Data Analytics

Assignment -Vector Space Model-

윤장혁 교수님

산업공학과

201811527

이영은

Week7

주어진 세 개의 document(data_week7.txt)를 세가지 방법의 tf based VSM으로 표현하는 과정을 다음과 같은 과정으로 수행하였습니다.

1) Konlpy.tag의 Twitter 클래스를 사용한 형태소 분석
2) **tf-based VSM representation (boolean)**
3) **tf-based VSM representation (simple)**
4) **tf-based VSM representation (logarithmatically scaled)**
5) 2)~4)의 과정을 엑셀 파일로 저장

■ 형태소 분석

- 형태소는 최소 의미 단위를 말합니다.
- 형태소 분석이란 언어 단위인 어절 또는 문장을 형태소로 나누어 가는 과정입니다.

  data_week7.txt 파일을 보면 긴 문장 1개가 하나의 document로 되어 있습니다.

  따라서 저는 형태소로 분절을 해야 stop-word와 의미있는 단어로 나눌 수 있다는 판단을 하였고, 각각의 document를 형태소로 분절하였습니다.

  예를 들어 "저는 어제 과제를 하느라 집에 늦게 갔습니다." 라는 문장을 형태소로 나누면 다음과 같습니다.

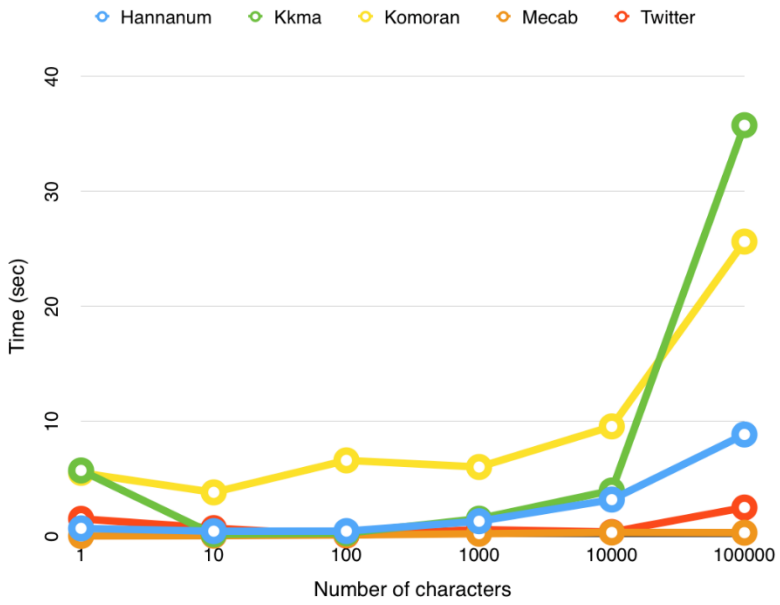
  저(대명사)+는(보조사) 어제(부사) 과제(명사)를(격조사) 하다(동사)+느라(연결어미) 집(명사)+에(격조사) 늦게(형용사) 가다(동사)+았(선어말어미)+습니다(어말어미)

  위의 예시와 같이 하나의 어절에 대해서도 여러 개의 분석 결과가 있을 수 있고, 이것을 제 기준으로는 일관적으로 판단하는 것이 어렵다는 생각이 들어 간단한 형태소 분석 라이브러리를 사용하였습니다.

- 저는 konlpy.tag를 사용하여 형태소 분석을 하였고, 이중 konlpy.tag 모듈이 제공하는 Twitter 클래스를 이용하였습니다.
- konlpy에는 Kkma, Komoran, Hannanum, Twitter, Mecab의 클래스가 있고, 제가

Twitter클래스를 사용하게 된 이유는 다음과 같습니다.

- konlpy 문서에 따르면 클래스 간의 성능은 다음과 같습니다.
- 로딩 시간: 사전 로딩을 포함하여 클래스를 로딩하는 시간.
  - `Kkma`: 5.6988 *secs*
  - `Komoran`: 5.4866 *secs*
  - `Hannanum`: 0.6591 *secs*
  - `Twitter`: 1.4870 *secs*
  - `Mecab`: 0.0007 *secs*
- 실행시간: 10 만 문자의 문서를 대상으로 각 클래스의 `pos` 메소드를 실행하는데 소요되는 시간.
  - `Kkma`: 35.7163 *secs*
  - `Komoran`: 25.6008 *secs*
  - `Hannanum`: 8.8251 *secs*
  - `Twitter`: 2.4714 *secs*
  - `Mecab`: 0.2838 *secs*



(출처 – http://konlpy.org/ko/v0.4.3/morph/)

- Mecab 클래스가 성능이 가장 우수하고, 그 다음 Twitter의 성능이 우수함을 알 수 있습니다.

Mecab의 경우는 빌드 과정을 거쳐야만 사용할 수 있으므로 이러한 과정들이 번거롭기 때문에 Twitter 클래스를 가지고 분석하였습니다.

- 저는 Twitter 클래스가 분석해준 tag, word결과 중에서 'Noun', 'Adjective', 'Verb' 를 tag로 갖는 word들이 의미를 가진다고 생각하였고, 이들을 가지고 분석을 진행하였습니다.

■ data_week7.txt의 형태소 분류(Python)

- txt파일을 읽고, 이를 형태소 별로 분류하기 위하여 python을 사용하였고, 한글 형태소를 분류 해주는 konlpy.tag의 Twitter 클래스를 사용하였습니다.
- 사용한 코드는 다음과 같습니다.

◆ 코드와 코드설명

코드	코드 설명
`from collections import Counter` `from konlpy.tag import Twitter` `import pandas as pd` `import math` `f = open(r"C:\Users\zdudd\Desktop\DA\20\week7\data_week7.txt", encoding = "utf-8")` `lines = f.readlines()`	- 단어가 한 document내에서 몇번 쓰였는지 알아볼 때 Counter를 사용하기 위하여 Counter를 import하였습니다. - 형태소 분석을 하기 위하여 Twitter를 import하였습니다. - DataFrame을 이용하기 위하여 pandas를 import하였습니다. - log 함수를 사용하기 위하여 math를 import 하였습니다. - week7 과제로 제공받은 data_week7.txt 파일을 오픈하였고, 한줄 씩 읽어서 lines에 리스트 형태로 저장하였습니다.
`doc1 = []` `doc1.append(lines[0])` `doc2 = []` `doc2.append(lines[2])`	- 각각의 document를 doc1, doc2, doc3에 저장하였습니다.

doc3 = [] doc3.append(lines[4])	
twitter = Twitter() def whole_word(doc): sentences_tag = [] for sentence in doc: morph = twitter.pos(sentence) sentences_tag.append(morph) term_corpus = [] for sentence in sentences_tag: for word, tag in sentence: if tag in['Noun', 'Adjective ', 'Verb']: term_corpus.append(word) return term_corpus	- whole_word 함수를 정의하였습니다. - 이 함수의 기능은 다음과 같습니다. document의 단어들의 형태소를 분류하여, sentences_tag에 append하고, 이 형태소들이 명사, 형용사, 동사의 tag를 갖을 때만 term_corpus리스트에 단어를 append합니다.
def word_counter(doc): sentences_tag = [] for sentence in doc: morph = twitter.pos(sentence) sentences_tag.append(morph) term_corpus = [] for sentence in sentences_tag: for word, tag in sentence: if tag in['Noun', 'Adjective ', 'Verb']: term_corpus.append(word) counts = Counter(term_corpus) return counts whole = list(set(whole_word(lines)))	- word_counter 함수를 정의하였습니다. - 이 함수의 기능은 다음과 같습니다. - whole_word와 같은 방식으로 형태소를 분류하고, 동사, 명사, 형용사 tag를 갖는 단어만 저장 한후 이들 단어의 개수를 counter를 통하여 dict형태로 반환합니다. - whole은 전체 document에 쓰인 단어중 명사, 형용사, 동사 태그를 가진 단어를 whole_word함수를 통하여 찾아내고, set, list과정을 통하여 중복을 제거해서 다시 list형태로 반환합니다.

```
#Boolean
df1 = pd.DataFrame(index = ['doc1', 'doc
          2', 'doc3'], columns = whole)
for i in whole:
    if i in whole_word(doc1):
        df1[i]['doc1'] = 1
    else:
        df1[i]['doc1'] = 0

for i in whole:
    if i in whole_word(doc2):
        df1[i]['doc2'] = 1
    else:
        df1[i]['doc2'] = 0

for i in whole:
    if i in whole_word(doc3):
        df1[i]['doc3'] = 1
    else:
        df1[i]['doc3'] = 0
df1
df1.to_excel('Boolean.xlsx')
#Boolean end
```

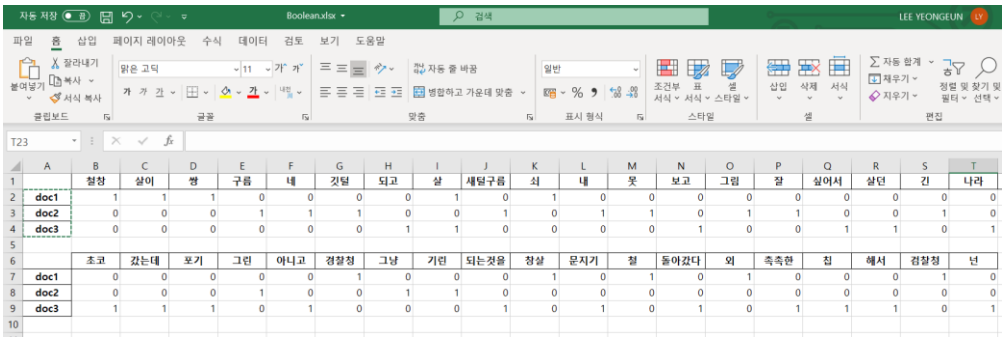
- Boolean
- df1이라는 이름을 가진 dataframe을 정의하고, 행이름을 doc1-doc3으로, 열 이름을 위에서 구한 whole로 정의하였습니다.
- 그 이후에 document1,2,3에 대하여 각각 for문을 돌렸습니다.
- 만약 whole_word(doc)에 열이름과 같은 단어가 있다면 1을, 없다면 0을 dataframe에 입력하였습니다.
- 이를 excel파일로 저장하였습니다.
따로 저장경로를 설정하지 않았기 때문에, 파이썬 실행파일과 같은 폴더에 저장되었습니다.

```
#Simple
df2 = pd.DataFrame(index = ['doc1', 'doc
          2', 'doc3'], columns = whole)
for i in whole:
    if i in word_counter(doc1):
        df2[i]['doc1'] = word_counter(do
                              c1).get(i)
    else:
        df2[i]['doc1'] = 0

for i in whole:
    if i in word_counter(doc2):
        df2[i]['doc2'] = word_counter(do
                              c2).get(i)
    else:
        df2[i]['doc2'] = 0

for i in whole:
    if i in word_counter(doc3):
        df2[i]['doc3'] = word_counter(do
                              c3).get(i)
```

- Simple
- df2이라는 이름을 가진 dataframe을 정의하고, 행이름을 doc1-doc3으로, 열 이름을 위에서 구한 whole로 정의하였습니다.
- 그 이후에 document1,2,3에 대하여 각각 for문을 돌렸습니다.
- 만약 whole_counter(doc)의 dict의 key중 열이름과 같은 단어가 있다면 그 key의 value값을, 없다면 0을 dataframe에 입력하였습니다.
- 이를 excel파일로 저장하였습니다.
따로 저장경로를 설정하지 않

`` else: df2[i]['doc3'] = 0 df2 df2.to_excel('Simple.xlsx') #Simple end ```	았기 때문에, 파이썬 실행파일과 같은 폴더에 저장되었습니다.
`` #log df3 = pd.DataFrame(index = ['doc1', 'doc2', 'doc3'], columns = whole) for i in whole: for j in ['doc1', 'doc2', 'doc3']: if df2[i][j] == 0: df3[i][j] = 0 else: df3[i][j] = math.log10(1+df2[i][j]) df3 df3.to_excel('Log_scaled.xlsx') ```	- log - df3이라는 이름을 가진 dataframe을 정의하고, 행이름을 doc1-doc3으로, 열 이름을 위에서 구한 whole로 정의하였습니다. - 그 이후에 위에서 정의한 df2의 dataframe값을 가지고, 그 값이 0이 아니라면 log(1+tf)를 갖도록 설정하였습니다. - 이를 excel파일로 저장하였습니다. 따로 저장경로를 설정하지 않았기 때문에, 파이썬 실행파일과 같은 폴더에 저장되었습니다.

◆ **tf-based VSM representation (boolean) 결과**

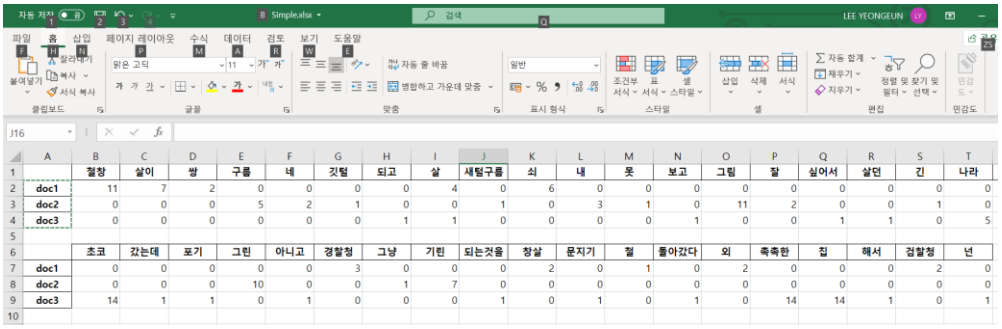


	철창	살이	쌍	구름	네	깃털	되고	살	새털구름	쇠	내	못	보고	그림	잘	싶어서	살던	긴	나라
doc1	1	1	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
doc2	0	0	0	1	1	1	0	0	1	0	1	1	0	1	1	0	0	1	0
doc3	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	1	1	0	1

	초코	갔는데	포기	그린	아니고	경찰청	그냥	기린	되는것을	창살	문지기	철	돌아갔다	외	촉촉한	칩	해서	검찰청	넌
doc1	0	0	0	0	0	1	0	0	0	1	0	1	0	1	0	0	0	1	0
doc2	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
doc3	1	1	1	0	1	0	0	0	1	0	1	0	1	0	1	1	1	0	1

- 한 화면에 담기 위해서, (초코 ~ 넌) 까지의 열을 임의로 밑으로 내린 후, 첨부하였습니다.

◆ **tf-based VSM representation (simple) 결과**



	철창	살이	쌍	구름	네	깃털	되고	살	새털구름	쇠	내	못	보고	그림	잘	싶어서	살던	긴	나라
doc1	11	7	2	0	0	0	0	4	0	6	0	0	0	0	0	0	0	0	0
doc2	0	0	0	5	2	1	0	0	1	0	3	1	0	11	2	0	0	1	0
doc3	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	1	1	0	5

	초코	갔는데	포기	그린	아니고	경찰청	그냥	기린	되는것을	창살	문지기	철	돌아갔다	외	촉촉한	칩	해서	검찰청	넌
doc1	0	0	0	0	0	3	0	0	0	2	0	1	0	2	0	0	0	2	0
doc2	0	0	0	10	0	0	1	7	0	0	0	0	0	0	0	0	0	0	0
doc3	14	1	1	0	1	0	0	0	1	0	1	0	1	0	14	14	1	0	1

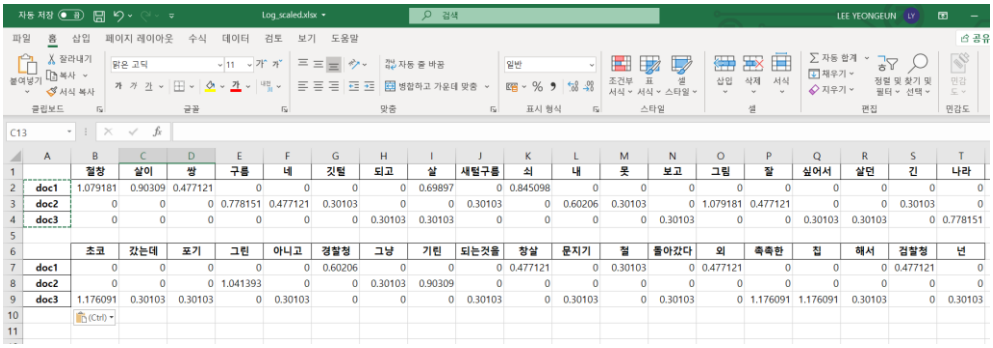
- 한 화면에 담기 위해서, (초코 ~ 넌) 까지의 열을 임의로 밑으로 내린 후, 첨부하였습니다

- doc1에서는 철창 > 살이> 살 순으로 단어가 많이 나왔습니다.

- doc2에서는 그림 > 그린 > 기린 순으로 단어가 많이 나왔습니다.

- doc3에서는 칩, 촉촉한, 초코 단어가 많이 나왔고, 나머지 단어는 많이 나오지 않았습니다.

◆ **tf-based VSM representation (logarithmatically scaled) 결과**



		철창	살이	방	구름	네	깃털	되고	살	새털구름	쇠	내	못	보고	그림	잘	싶어서	살던	긴	나라
2	doc1	1.079181	0.90309	0.477121	0	0	0	0	0.69897	0	0.845098	0	0	0	0	0	0	0	0	0
3	doc2	0	0	0	0.778151	0.477121	0.30103	0	0	0.30103	0	0.60206	0.30103	0	1.079181	0.477121	0	0	0.30103	0
4	doc3	0	0	0	0	0	0	0.30103	0.30103	0	0	0	0	0.30103	0	0	0.30103	0.30103	0	0.778151

		초코	갔는데	포기	그린	아니고	경찰청	그냥	기린	되는것을	창살	문지기	철	돌아갔다	외	촉촉한	집	해서	검찰청	넌
7	doc1	0	0	0	0	0	0.60206	0	0	0	0.477121	0	0.30103	0	0.477121	0	0	0	0.477121	0
8	doc2	0	0	0	1.041393	0	0	0.30103	0.90309	0	0	0	0	0	0	0	0	0	0	0
9	doc3	1.176091	0.30103	0.30103	0	0.30103	0	0	0	0.30103	0	0.30103	0	0.30103	0	1.176091	1.176091	0.30103	0	0.30103

- 10회가 넘을 경우, 1이 넘는 숫자를 가짐을 알 수 있고, doc3에서 가장 빈도가 많은 단어의 출현이 있음을 알 수 있습니다.

◆ 한계점

- 이번 과제를 진행하면서 document를 어떻게 나누어야 하는지에 대한 고민을 많이 하였습니다.

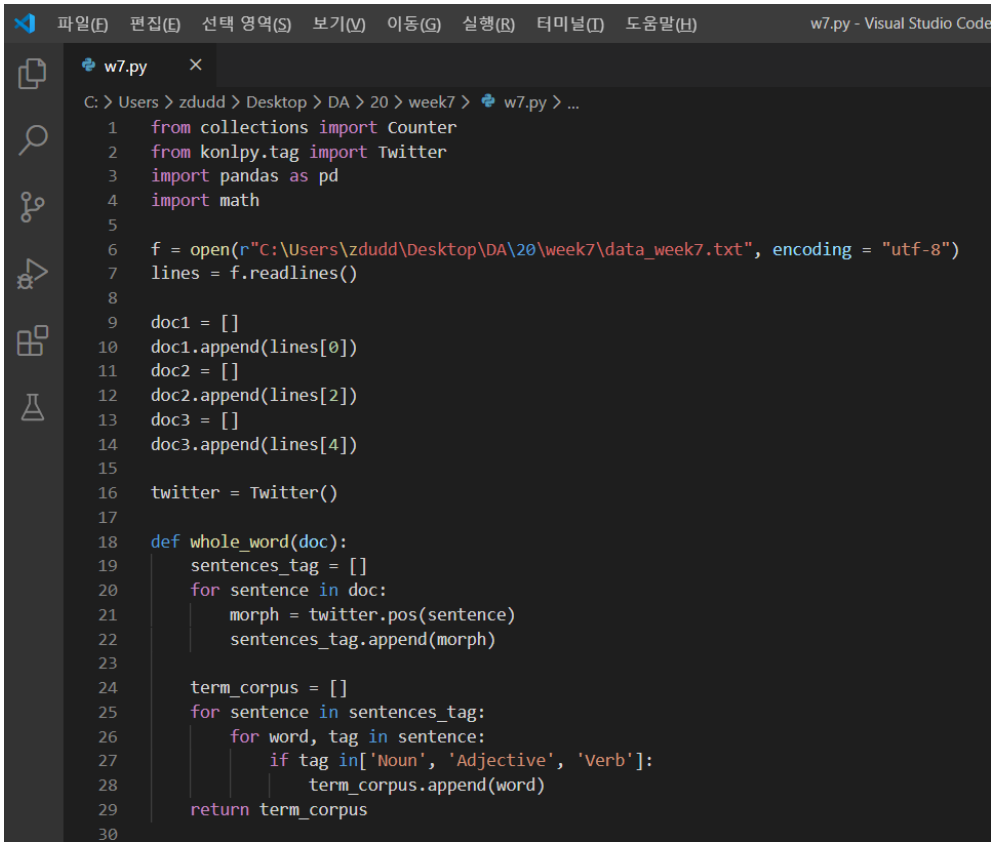
  영어를 분류를 했다면 좀 더 stop-word의 분류가 쉬웠을 것이라는 생각을 했고, 한국어의 특성상 어미가 붙거나, 접두사, 접미사가 많아서 단어의 분류에 어려움을 겪었습니다.

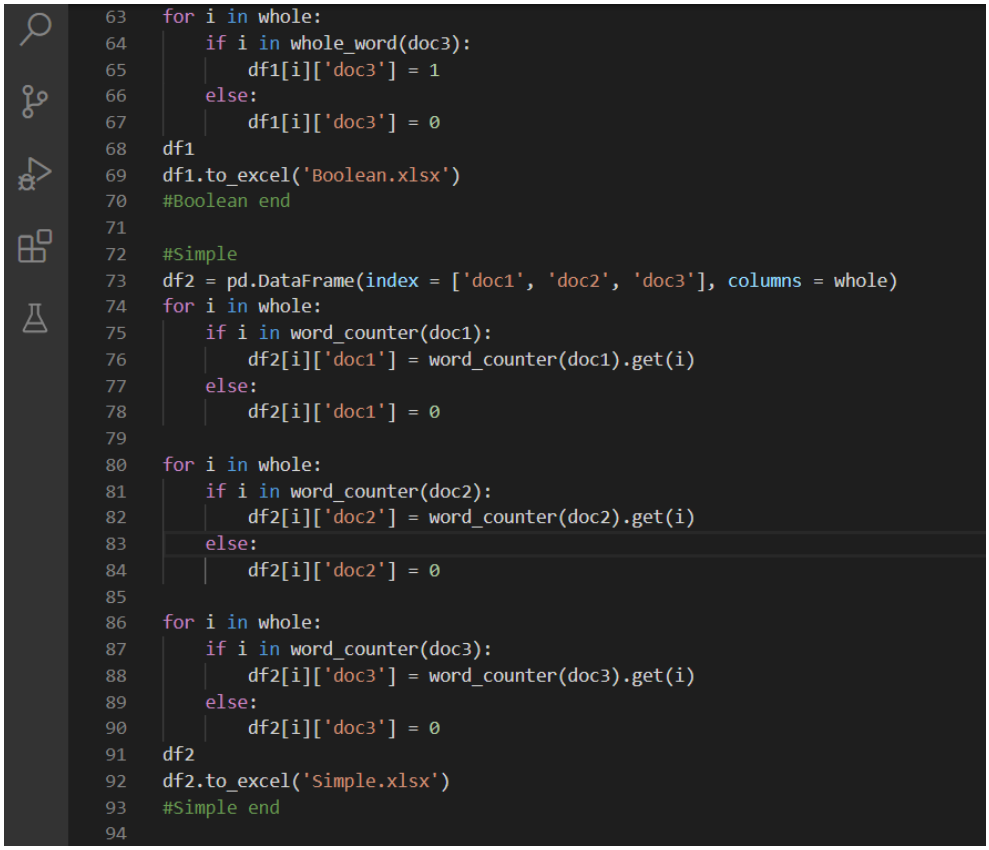
  konlpy의 Twitter 클래스를 사용했지만, '철창살이' 에서의 분류를 창살과 철로 분류하지 못하고, '철창'과 '살이' 로 분류한 점 등은 한계점이라고 할 수 있습니다.

- 조금 더 일관성이 있는 분류 기준을 통하여 단어들을 분류하고, 분석 과정을 수행하였다면 더 유의미한 결과가 있었을 것 이라는 아쉬움을 가졌습니다.

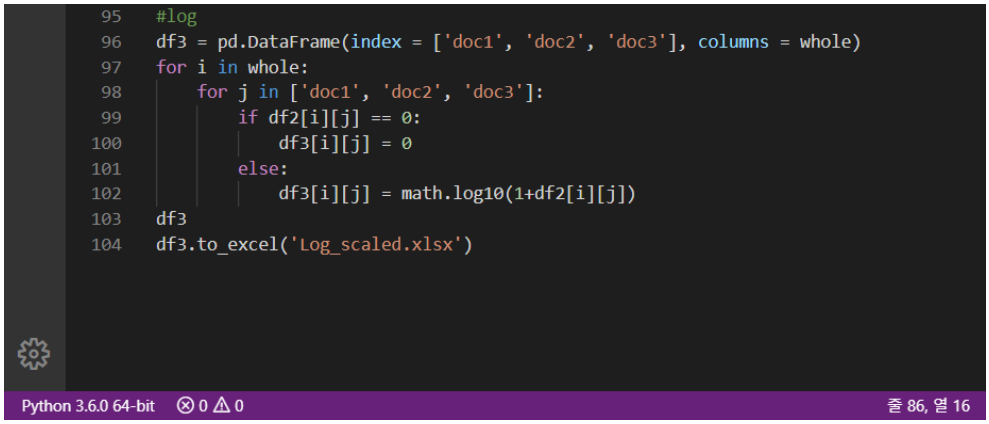
◆ 전체코드

- 사용된 전체 코드는 다음과 같습니다.





```
for i in whole:
    if i in whole_word(doc3):
        df1[i]['doc3'] = 1
    else:
        df1[i]['doc3'] = 0
df1
df1.to_excel('Boolean.xlsx')
#Boolean end

#Simple
df2 = pd.DataFrame(index = ['doc1', 'doc2', 'doc3'], columns = whole)
for i in whole:
    if i in word_counter(doc1):
        df2[i]['doc1'] = word_counter(doc1).get(i)
    else:
        df2[i]['doc1'] = 0

for i in whole:
    if i in word_counter(doc2):
        df2[i]['doc2'] = word_counter(doc2).get(i)
    else:
        df2[i]['doc2'] = 0

for i in whole:
    if i in word_counter(doc3):
        df2[i]['doc3'] = word_counter(doc3).get(i)
    else:
        df2[i]['doc3'] = 0
df2
df2.to_excel('Simple.xlsx')
#Simple end

```



```
#log
df3 = pd.DataFrame(index = ['doc1', 'doc2', 'doc3'], columns = whole)
for i in whole:
    for j in ['doc1', 'doc2', 'doc3']:
        if df2[i][j] == 0:
            df3[i][j] = 0
        else:
            df3[i][j] = math.log10(1+df2[i][j])
df3
df3.to_excel('Log_scaled.xlsx')
```

Python 3.6.0 64-bit ⊗ 0 ⚠ 0 줄 86, 열 16