CODING

```
from django.shortcuts import render
from django.template import RequestContext
from django.contrib import messages
from django.http import HttpResponse
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score
from sklearn import svm
from lightgbm import LGBMClassifier
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
import seaborn as sns

global precision, recall, fscore, accuracy

X = np.load("model/X.txt.npy")
Y = np.load("model/Y.txt.npy")
indices = np.arange(X.shape[0])
np.random.shuffle(indices)
X = X[indices]
Y = Y[indices]



with open('model/tfidf.txt', 'rb') as file:
    tfidf = pickle.load(file)
file.close()
X = tfidf.fit_transform(X).toarray()
print(X.shape)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)

if os.path.exists('model/svm.txt'):
    with open('model/svm.txt', 'rb') as file:
        svm_cls = pickle.load(file)
    file.close()
else:
    svm_cls = svm.SVC()
    svm_cls.fit(X_train, y_train)
    with open('model/svm.txt', 'wb') as file:
        pickle.dump(svm_cls, file)
    file.close()

if os.path.exists('model/lgbm.txt'):
    with open('model/lgbm.txt', 'rb') as file:
        lgbm_cls = pickle.load(file)
    file.close()
else:
    lgbm_cls = LGBMClassifier()
    lgbm_cls.fit(X_train, y_train)
    with open('model/lgbm.txt', 'wb') as file:
        pickle.dump(lgbm_cls, file)
    file.close()

with open('model/rf.txt', 'rb') as file:
    rf_cls = pickle.load(file)
file.close()

def RunSVM(request):
    if request.method == 'GET':
        global precision, recall, fscore, accuracy
        global X_train, X_test, y_train, y_test
        precision = []
        accuracy = []
        fscore = []
        recall = []
        predict = svm_cls.predict(X_test)
        acc = accuracy_score(y_test,predict)*100
        p = precision_score(y_test,predict,average='macro') * 100
        r = recall_score(y_test,predict,average='macro') * 100
        f = f1_score(y_test,predict,average='macro') * 100
        precision.append(p)
        recall.append(r)
        fscore.append(f)
        accuracy.append(acc)
        output = ""
        output+='<tr><td><font size="" color="black">SVM</td>'
        output+='<td><font size="" color="black">'+str(accuracy[0])+'</td>'
        output+='<td><font size="" color="black">'+str(precision[0])+'</td>'
        output+='<td><font size="" color="black">'+str(recall[0])+'</td>'
        output+='<td><font size="" color="black">'+str(fscore[0])+'</td>'

        LABELS = ['Normal URL','Phishing URL']
        conf_matrix = confusion_matrix(y_test, predict)
        plt.figure(figsize =(6, 6))
        ax = sns.heatmap(conf_matrix, xticklabels = LABELS, yticklabels = LABELS, annot = True, cmap="viridis" ,fmt ="g");
        ax.set_ylim([0,2])
        plt.title("SVM Confusion matrix")
        plt.ylabel('True class')
        plt.xlabel('Predicted class')
        plt.show()
        context= {'data':output}
        return render(request, 'ViewOutput.html', context)

def RunLGBM(request):
    if request.method == 'GET':
        global precision, recall, fscore, accuracy
        global X_train, X_test, y_train, y_test

        predict = lgbm_cls.predict(X_test)
        acc = accuracy_score(y_test,predict)*100
        p = precision_score(y_test,predict,average='macro') * 100
        r = recall_score(y_test,predict,average='macro') * 100
        f = f1_score(y_test,predict,average='macro') * 100
        precision.append(p)
        recall.append(r)
        fscore.append(f)
        accuracy.append(acc)
        output = ""
        output+='<tr><td><font size="" color="black">SVM</td>'
        output+='<td><font size="" color="black">'+str(accuracy[0])+'</td>'
        output+='<td><font size="" color="black">'+str(precision[0])+'</td>'
        output+='<td><font size="" color="black">'+str(recall[0])+'</td>'
        output+='<td><font size="" color="black">'+str(fscore[0])+'</td>'

        output+='<tr><td><font size="" color="black">Light GBM</td>'
        output+='<td><font size="" color="black">'+str(accuracy[1])+'</td>'
        output+='<td><font size="" color="black">'+str(precision[1])+'</td>'
        output+='<td><font size="" color="black">'+str(recall[1])+'</td>'
        output+='<td><font size="" color="black">'+str(fscore[1])+'</td>'

        LABELS = ['Normal URL','Phishing URL']
        conf_matrix = confusion_matrix(y_test, predict)
        plt.figure(figsize =(6, 6))
        ax = sns.heatmap(conf_matrix, xticklabels = LABELS, yticklabels =
LABELS, annot = True, cmap="viridis" ,fmt ="g");
        ax.set_ylim([0,2])
        plt.title("Decision Tree Confusion matrix")
        plt.ylabel('True class')
        plt.xlabel('Predicted class')
        plt.show()
        context= {'data':output}
        return render(request, 'ViewOutput.html', context)



def getData(arr):
    data = ""
    for i in range(len(arr)):
        arr[i] = arr[i].strip()
        if len(arr[i]) > 0:
            data += arr[i]+" "
    return data.strip()

def PredictAction(request):
    if request.method == 'POST':
        global rf_cls, tfidf
        url_input = request.POST.get('t1', False)
        test = []
        arr = url_input.split("/")
        if len(arr) > 0:
            data = getData(arr)
            print(data)
            test.append(data)
            test = tfidf.transform(test).toarray()
            print(test)
            print(test.shape)
            predict = rf_cls.predict(test)
            print(predict)
            predict = predict[0]
            output = ""
            if predict == 0:
                output = url_input+" Given URL Predicted as Genuine"
            if predict == 1:
                output = url_input+" PHISHING Detected in Given URL"
            context= {'data':output}
            return render(request, 'Predict.html', context)
        else:
            context= {'data':"Entered URL is not valid"}
            return render(request, 'Predict.html', context)



def index(request):
    if request.method == 'GET':
        return render(request, 'index.html', {})

def Predict(request):
    if request.method == 'GET':
       return render(request, 'Predict.html', {})

def AdminLogin(request):
    if request.method == 'GET':
       return render(request, 'AdminLogin.html', {})

def AdminLoginAction(request):
    if request.method == 'POST':
        global userid
        user = request.POST.get('t1', False)
        password = request.POST.get('t2', False)
        if user == "admin" and password == "admin":
            context= {'data':'Welcome '+user}
            return render(request, 'AdminScreen.html', context)
        else:
            context= {'data':'Invalid Login'}
            return render(request, 'AdminLogin.html', context)
```

```
{% load static %}
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta name="description" content="" />
<meta name="keywords" content="" />
<title>PHISHING WEBSITE DETECTION</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<link rel="stylesheet" type="text/css" href="{% static 'style.css' %}" />
<script language="javascript">
        function validate(formObj)
        {
```

```
	if(formObj.t1.value.length==0)
	{
	alert("Please Enter URL");
	formObj.t1.focus();
	return false;
	}

	return true;
	}
	</script>
</head>
<body>
<div id="wrapper">
	<div id="header">
		<div id="logo">
			<center><font size="4" color="yellow">Malware Cyber
Threat Attacks Using and Decision Tree Models </font></center>
		</div>
		<div id="slogan">

		</div>
	</div>
	<div id="menu">
		<ul><center>
			<li class="first current_page_item"><a href="{% url
'RunSVM' %}"><font size="" color="yellow">Run SVM Algorithm
</font></a></li>
			<li class="first current_page_item"><a href="{% url
'RunLGBM' %}"><font size="" color="yellow">Run Light GBM
Algorithm</font></a></li>

		<li class="first current_page_item"><a href="{% url 'Predict'
%}"><font size="" color="yellow">Test Your URL</font></a></li>

		<li class="first current_page_item"><a href="{% url 'index'
%}"><font size="" color="yellow">Logout</font></a></li>
		</center></ul>		<br class="clearfix" />
```

```
                                        </div>


        <div id="splash">
                    <img class="pic" src="{% static 'images/investor.jpg' %}"
width="870" height="230" alt="" />
        </div>
                                <center>
<form name="f1" method="post" action={% url 'PredictAction' %}
onsubmit="return validate(this);">
<br/>{% csrf_token %}<br/>
   <h5><b>Phishing URL Detection Screen</b></h5>

   <font size="" color="red"><center>{{ data|safe }}</center></font>



                                                    <table align="center" width="40" >
                                 <tr><td><font size=""
color="black">Enter&nbsp;URL</b></td><td><input type="text" name="t1"
style="font-family: Comic Sans MS" size="50"/></td></tr>



                                <tr><td></td><td><input type="submit" value="Submit">

                                </td>
                                </tr>
                                </table>
                                        </div>


                                        </div>
                                        <br/><br/>
        </body>
</html>
{% load static %}

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```
<meta name="description" content="" />
<meta name="keywords" content="" />
<title>PHISHING WEBSITE DETECTION</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<link rel="stylesheet" type="text/css" href="{% static 'style.css' %}"/>
</head>
<body>
<div id="wrapper">
    <div id="header">
        <div id="logo">
            <center><font size="4" color="yellow">Malware Cyber
Threat Attacks Using and Decision Tree Models</font></center>
        </div>
        <div id="slogan">

        </div>
    </div>
    <div id="menu">
        <ul><center>
            <li class="first current_page_item"><a href="{% url 'index'
%}"><font size="" color="yellow">Home</font></a></li>
            <li class="first current_page_item"><a href="{% url
'AdminLogin' %}"><font size="" color="yellow">Admin Login
Here</font></a></li>

        </center></ul>
        <br class="clearfix" />
    </div>
    <div id="splash">
        <img class="pic" src="{% static 'images/investor.jpg' %}"
width="870" height="230" alt="" />
    </div>
    <br/>
    <p align="justify"><font size="3" color="black"  style="font-family:
Comic Sans MS">
    PHISHING WEBSITE DETECTION</p>
</body>
```

```
</html>

{% load static %}
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta name="description" content="" />
<meta name="keywords" content="" />
<title>PHISHING WEBSITE DETECTION</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<link rel="stylesheet" type="text/css" href="{% static 'style.css' %}"/>
</head>
<body>
<div id="wrapper">
    <div id="header">
        <div id="logo">
            <center><font size="4" color="yellow">Malware Cyber
Threat Attacks Using and Decision Tree Models </font></center>
        </div>
        <div id="slogan">

        </div>
    </div>
    <div id="menu">

    <ul><center>
            <li class="first current_page_item"><a href="{% url
'RunSVM' %}"><font size="" color="yellow">Run SVM Algorithm
</font></a></li>
            <li class="first current_page_item"><a href="{% url
'RunLGBM' %}"><font size="" color="yellow">Run Light GBM
Algorithm</font></a></li>

        <li class="first current_page_item"><a href="{% url 'Predict'
%}"><font size="" color="yellow">Test Your URL</font></a></li>
```

```
                <li class="first current_page_item"><a href="{% url 'index'
 %}"><font size="" color="yellow">Logout</font></a></li>
                </center></ul>
                <br class="clearfix" />
        </div>
        <div id="splash">
                <img class="pic" src="{% static 'images/investor.jpg' %}"
width="870" height="230" alt="" />
        </div>
        <br/>
         <font size="" color="red"><center>{{ data|safe }}</center></font><br/>

<p align="justify"><font size="3" color="black"  style="font-family: Comic
Sans MS">
        Malware Cyber Threat Attacks Using and Decision Tree Models</p>
</body>
</html>
from django.shortcuts import render
from django.template import RequestContext
from django.contrib import messages
from django.http import HttpResponse
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score
from sklearn import svm
from lightgbm import LGBMClassifier
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

```
X = np.load("model/X.txt.npy")
Y = np.load("model/Y.txt.npy")
indices = np.arange(X.shape[0])
np.random.shuffle(indices)
X = X[indices]
Y = Y[indices]


with open('model/tfidf.txt', 'rb') as file:
    tfidf = pickle.load(file)
file.close()
X = tfidf.fit_transform(X).toarray()
print(X.shape)
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)

if os.path.exists('model/svm.txt'):
    with open('model/svm.txt', 'rb') as file:
        svm_cls = pickle.load(file)
    file.close()
else:
    svm_cls = svm.SVC()
    svm_cls.fit(X_train, y_train)
    with open('model/svm.txt', 'wb') as file:
        pickle.dump(svm_cls, file)
    file.close()

if os.path.exists('model/lgbm.txt'):
    with open('model/lgbm.txt', 'rb') as file:
        lgbm_cls = pickle.load(file)
    file.close()
else:
    lgbm_cls = LGBMClassifier()
    lgbm_cls.fit(X_train, y_train)
    with open('model/lgbm.txt', 'wb') as file:
        pickle.dump(lgbm_cls, file)
    file.close()

with open('model/rf.txt', 'rb') as file:
    rf_cls = pickle.load(file)
file.close()

def RunSVM(request):
    if request.method == 'GET':
        global precision, recall, fscore, accuracy
        global X_train, X_test, y_train, y_test
        precision = []
        accuracy = []
        fscore = []
        recall = []
        predict = svm_cls.predict(X_test)
        acc = accuracy_score(y_test,predict)*100
        p = precision_score(y_test,predict,average='macro') * 100
        r = recall_score(y_test,predict,average='macro') * 100
        f = f1_score(y_test,predict,average='macro') * 100
        precision.append(p)
        recall.append(r)
        fscore.append(f)
        accuracy.append(acc)
        output = ""
        output+='<tr><td><font size="" color="black">SVM</td>'
        output+='<td><font size="" color="black">'+str(accuracy[0])+'</td>'
        output+='<td><font size="" color="black">'+str(precision[0])+'</td>'
        output+='<td><font size="" color="black">'+str(recall[0])+'</td>'
        output+='<td><font size="" color="black">'+str(fscore[0])+'</td>'

        LABELS = ['Normal URL','Phishing URL']
        conf_matrix = confusion_matrix(y_test, predict)
        plt.figure(figsize =(6, 6))
        ax = sns.heatmap(conf_matrix, xticklabels = LABELS, yticklabels =
LABELS, annot = True, cmap="viridis" ,fmt ="g");
        ax.set_ylim([0,2])
        plt.title("SVM Confusion matrix")
        plt.xlabel('Predicted class')
        plt.show()
        context= {'data':output}
        return render(request, 'ViewOutput.html', context)

def RunLGBM(request):
    if request.method == 'GET':
        global precision, recall, fscore, accuracy
        global X_train, X_test, y_train, y_test

        predict = lgbm_cls.predict(X_test)
        acc = accuracy_score(y_test,predict)*100
        p = precision_score(y_test,predict,average='macro') * 100
        r = recall_score(y_test,predict,average='macro') * 100
        f = f1_score(y_test,predict,average='macro') * 100
        precision.append(p)
        recall.append(r)
        fscore.append(f)
        accuracy.append(acc)
        output = ""
        output+='<tr><td><font size="" color="black">SVM</td>'
        output+='<td><font size="" color="black">'+str(accuracy[0])+'</td>'
        output+='<td><font size="" color="black">'+str(precision[0])+'</td>'
        output+='<td><font size="" color="black">'+str(recall[0])+'</td>'
        output+='<td><font size="" color="black">'+str(fscore[0])+'</td>'

        output+='<tr><td><font size="" color="black">Light GBM</td>'
        output+='<td><font size="" color="black">'+str(accuracy[1])+'</td>'
        output+='<td><font size="" color="black">'+str(precision[1])+'</td>'
        output+='<td><font size="" color="black">'+str(recall[1])+'</td>'
        output+='<td><font size="" color="black">'+str(fscore[1])+'</td>'

        LABELS = ['Normal URL','Phishing URL']
        conf_matrix = confusion_matrix(y_test, predict)
        plt.figure(figsize =(6, 6))
        ax = sns.heatmap(conf_matrix, xticklabels = LABELS, yticklabels =
LABELS, annot = True, cmap="viridis" ,fmt ="g");
        ax.set_ylim([0,2])
        plt.title("Decision Tree Confusion matrix")
        plt.ylabel('True class')
        plt.xlabel('Predicted class')
        plt.show()
        context= {'data':output}
        return render(request, 'ViewOutput.html', context)



def getData(arr):
    data = ""
    for i in range(len(arr)):
        arr[i] = arr[i].strip()
        if len(arr[i]) > 0:
            data += arr[i]+" "
    return data.strip()

def PredictAction(request):
    if request.method == 'POST':
        global rf_cls, tfidf
        url_input = request.POST.get('t1', False)
        test = []
        arr = url_input.split("/")
        if len(arr) > 0:
            data = getData(arr)
            print(data)
            test.append(data)
            test = tfidf.transform(test).toarray()
            print(test)
            print(test.shape)
            predict = rf_cls.predict(test)
            print(predict)
            predict = predict[0]
            output = ""
            if predict == 0:
                output = url_input+" Given URL Predicted as Genuine"
            if predict == 1:
                output = url_input+" PHISHING Detected in Given URL"
            context= {'data':output}
            return render(request, 'Predict.html', context)
        else:
            context= {'data':"Entered URL is not valid"}
            return render(request, 'Predict.html', context)



def index(request):
    if request.method == 'GET':
       return render(request, 'index.html', {})

def Predict(request):
    if request.method == 'GET':
       return render(request, 'Predict.html', {})

def AdminLogin(request):
    if request.method == 'GET':
       return render(request, 'AdminLogin.html', {})

def AdminLoginAction(request):
    if request.method == 'POST':
        global userid
        user = request.POST.get('t1', False)
        password = request.POST.get('t2', False)
        if user == "admin" and password == "admin":
            context= {'data':'Welcome '+user}
            return render(request, 'AdminScreen.html', context)
        else:
            context= {'data':'Invalid Login'}
            return render(request, 'AdminLogin.html', context)
```