You might not need to tokenize ingredients and directions in this way, depending on your project's requirements. Let's evaluate the use case:

**When Tokenization is Needed**

1. **Preprocessing for NLP Models**:
   * If you plan to use a tokenizer that requires sentences or phrases split into individual tokens (e.g., for a custom embedding or retrieval system), this step is useful.
   * Tokenizing directions could help in splitting instructions into manageable steps, which might enhance coherence in outputs.
2. **Custom Processing**:
   * If you intend to process each direction or ingredient separately (e.g., lemmatization, stopword removal, or similarity checks), tokenization provides a clear structure.
3. **Dataset Cleaning**:
   * Tokenization can help detect unusual patterns, like malformed or excessively long steps or phrases.

**When Tokenization is NOT Needed**

1. **Preprocessing for GPT Fine-Tuning**:
   * GPT models already tokenize internally using their own tokenizer (e.g., byte-pair encoding for GPT-2). The raw text (preprocessed for formatting) is sufficient.
   * Tokenizing here might add an unnecessary layer of complexity.
2. **Direct Formatting**:
   * If you’re directly formatting ingredients and directions into text for fine-tuning, sentence-level tokenization isn’t required.

**Suggested Approach**

If your end goal is **fine-tuning GPT-2** or building a **RAG pipeline**, tokenization using PunktSentenceTokenizer is **not required**. Instead:

1. Focus on formatting the data into coherent, model-friendly text.
2. Let the model’s tokenizer handle the text processing.

# Get Sample Words: ingredients\_words = [word for tokens in df\_final['ingredients\_tokenized'] for word in tokens] directions\_words = [word for tokens in df\_final['directions\_tokenized'] for word in tokens] print("Sample Ingredient Tokens:", ingredients\_words[:20]) print("Sample Direction Tokens:", directions\_words[:20])

You **don’t need this step** if your goal is to directly format the data for fine-tuning GPT-2 or building the RAG pipeline. Here's why:

**When This Step is Useful:**

1. **Exploratory Data Analysis (EDA)**:
   * If you’re curious about the common words or patterns in ingredients and directions, this step helps understand the dataset better.
   * Useful for insights like:
     + Most frequently used ingredients or terms.
     + Identifying noise or irrelevant tokens.
2. **Text Preprocessing**:
   * If you're building a custom tokenizer or preprocessing pipeline (e.g., removing stopwords, lemmatization), extracting tokens can be helpful.
   * Analyzing tokens helps refine preprocessing steps.
3. **Feature Engineering**:
   * If you plan to use word-based features (e.g., term frequency) for other models or exploratory tasks, extracting tokens is essential.

**When This Step is NOT Needed:**

1. **Fine-Tuning GPT-2**:
   * The GPT-2 tokenizer will handle tokenization internally, so you don’t need to extract and inspect tokens manually.
   * Formatting the text into a coherent structure (as shown earlier) is sufficient.
2. **RAG Pipeline**:
   * For a retrieval system, token-level analysis isn’t required. You’ll index the full text of recipes (ingredients and directions) instead.
3. **Well-Cleaned Data**:
   * If your data is clean and ready for downstream tasks, token extraction is unnecessary unless you want additional insights.

**Final Recommendation:**

This step is **not required** for your current goals. However, if you’re interested in analyzing tokens for better understanding or validation, you can run it **once** as part of EDA.

from nltk.corpus import stopwords

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

# Remove stopwords from ingredients and directions

df\_final.loc[:, 'ingredients\_tokenized'] = df\_final['ingredients\_tokenized'].apply(

lambda tokens: [word for word in tokens if word.lower() not in stop\_words]

)

df\_final.loc[:, 'directions\_tokenized'] = df\_final['directions\_tokenized'].apply(

lambda tokens: [word for word in tokens if word.lower() not in stop\_words]

)

# Inspect after removing stopwords

print(df\_final[['ingredients\_tokenized', 'directions\_tokenized']].head(10))

Removing stopwords is generally not needed for your Enhanced Recipe Generator project if you are:

Fine-Tuning GPT-2:

GPT-2 works with raw text and learns context and relationships between words, including stopwords, during training. Removing stopwords could disrupt the natural structure of recipes, which might negatively impact the coherence of the generated outputs.

RAG Pipeline:

For retrieval, keeping stopwords is essential because they often contribute to meaningful context, especially in natural language queries and recipes (e.g., "add water" vs. "water" alone).

When Stopword Removal is Useful:

Text Preprocessing for Traditional NLP Models:

If you're building a classic ML model (e.g., Naive Bayes or SVM) or using embeddings where stopwords might introduce noise.

Example: Text classification or clustering tasks where stopwords do not add meaningful information.

Feature Extraction:

If you're focusing on extracting actionable words (e.g., key ingredients or steps), removing stopwords could help simplify the text.

Why It’s NOT Needed for Your Project:

Recipes rely on stopwords for clarity and natural flow:

Ingredients: Stopwords like "of" or "and" are critical in phrases like "2 cups of milk" or "salt and pepper."

Directions: Stopwords help maintain logical sentence structure, e.g., "Add the eggs and mix thoroughly."

Recommendation:

Skip stopword removal and proceed with formatting the raw text as described earlier. Focus on ensuring your text structure (e.g., title, ingredients, directions) is clean and coherent for fine-tuning or RAG.

Let me know if you’d like assistance refining your pipeline! 😊

Lemmatization

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

lemmatizer = WordNetLemmatizer()

**Apply lemmatization to tokenized columns**

df\_final.loc[:, 'ingredients\_tokenized'] = df\_final['ingredients\_tokenized'].apply( lambda tokens: [lemmatizer.lemmatize(word) for word in tokens] ) df\_final.loc[:, 'directions\_tokenized'] = df\_final['directions\_tokenized'].apply( lambda tokens: [lemmatizer.lemmatize(word) for word in tokens] )

**Inspect the results**

print(df\_final[['ingredients\_tokenized', 'directions\_tokenized']].head(10))

[22]:

For your **Enhanced Recipe Generator** project, **lemmatization** is generally **not required**, especially if you are fine-tuning GPT-2 or implementing a RAG pipeline. Here's why:

**Why You Likely Don’t Need Lemmatization**

1. **Fine-Tuning GPT-2**:
   * GPT-2 operates on raw text and is capable of understanding and generating contextually appropriate words. Lemmatizing tokens would strip valuable linguistic nuance and make the input less natural (e.g., "mixes" becoming "mix").
   * Recipes often rely on the full forms of words for clarity and flow. Lemmatizing could disrupt this.
2. **RAG Pipeline**:
   * Retrieval systems like FAISS or ElasticSearch work best with raw text for matching and relevance scoring.
   * Lemmatization may alter the language used in recipes, potentially leading to less accurate retrieval.
3. **Recipe-Specific Language**:
   * Cooking instructions often include domain-specific terms (e.g., "folds," "slices," "grates"). Lemmatization might reduce these to forms that feel unnatural in recipes.

**When Lemmatization Might Be Useful**

1. **Traditional NLP Pipelines**:
   * If you were building a classic NLP model or performing text analysis (e.g., clustering or sentiment analysis), lemmatization could simplify the language for better feature extraction.
2. **Keyword Extraction**:
   * For tasks like extracting unique terms or simplifying language for downstream processing, lemmatization could be useful.

**Recommendation**

Skip lemmatization and retain the natural text of your recipes. Instead, focus on:

1. **Formatting Your Data**: Ensure the ingredients and directions columns are cleaned and properly formatted into coherent text for fine-tuning or retrieval.
2. **Preprocessing for RAG**: If needed, focus on indexing the recipes with minimal alterations to the text.

Would you like help refining your text formatting or jumping to the next step in your project? 😊

from collections import Counter

# Analyze token frequencies

ingredients\_flat = [word for tokens in df\_final['ingredients\_tokenized'] for word in tokens]

directions\_flat = [word for tokens in df\_final['directions\_tokenized'] for word in tokens]

# Get word frequencies

ingredients\_counter = Counter(ingredients\_flat)

directions\_counter = Counter(directions\_flat)

print("Most common ingredients tokens:", ingredients\_counter.most\_common(10))

print("Most common directions tokens:", directions\_counter.most\_common(10))

This step of analyzing token frequencies is **not necessary** for your **Enhanced Recipe Generator** project if your main goal is fine-tuning GPT-2 or implementing a RAG pipeline. Here's why:

**Why You Likely Don’t Need It**

1. **Fine-Tuning GPT-2**:
   * GPT-2 does not require explicit information about token frequencies. It learns patterns in the dataset directly from the raw text during fine-tuning.
   * Analyzing token frequencies does not directly contribute to better model training or performance.
2. **RAG Pipeline**:
   * Retrieval systems focus on the relevance of entire recipes or queries, not individual token frequencies.
   * Token frequency analysis adds little value to the retrieval process.
3. **Already Processed Data**:
   * If your dataset is clean and structured, token frequency analysis won’t influence subsequent steps like model training or evaluation.

**When Token Frequency Analysis is Useful**

1. **Exploratory Data Analysis (EDA)**:
   * If you’re curious about patterns in the dataset, such as the most common ingredients or action words in directions.
   * For instance, you might learn that "water" or "salt" are frequently used ingredients, or "mix" is a common action.
2. **Dataset Cleaning**:
   * If you suspect there are irrelevant or noisy tokens (e.g., typos or extraneous characters), frequency analysis helps identify and filter them.
3. **Custom Features**:
   * If you were building features based on word usage (e.g., bag-of-words models), this step would be essential.

**Final Recommendation**

If you’ve already cleaned and preprocessed your dataset:

* Skip this step for fine-tuning or retrieval tasks, as it doesn’t contribute to those processes.

However, if you’re curious about common patterns in your dataset or want to ensure there’s no noise, feel free to run it **once** as part of exploratory data analysis.

Let me know if you’d like to explore alternative analyses or jump to the next step in your project! 😊