**SAMPLE CODE:**

**User side views**:

# 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

import pandas as pd

# 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 JruvikaDatasetView(request):

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

df = pd.read\_csv(path, nrows=100)

df = df.to\_html

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

def RealorFakeDatasetView(request):

fakeNews = settings.MEDIA\_ROOT + "//" + 'FakeNews.csv'

realNews = settings.MEDIA\_ROOT + "//" + 'RealNews.csv'

fakeN = pd.read\_csv(fakeNews, nrows=100)

realN = pd.read\_csv(realNews, nrows=100)

df = pd.concat([fakeN, realN], axis="columns")

df = df.to\_html

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

def usrJruvikaFNDML(request):

from .utility import JruvikaMLEDA

svm\_acc, svm\_report = JruvikaMLEDA.process\_SVM()

svm\_report = pd.DataFrame(svm\_report).transpose()

svm\_report = pd.DataFrame(svm\_report)

lg\_acc, lg\_report = JruvikaMLEDA.process\_LogisticRegression()

lg\_report = pd.DataFrame(lg\_report).transpose()

lg\_report = pd.DataFrame(lg\_report)

rf\_acc, rf\_report = JruvikaMLEDA.process\_randomForest()

rf\_report = pd.DataFrame(rf\_report).transpose()

rf\_report = pd.DataFrame(rf\_report)

nb\_acc, nb\_report = JruvikaMLEDA.process\_naiveBayes()

nb\_report = pd.DataFrame(nb\_report).transpose()

nb\_report = pd.DataFrame(nb\_report)

knn\_acc, knn\_report = JruvikaMLEDA.process\_knn()

knn\_report = pd.DataFrame(knn\_report).transpose()

knn\_report = pd.DataFrame(knn\_report)

return render(request, 'users/jruvikaMl.html',

{

'svm\_report': svm\_report.to\_html, 'svm\_acc': svm\_acc,

'lg\_report': lg\_report.to\_html, 'lg\_acc': lg\_acc,

'rf\_report': rf\_report.to\_html, 'rf\_acc': rf\_acc,

'nb\_report': nb\_report.to\_html, 'nb\_acc': nb\_acc,

'knn\_report': knn\_report.to\_html, 'knn\_acc': knn\_acc,

})

def usrRealorFakeML(request):

from .utility import ReaorFakeML

results = ReaorFakeML.proces\_real\_or\_fake\_dataset()

return render(request, 'users/usrRealorFakeML.html', {'data': results})

def predictTrustWorthy(request):

if request.method == 'POST':

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

print(news)

from .utility import JruvikaMLEDA

result = JruvikaMLEDA.fake\_news\_det(news)

return render(request, 'users/testform.html', {'msg': result})

else:

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

**base.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="webthemez">

<title>Trustworthiness Assessment</title>

<link rel="stylesheet"

href="https://cdnjs.cloudflare.com/ajax/libs/highlight.js/10.0.3/styles/default.min.css">

<script src="https://cdnjs.cloudflare.com/ajax/libs/highlight.js/10.0.3/highlight.min.js"></script>

<script>hljs.initHighlightingOnLoad();</script>

<!-- core CSS -->

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

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

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

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

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

<!--[if lt IE 9]>

<script src="{%static 'js/html5shiv.js'%}"></script>

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

<![endif]-->

<link rel="shortcut icon" href="{%static 'images/ico/favicon.ico'%}">

</head>

<body id="home">

<header id="header">

<nav id="main-nav" class="navbar navbar-default navbar-fixed-top" role="banner">

<div class="container">

<div class="navbar-header">

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

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

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

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

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

</button>

<a class="navbar-brand" href="{%url 'index'%}"><h2 class="logo">Trustworthiness Assessment</h2>

<!-- <img src="{%static 'images/logo.png'%}" alt="logo">-->

</a>

</div>

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

<ul class="nav navbar-nav">

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

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

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

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

</ul>

</div>

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

</nav><!--/nav-->

</header><!--/header-->

<section id="services">

<div class="container">

{%block contents%}

{%endblock%}

</section>

</div>

<footer id="footer">

<div class="container">

<div class="row">

<div class="col-sm-6">

&copy; 2023 Alex Corporations. Template by <a target="\_blank" href="#"

title="Free Bootstrap Themes and HTML Templates">Alex

Hales</a>

</div>

</div>

</div>

</footer><!--/#footer-->

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

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

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

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

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

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

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

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

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

</body>

</html>

**Index.html:**

{%extends 'base.html'%}

{%block contents%}

<div class="section-header">

<h2 class="section-title wow fadeInDown">Machine Learned Classifiers for Trustworthiness Assessment of Web Information Contents</h2>

<p class="wow fadeInDown">The classifiers are being trained and tested on two different datasets: Fake News Detection (Jruvika/FND) and Real or Fake News that contains full news articles in the form of headline and body. Experiments and result analysis verify that the highest accuracy attained by the projected method is 96.61% on the Fake News Detection dataset using the SVM classifier. The work is also compared with other contemporary techniques.</p>

</div>

<div class="row">

<div class="features">

<div class="col-md-6 col-sm-6 wow fadeInUp" data-wow-duration="300ms" data-wow-delay="0ms">

<div class="media service-box">

<div class="pull-left">

<i class="fa fa-futbol-o"></i>

</div>

<div class="media-body">

<h4 class="media-heading">PorterStemmer</h4>

<pre><code class="python">

def wordopt(text):

text = re.sub('[^a-zA-Z]', ' ',text)

text = text.lower()

text = text.split()

text = [ps.stem(word) for word in text if not word in stopwords.words('english')]

text = ' '.join(text)

return text

</pre>

</code>

</div>

</div>

</div><!--/.col-md-4-->

<div class="col-md-6 col-sm-6 wow fadeInUp" data-wow-duration="300ms" data-wow-delay="200ms">

<div class="media service-box">

<div class="pull-left">

<i class="fa fa-database"></i>

</div>

<div class="media-body">

<h4 class="media-heading">Model Fitting</h4>

<pre>

<code class="python">

vectorization = TfidfVectorizer()

xv\_train = vectorization.fit\_transform(x\_train)

xv\_test = vectorization.transform(x\_test)

svm\_model = SVC(kernel='linear')

#Fitting training set to the model

svm\_model.fit(xv\_train,y\_train)

#Predicting the test set results based on the model

svm\_y\_pred = svm\_model.predict(xv\_test)

#Calculate the accuracy score of this model

score = accuracy\_score(y\_test,svm\_y\_pred)

</code>

</pre>

</div>

</div>

</div><!--/.col-md-4-->

</div>

</div><!--/.row-->

{%endblock%}

**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(

max\_length=100)

status = forms.CharField(widget=forms.HiddenInpu attrs={'autocomplete': 'off', 'pattern': '[A-Za-z ]+', 'title': 'Enter Characters Only '}), required=True,

t(), initial='waiting', max\_length=100)

class Meta():

model = UserRegistrationModel

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