```
# ============================================================
# STEP 2 — Import Required Libraries
# ============================================================

# nltk is used for tokenization, stopwords, and sentence splitting
import nltk

# collections helps in counting words and n-grams efficiently
from collections import Counter, defaultdict

# numpy is used for mathematical operations like log and power
import numpy as np

# pandas is used to display frequency tables neatly
import pandas as pd

# string is used to remove punctuation
import string

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

from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
# ============================================================
# STEP 3 — Load Dataset
# ============================================================

# Sample dataset (you may replace this with a text file or larger corpus)
# Ensure total words >= 1500 for lab requirement

text_corpus = """
Natural language processing is a field of artificial intelligence.
It focuses on the interaction between computers and humans.
Language models are used to predict the probability of word sequences.
They are essential in applications like machine translation and speech recognition.
Statistical language models use probability to model language.
N-gram models are simple but effective language models.
They use previous words to predict the next word.
Unigram models consider one word at a time.
Bigram models consider pairs of words.
Trigram models consider sequences of three words.
Smoothing techniques help handle unseen words.
Laplace smoothing is a simple smoothing method.
Perplexity measures how well a model predicts text.
Lower perplexity indicates better performance.
""" * 120   # repeated to exceed 1500 words

print("Sample text:")
print(text_corpus[:500])
```

```
Sample text:

Natural language processing is a field of artificial intelligence.
It focuses on the interaction between computers and humans.
Language models are used to predict the probability of word sequences.
They are essential in applications like machine translation and speech recognition.
Statistical language models use probability to model language.
N-gram models are simple but effective language models.
They use previous words to predict the next word.
Unigram models consider one word at a time.
Bigr
```

```
# ============================================================
# STEP 4 — Text Preprocessing
# ============================================================

nltk.download('punkt_tab') # Add this line to download the missing resource

stop_words = set(stopwords.words('english'))
```

```
def preprocess_text(text):
    """
    This function performs:
    1. Lowercasing
    2. Sentence tokenization
    3. Word tokenization
    4. Removal of punctuation and numbers
    5. Optional stopword removal
    6. Adding start and end tokens
    """
    sentences = sent_tokenize(text.lower())
    processed_sentences = []

    for sent in sentences:
        # Remove punctuation
        sent = sent.translate(str.maketrans('', '', string.punctuation))

        # Tokenize words
        words = word_tokenize(sent)

        # Remove numbers and stopwords
        words = [w for w in words if w.isalpha() and w not in stop_words]

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

        processed_sentences.append(words)

    return processed_sentences


processed_data = preprocess_text(text_corpus)

print("Sample processed sentence:")
print(processed_data[0])
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
Sample processed sentence:
['<s>', 'natural', 'language', 'processing', 'field', 'artificial', 'intelligence', '</s>']
```

```
# ============================================================
# STEP 5 — Build N-Gram Models
# ============================================================

def build_ngrams(sentences, n):
    """
    Builds n-grams from tokenized sentences.
    Returns a list of n-grams.
    """
    ngrams = []
    for sent in sentences:
        for i in range(len(sent) - n + 1):
            ngrams.append(tuple(sent[i:i+n]))
    return ngrams


# Build unigram, bigram, trigram
unigrams = build_ngrams(processed_data, 1)
bigrams = build_ngrams(processed_data, 2)
trigrams = build_ngrams(processed_data, 3)

# Count frequencies
unigram_counts = Counter(unigrams)
bigram_counts = Counter(bigrams)
trigram_counts = Counter(trigrams)

# Vocabulary size
vocab = set([word[0] for word in unigrams])
V = len(vocab)

print("Vocabulary size:", V)
```

```
Vocabulary size: 58
```

```
# Display Unigram Frequency Table
unigram_df = pd.DataFrame(unigram_counts.items(), columns=['Word', 'Count'])
unigram_df.head()
```

	Word	Count
0	(<s>,)	1680
1	(natural,)	120
2	(language,)	600
3	(processing,)	120
4	(field,)	120

Next steps: Generate code with unigram_df New interactive sheet

```
# ============================================================
# STEP 6 — Add-One (Laplace) Smoothing
# ============================================================

# Smoothing is required to handle unseen words or n-grams.
# Without smoothing, unseen sequences get zero probability,
# which makes sentence probability zero and perplexity infinite.

def unigram_probability(word):
    return (unigram_counts[(word,)] + 1) / (sum(unigram_counts.values()) + V)


def bigram_probability(bigram):
    prev_word = (bigram[0],)
    return (bigram_counts[bigram] + 1) / (unigram_counts[prev_word] + V)


def trigram_probability(trigram):
    prev_bigram = (trigram[0], trigram[1])
    return (trigram_counts[trigram] + 1) / (bigram_counts[prev_bigram] + V)
```

```
# ============================================================
# STEP 7 — Sentence Probability Calculation
# ============================================================

test_sentences = [
    "language models predict words",
    "smoothing helps unseen words",
    "trigram models use context",
    "perplexity measures performance",
    "natural language processing models"
]

def sentence_probability(sentence, n):
    tokens = ['<s>'] + sentence.lower().split() + ['</s>']
    prob = 1

    if n == 1:
        for word in tokens:
            prob *= unigram_probability(word)

    elif n == 2:
        for i in range(len(tokens)-1):
            prob *= bigram_probability((tokens[i], tokens[i+1]))

    elif n == 3:
        for i in range(len(tokens)-2):
            prob *= trigram_probability((tokens[i], tokens[i+1], tokens[i+2]))

    return prob


for sent in test_sentences:
    print("\nSentence:", sent)
    print("Unigram Probability:", sentence_probability(sent, 1))
    print("Bigram Probability:", sentence_probability(sent, 2))
    print("Trigram Probability:", sentence_probability(sent, 3))
```

```

Sentence: language models predict words
Unigram Probability: 3.0485963515003315e-08
Bigram Probability: 9.577435414367187e-08
Trigram Probability: 4.834294449667436e-07

Sentence: smoothing helps unseen words
Unigram Probability: 1.093213503188477e-11
Bigram Probability: 1.309850158884881e-06
```

```
Trigram Probability: 1.1352444269443752e-06

Sentence: trigram models use context
Unigram Probability: 1.2760442870044732e-11
Bigram Probability: 3.6894849422987455e-07
Trigram Probability: 3.699111054088414e-07

Sentence: perplexity measures performance
Unigram Probability: 2.4340704090484448e-08
Bigram Probability: 0.00010795681670488084
Trigram Probability: 6.584417676277377e-05

Sentence: natural language processing models
Unigram Probability: 3.850423925114203e-09
Bigram Probability: 6.587937559224133e-06
Trigram Probability: 4.4759243754469804e-05
```

```python
# ============================================================
# STEP 8 — Perplexity Calculation
# ============================================================

def perplexity(sentence, n):
    tokens = ['<s>'] + sentence.lower().split() + ['</s>']
    N = len(tokens)
    log_prob = 0

    if n == 1:
        for word in tokens:
            log_prob += np.log(unigram_probability(word))

    elif n == 2:
        for i in range(len(tokens)-1):
            log_prob += np.log(bigram_probability((tokens[i], tokens[i+1])))

    elif n == 3:
        for i in range(len(tokens)-2):
            log_prob += np.log(trigram_probability((tokens[i], tokens[i+1], tokens[i+2])))

    return np.exp(-log_prob / N)


for sent in test_sentences:
    print("\nSentence:", sent)
    print("Unigram Perplexity:", perplexity(sent, 1))
    print("Bigram Perplexity:", perplexity(sent, 2))
    print("Trigram Perplexity:", perplexity(sent, 3))
```

```
Sentence: language models predict words
Unigram Perplexity: 17.891634286673145
Bigram Perplexity: 14.783994580338305
Trigram Perplexity: 11.287847934750829

Sentence: smoothing helps unseen words
Unigram Perplexity: 67.12472231542692
Bigram Perplexity: 9.560113840272153
Trigram Perplexity: 9.790805810248461

Sentence: trigram models use context
Unigram Perplexity: 65.4167617712738
Bigram Perplexity: 11.807892109876901
Trigram Perplexity: 11.802765313732346

Sentence: perplexity measures performance
Unigram Perplexity: 33.32217743992871
Bigram Perplexity: 6.213695771586399
Trigram Perplexity: 6.859564207433971

Sentence: natural language processing models
Unigram Perplexity: 25.25894702834624
Bigram Perplexity: 7.303680291311
Trigram Perplexity: 5.307046301260262
```

Start coding or generate with AI.