```
# Text cleaning
import re

# NLP tools
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords

# Counting N-grams
from collections import Counter

# Math operations
import math

# Data handling
import pandas as pd
import numpy as np
```

```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('gutenberg')
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 stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Package gutenberg is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
True
```

```
from nltk.corpus import gutenberg

# Load dataset
corpus = gutenberg.raw('austen-emma.txt')

# Show sample text
print(corpus[:700])
```

```
[Emma by Jane Austen 1816]

VOLUME I

CHAPTER I


Emma Woodhouse, handsome, clever, and rich, with a comfortable home
and happy disposition, seemed to unite some of the best blessings
of existence; and had lived nearly twenty-one years in the world
with very little to distress or vex her.

She was the youngest of the two daughters of a most affectionate,
indulgent father; and had, in consequence of her sister's marriage,
been mistress of his house from a very early period.  Her mother
had died too long ago for her to have more than an indistinct
remembrance of her caresses; and her place had been supplied
by an excellent woman as governess, who had fallen little short
of a mother in affectio
```

```
from nltk.tokenize import sent_tokenize
sentences = sent_tokenize(corpus)

split_index = int(0.8 * len(sentences))
train_sentences = sentences[:split_index]
test_sentences = sentences[split_index:]

print("Training sentences:", len(train_sentences))
print("Testing sentences:", len(test_sentences))
```

```
Training sentences: 5994
Testing sentences: 1499
```

```
sentences = sent_tokenize(corpus)

split_index = int(0.8 * len(sentences))
train_sentences = sentences[:split_index]
test_sentences = sentences[split_index:]
```

```
print("Training sentences:", len(train_sentences))
print("Testing sentences:", len(test_sentences))
```

```
Training sentences: 5994
Testing sentences: 1499
```

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

def preprocess(sentences):
    processed_sentences = []
    stop_words = set(stopwords.words('english'))
    for sentence in sentences:
        # Tokenize words, convert to lowercase, and remove stop words and non-alphabetic characters
        words = [word.lower() for word in word_tokenize(sentence) if word.isalpha() and word.lower() not in stop_words]
        processed_sentences.append(words)
    return processed_sentences

train_data = preprocess(train_sentences)
test_data = preprocess(test_sentences)

print(train_data[0])
```

```
['emma', 'jane', 'austen', 'volume', 'chapter', 'emma', 'woodhouse', 'handsome', 'clever', 'rich', 'comfortable', 'home', 'h
```

```
def build_ngrams(data, n):
    ngrams = []
    for sentence in data:
        for i in range(len(sentence) - n + 1):
            ngrams.append(tuple(sentence[i:i+n]))
    return ngrams
```

```
from collections import Counter
unigrams = build_ngrams(train_data, 1)
bigrams = build_ngrams(train_data, 2)
trigrams = build_ngrams(train_data, 3)

uni_counts = Counter(unigrams)
bi_counts = Counter(bigrams)
tri_counts = Counter(trigrams)
```

```
import pandas as pd
pd.DataFrame(uni_counts.most_common(10), columns=["Unigram", "Count"])
```

	Unigram	Count
0	(emma,)	624
1	(could,)	621
2	(would,)	610
3	(miss,)	486
4	(must,)	423
5	(said,)	386
6	(much,)	378
7	(harriet,)	363
8	(weston,)	354
9	(one,)	347

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

	Bigram	Count
0	(miss, woodhouse)	133
1	(frank, churchill)	108
2	(every, thing)	97
3	(every, body)	92
4	(miss, fairfax)	91
5	(miss, bates)	87
6	(jane, fairfax)	85
7	(young, man)	79
8	(miss, smith)	55
9	(great, deal)	53

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

	Trigram	Count
0	(dear, miss, woodhouse)	19
1	(poor, miss, taylor)	10
2	(said, frank, churchill)	10
3	(miss, woodhouse, would)	8
4	(fine, young, man)	7
5	(said, john, knightley)	7
6	(frank, churchill, miss)	6
7	(miss, bates, miss)	5
8	(miss, woodhouse, must)	5
9	(every, body, else)	5

```
vocab = set([w[0] for w in uni_counts])
V = len(vocab)

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

```
Vocabulary size: 6107
```

```
def unigram_prob(word):
    return (uni_counts.get((word,), 0) + 1) / (sum(uni_counts.values()) + V)

def bigram_prob(bigram):
    return (bi_counts.get(bigram, 0) + 1) / (uni_counts.get((bigram[0],), 0) + V)

def trigram_prob(trigram):
    return (tri_counts.get(trigram, 0) + 1) / (bi_counts.get(trigram[:2], 0) + V)
```

```
test_sentences_samples = [
    "emma was happy with her friend",
    "the house was full of people",
    "she loved her family",
    "he spoke with confidence",
    "the weather was pleasant"
]
```

```
def sentence_probability(sentence, model="unigram"):
    words = ['<s>'] + sentence.lower().split() + ['</s>']
    prob = 1

    if model == "unigram":
        for w in words:
            prob *= unigram_prob(w)

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

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

    return prob
```

```
for s in test_sentences_samples:
    print(s)
    print("Unigram:", sentence_probability(s, "unigram"))
    print("Bigram :", sentence_probability(s, "bigram"))
    print("Trigram:", sentence_probability(s, "trigram"))
    print()
```

```
emma was happy with her friend
Unigram: 4.792415922925862e-32
Bigram : 2.7594516248550867e-27
Trigram: 1.927669074355202e-23

the house was full of people
Unigram: 1.7291036649916503e-33
Bigram : 3.0519245297139922e-27
Trigram: 1.927669074355202e-23

she loved her family
Unigram: 2.284765243377151e-26
Bigram : 1.1623507771869188e-19
Trigram: 7.189328365149164e-16

he spoke with confidence
Unigram: 5.631914325411465e-27
Bigram : 1.170893203146739e-19
Trigram: 7.189328365149164e-16

the weather was pleasant
Unigram: 2.5386280481968342e-26
Bigram : 1.1644176278191582e-19
Trigram: 7.189328365149164e-16
```

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

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

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

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

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

```
import math
for s in test_sentences_samples:
    print(s)
    print("Unigram Perplexity:", perplexity(s,"unigram"))
    print("Bigram Perplexity :", perplexity(s,"bigram"))
    print("Trigram Perplexity:", perplexity(s,"trigram"))
    print()
```

```
emma was happy with her friend
Unigram Perplexity: 8221.11431614011
Bigram Perplexity : 2088.8014095275066
Trigram Perplexity: 690.8296039321608

the house was full of people
Unigram Perplexity: 12452.953633695983
Bigram Perplexity : 2062.662990395879
Trigram Perplexity: 690.8296039321608

she loved her family
Unigram Perplexity: 18772.685426474964
Bigram Perplexity : 1431.4531270462016
Trigram Perplexity: 334.1067732064736

he spoke with confidence
Unigram Perplexity: 23707.76753139801
Bigram Perplexity : 1429.7072489861407
Trigram Perplexity: 334.1067732064736
```

```
the weather was pleasant
Unigram Perplexity: 18445.912930391874
Bigram Perplexity : 1431.0293400463852
Trigram Perplexity: 334.1067732064736
```