Natural Language Processing (NLP) is a branch of Data Science which deals with Text data. Apart from numerical data, Text data is available to a great extent which is used to analyze and solve business problems. But before using the data for analysis or prediction, processing the data is important.

To prepare the text data for the model building we perform text preprocessing. It is the very first step of NLP projects. Some of the preprocessing steps are:

- Removing punctuations like . , ! $( ) * % @
- Removing URLs
- Removing Stop words
- Lower casing
- Tokenization
- Stemming
- Lemmatization

We need to use the required steps based on our dataset. In this article, we will use **SMS Spam data** to understand the steps involved in Text Preprocessing in NLP.

Let's start by importing the pandas library and reading the data.

Punctuation Removal:

In this step, all the punctuations from the text are removed. string library of Python contains some pre-defined list of punctuations such as '!"#$%&'()*+,-./:;?@[\]^_`{|}~'

```
#library that contains punctuation
import string
string.punctuation
#defining the function to remove punctuation
def remove_punctuation(text):
    punctuationfree="".join([i for i in text if i not in string.punctuation])
    return punctuationfree
#storing the puntuation free text
data['clean_msg']= data['v2'].apply(lambda x:remove_punctuation(x))
data.head()
```

Lowering the text:

It is one of the most common text preprocessing Python steps where the text is converted into the same case preferably lower case. But it is not necessary to do this step every time you are working on an NLP problem as for some problems lower casing can lead to loss of information.

For example, if in any project we are dealing with the emotions of a person, then the words written in upper cases can be a sign of frustration or excitement.

```
data['msg_lower']= data['clean_msg'].apply(lambda x: x.lower())
```

Output: All the text of clean_msg column are converted into lower case and stored in msg_lower column

Stop word removal:

Stopwords are the commonly used words and are removed from the text as they do not add any value to the analysis. These words carry less or no meaning.

NLTK library consists of a list of words that are considered stopwords for the English language. Some of them are : [i, me, my, myself, we, our, ours, ourselves, you, you're, you've, you'll, you'd, your, yours, yourself, yourselves, he, most, other, some, such, no, nor, not, only, own, same, so, then, too, very, s, t, can, will, just, don, don't, should, should've, now, d, ll, m, o, re, ve, y, ain, aren't, could, couldn't, didn't, didn't]

But it is not necessary to use the provided list as stopwords as they should be chosen wisely based on the project. **For example**, 'How' can be a stop word for a model but can be important for some other problem where we are working on the queries of the customers. We can create a customized list of stop words for different problems.

```
#importing nlp library
import nltk
#Stop words present in the library
stopwords = nltk.corpus.stopwords.words('english')
stopwords[0:10]
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
"you're"]
#defining the function to remove stopwords from tokenized text
def remove_stopwords(text):
    output= [i for i in text if i not in stopwords]
    return output
#applying the function
data['no_stopwords']= data['msg_tokenied'].apply(lambda
x:remove_stopwords(x))
```

Output: Stop words that are present in the nltk library such as in, until, to, I, here are removed from the tokenized text and the rest are stored in the no_stopwords column.

Stemming:

It is also known as the text standardization step where the words are stemmed or diminished to their root/base form. **For example**, words like 'programmer', 'programming, 'program' will be stemmed to 'program'.

But the **disadvantage** of stemming is that it stems the words such that its root form loses the meaning or it is not diminished to a proper English word. We will see this in the steps done below.

```
#importing the Stemming function from nltk library
from nltk.stem.porter import PorterStemmer
```

```
#defining the object for stemming
porter_stemmer = PorterStemmer()
#defining a function for stemming
def stemming(text):
stem_text = [porter_stemmer.stem(word) for word in text]
    return stem_text
data['msg_stemmed']=data['no_sw_msg'].apply(lambda x: stemming(x))
```

Output: In the below image, we can see how some words are stemmed to their base.

crazy-> crazi

available-> avail

entry-> entri

early-> earli

Lemmatization:

It stems the word but makes sure that it does not lose its meaning. Lemmatization has a pre-defined dictionary that stores the context of words and checks the word in the dictionary while diminishing.

The difference between Stemming and Lemmatization can be understood with the example provided below.

Original Word	After Stemming	After Lemmatization
goose	goos	goose
geese	gees	goose

```
from nltk.stem import WordNetLemmatizer
#defining the object for Lemmatization
wordnet_lemmatizer = WordNetLemmatizer()
#defining the function for lemmatization
def lemmatizer(text):
lemm_text = [wordnet_lemmatizer.lemmatize(word) for word in text]
    return lemm_text
data['msg_lemmatized']=data['no_stopwords'].apply(lambda x:lemmatizer(x))
```

Output: The difference between Stemming and Lemmatization can be seen in the below output.

In the first row- crazy has been changed to **crazi** which has no meaning but for lemmatization, it remained the same i.e **crazy**

In the last row- goes has changed to **goe** while stemming but for lemmatization, it has converted into **go** which is meaningful.