* Procedure in NLP *

1. Import the General libraries, NLP module like NLTK and SPACY.
2. Load the dataset.
3. Text Preprocessing:
    - i. Removing html tags
    - ii. Removing Punctuations
    - iii. Performing stemming
    - iv. Removing Stop words
    - v. Expanding contractions.
4. Apply Tokenization.
5. Apply Stemming.
6. Apply POS Tagging.
7. Apply Lemmatization.
8. Apply label encoding.
9. Feature Extraction.

10 Text to Numerical vector conversion:
- i. Apply BOW(Count-Vectorizer).
- ii. Apply TFIDF vectorizer.
- iii. Apply Word2Vector vectorizer.
- iv. Apply Glove.

11. Data preprocessing.
12. Model Building.

* Terms Used in NLP *

Document	:	Each row in dataset is called Document.
Corpus	:	Collection of Documents(all rows) is called Corpus.
Vocabulary	:	Unique Words in Corpus
Segmentation	:	Breaking multiple sentences into single individual sentence is called Segmentation.
Tokenization	:	Process of breaking sentence into Words is called Tokenization and the words are called Tokens.
StopWords	:	Common words used in any language are called Stop-Words
Stemming	:	Process of removing or replacing suffixes of word to get the root or base word is called Stemming. But sometimes meaning of word will lost.

Lemmatization : Process of removing or replacing suffixes of word to get the root or base word is called
Lemmatization. Here words have dictionary meaning.
NER Tagging : Process of Adding Tags to each word like "Person, Place, Currency" etc. is called NER Tagging.
POS Tagging : Process of Adding Part of Speech Tags to each word is called POS Tagging..
Chunking : Process of Conversion of sentence to a flat tree is called Chunking.

* Text Pre-Processing Steps *

Text preprocessing is a crucial step in NLP. Cleaning our text data in order to convert it into a presentable form that is analyzable and predictable for our task is known as text preprocessing.
Many steps can be taken in text preprocessing, few steps are,
A. Basic Techniques:
1. Lowering Case
2. Remove Punctuations
3. Removal of special characters and Numbers
4. Removal of HTML tags
5. Removal of URL's
6. Removal of Extra Spaces
7. Expanding Contraction
8. Text Correction

B. Advanced Techniques:
1. Apply Tokenization
2. Stop Word Removal
3. Apply Stemming
4. Apply Lemmatization

C. More Advanced Techniques:
1. POS(Part Of Speech) Tagging
2. NER(Name Entity Recognation)

A. Basic Techniques

1. Lowering Case

Lowering Case of text is essential step in text preprocessing due to following reasons:

1. The same words, one in upper case and other in lower case are

considered as different words while creating BOW, hence lowering add the same value for both the words.

2. In TF-IDF CountVectorization techniques the frequency of words is considered with irrespective of the case.

3. Lowering decreasing the size of the vocabulary and hence reduce the dimensionality.

In [ ]:

```
sentence="What is the STEP by step guide to invest In share market in india?"
sentence_lower=str(sentence).lower()
print("Original Sentence:", sentence)
print("--"*60)
print("Lowered Sentence:", sentence_lower)
```

```
Original Sentence: What is the STEP by step guide to invest In share marke
t in india?
--------------------------------------------------------------------------
----------------------------------------------
Lowered Sentence: what is the step by step guide to invest in share market
in india?
```

In the `Original Sentence` we have two `Step` with different cases and same meaning in sentence, after coverting everything to lower both words look similar and we reduced the dimensionality.

2. Removing Punctuations

To remove Punctuations we are going to use python "String" library.

In [ ]:

```
import string
punc=string.punctuation
punc
```

Out[2]:

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

Above are the Punctuations in any language

In [ ]:

```
sentence="Hello Everyone, this is team Data Dynamos ! We are got an project of Quora Ques
without_punc=[word for word in sentence.split(" ") if word not in list(punc)]
print("Original Sentence:", sentence)
print("--"*60)
print("Sentence without Punctuations:", " ".join(without_punc))
```

```
Original Sentence: Hello Everyone, this is team Data Dynamos ! We are got
an project of Quora Question SImilirity ^ . We are actually happy !! Becau
se we wanted this project * *
--------------------------------------------------------------------------
----------------------------------------------
Sentence without Punctuations: Hello Everyone, this is team Data Dynamos W
e are got an project of Quora Question SImilirity We are actually happy !!
Because we wanted this project
```

3. Removing Special Characters and Numbers

Special Characters and numbers like "!,@,#,%,^,&,$,+,*, 1 to 9" have no meaning in the sentence and they do not contribute to any sentence classification. And there is one senario when these special characters attached to any word will considered as different word which is already present in the sentence. eg. "Shocked" and "Shocked!" considered as different words but we know they have same meaning. Hence its better to remove any special characters there for dimensionality is also reduces.
We are going to use python "re package" to remove special characters and numbers.

In [ ]:

```
import re
sentence="Find the remainder when [math]23^{24}[/math] is divided by 24,23?"
sentence_clean=re.sub("[^a-zA-Z]", " ", sentence)
print("Original Sentence:", sentence)
print("--"*60)
print("Clean Sentence:", sentence_clean)
```

```
Original Sentence: Find the remainder when [math]23^{24}[/math] is divided
by 24,23?
--------------------------------------------------------------------------
----------------------------------------------
Clean Sentence: Find the remainder when  math          math  is divided by
```

In Original Sentence "{},[],/,?,^" are the special characters, Clean Sentence contains no special characters and numbers.

4. Removal of HTML Tags

When we Scrap data from any website then dataset contains HTML tags. We might face problem if HTML Tags present in our dataset. Hence it prefered to remove these tags.

In [ ]:

```
sentence='''<h3 style="color:red; font-family:Arial Black">Hello Guys How Are You</h3>'''
clean_sentence=re.sub("<.*?>", "", sentence)
print("Original Sentence:", sentence)
print("--"*60)
print("Clean Sentence:", clean_sentence)
```

```
Original Sentence: <h3 style="color:red; font-family:Arial Black">Hello Gu
ys How Are You</h3>
--------------------------------------------------------------------------
----------------------------------------------
Clean Sentence: Hello Guys How Are You
```

The Original Sentence contains HTML tags, after removing these tags using re.sub function of python regex, our Sentence looks human readable.

5. Removing URL's

Some times in the Quora question people provide some external links and url's. As we know that the urls are the random combinations of strings which does not cotains any specific meaning. Hence is useful to remove thes urls.

In [ ]:

```
sentence="I visited https://github.com/surajh8596/NLP-Sentiment-Analysis-/tree/main/Senti
clean_sentence=re.sub("(http|https|www)\S+", "", sentence)
print("Original Sentence:", sentence)
print("--"*60)
print("Clean Sentence:", clean_sentence)
```

```
Original Sentence: I visited https://github.com/surajh8596/NLP-Sentiment-A
nalysis-/tree/main/Sentiment%20Analysis (https://github.com/surajh8596/NLP
-Sentiment-Analysis-/tree/main/Sentiment%20Analysis) link and I found very
interesting sentiment analysis projects.
--------------------------------------------------------------------------
----------------------------------------------
Clean Sentence: I visited  link and I found very interesting sentiment ana
lysis projects.
```

Original sentence conatins an external website link, which cause problem in our analysis. So after removing this link check the clean sentence, with no url.

6. Removing Extra Spaces

There is some senario where users insert extra spaces at the start, at the end or at the anywhere in the sentence. We need to remove all the extra spaces inserted by an user.

In [ ]:

```
sentence="Hi Team    Data Dynamos, How is your     project going on            ?"
clean_sentence=re.sub(" +"," ", sentence)
print("Original Sentence:", sentence)
print("--"*60)
print("Clean Sentence:", clean_sentence)
```

```
Original Sentence: Hi Team    Data Dynamos, How is your     project going
on            ?
--------------------------------------------------------------------------
----------------------------------------------
Clean Sentence: Hi Team Data Dynamos, How is your project going on ?
```

7. Expanding Contraction

Contractions are words or combinations of words that are shortened by dropping letters and replacing them by an apostrophe. Nowadays, where everything is shifting online, we communicate with others more through text messages or posts on different social media like Facebook, Instagram, Whatsapp, Twitter, LinkedIn, etc. in the form of texts. With so many people to talk, we rely on abbreviations and shortened form of words for texting people.

We need to exapnd these contractions so that we can easliy apply tokenization and normalization(stemming and lemmatization). Here we are going to use contrations python library to exapand the constraction words.

In [ ]:

```
import contractions
```

In [ ]:

```
sentence="We've reached final step of our data science internship. We'll meet u in projec
clear_sentence=contractions.fix(sentence)
print("Original Sentence:", sentence)
print("--"*60)
print("Clear Sentence:", clear_sentence)
```

```
Original Sentence: We've reached final step of our data science internshi
p. We'll meet u in project presentation.
--------------------------------------------------------------------------
---------------------------------------------
Clear Sentence: We have reached final step of our data science internship.
We will meet you in project presentation.
```

Original Sentence contains contraction words like "We've","We'll","u" . And the expanded words for these constraction are "We have","We will", "You" .

8. Text Correction

To correct the text we are going to use TextBlob from NLTK

In [ ]:

```
from textblob import TextBlob
sentence="We have reachedd final step of our data science Trainig. We'll meet youu in pro
textblob=TextBlob(sentence)
correct_sentence=textblob.correct()
print("Original Sentence:", sentence)
print("--"*60)
print("Correct Sentence:", correct_sentence)
```

```
Original Sentence: We have reachedd final step of our data science Traini
g. We'll meet youu in project presentatiom.
--------------------------------------------------------------------------
---------------------------------------------
Correct Sentence: He have reached final step of our data science Training.
He'll meet you in project presentation.
```

B. Advanced Techniques

1. Apply Tokenization

Tokenization is a process of breaking down sentence into words. These words are called Tokens. Here, tokens can be either words, characters, or subwords.
Tokenization is broadly classified into 3 types:
  a. Sentence Tokenization
  b. Word Tokenization

c. SubWord(n-gram characters) Tokenization

Here we can use string "Split" method for word tokenization only. For Charcter and SubWord Tokenization we need to use "NLTK" inbuit funvtion.

a. Sentence Tokenization

In [ ]:

```
from nltk.tokenize import sent_tokenize
sentence='''Our Team name is Team Data Dynamos and we have selected Quora question simila
tokens=sent_tokenize(sentence)
print("Original Sentence:", sentence)
print("--"*60)
print("Sentence Tokens:", tokens)
```

```
Original Sentence: Our Team name is Team Data Dynamos and we have selected
Quora question similarity project. We have started working on this project
from 13th of May only. Working with team gives little extra space to apply
new things.
--------------------------------------------------------------------------
----------------------------------------------
Sentence Tokens: ['Our Team name is Team Data Dynamos and we have selected
Quora question similarity project.', 'We have started working on this proj
ect from 13th of May only.', 'Working with team gives little extra space t
o apply new things.']
```

b. Word Tokenization

In [ ]:

```
sentence='''Our Team name is Team Data Dynamos and we have selected Quora question simila
tokens=sentence.split(" ")
print("Original Sentence:", sentence)
print("--"*60)
print("Word Tokens:", tokens)
```

```
Original Sentence: Our Team name is Team Data Dynamos and we have selected
Quora question similarity project.?
--------------------------------------------------------------------------
----------------------------------------------
Word Tokens: ['Our', 'Team', 'name', 'is', 'Team', 'Data', 'Dynamos', 'an
d', 'we', 'have', 'selected', 'Quora', 'question', 'similarity', 'projec
t.?']
```

In [ ]:

```
from nltk.tokenize import word_tokenize
sentence='''Our Team name is Team Data Dynamos and we have selected Quora question simila
tokens=word_tokenize(sentence)
print("Original Sentence:", sentence)
print("--"*60)
print("Word Tokens:", tokens)
```

```
Original Sentence: Our Team name is Team Data Dynamos and we have selected
Quora question similarity project.?
--------------------------------------------------------------------------
----------------------------------------------
Word Tokens: ['Our', 'Team', 'name', 'is', 'Team', 'Data', 'Dynamos', 'an
d', 'we', 'have', 'selected', 'Quora', 'question', 'similarity', 'projec
t', '.', '?']
```

We can easily see the difference, when we tokenize using string method, it will consider all the special characters & punctuation attached to a word as a part of that word, but when we tokenize using NLTK word_tokenizer it consider those special characters & punctuation as a seperate toke.

c. Sub-Word(n-gram character) Tokenization

N-grams are continuous sequences of words or symbols, or tokens in a document. In technical terms, they can be defined as the neighboring sequences of items in a document.

In [ ]:

```
from nltk import ngrams
```

In [ ]:

```
sentence='''Our Team name is Team Data Dynamos and we have selected Quora question simila
n_gram_tokens=list(ngrams((sentence.split(" ")), n=3))
print("Original Sentence:", sentence)
print("--"*60)
print("N-gram Tokens:", n_gram_tokens)
```

```
Original Sentence: Our Team name is Team Data Dynamos and we have selected
Quora question similarity project. We have started working on this project
from 13th of May only. Working with team gives little extra space to apply
new things.
--------------------------------------------------------------------------
----------------------------------------------
N-gram Tokens: [('Our', 'Team', 'name'), ('Team', 'name', 'is'), ('name',
'is', 'Team'), ('is', 'Team', 'Data'), ('Team', 'Data', 'Dynamos'), ('Dat
a', 'Dynamos', 'and'), ('Dynamos', 'and', 'we'), ('and', 'we', 'have'),
('we', 'have', 'selected'), ('have', 'selected', 'Quora'), ('selected', 'Q
uora', 'question'), ('Quora', 'question', 'similarity'), ('question', 'sim
ilarity', 'project.'), ('similarity', 'project.', 'We'), ('project.', 'W
e', 'have'), ('We', 'have', 'started'), ('have', 'started', 'working'),
('started', 'working', 'on'), ('working', 'on', 'this'), ('on', 'this', 'p
roject'), ('this', 'project', 'from'), ('project', 'from', '13th'), ('fro
m', '13th', 'of'), ('13th', 'of', 'May'), ('of', 'May', 'only.'), ('May',
'only.', 'Working'), ('only.', 'Working', 'with'), ('Working', 'with', 'te
am'), ('with', 'team', 'gives'), ('team', 'gives', 'little'), ('gives', 'l
ittle', 'extra'), ('little', 'extra', 'space'), ('extra', 'space', 'to'),
('space', 'to', 'apply'), ('to', 'apply', 'new'), ('apply', 'new', 'thing
s.')]
```

2. Remove Stop Words

In [ ]:

```
from nltk.corpus import stopwords
stopwords_en=stopwords.words("english")
print("Total Stop Words in English=", len(stopwords_en))
```

```
Total Stop Words in English= 179
```

English language contains 179 Stop WOrds.

In [ ]:

```
sentence="Our Team name is Team Data Dynamos and we have selected Quora question similari
sentence_non_stopword=[word for word in sentence.split(" ") if not word in stopwords_en]
print("Sentence with StopWOrds:", sentence)
print("--"*60)
print("Sentence without StopWOrds:", " ".join(sentence_non_stopword))
```

```
Sentence with StopWOrds: Our Team name is Team Data Dynamos and we have se
lected Quora question similarity project
--------------------------------------------------------------------------
----------------------------------------------
Sentence without StopWOrds: Our Team name Team Data Dynamos selected Quora
question similarity project
```

3. Apply Stemming

Types of Stemmer in NLP:

a. Porter Stemmer
b. SnowBall Stemmer
c. Lancaster Stemmer
d. Regexp Stemmer

a. Porter Stemmer

Porter Stemmer is the original stemmer but the stem sometimes illogical or non-dictionary word.

In [ ]:

```
from nltk.stem import PorterStemmer
porter=PorterStemmer()
sentence="Connect Connection Connections Connecting Connected Connects Connectings Drivin
porter_stem=[porter.stem(word) for word in sentence.split(" ")]
print("Original Sentence:", sentence)
print("--"*60)
print("Sentence after Porter Stemming:", " ".join(porter_stem))
```

```
Original Sentence: Connect Connection Connections Connecting Connected Con
nects Connectings Driving Driven Drives Able Ables Enable Enables Enabling
--------------------------------------------------------------------------
----------------------------------------------
Sentence after Porter Stemming: connect connect connect connect connect co
nnect connect drive driven drive abl abl enabl enabl enabl
```

b. Snowball Stemmer

Snowball stemmer is faster and more logical than the Porter Stemmer.

In [ ]:

```
from nltk.stem import SnowballStemmer
snowball=SnowballStemmer(language="english")
sentence="Connect Connection Connections Connecting Connected Connects Connectings Drivin
snowball_stem=[snowball.stem(word) for word in sentence.split(" ")]
print("Original Sentence:", sentence)
print("--"*60)
print("Sentence after Porter Stemming:", " ".join(snowball_stem))
```

```
Original Sentence: Connect Connection Connections Connecting Connected Con
nects Connectings Driving Driven Drives Able Ables Enable Enables Enabling
--------------------------------------------------------------------------
----------------------------------------------
Sentence after Porter Stemming: connect connect connect connect connect co
nnect connect drive driven drive abl abl enabl enabl enabl
```

c. Lancaster Stemmer

The Lancaster stemmers are more aggressive and dynamic. The stemmer is really faster, but the algorithm is really confusing when dealing with small words. Lancaster Stemmer produces results with excessive stemming.

In [ ]:

```
from nltk.stem import LancasterStemmer
lancaster=LancasterStemmer()
sentence="Connect Connection Connections Connecting Connected Connects Connectings Drivin
lancaster_stem=[lancaster.stem(word) for word in sentence.split(" ")]
print("Original Sentence:", sentence)
print("--"*60)
print("Sentence after Porter Stemming:", " ".join(lancaster_stem))
```

```
Original Sentence: Connect Connection Connections Connecting Connected Con
nects Connectings Driving Driven Drives Able Ables Enable Enables Enabling
--------------------------------------------------------------------------
----------------------------------------------
Sentence after Porter Stemming: connect connect connect connect connect co
nnect connect driv driv driv abl abl en en en
```

d. Regexp Stemmer

Regexp stemmer identifies morphological affixes using regular expressions. Substrings matching the regular expressions will be discarded.

In [ ]:

```
from nltk.stem import RegexpStemmer
regex=RegexpStemmer(regexp="ing$|s$|e$", min=0)
sentence="Connect Connection Connections Connecting Connected Connects Connectings Drivin
regex_stem=[regex.stem(word) for word in sentence.split(" ")]
print("Original Sentence:", sentence)
print("--"*60)
print("Sentence after Porter Stemming:", " ".join(regex_stem))
```

```
Original Sentence: Connect Connection Connections Connecting Connected Con
nects Connectings Driving Driven Drives Able Ables Enable Enables Enabling
--------------------------------------------------------------------------
----------------------------------------------
Sentence after Porter Stemming: Connect Connection Connection Connect Conn
ected Connect Connecting Driv Driven Drive Abl Able Enabl Enable Enabl
```

All Stemmers are Different from each other. Ther is one common thing between all stemmers, sometimes they did not return the stem with logical or dictionary meaning.

4. Apply Lemmatization

Types of Lemmatization in NLP:

a. Wordnet Lemmatizer
b. TextBlob Lemmatizer

a. Wordnet Lemmatizer

In [ ]:

```
from nltk.stem import WordNetLemmatizer
lemma=WordNetLemmatizer()
sentence="The bats are hanging on their feet in upright positions"
sentence_lemma=[lemma.lemmatize(word, 'v') for word in sentence.split(" ")]
print("Original Sentence:", sentence)
print("--"*60)
print("Sentence after Lemmatization:", " ".join(sentence_lemma))
```

```
Original Sentence: The bats are hanging on their feet in upright positions
--------------------------------------------------------------------------
----------------------------------------------
Sentence after Lemmatization: The bat be hang on their feet in upright pos
ition
```

b. TextBlob Lemmatizer

In [ ]:

```
from textblob import TextBlob, Word
sentence="The bats are hanging on their feet in upright positions"
sent=TextBlob(sentence)
texblob_lemma=[w.lemmatize() for w in sent.words]
print("Original Sentence:", sentence)
print("--"*60)
print("Sentence after Lemmatization:", " ".join(texblob_lemma))
```

```
Original Sentence: The bats are hanging on their feet in upright positions
--------------------------------------------------------------------------
----------------------------------------------
Sentence after Lemmatization: The bat are hanging on their foot in upright
position
```

C. More Advanced Techniques

These Techniques are not used in all the tasks, these are problem specific. These techniques are mainly used in QA System(Question Answer), Word Sense Disambiguiation etc.

1. POS Tagging

Adding a Part of Speech tags to every word in the corpus is called POS tagging. If we want to perform POS tagging then no need to remove stopwords. This is one of the essential steps in the text analysis where we know the sentence structure and which word is connected to the other, which word is rooted from which, eventually, to figure out hidden connections between words which can later boost the performance of our Machine Learning Model.

POS Tagging can be performed using two Libraries

a. POS Tagging using NLTK
b. POS Tagging using Spacy

a. POS Tagging using NLTK

In [ ]:

```
from nltk.tag import pos_tag
from nltk.tokenize import word_tokenize
doc=word_tokenize("What is the step by step guide to invest in share market in india")
for i in range(len(doc)):
    print("Word:",pos_tag(doc)[i][0], "||", "POS Tag:", pos_tag(doc)[i][1])
```

```
Word: What || POS Tag: WP
Word: is || POS Tag: VBZ
Word: the || POS Tag: DT
Word: step || POS Tag: NN
Word: by || POS Tag: IN
Word: step || POS Tag: NN
Word: guide || POS Tag: RB
Word: to || POS Tag: TO
Word: invest || POS Tag: VB
Word: in || POS Tag: IN
Word: share || POS Tag: NN
Word: market || POS Tag: NN
Word: in || POS Tag: IN
Word: india || POS Tag: NN
```

b. POS Tagging using Spacy

In [ ]:

```
import spacy
```

In [ ]:

```
nlp=spacy.load("en_core_web_sm")
doc=nlp("What is the step by step guide to invest in share market in india")
for word in doc:
    print("Word:", word.text,"||","POS:", word.pos_, "||", "POS Tag:", word.tag_, "||", "
```

```
Word: What || POS: PRON || POS Tag: WP || Explanation: wh-pronoun, persona
l
Word: is || POS: AUX || POS Tag: VBZ || Explanation: verb, 3rd person sing
ular present
Word: the || POS: DET || POS Tag: DT || Explanation: determiner
Word: step || POS: NOUN || POS Tag: NN || Explanation: noun, singular or m
ass
Word: by || POS: ADP || POS Tag: IN || Explanation: conjunction, subordina
ting or preposition
Word: step || POS: NOUN || POS Tag: NN || Explanation: noun, singular or m
ass
Word: guide || POS: NOUN || POS Tag: NN || Explanation: noun, singular or
mass
Word: to || POS: PART || POS Tag: TO || Explanation: infinitival "to"
Word: invest || POS: VERB || POS Tag: VB || Explanation: verb, base form
Word: in || POS: ADP || POS Tag: IN || Explanation: conjunction, subordina
ting or preposition
Word: share || POS: NOUN || POS Tag: NN || Explanation: noun, singular or
mass
Word: market || POS: NOUN || POS Tag: NN || Explanation: noun, singular or
mass
Word: in || POS: ADP || POS Tag: IN || Explanation: conjunction, subordina
ting or preposition
Word: india || POS: PROPN || POS Tag: NNP || Explanation: noun, proper sin
gular
```

Spacy is more powerful than NLTK. Spacy is faster and Grammatically accurate.

2. NER Tagging

Named entity recognition (NER) is a natural language processing (NLP) method that extracts information from text. NER involves detecting and categorizing important information in text known as named entities. Named entities refer to the key subjects of a piece of text, such as names, locations, companies, events and products, as well as themes, topics, times, monetary values and percentages.
NER can be performed using two Libraries
a. NER using NLTK
b. NER using Spacy

a. NER using NLTK

In [ ]:

```
import nltk
stopwords_en=stopwords.words("english")
```

In [ ]:

```
sentence="TATA and Mahindra are the top companies in India. But the 'Gautam Adani' and 'M
words=[word for word in sentence.split(" ") if word not in stopwords_en]
tagged_tokens=nltk.pos_tag(words)
entities=nltk.ne_chunk(tagged_tokens)
for entity in entities:
    print(entity)
```

```
(ORGANIZATION TATA/NNP Mahindra/NNP)
('top', 'JJ')
('companies', 'NNS')
('India.', 'NNP')
('But', 'CC')
("'Gautam", 'NNP')
("Adani'", 'NNP')
("'Mukesh", 'POS')
("Ambani'", 'NNP')
('reachest', 'NN')
('person.', 'NN')
```

b. NER using Spacy

In [ ]:

```
nlp = spacy.load("en_core_web_sm")
sentence="TATA and Mahindra are the top companies in India. But the 'Gautam Adani' and 'M
doc = nlp(sentence)
for entity in doc.ents:
    print(entity.text, entity.label_)
```

```
TATA ORG
Mahindra ORG
India GPE
Gautam Adani' PERSON
Mukesh Ambani' PERSON
```

Spacy is a faster and more efficient library for NER. It provides a pre-trained NER model that is highly accurate than NLTK and can recognize a wide range of named entities. Additionally, SpaCy has more advanced features such as named entity linking and coreference resolution.

* Text to Numerical Vector Conversion Techniques *

Our Machine Learning and Deep Learning models take only numerical data as an input to train the model and do prediction, Hence it is necessary to perform conversion step to make texual data into equivalent numerical representation. There are many text to numerical vector conversion techniques, these techniques are,

1. BOW(Bag Of Word): Count Vectorizer
2. TF-IDF(Term Frequence-Inverse Document Frequency)
3. Word2Vec(Word to Vector)
4. GloVe(Global Vector)
5. BERT(Bidirectional Encoder Representations from Transformers)

1. Bag Of Word(Count Vectorizer)

It is a Collection of words represent a sentence with word count. Steps invloved in this process are Clean Text, Tookenize, Build Vocabulary and Generate Vecors. We can create vocabulory of size 1 to n using uni-ngram, bi-gram, n-gram.

Advantages:

a. Simple Procedure and easy to implement.
b. Easy to Understand

Disadvantages:

a. Does not consider the symmentic meaning of the word.
b. Due to large vector size computational time is high.
c. Count Vectorizer Generates Spars matrix.
d. Out of Vocabulary words are not captured.

2. TF-IDF(Term Frequence-Inverse Document Frequency)

It is a Statistical method. It measures how important a term or word is within a document or setence relative to a collection of documents or Corpus. Words within a text document are transformed into importance numbers by a text vectorization process.

Advantages:

a. Simple Procedure and easy to implement.

b. Easy to Understand

c. Here unlike BOW, weightage for those words is given high if that word occuring in that document but occuring less in corpus.

Disadvantages:

a. Does not consider the symmentic meaning of the word

3. Word2Vec(Word to Vector)

It is a pre-trained word embedded model. Word2Vec creates vectors of the words that are distributed numerical representations of word features. These word features represents the context for the each words present in vocabulary. Two different model architectures that can be used by Word2Vec to create the word embeddings are the Continuous Bag of Words (CBOW) model(Used when dataset is small) & the Skip-Gram model(Used when the dataset is large).

Advantages:

a. Word embeddings eventually help in establishing the association of a word with another similar meaning word through the created vectors.

b. Captures symmantic meaning.

c. Low Dimensional vectors hence the computational time reduces.

d. Dense vectors.

Disadvantages:

a. Contexual meaning only captured within the window size. or in other word it has local context scope.

b. Not able to generate vectors for unseen words.

4. GloVe(Global Vector)

It is also a Pre-trained word embedding technique used to overcome drawback of Word2Vec.

Advantages:

a. Contexual meaning captured for both local and global scope.

b. It uses co-occurance matrix to tell us how often two words occuring together.

c. Captures symmantic meaning.
d. Low Dimensional vectors hence the computational time reduces.
e. Dense vectors.

Disadvantages:

a. Utilizes massive memory and takes time to load.

5. BERT(Bidirectional Encoder Representations from Transformers)

BERT is the Pre-trained birectional trasformer for Language understanding. It has trained on 2500M Wikipedia words and 800M+ Books words. And BERT used by Google search Engine. BERT uses the encoder part of the Transformer, since it’s goal is to create a model that performs a number of different NLP tasks.

Advantages:

a. Contexual meaning captured for both local and global scope.
b. Captures symmantic meaning.
c. Powerful than all previous wod embedding techniques.

Disadvantages:

a. Utilizes massive memory and takes time to load and train.

There are manay techniques used in NLP, I just listed few basic fundamental steps.

```
In [ ]:
```