# main.py

import os

import base64

import io

import math

from flask import Flask, render\_template, Response, redirect, request, session, abort, url\_for

from flask\_mail import Mail, Message

from flask import send\_file

import mysql.connector

import hashlib

import datetime

from datetime import datetime

from datetime import date

import calendar

import random

from random import randint

from urllib.request import urlopen

import webbrowser

#from plotly import graph\_objects as go

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

from werkzeug.utils import secure\_filename

import urllib.request

import urllib.parse

import socket

import csv

import matplotlib as mpl

import seaborn as sns

from matplotlib import pyplot as plt

from collections import OrderedDict

import re # for regular expressions

#import nltk # for text manipulation

import string # for text manipulation

import warnings

#from nltk.stem.porter import \*

#from nltk.corpus import stopwords

#nltk.download()

#from nltk.stem import PorterStemmer

#from nltk.tokenize import sent\_tokenize, word\_tokenize

#from nltk.corpus import stopwords

###

import gensim

from gensim.parsing.preprocessing import remove\_stopwords, STOPWORDS

from gensim.parsing.porter import PorterStemmer

#from gensim.summarization.textcleaner import tokenize\_by\_word

from spacy.lang.es import Spanish

nlp = Spanish()

import email.policy

from bs4 import BeautifulSoup

#import tensorflow as tf

#from tensorflow.keras.preprocessing.text import Tokenizer

#from tensorflow.keras.preprocessing.sequence import pad\_sequences

from sklearn.metrics import confusion\_matrix

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

import sklearn

###

from imblearn.over\_sampling import SMOTEN

from sklearn.feature\_extraction.text import TfidfVectorizer, CountVectorizer

from sklearn.metrics import classification\_report, plot\_confusion\_matrix

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

from sklearn.naive\_bayes import MultinomialNB

from sklearn.pipeline import make\_pipeline

from wordcloud import WordCloud

plt.rc("axes.spines", right=False, top=False)

plt.rc("font", family="serif")

###

'''from tqdm import tqdm

from keras.preprocessing.text import Tokenizer

tqdm.pandas(desc="progress-bar")

#from gensim.models import Doc2Vec

from sklearn import utils

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

from keras.preprocessing.sequence import pad\_sequences

from sklearn.linear\_model import LogisticRegression

#from gensim.models.doc2vec import TaggedDocument

import re'''

###

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="",

charset="utf8",

database="email\_spam"

)

app = Flask(\_\_name\_\_)

##session key

app.secret\_key = 'abcdef'

#######

UPLOAD\_FOLDER = 'static/upload'

ALLOWED\_EXTENSIONS = { 'csv'}

app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER

#####

##email

mail\_settings = {

"MAIL\_SERVER": 'smtp.gmail.com',

"MAIL\_PORT": 465,

"MAIL\_USE\_TLS": False,

"MAIL\_USE\_SSL": True,

"MAIL\_USERNAME": "rnd1024.64@gmail.com",

"MAIL\_PASSWORD": "kazxlklvfrvgncse"

}

app.config.update(mail\_settings)

mail = Mail(app)

#######

def sendmail(usermail,mess1):

subj1="Spam-Spoiler"

with app.app\_context():

msg = Message(subject=subj1, sender=app.config.get("MAIL\_USERNAME"),recipients=[usermail], body=mess1)

mail.send(msg)

@app.route('/', methods=['GET', 'POST'])

def index():

return render\_template('index.html')

@app.route('/login\_user', methods=['GET', 'POST'])

def login\_user():

msg=""

act=request.args.get("act")

if request.method=='POST':

uname=request.form['uname']

pwd=request.form['pass']

cursor = mydb.cursor()

cursor.execute('SELECT \* FROM register WHERE uname = %s AND pass = %s', (uname, pwd))

account = cursor.fetchone()

if account:

session['username'] = uname

ff=open("user.txt","w")

ff.write(uname)

ff.close()

return redirect(url\_for('userhome'))

else:

msg = 'Incorrect username/password!'

return render\_template('login\_user.html',msg=msg,act=act)

@app.route('/login', methods=['GET', 'POST'])

def login():

msg=""

act=request.args.get("act")

#usermail=""

#mess1="mytest"

#sendmail(usermail,mess1)

if request.method=='POST':

uname=request.form['uname']

pwd=request.form['pass']

cursor = mydb.cursor()

cursor.execute('SELECT \* FROM admin WHERE username = %s AND password = %s', (uname, pwd))

account = cursor.fetchone()

if account:

session['username'] = uname

return redirect(url\_for('train\_data'))

else:

msg = 'Incorrect username/password!'

return render\_template('login.html',msg=msg,act=act)

@app.route('/register', methods=['GET', 'POST'])

def register():

msg=""

act=request.args.get("act")

if request.method=='POST':

name=request.form['name']

mobile=request.form['mobile']

#email=request.form['email']

uname=request.form['uname']

pass1=request.form['pass']

#password=request.form['password']

mycursor = mydb.cursor()

mycursor.execute("SELECT count(\*) FROM register where uname=%s",(uname,))

cnt = mycursor.fetchone()[0]

if cnt==0:

mycursor.execute("SELECT max(id)+1 FROM register")

maxid = mycursor.fetchone()[0]

if maxid is None:

maxid=1

sql = "INSERT INTO register(id,name,mobile,uname,pass) VALUES (%s, %s, %s, %s, %s)"

val = (maxid,name,mobile,uname,pass1)

mycursor.execute(sql, val)

mydb.commit()

#print(mycursor.rowcount, "Registered Success")

msg="success"

#if mycursor.rowcount==1:

return redirect(url\_for('register',act='1'))

else:

msg='Already Exist!'

return render\_template('register.html',msg=msg,act=act)

@app.route('/setting', methods=['GET', 'POST'])

def setting():

msg=""

uname=""

act=request.args.get("act")

if 'username' in session:

uname = session['username']

ff=open("user.txt","r")

uname=ff.read()

ff.close()

mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM register where uname=%s",(uname,))

det = mycursor.fetchone()

em=det[3]

pw=det[6]

if request.method=='POST':

email=request.form['email']

password=request.form['pass']

mycursor.execute("update register set email=%s,password=%s where uname=%s",(email,password,uname))

mydb.commit()

return redirect(url\_for('setting',act='1'))

return render\_template('setting.html',msg=msg,em=em,pw=pw,act=act,det=det)

def emailsink(usermail,pwd,uname):

import email

import imaplib

mail = imaplib.IMAP4\_SSL('imap.gmail.com')

(retcode, capabilities) = mail.login(usermail,pwd)

mail.list()

mail.select('inbox')

subj1="Spam-Spoiler"

n=0

(retcode, messages) = mail.search(None, '(UNSEEN)')

if retcode == 'OK':

for num in messages[0].split() :

print ('Processing ')

n=n+1

typ, data = mail.fetch(num,'(RFC822)')

for response\_part in data:

if isinstance(response\_part, tuple):

original = email.message\_from\_bytes(response\_part[1])

# print (original['From'])

# print (original['Subject'])

raw\_email = data[0][1]

raw\_email\_string = raw\_email.decode('utf-8')

email\_message = email.message\_from\_string(raw\_email\_string)

for part in email\_message.walk():

if (part.get\_content\_type() == "text/plain"): # ignore attachments/html

body = part.get\_payload(decode=True)

save\_string = str(r"data.txt" )

myfile = open(save\_string, 'a')

myfile.write(original['From']+'\n')

myfile.write(original['Subject']+'\n')

myfile.write(body.decode('utf-8'))

subj=original['Subject']

sender=original['From']

mess=body.decode('utf-8')

myfile.write('\*\*\*\*\*\*\*\*\*\*\n')

myfile.close()

if subj1==subj:

print("subject")

else:

###

x=0

spam\_st=""

f1=open("spammail.txt","r")

dat=f1.read()

f1.close()

dat1=dat.split("|")

for rd in dat1:

rd1=rd.split('##')

spam\_st=rd1[1]

t1=mess

t2=rd1[0] #rd.strip()

if t2 in t1:

x+=1

print("yes")

break

else:

print("no")

mail\_det=""

if x>0:

print(spam\_st)

if spam\_st=="1":

mail\_det="Fraudulent"

elif spam\_st=="2":

mail\_det="Harrasment"

elif spam\_st=="3":

mail\_det="Suspicious"

mycursor = mydb.cursor()

mycursor.execute("SELECT max(id)+1 FROM read\_data")

maxid = mycursor.fetchone()[0]

if maxid is None:

maxid=1

sql = "INSERT INTO read\_data(id,subject,sender,uname,message,spam\_st) VALUES (%s, %s, %s, %s, %s, %s)"

val = (maxid,subj,sender,uname,mess,mail\_det)

mycursor.execute(sql, val)

mydb.commit()

##

#Reply mail

mess1=mail\_det+" mail has deleted \*\*\* "+mess+" \*\*\* "

sendmail(usermail,mess1)

#Delete mail

mail.store(num,'+FLAGS',r'(\Deleted)')

###

else:

continue

typ, data = mail.store(num,'+FLAGS','\\Seen')

#print (n)

return n

@app.route('/userhome', methods=['GET', 'POST'])

def userhome():

msg=""

uname=""

if 'username' in session:

uname = session['username']

ff=open("user.txt","r")

uname=ff.read()

ff.close()

mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM register where uname=%s",(uname,))

det = mycursor.fetchone()

em=det[3]

email=det[3]

pwd=det[6]

return render\_template('userhome.html',msg=msg,det=det)

@app.route('/spam\_detect', methods=['GET', 'POST'])

def spam\_detect():

msg=""

uname=""

if 'username' in session:

uname = session['username']

ff=open("user.txt","r")

uname=ff.read()

ff.close()

mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM register where uname=%s",(uname,))

det = mycursor.fetchone()

em=det[3]

email=det[3]

pwd=det[6]

print(email)

print(pwd)

now = datetime.now()

rdate=now.strftime("%d-%m-%Y")

rtime=now.strftime("%H-M")

dtt=rdate+" "+rtime

#########

res=emailsink(email,pwd,uname)

unread=res

########

mycursor.execute("SELECT \* FROM read\_data where uname=%s order by id desc",(uname,))

data = mycursor.fetchall()

return render\_template('spam\_detect.html',msg=msg,em=em,data=data,unread=unread)

@app.route('/train\_data', methods=['GET', 'POST'])

def train\_data():

msg=""

act = request.args.get('act')

pd.set\_option("display.max\_colwidth", 200)

warnings.filterwarnings("ignore") #ignore warnings

#dataset/SEFACED\_Email\_Forensic\_Dataset1.csv

data = pd.read\_csv(

"static/dataset/train.csv",

header=0,

encoding="latin-1",

usecols=[0, 1],

names=["label", "text"],

)

#dat1 = pd.read\_csv("static/dataset/SEFACED\_Email\_Forensic\_Dataset1.csv", header=0)

#dat=dat1.head()

data1=[]

i=0

for ds in data.values:

#if i<=200:

data1.append(ds)

#i+=1

'''plt.rc("axes.spines", right=False, top=False)

plt.rc("font", family="serif")

data = pd.read\_csv(

"static/dataset/data1.csv",

header=0,

encoding="latin-1",

usecols=[0, 1],

names=["label", "text"],

)

dat=data.head()

data1=[]

for ds in dat.values:

data1.append(ds)'''

'''for label, cmap in zip(["ham", "spam"], ["winter", "autumn"]):

text = data.query("label == @label")["text"].str.cat(sep=" ")

plt.figure(figsize=(10, 6))

#wc = WordCloud(width=1000, height=600, background\_color="#f8f8f8", colormap=cmap)

#wc.generate\_from\_text(text)

#plt.imshow(wc)

#plt.axis("off")

#plt.title(f"Words Commonly Used in ${label}$ Messages", size=20)

#plt.show()'''

'''data["length (words)"] = data["text"].str.split().apply(len)

print(data.groupby("label").agg([min, max, "mean"]))

ax = data.boxplot(by="label", figsize=(6, 4.5))

\_ = ax.set\_title("")

#plt.show()

#plt.close()

######

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

data["text"], data["label"], random\_state=8, stratify=data["label"]

)

\_ = y\_train.value\_counts().plot.bar(

color=["aqua", "orangered"], edgecolor="#555", alpha=0.5

)

#plt.show()

#plt.close()'''

#############

return render\_template('train\_data.html',msg=msg,data1=data1)

@app.route('/process1', methods=['GET', 'POST'])

def process1():

pd.set\_option("display.max\_colwidth", 200)

warnings.filterwarnings("ignore") #ignore warnings

#dff = pd.read\_csv("static/dataset/SEFACED\_Email\_Forensic\_Dataset1.csv",encoding='latin-1')

dff = pd.read\_csv("static/dataset/train.csv",encoding='latin-1')

class\_weight = 1 / dff["Text"].value\_counts()

class\_weight = dict(class\_weight / class\_weight.sum())

####

df = pd.read\_csv("static/dataset/data1.csv",encoding='latin-1')

df.head()

#stop\_words = stopwords.words('english')

stop\_words = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't",',','.','I','\'','-','/']

##Tokenize

data1=[]

i=0

for ds in df.values:

dt=[]

if i<5:

dt.append(ds[1])

text=ds[1]

doc = nlp(text)

text\_tokens = [token.text for token in doc]

#text\_tokens=tokenize\_by\_word(text)

#text\_tokens =word\_tokenize(text)

tokens\_without\_sw = [word for word in text\_tokens if not word in stop\_words]

dt.append(tokens\_without\_sw)

data1.append(dt)

i+=1

#Stemming

ps = PorterStemmer()

# choose some words to be stemmed

#words = ["program", "programs", "programmer", "programming", "programmers"]

data2=[]

i=0

for ds2 in df.values:

dt2=[]

if i<5:

text2=ds2[1]

dt2.append(ds2[1])

doc = nlp(text)

text\_tokens = [token.text for token in doc]

#text\_tokens =word\_tokenize(text2)

tokens\_without\_sw = [word for word in text\_tokens if not word in stop\_words]

swrd=[]

for w in tokens\_without\_sw:

sw=ps.stem(w)

swrd.append(sw)

dt2.append(swrd)

data2.append(dt2)

i+=1

##Stop words

stopwords = [ "a", "about", "above", "after", "again", "against", "all", "am", "an", "and", "any", "are", "as", "at", "be", "because", "been", "before", "being", "below", "between", "both", "but", "by", "could", "did", "do", "does", "doing", "down", "during", "each", "few", "for", "from", "further", "had", "has", "have", "having", "he", "he'd", "he'll", "he's", "her", "here", "here's", "hers", "herself", "him", "himself", "his", "how", "how's", "i", "i'd", "i'll", "i'm", "i've", "if", "in", "into", "is", "it", "it's", "its", "itself", "let's", "me", "more", "most", "my", "myself", "nor", "of", "on", "once", "only", "or", "other", "ought", "our", "ours", "ourselves", "out", "over", "own", "same", "she", "she'd", "she'll", "she's", "should", "so", "some", "such", "than", "that", "that's", "the", "their", "theirs", "them", "themselves", "then", "there", "there's", "these", "they", "they'd", "they'll", "they're", "they've", "this", "those", "through", "to", "too", "under", "until", "up", "very", "was", "we", "we'd", "we'll", "we're", "we've", "were", "what", "what's", "when", "when's", "where", "where's", "which", "while", "who", "who's", "whom", "why", "why's", "with", "would", "you", "you'd", "you'll", "you're", "you've", "your", "yours", "yourself", "yourselves" ]

#data = "All work and no play makes jack dull boy. All work and no play makes jack a dull boy."

data3=[]

i=0

for ds3 in df.values:

dt3=[]

if i<5:

content=ds3[1]

dt3.append(ds3[1])

content = content.lower()

swrd=[]

# Remove stop words

for stopword in stopwords:

content = content.replace(stopword + " ", "")

content = content.replace(" " + stopword, "")

swrd.append(content)

data3.append(swrd)

i+=1

#DATASET\_COLUMNS = ["Text","Class\_Label"]

#data.columns = DATASET\_COLUMNS

#data.head()

#stop\_words = stopwords.words('english')

#df['clean\_Text'] = df['Text'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop\_words)]))

#dat=df['clean\_Text']

#print(dat)

return render\_template('process1.html',data1=data1,data2=data2,data3=data3,class\_weight=class\_weight)

@app.route('/process2', methods=['GET', 'POST'])

def process2():

##########

plt.rc("axes.spines", right=False, top=False)

plt.rc("font", family="serif")

data = pd.read\_csv(

"static/dataset/data1.csv",

header=0,

encoding="latin-1",

usecols=[0, 1],

names=["label", "text"],

)

dat=data.head()

data["length (words)"] = data["text"].str.split().apply(len)

dataval=data.groupby("label").agg([min, max, "mean"])

print(data.groupby("label").agg([min, max, "mean"]))

ax = data.boxplot(by="label", figsize=(6, 4.5))

\_ = ax.set\_title("")

#plt.show()

plt.savefig("static/dataset/graph2.png")

plt.close()

######

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

data["text"], data["label"], random\_state=8, stratify=data["label"]

)

\_ = y\_train.value\_counts().plot.bar(

color=["aqua", "orangered"], edgecolor="#555", alpha=0.5

)

#plt.show()

plt.savefig("static/dataset/graph3.png")

plt.close()

############################################################################

data = pd.read\_csv(

"static/dataset/data2.csv",

header=0,

encoding="latin-1",

usecols=[0, 1],

names=["label", "text"],

)

dat=data.head()

data["length (words)"] = data["text"].str.split().apply(len)

dataval2=data.groupby("label").agg([min, max, "mean"])

print(data.groupby("label").agg([min, max, "mean"]))

ax = data.boxplot(by="label", figsize=(6, 4.5))

\_ = ax.set\_title("")

#plt.show()

plt.savefig("static/dataset/graph4.png")

plt.close()

######

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

data["text"], data["label"], random\_state=8, stratify=data["label"]

)

\_ = y\_train.value\_counts().plot.bar(

color=["aqua", "orangered"], edgecolor="#555", alpha=0.5

)

#plt.show()

plt.savefig("static/dataset/graph5.png")

plt.close()

#############################################################################

data = pd.read\_csv(

"static/dataset/data3.csv",

header=0,

encoding="latin-1",

usecols=[0, 1],

names=["label", "text"],

)

dat=data.head()

data["length (words)"] = data["text"].str.split().apply(len)

dataval3=data.groupby("label").agg([min, max, "mean"])

print(data.groupby("label").agg([min, max, "mean"]))

ax = data.boxplot(by="label", figsize=(6, 4.5))

\_ = ax.set\_title("")

#plt.show()

plt.savefig("static/dataset/graph6.png")

plt.close()

######

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

data["text"], data["label"], random\_state=8, stratify=data["label"]

)

\_ = y\_train.value\_counts().plot.bar(

color=["aqua", "orangered"], edgecolor="#555", alpha=0.5

)

#plt.show()

plt.savefig("static/dataset/graph7.png")

plt.close()

#################################################################

return render\_template('process2.html',dataval=dataval,dataval2=dataval2,dataval3=dataval3)

@app.route('/process3', methods=['GET', 'POST'])

def process3():

df = pd.read\_csv('static/dataset/spam11.csv')

print(df.shape)

dat=df.head()

data1=[]

for ds1 in dat.values:

data1.append(ds1)

dat2=df.describe()

data2=[]

drr=['count','mean','std','min','25%','50%','75%','max']

i=0

for ds2 in dat2.values:

dt=[]

dt.append(drr[i])

dt.append(ds2)

i+=1

data2.append(dt)

#df.info()

dat3=df.corr()

data3=[]

for ds3 in dat3.values:

data3.append(ds3)

#visualize correlation of variable using pearson correlation

plt.figure(figsize = (8,6))

sns.heatmap(df.corr(), vmax = 0.9, cmap = 'YlGnBu')

plt.title('Pearson Correlation', fontsize = 15, pad = 12, color = 'r')

plt.savefig("static/dataset/ff\_g1.png")

#plt.show()

#transform spam column to categorical data

df.spam[df['spam'] == 0] = 'ham'

df.spam[df['spam'] == 1] = 'spam'

dat4=df.head()

data4=[]

for ds4 in dat4.values:

data4.append(ds4)

#analyze of spam status based on capital run length average

dat5=pd.pivot\_table(df, index = 'spam', values = 'capital\_run\_length\_average',

aggfunc = {'capital\_run\_length\_average' : np.mean}).sort\_values('capital\_run\_length\_average', ascending = False)

print(dat5)

#analyze of spam status based on count of capital run length longest

pd.pivot\_table(df, index = 'spam', values = 'capital\_run\_length\_longest',

aggfunc = {'capital\_run\_length\_longest' : np.sum}).sort\_values('capital\_run\_length\_longest', ascending = False)

#anayze of spam status based on count of capital run length total

pd.pivot\_table(df, index = 'spam', values = 'capital\_run\_length\_total',

aggfunc = {'capital\_run\_length\_total' : np.sum}).sort\_values('capital\_run\_length\_total', ascending = False)

#anayze of spam status based on capital run length average, capital run length longest and capital run length total

pd.pivot\_table(df, index = 'spam', values = ['capital\_run\_length\_average', 'capital\_run\_length\_longest',

'capital\_run\_length\_total'],

aggfunc = {'capital\_run\_length\_average' : np.mean, 'capital\_run\_length\_longest' : np.sum,

'capital\_run\_length\_total' : np.sum}).sort\_values(['capital\_run\_length\_average',

'capital\_run\_length\_longest',

'capital\_run\_length\_total'], ascending = False)

#visualize the factor of spam message based on capital run length average, capital run length longest and capital run length total

plt.figure(figsize = (14,6))

chart = df.boxplot()

chart.set\_xticklabels(chart.get\_xticklabels(), rotation = 90)

plt.title('The Factor of Spam Message', fontsize = 15, pad = 12, color = 'b')

plt.xlabel('Factor')

plt.ylabel('Count')

plt.savefig("static/dataset/ff\_g2.png")

#plt.show()

return render\_template('process3.html',data1=data1,data2=data2,data3=data3,data4=data4,dat5=dat5)

##LSTM-GRU

def load\_data(stock, seq\_len):

amount\_of\_features = len(stock.columns)

data = stock.as\_matrix() #pd.DataFrame(stock)

sequence\_length = seq\_len + 1

result = []

for index in range(len(data) - sequence\_length):

result.append(data[index: index + sequence\_length])

result = np.array(result)

row = round(0.9 \* result.shape[0])

train = result[:int(row), :]

x\_train = train[:, :-1]

y\_train = train[:, -1][:,-1]

x\_test = result[int(row):, :-1]

y\_test = result[int(row):, -1][:,-1]

# Scaling the training set

sc = MinMaxScaler(feature\_range=(0,1))

training\_set\_scaled = sc.fit\_transform(training\_set)

# Since LSTMs store long term memory state, we create a data structure with 60 timesteps and 1 output

# So for each element of training set, we have 60 previous training set elements

X\_train = []

y\_train = []

for i in range(60,2769):

X\_train.append(training\_set\_scaled[i-60:i,0])

y\_train.append(training\_set\_scaled[i,0])

X\_train, y\_train = np.array(X\_train), np.array(y\_train)

# Reshaping X\_train for efficient modelling

X\_train = np.reshape(X\_train, (X\_train.shape[0],X\_train.shape[1],1))

# The LSTM architecture

regressor = Sequential()

# First LSTM layer with Dropout regularisation

regressor.add(LSTM(units=50, return\_sequences=True, input\_shape=(X\_train.shape[1],1)))

regressor.add(Dropout(0.2))

# Second LSTM layer

regressor.add(LSTM(units=50, return\_sequences=True))

regressor.add(Dropout(0.2))

# Third LSTM layer

regressor.add(LSTM(units=50, return\_sequences=True))

regressor.add(Dropout(0.2))

# Fourth LSTM layer

regressor.add(LSTM(units=50))

regressor.add(Dropout(0.2))

# The output layer

regressor.add(Dense(units=1))

# Compiling the RNN

regressor.compile(optimizer='rmsprop',loss='mean\_squared\_error')

# Fitting to the training set

regressor.fit(X\_train,y\_train,epochs=50,batch\_size=32)

# The GRU architecture

regressorGRU = Sequential()

# First GRU layer with Dropout regularisation

regressorGRU.add(GRU(units=50, return\_sequences=True, input\_shape=(X\_train.shape[1],1), activation='tanh'))

regressorGRU.add(Dropout(0.2))

# Second GRU layer

regressorGRU.add(GRU(units=50, return\_sequences=True, input\_shape=(X\_train.shape[1],1), activation='tanh'))

regressorGRU.add(Dropout(0.2))

# Third GRU layer

regressorGRU.add(GRU(units=50, return\_sequences=True, input\_shape=(X\_train.shape[1],1), activation='tanh'))

regressorGRU.add(Dropout(0.2))

# Fourth GRU layer

regressorGRU.add(GRU(units=50, activation='tanh'))

regressorGRU.add(Dropout(0.2))

# The output layer

regressorGRU.add(Dense(units=1))

# Compiling the RNN

regressorGRU.compile(optimizer=SGD(lr=0.01, decay=1e-7, momentum=0.9, nesterov=False),loss='mean\_squared\_error')

# Fitting to the training set

regressorGRU.fit(X\_train,y\_train,epochs=50,batch\_size=150)

# Preparing X\_test and predicting the prices

X\_test = []

for i in range(60,311):

X\_test.append(inputs[i-60:i,0])

X\_test = np.array(X\_test)

X\_test = np.reshape(X\_test, (X\_test.shape[0],X\_test.shape[1],1))

predicted\_stock\_price = regressor.predict(X\_test)

predicted\_stock\_price = sc.inverse\_transform(predicted\_stock\_price)

@app.route('/process4', methods=['GET', 'POST'])

def process4():

##########

plt.rc("axes.spines", right=False, top=False)

plt.rc("font", family="serif")

data = pd.read\_csv(

"static/dataset/data1.csv",

header=0,

encoding="latin-1",

usecols=[0, 1],

names=["label", "text"],

)

dat=data.head()

data["length (words)"] = data["text"].str.split().apply(len)

dataval=data.groupby("label").agg([min, max, "mean"])

print(data.groupby("label").agg([min, max, "mean"]))

ax = data.boxplot(by="label", figsize=(6, 4.5))

\_ = ax.set\_title("")

#plt.show()

#plt.savefig("static/dataset/graph2.png")

#plt.close()

######

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

data["text"], data["label"], random\_state=8, stratify=data["label"]

)

#\_ = y\_train.value\_counts().plot.bar(

# color=["aqua", "orangered"], edgecolor="#555", alpha=0.5

#)

#plt.show()

#plt.xticks(rotation=0)

#plt.savefig("static/dataset/graph3.png")

#plt.close()

############################################################################

data = pd.read\_csv(

"static/dataset/data2.csv",

header=0,

encoding="latin-1",

usecols=[0, 1],

names=["label", "text"],

)

dat=data.head()

data["length (words)"] = data["text"].str.split().apply(len)

dataval2=data.groupby("label").agg([min, max, "mean"])

print(data.groupby("label").agg([min, max, "mean"]))

ax = data.boxplot(by="label", figsize=(6, 4.5))

\_ = ax.set\_title("")

#plt.show()

#plt.savefig("static/dataset/graph4.png")

#plt.close()

######

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

data["text"], data["label"], random\_state=8, stratify=data["label"]

)

#\_ = y\_train.value\_counts().plot.bar(

# color=["aqua", "orangered"], edgecolor="#555", alpha=0.5

#)

#plt.show()

#plt.xticks(rotation=0)

#plt.savefig("static/dataset/graph5.png")

#plt.close()

#############################################################################

data = pd.read\_csv(

"static/dataset/data3.csv",

header=0,

encoding="latin-1",

usecols=[0, 1],

names=["label", "text"],

)

dat=data.head()

data["length (words)"] = data["text"].str.split().apply(len)

dataval3=data.groupby("label").agg([min, max, "mean"])

print(data.groupby("label").agg([min, max, "mean"]))

ax = data.boxplot(by="label", figsize=(6, 4.5))

\_ = ax.set\_title("")

#plt.show()

#plt.savefig("static/dataset/graph6.png")

#plt.close()

######

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

data["text"], data["label"], random\_state=8, stratify=data["label"]

)

#\_ = y\_train.value\_counts().plot.bar(

# color=["aqua", "orangered"], edgecolor="#555", alpha=0.5

#)

#plt.show()

#plt.xticks(rotation=0)

#plt.savefig("static/dataset/graph7.png")

#plt.close()

#################################################################

df = pd.read\_csv('static/dataset/SEFACED\_Email\_Forensic\_Dataset1.csv',delimiter=',',encoding='latin-1')

df = df[['Class\_Label','Text']]

df = df[pd.notnull(df['Text'])]

df.rename(columns = {'Message':'Text'}, inplace = True)

print(df.head())

data1=[]

##for ds in data.values:

## data1.append(ds)

dsf=df.shape

print(dsf)

print(dsf[0])

df.index = range(dsf[0])

df['Text'].apply(lambda x: len(x.split(' '))).sum()

cnt\_pro = df['Class\_Label'].value\_counts()

#plt.figure(figsize=(12,4))

#sns.barplot(cnt\_pro.index, cnt\_pro.values, alpha=0.8)

#plt.ylabel('Number of Occurrences', fontsize=12)

#plt.xlabel('Class\_Label', fontsize=12)

#plt.xticks(rotation=90)

#plt.savefig("static/dataset/graph1.png")

#plt.show();

return render\_template('process4.html',dataval=dataval,dataval2=dataval2,dataval3=dataval3)

##########################

@app.route('/logout')

def logout():

# remove the username from the session if it is there

session.pop('username', None)

return redirect(url\_for('index'))

if \_\_name\_\_ == '\_\_main\_\_':

app.secret\_key = os.urandom(12)

app.run(debug=True,host='0.0.0.0', port=5000)