Part-1

1) Implement BPE Algorithm

Developed a Python implementation of the Byte Pair Encoding (BPE) algorithm, covering key steps such as learning byte pair merges and encoding/decoding using the learned merge operations.

In [10]:

```
from collections import defaultdict, Counter

# Function to compute statistics on pairs of characters in the vocabulary
def stats(vocab):
    pairs = defaultdict(int)
    for word, freq in vocab.items():
        symbols = word.split()
        for i in range(len(symbols) - 1):
            # Count frequency of pairs of characters in each word
            pairs[symbols[i], symbols[i+1]] += freq
    return pairs

# Function to merge a pair of characters in the vocabulary
def merge(pair, vocab):
    new_vocab = {}
    bigram = ' '.join(pair)
    replacement = ''.join(pair)
    for word in vocab:
        # Merge occurrences of the pair in each word
        new_word = word.replace(bigram, replacement)
        new_vocab[new_word] = vocab[word]
    return new_vocab

# Function to generate the initial vocabulary from the input text
def get_vocab(text):
    vocab = Counter(text.split())
    return {word: freq for word, freq in vocab.items()}
```

The code defines an `encode` function that recursively breaks a string into tokens based on a given vocabulary. The `decode` function reverses this process, reconstructing the original string from a list of tokens.

In [11]:

```
def encode(vocab, string):
    def recursive_encode(s):
        # If the substring 's' is in the vocabulary, return it as a single token
        if s in vocab:
            return [s]
        else:
            # Try to split the substring into two parts and recursively encode each part
            for i in range(1, len(s)):
                left = s[:i]
                right = s[i:]

                # If both left and right parts are in the vocabulary, recursively encode
them
                if left in vocab and right in vocab:
                    return recursive_encode(left) + recursive_encode(right)

            # If no valid split is found, return the original substring as a single toke
n
            return [s]

    # Start the recursive encoding process
```

```
    return recursive_encode(string)

def decode(tokens):
    # Join the tokens to reconstruct the original string
    return ''.join(tokens)
```

2. Train on NLTK Dataset

The code combines texts from three books into a single string, then performs Byte Pair Encoding (BPE) to create a vocabulary. It initializes with a set of characters and iteratively merges the most frequent character pairs. The resulting main vocabulary represents the characters and character pairs that frequently occur in the combined texts. The final vocabulary is displayed after a specified number of merges (10^6).

In [12]:

```
import nltk

# List of books to be trained
books = ['austen-emma.txt', 'blake-poems.txt', 'shakespeare-hamlet.txt']
texts = []

# Combine the texts of selected books into a single string
for book in books:
    texts.append(nltk.corpus.gutenberg.raw(book))

text = ' '.join(texts)

# Initialize the main vocabulary with unique characters and character pairs
main_vocab = set(text.replace(" ", "_"))
print(main_vocab)

# Perform Byte Pair Encoding (BPE) to create the vocabulary
vocab = get_vocab(text.replace(" ", "_ "))
vocab = {' '.join(word): freq for word, freq in vocab.items()}
print("*****")

# Set the number of merges
num_merges = 10**6

# Perform BPE merges iteratively
for i in range(num_merges):
    pairs = stats(vocab)

    # If no more pairs are found, exit the loop
    if not pairs:
        break

    # Select the most frequent pair
    best_pair = max(pairs, key=pairs.get)

    # Add the merged pair to the main vocabulary
    main_vocab.add(best_pair[0] + best_pair[1])

    # Update the vocabulary by merging the selected pair
    vocab = merge(best_pair, vocab)

# Display the final main vocabulary
print(main_vocab)
```

```
{'L', '"', 'j', ';', 'q', 'z', 'D', '8', 'b', '6', '9', 'l', 'S', "'", '[', '3', 'G', 'R'
, 'i', '_', 'M', 'f', 'K', 'c', 'A', 'W', 'I', '.', 'H', 'T', 'r', '(', 'F', 'p', 's', '?
', '`', 'w', 'h', 'n', 'd', '0', 'N', 'O', ')', '!', 'X', 'o', 'v', 'm', 'u', 'J', 't', '
g', ':', '2', '-', 'U', 'Z', 'E', ',', 'k', 'y', 'P', 'Q', '7', 'e', '4', 'x', 'C', '&',
'\n', 'Y', '5', ']', 'V', 'B', '1', 'a'}
*****

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[12], line 27
```

```
     25 # Perform BPE merges iteratively
     26 for i in range(num_merges):
---> 27     pairs = stats(vocab)
     29     # If no more pairs are found, exit the loop
     30     if not pairs:

Cell In[10], line 10, in stats(vocab)
      7     symbols = word.split()
      8     for i in range(len(symbols) - 1):
      9         # Count frequency of pairs of characters in each word
---> 10         pairs[symbols[i], symbols[i+1]] += freq
     11 return pairs

KeyboardInterrupt:
```

3. Test on NLTK Dataset

In [4]:

```
books = ["chesterton-thursday.txt", "edgeworth-parents.txt", "melville-moby_dick.txt"]
texts = []
for book in books:
    texts.append(nltk.corpus.gutenberg.raw(book))

# Concatenate the texts of the books into a single string
text = ' '.join(texts)

new_text=text.replace(" ","_ ")+"_"
print(new_text[:500])
```

```
[The_ Man_ Who_ Was_ Thursday_ by_ G._ K._ Chesterton_ 1908]

To_ Edmund_ Clerihew_ Bentley

A_ cloud_ was_ on_ the_ mind_ of_ men,_ and_ wailing_ went_ the_ weather,
Yea,_ a_ sick_ cloud_ upon_ the_ soul_ when_ we_ were_ boys_ together.
Science_ announced_ nonentity_ and_ art_ admired_ decay;
The_ world_ was_ old_ and_ ended:_ but_ you_ and_ I_ were_ gay;
Round_ us_ in_ antic_ order_ their_ crippled_ vices_ came--
Lust_ that_ had_ lost_ its_ laughter,_ fear_ that_ had_ lost_ its_ shame.
Like_ t
```

3.1) BPE Algorithm on the Test Dataset

In [5]:

```
#BPE algorithm

print(new_text[:500])
encoded=[]
# print(main_vocab)
for i in new_text.split():
    # print(encode(main_vocab,i))
    encoded+=encode(main_vocab,i)

# print(encoded)
encoded_text=" ".join(encoded)
print("******")
# print(encoded_text[:500])
print(encoded[:500])
```

```
[The_ Man_ Who_ Was_ Thursday_ by_ G._ K._ Chesterton_ 1908]

To_ Edmund_ Clerihew_ Bentley

A_ cloud_ was_ on_ the_ mind_ of_ men,_ and_ wailing_ went_ the_ weather,
Yea,_ a_ sick_ cloud_ upon_ the_ soul_ when_ we_ were_ boys_ together.
Science_ announced_ nonentity_ and_ art_ admired_ decay;
The_ world_ was_ old_ and_ ended:_ but_ you_ and_ I_ were_ gay;
```

```
Round_ us_ in_ antic_ order_ their_ crippled_ vices_ came--
Lust_ that_ had_ lost_ its_ laughter,_ fear_ that_ had_ lost_ its_ shame.
Like_ t
******
['[The_', 'Man_', 'Who_', 'W', 'as_', 'Thursday_', 'b', 'y_', 'G', '._', 'K', '._', 'Ches
terton_', '1908]', 'T', 'o_', 'Edmund_', 'Clerihew_', 'Bentley', 'A', '_', 'cloud_', 'was
_', 'on_', 'the_', 'mind_', 'of_', 'men,_', 'and_', 'wailing_', 'went_', 'the_', 'weather
,', 'Yea,_', 'a_', 'sick_', 'cloud_', 'upon_', 'the_', 'soul_', 'when_', 'w', 'e_', 'were
_', 'boys_', 'together.', 'Science_', 'announced_', 'nonentity_', 'and_', 'ar', 't_', 'ad
mired_', 'decay;', 'T', 'he_', 'world_', 'was_', 'o', 'ld_', 'and_', 'ended:_', 'but_', '
you_', 'and_', 'I_', 'were_', 'gay;', 'Round_', 'u', 's_', 'in_', 'antic_', 'order_', 'th
eir_', 'crippled_', 'vices_', 'came--', 'Lust_', 'that_', 'had_', 'lost_', 'it', 's_', 'l
aughter,_', 'fear_', 'that_', 'had_', 'lost_', 'it', 's_', 'shame.', 'Like_', 'the_', 'wh
ite_', 'lock_', 'of_', 'Whistler,_', 'that_', 'li', 't_', 'ou', 'r_', 'aimless_', 'gloom,
', 'M', 'en_', 'showed_', 'their_', 'own_', 'white_', 'feather_', 'as_', 'proudly_', 'as_
', 'a_', 'plume.', 'Life_', 'was_', 'a_', 'f', 'ly_', 'that_', 'faded,_', 'and_', 'death_
', 'a_', 'drone_', 'that_', 'stung;', 'T', 'he_', 'world_', 'was_', 'very_', 'o', 'ld_',
'indeed_', 'when_', 'you_', 'and_', 'I_', 'were_', 'young.', 'They_', 'twisted_', 'even_'
, 'decent_', 's', 'in_', 'to_', 'shapes_', 'not_', 'to_', 'b', 'e_', 'named:', 'M', 'en_'
, 'were_', 'ashamed_', 'of_', 'honour;_', 'but_', 'w', 'e_', 'were_', 'not_', 'ashamed.',
'Weak_', 'i', 'f_', 'w', 'e_', 'were_', 'and_', 'foolish,_', 'not_', 'thus_', 'w', 'e_',
'failed,_', 'not_', 'thus;', 'When_', 'that_', 'black_', 'Baal_', 'blocked_', 'the_', 'he
avens_', 'he_', 'had_', 'n', 'o_', 'hymns_', 'from_', 'u', 's', 'Children_', 'w', 'e_', '
were--our_', 'forts_', 'of_', 's', 'and_', 'were_', 'even_', 'as_', 'weak_', 'as_', 'eve,
', 'High_', 'as_', 'they_', 'went_', 'w', 'e_', 'piled_', 'them_', 'up_', 'to_', 'break_'
, 'that_', 'bitter_', 'sea.', 'Fools_', 'as_', 'w', 'e_', 'were_', 'in_', 'motley,_', 'a'
, 'll_', 'jangling_', 'and_', 'absurd,', 'When_', 'a', 'll_', 'church_', 'bells_', 'were_
', 'silent_', 'ou', 'r_', 'cap_', 'and_', 'beds_', 'were_', 'heard.', 'Not_', 'a', 'll_',
'unhelped_', 'w', 'e_', 'he', 'ld_', 'the_', 'fort,_', 'ou', 'r_', 'tiny_', 'flags_', 'un
furled;', 'Some_', 'giants_', 'laboured_', 'in_', 'that_', 'cloud_', 'to_', 'lift_', 'i',
't_', 'from_', 'the_', 'world.', 'I_', 'find_', 'again_', 'the_', 'book_', 'w', 'e_', 'fo
und,_', 'I_', 'feel_', 'the_', 'hour_', 'that_', 'flings', 'Far_', 'o', 'ut_', 'of_', 'fi
sh-shaped_', 'Paumanok_', 'some_', 'cry_', 'of_', 'cleaner_', 'things;', 'And_', 'the_',
'Green_', 'Carnation_', 'withered,_', 'as_', 'in_', 'forest_', 'fires_', 'that_', 'pass,'
, 'Roared_', 'in_', 'the_', 'wind_', 'of_', 'a', 'll_', 'the_', 'world_', 't', 'en_', 'mi
llion_', 'leaves_', 'of_', 'grass;', 'O', 'r_', 'sane_', 'and_', 'sweet_', 'and_', 'sudde
n_', 'as_', 'a_', 'bird_', 'sings_', 'in_', 'the_', 'rain--', 'Truth_', 'o', 'ut_', 'of_'
, 'Tusitala_', 'spoke_', 'and_', 'pleasure_', 'o', 'ut_', 'of_', 'pain.', 'Yea,_', 'cool_
', 'and_', 'clear_', 'and_', 'sudden_', 'as_', 'a_', 'bird_', 'sings_', 'in_', 'the_', 'g
rey,', 'Dunedin_', 'to_', 'Samoa_', 'spoke,_', 'and_', 'darkness_', 'unto_', 'day.', 'B',
'ut_', 'w', 'e_', 'were_', 'young;_', 'w', 'e_', 'lived_', 'to_', 'se', 'e_', 'God_', 'br
eak_', 'their_', 'bitter_', 'charms.', 'God_', 'and_', 'the_', 'good_', 'Republic_', 'com
', 'e_', 'riding_', 'back_', 'in_', 'arms:', 'W', 'e_', 'have_', 'se', 'en_', 'the_', 'Ci
ty_', 'of_', 'Mansoul,_', 'even_', 'as_', 'i', 't_', 'rocked,_', 'relieved--', 'Blessed_'
, 'ar', 'e_', 'they_', 'who_', 'did_', 'not_', 'see,_', 'but_', 'be', 'ing_', 'blind,_',
'believed.', 'This_', 'i', 's_', 'a_', 'tale_', 'of_', 'those_', 'o', 'ld_', 'fears,_', '
even_', 'of_', 'those_', 'emptied_', 'hells,', 'And_', 'none_', 'but_', 'you_', 'shall_',
'understand_', 'the_', 'true_', 'th', 'ing_', 'that_', 'i', 't_', 'tells--', 'O', 'f_', '
what_', 'colossal_', 'gods_', 'of_', 'shame_', 'c', 'ould_', 'cow_', 'm', 'en_', 'and_',
'yet_', 'crash,', 'O', 'f_', 'what_', 'huge_', 'devils_', 'hi', 'd_', 'the_', 'stars,_',
'yet_', 'fe', 'll_', 'a', 't_', 'a_', 'pistol_', 'flash.', 'T', 'he_', 'doubts_', 'that_'
]
```

4) NLTK Word Tokenizer (Refernce tokenizer) on the Test Dataset

In [6]:

```
#Word Tokenizer

import nltk
from nltk.tokenize import word_tokenize

# Download the Punkt tokenizer
# nltk.download('punkt')

# new_text = "This is an example sentence. Tokenize it!"
ref_tokens = nltk.word_tokenize(new_text)
```

```
print(ref_tokens[:500])
print(type(ref_tokens))
```

```
['[', 'The_', 'Man_', 'Who_', 'Was_', 'Thursday_', 'by_', 'G._', 'K._', 'Chesterton_', '1
908', ']', 'To_', 'Edmund_', 'Clerihew_', 'Bentley', 'A_', 'cloud_', 'was_', 'on_', 'the_
', 'mind_', 'of_', 'men', ',', '_', 'and_', 'wailing_', 'went_', 'the_', 'weather', ',',
'Yea', ',', '_', 'a_', 'sick_', 'cloud_', 'upon_', 'the_', 'soul_', 'when_', 'we_', 'were
_', 'boys_', 'together', '.', 'Science_', 'announced_', 'nonentity_', 'and_', 'art_', 'ad
mired_', 'decay', ';', 'The_', 'world_', 'was_', 'old_', 'and_', 'ended', ':', '_', 'but_
', 'you_', 'and_', 'I_', 'were_', 'gay', ';', 'Round_', 'us_', 'in_', 'antic_', 'order_',
'their_', 'crippled_', 'vices_', 'came', '--', 'Lust_', 'that_', 'had_', 'lost_', 'its_',
'laughter', ',', '_', 'fear_', 'that_', 'had_', 'lost_', 'its_', 'shame', '.', 'Like_', '
the_', 'white_', 'lock_', 'of_', 'Whistler', ',', '_', 'that_', 'lit_', 'our_', 'aimless_
', 'gloom', ',', 'Men_', 'showed_', 'their_', 'own_', 'white_', 'feather_', 'as_', 'proud
ly_', 'as_', 'a_', 'plume', '.', 'Life_', 'was_', 'a_', 'fly_', 'that_', 'faded', ',', '_
', 'and_', 'death_', 'a_', 'drone_', 'that_', 'stung', ';', 'The_', 'world_', 'was_', 've
ry_', 'old_', 'indeed_', 'when_', 'you_', 'and_', 'I_', 'were_', 'young', '.', 'They_', '
twisted_', 'even_', 'decent_', 'sin_', 'to_', 'shapes_', 'not_', 'to_', 'be_', 'named', '
:', 'Men_', 'were_', 'ashamed_', 'of_', 'honour', ';', '_', 'but_', 'we_', 'were_', 'not_
', 'ashamed', '.', 'Weak_', 'if_', 'we_', 'were_', 'and_', 'foolish', ',', '_', 'not_', '
thus_', 'we_', 'failed', ',', '_', 'not_', 'thus', ';', 'When_', 'that_', 'black_', 'Baal
_', 'blocked_', 'the_', 'heavens_', 'he_', 'had_', 'no_', 'hymns_', 'from_', 'us', 'Child
ren_', 'we_', 'were', '--', 'our_', 'forts_', 'of_', 'sand_', 'were_', 'even_', 'as_', 'w
eak_', 'as_', 'eve', ',', 'High_', 'as_', 'they_', 'went_', 'we_', 'piled_', 'them_', 'up
_', 'to_', 'break_', 'that_', 'bitter_', 'sea', '.', 'Fools_', 'as_', 'we_', 'were_', 'in
_', 'motley', ',', '_', 'all_', 'jangling_', 'and_', 'absurd', ',', 'When_', 'all_', 'chu
rch_', 'bells_', 'were_', 'silent_', 'our_', 'cap_', 'and_', 'beds_', 'were_', 'heard', '
.', 'Not_', 'all_', 'unhelped_', 'we_', 'held_', 'the_', 'fort', ',', '_', 'our_', 'tiny_
', 'flags_', 'unfurled', ';', 'Some_', 'giants_', 'laboured_', 'in_', 'that_', 'cloud_',
'to_', 'lift_', 'it_', 'from_', 'the_', 'world', '.', 'I_', 'find_', 'again_', 'the_', 'b
ook_', 'we_', 'found', ',', '_', 'I_', 'feel_', 'the_', 'hour_', 'that_', 'flings', 'Far_
', 'out_', 'of_', 'fish-shaped_', 'Paumanok_', 'some_', 'cry_', 'of_', 'cleaner_', 'thing
s', ';', 'And_', 'the_', 'Green_', 'Carnation_', 'withered', ',', '_', 'as_', 'in_', 'for
est_', 'fires_', 'that_', 'pass', ',', 'Roared_', 'in_', 'the_', 'wind_', 'of_', 'all_',
'the_', 'world_', 'ten_', 'million_', 'leaves_', 'of_', 'grass', ';', 'Or_', 'sane_', 'an
d_', 'sweet_', 'and_', 'sudden_', 'as_', 'a_', 'bird_', 'sings_', 'in_', 'the_', 'rain',
'--', 'Truth_', 'out_', 'of_', 'Tusitala_', 'spoke_', 'and_', 'pleasure_', 'out_', 'of_',
'pain', '.', 'Yea', ',', '_', 'cool_', 'and_', 'clear_', 'and_', 'sudden_', 'as_', 'a_',
'bird_', 'sings_', 'in_', 'the_', 'grey', ',', 'Dunedin_', 'to_', 'Samoa_', 'spoke', ',',
'_', 'and_', 'darkness_', 'unto_', 'day', '.', 'But_', 'we_', 'were_', 'young', ';', '_',
'we_', 'lived_', 'to_', 'see_', 'God_', 'break_', 'their_', 'bitter_', 'charms', '.', 'Go
d_', 'and_', 'the_', 'good_', 'Republic_', 'come_', 'riding_', 'back_', 'in_', 'arms', ':
', 'We_', 'have_', 'seen_', 'the_', 'City_', 'of_', 'Mansoul', ',', '_', 'even_', 'as_',
'it_', 'rocked', ',', '_', 'relieved', '--', 'Blessed_', 'are_', 'they_', 'who_', 'did_',
'not_', 'see', ',', '_', 'but_', 'being_', 'blind', ',', '_', 'believed', '.', 'This_', '
is_', 'a_', 'tale_', 'of_', 'those_', 'old_', 'fears', ',', '_', 'even_', 'of_', 'those_'
, 'emptied_', 'hells', ',', 'And_', 'none_', 'but_', 'you_', 'shall_', 'understand_', 'th
e_', 'true_', 'thing_', 'that_', 'it_', 'tells', '--', 'Of_', 'what_', 'colossal_', 'gods
_', 'of_', 'shame_', 'could_', 'cow_', 'men_', 'and_', 'yet_', 'crash', ',', 'Of_', 'what
_', 'huge_', 'devils_', 'hid_', 'the_']
<class 'list'>
```

5. Compare with Standard Tokenization

In [25]:

```
#Metrics

from nltk import ngrams

class Metric:
    def __init__(self, reference, predicted):
        self.reference = reference
        self.predicted = predicted

    def tokenization_accuracy(self):
        ref=set(self.reference)
        pred=self.predicted
        correct_tokens=0
        for i in pred:
```

```
            if i in ref:
                correct_tokens+=1

        total_tokens = len(self.reference)
        return (correct_tokens / total_tokens)

    def tokenization_coverage(self):
        ref = set(self.reference)
        pred = set(self.predicted)
        intersection = len(ref.intersection(pred))
        return (intersection / len(ref))

    def calculate_precision_recall_f1(self):
        tp = len(set(self.predicted).intersection(self.reference))
        fp = len(self.predicted) - tp
        fn = len(self.reference) - tp
        # print(tp,fp,fn)
        precision = tp / (tp + fp)
        recall = tp / (tp + fn)
        f1_score = 2 * (precision * recall) / (precision + recall)
        return precision, recall, f1_score

    def jaccard_similarity(self):
        reference = set(self.reference)
        predicted = set(self.predicted)
        intersection = len(reference.intersection(predicted))
        union = len(reference) + len(predicted) - intersection
        return intersection / union


reference_tokens = ref_tokens
predicted_tokens = encoded


metric_calculator = Metric(reference_tokens, predicted_tokens)

# Tokenization Accuracy
accuracy = metric_calculator.tokenization_accuracy()
print(f'Tokenization Accuracy: {accuracy:.2%}')

# Tokenization Coverage
coverage = metric_calculator.tokenization_coverage()
print(f'Tokenization Coverage: {coverage:.2%}')

# Precision, Recall, F1-score
precision, recall, f1 = metric_calculator.calculate_precision_recall_f1()
print(f'Precision: {precision}, Recall: {recall}, F1-score: {f1}')

# Jaccard Similarity
jaccard_similarity = metric_calculator.jaccard_similarity()
print(f'Jaccard Similarity: {jaccard_similarity}')
```

```
Tokenization Accuracy: 64.64%
Tokenization Coverage: 74.37%
29814 487739 550539
Precision: 0.057605694489260034, Recall: 0.051372182102961475, F1-score: 0.05431066047548
697
Jaccard Similarity: 0.4466583769045229
```

6. Visualizations:

In [33]:

```
text = "low low low low low lowest lowest newer newer newer newer newer newer wider wider
wider new new"

voc = get_vocab(text.replace(" ", "_ "))

main_vocab=set(text.replace(" ","_"))
print(main_vocab)
```

```
vocab = {' '.join(word): freq for word, freq in voc.items()}
print(vocab)
print("*"*50)

num_merges = 10
for i in range(num_merges):
    print(main_vocab)
    pairs = stats(vocab)
    print(pairs)
    if not pairs:
        break
    best_pair = max(pairs, key=pairs.get)
    print(best_pair)
    main_vocab.add(best_pair[0]+best_pair[1])
    vocab = merge(best_pair, vocab)
    print(main_vocab)
    print("="*50)
# print(main_vocab)
```

```
{'s', 'r', 'l', 'o', 'n', 'd', 'e', 't', 'i', '_', 'w'}
{'l o w _': 5, 'l o w e s t _': 2, 'n e w e r _': 6, 'w i d e r _': 3, 'n e w _': 1, 'n e
w': 1}
**************************************************
{'s', 'r', 'l', 'o', 'n', 'd', 'e', 't', 'i', '_', 'w'}
defaultdict(<class 'int'>, {('l', 'o'): 7, ('o', 'w'): 7, ('w', '_'): 6, ('w', 'e'): 8, (
'e', 's'): 2, ('s', 't'): 2, ('t', '_'): 2, ('n', 'e'): 8, ('e', 'w'): 8, ('e', 'r'): 9,
('r', '_'): 9, ('w', 'i'): 3, ('i', 'd'): 3, ('d', 'e'): 3})
('e', 'r')
{'s', 'r', 'l', 'o', 'n', 'd', 'e', 't', 'i', 'er', '_', 'w'}
==================================================
{'s', 'r', 'l', 'o', 'n', 'd', 'e', 't', 'i', 'er', '_', 'w'}
defaultdict(<class 'int'>, {('l', 'o'): 7, ('o', 'w'): 7, ('w', '_'): 6, ('w', 'e'): 2, (
'e', 's'): 2, ('s', 't'): 2, ('t', '_'): 2, ('n', 'e'): 8, ('e', 'w'): 8, ('w', 'er'): 6,
('er', '_'): 9, ('w', 'i'): 3, ('i', 'd'): 3, ('d', 'er'): 3})
('er', '_')
{'s', 'r', 'l', 'o', 'n', 'd', 'er_', 'e', 't', 'i', 'er', '_', 'w'}
==================================================
{'s', 'r', 'l', 'o', 'n', 'd', 'er_', 'e', 't', 'i', 'er', '_', 'w'}
defaultdict(<class 'int'>, {('l', 'o'): 7, ('o', 'w'): 7, ('w', '_'): 6, ('w', 'e'): 2, (
'e', 's'): 2, ('s', 't'): 2, ('t', '_'): 2, ('n', 'e'): 8, ('e', 'w'): 8, ('w', 'er_'): 6
, ('w', 'i'): 3, ('i', 'd'): 3, ('d', 'er_'): 3})
('n', 'e')
{'s', 'r', 'l', 'o', 'n', 'd', 'ne', 'er_', 'e', 't', 'i', 'er', '_', 'w'}
==================================================
{'s', 'r', 'l', 'o', 'n', 'd', 'ne', 'er_', 'e', 't', 'i', 'er', '_', 'w'}
defaultdict(<class 'int'>, {('l', 'o'): 7, ('o', 'w'): 7, ('w', '_'): 6, ('w', 'e'): 2, (
'e', 's'): 2, ('s', 't'): 2, ('t', '_'): 2, ('ne', 'w'): 8, ('w', 'er_'): 6, ('w', 'i'):
3, ('i', 'd'): 3, ('d', 'er_'): 3})
('ne', 'w')
{'s', 'r', 'new', 'l', 'o', 'n', 'd', 'ne', 'er_', 'e', 't', 'i', 'er', '_', 'w'}
==================================================
{'s', 'r', 'new', 'l', 'o', 'n', 'd', 'ne', 'er_', 'e', 't', 'i', 'er', '_', 'w'}
defaultdict(<class 'int'>, {('l', 'o'): 7, ('o', 'w'): 7, ('w', '_'): 5, ('w', 'e'): 2, (
'e', 's'): 2, ('s', 't'): 2, ('t', '_'): 2, ('new', 'er_'): 6, ('w', 'i'): 3, ('i', 'd'):
3, ('d', 'er_'): 3, ('new', '_'): 1})
('l', 'o')
{'s', 'r', 'new', 'l', 'o', 'n', 'd', 'ne', 'er_', 'e', 'lo', 't', 'i', 'er', '_', 'w'}
==================================================
{'s', 'r', 'new', 'l', 'o', 'n', 'd', 'ne', 'er_', 'e', 'lo', 't', 'i', 'er', '_', 'w'}
defaultdict(<class 'int'>, {('lo', 'w'): 7, ('w', '_'): 5, ('w', 'e'): 2, ('e', 's'): 2,
('s', 't'): 2, ('t', '_'): 2, ('new', 'er_'): 6, ('w', 'i'): 3, ('i', 'd'): 3, ('d', 'er_
'): 3, ('new', '_'): 1})
('lo', 'w')
{'low', 's', 'r', 'new', 'l', 'o', 'n', 'd', 'ne', 'er_', 'e', 'lo', 't', 'i', 'er', '_',
'w'}
==================================================
{'low', 's', 'r', 'new', 'l', 'o', 'n', 'd', 'ne', 'er_', 'e', 'lo', 't', 'i', 'er', '_',
'w'}
defaultdict(<class 'int'>, {('low', '_'): 5, ('low', 'e'): 2, ('e', 's'): 2, ('s', 't'):
2, ('t', '_'): 2, ('new', 'er_'): 6, ('w', 'i'): 3, ('i', 'd'): 3, ('d', 'er_'): 3, ('new
```

```
', '_'): 1})
('new', 'er_')
{'low', 's', 'r', 'new', 'l', 'o', 'n', 'd', 'ne', 'er_', 'e', 'lo', 'newer_', 't', 'i',
'er', '_', 'w'}
==================================================
{'low', 's', 'r', 'new', 'l', 'o', 'n', 'd', 'ne', 'er_', 'e', 'lo', 'newer_', 't', 'i',
'er', '_', 'w'}
defaultdict(<class 'int'>, {('low', '_'): 5, ('low', 'e'): 2, ('e', 's'): 2, ('s', 't'):
2, ('t', '_'): 2, ('w', 'i'): 3, ('i', 'd'): 3, ('d', 'er_'): 3, ('new', '_'): 1})
('low', '_')
{'s', 'er', 'newer_', 'w', 'n', 'd', 'e', 'er_', 'low', 'new', 'l', 'o', 'ne', 'low_', 'r
', '_', 't', 'i', 'lo'}
==================================================
{'s', 'er', 'newer_', 'w', 'n', 'd', 'e', 'er_', 'low', 'new', 'l', 'o', 'ne', 'low_', 'r
', '_', 't', 'i', 'lo'}
defaultdict(<class 'int'>, {('low', 'e'): 2, ('e', 's'): 2, ('s', 't'): 2, ('t', '_'): 2,
('w', 'i'): 3, ('i', 'd'): 3, ('d', 'er_'): 3, ('new', '_'): 1})
('w', 'i')
{'s', 'er', 'newer_', 'w', 'n', 'd', 'e', 'er_', 'low', 'new', 'l', 'o', 'ne', 'low_', 'r
', '_', 'wi', 't', 'i', 'lo'}
==================================================
{'s', 'er', 'newer_', 'w', 'n', 'd', 'e', 'er_', 'low', 'new', 'l', 'o', 'ne', 'low_', 'r
', '_', 'wi', 't', 'i', 'lo'}
defaultdict(<class 'int'>, {('low', 'e'): 2, ('e', 's'): 2, ('s', 't'): 2, ('t', '_'): 2,
('wi', 'd'): 3, ('d', 'er_'): 3, ('new', '_'): 1})
('wi', 'd')
{'s', 'wid', 'er', 'newer_', 'w', 'n', 'd', 'e', 'er_', 'low', 'new', 'l', 'o', 'ne', 'lo
w_', 'r', '_', 'wi', 't', 'i', 'lo'}
==================================================
```

7. Report and Discussion

7.1 Prepare a detailed report documenting the implementation, experimental setup, and results.

Ans. Implemented Byte Pair Encoding (BPE) algorithm, encompassing crucial stages like learning byte pair merges and encoding/decoding based on the acquired merge operations. The implementation involved consolidating text from three books into a unified string, subsequently applying BPE to generate a vocabulary. The process initiated with a predefined set of characters and iteratively merged the most frequently occurring character pairs. The resultant primary vocabulary encapsulates frequently encountered characters and character pairs within the amalgamated texts. The final vocabulary is revealed after a specified number of merges (10^6). Subsequently, the BPE algorithm encoded three books using this vocabulary. For performance evaluation, the BPE algorithm was compared against the NLTK library's word tokenizer, serving as a ground truth. The outcomes were subjected to analysis using various metrics.

7.2 Discuss the strengths and weaknesses of BPE, and compare it with standard tokenization methods.

Ans

-> Byte Pair Encoding (BPE):

- **Strengths - 1. BPE is adaptive to the data and doesn't require a predefined vocabulary. It dynamically builds a vocabulary based on the most frequent byte pairs, allowing it to handle rare or unseen words efficiently.**

1. **BPE naturally produces subword representations, which can be useful for handling morphologically rich languages and capturing meaningful subword units.**

- **Weaknesses - 1. BPE may create ambiguity in tokenization, especially when dealing with homographs or homophones. The same sequence of subword units might represent different words.**

1. **While BPE can handle rare words well, it can lead to a large vocabulary size, especially if subword units are not merged intelligently. This could impact computational efficiency**

- **BPE VS Word Tokenizer - Byte Pair Encoding (BPE) BPE excels in adaptability by dynamically constructing a** vocabulary based on frequent byte pairs, making it effective for handling rare words and producing subword representations. It compresses rare words efficiently and offers variable-length encoding. However, BPE's effectiveness depends on the training data, and it may introduce token ambiguity. On the other hand, NLTK's word tokenizer provides human-readable and interpretable tokenizations, but it may struggle with out-of-vocabulary words and produces fixed-length representations. It is more domain-independent but lacks the flexibility to handle variable-length subword units. The choice between BPE and NLTK's word tokenizer depends on specific task requirements, data characteristics, and the desired balance between adaptability and interpretability. BPE is suitable for scenarios where variable-length subword units and compression of rare words are crucial, while NLTK's word tokenizer may be preferable when human interpretability and domain independence are more important.

7.3 Address any challenges encountered during implementation and suggest potential improvements.

Ans - Implementing the Byte Pair Encoding (BPE) algorithm may pose challenges, including vocabulary size concerns, token ambiguity, training data dependency, computational complexity, and special character handling issues. Improvements can be made by optimizing the merging strategy, implementing sophisticated post-processing for token disambiguation, exploring regularization techniques, parallelizing computations, and customizing BPE for special character handling. Additionally, interactive merging or feedback mechanisms can enhance adaptability. Balancing these improvements can lead to a more robust BPE implementation, addressing challenges and ensuring effective subword tokenization across various datasets and tasks.