Text Preprocessing and Cleaning

ITAI 2373 Mod 02

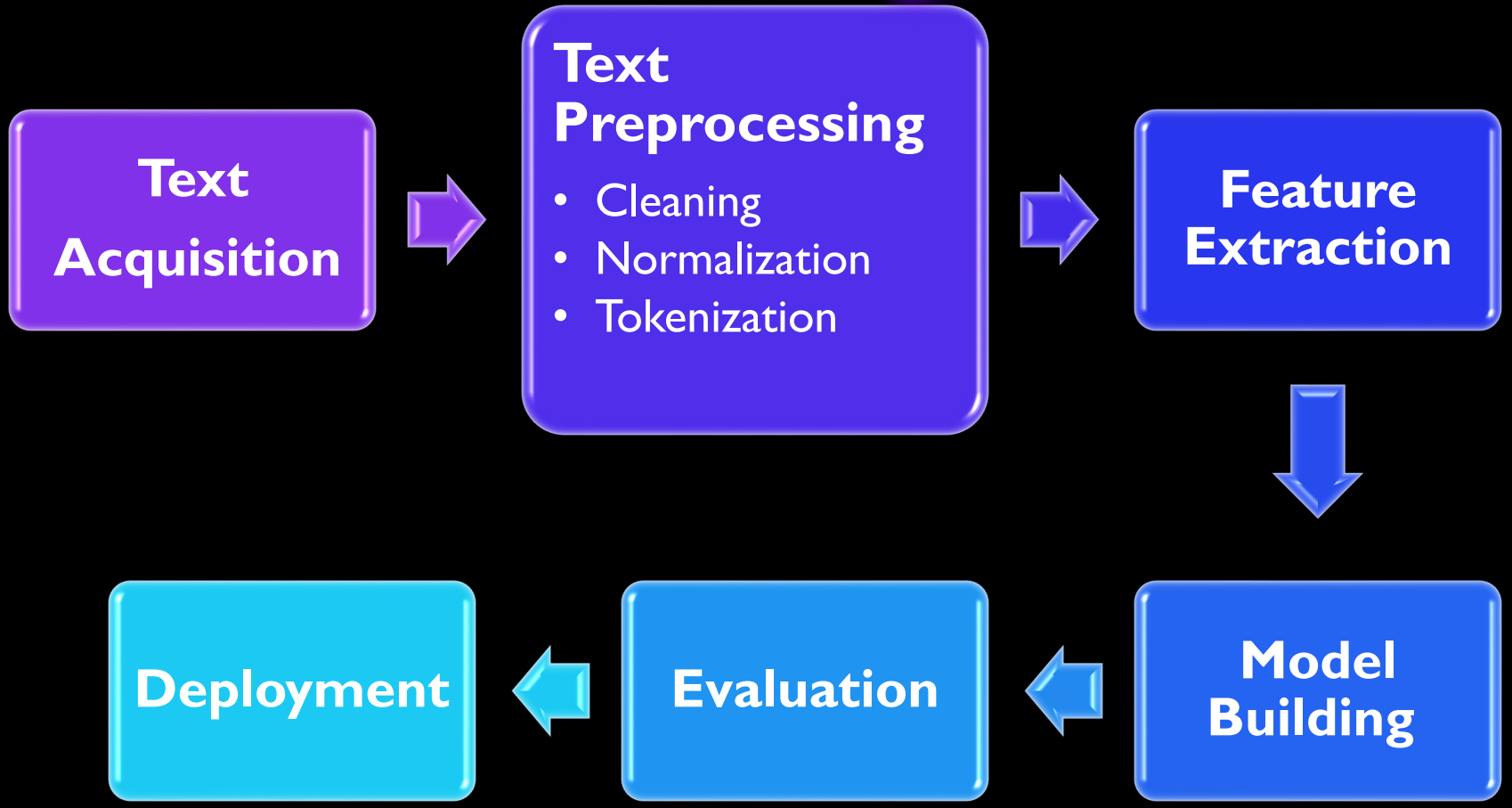
Learning Objectives

By the end of this module you will be able to:

Explain	Explain why text preprocessing is critical for NLP applications
Implement	Implement various tokenization methods for different use cases
Apply	Apply stop word removal techniques appropriately
Differentiate	Differentiate between stemming and lemmatization approaches
Develop	Develop strategies for handling punctuation, case, and special characters
Build	Build complete text cleaning pipelines using NLTK and SpaCy

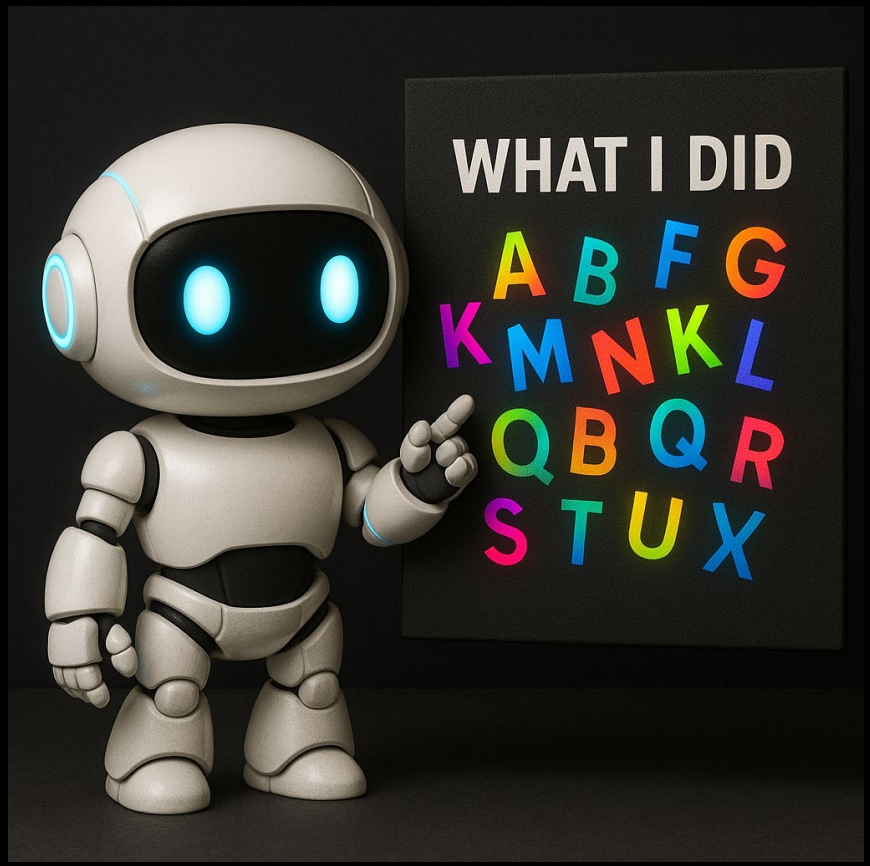
NLP Pipeline

We are here



What is textual data?

- Any data in the form of written or spoken language
- Characteristics:
  - Unstructured (not organized in a predefined format)
  - Contains diverse information (dates, names, locations, opinions)
  - Critical component for NLP applications
- Examples: Emails, social media posts, articles, books, transcribed speech



Challenges of Textual Data: Techinical

Acronyms and abbreviations: NASA, lol, ASAP, etc.

Special characters: Emojis, symbols, HTML tags

Formatting inconsistencies: Different date formats, number representations

Noise: Typos, grammatical errors, incomplete sentences

Challenges of Textual Data: Linguistics

Ambiguity: Words with multiple meanings ("bank" - financial institution or river edge)

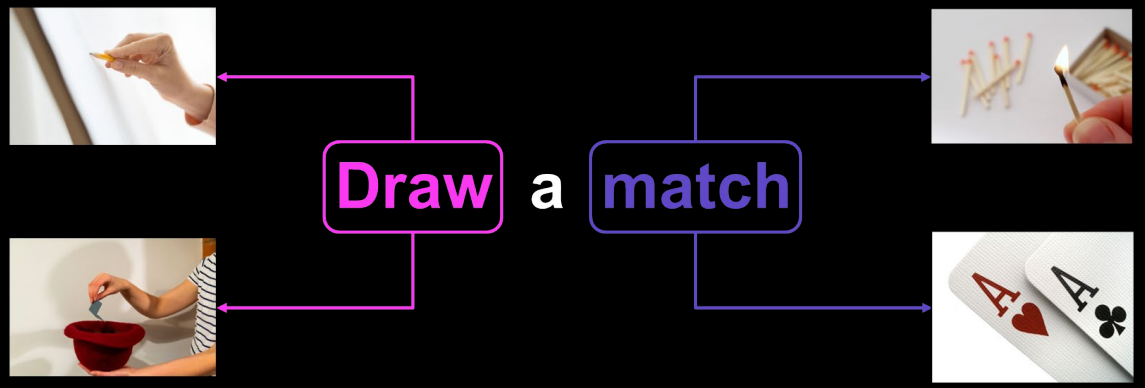
Homophones: Words that sound alike but have different meanings (their/there/they're)

Idiomatic expressions: Phrases whose meanings aren't literal ("kick the bucket")

Sarcasm and irony: When literal meaning differs from intended meaning

Regional variations: Different dialects and expressions across regions

Examples of NLP challenges



Flat-out like a lizard drinking

kangaroo in the headlights

Going off like a frog in a sock

I am not here to put socks in Centipedes

No time to waist

Really Busy

Frozen in panic or surprise,

Being extremely lively, energetic, or chaotic

Data quality Issues

Common Data Quality Problems:

- **Noisy data**:
  - Misspellings: "recieve" instead of "receive"
  - Typos: "teh" instead of "the"
  - Inconsistent punctuation: missing or excessive
  - Mixed formatting: ALL CAPS, lowercase, CamelCase
- **Incomplete data**:
  - Truncated sentences
  - Missing context
  - Abbreviated content
- **Inconsistent data**:
  - Multiple languages mixed
  - Different date/number formats
  - Varying terminology for same concepts

Why Preprocessing Matters

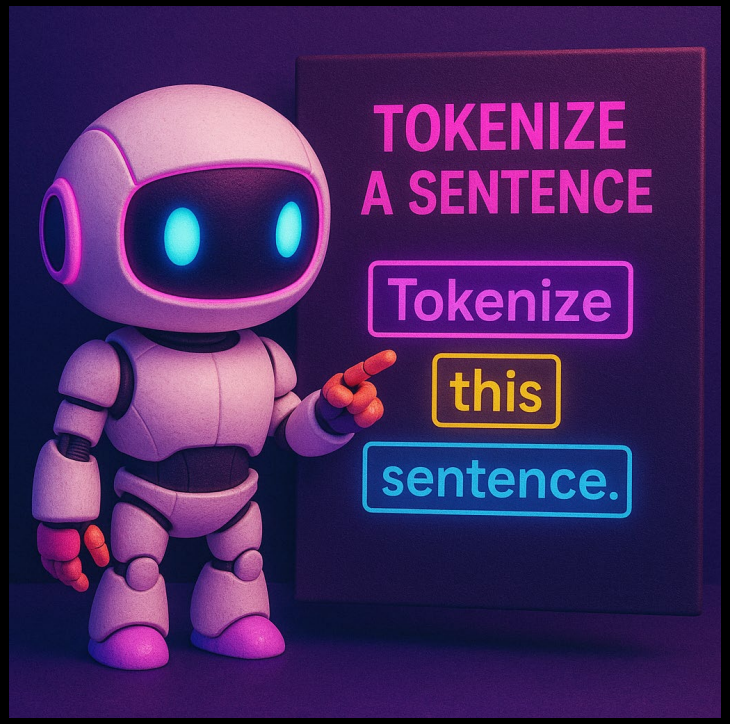
Impact of Preprocessing on NLP Tasks:

- **Improves model accuracy**: Clean data leads to better pattern recognition
- **Reduces dimensionality**: Fewer unique tokens means more efficient models
- **Standardizes input**: Consistent format for analysis
- **Removes noise**: Eliminates irrelevant information
- **Enhances feature extraction**: Better text representations
- **Addresses data sparsity**: Helps with limited training data

Tokenization - Breaking Text into Units

Tokenization: The process of splitting text into meaningful units (tokens)

- Types of Tokenization:
  - Word Tokenization: Split by word boundaries
    - "I love NLP!" → ["I", "love", "NLP", "!"]
  - Sentence Tokenization: Split by sentence boundaries
    - "Hello world. How are you?" → ["Hello world.", "How are you?"]
  - Subword Tokenization: Split into meaningful subunits
    - "preprocessing" → ["pre", "process", "ing"]
  - Character Tokenization: Split into individual characters
    - "Hello" → ["H", "e", "l", "l", "o"]



Tokenization Methods in Practice

- NLTK Tokenization:

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

text = "Hello world! This is an example of tokenization in NLTK."
sentences = sent_tokenize(text)  # ['Hello world!', 'This is an example of tokenization in NLTK.'
words = word_tokenize(text)      # ['Hello', 'world', '!', 'This', 'is', 'an', 'example', 'of', '
```

- SpaCy Tokenization:

```
import spacy

nlp = spacy.load("en_core_web_sm")
doc = nlp("Hello world! This is an example of tokenization in spaCy.")
sentences = [sent.text for sent in doc.sents]
words = [token.text for token in doc]
```

Common Tokenization Challenges:

- **Contractions**: "don't" → ["do", "n't"] or ["don't"]?
- **Hyphenated words**: "state-of-the-art" → one token or multiple?
- **Possessives**: "John's" → ["John", "'s"] or ["John's"]?
- **Compound words**: "database" vs "data base"
- **Numbers and dates**: "5/10/2025" → how to tokenize?
- **URLs and emails**: "user@example.com" → preserve or split?
- **Hashtags and mentions**: "#NLProc" "@user"
- **Emojis and emoticons**: "😊" ":)"
- **Multi-word expressions**: "New York" or "kick the bucket"

Stop Word Removal

- Words that frequently appear but don't contribute to the overall meaning
- Examples: a, the, so, is, it, at, in, this, there, that, my
- NLTK Implementation

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

stop_words = set(stopwords.words('english'))
text = "This is an example of stop word removal in NLP"
tokens = word_tokenize(text)
filtered_tokens = [word for word in tokens if word.lower() not in stop_words]
# Result: ['This', 'example', 'stop', 'word', 'removal', 'NLP']
```

NLTK list of Stop words

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', '
yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it'
s", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that',
"that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'a
t', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'bel
ow', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here
', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'suc
h', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "
don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "could
n't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", '
ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

- SpaCy Implementation

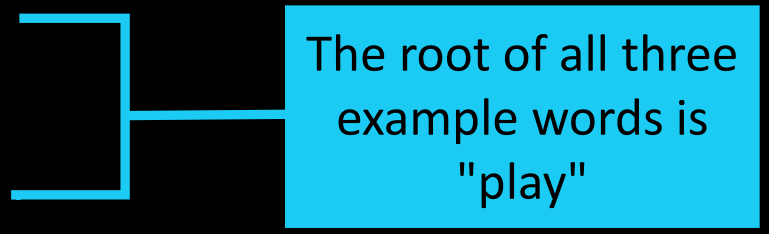
```python
import spacy

nlp = spacy.load("en_core_web_sm")
text = "This is an example of stop word removal in NLP"
doc = nlp(text)
filtered_tokens = [token.text for token in doc if not token.is_stop]
```

Stemming

- Set of rules to reduce a string to a substring that usually refers to a more general meaning.
- Goal: Remove word affixes (particularly suffixes), such as -s, -es, -ing, and -ed.

Example

- playing
- played
- plays



- Issue: Stemming doesn't usually work with irregular forms, such as irregular verbs (for example, *teach/taught* and *bring/brought*).

Lemmatization

- Reduces inflectional or variant forms to the base form
- Uses a vocabulary and morphological analysis
- Returns the base or dictionary form of the word, which is known as the *lemma*
- Examples:

Variant Forms	Lemma
car, cars	car
bought	buy
taught	teach
am, are, is	be

Stemming compared to lemmatization

Both convert words into their base forms.

Stemming	Lemmatization
Uses a set of rules to reduce a string to a substring that usually refers to a more general meaning.	Finds the base form by using a vocabulary and morphological analysis (more advanced than stemming).
Effective on simple cases, such as *teaching*/*teach*, *worked*/*work*, and *apples*/*apple*.	Effective on simple cases.
Cannot be used on challenging cases where rules cannot be applied, such as *taught*, *am*, and *better*.	Handles challenging cases, such as *taught*/*teach*, *am*/*to be*, and *better*/*good*.

Handling Punctuation, Case, & Special Characters

Common Approaches:

- Case Normalization:
  - Converting all text to lowercase (or uppercase)
  - Preserves: "Apple" vs. "apple" (company vs. fruit)
  - text.lower() or token.lower()

Punctuation Removal:

- Removing or separating punctuation marks
  - Preserves: "don't" → "don t" or "dont"
  - re.sub(r'[^\w\s]', '', text)

Special Character Handling:

- HTML tags: text → "text"
  - URLs: Preserve, remove, or replace with placeholders
  - Emojis: Remove or encode as text
  - Numbers: Keep, remove, or replace with tokens

Regular Expressions (regex) for Text Cleaning

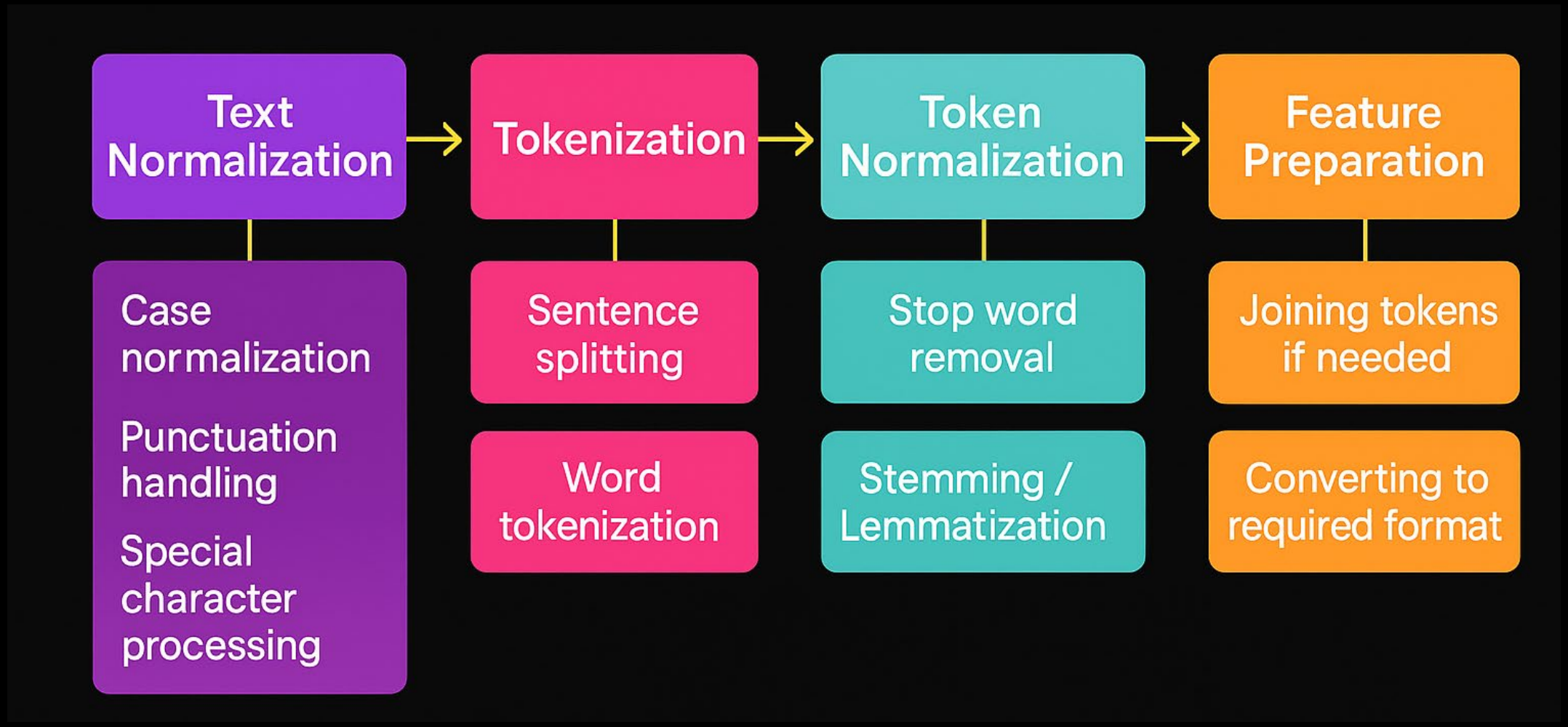
Powerful patterns for text matching and manipulation

Common Regex Patterns for Text Cleaning:

- Remove HTML tags: re.sub(r'<.*?>', '', text)
- Extract emails: re.findall(r'[\w\.-]+@[\w\.-]+', text)
- Remove URLs: re.sub(r'https?://\S+', '', text)
- Remove punctuation: re.sub(r'[^\w\s]', '', text)
- Remove extra whitespace: re.sub(r'\s+', ' ', text).strip()
- Extract hashtags: re.findall(r'#(\w+)', text)

```python
import re

text = "Check out https://example.com! Contact: user@example.com #NLP"
# Remove URLs
text = re.sub(r'https?://\S+', '', text )
# Result: "Check out ! Contact: user@example.com #NLP"
```

Text Processing Pipeline



NLTK vs. SpaCy for Preprocessing

- Comparison of key features

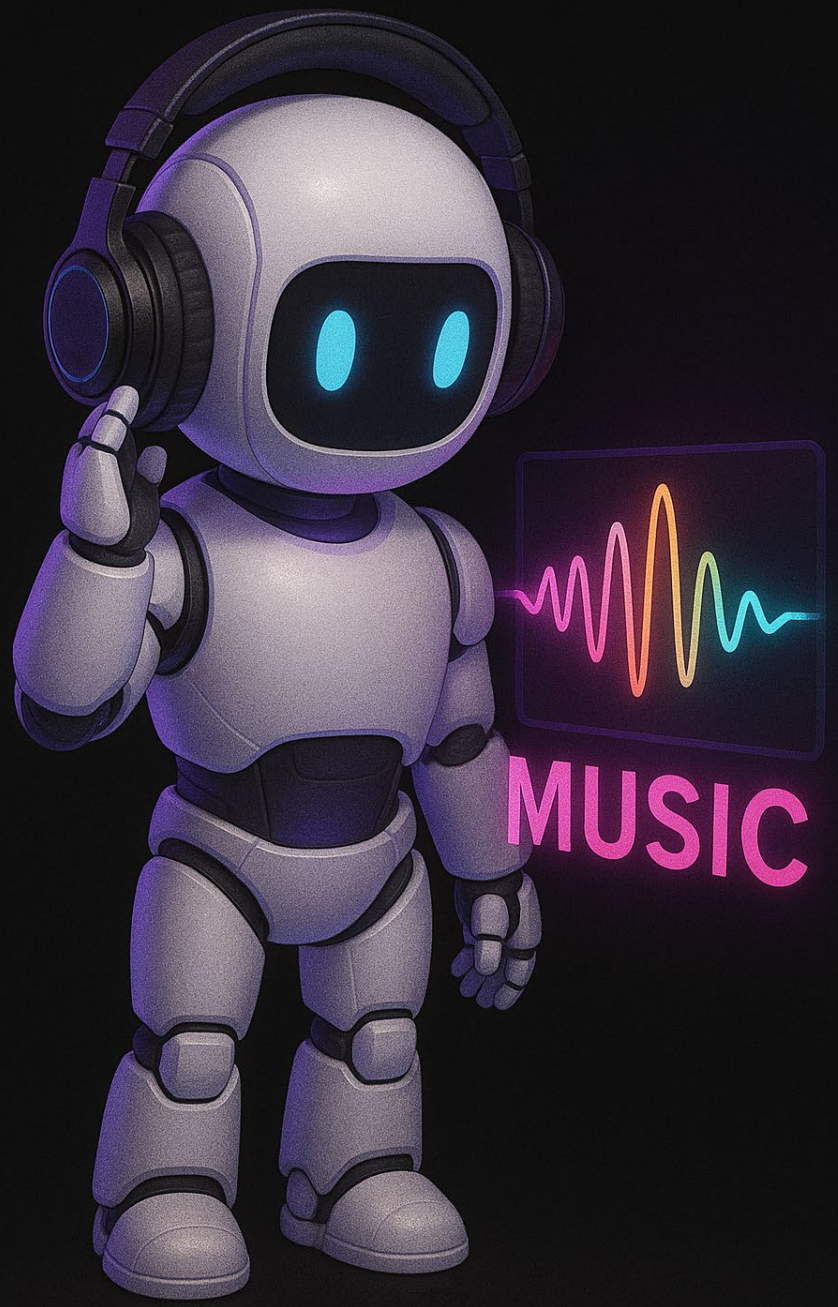
Feature	NLTK	SpaCy
Focus	Educational, research	Production, efficiency
Speed	Slower	Faster
Memory Usage	Lower	Higher
Pipeline Integration	Modular components	Integrated pipeline
Languages	Many (50+)	Fewer (20+) but better quality
Preprocessing	Separate steps	Integrated process
Learning Curve	Gentler	Steeper
Best For	Learning, experimentation	Production applications

Introduction to Audio as a Language Modality

- Language Beyond Text:
  - Spoken language predates written language
  - Speech is the primary form of human communication
- Audio captures nuances lost in text (tone, emphasis, emotion)
  - Audio in NLP Applications:
  - Voice assistants (Siri, Alexa, Google Assistant)
  - Transcription services
  - Call center analytics
  - Meeting summarization
  - Accessibility tools
- The Text-Audio Connection:
  - Speech-to-text converts audio to processable text
  - Text-to-speech converts processed text to audio output
  - Many NLP systems handle both modalities

Why Audio in Our NLP Course

- **Complete Language Understanding:**
  - Language exists in both written AND spoken forms
  - Complete NLP systems process both modalities
  - Audio provides context missing from text alone
- **Industry Reality:**
  - Most commercial NLP systems handle speech
  - Voice interfaces are increasingly common
  - Multimodal systems are the standard
- **Career Advantage:**
  - Skills in both text and audio processing
  - More versatile NLP practitioner
  - Broader job opportunities
- **Technical Convergence:**
  - Similar preprocessing concepts apply to both
  - Modern deep learning approaches handle both
  - Unified multimodal models emerging



From Text to Audio: Connecting the Modalities

- **Parallel Preprocessing Concepts:**
  - Text cleaning ↔ Audio noise reduction
  - Text normalization ↔ Audio normalization
  - Text tokenization ↔ Audio segmentation
  - Text feature extraction ↔ Audio feature extraction
- **Multimodal NLP Systems:**
  - Speech recognition → Text preprocessing → Analysis
  - Text generation → Text-to-speech synthesis
  - Joint processing of transcripts and audio
- **Example: Voice Assistant Pipeline**
  - Audio input → Speech recognition → Text
  - Text preprocessing → Intent recognition
  - Response generation → Text-to-speech → Audio output

Real-world Applications of Text Preprocessing

Search Engines:	• Query normalization • Document indexing • Relevance ranking
Sentiment Analysis:	• Social media monitoring • Customer feedback analysis • Brand reputation management
Content Recommendation:	• Document similarity • User preference modeling • Personalized content delivery
Information Extraction:	• Named entity recognition • Relation extraction • Knowledge graph construction

Best Practices and Common Pitfalls

Best Practices:

Understand your data and task requirements

Preserve original text alongside preprocessed versions

Document preprocessing decisions

Evaluate impact of preprocessing on downstream tasks

Consider domain-specific preprocessing needs

Common Pitfalls:

Over-preprocessing (removing useful information)

Under-preprocessing (leaving noise that harms performance)

Inconsistent preprocessing between training and inference

Ignoring language-specific considerations

Preprocessing without clear purpose

Finding the Right Balance:

Start simple, add complexity as needed

Test different preprocessing combinations

Measure impact on final task performance

Key Takeaways

- Text preprocessing is essential for effective NLP applications.
- Understanding challenges in textual data enhances model performance.
- Tokenization and stop word removal significantly impact analysis.
- Stemming and lemmatization help normalize text for better results.
- Regular expressions are powerful tools for cleaning text data.
- Combining multiple methods leads to optimal preprocessing strategies.