```
# Natural Language Toolkit for NLP tasks like tokenization
import nltk

# Regular expressions for cleaning text
import re

# Mathematical functions for probability and perplexity
import math

# Numerical computations
import numpy as np

# Data handling and table display
import pandas as pd

# Counter for counting words and N-grams
from collections import Counter

# Tokenizers for sentences and words
from nltk.tokenize import sent_tokenize, word_tokenize

# Download required NLTK resources
nltk.download('punkt')
nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
True
```

```
import nltk

# Download required tokenizer resources
nltk.download('punkt')
nltk.download('punkt_tab')
```

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

```
import os

corpus_path = "/content/corpus.txt"

# Check if corpus.txt exists, if not, create a dummy one
if not os.path.exists(corpus_path):
    print(f"'{corpus_path}' not found. Creating a dummy corpus for demonstration.")
    dummy_text_paragraph = (
        "Natural language processing, or NLP, is a fascinating field of artificial intelligence that focuses on the interac
        "It involves various tasks such as understanding, interpreting, and generating human language. "
        "Language models are a core component of NLP, designed to predict the next word in a sequence or to understand the
        "These models are crucial for applications like machine translation, spam detection, sentiment analysis, and conver
        "Training these models requires vast amounts of text data to learn patterns, grammar, and semantics. "
        "The complexity of human language, with its nuances, ambiguities, and ever-evolving nature, makes NLP a challenging
        "Different types of language models exist, including n-gram models, neural network-based models like recurrent neur
        "Each approach has its strengths and weaknesses, and the choice often depends on the specific task and available co
        "The goal is to enable computers to comprehend and process human language at a level approaching human understandin
        "This continuous advancement in NLP is transforming how we interact with technology and access information. "
    )
    # Repeat the paragraph to ensure the corpus has at least 1500 words
    dummy_text = (dummy_text_paragraph + " ") * 10 # 180 words/paragraph * 10 = 1800 words
    with open(corpus_path, "w", encoding="utf-8") as file:
        file.write(dummy_text.strip())
    print(f"Dummy corpus created at '{corpus_path}' with {len(dummy_text.split())} words.")

# Load text corpus (must contain at least 1500 words)
with open(corpus_path, "r", encoding="utf-8") as file:
    text = file.read()

# Display sample text
print(text[:500])
```

```
'/content/corpus.txt' not found. Creating a dummy corpus for demonstration.
Dummy corpus created at '/content/corpus.txt' with 1870 words.
Natural language processing, or NLP, is a fascinating field of artificial intelligence that focuses on the interaction betwe
```

```
def preprocess_text(text):
    # Tokenize text into sentences first to preserve sentence boundaries
    sentences = sent_tokenize(text)

    processed_sentences = []

    for sentence in sentences:
        # Convert sentence to lowercase
        sentence = sentence.lower()

        # Remove punctuation and numbers from the sentence
        # Now this will apply to individual sentences without affecting tokenization
        sentence = re.sub(r'[^a-z\s]', '', sentence)

        # Tokenize sentence into words
        words = word_tokenize(sentence)

        # Add start and end tokens
        words = ['<s>'] + words + ['</s>']
        processed_sentences.append(words)

    return processed_sentences
```

```
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^a-z\s.]', '', text)

    sentences = text.split('.')
    processed = []

    for sentence in sentences:
        words = sentence.strip().split()

        # CHANGE IS HERE 👇
        # Keep only sentences with at least 3 words
        if len(words) >= 3:
            processed.append(['<s>'] + words + ['</s>'])

    return processed
```

```
def build_unigram(data):
    return Counter([word for sentence in data for word in sentence])
```

```
def build_bigram(data):
    return Counter([
        (sentence[i], sentence[i+1])
        for sentence in data
        for i in range(len(sentence)-1)
    ])
```

```
def build_trigram(data):
    return Counter([
        (sentence[i], sentence[i+1], sentence[i+2])
        for sentence in data
        for i in range(len(sentence)-2)
    ])
```

```
# Preprocess the text to create training data
train_data = preprocess_text(text)

# Build models
unigrams = build_unigram(train_data)
bigrams = build_bigram(train_data)
trigrams = build_trigram(train_data)

vocab_size = len(unigrams)
```

```
# Unigram table
pd.DataFrame(unigrams.most_common(10), columns=["Word", "Count"])
```

	Word	Count
0	and	110
1	<s>	100
2	</s>	100
3	of	80
4	language	70
5	the	70
6	a	60
7	models	60
8	human	50
9	to	50

```
# Bigram table
pd.DataFrame(bigrams.most_common(10), columns=["Bigram", "Count"])
```

	Bigram	Count
0	(human, language)	40
1	(nlp, is)	20
2	(on, the)	20
3	(language, </s>)	20
4	(language, models)	20
5	(models, are)	20
6	(a, sequence)	20
7	(these, models)	20
8	(<s>, the)	20
9	(<s>, natural)	10

```
# Trigram table
pd.DataFrame(trigrams.most_common(10), columns=["Trigram", "Count"])
```

	Trigram	Count
0	(human, language, </s>)	20
1	(<s>, natural, language)	10
2	(natural, language, processing)	10
3	(language, processing, or)	10
4	(processing, or, nlp)	10
5	(or, nlp, is)	10
6	(nlp, is, a)	10
7	(is, a, fascinating)	10
8	(a, fascinating, field)	10
9	(fascinating, field, of)	10

```
def unigram_probability(word):
    return (unigrams[word] + 1) / (sum(unigrams.values()) + vocab_size)

def bigram_probability(w1, w2):
    return (bigrams[(w1, w2)] + 1) / (unigrams[w1] + vocab_size)

def trigram_probability(w1, w2, w3):
    return (trigrams[(w1, w2, w3)] + 1) / (bigrams[(w1, w2)] + vocab_size)
```

```
def sentence_probability(sentence, model):
    words = ['<s>'] + word_tokenize(sentence.lower()) + ['</s>']
    probability = 1
```

```
    if model == "unigram":
        for w in words:
            probability *= unigram_probability(w)

    elif model == "bigram":
        for i in range(len(words)-1):
            probability *= bigram_probability(words[i], words[i+1])

    elif model == "trigram":
        for i in range(len(words)-2):
            probability *= trigram_probability(words[i], words[i+1], words[i+2])

    return probability
```

```
sentences = [
    "language models are important",
    "this is a simple test",
    "n gram models predict words",
    "the system learns probabilities",
    "this sentence is unseen"
]
```

```
# Check if vocab_size is zero, indicating empty training data
if vocab_size == 0:
    print("Error: Vocabulary size is 0. This typically means the training data (train_data) is empty.")
    print("Please check the data splitting logic in cell 'ngGwxXL99JET' and ensure 'train_data' contains sentences.")
else:
    for s in sentences:
        print("\nSentence:", s)
        # If vocab_size is not zero, the denominators in probability functions should be non-zero
        # as they all add `vocab_size` to their respective counts.
        print("Unigram Probability:", sentence_probability(s, "unigram"))
        print("Bigram Probability:", sentence_probability(s, "bigram"))
        print("Trigram Probability:", sentence_probability(s, "trigram"))
```

```
Sentence: language models are important
Unigram Probability: 8.295509758767346e-12
Bigram Probability: 3.297345090448539e-08
Trigram Probability: 3.41040207865416e-07

Sentence: this is a simple test
Unigram Probability: 8.643413488492583e-16
Bigram Probability: 8.890898066525309e-12
Trigram Probability: 2.8093278463648836e-11

Sentence: n gram models predict words
Unigram Probability: 3.0670176894651093e-16
Bigram Probability: 9.280061954959077e-13
Trigram Probability: 3.2768000000000001e-11

Sentence: the system learns probabilities
Unigram Probability: 6.475807774213386e-15
Bigram Probability: 2.450598290598291e-10
Trigram Probability: 3.5310344827586205e-09

Sentence: this sentence is unseen
Unigram Probability: 3.110211902826429e-14
Bigram Probability: 1.4952874020974382e-10
Trigram Probability: 3.792592592592593e-09
```

```
def perplexity(sentence, model):
    words = ['<s>'] + word_tokenize(sentence.lower()) + ['</s>']
    N = len(words)
    log_prob = 0

    if model == "unigram":
        for w in words:
            log_prob += math.log(unigram_probability(w))

    elif model == "bigram":
        for i in range(len(words)-1):
            log_prob += math.log(bigram_probability(words[i], words[i+1]))

    elif model == "trigram":
        for i in range(len(words)-2):
            log_prob += math.log(trigram_probability(words[i], words[i+1], words[i+2]))

    return math.exp(-log_prob / N)
```

```python
for s in sentences:
    print("\nSentence:", s)
    print("Unigram Perplexity:", perplexity(s, "unigram"))
    print("Bigram Perplexity:", perplexity(s, "bigram"))
    print("Trigram Perplexity:", perplexity(s, "trigram"))
```

```
Sentence: language models are important
Unigram Perplexity: 70.28448823021849
Bigram Perplexity: 17.659264023855453
Trigram Perplexity: 11.963706108705308

Sentence: this is a simple test
Unigram Perplexity: 141.87376743134757
Bigram Perplexity: 37.90723020677107
Trigram Perplexity: 32.161960251288725

Sentence: n gram models predict words
Unigram Perplexity: 164.50662345968757
Bigram Perplexity: 52.35055659376793
Trigram Perplexity: 31.462473766045687

Sentence: the system learns probabilities
Unigram Perplexity: 231.62441841732593
Bigram Perplexity: 39.974954579308445
Trigram Perplexity: 25.626128932431712

Sentence: this sentence is unseen
Unigram Perplexity: 178.3206629838152
Bigram Perplexity: 43.405612364271136
Trigram Perplexity: 25.322736424086596
```