Deep Learning
CSC-Elective

Instructor : Dr. Muhammad Ismail Mangrio

Slides prepared by Dr. M Asif Khan

ismail@iba-suk.edu.pk

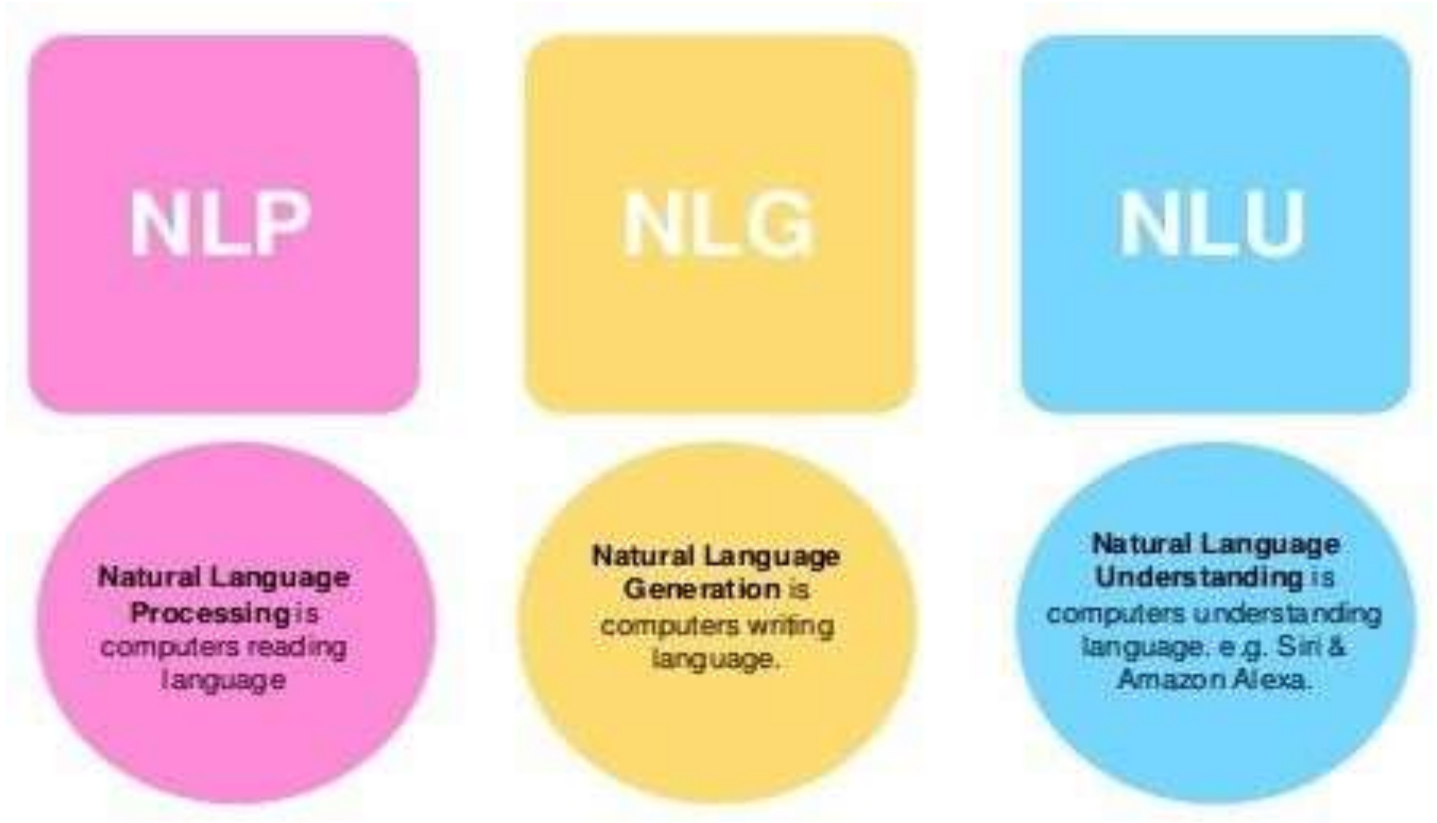
Unit 03 NLP Week 1

Contents

Natural Language Processing

- Natural language processing (NLP) is a subfield of linguistics, CS, information engineering, and AI concerned with the interactions between **computers** and **human** (natural) **languages**
  - Particularly, how to program computers to process and analyze large amounts of natural language data.
- Challenges:
  - Speech recognition
  - Natural Language Understanding (NLU)
  - Natural Language Generation (NLG)

Natural Language Processing


NLP
NLG
NLU
Natural Language Processing is computers reading language
Natural Language Generation is computers writing language.
Natural Language Understanding is computers understanding language. e.g. Siri & Amazon Alexa.

Natural Language Generation

- Based on NL-Understanding, it will suggest:
  - What should you say to the user?
  - Should be intelligent and Conversational as a human
  - Usage of structured data
  - With text and sentence-like planning.

NLP=NLU+NLG

Natural Language Vs Machine Language

- Natural vs Machine Languages:
  - Natural languages (e.g., English) evolve over time.
  - Machine languages are formally designed by humans (e.g., LISP, XML).
- Key Idea:
  - Rules in natural languages come after their use, unlike machine languages.
- Messiness of Natural Language:
  - Ambiguous, ever-changing, and lacks strict structure.
- Machine Language:
  - Highly structured, defined by formal rules.
- NLP Goal:
  - Making sense of human language despite its inherent messiness.

Natural Language Understanding

Ambiguity:

Lexical Ambiguity	**: The Tank is full of water.**
Syntactic Ambiguity	**: ill men and women get to hospital.**
Semantic Ambiguity	**: Everybody isn't here.**
Pragmatic Ambiguity	**: The Army is coming.**

Phonology – This science helps to deal with patterns present in the sound and speeches related to the sound as a physical entity.

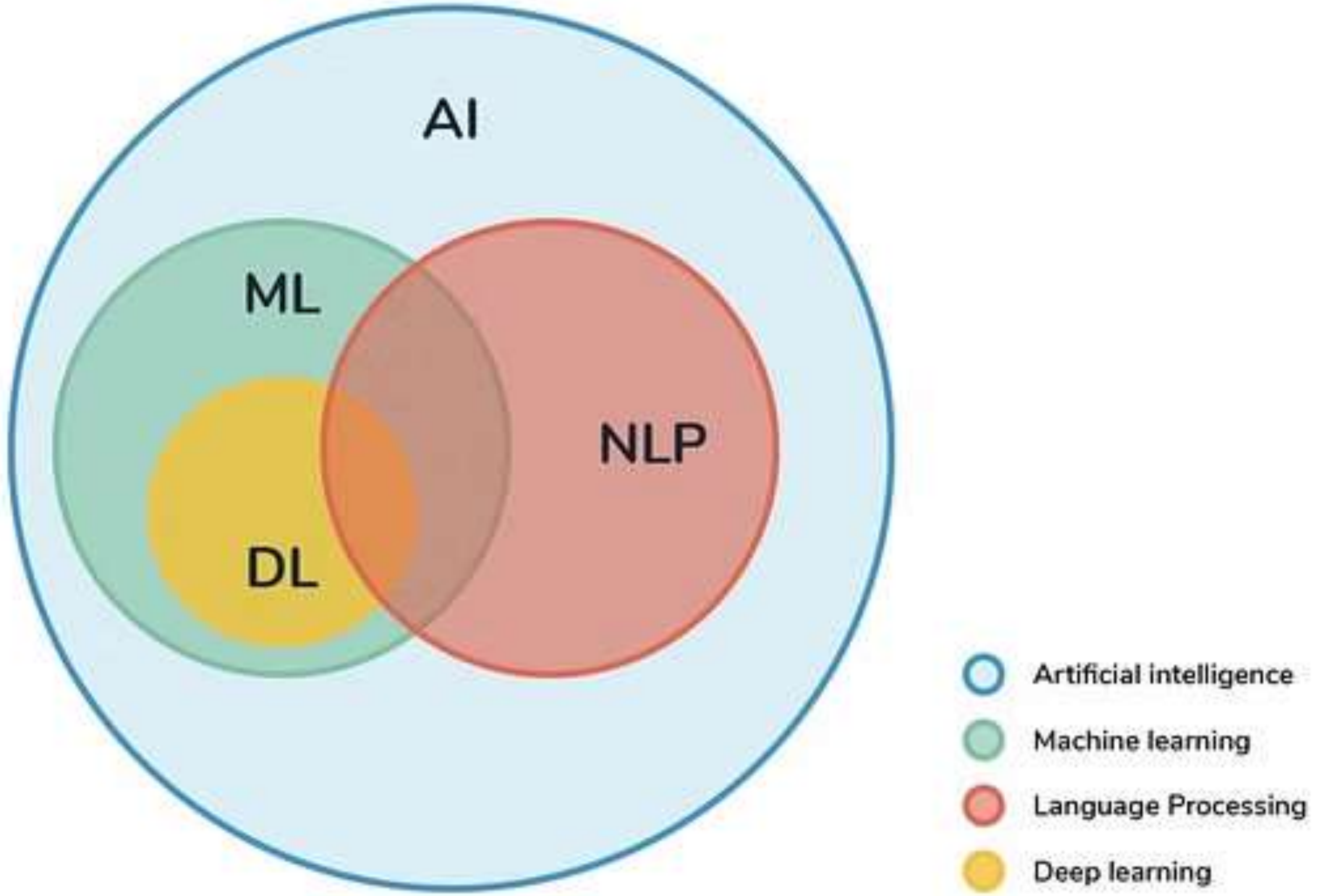
Pragmatics – This science studies the different uses of language.

Morphology – This science deals with the structure of the words and the systematic relations between them.

Syntax – This science deal with the structure of the sentences.

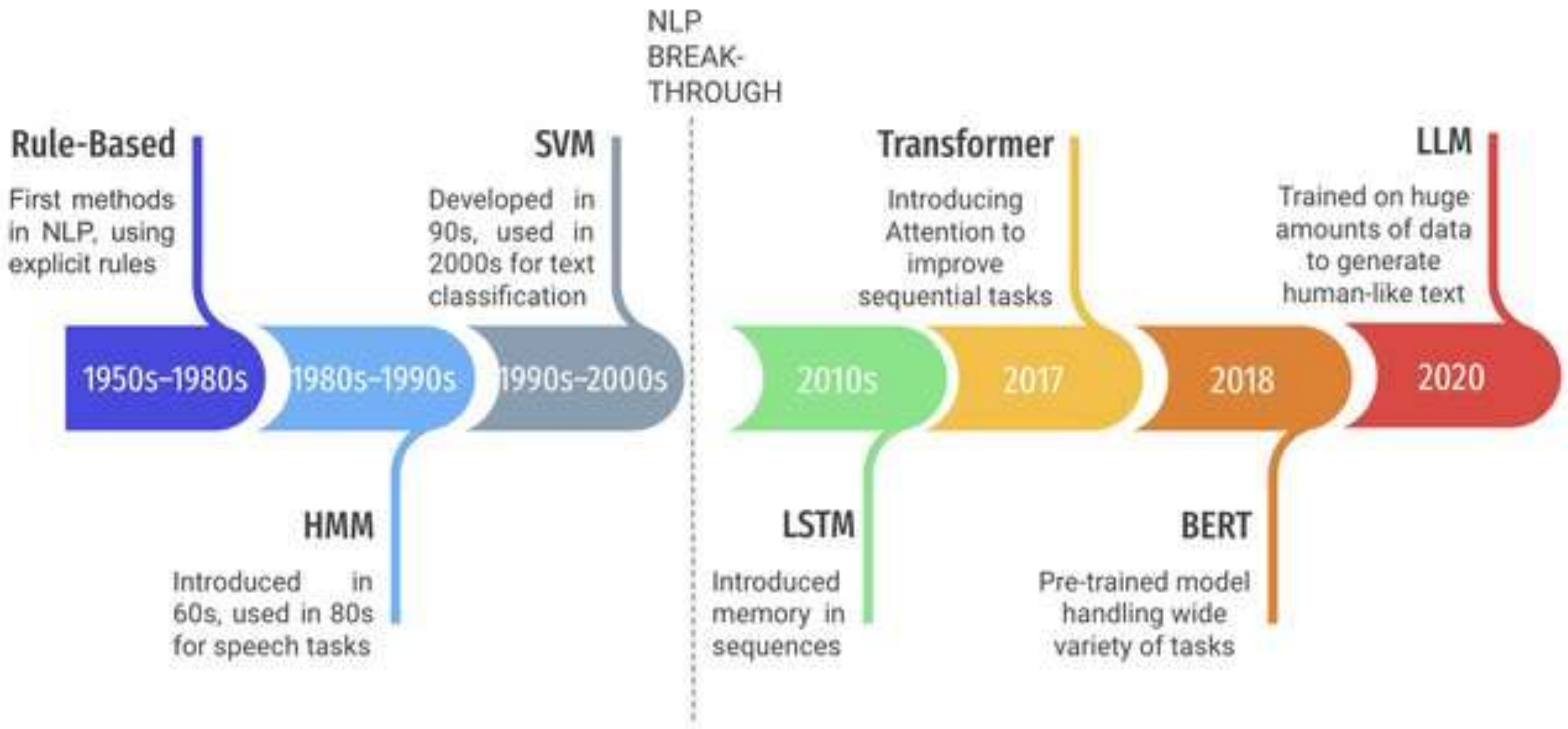
Semantics – This science deals with the literal meaning of the words, phrases as well as sentences.

Natural Language Vs Machine Language



The Shift to Machine Learning in NLP

- Data-Driven Approach:
  - From handcrafted rules to using data-driven models (late 1980s).
- Statistical Models:
  - Decision Trees, Logistic Regression.
- Key Question:
  - “Can we automate rule discovery from data?”


NLP BREAK-THROUGH
Rule-Based
First methods in NLP, using explicit rules
1950s–1980s
HMM
Introduced in 60s, used in 80s for speech tasks
1980s–1990s
SVM
Developed in 90s, used in 2000s for text classification
1990s–2000s
LSTM
Introduced memory in sequences
2010s
Transformer
Introducing Attention to improve sequential tasks
2017
BERT
Pre-trained model handling wide variety of tasks
2018
LLM
Trained on huge amounts of data to generate human-like text
2020

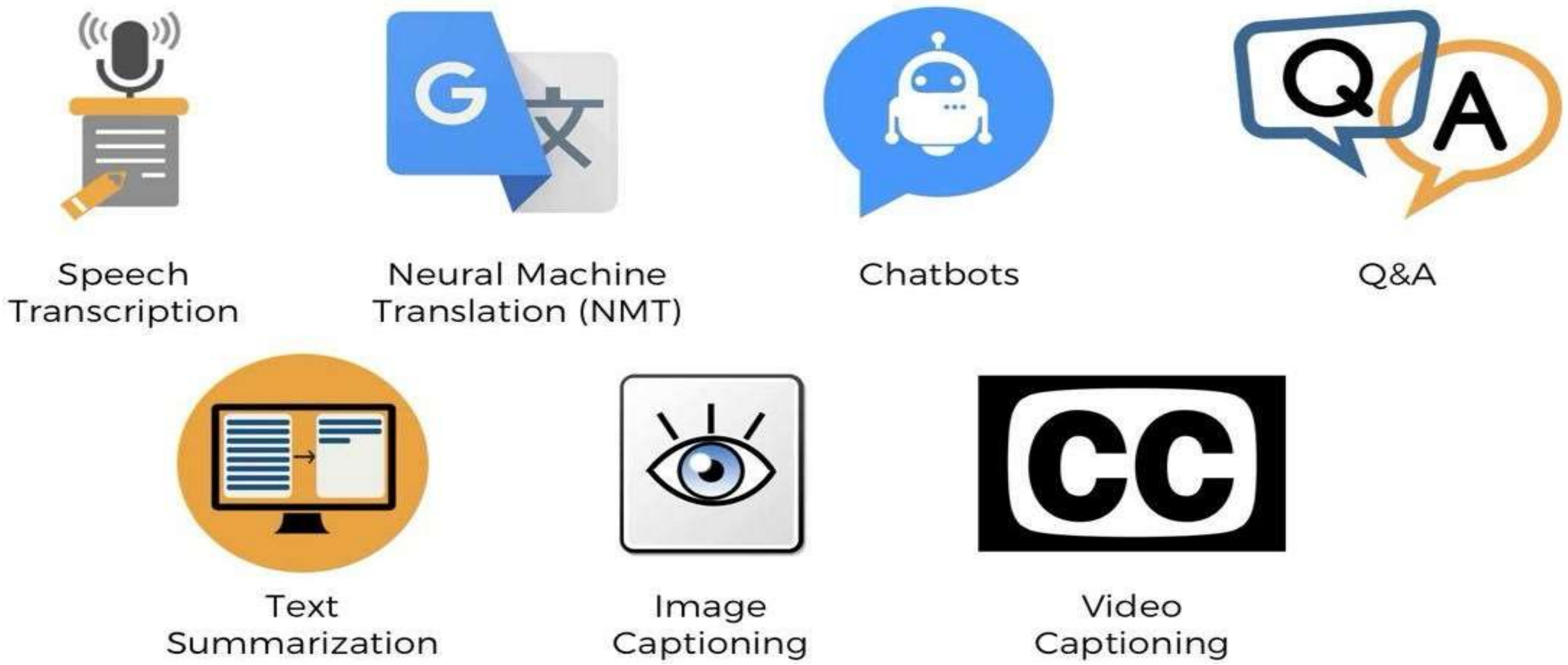
Modern NLP and Deep Learning

- Deep Learning Revolution:
  - Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) models (2014–2015).
  - Bidirectional LSTMs became the state-of-the-art for tasks like machine translation.
- Transformers (2017):
  - Replaced RNNs, unlocking significant advances in NLP.

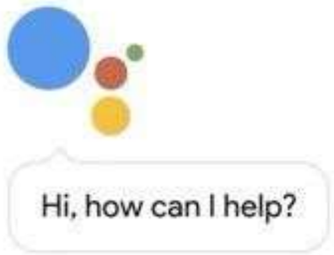
Application of NLP

- “What’s the topic of this text?” (**text classification**)
- “Does this text contain abuse?” (**content filtering**)
- “Does this text sound positive or negative?” (**sentiment analysis**)
- “What should be the next word in this incomplete sentence?” (**language modeling**)
- “How would you say this in German?” (**text translation**)
- “How would you summarize this article in one paragraph?” (**text summarization**)

Application of NLP



Application of NLP



Preparing text data

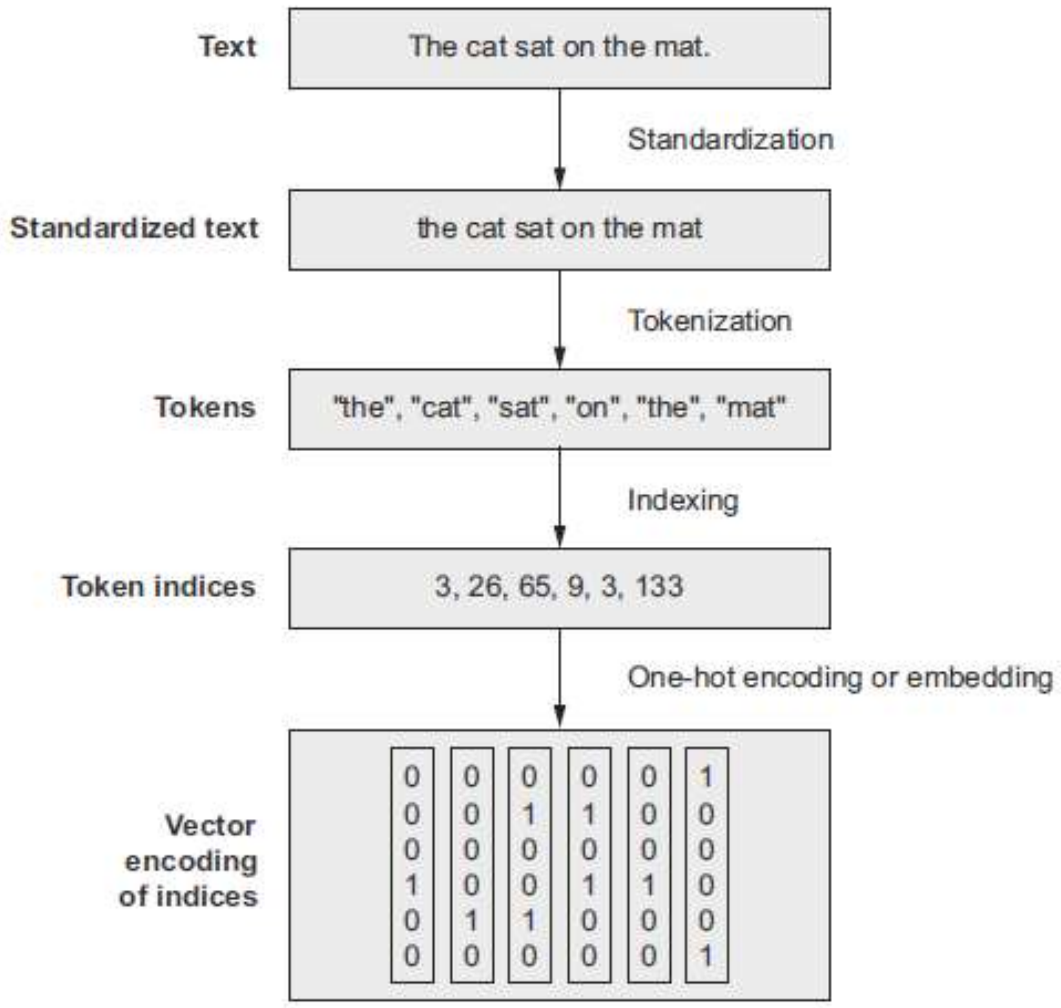


Figure 11.1 From raw text to vectors

Preparing text data (Standardization)

- Text standardization is a basic form of **feature engineering** that aims to **erase encoding differences** that you don’t want your model to have to deal with.
- Consider these two sentences:
  - “sunset came. i was staring at the Mexico sky. Isnt nature splendid??”
  - “Sunset came; I stared at the México sky. Isn’t nature splendid?”
  - They’re very similar—they’re almost identical. Yet, if you were to convert them to byte strings, they would end up with very different representations,
  - because “i” and “I” are two different characters, “Mexico” and “México” are two different words, “isnt” isn’t “isn’t,” and so on.

Preparing text data (Standardization)

- A machine learning model **doesn’t know a priori** that “i” and “I” are the same letter, that “é” is an “e” with an accent, or that “staring” and “stared” are two forms of the same verb..
- One of the simplest and most widespread standardization schemes is “**convert to lowercase** and **remove punctuation** characters.”
- Our two sentences would become:
  - “sunset came i was staring at the mexico sky isnt nature splendid”
  - “sunset came i stared at the méxico sky isnt nature splendid”

Preparing text data (Standardization)

Term	Definition	Example
Word	The smallest meaningful unit of language.	`"apple"`, `"is"`, `"sweet"`
Vocabulary	The **set of all unique words** in a dataset or corpus.	From sentences → “apple is sweet”, “apple is red”→ **Vocabulary:** {apple, is, sweet, red}
Document	A single piece of text — can be a sentence, paragraph, article, or tweet.	`"The movie was amazing!"` *(one document)*
Corpus (plural: Corpora)	A **collection of documents** used for NLP tasks like training or analysis.	1000 movie reviews = **corpus** of 1000 **documents**

Libraries / Frameworks for NLP

1. Natural Language Toolkit (NLTK)
2. TextBlob
3. CoreNLP
4. Gensim
5. spaCy
6. polyglot
7. scikit–learn
8. Pattern

Introduction to NLTK

- **What is NLTK?**
  - NLTK (Natural Language Toolkit) is a powerful Python library used for working with human language data (natural language processing).
  - It provides easy-to-use interfaces to over 50 corpora and lexical resources, along with a suite of text processing libraries.
- **Key Features:**
  - Tokenization
  - Part-of-Speech Tagging
  - Named Entity Recognition
  - Parsing and Semantic Reasoning
  - Text Classification and Sentiment Analysis

Introduction to spaCy

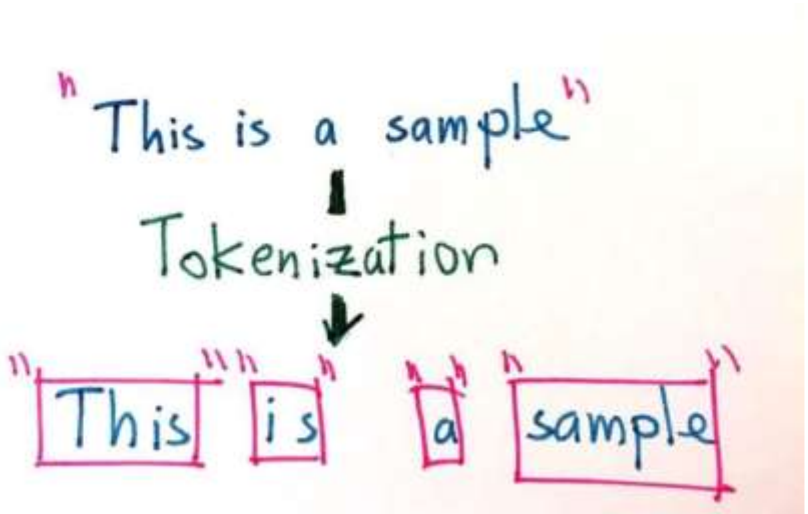
- What is **spaCy**?
- spaCy is an **open-source library** for advanced natural language processing (NLP) in Python.
- Designed specifically for production use, focusing on performance and efficiency.
- Key Features:
  - **Fast and Efficient**: Optimized for speed, allowing for large-scale processing of text data.
  - **Pre-trained Models**: Provides various pre-trained models for multiple languages that include word vectors and named entity recognition.
  - **Pipeline Processing**: Offers an easy-to-use pipeline for common NLP tasks such as tokenization, part-of-speech tagging, dependency parsing, and more.

Comparison of Popular NLP Libraries

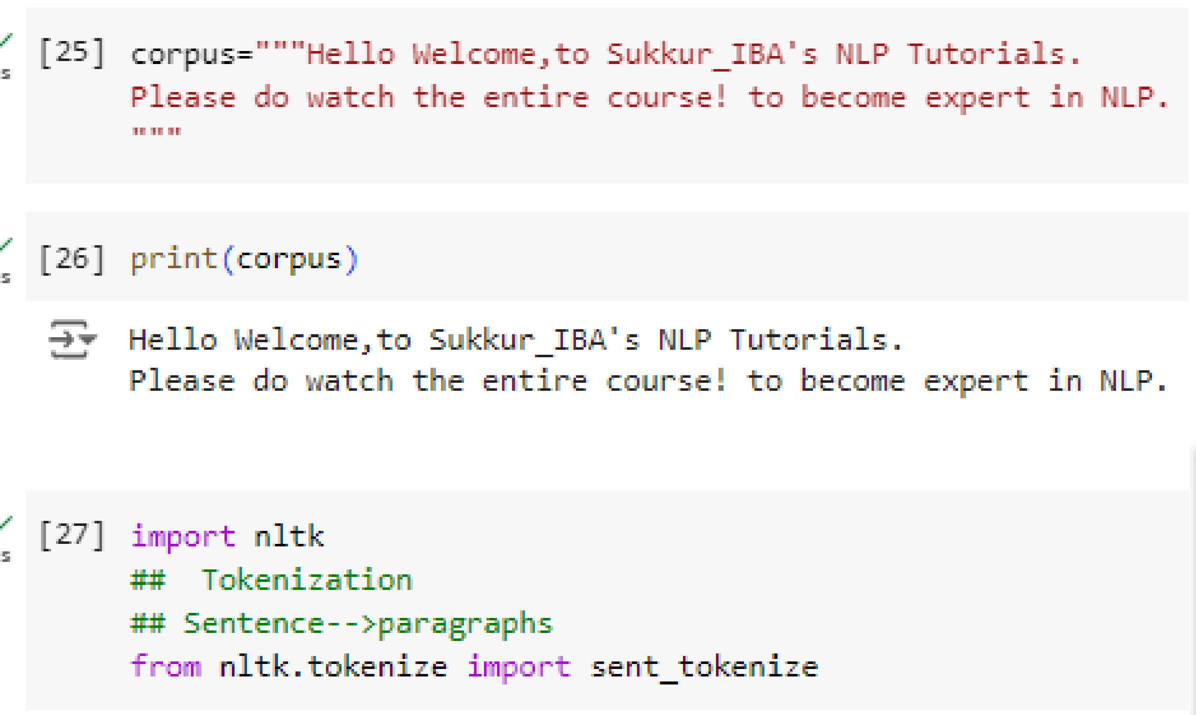
Library	Full Form / Meaning	Main Focus	Key Features	Best For
NLTK (Natural Language Toolkit)	Classic NLP library for teaching and research.	Linguistic analysis and text preprocessing.	- Tokenization, stemming, lemmatization- POS tagging, parsing- Large built-in text corpora- Easy for beginners	Learning NLP basics, small projects, education
spaCy	Industrial-strength NLP library (fast and modern).	Production-level NLP with high efficiency.	- Tokenization, POS tagging, NER- Dependency parsing- Built-in word vectors- Fast and memory-efficient	Real-world NLP applications, pipelines, deployment
scikit-learn	Machine learning library for data analysis.	Machine learning models (not language-specific).	- Feature extraction (TF–IDF, Bag of Words)- Classification, clustering, regression- Model evaluation and pipelines	Text classification (e.g., sentiment analysis), ML-based NLP tasks

Preparing text data (Tokenization)

- What is Tokenization?
  - Tokenization is the process of **breaking text into smaller units** called tokens.
  - Tokens can be **words, phrases, or sentences**.
  - Tokenization is the process of **converting a paragraph to a sentence** into tokens.
- Importance in NLP:
  - It is a **fundamental step** in text processing, allowing algorithms to understand and manipulate text data.
  - Prepares data for further analysis like parsing, part-of-speech tagging, and more.



Preparing text data (Tokenization)

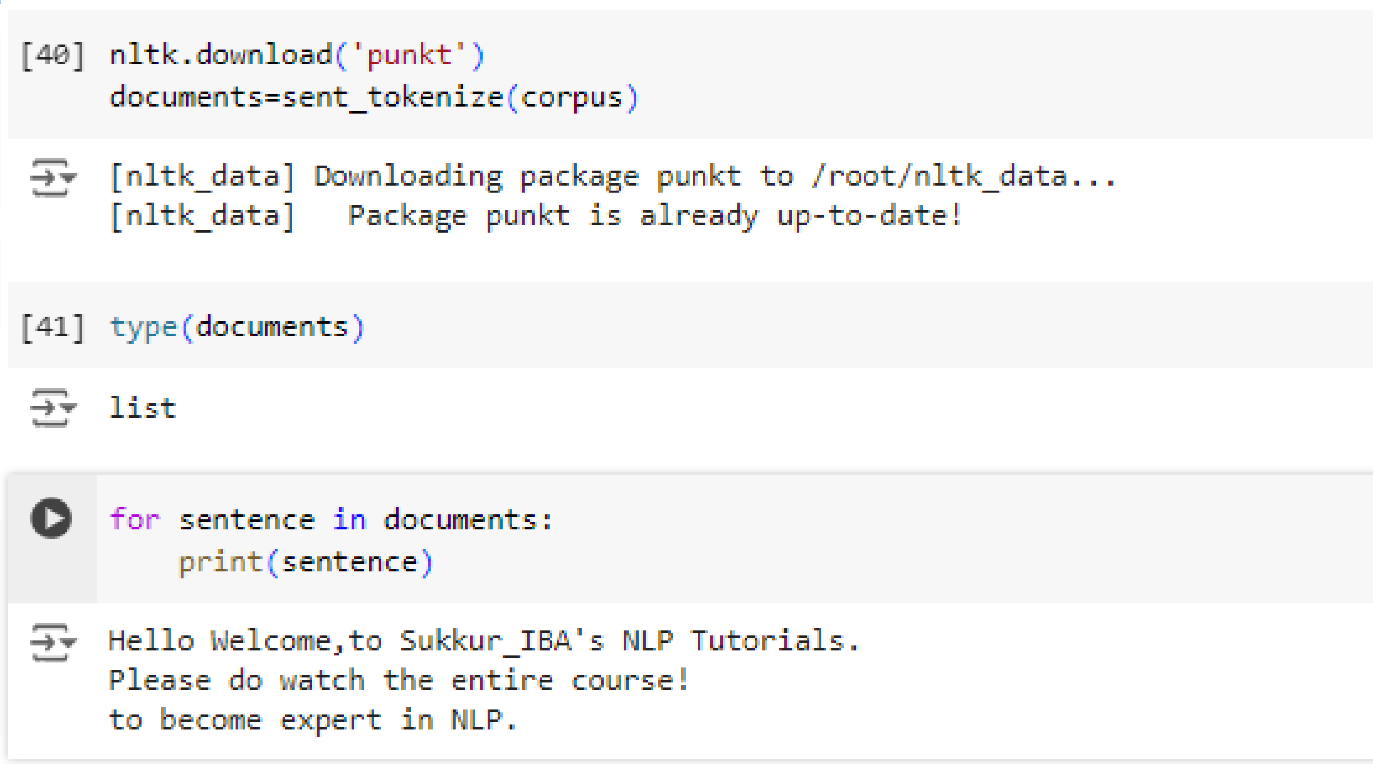


```
[25] corpus="""Hello Welcome,to Sukkur_IBA's NLP Tutorials.
     Please do watch the entire course! to become expert in NLP.
     """
```

```
[26] print(corpus)
```

```
Hello Welcome,to Sukkur_IBA's NLP Tutorials.
Please do watch the entire course! to become expert in NLP.
```

```
[27] import nltk
     ##  Tokenization
     ## Sentence-->paragraphs
     from nltk.tokenize import sent_tokenize
```



```
[40] nltk.download('punkt')
     documents=sent_tokenize(corpus)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

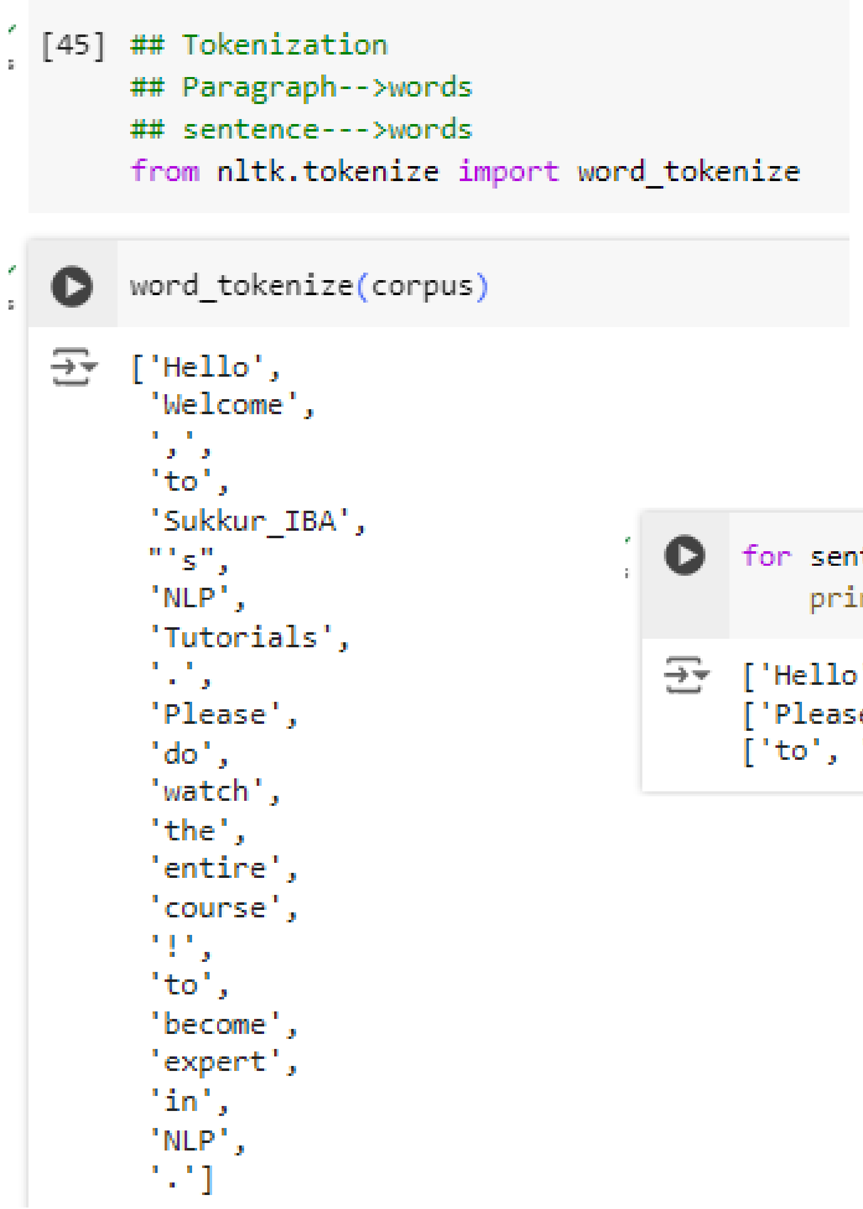
```
[41] type(documents)
```

```
list
```

```
for sentence in documents:
    print(sentence)
```

```
Hello Welcome,to Sukkur_IBA's NLP Tutorials.
Please do watch the entire course!
to become expert in NLP.
```

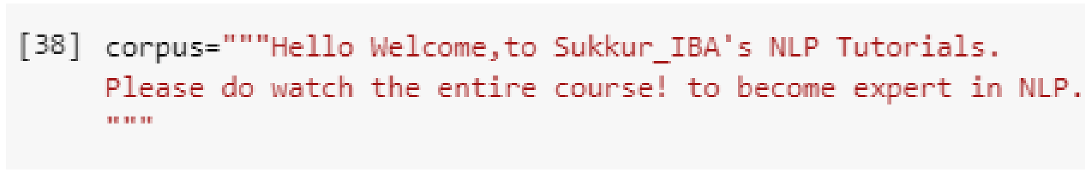
Preparing text data (Tokenization)



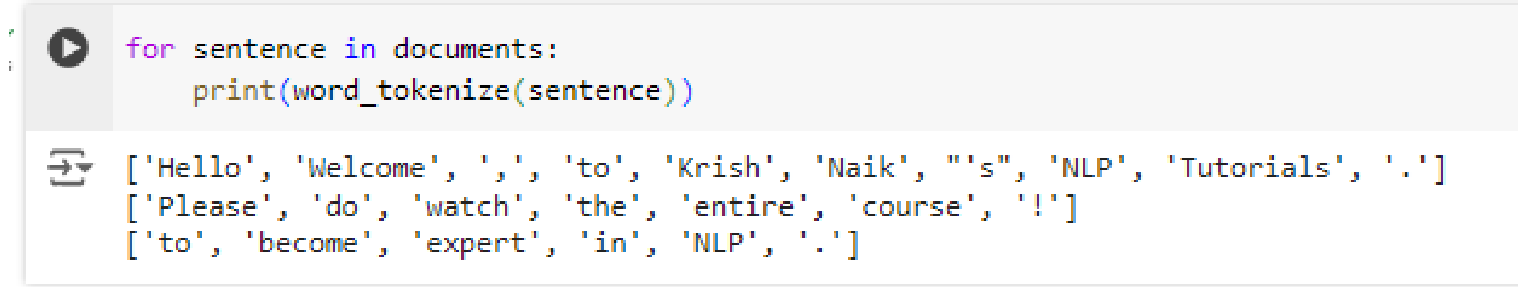
```
[45] ## Tokenization
     ## Paragraph-->words
     ## sentence--->words
     from nltk.tokenize import word_tokenize
```

```
word_tokenize(corpus)
```

```
['Hello',
 'Welcome',
 ',',
 'to',
 'Sukkur_IBA',
 "'s",
 'NLP',
 'Tutorials',
 '.',
 'Please',
 'do',
 'watch',
 'the',
 'entire',
 'course',
 '!',
 'to',
 'become',
 'expert',
 'in',
 'NLP',
 '.']
```



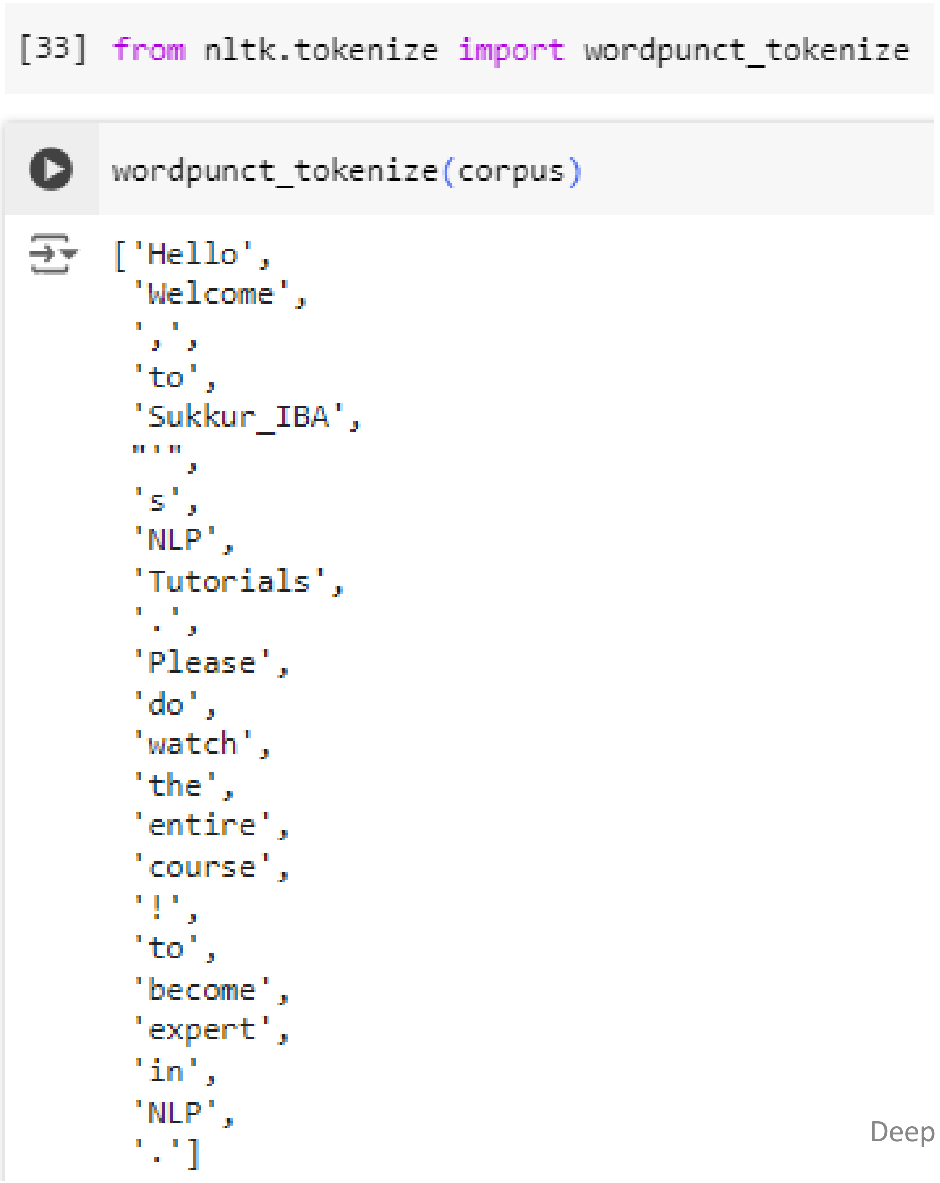
```
[38] corpus="""Hello Welcome,to Sukkur_IBA's NLP Tutorials.
     Please do watch the entire course! to become expert in NLP.
     """
```



```
for sentence in documents:
    print(word_tokenize(sentence))
```

```
['Hello', 'Welcome', ',', 'to', 'Krish', 'Naik', "'s", 'NLP', 'Tutorials', '.']
['Please', 'do', 'watch', 'the', 'entire', 'course', '!']
['to', 'become', 'expert', 'in', 'NLP', '.']
```

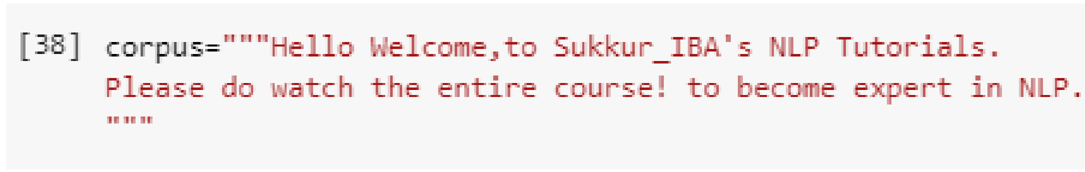
Preparing text data (Tokenization)



```
[33] from nltk.tokenize import wordpunct_tokenize
```

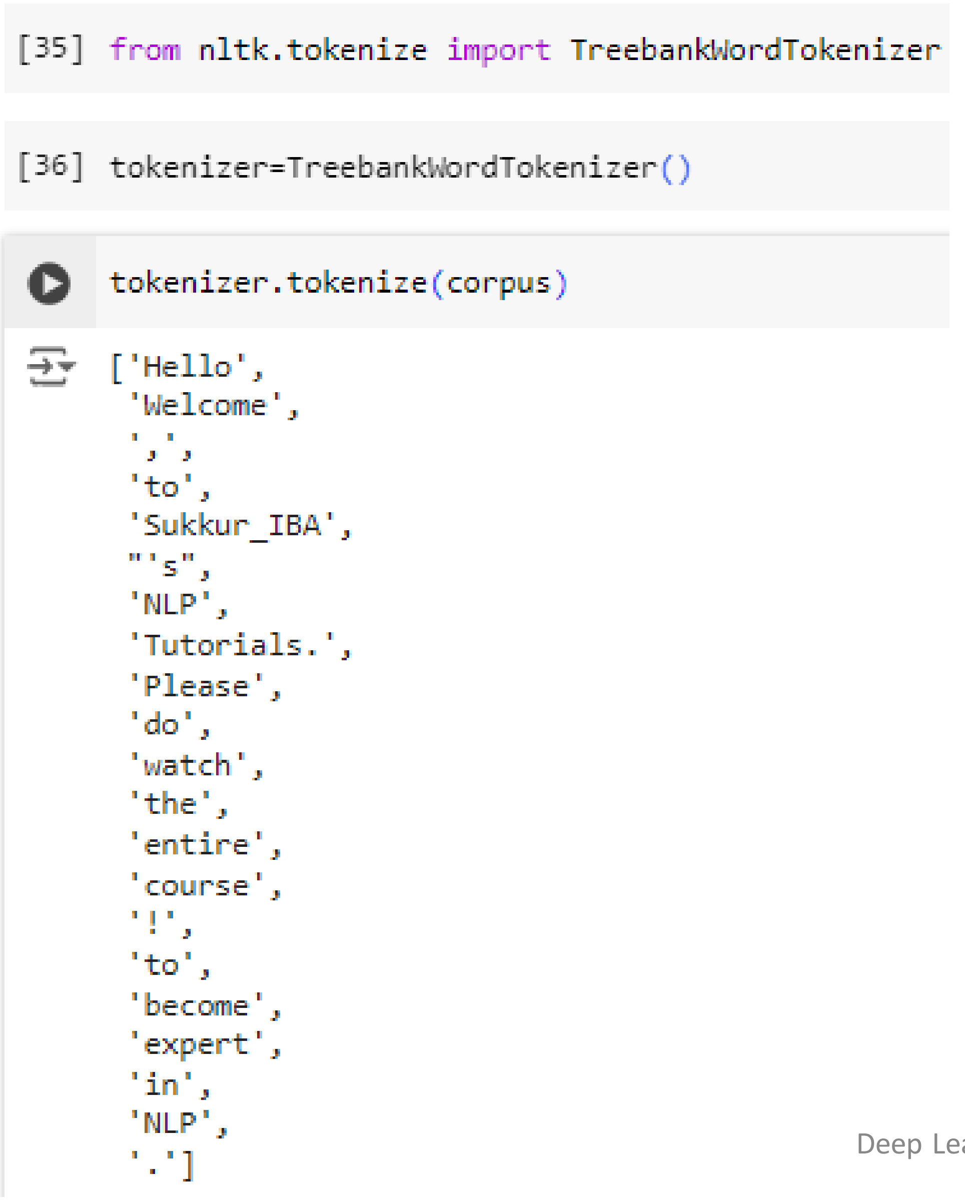
```
wordpunct_tokenize(corpus)
```

```
['Hello',
 'Welcome',
 ',',
 'to',
 'Sukkur_IBA',
 "'",
 's',
 'NLP',
 'Tutorials',
 '.',
 'Please',
 'do',
 'watch',
 'the',
 'entire',
 'course',
 '!',
 'to',
 'become',
 'expert',
 'in',
 'NLP',
 '.']
```



```
[38] corpus="""Hello Welcome,to Sukkur_IBA's NLP Tutorials.
     Please do watch the entire course! to become expert in NLP.
     """
```

Preparing text data (Tokenization)

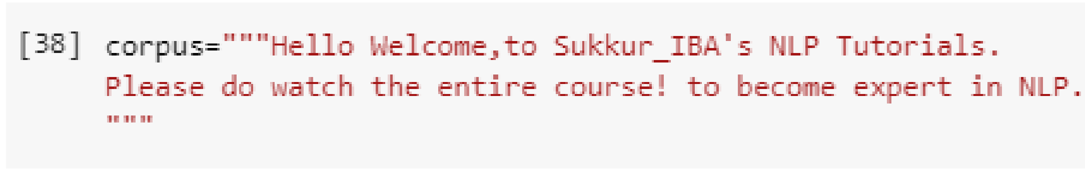


```
[35] from nltk.tokenize import TreebankWordTokenizer
```

```
[36] tokenizer=TreebankWordTokenizer()
```

```
tokenizer.tokenize(corpus)
```

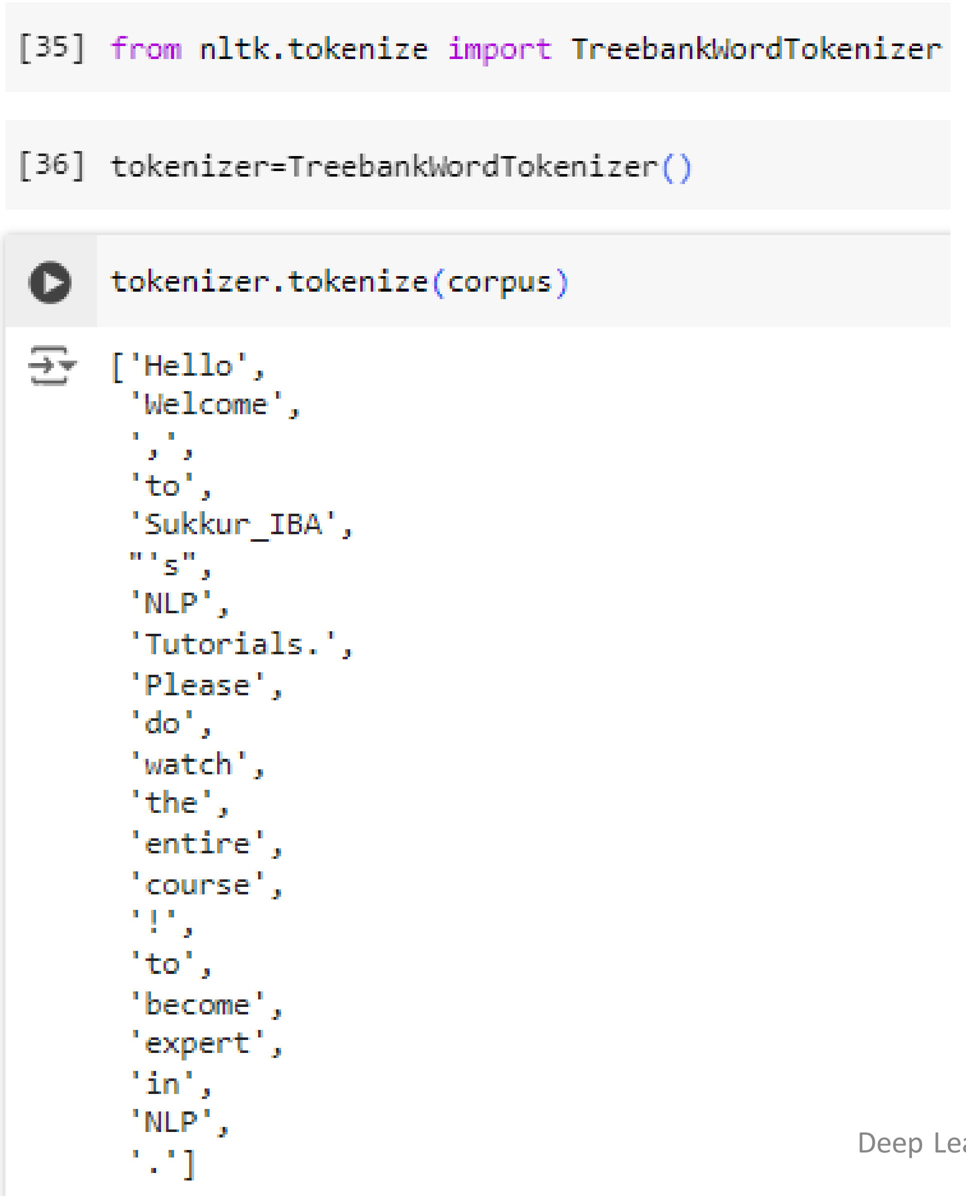
```
['Hello',
 'Welcome',
 ',',
 'to',
 'Sukkur_IBA',
 "'s",
 'NLP',
 'Tutorials.',
 'Please',
 'do',
 'watch',
 'the',
 'entire',
 'course',
 '!',
 'to',
 'become',
 'expert',
 'in',
 'NLP',
 '.']
```



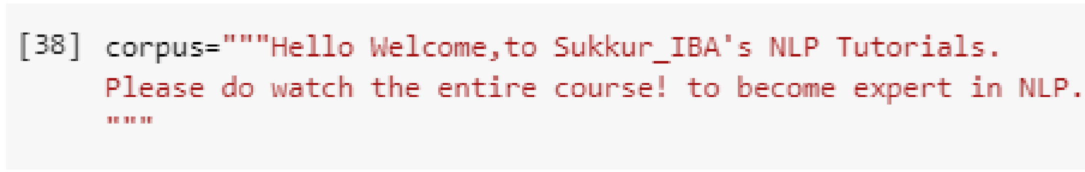
```
[38] corpus="""Hello Welcome,to Sukkur_IBA's NLP Tutorials.
     Please do watch the entire course! to become expert in NLP.
     """
```

What is the difference between word_tokenize(), wordpunct_tokenize() and TreebankTokenize()?

Preparing text data (Tokenization)



```
[35] from nltk.tokenize import TreebankWordTokenizer

[36] tokenizer=TreebankWordTokenizer()

tokenizer.tokenize(corpus)

['Hello',
 'Welcome',
 ',',
 'to',
 'Sukkur_IBA',
 "'s",
 'NLP',
 'Tutorials.',
 'Please',
 'do',
 'watch',
 'the',
 'entire',
 'course',
 '!',
 'to',
 'become',
 'expert',
 'in',
 'NLP',
 '.']
```



```
[38] corpus="""Hello Welcome,to Sukkur_IBA's NLP Tutorials.
     Please do watch the entire course! to become expert in NLP.
     """
```

What is the difference between word_tokenize(), wordpunct_tokenize() and TreebankTokenize()?
Answer: In the word_tokenize() only 's is not tokenized. The wordpunct_tokenize() will tokenize all words and punctuation marks; whereas, TreebankWordTokenize() will only tokenize the full stop of the last sentence.

Preparing text data (Stemming)

- What is Stemming?
  - Stemming is the process of **reducing words to their root** or **base form**, known as the "stem.“
  - It aims to **group together different forms of a word**, enabling better understanding and analysis of text.
  - e.g., {eating, eat and eaten} eat, {going, gone, goes} go are same words with root eat
- Purpose of Stemming:
  - **Dimensionality Reduction**: Reduces the number of unique words in the dataset, simplifying the analysis.
  - **Improved Retrieval**: Helps in matching related terms, enhancing information retrieval tasks.
  - **Text Normalization**: Prepares text data for various NLP tasks by treating different forms of a word as the same.

Preparing text data (Stemming)

- There are several types of stemming algorithms, each with its own approach and characteristics. Here are some of the most common types:

Porter Stemmer:

1. One of the most widely used stemming algorithms.
2. Uses a series of rules to iteratively **remove suffixes** from words.
3. Often **produces aggressive reductions**.

Snowball Stemmer:

1. **An improvement** on the Porter Stemmer.
2. **Supports multiple languages** and has a more extensive set of rules.
3. Designed to be more consistent and provide better results across different languages.

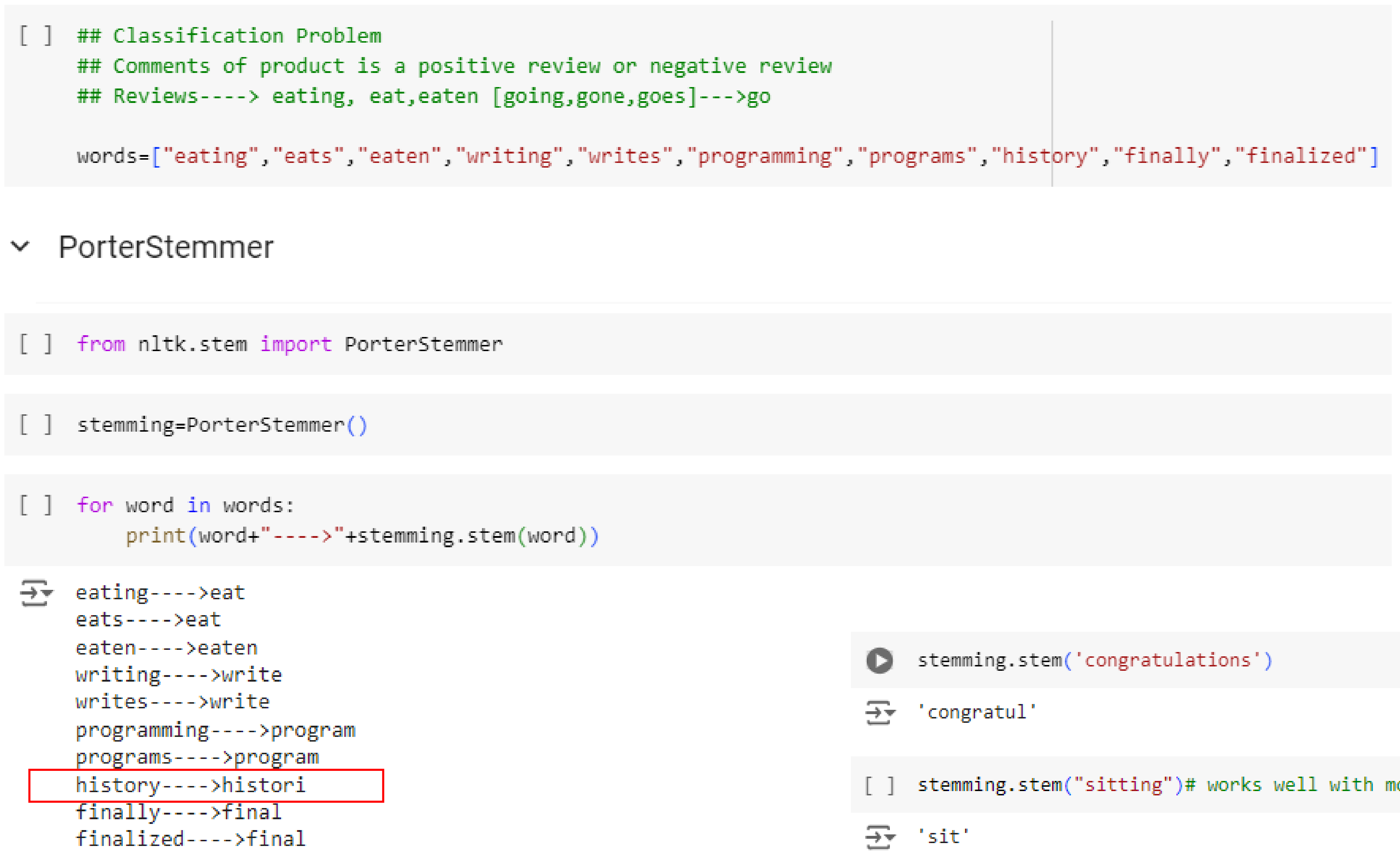
RegExp Stemmer:

1. Uses regular expressions to define **custom stemming** rules.
2. Allows for **flexibility and customization** based on specific patterns in the language or domain.

Preparing text data (Stemming)

- Stemming limitations
- Although **porter stemmer are better** compare to stemmer.
- Still there will be **some of the words there will be incorrect output** even for Snowball stemmer.
- That is the reason **Stemming is not used in chatbot** like applications.
- However, it **suits well for text classification**. e.g., Whether the given text is negative or positive. Or the email is spam or not.
- For chatbot purpose **we use lematization**.

Preparing text data (Stemming)



```
[ ] ## Classification Problem
    ## Comments of product is a positive review or negative review
    ## Reviews----> eating, eat,eaten [going,gone,goes]--->go

    words=["eating","eats","eaten","writing","writes","programming","programs","history","finally","finalized"]
```

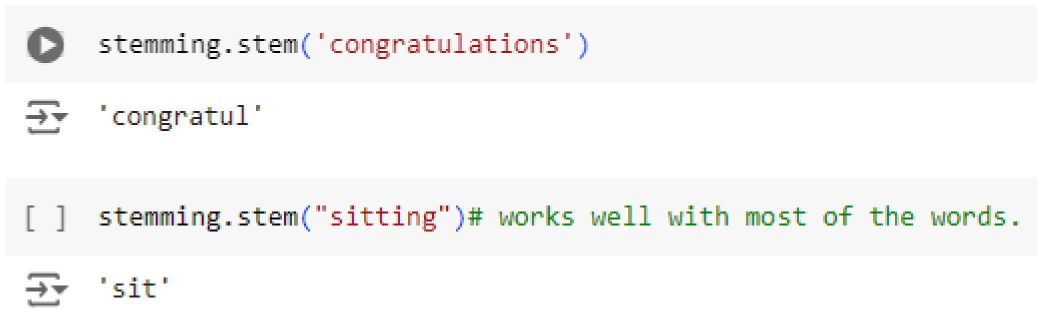
PorterStemmer

```
[ ] from nltk.stem import PorterStemmer
```

```
[ ] stemming=PorterStemmer()
```

```
[ ] for word in words:
        print(word+"---->"+stemming.stem(word))
```

```
eating---->eat
eats---->eat
eaten---->eaten
writing---->write
writes---->write
programming---->program
programs---->program
history---->histori
finally---->final
finalized---->final
```



```
stemming.stem('congratulations')
```

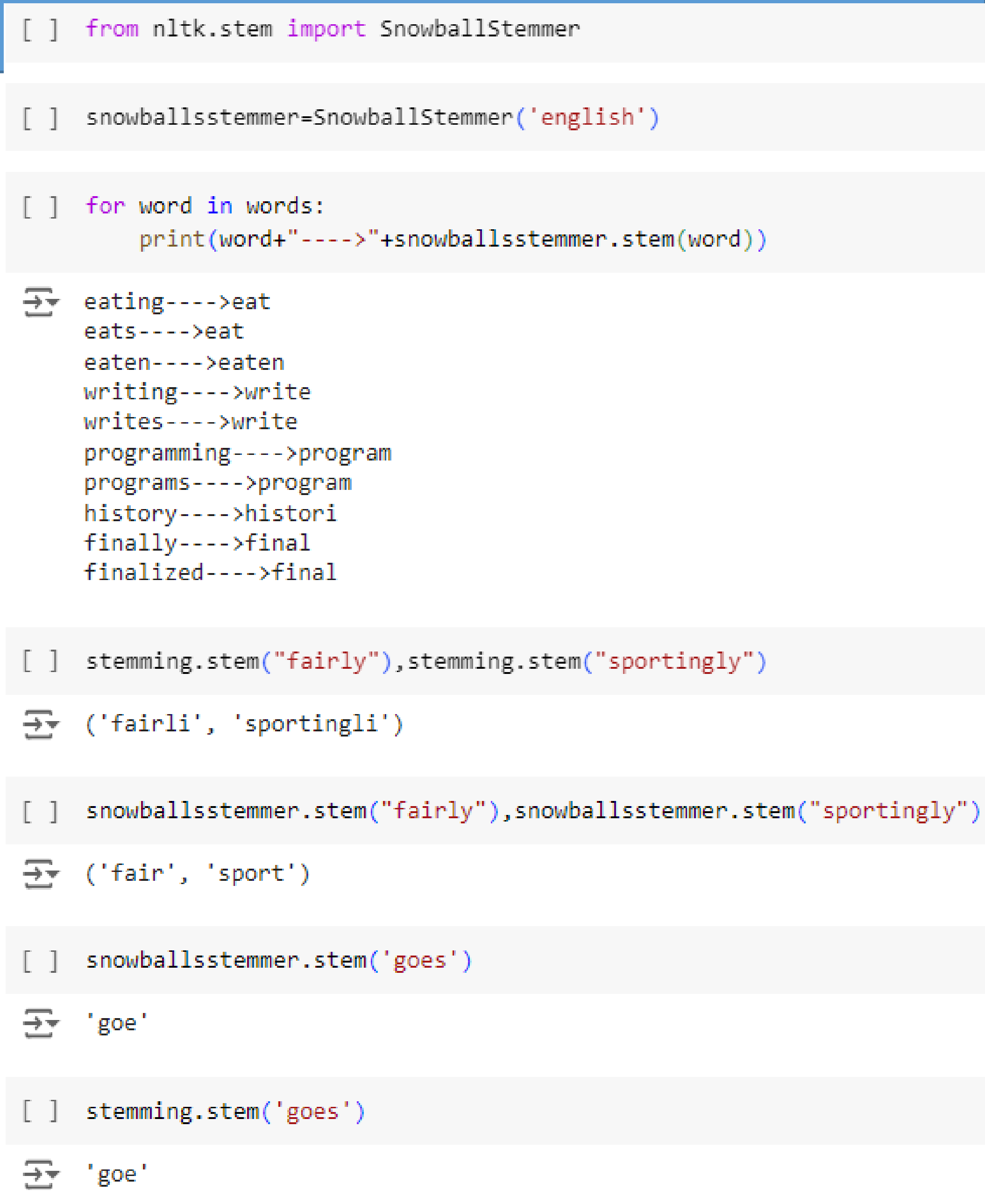
```
'congratul'
```

```
[ ] stemming.stem("sitting")# works well with most of the words.
```

```
'sit'
```

Preparing text data (Stemming)

- It is a stemming algorithm which is also known as the Porter2 stemming algorithm as it is a better version of the Porter Stemmer since some issues of it were fixed in this stemmer.
- Still there will be some of the words there will be incorrect output even for Snowball stemmer.
- That is the reason Stemming is not used in chatbot like applications.
- However, it suits well for text classification. e.g., Whether the given text is negative or positive. Or the email is spam or not.
- For chatbot purpose we used lematization.



```
[ ] from nltk.stem import SnowballStemmer
```

```
[ ] snowballsstemmer=SnowballStemmer('english')
```

```
[ ] for word in words:
        print(word+"---->"+snowballsstemmer.stem(word))
```

```
eating---->eat
eats---->eat
eaten---->eaten
writing---->write
writes---->write
programming---->program
programs---->program
history---->histori
finally---->final
finalized---->final
```

```
[ ] stemming.stem("fairly"),stemming.stem("sportingly")
```

```
('fairli', 'sportingli')
```

```
[ ] snowballsstemmer.stem("fairly"),snowballsstemmer.stem("sportingly")
```

```
('fair', 'sport')
```

```
[ ] snowballsstemmer.stem('goes')
```

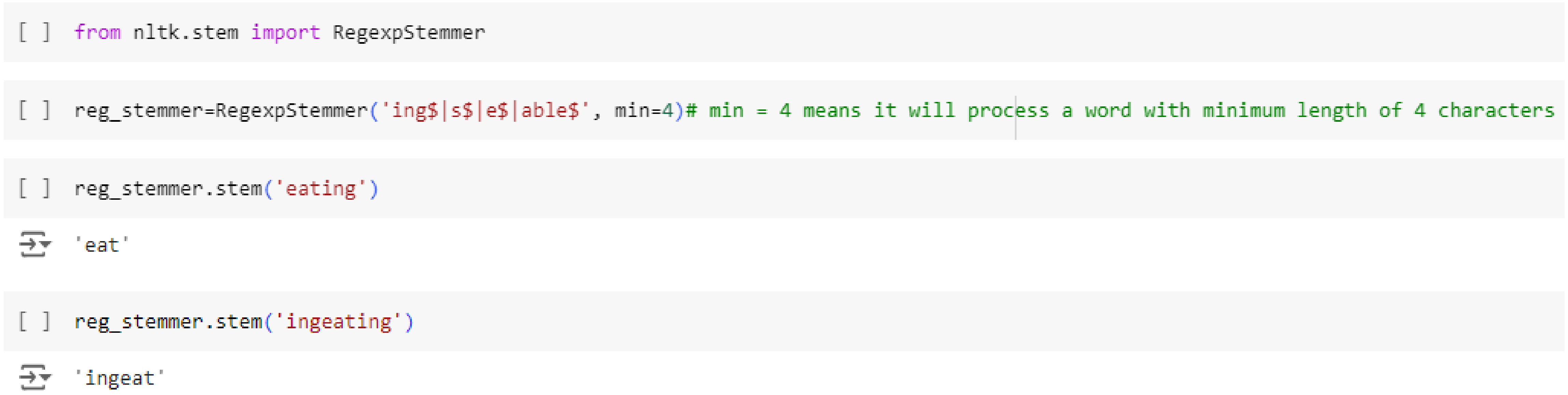
```
'goe'
```

```
[ ] stemming.stem('goes')
```

```
'goe'
```

Preparing text data (Stemming)

- NLTK has RegexpStemmer class, with the help of which we can easily implement Regular Expression Stemmer algorithms.
- It basically takes a single regular expression and removes any prefix or suffix that matches the expression. Let us see an example.



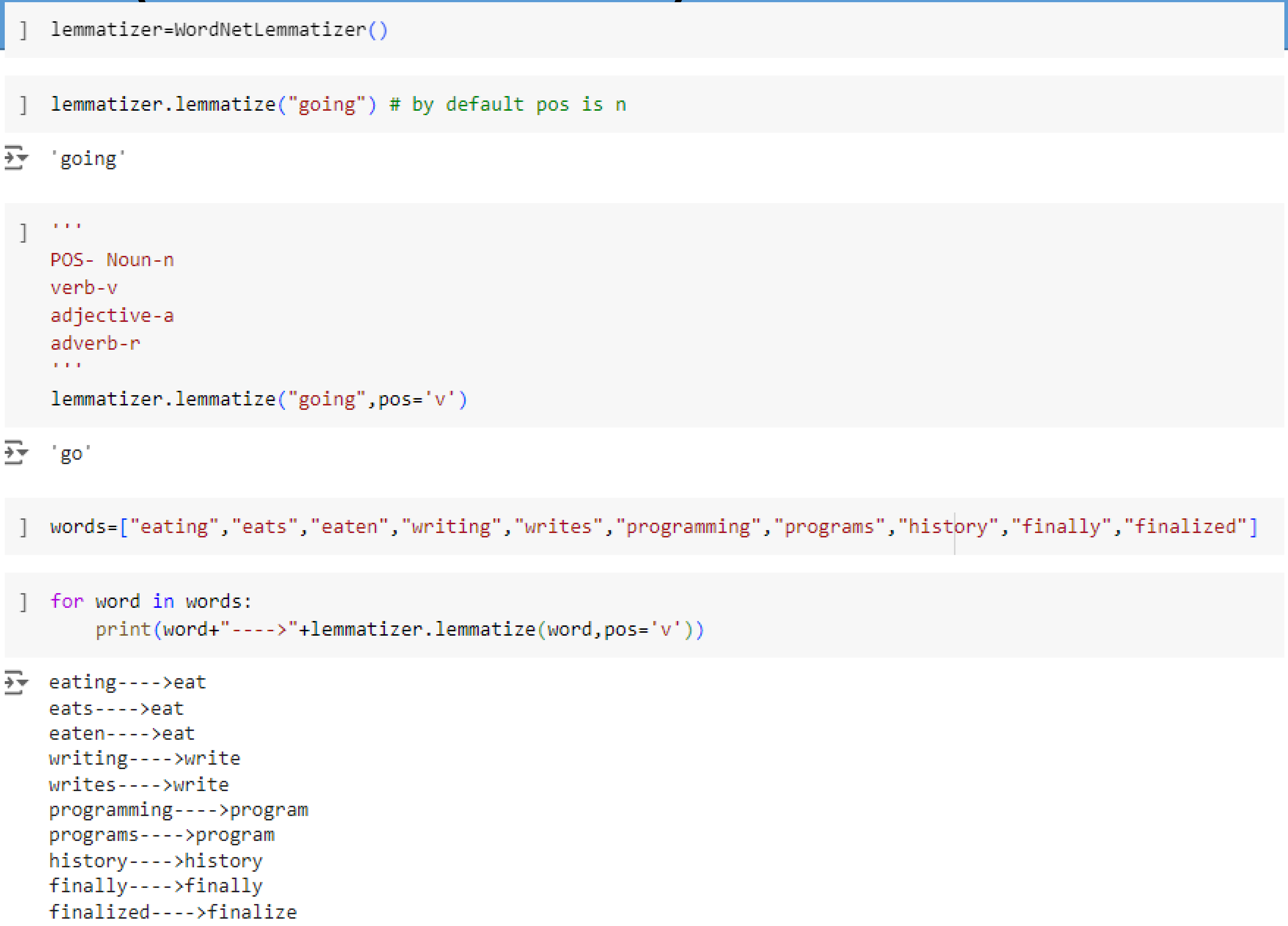
```
[ ]  from nltk.stem import RegexpStemmer

[ ]  reg_stemmer=RegexpStemmer('ing$|s$|e$|able$', min=4)# min = 4 means it will process a word with minimum length of 4 characters

[ ]  reg_stemmer.stem('eating')

     'eat'

[ ]  reg_stemmer.stem('ingeating')

     'ingeat'
```

Preparing text data (Lemmatization)

- **Stemming**: Reduces words to their base or root form by removing prefixes and suffixes, often resulting in non-dictionary forms.
- **Lemmatization**: Converts words to their base or dictionary form (lemma) using a morphological analysis, ensuring the result is a valid word.
- **Approach:**
  - **Stemming**: Uses simple algorithms and heuristic rules, often ignoring context and part of speech.
  - **Lemmatization**: Considers the **context and grammatical role** of the word, leading to more **accurate reductions**.
- **Output:**
  - **Stemming**: Can produce stems that are not actual words (e.g., “mice" → “mic”, “better“ → “better”).
  - **Lemmatization**: Always results in valid words (e.g., “mice" → “mouse", "better" → "good").

Preparing text data (Lemmatization)

- NLTK provides WordNetLemmatizer class which is a thin wrapper around the wordnet corpus.
- This class uses morphy() function to the WordNet CorpusReader class to find a lemma.
- Let us understand it with an example – It is used in Q&A, chatbots, text summarization etc.



```
lemmatizer=WordNetLemmatizer()
```

```
lemmatizer.lemmatize("going") # by default pos is n
```

```
'going'
```

```
'''
POS- Noun-n
verb-v
adjective-a
adverb-r
'''
lemmatizer.lemmatize("going",pos='v')
```

```
'go'
```

```
words=["eating","eats","eaten","writing","writes","programming","programs","history","finally","finalized"]
```

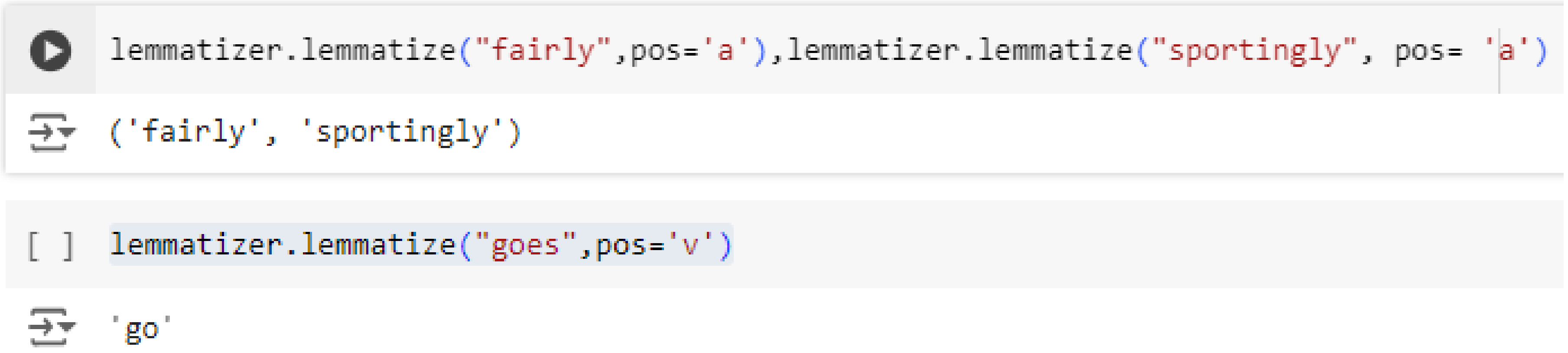
```
for word in words:
    print(word+"---->"+lemmatizer.lemmatize(word,pos='v'))
```

```
eating---->eat
eats---->eat
eaten---->eat
writing---->write
writes---->write
programming---->program
programs---->program
history---->history
finally---->finally
finalized---->finalize
```

SUKKUR IBA UNIVERSITY COMPUTER SCIENCE

Preparing text data (Lemmatization)

- Both **sportingly** and **fairly are adverb** that's why they are not changed.



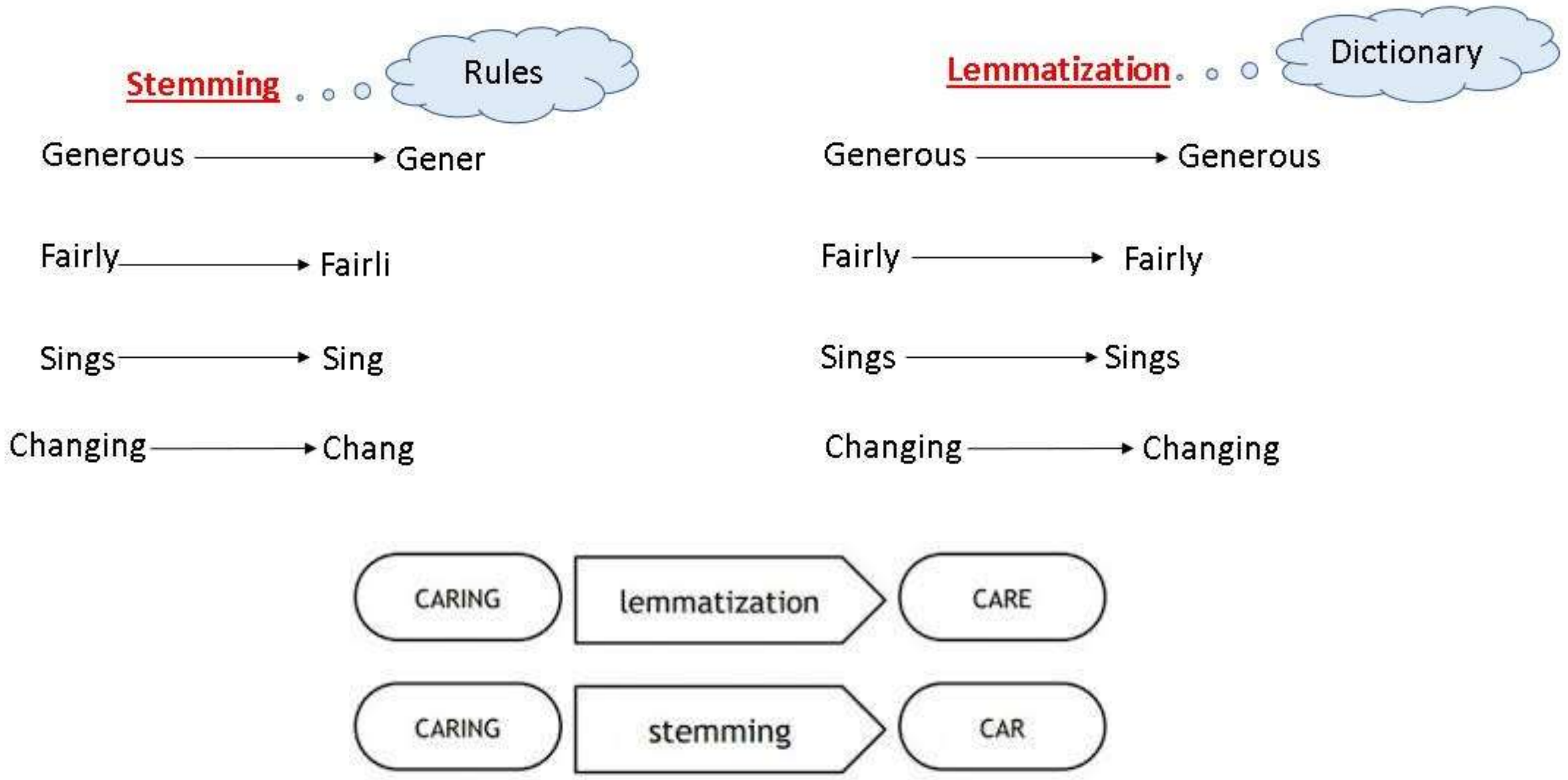
```
lemmatizer.lemmatize("fairly",pos='a'),lemmatizer.lemmatize("sportingly", pos= 'a')
```

```
('fairly', 'sportingly')
```

```
[ ] lemmatizer.lemmatize("goes",pos='v')
```

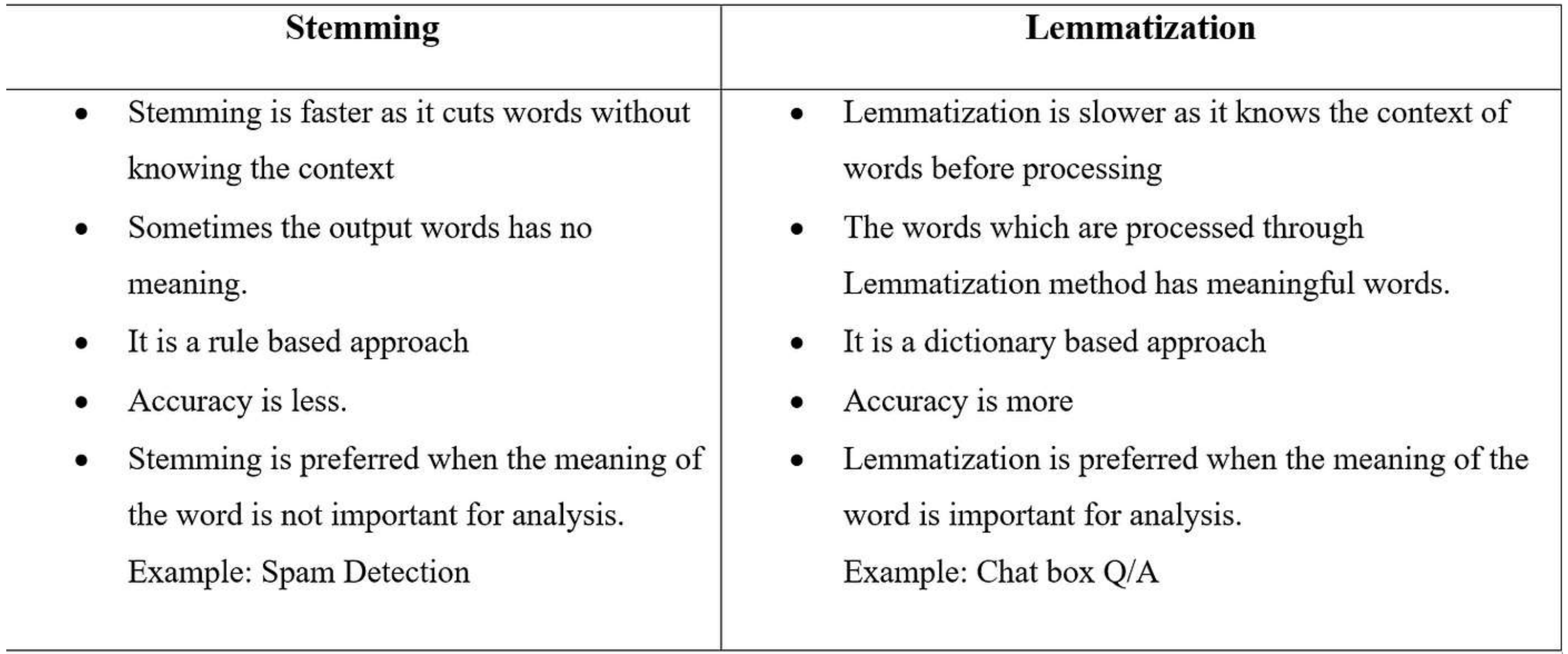
```
'go'
```

Stemming vs Lemmatization



SUKKUR IBA UNIVERSITY COMPUTER SCIENCE

Stemming vs Lemmatization



Stemming	**Lemmatization**
• Stemming is faster as it cuts words without knowing the context • Sometimes the output words has no meaning. • It is a rule based approach • Accuracy is less. • Stemming is preferred when the meaning of the word is not important for analysis. Example: Spam Detection	• Lemmatization is slower as it knows the context of words before processing • The words which are processed through Lemmatization method has meaningful words. • It is a dictionary based approach • Accuracy is more • Lemmatization is preferred when the meaning of the word is important for analysis. Example: Chat box Q/A

SUKKUR IBA UNIVERSITY
COMPUTER SCIENCE

Preparing text data (Stop words)

- **Stop words** are frequently used words in a language that add little or no value to the meaning of a sentence when analyzing text.

English Stop Words	**Examples**
Articles	a, an, the
Prepositions	in, on, at, by
Pronouns	he, she, it, they
Conjunctions	and, or, but, because
Auxiliary Verbs	is, am, are, was, were

Preparing text data (Stop words)

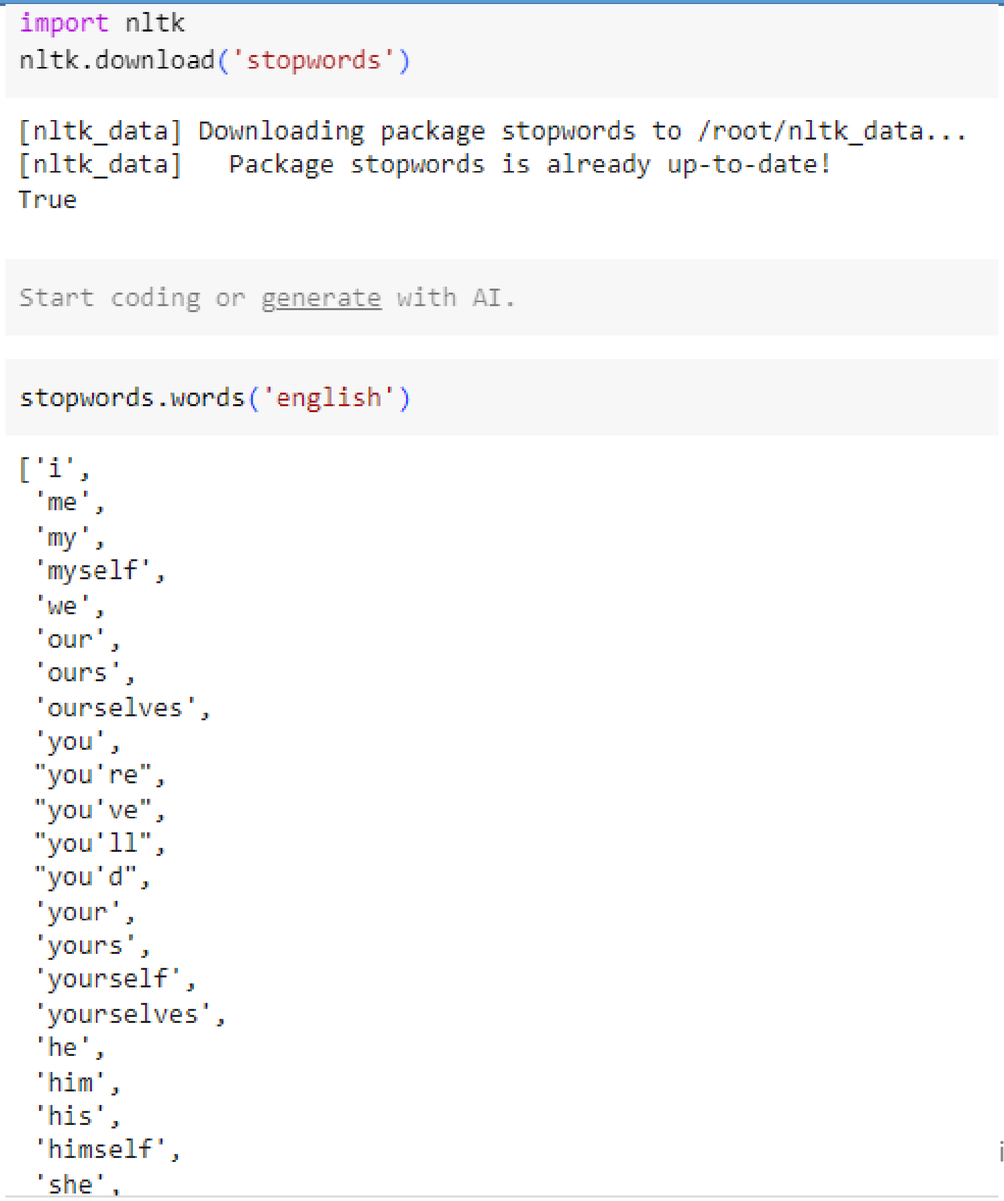
- **Why Remove Stop Words?**
  - To **reduce noise** in the text.
  - To **focus on meaningful words** (like *movie*, *amazing*, *bad*, *performance*).
  - To **improve processing efficiency** in algorithms such as TF–IDF, text classification, and sentiment analysis.

- **Example**
  - **Original sentence:**
    - “The movie was not very interesting.”
  - **After removing stop words:**
    - “movie not interesting”
- Now, the key words that carry meaning remain.

Preparing text data (Stop words)

- **Stop words** are common words in a language that are often removed during Natural Language Processing (NLP) tasks.
- Examples include words like "the," "is," "in," "and," etc.
- Why Remove Stop Words?
  - They **appear frequently** and **do not carry** significant **meaning**.
  - Removing them **helps reduce the dataset size and speeds up processing** without losing valuable information.
- Examples:
  - English: "a," "an," "the," "on," "in“
  - Spanish: "el," "la," "los," "de“
  - Note: **some tasks** (e.g., sentiment analysis), **stop words may be important** and should not be removed.

Preparing text data (Stop words)



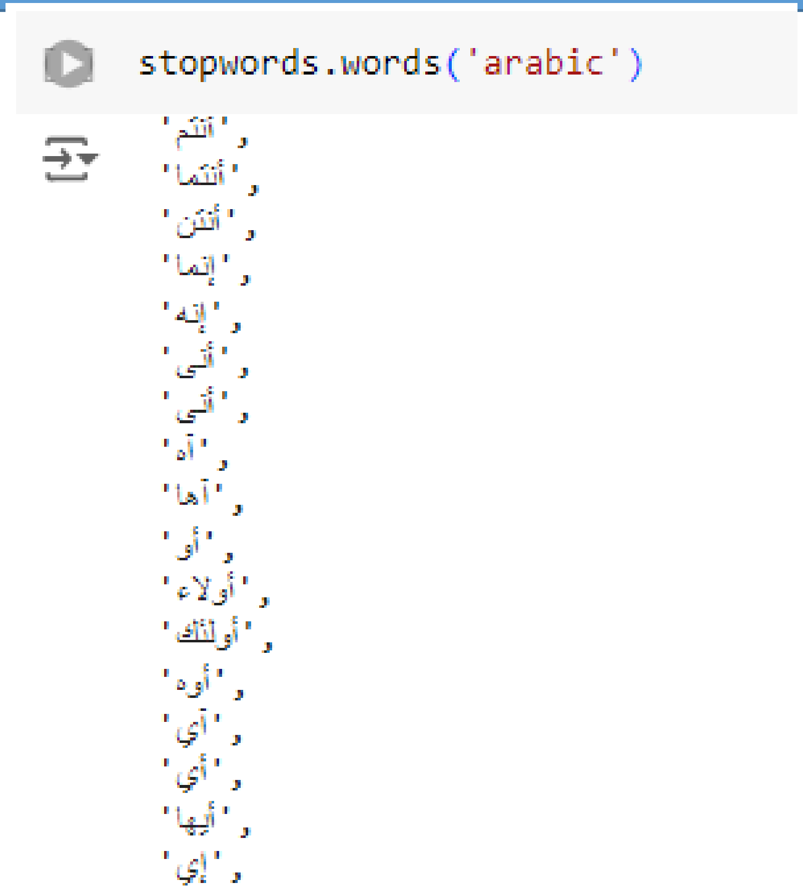
```python
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
Start coding or generate with AI.
```

```python
stopwords.words('english')
```

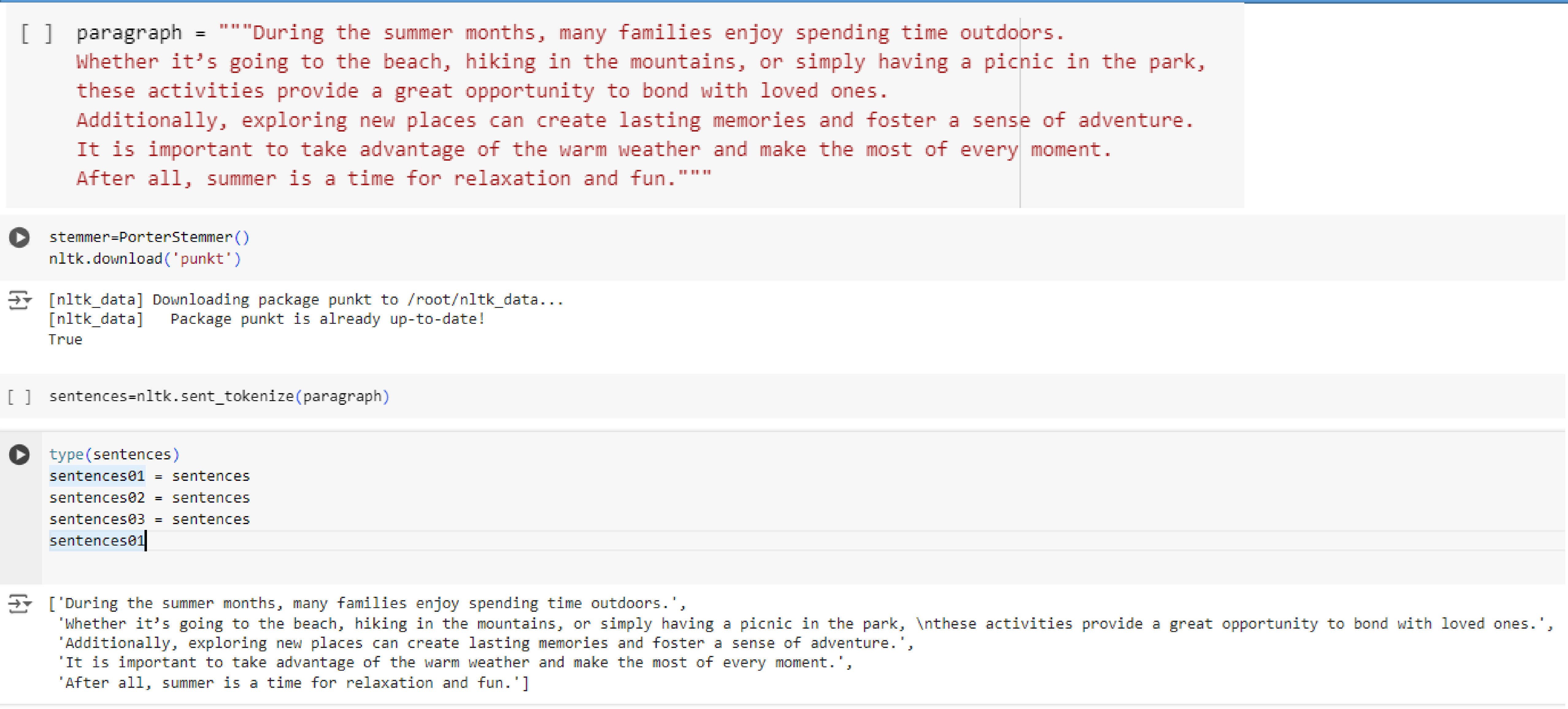
```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
 'him',
 'his',
 'himself',
 'she',
```



```python
stopwords.words('arabic')
```

```
'أنتم',
'أنتما',
'أنتن',
'إنما',
'إنه',
'أنى',
'أنى',
'آه',
'آها',
'أو',
'أولاء',
'أولئك',
'أوه',
'آي',
'أي',
'أيها',
'إي',
```

Preparing text data (Stop words)



```
paragraph = """During the summer months, many families enjoy spending time outdoors.
Whether it’s going to the beach, hiking in the mountains, or simply having a picnic in the park,
these activities provide a great opportunity to bond with loved ones.
Additionally, exploring new places can create lasting memories and foster a sense of adventure.
It is important to take advantage of the warm weather and make the most of every moment.
After all, summer is a time for relaxation and fun."""
```

```
stemmer=PorterStemmer()
nltk.download('punkt')
```

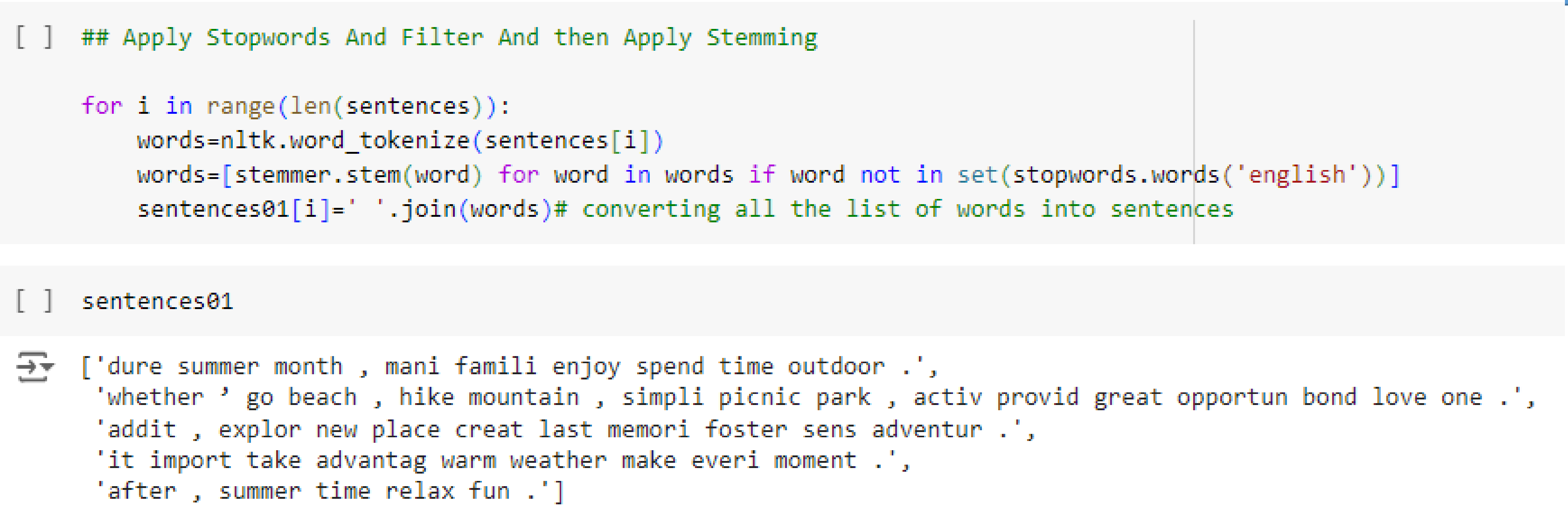
```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
sentences=nltk.sent_tokenize(paragraph)
```

```
type(sentences)
sentences01 = sentences
sentences02 = sentences
sentences03 = sentences
sentences01
```

```
['During the summer months, many families enjoy spending time outdoors.',
 'Whether it’s going to the beach, hiking in the mountains, or simply having a picnic in the park, \nthese activities provide a great opportunity to bond with loved ones.',
 'Additionally, exploring new places can create lasting memories and foster a sense of adventure.',
 'It is important to take advantage of the warm weather and make the most of every moment.',
 'After all, summer is a time for relaxation and fun.']
```

Preparing text data (Stop words)

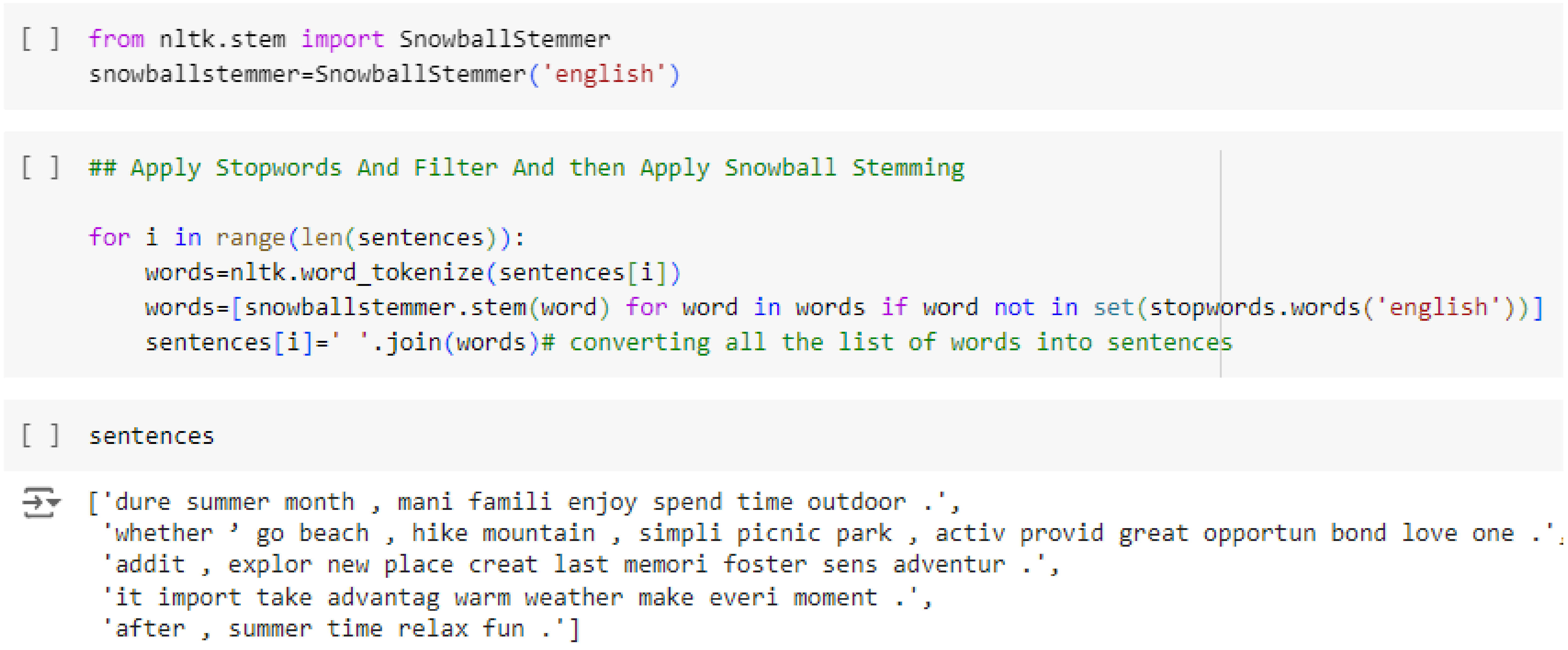


```
[ ]  ## Apply Stopwords And Filter And then Apply Stemming

     for i in range(len(sentences)):
         words=nltk.word_tokenize(sentences[i])
         words=[stemmer.stem(word) for word in words if word not in set(stopwords.words('english'))]
         sentences01[i]=' '.join(words)# converting all the list of words into sentences
```

```
[ ]  sentences01
```

```
['dure summer month , mani famili enjoy spend time outdoor .',
 'whether ’ go beach , hike mountain , simpli picnic park , activ provid great opportun bond love one .',
 'addit , explor new place creat last memori foster sens adventur .',
 'it import take advantag warm weather make everi moment .',
 'after , summer time relax fun .']
```

Preparing text data (Stop words)



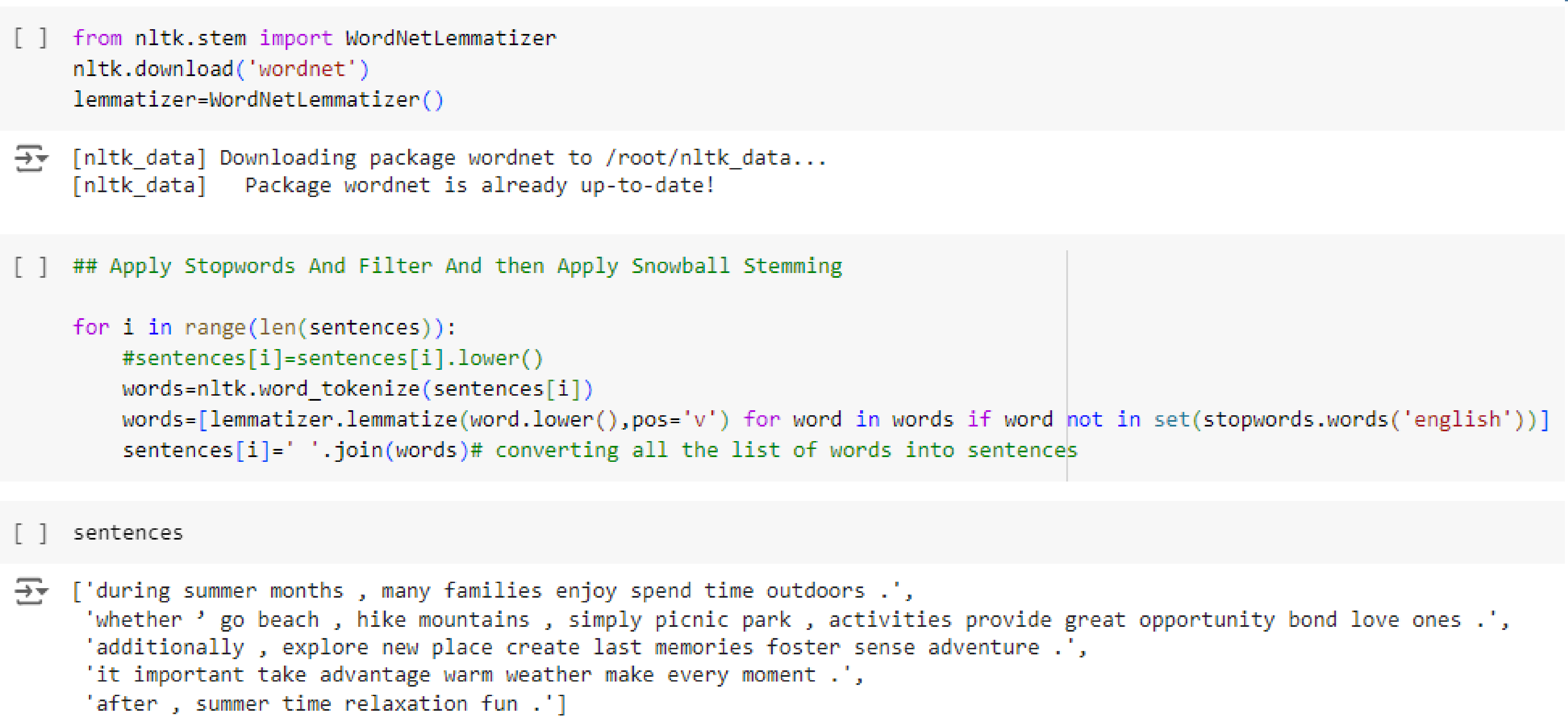
```
[ ]  from nltk.stem import SnowballStemmer
     snowballstemmer=SnowballStemmer('english')
```

```
[ ]  ## Apply Stopwords And Filter And then Apply Snowball Stemming

     for i in range(len(sentences)):
         words=nltk.word_tokenize(sentences[i])
         words=[snowballstemmer.stem(word) for word in words if word not in set(stopwords.words('english'))]
         sentences[i]=' '.join(words)# converting all the list of words into sentences
```

```
[ ]  sentences
```

```
['dure summer month , mani famili enjoy spend time outdoor .',
 'whether ’ go beach , hike mountain , simpli picnic park , activ provid great opportun bond love one .',
 'addit , explor new place creat last memori foster sens adventur .',
 'it import take advantag warm weather make everi moment .',
 'after , summer time relax fun .']
```

Preparing text data (Stop words)



```
[ ]  from nltk.stem import WordNetLemmatizer
     nltk.download('wordnet')
     lemmatizer=WordNetLemmatizer()
```

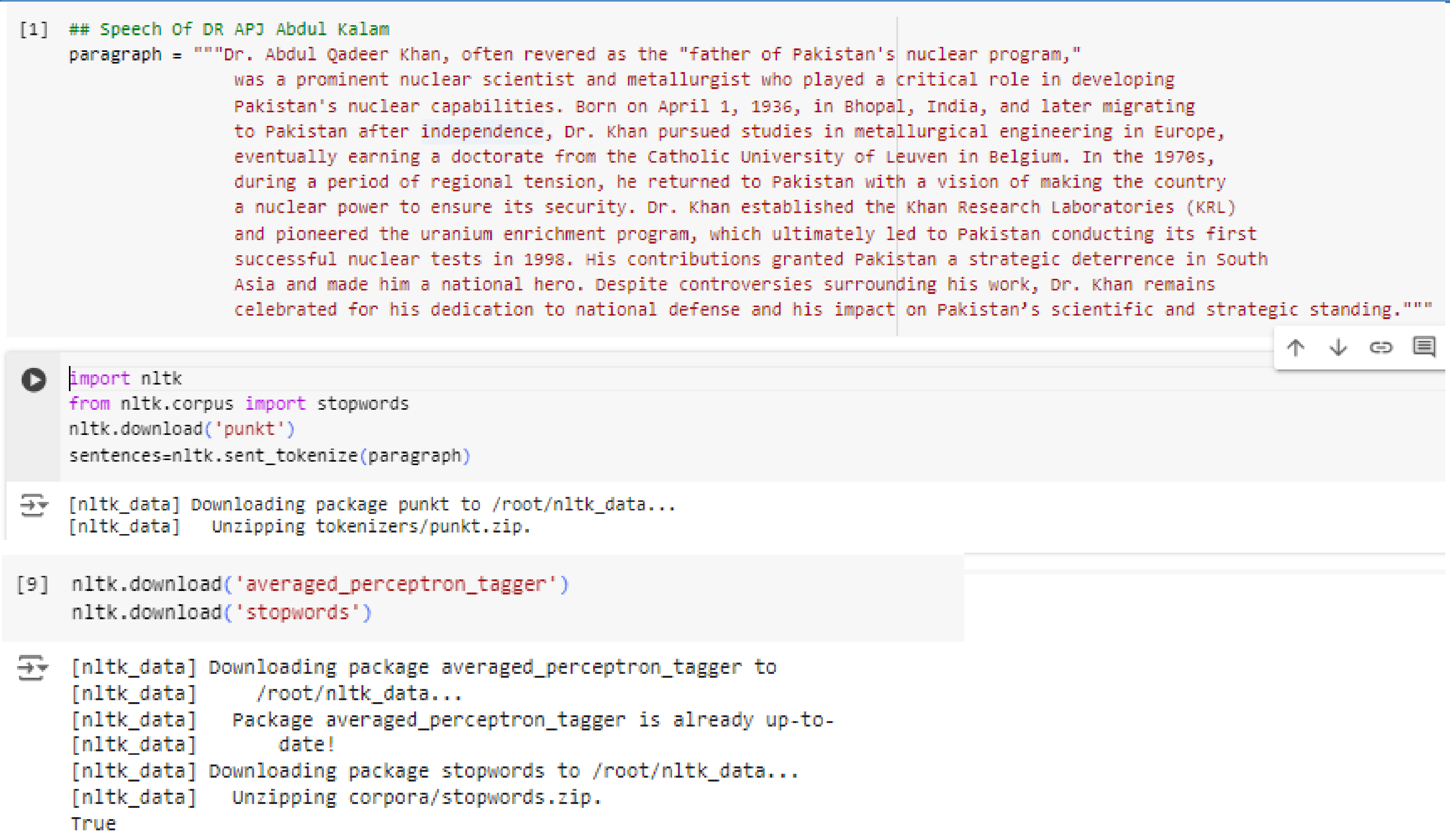
```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
[ ]  ## Apply Stopwords And Filter And then Apply Snowball Stemming

     for i in range(len(sentences)):
         #sentences[i]=sentences[i].lower()
         words=nltk.word_tokenize(sentences[i])
         words=[lemmatizer.lemmatize(word.lower(),pos='v') for word in words if word not in set(stopwords.words('english'))]
         sentences[i]=' '.join(words)# converting all the list of words into sentences
```

```
[ ]  sentences
```

```
['during summer months , many families enjoy spend time outdoors .',
 'whether ’ go beach , hike mountains , simply picnic park , activities provide great opportunity bond love ones .',
 'additionally , explore new place create last memories foster sense adventure .',
 'it important take advantage warm weather make every moment .',
 'after , summer time relaxation fun .']
```

Preparing text data (Parts of speech)



```
[1] ## Speech Of DR APJ Abdul Kalam
    paragraph = """Dr. Abdul Qadeer Khan, often revered as the "father of Pakistan's nuclear program,"
                    was a prominent nuclear scientist and metallurgist who played a critical role in developing
                    Pakistan's nuclear capabilities. Born on April 1, 1936, in Bhopal, India, and later migrating
                    to Pakistan after independence, Dr. Khan pursued studies in metallurgical engineering in Europe,
                    eventually earning a doctorate from the Catholic University of Leuven in Belgium. In the 1970s,
                    during a period of regional tension, he returned to Pakistan with a vision of making the country
                    a nuclear power to ensure its security. Dr. Khan established the Khan Research Laboratories (KRL)
                    and pioneered the uranium enrichment program, which ultimately led to Pakistan conducting its first
                    successful nuclear tests in 1998. His contributions granted Pakistan a strategic deterrence in South
                    Asia and made him a national hero. Despite controversies surrounding his work, Dr. Khan remains
                    celebrated for his dedication to national defense and his impact on Pakistan's scientific and strategic standing."""
```

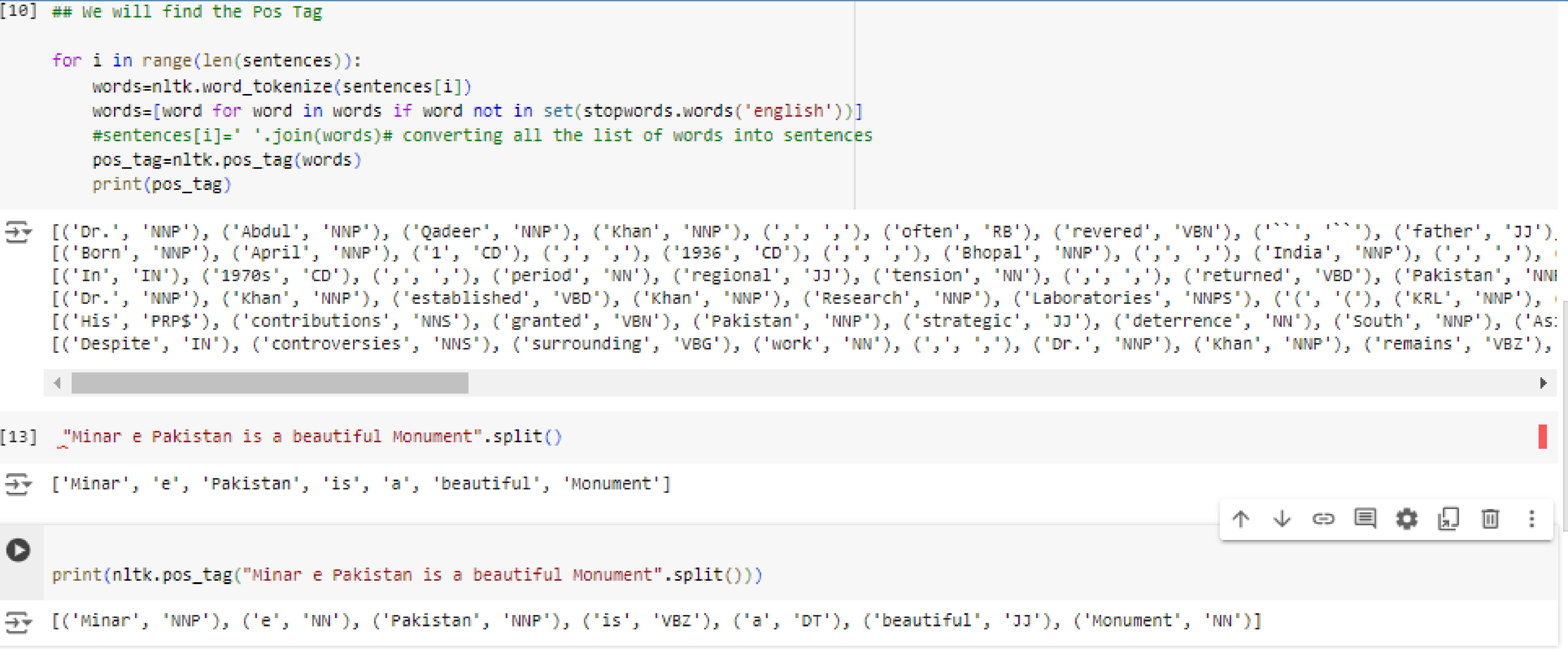
```
import nltk
from nltk.corpus import stopwords
nltk.download('punkt')
sentences=nltk.sent_tokenize(paragraph)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

```
[9] nltk.download('averaged_perceptron_tagger')
    nltk.download('stopwords')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

Preparing text data (Parts of speech)



```
[10] ## We will find the Pos Tag

     for i in range(len(sentences)):
         words=nltk.word_tokenize(sentences[i])
         words=[word for word in words if word not in set(stopwords.words('english'))]
         #sentences[i]=' '.join(words)# converting all the list of words into sentences
         pos_tag=nltk.pos_tag(words)
         print(pos_tag)
```

```
[('Dr.', 'NNP'), ('Abdul', 'NNP'), ('Qadeer', 'NNP'), ('Khan', 'NNP'), (',', ','), ('often', 'RB'), ('revered', 'VBN'), ('``', '``'), ('father', 'JJ').
[('Born', 'NNP'), ('April', 'NNP'), ('1', 'CD'), (',', ','), ('1936', 'CD'), (',', ','), ('Bhopal', 'NNP'), (',', ','), ('India', 'NNP'), (',', ','),
[('In', 'IN'), ('1970s', 'CD'), (',', ','), ('period', 'NN'), ('regional', 'JJ'), ('tension', 'NN'), (',', ','), ('returned', 'VBD'), ('Pakistan', 'NNI
[('Dr.', 'NNP'), ('Khan', 'NNP'), ('established', 'VBD'), ('Khan', 'NNP'), ('Research', 'NNP'), ('Laboratories', 'NNPS'), ('(', '('), ('KRL', 'NNP'),
[('His', 'PRP$'), ('contributions', 'NNS'), ('granted', 'VBN'), ('Pakistan', 'NNP'), ('strategic', 'JJ'), ('deterrence', 'NN'), ('South', 'NNP'), ('As:
[('Despite', 'IN'), ('controversies', 'NNS'), ('surrounding', 'VBG'), ('work', 'NN'), (',', ','), ('Dr.', 'NNP'), ('Khan', 'NNP'), ('remains', 'VBZ'),
```

```
[13] "Minar e Pakistan is a beautiful Monument".split()
```

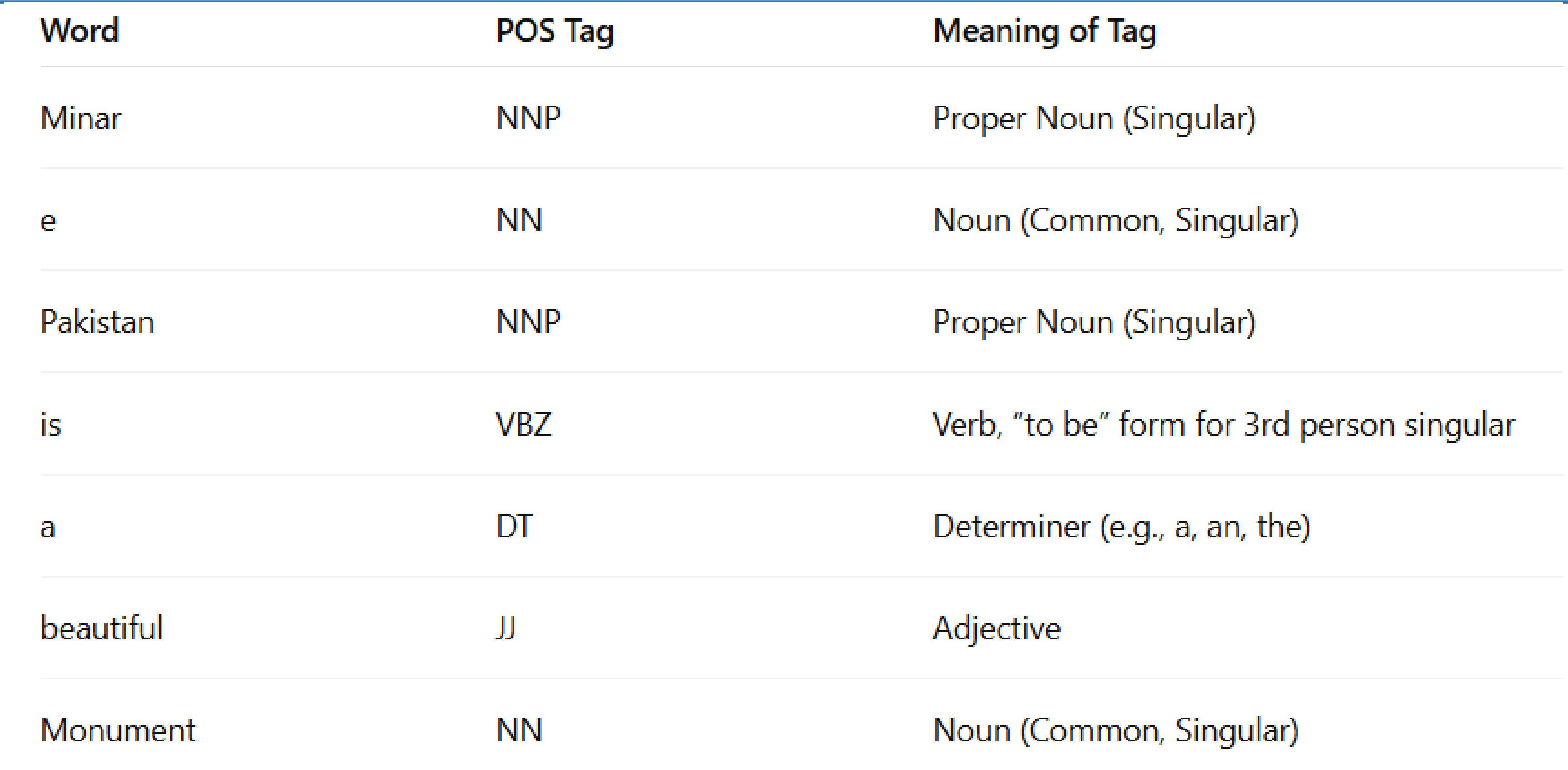
```
['Minar', 'e', 'Pakistan', 'is', 'a', 'beautiful', 'Monument']
```

```
print(nltk.pos_tag("Minar e Pakistan is a beautiful Monument".split()))
```

```
[('Minar', 'NNP'), ('e', 'NN'), ('Pakistan', 'NNP'), ('is', 'VBZ'), ('a', 'DT'), ('beautiful', 'JJ'), ('Monument', 'NN')]
```

SUKKUR IBA UNIVERSITY COMPUTER SCIENCE

Preparing text data (Parts of speech)

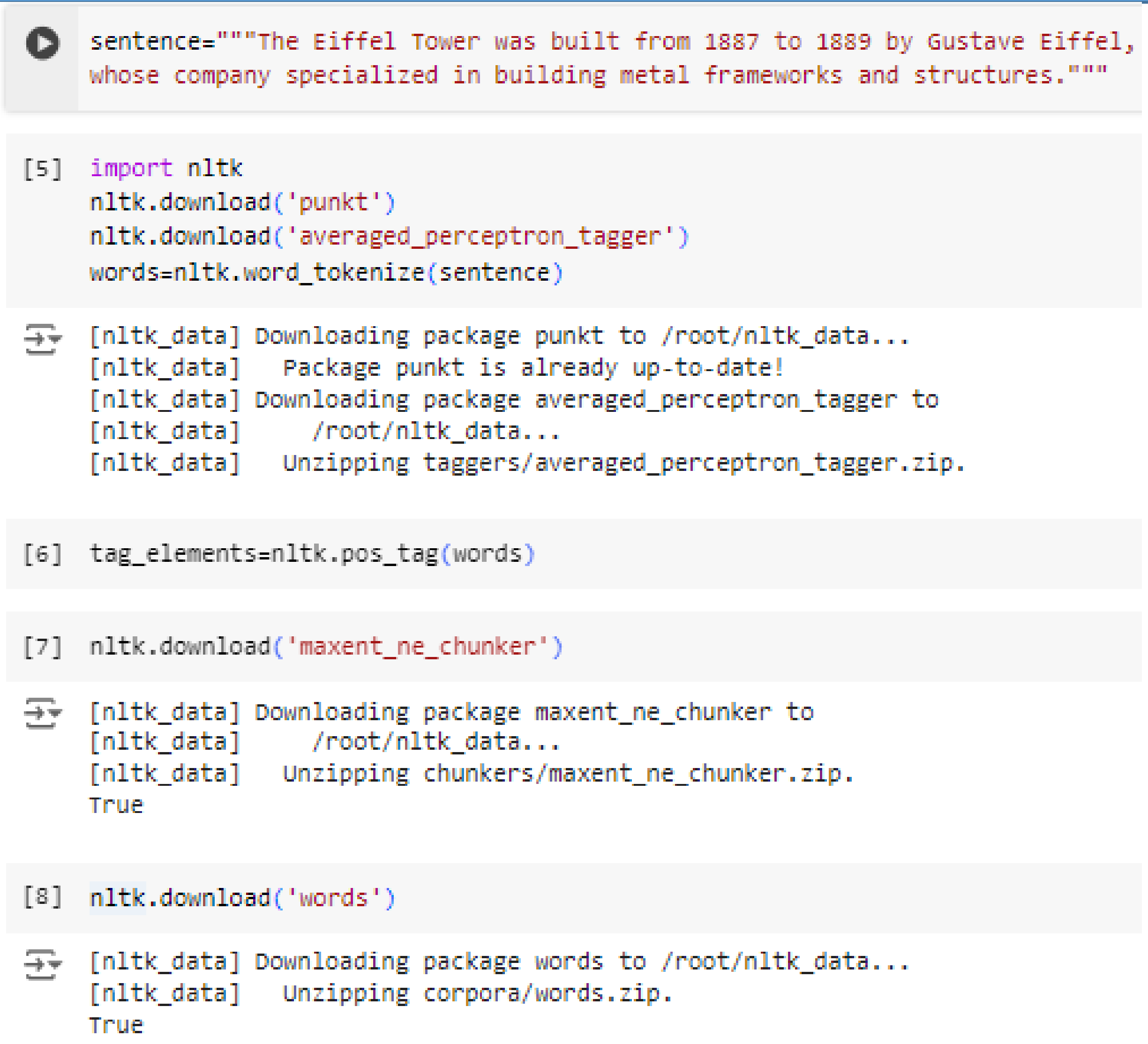


Word	POS Tag	Meaning of Tag
Minar	NNP	Proper Noun (Singular)
e	NN	Noun (Common, Singular)
Pakistan	NNP	Proper Noun (Singular)
is	VBZ	Verb, "to be" form for 3rd person singular
a	DT	Determiner (e.g., a, an, the)
beautiful	JJ	Adjective
Monument	NN	Noun (Common, Singular)

Tokenization	Breaking text into smaller units (tokens).
POS Tagging	Labeling each word with its grammatical role (noun, verb, adjective, etc.).
nltk.pos_tag()	A function that uses statistical models to automatically assign POS tags to words.

Preparing text data (Name entity recognition)

- Named Entity Recognition (NER) is an NLP technique that identifies and classifies named entities in text, such as people, locations, organizations, dates, and more.
- Helps in understanding the context of text by isolating key entities.



```
sentence="""The Eiffel Tower was built from 1887 to 1889 by Gustave Eiffel,
whose company specialized in building metal frameworks and structures."""
```

```
[5] import nltk
    nltk.download('punkt')
    nltk.download('averaged_perceptron_tagger')
    words=nltk.word_tokenize(sentence)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
```

```
[6] tag_elements=nltk.pos_tag(words)
```

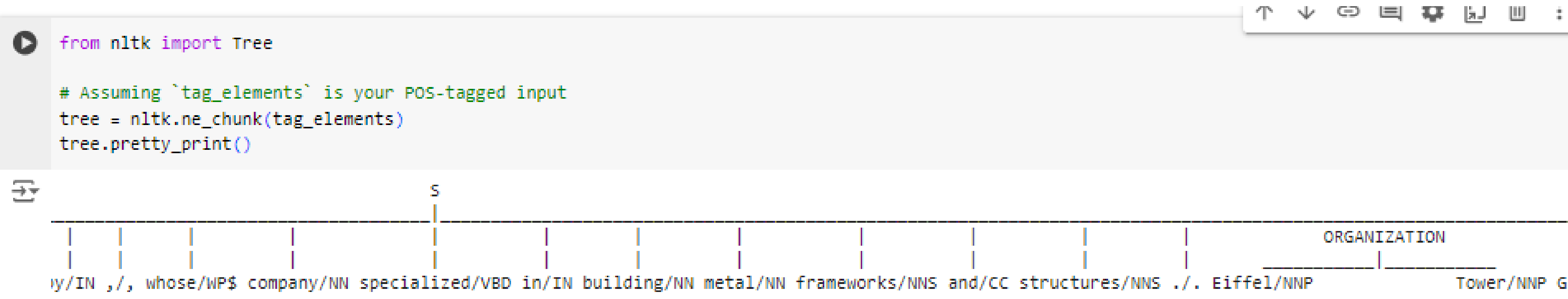
```
[7] nltk.download('maxent_ne_chunker')
```

```
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping chunkers/maxent_ne_chunker.zip.
True
```

```
[8] nltk.download('words')
```

```
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Unzipping corpora/words.zip.
True
```

Preparing text data (Name entity recognition)



```python
from nltk import Tree

# Assuming `tag_elements` is your POS-tagged input
tree = nltk.ne_chunk(tag_elements)
tree.pretty_print()
```

```
                                     S
_____________________________________|_________________________________________________________________________________________________________________
  |   |      |          |            |          |        |          |           |          |          |        |              ORGANIZATION
  |   |      |          |            |          |        |          |           |          |          |        |        ___________|___________
y/IN ,/, whose/WP$ company/NN specialized/VBD in/IN building/NN metal/NN frameworks/NNS and/CC structures/NNS ./. Eiffel/NNP              Tower/NNP G
```

Vectorization

- Vectorization is the process of converting **text data into numerical** format so that ML models can process it.
- Essential for transforming words and sentences into **a form** that **algorithms can understand**..
- Types of vectorization
    - One hot encoder
    - Bag of Words (BoW)
    - TF-IDF
    - Word2Vec
    - AvgWord2Vec

Vectorization(One hot encoding)

- **One-hot encoding** is a technique to represent words or tokens in text as binary vectors.
- Each unique word is assigned a vector, with all elements set to 0 except one element, which is set to 1 (indicating the word’s unique position in the vocabulary).
- It converts text data into numerical format for ML models.
- Suitable for small vocabularies and simple NLP tasks.
- Example Vocabulary: ["apple," "banana," "grape," "orange"]
  - Vector Representation:
  - apple: [1, 0, 0, 0]
  - banana: [0, 1, 0, 0]
  - grape: [0, 0, 1, 0]
  - orange: [0, 0, 0, 1]

One-hot encoding

- How Does One-Hot Encoding Working
- Steps:
  - Build Vocabulary: Create a list of all unique words in the text.
  - Assign Indexes: Assign each word an index in the vocabulary.
  - Create Binary Vectors: For each word, create a vector of length equal to the vocabulary size, with a 1 at the index of the word and 0s elsewhere.
  - Example Sentence: "I like apples“
  - Vocabulary: ["I", "like", "apples"]
  - One-Hot Encoded Vectors:
  - "I" = [1, 0, 0]
  - "like" = [0, 1, 0]
  - "apples" = [0, 0, 1]

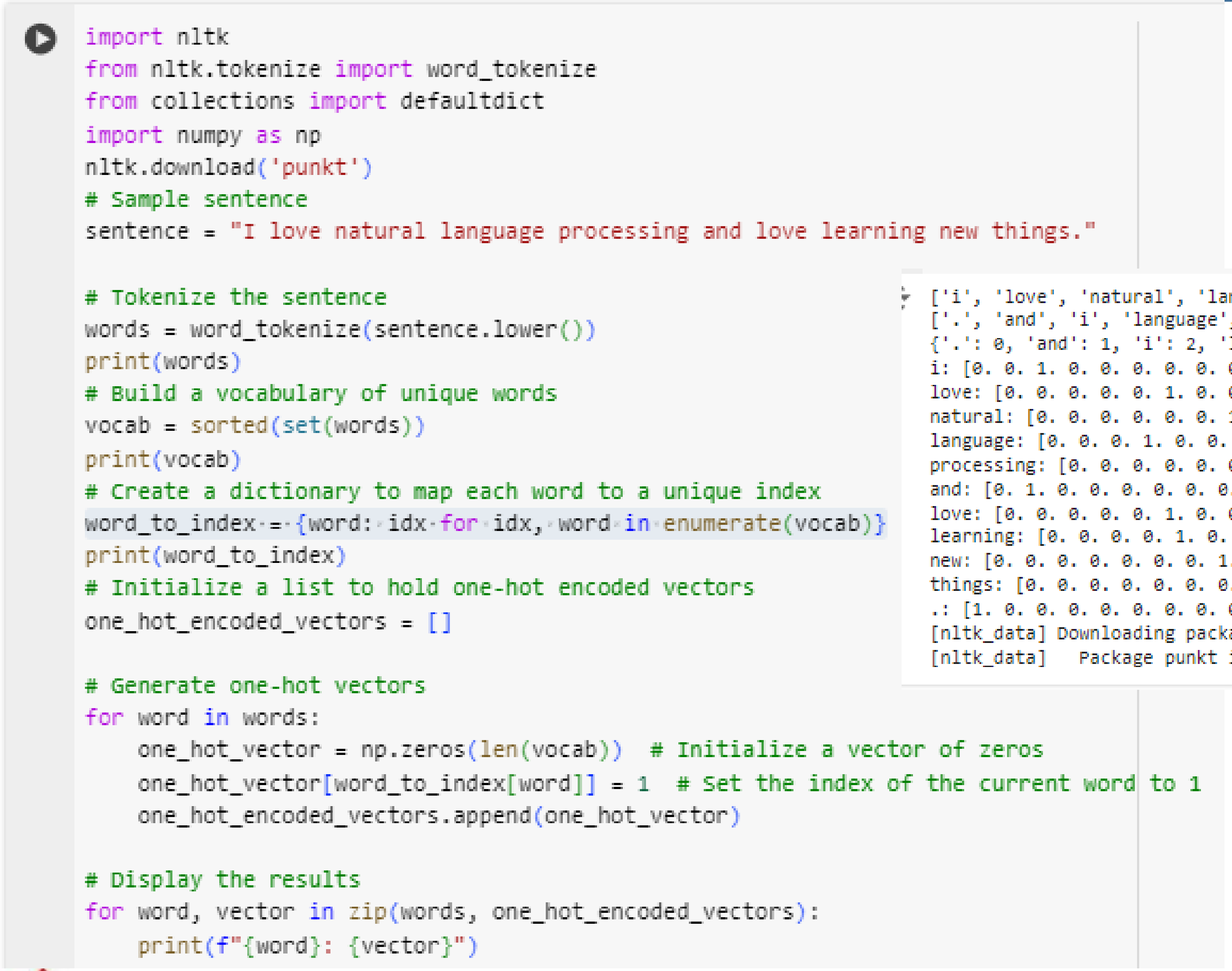
One hot encoding

- **Limitations**:
  - High Dimensionality: For large vocabularies, the vector size becomes very large, leading to memory inefficiency.
  - No Semantic Information: One-hot vectors don't capture relationships between words (e.g., with vocab love, processing, and natural. What is the distance between them in a document?).
  - ML algorithms need fixed input that one-hot might not provide?
    - Variable Length Input: ML models expect inputs to be the same size.
    - However, with sentences of varying lengths, one-hot encoding will lead to matrices with different dimensions (e.g., 7×47×4 for a 4-word sentence vs. 7×57×5 for a 5-word sentence).
    - Padding or truncating sentences to the same length can mitigate this issue, but it still doesn’t address the high dimensionality and sparsity.

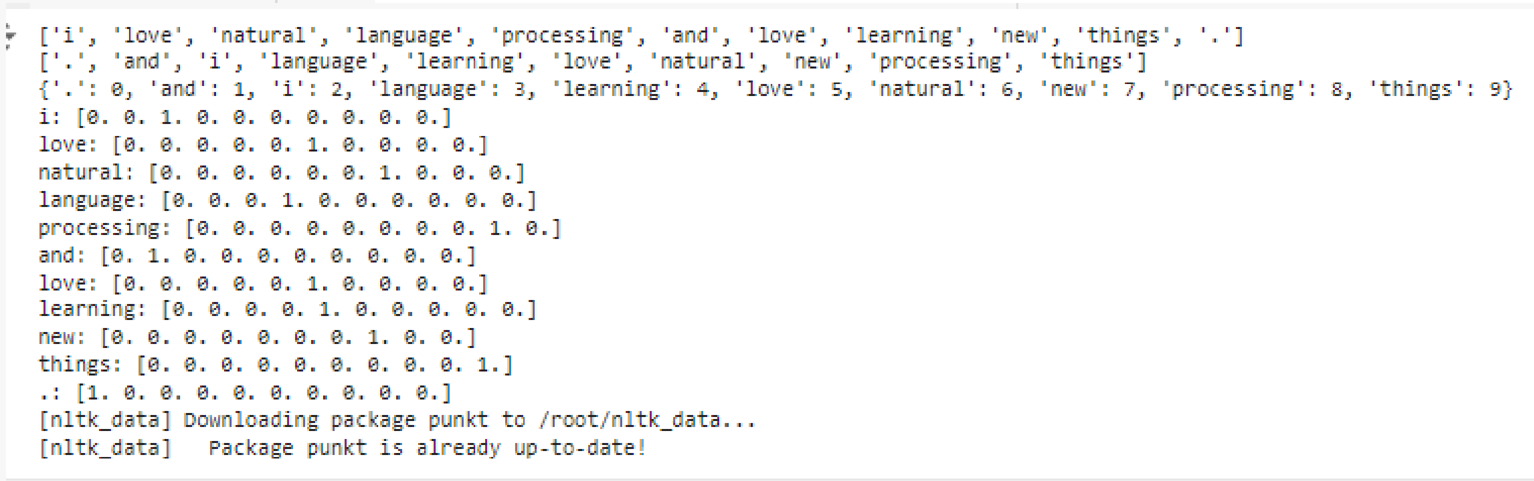
One hot encoding

- **Limitations**:
  - Out of vocabulary words
  - Real-World Example: Text: "I love natural language processing“
    - Vocabulary: ["I", "love", "natural", "language", "processing"]
    - One-Hot Encoded Vectors:
    - "I" = [1, 0, 0, 0, 0]“
    - love" = [0, 1, 0, 0, 0]
    - "natural" = [0, 0, 1, 0, 0]
    - "language" = [0, 0, 0, 1, 0]
    - "processing" = [0, 0, 0, 0, 1]
    - Final output = [[1, 0, 0, 0, 0], [0, 1, 0, 0, 0], [0, 0, 1, 0, 0], [0, 0, 0, 1, 0], [0, 0, 0, 0, 1]] (5x5)
    - For document= “I love natural language” = [[1, 0, 0, 0, 0], [0, 1, 0, 0, 0], [0, 0, 1, 0, 0], [0, 0, 0, 1, 0]] (5x4)

One hot encoding

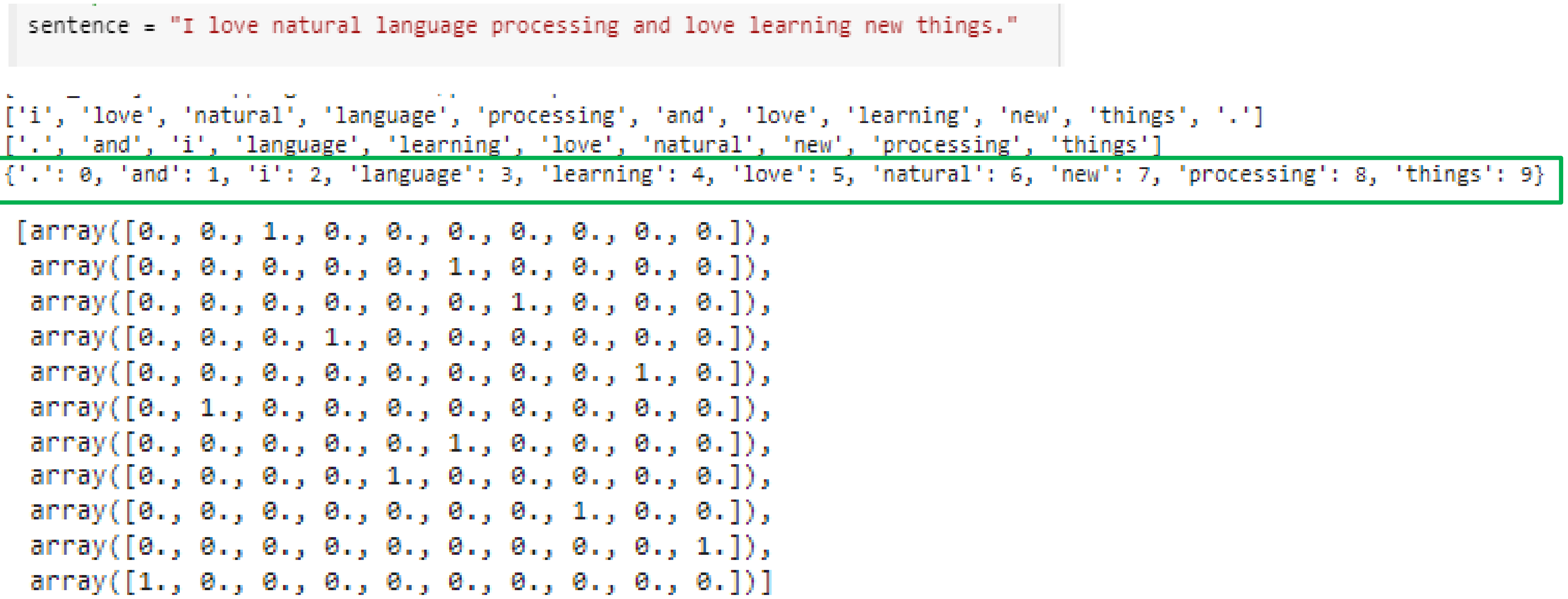


```python
import nltk
from nltk.tokenize import word_tokenize
from collections import defaultdict
import numpy as np
nltk.download('punkt')
# Sample sentence
sentence = "I love natural language processing and love learning new things."

# Tokenize the sentence
words = word_tokenize(sentence.lower())
print(words)
# Build a vocabulary of unique words
vocab = sorted(set(words))
print(vocab)
# Create a dictionary to map each word to a unique index
word_to_index = {word: idx for idx, word in enumerate(vocab)}
print(word_to_index)
# Initialize a list to hold one-hot encoded vectors
one_hot_encoded_vectors = []

# Generate one-hot vectors
for word in words:
    one_hot_vector = np.zeros(len(vocab))  # Initialize a vector of zeros
    one_hot_vector[word_to_index[word]] = 1  # Set the index of the current word to 1
    one_hot_encoded_vectors.append(one_hot_vector)

# Display the results
for word, vector in zip(words, one_hot_encoded_vectors):
    print(f"{word}: {vector}")
```



```
['i', 'love', 'natural', 'language', 'processing', 'and', 'love', 'learning', 'new', 'things', '.']
['.', 'and', 'i', 'language', 'learning', 'love', 'natural', 'new', 'processing', 'things']
{'.': 0, 'and': 1, 'i': 2, 'language': 3, 'learning': 4, 'love': 5, 'natural': 6, 'new': 7, 'processing': 8, 'things': 9}
i: [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
love: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
natural: [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
language: [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
processing: [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
and: [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
love: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
learning: [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
new: [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
things: [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
.: [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

One hot encoding



```
sentence = "I love natural language processing and love learning new things."
```

```
['i', 'love', 'natural', 'language', 'processing', 'and', 'love', 'learning', 'new', 'things', '.']
['.', 'and', 'i', 'language', 'learning', 'love', 'natural', 'new', 'processing', 'things']
{'.': 0, 'and': 1, 'i': 2, 'language': 3, 'learning': 4, 'love': 5, 'natural': 6, 'new': 7, 'processing': 8, 'things': 9}
```

```
[array([0., 0., 1., 0., 0., 0., 0., 0., 0., 0.]),
 array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.]),
 array([0., 0., 0., 0., 0., 0., 1., 0., 0., 0.]),
 array([0., 0., 0., 1., 0., 0., 0., 0., 0., 0.]),
 array([0., 0., 0., 0., 0., 0., 0., 0., 1., 0.]),
 array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.]),
 array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0.]),
 array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0.]),
 array([0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]),
 array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0.])]
```

SUKKUR IBA UNIVERSITY COMPUTER SCIENCE

One hot encoding

- document1 = “To become world class university is the vision of Sukkur IBA”
- document2 = “To become top university in south asia is the vision ”
- Vocab = ?
- One hot encoding for both document1 and document2

Vectorization (Bag of Words)

- **Bag of Words (BoW)** is a common NLP technique for text representation that converts words into numerical format.
- Represents text by **counting word occurrences** while **ignoring grammar** and word **order**.
- **Transforms** unstructured text **into structured numerical data** that models can analyze.
- Example Vocabulary:
  - Sentence: "I love NLP, and NLP loves me.“
  - Vocabulary: {"I", "love", "NLP", "and", "loves", "me“}
- Each unique word from the vocabulary is represented as a column in a matrix.

Bag of Words (BoW)

- Working of BoW of Words (BoW):
- Step:
  - **Tokenize:** Split text into individual words or tokens.
  - **Create Vocabulary**: Build a list of all unique words in the corpus.
  - **Count Occurrences**: For each sentence or document, count how often each word in the vocabulary appears.
- Example Matrix:
  - Document 1: "I love NLP"
  - Document 2: "NLP loves me"

BoW Matrix:

	I	love	NLP	and	loves	me
Doc1	1	1	1	0	0	0
Doc2	0	0	1	0	1	1

Bag of Words (BoW)

- Example Matrix:
  - Document 1: "I love NLP and NLP is good"
  - Document 2: "IBA loves me and NLP
  - Vocab: love, NLP, good, IBA, me

Binary BoW Matrix:

	NLP	good	IBA	me	love
Doc1	1	1	1	0	1
Doc2	0	0	1	0	1

Normal BoW Matrix:

	NLP	good	IBA	me	love
Doc1	2	1	0	0	1
Doc2	1	0	1	0	1

Bag of Words Model explanation

Corpus

A collection of text documents

↓

Tokenize

Divide the text into smaller units called tokens, usually words or phrases

↓

Count word frequencies

Create a vocabulary of unique words and count the number of times each word appears in each document.

↓

Encode the data

Encoding the text data as numerical values by creating a vector for each document, with each element of the vector representing the frequency count of a particular word in the document

Example

Corpus:

The dog is happy. The child makes the dog happy. The dog makes the child happy

↓

Tokenization:

D1: [The] [dog] [is] [happy]
D2: [The] [child] [makes] [the] [dog] [happy]
D3: [The] [dog] [makes] [the] [child] [happy]

↓

Documents	Counting word frequencies
D1	the: 1, dog: 1, is: 1, happy: 1
D2	the: 2, dog: 1, makes: 1, child: 1, happy: 1
D3	the: 2, child: 1, makes: 1, dog: 1, happy: 1

↓

Encode	child	dog	happy	is	makes	the	BoW Vector representations
D1	0	1	1	1	0	1	[0,1,1,1,0,1]
D2	1	1	1	0	1	2	[1,1,1,0,1,2]
D3	1	1	1	0	1	2	[1,1,1,0,1,2]

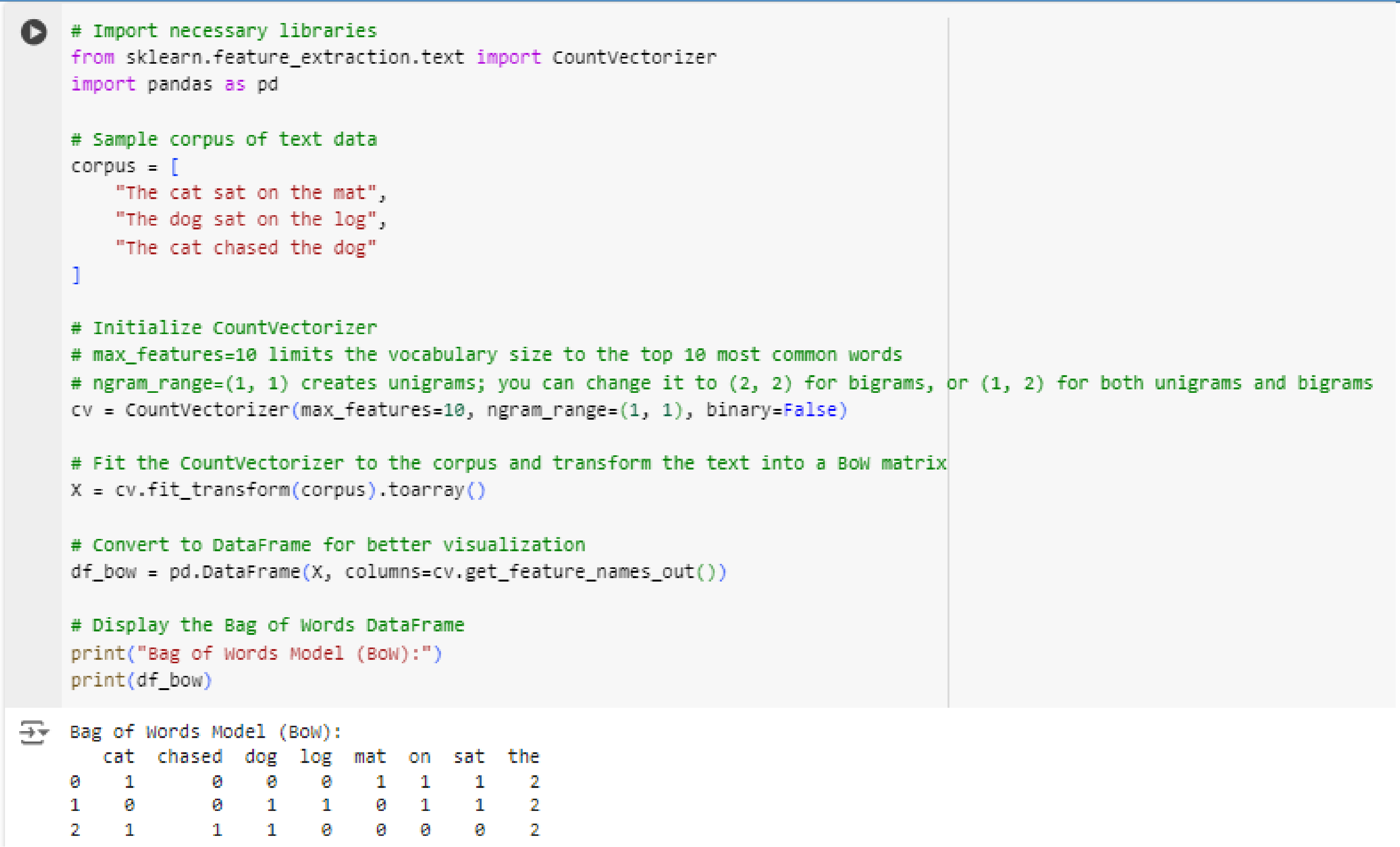
Bag of Words (BoW)

- **Advantages:**
  - Simple to Implement: Quick and easy to set up, even with basic text preprocessing.
  - Fixed size input - > good for ML algorithm
- **Limitations:**
  - Sparse matrix and can lead to overfitting
  - High Dimensionality: As vocabulary size grows, the matrix can become large and sparse.
  - Ignores Context: BoW does not capture word order or semantic meaning.
  - No Synonym Recognition: "Happy" and "joyful" are treated as completely different words.
  - Out of vocabulary problem persist
- **Usage:**
  - Common in text classification, sentiment analysis, and information retrieval where contextual meaning is not crucial.

Unigrams, Bigrams & N-grams

- **Unigrams**
  - They are single words in a sequence.
  - Example: For the sentence "I love NLP", the unigrams are: ["I", "love", "NLP"]
  - Captures individual word frequency but lacks information on word sequence.
- bigrams
  - They are Pairs of two consecutive words.
  - Example: "I love NLP" → ["I love", "love NLP"].
  - Captures word pair relationships, useful for phrase detection and simple context-based analysis.
- **N-grams:**
  - Choosing N (1, 2, 3, …)

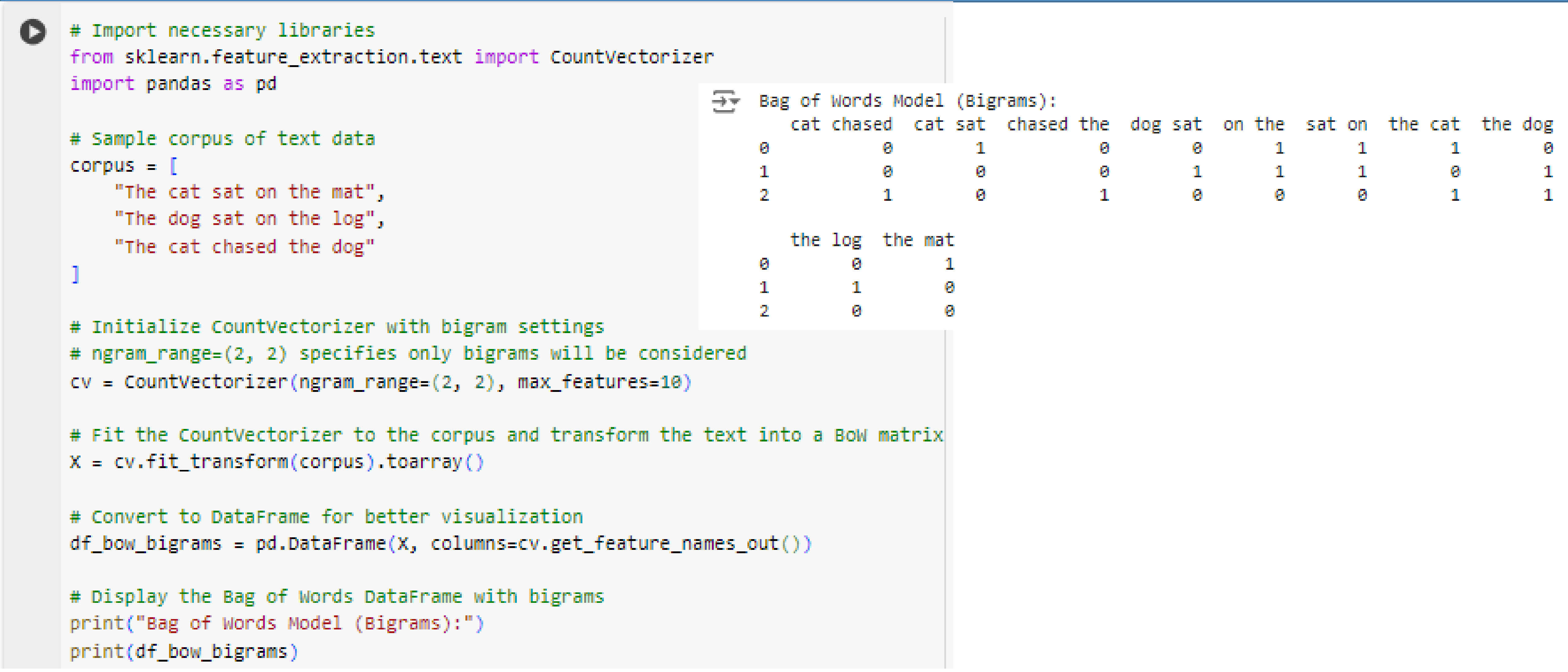
Bag of Words (BoW)



```
# Import necessary libraries
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd

# Sample corpus of text data
corpus = [
    "The cat sat on the mat",
    "The dog sat on the log",
    "The cat chased the dog"
]

# Initialize CountVectorizer
# max_features=10 limits the vocabulary size to the top 10 most common words
# ngram_range=(1, 1) creates unigrams; you can change it to (2, 2) for bigrams, or (1, 2) for both unigrams and bigrams
cv = CountVectorizer(max_features=10, ngram_range=(1, 1), binary=False)

# Fit the CountVectorizer to the corpus and transform the text into a BoW matrix
X = cv.fit_transform(corpus).toarray()

# Convert to DataFrame for better visualization
df_bow = pd.DataFrame(X, columns=cv.get_feature_names_out())

# Display the Bag of Words DataFrame
print("Bag of Words Model (BoW):")
print(df_bow)
```

```
Bag of Words Model (BoW):
   cat  chased  dog  log  mat  on  sat  the
0    1       0    0    0    1   1    1    2
1    0       0    1    1    0   1    1    2
2    1       1    1    0    0   0    0    2
```

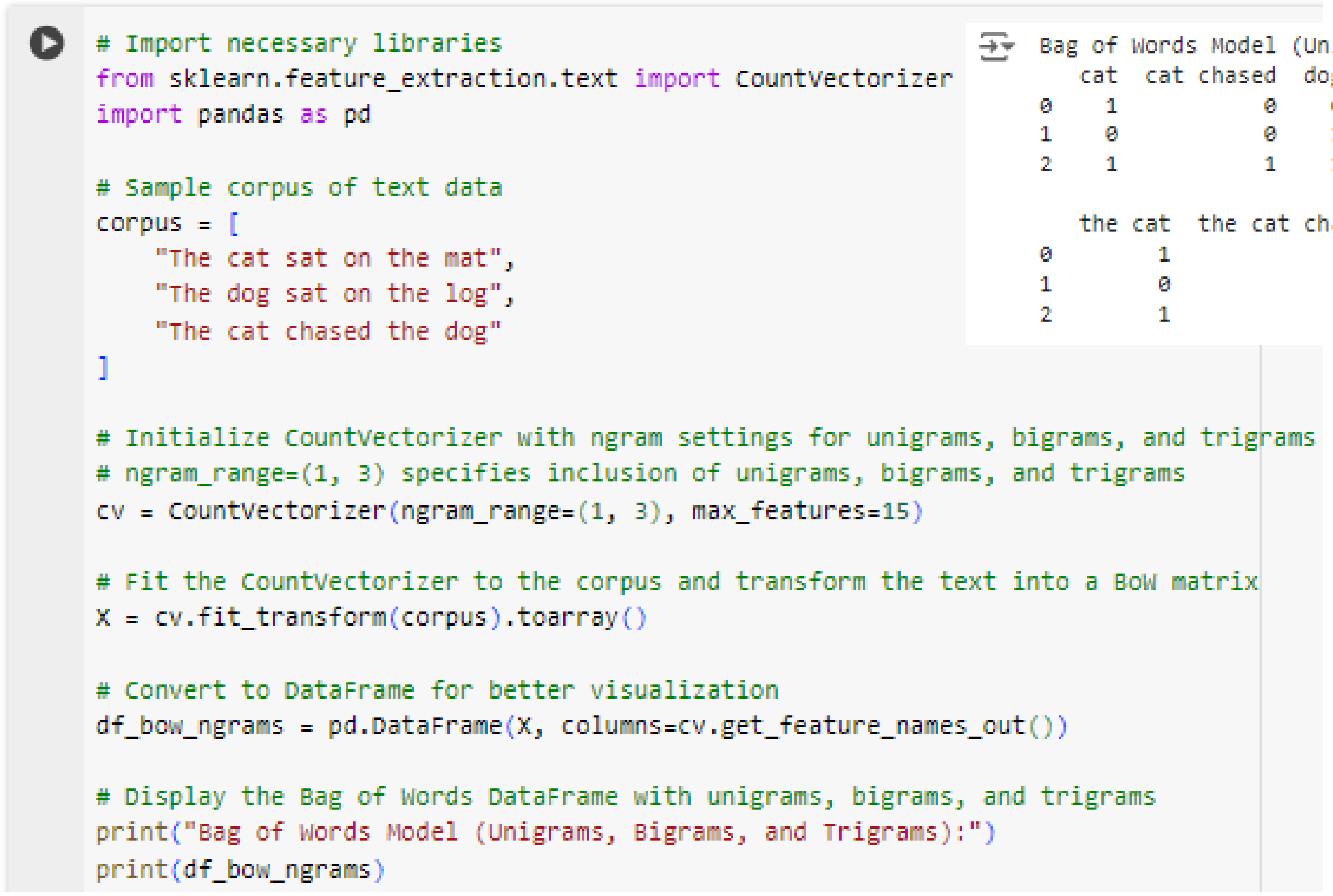
Bag of Words (BoW)



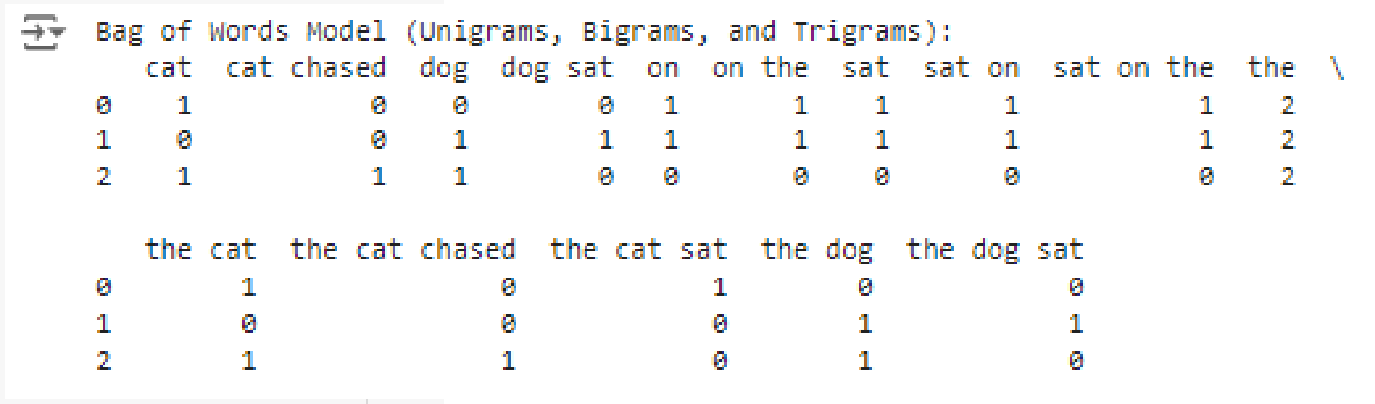
```python
# Import necessary libraries
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd

# Sample corpus of text data
corpus = [
    "The cat sat on the mat",
    "The dog sat on the log",
    "The cat chased the dog"
]

# Initialize CountVectorizer with bigram settings
# ngram_range=(2, 2) specifies only bigrams will be considered
cv = CountVectorizer(ngram_range=(2, 2), max_features=10)

# Fit the CountVectorizer to the corpus and transform the text into a BoW matrix
X = cv.fit_transform(corpus).toarray()

# Convert to DataFrame for better visualization
df_bow_bigrams = pd.DataFrame(X, columns=cv.get_feature_names_out())

# Display the Bag of Words DataFrame with bigrams
print("Bag of Words Model (Bigrams):")
print(df_bow_bigrams)
```

```
Bag of Words Model (Bigrams):
   cat chased  cat sat  chased the  dog sat  on the  sat on  the cat  the dog  \
0           0        1           0        0       1       1        1        0
1           0        0           0        1       1       1        0        1
2           1        0           1        0       0       0        1        1

   the log  the mat
0        0        1
1        1        0
2        0        0
```

Bag of Words (BoW)



```python
# Import necessary libraries
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd

# Sample corpus of text data
corpus = [
    "The cat sat on the mat",
    "The dog sat on the log",
    "The cat chased the dog"
]

# Initialize CountVectorizer with ngram settings for unigrams, bigrams, and trigrams
# ngram_range=(1, 3) specifies inclusion of unigrams, bigrams, and trigrams
cv = CountVectorizer(ngram_range=(1, 3), max_features=15)

# Fit the CountVectorizer to the corpus and transform the text into a BoW matrix
X = cv.fit_transform(corpus).toarray()

# Convert to DataFrame for better visualization
df_bow_ngrams = pd.DataFrame(X, columns=cv.get_feature_names_out())

# Display the Bag of Words DataFrame with unigrams, bigrams, and trigrams
print("Bag of Words Model (Unigrams, Bigrams, and Trigrams):")
print(df_bow_ngrams)
```



```
Bag of Words Model (Unigrams, Bigrams, and Trigrams):
   cat  cat chased  dog  dog sat  on  on the  sat  sat on  sat on the  the  \
0    1           0    0        0   1       1    1       1           1    2
1    0           0    1        1   1       1    1       1           1    2
2    1           1    1        0   0       0    0       0           0    2

   the cat  the cat chased  the cat sat  the dog  the dog sat
0        1               0            1        0            0
1        0               0            0        1            1
2        1               1            0        1            0
```

Bag of Words (BoW)

- **Advantages:**
  - Simple to Implement: Quick and easy to set up, even with basic text preprocessing.
  - Efficient for Small Corpora: Works well with small datasets and classification tasks.
- **Limitations:**
  - High Dimensionality: As vocabulary size grows, the matrix can become large and sparse.
  - Ignores Context: BoW does not capture word order or semantic meaning.
  - No Synonym Recognition: "Happy" and "joyful" are treated as completely different words.
- **Usage:**
  - Common in text classification, sentiment analysis, and information retrieval where contextual meaning is not crucial.

TF (Term Frequency)-IDF (Inverse Document Frequency)

- TF-IDF is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (corpus).
- Helps identify words that are important within specific documents, reducing the impact of common words across the entire dataset.
- Unlike simple word counts, TF-IDF accounts for the uniqueness of terms across documents, improving the performance of NLP models.

TF-IDF

- Term Frequency (TF):
  - Measures the frequency of a term in a document.
  - A word more present in the sentence, gets more importance.

$$TF = \frac{No\ of\ rep\ of\ word\ in\ sentence}{No\ of\ words\ in\ sentence}$$

- Inverse Document Frequency (IDF):
  - Measures how important a term is across all documents.
  - If a word is present in all sentences, then less importance is given to it.
  - Formula: $IDF = \log_e(\frac{No\ of\ sentences}{No\ of\ sentences\ containing\ the\ word})$

- TF-IDF Score:
  - Computed by multiplying TF and IDF.
  - Formula: $TF - IDF = TF \times IDF$

TF-IDF

- How to calculate TF-IDF:
- Corpus: ["The cat sat on the mat", "The dog sat on the log"]
- Step 1: Compute TF:
  - For "sat" in Document 1: TF = 1/6
- Step 2: Compute IDF:
  - For "sat": Appears in 2 documents, so IDF = log(2/2) = 0
  - For "cat": Appears in 1 document, so IDF = log(2/1) = 0.301
- Step 3: Calculate TF-IDF:
  - Multiply TF by IDF for each term in each document.

TF-IDF

- S1 -> good girl
- S2 -> bad work boy
- S3 -> boy girl good
- Vocab: good, bad, boy, girl, work

Term Frequency			
	S1	S2	S3
good	1/2 = 0.5	0/3 = 0	1/3 = 0.33
bad	0/2 = 0	1/3 = 0.33	0/3 = 0
boy	0/2 = 0	1/3 = 0.33	1/3 = 0.33
girl	½ = 0.5	0/3 = 0	1/3 = 0.33
work	0/2 = 0	1/3 = 0.33	0/3 = 0

$$TF = \frac{No\ of\ rep\ of\ word\ in\ sentence}{No\ of\ words\ in\ sentence}$$

$$IDF = \log_e\left(\frac{No\ of\ sentences}{No\ of\ sentences\ containing\ the\ word}\right)$$

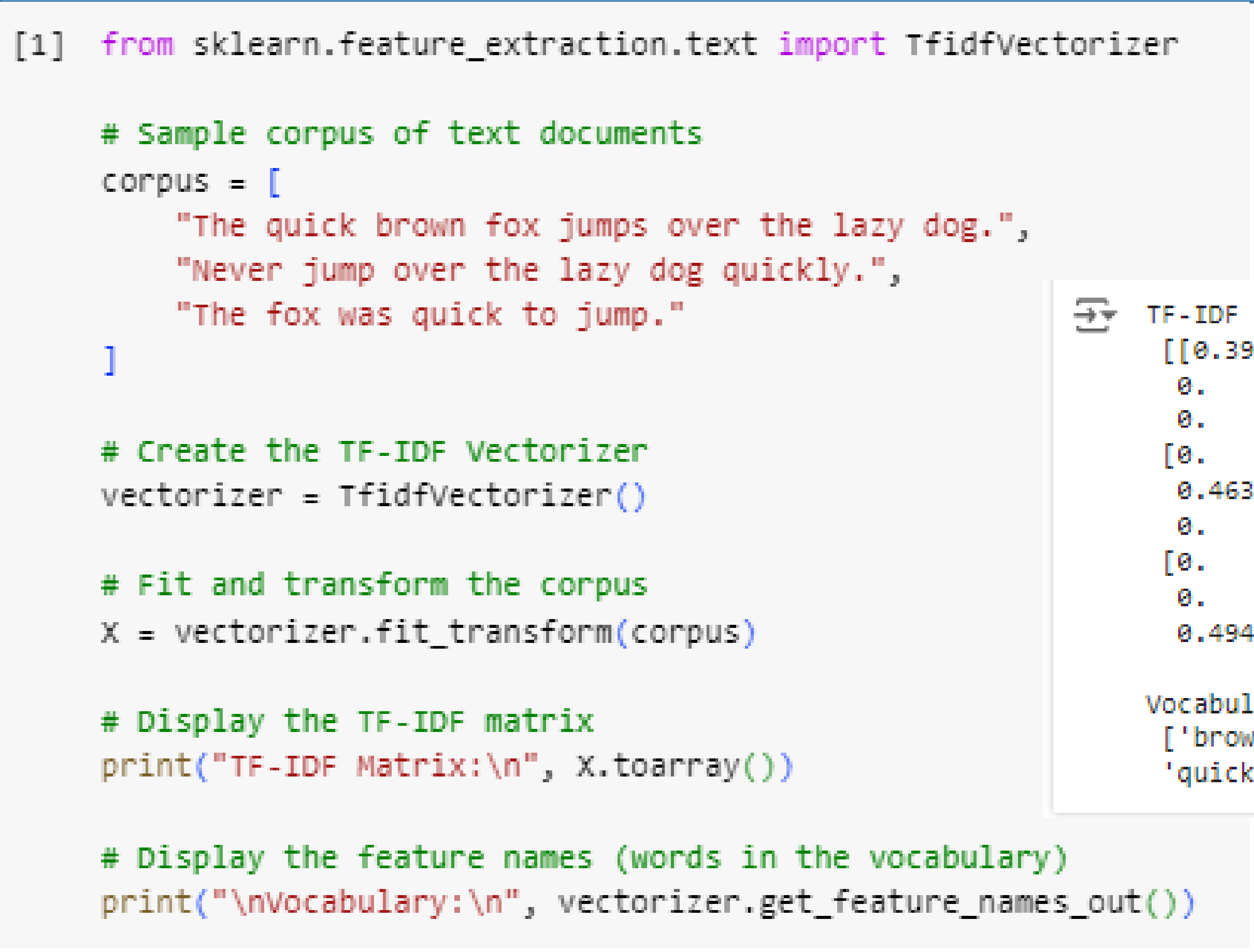
Inverse Document Frequency	
good	ln(3/2) = 0.405
bad	ln(3/1) = 1.099
boy	ln(3/2) = 0.405
girl	ln(3/2) = 0.405
work	ln(3/1) = 1.099

TF-IDF			
	S1	S2	S3
good	0.5x0.405= 0.2	0x0.405 = 0	0.33x0.405= 0
Bad	0x1.099= 0	0.33x1.099=0.36	0x1.099= 0
Boy	0x0.405= 0	0.33x0.405=0.13	0.33x0.405=0.13
Girl	0.5x0.405= 0.20	0x0.405= 0	0.33x0.405=0.13
work	0x1.099= 0	0.33x1.099=0.36	0x1.099= 0

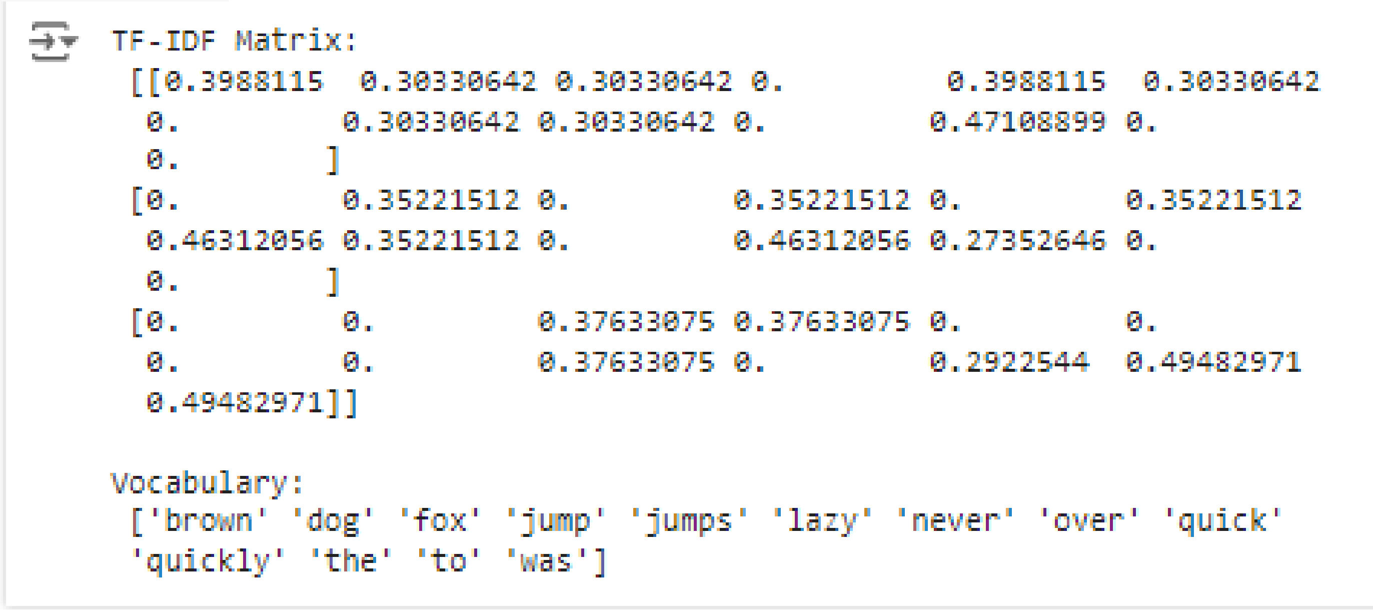
TF-IDF

- Top Features by TF-IDF Score: Prioritizes terms that are both frequent in individual documents and unique across the corpus.
- Using max_df or min_df Parameters: These parameters in Tfidf Vectorizer allow further control over feature selection:
- max_df can remove terms that appear in a high percentage of documents (too common).
- min_df ensures that only terms appearing in a minimum number of documents are included, excluding very rare words.
- Advantages
  - Intuitive
  - Fixed Size - > Vocab size
  - Word importance is being captured
- Disadvantages
  - Sparse matrix still exist
  - Out of vocabulary

TF-IDF



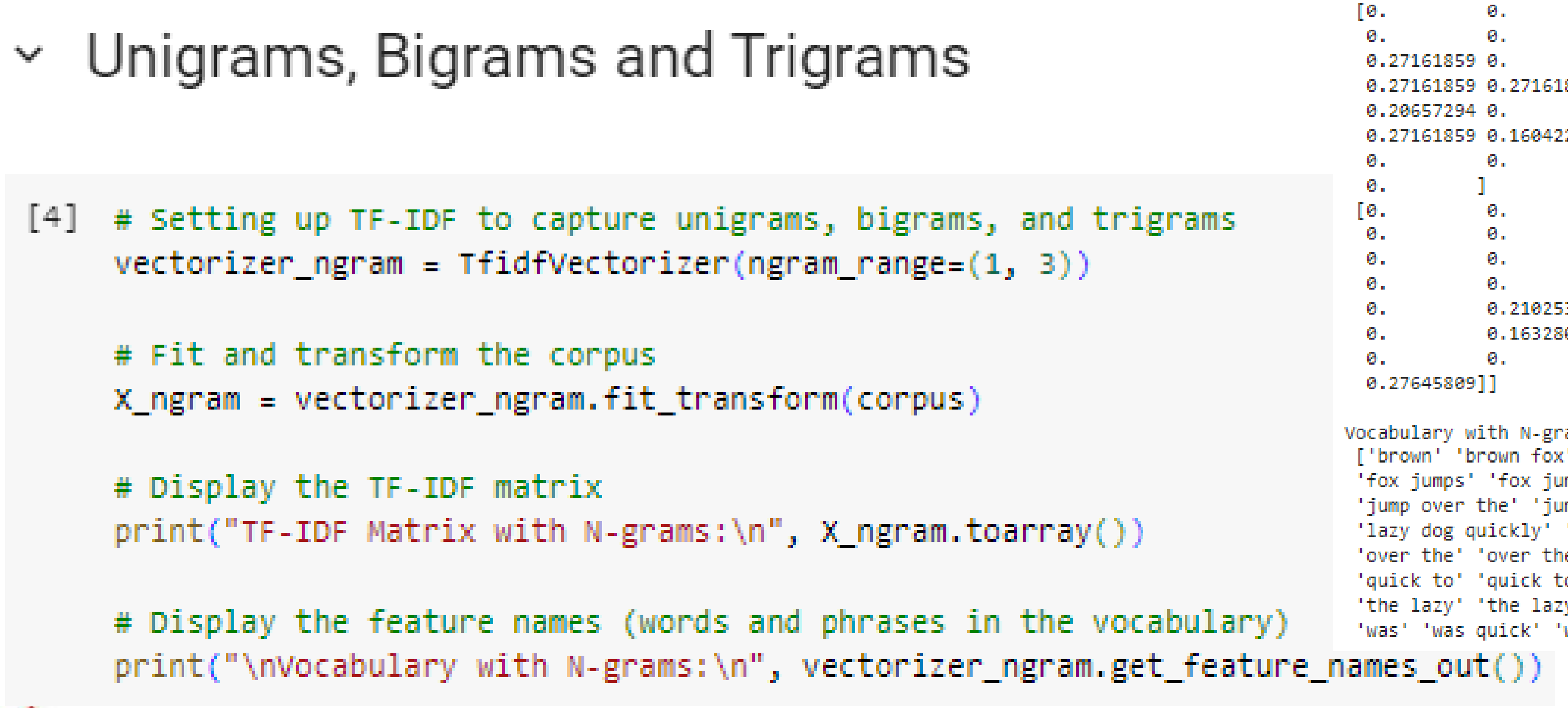
```
[1] from sklearn.feature_extraction.text import TfidfVectorizer

    # Sample corpus of text documents
    corpus = [
        "The quick brown fox jumps over the lazy dog.",
        "Never jump over the lazy dog quickly.",
        "The fox was quick to jump."
    ]

    # Create the TF-IDF Vectorizer
    vectorizer = TfidfVectorizer()

    # Fit and transform the corpus
    X = vectorizer.fit_transform(corpus)

    # Display the TF-IDF matrix
    print("TF-IDF Matrix:\n", X.toarray())

    # Display the feature names (words in the vocabulary)
    print("\nVocabulary:\n", vectorizer.get_feature_names_out())
```



```
TF-IDF Matrix:
 [[0.3988115  0.30330642 0.30330642 0.         0.3988115  0.30330642
  0.         0.30330642 0.30330642 0.         0.47108899 0.
  0.        ]
 [0.         0.35221512 0.         0.35221512 0.         0.35221512
  0.46312056 0.35221512 0.         0.46312056 0.27352646 0.
  0.        ]
 [0.         0.         0.37633075 0.37633075 0.         0.
  0.         0.         0.37633075 0.         0.2922544  0.49482971
  0.49482971]]

Vocabulary:
 ['brown' 'dog' 'fox' 'jump' 'jumps' 'lazy' 'never' 'over' 'quick'
 'quickly' 'the' 'to' 'was']
```

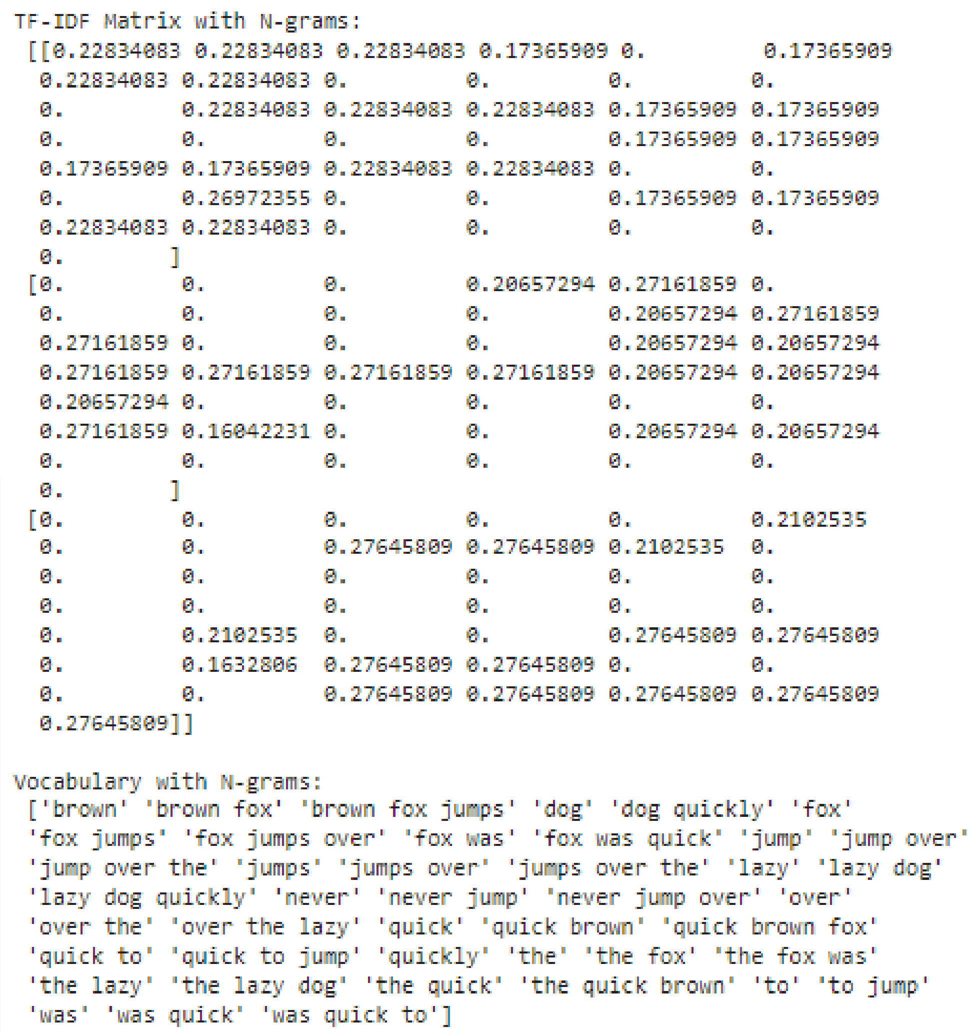
SUKKUR IBA UNIVERSITY COMPUTER SCIENCE

TF-IDF



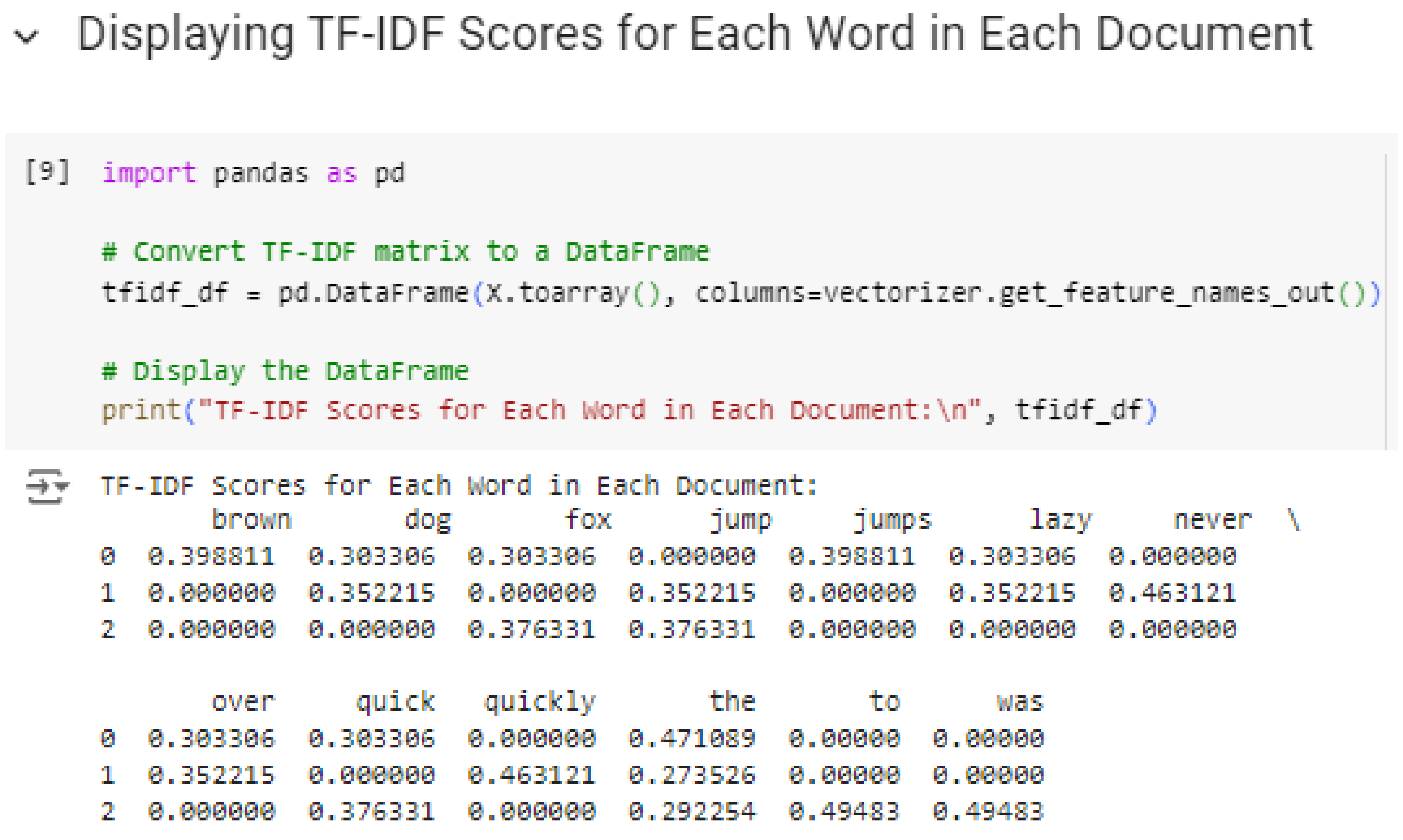
Unigrams, Bigrams and Trigrams

```
[4]  # Setting up TF-IDF to capture unigrams, bigrams, and trigrams
     vectorizer_ngram = TfidfVectorizer(ngram_range=(1, 3))

     # Fit and transform the corpus
     X_ngram = vectorizer_ngram.fit_transform(corpus)

     # Display the TF-IDF matrix
     print("TF-IDF Matrix with N-grams:\n", X_ngram.toarray())

     # Display the feature names (words and phrases in the vocabulary)
     print("\nVocabulary with N-grams:\n", vectorizer_ngram.get_feature_names_out())
```



```
TF-IDF Matrix with N-grams:
 [[0.22834083 0.22834083 0.22834083 0.17365909 0.         0.17365909
  0.22834083 0.22834083 0.         0.         0.         0.
  0.         0.22834083 0.22834083 0.22834083 0.17365909 0.17365909
  0.         0.         0.         0.         0.17365909 0.17365909
  0.17365909 0.17365909 0.22834083 0.22834083 0.         0.
  0.         0.26972355 0.         0.         0.17365909 0.17365909
  0.22834083 0.22834083 0.         0.         0.         0.
  0.        ]
 [0.         0.         0.         0.20657294 0.27161859 0.
  0.         0.         0.         0.         0.20657294 0.27161859
  0.27161859 0.         0.         0.         0.20657294 0.20657294
  0.27161859 0.27161859 0.27161859 0.27161859 0.20657294 0.20657294
  0.20657294 0.         0.         0.         0.         0.
  0.27161859 0.16042231 0.         0.         0.20657294 0.20657294
  0.         0.         0.         0.         0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.2102535
  0.         0.         0.27645809 0.27645809 0.2102535  0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.2102535  0.         0.         0.27645809 0.27645809
  0.         0.1632806  0.27645809 0.27645809 0.         0.
  0.         0.         0.27645809 0.27645809 0.27645809 0.27645809
  0.27645809]]

Vocabulary with N-grams:
 ['brown' 'brown fox' 'brown fox jumps' 'dog' 'dog quickly' 'fox'
 'fox jumps' 'fox jumps over' 'fox was' 'fox was quick' 'jump' 'jump over'
 'jump over the' 'jumps' 'jumps over' 'jumps over the' 'lazy' 'lazy dog'
 'lazy dog quickly' 'never' 'never jump' 'never jump over' 'over'
 'over the' 'over the lazy' 'quick' 'quick brown' 'quick brown fox'
 'quick to' 'quick to jump' 'quickly' 'the' 'the fox' 'the fox was'
 'the lazy' 'the lazy dog' 'the quick' 'the quick brown' 'to' 'to jump'
 'was' 'was quick' 'was quick to']
```

TF-IDF



Displaying TF-IDF Scores for Each Word in Each Document

```
[9] import pandas as pd

    # Convert TF-IDF matrix to a DataFrame
    tfidf_df = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names_out())

    # Display the DataFrame
    print("TF-IDF Scores for Each Word in Each Document:\n", tfidf_df)
```

```
TF-IDF Scores for Each Word in Each Document:
       brown       dog       fox      jump     jumps      lazy     never  \
0  0.398811  0.303306  0.303306  0.000000  0.398811  0.303306  0.000000
1  0.000000  0.352215  0.000000  0.352215  0.000000  0.352215  0.463121
2  0.000000  0.000000  0.376331  0.376331  0.000000  0.000000  0.000000

       over     quick   quickly       the       to      was
0  0.303306  0.303306  0.000000  0.471089  0.00000  0.00000
1  0.352215  0.000000  0.463121  0.273526  0.00000  0.00000
2  0.000000  0.376331  0.000000  0.292254  0.49483  0.49483
```

Word Embeddings

- In NLP, word embedding is a term used for representation of words for text analysis, typically in the form of a real-valued vector.
- That real-valued vector encodes the meaning of the word such that the words that are closer in the vector space are expected to be similar in the meaning.

Summary

- Introduced NLP
- Discussed and implemented Tokenization
- Discussed and implemented Lemmatization
- Discussed and implemented Stop words
- BoW & TF-IDF are better approaches compared to one hot encoding