NLP Assignment 1:

Name: Divita Phadakale (diph8836)

(Link to colab: https://colab.research.google.com/drive/1Ql_ZGk_poldT423-7TzV_pNd_egmgLAr?usp=sharing)

1. What do you expect the difference between the Brown bigram and trigram models to look like? Which model will provide you with more coherent text? How will the perplexity of each compare? You should test your predictor and perplexity function using the brown_bigrams and brown_trigrams to confirm your expectations. For perplexity, an average over 2-5 sentences from the Brown corpus should be fine, but make sure you use the same sentences both times. If something you did not expect occurs, explain what happened and why you believe it happened.

Answer:

- I expect to see the brown bigram models' output to have less coherent text as compared to that of brown trigram models' output. This is because of the difference in the training of models. For training a bigram model, the conditional probabilities of preceding unigrams or a single word was considered. And for training a trigram model, the conditional probabilities of preceding bigrams were considered (meaning likelihood of 3 words being together).
- For perplexity, where the measure is the notion of surprise. Higher the perplexity, more the surprise. Therefore, lower perplexity is good.
- When brown corpus perplexity was tested, it was strange to get a higher trigram perplexity. I expected the trigram perplexity to be low as compared to that of bigram perplexity. (When I tried with some random initial sentence, I did get an expected outcome, so I was surprised!)
- I believe it happened because I was testing just a few sentences arbitrarily and not the entirety of the corpus.

2. When testing our bigram models on the Reuters data, do you think a model trained on Brown or Webtext will perform best? Pick any 25 sentences from the Reuters corpus and calculate the average perplexity using each of your bigram datasets. Compare the results of each and provide explanation as to why you believe that one performed better than the other.

Output after calculating perplexity over Reuters corpus:

```
BIGRAMS
For brown corpus -  6919.75819928964
For webtext corpus -  1646.8463495772999
```

- Just as explained in the above questions, a few sentences won't be able to tell which would be better amongst Brown and Webtext. Looking at the similarities in datasets would help.
- According to me, if there are similarities in the dataset (like grammar, tone of speaking, etc) then that would give us good results.
- Looking at the results, webtext corpus performed better than brown. I believe this happened due to similarities between Reuters selected sentences and Webtext corpus.

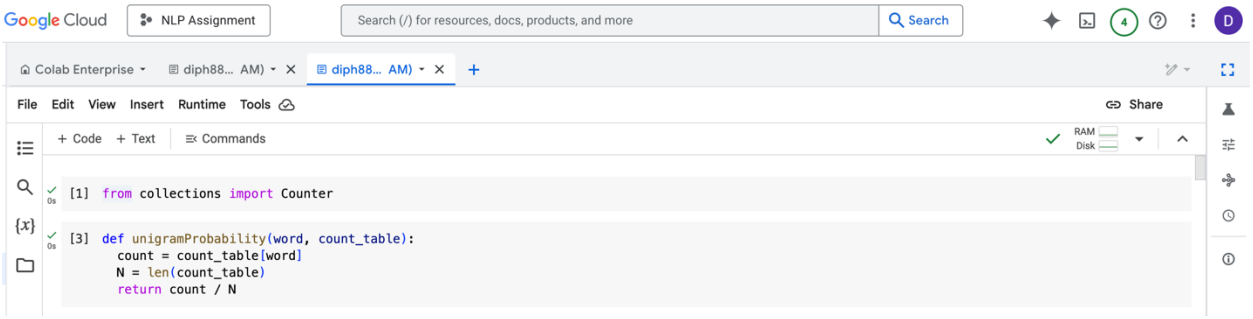
3. When predicting the next word in a sentence, what do you believe would happen if we increased the number of sentences in our training data?

- Increasing number of sentences may bring out 2 different outcomes based on the nature of base corpus.
- If the model is underfitting, then addition of sentences would be good for our training data.
- If the model is overfitting, then addition of sentences will result in training data to be confused and hallucinate.

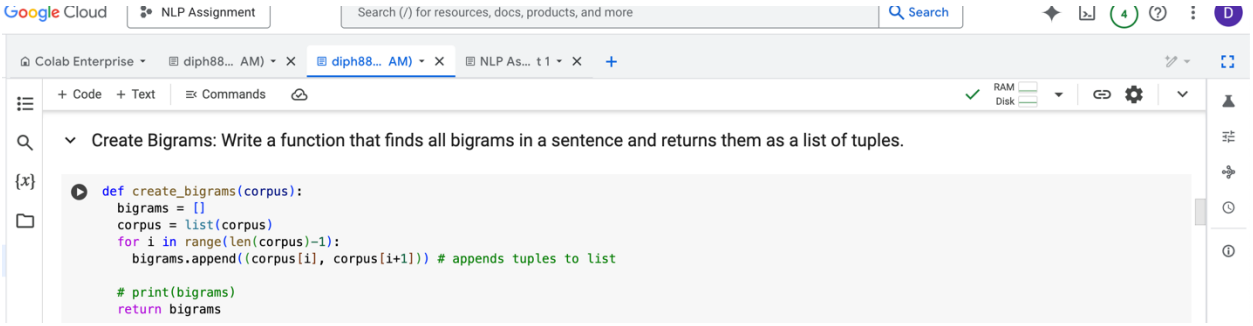
Some implementation screenshots:

1. Write a function that calculates the unigram probability of a word appearing in some corpus.

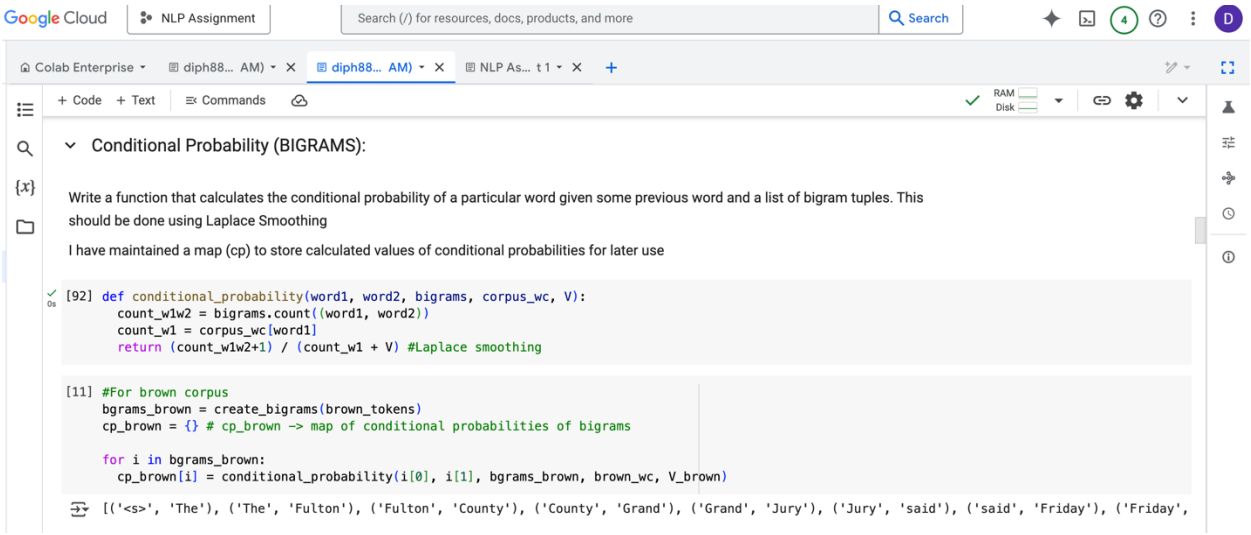
```
def unigramProbability(word, count_table):
  count = count_table[word]
  N = len(count_table)
  return count / N
```



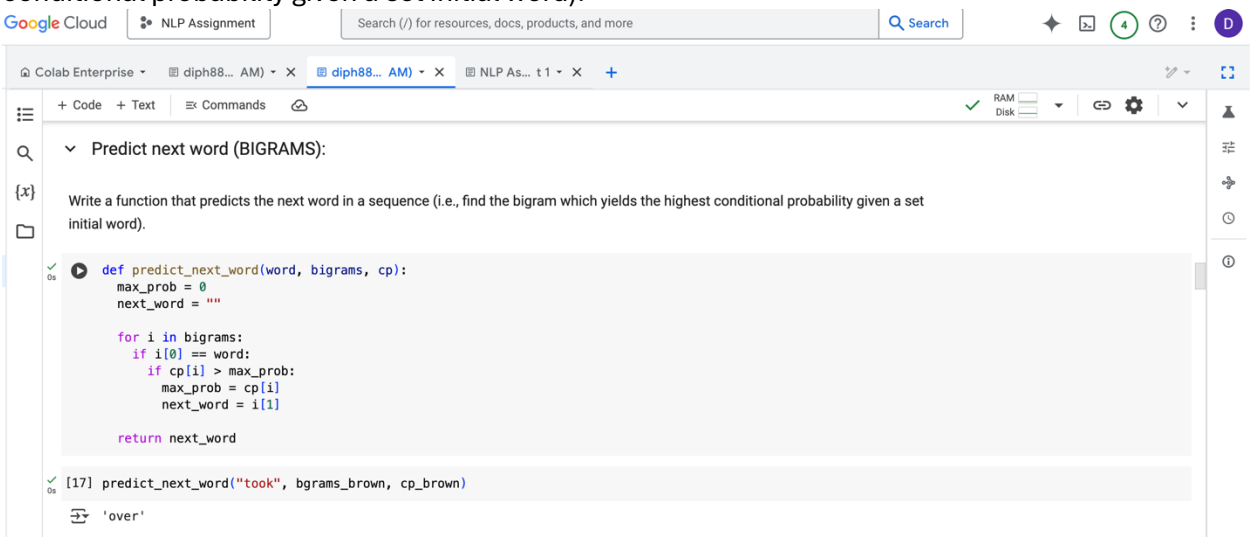
2. Create the bigram model.
i. Write a function that finds all bigrams in a sentence and returns them as a list of tuples.



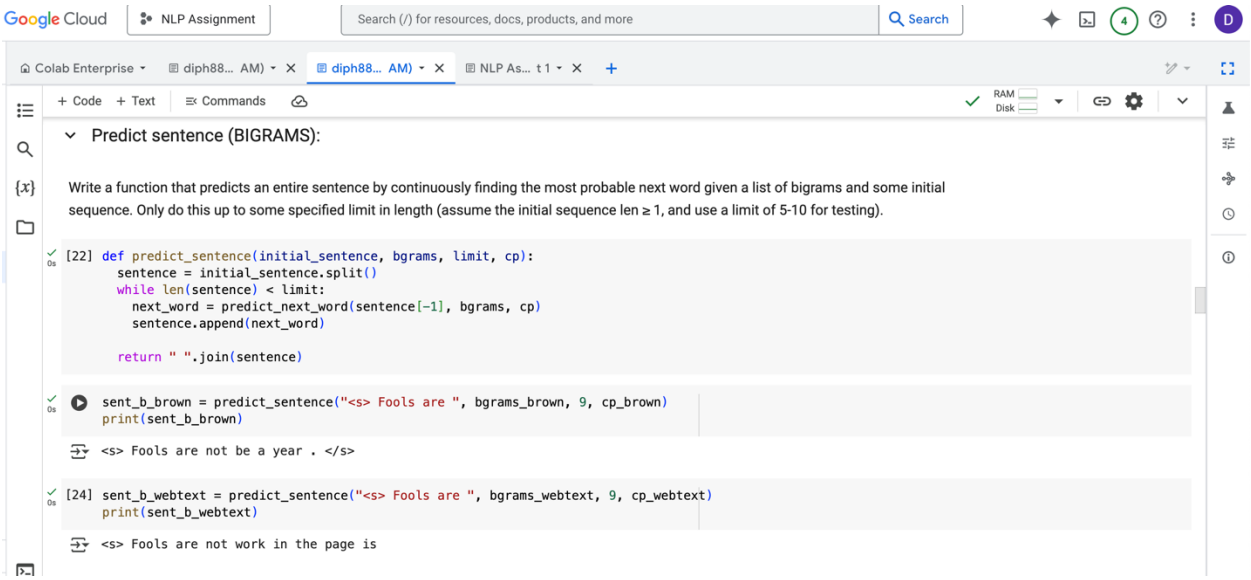
ii. Write a function that calculates the conditional probability of a particular word given some previous word and a list of bigram tuples. This should be done using Laplace Smoothing, see https://web. stanford.edu/~jurafsky/slp3/3.pdf, pg 15, Eq. (3.27).



iii. Write a function that predicts the next word in a sequence (i.e., find the bigram which yields the highest
conditional probability given a set initial word).



iv.
Writeafunctionthatpredictsanentiresentencebycontinuouslyfindingthemostprobablenextwordgiven a list of bigrams and some initial sequence. Only do this up to some specified limit in length (assume the initial sequence len ≥ 1, and use a limit of 5-10 for testing).



Predict sentence (BIGRAMS):

Write a function that predicts an entire sentence by continuously finding the most probable next word given a list of bigrams and some initial sequence. Only do this up to some specified limit in length (assume the initial sequence len ≥ 1, and use a limit of 5-10 for testing).

```
[22] def predict_sentence(initial_sentence, bgrams, limit, cp):
       sentence = initial_sentence.split()
       while len(sentence) < limit:
         next_word = predict_next_word(sentence[-1], bgrams, cp)
         sentence.append(next_word)

       return " ".join(sentence)
```

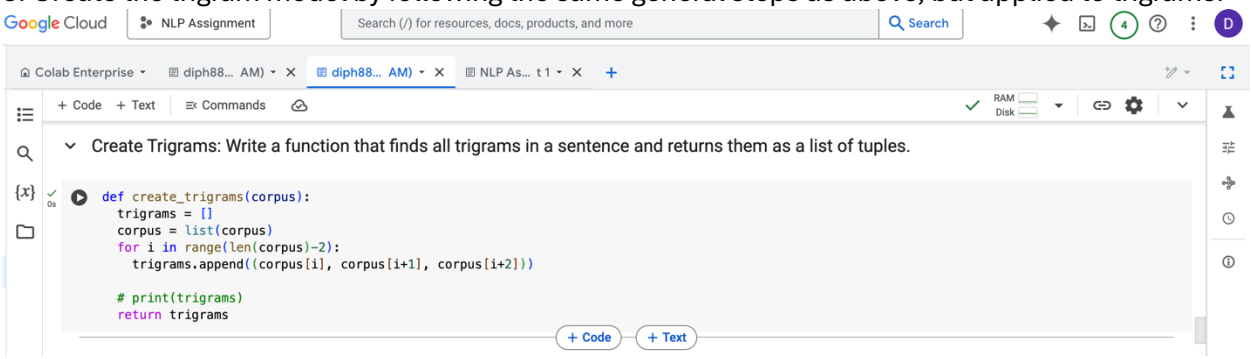
```
sent_b_brown = predict_sentence("<s> Fools are ", bgrams_brown, 9, cp_brown)
print(sent_b_brown)
```

```
<s> Fools are not be a year . </s>
```

```
[24] sent_b_webtext = predict_sentence("<s> Fools are ", bgrams_webtext, 9, cp_webtext)
     print(sent_b_webtext)
```

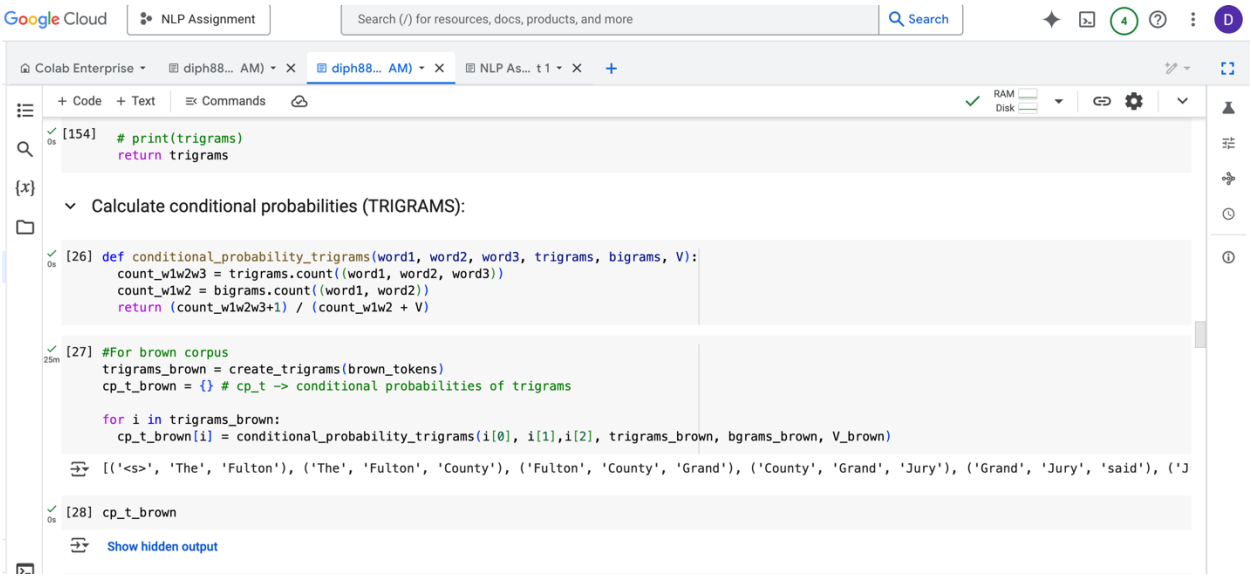
```
<s> Fools are not work in the page is
```

3. Create the trigram model by following the same general steps as above, but applied to trigrams.



Create Trigrams: Write a function that finds all trigrams in a sentence and returns them as a list of tuples.

```
def create_trigrams(corpus):
  trigrams = []
  corpus = list(corpus)
  for i in range(len(corpus)-2):
    trigrams.append((corpus[i], corpus[i+1], corpus[i+2]))

  # print(trigrams)
  return trigrams
```



```
[154]  # print(trigrams)
       return trigrams
```

Calculate conditional probabilities (TRIGRAMS):

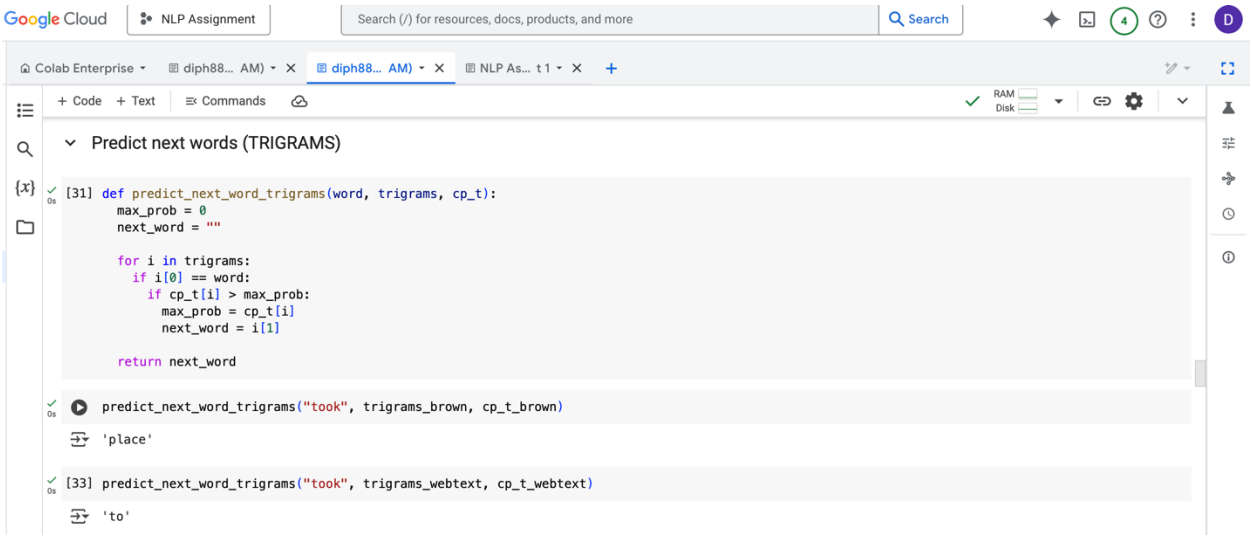
```
[26] def conditional_probability_trigrams(word1, word2, word3, trigrams, bigrams, V):
       count_w1w2w3 = trigrams.count((word1, word2, word3))
       count_w1w2 = bigrams.count((word1, word2))
       return (count_w1w2w3+1) / (count_w1w2 + V)
```

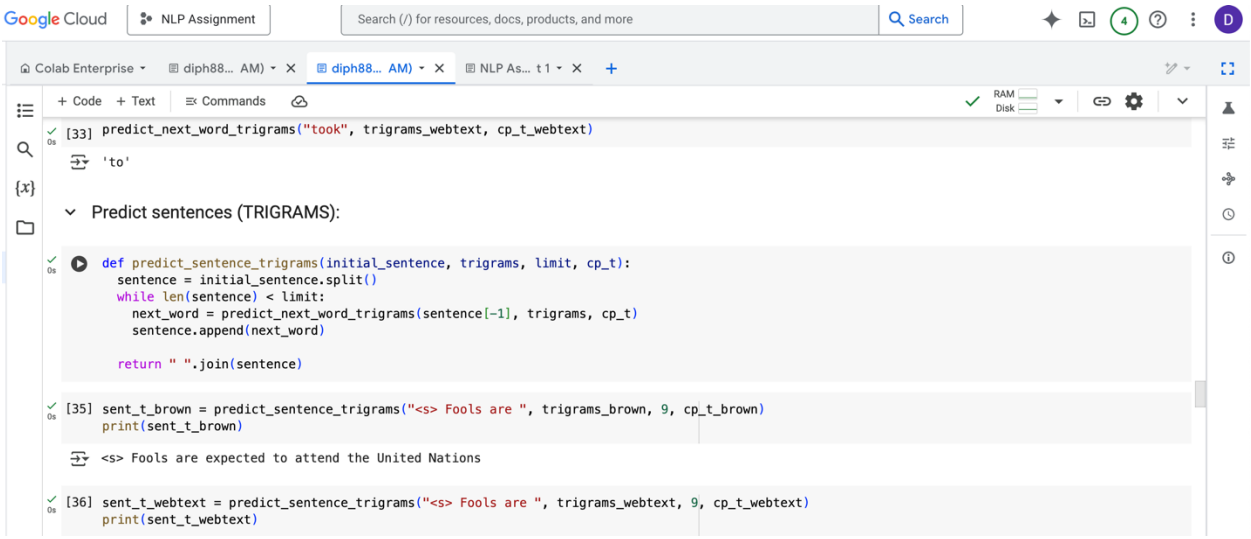
```
[27] #For brown corpus
     trigrams_brown = create_trigrams(brown_tokens)
     cp_t_brown = {} # cp_t -> conditional probabilities of trigrams

     for i in trigrams_brown:
       cp_t_brown[i] = conditional_probability_trigrams(i[0], i[1],i[2], trigrams_brown, bgrams_brown, V_brown)
```

```
[('<s>', 'The', 'Fulton'), ('The', 'Fulton', 'County'), ('Fulton', 'County', 'Grand'), ('County', 'Grand', 'Jury'), ('Grand', 'Jury', 'said'), ('J
```

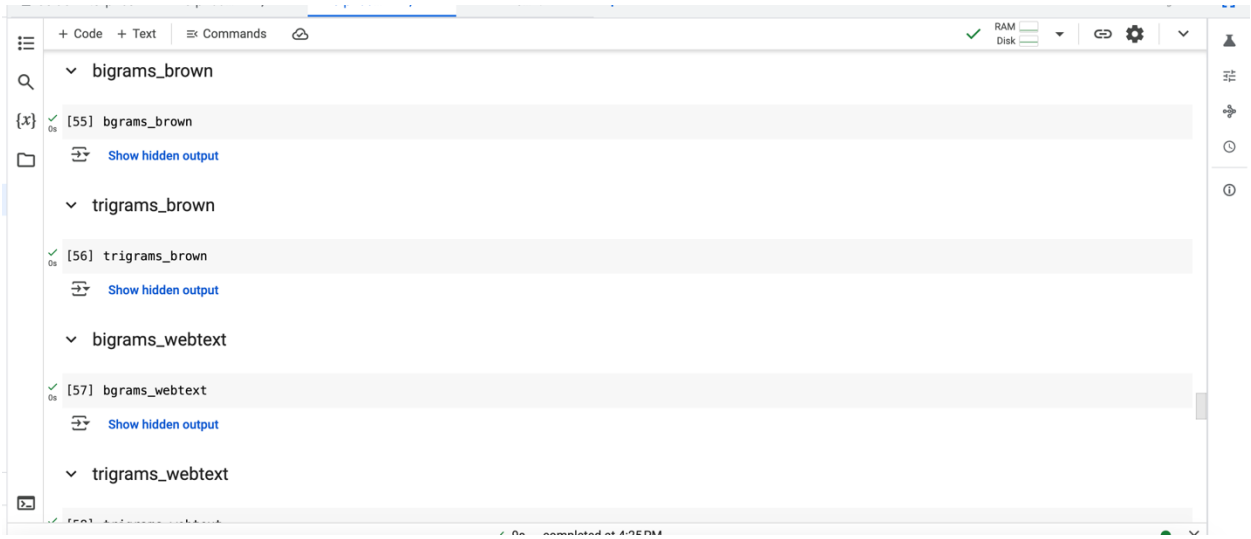
```
[28] cp_t_brown
```

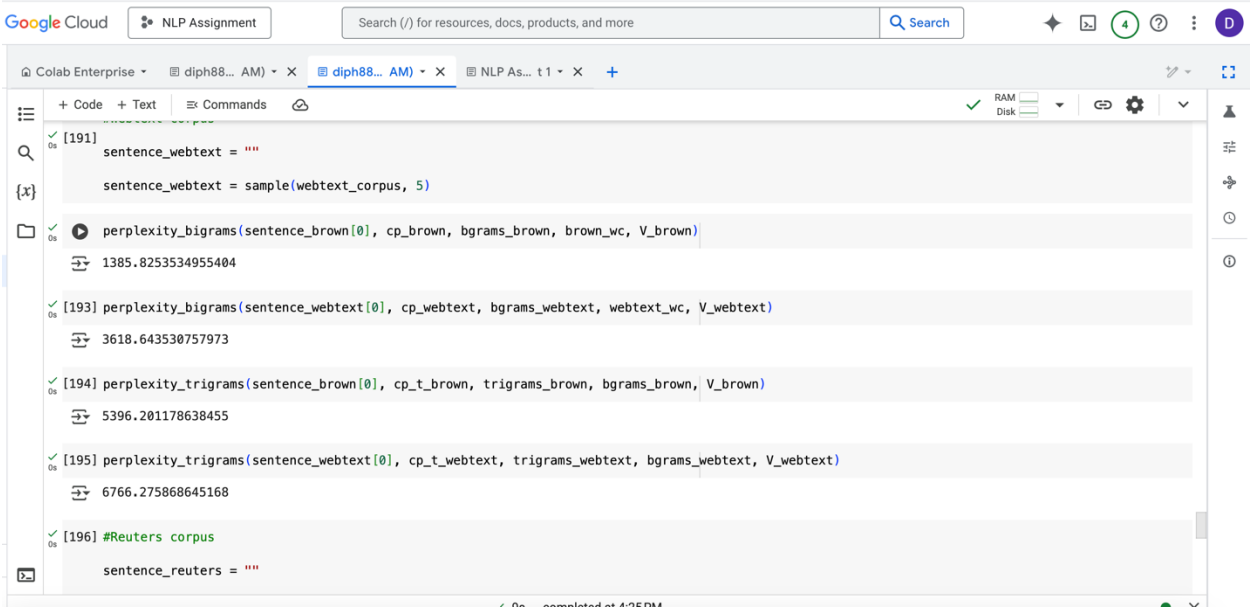
Show hidden output

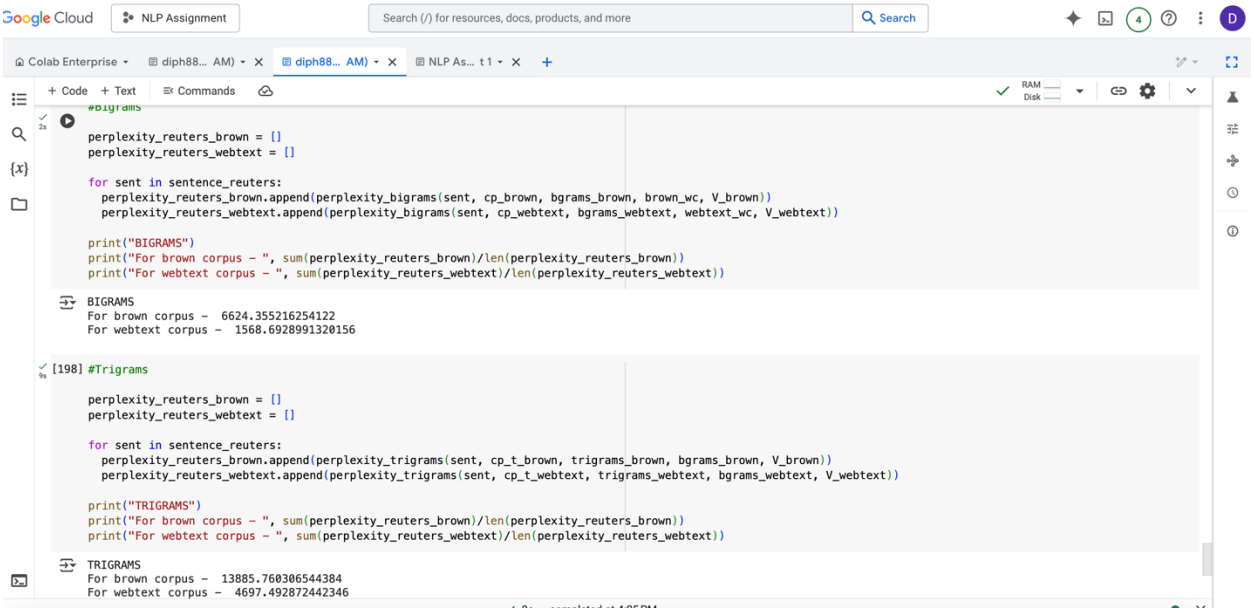




4. Download the Brown and Webtext corpora from NLTK and generate a set of bigrams and trigrams from each corpus to be used for sentence prediction. You should have 4 models to be used with your sentence predictor utility function: bigrams_brown, trigrams_brown, bigrams_webtext, trigrams_webtext. Also download the Reuters corpus, but do not create bigrams or trigrams for it (this will be used for testing later). You should use the following code to download and pre-process the corpora:


+ Code
+ Text
Commands
RAM
Disk
bigrams_brown
[55] bgrams_brown
Show hidden output
trigrams_brown
[56] trigrams_brown
Show hidden output
bigrams_webtext
[57] bgrams_webtext
Show hidden output
trigrams_webtext


Google Cloud
NLP Assignment
Search (/) for resources, docs, products, and more
Search
Colab Enterprise
+ Code
+ Text
Commands
RAM
Disk
[191]
sentence_webtext = ""
sentence_webtext = sample(webtext_corpus, 5)
perplexity_bigrams(sentence_brown[0], cp_brown, bgrams_brown, brown_wc, V_brown)
1385.8253534955404
[193] perplexity_bigrams(sentence_webtext[0], cp_webtext, bgrams_webtext, webtext_wc, V_webtext)
3618.643530757973
[194] perplexity_trigrams(sentence_brown[0], cp_t_brown, trigrams_brown, bgrams_brown, V_brown)
5396.201178638455
[195] perplexity_trigrams(sentence_webtext[0], cp_t_webtext, trigrams_webtext, bgrams_webtext, V_webtext)
6766.275868645168
[196] #Reuters corpus
sentence_reuters = ""



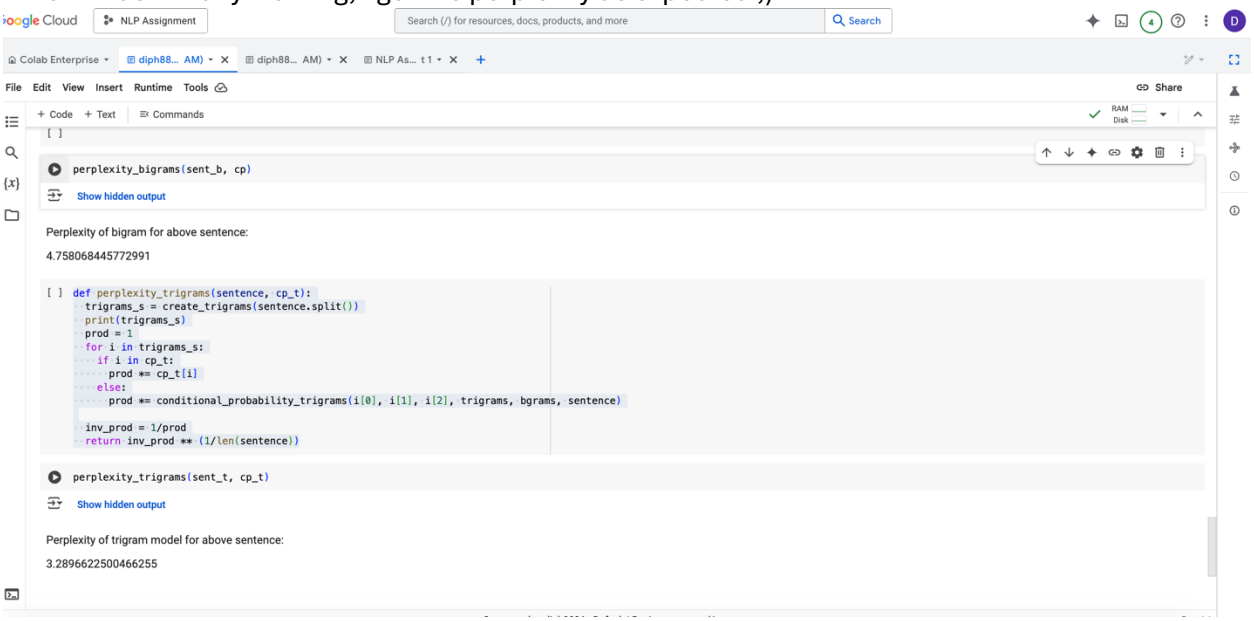
```
#Bigrams

perplexity_reuters_brown = []
perplexity_reuters_webtext = []

for sent in sentence_reuters:
  perplexity_reuters_brown.append(perplexity_bigrams(sent, cp_brown, bgrams_brown, brown_wc, V_brown))
  perplexity_reuters_webtext.append(perplexity_bigrams(sent, cp_webtext, bgrams_webtext, webtext_wc, V_webtext))

print("BIGRAMS")
print("For brown corpus - ", sum(perplexity_reuters_brown)/len(perplexity_reuters_brown))
print("For webtext corpus - ", sum(perplexity_reuters_webtext)/len(perplexity_reuters_webtext))
```

```
BIGRAMS
For brown corpus -  6624.355216254122
For webtext corpus -  1568.6928991320156
```

```
[198] #Trigrams

perplexity_reuters_brown = []
perplexity_reuters_webtext = []

for sent in sentence_reuters:
  perplexity_reuters_brown.append(perplexity_trigrams(sent, cp_t_brown, trigrams_brown, bgrams_brown, V_brown))
  perplexity_reuters_webtext.append(perplexity_trigrams(sent, cp_t_webtext, trigrams_webtext, bgrams_webtext, V_webtext))

print("TRIGRAMS")
print("For brown corpus - ", sum(perplexity_reuters_brown)/len(perplexity_reuters_brown))
print("For webtext corpus - ", sum(perplexity_reuters_webtext)/len(perplexity_reuters_webtext))
```

```
TRIGRAMS
For brown corpus -  13885.760306544384
For webtext corpus -  4697.492872442346
```

Side note:
When I was initially working, I got the perplexity as expected ;)



```
perplexity_bigrams(sent_b, cp)
```

Show hidden output

Perplexity of bigram for above sentence:

4.758068445772991

```
def perplexity_trigrams(sentence, cp_t):
  trigrams_s = create_trigrams(sentence.split())
  print(trigrams_s)
  prod = 1
  for i in trigrams_s:
    if i in cp_t:
      prod *= cp_t[i]
    else:
      prod *= conditional_probability_trigrams(i[0], i[1], i[2], trigrams, bgrams, sentence)

  inv_prod = 1/prod
  return inv_prod ** (1/len(sentence))
```

```
perplexity_trigrams(sent_t, cp_t)
```

Show hidden output

Perplexity of trigram model for above sentence:

3.2896622500466255