```
import nltk  # for tokenization (sentences & words)
import re    # for text cleaning using regex
import math  # for probability & perplexity calculation
import numpy as np  # numerical operations
import pandas as pd # creating tables
from collections import Counter, defaultdict  # counting n-grams
```

```
corpus = """
Technology is growing rapidly in the modern world. Artificial intelligence and ma
are transforming industries such as healthcare, education, finance, and transpor
Natural language processing helps computers understand human language.
Deep learning models require large datasets for training.

Sports play an important role in maintaining physical and mental health.
Football, cricket, and athletics are popular sports worldwide.
Regular exercise improves concentration and reduces stress.

Politics influences economic growth and public welfare.
Democracy allows citizens to choose their leaders.
Good governance promotes transparency and accountability.

Health awareness is essential for a balanced lifestyle.
Eating nutritious food and maintaining hygiene prevents diseases.
Mental health is as important as physical health.

Technology, sports, politics, and health together shape society.
""" * 30   # repeated to exceed 1500 words and create a sufficiently large corpu
```

```
print(corpus[:500]) # Helps verify that the dataset is loaded correctly by prir
```

```

Technology is growing rapidly in the modern world. Artificial intelligence and m
are transforming industries such as healthcare, education, finance, and transpor
Natural language processing helps computers understand human language.
Deep learning models require large datasets for training.

Sports play an important role in maintaining physical and mental health.
Football, cricket, and athletics are popular sports worldwide.
Regular exercise improves concentration and reduc
```

```
def preprocess_text(text):  # This function cleans and prepares the raw text
    # convert all text to lowercase to ensure consistency
    text = text.lower()

    # remove numbers and punctuation, keeping only lowercase letters and spaces
    text = re.sub(r'[^a-z\s]', '', text)
    # split the cleaned text into individual sentences based on newline charact
    sentences = text.split('\n')

    processed_sentences = [] # Initialize a list to hold processed sentences
    for sentence in sentences:
        words = sentence.split() # Split each sentence into words
        if len(words) > 0: # Ensure the sentence is not empty after splitting
            # add start (<s>) and end (</s>) tokens to each sentence for model
            processed_sentences.append(['<s>'] + words + ['</s>'])

    return processed_sentences
```

```
# Apply the preprocessing function to the raw corpus text
sentences = preprocess_text(corpus)
print(sentences[0]) # Print the first processed sentence to verify the output
```

```
['<s>', 'technology', 'is', 'growing', 'rapidly', 'in', 'the', 'modern', 'world'
```

```
# Count the frequency of each individual word (unigram) across all sentences
unigram_counts = Counter()  # Initialize a Counter object for unigrams
for sentence in sentences:
    unigram_counts.update(sentence) # Update counts for words in each sentence

total_unigrams = sum(unigram_counts.values()) # Calculate the total number of u
```

```
# Count the frequency of consecutive word pairs (bigrams)
bigram_counts = Counter()  # Initialize a Counter for bigrams
unigram_context = Counter() # Initialize a Counter for the first word of each bi

for sentence in sentences:
    for i in range(len(sentence) - 1): # Iterate up to the second-to-last word
        bigram = (sentence[i], sentence[i+1]) # Create a tuple for the bigram
        bigram_counts[bigram] += 1 # Increment bigram count
        unigram_context[sentence[i]] += 1 # Increment count for the context word
```

```
# Count the frequency of three consecutive words (trigrams)
trigram_counts = Counter() # Initialize a Counter for trigrams
bigram_context = Counter() # Initialize a Counter for the first two words of ea

for sentence in sentences:
    for i in range(len(sentence) - 2): # Iterate up to the third-to-last word
        trigram = (sentence[i], sentence[i+1], sentence[i+2]) # Create a tuple
        trigram_counts[trigram] += 1 # Increment trigram count
        bigram_context[(sentence[i], sentence[i+1])] += 1 # Increment count for
```

```
def unigram_probability(sentence):    # Calculates the probability of a sentenc
    prob = 1 # Initialize probability to 1
    V = len(unigram_counts) # Vocabulary size, used for Laplace smoothing
    for word in sentence:
        # Apply Laplace smoothing: (count(word) + 1) / (total_words + V)
        prob *= (unigram_counts[word] + 1) / (total_unigrams + V)
    return prob


def bigram_probability(sentence): # Calculates the probability of a sentence us
    prob = 1 # Initialize probability to 1
    V = len(unigram_counts) # Vocabulary size, used for Laplace smoothing
    for i in range(len(sentence) - 1):
        bigram = (sentence[i], sentence[i+1]) # Get the current bigram
        # Apply Laplace smoothing: (count(bigram) + 1) / (count(first_word_of_b
        prob *= (bigram_counts[bigram] + 1) / (unigram_context[sentence[i]] + V
    return prob


def trigram_probability(sentence): # Calculates the probability of a sentence u
    prob = 1 # Initialize probability to 1
    V = len(unigram_counts) # Vocabulary size, used for Laplace smoothing
    for i in range(len(sentence) - 2):
        trigram = (sentence[i], sentence[i+1], sentence[i+2]) # Get the current
        # Apply Laplace smoothing: (count(trigram) + 1) / (count(first_two_word
        prob *= (trigram_counts[trigram] + 1) / (bigram_context[(sentence[i], s
    return prob
```

```
test_sentences = [ # Define a list of test sentences, each split into words anc
    "<s> technology improves healthcare </s>".split(),
    "<s> sports improve health </s>".split(),
    "<s> democracy supports growth </s>".split(),
    "<s> artificial intelligence grows fast </s>".split(),
    "<s> nutrition improves mental health </s>".split()
]
```

```
for s in test_sentences: # Iterate through each test sentence
    print("Sentence:", " ".join(s)) # Print the original sentence
    print("Unigram:", unigram_probability(s)) # Calculate and print unigram prc
    print("Bigram:", bigram_probability(s))   # Calculate and print bigram prob
    print("Trigram:", trigram_probability(s)) # Calculate and print trigram prc
    print() # Print a newline for better formatting
```

```
Sentence: <s> technology improves healthcare </s>
Unigram: 6.021470510408388e-09
Bigram: 5.5374001452432835e-08
Trigram: 8.230452674897121e-07

Sentence: <s> sports improve health </s>
Unigram: 1.411457093593109e-09
Bigram: 9.536633583474542e-07
Trigram: 1.02880658436214e-06
```

```
Sentence: <s> democracy supports growth </s>
Unigram: 9.871263131817029e-11
Bigram: 4.690147664003873e-08
Trigram: 1.02880658436214e-06

Sentence: <s> artificial intelligence grows fast </s>
Unigram: 2.2232574621209523e-14
Bigram: 5.211275182226526e-10
Trigram: 1.1431184270690446e-08

Sentence: <s> nutrition improves mental health </s>
Unigram: 6.605943381805191e-12
Bigram: 1.8765633825546682e-08
Trigram: 2.83493369913123e-07
```

```
def perplexity(sentence, model): # Function to calculate the perplexity of a se
    N = len(sentence) # N is the number of words in the sentence
    log_prob = 0 # Initialize log probability

    if model == "unigram": # If unigram model is selected
        for word in sentence:
            # Accumulate the log probability of each word based on the unigram
            log_prob += math.log(unigram_probability([word]))
    elif model == "bigram": # If bigram model is selected
        # Use the pre-calculated bigram probability for the entire sentence
        log_prob = math.log(bigram_probability(sentence))
    elif model == "trigram": # If trigram model is selected
        # Use the pre-calculated trigram probability for the entire sentence
        log_prob = math.log(trigram_probability(sentence))

    # Perplexity formula: exp(- (1/N) * log_prob_of_sentence)
    return math.exp(-log_prob / N)
```

```
for s in test_sentences: # Iterate through each test sentence
    print("Sentence:", " ".join(s)) # Print the original sentence
    print("Unigram Perplexity:", perplexity(s, "unigram")) # Calculate and prir
    print("Bigram Perplexity:", perplexity(s, "bigram"))   # Calculate and prir
    print("Trigram Perplexity:", perplexity(s, "trigram")) # Calculate and prir
    print() # Print a newline for better formatting
```

```
Sentence: <s> technology improves healthcare </s>
Unigram Perplexity: 44.06152123963561
Bigram Perplexity: 28.270846883945705
Trigram Perplexity: 16.47840814959176

Sentence: <s> sports improve health </s>
Unigram Perplexity: 58.89337781117068
Bigram Perplexity: 16.00003676939737
Trigram Perplexity: 15.759166826422602

Sentence: <s> democracy supports growth </s>
```

```
Unigram Perplexity: 100.25948148909517
Bigram Perplexity: 29.225550255449416
Trigram Perplexity: 15.759166826422602

Sentence: <s> artificial intelligence grows fast </s>
Unigram Perplexity: 188.58263812685047
Bigram Perplexity: 35.25136991157913
Trigram Perplexity: 21.069365756459884

Sentence: <s> nutrition improves mental health </s>
Unigram Perplexity: 73.00358576081786
Bigram Perplexity: 19.398707458264983
Trigram Perplexity: 12.337945539935726
```