ASSIGNMENT 1

Instructions:

- Please download the provided IPython Notebook (ipynb) file and open it in Google Colab. Once opened, enter your code in the same file directly beneath the relevant question's code block.
- Insert a text block below your code to briefly explain it, mentioning any libraries or functions utilized. Answer the questions in brief with examples.
- Submit

1. The IPython Notebook (ipynb) file.
2. A PDF version of the notebook (converted from ipynb).

- The similarity score should be less than 15%

Task 1: Tokenization (25%)

(refer to the spacy tokenization concept which is explained after Question1 in activity-1)

Question -1:

How can we incorporate contextual information beyond individual words into the tokenization process to improve performance on downstream tasks like machine translation or question answering?

Answer Here: To do this we need a smarter way to look at words instead of just breaking them into individual words. There are different ways of doing it, some of which are as follows.

1. Byte Pair Encoding (BPE): This method looks at a pair of letters that occur more frequently, combines them, and does this iteratively until a fixed size is reached. It is used in tasks such as text classification, text generation, and machine translation.
2. Subword Tokenization: The words are divided into small parts which help to understand words better.
3. WordPiece Tokenization: It is similar to BPE, but here it looks at the whole word at a time.
4. SentencePiece Tokenization: In this method, the sentence is broken down into small parts which will help in understanding the grammar, etc.

Question 2

Develop a tokenizer that considers contractions like "can't" and "doesn't", splitting them into their constituent words.

In [34]:

```python
import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt')

text = "I can't believe it's not butter! You're going to love it."
#CODE HERE
# Custom contractions
custom_contractions = {
    "can't": "can not",
    "won't": "will not",
    "shan't": "shall not",
    "n't": "not",
    "'s": " is",
    "'re": " are",
    "'ve": " have",
    "'d": " would",
    "'ll": " will",
}

# Custom tokenizer function
def custom_tokenize(text):
    tokens = word_tokenize(text)
    tokens = [custom_contractions.get(token.lower(), token) for token in tokens]
    return tokens

# Tokenizing the text
tokens = custom_tokenize(text)
print(tokens)
```

```
['I', 'ca', 'not', 'believe', 'it', ' is', 'not', 'butter', '!', 'You', ' are', 'going',
'to', 'love', 'it', '.']
```

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

- **Import nltk library for word tokenization.**
- **Download Punkt from nltk.**
- **Add custom mappings.**
- **Create a function that tokenizes the text while considering the custom mappings as well.**
- **Finally, tokenize the sentence.**

Question 3:

Implement a Python script to remove Twitter username handles from a given twitter text.

Method-1

In [35]:

```
text = "Great meeting with @JohnDoe and @JaneSmith today! Looking forward to our next pro
ject. Thanks for the insights @TechGuru □ #innovation #teamwork"

words = text.split()
cleaned_words = []
for word in words:
    if word.startswith("@"):
        continue
    cleaned_words.append(word)

cleaned_text = " ".join(cleaned_words)

print(cleaned_text)
```

```
Great meeting with and today! Looking forward to our next project. Thanks for the insight
s □ #innovation #teamwork
```

- **The above code removes the words that start with '@' because they are Twitter usernames.**
- **The above code does not use regular expressions.**

Method-2

In [36]:

```
text = "Great meeting with @JohnDoe and @JaneSmith today! Looking forward to our next pro
ject. Thanks for the insights @TechGuru □ #innovation #teamwork"
```

```
#CODE HERE

words = text.split()
cleaned_words = []
for word in words:
    if word.startswith("@"):
        word = word[1:]
    cleaned_words.append(word)

cleaned_text = " ".join(cleaned_words)

print(cleaned_text)
```

```
Great meeting with JohnDoe and JaneSmith today! Looking forward to our next project. Than
ks for the insights TechGuru  #innovation #teamwork
```

- **The above code removes the username handle '@' and replaces it with only the name.**
- **This code does not use regular expressions.**

- **Create a regular expression that identifies the Twitter username.**
- **Then remove it from the sentence.**

Task 2. - Regular Expressions (25%)

Regular Expressions

A regular expression, often abbreviated as regex, is a powerful and flexible tool for pattern matching and text manipulation. It consists of a sequence of characters that defines a search pattern, allowing you to perform various text-related tasks such as text validation, data extraction, text cleaning, and more. Regular expressions are used in programming languages and text editors and are constructed using a combination of regular characters, special characters, and metacharacters to specify search criteria. Learning to use regular expressions effectively can greatly enhance text processing tasks, making them a valuable skill in fields like natural language processing, data extraction, and data validation.

Python includes a builtin module called `re` which provides regular expression matching operations (Click here for the official module documentation). Once the module is imported into your code, you can use all of the available capabilities for performing pattern-based matching or searching using regular expressions.

In [37]:

```
##code block -1
import re

def apply_regex(data, pattern):
  for text in data:
    if re.fullmatch(pattern, text):
      print(f"Test string {text} accepted.")
    else:
      print(f"Test string {text} failed!")
```

Question - 1

Same as previous question Implement a Python script to remove Twitter username handles from a given twitter text with Regular Expression

In [38]:

```
#CODE HERE
import re
```

```
text = "Great meeting with @JohnDoe and @JaneSmith today! Looking forward to our next pro
ject. Thanks for the insights @TechGuru  #innovation #teamwork"
#CODE HERE

def remove_username_handles(text):
  return re.sub(r'@\w+', '', text)

print(remove_username_handles(text))
```

```
Great meeting with  and  today! Looking forward to our next project. Thanks for the insig
hts   #innovation #teamwork
```

- **The above code removes the word that starts with '@' because they are Twitter usernames.**
- **It uses regular expressions to do that.**
- **The regular expression "@\w+" means '@' followed by one or more characters.**

Question- 2

Implement a Python program to find URLs in the given string.

In [39]:

```
from bs4 import BeautifulSoup

text= '<p>Contents :</p><a href="https://w3resource.com">Python Examples</a><a href="http
://github.com">Even More Examples</a><a href="https://openai.com">OpenAI Homepage</a><a h
ref="https://docs.python.org">Python Documentation</a>'
##Your code here
soup = BeautifulSoup(text, 'html.parser')

links = []
for link in soup.find_all('a'):
    links.append(link.get('href'))

print(links)
```

```
['https://w3resource.com', 'http://github.com', 'https://openai.com', 'https://docs.pytho
n.org']
```

- **Import BeautifulSoap**
- **Parse the HTML String**
- **Find all anchor tags '< a >'.**
- **Extract the href attribute from each link**
- **Append to a links list**
- **Print the list of links**

Using regular expressions based pattern matching on real world text

For the purposes of demonstration, here's a dummy paragraph of text. A few observations here:

- **The text has multiple paragraphs with each paragraph having more than one sentence.**
- **Some of the words are capitalized (first letter is in uppercase followed by lowercase letters).**

In [40]:

```
text = """Here is the First Paragraph and this is the First Sentence. here is the Second
Sentence. now is the Third Sentence. this is the Fourth Sentence of the first paragaraph.
this paragraph is ending now with a Fifth Sentence.
Now, it is the Second Paragraph and its First Sentence. here is the Second Sentence. now
is the Third Sentence. this is the Fourth Sentence of the second paragraph. this paragrap
h is ending now with a Fifth Sentence.
```

```
Finally, this is the Third Paragraph and is the First Sentence of this paragraph. here is
the Second Sentence. now is the Third Sentence. this is the Fourth Sentence of the third
paragagraph. this paragraph is ending now with a Fifth Sentence.
4th paragraph is not going to be detected by either of the regex patterns below.
"""

print(text)
```

```
Here is the First Paragraph and this is the First Sentence. here is the Second Sentence.
now is the Third Sentence. this is the Fourth Sentence of the first paragaraph. this para
graph is ending now with a Fifth Sentence.
Now, it is the Second Paragraph and its First Sentence. here is the Second Sentence. now
is the Third Sentence. this is the Fourth Sentence of the second paragraph. this paragrap
h is ending now with a Fifth Sentence.
Finally, this is the Third Paragraph and is the First Sentence of this paragraph. here is
the Second Sentence. now is the Third Sentence. this is the Fourth Sentence of the third
paragagraph. this paragraph is ending now with a Fifth Sentence.
4th paragraph is not going to be detected by either of the regex patterns below.
```

The following code block shows a regular expression that matches only those strings that:

1. **are at the start of a line and**
2. **the string does not start with a number or a whitespace**

`re.findall()` **finds all matches of the pattern in the text under consideration. The output is a list of strings that matched.**

Further, the regular expression defined below matches the words that are capitalized.

In [41]:

```
##code block - 3
re_pattern2 = r'[A-Z][a-z]+'
print(re.findall(re_pattern2, text))
```

```
['Here', 'First', 'Paragraph', 'First', 'Sentence', 'Second', 'Sentence', 'Third', 'Sente
nce', 'Fourth', 'Sentence', 'Fifth', 'Sentence', 'Now', 'Second', 'Paragraph', 'First', '
Sentence', 'Second', 'Sentence', 'Third', 'Sentence', 'Fourth', 'Sentence', 'Fifth', 'Sen
tence', 'Finally', 'Third', 'Paragraph', 'First', 'Sentence', 'Second', 'Sentence', 'Thir
d', 'Sentence', 'Fourth', 'Sentence', 'Fifth', 'Sentence']
```

Following is a text excerpt on "Inaugural Address" taken from the website of the Joint Congressional Committee on Inaugural Ceremonies:

In [42]:

```
inau_text="""The custom of delivering an address on Inauguration Day started with the ver
y first Inauguration—George Washington’s—on April 30, 1789(04-30-1789). ex:-18.5. After t
aking his oath of office on the balcony of Federal Hall in New York City, Washington proc
eeded to the Senate chamber where he read a speech before members of Congress and other d
ignitaries. His second Inauguration took place in Philadelphia on March 4, 1793(03/04/179
3), in the Senate chamber of Congress Hall. There, Washington gave the shortest Inaugural
address on record—just 135 words —before repeating the oath of office.
Every President since Washington has delivered an Inaugural address. While many of the ea
rly Presidents read their addresses before taking the oath, current custom dictates that
the Chief Justice of the Supreme Court administer the oath first, followed by the Preside
nt’s speech.
William Henry Harrison delivered the longest Inaugural address, at 8,445 words, on March
4, 1841—a bitterly cold, wet day. He died one month later of pneumonia, believed to have
been brought on by prolonged exposure to the elements on his Inauguration Day. John Adams
’ Inaugural address, which totaled 2,308 words, contained the longest sentence, at 737 wo
rds. After Washington’s second Inaugural address, the next shortest was Franklin D. Roose
velt’s fourth address on January 20, 1945(01-20-1945), at just 559.0 words. Roosevelt had
chosen to have a simple Inauguration at the White House in light of the nation’s involvem
ent in World War II.
In 1921, Warren G. Harding became the first President to take his oath and deliver his In
augural address through loud speakers. In 1925, Calvin Coolidge’s Inaugural address was t
he first to be broadcast nationally by radio. And in 1949, Harry S. Truman became the fir
```

```
st President to deliver his Inaugural address over television airwaves.
Most Presidents use their Inaugural address to present their vision of America and to set
forth their goals for the nation. Some of the most eloquent and powerful speeches are sti
ll quoted today. In 1865, in the waning days of the Civil War, Abraham Lincoln stated, “W
ith malice toward none, with charity for all, with firmness in the right as God gives us
to see the right, let us strive on to finish the work we are in, to bind up the nation’s
wounds, to care for him who shall have borne the battle and for his widow and his orphan,
to do all which may achieve and cherish a just and lasting peace among ourselves and with
all nations.” In 1933, Franklin D. Roosevelt avowed, “we have nothing to fear but fear it
self.” And in 1961, John F. Kennedy declared, “And so my fellow Americans: ask not what y
our country can do for you—ask what you can do for your country.”
Today, Presidents deliver their Inaugural address on the West Front of the Capitol, but t
his has not always been the case. Until Andrew Jackson’s first Inauguration in 1829, most
Presidents spoke in either the House or Senate chambers. Jackson became the first Preside
nt to take his oath of office and deliver his address on the East Front Portico of the U.
S. Capitol in 1829. With few exceptions, the next 37.0 Inaugurations took place there, un
til 1981, when Ronald Reagan’s Swearing-In Ceremony and Inaugural address occurred on the
West Front Terrace of the Capitol. The West Front has been used ever since. You should al
so need to extract the floating numbers such as -55.5, 20.8%, -3.0 using your regular exp
ression"""
```

Refer to above code block -3 for the following questions

Questions-3.A

Identify all the positive and neagtive numbers with type of both intergers,float in the "Inaugural Address" excerpt and write a regular expression that finds all occurrences of such words in the text. Then, run the Python code snippet to automatically display the matched strings according to the pattern.*.

NOTE: You can use the *re.findall()* method as demonstrated in the example before this exercise.

In [43]:

```
re_pattern = r'(-?\d*\.?\d+%?)'
print(re.findall(re_pattern, inau_text))
```

```
['30', '1789', '04', '-30', '-1789', '-18.5', '4', '1793', '03', '04', '1793', '135', '8'
, '445', '4', '1841', '2', '308', '737', '20', '1945', '01', '-20', '-1945', '559.0', '19
21', '1925', '1949', '1865', '1933', '1961', '1829', '1829', '37.0', '1981', '-55.5', '20.
8%', '-3.0']
```

- **re_pattern: Regular expression pattern to match positive and negative numbers (including integers and floats)**
- **Then find and print all occurrences of positive, negative and float point numbers in the text.**

Your explanation

- **The regular expression "(-?\d*.?\d+%?)" represents all the positive, negative and float numbers.**
- **'-?' means '-' is optional.**
- **'\d*' followed by 0 or more digits.**
- **'.?' means '.' is optional.**
- **'\d+' followed by 1 or more digits.**
- **And at last, '%?' means '%' is also optional.**

Question-3.B

Identify all the dates of all forms - text form(April 20, 1945) and digit form(xx-xx-xxxx, xx/xx/xxxx) in the "Inaugural Address" excerpt and

write a regular expression that finds all occurrences of the dates in the text. Then, run the Python code snippet to automatically display a list of all such dates identified.

NOTE: You can use the *re.findall()* method as demonstrated in the example before this exercise.

Method-1

```
In [44]:
```

```
##Your code here
pattern = r'\b(\w+ \d{1,2}, \d{4}|\d{1,2}/\d{1,2}/\d{4}|\d{1,2}-\d{1,2}-\d{4})\b'
print(re.findall(pattern, inau_text))
```

```
['April 30, 1789', '04-30-1789', 'March 4, 1793', '03/04/1793', 'March 4, 1841', 'January
20, 1945', '01-20-1945']
```

Method-2

```
In [45]:
```

```
re_pattern = r'(?:January|February|March|April|May|June|July|August|September|October|Nov
ember|December)\s+\d{1,2},\s+\d{4}|(?:\d{1,2}[-/]\d{1,2}[-/]\d{4})'
print(re.findall(re_pattern, inau_text))
```

```
['April 30, 1789', '04-30-1789', 'March 4, 1793', '03/04/1793', 'March 4, 1841', 'January
20, 1945', '01-20-1945']
```

- **The above regular expression finds the dates in all formats.**
- **Then print and find all the occurrences of the regular expression in the text.**

Your explanation

- **We can write 2 regular expressions for date formats.**
- **The first regular expression is a generic regular expression that does not care about the spelling of the month. Whereas the second regular expression checks for the spelling as well, which is a more specific regular expression.**

Task 3: Lemmatization/Stemming(25%)

Question -1:

How does the morphology of a language (e.g., agglutinative vs. fusional) impact the suitability of stemming vs. lemmatization?

Your answer here

- **The structure of a word in the language is known as the morphology of a language. The morphology of a language has a great impact on the suitability of stemming and lemmatization.**
- **Stemming: It is adding or removing prefixes and suffixes. Stemming does not require one to understand the language, instead, it follows a set of pre-defined rules.**
- **Lemmatization: It involves identifying a lemma that requires a proper understanding of the language.**

Agglutinative languages

- **Agglutinative languages like Turkish and Finnish forms words by combining morphemes, which are small**

- Agglutinative languages like Turkish and Finnish forms words by combining morphemes, which are small units of meanings. For example, the Turkish word "evlerinizdekilerden" (meaning "from those at your house") is built from smaller parts like "ev" (house), "ler" (plural marker), "iniz" (your).
- When considering agglutinate language stemming is more effective when compared with lemmatization because words are assembled from distinct morphenes.

Fusional languages

- Fusional languages like French and Spanish often change the morphemes and the form which requires language understanding to understand it.
- For example, the French verb "parler" (to speak) becomes "je parle" (I speak) by merging the pronoun and tense.
- So, in case fusional language stemming will not work as effectively as lemmatization will.
- Lemmatization uses more complex ways to map the words to its base word.

Question - 2 :

Create a Python function that takes a sentence as input, performs lemmatization using StanfordNLP, and removes stopwords from the lemmatized sentence. Use a list of stopwords. Return the cleaned and lemmatized sentence.

Reference:https://stanfordnlp.github.io/stanfordnlp/

In [46]:

```
!pip install stanza
```

```
Requirement already satisfied: stanza in /usr/local/lib/python3.10/dist-packages (1.7.0)
Requirement already satisfied: emoji in /usr/local/lib/python3.10/dist-packages (from sta
nza) (2.10.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from sta
nza) (1.25.2)
Requirement already satisfied: protobuf>=3.15.0 in /usr/local/lib/python3.10/dist-package
s (from stanza) (3.20.3)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from
stanza) (2.31.0)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from
stanza) (3.2.1)
Requirement already satisfied: toml in /usr/local/lib/python3.10/dist-packages (from stan
za) (0.10.2)
Requirement already satisfied: torch>=1.3.0 in /usr/local/lib/python3.10/dist-packages (f
rom stanza) (2.1.0+cu121)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from stan
za) (4.66.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from
torch>=1.3.0->stanza) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packag
es (from torch>=1.3.0->stanza) (4.9.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from tor
ch>=1.3.0->stanza) (1.12)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from to
rch>=1.3.0->stanza) (3.1.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from to
rch>=1.3.0->stanza) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (
from torch>=1.3.0->stanza) (2.1.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist
-packages (from requests->stanza) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (f
rom requests->stanza) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packa
ges (from requests->stanza) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packa
ges (from requests->stanza) (2024.2.2)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: markupsafe>=2.0 in /usr/local/lib/python3.10/dist-packages
(from jinja2->torch>=1.3.0->stanza) (2.1.5)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (f
rom sympy->torch>=1.3.0->stanza) (1.3.0)
```

- **Import Stanza**

In [47]:

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

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

Out[47]:

```
True
```

- **Download all the stopwords**

In [48]:

```
import stanza
import string
from nltk.corpus import stopwords

stopwords = stopwords.words('english')

def lemmatize_and_remove_stopwords(sentence):
  nlp = stanza.Pipeline(lang='en', processors='tokenize,mwt,pos,lemma')
  doc = nlp(sentence)

  lemmatized = [word.lemma for sent in doc.sentences for word in sent.words]

  cleaned = [word for word in lemmatized if word not in stopwords]

  return " ".join(cleaned)
```

- **Create a methods to remove stop words and lemmatization.**

In [49]:

```
sentence="""The custom of delivering an address on Inauguration Day started with the very
first Inauguration—George Washington’s—on April 30, 1789(04-30-1789). ex:-18.5. After tak
ing his oath of office on the balcony of Federal Hall in New York City, Washington procee
ded to the Senate chamber where he read a speech before members of Congress and other dig
nitaries. His second Inauguration took place in Philadelphia on March 4, 1793(03/04/1793)
, in the Senate chamber of Congress Hall. There, Washington gave the shortest Inaugural a
ddress on record—just 135 words —before repeating the oath of office.
Every President since Washington has delivered an Inaugural address. While many of the ea
rly Presidents read their addresses before taking the oath, current custom dictates that
the Chief Justice of the Supreme Court administer the oath first, followed by the Preside
nt’s speech.
William Henry Harrison delivered the longest Inaugural address, at 8,445 words, on March
4, 1841—a bitterly cold, wet day. He died one month later of pneumonia, believed to have
been brought on by prolonged exposure to the elements on his Inauguration Day. John Adams
’ Inaugural address, which totaled 2,308 words, contained the longest sentence, at 737 wo
rds. After Washington’s second Inaugural address, the next shortest was Franklin D. Roose
velt’s fourth address on January 20, 1945(01-20-1945), at just 559.0 words. Roosevelt had
chosen to have a simple Inauguration at the White House in light of the nation’s involvem
ent in World War II.
In 1921, Warren G. Harding became the first President to take his oath and deliver his In
augural address through loud speakers. In 1925, Calvin Coolidge’s Inaugural address was t
he first to be broadcast nationally by radio. And in 1949, Harry S. Truman became the fir
st President to deliver his Inaugural address over television airwaves.
Most Presidents use their Inaugural address to present their vision of America and to set
forth their goals for the nation. Some of the most eloquent and powerful speeches are sti
ll quoted today. In 1865, in the waning days of the Civil War, Abraham Lincoln stated, “W
```

```
ith malice toward none, with charity for all, with firmness in the right as God gives us
to see the right, let us strive on to finish the work we are in, to bind up the nation’s
wounds, to care for him who shall have borne the battle and for his widow and his orphan,
to do all which may achieve and cherish a just and lasting peace among ourselves and with
all nations.” In 1933, Franklin D. Roosevelt avowed, “we have nothing to fear but fear it
self.” And in 1961, John F. Kennedy declared, “And so my fellow Americans: ask not what y
our country can do for you—ask what you can do for your country.”
Today, Presidents deliver their Inaugural address on the West Front of the Capitol, but t
his has not always been the case. Until Andrew Jackson’s first Inauguration in 1829, most
Presidents spoke in either the House or Senate chambers. Jackson became the first Preside
nt to take his oath of office and deliver his address on the East Front Portico of the U.
S. Capitol in 1829. With few exceptions, the next 37.0 Inaugurations took place there, un
til 1981, when Ronald Reagan’s Swearing-In Ceremony and Inaugural address occurred on the
West Front Terrace of the Capitol. The West Front has been used ever since. You should al
so need to extract the floating numbers such as -55.5, 20.8%, -3.0 using your regular exp
ression"""
print(lemmatize_and_remove_stopwords(sentence))
```

```
INFO:stanza:Checking for updates to resources.json in case models have been updated.  Not
e: this behavior can be turned off with download_method=None or download_method=DownloadM
ethod.REUSE_RESOURCES
```

```
INFO:stanza:Loading these models for language: en (English):
=================================
| Processor | Package           |
---------------------------------
| tokenize  | combined          |
| mwt       | combined          |
| pos       | combined_charlm   |
| lemma     | combined_nocharlm |
=================================

INFO:stanza:Using device: cpu
INFO:stanza:Loading: tokenize
INFO:stanza:Loading: mwt
INFO:stanza:Loading: pos
INFO:stanza:Loading: lemma
INFO:stanza:Done loading processors!
```

```
custom deliver address Inauguration Day start first Inauguration — George Washington 's —
April 30 , 179( 04-30-1789 ) . ex :- 18.5 . take oath office balcony Federal Hall New Yor
k City , Washington proceed Senate chamber read speech member Congress dignitary . second
Inauguration take place Philadelphia March 4 , 1793 ( 03/04/1793 ) , Senate chamber Congr
ess Hall . , Washington give short inaugural address record — 135 word — repeat oath offi
ce . every President since Washington deliver inaugural address . many early president re
ad address take oath , current custom dictate Chief Justice Supreme Court administer oath
first , follow president 's speech . William Henry Harrison deliver long inaugural addres
s , 8,445 word , March 4 , 1841 — bitterly cold , wet day . die one month late pneumonia
, believe bring prolonged exposure element Inauguration Day . John Adams 's inaugural add
ress , total 2308 word , contain long sentence , 737 word . Washington 's second inaugura
l address , next short Franklin D. Roosevelt 's fourth address January 20 , 1945 ( 01-20
- 1945 ) , 559.0 word . Roosevelt choose simple inauguration White House light nation 's
involvement World War II . 1921 , Warren G. Harding become first President take oath deli
ver inaugural address loud speaker . 1925 , Calvin Coolidge 's inaugural address first br
oadcast nationally radio . 1949 , Harry S. Truman become first President deliver inaugura
l address television airwave . president use inaugural address present vision America set
forth goal nation . eloquent powerful speech still quote today . 1865 , wane day Civil Wa
r , Abraham Lincoln state , '' malice toward none , charity , firmness right God give see
right , let strive finish work , bind nation 's wound , care shall borne battle widow orp
han , may achieve cherish lasting peace among nation . '' 1933 , Franklin D. Roosevelt av
ow , '' nothing fear fear . '' 1961 , John F. Kennedy declare , '' fellow American : ask
country — ask country . '' today , president deliver inaugural address West Front Capitol
, always case . Andrew Jackson 's first Inauguration 1829 , president speak either House
Senate chamber . Jackson become first President take oath office deliver address East Fro
nt Portico U.S. Capitol 1829 . exception , next 37.0 inauguration take place , 1981 , Ron
ald Reagan 's Swearing - Ceremony inaugural address occur West Front Terrace Capitol . We
st Front use ever since . also need extract float number - 55.5 , 20.8 % , - 3.0 use regu
lar expression
```

- **This is an example sentence.**

Your answer here

- First, install, import and download all the required libraries
- Download the Stanza library and stop words from NLTK.
- Then I wrote a method that does the following tasks
    - Use the Stanza pipeline method to create a pipeline that handles running multiple NLP annotation tasks on text.
    - Lemmatize the words and store it in lemmatization.
    - Remove stopwords and store them in cleaned.
    - Join the lemmatized words into a sentence and return it.
- Finally, I used an example sentence to test the method and the output.

Question - 3

(refer to the byte pair encoding concept which is explained in activity-2 at the Tutorial of Subword Tokenization using HuggingFace after the Question6 in activity-2)

Consider the following two sentences:

S1:I like yellow roses better than red ones.

S2:Looks like John is bettering the working conditions at his organization

Create a Python function that encodes two sentences using the custom BPE tokenizer and identifies common subword tokens (tokens that appear in both encodings). Return a list of these common subword tokens. Is/Are there any interesting observations when you compare the tokens between the two encodings? What do you think is causing what you observe as part of your comparison?

Method-1

In [50]:

```
#code here
from transformers import BertTokenizer

def find_common_subwords(sentence1, sentence2):
    # Initialization of BERT tokenizer
    tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

    # Tokenizing the sentences using the BERT tokenizer
    encoded_sentence1 = tokenizer.tokenize(sentence1)
    encoded_sentence2 = tokenizer.tokenize(sentence2)

    # Finding the common subwords
    common_subwords = set(encoded_sentence1) & set(encoded_sentence2)

    return list(common_subwords)

# Example usage
sentence1 = "I like yellow roses better than red ones."
sentence2 = "Looks like John is bettering the working conditions at his organization"
common_subwords = find_common_subwords(sentence1, sentence2)
print("Common subwords:", common_subwords)
```

```
Common subwords: ['better', 'like']
```

Method-2

In [51]:

```
!pip install tokenizers
#This is a JSON file that contains the vocabulary (i.e., the set of words and subword pie
ces) used by the GPT-2 model
!wget https://s3.amazonaws.com/models.huggingface.co/bert/gpt2-medium-vocab.json
!wget https://s3.amazonaws.com/models.huggingface.co/bert/gpt2-merges.txt
```

```
Requirement already satisfied: tokenizers in /usr/local/lib/python3.10/dist-packages (0.1
5.2)
Requirement already satisfied: huggingface_hub<1.0,>=0.16.4 in /usr/local/lib/python3.10/
dist-packages (from tokenizers) (0.20.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from
huggingface_hub<1.0,>=0.16.4->tokenizers) (3.13.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-package
s (from huggingface_hub<1.0,>=0.16.4->tokenizers) (2023.6.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from
huggingface_hub<1.0,>=0.16.4->tokenizers) (2.31.0)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.10/dist-packages (f
rom huggingface_hub<1.0,>=0.16.4->tokenizers) (4.66.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (fr
om huggingface_hub<1.0,>=0.16.4->tokenizers) (6.0.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/di
st-packages (from huggingface_hub<1.0,>=0.16.4->tokenizers) (4.9.0)
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.10/dist-packages
(from huggingface_hub<1.0,>=0.16.4->tokenizers) (23.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist
-packages (from requests->huggingface_hub<1.0,>=0.16.4->tokenizers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (f
rom requests->huggingface_hub<1.0,>=0.16.4->tokenizers) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packa
ges (from requests->huggingface_hub<1.0,>=0.16.4->tokenizers) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packa
ges (from requests->huggingface_hub<1.0,>=0.16.4->tokenizers) (2024.2.2)
--2024-02-22 17:14:32--  https://s3.amazonaws.com/models.huggingface.co/bert/gpt2-medium-
vocab.json
Resolving s3.amazonaws.com (s3.amazonaws.com)... 52.217.115.248, 52.216.44.96, 52.217.172
.144, ...
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.217.115.248|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1042301 (1018K) [application/json]
Saving to: ‘gpt2-medium-vocab.json.1’

gpt2-medium-vocab.j 100%[===================>]   1018K  2.68MB/s    in 0.4s

2024-02-22 17:14:32 (2.68 MB/s) - ‘gpt2-medium-vocab.json.1’ saved [1042301/1042301]

--2024-02-22 17:14:33--  https://s3.amazonaws.com/models.huggingface.co/bert/gpt2-merges.
txt
Resolving s3.amazonaws.com (s3.amazonaws.com)... 52.217.115.248, 52.216.44.96, 52.217.172
.144, ...
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.217.115.248|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 456318 (446K) [text/plain]
Saving to: ‘gpt2-merges.txt.1’

gpt2-merges.txt.1   100%[===================>] 445.62K  1.47MB/s    in 0.3s

2024-02-22 17:14:33 (1.47 MB/s) - ‘gpt2-merges.txt.1’ saved [456318/456318]
```

- **Install the tokenizer**

In [52]:

```python
from tokenizers import ByteLevelBPETokenizer
import requests

def download_file(url, file_name):
    response = requests.get(url)
    with open(file_name, 'wb') as f:
        f.write(response.content)

def find_common_subwords(sentence1, sentence2):
    # Download vocabulary and merges files
    gpt2vocab_url = "https://s3.amazonaws.com/models.huggingface.co/bert/gpt2-medium-voca
b.json"
    gpt2merges_url = "https://s3.amazonaws.com/models.huggingface.co/bert/gpt2-merges.txt
"
    download_file(gpt2vocab_url, "gpt2-medium-vocab.json")
    download_file(gpt2merges_url, "gpt2-merges.txt")

    # Initialize the tokenizer
    bpe = ByteLevelBPETokenizer("gpt2-medium-vocab.json", "gpt2-merges.txt")

    # Encode the sentences
    encoding1 = bpe.encode(sentence1)
    encoding2 = bpe.encode(sentence2)

    # Find common subword tokens
    common_tokens = set(encoding1.tokens).intersection(encoding2.tokens)

    # Decode common subword tokens and get their corresponding encoded IDs
    decoded_common_tokens = [bpe.decode([bpe.token_to_id(token)]) for token in common_to
kens]
    encoded_common_tokens = [bpe.encode(token).ids[0] for token in decoded_common_tokens
]

    return decoded_common_tokens, encoded_common_tokens

# Example sentences
sentence1 = "I like yellow roses better than red ones."
sentence2 = "Looks like John is bettering the working conditions at his organization"

# Find and print both encoded and decoded words for common subword tokens
decoded_common_subwords, encoded_common_subwords = find_common_subwords(sentence1, senten
ce2)
print("Common Subword Tokens (Encoded):", encoded_common_subwords)
print("Common Subword Tokens (Decoded):", decoded_common_subwords)
```

```
Common Subword Tokens (Encoded): [1365, 588]
Common Subword Tokens (Decoded): [' better', ' like']
```

- **First, download, install and import the files and libraries required.**
- **Define the method/function that extracts the common subwords. The method does the following things.**
    - **Download the vocabulary of huggyface, and merge it.**
    - **Initialize ByteLevelBPETokenizer.**
    - **Encode the given sentences.**
    - **Then finding the common sub-words.**
    - **Finally, decode it.**
- **Print the encoded and decoded common sub-words**

Task - 4 : Minimum Edit distance (25%)

Minimum edit Distance

Minimum Edit Distance (also known as Levenshtein Distance) is a measure of similarity between two strings by calculating the minimum number of single-character edits (insertions, deletions, substitutions) required to transform one string into the other. It has applications in various fields, including natural language processing, spell checking, DNA sequence alignment, and more.

Character Based Text Similarity

"As an example, this technology is used by information retrieval systems, search engines, automatic indexing systems, text summarizers, categorization systems, plagiarism checkers, speech recognition, rating systems, DNA analysis, and profiling algorithms (IR/AI programs to automatically link data between people and what they do)."

In [53]:

```
##code block -4
# A Naive recursive Python program to find minimum number
# operations to convert str1 to str2


def editDistance(str1, str2, m, n):

    # If first string is empty, the only option is to
    # insert all characters of second string into first
    if m == 0:
        return n

    # If second string is empty, the only option is to
    # remove all characters of first string
    if n == 0:
        return m

    # If last characters of two strings are same, nothing
    # much to do. Ignore last characters and get count for
    # remaining strings.
    if str1[m-1] == str2[n-1]:
        return editDistance(str1, str2, m-1, n-1)

    # If last characters are not same, consider all three
    # operations on last character of first string, recursively
    # compute minimum cost for all three operations and take
    # minimum of three values.
    return 1 + min(editDistance(str1, str2, m, n-1),    # Insert
                   editDistance(str1, str2, m-1, n),    # Remove
                   editDistance(str1, str2, m-1, n-1)    # Replace
                   )


# Driver code
str1 = "sunday"
str2 = "saturday"
print (editDistance(str1, str2, len(str1), len(str2)))
```

```
3
```

Refer above code block -4 for the following question

Question-1

Assuming case sensitivity where changing a letter's case has a cost of 1, calculate the minimum cost to transform "Imagine" into "imagination" with the following operation costs: insertions = 2, deletions = 2, substitutions = 3

In [54]:

```
##your code here
# Operation costs
insertions = 2
deletions = 2
substitutions = 3

def editDistance(str1, str2, m, n):

  # If first string is empty, insert all chars of second string
  if m == 0:
    return n * insertions

  # If second string is empty, delete all chars of first string
  if n == 0:
      return m * deletions

  # If last characters are same, ignore them and recur for remaining strings
  if str1[m-1] == str2[n-1]:
    return editDistance(str1, str2, m-1, n-1)

  # Consider all possibilities and take minimum
  return min(
      editDistance(str1, str2, m, n-1) + insertions,
      editDistance(str1, str2, m-1, n) + deletions,
      editDistance(str1, str2, m-1, n-1) + (substitutions if str1[m-1] != str2[n-1] else
0)
  )

str1 = "Imagine"
str2 = "imagination"
print(editDistance(str1, str2, len(str1), len(str2)))
```

```
14
```

Your explanation

- **The edit distance function takes 4 inputs - string 1, its length, string 2, and its length.**
- **Cost of insertion, deletion, and substitution are already fixed and initialized.**
- **If string-1 is empty then insert all the characters to string-2, which means we need to insert all the characters from string-1, so the cost will be the length of string-1 multiplied by the insertion cost.**
- **If string-2 is empty, we need to delete all characters from string-1 which will cost length of string-2 multiplied by deletion cost.**
- **If the last characters are the same, in that case, we can ignore and recursively calculate for the remaining string.**
- **Finally, we need to perform 3 operations on the last character of strig-1 - insert, delete, and substitute.**
- **Then return the minimum of 3 values.**

Example

String-1: Imagine

String-2: imagination

i -> I : Substitution = 3

e -> a : Substitution = 3

t : Insertion = 2

i : Insertion = 2

o : Insertion = 2

n : Insertion = 2

Total = 14

Tutorial -2

Levenshtein Distance for Sentences

In [55]:

```
#code block - 5
# Simple Minimum Edit Distance
def levenshtein_distance(str1, str2):
    # Initialize a matrix to store edit distances
    m, n = len(str1), len(str2)
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    # Initialize the first row and column
    for i in range(m + 1):
        dp[i][0] = i
    for j in range(n + 1):
        dp[0][j] = j

    # Fill in the matrix using dynamic programming
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            cost = 0 if str1[i - 1] == str2[j - 1] else 2  # Substitution cost is 2
            dp[i][j] = min(
                dp[i - 1][j] + 1,  # Deletion
                dp[i][j - 1] + 1,  # Insertion
                dp[i - 1][j - 1] + cost,  # Substitution
            )

    # The final value in the matrix represents the Levenshtein distance
    return dp[m][n]

# Example usage
str1 = "This is a cat"
str2 = "That is a dog"
distance = levenshtein_distance(str1, str2)
print(f"The Levenshtein distance between '{str1}' and '{str2}' with substitution cost 2 i
s {distance}")
```

```
The Levenshtein distance between 'This is a cat' and 'That is a dog' with substitution co
st 2 is 10
```

Refer to above code block -5 from tutorial - 2 for the following question

Question:2

Assign different costs to insertions, deletions, and substitutions to reflect varying penalties for different types of edits. Calculate the Levenshtein distance with these weighted costs.

String1 = ("Natural language processing")

String2 = ("Computer science department")

Provide your explanation in the tex block below

In [56]:

```
##ENTER YOUR CODE HERE
# Weighted costs
INSERTION_COST = 2
DELETION_COST = 3
SUBSTITUTION_COST = 4

def levenshtein_distance(str1, str2):

  m, n = len(str1), len(str2)

  # Initialize a matrix to store edit distances
  dp = [[0] * (n + 1) for _ in range(m + 1)]

  # Initialize first row and column
  for i in range(m + 1):
    dp[i][0] = i * DELETION_COST
  for j in range(n + 1):
    dp[0][j] = j * INSERTION_COST

  # Fill in the matrix using dynamic programming
  for i in range(1, m + 1):
    for j in range(1, n + 1):
      if str1[i-1] == str2[j-1]:
        cost = 0
      else:
        cost = SUBSTITUTION_COST

      dp[i][j] = min(
        dp[i-1][j] + DELETION_COST,
        dp[i][j-1] + INSERTION_COST,
        dp[i-1][j-1] + cost)

  return dp[m][n]

# Example usage
str1 = "Natural language processing"
str2 = "Computer science department"

print(levenshtein_distance(str1, str2))
```

```
76
```

Your explanation

- **Standard Leveshtein distance uses cost as 1 for all operations, but in the above code, I have assigned different costs for different operations.**
  - **Insertion cost = 2**
  - **Deletion cost = 3**
  - **Substitution cost = 4**
- **levenshtein_distance is a method that takes 2 inputs - string-1 and string-2**
- **We filled up dynamic programming matrix, insted of increating by 1, I increaed with the particular cost assigned.**
- **The levenshtein_distance for the example string is 76.**

Question-3

Describe how MED is used in various NLP tasks, such as spell checking, speech recognition, text summarization, and machine translation. How does its effectiveness vary across these tasks?

Your Explanation Here

--> The below are the ways MED (Minimum Edit Distance) is used in various natural language processing tasks.

1. Spell checking:

- **MED helps in finding the correct spelling by looking for the word that requires fewer edits to change to the misspelled word. It is very useful in finding out the typos.**

2. Speech Recognition:

- **Sometimes the device does not understand the word that you told. So in that case it compares the word you told and the word with the word what it thinks you meant. It finds the word that required smallest number of changes and it picks that word**

3. Text Summarization:

- **Text summarization means shortening the paragraph/text without losing the main content. To do so MSD helps in picking the important sentence by looking how familier it is with the other sentence. If two sentences are similar then you need only one in the summary.**

4. Machine Translation:

- MSE is also used in machine translation. When converting the text from one language to another, it finds the best word/sentence/phrase to use in the other language by comparing it with the original and translated sentence and finds the smallest number of changes required.