Liver disease detection using machine learning

techniques

**Abstract:**

Around a million deaths occur due to liver diseases globally. There are several traditional methods to diagnose liver diseases, but they are expensive. Early prediction of liver disease would benefit all individuals prone to liver diseases by providing early treatment. As technology is growing in health care, machine learning significantly affects health care for predicting conditions at early stages. This study finds how accurate machine learning is in predicting liver disease. This present study introduces the liver disease prediction (LDP) method in predicting liver disease that can be utilised by health professionals, stakeholders, students and researchers. Five algorithms, namely Support Vector Machine (SVM), Random ForestClassifier, K-Nearest Neighbors (K-NN),Logistic Regression,DecisionTreeClassifier,ArtificialNeural Network(ANN), are selected. The accuracy is compared to uncover the best classification method for predicting liver disease using Python. From the results, K-NN obtains the best accuracy with 0.72%, and the autoencoder network achieved 0.73% accuracy, which is above the acceptable level of accuracy and can be considered for liver disease prediction.

**Keywords:** Liver disease, Machine learning, Prediction, Data analytics, Healthcare

**Introduction:**

The liver is one of the most critical organs of the human body. It plays an essential role in the body’s function. Primary purposes include removing toxins from the body, fighting against infections, and balancing the hormones and secretion of bile juice (Devikanniga et al., 2020). If these functions are not performed by the liver correctly, it will result in several complications and liver diseases. Therefore if a virus infects the liver or chemicals that injure the liver are consumed, or the immune system’s dysfunction occurs, severe damage to the liver or malfunctioning may happen, which ultimately might cause death (Nahar & Ara, 2018).

Liver disease is one of the most chronic and threatening diseases globally that can cause various side effects if not treated early (Dutta et al., 2022). According to World Health Organization (WHO) report in 2018, the number of deaths due to liver diseases is around one million and ranked 11th in the world with a critical number of fatalities (World Total Deaths, n.d.). As the symptoms of liver diseases cannot be visible until the condition becomes chronic, it is challenging and daunting for medical health professionals to identify liver disease at its early stages (Devikanniga et al., 2020). In addition, the traditional testing methods like sonography, MRI scans and CT scans that are available for detecting liver diseases are expensive and harmful with numerous side effects (Joloudari et al., 2019). Thus, a significant constraint found by health care workers is to predict liver diseases at an early stage, at minimal cost and at the same time provide a better health care system to treat liver diseases. Severe liver diseases include problems with indigestion, dry mouth, pain in the abdomen, skin colour turning yellow, numbness, memory loss and fainting problems (Shaheamlung et al., 2020). Unnoticed at the initial stages, these symptoms are only visible when the disease turns chronic. However, even though the liver is partially infected, it can still function (Devikanniga et al., 2020).

Diagnosis of liver diseases can be divided into three stages i.e., the first stage is liver inflammation, the second is liver scarring (cirrhosis), and the final stage is liver cancer or failure. Since these scenarios are present in liver disease, early prediction is significant to provide better health for New Zealanders. If liver disease is diagnosed early, there will be a chance of early treatment and control of deaths due to liver diseases (Arbain & Balakrishnan, 2019). But when the liver fails to function, few treatments are available except liver transplantation (Shaheamlung et al., 2020), which is very expensive, particularly in New Zealand (Hepatitis C, 2021). Apparently, in New Zealand, 35 - 40% of the population are not diagnosed with Hepatitis C at the early stages because of the asymptomatic behaviour of liver disease. Unfortunately, most of these individuals do not know the risks linked to liver disease. Due to the asymptomatic behaviour and higher costs of liver disease treatment, it is essential to prevent or diagnose early for better treatment.

With advancements in biomedical sciences, the health care system has significantly improved by predicting disease using machine learning techniques (El-Shafeiy et al., 2018). Machine Learning algorithms are one of the potential solutions to this problem due to their handling large amounts of data and employing different approaches like classification, association and clustering, which benefits in realistic arbitration of disease prediction (Naseem et al., 2020).

There are different learning techniques in ML methods, one of which is supervised learning. Supervised learning techniques use labelled data and map the input and output data. These supervised learning methods are widely used for prediction and classification (Osisanwo et al., 2017). Supervised learning techniques would be appropriate as this research predicts whether the patient has liver disease or has no liver disease. The supervised learning methods used in this study are Support Vector Machine (SVM) (Boser et al., 1992), Naïve Bayes (Decision Treeclassifier)(McCallum & Nigam, 1998), K-Nearest Neighbors (K-NN) (Fix & Hodges, 1951), LogisticRegression,RandomForestClassifier, and Artificial neural network(ANN). The main objective of this research is to compare the accuracies using five supervised learning algorithms, i.e., SVM, Naïve Bayes, K-NN,Log,RF,DT,ANN for predicting whether the patient has liver disease or not. This study also proposes the liver disease prediction (LDP) method to help relevant stakeholders pursue an effective healthcare strategy.

Moreover, this paper examines the techniques that indicate liver diseases at an acceptable level of accuracy and determines the methods that produce the best accuracy. This study selects a single data set of liver patients with five supervised learning techniques that are applied to that data set in R. The accuracy results from other learning techniques are also used to compare the best algorithm for predicting liver diseases. The stakeholders, including doctors, researchers, lab technicians, or

companies dealing with healthcare improvements, can use these results to predict liver diseases at a lower cost and provide better health care in liver treatment.

# **Literature Review:**

In a study conducted by Vijayarani and Dhayanand (2015), the liver disease prediction applied the SVM and Naïve Bayes (using MATLAB 2013 software) on the Indian Liver Patient Records dataset having 583 instances and 11 attributes, with accuracies of 79.66% (SVM) and 61.28% (Naïve Bayes). In their findings, the time taken to execute SVM was 3210ms, almost two times the time taken by Naïve Bayes (i.e., 1670ms), without preprocessing missing values. In addition to the accuracies, they found that SVM had better performance than Naïve Bayes.

Auxilia (2018) made an accurate prediction for liver disease using different ML methods, including SVM, Random Forest, Decision Trees, Artificial Intelligence and Naïve Bayes. The research was conducted using R on the Indian Liver Patient Records dataset, with 583 instances and 11 attributes. The accuracies were obtained from SVM (77%), Random Forest (77%), Decision Trees (81%), Artificial Intelligence (71%), and Naïve Bayes (37%), with the highest accuracy from the Decision Trees algorithm, and least with Naïve Bayes.

Wu et al. (2019) did a prediction analysis on patients having Fatty Liver Disease (FLD). The research collected 700 patient records from New Taipei Hospital, which had screening tests for fatty liver disease; out of 700 patients, 577 records were considered depending on the patient’s age and sufficient data. Of those 577 patients, 377 had fatty liver disease, and the remaining had no fatty liver disease. The dataset contains patient health details of age, gender, systolic and diastolic blood pressure, abdominal girth, glucose level, triglyceride, HDL-C, SGOT-AST, and SGPT-ALT. Synthetic Minority Over-Sampling Technique (SMOTE) was applied at the data preprocessing stage, and normalisation was done. Four ML algorithms, namely Random Forest, Naïve Bayes, Artificial Neural Network and Logistic Regression with 3, 5, and 10-fold cross-validation, were applied in the next step. In addition to the accuracies, the area under the receiver operating curve for all the algorithms was observed. Random Forest had given the best accuracy with all the cross-validations from all the results.

Singh et al. (2020) focused their research on predicting liver disease using different classification methods with feature selection and implementing software for easy prediction. The study was conducted on the Indian Liver Patient Records dataset. Some attributes were removed during the feature selection phase using the Correlation-based Feature Selection Subset Evaluator with the Greedy Stepwise search method in WEKA. Only five (5) attributes were selected through this method: Total Bilirubin, Direct Bilirubin, Alkaline Phosphatase, Alamine Aminotransferase, and Aspartate Aminotransferase. With this, six different classification methods were applied: Logistic Regression, Naïve Bayes, Sequential Minimal Optimization (SMO), Random Forest, Instant based Classification (IBk), and Logistic Regression has provided the highest accuracy with 74.36%. The least accuracy was produced by Naïve Bayes (55.9%).

Most of the past research concentrated on just the analysis but not the preprocessing part for this Indian Liver Patient Records dataset. So, this research bridges the gap by considering preprocessing as a significant stage in data analysis. Moreover, several other algorithms are also applied in this research.

# **Research Methodology:**

The proposed liver disease prediction (LDP) method used in this research is based on SEMMA (Santos & Azevedo, 2005), which stands for Sample, Explore, Modify, Model, and Assess (Azevedo & Santos, 2008).



**Figure 1:** SEMMA lifecycle (Mariscal et al., 2010)

SEMMA lifecycle (see Figure 1) is a simple process to understand, aiming to get the solutions quickly for data mining problems and determine business goals. This methodology has developed by an institute named SAS Institute.



**Figure 2.** The liver disease prediction (LDP) method

*Source: Developed for this study*

The LDP method involved in this research are Sample, Explore, Modify, Data preprocessing, Model, Assess and Results. Along with these steps from the SEMMA lifecycle, two more steps, Data preprocessing and Results, are added to this research process. These steps (see Figure 2) include:

## **Sample**

The first stage in the LDP method proposed in this study is ‘Sample’. After deciding on the topic for the study, the first step is data collection. It is referred to as data collection and considering the part of data useful for the study (Azevedo & Santos, 2008). So, the data sets related to liver diseases are searched on different platforms named UCI repository and Kaggle. The suitable dataset is found from the platform Kaggle, a binary classification dataset that determines whether the patient has liver disease. After observing the credibility of the dataset, this dataset named ‘Indian Liver Patient Records’ is selected.

## **Explore**

The second stage is ‘Explore’. Exploring the data stage involves data understanding. This exploration stage also comprises finding the surprising trends and patterns present in the data to generate new ideas (Azevedo & Santos, 2008). In this study, exploring the data is at two stages. One is the data exploration on the background of liver disease. The other stage is exploring the dataset, which shows the details regarding the attributes present and how these attributes are correlated with each other and how these input attributes are correlated with the output attribute are studied. In addition, missing values are also identified. This analysis is performed using R.

## **Modify**

The third stage is ‘Modify’. Modify refers to data transformation (Azevedo & Santos, 2008). In this study, the attributes in the dataset are not in the same format, and the attribute’s data type restricts the analysis to be done on the attribute. So, some of the features having the data type integers are converted into numerical, which makes all the attributes have the same numerical data type and makes the analysis be done efficiently.

## **Data Preprocessing**

The fourth stage is ‘Data preprocessing’. This data preprocessing refers to cleaning and preparing the data for modelling (Azevedo & Santos, 2008). This data preprocessing involves replacing the missing values and balancing the dataset as the class distribution of the dataset is imbalanced. This balancing is done using the Random Over Sampling Example (ROSE) (Menardi & Torelli, 2014).

## **Model**

The fifth stage is the ‘Model’. The modelling stage means applying the selected techniques or the algorithms to the data (Azevedo & Santos, 2008). The five algorithms, SVM, Naïve Bayes, LDA, CART and K-NN, are applied.

## **Assess**

Assess stage, which is the sixth stage, involves assessing the data by deciding whether the data produced from modelling techniques are reliable and accurate. This stage also evaluates how well the algorithms performed on the data (Azevedo & Santos, 2008).

## **Results**

The seventh stage of the proposed LDP method is ‘Results’. The results stage involves presenting the results after assessing the data. All the results of accuracies and confusion matrix metrics will be described.

# **Performance Analysis**

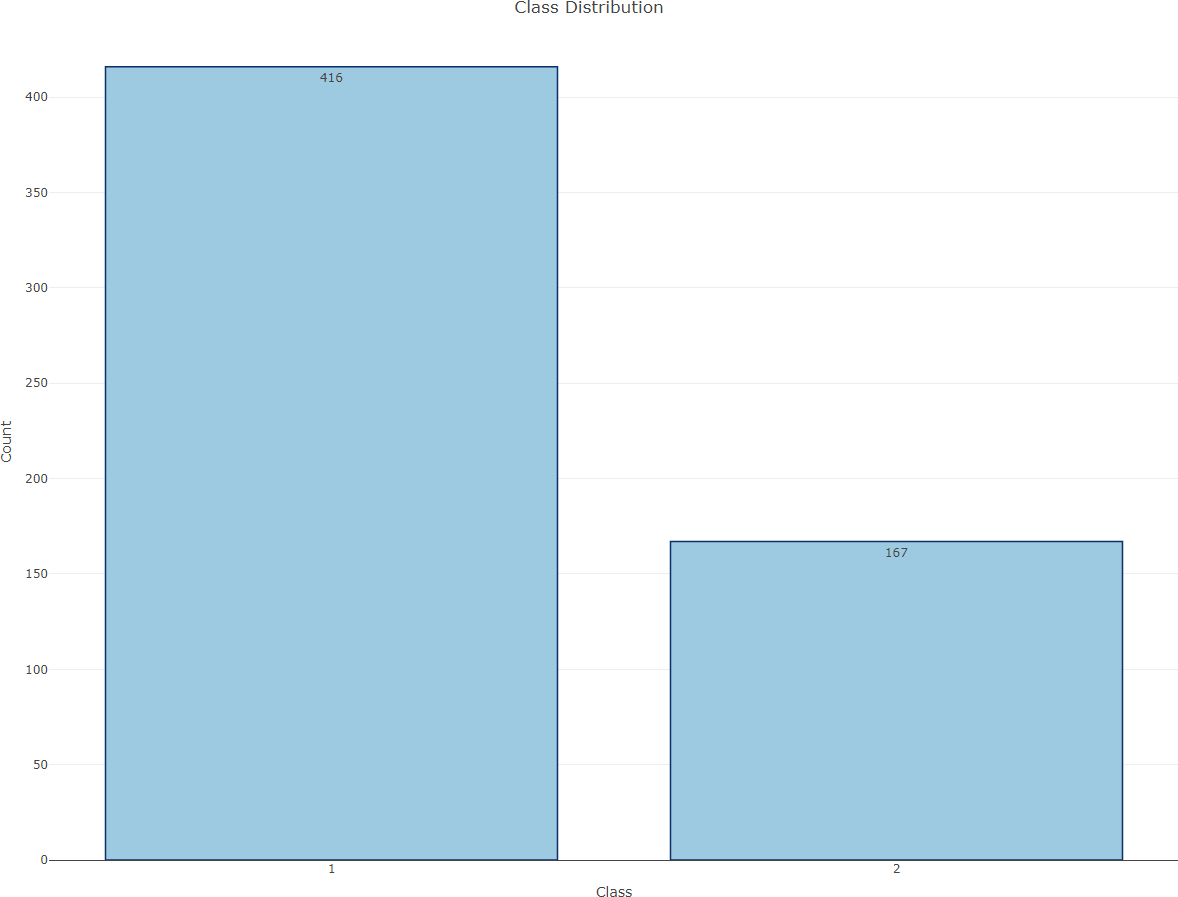
## **Descriptive Analysis**

The dataset selected for this study is the liver disease dataset. This dataset, named ‘Indian Liver Patient Records’, is obtained from Kaggle. The data from the dataset is collected from the North-East part of Andhra Pradesh, India (*Indian Liver Patient Records*, n.d.). It is a binary classification dataset predicting whether the patient has liver disease or not. As stated in Table 1, the dataset contains 583 instances and 11 attributes. Of those 11 attributes, one of the attributes is class which denotes whether the patient has liver disease.

|  |  |  |  |
| --- | --- | --- | --- |
| **Dataset Name** | # Of Instances | #Of Attributes | #Of Class |
| **Indian Liver Patient Records** | 583 | 11 | 2 |

**Table 1**: *General details of the dataset*

Of these 583 patient records, 416 have liver disease, and 167 have no liver disease. The metadata of the dataset is indicated in Table 2. Figure 3 shows the binary classification dataset, having class two values of ‘1’ and ‘2’, where ‘1’ denotes that the patients have liver disease and ‘2’ denotes those patients do not.



**Figure 3:** Class distribution

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number | Feature | Definition | Type | Explanation |
| 1 | Age | Age of the patient | Integer | Examination Results |
| 2 | Gender | Sex of the patient | Nominal | Male, Female |
| 3 | Total Bilirubin | Total Bilirubin in mg/dL | Numeric | Examination Results |
| 4 | Direct Bilirubin | Conjugated Bilirubin in mg/dL | Numeric | Examination Results |
| 5 | Alkaline Phosphotase | Alkaline Phosphotase in IU/L | Integer | Examination Results |
| 6 | Alamine  Aminotransferase | Alamine Aminotransferase in  IU/L | Integer | Examination Results |
| 7 | Aspartate  Aminotransferase | Aspartate Aminotransferase in  IU/L | Integer | Examination Results |
| 8 | Total Proteins | Total Proteins g/dL | Numeric | Examination Results |
| 9 | Albumin | Albumin in g/dL | Numeric | Examination Results |
| 10 | Albumin and  Globulin Ratio | Albumin & Globulin Ratio | Numeric | Examination Results |
| 11 | Dataset | A patient has liver disease or not | Nominal | 1. Has Liver Disease 2. No Liver Disease |
|  |  |  |  |  |

## **Data Modification**

In the dataset, the data types of all the attributes are not the same; therefore, to maintain consistency and better analysis, the attributes having integer data types are converted into numerical ones. Four attributes have the integer data type: Age, Alkaline Phosphatase, Alamine Aminotransferase, and Aspartate Aminotransferase (see Table 3). These are converted to numerical.

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Definition | Data type | Converted data type |
| Age | Age of the patient | Integer | Numerical |
| Alkaline Phosphatase | Alkaline Phosphatase in IU/L | Integer | Numerical |
| Alamine Aminotransferase | Alamine Aminotransferase in  IU/L | Integer | Numerical |
| Aspartate Aminotransferase | Aspartate Aminotransferase in  IU/L | Integer | Numerical |

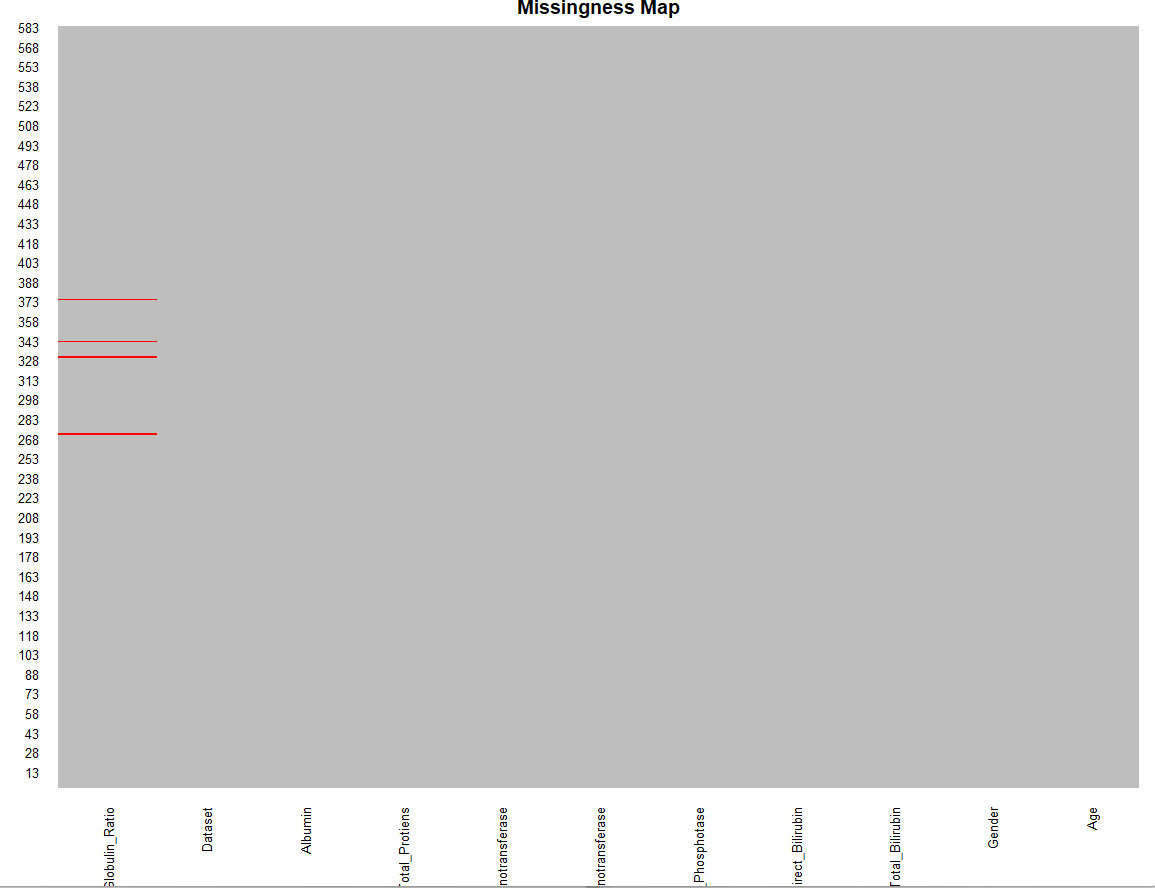
**Table 3**: *Details of the attributes after the data modification*

## **Data Preprocessing**

Data preprocessing is an important segment of data analysis. This study requires data processing for missing values and balancing the dataset.

## **Replacing Missing Values**

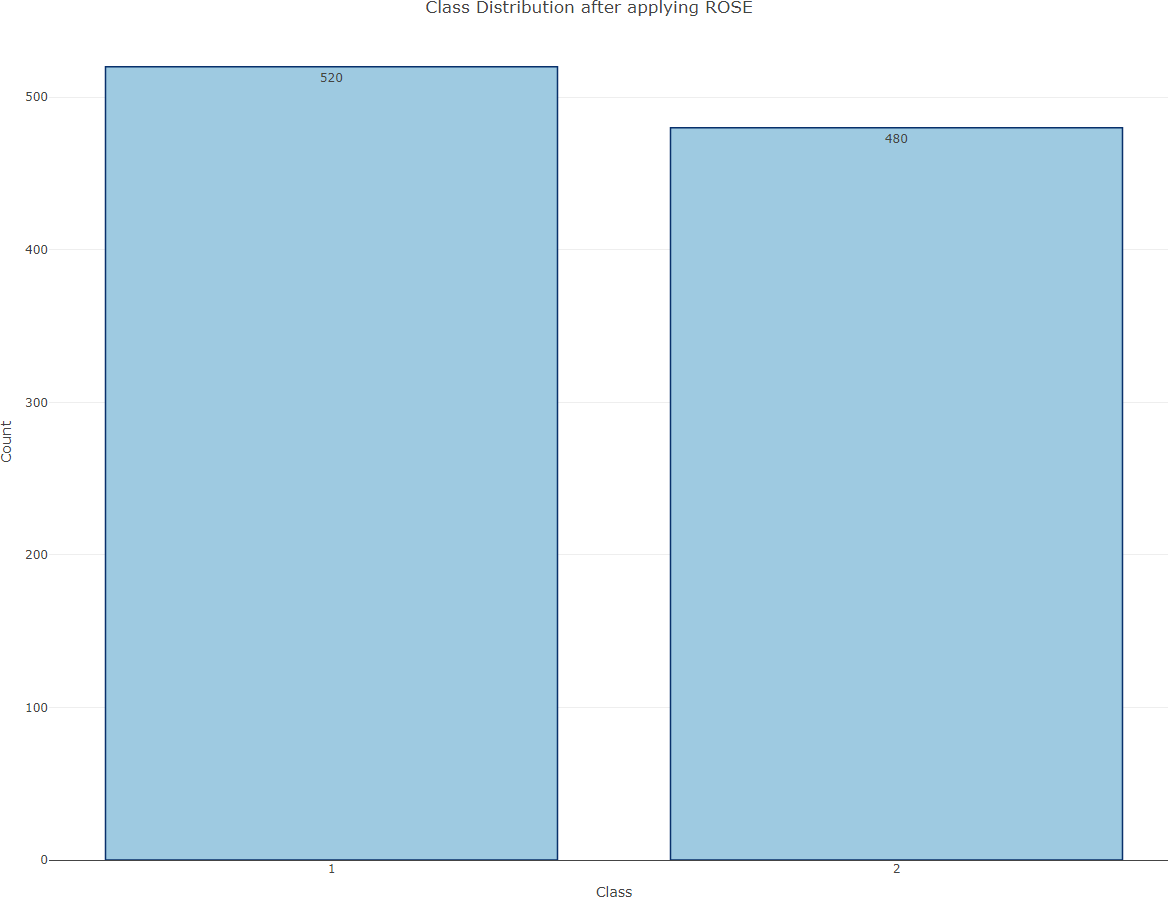
the mean value of the attribute. Figure 4 illustrates that the albumin and globulin ratio attribute has the missing value at 268, 328, 343 and 373 instances.



**Figure 4:** Missing value plot

## 4.3.2 **Balancing the Dataset**

The class distribution of the dataset is imbalanced, with 416 having liver disease and 167 without liver disease. This class distribution is imbalanced and balanced by applying ROSE using R. ROSE is a bootstrap method that produces balanced synthetic samples to balance the data (Lunardon et al., 2014). The reason for choosing ROSE is that the dataset is small, and the most reliable information might be lost if undersampling is conducted. The other reason for considering ROSE is that it generates samples similar to the rare class samples, which is also a consideration for an effective method for getting reliable accuracy from the balanced dataset as this study’s main aim is the accuracy metrics of the algorithms (Lunardon et al., 2014). After applying the ROSE method on the class attribute, the sample generated is 520 having liver disease denoted by ‘1’ and 480 without liver disease represented by ‘2’ in Figure 5.



**Figure 5:** Class distribution after applying ROSE

# **Experimental Environment**

## **Experimental settings and parameter settings**

The data analysis of applying algorithms and finding the accuracy is done using R with version 1.4.1717. The investigation starts by loading the dataset into R and then modifying and preprocessing the data. Then five different algorithms, namely Support Vector Machine (SVM), Linear Discriminant Analysis (LDA), Naïve Bayes, K-Nearest Neighbours (K-NN), and Classification and Regression Trees (CART), are applied to the dataset. For all the algorithms, the seed is set to 7 and a cross-fold validation of 10. For K-NN, the k value is set to 3.

## **Classifiers**

* + 1. **Support Vector Machine (SVM)**

SVM is a supervised machine learning technique that strives to search for a hyperplane with maximum margin. Then it separates the linearly independent variables onto either side of the hyperplane and classifies the data (Devikanniga et al., 2020).

## **Naïve Bayes**

Naïve Bayes is one of the basic probabilistic classifiers which classifies the specific class with the given tuple. It is categorised by hypothesising that every attribute has a solitary effect on the class attribute by not depending on other attribute values (Passi & Pandey, 2018).

## **3.2.3.K-Nearest Neighbours (K-NN)**

K-NN is one of the most straightforward and efficient classification methods. This method predicts the test data point label with the superior class of its *k* most identical points of training data (Zhang et al., 2017).

**3.2.4.Random Forest model:-**

A function named RandomForestClassifier is imported and train and test data are passed as the parameters. Inside the function, RandomForestClassifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in a new variable.

**3.2.5.Logistic Regression model:-**

A function named Logistic Regression is imported and train and test data are passed as the parameters. Inside the function, Logistic Regression algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix.

**3.2.6.ANN model:-**

Building and training an Artificial Neural Network (ANN) using the Keras library with TensorFlow as the backend. The ANN is initialised as an instance of the Sequential class, which is a linear stack of layers. Then, the input layer and two hidden layers are added to the model using the Dense class, where the number of units and activation function are specified. The output layer is also added using the Dense class with a sigmoid activation function. The model is then compiled withTensorFlow as the backend. The ANN is initialised as an instance of the Sequential class, which is a linear stack of layers. Then, the input layer and two hidden layers are added to the model using the Dense class, where the number of units and activation function are specified. The output layer is also added using the Dense class with a sigmoid activation function. The model is then compiled with the Adam optimizer, binary cross-entropyloss function, and accuracy metric. Finally, the model is fit to the training data with a batch size of 100, 20% validation split, and 100 epochs

# Experimental Results Analysis

After applying different algorithms to the liver disease data set, accuracy

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Classification methods | SVM | Logistic  Regression |  | K-NN | Random forest classifier | Decision tree  Classifier |
| Accuracy score | 0.7132867132867133 | 0.7352657342657343 |  | 0.7202797202797203 | 0.7272727272727273 | 0.6433566433566433 |
| Precision score | 0.7313432835820896 | 0.764227642276228 |  | 0.7560975609756098 | 0.7711864406779662 | 0.75 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**Table 4**: *Results of different experimented algorithms*

The results obtained from the experiment, except for two algorithms, SVM and LDA, the rest three algorithms gave an acceptable level of accuracy above 75%. Autoencoders (3 layered) achieved 92.1% (921 correctly classified instance) accuracy, with K-NN achieving an almost similar level of accuracy with correctly classified instances to 917. The lowest accuracy is for Naïve Bayes, 65.1%, with only 651 correctly classified instances.

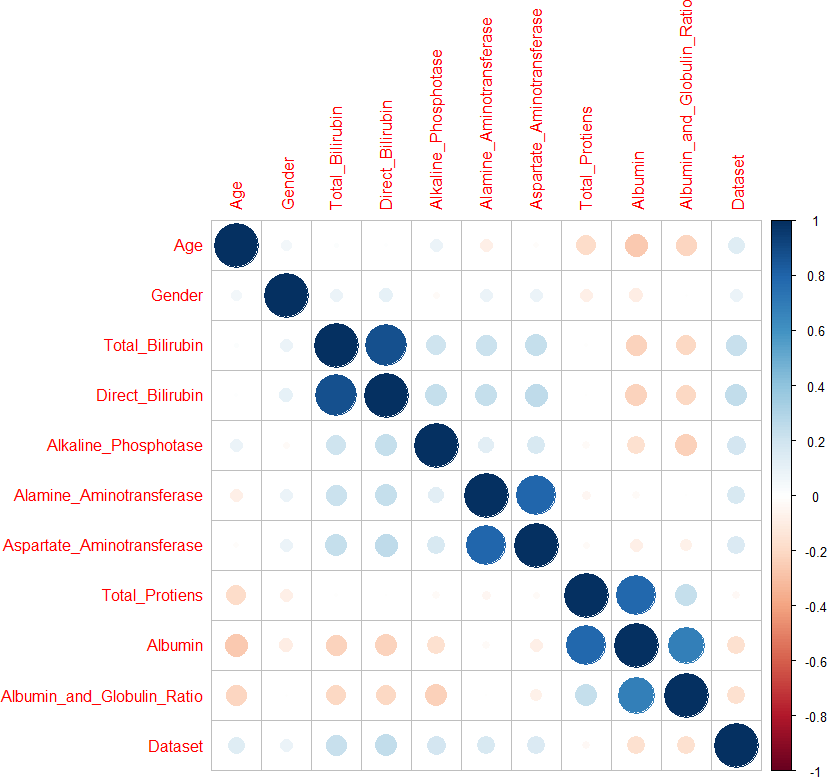
## **Correlation between the Attributes**

In the correlation plot, it can be observed that some attributes, namely total proteins, albumin and globulin ratio, and albumin, are not much likely correlated with the class attribute, and the remaining attributes are significantly correlated with the class attribute. This correlation plot is generated using Pearson Correlation in R to know how likely the attributes are correlated.

# **Discussions and Future Recommendations**

The proposed liver disease prediction (LDP) method has provided the right path for liver disease detection. From the results of this study, after balancing the dataset, SVM has 78.1%, and Naïve Bayes has 65.1%. This balancing of the dataset using ROSE significantly changes the accuracy compared to the accuracies produced by Auxilia (2018), which is 77% for SVM and 37% for Naïve Bayes.

Singh et al. (2020) also focused on the same dataset of liver patients with feature engineering done with WEKA. After feature engineering, only five attributes are selected for the analysis, and algorithms are applied. The common algorithm from this research and Singh et al. (2020) is Naïve Bayes and has an accuracy of 55.9% with only five attributes selected. By comparing that to the results of this study, Naïve Bayes has an accuracy of 65.1%. As shown in Figure 11, only three attributes are less likely correlated with class attributes, but the rest are correlated with the class attribute, affecting the accuracy. The attributes that are not correlated with class attributes can be removed, which gives the better performance of algorithms and maximised accuracy. So, from Singh et al. (2020) research, some relatable features are dissolved in feature engineering, impacting accuracy. Thus, if this research needs to be done differently, it can include some more instances for better prediction. As the given dataset has only 583 instances, they can be increased in number for a better prognosis. Along with increasing the instances, different attributes important to predict liver disease like triglycerides, urine copper, serum cholesterol, and serum glutamic-oxaloacetic transaminase (SGOT) could be added to improve the chances of liver disease prediction (Assegie et al., 2022).



**Figure 11:** Correlation between the attributes

#importing the libraries

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from matplotlib import rcParams

from scipy import stats

Read the Dataset

data = pd.read\_csv('indian\_liver\_patient.csv')

data

| **Age** | **Gender** | **Total\_Bilirubin** | **Direct\_Bilirubin** | **Alkaline\_Phosphotase** | **Alamine\_Aminotransferase** | **Aspartate\_Aminotransferase** | **Total\_Protiens** | **Albumin** | **Albumin\_and\_Globulin\_Ratio** | **Dataset** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 65 | Female | 0.7 | 0.1 | 187 | 16 | 18 | 6.8 | 3.3 | 0.90 | 1 |
| **1** | 62 | Male | 10.9 | 5.5 | 699 | 64 | 100 | 7.5 | 3.2 | 0.74 | 1 |
| **2** | 62 | Male | 7.3 | 4.1 | 490 | 60 | 68 | 7.0 | 3.3 | 0.89 | 1 |
| **3** | 58 | Male | 1.0 | 0.4 | 182 | 14 | 20 | 6.8 | 3.4 | 1.00 | 1 |
| **4** | 72 | Male | 3.9 | 2.0 | 195 | 27 | 59 | 7.3 | 2.4 | 0.40 | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **578** | 60 | Male | 0.5 | 0.1 | 500 | 20 | 34 | 5.9 | 1.6 | 0.37 | 2 |
| **579** | 40 | Male | 0.6 | 0.1 | 98 | 35 | 31 | 6.0 | 3.2 | 1.10 | 1 |
| **580** | 52 | Male | 0.8 | 0.2 | 245 | 48 | 49 | 6.4 | 3.2 | 1.00 | 1 |
| **581** | 31 | Male | 1.3 | 0.5 | 184 | 29 | 32 | 6.8 | 3.4 | 1.00 | 1 |
| **582** | 38 | Male | 1.0 | 0.3 | 216 | 21 | 24 | 7.3 | 4.4 | 1.50 | 2 |

583 rows × 11 columns

print(data.head())

Age Gender Total\_Bilirubin Direct\_Bilirubin Alkaline\_Phosphotase \

0 65 Female 0.7 0.1 187

1 62 Male 10.9 5.5 699

2 62 Male 7.3 4.1 490

3 58 Male 1.0 0.4 182

4 72 Male 3.9 2.0 195

Alamine\_Aminotransferase Aspartate\_Aminotransferase Total\_Protiens \

0 16 18 6.8

1 64 100 7.5

2 60 68 7.0

3 14 20 6.8

4 27 59 7.3

Albumin Albumin\_and\_Globulin\_Ratio Dataset

0 3.3 0.90 1

1 3.2 0.74 1

2 3.3 0.89 1

3 3.4 1.00 1

4 2.4 0.40 1

[ ]

print(data.shape)

print("no of rows:",data.shape[0])

print("no of columns:",data.shape[1])

print("total no of data:",data.size)

(583, 11)

no of rows: 583

no of columns: 11

total no of data: 6413

[ ]

data.columns



Index(['Age', 'Gender', 'Total\_Bilirubin', 'Direct\_Bilirubin',

'Alkaline\_Phosphotase', 'Alamine\_Aminotransferase',

'Aspartate\_Aminotransferase', 'Total\_Protiens', 'Albumin',

'Albumin\_and\_Globulin\_Ratio', 'Dataset'],

dtype='object')

Data Cleaning

data.duplicated()

0 False 1 False 2 False 3 False 4 False ... 578 False 579 False 580 False 581 False 582 False Length: 583, dtype: bool

data.duplicated().sum()

13

data=data.drop\_duplicates()

print(data.shape)

(570, 11)

Handling Missing Values

[ ] print(data.info())

print(data.info())

<class 'pandas.core.frame.DataFrame'>

Int64Index: 570 entries, 0 to 582

Data columns (total 11 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Age 570 non-null int64

1 Gender 570 non-null object

2 Total\_Bilirubin 570 non-null float64

3 Direct\_Bilirubin 570 non-null float64

4 Alkaline\_Phosphotase 570 non-null int64

5 Alamine\_Aminotransferase 570 non-null int64

6 Aspartate\_Aminotransferase 570 non-null int64

7 Total\_Protiens 570 non-null float64

8 Albumin 570 non-null float64

9 Albumin\_and\_Globulin\_Ratio 566 non-null float64

10 Dataset 570 non-null int64

dtypes: float64(5), int64(5), object(1)

memory usage: 53.4+ KB

None

CodeText

data.isnull().any()

Age False

Gender False

Total\_Bilirubin False

Direct\_Bilirubin False

Alkaline\_Phosphotase False

Alamine\_Aminotransferase False

Aspartate\_Aminotransferase False

Total\_Protiens False

Albumin False

Albumin\_and\_Globulin\_Ratio True Dataset False

dtype: bool

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

Age 0

Gender 0

Total\_Bilirubin 0

Direct\_Bilirubin 0

Alkaline\_Phosphotase 0

Alamine\_Aminotransferase 0

Aspartate\_Aminotransferase 0

Total\_Protiens 0

Albumin 0

Albumin\_and\_Globulin\_Ratio 4

Dataset 0

dtype: int64

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

4

<Axes: xlabel='Albumin\_and\_Globulin\_Ratio'>

filling null values

[ ] data['Albumin\_and\_Globulin\_Ratio'].mode()

data['Albumin\_and\_Globulin\_Ratio'].mode()

0 1.0

Name: Albumin\_and\_Globulin\_Ratio, dtype: float64

[ ] data['Albumin\_and\_Globulin\_Ratio'].median()

data['Albumin\_and\_Globulin\_Ratio'].median()

0.95

[ ] data['Albumin\_and\_Globulin\_Ratio'].mean()

data['Albumin\_and\_Globulin\_Ratio'].mean()

0.9480035335689044

[ ] data['Albumin\_and\_Globulin\_Ratio']=data['Albumin\_and\_Globulin\_Ratio'].fillna(data['Albumin\_and\_Globulin\_Ratio'].median())

data['Albumin\_and\_Globulin\_Ratio']=data['Albumin\_and\_Globulin\_Ratio'].fillna(data['Albumin\_and\_Globulin\_Ratio'].median())

[ ] data.isna().sum()

Gender 0

Total\_Bilirubin 0

Direct\_Bilirubin 0

Alkaline\_Phosphotase 0

Alamine\_Aminotransferase 0

Aspartate\_Aminotransferase 0

Total\_Protiens 0

Albumin 0

Albumin\_and\_Globulin\_Ratio 0

Dataset 0

dtype: int64

Male vs Female

[ ]import seaborn as sns

sns.countplot(data=data,x='Gender',label='count')

plt.show()

[ ]

Male,Female=data['Gender'].value\_counts()  
print('Number pf patients that are male:',Male)  
print('Number pf patients that are female:',Female)

Number pf patients that are male: 430

Number pf patients that are female: 140

handling categorical values

[ ]

from sklearn.preprocessing import LabelEncoder  
lc=LabelEncoder()  
data['Gender']=lc.fit\_transform(data['Gender'])  
data

data['Gender']=lc.fit\_transform(data['Gender'])

|  | **Age** | **Gender** | **Total\_Bilirubin** | **Direct\_Bilirubin** | **Alkaline\_Phosphotase** | **Alamine\_Aminotransferase** | **Aspartate\_Aminotransferase** | **Total\_Protiens** | **Albumin** | **Albumin\_and\_Globulin\_Ratio** | **Dataset** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 65 | 0 | 0.7 | 0.1 | 187 | 16 | 18 | 6.8 | 3.3 | 0.90 | 1 |
| **1** | 62 | 1 | 10.9 | 5.5 | 699 | 64 | 100 | 7.5 | 3.2 | 0.74 | 1 |
| **2** | 62 | 1 | 7.3 | 4.1 | 490 | 60 | 68 | 7.0 | 3.3 | 0.89 | 1 |
| **3** | 58 | 1 | 1.0 | 0.4 | 182 | 14 | 20 | 6.8 | 3.4 | 1.00 | 1 |
| **4** | 72 | 1 | 3.9 | 2.0 | 195 | 27 | 59 | 7.3 | 2.4 | 0.40 | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **578** | 60 | 1 | 0.5 | 0.1 | 500 | 20 | 34 | 5.9 | 1.6 | 0.37 | 2 |
| **579** | 40 | 1 | 0.6 | 0.1 | 98 | 35 | 31 | 6.0 | 3.2 | 1.10 | 1 |
| **580** | 52 | 1 | 0.8 | 0.2 | 245 | 48 | 49 | 6.4 | 3.2 | 1.00 | 1 |
| **581** | 31 | 1 | 1.3 | 0.5 | 184 | 29 | 32 | 6.8 | 3.4 | 1.00 | 1 |
| **582** | 38 | 1 | 1.0 | 0.3 | 216 | 21 | 24 | 7.3 | 4.4 | 1.50 | 2 |

570 rows × 11 columns

[ ]

def partition(x):  
  if x==2:  
    return 0  
  return 1  
  
data['Dataset']=data['Dataset'].map(partition)

data['Dataset']=data['Dataset'].map(partition)

[ ]

data['Dataset']

0 1

1 1

2 1

3 1

4 1

..

578 0

579 1

580 1

581 1

582 0

Name: Dataset, Length: 570, dtype: int64

[ ]

data.boxplot()

[ ]

data.boxplot(column="Age")  
plt.show()

[ ]

data.boxplot(column="Total\_Bilirubin")  
plt.show()

[ ]

def remove\_outlier(col):  
  sorted(col)  
  q1,q3=col.quantile([0.25,0.75])  
  IQR=q3-q1  
  lwr\_bound=q1-(1.5\*IQR)  
  upr\_bound=q3+(1.5\*IQR)  
  return lwr\_bound,upr\_bound

[ ]

low,high=remove\_outlier(data["Total\_Bilirubin"])

[ ]

data["Total\_Bilirubin"]=np.where(data["Total\_Bilirubin"]>high,high,data["Total\_Bilirubin"])

data["Total\_Bilirubin"]=np.where(data["Total\_Bilirubin"]>high,high,data["Total\_Bilirubin"])

[ ]

data["Total\_Bilirubin"]=np.where(data["Total\_Bilirubin"]<low,low,data["Total\_Bilirubin"])

data["Total\_Bilirubin"]=np.where(data["Total\_Bilirubin"]<low,low,data["Total\_Bilirubin"])

[ ]

data.boxplot(column="Total\_Bilirubin")

[ ]

data.boxplot(column="Direct\_Bilirubin")  
plt.show()

[ ]

def remove\_outlier(col):  
  sorted(col)  
  q1,q3=col.quantile([0.25,0.75])  
  IQR=q3-q1  
  lwr\_bound=q1-(1.5\*IQR)  
  upr\_bound=q3+(1.5\*IQR)  
  return lwr\_bound,upr\_bound

[ ]

low,high=remove\_outlier(data["Direct\_Bilirubin"])

[ ]

data["Direct\_Bilirubin"]=np.where(data["Direct\_Bilirubin"]>high,high,data["Direct\_Bilirubin"])

data["Direct\_Bilirubin"]=np.where(data["Direct\_Bilirubin"]>high,high,data["Direct\_Bilirubin"])

[ ]

data["Direct\_Bilirubin"]=np.where(data["Direct\_Bilirubin"]<low,low,data["Direct\_Bilirubin"])

data["Direct\_Bilirubin"]=np.where(data["Direct\_Bilirubin"]<low,low,data["Direct\_Bilirubin"])

[ ]

data.boxplot(column="Direct\_Bilirubin")  
plt.show()

[ ]

data.boxplot(column="Alkaline\_Phosphotase")  
plt.show()

[ ]

def remove\_outlier(col):  
  sorted(col)  
  q1,q3=col.quantile([0.25,0.75])  
  IQR=q3-q1  
  lwr\_bound=q1-(1.5\*IQR)  
  upr\_bound=q3+(1.5\*IQR)  
  return lwr\_bound,upr\_bound

[ ] low,high=remove\_outlier(data["Alkaline\_Phosphotase"])

[ ] data["Alkaline\_Phosphotase"]=np.where(data["Alkaline\_Phosphotase"]<high,high,data["Alkaline\_Phosphotase"])

data["Alkaline\_Phosphotase"]=np.where(data["Alkaline\_Phosphotase"]>high,high,data["Alkaline\_Phosphotase"])

[ ] data["Alkaline\_Phosphotase"]=np.where(data["Alkaline\_Phosphotase"]<low,low,data["Alkaline\_Phosphotase"])

data["Alkaline\_Phosphotase"]=np.where(data["Alkaline\_Phosphotase"]<low,low,data["Alkaline\_Phosphotase"])

[ ] data.boxplot(column="Alkaline\_Phosphotase")

plt.show()

[ ] data.boxplot(column="Alamine\_Aminotransferase")

plt.show()

[ ]

def remove\_outlier(col):  
  sorted(col)  
  q1,q3=col.quantile([0.25,0.75])  
  IQR=q3-q1  
  lwr\_bound=q1-(1.5\*IQR)  
  upr\_bound=q3+(1.5\*IQR)  
  return lwr\_bound,upr\_bound

[ ]

low,high=remove\_outlier(data["Alamine\_Aminotransferase"])

[ ]

data["Alamine\_Aminotransferase"]=np.where(data["Alamine\_Aminotransferase"]>high,high,data["Alamine\_Aminotransferase"])

data["Alamine\_Aminotransferase"]=np.where(data["Alamine\_Aminotransferase"]>high,high,data["Alamine\_Aminotransferase"])

[ ]

data["Alamine\_Aminotransferase"]=np.where(data["Alamine\_Aminotransferase"]<low,low,data["Alamine\_Aminotransferase"])

data["Alamine\_Aminotransferase"]=np.where(data["Alamine\_Aminotransferase"]<low,low,data["Alamine\_Aminotransferase"])

[ ]

data.boxplot(column="Alamine\_Aminotransferase")

[ ] data.boxplot(column="Aspartate\_Aminotransferase")  
plt.show()

[ ]

def remove\_outlier(col):  
  sorted(col)  
  q1,q3=col.quantile([0.25,0.75])  
  IQR=q3-q1  
  lwr\_bound=q1-(1.5\*IQR)  
  upr\_bound=q3+(1.5\*IQR)  
  return lwr\_bound,upr\_bound

[ ]

low,high=remove\_outlier(data["Aspartate\_Aminotransferase"])

[ ]

data["Aspartate\_Aminotransferase"]=np.where(data["Aspartate\_Aminotransferase"]>high,high,data["Aspartate\_Aminotransferase"])

data["Aspartate\_Aminotransferase"]=np.where(data["Aspartate\_Aminotransferase"]>high,high,data["Aspartate\_Aminotransferase"])

[ ]

data["Aspartate\_Aminotransferase"]=np.where(data["Aspartate\_Aminotransferase"]<low,low,data["Aspartate\_Aminotransferase"])

data["Aspartate\_Aminotransferase"]=np.where(data["Aspartate\_Aminotransferase"]<low,low,data["Aspartate\_Aminotransferase"])

[ ]

data.boxplot(column="Aspartate\_Aminotransferase")

[ ]

data.boxplot(column="Albumin")  
plt.show()

[ ]

data.boxplot(column="Total\_Protiens")  
plt.show()

[ ]

def remove\_outlier(col):  
  sorted(col)  
  q1,q3=col.quantile([0.25,0.75])  
  IQR=q3-q1  
  lwr\_bound=q1-(1.5\*IQR)  
  upr\_bound=q3+(1.5\*IQR)  
  return lwr\_bound,upr\_bound

[ ]

low,high=remove\_outlier(data["Total\_Protiens"])

[ ]

data["Total\_Protiens"]=np.where(data["Total\_Protiens"]>high,high,data["Total\_Protiens"])

data["Total\_Protiens"]=np.where(data["Total\_Protiens"]>high,high,data["Total\_Protiens"])

[ ]

data["Total\_Protiens"]=np.where(data["Total\_Protiens"]<low,low,data["Total\_Protiens"])

data["Total\_Protiens"]=np.where(data["Total\_Protiens"]<low,low,data["Total\_Protiens"])

[ ]

data.boxplot(column="Total\_Protiens")

[ ]

data.boxplot(column="Albumin\_and\_Globulin\_Ratio")  
plt.show()

[ ]

def remove\_outlier(col):  
  sorted(col)  
  q1,q3=col.quantile([0.25,0.75])  
  IQR=q3-q1  
  lwr\_bound=q1-(1.5\*IQR)  
  upr\_bound=q3+(1.5\*IQR)  
  return lwr\_bound,upr\_bound

[ ]

low,high=remove\_outlier(data["Albumin\_and\_Globulin\_Ratio"])

[ ]

data["Albumin\_and\_Globulin\_Ratio"]=np.where(data["Albumin\_and\_Globulin\_Ratio"]>high,high,data["Albumin\_and\_Globulin\_Ratio"])

data["Albumin\_and\_Globulin\_Ratio"]=np.where(data["Albumin\_and\_Globulin\_Ratio"]>high,high,data["Albumin\_and\_Globulin\_Ratio"])

[ ]

data["Albumin\_and\_Globulin\_Ratio"]=np.where(data["Albumin\_and\_Globulin\_Ratio"]<low,low,data["Albumin\_and\_Globulin\_Ratio"])

data["Albumin\_and\_Globulin\_Ratio"]=np.where(data["Albumin\_and\_Globulin\_Ratio"]<low,low,data["Albumin\_and\_Globulin\_Ratio"])

[ ]

data.boxplot(column="Albumin\_and\_Globulin\_Ratio")

[ ]

data.boxplot(column="Dataset")  
plt.show()

[ ]

data.plot.hist(subplots=True,grid=True)

[ ]

data.dtypes

Age int64

Gender int64

Total\_Bilirubin float64

Direct\_Bilirubin float64

Alkaline\_Phosphotase float64

Alamine\_Aminotransferase float64

Aspartate\_Aminotransferase float64

Total\_Protiens float64

Albumin float64

Albumin\_and\_Globulin\_Ratio float64

Dataset int64

dtype: object

[ ]

data.skew()

Age -0.046597

Gender -1.185073

Total\_Bilirubin 1.222655

Direct\_Bilirubin 1.252807

Alkaline\_Phosphotase 1.031079

Alamine\_Aminotransferase 1.081600

Aspartate\_Aminotransferase 1.198754

Total\_Protiens -0.228039

Albumin -0.060829

Albumin\_and\_Globulin\_Ratio 0.347884

Dataset -0.940320

dtype: float64

[ ]

data.mode()

[ ]

data.mean()

Age 44.849123

Gender 0.754386

Total\_Bilirubin 1.919298

Direct\_Bilirubin 0.880000

Alkaline\_Phosphotase 251.445614

Alamine\_Aminotransferase 47.535088

Aspartate\_Aminotransferase 64.283772

Total\_Protiens 6.500526

Albumin 3.148947

Albumin\_and\_Globulin\_Ratio 0.941842

Dataset 0.712281

dtype: float64

[ ]

data.median()

Age 45.00

Gender 1.00

Total\_Bilirubin 1.00

Direct\_Bilirubin 0.30

Alkaline\_Phosphotase 208.00

Alamine\_Aminotransferase 35.00

Aspartate\_Aminotransferase 41.00

Total\_Protiens 6.60

Albumin 3.10

Albumin\_and\_Globulin\_Ratio 0.95

Dataset 1.00

dtype: float64

[ ]

data.kurtosis()

Age -0.565802

Gender -0.597712

Total\_Bilirubin -0.065252

Direct\_Bilirubin 0.020904

Alkaline\_Phosphotase -0.064894

Alamine\_Aminotransferase -0.153063

Aspartate\_Aminotransferase 0.075977

Total\_Protiens -0.103618

Albumin -0.373112

Albumin\_and\_Globulin\_Ratio 0.006487

Dataset -1.119740

dtype: float64

[ ]

data.\_get\_numeric\_data()

Descriptive Statistical

[ ]

data.describe()

Visual Analysis

[ ]

sns.pairplot(data)

[ ]

corelation=data.corr()

Multivariate Analysis

[ ]

plt.figure(figsize=(10,7))  
sns.heatmap(corelation,xticklabels=corelation.columns,yticklabels=corelation.columns,annot=True)

Univariate Analysis

[ ]

sns.displot(data['Age'])

[ ]

sns.displot(data['Gender'])

[ ]

sns.displot(data['Total\_Bilirubin'])

[ ]

sns.displot(data['Direct\_Bilirubin'])

[ ]

sns.displot(data['Alkaline\_Phosphotase'])

[ ]

sns.displot(data['Alamine\_Aminotransferase'])

[ ]

sns.displot(data['Aspartate\_Aminotransferase'])

[ ]

sns.displot(data['Total\_Protiens'])

[ ]

sns.displot(data['Albumin'])

[ ]

sns.displot(data['Albumin\_and\_Globulin\_Ratio'])

Bivariate Analysis

[ ]

#boxplot  
sns.boxplot(x='Age',y='Gender',data=data)  
plt.show()

[ ]

#barplot  
sns.barplot(x='Age',y='Gender',data=data)  
plt.show()

[ ]

#stripplot  
sns.stripplot(x='Age',y='Gender',data=data)  
plt.show()

Data Preparation

[ ]

x=data.iloc[: , :-1].values  
y=data.iloc[: , -1].values

[ ]

#splitting data in to training data and test data  
from sklearn.model\_selection import train\_test\_split  
X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(x,y,test\_size=0.25,random\_state=42)

[ ]

#feature scaling  
from sklearn.preprocessing import StandardScaler  
sc=StandardScaler()  
X\_train=sc.fit\_transform(X\_train)  
X\_test=sc.transform(X\_test)

Machine Learning Models

[ ]

#Logistic Regression  
from sklearn.linear\_model import LogisticRegression  
log\_classifier=LogisticRegression(random\_state=0)  
log\_classifier.fit(X\_train,Y\_train)

[ ]

#predicting the output  
log\_y\_pred=log\_classifier.predict(X\_test)

[ ]

from sklearn.metrics import confusion\_matrix  
log\_cm=confusion\_matrix(Y\_test,log\_y\_pred)  
sns.heatmap(log\_cm,annot=True)

[ ]

from sklearn.metrics import accuracy\_score,precision\_score  
print(accuracy\_score(Y\_test,log\_y\_pred))  
print(precision\_score(Y\_test,log\_y\_pred))

0.7342657342657343

0.7642276422764228

[ ]

#knn algorithm  
X\_train.shape

(427, 10)

[ ]

from sklearn.neighbors import KNeighborsClassifier  
knn\_classifier=KNeighborsClassifier(n\_neighbors=21,metric='minkowski')  
knn\_classifier.fit(X\_train,Y\_train)

[ ]

knn\_y\_pred=knn\_classifier.predict(X\_test)

[ ]

from sklearn.metrics import confusion\_matrix  
knn\_cm=confusion\_matrix(Y\_test,log\_y\_pred)  
sns.heatmap(knn\_cm,annot=True)

[ ]

from sklearn.metrics import accuracy\_score,precision\_score  
print(accuracy\_score(Y\_test,knn\_y\_pred))  
print(precision\_score(Y\_test,knn\_y\_pred))

0.7202797202797203

0.7560975609756098

[ ]

#SVM  
from sklearn.svm import SVC  
svm\_classifier=SVC(kernel='rbf',random\_state=0)  
svm\_classifier.fit(X\_train,Y\_train)

[ ]

svm\_y\_pred=svm\_classifier.predict(X\_test)

[ ]

from sklearn.metrics import confusion\_matrix  
svm\_cm=confusion\_matrix(Y\_test,svm\_y\_pred)  
sns.heatmap(svm\_cm,annot=True)

[ ]

from sklearn.metrics import accuracy\_score,precision\_score  
print(accuracy\_score(Y\_test,svm\_y\_pred))  
print(precision\_score(Y\_test,svm\_y\_pred))

0.7132867132867133

0.7313432835820896

[ ]

#decision tree algorithm  
from sklearn.tree import DecisionTreeClassifier  
dt\_classifier=DecisionTreeClassifier()  
dt\_classifier.fit(X\_train,Y\_train)

[ ]

dt\_y\_pred=dt\_classifier.predict(X\_test)

[ ]

from sklearn.metrics import confusion\_matrix  
dt\_cm=confusion\_matrix(Y\_test,dt\_y\_pred)  
sns.heatmap(dt\_cm,annot=True)

[ ]

from sklearn.metrics import accuracy\_score,precision\_score  
print(accuracy\_score(Y\_test,dt\_y\_pred))  
print(precision\_score(Y\_test,dt\_y\_pred))

0.6433566433566433

0.75

[ ]

#randomforest classifier  
from sklearn.ensemble import RandomForestClassifier  
rf\_classifier=RandomForestClassifier()  
rf\_classifier.fit(X\_train,Y\_train)

[ ]

rf\_y\_pred=rf\_classifier.predict(X\_test)

[ ]

from sklearn.metrics import confusion\_matrix  
rf\_cm=confusion\_matrix(Y\_test,dt\_y\_pred)  
sns.heatmap(rf\_cm,annot=True)

[ ]

from sklearn.metrics import accuracy\_score,precision\_score  
print(accuracy\_score(Y\_test,rf\_y\_pred))  
print(precision\_score(Y\_test,rf\_y\_pred))

0.7272727272727273

0.7711864406779662

Artificial Neural Network

[ ]

import keras  
from keras.models import Sequential  
from keras.layers import Dense, Dropout

[ ]

#initialising the ANN  
classifier=Sequential()  
#Adding the input layer and the first hidden layer  
classifier.add(Dense(units=400,activation='relu',input\_dim=10))  
classifier.add(Dropout(rate=0.1))  
#Adding second hidden layer  
classifier.add(Dense(units=400,activation='relu'))  
classifier.add(Dropout(rate=0.1))  
#output layer  
classifier.add(Dense(units=1,activation='sigmoid'))

[ ]

classifier.compile(optimizer='adam',loss='binary\_crossentropy',metrics=['accuracy'])

[ ]

classifier.fit(X\_train,Y\_train,batch\_size=32,epochs=100)

Epoch 1/100

14/14 [==============================] - 1s 4ms/step - loss: 0.5596 - accuracy: 0.7026

Epoch 2/100

14/14 [==============================] - 0s 4ms/step - loss: 0.5047 - accuracy: 0.6979

Epoch 3/100

14/14 [==============================] - 0s 3ms/step - loss: 0.4869 - accuracy: 0.7424

Epoch 4/100

14/14 [==============================] - 0s 4ms/step - loss: 0.4788 - accuracy: 0.7354

Epoch 5/100

14/14 [==============================] - 0s 5ms/step - loss: 0.4743 - accuracy: 0.7424

Epoch 6/100

14/14 [==============================] - 0s 3ms/step - loss: 0.4661 - accuracy: 0.7447

Epoch 7/100

14/14 [==============================] - 0s 3ms/step - loss: 0.4571 - accuracy: 0.7564

Epoch 8/100

14/14 [==============================] - 0s 3ms/step - loss: 0.4502 - accuracy: 0.7658

Epoch 9/100

14/14 [==============================] - 0s 4ms/step - loss: 0.4440 - accuracy: 0.7588

Epoch 10/100

14/14 [==============================] - 0s 3ms/step - loss: 0.4454 - accuracy: 0.7494

Epoch 11/100

14/14 [==============================] - 0s 3ms/step - loss: 0.4315 - accuracy: 0.7892

Epoch 12/100

14/14 [==============================] - 0s 3ms/step - loss: 0.4285 - accuracy: 0.7775

Epoch 13/100

14/14 [==============================] - 0s 3ms/step - loss: 0.4169 - accuracy: 0.7775

Epoch 14/100

14/14 [==============================] - 0s 3ms/step - loss: 0.4203 - accuracy: 0.7635

Epoch 15/100

14/14 [==============================] - 0s 3ms/step - loss: 0.4083 - accuracy: 0.7963

Epoch 16/100

14/14 [==============================] - 0s 4ms/step - loss: 0.3987 - accuracy: 0.8056

Epoch 17/100

14/14 [==============================] - 0s 3ms/step - loss: 0.3871 - accuracy: 0.8173

Epoch 18/100

14/14 [==============================] - 0s 3ms/step - loss: 0.3819 - accuracy: 0.8244

Epoch 19/100

14/14 [==============================] - 0s 3ms/step - loss: 0.3669 - accuracy: 0.8056

Epoch 20/100

14/14 [==============================] - 0s 3ms/step - loss: 0.3727 - accuracy: 0.8197

Epoch 21/100

14/14 [==============================] - 0s 4ms/step - loss: 0.3501 - accuracy: 0.8337

Epoch 22/100

14/14 [==============================] - 0s 3ms/step - loss: 0.3418 - accuracy: 0.8337

Epoch 23/100

14/14 [==============================] - 0s 4ms/step - loss: 0.3296 - accuracy: 0.8361

Epoch 24/100

14/14 [==============================] - 0s 4ms/step - loss: 0.3472 - accuracy: 0.8314

Epoch 25/100

14/14 [==============================] - 0s 3ms/step - loss: 0.3505 - accuracy: 0.8197

Epoch 26/100

14/14 [==============================] - 0s 3ms/step - loss: 0.3275 - accuracy: 0.8431

Epoch 27/100

14/14 [==============================] - 0s 3ms/step - loss: 0.3117 - accuracy: 0.8501

Epoch 28/100

14/14 [==============================] - 0s 4ms/step - loss: 0.3038 - accuracy: 0.8618

Epoch 29/100

14/14 [==============================] - 0s 4ms/step - loss: 0.3051 - accuracy: 0.8618

Epoch 30/100

14/14 [==============================] - 0s 4ms/step - loss: 0.2892 - accuracy: 0.8665

Epoch 31/100

14/14 [==============================] - 0s 4ms/step - loss: 0.2834 - accuracy: 0.8712

Epoch 32/100

14/14 [==============================] - 0s 4ms/step - loss: 0.2806 - accuracy: 0.8689

Epoch 33/100

14/14 [==============================] - 0s 3ms/step - loss: 0.3119 - accuracy: 0.8454

Epoch 34/100

14/14 [==============================] - 0s 5ms/step - loss: 0.2614 - accuracy: 0.8852

Epoch 35/100

14/14 [==============================] - 0s 6ms/step - loss: 0.2714 - accuracy: 0.8759

Epoch 36/100

14/14 [==============================] - 0s 5ms/step - loss: 0.2866 - accuracy: 0.8665

Epoch 37/100

14/14 [==============================] - 0s 5ms/step - loss: 0.2545 - accuracy: 0.8993

Epoch 38/100

14/14 [==============================] - 0s 5ms/step - loss: 0.2551 - accuracy: 0.8806

Epoch 39/100

14/14 [==============================] - 0s 6ms/step - loss: 0.2440 - accuracy: 0.9110

Epoch 40/100

14/14 [==============================] - 0s 6ms/step - loss: 0.2441 - accuracy: 0.8993

Epoch 41/100

14/14 [==============================] - 0s 6ms/step - loss: 0.2396 - accuracy: 0.8899

Epoch 42/100

14/14 [==============================] - 0s 5ms/step - loss: 0.2128 - accuracy: 0.9180

Epoch 43/100

14/14 [==============================] - 0s 5ms/step - loss: 0.2347 - accuracy: 0.8852

Epoch 44/100

14/14 [==============================] - 0s 6ms/step - loss: 0.2198 - accuracy: 0.9157

Epoch 45/100

14/14 [==============================] - 0s 7ms/step - loss: 0.2135 - accuracy: 0.9204

Epoch 46/100

14/14 [==============================] - 0s 6ms/step - loss: 0.2252 - accuracy: 0.8993

Epoch 47/100

14/14 [==============================] - 0s 6ms/step - loss: 0.2339 - accuracy: 0.8876

Epoch 48/100

14/14 [==============================] - 0s 6ms/step - loss: 0.2245 - accuracy: 0.8876

Epoch 49/100

14/14 [==============================] - 0s 6ms/step - loss: 0.2058 - accuracy: 0.9133

Epoch 50/100

14/14 [==============================] - 0s 5ms/step - loss: 0.2062 - accuracy: 0.9133

Epoch 51/100

14/14 [==============================] - 0s 5ms/step - loss: 0.1813 - accuracy: 0.9204

Epoch 52/100

14/14 [==============================] - 0s 5ms/step - loss: 0.1880 - accuracy: 0.9227

Epoch 53/100

14/14 [==============================] - 0s 5ms/step - loss: 0.1928 - accuracy: 0.9204

Epoch 54/100

14/14 [==============================] - 0s 5ms/step - loss: 0.1709 - accuracy: 0.9321

Epoch 55/100

14/14 [==============================] - 0s 5ms/step - loss: 0.1804 - accuracy: 0.9321

Epoch 56/100

14/14 [==============================] - 0s 7ms/step - loss: 0.1698 - accuracy: 0.9274

Epoch 57/100

14/14 [==============================] - 0s 6ms/step - loss: 0.1524 - accuracy: 0.9415

Epoch 58/100

14/14 [==============================] - 0s 7ms/step - loss: 0.1424 - accuracy: 0.9625

Epoch 59/100

14/14 [==============================] - 0s 7ms/step - loss: 0.1697 - accuracy: 0.9415

Epoch 60/100

14/14 [==============================] - 0s 6ms/step - loss: 0.1705 - accuracy: 0.9344

Epoch 61/100

14/14 [==============================] - 0s 6ms/step - loss: 0.1704 - accuracy: 0.9344

Epoch 62/100

14/14 [==============================] - 0s 5ms/step - loss: 0.1443 - accuracy: 0.9532

Epoch 63/100

14/14 [==============================] - 0s 5ms/step - loss: 0.1646 - accuracy: 0.9391

Epoch 64/100

14/14 [==============================] - 0s 5ms/step - loss: 0.1627 - accuracy: 0.9344

Epoch 65/100

14/14 [==============================] - 0s 5ms/step - loss: 0.1562 - accuracy: 0.9461

Epoch 66/100

14/14 [==============================] - 0s 5ms/step - loss: 0.1432 - accuracy: 0.9368

Epoch 67/100

14/14 [==============================] - 0s 5ms/step - loss: 0.1340 - accuracy: 0.9461

Epoch 68/100

14/14 [==============================] - 0s 6ms/step - loss: 0.1325 - accuracy: 0.9602

Epoch 69/100

14/14 [==============================] - 0s 6ms/step - loss: 0.1531 - accuracy: 0.9508

Epoch 70/100

14/14 [==============================] - 0s 5ms/step - loss: 0.1206 - accuracy: 0.9719

Epoch 71/100

14/14 [==============================] - 0s 5ms/step - loss: 0.1193 - accuracy: 0.9625

Epoch 72/100

14/14 [==============================] - 0s 4ms/step - loss: 0.1459 - accuracy: 0.9368

Epoch 73/100

14/14 [==============================] - 0s 6ms/step - loss: 0.1256 - accuracy: 0.9578

Epoch 74/100

14/14 [==============================] - 0s 4ms/step - loss: 0.1339 - accuracy: 0.9508

Epoch 75/100

14/14 [==============================] - 0s 3ms/step - loss: 0.1113 - accuracy: 0.9696

Epoch 76/100

14/14 [==============================] - 0s 3ms/step - loss: 0.1083 - accuracy: 0.9672

Epoch 77/100

14/14 [==============================] - 0s 4ms/step - loss: 0.1090 - accuracy: 0.9719

Epoch 78/100

14/14 [==============================] - 0s 5ms/step - loss: 0.0964 - accuracy: 0.9766

Epoch 79/100

14/14 [==============================] - 0s 3ms/step - loss: 0.1138 - accuracy: 0.9485

Epoch 80/100

14/14 [==============================] - 0s 3ms/step - loss: 0.1024 - accuracy: 0.9625

Epoch 81/100

14/14 [==============================] - 0s 3ms/step - loss: 0.0943 - accuracy: 0.9859

Epoch 82/100

14/14 [==============================] - 0s 3ms/step - loss: 0.0932 - accuracy: 0.9719

Epoch 83/100

14/14 [==============================] - 0s 4ms/step - loss: 0.1053 - accuracy: 0.9578

Epoch 84/100

14/14 [==============================] - 0s 4ms/step - loss: 0.0929 - accuracy: 0.9742

Epoch 85/100

14/14 [==============================] - 0s 3ms/step - loss: 0.0918 - accuracy: 0.9742

Epoch 86/100

14/14 [==============================] - 0s 4ms/step - loss: 0.0824 - accuracy: 0.9813

Epoch 87/100

14/14 [==============================] - 0s 4ms/step - loss: 0.1056 - accuracy: 0.9649

Epoch 88/100

14/14 [==============================] - 0s 5ms/step - loss: 0.1002 - accuracy: 0.9649

Epoch 89/100

14/14 [==============================] - 0s 4ms/step - loss: 0.0987 - accuracy: 0.9625

Epoch 90/100

14/14 [==============================] - 0s 3ms/step - loss: 0.1008 - accuracy: 0.9742

Epoch 91/100

14/14 [==============================] - 0s 4ms/step - loss: 0.1161 - accuracy: 0.9625

Epoch 92/100

14/14 [==============================] - 0s 3ms/step - loss: 0.0988 - accuracy: 0.9602

Epoch 93/100

14/14 [==============================] - 0s 3ms/step - loss: 0.0898 - accuracy: 0.9719

Epoch 94/100

14/14 [==============================] - 0s 4ms/step - loss: 0.0815 - accuracy: 0.9742

Epoch 95/100

14/14 [==============================] - 0s 5ms/step - loss: 0.0965 - accuracy: 0.9602

Epoch 96/100

14/14 [==============================] - 0s 4ms/step - loss: 0.0888 - accuracy: 0.9719

Epoch 97/100

14/14 [==============================] - 0s 3ms/step - loss: 0.0919 - accuracy: 0.9672

Epoch 98/100

14/14 [==============================] - 0s 3ms/step - loss: 0.0901 - accuracy: 0.9696

Epoch 99/100

14/14 [==============================] - 0s 4ms/step - loss: 0.0750 - accuracy: 0.9836

Epoch 100/100

14/14 [==============================] - 0s 3ms/step - loss: 0.0743 - accuracy: 0.9836

<keras.callbacks.History at 0x7bc26b4b9ba0>

[ ]

ann\_y\_pred=classifier.predict(X\_test)

5/5 [==============================] - 0s 2ms/step

[ ]

ann\_y\_pred[0]

array([0.9999998], dtype=float32)

[ ]

ann\_y\_pred=ann\_y\_pred>=0.5

[ ]

from sklearn.metrics import confusion\_matrix  
ann\_cm=confusion\_matrix(Y\_test,ann\_y\_pred)  
sns.heatmap(ann\_cm,annot=True)

[ ]

from sklearn.metrics import accuracy\_score,precision\_score  
print(accuracy\_score(Y\_test,ann\_y\_pred))  
print(precision\_score(Y\_test,ann\_y\_pred))

0.6923076923076923

0.7757009345794392

Saving the models

[ ]

import pickle

pickle.dump(knn\_classifier,open('model.pkl','wb'))

pickle.dump(sc,open('sc.pkl','wb'))



Flask connection code:

App.py

from flask import Flask,request,url\_for,render\_template

import numpy as np

import pickle

sc=pickle.load(open('sc.pkl','rb'))

model=pickle.load(open('model.pkl','rb'))

app=Flask(\_\_name\_\_)

@app.route('/')

def home():

return render\_template('index.html')

@app.route('/predict',methods=['POST'])

def predict():

inputs=[float(x) for x in request.form.values()]

inputs=np.array([inputs])

inputs=sc.transform(inputs)

output=model.predict(inputs)

if output < 0.5:

output=0

else:

output=1

return render\_template('result.html',prediction=output)

if \_\_name\_\_=='\_\_main\_\_':

app.run(debug=True)

# **Conclusions**

Since the liver disease is not easy to diagnose, given the delicate nature of its signs, this research is pertinent in determining the algorithms that have better accuracy in predicting this dreadful disease. The stages in the proposed LDP method provide a better alignment of each phase. Once the dataset is selected, the preprocessing step is conducted by replacing the missing values and balancing the dataset. After that, using R, five different supervised learning methods are applied (i.e., SVM, Naïve Bayes, K- NN, LDA, and CART), and the accuracy with confusion matrix metrics are recorded. The result shows that K-NN has a better accuracy of 91.7% for liver disease prediction. Autoencoders are applied in this research as a test case for understanding the classification ability of unsupervised algorithms over other traditional approaches. In this study, the autoencoder with 3-layers achieved an accuracy of 92.1%, slightly higher than K-NN due to its ability to ascertain overlapping features better than conventional K-NNs. Most of the algorithms are more than the acceptable level of accuracy, which is 75%. The results from this study would be able to assist health care professionals and relevant stakeholders in the early detection of liver disease.

# 

# 