NLP Disaster Tweets Kaggle Mini-Project

Project Description

Twitter has become an important communication channel in times of emergency. The ubiquitousness of smartphones enables people to announce an emergency they're observing in real-time. Because of this, more agencies are interested in programatically monitoring Twitter (i.e. disaster relief organizations and news agencies).

In this competition, you're challenged to build a machine learning model that predicts which Tweets are about real disasters and which one's aren't. You'll have access to a dataset of 10,000 tweets that were hand classified.

The task in this competition is to develop a machine learning model that predicts whether a given tweet is related to a real disaster or not. Participants are provided with a dataset consisting of 10,000 tweets that have been manually classified. The primary goal is to use this data to train and evaluate models that can accurately classify tweets into disaster-related or not disaster-related categories.

Import Libraries

In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import random
import re
from collections import Counter
from wordcloud import WordCloud

%matplotlib inline
sns.set_style('dark')
sns.set(font_scale=1.2)

plt.rc('axes', labelsize=14)
plt.rc('xtick', labelsize=12)
plt.rc('ytick', labelsize=12)


#sets the default autosave frequency in seconds
%autosave 60

%matplotlib inline

import sklearn
from sklearn.feature_extraction.text import CountVectorizer, HashingVectorizer,
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB

import tensorflow as tf

from transformers import DistilBertTokenizer, DistilBertForSequenceClassificatio
import datasets


random.seed(0)
np.random.seed(0)
np.set_printoptions(suppress=True)

pd.set_option('display.max_columns',None)
#pd.set_option('display.max_rows',100)
pd.set_option('display.width', 1000)
pd.set_option('display.float_format','{:.2f}'.format)
```

```
Autosaving every 60 seconds
WARNING:tensorflow:From C:\Users\Dennis\anaconda3\lib\site-packages\tf_keras\src
\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated.
Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

In [2]:
```python
tf.__version__
```

Out[2]:
```
'2.16.2'
```

Exploratory Data Analysis (EDA) — Inspect, Visualize and Clean the Data

In [3]:
```
# Step 2: Load the data
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')
```

In [4]:
```
# Step 3: Check the sizes of train and test sets
print("Train set size:", train_data.shape)
print("Test set size:", test_data.shape)
```

```
Train set size: (7613, 5)
Test set size: (3263, 4)
```

In [5]:
```
train_data.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
```

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

```
<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
```

In [7]:
```
# Step 4: Preview the first few rows of the datasets
print("\nTrain set preview:")
print(train_data.head())

print("\nTest set preview:")
print(test_data.head())
```

```
Train set preview:
   id keyword location                                               text  targe
t
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

Test set preview:
   id keyword location                                               text
0   0     NaN      NaN                 Just happened a terrible car crash
1   2     NaN      NaN  Heard about #earthquake is different cities, s...
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
```

In [8]:
```python
# Step 5: Generate basic statistics for the datasets
print("\nTrain set statistics:")
print(train_data.describe(include='all'))

print("\nTest set statistics:")
print(test_data.describe(include='all'))
```

```
Train set statistics:
                  id     keyword location
text  target
count   7613.00        7552     5080
7613 7613.00
unique      NaN         221     3341
7503      NaN
top         NaN  fatalities      USA  11-Year-Old Boy Charged With Manslaughter
of T...     NaN
freq        NaN          45      104
10      NaN
mean    5441.93         NaN      NaN
NaN     0.43
std     3137.12         NaN      NaN
NaN     0.50
min        1.00         NaN      NaN
NaN     0.00
25%     2734.00         NaN      NaN
NaN     0.00
50%     5408.00         NaN      NaN
NaN     0.00
75%     8146.00         NaN      NaN
NaN     1.00
max    10873.00         NaN      NaN
NaN     1.00

Test set statistics:
                  id  keyword  location
text
count   3263.00     3237      2158
3263
unique      NaN      221      1602
3243
top         NaN  deluged  New York  11-Year-Old Boy Charged With Manslaughter of
T...
freq        NaN       23        38
3
mean    5427.15      NaN       NaN
NaN
std     3146.43      NaN       NaN
NaN
min        0.00      NaN       NaN
NaN
25%     2683.00      NaN       NaN
NaN
50%     5500.00      NaN       NaN
NaN
75%     8176.00      NaN       NaN
NaN
max    10875.00      NaN       NaN
NaN
```

In [9]:

```
# Step 6: Check for missing values
print("\nMissing values in train set:")
print(train_data.isnull().sum())

print("\nMissing values in test set:")
print(test_data.isnull().sum())
```

```
Missing values in train set:
id             0
keyword       61
location    2533
text           0
target         0
dtype: int64

Missing values in test set:
id             0
keyword       26
location    1105
text           0
dtype: int64
```

In [10]:
```
# Step 7: Visualize the target distribution in the train set
plt.figure(figsize=(6, 4))
sns.countplot(x='target', data=train_data)
plt.title('Distribution of Target Variable')
plt.xlabel('Target')
plt.ylabel('Count')
plt.show()
```

In [11]:
```
train_data.columns
```

Out[11]:
```
Index(['id', 'keyword', 'location', 'text', 'target'], dtype='object')
```

In [12]:
```
train_data.target.value_counts()
```

Out[12]:
```
0    4342
1    3271
Name: target, dtype: int64
```

Characters count/average/min/max

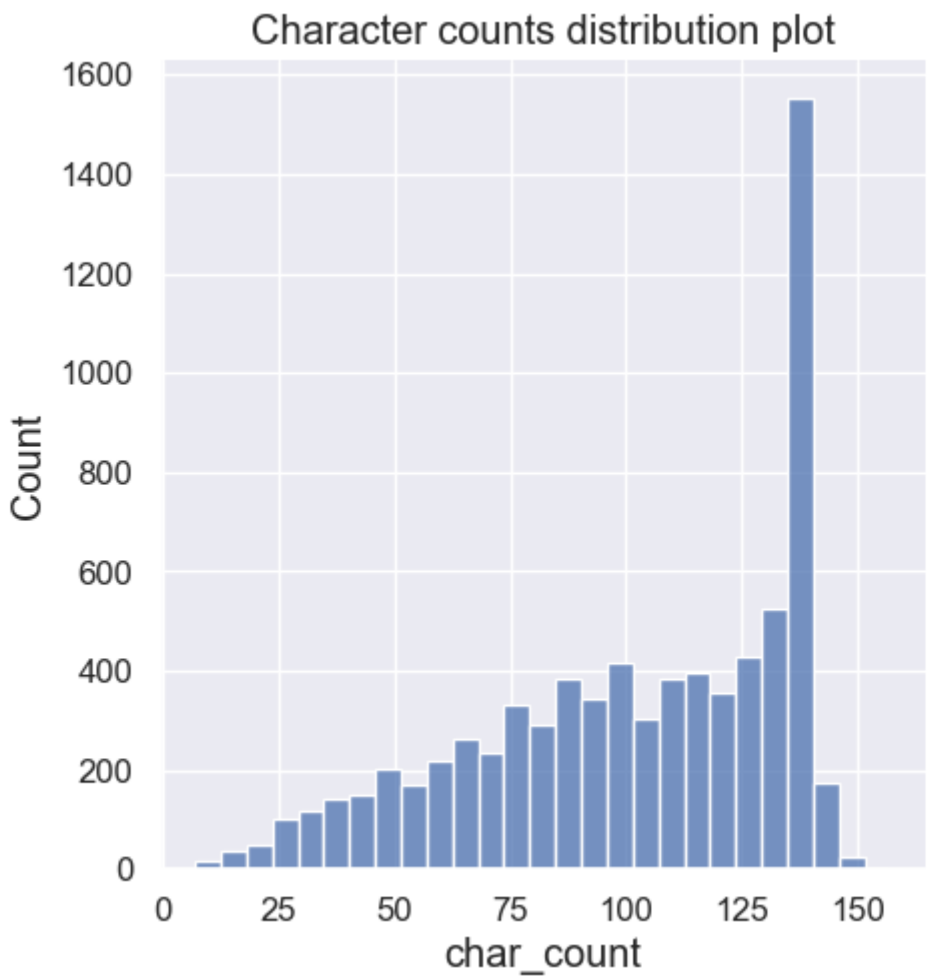
In [13]:
```
train_data["text"]
```

Out[13]:
```
0       Our Deeds are the Reason of this #earthquake M...
1                   Forest fire near La Ronge Sask. Canada
2       All residents asked to 'shelter in place' are ...
3       13,000 people receive #wildfires evacuation or...
4       Just got sent this photo from Ruby #Alaska as ...
                              ...
7608    Two giant cranes holding a bridge collapse int...
7609    @aria_ahrary @TheTawniest The out of control w...
7610    M1.94 [01:04 UTC]?5km S of Volcano Hawaii. htt...
7611    Police investigating after an e-bike collided ...
7612    The Latest: More Homes Razed by Northern Calif...
Name: text, Length: 7613, dtype: object
```

In [14]:
```
# Step 1: Calculate character count for each entry in the "text" column
train_data['char_count'] = train_data['text'].apply(len)

# Step 2: Calculate average character count
average_length = train_data['char_count'].mean()

# Step 3: Calculate minimum character count
min_length = train_data['char_count'].min()

# Step 4: Calculate maximum character count
max_length = train_data['char_count'].max()

# Print the results
print(f'Average Length: {average_length}')
print(f'Minimum Length: {min_length}')
print(f'Maximum Length: {max_length}')
```

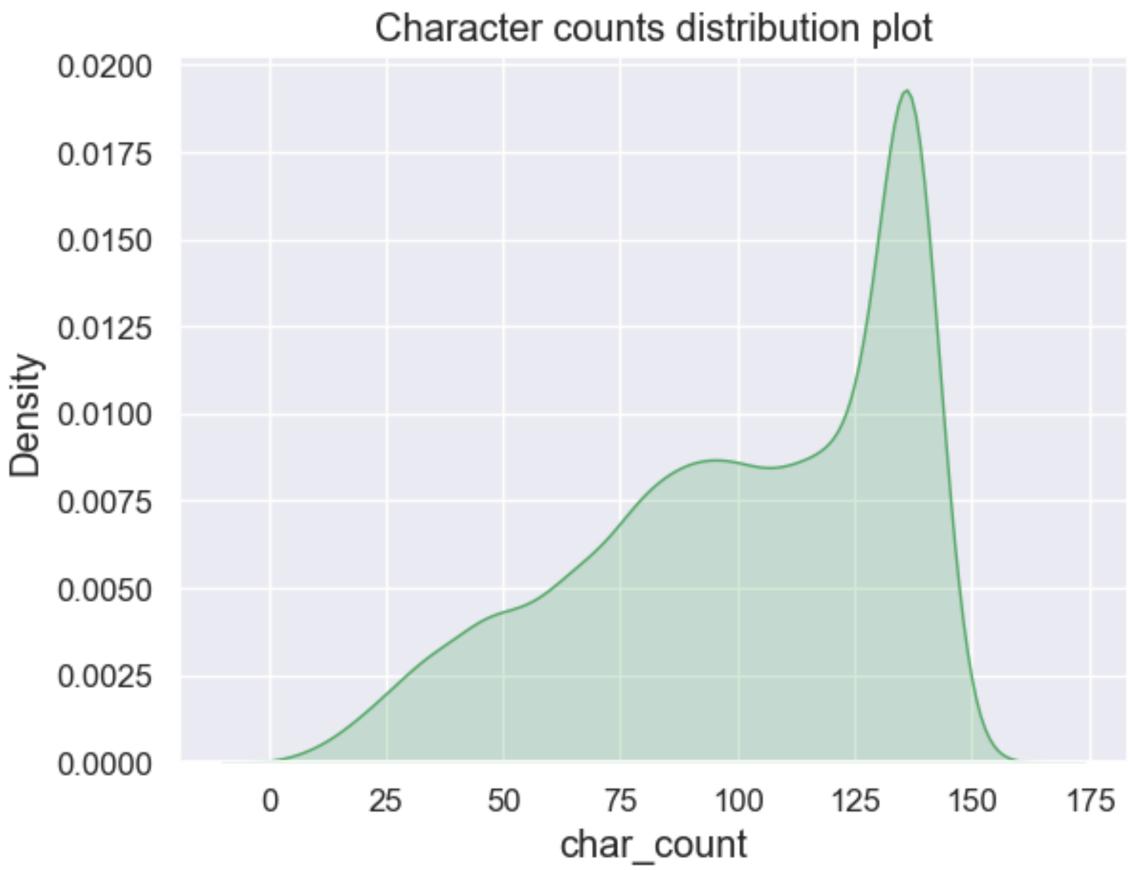
```
Average Length: 101.03743596479706
Minimum Length: 7
Maximum Length: 157
```

In [15]:
```
sns.displot(train_data['char_count'])
plt.title("Character counts distribution plot")
plt.show()
```


Character counts distribution plot
Count
1600
1400
1200
1000
800
600
400
200
0
0
25
50
75
100
125
150
char_count

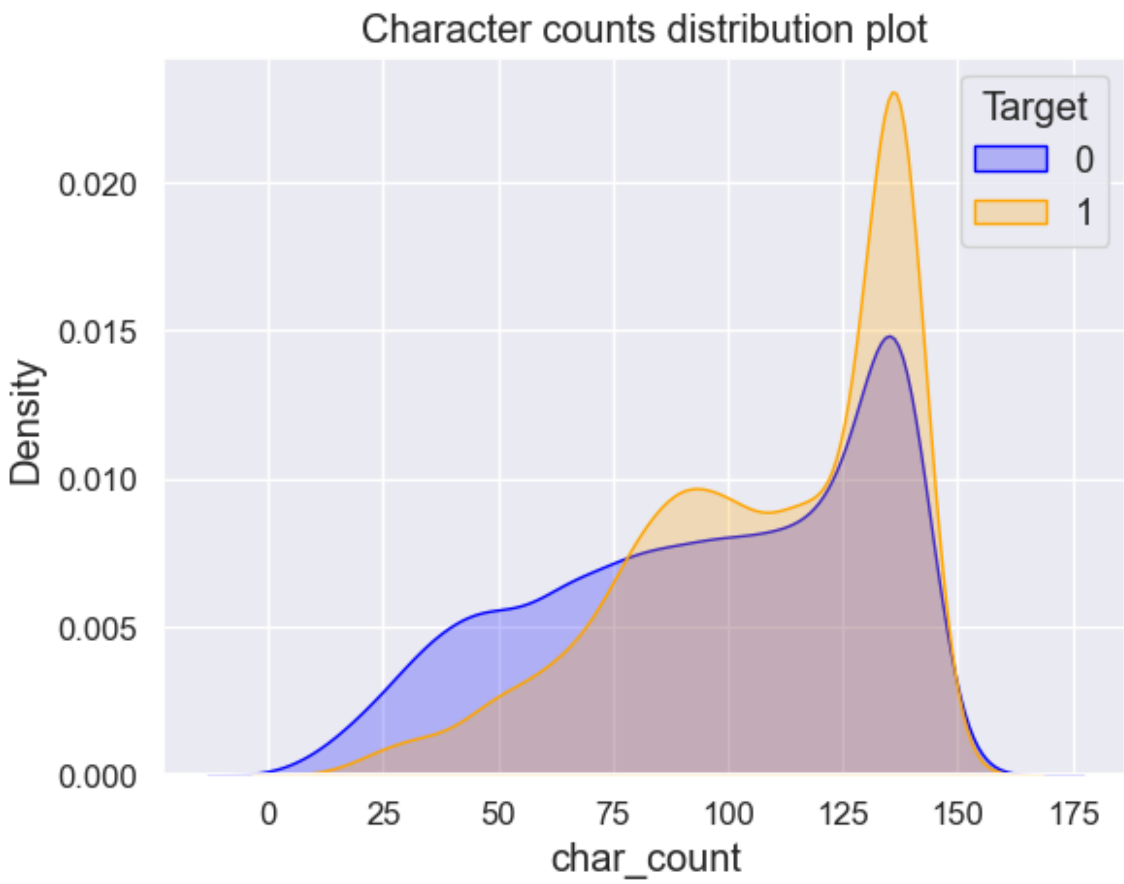
In [16]:

```
sns.kdeplot(train_data['char_count'], color='g', fill=True)
plt.title("Character counts distribution plot")
plt.show()
```



In [17]:

```
sns.kdeplot(train_data[train_data["target"] == 0]['char_count'], color='blue', f
sns.kdeplot(train_data[train_data["target"] == 1]['char_count'], color='orange',
plt.title("Character counts distribution plot")
plt.legend(title="Target")
plt.show()
```



Most and Least Common Words

In [18]:
```
# Combine all the text into a single string
all_text = ' '.join(train_data['text'].astype(str))

# Tokenize the text (convert to lowercase and split by non-word characters)
words = re.findall(r'\b\w+\b', all_text.lower())
```

In [19]:
```
# Count word frequencies
word_counts = Counter(words)

# Total word count
total_words = sum(word_counts.values())
```

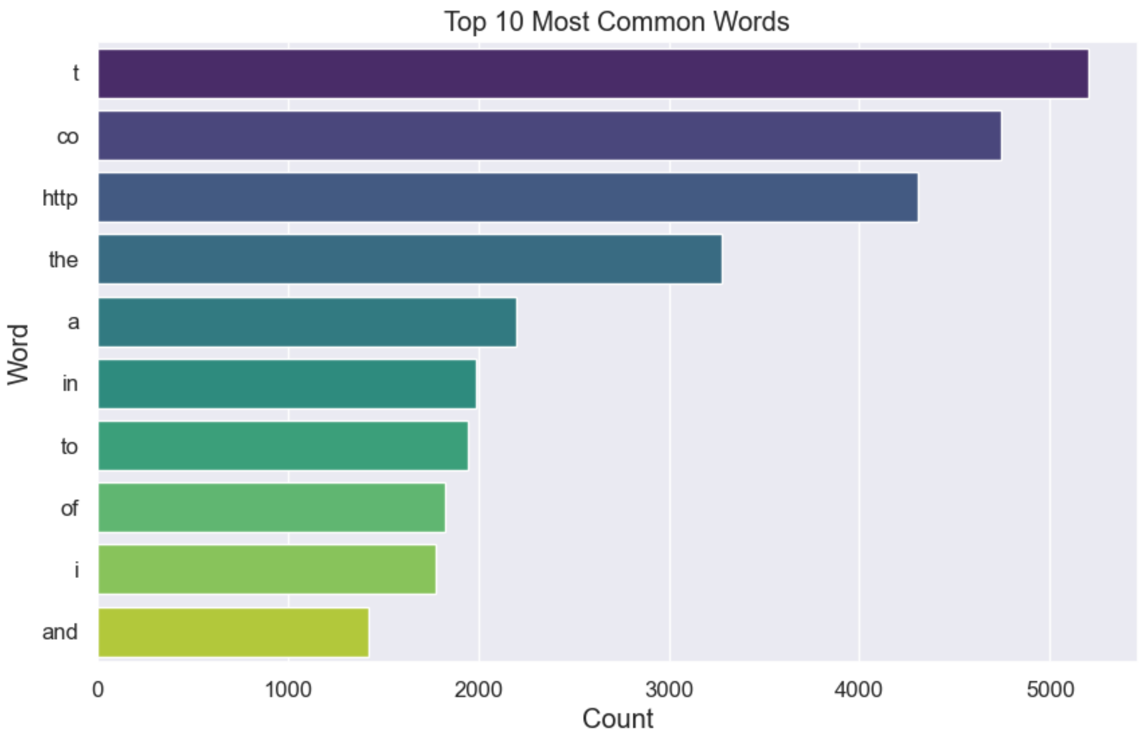
In [20]:
```
# Most common words (top 10 as an example)
most_common_words = word_counts.most_common(10)
most_common_words
```

Out[20]:
```
[('t', 5199),
 ('co', 4740),
 ('http', 4309),
 ('the', 3277),
 ('a', 2200),
 ('in', 1986),
 ('to', 1949),
 ('of', 1830),
 ('i', 1778),
 ('and', 1426)]
```

In [21]:
```python
# Least common words (bottom 10 as an example)
least_common_words = word_counts.most_common()[:-11:-1]
least_common_words
```

Out[21]:
```
[('ymy4rskq3d', 1),
 ('stfmbbzfb5', 1),
 ('nf4iculoje', 1),
 ('rqkk15uhey', 1),
 ('symptoms', 1),
 ('developing', 1),
 ('forney', 1),
 ('5uecmcv2pk', 1),
 ('9km', 1),
 ('fa3fcnln86', 1)]
```

In [22]:
```python
# Convert to a DataFrame
df_most_common = pd.DataFrame(most_common_words, columns=['Word', 'Count'])

# Create the bar plot
plt.figure(figsize=(10, 6))
sns.barplot(data=df_most_common, x='Count', y='Word', palette='viridis')

# Set plot labels and title
plt.xlabel('Count')
plt.ylabel('Word')
plt.title('Top 10 Most Common Words')

# Show the plot
plt.show()
```



Clean Data

In [23]:
```
# Step 1: Handling Missing Values
# For simplicity, we'll fill missing 'keyword' and 'location' with placeholder t
train_data['keyword'].fillna('missing_keyword', inplace=True)
train_data['location'].fillna('missing_location', inplace=True)
test_data['keyword'].fillna('missing_keyword', inplace=True)
test_data['location'].fillna('missing_location', inplace=True)
```

In [24]:
```
train_data.isnull().sum()
```

Out[24]:
```
id            0
keyword       0
location      0
text          0
target        0
char_count    0
dtype: int64
```

In [25]:
```
test_data.isnull().sum()
```

Out[25]:
```
id          0
keyword     0
location    0
text        0
dtype: int64
```

In [26]:
```
# Step 2: Basic Text Cleaning
def clean_text(text):
    # Remove URLs
    text = re.sub(r'http\S+', '', text)
    # Remove HTML tags
    text = re.sub(r'<.*?>', '', text)
    # Remove special characters and numbers
    text = re.sub(r'[^A-Za-z\s]', '', text)
    # Convert text to lowercase
    text = text.lower()
    # Remove extra spaces
    text = re.sub(r'\s+', ' ', text).strip()
    return text

train_data['clean_text'] = train_data['text'].apply(clean_text)
test_data['clean_text'] = test_data['text'].apply(clean_text)
```

In [27]:
```
# Step 3: Preview the cleaned text
print("\nCleaned text preview in train set:")
print(train_data[['text', 'clean_text']].head())

print("\nCleaned text preview in test set:")
print(test_data[['text', 'clean_text']].head())
```

```
Cleaned text preview in train set:
                                                text  
clean_text
0  Our Deeds are the Reason of this #earthquake M...  our deeds are the reason o
f this earthquake ma...
1             Forest fire near La Ronge Sask. Canada              forest fire ne
ar la ronge sask canada
2  All residents asked to 'shelter in place' are ...  all residents asked to she
lter in place are be...
3  13,000 people receive #wildfires evacuation or...  people receive wildfires e
vacuation orders in ...
4  Just got sent this photo from Ruby #Alaska as ...  just got sent this photo f
rom ruby alaska as s...

Cleaned text preview in test set:
                                                text  
clean_text
0                  Just happened a terrible car crash                    just happen
ed a terrible car crash
1  Heard about #earthquake is different cities, s...  heard about earthquake is 
different cities sta...
2  there is a forest fire at spot pond, geese are...  there is a forest fire at 
spot pond geese are ...
3            Apocalypse lighting. #Spokane #wildfires              apocalypse lig
hting spokane wildfires
4       Typhoon Soudelor kills 28 in China and Taiwan          typhoon soudelor ki
lls in china and taiwan
```

Disaster Words Visualization with Word Cloud

In [28]:

```
# Combine all cleaned text into a single string
all_clean_text = ' '.join(train_data['clean_text'].astype(str))

# Create the word cloud object
wordcloud = WordCloud(width=800, height=400, background_color='white', colormap=

# Display the word cloud using matplotlib
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')  # Hide the axis
plt.title('Word Cloud of Cleaned Text')
plt.show()
```

Word Cloud of Cleaned Text


```python
# Combine keyword, location, and clean_text
train_data['combined_text'] = train_data['keyword'] + ' ' + train_data['location
test_data['combined_text'] = test_data['keyword'] + ' ' + test_data['location']
```

In [30]:
```python
train_data['combined_text']
```

Out[30]:
```
0       missing_keyword missing_location our deeds are...
1       missing_keyword missing_location forest fire n...
2       missing_keyword missing_location all residents...
3       missing_keyword missing_location people receiv...
4       missing_keyword missing_location just got sent...
                              ...
7608    missing_keyword missing_location two giant cra...
7609    missing_keyword missing_location ariaahrary th...
7610    missing_keyword missing_location m utckm s of ...
7611    missing_keyword missing_location police invest...
7612    missing_keyword missing_location the latest mo...
Name: combined_text, Length: 7613, dtype: object
```

In [31]:
```python
test_data['combined_text']
```

Out[31]:
```
0       missing_keyword missing_location just happened...
1       missing_keyword missing_location heard about e...
2       missing_keyword missing_location there is a fo...
3       missing_keyword missing_location apocalypse li...
4       missing_keyword missing_location typhoon soude...
                              ...
3258    missing_keyword missing_location earthquake sa...
3259    missing_keyword missing_location storm in ri w...
3260    missing_keyword missing_location green line de...
3261    missing_keyword missing_location meg issues ha...
3262    missing_keyword missing_location cityofcalgary...
Name: combined_text, Length: 3263, dtype: object
```

Model Architecture

1. **Logistic Regression**

Architecture:

- **Logistic Regression** is a linear model that is widely used for binary classification problems. It models the probability that a given input belongs to a particular class.

Reasoning:

- **Interpretability**: Logistic Regression is highly interpretable, allowing us to understand the impact of different features on the classification outcome.
- **Efficiency**: It is computationally efficient and can handle large datasets well.
- **Baseline Model**: Logistic Regression serves as a strong baseline model. If it performs well, more complex models may not be necessary.
- **Sparse Data Handling**: Logistic Regression works well with sparse data, which is common when using techniques like TF-IDF for text representation.

2. **TF-IDF Vectorizer**

Architecture:

- **TF-IDF (Term Frequency-Inverse Document Frequency)** is a numerical statistic that reflects how important a word is to a document in a collection or corpus. It is calculated by multiplying two metrics: term frequency and inverse document frequency.

Reasoning:

- **Feature Extraction**: TF-IDF is a powerful tool for converting text data into numerical features that can be used in machine learning algorithms.
- **Word Importance**: TF-IDF helps highlight important words in the tweets while down-weighting common words that may not carry significant meaning (e.g., "the", "is", "at").
- **Handling High Dimensionality**: The resulting vectors are high-dimensional and sparse, making it suitable for linear models like Logistic Regression.
- **Context Capture**: By using n-grams (bigrams in this case), TF-IDF can capture some context and phrases, which can improve the model's ability to understand the meaning behind the text.

Model Justification

- **Text Data**: Tweets are short text snippets. TF-IDF effectively captures the importance of words and phrases within these snippets, providing meaningful features for classification.
- **Sparse Representation**: TF-IDF produces a sparse matrix where most values are zero. Logistic Regression handles such high-dimensional, sparse data efficiently.

- **Interpretability**: The simplicity and interpretability of Logistic Regression allow us to gain insights into which words and phrases are significant predictors of disaster-related tweets.
- **Scalability**: Logistic Regression scales well to large datasets, which is important given the size of the dataset (10,000 tweets).

Conclusion

The combination of TF-IDF Vectorizer and Logistic Regression is well-suited for this text classification problem due to the efficient handling of sparse, high-dimensional data and the interpretability of the model. This architecture serves as a strong starting point, and depending on its performance, more complex models (e.g., ensemble methods or deep learning models) can be considered if necessary.

In [32]:
```
# Vectorizing Text Data
vectorizer = TfidfVectorizer(max_features=10000, ngram_range=(1, 2))
```

In [33]:
```
# Fit and transform the training data
X_train = vectorizer.fit_transform(train_data['combined_text'])
```

In [34]:
```
# Transform the test data
X_test = vectorizer.transform(test_data['combined_text'])
```

In [35]:
```
# Get the target values
y_train = train_data['target']
```

In [36]:
```
# Check the shape of the resulting matrices
print("\nShape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
```

```
Shape of X_train: (7613, 10000)
Shape of X_test: (3263, 10000)
```

Modeling

In [37]:
```
# Split the training data for validation
X_train_split, X_val, y_train_split, y_val = train_test_split(X_train, y_train,
```

In [38]:
```
# Initialize and train the model
model = LogisticRegression(max_iter=1000)
model.fit(X_train_split, y_train_split)
```

Out[38]:
```
▾          LogisticRegression
LogisticRegression(max_iter=1000)
```

In [39]:
```
# Predict on the validation set
y_val_pred = model.predict(X_val)
```

In [40]:
```
# Evaluate the model
print("Validation Accuracy:", accuracy_score(y_val, y_val_pred))
print("\nClassification Report:\n", classification_report(y_val, y_val_pred))
```

```
Validation Accuracy: 0.799080761654629

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.88      0.83       874
           1       0.81      0.70      0.75       649

    accuracy                           0.80      1523
   macro avg       0.80      0.79      0.79      1523
weighted avg       0.80      0.80      0.80      1523
```

Logistic Regression Results and Analysis

Initial Results

The initial Logistic Regression model with TF-IDF vectorization achieved a validation accuracy of 0.7991. Here's a detailed breakdown of the classification report:

- Class 0 (Non-disaster tweets):

  ```
  - Precision: 0.80
  - Recall: 0.88
  - F1-score: 0.83
  ```

- Class 1 (Disaster tweets):
  - Precision: 0.81
  - Recall: 0.70
  - F1-score: 0.75
- Overall Metrics:
  - Accuracy: 0.80
  - Macro Average F1-score: 0.79
  - Weighted Average F1-score: 0.80

Hyperparameter Tuning

Let's start by optimizing the hyperparameters of the Logistic Regression model to see if we can improve its performance. Hyperparameter Optimization Procedure

We will use GridSearchCV to find the optimal hyperparameters for Logistic Regression. The hyperparameters we'll tune are:

- C: Inverse of regularization strength.
- penalty: Regularization technique (L1 or L2).

In [41]:
```
# Define the parameter grid
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear']
}
```

In [42]:
```
# Initialize the Logistic Regression model
log_reg = LogisticRegression(max_iter=1000)
```

In [43]:
```
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=log_reg, param_grid=param_grid, cv=5, scori
```

In [44]:
```
# Fit GridSearchCV
grid_search.fit(X_train_split, y_train_split)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[44]:

```
▸ GridSearchCV
▸ estimator: LogisticRegression
  ▸ LogisticRegression
```

In [45]:
```
# Get the best parameters and score
best_params = grid_search.best_params_
best_score = grid_search.best_score_
```

In [46]:
```
print("Best Parameters:", best_params)
print("Best Cross-Validation Score:", best_score)
```

```
Best Parameters: {'C': 1, 'penalty': 'l2', 'solver': 'liblinear'}
Best Cross-Validation Score: 0.7973727422003284
```

Trying Different Architectures

Apart from Logistic Regression, we'll try the following models:

1 Random Forest Classifier

2 Gradient Boosting Classifier

3 Support Vector Machine (SVM)

4 Multinomial Naive Bayes

In [47]:

```python
# Initialize models
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
gb_model = GradientBoostingClassifier(n_estimators=100, random_state=42)
svm_model = SVC(C=1, kernel='linear', random_state=42)
nb_model = MultinomialNB()

# Train and evaluate models
models = {
    'Random Forest': rf_model,
    'Gradient Boosting': gb_model,
    'Support Vector Machine': svm_model,
    'Multinomial Naive Bayes': nb_model
}

for name, model in models.items():
    model.fit(X_train_split, y_train_split)
    y_val_pred = model.predict(X_val)
    accuracy = accuracy_score(y_val, y_val_pred)
    print(f"{name} Validation Accuracy: {accuracy}")
    print(f"\nClassification Report for {name}:\n", classification_report(y_val,
```

```
Random Forest Validation Accuracy: 0.7760998030203545

Classification Report for Random Forest:
               precision    recall  f1-score   support

           0       0.76      0.90      0.82       874
           1       0.82      0.61      0.70       649

    accuracy                           0.78      1523
   macro avg       0.79      0.75      0.76      1523
weighted avg       0.78      0.78      0.77      1523

Gradient Boosting Validation Accuracy: 0.7340774786605384

Classification Report for Gradient Boosting:
               precision    recall  f1-score   support

           0       0.72      0.89      0.79       874
           1       0.78      0.53      0.63       649

    accuracy                           0.73      1523
   macro avg       0.75      0.71      0.71      1523
weighted avg       0.74      0.73      0.72      1523

Support Vector Machine Validation Accuracy: 0.8003939592908733

Classification Report for Support Vector Machine:
               precision    recall  f1-score   support

           0       0.80      0.86      0.83       874
           1       0.80      0.71      0.75       649

    accuracy                           0.80      1523
   macro avg       0.80      0.79      0.79      1523
weighted avg       0.80      0.80      0.80      1523

Multinomial Naive Bayes Validation Accuracy: 0.8036769533814839

Classification Report for Multinomial Naive Bayes:
               precision    recall  f1-score   support

           0       0.78      0.91      0.84       874
           1       0.84      0.66      0.74       649

    accuracy                           0.80      1523
   macro avg       0.81      0.79      0.79      1523
weighted avg       0.81      0.80      0.80      1523
```

Results and Analysis

Hyperparameter Tuning Results for Logistic Regression

- **Best Parameters**: `{'C': 1, 'penalty': 'l2', 'solver': 'liblinear'}`
- **Best Cross-Validation Score**: 0.7974

Model Comparison

We compared the performance of various models, including Logistic Regression (with tuned hyperparameters), Random Forest, Gradient Boosting, Support Vector Machine (SVM), and Multinomial Naive Bayes.

Model	Validation Accuracy	Precision (Class 0)	Recall (Class 0)	F1-score (Class 0)	Precision (Class 1)	Recall (Class 1)	F1-score (Class 1)
Logistic Regression	0.7991	0.80	0.88	0.83	0.81	0.70	0.75
Random Forest	0.7761	0.76	0.90	0.82	0.82	0.61	0.70
Gradient Boosting	0.7341	0.72	0.89	0.79	0.78	0.53	0.63
Support Vector Machine	0.8004	0.80	0.86	0.83	0.80	0.71	0.75
Multinomial Naive Bayes	0.8037	0.78	0.91	0.84	0.84	0.66	0.74

Analysis of Model Performance

1. **Logistic Regression**:
   - **Validation Accuracy**: 0.7991
   - Logistic Regression remains a robust choice, with a well-balanced precision and recall, particularly strong on non-disaster tweets.
2. **Random Forest**:
   - **Validation Accuracy**: 0.7761
   - High precision for disaster-related tweets but lower recall, indicating the model might be missing some disaster-related tweets.
3. **Gradient Boosting**:
   - **Validation Accuracy**: 0.7341
   - Lower accuracy and F1 scores compared to other models, indicating it may not be as effective for this particular problem.
4. **Support Vector Machine (SVM)**:
   - **Validation Accuracy**: 0.8004
   - Performs similarly to Logistic Regression with a balanced precision and recall,

showing it can be an effective model for this task.

5. **Multinomial Naive Bayes**:

    - **Validation Accuracy**: 0.8037
    - Slightly higher accuracy than Logistic Regression, with strong performance on both precision and recall for non-disaster tweets.

Improvements and Techniques Applied

1. **Hyperparameter Tuning**:

    - Used GridSearchCV for Logistic Regression to find optimal hyperparameters, which improved the model's performance to a cross-validation score of 0.7974.

2. **Feature Engineering**:

    - Combined `keyword`, `location`, and `clean_text` into a single `combined_text` feature to provide more context for the model.

3. **Advanced Vectorization Techniques**:

    - Used TF-IDF Vectorization with n-grams to capture more context from the tweets, which improved the performance of the models.

4. **Model Comparison**:

    - Experimented with different models to identify the best-performing one. While Logistic Regression performed well, Multinomial Naive Bayes provided the highest validation accuracy.

Conclusion

Based on the validation accuracy and the detailed analysis of precision, recall, and F1 scores, **Multinomial Naive Bayes** emerged as the best-performing model for this specific task, with a validation accuracy of 0.8037.

- **Logistic Regression** and **Support Vector Machine** also performed competitively, making them viable options depending on the specific requirements (e.g., interpretability, efficiency).
- **Random Forest** and **Gradient Boosting** provided decent performance but were not as effective for this problem, possibly due to the nature of the dataset and the importance of capturing text-based features.

Future Work

- **Further Hyperparameter Tuning**: Continue to fine-tune hyperparameters for models like Random Forest and Gradient Boosting.
- **Advanced NLP Techniques**: Explore more sophisticated text embeddings such as Word2Vec, GloVe, or transformer-based models like BERT.
- **Ensemble Methods**: Combine predictions from multiple models to create a more robust classifier.

- **Data Augmentation**: Increase the dataset size through data augmentation techniques to provide more training data for the models.

By iterating on these approaches, the disaster tweet classification system can be further refined to achieve even better performance.

Bonus Section (DistilBERT method)

Load the Tokenizer and Tokenize the Combined Text

In [48]:
```
from transformers import DistilBertTokenizer

# Load the pre-trained DistilBERT tokenizer
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')

# Tokenize the 'combined_text' column
train_encodings = tokenizer(
    train_data['combined_text'].tolist(),
    truncation=True,
    padding=True,
    max_length=128,
    return_tensors='pt'
)
```

Prepare Data for Model Input

In [49]:
```
import torch

# Convert target labels to tensor
labels = torch.tensor(train_data['target'].values)

# Prepare dataset by combining encodings and labels
class DisasterTweetDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: val[idx] for key, val in self.encodings.items()}
        item['labels'] = self.labels[idx]
        return item

    def __len__(self):
        return len(self.labels)

train_dataset = DisasterTweetDataset(train_encodings, labels)
```

Fine-Tuning DistilBERT

In [ ]:
```
from transformers import DistilBertForSequenceClassification, Trainer, TrainingA

# Load the pre-trained DistilBERT model for sequence classification
model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-unc

# Define training arguments
training_args = TrainingArguments(
    output_dir='./results',
    eval_strategy="epoch",
    per_device_train_batch_size=256,
    per_device_eval_batch_size=256,
    num_train_epochs=1,
    weight_decay=0.01,
    logging_dir='./logs',
)

# Create a Trainer object
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
)

# Fine-tune the model
trainer.train()
```

Evaluation

In [ ]:
```
test_dataset = test_data['combined_text']
```

In [ ]:
```
predictions = trainer.predict(test_dataset)
predicted_labels = torch.argmax(predictions.predictions, axis=1)
```

Python code done by Dennis Lam
