**각 기능 별 Workflow.**

1. **이메일 전송**

***원본 텍스트 1 – 직접적인 이메일이 주어졌을 때.***

@AI챗봇 [wwquen3568@naver.com](mailto:wwquen3568@naver.com)한테 메일전송해줘.

***데이터추출***

이메일 주소: 정규식으로 추출

예제) 정규식으로 이메일 추출

|  |
| --- |
| import re  txt2 = "wwquen3568@naver.com 으로 메일 보내줘."  email\_pattern = r'[a-z0-9]+@[a-z]+\.[a-z]{2,3}'  emails = re.findall(email\_pattern, txt2)  print(emails) |

***원본 텍스트 2 – 대상(사람)만 주어졌을 때.***

@AI챗봇 김광수한테 메일전송해줘.

***데이터추출***

이메일 주소: 직접적인 이메일 주소가 없기 때문에, Spacy를 통해 사람 명령을 찾고 “사람” 관련 단어를 찾고 User모델에서 쿼리를 통해 이메일을 찾아야함.

예제)

|  |
| --- |
| import spacy  nlp = spacy.load("ko\_core\_news\_lg") # 미리 학습된 모델  doc1 = nlp("김광수한테 메일전송해줘")  person = [ent.text for ent in doc1.ents if ent.label\_ == "PS"]  print(person) |

위 코드에서 person 명사를 추출하고

추후, 데이터베이스에서 쿼리를 통해 조회

**주의: person에 결과는 [“김광수한테”] 이며, 한국어 언어 특성상(고립어) 불용어(조사. 은,는,이,가,한테,)를 제거하는 작업이 필요함.**

**불용어 목록은 본인이 직접 작성해도 되고, github나 티스토리등 불용어 리스트 모아놓은 곳이 있는거 써도됨. (ex. https://deep.chulgil.me/hangugeo-bulyongeo-riseuteu/)**

**예제) 불용어 처리 예시**

|  |
| --- |
| **stop\_word = [“은”, “는”, “이”, “가”, “한테”]**  **for stop\_word in stop\_words:**  **person[0].replace(stop\_word, “”)** |

예제) 이메일 값 쿼리 조회

|  |
| --- |
| from django.contrib.auth import get\_user\_model  user = get\_user\_model().objects.get(name=person[0])  print(user.email) |

1. **스케줄 등록**

***원본 텍스트 1 – 정확한 날짜가 정해졌을 때.***

@AI챗봇 2024년 4월 31일 12시 30분에 캡스톤 발표가 있다.

***데이터추출***

시간: spacy를 통한 시간 추출(DT: 날짜, TI, 시간)

스케줄 등록 내용: BART 모델을 통한 요약문장 등록 (1줄 요약이면 될 듯)

|  |
| --- |
| import spacy  nlp = spacy.load("ko\_core\_news\_lg") # 미리 학습된 모델  doc1 = nlp("2024년 4월 31일 12시 30분에 캡스톤 발표가 있다.")  dates = [(ent.text, ent.label\_) for ent in doc1.ents if ent.label\_ == "DT"]  times = [(ent.text, ent.label\_) for ent in doc1.ents if ent.label\_ == "TI"]  print(dates) # [('2024년 4월 31일', 'DT')]  print(times) # [('12시 30분에', 'TI')] |

Dates: 년월일

Times: 시간 값

***원본 텍스트 1 – 추상적인 시간일 경우,***

@AI챗봇 내일 세미나를 해야한다.

@AI챗봇 2시간 후에 노기섭 교수님을 만나야한다.

***데이터추출***

시간: spacy를 똑같이 사용하면 되지만, 따로 나오는 단어에 대해 처리가 필요하다. spacy를 통한 시간 추출(DT: 날짜, TI, 시간)

스케줄 등록 내용: BART 모델을 통한 요약문장 등록 (1줄 요약이면 될 듯)

|  |
| --- |
| import spacy  nlp = spacy.load("ko\_core\_news\_lg") # 미리 학습된 모델  doc1 = nlp("2024년 4월 31일 12시 30분에 캡스톤 발표가 있다.")  doc2 = nlp("2시간 후에 노기섭 교수님을 만나야한다.")  dates1 = [(ent.text, ent.label\_) for ent in doc1.ents if ent.label\_ == "DT"]  times1 = [(ent.text, ent.label\_) for ent in doc1.ents if ent.label\_ == "TI"]  dates2 = [(ent.text, ent.label\_) for ent in doc2.ents if ent.label\_ == "DT"]  times2 = [(ent.text, ent.label\_) for ent in doc2.ents if ent.label\_ == "TI"]  print(dates1) # [('내일', 'DT')]  print(times1) # [ ]  print(dates2) #  print(times2) # [('2시간', 'TI')] <- 전 인지, 후 인지까지 안나옴. 이건 문제될 듯. |

1. **요약문 생성**

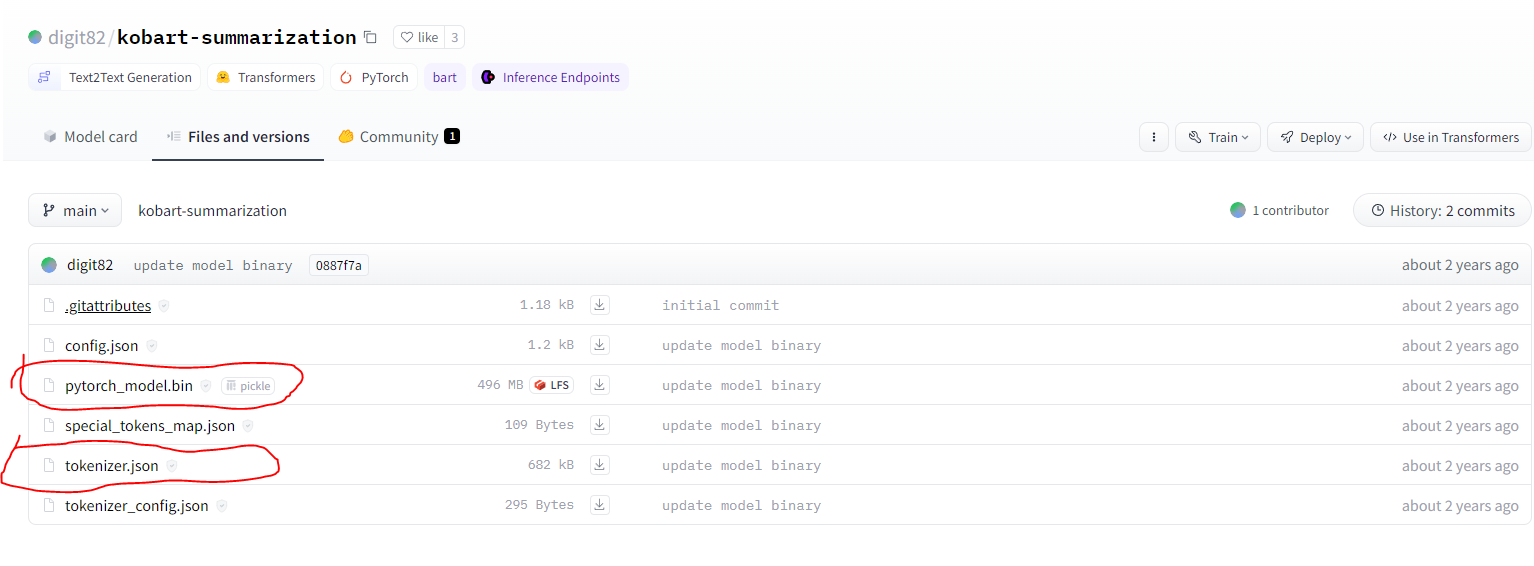
Github repo 클론

<https://github.com/seujung/KoBART-summarization>

(선택) 만약 허깅페이스 로그인 요청 권한 뭐 그런거 문제 생기면 모델 다운

<https://huggingface.co/digit82/kobart-summarization/tree/main>

tokenizer랑 model 둘 다



**추론 방법 (Github repo README.md 에 있음)**



**세부 설명**

tokenizer = PreTrainedTokenizerFast.from\_pretrained('digit82/kobart-summarization')

model = BartForConditionalGeneration.from\_pretrained('digit82/kobart-summarization')

'digit82/kobart-summarization’ 이 경로는 로컬 컴퓨터 경로가 아니라, huggingface repo 경로를 뜻함. (<https://huggingface.co/digit82/kobart-summarization/tree/main>)

거기에서 있는 tokenizer이랑 pytorch\_model.bin을 갖고오는 것.

summary\_ids = **model.generate**(torch.tensor([input\_ids]), num\_beams=4, max\_length=512, eos\_token\_id=1)

tokenizer.decode(summary\_ids.squeeze().tolist(), skip\_special\_tokens=True)

model.genertae() : 실제로 추론 결과 생성하는 함수. 이때, min\_length=30, max\_length=100 이런 추가 파라미터를 통해서 한문장 요약, 두문장 요약, 세문장 요약을 유도할 수 있다.

length\_penalty=1 라는 매개변수를 통해 길이 제한을 맞추지 않을 시 100% 패털티(벌점)을 부과함으로써 확실하게 그 길이를 지키게 할 수 있음.

더 세부적으로

summary\_ids = **model.generate**(torch.tensor([input\_ids]), num\_beams=4, max\_length=512, eos\_token\_id=1)

tokenizer.decode(summary\_ids.squeeze().tolist(), skip\_special\_tokens=True)

이 코드를 for문으로 감싸서

tokenizer.decode 의 문장이 정확히 원하는 문장 개수가 나올때까지 루프 돌려서 결과 낼수 있음.

(결국에는 확률형 모델이라서 가챠 좀 해야함).

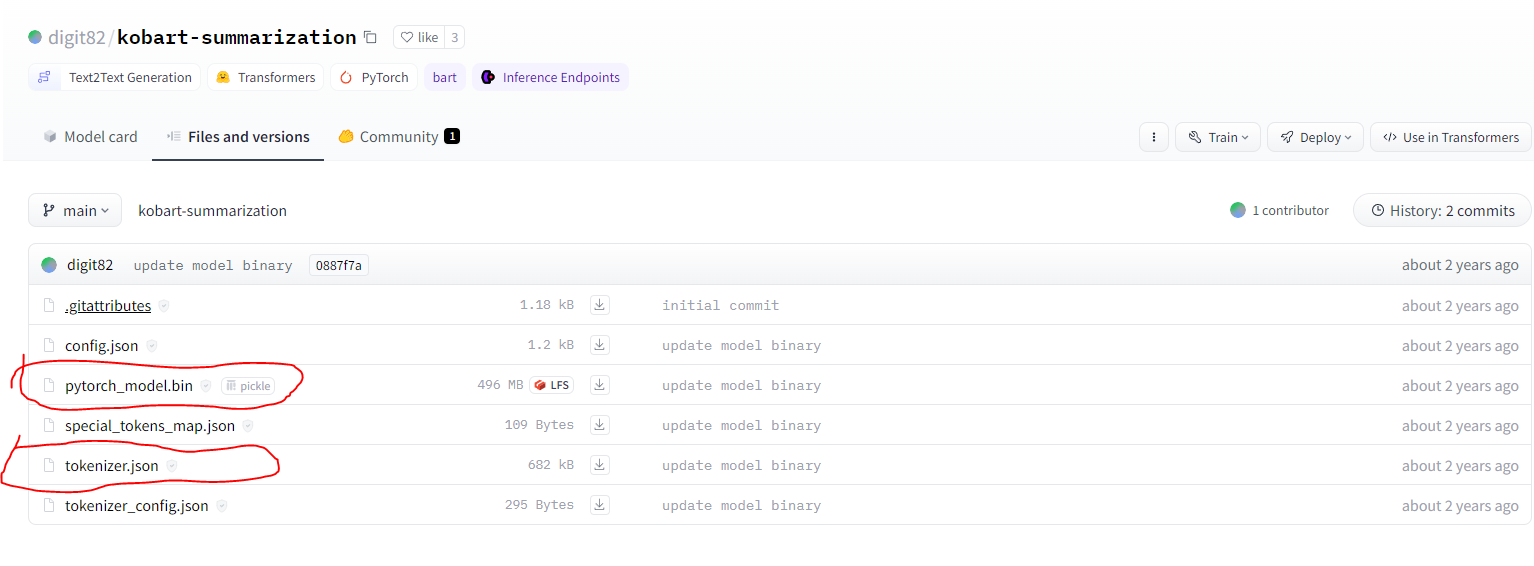
Ex)

|  |
| --- |
| # 한 줄 요약  summary\_ids = model.generate(input\_ids, max\_length=50, min\_length=25, length\_penalty=1.0)  print("한 줄 요약:", tokenizer.decode(summary\_ids[0], skip\_special\_tokens=True))  # 두 줄 요약  summary\_ids = model.generate(input\_ids, max\_length=100, min\_length=50, length\_penalty=1.0)  print("두 줄 요약:", tokenizer.decode(summary\_ids[0], skip\_special\_tokens=True))  # 세 줄 요약  summary\_ids = model.generate(input\_ids, max\_length=150, min\_length=75, length\_penalty=1.0)  print("세 줄 요약:", tokenizer.decode(summary\_ids[0], skip\_special\_tokens=True)) |

1. BART 학습 (제일 재밌는 파트 🤗)

깡통에서 학습해도 되고, 아니면 따로 아까 위에 사이트에서 미라학습된 모델 다운받아도되고. **(강추)**

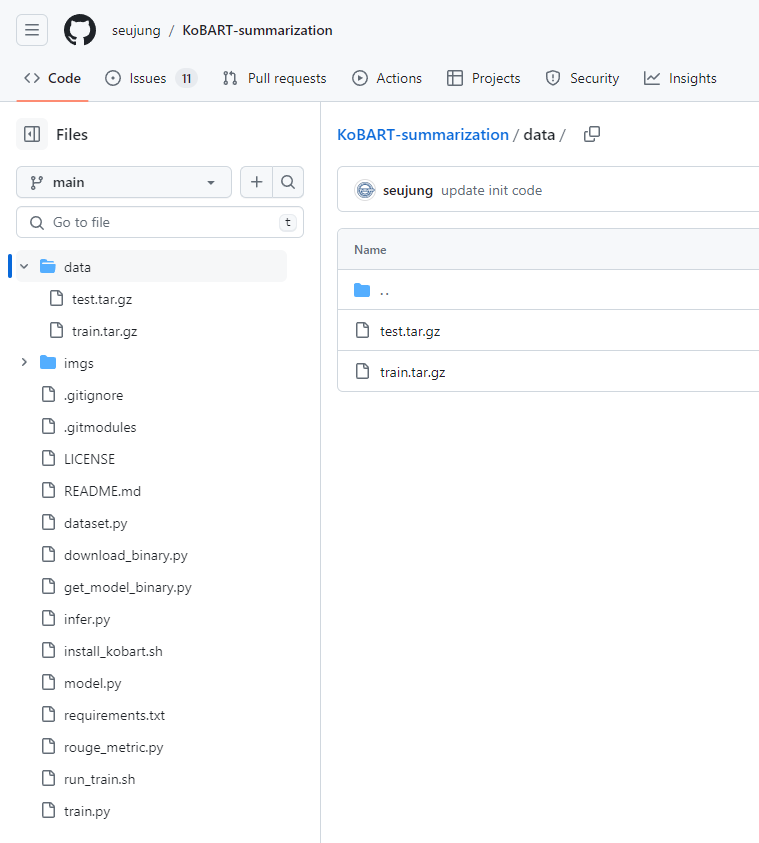
<https://huggingface.co/digit82/kobart-summarization/tree/main>



여기에서 마찬가지로 pytorch랑 tokenizer 둘다

학습하려면 아까 repo README.md에 나와있는것처럼 코드 돌리면되는데,

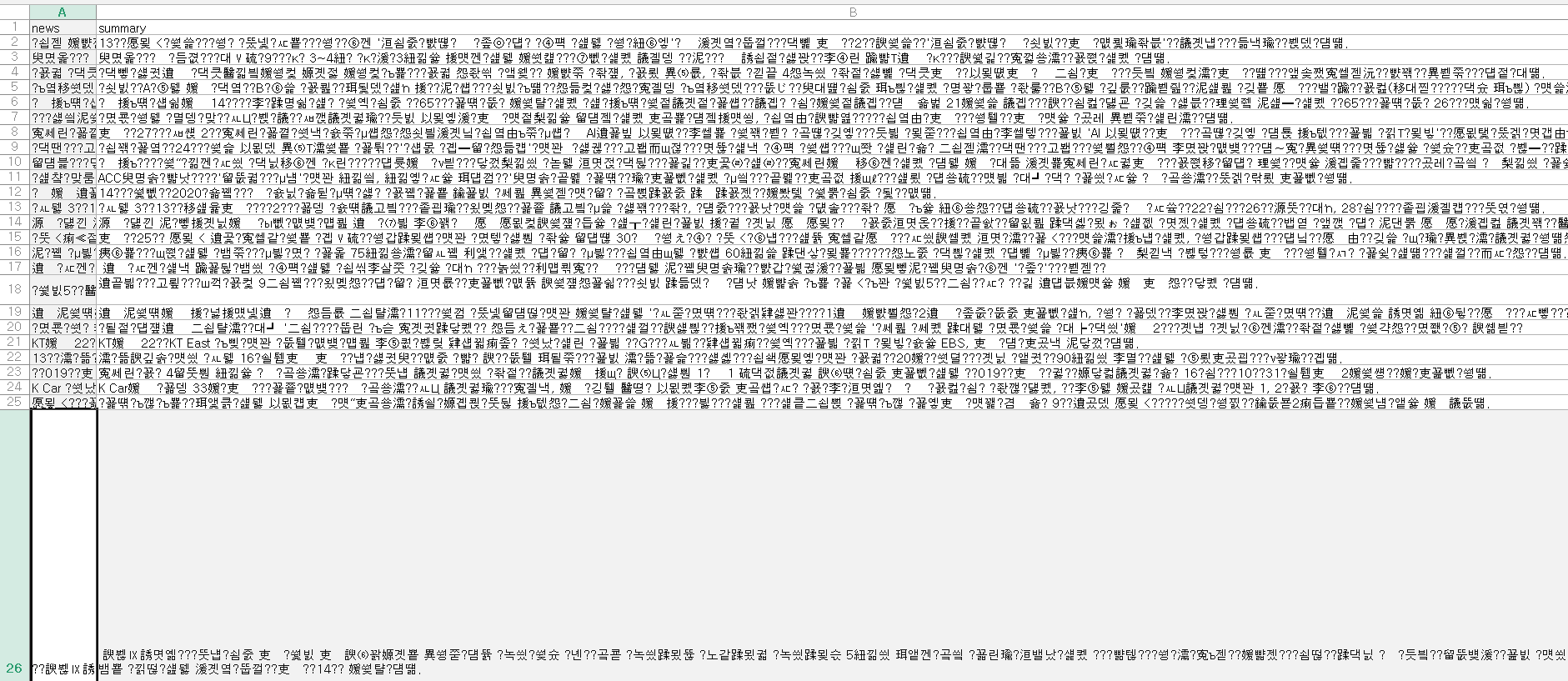
K- 깃허브 리포 답게 데이터셋 형식을 알려주지 않는다. 놀랍다.



여기 들어가서 test.tar.gz 를 까서 데이터를 확인해보면

학습데이터 형식은 단순히

Csv 형태로 열 2개로 이루어져있음 (news, summary).



Train.py 코드가 유연하게 설정 안되어있어서

반드시 csv 생성시, 칼럼명을 각각 news, summary로 해야함.

1. ***진짜 데이터 정제***

나만의 train.py를 만들어도 되긴 하다. Chatgpt한테 모델 이름이랑 이미 학습된 모델로 전이학습 물어보니깐 잘 알려준다.

하지만 이 문서에서는 여기 github에서 제공되는 train.py를 써보기로 한다.

|  |
| --- |
| import pandas as pd  # 데이터를 DataFrame으로 생성  data = {  "news": ["여기에 실제 뉴스 기사의 본문이 들어갑니다. 이 본문은 여러 문장으로 구성될 수 있습니다.", "또 다른 뉴스 기사 본문입니다. 이 역시 여러 문장으로 구성될 수 있습니다."],  "summary": ["기사의 핵심 내용을 짧게 요약한 문장입니다.", "이 기사의 요약문입니다."]  }  df = pd.DataFrame(data)  # TSV 파일로 저장  df.to\_csv("your\_dataset.tsv", sep='\t', index=False) |



이런식으로 데이터를 정제해서 pandas를 통해 탭 \t 로 데이터를 분리해서 하면 된다.

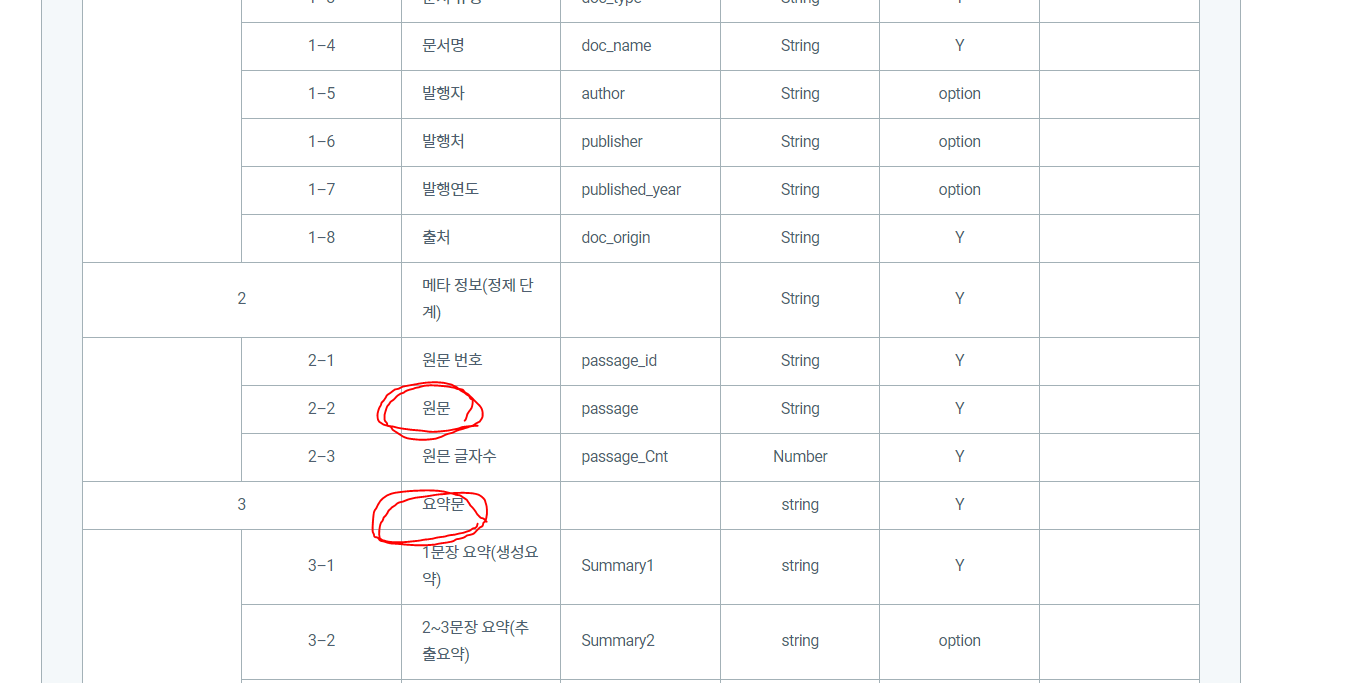
아래 index=False는 왼쪽에 추가적으로 id 칼럼을 추가해서 1, 2, 3 이런 순번행을 추가해준다. 당연히 우리는 이것이 필요 없기 때문에 제거한다.

1. **데이터셋 다운**

<https://www.aihub.or.kr/aihubdata/data/view.do?currMenu=&topMenu=&aihubDataSe=data&dataSetSn=582>



이 훌륭한 데이터셋을 다운받자.



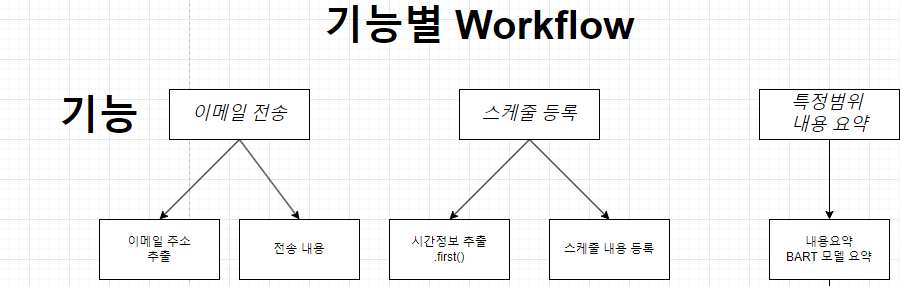
데이터셋을 유심히 잘 확인해서 원문 데이터를 news 칼럼에

요약문 데이터를 summary 칼럼에 잘 넣어서

아까 있는 데이터 저자 예시 코드처럼 넣어주면 된다. 여기에 for문을 쓰면 될것같다.



**기타**

****

**BART가 별로면 써볼만한 다른 모델들**

**KoBertSum**

* **내가 전에 연구실에서 사용했던 모델인데, 이건 사전에 학습된 모델이 없어서 너무 슬프다. 허깅페이스 어딘가에 있을 것 같긴한데… 일단 난 못 찾았다.**
* **BART는 Encoder-Decoder 형식으로 되어있는데, 이건 Encoder만 되어있는 모델이다. 즉, koBertSum+내가좋아하는 Transformer 학습된 모델을 붙여서 더 좋게 만들 수도 있다.**
* **또, Encoder만 되어있어서, 내가 Transformer를 받아서 뒤에 출력 개수를 3개로 만들면, 한번 추론할 때, 요약문이 3개가 튀어나오는 것도 만들 수 있다**

**링크**

[**https://github.com/uoneway/KoBertSum**](https://github.com/uoneway/KoBertSum)

**LLAMA2**

* **오픈소스 중에서는 최고봉인 LLM 모델이라고 한다.**
* **학습 방법은 다음 유튜브를 참고하면된다. LangChain 혹은 바닐라 pytorch로 학습시켜도 된다.**

[**https://www.youtube.com/watch?v=z2QE12p3kMM**](https://www.youtube.com/watch?v=z2QE12p3kMM)

[**https://www.youtube.com/watch?v=LslC2nKEEGU&list=LL&index=1&t=188s**](https://www.youtube.com/watch?v=LslC2nKEEGU&list=LL&index=1&t=188s)

**허깅페이스에 kor-llama2라해서 한글을 잘 쓰는 llama2 모델이 있다.**

**주의해야할게 llama2 모델을 쓰면, 학습 시간이 굉장히 오래걸린다. 이게 epoch을 못해도 300번이상은 해야하고, 데이터셋도 굉장히 많이 넣어야하는데,**

**그래서 영상 보면 epoch도 2번~10번만 돌리고, 데이터셋 양도 굉장히 적다. (50행 안쪽).**