

# 관광 HotSpot 빅데이터 분석 웹서비스



황세진

이성호

정지용

지도교수 홍봉희

---

## 목 차

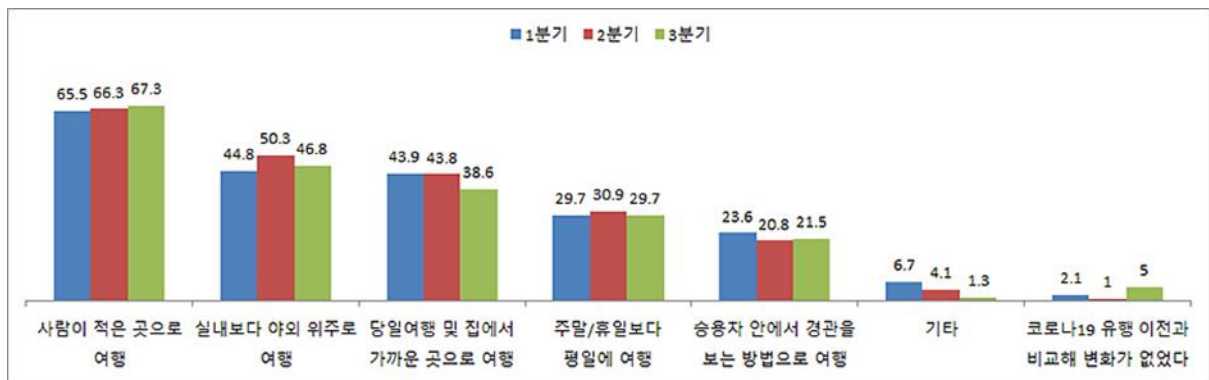
1. 과제 배경 및 목표	1
1.1. 과제 배경	1
1.2. 과제 목표	2
2. 설계 과정	3
2.1. 요구조건	3
2.2. 기존 제약사항에 대한 수정사항	3
2.3. 시스템 구성	4
2.4. 개발 환경	5
3. 연구 내용	6
3.1. 여행지 추천 모델	6
3.1.1. 데이터 수집	6
3.1.2. 감성 분석	14
3.2. 트렌드 시각화	31
3.2.1 데이터 수집	31
3.3. 웹페이지 구성	36
4. 연구결과	37
	37
5. 결론 및 향후 연구 방향	45
5.1. 결론	45
5.2. 향후 연구 방향	45
6. 참고 문헌	46

# 1.과제 배경 및 목표

## 1.1. 과제 배경

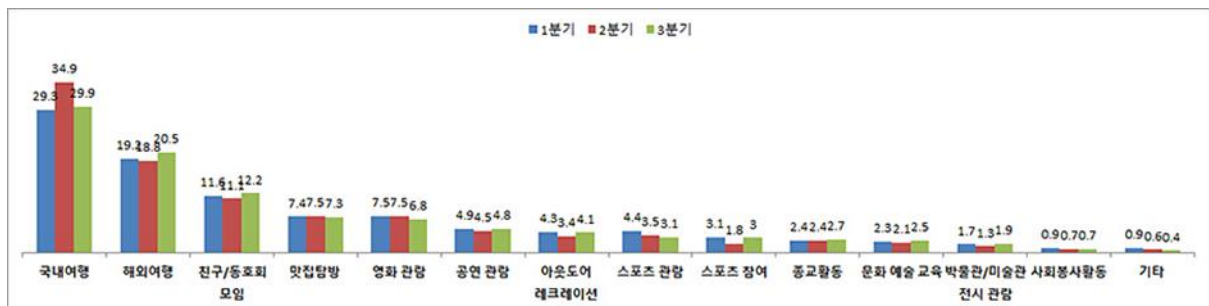
포스트 코로나 시대에 여행의 수요가 크게 늘 것으로 예상되고 관광 활동의 개별화와 소규모화가 가속화될 것으로 전망된다. 코로나19로 인해 타인과의 접촉이 많은 단체관광보다는 상호 안전을 신뢰할 수 있는 소규모 개별관광이나 자유여행을 선호하는 경향이 더욱 강해질 것으로 보인다. 더 나아가 코로나19의 영향으로 관광산업의 디지털 전환 요구가 증가되면서 관광시장에서 초개인화 기술을 활용한 개인 맞춤형 관광상품 개발 수요가 커질 것으로 전망된다.

### -국내 여행 행동 변화



코로나19로 인해 해외관광이 사실상 단절되면서 국내관광에 대한 관심이 증대되었다. 이에 '해외관광 대체형 시장'으로서 국내관광의 성장을 바라볼 수도 있을 것이다. 그러나 코로나19가 관광분야에 미친 막대한 피해 상황 속에서도 국내관광의 매력 재발견이라는 긍정적 영향을 끼쳤다는 점에서 포스트 코로나 시대에도 국내관광의 수요 증가는 계속해서 이어질 것으로 보인다.

### -현재 가장 하고 싶은 여가활동



---

## 1.2. 과제 목표

포스트 코로나 시대에 여행에 대한 수요가 늘고 관광 활동의 개별화와 소규모화가 가속화되고, 국내관광의 수요가 증가함에 따라 이에 맞는 빅데이터를 통한 부산 관광 hotspot 웹서비스로 주제를 선정하였다.

---

## 2. 설계 과정

### 2.1. 요구조건

#### 2.1.1. 데이터 수집

데이터 수집은 구글 지도, 네이버 지도, 카카오맵 3가지 플랫폼 중 카카오맵을 선택하여 관광지 정보와 댓글을 수집하였다.

댓글은 사용자 이름, 댓글 개수, 평균 별점, 작성 날짜, 별점, 댓글 내용으로 구성되고, 약 5만개의 관광지, 음식점 정보와 약 13만개의 댓글을 수집했다.

#### 2.1.2. 데이터 전처리

데이터 수집 단계에서 수집 항목에 대해 기재사항이 없는 경우 numpy의 NaN으로 처리하였다.

타겟 데이터는 댓글 평점에서 4,5점을 긍정인 1로 1,2,3을 부정인 0으로 라벨링하였고, 입력 데이터는 댓글에서 가장 많이 쓰는 단어 500개를 선정해 댓글들을 명사 단위로 나눠서 각각의 단어를 라벨링 하였고 없는 단어는 1로 패딩은 0으로 처리하였다.

#### 2.1.3. 자연어 처리

한글 자연어 처리를 위해 KoNLPy 파이썬 패키지를 사용해 댓글을 명사 단위로 나눠서 댓글을 분석하였다.

### 2.2. 기존 제약사항에 대한 수정사항

#### 2.2.1. 전국에서 부산으로 지역을 한정

전국을 대상으로 과제를 진행하려 했으나, 부산을 대상으로 과제를 진행하였다.

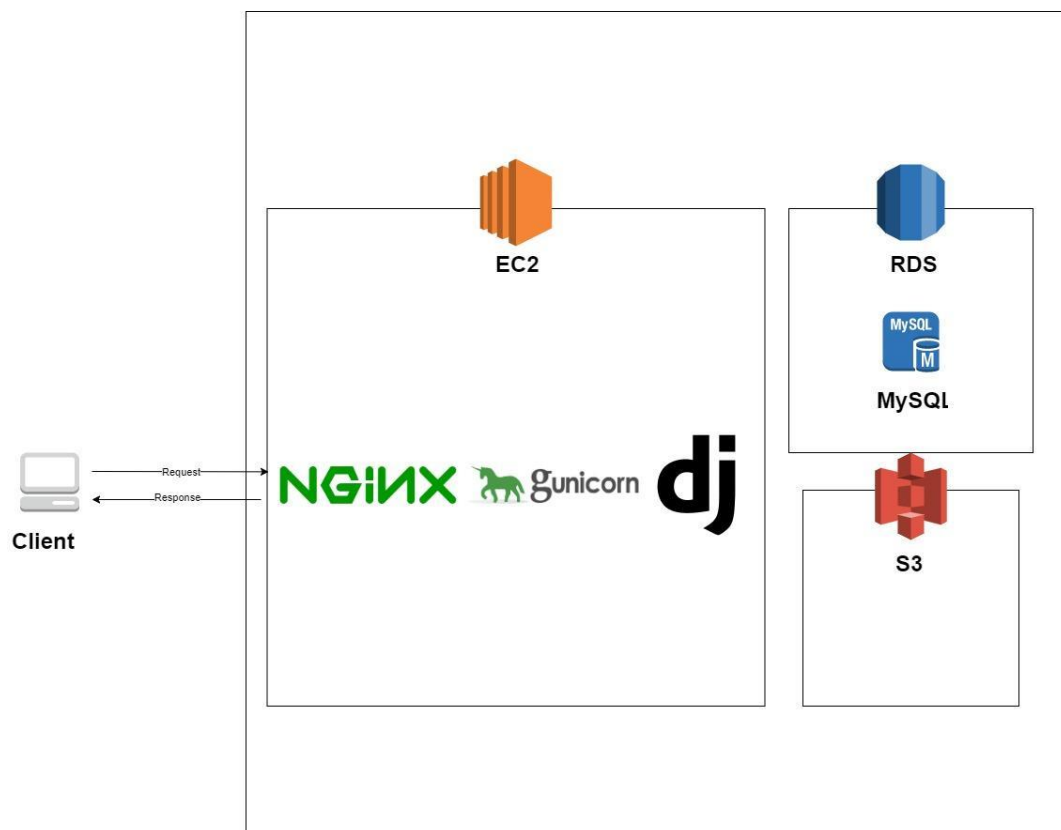
#### 2.2.2. 당일치기 코스 추천 기능 제외

사용자의 취향에 맞는 당일치기 코스를 추천하는 기능을 기획했으나, 댓글에 대한 감성분석, 추세분석만 진행하게 되어 해당 기능은 제외하였다. 이는 사용자가 출발지, 최대 5개의 경유지, 도착지를 설정하면 해당 코스의 경로를 표시해주는 기능으로 대체하였다.

### 2.2.3. 관광지에 대한 이미지 제공이 어려움

한국관광공사 무료 이미지, TourAPI 4.0 등의 공공 웹사이트를 통해 관광지 전체에 대한 이미지를 제공하려 했으나, DB에 저장된 관광지명과 달라 검색이 안되거나 해당 관광지에 대한 이미지를 제공하지 않는 경우가 많았다. 이는 무료 이미지 사이트를 통해 상위 8개의 관광지에 대한 사진을 웹사이트 메인에 보여주는 것으로 축소하였다.

## 2.3. 시스템 구성



---

## 2.4. 개발 환경

### 2.4.1. 개발 언어

백엔드 프레임워크와 크롤링, 자연어처리 모두 Python을 사용하였다.

### 2.4.2. 개발 도구

백엔드 프레임워크는 Django, 크롤링에는 Selenium, BeautifulSoup, 문장으로 이루어진 댓글을 단어 단위로 토큰화하는 작업에 koNLPy, 모델 학습은 keras를 사용하였다.

### 2.4.3. 데이터 베이스

AWS의 RDS 서비스를 통해 MySQL를 이용하였다.

---

### 3. 연구 내용

#### 3.1. 여행지 추천 모델

카카오맵을 크롤링해 수집한 데이터를 이용해 최근 3개월 간 작성된 리뷰들을 모아서 감성 분석한 결과와 원래 평점들을 곱해서 해당 장소에 대한 점수를 매긴다. 점수순으로 상위 10개를 뽑아서 여행지를 추천해준다.

##### 3.1.1. 데이터 수집

카카오맵에서 검색, 상세 정보를 보기위한 페이지 이동에 Selenium을 사용, html 파싱에는 BeautifulSoup을 사용하였다.

카카오맵을 통해 부산 XX구 관광지 또는 부산 XX구 YY동 맛집으로 검색하여 관광지와 음식점에 대한 상세정보, 댓글을 수집하였다.

수집한 관광지와 댓글을 담을 dataframe은 다음과 같다.

장소는 name, category, address, operation\_time, homepage, tel, tag, etc, facility, num\_of\_comments, avg\_rate, url 12개의 칼럼으로 구성된다.

댓글은 place\_name, nickname, total\_comments, avg\_rating, rating, comment, created\_at 7개의 칼럼으로 구성된다.

```
place_df = pd.DataFrame(columns=['name', 'category', 'address', 'operation_time', 'homepage',  
                                'tel', 'tag', 'etc', 'facility', 'num_of_comments', 'avg_rate', 'url'])  
  
comment_df = pd.DataFrame(columns=['place_name', 'nickname', 'total_comments', 'avg_rating',  
                                   'rating', 'comment', 'created_at'])
```



https://map.kakao.com/ 으로 이동 후, 부산의 구를 담은 리스트를 순회하며 검색어를 바꾸어준다.

```
busan_gu = ["중구", "서구", "동구", "영도구", "부산진구", "동래구", "남구", "북구", "해운대구",  
            "사하구", "금정구", "강서구", "연제구", "수영구", "사상구", "기장군"]
```

```
for gu in busan_gu:  
    scraped_page = 0  
    WebDriverWait(driver, 10).until(EC.presence_of_all_elements_located((By.CSS_SELECTOR, "#search")))  
  
    # 부산 XX구 관광지 검색  
    driver.find_element_by_xpath('//*[@id="search.keyword.query"]').clear()  
    driver.find_element_by_xpath('//*[@id="search.keyword.query"]').send_keys("부산 " + gu + " 관광지")  
    driver.find_element_by_xpath('//*[@id="search.keyword.submit"]').send_keys(Keys.ENTER)
```

카카오맵은 검색 결과를 한 페이지에 최대 15개씩 보여준다. 마지막 페이지가 포함된 경우, 5개의 페이지 목록 중 마지막 페이지 이후 페이지는 element class에 HIDDEN이 들어가 보이지 않으므로 HIDDEN이 들어간 페이지를 만날 시 다음 구를 검색하도록 하였다.



```
# 검색결과 페이지 1,2,3,4,5 중 hidden인 경우 마지막 페이지  
for search_page in range(1, 6):  
    if search_page != 1:  
        if "HIDDEN" in driver.find_element_by_xpath('//*[@id="info.search.page.no' + str(search_page) + '"]').get_attribute("class"):  
            break  
        else:  
            driver.find_element_by_xpath('//*[@id="info.search.page.no' + str(search_page) + '"]').send_keys(Keys.ENTER)
```

검색 결과 페이지 이동 후, 15개의 검색 결과와 1개의 광고가 담긴 리스트가 모두 위치한 상태에서 BeautifulSoup을 이용해 해당 페이지의 html을 파싱한다.

파싱 결과에서 15개의 검색 결과와 1개의 광고가 담긴 리스트를 places에 담는다.

```
html = driver.page_source  
soup = BeautifulSoup(html, 'html.parser')  
  
# 15개의 관광지 목록 + 광고1개  
places = soup.select('.placelist > li')
```

리스트의 각 아이템은 아래와 같이 이름, 카테고리, 리뷰 수와 평점, 주소, 상세 설명을 보기 위한 url을 포함하고 있다.

A 전포1배수지 저수지

0.0 ★★★★★ 0건 | 리뷰 0

부산 부산진구 전포동 10-17

[상세보기](#)



총 16개의 아이템을 순회하며 광고인 경우, 이미 수집한 경우, 주소지가 부산이 아닌 경우는 정보를 스크랩하지 않았다.

```
# 15개의 관광지 목록 상세정보 클릭
for i, p in enumerate(places):
    try:
        driver.find_element_by_xpath('//*[@id="info.search.place.list"]/li[' + str(i + 1) + ']')
    except NoSuchElementException:
        break

    # 광고 pass
    if "AdItem" in driver.find_element_by_xpath('//*[@id="info.search.place.list"]/li[' + str(i + 1) + ']').get_attribute("class"):
        continue

    # 리뷰(댓글) 0개이면 pass
    if int(driver.find_element_by_xpath('//*[@id="info.search.place.list"]/li[' + str(i + 1) + ']/div[4]/a/em').text.replace(",", "")) == 0:
        print(driver.find_element_by_xpath('//*[@id="info.search.place.list"]/li[' + str(i + 1) + ']/div[3]/strong/a[2]').text + "리뷰 0개")
        continue

    # 같은 이름의 관광지가 있을 시, 리뷰와 정보를 스크랩하지 않고 continue
    p_name = driver.find_element_by_xpath('//*[@id="info.search.place.list"]/li[' + str(i + 1) + ']/div[3]/strong/a[2]').text
    if not place_df[place_df["name"] == p_name].empty:
        sleep(0.8)
        continue

    # 주소가 부산이 아니면 continue
    p_address = driver.find_element_by_xpath('//*[@id="info.search.place.list"]/li[' + str(i + 1) + ']/div[5]/div[2]/p[1]').text
    if p_address != "" and not p_address.startswith("부산"):
        sleep(0.8)
        continue
```

위 조건을 제외한 경우, 상세보기 url을 통해 이동 후 로딩이 완료되면 BeautifulSoup을 이용해 상세 페이지의 html을 파싱한다.

```
# 상세정보 페이지로 탭 전환
driver.find_element_by_xpath('//*[@id="info.search.place.list"]/li[' + str(i + 1) + ']/div[5]/div[4]/a[1]').send_keys(Keys.ENTER)
driver.switch_to.window(driver.window_handles[-1])

WebDriverWait(driver, 5).until(EC.presence_of_all_elements_located((By.XPATH, '//*[@id="mArticle"]/div[1]')))

detail_page = driver.page_source
detail_page_soup = BeautifulSoup(detail_page, 'html.parser')
```

파싱 결과를 이용해 장소 이름(name), 카테고리(category), 주소(address), 영업시간(operation\_time), 홈페이지(homepage), 대표번호(tel), 태그(tag), 예약, 포장, 배달과 같은 기타사항(etc), 시설정보(facility), 리뷰 수(num\_of\_comments)와 평균 평점(avg\_rate), 상세 정보 페이지의 주소(url)을 수집한다.

(괄호 안의 내용은 dataframe의 칼럼명을 작성한 것이다.)

```
# 장소 이름
place_name = detail_page_soup.select('.place_details > .inner_place > .tit_location')
place_name = place_name[0].text if len(place_name) != 0 else np.nan

# 카테고리
place_category = detail_page_soup.select('.place_details > .inner_place > .location_evaluation > .txt_location')
place_category = place_category[0].text[4:] if len(place_category) != 0 else np.nan
place_category = place_category.split(",")

# 주소
place_address = detail_page_soup.select('.placeinfo_default > .location_detail > .txt_address')
place_address = place_address[0].text.replace('\n', ' ') if len(place_address) != 0 else np.nan

operation_time_list = detail_page_soup.select('.openhour_wrap > .location_present')
if len(operation_time_list) != 0:
    for elem in operation_time_list:
        splitted_txt = elem.text.replace("영업중", "").replace("금일영업마감", "")
        splitted_txt = ' '.join(splitted_txt.split())
        place_operation_time.append(splitted_txt)

# 홈페이지
place_homepage = detail_page_soup.select('.placeinfo_homepage > .location_detail > .location_present > a')
place_homepage = place_homepage[0].attrs.get("href") if len(place_homepage) != 0 else np.nan

# 대표번호
place_tel = detail_page_soup.select('.placeinfo_contact > .location_detail > .location_present > .num_contact > .txt_contact')
place_tel = place_tel[0].text if len(place_tel) != 0 else np.nan

# 태그
place_tags = detail_page_soup.select('.placeinfo_default > .location_detail > .txt_tag > .tag_g > a')
place_tags = [i.text for i in place_tags] if len(place_tags) != 0 else np.nan

# 예약, 포장, 배달
place_etc = ""
if detail_page_soup.find("span", "ico_delivery") is not None:
    place_etc = detail_page_soup.find("span", "ico_delivery").parent.parent
    place_etc = place_etc.select('.location_detail')[0].text

# 시설
place_facility = []
facilities = detail_page_soup.select(".placeinfo_facility")

if len(facilities) != 0:
    facilities = detail_page_soup.select(".placeinfo_facility > .list_facility > li")
    for li in facilities:
        place_facility.append(li.text.replace("\n", ""))
```

```
# 리뷰 수
total_evaluation = detail_page_soup.select('.total_evaluation > span')
total_evaluation = total_evaluation[0].text if len(total_evaluation) != 0 else np.nan

# 평균 평점
average_rate = detail_page_soup.select('.grade_star > em')
average_rate = average_rate[0].text if len(average_rate) != 0 else np.nan
```

댓글은 4개 이상일 시, 기본적으로 3개를 보여주고 '후기 더보기' 버튼을 클릭하여 5개씩 더 볼 수 있다.

더 이상 불러올 댓글이 없을 때까지 '후기 더보기' 버튼을 클릭하여 해당 장소에 대한 모든 댓글을 불러온다.

beautifulsoup을 이용해 파싱하고, 댓글 리스트를 순회하며 작성자 닉네임(nickname), 작성자가 작성한 후기 개수(total\_comments), 작성자가 부여한 평균 평점(avg\_rating), 후기 작성 날짜(created\_at), 해당 장소에 부여한 평점(rating), 댓글 내용(comment)을 수집하였다. (관광지와 마찬가지로 괄호 안의 내용은 dataframe의 컬럼명을 작성한 것이다.)

```
if len(comments) != 0:
    detail_page = driver.page_source
    detail_page_soup = BeautifulSoup(detail_page, 'html.parser')
    comments = detail_page_soup.select('.list_evaluation > li')
    for i, comment in enumerate(comments):
        txt_desc = comment.select('.txt_desc')
        nickname = comment.select('.link_user')[0].text
        total_comment = txt_desc[0].text
        avg_rating = txt_desc[1].text
        created_at = comment.select('.time_write')[0].text
        rating = int(comment.select('.inner_star')[0]['style'][6:-2]) / 20
        comment_txt = comment.select('.comment_info > .txt_comment > span')[0].text
```

음식점의 경우 구와 동을 딕셔너리로 구성하여 검색어를 바꾸어 가며 수집하였다. 수집한 결과는 관광지와 같은 칼럼명으로 구성된 dataframe에 저장하였다.

```
busan_region = {
    "중구"      : ["중앙동", "동래동", "대천동", "보수동", "부평동", "광복동", "남포동", "영주제1동", "영주제2동"],
    "동구"      : ["동래제1동", "동래제2동", "동대신제3동", "가대신제1동", "사대신제3동", "사대신제4동", "부민동", "아이동", "초량동", "송무동", "남부민제1동", "남부민제2동", "염남동"],
    "서구"      : ["영주제1동", "영주제2동", "중앙제3동", "중앙제4동", "수강제4동", "수강제5동", "천전동", "범일제2동", "범일제5동"],
    "영도구"     : ["남포동", "영선1동", "영선2동", "신선동", "홍제1동", "홍제2동", "영학1동", "영학2동", "동삼1동", "동삼2동", "송정3동", "송정4동"],
    "남산진구"  : ["부전제1동", "부전제2동", "영진동", "영정제1동", "영정제2동", "건포제1동", "부평제1동", "부평제2동", "당감제1동", "당감제2동", "당감제3동", "당감제4동", "가이제1동", "가이제2동"],
    "동래구"    : ["수민동", "북산동", "현동", "오천제1동", "오천제2동", "오천제3동", "사직제1동", "사직제2동", "사직제3동", "연락제1동", "영랑제2동", "영랑제3동", "영랑제4동", "가이제1동", "가이제2동"],
    "합천구"    : ["대곡제1동", "대곡제2동", "대곡제3동", "대곡제4동", "대곡제5동", "유호제1동", "유호제2동", "유호제3동", "유호제4동", "유호제5동", "감전제1동", "감전제2동", "유봉동", "문현제1동", "문현제2동"],
    "북구"      : ["구포제1동", "구포제2동", "구포제3동", "곡곡동", "회현제1동", "회현제2동", "회현제3동", "덕곡제1동", "덕곡제2동", "덕곡제3동", "민덕제1동", "민덕제2동", "민덕제3동"],
    "영도대구"  : ["구포제1동", "구포제2동", "구포제3동", "곡곡동", "회현제1동", "회현제2동", "회현제3동", "덕곡제1동", "덕곡제2동", "민덕제1동", "민덕제2동", "민덕제3동"],
    "시하구"    : ["괴정제1동", "괴정제2동", "괴정제3동", "괴정제4동", "당리동", "하대제1동", "하대제2동", "신명제1동", "신명제2동", "정원제1동", "정원제2동", "다대제1동", "다대제2동", "구평동", "검촌동"],
    "금정구"    : ["시하제1동", "시하제2동", "시하제3동", "금사동", "부곡제1동", "부곡제2동", "부곡제3동", "부곡제4동", "정경제1동", "정경제2동", "신두동", "철봉로동", "남산동", "구서제1동", "구서제2동"],
    "영도구"    : ["가대제1동", "가대제2동", "연산제1동", "연산제2동", "연산제3동", "연산제4동", "연산제5동", "연산제6동", "연산제7동", "연산제8동", "연산제9동"],
    "영도구"    : ["가대제1동", "가대제2동", "연산제1동", "연산제2동", "연산제3동", "연산제4동", "연산제5동", "연산제6동", "연산제7동", "연산제8동", "연산제9동"],
    "수영구"    : ["남천제1동", "남천제2동", "수송동", "평미제1동", "평미제2동", "평안제1동", "평안제2동", "평안제3동", "평안제4동", "민락동"],
    "시상구"    : ["삼곡동", "모라제1동", "모라제2동", "덕포제1동", "재평동", "검촌동", "주례제1동", "주례제2동", "주례제3동", "학정동", "영금동"],
    "기정구"    : ["기정동", "장안동", "정관동", "월평면", "철마면"]
}
```

수집한 데이터의 수는 다음과 같고, dataframe을 csv로 바꾸어 파일로 저장하였다.

관광지 1,558개

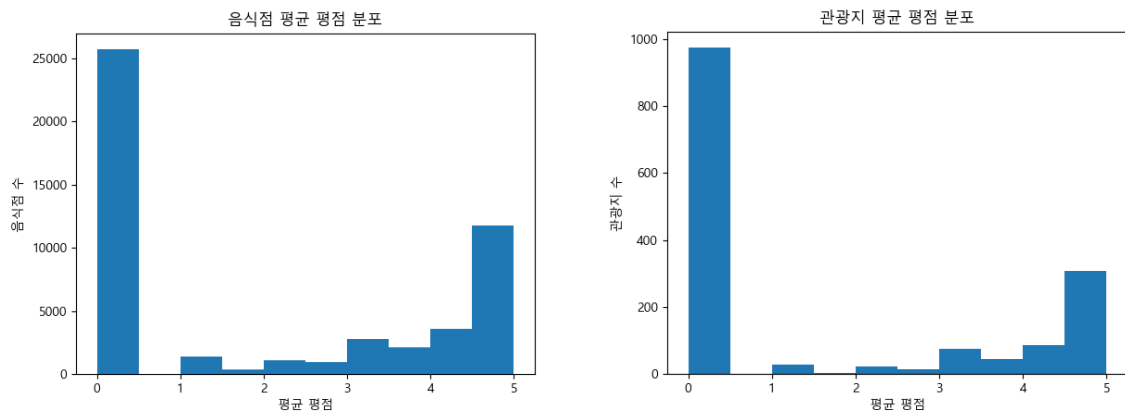
관광지에 대한 댓글 3,813개

음식점 49,804개

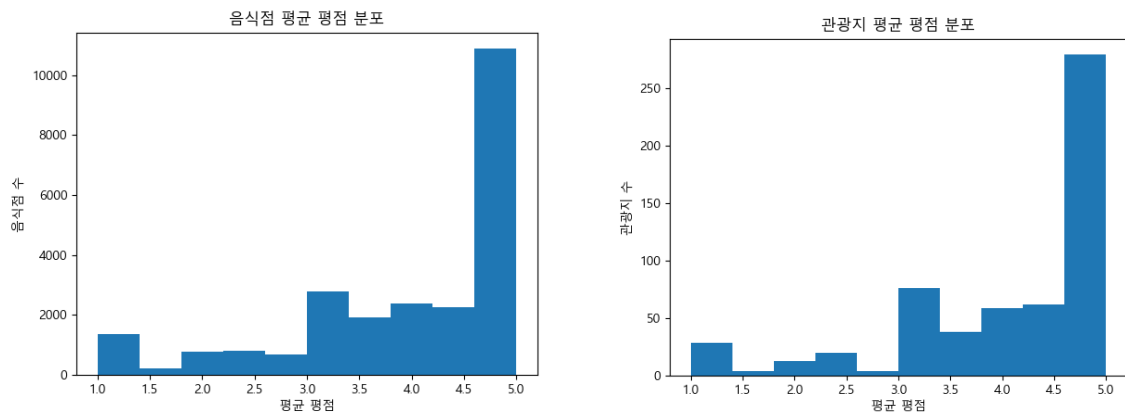
음식점에 대한 댓글 125,756개

아래는 관광지와 음식점의 평균 평점 분포이다.

0점으로 나타나는 것들은 사람들이 리뷰를 작성하지 않은 장소이다.

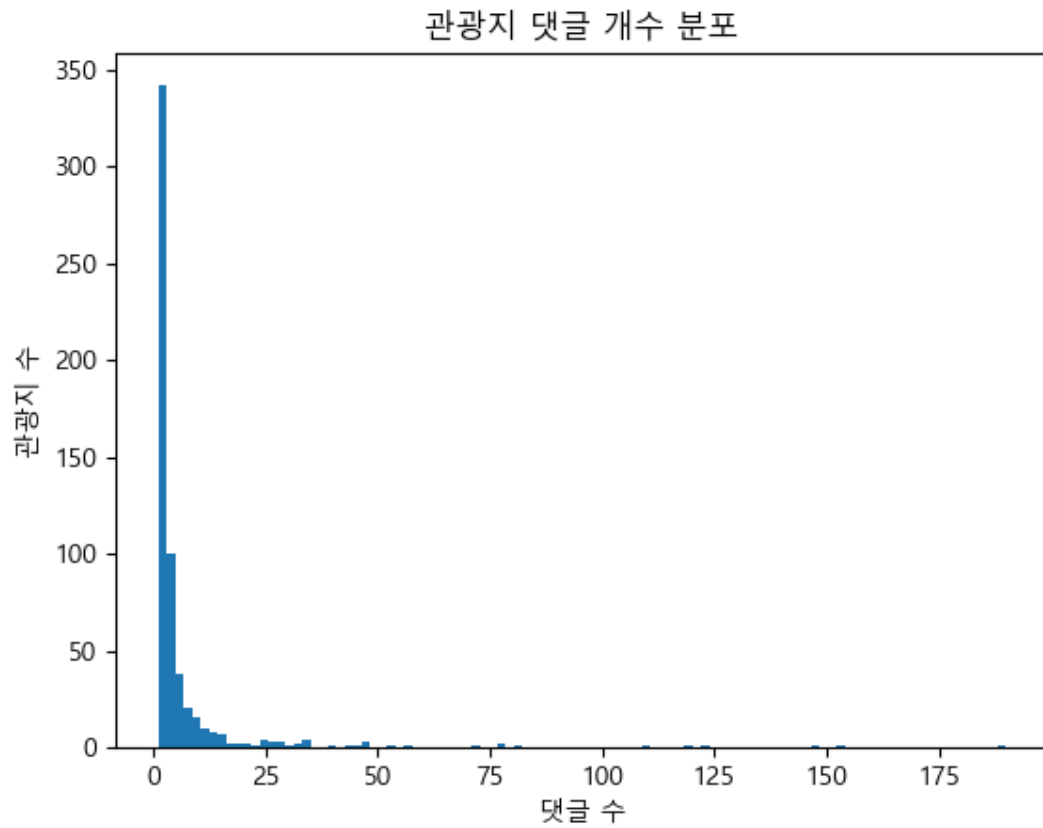


리뷰가 없는 장소를 제거한 평균 평점의 분포는 다음과 같다.

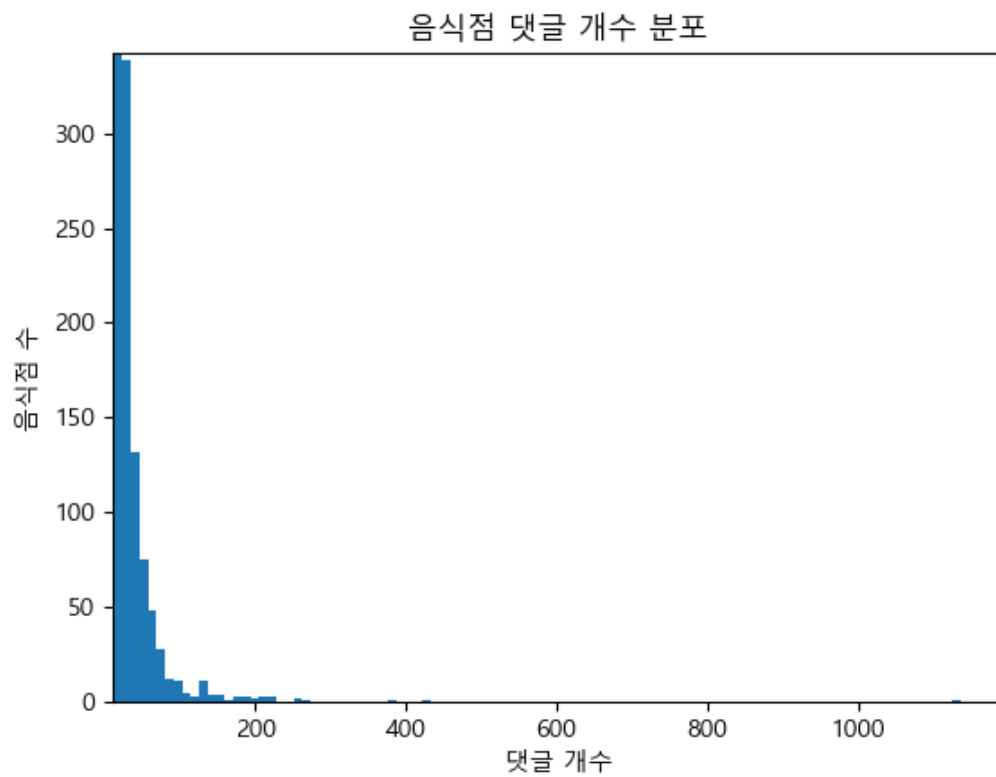
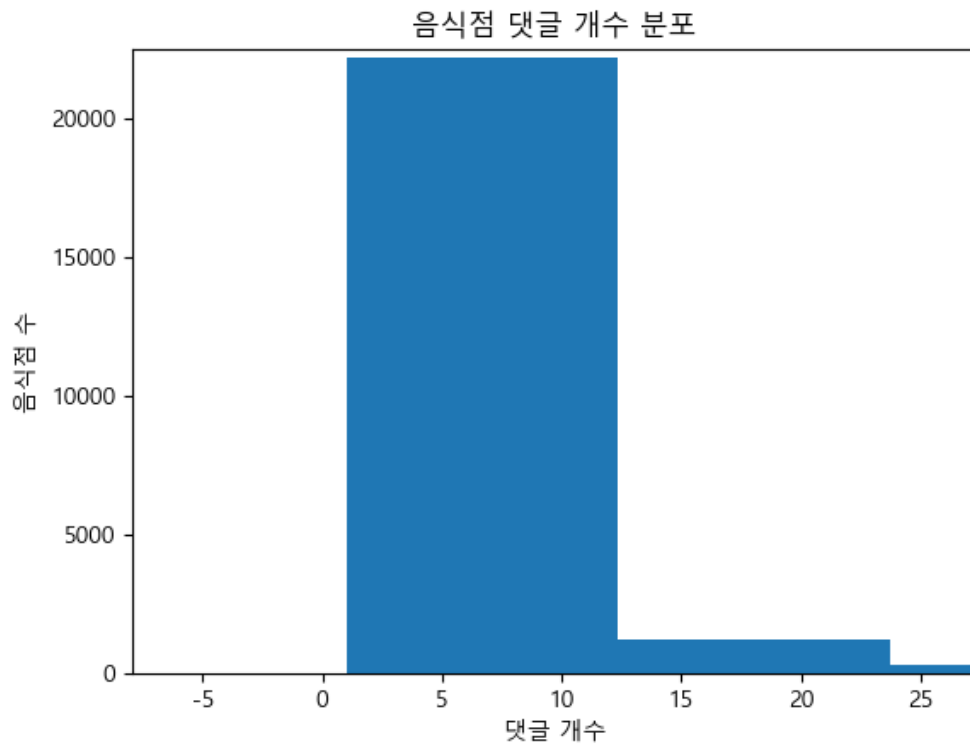


---

아래는 관광지의 댓글 개수 분포이다.



음식점의 경우 대부분이 댓글 1개~12개 사이에 분포하고 있어 히스토그램을 구간별로 잘랐다. 위는 댓글 1개부터 25개까지의 분포이고, 아래는 25개에서 최대 댓글 개수인 1137개까지의 분포이다.



### 3.1.2. 감성 분석

수집한 댓글들의 감정을 판단한다. 해당 기능을 구현하기 위해 머신러닝을 사용하였고 여러 가지 순환신경망 모델을 활용하여 테스트해보았다.

#### 3.1.2.1 데이터 전처리

먼저 크롤링한 데이터를 읽어 온다.

```
comment1 = pd.read_csv('/content/drive/MyDrive/data/food_comment.csv') # 음식점에 대한 댓글 정보
comment2 = pd.read_csv('/content/drive/MyDrive/data/comment_info.csv') # 관광지에 대한 댓글 정보
comment = pd.concat([comment1,comment2], ignore_index = True) # 두 파일을 합치기
```

장소 리뷰 평점을 보고 긍정과 부정으로 분류하였다. 평점이 4점과 5점이면 긍정을 의미하는 1로 라벨링하고 평점이 4점 미만이면 부정을 의미하는 0으로 라벨링하였다. 사용하는 전체 리뷰 수는 106063개이고 긍정 리뷰는 69635개이고 부정 리뷰는 36428개이다.

```
comment_target_list = comment['rating'] # 리뷰의 평점
comment_target = []
idx = comment[pd.notnull(comment['comment'])].index # 평점도 있고 텍스트도 있는 리뷰들의 인덱스
comment_1 = 0
comment_2 = 0
comment_3 = 0
comment_4 = 0
comment_5 = 0

for i in idx:
    if comment_target_list[i] == 1:
        comment_1 = comment_1 + 1
    elif comment_target_list[i] == 2:
        comment_2 = comment_2 + 1
    elif comment_target_list[i] == 3:
        comment_3 = comment_3 + 1
    elif comment_target_list[i] == 4:
        comment_4 = comment_4 + 1
        comment_target.append(1)
        continue
    elif comment_target_list[i] == 5:
        comment_5 = comment_5 + 1
        comment_target.append(1)
        continue
    comment_target.append(0)
print("사용할 전체 리뷰수 : {}".format(len(idx)))
print("comment_1,2,3 : {}".format(comment_1+comment_2+comment_3))
print("comment_4,5 : {}".format(comment_4+comment_5))
print(comment_target)
```

```
사용할 전체 리뷰수 : 106063
comment_1,2,3 : 36428
comment_4,5 : 69635
[1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1,
```



순환 신경망의 입력층에 데이터를 넣기 위해 각 댓글마다 명사별로 토큰화를 시켜서 라벨링한다. 한글 자연어 처리를 위해 koNLPy 라이브러리를 사용하고 가장 많이 쓰이는 명사들을 알아내기 위해 koNLPy에서 Okt 함수를 통해 댓글에서 명사들을 추출한다.

```
import konlpy
from konlpy.tag import Okt
from collections import Counter
import matplotlib.pyplot as plt

okt = Okt()

all_words = []
word_length = []

for i in idx:
    word_length.append(len(comment['comment'][i])) # 댓글의 길이

    words = okt.nouns(comment['comment'][i]) # 모든 댓글의 명사만 추출
    for word in words:
        all_words.append(word) # 뽑은 명사들을 리스트에 추가
```

어떤 명사가 많은지 파악하기 위해 모든 명사를 all\_words에 넣고 Counter함수를 통해 리스트 원소들의 개수를 파악한다. 이 중에서 가장 많이 쓰이는 단어 500개를 뽑아서 딕셔너리를 만들고 key값에는 단어명, value값에는 많이 나오는 순서대로 2부터 시작해서 라벨링할 번호를 넣어준다.

(0은 패딩용으로 쓰고, 1은 500개의 단어에 포함되지 않는 단어 라벨링 때 사용)

```
[ ] all_word_dic = Counter(all_words).most_common() # all_words 원소들의 개수 파악
word = []
count = []
for i in range(len(all_word_dic)):
    word.append(all_word_dic[i][0]) # 명사
    count.append(all_word_dic[i][1]) # 명사의 개수
word_data = {'word' : word,
             'count' : count
            }
df = pd.DataFrame(word_data)
df.to_csv("/content/drive/MyDrive/data/word.csv", encoding="utf-8-sig", index=False)

[ ] most_words_500 = word[:498]
most_words_500_dic = {most_words_500[i-2]:i for i in range(2,500)}

print(most_words_500)
print(most_words_500_dic)

['맛', '사장', '진짜', '곳', '가격', '집', '직원', '음식', '정말', '때', '여기', '맛집', '고기',
{'맛': 2, '사장': 3, '진짜': 4, '곳': 5, '가격': 6, '집': 7, '직원': 8, '음식': 9, '정말': 10, '때': 11, '여기': 12, '맛집': 13, '고기': 14}]
```

댓글별로 토큰화한 명사가 most\_words\_500\_dic의 key값에 있다면 해당하는 value 값으로 바꿔주고 없으면 1로 라벨링 해준다.

```
comment_input = []

print("총 데이터 수 : {}".format(len(idx)))
for i in idx:
    temp = []
    words = okt.nouns(comment['comment'][i]) # 댓글을 명사로 나눔
    for word in words:
        if word in most_words_500_dic:
            temp.append(most_words_500_dic[str(word)]) # 해당하는 단어가 있으면 해당 번호로 라벨링
        else:
            temp.append(1) # 해당하는 단어가 없으면 1
    comment_input.append(temp)

comment_input_df = DataFrame({'comment_input' : Series(comment_input),
                              'comment_target' : Series(comment_target)})
comment_input_df.to_csv("/content/drive/MyDrive/data/comment_input.csv", encoding="utf-8-sig", index=False)
```

### [학습 데이터의 형태]

```
print(commentInputList[:500]) # 댓글들을 라벨링한 결과
print(commentTargetList[:500]) # 해당하는 타겟 값들
```

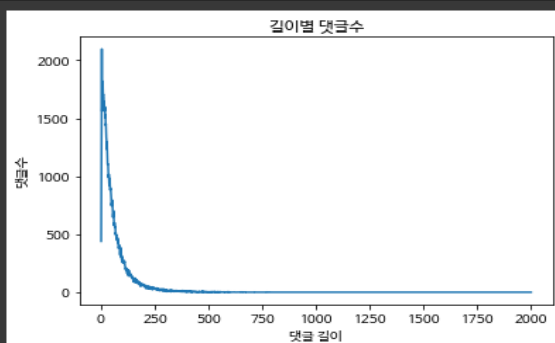
```
[[9, 78, 5], [1, 418, 19, 2, 1, 1, 47, 35, 207, 9, 215, 19, 130, 1, 101, 9, 1, 1, 122, 1, 29, 20, 1], [1, 2, 30, 280, 7,
[1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1
```

전체 댓글의 길이를 분석했을 때 길이가 대부분 250자를 넘지 못함을 볼 수 있다.

단어로 분석하면 대부분 100개의 단어를 넘지 못할 것이므로 입력층에 넣을 데이터의 길이를 100으로 한다.

```
[ ] words_count = Counter(word_length)
    plotList = words_count.items()
    plotList = sorted(plotList)
    x,y = zip(*plotList)

    plt.plot(x,y)
    plt.xlabel('댓글 길이')
    plt.ylabel('댓글수')
    plt.title('길이별 댓글수')
    plt.show()
```



지금까지 만들어둔 학습데이터들을 `train_test_split()` 함수를 사용해 학습데이터와 검증데이터로 나눠준다. 학습데이터의 30프로를 검증데이터로 만들었으며 타겟데이터에서 0과 1의 비율이 반반이 되도록 나누었다.

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split

train_input, val_input, train_target, val_target = train_test_split(commentInputList, commentTargetList,
                                                                    test_size=0.3, stratify=commentTargetList)

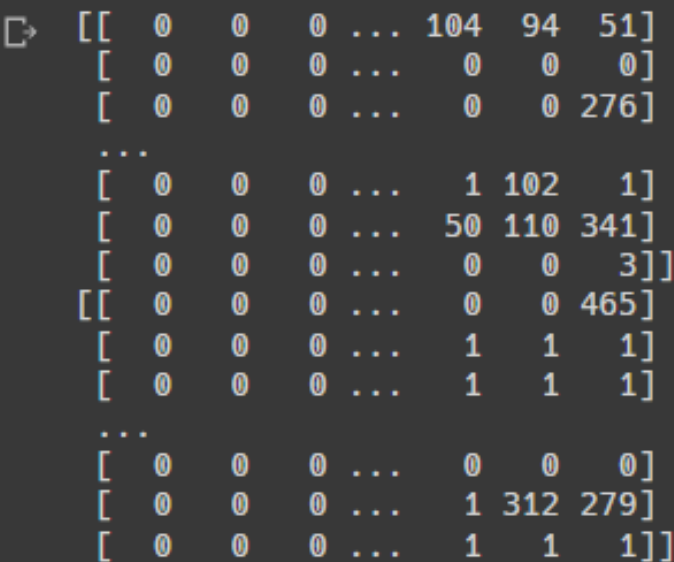
train_seq = pad_sequences(train_input, maxlen = 100)
val_seq = pad_sequences(val_input, maxlen = 100)
print(train_seq.shape)

train_target = np.array(train_target)
val_target = np.array(val_target)
train_seq = np.array(train_seq)
val_seq = np.array(val_seq)

(74244, 100)
```

마지막으로 입력층에 넣기 위해 입력데이터들을 `pad_sequences` 함수를 이용해 길이가 100이 되도록 패딩을 시켜주면 데이터 전처리 과정은 끝이 난다.

```
print(train_seq)
print(val_seq)
```



### 3.1.2.2 모델 설계 및 평가

텐서플로우의 keras를 이용해 모델을 설계 및 평가하였고 가장 좋은 성능을 가진 모델을 찾기 위해 순환층의 종류와 셀의 개수와 층의 개수를 다르게 하여 비교해보았다.

평가 방식은 검증 손실과 정확도를 사용해 얼마나 검증 손실이 낮은지, 정확도가 높은지 확인하며 평가한다.

먼저 초기 모델은 가장 간단한 순환층인 simpleRNN에 단어 임베딩으로 500개의 단어를 16개의 임베딩 벡터로 사용해 입력층에 넣었다. 은닉층에 있는 simpleRNN 순환층의 활성화 함수는 디폴트값인 하이퍼볼릭 텐젠트 함수를 사용한다.

마지막 출력층은 1개의 뉴런을 가지고, 활성화 함수는 감성분석이 이진분류이기 때문에 시그모이드함수를 사용했다.

```
[ ] from tensorflow import keras
```

```
model = keras.Sequential()  
model.add(keras.layers.Embedding(500, 16, input_length=100))  
model.add(keras.layers.SimpleRNN(8))  
model.add(keras.layers.Dense(1, activation='sigmoid'))  
model.summary()
```

Model: "sequential\_3"

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, 100, 16)	8000
simple_rnn_2 (SimpleRNN)	(None, 8)	200
dense_3 (Dense)	(None, 1)	9

```
=====  
Total params: 8,209  
Trainable params: 8,209  
Non-trainable params: 0  
=====
```

모델을 학습하기 전에 파라미터 설정은 learning\_rate는 1e-4, epoch는 100, batch\_size는 64 이고, 콜백 객체를 이용해 3번 연속 검증 점수가 향상되지 않으면 조기 종료를 하고 가장 낮은 검증 손실을 낸 모델 파라미터를 저장한다. 손실함수는 binary\_crossentropy를 사용하였다.

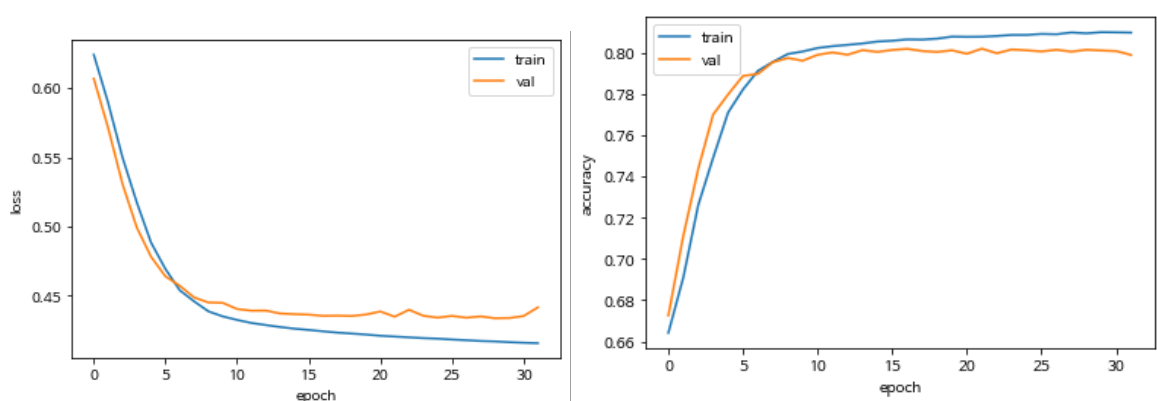
```
rmsprop = keras.optimizers.RMSprop(learning_rate = 1e-4)
model.compile(optimizer = rmsprop, loss = 'binary_crossentropy', metrics=['accuracy'])
checkpoint_cb = keras.callbacks.ModelCheckpoint('/content/drive/MyDrive/model/best-simplernn-model.h5')
early_stopping_cb = keras.callbacks.EarlyStopping(patience=3, restore_best_weights=True)
history = model.fit(train_seq, train_target, epochs=100, batch_size=64, validation_data=(val_seq, val_target),
                    callbacks=[checkpoint_cb, early_stopping_cb])

print('finish')
```

첫번째 모델은 에포크를 32번 거치고 완료되었다.

```
Epoch 29/100
1161/1161 [=====] - 106s 91ms/step - loss: 0.4168 - accuracy: 0.8093 - val_loss: 0.4336 - val_accuracy: 0.8013
Epoch 30/100
1161/1161 [=====] - 107s 92ms/step - loss: 0.4163 - accuracy: 0.8098 - val_loss: 0.4337 - val_accuracy: 0.8009
Epoch 31/100
1161/1161 [=====] - 106s 91ms/step - loss: 0.4158 - accuracy: 0.8097 - val_loss: 0.4352 - val_accuracy: 0.8007
Epoch 32/100
1161/1161 [=====] - 106s 91ms/step - loss: 0.4155 - accuracy: 0.8096 - val_loss: 0.4414 - val_accuracy: 0.7988
finish
```

첫번째 모델의 손실함수와 정확도 그래프이다. 손실 함수를 살펴봤을 때 에포크가 늘어남에 따라 검증 세트와 훈련 세트의 차이가 점점 커짐을 볼 수 있다.



두번째 모델은 고급 순환망인 LSTM를 사용하였고 LSTM 순환망의 뉴런의 개수는 8개로 설정하고 초기모델과 같이 500개의 단어를 16개의 임베딩 벡터로 사용해 입력층에 넣었다. 활성화 함수는 하이퍼볼릭 탄젠트 함수를 사용한다.

마지막 출력층은 1개의 뉴런을 가지고 이진분류이기 때문에 시그모이드 활성화 함수를 사용했다.

```
model2 = keras.Sequential()
model2.add(keras.layers.Embedding(500, 16, input_length=100))
model2.add(keras.layers.LSTM(8))
model2.add(keras.layers.Dense(1, activation='sigmoid'))
model2.summary()
```

Model: "sequential\_4"

Layer (type)	Output Shape	Param #
embedding_4 (Embedding)	(None, 100, 16)	8000
lstm (LSTM)	(None, 8)	800
dense_4 (Dense)	(None, 1)	9

```
=====
Total params: 8,809
Trainable params: 8,809
Non-trainable params: 0
=====
```

파라미터는 이전 모델과 동일하다.

```
rmsprop = keras.optimizers.RMSprop(learning_rate = 1e-4)
model2.compile(optimizer = rmsprop, loss = 'binary_crossentropy', metrics=['accuracy'])
checkpoint_cb = keras.callbacks.ModelCheckpoint('best-lstm-model2.h5')
early_stopping_cb = keras.callbacks.EarlyStopping(patience=3, restore_best_weights=True)
history2 = model2.fit(train_seq, train_target, epochs=100, batch_size=64, validation_data=(val_seq, val_target),
                      callbacks=[checkpoint_cb, early_stopping_cb])

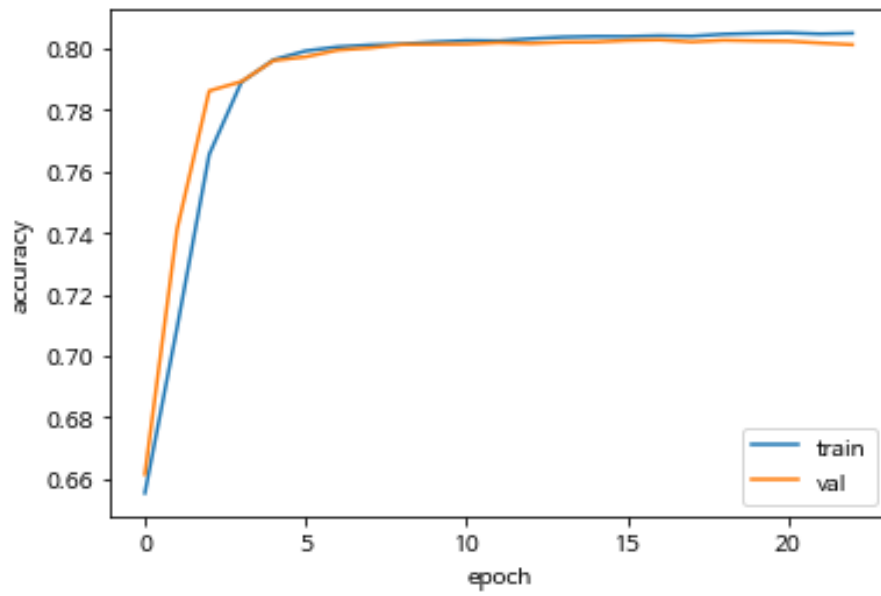
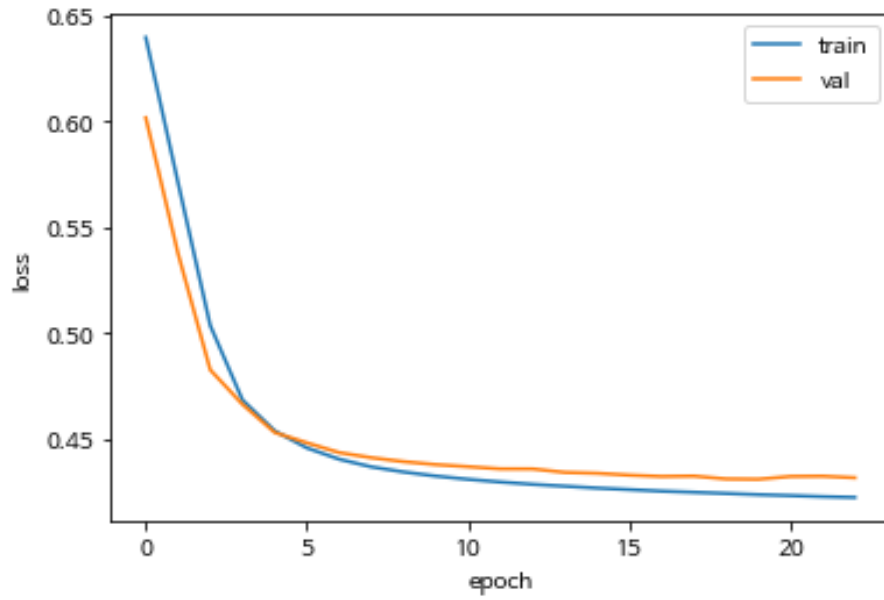
print('finish')
```

두번째 모델은 에포크를 23번 거치고 완료되었다.

```
Epoch 22/100
1161/1161 [=====] - 10s 9ms/step - loss: 0.4229 - accuracy: 0.8047 - val_loss: 0.4325 - val_accuracy: 0.8017
Epoch 23/100
1161/1161 [=====] - 12s 11ms/step - loss: 0.4225 - accuracy: 0.8049 - val_loss: 0.4318 - val_accuracy: 0.8012
finish
```

---

첫번째 모델과 비교하여 손실세트와 훈련 세트의 차이가 줄어들을 볼 수 있다.



세번째 모델은 이전 모델에서 LSTM층을 하나 더 추가하고 LSTM 순환층에 드롭아웃을 사용하여 뉴런의 출력을 랜덤하게 꺼서 특정 뉴런에 의존하는 것을 막았다. 나머지는 이전 모델과 동일하다.

```
model3 = keras.Sequential()
model3.add(keras.layers.Embedding(500, 16, input_length=100))
model3.add(keras.layers.LSTM(8, dropout=0.3, return_sequences = True))
model3.add(keras.layers.LSTM(8, dropout=0.3))
model3.add(keras.layers.Dense(1, activation='sigmoid'))
model3.summary()
```

Model: "sequential\_5"

Layer (type)	Output Shape	Param #
embedding_5 (Embedding)	(None, 100, 16)	8000
lstm_1 (LSTM)	(None, 100, 8)	800
lstm_2 (LSTM)	(None, 8)	544
dense_5 (Dense)	(None, 1)	9
Total params: 9,353		
Trainable params: 9,353		
Non-trainable params: 0		

파라미터는 이전 모델과 동일하다.

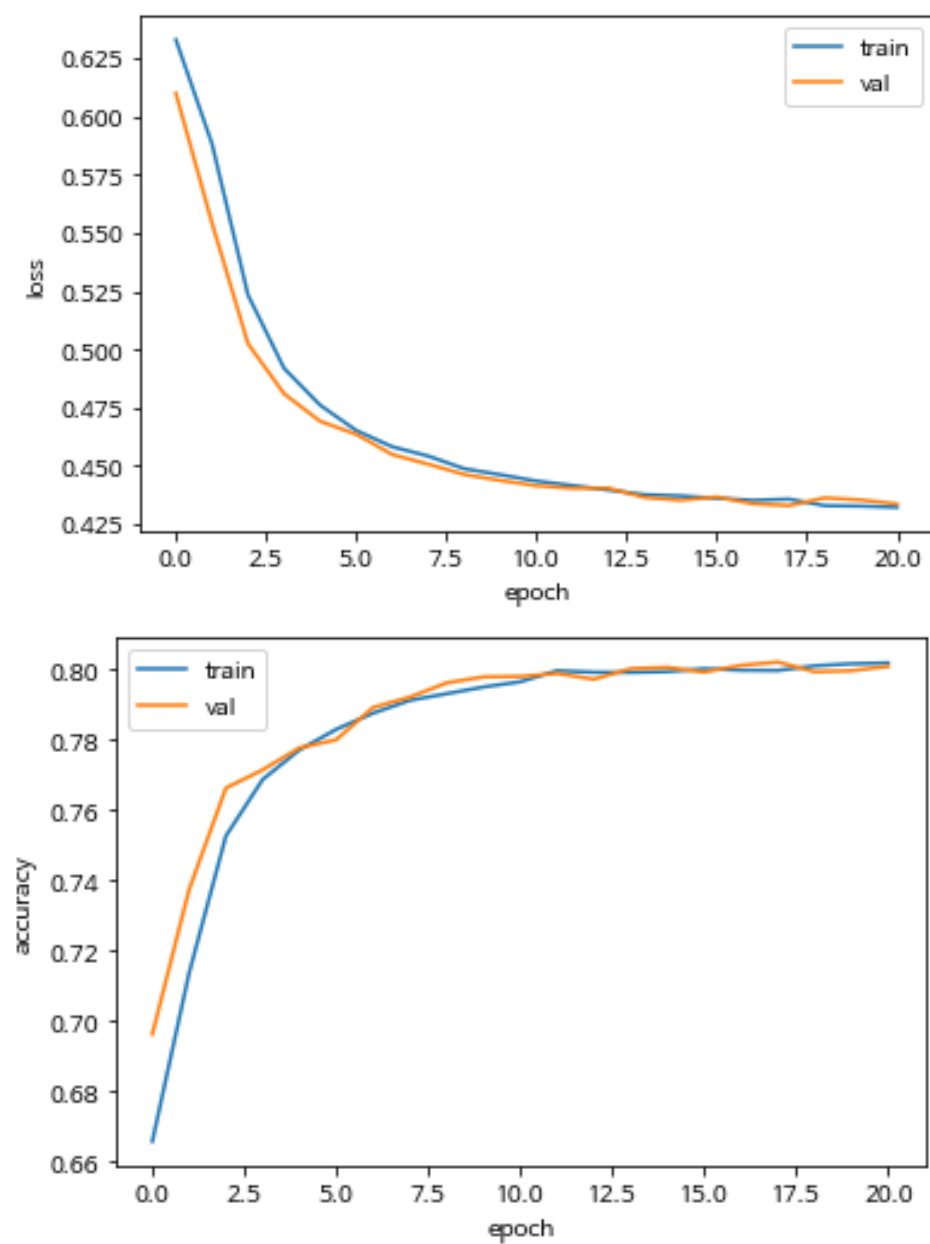
```
rmsprop = keras.optimizers.RMSprop(learning_rate = 1e-4)
model3.compile(optimizer = rmsprop, loss = 'binary_crossentropy', metrics=['accuracy'])
checkpoint_cb = keras.callbacks.ModelCheckpoint('/content/drive/MyDrive/model/best-lstm-model3.h5')
early_stopping_cb = keras.callbacks.EarlyStopping(patience=3, restore_best_weights=True)
history3 = model3.fit(train_seq, train_target, epochs=100, batch_size=64, validation_data=(val_seq, val_target),
                    callbacks=[checkpoint_cb, early_stopping_cb])
print('finish')
```

세번째 모델은 에포크를 21번 거치고 완료되었다.

```
1161/1161 [=====] - 17s 13ms/step - loss: 0.4329 - accuracy: 0.8010 - val_loss: 0.4363 - val_accuracy: 0.7993
Epoch 20/100
1161/1161 [=====] - 16s 14ms/step - loss: 0.4327 - accuracy: 0.8016 - val_loss: 0.4354 - val_accuracy: 0.7996
Epoch 21/100
1161/1161 [=====] - 16s 14ms/step - loss: 0.4322 - accuracy: 0.8018 - val_loss: 0.4337 - val_accuracy: 0.8008
finish
```



이전 모델에 비해 손실함수와 정확도 둘 다 손실 세트와 훈련세트가 더욱더 일치함을 볼 수 있다.



네번째 모델은 이전 모델에서 LSTM층을 하나 더 추가하고 LSTM의 뉴런의 수를 8개에서 16개로 늘려보았다.

```
model4 = keras.Sequential()
model4.add(keras.layers.Embedding(500, 16, input_length=100))
model4.add(keras.layers.LSTM(16, dropout=0.3, return_sequences = True))
model4.add(keras.layers.LSTM(16, dropout=0.3, return_sequences = True))
model4.add(keras.layers.LSTM(16, dropout=0.3))
model4.add(keras.layers.Dense(1, activation='sigmoid'))
model4.summary()
```

Model: "sequential\_6"

Layer (type)	Output Shape	Param #
embedding_6 (Embedding)	(None, 100, 16)	8000
lstm_3 (LSTM)	(None, 100, 16)	2112
lstm_4 (LSTM)	(None, 100, 16)	2112
lstm_5 (LSTM)	(None, 16)	2112
dense_6 (Dense)	(None, 1)	17
Total params: 14,353		
Trainable params: 14,353		
Non-trainable params: 0		

파라미터는 이전 모델과 동일하다.

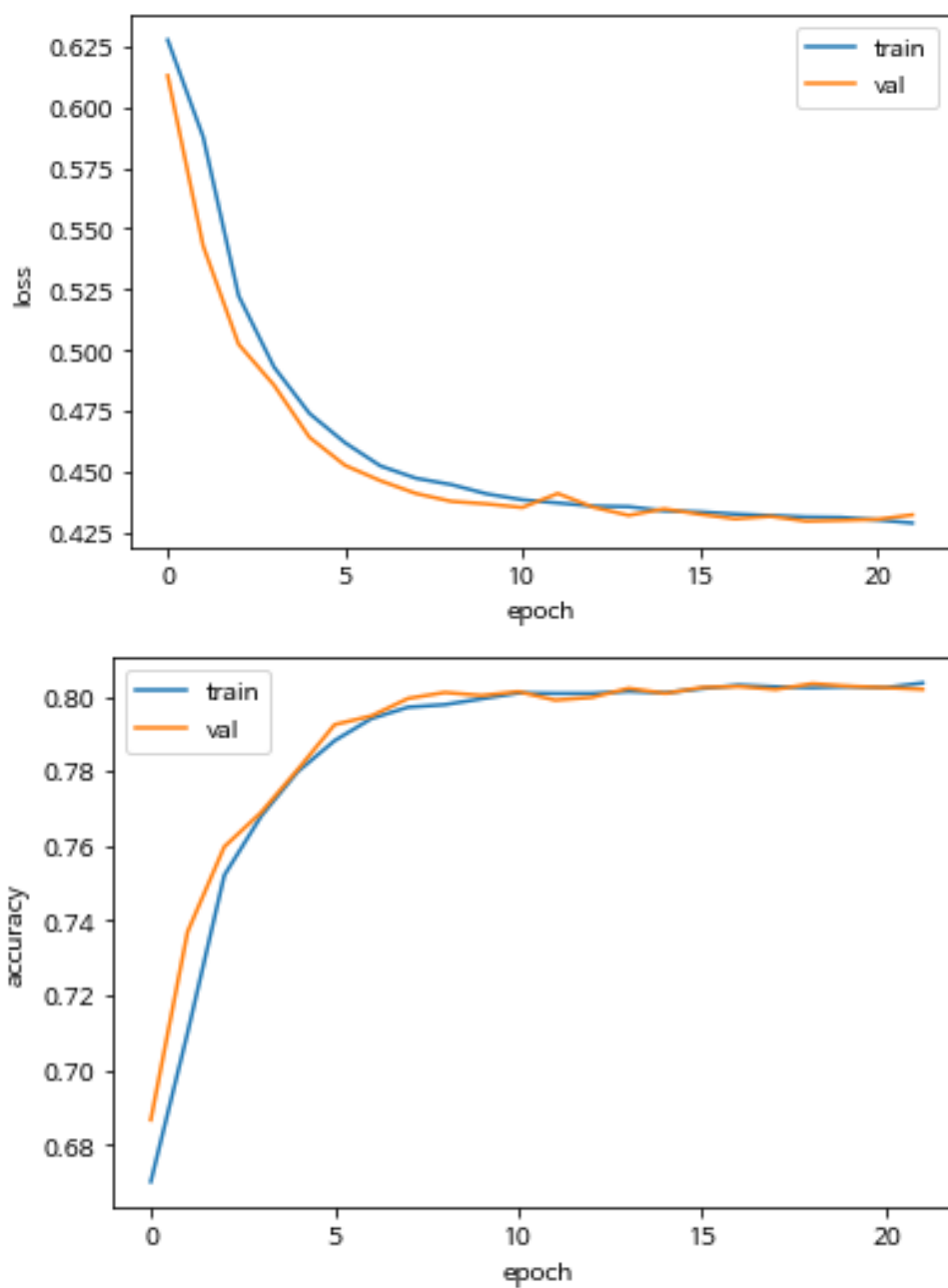
```
rmsprop = keras.optimizers.RMSprop(learning_rate = 1e-4)
model4.compile(optimizer = rmsprop, loss = 'binary_crossentropy', metrics=['accuracy'])
checkpoint_cb =keras.callbacks.ModelCheckpoint('/content/drive/MyDrive/model/best-lstm-model4.h5')
early_stopping_cb = keras.callbacks.EarlyStopping(patience=3, restore_best_weights=True)
history4 = model4.fit([train_seq, train_target, epochs=100, batch_size=64, validation_data=(val_seq,val_target),
callbacks=[checkpoint_cb,early_stopping_cb]])

print('finish')
```

네번째 모델은 에포크를 22번 거치고 완료되었다.

```
Epoch 21/100
1161/1161 [=====] - 23s 20ms/step - loss: 0.4302 - accuracy: 0.8023 - val_loss: 0.4304 - val_accuracy: 0.8023
Epoch 22/100
1161/1161 [=====] - 23s 20ms/step - loss: 0.4289 - accuracy: 0.8036 - val_loss: 0.4322 - val_accuracy: 0.8019
finish
```

이전 모델에 비해 성능이 별 차이가 없었다. 따라서 더 이상 모델을 복잡하게 하더라도 성능이 더 나이지지 않음을 보고 모델4를 이용해 감성분석을 해보았다.



### 3.1.2.3 결과 및 응용

학습시킨 모델을 들고 와서 최근 3개월 내에 작성된 댓글들을 모두 감성분석한다.  
댓글이 있는 장소에 대해 감성 분석한 결과와 평점을 곱한 값을 모두 더해서 점수를 구했다. 이것을 이용해 관광지 추천 모델을 만들었다.

먼저 이전에 학습시킨 모델과 단어 정보를 들고 온다.

```
model = tf.keras.models.load_model('/content/drive/MyDrive/model/best-lstm-model4.h5')
word = pd.read_csv('/content/drive/MyDrive/data/word.csv')
most_words_500_dic = {word['word'][i-2]:i for i in range(2,500)}
```

dateutil 라이브러리를 사용하여 현재 시간을 구하고 검색 기간을 정한다.

```
from dateutil.parser import parse
from dateutil.relativedelta import relativedelta
import datetime as dt

now = dt.datetime.now()

startMonth = now.month -3 # 최근 3개월
endMonth = now.month
year = str(now.year)

if startMonth < 10:
    startMonth = '0'+ str(startMonth)

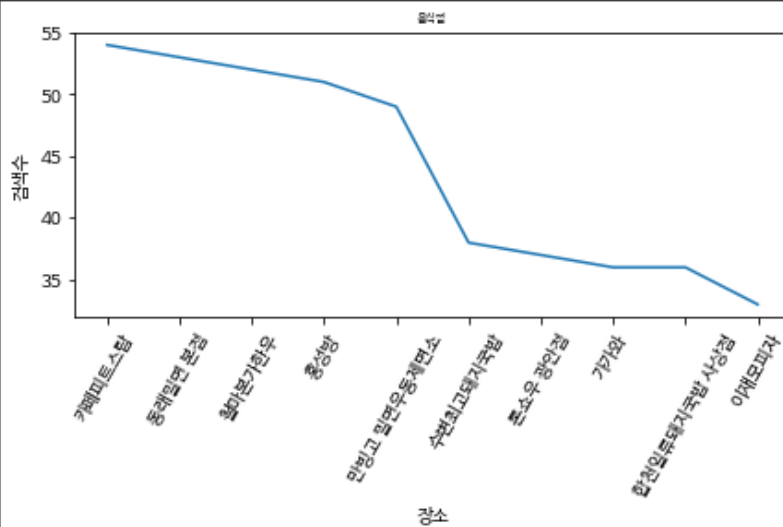
if endMonth < 10:
    endMonth = '0'+ str(endMonth)
```

기간을 구했으면 댓글 작성 일자를 보고 최근 3개월 간 작성된 댓글들을 모두 수집한다.

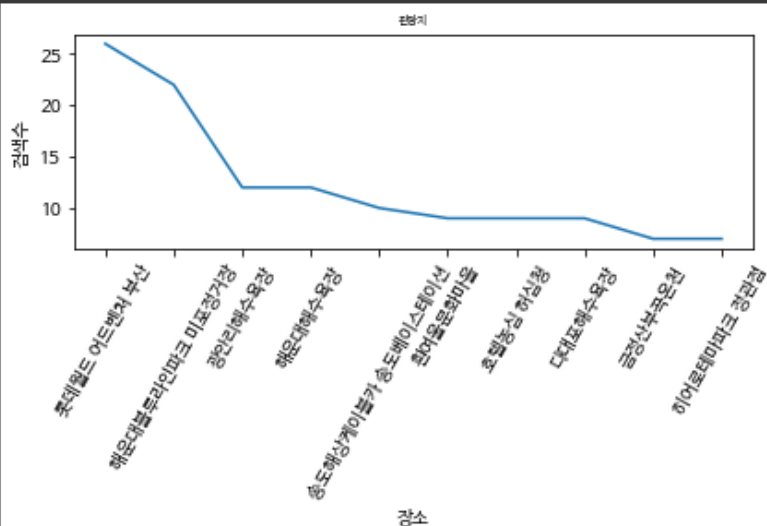
```
food_datelist = food_comment['created_at'].to_list() #음식점별 댓글 리스트
food_idx = []
food_notNullidx = food_comment[pd.notnull(food_comment['comment'])].index
for i in range(len(food_datelist)):
    date = food_datelist[i].split('.')
    if date[0] == year and startMonth <= date[1] <= endMonth and i in food_notNullidx: #최근 3개월 간 댓글인지 확인
        food_idx.append(i)

travel_datelist = travel_comment['created_at'].to_list() #관광지별 댓글 리스트
travel_idx = []
travel_notNullidx = travel_comment[pd.notnull(travel_comment['comment'])].index
for i in range(len(travel_datelist)):
    date = travel_datelist[i].split('.')
    if date[0] == year and startMonth <= date[1] <= endMonth and i in travel_notNullidx: #최근 3개월 간 댓글인지 확인
        travel_idx.append(i)
```

06월에서 09월까지 댓글 개수 : 14620



06월에서 09월까지 댓글 개수 : 306



가져온 단어 목록과 koNLPy 라이브러리를 사용하여 가져온 댓글들을 라벨링 시켜주는 함수를 만든다. 방식은 모델을 학습시킬 때 썼던 방식과 같다.

```
from tensorflow.keras.preprocessing.sequence import pad_sequences

def labeling(inputList, most_words_500_dic): #댓글 목록과 이전에 가져온 단어 리스트
    result = []
    okt = Okt()
    for i in range(len(inputList)):
        temp = []
        words = okt.nouns(inputList[i])
        for word in words:
            if word in most_words_500_dic:
                temp.append(most_words_500_dic[str(word)]) #단어가 dic에 들어있으면 그에 맞게 라벨링
            else:
                temp.append(1) #없으면 0
        result.append(temp)

    result = pad_sequences(result, maxlen=100) #길이가 100이 되도록 패딩
    return result
```

함수를 이용해 댓글들을 데이터 전처리시킨다.

```
#최근 3개월 음식점 댓글
food_labelList = labeling(food_comment['comment'][food_idx].to_list(),most_words_500_dic)
#최근 3개월 관광지 댓글
travel_labelList = labeling(travel_comment['comment'][travel_idx].to_list(),most_words_500_dic)

#모든 관광지 댓글
all_travel_labelList = labeling(travel_comment['comment'][travel_notNullidx].to_list(), most_words_500_dic)
#모든 음식점 댓글
all_food_labelList = labeling(food_comment['comment'][food_notNullidx].to_list(), most_words_500_dic)
```

이전에 학습시킨 모델을 가져와 predict함수로 결과값을 예측한다.

감성분석 결과값들을 리스트에 모두 저장해준다.

```
food_result = model.predict(food_labelList)
travel_result = model.predict(travel_labelList)
all_travel_result = model.predict(all_travel_labelList)
all_food_result = model.predict(all_food_labelList)
```

이제 특정 장소에 해당하는 댓글들을 모두 모아 댓글들의 감성분석한 결과값이랑 평점을 곱해 합산해서 점수를 구한다.

```
def score(sentiAnal, place): # 감성분석한 리스트와 장소 리스트
    score_list = []
    for i in range(len(place)):
        condition = (sentiAnal.place == place[i]) #같은 장소의 리뷰만 선택
        score = 0
        for j in range(len(sentiAnal[condition])):
            # 점수는 해당 장소에 대한 리뷰들을 감성 분석한 결과와 그 평점을 곱하고 합산한다.
            score += sentiAnal[condition]['rating'].to_list()[j] * sentiAnal[condition]['comment_sentimentAnalysis'].to_list()[j]
        score_list.append(score)
    return score_list
```

```
] food_score_list = score(food_sentiAnal, food_place)
   travel_score_list = score(travel_sentiAnal, travel_place)
   all_travel_score_list = score(all_travel_sentiAnal, all_travel_place)
   all_food_score_list = score(all_food_sentiAnal, all_food_place)
```

이것들을 파일로 만들어 홈페이지에 사용한다.

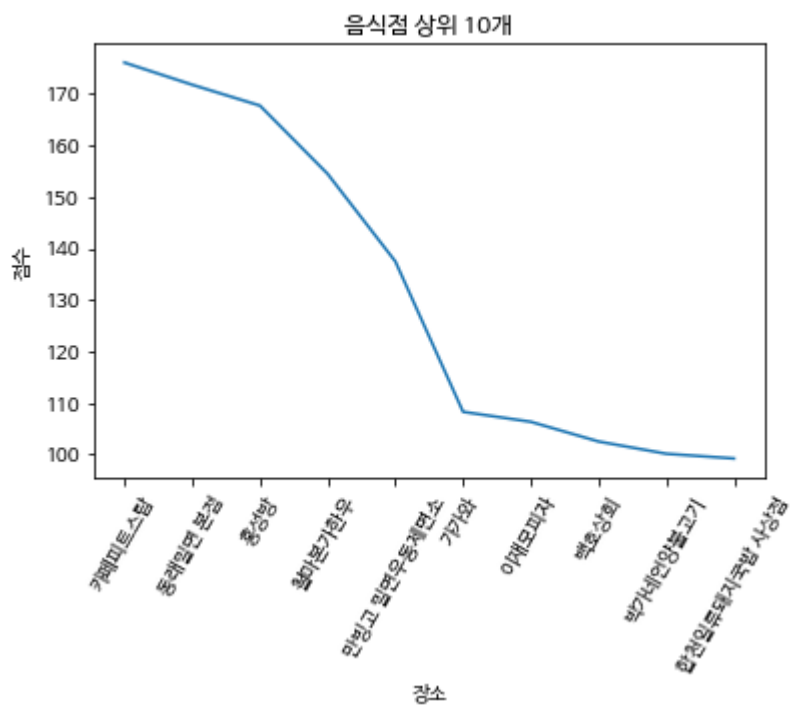
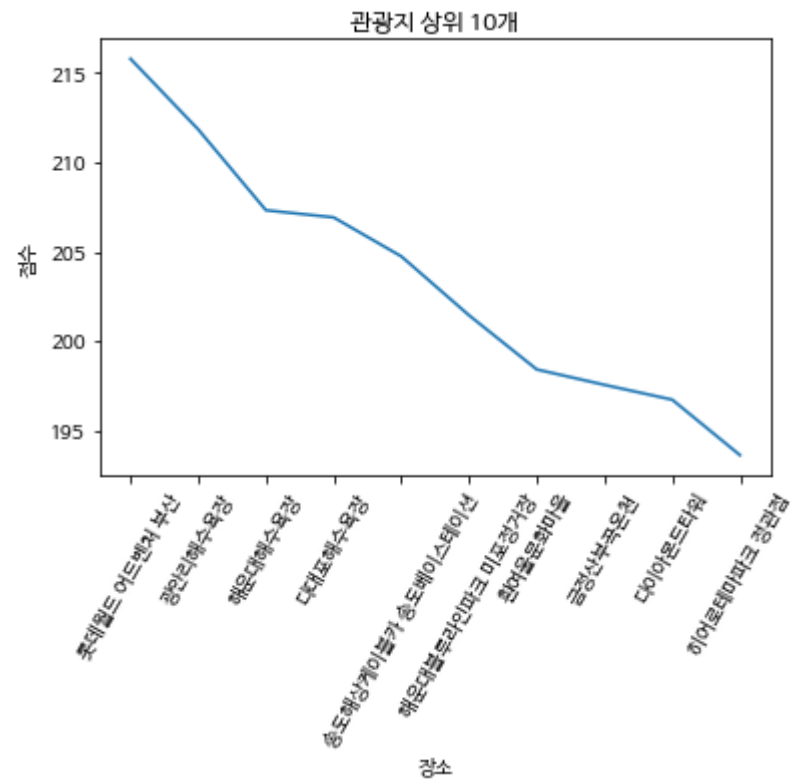
```
# 인기 관광지 추천에 사용
food_score_list_pd = {'place' : food_place,
                      'score' : food_score_list}
food_score_list_pd = pd.DataFrame(food_score_list_pd)
food_score_list_pd.to_csv("/content/drive/MyDrive/data/food_score_list.csv", index=False)
food_score_list_pd.to_excel("/content/drive/MyDrive/data/food_score_list.xlsx", index=False)

# 인기 관광지 추천에 사용
travel_score_list_pd = {'place' : travel_place,
                        'score' : travel_score_list}
travel_score_list_pd = pd.DataFrame(travel_score_list_pd)
travel_score_list_pd.to_csv("/content/drive/MyDrive/data/travel_score_list.csv", index=False)
travel_score_list_pd.to_excel("/content/drive/MyDrive/data/travel_score_list.xlsx", index=False)

# 전체 관광지 추천에 사용
all_travel_score_list_pd = {'place' : all_travel_place,
                             'score' : all_travel_score_list}
all_travel_score_list_pd = pd.DataFrame(all_travel_score_list_pd)
all_travel_score_list_pd.to_csv("/content/drive/MyDrive/data/all_travel_score_list.csv", index=False)
all_travel_score_list_pd.to_excel("/content/drive/MyDrive/data/all_travel_score_list.xlsx", index=False)

# 전체 음식점 추천에 사용
all_food_score_list_pd = {'place' : all_food_place,
                           'score' : all_food_score_list}
all_food_score_list_pd = pd.DataFrame(all_food_score_list_pd)
all_food_score_list_pd.to_csv("/content/drive/MyDrive/data/all_food_score_list.csv", index=False)
all_food_score_list_pd.to_excel("/content/drive/MyDrive/data/all_food_score_list.xlsx", index=False)
```

관광지와 음식점의 점수에 따른 상위 10개만 뽑은 데이터이다.





## 3.2. 트렌드 시각화

구, 군별로 인기 관광지과 가장 많이 쓰이는 단어를 이용하여 트렌드를 보여준다.

### 3.2.1 데이터 수집

#### 1. 최근 3개월 간 요일 별 리뷰수

최근 3개월 간의 댓글들이 언제 쓰였는지 확인하고 그 날짜가 어떤 요일인지 확인 후에 요일별로 몇개의 댓글이 있는지 구해서 데이터 수집한다

```
[3] comment = pd.read_csv('/content/drive/MyDrive/data/comment.csv') # 현재 댓글 정보를 가져온다.

현재 시간을 구하고 최근 3개월에 해당하는 기간을 구한다.

[4] now = dt.datetime.now() # 현재 시간 구하기

startMonth = now.month -3 # 최근 3개월
endMonth = now.month
year = str(now.year)

if startMonth < 10:
    startMonth = '0'+ str(startMonth)

if endMonth < 10:
    endMonth = '0'+ str(endMonth)

dateutil 라이브러리에서 제공하는 parse로 리뷰 작성 일자를 datetime 객체로 변형해주고 datetime 객체에서 자동으로 요일을 변환하는 weekday
함수를 쓰면 손쉽게 요일별 댓글 수를 구할 수 있다.

[ ] weekdayList = []
search_idx = []
comment_notNullidx = comment[pd.notnull(comment['comment'])].index
comment_datelist = comment['created_at'].to_list()
for i in range(len(comment_datelist)):
    date = comment_datelist[i].split('.')
    # 검색 기간에 속하는 리뷰 인덱스 구함
    if date[0] == year and startMonth <= date[1] <= endMonth and i in comment_notNullidx:
        search_idx.append(i)

for i in search_idx:
    weekdayList.append(parse(comment['created_at'][i]).weekday()) # {0:월, 1:화, 2:수, 3:목, 4:금, 5:토, 6:일}

[ ] weekdayCount = sorted(dict(Counter(weekdayList)).items())

[(0, 2134), (1, 1920), (2, 1934), (3, 2049), (4, 2042), (5, 2334), (6, 2513)]
```

## 2. 리뷰 건수를 이용한 구, 군별 데이터

최근 3개월 간의 댓글 수를 이용하여 각 구, 군별로 얼마나 많이 검색되었는지 확인한다.

먼저 음식점과 관광지 데이터에서 최근 3개월 이내에 작성된 댓글 수를 가져온다.

```
[5] now = dt.datetime.now()

startMonth = now.month - 3
endMonth = now.month
year = str(now.year)

if startMonth < 10:
    startMonth = '0'+ str(startMonth)

if endMonth < 10:
    endMonth = '0'+ str(endMonth)
```

음식점 데이터 중에서 댓글이 있고 최근 3개월 이내에 작성된 댓글들의 수를 가져옴

```
[6] food_datelist = food_comment['created_at'].to_list()
food_idx = []
food_notNullidx = food_comment[pd.notnull(food_comment['comment'])].index #댓글이 있는 것만 추출
for i in range(len(food_datelist)):
    date = food_datelist[i].split('.')
    if date[0] == year and startMonth <= date[1] <= endMonth and i in food_notNullidx:
        food_idx.append(i)
print(startMonth+'월에서 '+endMonth+'월까지 댓글 개수 : {}'.format(len(food_idx)))

place_food_count = dict(Counter(food_comment['place_name'][food_idx]))
print(place_food_count)
```

```
06월에서 09월까지 댓글 개수 : 14620
{'본참치': 5, '뚝보집': 7, '노티스': 1, '부산원조곰장어맛집 성일집': 3, '중왕모밀': 2, '그집곰도리탕 본점': 1}
```

관광지 데이터 중에서 댓글이 있고 최근 3개월 이내에 작성된 댓글들의 수를 가져옴

```
[7] travel_datelist = travel_comment['created_at'].to_list()
travel_idx = []
travel_notNullidx = travel_comment[pd.notnull(travel_comment['comment'])].index
for i in range(len(travel_datelist)):
    date = travel_datelist[i].split('.')
    if date[0] == year and startMonth <= date[1] <= endMonth and i in travel_notNullidx:
        travel_idx.append(i)
print(startMonth+'월에서 '+endMonth+'월까지 댓글 개수 : {}'.format(len(travel_idx)))

place_travel_count = dict(Counter(travel_comment['place_name'][travel_idx]))
print(place_travel_count)
```

```
06월에서 09월까지 댓글 개수 : 306
{'보수동책방글목': 2, '다이아몬드타워': 6, '국제시장먹자골목': 1, '남포동 포장마차거리': 2, '팔빙수골목': 1}
```

관광지와 음식점의 댓글이 적힌 장소의 주소를 들고 와서 장소, 주소, 댓글 수 형식으로 csv 파일을 만들어준다.

개별 관광지 장소에 몇개의 댓글이 달렸는지 확인했으니 해당 장소에 대응하는 주소를 들고와서 장소, 이름, 댓글수 정보가 담긴 csv 파일을 만든다.

```
[8] travel_address = travel_place['address'].to_list()
    address = []
    for i in travel_address:
        address.append(i.split()[1])

    find_idx = []
    length = len(travel_place)
    for i in range(length):
        if travel_place['place_name'][i] in list(place_travel_count.keys()):
            find_idx.append(i)

    place_address = []
    for i in find_idx:
        place_address.append(address[i])

    count = []
    for i in find_idx:
        count.append(place_travel_count[travel_place['place_name'][i]])

    regiontravelData = {'place_name' : travel_place['place_name'][find_idx].to_list(),
                        'address' : place_address,
                        'count' : count
                        }
    df = pd.DataFrame(regiontravelData)
    df.to_csv("/content/drive/MyDrive/data/regiontravelData.csv", index=False)
    df.to_excel("/content/drive/MyDrive/data/regiontravelData.xlsx", index=False)
```

음식점도 관광지와 마찬가지로 csv파일을 만들어 준다.

```
[9] food_address = food_place['address'].to_list()
    address = []
    for i in food_address:
        address.append(i.split()[1])

    find_idx = []
    length = len(food_place)
    for i in range(length):
        if food_place['name'][i] in list(place_food_count.keys()):
            find_idx.append(i)

    place_address = []
    for i in find_idx:
        place_address.append(address[i])

    count = []
    for i in find_idx:
        count.append(place_food_count[food_place['name'][i]])

    regionFoodData = {'place_name' : food_place['name'][find_idx].to_list(),
                      'address' : place_address,
                      'count' : count
                      }
    df = pd.DataFrame(regionFoodData)
    df.to_csv("/content/drive/MyDrive/data/regionFoodData.csv", index=False)
    df.to_excel("/content/drive/MyDrive/data/regionFoodData.xlsx", index=False)
```

만든 파일을 불러와서 어떤 구, 군이 있는지 확인한 후에 구, 군별로 검색 건수를 구한다.

만든 파일을 불러온다.

```
[10] regionTravel = pd.read_csv('/content/drive/MyDrive/data/regiontravelData.csv')
      regionFood = pd.read_csv('/content/drive/MyDrive/data/regionFoodData.csv')
```

전체 데이터의 주소를 확인해서 어떤 구와 군이 있는지 확인한다.

```
▶ travelRegion = np.unique(regionTravel['address'])
  foodRegion = np.unique(regionFood['address'])
  print(travelRegion)
  print(len(travelRegion))
  print(foodRegion)
  print(len(foodRegion))

Region = []
if len(travelRegion) < len(foodRegion):
    Region = foodRegion
else:
    Region = travelRegion

13 ['강서구' '금정구' '기장군' '남구' '동구' '동래구' '부산진구' '사하구' '서구' '수영구' '연제구' '영도구'
14      '중구' '해운대구']
15 ['강서구' '금정구' '기장군' '남구' '동구' '동래구' '부산진구' '북구' '사상구' '사하구' '서구' '수영구'
16      '연제구' '영도구' '중구' '해운대구']
```

관광지, 음식점 데이터를 이용해 구와 군별 검색 건수를 구한다.

```
[14] region = []
      count = []

      for re in Region:
          result = 0
          idx = regionFood[regionFood.address == re].index
          regionFoodcount = list(map(int, regionFood['count'][idx].to_list()))

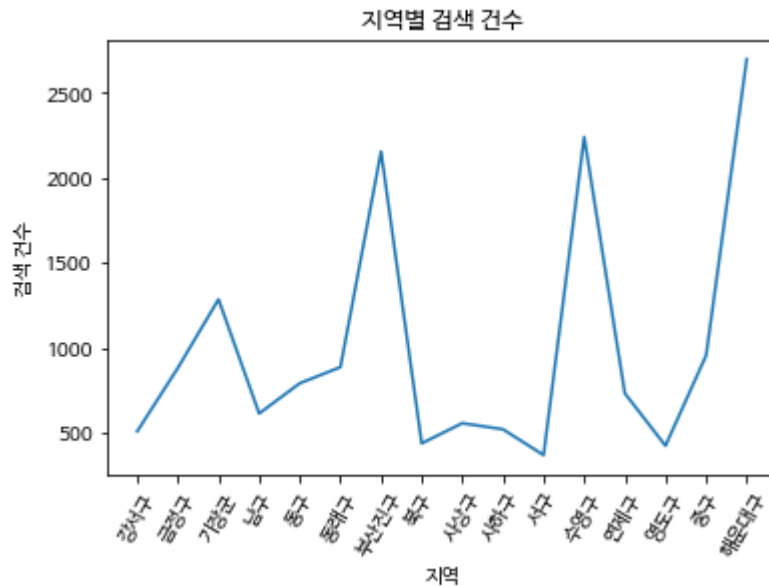
          idx = regionTravel[regionTravel.address == re].index
          regionTravelcount = list(map(int, regionTravel['count'][idx].to_list()))

          for i in regionFoodcount:
              result += i
          for i in regionTravelcount:
              result += i

          region.append(re)
          count.append(result)

      regionData = {'region_name' : region,
                    'count' : count
                    }
      df = pd.DataFrame(regionData)
      df.to_csv('/content/drive/MyDrive/data/regionData.csv', index=False)
      df.to_excel('/content/drive/MyDrive/data/regionData.xlsx', index=False)
```

결과를 그래프로 표현하면 다음과 같다.



### 3. 인기 관광지 순위 데이터

33쪽에 전체 관광지와 음식점을 점수 매길 때 따로 3개월간의 기간을 두고 수집한 데이터의 점수를 이용한다.

```
# 인기 관광지 추천에 사용
food_score_list_pd = {'place' : food_place,
                      'score' : food_score_list}
food_score_list_pd = pd.DataFrame(food_score_list_pd)
food_score_list_pd.to_csv("/content/drive/MyDrive/data/food_score_list.csv", index=False)
food_score_list_pd.to_excel("/content/drive/MyDrive/data/food_score_list.xlsx", index=False)

# 인기 관광지 추천에 사용
travel_score_list_pd = {'place' : travel_place,
                       'score' : travel_score_list}
travel_score_list_pd = pd.DataFrame(travel_score_list_pd)
travel_score_list_pd.to_csv("/content/drive/MyDrive/data/travel_score_list.csv", index=False)
travel_score_list_pd.to_excel("/content/drive/MyDrive/data/travel_score_list.xlsx", index=False)
```

---

### 3.3. 웹페이지 구성

#### 3.3.1. 메인 페이지

svg를 활용한 부산 지도를 통해 구별로 감성 분석 결과 상위 3개 관광지를 볼 수 있다.

또한 감성 분석 결과 상위 8개의 관광지를 이미지와 함께 보여주며 상세보기 클릭 시 세부 사항을 볼 수 있는 페이지로 이동하여 볼 수 있다.

맨 아래에는 현재 개최되고 있는 축제를 배치하였고 마찬가지로 '자세히 보기'를 클릭 시 해당 축제의 상세정보를 확인할 수 있다.

#### 3.3.2. 부산 지도를 통한 관광지 목록 시각화

카카오맵 API의 지도와 마커를 사용하여 관광지, 음식점의 위치를 시각화 하였다.

검색창을 통해 관광지, 음식점을 검색할 수 있고 이에 대한 상세 정보를 볼 수 있다.

관광지 또는 음식점을 출발지, 경유지, 도착지로 설정하여 차량으로 이동 시 최적 경로를 볼 수 있다.

#### 3.3.3. 부산과 인근 지역 축제 목록 제공

tourApi 3.0를 통하여 1년간 열리는 축제 이름, 축제 기간(시작일과 종료일), 주최단체, 주최자 전화번호, 주소, 개최 장소, 이용 요금, 상세 정보 등을 수집하였다.

부산과 인근 지역(울산광역시, 경상남도)에서 진행되고 있거나, 진행 예정인 축제의 목록을 보여주며 클릭 시 해당 축제에 대한 상세 정보를 볼 수 있다.

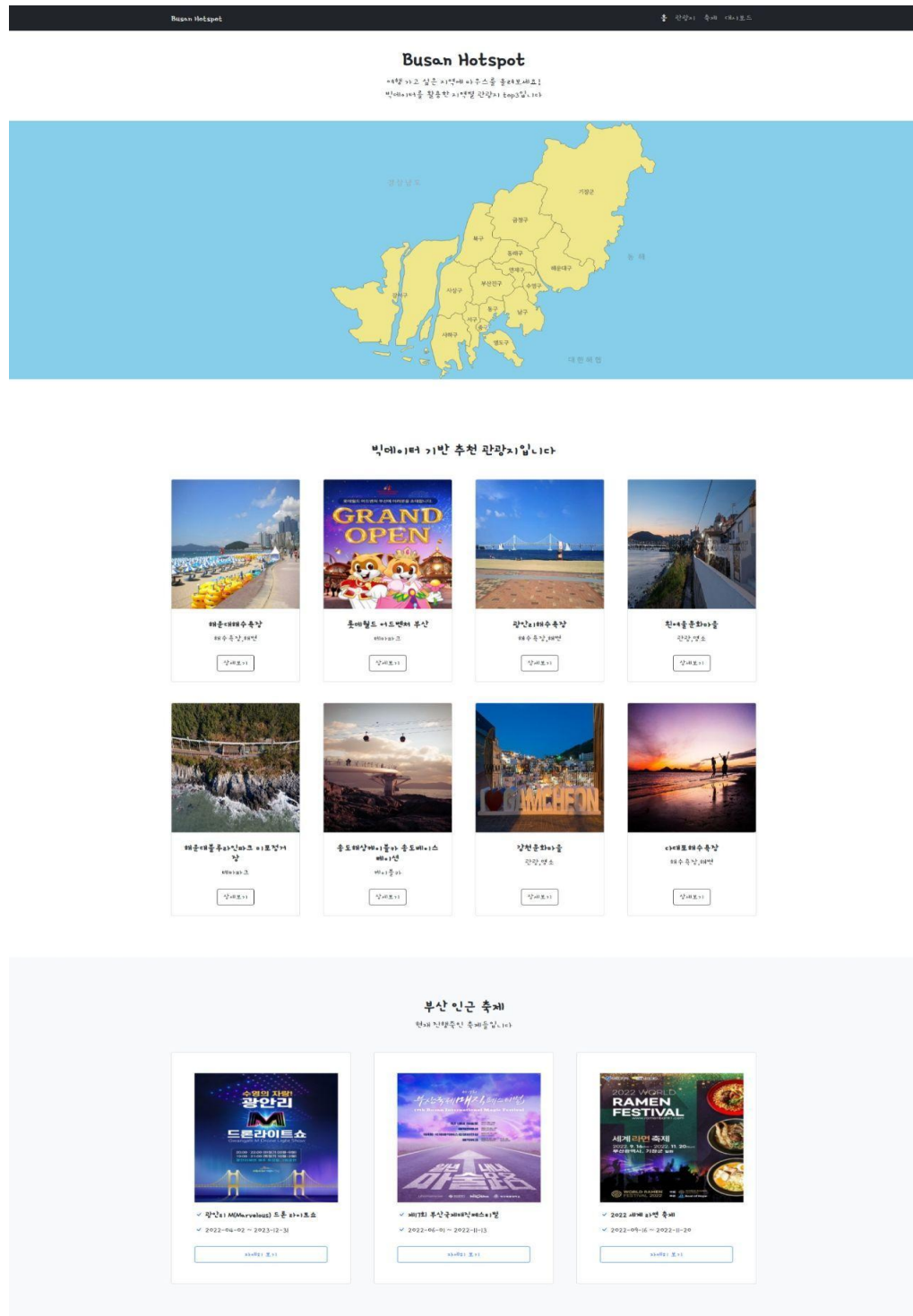
#### 3.3.4. 감성분석의 시각화

대시보드를 통하여 각종 차트를 활용하여 데이터 수집과 감성분석 결과를 시각화 하였다.

우선 바 차트를 통해 3개월간 요일 별 리뷰 수 평균을 시각화 하여 나타냈고 파이 차트로 각 지역(구, 군)이 몇 번 검색되었는지 비율을 표현하여 부산에서 인기있는 구, 군이 어디인지를 확인할 수 있도록 하였다. 맨 아래에는 테이블을 구현하여 각 관광지, 맛집 별 평점이나 리뷰 수, 감성 분석으로 얻은 스코어를 시각화 하여 나타냈다. 테이블의 분류를 누르면 분류 별 오름차순이나 내림차순으로 관광지, 맛집을 볼 수 있다.

## 4. 연구결과

### 홈화면



---

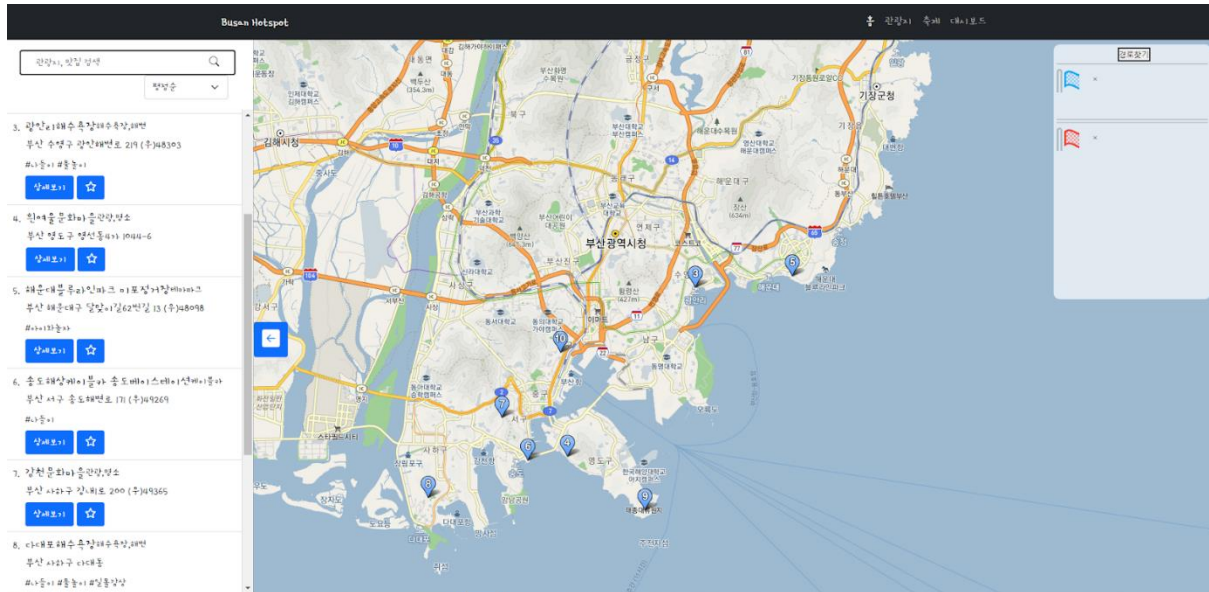
네비바의 홈을 클릭하거나 첫 화면에서 홈화면을 볼 수 있으며 지도에 마우스 오버 시 감성 분석으로 얻은 탑 3 관광지를 볼 수 있다.

지도의 구, 군 지역을 클릭하면 해당 지역의 관광지를 볼 수 있고 관광지 상세보기를 클릭하면 관광지의 상세보기를 볼 수 있다.

마찬가지로 부산 인근 축제도 자세히 보기를 누르면 상세히 볼 수 있는 화면으로 이동한다.



## 관광지 화면



네비바에서 관광지를 누르면 관광지 화면으로 이동할 수 있다.

한 페이지에 8개의 관광지를 보여주며 상세보기 클릭 시 아래 그림과 같이 관광지에 대한 상세정보를 확인할 수 있다.

해운대해수욕장해수욕장,해변	롯데월드 어드벤처 부산테마파크
<div> <div>부산광역시 해운대구 해운대해변로 264</div> <div> <div>이용시 주의 사항</div> <div>개장기간 - 2022년 6월 2일 ~ 8월 31일</div> </div> <div>051-749-7612</div> <div> <a href="http://www.haeundae.go.kr/tour/i...">http://www.haeundae.go.kr/tour/i...</a> </div> <div>#나들이#물놀이</div> <div>예약, 배달</div> <div> <div>카카오맵</div> </div> </div>	<div> <div>부산 기장군 기장읍 동부산관광로 42 (우)46083</div> <div> <div>매일 10:00 ~ 21:00</div> <div>1661-2000</div> <div> <a href="https://adventurebusan.lotteworld...">https://adventurebusan.lotteworld...</a> </div> <div>#아이와놀자</div> <div>예약가능</div> <div> <div>카카오맵</div> </div> </div> </div>

또한 관광지 사이의 경로를 알고 싶으면 관광지의 별 모양 버튼을 클릭하여 찾고 싶은 두개 이상의 관광지를 선택하고 '경로 찾기' 버튼을 눌러 확인할 수 있다.




## 축제 화면

Busan Hotspot


홈관광지축제대사보드

부산 근처 Festival List




광안리 M(Marvelous) 드론 라이트쇼

2022-04-02 ~ 2023-12-31




제17회 부산국제매직페스티벌

2022-06-01 ~ 2022-11-13



2022 세계 라면 축제

2022-09-16 ~ 2022-11-20



[문화관광축제] 울산용기축제

2022-09-30 ~ 2022-10-03

네비바에서 축제를 누르면 축제화면으로 이동할 수 있으며 부산 근처(부산, 울산, 경남)의 축제 리스트들을 확인할 수 있다.

한 페이지에 6개의 축제를 볼 수 있으며 현재 개최되고 있는 축제부터 나중에 개최되는 축제로 순서가 나열되어 있다.

## 광안리 M(Marvelous) 드론 라이트쇼

부산광역시 수영구 광안해변로 219(광안동) | 2022-04-02 ~ 2023-12-31

사진보기

상세정보

댓글



## 상세정보

행사소개:매주 토요일 저녁, 새롭고 다양한 콘텐츠로 광안리 밤하늘을 아름답게 장식하는<광안리 M 드론 라이트쇼>  
"M"은 '놀라운, 경이로운'이라는 뜻의 'Marvelous'를 의미하는 것으로 드론이 뿜어내는 불빛이 광안대교의 야경과 어우러져 광안리  
를 찾는 방문객들에게 경이로움과 신비로움을 선사할 것이다.

\* 코로나19 상황 및 기상상황(우천,강풍 등)에 따라 공연 일정 변경 가능  
행사내용:주제 및 계절에 맞는 300대 이상의 군집드론 비행



시작일	2022.04.02	종료일	2023.12.31
전화번호	051-610-4884	주소	부산광역시 수영구 광안해변로 219(광안동)
행사장소	광안리 해변 일원	주최	부산광역시 수영구
이용요금	무료		

댓글

○ ○

● ● ● ●

단위 글 입력

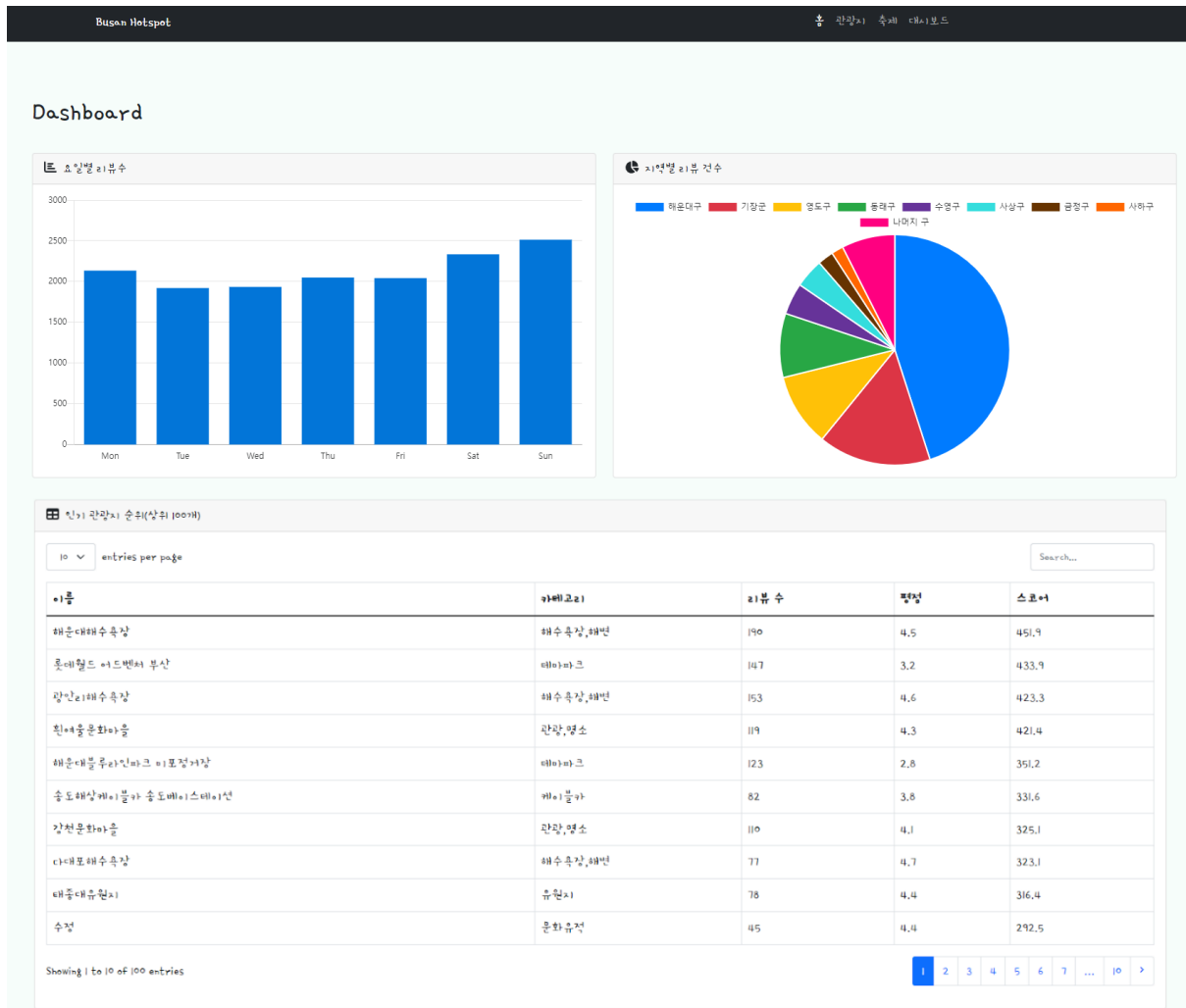
제출

댓글이 없습니다.

홈화면에서 축제 상세보기를 누르거나 축제 화면에서 리스트 중 하나를 클릭하면 축제 상세 화면으로 이동하게 되는데 위의 그림과 같이 상세정보와 위치, 주소 등 축제에 대한 상세한 정보를 확인할 수 있다.

아래에 댓글창을 두어 해당 축제를 방문했던 사람이 후기를 적을 수 있도록 구성하였다.

## 대시보드 화면



네비바의 대시보드를 클릭하면 대시보드 화면으로 이동한다.

대시보드 화면으로 데이터 분석한 결과를 차트로 볼 수 있으며 관광지순위 테이블로 각 관광지의 리뷰 수, 평점, 스코어에 따른 순위를 볼 수 있다.

---

## 5. 결론 및 향후 연구 방향

### 5.1. 결론

본 과제에서 관광지 추천 모델을 만들기 위해 감성분석 순환신경망 모델을 설계했다.

13만개의 관광지과 음식점 댓글을 가지고 LSTM 순환층을 사용하여 순환신경망을 학습시켰고

그 결과 정확도 80% 이상이고 이항 교차 엔트로피 손실함수가 0.4이하인 성능을 보여주었다.

이 모델을 가지고 최근 3개월 내에 작성된 댓글들을 모아 감성분석한 결과와 평점을 곱해서 점수를 합산하였고 가장 점수가 높은 장소 10곳을 추천하였다. 또 모델을 학습시키면서 구한 유의미한 데이터들을 대시보드에 표현하였다.

### 5.2. 향후 연구 방향

댓글에 대한 감성분석과 기존 관광지에 대한 평가를 기반으로 점수를 매길 수 있었지만, 음식점에 비해 관광지에 대한 댓글 양이 부족하여 같은 방식으로 점수를 매겼을 때, 관광지가 오히려 하위권으로 나타나는 현상이 보였다. 이는 다른 지도 플랫폼을 통해 관광지에 대한 댓글과 평점을 수집한다면 관광지를 주제로 하는 해당 프로젝트에 알맞은 결과를 볼 수 있을 것으로 판단된다.

또한 평점과 댓글 텍스트를 감성분석한 결과를 가지고 점수를 냈기 때문에 평점만 있는 댓글인 경우에도 점수를 낼 수 있는 방식을 구현해야 할 것 같다.

향후에는 위의 단점들을 보완해 전국 데이터를 활용해 여행 추천 사이트를 구현한다.

---

## 6. 참고 문헌

- [1] Django Documentation, Available: <https://docs.djangoproject.com/en/4.1/>
- [2] Bootstrap Documentation, Available: <https://getbootstrap.com/docs/5.2/getting-started/>
- [3] KoNLPy.org, Available: <https://konlpy.org/ko/latest/index.html>
- [4] kakaomap, Available: <https://map.kakao.com/>