#Importing the required libraries to read,visualize and model the givn dataset files  
import pandas as pd  
import numpy as np  
import seaborn as sns  
import matplotlib.pyplot as plt  
import plotly.graph\_objects as go  
import plotly.express as px  
import warnings  
warnings.filterwarnings("ignore")  
import re  
import re  
import string  
import nltk  
from nltk.corpus import stopwords  
from sklearn import model\_selection  
from sklearn.feature\_extraction.text import CountVectorizer,TfidfVectorizer  
from sklearn.model\_selection import train\_test\_split,GridSearchCV  
from sklearn.naive\_bayes import MultinomialNB  
from sklearn.metrics import f1\_score  
from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score  
from sklearn.pipeline import Pipeline  
from sklearn.feature\_extraction.text import TfidfTransformer  
from sklearn.linear\_model import LogisticRegression  
from sklearn.svm import SVC  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.multioutput import MultiOutputClassifier  
from sklearn.model\_selection import RepeatedStratifiedKFold  
import nltk  
from nltk.tokenize import word\_tokenize,RegexpTokenizer  
from nltk.stem import WordNetLemmatizer  
from platform import python\_version  
print (python\_version())

3.9.13

# Read the dataset csv files and create pandas datframes  
train\_df=pd.read\_csv("train\_data\_cleaning.csv")  
test\_df=pd.read\_csv("test\_data\_cleaning.csv")  
  
#X=train\_df.text  
#y=train\_df.target

# Define a function to explore the train and test dataframes  
def explore\_data(df):  
   
   
   
 print("-"\*50)  
 print('Shape of the dataframe:',df.shape)  
 print("Number of records in train data set:",df.shape[0])  
 print("Information of the dataset:")  
 df.info()  
 print("-"\*50)  
 print("First 5 records of the dataset:")  
 return df.head()  
 print("-"\*50)

# Lets use explore\_data() function to explore train data  
explore\_data(train\_df)

--------------------------------------------------  
Shape of the dataframe: (7613, 5)  
Number of records in train data set: 7613  
Information of the dataset:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 7613 entries, 0 to 7612  
Data columns (total 5 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 id 7613 non-null int64   
 1 keyword 7552 non-null object  
 2 location 5080 non-null object  
 3 text 7613 non-null object  
 4 target 7613 non-null int64   
dtypes: int64(2), object(3)  
memory usage: 297.5+ KB  
--------------------------------------------------  
First 5 records of the dataset:

id keyword location text \  
0 1 NaN NaN Our Deeds are the Reason of this # earthquake...   
1 4 NaN NaN Forest fire near La Ronge Sask . Canada   
2 5 NaN NaN All residents asked to ' shelter in place ' ...   
3 6 NaN NaN 13,000 people receive # wildfires evacuation ...   
4 7 NaN NaN Just got sent this photo from Ruby # Alaska a...   
  
 target   
0 1   
1 1   
2 1   
3 1   
4 1

# Lets use explore\_data() function to explore test data  
explore\_data(test\_df)

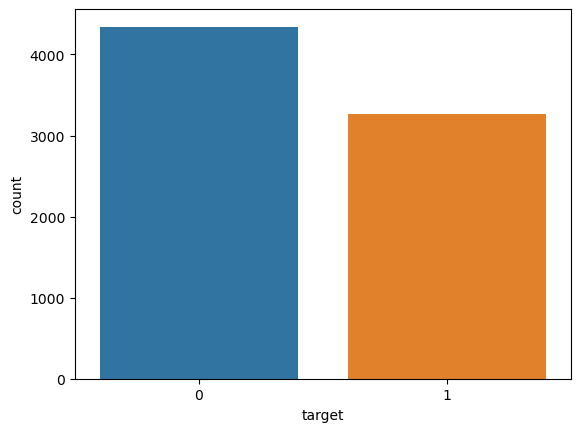
--------------------------------------------------  
Shape of the dataframe: (3263, 4)  
Number of records in train data set: 3263  
Information of the dataset:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 3263 entries, 0 to 3262  
Data columns (total 4 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 id 3263 non-null int64   
 1 keyword 3237 non-null object  
 2 location 2158 non-null object  
 3 text 3263 non-null object  
dtypes: int64(1), object(3)  
memory usage: 102.1+ KB  
--------------------------------------------------  
First 5 records of the dataset:

id keyword location text  
0 0 NaN NaN Just happened a terrible car crash  
1 2 NaN NaN Heard about # earthquake is different cities,...  
2 3 NaN NaN there is a forest fire at spot pond, geese are...  
3 9 NaN NaN Apocalypse lighting . # Spokane # wildfires  
4 11 NaN NaN Typhoon Soudelor kills 28 in China and Taiwan

def feature\_viz(df,feature):  
   
 '''Input- df=pandas dataframe  
 feature= column to be charted  
 Output- bar and scatter chart using plotly   
   
 '''  
 #Visualize the feature  
 if feature=='target':   
 sns.countplot(feature, data=df)  
 print('Target of 0 is {} % of total'.format(round(df[feature].value\_counts()[0]/len(df[feature])\*100)))  
 print('Target of 1 is {} % of total'.format(round(df[feature].value\_counts()[1]/len(df[feature])\*100)))  
 else:  
 #Distinct keywords in train dataset  
 feat=df[feature].value\_counts()  
 print(feat.head())  
 fig = px.scatter(feat, x=feat.values, y=feat.index,size=feat.values)  
 fig.show()

feature\_viz(train\_df,'target')

Target of 0 is 57 % of total  
Target of 1 is 43 % of total



#Calculate count and percentage of missing values in the dataframe  
  
def missing\_values(df):  
   
 '''Input- df=pandas dataframe  
 Output- print missing records count and % of the input dataframe and visualize using MSNO  
 '''  
   
 print("Number of records with missing location:",df.location.isnull().sum())  
 print("Number of records with missing keywords:",df.keyword.isnull().sum())  
 print('{}% of location values are missing from Total Number of Records.'.format(round((df.location.isnull().sum())/(df.shape[0])\*100),2))  
 print('{}% of keywords values are missing from Total Number of Records.'.format(round((df.keyword.isnull().sum())/(df.shape[0])\*100),2))  
 msno.matrix(df);

# Drop the column 'location' from the training dataset  
train\_df=train\_df.drop(['location'],axis=1)

# Lets find the length of the 'text' for each row and add a new cloumn to train dataframe   
train\_df['text\_length'] = train\_df['text'].apply(lambda x : len(x))  
train\_df.head(4)

id keyword text target \  
0 1 NaN Our Deeds are the Reason of this # earthquake... 1   
1 4 NaN Forest fire near La Ronge Sask . Canada 1   
2 5 NaN All residents asked to ' shelter in place ' ... 1   
3 6 NaN 13,000 people receive # wildfires evacuation ... 1   
  
 text\_length   
0 71   
1 40   
2 139   
3 67

train\_df.text\_length.describe()

count 7613.000000  
mean 92.453960  
std 34.833534  
min 6.000000  
25% 66.000000  
50% 94.000000  
75% 120.000000  
max 203.000000  
Name: text\_length, dtype: float64

# A disaster tweet exmaple  
train\_df[train\_df['target']==1]['text'][10:20]

10 Three people died from the heat wave so far  
11 Haha South Tampa is getting flooded hah - WAI...  
12 # raining # flooding # Florida # TampaBay ...  
13 # Flood in Bago Myanmar # We arrived Bago  
14 Damage to school bus on 80 in multi car crash ...  
31 @ bbcmtd Wholesale Markets ablaze   
33 # AFRICANBAZE : Breaking news : Nigeria flag...  
37 INEC Office in Abia Set Ablaze -   
38 Barbados # Bridgetown JAMAICA Two cars set a...  
46 How the West was burned : Thousands of wildfi...  
Name: text, dtype: object

#A non-disaster tweet example  
train\_df[train\_df['target']==0]['text'][10:20]

25 No way . I cannot eat that shit  
26 Was in NYC last week !   
27 Love my girlfriend  
28 Cooool : )   
29 Do you like pasta ?   
30 The end !   
32 We always try to bring the heavy . # metal ...  
34 Crying out for more ! Set me ablaze  
35 On plus side LOOK AT THE SKY LAST NIGHT IT WAS...  
36 @ PhDSquares # mufc they have built so much ...  
Name: text, dtype: object

#Lets have a quick look of the text data  
train\_df['text'][:5]

0 Our Deeds are the Reason of this # earthquake...  
1 Forest fire near La Ronge Sask . Canada  
2 All residents asked to ' shelter in place ' ...  
3 13,000 people receive # wildfires evacuation ...  
4 Just got sent this photo from Ruby # Alaska a...  
Name: text, dtype: object

# Create a function to clean the text  
  
def clean\_text(text):  
  
 '''  
 Input- 'text' to be cleaned  
   
 Output- Convert input 'text' to lowercase,remove square brackets,links,punctuation  
 and words containing numbers. Return clean text.  
   
 '''  
 text = text.lower()  
 text = re.sub('\[.\*?\]', '', text)  
 text = re.sub('https?://\S+|www\.\S+', '', text)  
 text = re.sub('<.\*?>+', '', text)  
 text = re.sub('[%s]' % re.escape(string.punctuation), '', text)  
 text = re.sub('\n', '', text)  
 text = re.sub('\w\*\d\w\*', '', text)  
 return text

# Lets apply the clean\_text function to both test and training datasets copies  
train\_df1=train\_df.copy()  
test\_df1=test\_df.copy()  
train\_df1['text'] = train\_df1['text'].apply(lambda x: clean\_text(x))  
test\_df1['text'] = test\_df1['text'].apply(lambda x: clean\_text(x))

#Lets look cleaned text data  
def text\_after\_preprocess(before\_text,after\_text):  
   
 '''  
 Input- before\_text=text column before cleanup  
 after\_text= text column after cleanup  
 Output- print before and after text to compare how it looks after cleanup  
   
 '''  
 print('-'\*60)  
 print('Text before cleanup')  
 print('-'\*60)  
 print(before\_text.head(5))  
 print('-'\*60)  
 print('Text after cleanup')  
 print('-'\*60)  
 print(after\_text.head(5))

text\_after\_preprocess(train\_df.text,train\_df1.text)

------------------------------------------------------------  
Text before cleanup  
------------------------------------------------------------  
0 Our Deeds are the Reason of this # earthquake...  
1 Forest fire near La Ronge Sask . Canada  
2 All residents asked to ' shelter in place ' ...  
3 13,000 people receive # wildfires evacuation ...  
4 Just got sent this photo from Ruby # Alaska a...  
Name: text, dtype: object  
------------------------------------------------------------  
Text after cleanup  
------------------------------------------------------------  
0 our deeds are the reason of this earthquake ...  
1 forest fire near la ronge sask canada  
2 all residents asked to shelter in place ar...  
3 people receive wildfires evacuation orders ...  
4 just got sent this photo from ruby alaska as...  
Name: text, dtype: object

text\_after\_preprocess(test\_df.text,test\_df1.text)

------------------------------------------------------------  
Text before cleanup  
------------------------------------------------------------  
0 Just happened a terrible car crash  
1 Heard about # earthquake is different cities,...  
2 there is a forest fire at spot pond, geese are...  
3 Apocalypse lighting . # Spokane # wildfires  
4 Typhoon Soudelor kills 28 in China and Taiwan  
Name: text, dtype: object  
------------------------------------------------------------  
Text after cleanup  
------------------------------------------------------------  
0 just happened a terrible car crash  
1 heard about earthquake is different cities s...  
2 there is a forest fire at spot pond geese are ...  
3 apocalypse lighting spokane wildfires  
4 typhoon soudelor kills in china and taiwan  
Name: text, dtype: object

# Example how tokenization of text works  
text = "Heard about #earthquake is different cities, stay safe everyone."  
  
tokenizer1 = nltk.tokenize.WordPunctTokenizer()  
tokenizer2 = nltk.tokenize.RegexpTokenizer(r'\w+')  
print("-"\*100)  
print("Example Text: ",text)  
print("-"\*100)  
  
print("Tokenization by punctuation:- ",tokenizer1.tokenize(text))  
print("Tokenization by regular expression:- ",tokenizer2.tokenize(text))

----------------------------------------------------------------------------------------------------  
Example Text: Heard about #earthquake is different cities, stay safe everyone.  
----------------------------------------------------------------------------------------------------  
Tokenization by punctuation:- ['Heard', 'about', '#', 'earthquake', 'is', 'different', 'cities', ',', 'stay', 'safe', 'everyone', '.']  
Tokenization by regular expression:- ['Heard', 'about', 'earthquake', 'is', 'different', 'cities', 'stay', 'safe', 'everyone']

# Lets Tokenize the training and the test dataset copies with RegEx tokenizer  
tokenizer = nltk.tokenize.RegexpTokenizer(r'\w+')  
train\_df1['text'] = train\_df1['text'].apply(lambda x: tokenizer.tokenize(x))  
test\_df1['text'] = test\_df1['text'].apply(lambda x: tokenizer.tokenize(x))

#Lets check tokenized text  
train\_df1['text'].head()

0 [our, deeds, are, the, reason, of, this, earth...  
1 [forest, fire, near, la, ronge, sask, canada]  
2 [all, residents, asked, to, shelter, in, place...  
3 [people, receive, wildfires, evacuation, order...  
4 [just, got, sent, this, photo, from, ruby, ala...  
Name: text, dtype: object

test\_df1['text'].head()

0 [just, happened, a, terrible, car, crash]  
1 [heard, about, earthquake, is, different, citi...  
2 [there, is, a, forest, fire, at, spot, pond, g...  
3 [apocalypse, lighting, spokane, wildfires]  
4 [typhoon, soudelor, kills, in, china, and, tai...  
Name: text, dtype: object

nltk.download('stopwords')  
#Create a funtion to remove stopwords  
def remove\_stopwords(text):  
   
 """  
 Input- text=text from which english stopwprds will be removed  
 Output- return text without english stopwords   
   
 """  
 words = [w for w in text if w not in stopwords.words('english')]  
 return words

[nltk\_data] Downloading package stopwords to  
[nltk\_data] C:\Users\lohith\AppData\Roaming\nltk\_data...  
[nltk\_data] Package stopwords is already up-to-date!

train\_df1['text'] = train\_df1['text'].apply(lambda x : remove\_stopwords(x))  
test\_df1['text'] = test\_df1['text'].apply(lambda x : remove\_stopwords(x))

train\_df1.text.head()

0 [deeds, reason, earthquake, may, allah, forgiv...  
1 [forest, fire, near, la, ronge, sask, canada]  
2 [residents, asked, shelter, place, notified, o...  
3 [people, receive, wildfires, evacuation, order...  
4 [got, sent, photo, ruby, alaska, smoke, wildfi...  
Name: text, dtype: object

test\_df1.text.head()

0 [happened, terrible, car, crash]  
1 [heard, earthquake, different, cities, stay, s...  
2 [forest, fire, spot, pond, geese, fleeing, acr...  
3 [apocalypse, lighting, spokane, wildfires]  
4 [typhoon, soudelor, kills, china, taiwan]  
Name: text, dtype: object

nltk.download('wordnet')  
# Stemming and Lemmatization examples  
text = "ran deduced dogs talking studies"  
  
tokenizer = nltk.tokenize.TreebankWordTokenizer()  
tokens = tokenizer.tokenize(text)  
  
# Stemmer  
stemmer = nltk.stem.PorterStemmer()  
print("Stemming the sentence: ", " ".join(stemmer.stem(token) for token in tokens))  
  
# Lemmatizer  
lemmatizer=nltk.stem.WordNetLemmatizer()  
print("Lemmatizing the sentence: ", " ".join(lemmatizer.lemmatize(token) for token in tokens))

[nltk\_data] Downloading package wordnet to  
[nltk\_data] C:\Users\lohith\AppData\Roaming\nltk\_data...  
[nltk\_data] Package wordnet is already up-to-date!

Stemming the sentence: ran deduc dog talk studi  
Lemmatizing the sentence: ran deduced dog talking study

# Lets combine text after processing it  
def combine\_text(text):  
   
 '''  
 Input-text= list cleand and tokenized text  
 Output- Takes a list of text and returns combined one large chunk of text.  
   
 '''  
 all\_text = ' '.join(text)  
 return all\_text

train\_df1['text'] = train\_df1['text'].apply(lambda x : combine\_text(x))  
test\_df1['text'] = test\_df1['text'].apply(lambda x : combine\_text(x))

train\_df1.head()

id keyword text target \  
0 1 NaN deeds reason earthquake may allah forgive us 1   
1 4 NaN forest fire near la ronge sask canada 1   
2 5 NaN residents asked shelter place notified officer... 1   
3 6 NaN people receive wildfires evacuation orders cal... 1   
4 7 NaN got sent photo ruby alaska smoke wildfires pou... 1   
  
 text\_length   
0 71   
1 40   
2 139   
3 67   
4 92

# Create a function to pre-process the tweets  
def pre\_process\_text\_combined(text):  
   
 """  
 Input- text= text to be pre-processed  
   
 Oputput- return cleaned and combined text to be vectorized for Machine learning.  
  
 """  
 #Initiate a tokenizer  
 tokenizer = nltk.tokenize.RegexpTokenizer(r'\w+')  
 # Clean the text using clean\_text function  
 cleaned\_txt = clean\_text(text)  
 tokenized\_text = tokenizer.tokenize(cleaned\_txt)  
 remove\_stopwords = [w for w in tokenized\_text if w not in stopwords.words('english')]  
 combined\_text = ' '.join(remove\_stopwords)  
 return combined\_text

# Create a function to pre-process the tweets  
def pre\_process\_text(text):  
 """  
 Input- text= text to be pre-processed  
   
 Oputput- return cleaned text to be vectrorized for Machine learning.  
  
 """  
 #Initiate a tokenizer  
 tokenizer = nltk.tokenize.RegexpTokenizer(r'\w+')  
 # Clean the text using clean\_text function  
 cleaned\_txt = clean\_text(text)  
 tokenized\_text = tokenizer.tokenize(cleaned\_txt)  
 remove\_stopwords = [w for w in tokenized\_text if w not in stopwords.words('english')]  
 return remove\_stopwords

# Text before pre-processing  
train\_df.text.head()

0 Our Deeds are the Reason of this # earthquake...  
1 Forest fire near La Ronge Sask . Canada  
2 All residents asked to ' shelter in place ' ...  
3 13,000 people receive # wildfires evacuation ...  
4 Just got sent this photo from Ruby # Alaska a...  
Name: text, dtype: object

# Lets pre-process train data text  
train\_df2=train\_df.copy()  
train\_df2['text'] = train\_df2['text'].apply(lambda x : pre\_process\_text\_combined(x))

# Text after pre-processing the text column  
train\_df2.head()

id keyword text target \  
0 1 NaN deeds reason earthquake may allah forgive us 1   
1 4 NaN forest fire near la ronge sask canada 1   
2 5 NaN residents asked shelter place notified officer... 1   
3 6 NaN people receive wildfires evacuation orders cal... 1   
4 7 NaN got sent photo ruby alaska smoke wildfires pou... 1   
  
 text\_length   
0 71   
1 40   
2 139   
3 67   
4 92

# Lets pre-process test data text  
test\_df2=test\_df.copy()  
test\_df2['text'] = test\_df2['text'].apply(lambda x : pre\_process\_text\_combined(x))

# Text after pre-processing the text column  
test\_df2.head(10)

id keyword location text  
0 0 NaN NaN happened terrible car crash  
1 2 NaN NaN heard earthquake different cities stay safe ev...  
2 3 NaN NaN forest fire spot pond geese fleeing across str...  
3 9 NaN NaN apocalypse lighting spokane wildfires  
4 11 NaN NaN typhoon soudelor kills china taiwan  
5 12 NaN NaN shaking earthquake  
6 21 NaN NaN probably still show life arsenal yesterday eh eh  
7 22 NaN NaN hey  
8 27 NaN NaN nice hat  
9 29 NaN NaN fuck

# Lets pre-process train data text  
train\_df3=train\_df.copy()  
train\_df3['text'] = train\_df3['text'].apply(lambda x : pre\_process\_text(x))

train\_df3.head()

id keyword text target \  
0 1 NaN [deeds, reason, earthquake, may, allah, forgiv... 1   
1 4 NaN [forest, fire, near, la, ronge, sask, canada] 1   
2 5 NaN [residents, asked, shelter, place, notified, o... 1   
3 6 NaN [people, receive, wildfires, evacuation, order... 1   
4 7 NaN [got, sent, photo, ruby, alaska, smoke, wildfi... 1   
  
 text\_length   
0 71   
1 40   
2 139   
3 67   
4 92

# Lets pre-process test data text  
test\_df3=test\_df.copy()  
test\_df3['text'] = test\_df3['text'].apply(lambda x : pre\_process\_text(x))

test\_df3.head()

id keyword location text  
0 0 NaN NaN [happened, terrible, car, crash]  
1 2 NaN NaN [heard, earthquake, different, cities, stay, s...  
2 3 NaN NaN [forest, fire, spot, pond, geese, fleeing, acr...  
3 9 NaN NaN [apocalypse, lighting, spokane, wildfires]  
4 11 NaN NaN [typhoon, soudelor, kills, china, taiwan]

# Vectorize the text using CountVectorizer  
count\_vectorizer = CountVectorizer()  
train\_cv = count\_vectorizer.fit\_transform(train\_df2['text'])  
test\_cv = count\_vectorizer.transform(test\_df2["text"])

# Vectorize the text using TFIDF  
tfidf = TfidfVectorizer(min\_df=2, max\_df=0.5, ngram\_range=(1, 2))  
train\_tf = tfidf.fit\_transform(train\_df2['text'])  
test\_tf = tfidf.transform(test\_df2["text"])

train\_df2['text']

0 deeds reason earthquake may allah forgive us  
1 forest fire near la ronge sask canada  
2 residents asked shelter place notified officer...  
3 people receive wildfires evacuation orders cal...  
4 got sent photo ruby alaska smoke wildfires pou...  
 ...   
7608 two giant cranes holding bridge collapse nearb...  
7609 aria ahrary thetawniest control wild fires cal...  
7610 volcano hawaii  
7611 police investigating e bike collided car littl...  
7612 latest homes razed northern california wildfir...  
Name: text, Length: 7613, dtype: object

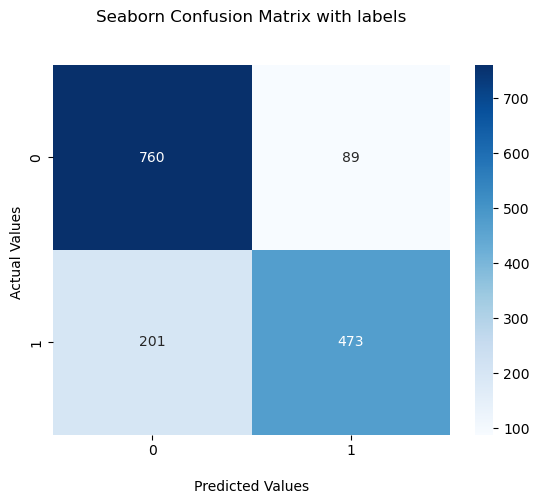
#Split the CountVector vectorized data into train and test datasets for model training and testing  
X\_train\_cv, X\_test\_cv, y\_train\_cv, y\_test\_cv =train\_test\_split(train\_cv,train\_df.target,test\_size=0.2,random\_state=2020)

#Define a function to fit and predict on training and test data sets  
def fit\_and\_predict(model,X\_train,y\_train,X\_test,y\_test):  
   
   
 # Fitting a simple Logistic Regression on Counts  
 clf = model  
 clf.fit(X\_train, y\_train)  
 predictions=clf.predict(X\_test)  
 #check=clf.predict(count\_vectorizer.fit\_transform(train\_df2['text'].head(1)))  
 #print(check)  
 cf\_matrix = confusion\_matrix(y\_test,predictions)  
 print(cf\_matrix)  
 ax = sns.heatmap(cf\_matrix, annot=True, fmt="d",cmap='Blues')  
  
 ax.set\_title('Seaborn Confusion Matrix with labels\n\n');  
  
 ax.set\_xlabel('\nPredicted Values')  
  
 ax.set\_ylabel('Actual Values ');  
  
 ax.xaxis.set\_ticklabels(['0','1'])  
  
 ax.yaxis.set\_ticklabels(['0','1'])  
  
 plt.show()  
   
 print(classification\_report(y\_test,predictions))  
 print('-'\*50)  
 print("{}" .format(model))  
 print('-'\*50)  
 print('Accuracy of classifier on training set:{}%'.format(round(clf.score(X\_train, y\_train)\*100)))  
 print('-'\*50)  
 print('Accuracy of classifier on test set:{}%' .format(round(accuracy\_score(y\_test,predictions)\*100)))  
 print('-'\*50)  
 #print(clf)  
 return clf

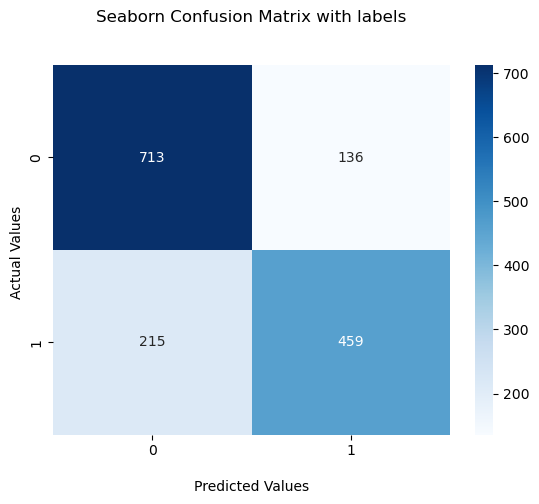
# Create a list of the regression models to be used  
models=[LogisticRegression(C=1.0),DecisionTreeClassifier(),  
 RandomForestClassifier()]

# Loop through the list of models and use 'fit\_and\_predict()' function to trian and make predictions  
prepared\_models = []  
for model in models:  
 prepared\_models.append(fit\_and\_predict(model,X\_train\_cv, y\_train\_cv,X\_test\_cv,y\_test\_cv))

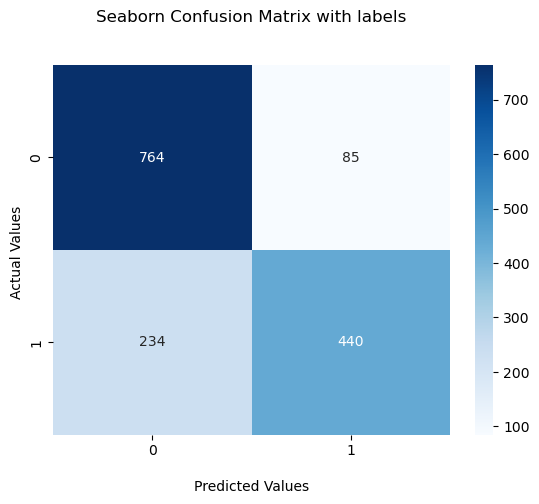
[[760 89]  
 [201 473]]



precision recall f1-score support  
  
 0 0.79 0.90 0.84 849  
 1 0.84 0.70 0.77 674  
  
 accuracy 0.81 1523  
 macro avg 0.82 0.80 0.80 1523  
weighted avg 0.81 0.81 0.81 1523  
  
--------------------------------------------------  
LogisticRegression()  
--------------------------------------------------  
Accuracy of classifier on training set:96%  
--------------------------------------------------  
Accuracy of classifier on test set:81%  
--------------------------------------------------  
[[713 136]  
 [215 459]]



precision recall f1-score support  
  
 0 0.77 0.84 0.80 849  
 1 0.77 0.68 0.72 674  
  
 accuracy 0.77 1523  
 macro avg 0.77 0.76 0.76 1523  
weighted avg 0.77 0.77 0.77 1523  
  
--------------------------------------------------  
DecisionTreeClassifier()  
--------------------------------------------------  
Accuracy of classifier on training set:99%  
--------------------------------------------------  
Accuracy of classifier on test set:77%  
--------------------------------------------------  
[[764 85]  
 [234 440]]



precision recall f1-score support  
  
 0 0.77 0.90 0.83 849  
 1 0.84 0.65 0.73 674  
  
 accuracy 0.79 1523  
 macro avg 0.80 0.78 0.78 1523  
weighted avg 0.80 0.79 0.79 1523  
  
--------------------------------------------------  
RandomForestClassifier()  
--------------------------------------------------  
Accuracy of classifier on training set:99%  
--------------------------------------------------  
Accuracy of classifier on test set:79%  
--------------------------------------------------

input = pre\_process\_text\_combined("Im going to college")  
#print(prepared\_models)  
  
#input.apply(lambda x : pre\_process\_text\_combined(x))  
  
inp = count\_vectorizer.transform([input])  
  
for model in prepared\_models:  
 print(model)  
 print(model.predict(inp))

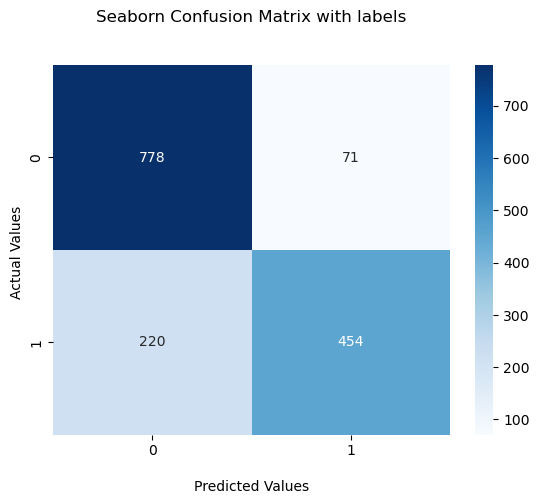
LogisticRegression()

---------------------------------------------------------------------------  
ValueError Traceback (most recent call last)  
Cell In[55], line 10  
 8 for model in prepared\_models:  
 9 print(model)  
---> 10 print(model.predict(inp))  
  
File C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear\_model\\_base.py:425, in LinearClassifierMixin.predict(self, X)  
 411 def predict(self, X):  
 412 """  
 413 Predict class labels for samples in X.  
 414   
 (...)  
 423 Vector containing the class labels for each sample.  
 424 """  
--> 425 scores = self.decision\_function(X)  
 426 if len(scores.shape) == 1:  
 427 indices = (scores > 0).astype(int)  
  
File C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear\_model\\_base.py:407, in LinearClassifierMixin.decision\_function(self, X)  
 387 """  
 388 Predict confidence scores for samples.  
 389   
 (...)  
 403 this class would be predicted.  
 404 """  
 405 check\_is\_fitted(self)  
--> 407 X = self.\_validate\_data(X, accept\_sparse="csr", reset=False)  
 408 scores = safe\_sparse\_dot(X, self.coef\_.T, dense\_output=True) + self.intercept\_  
 409 return scores.ravel() if scores.shape[1] == 1 else scores  
  
File C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py:585, in BaseEstimator.\_validate\_data(self, X, y, reset, validate\_separately, \*\*check\_params)  
 582 out = X, y  
 584 if not no\_val\_X and check\_params.get("ensure\_2d", True):  
--> 585 self.\_check\_n\_features(X, reset=reset)  
 587 return out  
  
File C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py:400, in BaseEstimator.\_check\_n\_features(self, X, reset)  
 397 return  
 399 if n\_features != self.n\_features\_in\_:  
--> 400 raise ValueError(  
 401 f"X has {n\_features} features, but {self.\_\_class\_\_.\_\_name\_\_} "  
 402 f"is expecting {self.n\_features\_in\_} features as input."  
 403 )  
  
ValueError: X has 15661 features, but LogisticRegression is expecting 11180 features as input.

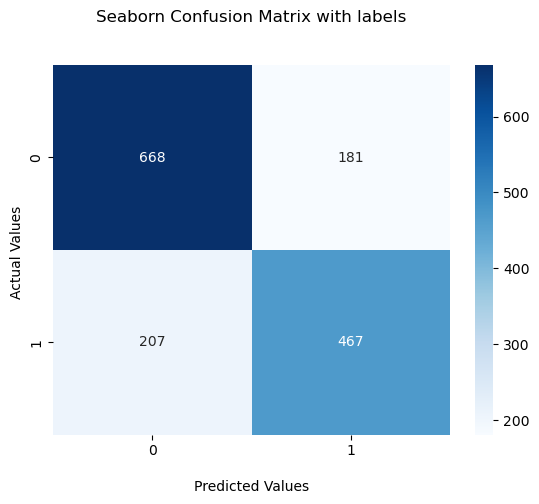
# Split the TFDIF vectorized data into train and test datasets for model training and testing  
X\_train\_tf, X\_test\_tf, y\_train\_tf, y\_test\_tf =train\_test\_split(train\_tf,train\_df.target,test\_size=0.2,random\_state=2020)

# Loop through the list of models and use 'fit\_and\_predict()' function to train and make predictions on the TFDIF vectororized data  
for model in models:  
 fit\_and\_predict(model,X\_train\_tf, y\_train\_tf,X\_test\_tf,y\_test\_tf)

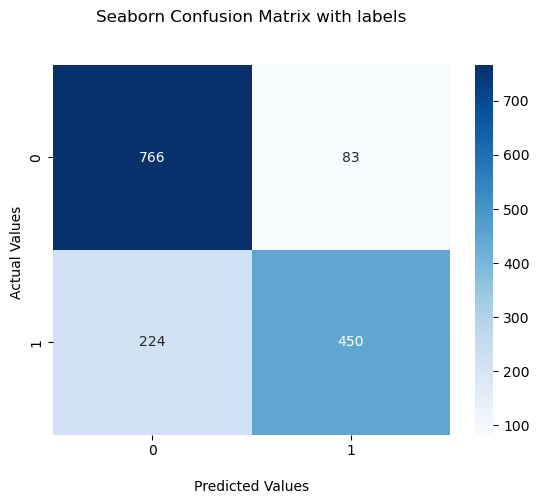
[[778 71]  
 [220 454]]



precision recall f1-score support  
  
 0 0.78 0.92 0.84 849  
 1 0.86 0.67 0.76 674  
  
 accuracy 0.81 1523  
 macro avg 0.82 0.79 0.80 1523  
weighted avg 0.82 0.81 0.80 1523  
  
--------------------------------------------------  
LogisticRegression()  
--------------------------------------------------  
Accuracy of classifier on training set:88%  
--------------------------------------------------  
Accuracy of classifier on test set:81%  
--------------------------------------------------  
[[668 181]  
 [207 467]]



precision recall f1-score support  
  
 0 0.76 0.79 0.77 849  
 1 0.72 0.69 0.71 674  
  
 accuracy 0.75 1523  
 macro avg 0.74 0.74 0.74 1523  
weighted avg 0.74 0.75 0.74 1523  
  
--------------------------------------------------  
DecisionTreeClassifier()  
--------------------------------------------------  
Accuracy of classifier on training set:98%  
--------------------------------------------------  
Accuracy of classifier on test set:75%  
--------------------------------------------------  
[[766 83]  
 [224 450]]



precision recall f1-score support  
  
 0 0.77 0.90 0.83 849  
 1 0.84 0.67 0.75 674  
  
 accuracy 0.80 1523  
 macro avg 0.81 0.78 0.79 1523  
weighted avg 0.80 0.80 0.79 1523  
  
--------------------------------------------------  
RandomForestClassifier()  
--------------------------------------------------  
Accuracy of classifier on training set:98%  
--------------------------------------------------  
Accuracy of classifier on test set:80%  
--------------------------------------------------

# Printing model performance results for countvectorizer.  
results\_dict={'Classifier':['Logistic regression', 'DecisionTreeClassifier',  
 'RandomForestClassifier'],  
 'F1-Score':[0.81, 0.76, 0.78],'Accuracy':['81%','76%','79%']}   
results=pd.DataFrame(results\_dict)  
results

Classifier F1-Score Accuracy  
0 Logistic regression 0.81 81%  
1 DecisionTreeClassifier 0.76 76%  
2 RandomForestClassifier 0.78 79%

# Printing model performance results for TF-IDF vectorizer.  
results\_dict={'Classifier':['Logistic regression', 'DecisionTreeClassifier',  
 'RandomForestClassifier'],  
 'F1-Score':[0.80, 0.75, 0.80],'Accuracy':['81%','75%','80%']}   
results=pd.DataFrame(results\_dict)  
results

Classifier F1-Score Accuracy  
0 Logistic regression 0.80 81%  
1 DecisionTreeClassifier 0.75 75%  
2 RandomForestClassifier 0.80 80%