SOURCE CODE

ML CODE:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="white", color_codes=True)
import warnings
warnings.filterwarnings("ignore")
import os

#data preprocessing
# import dataset
Liver=pd.read_csv("C:\sai\liver_data.csv”)
liver.columns
liver.head()
liver.describe()
liver.shape
#exploratory data analysis
liver.dtypes [liver.dtypes=='object']
liver.duplicated()
liver=liver.drop_duplicates()
print(liver.shape)
#Count Not a Number Values in Pandas DataFrame
liver.isna().sum()
sns.boxplot(data=liver,x='Albumin_and_Globulin_Ratio')
liver['Albumin_and_Globulin_Ratio'].mode()
liver['Albumin_and_Globulin_Ratio'].median()
liver['Albumin_and_Globulin_Ratio'].mean()
liver['Albumin_and_Globulin_Ratio']=liver['Albumin_and_Globulin_Ratio'].fillna(liver['
Albumin_and_Globulin_Ratio'].median())
liver.isna().sum()
liver['Dataset']
# Plot histogram grid
liver.hist(figsize=(15,15), xrot=-45, bins=10) ## Display the labels rotated by 45 degress
# Clear the text "residue"
plt.show()
liver.plot(kind="scatter", x="Total_Bilirubin", y="Direct_Bilirubin")
import seaborn as sns
sns.countplot(data = liver,x='Gender',label='count') #counts of observations in each
categorical bin using bars
Male,Female=liver['Gender'].value_counts()
print('Number of patients that are male:',Male)
print('Number of patients that are female:',Female)
sns.jointplot(x="Age", y="Total_Protiens", data=liver, size=5)
```

```
sns.FacetGrid(liver, hue="Total_Protiens")\
   .map(plt.scatter, "Total_Bilirubin", "Direct_Bilirubin")\
   .add_legend()
sns.boxplot(x="Age", y="Total_Protiens", data=liver)
sns.pairplot(liver.drop("Age", axis=1), hue="Total_Protiens", size=3)
liver.corr() # correlation of each column in a DataFrame
corr = liver.corr()
fig, ax = plt.subplots(figsize =(10,10))
sns.heatmap(corr, annot = True,ax = ax,cmap = 'coolwarm')
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
liver.head()
liver['Gender'].replace(to_replace=['Male','Female'], value=[0,1],inplace=True)
liver

#Into the realm of machine learning
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
X = liver.iloc[:, :-1].values
Y = liver.iloc[:, -1].values
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
print(X.shape, Y.shape, X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
#logistic regression
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train, Y_train)

Y_pred = classifier.predict(X_test)

print(classification_report(Y_test, Y_pred))
print(confusion_matrix(Y_test, Y_pred))

from sklearn.metrics import accuracy_score
print('accuracy is',accuracy_score(Y_pred,Y_test))
#knn
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=8)
classifier.fit(X_train, Y_train)

Y_pred = classifier.predict(X_test)

print(classification_report(Y_test, Y_pred))
print(confusion_matrix(Y_test, Y_pred))

from sklearn.metrics import accuracy_score
print('accuracy is',accuracy_score(Y_pred,Y_test))
#support vector machine
from sklearn.svm import SVC
```

```
classifier = SVC()
classifier.fit(X_train, Y_train)

Y_pred = classifier.predict(X_test)

print(classification_report(Y_test, Y_pred))
print(confusion_matrix(Y_test, Y_pred))

from sklearn.metrics import accuracy_score
print('accuracy is',accuracy_score(Y_pred,Y_test))
#random forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification

classifier = RandomForestClassifier(max_depth=2, random_state=0)
classifier.fit(X_train, Y_train)
Y_pred = classifier.predict(X_test)

print(classification_report(Y_test, Y_pred))
print(confusion_matrix(Y_test, Y_pred))
```

FLASK CODE:

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import pickle
from flask import Flask, render_template, request

app = Flask(_name_)

# Load the dataset
liver = pd.read_csv(' C:\sai\liver_data.csv ')

# Preprocess the data
le = LabelEncoder()
liver['Gender'] = le.fit_transform(liver['Gender'])

# Handle missing values
imputer = SimpleImputer(strategy='mean')
liver = pd.DataFrame(imputer.fit_transform(liver), columns=liver.columns)

# Drop rows with missing values in the target variable
liver.dropna(subset=['Dataset'], inplace=True)
```

```
# Split the data into features and target
X = liver.drop(['Dataset', 'Direct_Bilirubin'], axis=1)
y = liver['Dataset'].replace({'Yes': 1, 'No': 0})

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Get the list of feature names
feature_names = X.columns.tolist()

# Fit the LogisticRegression model on the training data
model = LogisticRegression()
model.fit(X_train, y_train)

# Save the model using pickle
filename = 'sai.pkl'
pickle.dump(model, open(filename, 'wb'))

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

@app.route('/predict', methods=['POST'])
def predict():
    # Load the model
    model = pickle.load(open('sai.pkl', 'rb'))

    # Get the input values from the form
    age = float(request.form['Age'])
    gender = int(request.form['Gender'])
    total_bilirubin = float(request.form['Total_Bilirubin'])
    alkaline_phosphotase = float(request.form['Alkaline_Phosphotase'])
    alamine_aminotransferase = float(request.form['Alamine_Aminotransferase'])
    aspartate_aminotransferase = float(request.form['Aspartate_Aminotransferase'])
    total_protiens = float(request.form['Total_Protiens'])
    albumin = float(request.form['Albumin'])
    albumin_globulin_ratio = float(request.form['Albumin_and_Globulin_Ratio'])

    # Create a DataFrame with the input values
    input_data = pd.DataFrame({
        'Age': [age],
        'Gender': [gender],
        'Total_Bilirubin': [total_bilirubin],
        'Alkaline_Phosphotase': [alkaline_phosphotase],
        'Alamine_Aminotransferase': [alamine_aminotransferase],
        'Aspartate_Aminotransferase': [aspartate_aminotransferase],
        'Total_Protiens': [total_protiens],
        'Albumin': [albumin],
        'Albumin_and_Globulin_Ratio': [albumin_globulin_ratio]
```

```
    })

    # Make predictions using the loaded model
    prediction = model.predict(input_data)

    # Render the result template with the prediction
    return render_template('result.html', prediction=prediction[0])

if__name__== '_main_':
    app.run(debug=True)
```