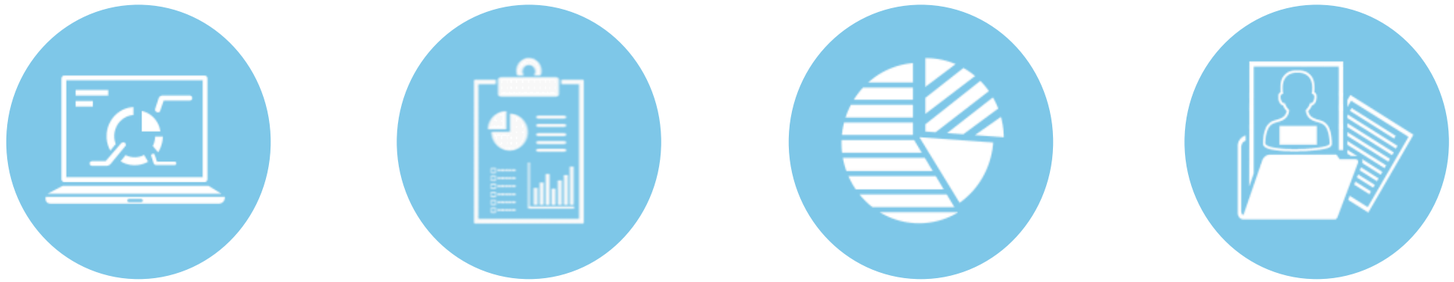
Machine Learning

Lesson 10: Text Mining



Learning Objectives

By the end of this lesson, you will be able to:

- Explain text mining
- Execute text processing tasks

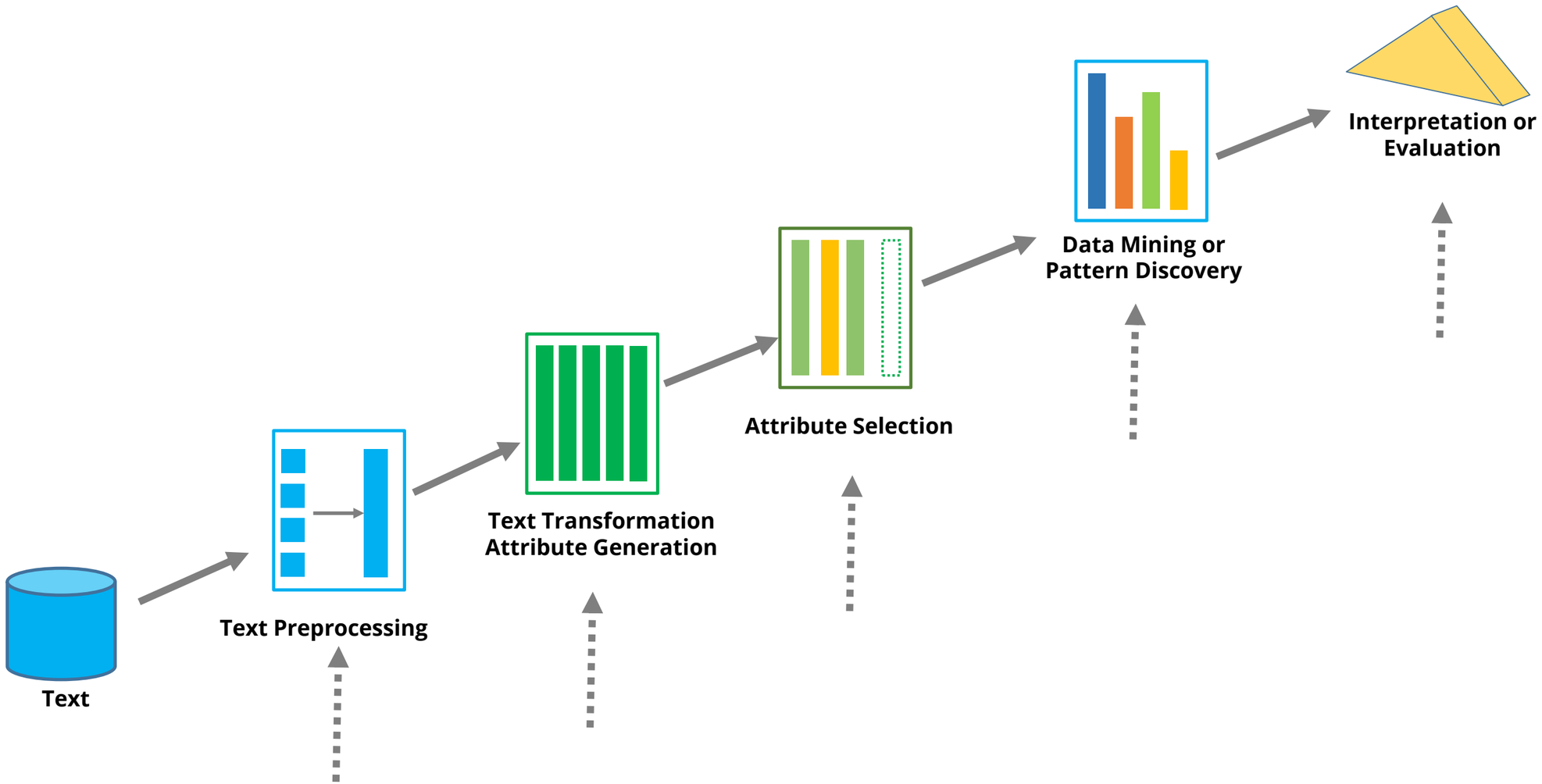
Concepts Covered

- NLTK corpora
- Text Extraction and Pre-processing
- Structuring Sentences

Text Mining

Topic 1: Overview

Definition



Text mining utilizes computational techniques to extract and summarize high-quality information from unstructured textual resources.

Significance

DOCUMENT CLUSTERING

Clustering makes it easy to group similar documents into meaningful groups.
News sections are often grouped as business, sports, politics

PRODUCT INSIGHTS

Mining consumer reviews can reveal insights like most loved feature, most hated feature, improvements required, reviews of competitors' products

PATTERN IDENTIFICATION

Features such as telephone numbers, e-mail addresses can be extracted using pattern matches

SECURITY MONITORING

Text mining helps in monitoring and extracting information from news articles and reports for national security purposes

Applications

SPEECH RECOGNITION

Recognition and translation of spoken language into text and vice - versa

SPAM FILTERING

Automatic detection of spam mail

SENTIMENT ANALYSIS

Determining if a given sentence expresses positive, neutral, or negative sentiment

E-COMMERCE PERSONALIZATION

Suggest products that fit into a user's profile

Text Mining

Topic 2: NLTK Library

Significance

Natural Language Toolkit (NLTK)

01

NLTK is a set of open source Python modules used to work with human language data for applying statistical natural language processing (NLP)

02

Provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet

03

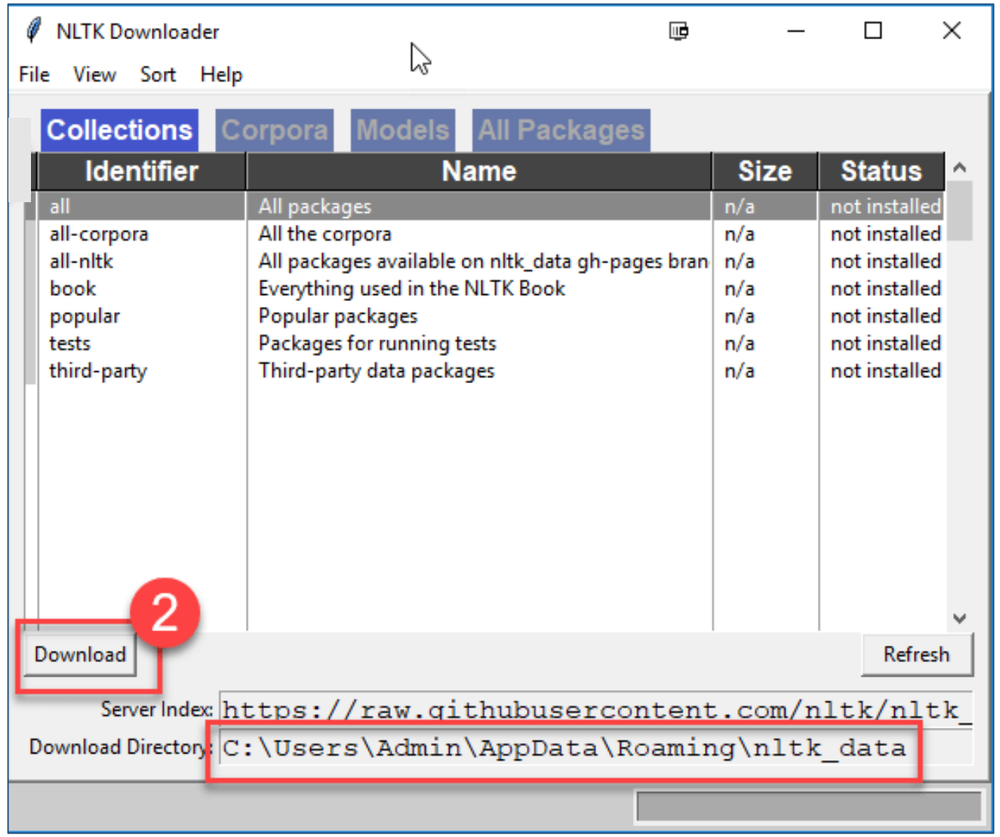
Provides text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning

Environment Setup

Step 1) Launch python interpreter from anaconda prompt and enter the highlighted commands

```
(base) C:\Users\       >python
Python 3.6.4 |Anaconda custom (64-bit)| (default, Jan 16 2018, 10:22:32) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import nltk
>>> nltk.download()
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
```

Step 2) NLTK downloaded window opens,
select all packages and click on download



Step 3) Test the setup using the below commands

```
In [1]: from nltk.corpus import brown

In [2]: brown.words()
Out[2]: ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
```

Reading NLTK corpora

Loading all items from NLTK's book module

```
In [4]: from nltk.book import *

*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
```

Exploring brown corpus

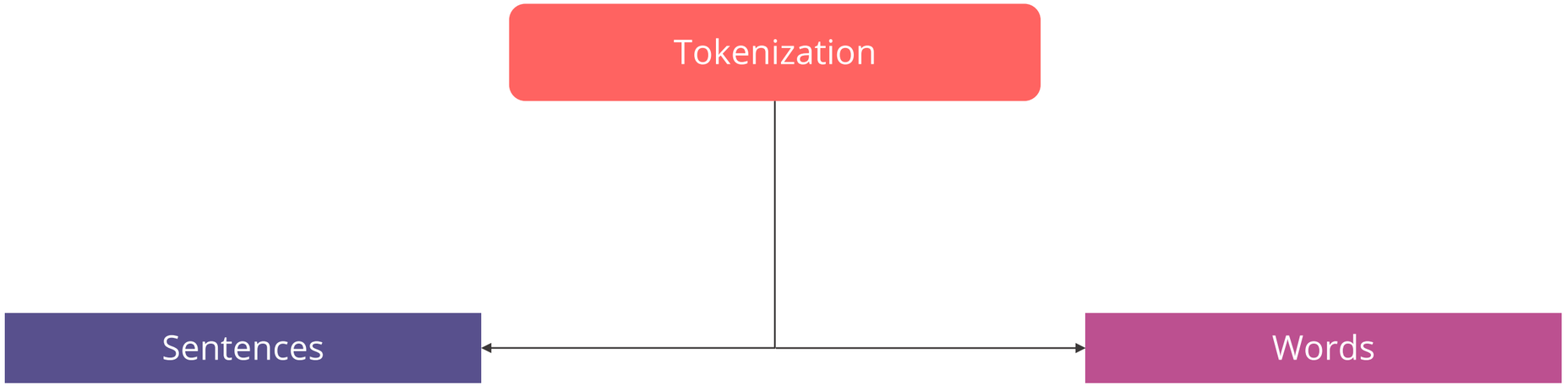
```
In [28]: from nltk.corpus import brown
         brown.categories()

Out[28]: ['adventure',
          'belles_lettres',
          'editorial',
          'fiction',
          'government',
          'hobbies',
          'humor',
          'learned',
          'lore',
          'mystery',
          'news',
          'religion',
          'reviews',
          'romance',
          'science_fiction']

In [26]: reviews_words = brown.words(categories='reviews')
         reviews_words

Out[26]: ['It', 'is', 'not', 'news', 'that', 'Nathan', ...]

In [27]: len(reviews_words)

Out[27]: 40704
```

Text Mining

Topic 3: Text Extraction and Pre-processing

Tokenization

Tokenization is a process of breaking running stream of text into words and sentences.
It works by separating words using spaces and punctuation.




```
from nltk.tokenize import sent_tokenize

EXAMPLE_SENT = "Hello World! This is sentence tokenizing."
print(sent_tokenize(EXAMPLE_TEXT))
```

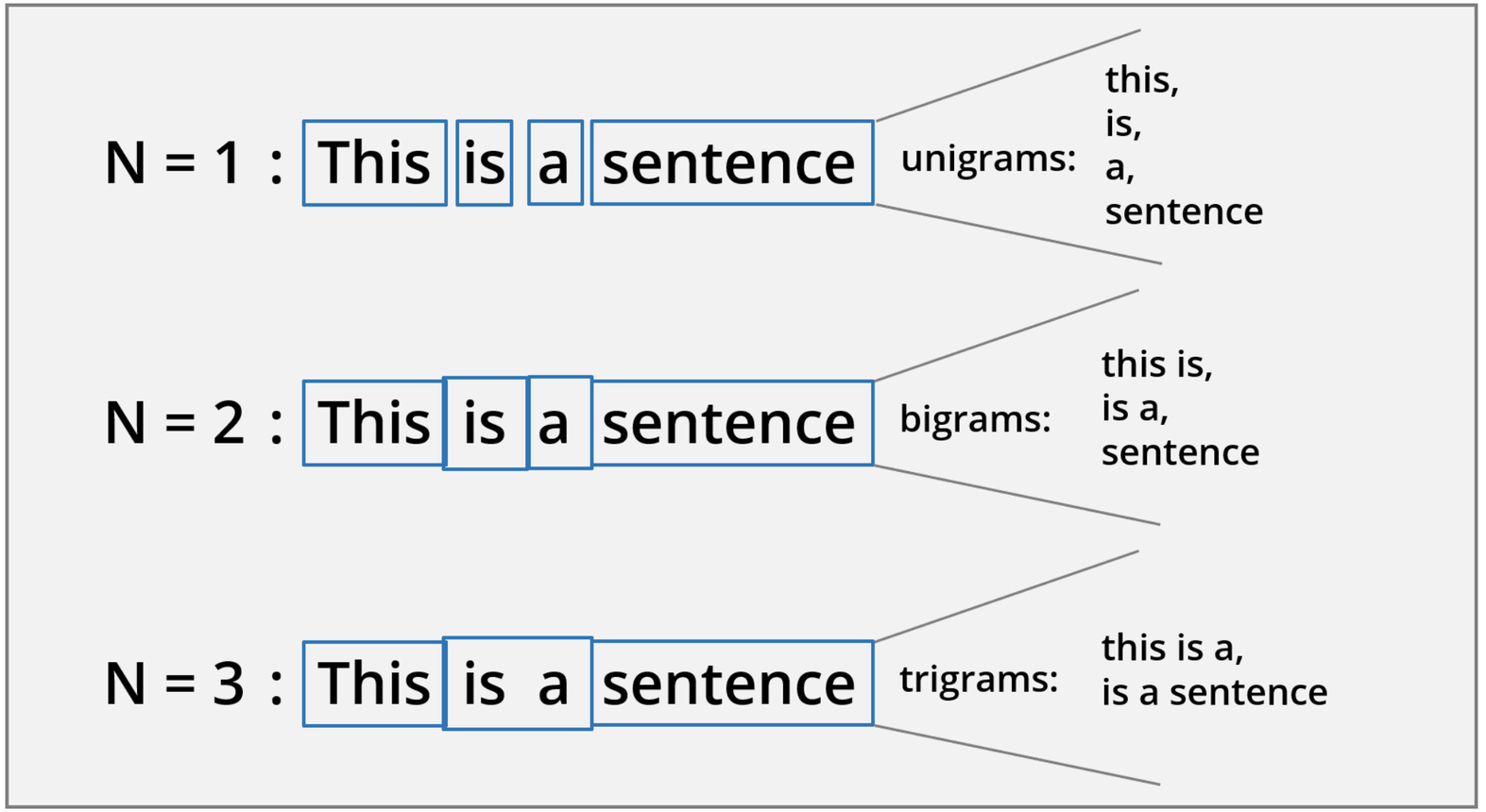
```
['Hello World!', 'This is sentence tokenizing.']
```

```
from nltk.tokenize import word_tokenize

EXAMPLE_WORD = "Hello World! This is word tokenizing."
print(word_tokenize(EXAMPLE_WORD))
```

```
['Hello', 'World', '!', 'This', 'is', 'word', 'tokenizing', '.']
```

N - grams

N-grams are combinations of adjacent words or letters of length *n* in the source text.



Stop Word Removal

1) Stop words are natural language words which have very little meaning, such as “a", “an", "and", “the", and similar words.

```
In [37]: import nltk
         from nltk.corpus import stopwords
         set(stopwords.words('english'))
```

```
Out[37]: {'a',
          'about',
          'above',
          'after',
          'again',
          'against',
          'ain',
          'all',
          'am',
          'an',
          'and',
          'any',
          'are',
          'aren',
          "aren't",
```

2) These are filtered out before processing of natural language data as they don’t reveal much information.

```
In [43]: from nltk.corpus import stopwords
         from nltk.tokenize import word_tokenize

         example_sent = "an Apple a day keeps diseases at bay."

         stop_words = set(stopwords.words('english'))
         word_tokens = word_tokenize(example_sent)

         filtered_sentence = [w for w in word_tokens if not w in stop_words]
         filtered_sentence = []

         for w in word_tokens:
          if w not in stop_words:
             filtered_sentence.append(w)

         print(filtered_sentence)
```

```
['Apple', 'day', 'keeps', 'diseases', 'bay', '.']
```

Stop words are language dependent

Stemming

Stemming involves reducing a word to *stem* or base (root) form by removing affixes.

```
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
ps = PorterStemmer()
new_text = "importantance of caving as explained by cavers"

words = word_tokenize(new_text)
for w in words:
    print(ps.stem(w))
```

```
important
of
cave
as
explain
by
caver
```

Form	Suffix	Stem
helps	-es	help
helping	-ing	help
helped	-ed	help
help	-	help
helper	-er	help

Removal of affixes

Various stemming algorithms : Porter stemmer, Lancaster stemmer, Snowball stemmer

Lemmatization

Lemmatization uses vocabulary list and morphological analysis (POS of a word) to get the root word.

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

print(lemmatizer.lemmatize("feet"))
print(lemmatizer.lemmatize("cacti"))
print(lemmatizer.lemmatize("geese"))

#Without a POS tag, lemmatizer assumes everything is a noun
print(lemmatizer.lemmatize("loving"))
print(lemmatizer.lemmatize("loving", 'v'))  #with POS tag
```

```
foot
cactus
goose
loving
love
```

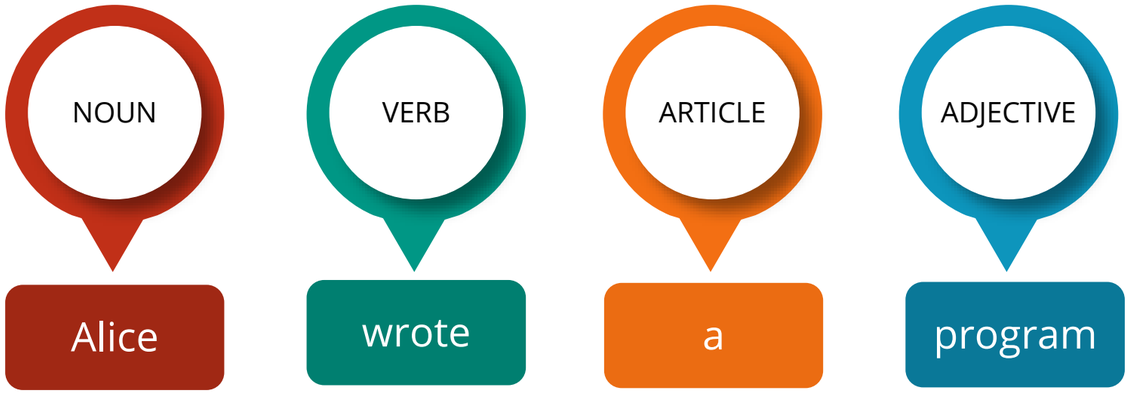
Form	Stem
saw	see (if token is verb)
saw	saw (if token is noun)

Context taken into account

Lemmatization uses **WordNet** database which has English words linked together by their semantic relationships

POS Tagging

POS tagging marks words in the corpus to a corresponding part of a speech tag based on its context and definition.



POS tags used in NLTK
• NNP proper noun, singular ‘Alice’ • NN noun, singular 'desk’ • RB adverb very, silently, • VBD verb, past tense took • JJ adjective ‘large’ • VBZ verb, 3rd person sing. present takes

POS Tagging (Contd.)

```
stop_words = set(stopwords.words('english'))
txt = '''
        Text mining, also referred to as text data mining,
       roughly equivalent to text analytics, is the process of deriving high-quality information from text.
       High-quality information is typically derived through the devising of patterns and trends through means
       such as statistical pattern learning.
      '''
tokenized = sent_tokenize(txt)
for i in tokenized:
    wordsList = nltk.word_tokenize(i)
    wordsList = [w for w in wordsList if not w in stop_words]
    tagged = nltk.pos_tag(wordsList)

    print(tagged)
```

```
[('Text', 'NNP'), ('mining', 'NN'), (',', ','), ('also', 'RB'), ('referred', 'VBD'), ('text', 'JJ'), ('data', 'NN'), ('mining',
'NN'), (',', ','), ('roughly', 'RB'), ('equivalent', 'JJ'), ('text', 'NN'), ('analytics', 'NNS'), (',', ','), ('process', 'N
N'), ('deriving', 'VBG'), ('high-quality', 'NN'), ('information', 'NN'), ('text', 'NN'), ('.', '.')]
[('High-quality', 'NNP'), ('information', 'NN'), ('typically', 'RB'), ('derived', 'VBD'), ('devising', 'VBG'), ('patterns', 'NN
S'), ('trends', 'NNS'), ('means', 'VBZ'), ('statistical', 'JJ'), ('pattern', 'NN'), ('learning', 'NN'), ('.', '.')]
```

POS Tags are useful for lemmatization, in building NERs and extracting relations between words

Named Entity Recognition (NER)

To further elaborate on the geographical trends, North America LOC has procured more than 50% PERCENT of the global share in 2017 DATE and has been leading the regional landscape of AI GPE in the retail market. The U.S. GPE has a significant credit in the regional trends with over 65% PERCENT of investments (including M&As, private equity, and venture capital) in artificial intelligence technology. Additionally, the region is a huge hub for startups in tandem with the presence of tech titans, such as Google ORG , IBM ORG , and Microsoft ORG .

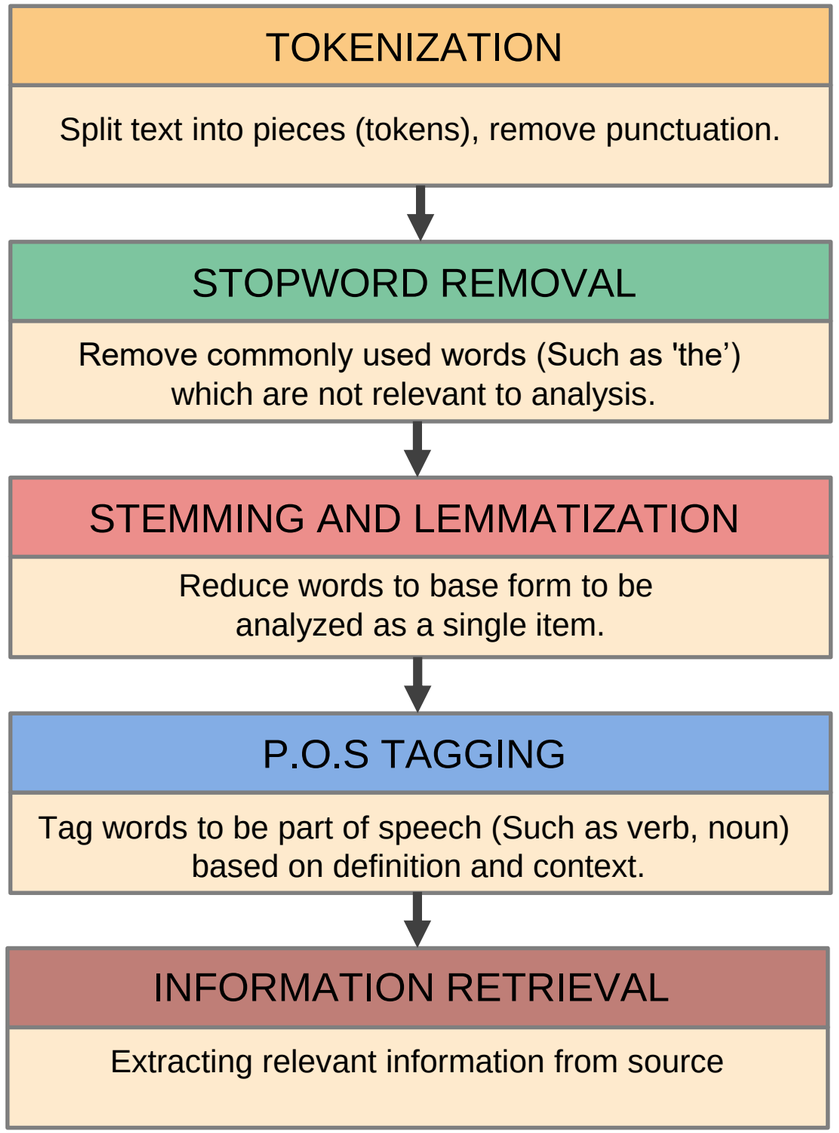
NER seeks to extract a real-world entity from the text and sort it into pre-defined categories such as the names of persons, organizations, locations, etc.

Named Entity Recognition (NER) (Contd.)

```
import nltk

doc = '''Google is an America multinational technology company that specializes in
-related services and products, which include online advertising technologies, search engine, cloud computing,
, and hardware. it was founded in 1998 by Larry Page and Sergey Brin while they were Ph.D. students
at Stanford University in California'''
# tokenize doc
tokenized_doc = nltk.word_tokenize(doc)
tagged_sentences = nltk.pos_tag(tokenized_doc)
ne_chunked_sents = nltk.ne_chunk(tagged_sentences)

# extract all named entities
named_entities = []
for tagged_tree in ne_chunked_sents:
    if hasattr(tagged_tree, 'label'):
        entity_name = ' '.join(c[0] for c in tagged_tree.leaves())
        entity_type = tagged_tree.label()
        named_entities.append((entity_name, entity_type))
print(named_entities)
```

```
[('Google', 'GPE'), ('America', 'GPE'), ('Larry Page', 'PERSON'), ('Sergey Brin', 'PERSON'), ('Stanford University', 'ORGANIZAT
ION'), ('California', 'GPE')]
```

NLP Process Workflow



Assisted Practice

Text Extraction and Pre-processing

Duration: 15 mins.

Problem Statement: The Brown University Standard Corpus of Present-Day American English (Brown Corpus) was compiled in the 1960s as a general corpus in the field of corpus linguistics. It contains 500 samples of English-language text, totalling roughly one million words, compiled from works published in the United States in 1961.
You are required to work on the subset: ca10 document.

Objective:

- Implement different types of tokenizers
- Perform stemming and lemmatization
- Evaluate POS tags for the tokens
- Remove the stop words and apply NER

Access: Click on the Labs tab on the left side panel of the LMS. Copy or note the username and password that are generated. Click on the Launch Lab button. On the page that appears, enter the username and password in the respective fields, and click Login.

Unassisted Practice

Text Extraction and Pre-processing

Duration: 20 mins.

Problem Statement: The wiki corpus contains the full text of Wikipedia, and it contains 1.9 billion words in more than 4.4 million articles. You are provided with a subset of the same.

Objective:

- Implement different types of tokenizers
- Perform stemming and lemmatization
- Evaluate POS tags for the tokens
- Remove the stop words and print NER words

Access: Click on the Labs tab on the left side panel of the LMS. Copy or note the username and password that are generated. Click on the Launch Lab button. On the page that appears, enter the username and password in the respective fields, and click Login.

Step 1: Data Import and Tokenization

Code

```
import nltk
with open('wiki_corpus.txt', 'r') as myfile:
    data=myfile.read().replace('\n', '')
for i, line in enumerate(data.split('\n')):
    if i > 10: # Lets take a look at the first 10 ads.
        break
    print(str(i) + ':\t' + line)
```

```
from nltk import sent_tokenize, word_tokenize
sent_tokenize(data)
for sent in sent_tokenize(data):
    print(word_tokenize(sent))
```

Step 2: Stemming and Lemmatization

Code

```
single_tokenized_lowered = list(map(str.lower, word_tokenize(data)))
from nltk.stem import PorterStemmer
porter = PorterStemmer()

for word in single_tokenized_lowered :
    print(porter.stem(word))
```

```
from nltk.stem import WordNetLemmatizer
wnl = WordNetLemmatizer()

for word in single_tokenized_lowered:
    print(wnl.lemmatize(word))
```

Step 3: Evaluate POS Tags

Code

```
stop_words = set(stopwords.words('english'))
tokenized = sent_tokenize(data)
for i in tokenized:
    wordsList = nltk.word_tokenize(i)
    wordsList = [w for w in wordsList if not w in stop_words]
    tagged = nltk.pos_tag(wordsList)
    print(tagged)
```

```
[('Apple', 'NNP'), ('Inc.', 'NNP'), ('American', 'NNP'), ('multinational', 'NNP'), ('technology', 'NN'), ('company', 'NN'),
eadquartered', 'VBD'), ('Cupertino', 'NNP'), (',', ','), ('California', 'NNP'), (',', ','), ('designs', 'NNS'), (',', ','),
```

Step 4: Remove Stop Words and NER

Code

```
with open('nltk_unassited.txt', 'r') as f:
    sample = f.read()
sentences = nltk.sent_tokenize(sample)
tokenized_sentences = [nltk.word_tokenize(sentence) for sentence in
sentences]
tagged_sentences = [nltk.pos_tag(sentence) for sentence in
tokenized_sentences]
chunked_sentences = nltk.ne_chunk_sents(tagged_sentences, binary=True)
def extract_entity_names(t):
    entity_names = []
 return entity_names
```

Step 4: Remove Stop Words and NER (Contd.)

Code

```
entity_names = []
for tree in chunked_sentences:
    entity_names.extend(extract_entity_names(tree))
print (set(entity_names))
```

```
{'iTunes', 'Jobs', 'Apple Music', 'iPod', 'Apple', 'Tim Cook', 'iLife', 'California', 'iTunes Store', 'iPhone', 'Steve Jobs',
'Steve Wozniak', 'NeXT', 'iOS', 'Apple Computer', 'HomePod', 'Mac', 'iPad', 'CEO Gil Amelio', 'Apple Inc.', 'Ronald Wayne', 'Lo
gic Pro', 'Wozniak', 'Apple II', 'Final Cut Pro', 'Safari', 'Xcode', 'American', 'CEO', 'Apple Watch', 'iOS App Store', 'iWor
k', 'Mac App Store', 'iCloud', 'Microsoft', 'Cupertino', 'macOS'}
```

Text Mining

Topic 4: Structuring Sentences

Syntax

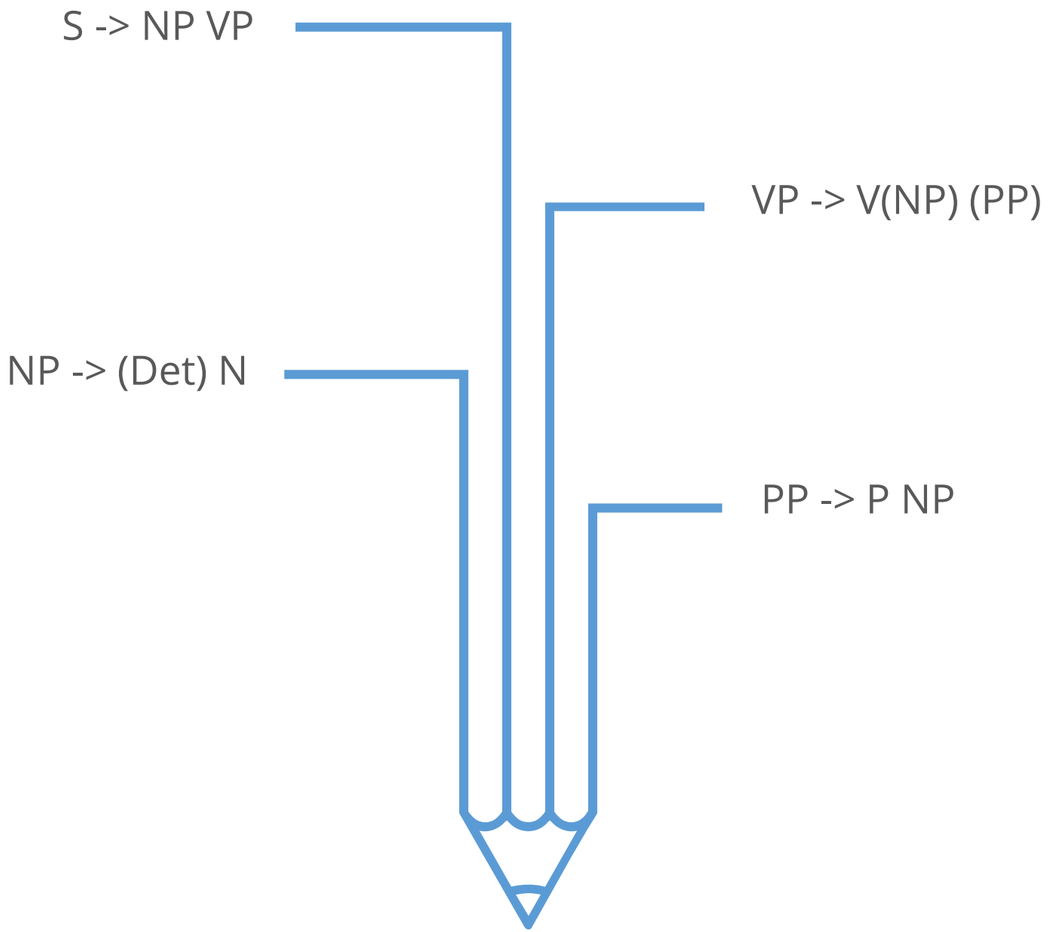
Syntax is the grammatical structure of the sentences.

```
//two string variables
string mySimpleName = inputTextBox.Text;

//a literal string and a string variable
string myString = "This is a string literal" + mySimpleName;

//an integer variable and a literal integer
int myInt = 5;
```

Can be interpreted as syntax similar to the ones you use while writing codes.

Phrase Structure Rules

Phrase structure rules are rewrite rules that generate phrase structure.



Syntax Tree Parsing

A tree representation of syntactic structure of formulation of sentences or strings.

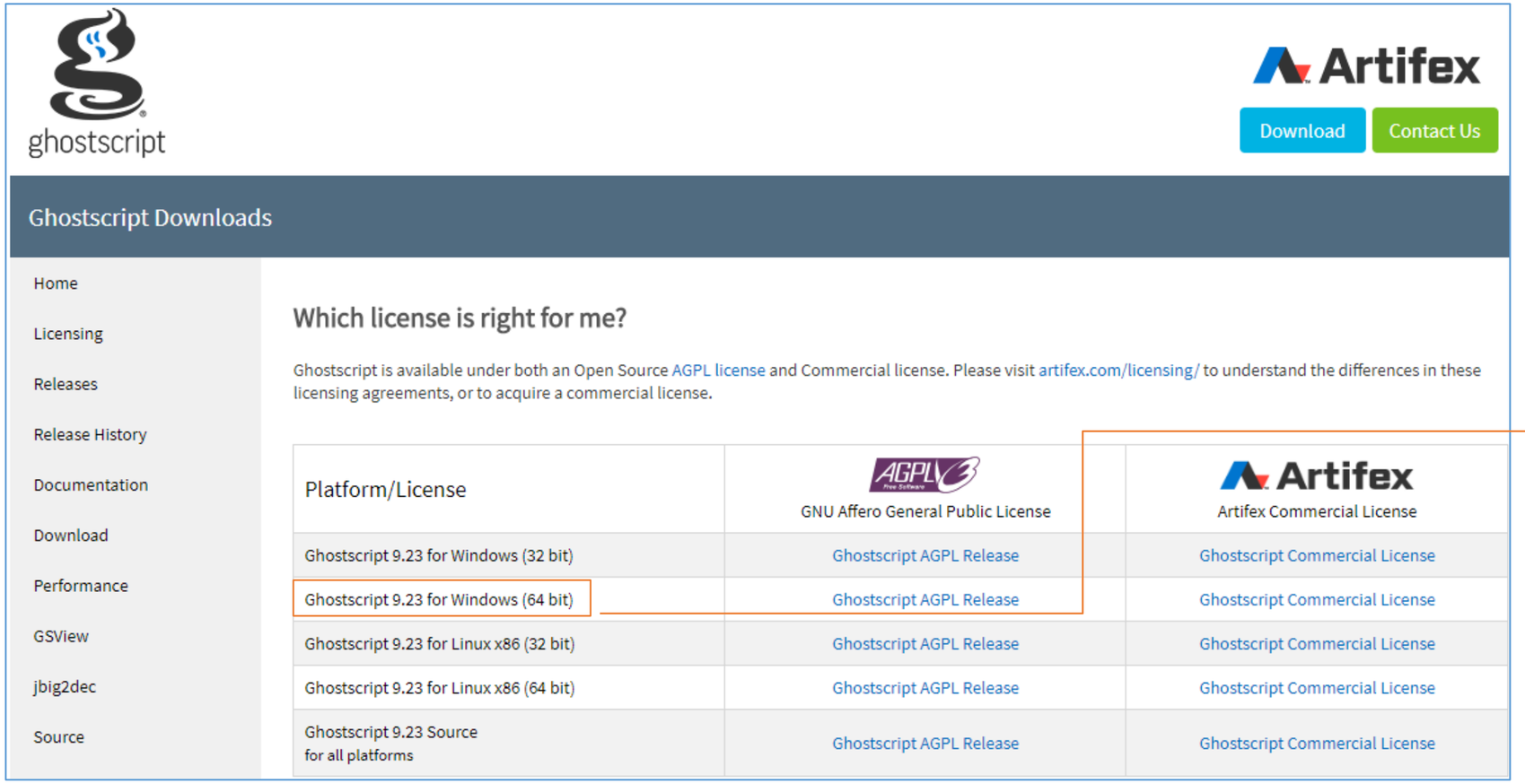
Consider the example: “**The factory employs 12.8 percent of Bradford County.”**

Syntax Parsing the above statement:

- A tree is produced that might help you understand that the subject of the sentence. “the factory”
- The predicate is “employs”, and the target is “12.8 percent” which in turn is modified by “Bradford County.”
- Syntax parses are often a first step toward deep information extraction or semantic understanding of text.

Rendering Syntax Trees

In order to render syntax trees in your notebook, you need to install **ghostscript** (a rendering engine) from the link: https://ghostscript.com/download/gsdnld.html

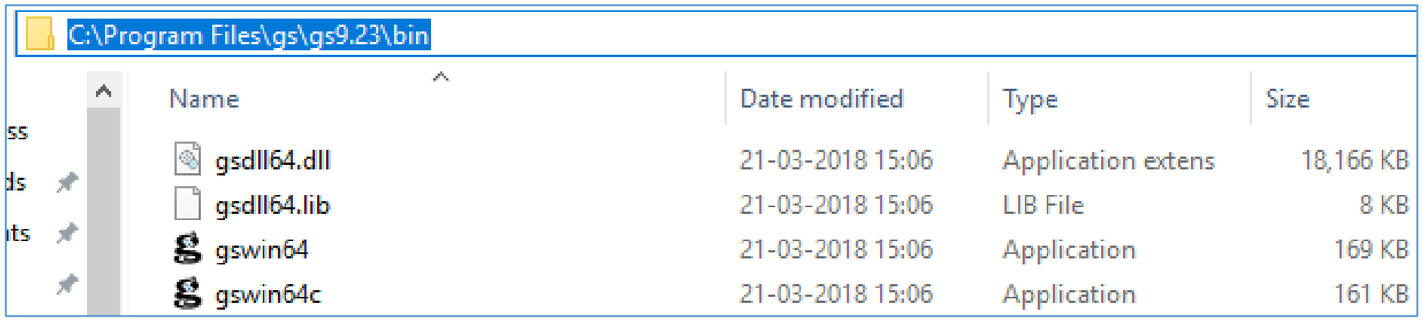


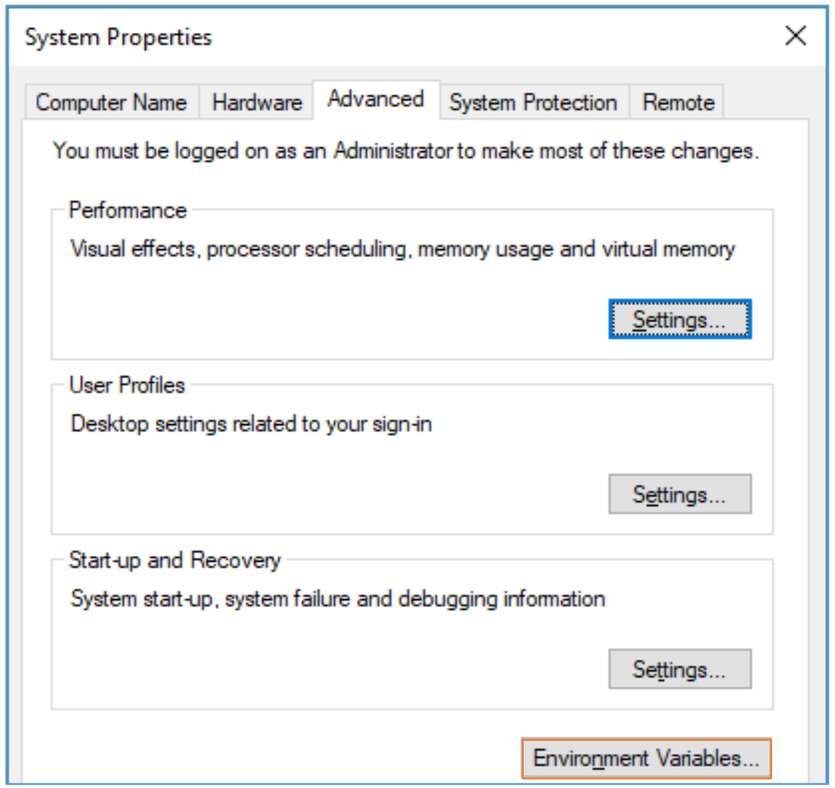
Download the corresponding .exe file based on your system configuration

Setting Up Environment Variables

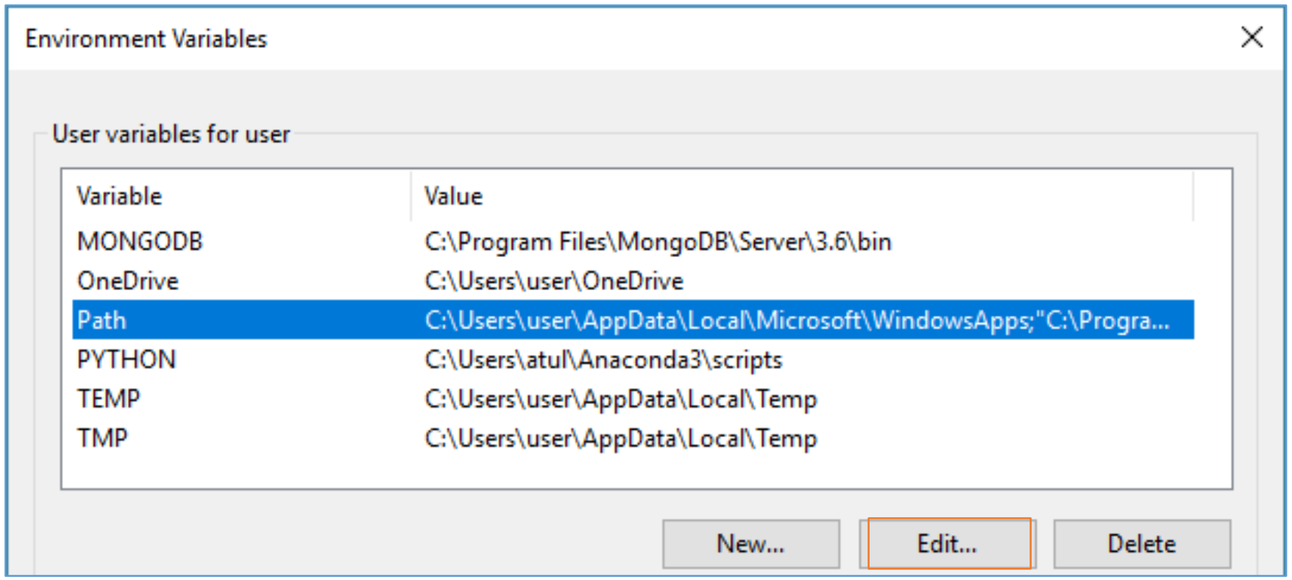
Once you have downloaded and installed the file:

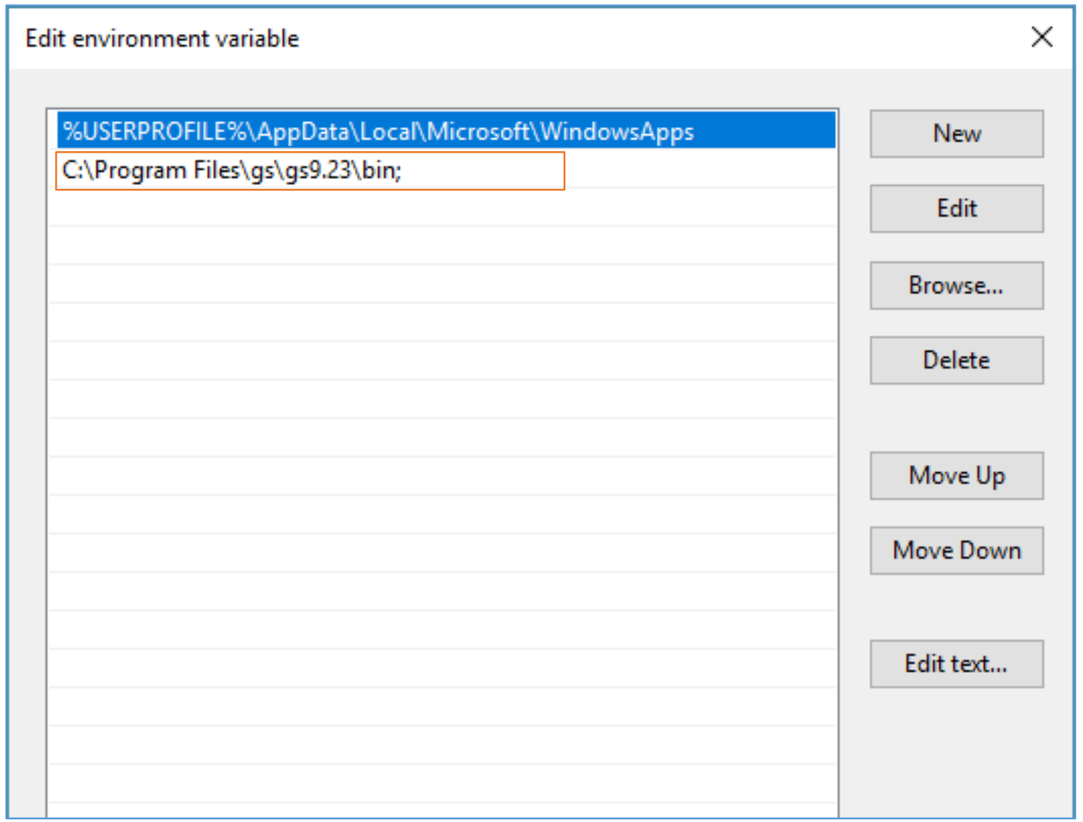
-> go to the folder where it is installed -> open the bin folder -> add the path to the bin folder in your environment variables





Setting Up Environment Variables (Contd.)





Setting Up Path Variable

Now, you will have to modify the path of the environment variable.

Code

```
import os
path_to_gs = "C:\\Program Files\\gs\\gs9.23\\bin"

os.environ['PATH']+=os.pathsep + path_to_gs #modifying environment
variable
```

Now, let's start with analyzing sentence structure.

Chunking and Chunk Parsing

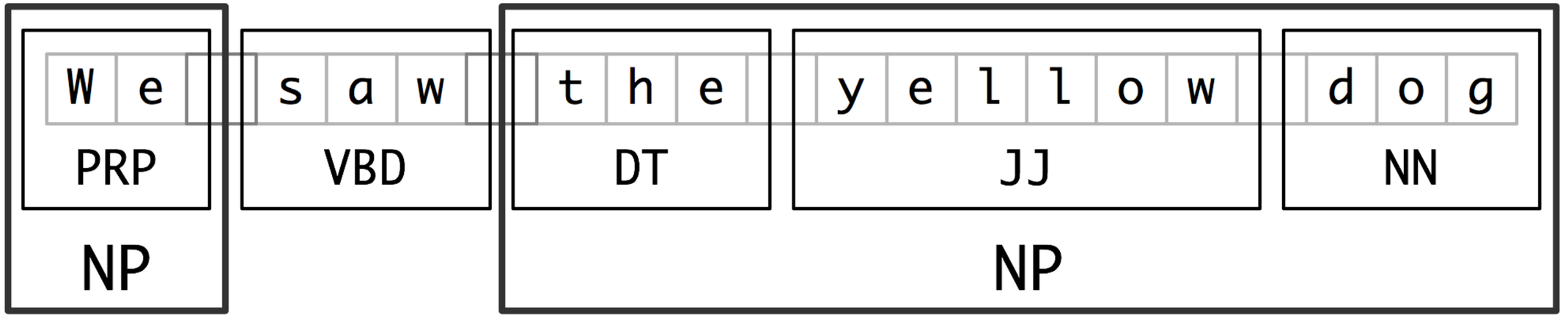
Chunking

- Segmentation: identifying tokens
- Labeling: identifying the correct tag

Chunk Parsing

- Segmentation: identifying strings of tokens
- Labeling: identifying the correct chunk type

Chunking: An Example



Source: nltk.org

You can see here: Yellow(adjective), dog(noun), and the determiner are chunked together into a noun phrase (NP)

Chunking Using Python

Let's consider the sentence below:

```
sent = "The little mouse ate the fresh cheese"
```

Code

```
sent_tokens = nltk.pos_tag(word_tokenize(sent))
sent_tokens
```

```
[('The', 'DT'),
 ('little', 'JJ'),
 ('mouse', 'NN'),
 ('ate', 'VB'),
 ('the', 'DT'),
 ('fresh', 'JJ'),
 ('cheeze', 'NN')]
```

NP Chunk and Parser

You will now create grammar from a noun phrase and will mention the tags you want in your chunk phrase within { }. Here you have created a regular expression matching the string.

Code

```
grammar_np = r"NP: {<DT>?<JJ>*<NN>}"
```

You will now have to parse the chunk. Therefore, you will create a chunk parser and pass your noun phrase string to it.

Code

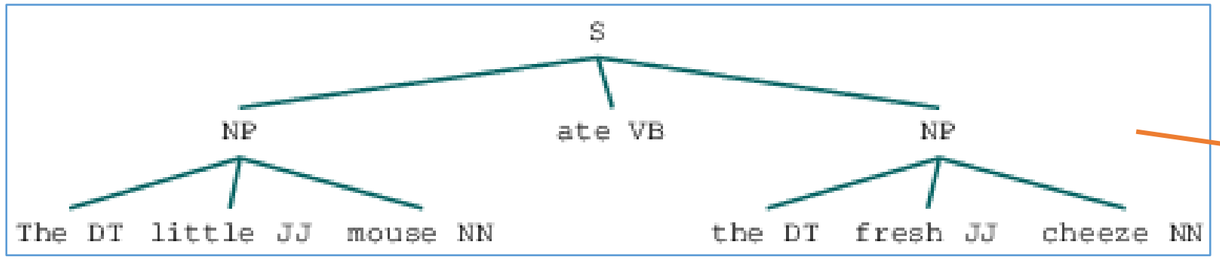
```
chunk_parser = nltk.RegexpParser(grammar_np)
```

NP Chunk and Parser (Contd.)

The parser is now ready. You will use the parse ( ) within your chunk parser to parse your sentence.

Code

```
chunk_result = chunk_parser.parse(sent_tokens)
chunk_result
```



The tokens that matched our regular expressions are chunked together into noun phrases(NP)

VP Chunk and Parser

Create a verb phrase chunk using regular expressions.

Code

```
grammar_vp = r"vp: {<PRP>?<VB|VBD|VBZ|VBG>*<RB|RBR>?}"
```

You'll now create another chunk_parser and pass the verb phrase string to it.

Code

```
chunk_parser2 = nltk.RegexpParser(grammar_vp)
```

VP Chunk and Parser (Contd.)

Create another sentence and tokenize it. Add POS tags to it.

Code

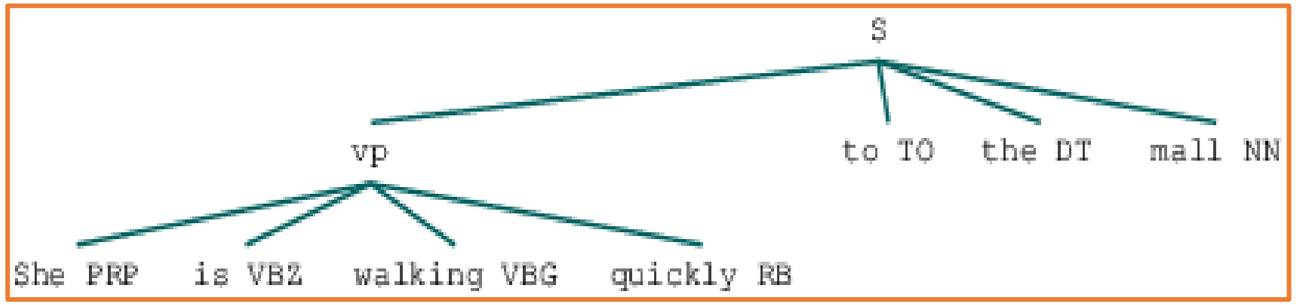
```
sent3 = "She is walking quickly to the mall“
sent_tokens3 = nltk.pos_tag(word_tokenize(sent3))
```

Now, use the new verb phrase parser to parse the tokens and run the results.

Code

```
chunk_result3 = chunk_parser2.parse(sent_tokens3)
chunk_result3
```

VP Chunk and Parser (Contd.)

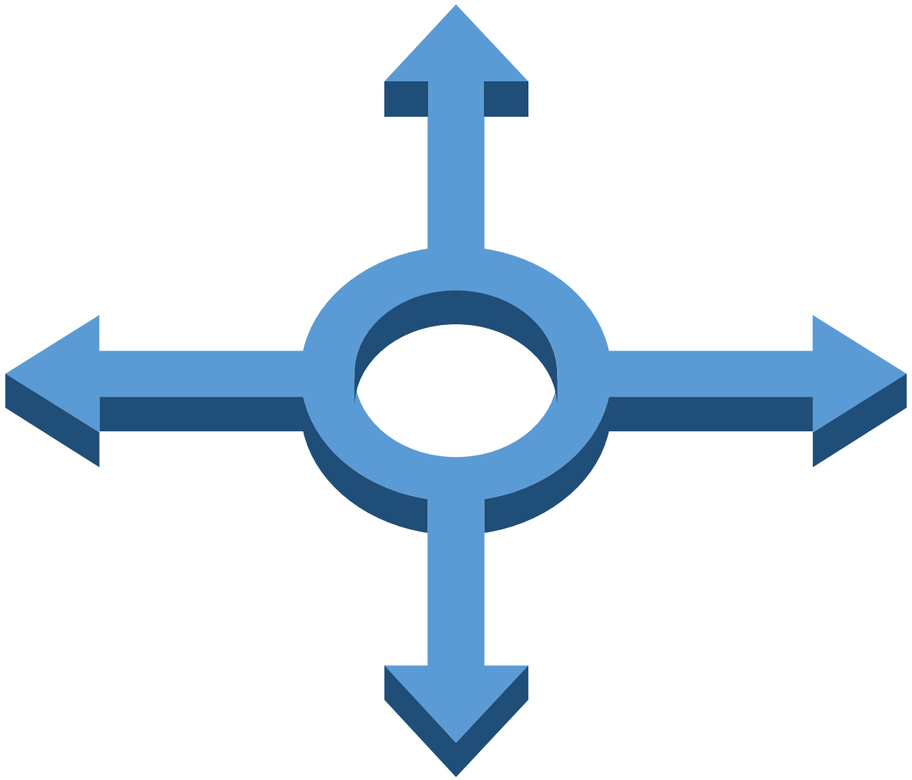


A verb parser where a pronoun followed by two verbs and an adverb are chunked together into a verb phrase

Chinking

Chinking is the process of removing a sequence of tokens from a chunk

If the sequence of tokens spans an entire chunk, then the whole chunk is removed



If the sequence of tokens appears in the middle of the chunk, these tokens are removed, leaving two chunks where there was only one before

If the sequence is at the beginning or end of the chunk, these tokens are removed, and a smaller chunk remains

Create Chink Grammar

Consider you create a chinking grammar string containing three things:

- Chunk name
- The regular expression sequence of a chunk
- The regular expression sequence of your chink

Code

```
chink_grammar = r"""
    chk_name: #chunk name

    {<PRP>?<VB|VBD|VBZ|VBG>*<RB|RBR>?} #chunk regex sequence

    }<RB>+{ #chink regex sequence - adverb
"""
```

Inside chinking block with } {, you have created one or more adverbs

Create Chink Parser

You will now create a parser from nltk.RegexpParser and pass the chink_grammar to it.

Code

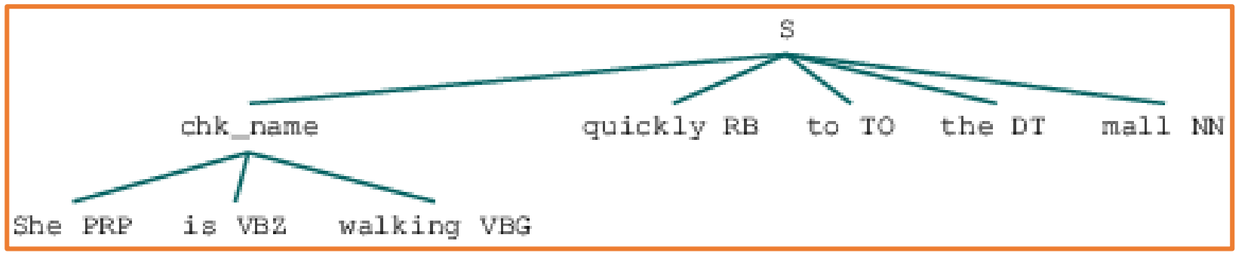
```
chink_parser = nltk.RegexpParser(chink_grammar)
```

Now, use the new chink parser to parse the tokens (sent3) and run the results.

Code

```
chink_parser.parse(sent_tokens3)
```

Create Chink Parser (Contd.)



While comparing the syntax tree of chink parser with that of the original chunk, you can see that the token quickly (adverb) is chinked out of the chunk

Text Mining

Topic 5: Context Free Grammar (CFG)

CFG

- A context-free grammar is a 4-tuple ($\sum$, NT, R, S), where:
  - $\sum$ is an alphabet (each character in $\sum$ is called terminal)
  - NT is a set (each element in NT is called nonterminal)
  - R, the set of rules, is a subset of NT $\times$ ($\sum \cup$ NT)*
  - S, the start symbol, is one of the symbols in NT
- Generates a language L by capturing constituency and ordering

CFG: Example

S -> NP VP

NP -> Det Nominal

Nominal -> Noun

VP -> Verb

Det -> a

Noun -> flight

Implementing CFG

Consider the string below, where you have certain rules:

Code

```
CFG_grammar = nltk.CFG.fromstring("""

S -> NP VP
VP -> V N
V -> "saw"|"met"
NP -> "John"|"Jim"
N -> "dog"|"cat"
""")
```

It should have a noun phrase followed by a verb phrase. A verb phrase is a verb followed by a noun. A verb can either be saw or met. Noun phrase can either be John or Jim, and a noun can either be a dog or a cat

Implementing CFG (Contd.)

Check the possible list of sentences that can be generated using the rules:

Code

```
for sentence in generate(CFG_grammar):
    print(" ".join(sentence))
```

```
John saw dog
John saw cat
John met dog
John met cat
Jim saw dog
Jim saw cat
Jim met dog
Jim met cat
```

Implementing CFG (Contd.)

You can check the different rules of grammar for the sentence formation using the productions( ):

Code

```
CFG_grammar.productions()
```

```
[S -> NP VP,
 VP -> V N,
 V -> 'saw',
 V -> 'met',
 NP -> 'John',
 NP -> 'Jim',
 N -> 'dog',
 N -> 'cat']
```

Assisted Practice

Structuring Sentences

Duration: 20 mins.

Problem Statement: ABC Company wants to perform text analysis for one of its dataset.
The dataset has been taken from Kaggle. (https://www.kaggle.com/crowdflower/twitter-airline-sentiment/home)
This dataset has tweets about six US Airlines along with their sentiments: positive, negative, and neutral. You are provided with this dataset named "Tweets.csv". It has tweets in 'text' column and sentiments in "airline_sentiment" column.

Objective: Retrieve all tags starting from "@" in the entire dataset and save in a file called "References.txt". Extract all noun phrases from the dataset and save them in different lines in a file named "Noun Phrases for <airline_sentiment> Review .txt" (You can choose your own grammar for noun phrase).
Here <airline_sentiment> will have three different values: positive, negative, and neutral. Hence, three files will be created.

Access: Click on the Labs tab on the left side panel of the LMS. Copy or note the username and password that are generated. Click on the Launch Lab button. On the page that appears, enter the username and password in the respective fields, and click Login.

Unassisted Practice

Structuring Sentences

Duration: 15 mins.

Problem Statement: ABC Company wants to perform some text analysis and make visualization for one it's dataset. The dataset has been taken from Kaggle. (https://www.kaggle.com/crowdflower/twitter-airline-sentiment/home). This is a dataset having tweets about six US Airlines along with their sentiments: positive, negative, and neutral. You are provided with this dataset named "Tweets.csv". It has tweets in 'text' column and sentiments in 'airline_sentiment' column.

Objective: Extract all verb phrases from their dataset and save them in different lines in a file named "Verb Phrases for <airline_sentiment> Review .txt" (You can choose your own grammar for noun phrase). Here <airline_sentiment> will have three different values: positive , negative, and neutral. Hence, three files will be created. For each sentiment, make a well labeled pie chart showing the distribution of noun phrases and verb phrases of that sentiment from the data set. Use the files created above to get the frequencies.

Note: This practice is not graded. It is only intended for you to apply the knowledge you have gained to solve real-world problems.

Access: Click on the Labs tab on the left side panel of the LMS. Copy or note the username and password that are generated. Click on the Launch Lab button. On the page that appears, enter the username and password in the respective fields, and click Login.

Step 01

Code

```
import pandas as pd
df=pd.read_csv('Tweets.csv')
df=df[['text','airline_sentiment']]
```

Step 02

```
import nltk
def GetVerbPhrases(s):
    try:
        sentences = nltk.sent_tokenize(s)
        sentences = [nltk.word_tokenize(sent) for sent in sentences]
        sentences = [nltk.pos_tag(sent) for sent in sentences]
    except:
        return []

    else:

        grammar=r"VP:{<VB|VBD|VBG|VBZ|VBP|VBN>*<VB|VBD|VBG|VBZ|VBP|VBN><RB|RBR>*<RB|RBR>}"

        cp = nltk.RegexpParser(grammar)


        noun_phrases_list = [[' '.join(leaf[0] for leaf in tree.leaves())
                              for tree in cp.parse(sent).subtrees()
                              if tree.label()=='VP']
                              for sent in sentences]
        return noun_phrases_list
```

Step 03

Code

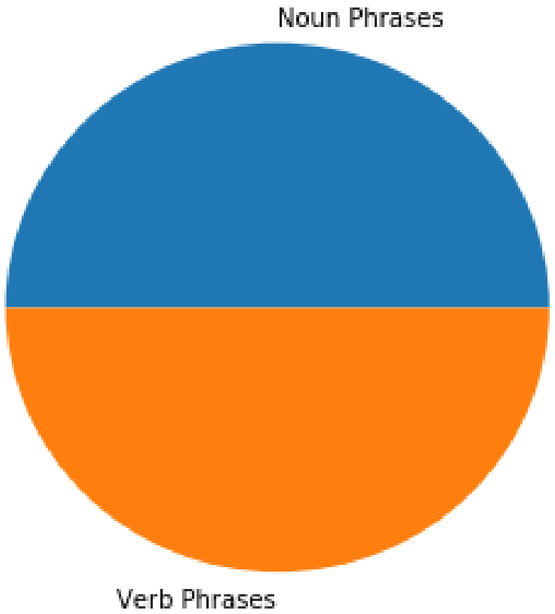
```
import itertools
for group,sub in df.groupby('airline_sentiment'):
    verb_phrases=map(lambda x: GetVerbPhrases(x),sub['text'])
    verb_phrases=list(itertools.chain.from_iterable(verb_phrases))
    AllVerbPhrases=set(list(itertools.chain.from_iterable(verb_phrases)))
    filename="Verb Phrases for "+str(group)+" Review .txt"
    file=open(filename,'a')
    for each in AllVerbPhrases:
        file.write(each+"\n")
    file.close()
```

Step 04

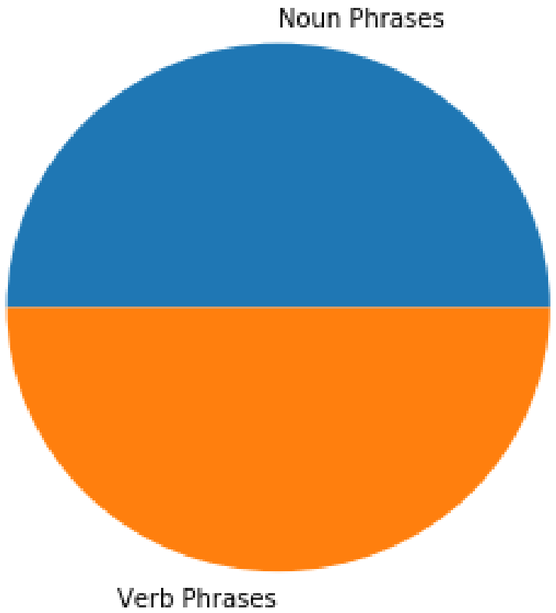
```
#Plotting a pie chart
def PlotPieChart(sentiment):
    noun_phrase_file="Noun Phrases for "+str(sentiment)+" Review .txt"
    verb_phrase_file="Verb Phrases for "+str(sentiment)+" Review .txt"
    noun_phrase_count=len(noun_phrase_file.split("\n"))
    verb_phrase_count=len(verb_phrase_file.split("\n"))
    counts=[noun_phrase_count,verb_phrase_count]
    labels=['Noun Phrases','Verb Phrases']

    import matplotlib.pyplot as plt
    %matplotlib inline
    plt.figure(figsize=(5,5))
    plt.pie(counts,labels=labels)
    plt.title("Phrases Distribution for "+str(sentiment)+" Review.")
    plt.show()

for each in df['airline_sentiment'].unique():
    PlotPieChart(each)
```

Step 04 (Contd.)

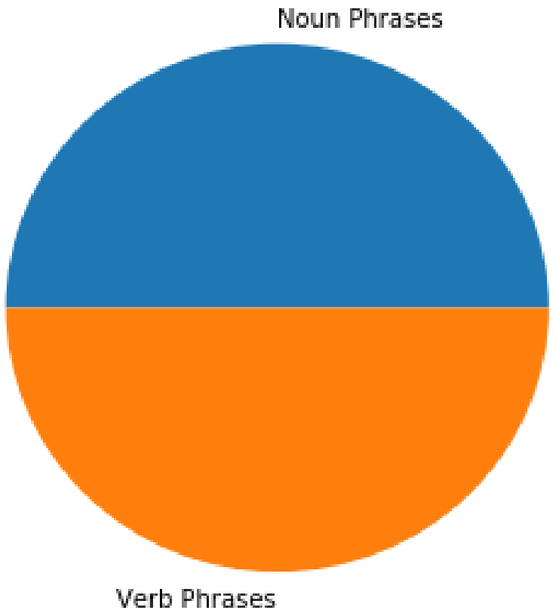
Phrases Distribution for neutral review



Phrases Distribution for negative review



Phrases Distribution for positive review



Key Takeaways

Now, you are able to:

- Explain text mining
- Execute text processing tasks

Knowledge Check

Knowledge Check 1	**Which of the following describes the file representation given below?** `[the/DT little/JJ cat/NN] sat/VBD on/IN [the/DT mat/NN]`

a. Chunked Text

b. Tagged Text

c. Chunked and Tagged Text

d. Chinked Text

Knowledge Check	Which of the following describes the file representation given below?
1	`[the/DT little/JJ cat/NN] sat/VBD on/IN [the/DT mat/NN]`

a. Chunked Text

b. Tagged Text

c. Chunked and Tagged Text

d. Chinked Text

The correct answer is **c. Chunked and Tagged Text**

The above text is segregated as well as POS tagged.

Knowledge Check 2	**Tokenization, is a way to:**

a. Find the grammar of the text

b. Split text data into words, phrases, and idioms

c. Analyze the sentence structure

d. Find ambiguities

Knowledge Check 1	**Tokenization, is a way to:**

a. Find the grammar of the text

b. Split text data into words, phrases, and idioms

c. Analyze the sentence structure

d. Find ambiguities

The correct answer is **b. Split text data into words, phrases, and idioms**

Splitting text data into words, phrases, and idioms is known as tokenization and each individual word is known as a token.

Lesson-End Project

Duration: 10 mins.

Problem Statement: Consider the FIFAWORLDCUP2018.txt file.

Objective: 1. Write separate Python functions that accept a string, a number 'n', and returns the following:

- N most frequent nouns (Take function name as "GetNMostFrequentNouns")
- N most frequent verbs (Take function name as "GetNMostFrequentVerbs")
- N most frequent delimiters (Take function name as "GetNMostFrequentDelimiters")
- N most frequent prepositions (Take function name as "GetNMostFrequentPrepositions")

Run all the functions on the file "FIFAWorldCup2018.txt" and print the results.

2. Write a Python function that accepts a string and prints the first sentence in the string along with its syntax tree.
Take function name as "PrintSyntaxTree"
Run this function on the file "FIFAWorldCup2018.txt"

Lesson-End Project

Duration: 10 mins.

Objective: 3. Write a Python function that accepts and returns a string using regular expressions:

- Text from the string after removing all the punctuations (Take function name as “TextAfterRemovingPunctuations”)
- Text from the string after removing all the numbers/digits (Take function name as “TextAfterRemovingDigits”)
- All the words that begin with the capital letter (Take function name as “AllCapitalizedWordsFromText”)
- All the emails from the string (Take function name as “AllEmailsFromText”)

Run all the above functions on the file “FIFAWorldCup2018.txt” and print the results.

4. Write Python functions that accept a string as an input and return the following chunks:

- Phrases having proper nouns followed by verbs (Take function name as “ChunkingVer1”)
- Verb phrases having verbs followed by adjectives (Take function name as “ChunkingVer2”)
- Noun phrases having determiners followed by nouns (Take function name as “ChunkingVer3”)
- Verb phrases having verbs followed by adverbs (Take function name as “ChunkingVer4”)
- Phrases having delimiter, adjectives, and nouns in the respective order. (Take function name as “ChunkingVer5”)
- Noun phrases having nouns and adjectives, terminated with nouns. (Take function name as “ChunkingVer5”)

Run all the functions for the first sentence in the file “FIFAWorldCup2018.txt” and print the results.

Lesson-End Project

Duration: 10 mins.

Objective: Make a content-free grammar having the following rules:

- Noun phrases are followed by verb phrases
- Verb phrase can have:
  - Verb and noun phrases
  - Verb, noun phrases, and preposition phrases
- Noun phrases can have:
  - Delimiters followed by noun
- Preposition phrase has a preposition followed by a noun phrase

The delimiters, verbs, prepositions, and nouns for the grammar should be the 2 most frequent words of each type from “FIFAWorldCup2018.txt”. Generate the CFG for “FIFAWorldCup2018.txt” file and save them in a file named “CFG.txt”

Access: Click the Labs tab in the left side panel of the LMS. Copy or note the username and password that are generated. Click the Launch Lab button. On the page that appears, enter the username and password in the respective fields and click Login.

Thank You