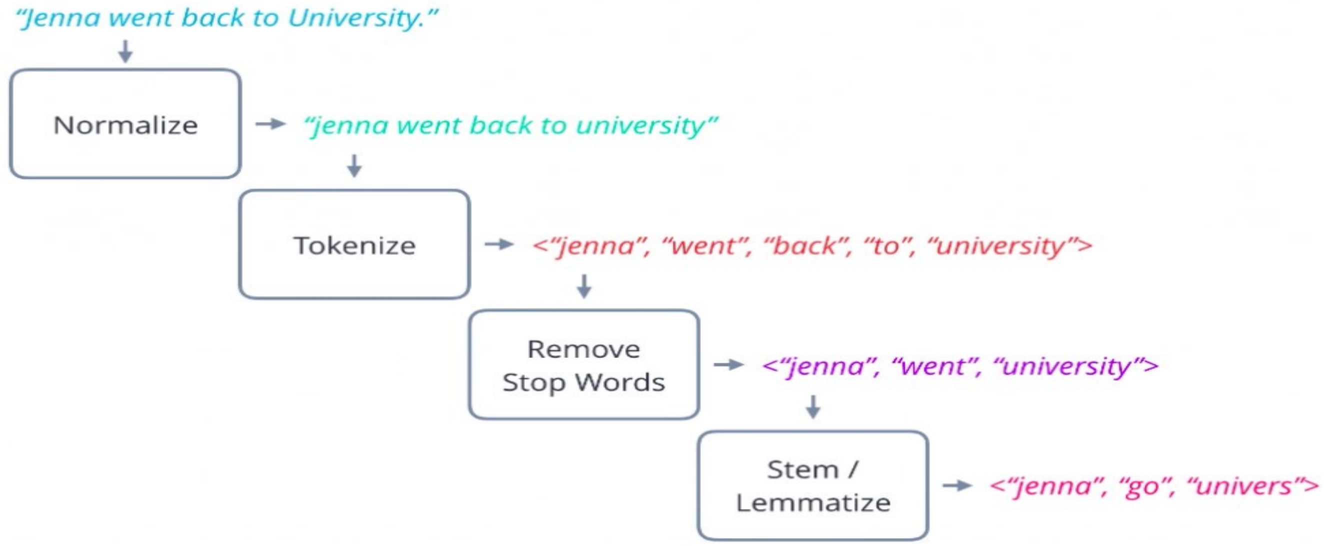
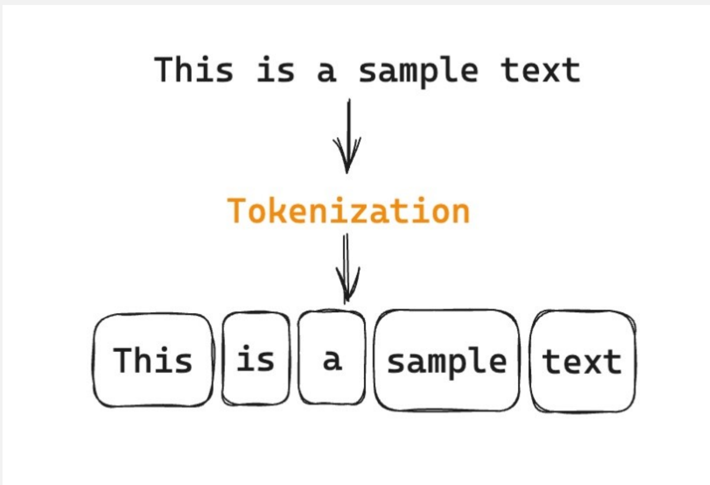
Natural Language Pre-processing (NLP)

Natural Language Processing (NLP) involves a series of pre-processing steps to transform raw text data into a format suitable for analysis or machine learning models. These steps help improve the quality of the data and make it easier for algorithms to understand and process the text. Below are the key pre-processing steps used in NLP, along with explanations and example code.

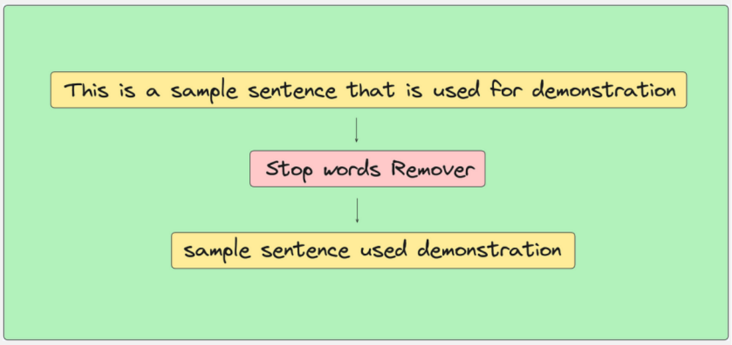
33 Common Pre-processing step commonly used before feeding data into an NLP model

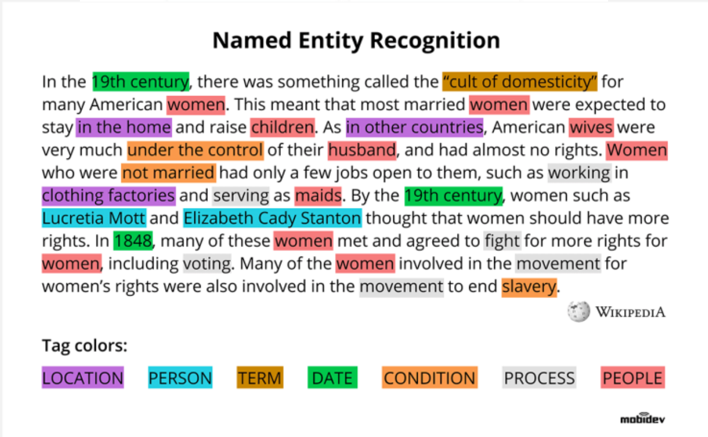


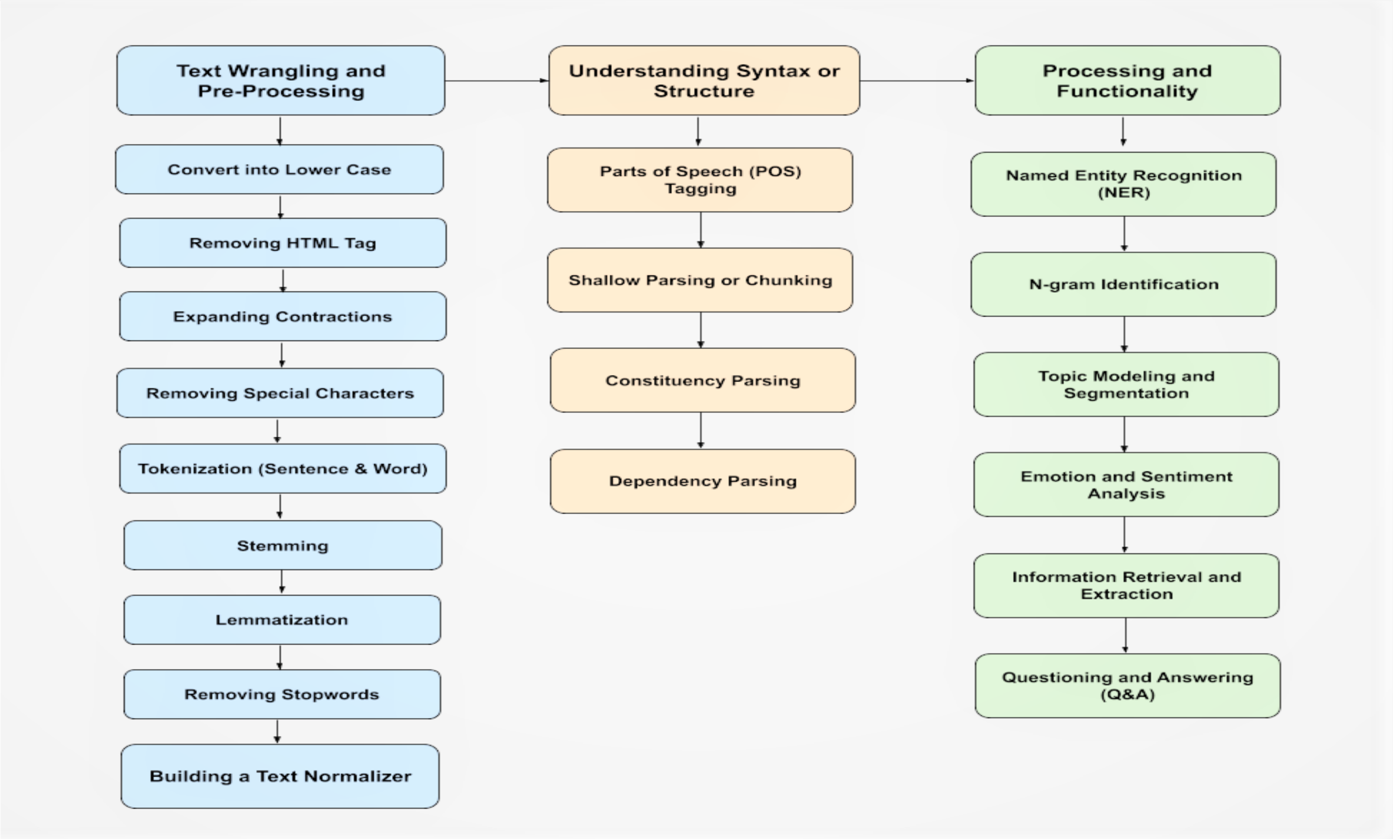
1. Lowercasing
2. Tokenization
3. Removing Punctuation
4. Removing Stopwords
5. Stemming
6. Lemmatization
7. Removing Numbers
8. Removing Extra Spaces
9. Handling Contractions
10. Removing Special Characters
11. Part-of-Speech (POS) Tagging
12. Named Entity Recognition (NER)



13. Vectorization
14. Handling Missing Data
15. Normalization
16. Spelling Correction
17. Handling Emojis and Emoticons
18. Removing HTML Tags
19. Handling URLs
20. Handling Mentions and Hashtags
21. Sentence Segmentation
22. Handling Abbreviations
23. Language Detection
24. Text Encoding
25. Handling Whitespace Tokens
26. Handling Dates and Times
27. Text Augmentation
28. Handling Negations
29. Dependency Parsing
30. Handling Rare Words
31. Text Chunking
32. Handling Synonyms
33. Text Normalization for Social Media







Detailed explanation of each pre-processing step commonly used before feeding data into an NLP model and during its use:

1. Lowercasing

- **Purpose**: Converts all text to lowercase to ensure uniformity.
- **Why**: Reduces the vocabulary size and avoids treating the same word in different cases as different tokens (e.g., "Apple" vs. "apple").

```
text = "Hello World! This is NLP."
text = text.lower()
print(text)
```

```
hello world! this is nlp.
```

2. Tokenization

- **Purpose:** Splits text into individual words, phrases, or sentences (tokens).
- **Why:** Breaks down text into manageable units for further processing.

```
import nltk

nltk.download('punkt_tab')
from nltk.tokenize import word_tokenize

text = "Hello World! This is NLP."
tokens = word_tokenize(text)
print(tokens)
```

```
['Hello', 'World', '!', 'This', 'is', 'NLP', '.']
```

3. Removing Punctuation

- **Purpose:** Removes punctuation marks like commas, periods, exclamation marks, etc.
- **Why:** Punctuation often doesn't contribute to the meaning in many NLP tasks and can add noise.

```
import string

text = "Hello, World! This is NLP."
text = text.translate(str.maketrans('', '',
string.punctuation))
print(text)
```

```
Hello World This is NLP
```

4. Removing Stopwords

- **Purpose:** Removes common words like "the," "is," "and," which don't carry significant meaning.
- **Why:** Reduces noise and focuses on meaningful words.

```
import nltk

# Download the 'stopwords' dataset
nltk.download('stopwords')

from nltk.corpus import stopwords

stop_words = set(stopwords.words('english'))
tokens = ["this", "is", "a", "sample", "sentence"]
filtered_tokens = [word for word in tokens if word.lower() not in
stop_words]
print(filtered_tokens)
```

```
['sample', 'sentence']
```

5. Stemming

- **Purpose:** Reduces words to their root form by chopping off suffixes (e.g., "running" → "run").
- **Why:** Simplifies words to their base form, reducing vocabulary size.

```
from nltk.stem import PorterStemmer

stemmer = PorterStemmer()
words = ["running", "runner", "ran"]
stemmed_words = [stemmer.stem(word) for word in words]
print(stemmed_words)
```

```
['run', 'runner', 'ran']
```

6. Lemmatization

- **Purpose:** Converts words to their base or dictionary form (e.g., "better" → "good").
- **Why:** More accurate than stemming as it uses vocabulary and morphological analysis.

```
import nltk
nltk.download('wordnet')

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
words = ["running", "runner", "ran"]
lemmatized_words = [lemmatizer.lemmatize(word, pos='v') for word
in words]
print(lemmatized_words)
```

```
['run', 'runner', 'ran']
```

7. Removing Numbers

- **Purpose:** Removes numeric values from the text.
- **Why:** Numbers may not be relevant in certain NLP tasks like sentiment analysis.

```
import re
text = "There are 3 apples and 5 oranges."
text = re.sub(r'\d+', '', text)
print(text)
```

```
There are  apples and  oranges.
```

8. Removing Extra Spaces

- **Purpose:** Eliminates multiple spaces, tabs, or newlines.
- **Why:** Ensures clean and consistent text formatting.

```
text = "  This   is   a   sentence.   "
text = ' '.join(text.split())
print(text)
```

```
This is a sentence.
```

9. Handling Contractions

- **Purpose:** Expands contractions (e.g., "can't" → "cannot").
- **Why:** Standardizes text for better processing.

```
!pip install contractions
from contractions import fix

text = "I can't do this."
text = fix(text)
print(text)
```

```
I cannot do this.
```

10. Removing Special Characters

- **Purpose:** Removes non-alphanumeric characters like @, #, $, etc.
- **Why:** Reduces noise and irrelevant symbols.

```
import re

text = "This is a #sample text with @special characters!"
text = re.sub(r'[^\w\s]', '', text)
print(text)
```

```
This is a sample text with special characters
```

11. Part-of-Speech (POS) Tagging

- **Purpose:** Assigns grammatical tags to words (e.g., noun, verb, adjective).
- **Why:** Helps in understanding the syntactic structure of sentences.

```
import nltk
from nltk import pos_tag
from nltk.tokenize import word_tokenize

# Download the required resource
nltk.download('averaged_perceptron_tagger_eng')

tokens = word_tokenize("This is a sample sentence.")
pos_tags = pos_tag(tokens)
print(pos_tags)
```

```
[('This', 'DT'), ('is', 'VBZ'), ('a', 'DT'), ('sample', 'JJ'), ('sentence', 'NN'), ('.', '.')]
```

12. Named Entity Recognition (NER)

- **Purpose:** Identifies and classifies entities like names, dates, locations, etc.
- **Why:** Useful for tasks like information extraction.

```
import nltk
from nltk import pos_tag, ne_chunk
from nltk.tokenize import word_tokenize

# Download the required resources
nltk.download('words')
nltk.download('maxent_ne_chunker')
nltk.download('averaged_perceptron_tagger')
# Download the 'maxent_ne_chunker_tab' resource
nltk.download('maxent_ne_chunker_tab') # This line is crucial to fix the error.

tokens = word_tokenize("John works at Google in New York.")
pos_tags = pos_tag(tokens)
ner_tags = ne_chunk(pos_tags)
print(ner_tags)
```

```
(S
  (PERSON John/NNP)
  works/VBZ
  at/IN
  (ORGANIZATION Google/NNP)
  in/IN
  (GPE New/NNP York/NNP)
  ./.)
```

13. Vectorization

- **Purpose:** Converts text into numerical vectors (e.g., Bag of Words, TF-IDF, Word Embeddings).
- **Why:** Machine learning models require numerical input.

```python
from sklearn.feature_extraction.text import
CountVectorizer

corpus = ["This is a sample sentence.", "Another example
sentence."]
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
print(X.toarray())  # Output: [[1 1 1 1 0], [0 1 1 0 1]]
print(vectorizer.get_feature_names_out())
```

```
[[0 0 1 1 1 1]
 [1 1 0 0 1 0]]
['another' 'example' 'is' 'sample' 'sentence' 'this']
```

14. Handling Missing Data

- **Purpose:** Fills or removes missing or incomplete text data.
- **Why:** Ensures the dataset is complete and consistent.

```python
import pandas as pd

data = {"text": ["Hello", None, "World"]}
df = pd.DataFrame(data)
df["text"].fillna("My Dear", inplace=True)  # Fill missing values
print(df)
```

```
      text
0    Hello
1  My Dear
2    World
```

15. Normalization

- **Purpose:** Standardizes text (e.g., converting all dates to a single format).
- **Why:** Ensures consistency in the dataset.

```python
import unicodedata

text = "Café"
text = unicodedata.normalize('NFKD', text).encode('ascii',
'ignore').decode('utf-8')
print(text)
```

```
Cafe
```

16. Spelling Correction

- **Purpose:** Corrects spelling errors in the text.
- **Why:** Improves the quality of the text for analysis.

```python
from textblob import TextBlob

text = "I made a many mistakes in Artificial intellengence"
blob = TextBlob(text)
corrected_text = blob.correct()
print(corrected_text)
```

```
I made a many mistakes in Artificial intelligence
```

17. Handling Emojis and Emoticons

- **Purpose:** Converts emojis and emoticons into text or removes them.
- **Why:** Emojis can carry sentiment or meaning that needs to be captured.

```python
!pip install emoji
import emoji

text = "I love Python! □"
# Convert emojis to text
text = emoji.demojize(text)
```

```
print(text)  # Output: "I love Python! :smiling_face_with_smiling_eyes:"

# Remove emojis
text = emoji.replace_emoji(text, replace="")
print(text)
```

```
I love Python! :smiling_face_with_smiling_eyes:
I love Python! :smiling_face_with_smiling_eyes:
```

18. Removing HTML Tags

- **Purpose:** Removes HTML tags from web scraped text.
- **Why:** HTML tags are irrelevant for most NLP tasks.

```
from bs4 import BeautifulSoup

text = "<p>This is a <b>sample</b> text.</p>"
soup = BeautifulSoup(text, "html.parser")
clean_text = soup.get_text()
print(clean_text)
```

```
This is a sample text.
```

19. Handling URLs

- **Purpose:** Removes or replaces URLs in the text.
- **Why:** URLs are often irrelevant for text analysis.

```
import re

text = "Visit my website at https://example.com."
text = re.sub(r'http\S+|www\S+|https\S+', '', text,
flags=re.MULTILINE)
print(text)
```

```
Visit my website at
```

20. Handling Mentions and Hashtags

- **Purpose:** Processes or removes social media mentions (@user) and hashtags (#topic).
- **Why:** Useful for social media text analysis.

```
text = "Hey @user, check out #NLP!"
text = re.sub(r'@\w+|#\w+', '', text)
print(text)
```

```
Hey , check out !
```

21. Sentence Segmentation

- **Purpose:** Splits text into individual sentences.
- **Why:** Important for tasks like machine translation or summarization.

```
from nltk.tokenize import sent_tokenize

text = "This is the first sentence. This is the second sentence."
sentences = sent_tokenize(text)
print(sentences)
```

```
['This is the first sentence.', 'This is the second sentence.']
```

22. Handling Abbreviations

- **Purpose:** Expands abbreviations (e.g., "ASAP" → "as soon as possible").
- **Why:** Ensures clarity and consistency.

```
!pip install contractions

import contractions
text = "I'll be there ASAP."
expanded_text = contractions.fix(text)
```

```
print(expanded_text)
```

```
I will be there AS SOON AS POSSIBLE.
```

23. Language Detection

- **Purpose:** Identifies the language of the text.
- **Why:** Ensures the correct NLP model is applied.

```
!pip install langdetect
from langdetect import detect
text = "Ceci est un texte en français."
language = detect(text)
print(language)
```

```
fr
```

24. Text Encoding

- **Purpose:** Converts text into a specific encoding format (e.g., UTF-8).
- **Why:** Ensures compatibility with NLP tools and models.

```
text = "Café"
text = text.encode('utf-8').decode('utf-8')
print(text)
```

```
Café
```

25. Handling Whitespace Tokens

- **Purpose:** Removes or processes tokens that are just spaces or empty strings.
- **Why:** Ensures clean and meaningful tokens.

```
tokens = ["This", " ", "is", " ", "a", " ", "sample", " "]
tokens = [token for token in tokens if token.strip()]
print(tokens)
```

```
['This', 'is', 'a', 'sample']
```

26. Handling Dates and Times

- **Purpose:** Standardizes or extracts date and time formats.
- **Why:** Useful for time-sensitive analysis.

```
import dateutil.parser as dparser

text = "The event is on 2023-10-15."
date = dparser.parse(text, fuzzy=True)
print(date)
```

```
2023-10-15 00:00:00
```

27. Text Augmentation

- **Purpose:** Generates additional training data by modifying existing text (e.g., synonym replacement).
- **Why:** Improves model robustness and performance.

```
#!pip install nlpaug # Install the nlpaug library
from nlpaug.augmenter.word import SynonymAug

aug = SynonymAug(aug_src='wordnet')
text = "This is a sample text."
augmented_text = aug.augment(text)
print(augmented_text)
```

```
['This is a sample schoolbook.']
```

28. Handling Negations

- **Purpose:** Identifies and processes negations (e.g., "not good").
- **Why:** Important for sentiment analysis and understanding context.

```python
from nltk import word_tokenize

text = "This is not good."
tokens = word_tokenize(text)
for i, token in enumerate(tokens):
    if token == "not" and i + 1 < len(tokens):
        tokens[i + 1] = "not_" + tokens[i + 1]
print(tokens)
```

```
['This', 'is', 'not', 'not_good', '.']
```

29. Dependency Parsing

- **Purpose:** Analyzes the grammatical structure of a sentence.
- **Why:** Helps in understanding relationships between words.

```python
import spacy

!python -m spacy download en_core_web_sm # Download the model if not already downloaded
nlp = spacy.load("en_core_web_sm")  # Load the model directly using spacy.load

# The rest of your code remains the same
text = "This is a sample sentence."
doc = nlp(text)
for token in doc:
    print(token.text, token.dep_, token.head.text)
```

```
This nsubj is
is ROOT is
a det sentence
sample compound sentence
sentence attr is
. punct is
```

30. Handling Rare Words

- **Purpose:** Replaces or removes rare words that occur infrequently.
- **Why:** Reduces noise and improves model efficiency.

```
from collections import Counter

tokens = ["this", "is", "a", "rare", "word", "word"]
word_counts = Counter(tokens)
rare_words = {word for word, count in word_counts.items()
if count < 2}
tokens = [token if token not in rare_words else "<UNK>" for
token in tokens]
print(tokens)
```

```
['<UNK>', '<UNK>', '<UNK>', '<UNK>', 'word', 'word']
```

31. Text Chunking

- **Purpose:** Groups words into "chunks" based on POS tags (e.g., noun phrases).
- **Why:** Useful for information extraction.

```
from nltk import pos_tag, word_tokenize
from nltk.chunk import RegexpParser

text = "This is a sample sentence."
tokens = word_tokenize(text)
pos_tags = pos_tag(tokens)
grammar = "NP: {<DT>?<JJ>*<NN>}"
chunk_parser = RegexpParser(grammar)
tree = chunk_parser.parse(pos_tags)
print(tree)
```

```
(S This/DT is/VBZ (NP a/DT sample/JJ sentence/NN) ./.)
```

32. Handling Synonyms

- **Purpose:** Replaces words with their synonyms.
- **Why:** Helps in text augmentation and reducing redundancy.

```
from nltk.corpus import wordnet

word = "happy"
synonyms = wordnet.synsets(word)
print([syn.lemmas()[0].name() for syn in synonyms])
```

```
['happy', 'felicitous', 'glad', 'happy']
```

33. Text Normalization for Social Media

- **Purpose:** Processes informal text (e.g., "u" → "you", "gr8" → "great").
- **Why:** Social media text often contains informal language and slang.

```
import re

text = "I loooove this!"
text = re.sub(r'(.)\1+', r'\1', text)
print(text)
```

```
I love this!
```

These pre-processing steps are crucial for cleaning, standardizing, and transforming raw text into a format suitable for NLP models. The specific steps used depend on the task (e.g., sentiment analysis, machine translation) and the nature of the text (e.g., formal documents, social media posts).

The importance of pre-processing steps in NLP depends on the **specific task**, **type of text data**, and **the NLP model being used**. However, some steps are generally considered **more critical** across most NLP tasks. Here's a breakdown:

Most Important Pre-processing Steps for NLP

1. **Tokenization**
   - **Why**: Tokenization is the foundation of NLP. It breaks text into meaningful units (words, sentences, etc.), which are necessary for any further processing.
   - **When**: Always required, regardless of the task.
2. **Lowercasing**
   - **Why**: Ensures consistency by treating words like "Apple" and "apple" as the same. Reduces vocabulary size and computational complexity.
   - **When**: Important for tasks like text classification, sentiment analysis, and information retrieval.
3. **Removing Stopwords**
   - **Why**: Stopwords (e.g., "the," "is," "and") add noise and don't contribute much to the meaning in many tasks.
   - **When**: Useful for tasks like text classification, topic modeling, and search engines.
4. **Handling Missing Data**
   - **Why**: Incomplete or missing data can lead to poor model performance.
   - **When**: Critical for all tasks, especially when working with real-world datasets.
5. **Vectorization**
   - **Why**: Converts text into numerical representations (e.g., Bag of Words, TF-IDF, Word Embeddings) that machine learning models can process.
   - **When**: Essential for all tasks involving machine learning or deep learning models.
6. **Removing Punctuation and Special Characters**
   - **Why**: Punctuation and special characters often don't contribute to the meaning and can add noise.
   - **When**: Important for tasks like sentiment analysis, text classification, and machine translation.
7. **Lemmatization or Stemming**

- **Why**: Reduces words to their base forms, simplifying the vocabulary and improving consistency.
- **When**: Useful for tasks like information retrieval, text classification, and topic modeling.

8. **Handling Contractions and Abbreviations**
   - **Why**: Expands contractions (e.g., "can't" → "cannot") and abbreviations (e.g., "ASAP" → "as soon as possible") for better understanding.
   - **When**: Important for tasks involving informal text (e.g., social media analysis).
9. **Handling URLs, Mentions, and Hashtags**
   - **Why**: Social media text often contains URLs, mentions (@user), and hashtags (#topic), which need to be processed or removed.
   - **When**: Critical for social media text analysis.
10. **Text Normalization**
    - **Why**: Standardizes text (e.g., converting dates, times, and numbers to a consistent format).
    - **When**: Important for tasks involving structured data or time-sensitive analysis.

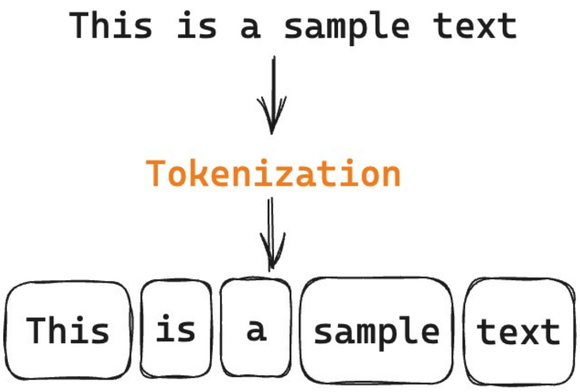
Task-Specific Importance

- **Sentiment Analysis**: Handling negations, emojis, and emoticons is crucial.
- **Machine Translation**: Sentence segmentation and POS tagging are important.
- **Named Entity Recognition (NER)**: Handling dates, times, and special characters is critical.
- **Social Media Analysis**: Handling emojis, hashtags, and informal language is essential.
- **Text Classification**: Removing stopwords, lowercasing, and vectorization are key.

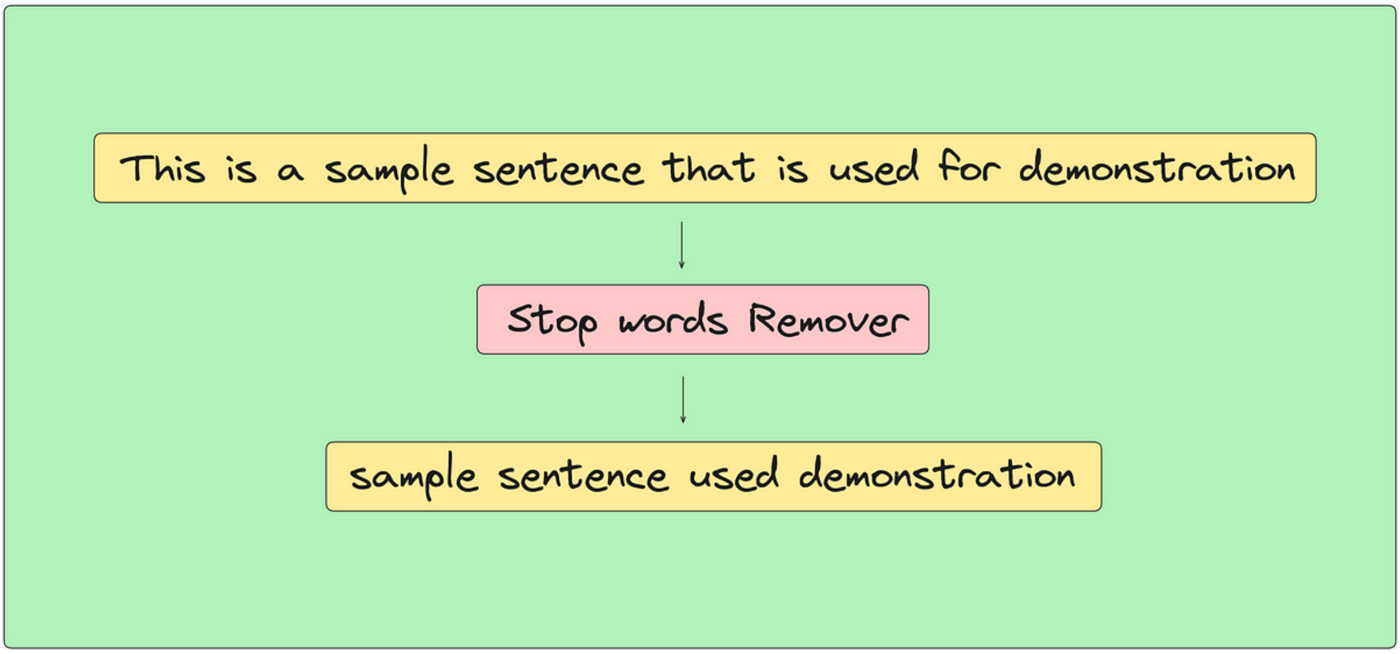
Summary

While **tokenization**, **lowercasing**, **stopword removal**, and **vectorization** are universally important, the relevance of other steps depends on the task and dataset. Always analyze your data and task requirements to determine the most critical preprocessing steps.

Prepared by: Syed Afroz Ali



Stemming		Lemmatization	
adjustable	adjust	was	(to) be
formality	formalit	better	good
formaliti	formal	meeting	meeting
airliner	airlin		



Named Entity Recognition

In the 19th century, there was something called the "cult of domesticity" for many American women. This meant that most married women were expected to stay in the home and raise children. As in other countries, American wives were very much under the control of their husband, and had almost no rights. Women who were not married had only a few jobs open to them, such as working in clothing factories and serving as maids. By the 19th century, women such as Lucretia Mott and Elizabeth Cady Stanton thought that women should have more rights. In 1848, many of these women met and agreed to fight for more rights for women, including voting. Many of the women involved in the movement for women's rights were also involved in the movement to end slavery.

Wikipedia

Tag colors:

LOCATION PERSON TERM DATE CONDITION PROCESS PEOPLE

mobidev