

# 트렌드 분석 : NAVER BLOG

AI빅데이터융합경영학과 20222855 김차미  
AI빅데이터융합경영학과 20222892 이지민



\* 코드 설명 일부는 코드 캡처 이미지 내 주석으로 대체하였습니다.

Topic

Trend



# INDEX

01

스크래핑 데이터 수집

02

데이터 전처리

03

모델링 1

Word Cloud

04

모델링 2

LSA, LDA + TF-IDF

05

모델링 3

딥러닝

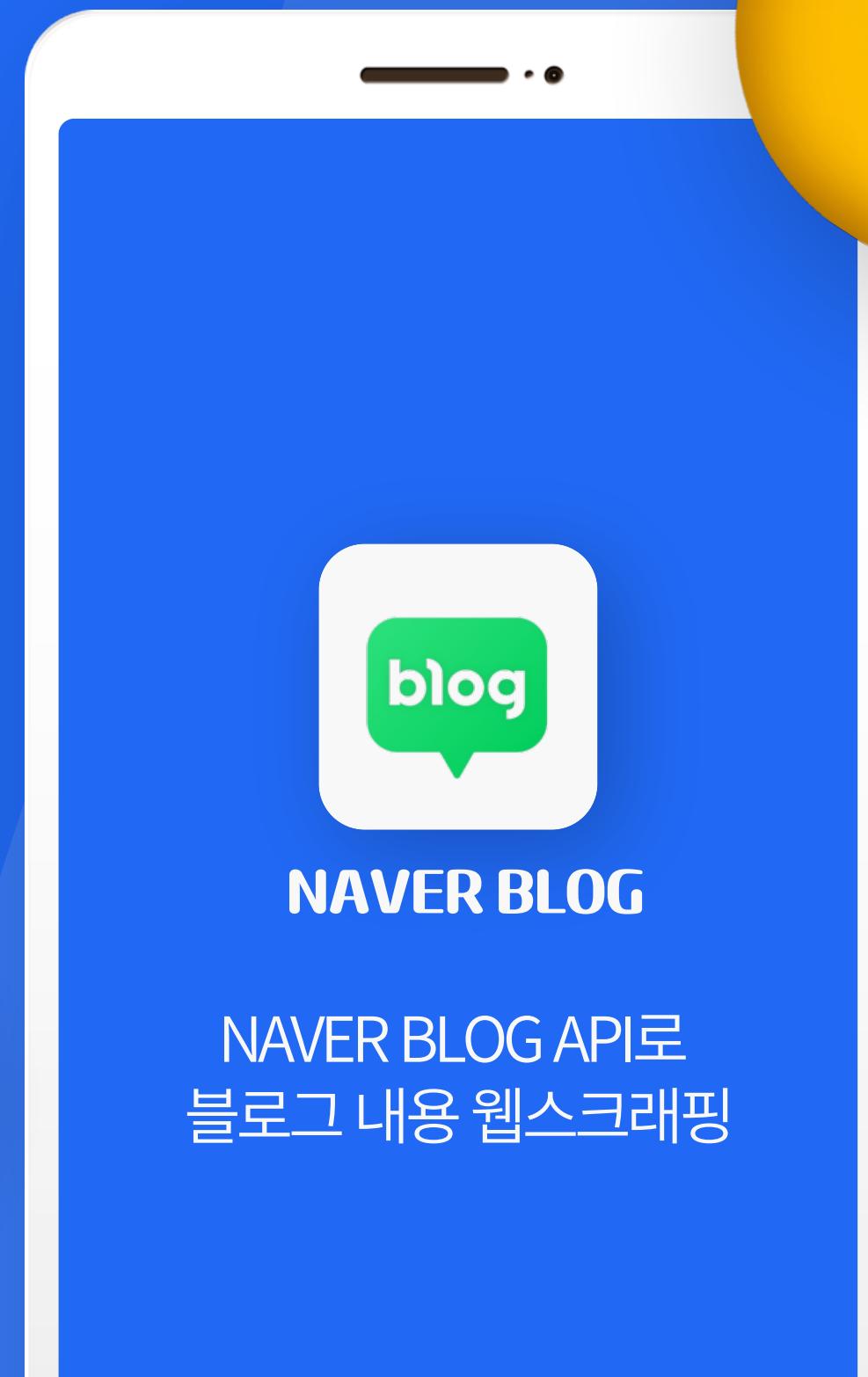
06

결론 및 한계점

최근 5년을 타겟으로  
**한 해를 돌아보자!**

유행어 / 트렌드 / 여행지  
드라마 / 영화 / 예능

- \* 트렌드는 14(10년 전), 23, 24년만 수집
- \* 드라마 / 영화 / 예능은 최근 3년(22-24년)만 수집





# 네이버 블로그 API

뉴스나 기사보다는 사람들이 본인의 일상이나 생각을 직접 공유하는  
블로그에 유행하는 것들이 많이 담겨져 있을 것이라고 생각  
→ [네이버 블로그](#)에서 데이터 수집

네이버 API는 한 번에 100개의 결과만 반환이 가능  
→ 10번에 나누어 요청을 보내서 한 번에 1000개 모으기

2000년대, 20-24년 유행어	연도마다 1000개씩	14, 23, 24년 트렌드	연도마다 1000개씩
20-24년 여행지	연도마다 1000개씩	22-24년 드라마	연도마다 500개씩
22-24년 영화	연도마다 500개씩	22-24년 예능	연도마다 500개씩



총 17,957개의 데이터 수집  
(중복 및 결측치 제외)

Oh-!



```

## 네이버 API는 한 번에 100개의 결과만 반환이 가능 -> So, 10번에 나누어 요청을 보내서 하루에 1000개 모으기
## 100개씩 10번의 요청을 보내 총 1000개의 블로그 포스트를 스크래핑

base_url = 'https://openapi.naver.com/v1/search/blog.json'
query = '2023년 유행어'
## 정렬 -> sim: 검색어 유사도 / date: 날짜순 (최신의 데이터 우선적으로 보여줌)
sort = 'sim'

def fetch_blogs(start_index):
    encQuery = urllib.parse.quote(query)
    url = f'{base_url}?query={encQuery}&display=100&start={start_index}&sort={sort}'
    my_request = urllib.request.Request(url)
    my_request.add_header("X-Naver-Client-Id", client_id)
    my_request.add_header("X-Naver-Client-Secret", client_secret)

    with urllib.request.urlopen(my_request) as response:
        if response.getcode() == 200:
            response_body = response.read()
            search_results = response_body.decode('utf-8') ## 검색 결과를 우리가 볼 수 있는 형태로 바꿔줌
            search_results = eval(search_results) ## 딕셔너리처럼 되어 있는 스트링을 실제 딕셔너리로 바꿔줌
            return search_results
        else:
            print("Error Code:", response.getcode())
            return None

    total_results = []
    for start in range(1, 1001, 100): # 1부터 1001까지 100씩 증가
        results = fetch_blogs(start)
        if results and 'items' in results:
            total_results.extend(results['items'])
        time.sleep(1) # 각 요청 사이 대기 시간 1초

    # 여기서 total_results에는 최대 1000개의 블로그 포스트 정보가 저장됨

```

검색 결과는 유사도 기준으로 정렬

```

## 태그를 지우는 함수

def remove_tag(my_str):
    p = re.compile('<([>]+)>')
    return p.sub('', my_str)

for item in total_results:
    print("제목 전처리 전: ", item['title'])
    print("제목 전처리 후: ", html.unescape(remove_tag(item['title'])))
    print("링크 전처리 전: ", item['link'])
    print("링크 전처리 후: ", html.unescape(item['link'].replace('\\\\'','')))

    print()

```

## 제목과 링크에 간단한 전처리 수행

```

driver = webdriver.Chrome()

all_results = dict()
i = 0

for item in total_results:
    link = html.unescape(item['link']).replace('\\\'','')
    postdate = item['postdate']
    bloggername = item['bloggername']
    bloggerlink = item['bloggerlink']
    description = html.unescape(item['description'])

    if 'blog.naver.com' in link:
        all_results[i] = dict()

    # scraping하려는 웹페이지 주소를 get()에 전달
    driver.get(link)

    try:
        driver.switch_to.frame('mainFrame')
        # 제목 추출하기
        title = driver.find_element(By.CLASS_NAME, 'pcoll').text
        # 본문 추출하기
        body = driver.find_element(By.CLASS_NAME, 'se-main-container').text.replace('\n', ' ')
    except Exception as e:
        title = "제목을 찾을 수 없습니다"
        body = "본문을 찾을 수 없습니다"

    all_results[i]['link'] = link
    all_results[i]['postdate'] = postdate
    all_results[i]['title'] = title
    all_results[i]['body'] = body
    all_results[i]['bloggername'] = bloggername
    all_results[i]['bloggerlink'] = bloggerlink
    all_results[i]['description'] = description

    i += 1

```

Selenium을 사용하여 블로그 페이지에 접근하고,  
필요한 데이터를 수집  
본문 외에도 활용 가능한 정보들도 함께 수집

## 수집한 데이터

**포스트링크, 작성일, 제목, 본문, 작성자,  
블로그링크, 요약 or 내용 일부**

# 데이터 전처리

```
# 전처리할 칼럼들
columns = ['body', 'description', 'title']

# 중복 제거
df.drop_duplicates(subset=columns, inplace=True)

for column in columns:
    # 정규표현식 - 한글, 숫자, 영어 및 공백 이외의 문자 제거
    df[column] = df[column].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣0-9a-zA-Z]", "", regex=True)
    # 앞뒤 공백 제거
    df[column] = df[column].str.strip()
    # 여러 개의 공백을 하나의 공백으로 변경
    df[column] = df[column].str.replace(' +', " ", regex=True)

# 공백만 있는 칼럼을 NaN으로 변경 후 제거
df[columns] = df[columns].replace('', np.nan)
df.dropna(subset=columns, how='any', inplace=True)
```

```
from kiwiapi import Kiwi

# Kiwi 초기화
kiwi = Kiwi()

# 불용어 리스트 정의 및 파일 로드
additional_stopwords = [
    '개봉작', '2023년', '영화', '개봉', '감독', '작품', '수익', '관객', '배우', '출연', '박스오피스',
    '이야기', '관람', '한국', '흥행', '누적', '기록', '국내', '주연', '예정', '액션', '생각', '극장',
    '시리즈', '상영', '연출', '사람', '연기', '시작', '드라마', '대비', '소개', '출처', '순위', '올해',
    '정보', '공개', '개봉일', '최고', '신규', '추천', '후기', '출연진', '주차', '신작', '제목', '연속',
    '리뷰', '순위', '상영', '줄거리', '평점', '결산', '정리', '돌파', '작품', '주말', '데하', '사랑',
    '세계', '보이', '기대', '내용', '이후', '제작비', '가족', '장르', '모습', '영화제', '정도', '장면',
    '주인공', '때문', '상황', '이름', '이미지', '기준', '구글', '이후', '원작', '오늘', '기대작', '주말',
    '프로필', '예고편', '거리', '베스트', '시간표', '성적', '스포', '등극', '필모그래피', '최신', '역대', '결말',
    '대학', '마지막', '달려', '캐릭터', '등급', '사건', '제작', '기간', '타임', '러닝', '시상식', '시기',
    '전체', '시청', '극장가', '슬램', '포스팅', '예매', '최초', '배급', '시상', '문단속', '공식',
    '등장인물', '영상', '부문', '안내', '선정'
]

# 불용어 텍스트 파일에서 불용어 읽기
def load_stopwords(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        stopwords_list = file.read().splitlines()
    return stopwords_list

# 파일에서 불용어 목록 로드
stopwords_file_path = 'stopwords.txt'
file_stopwords = load_stopwords(stopwords_file_path)

# 불용어 리스트 결합
all_stopwords = set(additional_stopwords + file_stopwords)
```

1

**postdate, body, description, title** 칼럼만  
사용해서 분석 진행

2

**기본적으로 수업 자료 코드 참고  
+ 필요한 부분 수정**

3

**중복 제거 / 정규표현식 사용 / 공백 처리의  
전처리 과정을 1차적으로 수행**

4

**불용어는 구글링으로 찾은 stopwords.txt와  
이후 결과를 보고 무의미한 단어들을 불용어로  
직접 정의**

# 데이터 전처리

```
# 불용어 제거 함수 정의
def remove_stopwords(text, stopwords):
    tokens = kiwi.tokenize(text)
    filtered_tokens = [token.form for token in tokens if token.form not in stopwords]
    return ' '.join(filtered_tokens)

# 전처리 함수 정의
def preprocess_korean(text, analyzer=kiwi, stopwords=all_stopwords):
    my_text = copy.copy(text)
    my_text = my_text.replace('\n', ' ') # (1) 줄바꿈 문자 제거
    my_text = analyzer.space(my_text) # (2) 띄어쓰기 교정
    sents = analyzer.split_into_sents(my_text) # (3) 문장 토큰화
    p = re.compile('[^ㄱ-ㅎㅏ-ㅣ가-힣]')
    all_result = []
    for sent in sents:
        token_result = remove_stopwords(sent.text, stopwords) # (4) 형태소 분석 및 불용어 제거
        token_result = p.sub(' ', token_result) # (5) 특수 문자 제거 (=한글을 제외한 문자 제거)
        all_result.append(token_result) # (6) 형태소 분석한 결과를 다시 join

    all_result = ' '.join(all_result) # (7) 모든 문장을 하나의 string으로 join
    return all_result

# 품사(명사, 동사, 형용사, 부사) 추출 함수 정의
def wordclass_korean(my_str, kiwi=kiwi):
    result = []
    tokens = kiwi.tokenize(my_str, normalize_coda=True)
    for token in tokens:
        if token.tag in ['NNG', 'NNP', 'NNB']: # 명사 태그만 추출
            result.append(token.form)
    result = ' '.join(result)
    return result
```

5

불용어 제거 함수 정의

6

텍스트에서 불용어와 특수 문자를 제거하고  
띄어쓰기를 교정

7

명사만을 추출해서 사용

8

body, description, title 칼럼에 전처리 진행해서  
데이터프레임에 열 추가  
→ 데이터프레임 csv로 저장

# 전처리한 칼럼들의 단어 빈도 계산

```
# 단어 빈도 계산 및 상위 단어 추출
def explode_and_count(df, column):
    exploded = df[column].str.split().explode() # 문자열을 단어 단위로 나누어 행으로 펼침
    word_counts = exploded.value_counts().reset_index() # 단어 빈도 계산
    word_counts.columns = ['word', 'count'] # 컬럼명 설정
    return word_counts
```

```
# 각 칼럼에 대해 단어 빈도 계산
word_counts_list = []
for column in ['preprocessed_body']:
    word_counts = explode_and_count(df, column)
    word_counts_list.append(word_counts)

# 모든 칼럼의 단어 빈도 합산 및 정렬
all_words_body = pd.concat(word_counts_list).groupby('word').sum().reset_index().sort_values(by='count', ascending=False)

# 한 글자 단어 제외
all_words_body = all_words_body[all_words_body['word'].str.len() > 1]

# 상위 30개 단어 출력
top_30_words = all_words_body.head(30)
top_30_words
```

## POINT 01

한글자 단어는 무의미하다고 판단  
→ 두글자 이상의 단어만 출력

## POINT 02

상위 30개의 단어를 출력해서 결과 확인  
→ 이때 빈도수는 높은데 무의미하다고  
판단되는 단어들은 불용어로 추가

이러한 과정을 모든 토픽에 동일하게 수행

# 모델링 1 - Word Cloud

문서의 키워드, 개념 등을 직관적으로  
파악할 수 있게 핵심 단어를  
시각적으로 돋보이게 하는 기법

```
## 데이터 프레임을 딕셔너리 형태로 변환해야 함  
  
dic_word = all_words_body.set_index('word').to_dict()['count']  
dic_word
```

```
font_path='C:\\Windows\\\\Fonts\\\\malgun.ttf'
```

한글 폰트 지정

```
## colormap 참고 사이트: https://wonhwa.tistory.com/20  
  
wc = WordCloud(random_state = 123, font_path = font_path, width = 400,  
               height = 400, background_color = 'black',  
               colormap = 'Blues')  
  
img_wordcloud = wc.generate_from_frequencies(dic_word)  
  
plt.figure(figsize = (10, 10)) # 크기 지정하기  
plt.axis('off') # 축 없애기  
plt.imshow(img_wordcloud) # 결과 보여주기
```

연도별

트렌드 토픽,

유행어,

인기여행지,

인기 드라마/영화/예능

의 결과를 기대함

앞에서 계산한 단어 빈도를 기반으로 워드 클라우드 생성

결과 확인하고 추가적으로 불용어 지정해서 최종 워드클라우드 생성

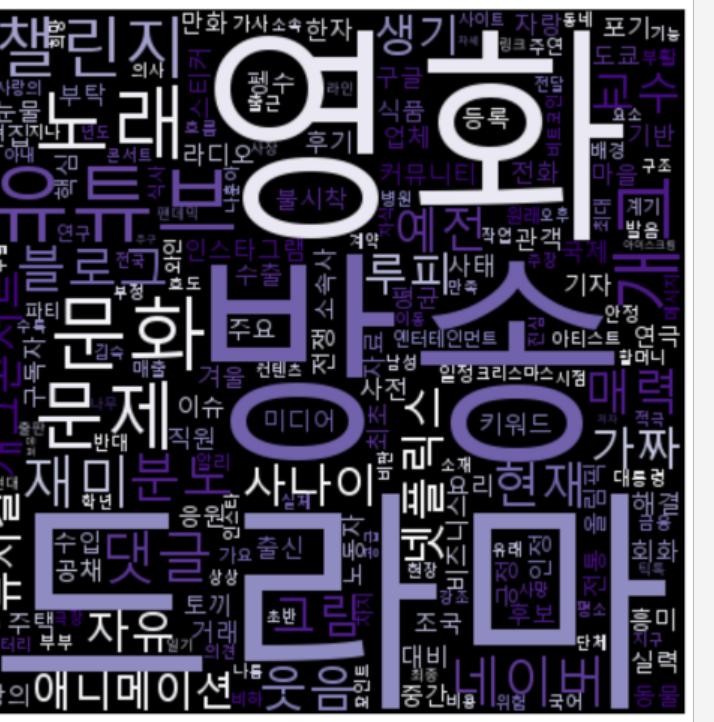
\* 워드클라우드 결과는 body/description/title 모두 뽑아서 제일 유의미한 것으로 선정



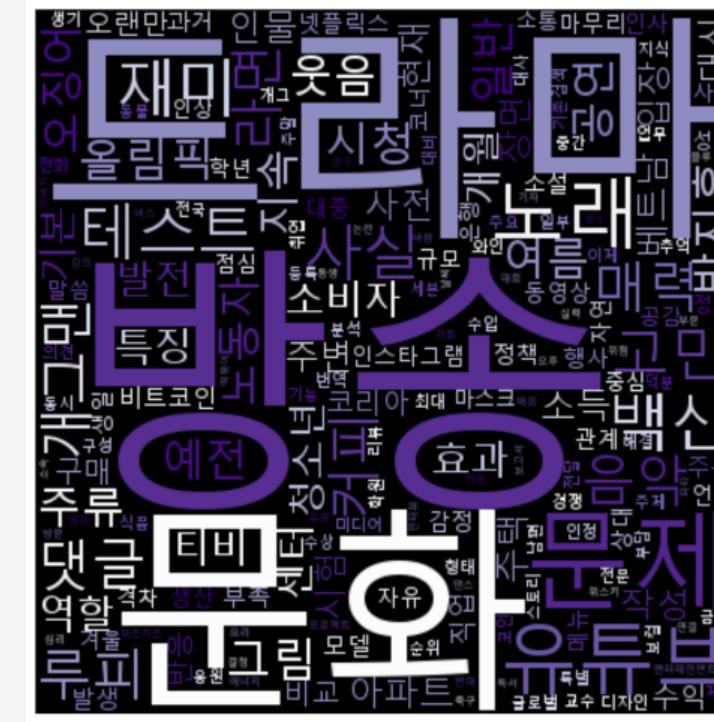
## 2000년대 유행어



## 2020년 유행어



## 2021년 유행어



## 유행어 분석에서의 어려움

### 1. 블로그 웹 스크래핑이라...

블로그 웹 스크래핑

→ 일상 단어 ↑

→ 불용어 처리를 일일이 확인해야함

(이 점은 유행어 뿐만 아니라 모든 주제가 마찬가지)

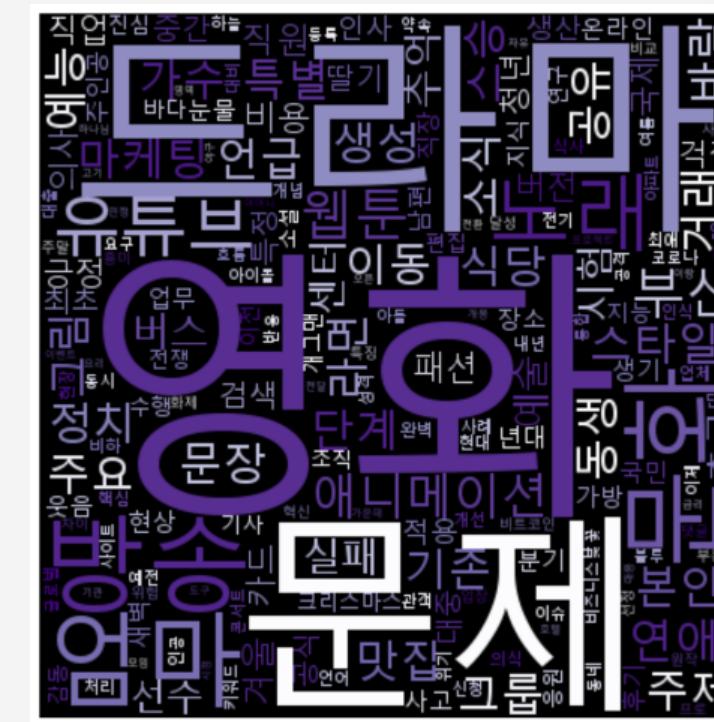
## 2022년 유행어



## 2023년 유행어



## 2024년 유행어



### 3. 워드 클라우드가 원하는 결과가 아니다...!

앞선 문제들로 인해서 워드 클라우드가 왼쪽 그림들과 같이 유행어가 전혀 보이지 않는 결과가 나타남

```
# 불용어 리스트 정의 및 파일 로드
additional_stopwords = ['유행어', '신조어', '2020년', '생각', '올해', '시간', '사람', '코로나', '대한', '시작',
    '일본', '사용', '한국', '사랑', '때문', '중국', '사진', '미국', '사회', '영상', '단어',
    '의미', '영상', '단어', '표현', '블로그', '정도', '인기', '배우', '모습', '이야기',
    '활동', '세대', '부모', '출처', '세계', '마음', '친구', '가지', '유행', '행복',
    '작품', '출연', '이후', '차트', '상황', '다양', '관련', '최고', '멤버', '제공',
    '가능', '게임', '대상', '처음', '일상', '요즘', '보이', '여행', '진행', '프로그램',
    '시대', '앨범', '소개', '성공', '느낌', '정보', '콘텐츠', '네이버', '오늘', '발매',
    '생활', '내용', '기업', '투자', '음악', '경제', '대표', '캐릭터', '시장', '광고',
    '트렌드', '이름', '부분', '엄마', '필요', '가족', '기억', '이유', '인터넷', '경우',
    '성장', '세상', '관심', '당시', '공부', '온라인', '감사', '마지막', '대회', '시즌',
    '등장', '기록', '사이트', '준비', '콘서트', '인하', '영어', '증명', '데뷔', '발표',
    '공개', '개인', '나라', '중요', '자리', '회사', '여자', '서울', '도전', '교육',
    '기준', '채널', '운동', '순위', '대한민국', '변화', '소비', '제품', '경험', '가격',
    '마스크', '판매', '감독', '축전', '기술', '포스팅', '국내', '학교', '여성', '하루',
    '스타', '소통', '영국', '나이', '유명', '이미지', '청년', '건강', '카페', '방법',
    '이웃', '연기', '활용', '지금', '어머니', '정리', '운영', '작년', '소리', '남자',
    '인생', '서비스', '제작', '정부', '중국어', '시절', '노력', '기사', '하나님', '개그맨',
    '예수', '가수', '그룹', '화제', '브랜드', '실패', '문장', '일본어', '이해', '거리',
    '']

additional_stopwords = ['유행어', '신조어', '2021년', '생각', '올해', '사람', '코로나', '대한', '시작', '일본',
    '사용', '한국', '사랑', '때문', '중국', '사진', '미국', '사회', '영상', '단어',
    '표현', '블로그', '정도', '인기', '배우', '모습', '이야기', '활동', '세대', '부모',
    '출처', '세계', '마음', '친구', '가지', '유행', '행복', '작품', '출연', '이후',
    '상황', '다양', '관련', '최고', '멤버', '제공', '가능', '게임', '대상', '처음',
    '일상', '요즘', '보이', '여행', '프로그램', '시대', '소개', '성공', '느낌',
    '정보', '콘텐츠', '네이버', '오늘', '생활', '내용', '기업', '투자', '경제', '대표',
    '캐릭터', '시장', '광고', '트렌드', '이름', '부분', '엄마', '필요', '가족', '기억',
    '이유', '인터넷', '경우', '성장', '세상', '관심', '당시', '공부', '감사', '마지막',
    '대회', '시즌', '등장', '기록', '사이트', '준비', '인하', '영어', '데뷔', '발표',
    '공개', '개인', '나라', '중요', '자리', '회사', '여자', '서울', '도전', '교육',
    '기준', '채널', '운동', '순위', '대한민국', '변화', '소비', '제품', '경험', '가격',
    '기술', '포스팅', '국내', '학교', '여성', '하루', '나이', '유명', '이미지', '청년',
    '건강', '카페', '방법', '연기', '활용', '지금', '정리', '운영', '작년', '남자',
    '인생', '서비스', '제작', '정부', '중국어', '시절', '노력', '기사', '그룹', '브랜드', '일본어',
    '이해', '거리', '언니', '최근', '결과', '이용', '선택', '현실', '평가', '설명',
    '아래', '수업', '지역', '확인', '인간', '아빠', '예능', '목표', '네이버', '의미',
    '대학', '환경', '부동산', '국민', '해당', '음식', '능력', '주식', '블로그', '산업',
    '마케팅', '증가', '선물', '국가', '뉴스', '이름', '용어', '작가', '관리', '학생',
    '언어', '상승', '단계', '선수', '상품', '포함', '영화', '과학', '화제', '사건',
    '']
```

## 매 해마다 400개가 넘는 단어 불용어 처리...

- 인터넷 서치를 통해 한국어 텍스트 전처리에서 많이 활용되는 표준 불용어 txt 파일에  
블로그 텍스트 전처리에 주로 쓰이는 불용어를 합쳐서 [stopwords.txt](#)로 만들어 활용
- 각각의 주제에 대해 불용어 처리를 개별적으로 해야 할 필요성  
→ 일일이 워드 클라우드를 확인하면서 불용어를 처리 OR 단어 빈도수  
상위 500개의 단어를 추출해 일상적인 단어를 불용어 리스트에  
추가하는 방식으로 진행

## 그렇다면 유행어를 추려서 워드 클라우드를 만들면 어떨까?

- 연도별로 유행어를 직접 조사해 txt 파일 생성 (중복 제외 약 241개)
- 데이터 프레임에 전처리된 명사 단어 칼럼 속 단어들 중 txt 파일 안에 있는 유행어 단어만  
필터링해 워드 클라우드 생성
- 한계: 미처 조사하지 못한 유행어가 존재  
→ BUT 연도별 유행어의 흐름을 볼 수 있다면 이는 충분히 유의할 것이라 판단



**new**

유행어 (신조어) txt 파일

2000년대, 2020-2024년도의 유행어들을 나열한 txt파일을 사용해서 이 txt파일에 있는 단어만 사용한다.  
이를 워드클라우드로 시각화하여 결과를 확인하였다.



문제, 이제, 엄마라는 키워드는 일상에서 사용하는 것과 유행어가 합쳐져 크게 나온 것으로 판단되며 그 외 엽기, 오타쿠, 감성, 지대, 얼짱과 같은 복고풍의 키워드가 나타났다. 특히 현재 2000년대를 그리워하며 '2000년대 감성 유행어'라는 식의 포스팅이 많은 것과 '@@감성'과 같이 사람 이름과 함께 쓰이는 경우가 합쳐서 감성이라는 단어가 크게 나타난 것을 알 수 있다.



만남이라는 키워드는 자만추에서 비롯된 것으로 '자연스러운 만남 추구'라는 뜻이다. 또한 유튜브에서 흥행했던 '가짜사나이 2'에서 이근 대위의 '너 인성 문제있어?' 유행어는 문제라는 키워드가 크게 나타난 것에 대한 이유라고 판단했다. 하지만 모든 해에 문제라는 키워드가 많이 나온 것으로 보아 일상적으로 많이 쓰인다는 사실을 유의해야 한다.



얼죽아라는 유행어로 인해 커피, 아이스, 아메리카노가 나타났다. 무한도전에서 나온 무야호, 멀티 페르소나의 열풍의 부캐, 하현우가 'Lazenca, Save us'를 리메이크해 화제가 되면서 가사 속 '스스로 불러온 재앙..'에서 비롯된 스불재도 나타났다. 또한 2013년 방영된 마스터셰프 코리아가 2019년 다시보기 영상으로 업로드되면서 출연자 최강록의 말투인 '근데 이제 ##를 겠들인...'이 휴먼 강록체로 21년에 유행해 이제라는 키워드가 전년도보다 확 커진 것을 볼 수 있다.

## 2022년 유행어

## 2023년 유행어

## 2024년 유행어

2000년대, 2020-2024년도의 유행어들을 나열한 txt파일을 사용해서 이 txt파일에 있는 단어만 사용한다.  
이를 워드클라우드로 시각화하여 결과를 확인하였다.



2022년에는 특징적으로 고양이, 중꺾마, 잡채 등의 유행어가 크게 나타났다. 최근 다시 유행하고 있는 '꽁꽁 얼어붙은 한강위로 고양이가 걸어다닙니다.'라는 유행어로부터 나온 고양이, 2022 리그 오브 레전드 월드 챔피언십 인터뷰 제목에서 유래해 2022 FIFA 카타르 월드컵에서 확산된 '중요한 건 꺾이지 않는 마음' 중꺾마, SNS에서 유행한 '그 자체'라는 말이 변형된 유행어인 잡채 등이 나타난 것을 알 수 있다.



2023년에는 특징적으로 엄마라는 키워드가 매우 크게 나타난다. 이는 스트레이키즈라는 보이그룹의 한 멤버가 뉴진스라는 걸그룹의 'OMG'라는 곡 속 '오마오마갓'을 '엄마엄마가'라고 발음해 유행이 되었다. 그리고 또한 20년 유행한 한강 고양이 또한 유행이 계속되고 있는 것을 알 수 있으며 22~23년 방영한 더 글로리가 유행하면서 그 속 대사들인 연진, 축복, 혜정과 같은 키워드들이 나타났다.



다시 한강 고양이가 릴스에서 유행하면서 고양이가 계속해서 크게 나타나며 새롭게 나타난 키워드인 원영은 유행한지 얼마 되지 않아 작지만 '원영적 사고'와 같은 최신 유행어를 반영하는 것을 볼 수 있다. 또한 외국인이 농협은행을 발음한 것을 '너무 예쁘네요'라고 알아들었다는 편의점 알바생의 이야기에서 비롯된 유행어인 농협은행도 작게 나왔지만 다른 연도에 비해 크게 나온 것을 보아 2024년의 유행어임을 알 수 있다.

## 2014년 트렌드



클래식, 실용주의 등과 같은 키워드가 떠오른 2014년!  
2013년 '응답하라 1994'가 복고 유행을 몰고오면서 레트로  
스타일, 패션이 유행한 것으로 보인다.  
또한 연출이라는 단어도 눈에 띄는데 이는 2014년 10월 개봉한 '인터스텔라'의 영향으로 유명한 크리스토퍼 нароль 감독  
의 연출이 흥행을 일으켰다.

## 2023년 트렌드



헤어와 컬러가 새롭게 떠오른 2023년!  
코로나라는 키워드가 나타났으며 헤어, 컬러가 크게 등장했다.  
글로벌 시장 조사 전문 업체 NPD 그룹은 영국의 럭셔리 헤어케어 제품 판매가 펜데믹 이전보다 67% 증가했다고 발표했으며  
한국도 헤어 케어에 대한 관심이 늘어났다.  
헤어 케어와 염색에 대한 관심이 늘어남으로 인해 이러한 결과  
가 나온 것으로 판단된다.

## 2024년 트렌드



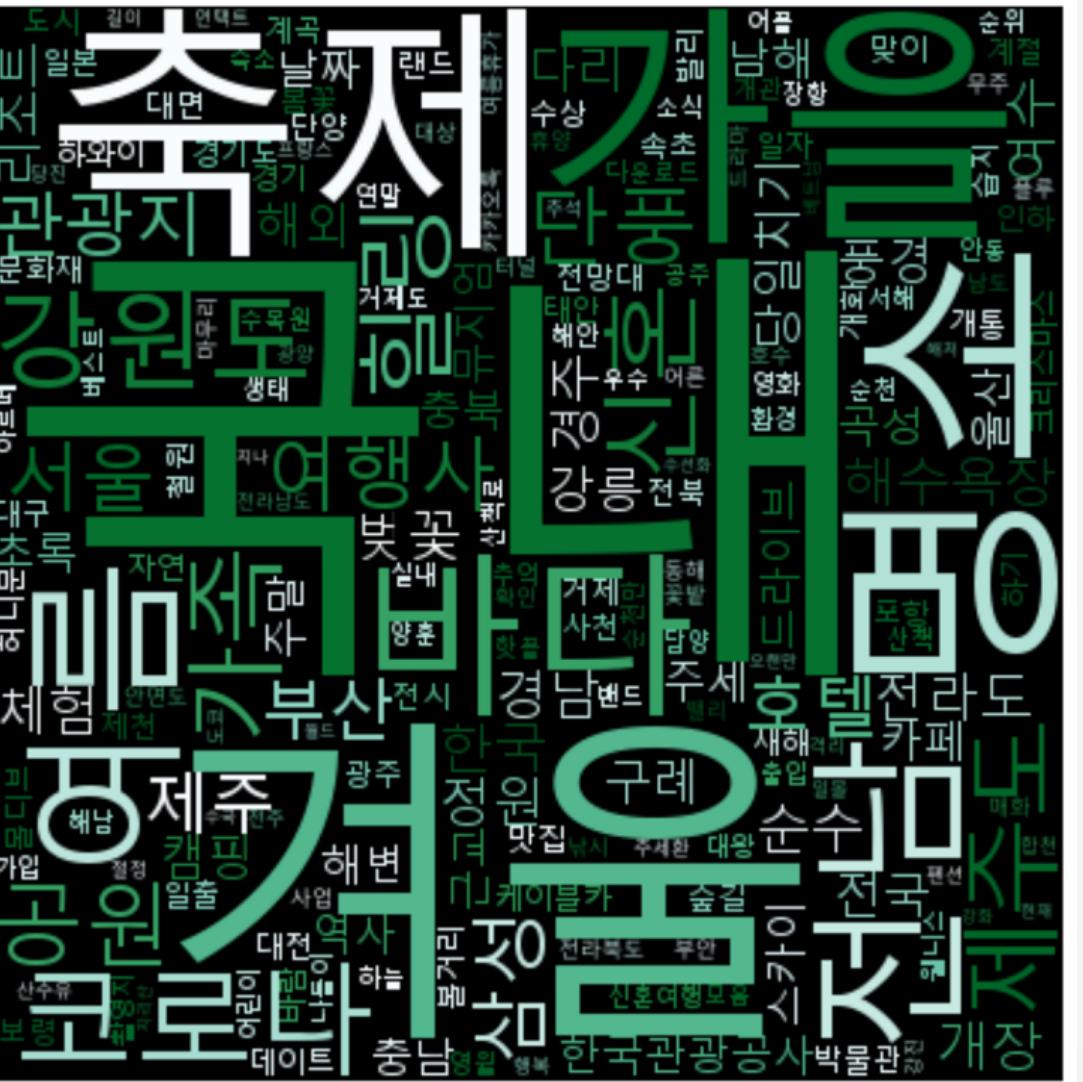
환경에 대한 관심이 점차 늘어나고 있는 2024년!  
2024년은 아직 끝나지 않았기 때문에 트렌드라는 키워드 자체  
에 관심이 많은 것으로 판단된다.  
특히 환경, 지속, 자연 등과 같은 키워드가 더 커진 것을 볼 수 있  
는데, 이는 코로나 이후 건강과 환경에 대한 니즈가 커지면서 환  
경보호와 관련된 단독주택 인테리어, 친환경 화장품, 친환경 패  
션 등에 대한 관심이 늘어난 것이다.

# 2020년 추천 여행지



2020년 1월 국내에서 코로나가 시작되면서  
연초인 겨울에는 여행을 다녀올 수 있었지만  
그 후엔 여행을 가지 못하는 상황이 벌어졌다.  
이로 인해 겨울이라는 키워드와 국내, 제주, 코로나라는  
키워드가 큰 비중을 차지한 것을 볼 수 있다.

# 2021년 추천 여행지



2021년에도 코로나의 영향은 계속되었다.  
하지만 코로나 19 백신 접종률이 빠르게 올라가면서  
국내여행이 전년도에 비해 늘어났으며  
제주, 전남, 강원도 등 국내 여행지 키워드들이  
눈에 띄게 늘어난 것을 볼 수 있다.

# 2022년 추천 여행지



2022년은 오랜 시간 끝에 실외 마스크가 해제되는 등  
코로나로 인한 갑갑한 일상에서 벗어나는 한 발짝을 디딘  
한해라고 볼 수 있다.  
이로 인해 그동안 하지 못했던 축제들이 다시 시작되었고  
실외 활동을 더 많이 하면서 자연과 함께하려는 움직임이  
늘어났고 공원, 체험, 풍경, 자연, 바다 등의 키워드들이  
나타났으며 특히나 축제라는 키워드가 가장 크게 나타났다.

## 2023년 추천 여행지

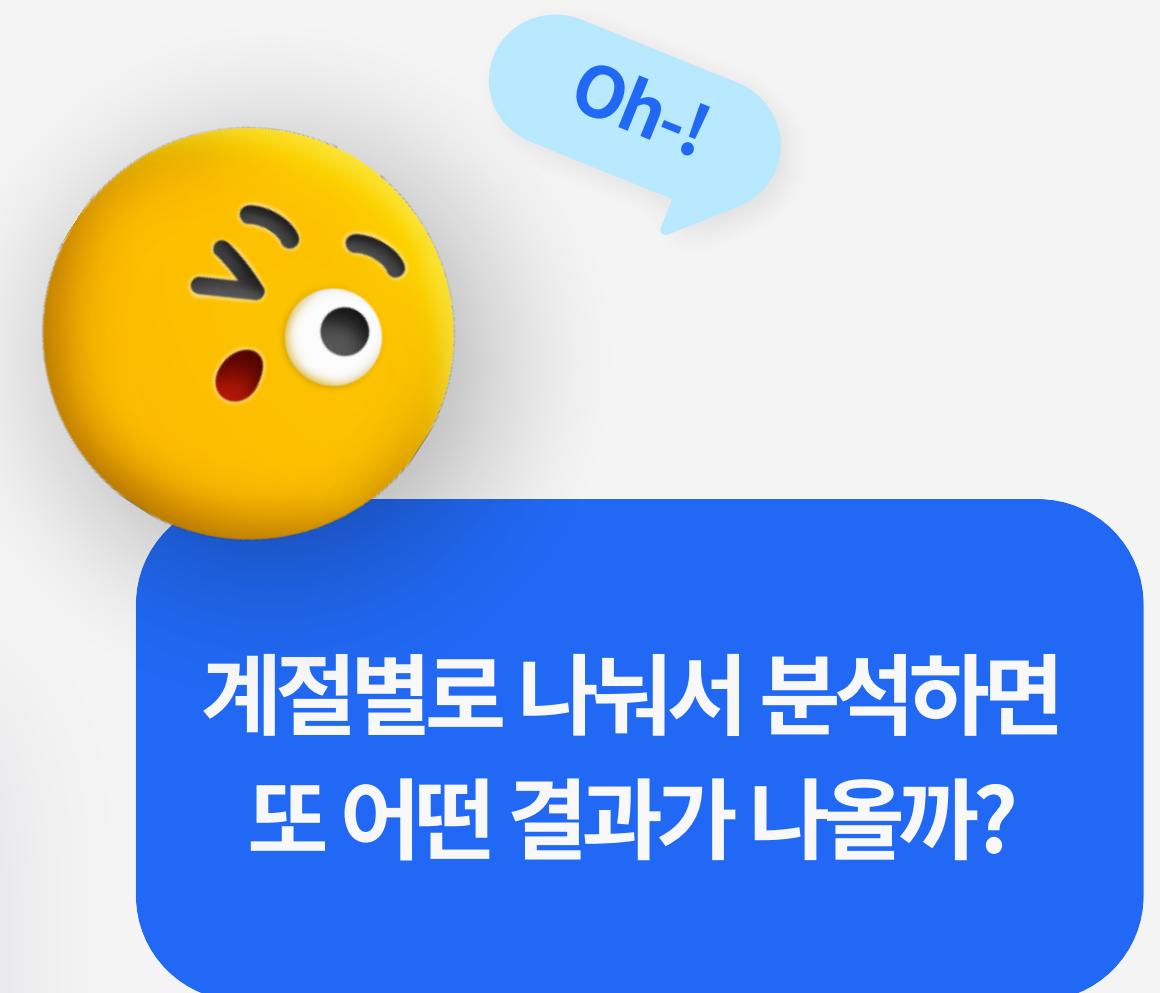


2023년 일본 환율 최저가 갭신되면서 일본 여행의 수요가 늘어났으며 일본이라는 키워드가 나타났다.  
또한 여전히 축제라는 키워드가 큰 비중을 차지했는데 코로나 때문에 하지 못했던 축제들이 점차 재개되고 늘어나면서 이에 따른 관심도 늘어난 것으로 알 수 있다.

## 2024년 추천 여행지

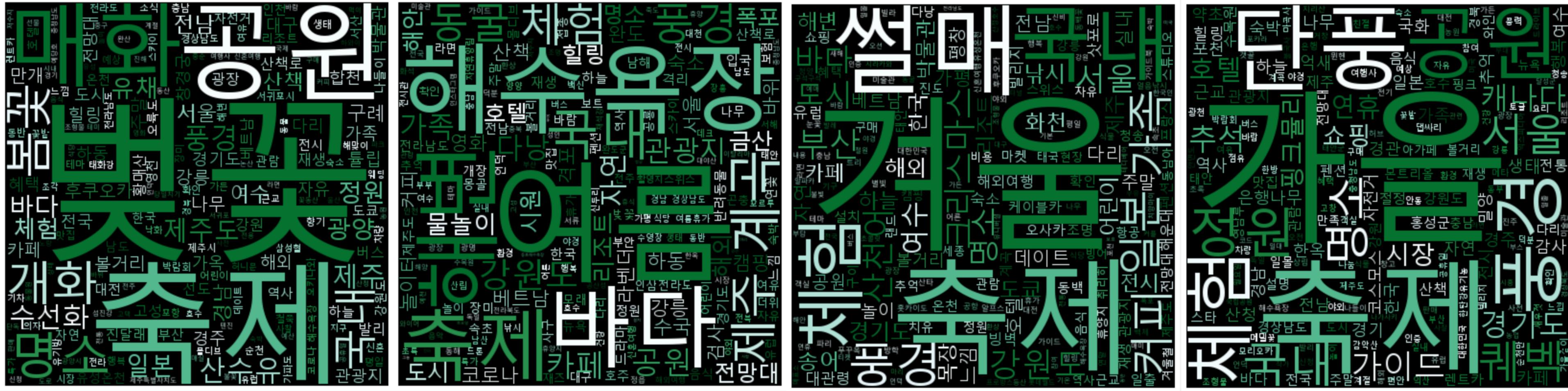


2024년에는 해외에 대한 관심이 늘어난 것을 알 수 있다.  
또 여전히 낮은 일본 환율 때문에 일본 여행에 대한 관심 또한 여전하며 현재 겨울, 봄만 지났기 때문에  
벚꽃, 개화, 겨울, 새해와 같은 키워드가 나타나는 것을 볼 수 있다.  
특히 코로나 이후 점차 시간이 지나면서  
더 다양한 여행지가 나타나는 것을 알 수 있다.



# 각 연도별 봄, 여름, 가을, 겨울

2020-2024년도의 여행지를 웹스크래핑한 결과를 사용해 각 계절별 키워드 ('봄', '여름', '가을', '겨울')만 각각 포함하게끔 데이터 프레임을 구성했다. 이를 워드클라우드로 시각화하여 결과를 확인하였다.

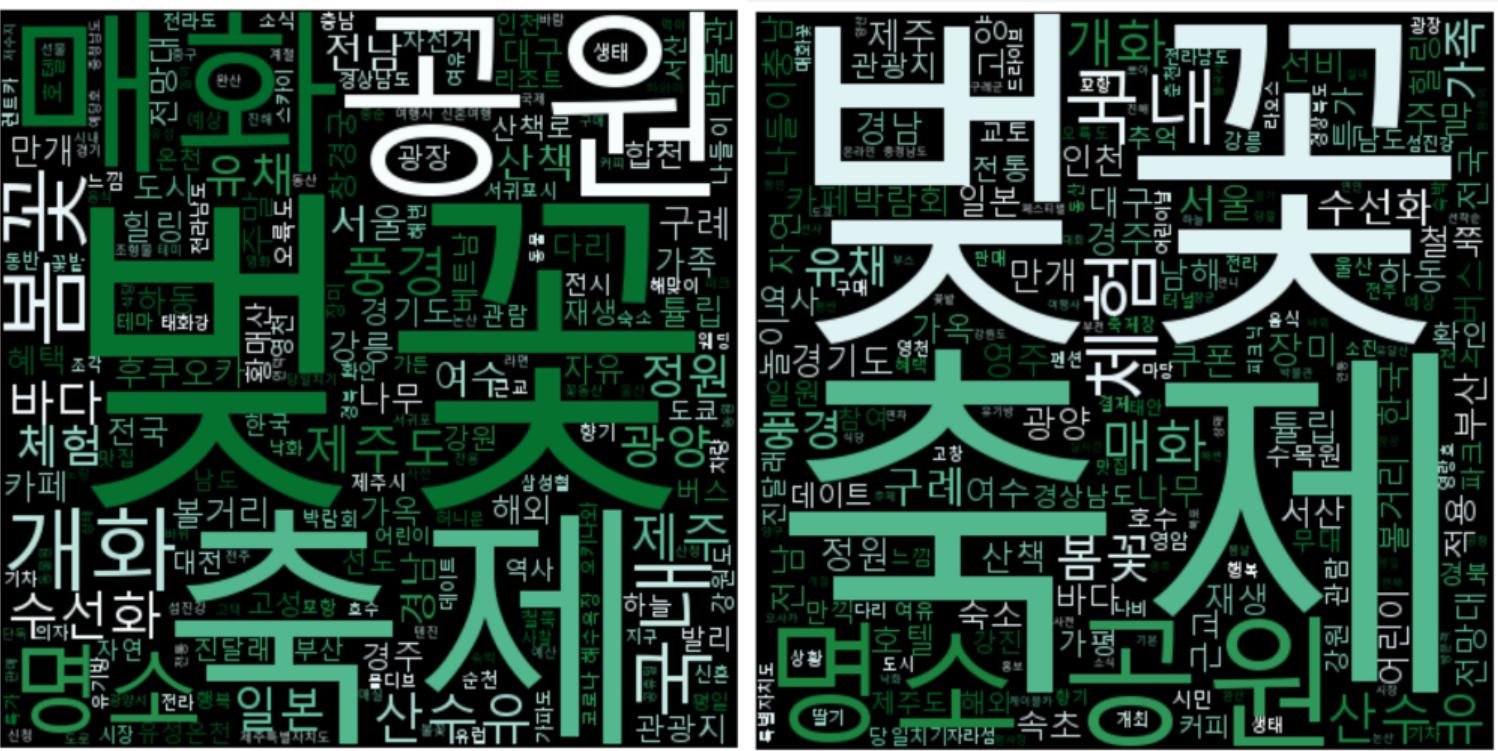




Spring



2020



2021



2022



2024

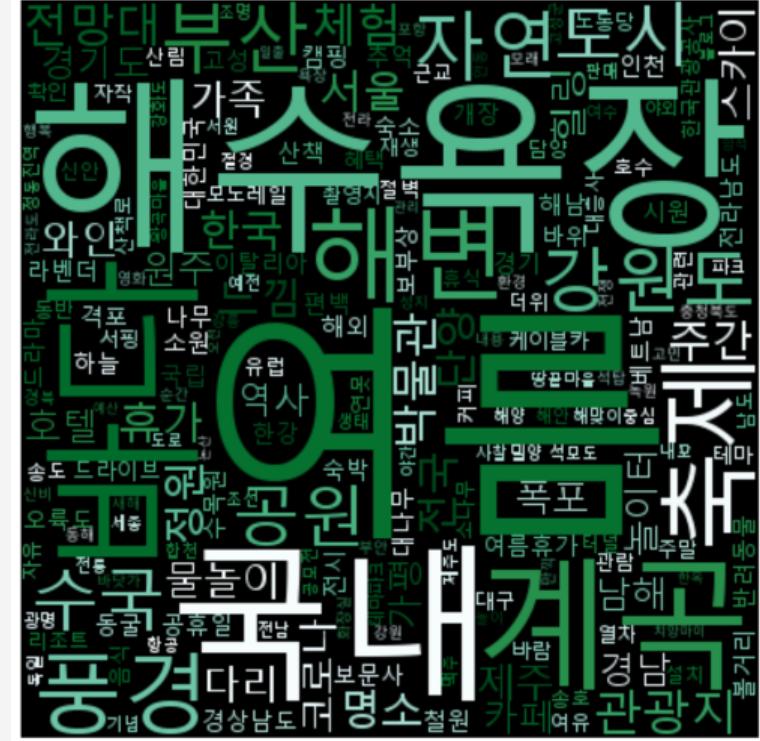
## 봄과 관련된 주요 키워드

벚꽃, 축제, 공원, 매화, 풍경, 툴립, 수선화, 체험,

명소, 유채, 정원, 봄꽃 등...



Summer



2020



2021



2022

## 여름과 관련된 주요 키워드

해수욕장, 계곡, 해변, 바다, 자연, 축제, 부산, 공원,

전망대, 폭포 등...



2023

2024



Autumn



2020

2021

2022

## 가을과 관련된 주요 키워드

단풍, 축제, 풍경, 공원, 명소, 수목원, 억새,  
코스모스, 꽃밭, 철원, 갈대밭 등...



2023

2024



Winter



2020

2021

2022

## 겨울과 관련된 주요 키워드

썰매, 축제, 케이블카, 호텔, 해외, 체험,  
크리스마스, 낚시, 도시, 바다, 풍경 등...



2023

2024



여행지가 나오는 것을 기대했으나  
이와 달리 여행지는 잘 나오지 않았다.  
추가 분석 진행

2020-2024년도의 여행지들을 나열한 txt파일을 사용해서 이 txt파일 내 프로그램 관련 단어만 사용한다.  
이를 워드클라우드로 시각화하여 결과를 확인하였다.



2020년은 다른 연도들보다 호텔이라는 키워드가 비중이 높은데 이는 코로나의 영향이 큰 것으로 판단할 수 있다. 또한 국내, 자연, 공원, 바다, 제주 등과 같은 키워드를 볼 수 있다.



2021년은 다른 연도들보다 전망대라는 키워드의 비중이 높은데 국내 여행이 늘어나면서 전망대에 대한 관심이 증가했다. 그외에도 카페, 해수욕장, 공원 등의 키워드를 볼 수 있다.



2022년에는 제주가 전년도보다 비중이 커졌다. 또한 서울과 강원도 키워드가 나타났으며 벚꽃, 단풍과 같은 봄, 가을의 계절적인 키워드도 볼 수 있다.



# 2023년 추천 여행지

# 2024년 추천 여행지

2020-2024년도의 여행지들을 나열한 txt파일을 사용해서 이 txt파일 내 프로그램 관련 단어만 사용한다.  
이를 워드클라우드로 시각화하여 결과를 확인하였다.



2023년에는 일본이라는 키워드가 전년도보다 훨씬 커졌으며  
이와 동시에 바다라는 키워드도 볼 수 있다.  
또한 부산, 강릉, 도쿄, 강원도, 경남, 전남, 속초 등과 같은  
구체적인 지역명들이 이전에 비해 많이 나타났다.



2024년은 역시 벚꽃, 매화, 정원, 개화 등  
봄과 관련한 키워드들이 가장 많았고  
특히 일본이라는 키워드가 전년도보다 더 커졌다.

# 2022년 방영 드라마



높은 시청률을 기록했던 <이상한 변호사 우영우>

→ 넷플릭스 시리즈로도 업로드

넷플릭스는 국내 OTT 1위 플랫폼

이외에도 OTT로 드라마를 시청하는 사람들이 많아지면서

티빙, 웨이브 등의 OTT 관련 키워드를 볼 수 있다.

또한 2022년 JTBC의 역대 드라마 시청률 1위를 찍은

<재벌집 막내아들>이 웹툰이 원작인 드라마였기 때문에

웹툰 키워드가 크게 나타난 것으로 예측된다.

# 2023년 방영 드라마



여전히 OTT 플랫폼 넷플릭스 크게 나타나고

이외에 티빙, 디즈니플러스, 웨이브도 보인다.

특히 갑자기 커진 디즈니플러스

→ 인기작이었던 무빙의 영향이 클 것이라고 판단

<모범택시>, <일타스캔들>, <연인>, <악구>

악귀의 작가인 김은희 작가 키워드도 꽤 크게 보인다.

2023년, 중국 드라마 업계는 다양한 변화와 도전 속에서

지속적인 발전을 이뤘다고 평가된다. → 크게 나타난 중국 키워드

## 2024년 방영 드라마



〈눈물의 여왕〉, 〈선재 업고 튀어〉, 〈내 남편과 결혼해줘〉,

<세작>, <수사반장> 등 인기 드라마 키워드 확인

## 갑자기 커진 웨이보, 텐센트 키워드

→ 중국의 선도적인 소셜 미디어 플랫폼들

중국과 웨이보, 텐센트의 키워드가 큰 것으로 보아

중국 드라마의 열풍이 불고 있는 것을 알 수 있다.

## 2022년 개봉작



## 2023년 개봉작



## 2024년 개봉작



<범죄도시2>, <아바타2> 천만영화

<쥬라기월드> 영화가 14년만에 실사로 개봉하며 화제가 되었다.

넷플릭스가 영화에서도 인기 있는 OTT 플랫폼

마블 영화의 인기 → <닥터스트레인지>, <블랙팬서>

추석에는 특선영화로 영화가 화제가 됨 → 추석 키워드

애니메이션 장르의 인기를 확인할 수 있다.

<범죄도시3>, <서울의 봄> 천만영화

마블 영화의 인기 → <앤트맨>

디즈니 픽사에서 만든 <엘리멘탈> 인기

<슬램덩크>, <스즈메의 문단속>

→ 일본 애니메이션 국내에서 큰 인기

추석 키워드가 여전히 큼

→ 추석에 영화의 화제성이 높다는 것을 알 수 있다.

2022, 2023년과 달리 5월 데이터까지만 수집했기에  
추석 키워드 사라짐

<범죄도시4>, <파묘> 천만영화

<찰리와 초콜릿 공장>을 원작으로 한

영화 <윙카>가 인기를 끌었다.

또한 OTT 넷플릭스의 여전한 영향력을 확인할 수 있다.

## 2022년 화제 예능



방송인 이효리 화제 → 2022년 국내 OTT 서비스 티빙에서 제작한 <서울체크인>이 큰 인기를 받으면서 주로 '드라마'와 '영화' 콘텐츠만 서비스하던 OTT 서비스에 예능방송이 자리매김을 하는 유행을 선도하였다. <나는 솔로>, <신발 벗고 돌싱포맨>, <환승연애> 등 연애와 결혼을 주제로 한 예능이 화제였다.

## 2023년 화제 예능



드라마와 영화 산업에 집중하던 OTT에서 예능에도 주력하기 시작했다. 넷플릭스 예능 <솔로지옥2>가 화제가 되었으며, 자체제작 예능까지 선보이는 티빙도 이용자가 급격히 늘어 워드클라우드에서 큰 비중을 차지하였다.

## 2024년 화제 예능



여전히 연예 프로그램이 대세인 것을 알 수 있다.



화제 예능 프로그램명의 결과를 기대했던 것과 달리, 프로그램 소재나 OTT 서비스에 집중된 결과가 나타났다.

추가 분석 진행

2022-2024년도의 예능 프로그램들을 나열한 txt파일을 사용해서 이 txt파일 내 프로그램 관련 단어만 사용한다.  
이를 워드클라우드로 시각화하여 결과를 확인하였다.



<환승연애>, <검은 양 게임>, <유 퀴즈 온 더 블럭>,  
<솔로지옥2>, <런닝맨>, <전지적 참견 시점>, <라디오스타>,  
<지구오락실>, <강철부대> 등 화제 예능을 볼 수 있다.



<환승연애>, <솔로지옥>, <최강야구>, <라디오스타>,  
<피지컬100> 등 화제 예능을 볼 수 있다.



<환승연애>, <연애남매>, <최강야구>, <라디오스타>,  
<나는 솔로> 등 화제 예능을 볼 수 있다.  
최근 3년동안 연애를 소재로 한 예능 프로그램이  
꾸준히 인기를 얻고 있다.

# 모델링 2 - LSA / LDA

## LSA

SVD를 이용해 Document Matrix를 분해하고 분해한 결과물로 Topic modeling 진행  
(문서-Topic 조합 행렬) X (Topic-단어 조합 행렬)

## [Topic Modeling]

말뭉치에서 잠재적인 Topic을  
발견하기 위한 모델링

```
# 상위 max_features개의 단어에 대해서만 vectorize -> 각각의 문서들을 벡터 형태로 만들기 위해서 사용
# max_df=0.5: 문서의 절반 이상에서 나타나는 단어를 무시 -> 자주 등장하지만 정보량이 적은 단어
vectorizer = TfidfVectorizer(max_features=300, max_df=0.5, smooth_idf=True)
X = vectorizer.fit_transform(df['preprocessed_body'])
```

상위 max\_features개의 단어에 대해서만 vectorize

```
model = TruncatedSVD(n_components=30, n_iter=100, random_state=42)
model.fit(X)
```

TF-IDF matrix에 Truncated SVD를 적용하여 LSA 수행  
n\_components=토픽 수

```
def get_keyword_by_topic(components, feature_name, n=5):
    for idx, topic in enumerate(components):
        sorted_keyword_idx = np.argsort(components[idx])[-1:-n-1:-1]
        sorted_keyword_result = [(feature_name[x], topic[x].round(4)) for x in sorted_keyword_idx]
    print(f'Topic {idx}: {sorted_keyword_result}')
```

각 topic에 대해서 상위 n개의 큰 값을 갖는 단어를 출력

\* 여행지를 제외한 모든 토픽에 대해 LSA/LDA 모델링 수행



# 모델링 2 - LSA / LDA

## LDA

문서가 다양한 Topic의 혼합으로 구성된다는 가정 하에 확률 모델을 사용해 Topic을 추정하는 기법

[Topic Modeling]  
말뭉치에서 잠재적인 Topic을  
발견하기 위한 모델링

```
# preprocessed_body에 있는 단어를 리스트에 넣은 칼럼 생성  
df['word_list'] = df['preprocessed_body'].apply(lambda x: x.split())
```

전처리한 칼럼에 있는 단어를 리스트에 넣은 칼럼 생성

```
from gensim import corpora  
word_dict = corpora.Dictionary(df['word_list'])  
corpus = [word_dict.doc2bow(text) for text in df['word_list']]
```

gensim 라이브러리를 사용하여 단어 사전을 만들고, 이 사전을 바탕으로  
각 문서에 대해 Bag-of-Words 표현을 생성

```
N_TOPICS = 30  
ldamodel = gensim.models.ldamodel.LdaModel(corpus,  
                                              num_topics = N_TOPICS, ## 주제 몇 개 할 건지  
                                              id2word=word_dict) ## vocabulary set
```

```
topics = ldamodel.print_topics(num_words=5) ## 각 토픽들에 대해서 몇 개의 단어(num_words)로 표현할 것인지  
## 대표적인 단어를 알 수 있음  
for topic in topics:  
    print(topic)
```

topic 수 설정해서 LDA 모델 훈련 후 각 topic의 대표 단어들을 출력

```
def get_topic_keywords(ldamodel, num_keywords=5):  
    topic_keywords = {}  
    for topic_id in range(ldamodel.num_topics):  
        keywords = [word for word, _ in ldamodel.show_topic(topic_id, topn=num_keywords)]  
        topic_keywords[topic_id] = keywords  
    return topic_keywords  
  
def get_topic_ratio_for_each_document(ldamodel, corpus):  
    topic_keywords = get_topic_keywords(ldamodel)  
    results = []  
    for i, topic_list in enumerate(ldamodel[corpus]):  
        sorted_topic_list = sorted(topic_list, key = lambda x: x[1], reverse=True)  
        most_important_topic, most_important_ratio = sorted_topic_list[0]  
        second_most_important_topic, second_most_important_ratio = sorted_topic_list[1] if len(sorted_topic_list) > 1 else (None, 0)  
  
        keywords_1 = ", ".join(topic_keywords[most_important_topic])  
        keywords_2 = ", ".join(topic_keywords[second_most_important_topic]) if second_most_important_topic is not None else ''  
        doc_result = [i, most_important_topic, most_important_ratio, sorted_topic_list, keywords_1, keywords_2]  
        results.append(doc_result)  
  
    results = pd.DataFrame(results, columns=['문서 번호', '가장 비중이 높은 토픽', '가장 높은 토픽의 비중', '각 토픽의 비중', 'Top1_Topic_keyword', 'Top2_Topic_keyword'])  
    return results
```

각 topic 별 키워드를 볼 수 있는 함수

[‘개성’ ‘개인’ ‘건강’ ‘게임’ ‘경제’ ‘광고’ ‘구매’ ‘구축’ ‘네일’ ‘다양’ ‘대하’ ‘도시’ ‘도전’ ‘독특’  
 ‘디자인’ ‘디지털’ ‘레드’ ‘맞춤’ ‘매력’ ‘메이크업’ ‘모바일’ ‘문화’ ‘미국’ ‘분위기’ ‘뷰티’ ‘브라운’ ‘브랜드’  
 ‘블루’ ‘사례’ ‘선택’ ‘수익’ ‘스마트’ ‘스타일’ ‘스타일링’ ‘스튜디오’ ‘시장’ ‘신발’ ‘아트’ ‘액세서리’ ‘얼굴’  
 ‘여행’ ‘연출’ ‘염색’ ‘영화’ ‘온라인’ ‘욕실’ ‘음악’ ‘인기’ ‘인테리어’ ‘일본’ ‘일상’ ‘자연’ ‘작품’ ‘적극’  
 ‘전통’ ‘조명’ ‘중국’ ‘지속’ ‘출처’ ‘취향’ ‘컬러’ ‘코로나’ ‘코리아’ ‘콘텐츠’ ‘클래식’ ‘투자’ ‘트렌드’  
 ‘트렌디’ ‘패션’ ‘포인트’ ‘표현’ ‘피부’ ‘핑크’ ‘한국’ ‘헤어’ ‘화이트’ ‘화장품’ ‘환경’ ‘활용’ ‘흐름’]

80

```
1 model = TruncatedSVD(n_components=3, n_iter=100, random_state=42)
2 ## 연도별 데이터프레임을 합쳤기 때문에 14년, 23년, 24년으로 나눔
3 model.fit(X)

TruncatedSVD(n_components=3, n_iter=100, random_state=42)
```

벡터 차원을 80으로, 토픽 수를 3으로 설정

→ 3개의 연도를 합쳤기 때문에 각 연도를 대표하는 키워드가 있을 것이라고 예상

Topic 0: [('트렌드', 0.5157), ('패션', 0.3395), ('컬러', 0.3264), ('스타일', 0.2822),  
 ('디자인', 0.2476), ('대하', 0.1553), ('활용', 0.1533),  
 ('시장', 0.1515), ('다양', 0.1505), ('브랜드', 0.1471)]  
 Topic 1: [('컬러', 0.4729), ('패션', 0.3371), ('스타일', 0.2982), ('헤어', 0.2446),  
 ('연출', 0.1297), ('스타일링', 0.0829), ('디자인', 0.0619),  
 ('염색', 0.0593), ('분위기', 0.058), ('피부', 0.0528)]  
 Topic 2: [('패션', 0.6464), ('스타일', 0.2787), ('브랜드', 0.094), ('액세서리', 0.0695),  
 ('클래식', 0.0455), ('인기', 0.038), ('지속', 0.0373),  
 ('개성', 0.0347), ('신발', 0.0345), ('디자인', 0.0251)]

## 결과 해석

Topic 0: [('트렌드', 0.5157), ('패션', 0.3395), ('컬러', 0.3264), ('스타일', 0.2822), ('디자인', 0.2476), ('대하', 0.1553), ('활용', 0.1533), ('시장', 0.1515), ('다양', 0.1505), ('브랜드', 0.1471)]

Topic 1: [('컬러', 0.4729), ('패션', 0.3371), ('스타일', 0.2982), ('헤어', 0.2446), ('연출', 0.1297), ('스타일링', 0.0829), ('디자인', 0.0619), ('염색', 0.0593), ('분위기', 0.058), ('피부', 0.0528)]

Topic 2: [('패션', 0.6464), ('스타일', 0.2787), ('브랜드', 0.094), ('액세서리', 0.0695), ('클래식', 0.0455), ('인기', 0.038), ('지속', 0.0373), ('개성', 0.0347), ('신발', 0.0345), ('디자인', 0.0251)]

각 토픽의 키워드들이 서로 비슷하고 뚜렷하고 토픽이 구분되지 않는 것을 볼 수 있다.

이는 LSA가 단순히 단어의 빈도와 분포를 기반으로 하기 때문에, 특징적으로 중요한 단어들을 파악하지 못해 주제의 명확성이 비교적 떨어진다.

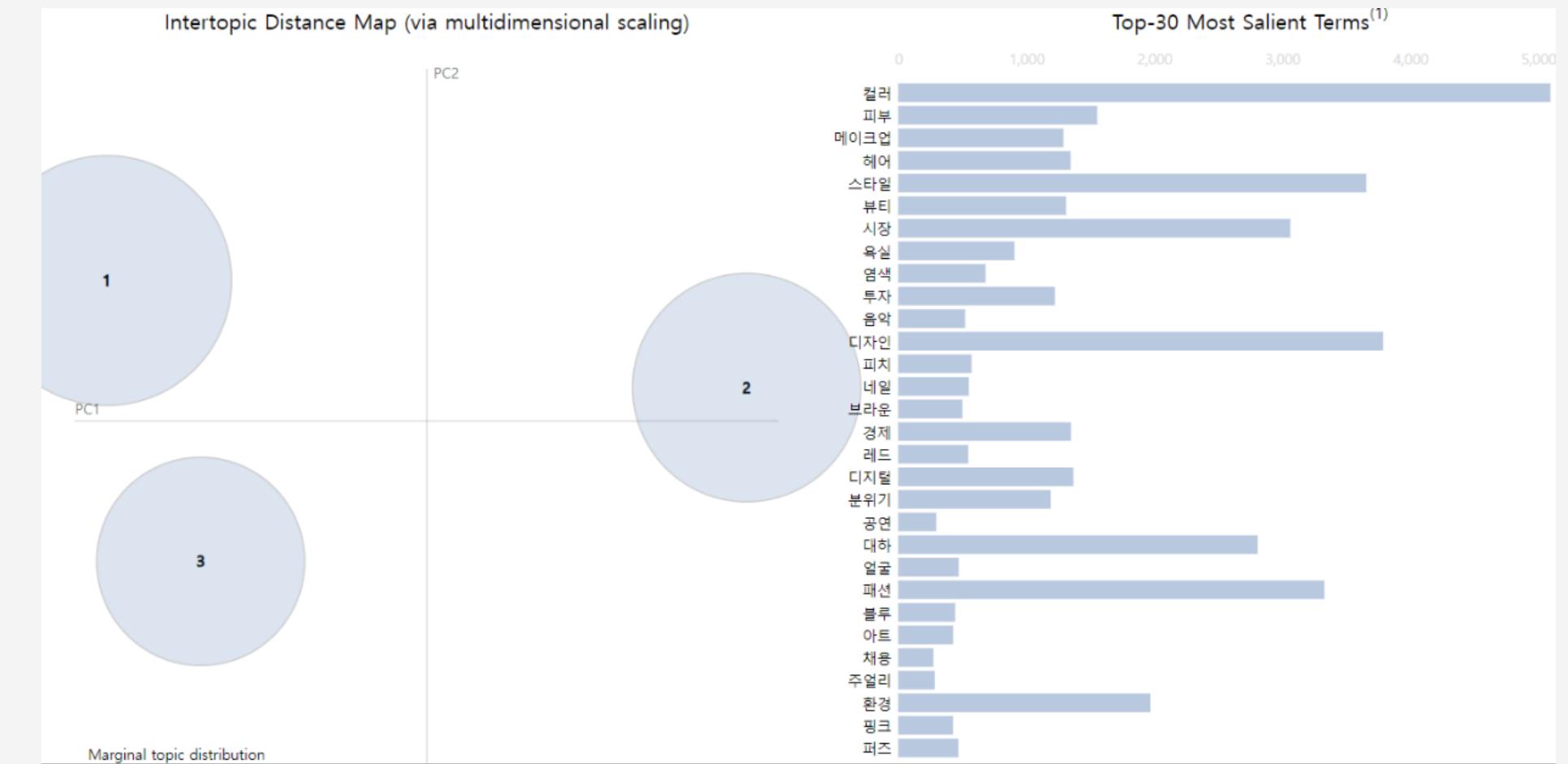
또한 트렌드의 키워드 자체의 모호함도 있어 추가적인 불용어 처리와 데이터 수집이 필요해보인다.

```

N_TOPICS = 3
ldamodel = gensim.models.ldamodel.LdaModel(corpus,
                                              num_topics = N_TOPICS, ## 주제 몇개할 건지
                                              id2word=word_dict) ## 보카불러리 셋

(0, '0.015*"트렌드" + 0.015*"디자인" + 0.014*"시장" + 0.008*"패션" + 0.008*"스타일"
 + 0.007*"대하" + 0.007*"투자" + 0.007*"욕실" + 0.005*"다양" + 0.005*"디지털"')
(1, '0.032*"트렌드" + 0.010*"대하" + 0.009*"활용" + 0.009*"환경" + 0.008*"다양"
 + 0.008*"시장" + 0.007*"지속" + 0.006*"개인" + 0.006*"브랜드" + 0.006*"디자인"')
(2, '0.033*"컬러" + 0.019*"트렌드" + 0.017*"스타일" + 0.011*"패션" + 0.011*"피부"
 + 0.009*"헤어" + 0.009*"메이크업" + 0.009*"디자인" + 0.009*"뷰티" + 0.008*"인테리어"')

```



## 결과 해석

연도별로 키워드의 연관성이 크게 없어서 LDA에서도 크게 그런 연관성이 나타나진 않았지만 비슷한 유형의 키워드끼리 잘 분류된 것을 볼 수 있다.  
하지만 처음부터 뚜렷한 트렌드가 눈에 띄지 않았던 것, 불용어 처리의 어려움 등과 같은 한계점을 넘을 수는 없었다.  
때문에 추가적인 불용어 처리와 블로그 이외의 트렌드 주제와 관련된 웹 스크래핑을 진행하는 것이 필요해보인다.

# 2000년대, 2020-2024년 유행어

[ '간지' '감성' '갑툭튀' '거북이' '겠냐' '고고씽' '고양이' '고진감래' '궁금' '귀여니' '기절' '꼬마' '꾸웨액' '꾸웨엑' '낭만' '너뭐돼' '농협은행' '뉴진스' '당모치' '대박' '댕댕이' '동료' '디토' '띵작' '레게노' '레알' '루돌프' '루삥뽕' '만남' '맑눈광' '맛꿀마' '머선' '먹방' '모에' '무야호' '문제' '웡미' '부캐' '분좋카' '브라보' '빙고' '빵꾸뚱꾸' '삼커다' '소고기' '손민수' '휩살재빙' '스라밸' '스불재' '신나' '신뢰' '쌉파서블' '아가씨' '아메리카노' '아이스' '알잘딱깔센' '어사' '어쩔티비' '억텐' '얼죽아' '얼짱' '엄마' '여신' '연진' '엽기' '오운완' '오이시쿠나레' '오타쿠' '우왕' '원영' '이유' '이제' '인성' '일취월장' '잡채' '재질' '잼민' '잼민이' '저메추' '저짤' '점메추' '존귀' '줄귀' '중꺾마' '중꺾마' '지대' '지못미' '쫌쫌따리' '찐텐' '초딩' '추구미' '축복' '캘박' '커피' '킹받네' '킹받드라슈' '킹왕짱' '하이' '한강' '혜정' '흉대' ]

100

```
model = TruncatedSVD(n_components=6, n_iter=100, random_state=42)
## 연도별 데이터프레임을 합쳤기 때문에 2000년대, 20년, 21년, 22년, 23년, 24년으로 나눔
model.fit(X)
```

```
1 get_keyword_by_topic(model.components_, terms)

Topic 0: [('문제', 0.9907), ('이제', 0.071), ('엄마', 0.0517), ('신뢰', 0.0482), ('동료', 0.0472)]
Topic 1: [('엄마', 0.9283), ('감성', 0.1975), ('궁금', 0.1816), ('이제', 0.1467), ('만남', 0.1028)]
Topic 2: [('감성', 0.6693), ('궁금', 0.5728), ('이제', 0.2732), ('동료', 0.0957), ('만남', 0.0687)]
Topic 3: [('궁금', 0.7796), ('신뢰', 0.0334), ('동료', 0.0135), ('오타쿠', 0.0088), ('소고기', 0.0062)]
Topic 4: [('이제', 0.8103), ('동료', 0.4206), ('신뢰', 0.086), ('만남', 0.0673), ('고양이', 0.0477)]
Topic 5: [('동료', 0.8635), ('만남', 0.1849), ('감성', 0.0618), ('고양이', 0.0444), ('신뢰', 0.0296)]
```

벡터의 차원을 100으로 설정

연도를 다 합쳤기 때문에 토픽 수를 연도 수인 6개로 설정

## 실제 각 연도의 유행어 top 10과 비교

2000년대:	문제, 감성, 엄마, 이제, 오타쿠, 만남, 신뢰, 낭만, 궁금, 동료
2020년:	문제, 감성, 궁금, 동료, 한강, 만남, 고양이, 뉴진스, 아이스, 축복
2021년:	문제, 커피, 이제, 궁금, 무야호, 감성, 스불재, 고양이, 동료, 부캐
2022년:	문제, 감성, 동료, 고양이, 궁금, 중꺾마, 만남, 잡채, 낭만, 신뢰
2023년:	엄마, 문제, 이제, 고양이, 감성, 신뢰, 궁금, 동료, 만남, 축복
2024년:	문제, 엄마, 이제, 고양이, 감성, 만남, 동료, 축복, 신뢰, 궁금



Topic 0은 2024년, Topic 1은 2023년, Topic 2는 2020년, Topic 3은 2000년대,

Topic 4는 2022년, Topic 5는 2021년이라고 예측했다.

하지만 각 연도별로 top10 유행어가 비슷하고 Topic의 단어들을 봤을 때 확실하게  
나뉘어졌다고 할 수 없다고 판단했다.

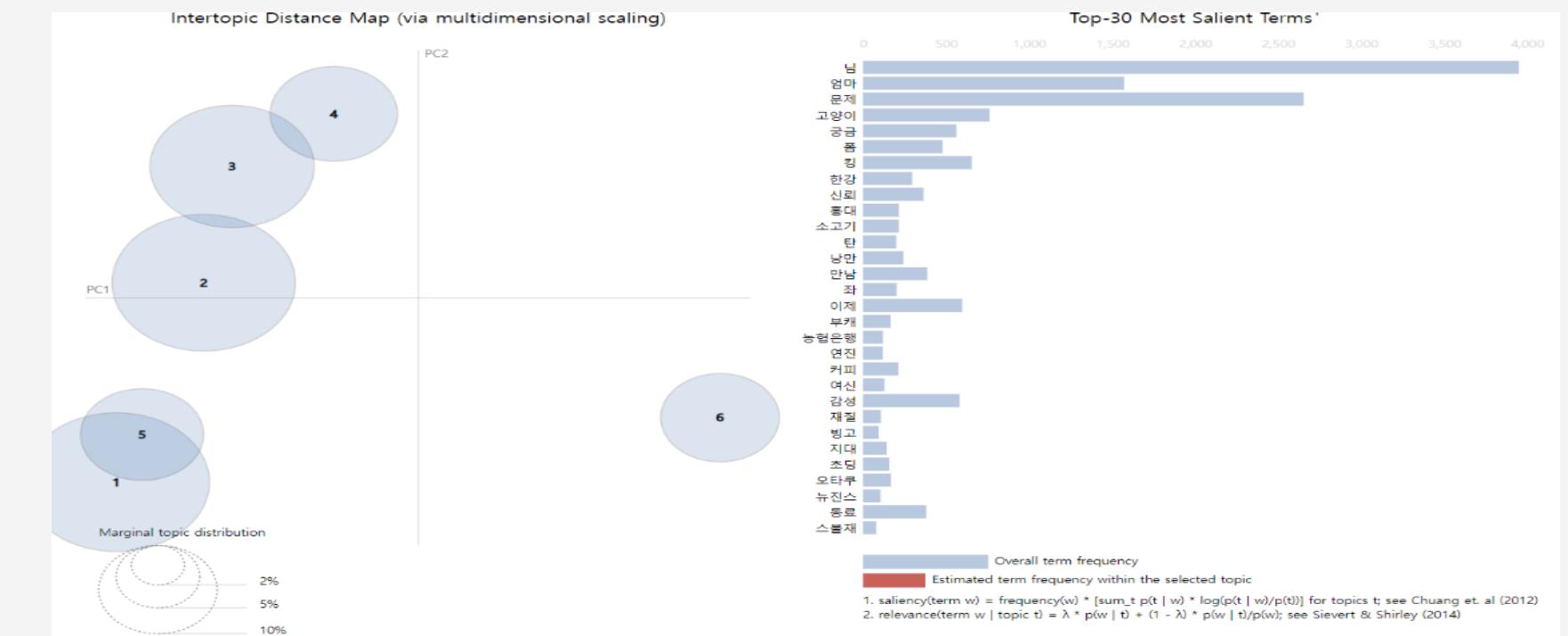
# LDA

## 2000년대, 2020-2024년 유행어

```
N_TOPICS = 6  
ldamodel = gensim.models.ldamodel.LdaModel(corpus,  
                                              num_topics = N_TOPICS, ## 주제 몇개 할 건지  
                                              id2word=word_dict) ## 보카불러리 셋
```

```
1 topics = ldamodel.print_topics(num_words=5)  
2 ## 대표적인 단어를 알 수 있음  
3 for topic in topics:  
4     print(topic)  
  
(0, '0.520*"문제" + 0.079*"고양이" + 0.069*"신뢰" + 0.058*"폼" + 0.029*"동료"')  
(1, '0.279*"궁금" + 0.080*"뉴진스" + 0.072*"낭만" + 0.069*"디토" + 0.069*"부캐")  
(2, '0.194*"만남" + 0.112*"좌" + 0.084*"먹방" + 0.083*"동료" + 0.037*"원영")  
(3, '0.694*"님" + 0.048*"축복" + 0.030*"짱" + 0.027*"동료" + 0.022*"하이")  
(4, '0.412*"엄마" + 0.094*"이제" + 0.039*"문제" + 0.033*"잡채" + 0.032*"님")  
(5, '0.242*"감성" + 0.225*"킹" + 0.112*"한강" + 0.067*"홍대" + 0.039*"커피")
```

유행어는 한 글자인 유행어도 있기 때문에 삭제하지 않고 Topic 수도 6개로 설정



### 실제 각 연도의 유행어 top 10과 비교

- 2000년대: 문제, 감성, 엄마, 이제, 오타쿠, 만남, 신뢰, 낭만, 궁금, 동료  
2020년: 문제, 감성, 궁금, 동료, 한강, 만남, 고양이, 뉴진스, 아이스, 축복  
2021년: 문제, 커피, 이제, 궁금, 무야호, 감성, 스쿨체, 고양이, 동료, 부캐  
2022년: 문제, 감성, 동료, 고양이, 궁금, 중꺾마, 만남, 잡채, 낭만, 신뢰  
2023년: 엄마, 문제, 이제, 고양이, 감성, 신뢰, 궁금, 동료, 만남, 축복  
2024년: 문제, 엄마, 이제, 고양이, 감성, 만남, 동료, 축복, 신뢰, 궁금



Topic 0은 2023년, Topic 1은 2024년, Topic 2는 2021년 or 2024년, Topic 3은 2020년,

Topic 4는 2023년 or 2022년, Topic 5는 2000년대라고 예측했다.

단순히 단어의 빈도와 분포를 기반으로 하는 LSA와 달리

LDA는 단순히 빈도수가 많은 유행어보다 각 연도별 특징적으로 나타나는 단어들을

Topic에 골고루 분류한 것을 알 수 있었다.

```
[‘가능’ ‘가정’ ‘가족’ ‘가지’ ‘감사’ ‘감상’ ‘감정’ ‘강사’ ‘개인’ ‘거짓말’ ‘검사’ ‘게임’ ‘결혼’ ‘경찰’
‘경찰서’ ‘공감’ ‘공동’ ‘공조’ ‘과거’ ‘과정’ ‘관련’ ‘관심’ ‘국가’ ‘국내’ ‘궁금’ ‘권력’ ‘그룹’ ‘금수저’
‘금요일’ ‘기간’ ‘기대작’ ‘기록’ ‘기사’ ‘기억’ ‘기획’ ‘김태리’ ‘나라’ ‘나의’ ‘나이’ ‘날짜’ ‘남녀’ ‘남편’
‘내용’ ‘네이버’ ‘넷플릭스’ ‘년대’ ‘눈물’ ‘느낌’ ‘능력’ ‘다양’ ‘닥터’ ‘대사’ ‘대상’ ‘대신’ ‘대표’ ‘대학’
‘대한민국’ ‘도시’ ‘동명’ ‘동생’ ‘동시’ ‘드래곤’ ‘등급’ ‘등장’ ‘디즈니’ ‘때문’ ‘라이프’ ‘러브’ ‘로맨스’
‘로맨틱’ ‘리뷰’ ‘링크’ ‘마우스’ ‘마을’ ‘마음’ ‘마이’ ‘마지막’ ‘막내아들’ ‘만화’ ‘맞선’ ‘매력’ ‘매혹’
‘멜로’ ‘모범’ ‘모습’ ‘문제’ ‘미국’ ‘미래’ ‘미스터리’ ‘미경’ ‘바이두’ ‘방송사’ ‘배경’ ‘백과’ ‘범죄’ ‘법정’
‘변호사’ ‘변화’ ‘병원’ ‘보이’ ‘복귀’ ‘복수’ ‘부모’ ‘부부’ ‘부분’ ‘분위기’ ‘비밀’ ‘사건’ ‘사고’ ‘사극’
‘사내’ ‘사실’ ‘사이’ ‘사장’ ‘사회’ ‘살인’ ‘상황’ ‘생활’ ‘선수’ ‘선택’ ‘설정’ ‘성격’ ‘성공’ ‘성장’
‘세계’ ‘세상’ ‘세작’ ‘소녀’ ‘소년’ ‘소리’ ‘소설’ ‘소식’ ‘소재’ ‘수사’ ‘수사반장’ ‘수상’ ‘수요일’ ‘순간’
‘스릴러’ ‘스캔들’ ‘스타’ ‘스토리’ ‘스튜디오’ ‘스트리밍’ ‘슬럼프’ ‘시대’ ‘시리즈’ ‘시절’ ‘시청자’ ‘아내’
‘아들’ ‘아래’ ‘아버지’ ‘아이돌’ ‘아이치’ ‘악귀’ ‘액션’ ‘얼굴’ ‘엄마’ ‘에피소드’ ‘엔터테인먼트’ ‘여성’ ‘여왕’
‘역할’ ‘연기’ ‘연애’ ‘연인’ ‘열혈’ ‘영상’ ‘영화’ ‘예고’ ‘예고편’ ‘예능’ ‘예상’ ‘오늘’ ‘오랜만’ ‘오리지널’
‘오피스’ ‘오후’ ‘올해’ ‘완벽’ ‘외모’ ‘요즘’ ‘욕망’ ‘우영우’ ‘우정’ ‘우주’ ‘운명’ ‘월요일’ ‘월화드라마’
‘웨이보’ ‘웨이보’ ‘웹툰’ ‘위기’ ‘유명’ ‘의사’ ‘이름’ ‘이미지’ ‘이상한’ ‘이유’ ‘이제훈’ ‘이해’ ‘이혼’
‘이후’ ‘인간’ ‘인기’ ‘인생’ ‘인하’ ‘일본’ ‘일상’ ‘자리’ ‘자체’ ‘장면’ ‘재미’ ‘재벌’ ‘저작’ ‘전개’
‘전문’ ‘전쟁’ ‘정도’ ‘정책’ ‘제공’ ‘제작사’ ‘조선’ ‘존재’ ‘졸업’ ‘종영’ ‘주말’ ‘주연’ ‘주요’ ‘죽음’
‘준비’ ‘중국’ ‘지니’ ‘지옥’ ‘진실’ ‘진행’ ‘집안’ ‘집풀’ ‘채널’ ‘처음’ ‘천재’ ‘청춘’ ‘촬영’ ‘최고’
‘추적’ ‘출신’ ‘친구’ ‘캐릭터’ ‘캐스팅’ ‘케미’ ‘코미디’ ‘코믹’ ‘콘텐츠’ ‘쿠팡’ ‘키스’ ‘탈출’ ‘택시’
‘텐센트’ ‘토요일’ ‘통쾌’ ‘특별’ ‘티빙’ ‘티저’ ‘팀장’ ‘파트’ ‘판타지’ ‘麝香’ ‘포스터’ ‘포스팅’ ‘포인트’
‘플랫폼’ ‘플러스’ ‘플레이’ ‘필요’ ‘하루’ ‘학교’ ‘학생’ ‘한국’ ‘해결’ ‘해결사’ ‘해당’ ‘행복’ ‘현실’ ‘현재’
‘형사’ ‘홈페이지’ ‘화재’ ‘확인’ ‘환호’ ‘활동’ ‘회사’ ‘회차’ ‘횟수’ ‘후속’ ‘후속작’ ‘휴면’ ‘흥미’ ‘히어로’
‘힐링’]
```

## 결과 해석

토픽별 해당 토픽을 대표하는 주요 키워드와 그 중요도(가중치)

Topic 1: 텐센트, 웨이보, 바이두, 중국, 백과

Topic 5: 변호사, 우영우, 사건, 검사, 넷플릭스

Topic 6: 수사반장, 형사, 디즈니, 플러스, 범죄

Topic 12: 디즈니, 플러스, 남편, 가족, 결혼

Topic 13: 디즈니, 여왕, 플러스, 눈물, 로맨스

중국 소설 미디어와 인기 드라마였던

<0|상한 변호사 우영우>, <수사반장>, <내 남편과 결혼해줘>, <눈물의 여왕>

등과 관련된 키워드들을 토픽별로 찾아볼 수 있다.

```
model = TruncatedSVD(n_components=30, n_iter=100, random_state=42)
model.fit(X)
```

벡터 차원을 300으로, 토픽 수를 30으로 설정

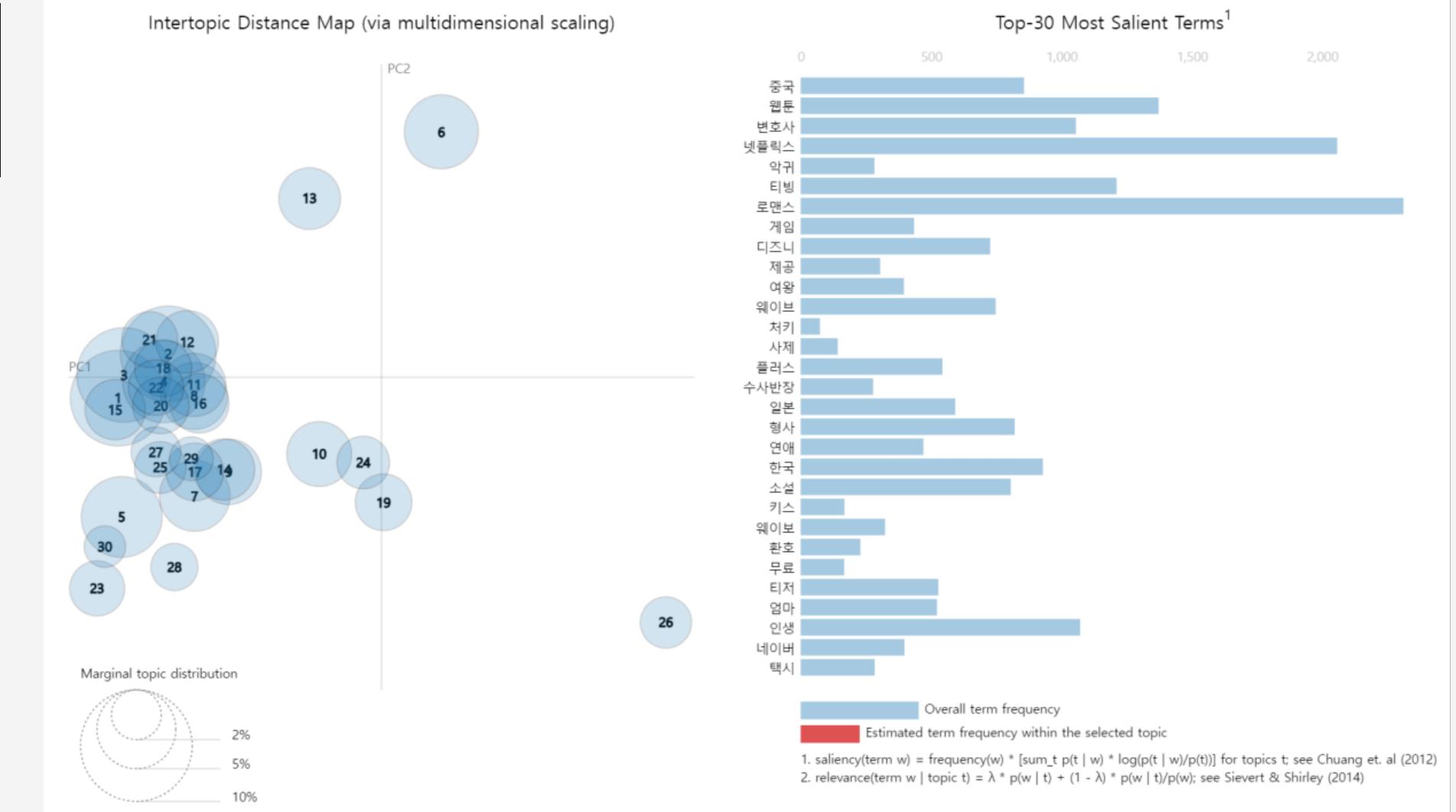
```
get_keyword_by_topic(model.components_, terms)
```

```
Topic 0: [('로맨스', 0.2339), ('넷플릭스', 0.1871), ('웹툰', 0.156), ('중국', 0.1492), ('웨이보', 0.1487)]
Topic 1: [('텐센트', 0.4871), ('웨이보', 0.4394), ('바이두', 0.2946), ('중국', 0.2694), ('백과', 0.2667)]
Topic 2: [('중국', 0.8221), ('아이치', 0.2162), ('링크', 0.1103), ('아래', 0.1041), ('리뷰', 0.0866)]
Topic 3: [('텐센트', 0.6), ('중국', 0.2204), ('넷플릭스', 0.2177), ('웹툰', 0.1476), ('티빙', 0.0866)]
Topic 4: [('넷플릭스', 0.507), ('웹툰', 0.2721), ('바이두', 0.2023), ('백과', 0.2), ('디즈니', 0.1659)]
Topic 5: [('변호사', 0.4665), ('우영우', 0.1743), ('사건', 0.1654), ('검사', 0.1606), ('넷플릭스', 0.1515)]
Topic 6: [('수사반장', 0.3371), ('형사', 0.3081), ('디즈니', 0.2201), ('플러스', 0.1757), ('범죄', 0.1615)]
Topic 7: [('일본', 0.61), ('게임', 0.1202), ('채널', 0.1104), ('결혼', 0.0941), ('때문', 0.0907)]
Topic 8: [('웹툰', 0.4163), ('형사', 0.3135), ('수사반장', 0.2222), ('일본', 0.1947), ('수사', 0.1856)]
Topic 9: [('티빙', 0.3741), ('디즈니', 0.3136), ('플러스', 0.2667), ('이미지', 0.1942), ('웨이보', 0.1712)]
Topic 10: [('로맨스', 0.3507), ('청춘', 0.2768), ('연애', 0.2407), ('소년', 0.1438), ('판타지', 0.0987)]
Topic 11: [('일본', 0.5152), ('채널', 0.2392), ('닥터', 0.2043), ('로맨스', 0.2041), ('디즈니', 0.1799)]
Topic 12: [('디즈니', 0.3285), ('플러스', 0.2591), ('남편', 0.2316), ('가족', 0.1945), ('결혼', 0.1769)]
Topic 13: [('디즈니', 0.2617), ('여왕', 0.2218), ('플러스', 0.2212), ('눈물', 0.1998), ('로맨스', 0.1818)]
Topic 14: [('닥터', 0.3964), ('슬럼프', 0.2793), ('세작', 0.1802), ('의사', 0.1799), ('해결사', 0.161)]
Topic 15: [('조선', 0.3846), ('사극', 0.3048), ('시대', 0.169), ('디즈니', 0.1503), ('악귀', 0.1378)]
Topic 16: [('게임', 0.3736), ('제작사', 0.3274), ('스튜디오', 0.2937), ('도시', 0.2183), ('닥터', 0.2136)]
Topic 17: [('도시', 0.4951), ('수사반장', 0.3322), ('이제훈', 0.1259), ('닥터', 0.1216), ('형사', 0.1001)]
Topic 18: [('게임', 0.4693), ('사극', 0.3229), ('티빙', 0.2065), ('변호사', 0.2052), ('연애', 0.2019)]
Topic 19: [('도시', 0.3736), ('아이치', 0.3721), ('웨이보', 0.2466), ('엄마', 0.123), ('감정', 0.1215)]
Topic 20: [('세작', 0.3311), ('매혹', 0.2615), ('악귀', 0.2048), ('김태리', 0.1481), ('넷플릭스', 0.1064)]
Topic 21: [('악귀', 0.3154), ('티빙', 0.2819), ('택시', 0.2271), ('연애', 0.1815), ('게임', 0.1541)]
Topic 22: [('청춘', 0.3964), ('소년', 0.3123), ('티빙', 0.2709), ('여왕', 0.2303), ('성장', 0.2016)]
Topic 23: [('수사반장', 0.3553), ('연애', 0.2026), ('이제훈', 0.1694), ('결혼', 0.1602), ('제작사', 0.1458)]
Topic 24: [('연애', 0.2296), ('의사', 0.2268), ('조선', 0.2265), ('아이치', 0.1891), ('사건', 0.1376)]
Topic 25: [('게임', 0.3721), ('악귀', 0.2685), ('도시', 0.2252), ('검사', 0.2186), ('이미지', 0.193)]
Topic 26: [('엄마', 0.207), ('麝香', 0.1874), ('기록', 0.1727), ('코미디', 0.1715), ('주말', 0.1707)]
Topic 27: [('악귀', 0.3923), ('김태리', 0.2126), ('비밀', 0.2056), ('수사반장', 0.1539), ('엄마', 0.1503)]
Topic 28: [('제공', 0.2843), ('졸업', 0.2094), ('웨이보', 0.1974), ('검사', 0.157), ('강사', 0.1509)]
Topic 29: [('감정', 0.3263), ('제공', 0.2738), ('링크', 0.1623), ('악귀', 0.1538), ('히어로', 0.1531)]
```

```
N_TOPICS = 30
ldamodel = gensim.models.ldamodel.LdaModel(corpus,
                                              num_topics = N_TOPICS, ## 주제 몇 개 할 건지
                                              id2word=word_dict) ## vocabulary set

topics = ldamodel.print_topics(num_words=5) ## 각 토픽들에 대해서 몇 개의 단어 (num_words)로 표현할 것인지
## 대표적인 단어를 알 수 있음
for topic in topics:
    print(topic)

(28, '0.005*"로맨스" + 0.004*"넷플릭스" + 0.003*"변호사" + 0.003*"비밀"')
(8, '0.012**"중국" + 0.006**"여왕" + 0.005**"로맨스" + 0.004**"눈물" + 0.004**"티빙"')
(6, '0.008**"로맨스" + 0.006**"넷플릭스" + 0.005**"소설" + 0.004**"웹툰" + 0.003**"코미디"')
(27, '0.005**"로맨스" + 0.005**"티빙" + 0.004**"넷플릭스" + 0.004**"웹툰" + 0.003**"내용"')
(29, '0.006**"중국" + 0.005**"로맨스" + 0.004**"넷플릭스" + 0.003**"판타지"')
(25, '0.009**"변호사" + 0.004**"티빙" + 0.004**"로맨스" + 0.003**"웹툰" + 0.003**"가족"')
(15, '0.005**"로맨스" + 0.005**"한국" + 0.004**"형사" + 0.003**"인생" + 0.003**"복수"')
(0, '0.006**"웹툰" + 0.005**"로맨스" + 0.004**"형사" + 0.004**"한국" + 0.003**"인생"')
(19, '0.008**"넷플릭스" + 0.006**"로맨스" + 0.005**"디즈니" + 0.004**"티빙" + 0.003**"플러스"')
(7, '0.011**"웹툰" + 0.005**"로맨스" + 0.005**"넷플릭스" + 0.003**"판타지" + 0.003**"티빙"')
(20, '0.013**"넷플릭스" + 0.007**"웹툰" + 0.005**"로맨스" + 0.003**"한국" + 0.003**"디즈니"')
(22, '0.007**"로맨스" + 0.006**"웹툰" + 0.005**"넷플릭스" + 0.004**"인생" + 0.003**"편성"')
(16, '0.005**"로맨스" + 0.005**"사제" + 0.005**"넷플릭스" + 0.004**"웹툰" + 0.004**"열혈"')
(1, '0.008**"넷플릭스" + 0.006**"일본" + 0.006**"게임" + 0.003**"한국" + 0.003**"전국"')
(11, '0.008**"로맨스" + 0.006**"웹툰" + 0.005**"디즈니" + 0.004**"넷플릭스" + 0.004**"인생"')
(17, '0.006**"악귀" + 0.004**"티빙" + 0.004**"디자인" + 0.003**"범죄" + 0.003**"넷플릭스"')
(18, '0.006**"로맨스" + 0.005**"인생" + 0.003**"의사" + 0.003**"결혼" + 0.003**"닥터"')
(12, '0.007**"넷플릭스" + 0.007**"로맨스" + 0.003**"스릴러" + 0.003**"한국" + 0.003**"남편"')
(2, '0.007**"로맨스" + 0.007**"넷플릭스" + 0.005**"티빙" + 0.004**"웹툰" + 0.003**"연애"')
(4, '0.006**"웹툰" + 0.004**"넷플릭스" + 0.004**"내용" + 0.003**"로맨스" + 0.003**"중국"')
```



## 결과 해석

LSA와 LDA 모두 '로맨스', '넷플릭스', '변호사', '중국', '디즈니', '형사', '가족', '판타지' 등의 키워드가 반복적으로 나타난다.

LSA와 LDA 두 모델링이 모두 비슷한 결과를 보였는데,  
이는 데이터의 특성의 영향을 받았다고 볼 수 있다.

하나의 문서에서 단일 주제(드라마 1개)를 집중적으로 다루고 있는 경우가 많기 때문이다.

[‘가능’ ‘가디언즈’ ‘가운데’ ‘가지’ ‘각본’ ‘감동’ ‘감상’ ‘감정’ ‘개인’ ‘갤럭시’ ‘게임’ ‘결과’ ‘결심’ ‘경우’  
 ‘경험’ ‘고양이’ ‘공간’ ‘공조’ ‘공포’ ‘과거’ ‘과정’ ‘관련’ ‘관심’ ‘괴물’ ‘국가’ ‘국제’ ‘귀멸의’ ‘규모’  
 ‘극장가’ ‘극장판’ ‘금주’ ‘기간’ ‘기억’ ‘기자’ ‘나라’ ‘나름’ ‘남자’ ‘남편’ ‘네이버’ ‘넷플릭스’ ‘년대’  
 ‘노래’ ‘노량’ ‘녹음’ ‘느낌’ ‘능력’ ‘다니엘’ ‘다양’ ‘다큐멘터리’ ‘닥터’ ‘달려’ ‘당시’ ‘대결’ ‘대사’ ‘대표’  
 ‘덩크’ ‘데드’ ‘데뷔’ ‘도시’ ‘독립’ ‘돌비’ ‘동시’ ‘동원’ ‘등급’ ‘디즈니’ ‘러닝’ ‘러브’ ‘로맨스’ ‘로스트’  
 ‘로튼’ ‘롯데시네마’ ‘마녀’ ‘마동석’ ‘마블’ ‘마음’ ‘마이클’ ‘마지막’ ‘매력’ ‘매버릭’ ‘매출’ ‘멀티’  
 ‘메가박스’ ‘모험’ ‘목소리’ ‘무대’ ‘무비’ ‘문단속’ ‘문제’ ‘뮤지컬’ ‘미국’ ‘미션’ ‘미스터리’ ‘바다’ ‘반응’  
 ‘배경’ ‘배급’ ‘배트맨’ ‘버스’ ‘범죄’ ‘범죄도시’ ‘부문’ ‘부분’ ‘북미’ ‘분기점’ ‘분위기’ ‘불가’ ‘브라더스’  
 ‘블랙’ ‘블록버스터’ ‘비교’ ‘비밀’ ‘사건’ ‘사실’ ‘사이’ ‘사진’ ‘사회’ ‘살인’ ‘상영관’ ‘상태’ ‘살라메’  
 ‘서비스’ ‘서울’ ‘선사’ ‘선언’ ‘선택’ ‘성공’ ‘세상’ ‘소녀’ ‘소년’ ‘소니’ ‘소설’ ‘소식’ ‘소재’ ‘속면’  
 ‘순익’ ‘수요일’ ‘수입’ ‘수준’ ‘순간’ ‘슈퍼’ ‘스릴러’ ‘스즈메의’ ‘스크린’ ‘스타’ ‘스토리’ ‘스튜디오’  
 ‘스트레인지’ ‘스파이’ ‘스파이더맨’ ‘시네마’ ‘시대’ ‘시민’ ‘시장’ ‘시절’ ‘시즌’ ‘시청’ ‘실패’ ‘실화’ ‘아들’  
 ‘아래’ ‘아바타’ ‘아이맥스’ ‘아카데미’ ‘애니메이션’ ‘얼굴’ ‘엄마’ ‘여름’ ‘여성’ ‘여자’ ‘역사’ ‘역할’ ‘연휴’  
 ‘영상’ ‘영화관’ ‘예상’ ‘예술’ ‘오리지널’ ‘오브’ ‘오펜하이머’ ‘오프닝’ ‘와이드’ ‘외계인’ ‘외국’ ‘운명’ ‘워너’  
 ‘월드’ ‘윙카’ ‘웨이’ ‘위기’ ‘유니버설’ ‘유명’ ‘유지’ ‘유해진’ ‘음악’ ‘의미’ ‘이유’ ‘이전’  
 ‘인간’ ‘인기’ ‘인도’ ‘인물’ ‘인상’ ‘인생’ ‘인하’ ‘일본’ ‘입장’ ‘자리’ ‘자체’ ‘작가’ ‘작년’ ‘작전’  
 ‘장면’ ‘재미’ ‘전국’ ‘전작’ ‘전쟁’ ‘전주’ ‘전체’ ‘점수’ ‘점유’ ‘제공’ ‘제임스’ ‘제작’ ‘존재’ ‘좌석’  
 ‘주간’ ‘주목’ ‘주요’ ‘죽음’ ‘준비’ ‘중국’ ‘중심’ ‘쥬라기’ ‘증강’ ‘지난주’ ‘진행’ ‘짱구’ ‘차지’ ‘참여’  
 ‘처음’ ‘촬영’ ‘최종’ ‘최초’ ‘추석’ ‘친구’ ‘칼날’ ‘캐릭터’ ‘캐스팅’ ‘코로나’ ‘코미디’ ‘콘텐츠’ ‘쿵푸’  
 ‘크루즈’ ‘크리스’ ‘타임’ ‘탄생’ ‘토론’ ‘토마토’ ‘통합’ ‘특별’ ‘티모시’ ‘티켓’ ‘파묘’ ‘판매’ ‘판타지’  
 ‘팬더’ ‘팬서’ ‘퍼스트’ ‘평가’ ‘포스터’ ‘포스팅’ ‘포에버’ ‘포인트’ ‘포함’ ‘표현’ ‘프랑스’ ‘플러스’ ‘필름’  
 ‘필요’ ‘하락’ ‘하루’ ‘학교’ ‘할리우드’ ‘해당’ ‘해외’ ‘해적’ ‘행복’ ‘힌트’ ‘헤어질’ ‘현실’ ‘형사’ ‘호평’  
 ‘흔돈’ ‘화제’ ‘확인’ ‘후보’ ‘흥미’ ‘히어로’]

300

## 결과 해석

**Topic 3: 북미, 파묘, 해외, 윙카, 넷플릭스**

**Topic 8: 문단속, 스즈메의, 애니메이션, 디즈니, 덩크**

**Topic 15: 아바타, 헤어질, 파묘, 매버릭, 결심**

**Topic 24: 배트맨, 스즈메의, 문단속, 해적, 멀티**

**Topic 29: 로맨스, 과물, 오펜하이머, 스파이더맨, 감정**

각 토픽의 키워드들이 관련성이 떨어지는 결과를 볼 수 있다.

이는 LSA 모델이 단어 간의 관계를 충분히 반영하지 못하기 때문이라고 유추할 수 있다. LSA는 단순히 단어의 빈도와 분포를 기반으로 하기 때문에, 주제의 명확성이 비교적 떨어지는 것이다.

```
model = TruncatedSVD(n_components=30, n_iter=100, random_state=42)
model.fit(X)
```

벡터 차원을 300으로, 토픽 수를 30으로 설정

```
get_keyword_by_topic(model.components_, terms)
```

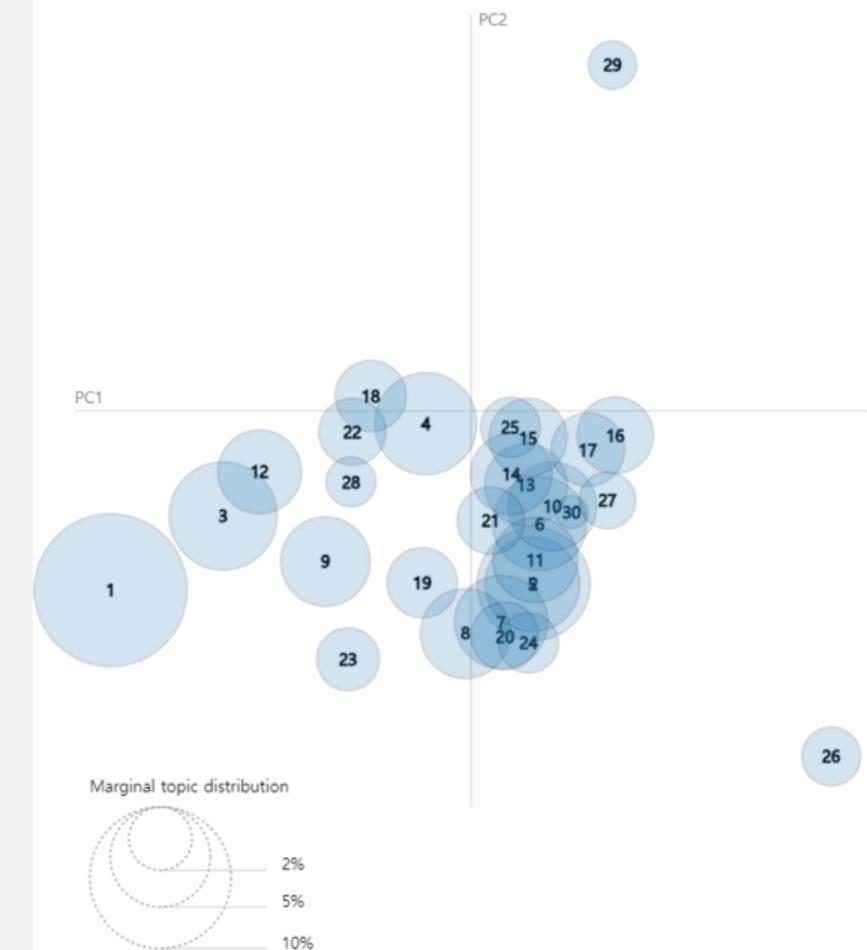
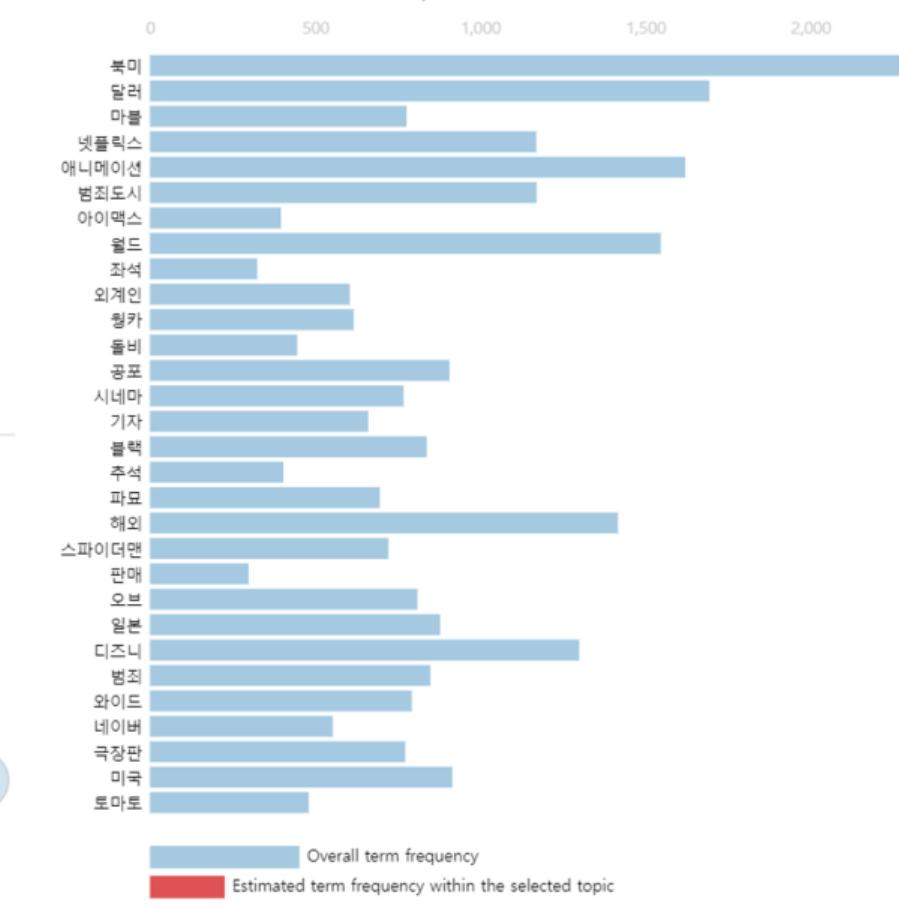
```
Topic 0: [('넷플릭스', 0.1593), ('애니메이션', 0.156), ('범죄도시', 0.1393), ('달려', 0.137), ('공포', 0.1198)]
Topic 1: [('달려', 0.536), ('북미', 0.3642), ('월드', 0.222), ('해외', 0.2027), ('와이드', 0.192)]
Topic 2: [('범죄도시', 0.701), ('마동석', 0.2903), ('범죄', 0.2455), ('파묘', 0.1696), ('도시', 0.1624)]
Topic 3: [('북미', 0.3921), ('파묘', 0.2382), ('해외', 0.2193), ('윙카', 0.1923), ('넷플릭스', 0.1876)]
Topic 4: [('넷플릭스', 0.7354), ('범죄도시', 0.2497), ('달려', 0.2048), ('마동석', 0.1336), ('범죄', 0.1193)]
Topic 5: [('기자', 0.3931), ('파묘', 0.3789), ('독립', 0.231), ('녹음', 0.2049)]
Topic 6: [('프랑스', 0.4978), ('경우', 0.3993), ('범죄도시', 0.0868), ('북미', 0.0726)]
Topic 7: [('프랑스', 0.5367), ('경우', 0.4571), ('마블', 0.2491), ('넷플릭스', 0.2324), ('파묘', 0.2061)]
Topic 8: [('문단속', 0.2842), ('스즈메의', 0.2838), ('애니메이션', 0.2817), ('디즈니', 0.2775), ('덩크', 0.1665)]
Topic 9: [('극장판', 0.3963), ('추석', 0.2021), ('애니메이션', 0.2018), ('넷플릭스', 0.1841), ('극장가', 0.1459)]
Topic 10: [('외계인', 0.4033), ('윙카', 0.3654), ('디즈니', 0.1705), ('시민', 0.1462), ('넷플릭스', 0.1268)]
Topic 11: [('디즈니', 0.4992), ('애니메이션', 0.335), ('파묘', 0.2363), ('달려', 0.1899), ('친구', 0.1077)]
Topic 12: [('추석', 0.5762), ('연휴', 0.2999), ('디즈니', 0.2076), ('달려', 0.2042), ('네이버', 0.1522)]
Topic 13: [('추석', 0.3278), ('기자', 0.2621), ('디즈니', 0.262), ('공포', 0.2572), ('연휴', 0.1522)]
Topic 14: [('서울', 0.2969), ('노량', 0.2874), ('바다', 0.2761), ('파묘', 0.1941), ('디즈니', 0.1918)]
Topic 15: [('아바타', 0.2353), ('헤어질', 0.2108), ('파묘', 0.2089), ('매버릭', 0.2062), ('결심', 0.1971)]
Topic 16: [('공포', 0.5254), ('외계인', 0.522), ('문단속', 0.1723), ('스즈메의', 0.1676), ('선언', 0.1279)]
Topic 17: [('돌비', 0.4732), ('아이맥스', 0.3369), ('시네마', 0.3108), ('공포', 0.1522), ('해적', 0.1481)]
Topic 18: [('네이버', 0.3829), ('엄마', 0.3799), ('돌비', 0.2137), ('포스터', 0.2086), ('시네마', 0.167)]
Topic 19: [('네이버', 0.5572), ('사건', 0.1286), ('타임', 0.1262), ('돌비', 0.1237), ('등급', 0.1206)]
Topic 20: [('아바타', 0.2537), ('좌석', 0.2214), ('디즈니', 0.2137), ('블랙', 0.1596), ('네이버', 0.1586)]
Topic 21: [('포스터', 0.2611), ('블랙', 0.2341), ('팬서', 0.2024), ('팬서', 0.1961), ('여자', 0.1491)]
Topic 22: [('쿵푸', 0.3086), ('팬더', 0.2769), ('좌석', 0.2732), ('외계인', 0.1554), ('판매', 0.1497)]
Topic 23: [('아바타', 0.4472), ('추석', 0.182), ('로맨스', 0.1531), ('인간', 0.1529), ('북미', 0.1488)]
Topic 24: [('배트맨', 0.4981), ('스즈메의', 0.1868), ('문단속', 0.1861), ('해적', 0.1491), ('멀티', 0.1028)]
Topic 25: [('배트맨', 0.4363), ('아바타', 0.3538), ('외계인', 0.1757), ('포스터', 0.1434), ('토마토', 0.1432)]
Topic 26: [('토마토', 0.2867), ('와이드', 0.1888), ('로튼', 0.171), ('점수', 0.1542), ('상영관', 0.1535)]
Topic 27: [('배트맨', 0.1995), ('해적', 0.1929), ('엄마', 0.1911), ('극장판', 0.1712), ('블랙', 0.1615)]
Topic 28: [('좌석', 0.2844), ('판매', 0.1744), ('결심', 0.1647), ('헤어질', 0.1595), ('배트맨', 0.1489)]
Topic 29: [('로맨스', 0.194), ('과물', 0.1809), ('오펜하이머', 0.1683), ('스파이더맨', 0.1668), ('감정', 0.1433)]
```

```
N_TOPICS = 30
ldamodel = gensim.models.ldamodel.LdaModel(corpus,
                                              num_topics = N_TOPICS, ## 주제 몇 개 할 건지
                                              id2word=word_dict) ## vocabulary set
```

```
topics = ldamodel.print_topics(num_words=5) ## 각 토픽들에 대해서 몇 개의 단어(num_words)로 표현할 것인지
## 대표적인 단어를 알 수 있음
for topic in topics:
    print(topic)

(6, '0.006*"북미" + 0.005*"디즈니" + 0.004**"애니메이션" + 0.004**"해외" + 0.004**"파묘"')
(16, '0.007**"애니메이션" + 0.004**"북미" + 0.004**"해외" + 0.003**"디즈니" + 0.003**"게임"')
(26, '0.005**"넷플릭스" + 0.003**"왕카" + 0.003**"애니메이션" + 0.003**"등급" + 0.002**"버스"')
(2, '0.005**"월드" + 0.005**"해외" + 0.004**"북미" + 0.004**"애니메이션" + 0.003**"디즈니"')
(14, '0.006**"애니메이션" + 0.005**"문단속" + 0.005**"스즈메의" + 0.005**"월드" + 0.004**"해외"')
(24, '0.005**"디즈니" + 0.005**"마블" + 0.003**"배트맨" + 0.003**"넷플릭스" + 0.003**"애니메이션"')
(23, '0.007**"디즈니" + 0.005**"넷플릭스" + 0.005**"아바타" + 0.003**"달러" + 0.003**"월드"')
(1, '0.010**"달러" + 0.007**"해외" + 0.005**"월드" + 0.005**"디즈니" + 0.005**"북미"')
(18, '0.004**"달러" + 0.003**"버스" + 0.003**"멀티" + 0.003**"스파이더맨" + 0.003**"북미"')
(4, '0.004**"디즈니" + 0.004**"판매" + 0.003**"작석" + 0.003**"애니메이션" + 0.003**"코미디"')
(20, '0.006**"왕카" + 0.005**"애니메이션" + 0.004**"해외" + 0.004**"넷플릭스" + 0.003**"블랙"')
(27, '0.005**"넷플릭스" + 0.005**"애니메이션" + 0.005**"디즈니" + 0.004**"달러" + 0.004**"북미"')
(28, '0.003**"월드" + 0.003**"아바타" + 0.003**"극장판" + 0.003**"미국" + 0.002**"북미"')
(8, '0.007**"범죄도시" + 0.004**"마블" + 0.004**"평가" + 0.004**"범죄" + 0.003**"월드"')
(13, '0.006**"북미" + 0.005**"파묘" + 0.004**"범죄도시" + 0.004**"애니메이션" + 0.003**"공포"')
(9, '0.006**"달러" + 0.005**"범죄도시" + 0.003**"넷플릭스" + 0.003**"블랙" + 0.003**"마동석"')
(10, '0.025**"북미" + 0.011**"달러" + 0.008**"월드" + 0.004**"와이드" + 0.004**"해외"')
(11, '0.003**"디즈니" + 0.002**"게임" + 0.002**"월드" + 0.002**"넷플릭스" + 0.002**"애니메이션"')
(25, '0.007**"넷플릭스" + 0.003**"디즈니" + 0.003**"월드" + 0.003**"시즌" + 0.002**"스릴러"')
(5, '0.008**"북미" + 0.005**"월드" + 0.004**"애니메이션" + 0.004**"해외" + 0.003**"토마토"')
```

Intertopic Distance Map (via multidimensional scaling)

Top-30 Most Salient Terms<sup>1</sup>

1. saliency(term w) = frequency(w) \* [sum\_t p(t | w) \* log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)

2. relevance(term w | topic t) =  $\lambda * p(w | t) + (1 - \lambda) * p(w | t)/p(w)$ ; see Sievert & Shirley (2014)

## 결과 해석

일부 토픽에서 주제가 명확했으나, 많은 토픽에서 단어 간의 연관성이 떨어졌다.

이는 데이터 전처리와 모델 파라미터 최적화 과정이 부족했다고 추측했다.

특히 **추가적인 불용어 처리**가 필요할 것으로 보인다. 또한 단순한 단어 빈도수 대신 **TF-IDF**를 사용하여 단어의 중요도를 반영할 수 있다.

→ 추가 분석 진행



# 2022-2024년 영화

TF-IDF 벡터화를 수행하고, 각 문서의 상위 10개의 TF-IDF 값을 출력

```
# TF-IDF 벡터화
tfidf_vectorizer = TfidfVectorizer(max_df=0.95, min_df=1)
tfidf_matrix = tfidf_vectorizer.fit_transform(df['preprocessed_body'])

# TF-IDF 결과를 데이터프레임으로 변환
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf_vectorizer.get_feature_names_out())

# 결과 출력 함수
def print_tfidf_results(tfidf_df, document_ids, top_n=10):
    for idx in document_ids:
        print(f"Document {idx}:")
        sorted_items = sorted(tfidf_df.loc[idx].items(), key=lambda x: x[1], reverse=True)[:top_n]
        for item in sorted_items:
            if item[1] > 0:
                print(f"\t{item[0]}: {item[1]:.4f}")
        print("-" * 20)

# 문서 ID 리스트
document_ids = list(range(10)) # 필요한 만큼의 문서 인덱스 지정

# TF-IDF 결과 출력 (상위 10개의 TF-IDF 값만 출력)
print_tfidf_results(tfidf_df, document_ids, top_n=10)
```

## 결과 해석

TF-IDF가 문서 내 단어의 중요도를 잘 반영하여, LSA와 LDA보다 주제 분류에서 더 명확한 결과를 보이고 있다.

Document 0:	공간: 0.4656	Document 1:	계정: 0.4131	Document 3:	돌비: 0.6806	Document 2:	무비: 0.3893
	디즈니: 0.3379		넷플릭스: 0.3912		시네마: 0.4496		무파사: 0.1581
	팝업: 0.2042		공유: 0.2894		포맷: 0.3236		쓰우: 0.1564
	신데렐라: 0.1940		인종: 0.2849		아이맥스: 0.2083		스파이더맨: 0.1415
	라푼젤: 0.1795		프리미엄: 0.2497		코돌비: 0.1561		소니: 0.1266
	레거시: 0.1590		광고: 0.2064		아가: 0.1547		엠파이어: 0.1233
	백설 공주: 0.1570		구성원: 0.1869		기획전: 0.1532		코엔: 0.1233
	공주: 0.1560		방법: 0.1674		화면: 0.1120		데드풀: 0.1206
	주년: 0.1477		스탠다드: 0.1625		용아맥: 0.1082		프로즌: 0.1159
	전시: 0.1416		번호: 0.1608		비율: 0.0970		에일리언: 0.1140

Document 4:	시사회: 0.2162	Document 5:	사진관: 0.3875	Document 6:	소탕: 0.2518	Document 7:	조커: 0.2874
	강동원: 0.1792		쿠첸: 0.2215		이동휘: 0.2292		데드풀: 0.2171
	크래신 스키: 0.1667		이선균: 0.1779		김무열: 0.2271		덕희: 0.1732
	포켓몬스터: 0.1395		현수: 0.1651		범죄도시: 0.2151		감정: 0.1612
	퓨리오사: 0.1382		전투: 0.1497		마동석: 0.1900		풀리: 0.1294
	매드: 0.1305		초원: 0.1472		지환: 0.1852		미키: 0.1268
	혹성탈출: 0.1258		단속: 0.1352		필리핀: 0.1791		시민: 0.1261
	동네: 0.1241		허구: 0.1334		수사대: 0.1707		퓨리오사: 0.1179
	맥스: 0.1210		차례: 0.1250		도박: 0.1672		할리: 0.1168
	시사: 0.1172		순서: 0.1191		마약: 0.1627		지난해: 0.1119

Document 8:	짱구: 0.3752	Document 9:	범죄도시: 0.5699
	노량: 0.3605		나흘: 0.3070
	조커: 0.2004		마동석: 0.2849
	바다: 0.2002		경경사: 0.2530
	아쿠아맨: 0.1898		도시: 0.2168
	죽음: 0.1763		속도: 0.2012
	리메이크: 0.1716		범죄: 0.1953
	김밥: 0.1562		박지화: 0.1879
	수제: 0.1562		에이: 0.1525
	호아킨: 0.1393		뉴스: 0.1369

['가능', '가요', '가운데', '가족', '가지', '감독', '감동', '감사', '감정', '개봉', '개월', '개인', '걸그룹', '게스트', '게임', '결과', '결정', '결혼', '결혼식', '경기', '경우', '경험', '고민', '공감', '공식', '공연', '과거', '과정', '관객', '관계', '관련', '광고', '구성', '국가', '국내', '국민', '그룹', '글로벌', '기대', '기록', '기사', '기안', '기억', '기자', '기준', '기회', '기획', '김호중', '나라', '남매', '남편', '내용', '네이버', '넷플릭스', '노래', '노력', '논란', '눈물', '뉴스', '느낌', '능력', '대상', '대중', '대학', '대한민국', '댄스', '데이트', '덱스', '도전', '동시', '동영상', '디즈니', '라디오', '라면', '라이브', '런닝맨', '로맨스', '리얼리티', '마지막', '맛집', '매력', '메인', '모델', '무대', '문제', '문화', '미국', '반응', '발매', '발표', '방식', '방영', '변화', '보컬', '본인', '봄날', '부모', '부문', '부부', '부분', '분석', '분야', '분위기', '브랜드', '사건', '사고', '사실', '사용', '사이', '사회', '상대', '상황', '생활', '서바이벌', '서비스', '서울', '선사', '선수', '선택', '성공', '성장', '세계', '세대', '세상', '소개', '소속', '소속사', '소재', '솔로', '수상', '순간', '순위', '스타', '스토리', '스포츠', '시대', '시리즈', '시상식', '시장', '시절', '시점', '시청자', '실력', '아나운서', '아내', '아들', '아버지', '아빠', '아이돌', '아티스트', '앨범', '야구', '어머니', '언급', '언니', '얼굴', '엄마', '에피소드', '엔터테인먼트', '여성', '여자', '여행', '역대', '역할', '연애', '연예', '연예인', '연인', '연출', '열애', '예고', '예고편', '예상', '예술', '오늘', '오디션', '오후', '온라인', '올해', '외모', '요리', '요즘', '우석', '우승', '운영', '웃음', '원작', '웨이브', '웹툰', '유명', '유재석', '유튜브', '음식', '음악', '옹원', '의미', '이름', '이미지', '이혼', '이효리', '인간', '인물', '인생', '인스타그램', '인정', '인터뷰', '인하', '일본', '일상', '입장', '자리', '자연', '자체', '작가', '작년', '장르', '장면', '재미', '전국', '전체', '전하', '전현무', '제공', '제작진', '제품', '조사', '조선', '존재', '졸업', '주목', '주연', '주인공', '준비', '중국', '중심', '중요', '지구', '지난해', '지역', '지옥', '지원', '직업', '집중', '참여', '처음', '촬영', '최강', '최고', '최근', '최종', '최초', '추리', '축구', '출생', '출신', '출연자', '친구', '캐릭터', '커플', '코리아', '콘서트', '퀴즈', '탄생', '투어', '투표', '트렌드', '트로트', '특별', '티빙', '티저', '패션', '편성', '평가', '포인트', '포함', '표현', '프로', '프로젝트', '플랫폼', '플러스', '플레이', '필요', '학교', '학력', '합류', '해당', '해외', '행복', '현실', '현재', '홈페이지', '홍보', '확인', '확정', '환승', '활약', '활용', '회사', '후보', '흥행']  
300

## 결과 해석

토픽별 해당 토픽을 대표하는 주요 키워드와 그 중요도(가중치)

Topic 1: 연애, 환승, 티빙 커플, 솔로

Topic 3: 넷플릭스, 지옥, 솔로, 결혼, 이혼

Topic 9: 야구, 최강, 선수, 이혼, 경기

OTT 플랫폼 티빙에서 방영했던 <환승연애>와

OTT 플랫폼 넷플릭스에서 방영했던 <솔로지옥>,

야구 선수들이 출연해 야구 경기를 치르는 <최강야구> 등의  
결과를 확인할 수 있다.

```
model = TruncatedSVD(n_components=30, n_iter=100, random_state=42)
model.fit(X)
```

벡터 차원을 300으로, 토픽 수를 30으로 설정

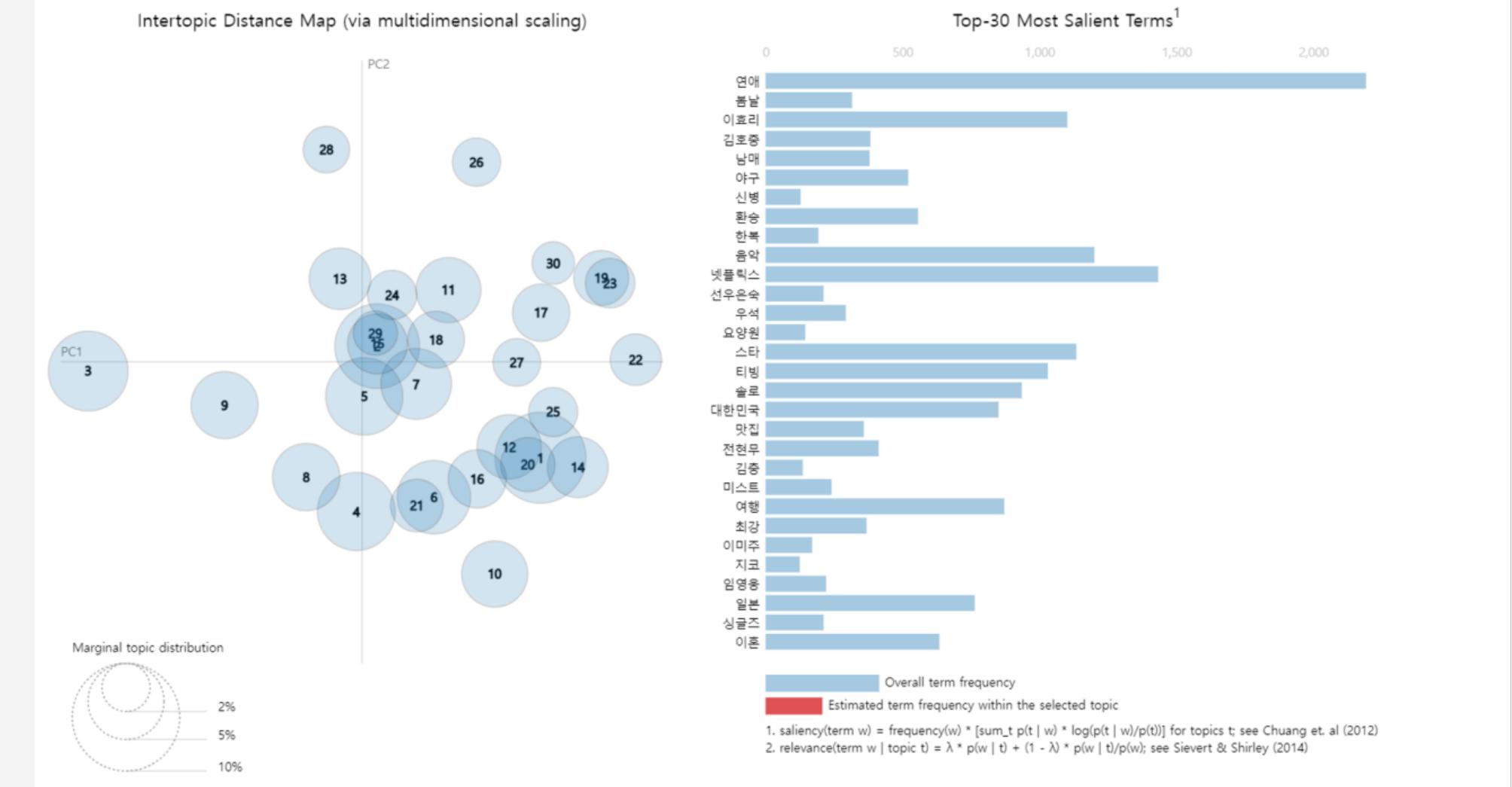
```
get_keyword_by_topic(model.components_, terms)
```

```
Topic 0: [(['연애', 0.1962), ('결혼', 0.1811), ('넷플릭스', 0.1657), ('게임', 0.1219), ('유튜브', 0.1198)]
Topic 1: [(['연애', 0.6993), ('환승', 0.3528), ('티빙', 0.2479), ('커플', 0.185), ('솔로', 0.1401)]
Topic 2: [(['결혼', 0.5808), ('이혼', 0.3934), ('남편', 0.1888), ('부부', 0.1696), ('결혼식', 0.1551)]
Topic 3: [(['넷플릭스', 0.5592), ('지옥', 0.3194), ('솔로', 0.2324), ('결혼', 0.223), ('이혼', 0.1812)]
Topic 4: [(['티빙', 0.4465), ('여행', 0.3079), ('이혼', 0.249), ('추리', 0.1031), ('지구', 0.0905)]
Topic 5: [(['음악', 0.3789), ('노래', 0.2541), ('티빙', 0.2221), ('이혼', 0.1946), ('무대', 0.1939)]
Topic 6: [(['선수', 0.2856), ('축구', 0.2625), ('야구', 0.2046), ('경기', 0.1766), ('일본', 0.1629)]
Topic 7: [(['티빙', 0.4066), ('선수', 0.2791), ('야구', 0.2474), ('축구', 0.2264), ('유재석', 0.1959)]
Topic 8: [(['여행', 0.6851), ('광고', 0.1144), ('솔로', 0.1083), ('음식', 0.0946), ('졸업', 0.0853)]
Topic 9: [(['야구', 0.3551), ('최강', 0.2282), ('선수', 0.2184), ('이혼', 0.2016), ('경기', 0.1468)]
Topic 10: [(['전현무', 0.2803), ('연예', 0.2781), ('대상', 0.2501), ('열애', 0.1994), ('여행', 0.1978)]
Topic 11: [(['티빙', 0.3417), ('중국', 0.2457), ('게임', 0.1934), ('광고', 0.1787), ('전현무', 0.1578)]
Topic 12: [(['게임', 0.277), ('유재석', 0.2605), ('열애', 0.2488), ('결혼', 0.2422), ('넷플릭스', 0.1431)]
Topic 13: [(['게임', 0.4248), ('김호중', 0.3604), ('중국', 0.2448), ('대한민국', 0.1246), ('트로트', 0.1143)]
Topic 14: [(['김호중', 0.4505), ('광고', 0.2981), ('넷플릭스', 0.2041), ('브랜드', 0.1366), ('축구', 0.1276)]
Topic 15: [(['중국', 0.3777), ('솔로', 0.2888), ('김호중', 0.2656), ('조사', 0.205), ('티빙', 0.1721)]
Topic 16: [(['우석', 0.4627), ('이혼', 0.2313), ('조사', 0.2276), ('넷플릭스', 0.215), ('뉴스', 0.1742)]
Topic 17: [(['게임', 0.4339), ('우석', 0.2336), ('출연자', 0.1154), ('조사', 0.1096), ('스포츠', 0.0938)]
Topic 18: [(['김호중', 0.4395), ('일본', 0.2389), ('열애', 0.2208), ('이혼', 0.1833), ('게임', 0.1162)]
Topic 19: [(['콘서트', 0.2299), ('공연', 0.1899), ('음악', 0.1788), ('문화', 0.1743), ('서울', 0.1397)]
Topic 20: [(['우석', 0.5619), ('음악', 0.1718), ('이효리', 0.1626), ('이혼', 0.1544), ('런닝맨', 0.1318)]
Topic 21: [(['이혼', 0.2884), ('축구', 0.2684), ('열애', 0.191), ('트로트', 0.1525), ('일본', 0.1421)]
Topic 22: [(['이효리', 0.6383), ('게임', 0.2414), ('열애', 0.1471), ('남매', 0.1394), ('조사', 0.1203)]
Topic 23: [(['디즈니', 0.2414), ('김호중', 0.1476), ('플러스', 0.1439), ('올해', 0.1399), ('연예', 0.1367)]
Topic 24: [(['콘서트', 0.3304), ('개봉', 0.1939), ('공연', 0.1927), ('일본', 0.1916), ('스타', 0.1864)]
Topic 25: [(['일본', 0.3403), ('디즈니', 0.2087), ('아빠', 0.1972), ('전현무', 0.179), ('아버지', 0.1597)]
Topic 26: [(['디즈니', 0.3331), ('열애', 0.2754), ('플러스', 0.1937), ('덱스', 0.1931), ('스타', 0.1928)]
Topic 27: [(['아나운서', 0.3995), ('스타', 0.222), ('환승', 0.2214), ('축구', 0.1903), ('퀴즈', 0.1617)]
Topic 28: [(['일본', 0.3134), ('음악', 0.2334), ('광고', 0.2101), ('노래', 0.1805), ('야구', 0.1448)]
Topic 29: [(['디즈니', 0.3349), ('일본', 0.2753), ('덱스', 0.2228), ('플러스', 0.1868), ('이효리', 0.1857)]
```

```
N_TOPICS = 30
ldamodel = gensim.models.ldamodel.LdaModel(corpus,
                                              num_topics = N_TOPICS, ## 주제 몇 개 할 건지
                                              id2word=word_dict) ## vocabulary set

topics = ldamodel.print_topics(num_words=5) ## 각 토픽들에 대해서 몇 개의 단어(num_words)로 표현할 것인지
## 대표적인 단어를 알 수 있음
for topic in topics:
    print(topic)

(13, '0.006*"결혼" + 0.004*"연애" + 0.003*"이효리" + 0.003*"음악" + 0.002*"유튜브"')
(11, '0.004*"솔로" + 0.004*"결혼" + 0.003*"지옥" + 0.003*"넷플릭스" + 0.003*"연애"')
(28, '0.007*"결혼" + 0.003*"이혼" + 0.003*"축구" + 0.003*"가족" + 0.003*"넷플릭스"')
(16, '0.003*"게임" + 0.003*"넷플릭스" + 0.003*"스타" + 0.003*"선수" + 0.003*"음악"')
(20, '0.006*"연애" + 0.005*"남매" + 0.004*"음악" + 0.004*"게임" + 0.003*"결혼"')
(29, '0.009*"연애" + 0.006*"결혼" + 0.003*"신병" + 0.003*"선수" + 0.002*"넷플릭스"')
(2, '0.007*"넷플릭스" + 0.005*"티빙" + 0.004*"서울" + 0.003*"결혼" + 0.003*"걸그룹"')
(26, '0.005*"넷플릭스" + 0.003*"결혼" + 0.003*"게임" + 0.003*"음악" + 0.002*"연애"')
(23, '0.004*"스타" + 0.003*"연예" + 0.003*"세계" + 0.003*"여행" + 0.002*"넷플릭스"')
(15, '0.004*"추리" + 0.004*"넷플릭스" + 0.003*"승범" + 0.003*"이미주" + 0.003*"정원"')
(5, '0.005*"신병" + 0.005*"티빙" + 0.004*"부대" + 0.004*"맛집" + 0.003*"야구"')
(4, '0.004*"여행" + 0.003*"중국" + 0.003*"스타" + 0.003*"대한민국" + 0.003*"음악"')
(17, '0.007*"연애" + 0.004*"직업" + 0.004*"환승" + 0.003*"커플" + 0.003*"여자"')
(22, '0.003*"대한민국" + 0.003*"여자" + 0.002*"스타" + 0.002*"중국" + 0.002*"아이돌"')
(3, '0.004*"게임" + 0.003*"일본" + 0.003*"커플" + 0.003*"연애" + 0.003*"스타"')
(21, '0.008*"티빙" + 0.004*"게임" + 0.004*"넷플릭스" + 0.004*"연애" + 0.004*"유튜브"')
(7, '0.011*"연애" + 0.004*"환승" + 0.004*"티빙" + 0.004*"솔로" + 0.004*"출연자"')
(14, '0.007*"지옥" + 0.005*"솔로" + 0.004*"넷플릭스" + 0.004*"게임" + 0.003*"이효리"')
(10, '0.012*"이효리" + 0.004*"넷플릭스" + 0.003*"김호중" + 0.003*"스타" + 0.003*"광고"')
(27, '0.009*"연애" + 0.003*"넷플릭스" + 0.003*"스타" + 0.003*"남매" + 0.003*"결혼"')
```



## 결과 해석

영화의 LDA 결과와 마찬가지로 많은 토픽에서 단어 간의 연관성이 떨어지는 결과를 확인했다.

단순한 단어 빈도수 대신 **TF-IDF**를 사용하여 단어의 중요도를 반영할 수 있도록

추가 분석을 진행하였다.

→ 추가 분석 진행



# 2022-2024년 예능

TF-IDF 벡터화를 수행하고, 각 문서의 상위 10개의 TF-IDF 값을 출력

```
# TF-IDF 벡터화
tfidf_vectorizer = TfidfVectorizer(max_df=0.95, min_df=1)
tfidf_matrix = tfidf_vectorizer.fit_transform(df['preprocessed_body'])

# TF-IDF 결과를 데이터프레임으로 변환
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf_vectorizer.get_feature_names_out())

# 결과 출력 함수
def print_tfidf_results(tfidf_df, document_ids, top_n=10):
    for idx in document_ids:
        print(f"Document {idx}:")
        sorted_items = sorted(tfidf_df.loc[idx].items(), key=lambda x: x[1], reverse=True)[:top_n]
        for item in sorted_items:
            if item[1] > 0:
                print(f"\t{item[0]}: {item[1]:.4f}")
        print("-" * 20)

# 문서 ID 리스트
document_ids = list(range(10)) # 필요한 만큼의 문서 인덱스 지정

# TF-IDF 결과 출력 (상위 10개의 TF-IDF 값만 출력)
print_tfidf_results(tfidf_df, document_ids, top_n=10)
```

## 결과 해석

각 문서에서 중요한 단어들이 추출된 결과, LDA보다 주제 분류가 명확해졌음을 알 수 있다. 그러나 여전히 프로그램명보다는 프로그램 내의 특정 요소나 인물들이 더 두드러지는 경향이 있다. 이를 통해 TF-IDF가 문서의 핵심 키워드를 잘 포착하지만, 프로그램명의 결과를 보는 데에는 한계가 있음을 알 수 있다. 추가적인 불용어 처리가 필요할 것으로 보인다.

Document 0:	우석: 0.7021
	런닝맨: 0.3262
	배고픔: 0.2226
	식사: 0.1958
	뻔 우: 0.1833
	뻔 우석: 0.1833
	그림: 0.1587
	탄생: 0.1250
	재미: 0.1186
	난입: 0.1158
Document 1:	특산물: 0.3130
	로컬: 0.2026
	안정환: 0.1866
	장윤정: 0.1866
	팝업: 0.1608
	광고: 0.1579
	가전: 0.1565
	녹색: 0.1458
	품목: 0.1346
	엔터테인먼트: 0.1333
Document 2:	분기: 0.6111
	넷플릭스: 0.3627
	주연: 0.1721
	소개: 0.1466
	박정민: 0.1160
	원작: 0.1128
	소재: 0.1057
	좀비: 0.1009
	촬영: 0.0919
	미스터리: 0.0824
Document 3:	정국: 0.3806
	여주: 0.3453
	밀림: 0.2732
	수식: 0.2409
	갯벌: 0.2020
	저택: 0.1876
	게임: 0.1856
	카드: 0.1787
	만점자: 0.1766
	열쇠: 0.1614

Document 4:	지락: 0.3088
	세븐: 0.2927
	자식: 0.2613
	등짝: 0.2327
	뛰뛰: 0.2016
	여권: 0.2016
	부모: 0.1891
	여행: 0.1815
	레전드: 0.1718
	로컬: 0.1682
	백종원: 0.1587
Document 5:	세븐: 0.2927
	쿠팡: 0.2827
	연애: 0.2634
	소년: 0.2604
	플레이: 0.2330
	시대: 0.2081
	여연자: 0.1904
	환승: 0.1878
	로버트: 0.1675
	바디: 0.1516
Document 6:	고준희: 0.4945
	김의성: 0.3744
	인생: 0.3584
	홍진호: 0.3192
	맞선: 0.2152
	진실: 0.1761
	설정: 0.1658
	고준: 0.1654
	어플: 0.1490
	뉴스: 0.1458
Document 7:	카리나: 0.6152
	이재욱: 0.4555
	운세: 0.2468
	에스: 0.2411
	열애: 0.2251
	에스파: 0.1799
	조명: 0.1216
	오미쿠지: 0.1201
	윤민: 0.1138
	대길: 0.1094

Document 8:	영숙: 0.5243
	영철: 0.5243
	결혼: 0.2983
	광수: 0.2182
	커플: 0.1935
	솔로: 0.1904
	내년: 0.1078
	국주행으: 0.1035
	라이브: 0.0961
	김경빈곽다영: 0.0912
Document 9:	대호: 0.6534
	오빠: 0.2768
	레드카펫: 0.2602
	핫플: 0.1767
	이효리: 0.1700
	이대호: 0.1682
	미식가: 0.1284
	팔도: 0.1235
	먹방: 0.1224
	시즌즈: 0.1178

# 모델링 3 - 딥러닝 (Logistic Regression)

```
# 여행 텍스트 파일에서 여행지 및 여행 관련 키워드 읽기
def load_travel(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        travel_list = file.read().splitlines()
    return travel_list

# 파일에서 여행 목록 로드
travel_file_path = '../travel.txt'
file_travel = load_travel(travel_file_path)
```

여행지 텍스트 파일에서 키워드 읽기

## 딥러닝

각 연도별 여행지 데이터에서 계절별 키워드가 뚜렷하게 나타났다는 것을 활용해 계절을 맞춰보자!

```
# 봄 데이터프레임 생성
spring_2020 = filter_inclusive(df_2020, '봄')
spring_2020 = filter_exclusive(spring_2020, ['여름', '가을', '겨울'])
# 계절이라는 컬럼을 만들어서 '봄' = 0 이라고 명시
spring_2020['계절'] = 0
```

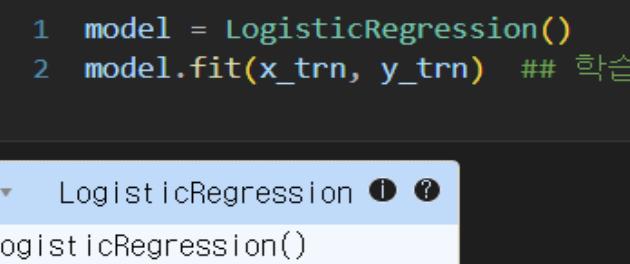
봄의 개수: (543, 8)  
여름의 개수: (484, 8)  
가을의 개수: (365, 8)  
겨울의 개수: (529, 8)

각 연도별로 계절을 나누고 타겟 칼럼을 만들기

```
from nltk.tokenize import TreebankWordTokenizer
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer()
tfidf_mat = tfidf.fit_transform(df_travel['keywords'])
```

```
1 model = LogisticRegression()
2 model.fit(x_trn, y_trn) ## 학습
```



The screenshot shows the Python code for creating a Logistic Regression model. A tooltip for the 'LogisticRegression' class is displayed, showing its constructor and methods.

TF-IDF 벡터를 생성하고 train, test 데이터로 나눠서 Logistic Regression 학습

```
1 from sklearn.metrics import accuracy_score
2
3 accuracy_score(y_tst, y_pred)
```

0.6727272727272727

정확도 예측

# 2020-2024 추천 여행지 → 키워드만 보고 계절 맞추기!

봄의 개수: (543, 8)  
여름의 개수: (484, 8)  
가을의 개수: (365, 8)  
겨울의 개수: (529, 8)

```
1 model = LogisticRegression()  
2 model.fit(x_trn, y_trn) ## 학습  
  
▼ LogisticRegression ⓘ ⓘ  
LogisticRegression()
```

```
1 from sklearn.metrics import accuracy_score  
2  
3 accuracy_score(y_tst, y_pred)  
  
0.6727272727272727
```

이렇게 진행했습니다!

1. 타겟의 고윳값을 확인하고 클래스 불균형이 크지 않다는 것을 확인
2. Logistic Regression을 fit() 메소드를 활용해 직접 학습시킴
3. accuracy\_score()를 이용해 정확도 확인

정확도가 조금 낮게 나왔다...!!

- 너무 다양한 여행지 키워드가 섞여있어서 집중해야 할 키워드에 모델이 집중하지 못하게 될 수 있음
- 하이퍼 파라미터 조정을 따로 하지 않았기 때문에 이로 인해 정확도가 낮게 나타날 수 있음
- 학습 데이터의 수가 너무 적어서 충분한 학습이 이루어지지 못해 정확도가 낮게 나타날 수 있음

직접 네트워크를  
짜보자!



# 모델링 3 - 딥러닝 (Model 생성)

```
# 여행 텍스트 파일에서 여행지 및 여행 관련 키워드 읽기
def load_travel(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        travel_list = file.read().splitlines()
    return travel_list

# 파일에서 여행 목록 로드
travel_file_path = '../travel.txt'
file_travel = load_travel(travel_file_path)
```

여행지 텍스트 파일에서 키워드 읽기

## 딥러닝

각 연도별 여행지 데이터에서 계절별 키워드가 뚜렷하게 나타났다는 것을 활용해 계절을 맞춰보자!

```
# 봄 데이터프레임 생성
spring_2020 = filter_inclusive(df_2020, '봄')
spring_2020 = filter_exclusive(spring_2020, ['여름', '가을', '겨울'])
# 계절이라는 컬럼을 만들어서 '봄' = 0 이라고 명시
spring_2020['계절'] = 0
```

봄의 개수: (543, 8)  
여름의 개수: (484, 8)  
가을의 개수: (365, 8)  
겨울의 개수: (529, 8)

각 연도별로 계절을 나누고 타겟 칼럼을 만들기

```
from nltk.tokenize import TreebankWordTokenizer
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer()
tfidf_mat = tfidf.fit_transform(df_travel['keywords'])
```

```
def forward(self, x):
    # (2) Input Flow 짜기
    h1 = self.fc1(x)
    h1 = torch.sigmoid(h1)
    h2 = self.fc2(h1)
    out = F.log_softmax(h2, dim=1)

    return out
```

TF-IDF 벡터를 생성하고 train, test 데이터로 나누고 네트워크 생성

```
Epoch: 01 | Time: 0m 0s
      Train Loss: 0.487 | Test Loss: 1.181 | Test Acc: 62
Epoch: 02 | Time: 0m 0s
      Train Loss: 0.489 | Test Loss: 1.188 | Test Acc: 62
Epoch: 03 | Time: 0m 0s
      Train Loss: 0.488 | Test Loss: 1.186 | Test Acc: 63
```

랜덤 모델의 정확도는 (in training set) 90.30%입니다.

모델 학습 및 정확도 예측

# 2020-2024 추천 여행지 → 키워드만 보고 계절 맞추기!

```
class MyNet(nn.Module): # nn.Module을 상속받아서 MyNet이라는 class를 생성
    def __init__(self, in_dim, h1_dim, out_dim):
        super(MyNet, self).__init__()

        # (1) 네트워크 구조 짜기 (with 단위 모듈)
        self.fc1 = nn.Linear(in_dim, h1_dim) # input -> 1st hidden layer에 대한 FFNN
        self.fc2 = nn.Linear(h1_dim, out_dim) # 1st -> output layer에 대한 FFNN

    ## 어떻게 연결이 되는지 정의 -> x라는 인풋 데이터가 들어왔을 때 인풋에서 첫번째 레고를 거쳐가면서 중간에 어떤 값이 나오고...
    ## 층을 2개 이상 쌓으면 과적합 증세 (?)가 나타나서 ... 그 층을 한 개만 쌓아봄
    def forward(self, x):
        # (2) Input Flow 짜기
        h1 = self.fc1(x) # step2
        h1 = torch.sigmoid(h1) # step3
        h2 = self.fc2(h1) # step4
        out = F.log_softmax(h2, dim=1) # step5

    return out
```

## 모델 클래스 생성 & 네트워크 구조 짜기

- 히드 레이어를 3개 이상 쌓아서 학습 진행
  - Train Loss는 점점 내려가는 반면, Test Loss는 점점 올라가 과적합이 있다고 판단
- 히든 레이어를 1개 쌓아서 학습 진행
  - 과적합 양상이 보이진 않음
- 히든 레이어를 2개 쌓아서 학습 진행
  - 과적합이 되어 보이지 않았고 정확도가 1개 쌓은 것보다 높았음

→ 히든 레이어 2개로 진행!

```
## model이라는 변수에 실제 뉴럴 네트워크 사용
```

```
model = MyNet(in_dim=196, # 입력하는 데이터의 차원 (196)
               h1_dim=64, # 1st hidden layer의 neuron 수 (64)
               out_dim=4) # data의 class 수 = 4 -> 계절
model = model.to(device) # network를 'gpu' 혹은 'cpu'에 옮겨놓읍시다.
```

## model이라는 객체 생성

- 입력 데이터 차원은 196이기 때문에 input 차원을 196으로 설정
- 첫번째 히든 레이어 뉴런 수는 64개로 설정
- 모델이 사계절 중 하나로 분류해야 하기 때문에 output 차원은 4개로 설정

# 2020-2024 추천 여행지 → 키워드만 보고 계절 맞추기!

```
# 알아서 매번 미니배치를 만들어주는 미니배치 생성기  
BATCH_SIZE = 200  
  
train_dataset = TensorDataset(x_trn, y_trn)  
test_dataset = TensorDataset(x_tst, y_tst)  
  
trn_loader = DataLoader(dataset=train_dataset,  
                        batch_size=BATCH_SIZE,  
                        shuffle=True,  
                        drop_last=False)  
  
tst_loader = DataLoader(dataset=test_dataset,  
                        batch_size=BATCH_SIZE,  
                        shuffle=False,  
                        drop_last=False)
```

## Mini-batch 데이터 생성기 만들기

- Batch size를 200으로 설정
- Train Loader만 shuffle 인자를 True로 설정  
(데이터셋을 섞을지에 대한 여부를 결정)
  - 모델이 데이터의 순서에 의존하지 않도록 하여 일반화 성능을 향상시키는 데에 도움을 줌

```
my_opt = optim.Adam(params = model.parameters(), lr = 2e-3)
```

## Optimizer & Learning rate 설정

- Adam을 Loss Function으로 설정했을 때가 가장 성능이 좋아 Adam으로 설정
- Learning rate는 크게 해보고 점차 작게 줄여가면서 학습시켜보고 성능이 그나마 좋고 loss가 가장 낮은 2e-3로 설정
  - lr = 2e-4일 때 ... trn\_acc = 51.63%, tst\_acc = 48.052%
  - lr = 2e-3일 때 ... trn\_acc = 78.45%, tst\_acc = 65.974% (Loss가 매우 낮음)
  - lr = 2e-2일 때 ... trn\_acc = 94.92%, tst\_acc = 66.494%
    - 과적합이 엄청남 (심지어 lr = 2e-1로 하면 trn\_acc = 99%...?)
    - Test Loss가 굉장히 크고 Train Loss가 점점 작아지는 양상을 보임
  - lr = 1e-2일 때 ... trn\_acc = 81.32%, tst\_acc = 62.597%
    - 역시나 이것도 과적합...

※ shuffle = True로 했기 때문에 매번 정확도가 같게 나오지 않음을 유의

# 2020-2024 추천 여행지 → 키워드만 보고 계절 맞추기!

```
def train(model, data_loader, optimizer, device):
    model.train() # 학습모드
    trn_loss = 0
    for i, (x, y) in enumerate(data_loader):
        # Step 1. mini-batch에서 x,y 데이터를 얻고, 원하는 device에 위치시키기
        x = x.view(-1, 196).to(device)
        y = y.to(device)

        # Step 2. gradient 초기화
        optimizer.zero_grad()

        # Step 3. Forward Propagation
        y_pred_prob = model(x)

        # Step 4. Loss Calculation
        loss = F.nll_loss(y_pred_prob, y, reduction='sum')

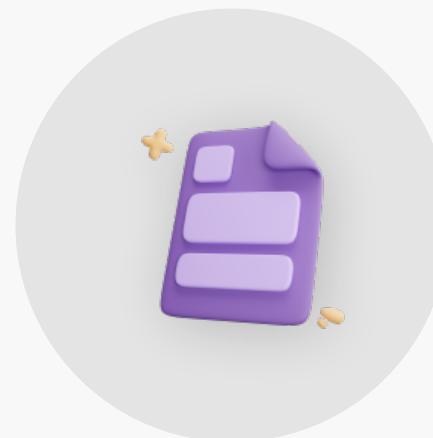
        # Step 5. Gradient Calculation (Backpropagation)
        loss.backward()

        # Step 6. Update Parameter (by Gradient Descent)
        optimizer.step()

        # Step 7. trn_loss 변수에 mini-batch loss를 누적해서 합산
        trn_loss += loss.item()

    # Step 8. 데이터 한 개당 평균 train loss
    avg_trn_loss = trn_loss / len(data_loader.dataset)
    return avg_trn_loss
```

결과



최종

Train Loss: 0.604 | Test Loss: 0.958 | Test Acc: 64.675%

## Forward Propagation 수행

- 하나의 Mini-batch data에 대해 Forward Propagation 수행
- 전체 데이터를 50번 반복
- `model.train()` 함수를 사용해 모델을 학습모드로 해서 `train` 함수 만들
- `model.eval()` 함수를 사용해 모델을 평가모드로 해서 `evaluate` 함수를 만들어 평가 모드로도 사용함

- 유의점 1: 데이터 수가 너무 적어 과적합을 피하기 어려웠음
- 유의점 2: 정확도가 높은 것보다 loss가 낮은 것을 선택해 정확도가 크게 높지 않음

Ah-!



# 결론 및 한계점

## 결론

1. 일상적인 단어들이 비교적 적은 드라마, 영화 같은 경우는 워드 클라우드가 다른 토픽들에 비해 잘 나왔다.
2. 블로그 특성 상, 주제 관련 단어에 비해 일상적인 단어의 빈도가 압도적으로 높다.  
→ 수집한 데이터의 특성을 잘 파악해서 **추가적으로 불용어를 처리할 필요성이 있다.**
3. 각 토픽 별로 연도별 차이가 유의미하게 나타난 경우가 대다수였으며 특히 특정 단어에 관해 찾아봤을 때 해당 연도에 관련 이슈가 있어 흥미로웠다. 이는 사람들이 직접 작성한 **블로그** 같은 글을 데이터로 분석했기 때문이라고 판단했다.

## 한계점

1. 설치 어려움으로 해보지 못했지만 **다른 한국어 전처리 패키지도 사용해봤으면 좋았을 것 같다.**  
ex. konlpy
2. 일상적인 단어에서 비롯된 유행어나 신조어 같은 경우 **불용어 처리가 어려웠다.**  
때문에 불용어 처리를 일일이 해야 해서 많은 시간이 소요됐다.
3. 모델링 중 딥러닝의 경우. 데이터의 수가 적어 **과적합을 피하고 성능을 높이기가 어려웠다.**
4. 유행어 자체가 일상적으로 많이 사용되는 단어 (ex. 문제, 엄마)라면 이것이 해당 연도의 유행어로 많이 쓰인 것인지, 아니면 그냥 일상적인 단어로 많이 쓰인 것인지 알 수 없다.