**FAKE NEWS DETECTION USING NLP**

**Program Code for Loading and Preprocessing Dataset:**

**( Based on Passive Aggressive Classifier )**

import pandas as pd

import sklearn

import itertools

import numpy as np

import seaborn as sb

import re

import nltk

import pickle

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

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn import metrics

from sklearn.metrics import confusion\_matrix

from matplotlib import pyplot as plt

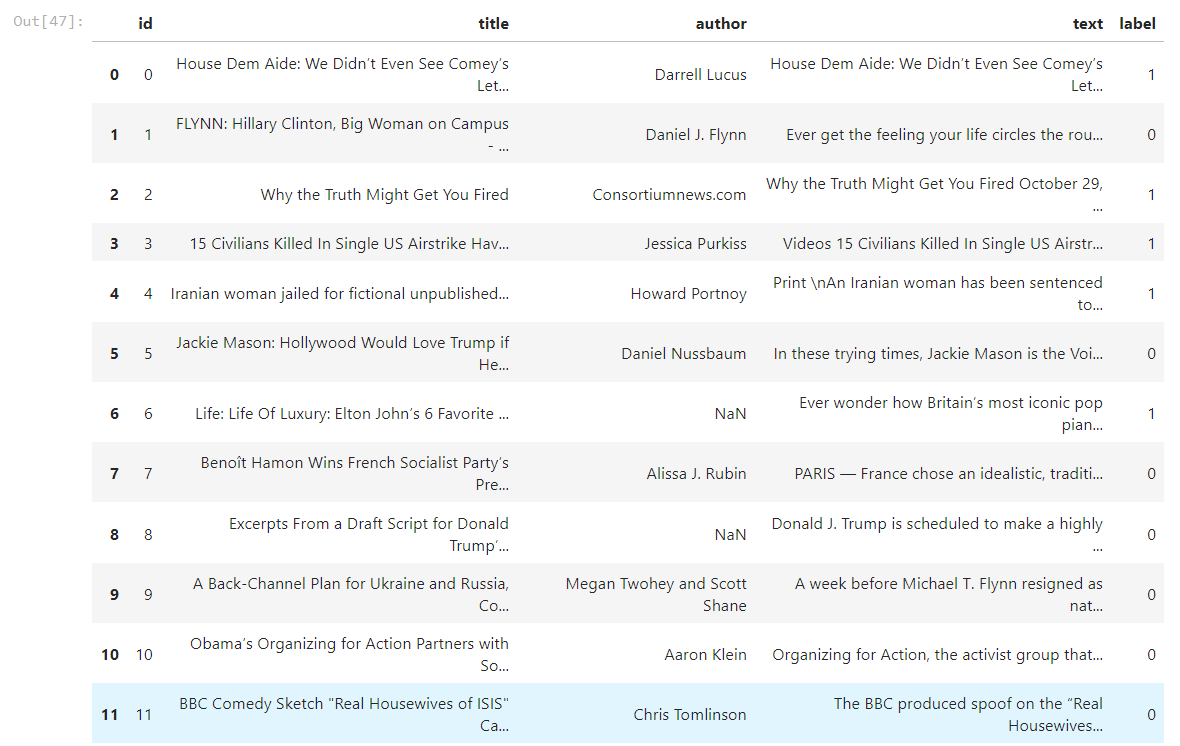
from sklearn.linear\_model import PassiveAggressiveClassifier

from nltk.stem import WordNetLemmatizer

from nltk.corpus import stopwords

train\_df = pd.read\_csv(r'C:\Users\Mayur\Downloads\train.csv')

train\_df.head(15)



train\_df = train\_df.drop("author", axis = 1)

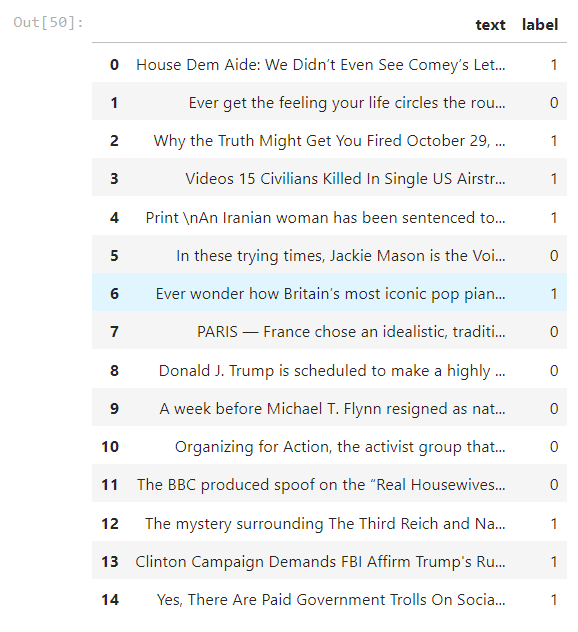
train\_df = train\_df.drop("title", axis = 1)

train\_df = train\_df.drop("id", axis = 1)

train\_df.shape



train\_df.head(15)



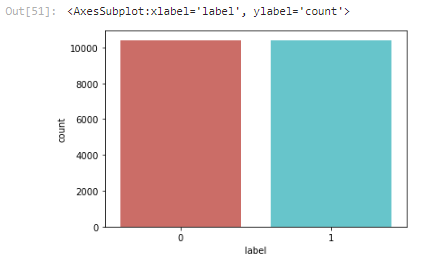
**Program Code for Feature-Extraction and Classification:**

def create\_distribution(dataFile):

return sb.countplot(x='label', data=dataFile, palette='hls')

# by calling below we can see that training, test and valid data seems to be failry evenly distributed between the classes

create\_distribution(train\_df)



def data\_qualityCheck():

print("Checking data qualitites...")

train\_df.isnull().sum()

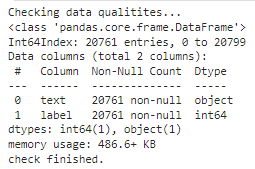
train\_df.info()

print("check finished.")

data\_qualityCheck()

train\_df = train\_df.dropna()

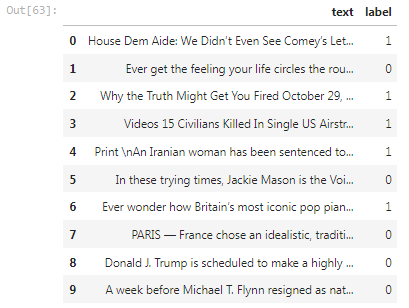
data\_qualityCheck()



train\_df.shape

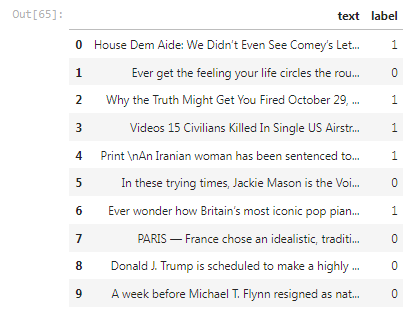


train\_df.head(10)



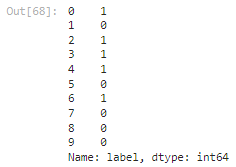
train\_df.reset\_index(drop= True,inplace=True)

train\_df.head(10)



label\_train = train\_df.label

label\_train.head(10)

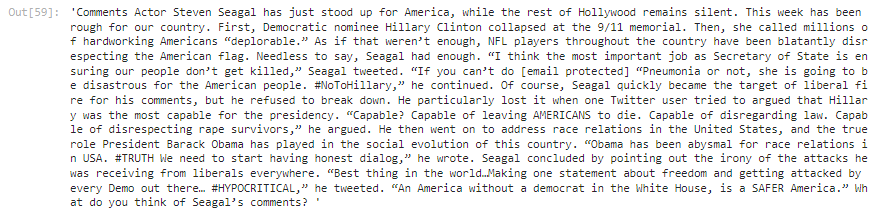


train\_df = train\_df.drop("label", axis = 1)

train\_df.head(10)



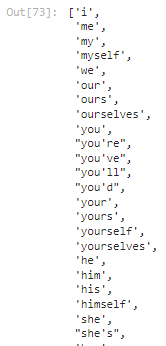
train\_df['text'][2188]



lemmatizer = WordNetLemmatizer()

stpwrds = list(stopwords.words('english'))

stpwrds



for x in range(len(train\_df)) :

corpus = []

review = train\_df['text'][x]

review = re.sub(r'[^a-zA-Z\s]', '', review)

review = review.lower()

review = nltk.word\_tokenize(review)

for y in review :

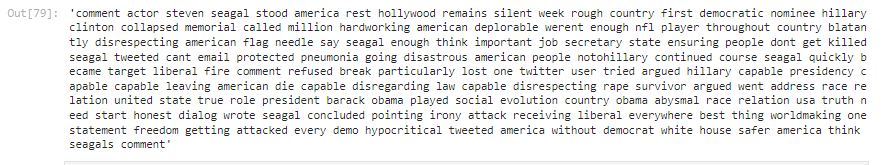
if y not in stpwrds :

corpus.append(lemmatizer.lemmatize(y))

review = ' '.join(corpus)

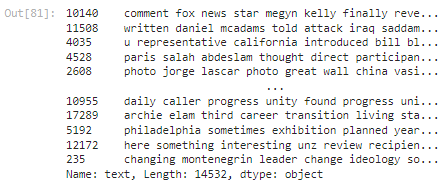
train\_df['text'][x] = review

train\_df['text'][2182]



X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(train\_df['text'], label\_train, test\_size=0.3, random\_state=1)

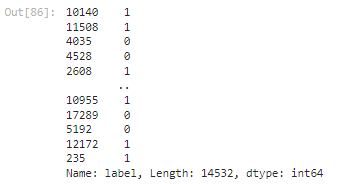
X\_train



X\_train.shape



Y\_train



tfidf\_v = TfidfVectorizer()

tfidf\_X\_train = tfidf\_v.fit\_transform(X\_train)

tfidf\_X\_test = tfidf\_v.transform(X\_test)

tfidf\_X\_train.shape



def plot\_confusion\_matrix(cm, classes,

normalize=False,

title='Confusion matrix',

cmap=plt.cm.Blues):

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

print("Normalized confusion matrix")

else:

print('Confusion matrix, without normalization')

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, cm[i, j],

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

classifier = PassiveAggressiveClassifier()

classifier.fit(tfidf\_X\_train,Y\_train)



Y\_pred = classifier.predict(tfidf\_X\_test)

score = metrics.accuracy\_score(Y\_test, Y\_pred)

print(f'Accuracy: {round(score\*100,2)}%')

cm = metrics.confusion\_matrix(Y\_test, Y\_pred)

plot\_confusion\_matrix(cm, classes=['FAKE Data', 'REAL Data'])

