Text Preprocessing - NLP LECTURE 3

Basic Preprocessing techniques

Author : Harshavardhan

INSTRUCTOR: Nitish Sir, CampusX

Text Representation

Text preprocessing in NLP refers to the series of steps taken to clean, transform, and prepare raw text data before feeding it into a machine learning or natural language processing model. The goal of text preprocessing is to standardize the text, remove noise, and create a consistent and meaningful representation for analysis and modeling.

Techniques covered in this lecture:

- Lower Casing
- Removing HTML Tags
- Removing URLs
- Removing Punctuation
- Chat words treatment
- Spelling Correction
- Removing Stopwords
- Handling Emojis
- Tokenization
- Stemming
- Lemmatization

Importing the csv file

[1]:
```python
# importing numpy and pandas
import pandas as pd
import numpy as np
```

•[2]:
```python
# Data set Link
# https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews
```

[3]:
```python
# Reading the CSV file and storing the data in a DataFrame 'df'
df = pd.read_csv("../archive/IMDB Dataset.csv")
```

[4]:
```python
df
```

[4]:

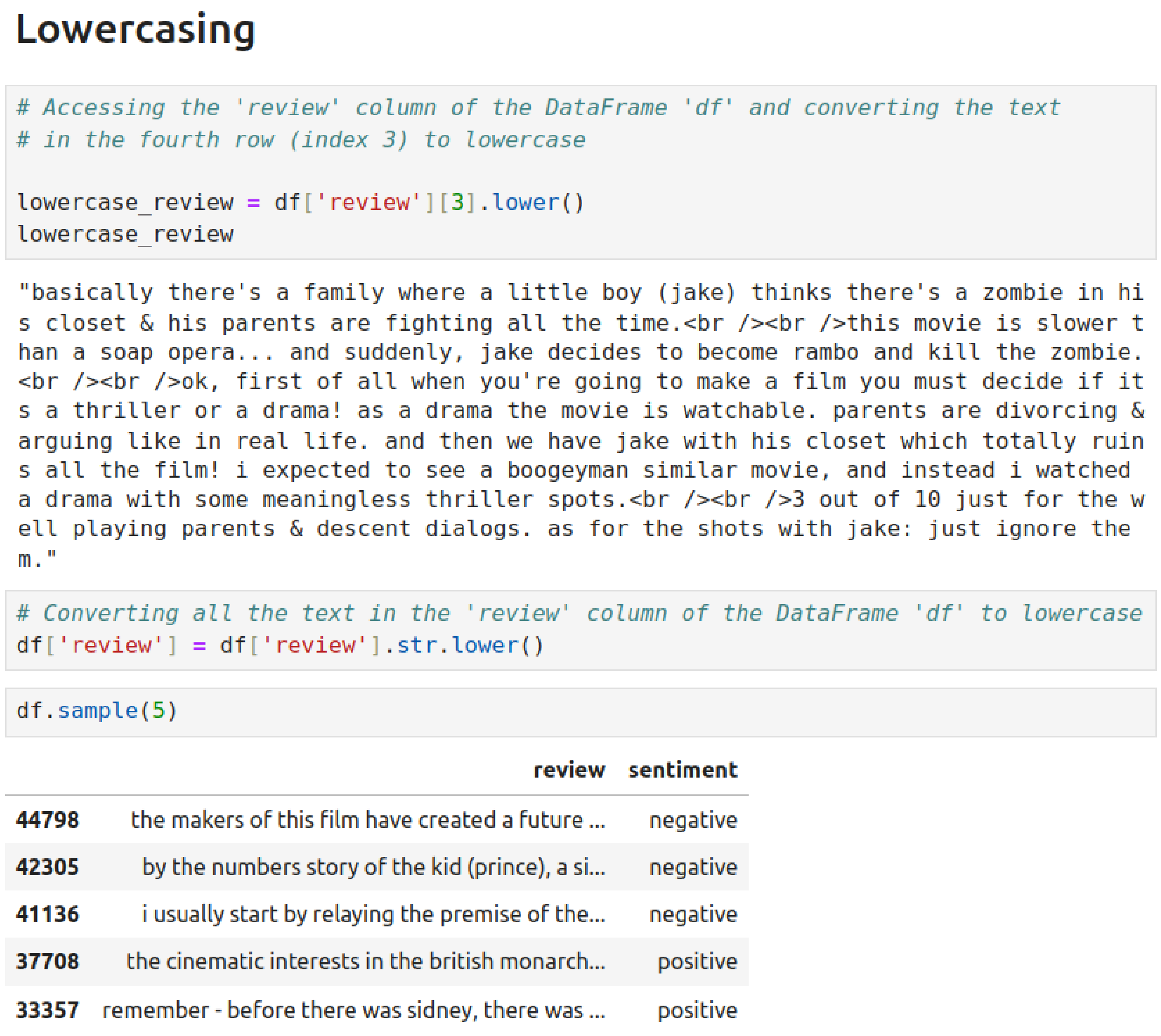
	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. <br /><br />The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
...	...	...
49995	I thought this movie did a down right good job...	positive
49996	Bad plot, bad dialogue, bad acting, idiotic di...	negative
49997	I am a Catholic taught in parochial elementary...	negative
49998	I'm going to have to disagree with the previou...	negative
49999	No one expects the Star Trek movies to be high...	negative

50000 rows × 2 columns

Lowercasing

Lowercasing refers to the process of converting all letters in a piece of text to their lowercase form. The significance of lowercasing in simple words is as follows:

- **Uniformity**: Lowercasing makes all the text consistent by treating different cases (uppercase and lowercase) of the same word as identical. This ensures that "apple," "Apple," and "APPLE" are considered the same word.
- **Word Comparisons**: Lowercasing enables easier word comparisons and matching, making it simpler to find occurrences of specific words regardless of their case.



Lowercasing

```python
# Accessing the 'review' column of the DataFrame 'df' and converting the text
# in the fourth row (index 3) to lowercase

lowercase_review = df['review'][3].lower()
lowercase_review
```

```
"basically there's a family where a little boy (jake) thinks there's a zombie in hi
s closet & his parents are fighting all the time.<br /><br />this movie is slower t
han a soap opera... and suddenly, jake decides to become rambo and kill the zombie.
<br /><br />ok, first of all when you're going to make a film you must decide if it
s a thriller or a drama! as a drama the movie is watchable. parents are divorcing &
arguing like in real life. and then we have jake with his closet which totally ruin
s all the film! i expected to see a boogeyman similar movie, and instead i watched
a drama with some meaningless thriller spots.<br /><br />3 out of 10 just for the w
ell playing parents & descent dialogs. as for the shots with jake: just ignore the
m."
```

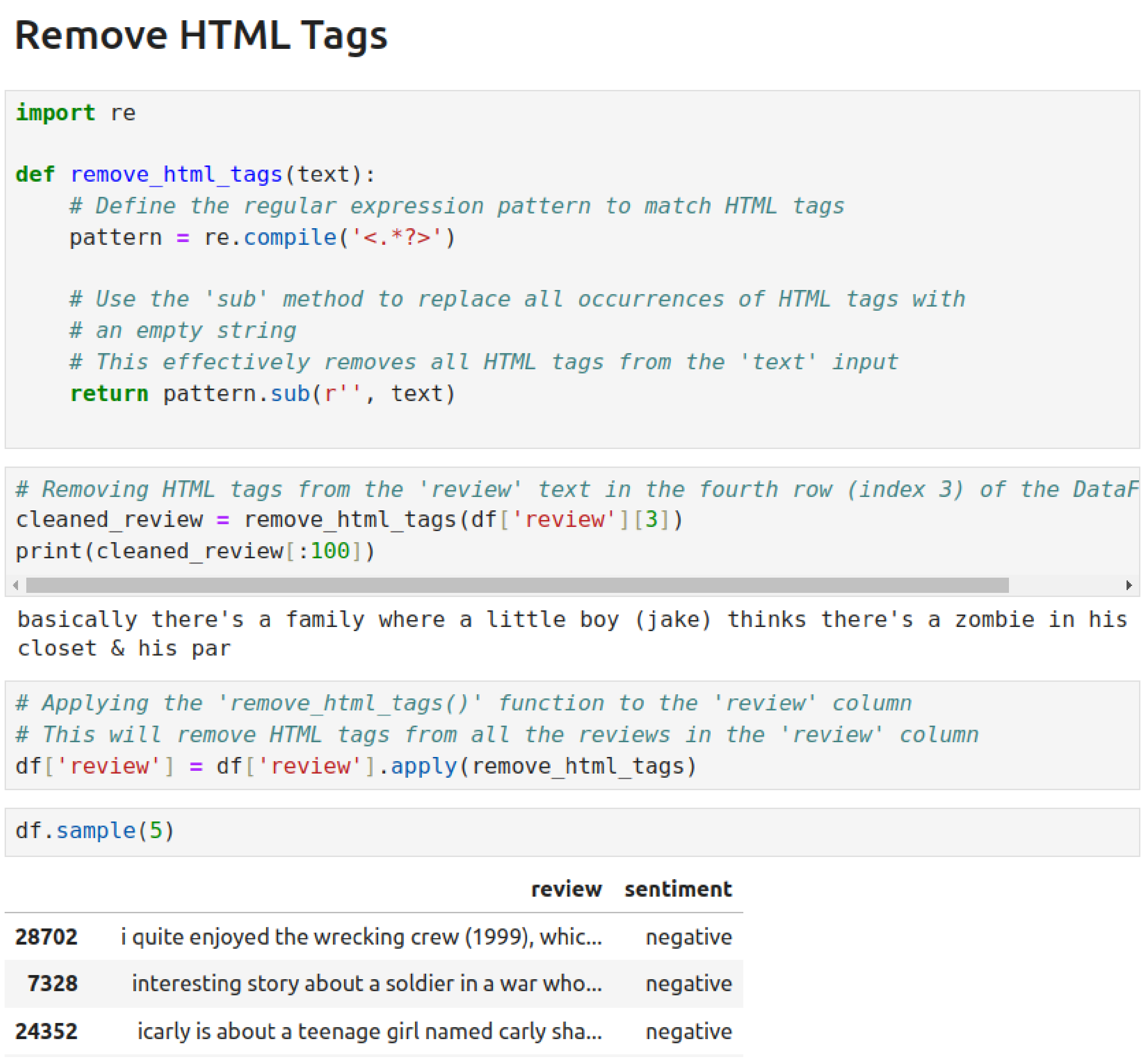
```python
# Converting all the text in the 'review' column of the DataFrame 'df' to lowercase
df['review'] = df['review'].str.lower()
```

```python
df.sample(5)
```

	review	sentiment
44798	the makers of this film have created a future ...	negative
42305	by the numbers story of the kid (prince), a si...	negative
41136	i usually start by relaying the premise of the...	negative
37708	the cinematic interests in the british monarch...	positive
33357	remember - before there was sidney, there was ...	positive

Removing HTML tags

Removing HTML tags in simple words involves eliminating any markup code or formatting elements present in the text data. These tags are used in web pages to structure and style content, but they may not carry meaningful information for certain NLP tasks. Removing HTML tags ensures that only the plain text content remains, making it more suitable for text analysis and processing.



Remove HTML Tags

```python
import re

def remove_html_tags(text):
    # Define the regular expression pattern to match HTML tags
    pattern = re.compile('<.*?>')

    # Use the 'sub' method to replace all occurrences of HTML tags with
    # an empty string
    # This effectively removes all HTML tags from the 'text' input
    return pattern.sub(r'', text)
```

```python
# Removing HTML tags from the 'review' text in the fourth row (index 3) of the DataF
cleaned_review = remove_html_tags(df['review'][3])
print(cleaned_review[:100])
```

```
basically there's a family where a little boy (jake) thinks there's a zombie in his
closet & his par
```

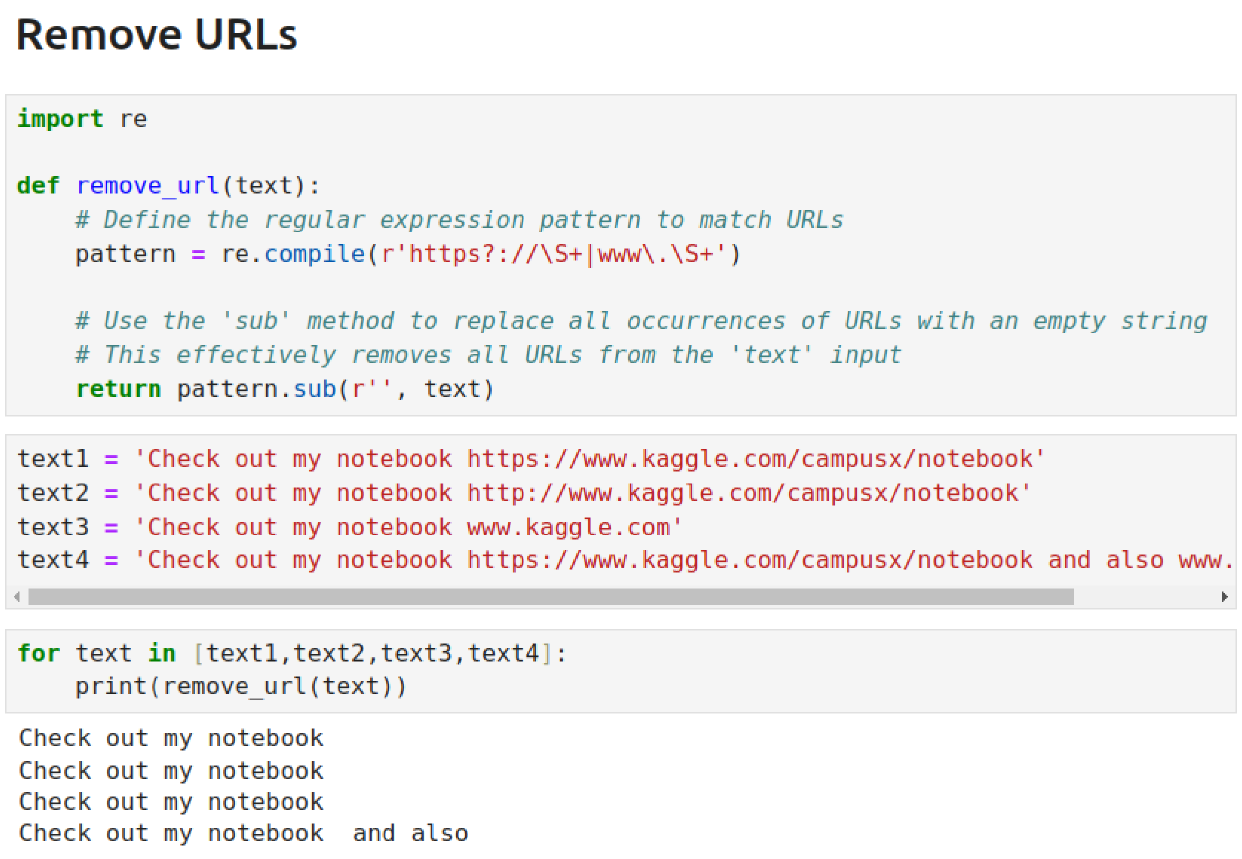
```python
# Applying the 'remove_html_tags()' function to the 'review' column
# This will remove HTML tags from all the reviews in the 'review' column
df['review'] = df['review'].apply(remove_html_tags)
```

```python
df.sample(5)
```

	review	sentiment
28702	i quite enjoyed the wrecking crew (1999), whic...	negative
7328	interesting story about a soldier in a war who...	negative
24352	icarly is about a teenage girl named carly sha...	negative

Removing URL's

In NLP preprocessing, removing URLs from the text data is an essential step to clean and prepare the text for further analysis or modeling. URLs, which are web links, often do not carry significant meaning for most NLP tasks and can introduce noise or disrupt the analysis.



Remove URLs

```python
import re

def remove_url(text):
    # Define the regular expression pattern to match URLs
    pattern = re.compile(r'https?://\S+|www\.\S+')

    # Use the 'sub' method to replace all occurrences of URLs with an empty string
    # This effectively removes all URLs from the 'text' input
    return pattern.sub(r'', text)
```

```python
text1 = 'Check out my notebook https://www.kaggle.com/campusx/notebook'
text2 = 'Check out my notebook http://www.kaggle.com/campusx/notebook'
text3 = 'Check out my notebook www.kaggle.com'
text4 = 'Check out my notebook https://www.kaggle.com/campusx/notebook and also www.
```

```python
for text in [text1,text2,text3,text4]:
    print(remove_url(text))
```

```
Check out my notebook
Check out my notebook
Check out my notebook
Check out my notebook  and also
```

Removing Punctuations

In NLP preprocessing, removing punctuations from the text data is a common practice to clean and simplify the text for further analysis and modeling. Punctuations are characters like periods, commas, exclamation marks, question marks, and other symbols used to indicate pauses, sentence boundaries, or emphasize certain expressions. While they are essential for grammatical correctness in human language, they often do not contribute much to the meaning or context in many NLP tasks.

Remove Punctuation

```python
import string
import time

# The 'string.punctuation' attribute contains all punctuation characters
# It includes symbols like !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
punctuation_chars = string.punctuation
punctuation_chars
```

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

```python
exclude = string.punctuation

def remove_punc(text):
    # Iterate through each character in 'exclude'
    for char in exclude:
        # Replace the current character with an empty string in 'text'
        text = text.replace(char, '')

    # Return the 'text' with all punctuation characters removed
    return text
```

```python
text = 'string . with ! #  punctuation ? '
```

```python
# Record the start time before calling the function
start = time.time()
# Call the 'remove_punc' function to remove punctuation from the 'text'
print(remove_punc(text))
# Calculate the time taken to execute the 'remove_punc' function
time1 = time.time() - start
# Print the time taken in seconds
print(time1)
```

```
string  with    punctuation
8.392333984375e-05
```

```python
# This approach is very slow and inefficient when the dataset is large
```

```python
def remove_punc1(text):
    return text.translate(str.maketrans('','',exclude))
```

```python
start = time.time()
print(remove_punc1(text))
time2 = time.time() - start
print(time2)
```

```
string  with    punctuation
0.00045633316040039O6
```

```python
time1/time2  # 18 times faster
```

```
0.1839080459770115
```

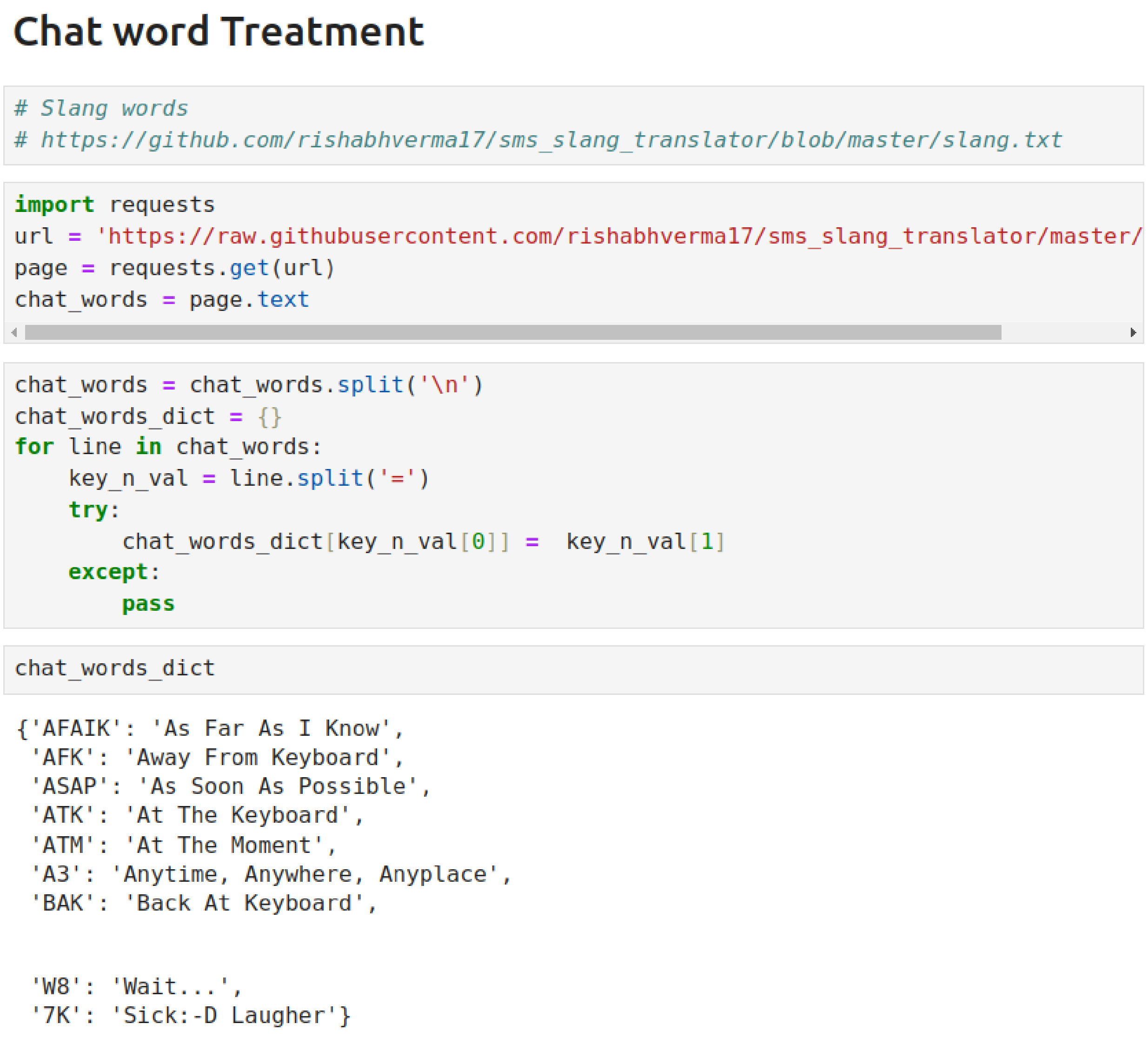
```python
# Applying the 'remove_punc1' function to the 'review' column of the DataFrame 'df'
# This will remove punctuation from all the reviews in the 'review' column
df['review'] = df['review'].apply(remove_punc1)
```

```python
df.sample(3)
```

	review	sentiment
8923	i love sandra bullockshes one of my alltime fa...	negative
7643	corky romano has to be one of the most jaw dro...	negative
13296	im not sure whether i like this film or not i ...	positive

Chat word Treatment

In NLP preprocessing, chat word treatment refers to the process of handling informal or abbreviated words commonly used in chats, social media, and online communication. These chat words may include acronyms, emojis, emoticons, and slang, which can be challenging for NLP models to interpret correctly.



Chat word Treatment

```
# Slang words
# https://github.com/rishabhverma17/sms_slang_translator/blob/master/slang.txt
```

```
import requests
url = 'https://raw.githubusercontent.com/rishabhverma17/sms_slang_translator/master/
page = requests.get(url)
chat_words = page.text
```

```
chat_words = chat_words.split('\n')
chat_words_dict = {}
for line in chat_words:
    key_n_val = line.split('=')
    try:
        chat_words_dict[key_n_val[0]] =  key_n_val[1]
    except:
        pass
```

```
chat_words_dict
```

```
{'AFAIK': 'As Far As I Know',
 'AFK': 'Away From Keyboard',
 'ASAP': 'As Soon As Possible',
 'ATK': 'At The Keyboard',
 'ATM': 'At The Moment',
 'A3': 'Anytime, Anywhere, Anyplace',
 'BAK': 'Back At Keyboard',


 'W8': 'Wait...',
 '7K': 'Sick:-D Laugher'}
```

```python
def chat_conversation(text):
    new_text = []
    for w in text.split():
        if w.upper() in chat_words_dict:
            new_text.append(chat_words_dict[w.upper()])
        else:
            new_text.append(w)
    return " ".join(new_text)
```

```python
chat_conversation('IMHO he is good')
```

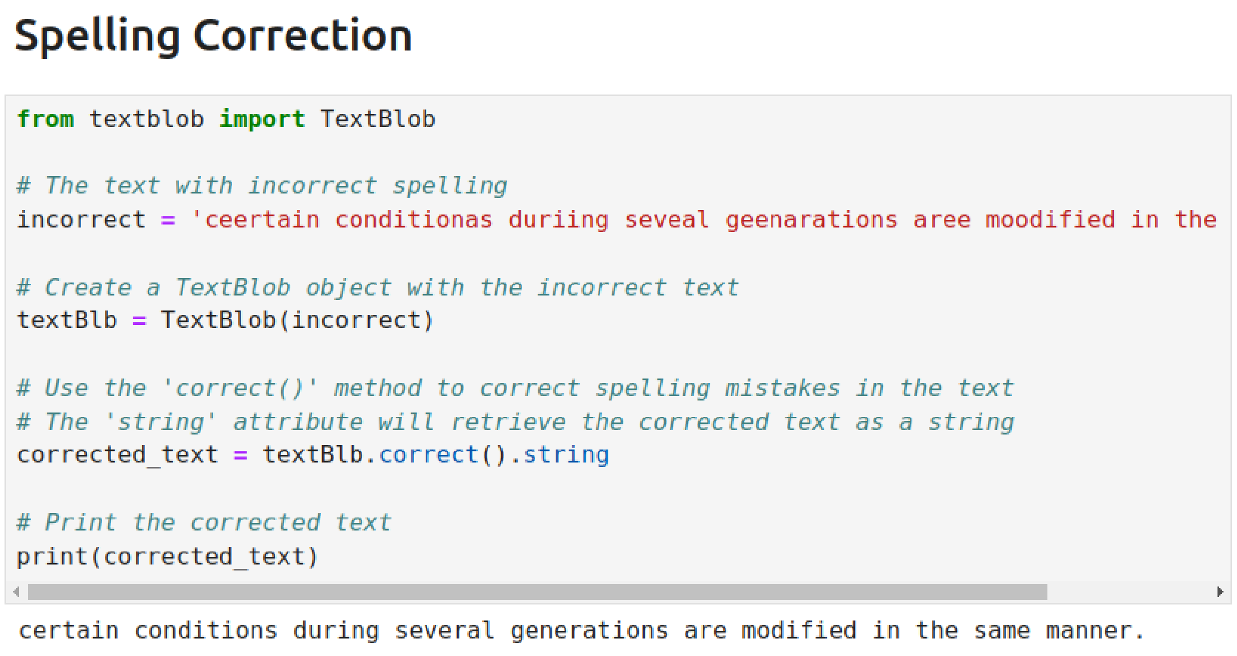
```
'In My Honest/Humble Opinion he is good'
```

```python
chat_conversation('FYI he is good, hes G9')
```

```
'For Your Information he is good, hes Genius'
```

Spelling Correction

Spelling correction is a crucial step in NLP preprocessing, aimed at fixing spelling errors in the text data. Spelling errors can occur due to typos, keyboard mistakes, or other human errors during data input or text generation processes. These errors can negatively impact the performance of NLP models and lead to incorrect analysis and interpretations.



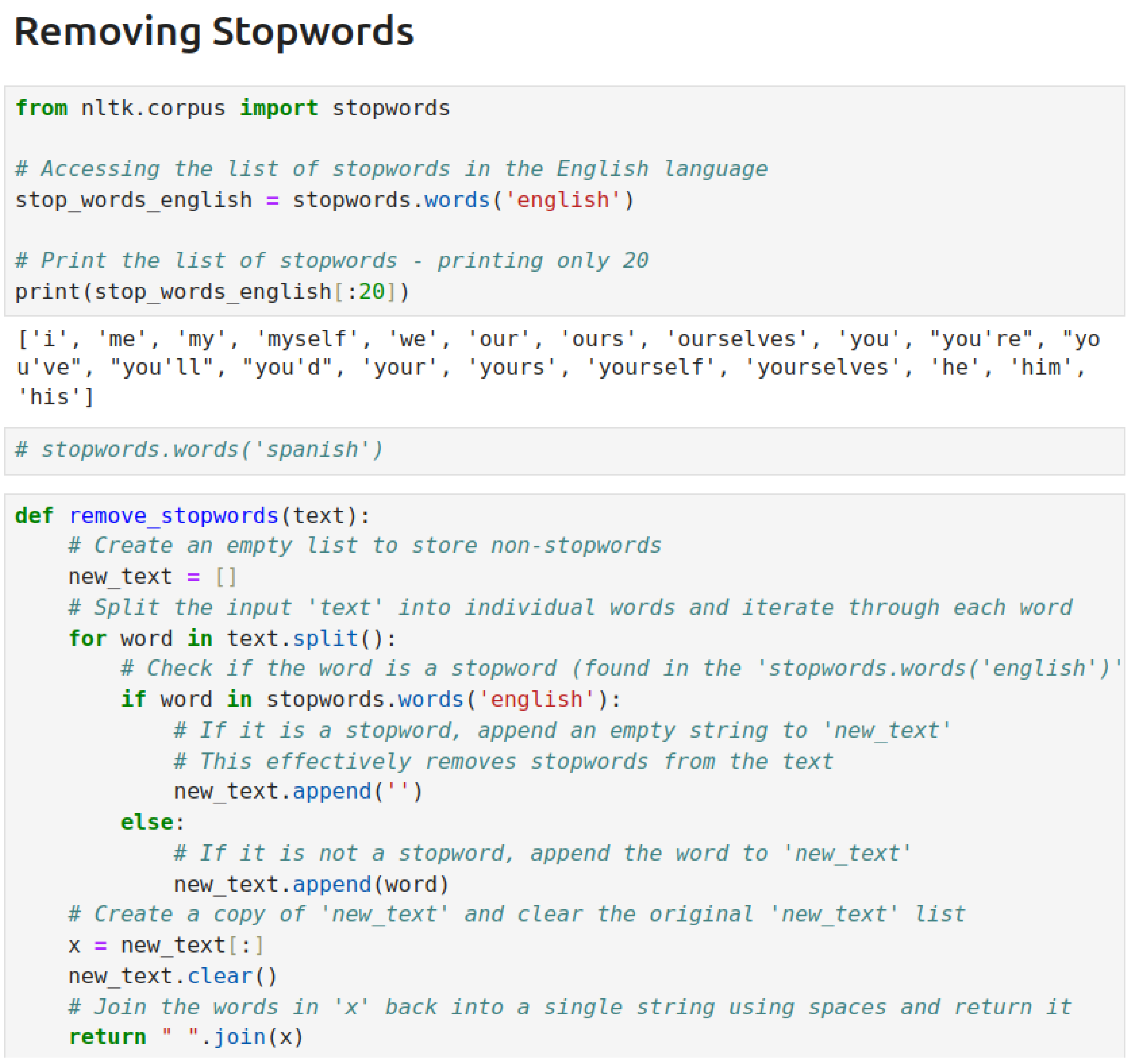
Spelling Correction

```python
from textblob import TextBlob

# The text with incorrect spelling
incorrect = 'ceertain conditionas duriing seveal geenarations aree moodified in the

# Create a TextBlob object with the incorrect text
textBlb = TextBlob(incorrect)

# Use the 'correct()' method to correct spelling mistakes in the text
# The 'string' attribute will retrieve the corrected text as a string
corrected_text = textBlb.correct().string

# Print the corrected text
print(corrected_text)
```

```
certain conditions during several generations are modified in the same manner.
```

Removing stopwords

In NLP preprocessing, removing stop words is a common practice to clean and prepare text data for analysis or modeling. Stop words are common words that appear frequently in the language but often do not carry significant meaning or contribute much to the context of the text. Examples of stop words include "the," "and," "in," "is," "of," "to," etc.



Removing Stopwords

```python
from nltk.corpus import stopwords

# Accessing the list of stopwords in the English language
stop_words_english = stopwords.words('english')

# Print the list of stopwords - printing only 20
print(stop_words_english[:20])
```

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

```python
# stopwords.words('spanish')
```

```python
def remove_stopwords(text):
    # Create an empty list to store non-stopwords
    new_text = []
    # Split the input 'text' into individual words and iterate through each word
    for word in text.split():
        # Check if the word is a stopword (found in the 'stopwords.words('english')'
        if word in stopwords.words('english'):
            # If it is a stopword, append an empty string to 'new_text'
            # This effectively removes stopwords from the text
            new_text.append('')
        else:
            # If it is not a stopword, append the word to 'new_text'
            new_text.append(word)
    # Create a copy of 'new_text' and clear the original 'new_text' list
    x = new_text[:]
    new_text.clear()
    # Join the words in 'x' back into a single string using spaces and return it
    return " ".join(x)
```

```
text = "As the gentle breeze rustled through the leaves, the vibrant colors of the a
print(remove_stopwords(text))
```

```
As  gentle breeze rustled   leaves,  vibrant colors   autumn foliage danced  harmon
y, creating  breathtaking tapestry  nature's artistry  captivated  hearts    beheld
it.
```

```
df['review'][:10].apply(remove_stopwords)
```

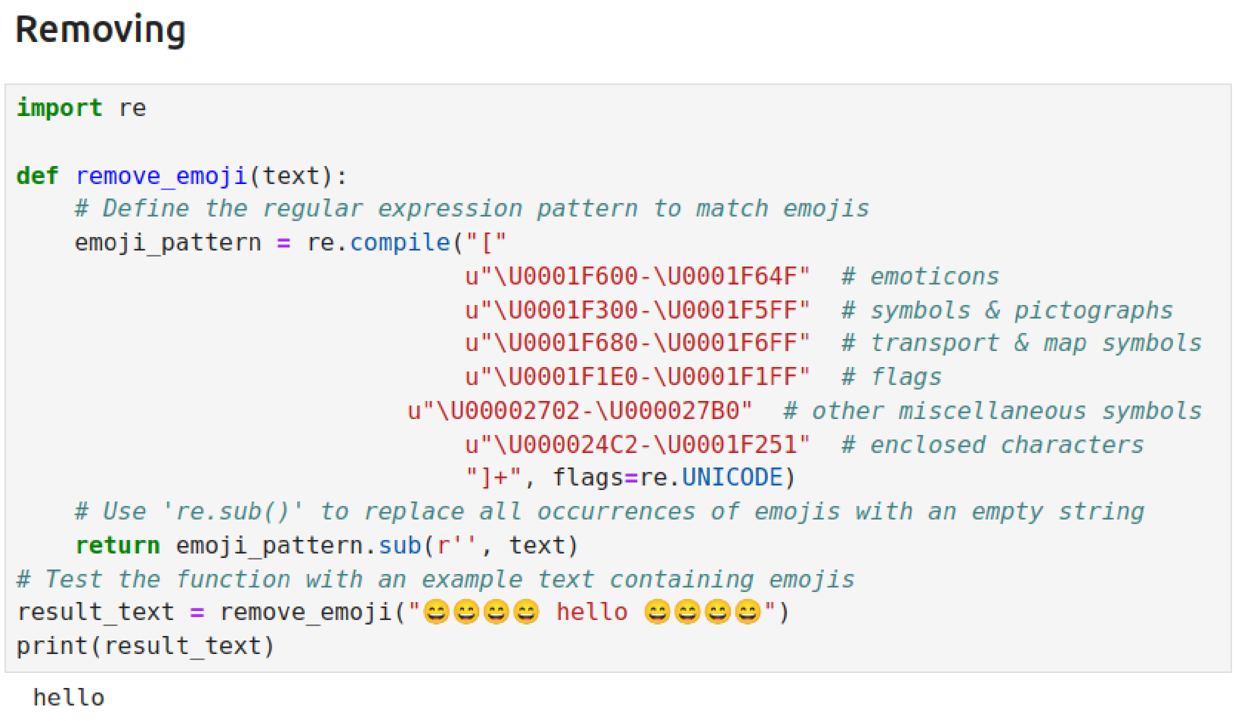
```
0    one    reviewers  mentioned   watching  1 oz e...
1     wonderful little production  filming techniqu...
2     thought    wonderful way  spend time    hot s...
3    basically theres  family   little boy jake thi...
4    petter matteis love   time  money   visually s...
5    probably  alltime favorite movie  story  selfl...
6     sure would like  see  resurrection    dated s...
7     show   amazing fresh innovative idea   70s   ...
8    encouraged   positive comments   film     look...
9      like original gut wrenching laughter   like ...
Name: review, dtype: object
```

Handling emojis

Handling emojis in NLP preprocessing involves managing and processing these visual representations of emotions, expressions, or objects that are commonly used in online communication. Emojis can be challenging for NLP models to interpret because they do not have direct linguistic meanings but can carry important contextual information in text data.

Here's how handling emojis is important and how it is done:

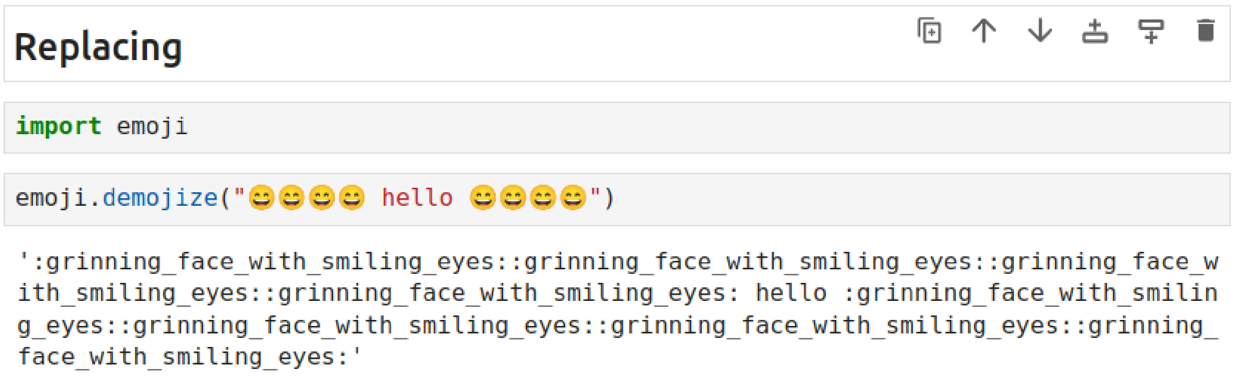
- Emoji Removal: In certain NLP tasks, emojis may not contribute to the analysis and could be treated as noise. Removing emojis can simplify the text and improve model performance.
- Sentiment Analysis: Emojis are commonly used to express emotions. For sentiment analysis tasks, mapping emojis to sentiment scores or categories can enhance the model's understanding of emotional context.



Removing

```
import re

def remove_emoji(text):
    # Define the regular expression pattern to match emojis
    emoji_pattern = re.compile("["
                                u"\U0001F600-\U0001F64F"  # emoticons
                                u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                                u"\U0001F680-\U0001F6FF"  # transport & map symbols
                                u"\U0001F1E0-\U0001F1FF"  # flags
                            u"\U00002702-\U000027B0"  # other miscellaneous symbols
                                u"\U000024C2-\U0001F251"  # enclosed characters
                                "]+", flags=re.UNICODE)
    # Use 're.sub()' to replace all occurrences of emojis with an empty string
    return emoji_pattern.sub(r'', text)
# Test the function with an example text containing emojis
result_text = remove_emoji("😄😄😄😄 hello 😄😄😄😄")
print(result_text)
```

```
 hello
```



Replacing

```python
import emoji
```

```python
emoji.demojize("😄😄😄😄 hello 😄😄😄😄")
```

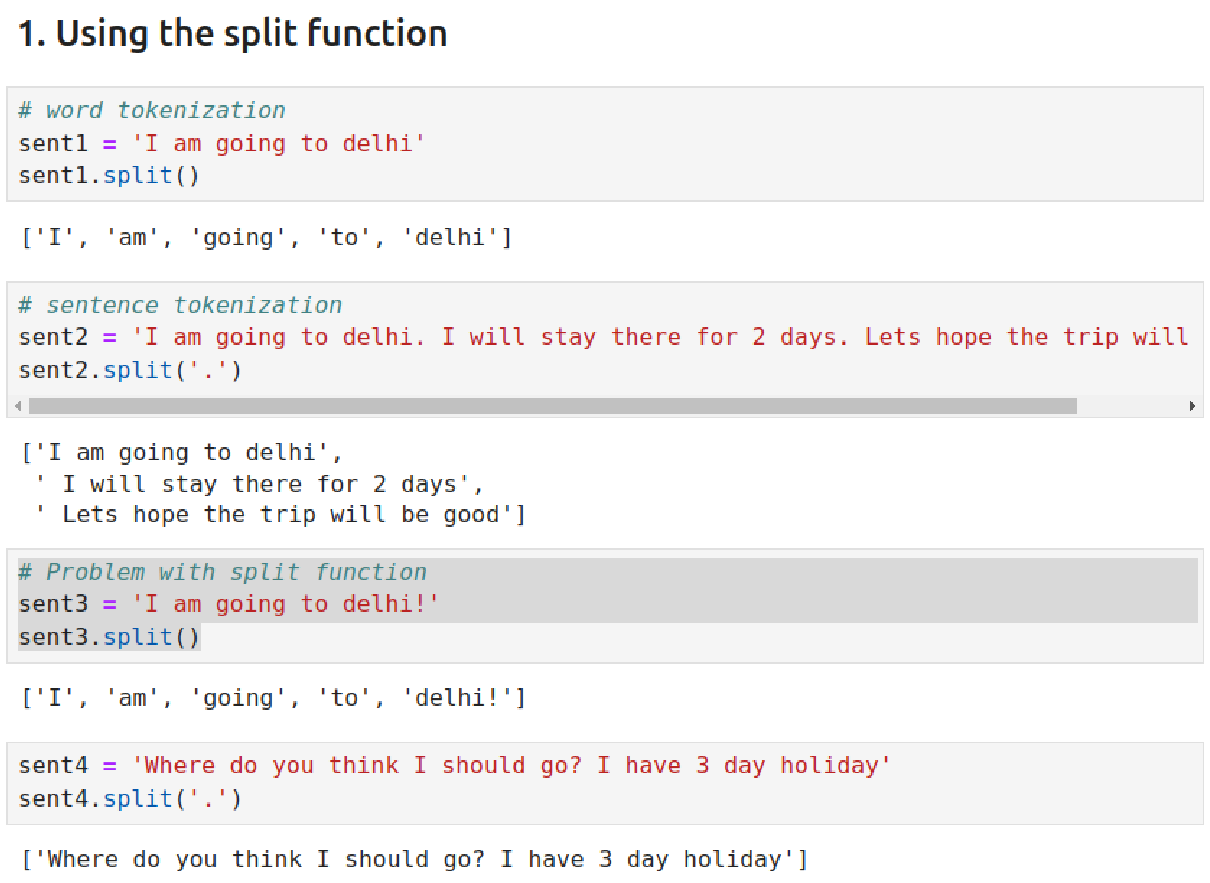
```
':grinning_face_with_smiling_eyes::grinning_face_with_smiling_eyes::grinning_face_w
ith_smiling_eyes::grinning_face_with_smiling_eyes: hello :grinning_face_with_smilin
g_eyes::grinning_face_with_smiling_eyes::grinning_face_with_smiling_eyes::grinning_
face_with_smiling_eyes:'
```

Tokenization

Tokenization in NLP preprocessing is the process of breaking down a text or a sentence into smaller units called tokens. These tokens are typically words or subwords, and tokenization is a fundamental step in preparing text data for various NLP tasks.

Tokenization can be performed at different levels:

- Word Tokenization: Dividing the text into individual words or word-level tokens. For example, "Tokenization is essential for NLP" becomes ['Tokenization', 'is', 'essential', 'for', 'NLP'].
- Subword Tokenization: Splitting words into subword units, such as morphemes or character n-grams. This is useful for languages with complex morphology or to handle out-of-vocabulary words.



1. Using the split function

```
# word tokenization
sent1 = 'I am going to delhi'
sent1.split()
```

```
['I', 'am', 'going', 'to', 'delhi']
```

```
# sentence tokenization
sent2 = 'I am going to delhi. I will stay there for 2 days. Lets hope the trip will
sent2.split('.')
```

```
['I am going to delhi',
 ' I will stay there for 2 days',
 ' Lets hope the trip will be good']
```

```
# Problem with split function
sent3 = 'I am going to delhi!'
sent3.split()
```

```
['I', 'am', 'going', 'to', 'delhi!']
```

```
sent4 = 'Where do you think I should go? I have 3 day holiday'
sent4.split('.')
```

```
['Where do you think I should go? I have 3 day holiday']
```

2. Use Regualr expressions

```
import re
# The sentence to tokenize
sent3 = 'I am going to delhi!'

# Use 're.findall()' to find all word and apostrophe tokens in the sentence
# '\w' matches any word character (letters, digits, or underscore), and "'" matches
# '+' matches one or more occurrences of the pattern (one or more word characters an
# The result will be a list of all tokens in the sentence
tokens = re.findall("[\w']+", sent3)

# Print the list of tokens
print(tokens)
```

```
['I', 'am', 'going', 'to', 'delhi']
```

```
text = '''Lorem ipsum is simply dummy text of the printing and typesetting industry?
 Lorem Ipsum has been the industry's standard dummy text ever since the 1500s,
 when an unknown printer took a gallery of type and scrambled it to make a type spec

sentences = re.compile('[.!?]').split(text)
sentences
```

```
['Lorem ipsum is simply dummy text of the printing and typesetting industry',
 "\n Lorem Ipsum has been the industry's standard dummy text ever since the 1500
s,\n when an unknown printer took a gallery of type and scrambled it to make a type
specimen book",
 '']
```

3. NLTK

```
from nltk.tokenize import word_tokenize,sent_tokenize
```

```
sent1 = 'I am going to visit delhi!'
word_tokenize(sent1)
```

```
['I', 'am', 'going', 'to', 'visit', 'delhi', '!']
```

```
text = '''Lorem ipsum is simply dummy text of the printing and typesetting industry?
 Lorem Ipsum has been the industry's standard dummy text ever since the 1500s,
 when an unknown printer took a gallery of type and scrambled it to make a type spec
sent_tokenize(text)
```

```
['Lorem ipsum is simply dummy text of the printing and typesetting industry?',
 "Lorem Ipsum has been the industry's standard dummy text ever since the 1500s,\n w
hen an unknown printer took a gallery of type and scrambled it to make a type speci
men book."]
```

```
sent5 = 'I have a Ph.D in A.I'
sent6 = "We're here to help! mail us at nks@gmail.com"
sent7 = 'A 5km ride cost $10.58'
```

```
for sent in [sent5,sent6,sent7]:
    print(word_tokenize(sent))
```

```
['I', 'have', 'a', 'Ph.D', 'in', 'A.I']
['We', "'re", 'here', 'to', 'help', '!', 'mail', 'us', 'at', 'nks', '@', 'gmail.co
m']
['A', '5km', 'ride', 'cost', '$', '10.58']
```

4. Spacy (BEST)

```
import spacy
nlp = spacy.load('en_core_web_sm')
```

```
doc1 = nlp(sent5)
doc2 = nlp(sent6)
doc3 = nlp(sent7)
# doc4 = nlp(sent8)
```

```
for token in doc1:
    print(token)
```

```
I
have
a
Ph
.
D
in
A.I
```

Stemming

In grammar, inflection is the modification of a word to express different grammatical categories such as tense,case,voice,aspect,person,number,gender,and mood.

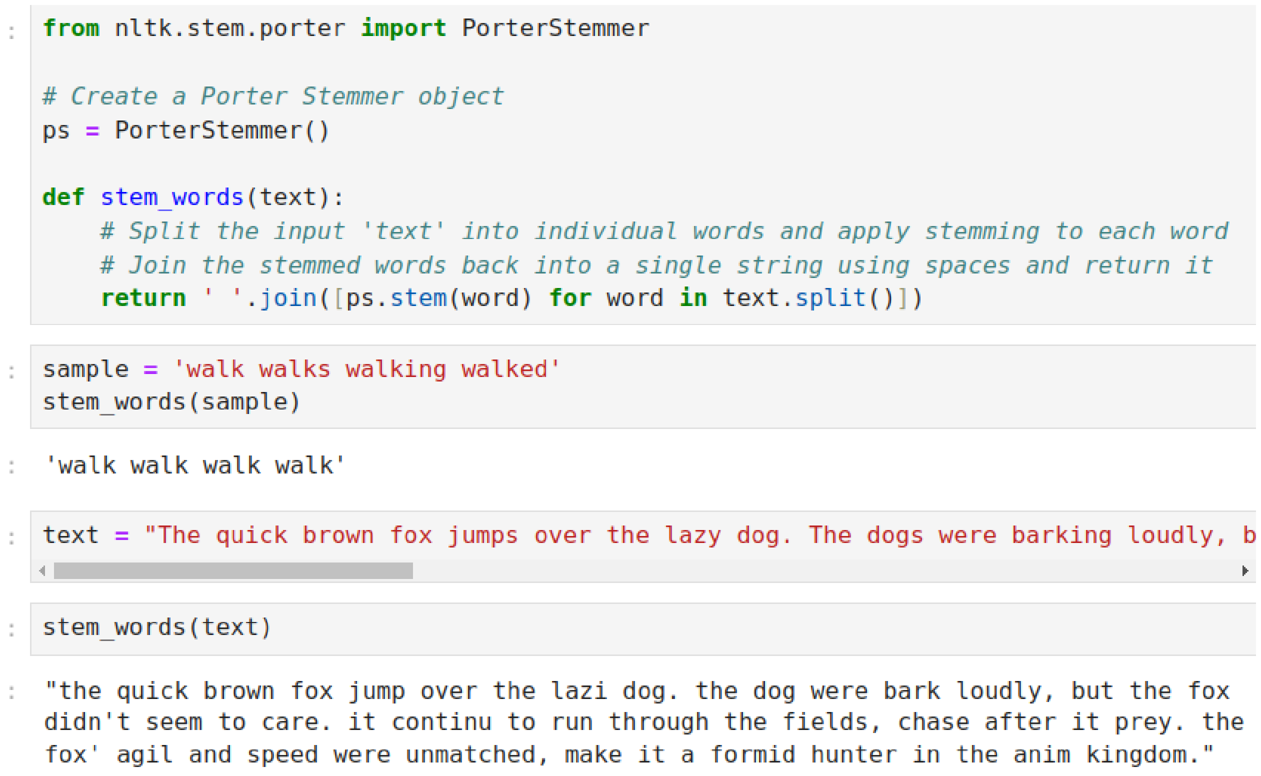
eg. Walk - walk,walking,walked,walker,walks

Stemming is the process of reducing inflection in words to their root forms such as mapping a group of words to the same stem even if the stem itself is not a valid word in the Language.

- its mostly used in information retrieval systems (google search)

Stemmer : Algorithms using which we can perform stemming, examples are

- Porter stemmer (for english)
- Snowball Stemmer(for other language)



```
from nltk.stem.porter import PorterStemmer

# Create a Porter Stemmer object
ps = PorterStemmer()

def stem_words(text):
    # Split the input 'text' into individual words and apply stemming to each word
    # Join the stemmed words back into a single string using spaces and return it
    return ' '.join([ps.stem(word) for word in text.split()])
```

```
sample = 'walk walks walking walked'
stem_words(sample)
```

```
'walk walk walk walk'
```

```
text = "The quick brown fox jumps over the lazy dog. The dogs were barking loudly, b
```

```
stem_words(text)
```

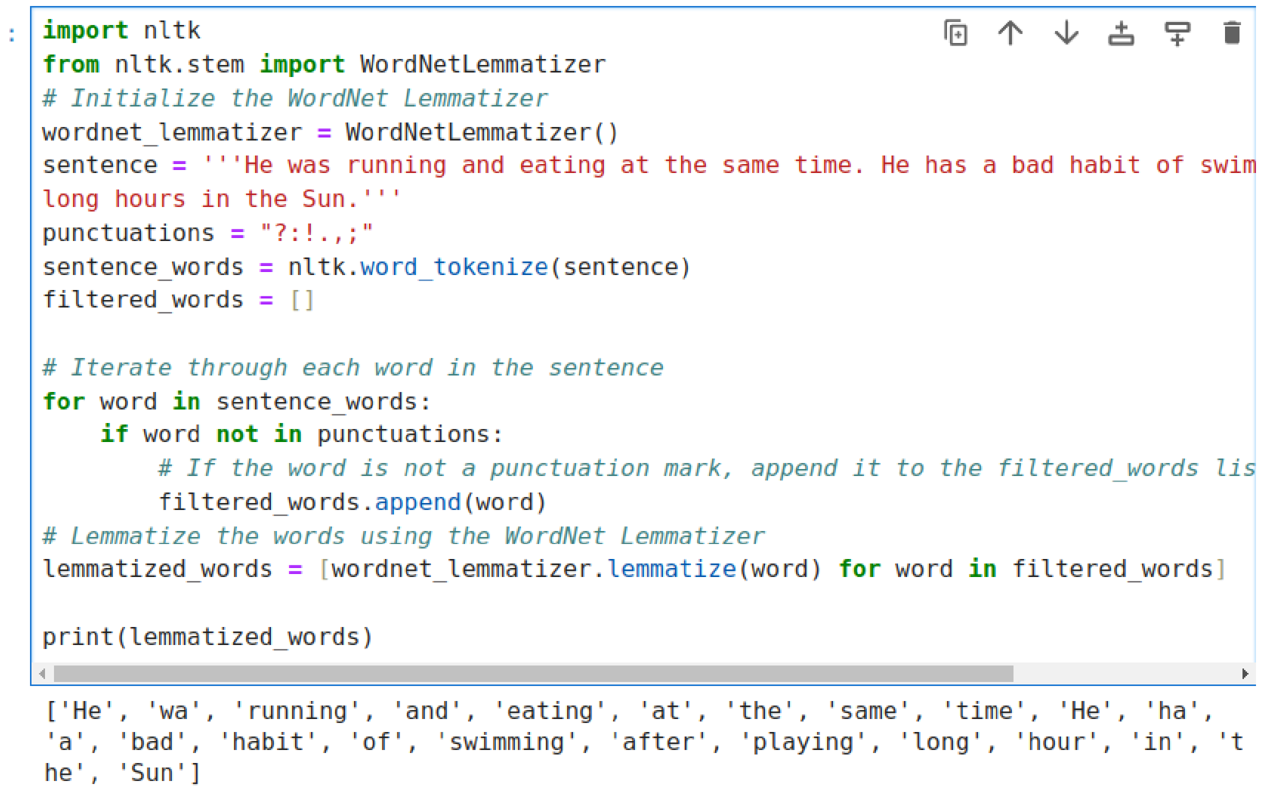
```
"the quick brown fox jump over the lazi dog. the dog were bark loudly, but the fox
didn't seem to care. it continu to run through the fields, chase after it prey. the
fox' agil and speed were unmatched, make it a formid hunter in the anim kingdom."
```

Lemmatization

Lemmatization, unlike stemming,reduces the inflected words properly ensuring that the root word belongs to the language. In Lemmatization the root word is called Lemma. A lemma (plural lemmas or lemmata) is the canonical form, dictionary form, or citation form of a set of words.

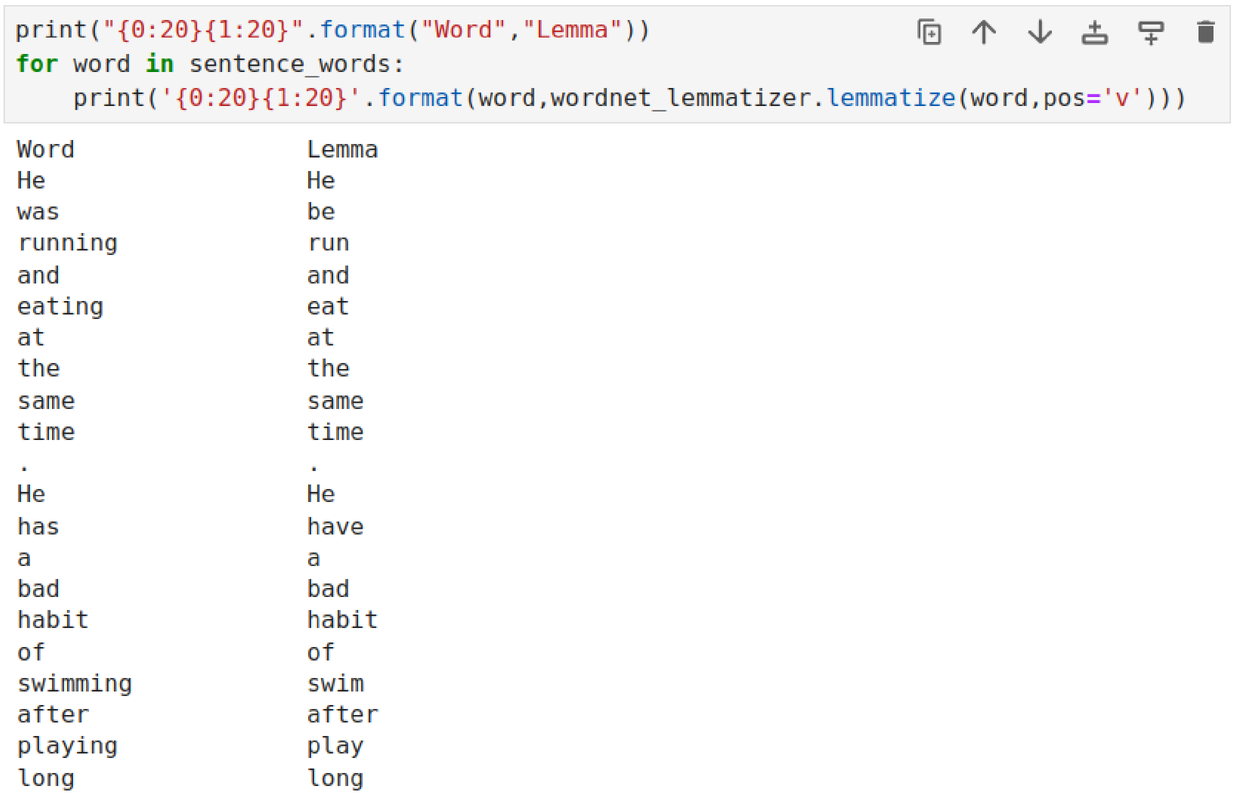
- almost same as stemming, but the root word here is a valid word
- It takes a little longer time when compared to stemming
- if we don't have to show the output to the user then we can use stemming
- else we can use lemmatization

Lemmatization is done using a lexical dictionary instead of an algorithm. The WORDNET lexical dictionary is used here



```
import nltk
from nltk.stem import WordNetLemmatizer
# Initialize the WordNet Lemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
sentence = '''He was running and eating at the same time. He has a bad habit of swim
long hours in the Sun.'''
punctuations = "?:!.,;"
sentence_words = nltk.word_tokenize(sentence)
filtered_words = []

# Iterate through each word in the sentence
for word in sentence_words:
    if word not in punctuations:
        # If the word is not a punctuation mark, append it to the filtered_words lis
        filtered_words.append(word)
# Lemmatize the words using the WordNet Lemmatizer
lemmatized_words = [wordnet_lemmatizer.lemmatize(word) for word in filtered_words]

print(lemmatized_words)
```

```
['He', 'wa', 'running', 'and', 'eating', 'at', 'the', 'same', 'time', 'He', 'ha',
'a', 'bad', 'habit', 'of', 'swimming', 'after', 'playing', 'long', 'hour', 'in', 't
he', 'Sun']
```



```python
print("{0:20}{1:20}".format("Word","Lemma"))
for word in sentence_words:
    print('{0:20}{1:20}'.format(word,wordnet_lemmatizer.lemmatize(word,pos='v')))
```

```
Word                Lemma
He                  He
was                 be
running             run
and                 and
eating              eat
at                  at
the                 the
same                same
time                time
.                   .
He                  He
has                 have
a                   a
bad                 bad
habit               habit
of                  of
swimming            swim
after               after
playing             play
long                long
```