WordNet Summary (Step 1)

WordNet is a database of nouns, verbs, adjectives, and adverbs that are placed relative to other nouns, verbs, adjectives, and adverbs in a hierarchy. In this database, you can get definitions of a word (known as glosses) as well as synonyms to a word (known as synsets).

Noun Synsets (Step 2)

Given a noun, we will collect the synsets of that noun

```
import nltk
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('sentiwordnet')
nltk.download('gutenberg')
nltk.download('genesis')
nltk.download('inaugural')
nltk.download('nps_chat')
nltk.download('webtext')
nltk.download('treebank')
nltk.download('stopwords')

from nltk.corpus import wordnet as wn
from nltk.wsd import lesk
from nltk.corpus import sentiwordnet as swn
from nltk.book import text4
import math
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package sentiwordnet to /root/nltk_data...
[nltk_data]   Package sentiwordnet is already up-to-date!
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Package gutenberg is already up-to-date!
[nltk_data] Downloading package genesis to /root/nltk_data...
[nltk_data]   Package genesis is already up-to-date!
[nltk_data] Downloading package inaugural to /root/nltk_data...
[nltk_data]   Package inaugural is already up-to-date!
[nltk_data] Downloading package nps_chat to /root/nltk_data...
[nltk_data]   Package nps_chat is already up-to-date!
[nltk_data] Downloading package webtext to /root/nltk_data...
[nltk_data]   Package webtext is already up-to-date!
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Package treebank is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
noun_syn = wn.synsets('fire')
noun_syn
```

```
[Synset('fire.n.01'),
 Synset('fire.n.02'),
 Synset('fire.n.03'),
 Synset('fire.n.04'),
 Synset('fire.n.05'),
 Synset('ardor.n.03'),
 Synset('fire.n.07'),
 Synset('fire.n.08'),
 Synset('fire.n.09'),
 Synset('open_fire.v.01'),
 Synset('fire.v.02'),
 Synset('fire.v.03'),
 Synset('displace.v.03'),
 Synset('fire.v.05'),
 Synset('fire.v.06'),
 Synset('arouse.v.01'),
 Synset('burn.v.01'),
 Synset('fuel.v.02')]
```

Synset Analysis (Step 3)

WordNet's hierarchy gets more and more generic with each step up until it reaches entity, the top synset for nouns. The synsets further up the hierarchy are less indicative of the original noun we started with. Rather, the higher hypernyms describe more so what a noun is in general rather than what the original noun was, since every noun is essentially some sort of entity.

```
ardor_syn = noun_syn[5]
print("Synset " + ardor_syn.name() + " definition: ", ardor_syn.definition())
print("Examples: ", ardor_syn.examples())

# splitting the printing so it can all be seen on one line
print("Lemmas: ", ardor_syn.lemmas()[:5])
print(ardor_syn.lemmas()[5:])

print("\ngo up the WordNet hierarchy from " + ardor_syn.name())
hyper = lambda s: s.hypernyms()
list(ardor_syn.closure(hyper))
```

```
Synset ardor.n.03 definition:  feelings of great warmth and intensity
Examples:  ['he spoke with great ardor']
Lemmas:  [Lemma('ardor.n.03.ardor'), Lemma('ardor.n.03.ardour'), Lemma('ardor.n.03.fervor'), Lemma('ardor.n.03.fervour'), Lemma('ardor.r
[Lemma('ardor.n.03.fire'), Lemma('ardor.n.03.fervidness')]

go up the WordNet hierarchy from ardor.n.03
[Synset('passion.n.01'),
 Synset('feeling.n.01'),
 Synset('state.n.02'),
 Synset('attribute.n.02'),
 Synset('abstraction.n.06'),
 Synset('entity.n.01')]
```

Synset Relations (Step 4)

Prints out the hypernyms, hyponyms, meronyms, holonyms, and antonym for my chosen synset

```
print(ardor_syn.hypernyms())
print(ardor_syn.hyponyms())
print(ardor_syn.part_meronyms())
print(ardor_syn.part_holonyms())
syn_lemmas = ardor_syn.lemmas()
print(syn_lemmas[0].antonyms())
```

```
[Synset('passion.n.01')]
[Synset('zeal.n.02')]
[]
[]
[]
```

Verb Synsets (Step 5)

Print synsets from a verb

```
verb_synsets = wn.synsets("climbed")
print(verb_synsets)
```

```
[Synset('climb.v.01'), Synset('climb.v.02'), Synset('wax.v.02'), Synset('climb.v.04'), Synset('climb.v.05'), Synset('rise.v.02')]
```

Verb Synset Analysis (Step 6)

Verbs don't seem to all have a common top hypernym in the hierarchy like nouns do. Instead, their top hypernym is dependent on what the original synset is. For instance, travel is the top hypernym of climb, but probably wouldn't be the top hypernym of another word like stop. The top hypernym describes generally what is being done in the original word, such as in the example where travel is generally what is done when you climb.

```
climb_syn = verb_synsets[0]
print("Synset " + climb_syn.name() + " definition: ", climb_syn.definition())
print("Examples: ", climb_syn.examples())
print("Lemmas:", climb_syn.lemmas())
```

```
print("go up the WordNet hierarchy from " + climb_syn.name())
hyper = lambda s: s.hypernyms()
list(climb_syn.closure(hyper))
```

```
Synset climb.v.01 definition:  go upward with gradual or continuous progress
Examples:  ['Did you ever climb up the hill behind your house?']
Lemmas: [Lemma('climb.v.01.climb'), Lemma('climb.v.01.climb_up'), Lemma('climb.v.01.mount'), Lemma('climb.v.01.go_up')]
go up the WordNet hierarchy from climb.v.01
[Synset('rise.v.01'), Synset('travel.v.01')]
```

Using morphy (Step 7)

Only the form "climb" was found, since the morphy correctly identified "climbed" as a verb

```
print(wn.morphy("climbed", wn.VERB))
print(wn.morphy("climbed", wn.NOUN))
print(wn.morphy("climbed", wn.ADJ))
print(wn.morphy("climbed", wn.ADV))
```

```
climb
None
None
None
```

Similar Words (Step 8)

Using the words 'eat' and 'devour', I picked those with synsets that had relatively similar definitions. Using the Wu-Palmer similarity metric, I saw that the words were fairly similar to each other with a metric of 0.667. I believe they are both similar because of both being related to having food, but they differ in the nature of how the food is eaten. For the Lesk algorithm, I intentionally used the same sentence except for the word in question. Therefore, the synsets that they outputted must be similar to some degree since they were used in the example sentence to mean relatively the same thing

```
eat = wn.synsets("eat")
devour = wn.synsets("devour")

# print the synsets
print(eat)
print(devour)

# get the appropriate definition
print("\ndefinition for synset " + eat[1].name() + ": " + eat[1].definition())
print("definition for synset " + devour[3].name() + ": " + devour[3].definition())
eat_syn = eat[1]
devour_syn = devour[3]

# Wu-Palmer similarity metric
print("\nWu-Palmer similarity for eat and devour: ", wn.wup_similarity(eat_syn, devour_syn))

# Lesk algorithm
sentence1 = ['I', 'eat', 'cereal', 'for', 'breakfast', 'every', 'morning', '.']
print("Lesk algorithm for eat: ", lesk(sentence1, 'eat', 'v'))

sentence2 = ['I', 'devour', 'cereal', 'for', 'breakfast', 'every', 'morning', '.']
print("Lesk algorithm for devour: ", lesk(sentence1, 'devour', 'v'))
```

```
[Synset('eat.v.01'), Synset('eat.v.02'), Synset('feed.v.06'), Synset('eat.v.04'), Synset('consume.v.05'), Synset('corrode.v.01')]
[Synset('devour.v.01'), Synset('devour.v.02'), Synset('devour.v.03'), Synset('devour.v.04')]

definition for synset eat.v.02: eat a meal; take a meal
definition for synset devour.v.04: eat greedily

Wu-Palmer similarity for eat and devour:  0.6666666666666666
Lesk algorithm for eat:  Synset('eat.v.02')
Lesk algorithm for devour:  Synset('devour.v.04')
```

SentiWordNet (Step 9)

Describing SentiWordNet

SentiWordNet is a tool built on top of WordNet that receives as input a synset and outputs 3 sentiment analysis scores: positive, negative, and objective. A use case for this could be to analyze a set of tweets and determine based on scores the tone of the speaker who wrote the tweet. In addition, any sort of text could be analyzed to determine the tone of voice for the author, which could be used to help determine the true meaning of the sentence taking into account the mood of the speaker.

Score Observations

For a single word that is clearly emotionally charged like "nasty", SentiWordNet did a pretty good job of determining that it was pretty negative. For the sentence, it was a bit more challenging because I took only the first synset for each word and used its polarity scores. This approach didn't always represent my intended word. For instance, it believed that 'I' meant 'Iodine' and gave it a very objective score. However, it did pick up the most emotionally charged word of the sentence, 'greatest', while the other words were fairly neutral, so overall it got the general idea of the sentence even though it didn't get all the words right. It would be good to know these scores in NLP applications in order to best know the intended meaning of a sentence, as the tone of the speaker (negative, positive, or neutral) adds a lot of meaning to the sentence beyond just the words themselves.

```
senti_list = list(swn.senti_synsets('nasty'))

# print the score for every synset for nasty
for item in senti_list:
    print(item)
    print("Positive score: ", item.pos_score())
    print("Negative score", item.neg_score())
    print("Objective score: ", item.obj_score())

print("\n")

polarity_sent = "That was the greatest meal I ever ate"
tokens = polarity_sent.split()
print("polarity for each word in sentence: ", polarity_sent)

# for each token, get its list of synsets, then take the first synset and calculate its scores
for token in tokens:
  syn_list = list(swn.senti_synsets(token))
  if syn_list:
    syn = syn_list[0]
    print(syn)
    print("Positive score: ", syn.pos_score())
    print("Negative score", syn.neg_score())
    print("Objective score: ", syn.obj_score())
```

```
 <nasty.a.01: PosScore=0.0 NegScore=0.875>
 Positive score:  0.0
 Negative score 0.875
 Objective score:  0.125
 <nasty.s.02: PosScore=0.0 NegScore=0.75>
 Positive score:  0.0
 Negative score 0.75
 Objective score:  0.25
 <cruddy.s.01: PosScore=0.125 NegScore=0.75>
 Positive score:  0.125
 Negative score 0.75
 Objective score:  0.125
 <filthy.s.01: PosScore=0.0 NegScore=0.875>
 Positive score:  0.0
 Negative score 0.875
 Objective score:  0.125


 polarity for each word in sentence:  That was the greatest meal I ever ate
 <washington.n.02: PosScore=0.0 NegScore=0.0>
 Positive score:  0.0
 Negative score 0.0
 Objective score:  1.0
 <greatest.s.01: PosScore=0.875 NegScore=0.0>
 Positive score:  0.875
 Negative score 0.0
 Objective score:  0.125
 <meal.n.01: PosScore=0.0 NegScore=0.0>
 Positive score:  0.0
 Negative score 0.0
 Objective score:  1.0
 <iodine.n.01: PosScore=0.0 NegScore=0.0>
 Positive score:  0.0
 Negative score 0.0
 Objective score:  1.0
```

```
<ever.r.01: PosScore=0.0 NegScore=0.0>
Positive score:  0.0
Negative score 0.0
Objective score:  1.0
<ate.n.01: PosScore=0.0 NegScore=0.0>
Positive score:  0.0
Negative score 0.0
Objective score:  1.0
```

Collocations (Step 10)

What is a Collocation

A collocation is a set of two words that are often said together to mean a specific thing and often its exact meaning cannot be replaced by another word. For instance, the words 'right' and 'now' are often seen together to emphasize that something needs to happen now, and its difficult to replace these words to get exactly the same meaning.

Mutual Information Commentary

the mutual information calculated was about 3.868. The fact that this number is positive shows that it is likely to be a collocation since the two words "Federal" and "Government" show up together often. However, the reason why the number is not higher is because its likely that the words "Federal" and "Government" do not show up all that often relative to the entire text. Its also likely that the word "Government" can show up on its own more often than "Federal" since Federal is an adjective describing something while Government is more general and can be used more often as shown with Government getting a higher probability than Federal.

```
print(text4.collocations())

print("\ncalculating mutual information for collocation Federal Government")

# get all of the text from text4
text = ' '.join(text4.tokens)

# get all unique words from text4
vocab = len(set(text4))

# get probability that the two words show up together
fg = text.count("Federal Government") / vocab
print("p(Federal Government) = ", fg)

# get probability of the first word showing up in the text
f = text.count("Federal") / vocab
print("p(Federal) = ", f)

# get probability of the second word showing up in the text
g = text.count("Government") / vocab
print("p(Government) = ", g)

# find the pmi (point-wise mutual information) of the two words
pmi = math.log2(fg / (f * g))
print('pmi = ', pmi)
```

```
United States; fellow citizens; years ago; four years; Federal
Government; General Government; American people; Vice President; God
bless; Chief Justice; one another; fellow Americans; Old World;
Almighty God; Fellow citizens; Chief Magistrate; every citizen; Indian
tribes; public debt; foreign nations
None

calculating mutual information for collocation Federal Government
p(Federal Government) =  0.0031920199501246885
p(Federal) =  0.006483790523690773
p(Government) =  0.03371571072319202
pmi =  3.868067366919006
```

✓ 0s completed at 1:32 PM