Introduction to Data Science

Indian Liver Patient Dataset

Data Detectives

Prepared by:

Anusheel 22ucs028
Ayush Jain 22ucs043
Piyush Kala 22ucs149
Rishabh Rathi 22ucs166

Course Instructors:

Dr. Subrat K Dash
Dr. Aloke Dutta
Dr. Manoj Kumar Yadav

LNMIIT
The LNM Institute of Information Technology
Department of Computer Science Engineering

Abstract

Liver diseases are a significant global health issue, contributing to substantial morbidity and mortality rates. These conditions exhibit distinct regional variations influenced by environmental, genetic, and lifestyle factors. India, characterized by its vast diversity and high population density, faces unique challenges in understanding and managing liver-related health issues. The "Indian Liver Patient Database" (ILPD) serves as a critical resource, containing detailed patient data, including demographic attributes, clinical indicators, and diagnostic outcomes.

This report leverages machine learning techniques, particularly logistic regression and random forest algorithms, to extract meaningful insights from the ILPD. The primary focus is on accurate disease classification, identifying influential features, and gaining a deeper understanding of liver health determinants. The preprocessing steps include addressing missing data, normalizing features, and converting categorical variables into numerical formats to ensure compatibility with the models. Logistic regression provides an interpretable model for estimating disease probabilities, while random forest enhances prediction accuracy and resilience by evaluating multiple decision trees.

The analysis offers valuable findings, such as demographic patterns of liver disease prevalence, age-specific vulnerabilities, and gender-based variations. Key predictors of liver disease, including bilirubin levels and protein ratios, are highlighted. The study also emphasizes the importance of early diagnosis and personalized healthcare strategies in addressing liver health challenges within the Indian population.

Through this data-driven approach, the report contributes to ongoing efforts to improve liver disease screening, treatment, and prevention. It underscores the potential of machine learning in advancing healthcare outcomes, particularly in regions with unique socio-cultural and epidemiological dynamics.

TABLE OF CONTENTS

Introduction

In modern healthcare, cutting-edge technologies such as Machine Learning (ML) play an essential role in enhancing the accuracy of disease diagnosis and prediction.

The initial steps involve importing necessary libraries and loading the dataset into Google Colab. Critical preprocessing tasks include addressing missing data, such as removing rows where the "Albumin-Globulin Ratio" is absent. Furthermore, categorical variables, such as gender, are transformed into numerical equivalents to ensure compatibility with ML algorithms. Additionally, normalization techniques are applied to create a balanced dataset, improving the precision of classification models.

The data analysis stage delves into demographic patterns, including gender distribution and the prevalence of liver disease. Visualizations provide insights into the dataset's composition, such as the proportion of males and females, allowing for a deeper understanding of the data. The diagnosed and undiagnosed cases are also examined, revealing valuable insights into liver disease prevalence.

This report emphasizes the application of ML algorithms, specifically Logistic Regression and Random Forest Classifier, to the Indian Liver Patient Database to refine the diagnosis of liver conditions. The dataset comprises 11 attributes, such as age, gender, and various liver metrics, necessitating meticulous preprocessing.

Visual analysis reveals age-specific patterns in disease frequency, with a significant portion of cases occurring among middle-aged and older adults. By analyzing age distributions by gender, the report identifies vulnerable groups. Exploring the distribution of liver fluids provides further understanding of potential abnormalities.

Description about dataset

This dataset integrates data on the concentrations of various pigments, proteins and enzymes within the livers of individuals across different age groups. Then, by utilizing Machine Learning, the likelihood of accurately diagnosing liver diseases is computed.

Moreover, the exploration of this dataset is not confined solely to the field of predictive modeling. Various visualization techniques, including graphical representations and charts, are employed to demonstrate the relation between age and the diagnosis of liver conditions. These visual aids serve as powerful tools in unraveling complex patterns and trends, providing a more intuitive understanding of dynamics of the dataset.

Description of Dataset

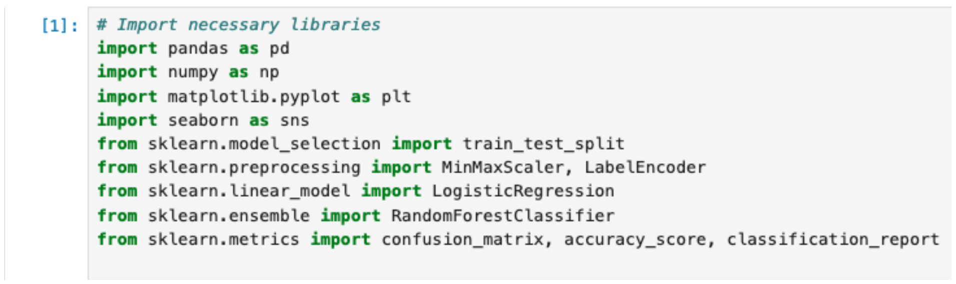
- Dataset Source: https://archive.ics.uci.edu/dataset/225/ilpd+indian+liver+patient+dataset
- Dataset Characteristics: Numerical
- Subject Area: Healthcare
- Number of Attributes: 11
- Classification Number of Instances: 583
- Programming Language Used: Python
- Platform: Google Colaboratory
- Attribute Information:

  a. Age : Integer
  b. Gender : String
  c. Total Bilirubin : Float
  d. Direct Bilirubin : Float
  e. Alkphos Alkaline Phosphatase : Integer
  f. Sgpt Alamine Aminotransferase : Integer
  g. Sgot Aspartate Aminotransferase : Integer
  h. Total Proteins : Float
  i. Albumin : Float
  j. Albumin-Globulin Ratio : Float
  k. Selector : Integer

In the **Selector** column: **'1'** means that the patient does suffer from liver disease and **'2'** means that the patient does not suffer from liver disease.

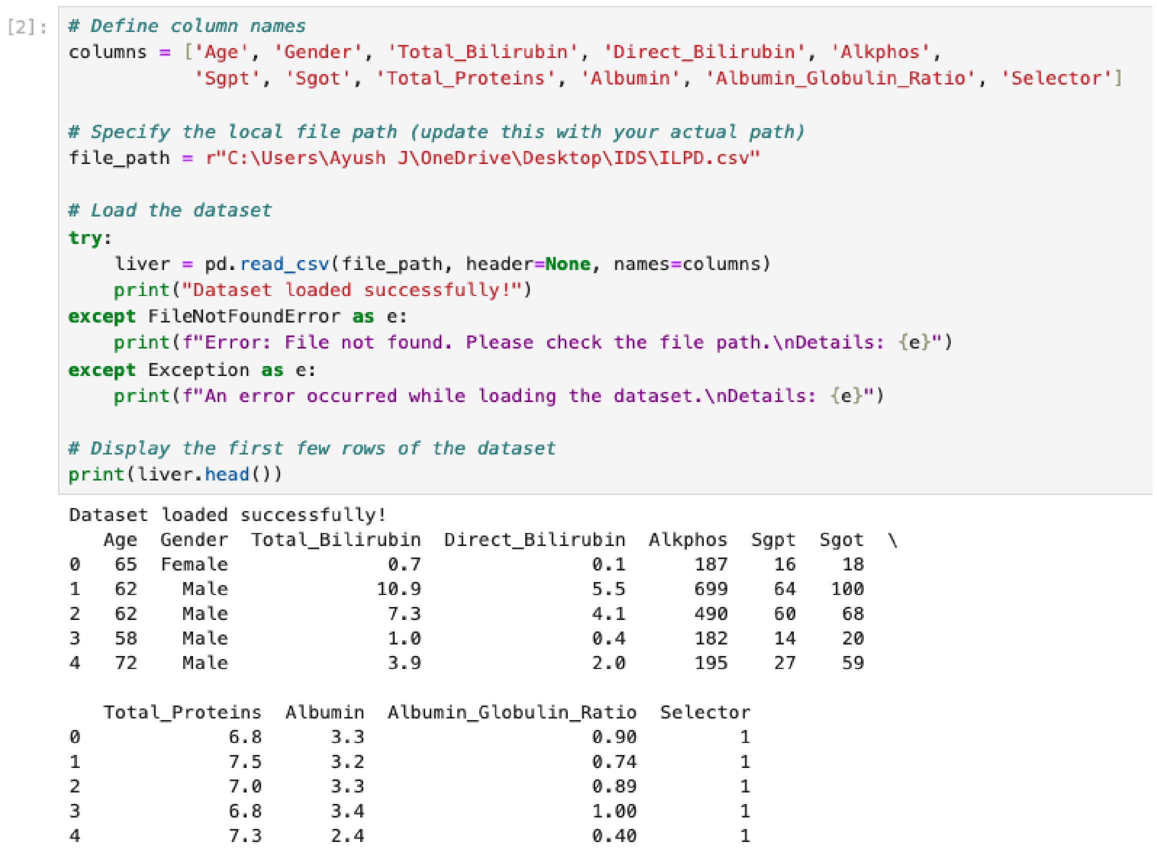
1. DATA PREPROCESSING

A. Firstly let us import all the required libraries into our notebook.



```
[1]: # Import necessary libraries
     import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import MinMaxScaler, LabelEncoder
     from sklearn.linear_model import LogisticRegression
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
```

B. Now, import the dataset into our google colab notebook, and displaying the first 5 rows of it.



```
[2]: # Define column names
     columns = ['Age', 'Gender', 'Total_Bilirubin', 'Direct_Bilirubin', 'Alkphos',
                'Sgpt', 'Sgot', 'Total_Proteins', 'Albumin', 'Albumin_Globulin_Ratio', 'Selector']

     # Specify the local file path (update this with your actual path)
     file_path = r"C:\Users\Ayush J\OneDrive\Desktop\IDS\ILPD.csv"

     # Load the dataset
     try:
         liver = pd.read_csv(file_path, header=None, names=columns)
         print("Dataset loaded successfully!")
     except FileNotFoundError as e:
         print(f"Error: File not found. Please check the file path.\nDetails: {e}")
     except Exception as e:
         print(f"An error occurred while loading the dataset.\nDetails: {e}")

     # Display the first few rows of the dataset
     print(liver.head())
```

```
Dataset loaded successfully!
   Age  Gender  Total_Bilirubin  Direct_Bilirubin  Alkphos  Sgpt  Sgot  \
0   65  Female              0.7               0.1      187    16    18
1   62    Male             10.9               5.5      699    64   100
2   62    Male              7.3               4.1      490    60    68
3   58    Male              1.0               0.4      182    14    20
4   72    Male              3.9               2.0      195    27    59

   Total_Proteins  Albumin  Albumin_Globulin_Ratio  Selector
0             6.8      3.3                    0.90         1
1             7.5      3.2                    0.74         1
2             7.0      3.3                    0.89         1
3             6.8      3.4                    1.00         1
4             7.3      2.4                    0.40         1
```

[We have given the variable name “**liver**” to the pandas dataframe.]

C. To get a rough overview of the data, let us dig deeper into its types and count.

```
[6]: # Check for missing values
     print("Missing values in each column:")
     print(liver.isnull().sum())

     # Basic statistics of the dataset
     print("\nBasic statistics of the dataset:")
     print(liver.describe())

     # Check the data types of columns
     print("\nData types of columns:")
     print(liver.dtypes)

     # Value counts for the target column (Selector)
     print("\nValue counts for the target column:")
     print(liver['Selector'].value_counts())
```

```
Missing values in each column:
Age                         0
Gender                      0
Total_Bilirubin             0
Direct_Bilirubin            0
Alkphos                     0
Sgpt                        0
Sgot                        0
Total_Proteins              0
Albumin                     0
Albumin_Globulin_Ratio      4
Selector                    0
dtype: int64

Basic statistics of the dataset:
              Age  Total_Bilirubin  Direct_Bilirubin      Alkphos  \
count  583.000000       583.000000        583.000000   583.000000
mean    44.746141         3.298799          1.486106   290.576329
std     16.189833         6.209522          2.808498   242.937989
min      4.000000         0.400000          0.100000    63.000000
25%     33.000000         0.800000          0.200000   175.500000
50%     45.000000         1.000000          0.300000   208.000000
75%     58.000000         2.600000          1.300000   298.000000
max     90.000000        75.000000         19.700000  2110.000000

              Sgpt         Sgot  Total_Proteins     Albumin  \
count   583.000000   583.000000      583.000000  583.000000
mean     80.713551   109.910806        6.483190    3.141852
std     182.620356   288.918529        1.085451    0.795519
min      10.000000    10.000000        2.700000    0.900000
25%      23.000000    25.000000        5.800000    2.600000
50%      35.000000    42.000000        6.600000    3.100000
75%      60.500000    87.000000        7.200000    3.800000
max    2000.000000  4929.000000        9.600000    5.500000

       Albumin_Globulin_Ratio    Selector
count              579.000000  583.000000
mean                 0.947064    1.286449
std                  0.319592    0.452490
min                  0.300000    1.000000
25%                  0.700000    1.000000
50%                  0.930000    1.000000
75%                  1.100000    2.000000
max                  2.800000    2.000000

Data types of columns:
Age                         int64
Gender                     object
Total_Bilirubin           float64
Direct_Bilirubin          float64
Alkphos                     int64
Sgpt                        int64
Sgot                        int64
Total_Proteins            float64
Albumin                   float64
Albumin_Globulin_Ratio    float64
Selector                    int64
dtype: object
```

—> We can see that the count of most of the features is 583. However for the attribute **"Albumin-Globulin Ratio"** the count is 579. That means 4 values of Albumin-Globulin Ratio **are NOT** present in our dataset.

1.1 Incomplete/Missing Data

1.1.1 What could be the reason for missing data?

- Information not collected for the specific attribute. People decline to give out information as a means to protect their privacy.
- Attributes may not be applicable to all cases.

1.1.2 Dealing with missing values.

- Eliminate data objects/or those variables.
- Estimate the missing values. Put a “default” value.
- Just Ignore the missing values in the analysis

1.2 Handling the missing values in the Albumin-Globulin Ratio attribute

—> We have decided to replace missing values in 'Albumin_Globulin_Ratio' with the mean of the column using **.fillna()** command.

—> By filling in the missing values with the mean, we ensure that the dataset remains compatible with algorithms sensitive to missing entries, such as regression models or machine learning pipelines. After applying this method, it's important to validate the results and check for any unintended biases that might affect the integrity of the analysis.

[9]:
```
# Replace missing values in 'Albumin_Globulin_Ratio' with the mean of the column
liver['Albumin_Globulin_Ratio'] = liver['Albumin_Globulin_Ratio'].fillna(liver['Albumin_Globulin_Ratio'].mean())

# Verify again for missing values
print("\nMissing values after handling:")
print(liver.isnull().sum())
```

```
Missing values after handling:
Age                       0
Gender                    0
Total_Bilirubin           0
Direct_Bilirubin          0
Alkphos                   0
Sgpt                      0
Sgot                      0
Total_Proteins            0
Albumin                   0
Albumin_Globulin_Ratio    0
Selector                  0
dtype: int64
```

1.3 Handling Categorical Data

Categorical features refer to string data types and can be easily understood by human beings. However, machines cannot interpret the categorical data directly. Therefore, the categorical data must be converted into numerical data for further processing.

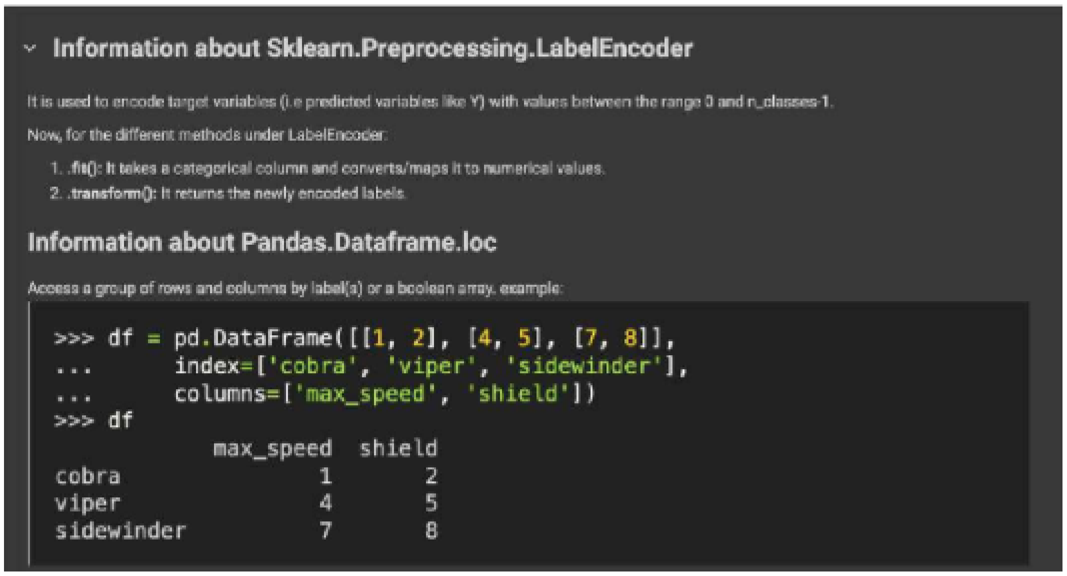
Let us see in our dataset if we have some categorical attributes or not.

```
[12]: # Encode the 'Gender' column (Female -> 0, Male -> 1)
      liver['Gender'] = liver['Gender'].map({'Female': 0, 'Male': 1})

      # Verify the encoding
      print("\nUnique values in the 'Gender' column after encoding:")
      print(liver['Gender'].unique())
```

```
Unique values in the 'Gender' column after encoding:
[0 1]
```

We see that the "Gender" column is a categorical feature having 2 values: Male and Female. We have converted these values into numerals.

Python Script:



Information about Sklearn.Preprocessing.LabelEncoder

It is used to encode target variables (i.e predicted variables like Y) with values between the range 0 and n_classes-1.

Now, for the different methods under LabelEncoder:

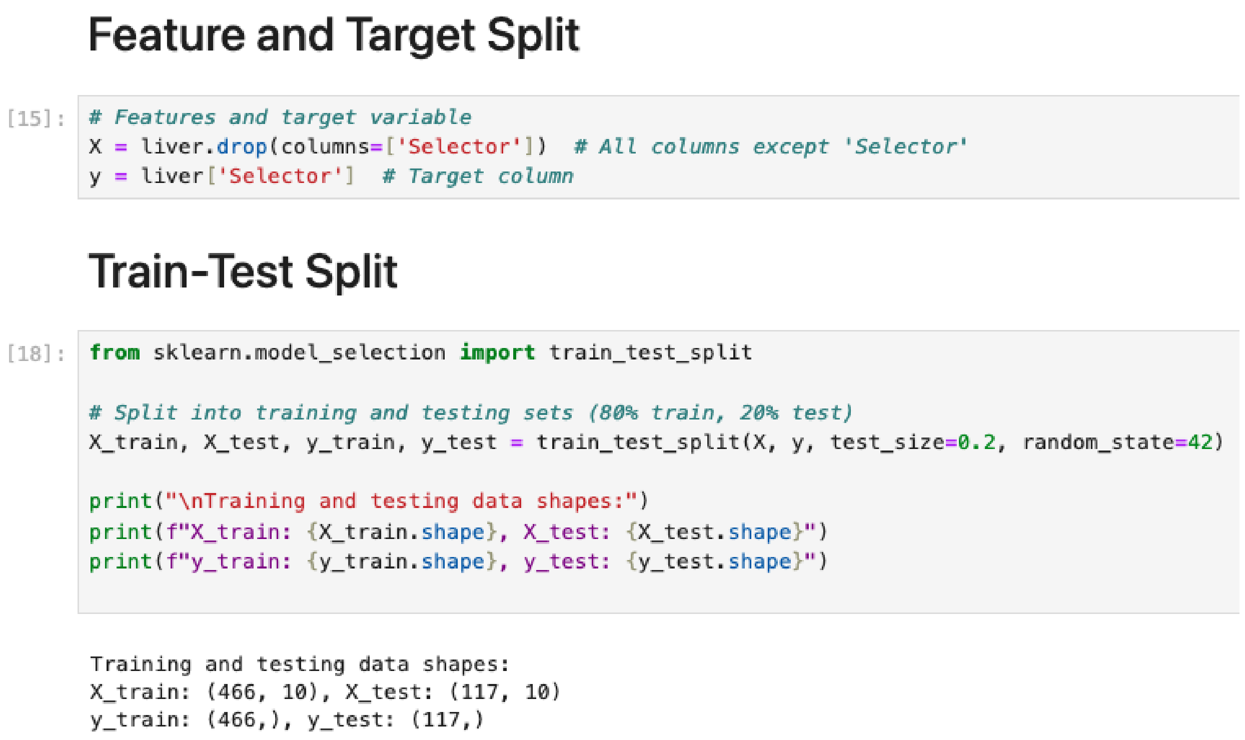
1. **.fit():** It takes a categorical column and converts/maps it to numerical values.
2. **.transform():** It returns the newly encoded labels.

Information about Pandas.Dataframe.loc

Access a group of rows and columns by label(s) or a boolean array. example:

```
>>> df = pd.DataFrame([[1, 2], [4, 5], [7, 8]],
...      index=['cobra', 'viper', 'sidewinder'],
...      columns=['max_speed', 'shield'])
>>> df
            max_speed  shield
cobra               1       2
viper               4       5
sidewinder          7       8
```

—> What this label encoder does is it maps MALE, FEMALE attribute values to '1' and '0'b respectively.



Feature and Target Split

[15]:
```python
# Features and target variable
X = liver.drop(columns=['Selector'])  # All columns except 'Selector'
y = liver['Selector']  # Target column
```

Train-Test Split

[18]:
```python
from sklearn.model_selection import train_test_split

# Split into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("\nTraining and testing data shapes:")
print(f"X_train: {X_train.shape}, X_test: {X_test.shape}")
print(f"y_train: {y_train.shape}, y_test: {y_test.shape}")
```

```
Training and testing data shapes:
X_train: (466, 10), X_test: (117, 10)
y_train: (466,), y_test: (117,)
```

—> Splitting columns into **Features (Independent) and Target (Dependent)** columns.
—> Splitting dataset into **Training and Testing datasets** with a train-test split of 80-20, using **train_test_split()** method.

1.4 Data Transformation

It is clear that values across different variables are distributed far too extensively. We could perform feature scaling or normalization so as to improve the classifier accuracy.

1.4.1 Attribute Transformation

An attribute transform is a function that maps the entire set of values of a given attribute to a new set of replacement values such that each old value can be identified with one of the new values.

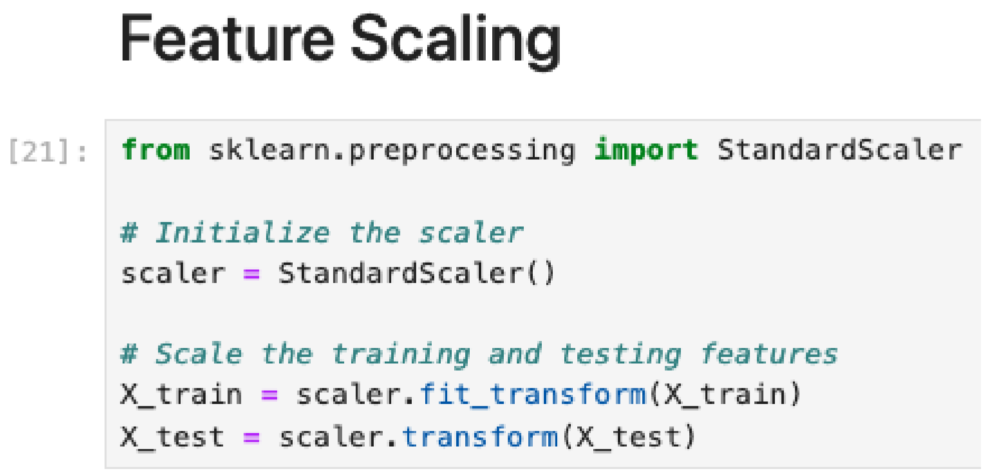
We can achieve so by Normalization, Standardization, Feature Construction, Aggregation, Discretization etc.

1.4.2 Standardization

The attributes are transformed to have a mean of 0 and a standard deviation of 1, ensuring a consistent scale across all features. This process is known as **Standardization** and can be achieved through the following methods:

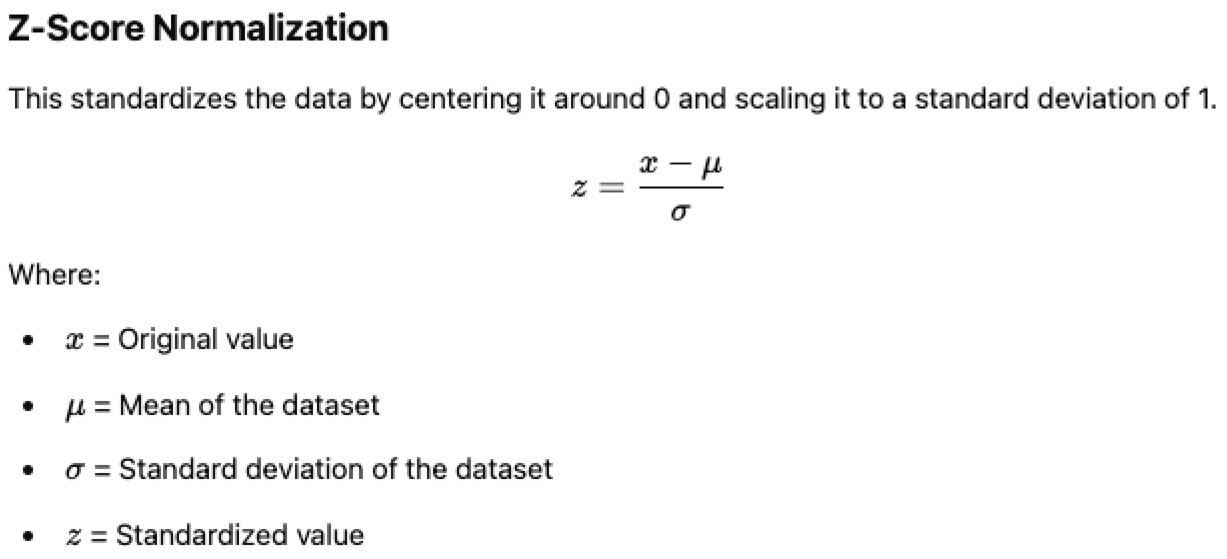
- **Z-Score Standardization**
- **Mean Normalization**
- **Scaling to Unit Variance**

→ We use **Z-Score Standardization** to standardize our data, which adjusts each attribute by subtracting the mean and dividing by the standard deviation. This method is particularly useful when the data has varying units or scales, ensuring that all features contribute equally to the analysis or model.



Feature Scaling

```
[21]: from sklearn.preprocessing import StandardScaler

      # Initialize the scaler
      scaler = StandardScaler()

      # Scale the training and testing features
      X_train = scaler.fit_transform(X_train)
      X_test = scaler.transform(X_test)
```



Z-Score Normalization

This standardizes the data by centering it around 0 and scaling it to a standard deviation of 1.

$$z = \frac{x - \mu}{\sigma}$$

Where:

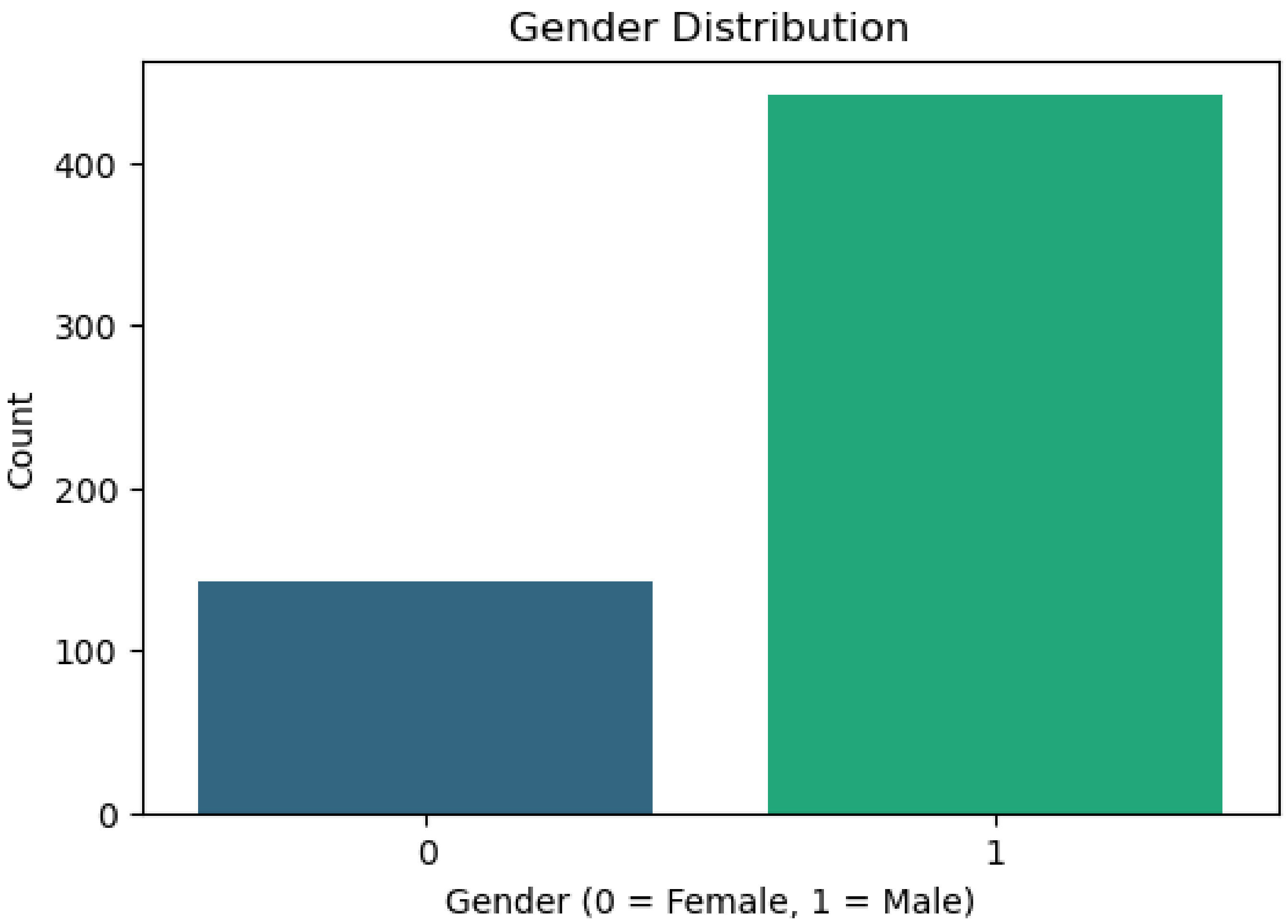
- x = Original value
- μ = Mean of the dataset
- σ = Standard deviation of the dataset
- z = Standardized value

2. Data Analysis

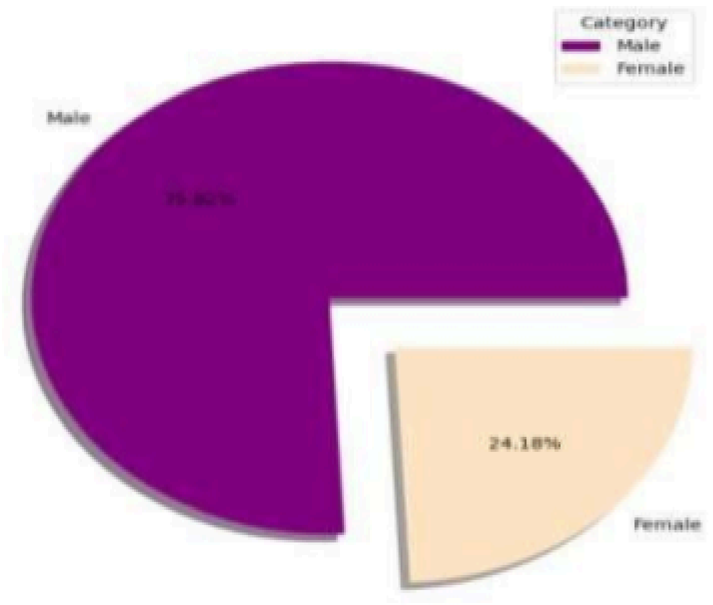
Through python scripts, we have delved into the dataset and generated visualizations of some essential summary statistics in order to derive some meaningful insights from the data..

2.1 Number of Males & Females Patients

There are 583 patients in our dataset overall; 439 of them are men, while 140 of them are women.

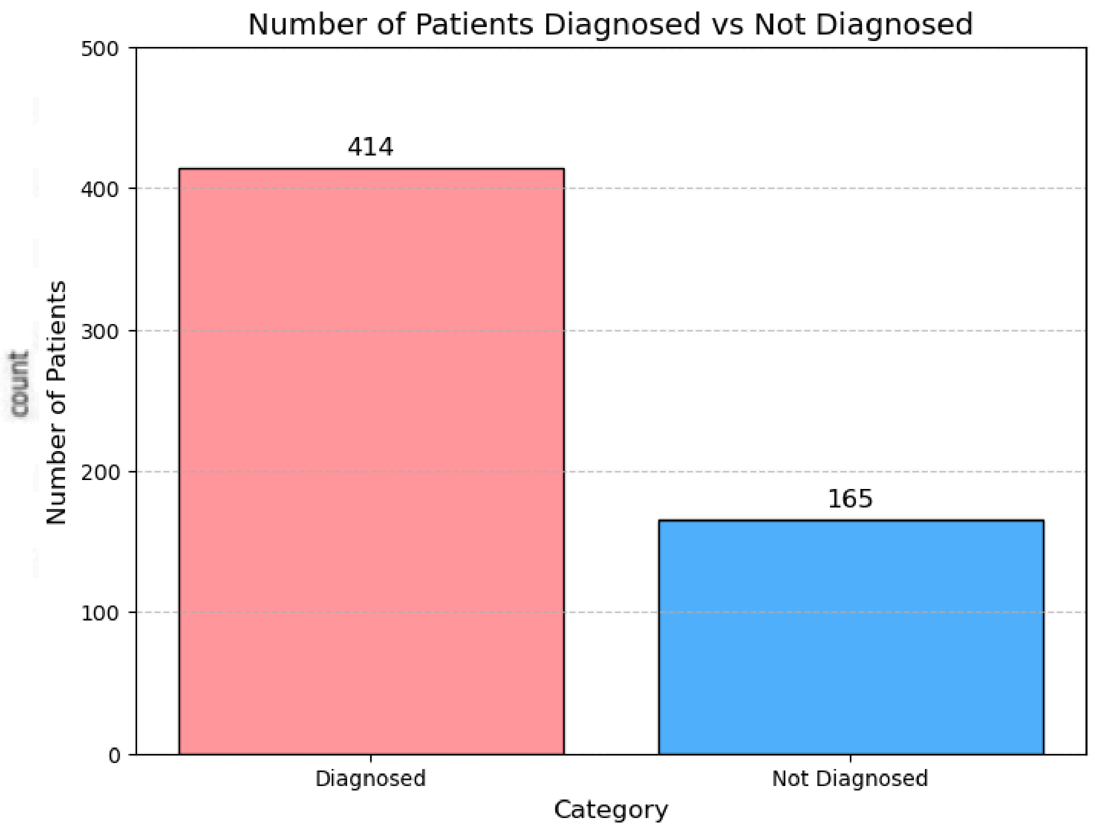


Out of the entire pool of analyzed patients, the number of men account for 75.82 percent, and the number of women account for 24.18 percent.



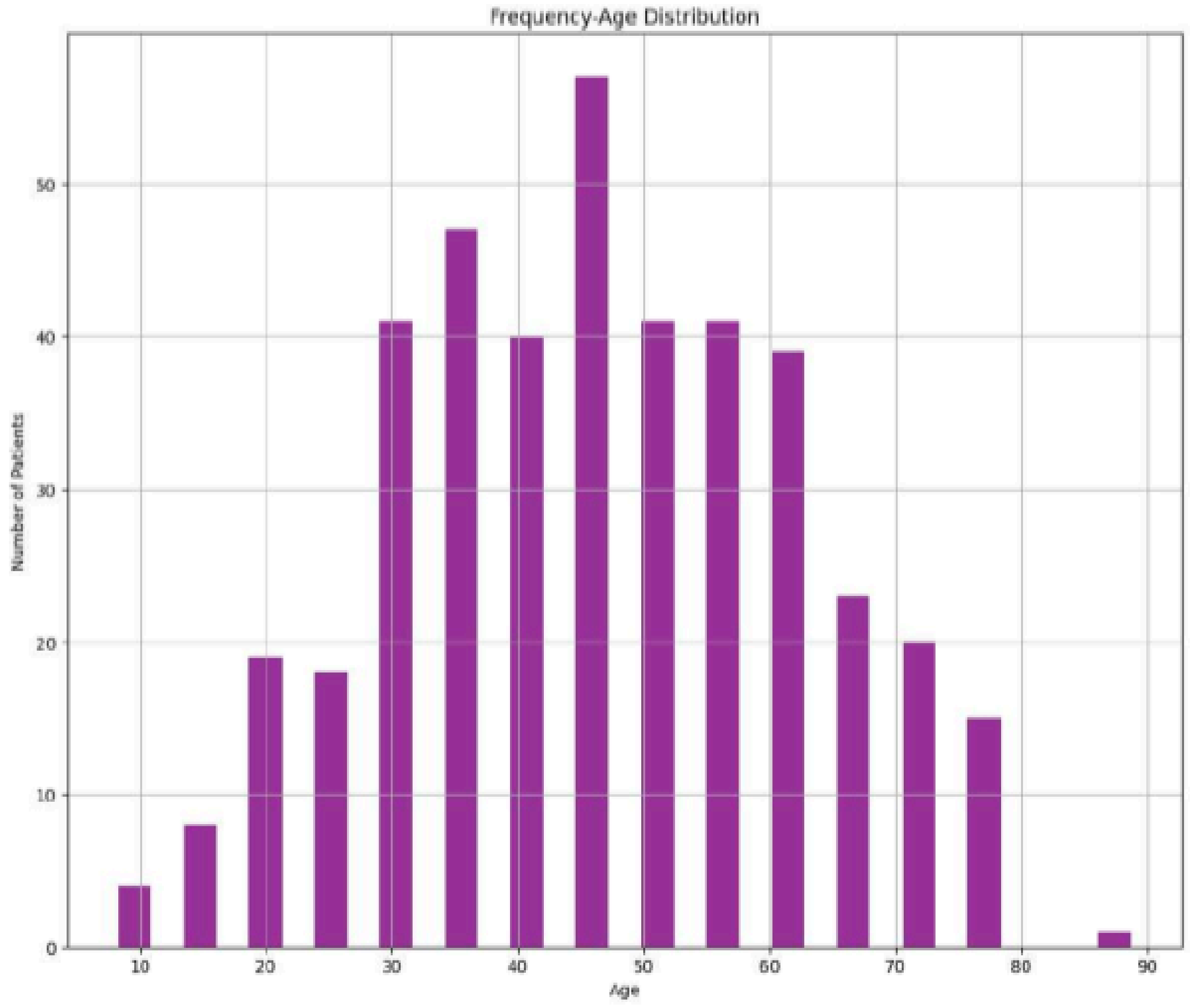
2.2 Number of Patients Diagnosed

A total of 579 were included in the examination, of which, 414 were concluded diagnosed with the liver disease, while the remaining 165 did not exhibit any indications of the disease and hence, were concluded undiagnosed.



2.3 Distribution of Frequency with Age

The distribution of frequency of patients with their ages can be observed through the graphical representation of frequency-age distribution shown below.

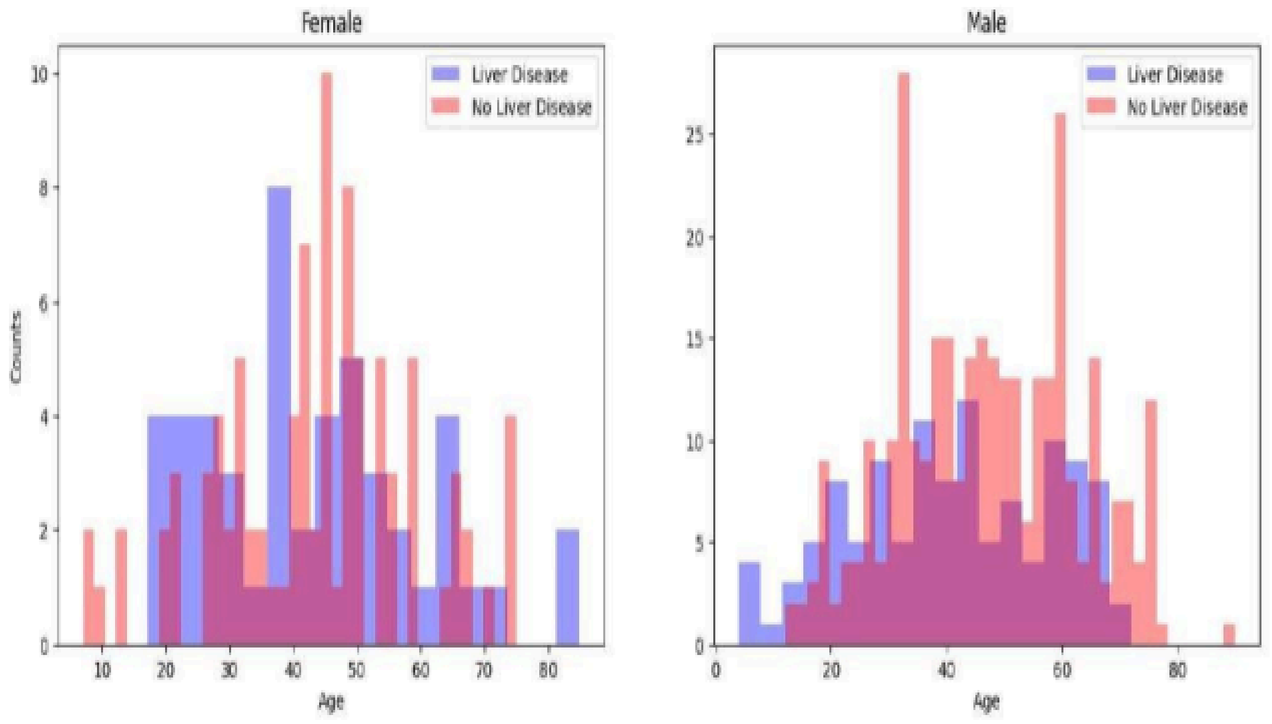


The observed distribution of patients across different age groups reveals that a significant proportion of individuals diagnosed with liver diseases are concentrated in the age range of 30 to 65. Additionally, the bar heights corresponding to ages between 65-80 indicate a higher susceptibility among elderly patients compared to younger individuals aged 20-30, as expected.

This suggests that individuals in the middle to early stages of old age are more susceptible to liver-related ailments

2.4 Gender Specific Frequency-Age Distribution

Now, let's dive in deeper and examine the distribution of patients in different age groups based on their gender. The graph facilitates the differentiation between diagnosed and undiagnosed patients through color-coded distinctions.

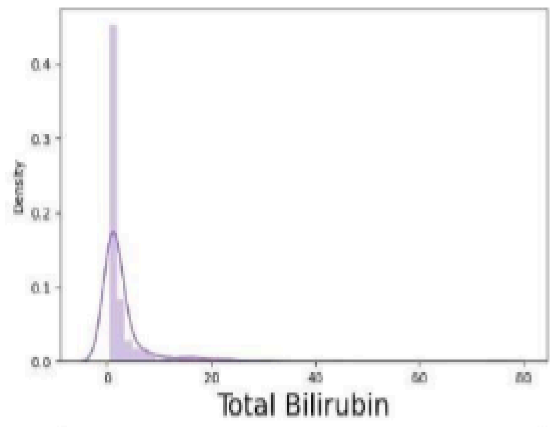


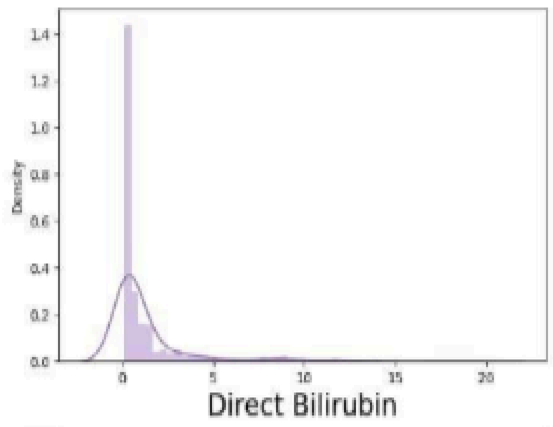
The following conclusions can be drawn from the above graphs:

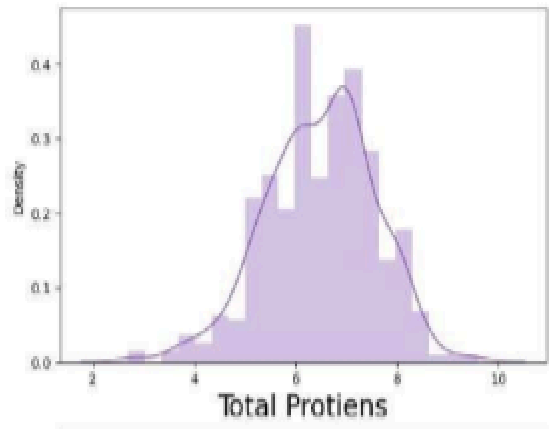
- Females between the ages 20-40 are more likely to have the disease and same for the males between the ages 30-40.
- Females between the ages 35-40 are the worst affected while the most susceptible males fall between the ages 40-45.
- Young females (aged between 18-40) are more prone to liver diseases than older ones, while the frequency of males is somewhat uniform between ranges 20-45 and 60-70.
- Young females (aged between 0-18) are less likely to get diagnosed than male children.

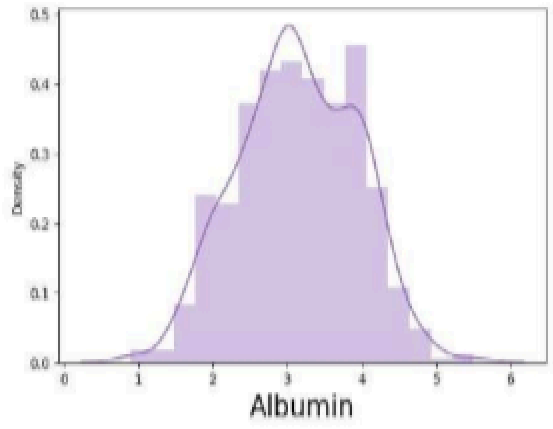
2.5 Distribution of Different Liver Fluids

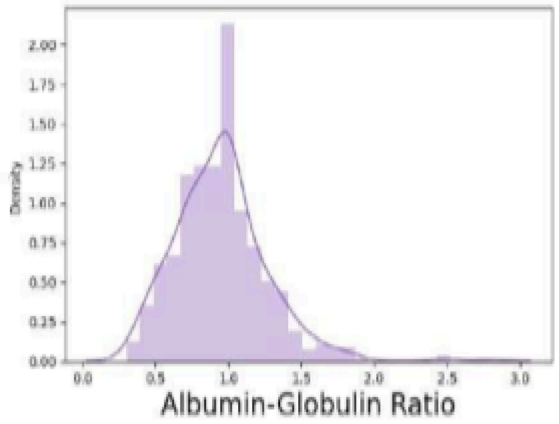
The given graphs exhibit the distribution of different fluids found in the human liver with their total density. The data corresponds to the fluid concentrations found in the livers of examined patients.











Total Bilirubin and Direct Bilirubin shows standard normal distribution and Total Proteins, Albumin and Albumin and Globulin Ratio shows normal distribution.

What is Normal Distribution?

Normal distribution is a probability distribution that is symmetric about the mean, showing that data near the mean are more frequent in occurrence than data far from the mean.

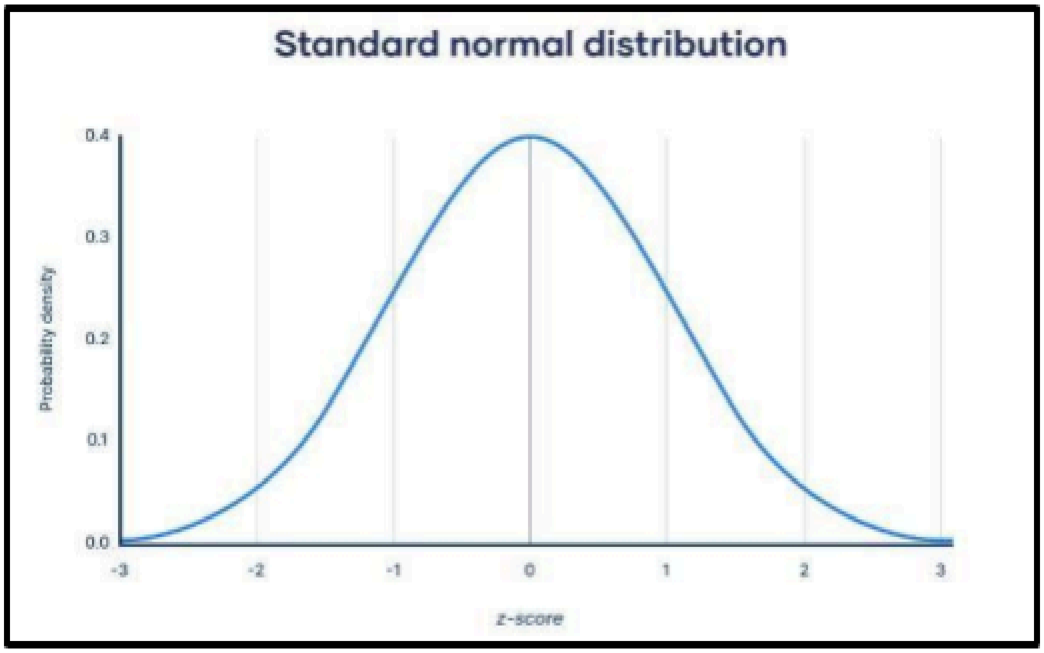
The probability density function (PDF) of a normal distribution is given by the formula:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

where,

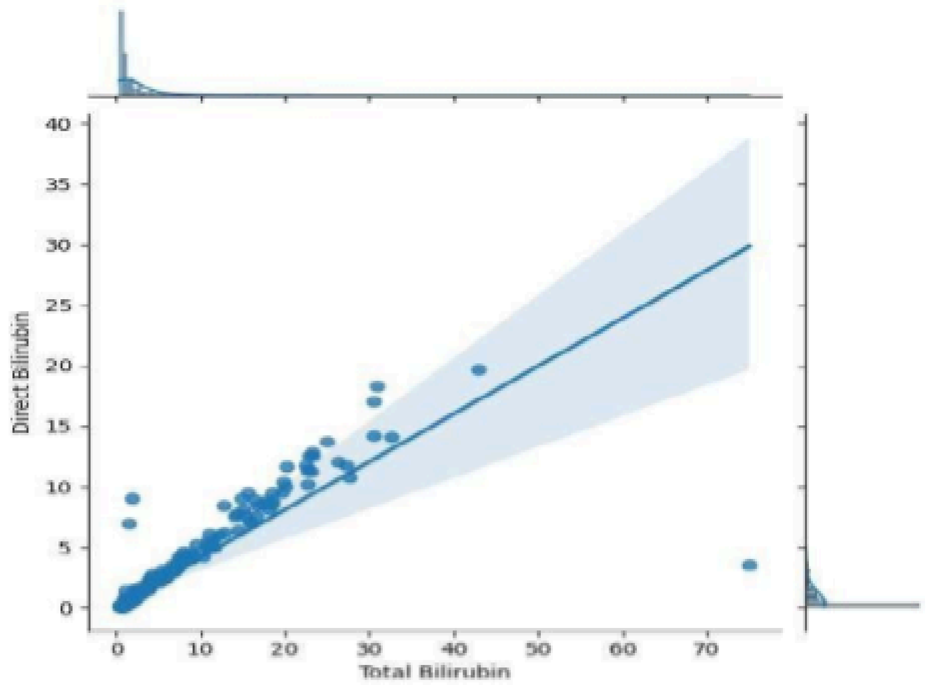
- $f(x;\mu,\sigma)$ is the probability density function.
- x is the variable.
- μ is the mean.
- σ is the standard deviation.
- e is the base of the natural logarithm.

In graphical form, the normal distribution appears as a bell curve. The graph is illustrated below:

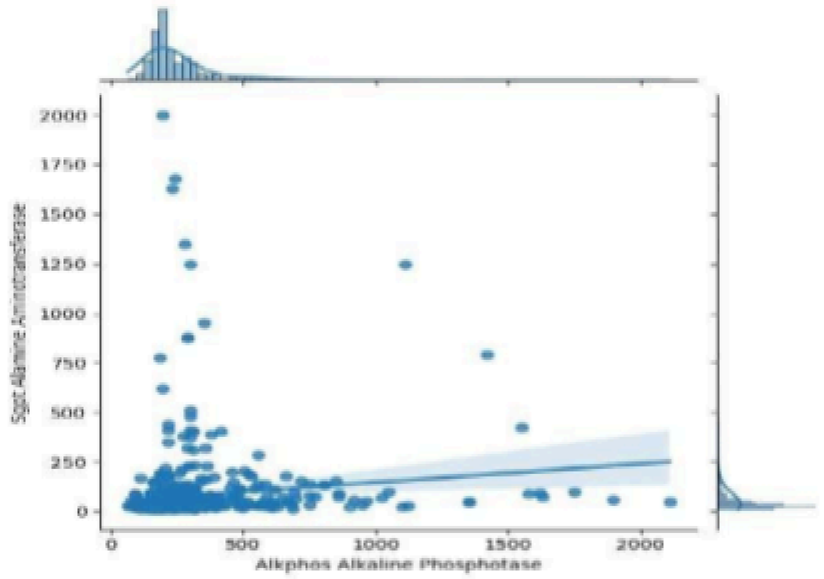


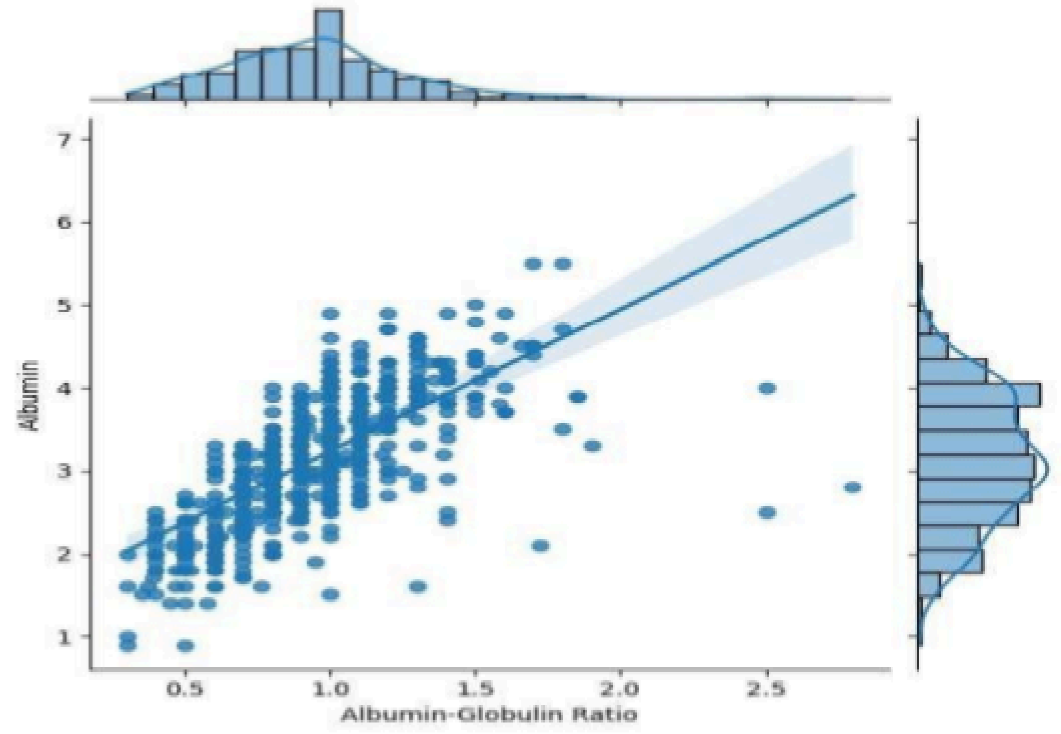
2.6 Correlation Between Concentrations of Different Fluids

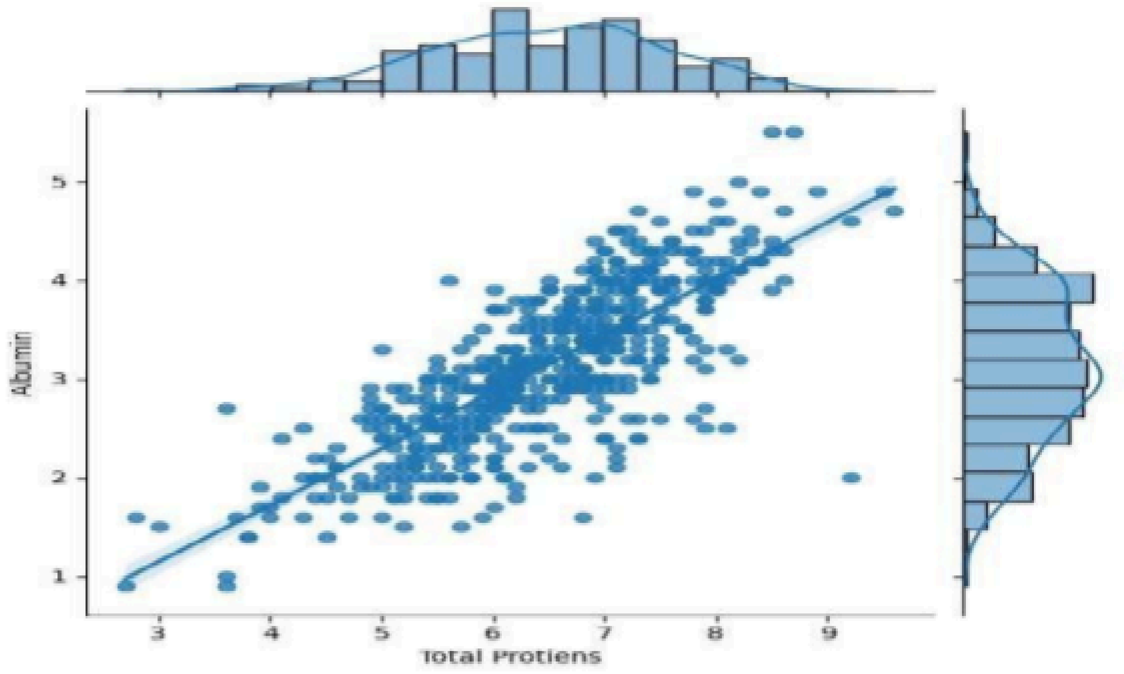
The plots charts given below demonstrate the correlation between the content of different liver fluids with each other.



No Linear Correlation between the SGPT Alanine Aminotransferase and Alkphos Alkaline Alkaline Phosphatase:



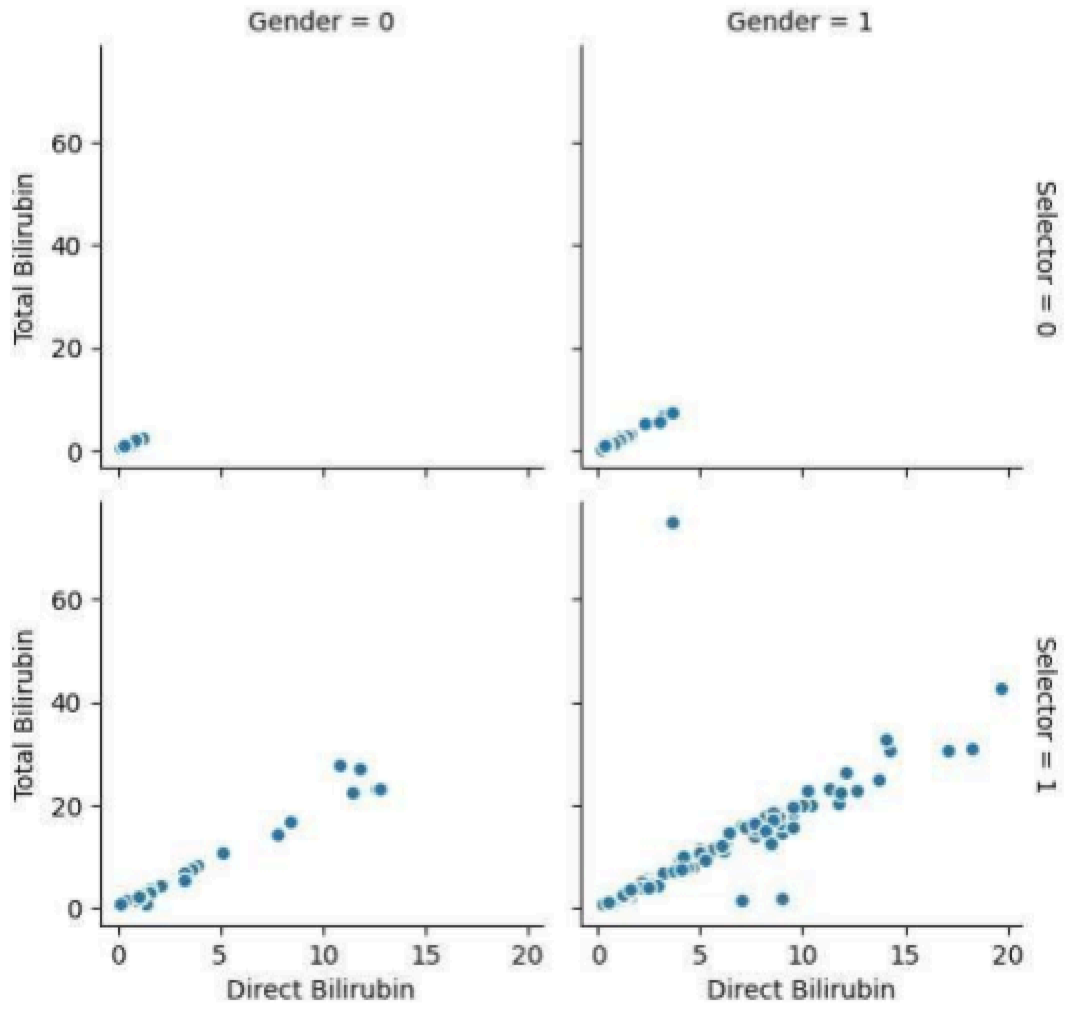




From the above joint plots and scatter plots, we find a direct relationship between the following features: Direct Bilirubin & Total Bilirubin Aspartate Aminotransferase & Alanine Aminotransferase Total Protein & Albumin Albumin and Globulin Ratio & Albumin.

2.7 Gender Specific Bilirubin Distribution

The scatter plots below illustrate the proportion of Direct Bilirubin to Total Bilirubin in the livers of both male and female individuals. In this context, a gender code of 0 denotes females, while a code of 1 represents males. Likewise, a selector code of 0 signifies individuals not diagnosed, and a code of 1 indicates those diagnosed with liver disease.

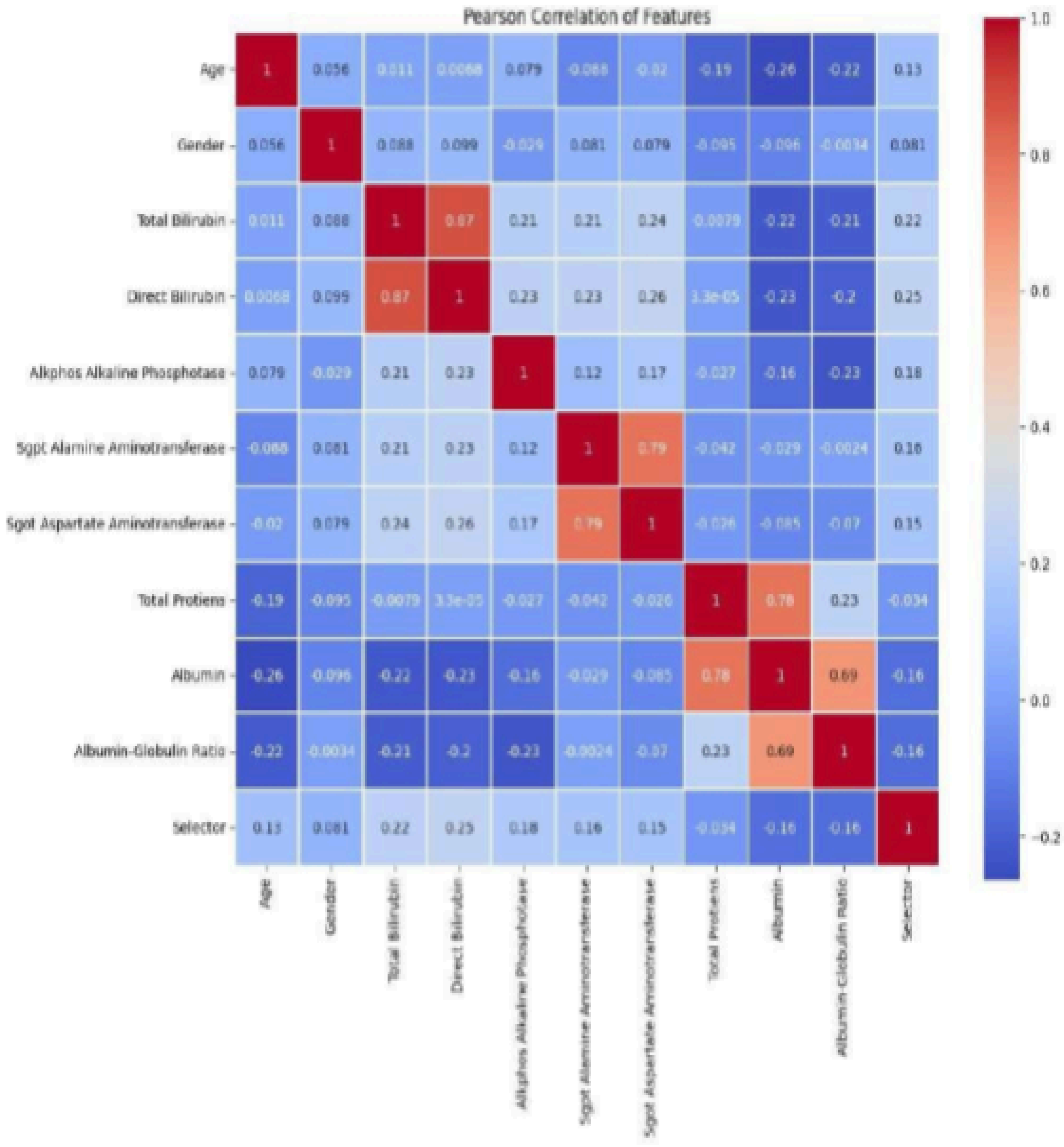


The key inferences are listed below:

- The ratio is sparser in undiagnosed patients compared to the diagnosed ones.
- The density of distribution is higher in males compared to females.

2.8 Correlation Map of All Features

We can discern patterns and associations through this map, providing a comprehensive overview of how different features interact with each other, contributing to a more thorough understanding of the dataset's internal dynamics. The diagonal elements equating to one can be attributed to the fact that identical features are being correlated with themselves during the mapping process.



3. Classification Models

What is Logistic regression?

Logistic regression is a statistical method used for binary classification. Logistic regression predicts binary outcomes, such as disease presence or absence. It uses the sigmoid function to map results between 0 and 1, offering probabilistic predictions. This method's simplicity and efficiency make it a robust baseline model for binary classification tasks. Despite its name, logistic regression is a classification algorithm rather than a regression algorithm.

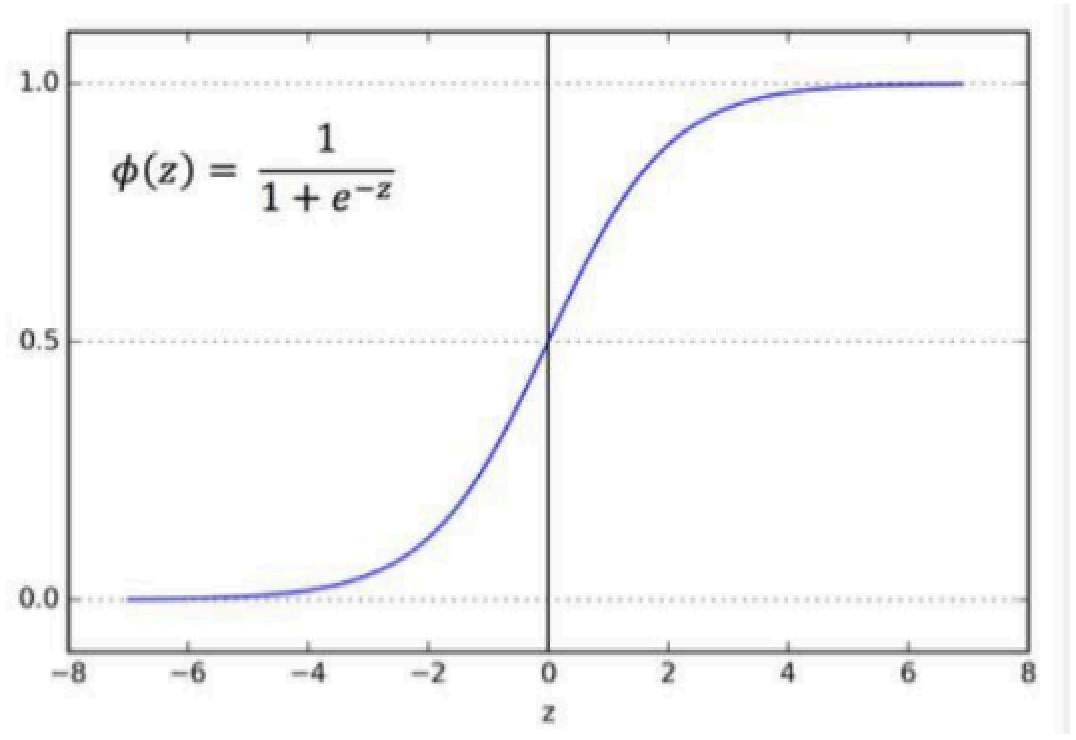
How does Logistic Regression work?

Logistic regression works in the following steps:

- Prepare the data: The data should be in a format where each row represents a single observation, and each column represents a different variable. The target variable (the variable you want to predict) should be binary (yes/no, true/false, 0/1).
- Train the model: We teach the model by showing it the training data. This involves finding the values of the model parameters that minimize the error in the training data.
- Evaluate the model: The model is evaluated on the held-out test data to assess its performance on unseen data.
- Use the model to make predictions: After the model has been trained and assessed, it can be used to forecast outcomes on new data.

Sigmoid Function (Logistic Function) :

The sigmoid function is a mathematical function used to map the predicted values to probabilities.
It maps any real value into another value within a range of 0 and 1. The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form.

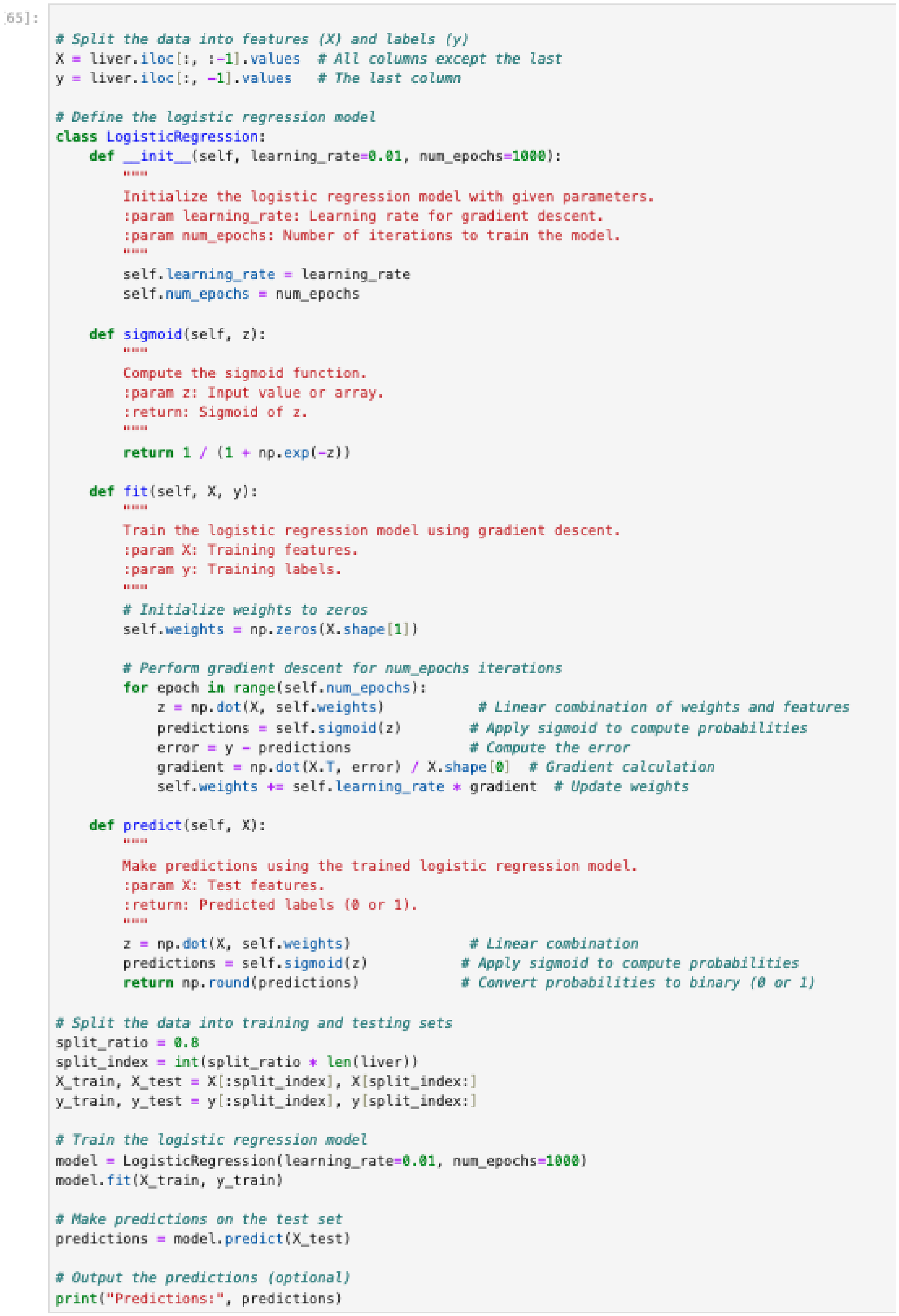


Here, z is the linear combination of input features and weights.

Why use Logistic Regression?

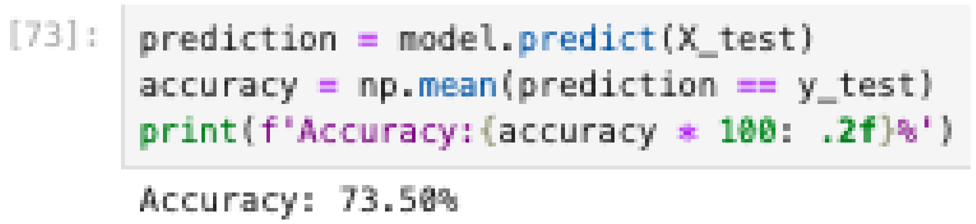
Logistic regression is chosen for the "Indian Liver Patient Database" analysis due to its suitability for binary classification problems, interpretability, and probabilistic outputs. In the healthcare context, it provides insights into factors influencing liver health, assesses resource importance, and offers a simple, efficient model. Logistic regression aligns with the study's goals of accurate classification and understanding data-driven patterns. It also serves as a baseline model, facilitating comparisons with more complex algorithms like random forest. Additionally, logistic regression is well-suited for preprocessed data and normalization, contributing to stable results in the analysis of liver disease patterns.

The python code shows the implementation of this technique on our dataset: -



```
65]:
# Split the data into features (X) and labels (y)
X = liver.iloc[:, :-1].values  # All columns except the last
y = liver.iloc[:, -1].values   # The last column

# Define the logistic regression model
class LogisticRegression:
    def __init__(self, learning_rate=0.01, num_epochs=1000):
        """
        Initialize the logistic regression model with given parameters.
        :param learning_rate: Learning rate for gradient descent.
        :param num_epochs: Number of iterations to train the model.
        """
        self.learning_rate = learning_rate
        self.num_epochs = num_epochs

    def sigmoid(self, z):
        """
        Compute the sigmoid function.
        :param z: Input value or array.
        :return: Sigmoid of z.
        """
        return 1 / (1 + np.exp(-z))

    def fit(self, X, y):
        """
        Train the logistic regression model using gradient descent.
        :param X: Training features.
        :param y: Training labels.
        """
        # Initialize weights to zeros
        self.weights = np.zeros(X.shape[1])

        # Perform gradient descent for num_epochs iterations
        for epoch in range(self.num_epochs):
            z = np.dot(X, self.weights)           # Linear combination of weights and features
            predictions = self.sigmoid(z)        # Apply sigmoid to compute probabilities
            error = y - predictions              # Compute the error
            gradient = np.dot(X.T, error) / X.shape[0]  # Gradient calculation
            self.weights += self.learning_rate * gradient  # Update weights

    def predict(self, X):
        """
        Make predictions using the trained logistic regression model.
        :param X: Test features.
        :return: Predicted labels (0 or 1).
        """
        z = np.dot(X, self.weights)              # Linear combination
        predictions = self.sigmoid(z)           # Apply sigmoid to compute probabilities
        return np.round(predictions)            # Convert probabilities to binary (0 or 1)

# Split the data into training and testing sets
split_ratio = 0.8
split_index = int(split_ratio * len(liver))
X_train, X_test = X[:split_index], X[split_index:]
y_train, y_test = y[:split_index], y[split_index:]

# Train the logistic regression model
model = LogisticRegression(learning_rate=0.01, num_epochs=1000)
model.fit(X_train, y_train)

# Make predictions on the test set
predictions = model.predict(X_test)

# Output the predictions (optional)
print("Predictions:", predictions)
```

Accuracy: 73.50%



```
[73]: prediction = model.predict(X_test)
      accuracy = np.mean(prediction == y_test)
      print(f'Accuracy:{accuracy * 100: .2f}%')

      Accuracy: 73.50%
```

Random Forest Classifier

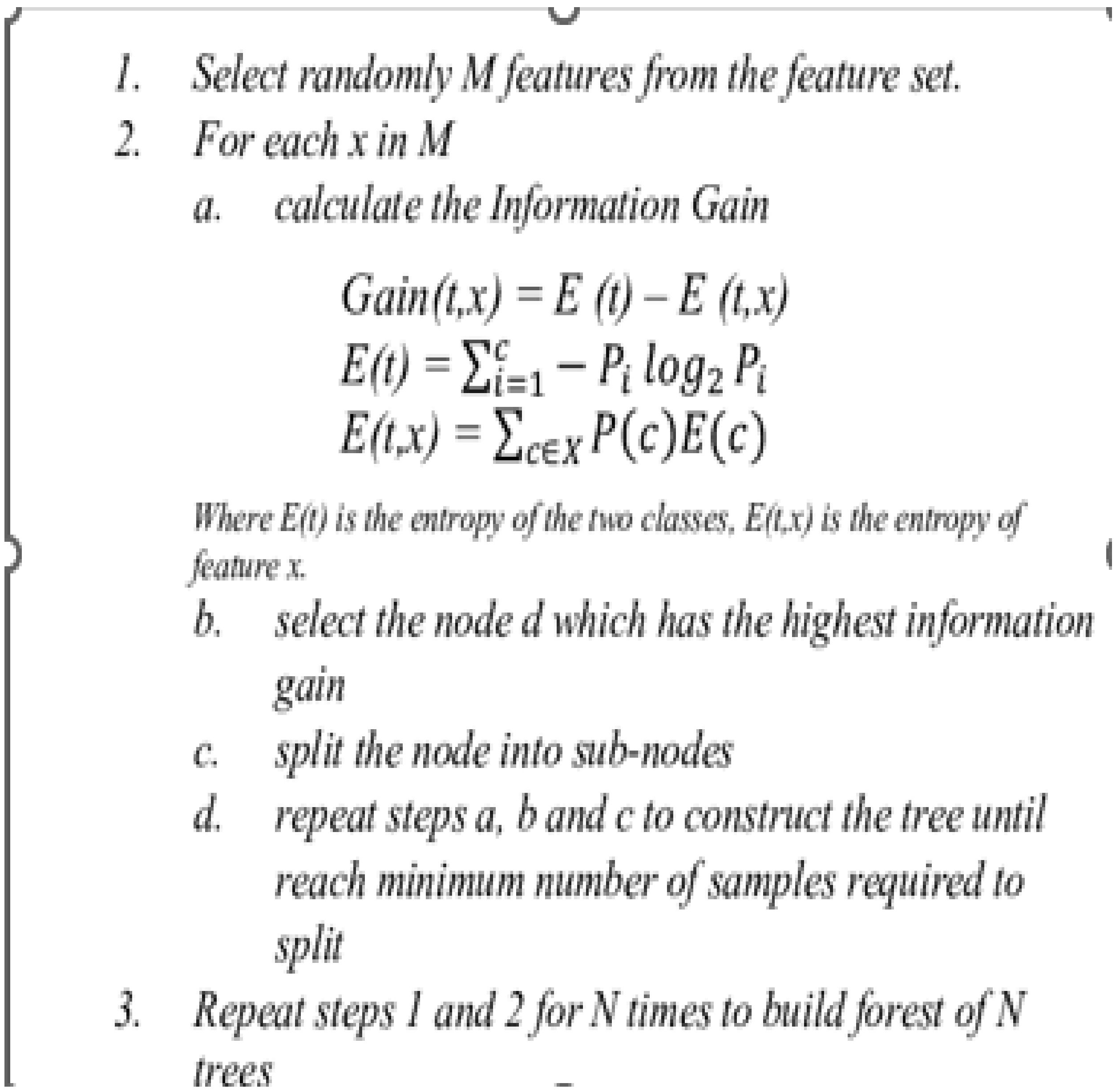
Random Forest Classifier is a popular machine learning algorithm that is used for classification tasks. It is an ensemble learning method that combines multiple decision trees to improve the accuracy and stability of the model.

The Random Forest Classifier combines multiple decision trees to enhance accuracy and reduce overfitting. By using bagging techniques, it generates predictions based on majority voting. Feature importance analysis identifies key predictors such as Direct Bilirubin and Total Bilirubin.

How Random Forest Classifier Works

Random Forest Classifier works by creating multiple decision trees, each based on a random subset of the training data. The algorithm then combines the predictions of the individual trees to make a final prediction. This helps to reduce overfitting and improve the accuracy of the model.

Random Forest Classifier uses several formulae and calculations to build the decision trees and make predictions. One of the many formulae and calculations approach that is used is as:



```
1. Select randomly M features from the feature set.
2. For each x in M
    a. calculate the Information Gain
            $Gain(t,x) = E(t) - E(t,x)$
            $E(t) = \sum_{i=1}^{c} - P_i \log_2 P_i$
            $E(t,x) = \sum_{c \in X} P(c)E(c)$
       Where E(t) is the entropy of the two classes, E(t,x) is the entropy of
       feature x.
    b. select the node d which has the highest information
       gain
    c. split the node into sub-nodes
    d. repeat steps a, b and c to construct the tree until
       reach minimum number of samples required to
       split
3. Repeat steps 1 and 2 for N times to build forest of N
   trees
```

Concept of Bagging

The Random Forest Classifier uses bagging (bootstrap aggregating) to construct individual decision trees, improving model stability and accuracy by reducing variance and overfitting. Bagging involves creating multiple random subsets of the training data with replacement, allowing some data points to appear multiple times while others are excluded. Each tree is trained on these subsets, using a random selection of features at every split to introduce diversity. Final predictions are made by majority voting (for classification) or averaging (for regression). This ensemble approach leverages the strengths of individual trees, ensuring robust performance and better generalization to unseen data.

Decision Trees

Decision trees are the building blocks of Random Forest Classifier. Each decision tree is built using a subset of the training data and a random subset of the features. The decision tree splits the data into smaller subsets based on the values of the features and creates a set of rules for predicting the target variable. The process is repeated for each subset until a stopping criterion is met.

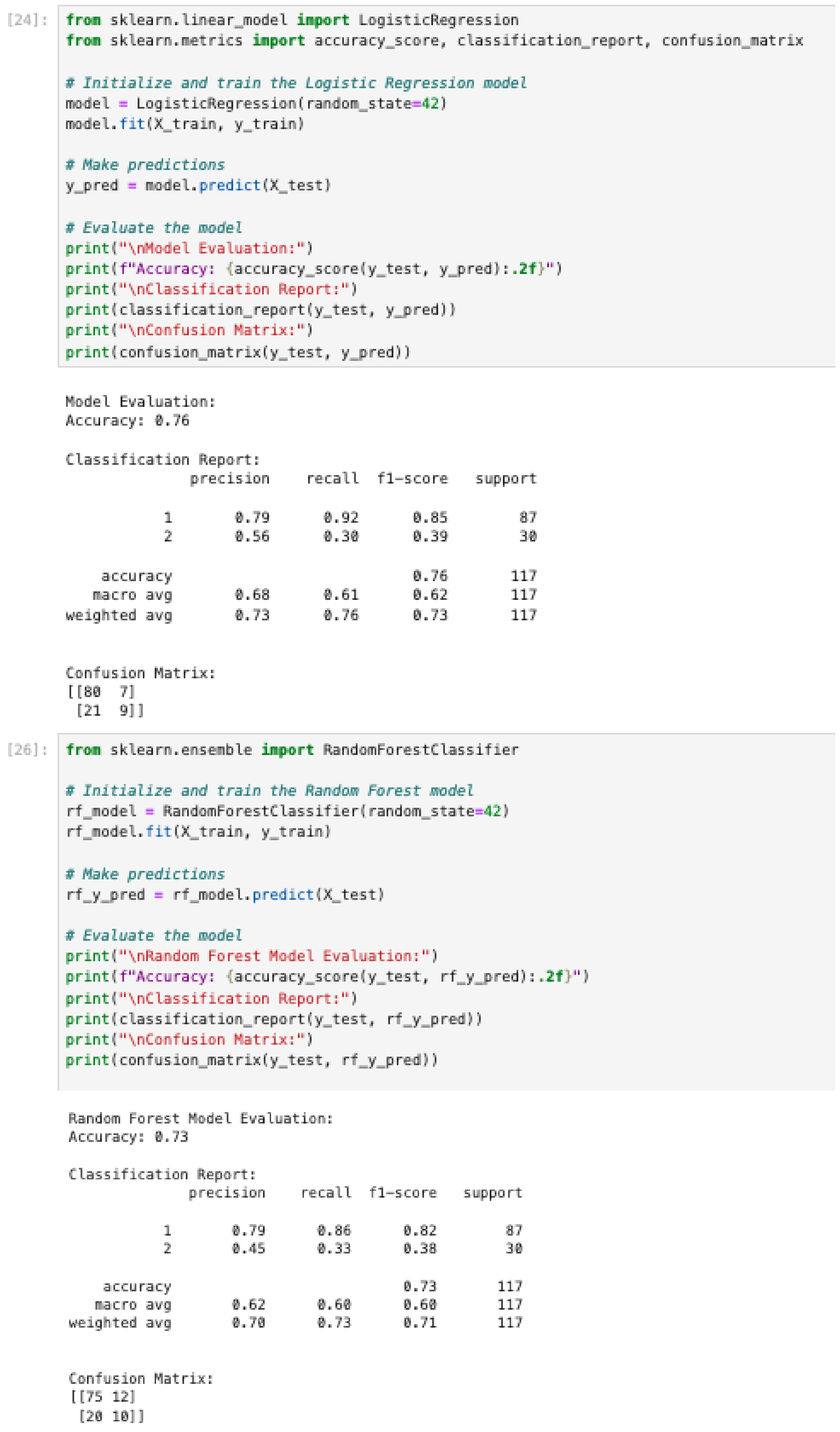
Application of RFC model on Indian Liver Patient Dataset

The Random Forest Classifier has shown promising results in the medical field, particularly in diagnosing liver diseases. We applied this on the Indian Liver Patient Dataset, with the data on various features such as age, gender, total bilirubin, direct bilirubin, and more.

We used RFC to predict whether a patient has a liver disease or not based on the given features. We trained the algorithm on a portion of the dataset and tested on the remaining data. The accuracy of the model was evaluated using metrics such as precision, recall, and F1 score.

The most important features for predicting liver disease were found to be Direct Bilirubin, Total Bilirubin, and Aspartate Aminotransferase.

The given code snippet shows how we applied rfc algorithm on the dataset we are working on:-



```
[24]: from sklearn.linear_model import LogisticRegression
      from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

      # Initialize and train the Logistic Regression model
      model = LogisticRegression(random_state=42)
      model.fit(X_train, y_train)

      # Make predictions
      y_pred = model.predict(X_test)

      # Evaluate the model
      print("\nModel Evaluation:")
      print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
      print("\nClassification Report:")
      print(classification_report(y_test, y_pred))
      print("\nConfusion Matrix:")
      print(confusion_matrix(y_test, y_pred))
```

```
Model Evaluation:
Accuracy: 0.76

Classification Report:
              precision    recall  f1-score   support

           1       0.79      0.92      0.85        87
           2       0.56      0.30      0.39        30

    accuracy                           0.76       117
   macro avg       0.68      0.61      0.62       117
weighted avg       0.73      0.76      0.73       117


Confusion Matrix:
[[80  7]
 [21  9]]
```

```
[26]: from sklearn.ensemble import RandomForestClassifier

      # Initialize and train the Random Forest model
      rf_model = RandomForestClassifier(random_state=42)
      rf_model.fit(X_train, y_train)

      # Make predictions
      rf_y_pred = rf_model.predict(X_test)

      # Evaluate the model
      print("\nRandom Forest Model Evaluation:")
      print(f"Accuracy: {accuracy_score(y_test, rf_y_pred):.2f}")
      print("\nClassification Report:")
      print(classification_report(y_test, rf_y_pred))
      print("\nConfusion Matrix:")
      print(confusion_matrix(y_test, rf_y_pred))
```

```
Random Forest Model Evaluation:
Accuracy: 0.73

Classification Report:
              precision    recall  f1-score   support

           1       0.79      0.86      0.82        87
           2       0.45      0.33      0.38        30

    accuracy                           0.73       117
   macro avg       0.62      0.60      0.60       117
weighted avg       0.70      0.73      0.71       117


Confusion Matrix:
[[75 12]
 [20 10]]
```

Interpreting the Confusion Matrix:-

The confusion matrix is composed of four elements: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).

True positives (TP) represent the number of correctly classified instances.

True negatives (TN) represent the number of correctly predicted instances.

False positives (FP) represent the number of incorrectly classified instances.

False negatives (FN) represent the number of instances that were not correctly predicted.

Precision

It measures the proportion of true positive predictions among all positive predictions made by the model. In simpler terms, precision assesses the model's ability to avoid false positives.

A high precision value indicates that the model has a low rate of false positives, meaning it is reliable when it predicts a positive outcome. However, precision alone may not provide a complete picture of a model's performance, as it does not consider false negatives. It is often used in conjunction with other metrics such as recall, F1 score, and accuracy to get a more comprehensive evaluation of a model's effectiveness in classification tasks.

Accuracy

The accuracy of the model can be calculated using the following formula: accuracy = (TP + TN) / (TP + TN + FP + FN).
The accuracy of the model is the percentage of correctly classified instances out of all instances. The output of the code will display the accuracy score for both the training and testing datasets, allowing for the evaluation of the model's performance.

We achieved 63% of accuracy.

Feature Importance

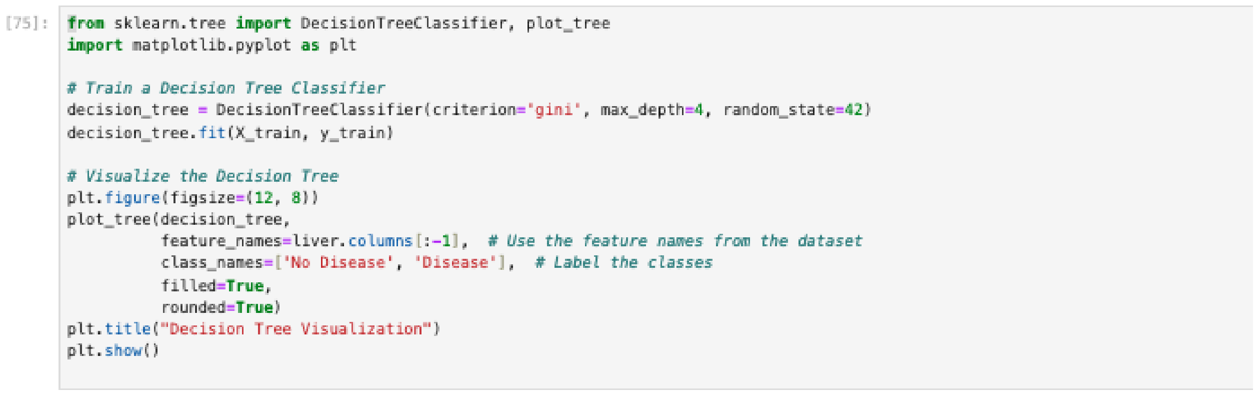
The Random Forest Classifier also provides information on the importance of each feature in the dataset. This information can be used to identify which features are most relevant in predicting the target variable. The output of the code will display a graph showing the relative importance of each feature.

Overall, the output of the Random Forest Classifier code provides valuable information for evaluating the performance of the model and identifying important features in the dataset.

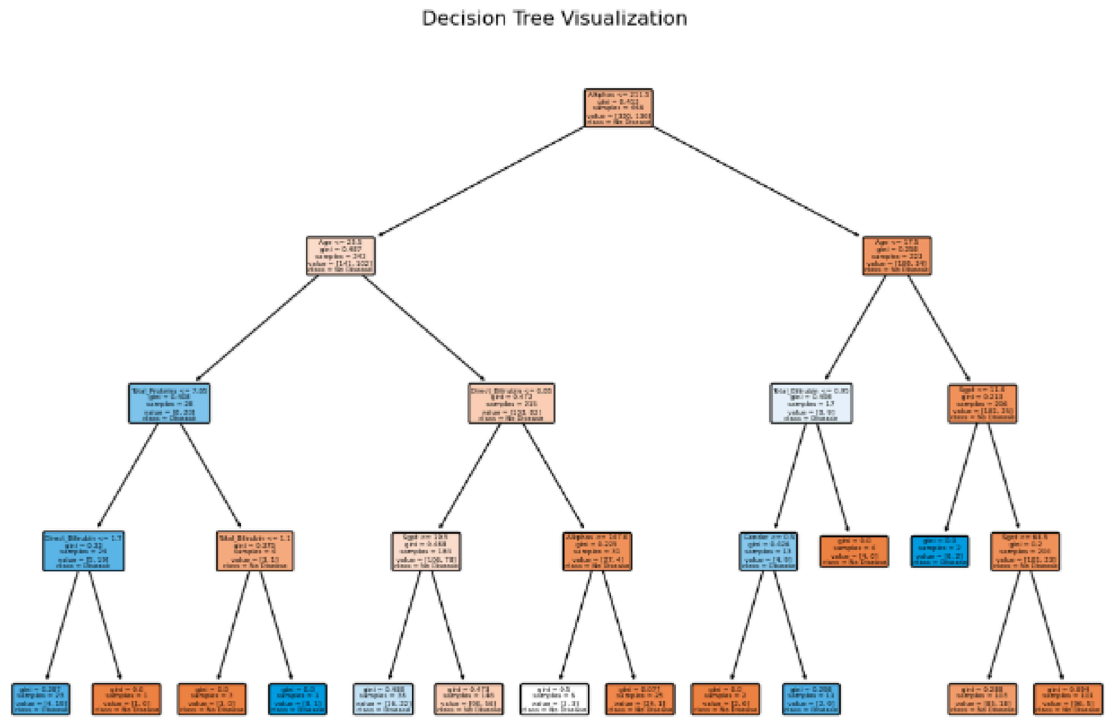
DECISION TREE:-

Decision tree visualization is an important tool to understand the inner workings of the Random Forest Classifier. It helps in understanding how the algorithm makes decisions and how it classifies data points.

The decision tree can be visualized using the plot_tree() function in the scikit-learn library. This function generates a tree diagram that shows the hierarchical structure of the decision-making process.The code is given as:



```
[75]: from sklearn.tree import DecisionTreeClassifier, plot_tree
      import matplotlib.pyplot as plt

      # Train a Decision Tree Classifier
      decision_tree = DecisionTreeClassifier(criterion='gini', max_depth=4, random_state=42)
      decision_tree.fit(X_train, y_train)

      # Visualize the Decision Tree
      plt.figure(figsize=(12, 8))
      plot_tree(decision_tree,
                feature_names=liver.columns[:-1],  # Use the feature names from the dataset
                class_names=['No Disease', 'Disease'],  # Label the classes
                filled=True,
                rounded=True)
      plt.title("Decision Tree Visualization")
      plt.show()
```



Why we used Random Forest Algorithm over Other Classifiers

– Support Vector Machine (SVM)

SVM is a popular classifier used for both classification and regression analysis. It works by finding the hyperplane that maximally separates the classes in the feature space. SVM can be used with both linear and nonlinear kernels, making it versatile for different types of datasets. However, SVM can be computationally expensive and sensitive to the choice of kernel and parameters.

– K-Nearest Neighbors (KNN)

KNN is a non-parametric classifier that works by finding the k-nearest neighbors to a given point in the feature space and assigning the class based on the majority vote of those neighbors. KNN can be effective for small datasets and nonlinear decision boundaries but can be sensitive to the choice of k and the distance metric used.

INFERENCES DRAWN

- More men (75.82%) than women (24.18%) in the dataset, raising questions about gender-based differences in susceptibility to liver diseases.

- Majority of liver diseases observed in individuals aged 30 to 65, prompting investigation into lifestyle factors contributing to vulnerability.

- Variations in vulnerability between men and women at different ages, sparking curiosity about potential hormonal or behavioral factors influencing susceptibility.

- Normal distribution of Total Bilirubin and Direct Bilirubin in liver fluids, prompting questions about their interactions and potential as indicators of a healthy liver.

- Use of Logistic Regression and Random Forest for predicting liver diseases highlights the need for real-world testing to ensure model reliability.

- Observations serve as a bridge between theoretical concepts and practical considerations for healthcare and disease prevention.

Thus, further research and testing are essential to fully understand the complexities of liver diseases and their susceptibility factors.

Questions:-

Concerns About the Data:

- Many people have expressed concerns about the missing values in the Albumin-Globulin Ratio and how it may affect decision making.

- Questions have been raised about whether these missing values should be imputed or excluded in further analyses.

- The lack of a linear correlation between liver fluids has raised questions and sparked inquiries into the independent roles of these fluids and their significance in liver health.

- There is still ongoing inquiry into the societal, lifestyle, and genetic factors contributing to the dynamics of age and gender in relation to liver diseases.

- The concerns revolve around the clinical relevance and interpretability of features identified by the Random Forest Classifier in ML models.

Questions Addressed by the Analysis:

- The analysis provides insights into the distribution of both diagnosed and undiagnosed cases, shedding light on the prevalence of liver diseases.

- The frequency of liver diseases is addressed and questions are answered through this analysis.

- A better understanding of demographic patterns is gained through the analysis of the impact of age and gender on liver disease diagnosis.

- The use of Logistic Regression and Random Forest models addresses questions about the feasibility of using machine learning for diagnosis, providing a quantitative understanding of disease probabilities.

- Analysis helps to understand potential interdependencies between liver fluids and addresses questions about how these fluids may affect liver function.

- Studying feature importance in ML models guides future analyses and decisions, addressing questions about the critical factors influencing disease prediction.

Conclusion

This report delves into the complex landscape of liver health in India, exploring it through the lens of the "Indian Liver Patient Database." By utilizing Machine Learning techniques, specifically Logistic Regression and Random Forest, the study not only aims to accurately classify diseases but also delves into the examination of crucial factors that impact liver health.

This study investigates liver health in India through the Indian Liver Patient Database. By applying ML techniques like Logistic Regression and Random Forest, it identifies disease patterns and contributes to targeted healthcare strategies. The integration of ML with data analytics enhances the understanding of liver disease, supporting improved patient outcomes and public health policies.

By identifying patterns within the Indian population, this research aids in formulating effective interventions and shaping healthcare policies that align with the unique challenges posed by liver diseases in a diverse and densely populated country like India. The fusion of machine learning methodologies with comprehensive data analysis opens new avenues for enhancing patient outcomes, advancing medical research, and ultimately addressing the distinctive nuances of liver health in the Indian context.

References

1. UCI Machine Learning Repository. (2012). *Indian Liver Patient Dataset (ILPD)*. Retrieved from https://archive.ics.uci.edu/dataset/225/ilpd+indian+liver+patient+dataset

2. Dr. Manoj Kumar *Descriptive and Inferential Statistics PPT*. Google Classroom.

3. Dr. Aloke Dutta. *Class Notes*(*Classification Models).*

4. ***An introduction to seaborn — seaborn 0.13.2 documentation***

5. ***Tutorials — Matplotlib 3.1.2 documentation***

6. Project's GitHub Repository