Step 1: Load the Dataset

Step 2: Data Exploration

Step 3: Data Cleaning

Handle missing values and invalid entries.

Step 4: Data Preprocessing

Convert categorical variables to numeric using Label Encoding.

Step 5: Prepare Data for Modeling

Separate features and target variable, and split the dataset into training and testing sets.

Split the dataset into training and testing sets

Step 6: Model Training

Step 7: Model Evaluation

Step 8: Save the Model

Save the trained model using pickle for future use.

Step 9: Load and Use the Model for Predictions

Conclusion

This code provides a complete workflow for loading, cleaning, preprocessing, training, evaluating, and saving a model for predicting chronic kidney disease using the provided dataset.

```
#Make necessary imports
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
sns.set()
```

Step 1: Load the Dataset

```
df = pd.read_csv("indian_liver_patient.csv")
df.head()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotr
0	65	Female	0.7	0.1	187	
1	62	Male	10.9	5.5	699	
2	62	Male	7.3	4.1	490	
3	58	Male	1.0	0.4	182	
4	72	Male	3.9	2.0	195	

Step 2: Data Exploration

```
# Check the shape of the dataset
print("Shape of the dataset:", df.shape)
```

```
Shape of the dataset: (583, 11)
<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  583 non-null    float64
 10  Dataset                     583 non-null    int64
dtypes: float64(5), int64(5), object(1)
memory usage: 50.2+ KB
None
Missing values in each column:
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
```

```
# Check data types and missing values
print(df.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  583 non-null    float64
 10  Dataset                     583 non-null    int64
dtypes: float64(5), int64(5), object(1)
memory usage: 50.2+ KB
None
```

Step 3: Data Cleaning

```
# Check for missing values
print("Missing values in each column:")
print(df.isnull().sum())
```

```
Missing values in each column:
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
```

Step 4: Data Preprocessing

```
# Convert categorical variables to numeric
df['Gender'] = df['Gender'].map({'Male': 1, 'Female': 0})
```

Step 5: Prepare Data for Modeling

```
X = df.drop('Dataset', axis=1)  # Features
y = df['Dataset']  # Target variable
```

```
# Split the dataset into training and testing sets

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, ran
```

Step 6: Model Training

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

Step 7: Model Evaluation

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

```
Accuracy: 0.7435897435897436
Confusion Matrix:
 [[75 12]
 [18 12]]
Classification Report:
              precision    recall  f1-score   support

           1       0.81      0.86      0.83        87
```

```
           2       0.50      0.40      0.44        30

    accuracy                           0.74       117
   macro avg       0.65      0.63      0.64       117
weighted avg       0.73      0.74      0.73       117
```

```
print("Accuracy:", accuracy)
```

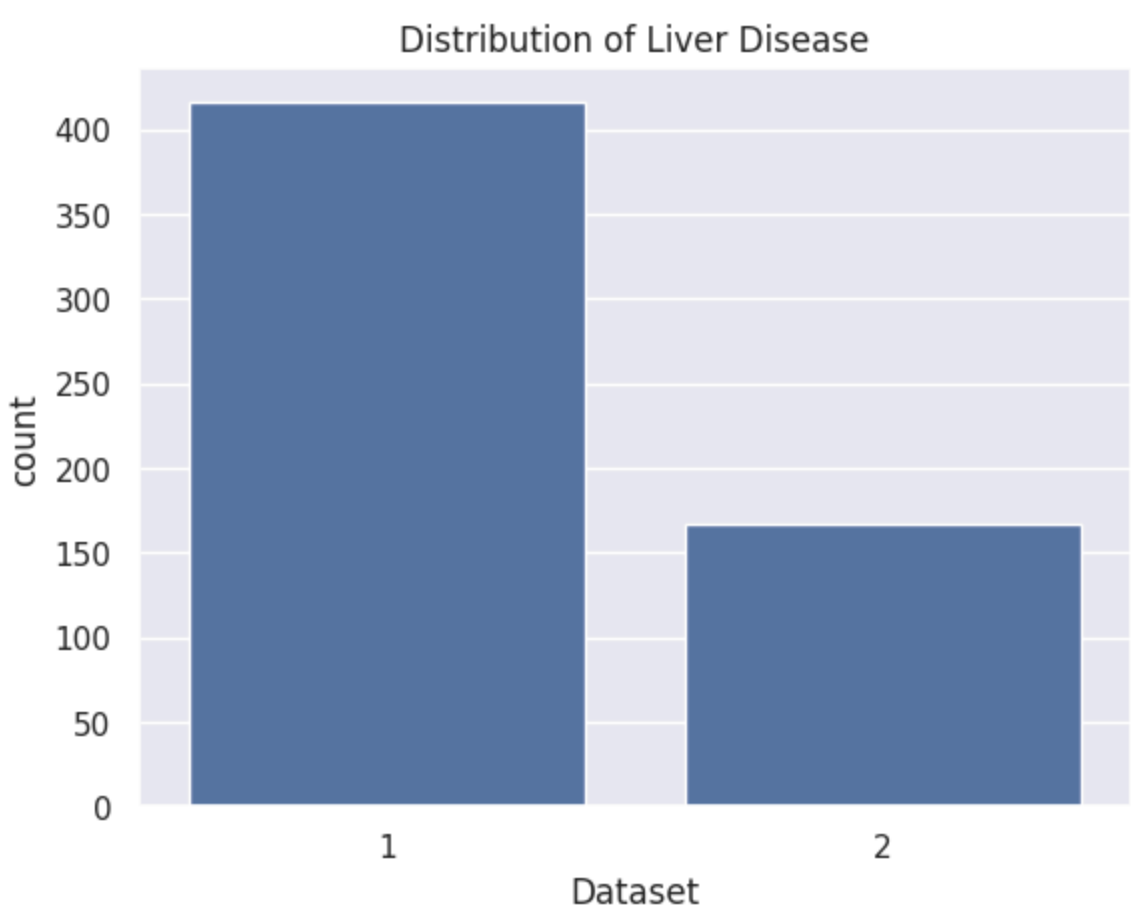
```
Accuracy: 0.7435897435897436
```

```
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

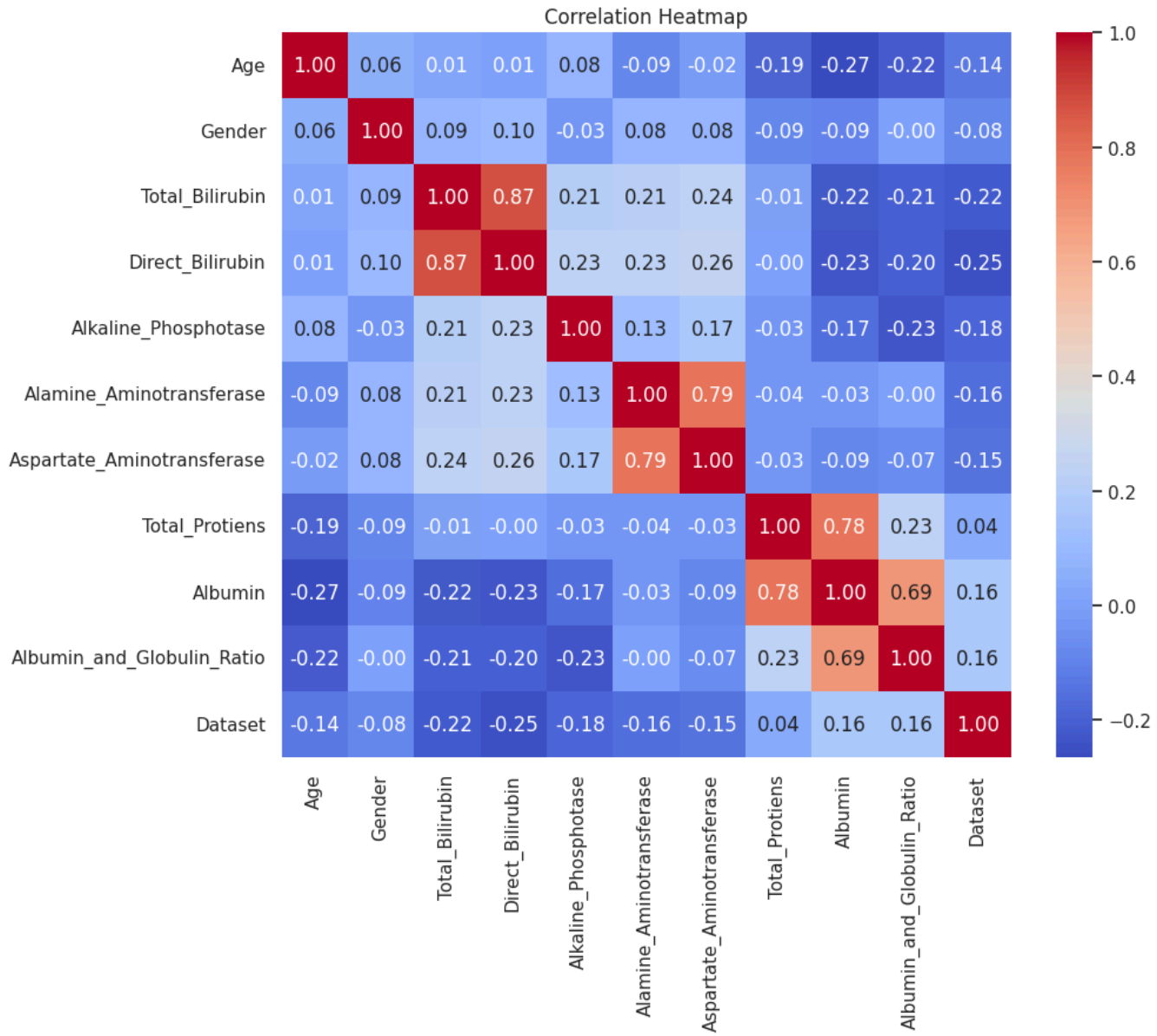
```
Confusion Matrix:
 [[75 12]
 [18 12]]
Classification Report:
               precision    recall  f1-score   support

           1       0.81      0.86      0.83        87
           2       0.50      0.40      0.44        30

    accuracy                           0.74       117
   macro avg       0.65      0.63      0.64       117
weighted avg       0.73      0.74      0.73       117
```

Step 8: Visualizations

```

import matplotlib.pyplot as plt
import seaborn as sns

sns.countplot(x='Dataset', data=df)
plt.title('Distribution of Liver Disease')
plt.show()


```




```
# Correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```


Correlation Heatmap
Age 1.00 0.06 0.01 0.01 0.08 -0.09 -0.02 -0.19 -0.27 -0.22 -0.14
Gender 0.06 1.00 0.09 0.10 -0.03 0.08 0.08 -0.09 -0.09 -0.00 -0.08
Total_Bilirubin 0.01 0.09 1.00 0.87 0.21 0.21 0.24 -0.01 -0.22 -0.21 -0.22
Direct_Bilirubin 0.01 0.10 0.87 1.00 0.23 0.23 0.26 -0.00 -0.23 -0.20 -0.25
Alkaline_Phosphotase 0.08 -0.03 0.21 0.23 1.00 0.13 0.17 -0.03 -0.17 -0.23 -0.18
Alamine_Aminotransferase -0.09 0.08 0.21 0.23 0.13 1.00 0.79 -0.04 -0.03 -0.00 -0.16
Aspartate_Aminotransferase -0.02 0.08 0.24 0.26 0.17 0.79 1.00 -0.03 -0.09 -0.07 -0.15
Total_Protiens -0.19 -0.09 -0.01 -0.00 -0.03 -0.04 -0.03 1.00 0.78 0.23 0.04
Albumin -0.27 -0.09 -0.22 -0.23 -0.17 -0.03 -0.09 0.78 1.00 0.69 0.16
Albumin_and_Globulin_Ratio -0.22 -0.00 -0.21 -0.20 -0.23 -0.00 -0.07 0.23 0.69 1.00 0.16
Dataset -0.14 -0.08 -0.22 -0.25 -0.18 -0.16 -0.15 0.04 0.16 0.16 1.00
Age
Gender
Total_Bilirubin
Direct_Bilirubin
Alkaline_Phosphotase
Alamine_Aminotransferase
Aspartate_Aminotransferase
Total_Protiens
Albumin
Albumin_and_Globulin_Ratio
Dataset
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

A GUI (Graphical User Interface) is a visual way for users to interact with a computer program or application GUIs make it easier for users to interact with applications without needing to type commands or understand complex syntax.

Some common uses of GUIs include:

1. Data analysis and visualization: Interactive dashboards for exploring data.
2. Machine learning model deployment: User-friendly interfaces for model predictions.
3. Productivity tools: Applications for managing tasks, notes, or projects.

wxPython: Cross-platform library for creating native-looking GUIs.

Streamlit: Library for creating web-based GUIs, especially for data science applications.

```
import ipywidgets as widgets
from IPython.display import display

# Create interactive widgets
age_slider = widgets.IntSlider(min=0, max=100, value=30)
gender_dropdown = widgets.Dropdown(options=['Male', 'Female'])

# Display widgets
display(age_slider)
display(gender_dropdown)

# Get values
age = age_slider.value
gender = gender_dropdown.value
print(f"Age: {age}, Gender: {gender}")
```

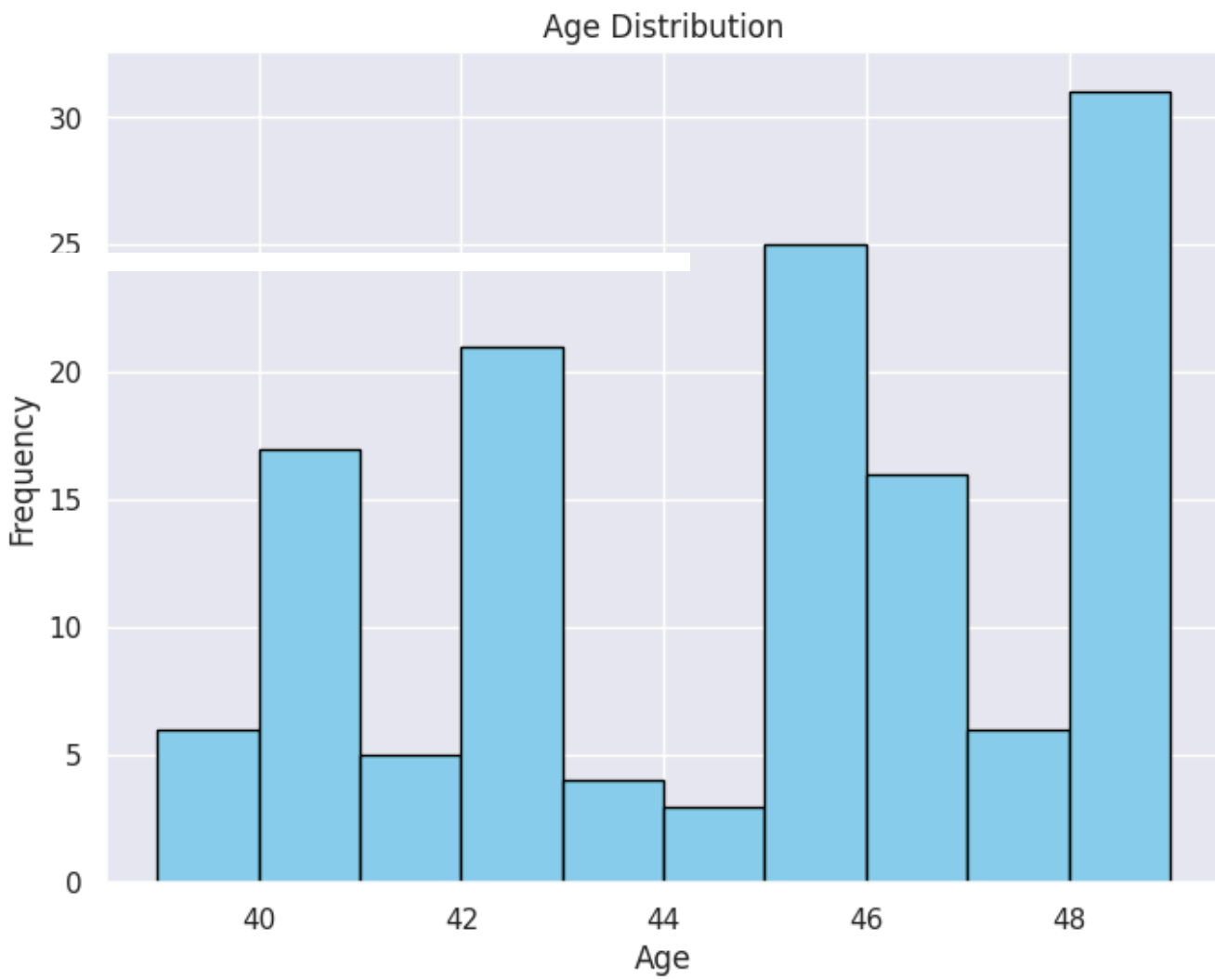

```
Age: 30, Gender: Male
```

```
# Create interactive widgets
dataset_filter = widgets.Dropdown(options=['All', 'Liver Disease', 'No Liver
age_slider = widgets.IntSlider(min=df['Age'].min(), max=df['Age'].max(), val

# Function to update plot based on widget values
def update_plot(dataset_filter_value, age_value):
    filtered_df = df[(df['Age'] >= age_value - 5) & (df['Age'] <= age_value

    if dataset_filter_value == 'Liver Disease':
        filtered_df = filtered_df[filtered_df['Dataset'] == 1]
```

```
    elif dataset_filter_value == 'No Liver Disease':
        filtered_df = filtered_df[filtered_df['Dataset'] == 2]

    plt.figure(figsize=(8, 6))
    plt.hist(filtered_df['Age'], bins=10, color='skyblue', edgecolor='black'
    plt.xlabel('Age')
    plt.ylabel('Frequency')
    plt.title('Age Distribution')
    plt.show()

# Create interactive plot
interactive_plot = widgets.interactive(update_plot, dataset_filter_value=dat

# Display interactive plot
display(interactive_plot)
```

Dataset Filt… All

Age: 44



Streamlit application (app.py) that allows users to interact with the Indian Liver Patient dataset. The app will display the dataset overview, allow users to input patient data, and predict the likelihood of liver disease based on the input features.

```
!wget -q -O - ipv4.icanhazip.com
```

```
34.16.145.86
```

```
%%writefile app.py
import streamlit as st
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Load the dataset
@st.cache_data
def load_data():
    data = pd.read_csv('indian_liver_patient.csv')
    return data

# Preprocess the data
def preprocess_data(data):
    # Handle missing values for numeric columns only
    numeric_cols = data.select_dtypes(include=[np.number]).columns
    data[numeric_cols] = data[numeric_cols].fillna(data[numeric_cols].mean()

    # Encode categorical variables
    label_encoder = LabelEncoder()
    data['Gender'] = label_encoder.fit_transform(data['Gender'])

    return data

# Train the model
@st.cache_data
def train_model(data):
    X = data.drop('Dataset', axis=1)
    y = data['Dataset']

    # Split the data
```

```
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

    # Train a Random Forest Classifier
    model = RandomForestClassifier(random_state=42)
    model.fit(X_train, y_train)

    return model

# Load and preprocess the data
data = load_data()
data = preprocess_data(data)

# Train the model
model = train_model(data)

# Streamlit UI
st.title("Indian Liver Patient Disease Prediction")

st.write("### Dataset Overview")
st.write(f"**Rows (Patients):** {data.shape[0]}")
st.write(f"**Columns (Features):** {data.shape[1]}")
st.write("**Target Column:** `Dataset` (1 = liver disease, 2 = no liver dise

# Input fields for patient data
st.write("### Patient Data Input")
age = st.number_input("Age", min_value=1, max_value=120, value=30)
gender = st.selectbox("Gender", options=["Male", "Female"])
total_bilirubin = st.number_input("Total Bilirubin", min_value=0.0, value=1.
direct_bilirubin = st.number_input("Direct Bilirubin", min_value=0.0, value=
alkaline_phosphotase = st.number_input("Alkaline Phosphotase", min_value=0,
alamine_aminotransferase = st.number_input("Alamine Aminotransferase", min_v
aspartate_aminotransferase = st.number_input("Aspartate Aminotransferase", m
total_proteins = st.number_input("Total Proteins", min_value=0.0, value=6.0,
albumin = st.number_input("Albumin", min_value=0.0, value=3.0, format="%.2f"
albumin_and_globulin_ratio = st.number_input("Albumin and Globulin Ratio", m

# Prepare input data for prediction
input_data = np.array([[age, 1 if gender == "Male" else 0, total_bilirubin,
                        alkaline_phosphotase, alamine_aminotransferase, aspa
                        total_proteins, albumin, albumin_and_globulin_ratio]

# Make prediction
if st.button("Predict"):
    prediction = model.predict(input_data)
    if prediction[0] == 1:
        st.success("The patient is likely to have liver disease.")
```

```
    else:
        st.success("The patient is likely to be healthy.")


```

```
Overwriting app.py
```

```
import streamlit as st
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from io import StringIO

# Load the dataset from uploaded file or default file
@st.cache_data
def load_data(uploaded_file):
    if uploaded_file is not None:
        data = pd.read_csv(uploaded_file)
    else:
        data = pd.read_csv('indian_liver_patient.csv')
    return data

# Preprocess the data
def preprocess_data(data):
    # Handle missing values for numeric columns only
    numeric_cols = data.select_dtypes(include=[np.number]).columns
    data[numeric_cols] = data[numeric_cols].fillna(data[numeric_cols].mean(

    # Encode categorical variables
    label_encoder = LabelEncoder()
    data['Gender'] = label_encoder.fit_transform(data['Gender'])

    return data

# Train the model
@st.cache_data
def train_model(data):
    X = data.drop('Dataset', axis=1)
    y = data['Dataset']

    # Split the data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

    # Train a Random Forest Classifier
```

```
    model = RandomForestClassifier(random_state=42)
    model.fit(X_train, y_train)

    return model

st.title("Indian Liver Patient Disease Prediction")

st.write("### Upload Dataset CSV")
uploaded_file = st.file_uploader("Upload Indian Liver Patient Dataset CSV f

if uploaded_file is not None or st.button("Use Default Dataset"):
    data = load_data(uploaded_file)
    data = preprocess_data(data)
    model = train_model(data)

    st.write("### Dataset Overview")
    st.write(f"**Rows (Patients):** {data.shape[0]}")
    st.write(f"**Columns (Features):** {data.shape[1]}")
    st.write("**Target Column:** `Dataset` (1 = liver disease, 2 = no liver

    # Input fields for patient data
    st.write("### Patient Data Input")
    age = st.number_input("Age", min_value=1, max_value=120, value=30)
    gender = st.selectbox("Gender", options=["Male", "Female"])
    total_bilirubin = st.number_input("Total Bilirubin", min_value=0.0, val
    direct_bilirubin = st.number_input("Direct Bilirubin", min_value=0.0, v
    alkaline_phosphotase = st.number_input("Alkaline Phosphotase", min_valu
    alamine_aminotransferase = st.number_input("Alamine Aminotransferase",
    aspartate_aminotransferase = st.number_input("Aspartate Aminotransferas
    total_proteins = st.number_input("Total Proteins", min_value=0.0, value
    albumin = st.number_input("Albumin", min_value=0.0, value=3.0, format="
    albumin_and_globulin_ratio = st.number_input("Albumin and Globulin Rati

    # Prepare input data for prediction
    input_data = np.array([[age, 1 if gender == "Male" else 0, total_biliru
                            alkaline_phosphotase, alamine_aminotransferase,
                            total_proteins, albumin, albumin_and_globulin_r

    # Make prediction
    if st.button("Predict"):
        prediction = model.predict(input_data)
        if prediction[0] == 1:
            st.success("The patient is likely to have liver disease.")
        else:
            st.success("The patient is likely to be healthy.")
else:
```

```
st.info("Upload a dataset CSV file or click 'Use Default Dataset' to be

```

```
2025-05-16 02:52:45.898 No runtime found, using MemoryCacheStorageManager
2025-05-16 02:52:45.903 No runtime found, using MemoryCacheStorageManager
2025-05-16 02:52:45.908 Thread 'MainThread': missing ScriptRunContext! This warning can
2025-05-16 02:52:45.910 Thread 'MainThread': missing ScriptRunContext! This warning can
2025-05-16 02:52:45.915 Thread 'MainThread': missing ScriptRunContext! This warning can
2025-05-16 02:52:45.918 Thread 'MainThread': missing ScriptRunContext! This warning can
2025-05-16 02:52:45.920 Thread 'MainThread': missing ScriptRunContext! This warning can
2025-05-16 02:52:45.921 Thread 'MainThread': missing ScriptRunContext! This warning can
2025-05-16 02:52:45.926 Thread 'MainThread': missing ScriptRunContext! This warning can
2025-05-16 02:52:45.928 Thread 'MainThread': missing ScriptRunContext! This warning can
2025-05-16 02:52:45.929 Thread 'MainThread': missing ScriptRunContext! This warning can
2025-05-16 02:52:45.934 Thread 'MainThread': missing ScriptRunContext! This warning can
2025-05-16 02:52:45.937 Thread 'MainThread': missing ScriptRunContext! This warning can
2025-05-16 02:52:45.938 Thread 'MainThread': missing ScriptRunContext! This warning can
2025-05-16 02:52:45.940 Thread 'MainThread': missing ScriptRunContext! This warning can
2025-05-16 02:52:45.942 Thread 'MainThread': missing ScriptRunContext! This warning can
2025-05-16 02:52:45.944 Thread 'MainThread': missing ScriptRunContext! This warning can
2025-05-16 02:52:45.945 Thread 'MainThread': missing ScriptRunContext! This warning can
```

```
!wget -q -O - ipv4.icanhazip.com
```

```
34.16.145.86
```

```
!streamlit run app.py & npx localtunnel --port 8501
```