**1. Tokenization is not one-size-fits-all.**

I discovered that the token stream is already an ideological decision when I ran the identical line through both NLTK and spaCy. I was given bare words and loose punctuation by NLTK's splitter, while I was given tokens dressed with stop-word flags, POS tags, and lemmas by spaCy. Extra metadata is superfluous in a sentiment-only project, but the tags are crucial for information extraction. The lesson is to consider a tokenizer's side effects in addition to its token list.

**2. Stop-word lists are design opinions.**

I figured stop-words were accepted science, however spaCy counts 326 English stops and NLTK counts 198, with the overlap barely half. Seeing "almost" and "just" on only one list prompted me to consider if eliminating those words would obscure nuance in my sector. For a sarcasm detector, the phrase "just great" loses its sting without the word. Going future, any default stop list will be treated as a draft rather than gospel.

**3. Morphology is where speed meets semantics.**

Porter's stemming felt satisfyingly quick—until "flying" and "flies" both became the non-word fli, while "better" remained better, breaking the link between good and well. SpaCy's lemmatizer fixed that, but it took a little extra time. The lab's side-by-side table reinforced the pragmatic rule: stem for recall-intensive tasks like search, lemma for meaning-sensitive activities like sentiment or summarization. For mixed cases (real-time chatbots), caching lightweight lemmas appears to be the optimal solution.

**Challenges I hit**

* **Emoji loss.** My initial "advanced clean" regex deleted all Unicode emojis. When the sentiment output became flat, I understood that the icons represented more polarity than any adjective in the statement. I modified the regex to save them and associated a few popular ones with sentiment scores.   
  - **Hyphenated compounds.** "Machine-learning-based" was first divided into three orphan tokens, which diluted the concept. I introduced a post-token merging rule for domain-specific compounds, as a reminder that generic libraries require domain patches.

**Real-world connections**

* **Customer-review pipeline.** The typical configuration (basic clean + lemmatize + stop-word removal) achieved a delicious 60% token reduction while retaining adjectives like outstanding and super. That ratio seems appropriate for an e-commerce sentiment engine: small enough for quick inference and rich enough for nuance.
* **Hashtag tracking.** Watching spaCy preserve "#yum" as two tokens ("#", "yum") reminded me of marketing dashboards. Stripping the hash but maintaining the word allows me to use the same classifier for tweets and normal sentences without having to retrain a hashtag-specific vocabulary.

**Trade-offs I will remember.**

* *Information vs. noise: Every character we drop speeds models up but risks deleting a weak signal (negations, punctuation, sarcasm).*
* *Speed vs. accuracy: Stemming is faster by around 2times in my timing tests but produced 3× more unique pseudo-words.*
* *Generality vs. domain fit: A legal corpus wants citations intact; a Twitter corpus wants emojis intact. One pipeline rarely rules all.*

**Next experiments**

1. **Negation-aware stop-word list.** I plan to auto-whitelist *not, no, never* before any bulk removal.
2. **Emoji sentiment lexicon.** Map twenty high-frequency emojis to ± sentiment scores and measure the accuracy lift.
3. **Dynamic pipeline knob.** Expose cleaning options as command-line flags so I can quickly A/B test minimal vs. aggressive settings on new datasets.

In conclusion, Lab 02 transformed preparation from a background task to a toolset of deliberate decisions. My first inquiry when starting a new NLP project will no longer be "What library do I import?" but "Which levers matter most for this text and this task?" That perspective adjustment is undoubtedly the most significant thing I'm taking away.