TP: Bag of word and N-gram

Name: KRY Senghort

Group: I4-AMS-B

ID: e20200706

I. N-gram

Problem-1:

1. **Import Libraries**: Import `nltk` for tokenization, `collections` for counting, and `sklearn` for TF-IDF.
2. **Download NLTK Resources** (if not done already)
3. **Define Sentences**: Define two sentences for analysis.

- sentence1 = "Today is a beautiful day for a walk."
- sentence2 = "The weather is too cloudy for a picnic."

4. **Text Preprocessing**: Write a `preprocess_text` function to convert text to lowercase and tokenize.
5. **Unigrams and Bigrams**:

- Create `find_unigrams` to identify single words in a sentence.
- Create `find_bigrams` to identify pairs of consecutive words in a sentence.

6. **Apply Functions**:

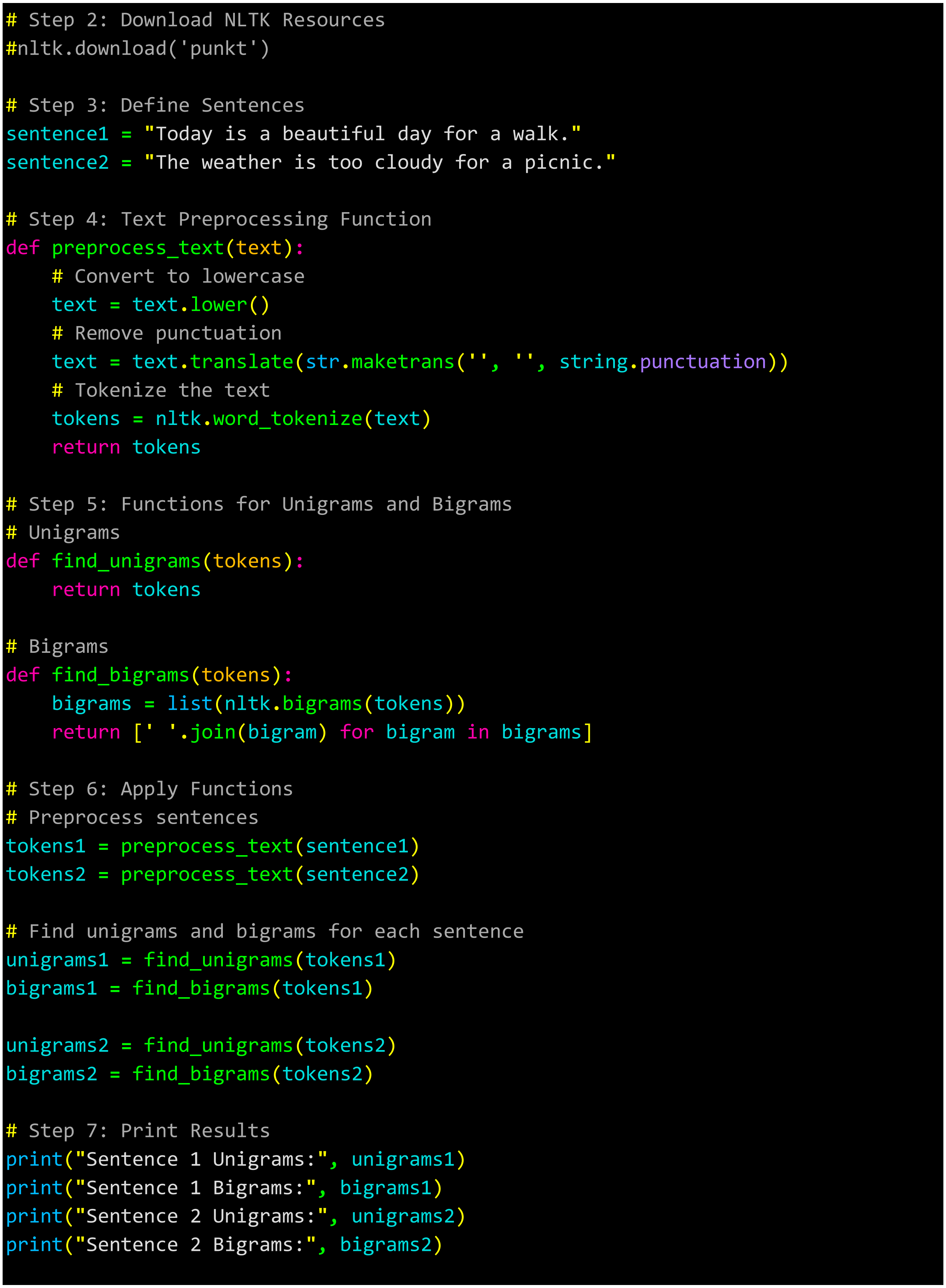
- Use `preprocess_text`, `find_unigrams`, and `find_bigrams` to analyze both sentences.

7. **Print Results**: Display unigrams and bigrams for each sentence.
8. **TF-IDF Vectorization**:

- Modify `preprocess_text` to remove punctuation.
- Create a `TfidfVectorizer` with bigram focus (`ngram_range=(2, 2)`) and fit it on the sentences.

8. **Display TF-IDF**: Show the TF-IDF matrix and individual bigram scores for each sentence.

```
# Step 1: Import Libraries
import pandas as pd
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
import string
```



```
# Step 2: Download NLTK Resources
#nltk.download('punkt')

# Step 3: Define Sentences
sentence1 = "Today is a beautiful day for a walk."
sentence2 = "The weather is too cloudy for a picnic."

# Step 4: Text Preprocessing Function
def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()
    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))
    # Tokenize the text
    tokens = nltk.word_tokenize(text)
    return tokens

# Step 5: Functions for Unigrams and Bigrams
# Unigrams
def find_unigrams(tokens):
    return tokens

# Bigrams
def find_bigrams(tokens):
    bigrams = list(nltk.bigrams(tokens))
    return [' '.join(bigram) for bigram in bigrams]

# Step 6: Apply Functions
# Preprocess sentences
tokens1 = preprocess_text(sentence1)
tokens2 = preprocess_text(sentence2)

# Find unigrams and bigrams for each sentence
unigrams1 = find_unigrams(tokens1)
bigrams1 = find_bigrams(tokens1)

unigrams2 = find_unigrams(tokens2)
bigrams2 = find_bigrams(tokens2)

# Step 7: Print Results
print("Sentence 1 Unigrams:", unigrams1)
print("Sentence 1 Bigrams:", bigrams1)
print("Sentence 2 Unigrams:", unigrams2)
print("Sentence 2 Bigrams:", bigrams2)
```

```
# Step 8: TF-IDF Vectorization
# Create a TfidfVectorizer with a bigram focus
vectorizer = TfidfVectorizer(ngram_range=(2, 2))

# Fit the vectorizer on the sentences
tfidf_matrix = vectorizer.fit_transform([sentence1, sentence2])
```

```
Sentence 1 Unigrams: ['today', 'is', 'a', 'beautiful', 'day', 'for', 'a', 'walk']
Sentence 1 Bigrams: ['today is', 'is a', 'a beautiful', 'beautiful day', 'day
for', 'for a', 'a walk']
Sentence 2 Unigrams: ['the', 'weather', 'is', 'too', 'cloudy', 'for', 'a',
'picnic']
Sentence 2 Bigrams: ['the weather', 'weather is', 'is too', 'too cloudy', 'cloudy
for', 'for a', 'a picnic']
```

```
# Step 9: Display TF-IDF Matrix
print("TF-IDF Matrix:")
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(),
columns=vectorizer.get_feature_names_out())
tfidf_df
```

TF-IDF Matrix:

beautiful day	cloudy for	day for	for picnic	for walk	is beautiful	is too	the weather	today is	too cloudy	weather is
0.447214	0.000000	0.447214	0.000000	0.447214	0.447214	0.000000	0.000000	0.447214	0.000000	0.000000
0.000000	0.408248	0.000000	0.408248	0.000000	0.000000	0.408248	0.408248	0.000000	0.408248	0.408248

```
print("Individual Bigram Scores:")
for sentence_index, sentence in enumerate([sentence1, sentence2]):
    print('-----------------------------------')
    print(f"Sentence {sentence_index + 1} TF-IDF Scores:")
    print('-----------------------------------')
    for bigram, score in zip(vectorizer.get_feature_names_out(),
tfidf_matrix.toarray()[sentence_index]):
        print(f"{bigram}: {score:.4f}")
```

```
Individual Bigram Scores:
-----------------------------------
Sentence 1 TF-IDF Scores:
-----------------------------------
beautiful day: 0.4472
cloudy for: 0.0000
day for: 0.4472
for picnic: 0.0000
for walk: 0.4472
is beautiful: 0.4472
is too: 0.0000
the weather: 0.0000
```

```
today is: 0.4472
too cloudy: 0.0000
weather is: 0.0000
-----------------------------------
Sentence 2 TF-IDF Scores:
-----------------------------------
beautiful day: 0.0000
cloudy for: 0.4082
day for: 0.0000
for picnic: 0.4082
for walk: 0.0000
is beautiful: 0.0000
is too: 0.4082
the weather: 0.4082
today is: 0.0000
too cloudy: 0.4082
weather is: 0.4082
```

II. Bag of word

Problem-2:

1. **Import Required Libraries**: We'll use `CountVectorizer` from `sklearn` for the Bag of Words model.
2. **Define Sentences**: Define the sample sentences in problem 1.
3. **Preprocess (Optional)**: Lowercase the text to ensure case consistency. This is optional if `CountVectorizer` is set to ignore case.
4. **Create CountVectorizer**: Initialize `CountVectorizer` to convert text into BoW format.
5. **Fit and Transform Sentences**:

   - Fit the vectorizer on the sentences to learn the vocabulary.
   - Transform the sentences into BoW vectors.

6. **Display Results**:

   - Print the matrix showing the count of each word in each sentence.
   - Print the vocabulary (feature names).

```
# Step 1: Import Required Libraries
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd

# Step 2: Define Sentences
sentences = [
    "Today is a beautiful day for a walk.",
    "The weather is too cloudy for a picnic."
]

# Step 3: Preprocess (Optional)
# CountVectorizer has lowercase=True by default, so it will automatically
ignore case.

# Step 4: Create CountVectorizer
# Initialize CountVectorizer to convert sentences into BoW format
vectorizer = CountVectorizer()

# Step 5: Fit and Transform Sentences
# Fit the vectorizer on the sentences to learn the vocabulary, then transform
the sentences into BoW vectors
bow_matrix = vectorizer.fit_transform(sentences)

# Step 6: Display Results
# Convert the matrix to a DataFrame for better readability
bow_df = pd.DataFrame(bow_matrix.toarray(),
columns=vectorizer.get_feature_names_out(), index=["Sentence 1", "Sentence
2"])
print("Bag of Words Matrix:")
bow_df
```

Bag of Words Matrix:

	beautiful	cloudy	day	for	is	picnic	the	today	too	walk	weather
Sentence 1	1	0	1	1	1	0	0	1	0	1	0
Sentence 2	0	1	0	1	1	1	1	0	1	0	1

```
print("\nVocabulary (Feature Names):")
vectorizer.get_feature_names_out()
```

```
Vocabulary (Feature Names):
array(['beautiful', 'cloudy', 'day', 'for', 'is', 'picnic', 'the',
       'today', 'too', 'walk', 'weather'], dtype=object)
```