**SOURCE CODE**

**Main Views.py**

from django.shortcuts import render,HttpResponseRedirect

from django.contrib import messages

from .forms import CustomerRegistrationForm

def index(request):

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

def custRegistration(request):

if request.method=='POST':

form = CustomerRegistrationForm(request.POST)

if form.is\_valid():

print('Data is Valid')

form.save()

messages.success(request, 'You have been successfully registered')

#return HttpResponseRedirect('./CustLogin')

form = CustomerRegistrationForm()

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

else:

print("Invalid form")

else:

form = CustomerRegistrationForm()

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

def CustLogin(request):

return render(request,"CustomerLogin.html",{})

def CloudLogin(request):

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

def CspLogin(request):

return render(request,"CspLogin.html",{})

def Logout(request):

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

**urls.py**

"""PresagingTechnique URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:

https://docs.djangoproject.com/en/2.0/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.contrib import admin

from django.urls import path

from PresagingTechnique import views as mainview

from users import views as usr

from cloud import views as cloud

from csp import views as csp

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [

path('admin/', admin.site.urls),

path('',mainview.index,name='index'),

path('index',mainview.index,name='index'),

path('custRegistration/',mainview.custRegistration,name='custRegistration'),

path('CustLogin/',mainview.CustLogin,name='CustLogin'),

path('CloudLogin/',mainview.CloudLogin,name='CloudLogin'),

path('CspLogin/',mainview.CspLogin,name='CspLogin'),

path('Logout/',mainview.Logout,name='Logout'),

####Users URLS

path('UserLoginCheck/',usr.UserLoginCheck,name='UserLoginCheck'),

path('CustExploreService/',usr.CustExploreService,name='CustExploreService'),

path('CustUploadData/',usr.CustUploadData,name='CustUploadData'),

path('CustomerViewData/',usr.CustomerViewData,name='CustomerViewData'),

path('CustDownload/',usr.CustDownload,name='CustDownload'),

path('CustSuggestions/',usr.CustSuggestions,name='CustSuggestions'),

###Cloud All URL paths

path('CloudLoginCheck/',cloud.CloudLoginCheck,name='CloudLoginCheck'),

path('CloudCustomers/',cloud.CloudCustomers,name='CloudCustomers'),

path('CloudActivateUsers/',cloud.CloudActivateUsers,name='CloudActivateUsers'),

path('CloudCSPAdding/',cloud.CloudCSPAdding,name='CloudCSPAdding'),

path('CloudCreateCsp/',cloud.CloudCreateCsp,name='CloudCreateCsp'),

path('CloudDataView/',cloud.CloudDataView,name='CloudDataView'),

###CSP ALL URL Configurations

path('CSPLoginCheck/',csp.CSPLoginCheck,name='CSPLoginCheck'),

path('getCSPLoginDetails/',csp.getCSPLoginDetails,name='getCSPLoginDetails'),

path('GetCSPLoginData/',csp.GetCSPLoginData,name='GetCSPLoginData'),

path('CspDataView/',csp.CspDataView,name='CspDataView'),

path('CspViewSuggested/',csp.CspViewSuggested,name='CspViewSuggested'),

]

if settings.DEBUG:

urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

Customer **Views.py**

from django.shortcuts import render, HttpResponse, redirect

from PresagingTechnique.models import CustomerRegistrationModel

from django.contrib import messages

from .models import CustomerCloudData,KNNSuggestionModel

from django.core.files.storage import FileSystemStorage

from sklearn.neighbors import KNeighborsClassifier

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

from sklearn.datasets import load\_iris

from cloud.models import CspRegisterModel

from django.db.models import Sum, Count, Avg

import matplotlib.pyplot as plt

import numpy as np

import os

from django.conf import settings

from django.http import HttpResponse, Http404

from .KnnAlgorithm import KnnRecommender

from django.conf import settings

sujjdict = {}

# Create your views here.

def UserLoginCheck(request):

if request.method == "POST":

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

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

print("Login ID = ", loginid, ' Password = ', pswd)

try:

check = CustomerRegistrationModel.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/CustomerPage.html', {})

else:

messages.success(request, 'Your Account Not at activated')

return render(request, 'CustomerLogin.html')

# return render(request, 'user/userpage.html',{})

except Exception as e:

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

pass

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

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

def CustExploreService(request):

email = request.session['email']

# knnalgorithm()

global knnDict

results = CspRegisterModel.objects.filter().values('service').annotate(price=Avg('price')).order\_by("service")

# print(results,type(results))

dict = {}

list = []

for x in results:

# print(x['service'],"===",x['price'])

dict.update({x['service']: x['price']})

list.append([x['price']])

# print(dict)

dpnt = []

# print(list)

for d in list:

if d[0] < 500000:

dpnt.append(1)

else:

dpnt.append(0)

# print(dpnt)

test = [[560000], [890000], [589444], [695870], [458962], [698745], [365500]]

# test = [[1], [0], [1], [1], [1], [0], [0]]

knn = KNeighborsClassifier(n\_neighbors=1)

# x = np.reshape(list,(-1,1)).T

knn.fit(list, dpnt)

rs = knn.predict(test)

#for rslt in rs:

# print(rslt)

custname = request.session['loginid']

# hstr = CustomerCloudData.objects.filter(custname=custname).values('servicename').distinct()

hstr = CustomerCloudData.objects.filter(custname=custname).values('servicename').distinct().annotate(

count=Count('servicename')).order\_by("servicename")

#print("Hello ", hstr)

hisDict = {}

mylist = []

for x in hstr:

hisDict.update({x['servicename']: x['count']})

#knnDict = {}

if len(hisDict) != 0:

for keys, values in hisDict.items():

print(keys, "==", values)

knnDict = knnalgorithm(keys)

for kk,vv in knnDict.items():

#print(keys,"==",kk,"==",vv)

KNNSuggestionModel.objects.create(username=custname,email=email,servicename=keys,knnsuggestions=kk,distance=vv)

#Store Suggestions In database

mylist.append(knnDict)

# if knnSu is not None:

# knnDict.update({knnSu:distVal})

#print('Ram=',knnDict)

#print('Alex List=', mylist)

for list in mylist:

print('List Dict ',list)

for xy,yz in list.items():

knnDict.update({xy:yz})

# knnalgorithm('Microsoft Office')

return render(request, 'users/ServiceExplore.html', {'dict': dict, 'hisDict': hisDict,'knndict':knnDict})

def CustUploadData(request):

if request.method == 'POST':

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

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

servicename = request.POST.get('service')

# filename = request.POST.get('file')

custname = request.session['loginid']

email = request.session['email']

myfile = request.FILES['file']

fs = FileSystemStorage()

filename = fs.save(myfile.name, myfile)

uploaded\_file\_url = fs.url(filename)

CustomerCloudData.objects.create(custname=custname, datatype=datatype, email=email, location=location,

servicename=servicename, filename=filename, file=uploaded\_file\_url)

messages.success(request, 'Data uploaded to Cloud')

return redirect('CustExploreService')

def CustomerViewData(request):

custname = request.session['loginid']

data = CustomerCloudData.objects.filter(custname=custname)

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

def CustSuggestions(request):

custname = request.session['loginid']

data = KNNSuggestionModel.objects.filter(username=custname)

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

def CustDownload(request):

path = request.GET.get('uid')

file\_path = os.path.join("/", path)

if os.path.exists(file\_path):

with open(file\_path, 'rb') as fh:

print('Path ', path)

response = HttpResponse(fh.read(), content\_type="application/vnd.ms-excel")

response['Content-Disposition'] = 'inline; filename=' + os.path.basename(file\_path)

return response

raise Http404

def knnalgorithm(movie\_name):

# get args

# args = parse\_args()

# data\_path = args.path

movies\_filename = settings.MEDIA\_ROOT + "\\" + 'movies.csv'

ratings\_filename = settings.MEDIA\_ROOT + "\\" + 'ratings.csv'

movie\_name = movie\_name #'Microsoft Office'

top\_n = 1 # args.top\_n

# initial recommender system

recommender = KnnRecommender(

os.path.join('', movies\_filename),

os.path.join('', ratings\_filename))

# set params

recommender.set\_filter\_params(50, 50)

recommender.set\_model\_params(20, 'brute', 'cosine', -1)

# make recommendations

rslt = recommender.make\_recommendations(movie\_name, top\_n)

#print('X value is =', x)

#print('Y value is =', y)

return rslt

**KnnAlgorithm.py**

**import** os  
**import** time  
**import** gc  
**import** argparse  
  
*# data science imports***import** pandas **as** pd  
**from** scipy.sparse **import** csr\_matrix  
**from** sklearn.neighbors **import** NearestNeighbors  
  
*# utils import***from** fuzzywuzzy **import** fuzz  
  
  
**class** KnnRecommender:  
 rsdict = {}  
  
 **def** \_\_init\_\_(self, path\_services, path\_ratings):  
  
 self.path\_services = path\_services  
 self.path\_ratings = path\_ratings  
 self.movie\_rating\_thres = 0  
 self.user\_rating\_thres = 0  
 self.model = NearestNeighbors()  
  
 **def** set\_filter\_params(self, movie\_rating\_thres, user\_rating\_thres):  
  
 self.movie\_rating\_thres = movie\_rating\_thres  
 self.user\_rating\_thres = user\_rating\_thres  
  
 **def** set\_model\_params(self, n\_neighbors, algorithm, metric, n\_jobs=**None**):  
  
 **if** n\_jobs **and** (n\_jobs > 1 **or** n\_jobs == -1):  
 os.environ[**'JOBLIB\_TEMP\_FOLDER'**] = **'/tmp'** self.model.set\_params(\*\*{  
 **'n\_neighbors'**: n\_neighbors,  
 **'algorithm'**: algorithm,  
 **'metric'**: metric,  
 **'n\_jobs'**: n\_jobs})  
  
 **def** \_prep\_data(self):  
 *# read data* df\_movies = pd.read\_csv(  
 os.path.join(self.path\_services),  
 usecols=[**'serviceId'**, **'servicename'**],  
 dtype={**'serviceId'**: **'int32'**, **'servicename'**: **'str'**})  
 df\_ratings = pd.read\_csv(  
 os.path.join(self.path\_ratings),  
 usecols=[**'userId'**, **'serviceId'**, **'rating'**],  
 dtype={**'userId'**: **'int32'**, **'serviceId'**: **'int32'**, **'rating'**: **'float32'**})  
 *# filter data* df\_movies\_cnt = pd.DataFrame(  
 df\_ratings.groupby(**'serviceId'**).size(),  
 columns=[**'count'**])  
 popular\_movies = list(set(df\_movies\_cnt.query(**'count >= @self.movie\_rating\_thres'**).index)) *# noqa* movies\_filter = df\_ratings.serviceId.isin(popular\_movies).values  
  
 df\_users\_cnt = pd.DataFrame(  
 df\_ratings.groupby(**'userId'**).size(),  
 columns=[**'count'**])  
 active\_users = list(set(df\_users\_cnt.query(**'count >= @self.user\_rating\_thres'**).index)) *# noqa* users\_filter = df\_ratings.userId.isin(active\_users).values  
  
 df\_ratings\_filtered = df\_ratings[movies\_filter & users\_filter]  
  
 *# pivot and create movie-user matrix* movie\_user\_mat = df\_ratings\_filtered.pivot(  
 index=**'serviceId'**, columns=**'userId'**, values=**'rating'**).fillna(0)  
 *# create mapper from movie servicename to index* hashmap = {  
 movie: i **for** i, movie **in** enumerate(list(df\_movies.set\_index(**'serviceId'**).loc[movie\_user\_mat.index].servicename)) *# noqa* }  
 *# transform matrix to scipy sparse matrix* movie\_user\_mat\_sparse = csr\_matrix(movie\_user\_mat.values)  
  
 *# clean up* **del** df\_movies, df\_movies\_cnt, df\_users\_cnt  
 **del** df\_ratings, df\_ratings\_filtered, movie\_user\_mat  
 gc.collect()  
 **return** movie\_user\_mat\_sparse, hashmap  
  
 **def** \_fuzzy\_matching(self, hashmap, fav\_movie):  
 match\_tuple = []  
 *# get match* **for** servicename, idx **in** hashmap.items():  
 ratio = fuzz.ratio(servicename.lower(), fav\_movie.lower())  
 **if** ratio >= 60:  
 match\_tuple.append((servicename, idx, ratio))  
 *# sort* match\_tuple = sorted(match\_tuple, key=**lambda** x: x[2])[::-1]  
 **if not** match\_tuple:  
 print(**'Oops! No match is found'**)  
 **return** 1  
 **else**:  
 print(**'Found possible matches in our database: {0}\n'**.format([x[0] **for** x **in** match\_tuple]))  
 *#print('This man ',match\_tuple[0][1])* **return** match\_tuple[0][1]  
  
 **def** \_inference(self, model, data, hashmap,fav\_movie, n\_recommendations):  
  
 *# fit* model.fit(data)  
 *# get input movie index* print(**'You have input movie:'**, fav\_movie)  
 idx = self.\_fuzzy\_matching(hashmap, fav\_movie)  
 *#print('Result is ', idx)* **if** idx == **None**:  
 idx = 1  
 *# inference* print(**'Recommendation system start to make inference'**)  
 print(**'......\n'**)  
 t0 = time.time()  
 distances, indices = model.kneighbors(data[idx],n\_neighbors=n\_recommendations + 1)  
 *# get list of raw idx of recommendations* raw\_recommends = \  
 sorted(  
 list(  
 zip(  
 indices.squeeze().tolist(),  
 distances.squeeze().tolist()  
 )  
 ),  
 key=**lambda** x: x[1]  
 )[:0:-1]  
 print(**'It took my system {:.2f}s to make inference \n\  
 '**.format(time.time() - t0))  
 *# return recommendation (serviceId, distance)* **return** raw\_recommends  
  
 **def** make\_recommendations(self, fav\_movie, n\_recommendations):  
 rsdict={}  
 *# get data* movie\_user\_mat\_sparse, hashmap = self.\_prep\_data()  
 *# get recommendations* raw\_recommends = self.\_inference(self.model, movie\_user\_mat\_sparse, hashmap,fav\_movie, n\_recommendations)  
 *# print results* reverse\_hashmap = {v: k **for** k, v **in** hashmap.items()}  
 *#print('Recommendations for {}:'.format(fav\_movie))* **for** i, (idx, dist) **in** enumerate(raw\_recommends):  
 *#print('{0}: {1}, with distance of {2}'.format(i + 1, reverse\_hashmap[idx], dist))  
 #return reverse\_hashmap[idx],dist* rsdict.update({reverse\_hashmap[idx]:dist})  
 **return** rsdict

**users Models.py**

**from** django.db **import** models  
  
*# Create your models here.***class** CustomerCloudData(models.Model):  
 custname = models.CharField(max\_length=100)  
 datatype = models.CharField(max\_length=100)  
 email = models.CharField(max\_length=100)  
 location = models.CharField(max\_length=100)  
 servicename = models.CharField(max\_length=100)  
 filename = models.CharField(max\_length=100)  
 file = models.FileField(upload\_to=**'files/'**)  
 cdate = models.DateTimeField(auto\_now\_add=**True**)  
  
 **def** \_\_str\_\_(self):  
 **return** self.id  
 **class** Meta:  
 db\_table=**'CustomerFiles'  
  
  
class** KNNSuggestionModel(models.Model):  
 username = models.CharField(max\_length=100)  
 email = models.CharField(max\_length=100)  
 servicename = models.CharField(max\_length=100)  
 knnsuggestions = models.CharField(max\_length=100)  
 distance = models.CharField(max\_length=100)  
 cdate = models.DateTimeField(auto\_now\_add=**True**)  
 **def** \_\_str\_\_(self):  
 **return** self.id  
 **class** Meta:  
 db\_table=**'KnnSuggestions'**

Cloud **Views.py**

**from** django.shortcuts **import** render,HttpResponse,redirect  
**from** django.contrib **import** messages  
**from** PresagingTechnique.models **import** CustomerRegistrationModel  
**from** faker **import** Faker  
**from** .models **import** CspRegisterModel  
**from** users.models **import** CustomerCloudData  
*# Create your views here.***def** CloudLoginCheck(request):  
 **if** request.method == **'POST'**:  
 usrid = request.POST.get(**'loginname'**)  
 pswd = request.POST.get(**'pswd'**)  
 print(**"User ID is = "**, usrid)  
 **if** usrid == **'admin' and** pswd == **'admin'**:  
 **return** render(request, **'clouds/CloudHome.html'**)  
 **elif** usrid == **'cloud' and** pswd == **'cloud'**:  
 **return** render(request, **'clouds/CloudHome.html'**)  
 **else**:  
 messages.success(request, **'Please Check Your Login Details'**)  
 **return** render(request, **'CloudLogin.html'**)  
  
  
**def** CloudCustomers(request):  
 cust = CustomerRegistrationModel.objects.all()  
 **return** render(request,**'clouds/CloudCust.html'**,{**'cust'**:cust})  
  
**def** CloudActivateUsers(request):  
 **if** request.method == **'GET'**:  
 id = request.GET.get(**'uid'**)  
 status = **'activated'** print(**"PID = "**, id, status)  
 CustomerRegistrationModel.objects.filter(id=id).update(status=status)  
 cust = CustomerRegistrationModel.objects.all()  
 **return** render(request,**'clouds/CloudCust.html'**,{**'cust'**:cust})  
**def** CloudCSPAdding(request):  
 csp = CspRegisterModel.objects.all()  
 **return** render(request,**'clouds/CloudCSP.html'**,{**'csp'**:csp})  
  
**def** CloudCreateCsp(request):  
 **if** request.method==**'POST'**:  
 cspname = request.POST.get(**'cspname'**)  
 loginid = request.POST.get(**'loginid'**)  
 mobile = request.POST.get(**'mobile'**)  
 email = request.POST.get(**'email'**)  
 location = request.POST.get(**'location'**)  
 service = request.POST.get(**'service'**)  
 faker = Faker()  
 password = faker.password()  
 price = faker.random\_int(50000, 1000000)  
 **try**:  
 CspRegisterModel.objects.create(name=cspname,loginid=loginid,password=password,mobile=mobile,email=email,locality=location,service=service,price=price)  
 s = **'CSP Created Success for '**+service  
 messages.success(request, s)  
 **except** Exception **as** ex:  
 print(**'Error Message '**,str(ex))  
 ermsg = **'CSP with mobile '**+mobile+ **' and email '**+email+**' already exist'** messages.success(request, ermsg)  
 **return** render(request, **'clouds/CloudCSP.html'**, {})  
  
 **return** redirect(**'CloudCSPAdding'**)  
  
  
**def** CloudDataView(request):  
 data = CustomerCloudData.objects.all()  
 **return** render(request, **'clouds/CloudDataView.html'**, {**'data'**: data})

csp **Views.py**

**from** django.shortcuts **import** render,HttpResponse  
**from** cloud.models **import** CspRegisterModel  
**from** django.contrib **import** messages  
**from** users.models **import** CustomerCloudData,KNNSuggestionModel  
  
*# Create your views here.***def** CSPLoginCheck(request):  
 **if** request.method==**'POST'**:  
 loginname = request.POST.get(**'loginname'**)  
 pswd = request.POST.get(**'pswd'**)  
 **try**:  
 check = CspRegisterModel.objects.get(loginid=loginname, password=pswd)  
 request.session[**'cspname'**]=check.name  
 request.session[**'csploginname'**]=check.loginid  
 request.session[**'service'**]=check.service  
 **return** render(request,**'csps/CspHome.html'**,{})  
  
 **except** Exception **as** ex:  
 messages.success(request, **'Please Check Your Login Details'**)  
 print(**'Invalid Login Details'**)  
 print(str(ex))  
  
 **return** HttpResponse(**'Works CSP great'**)  
  
**def** getCSPLoginDetails(request):  
 **return** render(request,**'GetCSPLoginDetails.html'**,{})  
  
**def** GetCSPLoginData(request):  
 **if** request.method == **'POST'**:  
 email = request.POST.get(**'email'**)  
 mobile = request.POST.get(**'mobile'**)  
 check = CspRegisterModel.objects.get(mobile=mobile,email=email)  
 logindata = check.loginid  
 pswd = check.password  
 print(**'Login ID '**,logindata,**' Password '**,pswd)  
 **return** render(request,**'CspLogin.html'**,{**'loginid'**:logindata,**'pswd'**:pswd})  
  
 messages.success(request, **'Login Details Not Found please approach Administrations'**)  
 **return** render(request,**'CspLogin.html'**,{})  
  
**def** CspDataView(request):  
 csploginid = request.session[**'csploginname'**]  
 cspservicename = request.session[**'service'**]  
 data = CustomerCloudData.objects.filter(servicename=cspservicename)  
 **return** render(request, **'csps/CspDataView.html'**, {**'data'**: data})  
  
**def** CspViewSuggested(request):  
 cspservicename = request.session[**'service'**]  
 data = KNNSuggestionModel.objects.filter(servicename=cspservicename)  
 **return** render(request, **'csps/CspSuggestions.html'**, {**'data'**: data})

**base.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 http-equiv="Content-Type" content="text/html; charset=utf-8"** />  
<**link rel="icon" href="{% static 'images/favicon.png'%}" type="image/png"**>  
<**title**></**title**>  
<**meta name="keywords" content=""** />  
<**meta name="description" content=""** />  
<**link href="http://fonts.googleapis.com/css?family=Source+Sans+Pro:200,300,400,600,700,900" rel="stylesheet"** />  
<**link href="{%static 'default.css'%}" rel="stylesheet" type="text/css" media="all"** />  
<**link href="{%static 'fonts.css'%}" rel="stylesheet" type="text/css" media="all"** />  
<**script type="text/javascript"**>  
 window.history.forward();  
 function noBack(){  
 window.history.forward();  
 }  
</**script**>  
</**head**>  
<**body onload="noBack();"onPageShow="if(event.persisted) noBack();" onUnload=""** >  
<**div id="logo"**>  
 <**h1**><**a href="#" class="icon icon-group"**><**span**>Machine Learning based Presaging Technique</**span**></**a**></**h1**>  
</**div**>  
<**div id="header"**>  
 <**div id="menu" class="container"**>  
 <**ul**>  
 <**li**><**a href="{%url 'index'%}" accesskey="1" title=""**>Homepage</**a**></**li**>  
 <**li**><**a href="{%url 'CustLogin'%}" accesskey="1" title=""**>Customer</**a**></**li**>  
 <**li**><**a href="{%url 'CspLogin'%}" accesskey="2" title=""**>CSP</**a**></**li**>  
 <**li**><**a href="{%url 'CloudLogin'%}" accesskey="3" title=""**>Cloud Admin</**a**></**li**>  
 <**li**><**a href="{%url 'custRegistration'%}" accesskey="4" title=""**>Registrations</**a**></**li**>  
  
 </**ul**>  
 </**div**>  
</**div**>  
{%block contents%}  
  
{%endblock%}  
<**div id="copyright" class="container"**>  
 <**p**>**&copy;** Alex and group of Corporations</**p**>  
</**div**>  
</**body**>  
</**html**>

**ServiceExplorer.html**

{% extends 'users/custbase.html'%}  
{%load static%}  
{%block contents%}  
  
<**div id="page-wrapper"**>  
<**div id="page" class="container"**>  
 <**div id="content"**>  
 <**div class="title"**>  
 <**form method="post" action="{%url 'CustUploadData'%}" enctype="multipart/form-data"**>  
 {%csrf\_token%}  
  
 <**table class="table table-bordered"**>  
 <**tr**>  
 <**th**>Fileds</**th**><**th**>Data</**th**>  
 </**tr**>  
 <**tr**><**td**>Data Type</**td**><**td**><**input type="text" name="datatype" pattern='[a-zA-Z]+' required="True"**> </**td**></**tr**>  
 <**tr**><**td**>Location</**td**><**td**><**input type="text" name="location" pattern='[a-zA-Z]+' required="True"**> </**td**></**tr**>  
 <**tr**> <**td**>Select Service</**td**><**td**>  
 <**select name="service" required="True"**>  
 <**option value=""**>---Select Cloud Service---</**option**>  
 <**option value="Microsoft Azure"**>Microsoft Azure </**option**>  
 <**option value="Amazon Web Services (AWS)"**>Amazon Web Services (AWS) </**option**>  
 <**option value="Cisco Metacloud"**>Cisco Metacloud </**option**>  
 <**option value="Google Compute Engine (GCE)"**>Google Compute Engine (GCE) </**option**>  
 <**option value="AWS Elastic Beanstalk"**>AWS Elastic Beanstalk </**option**>  
 <**option value="Apache Stratos"**>Apache Stratos </**option**>  
 <**option value="Google App Engine"**>Google App Engine </**option**>  
 <**option value="Microsoft Office 365"**>Microsoft Office 365 </**option**>  
 <**option value="Salesforce"**>Salesforce </**option**>  
 <**option value="Cisco WebEx"**>Cisco WebEx </**option**>  
 <**option value="Google Apps"**>Google Apps </**option**>  
  
 </**select**>  
 </**td**>  
 </**tr**>  
 <**tr**>  
 <**td**>Select File</**td**><**td**><**input type="file" name="file" required="True"**></**td**>  
 </**tr**>  
 <**tr**><**td**></**td**><**td**><**button type="submit" class="btn-primary" name="Submit"**>Submit</**button**> </**td**></**tr**>  
 </**table**>  
 </**form**>  
 <**center**>  
 {% if messages %}  
 {% for message in messages %}  
 <**font color='Green'**> {{ message }}</**font**>  
 {% endfor %}  
 {% endif %}  
 </**center**>  
 </**div**>  
 <**p**></**p**>  
  
 </**div**>  
 <**p**>Plans Here Suggested us  
 <**table**>  
 <**tr**>  
<**th**>Service Name</**th**>  
 <**th**>Expcted Price</**th**>  
 </**tr**>  
 {%for key,values in dict.items%}  
 <**tr**>  
 <**td**>{{key}}</**td**>  
 <**td**>${{values}}</**td**>  
 </**tr**>  
 {%endfor%}  
</**table**>  
 </**p**>  
 <**hr**>  
 <**p**>Suggested based on your History  
 <**table**>  
 <**tr**>  
 <**th**>Service Name</**th**>  
 <**th**>No of times used</**th**>  
 </**tr**>  
 {%for key,values in hisDict.items%}  
 <**tr**>  
 <**td**>{{key}}</**td**>  
 <**td**>{{values}}</**td**>  
 </**tr**>  
 {%endfor%}  
</**table**>  
  
  
<**em**>=============================================</**em**>  
 <**h2**>Suggestions based On Knn Algorithm</**h2**>  
 <**table**>  
 <**tr**>  
 <**th**>Service Name</**th**>  
 <**th**>Distance</**th**>  
 </**tr**>  
  
 {%for key,values in knndict.items%}  
 <**tr**>  
 <**td**>{{key}}</**td**>  
 <**td**>{{values}}</**td**>  
 </**tr**>  
 {%endfor%}  
</**table**>  
 <**stron** >Note:</**stron**><**p**> Here we took some sample dataset which is associated with movies and ther ranks, we established the connection to cloud services with the movies dataset </**p**>  
 </**p**>  
  
 <**div id="sidebar"**><**a href="#" class="image image-full"**><**img src="{%static 'images/cspservice.jpg'%}" alt=""** /></**a**></**div**>  
</**div**>  
</**div**>  
{%endblock%}