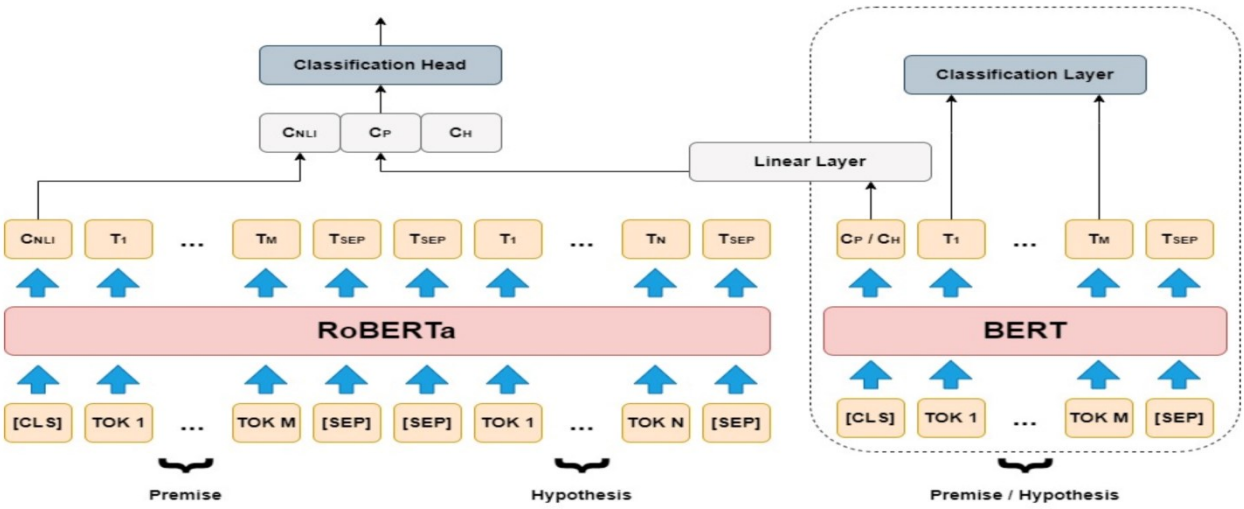
Sentiment Analysis with RoBERTa

Natural Language Processing (NLP) is a field of artificial intelligence that empowers computers to understand, interpret, and generate human language. It involves techniques like text analysis, speech recognition, and machine translation. NLP is crucial in today's data-driven world as it enables businesses to extract valuable insights from vast amounts of text data, automate tasks, and improve customer experiences.

RoBERTa (Robustly Optimized BERT Approach) is a powerful language model that can be effectively used for sentiment analysis. It builds upon the foundation of BERT, improving its performance through strategies like dynamic masking and larger training datasets.




```
import numpy as np
import pandas as pd

df = pd.read_csv("/content/data.csv",
                 encoding="utf-8",
encoding_errors="replace")
df.columns = ["text", "sentiment"]
df.head(10)
```

	text	sentiment
0	The GeoSolutions technology will leverage Bene...	positive
1	$ESI on lows, down $1.50 to $2.50 BK a real po...	negative
2	For the last quarter of 2010 , Componenta 's n...	positive
3	According to the Finnish-Russian Chamber of Co...	neutral
4	The Swedish buyout firm has sold its remaining...	neutral
5	$SPY wouldn't be surprised to see a green close	positive
6	Shell's $70 Billion BG Deal Meets Shareholder ...	negative
7	SSH COMMUNICATIONS SECURITY CORP STOCK EXCHANG...	negative
8	Kone 's net sales rose by some 14 % year-on-ye...	positive
9	The Stockmann department store will have a tot...	neutral

```
def color_sentiment(val):
    """Colors the sentiment column based on the sentiment."""
    if val == 'positive':
        color = 'lightgreen'
    elif val == 'negative':
        color = 'lightcoral'
    else:  # neutral
        color = 'lightblue'
    return f'background-color: {color}'

# Style the DataFrame
styled_df =
df.head(10).style.applymap(color_sentiment,
subset=['sentiment'])
# Display the styled DataFrame
styled_df
```

	text	sentiment
0	The GeoSolutions technology will leverage Benefon 's GPS solutions by providing Location Based Search Technology , a Communities Platform , location relevant multimedia content and a new and powerful commercial model .	positive
1	$ESI on lows, down $1.50 to $2.50 BK a real possibility	negative
2	For the last quarter of 2010 , Componenta 's net sales doubled to EUR131m from EUR76m for the same period a year earlier , while it moved to a zero pre-tax profit from a pre-tax loss of EUR7m .	positive
3	According to the Finnish-Russian Chamber of Commerce , all the major construction companies of Finland are operating in Russia .	neutral
4	The Swedish buyout firm has sold its remaining 22.4 percent stake , almost eighteen months after taking the company public in Finland .	neutral
5	$SPY wouldn't be surprised to see a green close	positive
6	Shell's $70 Billion BG Deal Meets Shareholder Skepticism	negative
7	SSH COMMUNICATIONS SECURITY CORP STOCK EXCHANGE RELEASE OCTOBER 14 , 2008 AT 2:45 PM The Company updates its full year outlook and estimates its results to remain at loss for the full year .	negative
8	Kone 's net sales rose by some 14 % year-on-year in the first nine months of 2008 .	positive
9	The Stockmann department store will have a total floor space of over 8,000 square metres and Stockmann 's investment in the project will have a price tag of about EUR 12 million .	neutral

```
def clean_sentiment_data(df):
    """
    Perform basic cleaning on sentiment analysis dataset
    """
    # Create a copy to avoid modifying original data
    cleaned_df = df.copy()

    # 1. Remove any duplicate rows
    cleaned_df = cleaned_df.drop_duplicates()

    # 2. Remove any rows with missing values
    cleaned_df = cleaned_df.dropna()

    # 3. Strip whitespace from text column
    cleaned_df['text'] = cleaned_df['text'].str.strip()

    # 4. Convert sentiment to lowercase for consistency
    cleaned_df['sentiment'] = cleaned_df['sentiment'].str.lower()

    # 5. Remove rows where text is empty after stripping
    cleaned_df = cleaned_df[cleaned_df['text'].str.len() > 0]
```

```
    # 6. Reset index after cleaning
    cleaned_df = cleaned_df.reset_index(drop=True)

    return cleaned_df

# Example usage:
cleaned_data = clean_sentiment_data(df)
display(cleaned_data.shape)
cleaned_data.head()
```

```
(5836, 2)
```

	text	sentiment
0	The GeoSolutions technology will leverage Bene...	positive
1	$ESI on lows, down $1.50 to $2.50 BK a real po...	negative
2	For the last quarter of 2010 , Componenta 's n...	positive
3	According to the Finnish-Russian Chamber of Co...	neutral
4	The Swedish buyout firm has sold its remaining...	neutral

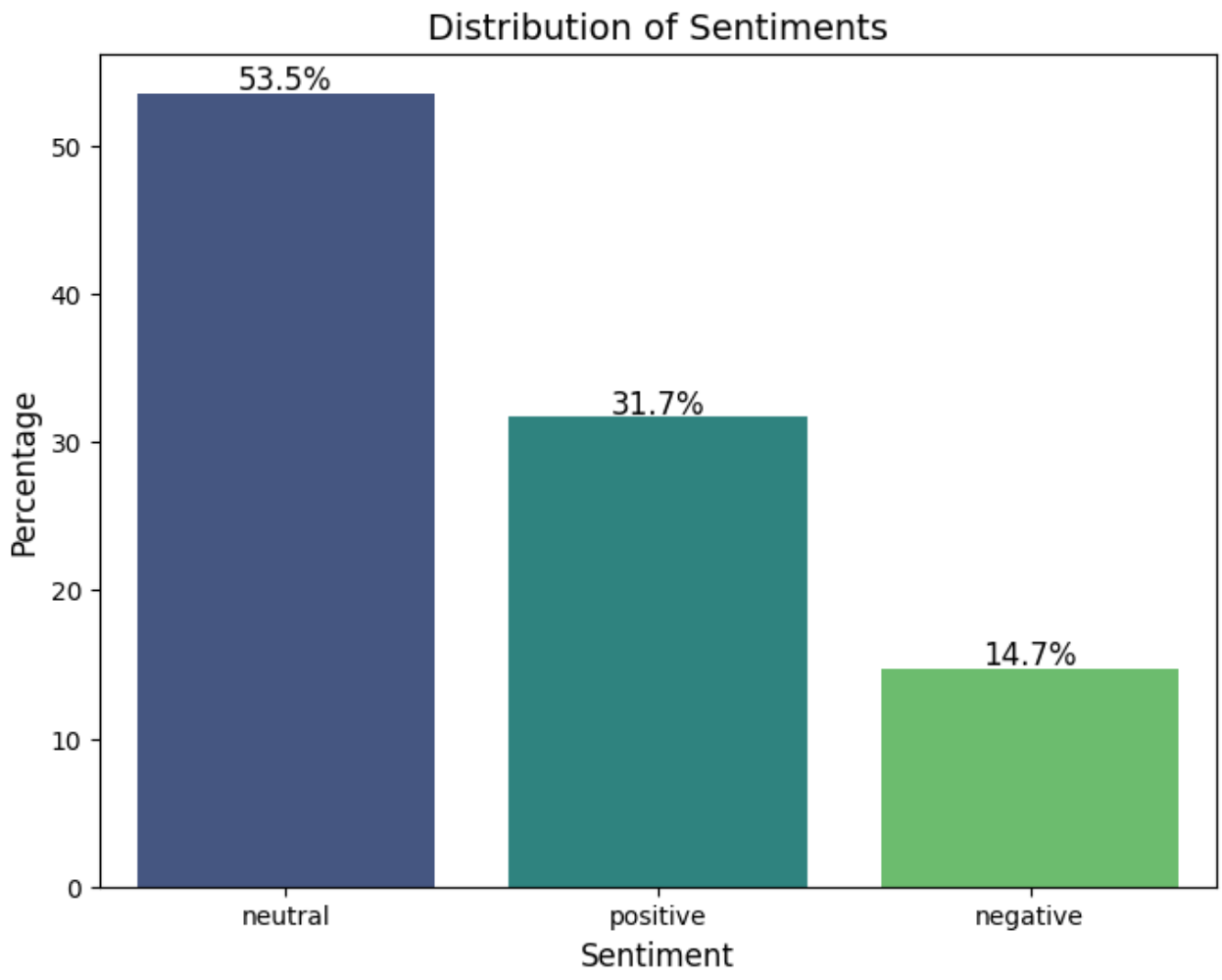
```
import seaborn as sns
import matplotlib.pyplot as plt

sentiment_counts =
cleaned_data['sentiment'].value_counts(normalize=True) * 100

plt.figure(figsize=(8, 6))
ax = sns.barplot(x=sentiment_counts.index,
y=sentiment_counts.values, palette="viridis")
```

```
# Add percentage labels to the bars
for p in ax.patches:
    ax.annotate(f'{p.get_height():.1f}%', (p.get_x() +
p.get_width() / 2., p.get_height()),
                ha='center', va='center', fontsize=12,
color='black', xytext=(0, 5),
                textcoords='offset points')


plt.xlabel("Sentiment", fontsize=12)
plt.ylabel("Percentage", fontsize=12)
plt.title("Distribution of Sentiments", fontsize=14)
plt.show()
```



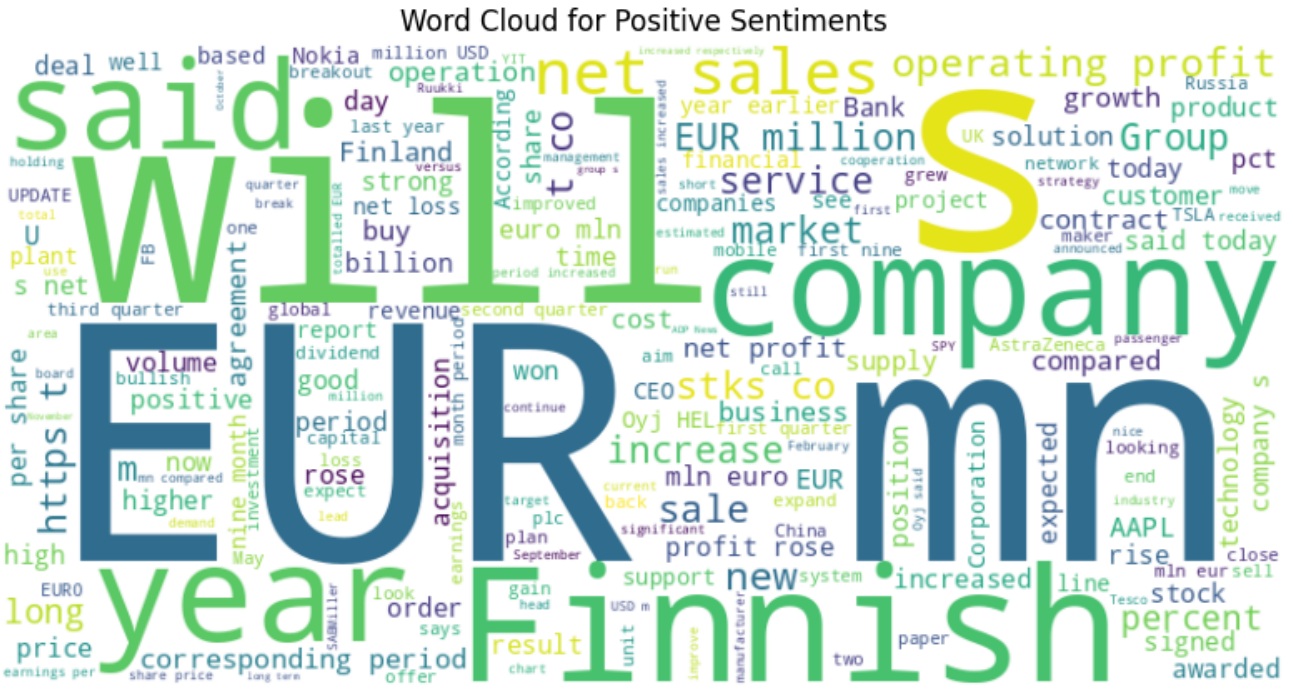
Filter for positive sentiments

```
from wordcloud import WordCloud, STOPWORDS
positive_reviews = cleaned_data[cleaned_data['sentiment'] ==
'positive']

# Combine all positive reviews into a single string
positive_text = " ".join(positive_reviews['text'])

# Create a WordCloud object
wordcloud = WordCloud(width=800, height=400,
background_color='white',
stopwords=STOPWORDS).generate(positive_text)

# Display the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud for Positive Sentiments')
plt.show()
```



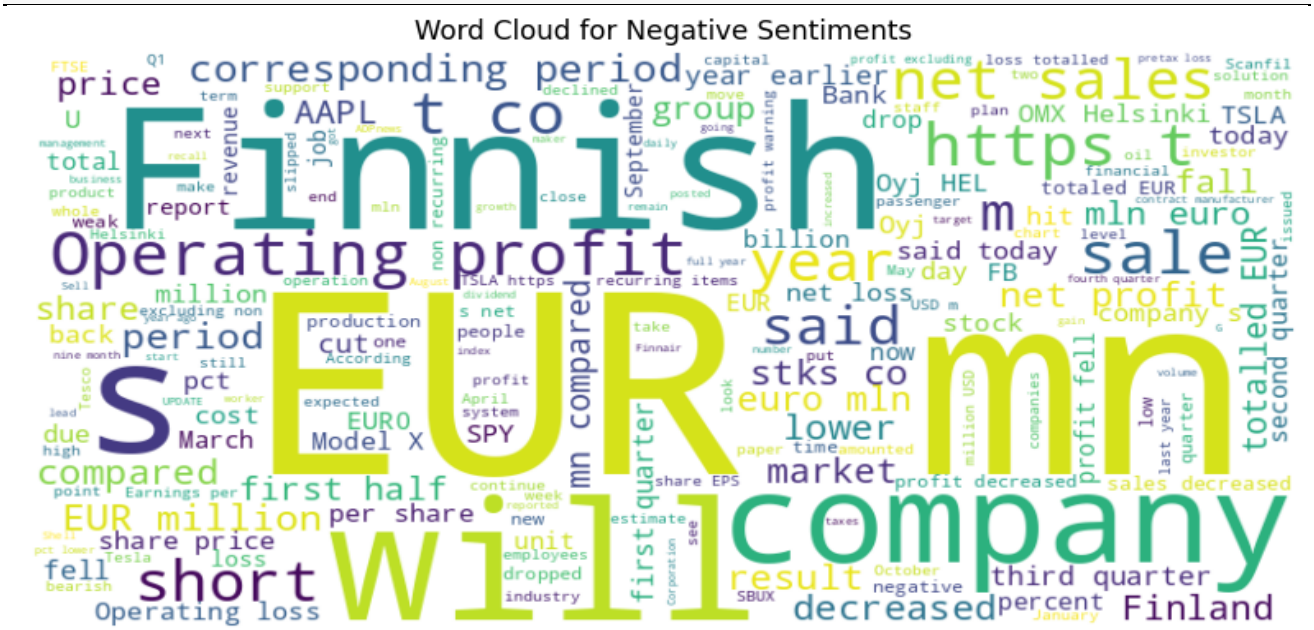
Filter for negative sentiments

```
negative_reviews =
cleaned_data[cleaned_data['sentiment'] == 'negative']

# Combine all negative reviews into a single string
negative_text = " ".join(negative_reviews['text'])

# Create a WordCloud object
wordcloud = WordCloud(width=800, height=400,
background_color='white',
stopwords=STOPWORDS).generate(negative_text)

# Display the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud for Negative Sentiments')
plt.show()
```



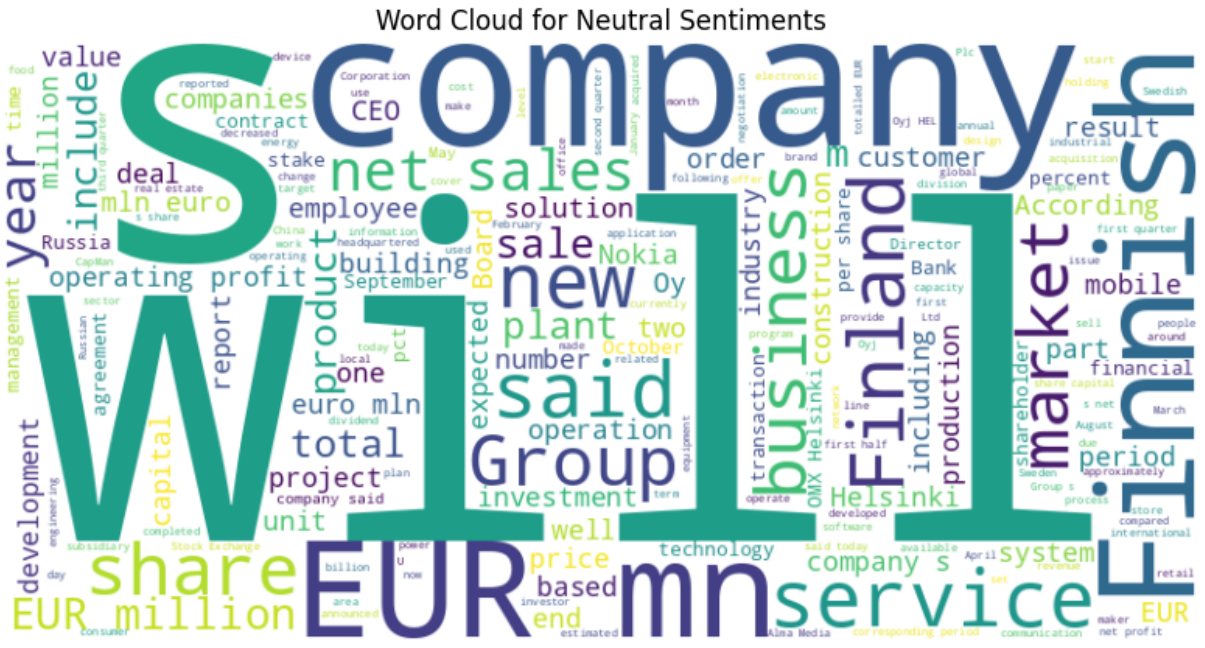
Filter for neutral sentiments

```
neutral_reviews =
cleaned_data[cleaned_data['sentiment'] == 'neutral']

# Combine all neutral reviews into a single string
neutral_text = " ".join(neutral_reviews['text'])

# Create a WordCloud object
wordcloud = WordCloud(width=800, height=400,
background_color='white',
stopwords=STOPWORDS).generate(neutral_text)

# Display the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud for Neutral Sentiments')
plt.show()
```



```
import pandas as pd
import numpy as np
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import nltk
from sklearn.feature_extraction.text import
TfidfVectorizer
from sklearn.preprocessing import LabelEncoder

# Download required NLTK data
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
True
```

```
def convert_to_lowercase(df):
    """Step 1: Convert text to lowercase"""
    df['text_lower'] = df['text'].str.lower()
    return df

df = convert_to_lowercase(cleaned_data)
df.head()
```

	text	sentiment	text_lower
0	The GeoSolutions technology will leverage Bene...	positive	the geosolutions technology will leverage bene...
1	$ESI on lows, down $1.50 to $2.50 BK a real po...	negative	$esi on lows, down $1.50 to $2.50 bk a real po...
2	For the last quarter of 2010 , Componenta 's n...	positive	for the last quarter of 2010 , componenta 's n...
3	According to the Finnish-Russian Chamber of Co...	neutral	according to the finnish-russian chamber of co...
4	The Swedish buyout firm has sold its remaining...	neutral	the swedish buyout firm has sold its remaining...

Why Text is Lowercased in NLP

Lowercasing text in NLP is a common pre-processing step that offers several key advantages:

1. **Reduces Data Sparsity:**
    - NLP models often rely on word frequencies to learn patterns and relationships.
    - Having the same word in different cases (e.g., "Apple", "apple") can lead to multiple entries, reducing their overall frequency and potentially hindering the model's ability to learn effectively.
    - Lowercasing ensures that all occurrences of a word are treated as the same, increasing their frequency and improving model performance.
2. **Simplifies Analysis:**
    - Lowercasing simplifies text analysis by treating variations of the same word as identical.
    - This makes it easier to compare words, identify patterns, and perform tasks like stemming and lemmatization, which aim to reduce words to their root forms.
3. **Improves Model Performance:**
    - In many NLP tasks, such as text classification, sentiment analysis, and machine translation, lowercasing can significantly improve model accuracy.
    - By reducing the number of unique words and focusing on their semantic meaning, models can learn more robust representations and make better predictions.

Use Cases and Advantages:

- **Text Classification:** Lowercasing helps models identify the same word regardless of its case, improving classification accuracy.
- **Sentiment Analysis:** Lowercasing ensures that sentiment-bearing words are treated consistently, leading to more accurate sentiment predictions.
- **Machine Translation:** Lowercasing simplifies the translation process by reducing the number of word variations that need to be considered.
- **Information Retrieval:** Lowercasing improves search accuracy by ensuring that queries and documents are compared consistently.
- **Natural Language Understanding:** Lowercasing aids in understanding the underlying meaning of text by focusing on the semantic content rather than superficial case variations.

```
def remove_special_characters(df):
    """Step 2: Remove special characters and numbers"""
    df['text_clean'] = df['text_lower'].apply(lambda x: re.sub(r'[^a-zA-Z\s]', '', x))
    return df

df = remove_special_characters(df)
df.head()
```

	text	sentiment	text_clean
0	The GeoSolutions technology will leverage Bene...	positive	the geosolutions technology will leverage bene...
1	$ESI on lows, down $1.50 to $2.50 BK a real po...	negative	esi on lows down to bk a real possibility
2	For the last quarter of 2010 , Componenta 's n...	positive	for the last quarter of componenta s net sal...
3	According to the Finnish-Russian Chamber of Co...	neutral	according to the finnishrussian chamber of com...
4	The Swedish buyout firm has sold its remaining...	neutral	the swedish buyout firm has sold its remaining...

Removing Special Characters and Numbers is Important in NLP Sentiment Analysis

Removing special characters and numbers in NLP sentiment analysis is a crucial pre-processing step for several reasons:

1. **Focus on Meaningful Words:** Special characters and numbers often do not carry significant semantic meaning for sentiment analysis. By removing them, we can focus the model's attention on the core words and phrases that express sentiment.
2. **Noise Reduction:** Special characters and numbers can introduce noise into the data, potentially misleading the model. For example, a string of exclamation marks might be interpreted as strong positive sentiment, even if the surrounding words are neutral. Removing these characters helps reduce such noise and improve the model's accuracy.
3. **Feature Engineering:** Sentiment analysis models often rely on word frequencies or embedding's to learn sentiment. Removing special characters and numbers can help reduce the dimensionality of the feature space, making it easier for the model to learn meaningful patterns.
4. **Model Consistency:** Different datasets may contain different special characters and numbers, leading to inconsistencies in the model's training and evaluation. By removing these characters, we can ensure that the model is trained and evaluated on a consistent set of features.

```
def remove_urls(df):
    """Step 3: Remove URLs"""
    df['text_no_urls'] = df['text_clean'].apply(lambda x:
re.sub(r'http\S+|www.\S+', '', x))
    return df

df = remove_urls(df)
df.head()
```

	text	sentiment	text_no_urls
0	The GeoSolutions technology will leverage Bene...	positive	the geosolutions technology will leverage bene...
1	$ESI on lows, down $1.50 to $2.50 BK a real po...	negative	esi on lows down to bk a real possibility
2	For the last quarter of 2010 , Componenta 's n...	positive	for the last quarter of componenta s net sal...
3	According to the Finnish-Russian Chamber of Co...	neutral	according to the finnishrussian chamber of com...
4	The Swedish buyout firm has sold its remaining...	neutral	the swedish buyout firm has sold its remaining...

Removing URLs is Important in NLP Sentiment Analysis

Removing URLs from text data is a crucial step in Natural Language Processing (NLP) sentiment analysis for several key reasons:

1. **Irrelevance to Sentiment:** URLs typically represent external links or references. They don't directly contribute to the sentiment expressed within the text itself. Including them can introduce noise and potentially mislead the sentiment analysis model.
2. **Model Focus:** Sentiment analysis models aim to understand and interpret the human language within the text. URLs can distract the model, diverting its attention from the core sentiment-bearing words and phrases.
3. **Data Consistency:** By removing URLs, you ensure that the text data is more consistent and uniform. This can improve the model's ability to learn patterns and relationships between words that are truly indicative of sentiment.
4. **Feature Engineering:** Some NLP techniques involve creating features based on the presence or absence of certain words or phrases. URLs, being essentially noise, can introduce spurious features that don't contribute to accurate sentiment prediction.
5. **Model Performance:** In many cases, removing URLs has been shown to improve the accuracy and performance of sentiment analysis models. This is because the models can focus on the relevant linguistic cues without being distracted by irrelevant information.

In Summary:

Removing URLs is a standard text preprocessing step in NLP sentiment analysis. It helps to:

- **Reduce noise:** By eliminating irrelevant information
- **Improve model focus:** By directing attention to sentiment-bearing words
- **Enhance data consistency:** By creating a more uniform dataset
- **Optimize feature engineering:** By avoiding spurious features
- **Boost model performance:** By enabling the model to learn more effectively

```
def remove_extra_whitespace(df):
    """Step 4: Remove extra whitespace"""
    df['text_stripped'] = df['text_no_urls'].apply(lambda x: ' '.join(x.split()))
    return df

df = remove_extra_whitespace(df)
df.head()
```

	text	sentiment	text_stripped
0	The GeoSolutions technology will leverage Bene...	positive	the geosolutions technology will leverage bene...
1	$ESI on lows, down $1.50 to $2.50 BK a real po...	negative	esi on lows down to bk a real possibility
2	For the last quarter of 2010 , Componenta 's n...	positive	for the last quarter of componenta s net sales...
3	According to the Finnish-Russian Chamber of Co...	neutral	according to the finnishrussian chamber of com...
4	The Swedish buyout firm has sold its remaining...	neutral	the swedish buyout firm has sold its remaining...

Importance of Removing Extra Whitespace in NLP Sentiment Analysis:

Removing extra whitespace is crucial in NLP sentiment analysis for several reasons:

1. **Consistent Representation:** Extra whitespace can lead to inconsistencies in how words are represented. For example, "hello world" and "hello world" might be treated as different words by the NLP model, affecting the accuracy of sentiment analysis.
2. **Improved Model Performance:** Most NLP models, including those used for sentiment analysis, rely on tokenization (breaking down text into individual words or subwords). Extra whitespace can interfere with this process, leading to incorrect tokenization and potentially misleading results.
3. **Efficient Processing:** Removing extra whitespace can improve the efficiency of NLP pipelines by reducing the amount of data that needs to be processed. This can be particularly important when dealing with large datasets.
4. **Better Interpretability:** Cleaned text with consistent spacing is easier for both humans and machines to read and interpret, making it easier to understand the results of sentiment analysis.

By removing extra whitespace, we ensure that the text is processed consistently and accurately by the NLP model, leading to more reliable sentiment analysis results.

```
def tokenize_text(df):
    """Step 5: Tokenization"""
    df['tokens'] = df['text_stripped'].apply(word_tokenize)
    return df
df = tokenize_text(df)
df.head()
```

	text	sentiment	tokens
0	The GeoSolutions technology will leverage Bene...	positive	[the, geosolutions, technology, will, leverage...
1	$ESI on lows, down $1.50 to $2.50 BK a real po...	negative	[esi, on, lows, down, to, bk, a, real, possibi...
2	For the last quarter of 2010 , Componenta 's n...	positive	[for, the, last, quarter, of, componenta, s, n...
3	According to the Finnish-Russian Chamber of Co...	neutral	[according, to, the, finnishrussian, chamber, ...
4	The Swedish buyout firm has sold its remaining...	neutral	[the, swedish, buyout, firm, has, sold, its, r...

Importance of Tokenization in NLP Sentiment Analysis

The provided code snippet defines a function tokenize_text(df) that performs tokenization on a given DataFrame df. It creates a new column tokens by applying the word_tokenize function from the nltk library to the text_stripped column.

Tokenization is a fundamental step in NLP, particularly for sentiment analysis, due to several key reasons:

1. **Breaking Down Text into Meaningful Units:**
    - Human language is composed of sentences, which are further divided into words. Tokenization breaks down the continuous stream of text into individual words (or subwords), making it easier for NLP models to process and understand the underlying meaning.
2. **Enabling Further NLP Tasks:**
    - Tokenization is a prerequisite for many subsequent NLP tasks, including:
        - **Part-of-speech tagging:** Identifying the grammatical role of each word (e.g., noun, verb, adjective).
        - **Named entity recognition:** Identifying and classifying named entities (e.g., people, organizations, locations).
        - **Sentiment analysis:** Analyzing the emotional tone or polarity of the text.
3. **Feature Engineering for Machine Learning:**

- Tokenized words can be used as features for machine learning models. For example, in sentiment analysis, we can create a vocabulary of words and use their frequencies or presence in a document as features to train a classifier.

4. **Handling Different Languages:**
   - Tokenization can be adapted to handle different languages, which may have different rules for word boundaries and sentence structures.

In summary, tokenization is a crucial step in NLP sentiment analysis as it prepares the text data for further processing and analysis by breaking it down into meaningful units, enabling subsequent NLP tasks, and providing features for machine learning models.

```
import nltk
from nltk.corpus import wordnet
import os

def lemmatize_text(df):
    """Step 7: Lemmatization (after ensuring WordNet corpus is downloaded)"""
    wordnet_path = os.path.join('corpora', 'wordnet')  # Create the relative path to 'wordnet'
    if not any(os.path.exists(os.path.join(path, wordnet_path)) for path in nltk.data.path):
        print("WordNet corpus not found. Downloading...")
        nltk.download('wordnet')
    lemmatizer = WordNetLemmatizer()
    df['lemmatized'] = df['tokens_no_stop'].apply(lambda x: [lemmatizer.lemmatize(word) for word in x])
```

```
    return df

df = lemmatize_text(df)
df.head()
```

	text	sentiment	lemmatized
0	The GeoSolutions technology will leverage Bene...	positive	[geosolutions, technology, leverage, benefon, ...
1	$ESI on lows, down $1.50 to $2.50 BK a real po...	negative	[esi, low, bk, real, possibility]
2	For the last quarter of 2010 , Componenta 's n...	positive	[last, quarter, componenta, net, sale, doubled...
3	According to the Finnish-Russian Chamber of Co...	neutral	[according, finnishrussian, chamber, commerce,...
4	The Swedish buyout firm has sold its remaining...	neutral	[swedish, buyout, firm, sold, remaining, perce...

Importance of Lemmatization in Sentiment Analysis

The provided code snippet defines a function called lemmatize_text that takes a DataFrame df as input and adds a new column named "lemmatized" containing the lemmas of the words in the "tokens_no_stop" column. Lemmatization is an important step in Natural Language Processing (NLP), especially for tasks like sentiment analysis. Here's why:

What is Lemmatization?

Lemmatization refers to the process of reducing a word to its base or dictionary form. For example, the lemma of "running" is "run," the lemma of "better" is "good," and the lemma of "dogs" is "dog."

Importance of Lemmatization in Sentiment Analysis

Sentiment analysis aims to understand the sentiment of a text, whether it's positive, negative, or neutral. When dealing with text data, words often appear in different inflected forms (e.g., verbs with different tenses, nouns with plurals, adjectives with comparatives/superlatives). These variations can affect how sentiment analysis tools interpret the text's sentiment.

Here's how lemmatization helps in sentiment analysis:

1. **Reduces Vocabulary Size:** By converting words to their base forms, lemmatization reduces the number of unique words the sentiment analysis model needs to consider. This can improve the efficiency and accuracy of the model.
2. **Improves Accuracy:** By considering the lemma instead of the inflected form, the sentiment analysis tool can better understand the core meaning of the word and its contribution to the overall sentiment. For example, "better" and "good" convey similar sentiment, and lemmatization ensures they are treated similarly.
3. **Allows for Lexicon Matching:** Sentiment analysis often relies on sentiment lexicons, which are lists of words with pre-assigned sentiment scores. Lemmatization ensures that words in the text and lexicon match even if they appear in different inflected forms.

In essence, lemmatization helps standardize the text data by reducing words to their base forms, leading to more consistent and accurate sentiment analysis.

Example (without actual code execution):

Consider the sentence "I am feeling very happy today!".

- Without lemmatization: The sentiment analysis tool might assign different sentiment scores to "feeling" and "happy" based on their individual inflected forms.
- With lemmatization: By converting "feeling" to "feel" and "happy" to "happy" (their lemmas), the tool can better capture the overall positive sentiment of the sentence.

Lemmatization is a crucial step in NLP tasks like sentiment analysis, as it helps improve the accuracy and efficiency of the analysis by ensuring consistent treatment of words with different inflections.

```
def join_tokens(df):
    """Step 8: Join tokens back to text"""
    df['processed_text'] = df['lemmatized'].apply(lambda
x: ' '.join(x))
    return df

df = join_tokens(df)
df.head()
```

	text	sentiment	processed_text
0	The GeoSolutions technology will leverage Bene...	positive	geosolutions technology leverage benefon gps s...
1	$ESI on lows, down $1.50 to $2.50 BK a real po...	negative	esi low bk real possibility
2	For the last quarter of 2010 , Componenta 's n...	positive	last quarter componenta net sale doubled eurm ...
3	According to the Finnish-Russian Chamber of Co...	neutral	according finnishrussian chamber commerce majo...
4	The Swedish buyout firm has sold its remaining...	neutral	swedish buyout firm sold remaining percent sta...

```
def remove_stopwords(df):
    """Step 6: Remove stopwords"""
    stop_words = set(stopwords.words('english'))
    df['tokens_no_stop'] = df['tokens'].apply(lambda x:
[word for word in x if word.lower() not in stop_words])
    return df

df = remove_stopwords(df)
df.head()
```

	text	sentiment	tokens_no_stop
0	The GeoSolutions technology will leverage Bene...	positive	[geosolutions, technology, leverage, benefon, ...
1	$ESI on lows, down $1.50 to $2.50 BK a real po...	negative	[esi, lows, bk, real, possibility]
2	For the last quarter of 2010 , Componenta 's n...	positive	[last, quarter, componenta, net, sales, double...
3	According to the Finnish-Russian Chamber of Co...	neutral	[according, finnishrussian, chamber, commerce,...
4	The Swedish buyout firm has sold its remaining...	neutral	[swedish, buyout, firm, sold, remaining, perce...

Importance and Use of Removing Stopwords in NLP Sentiment Analysis:

- **Reduced Noise:** Stopwords are common words that don't carry significant meaning in the context of sentiment analysis. Removing them helps to:
  - **Improve accuracy:** By focusing on the more informative words, sentiment analysis models can better identify the true sentiment expressed in the text.
  - **Reduce dimensionality:** Removing stopwords reduces the number of features (words) that the model needs to process, which can improve model performance and efficiency.
- **Enhanced Interpretability:**
  - The resulting analysis is more concise and easier to understand as it focuses on the core words that convey the sentiment.
- **Improved Model Efficiency:**
  - By removing irrelevant words, the model can be trained and run faster.

In Summary:

Removing stopwords is a crucial preprocessing step in NLP sentiment analysis. It helps to improve the accuracy, efficiency, and interpretability of the analysis by focusing on the most meaningful words in the text.

```
def encode_labels(df):
    """Step 9: Encode sentiment labels"""
    le = LabelEncoder()
    df['sentiment_encoded'] =
le.fit_transform(df['sentiment'])
    return df, le

df, label_encoder = encode_labels(df)
df.head()
```

	text	sentiment	sentiment_encoded
0	The GeoSolutions technology will leverage Bene...	positive	2
1	$ESI on lows, down $1.50 to $2.50 BK a real po...	negative	0
2	For the last quarter of 2010 , Componenta 's n...	positive	2
3	According to the Finnish-Russian Chamber of Co...	neutral	1
4	The Swedish buyout firm has sold its remaining...	neutral	1

Encoding Sentiment Labels in NLP Machine Learning: Why It's Important

In Natural Language Processing (NLP) machine learning tasks, especially those involving sentiment analysis or text classification, encoding sentiment labels is a crucial step. Here's why:

1. Machine Learning Models Require Numerical Input:

- Most machine learning algorithms, such as logistic regression, support vector machines, and neural networks, operate on numerical data.
- Sentiment labels, often represented as textual categories like "positive," "negative," or "neutral," need to be converted into numerical values for these models to process them effectively.

2. Label Encoding Facilitates Model Training:

- Label encoding assigns a unique integer to each distinct sentiment category.
- This numerical representation allows the model to learn patterns and relationships between the encoded labels and the input text data during the training process.

3. Improved Model Performance:

- By providing numerical representations of sentiment labels, label encoding enables the model to make more accurate predictions.

- The model can learn to associate specific numerical values with different sentiment categories, leading to better classification performance.

4. Consistent Representation:

- Label encoding ensures consistent representation of sentiment labels across the dataset, preventing inconsistencies that could hinder model training and performance.

In summary:

Encoding sentiment labels is a vital preprocessing step in NLP machine learning. It transforms categorical labels into numerical values, making them suitable for input to machine learning algorithms. This process is essential for effective model training and improved prediction accuracy in sentiment analysis and other text classification tasks.

```
def create_tfidf(df, max_features=5000):
    """Step 10: TF-IDF Vectorization"""
    tfidf = TfidfVectorizer(max_features=max_features)
    tfidf_matrix = tfidf.fit_transform(df['processed_text'])
    return tfidf_matrix, tfidf

tfidf_matrix, tfidf_vectorizer = create_tfidf(df)
df.head()
```

	text	sentiment	text_lower	text_clean	text_no_urls	text_stripped	tokens	tokens_no_stop	lemmatized	sentiment_encoded	processed_text
0	The GeoSolutions technology will leverage Bene...	positive	the geosolutions technology will leverage bene...	the geosolutions technology will leverage bene...	the geosolutions technology will leverage bene...	the geosolutions technology will leverage bene...	[the, geosolutions, technology, will, leverage...	[geosolutions, technology, leverage, benefon, ...	[geosolutions, technology, leverage, benefon, ...	2	geosolutions technology leverage benefon gps s...
1	$ESI on lows, down $1.50 to $2.50 BK a real po...	negative	$esi on lows, down $1.50 to $2.50 bk a real po...	esi on lows down to bk a real possibility	esi on lows down to bk a real possibility	esi on lows down to bk a real possibility	[esi, on, lows, down, to, bk, a, real, possibi...	[esi, lows, bk, real, possibility]	[esi, low, bk, real, possibility]	0	esi low bk real possibility
2	For the last quarter of 2010 , Componenta 's n...	positive	for the last quarter of 2010 , componenta 's n...	for the last quarter of componenta s net sal...	for the last quarter of componenta s net sal...	for the last quarter of componenta s net sales...	[for, the, last, quarter, of, componenta, s, n...	[last, quarter, componenta, net, sales, double...	[last, quarter, componenta, net, sale, doubled...	2	last quarter componenta net sale doubled eurm ...
3	According to the Finnish-Russian Chamber of Co...	neutral	according to the finnish-russian chamber of co...	according to the finnishrussian chamber of com...	according to the finnishrussian chamber of com...	according to the finnishrussian chamber of com...	[according, to, the, finnishrussian, chamber, ...	[according, finnishrussian, chamber, commerce,...	[according, finnishrussian, chamber, commerce,...	1	according finnishrussian chamber commerce majo...
4	The Swedish buyout firm has sold its remaining...	neutral	the swedish buyout firm has sold its remaining...	the swedish buyout firm has sold its remaining...	the swedish buyout firm has sold its remaining...	the swedish buyout firm has sold its remaining...	[the, swedish, buyout, firm, has, sold, its, r...	[swedish, buyout, firm, sold, remaining, perce...	[swedish, buyout, firm, sold, remaining, perce...	1	swedish buyout firm sold remaining percent sta...

Why is TF-IDF Important in Machine Learning?

- **TF (Term Frequency):** How often a word appears within a single document.
- **IDF (Inverse Document Frequency):** Measures how rare a word is across the entire corpus of documents.

1. **Feature Engineering for Text Data:**
    - **Transforms Text to Numerical Vectors:** Machine learning models primarily work with numerical data. TF-IDF converts text data into meaningful numerical vectors, where each dimension represents the importance of a word in a document.
    - **Handles Word Frequency Bias:**
        - Common words (like "the," "a," "is") often appear frequently but carry little semantic meaning.
        - IDF downweights these common words, giving more importance to rare words that are more likely to be relevant to the document's topic.
2. **Improved Model Performance:**
    - **Better Feature Representation:** By capturing the importance of words within a document and across the corpus, TF-IDF provides a more informative representation of the text data.
    - **Enhanced Model Accuracy:** Models trained on TF-IDF vectors often exhibit improved accuracy and performance in various NLP tasks, such as:
        - **Text Classification:** Categorizing documents into different classes (e.g., sentiment analysis, topic classification).
        - **Information Retrieval:** Ranking documents based on their relevance to a query (e.g., search engines).
        - **Clustering:** Grouping similar documents together.

In Summary

TF-IDF is a crucial technique in machine learning for effectively handling text data. By converting text into meaningful numerical representations, it enables the application of various machine learning models to solve a wide range of NLP problems.

RoBERTa for Sentiment Analysis

RoBERTa is a powerful transformer-based language model that has shown exceptional performance in various natural language processing (NLP) tasks, including sentiment analysis. It builds upon the foundation of BERT (Bidirectional Encoder Representations from Transformers) and incorporates several key improvements:

Key Improvements in RoBERTa:

1. **Larger Training Dataset:** RoBERTa is trained on a significantly larger dataset than BERT, exposing it to a wider range of linguistic patterns and nuances. This extensive training helps it capture more subtle and complex sentiment expressions.
2. **Dynamic Masking:** RoBERTa employs dynamic masking during training, where the masked tokens are randomly selected for each training epoch. This introduces more variability into the training process, forcing the model to learn more robust representations.
3. **Longer Training Sequences:** RoBERTa is trained on longer sequences, allowing it to capture longer-range dependencies within the text, which can be crucial for understanding sentiment in longer sentences or paragraphs.
4. **Removal of Next Sentence Prediction:** RoBERTa removes the next sentence prediction task, focusing solely on language modeling. This simplifies the training objective and allows the model to allocate more resources to improving language understanding.

How RoBERTa is Used for Sentiment Analysis:

1. **Fine-tuning:** A pre-trained RoBERTa model is fine-tuned on a specific sentiment analysis dataset. This involves adjusting the model's parameters to better suit the nuances of the target sentiment classification task.
2. **Text Encoding:** The input text is encoded into a sequence of numerical representations, which are then fed into the fine-tuned RoBERTa model.
3. **Sentiment Prediction:** The RoBERTa model processes the input sequence and generates a probability distribution over different sentiment classes (e.g., positive, negative, neutral). The class with the highest probability is typically selected as the predicted sentiment.

Benefits of Using RoBERTa for Sentiment Analysis:

- **High Accuracy:** RoBERTa's strong language understanding capabilities and extensive training enable it to achieve high accuracy in sentiment analysis tasks, even in challenging scenarios.
- **Robustness:** The dynamic masking and larger training dataset make RoBERTa more robust to variations in language style and sentiment expression.
- **Efficiency:** RoBERTa can be efficiently fine-tuned on specific sentiment analysis datasets, allowing for rapid adaptation to new tasks.
- **Versatility:** RoBERTa can be applied to a wide range of sentiment analysis tasks, including binary classification, multi-class classification, and fine-grained sentiment analysis.

In Summary:

RoBERTa is a state-of-the-art language model that has significantly advanced the field of sentiment analysis. Its ability to capture complex linguistic patterns and nuances, combined with its robustness and efficiency, makes it a valuable tool for a wide range of NLP applications.

```
%%time

import pandas as pd
import numpy as np
import torch
from torch.utils.data import Dataset, DataLoader
from transformers import RobertaTokenizer,
RobertaForSequenceClassification, AdamW
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split

def create_balanced_dataset(df, n_samples=5000):
    """
    Create a balanced dataset with specified number of samples
    """
    # Convert sentiment to numeric
    sentiment_map = {
        'negative': 0,
        'neutral': 1,
        'positive': 2
```

```
    }
    df['label'] = df['sentiment'].map(sentiment_map)

    # Calculate samples per class
    samples_per_class = n_samples // 3

    # Get balanced data for each class
    balanced_dfs = []
    for label in range(3):
        class_df = df[df['label'] == label]
        if len(class_df) > samples_per_class:
            balanced_dfs.append(class_df.sample(n=samples_per_class,
random_state=42))
        else:
            # If we don't have enough samples, oversample
            balanced_dfs.append(class_df.sample(n=samples_per_class,
replace=True, random_state=42))

    # Combine balanced datasets
    balanced_df = pd.concat(balanced_dfs)

    # Shuffle the final dataset
    return balanced_df.sample(frac=1, random_state=42)

class SentimentDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len=128):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len

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

    def __getitem__(self, idx):
        text = str(self.texts[idx])
        label = self.labels[idx]

        encoding = self.tokenizer.encode_plus(
            text,
            add_special_tokens=True,
```

```
            max_length=self.max_len,
            return_token_type_ids=False,
            padding='max_length',
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt',
        )

        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'labels': torch.tensor(label, dtype=torch.long)
        }

def train_sentiment_model(df, test_size=0.2, num_epochs=3,
batch_size=16, learning_rate=2e-5):
    # Create balanced dataset
    print("Creating balanced dataset...")
    balanced_df = create_balanced_dataset(df, n_samples=5000)
    print(f"Class distribution:\n{balanced_df['label'].value_counts()}")

    # Split the data
    train_df, test_df = train_test_split(balanced_df, test_size=test_size,
random_state=42, stratify=balanced_df['label'])

    # Get texts and labels
    train_texts = train_df['text'].tolist()
    train_labels = train_df['label'].tolist()
    test_texts = test_df['text'].tolist()
    test_labels = test_df['label'].tolist()

    # Initialize tokenizer
    print("Initializing RoBERTa tokenizer...")
    tokenizer = RobertaTokenizer.from_pretrained('roberta-base')

    # Create datasets
    train_dataset = SentimentDataset(train_texts, train_labels, tokenizer)
    test_dataset = SentimentDataset(test_texts, test_labels, tokenizer)

    # Create data loaders
```

```
    train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)
    test_loader = DataLoader(test_dataset, batch_size=batch_size,
shuffle=False)

    # Initialize model
    print("Initializing RoBERTa model...")
    model = RobertaForSequenceClassification.from_pretrained('roberta-
base', num_labels=3)
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model.to(device)

    # Set up optimizer
    optimizer = AdamW(model.parameters(), lr=learning_rate)

    # Training loop
    print(f"Training on {device}")
    for epoch in range(num_epochs):
        model.train()
        total_loss = 0
        print(f"\nEpoch {epoch + 1}/{num_epochs}")

        # Training
        for batch_idx, batch in enumerate(train_loader):
            optimizer.zero_grad()
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)

            outputs = model(input_ids, attention_mask=attention_mask,
labels=labels)
            loss = outputs.loss
            total_loss += loss.item()

            loss.backward()
            optimizer.step()

            if (batch_idx + 1) % 10 == 0:
                print(f"Batch {batch_idx + 1}/{len(train_loader)}, Loss:
{loss.item():.4f}")
```

```
        # Evaluation
        model.eval()
        test_preds = []
        test_true = []

        print("\nEvaluating...")
        with torch.no_grad():
            for batch in test_loader:
                input_ids = batch['input_ids'].to(device)
                attention_mask = batch['attention_mask'].to(device)
                labels = batch['labels']

                outputs = model(input_ids, attention_mask=attention_mask)
                preds = torch.argmax(outputs.logits, dim=1).cpu().numpy()

                test_preds.extend(preds)
                test_true.extend(labels.numpy())

        # Calculate metrics
        accuracy = accuracy_score(test_true, test_preds)
        print(f'Average training loss: {total_loss/len(train_loader):.4f}')
        print(f'Test Accuracy: {accuracy:.4f}')

        # Detailed classification report
        print('\nClassification Report:')
        print(classification_report(test_true, test_preds,
                                    target_names=['Negative', 'Neutral', 'Positive']))

    # Save the model
    print("\nSaving model...")
    torch.save({
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'tokenizer': tokenizer
    }, 'roberta_sentiment_model_balanced.pth')

    return model, tokenizer

def predict_sentiment(text, model, tokenizer, device):
    model.eval()
    encoding = tokenizer.encode_plus(
```

```
        text,
        add_special_tokens=True,
        max_length=128,
        return_token_type_ids=False,
        padding='max_length',
        truncation=True,
        return_attention_mask=True,
        return_tensors='pt',
    )

    input_ids = encoding['input_ids'].to(device)
    attention_mask = encoding['attention_mask'].to(device)

    with torch.no_grad():
        outputs = model(input_ids, attention_mask=attention_mask)
        pred = torch.argmax(outputs.logits, dim=1).cpu().numpy()[0]
        # Get confidence scores
        probs = torch.nn.functional.softmax(outputs.logits,
dim=1).cpu().numpy()[0]

    sentiment_map = {0: 'negative', 1: 'neutral', 2: 'positive'}
    return sentiment_map[pred], probs[pred]

# Main execution
if __name__ == "__main__":
    # Assuming df is your input dataframe
    print("Starting training process...")
    model, tokenizer = train_sentiment_model(df)

    # Example prediction
    example_text = "The geosolutions technology will leverage benefon
gps solutions"
    sentiment, confidence = predict_sentiment(example_text, model,
tokenizer,
                                              torch.device('cuda' if torch.cuda.is_available()
else 'cpu'))
    print(f"\nExample prediction for: '{example_text}'")
    print(f"Predicted sentiment: {sentiment} (confidence: {confidence:.2f})")
```

```
Evaluating...
Average training loss: 0.6564
Test Accuracy: 0.8400

Classification Report:
              precision    recall  f1-score   support

    Negative       0.77      0.95      0.85       333
     Neutral       0.88      0.70      0.78       333
    Positive       0.90      0.87      0.88       334

    accuracy                           0.84      1000
   macro avg       0.85      0.84      0.84      1000
weighted avg       0.85      0.84      0.84      1000
```

```
Evaluating...
Average training loss: 0.2803
Test Accuracy: 0.8570

Classification Report:
              precision    recall  f1-score   support

    Negative       0.79      0.97      0.87       333
     Neutral       0.90      0.69      0.78       333
    Positive       0.90      0.91      0.90       334

    accuracy                           0.86      1000
   macro avg       0.87      0.86      0.85      1000
weighted avg       0.87      0.86      0.85      1000
```

```
Example prediction for: 'The geosolutions technology will leverage benefon gps solutions'
Predicted sentiment: positive (confidence: 0.80)
CPU times: user 1min 59s, sys: 43.6 s, total: 2min 42s
Wall time: 2min 48s
```

Download Code + Data:

https://t.me/AIMLDeepThaught/557

Prepared By: Syed Afroz Ali