Here’s your **NLP Preprocessing & Feature Extraction Guide** — structured for clarity and practical reference. Each step includes explanations, formulas, advantages, and disadvantages.

## 1. **Import Required Libraries**

import pandas as pd  
import os  
import re  
import nltk  
import numpy as np  
from nltk.corpus import stopwords  
from nltk.stem.porter import PorterStemmer  
from nltk.stem import SnowballStemmer  
from nltk.stem import WordNetLemmatizer  
from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer

**Purpose:** Load CSVs, handle paths, clean text, process words, and extract features.

## 2. **Read CSV File**

file\_path = r"C:\Users\ADMIN\Documents\spam.csv"  
df = pd.read\_csv(file\_path, encoding="latin1").iloc[:, 0:2]

* `` fixes decoding errors.
* `` selects first 2 columns.

✅ Saves memory. ❌ Breaks if CSV format changes.

## 3. **Stopwords Download**

nltk.download('stopwords')

✅ Removes common filler words. ❌ May remove contextually important terms.

## 4. **Regex Cleaning**

review = re.sub('[^a-zA-Z]', ' ', df['v2'][i])

* Removes numbers/symbols. ✅ Keeps only words. ❌ Removes potentially useful symbols.

## 5. **Lowercasing & Tokenization**

review = review.lower().split()

✅ Consistency. ❌ Loses case info.

## 6. **Stopword Removal**

review = [w for w in review if w not in set(stopwords.words('english'))]

**Formula:**

Filtered = Tokens - Stopwords

## 7. **Stemming**

**Porter:**

p\_stem = PorterStemmer()  
review = [p\_stem.stem(w) for w in review]

**Snowball:**

s\_stem = SnowballStemmer('english')

✅ Reduces vocab. ❌ May distort words.

## 8. **Lemmatization**

W\_lim = WordNetLemmatizer()  
review = [W\_lim.lemmatize(w) for w in review]

✅ Produces valid words. ❌ Slower than stemming.

## 9. **Bag-of-Words (BoW)**

cv = CountVectorizer(max\_features=2500, binary=True)  
X = cv.fit\_transform(corpus).toarray()

**Formula:**

BoW\_ij = 1 if word\_j in doc\_i else 0

✅ Simple. ❌ Ignores context.

## 10. **N-Grams**

cv = CountVectorizer(max\_features=500, binary=True, ngram\_range=(2,3))

✅ Captures short phrases. ❌ High dimensionality.

## 11. **TF-IDF**

vectorizer = TfidfVectorizer(max\_features=100)  
X = vectorizer.fit\_transform(corpus).toarray()

**Formula:**

TF = count(term)/total\_terms  
IDF = log(total\_docs/docs\_with\_term)  
TF-IDF = TF \* IDF

✅ Highlights important words. ❌ Ignores meaning.

## 12. **TF-IDF N-Grams**

vectorizer = TfidfVectorizer(max\_features=100, ngram\_range=(2,2))

Captures bigram importance.

## 13. **Vocabulary & Shape**

vectorizer.vocabulary\_  
X.shape

Shows feature mapping and data dimensions.

**Summary Table**

| Step | Purpose | Formula | Pros | Cons |
| --- | --- | --- | --- | --- |
| Lowercasing | Normalize | — | Consistent | Lose case info |
| Tokenization | Split words | — | Easy | Lose spacing |
| Stopwords | Remove filler | Tokens - Stopwords | Cleaner | May lose meaning |
| Stemming | Root by rules | — | Smaller vocab | Breaks words |
| Lemmatization | Root by dict | — | Real words | Slower |
| BoW | Count presence | BoW\_ij formula | Simple | No context |
| N-Grams | Capture phrases | — | More context | High dim |
| TF-IDF | Weight terms | TF\*IDF | Importance-based | Ignores meaning |