**SAMPLE CODE**

**User side views.py**

# Create your views here.

from django.shortcuts import render,HttpResponse

from django.contrib import messages

from .forms import UserRegistrationForm

from .models import UserRegistrationModel

from django.conf import settings

# Create your views here.

def UserRegisterActions(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')

form = UserRegistrationForm()

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

else:

messages.success(request, 'Email or Mobile Already Existed')

print("Invalid form")

else:

form = UserRegistrationForm()

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

def UserLoginCheck(request):

if request.method == "POST":

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

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')

except Exception as e:

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

pass

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

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

def UserHome(request):

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

def DatasetView(request):

path = settings.MEDIA\_ROOT+"//"+'heart\_obecity.csv'

import pandas as pd

df = pd.read\_csv(path)

df =df.to\_html

return render(request, 'users/viewdataset.html', {'data': df})

def user\_cart\_analysis(request):

from .utility.ProcessCart import start\_process\_cart

rslt\_dict = start\_process\_cart()

return render(request, "users/cartresults.html", rslt\_dict)

def user\_gbdt\_analysis(request):

from.utility.ProcessCart import start\_process\_gbdt

rslt\_dict = start\_process\_gbdt()

return render(request, "users/gbdtresults.html", rslt\_dict)

def user\_predictions(request):

if request.method == 'POST':

age = int(request.POST.get('age'))

anaemia = int(request.POST.get('anaemia'))

creatinine\_phosphokinase = int(request.POST.get('creatinine\_phosphokinase'))

diabetes = int(request.POST.get('diabetes'))

ejection\_fraction = float(request.POST.get('ejection\_fraction'))

high\_blood\_pressure = float(request.POST.get('high\_blood\_pressure'))

platelets = int(request.POST.get('platelets'))

serum\_creatinine = float(request.POST.get('serum\_creatinine'))

serum\_sodium = int(request.POST.get('serum\_sodium'))

sex = int(request.POST.get('sex'))

smoking = int(request.POST.get('smoking'))

time = int(request.POST.get('time'))

test\_data = [age, anaemia, creatinine\_phosphokinase, diabetes, ejection\_fraction, high\_blood\_pressure, platelets, serum\_creatinine, serum\_sodium, sex, smoking, time]

from .utility.predections import test\_user\_data

test\_pred = test\_user\_data(test\_data)

if test\_pred[0] == 0:

rslt = False

else:

rslt = True

return render(request, "users/testform.html", {"test\_data": test\_data, "result": rslt})

else:

return render(request, "users/testform.html", {})

Models.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(unique=True, max\_length=100)

email = models.CharField(unique=True, 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 = 'UserRegistrations'

forms.py

from django import forms

from .models import UserRegistrationModel

class UserRegistrationForm(forms.ModelForm):

name = forms.CharField(widget=forms.TextInput(attrs={'pattern': '[a-zA-Z]+'}), required=True, max\_length=100)

loginid = forms.CharField(widget=forms.TextInput(attrs={'pattern': '[a-zA-Z]+'}), required=True, max\_length=100)

password = forms.CharField(widget=forms.PasswordInput(attrs={'pattern': '(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z]).{8,}',

'title': 'Must contain at least one number and one uppercase and lowercase letter, and at least 8 or more characters'}),

required=True, max\_length=100)

mobile = forms.CharField(widget=forms.TextInput(attrs={'pattern': '[56789][0-9]{9}'}), required=True,

max\_length=100)

email = forms.CharField(widget=forms.TextInput(attrs={'pattern': '[a-z0-9.\_%+-]+@[a-z0-9.-]+\.[a-z]{2,}$'}),

required=True, max\_length=100)

locality = forms.CharField(widget=forms.TextInput(), required=True, max\_length=100)

address = forms.CharField(widget=forms.Textarea(attrs={'rows': 4, 'cols': 22}), required=True, max\_length=250)

city = forms.CharField(widget=forms.TextInput(

attrs={'autocomplete': 'off', 'pattern': '[A-Za-z ]+', 'title': 'Enter Characters Only '}), required=True,

max\_length=100)

state = forms.CharField(widget=forms.TextInput(

attrs={'autocomplete': 'off', 'pattern': '[A-Za-z ]+', 'title': 'Enter Characters Only '}), required=True,

max\_length=100)

status = forms.CharField(widget=forms.HiddenInput(), initial='waiting', max\_length=100)

class Meta():

model = UserRegistrationModel

fields = '\_\_all\_\_'

Predictions.py

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

from django.conf import settings

import pandas as pd

from sklearn import tree

path = settings.MEDIA\_ROOT + "//" + "heart\_obecity.csv"

df = pd.read\_csv(path)

X = df.iloc[:, :-1].values # indipendent variable

y = df.iloc[:, -1].values # Dependent variable

def fitting(X, y, criterion, splitter, mdepth, clweight, minleaf):

# Create training and testing samples

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# Fit the model

model = tree.DecisionTreeClassifier(criterion=criterion,

splitter=splitter,

max\_depth=mdepth,

class\_weight=clweight,

min\_samples\_leaf=minleaf,

random\_state=0,

)

clf = model.fit(X\_train, y\_train)

return X\_train, X\_test, y\_train, y\_test, clf

def test\_user\_data(data):

X\_train, X\_test, y\_train, y\_test, clf = fitting(X, y, 'gini', 'best', mdepth=3, clweight=None, minleaf=1000)

from sklearn.ensemble import GradientBoostingClassifier

model = GradientBoostingClassifier()

model.fit(X\_train, y\_train)

y\_pred = model.predict([data])

return y\_pred

Predicitions

import pandas as pd

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

from django.conf import settings

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

import pandas as pd # for data manipulation

import numpy as np # for data manipulation

from sklearn.model\_selection import train\_test\_split # for splitting the data into train and test samples

from sklearn.metrics import classification\_report # for model evaluation metrics

from sklearn import tree # for decision tree models

import plotly.express as px # for data visualization

import plotly.graph\_objects as go # for data visualization

import graphviz # for plotting decision tree graphs

path = settings.MEDIA\_ROOT + "//" + "heart\_obecity.csv"

df = pd.read\_csv(path)

X = df.iloc[:, :-1].values # indipendent variable

y = df.iloc[:, -1].values # Dependent variable

# X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.80, random\_state=0)

def fitting(X, y, criterion, splitter, mdepth, clweight, minleaf):

# Create training and testing samples

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# Fit the model

model = tree.DecisionTreeClassifier(criterion=criterion,

splitter=splitter,

max\_depth=mdepth,

class\_weight=clweight,

min\_samples\_leaf=minleaf,

random\_state=0,

)

clf = model.fit(X\_train, y\_train)

# Predict class labels on training data

pred\_labels\_tr = model.predict(X\_train)

# Predict class labels on a test data

pred\_labels\_te = model.predict(X\_test)

# Tree summary and model evaluation metrics

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Tree Summary \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print('Classes: ', clf.classes\_)

print('Tree Depth: ', clf.tree\_.max\_depth)

print('No. of leaves: ', clf.tree\_.n\_leaves)

print('No. of features: ', clf.n\_features\_)

print('--------------------------------------------------------')

print("")

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Evaluation on Test Data \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

score\_te = model.score(X\_test, y\_test)

print('Accuracy Score: ', score\_te)

# Look at classification report to evaluate the model

print(classification\_report(y\_test, pred\_labels\_te))

print('--------------------------------------------------------')

print("")

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Evaluation on Training Data \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

score\_tr = model.score(X\_train, y\_train)

print('Accuracy Score: ', score\_tr)

# Look at classification report to evaluate the model

print(classification\_report(y\_train, pred\_labels\_tr))

print('--------------------------------------------------------')

# Return relevant data for chart plotting

return X\_train, X\_test, y\_train, y\_test, clf

def start\_process\_cart():

X\_train, X\_test, y\_train, y\_test, clf = fitting(X, y, 'gini', 'best', mdepth=3, clweight=None, minleaf=1000)

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

from sklearn import metrics

accuracy = metrics.accuracy\_score(y\_test, y\_pred)

print("Accuracy :", accuracy)

from sklearn.metrics import confusion\_matrix

tn, fp, fn, tp = confusion\_matrix(y\_test, y\_pred).ravel()

sensitivity = tp / (tp + fn)

print("Sensitivity: ", sensitivity)

specificity = tn / (tn + fp)

print("Specificity: ", specificity)

precision = tp/(tp+fp)

print("Precisions: ",precision)

f1\_score = metrics.f1\_score(y\_test, y\_pred)

print("F1 Score: ",f1\_score)

roc\_auc = metrics.roc\_auc\_score(y\_test, y\_pred)

print("Roc Auc Curve:", roc\_auc)

rslt\_dict = {

"accuracy": accuracy,

"sensitivity": sensitivity,

"specificity": specificity,

"precisions": precision,

"f1\_score": f1\_score,

"roc\_auc": roc\_auc

}

return rslt\_dict

def start\_process\_gbdt():

X\_train, X\_test, y\_train, y\_test, clf = fitting(X, y, 'gini', 'best', mdepth=3, clweight=None, minleaf=1000)

from sklearn.ensemble import GradientBoostingClassifier

model = GradientBoostingClassifier()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

from sklearn import metrics

accuracy = metrics.accuracy\_score(y\_test, y\_pred)

print("Accuracy :", accuracy)

from sklearn.metrics import confusion\_matrix

tn, fp, fn, tp = confusion\_matrix(y\_test, y\_pred).ravel()

sensitivity = tp / (tp + fn)

print("Sensitivity: ", sensitivity)

specificity = tn / (tn + fp)

print("Specificity: ", specificity)

precision = tp/(tp+fp)

print("Precisions: ",precision)

f1\_score = metrics.f1\_score(y\_test, y\_pred)

print("F1 Score: ",f1\_score)

roc\_auc = metrics.roc\_auc\_score(y\_test, y\_pred)

print("Roc Auc Curve:", roc\_auc)

rslt\_dict = {

"accuracy": accuracy,

"sensitivity": sensitivity,

"specificity": specificity,

"precisions": precision,

"f1\_score": f1\_score,

"roc\_auc": roc\_auc

}

return rslt\_dict

Base.html

{% load static %}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Obesity on Autonomic Nervous System</title>

<meta name="description" content="Free Bootstrap Theme by BootstrapMade.com">

<meta name="keywords" content="free website templates, free bootstrap themes, free template, free bootstrap, free website template">

<link rel="stylesheet" type="text/css" href="https://fonts.googleapis.com/css?family=Open+Sans|Raleway|Candal">

<link rel="stylesheet" type="text/css" href="{% static 'css/font-awesome.min.css' %]">

<link rel="stylesheet" type="text/css" href="{% static 'css/bootstrap.min.css' %}">

<link rel="stylesheet" type="text/css" href="{% static 'css/style.css' %}">

</head>

<body id="myPage" data-spy="scroll" data-target=".navbar" data-offset="60">

<section id="banner" class="banner">

<div class="bg-color">

<nav class="navbar navbar-default navbar-fixed-top">

<div class="container">

<div class="col-md-12">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#myNavbar">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

</div>

Machine Learning Obesity on Autonomic Nervous System

<div class="collapse navbar-collapse navbar-right" id="myNavbar">

<ul class="nav navbar-nav">

<li class=""><a href="{% url 'index' %}">Home</a></li>

<li class=""><a href="{% url 'UserLogin' %}">User</a></li>

<li class=""><a href="{% url 'AdminLogin' %}">Admin</a></li>

<li class=""><a href="{% url 'UserRegister' %}">Register</a></li>

</ul>

</div>

</div>

</div>

</nav>

<div class="container">

<div class="row">

<div class="banner-info">

<div class="banner-logo text-center">

</div>

<div class="banner-text text-center">

<h1 class="white">Machine Learning Approach to Study the Impact of

Obesity on Autonomic Nervous System using Heart

Rate Variability Features</h1>

{%block contents%}

<!-- <h3 style="color:white"> helloo</h3>-->

{%endblock%}

<!-- <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod <br>tempor incididunt ut labore et dolore magna aliqua.</p>-->

<!-- <a href="#contact" class="btn btn-appoint">Make an Appointment.</a>-->

</div>

<div class="overlay-detail text-center">

<a href="#service"><i class="fa fa-angle-down"></i></a>

</div>

</div>

</div>

</div>

</div>

</section>

<footer id="footer">

<div class="footer-line">

<div class="container">

<div class="row">

<div class="col-md-12 text-center">

© Copyright Alex Corporation. All Rights Reserved

<div class="credits">

<!--

All the links in the footer should remain intact.

You can delete the links only if you purchased the pro version.

Licensing information: https://bootstrapmade.com/license/

Purchase the pro version with working PHP/AJAX contact form: https://bootstrapmade.com/buy/?theme=Medilab

-->

<!-- Designed by <a href="https://bootstrapmade.com/">BootstrapMade.com</a>-->

</div>

</div>

</div>

</div>

</div>

</footer>

<!--/ footer-->

<script src="{% static 'js/jquery.min.js' %}"></script>

<script src="{% static 'js/jquery.easing.min.js' %}"></script>

<script src="{% static 'js/bootstrap.min.js' %}"></script>

<script src="{% static 'js/custom.js' %}"></script>

<script src="{% static 'contactform/contactform.js' %}"></script>

</body>

</html>