In [ ]:
```
#omkar shinde
#Text Analysis
```

In [1]:
```
!pip install nltk
```

```
Requirement already satisfied: nltk in c:\users\avcoe\anaconda\lib\site-packages (3.
8.1)
Requirement already satisfied: click in c:\users\avcoe\anaconda\lib\site-packages (f
rom nltk) (8.0.4)
Requirement already satisfied: joblib in c:\users\avcoe\anaconda\lib\site-packages
(from nltk) (1.2.0)
Requirement already satisfied: regex>=2021.8.3 in c:\users\avcoe\anaconda\lib\site-p
ackages (from nltk) (2022.7.9)
Requirement already satisfied: tqdm in c:\users\avcoe\anaconda\lib\site-packages (fr
om nltk) (4.65.0)
Requirement already satisfied: colorama in c:\users\avcoe\anaconda\lib\site-packages
(from click->nltk) (0.4.6)
```

In [2]:
```
#Loading nltk
import nltk
nltk.download('punkt')
```

```
[nltk_data] Error loading punkt: <urlopen error [Errno 11001]
[nltk_data]     getaddrinfo failed>
```

Out[2]:
```
False
```

In [ ]:
```
#Tokenization
```

In [4]:
```
from nltk.tokenize import sent_tokenize
text="""Hello Mr. Smith, how are you doing today? The weather is
great, and city is awesome. The sky is pinkish-blue. You shouldn't
eat cardboard"""
tokenized_text=sent_tokenize(text)
print(tokenized_text)
```

```
['Hello Mr. Smith, how are you doing today?', 'The weather is \ngreat, and city is a
wesome.', 'The sky is pinkish-blue.', "You shouldn't \neat cardboard"]
```

In [ ]:
```
#word Tokenization
```

In [5]:
```
from nltk.tokenize import word_tokenize
tokenized_word=word_tokenize(text)

print(tokenized_word)
```

```
['Hello', 'Mr.', 'Smith', ',', 'how', 'are', 'you', 'doing', 'today', '?', 'The', 'w
eather', 'is', 'great', ',', 'and', 'city', 'is', 'awesome', '.', 'The', 'sky', 'i
s', 'pinkish-blue', '.', 'You', 'should', "n't", 'eat', 'cardboard']
```

In [ ]:
```
#Frequency Distribution
```

In [6]:
```
from nltk.probability import FreqDist

# Creating a frequency distribution object for the tokenized words
fdist = FreqDist(tokenized_word)

print(fdist)
```

```
<FreqDist with 25 samples and 30 outcomes>
```

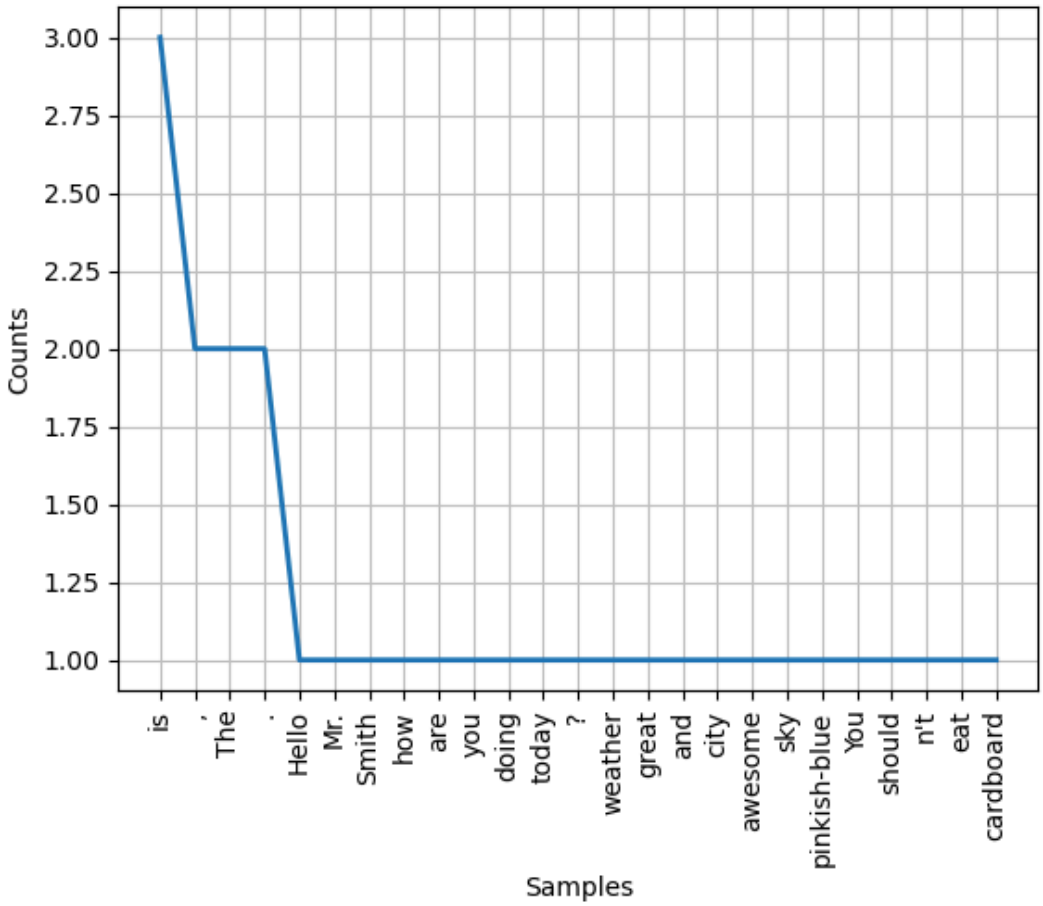
In [7]:
```
fdist.most_common(2)
```

Out[7]:
```
[('is', 3), (',', 2)]
```

In [ ]:
```
#Frequency Distribution Plot
```

In [8]:
```
import matplotlib.pyplot as plt

fdist.plot(30,cumulative=False)
plt.show()
```



In [ ]:
```
#POS Tagging
```

In [9]:
```
sent = "Albert Einstein was born in Ulm, Germany in 1879."
tokens=nltk.word_tokenize(sent)
print(tokens)
```

```
['Albert', 'Einstein', 'was', 'born', 'in', 'Ulm', ',', 'Germany', 'in', '1879',
'.']
```

In [10]:
```
nltk.download('averaged_perceptron_tagger')
nltk.pos_tag(tokens)
```

```
[nltk_data] Error loading averaged_perceptron_tagger: <urlopen error
[nltk_data]     [Errno 11001] getaddrinfo failed>
```

Out[10]:
```
[('Albert', 'NNP'),
 ('Einstein', 'NNP'),
 ('was', 'VBD'),
 ('born', 'VBN'),
 ('in', 'IN'),
 ('Ulm', 'NNP'),
 (',', ','),
 ('Germany', 'NNP'),
 ('in', 'IN'),
 ('1879', 'CD'),
 ('.', '.')]
```

In [ ]:
```
#StopWords
```

In [11]:
```
from nltk.corpus import stopwords
stop_words=set(stopwords.words("english"))
print(stop_words)
```

```
{'by', 'hers', 'they', "shouldn't", 'out', 'no', 'our', 'both', 'haven', 'to', 'who
m', 'than', "don't", 'am', "needn't", 'again', "didn't", 'wouldn', 'until', 'will',
'so', 'when', 'what', 'once', 'ourselves', 'how', 'we', 'i', 'd', 'won', 'not', 'did
n', 'm', 'did', 'needn', 'my', 'it', 'don', 'll', 'own', 'mightn', "mustn't", "does
n't", 'as', "couldn't", "weren't", 'but', 'same', 'had', "hasn't", 'all', 'any', 've
ry', "won't", 'if', 'hadn', 'before', 'between', "you've", 'wasn', 'hasn', 'up', 'he
r', 'about', 'ain', 'in', 'who', 'themselves', 'the', 'doesn', 'be', 'of', 'can', 'h
ere', "that'll", 's', 'now', 'do', 'have', 'under', 'isn', 'there', "you're", 'yours
elf', 'with', 'against', 'then', 'nor', "it's", 'she', 've', 'which', 'aren', "are
n't", 'was', 'does', 'you', 'herself', 'couldn', "wasn't", 'his', 'and', "wouldn't",
'most', 'that', 'some', 'those', 'he', 'because', 'is', 'more', "mightn't", 'mysel
f', 'an', 'below', 're', 'down', 'a', 'such', 'yours', "she's", 'them', 'at', 't',
"should've", "haven't", 'should', 'this', 'into', 'your', 'shouldn', 'while', 'thes
e', 'y', "you'd", 'few', 'too', 'me', 'its', "hadn't", 'on', 'him', 'ma', "shan't",
'their', 'himself', 'after', 'theirs', 'weren', 'only', 'having', 'for', 'ours', 'ar
e', 'doing', 'shan', 'why', 'over', 'o', 'just', 'been', 'mustn', 'during', 'above',
'off', 'other', "you'll", 'or', 'being', 'from', 'further', 'yourselves', "isn't",
'has', 'through', 'were', 'each', 'where', 'itself'}
```

In [ ]:
```
#removing Stop words
```

In [13]:
```
filtered_sent =[]
tokenized_sent =[]
for w in tokenized_sent:
    if w not in stop_words:
        filtered_sent.append(w)
print("Tokenized Sentence:",tokenized_sent)
print("Filtered Sentence:",filtered_sent)
```

```
Tokenized Sentence: []
Filtered Sentence: []
```

In [14]:
```
filtered_sent = []
tokenized_words = word_tokenize(text)

# Initialize an empty list to store the filtered words
filtered_words = []

for w in tokenized_words:
    # Check if the word is not in the set of stop words
    if w.lower() not in stop_words:
        filtered_words.append(w)
print("Filtered Words:", filtered_words)
print("\n")
print("Tokenized Words:", tokenized_words)
```

```
Filtered Words: ['Hello', 'Mr.', 'Smith', ',', 'today', '?', 'weather', 'great',
',', 'city', 'awesome', '.', 'sky', 'pinkish-blue', '.', "n't", 'eat', 'cardboard']


Tokenized Words: ['Hello', 'Mr.', 'Smith', ',', 'how', 'are', 'you', 'doing', 'toda
y', '?', 'The', 'weather', 'is', 'great', ',', 'and', 'city', 'is', 'awesome', '.',
'The', 'sky', 'is', 'pinkish-blue', '.', 'You', 'should', "n't", 'eat', 'cardboard']
```

In [ ]:
```
#stemming
```

In [15]:
```
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

# Initialize the Porter Stemmer
ps = PorterStemmer()

# Initialize a list to store stemmed words
stemmed_words = []

# Stemming each word in filtered_words
for w in filtered_words:
    stemmed_words.append(ps.stem(w))

print("Filtered Sentence:", filtered_words)
print("Stemmed Sentence:", stemmed_words)
```

```
Filtered Sentence: ['Hello', 'Mr.', 'Smith', ',', 'today', '?', 'weather', 'great',
',', 'city', 'awesome', '.', 'sky', 'pinkish-blue', '.', "n't", 'eat', 'cardboard']
Stemmed Sentence: ['hello', 'mr.', 'smith', ',', 'today', '?', 'weather', 'great',
',', 'citi', 'awesom', '.', 'sky', 'pinkish-blu', '.', "n't", 'eat', 'cardboard']
```

In [ ]:
```
#Lemmatization
```

In [16]:
```
#lexicon Normalization
#performing stemming and Lemmatization

from nltk.stem.wordnet import WordNetLemmatizer
lem = WordNetLemmatizer()

from nltk.stem.porter import PorterStemmer
stem = PorterStemmer()
word = "flying"
print("Lemmatized Word:",lem.lemmatize(word,"v"))
print("Stemmed Word:",stem.stem(word))
```

```
Lemmatized Word: fly
Stemmed Word: fli
```

In [ ]:
```
#TErm Frequency
```

In [17]:
```
import pandas as pd
import sklearn as sk
import math

first_sentence = "Data Science is the sexiest job of the 21st century"
second_sentence = "machine learning is the key for data science"

# Splitting each sentence into individual words
first_sentence = first_sentence.split(" ")
second_sentence = second_sentence.split(" ")

# Combining the words from both sentences and removing duplicates
total = set(first_sentence).union(set(second_sentence))
print(total)
```

```
{'Science', '21st', 'century', 'for', 'learning', 'Data', 'of', 'sexiest', 'is', 'ke
y', 'machine', 'data', 'the', 'science', 'job'}
```

In [18]:
```
wordDictA = dict.fromkeys(total, 0)
wordDictB = dict.fromkeys(total, 0)
for word in first_sentence:
    wordDictA[word] += 1
for word in second_sentence:
    wordDictB[word] += 1
```

In [19]:
```
df = pd.DataFrame([wordDictA, wordDictB])

df
```

Out[19]:

	Science	21st	century	for	learning	Data	of	sexiest	is	key	machine	data	the	science	job
0	1	1	1	0	0	1	1	1	1	0	0	0	2	0	1
1	0	0	0	1	1	0	0	0	1	1	1	1	1	1	0

In [20]:
```
def computeTF(wordDict, doc):
    tfDict = {}
    corpusCount = len(doc)
    for word, count in wordDict.items():
        tfDict[word] = count / float(corpusCount)
    return tfDict

tfFirst = computeTF(wordDictA, first_sentence)
tfSecond = computeTF(wordDictB, second_sentence)

tf = pd.DataFrame([tfFirst, tfSecond])

tf
```

Out[20]:

	Science	21st	century	for	learning	Data	of	sexiest	is	key	machine	data	the	scienc
0	0.1	0.1	0.1	0.000	0.000	0.1	0.1	0.1	0.100	0.000	0.000	0.000	0.200	0.0(
1	0.0	0.0	0.0	0.125	0.125	0.0	0.0	0.0	0.125	0.125	0.125	0.125	0.125	0.12

In [21]:
```
stopwords("english")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[21], line 1
----> 1 stopwords("english")

TypeError: 'WordListCorpusReader' object is not callable
```

In [22]:
```
# IDF ( Inverse Document Frequency)

def computeIDF(docList):
    idfDict = {}
    N = len(docList)
    idfDict = dict.fromkeys(docList[0].keys(), 0)
    for doc in docList:
        for word, val in doc.items():
            if val > 0:
                idfDict[word] += 1
    for word, val in idfDict.items():
        idfDict[word] = math.log10(N / float(val))

    return idfDict
idfs = computeIDF([wordDictA, wordDictB])

idfs
```

Out[22]:
```
{'Science': 0.3010299956639812,
 '21st': 0.3010299956639812,
 'century': 0.3010299956639812,
 'for': 0.3010299956639812,
 'learning': 0.3010299956639812,
 'Data': 0.3010299956639812,
 'of': 0.3010299956639812,
 'sexiest': 0.3010299956639812,
 'is': 0.0,
 'key': 0.3010299956639812,
 'machine': 0.3010299956639812,
 'data': 0.3010299956639812,
 'the': 0.0,
 'science': 0.3010299956639812,
 'job': 0.3010299956639812}
```

In [23]:
```
#  TF-IDF (Term Frequency-Inverse Document Frequency)

def computeTFIDF(tfBow, idfs):
    tfidf = {}
    for word, val in tfBow.items():
        tfidf[word] = val * idfs[word]
    return tfidf
idfFirst = computeTFIDF(tfFirst, idfs)
idfSecond = computeTFIDF(tfSecond, idfs)

idf = pd.DataFrame([idfFirst, idfSecond])

idf
```

Out[23]:

	Science	21st	century	for	learning	Data	of	sexiest	is	key	machin
0	0.030103	0.030103	0.030103	0.000000	0.000000	0.030103	0.030103	0.030103	0.0	0.000000	0.00000
1	0.000000	0.000000	0.000000	0.037629	0.037629	0.000000	0.000000	0.000000	0.0	0.037629	0.03762

In [24]:
```
# First step is to import the library
from sklearn.feature_extraction.text import TfidfVectorizer

# For the sentence, make sure all words are lowercase or you will run into error.
# For simplicity, I just made the same sentence all lowercase
first_sentence = "Data Science is the sexiest job of the 21st century"
second_sentence = "machine learning is the key for data science"

# Calling the TfidfVectorizer
vectorizer = TfidfVectorizer()

# Fitting the model and passing our sentences right away
response = vectorizer.fit_transform([first_sentence.lower(), second_sentence.lower()])
print(response)
```

```
  (0, 1)        0.34211869506421816
  (0, 0)        0.34211869506421816
  (0, 9)        0.34211869506421816
  (0, 5)        0.34211869506421816
  (0, 11)       0.34211869506421816
  (0, 12)       0.48684053853849035
  (0, 4)        0.24342026926924518
  (0, 10)       0.24342026926924518
  (0, 2)        0.24342026926924518
  (1, 3)        0.40740123733358447
  (1, 6)        0.40740123733358447
  (1, 7)        0.40740123733358447
  (1, 8)        0.40740123733358447
  (1, 12)       0.28986933576883284
  (1, 4)        0.28986933576883284
  (1, 10)       0.28986933576883284
  (1, 2)        0.28986933576883284
```

In [ ]: