WEEK -1(frequency distribution using Histogram)

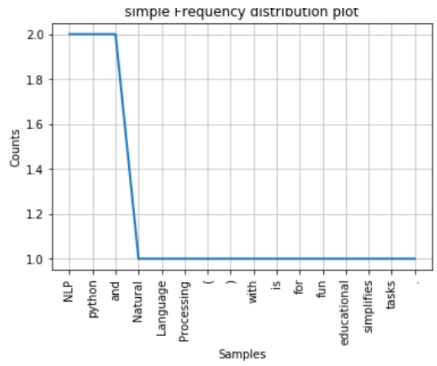
1-)Simple frequency Distribution

```
import nltk
from nltk.tokenize import TreebankWordTokenizer
from nltk.probability import FreqDist
import matplotlib.pyplot as plt
tokenizer=TreebankWordTokenizer()
text="Natural Language Processing(NLP) with python is for and fun and educational python simplifies NLP tasks."
tokens=tokenizer.tokenize(text)
print("tokens:",tokens)
fdist=FreqDist(tokens)
print("\n word frequency distribution:")
for word,Freq in fdist.items():
   print(f"{word}:{Freq}")
fdist.plot(title='simple Frequency distribution plot')
plt.show()
```

OUTPUT:-

```
tokens: ['Natural', 'Language', 'Processing', '(', 'NLP', ')', 'with', 'python', 'is', 'for', 'and', 'fun', 'and', 'educational', 'python', 'simplifies', 'NLP', 'tasks', '.']

 word frequency distribution:
Natural:1
Language:1
Processing:1
(:1
NLP:2
):1
with:1
python:2
is:1
for:1
and:2
fun:1
educational:1
simplifies:1
tasks:1
.:1
```



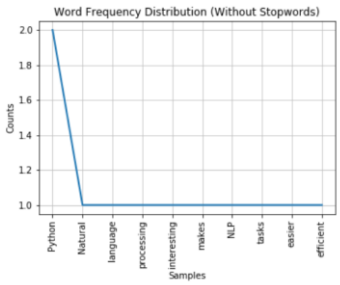
2-) Advanced frequency distribution with stopword removal

```
import nltk
from nltk.tokenize import TreebankWordTokenizer
from nltk.corpus import stopwords
from nltk.probability import FreqDist
import matplotlib.pyplot as plt
nltk.download('stopwords')
tokenizer = TreebankWordTokenizer()
stop_words = set(stopwords.words('english'))
text = "Natural language processing with Python is interesting and useful. Python makes NLP tasks easier and more efficient."
tokens = tokenizer.tokenize(text)
filtered_tokens = [word for word in tokens if word.lower() not in stop_words and word.isalpha()]
print("Filtered Tokens (without stopwords):", filtered_tokens)
fdist = FreqDist(filtered_tokens)
print("\nWord Frequency Distribution (without stopwords):")
for word, freq in fdist.items():
    print(f"{word}: {freq}")
fdist.plot(title='Word Frequency Distribution (Without Stopwords)')
plt.show()
```

OUTPUT:-

Filtered Tokens (without stopwords): ['Natural', 'language', 'processing', 'Python', 'interesting', 'Python', 'makes', 'NLP', 'tasks', 'easier', 'efficient']

Word Frequency Distribution (without stopwords):
Natural: 1
language: 1
processing: 1
Python: 2
interesting: 1
makes: 1
NLP: 1
tasks: 1
easier: 1
efficient: 1



WEEK-2-(nlp preprocessing:tokenization,stemming,lemmitazation)

#1-simple tokenization and stemming

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

from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
text="running run in a race."
tokens=word_tokenize(text)
stemmer=PorterStemmer()
stemmed=[stemmer.stem(token)for token in tokens]
print("stemmed words:",stemmed)
```

OUTPUT:-
stemmed words: ['run', 'run', 'in', 'a', 'race', '.']

#2 Advanced lemmatization

```
import nltk
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
from nltk import pos_tag, word_tokenize
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('punkt')

text = "The striped bats are hanging on their feet."
tokens = word_tokenize(text)
lemmatizer = WordNetLemmatizer()
def get_wordnet_pos(treebank_tag):
   if treebank_tag.startswith('J'):
      return wordnet.ADJ
   elif treebank_tag.startswith('V'):
      return wordnet.VERB
   elif treebank_tag.startswith('N'):
       return wordnet.NOUN
   elif treebank_tag.startswith('R'):
       return wordnet.ADV
```

```
    else:
        return wordnet.NOUN
lemmatized = [lemmatizer.lemmatize(w, get_wordnet_pos(t)) for w, t in
pos_tag(tokens)]
print("Lemmatized Words:", lemmatized)
```

OUTPUT:-
Lemmatized Words: ['The', 'striped', 'bat', 'be', 'hang', 'on', 'their', 'foot', '.']

WEEK-3-(DOCUMENT SIMLARITY USING VSM)

#1-simple cousine similarity

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
text1="Data science is a interseting field"
text2="Machine learning is a part of data science"
vectorizer=TfidfVectorizer()
x=vectorizer.fit_transform([text1,text2])
similarity=cosine_similarity(x[0:1],x[1:2])
print("cosine similarity:",similarity[0][0])
```

OUTPUT:-
cosine similarity: 0.34464214103805474

#2-advanced similarity using jaccard distance

```
text1="Data science involues statistics and programming"
text2="programming in python is essential for data science"
set1=set(text1.lower().split())
set2=set(text2.lower().split())
intersection=set1.intersection(set2)
union=set1.union(set2)
jaccard_similarity=len(intersection)/len(union)
print("jaccard similarity:",jaccard_similarity)
```

OUTPUT:-
jaccard similarity: 0.2727272727272727

WEEK -4-(POS TAGGING)

#1-simple pos tagging

```
import nltk
text="NLP is fun to learn"
tokens=nltk.word_tokenize(text)
pos_tags=nltk.pos_tag(tokens)
print("pos Tags:",pos_tags)
```

OUTPUT:-
pos Tags: [('NLP', 'NNP'), ('is', 'VBZ'), ('fun', 'VBN'), ('to', 'TO'), ('learn', 'VB')]

#2- Advanced pos tagging with frequency count

```
from collections import Counter
text = "NLP is a branch of artificial intelligence dealing with language."
tokens = nltk.word_tokenize(text)
pos_tags = nltk.pos_tag(tokens)
tags = [tag for word, tag in pos_tags]
frequency = Counter(tags)
print("POS Tag Frequency:", frequency)
```

OUTPUT:-

POS Tag Frequency: Counter({'NN': 3, 'IN': 2, 'NNP': 1, 'VBZ': 1, 'DT': 1, 'JJ': 1, 'VBG': 1, '.': 1})

WEEK -05(BIGRAM AND NGRAM TAGGING)

#1-simple Bigram generation

```
import nltk
from nltk import bigrams
text="NLP is amazing."
tokens = nltk.word_tokenize(text)
bigrams_list=list(bigrams(tokens))
print("bigrams:",bigrams_list)
```

OUTPUT:-

bigrams: [('NLP', 'is'), ('is', 'amazing'), ('amazing', '.')]

#2- advance Ngram generation

```
from nltk.util import ngrams
text = "NLP makes machines understand language."
tokens = nltk.word_tokenize(text)
trigrams_list = list(ngrams(tokens, 3))
print("Trigrams:", trigrams_list)
```

OUTPUT:-

Trigrams: [('NLP', 'makes', 'machines'), ('makes', 'machines', 'understand'), ('machines', 'understand', 'language'), ('understand', 'language', '.')]