

## ✓ Lab 02: Basic NLP Preprocessing Techniques

**Course:** ITAI 2373 - Natural Language Processing

**Module:** 02 - Text Preprocessing

**Duration:** 2-3 hours

**Student Name:** \_\_\_\_

**Date:** \_\_\_\_

---

### Learning Objectives

By completing this lab, you will:

1. Understand the critical role of preprocessing in NLP pipelines
2. Master fundamental text preprocessing techniques
3. Compare different libraries and their approaches
4. Analyze the effects of preprocessing on text data
5. Build a complete preprocessing pipeline
6. Load and work with different types of text datasets

## ✓ Introduction to NLP Preprocessing

Natural Language Processing (NLP) preprocessing refers to the initial steps taken to clean and transform raw text data into a format that's more suitable for analysis by machine learning algorithms.

Why is preprocessing crucial?

1. **Standardization:** Ensures consistent text format across your dataset
2. **Noise Reduction:** Removes irrelevant information that could confuse algorithms
3. **Complexity Reduction:** Simplifies text to focus on meaningful patterns
4. **Performance Enhancement:** Improves the efficiency and accuracy of downstream tasks

Real-world Impact

Consider searching for "running shoes" vs "Running Shoes!" - without preprocessing, these might be treated as completely different queries. Preprocessing ensures they're recognized as equivalent.

### Conceptual Question 1

**Before we start coding, think about your daily interactions with text processing systems (search engines, chatbots, translation apps). What challenges do you think these systems face when processing human language? List at least 3 specific challenges and explain why each is problematic.**

*Double-click this cell to write your answer:*

### Challenge 1:

Understanding that words can have many meanings / ambiguity for example the word light can mean brightness or weight. Which may cause confusion.

### Challenge 2:

Understanding the difference between sarcasm and actual feelings that a user may be inputting. The NLP may take the prompt in a different way that the user intended.

### Challenge 3:


Understanding Slang, Typos, and the different ways people may communicate with the commands. If the NLP isn't aware of the meaning you may get an off the wall response irrelevant to your goal.


---

## ✓ Part 1: Environment Setup

We'll be working with two major NLP libraries:

- **NLTK (Natural Language Toolkit):** Comprehensive NLP library with extensive resources
- **spaCy:** Industrial-strength NLP with pre-trained models

 **Note:** Installation might take 2-3 minutes to complete.

```
# Step 1: Install Required Libraries
print(" Installing NLP libraries...")
```

```
!pip install -q nltk spacy
!python -m spacy download en_core_web_sm
```

```
print(" Installation complete!")
```

```
  Installing NLP libraries...
Collecting en-core-web-sm==3.8.0
Downloading https://github.com/explosion/spacy-models/releases/download/en\_core\_web
```

✓ Download and installation successful

You can now load the package via `spacy.load('en_core_web_sm')`

⚠ Restart to reload dependencies

If you are in a Jupyter or Colab notebook, you may need to restart Python in order to load all the package's dependencies. You can do this by selecting the 'Restart kernel' or 'Restart runtime' option.

✓ Installation complete!

## ✓ 🤔 Conceptual Question 2

**Why do you think we need to install a separate language model (en\_core\_web\_sm) for spaCy? What components might this model contain that help with text processing? Think about what information a computer needs to understand English text.**

*I believe we should install a distinct language model for spaCy, such as en\_core\_web\_sm, because spaCy does not understand English by default. The language model provides it with the necessary tools to comprehend text. This model most likely contains crucial information such as a list of English terms, grammar rules, and examples of how words are used in sentences. It may also incorporate components that aid in activities such as recognizing elements of speech (such as nouns and verbs), determining the root form of words (such as changing "running" to "run"), and identifying names of people, places, or businesses. Essentially, the language model enables spaCy to "read" English in a computer-readable format.*

# Step 2: Import Libraries and Download NLTK Data

```
import nltk
import spacy
import string
import re
from collections import Counter
```

# Download essential NLTK data

```
print("📦 Downloading NLTK data packages...")
nltk.download('punkt')      # For tokenization
nltk.download('stopwords')  # For stop word removal
nltk.download('wordnet')    # For lemmatization
nltk.download('averaged_perceptron_tagger') # For POS tagging
nltk.download('punkt_tab')  # For sentence tokenization
```

```
print("\n✅ All imports and downloads completed!")
```



📦 Downloading NLTK data packages...

```
[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 wordnet to /root/nltk_data...
[nltk_data] Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data] /root/nltk_data...
```

```
[nltk_data] Package averaged_perceptron_tagger is already up-to-
[nltk_data] date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punkt_tab.zip.
```

✅ All imports and downloads completed!

## ✓ 📁 Part 2: Sample Text Data

We'll work with different types of text to understand how preprocessing affects various text styles:

- Simple text
- Academic text (with citations, URLs)
- Social media text (with emojis, hashtags)
- News text (formal writing)
- Product reviews (informal, ratings)

# Step 3: Load Sample Texts

```
simple_text = "Natural Language Processing is a fascinating field of AI. It's amazing!"
```

```
academic_text = ""
```

```
Dr. Smith's research on machine-learning algorithms is groundbreaking!
She published 3 papers in 2023, focusing on deep neural networks (DNNs).
The results were amazing – accuracy improved by 15.7%!
"This is revolutionary," said Prof. Johnson.
Visit https://example.com for more info. #NLP #AI @university
""
```

```
social_text = "OMG! Just tried the new coffee shop ☕ SO GOOD!!! Highly recommend 👍 #coffee"
```

```
news_text = ""
```

```
The stock market experienced significant volatility today, with tech stocks leading the d
Apple Inc. (AAPL) dropped 3.2%, while Microsoft Corp. fell 2.8%.
"We're seeing a rotation out of growth stocks," said analyst Jane Doe from XYZ Capital.
""
```

```
review_text = ""
```

```
This laptop is absolutely fantastic! I've been using it for 6 months and it's still super
The battery life is incredible – lasts 8–10 hours easily.
Only complaint: the keyboard could be better. Overall rating: 4.5/5 stars.
""
```

# Store all texts

```
sample_texts = {
    "Simple": simple_text,
    "Academic": academic_text.strip(),
    "Social Media": social_text,
    "News": news_text.strip(),
    "Product Review": review_text.strip()
}
```

```
print("📄 Sample texts loaded successfully!")
for name, text in sample_texts.items():
    preview = text[:80] + "..." if len(text) > 80 else text
    print(f"\n📄 {name}: {preview}")
```

🔄 📄 Sample texts loaded successfully!

- 📄 Simple: Natural Language Processing is a fascinating field of AI. It's amazing!
- 📄 Academic: Dr. Smith's research on machine-learning algorithms is groundbreaking! She publi...
- 📄 Social Media: OMG! Just tried the new coffee shop ☕ SO GOOD!!! Highly recommend 👍
- 📄 News: The stock market experienced significant volatility today, with tech stocks
- 📄 Product Review: This laptop is absolutely fantastic! I've been using it for 6 montl

### 🤔 Conceptual Question 3

**Looking at the different text types we've loaded, what preprocessing challenges do you anticipate for each type? For each text type below, identify at least 2 specific preprocessing challenges and explain why they might be problematic for NLP analysis.**

*Double-click this cell to write your answer:*

#### **Simple text challenges:**

1. Contractions - Words like "it's" must be stretched to "it is" so that the system can properly understand each word.
2. The sentence is very short, so there isn't much context to work with. This can make it difficult for the model to grasp the entire meaning.

#### **Academic text challenges:**

1. Special terms and abbreviations – Words like "DNNs" or "machine-learning" may need to be defined or explained to improve understanding.
2. URLs and Citations - Links like as <https://example.com> and references might cause noise and should be eliminated or handled properly during preprocessing.

#### **Social media text challenges:**

1. Emojis and hashtags provide meaning, but they are not traditional words, so the model may not understand how to read them.
2. Slang and casual language – Words like "OMG" and "SO GOOD!!!" are informal and sometimes written in all caps or with extra punctuation, which may confuse the model.

#### **News text challenges:**

1. With named entities companies such as "Apple Inc." and people's names must be correctly recognized and classified.
2. Formal language and quotations require a more complex writing style, and quotes from people must be handled correctly to avoid confusion with the reporter's statements.

### Product review challenges:

1. Mixed tones, reviews often contain both good and negative comments within the same text, making it difficult to detect the overall attitude.
2. Ratings in text, such as "4.5/5 stars," may need to be split or adapted to numerical form before analysis.

## ✓ Part 3: Tokenization

### What is Tokenization?

Tokenization is the process of breaking down text into smaller, meaningful units called **tokens**. These tokens are typically words, but can also be sentences, characters, or subwords.

### Why is it Important?

- Most NLP algorithms work with individual tokens, not entire texts
- It's the foundation for all subsequent preprocessing steps
- Different tokenization strategies can significantly impact results

### Common Challenges:

- **Contractions:** "don't" → "do" + "n't" or "don't"?
- **Punctuation:** Keep with words or separate?
- **Special characters:** How to handle @, #, URLs?

```
# Step 4: Tokenization with NLTK
from nltk.tokenize import word_tokenize, sent_tokenize
```

```
# Test on simple text
print("🔍 NLTK Tokenization Results")
print("=" * 40)
print(f"Original: {simple_text}")
```

```
# Word tokenization
nltk_tokens = word_tokenize(simple_text)
print(f"\nWord tokens: {nltk_tokens}")
print(f"Number of tokens: {len(nltk_tokens)}")
```

```
# Sentence tokenization
sentences = sent_tokenize(simple_text)
```

```
print(f"\nSentences: {sentences}")
print(f"Number of sentences: {len(sentences)}")
```



### NLTK Tokenization Results

```
=====
```

Original: Natural Language Processing is a fascinating field of AI. It's amazing!

Word tokens: ['Natural', 'Language', 'Processing', 'is', 'a', 'fascinating', 'field',  
Number of tokens: 14

Sentences: ['Natural Language Processing is a fascinating field of AI.', "It's amazing!  
Number of sentences: 2

## ✓ 🤔 Conceptual Question 4

**Examine the NLTK tokenization results above. How did NLTK handle the contraction "It's"? What happened to the punctuation marks? Do you think this approach is appropriate for all NLP tasks? Explain your reasoning.**

*Double-click this cell to write your answer:*

**How "It's" was handled:** NLTK divided the contraction "It's" into two tokens, "It" and "'s". This means that it treats the contraction as two independent words, which can be useful in situations where we want to better understand the root terms.

**Punctuation treatment:** Punctuation signs like the period and exclamation mark were considered as individual symbols. For example, the exclamation mark at the end of "amazing!" creates its own token.

**Appropriateness for different tasks:** This method is effective for fundamental NLP tasks such as part-of-speech tagging or lemmatization since it divides the text into manageable chunks. However, for tasks such as sentiment analysis or sarcasm comprehension, dividing contractions or punctuation may result in a loss of meaning or tone. So it depends on the task, sometimes we want to maintain the original structure more closely.

```
tokenization with spaCy
load('en_core_web_sm')

spaCy Tokenization Results")
})
val: {simple_text}")

1 spaCy
2 le_text)

3 ns
4 = [token.text for token in doc]
5 1 tokens: {spacy_tokens}")
6 2 of tokens: {len(spacy_tokens)}")

7 ed token information
8 Detailed Token Analysis:")
```

```

n':<12} {'POS':<8} {'Lemma':<12} {'Is Alpha':<8} {'Is Stop':<8}")
)
loc:
oken.text:<12} {token.pos_:<8} {token.lemma_:<12} {token.is_alpha:<8} {token.is_stop:<8}")

```

### 🔍 spaCy Tokenization Results

Original: Natural Language Processing is a fascinating field of AI. It's amazing!

Word tokens: ['Natural', 'Language', 'Processing', 'is', 'a', 'fascinating', 'field',  
Number of tokens: 14

#### 🔍 Detailed Token Analysis:

Token	POS	Lemma	Is Alpha	Is Stop
Natural	PROPN	Natural	1	0
Language	PROPN	Language	1	0
Processing	NOUN	processing	1	0
is	AUX	be	1	1
a	DET	a	1	1
fascinating	ADJ	fascinating	1	0
field	NOUN	field	1	0
of	ADP	of	1	1
AI	PROPN	AI	1	0
.	PUNCT	.	0	0
It	PRON	it	1	1
's	AUX	be	0	1
amazing	ADJ	amazing	1	0
!	PUNCT	!	0	0

### ✓ 🤔 Conceptual Question 5

**Compare the NLTK and spaCy tokenization results. What differences do you notice? Which approach do you think would be better for different NLP tasks? Consider specific examples like sentiment analysis vs. information extraction.**

*Double-click this cell to write your answer:*

**Key differences observed:** One significant difference is that spaCy treats contractions such as "It's" as a single token, but NLTK separates them into "It" and "'s". Furthermore, spaCy gives additional information such as the part of speech, lemma, and if the token is a stop word, which NLTK does not display by default. spaCy's handling of punctuation and word kinds appears to be more structured.

**Better for sentiment analysis:** I believe NLTK may be better for sentiment analysis in some situations because it breaks words more, such as "It's" into two halves. This can assist in detecting word origins, which may be important when attempting to assess emotions in text.

**Better for information extraction:** SpaCy is clearly better for tasks such as information extraction since it provides far more information about each word (such as POS, lemma, and entity recognition). This allows the system to better understand the meaning and function of each word.



**Overall assessment:** Both methods are important, but I believe spaCy is more competent and effective for the majority of real-world NLP jobs. It provides more information and handles text more intelligently, particularly when you need to do more than just separate words.

---

```
# Step 6: Test Tokenization on Complex Text
print("📝 Testing on Social Media Text")
print("=" * 40)
print(f"Original: {social_text}")

# NLTK approach
social_nltk_tokens = word_tokenize(social_text)
print(f"\nNLTK tokens: {social_nltk_tokens}")

# spaCy approach
social_doc = nlp(social_text)
social_spacy_tokens = [token.text for token in social_doc]
print(f"spaCy tokens: {social_spacy_tokens}")

print(f"\n🇺🇸 Comparison:")
print(f"NLTK token count: {len(social_nltk_tokens)}")
print(f"spaCy token count: {len(social_spacy_tokens)}")
```



📝 Testing on Social Media Text

=====

Original: OMG! Just tried the new coffee shop ☕ SO GOOD!!! Highly recommend 🍷 #coffee

NLTK tokens: ['OMG', '!', 'Just', 'tried', 'the', 'new', 'coffee', 'shop', '☕', 'SO', 'GOOD', '!!!', 'Highly', 'recommend', '🍷', '#coffee']

spaCy tokens: ['OMG', '!', 'Just', 'tried', 'the', 'new', 'coffee', 'shop', '☕', 'SO', 'GOOD', '!!!', 'Highly', 'recommend', '🍷', '#coffee']

🇺🇸 Comparison:

NLTK token count: 22

spaCy token count: 23



## Conceptual Question 6

**Looking at how the libraries handled social media text (emojis, hashtags), which library seems more robust for handling "messy" real-world text? What specific advantages do you notice? How might this impact a real-world application like social media sentiment analysis?**

*Double-click this cell to write your answer:*

**More robust library:** SpaCy appears to be more capable of handling chaotic, real-world text, such as social media posts.

**Specific advantages:** SpaCy is better at identifying emojis, hashtags, and symbols as distinct tokens without breaking them in strange ways. It also cleans up punctuation and offers additional information, such as part of speech, which can be important. NLTK occasionally breaks things awkwardly or fails to handle emojis properly.

**Impact on sentiment analysis:** Real world applications such as social media sentiment analysis require an understanding of the emotional tone behind emojis and hashtags. Emojis like "😍" and hashtags like "#yum" convey a pleasant message. SpaCy handles these better, which can lead to more accurate analysis of how people feel in their posts.

---

## ✓ 🛑 Part 4: Stop Words Removal

### What are Stop Words?

Stop words are common words that appear frequently in a language but typically don't carry much meaningful information about the content. Examples include "the", "is", "at", "which", "on", etc.

### Why Remove Stop Words?

1. **Reduce noise** in the data
2. **Improve efficiency** by reducing vocabulary size
3. **Focus on content words** that carry semantic meaning

### When NOT to Remove Stop Words?

- **Sentiment analysis:** "not good" vs "good" - the "not" is crucial!
- **Question answering:** "What is the capital?" - "what" and "is" provide context

```
# Step 7: Explore Stop Words Lists
from nltk.corpus import stopwords
```


```
# Get NLTK English stop words
nltk_stopwords = set(stopwords.words('english'))
print(f"🇺🇸 NLTK has {len(nltk_stopwords)} English stop words")
print(f"First 20: {sorted(list(nltk_stopwords))[:20]}")
```

```
# Get spaCy stop words
spacy_stopwords = nlp.Defaults.stop_words
print(f"\n🇺🇸 spaCy has {len(spacy_stopwords)} English stop words")
print(f"First 20: {sorted(list(spacy_stopwords))[:20]}")
```

```
# Compare the lists
common_stopwords = nltk_stopwords.intersection(spacy_stopwords)
nltk_only = nltk_stopwords - spacy_stopwords
spacy_only = spacy_stopwords - nltk_stopwords
```

```
print(f"\n🔍 Comparison:")
print(f"Common stop words: {len(common_stopwords)}")
print(f"Only in NLTK: {len(nltk_only)} - Examples: {sorted(list(nltk_only))[:5]}")
print(f"Only in spaCy: {len(spacy_only)} - Examples: {sorted(list(spacy_only))[:5]}")
```

```
🔗 🇺🇸 NLTK has 198 English stop words
First 20: ['a', 'about', 'above', 'after', 'again', 'against', 'ain', 'all', 'am', 'ar
```

 spaCy has 326 English stop words

First 20: ['"d"', '"ll"', '"m"', '"re"', '"s"', '"ve"', 'a', 'about', 'above', 'across', 'a

 Comparison:

Common stop words: 123

Only in NLTK: 75 – Examples: ['ain', 'aren', 'aren't', 'couldn', 'couldn't']

Only in spaCy: 203 – Examples: ['"d"', '"ll"', '"m"', '"re"', '"s"]

## ✓ Conceptual Question 7

**Why do you think NLTK and spaCy have different stop word lists? Look at the examples of words that are only in one list - do you agree with these choices? Can you think of scenarios where these differences might significantly impact your NLP results?**

*Double-click this cell to write your answer:*

**Reasons for differences:** NLTK and spaCy were developed by separate teams, so they may have differing perspectives on whether words are unimportant. Each library may select stop words based on the kind of jobs it focuses on. For example, NLTK may include more common English words, whereas spaCy may attempt to retain some words for more advanced analysis.

**Agreement with choices:** Some options make logic, but others are shocking. For example, spaCy uses "due" as a stop word, which could be useful in academic writing ("due to this reason"). So I don't always agree that a term can have meaning in one context but not in another.

**Scenarios where differences matter:** In activities such as sentiment analysis, deleting too many words may result in the loss of useful information. For example, removing a word such as "not" might change the meaning of a statement. Words like "may" or "due" may be critical in legal or academic texts, so selecting the correct stop word list may have a significant impact on the results.

```
# Step 8: Remove Stop Words with NLTK
# Test on simple text
original_tokens = nltk_tokens # From earlier tokenization
filtered_tokens = [word for word in original_tokens if word.lower() not in nltk_stopwords]

print("🟢 NLTK Stop Word Removal")
print("=" * 40)
print(f"Original: {simple_text}")
print(f"\nOriginal tokens ({len(original_tokens)}): {original_tokens}")
print(f"After removing stop words ({len(filtered_tokens)}): {filtered_tokens}")

# Show which words were removed
removed_words = [word for word in original_tokens if word.lower() in nltk_stopwords]
print(f"\nRemoved words: {removed_words}")

# Calculate reduction percentage
reduction = (len(original_tokens) - len(filtered_tokens)) / len(original_tokens) * 100
print(f"Vocabulary reduction: {reduction:.1f}%")
```

## 🔗 NLTK Stop Word Removal

Original: Natural Language Processing is a fascinating field of AI. It's amazing!

Original tokens (14): ['Natural', 'Language', 'Processing', 'is', 'a', 'fascinating',  
After removing stop words (10): ['Natural', 'Language', 'Processing', 'fascinating', 'field', 'moving', 'of', 'punctuation']

Removed words: ['is', 'a', 'of', 'It']

Vocabulary reduction: 28.6%

# Step 9: Remove Stop Words with spaCy

```
doc = nlp(simple_text)
```

```
spacy_filtered = [token.text for token in doc if not token.is_stop and not token.is_punct]
```

```
print("🔗 spaCy Stop Word Removal")
```

```
print("=" * 40)
```

```
print(f"Original: {simple_text}")
```

```
print(f"\nOriginal tokens ({len(spacy_tokens)}): {spacy_tokens}")
```

```
print(f"After removing stop words & punctuation ({len(spacy_filtered)}): {spacy_filtered}")
```

# Show which words were removed

```
spacy_removed = [token.text for token in doc if token.is_stop or token.is_punct]
```

```
print(f"\nRemoved words: {spacy_removed}")
```

# Calculate reduction percentage

```
spacy_reduction = (len(spacy_tokens) - len(spacy_filtered)) / len(spacy_tokens) * 100
```

```
print(f"Vocabulary reduction: {spacy_reduction:.1f}%")
```

## 🔗 Stop Word Removal

Original: Natural Language Processing is a fascinating field of AI. It's amazing!

Original tokens (14): ['Natural', 'Language', 'Processing', 'is', 'a', 'fascinating', 'field',  
After removing stop words & punctuation (7): ['Natural', 'Language', 'Processing', 'fascinating', 'field', 'moving', 'of', 'punctuation']

Removed words: ['is', 'a', 'of', 'It', 's', '!']

Vocabulary reduction: 50.0%

## 🤔 Conceptual Question 8

**Compare the NLTK and spaCy stop word removal results. Which approach removed more words? Do you think removing punctuation (as spaCy did) is always a good idea? Give a specific example where keeping punctuation might be important for NLP analysis.**

*Double-click this cell to write your answer:*

**Which removed more:** SpaCy deleted more words by filtering out both stop words and punctuation. NLTK merely deleted stop words, leaving punctuation in the results.

**Punctuation removal assessment:** Removing punctuation can be useful in some situations, such as when we're only trying to count words or locate keywords. However, it is not always a smart idea, especially if the punctuation changes the meaning of the text.

**Example where punctuation matters:** Punctuation can be used in sentiment analysis to convey mood. For example, "Great!!!" with three exclamation marks may convey more positive sentiment than simply "Great." Furthermore, using ellipses and the final word in a statement like "I like it... not" significantly alters its meaning. Keeping punctuation can be useful depending on the task.

---

## ✓ Part 5: Lemmatization and Stemming

### What is Lemmatization?

Lemmatization reduces words to their base or dictionary form (called a **lemma**). It considers context and part of speech to ensure the result is a valid word.

### What is Stemming?

Stemming reduces words to their root form by removing suffixes. It's faster but less accurate than lemmatization.

### Key Differences:

Aspect	Stemming	Lemmatization
Speed	Fast	Slower
Accuracy	Lower	Higher
Output	May be non-words	Always valid words
Context	Ignores context	Considers context


### Examples:

- **"running"** → Stem: "run", Lemma: "run"
- **"better"** → Stem: "better", Lemma: "good"
- **"was"** → Stem: "wa", Lemma: "be"

```
# Step 10: Stemming with NLTK
from nltk.stem import PorterStemmer

stemmer = PorterStemmer()

# Test words that demonstrate stemming challenges
test_words = ['running', 'runs', 'ran', 'better', 'good', 'best', 'flying', 'flies', 'was',

print( Stemming Demonstration")
print("=" * 30)
print(f"{'Original':<12} {'Stemmed':<12}")
print("-" * 25)

for word in test_words:
    stemmed = stemmer.stem(word)
    print(f"{word:<12} {stemmed:<12}")
```

```
# Apply to our sample text
sample_tokens = [token for token in nltk_tokens if token.isalpha()]
stemmed_tokens = [stemmer.stem(token.lower()) for token in sample_tokens]

print(f"\n🍃 Applied to sample text:")
print(f"Original: {sample_tokens}")
print(f"Stemmed: {stemmed_tokens}")
```

### 🔗 🍃 Stemming Demonstration

```
=====
Original      Stemmed
-----
running       run
runs          run
ran           ran
better        better
good          good
best          best
flying        fli
flies         fli
was           wa
were          were
cats          cat
dogs          dog
```

🍃 Applied to sample text:

Original: ['Natural', 'Language', 'Processing', 'is', 'a', 'fascinating', 'field', 'of']  
 Stemmed: ['natur', 'languag', 'process', 'is', 'a', 'fascin', 'field', 'of', 'ai', 'i']

## ✓ 🤔 Conceptual Question 9

**Look at the stemming results above. Can you identify any cases where stemming produced questionable results? For example, how were "better" and "good" handled? Do you think this is problematic for NLP applications? Explain your reasoning.**

*Double-click this cell to write your answer:*

**Questionable results identified:** Some stemmed words appear unusual or do not resemble actual words. For example, "flying" and "flies" were both shortened to "fli," which is not a valid word. Also, "was" was spelled "wa," which appears incorrect.

**Assessment of "better" and "good":** The stemmer made no changes to "better", "good", or "best". This is a little strange because they all signify something similar, yet the stemmer failed to put them together.

**Impact on NLP applications:** This might cause problems for tasks such as search engines and text matching. If related words are not reduced to the same form, the links between them may be lost. Furthermore, unusual stemmed words like "fli" or "wa" may make the content more difficult to understand or assess correctly.

```
# Step 11: Lemmatization with spaCy
print("🍃 spaCy Lemmatization Demonstration")
print("=" * 40)
```

```
# Test on a complex sentence
complex_sentence = "The researchers were studying the effects of running and swimming on better performance."
doc = nlp(complex_sentence)

print(f"Original: {complex_sentence}")
print(f"\n{'Token':<15} {'Lemma':<15} {'POS':<10} {'Explanation':<20}")
print("-" * 65)

for token in doc:
    if token.is_alpha:
        explanation = "No change" if token.text.lower() == token.lemma_ else "Lemmatized"
        print(f"{token.text:<15} {token.lemma_:<15} {token.pos_:<10} {explanation:<20}")

# Extract lemmas
lemmas = [token.lemma_.lower() for token in doc if token.is_alpha and not token.is_stop]
print(f"\n📄 Lemmatized tokens (no stop words): {lemmas}")
```

### 🔄 Lemmatization Demonstration

=====

researchers were studying the effects of running and swimming on better performance.

Lemma	POS	Explanation
the	DET	No change
researcher	NOUN	Lemmatized
be	AUX	Lemmatized
study	VERB	Lemmatized
the	DET	No change
effect	NOUN	Lemmatized
of	ADP	No change
run	VERB	Lemmatized
and	CCONJ	No change
swim	VERB	Lemmatized
on	ADP	No change
well	ADJ	Lemmatized
performance	NOUN	No change

tokens (no stop words): ['researcher', 'study', 'effect', 'run', 'swim', 'well', 'performance']

### # Step 12: Compare Stemming vs Lemmatization

```
comparison_words = ['better', 'running', 'studies', 'was', 'children', 'feet']
```

```
print("📊 Stemming vs Lemmatization Comparison")
print("=" * 50)
print(f"{'Original':<12} {'Stemmed':<12} {'Lemmatized':<12}")
print("-" * 40)
```

```
for word in comparison_words:
    # Stemming
    stemmed = stemmer.stem(word)

    # Lemmatization with spaCy
    doc = nlp(word)
    lemmatized = doc[0].lemma_
```

```
print(f"{word:<12} {stemmed:<12} {lemmatized:<12}")
```



### Stemming vs Lemmatization Comparison

Original	Stemmed	Lemmatized
better	better	well
running	run	run
studies	studi	study
was	wa	be
children	children	child
feet	feet	foot



### Conceptual Question 10

**Compare the stemming and lemmatization results. Which approach do you think is more suitable for:**

1. **A search engine** (where speed is crucial and you need to match variations of words)?
2. **A sentiment analysis system** (where accuracy and meaning preservation are important)?
3. **A real-time chatbot** (where both speed and accuracy matter)?

**Explain your reasoning for each choice.**

*Double-click this cell to write your answer:*

**1. Search engine:** Stemming is a better option because it is quicker and helps match similar words faster. For example, if someone searches for "running," stemming can match "ran" and "runs," even if the word is strange. Search engines prioritize speed and broad matching over exact grammar.

**2. Sentiment analysis:** Lemmatization is preferred here because it preserves the original meaning of the words. For example, "better" becomes "well," allowing us to better understand the sentiment. Stemming may overlook these linkages or confuse the model with unfamiliar word forms such as "wa" or "studi."

**3. Real-time chatbot:** Lemmatization is the best option since, while slightly slower, it generates more accurate results. A chatbot has to understand what users truly mean, and unusual derived phrases may make responses sound strange. Lemmatization helps to keep the conversation flowing and meaningful.

## ✓ 🗑️ Part 6: Text Cleaning and Normalization

### What is Text Cleaning?

Text cleaning involves removing or standardizing elements that might interfere with analysis:

- **Case normalization** (converting to lowercase)
- **Punctuation removal**
- **Number handling** (remove, replace, or normalize)
- **Special character handling** (URLs, emails, mentions)
- **Whitespace normalization**



## Why is it Important?

- Ensures consistency across your dataset
- Reduces vocabulary size
- Improves model performance
- Handles edge cases in real-world data

### # Step 13: Basic Text Cleaning

```
def basic_clean_text(text):
    """Apply basic text cleaning operations"""
    # Convert to lowercase
    text = text.lower()

    # Remove extra whitespace
    text = re.sub(r'\s+', ' ', text).strip()

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

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

    # Remove extra spaces again
    text = re.sub(r'\s+', ' ', text).strip()

    return text

# Test basic cleaning
test_text = "  Hello WORLD!!! This has 123 numbers and   extra spaces.  "
cleaned = basic_clean_text(test_text)

print("👉 Basic Text Cleaning")
print("=" * 30)
print(f"Original: '{test_text}'")
print(f"Cleaned: '{cleaned}'")
print(f"Length reduction: {(len(test_text) - len(cleaned))/len(test_text)*100:.1f}%")
```



### 👉 Basic Text Cleaning

=====

```
Original: '  Hello WORLD!!! This has 123 numbers and   extra spaces.  '
Cleaned: 'hello world this has numbers and extra spaces'
Length reduction: 26.2%
```

### # Step 14: Advanced Cleaning for Social Media

```
def advanced_clean_text(text):
    """Apply advanced cleaning for social media and web text"""
    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)

    # Remove email addresses
    text = re.sub(r'\S+@\S+', '', text)

    # Remove mentions (@username)
```

```

text = re.sub(r'@\w+', '', text)

# Convert hashtags (keep the word, remove #)
text = re.sub(r'#(\w+)', r'\1', text)

# Remove emojis (basic approach)
emoji_pattern = re.compile("[
    u"\U0001F600-\U0001F64F" # emoticons
    u"\U0001F300-\U0001F5FF" # symbols & pictographs
    u"\U0001F680-\U0001F6FF" # transport & map symbols
    u"\U0001F1E0-\U0001F1FF" # flags
    ]+", flags=re.UNICODE)
text = emoji_pattern.sub(r'', text)

# Convert to lowercase and normalize whitespace
text = text.lower()
text = re.sub(r'\s+', ' ', text).strip()

return text

# Test on social media text
print("🚀 Advanced Cleaning on Social Media Text")
print("=" * 45)
print(f"Original: {social_text}")

cleaned_social = advanced_clean_text(social_text)
print(f"Cleaned: {cleaned_social}")
print(f"Length reduction: {(len(social_text) - len(cleaned_social))/len(social_text)*100:.1f}%")

🔗 🚀 Advanced Cleaning on Social Media Text
=====
Original: OMG! Just tried the new coffee shop ☕ SO GOOD!!! Highly recommend 👍 #coffee
Cleaned: omg! just tried the new coffee shop ☕ so good!!! highly recommend coffee yur
Length reduction: 7.2%

```

## 🤔 Conceptual Question 11

**Look at the advanced cleaning results for the social media text. What information was lost during cleaning? Can you think of scenarios where removing emojis and hashtags might actually hurt your NLP application? What about scenarios where keeping them would be beneficial?**

*Double-click this cell to write your answer:*

**Information lost:** The cleaning process deleted emojis (such as ☕, 👍, 😍) and @mentions, which can have significant emotional or contextual meaning. Also, hashtags were saved as words (e.g., #yum → yum), but their purpose or tagging function was lost.

**Scenarios where removal hurts:** Emoji removal can have a severe negative impact on sentiment analysis or emotion recognition effectiveness. For example, 😍 may represent a strong happy mood, but 😞 denotes negativity. Hashtags such as #blessed or #fail frequently convey essential emotions. Losing these features may affect model accuracy.

**Scenarios where keeping helps:** Keeping emojis and hashtags is useful for brand monitoring, social media trend analysis, and opinion mining. Emojis can serve as sentiment proxies, while hashtags assist organize information by topic. For example, #delicious + 😊 clearly indicates a happy dining experience, making it beneficial for food delivery applications or customer feedback research.

---

## ✓ Part 7: Building a Complete Preprocessing Pipeline

Now let's combine everything into a comprehensive preprocessing pipeline that you can customize based on your needs.

### Pipeline Components:

1. **Text cleaning** (basic or advanced)
2. **Tokenization** (NLTK or spaCy)
3. **Stop word removal** (optional)
4. **Lemmatization/Stemming** (optional)
5. **Additional filtering** (length, etc.)

```
# Step 15: Complete Preprocessing Pipeline
def preprocess_text(text,
                    clean_level='basic',      # 'basic' or 'advanced'
                    remove_stopwords=True,
                    use_lemmatization=True,
                    use_stemming=False,
                    min_length=2):
    """
    Complete text preprocessing pipeline
    """
    # Step 1: Clean text
    if clean_level == 'basic':
        cleaned_text = basic_clean_text(text)
    else:
        cleaned_text = advanced_clean_text(text)

    # Step 2: Tokenize
    if use_lemmatization:
        # Use spaCy for lemmatization
        doc = nlp(cleaned_text)
        tokens = [token.lemma_.lower() for token in doc if token.is_alpha]
    else:
        # Use NLTK for basic tokenization
        tokens = word_tokenize(cleaned_text)
        tokens = [token for token in tokens if token.isalpha()]

    # Step 3: Remove stop words
    if remove_stopwords:
        if use_lemmatization:
            tokens = [token for token in tokens if token not in spacy_stopwords]
        else:
```

```

tokens = [token.lower() for token in tokens if token.lower() not in nltk_stop

# Step 4: Apply stemming if requested
if use_stemming and not use_lemmatization:
    tokens = [stemmer.stem(token.lower()) for token in tokens]

# Step 5: Filter by length
tokens = [token for token in tokens if len(token) >= min_length]

return tokens

print("🔧 Preprocessing Pipeline Created!")
print("✅ Ready to test different configurations.")

🔄 🔧 Preprocessing Pipeline Created!
✅ Ready to test different configurations.

# Step 16: Test Different Pipeline Configurations
test_text = sample_texts["Product Review"]
print(f"🎯 Testing on: {test_text[:100]}...")
print("=" * 60)

# Configuration 1: Minimal processing
minimal = preprocess_text(test_text,
                          clean_level='basic',
                          remove_stopwords=False,
                          use_lemmatization=False,
                          use_stemming=False)
print(f"\n1. Minimal processing ({len(minimal)} tokens):")
print(f"    {minimal[:10]}...")

# Configuration 2: Standard processing
standard = preprocess_text(test_text,
                           clean_level='basic',
                           remove_stopwords=True,
                           use_lemmatization=True)
print(f"\n2. Standard processing ({len(standard)} tokens):")
print(f"    {standard[:10]}...")

# Configuration 3: Aggressive processing
aggressive = preprocess_text(test_text,
                             clean_level='advanced',
                             remove_stopwords=True,
                             use_lemmatization=False,
                             use_stemming=True,
                             min_length=3)
print(f"\n3. Aggressive processing ({len(aggressive)} tokens):")
print(f"    {aggressive[:10]}...")

# Show reduction percentages
original_count = len(word_tokenize(test_text))
print(f"\n🇮🇹 Token Reduction Summary:")
print(f"    Original: {original_count} tokens")
print(f"    Minimal: {len(minimal)} ({(original_count-len(minimal))/original_count*100:.1f}

```

```
print(f"    Standard: {len(standard)} ({(original_count-len(standard))/original_count*100:
print(f"    Aggressive: {len(aggressive)} ({(original_count-len(aggressive))/original_coun
```

↔ 🎯 Testing on: This laptop is absolutely fantastic! I've been using it for 6 months at  
The ...

- =====
1. Minimal processing (34 tokens):  
['this', 'laptop', 'is', 'absolutely', 'fantastic', 'ive', 'been', 'using', 'it', '']
  2. Standard processing (18 tokens):  
['laptop', 'absolutely', 'fantastic', 've', 'use', 'month', 'super', 'fast', 'batter']
  3. Aggressive processing (21 tokens):  
['laptop', 'absolut', 'fantast', 'use', 'month', 'still', 'super', 'fast', 'batter']

📊 Token Reduction Summary:  
Original: 47 tokens  
Minimal: 34 (27.7% reduction)  
Standard: 18 (61.7% reduction)  
Aggressive: 21 (55.3% reduction)

## ✓ 🤔 Conceptual Question 12

**Compare the three pipeline configurations (Minimal, Standard, Aggressive). For each configuration, analyze:**

1. **What information was preserved?**
2. **What information was lost?**
3. **What type of NLP task would this configuration be best suited for?**

*Double-click this cell to write your answer:*

### Minimal Processing:

- Preserved: All original vocabulary, including stop words, punctuation (where necessary), and entire word forms. Complete grammatical structure and context. Emotional signals, tone, and modifiers (for example, "not" and "never").
- Lost: Minimal to none. Only minor cleanup (for example, token splitting).
- Best for: Language modeling, machine translation, text summarization, and text generation all require the preservation of entire context and grammatical structure in order to interpret and generate coherent natural language content.

### Standard Processing:

- Preserved: Lemmatization transforms core semantic content (e.g., "running" into "run" and "better" into "well"). Meaningful words, excluding popular stop words. The overall theme and tone of the document.
- Lost: Stop words can give grammatical structure or delicate context. Word inflections, which could be useful for stylistic or tense analysis.

- Best for: Sentiment analysis, topic modeling, document categorization, and semantic search all focus on understanding meaning and removing noise to improve performance.

### Aggressive Processing:

- Preserved: Only essential stemmed root words and a minimum vocabulary are required for deducing basic information. The text is simplified and easily presented.
- Lost: Stop words, punctuation, grammar, inflected forms, and, in some cases, semantic correctness. Fine-grained meaning (for example, the distinction between "running" and "ran" or the emotional tone).
- Best for: Search engines, text clustering, information retrieval, and spam filtering all require faster and more relevant term matches than advanced understanding.

# Step 17: Comprehensive Analysis Across Text Types

```
print("📊 Comprehensive Preprocessing Analysis")
print("=" * 50)
```

# Test standard preprocessing on all text types

```
results = {}
for name, text in sample_texts.items():
    original_tokens = len(word_tokenize(text))
    processed_tokens = preprocess_text(text,
                                       clean_level='basic',
                                       remove_stopwords=True,
                                       use_lemmatization=True)

    reduction = (original_tokens - len(processed_tokens)) / original_tokens * 100
    results[name] = {
        'original': original_tokens,
        'processed': len(processed_tokens),
        'reduction': reduction,
        'sample': processed_tokens[:8]
    }

print(f"\n📁 {name}:")
print(f"    Original: {original_tokens} tokens")
print(f"    Processed: {len(processed_tokens)} tokens ({reduction:.1f}% reduction)")
print(f"    Sample: {processed_tokens[:8]}")
```

# Summary table

```
print(f"\n\n📊 Summary Table")
print(f"{'Text Type':<15} {'Original':<10} {'Processed':<10} {'Reduction':<10}")
print("-" * 50)
for name, data in results.items():
    print(f"{name:<15} {data['original']:<10} {data['processed']:<10} {data['reduction']:.1f}%")
```

🔄 📊 Comprehensive Preprocessing Analysis

=====

```
📁 Simple:
Original: 14 tokens
Processed: 7 tokens (50.0% reduction)
```

Sample: ['natural', 'language', 'processing', 'fascinating', 'field', 'ai', 'amazir



Academic:

Original: 61 tokens

Processed: 26 tokens (57.4% reduction)

Sample: ['dr', 'smith', 'research', 'machinelearning', 'algorithm', 'groundbreake',



Social Media:

Original: 22 tokens

Processed: 10 tokens (54.5% reduction)

Sample: ['omg', 'try', 'new', 'coffee', 'shop', 'good', 'highly', 'recommend']



News:

Original: 51 tokens

Processed: 25 tokens (51.0% reduction)

Sample: ['stock', 'market', 'experience', 'significant', 'volatility', 'today', 'te



Product Review:

Original: 47 tokens

Processed: 18 tokens (61.7% reduction)

Sample: ['laptop', 'absolutely', 'fantastic', 've', 'use', 'month', 'super', 'fast'



Summary Table

Text Type	Original	Processed	Reduction	
Simple	14	7	50.0	%
Academic	61	26	57.4	%
Social Media	22	10	54.5	%
News	51	25	51.0	%
Product Review	47	18	61.7	%



## Final Conceptual Question 13

### Looking at the comprehensive analysis results across all text types:

#### 1. Which text type was most affected by preprocessing? Why do you think this happened?

Product Review was the most affected, with a token drop of 61.7%. Product reviews frequently use emotionally expressive language, adjectives, and repetitious sentences that are either stop words or lemmatized. Words such as "absolutely," "super," and contractions like "I've" are changed or eliminated. The casual and verbose manner of reviews tends to produce more noise, which preprocessing aggressively reduces.

#### 2. Which text type was least affected? What does this tell you about the nature of that text?

Simple text was the least affected, with a 50.0% decrease. The simple text most likely included short, full of data tokens with few filler words or superfluous modifiers. It depicts a plain writing style in which most words hold semantic weight, and so less is filtered away. This implies that simpler sentences are clearer and more focused even before preprocessing.

#### 3. If you were building an NLP system to analyze customer reviews for a business, which preprocessing approach would you choose and why?

The Standard Processing pipeline (basic cleaning, lemmatization, and stop word removal) would be the best option. It keeps the sense of sentiment-bearing words (such as "fantastic," "fast," and "recommend") while removing noise like stop words. Lemmatization guarantees consistency in representing comparable concepts (e.g., "using" Becomes "use"). It allows the model to find trends in opinions or satisfaction levels across different expressions without oversimplifying, as aggressive stemming might.

#### 4. What are the main trade-offs you need to consider when choosing preprocessing techniques for any NLP project?

Stemming is faster but can result in loss of meaning, whereas lemmatization is slower but more precise. Context versus Cleanliness. More aggressive cleaning removes useful context (such as negations and intensifiers). Generalization vs. Specificity: Excessive normalization might blur key distinctions in emotion or opinion analysis. Robustness vs Readability: Eliminating stop words increases model performance but may impair human interpretability or explainability.

---

## Lab Summary and Reflection

Congratulations! You've completed a comprehensive exploration of NLP preprocessing techniques.

### Key Concepts You've Mastered:

1. **Text Preprocessing Fundamentals** - Understanding why preprocessing is crucial
2. **Tokenization Techniques** - NLTK vs spaCy approaches and their trade-offs
3. **Stop Word Management** - When to remove them and when to keep them
4. **Morphological Processing** - Stemming vs lemmatization for different use cases
5. **Text Cleaning Strategies** - Basic vs advanced cleaning for different text types
6. **Pipeline Design** - Building modular, configurable preprocessing systems

### Real-World Applications:

These techniques form the foundation for search engines, chatbots, sentiment analysis, document classification, machine translation, and information extraction systems.

### Key Insights to Remember:

- **No Universal Solution:** Different NLP tasks require different preprocessing approaches
  - **Trade-offs Are Everywhere:** Balance information preservation with noise reduction
  - **Context Matters:** The same technique can help or hurt depending on your use case
  - **Experimentation Is Key:** Always test and measure impact on your specific task
- 

## Excellent work completing Lab 02!

For your reflection journal, focus on the insights you gained about when and why to use different techniques, the challenges you encountered, and connections you made to real-world applications.



