```
# Text preprocessing
import re
import string

# Tokenization
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize

# Stopwords (optional)
from nltk.corpus import stopwords

# Counting N-grams
from collections import Counter, defaultdict
from nltk.util import ngrams

# Numerical operations
import numpy as np

# Data handling
import pandas as pd

# Download required resources (run once)
nltk.download('punkt')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
# Example: load from text file
with open("dataset.txt", "r", encoding="utf-8") as file:
    corpus = file.read()

print(corpus[:500])  # display sample
```

```
This is a sample sentence for the dataset. It contains multiple words and punctuation. Another sentence
This is the second line of text. The quick brown fox jumps over the lazy dog.
```

```
def preprocess_text(text):
    # Lowercase
    text = text.lower()

    # Remove numbers
    text = re.sub(r'\d+', '', text)

    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))

    # Sentence tokenization
    sentences = sent_tokenize(text)

    processed_sentences = []

    for sentence in sentences:
        words = word_tokenize(sentence)

        # Add start and end tokens
        words = ['<s>'] + words + ['</s>']

        processed_sentences.append(words)

    return processed_sentences
```

```
processed_data = preprocess_text(corpus)
```

```
def build_unigram(sentences):
    words = [word for sentence in sentences for word in sentence]
    unigram_counts = Counter(words)
    total_words = sum(unigram_counts.values())
    return unigram_counts, total_words

unigram_counts, total_unigrams = build_unigram(processed_data)
```

```
def build_bigram(sentences):
    bigram_counts = Counter()
    for sentence in sentences:
        bigram_counts.update(ngrams(sentence, 2))
    return bigram_counts

bigram_counts = build_bigram(processed_data)
```

```
def build_trigram(sentences):
    trigram_counts = Counter()
    for sentence in sentences:
        trigram_counts.update(ngrams(sentence, 3))
    return trigram_counts

trigram_counts = build_trigram(processed_data)
```

```
def unigram_prob(sentence, unigram_counts, total_words, vocab_size):
    prob = 1
    for word in sentence:
        prob *= (unigram_counts[word] + 1) / (total_words + vocab_size)
    return prob
```

```
def bigram_prob(sentence, bigram_counts, unigram_counts, vocab_size):
    prob = 1
    bigrams = list(ngrams(sentence, 2))

    for w1, w2 in bigrams:
        prob *= (bigram_counts[(w1, w2)] + 1) / (unigram_counts[w1] + vocab_size)
    return prob
```

```
def trigram_prob(sentence, trigram_counts, bigram_counts, vocab_size):
    prob = 1
    trigrams = list(ngrams(sentence, 3))

    for w1, w2, w3 in trigrams:
        prob *= (trigram_counts[(w1, w2, w3)] + 1) / (bigram_counts[(w1, w2)] + vocab_size)
    return prob
```

```
test_sentences = [
    "the boy was happy",
    "she opened the door",
    "they went to school",
    "he said hello",
    "it was a sunny day"
]
```

```
def perplexity(prob, N):
```

```
    return pow(prob, -1/N)
```