Raw Data → Cleaning → Normalization

Python libraries →

NLTK (slow)	Spacy (fast)
1) Education, Research, Experimentation	1) Industry & Production.
2) Modular & provides a lot of tools & resources for nlp tasks.	2) Pipeline based & provides fast & integrated pipelines for nlp tasks.
- Modular but provides basic building blocks for nlp tasks.	- Provides limited but highly optimized nlp tools & resources.
3) Good for analytics & Exploration	3) Stable & production friendly.
4) Large community.	4) Growing community.
5) Requires more manual tasks.	5) Less manual work.

Text Cleaning →

1) Remove unwanted characters →

a) str.maketrans(chars to replace, chars to replace with, chars to delete)
- Creates a mapping for characters & their replacement.

b) text.translate(dict/mapping from make_trans)
- Does the actual replacement in text.
- Can use a custom dictionary instead of the one provided by maketrans.

ex: text.translate(str.maketrans('', '', str.punctuations))
- removes punctuations from text.

2) Lowercase → text.lower()

3) Remove urls, whitespace & digits →

```
import re                                  # Pattern for urls.
pattern = re.compile(r'http s?://\S+|www\.\S+')
pattern.sub(text to replace with, original text)
```

— Replace all the matched text with replacement text.

```
// white spaces →
re.sub(r'\S+', ' ')   # Replace extra whitespaces with a single white space.
re.sub(r'\d+', "")
```

Tokenization → Breaking text into smaller sections, called tokens.

```
from nltk.tokenize import sent_tokenize, word_tokenize.
import spacy
sentences = sent_tokenize(raw_text)
for i, sentence in enumerate(sentences):
    print(f'sentence {i+1}: {sentence}')

nlp = spacy.load('en_core_web_sm')   # Loads the spacy pipeline.
spacy_ret = nlp(raw_text)
```

— spacy passes the entire raw_text through its pipeline.

Stop words → Words that occur frequently.
— the, in, a, for...
— Don't have much value but reduces the worthiness of valuable words.

```
from nltk.corpus import stopwords
words = word_tokenize(text)
words_a = [word for word in words if word not in
                set(stopwords.
                words('english'))]
# using spacy #
words_spacy = [word.text for word in spacy_doc
                if not word.is_stop]
```

Stemming & Lemmatization # (both deal with bringing the word to its root form)

- Aim is to bring words to root form.
- By key stripping characters at the end of the words.
- Fast but can be inaccurate.

- Uses vocabulary & morphological analysis to bring words to their root form.
- Slow but accurate.

```
nltk # PorterStemmer(), WordNetLemmatizer.
stemmer = PorterStemmer()
stem_words = [stemmer.stem(word) for word in
                words_a]
// using spacy #
stem_words = [word.lemma_ for word in spacy_doc if not
                word.is_stop]
```

Sentiment Analysis →

gensim → Python library for gen ai.

polarity → sentiment [-ve < 0 < +ve] (-1 to 1)

Subjectivity → opinion are in part. (0 to 1)

1) TextBlob → Library for Sentiment & Subjectivity analysis.

```
from textblob import TextBlob
blob = TextBlob(text)
blob.sentiment.polarity, blob.sentiment.subjectivity.
```

→ polarity → Information about sentiment of the text. [-1,1]

→ Subjectivity → If the text is Subjective or objective.

opinionated part

(objective) 0 ——— 1 (Subjective)

2) NRC Lexicon → Maps words to 8 basic words that convey the different emotions of the text.

14K words - manually made

- Ultimately maps to two emotions.
- Can analyse multiple emotions beyond good & bad.

text → 8 emotions.
text → 2 sentiment.

```
from nrclex import NRCLex
emotion = NRCLex(text)
```

emotion.words
- raw_emotion_scores → freq of occurance of each score!
- top_emotions
- affect_frequencies → freq. of each score / total score count.

3) Afinn → Simplied sentiment analysis procedure.
2500 words. - fast & light-weight.

- provides a score b/w -5 to 5.
- aggregates the score for each word for a sentence.

```
from afinn import Afinn

af = Afinn()

score = af.score(text)
```

af.split(text) → Split text into words.
.find_all(text) → return all the emotional text.

.score_with_pattern(text) → return score of each emotion.
.score_with_wordlist(text) → total score of emotions.