## 6.3 SOURCE CODE:

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv) from datetime import datetime

import seaborn as sns

from sklearn.metrics import confusion\_matrix sns.set\_style('darkgrid')

# Input data files are available in the "../input/" directory.

# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

# Any results you write to the current directory are saved as output. df\_codes = pd.read\_csv('offense\_codes.csv', encoding='ISO-8859-1') df\_codes.head()

df = pd.read\_csv('crime.csv', encoding='ISO-8859-1') df.head()

df.isnull().sum()

df.drop(['DISTRICT', 'SHOOTING', 'UCR\_PART', 'STREET', 'Lat', 'Long'], axis=1,

inplace=True) sorted(df['REPORTING\_AREA'].unique())[:10] ## replace empty reporting areas with '-1'

df['REPORTING\_AREA'] = df['REPORTING\_AREA'].str.replace(' ', '-1')

sorted(df['REPORTING\_AREA'].unique()) df['REPORTING\_AREA'] = df['REPORTING\_AREA'].astype(int)

# code day of week to ints

df['OCCURRED\_ON\_DATE'] = pd.to\_datetime(df['OCCURRED\_ON\_DATE'])

df['DAY\_OF\_WEEK'] = df['OCCURRED\_ON\_DATE'].dt.dayofweek

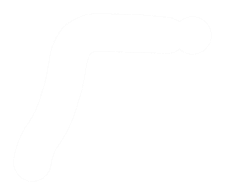
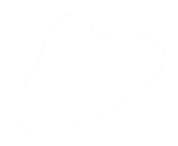
df['OFFENSE\_CODE\_GROUP'].value\_counts().plot(kind='bar', figsize=(20,5), title='Offense Code Group Counts')

df\_new = df.copy(deep=True)

df\_new['MV'] = np.where(df\_new['OFFENSE\_CODE\_GROUP'] == 'Motor Vehicle Accident Response', 1, 0)

df\_new.head()

file.close()



contract = web3.eth.contract(address=deployed\_contract\_address, abi=contract\_abi) getContract()

def getUsersList():

global usersList, contract usersList = []

count = contract.functions.getUserCount().call() for i in range(0, count):

user = contract.functions.getUsername(i).call() password = contract.functions.getPassword(i).call() email = contract.functions.getEmail(i).call() usersList.append([user, password, email])

def getPartyList():

global partyList, contract partyList = []

count = contract.functions.getPartyCount().call() for i in range(0, count):

cname = contract.functions.getCandidateName(i).call() pname = contract.functions.getPartyName(i).call() area = contract.functions.getArea(i).call()

symbol = contract.functions.getSymbol(i).call() partyList.append([cname, pname, area, symbol])

def getVoteList():

global voteList, contract voteList = []

count = contract.functions.getVotingCount().call() for i in range(0, count):

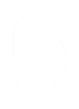
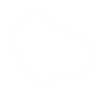
user = contract.functions.getUser(i).call() party = contract.functions.getParty(i).call() dd = contract.functions.getDate(i).call()

candidate = contract.functions.getCandidate(i).call() voteList.append([user, party, dd, candidate])



def loadModel():

global names, encodings



if os.path.exists("model/encoding.npy"):

encodings = np.load("model/encoding.npy") names = np.load("model/names.npy")

else:

encodings = [] names = []

getUsersList() getPartyList() getVoteList() loadModel()

face\_detection = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

def alreadyCastVote(candidate): global voteList

count = 0

for i in range(len(voteList)):

vl = voteList[i]

if vl[3] == candidate: count = 1

return count

def FinishVote(request):

if request.method == 'GET': global username, voteList

cname = request.GET.get('cname', False) pname = request.GET.get('pname', False) voter = ''

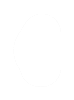
today = date.today()

status = 'Your vote casted to '+cname

msg = contract.functions.createVote(username, pname, str(today), cname).transact() web3.eth.waitForTransactionReceipt(msg)

voteList.append([username, pname, str(today), cname])

context= {'data':'<font size=3 color=black>Your Vote Accepted for Candidate '+cname} return render(request, 'UserScreen.html', context)



def getOutput(status): global partyList

output = '<h3><br/>'+status+'<br/><table border=1 align=center>' output+='<tr><th><font size=3 color=black>Candidate Name</font></th>' output+='<th><font size=3 color=black>Party Name</font></th>' output+='<th><font size=3 color=black>Area Name</font></th>' output+='<th><font size=3 color=black>Image</font></th>' output+='<th><font size=3 color=black>Cast Vote Here</font></th></tr>' for i in range(len(partyList)):

pl = partyList[i]

output+='<tr><td><font size=3 color=black>'+pl[0]+'</font></td>' output+='<td><font size=3 color=black>'+pl[1]+'</font></td>' output+='<td><font size=3 color=black>'+pl[2]+'</font></td>'

output+='<td><img src="/static/parties/'+pl[3]+'" width=200 height=200></img></td>' output+='<td><a href="FinishVote?cname='+pl[0]+'&pname='+pl[1]+'"><font size=3

color=black>Click Here</font></a></td></tr>' output+="</table><br/><br/><br/><br/><br/><br/>" return output

def ValidateUser(request):

if request.method == 'POST':

global username, encodings, names predict = "none"

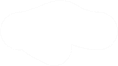
page = "UserScreen.html" status = "unable to predict user"

img = cv2.imread('VotingApp/static/photo/test.png') gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) face\_component = None

faces=face\_detection.detectMultiScale(img,scaleFactor=1.1,minNeighbors=5,minSize=(3 0,30),flags=cv2.CASCADE\_SCALE\_IMAGE)

status = "Unable to predict.Please retry"

faces = sorted(faces, reverse=True,key=lambda x: (x[2] - x[0]) \* (x[3] - x[1]))[0] (fX, fY, fW, fH) = faces

face\_component = gray[fY:fY + fH, fX:fX + fW] if face\_component is not None:

img = cv2.resize(img, (600, 600))

rgb\_small\_frame = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB) # Convert the frame to RGB color space

face\_locations = face\_recognition.face\_locations(rgb\_small\_frame) # Locate faces in the frame

face\_encodings = face\_recognition.face\_encodings(rgb\_small\_frame, face\_locations) # Encode faces in the frame

for face\_encoding in face\_encodings:

matches = face\_recognition.compare\_faces(encodings, face\_encoding) # Compare face encodings

face\_distance = face\_recognition.face\_distance(encodings, face\_encoding) # Calculate face distance

best\_match\_index = np.argmin(face\_distance) # Get the index of the best match print(best\_match\_index)

if matches[best\_match\_index]: # If the face is a match

name = names[best\_match\_index] # Get the corresponding name predict = name

break

if predict == username:

count = alreadyCastVote(username) if count == 0:

page = 'VotePage.html'

status = getOutput("User predicted as : "+predict+"<br/><br/>") else:

status = "You already casted your vote" page = "UserScreen.html"

else:

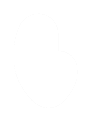
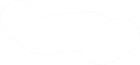
page = "UserScreen.html" status = "unable to predict user"

context= {'data':status}

20

return render(request, page, context) def UserLogin(request):

if request.method == 'POST':



global username, contract, usersList

username = request.POST.get('username', False) password = request.POST.get('password', False) status = "Login.html"

output = 'Invalid login details' for i in range(len(usersList)):

ulist = usersList[i] user1 = ulist[0] pass1 = ulist[1]

if user1 == username and pass1 == password: status = "UserScreen.html"

output = 'Welcome '+username break

context= {'data':output}

return render(request, status, context)

def AdminLogin(request):

if request.method == 'POST': global username

username = request.POST.get('username', False) password = request.POST.get('password', False) if username == 'admin' and password == 'admin':

context= {'data':'Welcome '+username}

return render(request, 'AdminScreen.html', context) if status == 'none':

context= {'data':'Invalid login details'} return render(request, 'Admin.html', context)

def AddParty(request):

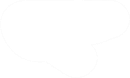
if request.method == 'GET':

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

def index(request):

if request.method == 'GET':

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

def Login(request):

if request.method == 'GET':

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

def CastVote(request):

if request.method == 'GET':

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

def AddVoter(request):

if request.method == 'GET':

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

def Admin(request):

if request.method == 'GET':

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

def AddVoterAction(request):

if request.method == 'POST':

global username, password, contact, email, address, usersList username = request.POST.get('username', False)

password = request.POST.get('password', False) contact = request.POST.get('contact', False) email = request.POST.get('email', False) address = request.POST.get('address', False) status = "none"

for i in range(len(usersList)): ul = usersList[i]

if username == ul[0]: status = "exists" break

if status == "none":

context= {'data':'Capture Your face'}

20

return render(request, 'CaptureFace.html', context) else:

context= {'data':username+' Username already exists'} return render(request, 'AddVoter.html', context)

def WebCam(request):

if request.method == 'GET': data = str(request)

formats, imgstr = data.split(';base64,') imgstr = imgstr[0:(len(imgstr)-2)] data = base64.b64decode(imgstr)

if os.path.exists("VotingApp/static/photo/test.png"): os.remove("VotingApp/static/photo/test.png")

with open('VotingApp/static/photo/test.png', 'wb') as f: f.write(data)

f.close()

context= {'data':"done"}

return HttpResponse("Image saved")

def saveFace():

global names, encodings encodings = np.asarray(encodings) names = np.asarray(names)

np.save("model/encoding", encodings) np.save("model/names", names)

if request.method == 'POST':

global username, password, contact, email, address, usersList, encodings, names img = cv2.imread('VotingApp/static/photo/test.png')

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) face\_component = None

faces = face\_detection.detectMultiScale(gray, 1.3,5) page = "AddVoter.html"

status = 'Unable to detect face. Please retry'

20

for (x, y, w, h) in faces:

face\_component = img[y:y+h, x:x+w] if face\_component is not None:

img = cv2.resize(img, (600, 600))

if os.path.exists("VotingApp/static/photo/test.png"): os.remove("VotingApp/static/photo/test.png")

cv2.imwrite("VotingApp/static/photo/test.png", img)

image = face\_recognition.load\_image\_file("VotingApp/static/photo/test.png") encoding = face\_recognition.face\_encodings(image)

print("encoding "+str(encoding))

if len(encoding) > 0 and username not in names: encoding = encoding[0]

if len(encodings) == 0: encodings.append(encoding) names.append(username)

else:

encodings = encodings.tolist() names = names.tolist() encodings.append(encoding) names.append(username)

saveFace()

page = "AddVoter.html"

status = 'User with Face Details added to Blockchain<br/><br/>'

msg = contract.functions.createUser(username, email, password, contact, address)

.transact()

status += str(web3.eth.waitForTransactionReceipt(msg)) usersList.append([username, password, email])

context= {'data': status}

return render(request, page, context)

def AddPartyAction(request):

if request.method == 'POST': global partyList

cname = request.POST.get('t1', False) pname = request.POST.get('t2', False)

20

area = request.POST.get('t3', False) myfile = request.FILES['t4'] imagename = request.FILES['t4'].name status = "none"

page = "AddParty.html"

for i in range(len(partyList)): pl = partyList[i]

if cname == pl[0] and pname == pl[1]:

status = "Candidate & Party Name Already Exists" break

if status == "none":

fs = FileSystemStorage()

filename = fs.save('VotingApp/static/parties/'+imagename, myfile) status = 'Candidate details added to Blockchain<br/><br/>'

msg = contract.functions.createParty(cname, pname, area, imagename).transact() status += str(web3.eth.waitForTransactionReceipt(msg)) partyList.append([cname, pname, area, imagename])

context= {'data': status}

return render(request, page, context)

def getVoteCount(cname, pname): global voteList

count = 0

for i in range(len(voteList)): vl = voteList[i]

if vl[1] == pname and vl[3] == cname: count += 1

return count

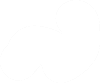
def ViewVotes(request):

if request.method == 'GET':

output = '<table border=1 align=center>'

output+='<tr><th><font size=3 color=black>Candidate Name</font></th>' output+='<th><font size=3 color=black>Party Name</font></th>'

20

output+='<th><font size=3 color=black>Area Name</font></th>' output+='<th><font size=3 color=black>Image</font></th>' output+='<th><font size=3 color=black>Vote Count</font></th>' for i in range(len(partyList)):

pl = partyList[i]

count = getVoteCount(pl[0], pl[1])

output+='<tr><td><font size=3 color=black>'+pl[0]+'</font></td>' output+='<td><font size=3 color=black>'+pl[1]+'</font></td>' output+='<td><font size=3 color=black>'+pl[2]+'</font></td>' output+='<td><img src="/static/parties/'+pl[3]+'"

width=200 height=200></img></td>'

output+='<td><font size=3 color=black>'+str(count)+'</font></td></tr>' output+="</table><br/><br/><br/><br/><br/><br/>"

context= {'data':output}

return render(request, 'ViewVotes.html', context)

def ViewParty(request):

if request.method == 'GET':

output = '<table border=1 align=center>'

output+='<tr><th><font size=3 color=black>Candidate Name</font></th>' output+='<th><font size=3 color=black>Party Name</font></th>' output+='<th><font size=3 color=black>Area Name</font></th>' output+='<th><font size=3 color=black>Image</font></th></tr>'

for i in range(len(partyList)): pl = partyList[i]

output+='<tr><td><font size=3 color=black>'+pl[0]+'</font></td>' output+='<td><font size=3 color=black>'+pl[1]+'</font></td>' output+='<td><font size=3 color=black>'+pl[2]+'</font></td>' output+='<td><img src="/static/parties/'+pl[3]+'" width=200 height=200></img></td></tr>'

output+="</table><br/><br/><br/><br/><br/><br/>" context= {'data':output}

return render(request, 'ViewParty.html', context)

20

