국민대 및 정릉시장 주변 맛집 분석

20182786 강민수

1. 연구배경

처음 이 연구를 생각을 한 것은 네이버 블로그 리뷰 빈도분석으로도 주위 맛집을 알 수 있지 않을까?라는 생각에서 시작되었다.
정릉시장에서 자취를 하면서 주위 맛집은 많이 가보았지만 아직 못가본 맛집이 많을 텐데 그러한 맛집을 찾고 싶기도 했고 “실생활에 접목시켜서 분석할 수 있지 않을까?”라는 생각에서 시작되었다.

2. 관련연구

1. 데이터 ddd 선택 :

1) VIEW 검색 (정릉시장맛집,정릉맛집,국민대맛집) 제목 및 미리보기 크롤링 데이터
2) 서울특별시 공공데이터 : “서울시 성북구 일반음식점 인허가 정보”
3) “다이닝코드” : https://www.diningcode.com/ 크롤링 데이터
4) “망고플레이트” : https://www.mangoplate.com/ 크롤링 데이터
ps. 다이닝코드, 망고플레이트는 유명한 맛집 추천사이트이며 정릉시장 주위 맛집 위주로 크롤링하였음

1) VIEW 검색 데이터를 선택한 이유

VIEW 검색은 실제로 방문한 사람들의 후기들이 나와 있으며 후기가 많다는 것은 ‘리뷰이벤트’를 해서 엄청나게 많은 경우가 아니면 사람들이 많이 방문한 음식점들이 대체로 후기가 많다. 이 점을 착안해서 VIEW 검색에서 많은 빈도로 노출되는 음식점이 맛집일 가능성이 높다고 판단하여 VIEW 검색 데이터를 선택하였다. 또한 연관단어 ex)정릉맛집, 정릉시장맛집, 국민대맛집 등등을 추가로 크롤링을 하여서 데이터를 확보 및 정확성을 높였다.
VIEW 검색은 “블로그+카페+포스트+주제별리뷰”가 나와있는데 여기서 처음에 VIEW 검색을 하면 제목, 미리보기, 사진이 뜬다. 여기서 제목 및 미리보기 식으로 제공하는 내용 3 줄만 크롤링 하였는데 이유는 블로거들의 대부분이 핵심 단어 근처에 원하는 정보를 심어놓기 때문이다.

VIEW 검색이 생기기 이전 네이버 블로그 리뷰를 크롤링하면 우리 동네 맛집이라고 검색하면 내가 뻔히 가본 곳인데도 (맛집이 아닌데도) 상위 목록에 맛집이라고 도배가 되어 있는 것을 흔하게 볼 수 있다. 그런 결과가 검색 품질을 떨어뜨리고 정보에 선택에 있어서 나쁜 영향을 미치는데 VIEW 검색의 경우는 그러한 문제점들이 개선되어 "블로그+카페+포스트+주제별리뷰" 검색결과가 통합되어 노출되기 때문에 VIEW 검색한 데이터를 크롤링하기로 결정했다.

2) 공공데이터 : "서울시 성북구 일반음식점 인허가 정보"를 선택한 이유

VIEW 검색 크롤링 데이터만으로는 음식점 이름 노출 빈도가 다른 단어들에 비하여 상대적으로 적어 분석하기가 힘들었다.

분석의 정확도를 높이기 위하여 음식점 이름 노출 빈도를 다른 단어들에 비해 높여줄 필요가 있었는데 공공데이터를 찾다보니 성북구에 인허가된 모든 음식점 데이터가 있었다. 공공데이터에서 음식점 이름만 따와서 VIEW 검색 크롤링 데이터와 합쳐주니 음식점 노출 빈도수가 조금 올라갔다.

3) "다이닝코드", "망고플레이트" 크롤링 데이터 선택 이유

"다이닝코드", "망고플레이트"의 경우는 유명한 맛집 추천 사이트이다. 이곳 말고도 메뉴판 닷컴 등등 많았지만 실질적으로 이 두 사이트가 내가 근처 가본 맛집과 많이 일치하여서 이 두 사이트를 선택하였다.

두 사이트에서 데이터 크롤링은 정릉 검색후에 나오는 음식점 이름들을 모든 페이지 다 크롤링 하였다. 실질적으로 크롤링한 데이터를 보니 내가 자주 가는 맛집들이 많이 보여서 이 두 사이트에서 크롤링한 데이터에 가중치를 많이 실어주었다.

2. 크롤링 방법 선택

정적 크롤링 VS 동적 크롤링

	정적 크롤링	동적 크롤링
연속성	주소를 통해 단발적으로 접근	브라우저를 사용하여 연속적으로 접근
수집 능력	수집 데이터의 한계가 존재	수집 데이터의 한계가 없음
속도	빠름	느림
라이브러리	requests, BeautifulSoup	selenium, chromedriver

데이터 크롤링을 할 때 selenium 을 사용하는 동적 크롤링을 진행하였다.

이유는 음식점의 경우는 중간에 폐업하기도 하고 맛집은 계속 바뀌기 때문에 VIEW 검색 데이터는 검색하는 시점에 따라서 다를 수 있다.

정적 크롤링을 통하여 연구를 진행하면 추후에 비슷한 연구를 진행할 때

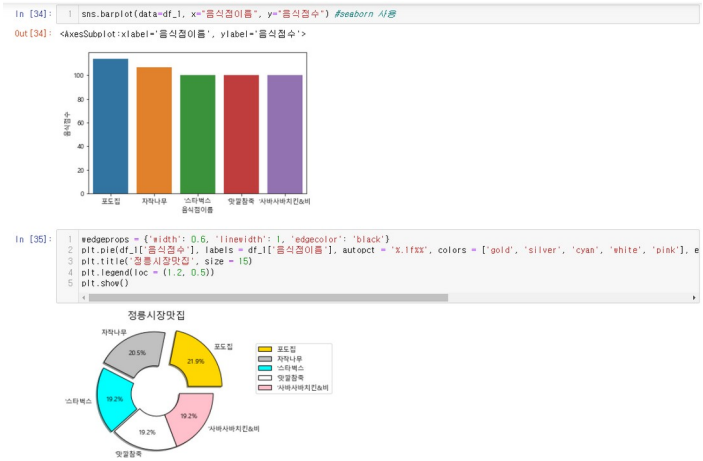
코드가 전혀 쓸모없어지기 때문에 동적 크롤링을 사용하는게 맞다고 판단하여 동적 크롤링으로 진행하였다.
또한 수업시간에 진행한 BeautifulSoup 말고도 동적 크롤링을 추가로 더 연구하여 발표시간에 동적 크롤링 사용 방법과 코드를 같은 학우들에게 공유를 하고 싶었다.

3. 형태소 분석기 설명

1) 코모란(Komoran) : 여러 어절을 하나의 품사로 분석 가능함으로써 형태소 분석기의 적용 분야에 따라 공백이 포함된 고유명사(영화 제목, 음식점명, 노래 제목, 전문 용어 등)를 더 정확하게 분석할 수 있다.
2) 한나눔(Hannanum) : 집합 총 69 개의 확장된 카이스트 태그셋을 기본으로 사용, 현재는 6 개 상위 태그에 대해서 20 개의 새로운 태그를 세분화하여 사용
3) Kkma(꼬고마) : 띄어쓰기 오류에 덜 민감한 한글 형태소 분석기
4) Mecab : 지능형 형태소 분석기(분석 결과를 수작업으로 수정가능), MACH, HAM, KKMA, UTagger 는 분석 사전이 분석기에 포함 되어 있어 새로운 단어 추가 불가능 → Mecab 이 이를 해결
5) Twitter : 어근화(stemming)이 가능하다. Kkma(꼬꼬마) 대비 분석 범주가 다소 적은 편이지만 이모티콘이나 해쉬태그 같은 인터넷 텍스트에 특화된 범주가 추가 되었다. 이로 인해 이모티콘이나 비표준어, 비속어 등이 많이 포함되어 있는 정제되지 않은 데이터에 대해서 강점을 갖는다.
6) Khaiii : 한국어 형태소 분석 결과는 원형 복원, 불규칙 활용 등의 이유로 입력 문자와는 형태와 길이가 달라짐.
Khaiii 같은 경우는 카카오에서 개발하였는데 신경망 알고리즘들 중 CNN 을 사용하는 최신의 딥러닝 방법이다. 하지만 C++로 개발했기 때문에 CentOS 7.4, Ubuntu 16.04, MacOS Mojave 이상의 컴파일러만을 지원한다고 합니다. 윈도우에서 설치할 수 있지만 상당히 복잡하다고 한다.

4. 시각화

시각화 같은 경우는 수업시간에 matplotlib 위주로 하고 seaborn 패키지는 사용을 많이 안한 것 같아서 seaborn 패키지로 시각화 하였다.



seaborn 은 matplotlib 보다 쉬운 문법을 가지고 있다. 사용하기에 더 편리하고 데이터 분석에 더 용이하다.

추가적으로 공부하고싶으면 아래 링크를 타고 들어가서 공부하면 된다.
링크 :
https://hyemin-kim.github.io/2020/07/03/S-Python-Seaborn1/

5. TF-IDF

1. 동시 발생 행렬
   - 특정 단어의 주변(맥락)에 어떤 단어가 몇번식 등장하는지 집계
   - 윈도우 크기가 1 인 경우 동시발생행렬(Co-Occurrence Matrix)

2. Vector Space Model(벡터공간모델)
- 텍스트 문서를 단어 색인 등의 식별자로 구성된 벡터로 표현하는 대수적 모델
  - Doc./Term Matrix
    - 단어의 문서 출현 빈도를 표현
    - 단어를 문서 벡터로, 문서를 단어 벡터로 해석 문서단어행렬
  - TF(t, d): Term Frequency
    - 문서 d 에서 용어 t 가 출현한 빈도
    - 문서 d 가 용어 t 를 포함하는지 여부 (0, 1)
    - Three Values (0, 1, 2)
    - Weighted Frequency

3. TF-IDF (Term Frequency-Inverse Document Frequency)
- 각 용어가 전체 문서에서 나타내는 출현 빈도의 역비율을 가중치로 반영
  - TF-IDF(d, t) = tf(d, t) * ln(N / df(t))

• tf(d, t): 문서 d 에서 용어 t 의 출현 빈도, N: 전체 문서 수, df(t): 전체 문서 중 용어 t 를 포함하는 문서 수

• 로그 밑은 임의의 수 가능 주로 자연로그 사용

- df(t) 대신 (df(t) +1)을 자주 사용

• 로그 내의 분모가 0 이 되는 것을 방지하기 위함

- TF-IDF 에 L2 정규화를 수행한 값을 자주 사용

• L2 Norm: 각 항의 제곱의 합이 1 이 되는 정규화

• 원소 / sqrt(문서 내 모든 원소의 제곱 합)

3. 제안 방법론

1. 형태소 분석기 사용

크롤링 및 전처리한 데이터를 Kkma, Twitter, Komoran, Hannanum 네가지 형태소 분석기로 분석해 보았다.

- Kkma

```
In [23]:
kkm = Kkma()
tokens_kkm = kkm.nouns(best_store_list)
tokens_kkm
kkm = nltk.Text(tokens_kkm)
kkm.tokens
kkm.vocab()
new_kkm=[]
for word in kkm:
  if len(word) > 1 and word != ' ':
        new_kkm.append(word)
new_kkm
kkm = nltk.Text(new_kkm)
kkm.tokens
kkm.vocab()
data_kkm = kkm.vocab().most_common(150)
data_kkm = dict(data_kkm)
data_kkm
```

```
Out[23]: {'분식': 3,
 '정릉점': 3,
 '정릉': 3,
 '종암': 2,
 '서울': 2,
 '장위': 2,
 '성북': 2,
 '월곡': 2,
 '제주': 2,
 '맛집': 2,
 '돈암': 2,
 '오리': 2,
 '솥갈루': 2,
 '케이': 2,
 '정릉시장': 2,
 '삼성': 2,
 '청춘': 2,
 '강북': 2,
 '성북동': 2,
```

- Twitter

• Twitter

```
In [24]:
twi = Twitter()
tokens_twi = twi.nouns(best_store_list)
tokens_twi
twi = nltk.Text(tokens_twi)
twi.tokens
twi.vocab()
new_twi=[]
for word in twi:
  if len(word) > 1 and word != ' ':
        new_twi.append(word)
new_twi
twi = nltk.Text(new_twi)
twi.tokens
twi.vocab()
data_twi = twi.vocab().most_common(150)
data_twi = dict(data_twi)
data_twi
```

```
C:₩Users₩daro9₩anaconda3₩lib₩site-packages₩konlpy₩tag₩_okt.py:17: UserWarning: "Twitter" has changed to "Okt" since KoNLPy v0.4.5.
  warn('"Twitter" has changed to "Okt" since KoNLPy v0.4.5.')
```

```
Out[24]: {'정릉': 2330,
 '맛집': 777,
 '식당': 639,
 '치킨': 543,
 '카페': 507,
 '선정릉역': 450,
 '커피': 424,
 '국민대': 305,
 '갈비': 303,
 '선정릉': 301,
 '칼국수': 283,
 '떡볶이': 280,
 '시장': 269,
 '공장': 254,
 '대국': 235,
 '족발': 217,
 '서울': 217,
```

-Komoran

- Komoran

```
In [25]:
kom = Komoran()
tokens_kom = kom.nouns(best_store_list)
tokens_kom
kom = nltk.Text(tokens_kom)
kom.tokens
kom.vocab()
new_kom=[]
for word in kom:
  if len(word) > 1 and word != ' ':
        new_kom.append(word)
new_kom
kom = nltk.Text(new_kom)
kom.tokens
kom.vocab()
data_kom = kom.vocab().most_common(150)
data_kom = dict(data_kom)
data_kom
```

```
Out[25]: {'점름': 2628,
 '맛집': 1158,
 '신당': 680,
 '치킨': 551,
 '카페': 514,
 '선점름역': 450,
 '국민대': 407,
 '시장': 405,
 '커피': 366,
 '선점름': 301,
 '떡볶이': 292,
 '칼국수': 286,
 '근점': 266,
 '갈비': 254,
 '분식': 251,
 '대국': 244,
 '서울': 229,
 '초밥': 203,
 '사바': 200,
```

-Hannanum

- Hannanum

```
In [26]:
han = Hannanum()
tokens_han = han.nouns(best_store_list)
tokens_han
han = nltk.Text(tokens_han)
han.tokens
han.vocab()
new_han=[]
for word in han:
  if len(word) > 1 and word != ' ':
        new_han.append(word)
new_han
han = nltk.Text(new_han)
han.tokens
han.vocab()
data_han = han.vocab().most_common(150)
data_han = dict(data_han)
data_han
```

```
Out[26]: {"(점름점)",": 800,
 '맛집': 406,
 "(선점름역점)",": 400,
 "(선점름점)",": 200,
 '서경대점': 152,
 '점름시장': 152,
 '점름': 151,
 '국민대': 144,
 '포도집': 114,
 '자작나무': 107,
 "'스타벅스": 100,
 "(롯데사로선점름점)",": 100,
 "'맛칼점죽": 100,
 "'사바사바치킨&비": 100,
 '성신여대점': 71,
 '도이칠란드': 68,
 '#점름맛집': 67,
 '지하세계': 60,
 '근처': 58,
```

분석결과 여러 어절을 하나의 품사로 분석 가능함으로써 형태소 분석기의 적용 분야에 따라 공백이 포함된 고유명사(영화 제목, 음식점명, 노래 제목, 전문 용어 등)를 더 정확하게 분석할 수 있는 Twitter 가 아닌 Hannanum 가 내가 원하는 데이터에 더 가까운 것을 볼 수 있다.

따라서 형태소 분석기 Hannanum 사용해 빈도분석 및 시각화 워드클라우딩을 진행하였다.

데이터에서 반복되는 패턴이 없어 TF-IDF 분석이 어려워 빈도출현이 잦은 음식점이름들로 임의의 리스트를 만들어줌

```
df['음식점이름'].head(6)
```

```
0          포도집
1         자작나무
2        '스타벅스
3        '맛깔참죽
4    '사바사바치킨&비
5         도이칠란드
Name: 음식점이름, dtype: object
```

```
# 각 문서별 단어 빈도 계수
import random

best_store_kookmin_1 = []
best_store_kookmin_2 = []
best_store_kookmin_3 = []
best_store_kookmin_4 = []
best_store_kookmin_5 = []

store_name = ['포도집', '자작나무', '스타벅스', '사바사바치킨', '도이칠란드']

for i in range(0,6):
    best_store_kookmin_1.append(random.choice(store_name))
for i in range(0,7):
    best_store_kookmin_2.append(random.choice(store_name))
for i in range(0,8):
    best_store_kookmin_3.append(random.choice(store_name))
for i in range(0,9):
    best_store_kookmin_4.append(random.choice(store_name))
for i in range(0,10):
    best_store_kookmin_5.append(random.choice(store_name))
```

```
best_store_kookmin_1 = " ".join(best_store_kookmin_1)
best_store_kookmin_2 = " ".join(best_store_kookmin_2)
best_store_kookmin_3 = " ".join(best_store_kookmin_3)
best_store_kookmin_4 = " ".join(best_store_kookmin_4)
best_store_kookmin_5 = " ".join(best_store_kookmin_5)
```

```
best_store_kookmin = [best_store_kookmin_1,best_store_kookmin_2,best_store_kookmin_3,best_store_kookmin_4,best_store_kookmin_5]
```

2. TF-IDF 데이터 사용

TF_IDF 를 활용하여 전체데이터를 분석할려고 했는데 양이 너무 많아 Kernel 이 다운되버려서 처음부터 다시 만들어야했다...

전체 데이터에서 추출하여 TF-IDF 분석을 진행하려고 했는데 반복되는 패턴이 없어서 random 함수를 사용하여 임의로 데이터를 만들어 주었다.

4. 실험 내용

실험의 순서

1) 실험에 필요한 모듈 불러오기 및 import
2) 데이터 crawling 및 공공데이터 전처리 후 합쳐서 데이터 가공
3) 4 가지 형태소 분석기 사용하여 분석 및 알맞은 분석기 선택
4) 분석한 데이터 시각화 및 워드클라우딩
5) TF-IDF 분석

1. 실험에 필요한 모듈 불러오기 및 import

필요한 모듈 불러오기

```
#!pip install selenium
```

```
#!pip install eunjeon
```

```
#!pip install konlpy
import nltk
from konlpy.tag import Kkma
from konlpy.tag import Twitter
from konlpy.tag import Komoran
from konlpy.tag import Hannanum
from konlpy.tag import Mecab
from selenium import webdriver
from konlpy.corpus import kolaw

import matplotlib.pyplot as plt
from wordcloud import WordCloud
from wordcloud import STOPWORDS
import numpy as np
from PIL import Image
from wordcloud import ImageColorGenerator

import requests
import pandas as pd
from bs4 import BeautifulSoup
```

2. 데이터 crawling 및 공공데이터 전처리 후 합쳐서 데이터 가공

- 맛집 추천 웹사이트에서 Crawling한 데이터 가중치 주고 원본데이터와 합치기

In [20]:
```
website_list = website_list*50
```

In [21]:
```
best_store_list.append(website_list)
```

In [22]:
```
best_store_list = "".join(map(str, best_store_list))
best_store_list = best_store_list.replace('₩n',' ').replace('.', ' ').replace(',
best_store_list
```

Out[22]:
```
'짜앤짬 대성각 청수만남 주식회사 국제 스카이웨이호텔 곰의집 중국관 오야오야 토속
집 장충족발 이모네포차 원복집 삼미원 국시집 화원정 옛날중국집 압구정봉구비어 오늘
김밥 100년 설렁탕 장위곱창 방이샤브샤브 칼국수 종암점 고려한방삼계탕 맘스터치 동
덕여대점 정오네고추장불고기 동화루 대머리식당 참맛감자탕 일락 고우곱창 우리집 전
주집 쌍다리식당 별미정 동일기사식당 미락 원조칼국수 꽃샘 서울강촌쌈밥 장위기사식
당 고(공)식당 화로상회 한성대점 야야(YAYA) 나팔바지 장충할매왕족발보쌈 참나라바베
큐치킨 토크쇼 황제 셰프의돈까스 숭례식당 승리장 마포갈비 대왕유통직영셀프민물장어
청수장 마산미더덕 마동장 영순관 오며가며 투다리 마당쇠호프치킨 마포갈비 피자스쿨
성신여대점 철판남자 서향&아리산 박성춘 남원추어탕 전문점 고대맛집 상만복집 장위동
유성집 고려성 이가한우곱창구이 한울 친구네 오비하우스 할매순대국 고려대점 이천통
닭 할매순대국 종로곱창 호수 마마칼국수 갈채 춘천골닭갈비 피엔씨(PNC)푸드 황제곱창
수라간 이야기 성신여대점 남한강 매운탕 베트남노상식당 뚱땡이왕족발 나드리식당 중
국관 영빈관 본죽앤비빔밥 돈암점 싸다횟집 국수나무 동덕여대점 장충동족발집 손국수
집 한일반점 성북왕족발 서병장대김일병 만리성 아부지가 어부 주당 삼선식당 오바케
돈부리 춘천닭갈비 백암토종순대 이태리총각 식샤 양지식당 송추집 서울돈까스 구식냉
삼식당 한마리채 서서갈비 옛날통닭(한성대점) 으뜸식당 아다미분식 양가네손칼국수 배
떼기곱창 오백집 한울감자국 구룡포계절회집 사천장 황소식당 성북동참치 금테두른구공
탄 투다리(석계역점) 달빛 내마음의 풍차 굴렁쇠 양자설렁탕 페리카나치킨 장원산꽃게
멕시칸바베큐 청목 산촌기사식당 열린호프 로터리감자국 만나식당 치킨주막 핏짜피자
```

3. 4 가지 형태소 분석기 사용하여 분석 및 알맞은 분석기 선택

- 가장 적합한 Hannanum 선택

- Hannanum

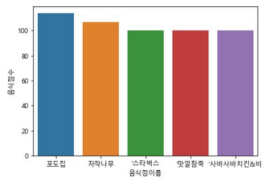
In [26]:
```
han = Hannanum()
tokens_han = han.nouns(best_store_list)
tokens_han
han = nltk.Text(tokens_han)
han.tokens
han.vocab()
new_han=[]
for word in han:
    if len(word) > 1 and word != ' ':
        new_han.append(word)
new_han
han = nltk.Text(new_han)
han.tokens
han.vocab()
data_han = han.vocab().most_common(150)
data_han = dict(data_han)
data_han
```

Out[26]:
```
{"(정통점)',": 800,
 '맛집': 406,
 "(선정통역점)',": 400,
 "(선정통점)',": 200,
 '서경대점': 152,
 '정통시장': 152,
 '정통': 151,
 '국민대': 144,
 '포도집': 114,
 '자작나무': 107,
 "'스타벅스": 100,
 "(봉은사로선정통점)',": 100,
 "'맛잘참죽": 100,
 "'사바사바치킨&비": 100,
 '성신여대점': 71,
 '도이힐랜드': 68,
 '#정통맛집': 67,
 '지하세계': 60,
 '근처': 58,
```

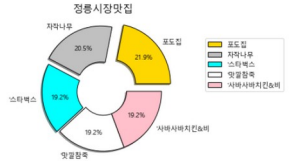
4. 분석한 데이터 시각화 및 워드클라우딩

- 시각화

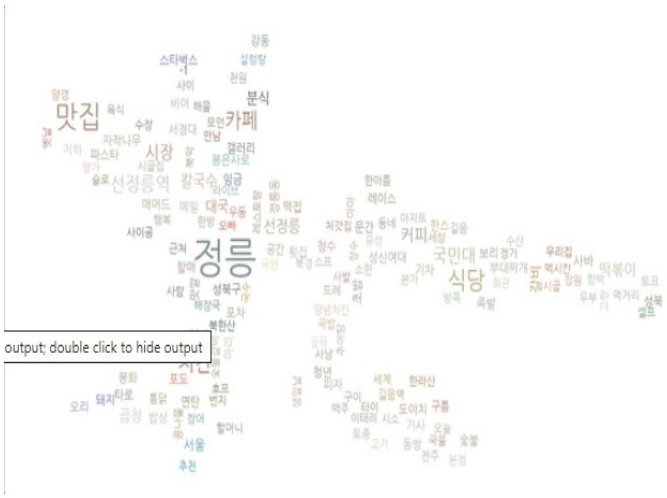
```
In [34]: sns.barplot(data=df_1, x="음식점이름", y="음식점수") #seaborn 사용
Out[34]: <AxesSubplot:xlabel='음식점이름', ylabel='음식점수'>
```




```
In [35]: wedgeprops = {'width': 0.6, 'linewidth': 1, 'edgecolor': 'black'}
plt.pie(df_1['음식점수'], labels = df_1['음식점이름'], autopct = '%.1f%%', colors = ['gold', 'silver', 'cyan', 'white', 'pink'], e
plt.title('정릉시장맛집', size = 15)
plt.legend(loc = (1.2, 0.5))
plt.show()
```



- 워드클라우딩



5. TF-IDF 분석

```
['도이칠란드', '사바사바치킨', '스타벅스', '자작나무', '포도집']
[[1.         2.         0.         1.         2.36464311]
 [1.         1.         2.81093022 1.         2.36464311]
 [3.         4.         0.         1.         0.        ]
 [1.         2.         1.40546511 3.         2.36464311]
 [1.         1.         2.81093022 3.         3.54696467]]
```

```python
# 단순 TF기반 문서/단어 행렬 생성
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(tokenizer=str.split, norm = None, use_idf = False
doc_term_mat = vectorizer.fit_transform(best_store_kookmin)
doc_term_mat_d = doc_term_mat.toarray()

print(vectorizer.get_feature_names())
print(doc_term_mat_d)
```

```
['도이칠란드', '사바사바치킨', '스타벅스', '자작나무', '포도집']
[[1. 2. 0. 1. 2.]
 [1. 1. 2. 1. 2.]
 [3. 4. 0. 1. 0.]
 [1. 2. 1. 3. 2.]
 [1. 1. 2. 3. 3.]]
```

```python
# 정규화 기준의 변경, norm = "l1", norm = "l2" (default), norm = None
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(tokenizer=str.split, norm = "l2")        #norm = "l2
doc_term_mat = vectorizer.fit_transform(best_store_kookmin)
doc_term_mat_d = doc_term_mat.toarray()

print(vectorizer.get_feature_names())
print(doc_term_mat_d)
```

```
['도이칠란드', '사바사바치킨', '스타벅스', '자작나무', '포도집']
[[0.29371727 0.58743454 0.         0.29371727 0.69453652]
 [0.24623622 0.24623622 0.69215284 0.24623622 0.58226079]
 [0.58834841 0.78446454 0.         0.19611614 0.        ]
 [0.21533095 0.43066189 0.30264013 0.64599284 0.50918084]
 [0.17822428 0.17822428 0.50097601 0.53467283 0.63215522]]
```

6. 결론

처음에 VIEW 검색 리뷰를 접목시켜서 빈도분석으로 맛집을 찾는 아이디어를 생각했을 때 과연 이게 될까? 라는 생각이 많이 들었다.

VIEW 데이터 크롤링해서 형태소 분석했을 때 예를 들어서 “정릉쭈꾸미” 라는 음식점이 “정릉” + “쭈꾸미”로 형태소 분석이 되는 경우도 많고 동적 크롤링 과정에서도 많은 어려움이 있었다.

주제도 바꿀까 생각을 해봤는데 정릉에 사는 자취생으로서 주위의 맛집은 도저히 포기할 수 없었다.

전처리한 데이터를 빈도분석 하여 맛집을 분석할 결과 실제로 내가 근처에 자주 가는 도이칠란드, 청년밥상 문간, 사이공, 처갓집 양념치킨, 먹거리곱창 등등이 상위 빈도에 나타나는 것을 확인 할 수 있었다.

분석 최종 결과물

국민대 및 정릉시장 주변 맛집 분석에서 맛집으로 판명난 음식점 리스트

“포도집, 자작나무, 스타벅스, 맛깔참죽, 사바사바치킨, 도이칠란드, 유성분식, 장수식당, 꿀꿀이생일날, 청년밥상문간, 사이공, 처갓집양념치킨,

한라산도새, 장원식당, 성원식당, 청록원, 황궁, 토크쇼, 시골집, 마마우동, 먹거리곱창”