정치커뮤니케이션 데이터분석실습(GCO2014) 과제 1

과제를 하시면서 참고할 사항

완성된 과제는 pdf 형태로 출력해서 iCampus에 제출해주세요.이 과제는 전체 성적의 10%를 차지하며, 과제의 마감은 **11월 26일 (금) 오후 11:59분**까지 입니다. 마감 시각 이후에 제출된 과제는 감점이 존재합니다 (강의계획서 참조).

1. 과제를 작성하시기 전에

- **꼭 본인의 구글 드라이브에 복사한 후 과제를 하세요.**
- 서명에 본인의 학번과 이름을 적어주세요.
- 코렙링크에 본인의 노트북 링크를 복사해주세요.

2. 과제를 작성하실 때

- 부교재나 다른 파이썬 교재를 참고하셔도 좋으나, 각 문제를 해결하기 위한 코드는 스스로 작성하시길 바랍니다.
- 코드를 작성하시면서 모든 코드에 주석을 다시길 바랍니다. 코드를 잘 이해하고 있는지를 보기 위해서 입니다 (아래의 예 참조)

```
a = 100 # a 변수에 100을 저장
a += 100 # a 변수에 100을 더한 결과를 a 변수에 저장
print(a) # a 변수의 값을 출력
```

3. 과제를 제출하실 때

- 문제를 해결하기 위한 코드는 실행 후 결과 값이 출력되어야 문제를 해결한 것으로 간주 하겠습니다.
- 노트북을 pdf 형태로 출력한 후 iCampus의 해당 과제물을 제출하셔야 과제를 제출한 것으로 인정됩니다.
- 과제 제출 기한을 넘기는 경우, 강의계획서에 공지한 것처럼 매 24시간 마다 1점씩 감점됩니다. `감점 = ((초과시간//24)+1)`

서명

저는 제 스스로 이 과제를 해결했으며, 제가 스스로 코드를 작성했습니다. 서명(예: 2020####23_김민철): 2021312108_송병도

코렙 링크

코랩링크: 본인의 코랩 노트북의 링크를 여기에 복사해 주세요.

https://colab.research.google.com/drive/1kNeS9H_BW4l83rygohp_qU1qPqOKRMMX?usp=sharing

데이터 셋

분석에 사용한 데이터를 구글 드라이브에 저장 후, 해당 드라이브를 공유해 주세요. **해당 드라이브의 데이터 내용을 확인할 수 있도록, 링크를 가진 모든 사람이 데이터를 볼 수 있는 권한을 주셔야 합니다.**

데이터셋 링크: 본인이 분석에 사용한 데이터 셋의 링크를 여기에 복사해주세요.

https://docs.google.com/spreadsheets/d/1zylhqxcYclNmsPwAtFBLSzkMuqVxEfG2/edit?usp=sharing&ouid=100152007294750602109&rtpof=true&sd=true

---

과제 설명:

이 과제는 학생 여러분 스스로 관심있는 주제를 정해 수업 시간에 배운 실습 내용(토픽 모델링)을 적용해 스스로 정치커뮤니케이션 분석을 실시하는 것을 목적으로 합니다. 실제 해결해야할 문제가 주어진 파이썬 실습 과제와는 다르게 학생 여러분 스스로 주제를 선택해 수업에서 다룬 내용을 활용해 분석을 실시하시면 됩니다.

가능하면 과제 전체의 내용을 파악한 후 과제를 시작해 주세요. 실습 과제는 다음 내용이 포함되어 있습니다. 각 부분에 포함되어야 하는 부분은 아래에 좀 더 자세히 기술되어 있습니다. </br

참조: 해당 단계마다 제시되는 예들은 단계별 가이드라인이며, 반드시 똑같은 분석을 실시해야하는 것은 아닙니다.

배점

1. 문제제기 (해결하고자 하는 문제 기술)
2. 데이터에 대한 기술 (수집 방법/적합성) - 0.5점
3. 데이터 탐색적 분석 (데이터에 대한 기술) - 1점
4. 데이터 전처리 (전처리 과정 기술) - 2점
5. 토픽 모델 학습 (토픽 모델링 수행) - 2점
6. 토픽 레이블링 (토픽에 이름 붙이기) - 2점
7. 분석 결과 시각화 (결과의 시각화) - 1.5점
8. 분석 결과 논의 - 1점

---

1. 문제제기 및 예측

여기에서는 토픽 모델링을 사용하여 답을 구해보려는 문제를 기술해 보세요. 자신이 해당 문제를 풀기 ...

참조: 아래의 예를 참조해서 토픽 모델링을 수행하기 위해 본인 관심이 있는 주제를 선정해 보세요.

예) 정파성에 따른 국내 언론의 코로나에 관한 보도 분석 비교할 계획입니다. 2020년부터 2021년까지 국내 언론사의 코로나 관련 보도를 분석하여 언론사의 정치 성향에 따라 코로나의 어떠한 측면을 부각시켰는 지를 분석하고자 합니다. 상대적으로 진보적 성향을 띈 매체 (예: 경향/한겨레)가 상대적으로 보수적 색체를 지닌 언론들(예: 조선/동아)가 서로 다른 부분에 초점을 맞추었을 것으로 예상합니다.

[문제제기 및 예측] 정파성에 따른 국내 언론의 대북 정책에 관한 보도 분석 비교를 진행할 계획입니다. 2017년 5월 10일 부터 현재까지 (문재인 정부 집권 시기) 국내 언론사의 대북 정책 관련 보도를 분석하여 언론사의 정치 성향에 따라 대북 정책에 대해 어떠한 측면을 부각시켰는 지를 분석하고자 합니다. 상대적으로 진보적 성향을 띈 매체 (경향/한겨레)와 상대적으로 보수적 색체를 지닌 언론들 (중앙/동아)이 각자의 정치 성향에 맞는, 서로 다른 부분에 초점을 맞추었을 것으로 예상합니다.

2. 데이터 수집과정 및 데이터 적합성 (0.5점)

여기에는 분석을 위한 데이터 수집과정과 데이터가 왜 본인이 해결하고자 하는 문제에 적절한지에 대해...

참고: Bigkinds [https://www.bigkinds.or.kr/]나 아니면 기존의 데이터 아카이브를 사용하여 분석에 필요한 데이터를 구하셔도 됩니다. 빅카인즈의 경우 한 번에 2만개의 데이터만을 다운도르 할 수 있기 때문에, 빅카인즈를 사용하여 데이터를 분석하기 위해서는 시기를 조금씩 변경하여 데이터를 수집하여야 합니다.

- **만약 분석에 사용하는 데이터셋에 데이터(예: 사설, 기사, 댓글 등)가 10,000개 이하라면 다른 주제를 찾아 보시길 권합니다.**
- 웹스크랩핑을 통해 직접 데이터를 수집하시는 경우 **보너스 점수(최대 2점)**를 드리도록 하겠습니다. 만약 웹스크랩핑을 통해 데이터를 수집하는 경우 데이터 수집을 위해 사용한 코드도 함께 제시되어야 합니다.
- 만약 지지율의 변화 등, 텍스트 이외의 외부 데이터를 사용하시는 경우 해당 데이터에 대한 설명 또한 필요합니다. 외부 데이터를 함께 사용하여 분석을 실시하는 경우 **보너스 점수(최대 1점)**를 드리도록 하겠습니다.

예) 분석에 사용한 데이터는 빅카인즈를 통해 수집했습니다. 한국에 첫 코로나 환자가 발생한 2020년 1월부터 2021년까지를 수집 범위로 지정했습니다. '코로나'라는 키워드를 포함한 기사들을 모두 수집했으며, 정파성에 따른 차이 비교를 용이하게 하기 위해서 진보 성향으로 인식되는 두 개의 매체(한겨레/경향)와 보수 성향으로 인식되는 두 개의 매체(조선/동아)의 사설들만을 사용했습니다.

분석에 사용한 데이터는 빅카인즈를 통해 수집했습니다. 현재의 정권 체제에서의 대북 정책에 대한 데이터 분석을 목표로 하였기에 문재인 정부가 집권한 2017년 5월 10일부터 현재 2021년 11월 26일까지를 수집 범위로 지정했습니다. '대북'이라는 키워드를 포함한 기사들 중 '정치'카테고리에 해당하는 기사들을 모두 수집했으며, 정파성에 따른 차이 비교를 용이하게 하기 위해서 진보 성향으로 인식되는 두 개의 매체(한겨레/경향)와 보수 성향으로 인식되는 두 개의 매체(조선/동아)의 사설들만을 사용했습니다.

데이터 탐색적 분석 (1점)

데이터의 기본적인 특성에 대해서 통계적 수치를 보여주시면 됩니다.

참고: 평균(mean), 표준편차(std), 데이터수(count) 등 데이터의 기본적인 특성을 보여주시며 됩니다. 파이썬의 복습 시간의 판다스 강좌를 참고하시면 됩니다. 만약, 외부의 데이터 (예: 대통령 지지율)를 사용한다면 해당 데이터 또한 탐색적 분석을 실시해야 합니다.

탐색적 분석의 예)

1. 데이터의 전체 개수를 데이터의 탐색적 분석을 위해 각 사설의 개수가 시간 별로 어떻게 변하는 지를 보여줍니다.
2. 신문사별로 매 월 '코로나 관련' 사설 수의 변화와 월 평균 사설 수의 변화를 함께 보여 줍니다.

```
1 1. 연-월 별 각 언론사에서 기고한 사설 개수의 변화를 시각화 하였습니다.
```

```
1 #한글 출력 위한 폰트 설치
2 !apt-get update -qq
3 !apt-get install fonts-nanum* -qq
```

```
Selecting previously unselected package fonts-nanum.
(Reading database ... 155222 files and directories currently installed.)
Preparing to unpack .../fonts-nanum_20170925-1_all.deb ...
Unpacking fonts-nanum (20170925-1) ...
Selecting previously unselected package fonts-nanum-eco.
Preparing to unpack .../fonts-nanum-eco_1.000-6_all.deb ...
Unpacking fonts-nanum-eco (1.000-6) ...
Selecting previously unselected package fonts-nanum-extra.
Preparing to unpack .../fonts-nanum-extra_20170925-1_all.deb ...
```

```
Unpacking fonts-nanum-extra (20170925-1) ...
Selecting previously unselected package fonts-nanum-coding.
Preparing to unpack .../fonts-nanum-coding_2.5-1_all.deb ...
Unpacking fonts-nanum-coding (2.5-1) ...
Setting up fonts-nanum-extra (20170925-1) ...
Setting up fonts-nanum (20170925-1) ...
Setting up fonts-nanum-coding (2.5-1) ...
Setting up fonts-nanum-eco (1.000-6) ...
Processing triggers for fontconfig (2.12.6-0ubuntu2) ...
```

```
#필요한 모듈 모두 불러오기

import pandas as pd

from tqdm import tqdm # apply 진행사항 표시
tqdm.pandas()

import re # 정규표현식

#시각화에 필요한 패키지 불러오기
import seaborn as sns
import matplotlib.pyplot as plt

# 계산에 필요한 라이브러리
from math import sqrt
import numpy as np

import tomotopy as tp
```

```
#데이터 불러오기
data = pd.read_excel("northKoreaData.xlsx", sheet_name="sheet")
data.describe()
```

	뉴스 식별자	일자
count	2.000000e+04	2.000000e+04
mean	1.100635e+06	2.019003e+07
std	3.499788e+02	1.189086e+04
min	1.100101e+06	2.017092e+07
25%	1.100401e+06	2.018053e+07
50%	1.100901e+06	2.019033e+07
75%	1.100901e+06	2.020062e+07
max	1.101001e+06	2.021113e+07

```
#칼럼별 데이터 정보
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
```

```
Data columns (total 19 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   뉴스 식별자           20000 non-null  float64
 1   일자                20000 non-null  int64
 2   언론사               20000 non-null  object
 3   기고자               18175 non-null  object
 4   제목                20000 non-null  object
 5   통합 분류1            20000 non-null  object
 6   통합 분류2            18412 non-null  object
 7   통합 분류3            14092 non-null  object
 8   사건/사고 분류1          3161 non-null   object
 9   사건/사고 분류2           712 non-null    object
 10  사건/사고 분류3           235 non-null    object
 11  인물                18922 non-null  object
 12  위치                19836 non-null  object
 13  기관                19995 non-null  object
 14  키워드               20000 non-null  object
 15  특성추출(가중치순 상위 50개)  20000 non-null  object
 16  본문                20000 non-null  object
 17  URL               20000 non-null  object
 18  분석제외 여부           270 non-null    object
dtypes: float64(1), int64(1), object(17)
memory usage: 2.9+ MB
```

```
#칼럼 목록 출력
data.columns
```

```
Index(['뉴스 식별자', '일자', '언론사', '기고자', '제목', '통합 분류1', '통합 분류2', '통합 분
       '사건/사고 분류1', '사건/사고 분류2', '사건/사고 분류3', '인물', '위치', '기관', '키워
       '특성추출(가중치순 상위 50개)', '본문', 'URL', '분석제외 여부'],
      dtype='object')
```

```
# 컬럼 이름 지정
data.columns = ['id','date','news_org','upload','title','sec1','sec2','sec3',
                'cat1','cat2','cat3',
                'people','location','orgs',
                'keywords',"attributes","maintext","url","exclude"]
```

```
import datetime as dt # 시간 데이터를 처리하기 위한 패키지 불러오기
data['date'] = data['date'].astype("str") #데이터 형식을 문자열로
data['date'] = data['date'].apply(lambda x: dt.datetime.strptime(x,'%Y%m%d')) # 시간 데이터로 바
```

```
#칼럼별 데이터 정보
data.info()
```

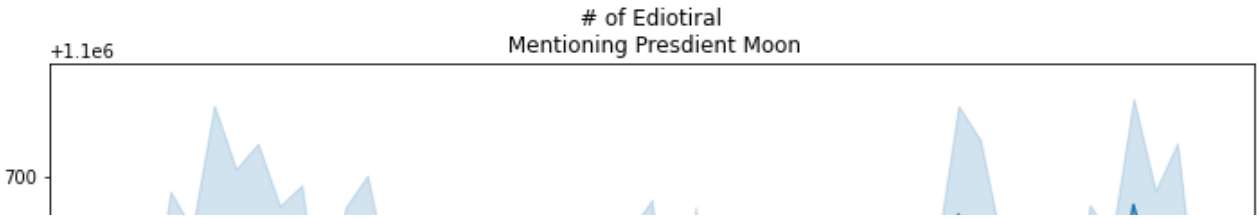
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 19 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   id          20000 non-null  float64
 1   date        20000 non-null  datetime64[ns]
 2   news_org    20000 non-null  object
 3   upload      18175 non-null  object
```

```
 4   title       20000 non-null  object
 5   sec1        20000 non-null  object
 6   sec2        18412 non-null  object
 7   sec3        14092 non-null  object
 8   cat1        3161 non-null   object
 9   cat2        712 non-null    object
 10  cat3        235 non-null    object
 11  people      18922 non-null  object
 12  location    19836 non-null  object
 13  orgs        19995 non-null  object
 14  keywords    20000 non-null  object
 15  attributes  20000 non-null  object
 16  maintext    20000 non-null  object
 17  url         20000 non-null  object
 18  exclude     270 non-null    object
dtypes: datetime64[ns](1), float64(1), object(17)
memory usage: 2.9+ MB
```

```
data['year-month'] = data['date'].dt.strftime('%Y-%m') # 연-월의 형식으로 변경하기 위함
raw_data = data.groupby('year-month').count()['id'] # 연-월별 사설 개수 세기
raw_data = raw_data.reset_index() # year-month 를 컬럼으로 변경
```

```
plt.figure(figsize=(12,8)) # 이미지 사이즈 정하기 (가로, 세로)
ax = sns.lineplot(x='year-month',y='id',data=data) # 원 데이터를 사용하여 데이터 시각화 매
# x축의 값 변경위해서 raw_data사용
freq = 10
ax.set_xticklabels(raw_data.iloc[::freq]["year-month"]) # x 축에 표시할 레이블 선택
xtix = ax.get_xticks()
ax.set_xticks(xtix[::freq]) # x 축에 레이블을 표시한 틱을 선택

# 데이터 정보 표시
ax.set(title='# of EdiotiralWn Mentioning Presdient Moon', xlabel="Year-Month", ylabel="Counts")
```

```
[Text(0, 0.5, 'Counts'),
 Text(0.5, 0, 'Year-Month'),
 Text(0.5, 1.0, '# of EdiotiralWn Mentioning Presdient Moon')]
```



데이터 전처리 및 정제 (2점)

데이터를 분석할 수 있도록 데이터를 처리하는 단계 입니다.

참고: 워드 클라우드 이후에 사용된 텍스트 전처리 관련 부분을 참고하시면 됩니다. 이 때, 왜 특정한 전처리 방식을 적용하였는지에 대해 가능하면 자세히 설명해주세요.

전처리 설명의 예)

1. 코로나의 경우 코로나 와 코로나19, 코비드 등으로 용어가 함께 사용되는 경우가 많아서 코로나 관련 단어들을 모두 코로나 로 통일하였습니다.
2. 형태소 분석기가 정은경 본부장 등의 인명을 제대로 분석하지 못해 사용자 지정 사전을 만들어 추가하였습니다.

---

1. 아래의 코드에서 알 수 있듯, 텍스트 전처리 방법 중 정규화를 수행하였습니다. 이는 여러 기사에서 등장하는 인물들의 지칭 표현을 통일해주기 위함이었습니다.

```
!pip install PyKomoran # 전처리를 위해 형태소 분석기 설치
```

```
Requirement already satisfied: PyKomoran in /usr/local/lib/python3.7/dist-packages (0.1.6.pos
Requirement already satisfied: py4j==0.10.9.2 in /usr/local/lib/python3.7/dist-packages (from
```

```
from PyKomoran import * # 형태소분석기 불러오기
komoran = Komoran("EXP") # 코모란을 사용하기 위한 객체 만들기
```

```
#[noun 추출하기]
data['tokens'] = data['maintext'].progress_map(lambda x:komoran.get_nouns(x))
```

```
100%|██████████| 20000/20000 [02:20<00:00, 142.25it/s]
```

```
#[한글자 단어 제거하기]
data['tokens2'] = data['tokens'].progress_map(lambda x:[w for w in x if len(w) > 1])
```

```
100%|██████████| 20000/20000 [00:00<00:00, 116249.14it/s]
```

```
import random
i = random.randint(0, len(data))

#[특수기호 없애기]
pattern = '[-=+,#/W?:;^$.A*"~%!WWWnWrWt▼♣§☆♡()%´∀ ' ノ♥▲▶*— “” · ‘’ ■⦿◇△◇『』]

text = re.sub(pattern, ' ', data['maintext'][i]).strip()
print(text)
```

```
정의용 실장의 극적인 백악관 앞뜰 회견이 열리기 직전  도쿄에서 또 하나의 기자회견이 열렸다  아
```

```
#[단어 정규화하기]
dic_standardization = {
    '문 대통령':'문재인 대통령', '윤 전 총장' : '윤석열', '윤' : '윤석열', '이' : '이재명', '윤
        '김 위원장':'김정은 북한 국무위원장', '김정은 국무위원장':'김정은 북한 국무위원장', '김

for old, new in dic_standardization.items():
  print("{}을(를) {}으로 변환 중입니다.".format (old, new))
  data['maintext'] = data['maintext'].progress_map(lambda x:x.replace(old, new))
```

```
문 대통령을(를) 문재인 대통령으로 변환 중입니다.
100%|██████████| 20000/20000 [00:00<00:00, 424322.84it/s]
윤 전 총장을(를) 윤석열으로 변환 중입니다.
100%|██████████| 20000/20000 [00:00<00:00, 501537.03it/s]
윤을(를) 윤석열으로 변환 중입니다.
100%|██████████| 20000/20000 [00:00<00:00, 466466.56it/s]
이을(를) 이재명으로 변환 중입니다.
100%|██████████| 20000/20000 [00:00<00:00, 367108.29it/s]
윤 후보을(를) 윤석열으로 변환 중입니다.
100%|██████████| 20000/20000 [00:00<00:00, 458363.82it/s]
이 후보을(를) 이재명으로 변환 중입니다.
100%|██████████| 20000/20000 [00:00<00:00, 473574.13it/s]
김 위원장을(를) 김정은 북한 국무위원장으로 변환 중입니다.
100%|██████████| 20000/20000 [00:00<00:00, 496455.47it/s]
김정은 국무위원장을(를) 김정은 북한 국무위원장으로 변환 중입니다.
100%|██████████| 20000/20000 [00:00<00:00, 372888.34it/s]
김정은을(를) 김정은 북한 국무위원장으로 변환 중입니다.
100%|██████████| 20000/20000 [00:00<00:00, 507646.71it/s]
```

토픽 모델 학습 (2점)

이 파트는 정제 된 데이터에 실제 분석 알고리즘을 적용하는 단계입니다.

참고: 토픽 모델링을 수행시 최적의 K값을 구하는 과정을 보여주시고, 이에 대해 설명하시면 됩니다.

모델 학습 과정의 예:)

1. 여러 K값을 사용하여 모델을 훈련시킨 결과 Perplexity 점수의 변화가 적어지는 N개의 토픽을 최적의 토픽으로 사용하였습니다.

2. 각 토픽 별로 주로 사용되는 단어들을 출력해서 보여줍니다.

1. 여러 K값을 사용하여 모델을 훈련시킨 결과 Perplexity 점수의 변화가 적어지는 90개의 토픽을 최적의 토픽으로 사용하였습니다.
2. 각 토픽 별로 주로 사용되는 단어들을 출력해서 보여줍니다.

```
!pip install tomotopy #tomotopy 설치
```

```
Requirement already satisfied: tomotopy in /usr/local/lib/python3.7/dist-packages (0.12.2)
Requirement already satisfied: numpy>=1.11.0 in /usr/local/lib/python3.7/dist-packages (from
```

```
mdl = tp.LDAModel(k=20, seed=830731) # 모델 설정
```

```
#문서 추가
for n, line in enumerate(data['tokens2']):
    line = ','.join(line)
    mdl.add_doc(line.strip().split(","))
```

```
#모델훈련
for i in range(0, 1000, 10):
    mdl.train(10)
    print('반복: {}Wt로그-우도: {}'.format(i, mdl.ll_per_word))
```

```
반복: 0 로그-우도: -8.54889350951179
반복: 10        로그-우도: -7.987309448350119
반복: 20        로그-우도: -7.777483596400036
반복: 30        로그-우도: -7.680062213463918
반복: 40        로그-우도: -7.629981307489837
반복: 50        로그-우도: -7.598209277957059
반복: 60        로그-우도: -7.574603285264753
반복: 70        로그-우도: -7.556271679892635
반복: 80        로그-우도: -7.541654530708011
반복: 90        로그-우도: -7.529752323418782
반복: 100       로그-우도: -7.523530423602289
반복: 110       로그-우도: -7.516767611790708
반복: 120       로그-우도: -7.51031604892799
반복: 130       로그-우도: -7.505330457865288
반복: 140       로그-우도: -7.502104940126264
반복: 150       로그-우도: -7.500646439422149
반복: 160       로그-우도: -7.498077815366474
반복: 170       로그-우도: -7.492187053554254
반복: 180       로그-우도: -7.492924940599143
반복: 190       로그-우도: -7.488541470604421
반복: 200       로그-우도: -7.489388046036014
반복: 210       로그-우도: -7.4876445880056
반복: 220       로그-우도: -7.484899647578293
반복: 230       로그-우도: -7.481878857625434
반복: 240       로그-우도: -7.47996325886934
반복: 250       로그-우도: -7.480636663995688
반복: 260       로그-우도: -7.476474984351382
```

```
반복: 270       로그-우도: -7.475481428009353
반복: 280       로그-우도: -7.476527487036563
반복: 290       로그-우도: -7.473070627236251
반복: 300       로그-우도: -7.473549901420744
반복: 310       로그-우도: -7.4687958900012745
반복: 320       로그-우도: -7.468239917446325
반복: 330       로그-우도: -7.4704307729672985
반복: 340       로그-우도: -7.468895431729533
반복: 350       로그-우도: -7.471068240924282
반복: 360       로그-우도: -7.469709438637298
반복: 370       로그-우도: -7.467356704119273
반복: 380       로그-우도: -7.467986258532078
반복: 390       로그-우도: -7.4684293648798885
반복: 400       로그-우도: -7.4685287098574555
반복: 410       로그-우도: -7.467495127351297
반복: 420       로그-우도: -7.465357749248856
반복: 430       로그-우도: -7.465224034067169
반복: 440       로그-우도: -7.4664119176803885
반복: 450       로그-우도: -7.468889346706713
반복: 460       로그-우도: -7.46715384608807
반복: 470       로그-우도: -7.467077678437921
반복: 480       로그-우도: -7.466990586366001
반복: 490       로그-우도: -7.467612633545273
반복: 500       로그-우도: -7.46729034816414
반복: 510       로그-우도: -7.465524943919049
반복: 520       로그-우도: -7.466366652104593
반복: 530       로그-우도: -7.46599758763453
반복: 540       로그-우도: -7.465942906197228
반복: 550       로그-우도: -7.464650844590142
반복: 560       로그-우도: -7.464123870541329
```

```
1 #토픽별 정보 출력
2 for k in range(mdl.k):
3     print('토픽 #{}에 가장 많이 등장한 단어들'.format(k))
4     print(mdl.get_topic_words(k, top_n=5))
```

```
토픽 #0에 가장 많이 등장한 단어들
[('중국', 0.07683780789375305), ('올림픽', 0.04872541129589081), ('북한', 0.04209649935364723
토픽 #1에 가장 많이 등장한 단어들
[('북한', 0.08051592856645584), ('미사일', 0.08006417751312256), ('발사', 0.04746874794363975
토픽 #2에 가장 많이 등장한 단어들
[('미국', 0.05777746066451073), ('행정부', 0.027094271034002304), ('현지', 0.026046087965369
토픽 #3에 가장 많이 등장한 단어들
[('대통령', 0.1139090433716774), ('정상회담', 0.05163053423166275), ('미국', 0.04776536300778
토픽 #4에 가장 많이 등장한 단어들
[('북한', 0.07931581884622574), ('김정은', 0.0588354133069515 2), ('노동당', 0.039148163050413
토픽 #5에 가장 많이 등장한 단어들
[('대표', 0.054522208869457245), ('대북', 0.037091631442308426), ('특별', 0.03703293949365616
토픽 #6에 가장 많이 등장한 단어들
[('지원', 0.06073800474405289), ('북한', 0.03673570975661278), ('코로나', 0.03424766659736633
토픽 #7에 가장 많이 등장한 단어들
[('북한', 0.06520427018404007), ('미국', 0.04014425724744797), ('정부', 0.018184006214141846)
토픽 #8에 가장 많이 등장한 단어들
[('사건', 0.021328965201973915), ('국가', 0.013728094287216663), ('국정원', 0.013728094287216
토픽 #9에 가장 많이 등장한 단어들
[('전단', 0.060931041836738586), ('대북', 0.05876899138092995), ('살포', 0.03680450841784477)
토픽 #10에 가장 많이 등장한 단어들
[('일본', 0.08284001052379608), ('총리', 0.031488388776779175), ('한국', 0.025609450414776802
토픽 #11에 가장 많이 등장한 단어들
```

```
[('장관', 0.046540990471839905), ('평화', 0.02254130132496357), ('통일부', 0.018651336431503
토픽 #12에 가장 많이 등장한 단어들
[('북한', 0.0690151229500706), ('제재', 0.06563601642847061), ('대북', 0.04109019413590431),
토픽 #13에 가장 많이 등장한 단어들
[('미국', 0.05175696313381195), ('폼페이', 0.045182015746831894), ('북한', 0.0368200354278087
토픽 #14에 가장 많이 등장한 단어들
[('훈련', 0.040765922516584396), ('연합', 0.03673914447426796), ('국방부', 0.022894268855452
토픽 #15에 가장 많이 등장한 단어들
[('의원', 0.03680307790637016), ('국회', 0.027394825592637062), ('대표', 0.02620283141732216)
토픽 #16에 가장 많이 등장한 단어들
[('북한', 0.0380081161856513), ('비핵화', 0.03352897986769676), ('미국', 0.02350088953971862
토픽 #17에 가장 많이 등장한 단어들
[('청와대', 0.08525597304105759), ('대통령', 0.05326684191823006), ('국가', 0.042372256517410
토픽 #18에 가장 많이 등장한 단어들
[('남북', 0.0826067253947258), ('공동', 0.023786718025803566), ('북한', 0.020791389048099518)
토픽 #19에 가장 많이 등장한 단어들
[('경제', 0.02206207625567913), ('북한', 0.01655864156782627), ('사업', 0.012299460358917713)
```

```
1 #토픽 모델 정보 출력
2 mdl.summary()
```

```
<Basic Info>
| LDAModel (current version: 0.12.2)
| 20000 docs, 661673 words
| Total Vocabs: 20604, Used Vocabs: 20604
| Entropy of words: 7.42300
| Entropy of term-weighted words: 7.42300
| Removed Vocabs: <NA>
|
<Training Info>
| Iterations: 1000, Burn-in steps: 0
| Optimization Interval: 10
| Log-likelihood per word: -7.45952
|
<Initial Parameters>
| tw: TermWeight.ONE
| min_cf: 0 (minimum collection frequency of words)
| min_df: 0 (minimum document frequency of words)
| rm_top: 0 (the number of top words to be removed)
| k: 20 (the number of topics between 1 ~ 32767)
| alpha: [0.1] (hyperparameter of Dirichlet distribution for document-topic, given as a si
| eta: 0.01 (hyperparameter of Dirichlet distribution for topic-word)
| seed: 830731 (random seed)
| trained in version 0.12.2
|
<Parameters>
| alpha (Dirichlet prior on the per-document topic distributions)
|  [0.04409008 0.05394424 0.06282306 0.12526847 0.07692941 0.05876188
|   0.03866345 0.13987422 0.04307252 0.03025907 0.05048582 0.06878018
|   0.06101773 0.05428463 0.04510652 0.0838603  0.21503149 0.05769943
|   0.06211988 0.04557203]
| eta (Dirichlet prior on the per-topic word distribution)
|  0.01
|
<Topics>
| #0 (20461) : 중국 올림픽 북한 평창 시진핑
| #1 (28571) : 북한 미사일 발사 탄도 도발
```

```
| #2 (31277) : 미국 행정부 현지 조 바이든 정책
| #3 (52573) : 대통령 정상회담 미국 도널드 트럼프 김정은
| #4 (35096) : 북한 김정은 노동당 위원장 국무위원
| #5 (33872) : 대표 대북 특별 협상 미국
| #6 (20292) : 지원 북한 코로나 대북 정부
| #7 (45604) : 북한 미국 정부 한국 보도
| #8 (21502) : 사건 국가 국정원 검찰 혐의
| #9 (20145) : 전단 대북 살포 인권 북한
| #10 (22247) : 일본 총리 한국 중국 외교
| #11 (28586) : 장관 평화 통일부 통일 한반도
| #12 (36786) : 북한 제재 대북 유엔 중국
| #13 (29452) : 미국 폼페이 북한 국무장관 현지
| #14 (21896) : 훈련 연합 국방부 한미 군사
| #15 (46774) : 의원 국회 대표 대통령 한국
| #16 (84855) : 북한 비핵화 미국 한반도 대화
| #17 (27147) : 청와대 대통령 국가 안보 특사
| #18 (33847) : 남북 공동 북한 판문점 연락
| #19 (20690) : 경제 북한 사업 남북 회장
|
```

```
#훈련된 모델의 Perplexity 값 출력
mdl.perplexity
```

```
1736.3183733708318
```

```
#훈련된 모델의 Coherence 값 출력
coh = tp.coherence.Coherence(mdl, coherence='u_mass') # 유사도를 구하는 여러 방법 중  `U-mass`에
average_coherence = coh.get_score()
coherence_per_topic = [coh.get_score(topic_id=n) for n in range(mdl.k)]

print(average_coherence)
print(coherence_per_topic)
```

```
-2.1015360027825287
[-2.7776348519525746, -2.110751597556496, -1.955887570176035, -1.12343014480128, -2.402191560
```

```
# LDA Model 설정

#K 값을 찾기 위해 LDA 모델을 반복적으로 수행해야함으로,
#분석의 용이를 위해서 Tomotopy를 이용한 모델링을 반복하는 사용자 지정함수 만들기
def lda(k_model, iteration, text, word_remove=0):

    # 모델 설정
    model = tp.LDAModel(k=k_model, rm_top=word_remove, seed=871017)

    # 문서 추가
    for n, line in enumerate(text):
        line = ','.join(line)
        model.add_doc(line.strip().split(','))

    model.burn_in = 100
    model.train(0)
```

```
    print('문서 개수:', len(model.docs), ', 단어 개수:', len(model.used_vocabs), ', 단어의 총수:
    print('제거된 단어들:', model.removed_top_words)

    print('훈련 중...',flush=True)
    for i in range(0, iteration, 10):
        model.train(10)
        print('반복: {}Wt로그-우도: {}'.format(i, model.ll_per_word))

    model.summary()
    return(model)
```

```
#k값 찾기 준비
k_list = []
k_list.extend(list(range(2,11, 1)))
k_list.extend(list(range(15,41, 5)))
k_list.extend(list(range(50,100, 10)))
k_list.extend(list(range(100,151, 25)))

perplexity_scores = {} # A dictionary for saving perplexity score.
df_coh = pd.DataFrame()

print(k_list)
```

```
[2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 35, 40, 50, 60, 70, 80, 90, 100, 125, 150]
```

```
# 주어진 K 모델 별 Perplexity  숫자 구하기

for k in k_list:
    d = {}
    #모델 학습
    model = lda(k, 1000, text=data['tokens2'])

    # perplexity score
    perplexity_scores[k]=model.perplexity # 모델의 혼란도(perplexity)값
    perplexity_df = pd.DataFrame.from_dict(perplexity_scores, orient='index', columns=['model_pe
    perplexity_df['k']=perplexity_df.index # 인덱스 저장

    # coherenece score
    coh = tp.coherence.Coherence(model, coherence='u_mass') # 유사도를 구하는 여러 방법 중  `U-m
    average_coherence = coh.get_score()
    coherence_per_topic = [coh.get_score(topic_id=n) for n in range(model.k)]

    d = {'k':[k]*k, 'avg_coherence':[average_coherence]*k, 'coherence':pd.Series(coherence_per_t

    df_coh = df_coh.append(pd.DataFrame(d))
```

```
문서 개수: 20000 , 단어 개수: 20604 , 단어의 총수: 661673
제거된 단어들: []
훈련 중...
반복: 0 로그-우도: -7.863077053044223
반복: 10        로그-우도: -7.693604816105488
반복: 20        로그-우도: -7.6368888262740695
```

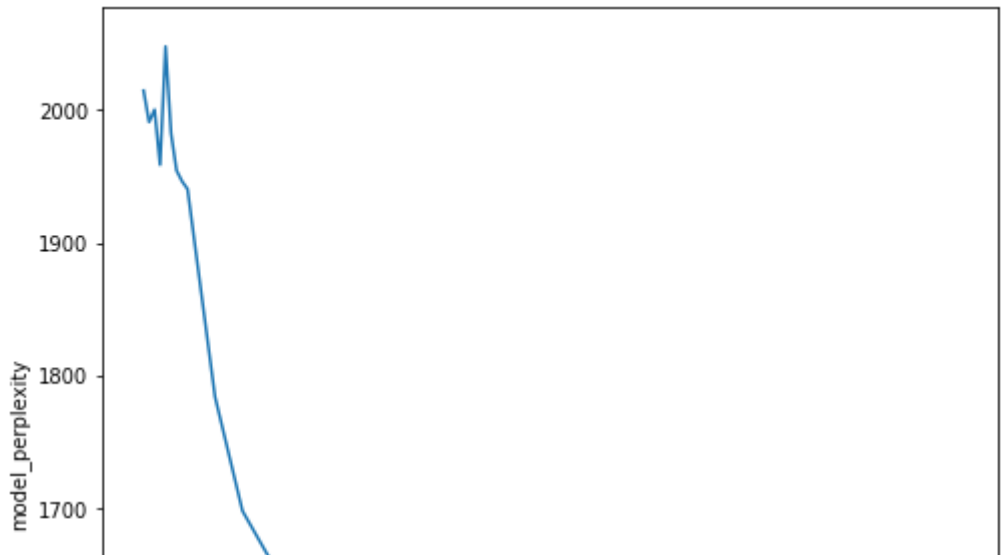
```
반복: 30        로그-우도: -7.618846600482606
반복: 40        로그-우도: -7.608161207695724
반복: 50        로그-우도: -7.603674765589291
반복: 60        로그-우도: -7.599500000411168
반복: 70        로그-우도: -7.598519779713242
반복: 80        로그-우도: -7.596198363048796
반복: 90        로그-우도: -7.594826636502836
반복: 100       로그-우도: -7.591429523753624
반복: 110       로그-우도: -7.602128382505616
반복: 120       로그-우도: -7.606351494441588
반복: 130       로그-우도: -7.60926864113475
반복: 140       로그-우도: -7.611261902701148
반복: 150       로그-우도: -7.61201685023949
반복: 160       로그-우도: -7.61218099185592
반복: 170       로그-우도: -7.610732001324508
반복: 180       로그-우도: -7.610548432736875
반복: 190       로그-우도: -7.6105410336332096
반복: 200       로그-우도: -7.611245140383068
반복: 210       로그-우도: -7.611558519615268
반복: 220       로그-우도: -7.610735113137681
반복: 230       로그-우도: -7.611334766187701
반복: 240       로그-우도: -7.611025446312884
반복: 250       로그-우도: -7.609240923418993
반복: 260       로그-우도: -7.610684206150705
반복: 270       로그-우도: -7.6097539507320295
반복: 280       로그-우도: -7.610978435764323
반복: 290       로그-우도: -7.609024316132098
반복: 300       로그-우도: -7.609970984788345
반복: 310       로그-우도: -7.609783228010294
반복: 320       로그-우도: -7.609022527011855
반복: 330       로그-우도: -7.609987713741326
반복: 340       로그-우도: -7.609360987790665
반복: 350       로그-우도: -7.609233527409003
반복: 360       로그-우도: -7.608047125381119
반복: 370       로그-우도: -7.608056175714905
반복: 380       로그-우도: -7.608618538523067
반복: 390       로그-우도: -7.608963980268293
반복: 400       로그-우도: -7.609449469260984
반복: 410       로그-우도: -7.6096368556328935
반복: 420       로그-우도: -7.610560014993815
반복: 430       로그-우도: -7.610548985242096
반복: 440       로그-우도: -7.609456369906257
반복: 450       로그-우도: -7.6089887730812675
반복: 460       로그-우도: -7.607453745734126
반복: 470       로그-우도: -7.607815793391417
반복: 480       로그-우도: -7.609372718126149
반복: 490       로그-우도: -7.608103447194865
반복: 500       로그-우도: -7.608850421411517
반복: 510       로그-우도: -7.609434187921674
반복: 520       로그-우도: -7.6092336710933175
반복: 530       로그-우도: -7.60708684170311
```

```
1 perplexity_df
```

	model_perplexity	k
2	2014.418051	2
3	1990.833779	3
4	2000.137763	4
5	1958.602909	5
6	2047.646459	6
7	1982.318913	7
8	1954.334934	8
9	1946.242180	9
10	1940.354546	10
15	1784.169069	15
20	1698.237196	20
25	1662.020136	25
30	1631.338698	30
35	1590.028491	35
40	1581.906288	40
50	1527.459050	50
60	1493.941835	60
70	1490.150469	70
80	1475.441072	80
90	1462.908722	90
100	1455.926735	100

```
# Plotting perplexity Score : Perplexity 값 시각화
fig, ((ax1))= plt.subplots(nrows=1, ncols=1)
# 전체 표들을 표시할 캔버스의 크기를 설정
fig.set_size_inches(8,8)

sns.lineplot(x="k", y="model_perplexity", data=perplexity_df, ax=ax1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e55305cd0>
```




```
# 최적화 모델 출력
k = 90 #그래프가 평탄해지기 시작하는 부분
mdl_k=lda(k, 1000, text=data['tokens2'])
```

```
문서 개수: 20000 , 단어 개수: 20604 , 단어의 총수: 661673
제거된 단어들: []
훈련 중...
반복: 0 로그-우도: -8.998727061883113
반복: 10        로그-우도: -8.23396076121107
반복: 20        로그-우도: -8.001923519541311
반복: 30        로그-우도: -7.893648969916188
반복: 40        로그-우도: -7.838704974933296
반복: 50        로그-우도: -7.802461701432384
반복: 60        로그-우도: -7.773535426924214
반복: 70        로그-우도: -7.7518711281787205
반복: 80        로그-우도: -7.739387829177775
반복: 90        로그-우도: -7.73010386096334
반복: 100       로그-우도: -7.685080000275698
반복: 110       로그-우도: -7.547338980681537
반복: 120       로그-우도: -7.486262396719594
반복: 130       로그-우도: -7.4519457684331405
반복: 140       로그-우도: -7.430497233045304
반복: 150       로그-우도: -7.414115281151076
반복: 160       로그-우도: -7.4035914530552995
반복: 170       로그-우도: -7.392329302411429
반복: 180       로그-우도: -7.383616254020388
반복: 190       로그-우도: -7.378533860554488
반복: 200       로그-우도: -7.369739601267377
반복: 210       로그-우도: -7.363902200678174
반복: 220       로그-우도: -7.356689129331016
반복: 230       로그-우도: -7.3526296804558715
반복: 240       로그-우도: -7.349752765630979
반복: 250       로그-우도: -7.346264864908892
반복: 260       로그-우도: -7.343250949895585
반복: 270       로그-우도: -7.3405535484624025
반복: 280       로그-우도: -7.339410070968803
반복: 290       로그-우도: -7.3351653360930875
반복: 300       로그-우도: -7.333318617158594
반복: 310       로그-우도: -7.332776843625361
반복: 320       로그-우도: -7.327846616123685
반복: 330       로그-우도: -7.323742287508264
```

```
반복: 340       로그-우도: -7.324539834818664
반복: 350       로그-우도: -7.326726753670681
반복: 360       로그-우도: -7.322781124736399
반복: 370       로그-우도: -7.322182209372375
반복: 380       로그-우도: -7.323945430052323
반복: 390       로그-우도: -7.320655649781862
반복: 400       로그-우도: -7.321083745538957
반복: 410       로그-우도: -7.317665404646851
반복: 420       로그-우도: -7.315678290915141
반복: 430       로그-우도: -7.3125108827747365
반복: 440       로그-우도: -7.3105670462878
반복: 450       로그-우도: -7.311047747174391
반복: 460       로그-우도: -7.311071423852416
반복: 470       로그-우도: -7.3098952368260575
반복: 480       로그-우도: -7.30684634598073
반복: 490       로그-우도: -7.304477027779521
반복: 500       로그-우도: -7.3066956847140645
반복: 510       로그-우도: -7.304358984714547
반복: 520       로그-우도: -7.305838683743246
```

```
topic_df = pd.DataFrame()
for i in range(0, k):
  temp = pd.DataFrame()
  temp = pd.DataFrame(mdl_k.get_topic_words(i, top_n=10))
  temp.columns = ["Topic"+str(i+1),"probs"+str(i+1)]
  temp = temp.reset_index()
  if (i==0):
      topic_df = topic_df.append(temp, ignore_index=True)
  else:
      topic_df = topic_df.merge(temp, left_on="index", right_on="index")
```

```
topic_df
```

```
topic_df.to_csv("Top_Frequent Words.txt", sep="Wt", encoding="UTF-8", index=False)
```

```
#각 문서 별 토픽 확률 분석
new_df = pd.DataFrame()

for i, line in enumerate(mdl_k.docs):
    temp=pd.DataFrame(line.get_topic_dist()).T
    new_df=new_df.append(temp)

new_df.shape
new_df = new_df.reset_index()
new_df = new_df.drop(['index'], axis=1)
```

```
new_df
```

```
# Assigning Column Names
new_df.columns = ['Topic'+ str(x) for x in range(1,k+1)]
```

```
new_df[new_df.columns]
```

```
data = data.reset_index().drop(['index'], axis=1)
```

```
#가장 확률이 높은 토픽 선택
theta_df = data.merge(new_df, left_index=True, right_index=True)
theta_df
```

	id	date	news_org	upload	title	sec1	sec2	se
0	1.100401e+06	2021-11-26	동아일보	이윤태	李 "北 남북 합의 일방 파기엔 할 말 하겠다"	정치>북한	정치>선거	정>오
1	1.100401e+06	2021-11-25	동아일보	이윤태	이재명 "北 남북 합의 일방 적 위반 파기엔 할 말 하겠다"	정치>북한	정치>선거	정>오
2	1.100101e+06	2021-11-25	경향신문	곽희양 기자 huiyang@kyunghyang.com	이재명 "바이든 김정은 과 직접 만나 북핵 문제 풀겠다"	정치>북한	정치>외교	정>오
3	1.101001e+06	2021-11-25	한겨레	심우삼 기자	이재명 "북핵, 바이든-김정은 만나서 풀겠다"	정치>북한	정치>외교	국>

						쓰니			
4	1.100901e+06	2021-11-25	중앙일보	오현석 (oh.hyunseok1@joongang.co.kr)	이재명 "대일 강경? 그건 오해 난 한마디로 실용외교주의자"	정치>선거	정치>북한	정치>외
...	...	...	...	...	...	...	...	
19995	1.100401e+06	2017-09-25	동아일보	손효주	B-1B에 조기경보기 '실전력' 총출동 美, 北 단독타격 경고	정치>외교	정치>북한	N
19996	1.100401e+06	2017-09-25	동아일보	장원재	이낙연 총리 "대북 지원, 핵무장 돕는다는 건 과장된 견해"	정치>북한	정치>외교	국>

```
1 theta_df['Highest_Topic']=theta_df[['Topic'+ str(s) for s in range(1, k+1)]].idxmax(axis=1)
```

```
1 theta_df.head(1)
```

	id	date	news_org	upload	title	sec1	sec2	sec3	cat1	cat2	cat3	peopl
0	1.100401e+06	2021-11-26	동아일보	이윤태	李 "北 남북 합의 일방 파기엔 할 말 하겠다"	정치>북한	정치>선거	정치>청와대	NaN	NaN	NaN	정부 문재 인,이 조 바 이든 김정 은,더 재명

1 rows × 113 columns

```
1 theta_df['Highest_Topic'].value_counts()
```

```
Topic71    766
Topic82    745
Topic8     579
Topic2     504
Topic51    493
          ...
Topic86     70
Topic47     69
Topic80     57
Topic84     47
Topic43     44
Name: Highest_Topic, Length: 90, dtype: int64
```

토픽 이름 붙이기 (2점)

학습된 모델의 결과를 사용하여 생성된 토픽의 이름을 붙이는 단계입니다.

참조: 문서 당 가장 확률이 높은 토픽을 지정하는 방법을 참조하시면 됩니다. 만약 토픽이 10개보다 많다면, 토픽의 비율이 가장 많은 10개의 토픽을 골라 각 토픽의 이름을 구해주시면 됩니다. 문서와 토픽에 주로 등장하는 단어들을 보기 위해서는 토픽 모델링 수업을 참조해주세요.

예) 코로나 보도에 관한 토픽 중 가장 비율이 많은 총 10개의 토픽을 선택했습니다. 매 토픽마다 토픽에 가장 잘 등장하는 단어들과 실제 문서들을 참조하여 본 결과 해당 토픽은 000에 대한 것으로 파악됩니다.

대북 정책 보도에 관한 토픽 중 가장 비율이 많은 총 41개의 토픽을 선택했습니다. 매 토픽마다 토픽에 가장 잘 등장하는 단어들과 실제 문서들을 참조하여 본 결과 해당 토픽은 '김정은'에 대한 것으로 파악됩니다.

```
i = 41

cnt = len(theta_df[theta_df['Highest_Topic']=='Topic12'])

print("토픽 {:8}: 토픽 개수 {:8}, 전체 퍼센트 {:8.3f}%".format(i, cnt, cnt/len(data)*100)) # 해당

print(topic_df['Topic'+str(i)]) # 해당 토픽에서 가장 많이 나오는 단어

theta_df[theta_df['Highest_Topic']=='Topic'+str(i)]['maintext'].sample(n=15).values # 임의로 선택
```

```
토픽       41: 토픽 개수      159, 전체 퍼센트    0.795%
0       장관
1      통일부
2      후보자
3      이인영
4       연철
5       이날
6    인사청문회
7       국회
8      조명균
9       취임
Name: Topic41, dtype: object
array(['23일 열린 이재명인영 통일부 장관 후보자에 대한 국회 인사청문회에서 “이재명 후보자는
       '문재인 대통령이재명 10일 취임 4주년 특별연설에서 “부동산 문제만큼은 정부가 할 말이재
       '북한이재명 개성공단 내 남북공동연락사무소를 폭파한 지 24시간 만에 김연철 통일부 장관(
       '23일 인사청문회를 앞둔 이재명인영 통일부 장관 후보자가 장관에 취임할 경우 “(남북 간(
       '김연철 통일부 장관이재명 접점을 찾지 못하고 있는 북-미 비핵화 협상에 대해 ‘잠정 합의
       '문재인 대통령이재명 8일 이재명인영 통일부 장관 후보자와 박지원 국가정보원장 후보자에
       '“(재개하도록) 노력해야겠죠.” ₩n ₩n 김연철 통일부 장관 후보자는 8일 개각 발표 후 서울
       '이재명인영 통일부 장관이재명 29일 마무리 단계에 있는 바이재명든 정부의 대북 정책과 관
       '・취임 3일 만에 방역물품 대북 반출 승인 등 남북관계 복원 행보₩n₩n₩n₩n이재명인영 통일부
       '나경원 자유한국당 원내대표는 10일, 김연철 통일부 장관 후보자에 대해 “매우 부적절한 인
       '이재명인영 통일부 장관 후보자는 23일 “제가 특사가 돼 평양을 방문하는 것이재명 경색된
       '김연철 통일부 장관 후보자의 과거 논문이재명 도마 위에 오르고 있다. 김 후보자는 각종 인
       '이재명인영 통일부 장관과 도미타 고지(富田浩司) 주한 일본 대사가 1일 북한 문제와 관련하
       '이재명인영 통일부 장관은 31일 “인도적 협력은 제재의 영역과는 분명히 구분되어야 한다”
       '문재인 대통령은 5일 국가인권위원회 위원장 후보자에 송두환 법무법인 한결 대표변호사, 김
      dtype=object)
```

토픽 시각화 (1.5점)

토픽 모델리의 결과를 시각화를 통해 보여줍니다.

참조: 앞서 선택한 총 10개의 토픽을 사용하여 토픽 모델링을 수행하여 해결하고자 하는 문제에 대해 시각화 해주세요. 해결하고자 하는 문제에 따라 최적의 시각화 방법이 다를 수 있습니다. 예를 들어, 시기의 변화의 경우 토픽의 시기 변화를, 언론사별 변화의 경우 언론사에 따라 토픽 분포가 어떻게 차이나는 지를 보여줘야 합니다.

시각화의 예:) 언론사의 정파성에 따라 코로나 관련 보도의 유형이 달라지는 지를 살펴보기 위해서, 진보(한겨레/경향)과 보수(조선/동아)의 토픽 별 보도량을 시각화하여 보여주었습니다.

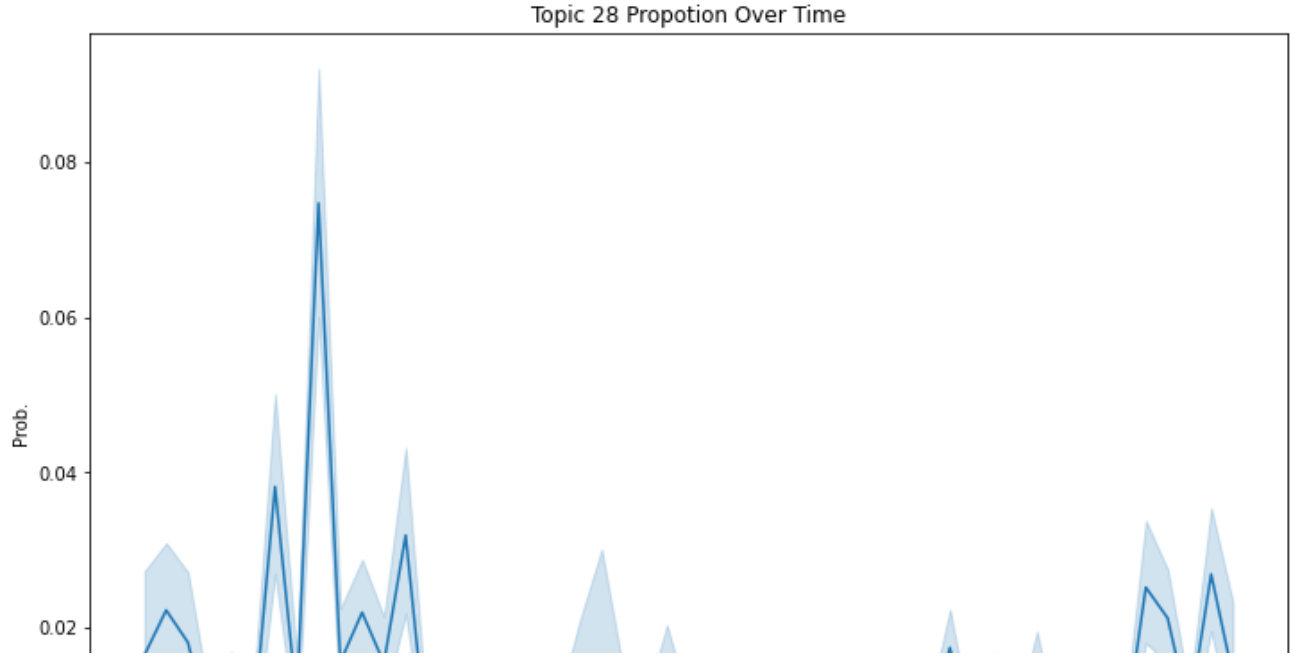
언론사의 정파성에 따라 대북 정책 관련 보도의 유형이 달라지는 지를 살펴보기 위해서, 진보(한겨레/경향)과 보수(중앙/동아)의 토픽 별 보도량을 시각화하여 보여주었습니다.

```
# 코드로 형식 지정됨
```

```
#시기별 각 토픽 분포 확률 시각화
raw_data = theta_df.groupby('year-month').count()['date']
raw_data = raw_data.reset_index() # year-month 를 컬럼으로 변경
```

```
plt.figure(figsize=(12,8)) # 이미지 사이즈 정하기 (가로, 세로)

i = random.randint(1,71)
topic = 'Topic'+str(i)

ax = sns.lineplot(x='year-month',y=theta_df[topic], data=theta_df) # 원 데이터를 사용하여 데이터


# x축의 값 변경위해서 raw_data사용

freq = 10

ax.set_xticklabels(raw_data.iloc[::freq]["year-month"]) # x 축에 표시할 레이블 선택
xtix = ax.get_xticks()
ax.set_xticks(xtix[::freq]) # x 축에 레이블을 표시한 틱을 선택

# 데이터 정보 표시
ax.set(title='Topic {} Propotion Over Time'.format(i), xlabel="Year-Month", ylabel="Prob.")

print(topic_df[topic])
```

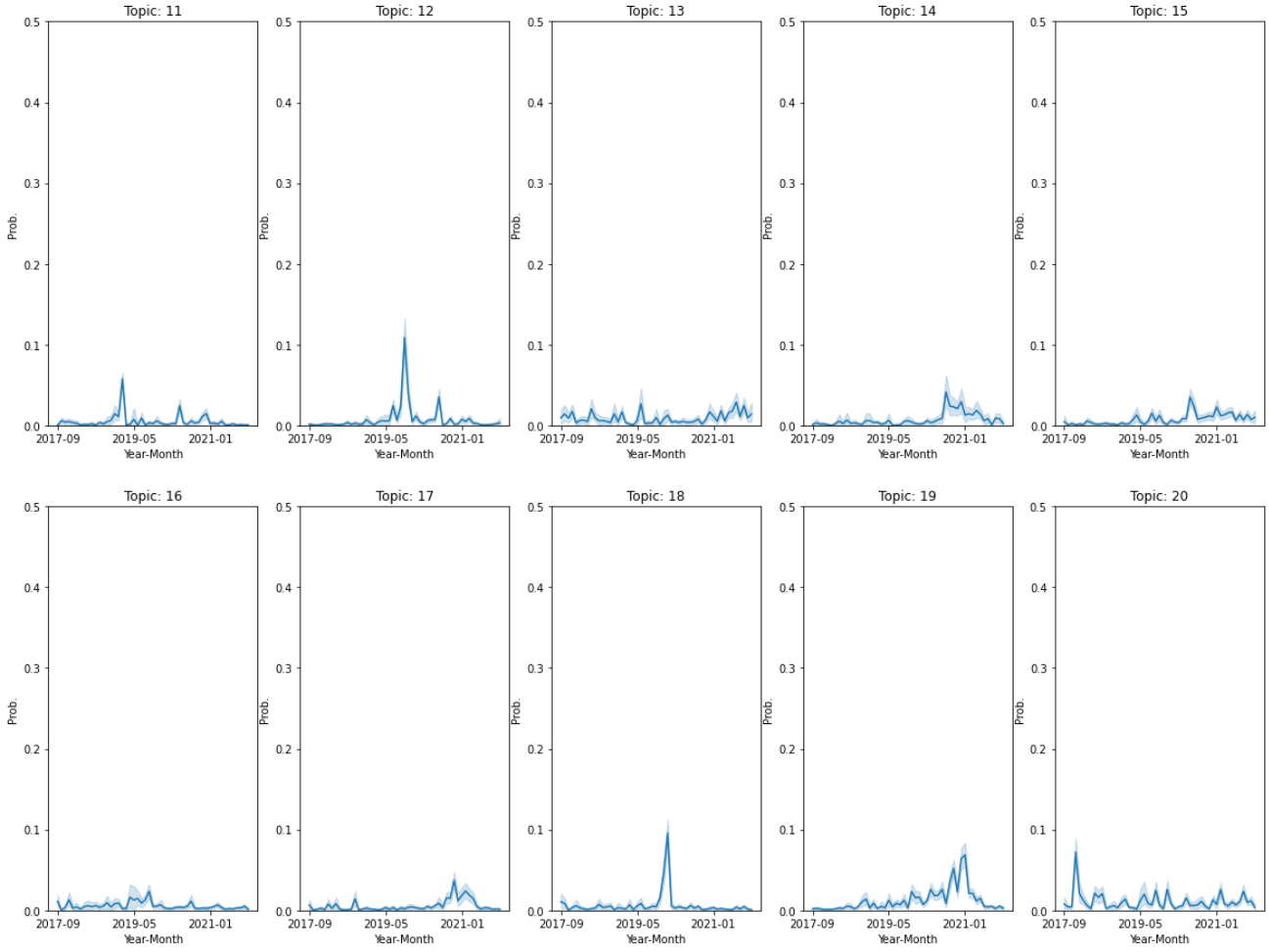
```
0      장관
1     외교부
2      외교
3      강경
4    국방장관
5      링컨
6      미국
7      회의
8     국방부
9      티스
Name: Topic28, dtype: object
```




```
#여러 토픽들 함께 출력하기
figure, axs = plt.subplots(ncols=5, nrows=2) # 총 열개의 이미지 표시
figure.set_size_inches(20,15)

base=10 # 총 10개씩의 토픽을 출력할 것임 십의자리를 입력
topic = ['Topic'+str(base+t+1) for t in range(0, 11)]


for k, ax in enumerate(['ax'+str(x) for x in range(1, 11)]):
  t = topic[k]
  ax = sns.lineplot(x='year-month',y=theta_df[t], data=theta_df, ax=axs[k//5, k%5])
  freq = 20

  ax.set_xticklabels(raw_data.iloc[::freq]["year-month"]) # x 축에 표시할 레이블 선택
  xtix = ax.get_xticks()
  ax.set_xticks(xtix[::freq]) # x 축에 레이블을 표시한 틱을 선택
  ax.set(ylim=(0, .5))

  # 데이터 정보 표시
  ax.set(title='Topic: {}'.format(base+k+1), xlabel="Year-Month", ylabel="Prob.")

print(topic_df[topic])
```

	Topic11	Topic12	Topic13	Topic14	...	Topic18	Topic19	Topic20	Topic21
0	연락	협상	혐의	조사	...	일본	폼페이	훈련	사람
1	사무소	실무	검찰	대통령	...	한국	마이크	연합	이야기
2	남북	대사	수사	여론	...	수출	국무장관	한미	당시
3	공동	미국	대통령	결과	...	정부	미국	군사훈련	요즘
4	북한	스웨덴	국가	평가	...	규제	장관	중단	얘기
5	폭파	북한	의혹	수행	...	조치	북한	실시	지금
6	개성	외무성	서울	포인트	...	경제	현지	연기	영화
7	개성공단	해리스	국정원	지지율	...	공사	시간	연습	생각
8	관계	부상	정보원	문재인	...	징용	방북	전환	인터뷰
9	설치	선희	중앙	국정	...	보복	비핵화	예정	정도

[10 rows x 11 columns]



```
#언론사 별 토픽 분포
theta_df.groupby('news_org')['Topic12'].agg(['mean','std','count'])
```

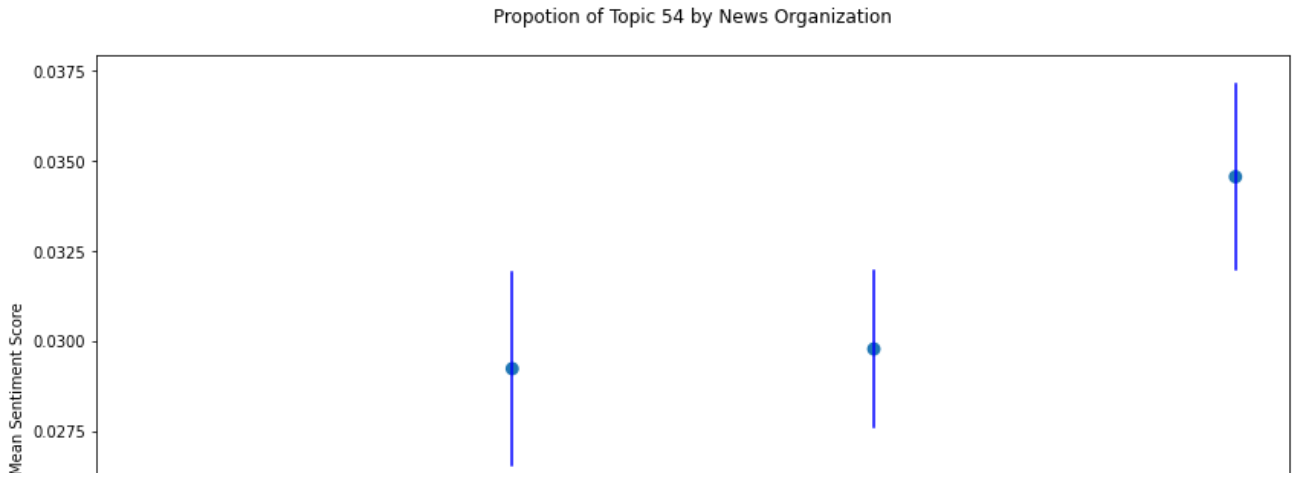
news_org	mean	std	count
경향신문	0.007544	0.047353	4230
동아일보	0.007371	0.050862	4518
중앙일보	0.007611	0.050622	8090
한겨레	0.005143	0.039019	3162

```
k = random.randint(1, 70)

raw2_df = theta_df.groupby('news_org')['Topic'+str(k)].agg(['count','mean','std']).sort_values(b
raw2_df['count_sq'] = raw2_df['count'].apply(lambda x:sqrt(x)) # 제곱근
raw2_df['ll'] = raw2_df['mean']-(1.96*(raw2_df['std']/raw2_df['count_sq']))
raw2_df['ul'] = raw2_df['mean']+(1.96*(raw2_df['std']/raw2_df['count_sq']))
raw2_df['num_row'] = np.arange(raw2_df.shape[0])
```

```
plt.figure(figsize=(14,8)) # 이미지 사이즈 정하기 (가로, 세로)

ax1 = sns.pointplot(x='num_row', y="mean", data=raw2_df, join=False)
ax1.set_xticklabels(raw2_df.reset_index()['news_org'])

for i, news in enumerate(raw2_df.reset_index()['news_org'].unique()):
  ll = raw2_df.iloc[i][4]
  ul = raw2_df.iloc[i][5]
  plt.plot(np.asarray([[i, i]]).T, np.asarray([[ll, ul]]).T, "b")

ax1.set(title='₩nPropotion of Topic {} by News Organization₩n'.format(k), xlabel="₩n₩nNews Organ

print(topic_df["Topic"+str(k)])
```

```
0      북한
1      대화
2      미국
3      협상
4     비핵화
5      재개
6      대북
7     메시지
8      정부
9      제안
Name: Topic54, dtype: object
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:214: RuntimeWarninc
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:214: RuntimeWarninc
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:214: RuntimeWarninc
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:214: RuntimeWarninc
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:214: RuntimeWarninc
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:214: RuntimeWarninc
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:214: RuntimeWarninc
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:214: RuntimeWarninc
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:214: RuntimeWarninc
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:214: RuntimeWarninc
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:214: RuntimeWarninc
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:214: RuntimeWarninc
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:214: RuntimeWarninc
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:183: RuntimeWarninc
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:183: RuntimeWarninc
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:183: RuntimeWarninc
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:183: RuntimeWarninc
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:183: RuntimeWarninc
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:183: RuntimeWarninc
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:183: RuntimeWarninc
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:183: RuntimeWarninc
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:183: RuntimeWarninc
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:183: RuntimeWarninc
  font.set_text(s, 0, flags=flags)
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:183: RuntimeWarning
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:183: RuntimeWarning
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:183: RuntimeWarning
  font.set_text(s, 0, flags=flags)
```



분석 결과 논의 (1점)

토픽 모델링 분석 결과를 통해 도출된 결론을 논의해보세요

참조: 분석 결과를 간단히 요약하고, 이를 통해 알게 된 사실에 대해서 논의해주세요. 문제 제기 과성에서 예측한 결과와 분석 결과가 일치했는지의 여부를 설명해주세요. 만약 분석 결과가 일치하지 않았다면 일치하지 않은 이유를 생각해보세요.

처음 분석을 시작했을 때는 위에서 언급한 것과 같이 보수 성향의 언론사 (중앙/동아), 진보 성향의 언론사 (한겨레/경향)에서 각자의 정치 성향과 유사한 양상의 데이터 분석 결과가 나올 것으로 예측하였습니다. 그리고 분석 이후 결과를 예측과 비교해보니 그와 일치하는 결과가 관측되었습니다.

- 교수님 제가 처음 실행했을 때는 마지막 시각화 단계에서 언론사 이름들이 한국어로 잘 출력이 되었는데 마무리 검토를 위해 재실행하는 단계에서 무슨 오류인지 (코렙 자체의 오류인 것 같습니다 ㅠㅠ) 제대로 출력이 되지 않는 것 같습니다. 시간 관계상 추가적인 수정 없이 제출해야 할 것 같습니다. 죄송합니다..