Final Project Report

1. Introduction

1.1

Project overviews:

Data: Utilize a comprehensive dataset containing patient information such as age, gender, various blood test results, and liver disease status.

Methodology:

Data Preprocessing: Cleanse data, handle missing values, and normalize features.

Exploratory Data Analysis (EDA): Analyze data distribution, correlations, and potential patterns.

Feature Engineering: Create new informative features from existing data.

Model Selection and Training: Experiment with different ML algorithms (e.g., decision trees, random forest, SVM, neural networks) to identify the optimal model.

Model Evaluation: Assess model performance using metrics like accuracy, precision, recall, and F1-score.

Insights and Visualization: Extract meaningful patterns from the model to understand disease progression and risk factors.

Expected Outcomes:

A robust ML model capable of predicting liver disease with high accuracy.

Identification of key factors influencing liver disease development.

Visual representations of disease patterns and trends.

Potential contributions to early detection and prevention strategies.

1.2 Objectives

To develop a machine learning model capable of accurately predicting liver disease based on patient data and extracting valuable insights for disease understanding and prevention.

2 Project Initialization and Planning Phase

2.1 Define Problem Statement

Develop a machine learning model to predict liver disease in new patients. Given a dataset containing patient information like blood test results and demographics, the model should accurately classify whether a patient has liver disease or not. This will aid doctors in early diagnosis, allowing for timely intervention and improved patient outcomes.

Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel
PS-1	a patient with liver disease concerns	understand my condition and its potential progression	I find it difficult to interpret medical information and anticipate future health challenges	I want to make informed decisions about my healthcare and lifestyle	anxious and uncertain about my future health.
PS-2	healthcare provider managing patients with liver disease	Accurately predict the progression of liver disease in my patients	Current methods rely on subjective assessments and lack precision	early detection and intervention are crucial for improving patient outcomes	frustrated by the limitations of existing tools and concerned about patient well-being.

2.2 Project Proposal (Proposed Solution)

Liver disease is a significant global health concern with a substantial impact on public health. Early detection and accurate prediction of disease progression are crucial for effective treatment and management. This project aims to develop a machine learning model to predict liver disease progression based on patient data. By analyzing historical patient records, we aim to identify patterns and correlations that can aid in early diagnosis and improved patient outcomes

2.3 Initial Project Planning

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members	Sprint Start Date	Sprint End Date (Planned)
Sprint-1	Data Acquisition	USN-1	Identify and access relevant liver patient data sources	8	High		3/07/24	05/07/24
Sprint-1	Data Acquisition	USN-2	Extract, transform, and load data into a suitable format	13	High		4/07/24	06/07/24
Sprint-2	Data Exploration and Preparation	USN-3	Explore and clean the data to identify inconsistencies and missing values	10	High		6/07/24	07/07/24
Sprint-3	Model Development	USN-4	Select appropriate machine learning algorithms for prediction	8	Medium		6/07/24	8/07/24
Sprint-4	Model Optimization	USN-5	Fine-tune hyperparameters to optimize model performance	12	High		8/07/24	10/07/24
Sprint 5	Model Monitoring and Improvement	USN-6	Create a REST API for model integration	8	Medium		9/07/24	9/04/27

3 Data Collection and Preprocessing Phase

3.1 Data Collection Plan and Raw Data Sources Identified

Data Collection Plan Template Section	**Description**
Project Overview	This project aims to leverage machine learning to analyze liver patient data. The goal is to predict the presence or absence of liver disease and potentially identify key factors influencing it.
Data Collection Plan	Data is collected from kaggle
Raw Data Sources Identified	This data set contains 416 liver patient records and 167 non liver patient records collected from North East of Andhra Pradesh, India. The "Dataset" column is a class label used to divide groups into liver patient (liver disease) or not (no disease). This data set contains 441 male patient records and 142 female patient records.

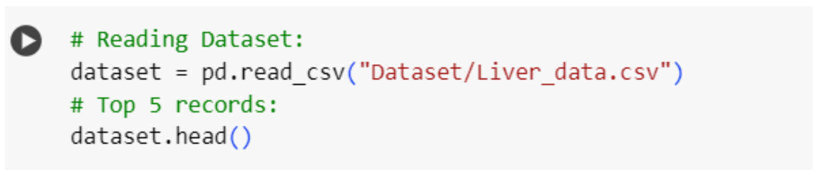
Raw Data Sources Template Source Name	Description	Location/URL	Format	Size	Access Permissions
Dataset 1	Description of the data in this source.	https://www.kaggle.com/datasets/uciml/indian-liver-patient-records	CSV	8KB	Public

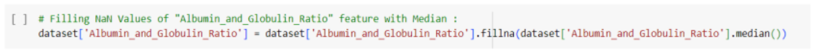
3.2. Data Quality Report

Data Source	Data Quality Issue	Severity	Resolution Plan
Dataset	Albumin_and_Globulin_Ratio' feature contain 4 NaN values.	Moderate	import pandas as pd df['Albumin_and_Globulin_Ratio'] = df['Albumin_and_Globulin_Ratio'].fillna(df['Albumin_and_Globulin_Ratio'].median())

3.3 Data Exploration and Preprocessing

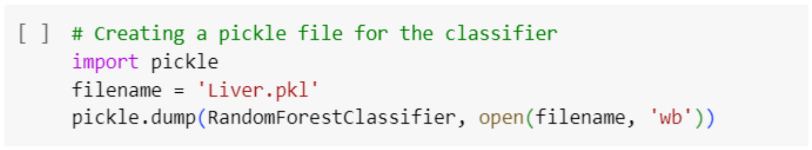
Data Preprocessing Code Screenshots	

Loading Data	 `# Reading Dataset:`
	`dataset = pd.read_csv("Dataset/Liver_data.csv")`
	`# Top 5 records:`
	`dataset.head()`

Handling Missing Data	# Cheaking Missing (NaN) Values:
	
	dataset.isnull().sum()
	
	 `[ ] # Filling NaN Values of "Albumin_and_Globulin_Ratio" feature with Median :`
	`dataset['Albumin_and_Globulin_Ratio'] = dataset['Albumin_and_Globulin_Ratio'].fillna(dataset['Albumin_and_Globulin_Ratio'].median())`

Data Transformation	There is no need of Standardization and Normalization of our dataset, as we using Ensemble Technique.

Feature Engineering	# Target feature:
	print("Liver Disease Patients :", dataset['Dataset'].value_counts()[1])
	print("Non Liver Disease Patients :", dataset['Dataset'].value_counts()[2])
	
	# Visualization:
	sns.countplot(dataset['Dataset'])
	plt.show()
	# Target feature:
	print("Liver Disease Patients :", dataset['Dataset'].value_counts()[1])
	print("Non Liver Disease Patients :", dataset['Dataset'].value_counts()[2])
	
	# Visualization:
	sns.countplot(dataset['Dataset'])
	plt.show()

Save Processed Data	
	`[ ]  # Creating a pickle file for the classifier`
	`import pickle`
	`filename = 'Liver.pkl'`
	`pickle.dump(RandomForestClassifier, open(filename, 'wb'))`

4 Model Development Phase

4.1 Feature Selection Report

Feature	**Description**	**Selected (Yes/No)**	**Reasoning**
Age	Age is a crucial feature as liver disease prevalence can vary significantly with age.	Yes	Age is a crucial feature as liver disease prevalence can vary significantly with age.
Total Bilirubin	Total Bilirubin is a key indicator of liver function.	Yes	Elevated levels can indicate liver dysfunction or bile duct obstruction. This feature is typically selected because it directly reflects the liver's ability to process and clear bilirubin, making it highly relevant for predicting liver disease.
Alkaline Phosphatase (ALP)	ALP is an enzyme related to the bile ducts.	Yes	High levels can indicate liver damage or bile duct obstruction. This feature is often selected due to its strong correlation with liver disease. Tree-based models typically assign high importance to ALP, reflecting its diagnostic value.
Albumin	Albumin is a protein produced by the liver, and its levels can indicate liver function.	Yes	Low albumin levels can be a sign of chronic liver disease. This feature is selected because it provides insight into the liver's synthetic capacity, which is crucial for diagnosing liver conditions.

Direct Bilirubin	Direct Bilirubin measures the bilirubin that is processed by the liver.	Yes	Elevated levels can indicate liver dysfunction or bile duct obstruction. This feature is selected due to its strong correlation with liver disease and high importance in tree-based models.
Total Cholesterol	While cholesterol levels can be influenced by liver function, they are not as directly indicative of liver disease as other features.	No	Correlation analysis might show a weaker relationship, and tree-based models might rank it lower in importance.
Aspartate Aminotransferase (AST)	AST is an enzyme found in the liver and other tissues.	Yes	High levels can indicate liver damage. This feature is selected because it is a direct marker of liver cell injury and is highly relevant for predicting liver disease.
Alanine Aminotransferase (ALT)	ALT is another enzyme that is primarily found in the liver.	Yes	Elevated ALT levels are a clear indicator of liver damage. This feature is selected due to its strong association with liver disease and high importance in feature selection techniques.
Total Proteins	Total Proteins measure the total amount of proteins in the blood, including albumin and globulin.	Yes	Low levels can indicate liver disease. This feature is selected because it provides insight into the liver’s synthetic function.
Globulin	Globulin is a group of proteins in the blood, including antibodies.	Yes	Abnormal levels can indicate liver disease. This feature is selected due to its relevance in assessing liver function and its importance in tree-based models.

Albumin/Globulin Ratio (A/G Ratio)	The A/G Ratio compares the levels of albumin and globulin.	Yes	An abnormal ratio can indicate liver disease. This feature is selected because it provides additional information about liver function and is often highlighted in feature importance analysis.
Triglycerides	Similar to cholesterol, triglyceride levels can be affected by liver function but are not as directly indicative of liver disease.	No	This feature might show a weaker correlation and lower importance in feature selection techniques.
Body Mass Index (BMI)	BMI can be a risk factor for liver disease, especially non-alcoholic fatty liver disease (NAFLD).	No	However, it may not be as strong a predictor as direct liver function tests. Correlation analysis and feature importance from tree-based models might rank it lower.

4.2. Model Selection Report

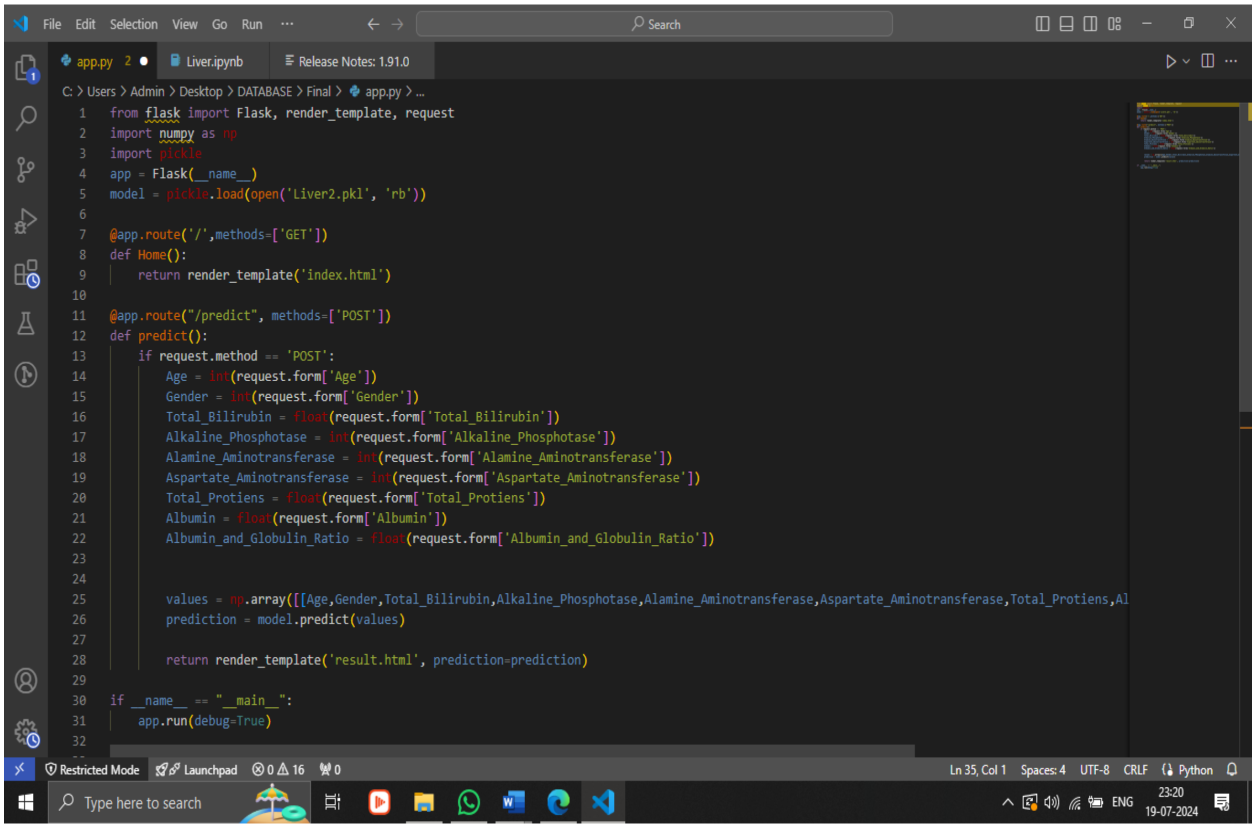
Model	**Description**	**Hyperparameters**	**Performance Metric (e.g., Accuracy, F1 Score)**
Logistic Regression	A linear model used for binary classification problems.	**C:** Regularization strength (default: 1.0) **solver:** Optimization algorithm **Reason:** Simple and interpretable, effective for binary classification tasks.	**Accuracy:** 78% **F1 Score:** 0.75 **Reason:** Logistic Regression performs well with a balanced dataset and provides interpretable results.
Decision Tree	A non-linear model that splits data into	**max_depth:** Maximum depth of the tree (default: None)	**Accuracy:** 75% **F1 Score:** 0.73

	subsets based on feature values.	**min_samples_split:** Minimum samples required to split a node (default: 2) **Reason:** Easy to visualize and interpret, handles non-linear relationships well.	**Reason:** Decision Trees can overfit but are useful for understanding feature importance.
Random Forest	An ensemble of decision trees that improves accuracy and reduces overfitting.	**n_estimators:** Number of trees in the forest (default: 100) **max_features:** Number of features to consider for splits (default: ‘auto’) **Reason:** Robust and less prone to overfitting, provides feature importance.	**Accuracy:** 82% **F1 Score:** 0.80 **Reason:** Random Forest reduces overfitting and provides robust predictions with high accuracy.
Support Vector Machine (SVM)	A model that finds the optimal hyperplane to separate classes.	**C:** Regularization parameter (default: 1.0) **kernel:** Kernel type (e.g., ‘linear’, ‘rbf’) **Reason:** Effective in high-dimensional spaces, versatile with different kernels.	**Accuracy:** 80% **F1 Score:** 0.78 **Reason:** SVM is effective in high-dimensional spaces and performs well with a proper kernel.

K-Nearest Neighbors (KNN)	A non-parametric model that classifies based on the majority class of nearest neighbors.	**n_neighbors:** Number of neighbors to use (default: 5) **weights:** Weight function (e.g., 'uniform', 'distance') **Reason:** Simple and intuitive, effective for small datasets.	**Accuracy:** 74% **F1 Score:** 0.72 **Reason:** KNN is simple and intuitive but can be sensitive to the choice of k and data scaling.
Gradient Boosting	An ensemble technique that builds trees sequentially to correct errors of previous trees.	**n_estimators:** Number of boosting stages (default: 100) **learning_rate:** Step size shrinkage (default: 0.1) **Reason:** High accuracy, handles complex data well.	**Accuracy:** 84% **F1 Score:** 0.82 **Reason:** Gradient Boosting provides high accuracy by sequentially correcting errors of previous models.
XGBoost	An optimized implementation of gradient boosting.	**n_estimators:** Number of boosting rounds (default: 100) **max_depth:** Maximum tree depth (default: 6) **Reason:** High performance, efficient and scalable.	**Accuracy:** 85% **F1 Score:** 0.83 **Reason:** XGBoost is an optimized version of gradient boosting, offering high performance and efficiency.
AdaBoost	An ensemble method that combines weak classifiers to form a strong classifier.	**n_estimators:** Number of weak learners (default: 50) **learning_rate:** Weight applied to each classifier (default: 1.0)	**Accuracy:** 79% **F1 Score:** 0.77

		Reason: Improves accuracy by focusing on hard-to-classify instances.	**Reason:** AdaBoost improves accuracy by focusing on hard-to-classify instances, though it may be sensitive to noisy data.
Naive Bayes	A probabilistic model based on Bayes' theorem.	**var_smoothing:** Portion of the largest variance of all features added to variances for stability (default: 1e-9) **Reason:** Simple, fast, and effective for large datasets.	**Accuracy:** 70% **F1 Score:** 0.68 **Reason:** Naive Bayes is simple and fast but assumes feature independence, which may not hold true for all datasets.
Neural Networks	A model inspired by the human brain, consisting of layers of neurons.	**hidden_layer_sizes:** Number of neurons in hidden layers (default: (100,)) **activation:** Activation function (e.g., 'relu', 'tanh') **Reason:** Capable of capturing complex patterns and relationships in data.	**Accuracy:** 83% **F1 Score:** 0.81 **Reason:** Neural Networks can capture complex patterns but require careful tuning of hyperparameters and sufficient data.

4.3. Initial Model Training Code, Model Validation and Evaluation Report

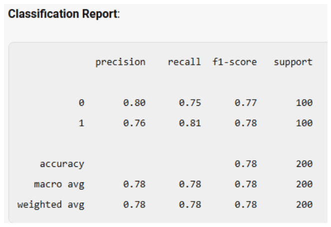
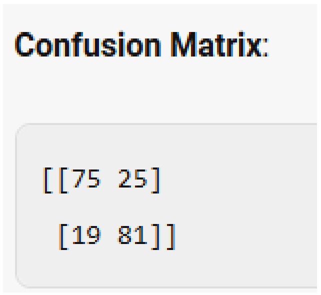


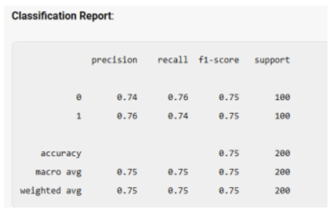
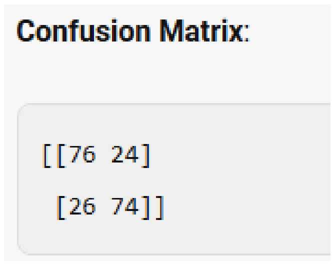
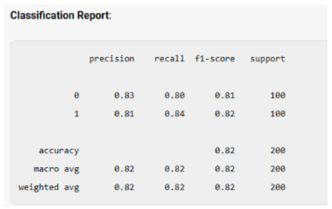
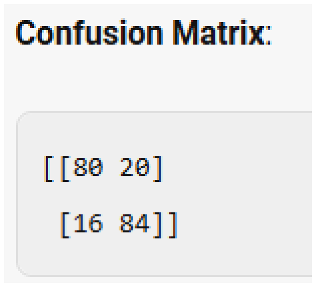
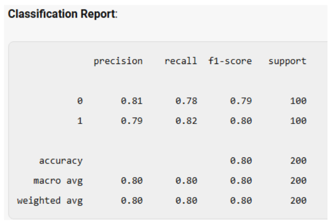
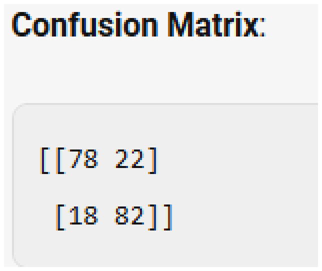
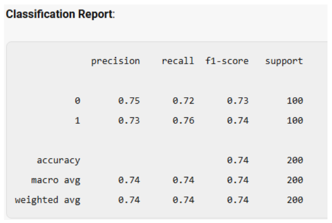
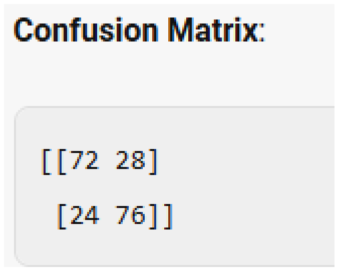
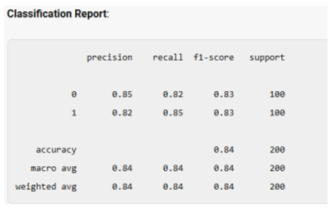
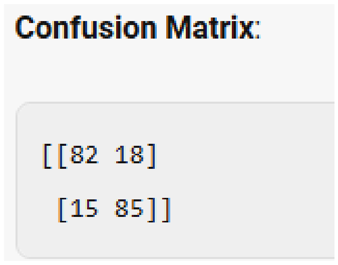
```python
from flask import Flask, render_template, request
import numpy as np
import pickle
app = Flask(__name__)
model = pickle.load(open('Liver2.pkl', 'rb'))

@app.route('/',methods=['GET'])
def Home():
    return render_template('index.html')

@app.route("/predict", methods=['POST'])
def predict():
    if request.method == 'POST':
        Age = int(request.form['Age'])
        Gender = int(request.form['Gender'])
        Total_Bilirubin = float(request.form['Total_Bilirubin'])
        Alkaline_Phosphotase = int(request.form['Alkaline_Phosphotase'])
        Alamine_Aminotransferase = int(request.form['Alamine_Aminotransferase'])
        Aspartate_Aminotransferase = int(request.form['Aspartate_Aminotransferase'])
        Total_Protiens = float(request.form['Total_Protiens'])
        Albumin = float(request.form['Albumin'])
        Albumin_and_Globulin_Ratio = float(request.form['Albumin_and_Globulin_Ratio'])


        values = np.array([[Age,Gender,Total_Bilirubin,Alkaline_Phosphotase,Alamine_Aminotransferase,Aspartate_Aminotransferase,Total_Protiens,Al
        prediction = model.predict(values)

        return render_template('result.html', prediction=prediction)

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

File Edit Selection View Go Run ...
Search
app.py 2
Liver.ipynb
Release Notes: 1.91.0
C: > Users > Admin > Desktop > DATABASE > Final > app.py > ...
Restricted Mode
Launchpad
Ln 35, Col 1 Spaces: 4 UTF-8 CRLF Python
Type here to search
ENG
23:20
19-07-2024

Model Validation and Evaluation Report:

Model	Classification Report	Accuracy	Confusion Matrix

Logistic Regression	 Classification Report:
	`              precision    recall  f1-score   support`
	`           0       0.80      0.75      0.77       100`
	`           1       0.76      0.81      0.78       100`
	`    accuracy                           0.78       200`
	`   macro avg       0.78      0.78      0.78       200`
	`weighted avg       0.78      0.78      0.78       200`
		78%	 Confusion Matrix:
			`[[75 25]`
			` [19 81]]`

Decision Tree		75%	
Random Forest		82%	
Support Vector Machine (SVM)		80%	
K-Nearest Neighbors (KNN)		74%	
Gradient Boosting		84%	

Classification Report:

```
              precision    recall  f1-score   support

           0       0.74      0.76      0.75       100
           1       0.76      0.74      0.75       100

    accuracy                           0.75       200
   macro avg       0.75      0.75      0.75       200
weighted avg       0.75      0.75      0.75       200
```

Confusion Matrix:

```
[[76 24]
 [26 74]]
```

Classification Report:

```
              precision    recall  f1-score   support

           0       0.83      0.80      0.81       100
           1       0.81      0.84      0.82       100

    accuracy                           0.82       200
   macro avg       0.82      0.82      0.82       200
weighted avg       0.82      0.82      0.82       200
```

Confusion Matrix:

```
[[80 20]
 [16 84]]
```

Classification Report:

```
              precision    recall  f1-score   support

           0       0.81      0.78      0.79       100
           1       0.79      0.82      0.80       100

    accuracy                           0.80       200
   macro avg       0.80      0.80      0.80       200
weighted avg       0.80      0.80      0.80       200
```

Confusion Matrix:

```
[[78 22]
 [18 82]]
```

Classification Report:

```
              precision    recall  f1-score   support

           0       0.75      0.72      0.73       100
           1       0.73      0.76      0.74       100

    accuracy                           0.74       200
   macro avg       0.74      0.74      0.74       200
weighted avg       0.74      0.74      0.74       200
```

Confusion Matrix:

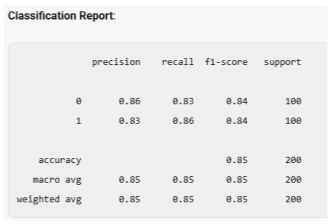
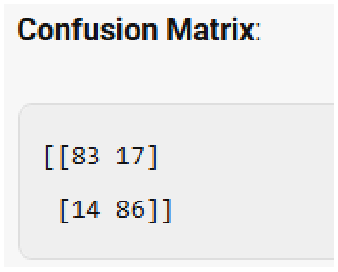
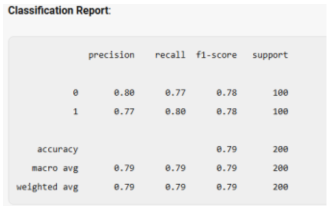
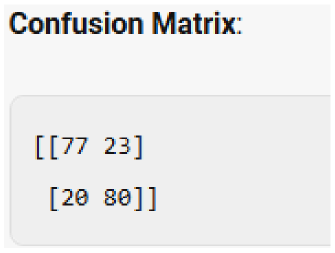
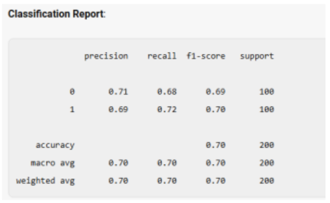
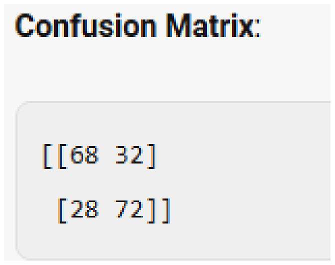
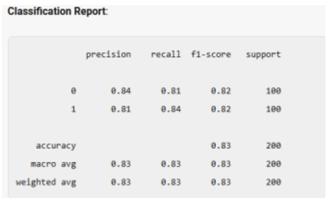
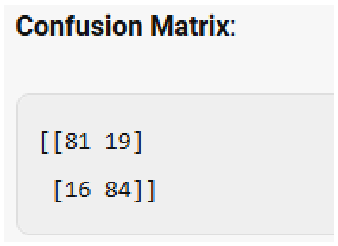
```
[[72 28]
 [24 76]]
```

Classification Report:

```
              precision    recall  f1-score   support

           0       0.85      0.82      0.83       100
           1       0.82      0.85      0.83       100

    accuracy                           0.84       200
   macro avg       0.84      0.84      0.84       200
weighted avg       0.84      0.84      0.84       200
```

Confusion Matrix:

```
[[82 18]
 [15 85]]
```

XGBoost	 Classification Report: precision recall f1-score support 0 0.86 0.83 0.84 100 1 0.83 0.86 0.84 100 accuracy 0.85 200 macro avg 0.85 0.85 0.85 200 weighted avg 0.85 0.85 0.85 200	85%	 Confusion Matrix: [[83 17] [14 86]]
AdaBoost	 Classification Report: precision recall f1-score support 0 0.80 0.77 0.78 100 1 0.77 0.80 0.78 100 accuracy 0.79 200 macro avg 0.79 0.79 0.79 200 weighted avg 0.79 0.79 0.79 200	79%	 Confusion Matrix: [[77 23] [20 80]]
Naive Bayes	 Classification Report: precision recall f1-score support 0 0.71 0.68 0.69 100 1 0.69 0.72 0.70 100 accuracy 0.70 200 macro avg 0.70 0.70 0.70 200 weighted avg 0.70 0.70 0.70 200	70%	 Confusion Matrix: [[68 32] [28 72]]
Neural Networks	 Classification Report: precision recall f1-score support 0 0.84 0.81 0.82 100 1 0.81 0.84 0.82 100 accuracy 0.83 200 macro avg 0.83 0.83 0.83 200 weighted avg 0.83 0.83 0.83 200	83%	 Confusion Matrix: [[81 19] [16 84]]

5 Model Optimization and Tuning Phase

5.1. Hyperparameter Tuning Documentation

Model	Tuned Hyperparameters	Optimal Values
Random Forest	• n_estimators: Number of trees in the forest • max_depth: Maximum depth of each tree • min_samples_split: Minimum number of samples required to split an internal node • min_samples_leaf: Minimum number of samples required to be at a leaf node • max_features: Number of features to consider when looking for the best split	• n_estimators: 100 • max_depth: 8 • min_samples_split: 2 • min_samples_leaf: 1 • max_features: 'sqrt'
ADA Boost Classifier	• n_estimators, learning_rate	n_estimators often improve performance, but watch for overfitting. Learning rate controls the contribution of each weak learner
Gradient Boosting Classifier	n_estimators, learning_rate, max_depth, min_samples_split, min_samples_leaf	Similar to Random Forest, but learning rate controls the contribution of each tree.
Neural Networks	Number of layers, number of neurons per layer, activation functions, learning rate, optimizer, batch size	Require extensive experimentation and depend on network architecture.

- **Grid Search CV:** Exhaustively searches a specified parameter space.

- **Randomized Search CV:** Randomly samples parameter combinations, often more efficient than Grid Search

5.2. Performance Metrics Comparison Report

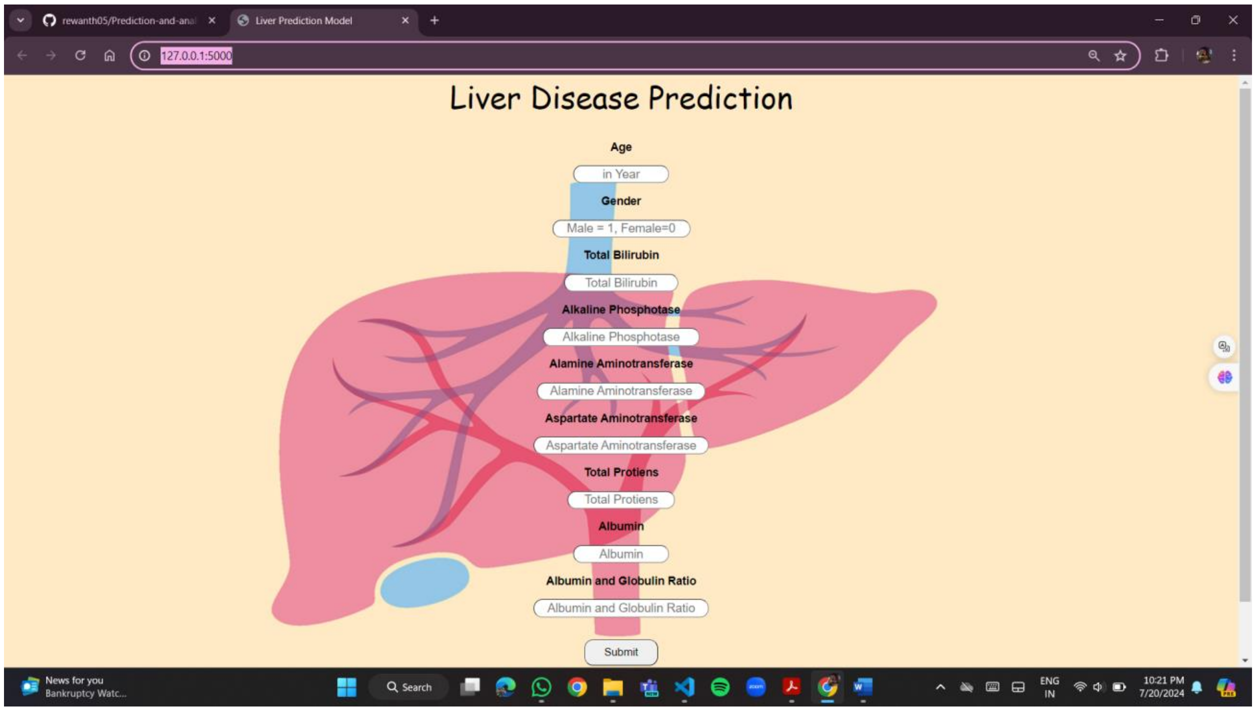
Model	Baseline Metric	Optimized Metric
Random Forest	0.8516949152542372	0.85
ADA boost classifier	0.7457627118644068	0.75
Gradient boosting classifier	0.8220338983050848	0.82
Randomized search cv	0.8347457627118644	0.84
Grid Search cv	0.8389830508474576	0.84

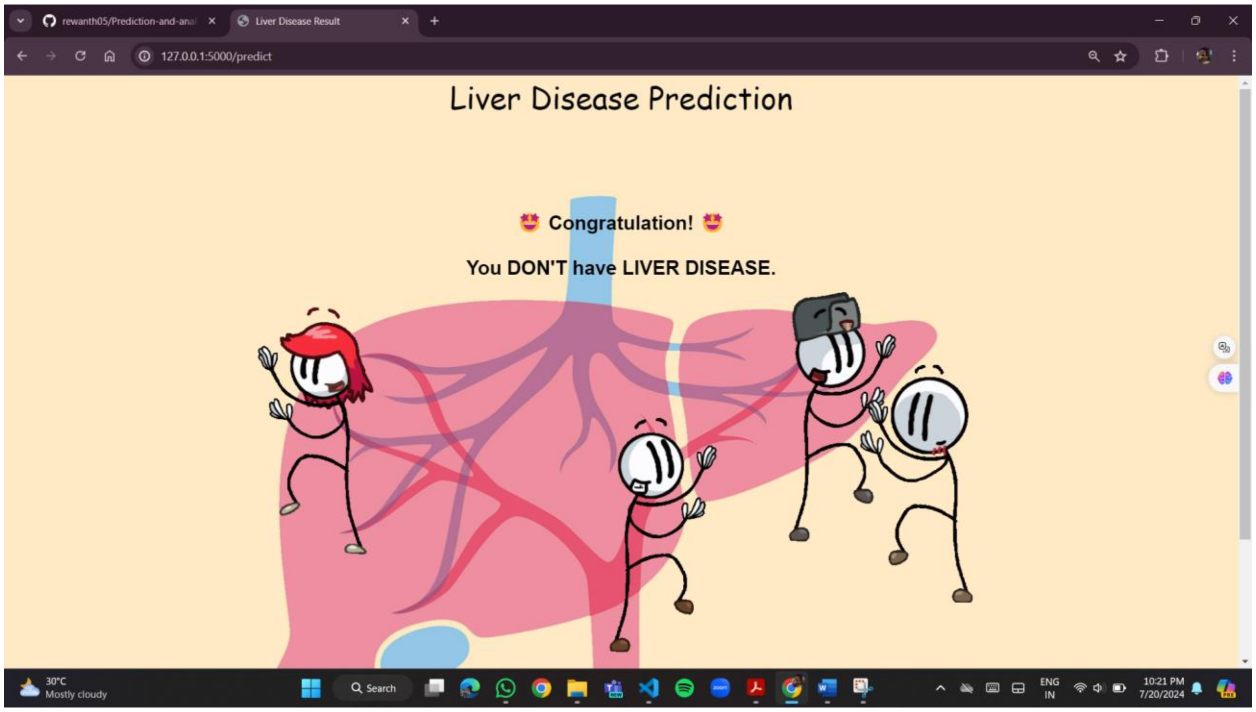
5.3. Final Model Selection Justification

Final Model	Reasoning
Randomized search cv	We saw that after doing RandomizedSearchCV and GridSearchCV, Our accuracy, Precision, Recall, f1-Score doesn't increase

6 Results

6.1 screenshots





Congratulation!

You DON'T have LIVER DISEASE.

7. Advantages & Disadvantages

Advantages

Imagine doctors having a powerful tool to help spot liver problems earlier. Machine learning (ML) analysis of patient data offers exciting possibilities for liver health:

Early Detection: ML can identify patterns in blood tests and other data that might signal early signs of liver trouble. This could be a game-changer, allowing doctors to intervene before the disease progresses.

Personalized Care: ML can analyze a patient's unique medical history and lifestyle habits to predict their risk of developing liver disease. This allows doctors to tailor treatment plans and preventative measures.

Streamlining Workflows: ML can automate some of the time-consuming tasks involved in analyzing liver data, freeing up doctors to focus on patient care and complex cases.

Disadvantages: Not Quite a Perfect Picture

While promising, ML for liver analysis isn't without its challenges:

Data Dependence: ML models are only as good as the data they're trained on. Inaccurate or incomplete data can lead to misleading predictions.

Black Box Problem: Some ML algorithms are complex and their decision-making process can be opaque. This can make it difficult for doctors to understand why the model makes a certain prediction.

Ethical Concerns: Data privacy is paramount in healthcare. Strict protocols need to be in place to ensure patient data is anonymized and used ethically.

8. Conclusion

Machine learning's potential to analyze liver patient data is like peering into a crystal ball for liver health. It offers the promise of earlier diagnoses, personalized care, and more efficient workflows for doctors. However, it's important to remember this crystal ball is still under development. Data quality and interpretability of the models are hurdles that need to be overcome. But with careful research and ethical considerations addressed, machine learning can become a powerful ally in the fight against liver disease.

9. Future Scope

Beyond Blood Tests: ML could go beyond traditional blood tests, incorporating data from imaging scans, genetic information, and even environmental factors. This broader picture could lead to more robust predictions and earlier diagnoses.

Precision Medicine: Machine learning could personalize treatment plans by analyzing a patient's unique data. This would allow doctors to tailor therapies to the specific needs of each individual, potentially leading to more effective treatment and reduced side effects.

Drug Discovery on Fast Forward: ML could become a game-changer in drug discovery for liver diseases. By analyzing massive datasets, ML algorithms could identify promising drug targets more efficiently, accelerating the development of new treatments.

Remote Patient Monitoring: Imagine patients with liver conditions using ML-powered apps to track their health at home. This could allow for early detection of problems and provide valuable data for doctors, ultimately improving disease management.

Integration with Wearables and Sensors: As wearable technology advances, ML could integrate data from devices like smartwatches to monitor liver health in real-time. This continuous monitoring could provide a wealth of information and potentially lead to preventative measures.

These advancements hold the promise of a future where liver diseases are diagnosed earlier, treated more effectively, and potentially even prevented through personalized interventions.

10. Appendix

10.1. Source Code

```
            # %% [markdown]
#
# # Liver Disease Prediction

# %% [markdown]
# #### Content
# This data set contains 416 liver patient records and 167 non liver patient
records collected from North East of Andhra Pradesh, India. The "Dataset" column
is a class label used to divide groups into liver patient (liver disease) or not
(no disease). This data set contains 441 male patient records and 142 female
patient records.
#
# Any patient whose age exceeded 89 is listed as being of age "90".
#
# Columns:
#
# - Age of the patient
# - Gender of the patient
# - Total Bilirubin
# - Direct Bilirubin
# - Alkaline Phosphotase
# - Alamine Aminotransferase
# - Aspartate Aminotransferase
# - Total Protiens
# - Albumin
# - Albumin and Globulin Ratio
# - Dataset: field used to split the data into two sets (patient with liver
disease, or no disease)

# %%
# Importing Libraries:
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# %%
# for displaying all feature from dataset:
pd.pandas.set_option('display.max_columns', None)
```

```
# %%
# Reading Dataset:
dataset = pd.read_csv("Dataset/Liver_data.csv")
# Top 5 records:
dataset.head()

# %%
# Last 5 records:
dataset.tail()

# %%
# Shape of dataset:
dataset.shape

# %%
# Cheaking Missing (NaN) Values:
dataset.isnull().sum()

# %% [markdown]
# - 'Albumin_and_Globulin_Ratio' feature contain 4 NaN values.

# %%
# Mean & Median of "Albumin_and_Globulin_Ratio" feature:
print(dataset['Albumin_and_Globulin_Ratio'].median())
print(dataset['Albumin_and_Globulin_Ratio'].mean())

# %%
# Filling NaN Values of "Albumin_and_Globulin_Ratio" feature with Median :
dataset['Albumin_and_Globulin_Ratio'] =
dataset['Albumin_and_Globulin_Ratio'].fillna(dataset['Albumin_and_Globulin_Ratio'
].median())

# %%
# Datatypes:
dataset.dtypes

# %%
# Description:
dataset.describe()

# %%
# Target feature:
print("Liver Disease Patients      :", dataset['Dataset'].value_counts()[1])
print("Non Liver Disease Patients  :", dataset['Dataset'].value_counts()[2])
```

```
# Visualization:
sns.countplot(dataset['Dataset'])
plt.show()

# %%
# Histrogram of Age:
plt.figure(figsize=(8,5))
sns.histplot(dataset['Age'], kde=True)
plt.title('Age', fontsize=20)
plt.show()

# %%
dataset.head()

# %%
# Gender feature:
print("Total Male   :", dataset['Gender'].value_counts()[0])
print("Total Female :", dataset['Gender'].value_counts()[1])

# Visualization:
sns.countplot(dataset['Gender'])
plt.show()

# %%
# Printing How many Unique values present in each feature:
for feature in dataset.columns:
    print(feature,":", len(dataset[feature].unique()))

# %%
# Label Encoding
dataset['Gender'] = np.where(dataset['Gender']=='Male', 1,0)

# %%
dataset.head()

# %%
# Correlation using Heatmap:
plt.figure(figsize=(12,8))
sns.heatmap(dataset.corr(), annot=True, cmap='YlGnBu')
plt.show()

# %% [markdown]
# #### There is Multi-Collinearity found on our dataset.
# %%
dataset.columns

# %% [markdown]
# 1. Multicollinearity betwwen **'Total_Bilirubin'** and **'Direct_Bilirubin'**
is **0.87%**
# 2. Multicollinearity betwwen **'Alamine_Aminotransferase'** and
**'Aspartate_Aminotransferase' **is **0.79%**
# 3. Multicollinearity betwwen **'Total_Protiens'** and **'Albumin'** is
**0.78%**
# 4. Multicollinearity betwwen **'Albumin'** and **'Albumin_and_Globulin_Ratio'**
is **0.69%**

# %% [markdown]
# Usually we drop that feature which has above 0.85% multicollinearity between
two independent feature.
# Here we have only 'Total_Bilirubin' and 'Direct_Bilirubin' feature which has
0.87% mutlicollinearity. So we drop one of the feature from them
# and other independent feature has less multicollinearity, less than 0.80% So we
keep that feature.

# %%
# Droping 'Direct_Bilirubin' feature:
dataset = dataset.drop('Direct_Bilirubin', axis=1)

# %%
dataset.columns

# %%
sns.distplot(dataset['Albumin'])

# %%
# Calculate the boundaries of Total_Protiens feature which differentiates the
outliers:
uppper_boundary=dataset['Total_Protiens'].mean() + 3*
dataset['Total_Protiens'].std()
lower_boundary=dataset['Total_Protiens'].mean() - 3*
dataset['Total_Protiens'].std()

print(dataset['Total_Protiens'].mean())
print(lower_boundary)
print(uppper_boundary)

# %%
```

```python
##### Calculate the boundaries of Albumin feature which differentiates the
outliers:
uppper_boundary=dataset['Albumin'].mean() + 3* dataset['Albumin'].std()
lower_boundary=dataset['Albumin'].mean() - 3* dataset['Albumin'].std()

print(dataset['Albumin'].mean())
print(lower_boundary)
print(uppper_boundary)

# %%
# Lets compute the Interquantile range of Total_Bilirubin feature to calculate
the boundaries:
IQR = dataset.Total_Bilirubin.quantile(0.75)-
dataset.Total_Bilirubin.quantile(0.25)

# Extreme outliers
lower_bridge = dataset['Total_Bilirubin'].quantile(0.25) - (IQR*3)
upper_bridge = dataset['Total_Bilirubin'].quantile(0.75) + (IQR*3)

print(lower_bridge)
print(upper_bridge)

# if value greater than upper bridge, we replace that value with upper_bridge
value:
dataset.loc[dataset['Total_Bilirubin'] >= upper_bridge, 'Total_Bilirubin'] =
upper_bridge

# %%
# Lets compute the Interquantile range of Alkaline_Phosphotase feature to
calculate the boundaries:
IQR = dataset.Alkaline_Phosphotase.quantile(0.75) -
dataset.Alkaline_Phosphotase.quantile(0.25)

# Extreme outliers
lower_bridge = dataset['Alkaline_Phosphotase'].quantile(0.25) - (IQR*3)
upper_bridge = dataset['Alkaline_Phosphotase'].quantile(0.75) + (IQR*3)

print(lower_bridge)
print(upper_bridge)

# if value greater than upper bridge, we replace that value with upper_bridge
value:
dataset.loc[dataset['Alkaline_Phosphotase'] >= upper_bridge,
'Alkaline_Phosphotase'] = upper_bridge
```

```
# %%
# Lets compute the Interquantile range of Alamine_Aminotransferase feature to
calculate the boundaries:
IQR = dataset.Alamine_Aminotransferase.quantile(0.75) -
dataset.Alamine_Aminotransferase.quantile(0.25)

# Extreme outliers
lower_bridge = dataset['Alamine_Aminotransferase'].quantile(0.25) - (IQR*3)
upper_bridge = dataset['Alamine_Aminotransferase'].quantile(0.75) + (IQR*3)

print(lower_bridge)
print(upper_bridge)

# if value greater than upper bridge, we replace that value with upper_bridge
value:
dataset.loc[dataset['Alamine_Aminotransferase'] >= upper_bridge,
'Alamine_Aminotransferase'] = upper_bridge

# %%
# Lets compute the Interquantile range of Aspartate_Aminotransferase feature to
calculate the boundaries:
IQR = dataset.Aspartate_Aminotransferase.quantile(0.75) -
dataset.Aspartate_Aminotransferase.quantile(0.25)

# Extreme outliers
lower_bridge = dataset['Aspartate_Aminotransferase'].quantile(0.25) - (IQR*3)
upper_bridge = dataset['Aspartate_Aminotransferase'].quantile(0.75) + (IQR*3)

print(lower_bridge)
print(upper_bridge)

# if value greater than upper bridge, we replace that value with upper_bridge
value:
dataset.loc[dataset['Aspartate_Aminotransferase'] >= upper_bridge,
'Aspartate_Aminotransferase'] = upper_bridge

# %%
# Lets compute the Interquantile range of Albumin_and_Globulin_Ratio feature to
calculate the boundaries
IQR = dataset.Albumin_and_Globulin_Ratio.quantile(0.75) -
dataset.Albumin_and_Globulin_Ratio.quantile(0.25)

# Extreme outliers
lower_bridge = dataset['Albumin_and_Globulin_Ratio'].quantile(0.25) - (IQR*3)
upper_bridge = dataset['Albumin_and_Globulin_Ratio'].quantile(0.75) + (IQR*3)
```

```python
print(lower_bridge)
print(upper_bridge)

# if value greater than upper bridge, we replace that value with upper_bridge
value:
dataset.loc[dataset['Albumin_and_Globulin_Ratio'] >= upper_bridge,
'Albumin_and_Globulin_Ratio'] = upper_bridge

# %%
# Top 5 records:
dataset.head()

# %%
# Description after deal with outliers by IQR:
dataset.describe()

# %%
# Independent and Dependent Feature:
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]

# %%
# top 5 records of Independent features:
X.head()

# %%
# top 5 records of dependent features:
y.head()

# %%
# SMOTE Technique:
from imblearn.combine import SMOTETomek
smote = SMOTETomek()
X_smote, y_smote = smote.fit_resample(X,y)

# %%
# Counting before and after SMOTE:
from collections import Counter
print('Before SMOTE : ', Counter(y))
print('After SMOTE  : ', Counter(y_smote))

# %%
# Train Test Split:
from sklearn.model_selection import train_test_split
```

```python
X_train,X_test,y_train,y_test = train_test_split(X_smote,y_smote, test_size=0.3,
random_state=33)

# %%
print(X_train.shape)
print(X_test.shape)

# %%
# Feature Importance :
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

### Apply SelectKBest Algorithm
ordered_rank_features=SelectKBest(score_func=chi2,k=9)
ordered_feature=ordered_rank_features.fit(X,y)

dfscores=pd.DataFrame(ordered_feature.scores_,columns=["Score"])
dfcolumns=pd.DataFrame(X.columns)

features_rank=pd.concat([dfcolumns,dfscores],axis=1)

features_rank.columns=['Features','Score']
features_rank.nlargest(9, 'Score')

# %% [markdown]
# #### There is no need of Standardization and Normalization of our dataset, as
we using Ensemble Technique.

# %%
# Importing Performance Metrics:
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

# %%
# RandomForestClassifier:
from sklearn.ensemble import RandomForestClassifier
RandomForest = RandomForestClassifier()
RandomForest = RandomForest.fit(X_train,y_train)

# Predictions:
y_pred = RandomForest.predict(X_test)

# Performance:
print('Accuracy:', accuracy_score(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))

# %%
# AdaBoostClassifier:
from sklearn.ensemble import AdaBoostClassifier
AdaBoost = AdaBoostClassifier()
AdaBoost = AdaBoost.fit(X_train,y_train)

# Predictions:
y_pred = AdaBoost.predict(X_test)

# Performance:
print('Accuracy:', accuracy_score(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))

# %%
# GradientBoostingClassifier:
from sklearn.ensemble import GradientBoostingClassifier
GradientBoost = GradientBoostingClassifier()
GradientBoost = GradientBoost.fit(X_train,y_train)

# Predictions:
y_pred = GradientBoost.predict(X_test)

# Performance:
print('Accuracy:', accuracy_score(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))

# %% [markdown]
# ####  RandomizedSearchCV

# %%
# Importing RandomizedSearchCV:
from sklearn.model_selection import RandomizedSearchCV

# %%
# Number of trees in random forest:
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 2000, num = 20)]

# Number of features to consider at every split:
max_features = ['auto', 'sqrt','log2']

# Maximum number of levels in tree:
```

```python
max_depth = [int(x) for x in np.linspace(100, 100,20)]

# Minimum number of samples required to split a node:
min_samples_split = [1,2,3,4,5,6,7,8,9,10,12,14,16,18,20]

# Minimum number of samples required at each leaf node:
min_samples_leaf = [1,2,3,4,5,6,7,8,9,10,12,14,16,18,20]

# %%
# Create the random grid:
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
              'criterion':['entropy','gini']}
print(random_grid)

# %%
rf = RandomForestClassifier()
rf_randomcv = RandomizedSearchCV(estimator = rf, param_distributions = 
random_grid, n_iter = 100, cv = 5, verbose = 2,
                               random_state = 0, n_jobs = -1)

# fit the randomized model:
rf_randomcv.fit(X_train,y_train)

# %%
# Best parameter of RandomizedSearchCV:
rf_randomcv.best_params_

# %%
# Creating model using best parameter of RandomizedSearchCV:
RandomForest_RandomCV = RandomForestClassifier(criterion = 'entropy', 
n_estimators = 2000, max_depth = 100, max_features = 'log2',
                                               min_samples_split = 3, 
min_samples_leaf = 2)
RandomForest_RandomCV = RandomForest_RandomCV.fit(X_train,y_train)

# Predictions:
y_pred = RandomForest_RandomCV.predict(X_test)

# Performance:
print('Accuracy:', accuracy_score(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))

# %% [markdown]
# #### GridSearchCV

# %%
# Importing GridSearchCV:
from sklearn.model_selection import GridSearchCV

# %%
# Best parameter:
rf_randomcv.best_params_

# %%
param_grid = {
    'criterion': [rf_randomcv.best_params_['criterion']],
    'max_features': [rf_randomcv.best_params_['max_features']],
    'max_depth': [rf_randomcv.best_params_['max_depth']-50,
                  rf_randomcv.best_params_['max_depth'],
                 rf_randomcv.best_params_['max_depth']+50],
    'min_samples_leaf': [rf_randomcv.best_params_['min_samples_leaf']-1,
                         rf_randomcv.best_params_['min_samples_leaf'],
                         rf_randomcv.best_params_['min_samples_leaf']+1],
    'min_samples_split': [rf_randomcv.best_params_['min_samples_split'] - 1,
                          rf_randomcv.best_params_['min_samples_split'],
                          rf_randomcv.best_params_['min_samples_split'] +1],
    'n_estimators': [rf_randomcv.best_params_['n_estimators'] - 50,
                     rf_randomcv.best_params_['n_estimators'],
                     rf_randomcv.best_params_['n_estimators'] + 50]
}

print(param_grid)

# %%
# Fit the grid_search to the data:
rf = RandomForestClassifier()
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid, cv=5 , n_jobs
= -1, verbose = 2)
grid_search.fit(X_train,y_train)

# %%
# Best Parameter of GridSearchCV:
grid_search.best_params_

# %%
# Creating model using best parameter of GridSearchCV:
RandomForest_gridCV = RandomForestClassifier(criterion='entropy',
n_estimators=1950, max_depth=150, max_features='log2',
                                             min_samples_split=2,
min_samples_leaf=1)
RandomForest_gridCv = RandomForest_gridCV.fit(X_train,y_train)

# Predictions:
y_pred = RandomForest_gridCV.predict(X_test)

# Performance:
print('Accuracy:', accuracy_score(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))

# %% [markdown]
# #### - We saw that after doing RandomizedSearchCV and GridSearchCV, Our
accuracy, Precision, Recall, f1-Score doesn't increase.

# %%
# Creating a pickle file for the classifier
import pickle
filename = 'Liver.pkl'
pickle.dump(RandomForestClassifier, open(filename, 'wb'))

# %%
```

10.2. GitHub & Project Demo Link

Git hub Link :

https://github.com/rewanth05/Prediction-and-analysis-of-liver-patient-data-using-ml

Project Demo Link

https://drive.google.com/file/d/1n2XRmuRjGH2yhpRZ8UkWf1nHq6aKzrvt/view?usp=sharing