#import the libraries import numpy as np import pandas as pd

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.model\_selection import train\_test\_split, StratifiedKFold, cross\_val\_score from sklearn.pipeline import make\_pipeline, Pipeline from sklearn.model\_selection import GridSearchCV

from sklearn.svm import SVC

from sklearn.naive\_bayes import MultinomialNB from sklearn.linear\_model import LogisticRegression from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.externals import joblib

from sklearn.metrics import make\_scorer, f1\_score, recall\_score, precision\_score from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score from sklearn.metrics import log\_loss

import warnings

warnings.simplefilter(action = 'ignore', category= FutureWarning)

from sklearn.ensemble import BaggingClassifier from sklearn.ensemble import AdaBoostClassifier import numpy as np

from flask import Flask,request,jsonify, render\_template import pickle

app=Flask( name ,template\_folder='template') app.\_static\_folder = 'static' model1=pickle.load(open('model1.pkl','rb')) model2=pickle.load(open('model2.pkl','rb')) @app.route('/home')

def homepage():

return render\_template('index.html') @app.route('/precautions')

def precautions():

return render\_template('precautions.html') @app.route('/advancedpage')

def advancedpage():

return render\_template('index.html') @app.route('/quick',methods=['POST']) def quick():

def bmi(height,weight):

bmi=int(weight)/((int(height)/100)\*\*2) return bmi

int\_features1 = [float(x) for x in request.form.values()]

age=int\_features1[1] cigs=int\_features1[3] height=int\_features1[8] weight=int\_features1[9] hrv=int\_features1[10] int\_features1.pop(8) int\_features1.pop(9) bmi=round(bmi(height,weight),2) int\_features1.insert(8,bmi)

if int(int\_features1[0])==1.0: sex="Male"

else:

sex="Female"

if int(int\_features1[2])==1.0: smoking="Yes"

else:

smoking="No"

if int(int\_features1[4])==1.0: stroke="Yes"

else:

stroke="No"

if int(int\_features1[5])==1.0: hyp="Yes"

else:

hyp="No"

if int(int\_features1[7])==1.0: dia="Yes"

else:

dia="No"

if int(int\_features1[6])==1.0: bpmeds="Yes"

else:

bpmeds="No"

final\_feature1=[np.array(int\_features1)] prediction1= model1.predict(final\_feature1) result=prediction1[0]

if result==0:

result="No need to worry"

else:

result="You are detected with heart problems. You need to consult

a doctor immediately"

return render\_template('quick\_report.html',prediction\_text1= result,gender=sex,age=age,smoking=smoking,cigs=cigs,stroke=stroke,hyp=hyp,dia=di a,bpmeds=bpmeds,bmi=bmi,hrv=hrv)

@app.route('/quickpage') def quickpage():

return render\_template('index1.html')

@app.route('/customersupport') def customersupport():

return render\_template('customercare.html') @app.route('/Doctorconsult')

def Doctorconsult():

return render\_template('Doctorconsult.html')

@app.route('/') def home():

return render\_template('Home.html')

@app.route('/advanced',methods=['POST']) def advanced():

int\_features2 = [int(x) for x in request.form.values()] final2\_feature=[np.array(int\_features2)] prediction2= model2.predict(final2\_feature) result=prediction2[0]

age=int\_features2[0] trestbps=int\_features2[3] chol=int\_features2[4] oldspeak=int\_features2[7] thalach=int\_features2[7] ca=int\_features2[10]

if int(int\_features2[1])==1: sex="Male"

else:

sex="Female"

if int(int\_features2[2])==1: cp="Typical angina"

elif int(int\_features2[2])==2:

cp="Atypical angina"

elif int(int\_features2[2])==3:

cp="Non-angina pain"

else:

cp="Asymtomatic"

if int(int\_features2[5])==1: fbs="Yes"

else:

fbs="No"

if int(int\_features2[6])==1:

Estes"

restecg="ST-T wave abnormality" elif int(int\_features2[6])==2:

restecg="showing probable or definite left ventricular hypertrophy by

else:

restecg="Normal"

if int(int\_features2[8])==1: exang="Yes"

else:

exang="No"

if int(int\_features2[9])==1: slope="upsloping"

elif int(int\_features2[9])==2: slope="flat"

else:

slope="downsloping"

if int(int\_features2[11])==3: thal="Normal"

elif int(int\_features2[11])==6: thal="Fixed defect"

else:

thal=" reversable defect"

if result==0:

result="No need to worry"

else:

result="You are detected with heart problems. You need to consult

a doctor immediately"

return render\_template('advance\_report.html',prediction\_text2= result,age=age,sex=sex,cp=cp,trestbps=trestbps,chol=chol,fbs=fbs,restecg=restecg,old peak=oldspeak,exang=exang,slope=slope,ca=ca,thal=thal)

if name ==" main ":

app.run(debug=True)

#read the csv dataset

data = pd.read\_csv("heart.csv", encoding='ANSI') data.columns

data.head()

#Total number of rows and columns data.shape

# Plot a line graph for Age V/s heart disease plt.subplots(figsize =(8,5))

classifiers = ['<=40', '41-50', '51-60','61 and Above']

heart\_disease = [13, 53, 64, 35] no\_heart\_disease =

[6, 23, 65, 44]

l1 = plt.plot(classifiers, heart\_disease , color='g', marker='o', linestyle ='dashed', markerfacecolor='y', markersize=10)

l2 = plt.plot(classifiers, no\_heart\_disease, color='r',marker='o', linestyle ='dashed', markerfacecolor='y', markersize=10 )

plt.xlabel('Age') plt.ylabel('Number of patients') plt.title('Age V/s Heart disease')

plt.legend((l1[0], l2[0]), ('heart\_disease', 'no\_heart\_disease')) plt.show()

# Plot a bar graph for Gender V/s target N = 2

ind = np.arange(N) width = 0.1

fig, ax = plt.subplots(figsize =(8,4))

heart\_disease = [93, 72]

rects1 = ax.bar(ind, heart\_disease, width, color='g') no\_heart\_disease = [114, 24]

rects2 = ax.bar(ind+width, no\_heart\_disease, width, color='y')

ax.set\_ylabel('Scores') ax.set\_title('Gender V/s target') ax.set\_xticks(ind) ax.set\_xticklabels(('Male','Female'))

ax.legend((rects1[0], rects2[0]), ('heart disease', 'no heart disease'))

plt.show()

#Pie charts for thal:Thalassemla # Having heart disease

labels= 'Normal', 'Fixed defect', 'Reversable defect' sizes=[6, 130, 28]

colors=['red', 'orange', 'green']

plt.pie(sizes, labels=labels, colors=colors, autopct='%.1f%%', shadow=True, startangle=140)

plt.axis('equal')

plt.title('Thalassemla blood disorder status of patients having heart disease') plt.show()

# Not having heart disease

labels= 'Normal', 'Fixed defect', 'Reversable defect' sizes=[12, 36, 89]

colors=['red', 'orange', 'green']

plt.pie(sizes, labels=labels, colors=colors, autopct='%.1f%%', shadow=True, startangle=140)

plt.axis('equal')

plt.title('Thalassemla blood disorder status of patients who do not have heart disease') plt.show()

## Feature selection

#get correlation of each feature in dataset

corrmat = data.corr() top\_corr\_features = corrmat.index plt.figure(figsize=(13,13))

#plot heat map g=sns.heatmap(data[top\_corr\_features].corr(),annot=True,cmap="RdYlGn")

data=data.drop(['sex', 'fbs', 'restecg', 'slope', 'chol', 'age', 'trestbps'], axis=1)

target=data['target']

data = data.drop(['target'],axis=1) data.head()

# We split the data into training and testing set:

x\_train, x\_test, y\_train, y\_test = train\_test\_split(data, target, test\_size=0.3, random\_state=10)

## Base Learners clfs = []

kfolds = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=1) np.random.seed(1)

#Support Vector Machine(SVM)

pipeline\_svm = make\_pipeline(SVC(probability=True, kernel="linear", class\_weight="balanced"))

grid\_svm = GridSearchCV(pipeline\_svm, param\_grid = {'svc C': [0.01, 0.1, 1]},

cv = kfolds, verbose=1,

n\_jobs=-1)

grid\_svm.fit(x\_train, y\_train) grid\_svm.score(x\_test, y\_test)

print("\nBest Model: %f using %s" % (grid\_svm.best\_score\_, grid\_svm.best\_params\_))

print('\n')

print('SVM LogLoss {score}'.format(score=log\_loss(y\_test, grid\_svm.predict\_proba(x\_test))))

clfs.append(grid\_svm)

# save best model to current working directory joblib.dump(grid\_svm, "heart\_disease.pkl")

# load from file and predict using the best configs found in the CV step model\_grid\_svm = joblib.load("heart\_disease.pkl" )

# get predictions from best model above y\_preds = model\_grid\_svm.predict(x\_test)

print('SVM accuracy score: ',accuracy\_score(y\_test, y\_preds)) print('\n')

import pylab as plt labels=[0,1]

cmx=confusion\_matrix(y\_test,y\_preds, labels) print(cmx)

fig = plt.figure()

ax = fig.add\_subplot(111) cax = ax.matshow(cmx)

plt.title('Confusion matrix of the classifier') fig.colorbar(cax)

ax.set\_xticklabels([''] + labels) ax.set\_yticklabels([''] + labels) plt.xlabel('Predicted') plt.ylabel('Actual')

plt.show()

print('\n')

print(classification\_report(y\_test, y\_preds))

# Multinomial Naive Bayes(NB) classifierNB=MultinomialNB() classifierNB.fit(x\_train,y\_train) classifierNB.score(x\_test, y\_test)

print('MultinomialNBLogLoss {score}'.format(score=log\_loss(y\_test, classifierNB.predict\_proba(x\_test))))

clfs.append(classifierNB)

# save best model to current working directory joblib.dump(classifierNB, "heart\_disease.pkl")

# load from file and predict using the best configs found in the CV step model\_classifierNB = joblib.load("heart\_disease.pkl" )

# get predictions from best model above y\_preds = model\_classifierNB.predict(x\_test)

print('MultinomialNB accuracy score: ',accuracy\_score(y\_test, y\_preds)) print('\n')

import pylab as plt labels=[0,1]

cmx=confusion\_matrix(y\_test,y\_preds, labels) print(cmx)

fig = plt.figure()

ax = fig.add\_subplot(111) cax = ax.matshow(cmx)

plt.title('Confusion matrix of the classifier') fig.colorbar(cax)

ax.set\_xticklabels([''] + labels) ax.set\_yticklabels([''] + labels) plt.xlabel('Predicted') plt.ylabel('Actual')

plt.show()

print('\n')

print(classification\_report(y\_test, y\_preds))

# Logistic Regression(LR) classifierLR=LogisticRegression()

classifierLR.fit(x\_train,y\_train) classifierLR.score(x\_test, y\_test)

print('LogisticRegressionLogLoss {score}'.format(score=log\_loss(y\_test, classifierLR.predict\_proba(x\_test))))

clfs.append(classifierLR)

# save best model to current working directory joblib.dump(classifierLR, "heart\_disease.pkl")

# load from file and predict using the best configs found in the CV step model\_classifierLR = joblib.load("heart\_disease.pkl" )

# get predictions from best model above y\_preds = model\_classifierLR.predict(x\_test)

print('Logistic Regression accuracy score: ',accuracy\_score(y\_test, y\_preds)) print('\n')

import pylab as plt labels=[0,1]

cmx=confusion\_matrix(y\_test,y\_preds, labels) print(cmx)

fig = plt.figure()

ax = fig.add\_subplot(111) cax = ax.matshow(cmx)

plt.title('Confusion matrix of the classifier') fig.colorbar(cax)

ax.set\_xticklabels([''] + labels) ax.set\_yticklabels([''] + labels) plt.xlabel('Predicted') plt.ylabel('Actual')

plt.show()

print('\n')

print(classification\_report(y\_test, y\_preds))

# Decision Tree (DT)

classifierDT=DecisionTreeClassifier(criterion="gini", random\_state=50, max\_depth=3, min\_samples\_leaf=5)

classifierDT.fit(x\_train,y\_train) classifierDT.score(x\_test, y\_test)

print('Decision Tree LogLoss {score}'.format(score=log\_loss(y\_test, classifierDT.predict\_proba(x\_test))))

clfs.append(classifierDT)

# save best model to current working directory joblib.dump(classifierDT, "heart\_disease.pkl")

# load from file and predict using the best configs found in the CV step model\_classifierDT = joblib.load("heart\_disease.pkl" )

# get predictions from best model above y\_preds = model\_classifierDT.predict(x\_test)

print('Decision Tree accuracy score: ',accuracy\_score(y\_test, y\_preds)) print('\n')

import pylab as plt labels=[0,1]

cmx=confusion\_matrix(y\_test,y\_preds, labels) print(cmx)

fig = plt.figure()

ax = fig.add\_subplot(111) cax = ax.matshow(cmx)

plt.title('Confusion matrix of the classifier') fig.colorbar(cax)

ax.set\_xticklabels([''] + labels) ax.set\_yticklabels([''] + labels) plt.xlabel('Predicted') plt.ylabel('Actual')

plt.show()

print('\n')

print(classification\_report(y\_test, y\_preds))

# Random Forest(RF) classifierRF=RandomForestClassifier() classifierRF.fit(x\_train,y\_train)

classifierRF.score(x\_test, y\_test)

print('RandomForestLogLoss {score}'.format(score=log\_loss(y\_test, classifierRF.predict\_proba(x\_test))))

clfs.append(classifierRF)

# save best model to current working directory joblib.dump(classifierRF, "heart\_disease.pkl")

# load from file and predict using the best configs found in the CV step model\_classifierRF = joblib.load("heart\_disease.pkl" )

# get predictions from best model above y\_preds = model\_classifierRF.predict(x\_test)

print('Random Forest accuracy score: ',accuracy\_score(y\_test, y\_preds)) print('\n')

import pylab as plt labels=[0,1]

cmx=confusion\_matrix(y\_test,y\_preds, labels) print(cmx)

fig = plt.figure()

ax = fig.add\_subplot(111) cax = ax.matshow(cmx)

plt.title('Confusion matrix of the classifier') fig.colorbar(cax)

ax.set\_xticklabels([''] + labels) ax.set\_yticklabels([''] + labels) plt.xlabel('Predicted') plt.ylabel('Actual')

plt.show()

print('\n')

print(classification\_report(y\_test, y\_preds))

print('\n')

print('Accuracy of svm: {}'.format(grid\_svm.score(x\_test, y\_test)))

print('Accuracy of naive bayes: {}'.format(classifierNB.score(x\_test, y\_test)))

print('Accuracy of logistic regression: {}'.format(classifierLR.score(x\_test, y\_test)))

print('Accuracy of decision tree: {}'.format(classifierDT.score(x\_test, y\_test)))

print('Accuracy of random forest: {}'.format(classifierRF.score(x\_test, y\_test)))

//#//

#import the libraries import numpy as np import pandas as pd

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.model\_selection import train\_test\_split, StratifiedKFold, cross\_val\_score from sklearn.pipeline import make\_pipeline, Pipeline from sklearn.model\_selection import GridSearchCV

from sklearn.svm import SVC

from sklearn.naive\_bayes import MultinomialNB from sklearn.linear\_model import LogisticRegression from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.externals import joblib

from sklearn.metrics import make\_scorer, f1\_score, recall\_score, precision\_score from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

from sklearn.metrics import log\_loss import warnings

warnings.simplefilter(action = 'ignore', category= FutureWarning) from sklearn.ensemble import BaggingClassifier

from sklearn.ensemble import AdaBoostClassifier

#read the csv dataset

data = pd.read\_csv("heart.csv", encoding='ANSI') data.columns

data.head()

#Total number of rows and columns data.shape

# Plot a line graph for Age V/s heart disease plt.subplots(figsize =(8,5))

classifiers = ['<=40', '41-50', '51-60','61 and Above']

heart\_disease = [13, 53, 64, 35] no\_heart\_disease =

[6, 23, 65, 44]

l1 = plt.plot(classifiers, heart\_disease , color='g', marker='o', linestyle ='dashed', markerfacecolor='y', markersize=10)

l2 = plt.plot(classifiers, no\_heart\_disease, color='r',marker='o', linestyle ='dashed', markerfacecolor='y', markersize=10 )

plt.xlabel('Age') plt.ylabel('Number of patients') plt.title('Age V/s Heart disease')

plt.legend((l1[0], l2[0]), ('heart\_disease', 'no\_heart\_disease')) plt.show()

# Plot a bar graph for Gender V/s target N = 2

ind = np.arange(N)

width = 0.1

fig, ax = plt.subplots(figsize =(8,4)) heart\_disease = [93, 72]

rects1 = ax.bar(ind, heart\_disease, width, color='g') no\_heart\_disease = [114, 24]

rects2 = ax.bar(ind+width, no\_heart\_disease, width, color='y')

ax.set\_ylabel('Scores') ax.set\_title('Gender V/s target') ax.set\_xticks(ind) ax.set\_xticklabels(('Male','Female'))

ax.legend((rects1[0], rects2[0]), ('heart disease', 'no heart disease'))

plt.show()

#Pie charts for thal:Thalassemla # Having heart disease

labels= 'Normal', 'Fixed defect', 'Reversable defect' sizes=[6, 130, 28]

colors=['red', 'orange', 'green']

plt.pie(sizes, labels=labels, colors=colors, autopct='%.1f%%', shadow=True, startangle=140)

plt.axis('equal')

plt.title('Thalassemla blood disorder status of patients having heart disease') plt.show()

# Not having heart disease

labels= 'Normal', 'Fixed defect', 'Reversable defect' sizes=[12, 36, 89]

colors=['red', 'orange', 'green']

plt.pie(sizes, labels=labels, colors=colors, autopct='%.1f%%', shadow=True, startangle=140)

plt.axis('equal')

plt.title('Thalassemla blood disorder status of patients who do not have heart disease') plt.show()

## Feature selection

#get correlation of each feature in dataset

corrmat = data.corr() top\_corr\_features = corrmat.index plt.figure(figsize=(13,13))

#plot heat map g=sns.heatmap(data[top\_corr\_features].corr(),annot=True,cmap="RdYlGn")

data=data.drop(['sex', 'fbs', 'restecg', 'slope', 'chol', 'age', 'trestbps'], axis=1)

target=data['target']

data = data.drop(['target'],axis=1) data.head()

# We split the data into training and testing set:

x\_train, x\_test, y\_train, y\_test = train\_test\_split(data, target, test\_size=0.3, random\_state=10)

## Base Learners clfs = []

kfolds = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=1) np.random.seed(1)

#Support Vector Machine(SVM)

pipeline\_svm = make\_pipeline(SVC(probability=True, kernel="linear", class\_weight="balanced"))

grid\_svm = GridSearchCV(pipeline\_svm, param\_grid = {'svc C': [0.01, 0.1, 1]},

cv = kfolds, verbose=1,

n\_jobs=-1)

grid\_svm.fit(x\_train, y\_train) grid\_svm.score(x\_test, y\_test)

print("\nBest Model: %f using %s" % (grid\_svm.best\_score\_, grid\_svm.best\_params\_))

print('\n')

print('SVM LogLoss {score}'.format(score=log\_loss(y\_test, grid\_svm.predict\_proba(x\_test))))

clfs.append(grid\_svm)

# save best model to current working directory joblib.dump(grid\_svm, "heart\_disease.pkl")

# load from file and predict using the best configs found in the CV step model\_grid\_svm = joblib.load("heart\_disease.pkl" )

# get predictions from best model above y\_preds = model\_grid\_svm.predict(x\_test)

print('SVM accuracy score: ',accuracy\_score(y\_test, y\_preds)) print('\n')

import pylab as plt labels=[0,1]

cmx=confusion\_matrix(y\_test,y\_preds, labels) print(cmx)

fig = plt.figure()

ax = fig.add\_subplot(111) cax = ax.matshow(cmx)

plt.title('Confusion matrix of the classifier') fig.colorbar(cax)

ax.set\_xticklabels([''] + labels) ax.set\_yticklabels([''] + labels) plt.xlabel('Predicted') plt.ylabel('Actual')

plt.show()

print('\n')

print(classification\_report(y\_test, y\_preds))

# Multinomial Naive Bayes(NB) classifierNB=MultinomialNB() classifierNB.fit(x\_train,y\_train) classifierNB.score(x\_test, y\_test)

print('MultinomialNBLogLoss {score}'.format(score=log\_loss(y\_test, classifierNB.predict\_proba(x\_test))))

clfs.append(classifierNB)

# save best model to current working directory joblib.dump(classifierNB, "heart\_disease.pkl")

# load from file and predict using the best configs found in the CV step model\_classifierNB = joblib.load("heart\_disease.pkl" )

# get predictions from best model above y\_preds = model\_classifierNB.predict(x\_test)

print('MultinomialNB accuracy score: ',accuracy\_score(y\_test, y\_preds)) print('\n')

import pylab as plt labels=[0,1]

cmx=confusion\_matrix(y\_test,y\_preds, labels) print(cmx)

fig = plt.figure()

ax = fig.add\_subplot(111) cax = ax.matshow(cmx)

plt.title('Confusion matrix of the classifier') fig.colorbar(cax)

ax.set\_xticklabels([''] + labels) ax.set\_yticklabels([''] + labels) plt.xlabel('Predicted') plt.ylabel('Actual')

plt.show()

print('\n')

print(classification\_report(y\_test, y\_preds))

# Logistic Regression(LR) classifierLR=LogisticRegression()

classifierLR.fit(x\_train,y\_train) classifierLR.score(x\_test, y\_test)

print('LogisticRegressionLogLoss {score}'.format(score=log\_loss(y\_test, classifierLR.predict\_proba(x\_test))))

clfs.append(classifierLR)

# save best model to current working directory joblib.dump(classifierLR, "heart\_disease.pkl")

# load from file and predict using the best configs found in the CV step model\_classifierLR = joblib.load("heart\_disease.pkl" )

# get predictions from best model above y\_preds = model\_classifierLR.predict(x\_test)

print('Logistic Regression accuracy score: ',accuracy\_score(y\_test, y\_preds)) print('\n')

import pylab as plt labels=[0,1]

cmx=confusion\_matrix(y\_test,y\_preds, labels) print(cmx)

fig = plt.figure()

ax = fig.add\_subplot(111) cax = ax.matshow(cmx)

plt.title('Confusion matrix of the classifier') fig.colorbar(cax)

ax.set\_xticklabels([''] + labels) ax.set\_yticklabels([''] + labels) plt.xlabel('Predicted') plt.ylabel('Actual')

plt.show()

print('\n')

print(classification\_report(y\_test, y\_preds))

# Decision Tree (DT)

classifierDT=DecisionTreeClassifier(criterion="gini", random\_state=50, max\_depth=3, min\_samples\_leaf=5)

classifierDT.fit(x\_train,y\_train) classifierDT.score(x\_test, y\_test)

print('Decision Tree LogLoss {score}'.format(score=log\_loss(y\_test, classifierDT.predict\_proba(x\_test))))

clfs.append(classifierDT)

# save best model to current working directory joblib.dump(classifierDT, "heart\_disease.pkl")

# load from file and predict using the best configs found in the CV step model\_classifierDT = joblib.load("heart\_disease.pkl" )

# get predictions from best model above y\_preds = model\_classifierDT.predict(x\_test)

print('Decision Tree accuracy score: ',accuracy\_score(y\_test, y\_preds)) print('\n')

import pylab as plt labels=[0,1]

cmx=confusion\_matrix(y\_test,y\_preds, labels) print(cmx)

fig = plt.figure()

ax = fig.add\_subplot(111) cax = ax.matshow(cmx)

plt.title('Confusion matrix of the classifier') fig.colorbar(cax)

ax.set\_xticklabels([''] + labels) ax.set\_yticklabels([''] + labels) plt.xlabel('Predicted') plt.ylabel('Actual')

plt.show()

print('\n')

print(classification\_report(y\_test, y\_preds))

# Random Forest(RF) classifierRF=RandomForestClassifier() classifierRF.fit(x\_train,y\_train) classifierRF.score(x\_test, y\_test)

print('RandomForestLogLoss {score}'.format(score=log\_loss(y\_test, classifierRF.predict\_proba(x\_test))))

clfs.append(classifierRF)

# save best model to current working directory joblib.dump(classifierRF, "heart\_disease.pkl")

# load from file and predict using the best configs found in the CV step model\_classifierRF = joblib.load("heart\_disease.pkl" )

# get predictions from best model above y\_preds = model\_classifierRF.predict(x\_test)

print('Random Forest accuracy score: ',accuracy\_score(y\_test, y\_preds)) print('\n')

import pylab as plt labels=[0,1]

cmx=confusion\_matrix(y\_test,y\_preds, labels) print(cmx)

fig = plt.figure()

ax = fig.add\_subplot(111) cax = ax.matshow(cmx)

plt.title('Confusion matrix of the classifier') fig.colorbar(cax)

ax.set\_xticklabels([''] + labels) ax.set\_yticklabels([''] + labels) plt.xlabel('Predicted') plt.ylabel('Actual')

plt.show() print('\n')

print(classification\_report(y\_test, y\_preds)) print('\n')

print('Accuracy of svm: {}'.format(grid\_svm.score(x\_test, y\_test))) print('Accuracy of naive bayes: {}'.format(classifierNB.score(x\_test, y\_test)))

print('Accuracy of logistic regression: {}'.format(classifierLR.score(x\_test, y\_test))) print('Accuracy of decision tree: {}'.format(classifierDT.score(x\_test, y\_test))) print('Accuracy of random forest: {}'.format(classifierRF.score(x\_test, y\_test)))

//#//

import numpy as np

from flask import Flask,request,jsonify, render\_template import pickle

app=Flask( name ,template\_folder='template') app.\_static\_folder = 'static' model1=pickle.load(open('model1.pkl','rb')) model2=pickle.load(open('model2.pkl','rb')) @app.route('/home')

def homepage():

return render\_template('index.html') @app.route('/precautions')

def precautions():

return render\_template('precautions.html') @app.route('/advancedpage')

def advancedpage():

return render\_template('index.html') @app.route('/quick',methods=['POST']) def quick():

def bmi(height,weight):

bmi=int(weight)/((int(height)/100)\*\*2) return bmi

int\_features1 = [float(x) for x in request.form.values()]

age=int\_features1[1] cigs=int\_features1[3] height=int\_features1[8] weight=int\_features1[9] hrv=int\_features1[10]

int\_features1.pop(8)

int\_features1.pop(9) bmi=round(bmi(height,weight),2) int\_features1.insert(8,bmi)

if int(int\_features1[0])==1.0: sex="Male"

else:

sex="Female"

if int(int\_features1[2])==1.0: smoking="Yes"

else:

smoking="No"

if int(int\_features1[4])==1.0: stroke="Yes"

else:

stroke="No"

if int(int\_features1[5])==1.0: hyp="Yes"

else:

hyp="No"

if int(int\_features1[7])==1.0: dia="Yes"

else:

dia="No"

if int(int\_features1[6])==1.0: bpmeds="Yes"

else:

bpmeds="No"

final\_feature1=[np.array(int\_features1)] prediction1= model1.predict(final\_feature1) result=prediction1[0]

if result==0:

result="No need to worry"

else:

result="You are detected with heart problems. You need to consult

a doctor immediately"

return render\_template('quick\_report.html',prediction\_text1= result,gender=sex,age=age,smoking=smoking,cigs=cigs,stroke=stroke,hyp=hyp,dia=di a,bpmeds=bpmeds,bmi=bmi,hrv=hrv)

@app.route('/quickpage') def quickpage():

return render\_template('index1.html')

@app.route('/customersupport') def customersupport():

return render\_template('customercare.html') @app.route('/Doctorconsult')

def Doctorconsult():

return render\_template('Doctorconsult.html') @app.route('/')

def home():

return render\_template('Home.html')

@app.route('/advanced',methods=['POST']) def advanced():

int\_features2 = [int(x) for x in request.form.values()] final2\_feature=[np.array(int\_features2)] prediction2= model2.predict(final2\_feature) result=prediction2[0]

age=int\_features2[0] trestbps=int\_features2[3] chol=int\_features2[4]

oldspeak=int\_features2[7]

thalach=int\_features2[7] ca=int\_features2[10]

if int(int\_features2[1])==1: sex="Male"

else:

sex="Female"

if int(int\_features2[2])==1: cp="Typical angina"

elif int(int\_features2[2])==2: cp="Atypical angina"

elif int(int\_features2[2])==3: cp="Non-angina pain"

else:

cp="Asymtomatic"

if int(int\_features2[5])==1: fbs="Yes"

else:

fbs="No"

Estes"

if int(int\_features2[6])==1:

restecg="ST-T wave abnormality" elif int(int\_features2[6])==2:

restecg="showing probable or definite left ventricular hypertrophy by

else:

restecg="Normal"

if int(int\_features2[8])==1: exang="Yes"

else:

exang="No"

if int(int\_features2[9])==1: slope="upsloping"

elif int(int\_features2[9])==2: slope="flat"

else:

slope="downsloping"

if int(int\_features2[11])==3: thal="Normal"

elif int(int\_features2[11])==6: thal="Fixed defect"

else:

thal=" reversable defect"

if result==0:

result="No need to worry"

else:

result="You are detected with heart problems. You need to consult

a doctor immediately"

return render\_template('advance\_report.html',prediction\_text2= result,age=age,sex=sex,cp=cp,trestbps=trestbps,chol=chol,fbs=fbs,restecg=restecg,old peak=oldspeak,exang=exang,slope=slope,ca=ca,thal=thal)

if name ==" main ": app.run(debug=True)