Team ID	PNT2022TMID52707
Project Name	Project - Statistical Machine Learning Approaches to Liver Disease Prediction.

Model building:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearchCV
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,classification_report
```

```
data=pd.read_csv('/content/indian_liver_patient.csv')
```

```
data.info()
```

```
def partition(x):
    if x=='Male':
        return 1
    return 0
data['Gender']=data['Gender'].map(partition)
```

```
def partition(x):
    if x==2:
        return 0
    return 1
data['Dataset']=data['Dataset'].map(partition)
```

```
data['Dataset']
```

```
x=data.drop(columns='Dataset',axis=1)
y=data['Dataset']
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,stratify=y,random_state=42)
```

```
print(x.shape,x_train.shape,x_test.shape)
```

```
(1636, 10) (1145, 10) (491, 10)
```

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
xtrain=sc.fit_transform(x_train)
xtest=sc.transform(x_test)
```

```
def my_confusion_matrix(y_test, y_pred, plt_title, accuracy_title):
    cm=confusion_matrix(y_test, y_pred)
    print(f'{accuracy_title} accuracy score:', '{:.2%}'.format(accuracy_score(y_test, y_pred)))
    print(classification_report(y_test, y_pred))
    sns.heatmap(cm, annot=True, fmt='g', cbar=False, cmap='BuPu')
    plt.xlabel('Predicted Values')
    plt.ylabel('Actual Values')
    plt.title(plt_title)
    plt.show()
    return cm
```

```
random_state = 42
classifier = [KNeighborsClassifier(),
              DecisionTreeClassifier(random_state = random_state),
             RandomForestClassifier(random_state = random_state),
             ]

knn_param_grid = {"n_neighbors": np.linspace(1,19,10, dtype = int).tolist(),
                  "weights": ["uniform","distance"],
                  "metric":["euclidean","manhattan"]}

dt_param_grid = {"min_samples_split" : range(10,500,20),
                 "max_depth": range(1,20,2)}


rf_param_grid = {"max_features": [1,3,10],
                 "min_samples_split":[2,3,10],
                 "min_samples_leaf":[1,3,10],
                "bootstrap":[False],
                 "n_estimators":[100,300],
                 "criterion":["gini"]}
classifier_param = [knn_param_grid,
                    dt_param_grid,
                   rf_param_grid,
                   ]
```

```
cv_result = []
best_estimators = []
for i in range(len(classifier)):
    clf = GridSearchCV(classifier[i], param_grid=classifier_param[i], cv = StratifiedKFold(n_splits = 10), scoring = "accuracy", n_jobs = -1,verbose
    clf.fit(x_train,y_train)
    cv_result.append(clf.best_score_ * 100)
    best_estimators.append(clf.best_estimator_)
    print(cv_result[i])
```

```
cv_results = pd.DataFrame({"Cross Validation Means":cv_result, "ML Models":[ "KNeighborsClassifier", "Decision Tree Classifier",
             "Random Forest Classifier",
            ]})

g = sns.barplot("Cross Validation Means", "ML Models", data = cv_results)
g.set_xlabel("Mean Accuracy")
g.set_title("Cross Validation Scores")
```

```
knn = KNeighborsClassifier(n_neighbors = 9)
knn.fit(x_train, y_train)
y_head_knn = knn.predict(x_test)

dt=DecisionTreeClassifier()
dt.fit(x_train,y_train)
y_head_dt = dt.predict(x_test)

rf = RandomForestClassifier(n_estimators = 250, random_state = 1)
rf.fit(x_train,y_train)
y_head_rf = rf.predict(x_test)
```

```
votingC = VotingClassifier(estimators = [("knn",best_estimators[0]),
                                         ("dt",best_estimators[1]),
                                         ("rf",best_estimators[2])],
                                         voting = "hard", n_jobs = -1)
votingC = votingC.fit(x_train, y_train)
y_pred=votingC.predict(x_test)
my_confusion_matrix(y_test, y_pred, 'Ensemble Model CM', 'Ensemble Model')
```

FLASK APP

main.py

```
from flask import Flask, render_template, request,redirect,session, url_for
from flask_mail import Mail, Message
from itsdangerous import URLSafeTimedSerializer, SignatureExpired
import mysql.connector
import os
import pickle
from flask_login import UserMixin, login_user, LoginManager, login_required, logout_user, current_user

from flask_mysqldb import MySQL

app = Flask(__name__)
app.secret_key=os.urandom(24)
app.config['MYSQL_HOST'] = 'localhost'
app.config['MYSQL_USER'] = 'root'
app.config['MYSQL_PASSWORD'] = ''
app.config['MYSQL_DB'] = 'liver'

mysql = MySQL(app)

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

@app.route('/register/')
def about():
    return render_template('register.html')

@app.route('/home')
def home():
    if 'email' in session:
        return render_template('form.html')
    else:
        return redirect('/')

@app.route('/login_validation',methods=['POST'])
def login_validation():
    email=request.form.get('email')
```

main.py

```
    email = request.form.get('email')
    password = request.form.get('password')
    occupation = request.form.get('occupation')
    phone = request.form.get('phone')
    if mysql:
        print("Connection Successful!")
        cursor = mysql.connection.cursor()
        cursor.execute(
            """INSERT INTO `user_details` (`username`,`email`,`phone`,`occupation`,`password`) VALUES ('{}','{}','{}','{}','{}')""".form
        mysql.connection.commit()
        cursor.close()
    else:
        print("Connection Failed!")
    return "User Registered Successfully."


@app.route('/logout')
def logout():
    session.pop('email')
    return redirect('/')
@app.route('/form',methods=['POST'])
def form():
    print("HOME")
    return redirect('form.html')
@app.route('/predict', methods=['POST'])
def predict():

    age = request.form['age']
    gender = request.form['gender']
    tb = request.form['tb']
    dbi = request.form['dbi']
    ap = request.form['ap']
    aa1 = request.form['aa1']
    aa2 = request.form['aa2']
    tp = request.form['tp']
    a = request.form['a']
    agr = request.form['agr']
    if gender == "Male":
```


main.py

```
    aa1 = request.form['aa1']
    aa2 = request.form['aa2']
    tp = request.form['tp']
    a = request.form['a']
    agr = request.form['agr']
    if gender == "Male":
        gender = 1
    else:
        gender = 0
    data = [[float(age),
             float(gender),
             float(tb),
             float(dbi),
             float(ap),
             float(aa1),
             float(aa2),
             float(tp),
             float(a),
             float(agr)]]

    model = pickle.load(open('liver1.pkl', 'rb'))

    prediction = model.predict(data)
    if (prediction == 1):
        return render_template('noChance.html',
                               prediction='You don\'t have disease.')
    else:
        return render_template('chance.html',
                               prediction='You dead boy.')


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


```