Natural Language Processing (NLP) is a subfield of Artificial Intelligence (AI) that focuses on the interaction between computers and human (natural) languages.

It enables machines to understand, interpret, and generate human language in a way that is meaningful and useful.

NLP lies at the intersection of **linguistics**, **computer science**, and **AI**, combining the rules and structure of **human language** with **computational algorithms** to bridge the gap between human communication and machine understanding.

A Quick Fun Introduction

Domains of AI

- Computer Vision [Image and Video - Image Processing -> CNN and RNN]
- Natural Language Processing [Text and Audio] Chatbots, Alexa. Siri
- Statistical Data [CSV,Excel -> Recommendation & Prediction using historical data]

Why Do We Need NLP?

1. **Human Language is Complex:**

- Human language is inherently ambiguous, nuanced, and diverse. Teaching machines to comprehend slang, context, sentiment, grammar, and syntax requires robust systems like NLP.

1. **Exponential Growth of Data:**

- With the explosion of digital communication, vast amounts of unstructured data (emails, social media, reviews, articles, etc.) are generated daily. NLP helps extract valuable insights from this data.

1. **Efficient Human-Machine Interaction:**

- NLP enables more natural and effective communication between humans and machines, eliminating the need for specialized commands or programming languages.

Text Classification:

- Categorizing text into predefined labels, such as spam detection or sentiment analysis.

Information Retrieval:

- Extracting and ranking relevant data or insights from large datasets based on specific user queries.

Natural Language Understanding:

- Recognizing patterns, analyzing meaning, and deriving insights from text or speech.

Natural Language Generation:

- Crafting coherent and contextually accurate responses, such as **chatbots** and **virtual assistants**.

Applications of NLP in the Real World

1. **Personal Assistants**:

- Google Assistant, Siri, and Alexa rely on NLP to understand and respond to voice commands.

1. **Machine Translation**:

- Tools like Google Translate leverage NLP for accurate translations across languages.

1. **Customer Support**:

- Chatbots powered by NLP provide 24/7 assistance for customer queries.

1. **Sentiment Analysis**:

- Businesses use NLP to analyze customer feedback and social media sentiment.

1. **Text Summarization**:

- Automatically condensing large texts into summaries (e.g., news articles).

1. **Healthcare**:

- Extracting information from patient records or assisting in diagnostics.

1. **Search Engines**:

- Enhancing user queries for more relevant search results.

Historical Background

1. **1950s – Foundational Era**:
    - Alan Turing proposed the **Turing Test** to assess a machine's ability to exhibit intelligent behavior akin to humans.

    - Early machine translation efforts, such as the **Georgetown-IBM experiment**, aimed to translate Russian to English.
2. **1960s – Rule-Based Systems**:
    - Initial NLP systems were based on hand-crafted rules and linguistic grammar.
3. **1980s – Statistical Methods**:
    - The introduction of statistical and probabilistic models revolutionized NLP, making systems more scalable and robust.
4. **2000s – Data-Driven Approaches**:
    - The rise of machine learning (ML) techniques, particularly supervised learning, boosted NLP's capabilities.
5. **2010s – Deep Learning Revolution**:
    - Neural networks, transformers (like **BERT**, **GPT**), and large language models led to unprecedented advancements in NLP.

Importance of NLP for Data Analytics

1. **Handling Unstructured Data:**
   Over 80% of data generated is unstructured, like text in emails, reviews, or social media posts. NLP helps convert this data into structured formats for analysis.

2. **Extracting Insights:**
   NLP enables sentiment analysis, topic modeling, and keyword extraction, helping analysts uncover trends, customer opinions, and patterns in textual data.

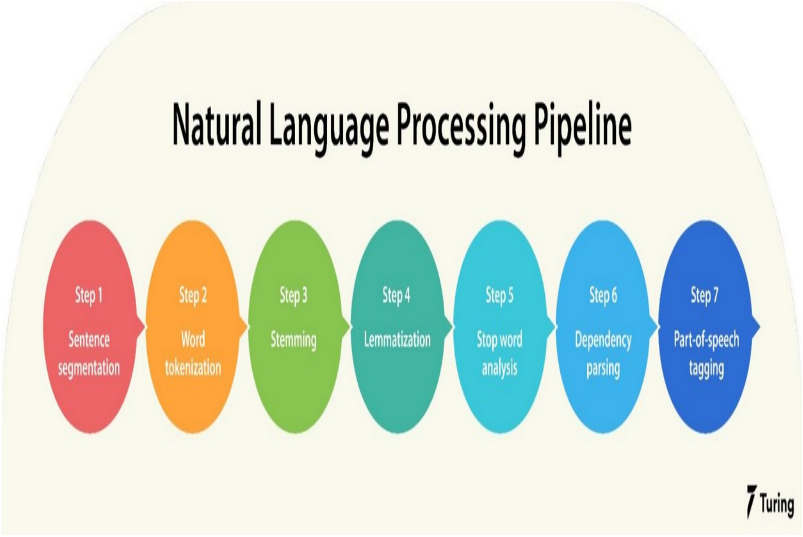
3. **Automation of Analysis:**
   Automates repetitive tasks such as text classification, summarization, or tagging, improving efficiency and scalability.

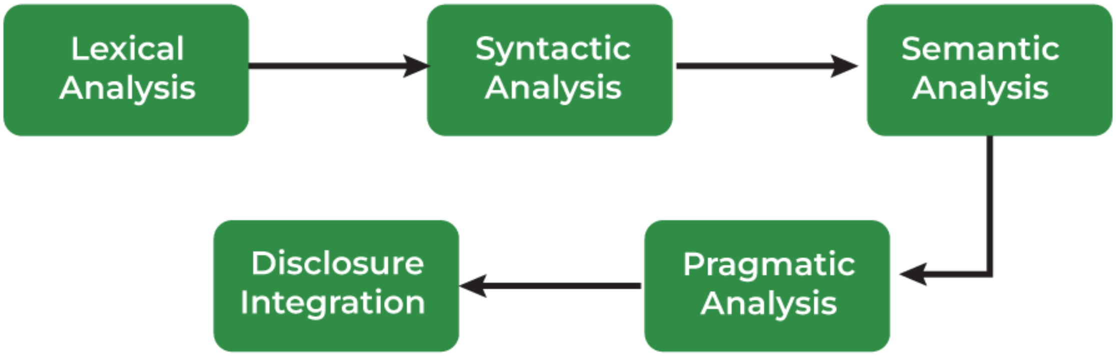
4. **Improved Decision-Making:**
   By analyzing textual feedback, NLP provides actionable insights for businesses to refine strategies and improve customer satisfaction.

5. **Enhancing Data Queries:**
   NLP-powered search and query systems allow users to interact with data analytics tools in natural language, making analytics more accessible.

Thus, NLP enhances data analytics by enabling meaningful analysis of textual data, thus broadening the scope and impact of insights.

There is no one pipeline tho:




Lexical Analysis
Syntactic Analysis
Semantic Analysis
Pragmatic Analysis
Disclosure Integration

Stages of NLP

Comprehensive NLP Pipeline

A well-defined NLP pipeline involves several processes that transform raw text into structured, meaningful data. Here's the sequence of processes commonly followed:

1. **Text Input** *Raw text data from documents, web pages, social media, etc.*
   - The text data is collected and fed into the system for processing.

---

2. **Text Preprocessing** *The text is cleaned and prepared for further analysis.*
   - **Tokenization** The text is split into smaller units like words or sentences.

   - **Lowercasing** All characters are converted to lowercase for consistency.

- **Stopword Removal** Common, non-informative words (like "and" or "the") are removed.

- **Punctuation Removal** Unnecessary punctuation marks (e.g., commas, periods) are removed.

- **Stemming/Lemmatization** Words are reduced to their root form (e.g., "running" → "run").

- **Spell Correction** Misspelled words are corrected to their correct form.

3. **Text Representation** *Converting text into a machine-readable format.*

- **Bag of Words (BoW)** The text is represented as a vector with the frequency of each word.

- **TF-IDF** Weights are applied to words based on their frequency and importance in the document.

- **Word Embeddings (e.g., Word2Vec, GloVe)** Words are converted into dense vectors that capture their meaning.

- **Sentence Embeddings (e.g., BERT)** Entire sentences are represented as vectors that capture the overall meaning.

4. **Named Entity Recognition (NER)** *Identify entities like names, dates, organizations.*

- Specific entities such as people, locations, and dates are recognized and classified.

---

5. **Syntactic Analysis** *Analyzing the grammatical structure of the text.*

- **Part-of-Speech (POS) Tagging** Each word is labeled with its grammatical role (e.g., noun, verb).

- **Dependency Parsing** The relationships between words in a sentence are identified.

- **Constituency Parsing** The sentence structure is broken down into nested components (e.g., noun phrases, verb phrases).

---

6. **Feature Engineering** *Extracting and selecting meaningful features to enhance model performance.*

- **N-gram Extraction** Word combinations (e.g., bigrams, trigrams) are generated for analysis.
- **TF-IDF Scaling** Weights are applied to words to highlight their importance in the context.
- **Custom Features** Specific features are created based on domain knowledge or problem requirements.
- **Sentiment Scores** Emotional tone (positive, negative, neutral) is extracted from the text.
- **POS Tags as Features** Part-of-speech tags are included as features to improve model predictions.
- **Text Length Metrics** Features like sentence or document length are included for better analysis.

---

7. **Task-Specific Processes** *Processing based on the desired NLP task.*

- **Sentiment Analysis** The emotional tone of the text is classified (positive, negative, neutral).
- **Text Summarization** A concise version of the text is generated.
- **Text Classification** The text is assigned to a predefined category or label.
- **Machine Translation** The text is translated from one language to another.

8. **Output Generation** *Generating meaningful insights or predictions based on the processed text.*

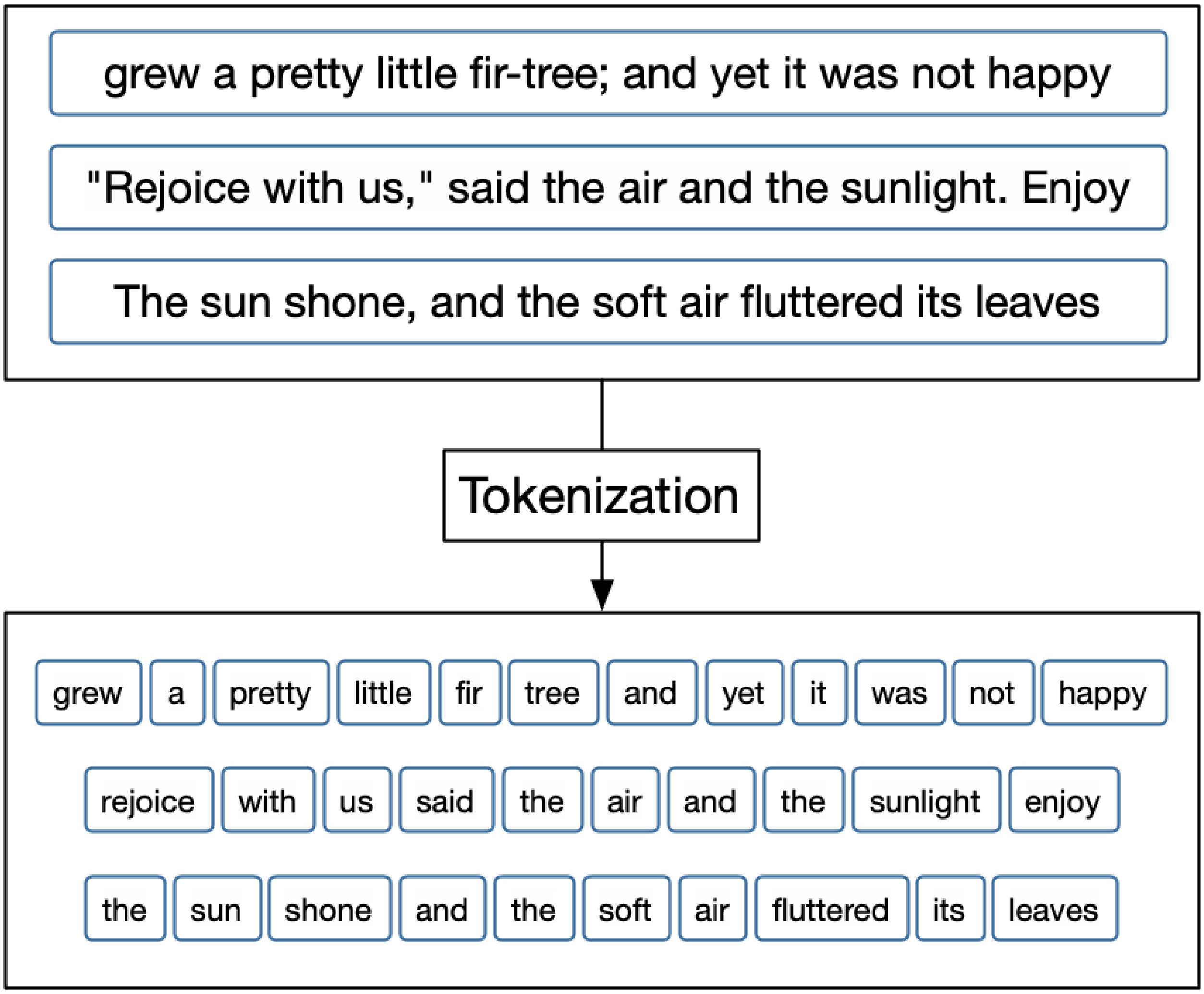
- The system produces structured output such as predictions, translations, summaries, or classifications.

Text preprocessing is the foundation of NLP, transforming raw text into a clean format suitable for analysis.

Here are key steps:

- Splitting text into smaller units, like words or sentences.
- Helps in analyzing text at the word or sentence level.

- **Example**:
  - Input: *"Natural language processing is fascinating."*
  - Output: `["Natural", "language", "processing", "is", "fascinating"]`
- Scenario:
  - In real-time, when analyzing product reviews, tokenizing the sentences allows you to understand each word individually.
  - For instance, in a customer review like "The laptop is fast and efficient," tokenization separates the words "laptop," "fast," and "efficient," which can help identify the core sentiments.


grew a pretty little fir-tree; and yet it was not happy
"Rejoice with us," said the air and the sunlight. Enjoy
The sun shone, and the soft air fluttered its leaves
Tokenization
grew
a
pretty
little
fir
tree
and
yet
it
was
not
happy
rejoice
with
us
said
the
air
and
the
sunlight
enjoy
the
sun
shone
and
the
soft
air
fluttered
its
leaves

Types of Tokenization

1. *Word Tokenization*:
   - Splitting text into individual words, typically by spaces or punctuation marks.
2. *Sentence Tokenization*:
   - Dividing text into individual sentences based on punctuation or predefined rules.

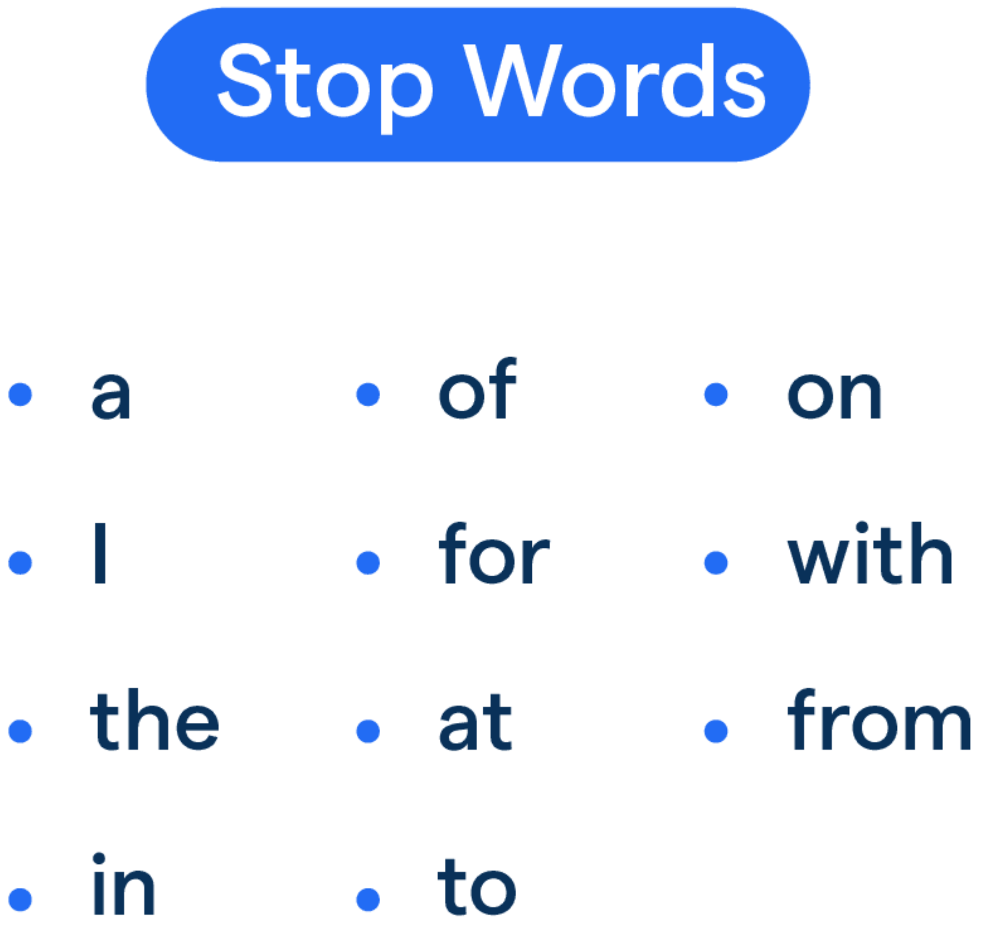
3. *Subword Tokenization*:
    - Breaking words into smaller units, such as prefixes, suffixes, or characters (used in models like BERT and GPT).
4. *Character Tokenization*:
    - Splitting text into individual characters, often used in languages with complex word structures.

After **tokenization**, the next step in text processing for NLP is typically Normalizing the Text.

Common **Text Normalization** Steps:

- Stop Word Removal
- Stemming
- Lemmatization
- Removing common common, frequently occurring words that don't carry much meaning (e.g., *is*, *the*, *and*).

- Reduces noise in text analysis.
- **Example**:
    - Input: *"The cat is on the mat."*
    - Output: `["cat", "mat"]`
- These include words like:
- **Articles**: the, a, an
- **Prepositions**: in, on, at
- **Conjunctions**: and, or, but
- **Pronouns**: I, you, he, she
- **Auxiliary verbs**: is, am, are



Stop Words

- a
- I
- the
- in
- of
- for
- at
- to
- on
- with
- from

Key Points to Consider:

1. **Language-Specific**: Stop words vary across different languages, and the list can be customized based on the context of the task.
2. **Context Matters**: In some cases, stop words may carry important meaning (e.g., in sentiment analysis), and removing them may not always be ideal.
3. **Efficiency**: Removing stop words can improve the efficiency of processing, especially for large datasets, by reducing dimensionality.
4. **Custom Stop Words**: A user-defined stop word list can be created based on domain-specific knowledge, improving the accuracy of the analysis.

- Reducing words to their root form by chopping off suffixes.

- Groups similar words for analysis (e.g., *run*, *running* → *run*).

- **Example**:
  - Input: *"running, runner"*

  - Output: `["run", "run"]`
- To simplify words we cut off parts of the word, called *affixes*, to get to its root form or base word.

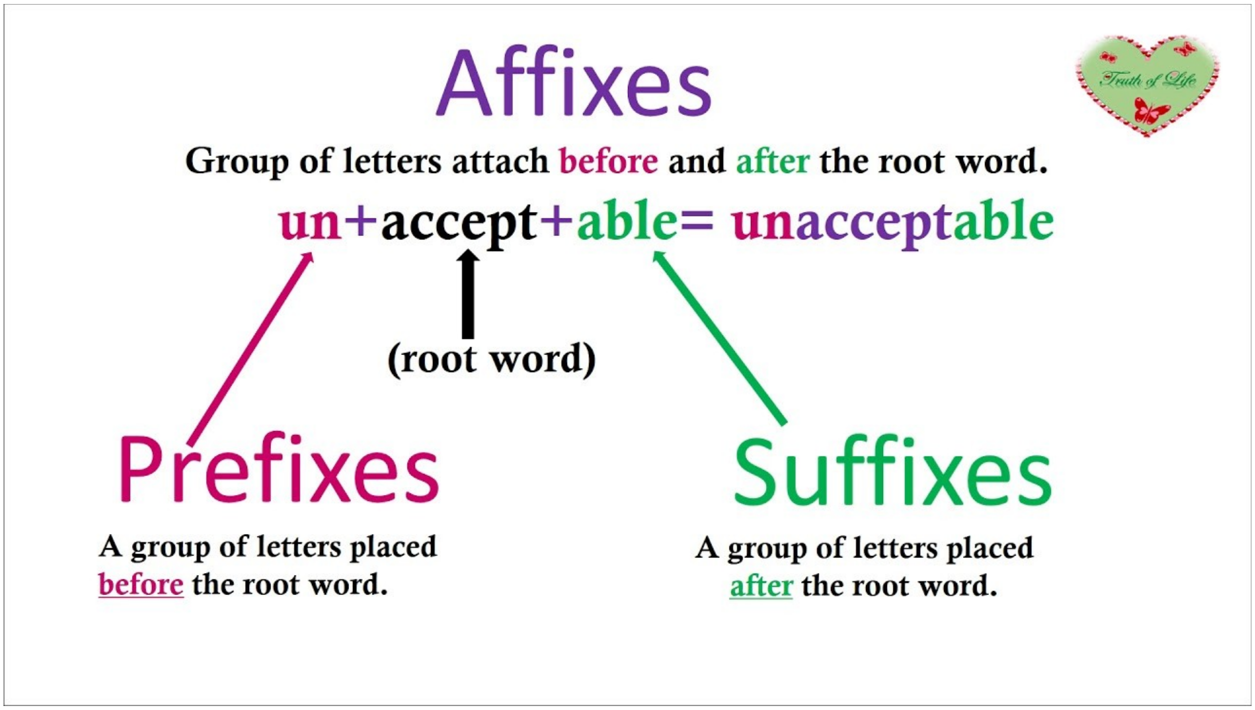
Affixes:

- Affixes are parts that are added to the base word (or root) to change its meaning. There are two main types of affixes:
  a. **Prefixes**: Added to the beginning of a word
  - (e.g., "*un-*" in "*undo*").
  a. **Suffixes**: Added to the end of a word
  - (e.g., "*-ing*" in "*running*" or "*-ed*" in "*played*").

1. **Infixes**:
   - These are inserted *within* a word. Infixes are rare in English but are found in some other languages.

   - Example: In some informal English, "*abso-bloody-lutely*" (inserting "bloody" for emphasis).
2. **Circumfixes**:
   - These are added *around* a word (both at the beginning and end). Circumfixes are not common in English but can be found in other languages.

   - Example: *en-* and *-en* are added to verbs to form the past participle (e.g., *en*light*en* from *enlighten*, meaning "to give someone greater knowledge or understanding").



Affixes

Group of letters attach before and after the root word.

un+accept+able= unacceptable

(root word)

Prefixes

A group of letters placed before the root word.

Suffixes

A group of letters placed after the root word.

How Stemming Works:

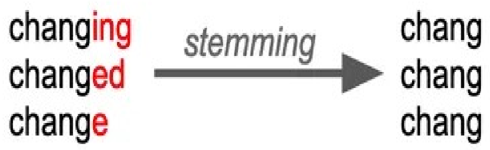
- Stemming removes these affixes to get to the base or root form of the word. For example:
  - "running" → "run"
  - "happier" → "happi"
  - "quickly" → "quick"

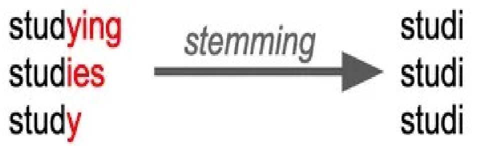
Why Use Stemming?

- Stemming helps in text analysis by treating different forms of the same word as equivalent, reducing the complexity of the text and improving tasks such as:
  - **Search**: Finding related words or content regardless of their forms.
  - **Text Classification**: Grouping similar meanings together, no matter how the word is written.

Challenges

- Stemming can sometimes produce words that don’t exist or make sense (e.g., "happier" becomes "happi"), but it simplifies language for analysis, especially when accuracy isn’t the priority over generalization.





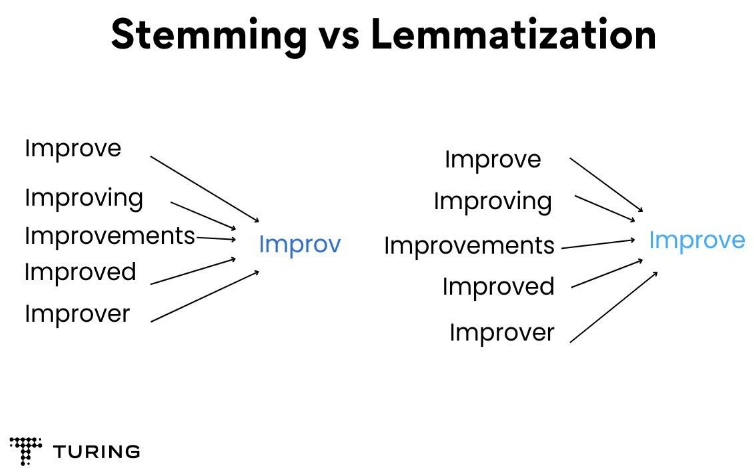
- Converting words to their base or dictionary form, considering grammar.

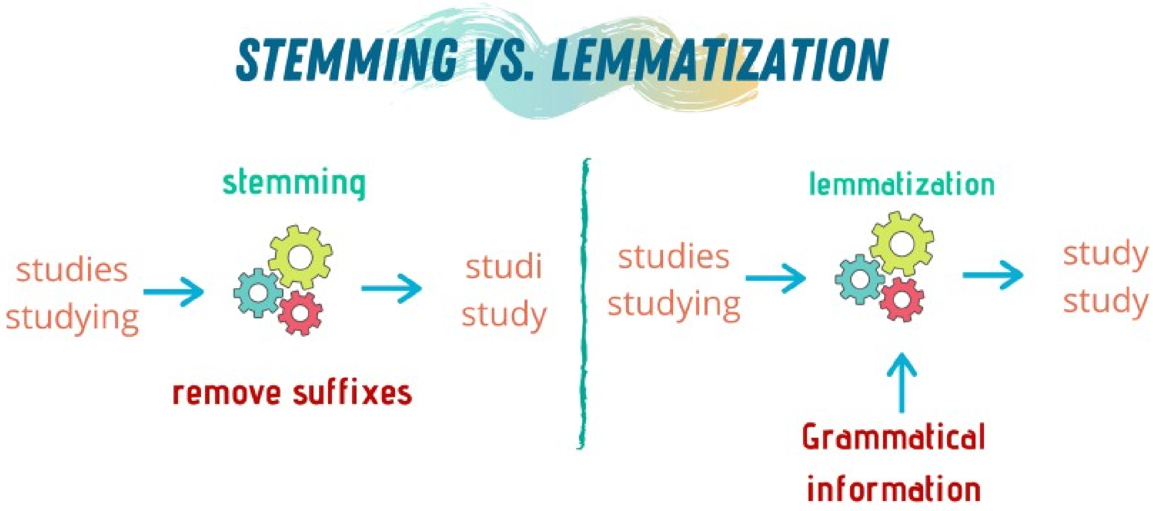
- Provides contextually meaningful base forms.

- **Example**:
  - Input: *"better"*
  - Output: `["good"]`

  Lemmatization is the process of reducing a word to its base form (called a *lemma*), ensuring that the word is returned to its correct dictionary form.

  Unlike stemming, which may cut off parts of words, lemmatization considers the meaning and context of the word to ensure the base form is grammatically correct.


Stemming vs Lemmatization



How Lemmatization Works:

- Lemmatization uses a dictionary and part-of-speech (POS) tagging to determine the correct base form of a word.
- Part-of-Speech (POS) refers to the grammatical categories that words belong to, such as noun, verb, adjective, adverb, etc., based on their function in a sentence.
  - Example: "was" (verb) → "be"
  - Example: "geese" (plural noun) → "goose"

Why Use Lemmatization?

- Lemmatization is more accurate than stemming because it produces valid words, making it better suited for tasks like:
  - **Text Classification**: Grouping similar words together without losing meaning.
  - **Sentiment Analysis**: Understanding the context and ensuring proper interpretation of words.
  - **Search Engines**: Improving results by matching terms to their base form.

Challenges

- Lemmatization requires more computational resources than stemming because it uses a dictionary and grammatical context.

Python libraries like NLTK and spaCy are powerful tools for text preprocessing in NLP tasks.

1. Tokenization

- Word Tokenization: Breaks the text into words.
- Sentence Tokenization: Splits the text into sentences.
- Character Tokenization: Breaks the text into individual characters.
- Whitespace Tokenization: Splits based on spaces between words.
- Punctuation-Aware Tokenization: Treats punctuation marks as separate tokens.

```
# pip install nltk
```

```
import nltk
# nltk.download('punkt')  # # This downloads the necessary data for
tokenization

from nltk.tokenize import word_tokenize

text = "I love programming!"
words = nltk.word_tokenize(text)  # Tokenizing into words
print(words)

['I', 'love', 'programming', '!']
```

- `NLTK` (Natural Language Toolkit) is a popular library in Python used for Natural Language Processing (NLP).
- `punkt` is a pre-trained tokenizer model available in NLTK, which is used for breaking down text into words or sentences. It is a necessary resource for tokenization tasks.

Sentence tokenization splits a text into sentences.

```

from nltk.tokenize import sent_tokenize

# Sample text
text = "I love programming. It's fun!"

# Sentence Tokenization
sentences = sent_tokenize(text)
print("Sentence Tokens:", sentences)

Sentence Tokens: ['I love programming.', "It's fun!"]
```

Character tokenization splits text into individual characters.

```
# Example word
word = "programming"

# Character Tokenization (manual splitting)
char_tokens = list(word)

# Print the result
print("Character Tokenization:", char_tokens)

Character Tokenization: ['p', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'i',
'n', 'g']
```

Whitespace tokenization splits the text based on spaces (whitespace characters).

```
# Example sentence
text = "I love programming."
```

```python
# Whitespace Tokenization (splitting based on space)
whitespace_tokens = text.split()

# Print the result
print("Whitespace Tokenization:", whitespace_tokens)
```

```
Whitespace Tokenization: ['I', 'love', 'programming.']
```

Punctuation-Aware Tokenization handles punctuation separately from words.

```python
# Example sentence with punctuation
text = "Hello! How are you doing?"

# Word Tokenization (with punctuation handling)
word_tokens_with_punct = word_tokenize(text)

# Print the result
print("Punctuation-Aware Tokenization:", word_tokens_with_punct)
```

```
Punctuation-Aware Tokenization: ['Hello', '!', 'How', 'are', 'you',
'doing', '?']
```

```python
from nltk.corpus import stopwords

# Example sentence (tokens)
tokens = ["I", "love", "programming", "and", "it", "is", "fun"]

# Get the list of stop words in English
stop_words = set(stopwords.words('english'))

# Print the stop words list (optional)
print("Stop Words List:", stop_words)
```

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

```
nltk.download('stopwords')  # Download the stop words list
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\devid\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
True
```

```
# Remove stop words
filtered_tokens = [word for word in tokens if word.lower() not in
stop_words]

# Print the filtered tokens
print("Tokens after Stop Word Removal:", filtered_tokens)
```

```
Tokens after Stop Word Removal: ['love', 'programming', 'fun']
```

Understanding **List Comprehension**

`[word for word in tokens if word.lower() not in stop_words]`

This is called List Comprehension in Python, which is a concise way to create a new list by iterating over an existing iterable and applying a condition.

Let's observe an example:

`new_list = [expression for item in iterable if condition]`

```
basket = ["apple", "banana", "orange", "kiwi", "grape"]

long_fruits = []

for fruit in basket:
    if len(fruit) > 5:
        long_fruits.append(fruit)

print(long_fruits)
```

```
['banana', 'orange']
```

```
# With List Comprehension

basket = ["apple", "banana", "orange", "kiwi", "grape"]
long_fruits = [fruit for fruit in basket if len(fruit) > 5]
print(long_fruits)
```

```
['banana', 'orange']

# another example

numbers = [1, 2, 3, 4, 5]

doubled = [num * 2 for num in numbers]

print(doubled)

[2, 4, 6, 8, 10]
```

Removing Punctuation

Punctuation can often be noise in text processing tasks, so we will remove them.

```
import string

# Removing punctuation
reviews_no_punct = [[word for word in tokens if word not in
string.punctuation] for tokens in filtered_tokens]
print(reviews_no_punct)

[['l', 'o', 'v', 'e'], ['p', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'i',
'n', 'g'], ['f', 'u', 'n']]
```

Removing Special Characters Special characters such as "#", "@", or emoji may need to be removed or treated based on their relevance.

```
import re

# Removing special characters
def remove_special_chars(tokens):
    return [re.sub(r'[^A-Za-z0-9]+', '', word) for word in tokens]

reviews_cleaned = [remove_special_chars(tokens) for tokens in
filtered_tokens]
print(reviews_cleaned)

[['l', 'o', 'v', 'e'], ['p', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'i',
'n', 'g'], ['f', 'u', 'n']]

from nltk.stem import PorterStemmer  # Stemming algorithm
from nltk.tokenize import word_tokenize  # For breaking text into
tokens

# The Porter Stemmer is a popular algorithm for stemming in English.
stemmer = PorterStemmer()

# Sample Text
text = "I am loving the process of learning and understanding NLP
```

```
concepts."

# Tokenize the Text
tokens = word_tokenize(text)

print("Tokens:", tokens)

Tokens: ['I', 'am', 'loving', 'the', 'process', 'of', 'learning', 
'and', 'understanding', 'NLP', 'concepts', '.']

# apply stemming to each word in the list of tokens

stemmed_words = [stemmer.stem(word) for word in tokens]
print("Stemmed Words:", stemmed_words)

Stemmed Words: ['i', 'am', 'love', 'the', 'process', 'of', 'learn', 
'and', 'understand', 'nlp', 'concept', '.']
```

WordNet

- **wordnet** refers to the WordNet lexical database, a large database of English words developed by Princeton University. It groups words into sets of synonyms called synsets and provides semantic relationships between these sets.

```
# Import the Required Libraries

from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk.corpus import wordnet

# We need WordNet and Punkt for tokenization and lemmatization

nltk.download('wordnet')  # For lexical database
nltk.download('omw-1.4')  # Optional for extended multilingual support
nltk.download('punkt')    # For tokenization

[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\devid\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\devid\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\devid\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!

True

# Initialize the Lemmatizer
lemmatizer = WordNetLemmatizer()

# Smaple text
```

```
text = "The leaves on the tree were falling. She was running quickly
but got tired."

# Break sentence into words
tokens = word_tokenize(text)
print("Tokens:", tokens)
```

```
Tokens: ['The', 'leaves', 'on', 'the', 'tree', 'were', 'falling', '.',
'She', 'was', 'running', 'quickly', 'but', 'got', 'tired', '.']
```

```
# apply lemmatization
lemmatized_words = [lemmatizer.lemmatize(word) for word in tokens]

print("Lemmatized Words:", lemmatized_words)
```

```
Lemmatized Words: ['The', 'leaf', 'on', 'the', 'tree', 'were',
'falling', '.', 'She', 'wa', 'running', 'quickly', 'but', 'got',
'tired', '.']
```

Lemmatization with POS Tagging

- When paired with Part-of-Speech (POS) tagging, the lemmatizer gains context about how the word is used (noun, verb, adjective, etc.), resulting in more accurate transformations.

```
#Lemmatization (with POS tagging)

# Import necessary libraries
from nltk.corpus import wordnet # for WordNet-compatible POS tags
from nltk.tag import pos_tag # to assign POS tags to words
from nltk.stem import WordNetLemmatizer # from NLTK for lemmatization

# Function to map POS tags to WordNet tags
def get_wordnet_pos(word):
    tag = pos_tag([word])[0][1][0].upper()  # Get the POS tag's first
letter
    tag_dict = {"J": wordnet.ADJ, "N": wordnet.NOUN, "V":
wordnet.VERB, "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)  # Default to noun if tag
is not in the dictionary
```

Code breakdown:

- `tag = pos_tag([word])[0][1][0].upper()`
- This code helps identify the **type of word** (like noun, verb, adjective, etc.) by analyzing its grammar role, which is called a **Part-of-Speech (POS) tag**.

Let's break it down step-by-step.

- **Input**: A word wrapped in a list. Example: `["running"]`.
- **Output**: A list of tuples where:

  - The first element is the **word**.
  - The second element is its **POS tag**.

1. pos_tag([word]): Returns [('running', 'VBG')]
   - V: verb; BG: verb is in the gerund or present participle form
2. Extract Tuple: [0] → ('running', 'VBG').
3. Extract POS Tag: [1] → 'VBG'
4. Extract First Character: [0] → 'V'
   - [0][1][0] is a way of indexing into nested data structures (like lists of tuples)
5. **Convert to Uppercase: 'V'**

- Other Verb Tags:

  VB: Base form of a verb. Example: "run" in "I will run."

  VBD: Past tense of a verb. Example: "ran" in "I ran yesterday."

  VBN: Past participle of a verb. Example: "run" in "I have run."

  VBP: Present tense, non-3rd person singular verb. Example: "run" in "I run every day."

  VBZ: Present tense, 3rd person singular verb. Example: "runs" in "He runs daily."

Similarly there are noun (N) tags, adverb (R) tags, adjective (J) tags etc.

- Code Breakdown:
- `tag_dict = {"J": wordnet.ADJ, "N": wordnet.NOUN, "V": wordnet.VERB, "R": wordnet.ADV}`
- This line of code creates a dictionary named tag_dict, which maps certain characters representing parts of speech (POS) to their corresponding **WordNet** constants.
- "J": wordnet.ADJ,
- "N": wordnet.NOUN,
- "V": wordnet.VERB,
- "R": wordnet.ADV
- `tag_dict.get(tag, wordnet.NOUN)`
- This line retrieves a value from the tag_dict dictionary for a given key (tag)
- If the key does not exist in the dictionary, it returns a default value, which in this case is `wordnet.NOUN`

Lemmatization with POS tagging

```
# Sample text
text = "The leaves are falling quickly from the trees, and the
children are happily playing."

# Tokenize the text
tokens = word_tokenize(text)

# Initialize the lemmatizer
lemmatizer = WordNetLemmatizer()

# Perform lemmatization with POS tagging
lemmatized_tokens = [lemmatizer.lemmatize(word, get_wordnet_pos(word))
for word in tokens]

# Output the result
print("Original Tokens:", tokens)
print("Lemmatized Tokens:", lemmatized_tokens)
```

```
Original Tokens: ['The', 'leaves', 'are', 'falling', 'quickly',
'from', 'the', 'trees', ',', 'and', 'the', 'children', 'are',
'happily', 'playing', '.']
Lemmatized Tokens: ['The', 'leaf', 'be', 'fall', 'quickly', 'from',
'the', 'tree', ',', 'and', 'the', 'child', 'be', 'happily', 'play',
'.']
```

```
# Find synonyms of a word
synonyms = wordnet.synsets("run")
print(synonyms[0].definition())
# find synonyms and displays the definition of the first synonym for
the word "run"
```

```
a score in baseball made by a runner touching all four bases safely
```

```
# Import libraries
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
from nltk import pos_tag, download
from nltk.corpus import wordnet

# Download necessary datasets
#download('punkt')      # For tokenization
#download('stopwords')  # For stop words
#download('wordnet')    # For lemmatization
#download('omw-1.4')    # For extended lemmatization support
#download('averaged_perceptron_tagger')  # For POS tagging

# Sample text
text = "The leaves are falling quickly from the trees, and the
```

```
children are happily playing."

# Step 1: Tokenization
tokens = word_tokenize(text)
print("Tokens:", tokens)

# Step 2: Stop Word Removal
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word.lower() not in
stop_words]
print("Filtered:", filtered_tokens)

# Step 3: Stemming
stemmer = PorterStemmer()
stemmed_tokens = [stemmer.stem(word) for word in filtered_tokens]
print("Stemmed:", stemmed_tokens)

# Step 4: Lemmatization (without POS tagging)
lemmatizer = WordNetLemmatizer()
lemmatized_tokens = [lemmatizer.lemmatize(word) for word in
filtered_tokens]
print("Lemmatized (without POS):", lemmatized_tokens)

# Step 5: Lemmatization (with POS tagging)
def get_wordnet_pos(word):
    tag = pos_tag([word])[0][1][0].upper()  # Get the POS tag
    tag_dict = {"J": wordnet.ADJ, "N": wordnet.NOUN, "V":
wordnet.VERB, "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)  # Default to noun

lemmatized_tokens_with_pos = [lemmatizer.lemmatize(word,
get_wordnet_pos(word)) for word in filtered_tokens]
print("Lemmatized (with POS):", lemmatized_tokens_with_pos)
```

```
Tokens: ['The', 'leaves', 'are', 'falling', 'quickly', 'from', 'the',
'trees', ',', 'and', 'the', 'children', 'are', 'happily', 'playing',
'.']
Filtered: ['leaves', 'falling', 'quickly', 'trees', ',', 'children',
'happily', 'playing', '.']
Stemmed: ['leav', 'fall', 'quickli', 'tree', ',', 'children',
'happili', 'play', '.']
Lemmatized (without POS): ['leaf', 'falling', 'quickly', 'tree', ',',
'child', 'happily', 'playing', '.']
Lemmatized (with POS): ['leaf', 'fall', 'quickly', 'tree', ',',
'child', 'happily', 'play', '.']
```

Using Spacy

Objective:

- Analyze sentiment a dataset of tweets to classify their sentiment as positive, neutral, or negative.
- You'll preprocess the tweets, extract features, and train a machine learning model to **predict sentiment**.
- Finally, you'll evaluate the model and test it on new tweets.

Possible Execution:

1. Get the Dataset

- Use a pre-labeled dataset from sources like Kaggle or UCI Machine Learning Repository.
- Scrape tweets using the Twitter API and manually label them.
- Download datasets shared in `.csv` or `.txt` formats from online repositories.

2. Load the Dataset

- Import the dataset into your Python environment using libraries like `pandas`. Ensure sentiment labels (e.g., positive, neutral, negative) are included.

3. Tokenize Text

- Split each tweet into individual words or tokens for easier processing.

4. Remove Noise

- Clean the text by removing:
  - URLs
  - Mentions (`@username`)
  - Hashtags (`#example`)
  - Emojis
  - Special characters

5. Normalize Text

- Convert all text to lowercase.
- Remove common stop words like "the", "is", and "and" to focus on meaningful words.

6. Perform Stemming or Lemmatization

- Reduce words to their root form (e.g., "running" → "run") to maintain uniformity in the dataset.

7. Feature Extraction

- Convert textual data into numerical features using:

  - **TF-IDF (Term Frequency-Inverse Document Frequency)**

  - **Word embeddings** (e.g., Word2Vec, GloVe).

8. Train a Model

- Use a machine learning algorithm like logistic regression, support vector machines (SVM), or neural networks to train a sentiment classifier.

9. Evaluate the Model

- Test the trained model on a validation set.

- Check performance metrics such as accuracy, precision, recall, and F1-score.

10. Visualize Results

- Create graphs to display:
  - Sentiment distribution in the dataset.

  - Model performance metrics like confusion matrix or ROC curves.

11. Analyze a New Tweet

- Test the trained model on new or unseen tweets.

- Predict their sentiment and validate the model's real-world usability.

Additional Resources:

- https://developers.google.com/machine-learning/guides/text-classification/
- http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/
- https://huggingface.co/learn/nlp-course/en/chapter1/1
- https://www.datacamp.com/tutorial/nlp-with-pytorch-a-comprehensive-guide
- https://campus.datacamp.com/courses/natural-language-processing-with-spacy/introduction-to-nlp-and-spacy

Now it's your turn!

Task 1: **Individual Exercise: Preprocess Your Own Text**

1. **Collect Data:** Copy a short paragraph (3-5 sentences) from a website or social media.

2. **Tokenize:** Split the text into individual words.

3. **Remove Stop Words:** Filter out common words like "the," "is," etc.

4. **Stem:** Reduce words to their root forms.

5. **Lemmatize:** Use POS tagging for context-aware lemmatization.

6. **Submit Results:** Share your paragraph, final tokens, and a brief reflection.

```
# Import necessary libraries
from nltk.tokenize import word_tokenize  # For splitting text into
tokens
from nltk.corpus import stopwords        # For stop word removal
from nltk.stem import PorterStemmer      # For stemming words
from nltk.stem import WordNetLemmatizer  # For lemmatizing words
from nltk import pos_tag, download       # For POS tagging
from nltk.corpus import wordnet          # For WordNet integration
(lemmatization)
import re                                # For special character
removal

# Download necessary datasets if not already downloaded
download('punkt')                       # For tokenization
download('stopwords')                   # For accessing stop word list
download('wordnet')                     # For lemmatization support
download('averaged_perceptron_tagger')  # For POS tagging

# Sample text examples (students can replace these with their own)
texts = [
    "AI is transforming industries rapidly! From healthcare to
education, its applications are endless.",
    "The sun rises in the east, setting a golden glow over the
horizon.",
    "John's cat, which is black and white, loves to play with its
toys.",
    "Weather forecasts have improved dramatically due to AI-powered
models."
]

# Iterate over each example text
for idx, text in enumerate(texts):
    print(f"\n**Example {idx + 1}: Original Text**")
    print(text)

    # Step 1: Tokenization - Splitting text into individual words
    tokens = word_tokenize(text)
    print("\nTokens:", tokens)

    # Step 2a: Stop Word Removal - Removing common words that add no
semantic value
    stop_words = set(stopwords.words('english'))  # Load English stop
words
    tokens_without_stopwords = [word for word in tokens if
word.lower() not in stop_words]
    print("\nTokens After Stop Word Removal:",
tokens_without_stopwords)

    # Step 2b: Special Character Removal - Removing punctuation or
```

```
special symbols
    tokens_cleaned = [re.sub(r'[^\w\s]', '', word) for word in
tokens_without_stopwords if re.sub(r'[^\w\s]', '', word)]
    print("\nTokens After Special Character Removal:", tokens_cleaned)

    # Step 3: Stemming - Reducing words to their root forms
    stemmer = PorterStemmer()
    stemmed_tokens = [stemmer.stem(word) for word in tokens_cleaned]
    print("\nStemmed Tokens:", stemmed_tokens)

    # Step 4: Lemmatization with POS Tagging - Context-aware word
normalization
    lemmatizer = WordNetLemmatizer()

    # Helper function to get WordNet POS tag
    def get_wordnet_pos(word):
        tag = pos_tag([word])[0][1][0].upper()  # Get the first letter
of POS tag
        tag_dict = {"J": wordnet.ADJ, "N": wordnet.NOUN, "V":
wordnet.VERB, "R": wordnet.ADV}
        return tag_dict.get(tag, wordnet.NOUN)  # Default to noun if
tag is not found

    # Apply lemmatization
    lemmatized_tokens = [lemmatizer.lemmatize(word,
get_wordnet_pos(word)) for word in tokens_cleaned]
    print("\nLemmatized Tokens (with POS):", lemmatized_tokens)

    # Final Reflection
    print(f"\n**Final Results for Example {idx + 1}:**")
    print("Original Tokens:", tokens)
    print("Tokens After Stop Word Removal:", tokens_without_stopwords)
    print("Tokens After Special Character Removal:", tokens_cleaned)
    print("Stemmed Tokens:", stemmed_tokens)
    print("Lemmatized Tokens:", lemmatized_tokens)

print("\nTask Complete! Replace the example texts with your own for
practice.")
```

```
**Example 1: Original Text**
AI is transforming industries rapidly! From healthcare to education,
its applications are endless.

Tokens: ['AI', 'is', 'transforming', 'industries', 'rapidly', '!',
'From', 'healthcare', 'to', 'education', ',', 'its', 'applications',
'are', 'endless', '.']

Tokens After Stop Word Removal: ['AI', 'transforming', 'industries',
'rapidly', '!', 'healthcare', 'education', ',', 'applications',
'endless', '.']

Tokens After Special Character Removal: ['AI', 'transforming',
'industries', 'rapidly', 'healthcare', 'education', 'applications',
'endless']

Stemmed Tokens: ['ai', 'transform', 'industri', 'rapidli',
'healthcar', 'educ', 'applic', 'endless']

Lemmatized Tokens (with POS): ['AI', 'transform', 'industry',
'rapidly', 'healthcare', 'education', 'application', 'endless']

**Final Results for Example 1:**
Original Tokens: ['AI', 'is', 'transforming', 'industries', 'rapidly',
'!', 'From', 'healthcare', 'to', 'education', ',', 'its',
'applications', 'are', 'endless', '.']
Tokens After Stop Word Removal: ['AI', 'transforming', 'industries',
'rapidly', '!', 'healthcare', 'education', ',', 'applications',
'endless', '.']
Tokens After Special Character Removal: ['AI', 'transforming',
'industries', 'rapidly', 'healthcare', 'education', 'applications',
'endless']
Stemmed Tokens: ['ai', 'transform', 'industri', 'rapidli',
'healthcar', 'educ', 'applic', 'endless']
Lemmatized Tokens: ['AI', 'transform', 'industry', 'rapidly',
'healthcare', 'education', 'application', 'endless']

**Example 2: Original Text**
The sun rises in the east, setting a golden glow over the horizon.

Tokens: ['The', 'sun', 'rises', 'in', 'the', 'east', ',', 'setting',
'a', 'golden', 'glow', 'over', 'the', 'horizon', '.']

Tokens After Stop Word Removal: ['sun', 'rises', 'east', ',',
'setting', 'golden', 'glow', 'horizon', '.']

Tokens After Special Character Removal: ['sun', 'rises', 'east',
'setting', 'golden', 'glow', 'horizon']

Stemmed Tokens: ['sun', 'rise', 'east', 'set', 'golden', 'glow',
'horizon']

Lemmatized Tokens (with POS): ['sun', 'rise', 'east', 'set', 'golden',
'glow', 'horizon']

**Final Results for Example 2:**
Original Tokens: ['The', 'sun', 'rises', 'in', 'the', 'east', ',',
'setting', 'a', 'golden', 'glow', 'over', 'the', 'horizon', '.']
Tokens After Stop Word Removal: ['sun', 'rises', 'east', ',',
'setting', 'golden', 'glow', 'horizon', '.']
```

```
Tokens After Special Character Removal: ['sun', 'rises', 'east',
'setting', 'golden', 'glow', 'horizon']
Stemmed Tokens: ['sun', 'rise', 'east', 'set', 'golden', 'glow',
'horizon']
Lemmatized Tokens: ['sun', 'rise', 'east', 'set', 'golden', 'glow',
'horizon']

**Example 3: Original Text**
John's cat, which is black and white, loves to play with its toys.

Tokens: ['John', "'s", 'cat', ',', 'which', 'is', 'black', 'and',
'white', ',', 'loves', 'to', 'play', 'with', 'its', 'toys', '.']

Tokens After Stop Word Removal: ['John', "'s", 'cat', ',', 'black',
'white', ',', 'loves', 'play', 'toys', '.']

Tokens After Special Character Removal: ['John', 's', 'cat', 'black',
'white', 'loves', 'play', 'toys']

Stemmed Tokens: ['john', 's', 'cat', 'black', 'white', 'love', 'play',
'toy']

Lemmatized Tokens (with POS): ['John', 's', 'cat', 'black', 'white',
'love', 'play', 'toy']

**Final Results for Example 3:**
Original Tokens: ['John', "'s", 'cat', ',', 'which', 'is', 'black',
'and', 'white', ',', 'loves', 'to', 'play', 'with', 'its', 'toys',
'.']
Tokens After Stop Word Removal: ['John', "'s", 'cat', ',', 'black',
'white', ',', 'loves', 'play', 'toys', '.']
Tokens After Special Character Removal: ['John', 's', 'cat', 'black',
'white', 'loves', 'play', 'toys']
Stemmed Tokens: ['john', 's', 'cat', 'black', 'white', 'love', 'play',
'toy']
Lemmatized Tokens: ['John', 's', 'cat', 'black', 'white', 'love',
'play', 'toy']

**Example 4: Original Text**
Weather forecasts have improved dramatically due to AI-powered models.

Tokens: ['Weather', 'forecasts', 'have', 'improved', 'dramatically',
'due', 'to', 'AI-powered', 'models', '.']

Tokens After Stop Word Removal: ['Weather', 'forecasts', 'improved',
'dramatically', 'due', 'AI-powered', 'models', '.']

Tokens After Special Character Removal: ['Weather', 'forecasts',
'improved', 'dramatically', 'due', 'AIpowered', 'models']
```

```
Stemmed Tokens: ['weather', 'forecast', 'improv', 'dramat', 'due', 
'aipow', 'model']

Lemmatized Tokens (with POS): ['Weather', 'forecast', 'improve', 
'dramatically', 'due', 'AIpowered', 'model']

**Final Results for Example 4:**
Original Tokens: ['Weather', 'forecasts', 'have', 'improved', 
'dramatically', 'due', 'to', 'AI-powered', 'models', '.']
Tokens After Stop Word Removal: ['Weather', 'forecasts', 'improved', 
'dramatically', 'due', 'AI-powered', 'models', '.']
Tokens After Special Character Removal: ['Weather', 'forecasts', 
'improved', 'dramatically', 'due', 'AIpowered', 'models']
Stemmed Tokens: ['weather', 'forecast', 'improv', 'dramat', 'due', 
'aipow', 'model']
Lemmatized Tokens: ['Weather', 'forecast', 'improve', 'dramatically', 
'due', 'AIpowered', 'model']

Task Complete! Replace the example texts with your own for practice.

[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\devid\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\devid\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\devid\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\devid\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

1. What is the purpose of text preprocessing in NLP?
Answer:
Text preprocessing prepares raw text data for analysis by cleaning, normalizing, and structuring it. It ensures consistency, removes noise (e.g., stop words, special characters), and converts text into a format suitable for NLP algorithms.

---

2. Explain the difference between stemming and lemmatization.
Answer:

- **Stemming:** Reduces words to their root form by chopping off suffixes, often resulting in non-dictionary words (e.g., *"running"* → *"run"*).

- **Lemmatization:** Normalizes words to their base or dictionary form using linguistic rules and context (e.g., *"better"* → *"good"*).

---

3. How do tokenization and stop word removal contribute to preprocessing?
Answer:

- **Tokenization:** Splits text into smaller units like words, sentences, or subwords, making it manageable for analysis.
- **Stop Word Removal:** Removes common words (e.g., *"the," "and," "is"*) that do not contribute significant meaning to the text, reducing noise.

---

4. Why is lowercasing important in text preprocessing?
Answer:
Lowercasing ensures uniformity by converting all characters to lowercase, preventing duplication and improving text consistency. For example, *"Apple"* and *"apple"* would be treated as the same word after lowercasing.

---

5. Write Python code to clean a text string by removing stop words and special characters.
Answer:

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

# Sample text

text = "Hello! NLP is amazing, isn't it?"

# Tokenization

tokens = word_tokenize(text)

# Stop Word Removal

stop_words = set(stopwords.words('english')) tokens_without_stopwords = [word for word in tokens if word.lower() not in stop_words]

# Special Character Removal

cleaned_tokens = [re.sub(r'[^\w\s]', '', word) for word in tokens_without_stopwords if re.sub(r'[^\w
```