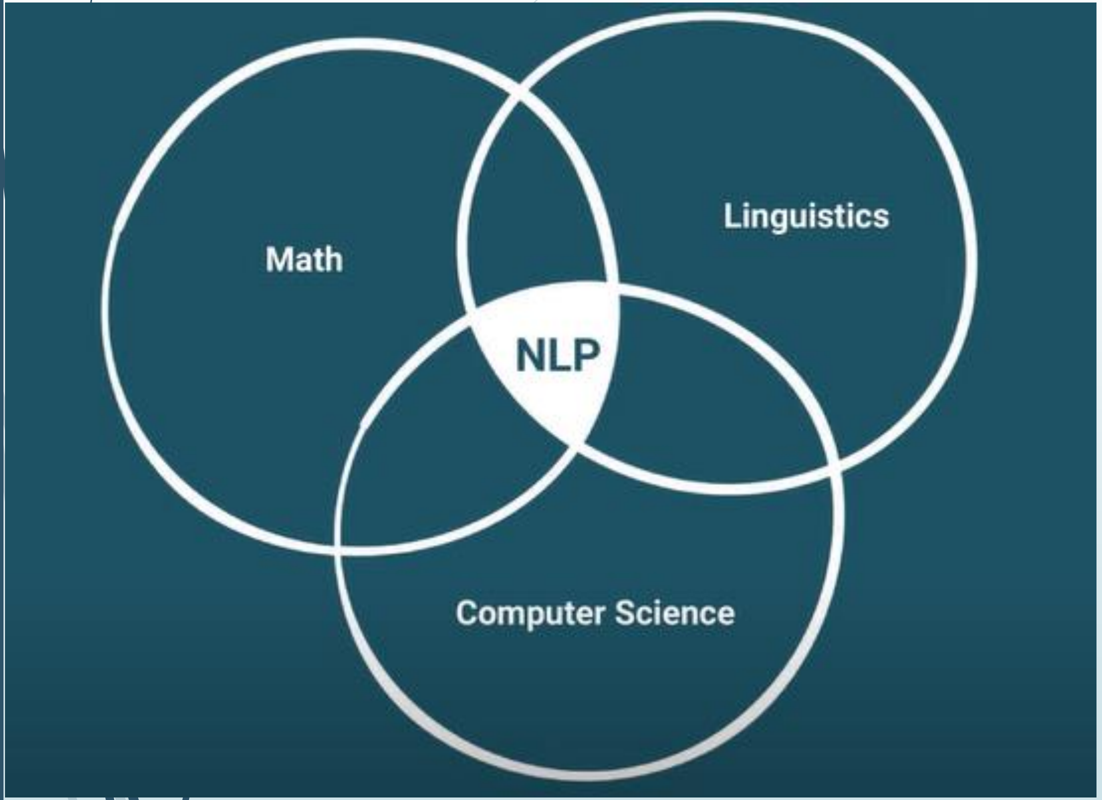
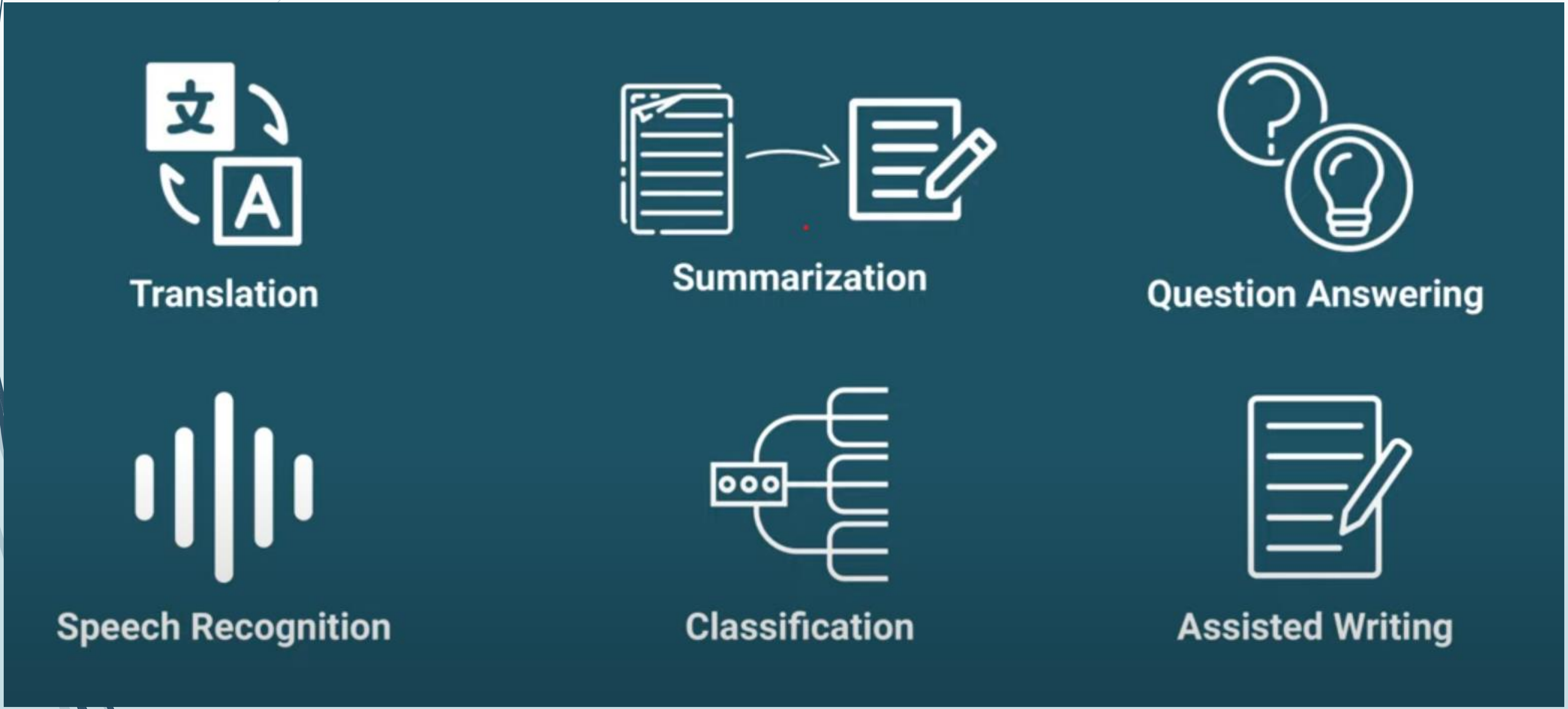
Introduction to NLP



Natural Language Processing or NLP is a multidisciplinary field combining math linguistics and computer science the goal is to get computer to do useful things with natural language data. And by Natural language we mean the language we read , write and speak with.

Common NLP Tasks and Applications



Why NLP is hard ?

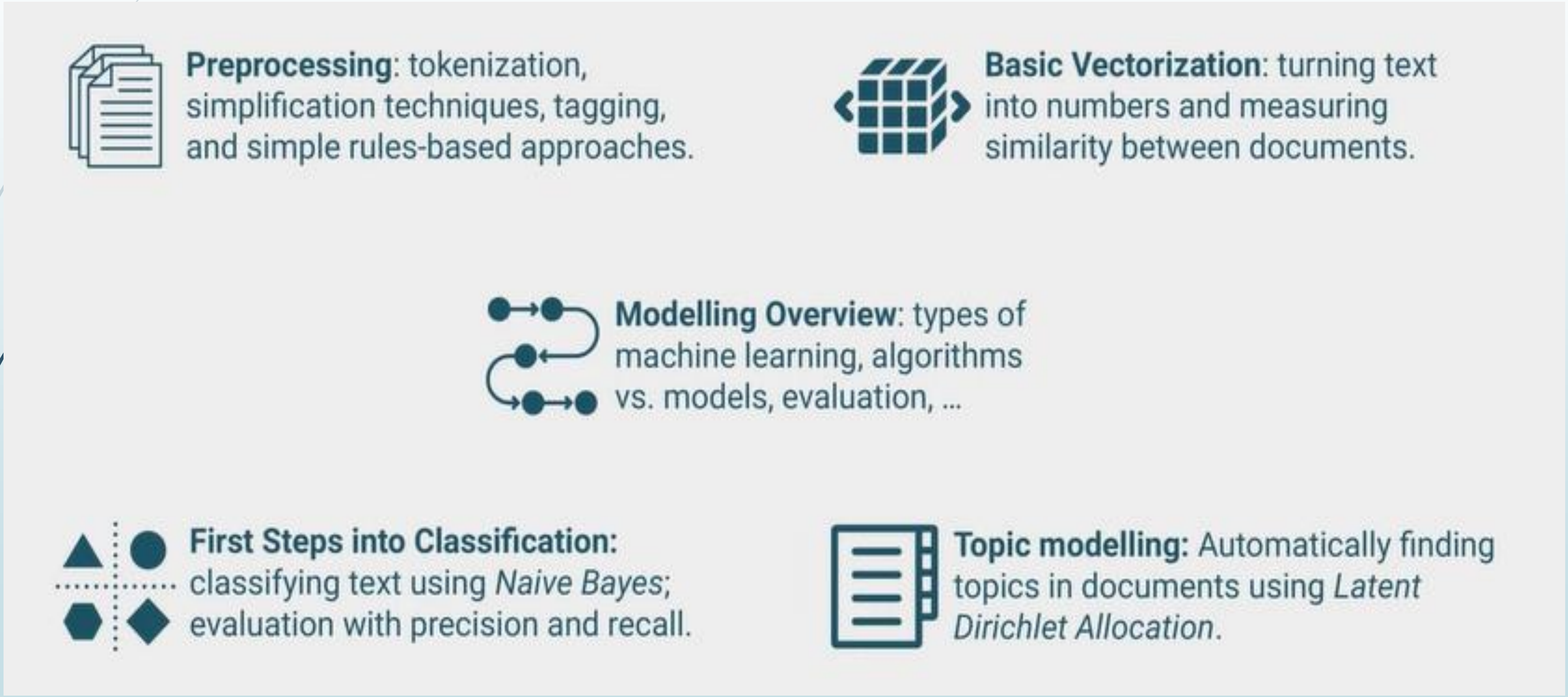
NLP is dependent on natural language and its is evolving with us regularly. So it is highly Contextual and full of ambiguity, metaphors and expressions. For example…

He is at bank

Some reasons of why NLP is hard :

1.Context
2.Inflections
3.Content
4. Misspelled or misused words
5. **Physical gestures**

Fundamentals of NLP



Preprocessing: tokenization, simplification techniques, tagging, and simple rules-based approaches.

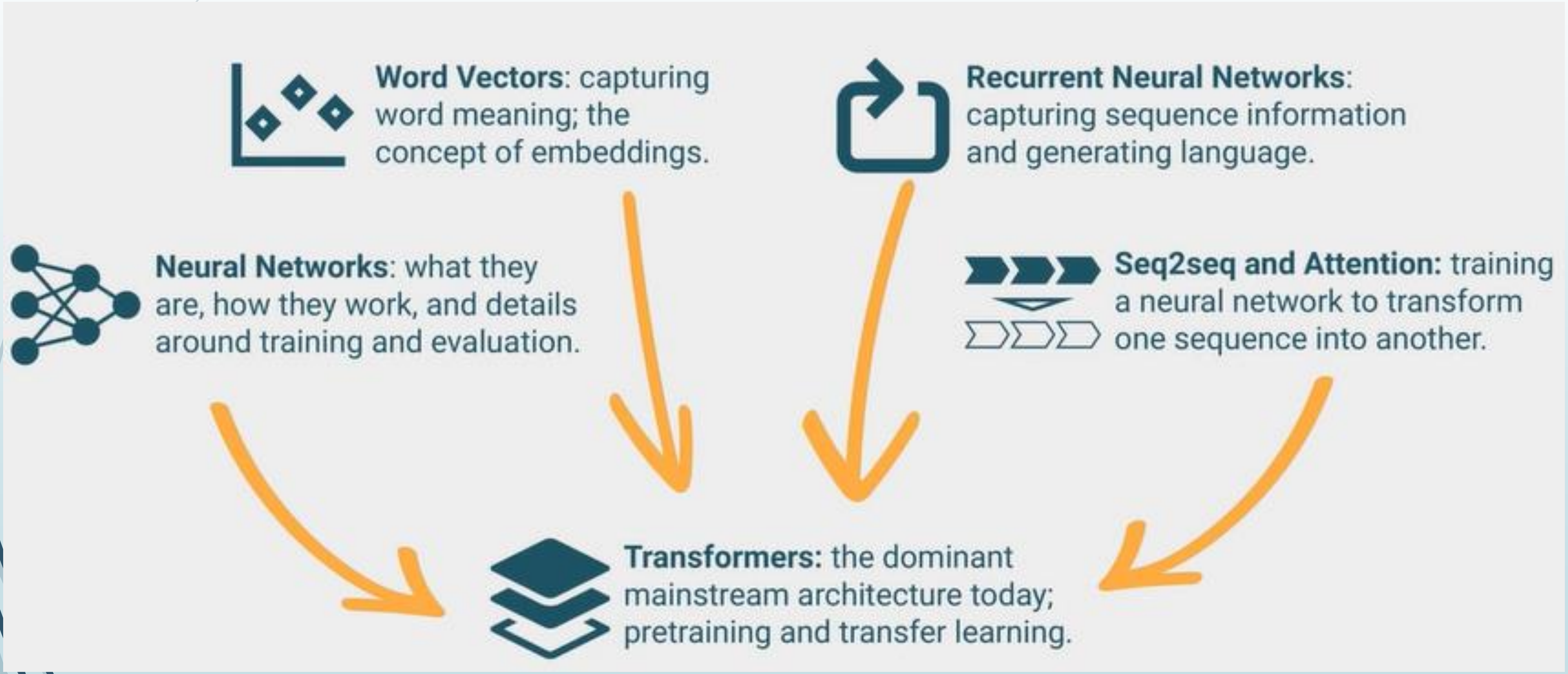
Basic Vectorization: turning text into numbers and measuring similarity between documents.

Modelling Overview: types of machine learning, algorithms vs. models, evaluation, …

First Steps into Classification: classifying text using *Naive Bayes*; evaluation with precision and recall.

Topic modelling: Automatically finding topics in documents using *Latent Dirichlet Allocation*.

Deep Learning for NLP



Word Vectors: capturing word meaning; the concept of embeddings.

Recurrent Neural Networks: capturing sequence information and generating language.

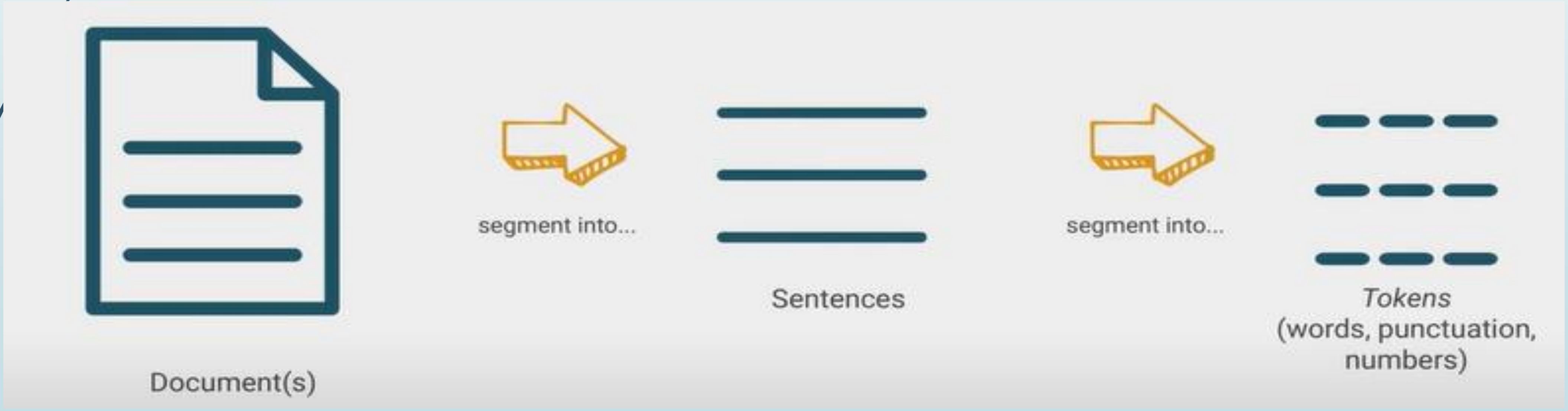
Neural Networks: what they are, how they work, and details around training and evaluation.

Seq2seq and Attention: training a neural network to transform one sequence into another.

Transformers: the dominant mainstream architecture today; pretraining and transfer learning.

Preprocessing : 1.Tokenization

We start any NLP project with a data set its called corpus(unstructured). It can be …
1.Collection of posts we scraped from social media
2.Full books
3. Audio of pod casts etc.
Our first task is to preprocess this corpus to give it some structure for further analysis. Typically for NLP the first preprocessing step is tokenization
The process of segmenting our documents into tokens is called tokenization .

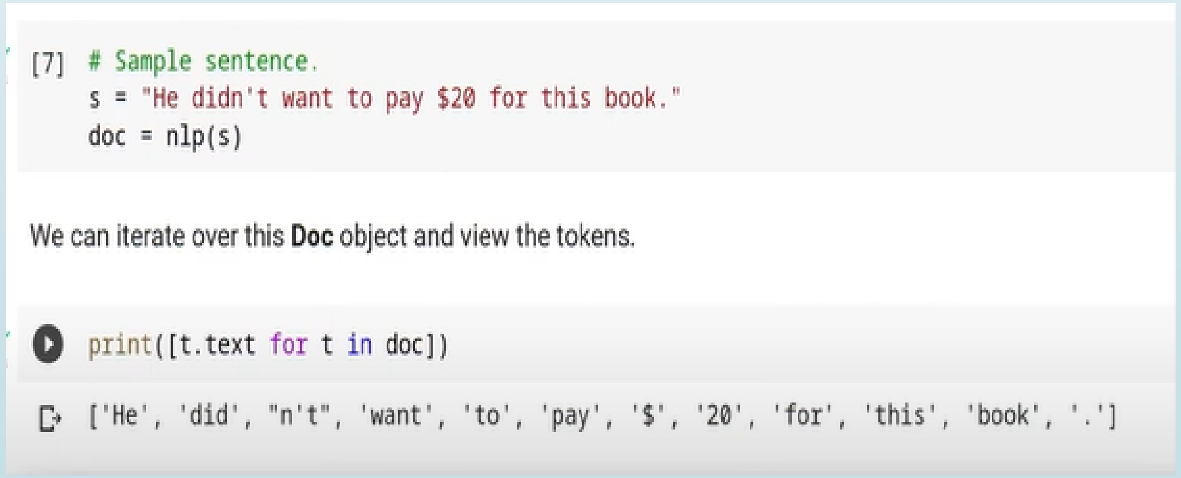


How to do tokenization

First need to import important libraries and a Small model to start our tokenization . We can do that with following code ..

```
() !pip install -U spacy==3.*
() !python -m spacy info
() import spacy
() !python -m spacy download en_core_web_sm
() nlp = spacy.load('en_core_web_sm’)
() type(nlp)
```

Next we can test tokenization with a sample text …

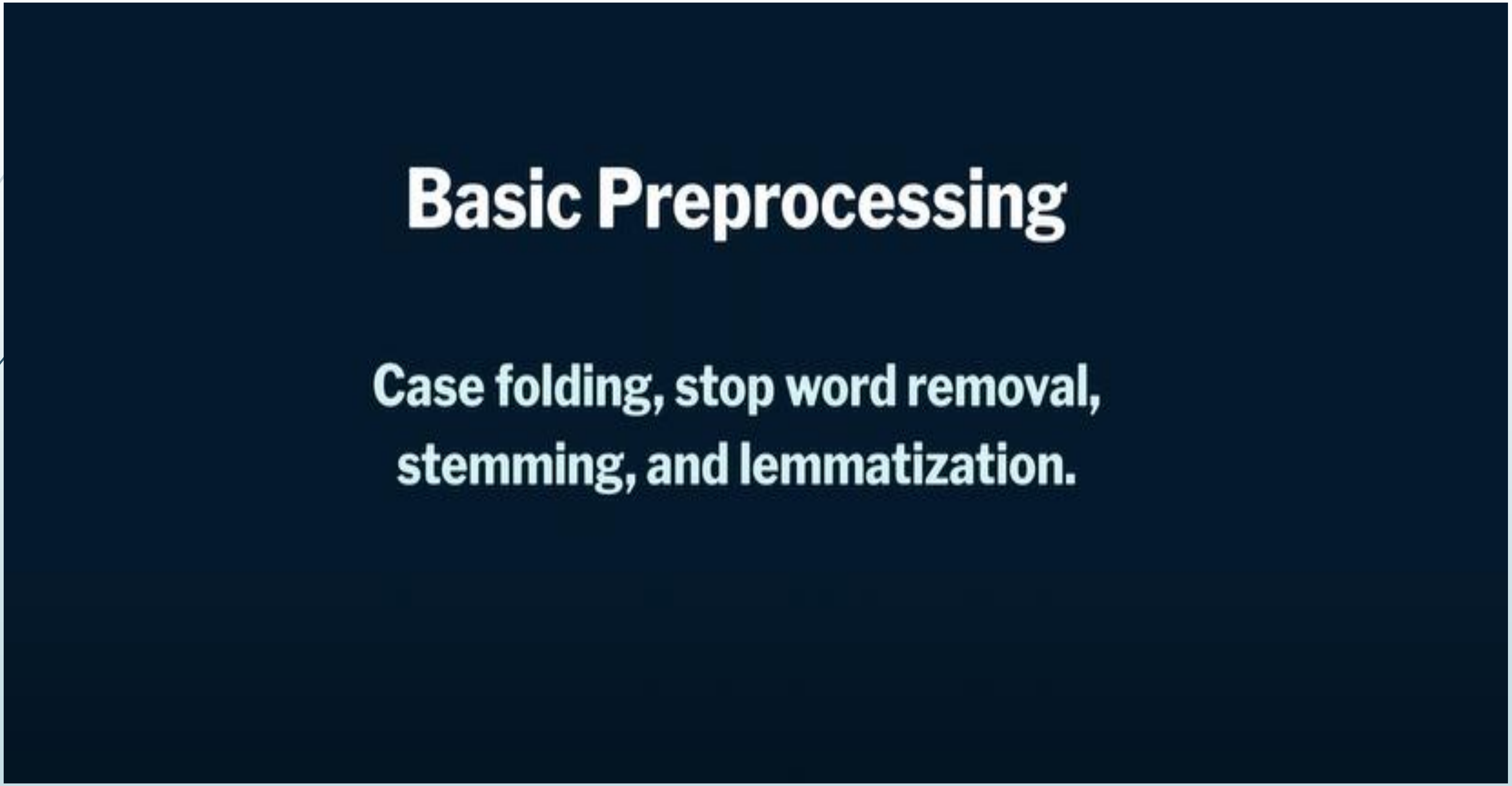


```
[7] # Sample sentence.
    s = "He didn't want to pay $20 for this book."
    doc = nlp(s)
```

We can iterate over this **Doc** object and view the tokens.

```
print([t.text for t in doc])
```

```
['He', 'did', "n't", 'want', 'to', 'pay', '$', '20', 'for', 'this', 'book', '.']
```


Basic Preprocessing

Case folding, stop word removal, stemming, and lemmatization.

Case Folding

Lower- or upper-casing all tokens.

Vocabulary: The set of all **unique** tokens in a corpus.

"Mr. Cook went into the kitchen to cook dinner."

Without c.f.

{ cook, dinner, into, kitchen, mr., the, to, went **Cook** }

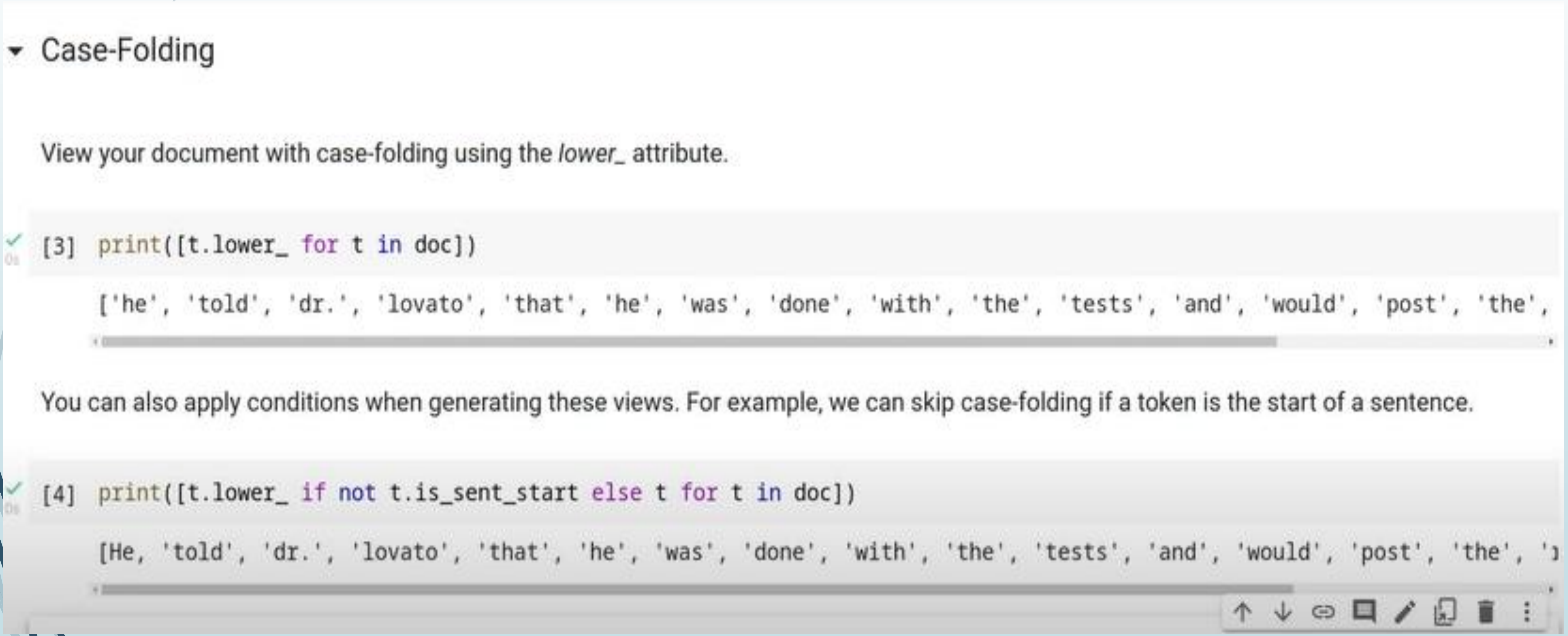
With c.f.

{ cook, dinner, into, kitchen, mr., the, to, went }

+ Smaller vocabulary can be more efficient in space and computation, and lead to higher number of search hits ("recall") in information retrieval.

− Information loss; "Cook" as a person is different from "cook" as an activity, which can lead to poorer quality search results ("precision").

Code implementation of Case folding



Case-Folding

View your document with case-folding using the *lower_* attribute.

```
[3] print([t.lower_ for t in doc])

    ['he', 'told', 'dr.', 'lovato', 'that', 'he', 'was', 'done', 'with', 'the', 'tests', 'and', 'would', 'post', 'the',
```

You can also apply conditions when generating these views. For example, we can skip case-folding if a token is the start of a sentence.

```
[4] print([t.lower_ if not t.is_sent_start else t for t in doc])

    [He, 'told', 'dr.', 'lovato', 'that', 'he', 'was', 'done', 'with', 'the', 'tests', 'and', 'would', 'post', 'the', '
```

Stop Word Removal

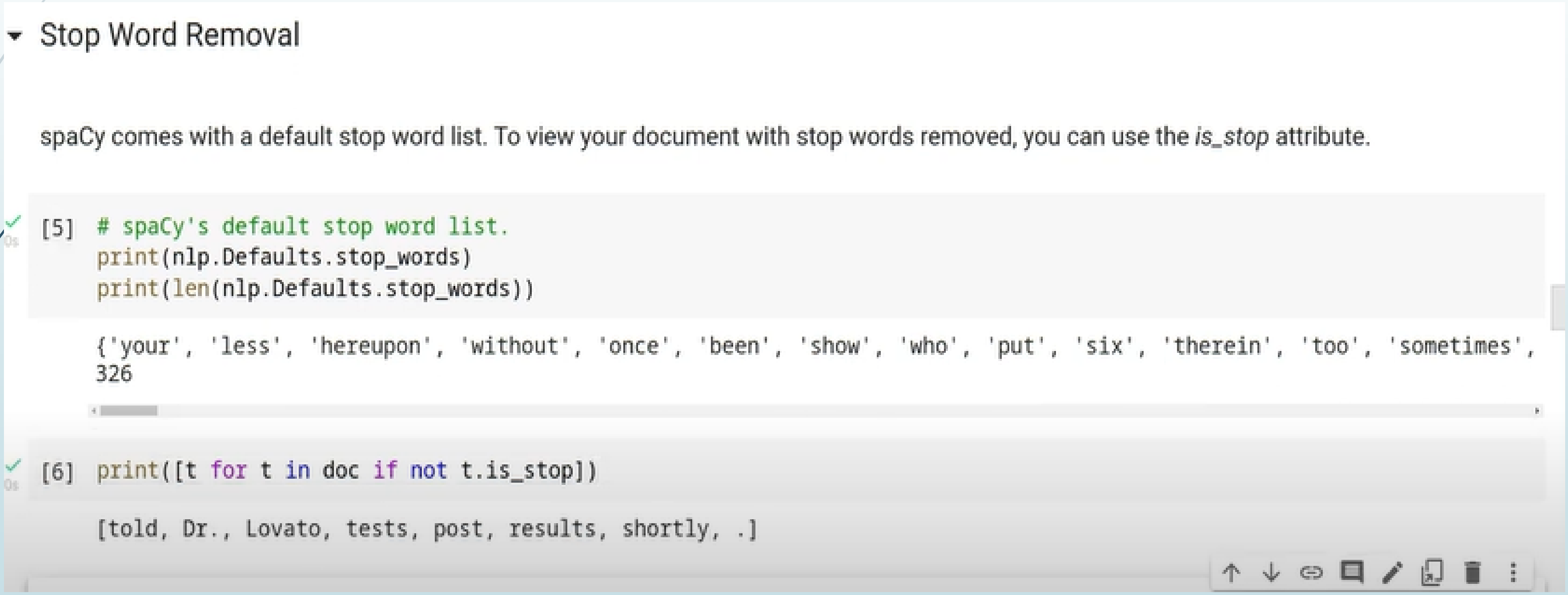
Removing words which occur frequently but carry little information.

{ the, a, of, an, this, that, ... }

Like case folding, this could lead to potential efficiency gains...

But whether we should do it depends on the task.

Code implementation of Stop Word Removal



Stop Word Removal

spaCy comes with a default stop word list. To view your document with stop words removed, you can use the *is_stop* attribute.

[5]
```
# spaCy's default stop word list.
print(nlp.Defaults.stop_words)
print(len(nlp.Defaults.stop_words))
```

```
{'your', 'less', 'hereupon', 'without', 'once', 'been', 'show', 'who', 'put', 'six', 'therein', 'too', 'sometimes',
326
```

[6]
```
print([t for t in doc if not t.is_stop])
```

```
[told, Dr., Lovato, tests, post, results, shortly, .]
```

Stemming

Removing word suffixes (and sometimes prefixes).

{ ing, s, y, ed, ... }

$$\left.\begin{array}{r}\text{“Banking”}\\ \text{“Banks”}\end{array}\right\} \Rightarrow \text{“Bank”}$$

Goal is to reduce a word to some base form.

Typically done through an algorithm, the most famous of which is *Porter's algorithm*.

Should you stem?

Similar tradeoffs to other (simple) techniques which reduce vocabulary.

+ Reduces vocabulary size and generalizes your model to behave the same for words with the same stem. May also help with new words outside of the current vocabulary.

− A stemmer can *overstem* ("university and "universe" both stem to "univers") and *understem* ("alumnus" and "alumni" stem to "alumnu" and "alumni") which can lead to poor results.

Search engines will search on both unstemmed and stemmed words and blend the results based on other relevancy metrics.

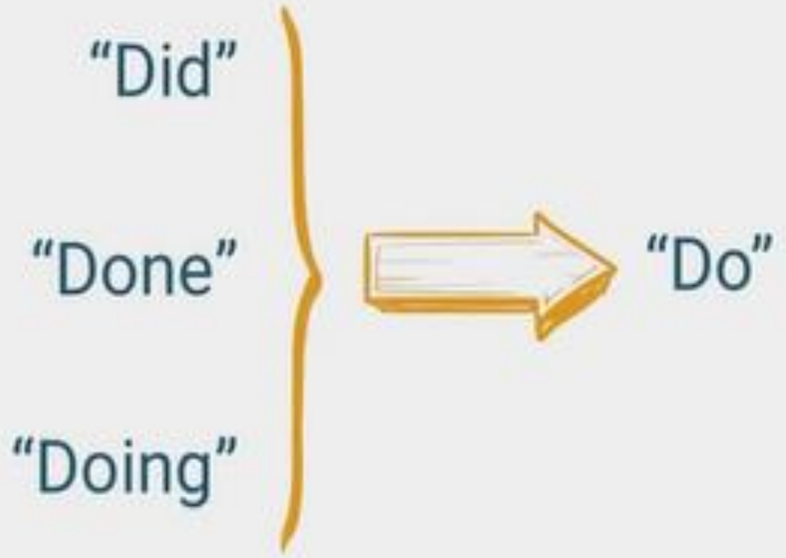
There's a better alternative to stemming...

Lemmatization

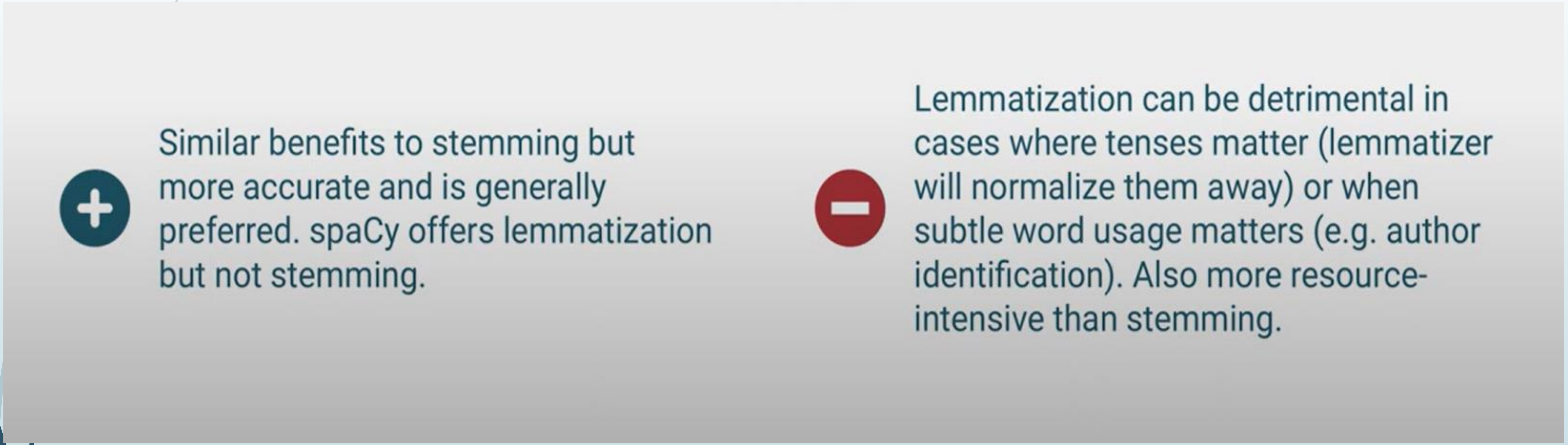
Reduce a word down to its *lemma*, or dictionary form.

(more sophisticated than stemming)

Maps different forms (e.g. inflections) of a word (and some synonyms) to its base form.



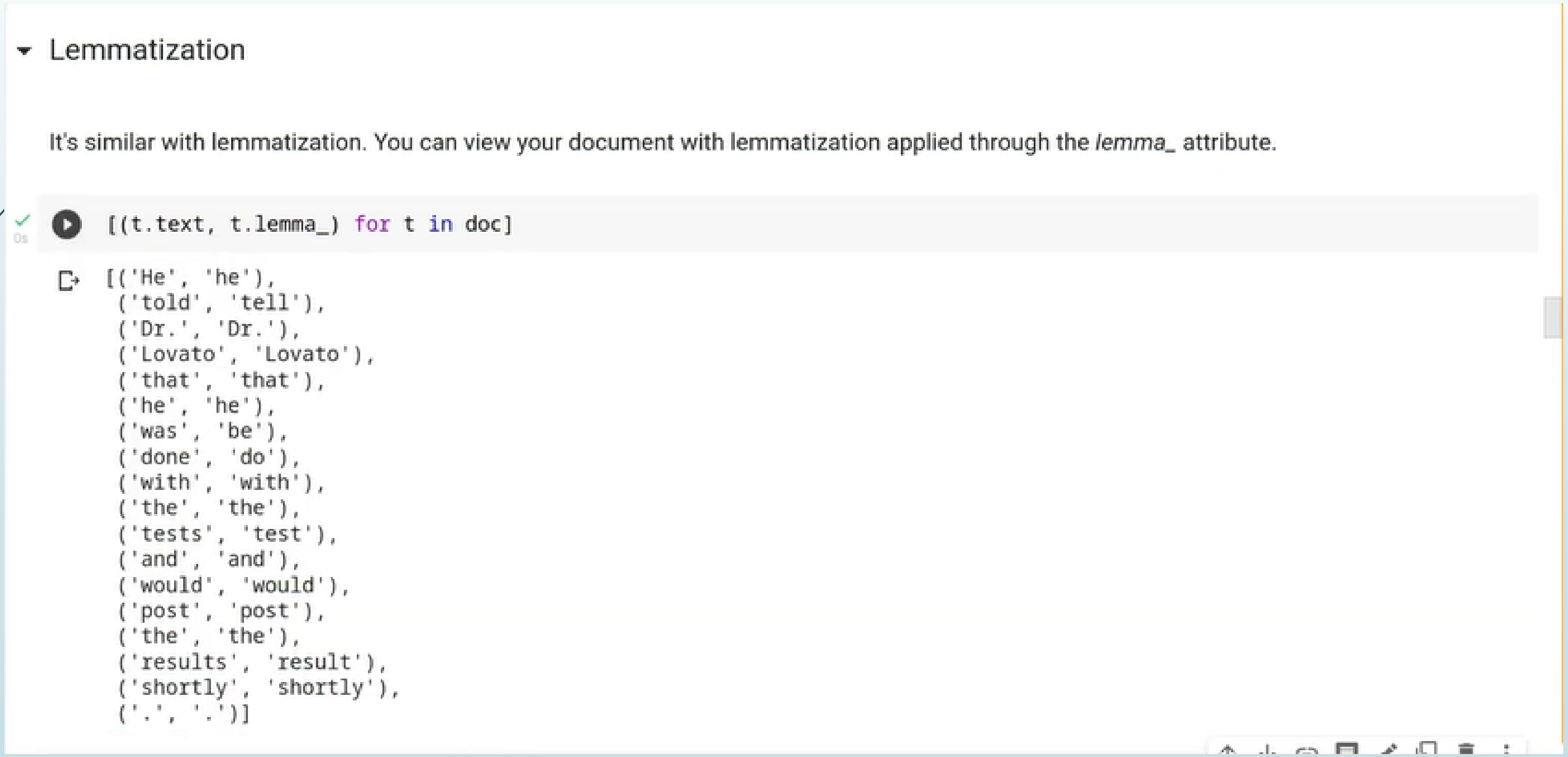
Tradeoffs of Lemmatization



➕ Similar benefits to stemming but more accurate and is generally preferred. spaCy offers lemmatization but not stemming.

➖ Lemmatization can be detrimental in cases where tenses matter (lemmatizer will normalize them away) or when subtle word usage matters (e.g. author identification). Also more resource-intensive than stemming.

Code implementation of Lemmatization



Lemmatization

It's similar with lemmatization. You can view your document with lemmatization applied through the *lemma_* attribute.

```
[(t.text, t.lemma_) for t in doc]
```

```
[('He', 'he'),
 ('told', 'tell'),
 ('Dr.', 'Dr.'),
 ('Lovato', 'Lovato'),
 ('that', 'that'),
 ('he', 'he'),
 ('was', 'be'),
 ('done', 'do'),
 ('with', 'with'),
 ('the', 'the'),
 ('tests', 'test'),
 ('and', 'and'),
 ('would', 'would'),
 ('post', 'post'),
 ('the', 'the'),
 ('results', 'result'),
 ('shortly', 'shortly'),
 ('.', '.')]
```

Thank You