MINISTRY OF EDUCATION
IMAM ABDULRAHMAN BIN
FAISAL UNIVERSITY
COLLEGE OF COMPUTER SCIENCE
& INFORMATION TECHNOLOGY

وزارة التعليم
جامعة الإمام
عبدالرحمن بن فيصل
كلية علوم الحاسب
وتقنية المعلومات

جامعة الإمام عبد الرحمن بن فيصل
IMAM ABDULRAHMAN BIN FAISAL UNIVERSITY

PROJECT INFORMATION

COURSE TITLE	DATA MINING & DATA WAREHOUSING
COURSE CODE	CIS 517
INSTRUCTOR NAME	MS. RAWAN ALGHAMDI

Team Contribution Table:

Group 9FS2			
Project Team #1			
Name	**ID**	**Task Contribution**	**Description Of Work**
Hawra Alsedrah	2210003421	Task 3	The entirety of task 3, this includes all codes provided for describing and visualizing categorical and numerical attributes.
Zahra Aleid	2210002616	Task 4	The entirety of task 4 in regard to analyzing the data and writing extra sections.
Jenan Albuzaid	2210003173	Task 1	The entirety of task 1 in regard to detecting and removing outliers.
Rose Hummusani	2210002644	Task 5	
Muneerh Alfaleh	2210002858	Task 5	
Zainab Al mousa	2210003212	Task 2	The entirety of task 2, this included preprocessing and standardizing the dataset.
Alaa	2210002443	Task 6	The entirety of task 4 in regard to analyzing and comparing performances of the different models.

1. Dataset Overview

The liver health dataset includes clinical and laboratory data of a given set of persons for the evaluation of their liver status/health. It acts as a source to undertake and understand the epidemiology of liver diseases and even perfect models aimed at assessing liver related ailments.

Purpose

This dataset has been created with the following objectives:

Classification: To determine if the person is a liver patient (1) or does not suffer from a liver patient (2) The feature sets provided assess the individual for a liver state.

Exploratory Data Analysis (EDA): It seeks to determine the association between clinical measures and the state of the liver/body.

Feature Analysis: To determine the effect of various measurement variables on the diagnosis of liver diseases.

Key Highlights

Multivariate Dataset: Contains data constructed from an assortment of numerical and categorical features hence facilitating the performance of advanced analysis/machine learning.

Health Focus: all features are primarily oriented in relation to liver function such as liver enzymes, bilirubin levels and protein ratios.

Real-World Relevance: The dataset reflects the very basic realities of clinical practice which enhances its utilization in medicine and teaching.

Applications

Medical Diagnosis: incorporate medical science practitioner knowledge with machine learning models in diagnosing various liver diseases.

Feature Importance: Examine how specific characteristics (such as bilirubin and enzyme levels) affect liver health.

Educational Use: Present data mining, categorization, and healthcare analytics principles to the students.

Overall, this dataset offers a thorough basis for investigating liver health using data analysis and machine learning methods.

Code Explanation

At the beginning, we started with reading the dataset and getting the information about the dataset columns:

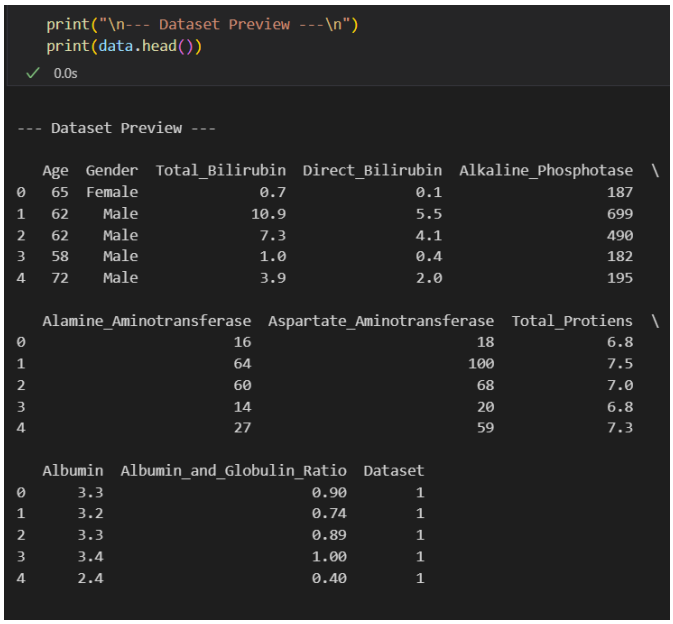
```python
import pandas as pd

# read the dataset
data = pd.read_csv('indian_liver_patient.csv')

# Print the information of our dataset
print(data.info())
```

✓ 4.1s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Age                         583 non-null    int64
 1   Gender                      583 non-null    object
 2   Total_Bilirubin             583 non-null    float64
 3   Direct_Bilirubin            583 non-null    float64
 4   Alkaline_Phosphotase        583 non-null    int64
 5   Alamine_Aminotransferase    583 non-null    int64
 6   Aspartate_Aminotransferase  583 non-null    int64
 7   Total_Protiens              583 non-null    float64
 8   Albumin                     583 non-null    float64
 9   Albumin_and_Globulin_Ratio  579 non-null    float64
 10  Dataset                     583 non-null    int64
dtypes: float64(5), int64(5), object(1)
memory usage: 50.2+ KB
None
```

Figure 1: Reading and getting info from the dataset

After that, we got an overview of the first few rows of the dataset.



```
print("\n--- Dataset Preview ---\n")
print(data.head())
```

✓ 0.0s

```
--- Dataset Preview ---

   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
```

Figure 2: Dataset preview

Then, we got a statistic summary about our dataset by writing the following line of code:

```
print("\n--- Summary Statistics ---\n")
print(data.describe())
```
✓ 0.0s

```
--- Summary Statistics ---

              Age  Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase  \
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

       Alamine_Aminotransferase  Aspartate_Aminotransferase  Total_Protiens  \
count                583.000000                  583.000000      583.000000
mean                  80.713551                  109.910806        6.483190
std                  182.620356                  288.918529        1.085451
min                   10.000000                   10.000000        2.700000
25%                   23.000000                   25.000000        5.800000
50%                   35.000000                   42.000000        6.600000
75%                   60.500000                   87.000000        7.200000
max                 2000.000000                 4929.000000        9.600000

          Albumin  Albumin_and_Globulin_Ratio     Dataset
count  583.000000                 579.000000  583.000000
...
25%      2.600000                   0.700000    1.000000
50%      3.100000                   0.930000    1.000000
75%      3.800000                   1.100000    2.000000
max      5.500000                   2.800000    2.000000
```

Figure 3: Dataset description

We checked if the dataset contained missing values by writing the following:

```
print("\n--- Missing Values ---\n")
print(data.isnull().sum())
```
✓ 0.0s

```
--- Missing Values ---

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
```

Figure 4: Finding missing values

To determine the type for each column, we used the following code:

```
print("\n--- Data Types ---\n")
print(data.dtypes)
```
✓ 0.0s

```
--- Data Types ---

Age                              int64
Gender                          object
Total_Bilirubin                float64
Direct_Bilirubin               float64
Alkaline_Phosphotase             int64
Alamine_Aminotransferase         int64
Aspartate_Aminotransferase       int64
Total_Protiens                 float64
Albumin                        float64
Albumin_and_Globulin_Ratio     float64
Dataset                          int64
dtype: object
```

Figure 5: Columns data type

We split the dataset into training and testing for model training and evaluation. X represents all the dataset inputs except the 'Dataset' column, and y represents the 'Dataset' column.

```
from sklearn.model_selection import train_test_split
# Separate features (X) and target (y)
X = data.drop(columns=['Dataset'])  # Features
y = data['Dataset']  # Target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Display the shape of the splits
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```
✓ 107m 49.2s

```
((466, 10), (117, 10), (466,), (117,))
```

Figure 6: Train/Test split

Lastly, we displayed a plot showing the comparison between the number of liver patients and non-liver patients.

Figure 7: Plot for parient comparison

2. Data Preprocessing

Missing data

Missing data occurs when one or more features (columns) in a dataset lack values. This issue can arise due to several reasons, including:

- **Errors in Data Entry**: Mistakes made while manually inputting data.
- **Non-Response in Surveys**: Participants failing to provide answers to specific questions.
- **Technical Issues**: System malfunctions or data loss during collection processes.
- **Optional Fields**: Respondents skipping optional questions in surveys or forms.

This Figure shows that we have 4 missing values in (Albumi_and_Globulin_Ratio) column

```
missing_data= data.isnull().sum()
print (missing_data)
```

```
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
```

Figure 8: missing data

We used imputation to fill missing values by replacing them with a specific value, such as 0 for numerical columns. The figure below shows the code used to fill the missing values and the result after handling.

```
# Fill missing values based on column data types
for column in data.columns:
    if data[column].dtype in ['int64', 'float64']:  # Numerical columns
        data[column] = data[column].fillna(0)

# Check if all missing values are handled
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   0
Dataset                      0
dtype: int64
```

Figure 9: handling missing data

Duplicates

This figure below showes that we dont have any duplicated rows that can effect the analysis and model training.

```
print (f"number of duplicate rows: {data.duplicated().sum()}")
```

```
number of duplicate rows: 0
```

```
# remove duplicates if any
data = data.drop_duplicates()
```

Figure 10: handling missing data

Cleanup

To ensures that column names are clean and prevents errors when referencing them we clead the data.

```
data.columns = data.columns.str.strip()

print("\ncolmn names after cleaning: ")
print(data.columns)
```

```
colmn names after cleaning:
Index(['Age', 'Gender', 'Total_Bilirubin', 'Direct_Bilirubin',
       'Alkaline_Phosphotase', 'Alamine_Aminotransferase',
       'Aspartate_Aminotransferase', 'Total_Protiens', 'Albumin',
       'Albumin_and_Globulin_Ratio', 'Dataset'],
      dtype='object')
```

Figure 11: Cleanup data from extra space

Encode categorical values

It is an essential preprocessing step in machine learning because most models cannot work directly with non-numeric (categorical) data. The figure below shows the code and the results.

```
# Encode categorical variables
label_encoder = LabelEncoder()
data['Gender'] = label_encoder.fit_transform(data['Gender'])
print("\nFirst 5 rows after encoding categorical columns:")
print(data.head())
```

```
First 5 rows after encoding categorical columns:
   Age  Gender  Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase  \
0   65       0              0.7               0.1                   187
1   62       1             10.9               5.5                   699
2   62       1              7.3               4.1                   490
3   58       1              1.0               0.4                   182
4   72       1              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
```

Figure 13: encode categorical values

Standardize Numerical Features

Normalizing values guarantees that all features are on an equal scale, stopping larger values from overshadowing the model. It enhances training stability, accelerates convergence, and boosts accuracy in algorithms sensitive to scale, such as SVM or neural networks. The figure below shows the code and the results.

```
# Normalize/Standardize Numerical Features
scaler = StandardScaler()
numerical_cols = ['Age', 'Total_Bilirubin', 'Direct_Bilirubin', 'Alkaline_Phosphotase',
                  'Alamine_Aminotransferase', 'Aspartate_Aminotransferase', 'Total_Protiens',
                  'Albumin', 'Albumin_and_Globulin_Ratio']
# Apply standardization to the numerical columns
data[numerical_cols] = scaler.fit_transform(data[numerical_cols])
print("\nFirst 5 rows after scaling numerical features:")
print(data.head())
```

```
First 5 rows after scaling numerical features:
        Age  Gender  Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase  \
0  1.241741       0        -0.418647         -0.493702             -0.427421
1  1.056874       1         1.210111          1.413923              1.661722
2  1.056874       1         0.635255          0.919354              0.808927
3  0.810385       1        -0.370743         -0.387723             -0.447823
4  1.673096       1         0.092336          0.177500             -0.394778

   Alamine_Aminotransferase  Aspartate_Aminotransferase  Total_Protiens  \
0                 -0.351482                   -0.314428        0.279290
1                 -0.086746                   -0.032278        0.923059
2                 -0.108807                   -0.142385        0.463224
3                 -0.362513                   -0.307546        0.279290
4                 -0.290813                   -0.173353        0.739125

    Albumin  Albumin_and_Globulin_Ratio  Dataset
0  0.189737                   -0.126100        1
```

Figure 14: standardize numerical feature

3. Statistical Methods

Numeric Attributes

The code in the figure below generates a statistical summary for all numeric columns in numeric_df which includes count, mean, standard deviation, min-max, and the 25th, 50th, and 75th percentiles.

```
# Descriptive statistics for numeric attributes
print("\nDescriptive statistics for numeric attributes:")
print(data[numerical_cols].describe())
```

```
Descriptive statistics for numeric attributes:
                Age  Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase  \
count  5.700000e+02     5.700000e+02      5.700000e+02          5.700000e+02
mean   6.232831e-18    -6.232831e-18     -1.246566e-17          4.986265e-17
std    1.000878e+00     1.000878e+00      1.000878e+00          1.000878e+00
min   -2.517211e+00    -4.665521e-01     -4.937018e-01         -9.333852e-01
25%   -7.301685e-01    -4.026792e-01     -4.583754e-01         -4.723049e-01
50%    9.297379e-03    -3.707428e-01     -4.230490e-01         -3.417335e-01
75%    8.103855e-01    -1.152513e-01     -6.978512e-02          2.549865e-02
max    2.782295e+00     1.144574e+01      6.430271e+00          7.419106e+00

       Alamine_Aminotransferase  Aspartate_Aminotransferase  Total_Protiens  \
count              5.700000e+02                5.700000e+02    5.700000e+02
mean              -2.493132e-17               -6.232831e-18   -4.736952e-16
std                1.000878e+00                1.000878e+00    1.000878e+00
min               -3.845741e-01               -3.419546e-01   -3.491362e+00
25%               -3.128747e-01               -2.903418e-01   -6.403815e-01
50%               -2.466906e-01               -2.352881e-01    9.535537e-02
75%               -1.088070e-01               -7.786897e-02    6.471580e-01
max                1.059096e+01                1.658361e+01    2.854369e+00

            Albumin  Albumin_and_Globulin_Ratio
count  5.700000e+02                5.700000e+02
mean  -3.864355e-16                2.742446e-16
std    1.000878e+00                1.000878e+00
min   -2.824907e+00               -2.870652e+00
25%   -6.895338e-01               -7.360000e-01
50%   -6.148288e-02               -3.461452e-02
75%    8.177884e-01                4.838008e-01
max    2.953162e+00                5.667954e+00
```

Figure 15 Descriptive statistics for numeric attributes

The code in the below figure provides more details of the mean, median, and standard deviation.

```python
# Another summary statistics of the dataset
print("Mean:\n", data[numerical_cols].mean())
print("\nMedian:\n", data[numerical_cols].median())
print("\nStandard Deviation:\n", data[numerical_cols].std())
```

```
Mean:
 Age                          6.232831e-18
Total_Bilirubin              -6.232831e-18
Direct_Bilirubin             -1.246566e-17
Alkaline_Phosphotase          4.986265e-17
Alamine_Aminotransferase     -2.493132e-17
Aspartate_Aminotransferase   -6.232831e-18
Total_Protiens               -4.736952e-16
Albumin                      -3.864355e-16
Albumin_and_Globulin_Ratio    2.742446e-16
dtype: float64

Median:
 Age                           0.009297
Total_Bilirubin              -0.370743
Direct_Bilirubin             -0.423049
Alkaline_Phosphotase         -0.341733
Alamine_Aminotransferase     -0.246691
Aspartate_Aminotransferase   -0.235288
Total_Protiens                0.095355
Albumin                      -0.061483
Albumin_and_Globulin_Ratio   -0.034615
dtype: float64
```

Figure 16 Mean, median, and standard deviation

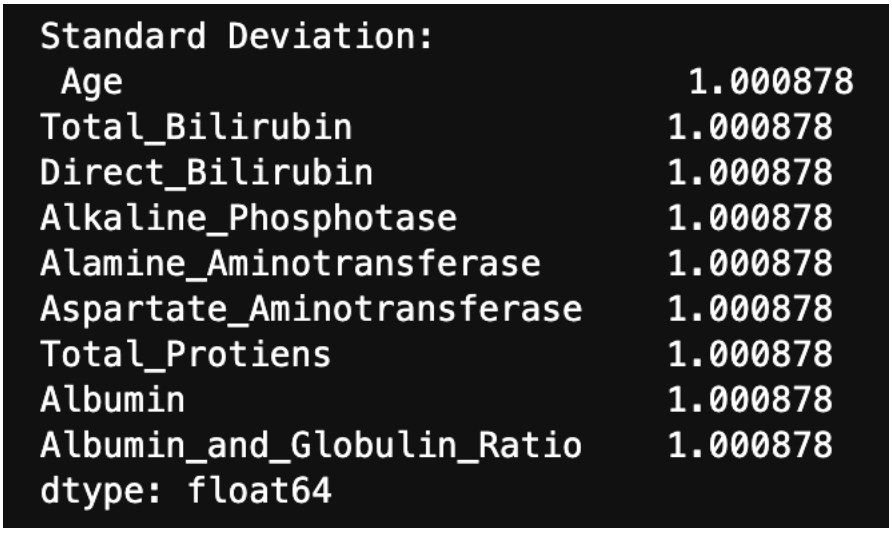


Figure 17 Mean, median, and standard deviation output

Visualizing Numeric Attributes

The code in the figure below generates a boxplot for each numeric attribute in the dataset.

```
# Boxplot for numeric attributes
print("\nVisualizing numeric attributes with boxplots:")
for column in data[numerical_cols].columns:
    plt.figure(figsize=(4, 2))
    sns.boxplot(y=numeric_df[column])
    plt.title(f"Boxplot for {column}")
    plt.show()
```

Figure 18 Visualizing numeric attributes using boxplot

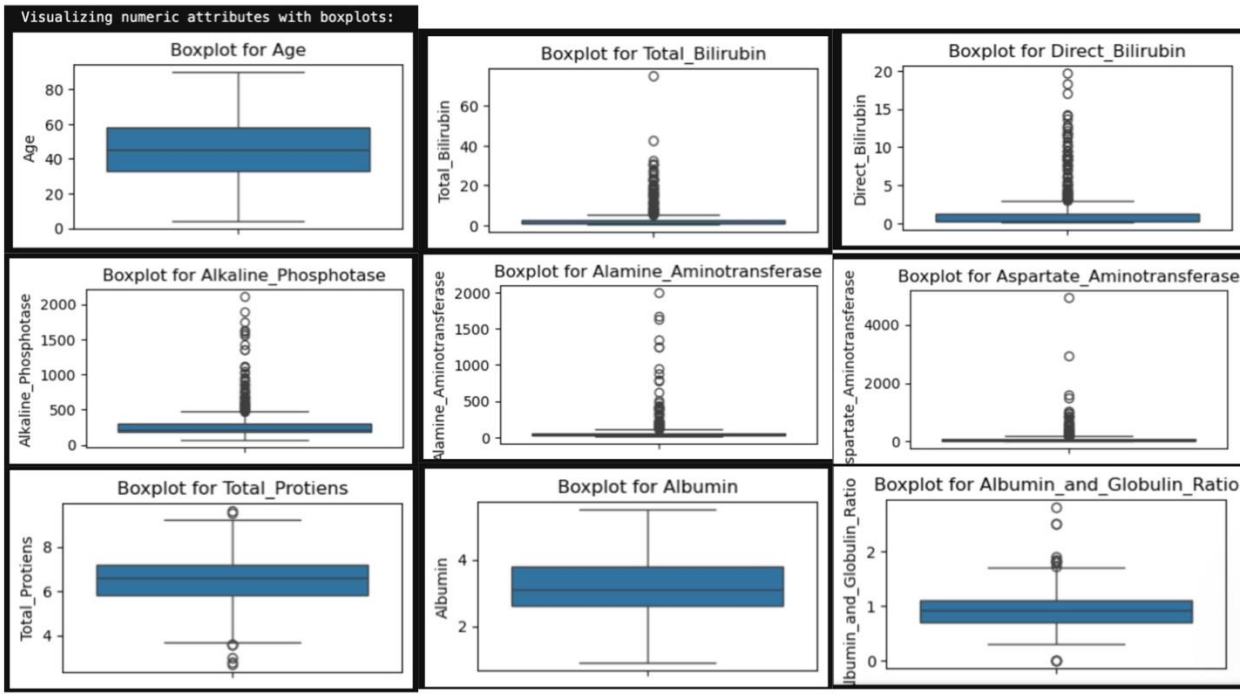


Figure 19 Combined boxplots output

The code in the below figure generates a correlation heatmap to visualize the numeric attributes.

```
# Correlation heatmap of numeric features
plt.figure(figsize=(6, 3))
sns.heatmap(data[numerical_cols].corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Feature Correlation")
plt.show()
```

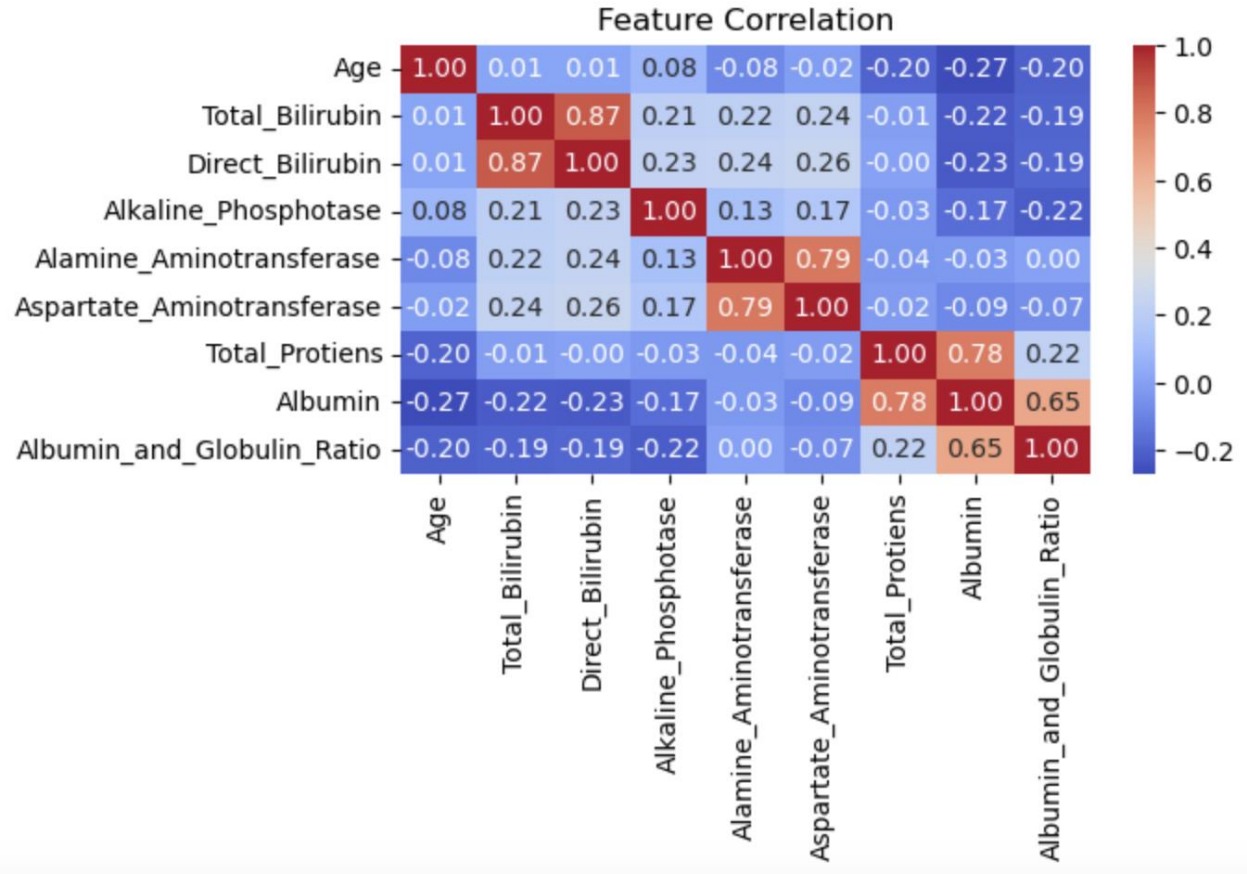


Figure 20 Correlation heatmap

Correlation Matrix

The code in the figure below generates a correlation matrix for the numeric attributes in the dataset.

```
# Correlation matrix for numeric columns
print("\nCorrelation matrix for numeric attributes:")
print(data[numerical_cols].corr())
```

```
Correlation matrix for numeric attributes:
                                Age  Total_Bilirubin  Direct_Bilirubin  \
Age                        1.000000         0.011500          0.007050
Total_Bilirubin            0.011500         1.000000          0.874116
Direct_Bilirubin           0.007050         0.874116          1.000000
Alkaline_Phosphotase       0.081673         0.206239          0.234609
Alamine_Aminotransferase  -0.083383         0.217471          0.237450
Aspartate_Aminotransferase -0.016753        0.238678          0.258489
Total_Protiens            -0.197052        -0.008588         -0.000875
Albumin                   -0.271170        -0.224124         -0.230751
Albumin_and_Globulin_Ratio -0.202579       -0.193647         -0.187039

                           Alkaline_Phosphotase  Alamine_Aminotransferase  \
Age                                    0.081673                 -0.083383
Total_Bilirubin                        0.206239                  0.217471
Direct_Bilirubin                       0.234609                  0.237450
Alkaline_Phosphotase                   1.000000                  0.126830
Alamine_Aminotransferase               0.126830                  1.000000
Aspartate_Aminotransferase             0.167230                  0.791857
Total_Protiens                        -0.030048                 -0.035193
Albumin                               -0.168318                 -0.027973
Albumin_and_Globulin_Ratio            -0.219500                  0.000241
```

Figure 21 Correlation matrix

	Aspartate_Aminotransferase	Total_Protiens	\
Age	-0.016753	-0.197052	
Total_Bilirubin	0.238678	-0.008588	
Direct_Bilirubin	0.258489	-0.000875	
Alkaline_Phosphotase	0.167230	-0.030048	
Alamine_Aminotransferase	0.791857	-0.035193	
Aspartate_Aminotransferase	1.000000	-0.022000	
Total_Protiens	-0.022000	1.000000	
Albumin	-0.085180	0.784731	
Albumin_and_Globulin_Ratio	-0.065676	0.222876	

	Albumin	Albumin_and_Globulin_Ratio
Age	-0.271170	-0.202579
Total_Bilirubin	-0.224124	-0.193647
Direct_Bilirubin	-0.230751	-0.187039
Alkaline_Phosphotase	-0.168318	-0.219500
Alamine_Aminotransferase	-0.027973	0.000241
Aspartate_Aminotransferase	-0.085180	-0.065676
Total_Protiens	0.784731	0.222876
Albumin	1.000000	0.651749
Albumin_and_Globulin_Ratio	0.651749	1.000000

Figure 22 Correlation matrix output

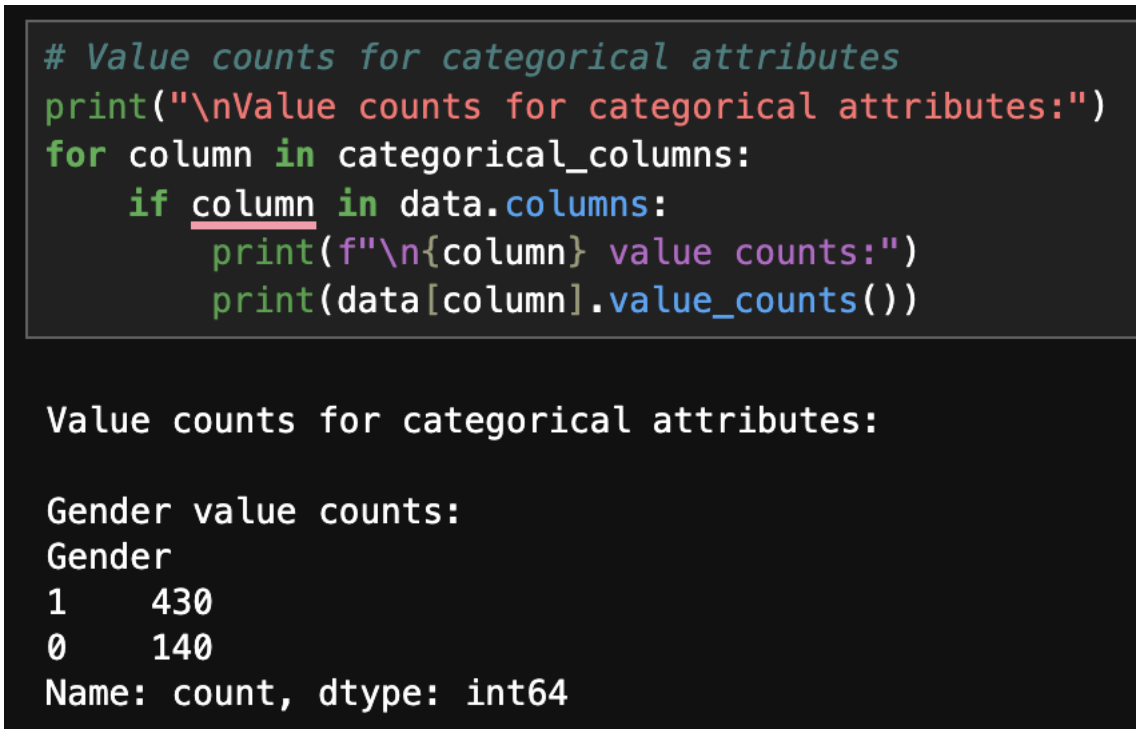
Categorical Attributes

The code in the figure below provides the count for both genders in the dataset, before standardization we could've used the function value_counts(), but due to it gender is now being treated as an integer, therefore, using these methods will provide us with the same output as the mentioned function. Note that 1 refers to male and 0 to female.

```
# Descriptive statistics for categorical attributes
categorical_columns = ['Gender']
print("\nDescriptive statistics for categorical attributes:")
for column in categorical_columns:
    if column in data.columns:
        print(f"\nDescriptive statistics for {column}:")
        print(f"Count: {data[column].count()}")
        print(f"Unique: {data[column].nunique()}")
        print(f"Top: {data[column].mode()[0]}")
        print(f"Freq: {data[column].value_counts().iloc[0]}")
```

```
Descriptive statistics for categorical attributes:

Descriptive statistics for Gender:
Count: 570
Unique: 2
Top: 1
Freq: 430
```

Figure 23 Gender statistics



```
# Value counts for categorical attributes
print("\nValue counts for categorical attributes:")
for column in categorical_columns:
    if column in data.columns:
        print(f"\n{column} value counts:")
        print(data[column].value_counts())
```

```
Value counts for categorical attributes:

Gender value counts:
Gender
1    430
0    140
Name: count, dtype: int64
```

Figure 24 Gender value counts

Visualizing Categorical Attributes

The code in the figure below generates a count plot for the categorical attributes in the dataset, please note again that 1 refers to male and 0 for female.

```
# Visualizing value counts for categorical attributes
print("\nVisualizing value counts for categorical attributes:")
for column in categorical_columns:
    if column in data.columns:
        plt.figure(figsize=(4, 2))
        sns.countplot(x=data[column], order=data[column].value_counts().index)
        plt.title(f"Value Counts for {column}")
        plt.show()
```

```
Visualizing value counts for categorical attributes:
```

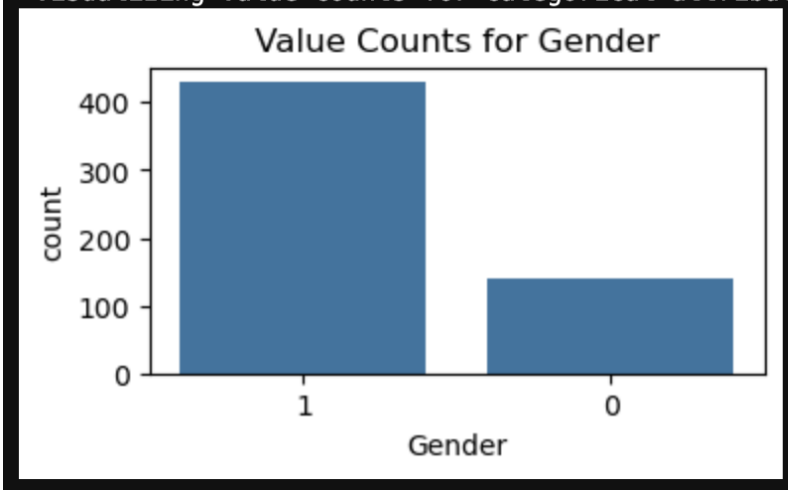


Figure 25 Visualizing value counts for gender

Handeling Outliers

Outliers were detetcted in figure 19 in { "Total_Bilirubin", "Direct_Bilirubin", "Alkaline_Phosphotase", "Alamine_Aminotransferase", "Aspartate_Aminotransferase", "Albumin_and_Globulin_Ratio" } boxplots. To visulize ouliers more clearly we viewd them using scatterplot as in fiqure 26, and we found that the highest outliers were in { "Total_Bilirubin", "Alamine_Aminotransferase", "Aspartate_Aminotransferase" } so we chose to handle them by using the IQR method.

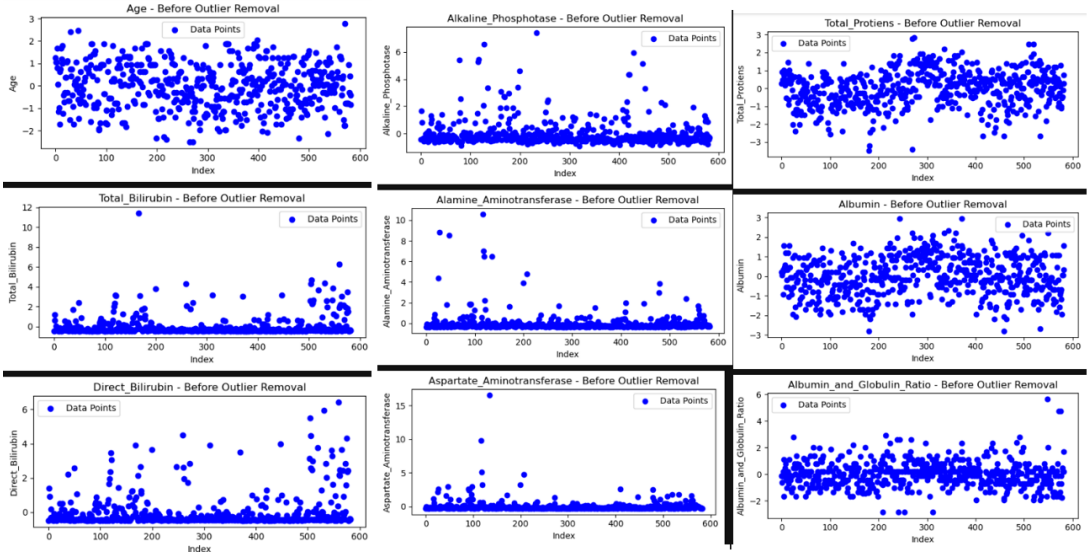


Figure 26 Scatter plot of numerical data

First, we defined an array called outliers_col to define the columns from which outliers should be removed. Then printed the dataset shape before and after removing the outliers to see the difference. In a for loop the IQR method was applied to remove outliers.

```
outliers_col = ['Total_Bilirubin', 'Alamine_Aminotransferase', 'Aspartate_Aminotransferase']

# Initial dataset shape
print("Initial Dataset Shape: ", data.shape)

# Initialize a copy of the data
data_no_outliers = data.copy()

for column in outliers_col:
    # Calculate Q1, Q3, and IQR
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1


    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Remove outliers
    data_no_outliers = data_no_outliers[
        (data_no_outliers[column] >= lower_bound) &
        (data_no_outliers[column] <= upper_bound)
    ]

# After IQR-based outlier removal
print("Dataset Shape After Outlier Removal: ", data_no_outliers.shape)

# Count rows removed
num_outliers_removed = data.shape[0] - data_no_outliers.shape[0]
print(f"Total number of rows removed due to outliers: {num_outliers_removed}")
```

```
Initial Dataset Shape:  (570, 11)
Dataset Shape After Outlier Removal:  (435, 11)
Total number of rows removed due to outliers: 135
```

Figure 27 removing outliers using IQR

Visualizing Outliers After Removal

We used scatter plot to compare the differences between the outliers' presence and the result after removing them.

```python
for column in outliers_col:

    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    fig, ax = plt.subplots(1, 2, figsize=(14, 5))

    # Before outliers removal
    ax[0].scatter(data.index, data[column], c='blue', label='Data Points')
    ax[0].set_title(f'{column} - Before Outlier Removal')
    ax[0].set_xlabel('Index')
    ax[0].set_ylabel(column)
    ax[0].legend()

    # After outliers removal
    ax[1].scatter(data_no_outliers.index, data_no_outliers[column], c='blue', label='Cleaned Data')
    ax[1].set_title(f'{column} - After Outlier Removal')
    ax[1].set_xlabel('Index')
    ax[1].set_ylabel(column)
    ax[1].legend()

    plt.tight_layout()
    plt.show()
```

Figure 28 visualizing outliers

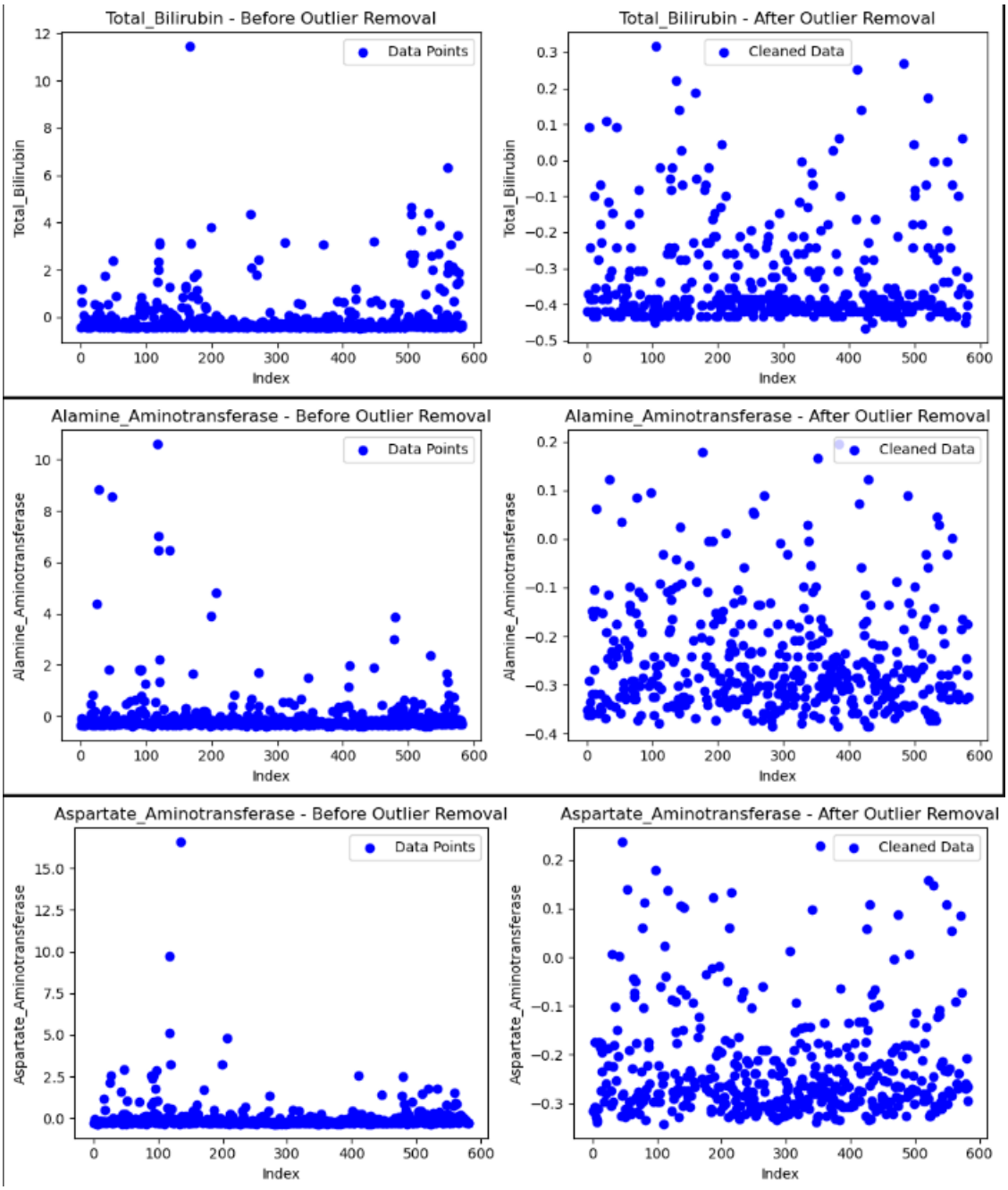


Figure 29 Scatter plot of before and after outliers

4. Performance Comparison

The following code is meant to print information about each model including the Confusion Matrix, the Classification Report, and Accuracy.

```
[91]: #task 5

[97]: # Results for Logistic Regression
print("Logistic Regression Classification Report:")
print(classification_report(y_test, lrPredict))
print("Logistic Regression Confusion Matrix:")
print(confusion_matrix(y_test, lrPredict))
print(f"Logistic Regression Accuracy: {accuracy_score(y_test, lrPredict) * 100:.2f}%")
print("\n" + "-"*50)

# Results for Decision Tree
print("Decision Tree Classification Report:")
print(classification_report(y_test, dtPredict))
print("Decision Tree Confusion Matrix:")
print(confusion_matrix(y_test, dtPredict))
print(f"Decision Tree Accuracy: {accuracy_score(y_test, dtPredict) * 100:.2f}%")
print("\n" + "-"*50)

# Results for Random Forest
print("Random Forest Classification Report:")
print(classification_report(y_test, rfPredict))
print("Random Forest Confusion Matrix:")
print(confusion_matrix(y_test, rfPredict))
print(f"Random Forest Accuracy: {accuracy_score(y_test, rfPredict) * 100:.2f}%")
print("\n" + "-"*50)

# Accuracy comparison
print("Accuracy Comparison:")
print(f"Logistic Regression: {accuracy_score(y_test, lrPredict) * 100:.2f}%")
print(f"Decision Tree: {accuracy_score(y_test, dtPredict) * 100:.2f}%")
print(f"Random Forest: {accuracy_score(y_test, rfPredict) * 100:.2f}%")
```

Figure 30 performance comparison code

The output showed:

```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           1       0.77      0.87      0.82        85
           2       0.39      0.24      0.30        29

    accuracy                           0.71       114
   macro avg       0.58      0.56      0.56       114
weighted avg       0.67      0.71      0.69       114

Logistic Regression Confusion Matrix:
[[74 11]
 [22  7]]
Logistic Regression Accuracy: 71.05%

--------------------------------------------------
Decision Tree Classification Report:
              precision    recall  f1-score   support

           1       0.77      0.85      0.80        85
           2       0.35      0.24      0.29        29

    accuracy                           0.69       114
   macro avg       0.56      0.54      0.55       114
weighted avg       0.66      0.69      0.67       114

Decision Tree Confusion Matrix:
[[72 13]
 [22  7]]
Decision Tree Accuracy: 69.30%


--------------------------------------------------
Random Forest Classification Report:
              precision    recall  f1-score   support

           1       0.84      0.89      0.86        85
           2       0.61      0.48      0.54        29

    accuracy                           0.79       114
   macro avg       0.72      0.69      0.70       114
weighted avg       0.78      0.79      0.78       114

Random Forest Confusion Matrix:
[[76  9]
 [15 14]]
Random Forest Accuracy: 78.95%

--------------------------------------------------
Accuracy Comparison:
Logistic Regression: 71.05%
Decision Tree: 69.30%
Random Forest: 78.95%
```

Figure 11 Performance Comparison Run

Performance Analysis

Analyzing the result, Random Forest has the highest accuracy of 78.95%, followed by Logistic Regression with an accuracy of 71.05%, and then Decision Tree with 69.30%.

Looking into the classification report in detail, we find that the positive class (1) indicating the presence of liver disease is comparatively well performed by both the algorithms, Logistic Regression and Decision Tree, but there lies a huge difference in its recall. Logistic Regression achieves a recall of 0.87, while Decision Tree has a lower recall of 0.54, suggesting that the Logistic Regression model is more effective at identifying all positive cases. In contrast, Random Forest surpasses both models with a higher recall of 0.89 and precision of 0.84.

For the negative class (2), which indicates the absence of liver disease, Random Forest also demonstrates superior precision and recall compared to Decision Tree. Therefore, Random Forest is more dependable in predicting negative cases while accurately identifying positive ones. This is further supported by the confusion matrix, which illustrates the number of true positives—actual liver disease cases correctly identified—and false negatives, which represent instances where liver disease was not predicted at all.

Ultimately, the Random Forest model presents a better balance of precision and recall. It is suitable in a medical context since it reduces the number of false negatives, i.e., the number of patients with liver disease who are misclassified as healthy.