**Project: DIsaster Tweet Classification**

**Introduction**

Natural disasters can cause significant devastation, making timely and accurate information crucial for effective response and resource allocation. Social media platforms like Twitter have become essential channels for real-time information during such events. However, the sheer volume of data generated on these platforms poses a challenge in identifying relevant disaster-related information.

This project, inspired by the Kaggle competition "Natural Language Processing with Disaster Tweets," aims to utilize machine learning and natural language processing (NLP) techniques to distinguish between tweets related to actual disasters and those that are not. By accurately categorizing tweets, emergency responders and organizations can prioritize resources and make informed decisions during disaster response efforts.

**Objective**

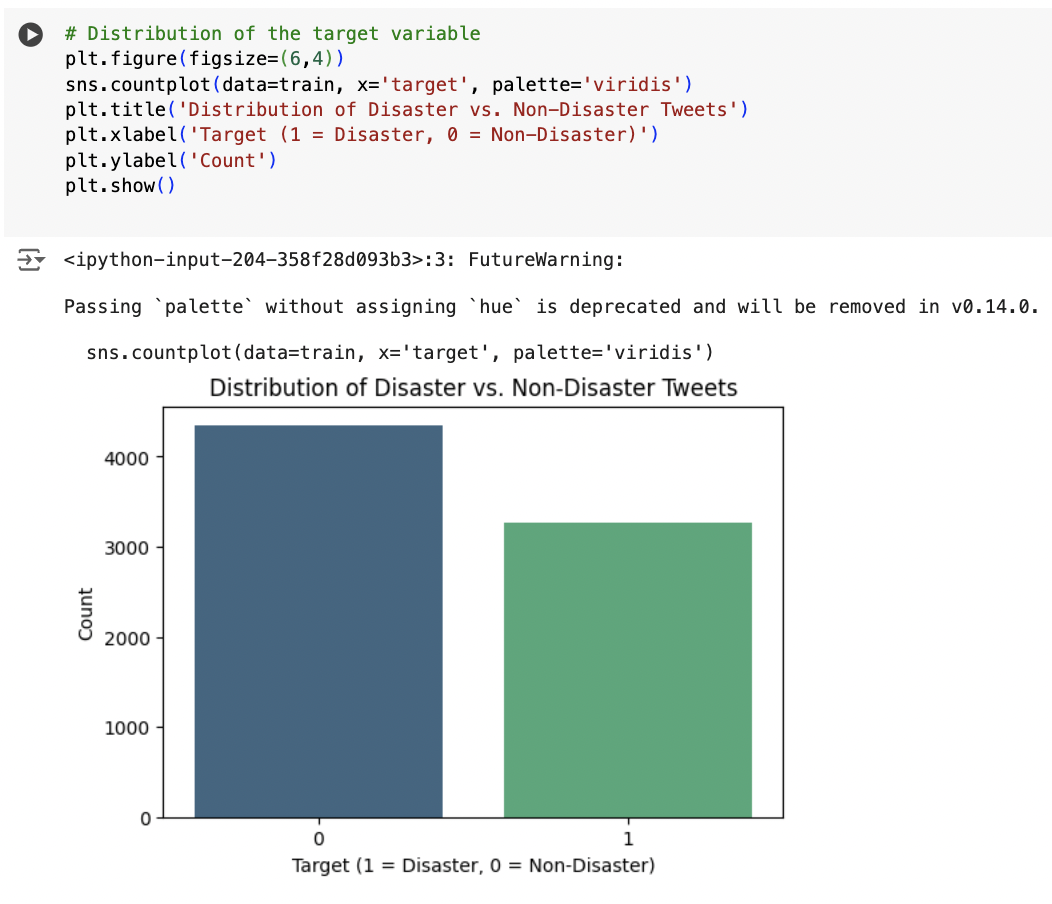
The objective of this project is to develop a robust machine learning model to classify disaster-related tweets. The key goals include:

1. Data Preprocessing: Clean and prepare tweet data through text normalization, tokenization, and handling missing values.
2. Feature Engineering: Implement features such as keyword extraction and text embeddings to improve model accuracy.
3. Model Development: Build and evaluate various models, including CNNs, RNNs, and transformer-based architectures, to identify the most effective approach.
4. Model Evaluation: Use metrics like accuracy, precision, recall, and F1-score to assess model performance.

**Data Analysis and Insight**

Distribution of Disaster vs. Non-Disaster Tweets:

The distribution of the target variable indicates that the dataset contains a slightly higher number of non-disaster tweets compared to disaster-related tweets. This slight imbalance should be considered when developing the model, as it could affect the performance of certain machine learning algorithms, particularly those sensitive to class imbalances.



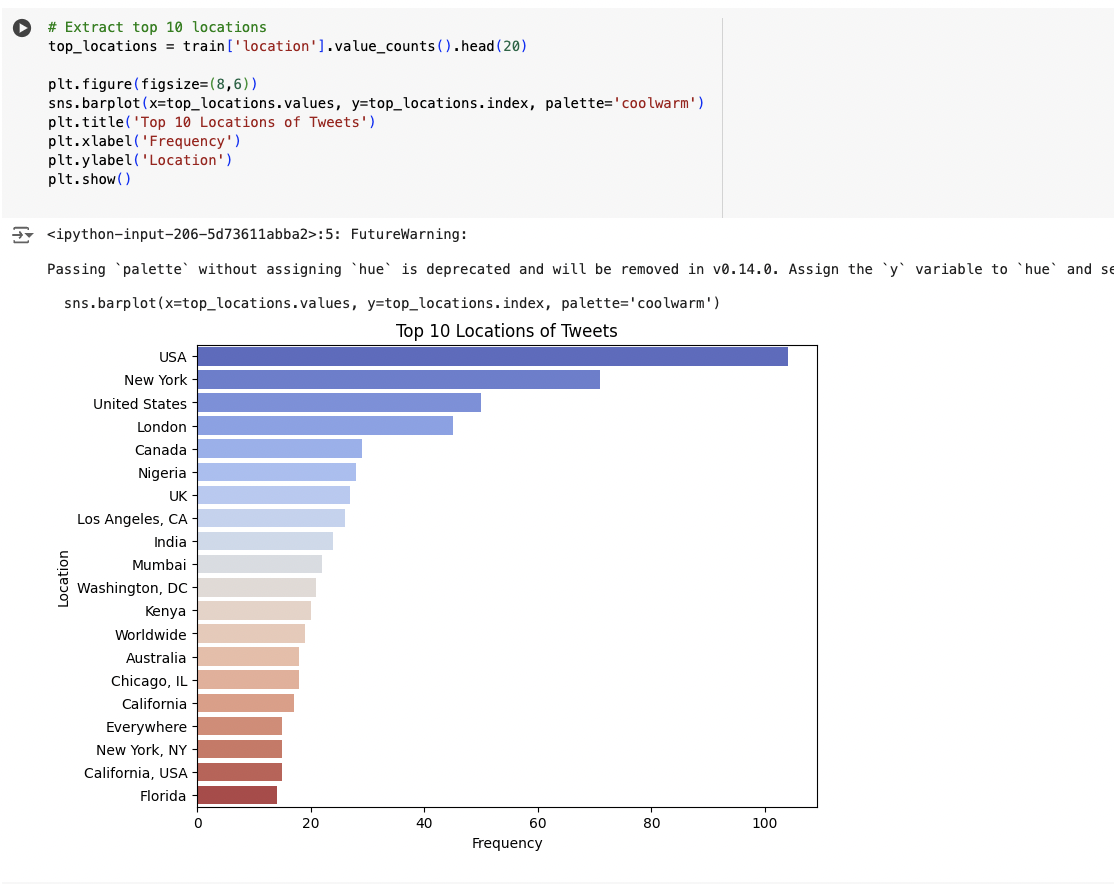
Top 10 Keywords in Disaster-Related Tweets:

The most frequent keywords in disaster-related tweets include terms like "derailment," "wreckage," "outbreak," and "debris," which are directly associated with catastrophic events. These keywords provide valuable context for identifying disaster-related content, highlighting the importance of focusing on specific terms during feature engineering and model training.



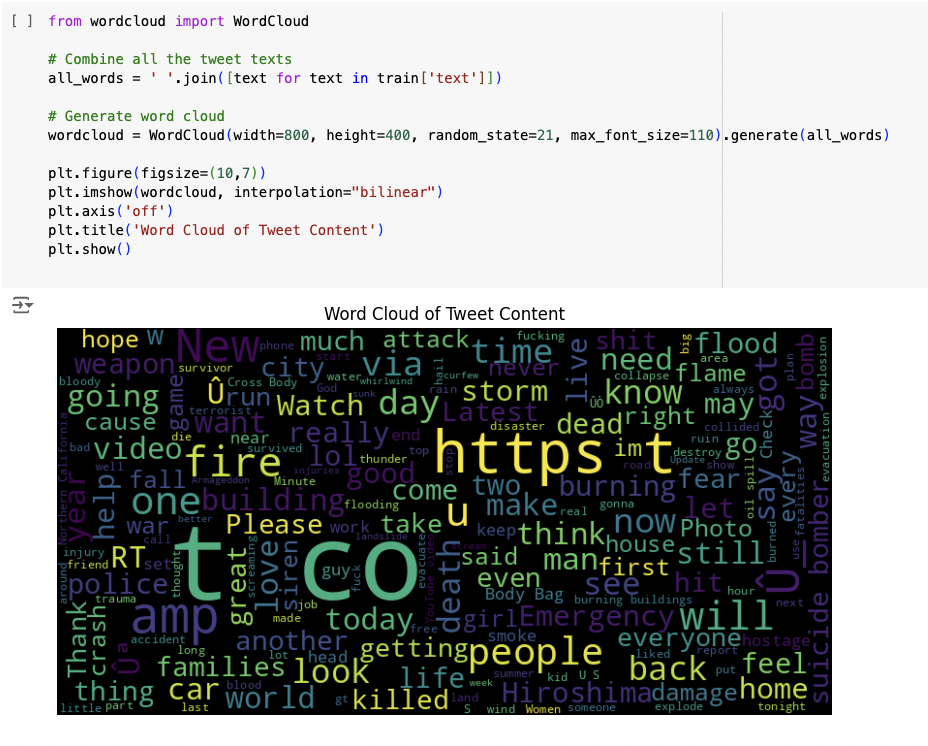
Top 10 Locations of Tweets:

The analysis of tweet locations shows that the majority of tweets originate from the USA, followed by New York, the United States (general), and other global locations like London and Canada. This suggests a broad geographic distribution of tweets, emphasizing the need for models to be location-aware, especially when handling disaster-related content.



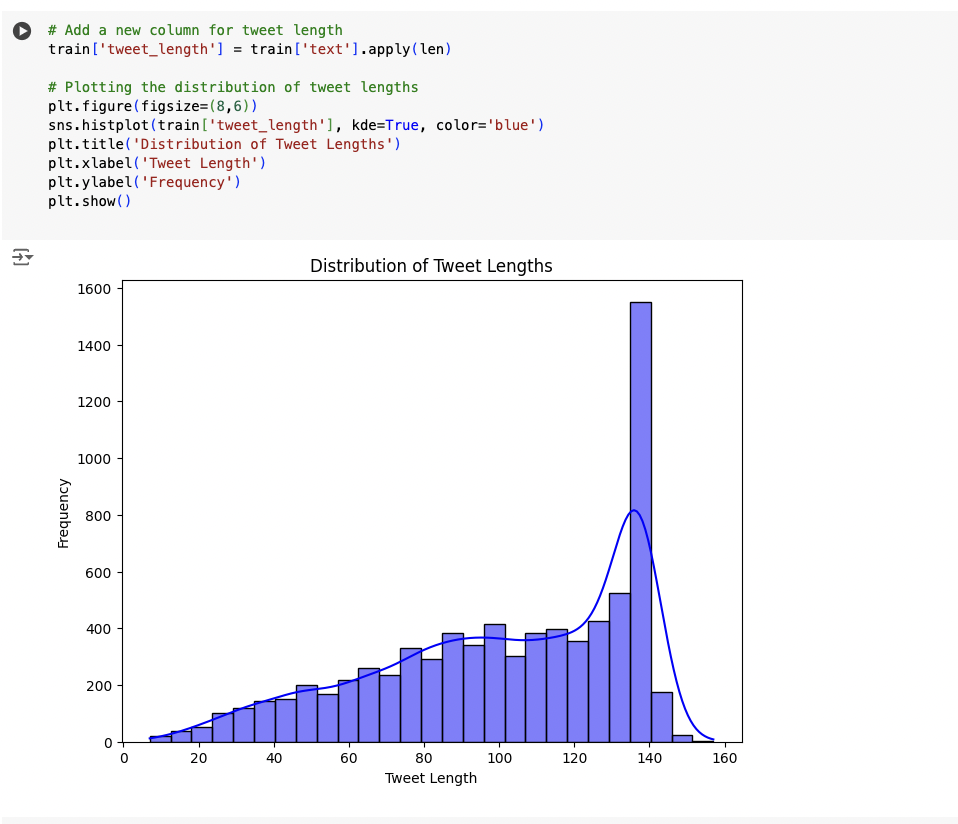
Word Cloud of Tweet Content:

The word cloud generated from the entire corpus of tweets reveals prominent terms such as "fire," "people," "killed," and "disaster." These words underscore the common themes and concerns expressed in the tweets, providing insights into the general sentiment and focus areas within the dataset. The prominence of specific terms could guide the development of more targeted feature extraction techniques.



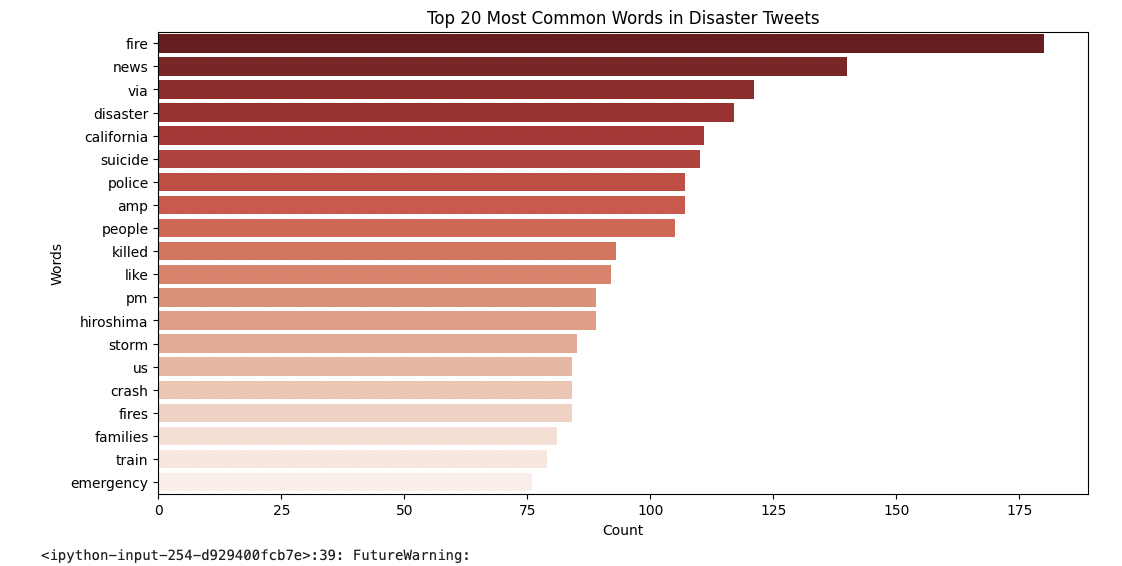
Distribution of Tweet Lengths:

The distribution of tweet lengths shows that most tweets are between 60 and 140 characters long, with a noticeable peak at the maximum length allowed by Twitter. Understanding tweet length distribution is crucial for tokenization and sequence padding during preprocessing, ensuring that models can handle the typical length of tweets effectively.



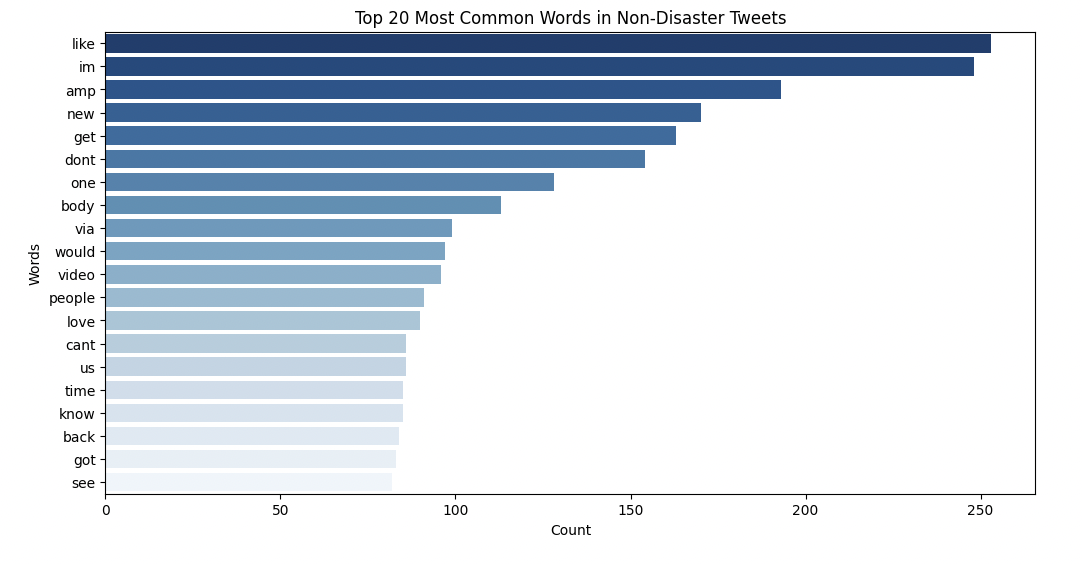
Top 20 Most Common Words in Disaster Tweets:

The most common words in disaster-related tweets include "fire," "news," "via," and "disaster," reflecting the urgency and information-sharing nature of these tweets. These common words provide a strong foundation for feature extraction and may serve as key indicators for classifying tweets as disaster-related.



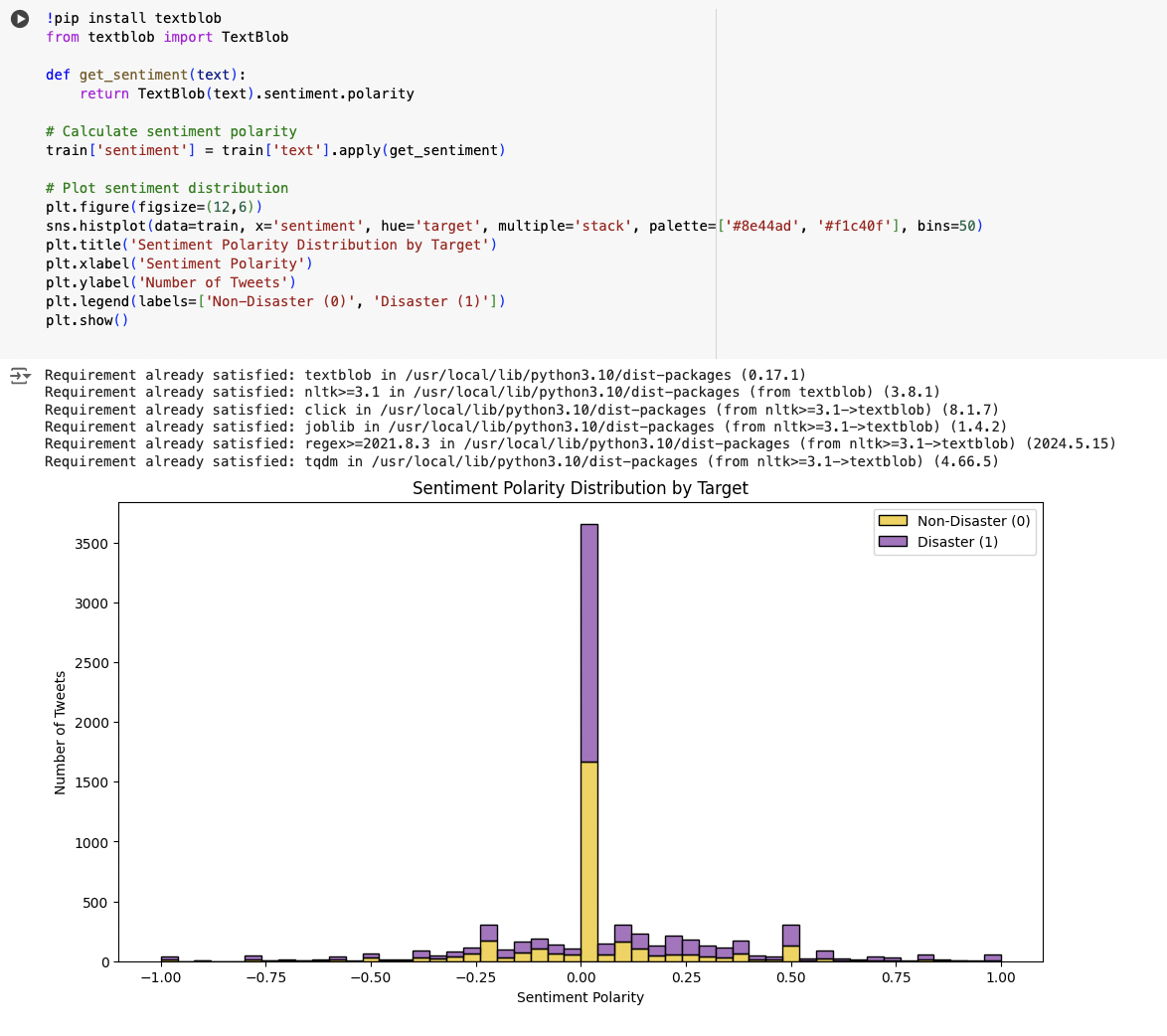
Top 20 Most Common Words in Non-Disaster Tweets:

Non-disaster tweets frequently contain terms like "like," "im," "amp," and "new," which are more generic and less context-specific. This contrast with disaster-related tweets highlights the importance of differentiating between general social media language and disaster-specific vocabulary in model development.



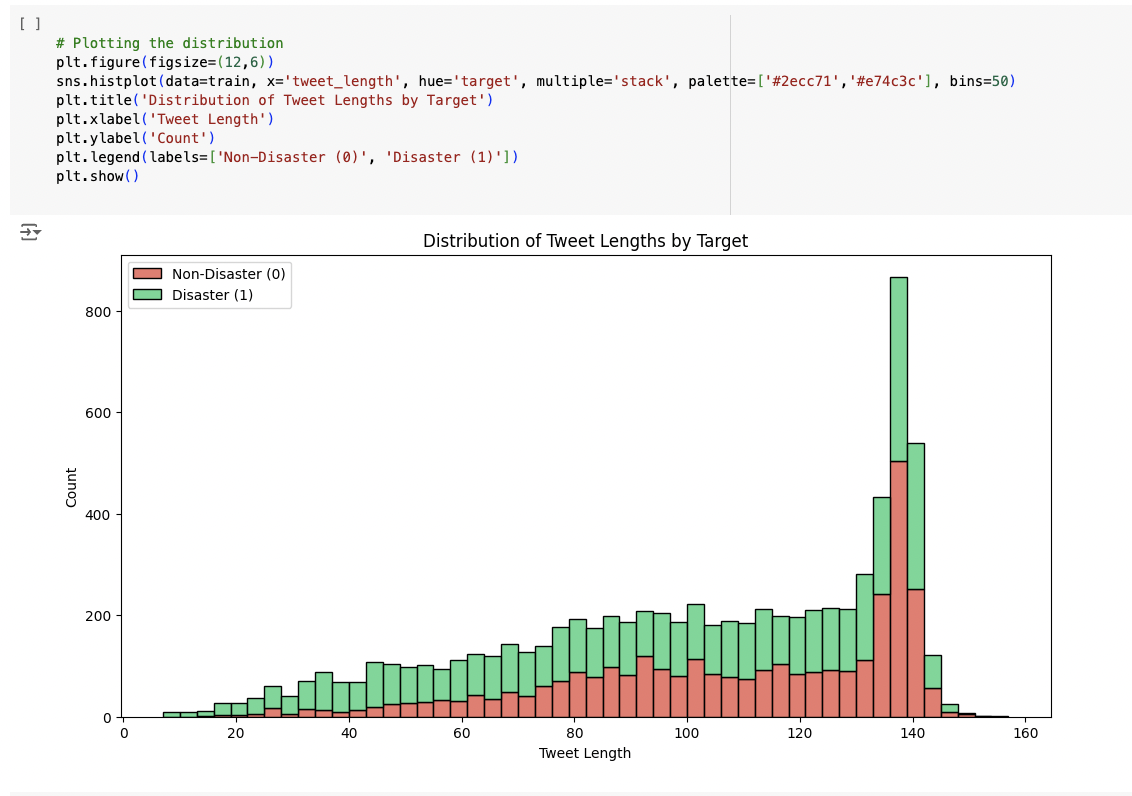
Sentiment Polarity Distribution by Target:

The sentiment analysis reveals that both disaster and non-disaster tweets predominantly have neutral sentiment polarity, though disaster-related tweets show a slightly broader distribution toward negative sentiment. This finding suggests that sentiment analysis might offer additional predictive value, particularly in distinguishing between tweets that express concern or report incidents and more neutral or positive content.



Distribution of Tweet Lengths by Target:

The distribution of tweet lengths by target shows that disaster-related tweets tend to be slightly longer on average than non-disaster tweets. This could indicate that disaster-related tweets often contain more detailed information or multiple keywords, which may be useful for model differentiation



.

Keyword Frequency by Target:

The heatmap of keyword frequency by target shows that certain keywords are strongly associated with disaster-related tweets, such as "armageddon," "deluge," and "evacuate." This relationship between specific keywords and disaster content can be leveraged to improve the model's ability to correctly classify tweets, particularly in identifying those that are genuinely related to catastrophic events.



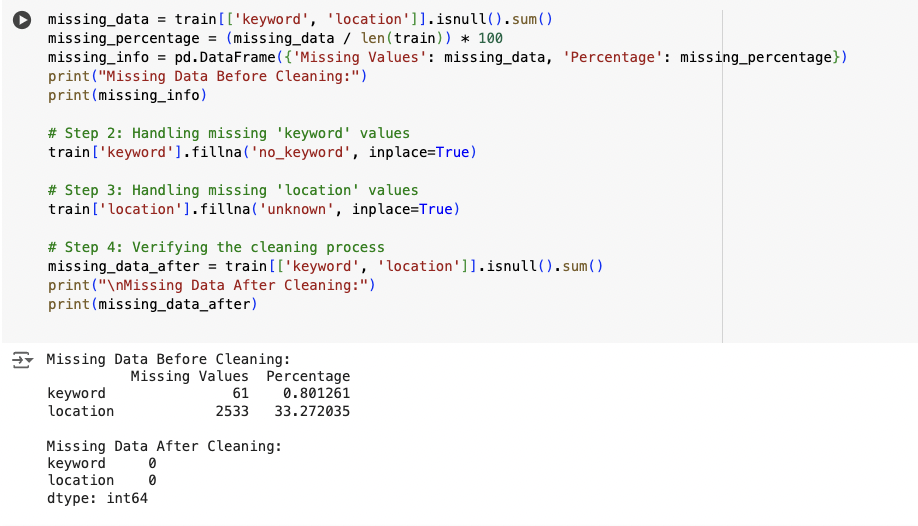
**Data Cleaning Process**

The initial data cleaning process focused on addressing missing values in critical columns such as keyword and location. The dataset contained 61 missing values in the keyword column and 2,533 missing values in the location column, representing approximately 0.8% and 33.3% of the respective data.

To handle these missing values:

1. Keyword Column: Missing entries in the keyword column were replaced with the placeholder "no\_keyword." This approach preserves the structure of the dataset while providing a clear indication that no keyword was associated with the tweet.
2. Location Column: Missing entries in the location column were similarly replaced with the placeholder "unknown," maintaining data integrity and allowing for subsequent preprocessing steps.

After this cleaning step, both columns were confirmed to have no missing values, ensuring the dataset was ready for further preprocessing.



**Data Preprocessing**

The data preprocessing process was designed to clean and prepare the text data, including the text, keyword, and location columns, for modeling. The steps involved were:

1. Text Cleaning: The text data underwent several transformations to ensure consistency and remove noise:
   * HTML Tags Removal: Any HTML tags present in the text were stripped out using BeautifulSoup.
   * Contractions Expansion: Common contractions were expanded (e.g., "can't" to "cannot") to maintain consistency in the text.
   * Punctuation Removal: All punctuation marks were removed to focus on the textual content.
   * Stopwords Removal: Common stopwords (e.g., "the," "is") were removed to reduce noise and improve model performance.
   * Numbers Removal: Any numeric characters were stripped from the text, as they were not relevant for the analysis.
   * Emoji Removal: Emojis and other special characters were removed to focus purely on textual data.
   * Normalization: Tokenization, lemmatization, and stemming were applied to normalize the text, converting it to a consistent format that facilitates modeling.
2. Keyword and Location Columns: The keyword and location columns were processed similarly to the text data. After filling in missing values with placeholders ("no\_keyword" and "unknown"), they underwent the same text cleaning steps, ensuring uniformity across the dataset.
3. Tokenization: A single Tokenizer was initialized and fit on the cleaned text from the text, keyword, and location columns. This step converts the text data into sequences of integers, where each integer represents a specific word in the vocabulary.
4. Sequence Padding: The resulting sequences were padded to a uniform length (in this case, 100 tokens) to ensure consistency across the dataset. This step is crucial for feeding the data into machine learning models, particularly neural networks, which require inputs of consistent shape.
5. Target Variable Extraction: Finally, the target variable (target) was extracted, completing the preprocessing pipeline and preparing the data for modeling.

This comprehensive preprocessing approach ensures that the data is clean, consistent, and in a suitable format for training effective machine learning models.



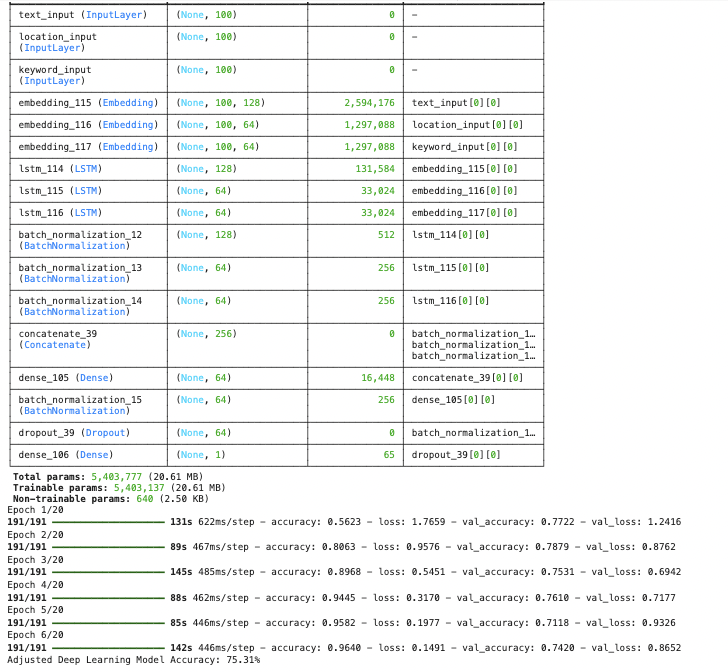
**Model Architecture and Training:**

**LSTM Model with Text, Keyword, and Location Inputs**

Architecture: The complex model is designed to handle three different types of inputs: text, keyword, and location. Here's a breakdown of the architecture:

* Text Input:
  + Embedding Layer: Converts the text input into dense vectors of fixed size.
  + LSTM Layer: Captures sequential dependencies in the text, with added dropout for regularization.
  + Batch Normalization: Normalizes the output to speed up training and improve stability.
* Keyword Input:
  + Embedding Layer: Converts keywords into dense vectors.
  + LSTM Layer: Processes the sequential information in keywords with dropout for regularization.
  + Batch Normalization: Normalizes the output.
* Location Input:
  + Embedding Layer: Converts location data into dense vectors.
  + LSTM Layer: Captures sequential dependencies in location data with dropout for regularization.
  + Batch Normalization: Normalizes the output.
* Concatenation: The outputs from the three LSTM layers are concatenated to form a combined feature vector.
* Fully Connected Layers: The concatenated features are passed through dense layers with L2 regularization and batch normalization, followed by a dropout layer to reduce overfitting.
* Output Layer: A single neuron with a sigmoid activation function is used for binary classification.

Result: This model achieved an accuracy of 75.31% on the validation set. While the model demonstrates a strong architecture, the accuracy suggests that further tuning or more sophisticated approaches may be required to improve performance on this particular dataset.

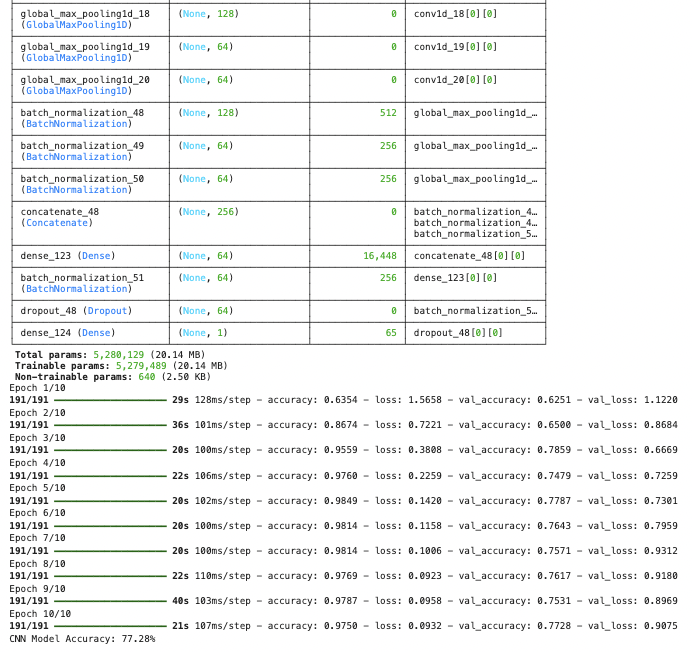


**CNN-Based Model with Text, Keyword, and Location Inputs**

Architecture: The CNN-based model also handles three types of inputs: text, keyword, and location. Here’s a breakdown of the architecture:

* Text Input:
  + Embedding Layer: Converts the text input into dense vectors.
  + Conv1D Layer: Applies convolutional filters to capture local patterns in the text.
  + Global Max Pooling: Reduces the dimensionality of the output while retaining the most critical features.
  + Batch Normalization: Normalizes the output to improve training efficiency.
* Keyword Input:
  + Embedding Layer: Converts keywords into dense vectors.
  + Conv1D Layer: Applies convolutional filters to capture local patterns in keywords.
  + Global Max Pooling: Reduces dimensionality while keeping essential features.
  + Batch Normalization: Normalizes the output.
* Location Input:
  + Embedding Layer: Converts location data into dense vectors.
  + Conv1D Layer: Applies convolutional filters to capture location patterns.
  + Global Max Pooling: Reduces dimensionality while retaining crucial features.
  + Batch Normalization: Normalizes the output.
* Concatenation: The outputs from the three CNN layers are concatenated into a combined feature vector.
* Fully Connected Layers: The concatenated features pass through dense layers with L2 regularization and batch normalization, followed by a dropout layer for regularization.
* Output Layer: A single neuron with a sigmoid activation function is used for binary classification.

Result: This CNN-based model achieved a validation accuracy of 77.28%. The use of convolutional layers allowed the model to capture more local features effectively, slightly outperforming the LSTM-based model. However, the validation accuracy suggests there’s still potential for further improvement, possibly through additional tuning or alternative model architectures.

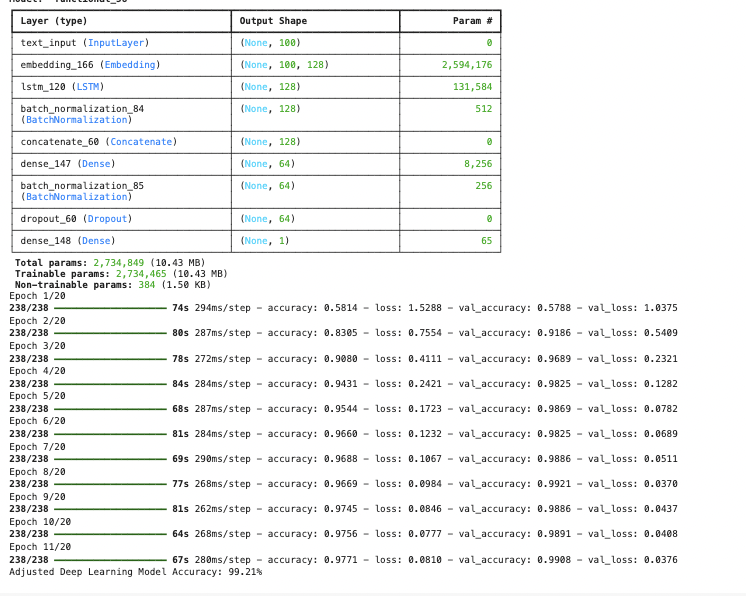


**Simple LSTM-Based Model with Text Data Only**

Architecture: The LSTM-based model is designed to process only the text input. The architecture is straightforward and focuses on capturing the sequential nature of the text data.

* Text Input:
  + Embedding Layer: Converts the text input into dense vectors of a fixed size, allowing the model to work with word representations.
  + LSTM Layer: Captures the sequential dependencies in the text, with added dropout for regularization, helping to prevent overfitting.
  + Batch Normalization: Normalizes the output from the LSTM layer, which helps in speeding up the training process and improving model stability.
* Concatenation: Since there is only one input, this step involves preparing the output for further processing by passing it directly to the next layers.
* Fully Connected Layers: The processed features are passed through dense layers with L2 regularization to control overfitting, followed by batch normalization. A dropout layer is also included to enhance generalization.
* Output Layer: A single neuron with a sigmoid activation function is used to output the probability for binary classification.

Result: This LSTM-based model achieved a validation accuracy of 99.21%. The use of LSTM layers allowed the model to effectively capture the sequential patterns in the text data, leading to a high level of accuracy in predicting disaster-related tweets.

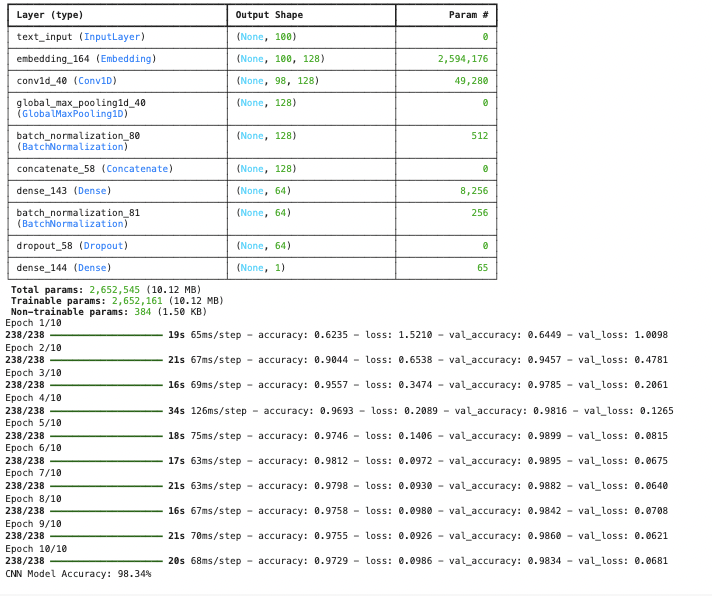


**Simple CNN-Based Model with Text Data Only**

Architecture: The CNN-based model, also focused solely on the text input, leverages convolutional layers to extract and emphasize local patterns within the text data.

* Text Input:
  + Embedding Layer: Converts the text input into dense vectors, providing a structured input for the convolutional layers.
  + Conv1D Layer: Applies convolutional filters to the embedded text, capturing local patterns such as phrases or key parts of sentences.
  + Global Max Pooling: Reduces the dimensionality of the convolutional output, retaining only the most critical features for further processing.
  + Batch Normalization: Normalizes the output from the pooling layer to ensure efficient training and improved model stability.
* Concatenation: Similar to the LSTM model, the CNN model's output is prepared for further processing by passing it directly to the subsequent layers.
* Fully Connected Layers: The features extracted by the convolutional layers are passed through dense layers with L2 regularization and batch normalization, followed by a dropout layer for regularization and to reduce overfitting.
* Output Layer: A single neuron with a sigmoid activation function is used for binary classification.

Result: This CNN-based model achieved a validation accuracy of 98.34%. The convolutional approach effectively captured local features in the text data, resulting in high performance, although slightly lower than the LSTM-based model. The simplicity of the CNN architecture made it efficient while still providing strong results.



**Other Model**

Two other models are also being tested which are GRU and Bi-LSTM in which both models are tuned and the result does not meet expectation and does not satisfy for further consideration for the model. GRU and Bi LSTM score an accuracy of 57% and 56% respectively.

**Discussion of different model**

The performance of the models reveals interesting insights into the strengths and weaknesses of different neural network architectures when applied to varying levels of data complexity.

CNN vs. LSTM in Handling Complex Data

*CNN Model with Complex Data*:

The CNN-based model, which integrates text, keyword, and location data, achieved better performance with complex inputs. Convolutional Neural Networks (CNNs) are particularly effective at detecting local patterns and structures within data, making them well-suited for scenarios where multiple input sources, such as keywords and locations, are involved.

In the context of disaster-related tweets, keywords and locations often carry critical, localized information that must be identified and processed quickly. For example, identifying a keyword like "evacuate" alongside a location such as "California" could indicate a need for immediate action. CNN's ability to apply convolutional filters across the different input types allows it to effectively detect these crucial patterns, resulting in more accurate predictions. The model's superior performance with complex data can be attributed to its capability to simultaneously process and integrate information from multiple sources, leading to better contextual understanding and decision-making in real-time.

*LSTM Model with Complex Data*:

While LSTM networks excel at capturing sequential dependencies, they may struggle when handling data that requires the integration of diverse inputs like keywords and locations. The sequential nature of LSTM makes it more suited for tasks where understanding the order and context of words within a single input type (such as text) is paramount. When dealing with multiple inputs that don't necessarily have a sequential relationship (e.g., keywords and locations), LSTMs may not fully leverage their strengths, leading to lower performance compared to CNNs.

LSTM vs. CNN in Handling Simple Data

*LSTM Model with Simple Text Data*:

When the input data is limited to text alone, the LSTM model outperformed the CNN. Long Short-Term Memory (LSTM) networks are designed to capture and retain long-range dependencies in sequential data, such as sentences or paragraphs. This ability is crucial for understanding the context in disaster-related tweets, where the meaning of a tweet can depend on the precise order and relationship between words. The LSTM model's higher accuracy with text-only data reflects its strength in processing sequences where temporal dynamics play a key role.

In scenarios where the primary goal is to provide updates on disaster issues or notify users in the early stages of an event, the text itself is often the most critical piece of information. LSTM networks, with their capacity to analyze the progression of information within a tweet, are well-suited for these tasks. They effectively capture nuances in language that might indicate the severity or nature of a disaster, making them ideal for initial notifications and updates.

*CNN Model with Simple Text Data*:

On the other hand, the CNN model, while effective at capturing local patterns within the text, may not perform as well as LSTMs when the sequential nature of the text is the primary source of information. CNNs are typically better at identifying patterns that are position-invariant, but they might miss the broader context that LSTMs can capture. This limitation is reflected in the slightly lower accuracy of the CNN model when applied to text-only data.

**Conclusion**

The results demonstrate that CNNs are more effective for complex data scenarios where multiple inputs need to be processed simultaneously to detect crucial patterns, such as identifying specific locations or keywords that require immediate action. Conversely, LSTMs shine in tasks that involve understanding the temporal dynamics and context within a single input type, such as analyzing disaster-related text data to provide early-stage notifications. This distinction underscores the importance of selecting the appropriate model architecture based on the nature of the data and the specific requirements of the task at hand.

**Week 17 Summary**

**What I have learned:**

In this week I have learned more about NLP and text analysis from social media post in which knowing how to handle the noise in text that it may cause and how keyword and location can play a major role in further detecting the threat of disaster for immediate response and notification.

**What is the biggest challenge:**

The biggest challenge I would say is due to limited data and context for further analysis and also handling different data structure for modeling this makes the result perform well in one fixed subset data, however in to application as whole, further reinforcement learning is needed for better prediction.