SAMPLE CODE

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'amachine_learning_based_classification_ddosattacks.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
from django.db.models import Count
from django.db.models import Q
from django.shortcuts import render, redirect, get_object_or_404
import datetime
import openpyxl


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
```

```
from sklearn.ensemble import VotingClassifier
from sklearn.tree import DecisionTreeClassifier
import warnings
warnings.filterwarnings("ignore")
plt.style.use('ggplot')
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score

# Create your views here.
from Remote_User.models import
ClientRegister_Model,ddos_attacks_prediction,detection_ratio,detection_accuracy

def login(request):

    if request.method == "POST" and 'submit1' in request.POST:

        username = request.POST.get('username')
        password = request.POST.get('password')
        try:
            enter = ClientRegister_Model.objects.get(username=username,password=password)
            request.session["userid"] = enter.id

            return redirect('ViewYourProfile')
        except:
            pass

    return render(request,'RUser/login.html')

def Register1(request):
    if request.method == "POST":
        username = request.POST.get('username')
        email = request.POST.get('email')
        password = request.POST.get('password')
        phoneno = request.POST.get('phoneno')
        country = request.POST.get('country')
        state = request.POST.get('state')
        city = request.POST.get('city')
        address = request.POST.get('address')
        gender = request.POST.get('gender')
        ClientRegister_Model.objects.create(username=username, email=email,
password=password, phoneno=phoneno,
                                            country=country, state=state, city=city, address=address,
gender=gender)
        obj = "Registered Successfully"
        return render(request, 'RUser/Register1.html', {'object': obj})
    else:
        return render(request,'RUser/Register1.html')

def ViewYourProfile(request):
    userid = request.session['userid']
    obj = ClientRegister_Model.objects.get(id= userid)
    return render(request,'RUser/ViewYourProfile.html',{'object':obj})

def predict_ddos_attack_type(request):
    if request.method == "POST":

        RID= request.POST.get('RID')
        Protocol= request.POST.get('Protocol')
        ip_src= request.POST.get('ip_src')
        ip_dst= request.POST.get('ip_dst')
        pro_srcport= request.POST.get('pro_srcport')
        pro_dstport= request.POST.get('pro_dstport')
        flags_ack= request.POST.get('flags_ack')
        ip_flags_mf= request.POST.get('ip_flags_mf')
        ip_flags_df= request.POST.get('ip_flags_df')
        ip_flags_rb= request.POST.get('ip_flags_rb')
```

```
pro_seq= request.POST.get('pro_seq')
pro_ack= request.POST.get('pro_ack')
frame_time= request.POST.get('frame_time')
Packets= request.POST.get('Packets')
Bytes1= request.POST.get('Bytes1')
Tx_Packets= request.POST.get('Tx_Packets')
Tx_Bytes= request.POST.get('Tx_Bytes')
Rx_Packets= request.POST.get('Rx_Packets')
Rx_Bytes= request.POST.get('Rx_Bytes')

df = pd.read_csv('Datasets.csv', encoding='latin-1')
df
df.columns

def apply_results(results):
    if (results == "normal"):
        return 0
    elif (results == "smurf"):
        return 1
    elif (results == "Fraggile"):
        return 2
df['Results'] = df['Label'].apply(apply_results)

X = df['RID']
y = df['Results']

print("Reading ID")
print(X)
print("Label")
print(y)

# cv = CountVectorizer(lowercase=False, strip_accents='unicode', ngram_range=(1, 1))
# X = cv.fit_transform(df['RID'].apply(lambda x: np.str_(x)))
cv = CountVectorizer()
```

```
X = cv.fit_transform(X)

models = []
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)
X_train.shape, X_test.shape, y_train.shape

print("Naive Bayes")

from sklearn.naive_bayes import MultinomialNB
NB = MultinomialNB()
NB.fit(X_train, y_train)
predict_nb = NB.predict(X_test)
naivebayes = accuracy_score(y_test, predict_nb) * 100
print(naivebayes)
print(confusion_matrix(y_test, predict_nb))
print(classification_report(y_test, predict_nb))
models.append(('naive_bayes', NB))

# SVM Model
print("SVM")
from sklearn import svm
lin_clf = svm.LinearSVC()
lin_clf.fit(X_train, y_train)
predict_svm = lin_clf.predict(X_test)
svm_acc = accuracy_score(y_test, predict_svm) * 100
print(svm_acc)
print("CLASSIFICATION REPORT")
print(classification_report(y_test, predict_svm))
print("CONFUSION MATRIX")
print(confusion_matrix(y_test, predict_svm))
models.append(('svm', lin_clf))

print("Logistic Regression")
```

```
from sklearn.linear_model import LogisticRegression
reg = LogisticRegression(random_state=0, solver='lbfgs').fit(X_train, y_train)
y_pred = reg.predict(X_test)
print("ACCURACY")
print(accuracy_score(y_test, y_pred) * 100)
print("CLASSIFICATION REPORT")
print(classification_report(y_test, y_pred))
print("CONFUSION MATRIX")
print(confusion_matrix(y_test, y_pred))
models.append(('logistic', reg))

classifier = VotingClassifier(models)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

RID1 = [RID]
vector1 = cv.transform(RID1).toarray()
predict_text = classifier.predict(vector1)

pred = str(predict_text).replace("[", "")
pred1 = pred.replace("]", "")

prediction = int(pred1)

if prediction == 0:
    val = 'Normal'
elif prediction == 1:
    val = 'smurf'
elif prediction == 2:
    val = 'Fraggile'

print(val)
print(pred1)
```

```
        ddos_attacks_prediction.objects.create(
        RID=RID,
        Protocol=Protocol,
        ip_src=ip_src,
        ip_dst=ip_dst,
        pro_srcport=pro_srcport,
        pro_dstport=pro_dstport,
        flags_ack=flags_ack,
        ip_flags_mf=ip_flags_mf,
        ip_flags_df=ip_flags_df,
        ip_flags_rb=ip_flags_rb,
        pro_seq=pro_seq,
        pro_ack=pro_ack,
        frame_time=frame_time,
        Packets=Packets,
        Bytes1=Bytes1,
        Tx_Packets=Tx_Packets,
        Tx_Bytes=Tx_Bytes,
        Rx_Packets=Rx_Packets,
        Rx_Bytes=Rx_Bytes,
       Prediction=val)

        return render(request, 'RUser/predict_ddos_attack_type.html',{'objs': val})
    return render(request, 'RUser/predict_ddos_attack_type.html')

from django.db import models

# Create your models here.
from django.db.models import CASCADE



class ClientRegister_Model(models.Model):
    username = models.CharField(max_length=30)
    email = models.EmailField(max_length=30)
    password = models.CharField(max_length=10)
    phoneno = models.CharField(max_length=10)
    country = models.CharField(max_length=30)
    state = models.CharField(max_length=30)
    city = models.CharField(max_length=30)
    address= models.CharField(max_length=3000)
    gender= models.CharField(max_length=30)

class ddos_attacks_prediction(models.Model):


    RID= models.CharField(max_length=3000)
    Protocol= models.CharField(max_length=3000)
    ip_src= models.CharField(max_length=3000)
    ip_dst= models.CharField(max_length=3000)
    pro_srcport= models.CharField(max_length=3000)
    pro_dstport= models.CharField(max_length=3000)
    flags_ack= models.CharField(max_length=3000)
    ip_flags_mf= models.CharField(max_length=3000)
    ip_flags_df= models.CharField(max_length=3000)
    ip_flags_rb= models.CharField(max_length=3000)
    pro_seq= models.CharField(max_length=3000)
    pro_ack= models.CharField(max_length=3000)
    frame_time= models.CharField(max_length=3000)
    Packets= models.CharField(max_length=3000)
    Bytes1= models.CharField(max_length=3000)
    Tx_Packets= models.CharField(max_length=3000)
    Tx_Bytes= models.CharField(max_length=3000)
    Rx_Packets= models.CharField(max_length=3000)
    Rx_Bytes= models.CharField(max_length=3000)
    Prediction= models.CharField(max_length=3000)

class detection_accuracy(models.Model):
    names = models.CharField(max_length=300)
    ratio = models.CharField(max_length=300)

class detection_ratio(models.Model):
    names = models.CharField(max_length=300)
    ratio = models.CharField(max_length=300)
from django import forms

from Remote_User.models import ClientRegister_Model



class ClientRegister_Form(forms.ModelForm):
    password = forms.CharField(widget=forms.PasswordInput())
    email = forms.EmailField(required=True)

    class Meta:
        model = ClientRegister_Model
        fields = ("username","email","password","phoneno","country","state","city")
# Generated by Django 2.0.5 on 2019-04-23 07:01

from django.db import migrations, models

class Migration(migrations.Migration):
    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='ClientRegister_Model',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
                ('username', models.CharField(max_length=30)),
                ('email', models.EmailField(max_length=30)),
                ('password', models.CharField(max_length=10)),
                ('phoneno', models.IntegerField()),
                ('country', models.CharField(max_length=30)),
                ('state', models.CharField(max_length=30)),
                ('city', models.CharField(max_length=30)),
            ],
        ),
    ]
# Generated by Django 2.0.5 on 2019-04-25 05:53

from django.db import migrations, models
import django.db.models.deletion

class Migration(migrations.Migration):

    dependencies = [
        ('Remote_User', '0001_initial'),
    ]

    operations = [
        migrations.CreateModel(
            name='ClientPosts_Model',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('tdesc', models.CharField(max_length=300)),
                ('uname', models.CharField(max_length=300)),
                ('topics', models.CharField(max_length=300)),
                ('sanalysis', models.CharField(max_length=300)),
                ('senderstatus', models.CharField(default='process', max_length=300)),
                ('ratings', models.IntegerField(default=0)),
                ('userId',       models.ForeignKey(on_delete=django.db.models.deletion.CASCADE,
to='Remote_User.ClientRegister_Model')),
            ],
        ),
    ]
# Generated by Django 2.0.5 on 2019-04-25 09:57

from django.db import migrations, models


class Migration(migrations.Migration):

    dependencies = [
        ('Remote_User', '0002_clientposts_model'),
    ]

    operations = [
        migrations.AddField(
            model_name='clientposts_model',
            name='uname',
            field=models.IntegerField(default=0),
        ),
    ]
# Generated by Django 2.0.5 on 2019-04-29 04:57

from django.db import migrations, models


class Migration(migrations.Migration):

    dependencies = [
        ('Remote_User', '0003_clientposts_model_usefulcounts'),
    ]
    operations = [
        migrations.AddField(
            model_name='clientposts_model',
            name='uses',
            field=models.CharField(default='', max_length=100),
            preserve_default=False,
        ),
        migrations.AddField(
            model_name='clientposts_model',
            name='tname',
            field=models.CharField(default='', max_length=50),
            preserve_default=False,
        ),
    ]
# Generated by Django 2.0.5 on 2019-04-29 05:15

from django.db import migrations, models


class Migration(migrations.Migration):

    dependencies = [
        ('Remote_User', '0004_auto_20190429_1027'),
    ]

    operations = [
        migrations.AddField(
            model_name='clientposts_model',
            name='dislikes',
            field=models.IntegerField(default=0),
        ),
    ]
# Generated by Django 2.0.5 on 2019-04-29 05:19
```

```
from django.db import migrations, models


class Migration(migrations.Migration):

    dependencies = [
        ('Remote_User', '0005_clientposts_model_dislikes'),
    ]

    operations = [
        migrations.CreateModel(
            name='review_Model',
            fields=[

                ('uname', models.CharField(max_length=100)),
                ('ureview', models.CharField(max_length=100)),
                ('tname', models.CharField(max_length=300)),
                ('suggestion', models.CharField(max_length=300)),
                ('dt', models.CharField(max_length=300)),
                ('sanalysis', models.CharField(max_length=300)),
            ],
        ),
    ]
# Generated by Django 2.0.5 on 2019-04-30 04:45

from django.db import migrations, models


class Migration(migrations.Migration):

    dependencies = [
        ('Remote_User', '0006_review_model'),
    ]
    operations = [
        migrations.AddField(
            model_name='clientposts_model',
            name='uname',
            field=models.CharField(default='', max_length=50),
            preserve_default=False,
        ),
    ]

from django.db.models import  Count, Avg
from django.shortcuts import render, redirect
from django.db.models import Count
from django.db.models import Q
import datetime
import xlwt
from django.http import HttpResponse

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import f1_score, matthews_corrcoef
from sklearn.tree import DecisionTreeClassifier

# Create your views here.
from Remote_User.models import
ClientRegister_Model,ddos_attacks_prediction,detection_ratio,detection_accuracy

def serviceproviderlogin(request):
    if request.method  == "POST":
        admin = request.POST.get('username')
        password = request.POST.get('password')
        if admin == "Admin" and password =="Admin":
            return redirect('View_Remote_Users')

    return render(request,'SProvider/serviceproviderlogin.html')

def Find_View_Prediction_DDOS_Attack_Type_Ratio(request):
    detection_ratio.objects.all().delete()
    ratio = ""
    kword = 'normal'
    print(kword)
    obj = ddos_attacks_prediction.objects.all().filter(Q(Prediction=kword))
    obj1 =ddos_attacks_prediction.objects.all()
    count = obj.count();
    count1 = obj1.count();
    ratio = (count / count1) * 100
    if ratio != 0:
        detection_ratio.objects.create(names=kword, ratio=ratio)

    ratio1 = ""
    kword1 = 'Fraggile'
    print(kword1)
    obj1 = ddos_attacks_prediction.objects.all().filter(Q(Prediction=kword1))
    obj11 = ddos_attacks_prediction.objects.all()
    count1 = obj1.count();
    count11 = obj11.count();
    ratio1 = (count1 / count11) * 100
    if ratio1 != 0:
        detection_ratio.objects.create(names=kword1, ratio=ratio1)

    ratio12 = ""
    kword12 = 'smurf'
    print(kword12)
    obj12 = ddos_attacks_prediction.objects.all().filter(Q(Prediction=kword12))
    obj112 = ddos_attacks_prediction.objects.all()
    count12 = obj12.count();
    count112 = obj112.count();
    ratio12 = (count12 / count112) * 100
    if ratio12 != 0:
        detection_ratio.objects.create(names=kword12, ratio=ratio12)

    obj = detection_ratio.objects.all()
    return render(request, 'SProvider/Find_View_Prediction_DDOS_Attack_Type_Ratio.html',
{'objs': obj})

def View_Remote_Users(request):
    obj=ClientRegister_Model.objects.all()
    return render(request,'SProvider/View_Remote_Users.html',{'objects':obj})

def ViewTrendings(request):
    topic =
ddos_attacks_prediction.objects.values('topics').annotate(dcount=Count('topics')).order_by('-
dcount')
    return  render(request,'SProvider/ViewTrendings.html',{'objects':topic})

def charts(request,chart_type):
    chart1 = detection_ratio.objects.values('names').annotate(dcount=Avg('ratio'))
    return render(request,"SProvider/charts.html", {'form':chart1, 'chart_type':chart_type})

def charts1(request,chart_type):
    chart1 = detection_accuracy.objects.values('names').annotate(dcount=Avg('ratio'))
    return render(request,"SProvider/charts1.html", {'form':chart1, 'chart_type':chart_type})

def View_Prediction_DDOS_Attack_Type(request):
    obj =ddos_attacks_prediction.objects.all()
    return render(request, 'SProvider/View_Prediction_DDOS_Attack_Type.html',
{'list_objects': obj})
```

```
def likeschart(request,like_chart):
    charts =detection_accuracy.objects.values('names').annotate(dcount=Avg('ratio'))
    return render(request,"SProvider/likeschart.html", {'form':charts, 'like_chart':like_chart})

def Download_Trained_DataSets(request):

    response = HttpResponse(content_type='application/ms-excel')
    # decide file name
    response['Content-Disposition'] = 'attachment; filename="PredictedData.xls"'
    # creating workbook
    wb = xlwt.Workbook(encoding='utf-8')
    # adding sheet
    ws = wb.add_sheet("sheet1")
    # Sheet header, first row
    row_num = 0
    font_style = xlwt.XFStyle()
    # headers are bold
    font_style.font.bold = True
    # writer = csv.writer(response)
    obj = ddos_attacks_prediction.objects.all()
    data = obj  # dummy method to fetch data.
    for my_row in data:
        row_num = row_num + 1

        ws.write(row_num, 0, my_row.RID, font_style)
        ws.write(row_num, 1, my_row.Protocol, font_style)
        ws.write(row_num, 2, my_row.ip_src, font_style)
        ws.write(row_num, 3, my_row.ip_dst, font_style)
        ws.write(row_num, 4, my_row.pro_srcport, font_style)
        ws.write(row_num, 5, my_row.pro_dstport, font_style)
        ws.write(row_num, 6, my_row.flags_ack, font_style)
        ws.write(row_num, 7, my_row.ip_flags_mf, font_style)
        ws.write(row_num, 8, my_row.ip_flags_df, font_style)
        ws.write(row_num, 9, my_row.ip_flags_rb, font_style)
        ws.write(row_num, 10, my_row.pro_seq, font_style)
        ws.write(row_num, 11, my_row.pro_ack, font_style)
        ws.write(row_num, 12, my_row.frame_time, font_style)
        ws.write(row_num, 13, my_row.Packets, font_style)
        ws.write(row_num, 14, my_row.Bytes1, font_style)
        ws.write(row_num, 15, my_row.Tx_Packets, font_style)
        ws.write(row_num, 16, my_row.Tx_Bytes, font_style)
        ws.write(row_num, 17, my_row.Rx_Packets, font_style)
        ws.write(row_num, 18, my_row.Rx_Bytes, font_style)
        ws.write(row_num, 19, my_row.Prediction, font_style)

    wb.save(response)
    return response

def Train_Test_DataSets(request):
    detection_accuracy.objects.all().delete()

    df = pd.read_csv('Datasets.csv',encoding='latin-1')
    df
    df.columns

    def apply_results(results):
        if (results == "normal"):
            return 0
        elif (results == "smurf"):
            return 1
        elif (results == "Fraggile"):
            return 2

    df['Results'] = df['Label'].apply(apply_results)

    X = df['RID'].apply(str)
    y = df['Results']
```

```
X.shape, y.shape



print("Reading ID")
print(X)
print("Label")
print(y)

cv = CountVectorizer(lowercase=False, strip_accents='unicode', ngram_range=(1, 1))
X = cv.fit_transform(df['RID'].apply(lambda x: np.str_(x)))
#cv = CountVectorizer()
#X = cv.fit_transform(X)

models = []
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)
X_train.shape, X_test.shape, y_train.shape

print("Naive Bayes")

from sklearn.naive_bayes import MultinomialNB
NB = MultinomialNB()
NB.fit(X_train, y_train)
predict_nb = NB.predict(X_test)
naivebayes = accuracy_score(y_test, predict_nb) * 100
print(naivebayes)
print(confusion_matrix(y_test, predict_nb))
print(classification_report(y_test, predict_nb))
models.append(('naive_bayes', NB))
detection_accuracy.objects.create(names="Naive Bayes", ratio=naivebayes)

# SVM Model
print("SVM")
```

```
from sklearn import svm
lin_clf = svm.LinearSVC()
lin_clf.fit(X_train, y_train)
predict_svm = lin_clf.predict(X_test)
svm_acc = accuracy_score(y_test, predict_svm) * 100
print(svm_acc)
print("CLASSIFICATION REPORT")
print(classification_report(y_test, predict_svm))
print("CONFUSION MATRIX")
print(confusion_matrix(y_test, predict_svm))
models.append(('svm', lin_clf))
detection_accuracy.objects.create(names="SVM", ratio=svm_acc)

print("Logistic Regression")

from sklearn.linear_model import LogisticRegression
reg = LogisticRegression(random_state=0, solver='lbfgs').fit(X_train, y_train)
y_pred = reg.predict(X_test)
print("ACCURACY")
print(accuracy_score(y_test, y_pred) * 100)
print("CLASSIFICATION REPORT")
print(classification_report(y_test, y_pred))
print("CONFUSION MATRIX")
print(confusion_matrix(y_test, y_pred))
models.append(('logistic', reg))

detection_accuracy.objects.create(names="Logistic Regression",
ratio=accuracy_score(y_test, y_pred) * 100)

print("Random Forest Classifier")
from sklearn.ensemble import RandomForestClassifier
rf_clf = RandomForestClassifier()
rf_clf.fit(X_train, y_train)
rfpredict = rf_clf.predict(X_test)
```

```
    print("ACCURACY")
    print(accuracy_score(y_test, rfpredict) * 100)
    print("CLASSIFICATION REPORT")
    print(classification_report(y_test, rfpredict))
    print("CONFUSION MATRIX")
    print(confusion_matrix(y_test, rfpredict))
    models.append(('RandomForestClassifier', rf_clf))
    detection_accuracy.objects.create(names="Random Forest Classifier", ratio=accuracy_score(y_test, rfpredict) * 100)

    predicts = 'predicts.csv'
    df.to_csv(predicts, index=False)
    df.to_markdown

    obj = detection_accuracy.objects.all()

    return render(request,'SProvider/Train_Test_DataSets.html', {'objs': obj})
"""
ASGI config for amachine_learning_based_classification_ddosattacks

It exposes the ASGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/3.0/howto/deployment/asgi/
"""

import os

from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'amachine_learning_based_classification_ddosattacks.settings')

application = get_asgi_application()
```

```
import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'm+1edl5m-5@u9u!b8-=4-4mq&o1%agco2xpl8c!7sn7!eowjk#'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'Remote_User',
    'Service_Provider',
]


MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'amachine_learning_based_classification_ddosattacks.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [(os.path.join(BASE_DIR,'Template/htmls'))],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION                                                          =
'amachine_learning_based_classification_ddosattacks.wsgi.application'

# Database
# https://docs.djangoproject.com/en/3.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'ddos_attacks_prediction',
        'USER':'root',
        'PASSWORD': '',
        'HOST' :'127.0.0.1',
        'PORT' :'3306',
    }
}

# Password validation
# https://docs.djangoproject.com/en/3.0/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Internationalization
# https://docs.djangoproject.com/en/3.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'
```

```
USE_I18N = True

USE_L10N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC_URL = '/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR,'Template/images')]
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'Template/media')

STATIC_ROOT = '/static/'

STATIC_URL = '/static/'

"""amachine_learning_based_classification_ddosattacks  URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.0/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.conf.urls import url
from django.contrib import admin
from Remote_User import views as remoteuser
from amachine_learning_based_classification_ddosattacks  import settings
from Service_Provider import views as serviceprovider
from django.conf.urls.static import static


urlpatterns = [
    url('admin/', admin.site.urls),
    url(r'^$', remoteuser.login, name="login"),
    url(r'^Register1/$', remoteuser.Register1, name="Register1"),
    url(r'^predict_ddos_attack_type/$', remoteuser.predict_ddos_attack_type,
name="predict_ddos_attack_type"),
    url(r'^ViewYourProfile/$', remoteuser.ViewYourProfile, name="ViewYourProfile"),
    url(r'^serviceproviderlogin/$',serviceprovider.serviceproviderlogin,
name="serviceproviderlogin"),

url(r'View_Remote_Users/$',serviceprovider.View_Remote_Users,name="View_Remote_Us
ers"),
    url(r'^charts/(?P<chart_type>\w+)', serviceprovider.charts,name="charts"),
    url(r'^charts1/(?P<chart_type>\w+)', serviceprovider.charts1, name="charts1"),
    url(r'^likeschart/(?P<like_chart>\w+)', serviceprovider.likeschart, name="likeschart"),
    url(r'^Find_View_Prediction_DDOS_Attack_Type_Ratio/$',
serviceprovider.Find_View_Prediction_DDOS_Attack_Type_Ratio,
name="Find_View_Prediction_DDOS_Attack_Type_Ratio"),
    url(r'^Train_Test_DataSets/$', serviceprovider.Train_Test_DataSets,
name="Train_Test_DataSets"),
    url(r'^View_Prediction_DDOS_Attack_Type/$',
serviceprovider.View_Prediction_DDOS_Attack_Type,
name="View_Prediction_DDOS_Attack_Type"),
    url(r'^Download_Trained_DataSets/$', serviceprovider.Download_Trained_DataSets,
name="Download_Trained_DataSets"),

]+ static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

```
from django.db.models import Count
from django.db.models import Q
from django.shortcuts import render, redirect, get_object_or_404
import datetime
import openpyxl

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
from sklearn.ensemble import VotingClassifier
from sklearn.tree import DecisionTreeClassifier
import warnings
warnings.filterwarnings("ignore")
plt.style.use('ggplot')
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score

# Create your views here.
from Remote_User.models import
ClientRegister_Model,ddos_attacks_prediction,detection_ratio,detection_accuracy

def login(request):


    if request.method == "POST" and 'submit1' in request.POST:

        username = request.POST.get('username')
        password = request.POST.get('password')
        try:
            enter = ClientRegister_Model.objects.get(username=username,password=password)
            request.session["userid"] = enter.id

            return redirect('ViewYourProfile')
        except:
            pass

    return render(request,'RUser/login.html')

def Register1(request):
    if request.method == "POST":
        username = request.POST.get('username')
        email = request.POST.get('email')
        password = request.POST.get('password')
        phoneno = request.POST.get('phoneno')
        country = request.POST.get('country')
        state = request.POST.get('state')
        city = request.POST.get('city')
        address = request.POST.get('address')
        gender = request.POST.get('gender')
        ClientRegister_Model.objects.create(username=username, email=email,
password=password, phoneno=phoneno,
                                            country=country, state=state, city=city, address=address,
gender=gender)
        obj = "Registered Successfully"
        return render(request, 'RUser/Register1.html', {'object': obj})
    else:
        return render(request,'RUser/Register1.html')

def ViewYourProfile(request):
    userid = request.session['userid']
    obj = ClientRegister_Model.objects.get(id= userid)
    return render(request,'RUser/ViewYourProfile.html',{'object':obj})
```

```
def predict_ddos_attack_type(request):
    if request.method == "POST":

        RID= request.POST.get('RID')
        Protocol= request.POST.get('Protocol')
        ip_src= request.POST.get('ip_src')
        ip_dst= request.POST.get('ip_dst')
        pro_srcport= request.POST.get('pro_srcport')
        pro_dstport= request.POST.get('pro_dstport')
        flags_ack= request.POST.get('flags_ack')
        ip_flags_mf= request.POST.get('ip_flags_mf')
        ip_flags_df= request.POST.get('ip_flags_df')
        ip_flags_rb= request.POST.get('ip_flags_rb')
        pro_seq= request.POST.get('pro_seq')
        pro_ack= request.POST.get('pro_ack')
        frame_time= request.POST.get('frame_time')
        Packets= request.POST.get('Packets')
        Bytes1= request.POST.get('Bytes1')
        Tx_Packets= request.POST.get('Tx_Packets')
        Tx_Bytes= request.POST.get('Tx_Bytes')
        Rx_Packets= request.POST.get('Rx_Packets')
        Rx_Bytes= request.POST.get('Rx_Bytes')




        df = pd.read_csv('Datasets.csv', encoding='latin-1')
        df
        df.columns

        def apply_results(results):
            if (results == "normal"):
                return 0
            elif (results == "smurf"):
                return 1
            elif (results == "Fraggile"):
                return 2

        df['Results'] = df['Label'].apply(apply_results)

        X = df['RID']
        y = df['Results']

        print("Reading ID")
        print(X)
        print("Label")
        print(y)

        # cv = CountVectorizer(lowercase=False, strip_accents='unicode', ngram_range=(1, 1))
        # X = cv.fit_transform(df['RID'].apply(lambda x: np.str_(x)))
        cv = CountVectorizer()
        X = cv.fit_transform(X)

        models = []
        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)
        X_train.shape, X_test.shape, y_train.shape

        print("Naive Bayes")

        from sklearn.naive_bayes import MultinomialNB
        NB = MultinomialNB()
        NB.fit(X_train, y_train)
        predict_nb = NB.predict(X_test)
        naivebayes = accuracy_score(y_test, predict_nb) * 100
        print(naivebayes)
        print(confusion_matrix(y_test, predict_nb))
        print(classification_report(y_test, predict_nb))
        models.append(('naive_bayes', NB))

        # SVM Model
        print("SVM")
        from sklearn import svm
        lin_clf = svm.LinearSVC()
        lin_clf.fit(X_train, y_train)
        predict_svm = lin_clf.predict(X_test)
        svm_acc = accuracy_score(y_test, predict_svm) * 100
        print(svm_acc)
        print("CLASSIFICATION REPORT")
        print(classification_report(y_test, predict_svm))
        print("CONFUSION MATRIX")
        print(confusion_matrix(y_test, predict_svm))
        models.append(('svm', lin_clf))

        print("Logistic Regression")

        from sklearn.linear_model import LogisticRegression
        reg = LogisticRegression(random_state=0, solver='lbfgs').fit(X_train, y_train)
        y_pred = reg.predict(X_test)
        print("ACCURACY")
        print(accuracy_score(y_test, y_pred) * 100)
        print("CLASSIFICATION REPORT")
        print(classification_report(y_test, y_pred))
        print("CONFUSION MATRIX")
        print(confusion_matrix(y_test, y_pred))
        models.append(('logistic', reg))



        classifier = VotingClassifier(models)
        classifier.fit(X_train, y_train)
```

```
y_pred = classifier.predict(X_test)

RID1 = [RID]
vector1 = cv.transform(RID1).toarray()
predict_text = classifier.predict(vector1)

pred = str(predict_text).replace("[", "")
pred1 = pred.replace("]", "")

prediction = int(pred1)

if prediction == 0:
    val = 'Normal'
elif prediction == 1:
    val = 'smurf'
elif prediction == 2:
    val = 'Fraggile'



print(val)
print(pred1)

ddos_attacks_prediction.objects.create(
RID=RID,
Protocol=Protocol,
ip_src=ip_src,
ip_dst=ip_dst,
pro_srcport=pro_srcport,
pro_dstport=pro_dstport,
flags_ack=flags_ack,
ip_flags_mf=ip_flags_mf,
ip_flags_df=ip_flags_df,
ip_flags_rb=ip_flags_rb,
    pro_seq=pro_seq,
    pro_ack=pro_ack,
    frame_time=frame_time,
    Packets=Packets,
    Bytes1=Bytes1,
    Tx_Packets=Tx_Packets,
    Tx_Bytes=Tx_Bytes,
    Rx_Packets=Rx_Packets,
    Rx_Bytes=Rx_Bytes,
   Prediction=val)

    return render(request, 'RUser/predict_ddos_attack_type.html',{'objs': val})
return render(request, 'RUser/predict_ddos_attack_type.html')
```