National University
Of Computer and Emerging Sciences

Name:

Dawood Sarfraz

Roll no:

20P-0153

Section:

BSCS-7A

Course:

Natural Language Processing

Lab:

01

Submitted to:

Dr. Omer Usman Khan

FAST National University of Computer and Emerging Sciences

Using the above information, fill the following table:

Corpus	Text1	Text2	Text3	Text4	Text5	Text6	Text7	Text8	Text9
Corpus Name	Moby Dick by Herman Melville 1851	Sense and Sensibility by Jane Austen 1811	The Book of Genesis	Inaugural Address Corpus	Chat Corpus	Monty Python and the Holy Grail	Wall Street Journal	Personals Corpus	The Man Who Was Thursday by G . K . Chesterton 1908
Corpus Length	260819	141576	44764	152901	45010	16967	100676	4867	69213
Unique Words	19317	6833	2789	10025	6066	2166	12408	1108	6807
Lexical Richness	0.074062855 85022564	0.048263 83002768 831	0.06230 453042 623537	0.065565 3004231 4962	0.1347 700510 997556 2	0.1276 59574 46808 51	0.1232 468512 853112 9	0.227655 64002465 585	0.09834857613 45412

```
========== Names of Texts Used ==========

Name of Text 1 is ----->>  <Text: Moby Dick by Herman Melville 1851>
Name of Text 2 is ----->>  <Text: Sense and Sensibility by Jane Austen 1811>
Name of Text 3 is ----->>  <Text: The Book of Genesis>
Name of Text 4 is ----->>  <Text: Inaugural Address Corpus>
Name of Text 5 is ----->>  <Text: Chat Corpus>
Name of Text 6 is ----->>  <Text: Monty Python and the Holy Grail>
Name of Text 7 is ----->>  <Text: Wall Street Journal>
Name of Text 8 is ----->>  <Text: Personals Corpus>
Name of Text 9 is ----->>  <Text: The Man Who Was Thursday by G . K . Chesterton 1908>

 ========== Length of Texts Used ==========

Length of Text 1 is ----->>  260819
Length of Text 2 is ----->>  141576
Length of Text 3 is ----->>  44764
Length of Text 4 is ----->>  152901
Length of Text 5 is ----->>  45010
Length of Text 6 is ----->>  16967
Length of Text 7 is ----->>  100676
Length of Text 8 is ----->>  4867
Length of Text 9 is ----->>  69213
```

```
 ========== Unique Words of Texts Used ==========

Length of Unique words in Text 1 ----->>  19317
Length of Unique words in Text 2 ----->>  6833
Length of Unique words in Text 3 ----->>  2789
Length of Unique words in Text 4 ----->>  10025
Length of Unique words in Text 5 ----->>  6066
Length of Unique words in Text 6 ----->>  2166
Length of Unique words in Text 7 ----->>  12408
Length of Unique words in Text 8 ----->>  1108
Length of Unique words in Text 9 ----->>  6807

 ========== Lexical Richness of Texts Used ==========

Lexical Richness of Text 1 ----->> 0.07406285585022564
Lexical Richness of Text 2 ----->> 0.04826383002768831
Lexical Richness of Text 3 ----->> 0.06230453042623537
Lexical Richness of Text 4 ----->> 0.06556530042314962
Lexical Richness of Text 5 ----->> 0.13477005109975562
Lexical Richness of Text 6 ----->> 0.1276595744680851
Lexical Richness of Text 7 ----->> 0.12324685128531129
Lexical Richness of Text 8 ----->> 0.22765564002465585
Lexical Richness of Text 9 ----->> 0.0983485761345412
```

Now, fill the table below:

	Text 1

Tokkens	**TF( )**	**LOGTF( )**	**IDF( )**
monster	0.018786974875296663	1.6989700043360185	-0.7359535705891886
evil	0.004217484155678842	1.0791812460476247	0.08715017571890013
devil	0.01955379017632918	1.716003343634799	0.7533276666586114
the	5.260736372733581	4.137417414990392	-3.1831432548946452
Common word **,**	7.174707364110744	4.272166625140787	-3.3179009081517252
Common word **the**	5.260736372733581	4.137417414990392	-3.1831432548946452
Common word **.**	2.630943297842565	3.836513998890671	-2.8822082042808295

```
Start for word ----->>  monster

Term Frequency of "monster" ----->>  0.018786974875296663
Term Frequency Log "monster" ----->>  1.6989700043360185
Inverse Document Frequency "monster" ----->>  -0.7359535705891886
==================================================

Start for word ----->>  evil

Term Frequency of "evil" ----->>  0.004217484155678842
Term Frequency Log "evil" ----->>  1.0791812460476247
Inverse Document Frequency "evil" ----->>  -0.08715017571890013
==================================================

Start for word ----->>  devil

Term Frequency of "devil" ----->>  0.01955379017632918
Term Frequency Log "devil" ----->>  1.716003343634799
Inverse Document Frequency "devil" ----->>  -0.7533276666586114
==================================================

Start for word ----->>  the

Term Frequency of "the" ----->>  5.260736372733581
Term Frequency Log "the" ----->>  4.137417414990392
Inverse Document Frequency "the" ----->>  -3.1831432548946452
==================================================
==================================================
```

```
Start for Most Common word ----->> ,

Term Frequency of "," ----->>  7.174707364110744
Term Frequency Log of"," ----->>  4.272166625140787
Inverse Document Frequency of "," ----->>  -3.3179009081517252
==================================================

Start for Most Common word ----->> the

Term Frequency of "the" ----->>  5.260736372733581
Term Frequency Log of"the" ----->>  4.137417414990392
Inverse Document Frequency of "the" ----->>  -3.1831432548946452
==================================================

Start for Most Common word ----->> .

Term Frequency of "." ----->>  2.630943297842565
Term Frequency Log of"." ----->>  3.836513998890671
Inverse Document Frequency of "." ----->>  -2.8822082042808295
==================================================
```

National University
Of Computer and Emerging Sciences

Name:

Dawood Sarfraz

Roll no:

20P-0153

Section:

BSCS-7A

Course:

Natural Language Processing

Lab:

02

Submitted to:

Dr. Omer Usman Khan

FAST National University of Computer and Emerging Sciences

Table - 1:

	Text1	**Text2**	**Tex3**
10 frequently occuring Bigrams	Sperm Whale; Moby Dick; White Whale; old man; Captain Ahab; sperm whale; Right Whale; Captain Peleg; New Bedford; Cape Horn;	Colonel Brandon; Sir John; Lady Middleton; Miss Dashwood; every thing; thousand pounds; dare say; Miss Steeles; said Elinor; Miss Steele;	said unto; pray thee; thou shalt; thou hast; thy seed; years old; spake unto; thou art; LORD God; every living;
5 frequently occuring Trigrams	[('AFTER', 'EXCHANGING', 'HAILS'), ('Anacharsis', 'Clootz', 'deputation'), ('CAULKING', 'ITS', 'SEAMS'), ('ELIZABETH', 'OAKES', 'SMITH'), ('Et', 'tu', 'Brute')]	[('Austen', '1811', ']'), ('Jane', 'Austen', '1811'), ('200', 'L', 'per'), ('Drury', 'Lane', 'lobby'), ('L', 'per', 'annum')]	[('olive', 'leaf', 'pluckt'), ('sewed', 'fig', 'leaves'), ('yield', 'royal', 'dainties'), ('Fifteen', 'cubits', 'upward'), ('leaf', 'pluckt', 'o')]
Number of words with length > 16	['cannibalistically', 'characteristically', 'circumnavigations', 'comprehensiveness', 'indispensableness', 'preternaturalness', 'subterraneousness', 'superstitiousness', 'uncomfortableness', 'uncompromisedness', 'uninterpenetratingly']	['companionableness', 'disinterestedness', 'disqualifications']	[ ] No word found
Number of words with Frequency > 500	['Ahab', 'But', 'I', 'The', 'a', 'all', 'an', 'and', 'are', 'as', 'at', 'be', 'but', 'by', 'for', 'from', 'had', 'have', 'he', 'him', 'his', 'in', 'into', 'is', 'it', 'like', 'man', 'me', 'more', 'my', 'not', 'now', 'of', 'on', 'one', 'or', 'out', 's', 'ship', 'so', 'some', 'that', 'the', 'their', 'then', 'there', 'they', 'this', 'to', 'up', 'upon', 'was', 'were', 'whale', 'when', 'which', 'with', 'you']	['I', 'The', 'a', 'all', 'an', 'and', 'are', 'as', 'at', 'be', 'but', 'by', 'for', 'from', 'had', 'have', 'he', 'him', 'his', 'in', 'into', 'is', 'it', 'like', 'man', 'me', 'more', 'my', 'not', 'now', 'of', 'on', 'one', 'or', 'out', 's', 'so', 'some', 'that', 'the', 'their', 'then', 'there', 'they', 'this', 'to', 'up', 'upon', 'was', 'were', 'when', 'which', 'with', 'you']	['I', 'The', 'a', 'all', 'an', 'and', 'are', 'as', 'at', 'be', 'but', 'by', 'for', 'from', 'had', 'have', 'he', 'him', 'his', 'in', 'into', 'is', 'it', 'like', 'man', 'me', 'more', 'my', 'not', 'now', 'of', 'on', 'one', 'or', 'out', 's', 'so', 'some', 'that', 'the', 'their', 'then', 'there', 'they', 'this', 'to', 'up', 'upon', 'was', 'were', 'when', 'which', 'with', 'you']

Number of words ending in “ed”	2196	902	281

Table - 2:

Number of Words In Corpus	**30 word generated sentence**
1628	computer vision , and generate human language . The future trajectory of NLP . The future trajectory of NLP . The future trajectory of NLP . The future trajectory of
1628	NLP . The future trajectory of NLP . The future trajectory of NLP . The future trajectory of NLP . The future trajectory of NLP . The future trajectory of

```
In [93]: generate_model(cfd, 'computer', 30)
         computer vision , and generate human language . The future trajectory of NLP . The future trajectory of NLP . The
         future trajectory of NLP . The future trajectory of

In [94]: len(wordlist)
Out[94]: 1628
```

```
In [91]: generate_model(cfd, 'NLP', 30)
         NLP . The future trajectory of NLP . The future trajectory of NLP . The future trajectory of NLP . The future traj
         ectory of NLP . The future trajectory of

In [92]: len(wordlist)
Out[92]: 1628
```

National University
Of Computer and Emerging Sciences

Name:

Dawood Sarfraz

Roll no:

20P-0153

Section:

BSCS-7A

Course:

Natural Language Processing

Lab:

04

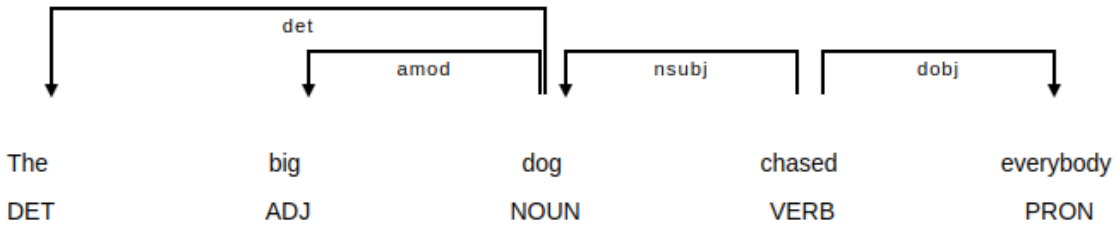
Submitted to:

Dr. Omer Usman Khan

FAST National University of Computer and Emerging Sciences

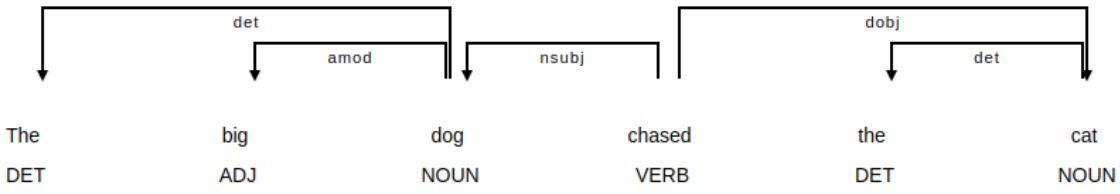
1.

What text and dependencies did the above code catch for the sentence “The big dog chased everybody”



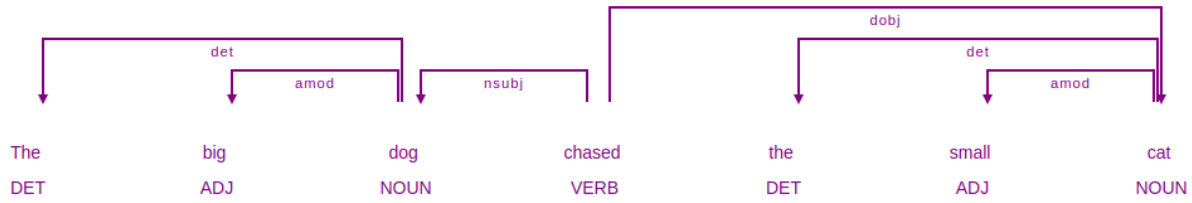

```
Matching Sentence: dog chased everybody
Pattern Type: SubRootObject
Dependency: dog--nsubj
Dependency: chased--ROOT
Dependency: everybody--dobj
```

2. Change the sentence to “The big dog chased the cat”. Does the pattern catch the SVO pattern? If not, add another pattern2 to the matcher. The pattern should be DEP: nsubj, DEP: ROOT, DEP: det, DEP:dobj. When done, update matcher.add("SubRootDetObject", [pattern2])




```
Matching Sentence: dog chased the cat
Pattern Type: SubRootDetObject
Dependency: dog--nsubj
Dependency: chased--ROOT
Dependency: the--det
Dependency: cat--dobj
```

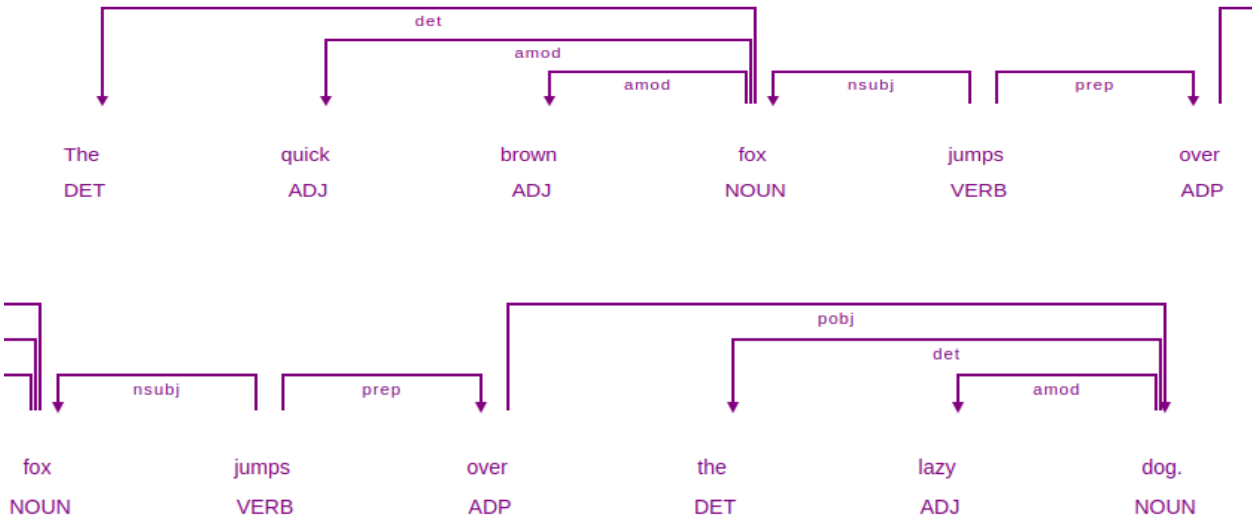
3. Now design a third pattern for the sentence “The big dog chased the small cat”.




```
Matching Sentence: dog chased the small cat
Pattern Type: SubRootAdjDetObject
Dependency: dog--nsubj
Dependency: chased--ROOT
Dependency: the--det
Dependency: small--amod
Dependency: cat--dobj
```

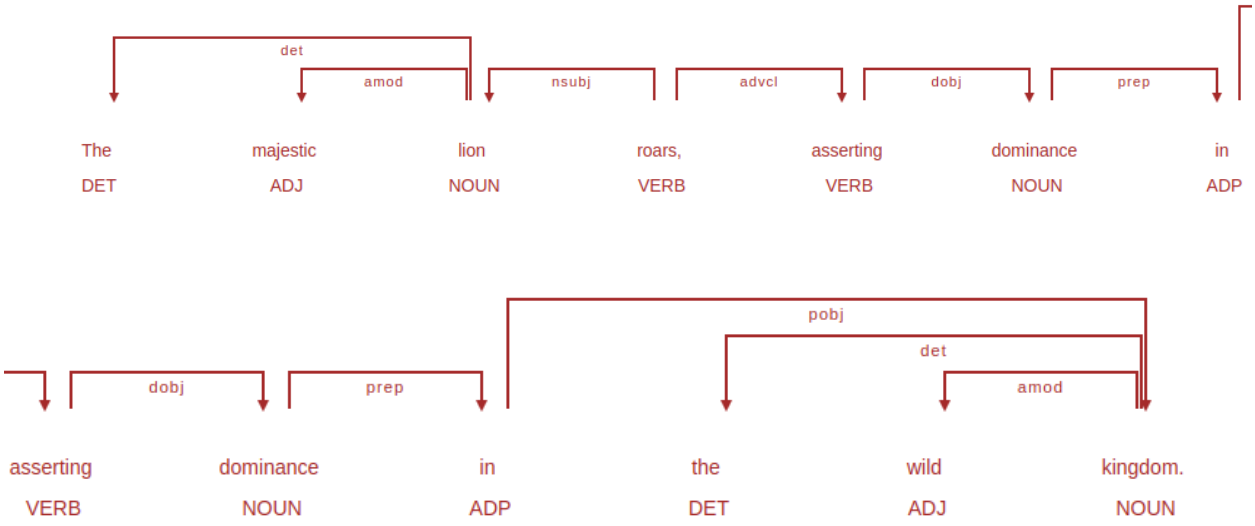
1. Design a pattern to identify a noun at least one time.

In [9]:
```python
pattern = [{"POS": "NOUN", "OP": "*"}]
matcher.add("OneOrMoreNouns", [pattern])

doc = nlp("The quick brown fox jumps over the lazy dog.")

matches = matcher(doc)

displacy.render(doc, style='dep',options ={ "color" : "purple","compact":True} )

# Iterate over matches
for pattern_id, start, end in matches:
    print("Matching Sentence:", doc[start:end].text)
    print("Pattern Type:", doc.vocab.strings[pattern_id])

    # Print information for each token in the match
    for token in doc[start:end]:
        print(f"Dependency: {token.text}-{token.dep_}")
        print("-"*20)
```



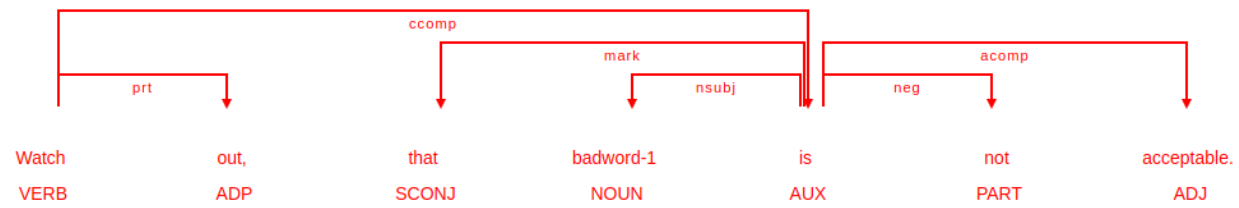
2. Design a pattern to identify a noun of length >= 10 characters.

```
pattern = [{"POS": "NOUN", "LENGTH": {">=": 10}}]
matcher.add("LongNoun", [pattern])

doc = nlp("The majestic lion roars, asserting dominance in the wild kingdom.")

matches = matcher(doc)

displacy.render(doc, style='dep', options ={ "color" : "brown","compact":True})
```



2. Design a pattern to identify vulgar language (Hint: you will need usage of IN, or NOT_IN).

```
bad_word = ["badword-1", "badword-2", "badword-3"]

pattern = [{"LOWER": {"IN": bad_word}}]
matcher.add("VulgarLanguage", [pattern])

doc = nlp("Watch out, that badword-1 is not acceptable.")

matches = matcher(doc)

displacy.render(doc, style='dep', options ={ "color" : "red","compact":True})

for pattern_id, start, end in matches:
    print("Matching Sentence:", doc[start:end].text)
    print("Pattern Type:", doc.vocab.strings[pattern_id])

    for token in doc[start:end]:
        print(f"Token: {token.text} | POS: {token.text.lower()}")
```



Task 6: Replies

1. **Extend the code by adding pattern and matches if a user enters: "I would like to order a pizza". The bot should ask about which pizza type he/she wants.**

In [23]:

```
def utterance(msg):
    nlp = spacy.load('en_core_web_sm')
    doc = nlp(msg)
    matcher = Matcher(nlp.vocab)

    greeting_pattern = [{"LEMMA": {"IN": ["salam", "assalam", "hi", "hello"]}}]
    matcher.add("greeting", [greeting_pattern])

    order_pizza_pattern = [{"LEMMA": {"IN": ["order"]}}, {"LOWER": "a"}, {"LOWER": "pizza"}]
    matcher.add("order_pizza", [order_pizza_pattern])

    matches = matcher(doc)

    if len(matches) == 0:
        print('Please rephrase your request. Be as specific as possible!')
        return None

    for pattern_id, start, end in matches:
        if doc.vocab.strings[pattern_id] == "greeting":
            return "Welcome to Pizza ordering system"
        elif doc.vocab.strings[pattern_id] == "order_pizza":
            return "Sure! What type of pizza would you like to order?"


user_input = "I would like to order a pizza"
bot_response = utterance(user_input)

print("Bot:", bot_response)
```

2. Extend the code by adding pattern and matches if a user enters: "I would like to complain about an order".

In [25]:
```
def utterance(msg):
    nlp = spacy.load('en_core_web_sm')
    doc = nlp(msg)
    matcher = Matcher(nlp.vocab)
    greeting_pattern = [{"LEMMA": {"IN": ["salam", "assalam", "hi", "hello"]}}]
    matcher.add("greeting", [greeting_pattern])

    order_pizza_pattern = [{"LEMMA": {"IN": ["order"]}}, {"LOWER": "a"}, {"LOWER": "pizza"}]
    matcher.add("order_pizza", [order_pizza_pattern])

    complaint_pattern = [{"LEMMA": {"IN": ["complain", "complaint"]}}, {"LOWER": "about"}, {"LOWER": "an"}, {"LOWER'
    matcher.add("complaint_order", [complaint_pattern])

    matches = matcher(doc)

    if len(matches) == 0:
        print('Please rephrase your request. Be as specific as possible!')
        return None

    for pattern_id, start, end in matches:
        if doc.vocab.strings[pattern_id] == "greeting":
            return "Welcome to Pizza ordering system"
        elif doc.vocab.strings[pattern_id] == "order_pizza":
            return "Sure! What type of pizza would you like to order?"
        elif doc.vocab.strings[pattern_id] == "complaint_order":
            return "I'm sorry to hear that. Please provide more details about your complaint."

user_input = "I would like to complain about an order"
bot_response = utterance(user_input)

print("Bot:", bot_response)
```

2. In respone to what pizza type user wants, the user may want to enter "Chief Special Pizza". Use the .lefts (mentioned in Lab 03) to get the pizza type. Ask about quantity. Use Cardinal as ent type to get the quantity, and place the order. Ask for address, and confirm the user with address.

In [28]:
```
def process_order(msg):
    nlp = spacy.load('en_core_web_sm')
    doc = nlp(msg)

    pizza_type = None
    for token in doc:
        if "pizza" in token.text.lower() and token.dep_ == "compound":
            pizza_type = " ".join(t.text for t in token.lefts) + " " + token.text
            break

    quantity = None
    for ent in doc.ents:
        if ent.label_ == "CARDINAL":
            quantity = ent.text
            break

    if pizza_type and quantity:
        address_prompt = "Great choice! How many {} pizzas would you like to order?".format(pizza_type)
        return address_prompt

    if not pizza_type:
        return "I'm sorry, but I couldn't determine the pizza type. Please specify the pizza type."

    if not quantity:
        return "I'm sorry, but I couldn't determine the quantity. Please specify the quantity."

user_input = "I would like to order a Chief Special Pizza"
bot_response = process_order(user_input)

print("Bot:", bot_response)
```

Name:

Dawood Sarfraz

Roll no:

20P-0153

Section:

BSCS-7A

Course:

Natural Language Processing

Lab:

03

Submitted to:

Dr. Omer Usman Khan

FAST National University of Computer and Emerging Sciences

Task 1:

1. What is en_core_web_sm?

In natural language processing (NLP), **`en_core_web_sm`** refers to a specific English language model provided by the **spaCy** library. The **"en"** signifies English, **"core"** suggests that it's a core or base model, **"web"** implies it has been trained on a mixture of web text for versatility, and **"sm"** denotes that it is a small model. This model provides **pre-trained** capabilities for tasks such as **part-of-speech tagging**, **named entity recognition**, and **dependency parsing**. It strikes a balance between model size and performance, making it a practical choice for various NLP applications.

Tokenizer:

- Function: Breaks down a given text into individual words, punctuations, and other meaningful units called tokens.
- Example: "The quick brown fox" would be tokenized into ["The", "quick", "brown", "fox"].

Tagger:

- Function: Assigns parts-of-speech (POS) tags to each token, indicating the grammatical category of the word.
- Example: Tagging "dog" as a noun (NN) and "run" as a verb (VB).

Parser:

- Function: Analyzes the grammatical structure of sentences, determining how words relate to each other syntactically.
- Example: Identifying subject-verb-object relationships in a sentence.

Named Entity Recognition (NER):

- Function: Identifies and classifies named entities (e.g., persons, organizations, locations) within the text.
- Example: Recognizing "Apple" as an organization and "New York" as a location.

Attribute Ruler:

- Function: Applies custom rules to extract additional information or attributes from the text.
- Example: Extracting dates, quantities, or custom patterns based on predefined rules.

Lemmatizer:

- Function: Reduces words to their base or root form (lemma), simplifying variations of a word to a common base.
- Example: Lemmatizing "running" to "run" or "better" to "good."

2. What is the size of en_core_web_sm?

12.8 MB

```
!python3 -m spacy download en_core_web_sm

Collecting en-core-web-sm==3.7.1
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.7.1/en_core_web_sm-3.7.
1-py3-none-any.whl (12.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.8/12.8 MB 2.0 MB/s eta 0:00:00m eta 0:00:01[36m0:00:01
Requirement already satisfied: spacy<3.8.0,>=3.7.2 in /home/chattha/anaconda3/lib/python3.11/site-packages (from e
n-core-web-sm==3.7.1) (3.7.2)
```

3. What other variations can be used?

"en_core_web_md "(Medium):

- Description: This is a medium-sized English model in spaCy.
- Features: It includes more vectors for word representations, making it more suitable for tasks requiring a richer understanding of word meanings.
- Use Cases: It's a good choice when more detailed word embeddings are needed, and the computational resources allow for a larger model.

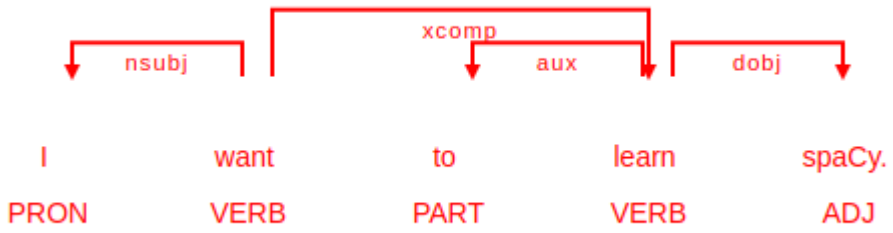
"en_core_web_lg" (Large):

- Description: This is the large English model in spaCy.
- Features: It includes even more vectors for word representations, providing a more extensive and detailed language understanding.
- Use Cases: Suitable for tasks demanding a high level of accuracy and semantic understanding. It's a larger model, so it requires more computational resources.

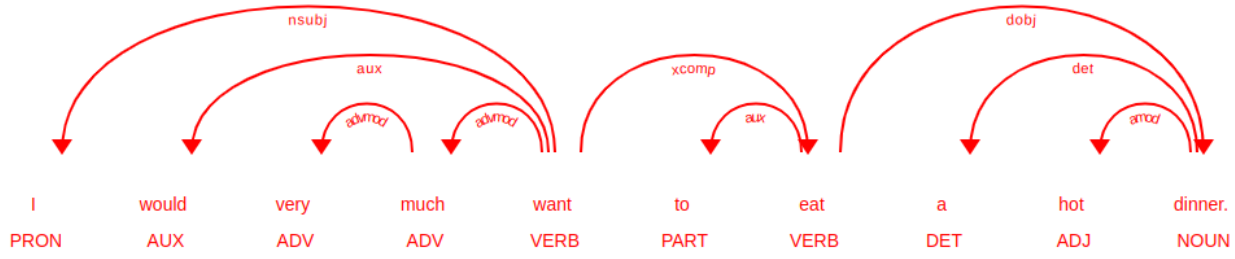
Task 2:

1. ***Draw the left and right dependencies for the sentence: I want to learn spaCy.***

2. ***Draw the children for the sentence: I want to learn spaCy.***



3. ***Draw the left and right dependencies for the sentence: I would very much want to eat a hot dinner.***



4. ***Present a list of all dependency grammars of your sentences above.***

```
Dependency Grammar of Sentence 1
I =====>(nsubj) =====> want
want =====>(ROOT) =====> want
to =====>(aux) =====> learn
learn =====>(xcomp) =====> want
spaCy =====>(dobj) =====> learn
. =====>(punct) =====> want
End of Sentence  1
```

```
Dependency Grammar of Sentence 2
I =====>(nsubj) =====> want
want =====>(ROOT) =====> want
to =====>(aux) =====> learn
learn =====>(xcomp) =====> want
spaCy =====>(dobj) =====> learn
. =====>(punct) =====> want
End of Sentence  2
```

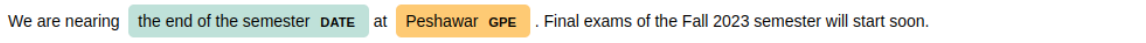
```
Dependency Grammar of Sentence 3
I =====>(nsubj) =====> want
would =====>(aux) =====> want
very =====>(advmod) =====> much
much =====>(advmod) =====> want
want =====>(ROOT) =====> want
to =====>(aux) =====> eat
eat =====>(xcomp) =====> want
a =====>(det) =====> dinner
hot =====>(amod) =====> dinner
dinner =====>(dobj) =====> eat
. =====>(punct) =====> want
End of Sentence  3
```

1. ***How did the Named Entity Output of the NLTK pipeline look like? Present its output.***

```
=== Input Sentense ===

 Final exams of the Fall 2023 semester will start soon.

 === Input Sentense Segmentation ===

 ['We are nearing the end of the semester at Peshawar.', 'Final exams of the Fall 2023 semester will start soon.']

 === Sentense Tokenization ===

 ['We', 'are', 'nearing', 'the', 'end', 'of', 'the', 'semester', 'at', 'Peshawar', '.']

 === Sentense Tokenization ===

 [('We', 'PRP'), ('are', 'VBP'), ('nearing', 'VBG'), ('the', 'DT'), ('end', 'NN'), ('of', 'IN'), ('the', 'DT'),
('semester', 'NN'), ('at', 'IN'), ('Peshawar', 'NNP'), ('.', '.')]



 === Name Entity Reconization ===

 (S
  We/PRP
  are/VBP
  nearing/VBG
  the/DT
  end/NN
  of/IN
  the/DT
  semester/NN
  at/IN
  (ORGANIZATION Peshawar/NNP)
  ./.)

 === Sentense Tokenization ===

 ['Final', 'exams', 'of', 'the', 'Fall', '2023', 'semester', 'will', 'start', 'soon', '.']

 === Sentense Tokenization ===

 [('Final', 'JJ'), ('exams', 'NN'), ('of', 'IN'), ('the', 'DT'), ('Fall', 'NN'), ('2023', 'CD'), ('semester', 'N
N'), ('will', 'MD'), ('start', 'VB'), ('soon', 'RB'), ('.', '.')]



   === Name Entity Reconization ===

   (S
    Final/JJ
    exams/NN
    of/IN
    the/DT
    Fall/NN
    2023/CD
    semester/NN
    will/MD
    start/VB
    soon/RB
    ./.)
```

1. ***How did the Named Entity Output of the spaCy pipeline look like? Present its output.***

```
In [65]: from spacy import displacy
         doc = nlp(u'We are nearing the end of the semester at Peshawar. Final exams of the Fall 2023 semester will start soo
         displacy.render(doc, style='ent')
```




```
In [66]: for ent in doc.ents:
             print(ent.text, ent.label_)
```

```
the end of the semester DATE
Peshawar GPE
```

1. ***What is the default pipeline structure of spaCy?***

```
In [67]: import spacy
         nlp = spacy.load('en_core_web_sm')
         nlp.pipe_names
```

```
Out[67]: ['tok2vec', 'tagger', 'parser', 'attribute_ruler', 'lemmatizer', 'ner']
```