# **Text Preprocessing in NLP**

## **1️⃣ Lesson Introduction**

**Mini-Agenda for Today:**

1. Text Cleaning
2. Tokenization: Word, Sentence, Subword
3. Normalization techniques
4. Stopwords, Stemming & Lemmatization
5. Spelling Correction
6. Libraries for Text Cleaning: NLTK, spaCy, TextBlob, clean-text
7. Practical: Build a basic text preprocessing pipeline
8. Business use cases
9. Practice exercises
10. Case study / mini-project
11. Recap, quiz, and interview questions

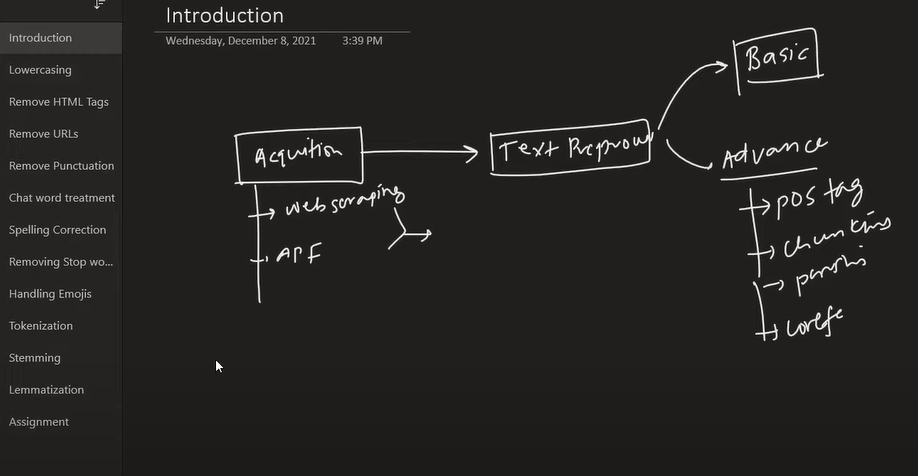
## **2️⃣ Real-World Hook**

* **Question:** “How can search engines, chatbots, or recommendation systems understand messy human text?”
* **Problem:** Raw text from users contains inconsistent capitalization, punctuation, numbers, URLs, and typos.
* **Solution:** **Text Preprocessing** cleans, normalizes, and structures text for NLP applications.

## **3️⃣ Introduce Topic as a Solution**

Text preprocessing transforms unstructured text into a **clean and analyzable format**, enabling:

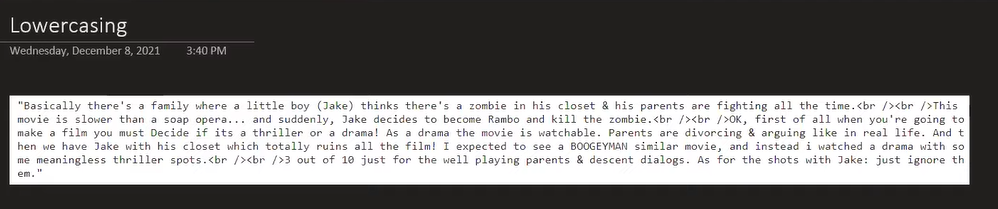
* Accurate sentiment analysis
* Improved search and retrieval
* Better machine learning model performance

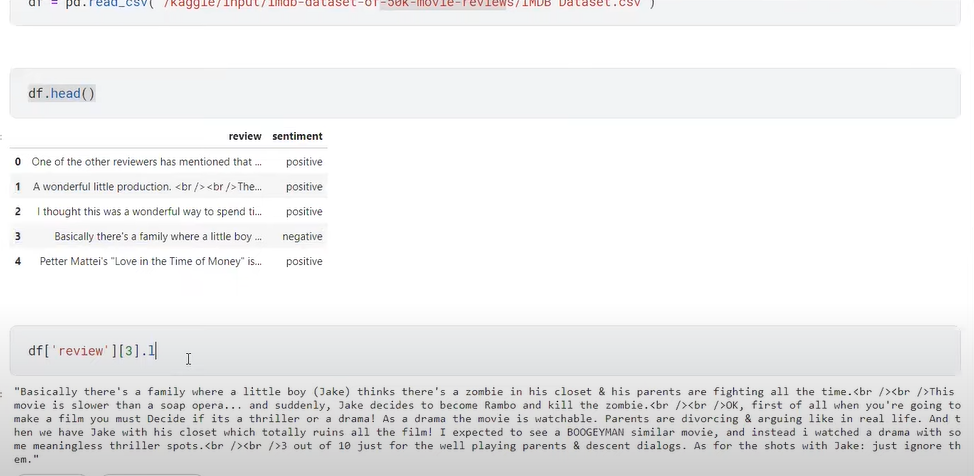


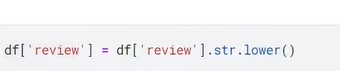
## **4️⃣ Theory Explanation**

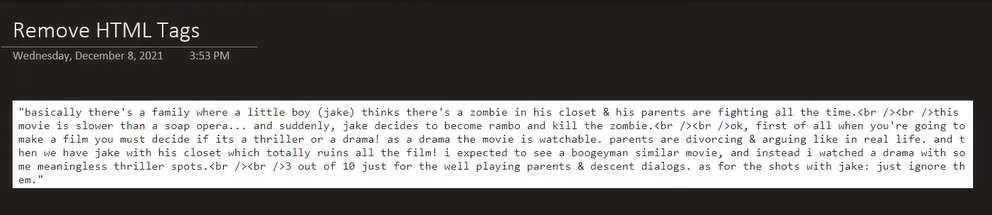
### **4.1 Text Cleaning**

* **What:** Removing noise from text
* **Why:** Clean text improves model accuracy
* **How:** Lowercasing, removing punctuation, numbers, URLs, emojis





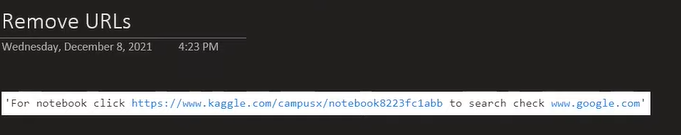


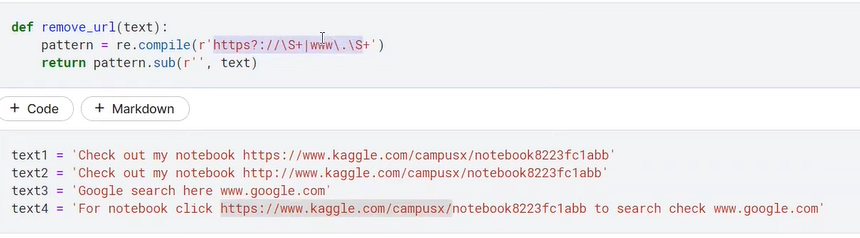
**2.** 

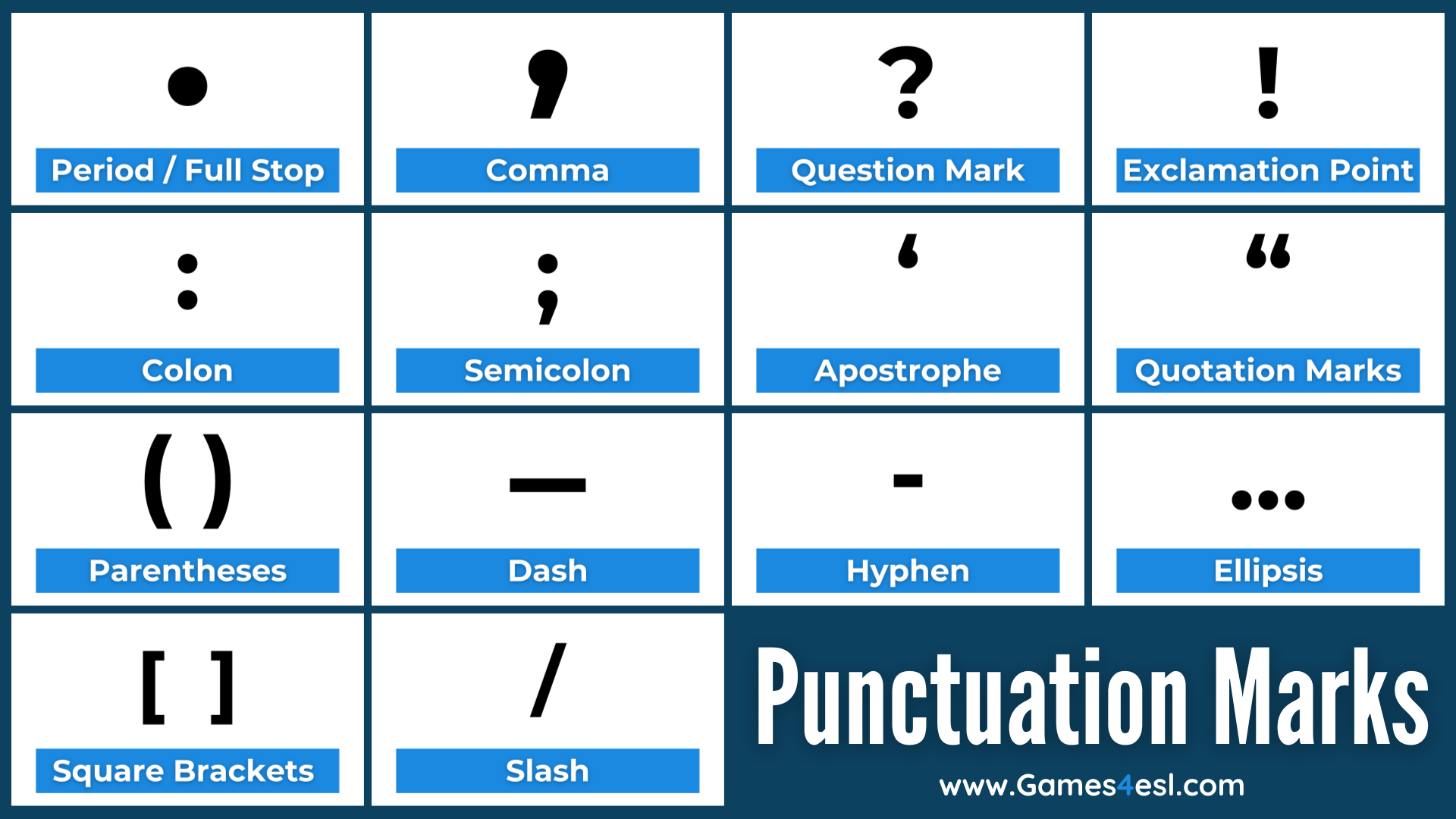


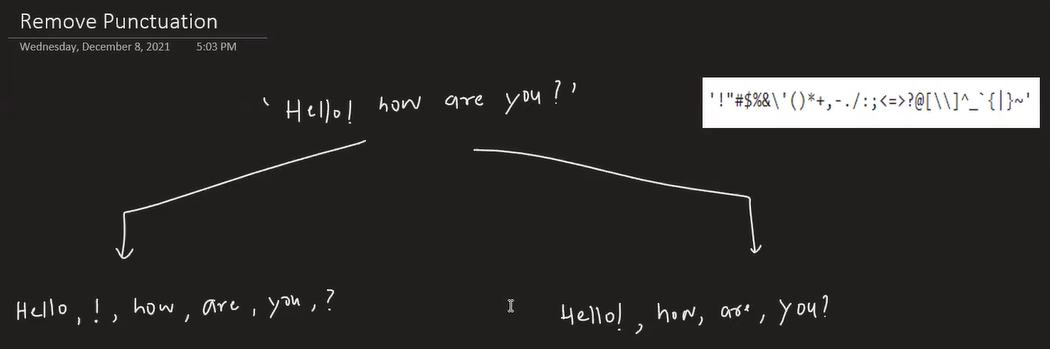


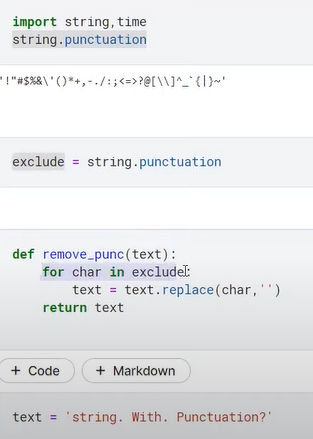
3.





**4**. Punctuation the marks used for dividing writing into sentences and phrases

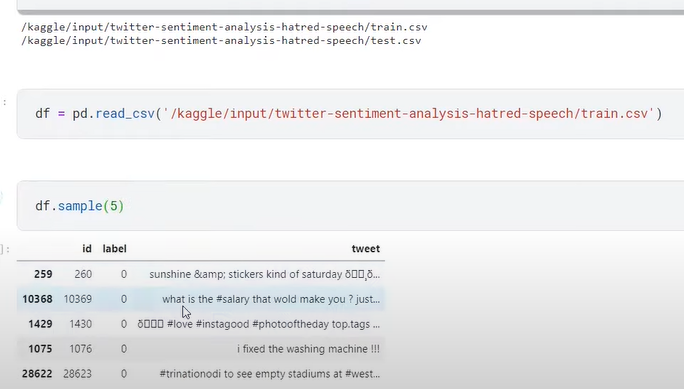


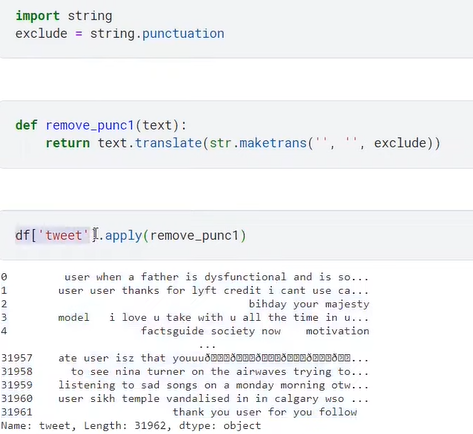


Reduce the time

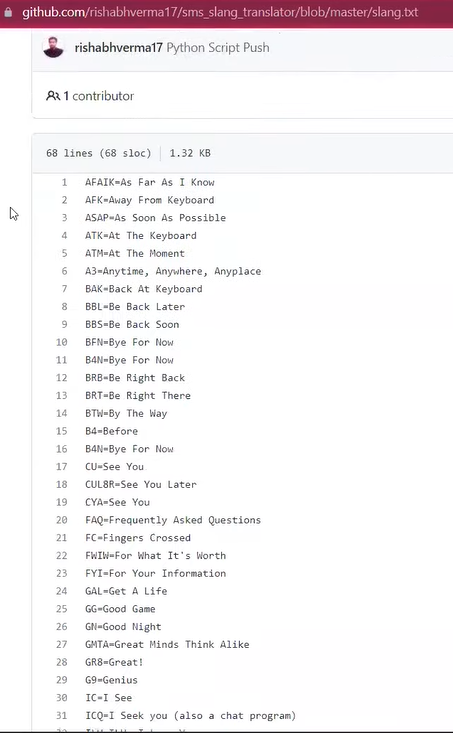


Example

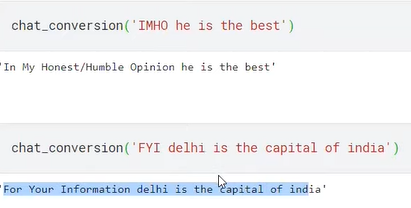




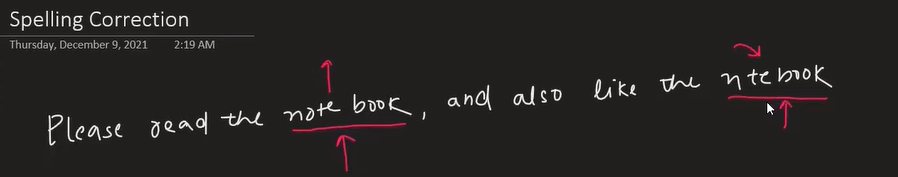
**5.**<https://github.com/rishabhverma17/sms_slang_translator/blob/master/slang.txt>

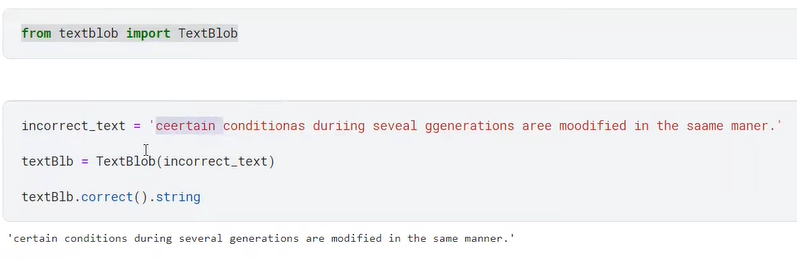
****

****

****

**7.**

****

****

**8. Stop words are extremely common words, like "the," "a," and "is," that are often filtered out in natural language processing (NLP) and search engine algorithms because they typically don't carry significant meaning. Removing them helps improve the efficiency and accuracy of text analysis by focusing on more meaningful keywords, although they can be important in specific cases where they differentiate meaning, such as "the New York Jets" versus "Jets New York".**

Corpus a collection of written or spoken texts

****

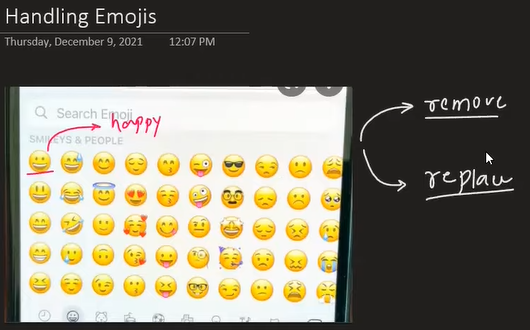
****

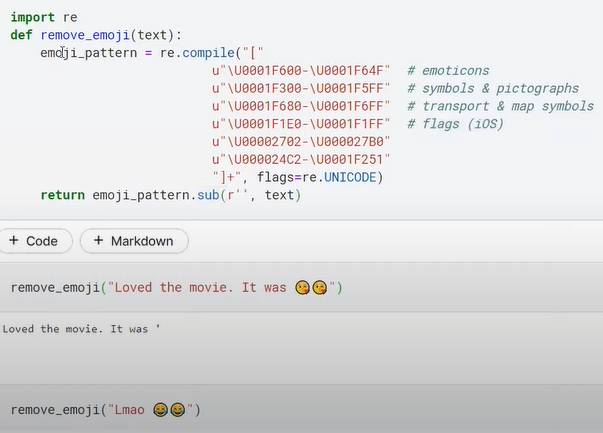
****

****

**9.**

****

****

****

import re

import demoji

demoji.download\_codes() # Download emoji data first

# Get a compiled regex pattern for all emojis

emoji\_pattern = demoji.get\_emoji\_regexp()

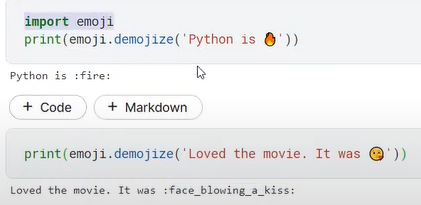
text = "This is a string with some emojis like 😄 and 🚀!"

# Use the pattern to remove emojis

no\_emoji\_text = emoji\_pattern.sub('', text)

print(no\_emoji\_text)

# Output: This is a string with some emojis like and !

****

import emoji

message = emoji.emojize("Hello, :wave:! I'm feeling :grinning\_face\_with\_big\_eyes: today.")

print(message)

# Output: Hello, 👋! I'm feeling 😃 today.

spanish\_message = emoji.emojize("Python es :pulgar\_hacia\_arriba:", language='es')

print(spanish\_message)

# Output: Python es 👍

===============================

import emoji

emoji\_string = "We will eat some 🍔 and 🍟 tonight!"

text\_string = emoji.demojize(emoji\_string)

print(text\_string)

# Output: We will eat some :hamburger: and :fries: tonight!

spanish\_emoji\_string = "Python es 👍"

spanish\_text\_string = emoji.demojize(spanish\_emoji\_string, language='es')

print(spanish\_text\_string)

# Output: Python es :pulgar\_hacia\_arriba:

===================================

**Example in Python:**

import re

text = "Hello World! Visit https://example.com 😊 #NLP2025"

text = text.lower() # Lowercasing

text = re.sub(r"http\S+", "", text) # Remove URLs

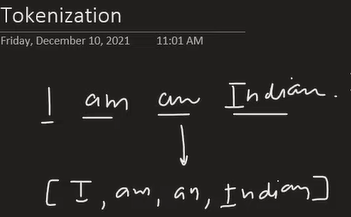
text = re.sub(r"[^\w\s]", "", text) # Remove punctuation

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

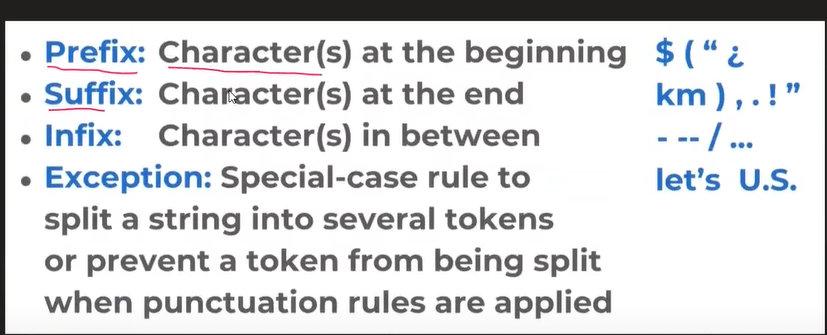
print(text)

### **4.2 Tokenization**

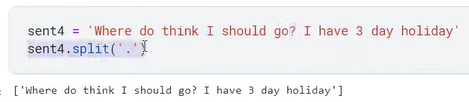
* **What:** Splitting text into smaller units (tokens)
* **Types:**
  1. **Word-level**: "I love NLP" → ["I", "love", "NLP"]
  2. **Sentence-level**: "I love NLP. It is fun." → ["I love NLP.", "It is fun."]
  3. **Subword-level** (BPE, WordPiece): Handles rare words and unseen tokens in ML

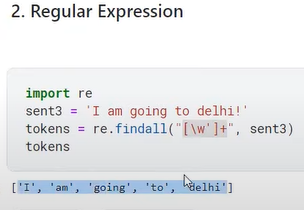


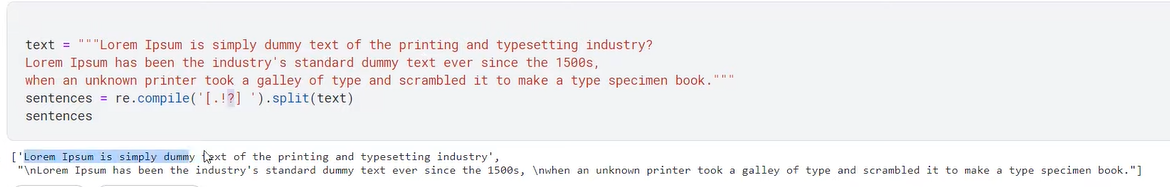
Why











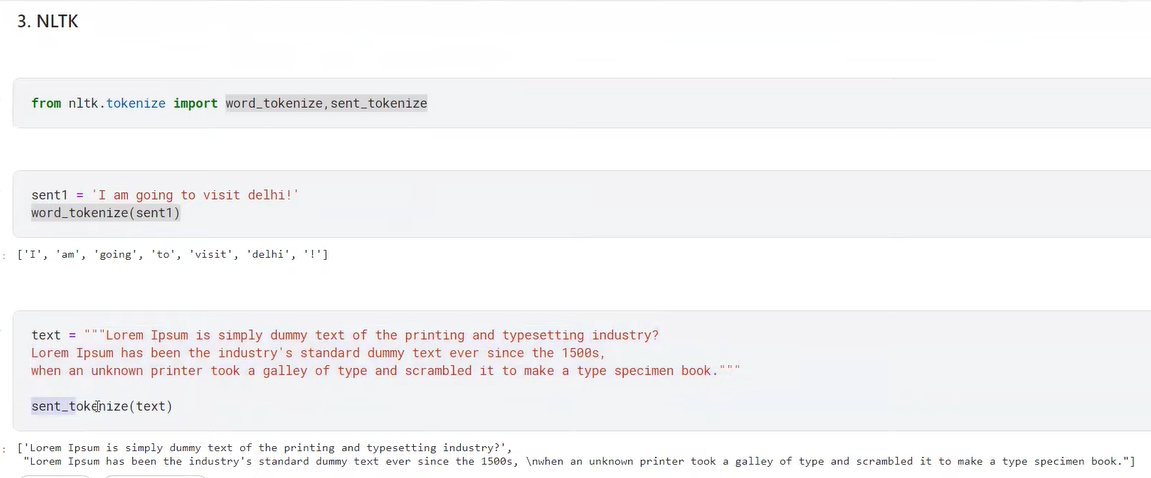
**Example (Word-level with NLTK):**

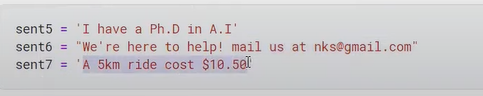
from nltk.tokenize import word\_tokenize

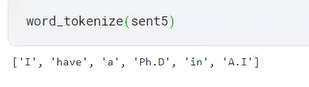
text = "I love NLP!"

tokens = word\_tokenize(text)

print(tokens)

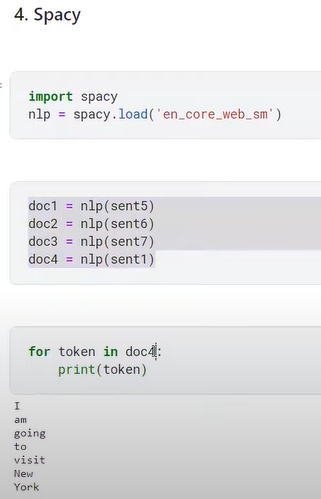








<https://spacy.io/models/en>



### **4.3 Normalization**

* **Lowercasing**: "Hello" → "hello"
* **Remove punctuation, stopwords**
* **Handling numbers, URLs, emojis**

**Example (Removing stopwords):**

from nltk.corpus import stopwords

stop\_words = set(stopwords.words('english'))

filtered = [w for w in tokens if w not in stop\_words]

print(filtered)

### **4.4 Stemming & Lemmatization**

* **Stemming:** Reduce words to root form (may be crude)  
  + **Porter Stemmer:** running → run
  + **Snowball Stemmer:** More aggressive/stable than Porter

**Example:**

from nltk.stem import PorterStemmer, SnowballStemmer

ps = PorterStemmer()

ss = SnowballStemmer("english")

print(ps.stem("running")) # run

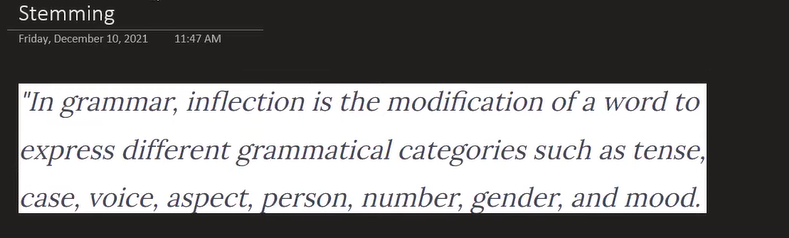
print(ss.stem("running")) # run

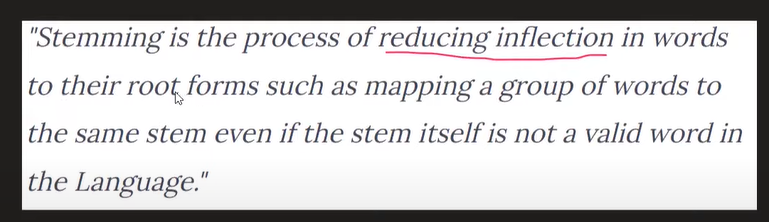
* **Lemmatization:** Reduces words to dictionary form using context

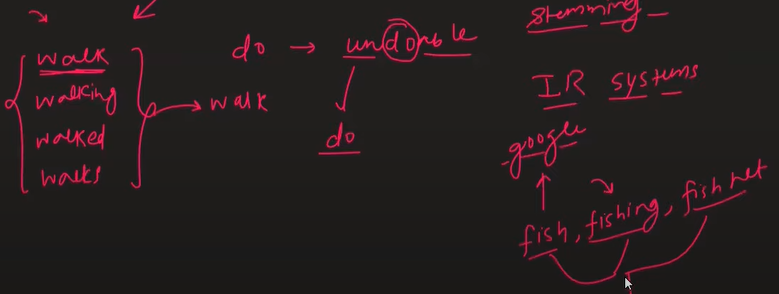
from nltk.stem import WordNetLemmatizer

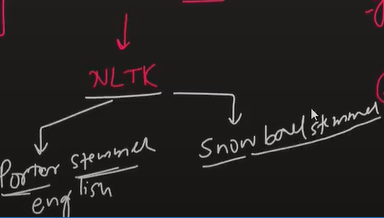
wn = WordNetLemmatizer()

print(wn.lemmatize("running", pos="v")) # run

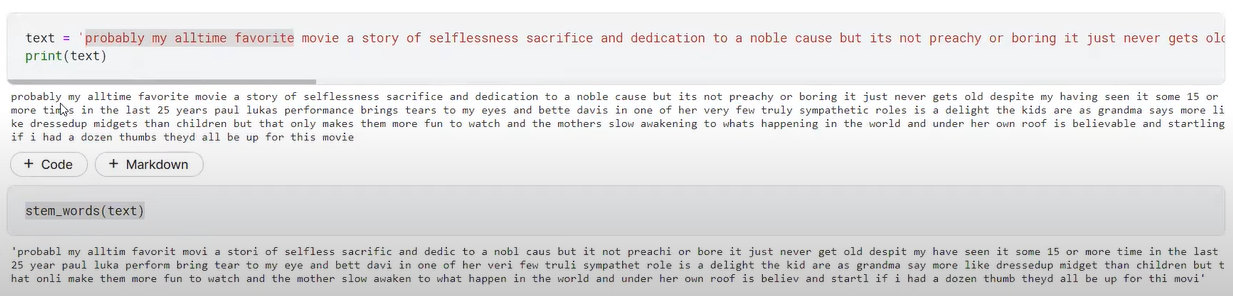


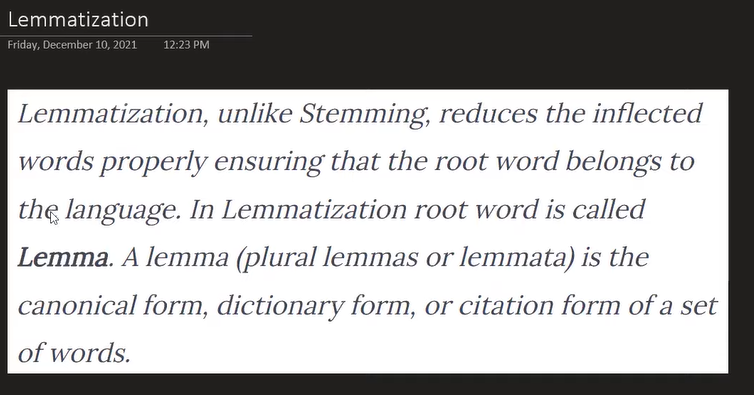














en\_core\_web\_sm'

Stemming and lemmatization are text normalization techniques in Natural Language Processing (NLP) that reduce words to their base or root form. The main difference lies in their approach: stemming uses a heuristic process of chopping off word endings, while lemmatization uses linguistic knowledge and vocabulary to return the valid base word.

| **Aspect** | **Stemming** | **Lemmatization** |
| --- | --- | --- |
| Approach | Applies a series of rule-based heuristics to cut off the ends of words. | Uses dictionaries and morphological analysis to return a valid dictionary word. |
| Output | The result is a "stem," which may not be a real word.  Example: studies → studi  Example: running → runn | The result is a "lemma," which is a valid word found in a language's dictionary.  Example: studies → study  Example: running → run |
| Accuracy | Less accurate, as the simple rules can be too aggressive (over-stemming) or not aggressive enough (under-stemming). | More accurate because it considers the word's part of speech and context. |
| Speed | Faster and less computationally expensive, since it relies on simple, pre-defined rules. | Slower due to the complexity of a dictionary lookup and linguistic analysis. |
| Context | Context-agnostic; it does not consider the meaning of the word in a sentence.  Example: The stemmer cannot distinguish between the verb meeting ("we are meeting") and the noun meeting ("in our last meeting"). | Context-aware; it analyzes the word's context and part of speech to find the correct base form. |
| Use Cases | Ideal for tasks where speed is prioritized over linguistic accuracy, such as large-scale search engine indexing and information retrieval. | Better for tasks that require precise linguistic understanding, such as chatbots, sentiment analysis, and machine translation. |

Examples to illustrate the difference

| **Original Words** | **Stemmed Output** | **Lemmatized Output** |
| --- | --- | --- |
| better | bett (non-word) | good (correct base word) |
| caring | car (incorrect context) | care (correct context) |
| was | was (ignored) | be (correct base word) |
| mice | mic (non-word) | mouse (correct base word) |

Stemming is used for applications like search engines and spam detection where speed and efficiency are prioritized, by chopping off word endings to find a root form (e.g., "running" -> "run"). Lemmatization is better for applications requiring semantic accuracy, such as chatbots or question-answering systems, as it uses a vocabulary to find the actual dictionary form of a word (e.g., "running" -> "run" and "ran" -> "run").

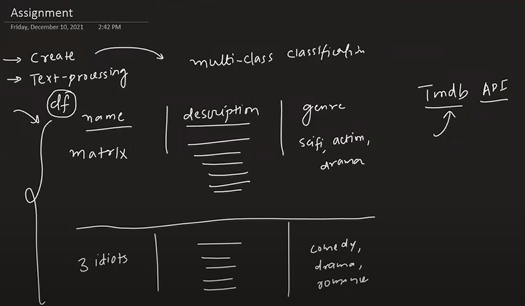
Stemming use cases

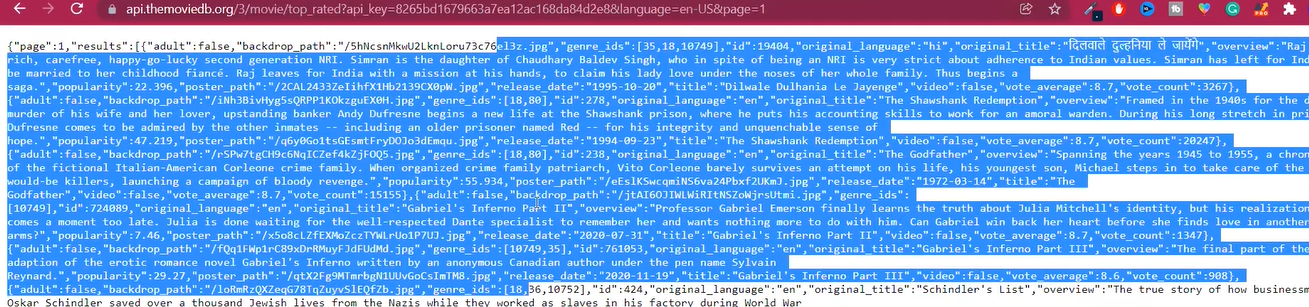
* [**Search engines**](https://www.google.com/search?cs=0&sca_esv=f95eb2d3c20a0c17&sxsrf=AE3TifNIfm5lh9wpyTvwK7n0lE9CuLxq7g%3A1761374544854&q=Search+engines&sa=X&ved=2ahUKEwj4zP6_376QAxXPSGwGHQXtD_YQxccNegQIDhAB&mstk=AUtExfBXR2aLJA5EEh1C9wPcMmRH2-Gn9c2gjby-4sUXhKNWzkO6pSlh3-Y9PmRrZ95azjGlohdekMOp1OVagJ4OBLuWzYThzBwt0H_LZTtW7YO02MPRcofCMKtPk21avWculN_PAVRDUAHGIIILqHHHoCiPRUi6iZrsY5poB0RxqsBz1XWRNpa8Ij6vOtTsdvhSaFd34-7a1tp2SyP1ITiGilaUrIC4PS-neLPWP4Q8qfoY3yqCY5U8Syx5Kj9Onjc1WGKkEzqcsOqwVhXrjZf3Klai&csui=3)**:** Improves search results by matching queries with different word forms, as the exact root is less important than identifying related terms.
* [**Spam detection**](https://www.google.com/search?cs=0&sca_esv=f95eb2d3c20a0c17&sxsrf=AE3TifNIfm5lh9wpyTvwK7n0lE9CuLxq7g%3A1761374544854&q=Spam+detection&sa=X&ved=2ahUKEwj4zP6_376QAxXPSGwGHQXtD_YQxccNegQIEhAB&mstk=AUtExfBXR2aLJA5EEh1C9wPcMmRH2-Gn9c2gjby-4sUXhKNWzkO6pSlh3-Y9PmRrZ95azjGlohdekMOp1OVagJ4OBLuWzYThzBwt0H_LZTtW7YO02MPRcofCMKtPk21avWculN_PAVRDUAHGIIILqHHHoCiPRUi6iZrsY5poB0RxqsBz1XWRNpa8Ij6vOtTsdvhSaFd34-7a1tp2SyP1ITiGilaUrIC4PS-neLPWP4Q8qfoY3yqCY5U8Syx5Kj9Onjc1WGKkEzqcsOqwVhXrjZf3Klai&csui=3)**:** Works well for classifying texts where a word's exact meaning is less critical than identifying the base word for patterns.
* [**Information retrieval**](https://www.google.com/search?cs=0&sca_esv=f95eb2d3c20a0c17&sxsrf=AE3TifNIfm5lh9wpyTvwK7n0lE9CuLxq7g%3A1761374544854&q=Information+retrieval&sa=X&ved=2ahUKEwj4zP6_376QAxXPSGwGHQXtD_YQxccNegQIERAB&mstk=AUtExfBXR2aLJA5EEh1C9wPcMmRH2-Gn9c2gjby-4sUXhKNWzkO6pSlh3-Y9PmRrZ95azjGlohdekMOp1OVagJ4OBLuWzYThzBwt0H_LZTtW7YO02MPRcofCMKtPk21avWculN_PAVRDUAHGIIILqHHHoCiPRUi6iZrsY5poB0RxqsBz1XWRNpa8Ij6vOtTsdvhSaFd34-7a1tp2SyP1ITiGilaUrIC4PS-neLPWP4Q8qfoY3yqCY5U8Syx5Kj9Onjc1WGKkEzqcsOqwVhXrjZf3Klai&csui=3)**:** Groups similar words together for faster and more efficient processing of large text datasets.
* [**Sentiment analysis**](https://www.google.com/search?cs=0&sca_esv=f95eb2d3c20a0c17&sxsrf=AE3TifNIfm5lh9wpyTvwK7n0lE9CuLxq7g%3A1761374544854&q=Sentiment+analysis&sa=X&ved=2ahUKEwj4zP6_376QAxXPSGwGHQXtD_YQxccNegQIEBAB&mstk=AUtExfBXR2aLJA5EEh1C9wPcMmRH2-Gn9c2gjby-4sUXhKNWzkO6pSlh3-Y9PmRrZ95azjGlohdekMOp1OVagJ4OBLuWzYThzBwt0H_LZTtW7YO02MPRcofCMKtPk21avWculN_PAVRDUAHGIIILqHHHoCiPRUi6iZrsY5poB0RxqsBz1XWRNpa8Ij6vOtTsdvhSaFd34-7a1tp2SyP1ITiGilaUrIC4PS-neLPWP4Q8qfoY3yqCY5U8Syx5Kj9Onjc1WGKkEzqcsOqwVhXrjZf3Klai&csui=3)**:** Can be used to quickly analyze large volumes of customer reviews to identify positive or negative sentiments.

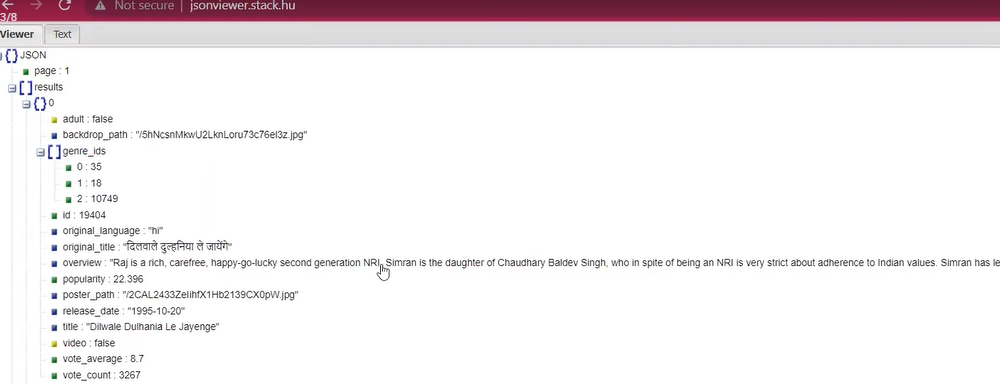
Lemmatization use cases

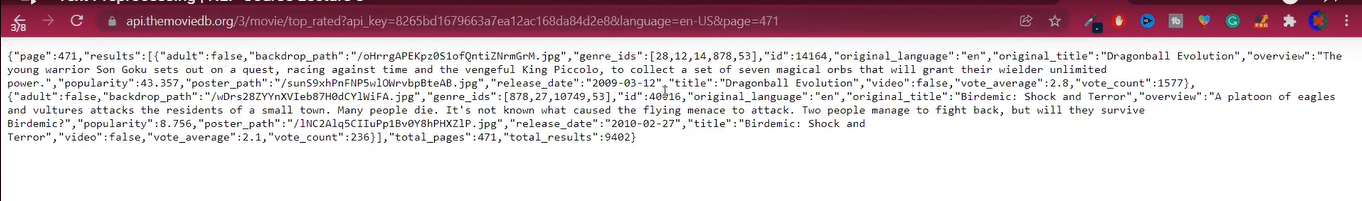
* [**Chatbots and question-answering systems**](https://www.google.com/search?cs=0&sca_esv=f95eb2d3c20a0c17&sxsrf=AE3TifNIfm5lh9wpyTvwK7n0lE9CuLxq7g%3A1761374544854&q=Chatbots+and+question-answering+systems&sa=X&ved=2ahUKEwj4zP6_376QAxXPSGwGHQXtD_YQxccNegQIMxAB&mstk=AUtExfBXR2aLJA5EEh1C9wPcMmRH2-Gn9c2gjby-4sUXhKNWzkO6pSlh3-Y9PmRrZ95azjGlohdekMOp1OVagJ4OBLuWzYThzBwt0H_LZTtW7YO02MPRcofCMKtPk21avWculN_PAVRDUAHGIIILqHHHoCiPRUi6iZrsY5poB0RxqsBz1XWRNpa8Ij6vOtTsdvhSaFd34-7a1tp2SyP1ITiGilaUrIC4PS-neLPWP4Q8qfoY3yqCY5U8Syx5Kj9Onjc1WGKkEzqcsOqwVhXrjZf3Klai&csui=3)**:** Ensures that the generated responses are grammatically correct and contextually relevant because it produces a meaningful, valid dictionary word.
* [**Text summarization**](https://www.google.com/search?cs=0&sca_esv=f95eb2d3c20a0c17&sxsrf=AE3TifNIfm5lh9wpyTvwK7n0lE9CuLxq7g%3A1761374544854&q=Text+summarization&sa=X&ved=2ahUKEwj4zP6_376QAxXPSGwGHQXtD_YQxccNegQINhAB&mstk=AUtExfBXR2aLJA5EEh1C9wPcMmRH2-Gn9c2gjby-4sUXhKNWzkO6pSlh3-Y9PmRrZ95azjGlohdekMOp1OVagJ4OBLuWzYThzBwt0H_LZTtW7YO02MPRcofCMKtPk21avWculN_PAVRDUAHGIIILqHHHoCiPRUi6iZrsY5poB0RxqsBz1XWRNpa8Ij6vOtTsdvhSaFd34-7a1tp2SyP1ITiGilaUrIC4PS-neLPWP4Q8qfoY3yqCY5U8Syx5Kj9Onjc1WGKkEzqcsOqwVhXrjZf3Klai&csui=3)**:** Essential for tasks that require a deeper understanding of the text to preserve meaning and accuracy.
* [**Machine translation**](https://www.google.com/search?cs=0&sca_esv=f95eb2d3c20a0c17&sxsrf=AE3TifNIfm5lh9wpyTvwK7n0lE9CuLxq7g%3A1761374544854&q=Machine+translation&sa=X&ved=2ahUKEwj4zP6_376QAxXPSGwGHQXtD_YQxccNegQINRAB&mstk=AUtExfBXR2aLJA5EEh1C9wPcMmRH2-Gn9c2gjby-4sUXhKNWzkO6pSlh3-Y9PmRrZ95azjGlohdekMOp1OVagJ4OBLuWzYThzBwt0H_LZTtW7YO02MPRcofCMKtPk21avWculN_PAVRDUAHGIIILqHHHoCiPRUi6iZrsY5poB0RxqsBz1XWRNpa8Ij6vOtTsdvhSaFd34-7a1tp2SyP1ITiGilaUrIC4PS-neLPWP4Q8qfoY3yqCY5U8Syx5Kj9Onjc1WGKkEzqcsOqwVhXrjZf3Klai&csui=3)**:** Crucial for translating phrases correctly by understanding the true base form of words.
* [**Knowledge graphs**](https://www.google.com/search?cs=0&sca_esv=f95eb2d3c20a0c17&sxsrf=AE3TifNIfm5lh9wpyTvwK7n0lE9CuLxq7g%3A1761374544854&q=Knowledge+graphs&sa=X&ved=2ahUKEwj4zP6_376QAxXPSGwGHQXtD_YQxccNegQINBAB&mstk=AUtExfBXR2aLJA5EEh1C9wPcMmRH2-Gn9c2gjby-4sUXhKNWzkO6pSlh3-Y9PmRrZ95azjGlohdekMOp1OVagJ4OBLuWzYThzBwt0H_LZTtW7YO02MPRcofCMKtPk21avWculN_PAVRDUAHGIIILqHHHoCiPRUi6iZrsY5poB0RxqsBz1XWRNpa8Ij6vOtTsdvhSaFd34-7a1tp2SyP1ITiGilaUrIC4PS-neLPWP4Q8qfoY3yqCY5U8Syx5Kj9Onjc1WGKkEzqcsOqwVhXrjZf3Klai&csui=3)**:** Helps in extracting entities (like people and places) and connecting them to other entities with accuracy.

Assignmnet









### **4.5 Spelling Correction**

* **Why:** Correct typos improve NLP results  
   **Example using TextBlob:**

from textblob import TextBlob

text = "I lovv NLP"

corrected = TextBlob(text).correct()

print(corrected)

### **4.6 Text Cleaning with Libraries**

| **Library** | **Purpose** | **Example** |
| --- | --- | --- |
| nltk | Tokenization, stopwords, stemming, lemmatization | word\_tokenize(), stopwords |
| spaCy | Tokenization, lemmatization, pipelines | nlp("Text").lemma\_ |
| textblob | Spelling correction, basic NLP | TextBlob(text).correct() |
| clean-text | Quick text cleaning | clean(text) |

## **5️⃣ Practical Example: Build a Text Preprocessing Pipeline**

**Dataset:** customer\_reviews.txt (50 reviews)

**Goal:** Clean text, tokenize, remove stopwords, lemmatize

import re

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

# Initialize

stop\_words = set(stopwords.words('english'))

lemmatizer = WordNetLemmatizer()

# Read dataset

with open("customer\_reviews.txt", "r") as f:

reviews = f.readlines()

# Preprocessing pipeline

cleaned\_reviews = []

for review in reviews:

text = review.lower()

text = re.sub(r"http\S+|www\S+", "", text)

text = re.sub(r"[^\w\s]", "", text)

text = re.sub(r"\d+", "", text)

tokens = word\_tokenize(text)

tokens = [lemmatizer.lemmatize(w) for w in tokens if w not in stop\_words]

cleaned\_reviews.append(tokens)

print(cleaned\_reviews[:5])

## **6️⃣ Business Scenario / Company Use Case**

* **Amazon:** Clean and tokenize millions of product reviews to feed ML models
* **Social Media Monitoring:** Detect trending topics by preprocessing tweets
* **Chatbots:** Preprocess user queries for accurate response generation

## **7️⃣ Practice Session (5–10 Questions)**

1. Lowercase all sentences in reviews.txt
2. Remove punctuation and numbers from a text dataset
3. Tokenize reviews at word and sentence level
4. Remove stopwords from tokens
5. Apply stemming using Porter Stemmer
6. Apply lemmatization using WordNet
7. Correct spelling errors in short text snippets
8. Count the number of unique tokens after preprocessing
9. Build a simple function that runs the entire pipeline on new text
10. Visualize the top 10 frequent words after preprocessing

## **8️⃣ Case Study / Mini Project**

**Project:** Preprocessing Twitter Data for Sentiment Analysis

**Dataset:** 50–100 tweets  
 **Steps:**

1. Load tweets from CSV
2. Clean text (lowercase, remove URLs, emojis, punctuation)
3. Tokenize tweets
4. Remove stopwords
5. Lemmatize tokens
6. Count most frequent words
7. Visualize using matplotlib / seaborn

## **9️⃣ Recap**

* Text preprocessing transforms raw text into clean and structured format
* Techniques: Cleaning, tokenization, normalization, stopwords removal, stemming, lemmatization, spelling correction
* Python libraries: nltk, spaCy, textblob, clean-text
* Essential for NLP tasks like sentiment analysis, chatbot, recommendation systems

## **🔟 Quiz / Interview Questions**

**Theoretical:**

1. What is tokenization?
2. Difference between stemming and lemmatization
3. Why remove stopwords?
4. How does subword tokenization help in NLP?
5. Name three libraries for text preprocessing

**Practical:**

1. Tokenize a sentence into words and sentences
2. Remove punctuation and numbers from text
3. Apply lemmatization on a word list
4. Build a mini pipeline to clean a text file
5. Correct spelling mistakes in a text

## **1️⃣1️⃣ Resource Links**

* [NLTK Documentation](https://www.nltk.org/)
* [spaCy Tutorial](https://spacy.io/usage/spacy-101)
* [TextBlob Docs](https://textblob.readthedocs.io/en/dev/)
* [Clean-Text GitHub](https://github.com/jfilter/clean-text)

✅ **This lesson is ready for students with theory, practicals, datasets, exercises, and case study.**

I can next create a **Streamlit app** for this lesson where students can **upload text and see full preprocessing (cleaning, tokenization, stopwords removal, lemmatization) with visualization**.

Do you want me to make that Streamlit app?

This response builds upon the detailed notes provided previously by extracting explicit and rhetorical questions, answers, rationales, and specific examples used throughout the source material.

## **Q&A and Examples from the Video Lecture**

The following section outlines the key questions, answers, and illustrative examples used to explain text preprocessing concepts.

### **1. Introduction and Necessity**

| **Question/Topic** | **Answer/Rationale** | **Source Examples/Details** |
| --- | --- | --- |
| **What is Text Preprocessing?** | It is the second step in the NLP pipeline, following data acquisition. It involves **data cleaning and processing**. | The main goal is to ensure the data is in the correct form for feature engineering, which helps simplify complexity for the model. |
| **Is it mandatory to use all preprocessing steps?** | No. You must use **common sense and practice** to decide which steps are applicable to your specific textual dataset and problem. | The steps covered include Lowercasing, HTML/URL/Punctuation removal, Chat word treatment, Spelling correction, Stop word removal, Emojis, Tokenization, Stemming, and Lemmatization. |
| **What are the two types of Text Preprocessing?** | **Basic Text Processing** (covered in this video) and **Advanced Text Processing**. | Advanced topics include Parts-of-Speech (POS) tagging, chunking, parsing, and coreference resolution. |

### **2. Basic Text Preprocessing Steps**

| **Question/Topic** | **Answer/Rationale** | **Source Examples/Details** |
| --- | --- | --- |
| **Why is Lowercasing the first step?** | It prevents instances of the same word that differ only in casing (e.g., "Basically" vs. "basically") from being treated as two separate, unique words, thus avoiding unnecessary complexity in the model. | **Implementation Example:** For a simple string: use .lower(). For an entire Pandas Series (column): use df['reviews'].str.lower(). |
| **Why remove HTML Tags?** | HTML tags instruct the browser on how to display data, but they are **not needed** by machine learning models for NLP tasks like sentiment analysis and will only cause confusion. | **Technique:** Use Regular Expressions (regex) to create patterns that match and remove these tags. Data scraped from sites like IMDB often contains HTML tags. |
| **Why remove URLs?** | URLs generally **do not contribute** significantly to the analysis (like classification or sentiment analysis) and can confuse the model. | **Context:** This is common when working with social media data (Twitter, WhatsApp). **Technique:** Use Regular Expressions. Patterns must handle variants like https, http, and www. |
| **Why remove Punctuation? (2 Reasons)** | 1. Punctuation marks (like ? or !) can be tokenized as separate words, unnecessarily increasing the document size. 2. If punctuation remains attached (e.g., "Hello!" vs. "Hello"), the algorithm treats them as two distinct words. | **Usage Note:** Punctuation should be removed 99% of the time in classification/sentiment analysis. |
| **What is the best way to remove Punctuation quickly?** | A self-coded loop method is too slow for big data. The fast, standard technique uses Python's built-in string methods. | **Recommended Technique:** Use text.translate(str.maketrans('', '', excluded\_punctuation)). This method was demonstrated to be **18 times faster** than manual looping. |
| **How should Chat Words (Short forms) be treated?** | If working with social media or messaging data, short forms (e.g., ROFL, GN, IMHO) must be converted to their **normal, complete forms** (e.g., GN to Good Night). | **Implementation:** Requires a dictionary to map the short form (key) to the full word (value). **Example Output:** "IMHO he is the best" is converted to "In My Honest Opinion he is the best". |
| **Why is Spelling Correction necessary?** | Spelling mistakes (e.g., "notebok" for "notebook") result in unnecessary tokens, adding complexity and negatively impacting model performance. | **Libraries:** TextBlob, NLTK, or pyspellchecker can be used. **Example:** Using TextBlob involves creating a TextBlob object and calling .correct(). |
| **What are Stop Words and why remove them?** | They are words (e.g., 'the', 'a', 'of', 'is') that help in sentence formation but **do not contribute to the overall meaning**. They are usually removed for tasks like sentiment analysis. | **Exception:** Stop words are kept for certain tasks, such as **Part-of-Speech tagging**. **Library:** NLTK provides pre-defined lists of stop words for English and other major languages. |
| **How should Emojis be handled?** | Emojis are used to express emotion but are typically **not understood** by standard machine learning algorithms. | **Option 1 (Removal):** Use simple regular expressions based on Unicode patterns (for emotion, symbols, transport, flags). **Option 2 (Replacement):** Replace the emoji with its text meaning (recommended). Use modules like emoji and its demojize function. |

### **3. Advanced Text Processing Steps**

| **Question/Topic** | **Answer/Rationale** | **Source Examples/Details** |
| --- | --- | --- |
| **What is Tokenization and why is it crucial?** | It is the process of **breaking a text document into smaller parts (tokens)**, which can be words, sentences, or phrases. | It is crucial because incorrect tokenization confuses the model during **feature engineering** (like counting unique words), leading to poor results. |
| **What are the challenges in Tokenization?** | Handling prefixes/suffixes, punctuation attached to words, abbreviations (e.g., U.S.), and deciding whether to split or keep multi-word phrases (e.g., "New York," "ten kilometers"). | **Technique Comparison:** 1. **Basic Split:** Fails on difficult scenarios (attached punctuation, multiple delimiters). 2. **Regular Expressions:** Better, but requires extensive pattern creation. 3. **NLTK:** Good internal algorithms, but fails on complex tokens (e.g., splitting email addresses). 4. **spaCy (Recommended):** Most sophisticated and provides the most reliable results, requiring the text to be converted to a document object. |
| **What is Inflection?** | Grammatical modification of a word to express different categories like tense, case, voice, or number (e.g., 'walk' becoming 'walking' or 'walked'). | This creates problems for NLP systems because different forms of the same word are treated as unique. |
| **What is Stemming?** | Reducing inflected words to their root form, or **stem**. The stem is **often NOT a valid word** in the language (e.g., 'probabl' instead of 'probable'; 'stori' instead of 'story'). | **Use Case:** Primarily used in **Information Retrieval Systems** (like search engines), where speed is critical. **Techniques:** Uses algorithms like the **Porter Stemmer** (for English) or Snowball Stemmer (for other languages). |
| **What is Lemmatization?** | Reducing inflected words to their root word, known as the **lemma** (the dictionary or canonical form). The lemma is **always a valid word** in the language. | **Process:** Requires dictionary look-up (e.g., using WordNet, a lexical dictionary) to find the correct base word. It is **slower** than stemming. **Requirement:** Requires specifying the **Part-of-Speech (POS)** for accurate conversion. |
| **When should you use Stemming vs. Lemmatization?** | Use **Stemming** if speed matters and the output is not intended to be shown to the user. Use **Lemmatization** if the output needs to be displayed or shown to the user. | NLTK provides modules for both techniques. |

### **4. Self-Learning Assignment Details**

| **Question/Topic** | **Answer/Description** | **Source Examples/Details** |
| --- | --- | --- |
| **What is the goal of the assignment?** | To create a multi-class classification dataset and apply all learned basic and advanced text preprocessing steps to the data. | The project is designed to enhance interest and learning by working on a newly created dataset. |
| **What data must be collected?** | A movie dataset containing three columns: Movie Name, Description (text data, labeled 'overview' in the API), and Genre (the classification target). | **Data Volume:** Approximately 9,400+ movies across 471 pages. |
| **What are the Data Sources?** | The Movie Database (TMDb) API. | **API 1 (Movies):** Used to retrieve top-rated movies, providing the overview and genre ID. **API 2 (Genres):** Used to retrieve the actual genre names corresponding to the Genre IDs (e.g., converting ID 35 to Comedy). |
| **What preprocessing steps must be applied to the 'Description' column?** | Lowercasing, HTML tag removal, URL removal, Punctuation removal, Spelling correction, Stop word removal, Tokenization, and Stemming/Lemmatization. |  |