```python
from preprocessing import Preprocessing
from ngrams import NGrams
from probability import Probability
from random import shuffle


def main():
    # Set the dataset path
    dataset_path = "../data/berkeley_restaurant_dataset.txt"

    # Create an instance of the Preprocessing class with the dataset path
    data = Preprocessing(dataset_path)

    # Preprocess the dataset
    data.preprocess()

    # Grab the preprocessed sentences to print them
    preprocessed_data = data.filtered_sentences

    # Create an instance of the NGrams class with the preprocessed data
    ngrams = NGrams(preprocessed_data)

    # Generate unigrams, bigrams, and trigrams
    ngrams.generate_ngrams()

    # Access the generated ngrams
    unigrams = ngrams.unigrams
    bigrams = ngrams.bigrams
    trigrams = ngrams.trigrams

    # For testing - Print some example ngrams
    # print("Unigrams:", unigrams[:10])
    # print("Bigrams:", bigrams[:10])
    # print("Trigrams:", trigrams[:10])

    # Print the requested statistics on word count, vocabulary size,
```

```
    # and total number of sentences
    ngrams.print_statistics()

    print("""

        PART 3
        ------

        Read the chapter on N-grams and generate figures 4.1 and 4.2
        for bigram
        counts. The figures do not have to be exact.

        """)
    print("------ Bigram Count Table - Figure 4.1 -------\n")
    ngrams.bigram_count_table()

    print("\n------ Bigram Probability Table - Figure 4.2 -------\n")
    ngrams.bigram_probability_table()

    print("""

        PART 4 & 5
        ------

        Calculate the joint probability for at least five sentences (with vocabulary in the dataset) using bigrams.
        Repeat step 2 using trigrams. Observe if the estimates have changed.

    """)
    print("\n------ Calculate bigram and trigram probabilities --------\n")

    probability = Probability(ngrams.bigrams, ngrams.trigrams)
    bigram_probs = probability.bigram_probabilities()
    trigram_probs = probability.trigram_probabilities()

    sentences = data.filtered_sentences
```

```python

    shuffle(sentences)
    examples = []
    for sentence in sentences:
        if len(sentence[1:]) > 2:
            examples.append(sentence)

    for sentence in examples[:5]:
        if len(sentence[1:]) > 2:
            print(f"Sentence: {' '.join(sentence[1:])}")
            print(f"Bigram Probability: {bigram_probs[(sentence[1], sentence[2])]}")
            print(f"Trigram Probability: {trigram_probs[(sentence[1], sentence[2], sentence[3])]}")
            print()


if __name__ == "__main__":
    main()


```

```python
import random
import nltk
from collections import Counter
from tabulate import tabulate


class NGrams:
    def __init__(self, preprocessed_data):
        self.preprocessed_data = preprocessed_data
        self.unigrams = []
        self.bigrams = []
        self.trigrams = []

    def generate_unigrams(self):
        for sentence in self.preprocessed_data:
            for token in sentence[1:]:
                self.unigrams.append(token)

    def generate_bigrams(self):
        for sentence in self.preprocessed_data:
            bigrams = list(nltk.bigrams(sentence[1:]))
            self.bigrams.extend(bigrams)

    def generate_trigrams(self):
        for sentence in self.preprocessed_data:
            trigrams = list(nltk.trigrams(sentence[1:]))
            self.trigrams.extend(trigrams)

    def generate_ngrams(self):
        self.generate_unigrams()
        self.generate_bigrams()
        self.generate_trigrams()

    def count_words(self):
        return len(self.unigrams)

```

```python
    def count_vocabulary(self):
        return len(set(self.unigrams))

    def count_sentences(self):
        return len(self.preprocessed_data)

    def print_statistics(self):
        word_count = self.count_words()
        vocab_size = self.count_vocabulary()
        sentence_count = self.count_sentences()

        print("""
        PART 2
        ------

        - Count the words
        - Report the size of the vocabulary
        - report the number of sentences in the dataset

        """)

        print(f"WORD COUNT - Total number of words in the dataset:"
              f" {word_count}")
        print(f"VOCAB SIZE - Total number of unique words in the dataset:"
              f" {vocab_size}")
        print(f"SENTENCE COUNT - Number of sentences in the dataset:"
              f" {sentence_count}")

    def bigram_count_table(self, num_words = 8):
        # Get a random set of unique words < -- Original goal
        # commented out to use the words from the mentioned text
        # unique_words = list(set(self.unigrams))
        # selected_words = random.sample(unique_words, num_words)
        selected_words = ['i', 'want', 'to', 'eat', 'chinese', 'food',
                          'lunch', 'spend']

```

```python
        # Calculate bigram counts
        bigram_counts = Counter(self.bigrams)

        # Create a matrix with bigram counts
        matrix = []
        for w1 in selected_words:
            row = []
            for w2 in selected_words:
                row.append(bigram_counts[(w1, w2)])
            matrix.append(row)

        # Display the matrix in a tabular format
        headers = [''] + selected_words
        table = tabulate(
            matrix, headers = headers, showindex = selected_words,
            tablefmt = "grid", numalign = "right"
            )
        print(table)

    def bigram_probability_table(self, num_words = 8):
        # Get a random set of unique words < -- Original goal
        # commented out to use the words from the mentioned text
        # unique_words = list(set(self.unigrams))
        # selected_words = random.sample(unique_words, num_words)
        selected_words = ['i', 'want', 'to', 'eat', 'chinese', 'food',
                          'lunch', 'spend']

        # Calculate unigram counts
        unigram_counts = Counter(self.unigrams)

        # Calculate bigram counts
        bigram_counts = Counter(self.bigrams)

        # Create a matrix with bigram probabilities
        matrix = []
        for w1 in selected_words:
```

```python
            row = []
            for w2 in selected_words:
                bigram_count = bigram_counts[(w1, w2)]
                unigram_count = unigram_counts[w1]
                probability = bigram_count / unigram_count if unigram_count > 0 else 0
                row.append(probability)
            matrix.append(row)

        # Display the matrix in a tabular format
        headers = [''] + selected_words
        table = tabulate(
            matrix, headers = headers, showindex = selected_words,
            tablefmt = "grid", floatfmt = ".4f", numalign = "right"
            )
        print(table)

```

```python
import nltk
from collections import defaultdict, Counter


class Probability:
    def __init__(self, bigrams, trigrams):
        self.bigrams = bigrams
        self.trigrams = trigrams

    def count_bigrams(self):
        bigram_counts = defaultdict(Counter)
        for bigram in self.bigrams:
            bigram_counts[bigram[0]][bigram[1]] += 1

        return bigram_counts

    def bigram_probabilities(self):
        bigram_counts = self.count_bigrams()

        bigram_probs = {}
        for first_word in bigram_counts:
            total_count = sum(bigram_counts[first_word].values())
            for second_word in bigram_counts[first_word]:
                bigram_probs[(first_word, second_word)] = bigram_counts[first_word][second_word] / total_count

        return bigram_probs

    def count_trigrams(self):
        trigram_counts = defaultdict(Counter)
        for trigram in self.trigrams:
            trigram_counts[(trigram[0], trigram[1])][trigram[2]] += 1

        return trigram_counts

    def trigram_probabilities(self):
```

```python
        bigram_counts = self.count_bigrams()
        trigram_counts = self.count_trigrams()

        trigram_probs = {}
        for bigram in trigram_counts:
            for third_word in trigram_counts[bigram]:
                trigram_probs[bigram + (third_word,)] = trigram_counts[bigram][third_word] / bigram_counts[bigram[0]][bigram[1]]

        return trigram_probs
```

```python
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize


class Preprocessing:
    def __init__(self, dataset_path, disfluencies=None):
        self.dataset_path = dataset_path
        self.raw_text = ""
        self.sentences = []
        self.filtered_sentences = []
        if disfluencies is None:
            self.disfluencies = ["uh", "uhm"]
        else:
            self.disfluencies = disfluencies

    def read_data(self):
        with open(self.dataset_path, "r") as file:
            self.raw_text = file.read()

    def remove_punctuation(self):
        self.raw_text = self.raw_text.replace("'", "")

    def tokenize_sentences(self):
        self.sentences = sent_tokenize(self.raw_text)
        # Print Raw Data
        print("""
        PART 1
        ------

        Load and preprocess the dataset provided:
        - Tokenize the text, keeping only actual words while removing disfluencies such as “uh”
        and “uhm”
        - Add special tokens to indicate the beginning of each sentence

        """)
        print("\n------ Sampled Raw Data - START------\n")
```

```
        for i in range(5):
            print(self.sentences[i])
        print("\n------ END ------\n")

    def tokenize_words(self):
        self.filtered_sentences = [word_tokenize(sentence) for sentence in self.sentences]

    def contains_non_alpha_chars(self, token):
        for char in token:
            if not char.isalpha():
                return True
        return False

    def remove_disfluencies(self):
        for i, sentence in enumerate(self.filtered_sentences):
            self.filtered_sentences[i] = [
                token.strip("_") for token in sentence
                if token.lower() not in self.disfluencies
                and not self.contains_non_alpha_chars(token)
            ]

    def add_start_tokens(self, start_token):
        for sentence in self.filtered_sentences:
            sentence.insert(0, start_token)

        print("\n------ Sampled Processed Data - START------\n")
        for i in range(5):
            print(self.filtered_sentences[i])
        print("\n------ END ------\n")

    def preprocess(self, start_token="</s>"):
        nltk.download("punkt")
        self.read_data()
        self.remove_punctuation()
        self.tokenize_sentences()
        self.tokenize_words()
```

```
        self.remove_disfluencies()
        self.add_start_tokens(start_token)

```