**SAMPLE CODE**

**User side views**

from django.shortcuts import render

# Create your views here.

from django.shortcuts import render, HttpResponse

from django.contrib import messages

from .forms import UserRegistrationForm

from .models import UserRegistrationModel

# 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 user\_view\_data(request):

from django.conf import settings

import os

import pandas as pd

path = os.path.join(settings.MEDIA\_ROOT, 'diabetes.csv')

df = pd.read\_csv(path)

df = df.to\_html

return render(request, "users/dataset\_view.html", {"data": df})

def user\_supervised(request):

from .supervised import DiabetesSupervised

dt\_accuracy = DiabetesSupervised.decisionTree\_calc()

ada\_accuracy = DiabetesSupervised.adaBoost\_calc()

xg\_accuracy = DiabetesSupervised.xgBoost\_calc()

svm\_accuracy = DiabetesSupervised.svm\_classifiers()

result\_dict = {"dt": dt\_accuracy, "ada": ada\_accuracy, "xg": xg\_accuracy, "svm": svm\_accuracy}

return render(request, "users/supervised\_results.html", result\_dict)

def user\_unsupervised(request):

from .unsupervised import DIabetesUnsupervised

result = DIabetesUnsupervised.calc\_kmeans\_cluster()

print(result)

return render(request, "users/unsupervised\_results.html", result)

def user\_test\_model(request):

if request.method == 'POST':

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

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

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

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

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

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

from .supervised import DiabetesSupervised

msg, risk = DiabetesSupervised.diabetes\_risk\_prediction(glucose, bp, skinthickness, insulin, bmi, age)

return render(request, "users/test\_model\_form.html", {'msg': msg, 'risk': risk})

else:

return render(request, "users/test\_model\_form.html", {})

supervised.py

import os

import numpy as np

import pandas as pd

from django.conf import settings

path = os.path.join(settings.MEDIA\_ROOT, 'diabetes.csv')

diabetes = pd.read\_csv(path)

print(diabetes.shape)

print(diabetes.head())

diabetes.info()

import matplotlib.pyplot as plt

# Making a list of col names:

cols = diabetes.columns.tolist()

fig = plt.figure(figsize=(14, 14))

for position, col in enumerate(cols):

ax = fig.add\_subplot(3, 3, position + 1)

ax.hist(diabetes[col], bins=30)

plt.title(col)

# plt.show()

# Visualizing distribution of Data based on Diabetes outcome

# Splitting dataframe

with\_diabetes = diabetes[diabetes['Outcome'] == 1]

without\_diabetes = diabetes[diabetes['Outcome'] == 0]

fig = plt.figure(figsize=(14, 14))

for position, col in enumerate(cols):

ax = fig.add\_subplot(3, 3, position + 1)

ax.hist(with\_diabetes[col], histtype='step', label='With Diabetes', bins=20)

ax.hist(without\_diabetes[col], histtype='step', label='No Diabetes', bins=20)

ax.legend(loc=0)

plt.title(col)

# plt.show()

diabetes = diabetes.drop\_duplicates(keep='first')

diabetes.shape

# Handling incomplete Data

diabetes = diabetes.drop(diabetes[diabetes['BMI'] == 0].index)

diabetes = diabetes.drop(diabetes[diabetes['BloodPressure'] == 0].index)

diabetes = diabetes.drop(diabetes[diabetes['Insulin'] == 0].index)

diabetes = diabetes.drop(diabetes[diabetes['Glucose'] == 0].index)

diabetes = diabetes.drop(diabetes[diabetes['SkinThickness'] == 0].index)

diabetes.shape

# Feature Selection

features = cols.copy()

features.remove('Outcome')

features.remove('DiabetesPedigreeFunction')

features.remove('Pregnancies')

print(features)

import warnings

warnings.filterwarnings("ignore")

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import KFold

# Instantiate the model

weight = {

1: 1.05,

0: 1

}

log = LogisticRegression(class\_weight=weight)

kf = KFold(n\_splits=6)

score = cross\_val\_score(log, diabetes[features], diabetes['Outcome'], cv=kf, scoring='accuracy')

print(score)

print("The mean accuracy is:", score.mean())

weight = {

1: 1.05,

0: 1

}

log\_model = LogisticRegression(class\_weight=weight)

log\_model.fit(diabetes[features], diabetes['Outcome'])

log\_model.predict\_proba(np.array(diabetes[features].iloc[0]).reshape(1, -1))

print(diabetes[features].iloc[0])

testing1 = [120, 10, 100, 190, 50, 60]

features

log\_model.predict\_proba(np.array(testing1).reshape(1, -1))[0, 1]

print(features)

def decisionTree\_calc():

from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(class\_weight=weight)

kf = KFold(n\_splits=6)

score = cross\_val\_score(model, diabetes[features], diabetes['Outcome'], cv=kf, scoring='accuracy')

return score

def adaBoost\_calc():

from sklearn.ensemble import AdaBoostClassifier

model = AdaBoostClassifier(learning\_rate=1.0)

kf = KFold(n\_splits=6)

score = cross\_val\_score(model, diabetes[features], diabetes['Outcome'], cv=kf, scoring='accuracy')

return score

def xgBoost\_calc():

from xgboost import XGBClassifier

model = XGBClassifier(learning\_rate=1)

kf = KFold(n\_splits=6)

score = cross\_val\_score(model, diabetes[features], diabetes['Outcome'], cv=kf, scoring='accuracy')

return score

def svm\_classifiers():

from sklearn.svm import SVC

model = SVC(class\_weight=weight)

kf = KFold(n\_splits=6)

score = cross\_val\_score(model, diabetes[features], diabetes['Outcome'], cv=kf, scoring='accuracy')

return score

def diabetes\_risk\_prediction(glucose, bp, skinthickness, insulin, bmi, age):

indicator\_list = [glucose, bp, skinthickness, insulin, bmi, age]

predictions = log\_model.predict\_proba(np.array(indicator\_list).reshape(1, -1))

risk = predictions[0, 1]

print("-" \* len("Health Indicator Analysis"))

print("Health Indicator Analysis")

print("-" \* len("Health Indicator Analysis"))

msg = ''

if risk < 0.3:

msg = "You are probably in good health, keep it up."

# print("-"\*len("You are probably in good health, keep it up"))

elif risk > 0.7:

msg = "See a doctor as soon as you can and listen to their recommendations. You might be on the way to " \

"developing diabetes if you don't change your lifestyle. "

elif risk > 0.9:

msg = "Go to a hospital right away. Odds are high you have diabetes."

else:

msg = "You should be alright for the most part, but take care not to let your health slip."

rsk = "Your Diabetes Risk Index is {:.2f}/50.".format(risk \* 0.5 \* 100)

return msg, rsk

def diabetes\_risk\_prediction2(glucose, bp, skinthickness, insulin, bmi, age):

weight = {

1: 1.05,

0: 1}

log\_model2 = LogisticRegression(class\_weight=weight)

log\_model2.fit(diabetes[features], diabetes['Outcome'])

indicator\_list = [glucose, bp, skinthickness, insulin, bmi, age]

predictions = log\_model2.predict\_proba(np.array(indicator\_list).reshape(1, -1))

risk = predictions[0, 1]

return risk

# diabetes\_risk\_prediction(90, 80, 30, 120, 60, 50)

# diabetes\_risk\_prediction2(100, 80, 30, 120, 30, 45) # Test array.

Unsupervised.py

import os

import pandas as pd

from django.conf import settings

path = os.path.join(settings.MEDIA\_ROOT, 'diabetes.csv')

data = pd.read\_csv(path)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

dataset\_new = data

# Replacing zero values with NaN

dataset\_new[["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]] = dataset\_new[

["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]].replace(0, np.NaN)

# Count of NaN

dataset\_new.isnull().sum()

# Replacing NaN with mean values

dataset\_new["Glucose"].fillna(dataset\_new["Glucose"].mean(), inplace=True)

dataset\_new["BloodPressure"].fillna(dataset\_new["BloodPressure"].mean(), inplace=True)

dataset\_new["SkinThickness"].fillna(dataset\_new["SkinThickness"].mean(), inplace=True)

dataset\_new["Insulin"].fillna(dataset\_new["Insulin"].mean(), inplace=True)

dataset\_new["BMI"].fillna(dataset\_new["BMI"].mean(), inplace=True)

# Feature scaling using MinMaxScaler

from sklearn.preprocessing import MinMaxScaler

sc = MinMaxScaler(feature\_range=(0, 1))

dataset\_scaled = sc.fit\_transform(dataset\_new)

data1 = pd.DataFrame(dataset\_scaled)

data1

# Heatmap

sns.heatmap(data1.corr(), annot=True)

plt.show()

# Detailed distribution of the features in the dataset

sns.pairplot(data=data, hue='Outcome')

plt.show()

# Selecting features - [Glucose, Insulin, BMI]

X = data1.iloc[:, [1, 4, 5]].values

Y = data1.iloc[:, 8].values

# Splitting X and Y

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

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.20, random\_state=42,

stratify=dataset\_new['Outcome'])

# Checking dimensions

print("X\_train shape:", X\_train.shape)

print("X\_test shape:", X\_test.shape)

print("Y\_train shape:", Y\_train.shape)

print("Y\_test shape:", Y\_test.shape)

def calc\_kmeans\_cluster():

import sklearn

from sklearn.cluster import KMeans

KMeans\_Clustering = KMeans(n\_clusters=2, random\_state=0)

KMeans\_Clustering.fit(X\_train)

print(KMeans\_Clustering.cluster\_centers\_)

# prediction using kmeans and accuracy

kpred = KMeans\_Clustering.predict(X\_test)

from sklearn.metrics import cluster

amis = cluster.adjusted\_mutual\_info\_score(Y\_test, kpred)

adrs = cluster.adjusted\_rand\_score(Y\_test, kpred)

complete\_score = cluster.completeness\_score(Y\_test, kpred)

norm = cluster.normalized\_mutual\_info\_score(Y\_test, kpred)

random\_score = cluster.rand\_score(Y\_test, kpred)

vms = cluster.v\_measure\_score(Y\_test, kpred)

result = {

'amis': amis,

'adrs': adrs,

'complete\_score': complete\_score,

'norm': norm,

'random\_score': random\_score,

'vms': vms

}

return result

index.html

{% load static %}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta name="description" content="">

<meta name="author" content="">

<title>Diabetes Reviews</title>

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

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

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

<link href='http://fonts.googleapis.com/css?family=Open+Sans:600italic,400,800,700,300' rel='stylesheet' type='text/css'>

<link href='http://fonts.googleapis.com/css?family=BenchNine:300,400,700' rel='stylesheet' type='text/css'>

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

<!--[if lt IE 9]>

<script src="js/html5shiv.js"></script>

<script src="js/respond.min.js"></script>

<![endif]-->

</head>

<body>

<!-- ====================================================

header section -->

<header class="top-header">

<div class="container">

<div class="row">

<div class="col-xs-5 header-logo">

<br>

<a href="{%url 'index'%}"><h1 alt="" class="img-responsive logo">Diabetes Reviews</h1></a>

</div>

<div class="col-md-7">

<nav class="navbar navbar-default">

<div class="container-fluid nav-bar">

<!-- Brand and toggle get grouped for better mobile display -->

<div class="navbar-header">

<button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#bs-example-navbar-collapse-1">

<span class="sr-only">Toggle navigation</span>

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

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

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

</button>

</div>

<!-- Collect the nav links, forms, and other content for toggling -->

<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">

<ul class="nav navbar-nav navbar-right">

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

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

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

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

<!-- <li><a class="menu" href="{% url 'index' %}"> contact us</a></li>-->

</ul>

</div><!-- /navbar-collapse -->

</div><!-- / .container-fluid -->

</nav>

</div>

</div>

</div>

</header> <!-- end of header area -->

<section class="slider" id="home">

<div class="container-fluid">

<div class="row">

<div id="carouselHacked" class="carousel slide carousel-fade" data-ride="carousel">

<div class="header-backup"></div>

<!-- Wrapper for slides -->

<div class="carousel-inner" role="listbox">

<div class="item active">

<img src="{% static 'img/slide-one.jpg' %}" alt="">

<div class="carousel-caption">

<h3 style="font-size:4.5rem;color:black">Supervised and Unsupervised Machine Learning based Review on Diabetes Care</h3>

<!-- <p>highquality service for men &amp; women</p>-->

<!-- <button>learn more</button>-->

</div>

</div>

<div class="item">

<img src="{% static 'img/slide-two.jpg' %}" alt="">

<div class="carousel-caption">

<h3 style="font-size:4.5rem;color:black">Supervised and Unsupervised Machine Learning based Review on Diabetes Care</h3>

<!-- <p>highquality service for men &amp; women</p>-->

<!-- <button>learn more</button>-->

</div>

</div>

<div class="item">

<img src="{% static 'img/slide-three.jpg' %}" alt="">

<div class="carousel-caption">

<h3 style="font-size:4.5rem;color:black">Supervised and Unsupervised Machine Learning based Review on Diabetes Care</h3>

<!-- <p>highquality service for men &amp; women</p>-->

<!-- <button>learn more</button>-->

</div>

</div>

<!-- <div class="item">-->

<!-- <img src="{% static 'img/slide-four.jpg' %}" alt="">-->

<!-- <div class="carousel-caption">-->

<!-- <h3 style="font-size:4.5rem;color:black">Supervised and Unsupervised Machine Learning based Review on Diabetes Care</h3>-->

<!--&lt;!&ndash; <p>highquality service for men &amp; women</p>&ndash;&gt;-->

<!--&lt;!&ndash; <button>learn more</button>&ndash;&gt;-->

<!-- </div>-->

<!-- </div>-->

</div>

<!-- Controls -->

<a class="left carousel-control" href="#carouselHacked" role="button" data-slide="prev">

<span class="glyphicon glyphicon-chevron-left" aria-hidden="true"></span>

<span class="sr-only">Previous</span>

</a>

<a class="right carousel-control" href="#carouselHacked" role="button" data-slide="next">

<span class="glyphicon glyphicon-chevron-right" aria-hidden="true"></span>

<span class="sr-only">Next</span>

</a>

</div>

</div>

</div>

</section><!-- end of slider section --><br><br><br><br>

<!-- footer starts here -->

<center><footer >

<p>&copy;2021 Alex Corporation All right reserved</p>

</footer></center>

<script src="{% static 'js/jquery-2.1.1.js' %}"></script>

<script src="http://maps.google.com/maps/api/js?sensor=true"></script>

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

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

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

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

</body>

</html>