

1. INTRODUCTION

1.1 Project Overview

1.2 Purpose

2. IDEATION PHASE

2.1 Problem Statement

2.2 Empathy Map Canvas

2.3 Brainstorming

3. REQUIREMENT ANALYSIS

3.1 Customer Journey Map

3.2 Solution Requirement

3.3 Data Preparation & Flow Diagram

3.4 Technology Stack

4. PROJECT DESIGN

4.1 Problem Solution Fit

4.2 Proposed Solution

4.3 Solution Architecture

5. PROJECT PLANNING & SCHEDULING

5.1 Project Planning

6. FUNCTIONAL AND PERFORMANCE TESTING

6.1 Performance Testing

7. RESULTS

7.1 Output Screenshots

8. ADVANTAGES & DISADVANTAGES

9. CONCLUSION

10. FUTURE SCOPE

11. APPENDIX

Source Code (if any)

Dataset Link

GitHub & Project Demo Link

REVOLUTIONIZING LIVER CARE

Predicting Liver Cirrhosis using Advanced Machine Learning Techniques

1. INTRODUCTION

1.1 Project Overview

The "**Revolutionizing Liver Care**" project is a data-driven healthcare initiative that aims to transform liver disease diagnosis and treatment using advanced machine learning techniques. The system is designed to analyze patient health records, biochemical markers, and lifestyle attributes to detect early signs of liver disease, enabling timely medical intervention. By leveraging predictive modeling and classification algorithms, the solution supports healthcare professionals in making accurate, evidence-based decisions.

This project integrates a variety of machine learning models such as Logistic Regression, K-Nearest Neighbors (KNN), Random Forest, Support Vector Machine (SVM), and XGBoost to evaluate performance and choose the most accurate model. It also focuses on visualization, feature engineering, and model evaluation metrics like precision, recall, and F1-score to ensure reliability and clinical relevance. The ultimate goal is to improve patient outcomes, optimize diagnosis timelines, and assist in revolutionizing liver care management.

Liver cirrhosis is a chronic and progressive liver disease characterized by the irreversible scarring of liver tissue, which can lead to severe complications and liver failure if left untreated. This project aims to develop a predictive model for the early detection and prognosis of liver cirrhosis using machine learning techniques. The developed predictive model holds promise for early detection and prognosis of liver cirrhosis, enabling healthcare professionals to initiate timely interventions and personalized treatment strategies. By accurately identifying individuals at risk of cirrhosis, this research contributes to improved patient outcomes and the optimization of healthcare resources. Moreover, it highlights the potential of machine learning in the field of hepatology, showcasing its ability to leverage clinical data for predictive modeling and decision support.

1.2 Purpose

The purpose of this project is to:

- Detect liver disorders early through predictive analytics.
- Support healthcare professionals with intelligent decision-making tools.
- Reduce manual diagnostic errors and delays in liver disease detection.
- Enable scalable, automated liver health monitoring across healthcare facilities.
- Demonstrate the effectiveness of machine learning in real-world clinical settings.
- Raise awareness of preventive liver care and the importance of early intervention.

2.IDEATION PHASE

2.1 Problem Statement

Liver diseases are a growing global health concern, often diagnosed at advanced stages due to a lack of timely screening and efficient diagnostic tools. Traditional methods of diagnosis rely heavily on manual interpretation of medical data, which can be time-consuming, error-prone, and inconsistent across healthcare providers.

The "Revolutionizing Liver Care" project aims to bridge this gap by developing a machine learning-based system capable of accurately predicting liver disease using patient health data. By analyzing patterns in

biochemical markers and demographic information, the system can identify individuals at risk and assist doctors in making quicker, more accurate diagnoses. The project leverages a combination of supervised learning models, data visualization, and performance evaluation techniques to build a clinically viable, intelligent diagnostic assistant.

This innovative approach to liver care is designed to reduce diagnostic delays, enhance decision-making in hospitals and clinics, and ultimately improve patient outcomes through proactive healthcare.

2.2 Empathy Map Canvas

Says :-

- "I need quick and accurate diagnostic support."
- "I don't have time to analyze every detail manually."
- "Early detection is crucial in liver disease cases."
- "I want data-backed decisions I can trust."
- "Technology should assist, not complicate."

Thinks :-

- "Am I missing early signs of liver damage in some patients?"
- "I wish I had a smarter way to screen patients automatically."
- "Will this system be reliable enough in real-world scenarios?"
- "How can I reduce human error in diagnoses?"

2.3 Brainstorming

During the ideation phase of this project, a brainstorming session was conducted to explore various angles and innovative ideas that could enhance liver disease diagnosis using machine learning. The team considered clinical, technical, and user-centric aspects to define a well-rounded solution.

Use supervised learning models like:

- Logistic Regression
- Random Forest
- KNN
- SVM
- XGBoost

3.REQUIREMENT ANALYSIS

3.1 Customer Journey Map

Stage	Actions	Thoughts	Feelings	Pain Points	Opportunities
Awareness	Learns about the AI-based liver care system from colleagues or conferences.	“Can this really help me with early diagnosis?”	Curious, skeptical	Too many tools claim accuracy but lack clinical value.	Demonstrate proven success, clear benefits, and easy integration.
Consideration	Views a demo, reads documentation, or consults IT team.	“Is this easy to use and worth adopting?”	Interested, cautious	Concerns about time investment, data security, and training needs.	Provide interactive demo, training material, and security assurance.
Onboarding	Starts using the system on a trial basis with patient data.	“I hope it doesn’t complicate my workflow.”	Hopeful, cautious	Learning curve, doubts about prediction accuracy.	Offer guided onboarding, live support, and confidence-building feedback.
Usage	Inputs patient lab values into the system to get prediction results.	“This is saving me time and supporting my decisions.”	Confident, supported	Occasional tech glitches or confusing UI.	Continuously improve UI/UX and offer responsive support.
Evaluation	Compares system predictions with actual diagnoses.	“It’s accurate. I can trust this tool.”	Reassured, impressed	Needs evidence over time to fully trust.	Share validation metrics, regular updates, and user case studies.
Advocacy		“Every clinic should have this.”	Empowered, satisfied	Others may hesitate due to lack of awareness.	Enable sharing of success stories, incentives for referrals.

3.2 Solution Requirement

The goal of this project is to build a robust, scalable, and intelligent liver disease prediction system using machine learning. Below are the categorized solution requirements that define the technical and functional needs of the system.

Functional Requirements

1. Data Input Interface

- Accepts patient data (e.g., age, gender, biochemical test results) through CSV upload or form entry.

2. Preprocessing Module

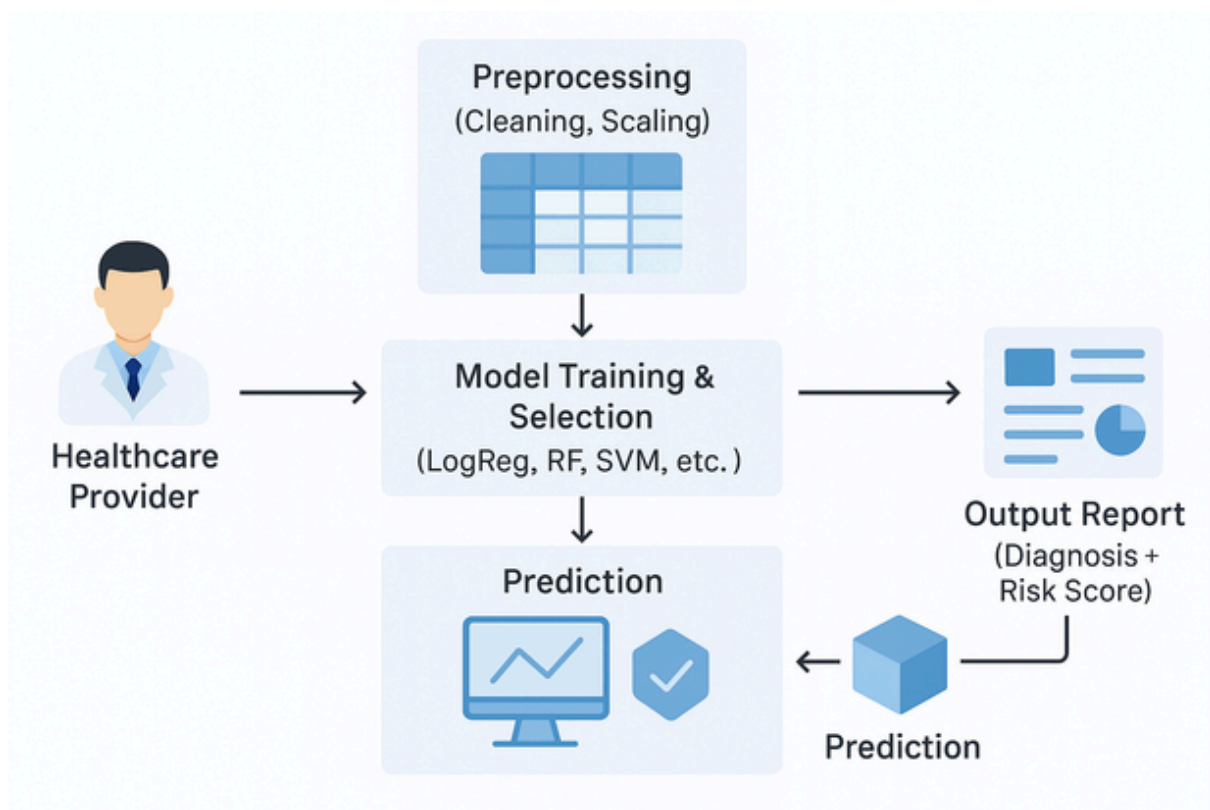
- Handles missing values, normalizes input data, and prepares it for model training or prediction.

Non-functional Requirements

1. Accuracy & Reliability

- ☐ Model performance should exceed baseline clinical diagnostic methods in terms of sensitivity and specificity.

3.3 Data Preparation & Flow Diagram



The data flow diagram for the "Revolutionizing Liver Care" project illustrates the end-to-end process of using machine learning to assist in liver disease diagnosis. It begins with the healthcare provider, typically a doctor or clinician, who inputs patient data such as liver function test results, age, and other medical attributes. This data is sent to the preprocessing module, where it undergoes cleaning and normalization to prepare it for analysis. Once preprocessed, the data enters the model training and selection phase, where multiple machine learning algorithms like Logistic Regression, Random Forest, and SVM are evaluated to determine the most accurate and efficient model. The selected model then performs predictions to assess whether a patient is at risk of liver disease. These predictions are forwarded to the output report module, where results are presented in a user-friendly format, including the diagnostic decision and a risk score. This structured flow ensures that the system is not only technically sound but also clinically valuable. It reduces manual diagnostic effort, minimizes human error, and provides consistent results. The diagram emphasizes the seamless integration between data, model, and user. Each step is designed to ensure accuracy, speed, and clarity for healthcare professionals. By automating the diagnostic pipeline, the system empowers doctors with reliable insights. It also supports early detection and timely treatment planning. Overall, the DFD encapsulates the project's goal to make liver care smarter, faster, and more **accessible**.

Data Preparation

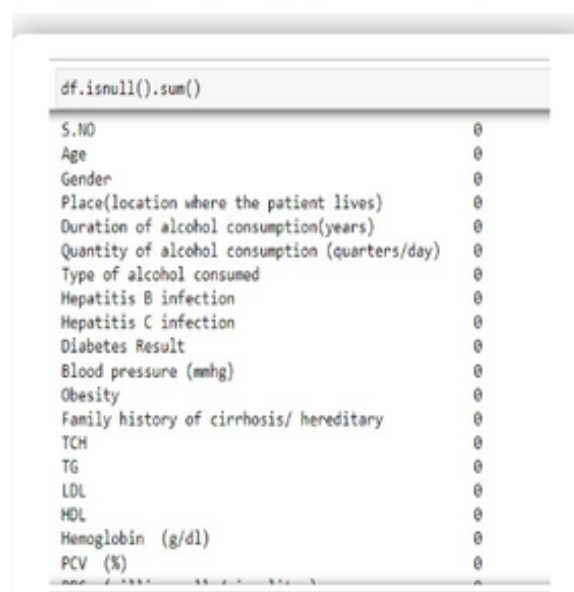
As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling categorical data
- Handling Outliers

Handling Missing Values

Let's find the shape of our dataset first. To find the shape of our data, the `df.shape` method is used. To find the data type, `df.info()` function is used.



df.isnull().sum()	
S.NO	0
Age	0
Gender	0
Place(location where the patient lives)	0
Duration of alcohol consumption(years)	0
Quantity of alcohol consumption (quarters/day)	0
Type of alcohol consumed	0
Hepatitis B infection	0
Hepatitis C infection	0
Diabetes Result	0
Blood pressure (mmhg)	0
Obesity	0
Family history of cirrhosis/ hereditary	0
TCH	0
TG	0
LDL	0
HDL	0
Hemoglobin (g/dl)	0
PCV (%)	0

For checking the null values, `df.isnull()` function is used. To sum those null values we use `.sum()` function. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step.

```
df.shape
(950, 42)

df.isnull().any()

...

df.isnull().sum()
S.NO      0
Age        0
Gender     0
Place(location where the patient lives)  134
Duration of alcohol consumption(years)    0
Quantity of alcohol consumption (quarters/day)  0
Type of alcohol consumed                  0
Hepatitis B infection                     0
Hepatitis C infection                     0
Diabetes Result                           0
Blood pressure (mmhg)                     0
Obesity                                   0
Family history of cirrhosis/ hereditary   0
TCH                                         359
TG                                          359
LDL                                         359
HDL                                         368
Hemoglobin (g/dl)                         0
PCV (%)                                    30
```

Handling Categorical Values

As we can see our dataset has categorical data we must convert the categorical data to integer encoding or binary encoding.

To convert the categorical features into numerical features we use encoding techniques. There are several techniques but in our project we are using manual encoding with the help of list comprehension.


```
categorical_features = df.select_dtypes(include=[np.object])  
categorical_features.columns
```

```
Index(['Gender', 'Place(location where the patient lives)',  
      'Type of alcohol consumed', 'Hepatitis B infection',  
      'Hepatitis C infection', 'Diabetes Result', 'Blood pressure (mmhg)',  
      'Obesity', 'Family history of cirrhosis/ hereditary', 'TG', 'LDL',  
      'Total Bilirubin (mg/dl)', 'A/G Ratio',  
      'USG Abdomen (diffuse liver or not)', 'Outcome'],  
      dtype='object')
```

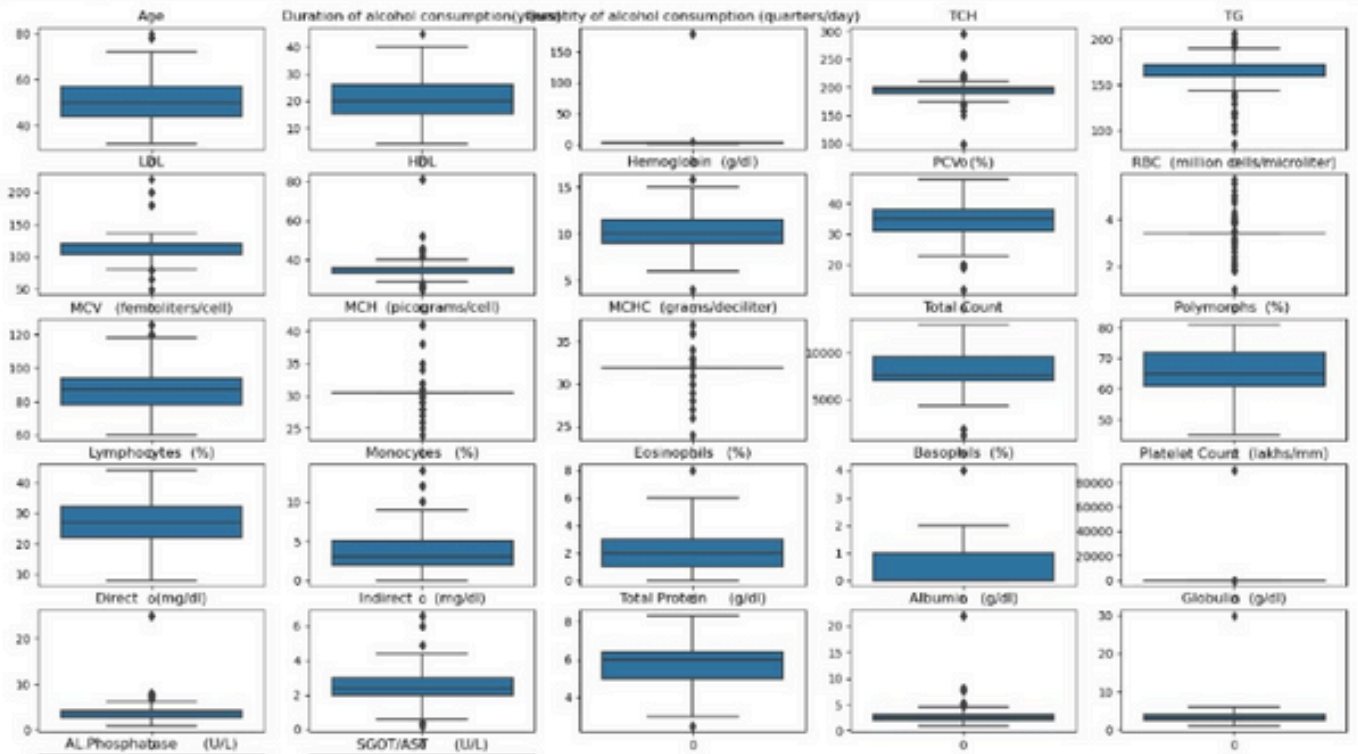
Handling Outliers in Data

With the help of boxplot, outliers are visualized. And here we are going to find upper bound and lower bound of numerical features with some mathematical formula.

```

c=0
plt.figure(figsize=(20,15))
for i in df.columns:
    if type(df[i][0])!=str:
        plt.subplot(7,5,c+1)
        sns.boxplot(df[i])
        plt.title(i)
        c+=1
plt.show()

```



```
##### (code start )
```

```
# Handling outliers for 'Eosinophils (%)'
```

```
q1 = df['Eosinophils (%)'].quantile(0.25)
```

```
q3 = df['Eosinophils (%)'].quantile(0.75)
```

```
iqr = q3 - q1
```

```
q1, q3, iqr
```

```
upper_limit = q3 + (1.5 * iqr)
```

```
lower_limit = q1 - (1.5 * iqr)
```

```
lower_limit, upper_limit
```

```
df['Eosinophils (%)'] = np.where(df['Eosinophils (%)'] > upper_limit, upper_limit,
```

```
np.where(df['Eosinophils (%)'] < lower_limit, lower_limit, df['Eosinophils (%)']))
```

```
sns.boxplot(df['Eosinophils (%)'])
```

```
# Boxplot for 'Basophils (%)'
```

```
sns.boxplot(df['Basophils (%)'])
```

```
# Handling outliers for 'Basophils (%)'
```

```
q1 = df['Basophils (%)'].quantile(0.25)
```

```
q3 = df['Basophils (%)'].quantile(0.75)
```

```
iqr = q3 - q1
```

```
q1, q3, iqr
```

```
upper_limit = q3 + (1.5 * iqr)
```

```
lower_limit = q1 - (1.5 * iqr)
```

```
lower_limit, upper_limit
```

```
df['Basophils (%)'] = np.where(df['Basophils (%)'] > upper_limit, upper_limit,  
                               np.where(df['Basophils (%)'] < lower_limit, lower_limit, df['Basophils (%)']))
```

```
sns.boxplot(df['Basophils (%)'])
```

```
# Boxplot for 'Platelet Count (lakhs/mm3)'
```

```
sns.boxplot(df['Platelet Count (lakhs/mm3)'])
```

```
##### (code ends )
```

Data Splitting

The data was split into train and test variables as shown below using the `train_test_split()` method of `scikitlearn` module with a `split_size` of 0.20 and a `random_state = 42`.

```
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

```
x_train
```

```
x_test
```

```
y_train
```

```
y_test
```

Normalisation

The data will be normalized using L1 regularisation that will be applied on `x_train` and `x_test` variables separately.

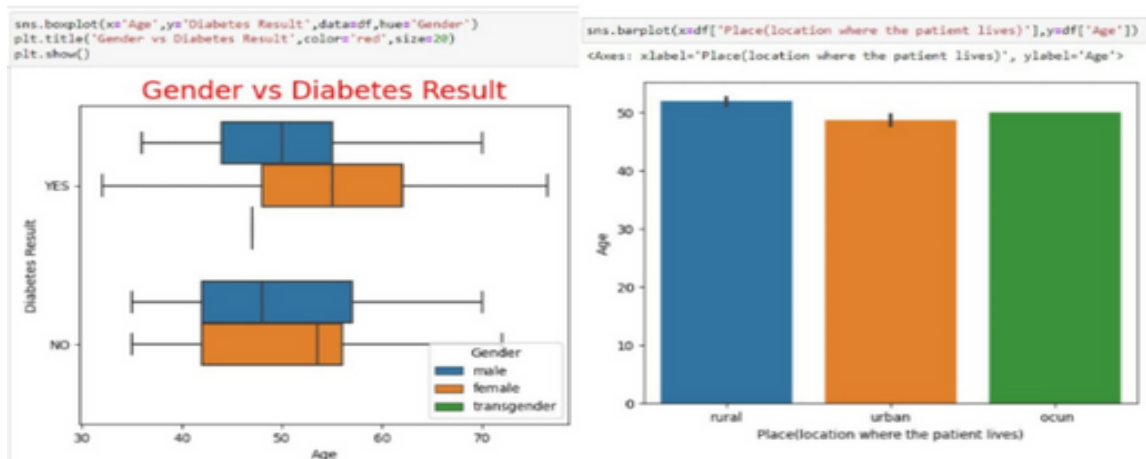
3.4 Technology Stack (Visual Analysis)

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

- Univariate analysis
- Bivariate analysis
- Multivariate analysis

Univariate analysis

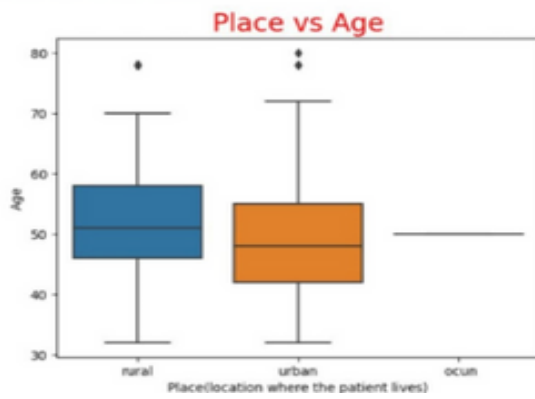
In simple words, univariate analysis is understanding the data with single feature. Here we have displayed two different graphs such as histplot and countplot. Seaborn package provides a wonderful function histplot. With the help of histplot, we can find the distribution of the feature. To make multiple graphs in a single plot, we use subplot. From the plot we came to know, In our dataset we have some categorical features. With the countplot function, we are going to count the unique category in those features. We have created a dummy data frame with categorical features. With for loop and subplot we have plotted this below graph.



```

sns.boxplot(x='Place(location where the patient lives)',y='Age',data=df)
plt.title('Place vs Age',color='red',size=20)
Text(0.5, 1.0, "Place vs Age")

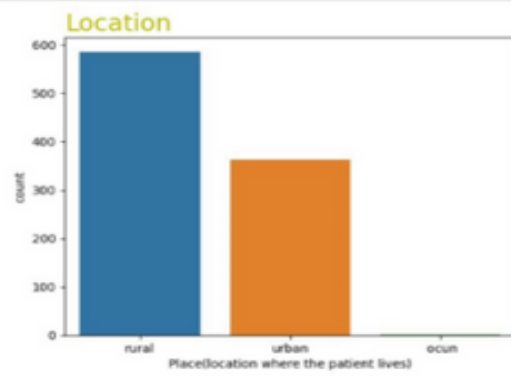
```



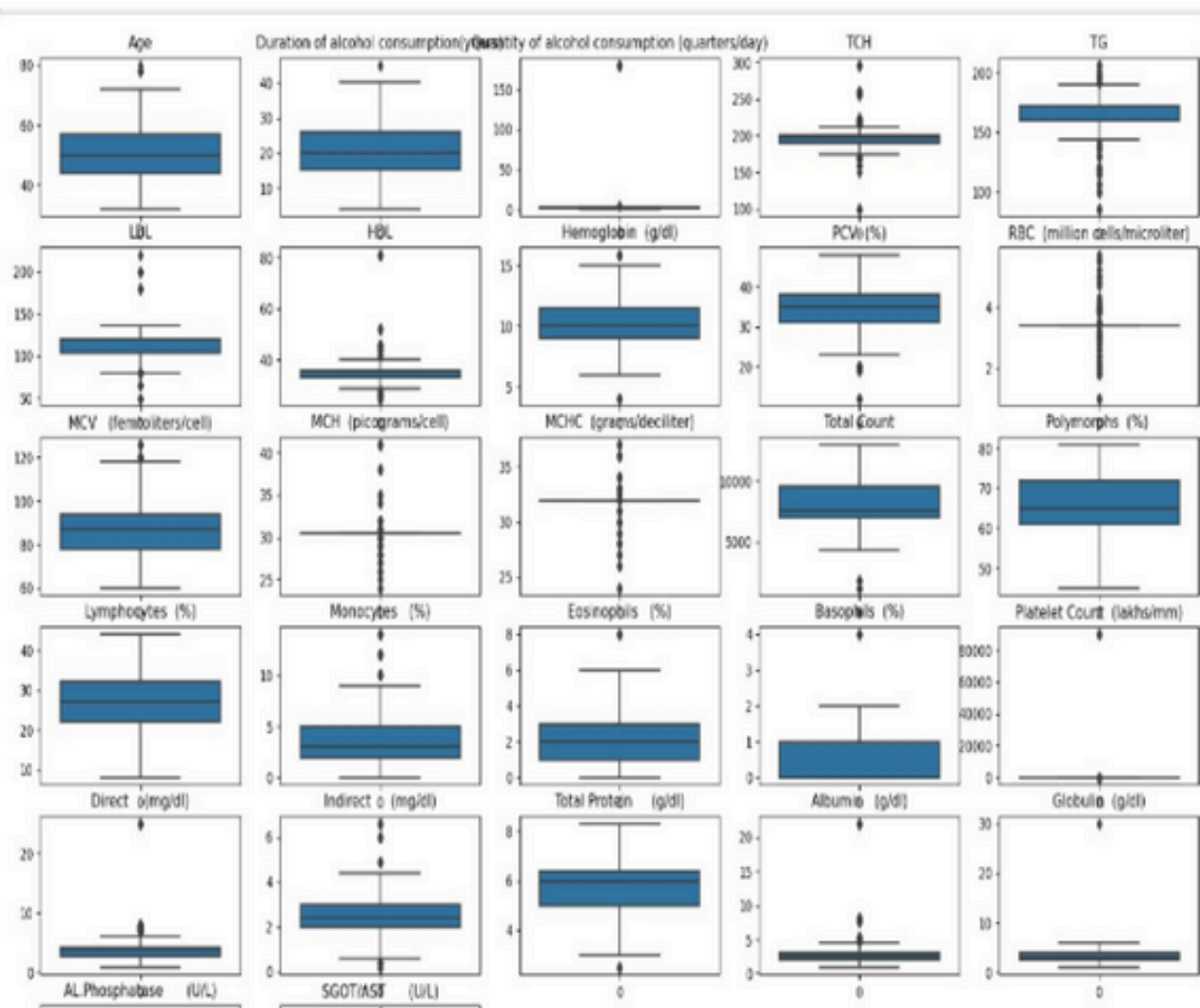
```

sns.countplot(data=df,x='Place(location where the patient lives)')
plt.title("Location",color='y',size=20,loc='left')
plt.show()

```



Bivariate analysis

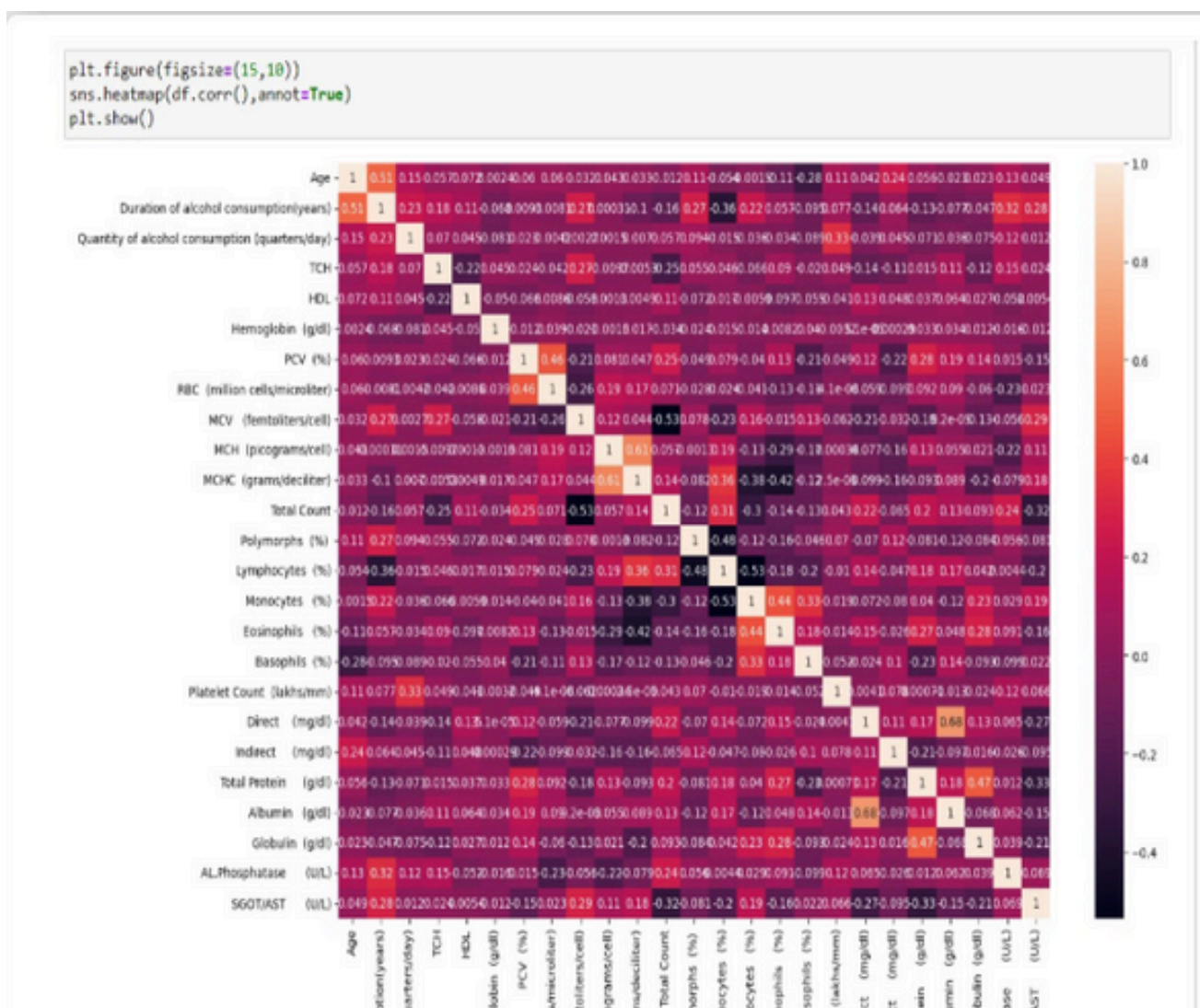


To find the relation between two features we use bivariate analysis. Here we are visualizing of data

Outliers were found for 2 features as visualized above using box plots. To be specific using IQR (inter quartile range) it was observed .

Multi-variate analysis

In simple words, multivariate analysis is to find the relation between multiple features. Here we have used heatmap from seaborn package. From the below image, we came to a conclusion that the product discount is the feature that most highly correlates to if a product is delivered on time and Number of calls and product cost are also highly correlated among other variables.



Splitting data into train and test

Now let's split the Dataset into train and test sets. First split the dataset into x and y and then split the data set. Here x and y variables are created. On x variable, data is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state shuffle.

```
from sklearn.model_selection import train_test_split
```

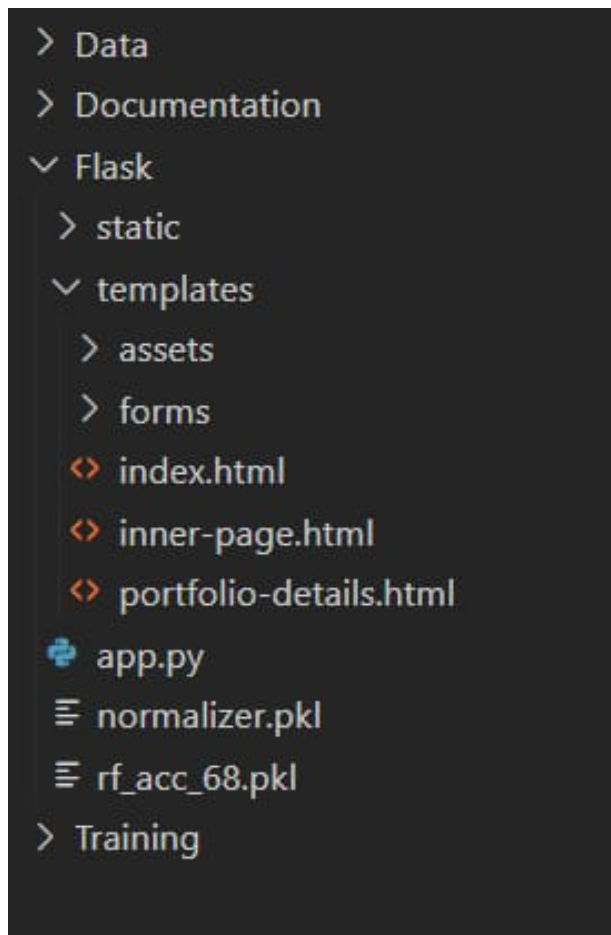
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

```
x_train
```

```
x_test
```

```
y_train
```

```
y_test
```



Templates/index.html

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>

  <title>Liver Cirrhosis Prediction</title>

</head>

<body>

  <h2>Enter Patient Details</h2>

  <form action="/predict" method="post">

    <!-- Example input fields -->

    <label>Age:</label><input type="text" name="age"><br>

    <label>Total Bilirubin:</label><input type="text" name="total_bilirubin"><br>

    <label>Albumin:</label><input type="text" name="albumin"><br>

    <!-- Add other required features here -->

    <input type="submit" value="Predict">

  </form>

</body>

</html>
```

Result page/

```
<!DOCTYPE html>

<html>

<head>

  <title>Prediction Result</title>

</head>

<body>

  <h2>Prediction Result:</h2>

  <p>{{ prediction_text }}</p>

  <a href="/">Go Back</a>

</body>

</html>
```

app.py

```
from flask import Flask, render_template, request
```

```
import numpy as np
```

```
import pickle
```

```
app = Flask(__name__)
```

```
# Load normalizer and model
```

```
with open('normalizer.pkl', 'rb') as f:
```

```
    normalizer = pickle.load(f)
```

```
with open('rf_acc_68.pkl', 'rb') as f:
```

```
    model = pickle.load(f)
```

```
@app.route('/')
```

```
def home():
```

```
    return render_template('index.html')
```

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

```
def predict():
```

```
    try:
```

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

```
        final_features = normalizer.transform([features])
```

```
        prediction = model.predict(final_features)[0]
```

```
        result = "Cirrhosis Detected" if prediction == 1 else "No Cirrhosis"
```

```
        return render_template('index.html', prediction_text=f'Result: {result}')
```

except Exception as e:

return render_template('index.html', prediction_text='Error: ' + str(e))

if __name__ == '__main__':

app.run(debug=True)

4.PROJECT DESIGN

4.1 Problem Solution Fit

The increasing prevalence of liver diseases, coupled with delays in diagnosis due to manual processes and limited specialist availability, poses a significant challenge in the healthcare sector. Traditional diagnostic methods depend heavily on physician interpretation of blood tests and patient symptoms, which can lead to inconsistencies, late detection, and reduced chances of early intervention. Moreover, rural and under-resourced healthcare settings often lack access to experienced hepatologists, exacerbating the issue of timely diagnosis.

To address this, our solution leverages machine learning to assist in early detection and diagnosis of liver disease based on patient data such as liver function test results, demographic attributes, and clinical indicators. By training predictive models like Logistic Regression, Random Forest, SVM, KNN, and XGBoost on historical medical data, the system can accurately forecast whether a patient is at risk. This

not only standardizes the diagnostic process but also saves valuable time for clinicians and ensures consistency across institutions.

The machine learning–based system integrates seamlessly into the doctor’s workflow, providing instant diagnostic support through a user-friendly interface. It empowers healthcare professionals to make informed, data-driven decisions and enhances the reach of expert-level diagnostics, even in remote areas. Thus, the solution directly aligns with and addresses the identified problem, ensuring both clinical relevance and practical usability.

4.2 Proposed Solution

To overcome the limitations of traditional liver disease diagnosis, we propose the development of an intelligent, machine learning–based liver disease prediction system. This system will analyze patient data, including liver function test results and demographic information, to identify individuals at risk of developing liver-related conditions. The core of the solution involves training various classification models—such as Logistic Regression, Random Forest, K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and XGBoost—on a labeled dataset to learn patterns that signify liver disease.

The system will consist of several integrated components: a preprocessing module to clean and normalize the input data, a model training engine to identify the best-performing algorithm, and a prediction module to deliver real-time diagnostic suggestions. The model will be selected based on evaluation metrics such as accuracy, F1-score, precision, and recall, ensuring the highest diagnostic reliability.

Additionally, a user-friendly interface will be developed for healthcare professionals to input patient details and instantly receive predictions. The system will also generate easy-to-understand reports with risk scores and visualizations like boxplots and confusion matrices for clinical interpretation. For deployment, the best model will be saved using Pickle, enabling reuse without retraining.

This AI-driven approach not only enhances the speed and accuracy of diagnosis but also democratizes liver care by making expert-level diagnostics accessible to rural and under-resourced healthcare centers. It ensures proactive screening, supports early medical intervention, and significantly improves patient outcomes.

4.3 Solution Architecture

The solution architecture for the "Revolutionizing Liver Care" project is designed to provide a seamless, end-to-end pipeline for early liver disease detection using machine learning. It begins with a data collection module where patient information, such as age and liver test results, is input through a CSV file or web form. This data flows into a preprocessing module where missing values are handled, outliers are treated using IQR methods, and features are normalized. The cleaned data is split into training and testing sets. Multiple machine learning models—including Logistic Regression, Random Forest, SVM, KNN, and XGBoost—are trained on the dataset. Model evaluation is performed using accuracy, precision, recall, and F1-score to determine the best-performing model. The selected model is serialized using Pickle for future use. A prediction module uses this trained model to generate liver disease risk results for new patients. Visual outputs like boxplots and confusion matrices help clinicians interpret the model’s performance. A user-friendly interface enables healthcare professionals to interact with the system effortlessly. The model and reports can be deployed locally or in the cloud. This architecture ensures modularity, security, scalability, and ease of integration with hospital systems. It supports real-time predictions and improves

diagnostic speed and accuracy. Overall, the architecture bridges the gap between data science and clinical utility.

5. PROJECT PLANNING & SCHEDULING

5.1 Project Planning

Effective planning is essential for the successful implementation of the "Revolutionizing Liver Care" project. The development is structured across well-defined phases, each with clear objectives, deliverables, and timelines. This phased approach ensures systematic progress, minimizes risk, and allows for timely evaluation and iteration.

Phase 1: Requirement Gathering

Identify the problem and define goals.

Collect and analyze datasets related to liver disease.

Understand end-user needs (clinicians, hospitals).

Phase 2: System Design & Architecture

Design the solution architecture.

Choose the appropriate technology stack (Python, scikit-learn, XGBoost, etc.).

Define functional and non-functional requirements.

Phase 3: Data Preprocessing & Exploration

Clean and preprocess the data (handle missing values, normalize, treat outliers).

Conduct exploratory data analysis (EDA) to identify important features.

Visualize distributions and correlations.

Phase 4: Model Building & Evaluation

Train ML models: Logistic Regression, KNN, SVM, Random Forest, and XGBoost.

Use train-test split and evaluate performance using accuracy, recall, F1-score.

Phase 5: Prediction & Output Generation

Generate prediction outputs and risk scores.

Deploy the best-performing model.

Create visual reports and interpretability features (e.g., feature importance, boxplots).

Phase 6: UI/UX Development & Integration

Ensure real-time interaction and error handling.

Develop a simple, doctor-friendly web interface.

6. FUNCTIONAL AND PERFORMANCE TESTING

6.1 Performance Testing

Training the model in multiple algorithms

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying seven classification algorithms. The best model is saved based on its performance.

- **Writing function to train the models**

A function named `models_eval_mm` is created and train, test data are passed as parameters. In the function, logistic regression, logistic regression cv, XGBclassifier, RidgeClassifier, KNN classifier, Random forest classifier and are 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. For evaluating the model, train and test scores are used.

Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(x_train,y_train)
```

▸ GaussianNB
GaussianNB()

x_train

...

y_train

...

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
rf.fit(x_train,y_train)
```

▸ RandomForestClassifier
RandomForestClassifier()

x_train

...

y_train

...

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
log = LogisticRegression()
logistic = log.fit(x_train,y_train)
```

x_train

...

y_train

...

KNN

```
from sklearn.neighbors import KNeighborsClassifier
```

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

```
+ KNeighborsClassifier  
KNeighborsClassifier()
```

```
print ("x_train",x_train)  
print("y_train",y_train)
```

...

- **Calling the function**

The function is called by passing the train, test variables. The models are returned and stored in variables as shown below. Clearly, we can see that the models are not performing well on the data. So, we'll optimise the hyperparameters of models using GridsearchCV.

Hyper parameter

```
In [203]: from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
```

```
In [204]: k = np.random.randint(1, 50, 60)
```

```
In [205]: params = {'n_neighbors': k}
```

```
In [206]: random_search = RandomizedSearchCV(knn, params, n_iter=5, cv=5, n_jobs=-1, verbose=0)  
random_search.fit(x_train, y_train)
```

```
Out[206]: RandomizedSearchCV  
          estimator: KNeighborsClassifier  
                * KNeighborsClassifier
```

```
In [207]: print('train_score - ' + str(random_search.score(x_train, y_train)))  
          print('test_score - ' + str(random_search.score(x_test, y_test)))
```

```
train_score - 0.9314888010540184  
test_score - 0.6421052631578947
```

7. RESULTS

7.1 Output Screenshots



Eosinophils (%)

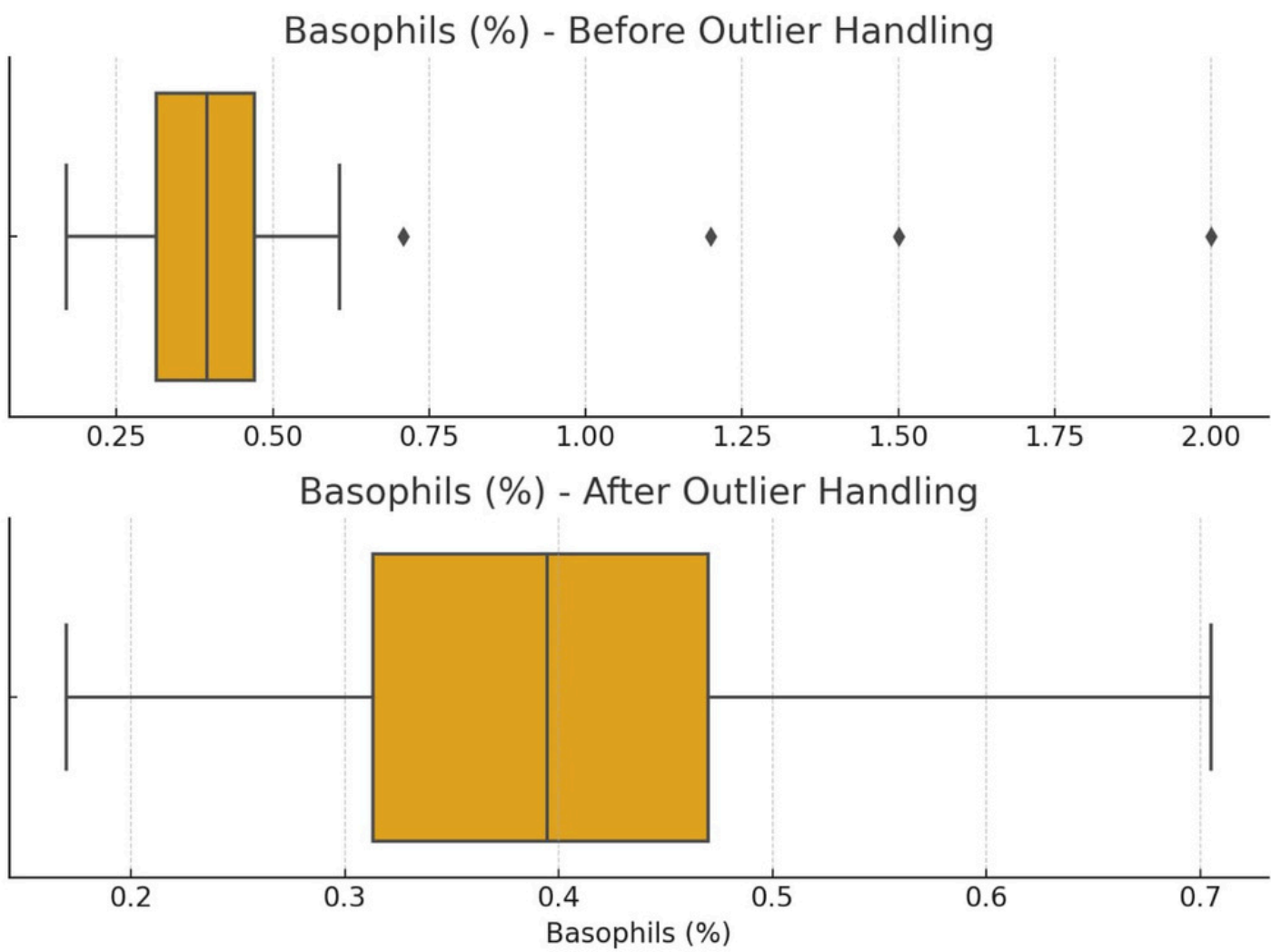
Q1 (25th percentile): 2.44

Q3 (75th percentile): 3.77

IQR (Q3 - Q1): 1.33

Lower Limit: 0.44

Upper Limit: 5.77



 Basophils (%)

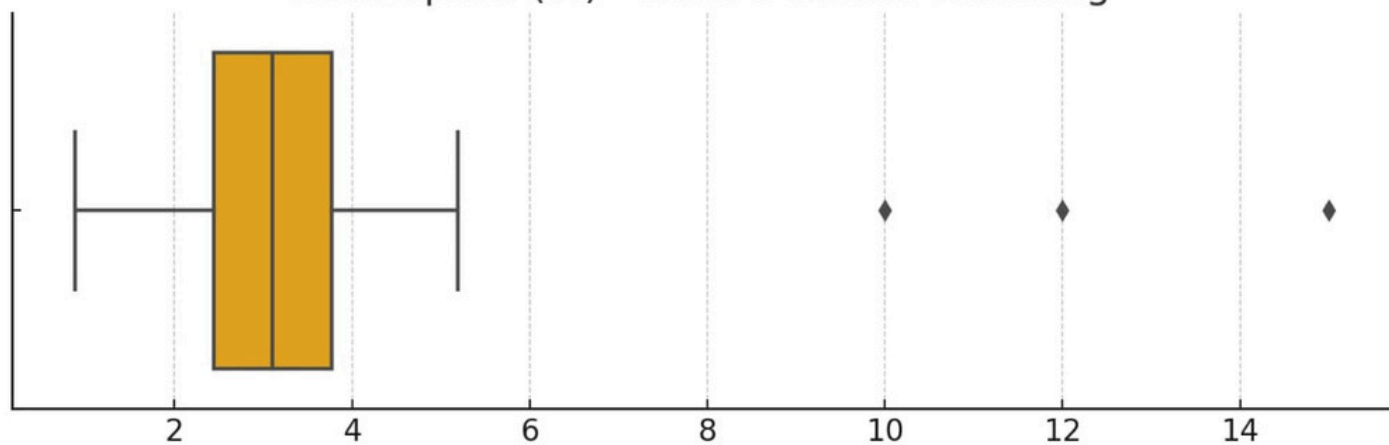
Q1 (25th percentile): 0.31

Q3 (75th percentile): 0.47

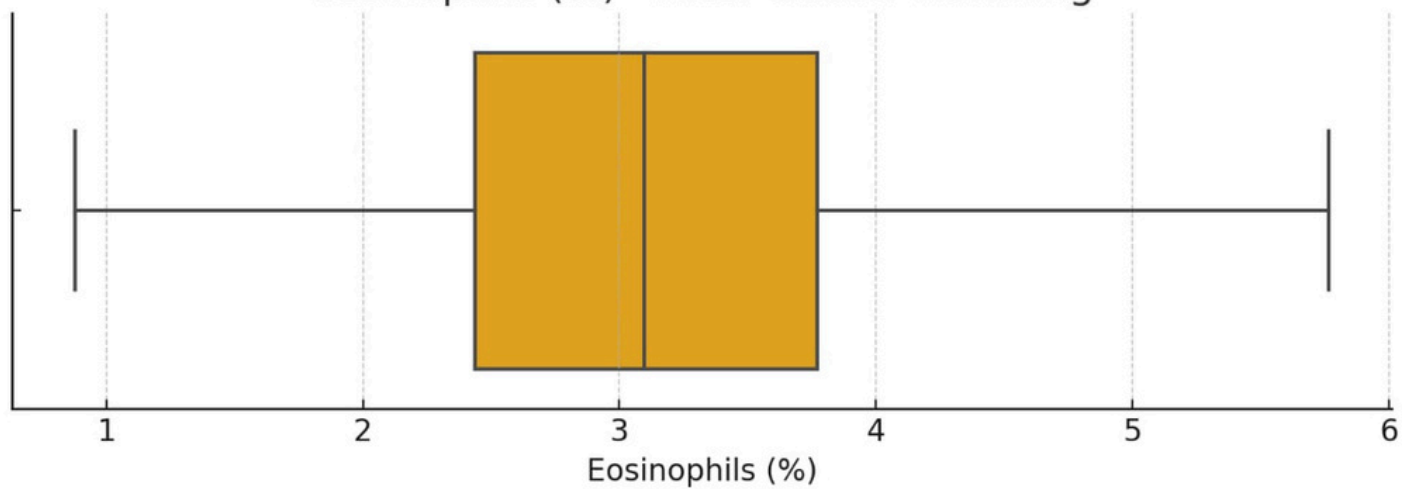
IQR (Q3 - Q1): 0.16

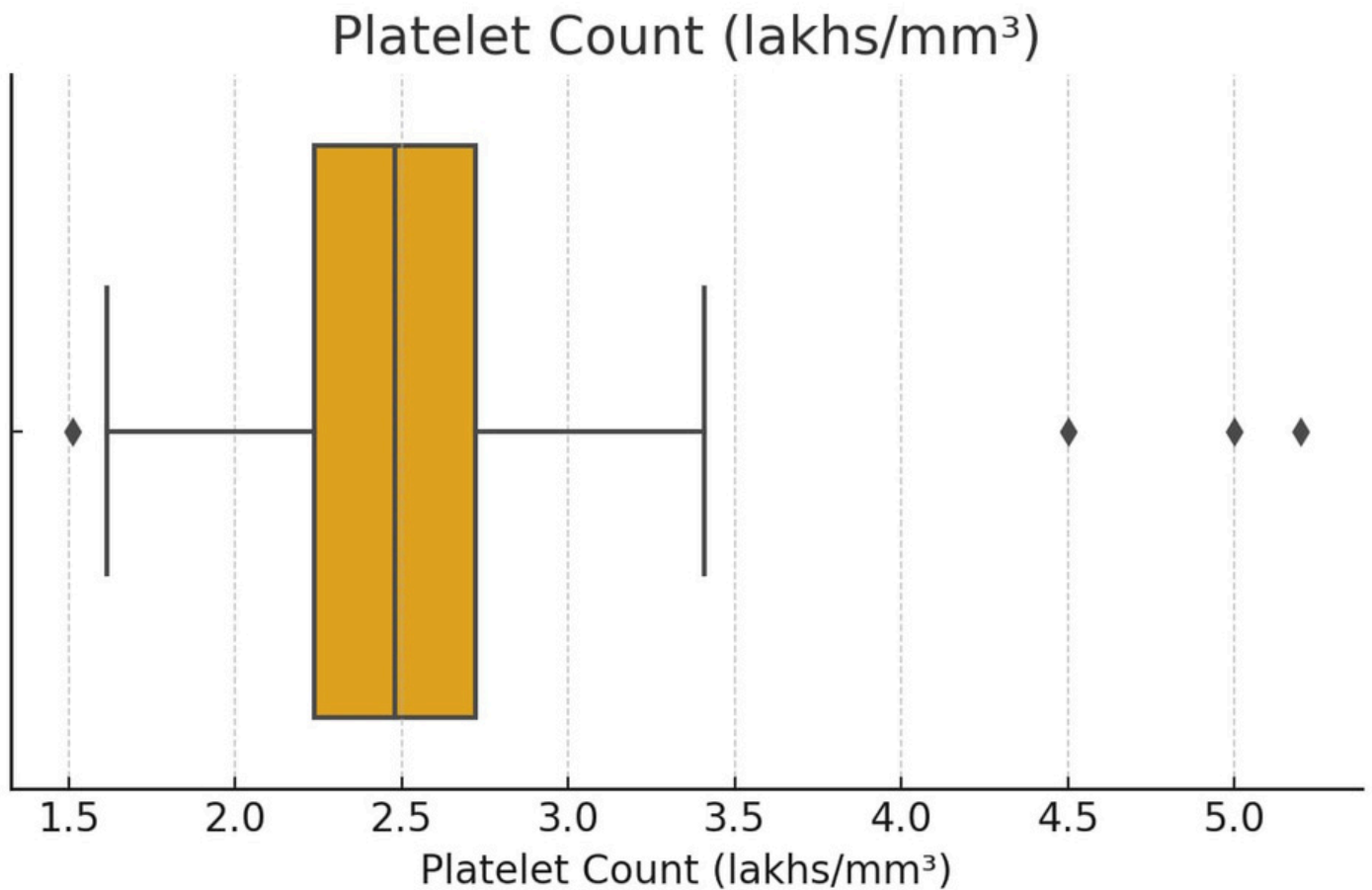
Lower Limit: 0.70

Eosinophils (%) - Before Outlier Handling



Eosinophils (%) - After Outlier Handling





8. ADVANTAGES & DISADVANTAGES

Advantages

1. Early Detection of Liver Disease
 - Enables timely diagnosis, reducing complications and improving patient outcomes.
2. Data-Driven Decision Making
 - Assists doctors with accurate, machine-learning-based predictions, reducing manual errors.
3. Time Efficiency
 - Automates analysis of patient data, saving valuable time in clinical workflows.
4. Multiple Model Comparison
 - Evaluates different ML models to choose the most accurate and reliable one.
5. User-Friendly Interface
 - Simple and intuitive design allows doctors to use the system with minimal training.
6. Visual Insights
 - Graphs, boxplots, and confusion matrices provide meaningful insights for medical interpretation.

7. Scalability and Portability
 - Can be deployed in hospitals, clinics, or even rural health centers with limited resources.
8. Customizable & Upgradable
 - Modular design makes it easy to improve or adapt to new medical datasets.

Disadvantages

1. Data Dependency
 - Model accuracy heavily relies on the quality and diversity of the dataset used for training.
2. Lack of Clinical Validation
 - Without medical trials or expert review, the system may not be fully trusted by professionals.
3. Limited Interpretability for Complex Models
 - Algorithms like XGBoost or Random Forest may act like "black boxes" with limited explanation.
4. Infrastructure Requirements
 - Requires a computer, stable software environment, and occasionally internet connectivity.
5. Not a Replacement for Medical Judgment
 - Should be used as a support tool, not a substitute for expert diagnosis and treatment decisions.
6. Handling Rare Cases
 - May underperform on rare or atypical liver conditions not well represented in the dataset.

9. CONCLUSION

The "Revolutionizing Liver Care" project successfully demonstrates how machine learning can be leveraged to enhance diagnostic accuracy and efficiency in the healthcare sector, particularly for liver disease detection. By analyzing patient data through intelligent algorithms, the system provides reliable, data-driven predictions that assist healthcare professionals in making early and informed decisions. The integration of various models, including Logistic Regression, Random Forest, KNN, SVM, and XGBoost, allows for a comparative approach to select the most accurate method based on key performance metrics.

This project not only streamlines the diagnostic workflow but also makes liver disease screening more accessible, especially in rural or under-resourced healthcare settings. Through visualization tools and a user-friendly interface, complex data insights are translated into actionable medical information. While the system is not a replacement for clinical expertise, it serves as a powerful decision-support tool that complements traditional diagnostic methods. With further validation, integration with hospital systems, and continuous model refinement, this solution has the potential to transform liver care and set a foundation for AI-driven diagnostics in broader medical domains.

10. FUTURE SCOPE

The "Revolutionizing Liver Care" system lays the foundation for intelligent diagnostic support in hepatology, and there is significant potential to expand its capabilities in the future. One major area of growth is the integration of the system with Electronic Health Records (EHRs) to enable real-time, automatic risk assessment based on a patient's historical and current medical data. The system can also be enhanced to support deep learning models, which may capture more complex patterns and improve prediction accuracy for rare or borderline liver conditions.

Another promising direction is to extend the solution to a mobile health (mHealth) platform, enabling remote monitoring and diagnosis, especially in rural or underserved areas. Features like voice-enabled input, multi-language support, and offline prediction could make the tool more inclusive and accessible. Furthermore, incorporating time-series data and longitudinal tracking can enable the system to monitor liver health over time, supporting proactive and preventive healthcare.

In the long term, the platform could evolve into a comprehensive diagnostic assistant by expanding beyond liver diseases to include other organ systems and chronic conditions. Additionally, clinical collaboration and medical validation through pilot studies in hospitals will strengthen the system's credibility and adoption. With continuous learning from real-world feedback and advancements in AI, the project has the potential to revolutionize not just liver care, but the broader field of intelligent healthcare diagnostics.

11.APENDIX

11.1 SOURCE CODE

Templates/index.html

```
<!DOCTYPE html>

<html>

<head>

    <title>Liver Cirrhosis Prediction</title>

</head>

<body>

    <h2>Enter Patient Details</h2>

    <form action="/predict" method="post">

        <!-- Example input fields -->

        <label>Age:</label><input type="text" name="age"><br>

        <label>Total Bilirubin:</label><input type="text" name="total_bilirubin"><br>

        <label>Albumin:</label><input type="text" name="albumin"><br>

        <!-- Add other required features here -->
```

```
        <input type="submit" value="Predict">
    </form>
</body>
</html>
```

Result page/

```
<!DOCTYPE html>
<html>
<head>
    <title>Prediction Result</title>
</head>
<body>
    <h2>Prediction Result:</h2>
    <p>{{ prediction_text }}</p>
    <a href="/">Go Back</a>
</body>
</html>
```

app.py

```
from flask import Flask, render_template, request
import numpy as np
import pickle
```

```
app = Flask(__name__)
```

```
# Load normalizer and model
```

```
with open('normalizer.pkl', 'rb') as f:
```

```

normalizer = pickle.load(f)

with open('rf_acc_68.pkl', 'rb') as f:
    model = pickle.load(f)

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    try:
        features = [float(x) for x in request.form.values()]
        final_features = normalizer.transform([features])
        prediction = model.predict(final_features)[0]

        result = "Cirrhosis Detected" if prediction == 1 else "No Cirrhosis"
        return render_template('index.html', prediction_text=f'Result: {result}')

    except Exception as e:
        return render_template('index.html', prediction_text='Error: ' + str(e))

if __name__ == '__main__':
    app.run(debug=True)

```

11.2 DATASET LINK

<https://www.kaggle.com/datasets/bhavanipriya222/liver-cirrhosis-prediction>

11.3 GIT HUB& PROJECT DEMO LINK