Disaster Tweets Analysis

Today, social media has become a very important source of information. Like other platforms, Twitter has developed into a platform to get the first hand information from the people directly involved in certain events. The informations and updates of natural disasters can be easily collected from multiple official sources; however, the tweets about such disasters provide the ground realities from the people affected by them.

So, in the analysis below we will use NLP and machine learning to generate some meaningful insights from the given tweets data. We will look at the most common disasters that were tweeted, and the popular words and hastags that are used in such tweets. We will also use ML to identify if a tweet is actually about a disater or not.

Importing libraries for analysis

In [2]:

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
```

In [3]:

```
#!pip install wordcloud
from wordcloud import WordCloud
```

In [4]:

```
import regex as re
#!pip install emoji
import emoji
```

In [5]:

```
from nltk import tokenize
from nltk.tokenize import TweetTokenizer
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
```

Importing Tweets Dataset

In [31]:

```
tweets = pd.read_csv('tweets.csv')
tweets.head()
```

Out[31]:

	id	keyword	location	text	target
0	0	ablaze	NaN	Communal violence in Bhainsa, Telangana. "Ston...	1
1	1	ablaze	NaN	Telangana: Section 144 has been imposed in Bha...	1
2	2	ablaze	New York City	Arsonist sets cars ablaze at dealership https:...	1
3	3	ablaze	Morgantown, WV	Arsonist sets cars ablaze at dealership https:...	1
4	4	ablaze	NaN	"Lord Jesus, your love brings freedom and pard...	0

Id: Unique identifier for each tweet

text: Shows the text of the tweet

location: Shows the location the tweet was sent from

keyword: Keyword from the tweet to identify a disaster

target: Shows whether a tweet is about a real disaster (1) or not (0)

Data Overview

In [32]:

```
tweets['text'].values[4]
```

Out[32]:

```
'"Lord Jesus, your love brings freedom and pardon. Fill me with your Holy Spirit and set my heart ablaze with your
l… https://t.co/VlTznnPNi8' (https://t.co/VlTznnPNi8')
```

In [33]:

```
tweets.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11370 entries, 0 to 11369
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   id        11370 non-null  int64 
 1   keyword   11370 non-null  object
 2   location  7952 non-null   object
 3   text      11370 non-null  object
 4   target    11370 non-null  int64 
dtypes: int64(2), object(3)
memory usage: 444.3+ KB
```

Null count

In [34]:

```
tweets.isna().sum()/tweets.count()
```

Out[34]:

```
id          0.000000
keyword     0.000000
location    0.429829
text        0.000000
target      0.000000
dtype: float64
```

About 43% of location information is blank. Besides, on further analysis of location data below it was found that the location data was inconsistent. For example, the location had only country or only city, or both country and city, or sometimes meaningless text. So, it is a good idea to drop this column.

Data cleaning

Drop unrequired columns

In [35]:

```
tweets['location'].value_counts()
```

Out[35]:

```
United States            96
Australia                83
London, England          81
UK                       77
India                    74
                         ..
Great State of Texas      1
Karatina, Kenya           1
The internet or the gym   1
Reston, VA                1
auroraborealis            1
Name: location, Length: 4504, dtype: int64
```

In [36]:

```
tweets.drop(columns = ['location'], inplace = True)
```

In [37]:

```
tweets.head(3)
```

Out[37]:

	id	keyword	text	target
0	0	ablaze	Communal violence in Bhainsa, Telangana. "Ston...	1
1	1	ablaze	Telangana: Section 144 has been imposed in Bha...	1
2	2	ablaze	Arsonist sets cars ablaze at dealership https:...	1

Extracting HashTags

In [38]:

```
tkn = TweetTokenizer()
```

In [39]:

```
tweets['tokenized'] = tweets['text'].apply(lambda words: [w for w in tkn.tokenize(words)])
```

In [40]:

```
tweets['hashtags'] = tweets['tokenized'].apply(lambda words: [re.sub(r'#(.+)',r'\1',w) for w in words if re.match('#.+',w)!=None
```

In [41]:

```
tweets.drop(columns='tokenized', inplace=True)
```

In [42]:

```
tweets[tweets['hashtags'].apply(lambda x: len(x)>0)].head()
```

Out[42]:

	id	keyword	text	target	hashtags
10	10	ablaze	Images showing the havoc caused by the #Camero...	1	[Cameroon, Oku]
11	11	ablaze	Social media went bananas after Chuba Hubbard ...	0	[okstate]
13	13	ablaze	Under #MamataBanerjee political violence &...	1	[MamataBanerjee]
15	15	ablaze	Images showing the havoc caused by the #Camero...	1	[Cameroon, Oku]
16	16	ablaze	No cows today but our local factory is sadly s...	1	[REDJanuary2020]

Hashtags frequency

In [43]:

```
from collections import Counter
tags_freq = Counter()

for word in tweets['hashtags']:
    tags_freq.update(word)

print(tags_freq.most_common(50))
```

```
[('Iran', 24), ('StormBrendan', 22), ('China', 14), ('earthquake', 13), ('iHeartAwards', 12), ('NeelumValley', 1
1), ('TaalVolcano', 11), ('PuertoRico', 9), ('Kashmir', 9), ('bushfires', 9), ('vichazmat', 9), ('Hitchin', 9),
('auspol', 8), ('NowPlaying', 8), ('Israel', 8), ('abc730', 8), ('TaalEruption2020', 8), ('Australia', 8), ('Ben_W
ounds', 8), ('Libya', 7), ('BTSARMY', 6), ('BestFanArmy', 6), ('Pakistan', 6), ('M20', 6), ('Maidstone', 6), ('new
s', 6), ('Palestinian', 6), ('MorningCrossfire', 6), ('Ireland', 6), ('HongKong', 6), ('RT', 5), ('Yemen', 5), ('A
ustraliaBushfires', 5), ('ITrustBernie', 5), ('Prisoners', 5), ('WestBank', 5), ('WarCrimes', 5), ('nuclear', 5),
('SPC', 5), ('Cameroon', 4), ('USA', 4), ('ClimateChange', 4), ('Iraq', 4), ('Iranian', 4), ('JUSTICEFORX1', 4),
('thriller', 4), ('Flybe', 4), ('RefundWarren', 4), ('AustralianBushfires', 4), ('UhuruAddress', 4)]
```

The top 50 hashtags as seen above provide very limited useful information about the natural disasters. So, it is hard to rely on hashtags information to make any inference about any diaster.

Cleaning tweets' text

In [44]:

```
def data_prep(tweet):
    tweet = tweet.lower()
    tweet = re.sub("@[A-Za-z0-9]+","",tweet) #Remove @ sign
    tweet = re.sub(r'https?://\S+', '', tweet) # remove URLs
    tweet = re.sub(r'[^a-zA-Z0-9\s]', '', tweet) # remove punctuation and special characters
    tweet = re.sub(r"#", "", tweet)

    return tweet
```

In [45]:

```
tweets['text1'] = tweets['text'].apply(data_prep)
```

In [46]:

```
tweets.head(3)
```

Out[46]:

	id	keyword	text	target	hashtags	text1
0	0	ablaze	Communal violence in Bhainsa, Telangana. "Ston...	1	[]	communal violence in bhainsa telangana stones ...
1	1	ablaze	Telangana: Section 144 has been imposed in Bha...	1	[]	telangana section 144 has been imposed in bhai...
2	2	ablaze	Arsonist sets cars ablaze at dealership https:...	1	[]	arsonist sets cars ablaze at dealership

Remove duplicate tweets

In [47]:

```
tweets.drop_duplicates(['text1'], keep = 'first', inplace = True)
```

In [48]:

```
tweets.shape
```

Out[48]:

```
(10965, 6)
```

Word Tokenization

Tokenization is used in natural language processing to split paragraphs and sentences into smaller units that can be more easily assigned meaning. The first step of the NLP process is gathering the data (a sentence) and breaking it into understandable parts (words).

In [49]:

```
tnk = TweetTokenizer()
```

In [50]:

```
tweets['t_words'] = tweets['text1'].apply(tnk.tokenize)
```

In [51]:

```
tweets.head()
```

Out[51]:

	id	keyword	text	target	hashtags	text1	t_words
0	0	ablaze	Communal violence in Bhainsa, Telangana. "Ston...	1	[]	communal violence in bhainsa telangana stones ...	[communal, violence, in, bhainsa, telangana, s...
1	1	ablaze	Telangana: Section 144 has been imposed in Bha...	1	[]	telangana section 144 has been imposed in bhai...	[telangana, section, 144, has, been, imposed, ...
2	2	ablaze	Arsonist sets cars ablaze at dealership https:...	1	[]	arsonist sets cars ablaze at dealership	[arsonist, sets, cars, ablaze, at, dealership]
3	3	ablaze	Arsonist sets cars ablaze at dealership https:...	1	[]	arsonist sets cars ablaze at dealership	[arsonist, sets, cars, ablaze, at, dealership]
4	4	ablaze	"Lord Jesus, your love brings freedom and pard...	0	[]	lord jesus your love brings freedom and pardon...	[lord, jesus, your, love, brings, freedom, and...

Stopwords

In NLP and text mining applications, stop words are used to eliminate unimportant words, allowing applications to focus on the important words instead.

In [52]:

```
stpwrd = stopwords.words('english')
```

In [53]:

```
len(stpwrd)
```

Out[53]:

179

In [54]:

```
tweets['filtered'] = tweets['t_words'].apply(lambda words: [word for word in words if word not in stpwrd])
```

In [55]:

```
tweets.head(3)
```

Out[55]:

	id	keyword	text	target	hashtags	text1	t_words	filtered
0	0	ablaze	Communal violence in Bhainsa, Telangana. "Ston...	1	[]	communal violence in bhainsa telangana stones ...	[communal, violence, in, bhainsa, telangana, s...	[communal, violence, bhainsa, telangana, stone...
1	1	ablaze	Telangana: Section 144 has been imposed in Bha...	1	[]	telangana section 144 has been imposed in bhai...	[telangana, section, 144, has, been, imposed, ...	[telangana, section, 144, imposed, bhainsa, ja...
2	2	ablaze	Arsonist sets cars ablaze at dealership https:...	1	[]	arsonist sets cars ablaze at dealership	[arsonist, sets, cars, ablaze, at, dealership]	[arsonist, sets, cars, ablaze, dealership]

Lemmatization

Lemmatization is a text pre-processing technique used in natural language processing (NLP) models to break a word down to its root meaning to identify similarities. For example, a lemmatization algorithm would reduce the word better to its root word, or lemme, good.

In [56]:

```
lem = WordNetLemmatizer()

tweets['words'] = tweets['filtered'].apply(lambda words: [lem.lemmatize(w) for w in words])

tweets.head()
```

Out[56]:

	id	keyword	text	target	hashtags	text1	t_words	filtered	words
0	0	ablaze	Communal violence in Bhainsa, Telangana. "Ston...	1	[]	communal violence in bhainsa telangana stones ...	[communal, violence, in, bhainsa, telangana, s...	[communal, violence, bhainsa, telangana, stone...	[communal, violence, bhainsa, telangana, stone...
1	1	ablaze	Telangana: Section 144 has been imposed in Bha...	1	[]	telangana section 144 has been imposed in bhai...	[telangana, section, 144, has, been, imposed, ...	[telangana, section, 144, imposed, bhainsa, ja...	[telangana, section, 144, imposed, bhainsa, ja...
2	2	ablaze	Arsonist sets cars ablaze at dealership https:...	1	[]	arsonist sets cars ablaze at dealership	[arsonist, sets, cars, ablaze, at, dealership]	[arsonist, sets, cars, ablaze, dealership]	[arsonist, set, car, ablaze, dealership]
3	3	ablaze	Arsonist sets cars ablaze at dealership https:...	1	[]	arsonist sets cars ablaze at dealership	[arsonist, sets, cars, ablaze, at, dealership]	[arsonist, sets, cars, ablaze, dealership]	[arsonist, set, car, ablaze, dealership]
4	4	ablaze	"Lord Jesus, your love brings freedom and pard...	0	[]	lord jesus your love brings freedom and pardon...	[lord, jesus, your, love, brings, freedom, and...	[lord, jesus, love, brings, freedom, pardon, f...	[lord, jesus, love, brings, freedom, pardon, f...

Cleaning Keywords

Keywords shows the word related to disasters. However, in case of some tweets which are not related to disaster, the keyword can be misleading. So below analysis will show top keywords.

In [57]:

```
tweets['keyword'] = tweets['keyword'].apply(lambda x : x.replace("%20"," "))
tweets['root_keyword'] = tweets['keyword'].apply(lambda words: " ".join([lem.lemmatize(w) for w in words.split(" ")]))
```

In [58]:

```
tweets[tweets['keyword'] != tweets['root_keyword']][['keyword','root_keyword']].drop_duplicates()
```

Out[58]:

	keyword	root_keyword
1404	body bags	body bag
1682	buildings burning	building burning
1750	buildings on fire	building on fire
1924	bush fires	bush fire
1994	casualties	casualty
3178	deaths	death
4830	emergency services	emergency service
5351	fatalities	fatality
5645	first responders	first responder
5693	flames	flame
5913	floods	flood
5989	forest fires	forest fire
6634	hostages	hostage
6796	injuries	injury
8451	refugees	refugee
8592	rescuers	rescuer
9024	screams	scream
9243	sirens	siren
9921	survivors	survivor
10921	weapons	weapon
11012	wild fires	wild fire
11185	wounds	wound

In [59]:

```
tweets['root_keyword'].replace('building burning','building on fire', inplace=True)
```

In [60]:

```
tweets[tweets['root_keyword']=='building burning']
```

Out[60]:

id	keyword	text	target	hashtags	text1	t_words	filtered	words	root_keyword

Keywords analysis

In [61]:

```
disasters = tweets[tweets['target']==1][['root_keyword','words']]
disasters.head()
```

Out[61]:

	root_keyword	words
0	ablaze	[communal, violence, bhainsa, telangana, stone...
1	ablaze	[telangana, section, 144, imposed, bhainsa, ja...
2	ablaze	[arsonist, set, car, ablaze, dealership]
3	ablaze	[arsonist, set, car, ablaze, dealership]
6	ablaze	[several, house, set, ablaze, ngemsibaa, villa...

In [62]:

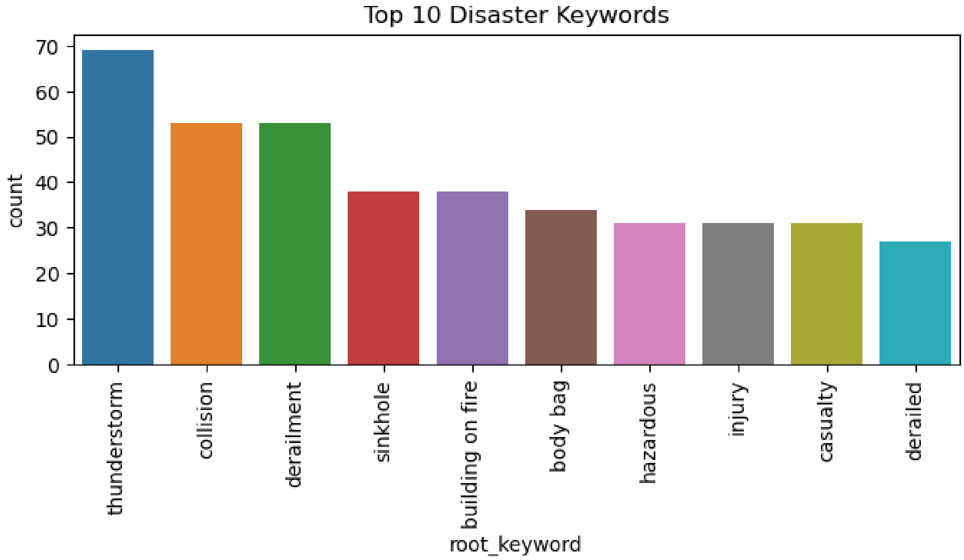
```
top_10_disaster = disasters['root_keyword'].value_counts().to_frame().reset_index()[:10]
top_10_disaster.columns =  ['root_keyword','count']
top_10_disaster
```

Out[62]:

	root_keyword	count
0	thunderstorm	69
1	collision	53
2	derailment	53
3	sinkhole	38
4	building on fire	38
5	body bag	34
6	hazardous	31
7	injury	31
8	casualty	31
9	derailed	27

In [63]:

```
plt.figure(figsize=(8,3))
sns.barplot(data = top_10_disaster, x= 'root_keyword', y='count')
plt.xticks(rotation = 90)
plt.title("Top 10 Disaster Keywords")
plt.show()
```



In [64]:

```
disaster_words = ''
disaster_words += " ".join(disasters['root_keyword'])+" "

wordcloud = WordCloud(width = 900, height = 400,
                background_color ='white',
                stopwords = stpwrd,
                min_font_size = 10).generate(disaster_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

In [65]:

```
others = tweets[tweets['target']==0][['text1','root_keyword','words']]

top_10_other = others['root_keyword'].value_counts().to_frame().reset_index()[:10]
top_10_other.columns =  ['root_keyword','count']
top_10_other
```

Out[65]:

	root_keyword	count
0	death	125
1	hostage	121
2	weapon	108
3	body bag	98
4	fatality	98
5	building on fire	93
6	siren	84
7	injury	79
8	fear	76
9	mass murder	76

Keywords like 'Mass Murder' do sound like disaster. However, if you look at the original tweets, those terms are not real disasters, they are just used as expressions, hypothetical situtations or exaggerations!

In [66]:

```
others[others['root_keyword']=='mass murder']['text1'][7316]
```

Out[66]:

```
'trumps actually a genius he takes out the mass murder while obama funded him '
```

In [67]:

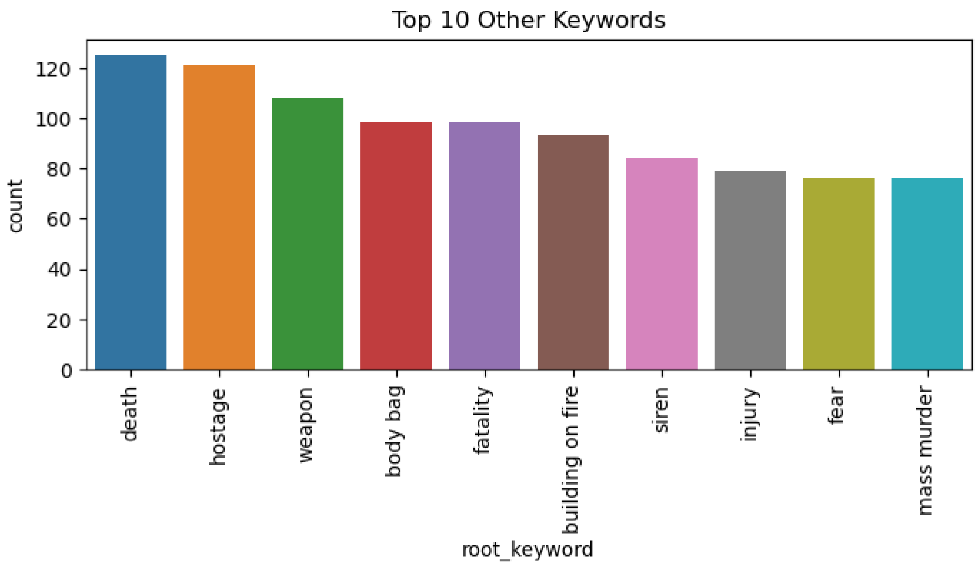
```
others[others['root_keyword']=='building on fire']['text1'][1683]
```

Out[67]:

```
'how many burning buildings have you run into to save someone'
```

In [68]:

```
plt.figure(figsize=(8,3))
sns.barplot(data = top_10_other, x= 'root_keyword', y='count')
plt.xticks(rotation = 90)
plt.title("Top 10 Other Keywords")
plt.show()
```



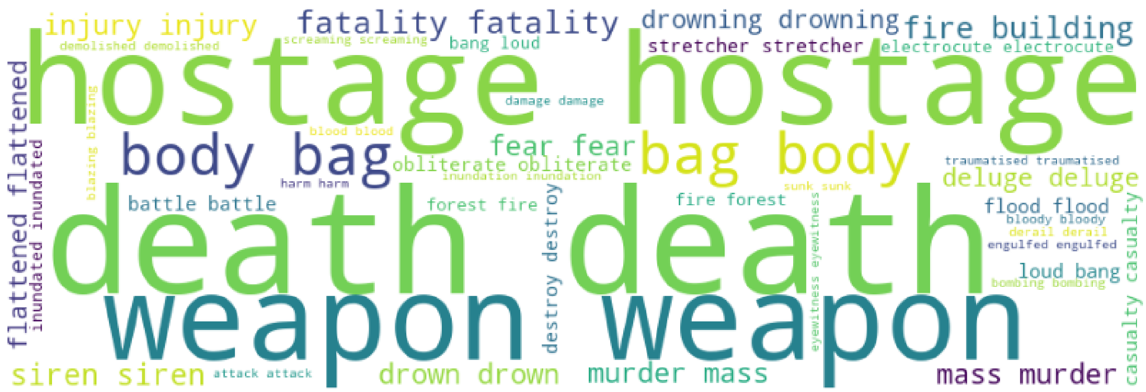
In [69]:

```
disaster_words = ''
disaster_words += " ".join(others['root_keyword'])+" "

wordcloud = WordCloud(width = 900, height = 300,
                background_color ='white',
                stopwords = stpwrd,
                min_font_size = 10).generate(disaster_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



TF-IDF vectorizer

TF-IDF (Term Frequency - Inverse Document Frequency) is a handy algorithm that uses the frequency of words to determine how relevant those words are to a given document. It's a relatively simple but intuitive approach to weighting words, allowing it to act as a great jumping off point for a variety of tasks.

In [70]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [47]:

```
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(tweets['text1'])
vectorizer.get_feature_names_out()
```

Out[47]:

```
array(['000009', '0019', '007', ..., 'zuma', 'zurich', 'zw'], dtype=object)
```

Train Test Split

In [48]:

```
X = X.toarray()
y = tweets['target']
```

In [49]:

```
from sklearn.model_selection import train_test_split
```

In [50]:

```
X_train, X_test, y_train, y_test=train_test_split(X, y, train_size=0.8)
```

Modelling

Logistic Regression

Logistic regression is a supervised machine learning algorithm mainly used for classification tasks where the goal is to predict the probability that an instance of belonging to a given class or not. It is a kind of statistical algorithm, which analyze the relationship between a set of independent variables and the dependent binary variables. It is a powerful tool for decision-making.

In [51]:

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(penalty='l2',class_weight='balanced', verbose=1)
```

In [62]:

```
lr.fit(X_train,y_train)
y_hat_lr = lr.predict(X_test)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    6.5s finished
```

In [63]:

```
print(y_hat_lr.shape, y_test.shape)
```

```
(2193,) (2193,)
```

In [64]:

```
from sklearn import metrics

metrics.accuracy_score(y_test, y_hat_lr)
```

Out[64]:

```
0.8873689010487916
```

In [65]:

```
metrics.confusion_matrix(y_test, y_hat_lr)
```

Out[65]:

```
array([[1641,  145],
       [ 102,  305]], dtype=int64)
```

Naive Bayes

The Naive Bayes family of statistical algorithms are some of the most used algorithms in text classification and text analysis , and one of the members of that family is Multinational Naive Bayes (MNB) with a huge advantage, that you can get really good results even when your data set isn’t very large (a couple of thousand tagged samples) and computational resources are scarce.

In [58]:

```
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB()         #create instance
```

In [59]:

```
nb.fit(X_train, y_train)  #fit model
y_hat_nb = nb.predict(X_test)     #make prediction for entire testing set

y_hat_nb[0:10]
```

Out[59]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

In [66]:

```
metrics.accuracy_score(y_test, y_hat_nb)
```

Out[66]:

```
0.8331053351573188
```

In [67]:

```
metrics.confusion_matrix(y_test, y_hat_nb)
```

Out[67]:

```
array([[1782,    4],
       [ 362,   45]], dtype=int64)
```

Logistic Regression Model performed slightly better than Naive Bayes Model.

In [ ]:

```

```