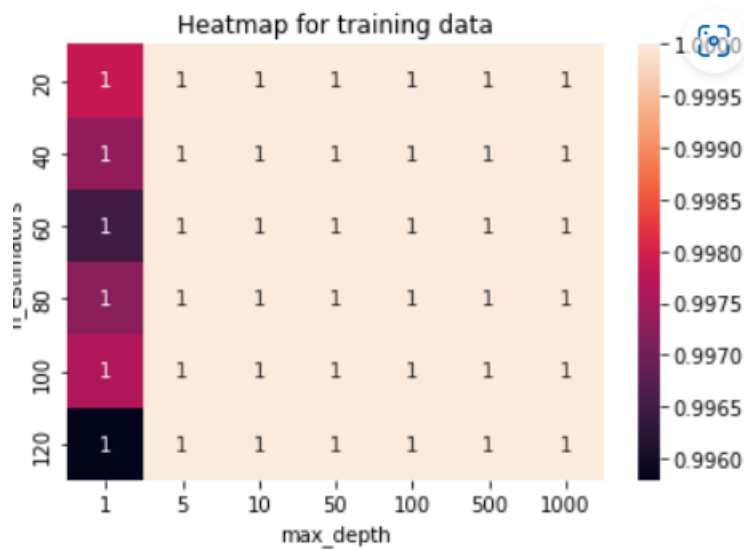
```
In [80]: import seaborn as sns
X = []
Y = []
test_auc = []
train_auc = []
for n in n_estimators:
    for d in dept:
        clf = RandomForestClassifier(max_depth = d,n_estimators = n)
        clf.fit(X_train,y_train)
        pred_cv = clf.predict_proba(X_test)[: ,1]
        pred_train = clf.predict_proba(X_train)[: ,1]
        X.append(n)
        Y.append(d)
        test_auc.append(roc_auc_score(y_test,pred_cv))
        train_auc.append(roc_auc_score(y_train,pred_train))

#Heatmap for cross validation data
data = pd.DataFrame({'n_estimators': X, 'max_depth': Y, 'AUC': test_auc})
data_pivoted = data.pivot("n_estimators", "max_depth", "AUC")
ax = sns.heatmap(data_pivoted,annot=True)
plt.title('Heatmap for cross validation data')
plt.show()
```



HEATMAP FOR TRAINING DATA:

```
#Heatmap for training data
data = pd.DataFrame({'n_estimators': X, 'max_depth': Y, 'AUC': train_auc})
data_pivoted = data.pivot("n_estimators", "max_depth", "AUC")
ax = sns.heatmap(data_pivoted,annot=True)
plt.title('Heatmap for training data')
plt.show()
```



RESULT:

```
In [83]: new = ['Random Forest','RandomForestClassifier',"n_estimators= 80 ,max_depth= 100 ",1.
results.loc[1] = new
```

```
In [84]: predicted = rf.predict_proba(X_test[:20])[:,1]
predicted[predicted>0.5] = 1
predicted[predicted<0.5] = 0
original = ["CKD" if x==1 else "No_CKD" for x in y_test[:20]]
pred = []
for i in predicted:
    if i == 1:
        k = "CKD"
        pred.append(k)
    else:
        k = "No_CKD"
        pred.append(k)
# Creating a data frame
df = pd.DataFrame(list(zip(original, pred)),
                  columns=['original_classlabel', 'predicted_classlabel'])
df
```

FINAL OUTPUT:

Results

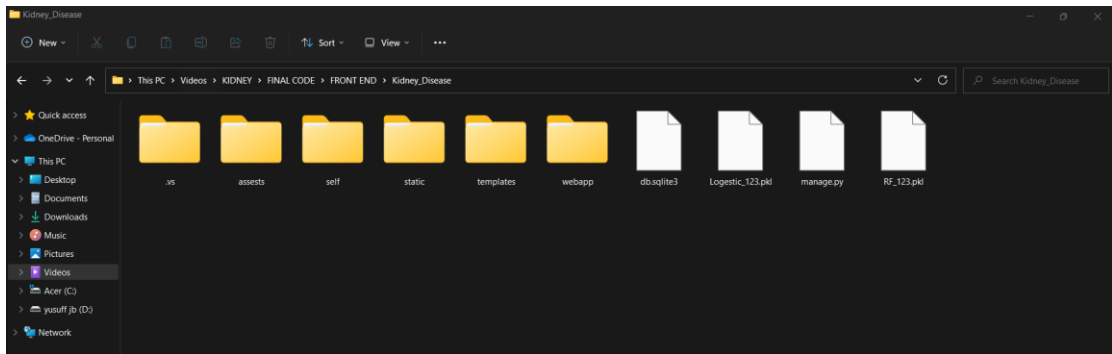
```
In [85]: results
```

```
Out[85]:
```

	model	classifier	hyper parameter	Train-AUC	Test-AUC
0	Logistic Regression	LogisticRegression	c = 1000	1.0	1.0
1	Random Forest	RandomForestClassifier	n_estimators= 80 ,max_depth= 100	1.0	1.0

FRONT END:

FILE FOLDER:



TO GET LINK:

```
Anaconda Prompt (anaconda3) - python manage.py runserver
assifier from version 0.23.2 when using version 0.24.2. This might lead to breaking code or invalid results. Use at your
own risk.
  warnings.warn(
C:\Users\yusuf\anaconda3\lib\site-packages\sklearn\base.py:310: UserWarning: Trying to unpickle estimator RandomForestCl
assifier from version 0.23.2 when using version 0.24.2. This might lead to breaking code or invalid results. Use at your
own risk.
  warnings.warn(
C:\Users\yusuf\anaconda3\lib\site-packages\sklearn\base.py:310: UserWarning: Trying to unpickle estimator GridSearchCV f
rom version 0.23.2 when using version 0.24.2. This might lead to breaking code or invalid results. Use at your own risk.
  warnings.warn(
C:\Users\yusuf\anaconda3\lib\site-packages\sklearn\base.py:310: UserWarning: Trying to unpickle estimator LogisticRegres
sion from version 0.23.2 when using version 0.24.2. This might lead to breaking code or invalid results. Use at your own
risk.
  warnings.warn(
System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin,
auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
November 10, 2022 - 20:10:28
Django version 4.1.3, using settings 'self.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[10/Nov/2022 20:10:40] "GET / HTTP/1.1" 200 5593
[10/Nov/2022 20:10:40] "GET /static/mages/icons/favicon.ico HTTP/1.1" 404 1828
[10/Nov/2022 20:10:56] "GET /input?email=venkat%40gmail.com&pass=12345 HTTP/1.1" 200 7895
[10/Nov/2022 20:11:18] "GET /input?email=yusuffsheriff420%40gmail.com&pass=12345 HTTP/1.1" 200 7895
```

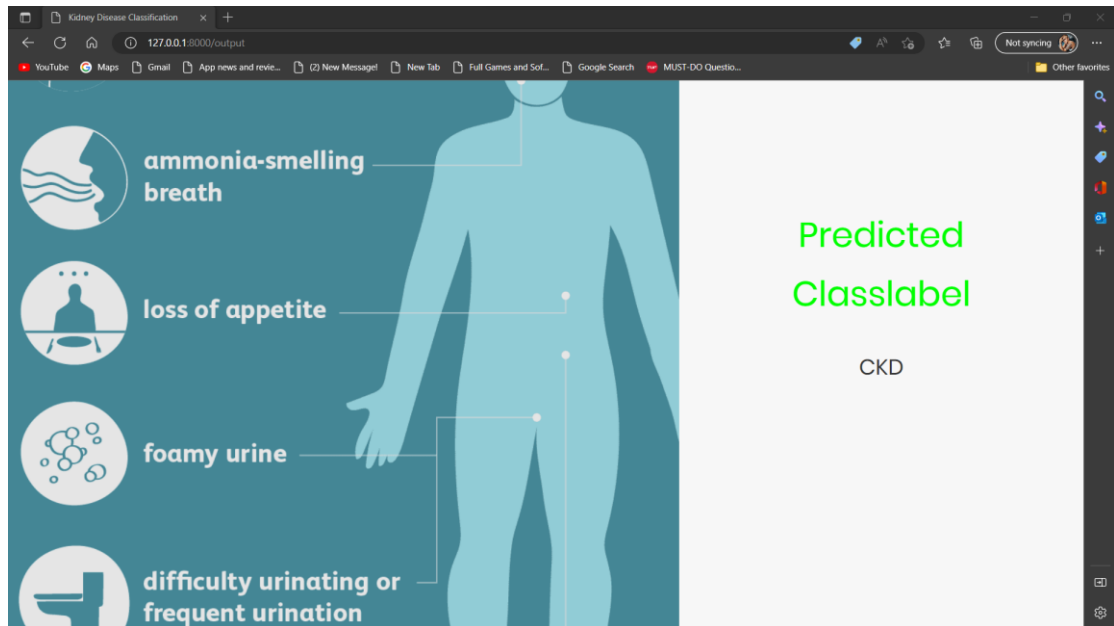
LOGIN PAGE:

The screenshot shows a web browser window with the title "Kidney Disease Classification". The address bar shows the URL "127.0.0.1:8000". The page features a large teal circular logo on the left with the text "Help BEAT Kidney Disease" and a stylized illustration of two kidneys. On the right, there is a login form titled "Login to continue". The form includes fields for "Email" (containing "yusuffsheriff420@gmail.com") and "Password". Below these fields are checkboxes for "Remember me" and a link for "Forgot Password?". A blue "LOGIN" button is positioned below the form. At the bottom, there is a link "or sign up using" followed by Facebook and Twitter social media icons.

DATA PROVIDING PAGE:

The screenshot shows a web browser window with the title "Kidney Disease Classification". The address bar shows the URL "127.0.0.1:8000/input?email=yusuffsheriff420@gmail.com&pass=12345". The page features a large graphic on the left showing two kidneys with yellow branching structures, set against a background of blue and green leaves. On the right, there is a form titled "Provide the data" in green text. The form includes input fields for "Age" (15), "BP" (70), "SG" (1.020), "AL" (4.8), and "Hemo" (17). Below these fields is a dropdown menu currently showing "Logistic Regression". A blue "PREDICT" button is located at the bottom of the form.

OUTPUT PAGE:



Chapter 6

RESULTS

6.1 CONCLUSIONS AND FUTURE WORK

Before the kidneys stop functioning, Chronic Kidney Disease (CKD) should be diagnosed earlier. A Chronic Kidney Disease Prediction System (CKDPS) based on the Random Forest Algorithm has been developed in this paper to make it easier for medical professionals to predict CKD in patients. The Random Forest Algorithm is a machine learning algorithm that improves prediction accuracy and stability by combining decision trees. Cases of 400 patients with 25 attributes (such as red blood cell count, white blood cell count, etc.) are included in the dataset. The strategy for framework execution follows three stages; The collection of the dataset, it is pre-processing, the selection of the target class, and the division of the dataset into training and testing datasets are the primary steps in putting CKDPS into action. The CKD Dataset is subjected to the application of machine learning algorithms like the Logistic Regression (LR) and Random Forest (RF) algorithms in the second step. The accuracy outcomes produced by the various machine learning algorithms differ. Which Random Forest algorithm is used to build CKDPS because it has 99 percent accuracy, precision, and recall? The user submits a query to the system in the final step, and the system provides the user with a classification result.

6.2 REFERENCES

1. Gunarathne, W. H. S. D., Perera, K. D. M., & Kahandawaarachchi, K. A. D. C. P. (2017, October). Performance evaluation on machine learning classification techniques for disease classification and forecasting through data analytics for chronic kidney disease (CKD). In *2017 IEEE 17th international conference on bioinformatics and bioengineering (BIBE)* (pp. 291-296). IEEE.
2. Tekale, S., Shingavi, P., Wandhekar, S., & Chatorikar, A. (2018). Prediction of chronic kidney disease using machine learning algorithm. *International Journal of Advanced Research in Computer and Communication Engineering*, 7(10), 92-96.
3. Arasu, S. D., & Thirumalaiselvi, R. (2017). Review of chronic kidney disease based on data mining techniques. *International Journal of Applied Engineering Research*, 12(23), 13498-13505.
4. Ani, R., Sasi, G., Sankar, U. R., & Deepa, O. S. (2016, September). Decision support system for diagnosis and prediction of chronic renal failure using random subspace classification. In *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)* (pp. 1287-1292). IEEE.
5. Baby, P. S., & Vital, T. P. (2015). Statistical analysis and prediction of kidney diseases using machine learning algorithms. *International Journal of Engineering Research and Technology*, 4(7), 206-210.
6. Sharma, H., & Rizvi, M. A. (2017). Prediction of heart disease using machine learning algorithms: A survey. *International Journal on Recent and Innovation Trends in Computing and Communication*, 5(8), 99-104.
7. Salekin, A., & Stankovic, J. (2016, October). Detection of chronic kidney disease and selecting important predictive attributes. In *2016 IEEE International Conference on Healthcare Informatics (ICHI)* (pp. 262-270). IEEE.
8. Varshney, T., & Verma, K. (2017, July). Rectifying flow of duplicity using bloom filter. In *2017 International Conference on Computer, Communications, and Electronics (Comptelix)* (pp. 300-304). IEEE.
9. Sharma, S., Sharma, V., & Sharma, A. (2016). Performance-based evaluation of various machine learning classification techniques for chronic kidney disease diagnosis. *arXiv preprint arXiv:1606.09581*.