Natural Language Processing

CSE 484

HW#2

Ömer Faruk Akduman

1801042094

Content

Homework Concept

In this homework we will develop a statistical language model of Turkish that will use N-grams of Turkish syllables fallowing these steps

1. Download the Turkish Wikipedia dump https://www.kaggle.com/mustfkeskin/turkish-wikipedia-dump

2. Separate each word into its syllables using a program that you can find off the net or implement.

3. Calculate the 1-Gram, 2-Gram, and 3-Gram tables for this set using 95% of the set (If the set is too large, you may use a subset). Note that your N-gram tables will be mostly empty, so you need to use smart ways of storing this information. You also need to use smoothing, which will be GT smoothing that we have learned in the class.

4. Calculate perplexity of the 1-Gram to 3-Gram models using the chain rule with the Markov assumption for each sentence. You will use the remaining 5% of the set for these calculations. Make a table of your findings in your report and explain your results.

5. Produce random sentences for each N-Gram model. You should pick one of the best 5 syllables randomly. Include these random sentences in your report and discuss the produced sentences.

1. Download the dataset & Gain Information about this file & Data Preprocessing

I downloaded data set from given site “Kaggle”

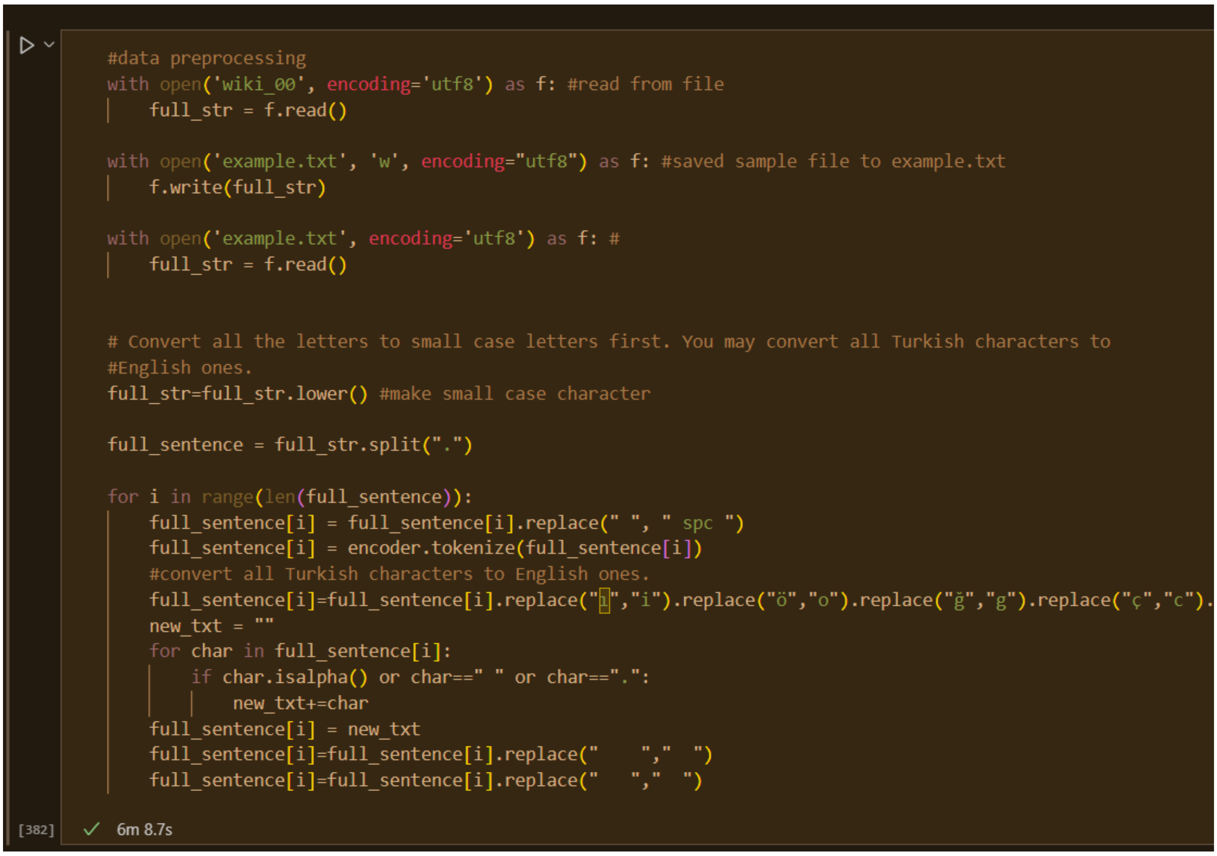
About this file:

This data extracts and cleans text from a Wikipedia database dump. You can access all wikipedia articles written in Turkish language from wikimedia dumps("https://dumps.wikimedia.org/trwiki/")

Note from homework#2 paper:

Convert all the letters to small case letters first. You may convert all Turkish characters to English ones. For example, ş -> s and ğ -> g

I converted Turkish characters below code script



```
#data preprocessing
with open('wiki_00', encoding='utf8') as f: #read from file
    full_str = f.read()

with open('example.txt', 'w', encoding="utf8") as f: #saved sample file to example.txt
    f.write(full_str)

with open('example.txt', encoding='utf8') as f: #
    full_str = f.read()


# Convert all the letters to small case letters first. You may convert all Turkish characters to
#English ones.
full_str=full_str.lower() #make small case character

full_sentence = full_str.split(".")

for i in range(len(full_sentence)):
    full_sentence[i] = full_sentence[i].replace(" ", " spc ")
    full_sentence[i] = encoder.tokenize(full_sentence[i])
    #convert all Turkish characters to English ones.
    full_sentence[i]=full_sentence[i].replace("ı","i").replace("ö","o").replace("ğ","g").replace("ç","c").
    new_txt = ""
    for char in full_sentence[i]:
        if char.isalpha() or char==" " or char==".":
            new_txt+=char
    full_sentence[i] = new_txt
    full_sentence[i]=full_sentence[i].replace("    ","  ")
    full_sentence[i]=full_sentence[i].replace("   ","  ")
```

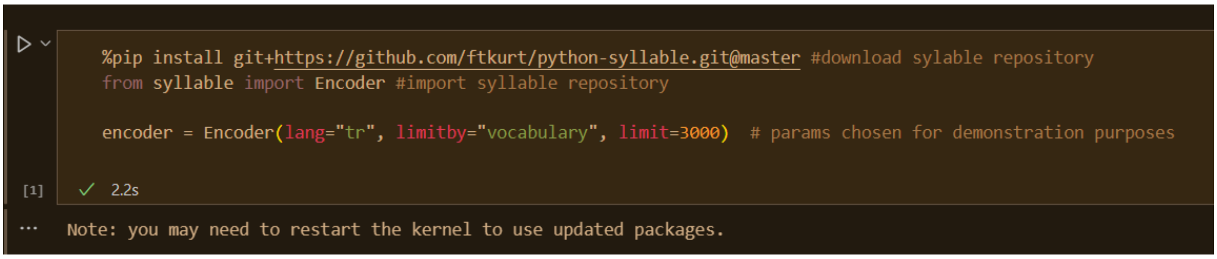
[382] ✓ 6m 8.7s

Note from homework#2 paper:

Just lower case letters and space character will be enough

I divided the characters in the text file according to the sentences, since there is a dot at the end of each sentence, I added the dots to the n-gram, and also spaces were added.

2. Separate each word into its syllables using a program that you can find off the net or implement.



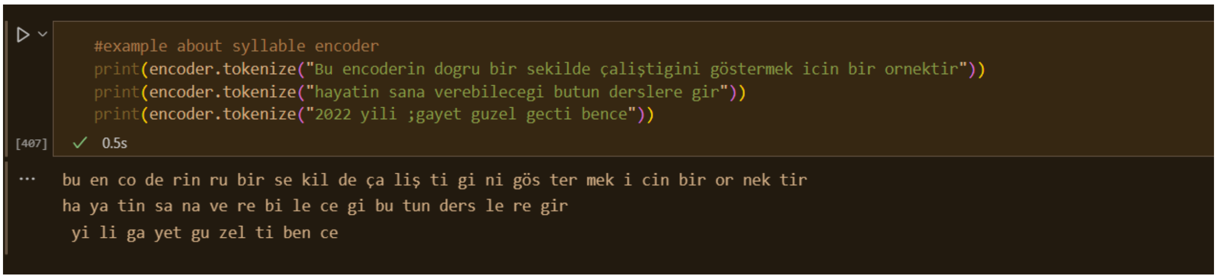
```
%pip install git+https://github.com/ftkurt/python-syllable.git@master #download sylable repository
from syllable import Encoder #import syllable repository

encoder = Encoder(lang="tr", limitby="vocabulary", limit=3000)  # params chosen for demonstration purposes
```

[1] ✓ 2.2s

```
Note: you may need to restart the kernel to use updated packages.
```

I find a GitHub repository to separate word to syllables. It's very useful and easy to use here is the link.

Some examples about syllable repository



```
#example about syllable encoder
print(encoder.tokenize("Bu encoderin dogru bir sekilde çaliştigini göstermek icin bir ornektir"))
print(encoder.tokenize("hayatin sana verebilecegi butun derslere gir"))
print(encoder.tokenize("2022 yili ;gayet guzel gecti bence"))
```

[407] ✓ 0.5s

```
bu en co de rin ru bir se kil de ça liş ti gi ni gös ter mek i cin bir or nek tir
ha ya tin sa na ve re bi le ce gi bu tun ders le re gir
 yi li ga yet gu zel ti ben ce
```

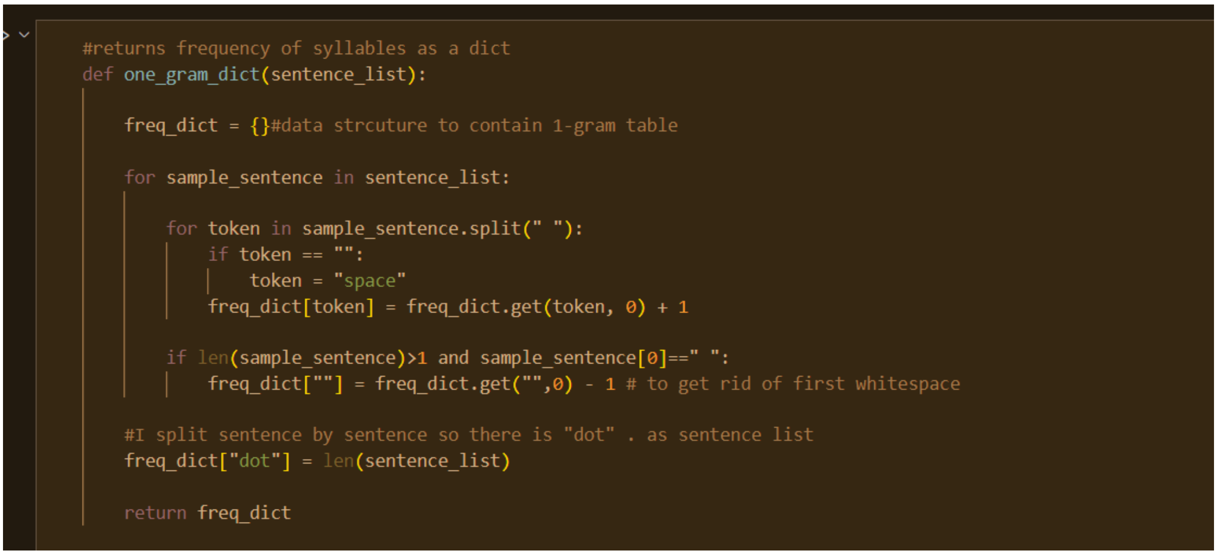
3. Calculate the 1-Gram, 2-Gram, and 3-Gram tables & G-T Smoothing

Hint from homework#2 paper: *Note that your N-gram tables will be mostly empty, so you need to use smart ways of storing this information.*

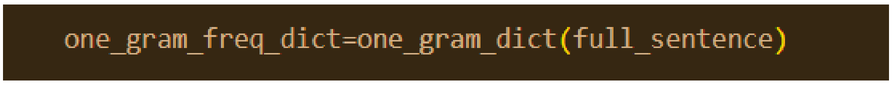
To reduce the space complexity I calculate 1,2,3- Gram tables by **used the dictionary** structure in Python language. So, there is no syllable pair, it does not take up disk space. Hereby I maximize space efficiency. Also in python dictionary container act like hasp map so access any element average time complexity is O(1)

A) 1-Gram

1-Gram Python Script

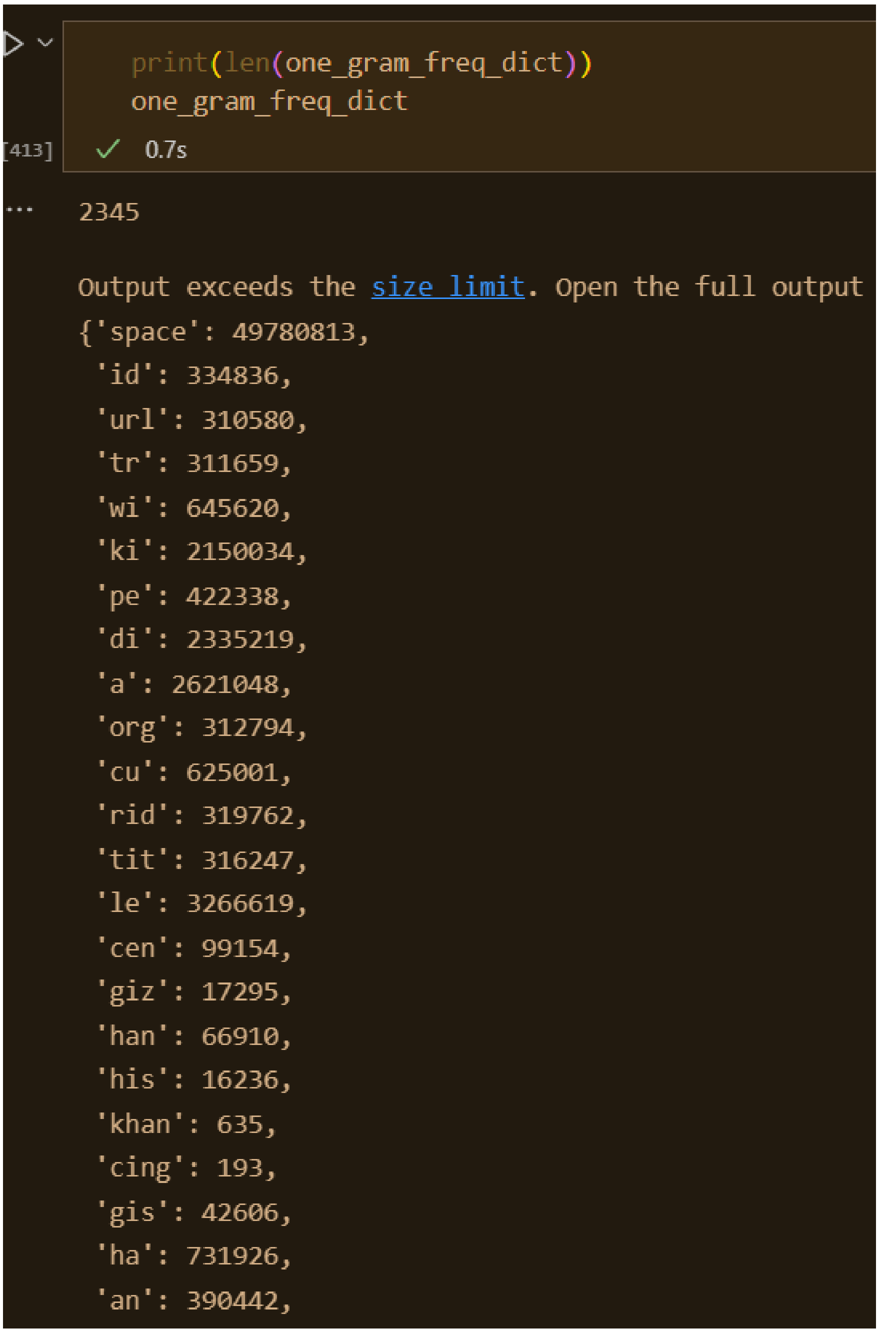


```
#returns frequency of syllables as a dict
def one_gram_dict(sentence_list):

    freq_dict = {}#data strcuture to contain 1-gram table

    for sample_sentence in sentence_list:

        for token in sample_sentence.split(" "):
            if token == "":
                token = "space"
            freq_dict[token] = freq_dict.get(token, 0) + 1

        if len(sample_sentence)>1 and sample_sentence[0]==" ":
            freq_dict[""] = freq_dict.get("",0) - 1 # to get rid of first whitespace

    #I split sentence by sentence so there is "dot" . as sentence list
    freq_dict["dot"] = len(sentence_list)

    return freq_dict
```



```
one_gram_freq_dict=one_gram_dict(full_sentence)
```

1-Gram Table

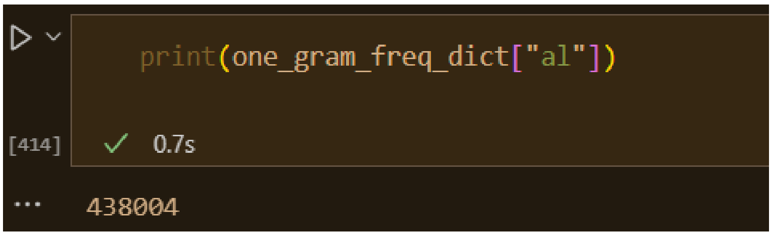


```
print(len(one_gram_freq_dict))
one_gram_freq_dict
```

[413] ✓ 0.7s

```
2345

Output exceeds the size limit. Open the full output
{'space': 49780813,
 'id': 334836,
 'url': 310580,
 'tr': 311659,
 'wi': 645620,
 'ki': 2150034,
 'pe': 422338,
 'di': 2335219,
 'a': 2621048,
 'org': 312794,
 'cu': 625001,
 'rid': 319762,
 'tit': 316247,
 'le': 3266619,
 'cen': 99154,
 'giz': 17295,
 'han': 66910,
 'his': 16236,
 'khan': 635,
 'cing': 193,
 'gis': 42606,
 'ha': 731926,
 'an': 390442,
```

“space” = “ “



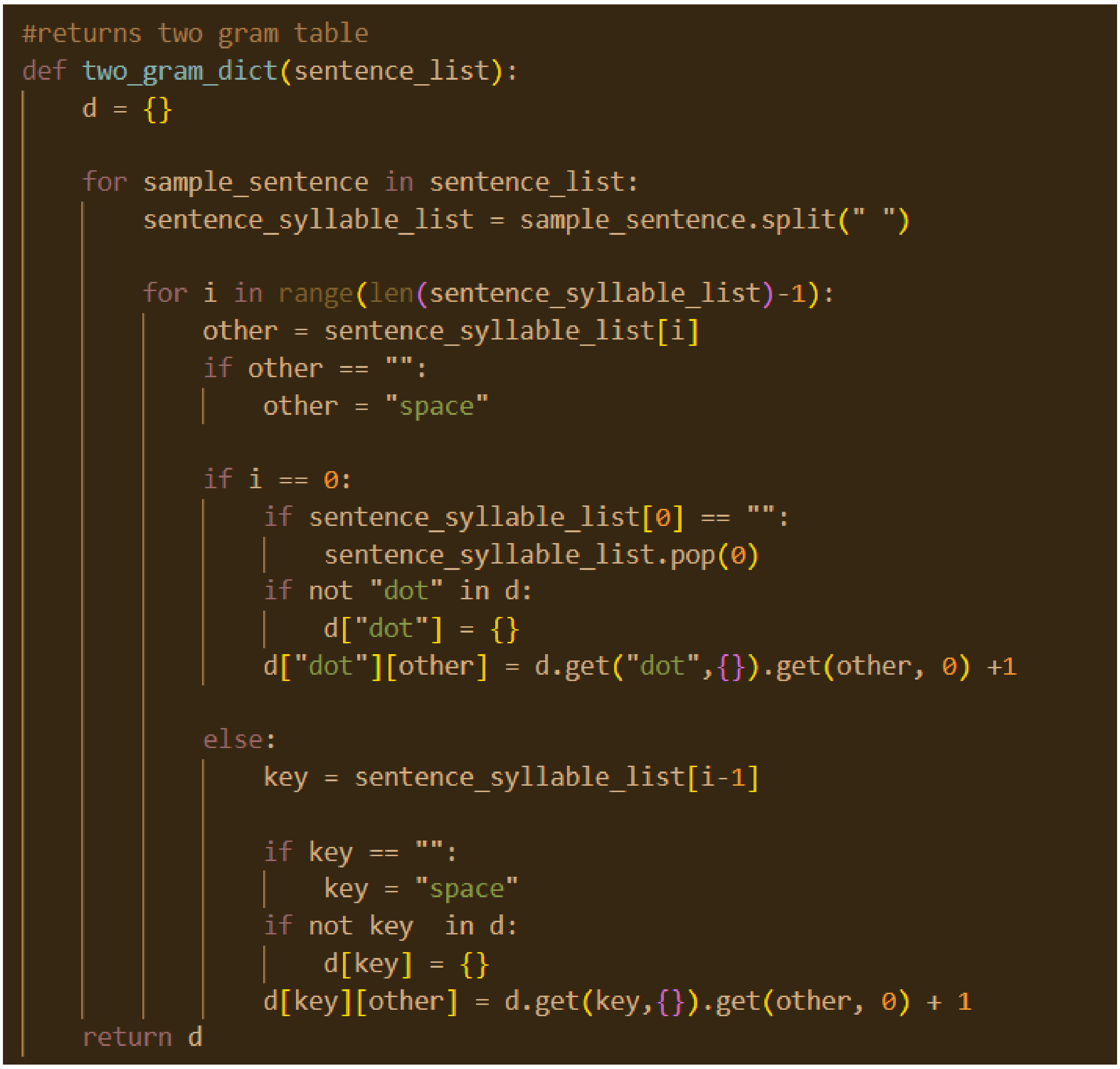
```
print(one_gram_freq_dict["al"])
```

[414] ✓ 0.7s

```
438004
```

B) 2-Gram

2-Gram Python Script



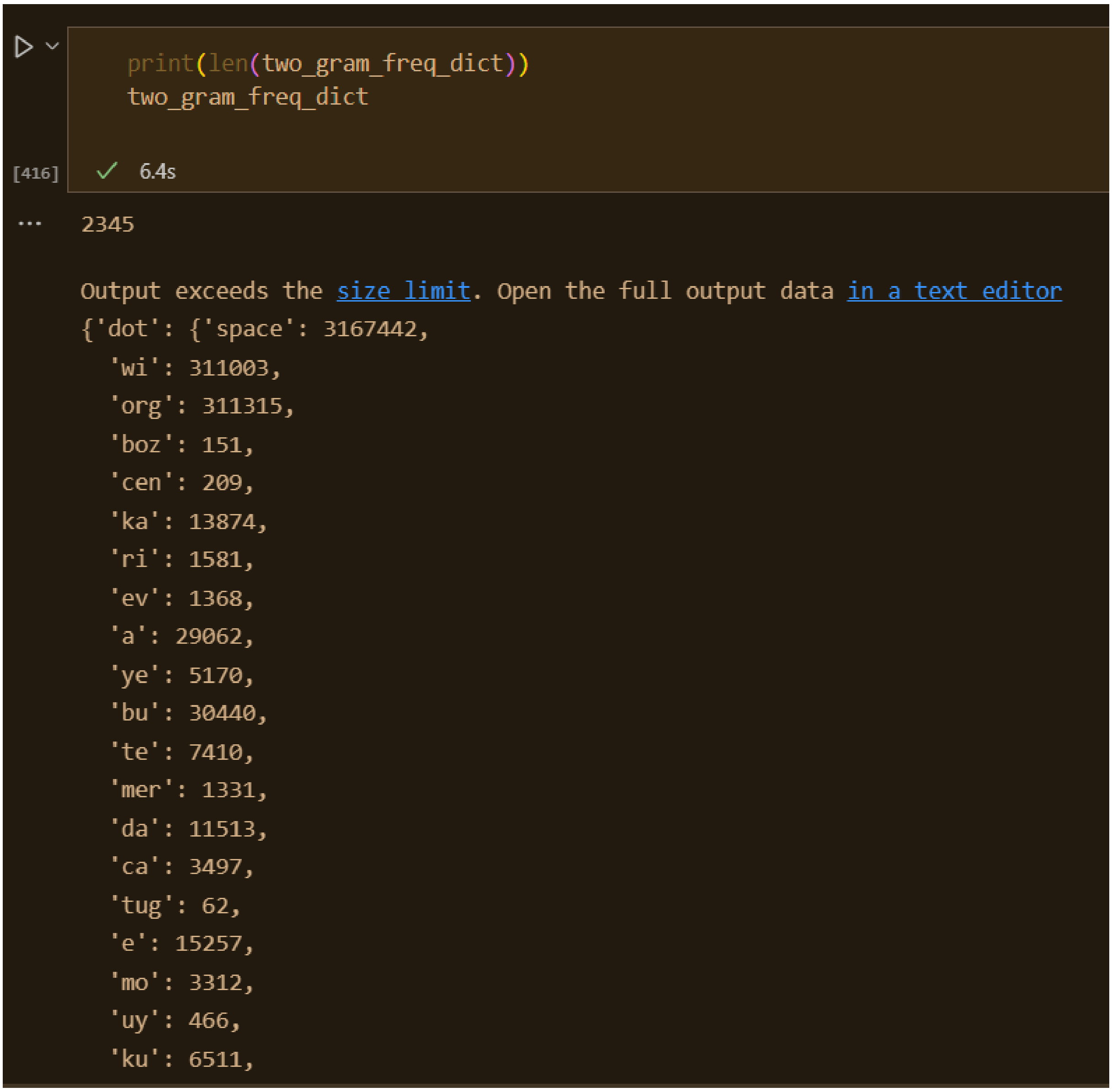
```
#returns two gram table
def two_gram_dict(sentence_list):
    d = {}

    for sample_sentence in sentence_list:
        sentence_syllable_list = sample_sentence.split(" ")

        for i in range(len(sentence_syllable_list)-1):
            other = sentence_syllable_list[i]
            if other == "":
                other = "space"

            if i == 0:
                if sentence_syllable_list[0] == "":
                    sentence_syllable_list.pop(0)
                if not "dot" in d:
                    d["dot"] = {}
                d["dot"][other] = d.get("dot",{}).get(other, 0) +1

            else:
                key = sentence_syllable_list[i-1]

                if key == "":
                    key = "space"
                if not key  in d:
                    d[key] = {}
                d[key][other] = d.get(key,{}).get(other, 0) + 1
    return d
```

2-Gram Table

In here first syllable is dot and there are 3167442 pair “. ”

311003 pair “.wi”

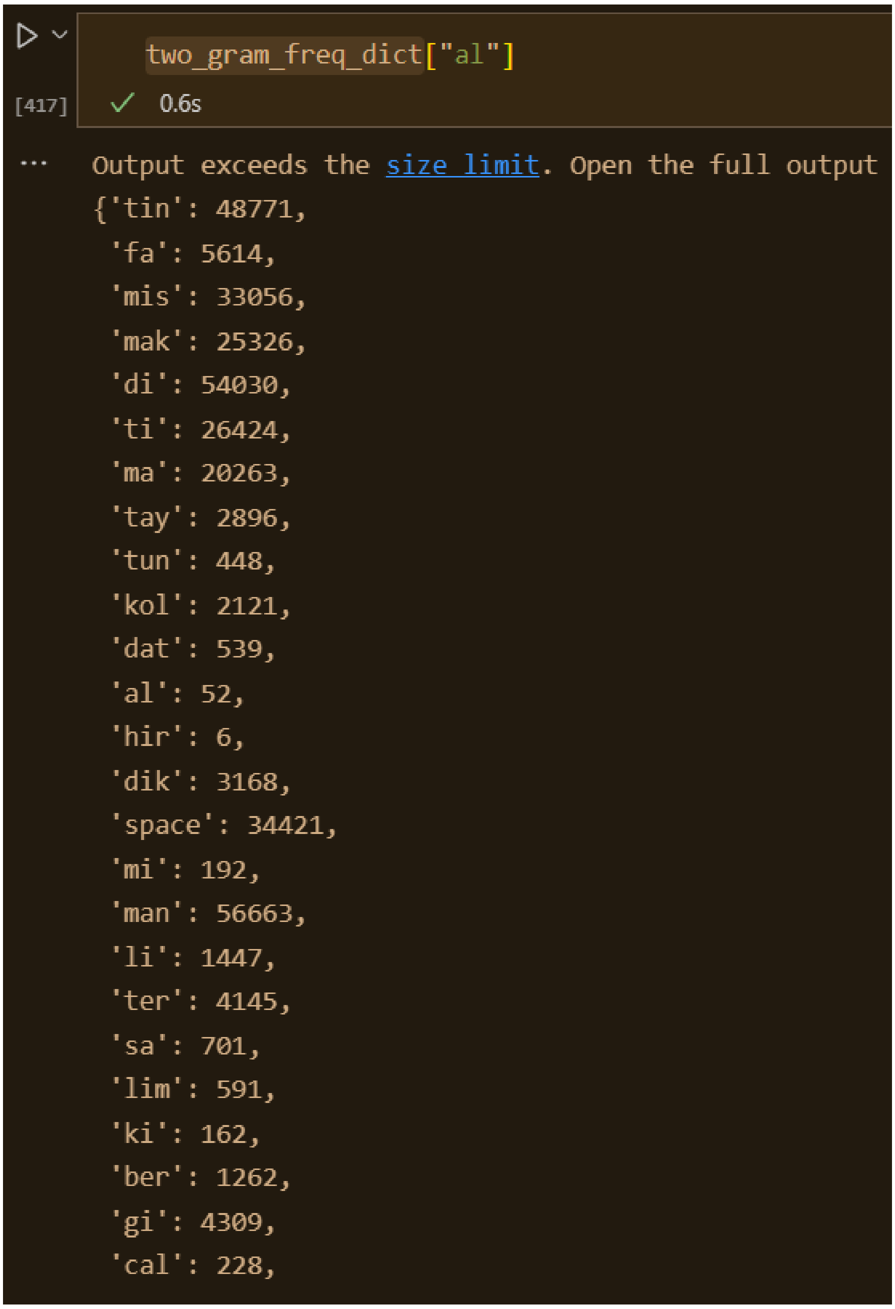
311315 pair “.org”



```
print(len(two_gram_freq_dict))
two_gram_freq_dict
```

[416] ✓ 6.4s

```
2345

Output exceeds the size limit. Open the full output data in a text editor
{'dot': {'space': 3167442,
  'wi': 311003,
  'org': 311315,
  'boz': 151,
  'cen': 209,
  'ka': 13874,
  'ri': 1581,
  'ev': 1368,
  'a': 29062,
  'ye': 5170,
  'bu': 30440,
  'te': 7410,
  'mer': 1331,
  'da': 11513,
  'ca': 3497,
  'tug': 62,
  'e': 15257,
  'mo': 3312,
  'uy': 466,
  'ku': 6511,
```



```
two_gram_freq_dict["al"]
```

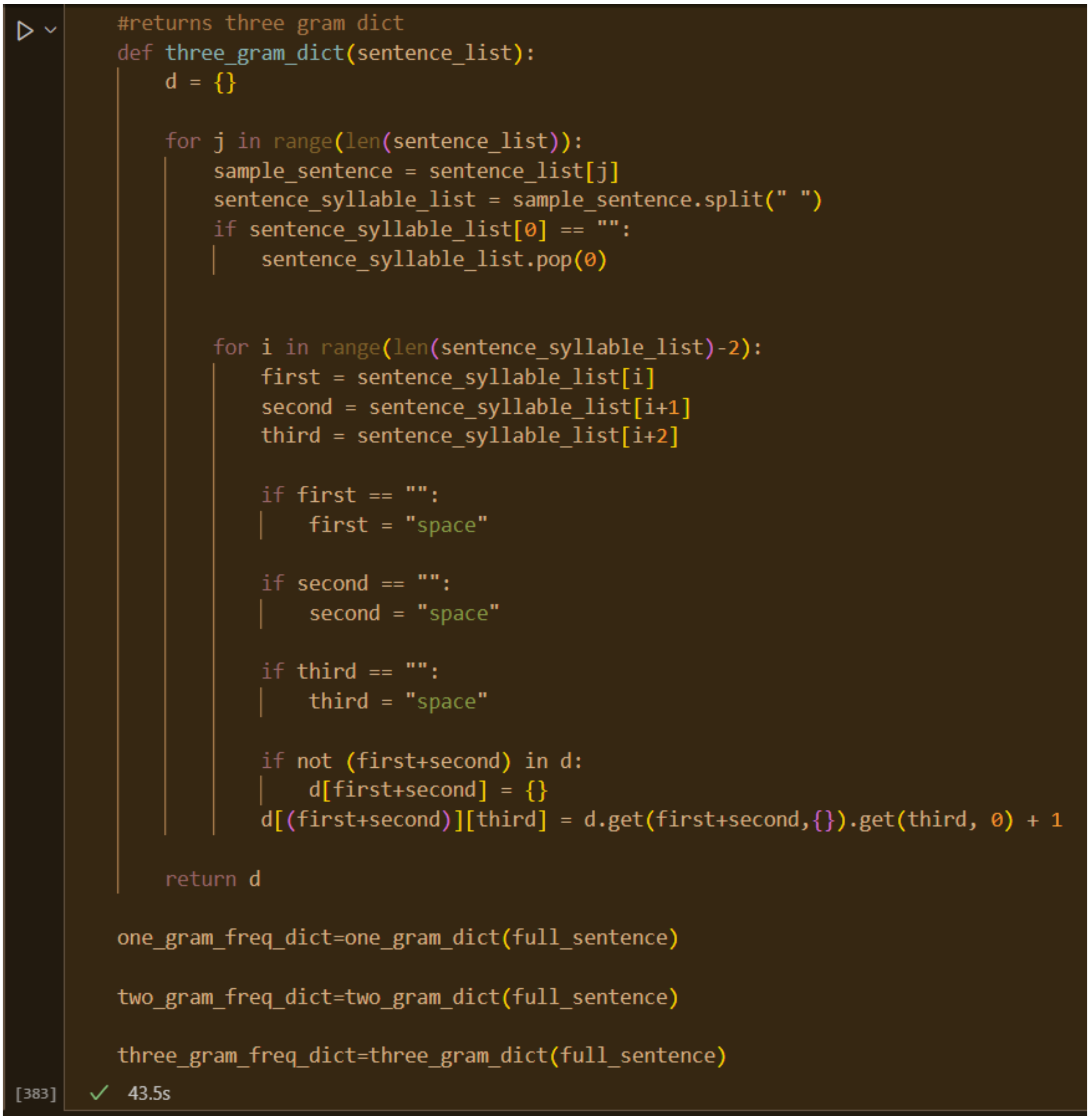
```
Output exceeds the size limit. Open the full output
{'tin': 48771,
 'fa': 5614,
 'mis': 33056,
 'mak': 25326,
 'di': 54030,
 'ti': 26424,
 'ma': 20263,
 'tay': 2896,
 'tun': 448,
 'kol': 2121,
 'dat': 539,
 'al': 52,
 'hir': 6,
 'dik': 3168,
 'space': 34421,
 'mi': 192,
 'man': 56663,
 'li': 1447,
 'ter': 4145,
 'sa': 701,
 'lim': 591,
 'ki': 162,
 'ber': 1262,
 'gi': 4309,
 'cal': 228,
```

For example in here first syllable is “al”

There are 48771 “altin” syllable pair

C) 3-Gram

Python Script



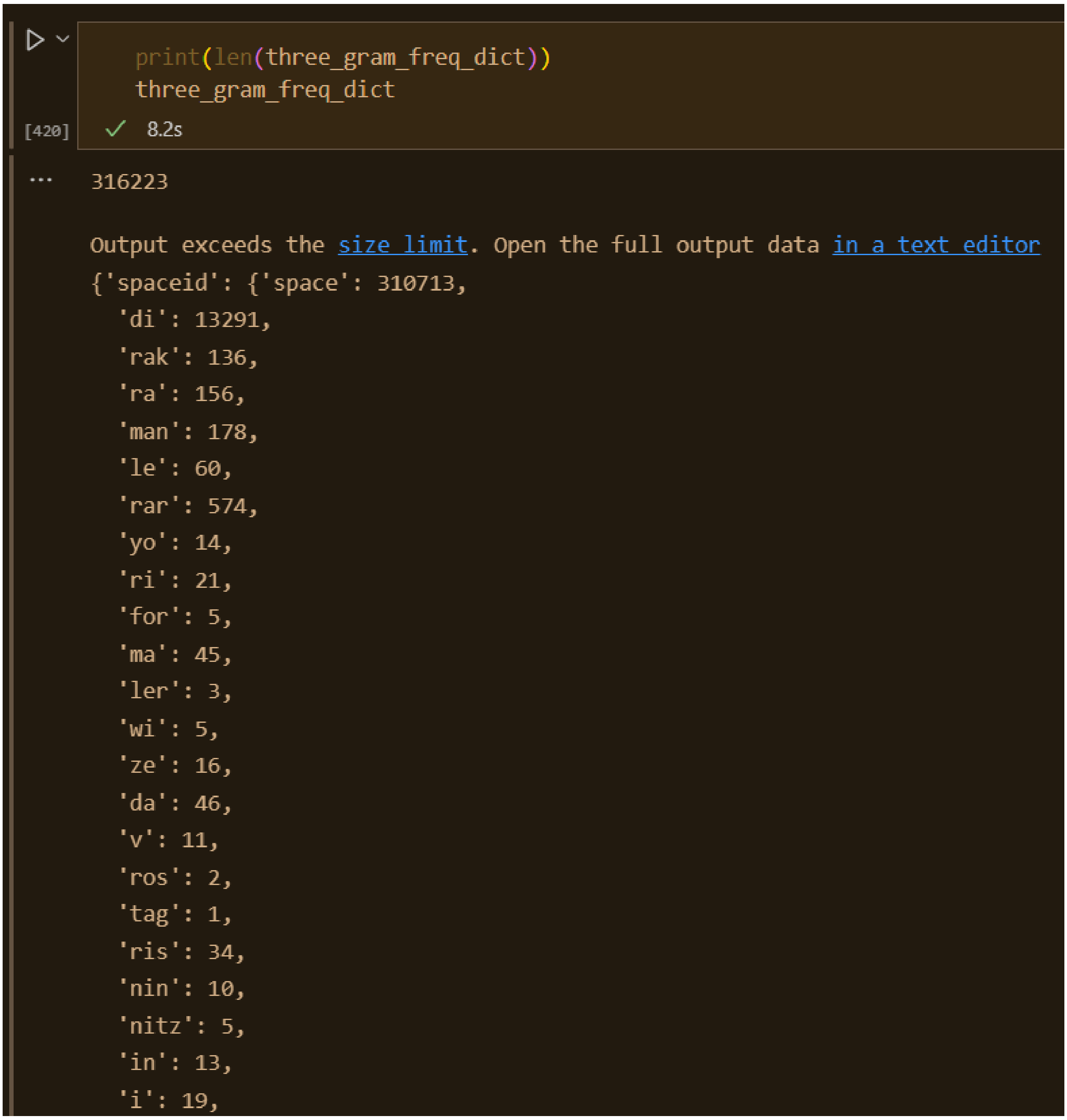
```python
#returns three gram dict
def three_gram_dict(sentence_list):
    d = {}

    for j in range(len(sentence_list)):
        sample_sentence = sentence_list[j]
        sentence_syllable_list = sample_sentence.split(" ")
        if sentence_syllable_list[0] == "":
            sentence_syllable_list.pop(0)


        for i in range(len(sentence_syllable_list)-2):
            first = sentence_syllable_list[i]
            second = sentence_syllable_list[i+1]
            third = sentence_syllable_list[i+2]

            if first == "":
                first = "space"

            if second == "":
                second = "space"

            if third == "":
                third = "space"

            if not (first+second) in d:
                d[first+second] = {}
            d[(first+second)][third] = d.get(first+second,{}).get(third, 0) + 1

    return d

one_gram_freq_dict=one_gram_dict(full_sentence)

two_gram_freq_dict=two_gram_dict(full_sentence)

three_gram_freq_dict=three_gram_dict(full_sentence)
```

3-Gram Table

In here first syllable is dot and there are 310713 pair “ id ”

13291 pair “ iddi”

136 pair “ idrak”



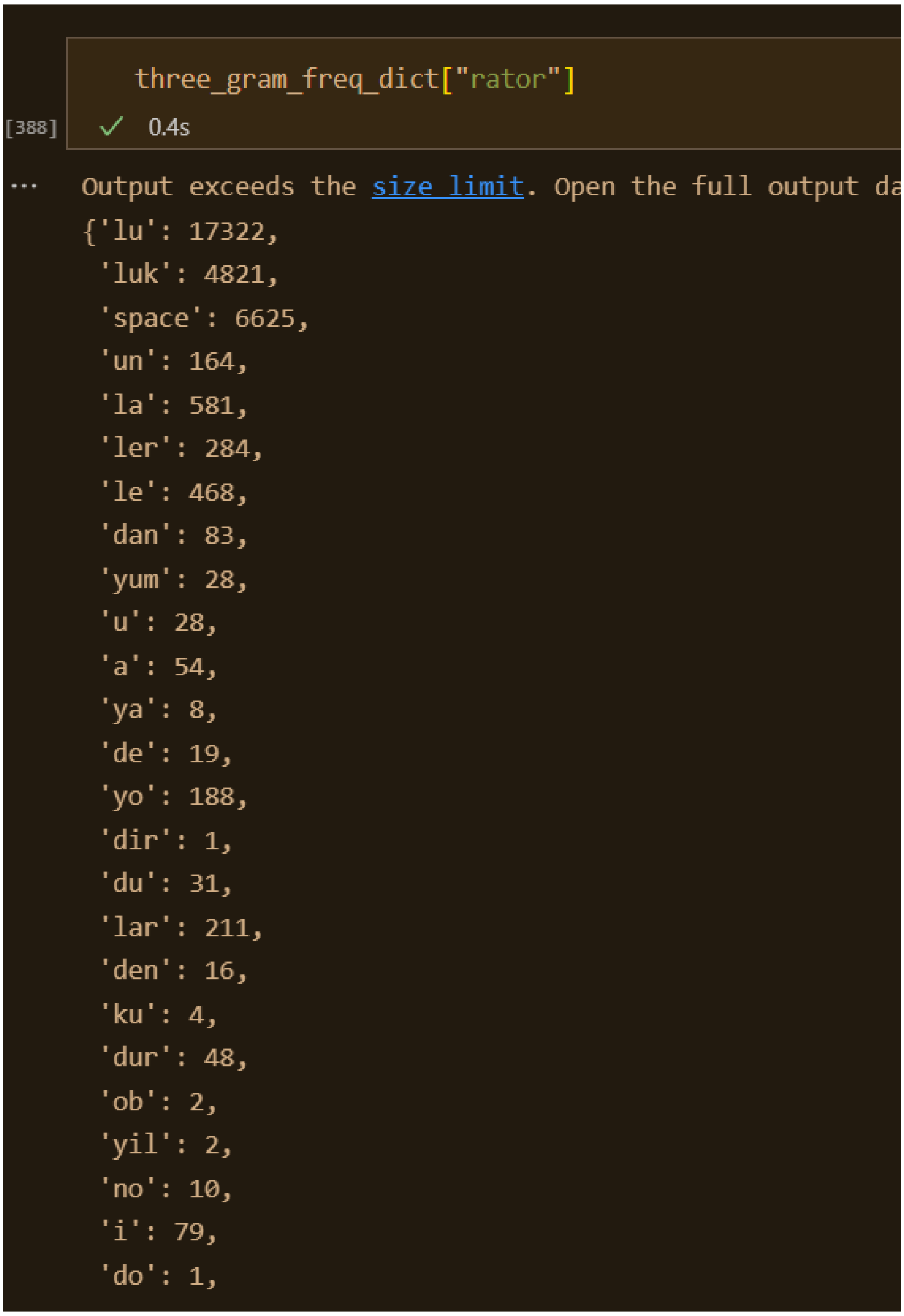
```
print(len(three_gram_freq_dict))
three_gram_freq_dict
```

```
316223

Output exceeds the size limit. Open the full output data in a text editor
{'spaceid': {'space': 310713,
  'di': 13291,
  'rak': 136,
  'ra': 156,
  'man': 178,
  'le': 60,
  'rar': 574,
  'yo': 14,
  'ri': 21,
  'for': 5,
  'ma': 45,
  'ler': 3,
  'wi': 5,
  'ze': 16,
  'da': 46,
  'v': 11,
  'ros': 2,
  'tag': 1,
  'ris': 34,
  'nin': 10,
  'nitz': 5,
  'in': 13,
  'i': 19,
```

In here first syllable is dot and there are 17322 pair “ratorlu”

4821 pair “ratorluk”

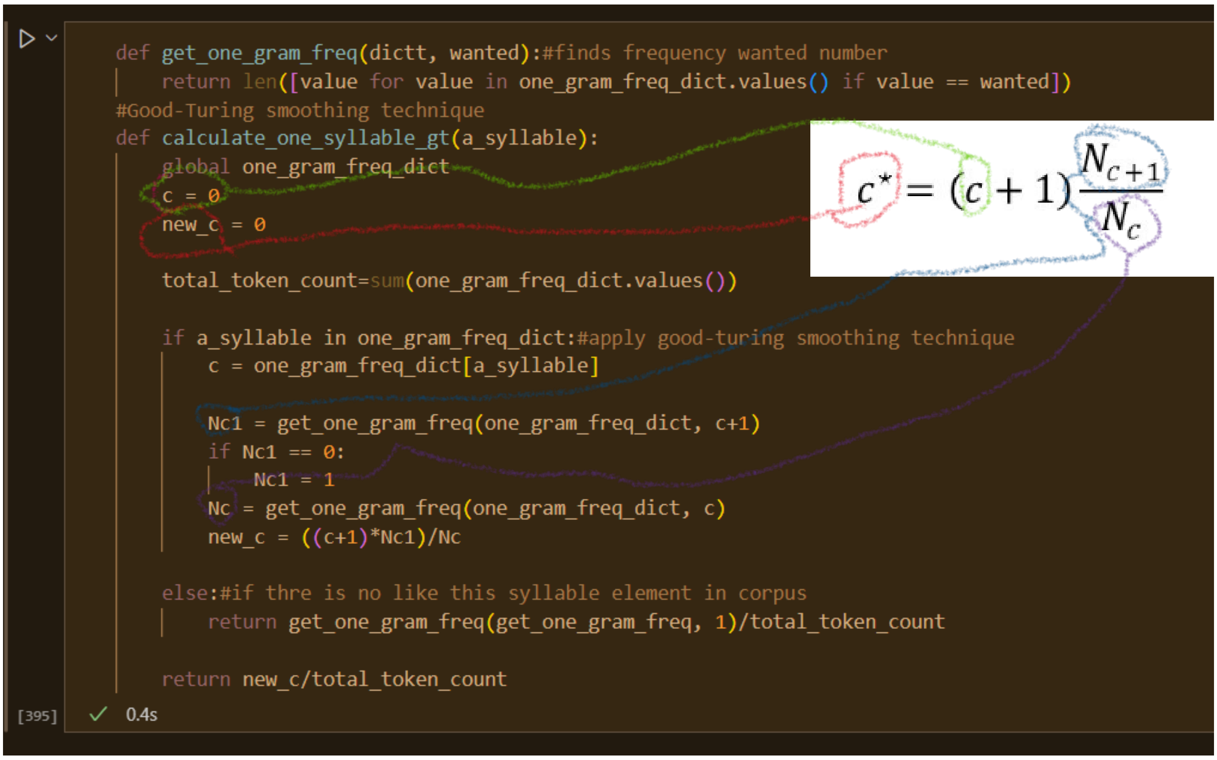
6625 pair “rator ”



```
[388]   three_gram_freq_dict["rator"]
        ✓ 0.4s

...   Output exceeds the size limit. Open the full output da
      {'lu': 17322,
       'luk': 4821,
       'space': 6625,
       'un': 164,
       'la': 581,
       'ler': 284,
       'le': 468,
       'dan': 83,
       'yum': 28,
       'u': 28,
       'a': 54,
       'ya': 8,
       'de': 19,
       'yo': 188,
       'dir': 1,
       'du': 31,
       'lar': 211,
       'den': 16,
       'ku': 4,
       'dur': 48,
       'ob': 2,
       'yil': 2,
       'no': 10,
       'i': 79,
       'do': 1,
```

D) G-T Smoothing

G-T Smoothing Python Script

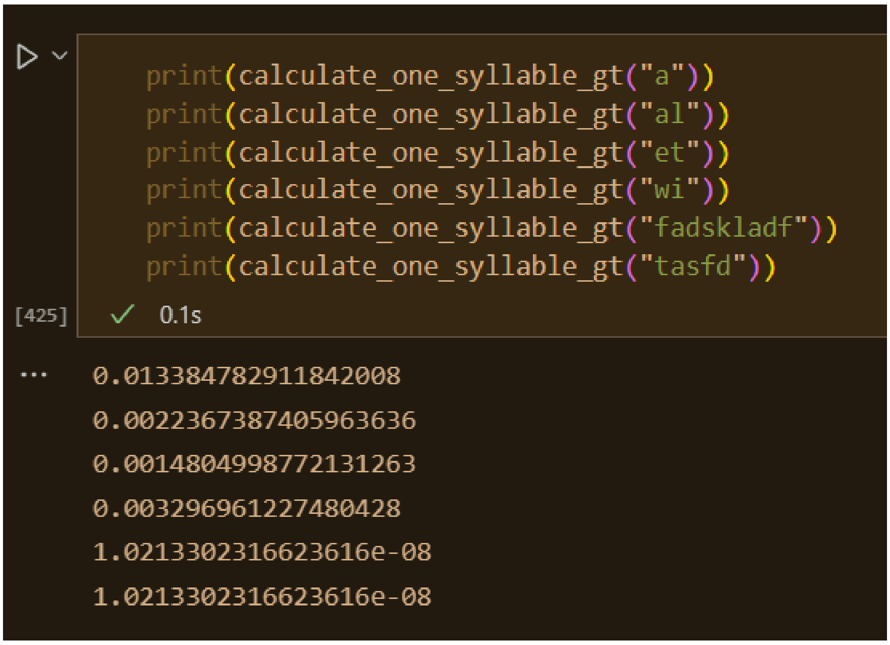


```python
def get_one_gram_freq(dictt, wanted):#finds frequency wanted number
    return len([value for value in one_gram_freq_dict.values() if value == wanted])
#Good-Turing smoothing technique
def calculate_one_syllable_gt(a_syllable):
    global one_gram_freq_dict
    c = 0
    new_c = 0

    total_token_count=sum(one_gram_freq_dict.values())

    if a_syllable in one_gram_freq_dict:#apply good-turing smoothing technique
        c = one_gram_freq_dict[a_syllable]

        Nc1 = get_one_gram_freq(one_gram_freq_dict, c+1)
        if Nc1 == 0:
            Nc1 = 1
        Nc = get_one_gram_freq(one_gram_freq_dict, c)
        new_c = ((c+1)*Nc1)/Nc

    else:#if thre is no like this syllable element in corpus
        return get_one_gram_freq(get_one_gram_freq, 1)/total_token_count

    return new_c/total_token_count
```

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

[395] ✓ 0.4s

G-T Smoothing Drive Script



```python
print(calculate_one_syllable_gt("a"))
print(calculate_one_syllable_gt("al"))
print(calculate_one_syllable_gt("et"))
print(calculate_one_syllable_gt("wi"))
print(calculate_one_syllable_gt("fadskladf"))
print(calculate_one_syllable_gt("tasfd"))
```

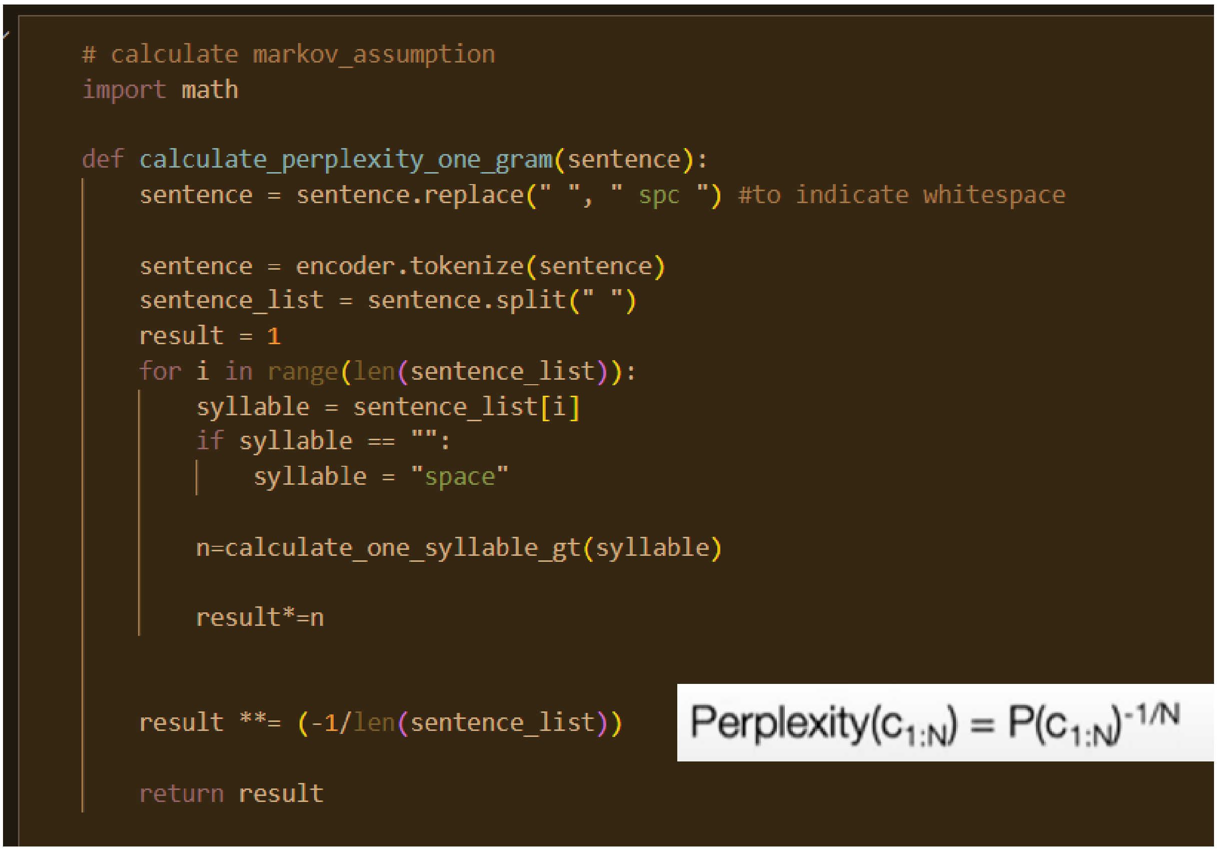
[425] ✓ 0.1s

```
0.013384782911842008
0.0022367387405963636
0.0014804998772131263
0.003296961227480428
1.0213302316623616e-08
1.0213302316623616e-08
```

4-Calculate perplexity of the 1-Gram to 3-Gram models using the chain rule with the Markov assumption

A)1-Gram

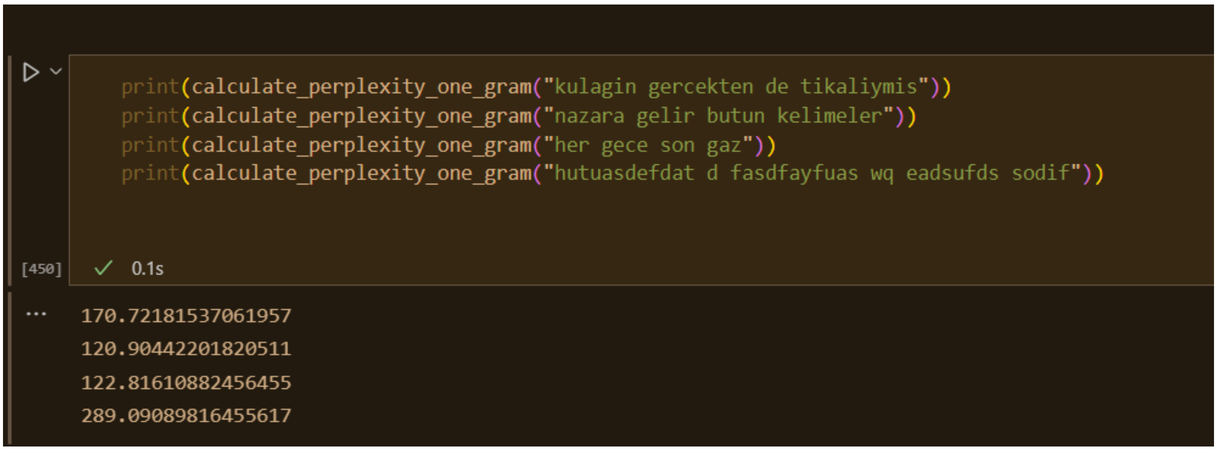
1-Gram Perplexity Python Script



```python
# calculate markov_assumption
import math

def calculate_perplexity_one_gram(sentence):
    sentence = sentence.replace(" ", " spc ") #to indicate whitespace

    sentence = encoder.tokenize(sentence)
    sentence_list = sentence.split(" ")
    result = 1
    for i in range(len(sentence_list)):
        syllable = sentence_list[i]
        if syllable == "":
            syllable = "space"

        n=calculate_one_syllable_gt(syllable)

        result*=n


    result **= (-1/len(sentence_list))

    return result
```

$$\text{Perplexity}(c_{1:N}) = P(c_{1:N})^{-1/N}$$

1-Gram Perplexity Drive Script

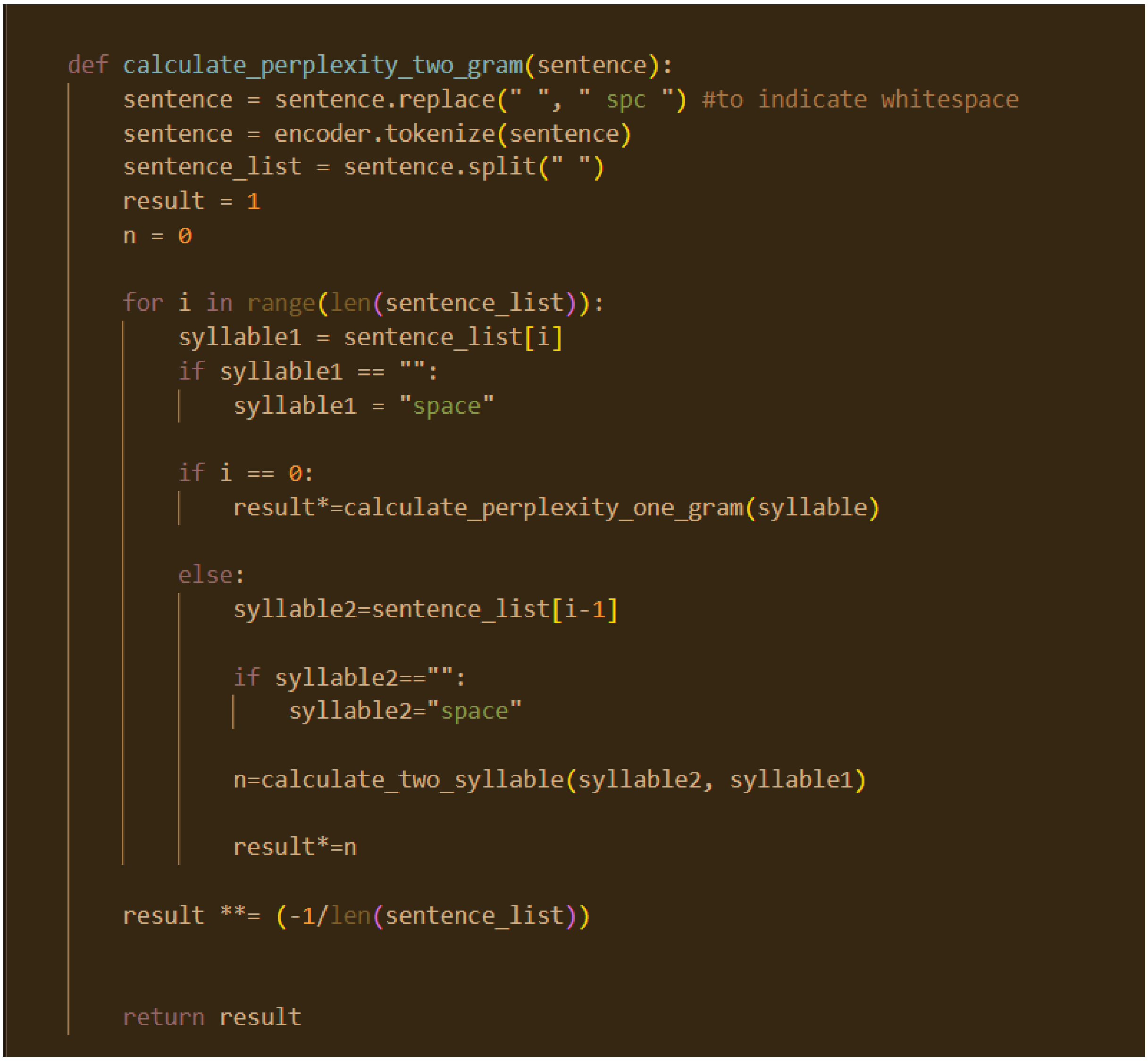


```python
print(calculate_perplexity_one_gram("kulagin gercekten de tikaliymis"))
print(calculate_perplexity_one_gram("nazara gelir butun kelimeler"))
print(calculate_perplexity_one_gram("her gece son gaz"))
print(calculate_perplexity_one_gram("hutuasdefdat d fasdfayfuas wq eadsufds sodif"))
```

```
[450]  ✓ 0.1s

170.72181537061957
120.90442201820511
122.81610882456455
289.09089816455617
```

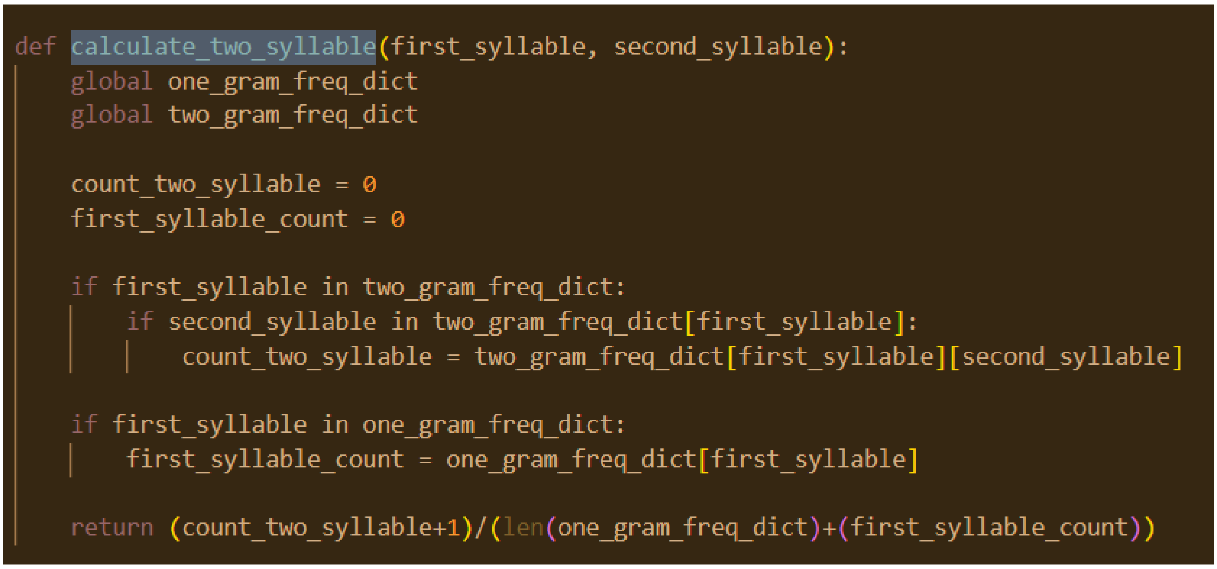
B)2-Gram

2-Gram Perplexity Python Script



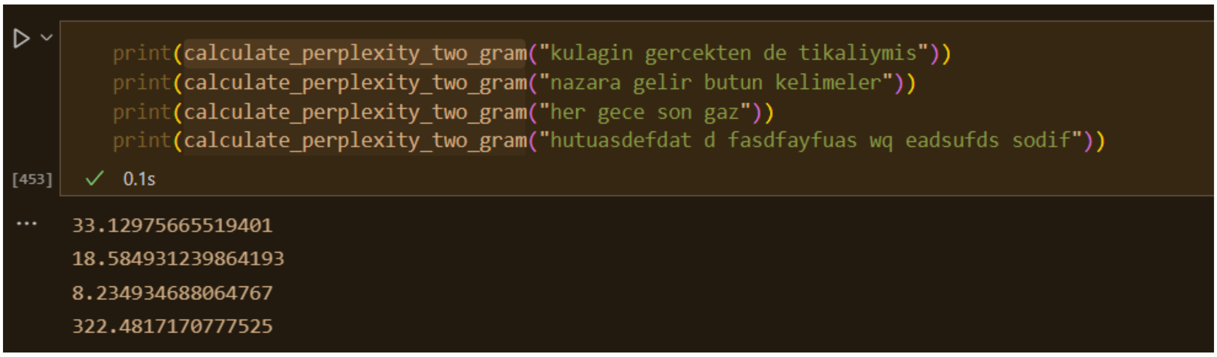
```python
def calculate_perplexity_two_gram(sentence):
    sentence = sentence.replace(" ", " spc ") #to indicate whitespace
    sentence = encoder.tokenize(sentence)
    sentence_list = sentence.split(" ")
    result = 1
    n = 0

    for i in range(len(sentence_list)):
        syllable1 = sentence_list[i]
        if syllable1 == "":
            syllable1 = "space"

        if i == 0:
            result*=calculate_perplexity_one_gram(syllable)

        else:
            syllable2=sentence_list[i-1]

            if syllable2=="":
                syllable2="space"

            n=calculate_two_syllable(syllable2, syllable1)

            result*=n

    result **= (-1/len(sentence_list))


    return result
```

calculate_two_syllable() function: finds the probability of P(second_syllable|first_syllable)



```python
def calculate_two_syllable(first_syllable, second_syllable):
    global one_gram_freq_dict
    global two_gram_freq_dict

    count_two_syllable = 0
    first_syllable_count = 0

    if first_syllable in two_gram_freq_dict:
        if second_syllable in two_gram_freq_dict[first_syllable]:
            count_two_syllable = two_gram_freq_dict[first_syllable][second_syllable]

    if first_syllable in one_gram_freq_dict:
        first_syllable_count = one_gram_freq_dict[first_syllable]

    return (count_two_syllable+1)/(len(one_gram_freq_dict)+(first_syllable_count))
```

2-Gram Perplexity Drive Script

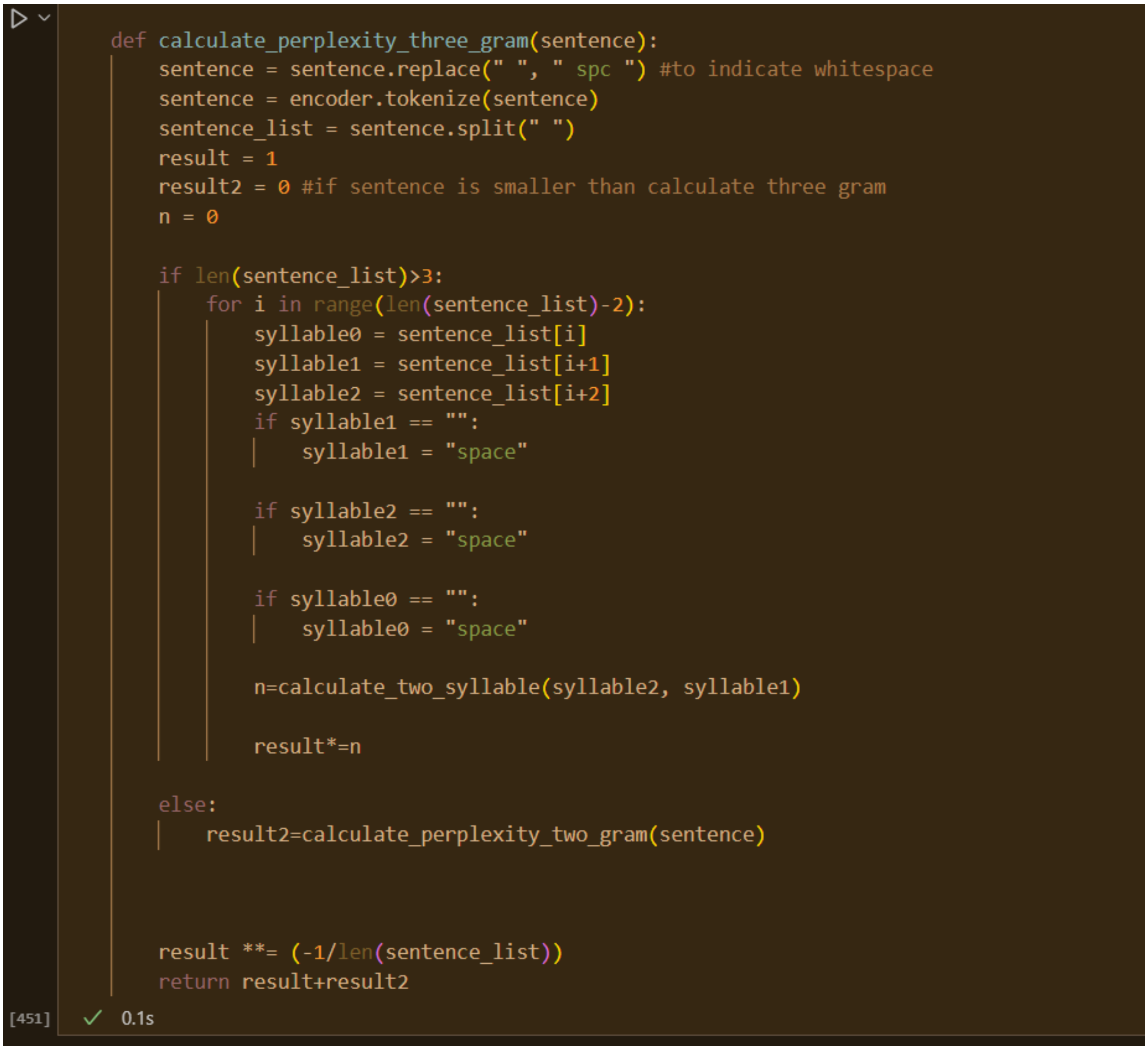


```python
print(calculate_perplexity_two_gram("kulagin gercekten de tikaliymis"))
print(calculate_perplexity_two_gram("nazara gelir butun kelimeler"))
print(calculate_perplexity_two_gram("her gece son gaz"))
print(calculate_perplexity_two_gram("hutuasdefdat d fasdfayfuas wq eadsufds sodif"))
```

```
33.12975665519401
18.584931239864193
8.234934688064767
322.4817170777525
```

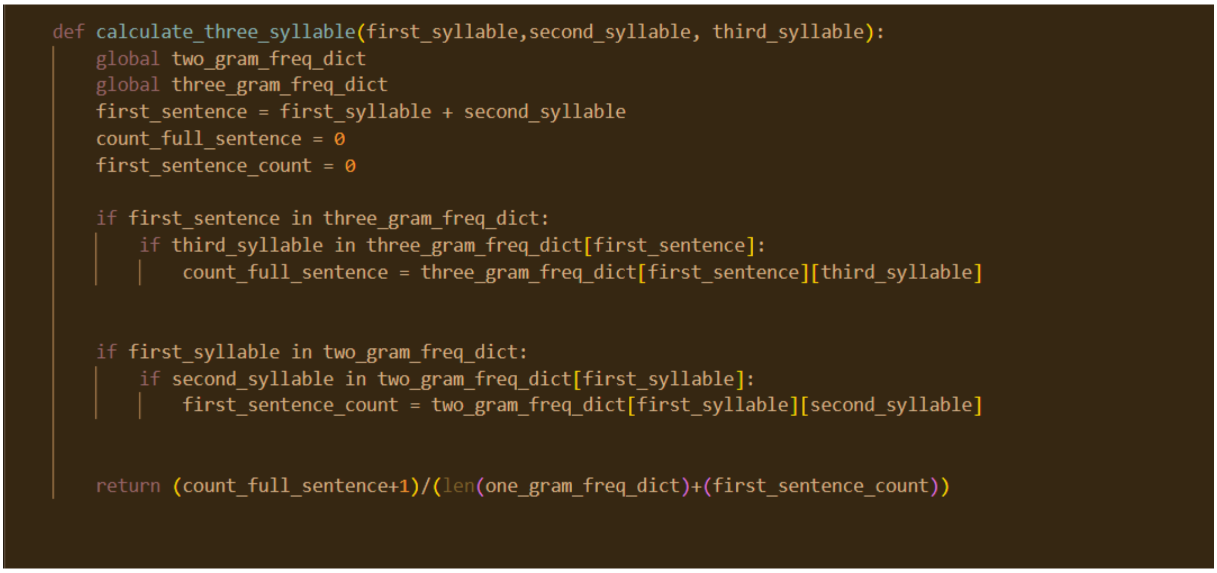
C)3-Gram

3-Gram Perplexity Python Script



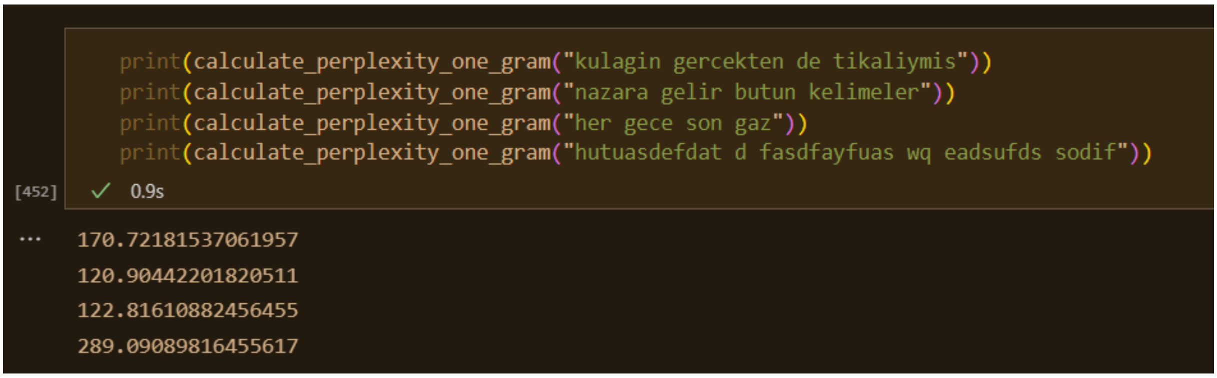
```python
def calculate_perplexity_three_gram(sentence):
    sentence = sentence.replace(" ", " spc ") #to indicate whitespace
    sentence = encoder.tokenize(sentence)
    sentence_list = sentence.split(" ")
    result = 1
    result2 = 0 #if sentence is smaller than calculate three gram
    n = 0

    if len(sentence_list)>3:
        for i in range(len(sentence_list)-2):
            syllable0 = sentence_list[i]
            syllable1 = sentence_list[i+1]
            syllable2 = sentence_list[i+2]
            if syllable1 == "":
                syllable1 = "space"

            if syllable2 == "":
                syllable2 = "space"

            if syllable0 == "":
                syllable0 = "space"

            n=calculate_two_syllable(syllable2, syllable1)

            result*=n

    else:
        result2=calculate_perplexity_two_gram(sentence)


    result **= (-1/len(sentence_list))
    return result+result2
```

calculate_three_syllable(f_s, s_s, t_s): finds the probability of P(third_syllable|first_syllable+second_syllable)



```python
def calculate_three_syllable(first_syllable,second_syllable, third_syllable):
    global two_gram_freq_dict
    global three_gram_freq_dict
    first_sentence = first_syllable + second_syllable
    count_full_sentence = 0
    first_sentence_count = 0

    if first_sentence in three_gram_freq_dict:
        if third_syllable in three_gram_freq_dict[first_sentence]:
            count_full_sentence = three_gram_freq_dict[first_sentence][third_syllable]


    if first_syllable in two_gram_freq_dict:
        if second_syllable in two_gram_freq_dict[first_syllable]:
            first_sentence_count = two_gram_freq_dict[first_syllable][second_syllable]


    return (count_full_sentence+1)/(len(one_gram_freq_dict)+(first_sentence_count))
```

3-Gram Perplexity Drive Script



```
print(calculate_perplexity_one_gram("kulagin gercekten de tikaliymis"))
print(calculate_perplexity_one_gram("nazara gelir butun kelimeler"))
print(calculate_perplexity_one_gram("her gece son gaz"))
print(calculate_perplexity_one_gram("hutuasdefdat d fasdfayfuas wq eadsufds sodif"))
```

[452] ✓ 0.9s

```
170.72181537061957
120.90442201820511
122.81610882456455
289.09089816455617
```

D)Discussion about Perplexities given sentences

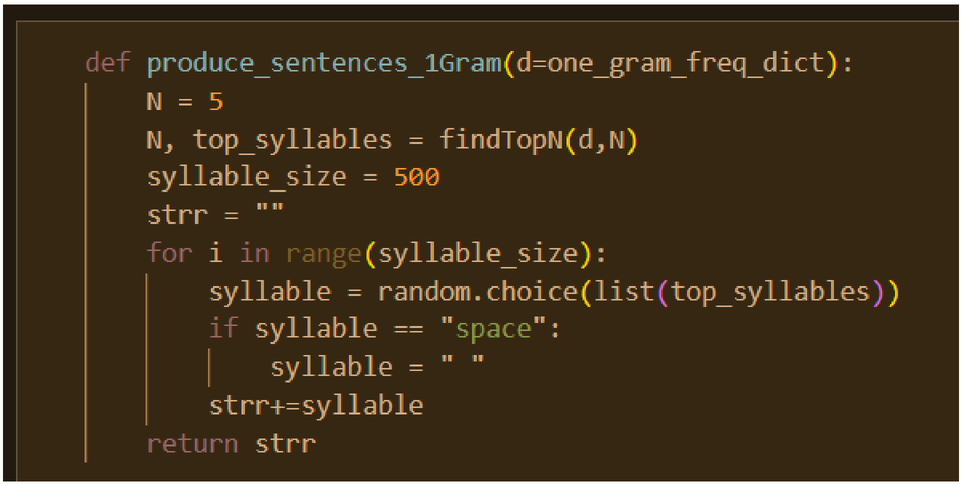
In general, perplexity is a measurement of how well a probability model predicts a sample. In the context of Natural Language Processing, perplexity is one way to evaluate language models.
Source From towardsdatascience

So, low perplexity is high probability. Every case (1,2,3-Gram) last sentences perplexity is higher than others.

5-Produce random sentences for each N-Gram model.

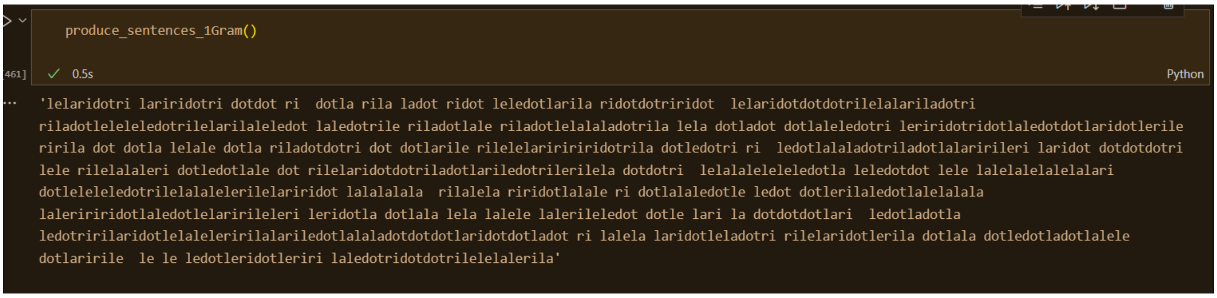
A)1-Gram

Produce 1-Gram Python Script



```python
def produce_sentences_1Gram(d=one_gram_freq_dict):
    N = 5
    N, top_syllables = findTopN(d,N)
    syllable_size = 500
    strr = ""
    for i in range(syllable_size):
        syllable = random.choice(list(top_syllables))
        if syllable == "space":
            syllable = " "
        strr+=syllable
    return strr
```

Produce 1-Gram function output

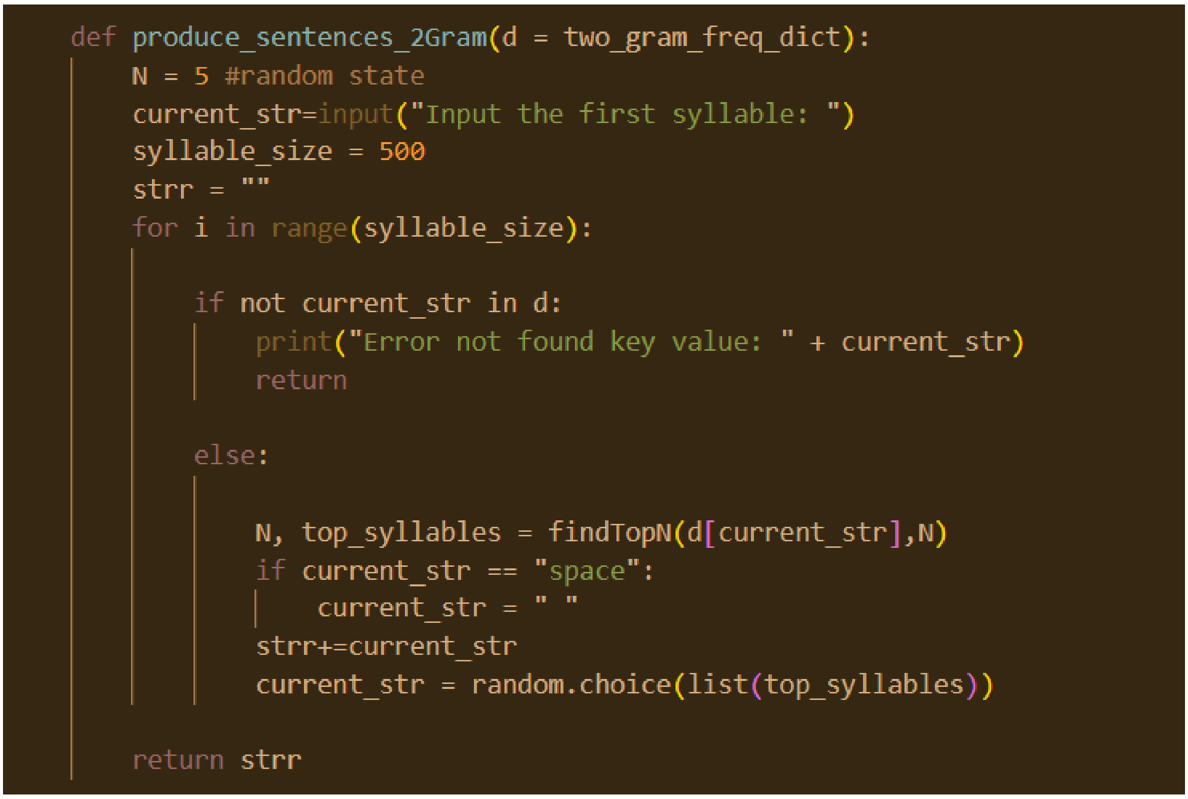


```python
produce_sentences_1Gram()
```

```
'lelaridotri lariridotri dotdot ri  dotla rila ladot ridot leledotlarila ridotdotriridot  lelaridotdotdotrilelalariladotri
riladotleleleledotrilelarilaleledot laledotrile riladotlale riladotlelalaladotrila lela dotladot dotlaleledotri leriridotridotlaledotdotlaridotlerile
ririla dot dotla lelale dotla riladotdotri dot dotlarile rilelelariririridotrila dotledotri ri  ledotlalaladotriladotlalaririleri laridot dotdotdotri
lele rilelalaleri dotledotlale dot rilelaridotdotriladotlariledotrilerilela dotdotri  lelalaleleleledotla leledotdot lele lalelalelalelalari
dotleleleledotrilelalalelerilelariridot lalalalala  rilalela riridotlalale ri dotlalaledotle ledot dotlerilaledotlalelalala
laleririridotlaledotlelaririleleri leridotla dotlala lela lalele lalerileledot dotle lari la dotdotdotlari  ledotladotla
ledotririlaridotlelaleleririlalariledotlalaladotdotdotlaridotdotladot ri lalela laridotleladotri rilelaridotlerila dotlala dotledotladotlalele
dotlaririle  le le ledotleridotleriri laledotridotdotrilelelalerila'
```

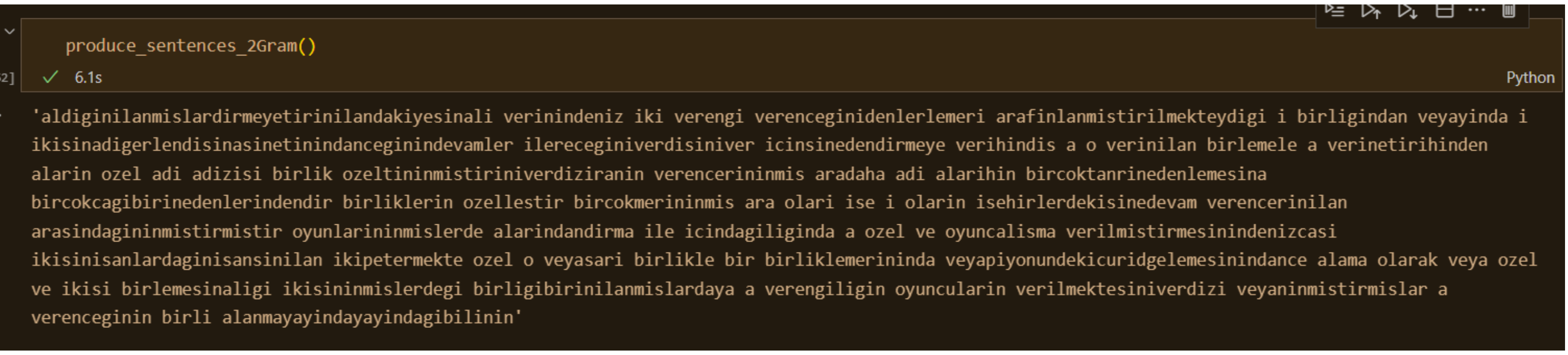
B)2-Gram

Produce 2-Gram Python Script



```python
def produce_sentences_2Gram(d = two_gram_freq_dict):
    N = 5 #random state
    current_str=input("Input the first syllable: ")
    syllable_size = 500
    strr = ""
    for i in range(syllable_size):

        if not current_str in d:
            print("Error not found key value: " + current_str)
            return

        else:

            N, top_syllables = findTopN(d[current_str],N)
            if current_str == "space":
                current_str = " "
            strr+=current_str
            current_str = random.choice(list(top_syllables))

    return strr
```

Produce 2-Gram function output

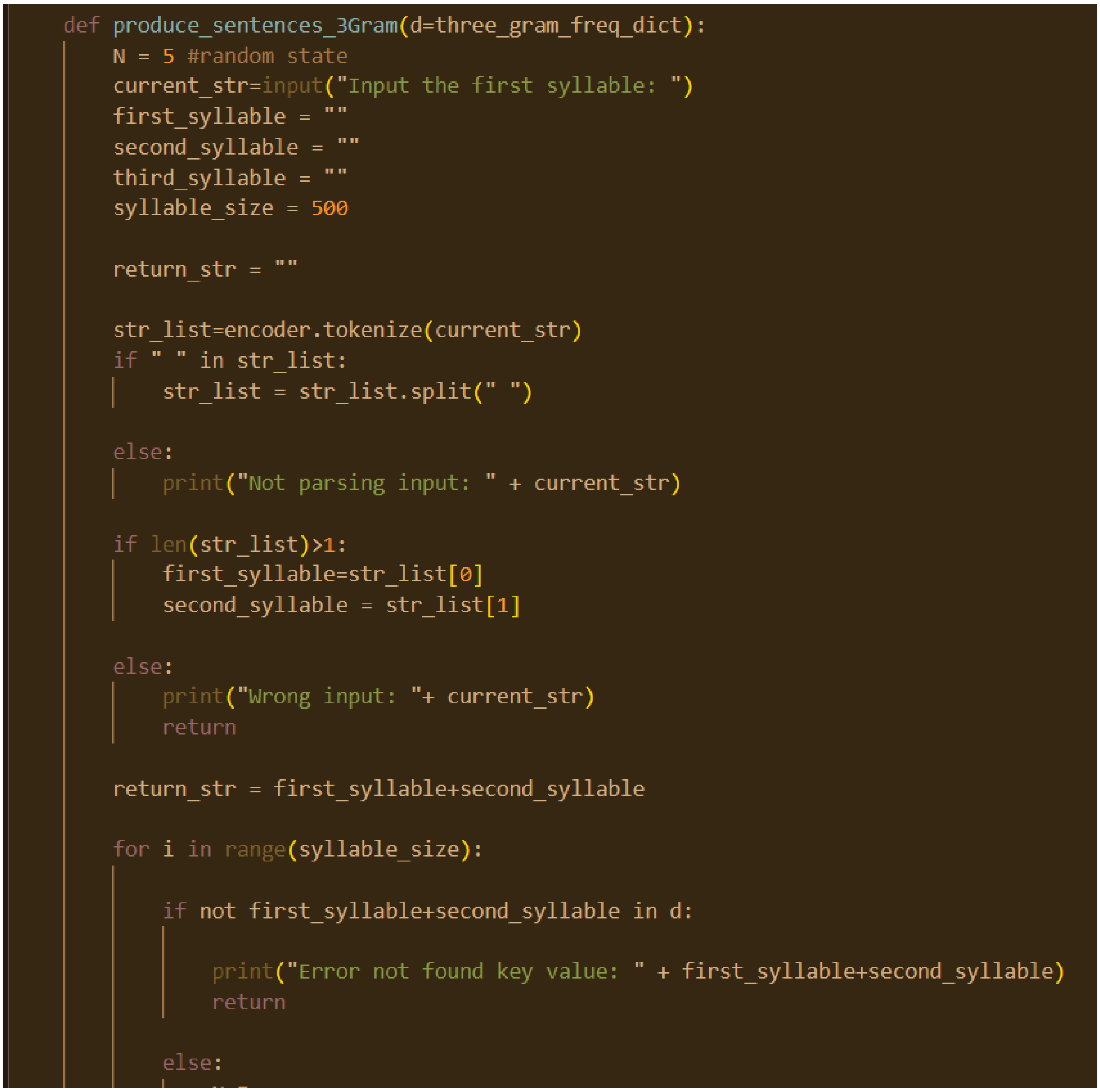


```
produce_sentences_2Gram()
✓ 6.1s
```

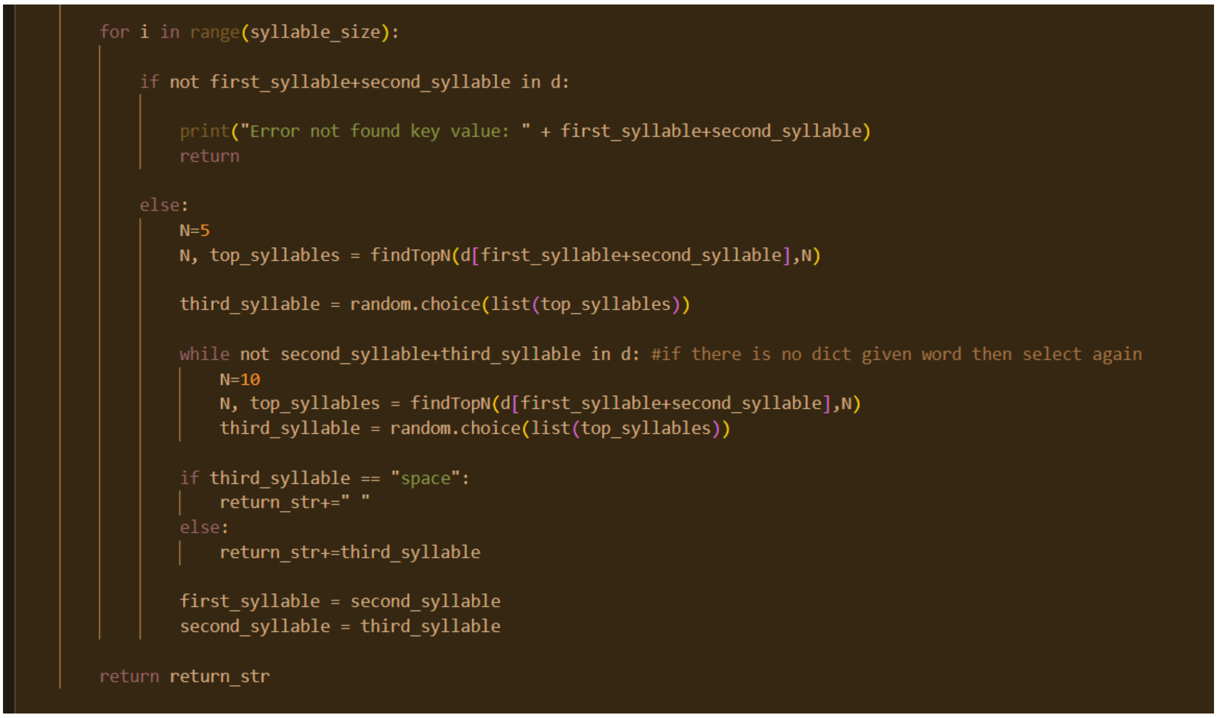
```
'aldiginilanmislardirmeyetirinilandakiyesinali verinindeniz iki verengi verenceginidenlerlemeri arafinlanmistirilmekteydigi i birligindan veyayinda i
ikisinadigerlendisinasinetinindanceginindevamler ilereceginiverdisiniver icinsinedendirmeye verihindis a o verinilan birlemele a verinetirihinden
alarin ozel adi adizisi birlik ozeltininmistiriniverdiziranin verencerininmis aradaha adi alarihin bircoktanrinedenlemesina
bircokcagibirinedenlerindendir birliklerin ozellestir bircokmerininmis ara olari ise i olarin isehirlerdekisinedevam verencerinilan
arasindagininmistirmistir oyunlarininmislerde alarindandirma ile icindagiliginda a ozel ve oyuncalisma verilmistirmesinindenizcasi
ikisinisanlardaginisansinilan ikipetermekte ozel o veyasari birlikle bir birliklemerininda veyapiyonundekicuridgelemesinindance alama olarak veya ozel
ve ikisi birlemesinaligi ikisininmislerdegi birligibirinilanmislardaya a verengiligin oyuncularin verilmektesiniverdizi veyaninmistirmislar a
verenceginin birli alanmayayindayayindagibilinin'
```

C)3-Gram

Produce 3-Gram Python Script

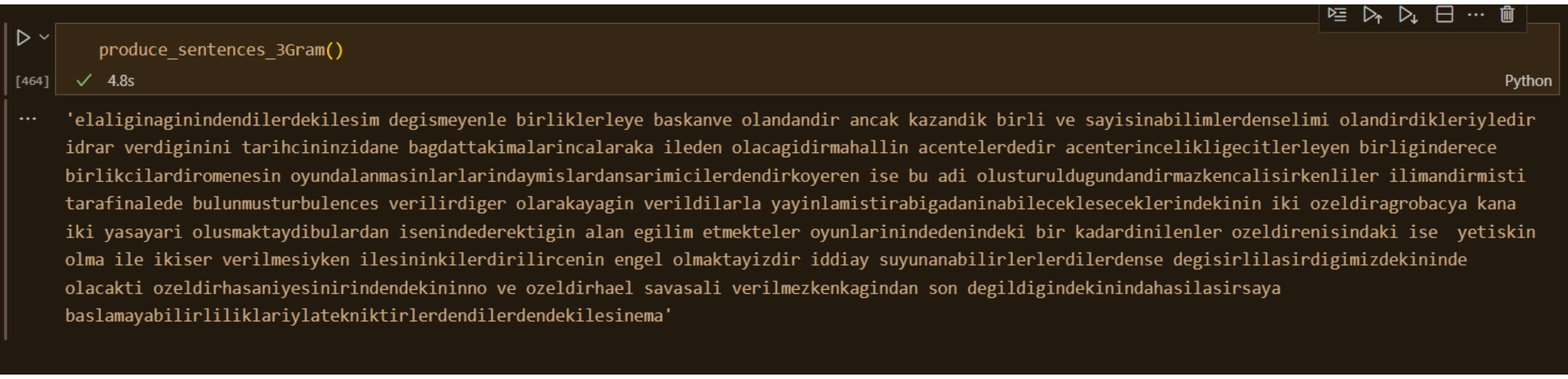


```
def produce_sentences_3Gram(d=three_gram_freq_dict):
    N = 5 #random state
    current_str=input("Input the first syllable: ")
    first_syllable = ""
    second_syllable = ""
    third_syllable = ""
    syllable_size = 500

    return_str = ""

    str_list=encoder.tokenize(current_str)
    if " " in str_list:
        str_list = str_list.split(" ")

    else:
        print("Not parsing input: " + current_str)

    if len(str_list)>1:
        first_syllable=str_list[0]
        second_syllable = str_list[1]

    else:
        print("Wrong input: "+ current_str)
        return

    return_str = first_syllable+second_syllable

    for i in range(syllable_size):

        if not first_syllable+second_syllable in d:

            print("Error not found key value: " + first_syllable+second_syllable)
            return

        else:
```



```python
    for i in range(syllable_size):

        if not first_syllable+second_syllable in d:

            print("Error not found key value: " + first_syllable+second_syllable)
            return

        else:
            N=5
            N, top_syllables = findTopN(d[first_syllable+second_syllable],N)

            third_syllable = random.choice(list(top_syllables))

            while not second_syllable+third_syllable in d: #if there is no dict given word then select again
                N=10
                N, top_syllables = findTopN(d[first_syllable+second_syllable],N)
                third_syllable = random.choice(list(top_syllables))

            if third_syllable == "space":
                return_str+=" "
            else:
                return_str+=third_syllable

            first_syllable = second_syllable
            second_syllable = third_syllable

    return return_str
```

Produce 3-Gram function output



```python
produce_sentences_3Gram()
```

```
'elaliginaginindendilerdekilesim degismeyenle birliklerleye baskanve olandandir ancak kazandik birli ve sayisinabilimlerdenselimi olandirdikleriyledir
idrar verdiginini tarihcininzidane bagdattakimalarincalaraka ileden olacagidirmahallin acentelerdedir acenterincelikligecitlerleyen birliginderece
birlikcilardiromenesin oyundalanmasinlarlarindaymislardansarimicilerdendirkoyeren ise bu adi olusturuldugundandirmazkencalisirkenliler ilimandirmisti
tarafinalede bulunmusturbulences verilirdiger olarakayagin verildilarla yayinlamistirabigadaninabilecekleseceklerindekinin iki ozeldiragrobacya kana
iki yasayari olusmaktaydibulardan isenindederektigin alan egilim etmekteler oyunlarinindedenindeki bir kadardinilenler ozeldirenisindaki ise  yetiskin
olma ile ikiser verilmesiyken ilesininkilerdirilircenin engel olmaktayizdir iddiay suyunanabilirlerlerdilerdense degisirlilasirdigimizdekininde
olacakti ozeldirhasaniyesinirindendekininno ve ozeldirhael savasali verilmezkenkagindan son degildigindekinindahasilasirsaya
baslamayabilirliliklariylatekniktirlerdendilerdendekilesinema'
```

D)Discussion about produced sentences

1-Gram scored random results, in fact, it took the 5 most repetitive ones in 1-Gram and gave a string output among them. 2-Grams gave much more distinguishable results than 1-Gram because it gave statistical next word prediction based on the previous word.

Finally, 3-Gram gave more satisfactory results than 2 grams, which may be due to the high number of 3 syllables in a word in Turkish Language.

Resources

1- https://www.youtube.com/watch?v=KuyzNuOlmBY&ab_channel=YusufSinanAkgul (Lecture Video)

2- Stacoverflow

3- Wikipedia

4- https://github.com/ftkurt/python-syllable in this homework used syllable library's github repos

5- https://towardsdatascience.com/perplexity-intuition-and-derivation-105dd481c8f3 ()

6- https://www.youtube.com/watch?v=PiPqqd2b25o&t=332s&ab_channel=DesiSkill (GT smoothing example)