In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
train=pd.read_csv("train.csv")
test=pd.read_csv("test.csv")
```

In [3]:
```
#displaying train data
train.head()
```

Out[3]:

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

In [4]:
```
train.info()
```

```
<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
```

1.Data Cleaning

In [5]:
```
#dropping location column as there is no relation for disaster msgs with location a
#Dropping the keword column also as the msg will be given to the model to classify
train.drop(columns=['location','keyword'],inplace=True)
train = train.dropna()
train.head()
```

Out[5]:

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

In [6]:
```
train.info()
```

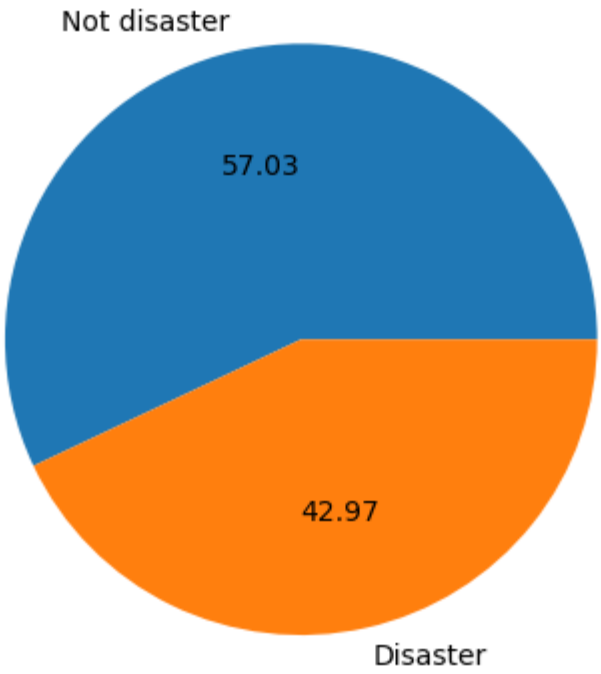
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7613 entries, 0 to 7612
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   id      7613 non-null   int64
 1   text    7613 non-null   object
 2   target  7613 non-null   int64
dtypes: int64(2), object(1)
memory usage: 178.6+ KB
```

In [7]:
```
#checking for duplicates
train.duplicated().sum()
```

Out[7]:
```
0
```

2.EDA

In [8]:
```
#plotting graph to check the balance in the data
import matplotlib.pyplot as plt
plt.pie(train['target'].value_counts(),labels=['Not disaster','Disaster'],autopct='
plt.show()
```



Performing NLP and Data Preprocessing

In [9]:
```
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import TweetTokenizer
import emoji
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
import string

stop_words = set(stopwords.words('english'))
tk = TweetTokenizer()
lemmatizer = WordNetLemmatizer()
```

In [10]:
```
def transform_text(text):
    # Remove URLs
    text = re.sub(r'http\S+', '', text)
    # Remove user mentions
    text = re.sub(r"@\S+", "", text)
    # Remove punctuation
    text = re.sub(f"[{string.punctuation}]", "", text)
    # Remove emojis
    text = emoji.emojize(text, variant='emoji_type')
    # Converting the text to lowercase
    text = text.lower()
    # Tokenize the text
    words = tk.tokenize(text)
    # Lemmatize the text
    words = [lemmatizer.lemmatize(w) for w in words]
    # Remove stop words
    words = [w for w in words if w not in stop_words]
    # Join the tokens back together
    cleaned_text = ' '.join(words)

    return cleaned_text
```

In [13]:
```
transform_text("more than 2000 families are destroyed in israel-hamaas war")
```

Out[13]:
```
'2000 family destroyed israelhamaas war'
```

In [14]:
```
train['text']=train['text'].apply(transform_text)
train.sample(5)
```

Out[14]:

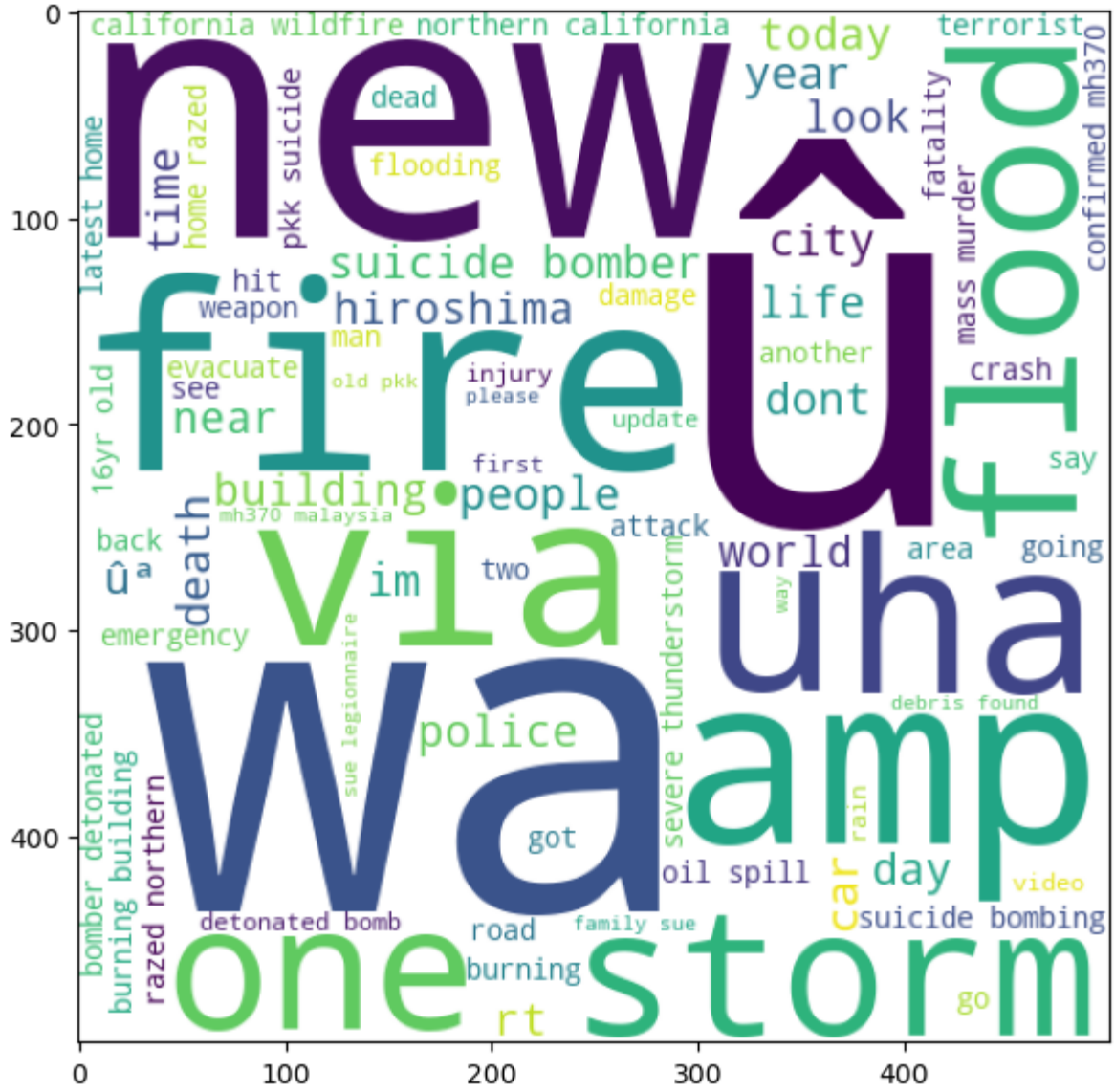
	id	text	target
4789	6813	tianta breaking news unconfirmed heard loud ba...	1
5729	8176	news many death shipwreck rescuer trying save ...	1
5002	7135	people died human experiment unit 731 japanese...	1
6216	8869	im tryna smoke mf	0
6493	9283	like never left sunk background	0

In [15]:
```
from wordcloud import WordCloud
wc = WordCloud(width=500,height=500,min_font_size=10,background_color='white')
```

In [16]:
```
disaster_msg= wc.generate(train[train['target']==1]['text'].str.cat(sep=' '))
```

In [17]:
```
plt.figure(figsize=(7,7))
plt.imshow(disaster_msg)
```

Out[17]:
```
<matplotlib.image.AxesImage at 0x2535bb6c090>
```



Model Training

Text vectorization

In [18]:
```
#using BogOfWords
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
cv = CountVectorizer()
Tfidf=TfidfVectorizer(max_features=5500)
```

In [19]:
```
x=cv.fit_transform(train['text']).toarray()
x.shape
```

Out[19]:
```
(7613, 14142)
```

In [20]:
```
xt=Tfidf.fit_transform(train['text']).toarray()
xt.shape
```

Out[20]: (7613, 5500)

In [21]:
```
y=train['target'].values
y.shape
```

Out[21]: (7613,)

In [22]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.2,random_state=2)
xt_train,xt_test,y_train,y_test= train_test_split(xt,y,test_size=0.2,random_state=2
```

In [23]:
```
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
from sklearn.metrics import accuracy_score, confusion_matrix,precision_score
```

In [24]:
```
gnb=GaussianNB()
mnb=MultinomialNB()
bnb=BernoulliNB()
```

1 Using Gaussian Naive Bayes classifier

2 Using multinomialNB

3 Using BernoulliNB

In [25]:
```
#1 using gnb
gnb.fit(x_train,y_train)
ypred1=gnb.predict(x_test)
print(accuracy_score(y_test,ypred1))
print(confusion_matrix(y_test,ypred1))
print(precision_score(y_test,ypred1))
```

```
0.5883125410374261
[[389 481]
 [146 507]]
0.5131578947368421
```

In [26]:
```
#2 using mnb
mnb.fit(x_train,y_train)
ypred2=mnb.predict(x_test)
print(accuracy_score(y_test,ypred2))
print(confusion_matrix(y_test,ypred2))
print(precision_score(y_test,ypred2))
```

```
0.7912015758371634
[[732 138]
 [180 473]]
0.7741407528641571
```

In [27]:
```
#3 using bnb
bnb.fit(x_train,y_train)
ypred3=bnb.predict(x_test)
print(accuracy_score(y_test,ypred3))
print(confusion_matrix(y_test,ypred3))
print(precision_score(y_test,ypred3))
```

```
0.8030203545633617
[[787  83]
 [217 436]]
0.8400770712909441
```

Repeating same procedures with Tfidf

In [28]:
```
#1 using gnb
gnb.fit(xt_train,y_train)
ypred1=gnb.predict(xt_test)
print(accuracy_score(y_test,ypred1))
print(confusion_matrix(y_test,ypred1))
print(precision_score(y_test,ypred1))
```

```
0.603414313854235
[[438 432]
 [172 481]]
0.5268346111719606
```

In [29]:
```
#2 using mnb
mnb.fit(xt_train,y_train)
ypred2=mnb.predict(xt_test)
print(accuracy_score(y_test,ypred2))
print(confusion_matrix(y_test,ypred2))
print(precision_score(y_test,ypred2))
```

```
0.8023637557452397
[[779  91]
 [210 443]]
0.8295880149812734
```

In [30]:
```
#3 using bnb
bnb.fit(xt_train,y_train)
ypred3=bnb.predict(xt_test)
print(accuracy_score(y_test,ypred3))
print(confusion_matrix(y_test,ypred3))
print(precision_score(y_test,ypred3))
```

```
0.81483913328956
[[785  85]
 [197 456]]
0.8428835489833642
```

In [ ]:
```
#choosing bnb with tf-idf because of high precision and accuracy score
```

In [ ]:
```
#Trying other classifiers, randome forests, SVM
```

In [37]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [38]:
```
#using tfidf data
clf = RandomForestClassifier(n_estimators=100, random_state=2)
clf.fit(xt_train, y_train)
ypred4=clf.predict(xt_test)
print(accuracy_score(y_test,ypred4))
print(confusion_matrix(y_test,ypred4))
print(precision_score(y_test,ypred4))
```

```
0.7721602101116218
[[742 128]
 [219 434]]
0.7722419928825622
```

In [ ]:
```
# using countvectorizer
```

In [41]:
```
from sklearn.svm import SVC
svc = SVC(kernel='sigmoid',gamma=1.0)
```

In [42]:
```
#using tfidf data
svc.fit(xt_train, y_train)
ypred5=svc.predict(xt_test)
print(accuracy_score(y_test,ypred5))
print(confusion_matrix(y_test,ypred5))
print(precision_score(y_test,ypred5))
```

```
0.7879185817465528
[[757 113]
 [210 443]]
0.7967625899280576
```

In [43]:
```
#using countvectorizer
svc.fit(x_train, y_train)
ypred6=svc.predict(x_test)
print(accuracy_score(y_test,ypred6))
print(confusion_matrix(y_test,ypred6))
print(precision_score(y_test,ypred6))
```

```
0.6966513460275772
[[638 232]
 [230 423]]
0.6458015267175573
```

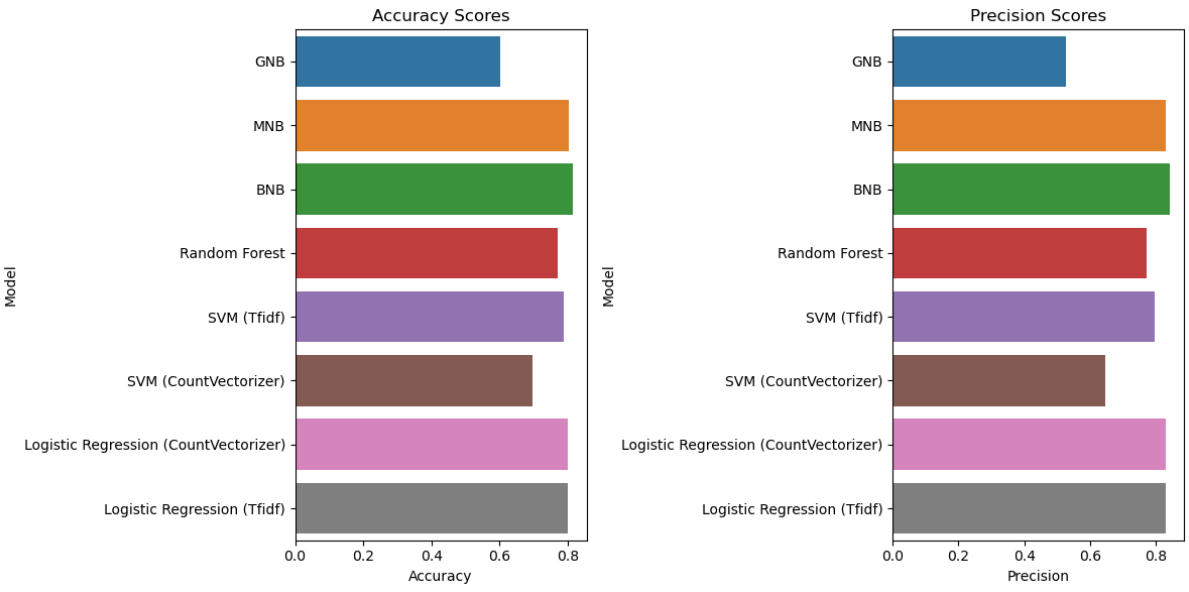
In [33]:
```
#using logistic regression with countvectorizer and tfidf
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(random_state=42)
logreg.fit(x_train, y_train)

# Make predictions on the selected features using countvectorizer
y_pred_logreg = logreg.predict(x_test)

# Evaluate the accuracy of the Logistic Regression model
accuracy_logreg = accuracy_score(y_test, y_pred_logreg)
print("Accuracy with Logistic Regression:", accuracy_logreg)

# Print confusion matrix and precision score
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_logreg))

print("Precision Score:", precision_score(y_test, y_pred_logreg))
```

```
Accuracy with Logistic Regression: 0.7931713722915299
Confusion Matrix:
[[757 113]
 [202 451]]
Precision Score: 0.799645390070922
```

In [45]:
```
#using tfidf
logreg.fit(xt_train, y_train)

# Make predictions on the selected features using tfidf
y_pred_logreg_tfidf = logreg.predict(xt_test)

# Evaluate the accuracy of the Logistic Regression model
accuracy_logreg = accuracy_score(y_test, y_pred_logreg_tfidf)
print("Accuracy with Logistic Regression:", y_pred_logreg_tfidf)

# Print confusion matrix and precision score
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_logreg_tfidf))

print("Precision Score:", precision_score(y_test, y_pred_logreg_tfidf))
```

```
Accuracy with Logistic Regression: [1 0 1 ... 0 1 0]
Confusion Matrix:
[[780  90]
 [215 438]]
Precision Score: 0.8295454545454546
```

In [55]:
```
import seaborn as sns
#Define the models and their corresponding predictions
models = ['GNB', 'MNB', 'BNB', 'Random Forest', 'SVM (Tfidf)', 'SVM (CountVectorize
predictions = [ypred1, ypred2, ypred3, ypred4, ypred5, ypred6, y_pred_logreg, y_pre

# Initialize lists to store accuracy and precision scores
accuracy_scores = []
precision_scores = []

# Calculate accuracy and precision scores for each model
for y_pred in predictions:
    accuracy_scores.append(accuracy_score(y_test, y_pred))
    precision_scores.append(precision_score(y_test, y_pred))

# Create a DataFrame for easy plotting
df = pd.DataFrame({'Model': models, 'Accuracy': accuracy_scores, 'Precision': preci

# Plotting
plt.figure(figsize=(12, 6))

# Plot accuracy scores
plt.subplot(1, 2, 1)
sns.barplot(x='Accuracy', y='Model', data=df)
plt.title('Accuracy Scores')

# Plot precision scores
plt.subplot(1, 2, 2)
sns.barplot(x='Precision', y='Model', data=df)
plt.title('Precision Scores')

plt.tight_layout()
plt.show()
```



Conclusion

The best model we can see is bnb with tf-idf because of high precision and accuracy score so choosing BernoulliNB model

In [57]:
```
import pickle
pickle.dump(Tfidf,open('vectorizer.pkl','wb'))
pickle.dump(bnb,open('model.pkl','wb'))
```

In [ ]: