|  |
| --- |
| 진로탐색학점제 최종보고서 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **학생** | **성명** | **이연준** | **학번** | **2071012** |
| **전공** | **모바일소프트웨어** | **연락처** | **010-3107-8786** |
| **활동주제명** | **인플루언서 통계 분석 및 멀티 모달리티를 활용한 유틸리티** | | | |
| **지도교수** | **성명** | **송미화** | **소속** | **컴퓨터공학부** |
| 진로탐색학점제 최종보고서에 대하여 위와 같이 제출합니다.  2024년 월 일  지도교수 : 송미화 (인) | | | |

※*글꼴 : 바탕 / 크기 : 10pt / 장평 : 100% / 줄간격 : 160% ※다음 장 추가 작성 가능*

**[최종보고서는 학점 인정을 위한 중요한 근거 자료입니다. 성실하고 구체적으로 작성해주세요]**

**1. 진로탐색학점제 활동 목표 및 달성 목표**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **- 초기 목표(팀 목표) –**  전공수업(데이터 마이닝)에서 배운 데이터 마이닝 기법을 통해 소셜 미디어(유튜브)에서 데이터를 추출하고(웹 크롤링을 통한 유튜브 댓글 수집 및 유튜브 영상 자료에 google video intelligence를 통한 text 추출) 영상 댓글을 평가하여 우호적인 댓글과 비우호적 댓글을 점수를 매기는 알고리즘을 AI 모델링을 통해 개발한다. 이는 직접 AI를 모델링하거나 만들어진 AI 모델을 가져와 (chat GPT API 등) 우리가 사용할 목적에 맞게 AI 파인튜닝할 수 있다.  댓글 평가 알고리즘으로 팔로워 중 우호적인 의견을 가진 사람과 비우호적인 의견을 가진 사람의 수를 카툰트하여 팔로워 지지율(전체 댓글 중 우호적인 댓글의 비율)을 도출해내고, 해당 유튜버의 모든 영상에 대해 지지율을 도출해 해당 유튜버의 팔로워 지지율까지 도출해낸다.   |  |  |  | | --- | --- | --- | | **주차** | **활동 목표 및 내용** | **투입시간**  **(1주일기준)** | | 1주차 | 파이썬 언어로 웹 크롤링을 통한 영상 댓글 수집 모듈 개발 및 문장 벡터화( 벡터화 방법론 탐색 ) | 20시간 | | 2주차 | 벡터화 된 문장을 TF-IDF방식으로 변환하고, SQL DB에 연결하여 URL별 테이블 생성 및 유튜버의 영상 테이블들의 집합을 유튜버로 설정하여 데이터베이스 구조 작성 | 20시간 | | 3주차 | 교수님에게 1차 지도점검 후 방법론 변화 및 중간보고서 작성(7월 12일 중간 보고서 제출) | 20시간 | | 4주차 | 사이킷 런을 사용한 지도학습-회귀 방식 및 텐서플로우를 사용한 인공신경망 방식 중 더 잘 맞는다고 생각하는 방식을 택해 댓글 감정 분석 진행. | 20시간 | | 5주차 | 교수님에게 2차 지도점검 후 댓글 평가 AI 모델을 개선하여 정확도 증진 후 구글 vertex ai를 사용한 멀티 모달리티 영상분석 도전 | 20시간 | | 6주차 | 지지도가 낮고 조회수가 높은 나락 베스트 및 모든 영상의 평균지지도가 높고, 조회수가 높은 홍보효과 베스트 그리고 사용자에게 직접 댓글을 입력받아 영상에 대한 감정을 분석해보는 참여형 댓글분석 기능을 웹페이지에 구현한다. | 20시간 | | 7주차 | 최종 보고서 및 ppt작성 | 20시간 |   **- 진행 중 변경점 –**  3주차 시점에 4~5주차에 해당하는 사항까지 빠르게 개발이 진행되어 이로 인해 추가적인 여유시간이 생겼다. 따라서 추가적인 멀티 모달리티 ai 모델을 사용하여 추가적인 유틸리티 및 웹서비스 구축에 시간을 더 할애하기 위해 전체적인 계획을 앞당겨서 진행하였다.  또한 멀티 모달리티를 구현함에 있어서 여러 논문들을 참고하게 되었고, 웹 페이지 구성에 초점을 맞춰 여러가지 메타 데이터를 활용한 종합적인 수치를 도출하고자 하였다.  **- 개인 목표 –**  3학년 1학기 데이터마이닝 수업에서 학습한 텍스트 마이닝 기법을 활용하여 유의미한 데이터를 분석할 수 있게 된다.  데이터의 전반적인 전처리 과정을 원할하게 진행해 MySql Database에 업로드하고, 제공하고자 하는 서비스에 맞춰 활용할 수 있도록 하는 능력을 기른다.  빅데이터와 AI에 기반한 프로젝트를 진행하며 이를 활용할 수 있는 능력을 기른다.  **- 달성한 목표 –**  구글 API를 활용하여 데이터 수집 및 분석 모듈을 개발  Google api youtube Version3 API를 활용하여 유튜브에 관련된 데이터를 자동적으로 수집하는 모듈을 개발하였고, google에서 제공하는 여러 python 라이브러리를 활용하였다. 이를 통해 유투브 댓글을 웹 클롤링하였고, Google Trend를 통한 검색횟수 및 연관검색어를 수집하여 유튜버에 대한 관심도를 차트로 시각화하였다. 또한 Ocr을 활용하여 영상자체에 대한 텍스트 데이터를 수집하였다.  Open AI와 KoBert 모델을 파인튜닝을 진행해 다중 감정분석 모듈을 개발  댓글을 분석하기 위하여 KoBert모델을 사용하여 훈련 데이터와 테스트 데이터를 이용해 7가지 감정으로 분류하여 텍스트를 분석하였다. 정확도를 높이기 위하여 GPT모델을 활용한 Open AI에서 제공하는 jsonl형식의 DataSet을 기반으로 파인튜닝하여 정확도를 87%까지 높여 좋은 성능을 보여주었다.  Sql DB 생성 및 관리와 웹 페이지와의 연동  수집한 데이터들을 전처리화시킨 후 엑셀로 추출했던 데이터들을 MySql Database에 업로드하고 이를 활용할수 있도록 Sql 자체 문법을 사용하여 유의미한 데이터를 생성하여 새로운 스키마로 표현하였다. Django Server와 Database를 연동하여 웹페이지에서 DB 데이터를 활용할 수 있게 하였고, 주기적으로 업데이트 하여 하루마다 데이터를 연동시켜 웹 페이지에 반영할 수 있도록 하였다.  이미지 및 영상을 분석하여 텍스트로 추출하여 Summary  영상을 프레임별 이미지로 나누고 해당하는 프레임을 Ocr(광학 문자 인식)을 통해 추출한다. 또한 Speech to Text API도 사용하여 영상에 사용된 음성을 텍스트로 추출한다. 이 두가지를 통해 추출한 텍스트 데이터를 요약하여 유튜브 컨텐츠에 대한 전반적인 정보와 주제를 파악할 수 있게 하였다.  Prompt를 입력받아 이미지를 분석하고 Prompt기반으로 이미지를 변경하거나 생성하는 모듈 개발  Pix2Pix\_Trubo를 활용해 작성된 이미지를 변환하거나 새로 생성하는 모델을 이용해 웹 페이지에서 새로운 이미지나 썸네일을 변경해줄 수 있도록 하였다.  **- 달성한 개인 목표 –**  수집된 데이터셋 전처리 및 텍스트 마이닝  웹 크롤링을 통해 수집된 댓글들에 대해서 형태소를 분석하고 이에 대한 오탈자 수정 및 전처리과정을 진행하였다. Konlpy Okt라이브러리를 사용하여 오탈자 및 형태소 분서을 진행하였고, 이를 통해 TF-IDF 값으로 벡터화 시키는 작업에 필요한 데이터셋을 구축하였다.  MySql DataBase 전처리 및 유의미한 스키마 생성  수집된 여러 메타 데이터들을 MySql에 업로드하는 과정에서 유의미한 데이터로 변환하여 새로운 스키마를 생성하고 이에 해당하는 값들을 Sql 구문을 활용하여 삽입하고 웹 페이지로 연동하기 위해 테이블 및 데이터셋을 구축하였다.  빅데이터 및 AI를 활용한 프로젝트 진행 경험치 축적  빅데이터 분석에 활용되는 여러 API에 대해 습득하고, 빅데이터를 활용하는 AI에 대한 이해도 역시 상승하였다. AI모델 구축에 파인튜닝 및 멀티 모달리티 개념에 대해서도 이해도가 상승하였고, 이를 활용하여 웹 퍼블링싱 또한 추후 프로젝트 및 개발에서 무리없이 진행할 수 있을정도의 수준이 되었다. 추구하던 프로젝트 경험 증진에 적합한 활동이었다. |

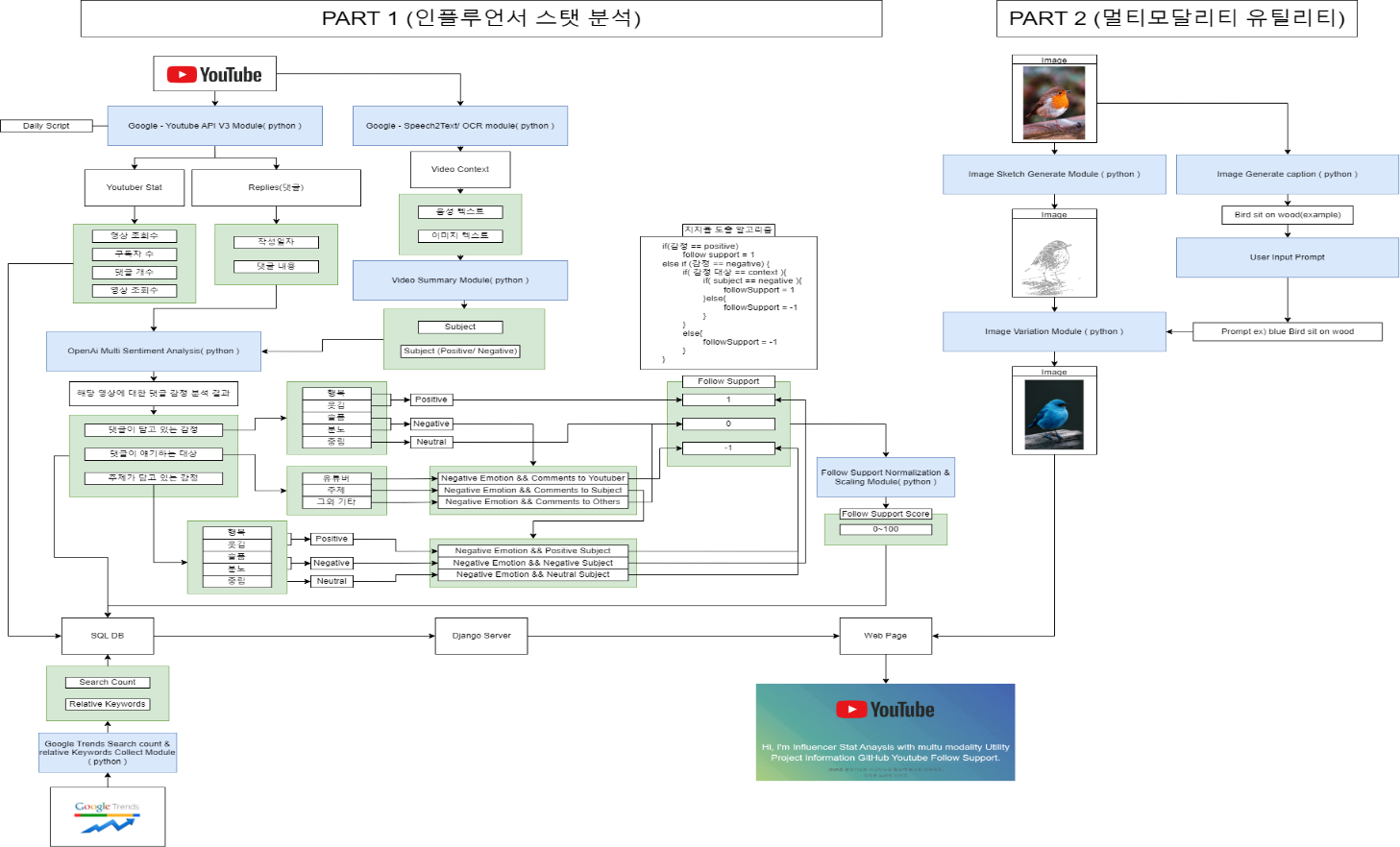
**2. 팀원별 기여도 평가**

|  |  |  |
| --- | --- | --- |
| 팀원 이름 | 기여도 | 의견 |
| 이연준 | 10% | 멀티 모달리티 및 AI에 대해 학습의 과정에서 이해도 문제와 능력부족으로 참여율이 떨어졌다. |
| 노태원 | 20% | 서브개발자로서의 역할을 충실히 하였다. |
| 김정훈 | 70% | 메인개발자로서 프로젝트에 대한 이해도가 다른 두명보다 월등히 높았고, 주도적으로 프로젝트를 이끌어서 수행하였다.  협업 프로젝트라기 보다는 메인개발자와 서브개발자형식인 튜터링과 같은 느낌이 다소 형성되어 시간 및 능력적으로 다른 팀원에게 도움을 많이 주었다. |

**3. 진로탐색학점제 활동내용 및 결과, 활동 성과** ※주요 활동 내용 및 결과에 대해 상세히 기술

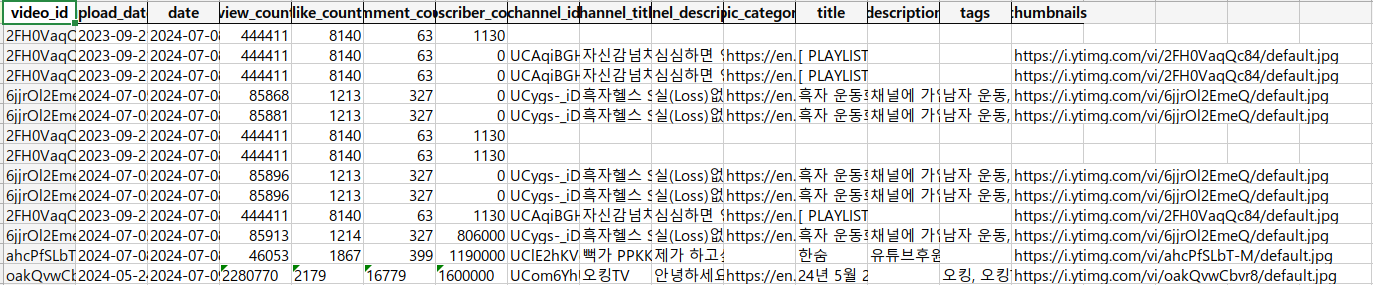
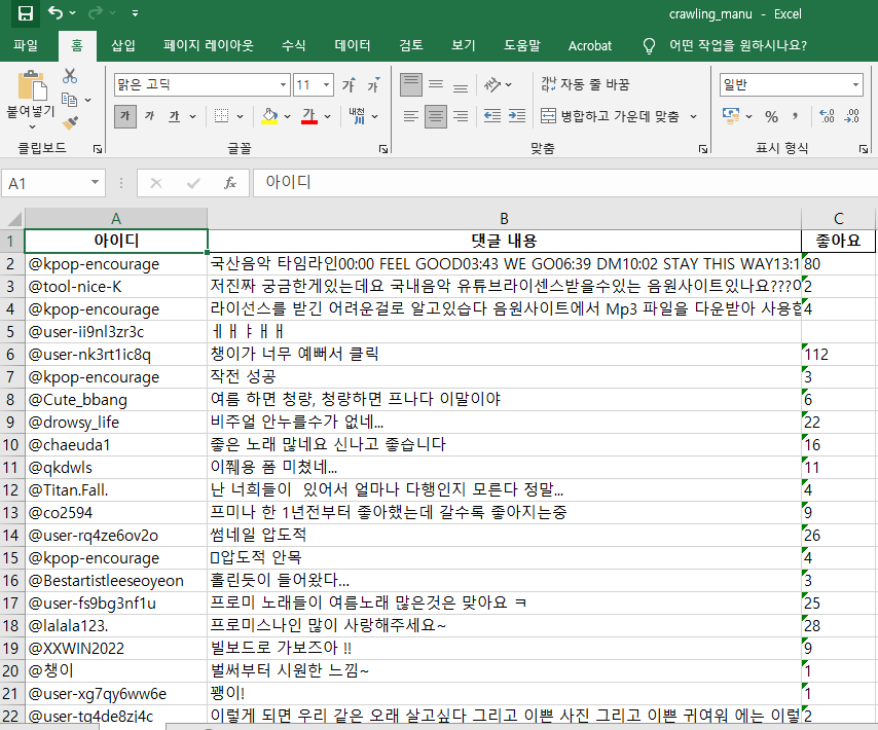
**팀 개발 Github Repository** - <https://github.com/thuygom/youtube_follow_support>

**- 개발 흐름도 -**



**- URL을 통한 유튜브 웹 크롤링 (데이터 수집) –**

|  |
| --- |
| - 수동 버전 -  buttons = driver.find\_elements(By.CSS\_SELECTOR, "ytd-button-renderer#more-replies > yt-button-shape > button")  print(buttons[1])  for i in range(len(buttons)):  buttons[i].send\_keys(Keys.ENTER)  print("Preprocessing complete")  #상호작용할 수 있는 버튼을 가져와 답글 더보기 버튼에 Enter키를 보내는 로직    id\_list = soup.select("div#header-author > h3 > #author-text > span")  comment\_list = soup.select("div#content > yt-attributed-string#content-text > span")  like\_list = soup.select("span#vote-count-middle")  #CSS Selector로 원하는 정보를 선택해 가져오는 로직 코드  - 자동 버전 –  # Function to get video statistics  def get\_video\_info(api\_obj, video\_id):  response = api\_obj.videos().list(  part='snippet,statistics,topicDetails',  id=video\_id  ).execute()  video\_info = response['items'][0]  snippet = video\_info['snippet']  statistics = video\_info['statistics']  topic\_details = video\_info.get('topicDetails', {}) # Handle cases where topicDetails might not exist  return snippet, statistics, topic\_details  #google API를 통해 필요한 stat정보를 담고있는 객체를 불러오는 핵심 로직 |



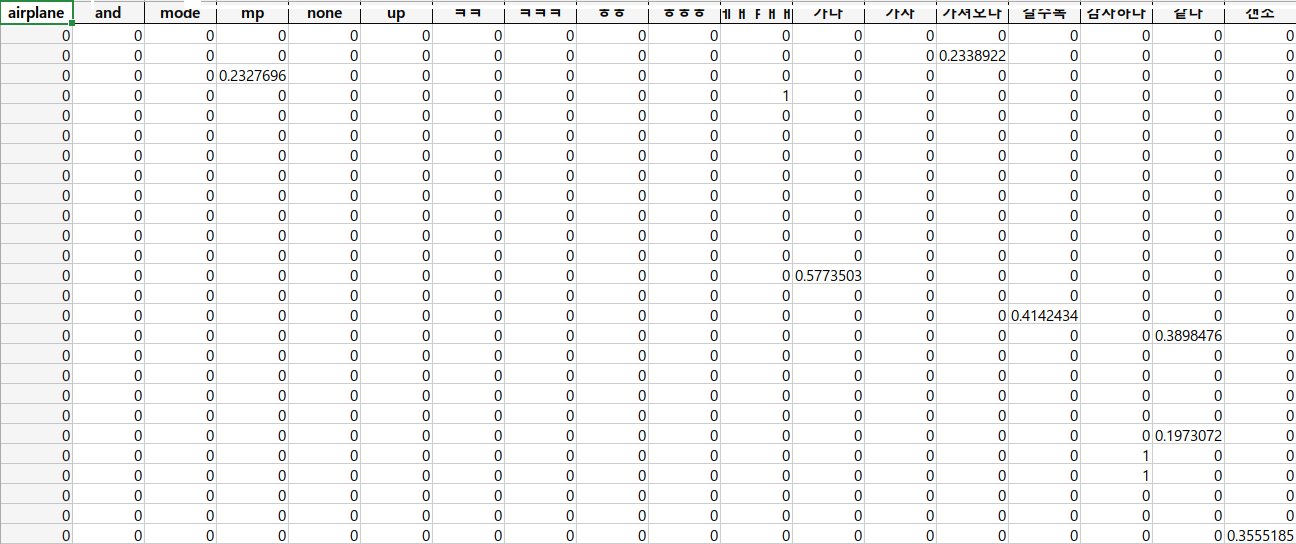
**- 은어 및 욕설 치환과 단어 벡터화 (데이터 전처리) –**

**은어 및 욕설 치환**

|  |
| --- |
| # 형태소 분석기 초기화  okt = Okt()  # 은어를 표준어로 변경하는 딕셔너리  replace\_dict = {  "ㅅㅂ": "싫어",  "ㅈㄹ": "뭐라",  "ㅈㄴ": "매우",  "존나": "매우",  "ㄲㅈ": "나가",  "빡친다": "화나다",  "개꿀": "좋다",  "열폭": "열등감 폭발",  "조졌다": "망하다",  "쩐다":"대단하다",  "ㄹㅇ": "진짜",  "듣보": "흔하지 않다",  "커엽다": "귀엽다",  "인싸": "친구가 많다",  "아싸": "친구가 적다",  "존버": "오래 버티기",  "국산": "테스트용" # 첫번째 문장 첫번째 단어 변경사항 테스트용  }  # 비속어 및 특정 단어 변경 함수  def replace\_words(morphs, replacement\_dict):  return [replacement\_dict.get(word, word) for word in morphs]  # 두 번째 컬럼의 각 문장에 대해 형태소 분석 수행 및 문장 변환  def analyze\_and\_replace(sentence):  morphs = okt.morphs(sentence, stem=True, norm=True)  replaced\_morphs = replace\_words(morphs, replace\_dict)  return ' '.join(replaced\_morphs) |

**TF-IDF 단어 벡터화**

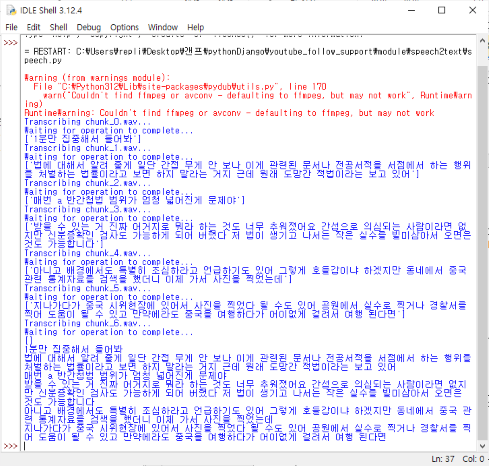
|  |
| --- |
| # 전처리된 댓글 문장 추출  preprocessed\_sentences = preprocessed\_df['Processed\_Comments'].tolist()  # TF-IDF 모델 생성 및 데이터프레임 저장  def save\_tfidf\_to\_excel(corpus, output\_file\_path, threshold=0.2):  # TF-IDF 모델 생성  vectorizer = TfidfVectorizer()  X = vectorizer.fit\_transform(corpus)  # TF-IDF 값 데이터프레임 생성  tfidf\_df = pd.DataFrame(X.toarray(), columns=vectorizer.get\_feature\_names\_out())  # 임계값을 넘는 열 확인  above\_threshold\_cols = tfidf\_df.columns[(tfidf\_df > threshold).any()]  # 임계값을 넘는 열만 포함하여 저장  tfidf\_df\_filtered = tfidf\_df[above\_threshold\_cols]  # 결과를 엑셀 파일로 저장  tfidf\_df\_filtered.to\_excel(output\_file\_path, index=False)  print(f"TF-IDF 결과가 {output\_file\_path} 파일로 저장되었습니다.")  # TF-IDF 결과 저장  output\_file\_path = '../xlsx/result\_tfidf.xlsx'  save\_tfidf\_to\_excel(preprocessed\_sentences, output\_file\_path) |



**- 영상 자체 메타 데이터 수집 및 요약 (데이터 수집, 데이터 전처리) –**

**Speech to Text python module를 활용한 데이터 정제 및 수집**

|  |
| --- |
| from google.cloud import speech\_v1p1beta1 as speech  from pydub import AudioSegment  import io  import os  import wave  def get\_sample\_rate(file\_path):  """WAV 파일의 샘플 레이트를 확인합니다."""  with wave.open(file\_path, 'rb') as wf:  sample\_rate = wf.getframerate()  return sample\_rate  def convert\_to\_mono(file\_path):  """오디오 파일을 모노로 변환합니다."""  audio = AudioSegment.from\_file(file\_path)  if audio.channels != 1:  audio = audio.set\_channels(1)  mono\_file = f"mono\_{os.path.basename(file\_path)}"  audio.export(mono\_file, format="wav")  return mono\_file  return file\_path  def resample\_audio(file\_path, target\_sample\_rate=48000):  """오디오 파일을 리샘플링합니다."""  audio = AudioSegment.from\_file(file\_path)  if audio.frame\_rate != target\_sample\_rate:  audio = audio.set\_frame\_rate(target\_sample\_rate)  resampled\_file = f"resampled\_{os.path.basename(file\_path)}"  audio.export(resampled\_file, format="wav")  return resampled\_file  return file\_path  def convert\_to\_16bit(file\_path):  """WAV 파일을 16비트 샘플로 변환합니다."""  audio = AudioSegment.from\_file(file\_path)  audio = audio.set\_sample\_width(2) # 16비트 샘플  bit16\_file = f"bit16\_{os.path.basename(file\_path)}"  audio.export(bit16\_file, format="wav")  return bit16\_file  def split\_audio(file\_path, chunk\_length\_ms=10000):  """오디오 파일을 주어진 길이(밀리초)로 자릅니다."""  audio = AudioSegment.from\_file(file\_path)  chunks = []  for i in range(0, len(audio), chunk\_length\_ms):  chunk = audio[i:i + chunk\_length\_ms]  chunk\_file = f"chunk\_{i // chunk\_length\_ms}.wav"  chunk.export(chunk\_file, format="wav")  chunks.append(chunk\_file)  return chunks  def transcribe\_audio\_file(file\_path):  """Google Cloud Speech-to-Text API를 사용하여 음성을 텍스트로 변환합니다."""  client = speech.SpeechClient.from\_service\_account\_file('myKey.json')  # 오디오 파일의 샘플 레이트 확인  sample\_rate = get\_sample\_rate(file\_path)  # 오디오 파일을 모노로 변환  mono\_file\_path = convert\_to\_mono(file\_path)  # 오디오 파일 리샘플링  resampled\_file\_path = resample\_audio(mono\_file\_path, target\_sample\_rate=sample\_rate)  # 오디오 파일 비트 깊이 변환  bit16\_file\_path = convert\_to\_16bit(resampled\_file\_path)  with io.open(bit16\_file\_path, "rb") as audio\_file:  content = audio\_file.read()  audio = speech.RecognitionAudio(content=content)  config = speech.RecognitionConfig(  encoding=speech.RecognitionConfig.AudioEncoding.LINEAR16,  sample\_rate\_hertz=sample\_rate, # 리샘플링한 샘플 레이트 사용  language\_code="ko-KR",  )  # 파일이 길 경우, long\_running\_recognize 사용  operation = client.long\_running\_recognize(config=config, audio=audio)  print('Waiting for operation to complete...')  response = operation.result(timeout=90)  transcripts = []  for result in response.results:  transcripts.append(result.alternatives[0].transcript)  return transcripts  def transcribe\_chunks(chunk\_files):  """여러 오디오 조각을 텍스트로 변환합니다."""  all\_transcripts = []  for chunk\_file in chunk\_files:  print(f"Transcribing {chunk\_file}...")  transcripts = transcribe\_audio\_file(chunk\_file)  print(transcripts)  all\_transcripts.extend(transcripts)  os.remove(chunk\_file) # 임시 조각 파일 삭제  return all\_transcripts  # 사용 예제  audio\_file\_path = 'audio.wav'  chunk\_files = split\_audio(audio\_file\_path, chunk\_length\_ms=10000) # 10초 길이의 조각으로 나누기  transcripts = transcribe\_chunks(chunk\_files)  # 전체 텍스트 출력  print("\n".join(transcripts)) |



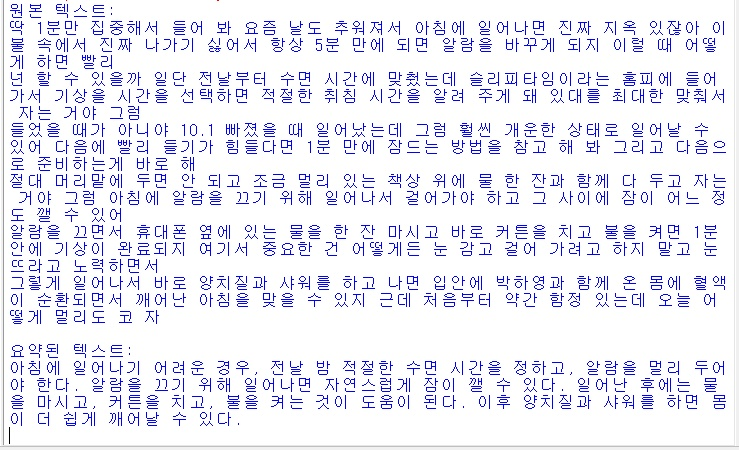
**- OCR을 활용한 이미지 데이터 -**

|  |
| --- |
| from google.cloud import vision\_v1  import io  # 이미지에서 텍스트 추출  def detect\_text\_in\_image(image\_path, client):  with io.open(image\_path, 'rb') as image\_file:  content = image\_file.read()  image = vision\_v1.Image(content=content)  response = client.text\_detection(image=image)  texts = response.text\_annotations  if texts:  print(f"Detected text in {image\_path}:")  for text in texts:  print(text.description)  else:  print(f"No text detected in {image\_path}.")  # Google Cloud Vision 클라이언트 설정  def setup\_vision\_client():  return vision\_v1.ImageAnnotatorClient.from\_service\_account\_file('myKey.json')  # 메인 함수  def main():  image\_path = './a.png' # 텍스트를 추출할 이미지 파일의 경로  client = setup\_vision\_client()  detect\_text\_in\_image(image\_path, client)  if \_\_name\_\_ == '\_\_main\_\_':  main() |

텍스트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

**수집된 영상정보 Summary**



**- OpenAi 및 KoBert 파인튜닝 감정분석 모델 –**

**OpenAI 모델**

|  |
| --- |
| # OpenAI API를 사용하여 감정 분석을 수행하는 함수 정의  def predict\_sentiment(text):  try:  messages = [  {"role": "system", "content": "You are a helpful assistant."},  {"role": "user", "content": f"'{context}' 문맥을 기반으로 '{text}'를 행복, 웃김, 슬픔, 분노, 중립 5가지 감정중 하나로 라벨링 후 감정 라벨을 반환하라. "}  ]  response = openai.ChatCompletion.create(  model="gpt-3.5-turbo",  messages=messages,  max\_tokens=50,  stop=["\n", "감정 분석 결과:"],  )  # OpenAI API의 응답에서 감정 라벨 추출  predicted\_label = response.choices[0].message['content'].strip().lower()  return predicted\_label  except openai.error.APIError as e:  print(f"OpenAI API 호출 오류: {e}")  return "API 호출 오류"  # OpenAI API를 사용하여 감정 분석을 수행하는 함수 정의  def predict\_object(text, num\_predictions=5):  try:  messages = [  {"role": "system", "content": "You are a helpful assistant."},  {"role": "user", "content": f"문맥: {context}. 유튜버: {youtuber}."},  {"role": "user", "content": "다음 문장이 얘기하는 대상이 context, youtuber, other중 어떤 것인지 선택하세요. 예: '안드레진 미친놈이네' -> context"},  {"role": "user", "content": f"문장: '{text}'"}  ]  results = []  for \_ in range(num\_predictions):  response = openai.ChatCompletion.create(  model="gpt-3.5-turbo",  messages=messages,  max\_tokens=50,  )  predicted\_label = response.choices[0].message['content'].strip().lower()  results.append(predicted\_label)  most\_common\_label = Counter(results).most\_common(1)[0][0]  return most\_common\_label  except openai.error.APIError as e:  print(f"OpenAI API 호출 오류: {e}")  return "API 호출 오류" |

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

**KoBert 모델**

|  |
| --- |
| # 데이터셋 클래스 정의  class CommentDataset(Dataset):  def \_\_init\_\_(self, data\_list, tokenizer, max\_length):  self.data\_list = data\_list  self.tokenizer = tokenizer  self.max\_length = max\_length    def \_\_len\_\_(self):  return len(self.data\_list)    def \_\_getitem\_\_(self, idx):  comment, label = self.data\_list[idx]    encoding = self.tokenizer(comment, truncation=True, padding='max\_length', max\_length=self.max\_length, return\_tensors='pt')  inputs = {  'input\_ids': encoding['input\_ids'].flatten(),  'attention\_mask': encoding['attention\_mask'].flatten(),  'labels': torch.tensor(label, dtype=torch.long)  }  return inputs  # 학습 데이터셋 로딩  data\_list = load\_dataset(train\_file\_path)  print(f"전체 데이터셋 크기: {len(data\_list)}")  # 데이터셋을 학습용과 검증용으로 분리 (60% 학습, 40% 검증)  train\_data\_list, eval\_data\_list = train\_test\_split(data\_list, train\_size=0.6, test\_size=0.4, random\_state=42)  train\_dataset = CommentDataset(train\_data\_list, tokenizer, max\_length=128)  eval\_dataset = CommentDataset(eval\_data\_list, tokenizer, max\_length=128)  training\_args = TrainingArguments(  per\_device\_train\_batch\_size=128,  per\_device\_eval\_batch\_size=32,  num\_train\_epochs=5,  logging\_dir='./logs',  logging\_steps=100,  evaluation\_strategy="epoch", # 에포크마다 검증  save\_steps=500, # 500 steps 마다 모델 저장  output\_dir='./results2', # 모델 저장 디렉토리  overwrite\_output\_dir=True,  )  # 정확도와 F1 score를 계산하는 함수 정의  def compute\_metrics(pred):  labels = pred.label\_ids  preds = pred.predictions.argmax(-1)  acc = accuracy\_score(labels, preds)  f1 = f1\_score(labels, preds, average='weighted') # weighted average F1 score  return {  'accuracy': acc,  'f1\_score': f1,  }  # Trainer 초기화 및 Fine-tuning  trainer = Trainer(  model=model,  args=training\_args,  train\_dataset=train\_dataset,  eval\_dataset=eval\_dataset,  compute\_metrics=compute\_metrics,  )  # 모델 가중치 일부 확인  print("모델 가중치 확인:")  print(model.bert.encoder.layer[0].attention.self.query.weight[:5]) # 일부 가중치 출력  # Fine-tuning 진행  trainer.train()  # 학습된 모델 및 토크나이저 저장  trainer.save\_model('./results2')  tokenizer.save\_pretrained('./results2')  print("모델과 토크나이저가 ./results2 디렉토리에 저장되었습니다.") |

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

**텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

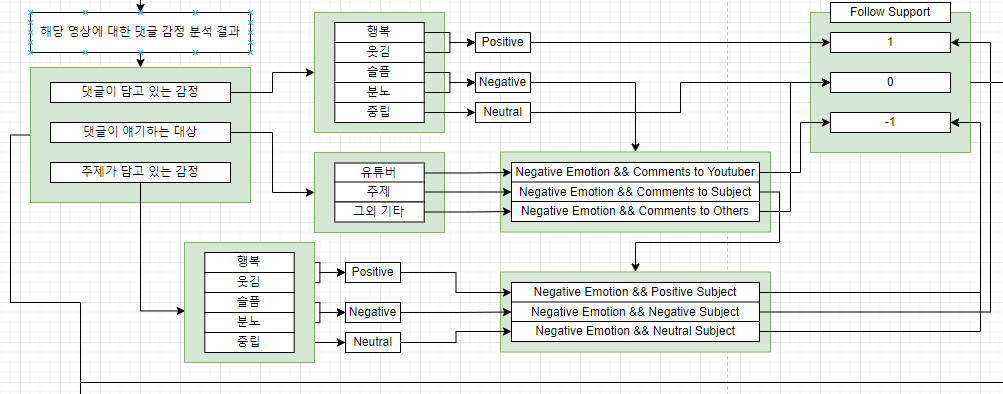
자동 생성된 설명**

**텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명**

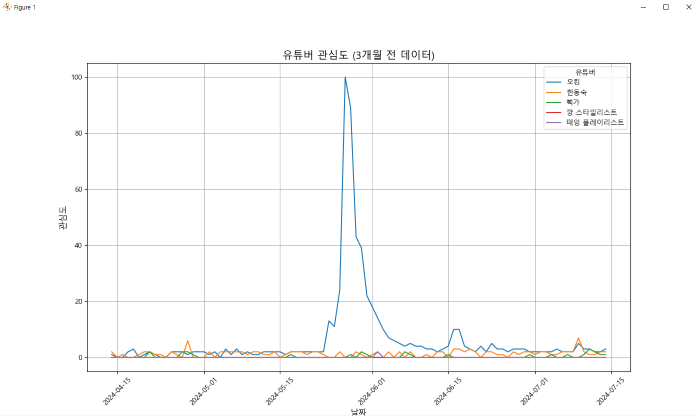
**- 감정 분석을 통한 Follow Support 점수 산정 및 Google trend, 연관검색어 점수 산정 –**

**Follow Support 점수 도출**



**Google API를 활용한 google trend 및 연관검색어 메타 데이터 수집 및 전처리**

|  |
| --- |
| import time  from pytrends.request import TrendReq  import matplotlib.pyplot as plt  from matplotlib import font\_manager, rc  import pandas as pd  # 한글 폰트 설정 (Windows에서 기본 폰트 사용)  font\_path = 'C:/Windows/Fonts/malgun.ttf' # Windows 기본 폰트 경로 설정  # 폰트 설정  font\_name = font\_manager.FontProperties(fname=font\_path).get\_name()  rc('font', family=font\_name)  # pytrends 세션 시작  pytrends = TrendReq(hl='ko', tz=360) # 'hl'을 'ko'로 설정하여 한글 결과 받기  # 유튜버 키워드 리스트  keywords = ["오킹", "한동숙", "뻑가", "깡 스타일리스트", "때잉 플레이리스트"]  # 3개월 전 데이터를 가져오기 위해 시간 범위 설정  timeframe = 'today 3-m'  # pytrends로 데이터 요청  pytrends.build\_payload(keywords, cat=0, timeframe=timeframe, geo='', gprop='')  time.sleep(60) # 요청 후 대기  # 관심도 데이터 가져오기  interest\_over\_time\_df = pytrends.interest\_over\_time()  # 데이터프레임을 Excel 파일로 저장  interest\_over\_time\_df.to\_excel('trend\_interest.xlsx', index=True)  # 데이터 시각화  plt.figure(figsize=(14, 8))  for keyword in keywords:  plt.plot(interest\_over\_time\_df.index, interest\_over\_time\_df[keyword], label=keyword)  plt.title('유튜버 관심도 (3개월 전 데이터)', fontsize=15)  plt.xlabel('날짜', fontsize=12)  plt.ylabel('관심도', fontsize=12)  plt.legend(title='유튜버', fontsize=10)  plt.xticks(rotation=45)  plt.grid(True)  plt.show() |



텍스트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

**관심도 및 메타데이터 점수 산정**

|  |
| --- |
| import time  from pytrends.request import TrendReq  import pandas as pd  # pytrends 세션 시작  pytrends = TrendReq(hl='ko', tz=360)  # 유튜버 키워드 리스트  keywords = ["오킹", "한동숙", "뻑가", "깡 스타일리스트", "때잉 플레이리스트"]  # 연관 검색어를 저장할 데이터프레임 초기화  related\_queries\_df = pd.DataFrame(columns=['유튜버', '연관 검색어'])  # 각 유튜버 키워드에 대해 연관 검색어 가져오기  for keyword in keywords:  time.sleep(600) # 요청 간 대기  pytrends.build\_payload([keyword], cat=0, timeframe='today 3-m', geo='', gprop='')  related\_queries = pytrends.related\_queries()[keyword]['top']    if related\_queries is not None:  # 각 연관 검색어를 데이터프레임에 추가  for \_, row in related\_queries.iterrows():  new\_row = pd.DataFrame({'유튜버': [keyword], '연관 검색어': [row['query']]})  related\_queries\_df = pd.concat([related\_queries\_df, new\_row], ignore\_index=True)  print(related\_queries)  # 데이터프레임을 Excel 파일로 저장  related\_queries\_df.to\_excel('related\_queries.xlsx', index=False)  print("연관 검색어가 'related\_queries.xlsx'로 저장되었습니다.") |

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

**- Sql Database 생성과 웹 페이지와의 연동 –**

**Sql Database 연동**

|  |
| --- |
| # 데이터프레임의 각 행을 Youtuber 테이블에 삽입  for index, row in df.iterrows():  insert\_query = """  INSERT INTO VideoStat (video\_id, upload\_date, date, view\_count, like\_count, comment\_count, subscriber\_count, channel\_id, channel\_title, channel\_description, topic\_categories, title, description, tags, thumbnails)  VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)  """  data = (  row['video\_id'] if pd.notnull(row['video\_id']) else None,  row['upload\_date'] if pd.notnull(row['upload\_date']) else None,  row['date'] if pd.notnull(row['date']) else None,  row['view\_count'] if pd.notnull(row['view\_count']) else None,  row['like\_count'] if pd.notnull(row['like\_count']) else None,  row['comment\_count'] if pd.notnull(row['comment\_count']) else None,  row['subscriber\_count'] if pd.notnull(row['subscriber\_count']) else None,  row['channel\_id'] if pd.notnull(row['channel\_id']) else None,  row['channel\_title'] if pd.notnull(row['channel\_title']) else None,  row['channel\_description'] if pd.notnull(row['channel\_description']) else None,  row['topic\_categories'] if pd.notnull(row['topic\_categories']) else None,  row['title'] if pd.notnull(row['title']) else None,  row['description'] if pd.notnull(row['description']) else None,  row['tags'] if pd.notnull(row['tags']) else None,  row['thumbnails'] if pd.notnull(row['thumbnails']) else None  )  cursor.execute(insert\_query, data)  for index, row in df2.iterrows():  insert\_query = """  INSERT INTO Comments (comment, author, date, num\_likes, video\_id)  VALUES (%s, %s, %s, %s, %s)  """  data = (  row['comment'],  row['author'],  row['date'],  row['num\_likes'],  row['video\_id']  )  cursor.execute(insert\_query, data) |

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

**텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**자동 로그 수집**

|  |
| --- |
| VIDEO\_Oking = ['oakQvwCbvr8', '-HZeqsIgGHo', 'nAy-7zuCVQs', 'QojVuirFx58', 'ibI5OOZXSj8']  VIDEO\_STYLE = ['g5KDoSqT24Q', 'mnn1\_yu0aDQ', 'pp\_C0MGj9ZM', '\_Otk-iMD\_X0', '\_HZ63R-8z4E']  VIDEO\_GAME = ['yLlKOd3I8CA', 'BYWO-z-4tfo', 'uNq7RMRwIHs', 'ZLXz98YW\_U0', 'qkXc1M3d7g4']  VIDEO\_MUSIC = ['-rHqqCxjhM4', 'FGjrtBFgTXY', 'TOSrdHy5KRc', 'wdUu3XQK8cE', 'LamRCcz4zqg']  VIDEO\_ISSUE = ['ahcPfSLbT-M', '8l4GZ4datyM', '7I790Er-zkc', '8SJs1Cg7hpU', 'VWmWScovllY']  # File paths  STATS\_FILE\_PATH = '../xlsx/stats0709.xlsx'  COMMENTS\_FILE\_PATH = '../xlsx/crawling\_auto0709.xlsx'  extract(VIDEO\_Oking,STATS\_FILE\_PATH, COMMENTS\_FILE\_PATH)  extract(VIDEO\_STYLE,STATS\_FILE\_PATH, COMMENTS\_FILE\_PATH)  extract(VIDEO\_GAME,STATS\_FILE\_PATH, COMMENTS\_FILE\_PATH)  extract(VIDEO\_MUSIC,STATS\_FILE\_PATH, COMMENTS\_FILE\_PATH)  extract(VIDEO\_ISSUE,STATS\_FILE\_PATH, COMMENTS\_FILE\_PATH) |

**MySql과 Django Server 연동 및 웹 퍼블리싱**

|  |
| --- |
| # followSupportTest/models.py  from django.db import models  class VideoStat(models.Model):  video\_num = models.IntegerField(primary\_key=True)  video\_id = models.CharField(max\_length=50)  upload\_date = models.DateTimeField()  date = models.DateField()  view\_count = models.IntegerField()  like\_count = models.IntegerField()  comment\_count = models.IntegerField()  subscriber\_count = models.IntegerField()  channel\_id = models.CharField(max\_length=50)  channel\_title = models.CharField(max\_length=100)  channel\_description = models.TextField(null=True)  topic\_categories = models.TextField(null=True)  title = models.CharField(max\_length=200, null=True)  description = models.TextField(null=True)  tags = models.TextField(null=True)  thumbnails = models.CharField(max\_length=255)  class Meta:  managed = False  db\_table = 'VideoStat'  def \_\_str\_\_(self):  return self.video\_id  class Comment(models.Model):  comment\_id = models.IntegerField(primary\_key=True)  comment = models.TextField()  author = models.CharField(max\_length=100)  date = models.DateTimeField()  num\_likes = models.IntegerField()  video\_id = models.CharField(max\_length=100)  emotion = models.CharField(max\_length=100)  object = models.CharField(max\_length=100)  class Meta:  managed = False  db\_table = 'Comments'  def \_\_str\_\_(self):  return self.comment |

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

**텍스트, 스크린샷, 번호, 평행이(가) 표시된 사진

자동 생성된 설명**

**- 이미지 분석을 통한 신규 썸네일 및 프로필 생성 –**

**컨텐츠 썸네일 혹은 유튜브 이미지를 분석해 캡선 생성 및 저장**

|  |
| --- |
| import torch  from PIL import Image  from transformers import BlipProcessor, BlipForConditionalGeneration  import cv2  import matplotlib.pyplot as plt  # Hugging Face 모델 허브에서 blip 모델 로드  model\_name = "Salesforce/blip-image-captioning-base"  processor = BlipProcessor.from\_pretrained(model\_name)  model = BlipForConditionalGeneration.from\_pretrained(model\_name)  # 이미지 로드  image\_path = "../images/default.jpg"  image = Image.open(image\_path).convert("RGB")  # 입력 데이터 처리  inputs = processor(images=image, return\_tensors="pt")  # 모델 예측  outputs = model.generate(\*\*inputs)  caption = processor.decode(outputs[0], skip\_special\_tokens=True)  print(f"Generated Caption: {caption}")  def add\_keywords\_to\_caption(caption, keywords):  # 간단한 예시로, 키워드를 캡션 끝에 추가합니다.  new\_caption = caption + " " + " ".join(keywords)  return new\_caption  keywords = ["new", "keywords"]  modified\_caption = add\_keywords\_to\_caption(caption, keywords)  print(f"Modified Caption: {modified\_caption}")  def extract\_image\_features(image\_path):  image = cv2.imread(image\_path)  gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  # 여기서는 간단히 SIFT를 사용하여 주요 특징점을 추출합니다.  sift = cv2.SIFT\_create()  keypoints, descriptors = sift.detectAndCompute(gray, None)  return keypoints, descriptors  def print\_keypoints\_and\_descriptors(keypoints, descriptors):  print(f"Extracted {len(keypoints)} keypoints and descriptors.")    for i, kp in enumerate(keypoints):  print(f"Keypoint {i}:")  print(f" - pt: {kp.pt}")  print(f" - size: {kp.size}")  print(f" - angle: {kp.angle}")  print(f" - response: {kp.response}")  print(f" - octave: {kp.octave}")  print(f" - class\_id: {kp.class\_id}")  print(f"Descriptor {i}: {descriptors[i]}")  keypoints, descriptors = extract\_image\_features(image\_path)  print\_keypoints\_and\_descriptors(keypoints, descriptors) |

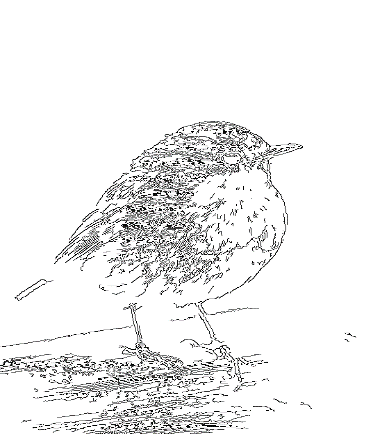
텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

**Pix2pix를 활용한 prompt기반 이미지 변경 및 생성**

|  |
| --- |
| import os  import requests  import sys  import copy  from tqdm import tqdm  import torch  from transformers import AutoTokenizer, CLIPTextModel  from diffusers import AutoencoderKL, UNet2DConditionModel  from diffusers.utils.peft\_utils import set\_weights\_and\_activate\_adapters  from peft import LoraConfig  p = "src/"  sys.path.append(p)  from model import make\_1step\_sched, my\_vae\_encoder\_fwd, my\_vae\_decoder\_fwd  class TwinConv(torch.nn.Module):  def \_\_init\_\_(self, convin\_pretrained, convin\_curr):  super(TwinConv, self).\_\_init\_\_()  self.conv\_in\_pretrained = copy.deepcopy(convin\_pretrained)  self.conv\_in\_curr = copy.deepcopy(convin\_curr)  self.r = None  def forward(self, x):  x1 = self.conv\_in\_pretrained(x).detach()  x2 = self.conv\_in\_curr(x)  return x1 \* (1 - self.r) + x2 \* (self.r)  class Pix2Pix\_Turbo(torch.nn.Module):  def \_\_init\_\_(self, pretrained\_name=None, pretrained\_path=None, ckpt\_folder="checkpoints", lora\_rank\_unet=8, lora\_rank\_vae=4):  super().\_\_init\_\_()  self.tokenizer = AutoTokenizer.from\_pretrained("stabilityai/sd-turbo", subfolder="tokenizer")  self.text\_encoder = CLIPTextModel.from\_pretrained("stabilityai/sd-turbo", subfolder="text\_encoder").cuda()  self.sched = make\_1step\_sched()  vae = AutoencoderKL.from\_pretrained("stabilityai/sd-turbo", subfolder="vae")  vae.encoder.forward = my\_vae\_encoder\_fwd.\_\_get\_\_(vae.encoder, vae.encoder.\_\_class\_\_)  vae.decoder.forward = my\_vae\_decoder\_fwd.\_\_get\_\_(vae.decoder, vae.decoder.\_\_class\_\_)  # add the skip connection convs  vae.decoder.skip\_conv\_1 = torch.nn.Conv2d(512, 512, kernel\_size=(1, 1), stride=(1, 1), bias=False).cuda()  vae.decoder.skip\_conv\_2 = torch.nn.Conv2d(256, 512, kernel\_size=(1, 1), stride=(1, 1), bias=False).cuda()  vae.decoder.skip\_conv\_3 = torch.nn.Conv2d(128, 512, kernel\_size=(1, 1), stride=(1, 1), bias=False).cuda()  vae.decoder.skip\_conv\_4 = torch.nn.Conv2d(128, 256, kernel\_size=(1, 1), stride=(1, 1), bias=False).cuda()  vae.decoder.ignore\_skip = False  unet = UNet2DConditionModel.from\_pretrained("stabilityai/sd-turbo", subfolder="unet")  if pretrained\_name == "edge\_to\_image":  url = "https://www.cs.cmu.edu/~img2img-turbo/models/edge\_to\_image\_loras.pkl"  os.makedirs(ckpt\_folder, exist\_ok=True)  outf = os.path.join(ckpt\_folder, "edge\_to\_image\_loras.pkl")  if not os.path.exists(outf):  print(f"Downloading checkpoint to {outf}")  response = requests.get(url, stream=True)  total\_size\_in\_bytes = int(response.headers.get('content-length', 0))  block\_size = 1024 # 1 Kibibyte  progress\_bar = tqdm(total=total\_size\_in\_bytes, unit='iB', unit\_scale=True)  with open(outf, 'wb') as file:  for data in response.iter\_content(block\_size):  progress\_bar.update(len(data))  file.write(data)  progress\_bar.close()  if total\_size\_in\_bytes != 0 and progress\_bar.n != total\_size\_in\_bytes:  print("ERROR, something went wrong")  print(f"Downloaded successfully to {outf}")  p\_ckpt = outf  sd = torch.load(p\_ckpt, map\_location="cpu")  unet\_lora\_config = LoraConfig(r=sd["rank\_unet"], init\_lora\_weights="gaussian", target\_modules=sd["unet\_lora\_target\_modules"])  vae\_lora\_config = LoraConfig(r=sd["rank\_vae"], init\_lora\_weights="gaussian", target\_modules=sd["vae\_lora\_target\_modules"])  vae.add\_adapter(vae\_lora\_config, adapter\_name="vae\_skip")  \_sd\_vae = vae.state\_dict()  for k in sd["state\_dict\_vae"]:  \_sd\_vae[k] = sd["state\_dict\_vae"][k]  vae.load\_state\_dict(\_sd\_vae)  unet.add\_adapter(unet\_lora\_config)  \_sd\_unet = unet.state\_dict()  for k in sd["state\_dict\_unet"]:  \_sd\_unet[k] = sd["state\_dict\_unet"][k]  unet.load\_state\_dict(\_sd\_unet)  elif pretrained\_name == "sketch\_to\_image\_stochastic":  # download from url  url = "https://www.cs.cmu.edu/~img2img-turbo/models/sketch\_to\_image\_stochastic\_lora.pkl"  os.makedirs(ckpt\_folder, exist\_ok=True)  outf = os.path.join(ckpt\_folder, "sketch\_to\_image\_stochastic\_lora.pkl")  if not os.path.exists(outf):  print(f"Downloading checkpoint to {outf}")  response = requests.get(url, stream=True)  total\_size\_in\_bytes = int(response.headers.get('content-length', 0))  block\_size = 1024 # 1 Kibibyte  progress\_bar = tqdm(total=total\_size\_in\_bytes, unit='iB', unit\_scale=True)  with open(outf, 'wb') as file:  for data in response.iter\_content(block\_size):  progress\_bar.update(len(data))  file.write(data)  progress\_bar.close()  if total\_size\_in\_bytes != 0 and progress\_bar.n != total\_size\_in\_bytes:  print("ERROR, something went wrong")  print(f"Downloaded successfully to {outf}")  p\_ckpt = outf  convin\_pretrained = copy.deepcopy(unet.conv\_in)  unet.conv\_in = TwinConv(convin\_pretrained, unet.conv\_in)  sd = torch.load(p\_ckpt, map\_location="cpu")  unet\_lora\_config = LoraConfig(r=sd["rank\_unet"], init\_lora\_weights="gaussian", target\_modules=sd["unet\_lora\_target\_modules"])  vae\_lora\_config = LoraConfig(r=sd["rank\_vae"], init\_lora\_weights="gaussian", target\_modules=sd["vae\_lora\_target\_modules"])  vae.add\_adapter(vae\_lora\_config, adapter\_name="vae\_skip")  \_sd\_vae = vae.state\_dict()  for k in sd["state\_dict\_vae"]:  \_sd\_vae[k] = sd["state\_dict\_vae"][k]  vae.load\_state\_dict(\_sd\_vae)  unet.add\_adapter(unet\_lora\_config)  \_sd\_unet = unet.state\_dict()  for k in sd["state\_dict\_unet"]:  \_sd\_unet[k] = sd["state\_dict\_unet"][k]  unet.load\_state\_dict(\_sd\_unet)  elif pretrained\_path is not None:  sd = torch.load(pretrained\_path, map\_location="cpu")  unet\_lora\_config = LoraConfig(r=sd["rank\_unet"], init\_lora\_weights="gaussian", target\_modules=sd["unet\_lora\_target\_modules"])  vae\_lora\_config = LoraConfig(r=sd["rank\_vae"], init\_lora\_weights="gaussian", target\_modules=sd["vae\_lora\_target\_modules"])  vae.add\_adapter(vae\_lora\_config, adapter\_name="vae\_skip")  \_sd\_vae = vae.state\_dict()  for k in sd["state\_dict\_vae"]:  \_sd\_vae[k] = sd["state\_dict\_vae"][k]  vae.load\_state\_dict(\_sd\_vae)  unet.add\_adapter(unet\_lora\_config)  \_sd\_unet = unet.state\_dict()  for k in sd["state\_dict\_unet"]:  \_sd\_unet[k] = sd["state\_dict\_unet"][k]  unet.load\_state\_dict(\_sd\_unet)  elif pretrained\_name is None and pretrained\_path is None:  print("Initializing model with random weights")  torch.nn.init.constant\_(vae.decoder.skip\_conv\_1.weight, 1e-5)  torch.nn.init.constant\_(vae.decoder.skip\_conv\_2.weight, 1e-5)  torch.nn.init.constant\_(vae.decoder.skip\_conv\_3.weight, 1e-5)  torch.nn.init.constant\_(vae.decoder.skip\_conv\_4.weight, 1e-5)  target\_modules\_vae = ["conv1", "conv2", "conv\_in", "conv\_shortcut", "conv", "conv\_out",  "skip\_conv\_1", "skip\_conv\_2", "skip\_conv\_3", "skip\_conv\_4",  "to\_k", "to\_q", "to\_v", "to\_out.0",  ]  vae\_lora\_config = LoraConfig(r=lora\_rank\_vae, init\_lora\_weights="gaussian",  target\_modules=target\_modules\_vae)  vae.add\_adapter(vae\_lora\_config, adapter\_name="vae\_skip")  target\_modules\_unet = [  "to\_k", "to\_q", "to\_v", "to\_out.0", "conv", "conv1", "conv2", "conv\_shortcut", "conv\_out",  "proj\_in", "proj\_out", "ff.net.2", "ff.net.0.proj"  ]  unet\_lora\_config = LoraConfig(r=lora\_rank\_unet, init\_lora\_weights="gaussian",  target\_modules=target\_modules\_unet  )  unet.add\_adapter(unet\_lora\_config)  self.lora\_rank\_unet = lora\_rank\_unet  self.lora\_rank\_vae = lora\_rank\_vae  self.target\_modules\_vae = target\_modules\_vae  self.target\_modules\_unet = target\_modules\_unet  # unet.enable\_xformers\_memory\_efficient\_attention()  unet.to("cuda")  vae.to("cuda")  self.unet, self.vae = unet, vae  self.vae.decoder.gamma = 1  self.timesteps = torch.tensor([999], device="cuda").long()  self.text\_encoder.requires\_grad\_(False)  def set\_eval(self):  self.unet.eval()  self.vae.eval()  self.unet.requires\_grad\_(False)  self.vae.requires\_grad\_(False)  def set\_train(self):  self.unet.train()  self.vae.train()  for n, \_p in self.unet.named\_parameters():  if "lora" in n:  \_p.requires\_grad = True  self.unet.conv\_in.requires\_grad\_(True)  for n, \_p in self.vae.named\_parameters():  if "lora" in n:  \_p.requires\_grad = True  self.vae.decoder.skip\_conv\_1.requires\_grad\_(True)  self.vae.decoder.skip\_conv\_2.requires\_grad\_(True)  self.vae.decoder.skip\_conv\_3.requires\_grad\_(True)  self.vae.decoder.skip\_conv\_4.requires\_grad\_(True)  def forward(self, c\_t, prompt=None, prompt\_tokens=None, deterministic=True, r=1.0, noise\_map=None):  # either the prompt or the prompt\_tokens should be provided  assert (prompt is None) != (prompt\_tokens is None), "Either prompt or prompt\_tokens should be provided"  if prompt is not None:  # encode the text prompt  caption\_tokens = self.tokenizer(prompt, max\_length=self.tokenizer.model\_max\_length,  padding="max\_length", truncation=True, return\_tensors="pt").input\_ids.cuda()  caption\_enc = self.text\_encoder(caption\_tokens)[0]  else:  caption\_enc = self.text\_encoder(prompt\_tokens)[0]  if deterministic:  encoded\_control = self.vae.encode(c\_t).latent\_dist.sample() \* self.vae.config.scaling\_factor  model\_pred = self.unet(encoded\_control, self.timesteps, encoder\_hidden\_states=caption\_enc,).sample  x\_denoised = self.sched.step(model\_pred, self.timesteps, encoded\_control, return\_dict=True).prev\_sample  self.vae.decoder.incoming\_skip\_acts = self.vae.encoder.current\_down\_blocks  output\_image = (self.vae.decode(x\_denoised / self.vae.config.scaling\_factor).sample).clamp(-1, 1)  else:  # scale the lora weights based on the r value  self.unet.set\_adapters(["default"], weights=[r])  set\_weights\_and\_activate\_adapters(self.vae, ["vae\_skip"], [r])  encoded\_control = self.vae.encode(c\_t).latent\_dist.sample() \* self.vae.config.scaling\_factor  # combine the input and noise  unet\_input = encoded\_control \* r + noise\_map \* (1 - r)  self.unet.conv\_in.r = r  unet\_output = self.unet(unet\_input, self.timesteps, encoder\_hidden\_states=caption\_enc,).sample  self.unet.conv\_in.r = None  x\_denoised = self.sched.step(unet\_output, self.timesteps, unet\_input, return\_dict=True).prev\_sample  self.vae.decoder.incoming\_skip\_acts = self.vae.encoder.current\_down\_blocks  self.vae.decoder.gamma = r  output\_image = (self.vae.decode(x\_denoised / self.vae.config.scaling\_factor).sample).clamp(-1, 1)  return output\_image  def save\_model(self, outf):  sd = {}  sd["unet\_lora\_target\_modules"] = self.target\_modules\_unet  sd["vae\_lora\_target\_modules"] = self.target\_modules\_vae  sd["rank\_unet"] = self.lora\_rank\_unet  sd["rank\_vae"] = self.lora\_rank\_vae  sd["state\_dict\_unet"] = {k: v for k, v in self.unet.state\_dict().items() if "lora" in k or "conv\_in" in k}  sd["state\_dict\_vae"] = {k: v for k, v in self.vae.state\_dict().items() if "lora" in k or "skip" in k}  torch.save(sd, outf) |

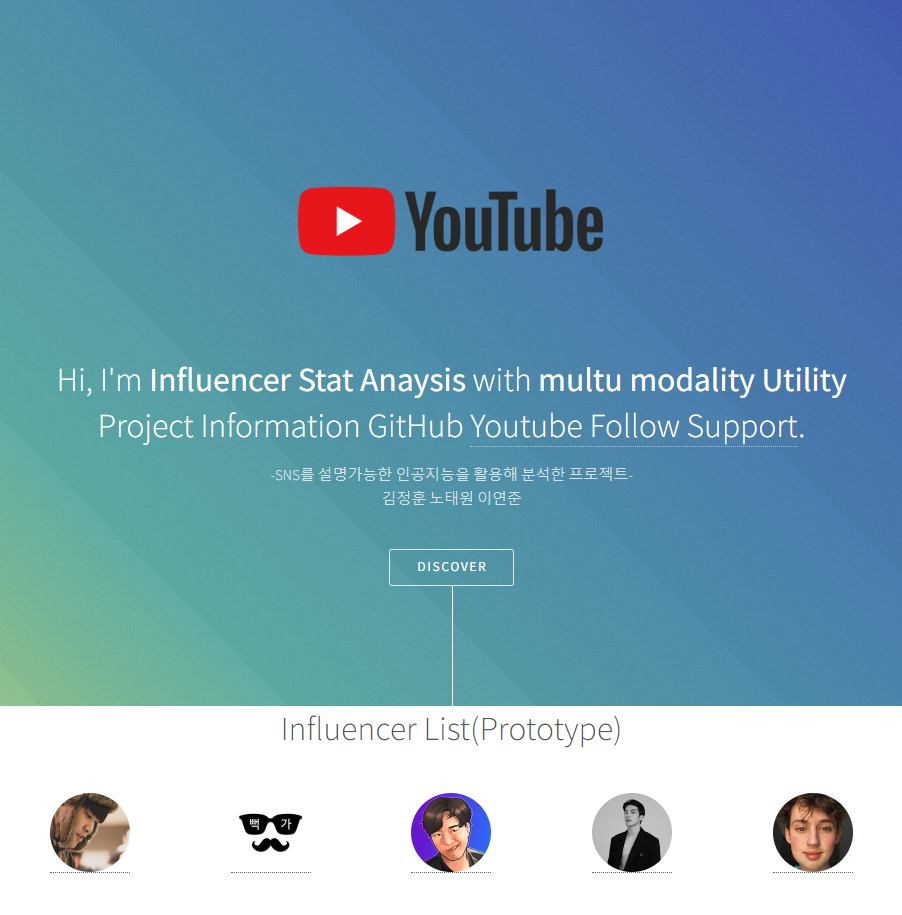
**왼쪽부터 오른쪽 순으로 prompt 기반으로 변경된 사진 모습**



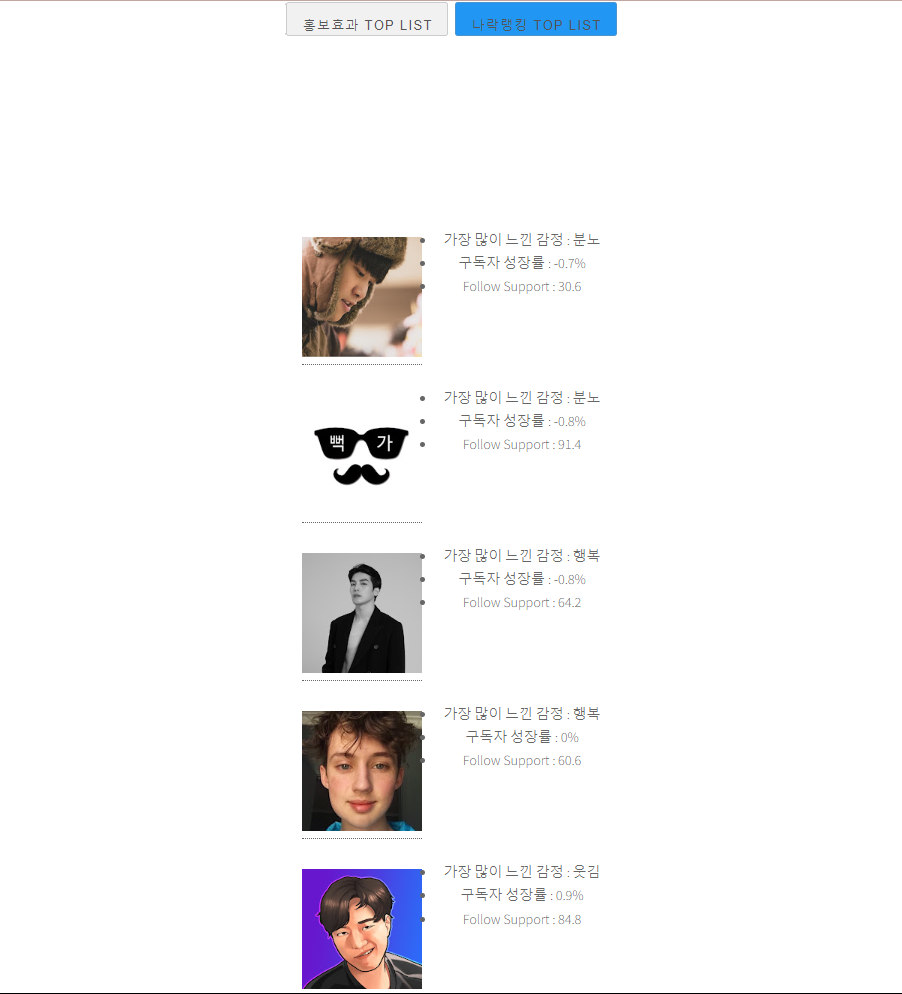
**4. 최종 결과물**

**- 웹 페이지 결과물 -**

**메인 홈 화면**

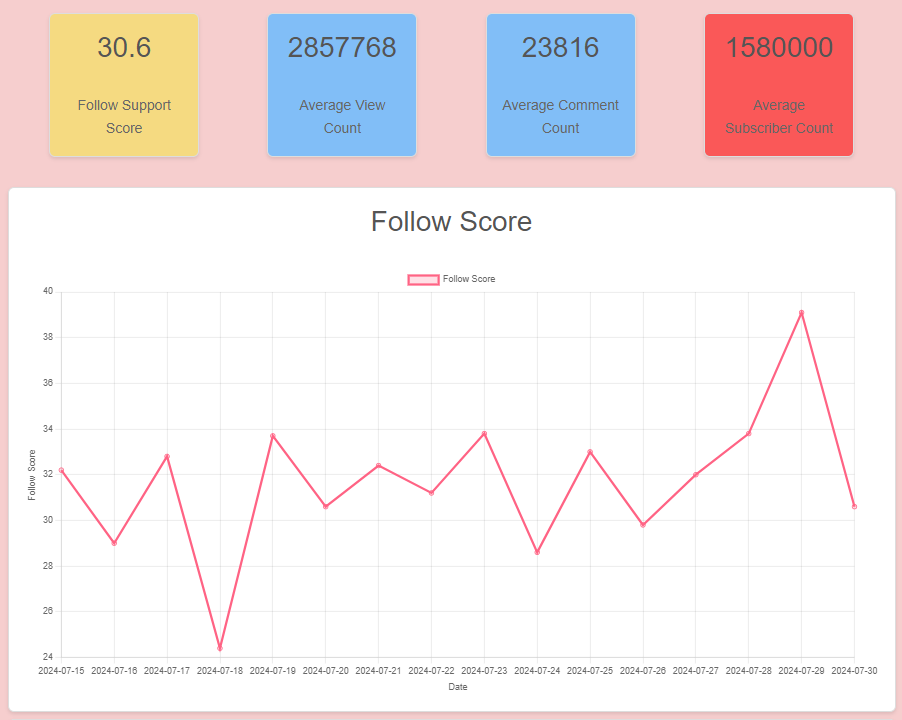


**랭킹화면**

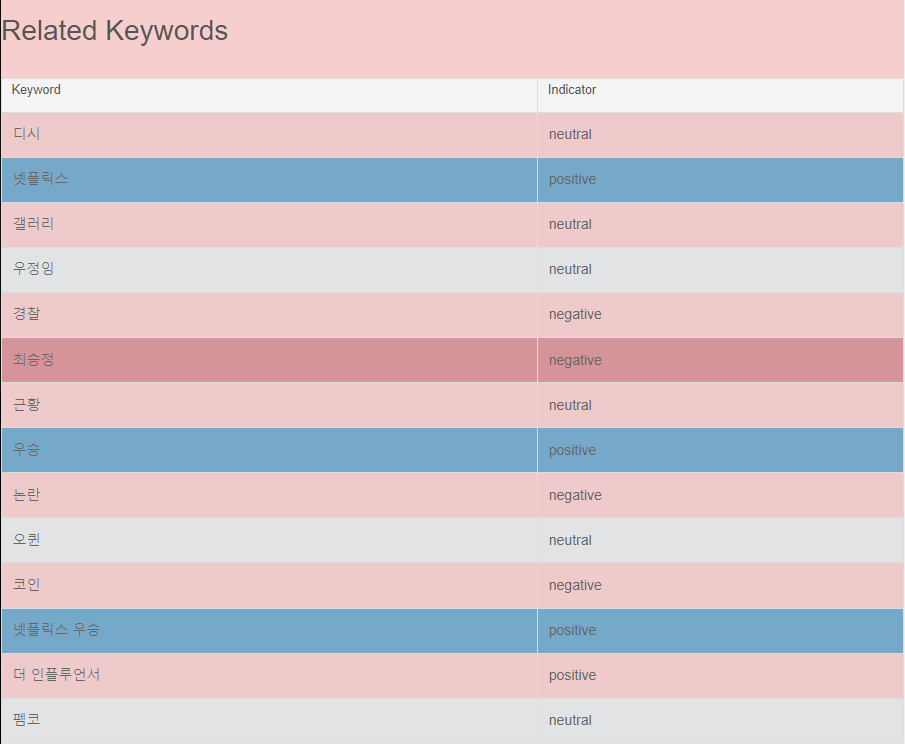
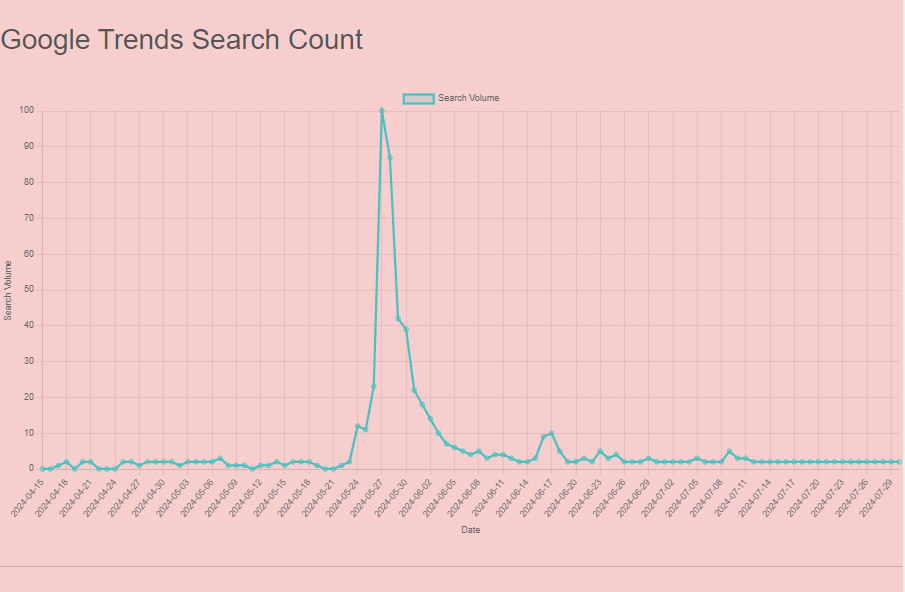


**대시보드 및 그래프 화면**

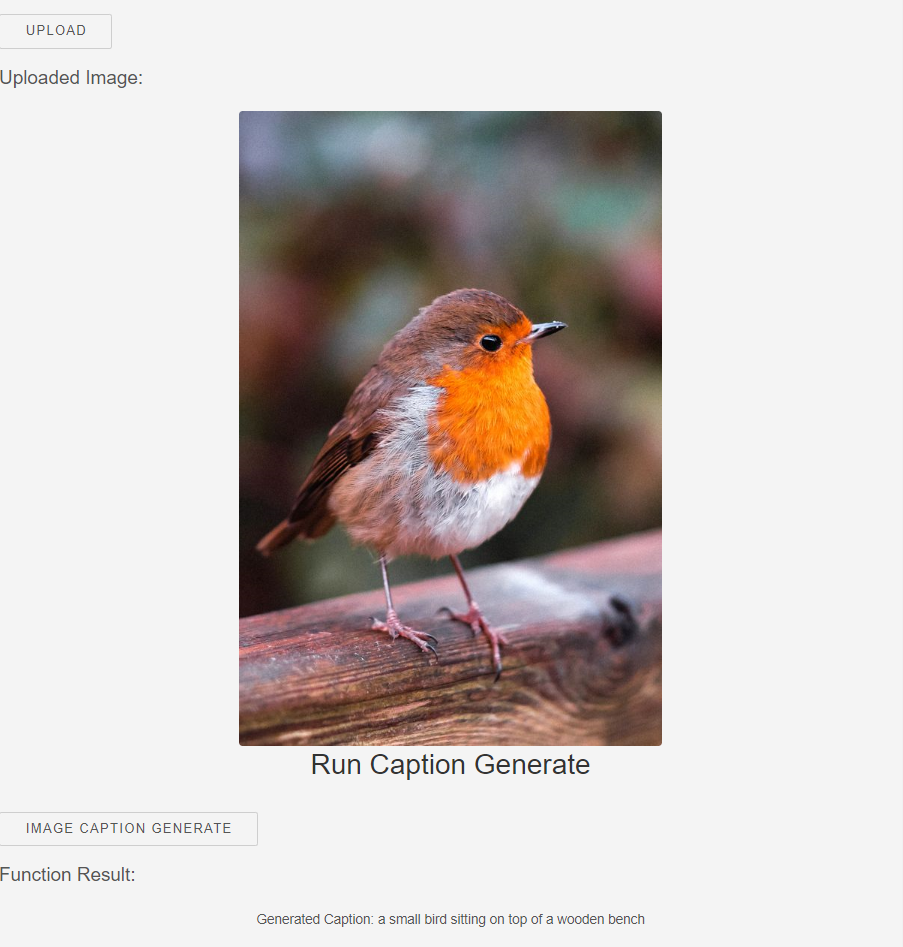


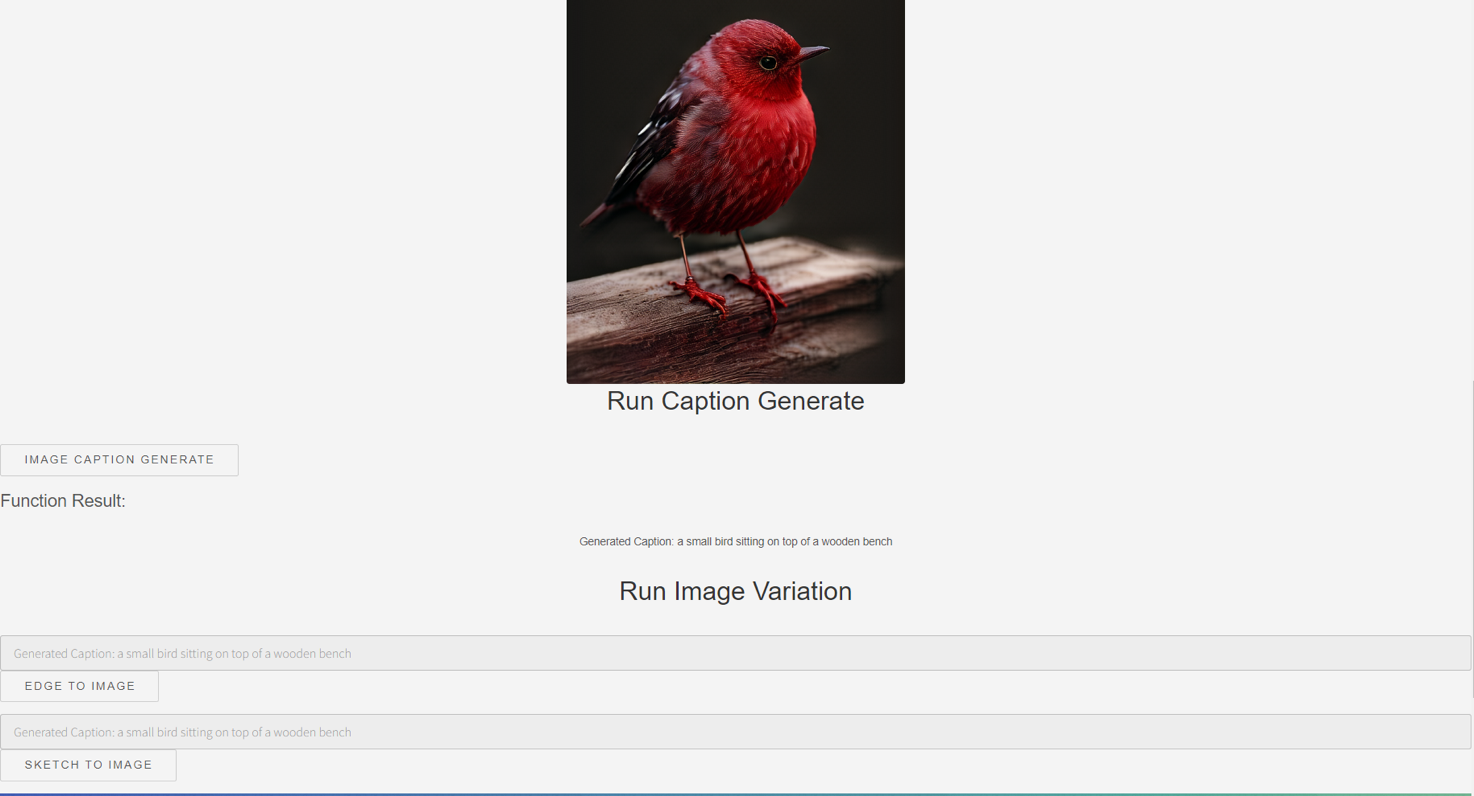


**Google trend 및 연관검색어 화면**



**이미지 캡션 및 Generate 화면**





**시현 영상**

인플루언서 스탯분석에 관한 발표 영상

<https://www.youtube.com/watch?v=e9kM0qYI3K4&t>

이미지 캡션 모듈 연동 추가후 추가 발표 영상

<https://www.youtube.com/watch?v=w0yP7oYiSQo>

**5. 자기평가**

|  |
| --- |
| 학기중 수강한 과목을 넘어서 새롭게 적용하고 개발환경에서 접하는 부분이 많았다.  텍스트 마이닝 기법을 실 개발 환경에 적용함에 있어서 학습한 내용 이상의 전처리 과정이 요구되었고 이를 적절한 API와 라이브러리를 활용하여 유의미한 데이터셋으로 발전시켜 나가게 하는 과정이 좋은 학습과 경험이 되었다.  텍스트 형식의 데이터를 수치화 하여 감정분석을 진행하는 과정에서 벡터화 및 감정분석 정확도 측면에서 어려웠던 부분이 많았는데 이를 팀 리더와 상의하며 보다 더 어렵지만 끝끝내 원하는 정확도를 도출해내도록 학습시킨는 모델을 도출해 내었을때 개발자로서 한단계 성장하였다.  개발현장에서 제일 많이 사용되는 데이터베이스로 MySql을 꼽을 수 있는데 이를 통한 전처리 과정 및 연동과정에 대해 직접 참여하면서 웹 페이지와 데이터베이스 연동과정에 대해 심도 있게 이해하고 활용할 수 있게 되었다.  여러 API와 라이브러리를 2차적으로 활용하면서 이에 따른 추가 기능 및 활용이 가능하다고 판단되어 이를 이행하면서 새로운 개발 관점에 대해 생각보게 되었다. 완전히 새로운 API 및 라이브러리 개발 보다는 기존 개발 레퍼런스를 보다 더 발전시키는 방향이 유익하고 효율적이라는 관점도 습득하였다.  이미지 캡션 및 영상 메타데이터 추출 및 요약과정을 진행함에 있어서 AI 모듈을 활용하고 이를 적용시키는 관점에 있어서 모듈 알고리즘 및 input과 output설정이 중요함을 인지하였다. 모듈이 생성되고 활용될때 개발자의 역량이 그대로 반영되는 부분이라고 생각되어 신중하고 목표치에 맞는 함수와 라이브러리를 적절히 사용해야하는 부분이라고 느꼈다.  프로젝트 참여에 있어서 외부적인 다른 요소를 고려하지 않고 가벼운 마음으로 접근하는것은 좋지 않다는것을 인지하였다. 개발에 있어서 다른 외부요인들이 개발 및 프로젝트 참여를 지연시킨다면 이 또한 실패한 개발자라 생각이 들었다. 외부요인 및 본인의 건강, 스케줄, 시간을 통제하고 활용하는것 또한 성장한 개발자가 갖춰야할 덕목과 자질이라고 생각한다. |

**6. 향후계획**

|  |
| --- |
| 이번 프로젝트를 진행함에 있어서 개발자로서 많은 경험을 하였다.  AI 모듈적용부터 텍스트마이닝, DB 와 웹 연동, API 활용, 모듈 파인튜닝, 이미지 캡션 등  실생활 및 서비스에 근접한 개발 기술들을 습득하였으며 이를 통해 추후 캡스톤 및 프로젝트에 쉽게 적용할 수 있도록 진입장벽이 낮아졌다.  이러한 성과를 기반으로 추후 프로젝트에서 AI 적용 방향이 그리 어렵지 않음을 인지하였으니, AI를 접목시킨 다양한 서비스기반 아이디어 선정에 무리가 없을 거라는 생각이 들었다.  또한 웹 서버와 Database연동과정역시 쉽지 않았지만 습득하였고 이를 통해 앱 및 웹 개발에 있어서 Database 연동을 쉽게 진행할 수 있을거 같다. 이렇게 된다면 웹 디자인 및 앱 디자인과 서버, DB연동 또한 모두 가능한 개발자로 성장하여 1인 개발자로서 부족함이 없도록 더욱 더 공부할 예정이다.  Django 서버를 사용하여 웹 개발을 진행하였지만 추후 NodeJs, Spring boot 등을 활용한 서버 개발분야에 대한 추가적인 학습이 필요하다고 생각되었다.  빅데이터를 다루는 과정에서 전처리 과정에 대한 중요점을 다시한번 인지하게 되었고 이를 통한 모듈과 서비스에 대한 정확도 역시 중요하다고 생각되었다. 이를 접목시킨 전처리 과정에 간소화 및 정확성을 어떻게 하면 올릴수 있을지에 대한 전반적인 이해와 심도있는 공부에 대한 필요성을 느끼고 2024학년도 2학기 3학년 수업인 데이터베이스 설계를 수강하며 이러한 부분들을 적용해보려 한다. |