Prediction And Analysis Of Liver Patient Data Using Machine Learning

Introduction

1.1. Project overviews

1.2. Objectives

2. Project Initialization and Planning Phase

2.1. Define Problem Statement

2.2. Project Proposal (Proposed Solution)

2.3. Initial Project Planning

3. Data Collection and Preprocessing Phase

3.1. Data Collection Plan and Raw Data Sources Identified

3.2. Data Quality Report

3.3. Data Exploration and Preprocessing

4. Model Development Phase

4.1. Feature Selection Report

4.2. Model Selection Report

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

5. Model Optimization and Tuning Phase

5.1. Hyperparameter Tuning Documentation

5.2. Performance Metrics Comparison Report

5.3. Final Model Selection Justification

6. Results

6.1. Output Screenshots

7. Advantages & Disadvantages

8. Conclusion

9. Future Scope

10. Appendix

    10.1. Source Code

    10.2. GitHub & Project Demo Link

1.INTRODUCTION

1.1 Project overview

Liver is the largest organ in the abdomen. This is the primary organ for maintaining the chemicals like glucose, balancing so many nutrients, fat, vitamins, cholesterol and hormones. Liver disease prevents normal liver function. Mainly due to the large amount of alcohol consumption liver disease develops. Early detection of liver disease using classification algorithms is an effective activity that can help doctors diagnose the disease in a short period of time. Early detection of liver disease is a daunting task for doctors. The main purpose of our project is to analyse the parameters of the various classification algorithms and compare their predictive accuracy to determine the best stage for determining liver disease.

1.2 Objectives

The primary objective is to develop a machine learning model that accurately predicts the likelihood of a patient developing a liver disease based on their medical history, symptoms, and other relevant data, enabling early intervention and improved health outcomes

2. Project Initialization and Planning Phase

2.1 Define Problem Statement

The early prediction of liver disease can significantly improve patient outcomes, reduce healthcare costs, and enhance the quality of life. With the advent of big data, machine learning, and advanced analytics, there is a growing potential to predict diseases before they manifest clinically. This predictive capability can aid in proactive treatment, preventive measures, and personalized healthcare. In the realm of healthcare, accurately predicting the onset and progression of disease is crucial for early intervention and improved patient outcomes. Despite advancements in medical technology and data analytics, current methods often lack precision and timeliness, leading to delayed diagnoses and suboptimal treatment plans. This project aims to develop a robust and scalable liver disease prediction model that leverages patient data, including medical history, genetic information, lifestyle factors, and real-time health indicators, to forecast the likelihood of disease occurrence with high accuracy. The goal is to enhance

predictive capabilities, reduce healthcare costs, and ultimately improve patient care through timely and personalized interventions.

2.2 Project Proposal (Proposed solution)

This project proposal outlines a solution to address the problem of early liver disease detection through machine learning. With a clear objective to develop a predictive model for assessing disease risk based on symptoms, lifestyle factors, and health data, the proposal defines the scope of the project, including data collection, model development, and deployment. The proposed solution details the approach to be used, key features of the model, and specifies the resource requirements including hardware, software, and personnel. By creating an accurate and user friendly tool, the project aims to enable proactive health management and improve early liver disease detection.

2.3 Initial Project Planning

Develop a machine learning model to predict the likelihood of a liver disease based on patient data. The scope includes data collection, basic preprocessing, model development, and initial evaluation for accuracy. The model will be deployed in a simple application for healthcare use. Ongoing maintenance and updates are not included.

3. Data Collection and Preprocessing Phase

3.1 Data Collection Plan and Raw Data Sources Identified

- **Data Availability**: Determine which sources provide the required data and whether they are accessible.
- **Data Quality**: Evaluate the reliability, completeness, and accuracy of the data from each source.
- **Data Consistency**: Ensure consistency across different sources and data formats.Time: time in format hh:mm:ss

- **Data Extraction**: Design protocols for extracting data from electronic health records (EHRs), databases, or files.
- **Data Integration**: Merge data from multiple sources into a unified dataset suitable for analysis.
- **Data Cleaning**: Preprocess data to handle missing values, outliers, and inconsistencies.
- **Hospital EHRs**: Patient demographics, medical history, medications, and laboratory results.
- **Public Datasets**: Liver disease datasets from repositories like UCI Machine Learning Repository.
- **Research Databases**: Specific datasets from liver disease research studies.
- **Health Insurance Claims**: Historical claims data with diagnostic codes related to liver diseases.

- **3.2 Data Quality Report**

- Data quality is ensured through thorough verification, addressing missing values, and maintaining adherence to ethical guidelines, establishing a reliable foundation for predictive modeling.

3.3 Data Exploration and preprocessing

- Data Exploration involves analyzing the liver disease patient dataset to understand patterns, distributions, and outliers.
- Preprocessing includes handling missing values, scaling, and encoding categorical variables.
- These crucial steps enhance data quality, ensuring the reliability and effectiveness of subsequent analysis.

4. Model Development Phase

4.1 Feature Selection Report

- The Feature Selection Report outlines the rationale behind choosing specific features (e.g., age, food plan, gender) for the liver disease prediction model.
- It evaluates relevance, importance, and impact on predictive accuracy, ensuring the inclusion of key factors influencing the model's ability.

4.2 Model Selection Report

- The Model Selection Report details the rationale behind choosing Random Forest, KNN, and SVM models for liver disease prediction.
- It considers each model's strengths in handling complex relationships, interpretability, adaptability, and overall predictive performance, ensuring an informed choice aligned with project objectives.

4.3 Initial Model Training Code, Model Validation and Evaluation Report

- The Initial Model Training Code employs selected algorithms on the liver disease dataset, setting the foundation for predictive modeling.
- The subsequent Model Validation and Evaluation Report rigorously assesses model performance, employing classification metrics f_score, recall, precision ensure reliability and effectiveness in predicting liver disease.

5.Model Optimization and Tuning Phase

Final Model Selection Justification

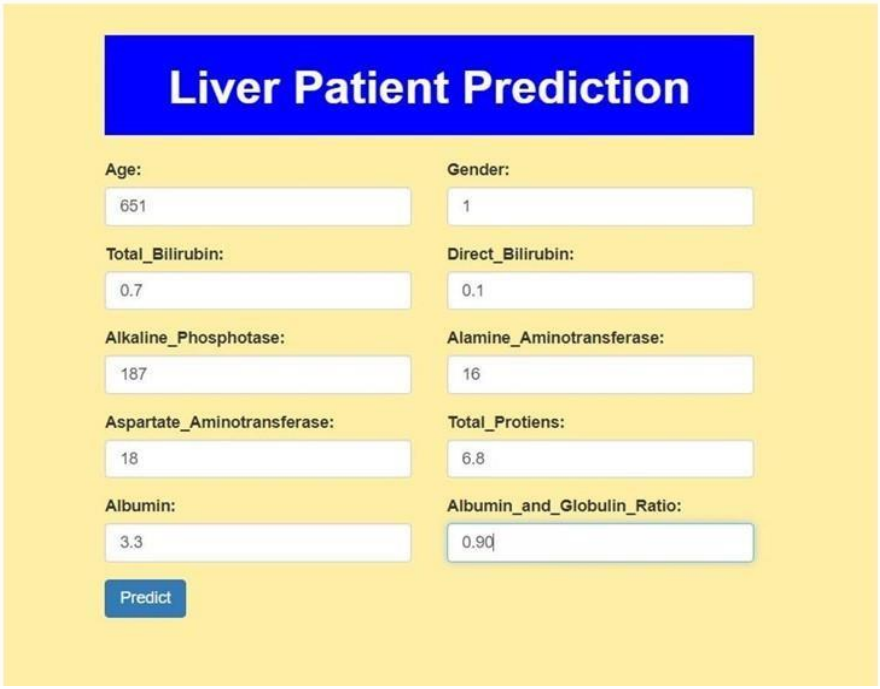
- The SVM Regressor is the final model chosen because of its best overall performance compared to the other models.

6. RESULTS

6.1 Output Screenshots

PCA.HTML

HOME PAGE



OUTPUT PAGE

RESULT.HTML

7.ADVANTAGES AND DISADVANTAGES

Advantages:

A. It can be widely used in Medical Field to predict liver disease.

B. It would reduce the burden on doctors and medical staff leading to a more accurate treatment of the patients.

Disadvantages:

A. The cost of misclassification of a single patient can be very high.

B. Even the best performing Machine Learning Algorithm 20

8.Conclusion

The main objective of this study was to provide a summary of the classification algorithms in the field of data driven predictive data for liver disease. In this study, various classification algorithms were analysed to help doctors predict liver disease early. The purpose of this study was achieved by conducting comparative research in various papers. Based on this study, Random Forest has a much higher accuracy than other algorithms and can be used continuously in predicting user-recommended liver disease.

9. FUTURE SCOPE

- Improved Accuracy: As more data becomes available and algorithms become more sophisticated, the accuracy of predicting liver disease will continue to improve. This includes both diagnosing the presence of liver disease and predicting its progression.
- Early Detection: Machine learning models can potentially detect liver disease at earlier stages than traditional methods, allowing for timely interventions and better outcomes for patients.
- Personalized Medicine: ML algorithms can analyze large datasets to identify patterns and correlations that can lead to personalized treatment plans. This could involve tailoring treatment strategies based on individual patient data, genetics, lifestyle factors, etc.
- Integration with Healthcare Systems: There is a growing trend towards integrating machine learning models into electronic health records (EHRs) and clinical decision support systems. This integration can streamline diagnosis and treatment planning processes.

- Feature Importance and Interpretability: Future research can focus on improving the interpretability of machine learning models used for liver disease prediction. Understanding which features (e.g., biomarkers, genetic markers) contribute most to predictions can enhance medical understanding and guide further research.
- Handling Imbalanced Data: Imbalanced datasets (where one class is more prevalent than another) are common in medical data. Future research can explore techniques to handle imbalanced data effectively, ensuring that models are robust and reliable.
- Telemedicine and Remote Monitoring: ML models can support telemedicine initiatives by enabling remote monitoring of liver disease patients. This can include predicting exacerbations, monitoring treatment effectiveness, and providing real-time alerts for healthcare providers.
- Drug Discovery and Development: Machine learning can aid in identifying potential drug targets and predicting the efficacy of treatments for liver disease. This can accelerate the drug discovery process and improve outcomes for patients with liver diseases.
- Ethical Considerations: As with any AI application in healthcare, there will be ongoing discussions about ethical considerations such as patient privacy, consent, and the responsible use of AI in medical decision-making.
- Global Health Impact: Machine learning can potentially have a significant impact on global health by improving access to diagnostic tools and personalized treatment options, particularly in underserved areas where access to healthcare is limited.

10.Appendix

10.1. Source Code

Code Snippets

```
# showing the data from top 5
data.head()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferas
0	65	Female	0.7	0.1	187	16	18
1	62	Male	10.9	5.5	699	64	100
2	62	Male	7.3	4.1	490	60	68
3	58	Male	1.0	0.4	182	14	20
4	72	Male	3.9	2.0	195	27	59

```
data.tail()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransfer
578	60	Male	0.5	0.1	500	20	34
579	40	Male	0.6	0.1	98	35	31
580	52	Male	0.8	0.2	245	48	49
581	31	Male	1.3	0.5	184	29	32
582	38	Male	1.0	0.3	216	21	24

```
data.info()
```

```
<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
```

```
data.describe()
```

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransfera
count	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000
mean	44.746141	3.298799	1.486106	290.576329	80.713551	109.910806
std	16.189833	6.209522	2.808498	242.937989	182.620356	288.918529
min	4.000000	0.400000	0.100000	63.000000	10.000000	10.000000
25%	33.000000	0.800000	0.200000	175.500000	23.000000	25.000000
50%	45.000000	1.000000	0.300000	208.000000	35.000000	42.000000
75%	58.000000	2.600000	1.300000	298.000000	60.500000	87.000000
max	90.000000	75.000000	19.700000	2110.000000	2000.000000	4929.000000

```
data.isnull().any()
```

```
Age                         False
Gender                      False
Total_Bilirubin             False
Direct_Bilirubin            False
Alkaline_Phosphotase        False
Alamine_Aminotransferase    False
Aspartate_Aminotransferase  False
Total_Protiens              False
Albumin                     False
Albumin_and_Globulin_Ratio   True
Dataset                     False
dtype: bool
```

```
#checking for missing data
data.isnull().sum()
```

```
Age                         0
Gender                      0
Total_Bilirubin             0
Direct_Bilirubin            0
Alkaline_Phosphotase        0
Alamine_Aminotransferase    0
Aspartate_Aminotransferase  0
Total_Protiens              0
Albumin                     0
Albumin_and_Globulin_Ratio  4
Dataset                     0
dtype: int64
```

```
#checking for the missing data after cleaning data
data['Albumin_and_Globulin_Ratio'] = data.fillna(data['Albumin_and_Globulin_Ratio'].mode()[0])
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
```

```
# dividing the data into input and output
x=data.iloc[:,0:-1]
y=data.iloc[:,-1]
```

```
# importing the train_test_split from scikit-learn
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2)
```

```
# Importing the machine learning model
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
```

```
# Initializing the machine learning models
svm=SVC()
RFmodel=RandomForestClassifier()
KNNmodel=KNeighborsClassifier()
```

```
#Support Vector Machine Model
from sklearn.svm import SVC
svm=SVC()

# train the data with SVM model
svm.fit(xtrain, ytrain)
```

```
SVC()
```

```
#Random Forest Classifier Model
from sklearn.ensemble import RandomForestClassifier
RFmodel=RandomForestClassifier()

# train the data with Random Forest model
RFmodel.fit(xtrain, ytrain)
```

```
RandomForestClassifier()
```

```
# Checking for accuracy score from actual data and predicted data
SVMaccuracy=accuracy_score(SVMpred, ytest)
SVMaccuracy
```

```
0.7606837606837606
```

```
#Random Forest Classifier Model
from sklearn.ensemble import RandomForestClassifier
RFmodel=RandomForestClassifier()
```

```
# train the data with Random Forest model
RFmodel.fit(xtrain, ytrain)
```

```
RandomForestClassifier()
```

```
RFpred=RFmodel.predict(xtest)
```

```
# Checking for accuracy score from actual data and predicted data
RFaccuracy=accuracy_score(RFpred, ytest)
RFaccuracy
```

```
0.7094017094017094
```

```
# showing the confusion matrix
RFcm=confusion_matrix(RFpred, ytest)
RFcm
```

```
array([[77, 22],
       [12,  6]], dtype=int64)
```

```
# K-Nearest Neighbors Model
from sklearn.neighbors import KNeighborsClassifier
KNN = KNeighborsClassifier()
```

```
# train the data with K-Nearest Neighbors Model
KNN.fit(xtrain, ytrain)
```

```
KNeighborsClassifier()
```

```
KNNpred=KNN.predict(xtest)
```

```
# Checking for accuracy score from actual data and predicted data
KNNaccuracy=accuracy_score(KNNpred, ytest)
KNNaccuracy
```

```
0.6495726495726496
```

```
# showing the confusion matrix
KNNcm=confusion_matrix(KNNpred, ytest)
KNNcm
```

```
array([[70, 22],
       [19,  6]], dtype=int64)
```

```
# saving the model
import pickle
pickle.dump(svm, open('liver_analysis.pkl','wb'))
```

```
from flask import Flask, render_template, request # Flask is a application
# used to run/serve our application
# request is used to access the file which is uploaded by the user in out application
# render_template is used for rendering the html pages
import pickle # pickle is used for serializing and de-serializing Python object structures
```

```python
@app.route('/') # rendering the html template
def home():
    return render_template('home.html')
@app.route('/predict') # rendering the html template
def index() :
    return render_template("index.html")
```

```python
@app.route('/data_predict', methods=['POST']) # route for our prediction
def predict():
    age = request.form['age'] # requesting for age data
    gender = request.form['gender'] # requesting for gender data
    tb = request.form['tb'] # requesting for Total_Bilirubin data
    db = request.form['db'] # requesting for Direct_Bilirubin data
    ap = request.form['ap'] # requesting for Alkaline_Phosphotase data
    aa1 = request.form['aa1'] # requesting for Alamine_Aminotransferase data
    aa2 = request.form['aa2'] # requesting for Aspartate_Aminotransferase data
    tp = request.form['tp'] # requesting for Total_Protiens data
    a = request.form['a'] # requesting for Albumin data
    agr = request.form['agr'] # requesting for Albumin_and_Globulin_Ratio data

    # coverting data into float format
    data = [[float(age), float(gender), float(tb), float(db), float(ap), float(aa1), float(aa2), float(tp),

    # loading model which we saved
    model = pickle.load(open('liver_analysis.pkl', 'rb'))

    prediction= model.predict(data)[0]
    if (prediction == 1):
        return render_template('noChance.html', prediction='You have a liver desease problem, You must and
    else:
        return render_template('chance.html', prediction='You dont have a liver desease problem')

if __name__ == '__main__':
    app.run()
```

```python
app=Flask(__name__) # our flask app

@app.route('/') # rendering the html template
def home():
    return render_template('home.html')
@app.route('/predict') # rendering the html template
def index() :
    return render_template("index.html")

@app.route('/data_predict', methods=['POST']) # route for our prediction
def predict():
    age = request.form['age'] # requesting for age data
    gender = request.form['gender'] # requesting for gender data
    tb = request.form['tb'] # requesting for Total_Bilirubin data
    db = request.form['db'] # requesting for Direct_Bilirubin data
    ap = request.form['ap'] # requesting for ALkaline_Phosphotase data
    aa1 = request.form['aa1'] # requesting for ALamine_Aminotransferase data
    aa2 = request.form['aa2'] # requesting for Aspartate_Aminotransferase data
    tp = request.form['tp'] # requesting for Total_Protiens data
    a = request.form['a'] # requesting for ALbumin data
    agr = request.form['agr'] # requesting for ALbumin_and_Globulin_Ratio data

    # coverting data into float format
    data = [[float(age), float(gender), float(tb), float(db), float(ap), float(aa1), float(aa2), float(tp),

    # loading model which we saved
    model = pickle.load(open('liver_analysis.pkl', 'rb'))

    prediction= model.predict(data)[0]
    if (prediction == 1):
        return render_template('noChance.html', prediction='You have a liver desease problem, You must and :
    else:
        return render_template('chance.html', prediction='You dont have a liver desease problem')

if __name__ == '__main__':
    app.run()
```

```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off

 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Predict.HTML:

```
<!DOCTYPE html>
<html>
<head>
  <title>Liver Patient Prediction</title>
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.mi
n.css">
  <style type="text/css">
    body {
      background-color: #ffcf0059;
    }
    .page-header {
      background-color: blue;
      width: 100%;
      height: auto;
      text-align: center;
      padding-top: 5px;
      color: #fff;
    }
    h1 {
      font-size: 40px;
      font-weight: bold;
```

```
    }
  </style>
</head>
<body>

  <div class="container">
    <div class="row">
      <div class="col-md-3"></div>
      <div class="col-md-6">
        <div class="page-header">
          <h1>Liver Patient Prediction</h1>
        </div>
      </div>
    </div>
  </div>

  <div class="container">
    <div class="row">
      <div class="col-md-3"></div>
      <div class="col-md-6">
        <form action="/data_predict" method="POST">
          <div class="row">
            <div class="col-md-6">
              <div class="form-group">
```

```
            <label for="age">Age:</label>
            <input type="text" class="form-control" id="age"
name="age">
          </div>
        </div>
 <div class="col-md-6">
          <div class="form-group">
            <label for="gender">Gender:</label>
            <input type="text" class="form-control" id="gender"
name="gender" placeholder="Enter 0 as male, 1 as female">
          </div>
        </div>
      </div>
      <div class="row">
        <div class="col-md-6">
          <div class="form-group">
            <label for="tb">Total_Bilirubin:</label>
            <input type="text" class="form-control" id="tb" name="tb">
          </div>
        </div>
        <div class="col-md-6">
          <div class="form-group">
            <label for="db">Direct_Bilirubin:</label>
            <input type="text" class="form-control" id="db" name="db">
```

```
    </div>
  </div>
</div>

<div class="row">
  <div class="col-md-6">
    <div class="form-group">
      <label for="ap">Alkaline_Phosphotase:</label>
      <input type="text" class="form-control" id="ap" name="ap">
    </div>
  </div>

  <div class="col-md-6">
    <div class="form-group">
      <label for="aa1">Alamine_Aminotransferase:</label>
      <input type="text" class="form-control" id="aa1"
name="aa1">
    </div>
  </div>
</div>

<div class="row">
  <div class="col-md-6">
    <div class="form-group">
```

```
            <label for="aa2">Aspartate_Aminotransferase:</label>
            <input type="text" class="form-control" id="aa2"
name="aa2">
          </div>
        </div>
        <div class="col-md-6">
          <div class="form-group">
            <label for="tp">Total_Protiens:</label>
            <input type="text" class="form-control" id="tp" name="tp">
          </div>
        </div>
      </div>

      <div class="row">
        <div class="col-md-6">
          <div class="form-group">
            <label for="a">Albumin:</label>
            <input type="text" class="form-control" id="a" name="a">
          </div>
        </div>

        <div class="col-md-6">
          <div class="form-group">
            <label for="agr">Albumin_and_Globulin_Ratio:</label>
```

```
            <input type="text" class="form-control" id="agr"
name="agr">
          </div>
        </div>
      </div>
 <button type="submit" class="btn btn-primary">Predict</button>
     </form>
    </div>
  </div>
 </div>
</body>
</html>
```

Result.html:

```
<!DOCTYPE html>
<html>
<head>
 <title>Prediction Result</title>
 <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.mi
n.css">
 <style type="text/css">
  body {
   background-color: #ffcf0059;
  }
  .page-header {
```

```
    background-color: blue;
    width: 100%;
    height: auto;
    text-align: center;
    padding-top: 5px;
    color: #fff;
   }
   h1 {
    font-size: 40px;
    font-weight: bold;
   }
  </style>
</head>
<body>

  <div class="container">
   <div class="row">
    <div class="col-md-3"></div>
    <div class="col-md-6">
      <div class="page-header">
       <h1>Prediction Result</h1>
      </div>
    </div>
   </div>
```

```
 </div>

 <div class="container">
  <div class="row">
    <div class="col-md-3"></div>
    <div class="col-md-6">
      <div class="alert alert-info">
        <strong>Result: {{ prediction }}</strong>
      </div>
    </div>
  </div>
 </div>

</body>
</html>
```

App.py:

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

# app = Flask(__name__)
# model = pickle.load(open('indian_liver_patient.pkl', 'rb'))

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

```
# def home():
#     return render_template('index.html')


# @app.route('/predict', methods=['POST'])
# def predict():
#     '''
#     For rendering results on HTML GUI
#     '''
#     try:
#         int_features = [int(x) for x in request.form.values()]
#         final_features = [np.array(int_features)]
#         prediction = model.predict(final_features)
#         output = round(prediction[0], 2)
#         return render_template('index.html', prediction_text='Predicted Value: $
{}'.format(output))
#     except Exception as e:
#         return str(e)


# @app.route('/predict_api', methods=['POST'])
# def predict_api():
#     '''
#     For direct API calls through request
#     '''
#     try:
```

```
#         data = request.get_json(force=True)
#         prediction = model.predict([np.array(list(data.values()))])
#         output = prediction[0]
#         return jsonify(output)
#     except Exception as e:
#         return jsonify({'error': str(e)})

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



"""from flask import Flask, request, render_template
import pandas as pd
import pickle

app = Flask(__name__)

# Load the trained model
try:
    with open('indian_liver_patient.pkl', 'rb') as model_file:
        model = pickle.load(model_file)
except (EOFError, FileNotFoundError) as e:
```

```
    model = None
    print(f"Error loading model: {e}")

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



@app.route('/predict', methods=['GET', 'POST'])
def predict():
    if request.method == 'POST':
        try:
            if model is None:
                raise ValueError("Model not loaded properly")

            # Get input data from form
            data = request.form.to_dict()
            data = {k: float(v) for k, v in data.items()}

            # Convert to DataFrame
            df = pd.DataFrame([data])

            # Make prediction
```

```
            prediction = model.predict(df)

            # Return result
            result = 'Liver disease' if prediction[0] == 1 else 'No liver disease'
            return render_template('result.html', result=result)

        except Exception as e:
            return str(e)
    return render_template('predict.html')

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



from flask import Flask, request, render_template
import pandas as pd
import pickle

app = Flask(__name__)

# Load the model once during the app initialization
model = pickle.load(open('liver_analysis.pkl', 'rb'))
```

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


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


@app.route('/data_predict', methods=['POST'])
def predict():
    if request.method == 'POST':
        age = request.form['age']
        gender = request.form['gender']
        tb = request.form['tb']
        db = request.form['db']
        ap = request.form['ap']
        aa1 = request.form['aa1']
        aa2 = request.form['aa2']
        tp = request.form['tp']
        a = request.form['a']
        agr = request.form['agr']


        data = [[float(age), float(gender), float(tb), float(db), float(ap), float(aa1),
float(aa2), float(tp), float(a), float(agr)]]
```

```
        prediction = model.predict(data)[0]
        if prediction == 1:
            result = "You have a liver disease"
        else:
            result = "You don't have a liver disease"

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

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

10.2 GitHub and project Demo link:

Github link: Click Here

Project Demo link: Click Here