In [30]:
```
import warnings
warnings.filterwarnings("ignore")
import pickle
import numpy as np
import re
import pandas as pd
from tqdm import tqdm
import string
from nltk import word_tokenize
from nltk.corpus import stopwords
import nltk
nltk.download('punkt')
nltk.download('stopwords')
import seaborn as sns
from string import punctuation
import matplotlib.pyplot as plt
import time
from sklearn.metrics import accuracy_score,confusion_matrix,precision_score
from sklearn.preprocessing  import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import train_test_split
import joblib
import pickle
```

```
[nltk_data] Downloading package punkt to /usr/share/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Defining all the preprocessing function that i needed before final prediction

In [31]:
```
def count_stopwords(text):
    '''this function will count total number of stop words in text and title column'''

    stop_words = set(stopwords.words('english'))
    word_tokens = nltk.word_tokenize(text)
    c_stopwords = [w for w in word_tokens if w in stop_words]

    return len(c_stopwords)

def count_unique_words(text):
    '''this function will count total number of unique words in text and title column'''
    word_tokens = nltk.word_tokenize(text)
    unique_words=[]
    for i in range(len(word_tokens)):
        if word_tokens[i] not in unique_words:
            unique_words.append(word_tokens[i])
    return len(unique_words)


def data_preprocess(text):
    '''This function will will preprocess the data by removing the puchuation digit email address and all non alphabet words from text.'''

    final_text=text.lower()
    final_text = re.sub('((www\.[^\s]+)|(https?://[^\s]+))','',final_text) #remove http links from text
    final_text=re.sub("\d+", " ", final_text) #remove any digit from text
    final_text=re.sub("[^a-zA-Z]+", " ", final_text) #remove anything except Alphabets from text

    #removing the punctuation
    punctuations = '!{}:[];"\,<>./?@#$%^&*_~'
    no_punc = " "
    for char in final_text:
        if char in punctuations:
            no_punc = no_punc + char.lower()
        else:
            no_punc+= " " #if punchuation found add space
    final_text=no_punc.strip()
    return final_text

CONTRACTION_MAP=pickle.load(open('../input/fake-news-case-study-preprocessed-daa/CONTRACTION_MAP.pkl',"rb")) # loading the contraction map

def deconstracted(text):
    '''this function will Replace all apostrophe/short words from text data'''
    for word in text.split():
        if word.lower() in CONTRACTION_MAP:
            text = text.replace(word, CONTRACTION_MAP[word.lower()])
    return text


def remove_stopwords(text):
    '''This function will remove stopwords from text data.'''
    stop_words = set(stopwords.words('english'))
    word_tokens = word_tokenize(text)

    filtered_sentence = " ".join([w for w in word_tokens if not w.lower() in stop_words])

    return filtered_sentence
```

Defining the function_1 which will take input here input can be single or set of datapoint and return the prediction of output

In [32]:
```
def function_1(test):
    """this function will take dataset as input(single of set of datapoint) and perfrom data preprocessing and return the predictions as output"""
    df_data=test.copy(deep=True)

    #filling nan value using ffil method
    df_data.fillna(method="ffill",inplace=True)

    #droping the duplicates
    df_data.drop_duplicates(keep='first',inplace=True)

    # creating new features from text data
    df_data["num_characters_title"] = df_data['title'].apply(len)
    df_data["num_characters_text"] = df_data['text'].apply(len)

    df_data["num_word_title"] = df_data['title'].apply(lambda x:len(nltk.word_tokenize(x)))
    df_data["num_word_text"] = df_data['text'].apply(lambda x:len(nltk.word_tokenize(x)))

    df_data["num_sentences_title"] = df_data['title'].apply(lambda x:len(nltk.sent_tokenize(x)))
    df_data["num_sentences_text"] = df_data['text'].apply(lambda x:len(nltk.sent_tokenize(x)))

    df_data["Count_unique_words_title"] = df_data['title'].apply(lambda x:count_unique_words(x))
    df_data["Count_unique_words_text"] = df_data['text'].apply(lambda x:count_unique_words(x))

    df_data["Count_Stop_words_title"] = df_data['title'].apply(lambda x:count_stopwords(x))
    df_data["Count_Stop_words_text"] = df_data['text'].apply(lambda x:count_stopwords(x))

    #Calculating average word length
    #This can be calculated by dividing the counts of characters by counts of words.
    df_data["Avg_word_length_title"] = df_data["num_characters_title"]/df_data["num_word_title"]
    df_data["Avg_word_length_text"] = df_data["num_characters_text"]/df_data["num_word_text"]

    #Calculating average sentence length
    #This can be calculated by dividing the counts of words by the counts of sentences.
    df_data["Avg_sentence_length_title"] = df_data["num_word_title"]/df_data["num_sentences_title"]
    df_data["Avg_sentence_length_text"] = df_data["num_word_text"]/df_data["num_sentences_text"]

    #Stopwords count vs words counts Ratio
    #This feature is also the ratio of counts of stopwords to the total number of words.
    df_data["Stopword_count_ratio_title"] = df_data["Count_Stop_words_title"]/df_data["num_word_title"]
    df_data["Stopword_count_ratio_text"] = df_data["Count_Stop_words_text"]/df_data["num_word_text"]

    #unnique words vs word count Ratio
    #This feature is basically the ratio of unique words to a total number of words.
    df_data["Unique_words_count_ratio_title"] = df_data["Count_unique_words_title"]/df_data["num_word_title"]
    df_data["Unique_words_count_ratio_text"] = df_data["Count_unique_words_text"]/df_data["num_word_text"]

    #To avoid any +/- inf nan values issue
    df_data = df_data.replace([np.inf, -np.inf, -0,np.nan], 0)#fix for infinly neg issue

    # performing the decontraction on the dataset
    df_data["cleaned_text"] = df_data.apply(lambda x: deconcratected(x["text"]),axis=1)
    df_data["cleaned_title"] = df_data.apply(lambda x: deconcratected(x["title"]),axis=1)

    #apply preprocessing on the dataset
    df_data["cleaned_text"] = df_data.apply(lambda x: data_preprocess(x["cleaned_text"]),axis=1)
    df_data["cleaned_title"] = df_data.apply(lambda x: data_preprocess(x["cleaned_title"]),axis=1)

    df_data["Without_Stopwords_text"]=df_data.apply(lambda x:remove_stopwords(x["cleaned_text"]),axis=1)
    df_data["Without_Stopwords_title"]=df_data.apply(lambda x:remove_stopwords(x["cleaned_title"]),axis=1)

    # # fatching the required features in our varriable X
    X=df_data.drop(["id","title","text","cleaned_text","cleaned_title","author"], axis=1, inplace=False)

    #loading the standard scaler
    Std_Scaler=pickle.load(open('../input/fake-news-case-study-preprocessed-daa/Std_Scaler.pkl','rb'))

    #fetching the numerical features
    df_Num_Ft=X.drop(["Without_Stopwords_text","Without_Stopwords_title"],axis=1, inplace=False)

    #standerdizing the numerical features
    df_Num_Std=Std_Scaler.transform(df_Num_Ft)

    #loading the tfidf vectorizer
    vectorizer_text_tfidf=pickle.load(open('../input/fake-news-case-study-preprocessed-daa/vectorizer_text_tfidf.pkl','rb'))
    vectorizer_title_tfdf=pickle.load(open('../input/fake-news-case-study-preprocessed-daa/vectorizer_title_tfdf.pkl','rb'))

    #vectorizing the both title and text using tfidf vectorizer
    df_text_tfidf = vectorizer_text_tfidf.transform(X['Without_Stopwords_title'].values).toarray()
    df_text_tfidf = vectorizer_text_tfidf.transform(X['Without_Stopwords_text'].values).toarray()

    # stacking all the features for train and test dataset
    df_final = np.hstack((df_title_tfidf, df_text_tfidf, df_Num_Std))

    #loading the our best model which is voting classifier
    Voting_clf = pickle.load(open('../input/fake-news-case-study-preprocessed-daa/voting.pkl', 'rb'))

    #doing the prediction on final data
    y_test_pred_voting = Voting_clf.predict(df_final)

    return y_test_pred_voting
```

loading the testdata set

In [33]:
```
test=pd.read_csv("../input/case-studyfake-news-dataset/test.csv",error_bad_lines=False)
```

In [34]:
```
test.shape
```

Out[34]:
```
(5200, 4)
```

Passing the single data point to our function_1 and getting the prediction for that data point

In [36]:
```
y_pred=function_1(test.iloc[[0],:])#test.iloc[[0] this will give single row from test dataset
```

Prediction from our model

In [37]:
```
y_pred[0]
```

Out[37]:
```
0
```

Passing the whole test dataset to our function_1 and getting the prediction

In [38]:
```
y_pred=function_1(test)
```

got the prediction for 5200 data points

In [40]:
```
len(y_pred)
```

Out[40]:
```
5200
```

In [41]:
```
y_pred
```

Out[41]:
```
array([0, 1, 1, ..., 0, 1, 0])
```

Defining the function which will take dataset with X and Y values and return the accuracy on same dataset

In [43]:
```
def function_2(test):
    """this function will take dataset (with x and y) as input and perform data preprocessing and return the accuracy of model"""

    df_data=test.copy(deep=True) # creating deep copy or original dataset

    #dropping the NAN rows from dataset
    df_data.dropna(inplace=True)
    df_data.reset_index(drop=True,inplace=True)

    #droping the duplicates
    df_data.drop_duplicates(keep='first',inplace=True)

    # creating new features from text data
    df_data["num_characters_title"] = df_data['title'].apply(len)
    df_data["num_characters_text"] = df_data['text'].apply(len)

    df_data["num_word_title"] = df_data['title'].apply(lambda x:len(nltk.word_tokenize(x)))
    df_data["num_word_text"] = df_data['text'].apply(lambda x:len(nltk.word_tokenize(x)))

    df_data["num_sentences_title"] = df_data['title'].apply(lambda x:len(nltk.sent_tokenize(x)))
    df_data["num_sentences_text"] = df_data['text'].apply(lambda x:len(nltk.sent_tokenize(x)))

    df_data["Count_unique_words_title"] = df_data['title'].apply(lambda x:count_unique_words(x))
    df_data["Count_unique_words_text"] = df_data['text'].apply(lambda x:count_unique_words(x))

    df_data["Count_Stop_words_title"] = df_data['title'].apply(lambda x:count_stopwords(x))
    df_data["Count_Stop_words_text"] = df_data['text'].apply(lambda x:count_stopwords(x))

    #Calculating average word length
    #This can be calculated by dividing the counts of characters by counts of words.
    df_data["Avg_word_length_title"] = df_data["num_characters_title"]/df_data["num_word_title"]
    df_data["Avg_word_length_text"] = df_data["num_characters_text"]/df_data["num_word_text"]

    #Calculating average sentence length
    #This can be calculated by dividing the counts of words by the counts of sentences.
    df_data["Avg_sentence_length_title"] = df_data["num_word_title"]/df_data["num_sentences_title"]
    df_data["Avg_sentence_length_text"] = df_data["num_word_text"]/df_data["num_sentences_text"]

    #Stopwords count vs words counts Ratio
    #This feature is also the ratio of counts of stopwords to the total number of words.
    df_data["Stopword_count_ratio_title"] = df_data["Count_Stop_words_title"]/df_data["num_word_title"]
    df_data["Stopword_count_ratio_text"] = df_data["Count_Stop_words_text"]/df_data["num_word_text"]

    #unnique words vs word count Ratio
    #This feature is basically the ratio of unique words to a total number of words.
    df_data["Unique_words_count_ratio_title"] = df_data["Count_unique_words_title"]/df_data["num_word_title"]
    df_data["Unique_words_count_ratio_text"] = df_data["Count_unique_words_text"]/df_data["num_word_text"]

    #To avoid any +/- inf nan values issue
    df_data = df_data.replace([np.inf, -np.inf, -0,np.nan], 0)#fix for infinly neg issue

    # performing the decontraction on the dataset
    df_data["cleaned_text"] = df_data.apply(lambda x: deconcratected(x["text"]),axis=1)
    df_data["cleaned_title"] = df_data.apply(lambda x: deconcratected(x["title"]),axis=1)

    #apply preprocessing on the dataset
    df_data["cleaned_text"] = df_data.apply(lambda x: data_preprocess(x["cleaned_text"]),axis=1)
    df_data["cleaned_title"] = df_data.apply(lambda x: data_preprocess(x["cleaned_title"]),axis=1)

    df_data["Without_Stopwords_text"]=df_data.apply(lambda x:remove_stopwords(x["cleaned_text"]),axis=1)
    df_data["Without_Stopwords_title"]=df_data.apply(lambda x:remove_stopwords(x["cleaned_title"]),axis=1)

    #splitting class labels and all the features
    Y=df_data["label"]
    X=df_data.drop(["id","title","text","label","cleaned_text","cleaned_title","author"], axis=1, inplace=False)

    # performing train test split 25% data for testing and 75% data for training the model
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, stratify=Y,random_state=42)

    X_train_Num_Ft=X_train.drop(["Without_Stopwords_text","Without_Stopwords_title"],axis=1,inplace=False)
    X_test_Num_Ft=X_test.drop(["Without_Stopwords_text","Without_Stopwords_title"],axis=1,inplace=False)

    #installing the standard scaler for numberical features
    Std_Scaler=StandardScaler()

    X_train_Num_Std=Std_Scaler.fit_transform(X_train_Num_Ft)
    X_test_Num_Std=Std_Scaler.transform(X_test_Num_Ft)

    #vectorizing the title features using tfidf vectorizer
    vectorizer_title_tfidf =TfidfVectorizer(max_features=3500)
    vectorizer_title_tfidf.fit(X_train['Without_Stopwords_title'].values)

    # we use the fitted tfidfVectorizer to convert the text to vector
    X_train_title_tfidf = vectorizer_title_tfidf.transform(X_train['Without_Stopwords_title']).toarray()
    X_test_title_tfidf = vectorizer_title_tfidf.transform(X_test['Without_Stopwords_title']).toarray()

    #vectorizing the text features using tfidf vectorizer
    vectorizer_text_tfidf =TfidfVectorizer(max_features=4500)
    vectorizer_text_tfidf.fit(X_train['Without_Stopwords_text'].values)

    # we use the fitted countVectorizer to convert the text to vector
    X_train_text_tfidf = vectorizer_text_tfidf.transform(X_train['Without_Stopwords_text'].values).toarray()
    X_test_text_tfidf = vectorizer_text_tfidf.transform(X_test['Without_Stopwords_text'].values).toarray()

    # # stacking all the features for train and test dataset
    X_train_final_tfidf = np.hstack((X_train_title_tfidf, X_train_text_tfidf,X_train_Num_Std))
    X_test_final_tfidf = np.hstack((X_test_title_tfidf , X_test_text_tfidf,X_test_Num_Std))

    #loading our best model which is voting classifier
    Voting_clf = pickle.load(open('../input/fake-news-case-study-preprocessed-daa/voting.pkl', 'rb'))

    #fitting on train dataset
    Voting_clf.fit(X_train_final_tfidf,y_train)
    #performing the prediction on test dataset
    y_pred = Voting_clf.predict(X_test_final_tfidf)
    #calculating the accuracy on testdataset
    accuracy = accuracy_score(y_test,y_pred)

    return accuracy
```

Reading the train dataset

In [44]:
```
train=pd.read_csv("../input/case-studyfake-news-dataset/train.csv",error_bad_lines=False)
```

In [45]:
```
train.shape
```

Out[45]:
```
(20800, 5)
```

Calling function_2 and passing whole train dataset and getting the accuracy as output

In [46]:
```
accuracy=function_2(train)
```

Got the 98.6% accuracy

In [47]:
```
accuracy
```

Out[47]:
```
0.9866017497812773
```

In [ ]:
```

```