**CODE:**

**user/views.py:**

import gc

from django.shortcuts import render

from django.contrib import messages

# Create your views here.

from users.forms import UserRegistrationForm, HeartDataForm

from users.models import UserRegistrationModel, HeartDataModel

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import accuracy\_score

from django\_pandas.io import read\_frame

#%matplotlib inline

from sklearn.model\_selection import train\_test\_split

import os

#print(os.listdir())

import warnings

from django.core.paginator import Paginator, PageNotAnInteger, EmptyPage

def UserLogin(request):

return render(request, 'UserLogin.html', {})

def UserRegisterAction(request):

if request.method == 'POST':

form = UserRegistrationForm(request.POST)

if form.is\_valid():

print('Data is Valid')

form.save()

messages.success(request, 'You have been successfully registered')

# return HttpResponseRedirect('./CustLogin')

form = UserRegistrationForm()

return render(request, 'Register.html', {'form': form})

else:

print("Invalid form")

else:

form = UserRegistrationForm()

return render(request, 'Register.html', {'form': form})

def UserLoginCheck(request):

if request.method == "POST":

loginid = request.POST.get('loginname')

pswd = request.POST.get('pswd')

print("Login ID = ", loginid, ' Password = ', pswd)

try:

check = UserRegistrationModel.objects.get(loginid=loginid, password=pswd)

status = check.status

print('Status is = ', status)

if status == "activated":

request.session['id'] = check.id

request.session['loggeduser'] = check.name

request.session['loginid'] = loginid

request.session['email'] = check.email

print("User id At", check.id, status)

return render(request, 'users/UserHomePage.html', {})

else:

messages.success(request, 'Your Account Not at activated')

return render(request, 'UserLogin.html')

# return render(request, 'user/userpage.html',{})

except Exception as e:

print('Exception is ', str(e))

pass

messages.success(request, 'Invalid Login id and password')

return render(request, 'UserLogin.html', {})

def UserAddData(request):

if request.method == 'POST':

form = HeartDataForm(request.POST)

if form.is\_valid():

print('Data is Valid')

form.save()

messages.success(request, 'Data Added Successfull')

# return HttpResponseRedirect('./CustLogin')

form = HeartDataForm()

return render(request, 'users/UserAddData.html', {'form': form})

else:

print("Invalid form")

else:

form = HeartDataForm()

return render(request, 'users/UserAddData.html', {'form': form})

def UserDataView(request):

data\_list = HeartDataModel.objects.all()

page = request.GET.get('page', 1)

paginator = Paginator(data\_list, 10)

try:

users = paginator.page(page)

except PageNotAnInteger:

users = paginator.page(1)

except EmptyPage:

users = paginator.page(paginator.num\_pages)

return render(request, 'users/DataView\_list.html', {'users': users})

def UserMachineLearning(request):

#gc.collect()

dataset = HeartDataModel.objects.all()

dataset = read\_frame(dataset)

#dataset.fillna

print(dataset.head())

print(type(dataset))

print(dataset.shape)

print(dataset.head(5))

print(dataset.sample(5))

print(dataset.describe())

dataset.info()

info = ["age", "1: male, 0: female",

"chest pain type, 1: typical angina, 2: atypical angina, 3: non-anginal pain, 4: asymptomatic",

"resting blood pressure", " serum cholestoral in mg/dl", "fasting blood sugar > 120 mg/dl",

"resting electrocardiographic results (values 0,1,2)", " maximum heart rate achieved",

"exercise induced angina", "oldpeak = ST depression induced by exercise relative to rest",

"the slope of the peak exercise ST segment", "number of major vessels (0-3) colored by flourosopy",

"thal: 3 = normal; 6 = fixed defect; 7 = reversable defect"]

for i in range(len(info)):

print(dataset.columns[i] + ":\t\t\t" + info[i])

#X = dataset.drop(['target'], axis=1).values

#print("x",X)

dataset["target"].describe()

print(dataset["target"].unique())

print(dataset.corr()["target"].abs().sort\_values(ascending=False))

y = dataset["target"]

print("y",y)

sns.countplot(y)

plt.show()

print("Dataset Head",dataset.head(25))

target\_temp = dataset.target.value\_counts()

print("target Label Count=",target\_temp)

print("Percentage of patience without heart problems: " + str(round(target\_temp[0] \* 100 / 303, 2)))

print("Percentage of patience with heart problems: " + str(round(target\_temp[1] \* 100 / 303, 2)))

print(dataset["sex"].unique())

sns.barplot(dataset["sex"], y)

plt.show()

dataset["cp"].unique()

sns.barplot(dataset["cp"], y)

plt.show()

dataset["fbs"].describe()

dataset["fbs"].unique()

sns.barplot(dataset["fbs"], y)

plt.show()

dataset["restecg"].unique()

sns.barplot(dataset["restecg"], y)

plt.show()

dataset["exang"].unique()

sns.barplot(dataset["exang"], y)

plt.show()

dataset["slope"].unique()

sns.barplot(dataset["slope"], y)

plt.show()

dataset["ca"].unique()

sns.countplot(dataset["ca"])

plt.show()

sns.barplot(dataset["ca"], y)

plt.show()

dataset["thal"].unique()

sns.barplot(dataset["thal"], y)

plt.show()

sns.distplot(dataset["thal"])

plt.show()

from sklearn.model\_selection import train\_test\_split

predictors = dataset.drop("target", axis=1)

target = dataset["target"]

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(predictors, target, test\_size=0.20, random\_state=0)

X\_train.shape

print(X\_train.shape)

X\_test.shape

print(X\_test.shape)

Y\_train.shape

print(Y\_train.shape)

Y\_test.shape

print(Y\_test.shape)

# Linear regression

from sklearn.metrics import accuracy\_score

from sklearn.linear\_model import LogisticRegression

lr = LogisticRegression()

lr.fit(X\_train, Y\_train)

Y\_pred\_lr = lr.predict(X\_test)

Y\_pred\_lr.shape

print(Y\_pred\_lr.shape)

score\_lr = round(accuracy\_score(Y\_pred\_lr, Y\_test) \* 100, 2)

print("The accuracy score achieved using Linear regression is: " + str(score\_lr) + " %")

# Naive Bayes

from sklearn.naive\_bayes import GaussianNB

nb = GaussianNB()

nb.fit(X\_train, Y\_train)

Y\_pred\_nb = nb.predict(X\_test)

Y\_pred\_nb.shape

print(Y\_pred\_nb.shape)

score\_nb = round(accuracy\_score(Y\_pred\_nb, Y\_test) \* 100, 2)

print("The accuracy score achieved using Naive Bayes is: " + str(score\_nb) + " %")

# SVM

from sklearn import svm

sv = svm.SVC(kernel='linear')

sv.fit(X\_train, Y\_train)

Y\_pred\_svm = sv.predict(X\_test)

Y\_pred\_svm.shape

print(Y\_pred\_svm.shape)

score\_svm = round(accuracy\_score(Y\_pred\_svm, Y\_test) \* 100, 2)

print("The accuracy score achieved using Linear SVM is: " + str(score\_svm) + " %")

# K Nearest Neighbors

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=7)

knn.fit(X\_train, Y\_train)

Y\_pred\_knn = knn.predict(X\_test)

Y\_pred\_knn.shape

print(Y\_pred\_knn.shape)

score\_knn = round(accuracy\_score(Y\_pred\_knn, Y\_test) \* 100, 2)

print("The accuracy score achieved using KNN is: " + str(score\_knn) + " %")

# Decision Tree

from sklearn.tree import DecisionTreeClassifier

max\_accuracy = 0

for x in range(200):

dt = DecisionTreeClassifier(random\_state=x)

dt.fit(X\_train, Y\_train)

Y\_pred\_dt = dt.predict(X\_test)

current\_accuracy = round(accuracy\_score(Y\_pred\_dt, Y\_test) \* 100, 2)

if (current\_accuracy > max\_accuracy):

max\_accuracy = current\_accuracy

best\_x = x

# print(max\_accuracy)

# print(best\_x)

dt = DecisionTreeClassifier(random\_state=best\_x)

dt.fit(X\_train, Y\_train)

Y\_pred\_dt = dt.predict(X\_test)

print(Y\_pred\_dt.shape)

score\_dt = round(accuracy\_score(Y\_pred\_dt, Y\_test) \* 100, 2)

print("The accuracy score achieved using Decision Tree is: " + str(score\_dt) + " %")

# Neural Network

from keras.models import Sequential

from keras.layers import Dense

model = Sequential()

model.add(Dense(11, activation='relu', input\_dim=14))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# model.fit(X\_train,Y\_train,epochs=300)

model.fit(X\_train, Y\_train, epochs=300)

Y\_pred\_nn = model.predict(X\_test)

Y\_pred\_nn.shape

print(Y\_pred\_nn.shape)

rounded = [round(x[0]) for x in Y\_pred\_nn]

Y\_pred\_nn = rounded

score\_nn = round(accuracy\_score(Y\_pred\_nn, Y\_test) \* 100, 2)

print("The accuracy score achieved using Neural Network is: " + str(score\_nn) + " %")

scores = [score\_lr, score\_nb, score\_svm, score\_knn, score\_dt, score\_nn]

algorithms = ["LR", "Naive Bayes", "SVM", "K-Nearest Neighbors", "Decision Tree", "Neural Network"]

for i in range(len(algorithms)):

print("The accuracy score achieved using " + algorithms[i] + " is: " + str(scores[i]) + " %")

sns.set(rc={'figure.figsize': (15, 8)})

plt.xlabel("Algorithms")

plt.ylabel("Accuracy score")

sns.barplot(algorithms, scores)

plt.show()

dict = {

"score\_lr" :score\_lr,

"score\_nb" :score\_nb,

"score\_svm" :score\_svm,

"score\_knn" :score\_knn,

"score\_dt" :score\_dt,

"score\_nn" :score\_nn,

}

return render(request, 'users/Machinelearning.html', dict)

**admin/view.py:**

from django.shortcuts import render

from django.contrib import messages

# Create your views here.

from users.models import UserRegistrationModel, HeartDataModel

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import accuracy\_score

from django\_pandas.io import read\_frame

#%matplotlib inline

from sklearn.model\_selection import train\_test\_split

import os

#print(os.listdir())

import warnings

from django.core.paginator import Paginator, PageNotAnInteger, EmptyPage

def AdminLogin(request):

return render(request,'AdminLogin.html',{})

def AdminLoginCheck(request):

if request.method == 'POST':

usrid = request.POST.get('loginname')

pswd = request.POST.get('pswd')

print("User ID is = ", usrid)

if usrid == 'admin' and pswd == 'admin':

return render(request, 'admins/AdminHome.html')

else:

messages.success(request, 'Please Check Your Login Details')

return render(request, 'AdminLogin.html',{})

def RegisterUsersView(request):

data = UserRegistrationModel.objects.all()

return render(request,'admins/ViewRegisterUsers.html',{'data':data})

def ActivaUsers(request):

if request.method == 'GET':

id = request.GET.get('uid')

status = 'activated'

print("PID = ", id, status)

UserRegistrationModel.objects.filter(id=id).update(status=status)

data = UserRegistrationModel.objects.all()

return render(request,'admins/ViewRegisterUsers.html',{'data':data})

def adminML(request):

#gc.collect()

dataset = HeartDataModel.objects.all()

dataset = read\_frame(dataset)

#dataset.fillna

print(dataset.head())

print(type(dataset))

print(dataset.shape)

print(dataset.head(5))

print(dataset.sample(5))

print(dataset.describe())

dataset.info()

info = ["age", "1: male, 0: female",

"chest pain type, 1: typical angina, 2: atypical angina, 3: non-anginal pain, 4: asymptomatic",

"resting blood pressure", " serum cholestoral in mg/dl", "fasting blood sugar > 120 mg/dl",

"resting electrocardiographic results (values 0,1,2)", " maximum heart rate achieved",

"exercise induced angina", "oldpeak = ST depression induced by exercise relative to rest",

"the slope of the peak exercise ST segment", "number of major vessels (0-3) colored by flourosopy",

"thal: 3 = normal; 6 = fixed defect; 7 = reversable defect"]

for i in range(len(info)):

print(dataset.columns[i] + ":\t\t\t" + info[i])

#X = dataset.drop(['target'], axis=1).values

#print("x",X)

dataset["target"].describe()

print(dataset["target"].unique())

print(dataset.corr()["target"].abs().sort\_values(ascending=False))

y = dataset["target"]

print("y",y)

sns.countplot(y)

print("Dataset Head",dataset.head(25))

target\_temp = dataset.target.value\_counts()

print("target Label Count=",target\_temp)

print("Percentage of patience without heart problems: " + str(round(target\_temp[0] \* 100 / 303, 2)))

print("Percentage of patience with heart problems: " + str(round(target\_temp[1] \* 100 / 303, 2)))

print(dataset["sex"].unique())

sns.barplot(dataset["sex"], y)

dataset["cp"].unique()

sns.barplot(dataset["cp"], y)

dataset["fbs"].describe()

dataset["fbs"].unique()

sns.barplot(dataset["fbs"], y)

dataset["restecg"].unique()

sns.barplot(dataset["restecg"], y)

dataset["exang"].unique()

sns.barplot(dataset["exang"], y)

dataset["slope"].unique()

sns.barplot(dataset["slope"], y)

dataset["ca"].unique()

sns.countplot(dataset["ca"])

sns.barplot(dataset["ca"], y)

dataset["thal"].unique()

sns.barplot(dataset["thal"], y)

sns.distplot(dataset["thal"])

from sklearn.model\_selection import train\_test\_split

predictors = dataset.drop("target", axis=1)

target = dataset["target"]

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(predictors, target, test\_size=0.20, random\_state=0)

X\_train.shape

print(X\_train.shape)

X\_test.shape

print(X\_test.shape)

Y\_train.shape

print(Y\_train.shape)

Y\_test.shape

print(Y\_test.shape)

# Linear regression

from sklearn.metrics import accuracy\_score

from sklearn.linear\_model import LogisticRegression

lr = LogisticRegression()

lr.fit(X\_train, Y\_train)

Y\_pred\_lr = lr.predict(X\_test)

Y\_pred\_lr.shape

print(Y\_pred\_lr.shape)

score\_lr = round(accuracy\_score(Y\_pred\_lr, Y\_test) \* 100, 2)

print("The accuracy score achieved using Linear regression is: " + str(score\_lr) + " %")

# Naive Bayes

from sklearn.naive\_bayes import GaussianNB

nb = GaussianNB()

nb.fit(X\_train, Y\_train)

Y\_pred\_nb = nb.predict(X\_test)

Y\_pred\_nb.shape

print(Y\_pred\_nb.shape)

score\_nb = round(accuracy\_score(Y\_pred\_nb, Y\_test) \* 100, 2)

print("The accuracy score achieved using Naive Bayes is: " + str(score\_nb) + " %")

# SVM

from sklearn import svm

sv = svm.SVC(kernel='linear')

sv.fit(X\_train, Y\_train)

Y\_pred\_svm = sv.predict(X\_test)

Y\_pred\_svm.shape

print(Y\_pred\_svm.shape)

score\_svm = round(accuracy\_score(Y\_pred\_svm, Y\_test) \* 100, 2)

print("The accuracy score achieved using Linear SVM is: " + str(score\_svm) + " %")

# K Nearest Neighbors

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=7)

knn.fit(X\_train, Y\_train)

Y\_pred\_knn = knn.predict(X\_test)

Y\_pred\_knn.shape

print(Y\_pred\_knn.shape)

score\_knn = round(accuracy\_score(Y\_pred\_knn, Y\_test) \* 100, 2)

print("The accuracy score achieved using KNN is: " + str(score\_knn) + " %")

# Decision Tree

from sklearn.tree import DecisionTreeClassifier

max\_accuracy = 0

for x in range(200):

dt = DecisionTreeClassifier(random\_state=x)

dt.fit(X\_train, Y\_train)

Y\_pred\_dt = dt.predict(X\_test)

current\_accuracy = round(accuracy\_score(Y\_pred\_dt, Y\_test) \* 100, 2)

if (current\_accuracy > max\_accuracy):

max\_accuracy = current\_accuracy

best\_x = x

# print(max\_accuracy)

# print(best\_x)

dt = DecisionTreeClassifier(random\_state=best\_x)

dt.fit(X\_train, Y\_train)

Y\_pred\_dt = dt.predict(X\_test)

print(Y\_pred\_dt.shape)

score\_dt = round(accuracy\_score(Y\_pred\_dt, Y\_test) \* 100, 2)

print("The accuracy score achieved using Decision Tree is: " + str(score\_dt) + " %")

# Neural Network

from keras.models import Sequential

from keras.layers import Dense

model = Sequential()

model.add(Dense(11, activation='relu', input\_dim=14))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# model.fit(X\_train,Y\_train,epochs=300)

model.fit(X\_train, Y\_train, epochs=300)

Y\_pred\_nn = model.predict(X\_test)

Y\_pred\_nn.shape

print(Y\_pred\_nn.shape)

rounded = [round(x[0]) for x in Y\_pred\_nn]

Y\_pred\_nn = rounded

score\_nn = round(accuracy\_score(Y\_pred\_nn, Y\_test) \* 100, 2)

print("The accuracy score achieved using Neural Network is: " + str(score\_nn) + " %")

scores = [score\_lr, score\_nb, score\_svm, score\_knn, score\_dt, score\_nn]

algorithms = ["LR", "Naive Bayes", "SVM", "K-Nearest Neighbors", "Decision Tree", "Neural Network"]

for i in range(len(algorithms)):

print("The accuracy score achieved using " + algorithms[i] + " is: " + str(scores[i]) + " %")

sns.set(rc={'figure.figsize': (15, 8)})

plt.xlabel("Algorithms")

plt.ylabel("Accuracy score")

sns.barplot(algorithms, scores)

plt.show()

return render(request, 'admins/AdminHome.html', )

**Urls.py**

"""Heart URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:

https://docs.djangoproject.com/en/2.2/topics/http/urls/

Examples:

Function views

1. Add an import: from my\_app import views

2. Add a URL to urlpatterns: path('', views.home, name='home')

Class-based views

1. Add an import: from other\_app.views import Home

2. Add a URL to urlpatterns: path('', Home.as\_view(), name='home')

Including another URLconf

1. Import the include() function: from django.urls import include, path

2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))

"""

from django.contrib import admin

from django.urls import path

from Heart import views as view

from users import views as users

from admins import views as admins

urlpatterns = [

path('admin/', admin.site.urls),

path('', view.index, name='index'),

path('logout/', view.logout, name='logout'),

path('UserLogin/', users.UserLogin, name='UserLogin'),

path('UserRegisterAction/', users.UserRegisterAction, name='UserRegisterAction'),

path('UserLoginCheck/', users.UserLoginCheck, name='UserLoginCheck'),

path('UserAddData/', users.UserAddData, name='UserAddData'),

path('UserDataView/', users.UserDataView, name='UserDataView'),

path('UserMachineLearning/', users.UserMachineLearning, name='UserMachineLearning'),

path('AdminLogin/', admins.AdminLogin, name='AdminLogin'),

path('AdminLoginCheck/', admins.AdminLoginCheck, name='AdminLoginCheck'),

path('RegisterUsersView/', admins.RegisterUsersView, name='RegisterUsersView'),

path('ActivaUsers/', admins.ActivaUsers, name='ActivaUsers'),

path('adminML/', admins.adminML, name='adminML')

]

**Model.py**

from django.db import models

# Create your models here.

class UserRegistrationModel(models.Model):

name = models.CharField(max\_length=100)

loginid = models.CharField(unique=True,max\_length=100)

password = models.CharField(max\_length=100)

mobile = models.CharField(max\_length=100)

email = models.CharField(max\_length=100)

locality = models.CharField(max\_length=100)

address = models.CharField(max\_length=1000)

city = models.CharField(max\_length=100)

state = models.CharField(max\_length=100)

status = models.CharField(max\_length=100)

def \_\_str\_\_(self):

return self.loginid

class Meta:

db\_table='Users'

class HeartDataModel(models.Model):

age = models.IntegerField()

sex = models.IntegerField()

cp = models.IntegerField()

trestbps = models.IntegerField()

chol = models.IntegerField()

fbs = models.IntegerField()

restecg = models.IntegerField()

thalach = models.IntegerField()

exang = models.IntegerField()

oldpeak = models.FloatField()

slope = models.IntegerField()

ca = models.IntegerField()

thal = models.IntegerField()

target = models.IntegerField()

def \_\_str\_\_(self):

return self.id

class Meta:

db\_table = 'HeartDatabase'