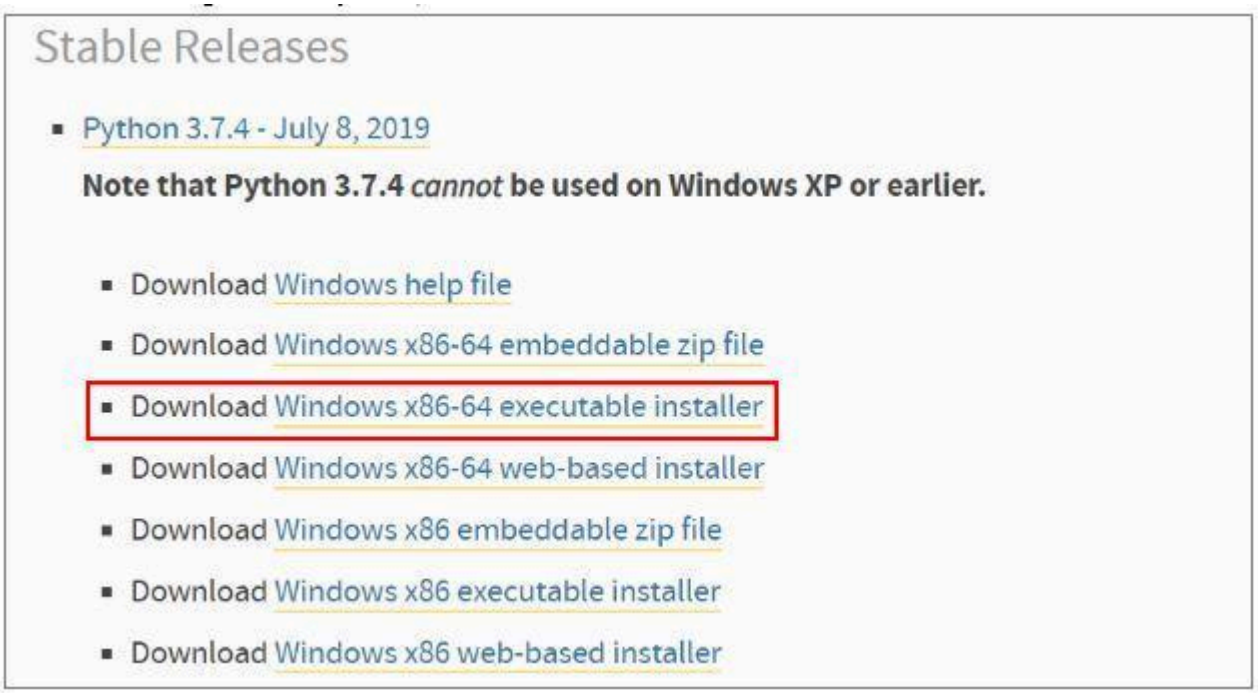
EXPERIMENT – 1:: Install NLP TOOLKIT

Process::

To download Python, you need to visit www.python.org, which is the official Python website.

Click on the Downloads tab and then select the Windows option.

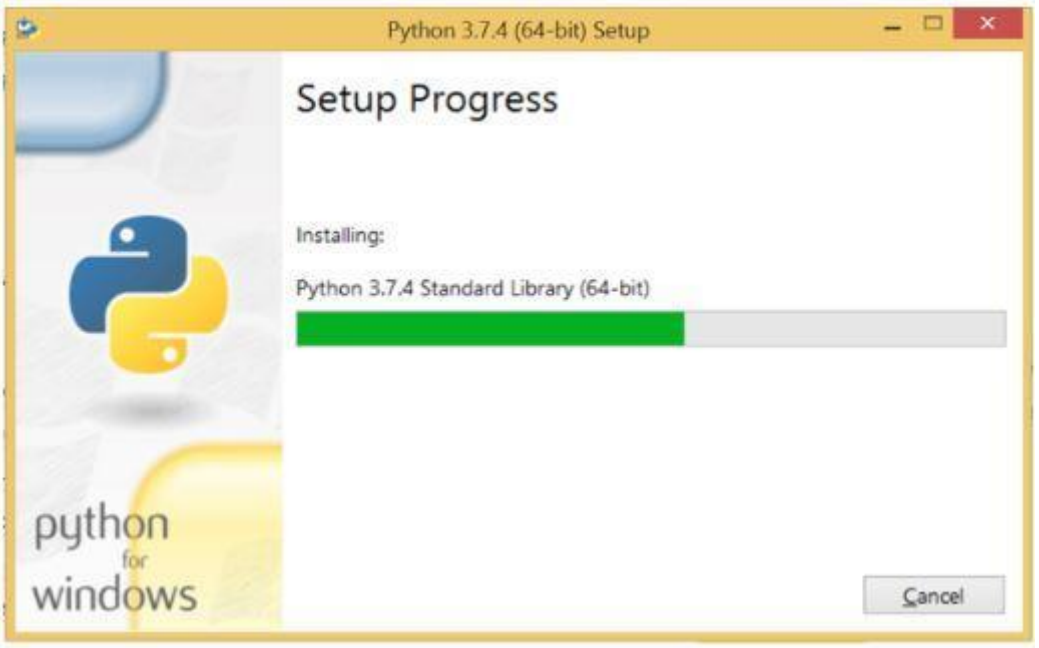
This will take you to the page where the different Python releases for Windows can be found. Since I am using a 64bit system, I'll select "Windows x86-64 executable installer".



Once the executable file download is complete, you can open it to install Python.

Click on Run, which will start the installation process.

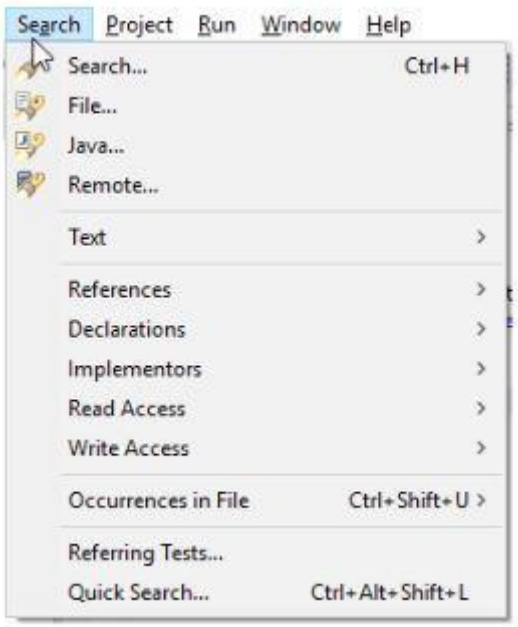
If you want to save the installation file in a different location, click on Customize installation; otherwise, continue with Install Now. Also, select the checkbox at the bottom to Add Python 3.7 to PATH.



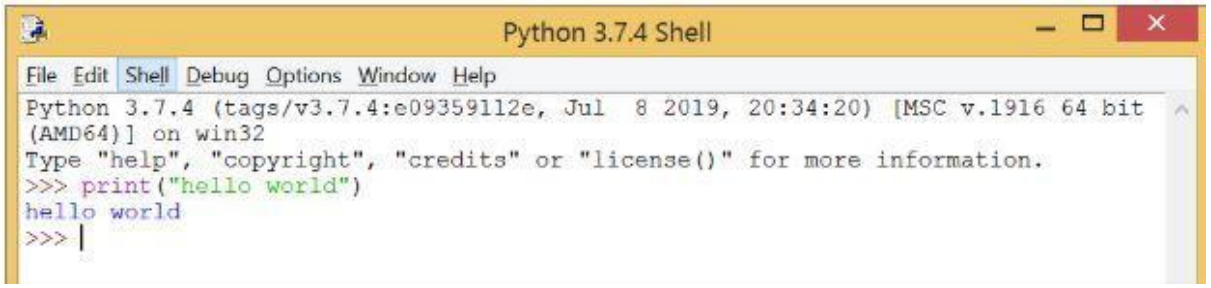
Once the installation is complete, the below pop-up box will appear: Setup was successful.

Now that the installation is complete, you need to verify that everything is working fine.

Go to Start and search for Python.



You can see Python 3.7 (64-bit) and IDLE. Let's open IDLE, which is the short form for Integrated Development Environment, and run a simple print statement.



Now , Python is successfully installed on your windows

As the same install jupyter notebook by typing the following in Command Prompt

```
C:\Users\DE LAB>pip install jupyter notebook
Collecting jupyter
  Downloading jupyter-1.0.0-py2.py3-none-any.whl (2.7 kB)
Collecting notebook
  Obtaining dependency information for notebook from https://files.pythonhosted.org/packages/f3/2b/b904c57709b83c6cbd818d21040db36719207f3d17db9b124c60cd483d94/notebook-7.0.6-py3-none-any.whl.metadata
  Downloading notebook-7.0.6-py3-none-any.whl.metadata (10 kB)
Collecting qtconsole (from jupyter)
  Obtaining dependency information for qtconsole from https://files.pythonhosted.org/packages/24/e2/7f22137bbb7270b016f6b0efa55d7598fef6ef354ba77515956bb28e8e54/qtconsole-5.4.4-py3-none-any.whl.metadata
  Downloading qtconsole-5.4.4-py3-none-any.whl.metadata (5.0 kB)
Collecting jupyter-console (from jupyter)
  Downloading jupyter_console-6.6.3-py3-none-any.whl (24 kB)
Collecting nbconvert (from jupyter)
  Obtaining dependency information for nbconvert from https://files.pythonhosted.org/packages/5b/08/6af17268360385f767c7a53dd2b71b9718c61911464fb34f5453c80cfe48/nbconvert-7.9.2-py3-none-any.whl.metadata
  Downloading nbconvert-7.9.2-py3-none-any.whl.metadata (7.9 kB)
Collecting ipykernel (from jupyter)
  Obtaining dependency information for ipykernel from https://files.pythonhosted.org/packages/c8/7d/2df9b38e2310e36a1b4a92bfe85f53ce24c638f9b7d9bd992bded11ce604/ipykernel-6.26.0-py3-none-any.whl.metadata
```

To install nltk type command in cmd as

Pip install nltk

```
C:\Users\DE LAB>pip install nltk
Collecting nltk
  Downloading nltk-3.8.1-py3-none-any.whl (1.5 MB)
     ---------------------------------------- 1.5/1.5 MB 4.0 MB/s eta 0:00:00
Collecting click (from nltk)
  Obtaining dependency information for click from https://files.pythonhosted.org/packages/00/2e/d53fa4befbf2cfa713304affc7ca780ce4fc1fd8710527771b58311a3229/click-8.1.7-py3-none-any.whl.metadata
  Downloading click-8.1.7-py3-none-any.whl.metadata (3.0 kB)
Collecting joblib (from nltk)
  Obtaining dependency information for joblib from https://files.pythonhosted.org/packages/10/40/d551139c85db202f1f384ba8bcf96aca2f329440a844f924c8a0040b6d02/joblib-1.3.2-py3-none-any.whl.metadata
  Downloading joblib-1.3.2-py3-none-any.whl.metadata (5.4 kB)
Collecting regex>=2021.8.3 (from nltk)
  Obtaining dependency information for regex>=2021.8.3 from https://files.pythonhosted.org/packages/d3/10/6f2d5f8635d7714ad97ce6ade7a643358c4f3e45cde4ed12b7150734a8f3/regex-2023.10.3-cp312-cp312-win_amd64.whl.metadata
  Downloading regex-2023.10.3-cp312-cp312-win_amd64.whl.metadata (41 kB)
     ---------------------------------------- 42.0/42.0 kB 1.0 MB/s eta 0:00:00
Collecting tqdm (from nltk)
  Obtaining dependency information for tqdm from https://files.pythonhosted.org/packages/00/e5/f12a80907d0884e6dff9c16d0c0114d81b8cd07dc3ae54c5e962cc83037e/tqdm-4.66.1-py3-none-any.whl.metadata
  Downloading tqdm-4.66.1-py3-none-any.whl.metadata (57 kB)
     ---------------------------------------- 57.6/57.6 kB 1.0 MB/s eta 0:00:00
Requirement already satisfied: colorama in c:\users\de lab\appdata\local\programs\python\python312\lib\site-packages (from click->nltk) (0.4.6)
Downloading regex-2023.10.3-cp312-cp312-win_amd64.whl (268 kB)
   ---------------------------------------- 269.0/269.0 kB 5.5 MB/s eta 0:00:00
Downloading click-8.1.7-py3-none-any.whl (97 kB)
   ---------------------------------------- 97.9/97.9 kB 1.9 MB/s eta 0:00:00
Downloading joblib-1.3.2-py3-none-any.whl (302 kB)
   ---------------------------------------- 302.2/302.2 kB 4.6 MB/s eta 0:00:00
Downloading tqdm-4.66.1-py3-none-any.whl (78 kB)
   ---------------------------------------- 78.3/78.3 kB 723.6 kB/s eta 0:00:00
Installing collected packages: tqdm, regex, joblib, click, nltk
Successfully installed click-8.1.7 joblib-1.3.2 nltk-3.8.1 regex-2023.10.3 tqdm-4.66.1

[notice] A new release of pip is available: 23.2.1 -> 23.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Users\DE LAB>
```

EXPERIMENT – 2:: NLTK Tokenize: Words and Sentences Tokenizer with Example.

INTRODUCTION ::

Tokenization is one of the first step in any NLP pipeline. Tokenization is nothing but splitting the raw text into small chunks of words or sentences, called tokens. If the text is split into words, then its called as 'Word Tokenization' and if it's split into sentences then its called as 'Sentence Tokenization'. Generally 'space' is used to perform the word tokenization and characters like 'periods, exclamation point and newline char are used for Sentence Tokenization. We have to choose the appropriate method as per the task in hand. While performing the tokenization few characters like spaces, punctuations are ignored and will not be the part of final list of tokens.

Types of Tokenization:

- **Word Tokenization**
- **Character Tokenization**
- **Sub Word Tokenization**

PROGRAM ::

```
text="Tokenization, is 1st step in NLP. It splits raw text. into small chunks of words, called tokens."


#----------------------sentence Tokenization----------------------------------

from nltk.tokenize import sent_tokenize

print(sent_tokenize(text))

print("\n")


#----------------------Word Tokenization----------------------------------

from nltk.tokenize import word_tokenize

word_tokenize(text)
```

OUTPUT ::

```
['Tokenization, is 1st step in NLP.', 'It splits raw text.', 'into small
chunks of words, called tokens.']
```

Out[6]:

```
['Tokenization',
 ',',
 'is',
 '1st',
 'step',
 'in',
 'NLP',
 '.',
 'It',
 'splits',
 'raw',
 'text',
 '.',
 'into',
 'small',
 'chunks',
 'of',
 'words',
 ',',
 'called',
 'tokens',
 '.']
```

EXPERIMENT – 3:: Pre-processing of text (Tokenization, Filtration, Script Validation, Stop Word Removal, Lower case conversion, Stemming).

INTRODUCTION ::

Tokenization: Splitting the sentence into words.

Filtration: It the process of removing stop words or any unnecessary data from the sentence

Script Validation :: At the stage of training a model in a data science project, and after training a model, you will want to know the performance of your model on unseen data. We can make changes to some parameters the model used in learning, this is called **hyper-parameter Tuning.**

Stop Word Removal :: Stop word removal is one of the most commonly used preprocessing steps across different NLP applications.The idea is simply removing the words that occur commonly across all the documents in the corpus.

Lower case conversion :: Converting all your data to lowercase helps in the process of preprocessing and in later stages in the NLP application, when you are doing parsing.So, converting the text to its lowercase format is quite easy.

Stemming :: Stemming, in Natural Language Processing (NLP), refers to the process of reducing a word to its word stem that affixes to suffixes and prefixes or the roots.

PROGRAM ::

```
# module -3  ----------  Tokenization  ----------


from nltk.tokenize import sent_tokenize

text="Text preprocessing. is an important step in Natural Language Processing NLP. It involves cleaning and transforming raw data suitable format"

print("--------------------SENTENCE TOKENIZATION-----------------------")

print(sent_tokenize(text))

print("\n")


# ---------- pre-processing (script validation)  ----------
```

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
nltk.download('stopwords')
nltk.download('punkt')

def preprocess_text(text):
    text = text.lower()
    tokens = word_tokenize(text)
    words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word.isalnum() and word not in words]
    stemmer = PorterStemmer()
    tokens = [stemmer.stem(word) for word in tokens]
    perprocessed_text = ' '.join(tokens)
    return perprocessed_text


if __name__ == '__main__':
    input_text = "Text preprocessing is an important step in Natural Language Processing NLP.It involves cleaning and transforming raw text data into a format suitable for analysis or modeling."
    perprocessed_text = preprocess_text(input_text)
    print("----------(script validation) ----------")
    print(perprocessed_text)


# ----------  Stop Word Removal and Fliteration ----------

import nltk
```

```
nltk.download('stopwords')


from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
text="Text preprocessing is an important step in Natural Language Processing NLP.It involves cleaning and transforming raw text data into a format suitable for analysis or modeling."
stop_words=set(stopwords.words('english'))
word_tokens=word_tokenize(text)
filtered_scentence=[w for w in word_tokens if not w in stop_words]
print("\n")
print("------------------------------STOP WORD REMOVAL & FILTERATION------------------------------")
print(filtered_scentence)



# ---------- Lower Case Converstion ----------
text="REGISTERED students ARE required to ATTEND THE INTERVIEW WITHOUt FAIL"
text=text.lower()
print("\n")
print("-----------------LOWER CASE CONVERSTION-----------------------")
print(text)


# ------------------  ****Stemming****  -------------------------
print("\n")
print("--------------------STEMMING-----------------------------")
from nltk.stem import PorterStemmer
ps=PorterStemmer()
text="final year is going to complete within 6 months"
for word in text.split():
   print(ps.stem(word))
```

OUTPUT ::

```
--------------------SENTENCE TOKENIZATION-----------------------
['Text preprocessing.', 'is an important step in Natural Language
Processing NLP.', 'It involves cleaning and transforming raw data
suitable format']


---------------------(script validation) ---------------------------
text preprocess import step natur languag process involv clean
transform raw text data format suitabl analysi model


------------------------------STOP WORD REMOVAL &
FILTERATION---------------------------------
['Text', 'preprocessing', 'important', 'step', 'Natural', 'Language',
'Processing', 'NLP.It', 'involves', 'cleaning', 'transforming', 'raw',
'text', 'data', 'format', 'suitable', 'analysis', 'modeling', '.']


-----------------LOWER CASE CONVERSTION-----------------------
registered students are required to attend the interview without fail


--------------------STEMMING------------------------------
final
year
is
go
to
complet
within
6
month
```

EXPERIMENT – 4 :: EXPLAIN ABOUT WORD ANALYSIS

INTRODUCTION ::

Analysis of a word into root and affix(es) is called as Morphological analysis of a word. It is mandatory to identify root of a word for any natural language processing task. A root word can have various forms. For example, the word 'play' in English has the following forms: 'play', 'plays', 'played' and 'playing'.

Types of Morphology

Morphology is of two types,

1. Inflectional morphology

Deals with word forms of a root, where there is no change in lexical category. For example, 'played' is an inflection of the root word 'play'. Here, both 'played' and 'play' are verbs.

2. Derivational morphology

Deals with word forms of a root, where there is a change in the lexical category. For example, the word form 'happiness' is a derivation of the word 'happy'. Here, 'happiness' is a derived noun form of the adjective 'happy'.

Morphological Features:

All words will have their lexical category attested during morphological analysis. A noun and pronoun can take suffixes of the following features: gender, number, person, case.

PROGRAM ::

```
# -------------------------  4. Word Analysis.  ----------------------

#word Analysis


import nltk

from nltk.tokenize import word_tokenize

from nltk import pos_tag

nltk.download('punkt')

nltk.download('averaged_perceptron_tagger')


def word_analysis(text):
```

```
    words = word_tokenize(text)
    pos_tags = pos_tag(words)
    return pos_tags


if __name__ == "__main__":
    text = "Eating Natural Language Processing is a sub field of AI " \
    "1)This is NLP Class" \
    "2)Good Moring"
    res = word_analysis(text)
    for word,pos in res:
        #wordsList = [w for w in  if not w in stop_words]
        print(f"word: {word}, parts of speech : {pos}")
```

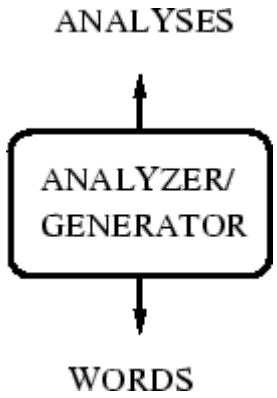
OUTPUT ::

```
word: Eating, parts of speech : VBG
word: Natural, parts of speech : NNP
word: Language, parts of speech : NNP
word: Processing, parts of speech : NNP
word: is, parts of speech : VBZ
word: a, parts of speech : DT
word: sub, parts of speech : JJ
word: field, parts of speech : NN
word: of, parts of speech : IN
word: AI, parts of speech : NNP
word: 1, parts of speech : CD
word: ), parts of speech : )
word: This, parts of speech : DT
word: is, parts of speech : VBZ
word: NLP, parts of speech : NNP
word: Class2, parts of speech : NNP
word: ), parts of speech : )
word: Good, parts of speech : NNP
word: Moring, parts of speech : VBG
```

EXPERIMENT 5 :: EXPLAIN ABOUT WORD GENERATION

INTRODUCTION ::

A word can be simple or complex. For example, the word 'cat' is simple because one cannot further decompose the word into smaller part. On the other hand, the word 'cats' is complex, because the word is made up of two parts: root 'cat' and plural suffix '-s'



PROGRAM ::

```
# ---------------------- word Generation  -------------------------#

import nltk
import random
nltk.download('punkt')
nltk.download('reuters')

from nltk.corpus import reuters

def build_word_model(corpus,order=2):
    word_model = {}
    for sc in corpus:
        words = nltk.word_tokenize(sc.lower())
        for i in range(len(words)-order):
            cur_state,next = tuple(words[i:i+order]),words[i + order]
            if cur_state not in word_model:
```

```
            word_model[cur_state] = []
            word_model[cur_state].append(next)
        return word_model

def generate_words(word_model,seed,length=10):
    cur_state = tuple(seed)
    generated_words = list(cur_state)

    for _ in range(length):
        if cur_state in word_model:
            next = random.choice(word_model[cur_state])
            generated_words.append(next)
            cur_state = tuple(generated_words[-len(seed):])
        else:
            break
        return "".join(generated_words)

if __name__=="__main__":
    corpus = reuters.sents()
    word_model = build_word_model(["".join(sent) for sent in corpus])

    seed =
["Natural","Processing","Language","Artificial","Deep","Learning","Welcome"]
    generated_text = generate_words(word_model,seed,length=20)
    print("Generated Text:")
    print(generated_text)
print("\n")


# ---------- another process word Generation  ----------#
```

```
import random

corpus =
["Natural","Processing","Language","Artificial","Deep","Learning","Welcome"]

r_words = random.choice(corpus)

print(r_words)
```

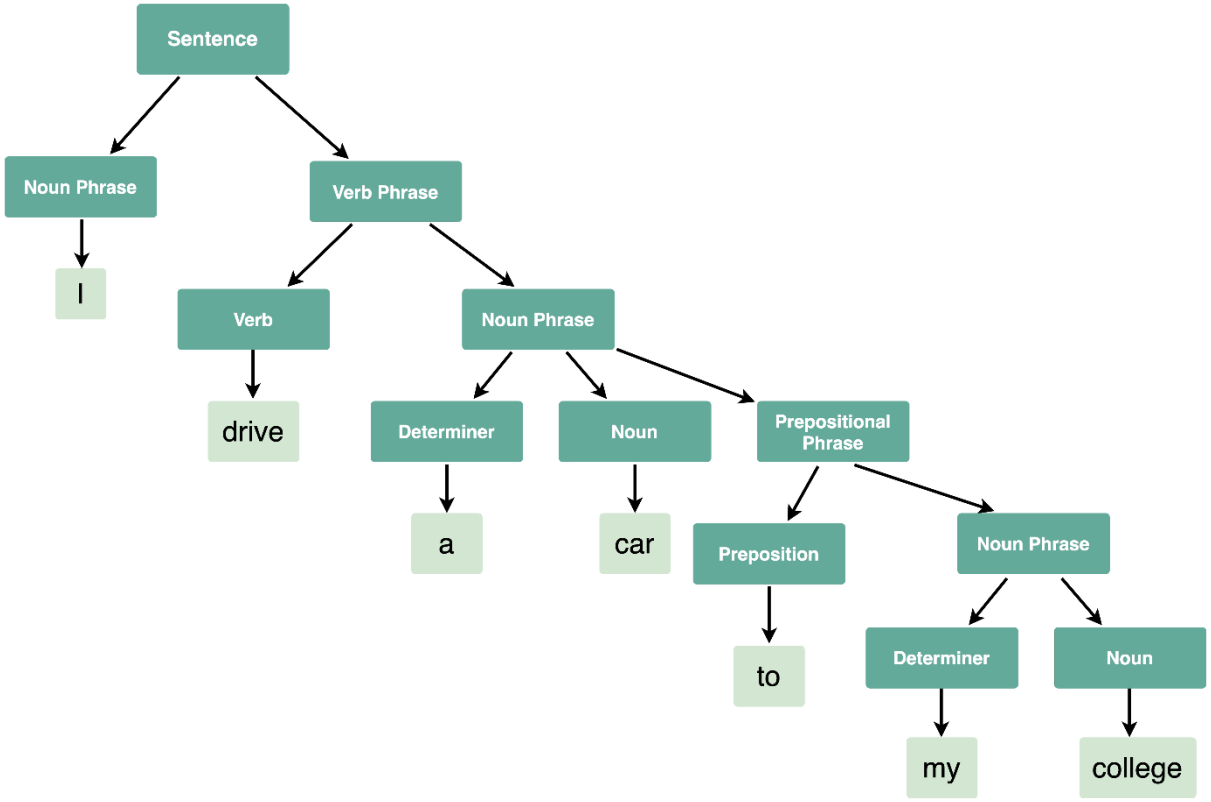
OUTPUT ::

```
Generated Text:
None


Learning
```

EXPERIMENT 6 :: EXPLAIN ABOUT PARSE TREE OR SYNTAX TREE GENERATION

INTRDUCTION ::

A Syntax tree or a parse tree is a tree representation of different syntactic categories of a sentence. It helps us to understand the syntactical structure of a sentence.
Example:
The syntax tree for the sentence given below is as follows:
I drive a car to my college.



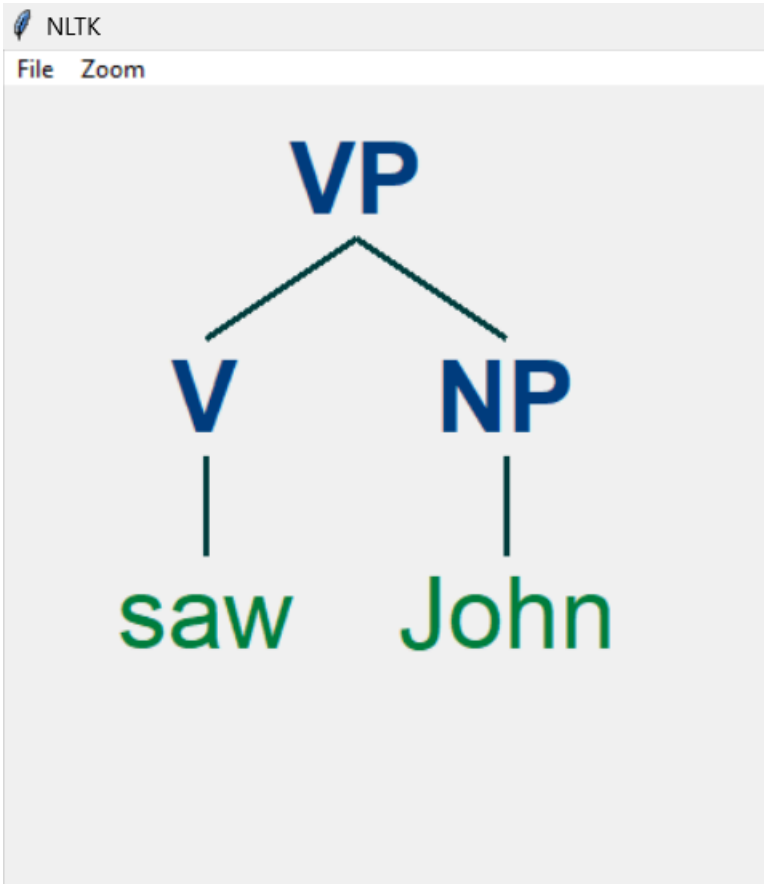
PROGRAM ::

```
# ---------------------  6. Parse tree or Syntax Tree generation  ------------------



# ----------------  Parse Tree  -------------------


grammar1 = nltk.CFG.fromstring("""

s -> NP VP

VP -> V NP | V NP PP

PP -> P NP
```

```
V -> "saw"|"ate"|"walked"
NP ->"John"|"Mary"|"Bob"|Det N|Det N PP
Det ->"a"|"an"|"the"|"my"
N ->"man"|"dog"|"cat"|"telescope"|"park"
p ->"in"|"on"|"by"|"with"
""")

sent = "Mary saw John".split()
rd_parser = nltk.RecursiveDescentParser(grammar1)
for tree in rd_parser.parse(sent):
    print(tree)
    tree[1].draw()
```

OUTPUT ::

```
(s (NP Mary) (VP (V saw) (NP John)))
```


NLTK
File
Zoom
VP
V
NP
saw
John

EXPERIMENT 7 :: EXPLAIN ABOUT N-GRAM MODEL.

INTRODUCTION ::

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. They come into play when we deal with text data in NLP (Natural Language Processing) tasks. They have a wide range of applications, like language models, semantic features, spelling correction, machine translation, text mining, etc.

PROGRAM ::

```
import re

from nltk.util import ngrams

txt="Deep Learning is subset of ML and ML is subset of AI" " and these are emerging technologies"

tokens=[token for token in txt.split(" ")]

op=list(ngrams(tokens,2))

print(op)

print("\n")

op1=list(ngrams(tokens,4))

print(op1)
```

OUTPUT ::

```
----------------N-GRAMS WITH 2 SETS-----------------------
[('Deep', 'Learning'), ('Learning', 'is'), ('is', 'subset'), ('subset',
'of'), ('of', 'ML'), ('ML', 'and'), ('and', 'ML'), ('ML', 'is'), ('is',
'subset'), ('subset', 'of'), ('of', 'AI'), ('AI', 'and'), ('and',
'these'), ('these', 'are'), ('are', 'emerging'), ('emerging',
'technologies')]


----------------N-GRAMS WITH 4 SETS-----------------------
[('Deep', 'Learning', 'is', 'subset'), ('Learning', 'is', 'subset',
'of'), ('is', 'subset', 'of', 'ML'), ('subset', 'of', 'ML', 'and'),
('of', 'ML', 'and', 'ML'), ('ML', 'and', 'ML', 'is'), ('and', 'ML',
'is', 'subset'), ('ML', 'is', 'subset', 'of'), ('is', 'subset', 'of',
'AI'), ('subset', 'of', 'AI', 'and'), ('of', 'AI', 'and', 'these'),
```

```
('AI', 'and', 'these', 'are'), ('and', 'these', 'are', 'emerging'),
('these', 'are', 'emerging', 'technologies')]
```

EXPERIMENT 8 :: EXPLAIN ABOUT POS TAGGING

INTRODUCTION ::

Part-of-speech (POS) tagging is a process in natural language processing (NLP) where each word in a text is labeled with its corresponding part of speech. This can include nouns, verbs, adjectives, and other grammatical categories.

POS tagging is useful for a variety of NLP tasks, such as information extraction, named entity recognition, and machine translation. It can also be used to identify the grammatical structure of a sentence and to disambiguate words that have multiple meanings.

PROGRAM ::

```
# ----------------------   8. POS tagging.   -------------------------


import nltk

nltk.download('averaged_perceptron_tagger')

nltk.download('punkt')



txt=nltk.word_tokenize("Access to online labs for lab facilities")

nltk.pos_tag(txt)
```

OUTPUT ::

```
[('Access', 'NN'),
 ('to', 'TO'),
 ('online', 'VB'),
 ('labs', 'NN'),
 ('for', 'IN'),
 ('lab', 'NN'),
 ('facilities', 'NNS')]
```

EXPERIMENT 9 :: EXPLAIN ABOUT CHUNKING

INTRODUCTION ::

Chunking is a process of extracting phrases from unstructured text, which means analyzing a sentence to identify the constituents(Noun Groups, Verbs, verb groups, etc.) However, it does not specify their internal structure, nor their role in the main sentence.

It works on top of POS tagging. It uses POS-tags as input and provides chunks as output.

sentence → clauses → phrases → words

Group of words make up phrases and there are five major categories.

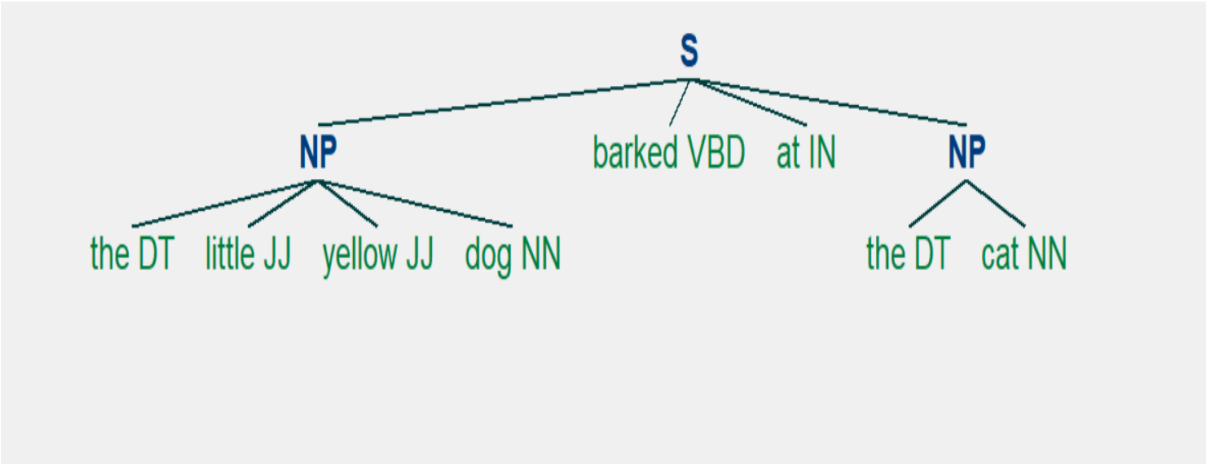
- Noun Phrase (NP)
- Verb phrase (VP)
- Adjective phrase (ADJP)

PROGRAM ::

```
#  ----------------------       9. Chunking. ----------------------


#chunking=it is a process of meaningful extracting short phrases from scentence.

import nltk

sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"), ("dog", "NN"), ("barked", "VBD"), ("at", "IN"), ("the", "DT"), ("cat", "NN")]

grammar = "NP: {<DT>?<JJ>*<NN>}"

cp = nltk.RegexpParser(grammar)

result = cp.parse(sentence)

print(result)

result.draw()
```

OUTPUT ::

```
(S
  (NP the/DT little/JJ yellow/JJ dog/NN)
  barked/VBD
  at/IN
  (NP the/DT cat/NN))
```



EXPERIMENT 10 :: EXPLAIN ABOUT NAMED ENTITY RECOGNITION.

INTRODUCTION ::

The named entity recognition (NER) is one of the most popular data preprocessing task. It involves the identification of key information in the text and classification into a set of predefined categories. An entity is basically the thing that is consistently talked about or refer to in the text.

NER is the form of NLP.

At its core, NLP is just a two-step process, below are the two steps that are involved:

- Detecting the entities from the text
- Classifying them into different categories

Some of the categories that are the most important architecture in NER such that:

- Person
- Organization
- Place/ location

PROGRAM ::

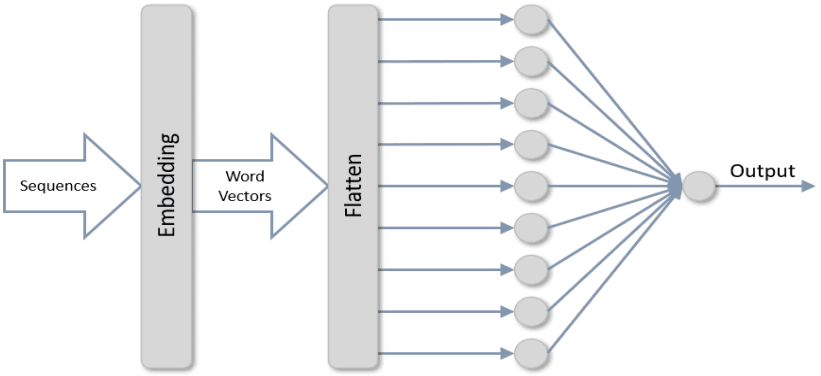
```
# ----------------------   10. Named Entity Recognition.    ----------

#name entity recognization ::: This is used to find the names of the person and locality from which they are working

loc = [('Prathyusha','in','Delhi'),('Gayathri','HELLO','USA'),('Mythri','in','NewYork'),('AVNSir','in','UK')]


Q=[e for(e,rel,e1) in loc if e1=='Delhi' or e1=='USA']

print(Q)
```

OUTPUT ::

```
['Prathyusha', 'Gayathri']
```

EXPERIMENT 11:: IMPLEMENT TEXT PROCESSING WITH NEURAL NETWORK.

INTRODUCTION ::



PROGRAM ::

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
```

```
# Sample data
texts = [
    "I love this product!",
    "Terrible experience, never buying again.",
    "It's okay, not great, not terrible.",
    "Fantastic service and quality!",
]
```

```
labels = np.array([1, 0, 0, 1])  # Convert labels to NumPy array
```

```
# Tokenize the text
max_words = 1000
tokenizer = Tokenizer(num_words=max_words, oov_token="<OOV>")
tokenizer.fit_on_texts(texts)
word_index = tokenizer.word_index
```

```
# Convert text to sequences
sequences = tokenizer.texts_to_sequences(texts)

# Pad sequences to have the same length
max_sequence_length = 5
padded_sequences = pad_sequences(sequences, maxlen=max_sequence_length, padding='post', truncating='post')
```

```
# Create a simple neural network
model = keras.Sequential([
    keras.layers.Embedding(input_dim=max_words, output_dim=16, input_length=max_sequence_length),
    keras.layers.Flatten(),
    keras.layers.Dense(8, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])
```

```
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(padded_sequences, labels, epochs=10)
```

```
Epoch 1/10
1/1 [==============================] - 1s 542ms/step - loss: 0.6920 - accuracy: 0.2500
Epoch 2/10
1/1 [==============================] - 0s 8ms/step - loss: 0.6864 - accuracy: 0.7500
Epoch 3/10
1/1 [==============================] - 0s 9ms/step - loss: 0.6819 - accuracy: 0.7500
Epoch 4/10
1/1 [==============================] - 0s 6ms/step - loss: 0.6775 - accuracy: 0.7500
Epoch 5/10
1/1 [==============================] - 0s 11ms/step - loss: 0.6730 - accuracy: 0.7500
Epoch 6/10
1/1 [==============================] - 0s 10ms/step - loss: 0.6690 - accuracy: 0.7500
Epoch 7/10
1/1 [==============================] - 0s 8ms/step - loss: 0.6651 - accuracy: 0.7500
Epoch 8/10
1/1 [==============================] - 0s 9ms/step - loss: 0.6612 - accuracy: 0.7500
Epoch 9/10
1/1 [==============================] - 0s 10ms/step - loss: 0.6574 - accuracy: 0.7500
Epoch 10/10
1/1 [==============================] - 0s 6ms/step - loss: 0.6537 - accuracy: 1.0000
<keras.src.callbacks.History at 0x7f9d818c7d30>
```

```
# Now, you can use the trained model for text classification
# For example, classify a new text
new_text = ["This is a great product!"]
new_sequence = tokenizer.texts_to_sequences(new_text)
new_padded_sequence = pad_sequences(new_sequence, maxlen=max_sequence_length, padding='post', truncating='post')
prediction = model.predict(new_padded_sequence)
```

```
1/1 [==============================] - 0s 61ms/step
```

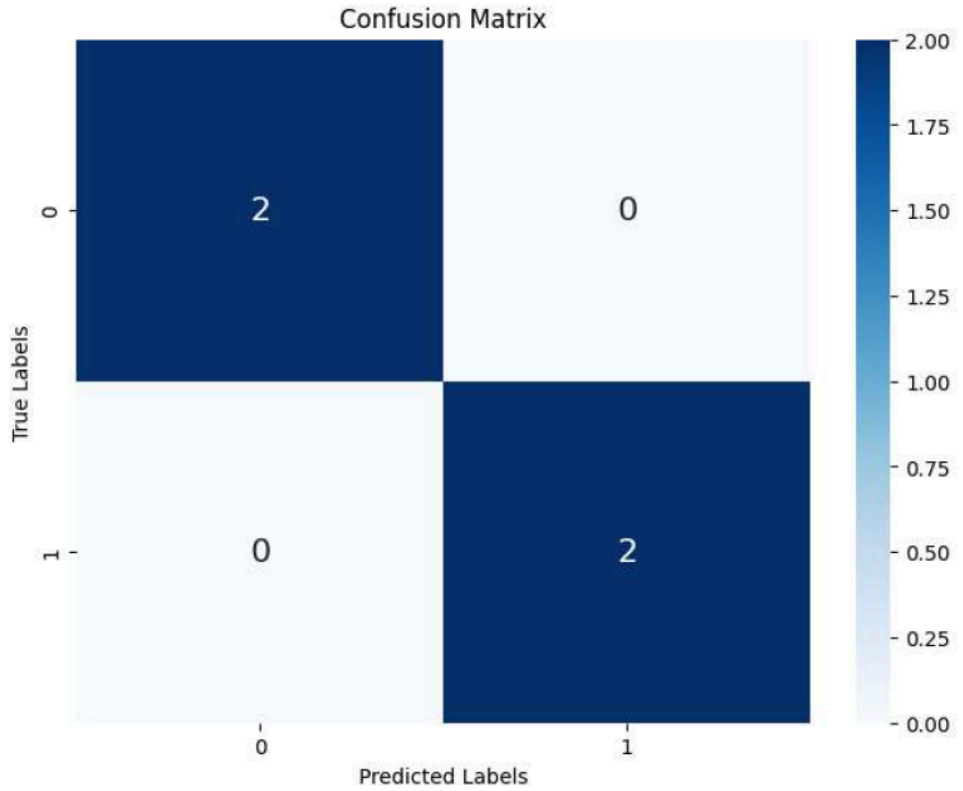
```
# Interpret the prediction (assuming 0.5 threshold for binary classification)
if prediction[0][0] > 0.5:
    print("Positive sentiment")
else:
    print("Negative sentiment")
```

```
Negative sentiment
```

```
# Calculate evaluation metrics
true_labels = labels
predicted_labels = (model.predict(padded_sequences) > 0.5).astype(int)
accuracy = accuracy_score(true_labels, predicted_labels)
precision = precision_score(true_labels, predicted_labels)
recall = recall_score(true_labels, predicted_labels)
f1 = f1_score(true_labels, predicted_labels)
conf_matrix = confusion_matrix(true_labels, predicted_labels)

# Create a heatmap for the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", annot_kws={"size": 16})
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()

# Create a table for evaluation metrics
metrics_df = pd.DataFrame({
    'Metric': ['Accuracy', 'Precision', 'Recall', 'F1-Score'],
    'Value': [accuracy, precision, recall, f1]
})

print(metrics_df)
```

OUTPUT ::

```
1/1 [==============================] - 0s 14ms/step
```




```
      Metric  Value
0   Accuracy    1.0
1  Precision    1.0
2     Recall    1.0
3   F1-Score    1.0
```

EXPERIMENT 12 :: IMPLEMENT TEXT PROCESSING WITH LSTM

PROGRAM ::

- a recurrent neural network (RNN) with a specific type of recurrent layer called Long Short-Term Memory (LSTM). LSTMs are a type of RNN that are well-suited for handling sequential data, making them a popular choice for natural language processing (NLP) tasks.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
```

+ Code + Text

```
# Sample data
texts = [
    "I love this product!",
    "Terrible experience, never buying again.",
    "It's okay, not great, not terrible.",
    "Fantastic service and quality!",
]
```

```
labels = np.array([1, 0, 0, 1])  # Convert labels to NumPy array
```

```
# Tokenize the text
max_words = 1000
tokenizer = Tokenizer(num_words=max_words, oov_token="<OOV>")
tokenizer.fit_on_texts(texts)
word_index = tokenizer.word_index
```

```
# Convert text to sequences
sequences = tokenizer.texts_to_sequences(texts)

# Pad sequences to have the same length
max_sequence_length = 5
padded_sequences = pad_sequences(sequences, maxlen=max_sequence_length, padding='post', truncating='post')
```

```
# Create an LSTM-based neural network
model = keras.Sequential([
    keras.layers.Embedding(input_dim=max_words, output_dim=16, input_length=max_sequence_length),
    keras.layers.LSTM(32),  # LSTM layer with 32 units
    keras.layers.Dense(8, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])
```

```
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(padded_sequences, labels, epochs=10)
```

```
Epoch 1/10
1/1 [==============================] - 3s 3s/step - loss: 0.6930 - accuracy: 0.5000
Epoch 2/10
1/1 [==============================] - 0s 19ms/step - loss: 0.6925 - accuracy: 1.0000
Epoch 3/10
1/1 [==============================] - 0s 14ms/step - loss: 0.6917 - accuracy: 1.0000
Epoch 4/10
1/1 [==============================] - 0s 14ms/step - loss: 0.6913 - accuracy: 1.0000
Epoch 5/10
1/1 [==============================] - 0s 13ms/step - loss: 0.6908 - accuracy: 1.0000
Epoch 6/10
1/1 [==============================] - 0s 14ms/step - loss: 0.6903 - accuracy: 1.0000
Epoch 7/10
1/1 [==============================] - 0s 19ms/step - loss: 0.6897 - accuracy: 1.0000
Epoch 8/10
1/1 [==============================] - 0s 21ms/step - loss: 0.6891 - accuracy: 1.0000
Epoch 9/10
1/1 [==============================] - 0s 22ms/step - loss: 0.6884 - accuracy: 1.0000
Epoch 10/10
1/1 [==============================] - 0s 18ms/step - loss: 0.6878 - accuracy: 1.0000
<keras.src.callbacks.History at 0x7a616a698be0>
```

```
# Now, you can use the trained model for text classification
# For example, classify a new text
new_text = ["This is not a great product, don't buy it."]
new_sequence = tokenizer.texts_to_sequences(new_text)
new_padded_sequence = pad_sequences(new_sequence, maxlen=max_sequence_length, padding='post', truncating='post')
prediction = model.predict(new_padded_sequence)
```

```
1/1 [==============================] - 2s 2s/step
```

```
# Interpret the prediction (assuming 0.5 threshold for binary classification)
if prediction[0][0] > 0.5:
    print("Positive sentiment")
else:
    print("Negative sentiment")
```

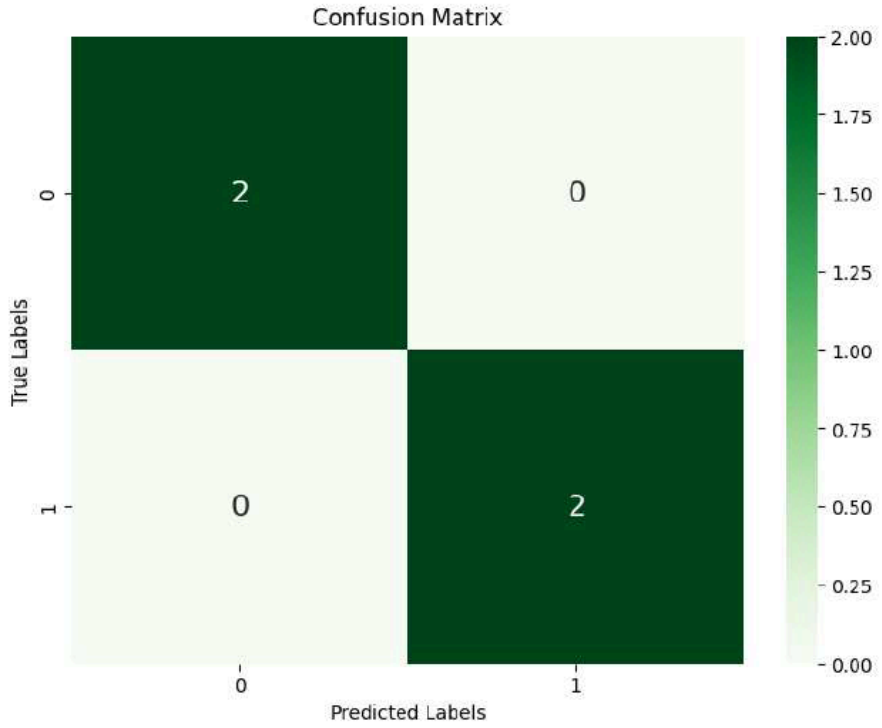
```
Negative sentiment
```

```
# Calculate evaluation metrics
true_labels = labels
predicted_labels = (model.predict(padded_sequences) > 0.5).astype(int)
accuracy = accuracy_score(true_labels, predicted_labels)
precision = precision_score(true_labels, predicted_labels)
recall = recall_score(true_labels, predicted_labels)
f1 = f1_score(true_labels, predicted_labels)
conf_matrix = confusion_matrix(true_labels, predicted_labels)
```

```
1/1 [==============================] - 0s 100ms/step
```

```
# Create a heatmap for the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Greens", annot_kws={"size": 16})
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```

OUTPUT ::




```
# Create a table for evaluation metrics
metrics_df = pd.DataFrame({
    'Metric': ['Accuracy', 'Precision', 'Recall', 'F1-Score'],
    'Value': [accuracy, precision, recall, f1]
})

print(metrics_df)
```

```
       Metric  Value
0    Accuracy    1.0
1   Precision    1.0
2      Recall    1.0
3    F1-Score    1.0
```