# Lesson 37: Text Preprocessing

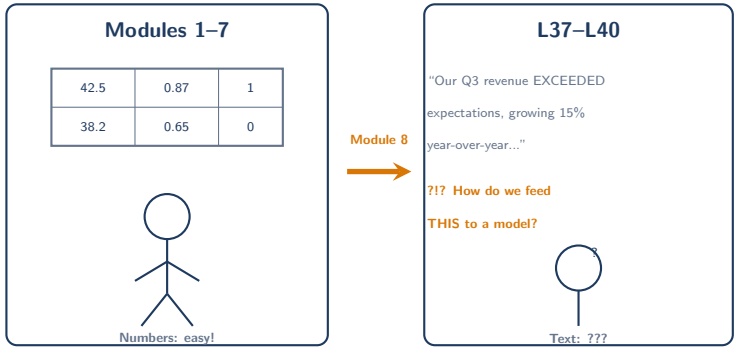
## Data Science with Python – BSc Course

Data Science Program

BSc Course

45 Minutes

# Module 8: Numbers → TEXT – The New Frontier




**Until now,** every dataset was numeric. Text changes everything.

---

**4 lessons: L37 Preprocessing → L38 BoW/TF-IDF → L39 Embeddings → L40 Sentiment**

# Learning Objectives

**The Problem:** ML models need numbers, not text. How do we convert messy, unstructured text into clean tokens ready for numerical encoding?

**After this lesson, you will be able to:**

1. **Explain** why text preprocessing is necessary (Remember)
2. **Apply** tokenization, stopword removal, stemming, lemmatization (Apply)
3. **Compare** stemming vs lemmatization trade-offs (Analyze)
4. **Construct** a complete cleaning pipeline for financial text (Create)

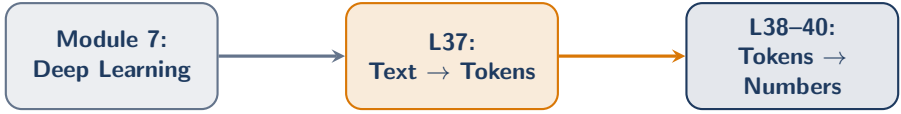
**Key Insight:** Preprocessing quality determines model quality – garbage in, garbage out.

---

**Finance Application: Processing earnings calls, SEC filings, and news for sentiment analysis**

# Bridge: From Neural Networks to Natural Language

**Where we left off:** Neural networks can learn any function from *numbers*.




- 80% of business data is **unstructured text** (emails, reports, filings)
- Before we can use any model, text must become **clean tokens**
- Today: the critical first step – **text preprocessing**

---

**Text preprocessing is the foundation every NLP technique builds on**

# How Does a Computer "Read"?

**Humans vs Machines:**

- Humans read "Apple's revenue grew" and understand meaning instantly
- A computer sees: `0x41 0x70 0x70 0x6C 0x65 ...` – raw bytes
- We must bridge this gap in stages

**Analogy – Cooking Ingredients:**

- Raw text = unwashed vegetables from the market
- Preprocessing = washing, peeling, chopping into uniform pieces
- Only *then* can you cook (analyze) them properly

**The Pipeline Preview:**

Raw text → Lowercase → Tokenize → Remove stopwords → Stem/Lemmatize → Clean tokens

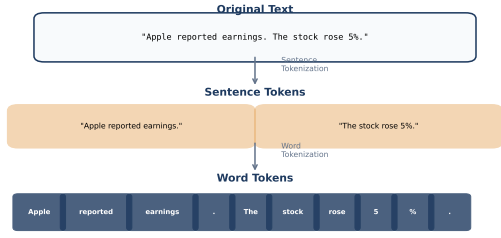
---

**Every NLP project starts here – even ChatGPT preprocesses input text**

# Step 1: Tokenization

## Break text into atomic units

- A token is usually a word, but can be a subword or sentence
- "The stock rose 5%" → [`The`, `stock`, `rose`, `5%`]



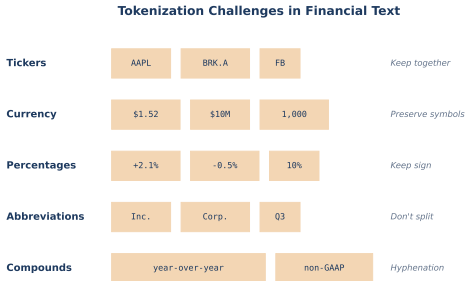

*Tokenization: Split text into analyzable units*

**Tokenization is always the FIRST step in any NLP pipeline**

# Tokenization Challenges in Finance

## Not every split is straightforward

- “New York” – one concept or two tokens?
- “$42.50” – dollar sign, number, or one token?
- “year-over-year” – one hyphenated word or three?

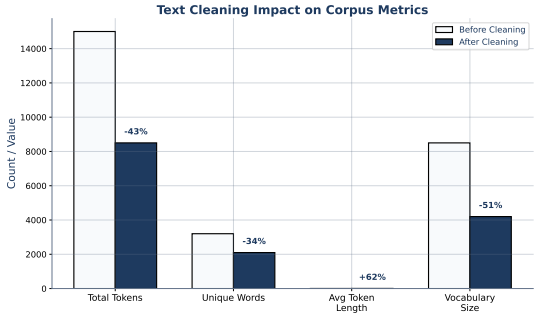



**Domain-specific tokenizers handle financial terms better than generic ones**

# Step 2: Text Cleaning

## Standardize before you analyze

- Lowercase: “Apple” and “apple” become the same token
- Remove HTML tags, URLs, special characters




**Cleaning reduces noise while preserving meaningful content**

# Step 3: Remove Stopwords

## Filter out words that carry no meaning

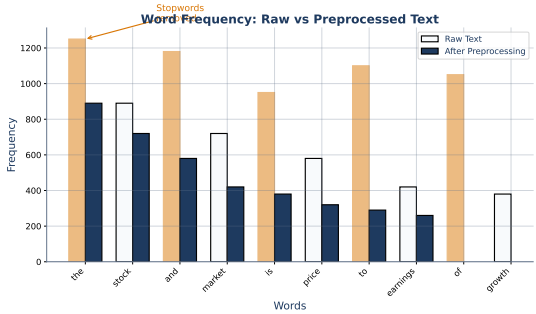
- Stopwords: “the”, “is”, “and”, “of” – always present, never informative
- Removing them shrinks vocabulary and reduces noise
- **Careful:** keep negation words! “not profitable” $\neq$ “profitable”


**Be careful with negation – “not profitable” becomes “profitable” without “not”!**

# Before vs After: Word Frequency

## Stopwords dominate raw text

- Before cleaning: “the”, “is”, “and” are top words
- After cleaning: meaningful content words emerge

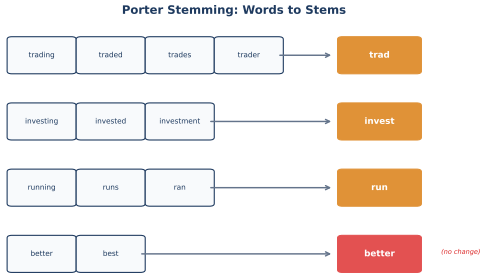



**Financial vocabulary emerges only after removing stopwords**

# Step 4a: Stemming – Chop the Endings

## Reduce words to their root by removing suffixes

- “running” → “run”, “stocks” → “stock”, “trading” → “trad”
- Fast but crude – sometimes produces non-words

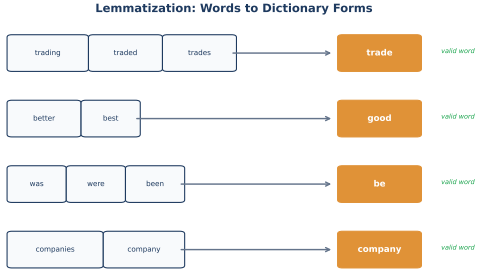



**Code:** `from nltk.stem import PorterStemmer; ps.stem('trading')`

# Step 4b: Lemmatization – Dictionary Lookup

## Reduce to proper dictionary form (lemma)

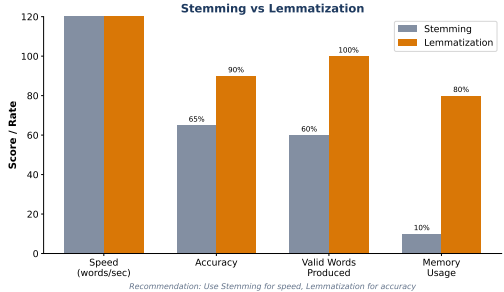
- “running” → “run”, “better” → “good”, “studies” → “study”
- Uses grammar rules – always produces real words




**Code:** `from nltk.stem import WordNetLemmatizer; wnl.lemmatize('stocks')`

# Checkpoint: Stemming vs Lemmatization

**Quick Question:** "studies" → A: "studi" or B: "study" – which is stemming?




- **Stemming:** Speed matters, large corpus, search engines
- **Lemmatization:** Accuracy matters, sentiment analysis

---

**Answer: A = stemming (crude chop), B = lemmatization (dictionary lookup)**

# Beyond Single Words: N-grams

**Single words lose context**

- "not good" – if split into ["not", "good"], the meaning is lost
- Bigrams keep pairs together: "not_good" captures negation
- Trigrams: "stock market crash" is one concept

**Types:**

- **Unigram** (n=1): "stock", "market", "crash"
- **Bigram** (n=2): "stock market", "market crash"
- **Trigram** (n=3): "stock market crash"

**Trade-off:** More n-grams = richer features but much larger vocabulary

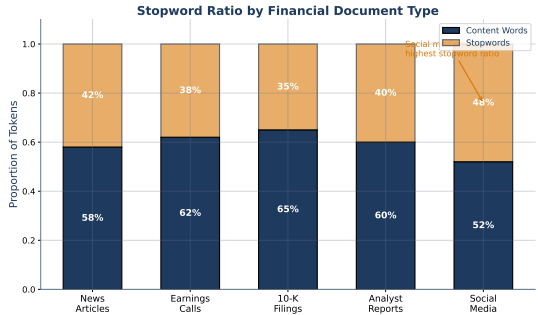
---

**Bigrams capture phrases like "interest rate" and negations like "not profitable"**

# Stopword Ratios Vary by Document

## Different documents need different strategies

- News articles: ∼40–50% stopwords (informal language)
- SEC filings: ∼30–40% stopwords (formal, legal)




**Document type affects preprocessing strategy – one size does NOT fit all**

# Regular Expressions for Text Cleaning

## Regex = pattern matching on steroids

- Remove URLs: `re.sub(r'http\S+', '', text)`
- Remove punctuation: `re.sub(r'[^\w\s]', '', text)`

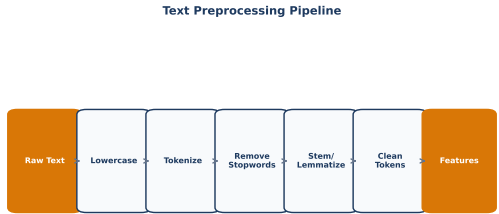
**Essential Regex Patterns for NLP**

| Pattern | Matches | Examples |
|---|---|---|
| `\d+` | One or more digits | 123, 45 |
| `\w+` | Word characters | hello, ABC |
| `\s+` | Whitespace | spaces, tabs |
| `[A-Z]{2,5}` | Ticker symbols | AAPL, TSLA |
| `$[\d,]+\.?\d*` | Currency amounts | 1.52,10,800 |
| `[+-]?\d+\.?\d*%` | Percentages | +2.1%, -0.5% |

**Regex is essential for cleaning real-world financial text**

# The Complete Preprocessing Pipeline

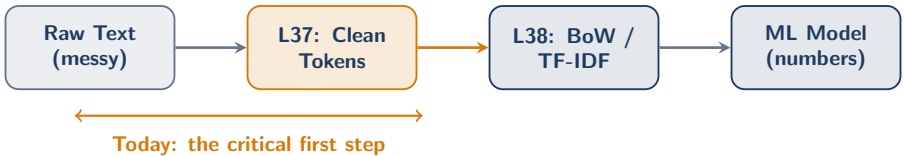
**7 Steps:** Lowercase → Remove HTML/URLs → Remove punctuation → Tokenize → Remove stopwords → Lemmatize → Rejoin




**Order matters: lowercase before tokenize, tokenize before stopword removal**

# The Bridge: Clean Tokens → Numbers

**Where preprocessing fits in the NLP pipeline:**



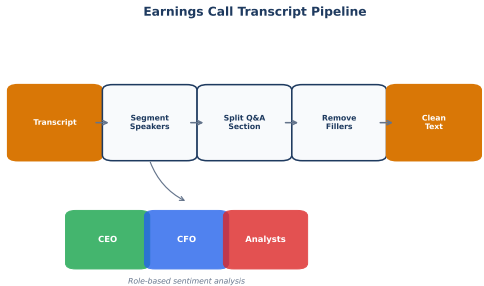

- Clean tokens are the **input** to every text encoding method
- Bad preprocessing → bad features → bad model, no matter how fancy
- Next lesson (L38): convert these tokens to actual numbers

**Preprocessing is invisible in the final model – but it makes or breaks performance**

# Finance: Preprocessing Earnings Calls

## Real-World Example: Quarterly Earnings Call

- Raw: "Our Q3 revenue EXCEEDED expectations, growing 15% YoY..."
- Clean: [q3, `revenue`, `exceed`, `expectation`, `grow`, 15, `yoy`]

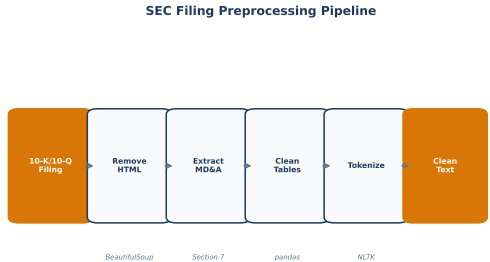



**Domain knowledge determines preprocessing choices – no one-size-fits-all**

# Finance: Preprocessing SEC Filings

## SEC 10-K filings have unique challenges

- Legal language, boilerplate paragraphs, XBRL tags
- Must preserve financial terms: “EBITDA”, “10-K”, “MD&A”
- Custom domain stopwords: “company”, “fiscal”, “quarter”




**SEC filings require domain-specific preprocessing beyond standard NLP**

# Preview: From Tokens to sklearn

**After preprocessing, sklearn takes over:**

**Step 1 – Clean (today):**

- `tokens = preprocess(raw_text)` ← **L37**

**Step 2 – Encode (next lesson):**

- `from sklearn.feature_extraction.text import TfidfVectorizer`
- `vec = TfidfVectorizer(max_features=5000)`
- `X = vec.fit_transform(clean_texts)` ← **L38**

**Step 3 – Model (L40):**

- `model = LogisticRegression().fit(X_train, y_train)`

**Key Point:** `TfidfVectorizer` *can* do some preprocessing, but custom cleaning always wins.

---

**Clean input + TfidfVectorizer + LogisticRegression = strong NLP baseline**

# Hands-On Exercise (25 min)

**Task: Clean Financial News Headlines**

1. Load 50 financial news headlines (provided CSV or web scrape)
2. Build a `preprocess(text)` function:
   - Lowercase → remove punctuation → tokenize → remove stopwords → lemmatize
3. Compare word frequency distributions before and after cleaning
4. Visualize top 20 words in both cases (bar chart)
5. Try both stemming and lemmatization – compare results

**Deliverable:** Before/after word frequency bar charts + cleaned text samples.

---

**Extension: Add custom finance stopwords ("company", "inc", "ltd") and compare**

# Key Takeaways

**Problem Solved:** We can now convert raw, messy text into clean, standardized tokens ready for numerical encoding.

**What You Learned:**

1. **Tokenize** text into meaningful units (words, subwords, sentences)
2. **Clean** text: lowercase, remove HTML/URLs, strip punctuation
3. **Remove stopwords** to focus on content (but keep negation!)
4. **Stemming** is fast but crude; **lemmatization** is accurate but slower
5. **Adapt** preprocessing to document type and task

**One Rule:** Preprocessing quality determines model quality.

---

**Memory: Tokenize → Remove stopwords → Lemmatize. Order matters!**

# Next: From Words to Numbers (L38)

**We have clean tokens. Now what?**


**L38 Preview – Two Key Questions:**

1. **Bag of Words:** What if we just *count* each word?
2. **TF-IDF:** What if common words count *less*?

**Spoiler:** TF-IDF + Logistic Regression is a surprisingly strong baseline.

---

**Next lesson: the simplest way to turn words into numbers that actually works**