Team2 Project

챗봇 기반 사용자 맞춤형 일정 추천 서비스

오늘 뭐하지?

텍스트마이닝 Team 2
오승준 이원빈 이현지 주다영

목차

주제 선정 배경

오늘 뭐하지?

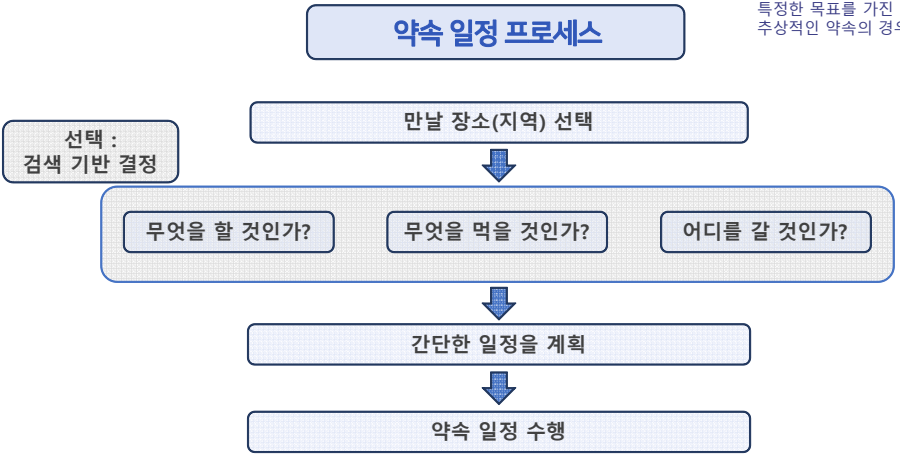
주제 선정 배경

타겟이 불분명한 검색에는

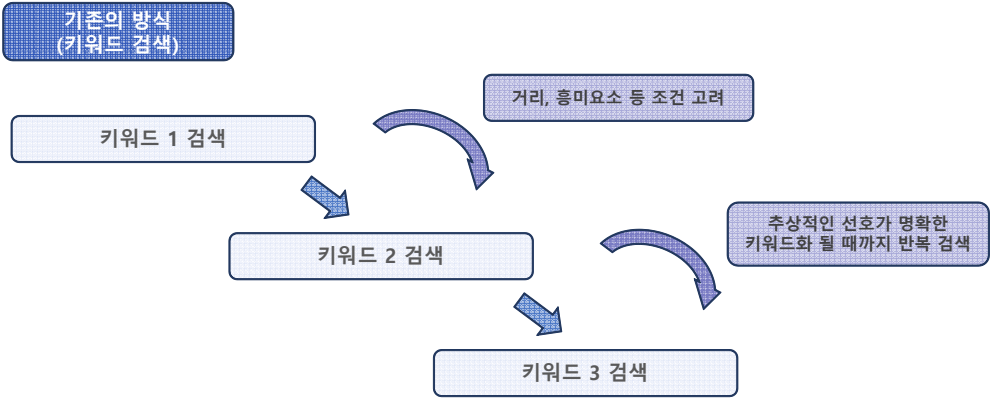
잠재적 NEEDS가 존재할 것이다.

주제 선정 배경

특정한 목표를 가진 약속을 제외한
추상적인 약속의 경우



주제 선정 배경


기존의 방식
(키워드 검색)
키워드 1 검색
거리, 흥미요소 등 조건 고려
키워드 2 검색
추상적인 선호가 명확한
키워드화 될 때까지 반복 검색
키워드 3 검색

주제 선정 배경

프로젝트 방식
(문장형 검색)

추상적인 요인들을 반영한 검색
(~같은, 흥미요소, 느낌, 분위기 등)

메인 키워드에 의한 추천

+

추상 요소에 의한 추천

추상적인 선호가 반영된 구체적 예시 제공

성공적인 추천
또는
추상적 개념의 구체화 (예시 제공)

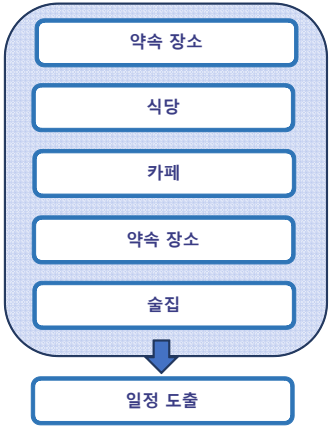
프로젝트 주제

서울시에서 보내는 최고의 하루를 위한

'챗봇 기반 사용자 맞춤형 일정 추천 서비스'

파이프라인

< 대화 형식 >

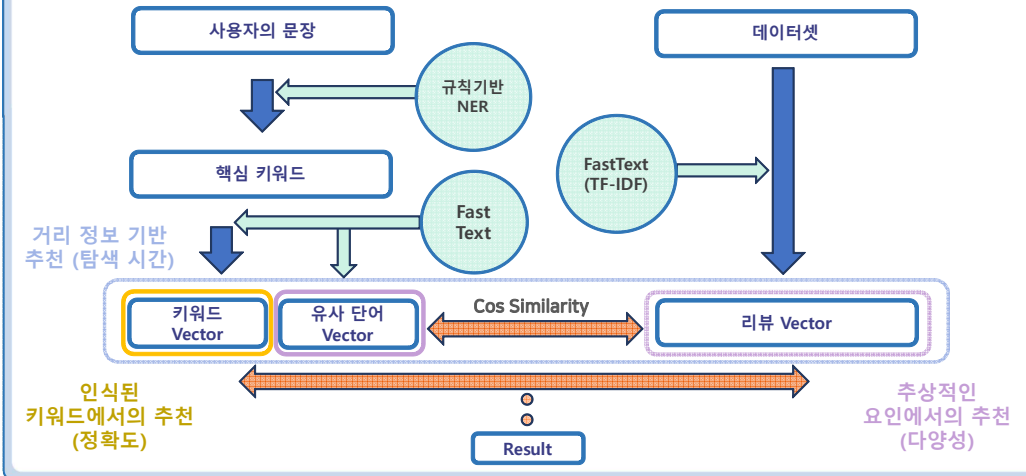


< 출력 예시 >

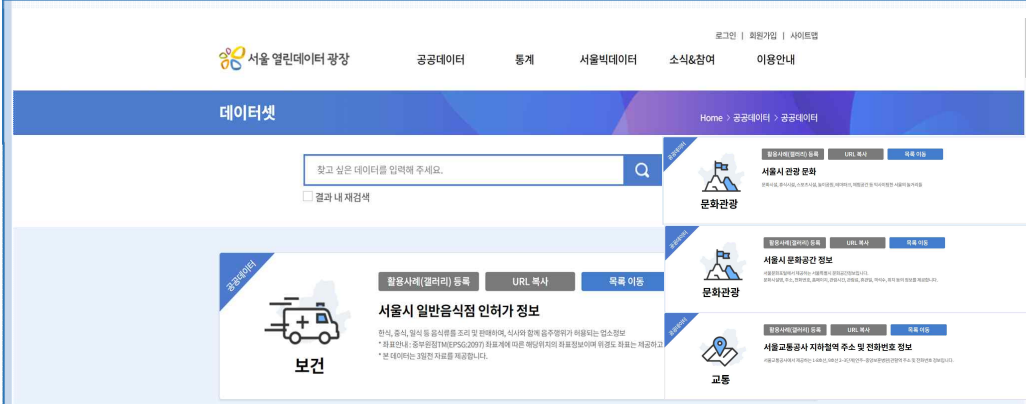
```
안녕하세요! 멋진 하루를 보낼 수 있도록 도와드리겠습니다!
오늘 어디를 방문하실 계획인가요? 방문 계획이 있다면 언근 지하철 역을 말씀해주시겠어요?
(지하철 역 입력을 원하지 않으시면 x를 입력해주세요)

감사합니다! 그럼 이제 일정을 세워보겠습니다!
금강산도 식후경! 우선 밥을 먹는 것은 어떨까요? 드시고 싶은 음식의 느낌을 자유롭게 적어주세요.
(식당 추천을 원하지 않으시면 x를 입력해주세요)


느낌있는 카페는 어떠신가요? 카페에 대해 자유롭게 적어주세요
(카페 추천을 원하지 않으시면 x를 입력해주세요)


재미있는 액티비티나 문화생활을 즐겨보실래요? 가고 싶은 장소에 대한 느낌을 자유롭게 적어주세요
(장소 추천을 원하지 않으시면 x를 입력해주세요)

말씀해주신 결과를 바탕으로 다음과 같은 장소들을 찾아보았습니다! 마음에 드는 번호을 입력해주세요

1: 장소1
2: 장소2
3: 장소3

마지막으로 하루를 마무리하는 술집은 어떠신가요? 가고 싶은 술집에 대해 자유롭게 적어주세요
(술집 추천을 원하지 않으시면 x를 입력해주세요)

말씀해주신 결과를 바탕으로 다음과 같은 술집들을 찾아보았습니다! 마음에 드는 번호을 입력해주세요
```

```
말씀해주신 결과를 바탕으로 다음과 같은 술집들을 찾아보았습니다! 마음에 드는 번호를 입력해주세요

1: 술집1 / 대표메뉴: 술집1메뉴1, 술집1메뉴2
2: 술집2 / 대표메뉴: 술집2메뉴1, 술집2메뉴2
3: 술집3 / 대표메뉴: 술집3메뉴1, 술집3메뉴2

감사합니다~ 오늘 일정에 대해 정리해드리겠습니다!
장소: 장소1
술집: 술집1 / 대표메뉴: 술집1메뉴1, 술집1메뉴2
```

파이프라인


사용자의 문장
규칙기반 NER
핵심 키워드
Fast Text
거리 정보 기반 추천 (탐색 시간)
데이터셋
FastText (TF-IDF)
키워드 Vector
유사 단어 Vector
Cos Similarity
리뷰 Vector
인식된 키워드에서의 추천 (정확도)
추상적인 요인에서의 추천 (다양성)
Result

데이터 수집



자료 수집: 서울 열린 데이터 광장 – 음식점 및 관광지, 문화 공간 리스트, 지하철역 정보

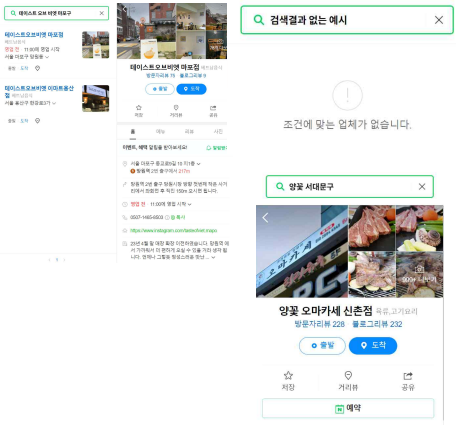
데이터 수집

Web Crawling Method:
Selenium
&
Beaultiful Soap

자료 수집: 네이버 지도 – 음식점 및 관광지 리스트 기반 업종/별점/메뉴/리뷰 크롤링

데이터 수집

```
try:
    search_url = f'https://map.naver.com/v5/search/' + keyword
    driver.get(search_url)
    driver.implicitly_wait(3)
    driver.switch_to.frame('searchIframe')
    driver.implicitly_wait(3)
    try:
    # 검색결과 없을 경우
    #app-root > div > div.XUrfU > div.vEAWt > div.FYvSc
        if(driver.find_element(By.CSS_SELECTOR, 'div.FYvSc').text == '조건에 맞는 업체가 없습니다.'):
            # dataframe에 해당 ROW에 NaN 처리하고 continue
            for key in result:
                if key == '리뷰':
                    for i in range(10):
                        df.loc[index, f'{key}_{i}'] = result[key][i]
                else:
                    df.loc[index, key] = result[key]
            continue
    except:
        pass

    # 첫번째 가게 들어가기 ... 검색결과에서 골라야되는 4가지 상황
    #_pcmap_list_scroll_container > ul > li:nth-child(1) > div.CHC5F > a
    #_pcmap_list_scroll_container > ul > li > div.CHC5F > a > div > div > span.place_bluelink.TYaxT
    #_pcmap_list_scroll_container > ul > li:nth-child(1) > div.CHC5F > a > div > div > span.place_bluelink.TYaxT
    #_pcmap_list_scroll_container > ul > li:nth-child(1) > div.qbGlu > div > a:nth-child(1) > div > div > span.place_bluelink.YwYLL
    driver.find_element(By.CSS_SELECTOR, 'span.place_bluelink').click()
    driver.implicitly_wait(3)
    driver.switch_to.default_content()
    driver.switch_to.frame('entryIframe')
    driver.implicitly_wait(3)
    html = driver.page_source
    soup = BeautifulSoup(html, 'lxml')
except:
    # 검색결과말고 바로 세부창 갔을 때
    try:
        driver.switch_to.default_content()
        driver.switch_to.frame('entryIframe')
        driver.implicitly_wait(3)
        html = driver.page_source
        soup = BeautifulSoup(html, 'lxml')
```



검색 결과가 일부 없는 케이스 별 대처 -> HTML Tag 기준 크롤링

데이터 수집

```
# 별점과 업종
#app-root > div > div > div > div.place_section.OP4V8 > div.zD
#_title > span.DJJvD
try:
    rating = soup.select('span.PXMot.LXIwF > em')
    result['별점'] = rating[0].text
except:
    # dataframe - rating column에 NaN 처리
    pass
try:
    type = soup.select('span.DJJvD')
    result['업종'] = type[0].text
except:
    # dataframe - type column에 NaN 처리
    pass

# 대표 메뉴 1. 옵션 1 / 2. 옵션 2
#app-root > div > div > div > div:nth-child(7) > div > div:nt
#app-root > div > div > div > div:nth-child(6) > div > div:nt
try:
    menus = soup.find_all("div", "erVoL")
    if menus == []:
        menus = soup.find_all("a", "ihmWt")
    menu_texts = [menu.get_text().strip() for menu in menus]
    result['대표메뉴'] = ', '.join(menu_texts)
#app-root > div > div > div > div:nth-child(6) > div > div:nt
except:
    # dataframe - menu column에 NaN 처리
    pass
```

```
# 리뷰 1. 리뷰유무 확인 및 접속용 텍스트 / 2. 더보기 버튼 / 3. 리뷰 데이터
#app-root > div > div > div > div.place_section.OP4V8 > div.zD5Nm.f7aZ0 > div.dAaGb > span.
#app-root > div > div > div > div:nth-child(7) > div:nth-child(3) > div:nth-child(3) > div.
#app-root > div > div > div > div:nth-child(7) > div:nth-child(3) > div:nth-child(3) > div.
try:
    reviewnum = soup.select('span.PXMot > a.place_bluelink > em')
    if (reviewnum[0].text != "0"):
        driver.find_element(By.CSS_SELECTOR, 'span.PXMot > a.place_bluelink > em').click()
        driver.implicitly_wait(3)
        while True:
            try:
                driver.find_element(By.CSS_SELECTOR, 'span.rvCSr').click()
                time.sleep(0.1)
            except:
                break
        html = driver.page_source
        soup = BeautifulSoup(html, 'lxml')
        reviews = soup.find_all("span", "zPfVt")
        review_texts = [review.get_text().strip() for review in reviews]
        # review column 0 ~ 9에 순서적으로 저장 / 없는 만큼은 NaN 처리
        for i, review in enumerate(review_texts):
            if i < 10:
                result['리뷰'][i] = review
except:
    # dataframe - review column에 NaN 처리
    pass
try:
    for key, value in result.items():
        if key == '리뷰':
            for i in range(10):
                df.loc[index, f'{key}_{i}'] = value[i]
        else:
            df.loc[index, key] = value
except:
    continue
```

별점, 업종, 메뉴, 리뷰 크롤링

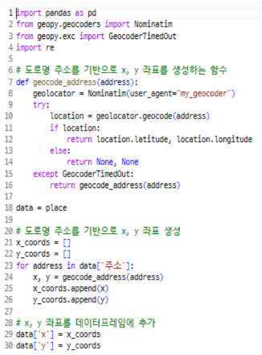
데이터 전처리

<수집된 데이터 : 약 50,000개>

- 네이버 평점 3.0 이하의 장소들은 Filtering
- 대표메뉴에 (300ml, 中)등이 들어간 경우 정규화
- Geopy를 위한 도로명 주소 정규화
- 업종, 분류 항목 재편성 및 재분류
- 결측치(리뷰, 평점, 주소 등)가 너무 많은 경우 제거
- 동일한 장소 Filtering

(동일한 식당이 1F, 4F를 모두 운영)

식당 카페 장소 술집



```python
import pandas as pd
from geopy.geocoders import Nominatim
from geopy.exc import GeocoderTimedOut
import re

# 도로명 주소를 기반으로 x, y 좌표를 생성하는 함수
def geocode_address(address):
    geolocator = Nominatim(user_agent="my_geocoder")
    try:
        location = geolocator.geocode(address)
        if location:
            return location.latitude, location.longitude
        else:
            return None, None
    except GeocoderTimedOut:
        return geocode_address(address)

data = place

# 도로명 주소를 기반으로 x, y 좌표 생성
x_coords = []
y_coords = []
for address in data['주소']:
    x, y = geocode_address(address)
    x_coords.append(x)
    y_coords.append(y)

# x, y 좌표를 데이터프레임에 추가
data['x'] = x_coords
data['y'] = y_coords
```

<도로명 주소 -> 위도, 경도>

<Data Cleansing>

데이터 전처리

<업종+메뉴>

```python
import re
import pandas as pd

def menu_cleansing(df):
    # '대표메뉴' column을 문자열로 변환
    df['대표메뉴'] = df['대표메뉴'].astype(str)

    # 한글이 아닌 문자 및 괄호 안의 문자를 제거하여 메뉴 이름 추출
    menu_list = []
    for menu in df['대표메뉴']:
        if menu.strip() != '' and menu.strip().lower() != 'none':
            # 괄호와 괄호 안의 문자를 제거
            cleaned_menu = re.sub('\(.*?\)', '', menu)
            # 한글과 쉼표(,)와 공백을 제외한 문자 제거
            cleaned_menu = re.sub('[^가-힣,\s]', '', cleaned_menu)
            # 공백을 기준으로 분할하여 메뉴 이름을 추출
            parts = cleaned_menu.split()
            menu_list.append(' '.join(parts))
        else:
            menu_list.append(menu)

    return menu_list

# 가정: food, cafe, bar는 이전 단계에서 정의된 데이터프레임이라고 가정합니다.

# 대표메뉴 열을 정제된 메뉴 리스트로 갱신
food['대표메뉴'] = menu_cleansing(food)
cafe['대표메뉴'] = menu_cleansing(cafe)
bar['대표메뉴'] = menu_cleansing(bar)
```

```python
food['대표메뉴'] = food['대표메뉴'].fillna(' ')
cafe['대표메뉴'] = cafe['대표메뉴'].fillna(' ')
bar['대표메뉴'] = bar['대표메뉴'].fillna(' ')
place['업종'] = place['업종'].fillna(' ')

food['특성'] = food['업종'] + ' ' + food['대표메뉴']
cafe['특성'] = cafe['업종'] + ' ' + cafe['대표메뉴']
bar['특성'] = bar['업종'] + ' ' + bar['대표메뉴']
place['특성'] = place['업종'] + ' ' + place['장소명']
```

```
1 food['특성']

1          양식 치지 슈프림 프라이즈, 셰퍼드 파이 , 풀드 포크 파히타, 바나나 쵸코칩 와플
2          이탈리아음식 레드 크림 파스타, 레드 갈릭 밥스타, 레드 커리 파스타, 레드 토마토...
3                        육류,고기요리 한우등심 , 안심 , 채끝등심 , 부채살
4                        중식당 해물누룽지탕, 자장면, 간자장면, 삼선간자장면
5                 샐러드 스테이크 샐러드, 햄치즈 샌드위치, 불고기 샐러드, 새우 샐러드
                                  ...
26625                                   칼국수,만두
26626      중식당 ,짜장탕수육칠리새우, ,차돌짬뽕탕수육칠리새우, 굴차돌짬뽕, 차돌수제비짬뽕
26627                 분식 차돌떡볶이, 차돌떡볶이, 통큰오짱떡볶이, 통큰오짱떡볶이
26631                   아시아음식 모모, 치킨차우면, 베지차우면, 믹스차우면
26632                   한식 돔베고기 , 갈치튀김, 제주창도름순대, 고사리육개장
Name: 특성, Length: 22101, dtype: object
```

<리뷰>

```python
import re
from konlpy.tag import Okt

# 형태소 분석기 인스턴스 생성
okt = Okt()

# Stopwords
stop_words = stop_words

# 길이가 1인 글자 제거
def remove_single_chars(tokens):
    filtered_tokens = [token for token in tokens if len(token) > 1]
    return filtered_tokens

# Stopwords 제거
def remove_stopwords(tokens):
    filtered_tokens = [token for token in tokens if token not in stop_words]
    return filtered_tokens

# 데이터 프레임의 열에 대해 길이가 1인 글자 제거 및 Stopwords 제거
def preprocess_text(text):
    # 쉼표로 문자열 분리하여 토큰 리스트 생성
    tokens = text.split(',')

    # 길이가 1인 글자 제거
    tokens = remove_single_chars(tokens)

    # Stopwords 제거
    tokens = remove_stopwords(tokens)

    return tokens

# 데이터 프레임의 열에 적용하여 전처리 수행
data1['doc_key'] = data1['doc_key'].apply(lambda x: preprocess_text(x))
data2['doc_key'] = data2['doc_key'].apply(lambda x: preprocess_text(x))
```

```
1 data1['doc_key']

0        [맛있다, 분위기, 좋다, 요즘, 날씨, 가면, 정말, 좋다, 치킨, 맛있다 마늘,...
1        [커피, 맛있다, 분위, 좋다, 친절하다 부리또, 따뜻하다, 맛있다, 많다, 샐러드...
2        [아주, 옛날, 방문, 강남, 인데, 그대로, 뭔가, 추억, 돋다, 역시, 그대, ...
3        [코다리, 조림, 매콤, 이랑, 먹다, 맛있다 음식, 짜다, 정석, 코다리, 큼직하...
4        [주말, 이나, 공휴일, 종종, 찾다, 좋다, 가성, 훌륭하다, 중국집, 오늘, 고...
                                  ...
26628    [어흑너무, 맛있다, 조용하다, 분위기, 맛있다, 음식, 좋다, 사시미, 에는, 새...
26629    [꽈배기, 맛있다, 가격, 착하다, 사장, 완전, 친절하다, 이라, 가격, 착하다,...
26630    [모임, 방문, 메뉴, 다양하다, 여러가지, 보다, 맛있다, 개인, 해물, 보단, ...
26631    [어딘가, 히말라야, 산간, 지방, 동네, 이웃집, 크다, 아주머니, 받다, 보다,...
26632    [고기국수, 국물, 끝내주다, 맛있다, 어유, 부랴부랴, 먹다, 사진, 찍다, 따다...
Name: doc_key, Length: 26633, dtype: object
```

<Stopword.txt>

```
stopword.txt ×
탕탕
쾅쾅
둥둥
봐
봐라
아이야
아니
와아
응
아이
참나
년
월
일
령
영
일
이
```

- 추후 벡터화 할 column들에 대해선 조금 더 신경쓴 전처리
- 리뷰 데이터는 Hanspell, 한글만 남긴 뒤 Okt를 이용한 품사 태깅, Lemmatization
  -> 명사, 형용사, 부사 추출
- Stopword Dictionary 구성
- 업종 + 메뉴 데이터는 자체적으로 키워드화

키워드 추출

사과가 먹고 싶어.

나는 오늘 색다른 파스타를 먹고 싶어.

나는 오늘 뭔가 달콤한 것이 먹고 싶어. 그런데 도넛은 빼고.

키워드 추출

가정

1. 사용자의 응답은 '나는', '오늘', '식당'과 같이 유사한 stopword를 갖는 경우가 많을 것이다.

2. 사용자의 타겟이 뚜렷해서 해당 키워드에 대한 검색 결과를 원했다면
'~같은', '~비슷한' 이라는 표현을 쓰지 않을 것이다.

3. 키워드를 검색 결과에서 제외하는 경우는 부정 단어 앞쪽에 제외 대상이 존재한다.
(~빼고, ~말고, ~것은 싫어)

키워드 추출

가정

1. 사용자의 응답은 '나는', '오늘', '식당'과 같이 유사한 stopword를 갖는 경우가 많을 것이다.

2. 사용자의 타겟이 뚜렷해서 해당 키워드에 대한 검색 결과를 원했다면 '~같은', '~비슷한' 이라는 표현을 쓰지 않을 것이다.

3. 키워드를 검색 결과에서 제외하는 경우는 부정 단어 앞쪽에 제외 대상이 존재한다. (~빼고, ~말고, ~것은 싫어)

- 문장에 대해 형태소 분석(Okt) 후, Lemmatization.

main_keyword -> 명사

sub_keyword -> 부사, 형용사

(단, keyword에서 제외할 단어 적절히 정의)

'~같다' '비슷하다' 등의 단어가 있다면 주변의 main_keyword로 분류된 단어를 지우고, Islike 변수를 true로 받은 다음 추천에서 추상 항목들의 비중을 늘림

'~빼고', '~는 싫어' 등의 단어가 있다면 그 앞의 명사, 형용사, 부사 등을 제외 리스트에 넣음

키워드 추출

```
def extract_keywords(sentence):
    # 형태소 분석기 인스턴스 생성
    okt = Okt()

    # 형태소 분석 수행
    morphemes = okt.pos(sentence, stem=True)

    # 부사, 형용사, 명사 추출
    keywords_main = [word for word, tag in morphemes if tag in ('Noun')]
    keywords_sub = [word for word, tag in morphemes if tag in ('Adjective', 'Adverb')]

    # 제외하고 싶은 단어들
    exclude_words_main = ["나", "오늘", "가다", "먹다", "곳", "식당", "카페", "술", "얼다", "어떻다"]

    # 제외할 동사 및 조사 설정
    exclude_verbs_and_josa = ['특다', '제외하다', '말고', '빼고', '싫다', '싫어', '싫고']

    # '비슷한, ~같은'과 키워드가 있는가?
    Islike = False

    # 이후 검색에서 제외할 키워드
    exclude_keywords = []

    # '말다' 또는 '비슷하다' 키워드가 있으면 Islike를 True로 설정하고 키워드 제거
    for i, (word, tag) in enumerate(morphemes):
        if word in ['말다', '같이', '비슷하다']:
            Islike = True
            # 이전 단어가 명사, 형용사, 부사인 경우 제외 리스트에 추가
            j = i - 1
            while j >= 0:
                if morphemes[j][1] in ('Noun', 'Adjective', 'Adverb'):
                    exclude_words_main.append(morphemes[j][0])
                    exclude_keywords.append(morphemes[j][0])
                    break
                j -= 1

    # 동사 및 조사를 찾아 그 앞에 있는 명사, 형용사, 부사 제외
    for i, (word, tag) in enumerate(morphemes):
        if word in exclude_verbs_and_josa:
            j = i - 1
            # 명사, 형용사, 부사 찾기
            while j >= 0:
                if morphemes[j][1] in ('Noun', 'Adjective', 'Adverb'):
                    exclude_words_main.append(morphemes[j][0])
                    exclude_keywords.append(morphemes[j][0])
                    break
                j -= 1

    # 제외할 단어들을 제외한 새로운 리스트 생성
    keywords_main = [word for word in keywords_main if word not in exclude_words_main]
    keywords_sub = [word for word in keywords_sub if word not in exclude_words_main and word not in exclude_verbs_and_josa]
    exclude_keywords = [word for word in exclude_keywords if word not in ['비슷하다', '같이']]

    return keywords_main, keywords_sub, exclude_keywords, Islike
```

문장을 입력하세요: 과자나 음료수 가게를 추천해줘
추출된 주요 키워드 : ['과자', '음료수']
추출된 추상 키워드 (형용사, 부사) : []
제외 키워드 : []
Islike : False

문장을 입력하세요: 나는 오늘 색다른 파스타가 먹고 싶어
추출된 주요 키워드 : ['파스타']
추출된 추상 키워드 (형용사, 부사) : ['색다르다']
제외 키워드 : []
Islike : False

문장을 입력하세요: 나는 오늘 뭔가 달콤한 것이 먹고 싶어. 도넛 빼고
추출된 주요 키워드 : []
추출된 추상 키워드 (형용사, 부사) : ['달콤하다']
제외 키워드 : ['도넛']
Islike : False

문장을 입력하세요: 시원한 음식 없을까? 치킨같은거 말고
추출된 주요 키워드 : []
추출된 추상 키워드 (형용사, 부사) : ['시원하다']
제외 키워드 : ['치킨']
Islike : True

키워드 임베딩

fastText

```python
from gensim.models import fasttext

model = fasttext.load_facebook_model("/content/drive/MyDrive/wiki.ko.bin", encoding='utf-8')
```

```python
from gensim.models import FastText


# 추가 학습 데이터로 모델 업데이트
model.build_vocab(corpus, update=True)
model.train(corpus, total_examples=model.corpus_count, epochs=model.epochs)

# 모델을 파일로 저장
model.save('/content/drive/MyDrive/FastTextModel')
```

Korpora Dataset

Fine_Tuning 전

```python
# 유사 키워드 추출

word = "피자"
sim_words_list = model.wv.most_similar(word, topn=100)

print(sim_words_list[:10])
print(sim_words_list[40:50])
```

Spearman Correlation: 0.5264317163147282

```
[('피자집', 0.8375698328018188), ('피자로', 0.82587730884552), ('피자와', 0.8230692148208618), ('피자가', 0.8190242648124695), ('피자헛', 0.8169077038764954),
[('레스토랑에서는', 0.6795679330825806), ('닭고기', 0.6787909269332886), ('햄버거가', 0.6771535277366638), ('닭고기를', 0.6769299507141113), ('감자빵은', 0.6
```

Embedding 성능
소폭 하락

Fine_Tuning 후

```python
# 유사 키워드 추출

word = "피자"
sim_words_list = model.wv.most_similar([word], topn=100)

print(sim_words_list[:10])
print(sim_words_list[40:50])
```

Spearman Correlation: 0.5005037342263277

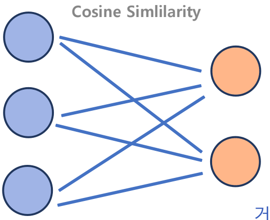
```
[('피자로', 0.9269040822982788), ('피자가', 0.9191816449165344), ('피자집', 0.9183799028396606), ('치즈피자', 0.9183230400085449), ('피자도', 0.9118190407752991), ('피자가게가', 0.90
[('고르곤졸라피자', 0.8599676489830017), ('치즈돈까스', 0.8590410947799683), ('블랙트러플스테이크시카고피자', 0.8589991331100464), ('치킨감자', 0.8588685393333435), ('샐러드스테이크'
```

키워드 임베딩

```
from sklearn.metrics.pairwise import cosine_similarity

# 코사인 유사도 계산 함수
def calculate_similarity(a, b):
    return cosine_similarity([a], [b])[0][0]

# 각 행에 대해 상위 5개 코사인 유사도의 평균을 계산하는 함수
def calculate_average_top_similarities(word_vec, key_vec):
    similarities = []
    for w in word_vec:
        for k in key_vec:
            similarities.append(calculate_similarity(w, k))
    similarities.sort(reverse=True)
    top_similarities = similarities[:5]
    return np.mean(top_similarities)

# 데이터프레임의 각 'key_vec'와의 코사인 유사도 계산
average_similarities = []
for index, row in df.iterrows():
    avg_similarity = calculate_average_top_similarities(word_vec, row['main_vec'])
    average_similarities.append((index, avg_similarity))

# 평균 유사도를 기준으로 내림차순 정렬
average_similarities.sort(key=lambda x: x[1], reverse=True)

# 상위 20개 결과 출력
top_20 = average_similarities[:20]
for index, avg_similarity in top_20:
    print(f"Index: {index}, Average Similarity: {avg_similarity}, Data: {df.loc[index]['장소명']}")
```

```
Index: 47142, Average Similarity: 0.9480803608894348, Data: 구구치킨 을지로골뱅이
Index: 4114, Average Similarity: 0.9454669952392578, Data: 하이덴치킨
Index: 47503, Average Similarity: 0.9443053007125854, Data: 오케이치킨
Index: 39013, Average Similarity: 0.9434806704521179, Data: 멕켄치킨
Index: 8205, Average Similarity: 0.9427908062934875, Data: 엘리트닭
Index: 51484, Average Similarity: 0.9425684213638306, Data: 미락
Index: 52211, Average Similarity: 0.9425684213638306, Data: 미락치킨 경복궁 본점
Index: 9274, Average Similarity: 0.9423180818557739, Data: 페리카나치킨
Index: 16374, Average Similarity: 0.9423180818557739, Data: 페리카나
Index: 16375, Average Similarity: 0.9423180818557739, Data: 페리카나
Index: 48909, Average Similarity: 0.9423180818557739, Data: 페리카나
Index: 48910, Average Similarity: 0.9423180818557739, Data: 페리카나
Index: 10096, Average Similarity: 0.941301345825l953, Data: 페리카나치킨
Index: 10097, Average Similarity: 0.9413013458251953, Data: 페리카나치킨
Index: 12371, Average Similarity: 0.9413013458251953, Data: 페리카나
Index: 12372, Average Similarity: 0.9413013458251953, Data: 페리카나
Index: 28416, Average Similarity: 0.9413013458251953, Data: 페리카나
Index: 28417, Average Similarity: 0.9413013458251953, Data: 페리카나
Index: 44771, Average Similarity: 0.9413013458251953, Data: 페리카나치킨
Index: 44772, Average Similarity: 0.9413013458251953, Data: 페리카나치킨
```

Keyword : 치킨



< 추천 알고리즘 >

행 사이의 코사인 유사도를
모두 구하고,
상위 5개의 평균값을 비교한다.

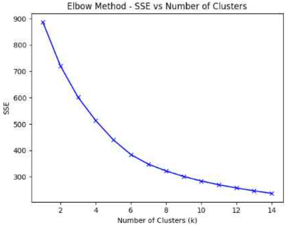
(연산량이 많지만,
두 벡터의 차원이 그렇게 높지 않고,
거리 정보 반영으로 탐색 범위가 넓지 않다)

```
for index, avg_similarity in top_20:
    print(f"Index: {index}, Average Similarity: {avg_similarity}, Data: {df.loc[index]['특성']}")
```

```
Index: 47142, Average Similarity: 0.9480803608894348, Data: ['마늘치킨', '치킨', '고추치킨', '후라이드치킨', '닭강정', '양념치킨']
Index: 4114, Average Similarity: 0.9454669952392578, Data: ['치킨', '반반치킨', '후라이드치킨', '닭강정', '양념치킨', '순살치킨']
Index: 47503, Average Similarity: 0.9443053007125854, Data: ['크리스피치킨', '파닭치킨', '치킨', '닭강정', '양념치킨', '불닭치킨']
Index: 39013, Average Similarity: 0.9434806704521179, Data: ['치킨', '반반치킨', '핫치킨', '후라이드치킨', '닭강정', '양념치킨']
Index: 8205, Average Similarity: 0.9427908062934875, Data: ['크리스피치킨', '치킨', '반반치킨', '후라이드치킨', '닭강정', '양념치킨']
Index: 51484, Average Similarity: 0.9425684213638306, Data: ['치킨', '양념', '반반치킨', '후라이드치킨', '닭강정', '철판마늘치킨', '양념치킨', '후라이드']
Index: 52211, Average Similarity: 0.9425684213638306, Data: ['치킨', '양념', '반반치킨', '후라이드치킨', '닭강정', '철판마늘치킨', '양념치킨', '후라이드']
Index: 9274, Average Similarity: 0.9423180818557739, Data: ['매운맛양념치킨', '마늘치킨', '치킨', '후라이드치킨', '닭강정', '양념치킨']
Index: 16374, Average Similarity: 0.9423180818557739, Data: ['매운맛양념치킨', '마늘치킨', '치킨', '후라이드치킨', '닭강정', '양념치킨']
Index: 16375, Average Similarity: 0.9423180818557739, Data: ['매운맛양념치킨', '마늘치킨', '치킨', '후라이드치킨', '닭강정', '양념치킨']
Index: 48909, Average Similarity: 0.9423180818557739, Data: ['매운맛양념치킨', '마늘치킨', '치킨', '후라이드치킨', '닭강정', '양념치킨']
Index: 48910, Average Similarity: 0.9423180818557739, Data: ['매운맛양념치킨', '마늘치킨', '치킨', '후라이드치킨', '닭강정', '양념치킨']
Index: 10096, Average Similarity: 0.9413013458251953, Data: ['마늘치킨', '치킨', '후라이드치킨', '닭강정', '양념치킨', '핫칠리치킨']
Index: 10097, Average Similarity: 0.9413013458251953, Data: ['마늘치킨', '치킨', '후라이드치킨', '닭강정', '양념치킨', '핫칠리치킨']
Index: 12371, Average Similarity: 0.9413013458251953, Data: ['마늘치킨', '치킨', '후라이드치킨', '닭강정', '양념치킨', '핫칠리치킨']
Index: 12372, Average Similarity: 0.9413013458251953, Data: ['마늘치킨', '치킨', '후라이드치킨', '닭강정', '양념치킨', '핫칠리치킨']
Index: 28416, Average Similarity: 0.9413013458251953, Data: ['마늘치킨', '치킨', '후라이드치킨', '닭강정', '양념치킨', '핫칠리치킨']
Index: 28417, Average Similarity: 0.9413013458251953, Data: ['마늘치킨', '치킨', '후라이드치킨', '닭강정', '양념치킨', '핫칠리치킨']
Index: 44771, Average Similarity: 0.9413013458251953, Data: ['마늘치킨', '치킨', '후라이드치킨', '닭강정', '양념치킨', '핫칠리치킨']
Index: 44772, Average Similarity: 0.9413013458251953, Data: ['마늘치킨', '치킨', '후라이드치킨', '닭강정', '양념치킨', '핫칠리치킨']
```

리뷰 임베딩

<LSA(5차원) TF-IDF Clustering>




```python
from sklearn.metrics import silhouette_score

# K-means 모델 생성 및 훈련
kmeans = KMeans(n_clusters=4, random_state=42, init='k-means++')
cluster_labels = kmeans.fit_predict(lsa_matrix1)

# 실루엣 스코어 계산
silhouette_avg = silhouette_score(lsa_matrix1, cluster_labels)

print(f"Silhouette Score: {silhouette_avg:.4f}")
```

```
Silhouette Score: 0.3205
```

실루엣 스코어 : 0.3205

```python
word_counts = [len(doc) for doc in mm['doc_vec'].values]  # 각 문서의 단어 개수

average_word_count = np.mean(word_counts)  # 평균 단어 개수
max_word_count = np.max(word_counts)  # 최대 단어 개수
min_word_count = np.min(word_counts)  # 최소 단어 개수

# 결과 출력
print(f"Average number of words in a document: {average_word_count}")
print(f"Maximum number of words in a document: {max_word_count}")
print(f"Minimum number of words in a document: {min_word_count}")
```

```
Average number of words in a document: 80.59430030413397
Maximum number of words in a document: 394
Minimum number of words in a document: 1
```

단어의 최대 개수 : 394개
단어의 평균 개수 : 80개
단어의 최소 개수 : 1개

<TF-IDF를 이용한 키워드 추출>

```python
from sklearn.feature_extraction.text import TfidfVectorizer

# TF-IDF 벡터화를 위해 리스트를 문자열로 변환
food['doc_key_joined'] = food['doc_key'].apply(lambda x: ' '.join(x))
place['doc_key_joined'] = place['doc_key'].apply(lambda x: ' '.join(x))

# TF-IDF 벡터화
vectorizer1 = TfidfVectorizer()
vectorizer2 = TfidfVectorizer()
X1 = vectorizer1.fit_transform(food['doc_key_joined'])
X2 = vectorizer2.fit_transform(place['doc_key_joined'])

# TF-IDF 값과 단어 매핑
feature_names1 = vectorizer1.get_feature_names_out()
feature_names2 = vectorizer2.get_feature_names_out()
tfidf_values1 = X1.toarray()
tfidf_values2 = X2.toarray()

# 상위 단어 개수 설정
num_top_words = 6

# 결과를 저장할 리스트
selected_words1 = []
selected_words2 = []

# 각 문서에서 TF-IDF 값이 높은 상위 단어 선택
for doc_index, tfidf_value in enumerate(tfidf_values1):
    # 단어와 TF-IDF 값을 매핑하고 내림차순
    word_scores = {feature_names1[word_index]: score for word_index, score in enumerate(tfidf_value)}
    sorted_word_scores = sorted(word_scores.items(), key=lambda x: x[1], reverse=True)

    # 상위 단어 선택
    top_words = [word for word, score in sorted_word_scores[:num_top_words]]
    selected_words1.append(" ".join(top_words))

# 각 문서에서 TF-IDF 값이 높은 상위 단어 선택
for doc_index, tfidf_value in enumerate(tfidf_values2):
    # 단어와 TF-IDF 값을 매핑하고 내림차순
    word_scores = {feature_names2[word_index]: score for word_index, score in enumerate(tfidf_value)}
    sorted_word_scores = sorted(word_scores.items(), key=lambda x: x[1], reverse=True)

    # 상위 단어 선택
    top_words = [word for word, score in sorted_word_scores[:num_top_words]]
    selected_words2.append(" ".join(top_words))

# 데이터프레임에 결과 추가
food['doc_key'] = selected_words1
place['doc_key'] = selected_words2
```

```
0        [맛있다, 분위기, 좋다, 요즘, 날씨, 가면, 정말, 좋다, 치킨, 맛있다 마늘,...
1        [커피, 맛있다, 분위, 좋다, 친절하다 부리또, 따뜻하다, 맛있다, 많다, 샐러드.
2        [아주, 옛날, 방문, 강남, 인데, 그대로, 뭔가, 추억, 돋다, 역시, 그대, ...
3        [코다리, 조림, 매콤, 이랑, 먹다, 맛있다 음식, 짜다, 정석, 코다리, 큼직하...
4        [주말, 이나, 공휴일, 종종, 찾다, 좋다, 가성, 훌륭하다, 중국집, 오늘, 고...
```

키워드 추출

```
'쭈꾸미 잡채 맛점 혼밥 늘리다 비율',
'하노이 쌀국수 레귤러양 홍두깨 찾다 다른',
'다코야끼 닭집 가쓰오부시 산더미 기름 가운',
'나누다 적정하다 퀄리티 송년회 가기 좋다',
'쌀국수 잠실 직화 새다 짜조 베트남',
'닭갈비 홍천 볶다 쏘다 나가요 라스트',
'빠르다 친절하다 완전 맛있다 사장 가가',
'유산균 치실 서든 단체 생수 보양',
'대국 정서 로컬 최오 닥다 저혀',
```

추천 알고리즘

<추천 함수>

```python
def recommend_rows_based_on_keywords(df, model, keywords_main, keywords_sub, exclude_keywords, Islike, loc_x, loc_y):
    # 단계 2: loc_x, loc_y와의 직선 거리가 1.5km 이내인 행으로 tmp_df
    df['distance'] = df.apply(calculate_distance, args=(loc_x, loc_y), axis=1)

    # 거리가 1.5km 이내인 행만 선택
    tmp_df = df[df['distance'] <= 3].copy()

    # 단계 3: keywords_main으로 main_word_vec 생성
    main_word_vec = [model.wv[word] for word in keywords_main if word in model.wv]

    # 단계 4: Islike가 True인 경우, sim_words_list 생성 후 sim_word_vec 생성
    if Islike:
        sim_words = model.wv.most_similar(positive=keywords_main, topn=50)
        sim_words_list = [word for word, similarity in sim_words[10:15]] + [word for word, similarity in sim_words[40:45]]
        sim_word_vec = [model.wv[word] for word in sim_words_list if word in model.wv]

    # 단계 6: exclude_keywords가 빈 리스트가 아닌 경우, except_word_vec 생성
    if exclude_keywords:
        except_word_vec = [model.wv[word] for word in exclude_keywords if word in model.wv]

    # 단계 7: keywords_sub가 빈 리스트가 아닌 경우, sub_word_vec 생성
    if keywords_sub:
        sub_word_vec = [model.wv[word] for word in keywords_sub if word in model.wv]

    # 단계 9: main_word_vec과 df의 'main_vec' column 사이의 코사인 유사도 측정
    main_sim_list = tmp_df['main_vec'].apply(lambda x: calculate_average_top_similarities(main_word_vec, x))

    # 단계 10: sub_word_vec이 존재하는 경우, sub_word_vec과 df의 'doc_vec' column 사이의 코사인 유사도 측정
    if keywords_sub:
        sub_sim_list = tmp_df['doc_vec'].apply(lambda x: calculate_average_top_similarities(sub_word_vec, x))

    # 단계 11: except_word_vec이 존재하는 경우
    if exclude_keywords:
        except_sim_list_1 = tmp_df['main_vec'].apply(lambda x: calculate_average_top_similarities(except_word_vec, x))
        except_sim_list_2 = tmp_df['doc_vec'].apply(lambda x: calculate_average_top_similarities(except_word_vec, x)) if keywords_sub else [0]*len(tmp_df)

    # 단계 12: sim_word_vec이 존재하는 경우 (Islike가 True인 경우)
    if Islike:
        sim_sim_list = tmp_df['main_vec'].apply(lambda x: calculate_average_top_similarities(sim_word_vec, x))

    # 단계 13: 리스트를 정렬하고 최종 점수 계산
    tmp_df['main_similarity'] = main_sim_list
    if keywords_sub:
        tmp_df['sub_similarity'] = sub_sim_list
    if exclude_keywords:
        tmp_df['except_similarity'] = [max(e1, e2) for e1, e2 in zip(except_sim_list_1, except_sim_list_2)]
    if Islike:
        tmp_df['sim_similarity'] = sim_sim_list

    # 최종 점수 계산
    if keywords_sub and exclude_keywords and Islike:
        tmp_df['final_score'] = tmp_df['main_similarity'] + tmp_df['sub_similarity'] - tmp_df['except_similarity'] + tmp_df['sim_similarity']
    elif keywords_sub and exclude_keywords:
        tmp_df['final_score'] = tmp_df['main_similarity'] + tmp_df['sub_similarity'] - tmp_df['except_similarity']
    elif keywords_sub and Islike:
        tmp_df['final_score'] = tmp_df['main_similarity'] + tmp_df['sub_similarity'] + tmp_df['sim_similarity']
    elif exclude_keywords and Islike:
        tmp_df['final_score'] = tmp_df['main_similarity'] - tmp_df['except_similarity'] + tmp_df['sim_similarity']
    elif keywords_sub:
        tmp_df['final_score'] = tmp_df['main_similarity'] + tmp_df['sub_similarity']
    elif exclude_keywords:
        tmp_df['final_score'] = tmp_df['main_similarity'] - tmp_df['except_similarity']
    elif Islike:
        tmp_df['final_score'] = tmp_df['main_similarity'] + tmp_df['sim_similarity']
    else:
        tmp_df['final_score'] = tmp_df['main_similarity']

    # 단계 14, 15: 최종 추천
    result_df = tmp_df.sort_values(by='final_score', ascending=False)

    # 최종 추천된 데이터프레임 반환
    return result_df.head(10)
```

키워드 추출 함수에서의 인자를 기반으로 벡터화

좌표(직선거리)를 기준으로 데이터 프레임을 분할

케이스에 맞게 전달받은 (main,sub) keyword 및 Islike, exclude_keywords를 임베딩

데이터 프레임 내의 벡터과의 코사인 유사도 중 상위 5개의 평균 계산 및 정렬

각 상황에 맞게 설정된 main keyword를 제외한 나머지 3개 변수의 조합(3!)의 케이스별 코사인 유사도 기반 Scoring

최종 Score를 기준으로 추천 항목을 반환

- main key : 명사, 핵심 키워드
- sub key : 부사, 형용사 보조 키워드
- exclude key : 추천 항목에서 제외할 키워드
- similar key : 핵심 키워드로부터 40~50번 정도쯤 떨어진 키워드(다양성)
- Islike : 문장에 '~와 같은', '~비슷한' 여부

추천 알고리즘

'한식류가 먹고 싶어. 감자탕은 빼고' (지역 : 강남역 주변)

```
문장을 입력하세요: 한식류가 먹고 싶어. 감자탕은 빼고
추출된 주요 키워드 : ['한식']
추출된 추상 키워드 (형용사, 부사) : []
제외 키워드 : ['감자탕']
Islike : False
```

```
[99]
rec = recommend_rows_based_on_keywords(df=food, model=model,
rec[['장소명','main_similarity']].head(5)
```

상위 5개 추천리스트

	장소명	main_similarity
20742	진심우	1.0
8688	나비식탁	1.0
15849	순천만	1.0
6192	하이들	1.0
6066	유연정	1.0

```
rec[['장소명','except_similarity']].tail(5)
```

하위 5개 추천리스트

	장소명	except_similarity
19619	한남동24시뼈다귀감자탕 본점	0.878515
6712	명가뼈다귀감자탕	0.892127
11614	원당뼈다귀감자탕	0.838773
10388	원조감자탕일미집 역삼점	0.934834
20120	원조감자탕 일미집	0.934834

'독특한 파스타가 먹고 싶어' (지역 : 강남역 주변)

```
문장을 입력하세요: 독특한 파스타가 먹고 싶어
추출된 주요 키워드 : ['파스타']
추출된 추상 키워드 (형용사, 부사) : ['독특하다']
제외 키워드 : []
Islike : False
```

```
rec = recommend_rows_based_on_keywords(df=food, model=model, keywords_main=keywords_main, keywords_sub=keywords_sub, e
rec[['장소명','대표메뉴','sub_similarity']].head(5)
```

	장소명	대표메뉴	sub_similarity
20487	시엘	앤초비 오일 파스타, 한우 채끝등심 스테이크, 이탈리안 전통 토마토 스프, 까르보나라	0.527335
14637	그랑씨엘	앤초비 오일 파스타, 한우 채끝등심 스테이크, 이탈리안 전통 토마토 스프, 까르보나라	0.527335
20823	아이수마(AiSuma)	파스타리조또	0.514662
16574	파스토보이 역삼점	강남구 맛집랭킹 위 베이컨 까르보나라, 김치 필라프, 목살 스테이크 샐러드, 매콤 ...	0.526296
17307	파티오42	랍스타 로제 파스타, 아보카도와 스테이크 피자, 성게알 파스타, 스테이크 트러플 크...	0.509773

```
rec[['장소명','대표메뉴','except_similarity']].tail(5)
```

	장소명	대표메뉴	except_similarity
26603	루이스카페	갈릭브런치, 루이스브런치, 바나나브런치, 미숫가루	0.771696
26618	라미띠에		0.763901
26621	마라공방 역삼점	마라탕 , 마라반 , 마라샹궈 , 꿔바로우	0.819523
26627	청년다방 선릉역점	차돌떡볶이, 차돌떡볶이, 통큰오짱떡볶이, 통큰오짱떡볶이	0.879410
26632	오라방	돔베고기 , 갈치튀김, 제주창도름순대, 고사리육개장	0.866327

챗봇

Low-Level ChatBot

```python
import re

# 추천 결과
food_final = None
cafe_final = None
place_final = None
bar_final = None

def extract_stations(text):
    # "00역" 또는 "00 00역"을 찾는 정규 표현식
    pattern = re.compile(r'\b\w+(?:\s\w+)?역\b')

    # text에서 패턴에 일치하는 문자열 탐색
    matches = pattern.findall(text)

    # 띄어쓰기 제거 후 일치하는 문자열 반환
    stations = [match.replace(" ", "") for match in matches]

    # 일치하는 문자열 반환
    return stations[0]

def x_check(usrInput):
    if (usrInput == 'X' or usrInput == 'x'):
        return

def extract_integers(text):
    pattern = re.compile(r'\d+')  # 정수를 찾는 정규 표현식
    integers = pattern.findall(text)  # 문자열에서 정수 추출
    return [int(num) for num in integers]  # 추출된 정수들을 정수형으로 변환하여 반환

print("안녕하세요! 멋진 하루를 보낼 수 있도록 도와드리겠습니다!\n")
print("오늘 어디를 방문하실 계획인가요? 방문 계획이 있다면 인근 지하철 역을 말씀해주시겠어요? \n(지하철 역 입력을 원하지 않으시면 X를 입력해주세요)\n")
user_loc = input()
if (user_loc == 'X' or user_loc == 'x'):
    user_loc = extract_stations('강남역')
else:
    user_loc = extract_stations(user_loc)


print("\n감사합니다! 그럼 이제 일정을 세워보겠습니다!")
print("금강산도 식후경! 우선 밥을 먹는 것은 어떨까요? 드시고 싶은 음식의 느낌을 자유롭게 적어주세요. \n(식당 추천을 원하지 않으시면 X를 입력해주세요)\n")

food_sentence = input()
if (food_sentence == 'X' or food_sentence == 'x'):
    pass
else:
    keywords_main, keywords_sub, exclude_keywords, Islike = extract_keywords(food_sentence)
    food_rec = recommend_rows_based_on_keywords(df=food, model=model, keywords_main=keywords_main, keywords_sub=keywords_sub, exclude_keywords=exclude_keywords, Islike=Islike, \
                                                loc_x=metro[metro['역명']==user_loc]['위도'].values[0], loc_y=metro[metro['역명']==user_loc]['경도'].values[0])

    print("\n말씀해주신 결과를 바탕으로 다음과 같은 식당들을 찾아보았습니다! 마음에 드는 번호를 입력해주세요\n")
```

```python
        menus = row['대표메뉴']
        print(f"{rec}: {row['장소명']} / 대표메뉴: {menus} / 주소: {row['주소']}")
```

```
안녕하세요! 멋진 하루를 보낼 수 있도록 도와드리겠습니다!

오늘 어디를 방문하실 계획인가요? 방문 계획이 있다면 인근 지하철 역을 말씀해주시겠어요?
(지하철 역 입력을 원하지 않으시면 X를 입력해주세요)

잠실역
```

<-실행결과

인근 지하철 역의 주소 질문 후
사용자 응답에 따라 초기 좌표 설정

1 Block

- 식당에 대한 질문 / 사용자 응답
- 추천 결과 출력 /사용자 응답
- 추천 결과 저장

카페에 대한 질문 / 추천 결과 저장

장소에 대한 질문 / 추천 결과 저장

술집에 대한 질문 / 추천 결과 저장

일정 추천

챗봇

안녕하세요! 멋진 하루를 보낼 수 있도록 도와드리겠습니다!

오늘 어디를 방문하실 계획인가요? 방문 계획이 있다면 언근 지하철 역을 말씀해주시겠어요?
(지하철 역 입력을 원하지 않으시면 X를 입력해주세요)

잠실역

감사합니다! 그럼 이제 질문을 세워보겠습니다!
롯데월드 식후경! 우선 밥을 먹는 것은 어떨까요? 도시고 싶은 음식의 느낌을 자유롭게 적어주세요.
(식당 추천을 원하지 않으시면 X를 입력해주세요)

따뜻하고 든든한 음식 없을까? 양식이나 한식같은거

말씀해주신 결과를 바탕으로 다음과 같은 식당들을 찾아보았습니다! 마음에 드는 번호를 입력해주세요

[1번] 장소명: 빕 KEP, 대표메뉴: [' 보쌈정식', ' 비프 에그 베네딕트 ']
[2번] 장소명: 릭스토랑파스타, 대표메뉴: [' 수비드 바질 스테이크', ' 홈메크림파스타']
[3번] 장소명: 오로라, 대표메뉴: [' 돈까스', ' 함박스테이크']
[4번] 장소명: 파스타집이야 송파점, 대표메뉴: [' 해물의 변화무쌍 파스타', ' 홍운화 와소인마이볼 오늘내일 ']
[5번] 장소명: 릭스트로 에너토, 대표메뉴: ['라멜빵', ' 오늘의 스프 ']
[6번] 장소명: greasy house, 대표메뉴: ['갈릭 멜버그 스테이크', ' 슬로피조 멜버그 스테이크']
[7번] 장소명: 차안다, 대표메뉴: ['한 여름철의 꿈 새우', '랑고']
[8번] 장소명: 엘, 대표메뉴: [' ', ' ']
[9번] 장소명: 미조미니멀리한 감성, 대표메뉴: [' ']
[10번] 장소명: 하트담빕 파스타, 대표메뉴: [' ']
4번

느낌있는 카페는 어떤신가요? 카페에 대해 자유롭게 적어주세요
(카페 추천을 원하지 않으시면 X를 입력해주세요)

디저트가 괜찮고 조용한 카페 추천해줘.

말씀해주신 결과를 바탕으로 다음과 같은 카페들을 찾아보았습니다! 마음에 드는 번호를 입력해주세요

[1번] 장소명: 케키하우스, 대표메뉴: [' 보틀케이크미디엄', ' 옥수수당']
[2번] 장소명: 엘스카페, 대표메뉴: [' 얼그레이 쉬폰케이크 호', ' 얼그레이 쉬폰케이크 호']
[3번] 장소명: 빽다방 송파하이오점, 대표메뉴: [' 바나나밀크쉐이크 ', ' 아메리카노']
[4번] 장소명: 빽다방 오금역점, 대표메뉴: [' 바나나밀크쉐이크 ', ' 아메리카노']
[5번] 장소명: Pet a cake(펫어케이크), 대표메뉴: [' 당질초케이크 ', ' 조각호돔츠케이크']
[6번] 장소명: 커피 앤비언스, 대표메뉴: [' 아메리카노 ', ' 에스프레소']
[7번] 장소명: 스테이 커피, 대표메뉴: [' 아메리카노 ', ' 카페라떼']
[8번] 장소명: 하라랜즈 커피, 대표메뉴: [' 에스프레소 ', ' 아메리카노']
[9번] 장소명: 돌츠커피, 대표메뉴: [' 아메리카노 ', ' 바닐라라떼 ']
[10번] 장소명: 마린커피, 대표메뉴: [' 아메리카노 ', ' 더치연유라떼']
1번

제미있는 엔터비티나 문화생활을 즐겨보실래요? 가고 싶은 장소에 대한 느낌을 자유롭게 적어주세요
(장소 추천을 원하지 않으시면 X를 입력해주세요)

밖은 분위기에서 스트레스를 풀고 싶어

말씀해주신 결과를 바탕으로 다음과 같은 장소들을 찾아보았습니다! 마음에 드는 번호를 입력해주세요

[1번] 장소명: 롯데월드 아이스링크, 업종: 아이스링크
[2번] 장소명: 송파어린이영어도서관, 업종: 어린이도서관
[3번] 장소명: 롯데월드 시네마, 업종: 아할리아음식
[4번] 장소명: 사각사각 플레이스, 업종: 문화시설
[5번] 장소명: 서울시농수산식품공사 가락몰 쿠킹스튜디오, 업종:
[6번] 장소명: 롯데월드 가든스테이지, 업종:
[7번] 장소명: 강남스포츠문화센터, 업종: 스포츠센터
[8번] 장소명: 케이홈 갤러리 , 업종: 갤러리,화랑
[9번] 장소명: 롯데월드 아쿠아리움, 업종: 아쿠아리움
[10번] 장소명: 롯데월드 어드벤처, 업종: 테마파크
10번

마지막으로 아직 아쉽다면 마무리로 술집은 어떤신가요? 가고 싶은 술집에 대해 자유롭게 적어주세요
(술집 추천을 원하지 않으시면 X를 입력해주세요)

조금 촐촐할거같은데, 적당한 안주가 있으면 좋겠다. 칵테일도 괜찮아

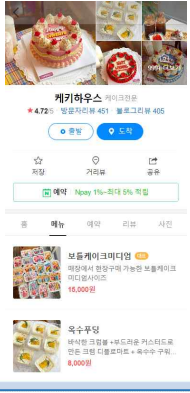
말씀해주신 결과를 바탕으로 다음과 같은 장소들을 찾아보았습니다! 마음에 드는 번호를 입력해주세요

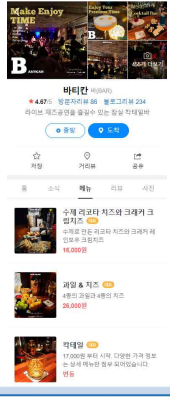
[1번] 장소명: 바타칸, 대표메뉴: ['수제 리코타 치즈와 크래커 크림치즈', ' 과일 치즈']
[2번] 장소명: 구지라운지(9G LOUNGE), 대표메뉴: ['멕시칸치즈나초', ' 시사']
[3번] 장소명: 세컨드 스테이지, 대표메뉴: [' 르로슈토 멜론', ' 패피로니피자']
[4번] 장소명: 역전할머니맥주 가락시장역점, 대표메뉴: [' ', ' 바지락 순두부']
[5번] 장소명: 코코라운지, 대표메뉴: ['투움바 파스타', ' 시그니쳐 칵테일']
[6번] 장소명: 맥주어클락, 대표메뉴: ['왕 새우 튀김 땅콩', ' 쉬림프 갈릭 알리오 올리오']
[7번] 장소명: 만주, 대표메뉴: [' 특 사시미 ', ' 모둠 사시미']
[8번] 장소명: 스위댄 피크닉, 대표메뉴: [' 육상겨 론계 떡볶이', ' 감바스 파스타']
[9번] 장소명: K.PUB (케이펍)스테이크와 치킨, 대표메뉴: ['후라이드 치킨', ' 반반 치킨']
[10번] 장소명: 케이펍, 대표메뉴: ['후라이드 치킨', ' 반반 치킨']
1번

챗봇

감사합니다~ 오늘 일정에 대해 정리해드리겠습니다!

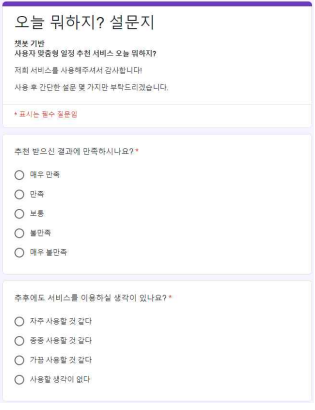
식당: 파스타집이야 송파점 / 대표메뉴: 해물비 벤트마제 파스타, 홍윤화 와츠인마이룸 오늘내일 , 새우누룰지 알리오올리오, 숯불샐러드 파스타 / 주소: 서울특별시 송파구 오금로15길 9-13, 지하1,지상1,2층 (방이동)
카페: 케키하우스 / 대표메뉴: 보틈케이크미디엄, 옥수푸딩, 아메리카노, 우유생크림 도시락케이크 / 주소: 서울특별시 송파구 백제고분로41길 35, 2층 201호 (송파동, 지건)
장소: 롯데월드 어드벤처 / 주소: 송파구 올림픽로 240
술집: 바티칸 / 대표메뉴: 수제 리코타 치즈와 크래커 크림치즈, 과일 치즈, 라데일, 쿠키 크림이 가득 올라간 아이스크림 / 주소: 서울특별시 송파구 올림픽로32길 22-9, 지하1층 (방이동)





챗봇

Simple MOS



오늘 뭐하지? 설문지

챗봇 기반
사용자 맞춤형 일정 추천 서비스 오늘 뭐하지?

저희 서비스를 사용해주셔서 감사합니다!

사용 후 간단한 설문 몇 가지만 부탁드리겠습니다.

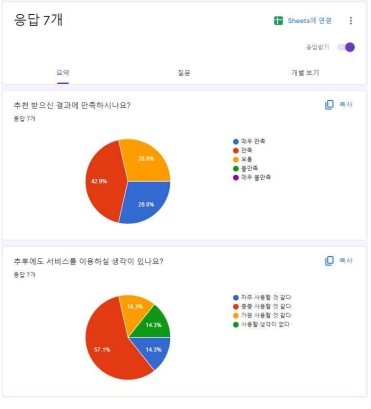
* 표시는 필수 질문임

추천 받으신 결과에 만족하시나요? *

- 매우 만족
- 만족
- 보통
- 불만족
- 매우 불만족

추후에도 서비스를 이용하실 생각이 있나요? *

- 자주 사용할 것 같다
- 종종 사용할 것 같다
- 가끔 사용할 것 같다
- 사용할 생각이 없다



기여점 및 한계점

프로젝트 기여점

- 거리 정보가 반영된 실용적인 일정 구성 자체의 설계
- 추상적인 키워드를 고려한 다양한 추천
- FastText 기반의 자유로운 임베딩
- 대화 형식으로, 생각나는 정보를 모두 검색 조건으로 설정 가능

프로젝트 한계점

- 데이터셋의 부족
- 추천 과정에서의 많은 연산량
- 키워드 도출에 예외 존재
- 챗봇의 완성도 및 접근성
- 아직은 완벽하지 않은 정확도

Q&A