ANALYSIS VARIOUS MACHINE LEARNING ALGORITHMS FOR PREDICTION OF CHRONIC KIDNEY DISEASE

A Project Report submitted in the partial fulfilment of the requirements for the award of the Degree of

BACHELOR OF TECHNOLOGY

Submitted By

Ms.M.BHARGAVI	**Regd.No: 19NE1A1233**
Ms.A.TEJASWINI	**Regd.No: 19NE1A1202**
Ms.S.RAMYA SRI	**Regd.No: 19NE1A1254**
Ms.S.SAI SOWMYA	**Regd.No: 19NE1A1256**
Mr.Sd.TAUFEEQ	**Regd.No: 19NE1A1257**

Under The Esteemed Guidance Of

Mr. K. GOPI B. Tech, M. Tech, (Ph. D).,

Associate Professor

Department of Information Technology

TIRUMALA ENGINEERING COLLEGE

(Approved by AICTE & Affiliated to JNTU, KAKINADA, Accredited By NAAC&NBA) Jonnalagadda, Narasaraopet, GUNTUR (Dt.),A.P.

2019-2023

TIRUMALA ENGINEERING COLLEGE

(Approved by AICTE & Affiliated to JNTU KAKINADA, Accredited by NAAC & NBA) Jonnalagadda, Narasaraopet-522601, Guntur (Dist) A.P

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that the project report entitled **"ANALYSIS VARIOUS MACHINE LEARNING ALGORITHMS FOR PREDICTION OF CHRONIC KIDNEY DISEASE"** is the bonafied work carried out by **M. Bhargavi (19NE1A1233) , A. Tejaswini (19NE1A1202), S. Ramya Sri (19NE1A1254), S. Sai Sowmya (19NE1A1256), Sd. Taufeeq (19NE1A1257)** in partial fulfillment of the requirements for the award of **"Bachelor of Technology"** degree in the **Department of IT** from J.N.T.U. KAKINADA during the year 2022 under our guidance and supervision and worth of acceptance of requirements of the university.

Project Guide

Mr. K. Gopi B. Tech, M. Tech, (Ph. D).,

Head of the Department

Mr. K. Gopi B. Tech, M. Tech, (Ph. D).,

Project Coordinator

Mr. D. Pavan Kumar B. Tech, M. Tech.,

External Examiner

ACKNOWLEDGEMENT

We wish to express our thanks to carious personalities who are responsible for the completion of the project. We are extremely thankful to our beloved chairman **Sri. Bolla Brahma Naidu**, our secretary **Sri. R. Satyanarayana**, Who took keen interest in our every effort throughout this course. We owe out gratitude to our principal sir **Dr. Y. V. Narayana** **M.E., Ph.D, FIETE.,** for his kind attention and valuable guidance throughout the course.

We express our deep felt gratitude to our **Mr. K. Gopi** **B. Tech, M. Tech, (Ph. D).,** and **Mr. D. Pavan Kumar** **B. Tech, M. Tech.,** coordinator of the project for extending their encouragement. Their profound knowledge and willingness have been a constant source of inspiration for us throughout the project work.

We wish to express our sincere deep sense of gratitude to our guide **Mr. K. Gopi** **B. Tech, M. Tech, (Ph. D).,** for significant suggestions and help in every respect to accomplish the project work. Her persisting encouragement, everlasting patience and keen interest in discussions have benefited us to be extent that cannot be spanned by words to our college management for providing excellent lab facilities for completion of project within our campus.

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

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

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

By

Ms.M.Bhargavi (19NE1A1233)

Ms.A.Tejaswini (19NE1A1202)

Ms.S.Ramya Sri (19NE1A1254)

Ms.S.Sai Sowmya (19NE1A1256)

Mr.Sd.Taufeeq (19NE1A1257)

ABSTRACT

Chronic Kidney Disease is a serious lifelong condition that induced by either kidney pathology or reduced kidney functions. We examine the ability of several machine-learning methods for early pre- diction of Chronic Kidney Disease. Predictive analytic is used to examine the relationship between data parameters as well as with the target class attribute. It enables us to introduce the optimal subset of parameters to feed machine learning to build a set of predictive models. Using confusion matrix we received the accuracy of all four methodologies such as K-Nearest Neighbour, Random Forest, Decision Tree and Light Gradient Boosted Machine. We receive the accuracy of KNN is 94 percentage and Random Forest, Decision Tree, Light Gradient Boosted Machine accuracy are 98 percentage. Compared to Light Gradient Boosted Machine, Decision Tree and Random Forest we receive less accuracy in K-Nearest Neighbour. The experimental procedure concludes that advances in machine learning and predictive analytic, represent a promising model to recognize the intelligent solutions, which in turn prove the ability of predication in the kidney disease.

Keywords: Chronic Kidney Disease , Random Forest , Decision Tree, Light Gradient Boosted Machine, K-Nearest Neighbour, Machine Learning, Prediction

LIST OF ACRONYMS AND ABBREVIATIONS

DT	Decision Tree
KNN	K-Nearest Neighbour
LGBM	Light Gradient Boosted Machine
RF	Random Forest
CKD	Chronic Kidney Disease
ESRD	End Stage Renal Disease
GFR	Glamorise Filtration Rate
ML	Machine Learning
GBM	Gradient Boosting Framework
UCI	Unique Client Identifier

INDEX

INTRODUCTION

INTRODUCTION

1.1 Introduction

Chronic kidney disease (CKD) is a significant public health problem for worldwide, especially for a low and medium-income countries. CKD means that the kidney does not work and cannot cor- rectly filter the blood. About 10 percent of the population for a worldwide suffering from (CKD), and millions of die each year because of they couldn't get the affordable treatment, with the number in- creasing in the elderly. In 2010, a study was conducted by International Society of Numerology(ISN) on global burden disease, they reported that CKD has been raised an important cause of the mortality worldwide with the number of deaths increasing by 82.3 percent in the last two decades. Also, the number of patients reaching End-Stage Renal Disease (ESRD) is increasing, which requires kidney

transplantation or dialysis to save the patient's lives.

The worst possible outcome of the chronic kidney disease and the symptoms causing any reduced kidney functioning would lead to kidney failure. When the symptoms are become severe and un- controllable they can be treated through the dialysis and transplantation. Early detection and treatment of CKD can slow or stop the progression of the kidney disease. But the CKD, in it's early stages, has no symptoms. The Proper diagnosis and the testing may be the only way to find out whether the patient has been affected by kidney disease. Early detection of CKD in its initial stages can help the patient get the effective treatment and then prohibit the progression to ESRD. CKD is a Long term condition induced the damage done to both kidneys. Kidney damage refers to any kind of kidney pathology that gives the possibility to reduce the capacity of kidney functions, particularly the reduction in Glamorise Filtration Rate (GFR).

1.2 Aim of the project

The main aim is to identify whether a particular patient is affected by CKD (or) not and it has to be
accurate and precise. So, for that we are going to purpose a correlate four pre-existing Machine
Learning Algorithms to find the best among all. For this purpose, we gathered a CKD data set from
UCI machine learning repository and we examining the correlation between the development of the CKD and predictors using a predictive approach of the analysis. This will help us to reduce the number of required parameters to predict the CKD disease occurrence as well as eliminating the missing, redundant and noisy data. And we have to use certain features to measure its accuracy and predictions.

1.3 Project Domain

Machine Learning is a sub-branch of artificial intelligence which is mostly used for predictive analysis of data, mainly in ML, we have to predict the outcome or the class label and in deep learning it is the exact opposite. Most of the predictive models are built on the principle of Machine Learning.

1.4 Methodology

DECISION TREE:

The UCI's CKD dataset which is selected for decision tree is consisting of attributes like age, blood pressure, specific gratuity, albumin, sugar, red blood cells, plus cell, pus cell clumps, bacteria, blood glucose random, and blood urea. The main purpose is to calculate the performance of various decision tree algorithm and compare their performance. The decision tree techniques used in Random Forest, KNN and Light Gradient Boosted Machine. The results show that Random Forest, LGBM serves the highest accuracy in identifying CKD.

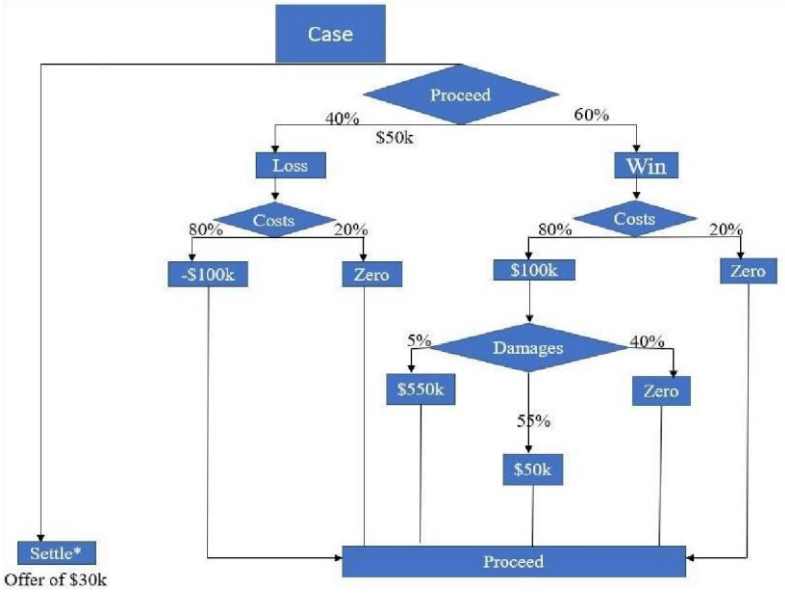


Figure 1.1: **Decision Tree**

Decision Tree Terminologies

- **Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- **Branch/Sub Tree:** A tree formed by splitting the tree.
- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.** By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

- **Information Gain**
- **Gini Index**

Information Gain:

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)]

Entropy:

Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)

Where,

- **S= Total number of samples**
- **P(yes)= probability of yes**
- **P(no)= probability of no**

Gini Index:

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:

$$\textbf{Gini Index} = 1 - \sum_j P_j^2$$

Applications of Decision Trees

1. Assessing prospective growth opportunities

One of the applications of decision trees involves evaluating prospective growth opportunities for businesses based on historical data. Historical data on sales can be used in decision trees that may lead to making radical changes in the strategy of a business to help aid expansion and growth.

2. Using demographic data to find prospective clients

Another application of decision trees is in the use of demographic data to find prospective clients. They can help streamline a marketing budget and make informed decisions on the target market that the business is focused on. In the absence of decision trees, the business may spend its marketing market without a specific demographic in mind, which will affect its overall revenues.

3. Serving as a support tool in several fields

Lenders also use decision trees to predict the probability of a customer defaulting on a loan by applying predictive model generation using the client's past data. The use of a decision tree support tool can help lenders evaluate a customer's creditworthiness to prevent losses.

Decision trees can also be used in operations research in planning logistics and strategic management. They can help in determining appropriate strategies that will help a company achieve its intended goals. Other fields where decision trees can be applied include engineering, education, law, business, healthcare, and finance.

RANDOM FOREST:

In number of different ML classifiers are experimentally validated to a real data set, taken from the UCI Machine Learning Repository. The results are quantitatively and qualitatively discussed and our findings reveal that the random forest (RF) classifier achieves the near-optimal performances on the identification of CKD subjects. RF can also be utilized for the diagnosis of similar diseases. RF con- sists of an outnumbered decision trees where the DT collects their features by the bootstrap training set. Trees in RF are grown using Decision Tree and LGBM method with no pruning. Generalization errors will also increase with outsize trees in the forest.

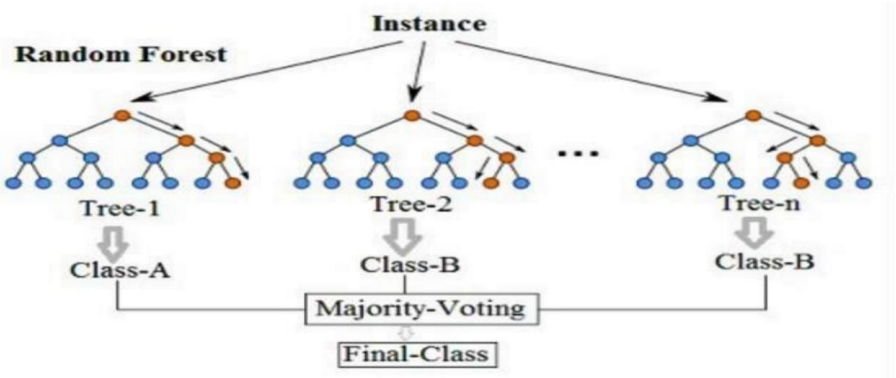


Figure 1.2: **Random Forest**

Why use Random Forest?

Below are some points that explain why we should use the Random Forest algorithm:

<=”” li=””>

- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing.

Applications of Random Forest

There are mainly four sectors where Random forest mostly used:

1. **Banking:** Banking sector mostly uses this algorithm for the identification of loan risk.
2. **Medicine:** With the help of this algorithm, disease trends and risks of the disease can be identified.
3. **Land Use:** We can identify the areas of similar land use by this algorithm.
4. **Marketing:** Marketing trends can be identified using this algorithm.

K-NEAREST NEIGHBOUR:

Chronic Kidney Disease dataset is taken from UCI database which consists of 25 variables with 400 instances. In that we have continuous, nominal and binary variables. Hence nominal variables attributes such as specific gravity, albumin and sugar are taken. We convert all the nominal variables to binary and we use knn classification. k values are chosen.

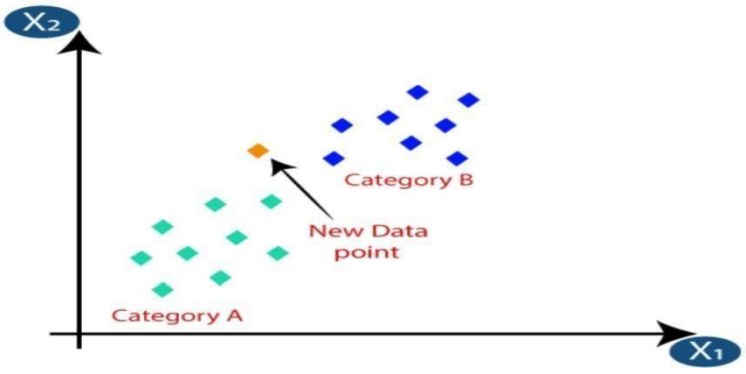


Figure 1.3: **K-Nearest Neighbour**

Distance metrics and K-Nearest Neighbour (KNN)

While outside the scope of this post, in fact, it is worth to mention the K-means algorithm used for classification tasks that uses a similar principle together with KNN of clustering data points together using a distance metric in order to create homogeneous groupings.

Depending on the context of the problem several distance metrics can be used.

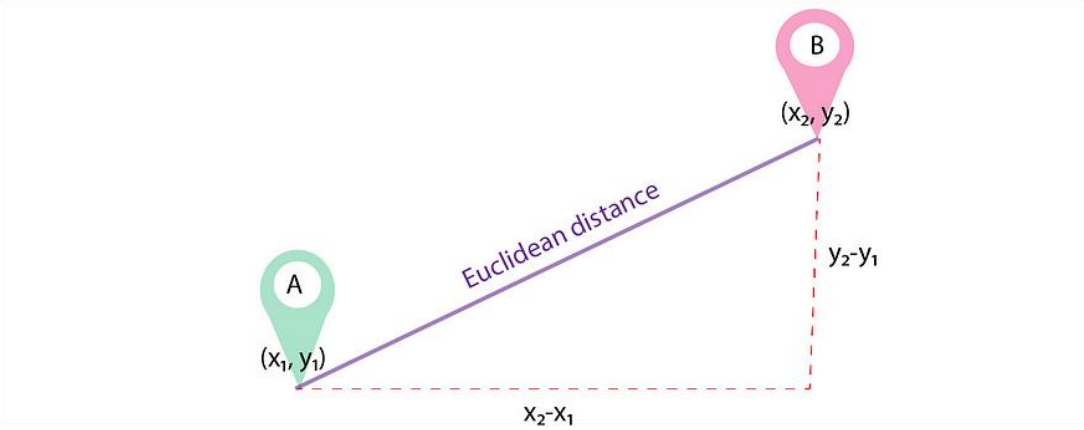
We will be going over:

1. Euclidean distance

2. Manhattan distance

3. Minkowski distance

EUCLIDEAN DISTANCE



The equation at the heart of this distance is the *Pythagorean theorem*!: $a2+b2=c2$.

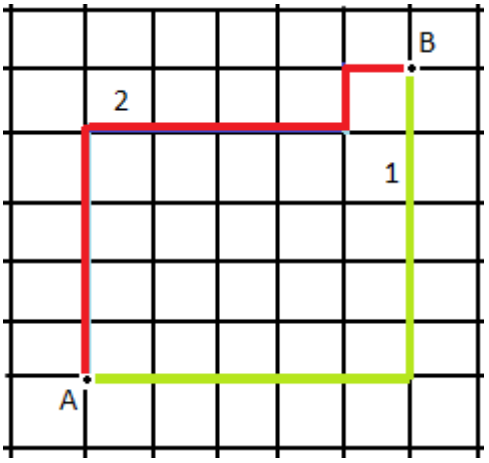
The formula to calculate Euclidean distance is:

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

For each dimension, we subtract one point's value from the other's to get the length of that "side" of the triangle in that dimension, square it, and add it to our running total. The square root of that running total is our Euclidean distance.

Just as with Manhattan distance, we can generalize this equation to n dimensions.

MANHATTAN DISTANCE



The easiest way to remember Manhattan distance is to use the analogy that provides this distance metric it's name look at the picture above, but picture this grid as the famous grid of streets in Manhattan.

The formula to calculate Manhattan distance is:

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

The left side of the equals sign just means "the distance between point x and point y". The ∑ just means "the cumulative sum of each step".

So far, this discussion has explored Manhattan distance in a 2-dimensional space. However, all of the distance metrics we are going to learn can generalize to an n-dimensional space.

MINKOWSKI DISTANCE

The Minkowski distance is a generalized distance metric across a *Normed Vector Space*. A Normed Vector Space is just a fancy way of saying a collection of space where each point has been run through a function. It can be any function, as long it meets two criteria:

1. The zero vector (just a vector filled with zeros) will output a length of 0, and

2. every other vector must have a positive length

Don't worry too much about the specifics of the mathematical definition above. Instead let's try to gain an intuition for what Minkowski distance actually measures. Both the Manhattan and Euclidean distances are actually *special cases of Minkowski distance*, the only thing that changes is the exponent. Let's take a look at the formula:

$$d(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^c \right)^{\frac{1}{c}}$$

Applications of KNN

1. Text mining
2. Agriculture
3. Finance
4. Medical
5. Facial recognition
6. Recommendation systems (Amazon, Hulu, Netflix, etc)

LIGHT GRADIENT BOOSTED MACHINE [LGBM]:

Light GBM can handle the large size of the data and takes least memory to run. Another rea- son of why Light GBM is popular is because it is focuses on the accuracy of results. LGBM also supports GPU learning and thus data scientists are widely using LGBM for data science application development.

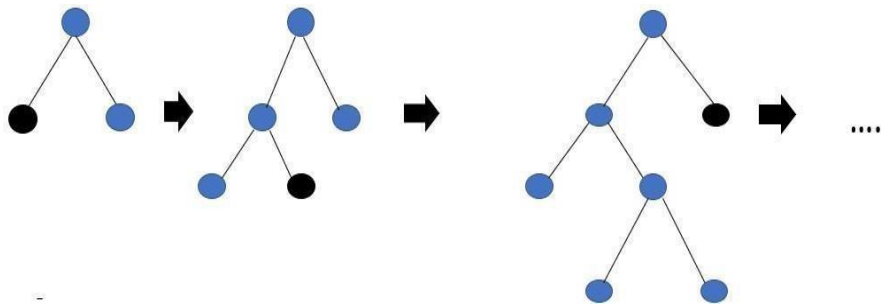


Figure 1.4: Light Gradient Boosted Machine

LITERATURE SURVEY

LITERATURE SURVEY

A. J. Aljaaf et al.[1] is presented by the ability to detect CKD using machine learning algorithms while considering the least number of tests or features. They approach this aim by applying four ma- chine learning classifiers: logistic regression, SVM, random forest, the gradient boosting on a small data set of 400 records. In order to reduce the number of the features and remove redundancy, the association between variables have been studied. A filter feature selection method has been applied to the remaining attributes and found that there are hemoglobin, albumin, and specific gravity have the most impact to predict the CKD.

S. Vijayarani et al.[2] is presented a prediction algorithm to predict CKD at an early stage. The dataset shows input parameters collected from the CKD patients and the models are trained and val- idated for the given input parameters. Decision tree, Random Forest and Support Vector Machine learning models are constructed to carry out the diagnosis of CKD. The performance of models are evaluated based on the accuracy of prediction. The results of the research showed that the Random Forest Classifier model better predicts CKD in comparison to Decision trees and Support the Vector machines.

T Shaikhina and Torgyn, et al.[3] is developed by the objective of the algorithmes to analyses and predict CKD. They have compared the performance of five classifiers in the prognosis of CKD. The experimental results of our proposed method have demonstrated that RF and XGB have produced superior prediction performance in terms of classification accuracy for our considered data set. For the future, they working enhancing the performance of prediction system accuracy by ensemble different classifier algorithms.

Jaymin Patel,et al.[4] is represented by the prediction of chronic kidney disease is very important and now-a-days it is the leading cause of death. The performance of Decision tree method was found to be 99.25 percent accurate compared to naive Bayes method. Classification algorithm on chronic kidney

disease data set the performance is obtained as 99.33 percent Specificity and 99.20 percent Sensitivity. They are also further working on enhancing the performance of prediction system accuracy in neural network and deep learning algorithm .

T Shaikhina, et al.[5] to observe classifiction algorithms to analyses and predict CKD. They have compared the performance of five classifiers in the prognosis of CKD. The experimental results of our proposed method have demonstrated that RF and XGB have produced superior prediction perfor- mance in terms of classification accuracy for our considered data set. For the future, we are going to working for enhancing the performance of prediction system accuracy by ensemble diffe`rent classifier algorithmes.

C.T. Tran, et al.[6] is represented by the ability of machine learning, supported by predictive anal- ysis, for early predication of CKD, an experimental procedure has been undertaken in this study, considering a data set collected from Apollo Hospitals—India, containing 400 instances. Two class labels used as targets in the study (i.e. patients with CKD and healthy individuals), over which four machine-learning methods were simulated.

J.Xiao et al.[7] The linear models including Elastic Net, lasso regression, ridge regression and logistic regression showed the highest overall predictive power, with an average AUC and a precision above 0.87 and 0.8, respectively. Logistic regression ranked first, reaching an AUC of 0.873, with a sensitivity and specificity of 0.83 and 0.82, respectively.

R.Devika, S.V.Avilala and V.Subramaniyaswamy,[8] ”Comparative Study of Classifier for

Chronic Kidney Disease prediction using Naive Bayes, KNN and Random Forest,” 2019 International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2019, pp.
679-684.

SYSTEM ANALYSIS AND DESIGN

SYSTEM ANALYSIS AND DESIGN

3.1 Existing System

As we have mentioned the earlier that it is better to search for an alternative approach with promising results and we are going to do that. So, in this study, we are going to compare four very commonly known machine learning algorithms and we are going to state the best among them using a case study on Chronic Kidney disease Prediction. As stated, the four machine learning algorithms are: Random Forest Decision Tree LGBM K- Near Neighbour (KNN) Why we are going with only four algorithms is that we don't want to overload and confuse the learning procedure by including more machine learning algorithms.

So, we thought that four would be an optimal choice and the machine would work just fine. Using confusion matrix we received the accuracy of all four methodologies such as K-Nearest Neighbor, Random Forest, Decision Tree and Light Gradient Boosted Machine. We receive the accuracy of KNN is 94 percentage and Random Forest, Decision Tree, Light Gradient Boosted Machine accuracy are 98 percentage. Compared to Light Gradient Boosted Machine, Decision Tree and Random Forest we receive less accuracy in K-Nearest Neighbor.

3.2 Proposed System

It is better to search for an alternative approach with promising results and we are going to do that. We going to compare four very commonly known machine learning algorithms and are going to state the best among them using a case study on Chronic Kidney disease Prediction. As stated, the four machine learning algorithms are: Random Forest, Decision Tree, LGBM, and K- Near Neighbour (KNN).

3.3 Feasibility Study

Feasibility of the project is to be verified and analysed that particular phase, based on the feasi- ability report. During this analysis, the proposed system's feasibility produced. There are three key considerations involved in the feasibility analysis are

1. ECONOMIC FEASIBILITY
2. TECHNICAL FEASIBILITY
3. SOCIAL FEASIBILITY

3.3.1 Economic Feasibility

Economic Feasibility is processed to verify whether the user and the company are going to be affected adversely in the terms of money.

3.3.2 Technical Feasibility

This study is processed to verify the technical feasibility, that is, the technical requirements of the proposed system. The new proposed model might the possibility of stealing the data from the existing or the old models. So, in that case, we have to verify whether there is a loss in data or not. Any system that has developed must not have demand on the available technical resources.

3.3.3 Social Feasibility

The main motto of social feasibility is to verify whether the proposed system is adequate by the user or not. That will also include whether the user can access everything which we have proposed.
The user must not be threatened, provided by the system.

3.4 Requirement Specification

3.4.1 Hardware Specification

- Processor : Intel(R) Core(TM) i3-10110U CPU @ 2.10GHz 2.60 GHz
- RAM : 8.00 GB (7.78 GB usable)
- Hard Disc Space : 8GB

3.4.2 Software Specification

- Operating System : Windows 7,8,10 (or) Mac (or) Linux
- Coding Language : Python - version(3.10.8)
- Platform Requirement: Google Colab
- Framework: Kera’s (or) Tensor flow (or) Kaggle

3.5 General Architecture

The overall schema of the predictive model is illustrated in figure 4.1. At the beginning of this study, we have only the CKD data set. After that, we sent the CKD data set into feature selection stage where we receive the prominent and crucial features of our data which we can in the future purposes. And after getting feature we now move on to the next stage where we prepare the data set for a implementation of Random Forest algorithm. From that we are going to receive the best rules with that we are going to train the data using a Decision Tree algorithm and prepare the training data set.

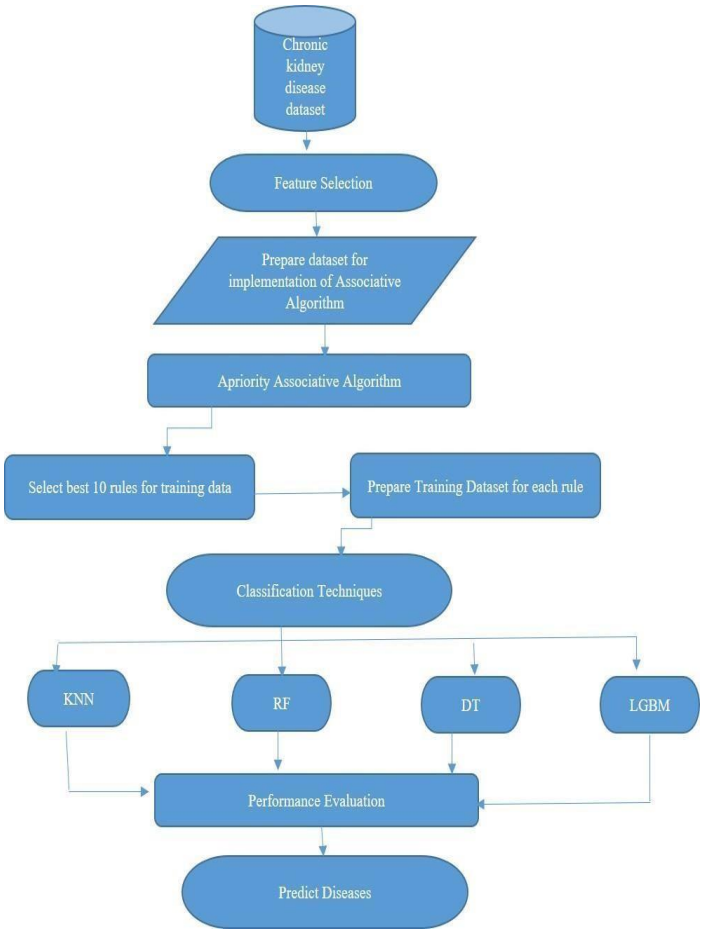


Figure 4.1**: CKD-Architecture Diagram**

3.6 Design Phase

3.6.1 Data Flow Diagram

In Data Flow take the input from the data set. Pre-processing the data it should be training data and testing data. Using different algorithms we can classify the results.

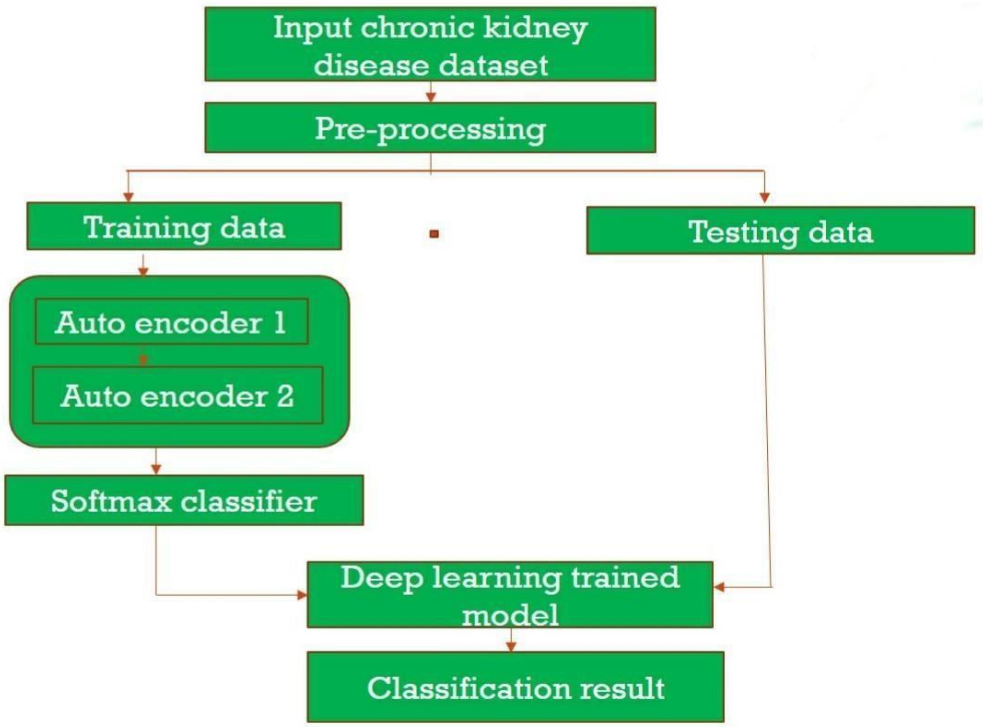


Figure 4.2: Data Flow Diagram

In Figure4.2.1 describe deep learning framework for chronic kidney disease classification using stacked autoencoder model utilizing multimedia data with a softmax classifier. The stacked autoen- coder helps to extract the useful features from the dataset and then a softmax classifier is used to predict the final class. It has experimented on UCI dataset which contains early stages of 400 CKD patients with 25 attributes,

3.7 UML Diagram

In the UML diagram, it represents the various models to be selected and they are connected to the model selection. And the key generation is drawn concerned with the model.

Class Diagram:

The class diagram is the main building block of object oriented modeling. It is used bothfor general conceptual modeling of the systematic of the application, and for detailed modelingtranslating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed.

A class with three sections, in the diagram,classes is represented with boxes which contain three parts:

- The upper part holds the name of the class
- The middle part contains the attributes of the class
- The bottom part gives the methods or operations the class can take or undertake.

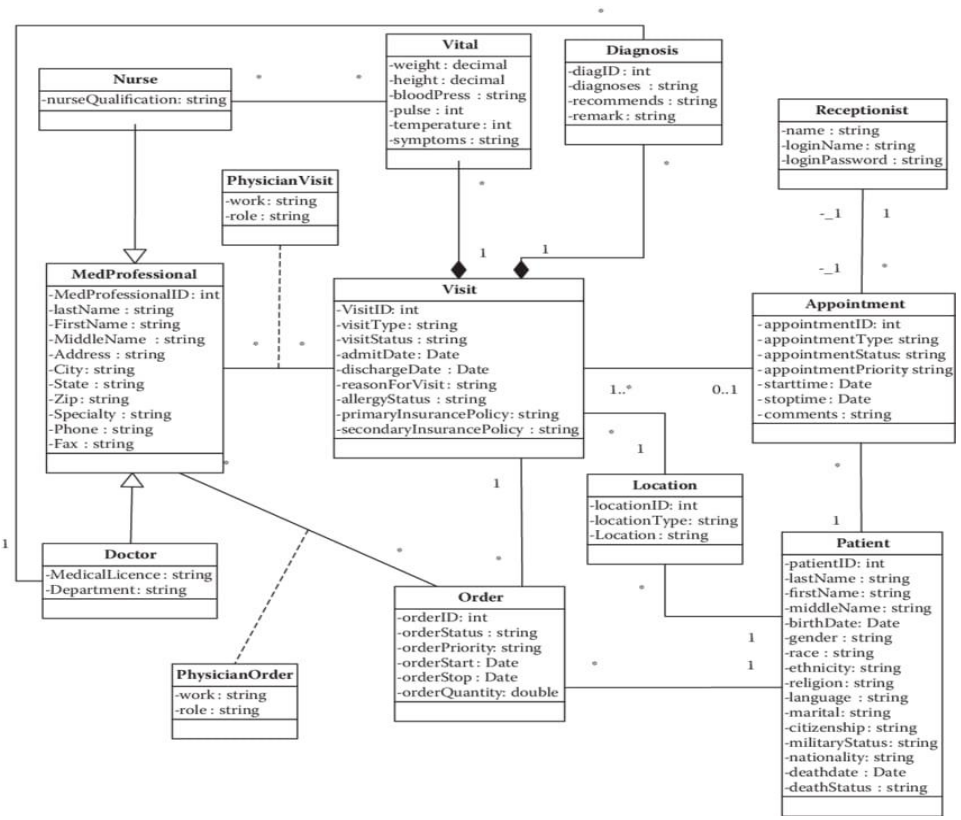


Fig: Class diagram

Use Case Diagram:

A Use Case Diagram at its simplest is a representation of a user's interaction with thesystem and depicting the specifications of a use case. A use case diagram can portray the different types of users of a system and the various ways that they interact with the system. This type of diagram is typically used in conjunction with the textual use case and will oftenbe accompanied by other types of diagrams as well.

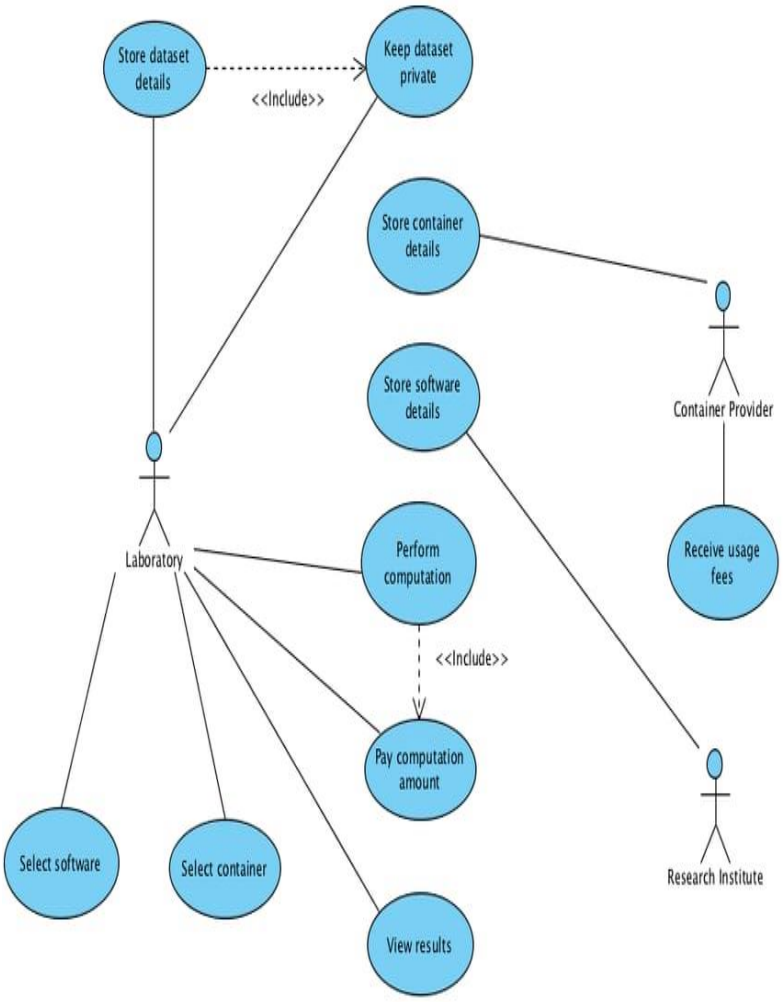


Fig: Use case Diagram

Component Diagram:

In the Unified Modeling Language, a component diagram depicts how components are wired together to form larger components and or software systems. They are used to illustrate the structure of arbitrarily complex systems.

Components are wired together by using an assembly connector to connect the required interface of on component with the provided interface of another component. This illustrates the service consumer - service provider relationship between the two components.

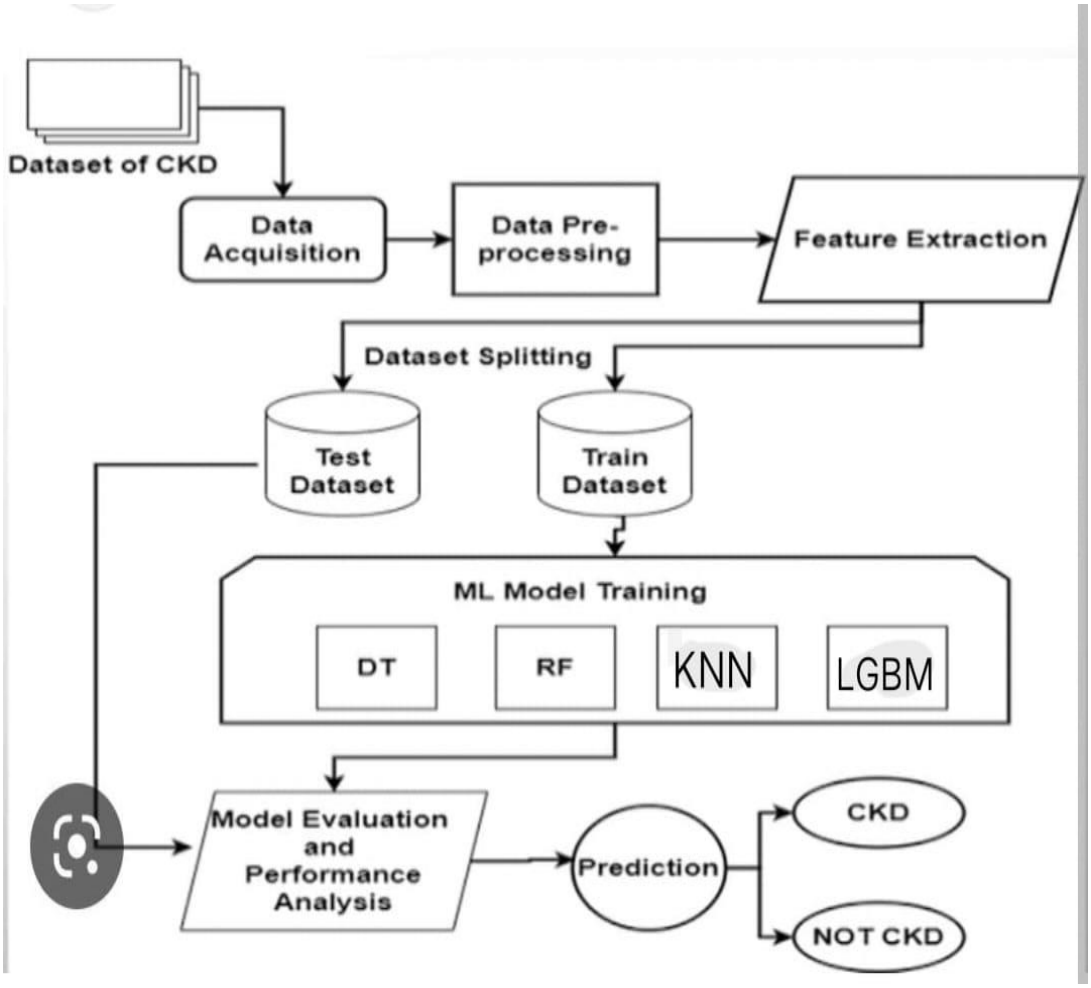


Fig Component Diagram

3.8 Module Description

To implement a predictive model with efficient algorithms we are designed a general depiction of the predictive model along with this framework. Where the selected four machine learning algorithms intention is to provide the same output i.e., to predict the CKD disease in a patient and that is carried out by using several associative rules that we have selected.

3.9 Scope of Project

Predict the kidney disease whether the patient is suffering from kidney diseases by using machine learning algorithms. We can minimize the risk caused by the diseases because "prevention is better

than cure".

Stages of kidney disease

Five stages of kidney disease

The National Kidney Foundation (NKF) divided kidney disease into five stages. This helps doctors provide the best care, as each stage calls for different tests and treatments.

Doctors determine the stage of kidney disease using the glomerular filtration rate (GFR), a math formula using a person's age, gender, and their serum creatinine level (identified through a blood test). Creatinine, a waste product that comes from muscle activity, is a key indicator of kidney function. When kidneys are working well they remove creatinine from the blood; but as kidney function slows, blood levels of creatinine rise.

Use the links below to learn about each stage of kidney disease:

- **Stage 1** with normal or high GFR (GFR > 90 mL/min)
- **Stage 2** Mild CKD (GFR = 60-89 mL/min)
- **Stage 3A** Moderate CKD (GFR = 45-59 mL/min)
- **Stage 3B** Moderate CKD (GFR = 30-44 mL/min)
- **Stage 4** Severe CKD (GFR = 15-29 mL/min)
- **Stage 5** End Stage CKD (GFR <15 mL/min)

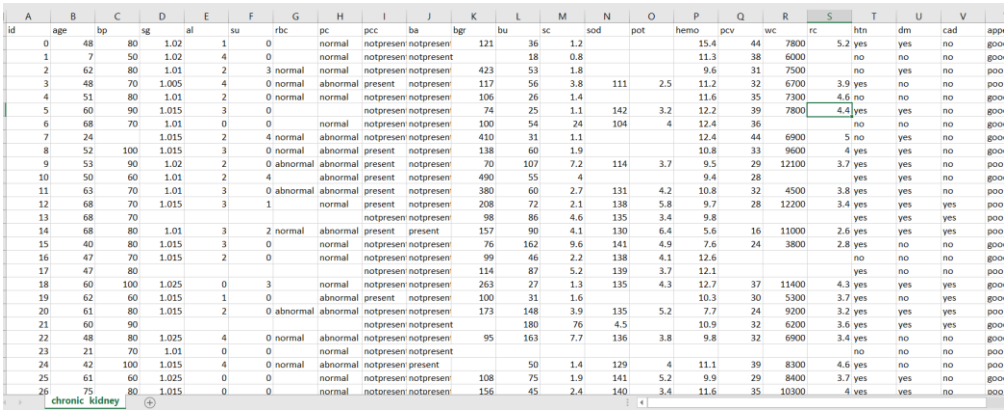
SYMPTOMS:

- weight loss and poor appetite
- swollen ankles, feet or hands – as a result of water retention (oedema)
- shortness of breath
- tiredness
- blood in your pee (urine)
- an increased need to pee – particularly at nights
- difficulty sleeping (insomnia)
- itchy skin
- muscle cramps
- feeling sick
- headaches
- erectile dysfunction in men

Analysis

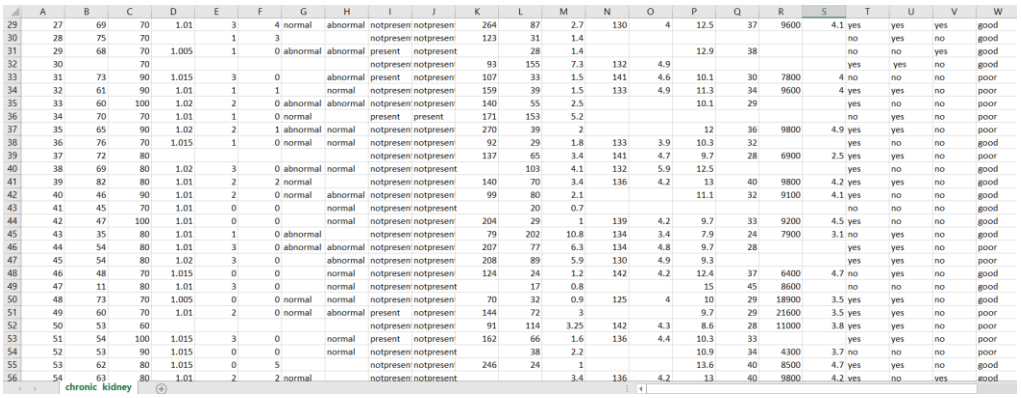
The dataset that we use for the loan prediction was taken from the Kaggle which contain

the details of following attributes to train and test the system for the prediction.



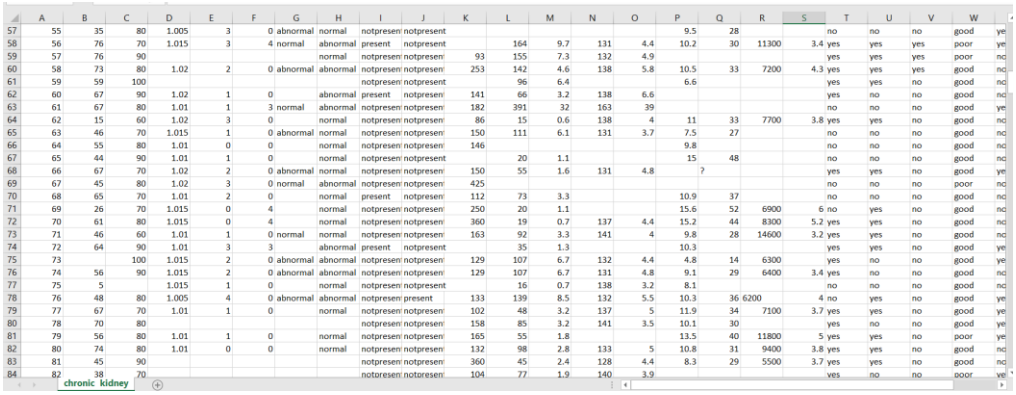
id	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	sod	pot	hemo	pcv	wc	rc	htn	dm	cad	appe
0	48	80	1.02	1	0		normal	notpresen	notpresen	121	36	1.2			15.4	44	7800	5.2	yes	yes	no	goo
1	7	50	1.02	4	0		normal	notpresen	notpresent		18	0.8			11.3	38	6000		no	no	no	goo
2	62	80	1.01	2	3	normal	normal	notpresen	notpresen	423	53	1.8			9.6	31	7500		no	yes	no	poo
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	poo
4	51	80	1.01	2	0	normal	normal	notpresen	notpresen	106	26	1.4			11.6	35	7300	4.6	no	no	no	goo
5	60	90	1.015	3	0			notpresen	notpresen	74	25	1.1	142	3.2	12.2	39	7800	4.4	yes	yes	no	goo
6	68	70	1.01	0	0		normal	notpresen	notpresen	100	54	24	104	4	12.4	36			no	no	no	goo
7	24		1.015	2	4	normal	abnormal	notpresen	notpresen	410	31	1.1			12.4	44	6900	5	no	yes	no	goo
8	52	100	1.015	3	0	normal	abnormal	present	notpresen	138	60	1.9			10.8	33	9600	4	yes	yes	no	goo
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	poo
10	50	60	1.01	2	4		abnormal	present	notpresen	490	55	4			9.4	28			yes	yes	no	goo
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	poo
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	poo
13	68	70						notpresen	notpresen	98	86	4.6	135	3.4	9.8				yes	yes	yes	poo
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	poo
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	goo
16	47	70	1.015	2	0		normal	notpresen	notpresen	99	46	2.2	138	4.1	12.6				no	no	no	goo
17	47	80						notpresen	notpresen	114	87	5.2	139	3.7	12.1				yes	no	no	poo
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	goo
19	62	60	1.015	1	0		abnormal	present	notpresen	100	31	1.6			10.3	30	5300	3.7	yes	no	yes	goo
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	poo
21	60	90						notpresen	notpresent		180	76	4.5		10.9	32	6200	3.6	yes	yes	yes	goo
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	goo
23	21	70	1.01	0	0		normal	notpresen	notpresent										no	no	no	poo
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	poo
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	goo
26	75	80	1.015	0	0		normal	notpresen	notpresen	156	45	2.4	140	3.4	11.6	35	10300	4	yes	yes	no	goo

chronic_kidney



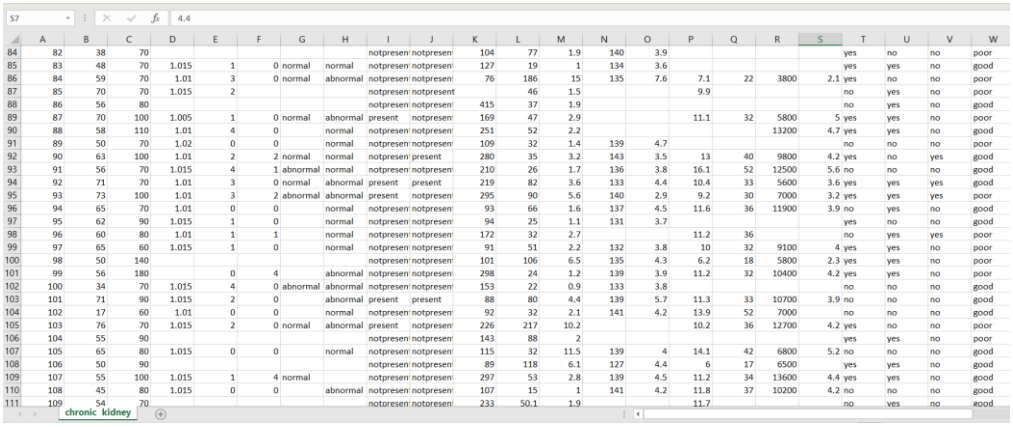
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
29	27	69	70	1.01	3	4	normal	abnormal	notpresen	notpresen	264	87	2.7	130	4	12.5	37	9600	4.1	yes	yes	yes	good
30	28	75	70		1	3			notpresen	notpresen	123	31	1.4							no	yes	no	good
31	29	68	70	1.005	1	0	abnormal	abnormal	present	notpresent		28	1.4			12.9	38			no	no	yes	good
32	30		70						notpresen	notpresen	93	155	7.3	132	4.9					yes	yes	no	good
33	31	73	90	1.015	3	0		abnormal	present	notpresen	107	33	1.5	141	4.6	10.1	30	7800	4	no	no	no	poor
34	32	61	90	1.01	1	1		normal	notpresen	notpresen	159	39	1.5	133	4.9	11.3	34	9600	4	yes	yes	no	poor
35	33	60	100	1.02	2	0	abnormal	abnormal	notpresen	notpresen	140	55	2.5			10.1	29			yes	no	no	poor
36	34	70	70	1.01	1	0	normal		present	present	171	153	5.2							no	yes	no	poor
37	35	65	90	1.02	2	1	abnormal	normal	notpresen	notpresen	270	39	2			12	36	9800	4.9	yes	yes	no	poor
38	36	76	70	1.015	1	0	normal	normal	notpresen	notpresen	92	29	1.8	133	3.9	10.3	32			yes	no	no	good
39	37	72	80						notpresen	notpresen	137	65	3.4	141	4.7	9.7	28	6900	2.5	yes	yes	no	poor
40	38	69	80	1.02	3	0	abnormal	normal	notpresen	notpresent		103	4.1	132	5.9	12.5				yes	no	no	good
41	39	82	80	1.01	2	2	normal		notpresen	notpresen	140	70	3.4	136	4.2	13	40	9800	4.2	yes	yes	no	good
42	40	46	90	1.01	2	0	normal	abnormal	notpresen	notpresen	99	80	2.1			11.1	32	9100	4.1	yes	no	no	good
43	41	45	70	1.01	0	0		normal	notpresen	notpresent		20	0.7							no	no	no	good
44	42	47	100	1.01	0	0		normal	notpresen	notpresen	204	29	1	139	4.2	9.7	33	9200	4.5	yes	no	no	good
45	43	35	80	1.01	1	0	abnormal		notpresen	notpresen	79	202	10.8	134	3.4	7.9	24	7900	3.1	no	yes	no	good
46	44	54	80	1.01	3	0	abnormal	abnormal	notpresen	notpresen	207	77	6.3	134	4.8	9.7	28			yes	yes	no	poor
47	45	54	80	1.02	3	0		abnormal	notpresen	notpresen	208	89	5.9	130	4.9	9.3				yes	yes	no	poor
48	46	48	70	1.015	0	0		normal	notpresen	notpresen	124	24	1.2	142	4.2	12.4	37	6400	4.7	no	yes	no	good
49	47	11	80	1.01	3	0		normal	notpresen	notpresent		17	0.8			15	45	8600		no	no	no	good
50	48	73	70	1.005	0	0	normal	normal	notpresen	notpresen	70	32	0.9	125	4	10	29	18900	3.5	yes	yes	no	good
51	49	60	70	1.01	2	0	normal	abnormal	present	notpresen	144	72	3			9.7	29	21600	3.5	yes	yes	no	poor
52	50	53	60						notpresen	notpresen	91	114	3.25	142	4.3	8.6	28	11000	3.8	yes	yes	no	poor
53	51	54	100	1.015	3	0		normal	present	notpresen	162	66	1.6	136	4.4	10.3	33			yes	yes	no	poor
54	52	53	90	1.015	0	0		normal	notpresen	notpresent		38	2.2			10.9	34	4300	3.7	no	no	no	poor
55	53	62	80	1.015	0	5			notpresen	notpresen	246	24	1			13.6	40	8500	4.7	yes	yes	no	good
56	54	63	80	1.01	2	2	normal		notpresen	notpresent			3.4	136	4.2	13	40	9800	4.2	yes	no	yes	good

chronic_kidney

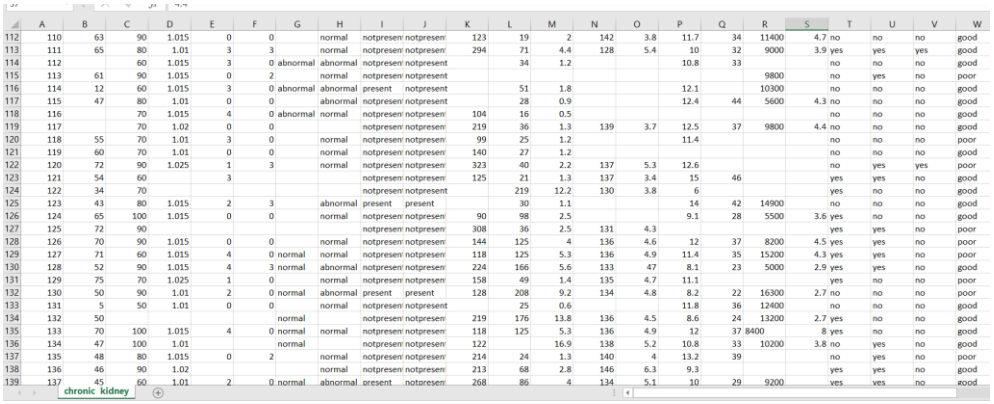


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
57	55	35	80	1.005	3	0	abnormal	normal	notpresen	notpresent						9.5	28			no	no	no	good
58	56	76	70	1.015	3	4	normal	abnormal	present	notpresent		164	9.7	131	4.4	10.2	30	11300	3.4	yes	yes	yes	poor
59	57	76	90					normal	notpresen	notpresen	93	155	7.3	132	4.9					yes	yes	yes	poor
60	58	73	80	1.02	2	0	abnormal	abnormal	notpresen	notpresen	253	142	4.6	138	5.8	10.5	33	7200	4.3	yes	yes	yes	good
61	59	59	100						notpresen	notpresent		96	6.4			6.6				yes	yes	no	good
62	60	67	90	1.02	1	0		abnormal	present	notpresen	141	66	3.2	138	6.6					yes	no	no	good
63	61	67	80	1.01	1	3	normal	abnormal	notpresen	notpresen	182	391	32	163	39					no	no	no	good
64	62	15	60	1.02	3	0		normal	notpresen	notpresen	86	15	0.6	138	4	11	33	7700	3.8	yes	yes	no	good
65	63	46	70	1.015	1	0	abnormal	normal	notpresen	notpresen	150	111	6.1	131	3.7	7.5	27			no	no	no	good
66	64	55	80	1.01	0	0		normal	notpresen	notpresen	146					9.8				no	no	no	good
67	65	44	90	1.01	1	0		normal	notpresen	notpresent		20	1.1			15	48			no	no	no	good
68	66	67	70	1.02	2	0	abnormal	normal	notpresen	notpresen	150	55	1.6	131	4.8	?				yes	yes	no	good
69	67	45	80	1.02	3	0	normal	abnormal	notpresen	notpresen	425									no	no	no	poor
70	68	65	70	1.01	2	0		normal	present	notpresen	112	73	3.3			10.9	37			no	no	no	good
71	69	26	70	1.015	0	4		normal	notpresen	notpresen	250	20	1.1			15.6	52	6900	6	no	yes	no	good
72	70	61	80	1.015	0	4		normal	notpresen	notpresen	360	19	0.7	137	4.4	15.2	44	8300	5.2	yes	yes	no	good
73	71	46	60	1.01	1	0	normal	normal	notpresen	notpresen	163	92	3.3	141	4	9.8	28	14600	3.2	yes	yes	no	good
74	72	64	90	1.01	3	3		abnormal	present	notpresent		35	1.3			10.3				yes	yes	no	good
75	73		100	1.015	2	0	abnormal	abnormal	notpresen	notpresen	129	107	6.7	132	4.4	4.8	14	6300		yes	no	no	good
76	74	56	90	1.015	2	0	abnormal	abnormal	notpresen	notpresen	129	107	6.7	131	4.8	9.1	29	6400	3.4	yes	no	no	good
77	75	5		1.015	1	0		normal	notpresen	notpresent		16	0.7	138	3.2	8.1				no	no	no	good
78	76	48	80	1.005	4	0	abnormal	abnormal	notpresen	present	133	139	8.5	132	5.5	10.3	36	6200	4	no	yes	no	good
79	77	67	70	1.01	1	0		normal	notpresen	notpresen	102	48	3.2	137	5	11.9	34	7100	3.7	yes	yes	no	good
80	78	70	80						notpresen	notpresen	158	85	3.2	141	3.5	10.1	30			yes	no	no	good
81	79	56	80	1.01	1	0		normal	notpresen	notpresen	165	55	1.8			13.5	40	11800	5	yes	yes	no	poor
82	80	74	80	1.01	0	0		normal	notpresen	notpresen	132	98	2.8	133	5	10.8	31	9400	3.8	yes	yes	no	good
83	81	45	90						notpresen	notpresen	360	45	2.4	128	4.4	8.3	29	5500	3.7	yes	yes	no	good
84	82	38	70						notpresen	notpresen	104	77	1.9	140	3.9					yes	no	no	poor

chronic_kidney



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
84	82	38	70						notpresen	notpresen	104	77	1.9	140	3.9					yes	no	no	poor
85	83	48	70	1.015	1	0	normal	normal	notpresen	notpresen	127	19	1	134	3.6					yes	yes	no	good
86	84	59	70	1.01	3	0	normal	abnormal	notpresen	notpresen	76	186	15	135	7.6	7.1	22	3800	2.1	yes	no	no	poor
87	85	70	70	1.015	2				notpresen	notpresent		46	1.5			9.9				no	yes	no	poor
88	86	56	80						notpresen	notpresen	415	37	1.9							no	yes	no	good
89	87	70	100	1.005	1	0	normal	abnormal	present	notpresen	169	47	2.9			11.1	32	5800	5	yes	yes	no	poor
90	88	58	110	1.01	4	0		normal	notpresen	notpresen	251	52	2.2					13200	4.7	yes	yes	no	good
91	89	50	70	1.02	0	0		normal	notpresen	notpresen	109	32	1.4	139	4.7					no	no	no	poor
92	90	63	100	1.01	2	2	normal	normal	notpresen	present	280	35	3.2	143	3.5	13	40	9800	4.2	yes	no	yes	good
93	91	56	70	1.015	4	1	abnormal	normal	notpresen	notpresen	210	26	1.7	136	3.8	16.1	52	12500	5.6	no	no	no	good
94	92	71	70	1.01	3	0	normal	abnormal	present	present	219	82	3.6	133	4.4	10.4	33	5600	3.6	yes	yes	yes	good
95	93	73	100	1.01	3	2	abnormal	abnormal	present	notpresen	295	90	5.6	140	2.9	9.2	30	7000	3.2	yes	yes	yes	poor
96	94	65	70	1.01	0	0		normal	notpresen	notpresen	93	66	1.6	137	4.5	11.6	36	11900	3.9	no	yes	no	good
97	95	62	90	1.015	1	0		normal	notpresen	notpresen	94	25	1.1	131	3.7					yes	no	no	good
98	96	60	80	1.01	1	1		normal	notpresen	notpresen	172	32	2.7			11.2	36			no	yes	yes	poor
99	97	65	60	1.015	1	0		normal	notpresen	notpresen	91	51	2.2	132	3.8	10	32	9100	4	yes	yes	no	poor
100	98	50	140						notpresen	notpresen	101	106	6.5	135	4.3	6.2	18	5800	2.3	yes	yes	no	poor
101	99	56	180		0	4		abnormal	notpresen	notpresen	298	24	1.2	139	3.9	11.2	32	10400	4.2	yes	yes	no	poor
102	100	34	70	1.015	4	0	abnormal	abnormal	notpresen	notpresen	153	22	0.9	133	3.8					no	no	no	good
103	101	71	90	1.015	2	0		abnormal	present	present	88	80	4.4	139	5.7	11.3	33	10700	3.9	no	no	no	good
104	102	17	60	1.01	0	0		normal	notpresen	notpresen	92	32	2.1	141	4.2	13.9	52	7000		no	no	no	good
105	103	76	70	1.015	2	0	normal	abnormal	present	notpresen	226	217	10.2			10.2	36	12700	4.2	yes	no	no	poor
106	104	55	90						notpresen	notpresen	143	88	2							yes	yes	no	poor
107	105	65	80	1.015	0	0		normal	notpresen	notpresen	115	32	11.5	139	4	14.1	42	6800	5.2	no	no	no	good
108	106	50	90						notpresen	notpresen	89	118	6.1	127	4.4	6	17	6500		yes	yes	no	good
109	107	55	100	1.015	1	4	normal		notpresen	notpresen	297	53	2.8	139	4.5	11.2	34	13600	4.4	yes	yes	no	good
110	108	45	80	1.015	0	0		abnormal	notpresen	notpresen	107	15	1	141	4.2	11.8	37	10200	4.2	no	no	no	good
111	109	54	70						notpresen	notpresen	233	50.1	1.9			11.7				no	yes	no	good



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
112	110	63	90	1.015	0	0		normal	notpresen	notpresen	123	19	2	142	3.8	11.7	34	11400	4.7	no	no	no	good
113	111	65	80	1.01	3	3		normal	notpresen	notpresen	294	71	4.4	128	5.4	10	32	9000	3.9	yes	yes	yes	good
114	112		60	1.015	3	0	abnormal	abnormal	notpresen	notpresent		34	1.2			10.8	33			no	no	no	good
115	113	61	90	1.015	0	2		normal	notpresen	notpresent								9800		no	yes	no	poor
116	114	12	60	1.015	3	0	abnormal	abnormal	present	notpresent		51	1.8			12.1		10300		no	no	no	good
117	115	47	80	1.01	0	0		abnormal	notpresen	notpresent		28	0.9			12.4	44	5600	4.3	no	no	no	good
118	116		70	1.015	4	0	abnormal	normal	notpresen	notpresen	104	16	0.5							no	no	no	good
119	117		70	1.02	0	0			notpresen	notpresen	219	36	1.3	139	3.7	12.5	37	9800	4.4	no	no	no	good
120	118	55	70	1.01	3	0		normal	notpresen	notpresen	99	25	1.2			11.4				no	no	no	poor
121	119	60	70	1.01	0	0		normal	notpresen	notpresen	140	27	1.2							no	no	no	good
122	120	72	90	1.025	1	3		normal	notpresen	notpresen	323	40	2.2	137	5.3	12.6				no	yes	yes	poor
123	121	54	60		3				notpresen	notpresen	125	21	1.3	137	3.4	15	46			yes	yes	no	good
124	122	34	70						notpresen	notpresent		219	12.2	130	3.8	6				yes	no	no	good
125	123	43	80	1.015	2	3		abnormal	present	present		30	1.1			14	42	14900		no	no	no	good
126	124	65	100	1.015	0	0		normal	notpresen	notpresen	90	98	2.5			9.1	28	5500	3.6	yes	no	no	good
127	125	72	90						notpresen	notpresen	308	36	2.5	131	4.3					yes	yes	no	poor
128	126	70	90	1.015	0	0		normal	notpresen	notpresen	144	125	4	136	4.6	12	37	8200	4.5	yes	yes	no	poor
129	127	71	60	1.015	4	0	normal	normal	notpresen	notpresen	118	125	5.3	136	4.9	11.4	35	15200	4.3	yes	yes	no	poor
130	128	52	90	1.015	4	3	normal	abnormal	notpresen	notpresen	224	166	5.6	133	47	8.1	23	5000	2.9	yes	yes	no	good
131	129	75	70	1.025	1	0		normal	notpresen	notpresen	158	49	1.4	135	4.7	11.1				yes	no	no	poor
132	130	50	90	1.01	2	0	normal	abnormal	present	present	128	208	9.2	134	4.8	8.2	22	16300	2.7	no	no	no	poor
133	131	5	50	1.01	0	0		normal	notpresen	notpresent		25	0.6			11.8	36	12400		no	no	no	good
134	132	50					normal		notpresen	notpresen	219	176	13.8	136	4.5	8.6	24	13200	2.7	yes	no	no	good
135	133	70	100	1.015	4	0	normal	normal	notpresen	notpresen	118	125	5.3	136	4.9	12	37	8400	8	yes	no	no	good
136	134	47	100	1.01			normal		notpresen	notpresen	122		16.9	138	5.2	10.8	33	10200	3.8	no	yes	no	good
137	135	48	80	1.015	0	2		normal	notpresen	notpresen	214	24	1.3	140	4	13.2	39			no	yes	no	poor
138	136	46	90	1.02				normal	notpresen	notpresen	213	68	2.8	146	6.3	9.3				yes	yes	no	good
139	137	45	60	1.01	2	0	normal	abnormal	present	notpresen	268	86	4	134	5.1	10	29	9200		yes	yes	no	good

IMPLEMENTATION

IMPLEMENTATION:

PYTHON:

Python is a computer programming language often used to build websites and software,automate

tasks, and conduct data analysis.

Python is a general-purpose language, meaning it can be used to create a variety of differentprograms and isn't specialized for any specific problems

FEATURES OF PYTHON

- Easy to Code. Python is a very high-level programming language, yet it is effortless to learn.
- Easy to Read. Python code looks like simple English words.
- Free and Open-Source.
- Robust Standard Library.
- Interpreted.
- Portable.
- Object-Oriented and Procedure-Oriented.
- Extensible.

APPLICATIONS OF PYTHON

- Web Development.
- Game Development.
- Scientific and Numeric Applications.
- Artificial Intelligence and Machine Learning.
- Software Development.
- Enterprise-level/Business Applications.
- Education programs and training courses.

MACHINE LEARNING

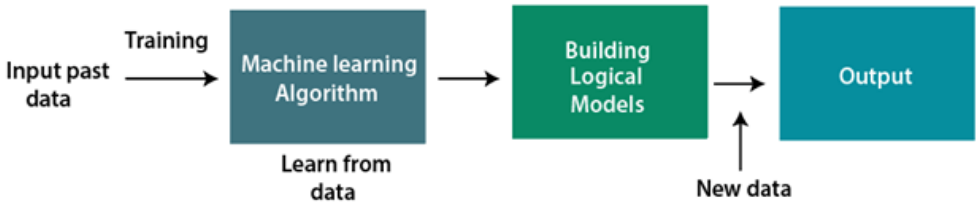
Machine learning is a growing technology which enables computers to learn automatically from past data. Machine learning uses various algorithms for **building mathematical models and making predictions using historical data or information**. Currently, it is being used for various tasks such as **image recognition**, **speech recognition**, **email filtering**, **Facebook auto-tagging**, **recommender system**, and many more.

This machine learning tutorial gives you an introduction to machine learning along with the wide range of machine learning techniques such as **Supervised**, **Unsupervised**, and **Reinforcement** learning. You will learn about regression and classification models, clustering methods, hidden Markov models, and various sequential models.

How does Machine Learning work

A Machine Learning system **learns from historical data, builds the prediction models, and whenever it receives new data, predicts the output for it**. The accuracy of predicted output depends upon the amount of data, as the huge amount of data helps to build a better model which predicts the output more accurately.

Suppose we have a complex problem, where we need to perform some predictions, so instead of writing a code for it, we just need to feed the data to generic algorithms, and with the help of these algorithms, machine builds the logic as per the data and predict the output. Machine learning has changed our way of thinking about the problem. The below block diagram explains the working of Machine Learning algorithm:



Features of Machine Learning:

- Machine learning uses data to detect various patterns in a given dataset.
- It can learn from past data and improve automatically.
- It is a data-driven technology.
- Machine learning is much similar to data mining as it also deals with the huge amount of the data.

Glob:

The glob module, which is short for global, is a function that's used to search for files that match a specific file pattern or name. It can be used to search CSV files and for text in files. Python glob. glob() method returns a list of files or folders that matches the path specified in the pathname argument. This function takes two arguments, namely pathname, and recursive flag. pathname : Absolute (with full path and the file name) or relative (with UNIX shell-style wildcards).

Implementation of machine learning using Python:

The most recent major version of Python is Python 3. It is possible to write Python in an Integrated Development Environment, such as Pycharm, Netbeans or Eclipse, Anaconda which are particularly useful when managing larger collections of Python files. In the older days, people used to perform Machine Learning tasks manually by coding all the algorithms and mathematical and statistical formula. This made the process time consuming, tedious and inefficient. But in the modern days, it is become very much easy and efficient compared to the olden days by various python libraries, frameworks, and modules. Today, Python is one of the most popular programming languages for this task and it has replaced many languages in the industry, one of the reason is its vast collection of libraries.

Python libraries that used in Machine Learning are:

- NumPy
- Pandas
- Matplotlib
- Scikit-learn
- TensorFlow

NumPy:

It is a very popular python library for large multi-dimensional array and matrix processing, with the help of a large collection of high-level mathematical functions. It is very useful for fundamental scientific computations in Machine Learning. It is particularly useful for linear algebra, Fourier transform, and random number capabilities. High-end libraries like TensorFlow uses NumPy internally for manipulation of Tensors.

Pandas:

It is a popular Python library for data analysis. It is not directly related to MachineLearning. As we know that the dataset must be prepared before training. In this case, Pandas comes handy as it was developed specifically for data extraction and preparation. It provides high-level data structures and wide variety tools for data analysis. It provides many inbuilt methods for groping, combining and filtering data.

Matplotlib:

It is a very popular Python library for data visualization. Like Pandas, it is notdirectly related to Machine Learning. It particularly comes in handy when a programmer wants to visualize the patterns in the data. It is a 2D plotting library used for creating 2D graphs and plots. A module named pyplot makes it easy for programmers for plotting as it provides features to control line styles, font properties, formatting axes. It provides variouskinds of graphs and plots for data visualization, histogram, error charts, bar chats, etc.

Scikit-learn:

It is one of the most popular Machine Learning libraries for classical MachineLearning algorithms. It is built on top of two basic Python libraries, NumPy and SciPy.

Scikit-learn supports most of the supervised and unsupervised learning algorithms

TensorFlow:

It is a very popular open-source library for high performance numerical computation developed by the Google Brain team in Google. As the name suggests, TensorFlow is a framework that involves defining and running computations involving tensors. It can train and run deep neuralnetworks that can be used to develop several AI applications. TensorFlow is widely used in the field of deep learning research and application.

Kera's:

Kera's is a high-level, deep learning API developed by Google for implementing neural networks. It is written in Python and is used to make the implementation of neural networks easy. It also supports multiple backend neural network computation.

Kera's is relatively easy to learn and work with because it provides a python frontend with a high level of abstraction while having the option of multiple back-ends for computation purposes. This makes Keras slower than other deep learning frameworks, but extremely beginner-friendly.

MinMaxScaller:

For each value in a feature, MinMaxScaler subtracts the minimum value in the feature and then divides by the range. The range is the difference between the original maximum and original minimum.

MinMaxScaler preserves the shape of the original distribution. It doesn't meaningfully change the information embedded in the original data.

Note that MinMaxScaler doesn't reduce the importance of outliers.

The default range for the feature returned by MinMaxScaler is 0 to 1.

Confusion Matrix:

The confusion matrix is a matrix used to determine the performance of the classification models for a given set of test data. It can only be determined if the true values for test data are known. The matrix itself can be easily understood, but the related terminologies may be confusing.

n = total predictions	Actual: No	Actual: Yes
Predicted: No	True Negative	False Positive
Predicted: Yes	False Negative	True Positive

The above table has the following cases:

- **True Negative:** Model has given prediction No, and the real or actual value was also No.
- **True Positive:** The model has predicted yes, and the actual value was also true.
- **False Negative:** The model has predicted no, but the actual value was Yes, it is also called as **Type-II error**.
- **False Positive:** The model has predicted Yes, but the actual value was No. It is also called a **Type-I error.**

Calculations using Confusion Matrix:

➢ **Classification Accuracy:** It is one of the important parameters to determine the accuracy of the classification problems. It defines how often the model predicts the correct output. It can be calculated as the ratio of the number of correct predictions made by the classifier to all number of predictions made by the classifiers. The formula is given below:

$$\textbf{Accuracy} = (TP+TN)/(TP+FP+FN+TN)$$

➢ **Misclassification rate:** It is also termed as Error rate, and it defines how often the model gives the wrong predictions. The value of error rate can be calculated as the number of incorrect predictions to all number of the predictions made by the classifier. The formula is given below:

$$\textbf{Error rate} = (FP+FN)/(TP+FP+FN+TN)$$

➢ **Precision:** It can be defined as the number of correct outputs provided by the model or out of all positive classes that have predicted correctly by the model, how many of them were actually true. It can be calculated using the below formula:

$$\textbf{Precision} = TP/(TP+FP)$$

- **Recall:** It is defined as the out of total positive classes, how our model predicted correctly. The recall must be as high as possible.

Recall=TP/(TP+FN)

- **F-measure:** If two models have low precision and high recall or vice versa, it is difficult to compare these models. So, for this purpose, we can use F-score. This score helps us to evaluate the recall and precision at the same time. The F-score is maximum if the recall is equal to the precision. It can be calculated using the below formula:

F-Measure=(2*Recall+Precision)/(Recall+Precision)

SAMPLE CODE

```
#import libraries
import glob
from keras.models import Sequential,load_model
import numpy as np
import pandas as pd
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder,MinMaxScaler
import matplotlib.pyplot as plt
import keras as k


#data set insertion
from google.colab import files
uploaded = files.upload()
df = pd.read_csv('chronic  kidney.csv')
df.head(5)


df.shape

columns_to_retain =
['sg','al','sc','hemo','pcv','wbcc','rbcc','htn','classification']
df = df.drop( [col for col in df.columns if not col in columns_to_retain] ,
axis=1 )
df = df.dropna(axis=0)


for column in df.columns:
    if df[column].dtype == np.number:
      continue
    df[column] = LabelEncoder().fit_transform( df[column] )


df.head()


x = df.drop(['classification'],axis=1)
y = df['classification']
```

```
x_scaler = MinMaxScaler()
x_scaler.fit(x)
column_names = x.columns
x[column_names] = x_scaler.transform(x)


x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,
shuffle=True)



model = Sequential()
model.add( Dense(256, input_dim= len(x.columns),
kernel_initializer=k.initializers.random_normal(seed=13), activation='relu'))
model.add( Dense(1, activation='hard_sigmoid'))


model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])


history = model.fit(x_train, y_train, epochs=2000, batch_size=
x_train.shape[0])

model.save('ckd.model')

plt.plot(history.history['accuracy'])
plt.plot(history.history['loss'])
plt.title('model accuracy & loss')
plt.ylabel('accuracy and loss')
plt.xlabel('epoch')

print('shape of traning data:',x_train.shape)
print('shape of test data:',x_test.shape)


pred = model.predict(x_test)
pred = [1 if y>=0.5 else 0 for y in pred]
pred
print('Original : {0}'.format(", ".join(str(x) for x in y_test)))
print('Predicted : {0}'.format(", ".join(str(x) for x in pred)))


from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
```

```
knn = KNeighborsClassifier()
knn.fit(x_train, y_train)

knn_acc = accuracy_score(y_test, knn.predict(x_test))

print(f"Training Accuracy of KNN is {accuracy_score(y_train,
knn.predict(x_train))}")
print(f"Test Accuracy of KNN is {knn_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test,
knn.predict(x_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test,
knn.predict(x_test))}")

from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier()
dtc.fit(x_train, y_train)

# accuracy score, confusion matrix and classification report of decision tree

dtc_acc = accuracy_score(y_test, dtc.predict(x_test))

print(f"Training Accuracy of Decision Tree Classifier is
{accuracy_score(y_train, dtc.predict(x_train))}")
print(f"Test Accuracy of Decision Tree Classifier is {dtc_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test,
dtc.predict(x_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test,
dtc.predict(x_test))}")



from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier()
dtc.fit(x_train, y_train)

# accuracy score, confusion matrix and classification report of decision tree

dtc_acc = accuracy_score(y_test, dtc.predict(x_test))

print(f"Training Accuracy of Decision Tree Classifier is
{accuracy_score(y_train, dtc.predict(x_train))}")
print(f"Test Accuracy of Decision Tree Classifier is {dtc_acc} \n")
```

```
print(f"Confusion Matrix :- \n{confusion_matrix(y_test,
dtc.predict(x_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test,
dtc.predict(x_test))}")
```

```
from lightgbm import LGBMClassifier

lgbm = LGBMClassifier(learning_rate = 1)
lgbm.fit(x_train, y_train)

# accuracy score, confusion matrix and classification report of lgbm
classifier

lgbm_acc = accuracy_score(y_test, lgbm.predict(x_test))

print(f"Training Accuracy of LGBM Classifier is {accuracy_score(y_train,
lgbm.predict(x_train))}")
print(f"Test Accuracy of LGBM Classifier is {lgbm_acc} \n")

print(f"{confusion_matrix(y_test, lgbm.predict(x_test))}\n")
print(classification_report(y_test, lgbm.predict(x_test)))
```

```
from sklearn.ensemble import RandomForestClassifier

rd_clf = RandomForestClassifier(criterion = 'entropy', max_depth = 11,
max_features = 'auto', min_samples_leaf = 2, min_samples_split = 3,
n_estimators = 130)
rd_clf.fit(X_train, y_train)

# accuracy score, confusion matrix and classification report of random
forest

rd_clf_acc = accuracy_score(y_test, rd_clf.predict(X_test))

print(f"Training Accuracy of Random Forest Classifier is
{accuracy_score(y_train, rd_clf.predict(X_train))}")
```

```
print(f"Test Accuracy of Random Forest Classifier is {rd_clf_acc} \n")


models = pd.DataFrame({
    'Model' : [ 'KNN', 'Decision Tree Classifier', 'LGBM Classifier'],
    'Score' : [knn_acc, dtc_acc, lgbm_acc]})


models.sort_values(by = 'Score', ascending = False)
```

OUTPUT SCREENS

```
[1] import glob
    from keras.models import Sequential,load_model
    import numpy as np
    import pandas as pd
    from keras.layers import Dense
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import LabelEncoder,MinMaxScaler
    import matplotlib.pyplot as plt
    import keras as k
```

```
df = pd.read_csv('/chronic  kidney.csv')
df.head(5)
```

	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

```
[3] df.shape
```

```
(400, 26)
```

```
[4] columns_to_retain = ['sg','al','sc','hemo','pcv','wbcc','rbcc','htn','classification']
    df = df.drop( [col for col in df.columns if not col in columns_to_retain] , axis=1 )
    df = df.dropna(axis=0)
```

```
for column in df.columns:
    if df[column].dtype == np.number:
      continue
    df[column] = LabelEncoder().fit_transform( df[column] )
```

```
<ipython-input-5-aeba061634c7>:2: DeprecationWarning: Converting `np.inexact` or `np.floating` to a dtype is deprecated. The current result is `f
  if df[column].dtype == np.number:
<ipython-input-5-aeba061634c7>:2: DeprecationWarning: Converting `np.inexact` or `np.floating` to a dtype is deprecated. The current result is `f
  if df[column].dtype == np.number:
<ipython-input-5-aeba061634c7>:2: DeprecationWarning: Converting `np.inexact` or `np.floating` to a dtype is deprecated. The current result is `f
  if df[column].dtype == np.number:
```

```
df.head()
```

	sg	al	sc	hemo	pcv	htn	classification
0	1.020	1.0	1.2	15.4	28	1	0
1	1.020	4.0	0.8	11.3	22	0	0
2	1.010	2.0	1.8	9.6	15	0	0
3	1.005	4.0	3.8	11.2	16	1	0
4	1.010	2.0	1.4	11.6	19	0	0

```
[7] x = df.drop(['classification'],axis=1)
    y = df['classification']
```

```
[8] x_scaler = MinMaxScaler()
    x_scaler.fit(x)
    column_names = x.columns
    x[column_names] = x_scaler.transform(x)
```

```
[9] x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.4, shuffle=True)
```

```
history = model.fit(x_train, y_train, epochs=200, batch_size= x_train.shape[0])
```

```
Epoch 1/200
1/1 [==============================] - 1s 691ms/step - loss: 0.6931 - accuracy: 0.4128
Epoch 2/200
1/1 [==============================] - 0s 9ms/step - loss: 0.6855 - accuracy: 0.8198
Epoch 3/200
1/1 [==============================] - 0s 9ms/step - loss: 0.6780 - accuracy: 0.8198
Epoch 4/200
1/1 [==============================] - 0s 8ms/step - loss: 0.6706 - accuracy: 0.8256
Epoch 5/200
1/1 [==============================] - 0s 8ms/step - loss: 0.6634 - accuracy: 0.8256
Epoch 6/200
1/1 [==============================] - 0s 7ms/step - loss: 0.6564 - accuracy: 0.8256
Epoch 7/200
1/1 [==============================] - 0s 7ms/step - loss: 0.6495 - accuracy: 0.8372
Epoch 8/200
1/1 [==============================] - 0s 7ms/step - loss: 0.6427 - accuracy: 0.8430
Epoch 9/200
1/1 [==============================] - 0s 6ms/step - loss: 0.6361 - accuracy: 0.8488
Epoch 10/200
1/1 [==============================] - 0s 8ms/step - loss: 0.6296 - accuracy: 0.8547
Epoch 11/200
1/1 [==============================] - 0s 11ms/step - loss: 0.6231 - accuracy: 0.8547
Epoch 12/200
1/1 [==============================] - 0s 9ms/step - loss: 0.6167 - accuracy: 0.8547
Epoch 13/200
1/1 [==============================] - 0s 9ms/step - loss: 0.6104 - accuracy: 0.8605
Epoch 14/200
1/1 [==============================] - 0s 9ms/step - loss: 0.6041 - accuracy: 0.8605
```

✓ 5s completed at 1:11 PM

```
history = model.fit(x_train, y_train, epochs=200, batch_size= x_train.shape[0])
```

```
Epoch 15/200
1/1 [==============================] - 0s 10ms/step - loss: 0.5979 - accuracy: 0.8605
Epoch 16/200
1/1 [==============================] - 0s 10ms/step - loss: 0.5916 - accuracy: 0.8663
Epoch 17/200
1/1 [==============================] - 0s 13ms/step - loss: 0.5853 - accuracy: 0.8721
Epoch 18/200
1/1 [==============================] - 0s 9ms/step - loss: 0.5790 - accuracy: 0.8721
Epoch 19/200
1/1 [==============================] - 0s 12ms/step - loss: 0.5726 - accuracy: 0.8779
Epoch 20/200
1/1 [==============================] - 0s 9ms/step - loss: 0.5663 - accuracy: 0.8779
Epoch 21/200
1/1 [==============================] - 0s 10ms/step - loss: 0.5600 - accuracy: 0.8779
Epoch 22/200
1/1 [==============================] - 0s 10ms/step - loss: 0.5537 - accuracy: 0.8779
Epoch 23/200
1/1 [==============================] - 0s 11ms/step - loss: 0.5473 - accuracy: 0.8779
Epoch 24/200
1/1 [==============================] - 0s 11ms/step - loss: 0.5410 - accuracy: 0.8779
Epoch 25/200
1/1 [==============================] - 0s 9ms/step - loss: 0.5346 - accuracy: 0.8779
Epoch 26/200
1/1 [==============================] - 0s 8ms/step - loss: 0.5282 - accuracy: 0.8779
Epoch 27/200
1/1 [==============================] - 0s 8ms/step - loss: 0.5217 - accuracy: 0.8779
Epoch 28/200
1/1 [==============================] - 0s 10ms/step - loss: 0.5153 - accuracy: 0.8779
```

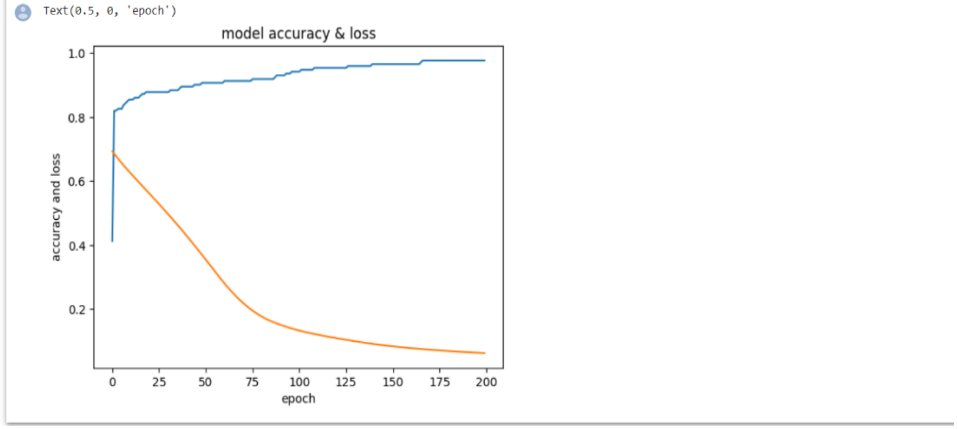
```
history = model.fit(x_train, y_train, epochs=200, batch_size= x_train.shape[0])
```

```
Epoch 29/200
1/1 [==============================] - 0s 8ms/step - loss: 0.5088 - accuracy: 0.8779
Epoch 30/200
1/1 [==============================] - 0s 8ms/step - loss: 0.5022 - accuracy: 0.8779
Epoch 31/200
1/1 [==============================] - 0s 8ms/step - loss: 0.4956 - accuracy: 0.8779
Epoch 32/200
1/1 [==============================] - 0s 8ms/step - loss: 0.4890 - accuracy: 0.8837
Epoch 33/200
1/1 [==============================] - 0s 6ms/step - loss: 0.4824 - accuracy: 0.8837
Epoch 34/200
1/1 [==============================] - 0s 8ms/step - loss: 0.4757 - accuracy: 0.8837
Epoch 35/200
1/1 [==============================] - 0s 8ms/step - loss: 0.4689 - accuracy: 0.8837
Epoch 36/200
1/1 [==============================] - 0s 11ms/step - loss: 0.4621 - accuracy: 0.8837
Epoch 37/200
1/1 [==============================] - 0s 8ms/step - loss: 0.4552 - accuracy: 0.8895
Epoch 38/200
1/1 [==============================] - 0s 9ms/step - loss: 0.4483 - accuracy: 0.8953
Epoch 39/200
1/1 [==============================] - 0s 9ms/step - loss: 0.4413 - accuracy: 0.8953
Epoch 40/200
1/1 [==============================] - 0s 8ms/step - loss: 0.4343 - accuracy: 0.8953
Epoch 41/200
1/1 [==============================] - 0s 8ms/step - loss: 0.4272 - accuracy: 0.8953
Epoch 42/200
1/1 [==============================] - 0s 8ms/step - loss: 0.4202 - accuracy: 0.8953
```

✓ 5s completed at 1:11 PM

```
plt.ylabel('accuracy and loss')
plt.xlabel('epoch')
```

```
Text(0.5, 0, 'epoch')
```




```
print('shape of traning data:',x_train.shape)
print('shape of test data:',x_test.shape)
```

```
shape of traning data: (172, 6)
shape of test data: (115, 6)
```

```
[16] pred = model.predict(x_test)
pred = [1 if y>=0.5 else 0 for y in pred]
pred
print('Original : {0}'.format(", ".join(str(x) for x in y_test)))
print('Predicted : {0}'.format(", ".join(str(x) for x in pred)))
```

```
4/4 [==============================] - 0s 2ms/step
Original : 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0,
Predicted : 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0,
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

knn = KNeighborsClassifier()
knn.fit(x_train, y_train)
```

```
KNeighborsClassifier
```

```
[18] knn_acc = accuracy_score(y_test, knn.predict(x_test))
```

```
[19] print(f"Training Accuracy of KNN is {accuracy_score(y_train, knn.predict(x_train))}")
print(f"Test Accuracy of KNN is {knn_acc} \n")
```

```
Training Accuracy of KNN is 0.9883720930232558
Test Accuracy of KNN is 0.9826086956521739
```

```
print(f"Confusion Matrix :- \n{confusion_matrix(y_test, knn.predict(x_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, knn.predict(x_test))}")
```

```
Confusion Matrix :-
[[62  2]
 [ 0 51]]

Classification Report :-
              precision    recall  f1-score   support

           0       1.00      0.97      0.98        64
           1       0.96      1.00      0.98        51

    accuracy                           0.98       115
   macro avg       0.98      0.98      0.98       115
weighted avg       0.98      0.98      0.98       115
```

```
print(f"Confusion Matrix :- \n{confusion_matrix(y_test, dtc.predict(x_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, dtc.predict(x_test))}")
```

```
Confusion Matrix :-
[[64  0]
 [ 0 51]]

Classification Report :-
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        64
           1       1.00      1.00      1.00        51

    accuracy                           1.00       115
   macro avg       1.00      1.00      1.00       115
weighted avg       1.00      1.00      1.00       115
```

```
from lightgbm import LGBMClassifier

lgbm = LGBMClassifier(learning_rate = 1)
lgbm.fit(x_train, y_train)
```

```
▸ LGBMClassifier
```

```
[28] # accuracy score, confusion matrix and classification report of lgbm classifier

     lgbm_acc = accuracy_score(y_test, lgbm.predict(x_test))
```

```
[29] print(f"Training Accuracy of LGBM Classifier is {accuracy_score(y_train, lgbm.predict(x_train))}")
     print(f"Test Accuracy of LGBM Classifier is {lgbm_acc} \n")
```

```
Training Accuracy of LGBM Classifier is 1.0
Test Accuracy of LGBM Classifier is 1.0
```

```
print(f"{confusion_matrix(y_test, lgbm.predict(x_test))}\n")
print(classification_report(y_test, lgbm.predict(x_test)))
```

```
[[64  0]
 [ 0 51]]

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        64
           1       1.00      1.00      1.00        51

    accuracy                           1.00       115
   macro avg       1.00      1.00      1.00       115
weighted avg       1.00      1.00      1.00       115
```

```
from sklearn.ensemble import RandomForestClassifier

rd_clf = RandomForestClassifier(criterion = 'entropy', max_depth = 11, max_features = 'auto', min_samples_leaf = 2, min_samples_split = 3, n_est
rd_clf.fit(x_train, y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/ensemble/_forest.py:424: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will
  warn(
```

```
▸ RandomForestClassifier
```

+ Code + Text

```
[32] # accuracy score, confusion matrix and classification report of random forest

     rd_clf_acc = accuracy_score(y_test, rd_clf.predict(x_test))
```

```
print(f"Training Accuracy of Random Forest Classifier is {accuracy_score(y_train, rd_clf.predict(x_train))}")
print(f"Test Accuracy of Random Forest Classifier is {rd_clf_acc} \n")
```

```
Training Accuracy of Random Forest Classifier is 1.0
Test Accuracy of Random Forest Classifier is 1.0
```

```
[ ] print(f"Confusion Matrix :- \n{confusion_matrix(y_test, dtc.predict(x_test))}\n")
    print(f"Classification Report :- \n {classification_report(y_test, rd_clf.predict(x_test))}")
```

```
[34] print(f"Confusion Matrix :- \n{confusion_matrix(y_test, dtc.predict(x_test))}\n")
     print(f"Classification Report :- \n {classification_report(y_test, rd_clf.predict(x_test))}")
```

```
Confusion Matrix :-
[[64  0]
 [ 0 51]]

Classification Report :-
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        64
           1       1.00      1.00      1.00        51

    accuracy                           1.00       115
   macro avg       1.00      1.00      1.00       115
weighted avg       1.00      1.00      1.00       115
```

```
models = pd.DataFrame({
    'Model' : [ 'KNN', 'Decision Tree Classifier', 'LGBM Classifier', 'Random Forest Classifier'],
    'Score' : [knn_acc, dtc_acc, lgbm_acc, rd_clf_acc]
})
```

```
models.sort_values(by = 'Score', ascending = False)
```

	Model	Score
2	LGBM Classifier	1.000000
3	Random Forest Classifier	1.000000
0	KNN	0.991304
1	Decision Tree Classifier	0.973913

SYSTEM TESTING

SYSTEM TESTING

5.1 Test Objective

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

TYPES OF TESTING

Unit Testing

Unit testing is a method of testing individual units or components of a software application. It is typically done by developers and is used to ensure that the individual units of the software are working as intended. Unit tests are usually automated and are designed to test specific parts of the code, such as a particular function or method. Unit testing is done at the lowest level of the software development process, where individual units of code are tested in isolation.

- It helps to identify bugs early in the development process, before they become more difficult and expensive to fix.
- It helps to ensure that changes to the code do not introduce new bugs.
- It makes the code more modular and easier to understand and maintain.
- It helps to improve the overall quality and reliability of the software

Integration Testing

Integration testing is a method of testing how different units or components of a software application interact with each other. It is used to identify and resolve any issues that may arise when different units of the software are combined. Integration testing is typically done after unit testing and before functional testing, and is used to verify that the different units of the software work together as intended.

- It helps to identify and resolve issues that may arise when different units of the software are combined.
- It helps to ensure that the different units of the software work together as intended.
- It helps to improve the overall reliability and stability of the software.

- It’s important to keep in mind that Integration testing is essential for complex systems where different components are integrated together.
- As with unit testing, integration testing is only one aspect of software testing and it should be used in combination with other types of testing such as unit testing, functional testing, and acceptance testing to ensure that the software meets the needs of its users.

Regression Testing

Regression testing is a method of testing that is used to ensure that changes made to the software do not introduce new bugs or cause existing functionality to break. It is typically done after changes have been made to the code, such as bug fixes or new features, and is used to verify that the software still works as intended.

- It helps to ensure that changes made to the software do not introduce new bugs or cause existing functionality to break.
- It helps to ensure that the software continues to work as intended after changes have been made.
- It helps to improve the overall reliability and stability of the software.
- It’s important to keep in mind that regression testing is an ongoing process that should be done throughout the software development lifecycle to ensure that the software continues to work as intended.
- It should be automated as much as possible to save time and resources. Additionally, it’s important to have a well-defined regression test suite that covers

Smoke Testing

This test is done to make sure that the software under testing is ready or stable for further testing . It is called a smoke test as the testing of an initial pass is done to check if it did not catch the fire or smoke in the initial switch on.
The term 'smoke testing” likely originated in the plumbing industry.

Alpha Testing

Alpha testing is the initial phase of validating whether a new product will perform as expected. Alpha tests are carried out early in the development process by internal staff and are followed up with beta tests, in which a sampling of the intended audience actually tries the product out.

Beta Testing

The beta test is conducted at one or more customer sites by the end-user of the software. This version is released for a limited number of users for testing in a real-time environment

System Testing

This software is tested such that it works fine for the different operating systems. It is covered under the black box testing technique. In this, we just focus on the required input and output without focusing on internal working. In this, we have security testing, recovery testing, stress testing, and performance testing.

Performance Testing

It is designed to test the run-time performance of software within the context of an integrated system. It is used to test the speed and effectiveness of the program. It is also called load testing. In it we check, what is the performance of the system in the given load.

Acceptance Testing

Acceptance testing is done by the customers to check whether the delivered products perform the desired tasks or not, as stated in requirements.

- Improved software quality and reliability
- Early identification and fixing of defects
- Improved customer satisfaction
- Increased stakeholder confidence
- Reduced maintenance costs

White Box Testing

While Box Testing is a testing in which the software tester has knowledge of the inner workings, structure and language of the system, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings,structure or language of the module being tested. Black box tests, as most other kinds of tests,must be written from a definitive source document,

such as specification or requirements document, such as specification or requirements document. It is a testing in which the softwareunder test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

5.2 Test Activities

Training and Testing Training:

The training data is the biggest (in -size) subset of the original dataset, which is used to train or fit the machine learning model. Firstly, the training data is fed to the ML algorithms, which lets them learn how to make predictions for the given task.

The training data varies depending on whether we are using Supervised Learning or Unsupervised Learning Algorithms.

For **Unsupervised learning**, the training data contains unlabelled data points, i.e., inputs are not tagged with the corresponding outputs. Models are required to find the patterns from the given training datasets in order to make predictions.

On the other hand, for supervised learning, the training data contains labels in order to train the model and make predictions.

The type of training data that we provide to the model is highly responsible for the model's accuracy and prediction ability. It means that the better the quality of the training data, the better will be the performance of the model. Training data is approximately more than or equal to 60% of the total data for an ML project.

Testing:

Once we train the model with the training dataset, it's time to test the model with the test dataset. This dataset evaluates the performance of the model and ensures that the model can generalize well with the new or unseen dataset. **The test dataset is another subset of original data, which is independent of the training dataset**. However, it has some similar types of features and class probability distribution and uses it as a benchmark for model evaluation once the model training is completed. Test data is a well-organized dataset that contains data for each type of scenario for a given problem that the model would be facing when used in the real world. Usually, the test dataset is approximately 20-25% of the total original data for an ML project.

At this stage, we can also check and compare the testing accuracy with the training accuracy, which means how accurate our model is with the test dataset against the training dataset. If the accuracy of the model on training data is greater than that on testing data, then the model is said to have overfitting.

The testing data should:

- Represent or part of the original dataset.
- It should be large enough to give meaningful predictions.

1. KNN Training Data

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

knn = KNeighborsClassifier()
knn.fit(x_train, y_train)

knn_acc = accuracy_score(y_test, knn.predict(x_test))

print(f"Training Accuracy of KNN is {accuracy_score(y_train, knn.predict(x_train))}")
print(f"Test Accuracy of KNN is {knn_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, knn.predict(x_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, knn.predict(x_test))}")
```

```
Training Accuracy of KNN is 0.9956331877729258
Test Accuracy of KNN is 0.9827586206896551
```

KNN Testing Data

```
Test Accuracy of KNN is 0.9416666666666667

Confusion Matrix :-
[[69  0  2]
 [ 1  0  0]
 [ 4  0 44]]

Classification Report :-
              precision    recall  f1-score   support

           0       0.93      0.97      0.95        71
           1       0.00      0.00      0.00         1
           2       0.96      0.92      0.94        48

    accuracy                           0.94       120
   macro avg       0.63      0.63      0.63       120
weighted avg       0.93      0.94      0.94       120

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and F-score are ill-defined and being
  _warn_prf(average, modifier, msg_start, len(result))
```

Here we have the precision and Recall, upon using confusion matrix we have received the accuracy of 94 percentage.

2. Decision Tree Training Data

```
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier()
dtc.fit(x_train, y_train)

# accuracy score, confusion matrix and classification report of decision tree

dtc_acc = accuracy_score(y_test, dtc.predict(x_test))

print(f"Training Accuracy of Decision Tree Classifier is {accuracy_score(y_train, dtc.predict(x_train))}")
print(f"Test Accuracy of Decision Tree Classifier is {dtc_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, dtc.predict(x_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, dtc.predict(x_test))}")
```

```
Training Accuracy of Decision Tree Classifier is 1.0
```

Decision Tree Testing Data

```
print(f"Confusion Matrix :- \n{confusion_matrix(y_test, dtc.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, dtc.predict(X_test))}")
```

```
Training Accuracy of Decision Tree Classifier is 1.0
Test Accuracy of Decision Tree Classifier is 0.9833333333333333

Confusion Matrix :-
[[71  0  0]
 [ 1  0  0]
 [ 1  0 47]]

Classification Report :-
              precision    recall  f1-score   support

           0       0.97      1.00      0.99        71
           1       0.00      0.00      0.00         1
           2       1.00      0.98      0.99        48

    accuracy                           0.98       120
   macro avg       0.66      0.66      0.66       120
weighted avg       0.98      0.98      0.98       120
```

Here we have the precision and Recall, upon using confusion matrix we have received the accuracy of 98 percentage.

3. Random Forest Training Data

```
from sklearn.ensemble import RandomForestClassifier

rd_clf = RandomForestClassifier(criterion = 'entropy', max_depth = 11, max_features = 'auto', min_samples_leaf = 2, min_samples_split = 3, n_estimator
rd_clf.fit(x_train, y_train)

# accuracy score, confusion matrix and classification report of random forest

rd_clf_acc = accuracy_score(y_test, rd_clf.predict(x_test))

print(f"Training Accuracy of Random Forest Classifier is {accuracy_score(y_train, rd_clf.predict(x_train))}")
print(f"Test Accuracy of Random Forest Classifier is {rd_clf_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, rd_clf.predict(x_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, rd_clf.predict(x_test))}")
```

```
Training Accuracy of Random Forest Classifier is 1.0
```

Random Forest Testing Data

```
[ ] print(f"Confusion Matrix :- \n{confusion_matrix(y_test, rd_clf.predict(X_test))}\n")
    print(f"Classification Report :- \n {classification_report(y_test, rd_clf.predict(X_test))}")

Training Accuracy of Random Forest Classifier is 0.9964285714285714
Test Accuracy of Random Forest Classifier is 0.9833333333333333

Confusion Matrix :-
[[71  0  0]
 [ 1  0  0]
 [ 1  0 47]]

Classification Report :-
              precision    recall  f1-score   support

           0       0.97      1.00      0.99        71
           1       0.00      0.00      0.00         1
           2       1.00      0.98      0.99        48

    accuracy                           0.98       120
   macro avg       0.66      0.66      0.66       120
weighted avg       0.98      0.98      0.98       120
```

Here we have the precision and Recall, upon using confusion matrix we have received the accuracy of 98 percentage.

4. LGBM Training Data

```
from lightgbm import LGBMClassifier

lgbm = LGBMClassifier(learning_rate = 1)
lgbm.fit(x_train, y_train)

# accuracy score, confusion matrix and classification report of lgbm classifier

lgbm_acc = accuracy_score(y_test, lgbm.predict(x_test))

print(f"Training Accuracy of LGBM Classifier is {accuracy_score(y_train, lgbm.predict(x_train))}")
print(f"Test Accuracy of LGBM Classifier is {lgbm_acc} \n")

print(f"{confusion_matrix(y_test, lgbm.predict(x_test))}\n")
print(classification_report(y_test, lgbm.predict(x_test)))
```

LGBM Testing Data

```
print(f"Test Accuracy of LGBM Classifier is {lgbm_acc} \n")

print(f"{confusion_matrix(y_test, lgbm.predict(X_test))}\n")
print(classification_report(y_test, lgbm.predict(X_test)))
```

```
Training Accuracy of LGBM Classifier is 1.0
Test Accuracy of LGBM Classifier is 0.9833333333333333

[[71  0  0]
 [ 1  0  0]
 [ 1  0 47]]

              precision    recall  f1-score   support

           0       0.97      1.00      0.99        71
           1       0.00      0.00      0.00         1
           2       1.00      0.98      0.99        48

    accuracy                           0.98       120
   macro avg       0.66      0.66      0.66       120
weighted avg       0.98      0.98      0.98       120
```

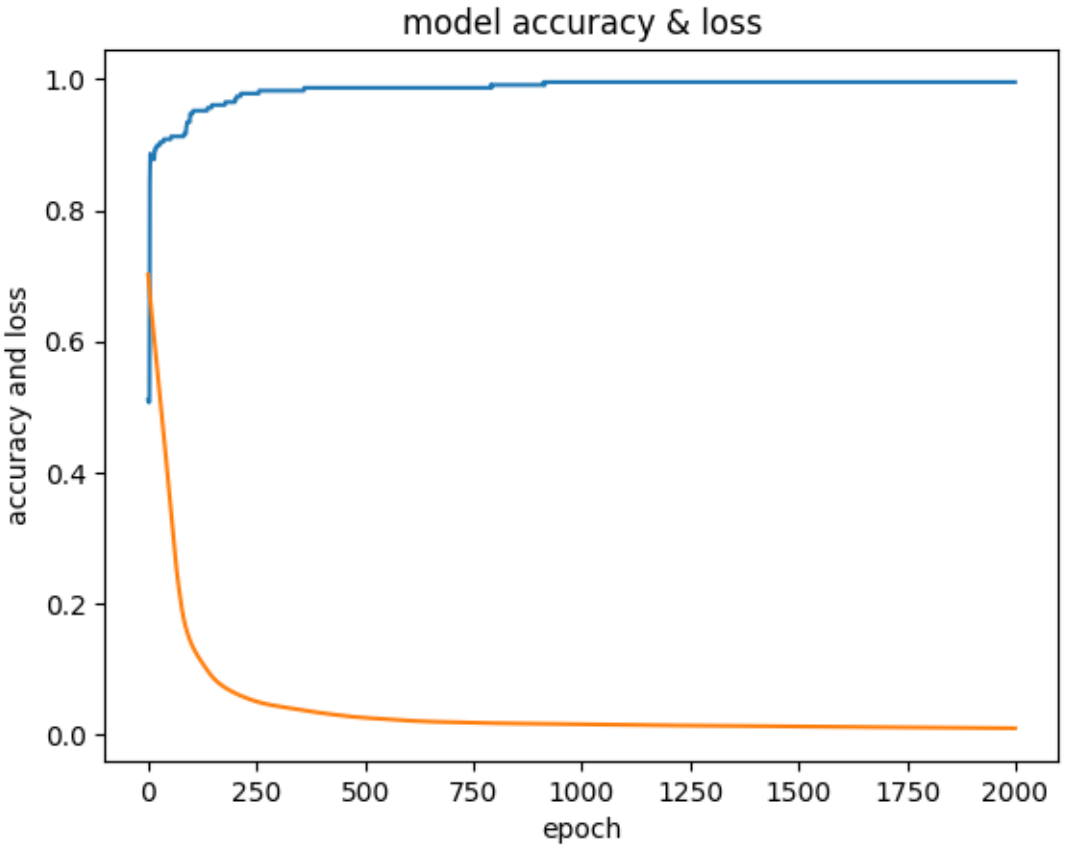
Here we have the precision and Recall, upon using confusion matrix we have received the accuracy of 98 percentage.

Testing Strategy

A strategy in this system testing is integrating the systems test cases along with the design tech- niques into a well planned series of steps with the results in the successful construction of software. This testing strategy should work along with the test planning, test execution, test case design, and the data collection resultant and evaluation. To accommodate a low-level test is a strategy for

software testing. Software testing is a critical element of software quality assurance which represents the ultimate review of specification design and coding. Interesting anomalies from the software are being to represented by using Testing. Thus, the period of testing is performed for the proposed system and there by is ready for user acceptance testing.

5.3 Result Analysis



Epoch:

An epoch is when all the training data is used at once and is defined as the total number of iterations of all the training data in one cycle for training the machine learning model.

RESULTS AND DISCUSSIONS

RESULTS AND DISCUSSIONS

As we have mentioned the earlier that it is better to search for an alternative approach with promis- ing results and we are going to do that. So, in this study, we are going to compare four very commonly known machine learning algorithms and we are going to state the best among them using a case study on Chronic Kidney disease Prediction. As stated, the four machine learning algorithms are: Random Forest Decision Tree LGBM K- Near Neighbour (KNN) Why we are going with only four algorithms is that we don't want to overload and confuse the learning procedure by including more machine learning algorithms. So, we thought that four would be an optimal choice and the machine would work just fine.

Using confusion matrix we received the accuracy of all four methodologies such as K-Nearest Neighbour, Random Forest, Decision Tree and Light Gradient Boosted Machine. We receive the accuracy of KNN is 94 percentage and Random Forest, Decision Tree, Light Gradient Boosted Machine accuracy are 98 percentage. Compared to Light Gradient Boosted Machine, Decision Tree and Ran- dom Forest we receive less accuracy in K-Nearest Neighbour.

CONCLUSION

CONCLUSION

CKD means that the kidney does not work and cannot correctly filter the blood. About 10 percent of the population for a worldwide suffering from (CKD), and millions of die each year because of they couldn't get the affordable treatment, with the number in- creasing in the elderly. In 2010, a study was conducted by International Society of Numerology (ISN) on global burden disease, they reported that CKD has been raised an important cause of the mortality worldwide with the number of deaths increasing by 82.3 percent in the last two decades. Also, the number of patients reaching End-Stage Renal Disease (ESRD) is increasing, which requires kidney transplantation or dialysis to save the patient 's lives. Machine learning models can effectively aid clinicians to achieve this goal due to their fast and accurate recognition performance. By using these ML algorithms, it is easy to identify the kidney disease in early stage. Random Forest, Light Gradient Boosted Machine, these two are best algorithms.

FUTURE ENHANCEMENTS

FUTURE ENHANCEMENTS

As a complementary solution, we could build a mobile application, to make it available to everyone to detect whether they are CKD or not and as of now if we have a correct data set we could use it to predict many more diseases and also if the application collaborates with a highly reputed company then we can allow making more advancements to the existing model. There would be disadvantages because of we need to have higher graphic card compatibility to run the model and the solution for that would be, the involvement of a framework such as Kara's or Tensor- Flow at the back end.

REFERENCES

REFERENCES

[1] https://www.researchgate.net/publication/344319206_Chronic_Kidney_Disease_Prediction_Usi ng_Machine_Learning_Methods

[2] J. Aljaaf et al, ”Early prediction of chronic kidney disease using machine learning supported by predictive analytic,” in 2020 IEEE Congress on Evolutionary Computation (CEC), 2020, .

[3] S. Vijayarani et al, ”KIDNEY DISEASE PREDICTION USING SVM AND ANN ALGORITHMS”. International Journal of Computing and Business Research (IJCBR), vol. 6,no. 2,(2019).

[4] T Shaikhina, Torgyn, et al. ”Decision tree and random forest models for outcome prediction in anti body incompatible kidney transplantation.” Biomedical Signal Processing and Control (2019).

[5] Jaymin Patel, Prof.Tejal Upadhyay, Dr. Samir Patel,et al., “Heart Disease Prediction Using Ma- chine learning and Data Mining Technique”, International Journal Of Computer Science Com- munication, Vol. 7, No. 1, pp.129 – 137. (2019)

[6] T Shaikhina, Torgyn, et al. ”Decision tree and random forest models for outcome prediction in anti body incompatible kidney transplantation.” Biomedical Signal Processing and Control (2019).

[7] C.T. Tran, et al., Multiple Imputation and Ensemble Learning for Classification with Incomplete Data,Springer International Publishing, pp. 401-415. (2020)

[8] J. Xiao et al, ”Comparison and development of machine learning tools in the prediction of chronic kidney disease progression,” Journal of Transnational Medicine, vol. 17, (1), pp. 119, 2019.

[9] L. R. Nair, S. D. Shetty, and S. D. Shetty, “Applying spark based machine learning model on streaming big data for health status prediction,” *Computers & Electrical Engineering*, vol. 65, pp. 393–399, 2018.

[10] World Health Organization, Public Health Agency of Canada, and Canada. Public Health Agency of Canada, *Preventing Chronic Diseases: A Vital Investment*, World Health Organization, Geneva, Switzerland, 2005.

[11] B. Bikbov, N. Perico, and G. Remuzzi, “Disparities in chronic kidney disease prevalence among males and females in 195 countries: analysis of the global burden of disease 2016 study,” *Nephron*, vol. 139, no. 4, pp. 313–318, 2018.

[12] K. Disease, “Improving global outcomes (kdigo) transplant work group. kdigo clinical practice guideline for the care of kidney transplant

recipients," *American Journal of Transplantation*, vol. 9, no. 3, pp. S1–S155, 2009.

[13] L. Deng and X. Li, "Machine learning paradigms for speech recognition: an overview," *IEEE Transactions on Audio Speech and Language Processing*, vol. 21, no. 5, pp. 1060–1089, 2013.

[14] M. Q. Huang, J. Ninić, and Q. B. Zhang, "Bim, machine learning and computer vision techniques in underground construction: current status and future perspectives," *Tunnelling and Underground Space Technology*, vol. 108, Article ID 103677, 2021.

[15] P. Oza, P. Sharma, and S. Patel, "Machine learning applications for computer-aided medical diagnostics," in *Proceedings of the Second International Conference on Computing, Communications, and Cyber-Security*, Springer, New York, NY, USA, 2021.

[16] T. Bikmukhametov and J. Jäschke, "Combining machine learning and process engineering physics towards enhanced accuracy and explainability of data-driven models," *Computers & Chemical Engineering*, vol. 138, Article ID 106834, 2020.

[17] R. J. Palma-Mendoza, D. Rodriguez, and L. D. Marcos, "Distributed relieff-based feature selection in spark," *Knowledge and Information Systems*, vol. 57, no. 1, pp. 1–20, 2018.

[18] M. Nassar, H. Safa, A. A. Mutawa, A. Helal, and I. Gaba, "Chi squared feature selection over Apache spark," in *Proceedings of the 23rd International Database Applications & Engineering Symposium*, pp. 1–5, Athens Greece, June 2019.

[19] A. Charleonnan, T. Fufaung, T. Niyomwong, W. Chokchueypattanakit, S. Suwannawach, and N. Ninchawee, "Predictive analytics for chronic kidney disease using machine learning techniques," in *Proceedings of the 2016 management and innovation technology international conference (MITicon)*, Bang-San, Thailand, October 2016.

[20] P. Chittora, S. Chaurasia, P. Chakrabarti et al., "Prediction of chronic kidney disease-a machine learning perspective," *IEEE Access*, vol. 9, no. 17312, p. 17334, 2021.