NLP Mini-Project: Constituency Parsing and Dependency Parsing

Aryan Chouhan
60004180009
BE – A

Problem Statement

To carry out and visualize Constituency Parsing and Dependency Parsing

Platform/Languages

Python, NLTK, CoreNLP

Abstract

Sentence parsing can be helpful in understanding the meaning, structure, and syntactical relationships in sentences. Two common types of parsing are Constituency Parsing (also known as Syntactical Parsing) and Dependency Parsing. They are used in word processing systems, information extraction systems, and grammar checking.

A constituency parsed tree displays the syntactic structure of a sentence using context-free grammar unlike dependency parsing which relies on dependency grammar. The sentence is broken down into sub-phrases also known as constituents. These sub-phrases belong to a specific category of grammar like NP (noun phrase) and VP (verb phrase).

Dependency parsing is the process of analyzing the grammatical structure of a sentence based on the dependencies between the words in a sentence. Dependency tags are used to represent the relationship between the head and the dependent words in a sentence. Dependency trees are often more concise than constituency trees because they only display grammar between a governor and is dependents.

Working of Project

Constituency parsing can be achieved with multiple algorithms. The Cocke-Kasami-Younger (CKY) algorithm (a probabilistic bottom-up approach) is a

popular approach, along with the Probabilistic Context-Free Grammars (PCFGs) algorithm. A major disadvantage of CKY is that the grammar must be of Chomsky Normal Form (CNF).

This project relies on Stanford's CoreNLP library which uses a Shift-Reducer algorithm for syntactical parsing, primarily because it is more efficient than PCFG or CKY. Shift-reducing is a stack-based approach to parsing using context-free grammar. All tokens in the sentence are pushed onto the stack, then the top two tokens are popped off and matched to grammar rules and placed back onto the stack in their reduced form.

Common algorithms used in dependency parsing are treebank searching algorithms like Arc-Eager or Beam Search, and graph-based approaches such as Edge-Based. This project utilizes Stanford's CoreNLP library which relies on a Neural Network-based approach.

Stanford's CoreNLP is a Java-based suite of NLP tools. A server is setup and accessed via the Python library NLTK, which provides a Python-based interface to the suite.

Constituency Parsing: https://stanfordnlp.github.io/CoreNLP/parse.html

Dependency Parsing: https://stanfordnlp.github.io/CoreNLP/depparse.html

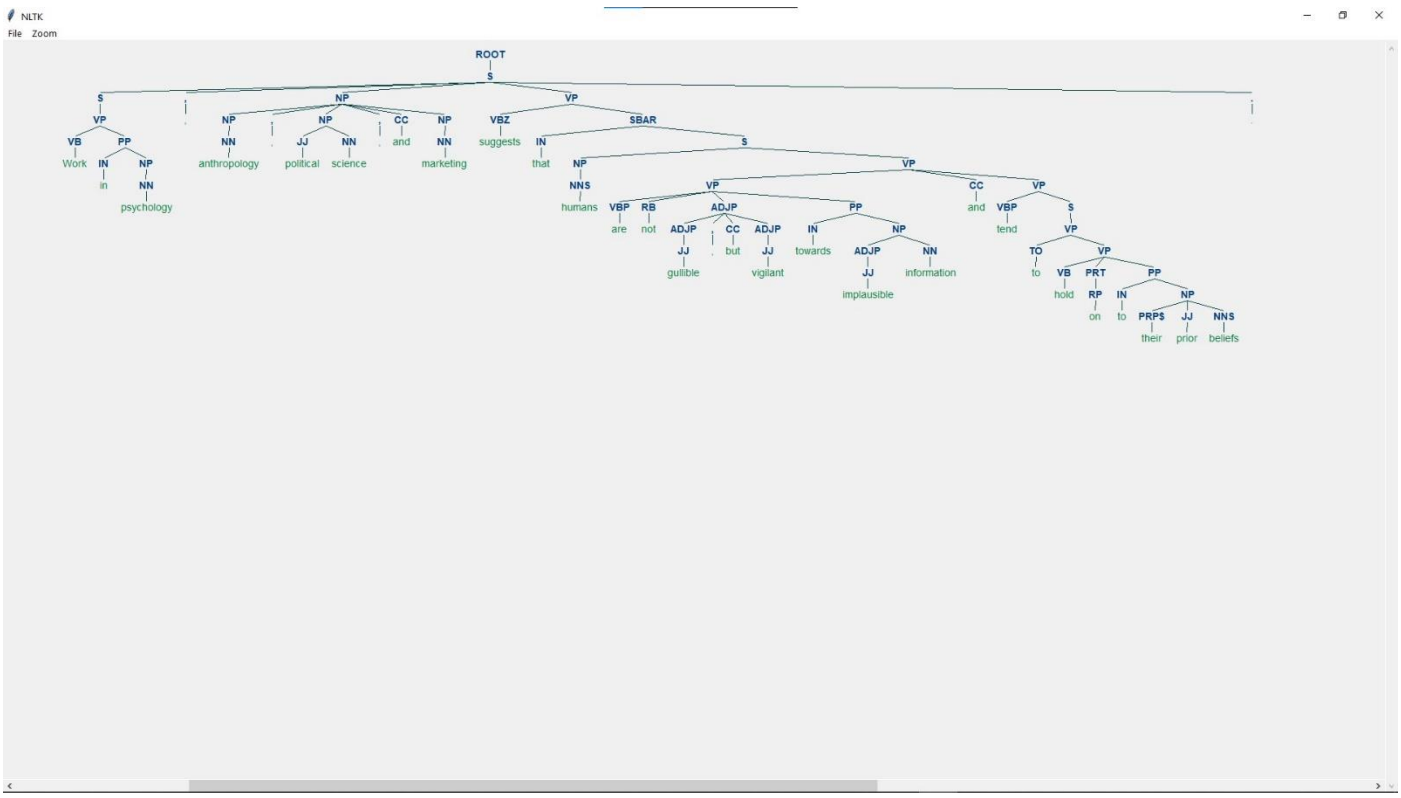
Source Code

```
import os

from nltk.parse.corenlp import CoreNLPServer
from nltk.parse.corenlp import CoreNLPParser
from nltk.parse.corenlp import CoreNLPDependencyParser

java_path = "C:/Program Files/Java/jdk-18/bin/java.exe"
os.environ["JAVAHOME"] = java_path

STANFORD = os.path.join("models")

# Create the server
server = CoreNLPServer(
    os.path.join(STANFORD, "stanford-corenlp-4.4.0.jar"),
    os.path.join(STANFORD, "stanford-corenlp-4.4.0-models.jar"),
)

# Start the server
server.start()
```

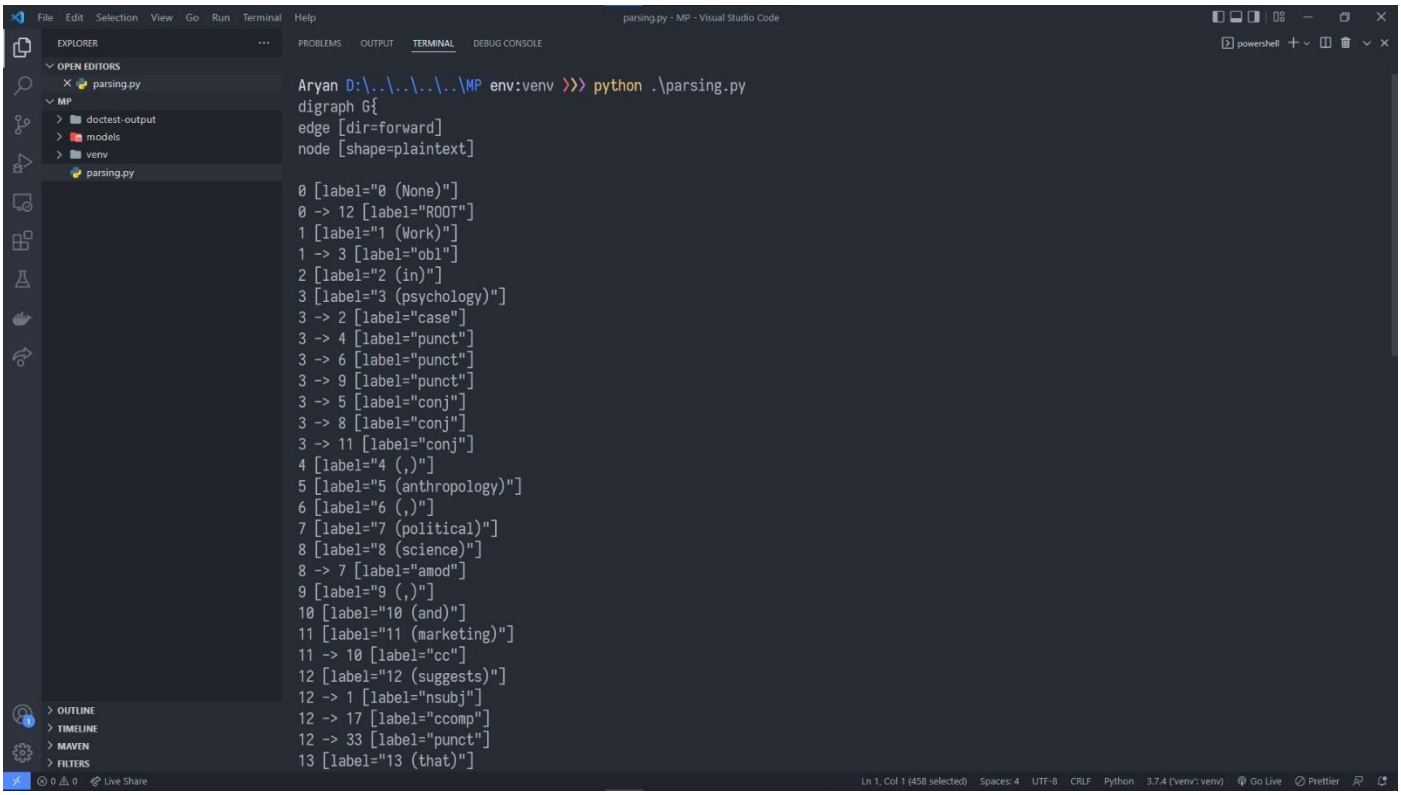
```
# Constituency parsing
parser = CoreNLPParser()
parse = next(
    parser.raw_parse(
        "Work in psychology, anthropology, political science, and marketing suggests that
humans are not gullible, but vigilant towards implausible information and tend to hold on to
their prior beliefs."
    )
)
parse.draw()

# Dependency parsing
parser = CoreNLPDependencyParser()
parse = next(
    parser.raw_parse(
        "Work in psychology, anthropology, political science, and marketing suggests that
humans are not gullible, but vigilant towards implausible information and tend to hold on to
their prior beliefs."
    )
)
dot = parse.to_dot()
print(dot)

# Stop the server
server.stop()
```

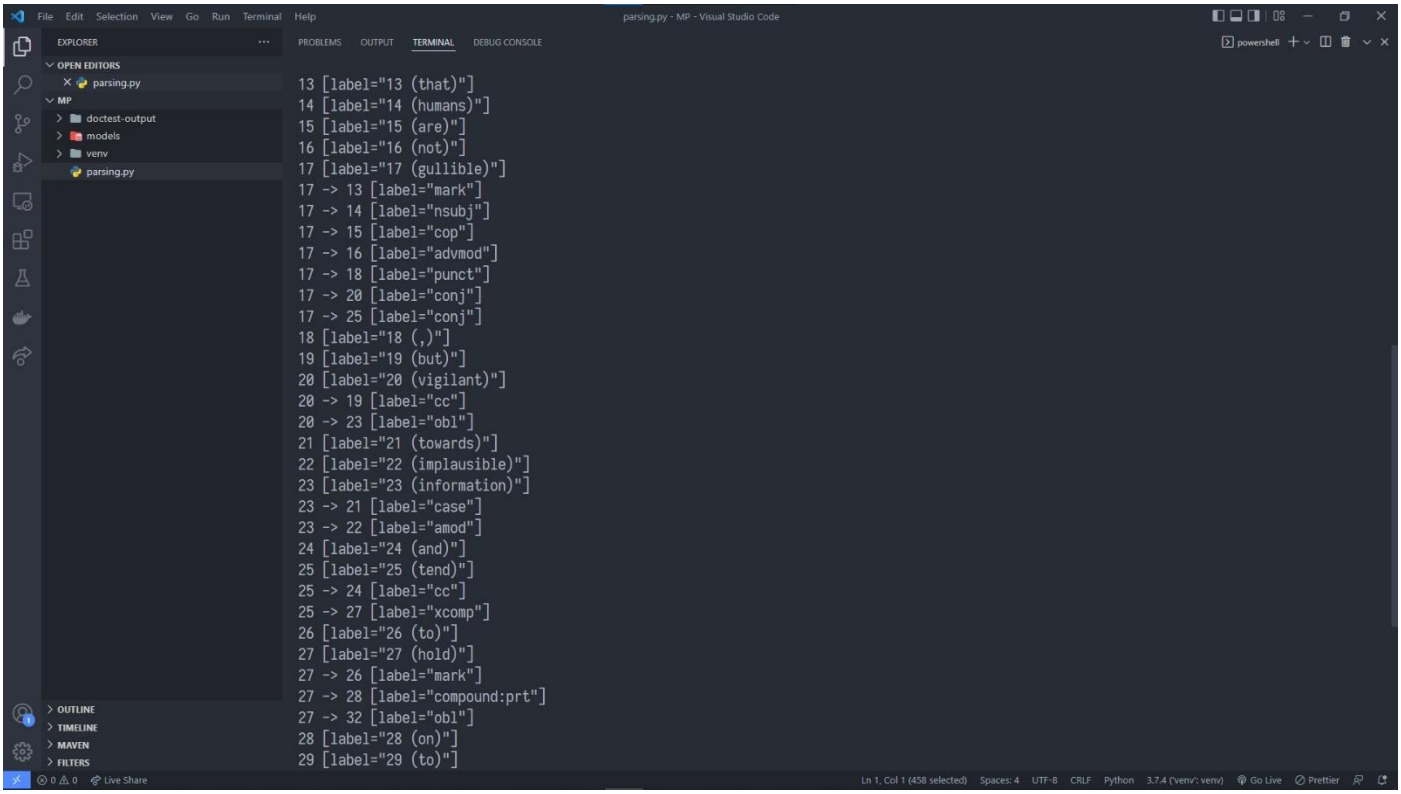
Results

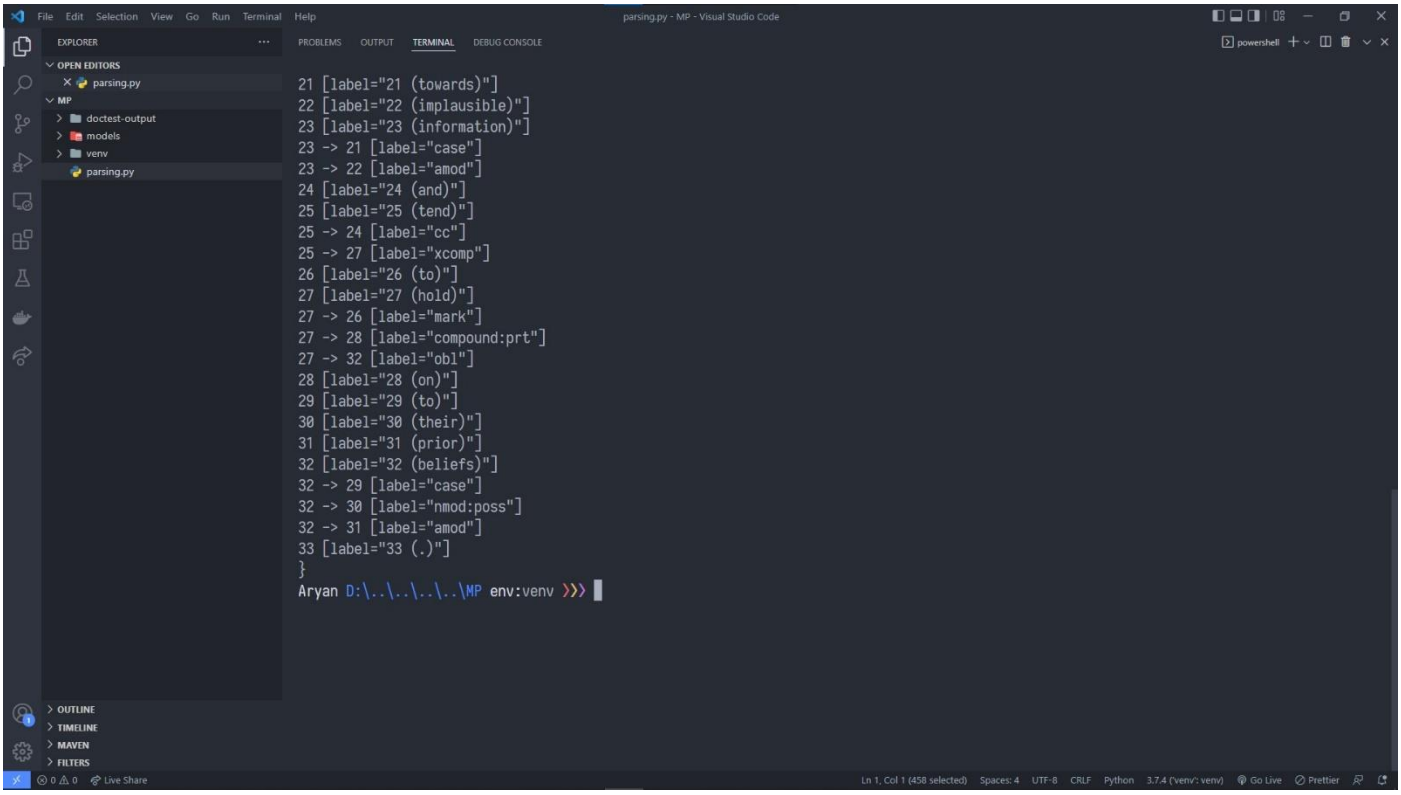
Constituency Parsing

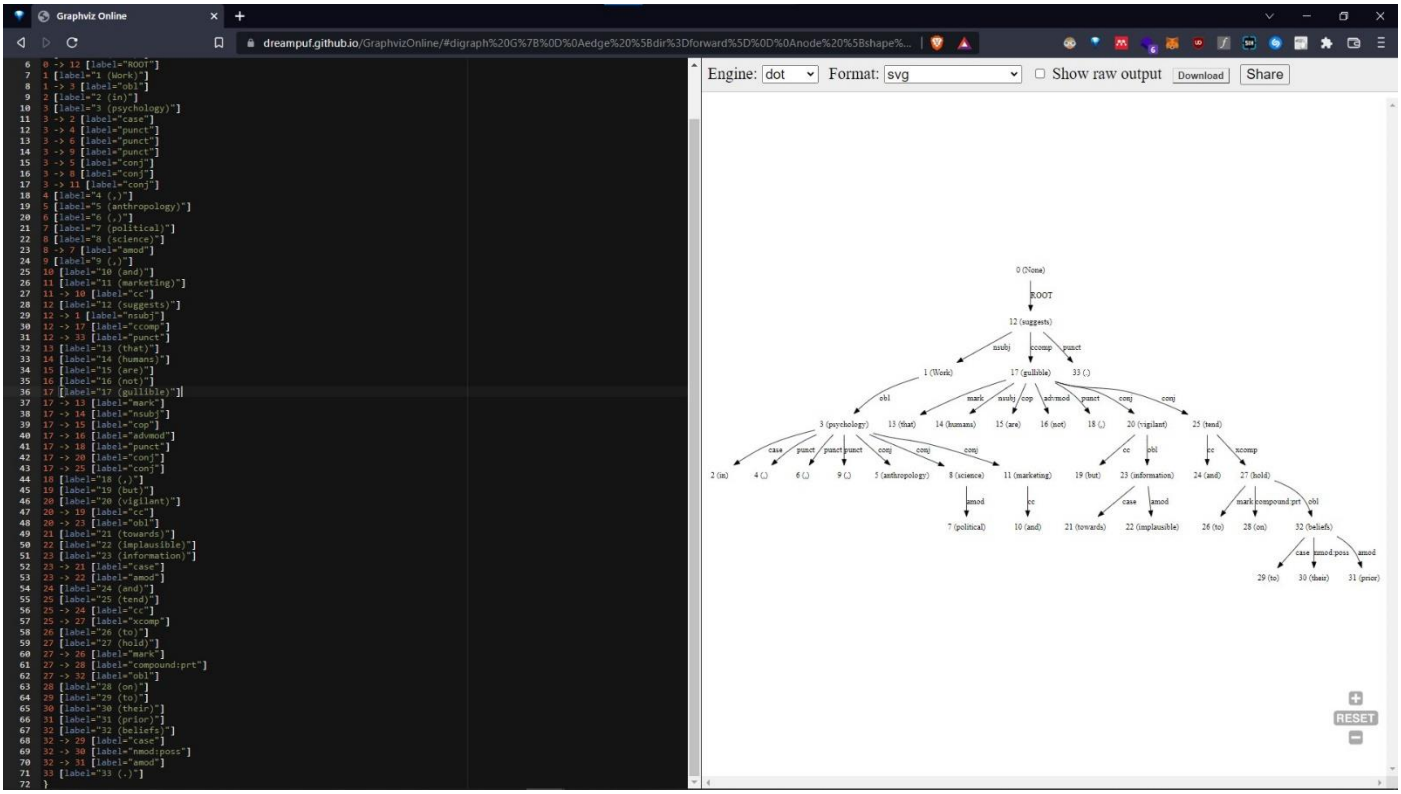


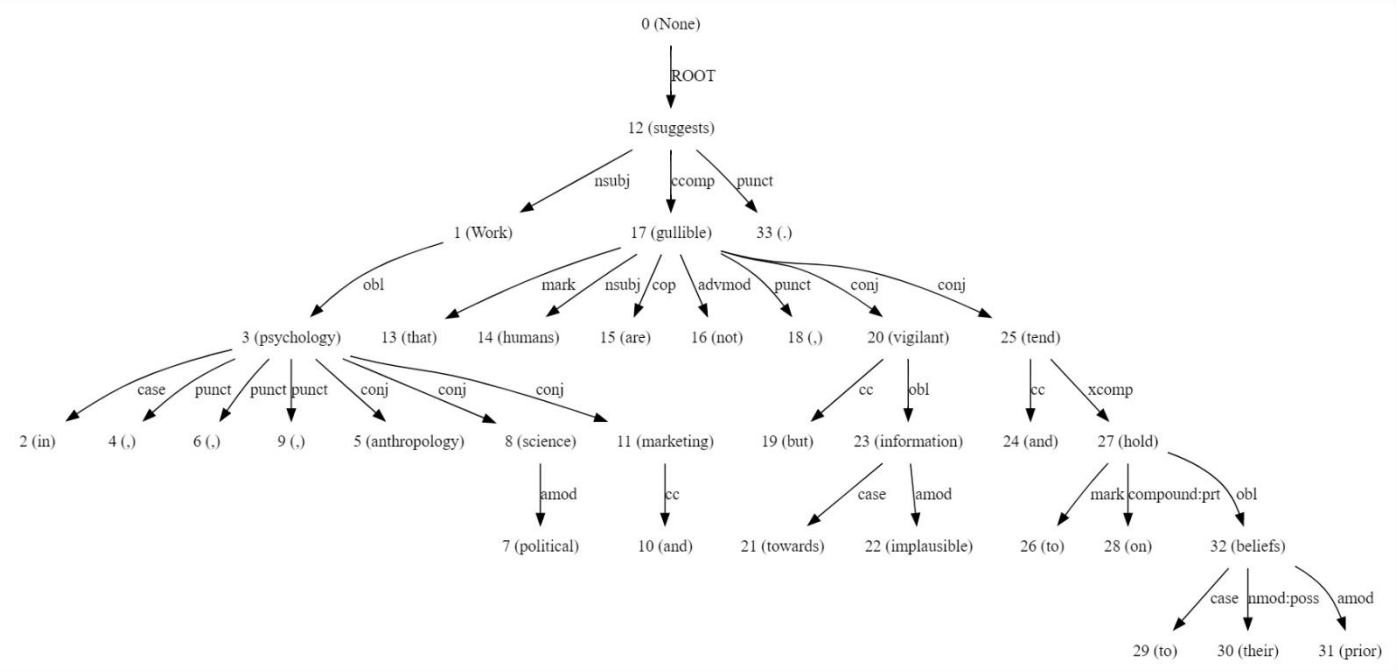
Dependency Parsing






File Edit Selection View Go Run Terminal Help
parsing.py - MP - Visual Studio Code
EXPLORER
OPEN EDITORS
parsing.py
MP
doctest-output
models
venv
parsing.py
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
powershell
21 [label="21 (towards)"]
22 [label="22 (implausible)"]
23 [label="23 (information)"]
23 -> 21 [label="case"]
23 -> 22 [label="amod"]
24 [label="24 (and)"]
25 [label="25 (tend)"]
25 -> 24 [label="cc"]
25 -> 27 [label="xcomp"]
26 [label="26 (to)"]
27 [label="27 (hold)"]
27 -> 26 [label="mark"]
27 -> 28 [label="compound:prt"]
27 -> 32 [label="obl"]
28 [label="28 (on)"]
29 [label="29 (to)"]
30 [label="30 (their)"]
31 [label="31 (prior)"]
32 [label="32 (beliefs)"]
32 -> 29 [label="case"]
32 -> 30 [label="nmod:poss"]
32 -> 31 [label="amod"]
33 [label="33 (.)"]
}
Aryan D:\..\..\..\..\MP env:venv
OUTLINE
TIMELINE
MAVEN
FILTERS
Live Share
Ln 1, Col 1 (458 selected) Spaces: 4 UTF-8 CRLF Python 3.7.4 ('venv': venv) Go Live Prettier


Graphviz Online
dreampuf.github.io/GraphvizOnline/#digraph%20G%7B%0D%0Aedge%20%5Bdir%3Dforward%5D%0D%0Anode%20%5Bshape%...
0 -> 12 [label="ROOT"]
1 [label="1 (Work)"]
1 -> 3 [label="obl"]
2 [label="2 (in)"]
3 [label="3 (psychology)"]
3 -> 2 [label="case"]
3 -> 4 [label="punct"]
3 -> 6 [label="punct"]
3 -> 9 [label="punct"]
3 -> 5 [label="conj"]
3 -> 8 [label="conj"]
3 -> 11 [label="conj"]
4 [label="4 (,)"]
5 [label="5 (anthropology)"]
6 [label="6 (,)"]
7 [label="7 (political)"]
8 [label="8 (science)"]
8 -> 7 [label="amod"]
9 [label="9 (,)"]
10 [label="10 (and)"]
11 [label="11 (marketing)"]
11 -> 10 [label="cc"]
12 [label="12 (suggests)"]
12 -> 1 [label="nsubj"]
12 -> 17 [label="ccomp"]
12 -> 33 [label="punct"]
13 [label="13 (that)"]
14 [label="14 (humans)"]
15 [label="15 (are)"]
16 [label="16 (not)"]
17 [label="17 (gullible)"]
17 -> 13 [label="mark"]
17 -> 14 [label="nsubj"]
17 -> 15 [label="cop"]
17 -> 16 [label="advmod"]
17 -> 18 [label="punct"]
17 -> 20 [label="conj"]
17 -> 25 [label="conj"]
18 [label="18 (,)"]
19 [label="19 (but)"]
20 [label="20 (vigilant)"]
20 -> 19 [label="cc"]
20 -> 23 [label="obl"]
21 [label="21 (towards)"]
22 [label="22 (implausible)"]
23 [label="23 (information)"]
23 -> 21 [label="case"]
23 -> 22 [label="amod"]
24 [label="24 (and)"]
25 [label="25 (tend)"]
25 -> 24 [label="cc"]
25 -> 27 [label="xcomp"]
26 [label="26 (to)"]
27 [label="27 (hold)"]
27 -> 26 [label="mark"]
27 -> 28 [label="compound:prt"]
27 -> 32 [label="obl"]
28 [label="28 (on)"]
29 [label="29 (to)"]
30 [label="30 (their)"]
31 [label="31 (prior)"]
32 [label="32 (beliefs)"]
32 -> 29 [label="case"]
32 -> 30 [label="nmod:poss"]
32 -> 31 [label="amod"]
33 [label="33 (.)"]
}
Engine: dot
Format: svg
Show raw output
Download
Share
0 (None)
ROOT
12 (suggests)
nsubj
ccomp
punct
1 (Work)
17 (gullible)
33 (.)
obl
mark
nsubj
cop
advmod
punct
conj
conj
3 (psychology)
13 (that)
14 (humans)
15 (are)
16 (not)
18 (,)
20 (vigilant)
25 (tend)
case
punct
punct
punct
conj
conj
conj
cc
obl
cc
xcomp
2 (in)
4 (,)
6 (,)
9 (,)
5 (anthropology)
8 (science)
11 (marketing)
19 (but)
23 (information)
24 (and)
27 (hold)
amod
cc
case
amod
mark
compound:prt
obl
7 (political)
10 (and)
21 (towards)
22 (implausible)
26 (to)
28 (on)
32 (beliefs)
case
nmod:poss
amod
29 (to)
30 (their)
31 (prior)
RESET



Conclusion

Using Stanford's CoreNLP suite and NLTK, we were successfully able to carry out both, constituency and dependency parsing of sentences. Visualizations based on the results were also created to adequately represent the parsing trees created.

References

1. Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55-60.
2. Bird, Steven, Edward Loper and Ewan Klein. 2009. *Natural Language Processing with Python*. O'Reilly Media Inc.
3. Jurafsky, Dan, and James H. Martin. 2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Upper Saddle River, N.J: Prentice Hall. Print.