In [1]:

```
#Loading NLTK
#TEXT MINING ANALYSIS
#1.NLTK IS A POWERFUL PACKAGE THAT PROVIDES A SET OF DIVERSE NATURAL LANGUAGES AL
#2.IT IS FREE,OPENSOURCE EASY TO USE AND WEEL DOCUMENTED.
#3.NLTK CONSISTS OF THE MOST COMMON ALGORITHMS SUCH AS TOKENZING,PART OD SPEECH T
# TOPIC SEGMENTATION,AND NAMED ENTITY RECOGNITION NLTK HELPS THE COMPUTER TO ANAL
import nltk
```

In [10]:

```
#Tokenization is the first step in Text Analytics.
#The Process of Breaking Down a Text Paragraph into Smaller Chunks Such as Words
#Token is Single Entity That is Building Blocks For Sentence or Paragraph.
#SENTENCE TOKENIZATION
from nltk.tokenize import sent_tokenize
text="""Hello Miss.Vanita,what are you doing today? the weather is great,and city
tokenized_sent=sent_tokenize(text)
print(tokenized_sent)
```

```
['Hello Miss.Vanita,what are you doing today?', 'the weather is great,and city
is awesome.', 'The Sky is Pinkish-Blue.']
```

In [11]:

```
# Word Tokenizer Breaks Text Paragraph into Words.
# WORD TOKENIZATION
from nltk.tokenize import word_tokenize
text="""Hello Miss.Vanita,what are you doing today? the weather is great,and city
tokenized_word=word_tokenize(text)
print(tokenized_word)
```

```
['Hello', 'Miss.Vanita', ',', 'what', 'are', 'you', 'doing', 'today', '?', 'th
e', 'weather', 'is', 'great', ',', 'and', 'city', 'is', 'awesome', '.', 'The',
'Sky', 'is', 'Pinkish-Blue', '.']
```

In [8]:

```
#FREQUENCY DISTRIBUTION
from nltk.probability import FreqDist
fdist=FreqDist(tokenized_word)
print(fdist)
```

```
<FreqDist with 20 samples and 24 outcomes>
```

In [6]:

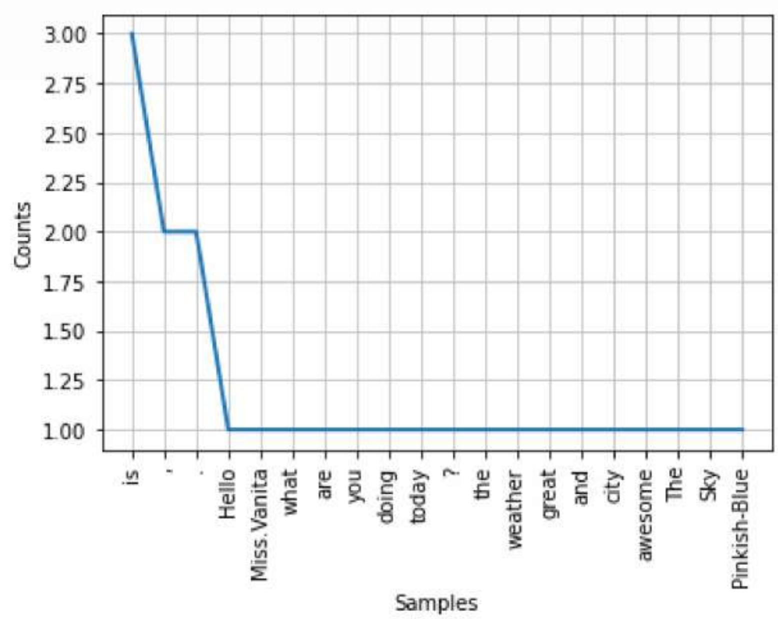
```
fdist.most_common(2)
```

Out[6]:

```
[('is', 3), (',', 2)]
```

In [9]:

```
#FREQUENCY DISTRIBUTION PLOT
import matplotlib.pyplot as plt
fdist.plot(30,cumulative=False)
plt.show()
```



In [10]:

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

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Owner\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Owner\AppData\Roaming\nltk_data...
```

Out[10]:

```
True
```

In [22]:

```
nltk.word_tokenize("hi How are you")
```

Out[22]:

```
['hi', 'How', 'are', 'you']
```

In [12]:

```
#STOPWORDS CONSIDERED AS NOISE IN THE TEXT.TEXT MAY CONTAIN STOP WORDS SUCH AS IS
#STOPWORDS
from nltk.corpus import stopwords
stop_words=stopwords.words("english")
print(stop_words)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're",
"you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',
'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "i
t's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what',
'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'i
s', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'havin
g', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'a
gainst', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'b
elow', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under',
'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'h
ow', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'suc
h', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very',
's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "could
n't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't",
'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "must
n't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wa
sn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

In [23]:

```
print(len(stopwords))
print(stopwords)
```

```
179
{'now', 'as', 'weren', 'of', 'between', "aren't", "didn't", 'once', 'won', 'abo
ut', "that'll", 'yourself', 'i', 'some', 'needn', 'through', 'and', 'again', 'a
in', 'own', "wouldn't", 'were', 'further', 'mightn', 'these', 'both', 'll', 'yo
ur', 'with', "you've", 'all', 'too', 'y', 'any', 'into', 'or', 'herself', 'at',
'down', 'shouldn', 'in', 'what', "hadn't", 'the', 'shan', 'such', 'during',
'o', 'nor', 'being', "you'd", 'was', 'above', 'who', 'after', 'there', 'for',
'did', "couldn't", 'm', 'it', 'just', 't', "don't", 'which', 'him', 'doesn', 'y
ou', 'my', 'more', 'on', "shouldn't", 'so', 'will', 'no', 'this', 'by', 'we',
'having', "haven't", 'myself', 're', 'where', 's', 'didn', 'yourselves', 'had
n', 'ma', 'to', 'its', 'himself', 'is', 'ours', "mustn't", 'other', 'if', 'fro
m', 'ourselves', 'than', 'not', 'aren', "isn't", 'up', 'under', 'most', 'wasn',
'hasn', 'me', 'should', "should've", 'be', 'are', "hasn't", "mightn't", 'hers',
'because', 'been', 'have', 'wouldn', 'each', 'they', "weren't", "she's", 'thos
e', "you'll", 'she', 'has', 'mustn', 'their', 'yours', 'our', 'itself', 'when',
'below', 'does', 'why', 'an', 'few', 'off', "wasn't", 'whom', "needn't", 'a',
'd', 'can', 'very', "doesn't", 'doing', 'don', 'her', 'themselves', 'isn', 'ha
d', 'over', "shan't", 'while', 'against', 'them', 'am', 'he', 'how', 'same', 'h
aven', 'until', 'before', 'then', 'only', "won't", 'that', "you're", "it's", 't
heirs', 'here', 've', 'do', 'couldn', 'out', 'but', 'his'}
```

In [18]:

```
stop_words.append('work')
print(stop_words)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're",
"you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',
'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "i
t's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what',
'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'i
s', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'havin
g', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'a
gainst', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'b
elow', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under',
'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'h
ow', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'suc
h', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very',
's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "could
n't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't",
'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "must
n't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wa
sn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't", 'work']
```

In [21]:

```
# Removing Stopwords
from nltk.tokenize import sent_tokenize,word_tokenize
from nltk.corpus import stopwords

data="AI was introduced in the year 1956 but it gained popularity recently."
stopwords=set(stopwords.words('english'))
words=word_tokenize(data)
wordsFiltered=[]

for w in words:
    if w not in stopwords:
        wordsFiltered.append(w)

print(wordsFiltered)
```

```
['AI', 'introduced', 'year', '1956', 'gained', 'popularity', 'recently', '.']
```

In [15]:

```
#Stemming is The Process of Bringing Words Back to Their Root form This Way You E
#For Example: Connection, Connected,Connected Word Reduce to a Common Word 'Conne
import nltk
from nltk.stem import PorterStemmer
#from nltk.tokenize import word_tokenize
stemmer= PorterStemmer()
Input_str="There are several types of stemming Algorithms."
Input_str=nltk.word_tokenize(Input_str)
for word in Input_str:
    print(stemmer.stem(word))
```

```
there
are
sever
type
of
stem
algorithm
.
```

In [6]:

```
#Lemmatization is Same Like Stemming i.e it Hve Same Goals As Like Stemming But D
import nltk
wn=nltk.WordNetLemmatizer()
ps=nltk.PorterStemmer()
dir(wn)
```

Out[6]:

```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'lemmatize']
```

In [38]:
```
print(ps.stem('goose'))
print(ps.stem('geese'))
```

```
goos
gees
```

In [39]:
```
print(wn.lemmatize('cactus'))
print(wn.lemmatize('cacti'))
```

```
cactus
cactus
```

In [16]:
```
#Stemming Code
import nltk
from nltk.stem.porter import PorterStemmer
porter_stemmer=PorterStemmer()
text="studies studying cries cry"
tokenization=nltk.word_tokenize(text)
for w in tokenization:
    print("Stemming for {} is {}".format(w,porter_stemmer.stem(w)))
```

```
Stemming for studies is studi
Stemming for studying is studi
Stemming for cries is cri
Stemming for cry is cri
```

In [17]:
```
# Lemmatization Code
import nltk
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer=WordNetLemmatizer()
text="studies studying cries cry"
tokenization=nltk.word_tokenize(text)
for w in tokenization:
    print("lemma for {} is {}".format(w,wordnet_lemmatizer.lemmatize(w)))

```

```
lemma for studies is study
lemma for studying is studying
lemma for cries is cry
lemma for cry is cry
```

In [24]:
```
# POS Tagging(Part of Speech Tagging) is The Process of attributing a Grammatical
import nltk
text=nltk.word_tokenize("It is a pleasant day today")
nltk.pos_tag(text)
```

Out[24]:
```
[('It', 'PRP'),
 ('is', 'VBZ'),
 ('a', 'DT'),
 ('pleasant', 'JJ'),
 ('day', 'NN'),
 ('today', 'NN')]
```

In [34]:
```
nltk.help.upenn_tagset('NNS')
```

```
NNS: noun, common, plural
    undergraduates scotches bric-a-brac products bodyguards facets coasts
    divestitures storehouses designs clubs fragrances averages
    subjectivists apprehensions muses factory-jobs ...
```

In [38]:
```
nltk.help.upenn_tagset('VB,*')
```

```
VB: verb, base form
    ask assemble assess assign assume atone attention avoid bake balkanize
    bank begin behold believe bend benefit bevel beware bless boil bomb
    boost brace break bring broil brush build ...
VBD: verb, past tense
    dipped pleaded swiped regummed soaked tidied convened halted registered
    cushioned exacted snubbed strode aimed adopted belied figgered
    speculated wore appreciated contemplated ...
VBG: verb, present participle or gerund
    telegraphing stirring focusing angering judging stalling lactating
    hankerin' alleging veering capping approaching traveling besieging
    encrypting interrupting erasing wincing ...
VBN: verb, past participle
    multihulled dilapidated aerosolized chaired languished panelized used
    experimented flourished imitated reunifed factored condensed sheared
    unsettled primed dubbed desired ...
VBP: verb, present tense, not 3rd person singular
    predominate wrap resort sue twist spill cure lengthen brush terminate
    appear tend stray glisten obtain comprise detest tease attract
    emphasize mold postpone sever return wag ...
VBZ: verb, present tense, 3rd person singular
    bases reconstructs marks mixes displeases seals carps weaves snatches
    slumps stretches authorizes smolders pictures emerges stockpiles
    seduces fizzes uses bolsters slaps speaks pleads ...
```

In [39]:
```
import nltk
text=nltk.word_tokenize("I cannot bear the pain of bear")
nltk.pos_tag(text)
```

Out[39]:
```
[('I', 'PRP'),
 ('can', 'MD'),
 ('not', 'RB'),
 ('bear', 'VB'),
 ('the', 'DT'),
 ('pain', 'NN'),
 ('of', 'IN'),
 ('bear', 'NN')]
```

In [51]:
```
# Bag of Words: Bag of Words is The Simplest Way of Structuring Textual data Ever
import sklearn
from sklearn.feature_extraction.text import CountVectorizer
```

In [48]:

```
phrases=["the quick brown fox jumped over the lazy dog"]
```

In [46]:

```
vect = CountVectorizer()
vect.fit(phrases)
```

Out[46]:

```
CountVectorizer()
```

In [47]:

```
print("Vocabulary size: {}".format(len(vect.vocabulary_)))
print("Vocabulary content:\n {}".format(vect.vocabulary_))
```

```
Vocabulary size: 8
Vocabulary content:
 {'the': 7, 'quick': 6, 'brown': 0, 'fox': 2, 'jumped': 3, 'over': 5, 'lazy':
4, 'dog': 1}
```

In [49]:

```
bag_of_words = vect.transform(phrases)
```

In [50]:

```
print(bag_of_words)
```

```
  (0, 0)	1
  (0, 1)	1
  (0, 2)	1
  (0, 3)	1
  (0, 4)	1
  (0, 5)	1
  (0, 6)	1
  (0, 7)	2
```

In [52]:

```
print("bag_of_words as an array:\n{}".format(bag_of_words.toarray()))
```

```
bag_of_words as an array:
[[1 1 1 1 1 1 1 2]]
```

In [53]:

```
 vect.get_feature_names()
```

Out[53]:

```
['brown', 'dog', 'fox', 'jumped', 'lazy', 'over', 'quick', 'the']
```

In [ ]:

```

```