



PUSL3189 - Natural Language Processing

Individual Coursework

Name: B.W.M.O Wicramasinhe
Plymouth Index Number: 10899497
Degree Program: BSc (Hons) in Data Science

Table of Contents

1. Introduction to NLP and Data Collection	3
1.1 Introduction to NLP and Its Significance	4
1.2 Data Collection Methods	4
2. Text Preprocessing and Tokenization	5
2.1 Preprocessing Steps	5
2.2 Tokenization and stemming	7
2.3 N-Gram Creation	10
3. POS Tagging and Named Entity Recognition (NER)	12
3.1 Part-of-Speech Tagging Code and Results	12
3.2 Named Entity Recognition Code and Recognized Entities	14
4. Sentiment Analysis	15
4.1 Sentiment Analysis Method and Code	15
5. Topic Modeling	18
5.1 Topic Modeling Code and Technique	18
6. Stylometric Analysis and Visualization	20
6.1 Stylometric Analysis Techniques	22
6.2 Visualization	23
7. Document Clustering with Word2Vec	28
7.1 Text to Vector Conversion Code	28
7.2 K-Means Clustering and Results	29
7.3 Visualization of Clusters	30
8. Dependency Parsing and Advanced Structures	31
8.1 Dependency Parsing Code and Syntax Analysis	32
8.2 Parsed Sentence Structures and Visualizations	33
9. Insights and Real-World Application	34
9.1 Summary of Key NLP Insights	34
9.2 Real-World Applications of the Analysis	35
10. Advanced NLP Technique	37
10.1 Code Implementation of Advanced NLP Technique	37
10.2 Discussion of Results and Relevance	41

Introduction

Natural Language Processing is a technology that lets human beings talk to machines in their natural language. NLP can be achieved through translating the data into a form that the computer understands using smarter algorithms and enabling the machine for deep learning, machine learning, and artificial intelligence. Natural Language Processing employs linguistic, computer science and machine learning technologies to analysis human language. Sentiment Analysis, Machine Translation, Chatbot, Speech Recognition, Information Extraction are some uses of NLP. Also, in sectors like text mining, social media mining, and web mining, and including the study and implementation of methodologies, tools, and algorithms that help in comprehending natural language and their combinations. NLP is altering the way individuals and companies communicate with technology, from streamlining document summaries to enabling real-time language translation, encouraging innovation in several industries including health, finance, and education.

To explore the capabilities of Natural language processing techniques using python, this project is based on classifying the Facebook comments and posts under “Gender and Sexual Orientation” as positive or negative outcomes. Since people bear very different views and opinions when it comes to this topic, people post and comment on their feelings through social media. This results some user to support the individuals and some to criticize and hate other’s views. This project helps to reveal if there are campaigns carried out by groups of users or organizations to purposely offend the Facebook users by commenting on their posts.

Task 1: Introduction to NLP and Data Collection

The data collection was done by using a python library called “facebook-page-scraper”. This library scrapes details from the post and comment and saves it on an Excel file. The main columns that were used are “commentText” and “postText”.


commentId	commentText	commentAuthorName
Y29tbWVudDoyMDQ5MTExMjA4ODk4NzcyXzlwNDkxNDEzMDU1Nj0MjE=	Oh, no. Not our bloodlines! Whatever will we do? "reclines on fainting couch"	Jennifer LeBlanc Easterday
Y29tbWVudDoyMDQ5MTExMjA4ODk4NzcyXzlwNDkxNDEzMDU1Nj0MjE=	That's a lot of words for "no one will sleep with me"...	Niki Thornton
Y29tbWVudDoyMDQ5MTExMjA4ODk4NzcyXzlwNDkxNzgzODU1Mzg3MjE=	Just another way to shame	Cathy Bergstrom
Y29tbWVudDoyMDQ5MTExMjA4ODk4NzcyXzlwNDkxMjEyNzg4OTc3NjU=	Spoken like an old resentful man who has been bested in some way by a woman.	Sara Lamb
Y29tbWVudDoyMDQ5MTExMjA4ODk4NzcyXzlwNDkxMzgzNDgzNDYyMTU=	I don't know if any of my ancestors fought mountain lions, but I do know that my (p	ReVdr Sharon Wozencraft
Y29tbWVudDoyMDQ5MTExMjA4ODk4NzcyXzlwNDkxNjg4Mjg4NzYwMTA=	Some idiots will never understand and it is not even worth our time or effort to give a	Diane Wagner
Y29tbWVudDoyMDQ5MTExMjA4ODk4NzcyXzlwNDkxMDE5MTU1NTYzNjg=	The audacity, are we just walking wombs to them and that's it?!!	Tabs Razi
Y29tbWVudDoyMDQ5MTExMjA4ODk4NzcyXzlwNDkxODM1NzU1NDgzYMDI=	The fact I'm still attracted to men proves sexuality isn't a choice.	Rachel Kemper
Y29tbWVudDoyMDQ5MTExMjA4ODk4NzcyXzlwNDkxMjg0NzYyMjA5Nzk=	I have a lot to show for my life. 10 years higher education. I owned 2 businesses, wa	Patricia Ann Seward
Y29tbWVudDoyMDQ5MTExMjA4ODk4NzcyXzlwNDkxMzcwOTg4OTYxNzNm=	But how many of "mags men" have provided a better life for all their children. And t	Dana-Rae Dancy
Y29tbWVudDoyMDQ5MjgzNzYyMjE0ODUwXzlwNDkxNj00ODUwY0DA=	Look. I don't want to get banned for saying anything. I was claiming feminism since	John Jackson
Y29tbWVudDoyMDQ5MjgzNzYyMjE0ODUwXzlwNDkxOTA0MzlyMTQxODM=	Yes, I've experienced complete bewilderment from people as I've traveled solo thro	Greta Grigorian
Y29tbWVudDoyMDQ5MjgzNzYyMjE0ODUwXzlwNDkxOTk0NTU1NDY2MTQ=		Marilynn Luberecki Eldridge
Y29tbWVudDoyMDQ5MjgzNzYyMjE0ODUwXzlwNDkxMjY4MTg4NzcyMTE=	Have found this to be absolutely true.	Terri Kimball Hart
Y29tbWVudDoyMDQ5MjgzNzYyMjE0ODUwXzlwNDkxMDkxMjU1NDU2NDc=	I am so sorry for what my gender has done to you all.	Bryan Yule
Y29tbWVudDoyMDQ5MzlyNjYyMjEwOTYyXzlwNDkxNDM1MzlyMDg4NzNm=	Old lady here. We couldn't get enough states to pass the Equal Rights Amendment.	Marilynn Luberecki Eldridge
Y29tbWVudDoyMDQ3NzAyNTA5MDM5NjQxXzlwNDg1MTM5NTU2MjUxNjC1	I don't know why they'd bother. Popular vote doesn't matter.	Shelly Williams
Y29tbWVudDoyMDQ3NzAyNTA5MDM5NjQxXzlwNDg1Njg3MTlyODYzNTU=	While women's suffrage has made significant progress globally, there have been in	Laura Gaffney
Y29tbWVudDoyMDQ3NzAyNTA5MDM5NjQxXzlwNDgzNjEyMDg5NzM3NzI=	Margaret Atwood is a seer!	TJ Norris
Y29tbWVudDoyMDQ3NzAyNTA5MDM5NjQxXzlwNDgzOTkwMjU2NTY2NTc=	Absolutely disgusting creeps!!! We won't back down!!!	Gill Banks
Y29tbWVudDoyMDQ3NzAyNTA5MDM5NjQxXzlwNDgzOTY1MTU2NTczMDg=	What book is this from?	Julia Bailey
Y29tbWVudDoyMDQ4MjA5MDA1NjU2MjU5XzlwNDgzMTJmX0TYOTA5MDc=	This is every American's fault that considered him in the first place. He should have	Teresa Kenny
Y29tbWVudDoyMDQ4MjA5MDA1NjU2MjU5XzlwNDgzNDEyMzg5NzU3NjE=	WE WILL CONTINUE TO FIGHT	Linda Dupell
Y29tbWVudDoyMDQ4NjMyNTQ1NjEzMzA1XzlwNDkxODI5Nzg4ODE1OTU=	FUCK THAT	Andrea Gronau
Y29tbWVudDoyMDQ4NjMyNTQ1NjEzMzA1XzlwNDkxNzK4MTg4ODE5MTE=	lol, I have no problem telling ppl no as many times as they need to hear it. It actually	Andrea Gronau
Y29tbWVudDoyMDQ4NjMyNTQ1NjEzMzA1XzlwNDkxNzK4Nzg4ODE5MjU=	Can you imagine telling a man, "You look thirsty" and trying to manipulate him into	Andrea Gronau
Y29tbWVudDoyMDM4MzcyNjc5OTcyNjI1XzlwMzgzODI0NzIzMzE5Nzk=	This reminds me of when my mom asked me if I wanted a certain sweater or to do a	Nancy Gaines
Y29tbWVudDoyMDM4MzcyNjc5OTcyNjI1XzlwMzgzODI1MTI1MzY2MjEzMDY=	If you say "no thank you so much" and someone still insists... red flag	Nikki Quinn
Y29tbWVudDoyMDM4MzcyNjc5OTcyNjI1XzlwMzgzODI0NjYyMjEwOTcxOTk=	I would be the AH what part of the No wasn't clear Mr man	Carrol Weatherford
Y29tbWVudDoyMDM4MzcyNjc5OTcyNjI1XzlwMzgzODI0NjYyMjEwOTcxOTk=	No means no, and this is dangerous anyway, as it could be doctored. Always watc!	Marilynn Luberecki Eldridge
Y29tbWVudDoyMDM4MzcyNjc5OTcyNjI1XzlwMzgzODI0NjYyMjEwOTcxOTk=	I am trying to understand what happened to you that you feel "rude" for turning so	Amy Erlanger
Y29tbWVudDoyMDM4MzcyNjc5OTcyNjI1XzlwNDA1OTY3NTk3NTAyMTc=	You should never take an already poured drink from someone unless you trust them	Nancy Bernstein
Y29tbWVudDoyMDM4MzcyNjc5OTcyNjI1XzlwMzkwMjg5MDMyNDAzMzY=	Been there ...felt like that	Amber Roper
Y29tbWVudDoyMDM4MzcyNjc5OTcyNjI1XzlwNDQ2NTQzMjYwMTExMjE=	Oh my word, that's brilliant! ☺	DeeAnn Eubanks Pochedly
Y29tbWVudDoyMDM4MzcyNjc5OTcyNjI1XzlwNDc4NTMzODU2OTEyMjE=	Aa q a b	Sheryle Lyman
Y29tbWVudDoyMDM4MzcyNjc5OTcyNjI1XzlwNDQ1MTM4MzkyNTg1ODY=	Heads up, I'll be inviting Leslie to our BD lunch Jody and Sammy ☺	Claire Allen
Y29tbWVudDoyMDM0NzIzMzNzNzNiYyNTU1XzlwNDgzNDEyNTQ0NjYzNDk4MDTK=	Powerful	Marilynn Luberecki Eldridge

This column contains rows 2905 and 16 columns. There are many empty cells about the details of the user since some Facebook accounts were set to private mode. To have a clear output and increase the efficiency, a new data frame has created by only using “commentText” and “postText” columns. Since there are two columns containing the text, operations are done to both columns separately.

Task 2: Text Preprocessing and Tokenization

The corpus should be created by adding "comment_Text" and "post_Text" column. We need to remove stop words, any special characters or punctuation. The text corpus should be cleaned before using for analysis since it can affect the final output of the analysis. These preprocessing techniques are used.

1. Remove duplicate text from the columns.

```
 # Remove duplicates values of dataframe commentText and postText columns.  
  
df = df.drop_duplicates(subset=['commentText', 'postText'])
```

2. To remove stop words from the text, the “NLTK” library is imported.

```
[ ] # remove stopwords  
stopwords = set(stopwords.words('english'))  
  
def remove_stopwords(text):  
    no_stopword_text = [w for w in text.split() if not w in stopwords]  
    return " ".join(no_stopword_text)  
  
df['postText'] = df['postText'].astype(str).apply(lambda x: remove_stopwords(x))  
df['commentText'] = df['commentText'].astype(str).apply(lambda x: remove_stopwords(x))
```

3. To remove special characters and emojis “Re” library is used.

```
df['postText'] = df['postText'].str.replace("[^a-zA-Z# ]", " ", regex=True)  
df['commentText'] = df['commentText'].str.replace("[^a-zA-Z# ]", " ", regex=True)  
  
# Define a function to remove emojis and exclamation marks  
def emojisremove(text):  
    # Remove emojis  
    emoji_pattern = re.compile("[  
        u\"\\U0001F600-\\U0001F64F\" # emoticons  
        u\"\\U0001F300-\\U0001F5FF\" # symbols & pictographs  
        u\"\\U0001F680-\\U0001F6FF\" # transport & map symbols  
        u\"\\U0001F1E0-\\U0001F1FF\" # flags (iOS)  
        u\"\\U00002702-\\U000027B0\" # Dingbats  
        u\"\\U000024C2-\\U0001F251\" # Enclosed characters  
    ]+", flags=re.UNICODE)  
    text = emoji_pattern.sub(r"", text)  
  
    # Remove exclamation marks  
    text = text.replace("!", "")  
  
    return text
```

4. Cleaning the text

To further clean and give more meaning to the text, all text is converted to lowercase to ensure uniformity and consistency. It replaces common contractions like "what's" with their expanded forms such as "what is," and similarly handles contractions like "can't," "i'm," and "won't" by converting them into their full phrases. Special characters and unwanted symbols are removed using regular expressions, and excess whitespace is cleaned to ensure well-structured, uniform text. This function ensures that textual data is standardized, free from unnecessary punctuation or symbols, and ready for downstream NLP tasks.

```
def clean_text(text):
    text = text.lower()
    text = re.sub(r"what's", "what is ", text)
    text = re.sub(r"\'s", " ", text)
    text = re.sub(r'\ve', " have ", text)
    text = re.sub(r"can't", "can not ", text)
    text = re.sub(r"n't", " not ", text)
    text = re.sub(r"i'm", "i am ", text)
    text = re.sub(r'\re', " are ", text)
    text = re.sub(r'\d", " would ", text)
    text = re.sub(r'\ll", " will ", text)
    text = re.sub(r'\scuse", " excuse ", text)
    text = re.sub('\W', ' ', text)
    text = re.sub('\s+', ' ', text)
    text = text.strip(' ')
    return text

df['postText'] = df['postText'].astype(str).apply(lambda x: clean_text(x))
df['commentText'] = df['commentText'].astype(str).apply(lambda x: clean_text(x))
```

By this step it will enable the processing techniques to be more efficient and accurate since social media users use punctuation marks and short words. In the processing part, this will cause confusion and misleads when classifying the text.

5. Tokenization

This code uses “SpaCy” to tokenize text data in a DataFrame. It defines a function, `tokenize_with_spacy`, that processes text into tokens. The function is applied to the columns of the DataFrame, converting their content into lists of tokens.

```
import spacy

# Load the SpaCy language model
nlp = spacy.load("en_core_web_sm")

# Tokenize using SpaCy

def tokenize_with_spacy(text):
    if isinstance(text, list): # Check if text is a list
        text = " ".join(text) # Join list elements into a string
    doc = nlp(text) # Process the text with SpaCy
    tokens = [token.text for token in doc] # Extract tokens
    return tokens

# individual words considered as tokens
df['postText'] = df['postText'].apply(lambda sentence: tokenize_with_spacy(sentence))
df['commentText'] = df['commentText'].apply(lambda sentence: tokenize_with_spacy(sentence))
df.head()
```

6. Stemming

This code applies stemming to text data in a DataFrame using NLTK's.

‘PorterStemmer’ object is created to reduce words to their base form.

```
# stemming the words
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()

#Applying for the columns
df['stemmedpostText'] = df['postText'].apply(lambda sentence: [stemmer.stem(word) for word in sentence])
df['stemmedcommentText'] = df['commentText'].apply(lambda sentence: [stemmer.stem(word) for word in sentence])
df.head()
```

7. Lemmatization

Lemmatization is carried using 'spacy' library. Lemmatization is done to reduce words to their base or root form while preserving their contextual meaning. It helps standardize text, reduces word variations, and improves the efficiency of NLP tasks by focusing on core word meanings rather than inflected forms.

```
import spacy

# Lemmatization function
def lemmatize_with_spacy(tokens):
    if isinstance(tokens, list):
        tokens = " ".join(tokens)
    doc = nlp(tokens) # Process the text
    lemmatized_tokens = [token.lemma_ for token in doc]
    return lemmatized_tokens

# Apply stemming directly to the tokenized sentences
dfcleaned['lemmatizedpostText'] = dfcleaned['cleanedpostText'].astype(str).apply(lemmatize_with_spacy)
dfcleaned['lemmatizedcommentText'] = dfcleaned['cleanedcommentText'].astype(str).apply(lemmatize_with_spacy)
```

8. Extra step - checking lemmatization or stemming is best for this dataset.

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Function to calculate cosine similarity
def compare_similarity(lemmatized_texts, stemmed_texts):
    vectorizer = CountVectorizer(analyzer=lambda x: x) # Use word lists directly
    # Combine texts into a single list of sentences
    combined_text = lemmatized_texts + stemmed_texts
    vectors = vectorizer.fit_transform(combined_text).toarray()

    # Split vectors
    lemmatized_vectors = vectors[: len(lemmatized_texts)]
    stemmed_vectors = vectors[len(lemmatized_texts):]

    # Compute cosine similarity between the lemmatized and stemmed vectors
    similarity_scores = cosine_similarity(stemmed_vectors, lemmatized_vectors)

    return similarity_scores

# Calculate similarity between lemmatized and stemmed sentences
similarity_scores = compare_similarity(dfcleaned['lemmatizedpostText'].tolist(), dfcleaned['stemmedpostText'].tolist())

# Print similarity scores
for i, score in enumerate(similarity_scores):
    print(f"Sentence {i + 1}:")
    print(f"Original lemmatized sentence: {dfcleaned['lemmatizedpostText'][i]}")
    print(f"Stemmed sentence: {dfcleaned['stemmedpostText'][i]}")
    print(f"Similarity score: {score[0]}\n")
```


After computing the similarity scores, we can define a decision-making threshold to determine whether the similarity between the lemmatized and stemmed sentences is considered high or low.

```
Sentence 1:
Original lemmatized sentence: ['mic', 'join']
Stemmed sentence: ['mic', 'join']
Similarity score: 0.9999999999999998

Sentence 2:
Original lemmatized sentence: ['https', ' ', 'www', 'borowitzreport', 'com', 'p', 'kari', 'lake', 'call', 'it', 'totally', 'unfair', 'r', '3ju40n', 'utm', 'campaign', 'post', 'utm',
Stemmed sentence: ['http', ' ', 'www', 'borowitzreport', 'com', 'p', 'kari', 'lake', 'call', 'it', 'total', 'unfair', 'r', '3ju40n', 'utm', 'campaign', 'post', 'utm', 'medium', 'web']
Similarity score: 0.0

Sentence 3:
Original lemmatized sentence: ['Mmmm', ' ']
Stemmed sentence: ['mmmm', ' ']
Similarity score: 0.0
```

Since lemmatization considers the context and hence does an accurate transformation of the word, it is preferred. Unlike stemming, it keeps the sense of the word after conversion. The text is cleaner and more consistent which brings significant improvement in performance in Natural Language Processing.

9. combine words into a single sentence.

```
def combine_words(tokens):
    return " ".join(tokens)

# Apply the transformation using apply()
dfcleaned['sentencedlemmatizedpostText'] = dfcleaned['lemmatizedpostText'].apply(lambda x: combine_words(x))
dfcleaned['sentencedlemmatizedcommentText'] = dfcleaned['lemmatizedcommentText'].apply(lambda x: combine_words(x))

dfcleaned.head()
```

10. N-gram creation

N-grams are used in NLP to capture word sequences, preserving context for tasks like text generation, search engines, and spam detection. They improve understanding of patterns and relationships between words, enhancing analysis and model performance.

This code generates bigrams and trigrams for the `sentencedlemmatizedcommentText` and `sentencedlemmatizedpostText` columns in the dataframe. It tokenizes the text using NLTK and combines consecutive tokens into groups of two or three, storing them as lists in new columns. This helps capture contextual word relationships, which can improve text analysis tasks like classification or clustering.

```
# Function to perform n-gram
def get_n_grams(text, n):
    if isinstance(text, str):
        tokens = nltk.word_tokenize(text)
        n_grams = zip(*[tokens[i:] for i in range(n)])
        return [" ".join(ngram) for ngram in n_grams]
    else:
        return []

# Example usage: bigram (n=2)
def generate_ngrams(df, column_name, n):
    return df[column_name].apply(lambda x: get_n_grams(x, n))

# Generate bigrams for the 'stemmedcommentText' column
dfcleaned['bigrams_stemmedcommentText'] = generate_ngrams(dfcleaned, 'sentencedlemmatizedcommentText', 2)
dfcleaned['trigrams_stemmedcommentText'] = generate_ngrams(dfcleaned, 'sentencedlemmatizedcommentText', 3)

# Generate trigrams for the 'stemmedcommentText' column
dfcleaned['bigrams_stemmedpostText'] = generate_ngrams(dfcleaned, 'sentencedlemmatizedpostText', 2)
dfcleaned['trigrams_stemmedpostText'] = generate_ngrams(dfcleaned, 'sentencedlemmatizedpostText', 3)
```

The pattern between the text by reading two and three words at a time.

bigrams_stemmedcommentText	trigrams_stemmedcommentText	bigrams_stemmedpostText	trigrams_stemmedpostText
[oh no, no not, not bloodline, bloodline whate...]	[oh no not, no not bloodline, not bloodline wh...]	[mic join]	[]
[that s, s lot, lot word, word no, no one, one...]	[that s lot, s lot word, lot word no, word no ...]	[https www, www borowitzreport, borowitzreport...]	[https www borowitzreport, www borowitzreport ...]
[just another, another way, way shame]	[just another way, another way shame]	[]	[]
[speak like, like old, old resentful, resentfu...]	[speak like old, like old resentful, old resen...]	[Smokheads545 7251971, 7251971 need, need 3]	[Smokheads545 7251971 need, 7251971 need 3]
[I don, don t, t know, know ancestor, ancestor...]	[I don t, don t know, t know ancestor, know an...]	[Green Party, Party Tennessee, Tennessee the, ...]	[Green Party Tennessee, Party Tennessee the, T...]
...
[Palm Outdoor, Outdoor Australian, Australian ...]	[Palm Outdoor Australian, Outdoor Australian o...]	[I m, m 79, 79 look, look old]	[I m 79, m 79 look, 79 look old]
[Krix Australian, Australian make, make owned,...]	[Krix Australian make, Australian make owned, ...]	[feel like, like Cinderella, Cinderella want, ...]	[feel like Cinderella, like Cinderella want, C...]

Preprocessing steps such as tokenization, lemmatization, stop word removal, and text cleaning are necessary for preparing raw text data for analysis. These processes standardize the text by splitting it into coherent pieces, reducing words to their base forms, and deleting unneeded or repetitive material. This guarantees that the model concentrates on the essential semantic information, enhancing the efficiency and accuracy of downstream tasks like clustering, sentiment analysis, or topic modeling. Proper preprocessing minimizes noise, boosts feature quality, and helps models discover significant patterns from the text input.

Task 3: POS Tagging and Named Entity Recognition

1. POS Tagging

Part-of-Speech tagging is an NLP technique that attaches words with nouns, verbs, and adjectives according to grammatical rules of a sentence. It helps comprehend the sentence structure and assists in many tasks like parsing, sentiment analysis, translation, etc.

The most common POS is applied to “sentencedlemmatizedcommentText” column.

```
nlp = spacy.load("en_core_web_sm")

for text in dfcleaned['sentencedlemmatizedcommentText']:
    doc = nlp(text) # Create a SpaCy Doc object
    for token in doc:
        print(token.text, " | ", spacy.explain(token.pos_), " | ")
```

This code outputs the grammatical rules of the words in the text. It will provide an understanding of the text.

```
you | pronoun |
re | verb |
boil | verb |
meat | noun |
instead | adverb |
pressure | noun |
cook | verb |
it | pronoun |
| space |
if | subordinating conjunction |
large | adjective |
piece | noun |
well | adverb |
cut | verb |
```

Using SpaCy to perform Part-of-Speech tagging on text data in a DataFrame column It loads SpaCy's English model (en_core_web_sm) and processes each text entry to extract POS tags for all tokens. These tags are added to a list (all_pos_tags), which is then analyzed using Python's Counter to count the frequency of each POS tag.

```
import spacy
from collections import Counter

# Load English model
nlp = spacy.load('en_core_web_sm')

# Process each text in the Series individually
all_pos_tags = []
for text in dfcleaned['sentencedlemmatizedcommentText']:
    doc = nlp(text) # Create a SpaCy Doc object for each text
    all_pos_tags.extend([token.pos_ for token in doc]) # Extend the list with new POS tags

# Calculate POS counts from the combined list
pos_counts = Counter(all_pos_tags)

# Display the most common parts of speech
print("Most Common Parts of Speech:")
for pos, count in pos_counts.most_common():
    print(f"{pos}: {count}")
```

Finally, it prints the most common POS tags and their counts, showing the distribution of grammatical categories in the text.

```
Most Common Parts of Speech:
NOUN: 2047
VERB: 1131
SPACE: 1029
ADJ: 715
PRON: 620
PROPN: 600
ADV: 389
NUM: 230
AUX: 151
ADP: 129
DET: 115
SCONJ: 94
INTJ: 76
PART: 54
CCONJ: 33
```

2. Named Entity Recognition

NER is a process through which Natural Language Processing identifies and classifies key information in the text. It is useful in extraction of information, question answering, content categorization, etc.

Named Entity Recognition is applied to 'sentencedlemmatizedpostText' column.

```
# Apply nlp to each row of the 'sentencedlemmatizedpostText' column
for text in dfcleaned['sentencedlemmatizedpostText']:
    doc = nlp(text)
    for ent in doc.ents:
        print(ent, " | ", ent.label_, " | ", spacy.explain(ent.label_))
```

Using the spacy NLP library. For each text, it processes the text with the NLP model and extracts entities such as names, locations, dates, etc. It then prints each entity, its label, and an explanation of the label provided by spacy.explain(), helping to identify and understand the types of entities recognized in the text.

```
the Borowitz Report | ORG | Companies, agencies, institutions, etc.
US Senate | ORG | Companies, agencies, institutions, etc.
two | CARDINAL | Numerals that do not fall under another type
Kari | PERSON | People, including fictional
Monday | DATE | Absolute or relative dates or periods
Republican Party | ORG | Companies, agencies, institutions, etc.
Republican Party | ORG | Companies, agencies, institutions, etc.
Lake | FAC | Buildings, airports, highways, bridges, etc.
deni deni | PERSON | People, including fictional
QAnon | ORG | Companies, agencies, institutions, etc.
Ginni Thomas | PERSON | People, including fictional
one | CARDINAL | Numerals that do not fall under another type
Proud | PERSON | People, including fictional
Mmmm | PERSON | People, including fictional
7251971 | DATE | Absolute or relative dates or periods
3 | CARDINAL | Numerals that do not fall under another type
Green Party | ORG | Companies, agencies, institutions, etc.
Tennessee | GPE | Countries, cities, states
https | PERSON | People, including fictional
https | PERSON | People, including fictional
Green Party | ORG | Companies, agencies, institutions, etc.
Tennessee | GPE | Countries, cities, states
USA | GPE | Countries, cities, states
```

Task 4: Sentiment Analysis

Sentiment analysis in NLP discovers the underlying emotion in a piece of text and classifies it into positive, negative or neutral. Monitoring social media info helps businesses gain valuable insights into customer feedback, sentiment analysis, and industry trends.

The “analyze_sentiment” function uses the “TextBlob” library to classify text sentiment based on polarity. If the polarity is positive, it returns "Positive"; if zero, "Neutral"; and if negative, "Negative." This helps quickly categorize text sentiment for applications like feedback or social media analysis.

```
# Define the analyze_sentiment function using TextBlob
def analyze_sentiment(text):
    analysis = TextBlob(text)
    if analysis.sentiment.polarity > 0:
        return 'Positive'
    elif analysis.sentiment.polarity == 0:
        return 'Neutral'
    else:
        return 'Negative'
```

The code applies sentiment analysis to the lemmatized post text in the DataFrame by using the analyze_sentiment function. It then calculates the distribution of sentiment values using “value_counts()” and prints the sentiment counts.

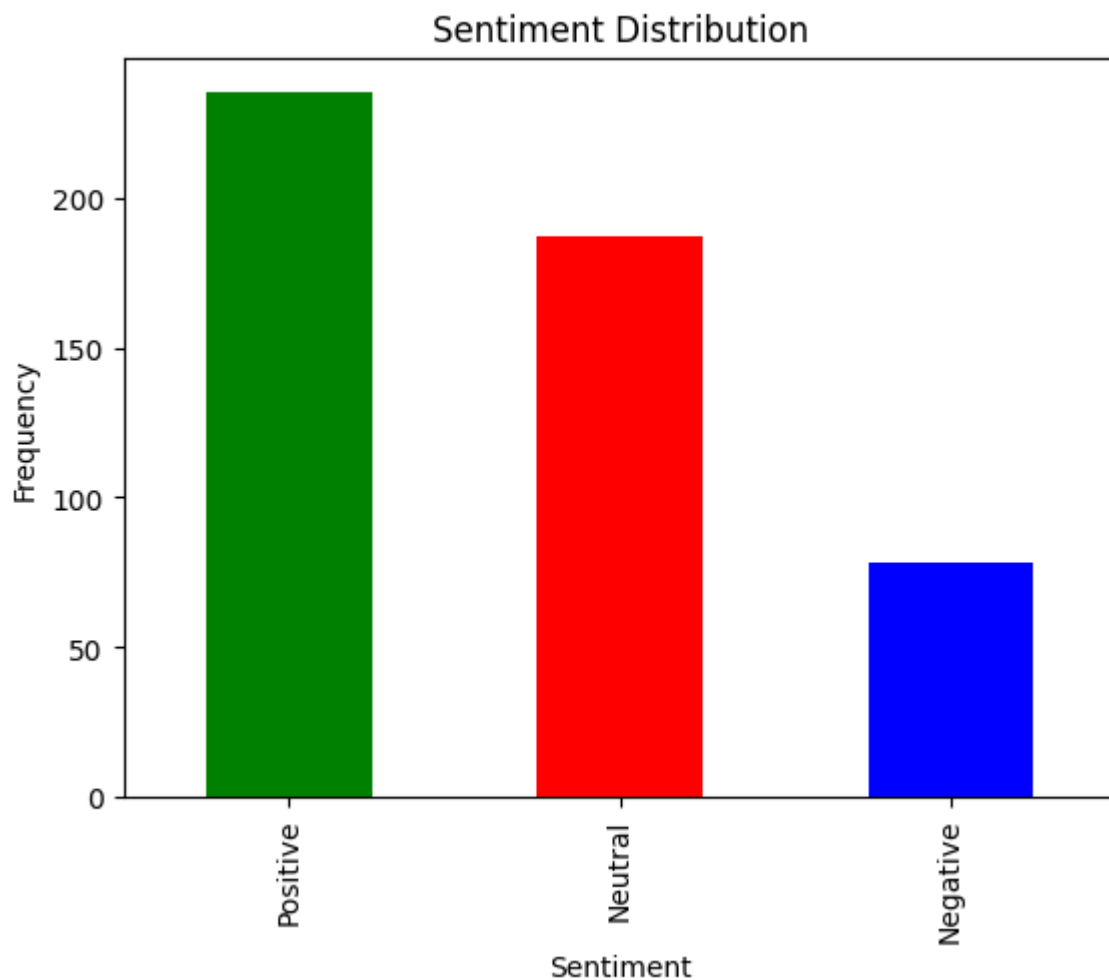
```
# Apply sentiment analysis
dfcleaned['SentimentofpostText'] = dfcleaned['sentencedlemmatizedpostText'].apply(analyze_sentiment)

# Display sentiment counts
# Changed from df['Sentiment'] to df['SentimentofpostText']
sentiment_counts = dfcleaned['SentimentofpostText'].value_counts()
print("Sentiment Distribution:")
print(sentiment_counts)

# Plot sentiment distribution
sentiment_counts.plot(kind='bar', color=['green', 'red', 'blue'])
plt.title('Sentiment Distribution')
plt.xlabel('Sentiment')
plt.ylabel('Frequency')
plt.show()
```

Finally, it visualizes the sentiment distribution using a bar chart, where each sentiment (positive, negative, neutral) is represented by a different color (green, red, blue), and the chart shows the frequency of each sentiment type.

```
Sentiment Distribution:  
SentimentofpostText  
Positive      235  
Neutral       187  
Negative       78  
Name: count, dtype: int64
```



By the output, a decision can be made about the text as positive, negative and neutral. This classification helps in protecting the guidelines in social media community.

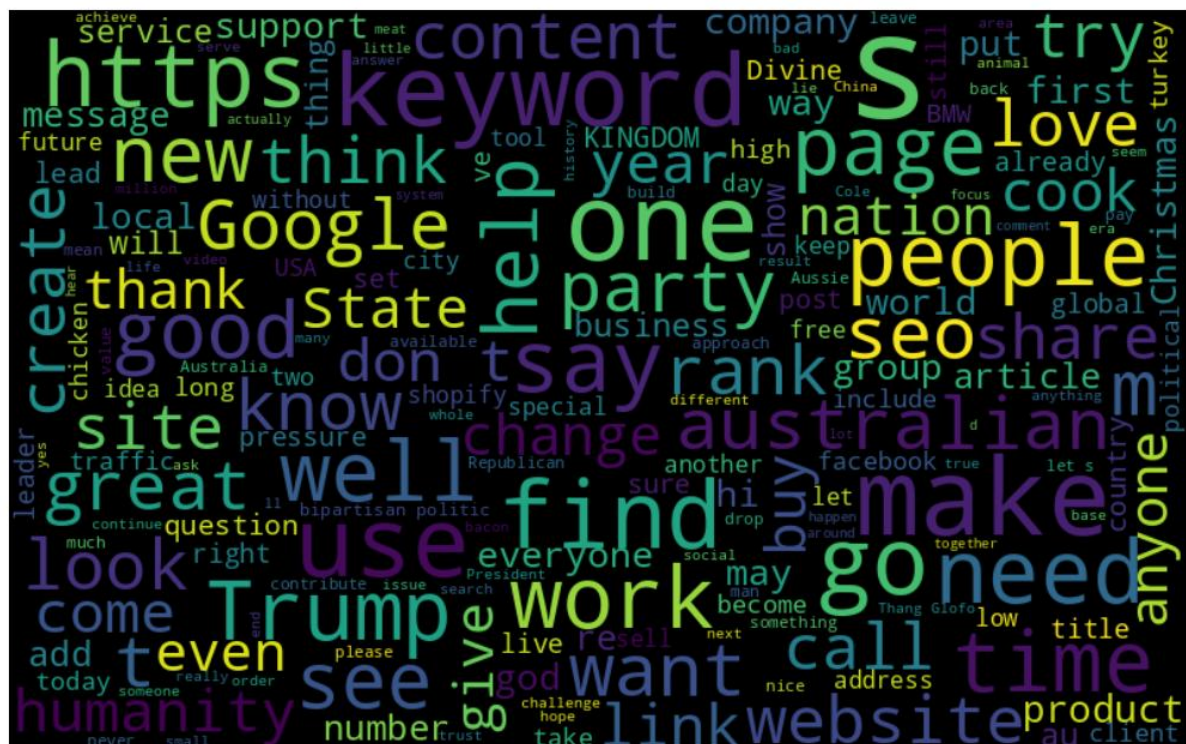
This code generates a word cloud to visualize the most frequent words in the “|sentencedlemmatizedpostText” column. Using the “WordCloud” library to create a word cloud, where word size indicates frequency.

```
# visualize the frequent words
all_words = " ".join([sentence for sentence in dfcleaned['sentencedlemmatizedpostText']])

from wordcloud import WordCloud
wordcloud = WordCloud(width=800, height=500, random_state=42, max_font_size=100).generate(all_words)

# plot the graph
plt.figure(figsize=(15,8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

Finally, it displays the word cloud using Matplotlib.



The word cloud reveals a diverse range of topics and themes. Common words like "thank," "good," "great," and "love" suggest a positive and appreciative tone. There's a focus on communication with terms like "message," "say," "talk," and "tell." "Work," "business," and "company" hint at professional contexts. The appearance of "Trump" and "Republican" suggests political discussions. Overall, the word cloud reflects a mix of personal, professional, and political conversations.

Task 5: Topic Modeling

Topic modeling in NLP is employed to identify the fundamental themes or subjects within a corpus of texts. It facilitates the identification of patterns and the categorization of linked documents. Methods such as Latent Dirichlet Allocation are frequently employed for this objective. Topic modeling is beneficial for applications such content categorization, document summarization, and recommendation systems, as comprehending the principal subjects of text enhances organization and insights.

1. Latent Dirichlet Allocation

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
import pandas as pd

vectorizer = CountVectorizer(max_df=0.9, min_df=2, stop_words='english') # Adjust thresholds
dtm = vectorizer.fit_transform(df1['sentencedlemmatizedcommentText'])

# Apply LDA
lda = LatentDirichletAllocation(n_components=2, random_state=42) # Adjust n_components (number of topics)
lda.fit(dtm)

# Extract topics and keywords
def display_topics(model, feature_names, no_top_words):
    topics = {}
    for topic_idx, topic in enumerate(model.components_):
        keywords = [feature_names[i] for i in topic.argsort()[::-no_top_words - 1:-1]]
        topics[f"Topic {topic_idx+1}"] = keywords
    return topics

no_top_words = 5
keywords = vectorizer.get_feature_names_out()
topics = display_topics(lda, keywords, no_top_words)

# Assign a topic to each sentence
topic_assignments = lda.transform(dtm) # Get topic probabilities for each sentence
df1['Assigned_Topic'] = topic_assignments.argmax(axis=1) + 1 # Assign the topic with the highest probability # Changed df to df1

# Map assigned topics to keywords for interpretation
topic_keywords = {idx + 1: ', '.join(words) for idx, words in enumerate(topics.values())}
df1['commentTopic_Keywords'] = df1['Assigned_Topic'].map(topic_keywords) # Changed df to df1
```

Above code performs topic modeling on a text dataset using Latent Dirichlet Allocation in scikit-learn. First, it preprocesses the text using “CountVectorizer” to create a document-term matrix, adjusting the frequency thresholds.

Then, it applies LDA to extract two topics from the text. The “display_topics” function extracts and displays the top 5 keywords for each topic.

Finally, a new column is created to map each assigned topic to its corresponding keywords for interpretation.

	sentencedlemmatizedcommentText	commentTopic_Keywords
0	oh no not bloodline whatever do recl...	make, use, pressure, pot, cook
1	that s lot word no one sleep I	make, use, pressure, pot, cook
2	just another way shame	make, use, pressure, pot, cook
3	speak like old resentful man best way woman	woman, love, man, say, people
4	I don t know ancestor fight mountain lion I ...	woman, love, man, say, people
...
495	Palm Outdoor Australian own make caravan	make, use, pressure, pot, cook
496	Krix Australian make owned speaker manufacture...	make, use, pressure, pot, cook
497	crochet plant make lovely desk ornament I al...	make, use, pressure, pot, cook
498	t s Creative Emporium	make, use, pressure, pot, cook
499	cool loose	make, use, pressure, pot, cook

Topic Modeling Analysis

- Topic 1: "Making/Crafting": Focuses on practical skills linked to making, crafting, and using tools or materials.
- Topic 2: "Relationships/Emotions": Focuses on human connections, emotions, and social interactions.

Topic modeling is commonly used in NLP to identify latent theme structures within massive collections of text. By automatically grouping words and phrases that commonly appear together, it enables the discovery of subjects that can summarize vast volumes of text data. In the analysis of customer reviews, topic modeling helps elucidate persistent topics such as product quality, delivery service, or customer assistance. This allows firms to acquire insights into customer sentiment and feedback without physically reviewing every review.

Task 6: Stylometric Analysis and Visualization

Stylometric analysis in NLP refers to the study and analysis of writing style patterns in text data, generally to identify authorship, detect plagiarism, or explain distinctive stylistic elements of a piece of writing. By assessing factors such as sentence length, word choice, punctuation usage, syntax, and frequency of certain terms, stylometry provides insights into the qualities of a text that can differentiate writers or show writing trends. Stylometric analysis is widely employed in authorship attribution tasks.

1. 50 most frequent words.

Below method does stylometric analysis on a text dataset by identifying the 50 most frequent words using CountVectorizer from scikit-learn, building a feature matrix from the lemmatized text. It then visualizes these frequent terms in a word cloud using the “WordCloud” library, displaying a graphical representation where larger words indicate higher frequency. The word cloud is displayed with defined dimensions and parameters, delivering a clear view of the most prevalent words in the dataset.

```
# Feature Extraction (Stylometric Features)
vectorizer = CountVectorizer(analyzer='word', stop_words='english', max_features=50) # Extract up to 50 most frequent words
X = vectorizer.fit_transform(df1['sentencedlemmatizedcommentText']).toarray() # Convert text to feature matrix

# Display extracted features
feature_names = vectorizer.get_feature_names_out()
print("Extracted Features (Top 50 words):")
print(feature_names)

# Check the shape of the resulting matrix
print(f"\nFeature matrix shape: {X.shape}")

# visualize the frequent words
all_words = " ".join([sentence for sentence in feature_names])

from wordcloud import WordCloud
wordcloud = WordCloud(width=800, height=500, random_state=42, max_font_size=100).generate(all_words)

# plot the graph
plt.figure(figsize=(15,8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

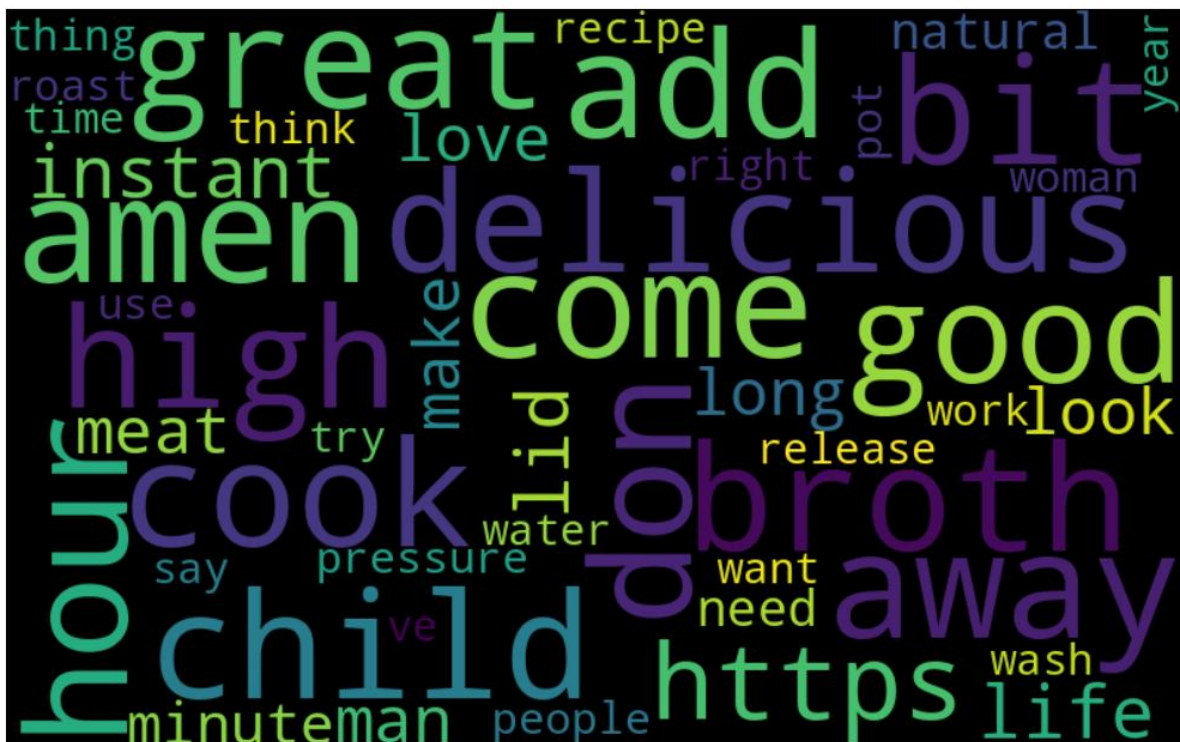
This outputs the up to 50 most frequent words in the text of the data frame.

```

Extracted Features (Top 50 words):
['add' 'amen' 'away' 'bit' 'broth' 'child' 'com' 'come' 'cook' 'delicious'
'don' 'good' 'great' 'high' 'hour' 'https' 'instant' 'lid' 'life' 'like'
'long' 'look' 'love' 'make' 'man' 'meat' 'minute' 'natural' 'need'
'people' 'pot' 'pressure' 'recipe' 'release' 'right' 'roast' 'say'
'thing' 'think' 'time' 'try' 'use' 've' 'want' 'wash' 'water' 'woman'
'work' 'www' 'year']

```

The word cloud with most frequent words.



The word cloud appears to be focused on cooking and food. Prominent words like "cook," "recipe," "delicious," and "broth" suggest a culinary theme. Terms like "meat" and "roast" indicate a focus on savory dishes. The presence of "instant" and "pressure" might suggest the use of quick cooking methods. Overall, the word cloud seems to depict a conversation or text related to cooking and food preparation.

2. Principal Component Analysis

Principal Component Analysis to lower the dimensionality of the feature matrix X. By setting `n_components=2`, PCA converts the original feature space into two principal components that capture the most important variance in the data. This reduced representation can then be utilized for displaying the underlying structure of the data, helping to identify patterns.

```
# Principal Component Analysis (PCA)
pca = PCA(n_components=2) # Reduce to 2 dimensions for visualization
X_pca = pca.fit_transform(X)
```

3. K-Means clustering

This code applies K-Means clustering to group the text data into 3 clusters based on the extracted features in matrix X. By setting `n_clusters=3`, the algorithm divides the data into three distinct clusters based on their similarity. The `fit_predict` method assigns each text sample to a cluster, and the resulting cluster labels are stored in a new column, `postCluster`, in the dataframe. This allows the data to be categorized into groups.

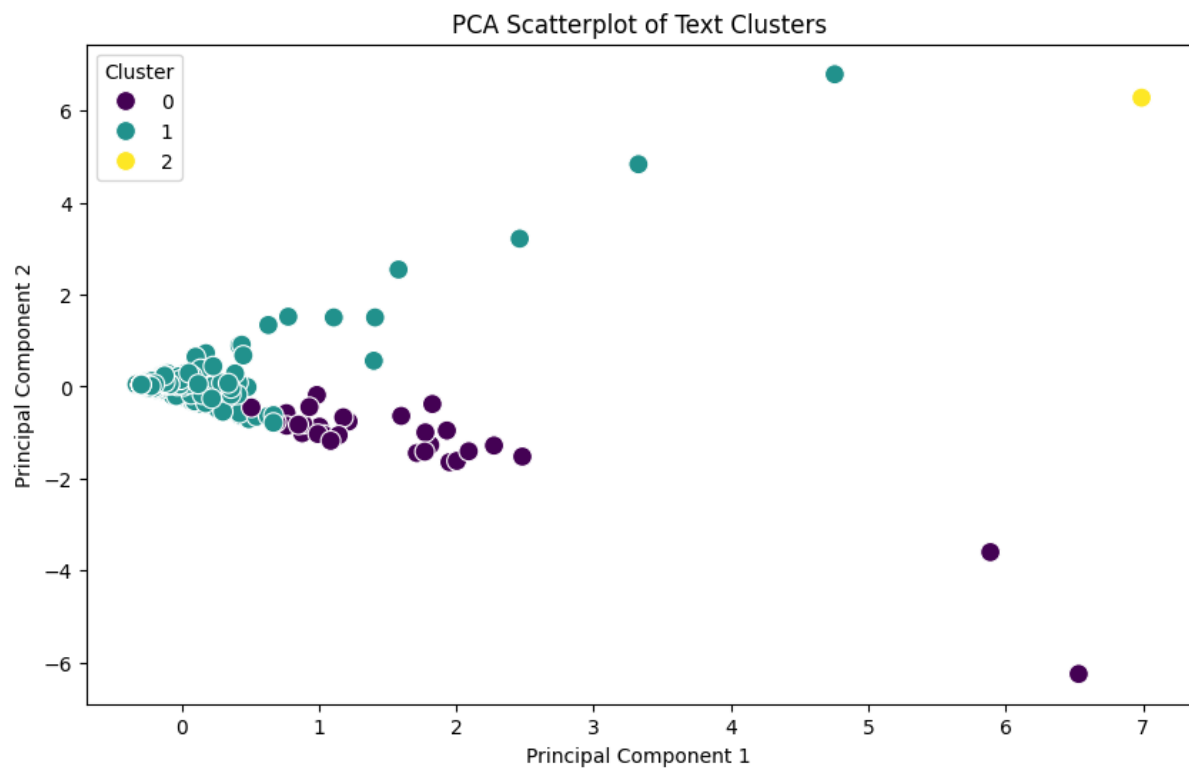
```
#K-Means Clustering
kmeans = KMeans(n_clusters=3, random_state=42) # Adjust number of clusters
df1['postCluster'] = kmeans.fit_predict(X)
```

4. PCA Scatterplot creation

This code generates a scatterplot to visualize the 2D representation of the clustered text data after performing PCA. It uses the first and second principal components as the x and y axes, respectively.

```
# PCA Scatterplot
plt.figure(figsize=(10, 6))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=df1['commentCluster'], palette='viridis', s=100)
plt.title('PCA Scatterplot of Text Clusters')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Cluster')
plt.show()
```

The plot is customized with a title, axis labels, and a legend to provide a clear visual representation of how the text samples are distributed across the clusters in the reduced 2D space.



Text Data Analysis Overview

- Grouping text data into three categories based on semantic or topical similarities.
- Data points are distributed across the plot, with some clusters appearing more compact.
- PCA scatterplot allows visualization of text data post -dimensionality reduction.
- separate clusters reflect semantic or thematic content.
- Separation between clusters suggests differences in content, with higher separation indicating more separate themes and lower separation indicating more overlap.

5. Dendrogram creation

In NLP, dendrograms are used for hierarchical clustering to categorize text data based on similarity. They help show how documents or words are related, making them valuable for tasks like document categorization, topic modeling, and sentiment analysis. By presenting groups of similar texts, dendrograms highlight patterns and linkages within vast text databases.

Below code builds a dendrogram for hierarchical grouping of the text data based on cosine similarity. First, it calculates the cosine similarity matrix between the text samples in the feature matrix 'X', and then converts it to a distance matrix. The linkage function is used to achieve hierarchical clustering using the Ward technique on the distance matrix.

```
# Dendrogram for Hierarchical Clustering
from sklearn.metrics.pairwise import cosine_similarity # Import cosine_similarity

# Calculate cosine similarity matrix (distance matrix)
distances = 1 - cosine_similarity(X) # Convert similarity to distance

# Perform linkage on the distance matrix
linked = linkage(distances, method='ward') # Hierarchical clustering

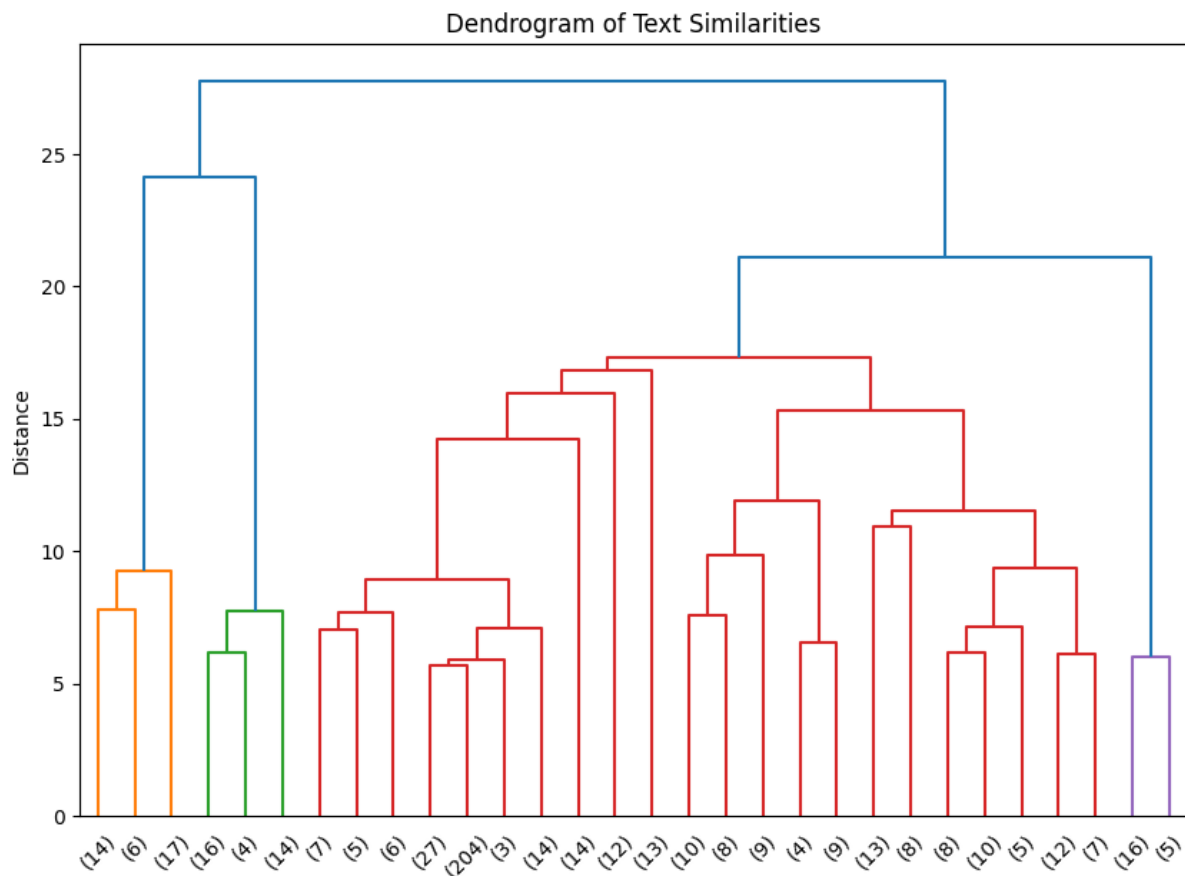
plt.figure(figsize=(10, 7))

# Use truncate_mode to limit the dendrogram size
dendrogram(linked,
            labels=df1['sentencedlemmatizedcommentText'].values,
            orientation='top',
            distance_sort='descending',
            show_leaf_counts=True,
            truncate_mode='lastp', # Show only the last 'p' merged clusters
            p=30 # Display 30
            )

plt.title('Dendrogram of Text Similarities')

plt.ylabel('Distance')
plt.show()
```


The `truncate_mode='lastp'` option limits the dendrogram to the last 30 merged clusters for clearer visualization, with the distance between clusters shown on the y-axis.



Dendrogram Analysis

- Visualizes the clustering of text texts into clusters based on similarity.
- Shows how data points are organized into clusters based on similarity.
- Starts with individual data points and eventually merges them into larger clusters.
- Clusters created can be regarded as groups of text documents that are more similar.
- Distinct Clusters: Indicates various groups based on similarities.
- Some clusters are larger than others, signifying closer relatedness.
- Data points far from main clusters could reflect fundamentally distinct texts.

6. Using 'TfidfVectorizer' module to extract top 50 Important Words in the text.

The TfidfVectorizer module in scikit-learn is used to turn text data into a matrix of TF-IDF features. It helps discover the most significant words by examining both the frequency of terms within a document and their inverse frequency across all documents.

This method utilizes the TfidfVectorizer from scikit-learn to extract the top 50 key words from the lemmatized text in the `sentencelemmatizedpostText` column. The TfidfVectorizer turns the text into a TF-IDF feature matrix, where each word is weighted by its term frequency in the document and its inverse frequency across all documents.

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize the TF-IDF Vectorizer (same as CountVectorizer but with TF-IDF)
vectorizer2 = TfidfVectorizer(stop_words='english', max_features=50) # Top 50 most important words based on TF-IDF
X_tfidf = vectorizer2.fit_transform(df1['sentencelemmatizedpostText']).toarray() # Convert text to TF-IDF feature matrix

# Retrieve the feature names (words)
top_words = vectorizer2.get_feature_names_out()

# Display the most important words
print("Top 50 Important Words (Based on TF-IDF) from LemmatizedHeadline:")
print(top_words)

# Display the shape of the resulting matrix
print(f"\nTF-IDF Feature Matrix Shape: {X_tfidf.shape}")

# visualize the frequent words
all_words = " ".join([sentence for sentence in feature_names])

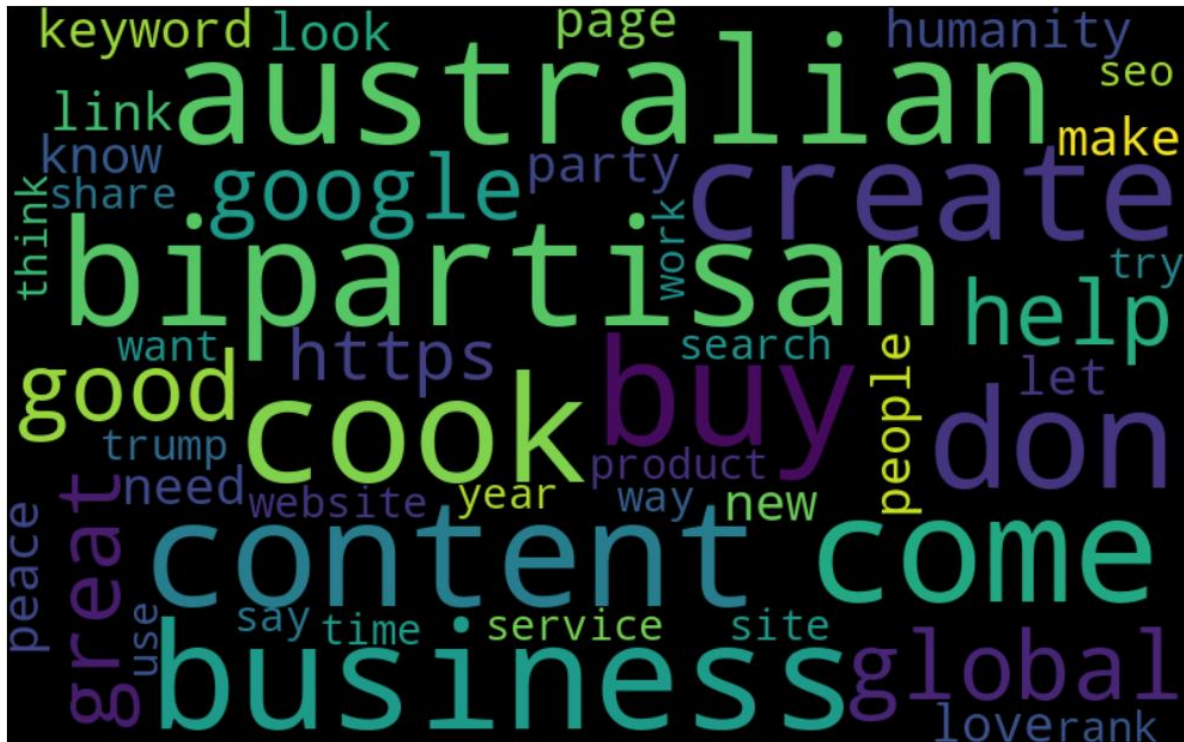
from wordcloud import WordCloud
wordcloud = WordCloud(width=800, height=500, random_state=42, max_font_size=100).generate(all_words)

# plot the graph
plt.figure(figsize=(15,8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

The code then prints these top words and the shape of the resulting TF-IDF matrix.

```
Top 50 Important Words (Based on TF-IDF) from LemmatizedHeadline:
['australian' 'bipartisan' 'business' 'buy' 'com' 'come' 'content' 'cook'
 'create' 'don' 'global' 'good' 'google' 'great' 'help' 'https' 'humanity'
 'keyword' 'know' 'let' 'like' 'link' 'look' 'love' 'make' 'need' 'new'
 'page' 'party' 'peace' 'people' 'product' 'rank' 'say' 'search' 'seo'
 'service' 'share' 'site' 'think' 'time' 'trump' 'try' 'use' 'want' 'way'
 'website' 'work' 'www' 'year']
```

Finally, it creates a word cloud to visualize the most frequent words, displaying them with varying sizes based on their importance in the dataset.



Word Cloud Analysis

- Prominent terms: "Australian," "Google," "business," "create," "cook," "buy," "help," and "search" suggest focus on business, online activities, and potentially food-related topics.
- Word size in the cloud is proportional to its frequency in the text, suggesting "Australian" and "Google" are more frequently mentioned.
- Different colors add visual interest but lack specific meaning or categorization.
- Based on prominent terms, the word cloud likely represents a collection of text related to business, online activities, cooking or food, and Australian context.

Task 7: Document Clustering with Word2Vec

Document clustering with Word2Vec is a technique in NLP used to group comparable documents based on the semantic meaning of words. Word2Vec, a neural network model, turns words into dense vector representations that capture their contextual links. This strategy is beneficial for organizing big text corpora, enhancing search results, and recommending comparable documents in applications like news aggregation or content-based filtering.

1. Developing Word2Vec model.

This code uses the Word2Vec model from the “gensim” library to generate word embeddings for text data in the column, which contains tokenized sentences. The model is trained with a vector size of 50, a context window of 5, and a minimum word count of 1. It then defines a helper function “get_average_word_vector”, which generates document vectors by averaging the embeddings of the words in each document, ensuring that only words present in the Word2Vec model are included.

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import matplotlib.pyplot as plt
import seaborn as sns
import re

# Set the visualization style
sns.set(style="whitegrid")

# Train Word2Vec model
# Train Word2Vec on tokenized sentences
word2vec_model = Word2Vec(sentences=df1['sentencedlemmatizedpostText'], vector_size=50, window=5, min_count=1, workers=4)

# Helper Function: Create document vectors by averaging word embeddings
def get_average_word_vector(tokens, model):
    # Create vector by averaging word embeddings
    valid_tokens = [token for token in tokens if token in model.wv]
    if not valid_tokens: # Handle empty token cases
        return np.zeros(50)
    return np.mean([model.wv[token] for token in valid_tokens], axis=0)

df1['doc_vector'] = df1['sentencedlemmatizedpostText'].apply(lambda x: get_average_word_vector(x, word2vec_model))
```

2. K-Means clustering

This code does document clustering on the text data using Word2Vec embeddings, K-Means clustering, and t-SNE for visualization. First, it turns the document vectors into a feature matrix. K-Means clustering with 3 clusters to categorize the documents based on their semantic similarity, storing the cluster assignments in a new column. To visualize the clusters, it decreases the dimensionality of the document vectors using t-SNE.

```
# Convert document vectors into a feature matrix
doc_vectors = np.array(list(df1['doc_vector']))

# Apply K-Means Clustering
n_clusters = 3 # Set number of clusters
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
df1['cluster'] = kmeans.fit_predict(doc_vectors)

# Dimensionality Reduction with t-SNE for Visualization
tsne = TSNE(n_components=2, random_state=42)
reduced_features = tsne.fit_transform(doc_vectors)

# Visualize Clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(x=reduced_features[:, 0], y=reduced_features[:, 1], hue=df1['cluster'], palette="viridis", s=100)
plt.title("Document Clustering with Word2Vec + K-Means + t-SNE")
plt.xlabel("t-SNE Component 1")
plt.ylabel("t-SNE Component 2")
plt.legend(title="Cluster")
plt.show()

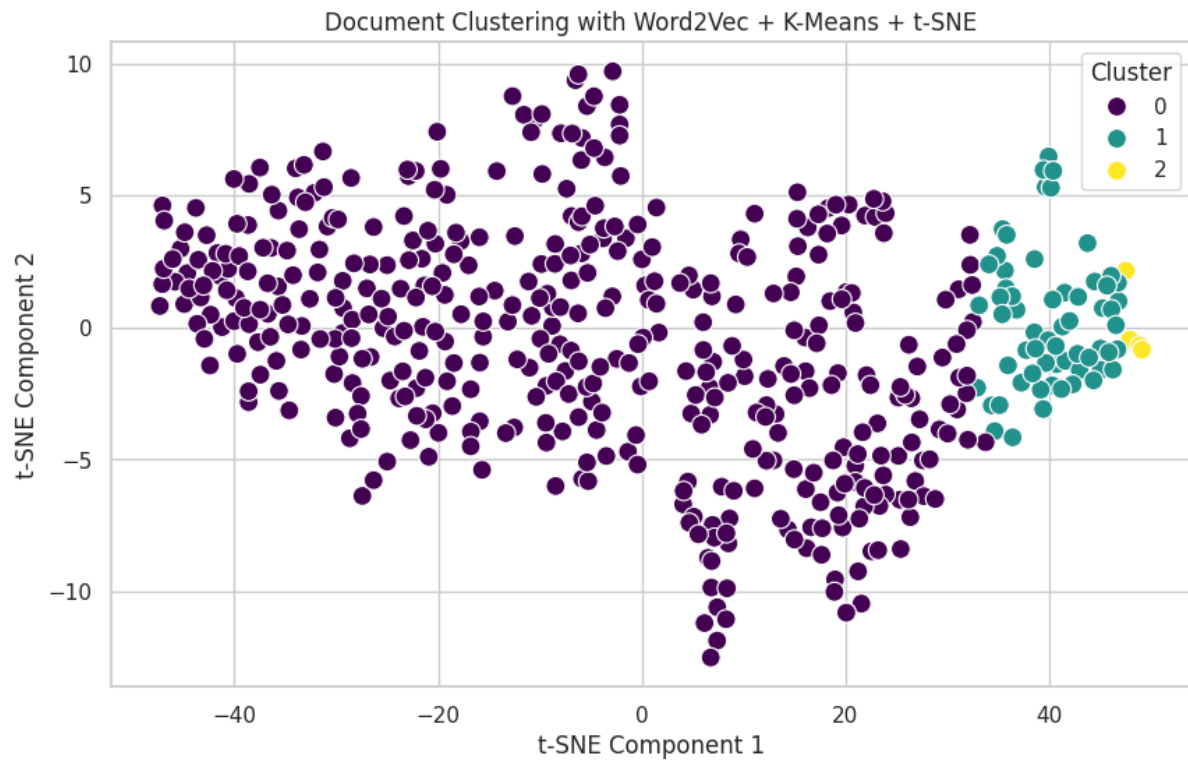
# Display the clustered documents
print("\nClustered Documents:")
for cluster_id in range(n_clusters):
    print(f"\nCluster {cluster_id}:")
    # Assuming 'sentencedlemmatizedpostText' contains the text data you want to display
    cluster_texts = df1[df1['cluster'] == cluster_id]['sentencedlemmatizedpostText'].tolist() # Changed 'text_samples' to 'sentencedlemmatizedpostText'
    for text in cluster_texts:
        print(f" - {text}")
```

Finally, the code prints the documents in each cluster, showing how the text samples are grouped by similarity.

```
Clustered Documents:

Cluster 0:
- mic join
- https www borowitzreport com p kari lake call it totally unfair r 3ju40n utm campaign post utm medium web PHOENIX the Borowitz Report
- Green Party Tennessee the party people planet peace https www facebook com greenpartyoftennessee
- end violence girl woman worldwide
- father matter
- his bid table immediately he vet properly qualified operate Town Hempstead Animal shelter he find guiltly animal abuse
- fantastic footage thank share
- joyous German festival
- million good hearted people around world continue support contribute Gofundme page keep Peanuts Freedom Animal Sanctuary go Peanut animal
- many kind compassionate people world continue hear terrible story invasive action DEC agent home seizure kill little animal companie
- let s raise Awareness by all mean necessary
- a truly lose golden era
- South Africans undergo spiritual warfare battle selfish desire fight system administration grossly violate will the People total disregard
- the Truth need Entertained Defended exist lie Falsehood do therefore use double edged Sword Truth destroy chain Oppression keep enslave
- follow like sand direct message claim Christmas present
- Elon Musk s groundbreaking innovation transform industry reshape future from Tesla s electric car revolutionize transportation SpaceX s
- what actually come mind see I sincere
- resource finite demand infinite bad lie history
- I Christmas gift text do
- I m happy lot devoted fan don t wish happy new month
- greeting as certify crypto trader I make rich forex trading mining crypto within short period time know work give message interested
```

A scatter plot is generated to display the clustered documents, where each point is colored based on its assigned cluster.



Document Clustering Analysis

- Clusters are well-separated, indicating more similarity between documents.
- Data points are spread across the plot, with some clusters appearing more compact.
- Distinct clusters indicate distinct categories based on semantic content.
- Examine cluster characteristics to identify key topics or themes.
- Investigate outliers to identify unique or anomalous documents.
- Compare results with other clustering methods for validation and different perspectives.

Task 8: Dependency Parsing and Advanced Structures

Dependency parsing is a key approach in NLP used to evaluate the grammatical structure of a phrase by finding the relationships between words. In this strategy, each word in the phrase is related to another word using direct edges, generating a tree structure. The fundamental purpose of dependency parsing is to discern which words are the syntactic head of others and how words depend on one another. This is essential in jobs like machine translation, question answering, and information extraction, where knowing the syntactic links between words is crucial.

Advanced structures, such as syntactic trees and dependency graphs, enhance the depth of NLP analysis by offering a more thorough representation of phrase structure. These structures allow for the extraction of fine-grained syntactic and semantic linkages, which are vital for complicated tasks like sentiment analysis, document classification, and relation extraction. Additionally, dependency parsing helps capture details like subject-verb-object relationships, helping models better understand phrase meaning, detect ambiguities, and make more accurate predictions in NLP applications.

1. Performing Dependency parsing

This code uses the “spaCy” library to perform dependency parsing on text data in the `df1['sentencedlemmatizedcommentText']` column, which contains tokenized sentences. The function “`parse_sentence`” processes each sentence by applying the “spaCy” model to parse the sentence, extracting the token's text, its syntactic dependency, its syntactic head, and its POS tag. The results are stored in a Pandas Data Frame for each sentence.

```

# Load English model
nlp = spacy.load("en_core_web_sm")

#Perform Dependency Parsing

def parse_sentence(sentence):
    # Changed from df2['sentencedlemmatizedpostText'] to sentence
    doc = nlp(sentence)
    parsed_data = []
    for token in doc:
        parsed_data.append({
            "Token": token.text,
            "Dependency": token.dep_,
            "Head": token.head.text,
            "POS": token.pos_
        })
    return pd.DataFrame(parsed_data)

# Analyze the first few sentences for dependency parsing
print("Dependency Parsing for Sample Sentences:")
for idx, sentence in enumerate(df1['sentencedlemmatizedcommentText'][:3]): # Limit to first 3 sentences
    print(f"\nSentence {idx + 1}: {sentence}")
    print(parse_sentence(sentence))

# Visualize Dependency Parsing
print("\nGenerating Dependency Visualizations...")
for idx, sentence in enumerate(df1['sentencedlemmatizedcommentText'][:3]): # Limit to first 3 sentences
    doc = nlp(sentence)
    print(f"\nSentence {idx + 1}: {sentence}")
    displacy.render(doc, style="dep", jupyter=True, options={"distance": 90})

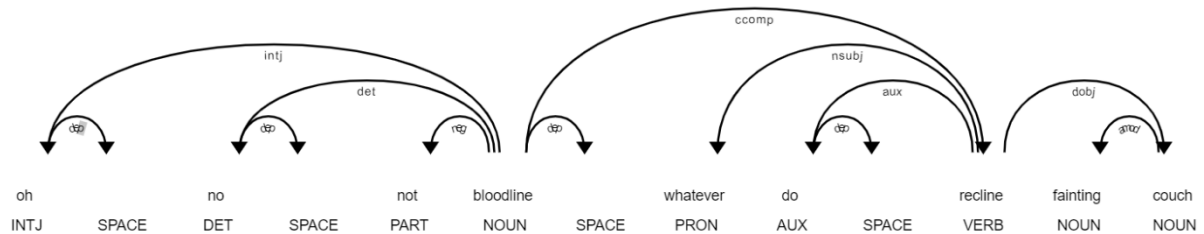
```

The first three sentences are analyzed, with their parsed dependency data printed out.

Dependency Parsing for Sample Sentences:

	Sentence 1:	oh	no	not	bloodline	whatever	do	recline	fainting	couch
	Token	Dependency		Head	POS					
0	oh	intj	bloodline	INTJ						
1		dep	oh	SPACE						
2	no	det	bloodline	DET						
3		dep	no	SPACE						
4	not	neg	bloodline	PART						
5	bloodline	ROOT	bloodline	NOUN						
6		dep	bloodline	SPACE						
7	whatever	nsubj	recline	PRON						
8	do	aux	recline	AUX						
9		dep	do	SPACE						
10	recline	ccomp	bloodline	VERB						
11	fainting	amod	couch	NOUN						
12	couch	dobj	recline	NOUN						

Additionally, the code visualizes the dependency trees of these sentences using “spaCy's” displacy module, rendering the relationships between words in a graphical format for easier interpretation.



Dependency Parse Analysis of Sentence

- Arcs are drawn between words to indicate their grammatical links. The label on each arc specifies the type of dependency.
- The diagram displays the hierarchical structure of the sentence, demonstrating which words depend on which others. For instance, "bloodline" is the direct object of "do," while "do" is the auxiliary verb of the sentence.
- The use of "oh no" conveys a strong emotional response.
- The phrase "not bloodline" denotes a poor or unexpected consequence.
- The verb "do" is used in a non-standard fashion, presumably for emphasis or to produce a dramatic impact.

Task 9: Insights and Real-World Application

NLP analysis of comments and postings involving gender and sexual orientation groups can discover key trends, attitudes, and influences in public conversation. By studying such content, our initiative provides significant insights into how these groups are depicted, discussed, and perceived on digital platforms. The analysis has uncovered themes like support, awareness, and inclusion. Some instances of bias, stereotyping, or aggression, which typically differ across contexts and platforms can be seen.

1. Sentiment Patterns: Analysis of sentiment demonstrates that debates regarding gender and sexual orientation groupings often hold heated viewpoints. Supportive remarks usually include language suggesting solidarity, such as "respect," "rights," and "equality."

2. Topic Clustering: Topic modeling exposes topics such as advocacy, legal rights, workplace diversity, and representation in the media. Many posts center on celebrating Pride events or campaigning for equal rights, while others address issues including discrimination and job biases.

3. Emerging Trends: Temporal analysis demonstrates increased conversations surrounding gender-neutral terminology and non-binary identification. This development reflects rising awareness and shifts in societal attitudes, particularly among younger groups.

Real-World Applications

1. **Social Media Analysis:** NGOs and legislators can utilize this information to analyze public sentiment towards gender and sexual orientation issues on platforms like Twitter, Instagram, or Reddit. Organizations can measure the acceptance of LGBTQ+ inclusive advertising campaigns or discover developing trends to drive marketing strategies.

2. **Content Moderation:** By finding patterns in hate speech or discriminatory language, social media companies can boost content moderation algorithms. Recognizing surges in hateful language during specific events can help platforms engage to ensure safe online spaces.

3. **Corporate Inclusion Policies:** Insights from workplace-related blogs can aid firms in building more inclusive settings. Assessing employee input, firms might identify areas where gender or sexual orientation biases can exist and apply training or policy modifications.

4. **Encouragement and Policymaking:** Using these findings to highlight concerns impacting communities and inform campaigns. Policymakers can also identify areas where intervention or regulation is needed, such as eliminating employment discrimination or boosting representation in the media.

Impact on Decision-Making

The insights gained influences decision-making by offering a data-driven understanding of public conversation. Businesses can match their branding and product strategy with values that resonate with their audience, improving consumer loyalty.

Policymakers may prioritize topics that reflect public concerns, while social media sites can develop algorithms to promote better online interactions.

Specific Examples

A fashion firm launching a gender-neutral clothing line might examine public input to adjust their campaign language and product designs. If the analysis finds strong support for diversity but concerns about price, the company can modify pricing tactics to cater to their audience.

Academics studying societal views toward LGBTQ+ populations might utilize the findings to chart changes in public sentiment over time, identify places with higher bias levels, or analyze the efficacy of awareness efforts.

NLP analysis of gender and sexual orientation-related text gives important insights that are both actionable and meaningful. From developing inclusive corporate strategies to optimizing social media experiences, these findings have extensive applicability in addressing real-world difficulties and promoting justice and understanding in society. Forming guidelines and policies to automatically remove or restrict comments and posts that are related to hate and discrimination can assist in maintaining a healthy community.

Task 10: Implement an Advanced NLP Technique

Text summary in NLP entails compressing a big body of text into a shorter version while keeping its key information. Using a pipeline for text summarization often comprises multiple stages. A summarizing pipeline automates this process, including preprocessing, summarizing, and post-processing phases, enabling efficient and scalable summarization of huge texts, valuable in applications like news aggregation, document summarization, and content recommendation systems.

1. Text summarization

This code uses the Hugging Face transformers library to perform text summarization on a subset of text data. The summarizer pipeline, which utilizes the "facebook/bart-large-cnn" model, is loaded for performing summarization tasks. The "summarize_text" function is defined to take a text input, summarize it using the model, and return the summary with a specified maximum and minimum length.

```
from transformers import pipeline

# Load the summarization pipeline
summarizer = pipeline("summarization", model="facebook/bart-large-cnn")

dfsum = df1.head(10)

# Function to summarize text
def summarize_text(text):
    try:
        # Perform summarization
        summary = summarizer(text, max_length=50, min_length=25, do_sample=False)
        return summary[0]['summary_text']
    except Exception as e:
        # Return the original text in case of an error
        return text

# Apply summarization to the 'commentText' column
dfsum['summarizedCommentText'] = dfsum['sentencedlemmatizedcommentText'].apply(summarize_text)
```

The function is then applied to the `sentenceLemmatizedCommentText` column of the first 10 rows of the Data Frame, creating a new column containing the summaries. This process automates the task of summarizing text data for easier analysis and presentation.

	<code>sentenceLemmatizedCommentText</code>	<code>summarizedCommentText</code>
0	oh no not bloodline whatever do recl...	Oh no not bloodline whatever do recl...
1	that s lot word no one sleep I	that s a lot word no one sleep I I'm not...
2	just another way shame	CNN.com will feature iReporter photos in a wee...
3	speak like old resentful man best way woman	CNN.com will feature iReporter photos in a wee...
4	I don t know ancestor fight mountain lion I ...	I don t know ancestor fight mountain lion I ...
5	some idiot never understand even worth time ef...	Some idiot never understand even worth time ef...
6	the audacity walk womb that s it	The audacity walk womb that s it. The audaci...
7	the fact I m still attract man prove sexuality...	"The fact I m still attract man prove sexualit...
8	I lot show life year high education I o...	"I spend life teach nurture guide child" "...
9	but many mag man provide well life child ...	Many mag man provide well life child and t...

2. Machine Translation

Machine translation in Natural Language Processing (NLP) includes mechanically translating text or speech using algorithms and models. It's commonly utilized in real-time communication, content localization, and multilingual information retrieval. Deep learning-based models increase translation quality, making it beneficial in businesses like e-commerce, travel, and worldwide communications.

I. English-to-French translation from a pre-trained model

This code uses the “MarianMT” model from the Hugging Face transformers library to perform English-to-French translation on a subset of a dataframe. It loads a pre-trained “MarianMT” model and tokenizer for English-to-French translation (Helsinki-NLP/opus-mt-en-fr). Then, it defines a function “`translate_text`”, which takes a piece of text, tokenizes it, generates the translated text using the model, and decodes it back to a human-readable form.

```

from transformers import MarianMTModel, MarianTokenizer
import pandas as pd

# Step 1: Load Pre-trained MarianMT Model for English-to-French Translation
model_name = "Helsinki-NLP/opus-mt-en-fr"
tokenizer = MarianTokenizer.from_pretrained(model_name)
model = MarianMTModel.from_pretrained(model_name)

dftrans = df1.head(10)

# Define a Function for Translation
def translate_text(text, tokenizer, model):
    try:
        # Tokenize input text
        tokenized_text = tokenizer(text, return_tensors="pt", padding=True, truncation=True)
        # Generate translation
        translated = model.generate(**tokenized_text)
        # Decode translation to text
        return tokenizer.decode(translated[0], skip_special_tokens=True)
    except Exception as e:
        return str(e) # Handle any translation errors gracefully

# Translate the `commentText` Column and Save in a New Column
dftrans['translatedCommentText'] = dftrans['sentencedlemmatizedcommentText'].apply(lambda x: translate_text(x, tokenizer, model))

```

Finally, the code applies this translation function to the `sentencedlemmatizedcommentText` column of the dataframe, translating the first 10 rows and saving the results in a new column. This allows for batch translation of text data from English to French.

	sentencedlemmatizedcommentText	translatedCommentText
0	oh no not bloodline whatever do recli...	Oh non, pas de sang, quoi qu'il en soit, évano...
1	that s lot word no one sleep I	Je ne dors pas.
2	just another way shame	juste une autre façon de honte
3	speak like old resentful man best way woman	parler comme un vieil homme rancunier meilleur...
4	I don t know ancestor fight mountain lion I ...	Je ne sais pas l'ancêtre combat lion de montag...
5	some idiot never understand even worth time ef...	certaines idiots ne comprennent jamais même la ...
6	the audacity walk womb that s it	l'audace marche le ventre qui l'est
7	the fact I m still attract man prove sexuality...	Le fait que j'attire toujours l'homme prouve q...
8	I lot show life year high education I o...	I lot show life year high education I own busi...
9	but many mag man provide well life child ...	mais beaucoup d'homme de mag fournissent bien ...

3. Transformer-based models

Transformer-based models, such as BERT, GPT, and T5, are frequently used in Natural Language Processing for tasks like text classification, translation, summarization, and question answering. They evaluate long-range dependencies in text, catch complicated patterns, and are pre-trained on enormous text samples for high-quality language representations.

I. Developing a Transformer-based model

This code utilizes the pre-trained T5 model from the transformers library to generate summaries for text data. It first loads the T5 tokenizer and model. Then, it defines a “summarize_text” function, which takes a text input, adds a "summarize:" prefix, tokenizes the input, and generates a summary with specified maximum input and output lengths. The “model.generate” method uses beam search to generate a concise summary of the text.

```
from transformers import T5Tokenizer, T5ForConditionalGeneration
import pandas as pd

# Load Pre-trained T5 Model
model_name = "t5-small" # You can also use "t5-base" or "t5-large" for better performance
tokenizer = T5Tokenizer.from_pretrained(model_name)
model = T5ForConditionalGeneration.from_pretrained(model_name)

# Define a Function for Summarization
def summarize_text(text, max_input_length=512, max_output_length=150):
    try:
        # Preprocess input
        input_text = f"summarize: {text}"
        inputs = tokenizer(input_text, return_tensors="pt", max_length=max_input_length, truncation=True)

        # Generate summary
        summary_ids = model.generate(inputs["input_ids"], max_length=max_output_length, num_beams=4, early_stopping=True)

        # Decode and return the summary
        return tokenizer.decode(summary_ids[0], skip_special_tokens=True)
    except Exception as e:
        return str(e) # Handle any errors gracefully

dfmodel = df1.head(10)

# Apply Summarization to the `commentText` Column
dfmodel['modelCommentText'] = dfmodel['sentencedlemmatizedcommentText'].apply(lambda x: summarize_text(x))
```

The function is then applied to the `sentencedlemmatizedcommentText` column of the first 10 rows of the dataframe, and the resulting summaries are stored in a new column, in the dataframe.

	sentencedlemmatizedcommentText	modelCommentText
0	oh no not bloodline whatever do recli...	recline fainting couch. recline fainting.
1	that s lot word no one sleep I	thats lot word no one sleeps I'm a s lot word ...
2	just another way shame	just another way shame
3	speak like old resentful man best way woman	speak like old resentful man best way woman. s...
4	I don t know ancestor fight mountain lion I ...	ancestor fight mountain lion I know paternal g...
5	some idiot never understand even worth time ef...	some idiot never understand even worth time ef...
6	the audacity walk womb that s it	the audacity walk womb that s it s it.
7	the fact I m still attract man prove sexuality...	the fact I m still attract man prove sexuality...
8	I lot show life year high education I o...	my ex husband beat we rape daughter he prison ...
9	but many mag man provide well life child ...	many mag man provide well life child and thise...

The advanced NLP approach employed in this challenge is text summarization using the T5 transformer-based model. This model is particularly effective at creating short summaries of lengthier text inputs. The summarization approach is applied to the column of the dataset, where it condenses each text into a more readable style. The results of the summarization indicate the model's ability to capture crucial aspects of the text while discarding less critical information.

The usefulness of this technique rests in its capacity to automate the summarizing of vast quantities of text, making it easier for users to obtain vital information quickly without going through full publications. This is particularly beneficial in applications like customer feedback analysis, news aggregation, or scientific research, where summarizing massive datasets or lengthy texts can save time and increase decision-making. The usage of a transformer-based model like T5 enables greater performance in text generation tasks, delivering summaries that are coherent and contextually relevant. By implementing this technique offering summarized material that enhances the overall user experience and promotes quicker insights from text data.

Datasets and documentation: -

<https://github.com/malindu101/PUSL3189-Natural-Language-Processing>

References

1. aswintechguy (2024). *Machine-Learning-Projects/Twitter Sentiment Analysis - NLP/Twitter Sentiment Analysis - NLP.ipynb at master · aswintechguy/Machine-Learning-Projects*. [online] GitHub. Available at: <https://github.com/aswintechguy/Machine-Learning-Projects/blob/master/Twitter%20Sentiment%20Analysis%20-%20NLP/Twitter%20Sentiment%20Analysis%20-%20NLP.ipynb>
2. aswintechguy (2020). *Machine-Learning-Projects/Toxic Comment Classification - Multi Label - NLP/Toxic Comment Classification - Multi Label - NLP.ipynb at master · aswintechguy/Machine-Learning-Projects*. [online] GitHub. Available at: <https://github.com/aswintechguy/Machine-LearningProjects/blob/master/Toxic%20Comment%20Classification%20-%20Multi%20Label%20-%20NLP/Toxic%20Comment%20Classification%20-%20Multi%20Label%20-%20NLP.ipynb> [Accessed 19 Dec. 2024].
3. Proxyway (2022). *How to Scrape Data From Facebook Accounts | Python Tutorial*. [online] YouTube. Available at: <https://www.youtube.com/watch?v=WnFAaWYecJs> [Accessed 19 Dec. 2024].
4. Twitter Sentiment Analysis (NLP) | Machine Learning | Python (2021). *Twitter Sentiment Analysis (NLP) | Machine Learning | Python*. [online] YouTube. Available at: <https://youtu.be/RLfUyn3HoaE?si=92Yvy9xAIRXGSY6a> [Accessed 19 Dec. 2024].
5. Model, P. (2022). *Realtime Twitter Sentiment Analysis using Pretrained Model (NLP) | Python*. [online] YouTube. Available at: https://youtu.be/26ZSHmUoBeM?si=-KCE7M_ToO5fddrx [Accessed 19 Dec. 2024].