



금융핀 BackLog - 하얀

▼ fetch_news.py 참고 예시

검색 > 블로그 - Search API

검색 > 블로그 블로그 검색 개요 개요 사전 준비 사항 블로그 검색 API 레퍼런스 블로그 검색 결과 조회 오류 코드 검색 API 블로그 검색 구현 예제 Java PHP Node.js Python C# 블로그 검색 개요 개요 사전 준비 사항 개요 검색 API와 블

<https://developers.naver.com/docs/serviceapi/search/blog/blog.md#python>

2025.01.12.(일)

LCTon

app7pm.py

- 날짜와 키워드를 분명하게 하면 출처가 제대로 나옴.
 - 그렇지 않은 경우 요약 내용은 나오나, 내용과 출처가 불일치함
- 하나의 기사에 대한 요약만 나오는 문제가 있음.

```
import streamlit as st
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI
from langchain_community.embeddings import OpenAIEmbeddings
from langchain_community.vectorstores import Chroma
import openai
from fetch_news import fetch_naver_news
import os
from dotenv import load_dotenv
from datetime import datetime
from bs4 import BeautifulSoup
import re

# 환경 변수 로드
load_dotenv()

# API 키 읽어오기
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
if OPENAI_API_KEY:
    print("API 키 로드 성공")
else:
    print("API 키 로드 실패")

# OpenAI 모델 설정
embedding_model = OpenAIEmbeddings(model="text-embedding-ada-002", api_key=OPENAI_API_KEY)
llm = ChatOpenAI(model="gpt-3.5-turbo", api_key=OPENAI_API_KEY)

# Chroma에 persist_directory 경로 지정
persist_directory = "./chroma_db"

# Streamlit 앱 제목
st.title("금융 뉴스 요약 챗봇 🗨️")
st.write("챗봇에 질문을 입력하면 최신 금융 뉴스를 요약하여 답변해드립니다.")

# 세션 상태 초기화
if "messages" not in st.session_state:
    st.session_state["messages"] = []

# 최초 접속 시 인사말 추가
if len(st.session_state.messages) == 0:
    st.session_state["messages"].append({"role": "assistant", "content": "안녕하세요! 질문을 입력해 주세요."})

# 이전 메시지 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

# 사용자 입력 받기
user_input = st.chat_input("질문을 입력하세요 (예: Tesla, Bitcoin, 또는 특정 날짜):")
```

```

# 날짜 추출 함수
def extract_date(text):
    """입력된 텍스트에서 날짜를 추출합니다. (YYYY년 MM월 DD일 형식)"""
    match = re.search(r"(\d{4})년 (\d{1,2})월 (\d{1,2})일", text)
    if match:
        return datetime.strptime(match.group(), "%Y년 %m월 %d일")
    return None

# 키워드 추출 함수 (특정 단어 찾기)
def extract_keyword(text):
    """입력된 텍스트에서 키워드를 추출합니다."""
    keywords = ['Tesla', 'Bitcoin', 'Stock', '경제', '금융', '주식', '테슬라', '비트코인']
    for keyword in keywords:
        if keyword.lower() in text.lower():
            return keyword
    return None

# HTML 태그를 제거하는 함수
def clean_html(text):
    """HTML 태그를 제거합니다."""
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

# 기사 내용을 청크로 나누는 함수
def chunk_text(text, chunk_size=500):
    """기사 내용을 500자 이하로 청크로 나눕니다."""
    chunks = [text[i:i + chunk_size] for i in range(0, len(text), chunk_size)]
    return chunks

if user_input:
    st.session_state.messages.append({"role": "user", "content": user_input})

    # 날짜와 키워드 추출
    date = extract_date(user_input)
    keyword = extract_keyword(user_input)

    # 기본 쿼리 생성
    query = user_input if not keyword else keyword

    with st.spinner("네이버에서 뉴스 가져오는 중..."):
        try:
            # 뉴스 검색
            news_items = fetch_naver_news(query, display=10)

            # 날짜 필터링 (사용자가 입력한 날짜에 해당하는 기사만 선택)
            filtered_news = []
            if date:
                for item in news_items:
                    pub_date = datetime.strptime(item["pubDate"], "%a, %d %b %Y %H:%M:%S %z")
                    if pub_date.date() == date.date():
                        filtered_news.append(item)
            else:
                filtered_news = news_items

            if not filtered_news:
                st.warning("지정된 날짜나 쿼리에 맞는 뉴스가 없습니다.")
            else:
                # 뉴스 내용 전처리 및 벡터화
                documents = []
                sources = set() # 출처 중복 제거를 위한 set 사용
                for item in filtered_news:
                    clean_article = clean_html(item["description"])
                    chunks = chunk_text(clean_article)
                    for chunk in chunks:
                        documents.append({"content": chunk, "source": item["originallink"]})
                    sources.add(item["originallink"]) # 출처 추가

                texts = [doc["content"] for doc in documents]
                metadatas = [{"source": doc["source"]} for doc in documents]

                # Chroma 벡터스토어 생성, persist_directory 경로 설정

```

```

vectorstore = Chroma.from_texts(texts, embedding_model, persist_directory=persist_directory, metadatas=metadatas)
retriever = vectorstore.as_retriever()

# QA 체인 설정
qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever, return_source_documents=True)

# 쿼리 처리
result = qa_chain(query)
answer = result["result"]

# 챗봇 형식의 응답 출력
st.session_state.messages.append({"role": "assistant", "content": answer})

# 출처는 한 번만 출력
if sources:
    st.session_state.messages.append({"role": "assistant", "content": "📖 출처"})
    # 하나의 출처만 추가되도록 처리
    source_list = list(sources)
    st.session_state.messages.append({"role": "assistant", "content": f"- [더 읽기]({source_list[0]})"})

except Exception as e:
    st.session_state.messages.append({"role": "assistant", "content": f"오류 발생: {e}"})

# 대화 업데이트
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

```

LCTon_0112night

appTFIDF4oMulti.py

1. 사용자의 입력에서 키워드를 자동 추출해주는 spaCy 라이브러리 사용
2. 인식이 안될 경우 → 특정 키워드 리스트인 `predefined_keywords` 에 있는 키워드로 감지
3. 모두 없을 경우 → 결과가 없다는 결론으로..!

- 청크의 경우 500자는 다소 짧다보니 요약이 잘 안되는 문제가 있었음
 - 청크를 1000자로 수정
- 날짜 + 키워드로 질문할 경우에 가장 좋은 성능을 보임
- GPT 3.5 Turbo → GPT 4o-mini로 모델 변경
 - v4의 성능이 더 좋은 이유 : 기사의 경우, 다른 텍스트들이 그렇듯 문맥이 중요한 텍스트이기 때문에 문맥적 요소를 잘 유지하는 모델이 v4이기 때문일 것으로 추정
 - 또한, 대답의 일관성이 개선되었음
 - https://seo.tbwakorea.com/blog/gpt-4-and-7-elements-of-improvement/?utm_source=chatgpt.com
 - <https://brunch.co.kr/%40cozycanvas/125>

```

# appTFIDF4oMulti.py(250112 오후 10시 56분 ver)

import streamlit as st
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI
from langchain_community.embeddings import OpenAIEmbeddings
from langchain_community.vectorstores import Chroma
import openai
from fetch_news import fetch_naver_news
import os
from dotenv import load_dotenv
from datetime import datetime
from bs4 import BeautifulSoup
import re
import spacy
from spacy.matcher import PhraseMatcher

# 환경 변수 로드
load_dotenv()

```

```

# API 키 읽어오기
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
if OPENAI_API_KEY:
    print("API 키 로드 성공")
else:
    print("API 키 로드 실패")

# OpenAI 모델 설정
embedding_model = OpenAIEmbeddings(model="text-embedding-ada-002", api_key=OPENAI_API_KEY)
llm = ChatOpenAI(model="gpt-4o-mini", api_key=OPENAI_API_KEY) # gpt-4 모델 사용

# Chroma에 persist_directory 경로 지정
persist_directory = "./chroma_db"

# Streamlit 앱 제목
st.title("금융 뉴스 요약 챗봇 🗨️")
st.write("챗봇에 질문을 입력하면 최신 금융 뉴스를 요약하여 답변해드립니다.")

# 세션 상태 초기화
if "messages" not in st.session_state:
    st.session_state["messages"] = []

# 최초 접속 시 인사말 추가
if len(st.session_state.messages) == 0:
    st.session_state["messages"].append({"role": "assistant", "content": "안녕하세요! 질문을 입력해 주세요."})

# 이전 메시지 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

# 사용자 입력 받기
user_input = st.chat_input("질문을 입력하세요 (예: Tesla, Bitcoin, 또는 특정 날짜):")

# spaCy 모델 로드 (한국어)
nlp = spacy.load("ko_core_news_sm")

# 특정 키워드 리스트
predefined_keywords = [
    'Tesla', 'Bitcoin', 'Stock', '경제', '금융', '주식', '테슬라', '비트코인',
    '금리', '채권', '주식시장', '증권', '주식거래', 'ETF', '포트폴리오', '펀드',
    '주식투자', '가상화폐', '크립토', '블록체인', '상장', '코스피', '코스닥',
    '상장폐지', '상장주식', '배당', '주식배당', '시가총액', '이자율', '자산',
    '자산운용', '리스크관리', '채권시장', '헤지펀드', '투자전략', '경제지표',
    '물가', '소비자물가', '환율', '금융위기', '금융정책', '금융시스템', '국채',
    '지수', '상승률', '하락률', '증시', '매수', '매도', '자산관리', '고배당주',
    '금융상품', '부동산', '모기지', '대출', '채권금리', '금융기관', '거래소',
    '리보금리', '금융규제', 'FOMC', 'IMF', 'OECD', 'GDP', '실업률', '인플레이션',
    '유동성', '마진', '헤지', '옵션', '선물', '주식분석', '기업분석', '매출', '순이익',
    '영업이익', '비즈니스모델', '가치투자', '성장투자', '워런버핏', '테크주', '그로스',
    '인덱스펀드', '투자자', '주요지표', '저금리', '금융사기', '핀테크', '모바일뱅킹',
    '디지털자산', '핀테크기업', '금융기술', '블록체인기술', '디지털화폐', '리브라',
    '스테이블코인', '대체투자', '경기지표', '증권사', '금융컨설팅', '고정금리',
    '변동금리', '국제금융', '금융분석', '경제위기', '경제성장', '고용지표', '상장기업',
    '투자기회', '정책금리', '기준금리', '금융거래', '가치주', '성장주', '신용카드',
    '해외주식', '자본시장', '중앙은행', '금융업계', '회계기준', '기업회계', '회계사'
]

# 사용자 입력에서 키워드를 추출하는 함수
def extract_keyword(text):
    """사용자 입력에서 키워드를 추출합니다."""
    doc = nlp(text)

    # PhraseMatcher를 사용하여 미리 정의된 키워드를 문장에서 찾기
    matcher = PhraseMatcher(nlp.vocab)
    patterns = [nlp.make_doc(keyword) for keyword in predefined_keywords]
    matcher.add("PredefinedKeywords", patterns)

    matches = matcher(doc)
    matched_keywords = [doc[start:end].text for _, start, end in matches]

    # 키워드가 있으면 추출하고, 없으면 None 반환
    if matched_keywords:

```

```

        return matched_keywords[0] # 첫 번째 키워드 반환
    else:
        return None

# 날짜 추출 함수
def extract_date(text):
    """입력된 텍스트에서 날짜를 추출합니다. (YYYY년 MM월 DD일 형식)"""
    match = re.search(r"(\d{4})년 (\d{1,2})월 (\d{1,2})일", text)
    if match:
        return datetime.strptime(match.group(), "%Y년 %m월 %d일")
    return None

# HTML 태그를 제거하는 함수
def clean_html(text):
    """HTML 태그를 제거합니다."""
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

# 기사 내용을 청크로 나누는 함수
def chunk_text(text, chunk_size=1000):
    """기사 내용을 1000자 이하로 청크로 나눕니다."""
    chunks = [text[i:i + chunk_size] for i in range(0, len(text), chunk_size)]
    return chunks

if user_input:
    st.session_state.messages.append({"role": "user", "content": user_input})

    # 날짜와 키워드 추출
    date = extract_date(user_input)
    keyword = extract_keyword(user_input)

    # 기본 쿼리 생성
    query = user_input if not keyword else keyword

    with st.spinner("네이버에서 뉴스 가져오는 중..."):
        try:
            # 뉴스 검색
            news_items = fetch_naver_news(query, display=10)

            # 날짜 필터링 (사용자가 입력한 날짜에 해당하는 기사만 선택)
            filtered_news = []
            if date:
                for item in news_items:
                    pub_date = datetime.strptime(item["pubDate"], "%a, %d %b %Y %H:%M:%S %z")
                    if pub_date.date() == date.date():
                        filtered_news.append(item)
            else:
                filtered_news = news_items

            if not filtered_news:
                st.warning("지정된 날짜나 쿼리에 맞는 뉴스가 없습니다.")
            else:
                # 뉴스 내용 전처리 및 벡터화
                documents = []
                summaries = [] # 요약 리스트 저장
                sources = [] # 출처 리스트 저장

                for item in filtered_news:
                    clean_article = clean_html(item["description"]) # HTML 태그 제거 후 내용 정리
                    chunks = chunk_text(clean_article)
                    for chunk in chunks:
                        documents.append({"content": chunk, "source": item["originallink"]})
                    summaries.append(clean_html(item["description"])) # HTML 태그 제거 후 요약 추가
                    sources.append(item["originallink"]) # 출처 추가

                texts = [doc["content"] for doc in documents]
                metadatas = [{"source": doc["source"]} for doc in documents]

                # Chroma 벡터스토어 생성, persist_directory 경로 설정
                vectorstore = Chroma.from_texts(
                    texts,
                    embedding_model,

```

```

        persist_directory=persist_directory,
        metadatas=metadatas
    )
    retriever = vectorstore.as_retriever()

# QA 체인 설정
qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever, return_source_documents=True)

# 쿼리 처리
result = qa_chain(query)
answer = result["result"]

# 챗봇 형식의 응답 출력
st.session_state.messages.append({"role": "assistant", "content": answer})

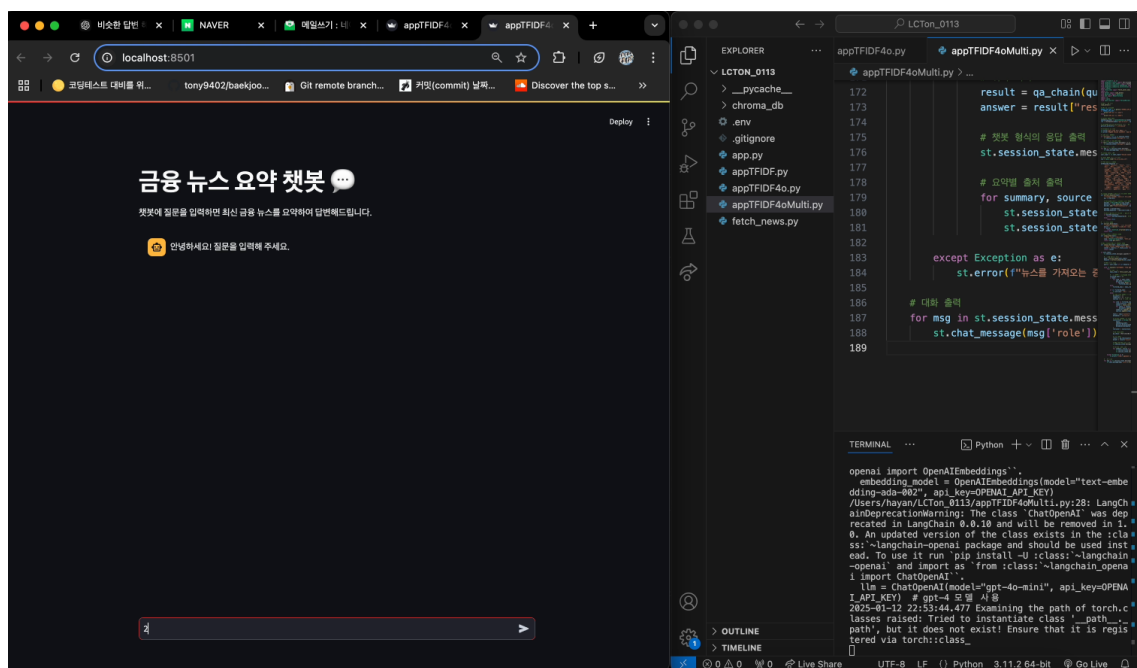
# 요약별 출처 출력
for summary, source in zip(summaries, sources):
    st.session_state.messages.append({"role": "assistant", "content": f"**요약**: {summary}"})
    st.session_state.messages.append({"role": "assistant", "content": f"**출처**: [원문 보기]({source})"})

except Exception as e:
    st.error(f"뉴스를 가져오는 중 오류가 발생했습니다: {e}")

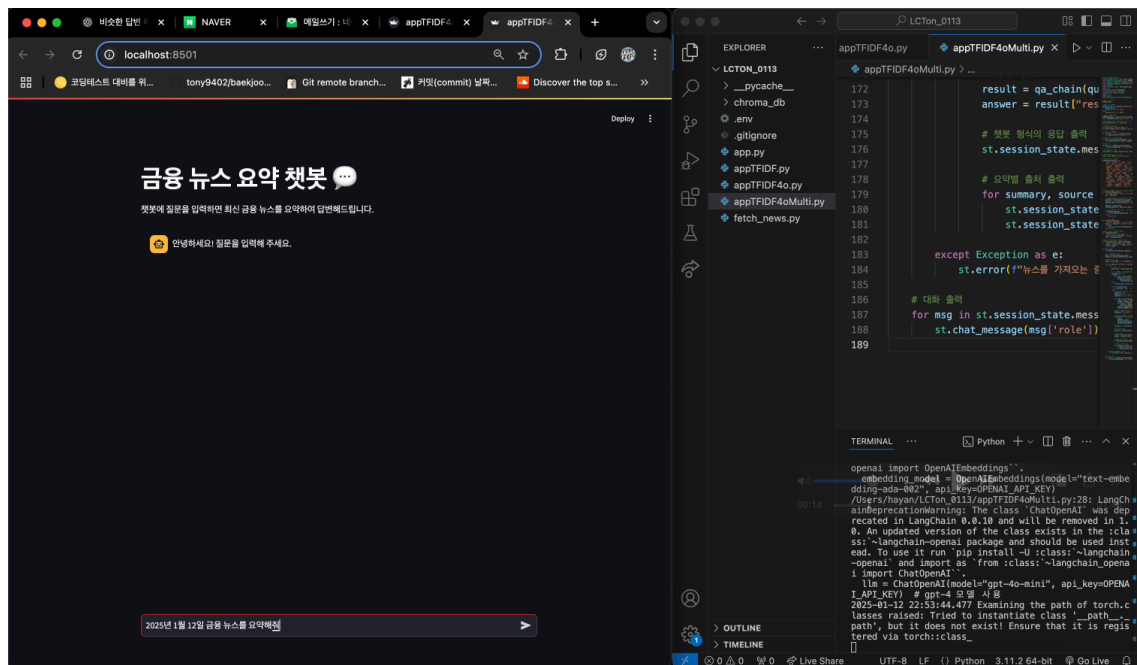
# 대화 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

```

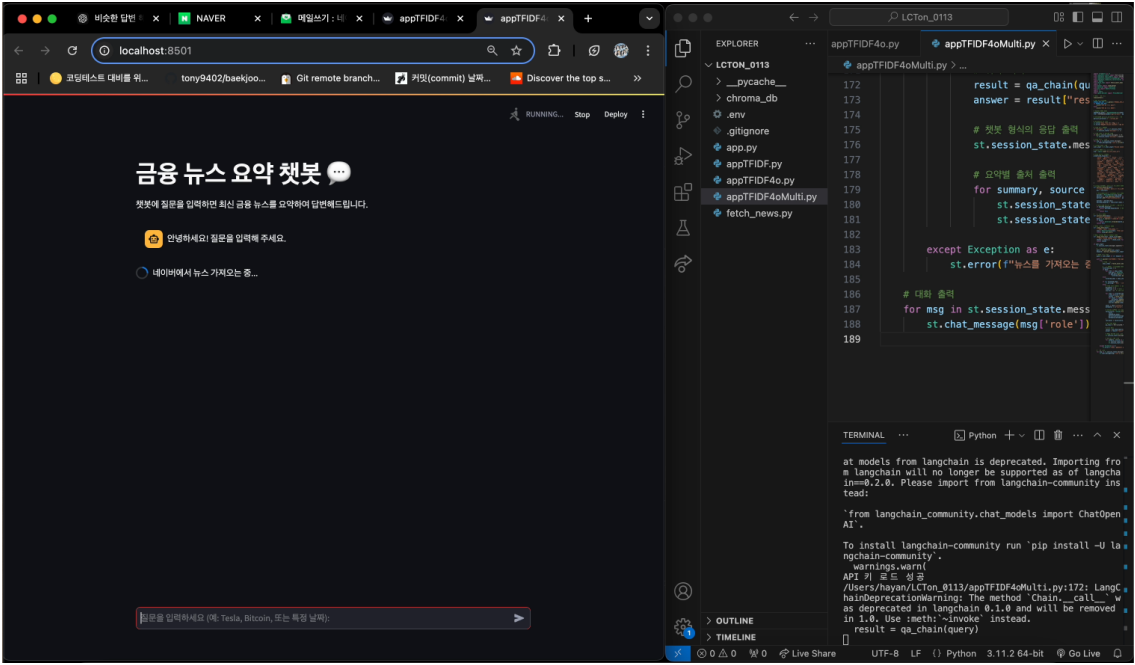
- 초기 화면



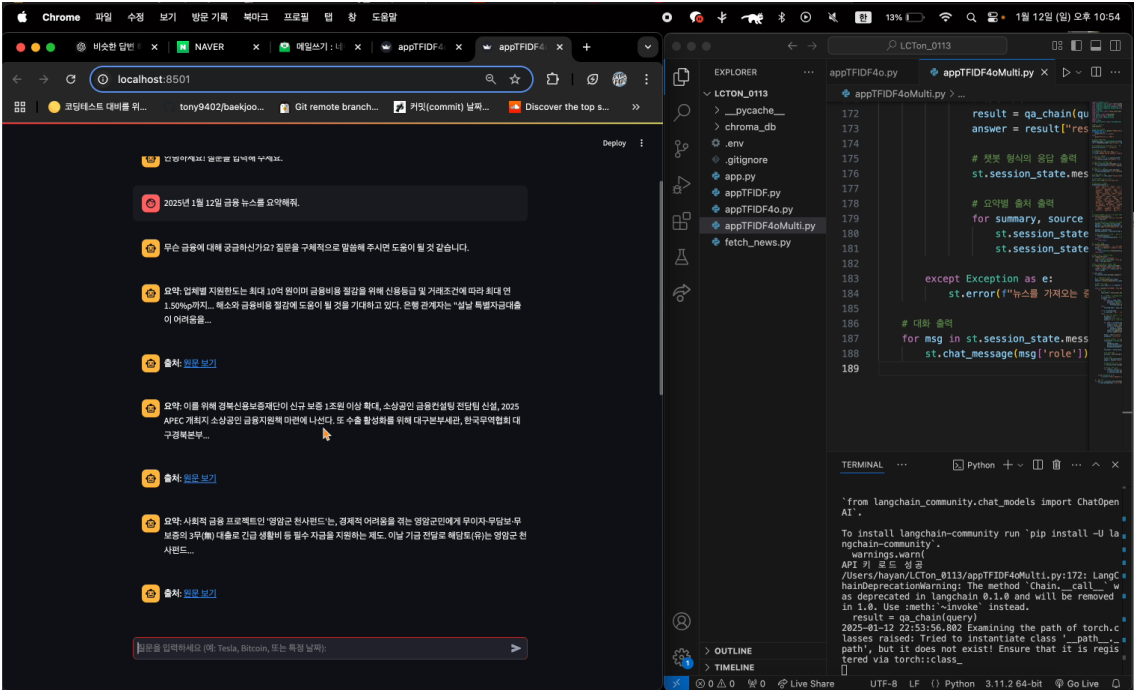
- 질문



- 답변 받는 중



- 답변 완료
 - 각 요약과 그 출처를 밝힘
 - 다만, 요약이 그대로 가져와 자르는 형태로 진행되는 문제가 있음



2025.01.13 (월)

LCTon_0113

appTFIDF4oMulti.py

- 날짜를 기반으로 입력받을 경우, 당일 날짜만 가능
 - 그 이유 : 네이버 검색 API가 가져오는 쿼리 한도가 작게 설정되어 있음.

appTFIDF4oMulti.py(250112 오후 10시 56분 ver)

```
import streamlit as st
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI
from langchain_community.embeddings import OpenAIEmbeddings
from langchain_community.vectorstores import Chroma
import openai
from fetch_news import fetch_naver_news
import os
from dotenv import load_dotenv
from datetime import datetime
from bs4 import BeautifulSoup
```



```

import re
import spacy
from spacy.matcher import PhraseMatcher

# 환경 변수 로드
load_dotenv()

# API 키 읽어오기
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
if OPENAI_API_KEY:
    print("API 키 로드 성공")
else:
    print("API 키 로드 실패")

# OpenAI 모델 설정
embedding_model = OpenAIEmbeddings(model="text-embedding-ada-002", api_key=OPENAI_API_KEY)
llm = ChatOpenAI(model="gpt-4o-mini", api_key=OPENAI_API_KEY) # gpt-4 모델 사용

# Chroma에 persist_directory 경로 지정
persist_directory = "./chroma_db"

# Streamlit 앱 제목
st.title("금융 뉴스 요약 챗봇 🗨️")
st.write("챗봇에 질문을 입력하면 최신 금융 뉴스를 요약하여 답변해드립니다.")

# 세션 상태 초기화
if "messages" not in st.session_state:
    st.session_state["messages"] = []

# 최초 접속 시 인사말 추가
if len(st.session_state.messages) == 0:
    st.session_state["messages"].append({"role": "assistant", "content": "안녕하세요! 질문을 입력해 주세요."})

# 이전 메시지 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

# 사용자 입력 받기
user_input = st.chat_input("질문을 입력하세요 (예: Tesla, Bitcoin, 또는 특정 날짜):")

# spaCy 모델 로드 (한국어)
nlp = spacy.load("ko_core_news_sm")

# 특정 키워드 리스트
predefined_keywords = [
    'Tesla', 'Bitcoin', 'Stock', '경제', '금융', '주식', '테슬라', '비트코인',
    '금리', '채권', '주식시장', '증권', '주식거래', 'ETF', '포트폴리오', '펀드',
    '주식투자', '가상화폐', '크립토', '블록체인', '상장', '코스피', '코스닥',
    '상장폐지', '상장주식', '배당', '주식배당', '시가총액', '이자율', '자산',
    '자산운용', '리스크관리', '채권시장', '헤지펀드', '투자전략', '경제지표',
    '물가', '소비자물가', '환율', '금융위기', '금융정책', '금융시스템', '국채',
    '지수', '상승률', '하락률', '증시', '매수', '매도', '자산관리', '고배당주',
    '금융상품', '부동산', '모기지', '대출', '채권금리', '금융기관', '거래소',
    '리보금리', '금융규제', 'FOMC', 'IMF', 'OECD', 'GDP', '실업률', '인플레이션',
    '유동성', '마진', '헤지', '옵션', '선물', '주식분석', '기업분석', '매출', '순이익',
    '영업이익', '비즈니스모델', '가치투자', '성장투자', '워런버핏', '테크주', '그로쓰',
    '인덱스펀드', '투자자', '주요지표', '저금리', '금융사기', '핀테크', '모바일뱅킹',
    '디지털자산', '핀테크기업', '금융기술', '블록체인기술', '디지털화폐', '리브라',
    '스테이블코인', '대체투자', '경기지표', '증권사', '금융컨설팅', '고정금리',
    '변동금리', '국제금융', '금융분석', '경제위기', '경제성장', '고용지표', '상장기업',
    '투자기회', '정책금리', '기준금리', '금융거래', '가치주', '성장주', '신용카드',
    '해외주식', '자본시장', '중앙은행', '금융업계', '회계기준', '기업회계', '회계사'
]

# 사용자 입력에서 키워드를 추출하는 함수
def extract_keyword(text):
    """사용자 입력에서 키워드를 추출합니다."""
    doc = nlp(text)

    # PhraseMatcher를 사용하여 미리 정의된 키워드를 문장에서 찾기
    matcher = PhraseMatcher(nlp.vocab)
    patterns = [nlp.make_doc(keyword) for keyword in predefined_keywords]

```



```

matcher.add("PredefinedKeywords", patterns)

matches = matcher(doc)
matched_keywords = [doc[start:end].text for _, start, end in matches]

# 키워드가 있으면 추출하고, 없으면 None 반환
if matched_keywords:
    return matched_keywords[0] # 첫 번째 키워드 반환
else:
    return None

# 날짜 추출 함수
def extract_date(text):
    """입력된 텍스트에서 날짜를 추출합니다. (YYYY년 MM월 DD일 형식)"""
    match = re.search(r"(\d{4})년 (\d{1,2})월 (\d{1,2})일", text)
    if match:
        return datetime.strptime(match.group(), "%Y년 %m월 %d일")
    return None

# HTML 태그를 제거하는 함수
def clean_html(text):
    """HTML 태그를 제거합니다."""
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

# 기사 내용을 청크로 나누는 함수
def chunk_text(text, chunk_size=1000):
    """기사 내용을 1000자 이하로 청크로 나눕니다."""
    chunks = [text[i:i + chunk_size] for i in range(0, len(text), chunk_size)]
    return chunks

if user_input:
    st.session_state.messages.append({"role": "user", "content": user_input})

    # 날짜와 키워드 추출
    date = extract_date(user_input)
    keyword = extract_keyword(user_input)

    # 기본 쿼리 생성
    query = user_input if not keyword else keyword

    with st.spinner("네이버에서 뉴스 가져오는 중..."):
        try:
            # 뉴스 검색
            news_items = fetch_naver_news(query, display=10)

            # 날짜 필터링 (사용자가 입력한 날짜에 해당하는 기사만 선택)
            filtered_news = []
            if date:
                for item in news_items:
                    pub_date = datetime.strptime(item["pubDate"], "%a, %d %b %Y %H:%M:%S %z")
                    if pub_date.date() == date.date():
                        filtered_news.append(item)
            else:
                filtered_news = news_items

            if not filtered_news:
                st.warning("지정된 날짜나 쿼리에 맞는 뉴스가 없습니다.")
            else:
                # 뉴스 내용 전처리 및 벡터화
                documents = []
                summaries = [] # 요약 리스트 저장
                sources = [] # 출처 리스트 저장

                for item in filtered_news:
                    clean_article = clean_html(item["description"]) # HTML 태그 제거 후 내용 정리
                    chunks = chunk_text(clean_article)
                    for chunk in chunks:
                        documents.append({"content": chunk, "source": item["originallink"]})
                    summaries.append(clean_html(item["description"])) # HTML 태그 제거 후 요약 추가
                    sources.append(item["originallink"]) # 출처 추가

```

```

texts = [doc["content"] for doc in documents]
metadatas = [{"source": doc["source"]} for doc in documents]

# Chroma 벡터스토어 생성, persist_directory 경로 설정
vectorstore = Chroma.from_texts(
    texts,
    embedding_model,
    persist_directory=persist_directory,
    metadatas=metadatas
)
retriever = vectorstore.as_retriever()

# QA 체인 설정
qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever, return_source_documents=True)

# 쿼리 처리
result = qa_chain(query)
answer = result["result"]

# 챗봇 형식의 응답 출력
st.session_state.messages.append({"role": "assistant", "content": answer})

# 요약별 출처 출력
for summary, source in zip(summaries, sources):
    st.session_state.messages.append({"role": "assistant", "content": f"**요약**: {summary}"})
    st.session_state.messages.append({"role": "assistant", "content": f"**출처**: [원문 보기]({source})"})

except Exception as e:
    st.error(f"뉴스를 가져오는 중 오류가 발생했습니다: {e}")

# 대화 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

```

<https://drive.google.com/file/d/165PLRztHt3keyT5PRVt83BTSAiwNDEpJ/view?usp=sharing>

TroubleShooting

네이버 검색 API가 가져오는 쿼리 한도가 작게 설정되어 있는 문제

- fetch_news.py에서 display를 10 → 1000으로 올려보는 시도

```

import requests
import os
from dotenv import load_dotenv

load_dotenv()

NAVER_CLIENT_ID = os.getenv("NAVER_CLIENT_ID")
NAVER_CLIENT_SECRET = os.getenv("NAVER_CLIENT_SECRET")

print(f"Client ID: {NAVER_CLIENT_ID}")
print(f"Client Secret: {NAVER_CLIENT_SECRET}")

def fetch_naver_news(query, display=1000): # 이 부분!
    url = "https://openapi.naver.com/v1/search/news.json"
    headers = {
        "X-Naver-Client-Id": NAVER_CLIENT_ID,
        "X-Naver-Client-Secret": NAVER_CLIENT_SECRET

```

```

    }
    params = {
        "query": query,
        "display": display,
        "sort": "date"
    }
    response = requests.get(url, headers=headers, params=params)
    if response.status_code == 200:
        return response.json()["items"]
    else:
        raise Exception(f"Error {response.status_code}: {response.text}")

```

- 그러나, 실시간으로 계속해서 기사가 쏟아져나오기 때문에, 동일하게 당일의 아닌 경우 제대로 가져오지 못하는 것으로 보임
 - 정확한 문제점 : 현재의 벡터 DB는 임시 저장되는 형태로 사용되고 있고 이로 인해, 데이터가 충분히 쌓이지 못함 → display를 아주 크게 늘린다고 하더라도 쏟아져 나오는 기사의 양을 감당하기는 어려움 → 이전 날짜의 데이터의 경우, 결국 저장되지 않아 날짜 필터링이 제대로 됨에도 나올 수가 없는 것!
 - 해결 방안 : DB를 MySQL 등과 같은 데이터베이스로 변경을 한다면, 더 많은 데이터들을 저장해두고, 계속 쌓아가면서 사용할 수 있을 것으로 보임

fetch_news.py에서 sort 기준 변경

- sort 종류
 - **sim** : 정확도순으로 내림차순 정렬(기본값)
 - **date** : 날짜순으로 내림차순 정렬
- sort 기준을 date → sim으로 변경 + display도 100,000으로 크게 설정!

```

import requests
import os
from dotenv import load_dotenv

load_dotenv()

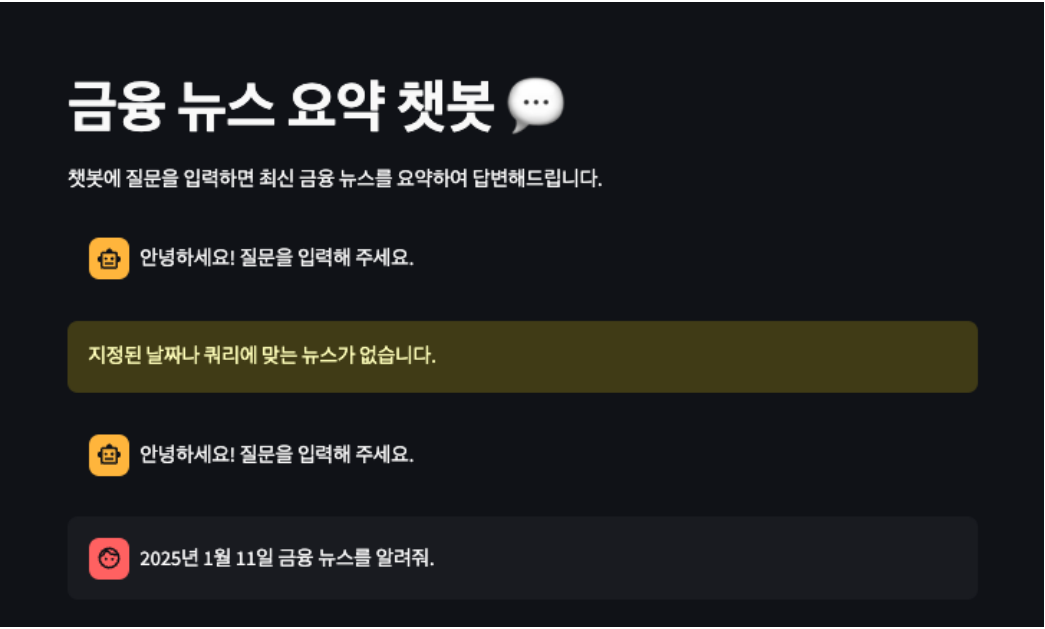
NAVER_CLIENT_ID = os.getenv("NAVER_CLIENT_ID")
NAVER_CLIENT_SECRET = os.getenv("NAVER_CLIENT_SECRET")

print(f"Client ID: {NAVER_CLIENT_ID}")
print(f"Client Secret: {NAVER_CLIENT_SECRET}")

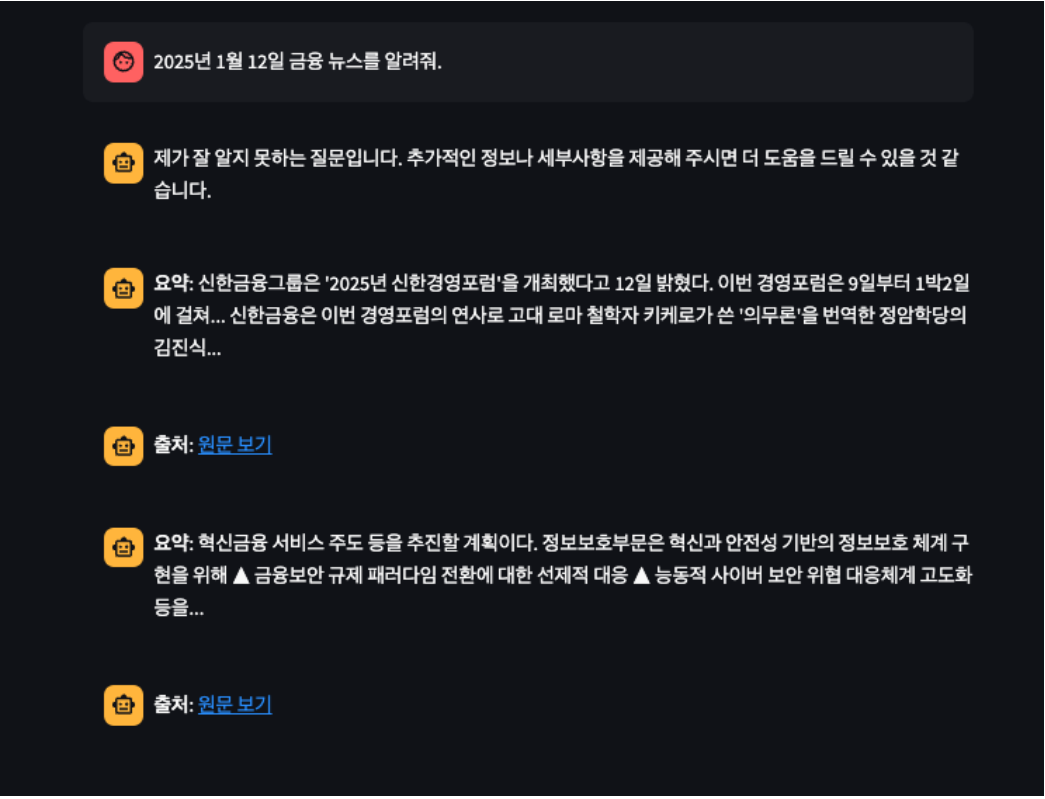
def fetch_naver_news(query, display=100):
    url = "https://openapi.naver.com/v1/search/news.json"
    headers = {
        "X-Naver-Client-Id": NAVER_CLIENT_ID,
        "X-Naver-Client-Secret": NAVER_CLIENT_SECRET
    }
    params = {
        "query": query,
        "display": display,
        "sort": "sim"
    }
    response = requests.get(url, headers=headers, params=params)
    if response.status_code == 200:
        return response.json()["items"]
    else:
        raise Exception(f"Error {response.status_code}: {response.text}")

```


- 결과
 - 당일과 그 전날까지는 최대한 끌어오게 됨.
 - 2일 이상의 날짜는 가져오지 못함(쿼리 한계로 보임)
- display를 100으로 줄였음에도 → 유사한 결과
 - “2025년 1월 11일 금융 뉴스를 알려줘.” → 2일전





◦ “2025년 1월 12일 금융 뉴스를 알려줘.” → 1일전




◦ “2025년 1월 13일 금융 뉴스를 알려줘.” → 당일


 2025년 1월 13일 금융 뉴스를 알려줘.


 이 질문에 대한 구체적인 답변을 제공하기 위해서는 더 많은 정보가 필요합니다. "금융"이라는 주제에 대해 어떤 내용을 알고 싶으신가요?


 요약: 17조원을 금융지원할 계획이라고 13일 밝혔다. 농협은행에 따르면 신용보증기금, 기술보증기금... 고객과의 동반성장"이라며 "국가경제의 기반인 중소기업에 실질적 도움이 되는 금융지원을 지속해나갈 것"이라고 강조했다.


 출처: [원문 보기](#)


 요약: 신한금융지주 회장은 “목표는 무엇을 해야 하는지, 목적은 왜 해야 하는지를 알려준다”며 “구성원이 목표보다 목적에 공감할 때 ‘일류(一流) 신한’에 더 가까워질 것”이라고 밝혔다. 12일 신한금융지주에...


 출처: [원문 보기](#)


 요약: 가수 G-DRAGON(지드래곤)이 하나금융그룹의 새로운 얼굴로 발탁됐다. 하나금융그룹 측은 “세대를 아우르는 시대의 아이콘인 G-DRAGON의 이미지가 ‘하나로 연결된 모두의 금융’이라는 그룹의 비전 아래 ‘하나’...


 출처: [원문 보기](#)

 요약: 가수 지드래곤(G-DRAGON)이 하나금융그룹의 새로운 얼굴로 발탁됐다. 지드래곤 하나금융그룹 측은 13일 “세대를 아우르는 시대의 아이콘인 지드래곤의 이미지가 ‘하나로 연결된 모두의 금융’이라는 그룹의 비전 아래...


 출처: [원문 보기](#)


 요약: 농협은 디지털금융 시스템 개편 작업에 따라 설 연휴 중인 오는 28~29일 전자금융서비스를 일시 중단한다고 13일 밝혔다. 서비스 중단 기간 농협 개인·기업 디지털채널을 통한 금융거래가 모두 중단된다. 곡뱅크...


 출처: [원문 보기](#)


 요약: 가수 지드래곤(G-DRAGON)이 하나금융그룹의 새로운 얼굴로 발탁됐다. 하나금융그룹 1월 13일 “세대를 아우르는 시대의 아이콘인 G-DRAGON의 이미지가 ‘하나로 연결된 모두의 금융’이라는 그룹의 비전 아래 ‘하나’만의...

 출처: [원문 보기](#)

 요약: 가수 지드래곤(G-DRAGON)이 하나금융그룹의 새로운 얼굴로 발탁됐다. 하나금융그룹 측은 13일 “세대를 아우르는 시대의 아이콘인 지드래곤의 이미지가 ‘하나로 연결된 모두의 금융’이라는 그룹의 비전 아래...

 출처: [원문 보기](#)

 요약: 메리츠증권은 한국주택금융공사로부터 2024년 국내 주택저당증권(MBS·Mortgage Backed Securities) 발행 최우수 주관사로 선정됐다고 13일 밝혔다. 지난해 발행된 주택금융공사 MBS 16조4600억원 중 2조 1000억원(12.7%)을...

 출처: [원문 보기](#)

<https://drive.google.com/file/d/1twFbVzykZsX4IDIWA1XoqS2saezOERo-/view?usp=sharing>

Test

▼ 질문 케이스 정리

- Q. 현재 가장 최근 금융 뉴스는 뭐야? - 실시간 반영이 잘되는지 테스트하는 항목
- Q. 2025년 1월 13일 금융 뉴스를 알려줘. - 특정 날짜를 잘 반영하는지 테스트하는 항목
- Q. 2025년 1월 12일의 금융 뉴스를 알려줄래?
- Q. 2025년 1월 11일의 금융 뉴스를 알려줄래?

첫 번째 케이스. fetch_news.py의 sort 파라미터를 date로 설정

- fetch_news.py

```
import requests
import os
from dotenv import load_dotenv

load_dotenv()

NAVER_CLIENT_ID = os.getenv("NAVER_CLIENT_ID")
NAVER_CLIENT_SECRET = os.getenv("NAVER_CLIENT_SECRET")

print(f"Client ID: {NAVER_CLIENT_ID}")
print(f"Client Secret: {NAVER_CLIENT_SECRET}")

def fetch_naver_news(query, display=100):
    url = "https://openapi.naver.com/v1/search/news.json"
    headers = {
        "X-Naver-Client-Id": NAVER_CLIENT_ID,
        "X-Naver-Client-Secret": NAVER_CLIENT_SECRET
    }
    params = {
        "query": query,
        "display": display,
        "sort": "date"
    }
    response = requests.get(url, headers=headers, params=params)
    if response.status_code == 200:
        return response.json()["items"]
    else:
        raise Exception(f"Error {response.status_code}: {response.text}")
```

- appV2.py

```
# appV2.py

import streamlit as st
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI
from langchain_community.embeddings import OpenAIEmbeddings
from langchain_community.vectorstores import Chroma
import openai
from fetch_news import fetch_naver_news
import os
from dotenv import load_dotenv
from datetime import datetime
from bs4 import BeautifulSoup
import re
import spacy
from spacy.matcher import PhraseMatcher

# 환경 변수 로드
load_dotenv()

# API 키 읽어오기
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
if OPENAI_API_KEY:
```

```

        print("API 키 로드 성공")
    else:
        print("API 키 로드 실패")

# OpenAI 모델 설정
embedding_model = OpenAIEmbeddings(model="text-embedding-ada-002", api_key=OPENAI_API_KEY)
llm = ChatOpenAI(model="gpt-4o-mini", api_key=OPENAI_API_KEY) # gpt-4 모델 사용

# Chroma에 persist_directory 경로 지정
persist_directory = "./chroma_db"

# Streamlit 앱 제목
st.title("금융 뉴스 요약 챗봇 😊")
st.write("챗봇에 질문을 입력하면 최신 금융 뉴스를 요약하여 답변해드립니다!")

# 세션 상태 초기화
if "messages" not in st.session_state:
    st.session_state["messages"] = []

# 최초 접속 시 인사말 추가
if len(st.session_state.messages) == 0:
    st.session_state["messages"].append({"role": "assistant", "content": "안녕하세요! 질문을 입력해 주세요."})

# 이전 메시지 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

# 사용자 입력 받기
user_input = st.chat_input("질문을 입력해주세요 😊(ex. 2025년 1월 13일의 금융 뉴스를 알려줄래?, 최신 금융 동향을 알려줘, ...)")

# spaCy 모델 로드 (한국어)
nlp = spacy.load("ko_core_news_sm")

# 특정 키워드 리스트
predefined_keywords = [
    'Tesla', 'Bitcoin', 'Stock', '경제', '금융', '주식', '테슬라', '비트코인',
    '금리', '채권', '주식시장', '증권', '주식거래', 'ETF', '포트폴리오', '펀드',
    '주식투자', '가상화폐', '크립토', '블록체인', '상장', '코스피', '코스닥',
    '상장폐지', '상장주식', '배당', '주식배당', '시가총액', '이자율', '자산',
    '자산운용', '리스크관리', '채권시장', '헤지펀드', '투자전략', '경제지표',
    '물가', '소비자물가', '환율', '금융위기', '금융정책', '금융시스템', '국채',
    '지수', '상승률', '하락률', '증시', '매수', '매도', '자산관리', '고배당주',
    '금융상품', '부동산', '모기지', '대출', '채권금리', '금융기관', '거래소',
    '리보금리', '금융규제', 'FOMC', 'IMF', 'OECD', 'GDP', '실업률', '인플레이션',
    '유동성', '마진', '헤지', '옵션', '선물', '주식분석', '기업분석', '매출', '순이익',
    '영업이익', '비즈니스모델', '가치투자', '성장투자', '워런버핏', '테크주', '그로스',
    '인덱스펀드', '투자자', '주요지표', '저금리', '금융사기', '핀테크', '모바일뱅킹',
    '디지털자산', '핀테크기업', '금융기술', '블록체인기술', '디지털화폐', '리브라',
    '스테이블코인', '대체투자', '경기지표', '증권사', '금융컨설팅', '고정금리',
    '변동금리', '국제금융', '금융분석', '경제위기', '경제성장', '고용지표', '상장기업',
    '투자기회', '정책금리', '기준금리', '금융거래', '가치주', '성장주', '신용카드',
    '해외주식', '자본시장', '중앙은행', '금융업계', '회계기준', '기업회계', '회계사'
]

# 사용자 입력에서 키워드를 추출하는 함수
def extract_keyword(text):
    """사용자 입력에서 키워드를 추출합니다."""
    doc = nlp(text)

    # PhraseMatcher를 사용하여 미리 정의된 키워드를 문장에서 찾기
    matcher = PhraseMatcher(nlp.vocab)
    patterns = [nlp.make_doc(keyword) for keyword in predefined_keywords]
    matcher.add("PredefinedKeywords", patterns)

    matches = matcher(doc)
    matched_keywords = [doc[start:end].text for _, start, end in matches]

    # 키워드가 있으면 추출하고, 없으면 None 반환
    if matched_keywords:
        return matched_keywords[0] # 첫 번째 키워드 반환
    else:
        return None

```



```

# 날짜 추출 함수
def extract_date(text):
    """입력된 텍스트에서 날짜를 추출합니다. (YYYY년 MM월 DD일 형식)"""
    match = re.search(r"(\d{4})년 (\d{1,2})월 (\d{1,2})일", text)
    if match:
        return datetime.strptime(match.group(), "%Y년 %m월 %d일")
    return None

# HTML 태그를 제거하는 함수
def clean_html(text):
    """HTML 태그를 제거합니다."""
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

# 기사 내용을 청크로 나누는 함수
def chunk_text(text, chunk_size=1000):
    """기사 내용을 1000자 이하로 청크로 나눕니다."""
    chunks = [text[i:i + chunk_size] for i in range(0, len(text), chunk_size)]
    return chunks

if user_input:
    st.session_state.messages.append({"role": "user", "content": user_input})

    # 날짜와 키워드 추출
    date = extract_date(user_input)
    keyword = extract_keyword(user_input)

    # 기본 쿼리 생성
    query = user_input if not keyword else keyword

    with st.spinner("네이버에서 뉴스 가져오는 중..."):
        try:
            # 뉴스 검색
            news_items = fetch_naver_news(query, display=10)

            # 날짜 필터링 (사용자가 입력한 날짜에 해당하는 기사만 선택)
            filtered_news = []
            if date:
                for item in news_items:
                    pub_date = datetime.strptime(item["pubDate"], "%a, %d %b %Y %H:%M:%S %z")
                    if pub_date.date() == date.date():
                        filtered_news.append(item)
            else:
                filtered_news = news_items

            if not filtered_news:
                st.warning("지정된 날짜나 쿼리에 맞는 뉴스가 없습니다.")
            else:
                # 뉴스 내용 전처리 및 벡터화
                documents = []
                summaries = [] # 요약 리스트 저장
                sources = [] # 출처 리스트 저장

                for item in filtered_news:
                    clean_article = clean_html(item["description"]) # HTML 태그 제거 후 내용 정리
                    chunks = chunk_text(clean_article)
                    for chunk in chunks:
                        documents.append({"content": chunk, "source": item["originallink"]})
                    summaries.append(clean_html(item["description"])) # HTML 태그 제거 후 요약 추가
                    sources.append(item["originallink"]) # 출처 추가

                texts = [doc["content"] for doc in documents]
                metadatas = [{"source": doc["source"]} for doc in documents]

                # Chroma 벡터스토어 생성, persist_directory 경로 설정
                vectorstore = Chroma.from_texts(
                    texts,
                    embedding_model,
                    persist_directory=persist_directory,
                    metadatas=metadatas
                )

```

```

        retriever = vectorstore.as_retriever()

        # QA 체인 설정
        qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever, return_source_documents=True)

        # 쿼리 처리
        result = qa_chain(query)
        answer = result["result"]

        # 챗봇 형식의 응답 출력
        st.session_state.messages.append({"role": "assistant", "content": answer})

        # 요약별 출처 출력
        for summary, source in zip(summaries, sources):
            st.session_state.messages.append({"role": "assistant", "content": f"**요약**: {summary}"})
            st.session_state.messages.append({"role": "assistant", "content": f"**출처**: [원문 보기]({source})"})

    except Exception as e:
        st.error(f"뉴스를 가져오는 중 오류가 발생했습니다: {e}")

# 대화 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

```

https://drive.google.com/file/d/1O4_uCiXkYBcFvSWzJ_d5c2ckRSt6kFJE/view?usp=sharing

두 번째 케이스. fetch_news.py의 sort 파라미터를 sim으로 설정

- 오전 기준 테스트 시에는 전날의 금융 뉴스도 일부 가져왔지만 오후에는 전날 금융 뉴스를 가져오지 못함
 - 시간이 흐름에 따라, 최대로 가져올 수 있는 기사의 양 안에 전날의 기록은 포함되지 않는 것으로 파악
- fetch_news.py

```

import requests
import os
from dotenv import load_dotenv

load_dotenv()

NAVER_CLIENT_ID = os.getenv("NAVER_CLIENT_ID")
NAVER_CLIENT_SECRET = os.getenv("NAVER_CLIENT_SECRET")

print(f"Client ID: {NAVER_CLIENT_ID}")
print(f"Client Secret: {NAVER_CLIENT_SECRET}")

def fetch_naver_news(query, display=100):
    url = "https://openapi.naver.com/v1/search/news.json"
    headers = {
        "X-Naver-Client-Id": NAVER_CLIENT_ID,
        "X-Naver-Client-Secret": NAVER_CLIENT_SECRET
    }
    params = {
        "query": query,
        "display": display,
        "sort": "sim"
    }
    response = requests.get(url, headers=headers, params=params)
    if response.status_code == 200:
        return response.json()["items"]
    else:
        raise Exception(f"Error {response.status_code}: {response.text}")

```

- appV2.py

```

# appV2.py

import streamlit as st
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI
from langchain_community.embeddings import OpenAIEmbeddings
from langchain_community.vectorstores import Chroma
import openai
from fetch_news import fetch_naver_news
import os
from dotenv import load_dotenv
from datetime import datetime
from bs4 import BeautifulSoup
import re
import spacy
from spacy.matcher import PhraseMatcher

# 환경 변수 로드
load_dotenv()

# API 키 읽어오기
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
if OPENAI_API_KEY:
    print("API 키 로드 성공")
else:
    print("API 키 로드 실패")

# OpenAI 모델 설정
embedding_model = OpenAIEmbeddings(model="text-embedding-ada-002", api_key=OPENAI_API_KEY)
llm = ChatOpenAI(model="gpt-4o-mini", api_key=OPENAI_API_KEY) # gpt-4 모델 사용

# Chroma에 persist_directory 경로 지정
persist_directory = "./chroma_db"

# Streamlit 앱 제목
st.title("금융 뉴스 요약 챗봇 💬")
st.write("챗봇에 질문을 입력하면 최신 금융 뉴스를 요약하여 답변해드립니다!")

# 세션 상태 초기화
if "messages" not in st.session_state:
    st.session_state["messages"] = []

# 최초 접속 시 인사말 추가
if len(st.session_state.messages) == 0:
    st.session_state["messages"].append({"role": "assistant", "content": "안녕하세요! 질문을 입력해 주세요."})

# 이전 메시지 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

# 사용자 입력 받기
user_input = st.chat_input("질문을 입력해주세요 😊(ex. 2025년 1월 13일의 금융 뉴스를 알려줄래?, 최신 금융 동향을 알려줘, ...)")

# spaCy 모델 로드 (한국어)
nlp = spacy.load("ko_core_news_sm")

# 특정 키워드 리스트
predefined_keywords = [
    'Tesla', 'Bitcoin', 'Stock', '경제', '금융', '주식', '테슬라', '비트코인',
    '금리', '채권', '주식시장', '증권', '주식거래', 'ETF', '포트폴리오', '펀드',
    '주식투자', '가상화폐', '크립토', '블록체인', '상장', '코스피', '코스닥',
    '상장폐지', '상장주식', '배당', '주식배당', '시가총액', '이자율', '자산',
    '자산운용', '리스크관리', '채권시장', '헤지펀드', '투자전략', '경제지표',
    '물가', '소비자물가', '환율', '금융위기', '금융정책', '금융시스템', '국채',
    '지수', '상승률', '하락률', '증시', '매수', '매도', '자산관리', '고배당주',
    '금융상품', '부동산', '모기지', '대출', '채권금리', '금융기관', '거래소',
    '리보금리', '금융규제', 'FOMC', 'IMF', 'OECD', 'GDP', '실업률', '인플레이션',
    '유동성', '마진', '헤지', '옵션', '선물', '주식분석', '기업분석', '매출', '순이익',
    '영업이익', '비즈니스모델', '가치투자', '성장투자', '워런버핏', '테크주', '그로스',
    '인덱스펀드', '투자자', '주요지표', '저금리', '금융사기', '핀테크', '모바일뱅킹',
    '디지털자산', '핀테크기업', '금융기술', '블록체인기술', '디지털화폐', '리브라',
    '스테이블코인', '대체투자', '경기지표', '증권사', '금융컨설팅', '고정금리',

```

```

'변동금리', '국제금융', '금융분석', '경제위기', '경제성장', '고용지표', '상장기업',
'투자기회', '정책금리', '기준금리', '금융거래', '가치주', '성장주', '신용카드',
'해외주식', '자본시장', '중앙은행', '금융업계', '회계기준', '기업회계', '회계사'
]

# 사용자 입력에서 키워드를 추출하는 함수
def extract_keyword(text):
    """사용자 입력에서 키워드를 추출합니다."""
    doc = nlp(text)

    # PhraseMatcher를 사용하여 미리 정의된 키워드를 문장에서 찾기
    matcher = PhraseMatcher(nlp.vocab)
    patterns = [nlp.make_doc(keyword) for keyword in predefined_keywords]
    matcher.add("PredefinedKeywords", patterns)

    matches = matcher(doc)
    matched_keywords = [doc[start:end].text for _, start, end in matches]

    # 키워드가 있으면 추출하고, 없으면 None 반환
    if matched_keywords:
        return matched_keywords[0] # 첫 번째 키워드 반환
    else:
        return None

# 날짜 추출 함수
def extract_date(text):
    """입력된 텍스트에서 날짜를 추출합니다. (YYYY년 MM월 DD일 형식)"""
    match = re.search(r"(\d{4})년 (\d{1,2})월 (\d{1,2})일", text)
    if match:
        return datetime.strptime(match.group(), "%Y년 %m월 %d일")
    return None

# HTML 태그를 제거하는 함수
def clean_html(text):
    """HTML 태그를 제거합니다."""
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

# 기사 내용을 청크로 나누는 함수
def chunk_text(text, chunk_size=1000):
    """기사 내용을 1000자 이하로 청크로 나눕니다."""
    chunks = [text[i:i + chunk_size] for i in range(0, len(text), chunk_size)]
    return chunks

if user_input:
    st.session_state.messages.append({"role": "user", "content": user_input})

    # 날짜와 키워드 추출
    date = extract_date(user_input)
    keyword = extract_keyword(user_input)

    # 기본 쿼리 생성
    query = user_input if not keyword else keyword

    with st.spinner("네이버에서 뉴스 가져오는 중..."):
        try:
            # 뉴스 검색
            news_items = fetch_naver_news(query, display=10)

            # 날짜 필터링 (사용자가 입력한 날짜에 해당하는 기사만 선택)
            filtered_news = []
            if date:
                for item in news_items:
                    pub_date = datetime.strptime(item["pubDate"], "%a, %d %b %Y %H:%M:%S %z")
                    if pub_date.date() == date.date():
                        filtered_news.append(item)
            else:
                filtered_news = news_items

            if not filtered_news:
                st.warning("지정된 날짜나 쿼리에 맞는 뉴스가 없습니다.")
            else:

```

```

# 뉴스 내용 전처리 및 벡터화
documents = []
summaries = [] # 요약 리스트 저장
sources = [] # 출처 리스트 저장

for item in filtered_news:
    clean_article = clean_html(item["description"]) # HTML 태그 제거 후 내용 정리
    chunks = chunk_text(clean_article)
    for chunk in chunks:
        documents.append({"content": chunk, "source": item["originallink"]})
    summaries.append(clean_html(item["description"])) # HTML 태그 제거 후 요약 추가
    sources.append(item["originallink"]) # 출처 추가

texts = [doc["content"] for doc in documents]
metadatas = [{"source": doc["source"]} for doc in documents]

# Chroma 벡터스토어 생성, persist_directory 경로 설정
vectorstore = Chroma.from_texts(
    texts,
    embedding_model,
    persist_directory=persist_directory,
    metadatas=metadatas
)
retriever = vectorstore.as_retriever()

# QA 체인 설정
qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever, return_source_documents=True)

# 쿼리 처리
result = qa_chain(query)
answer = result["result"]

# 챗봇 형식의 응답 출력
st.session_state.messages.append({"role": "assistant", "content": answer})

# 요약별 출처 출력
for summary, source in zip(summaries, sources):
    st.session_state.messages.append({"role": "assistant", "content": f"**요약**: {summary}"})
    st.session_state.messages.append({"role": "assistant", "content": f"**출처**: [원문 보기]({source})"})

except Exception as e:
    st.error(f"뉴스를 가져오는 중 오류가 발생했습니다: {e}")

# 대화 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

```

https://drive.google.com/file/d/1tFmkg9M_oMHSYdow4AW17W53aYnR3Cpd/view?usp=sharing

현재 문제점

- fetch_news.py
 - sort를 **date** 로 설정할 경우
 - 장점 : 실시간 정보를 빠르게 잘 가져옴
 - 단점 : 아주 최근의 날짜(ex.당일)가 아닌 경우 쿼리를 가져오지 못함
 - sort를 **sim** 으로 설정할 경우
 - 장점 : 당일뿐만 아니라 전날 정보까지 일부 가져올 수 있음
 - 단점 : date의 장점인 실시간 정보를 활용하지 못함 → 관련도가 기준이기 때문에 실시간 반영에는 한계가 있는 상태
- 현재의 Naver Search API의 경우, 파라미터 설정에 정해진 규칙이 있고 다른 방법을 적용하기 어려움
 - 추후 해결 방안 : 파일 활용 혹은 크롤링
 - 파일의 경우, 직접 구성을 해야하기 때문에 실시간 연동성이 아주 좋지 않아 “크롤링”을 하는 것으로 잠정 결론

세 번째 케이스. fetch_news.py의 페이지 지정

- fetch_news.py 수정
 - 네이버 검색 API는 한 번에 최대 100개의 결과만 반환할 수 있음
 - `start` 파라미터를 적용해 여러 페이지 요청하는 것으로 수정 시도

```
import requests
import os
from dotenv import load_dotenv

load_dotenv()

NAVER_CLIENT_ID = os.getenv("NAVER_CLIENT_ID")
NAVER_CLIENT_SECRET = os.getenv("NAVER_CLIENT_SECRET")

print(f"Client ID: {NAVER_CLIENT_ID}")
print(f"Client Secret: {NAVER_CLIENT_SECRET}")

def fetch_naver_news(query, display=100, start=1):
    url = "https://openapi.naver.com/v1/search/news.json"
    headers = {
        "X-Naver-Client-Id": NAVER_CLIENT_ID,
        "X-Naver-Client-Secret": NAVER_CLIENT_SECRET
    }
    params = {
        "query": query,
        "display": display,
        "start": start, # 시작 페이지를 지정
        "sort": "date"
    }
    response = requests.get(url, headers=headers, params=params)
    if response.status_code == 200:
        return response.json()["items"]
    else:
        raise Exception(f"Error {response.status_code}: {response.text}")

all_news = []
for start in range(1, 301, 100): # 1, 101, 201 페이지를 요청
    all_news.extend(fetch_naver_news("경제", start=start))
```

- appV2.py

```
# appV2.py

import streamlit as st
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI
from langchain_community.embeddings import OpenAIEmbeddings
from langchain_community.vectorstores import Chroma
import openai
from fetch_news import fetch_naver_news
import os
from dotenv import load_dotenv
from datetime import datetime
from bs4 import BeautifulSoup
import re
import spacy
from spacy.matcher import PhraseMatcher

# 환경 변수 로드
load_dotenv()

# API 키 읽어오기
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
if OPENAI_API_KEY:
```

```

        print("API 키 로드 성공")
    else:
        print("API 키 로드 실패")

# OpenAI 모델 설정
embedding_model = OpenAIEmbeddings(model="text-embedding-ada-002", api_key=OPENAI_API_KEY)
llm = ChatOpenAI(model="gpt-4o-mini", api_key=OPENAI_API_KEY) # gpt-4 모델 사용

# Chroma에 persist_directory 경로 지정
persist_directory = "./chroma_db"

# Streamlit 앱 제목
st.title("금융 뉴스 요약 챗봇 😊")
st.write("챗봇에 질문을 입력하면 최신 금융 뉴스를 요약하여 답변해드립니다!")

# 세션 상태 초기화
if "messages" not in st.session_state:
    st.session_state["messages"] = []

# 최초 접속 시 인사말 추가
if len(st.session_state.messages) == 0:
    st.session_state["messages"].append({"role": "assistant", "content": "안녕하세요! 질문을 입력해 주세요."})

# 이전 메시지 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

# 사용자 입력 받기
user_input = st.chat_input("질문을 입력해주세요 😊(ex. 2025년 1월 13일의 금융 뉴스를 알려줄래?, 최신 금융 동향을 알려줘, ...)")

# spaCy 모델 로드 (한국어)
nlp = spacy.load("ko_core_news_sm")

# 특정 키워드 리스트
predefined_keywords = [
    'Tesla', 'Bitcoin', 'Stock', '경제', '금융', '주식', '테슬라', '비트코인',
    '금리', '채권', '주식시장', '증권', '주식거래', 'ETF', '포트폴리오', '펀드',
    '주식투자', '가상화폐', '크립토', '블록체인', '상장', '코스피', '코스닥',
    '상장폐지', '상장주식', '배당', '주식배당', '시가총액', '이자율', '자산',
    '자산운용', '리스크관리', '채권시장', '헤지펀드', '투자전략', '경제지표',
    '물가', '소비자물가', '환율', '금융위기', '금융정책', '금융시스템', '국채',
    '지수', '상승률', '하락률', '증시', '매수', '매도', '자산관리', '고배당주',
    '금융상품', '부동산', '모기지', '대출', '채권금리', '금융기관', '거래소',
    '리보금리', '금융규제', 'FOMC', 'IMF', 'OECD', 'GDP', '실업률', '인플레이션',
    '유동성', '마진', '헤지', '옵션', '선물', '주식분석', '기업분석', '매출', '순이익',
    '영업이익', '비즈니스모델', '가치투자', '성장투자', '워런버핏', '테크주', '그로스',
    '인덱스펀드', '투자자', '주요지표', '저금리', '금융사기', '핀테크', '모바일뱅킹',
    '디지털자산', '핀테크기업', '금융기술', '블록체인기술', '디지털화폐', '리브라',
    '스테이블코인', '대체투자', '경기지표', '증권사', '금융컨설팅', '고정금리',
    '변동금리', '국제금융', '금융분석', '경제위기', '경제성장', '고용지표', '상장기업',
    '투자기회', '정책금리', '기준금리', '금융거래', '가치주', '성장주', '신용카드',
    '해외주식', '자본시장', '중앙은행', '금융업계', '회계기준', '기업회계', '회계사'
]

# 사용자 입력에서 키워드를 추출하는 함수
def extract_keyword(text):
    """사용자 입력에서 키워드를 추출합니다."""
    doc = nlp(text)

    # PhraseMatcher를 사용하여 미리 정의된 키워드를 문장에서 찾기
    matcher = PhraseMatcher(nlp.vocab)
    patterns = [nlp.make_doc(keyword) for keyword in predefined_keywords]
    matcher.add("PredefinedKeywords", patterns)

    matches = matcher(doc)
    matched_keywords = [doc[start:end].text for _, start, end in matches]

    # 키워드가 있으면 추출하고, 없으면 None 반환
    if matched_keywords:
        return matched_keywords[0] # 첫 번째 키워드 반환
    else:
        return None

```



```

# 날짜 추출 함수
def extract_date(text):
    """입력된 텍스트에서 날짜를 추출합니다. (YYYY년 MM월 DD일 형식)"""
    match = re.search(r"(\d{4})년 (\d{1,2})월 (\d{1,2})일", text)
    if match:
        return datetime.strptime(match.group(), "%Y년 %m월 %d일")
    return None

# HTML 태그를 제거하는 함수
def clean_html(text):
    """HTML 태그를 제거합니다."""
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

# 기사 내용을 청크로 나누는 함수
def chunk_text(text, chunk_size=1000):
    """기사 내용을 1000자 이하로 청크로 나눕니다."""
    chunks = [text[i:i + chunk_size] for i in range(0, len(text), chunk_size)]
    return chunks

if user_input:
    st.session_state.messages.append({"role": "user", "content": user_input})

    # 날짜와 키워드 추출
    date = extract_date(user_input)
    keyword = extract_keyword(user_input)

    # 기본 쿼리 생성
    query = user_input if not keyword else keyword

    with st.spinner("네이버에서 뉴스 가져오는 중..."):
        try:
            # 뉴스 검색
            news_items = fetch_naver_news(query, display=10)

            # 날짜 필터링 (사용자가 입력한 날짜에 해당하는 기사만 선택)
            filtered_news = []
            if date:
                for item in news_items:
                    pub_date = datetime.strptime(item["pubDate"], "%a, %d %b %Y %H:%M:%S %z")
                    if pub_date.date() == date.date():
                        filtered_news.append(item)
            else:
                filtered_news = news_items

            if not filtered_news:
                st.warning("지정된 날짜나 쿼리에 맞는 뉴스가 없습니다.")
            else:
                # 뉴스 내용 전처리 및 벡터화
                documents = []
                summaries = [] # 요약 리스트 저장
                sources = [] # 출처 리스트 저장

                for item in filtered_news:
                    clean_article = clean_html(item["description"]) # HTML 태그 제거 후 내용 정리
                    chunks = chunk_text(clean_article)
                    for chunk in chunks:
                        documents.append({"content": chunk, "source": item["originallink"]})
                    summaries.append(clean_html(item["description"])) # HTML 태그 제거 후 요약 추가
                    sources.append(item["originallink"]) # 출처 추가

                texts = [doc["content"] for doc in documents]
                metadatas = [{"source": doc["source"]} for doc in documents]

                # Chroma 벡터스토어 생성, persist_directory 경로 설정
                vectorstore = Chroma.from_texts(
                    texts,
                    embedding_model,
                    persist_directory=persist_directory,
                    metadatas=metadatas
                )

```

```

retriever = vectorstore.as_retriever()

# QA 체인 설정
qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever, return_source_documents=True)

# 쿼리 처리
result = qa_chain(query)
answer = result["result"]

# 챗봇 형식의 응답 출력
st.session_state.messages.append({"role": "assistant", "content": answer})

# 요약별 출처 출력
for summary, source in zip(summaries, sources):
    st.session_state.messages.append({"role": "assistant", "content": f"**요약**: {summary}"})
    st.session_state.messages.append({"role": "assistant", "content": f"**출처**: [원문 보기]({source})"})

except Exception as e:
    st.error(f"뉴스를 가져오는 중 오류가 발생했습니다: {e}")

# 대화 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

```

- 결론 : 큰 차이는 없음

https://drive.google.com/file/d/196pjnL_1WF9W0rb8Jytu6wLPjZHNxX3b/view?usp=sharing

네 번째 케이스. app.py의 뉴스 검색 display를 크게 변경

```

# 기본 쿼리 생성
query = user_input if not keyword else keyword

with st.spinner("네이버에서 뉴스 가져오는 중..."):
    try:
        # 뉴스 검색
        news_items = fetch_naver_news(query, display=10)

        # 날짜 필터링 (사용자가 입력한 날짜에 해당하는 기사만 선택)
        filtered_news = []
        if date:
            for item in news_items:
                pub_date = datetime.strptime(item["pubDate"],
                if pub_date.date() == date.date():
                    filtered_news.append(item)

```

- fetch_news.py

```

import requests
import os
from dotenv import load_dotenv

load_dotenv()

NAVER_CLIENT_ID = os.getenv("NAVER_CLIENT_ID")
NAVER_CLIENT_SECRET = os.getenv("NAVER_CLIENT_SECRET")

print(f"Client ID: {NAVER_CLIENT_ID}")
print(f"Client Secret: {NAVER_CLIENT_SECRET}")

def fetch_naver_news(query, display=100):
    url = "https://openapi.naver.com/v1/search/news.json"
    headers = {
        "X-Naver-Client-Id": NAVER_CLIENT_ID,

```

```

        "X-Naver-Client-Secret": NAVER_CLIENT_SECRET
    }
    params = {
        "query": query,
        "display": display,
        "sort": "sim"
    }
    response = requests.get(url, headers=headers, params=params)
    if response.status_code == 200:
        return response.json()["items"]
    else:
        raise Exception(f"Error {response.status_code}: {response.text}")

```

- `appV2.py` → display를 100으로 확장

```

# appV2.py

import streamlit as st
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI
from langchain_community.embeddings import OpenAIEmbeddings
from langchain_community.vectorstores import Chroma
import openai
from fetch_news import fetch_naver_news
import os
from dotenv import load_dotenv
from datetime import datetime
from bs4 import BeautifulSoup
import re
import spacy
from spacy.matcher import PhraseMatcher

# 환경 변수 로드
load_dotenv()

# API 키 읽어오기
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
if OPENAI_API_KEY:
    print("API 키 로드 성공")
else:
    print("API 키 로드 실패")

# OpenAI 모델 설정
embedding_model = OpenAIEmbeddings(model="text-embedding-ada-002", api_key=OPENAI_API_KEY)
llm = ChatOpenAI(model="gpt-4o-mini", api_key=OPENAI_API_KEY) # gpt-4 모델 사용

# Chroma에 persist_directory 경로 지정
persist_directory = "./chroma_db"

# Streamlit 앱 제목
st.title("금융 뉴스 요약 챗봇 💬")
st.write("챗봇에 질문을 입력하면 최신 금융 뉴스를 요약하여 답변해드립니다!")

# 세션 상태 초기화
if "messages" not in st.session_state:
    st.session_state["messages"] = []

# 최초 접속 시 인사말 추가
if len(st.session_state.messages) == 0:
    st.session_state["messages"].append({"role": "assistant", "content": "안녕하세요! 질문을 입력해 주세요."})

# 이전 메시지 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

# 사용자 입력 받기
user_input = st.chat_input("질문을 입력해주세요😊(ex. 2025년 1월 13일의 금융 뉴스를 알려줄래?, 최신 금융 동향을 알려줘, ...)")

# spaCy 모델 로드 (한국어)
nlp = spacy.load("ko_core_news_sm")

```

```

# 특정 키워드 리스트
predefined_keywords = [
    'Tesla', 'Bitcoin', 'Stock', '경제', '금융', '주식', '테슬라', '비트코인',
    '금리', '채권', '주식시장', '증권', '주식거래', 'ETF', '포트폴리오', '펀드',
    '주식투자', '가상화폐', '크립토', '블록체인', '상장', '코스피', '코스닥',
    '상장폐지', '상장주식', '배당', '주식배당', '시가총액', '이자율', '자산',
    '자산운용', '리스크관리', '채권시장', '헤지펀드', '투자전략', '경제지표',
    '물가', '소비자물가', '환율', '금융위기', '금융정책', '금융시스템', '국채',
    '지수', '상승률', '하락률', '증시', '매수', '매도', '자산관리', '고배당주',
    '금융상품', '부동산', '모기지', '대출', '채권금리', '금융기관', '거래소',
    '리보금리', '금융규제', 'FOMC', 'IMF', 'OECD', 'GDP', '실업률', '인플레이션',
    '유동성', '마진', '헤지', '옵션', '선물', '주식분석', '기업분석', '매출', '순이익',
    '영업이익', '비즈니스모델', '가치투자', '성장투자', '워런버핏', '테크주', '그로쓰',
    '인덱스펀드', '투자자', '주요지표', '저금리', '금융사기', '핀테크', '모바일뱅킹',
    '디지털자산', '핀테크기업', '금융기술', '블록체인기술', '디지털화폐', '리브라',
    '스테이블코인', '대체투자', '경기지표', '증권사', '금융컨설팅', '고정금리',
    '변동금리', '국제금융', '금융분석', '경제위기', '경제성장', '고용지표', '상장기업',
    '투자기회', '정책금리', '기준금리', '금융거래', '가치주', '성장주', '신용카드',
    '해외주식', '자본시장', '중앙은행', '금융업계', '회계기준', '기업회계', '회계사'
]

# 사용자 입력에서 키워드를 추출하는 함수
def extract_keyword(text):
    """사용자 입력에서 키워드를 추출합니다."""
    doc = nlp(text)

    # PhraseMatcher를 사용하여 미리 정의된 키워드를 문장에서 찾기
    matcher = PhraseMatcher(nlp.vocab)
    patterns = [nlp.make_doc(keyword) for keyword in predefined_keywords]
    matcher.add("PredefinedKeywords", patterns)

    matches = matcher(doc)
    matched_keywords = [doc[start:end].text for _, start, end in matches]

    # 키워드가 있으면 추출하고, 없으면 None 반환
    if matched_keywords:
        return matched_keywords[0] # 첫 번째 키워드 반환
    else:
        return None

# 날짜 추출 함수
def extract_date(text):
    """입력된 텍스트에서 날짜를 추출합니다. (YYYY년 MM월 DD일 형식)"""
    match = re.search(r"(\d{4})년 (\d{1,2})월 (\d{1,2})일", text)
    if match:
        return datetime.strptime(match.group(), "%Y년 %m월 %d일")
    return None

# HTML 태그를 제거하는 함수
def clean_html(text):
    """HTML 태그를 제거합니다."""
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

# 기사 내용을 청크로 나누는 함수
def chunk_text(text, chunk_size=1000):
    """기사 내용을 1000자 이하로 청크로 나눕니다."""
    chunks = [text[i:i + chunk_size] for i in range(0, len(text), chunk_size)]
    return chunks

if user_input:
    st.session_state.messages.append({"role": "user", "content": user_input})

    # 날짜와 키워드 추출
    date = extract_date(user_input)
    keyword = extract_keyword(user_input)

    # 기본 쿼리 생성
    query = user_input if not keyword else keyword

    with st.spinner("네이버에서 뉴스 가져오는 중..."):
        try:

```

```

# 뉴스 검색
news_items = fetch_naver_news(query, display=100)

# 날짜 필터링 (사용자가 입력한 날짜에 해당하는 기사만 선택)
filtered_news = []
if date:
    for item in news_items:
        pub_date = datetime.strptime(item["pubDate"], "%a, %d %b %Y %H:%M:%S %z")
        if pub_date.date() == date.date():
            filtered_news.append(item)
else:
    filtered_news = news_items

if not filtered_news:
    st.warning("지정된 날짜나 쿼리에 맞는 뉴스가 없습니다.")
else:
    # 뉴스 내용 전처리 및 벡터화
    documents = []
    summaries = [] # 요약 리스트 저장
    sources = [] # 출처 리스트 저장

    for item in filtered_news:
        clean_article = clean_html(item["description"]) # HTML 태그 제거 후 내용 정리
        chunks = chunk_text(clean_article)
        for chunk in chunks:
            documents.append({"content": chunk, "source": item["originallink"]})
            summaries.append(clean_html(item["description"])) # HTML 태그 제거 후 요약 추가
            sources.append(item["originallink"]) # 출처 추가

    texts = [doc["content"] for doc in documents]
    metadatas = [{"source": doc["source"]} for doc in documents]

    # Chroma 벡터스토어 생성, persist_directory 경로 설정
    vectorstore = Chroma.from_texts(
        texts,
        embedding_model,
        persist_directory=persist_directory,
        metadatas=metadatas
    )
    retriever = vectorstore.as_retriever()

    # QA 체인 설정
    qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever, return_source_documents=True)

    # 쿼리 처리
    result = qa_chain(query)
    answer = result["result"]

    # 챗봇 형식의 응답 출력
    st.session_state.messages.append({"role": "assistant", "content": answer})

    # 요약별 출처 출력
    for summary, source in zip(summaries, sources):
        st.session_state.messages.append({"role": "assistant", "content": f"**요약**: {summary}"})
        st.session_state.messages.append({"role": "assistant", "content": f"**출처**: [원문 보기]({source})"})

except Exception as e:
    st.error(f"뉴스를 가져오는 중 오류가 발생했습니다: {e}")

# 대화 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

```

- 답변 개수가 늘었고, 전날까지 반영됨
 - display를 계속해서 높이면 더 이전의 뉴스들도 나오겠지만, 그 시점의 뉴스를 가져오기 위해 답변 개수가 1,000개, 10,000개... 이런 방식으로 함께 커질 것

- display는 20개로 조정

▼ appV2.py

```
# appV2.py
```

```

import streamlit as st
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI
from langchain_community.embeddings import OpenAIEmbeddings
from langchain_community.vectorstores import Chroma
import openai
from fetch_news import fetch_naver_news
import os
from dotenv import load_dotenv
from datetime import datetime
from bs4 import BeautifulSoup
import re
import spacy
from spacy.matcher import PhraseMatcher

# 환경 변수 로드
load_dotenv()

# API 키 읽어오기
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
if OPENAI_API_KEY:
    print("API 키 로드 성공")
else:
    print("API 키 로드 실패")

# OpenAI 모델 설정
embedding_model = OpenAIEmbeddings(model="text-embedding-ada-002", api_key=OPENAI_API_KEY)
llm = ChatOpenAI(model="gpt-4o-mini", api_key=OPENAI_API_KEY) # gpt-4 모델 사용

# Chroma에 persist_directory 경로 지정
persist_directory = "./chroma_db"

# Streamlit 앱 제목
st.title("금융 뉴스 요약 챗봇 💬")
st.write("챗봇에 질문을 입력하면 최신 금융 뉴스를 요약하여 답변해드립니다!")

# 세션 상태 초기화
if "messages" not in st.session_state:
    st.session_state["messages"] = []

# 최초 접속 시 인사말 추가
if len(st.session_state.messages) == 0:
    st.session_state["messages"].append({"role": "assistant", "content": "안녕하세요! 질문을 입력해 주세요."})

# 이전 메시지 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

# 사용자 입력 받기
user_input = st.chat_input("질문을 입력해주세요 😊(ex. 2025년 1월 13일의 금융 뉴스를 알려줄래?, 최신 금융 동향을 알려줘, ...)")

# spaCy 모델 로드 (한국어)
nlp = spacy.load("ko_core_news_sm")

# 특정 키워드 리스트
predefined_keywords = [
    'Tesla', 'Bitcoin', 'Stock', '경제', '금융', '주식', '테슬라', '비트코인',
    '금리', '채권', '주식시장', '증권', '주식거래', 'ETF', '포트폴리오', '펀드',
    '주식투자', '가상화폐', '크립토', '블록체인', '상장', '코스피', '코스닥',
    '상장폐지', '상장주식', '배당', '주식배당', '시가총액', '이자율', '자산',
    '자산운용', '리스크관리', '채권시장', '헤지펀드', '투자전략', '경제지표',
    '물가', '소비자물가', '환율', '금융위기', '금융정책', '금융시스템', '국채',
    '지수', '상승률', '하락률', '증시', '매수', '매도', '자산관리', '고배당주',
    '금융상품', '부동산', '모기지', '대출', '채권금리', '금융기관', '거래소',
    '리보금리', '금융규제', 'FOMC', 'IMF', 'OECD', 'GDP', '실업률', '인플레이션',
    '유동성', '마진', '헤지', '옵션', '선물', '주식분석', '기업분석', '매출', '순이익',
    '영업이익', '비즈니스모델', '가치투자', '성장투자', '워런버핏', '테크주', '그로스',
    '인덱스펀드', '투자자', '주요지표', '저금리', '금융사기', '핀테크', '모바일뱅킹',
    '디지털자산', '핀테크기업', '금융기술', '블록체인기술', '디지털화폐', '리브라',
    '스테이블코인', '대체투자', '경기지표', '증권사', '금융컨설팅', '고정금리',
    '변동금리', '국제금융', '금융분석', '경제위기', '경제성장', '고용지표', '상장기업',
    '투자기회', '정책금리', '기준금리', '금융거래', '가치주', '성장주', '신용카드',

```

```

'해외주식', '자본시장', '중앙은행', '금융업계', '회계기준', '기업회계', '회계사'
]

# 사용자 입력에서 키워드를 추출하는 함수
def extract_keyword(text):
    """사용자 입력에서 키워드를 추출합니다."""
    doc = nlp(text)

    # PhraseMatcher를 사용하여 미리 정의된 키워드를 문장에서 찾기
    matcher = PhraseMatcher(nlp.vocab)
    patterns = [nlp.make_doc(keyword) for keyword in predefined_keywords]
    matcher.add("PredefinedKeywords", patterns)

    matches = matcher(doc)
    matched_keywords = [doc[start:end].text for _, start, end in matches]

    # 키워드가 있으면 추출하고, 없으면 None 반환
    if matched_keywords:
        return matched_keywords[0] # 첫 번째 키워드 반환
    else:
        return None

# 날짜 추출 함수
def extract_date(text):
    """입력된 텍스트에서 날짜를 추출합니다. (YYYY년 MM월 DD일 형식)"""
    match = re.search(r"(\d{4})년 (\d{1,2})월 (\d{1,2})일", text)
    if match:
        return datetime.strptime(match.group(), "%Y년 %m월 %d일")
    return None

# HTML 태그를 제거하는 함수
def clean_html(text):
    """HTML 태그를 제거합니다."""
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

# 기사 내용을 청크로 나누는 함수
def chunk_text(text, chunk_size=1000):
    """기사 내용을 1000자 이하로 청크로 나눕니다."""
    chunks = [text[i:i + chunk_size] for i in range(0, len(text), chunk_size)]
    return chunks

if user_input:
    st.session_state.messages.append({"role": "user", "content": user_input})

    # 날짜와 키워드 추출
    date = extract_date(user_input)
    keyword = extract_keyword(user_input)

    # 기본 쿼리 생성
    query = user_input if not keyword else keyword

    with st.spinner("네이버에서 뉴스 가져오는 중..."):
        try:
            # 뉴스 검색
            news_items = fetch_naver_news(query, display=20)

            # 날짜 필터링 (사용자가 입력한 날짜에 해당하는 기사만 선택)
            filtered_news = []
            if date:
                for item in news_items:
                    pub_date = datetime.strptime(item["pubDate"], "%a, %d %b %Y %H:%M:%S %Z")
                    if pub_date.date() == date.date():
                        filtered_news.append(item)
            else:
                filtered_news = news_items

            if not filtered_news:
                st.warning("지정된 날짜나 쿼리에 맞는 뉴스가 없습니다.")
            else:
                # 뉴스 내용 전처리 및 벡터화
                documents = []

```



```

summaries = [] # 요약 리스트 저장
sources = [] # 출처 리스트 저장

for item in filtered_news:
    clean_article = clean_html(item["description"]) # HTML 태그 제거 후 내용 정리
    chunks = chunk_text(clean_article)
    for chunk in chunks:
        documents.append({"content": chunk, "source": item["originallink"]})
    summaries.append(clean_html(item["description"])) # HTML 태그 제거 후 요약 추가
    sources.append(item["originallink"]) # 출처 추가

texts = [doc["content"] for doc in documents]
metadatas = [{"source": doc["source"]} for doc in documents]

# Chroma 벡터스토어 생성, persist_directory 경로 설정
vectorstore = Chroma.from_texts(
    texts,
    embedding_model,
    persist_directory=persist_directory,
    metadatas=metadatas
)
retriever = vectorstore.as_retriever()

# QA 체인 설정
qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever, return_source_documents=True)

# 쿼리 처리
result = qa_chain(query)
answer = result["result"]

# 챗봇 형식의 응답 출력
st.session_state.messages.append({"role": "assistant", "content": answer})

# 요약별 출처 출력
for summary, source in zip(summaries, sources):
    st.session_state.messages.append({"role": "assistant", "content": f"**요약**: {summary}"})
    st.session_state.messages.append({"role": "assistant", "content": f"**출처**: [원문 보기]({source})"})

except Exception as e:
    st.error(f"뉴스를 가져오는 중 오류가 발생했습니다: {e}")

# 대화 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

```

코드 수정1: 날짜를 쿼리에 포함할 수 있도록 코드 추가

```

# 날짜가 있으면 쿼리에 포함
if date:
    query = f"{query} {date.strftime('%Y-%m-%d')}" # 날짜를 쿼리에 포함

```

코드 수정2: 기사 제목을 포함하도록 수정

appV3.py

- 변경 부분
 - ▼ 코드 전문(appV3.py)

```

# appV3.py

import streamlit as st
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI
from langchain_community.embeddings import OpenAIEmbeddings
from langchain_community.vectorstores import Chroma
import openai
from fetch_news import fetch_naver_news
import os
from dotenv import load_dotenv
from datetime import datetime
from bs4 import BeautifulSoup
import re

```

```

import spacy
from spacy.matcher import PhraseMatcher

# 환경 변수 로드
load_dotenv()

# API 키 읽어오기
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
if OPENAI_API_KEY:
    print("API 키 로드 성공")
else:
    print("API 키 로드 실패")

# OpenAI 모델 설정
embedding_model = OpenAIEmbeddings(model="text-embedding-ada-002", api_key=OPENAI_API_KEY)
llm = ChatOpenAI(model="gpt-4o-mini", api_key=OPENAI_API_KEY) # gpt-4 모델 사용

# Chroma에 persist_directory 경로 지정
persist_directory = "./chroma_db"

# Streamlit 앱 제목
st.title("금융 뉴스 요약 챗봇 🗨️")
st.write("챗봇에 질문을 입력하면 최신 금융 뉴스를 요약하여 답변해드립니다!")

# 세션 상태 초기화
if "messages" not in st.session_state:
    st.session_state["messages"] = []

# 최초 접속 시 인사말 추가
if len(st.session_state.messages) == 0:
    st.session_state["messages"].append({"role": "assistant", "content": "안녕하세요! 질문을 입력해 주세요."})

# 이전 메시지 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

# 사용자 입력 받기
user_input = st.chat_input("질문을 입력해주세요 😊 (ex. 2025년 1월 13일의 금융 뉴스를 알려줄래?, 최신 금융 동향을 알려줘, ...)")

# spaCy 모델 로드 (한국어)
nlp = spacy.load("ko_core_news_sm")

# 특정 키워드 리스트
predefined_keywords = [
    'Tesla', 'Bitcoin', 'Stock', '경제', '금융', '주식', '테슬라', '비트코인',
    '금리', '채권', '주식시장', '증권', '주식거래', 'ETF', '포트폴리오', '펀드',
    '주식투자', '가상화폐', '크립토', '블록체인', '상장', '코스피', '코스닥',
    '상장폐지', '상장주식', '배당', '주식배당', '시가총액', '이자율', '자산',
    '자산운용', '리스크관리', '채권시장', '헤지펀드', '투자전략', '경제지표',
    '물가', '소비자물가', '환율', '금융위기', '금융정책', '금융시스템', '국채',
    '지수', '상승률', '하락률', '증시', '매수', '매도', '자산관리', '고배당주',
    '금융상품', '부동산', '모기지', '대출', '채권금리', '금융기관', '거래소',
    '리보금리', '금융규제', 'FOMC', 'IMF', 'OECD', 'GDP', '실업률', '인플레이션',
    '유동성', '마진', '헤지', '옵션', '선물', '주식분석', '기업분석', '매출', '순이익',
    '영업이익', '비즈니스모델', '가치투자', '성장투자', '워런버핏', '테크주', '그로스',
    '인덱스펀드', '투자자', '주요지표', '저금리', '금융사기', '핀테크', '모바일뱅킹',
    '디지털자산', '핀테크기업', '금융기술', '블록체인기술', '디지털화폐', '리브라',
    '스테이블코인', '대체투자', '경기지표', '증권사', '금융컨설팅', '고정금리',
    '변동금리', '국제금융', '금융분석', '경제위기', '경제성장', '고용지표', '상장기업',
    '투자기회', '정책금리', '기준금리', '금융거래', '가치주', '성장주', '신용카드',
    '해외주식', '자본시장', '중앙은행', '금융업계', '회계기준', '기업회계', '회계사'
]

# 사용자 입력에서 키워드를 추출하는 함수
def extract_keyword(text):
    """사용자 입력에서 키워드를 추출합니다."""
    doc = nlp(text)

    # PhraseMatcher를 사용하여 미리 정의된 키워드를 문장에서 찾기
    matcher = PhraseMatcher(nlp.vocab)
    patterns = [nlp.make_doc(keyword) for keyword in predefined_keywords]
    matcher.add("PredefinedKeywords", patterns)

```

```

matches = matcher(doc)
matched_keywords = [doc[start:end].text for _, start, end in matches]

# 키워드가 있으면 추출하고, 없으면 None 반환
if matched_keywords:
    return matched_keywords[0] # 첫 번째 키워드 반환
else:
    return None

# 날짜 추출 함수
def extract_date(text):
    """입력된 텍스트에서 날짜를 추출합니다. (YYYY년 MM월 DD일 형식)"""
    match = re.search(r"(\d{4})년 (\d{1,2})월 (\d{1,2})일", text)
    if match:
        return datetime.strptime(match.group(), "%Y년 %m월 %d일")
    return None

# HTML 태그를 제거하는 함수
def clean_html(text):
    """HTML 태그를 제거합니다."""
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

# 기사 내용을 청크로 나누는 함수
def chunk_text(text, chunk_size=1000):
    """기사 내용을 1000자 이하로 청크로 나눕니다."""
    chunks = [text[i:i + chunk_size] for i in range(0, len(text), chunk_size)]
    return chunks

if user_input:
    st.session_state.messages.append({"role": "user", "content": user_input})

    # 날짜와 키워드 추출
    date = extract_date(user_input)
    keyword = extract_keyword(user_input)

    # 기본 쿼리 생성
    query = user_input if not keyword else keyword

    with st.spinner("네이버에서 뉴스 가져오는 중..."):
        try:
            # 뉴스 검색
            news_items = fetch_naver_news(query, display=20)

            # 날짜 필터링 (사용자가 입력한 날짜에 해당하는 기사만 선택)
            filtered_news = []
            if date:
                for item in news_items:
                    pub_date = datetime.strptime(item["pubDate"], "%a, %d %b %Y %H:%M:%S %z")
                    if pub_date.date() == date.date():
                        filtered_news.append(item)
            else:
                filtered_news = news_items

            if not filtered_news:
                st.warning("지정된 날짜나 쿼리에 맞는 뉴스가 없습니다.")
            else:
                # 뉴스 내용 전처리 및 벡터화
                documents = []
                summaries = [] # 요약 리스트 저장
                sources = [] # 출처 리스트 저장
                titles = [] # 제목 리스트 저장

                for item in filtered_news:
                    clean_article = clean_html(item["description"]) # HTML 태그 제거 후 내용 정리
                    chunks = chunk_text(clean_article)
                    for chunk in chunks:
                        documents.append({"content": chunk, "source": item["originallink"]})
                    summaries.append(clean_html(item["description"])) # HTML 태그 제거 후 요약 추가
                    sources.append(item["originallink"]) # 출처 추가
                    titles.append(clean_html(item["title"])) # 제목 추가

```

```

texts = [doc["content"] for doc in documents]
metadatas = [{"source": doc["source"]} for doc in documents]

# Chroma 벡터스토어 생성, persist_directory 경로 설정
vectorstore = Chroma.from_texts(
    texts,
    embedding_model,
    persist_directory=persist_directory,
    metadatas=metadatas
)
retriever = vectorstore.as_retriever()

# QA 체인 설정
qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever, return_source_documents=True)

# 쿼리 처리
result = qa_chain(query)
answer = result["result"]

# 챗봇 형식의 응답 출력
st.session_state.messages.append({"role": "assistant", "content": answer})

# 요약, 제목 및 출처 출력
for title, summary, source in zip(titles, summaries, sources):
    st.session_state.messages.append({"role": "assistant", "content": f"**제목**: {title}"})
    st.session_state.messages.append({"role": "assistant", "content": f"**요약**: {summary}"})
    st.session_state.messages.append({"role": "assistant", "content": f"**출처**: [원문 보기]({source})"})
except Exception as e:
    st.error(f"오류가 발생했습니다: {e}")

```

- 뉴스 내용 전체리에 타이틀 추가

```

# 뉴스 내용 전체리 및 벡터화
documents = []
summaries = [] # 요약 리스트 저장
sources = [] # 출처 리스트 저장
titles = [] # 제목 리스트 저장

for item in filtered_news:
    clean_article = clean_html(item["description"]) # HTML 태그 제거 후 내용 정리
    chunks = chunk_text(clean_article)
    for chunk in chunks:
        documents.append({"content": chunk, "source": item["originallink"]})
    summaries.append(clean_html(item["description"])) # HTML 태그 제거 후 요약 추가
    sources.append(item["originallink"]) # 출처 추가
    titles.append(clean_html(item["title"])) # 제목 추가

texts = [doc["content"] for doc in documents]
metadatas = [{"source": doc["source"]} for doc in documents]

```

- 답변 내용에 타이틀 추가

```

# 요약, 제목 및 출처 출력
for title, summary, source in zip(titles, summaries, sources):
    st.session_state.messages.append({"role": "assistant", "content": f"**제목**: {title}"})
    st.session_state.messages.append({"role": "assistant", "content": f"**요약**: {summary}"})
    st.session_state.messages.append({"role": "assistant", "content": f"**출처**: [원문 보기]({source})"})

```

- 문제 : 답변이 길어져 바로 나오지 않고 rerun을 해야 이전 답변이 보임

현재 가장 잘되는 버전은 appV2.py

코드 리팩토링

- 제목, 요약, 출처가 한번에 잘 나오도록 수정
- 이전 답변이 잘 유지되도록 수정
- fetch_news.py

```

import requests
import os

```

```

from dotenv import load_dotenv

load_dotenv()

NAVER_CLIENT_ID = os.getenv("NAVER_CLIENT_ID")
NAVER_CLIENT_SECRET = os.getenv("NAVER_CLIENT_SECRET")

print(f"Client ID: {NAVER_CLIENT_ID}")
print(f"Client Secret: {NAVER_CLIENT_SECRET}")

def fetch_naver_news(query, display=100):
    url = "https://openapi.naver.com/v1/search/news.json"
    headers = {
        "X-Naver-Client-Id": NAVER_CLIENT_ID,
        "X-Naver-Client-Secret": NAVER_CLIENT_SECRET
    }
    params = {
        "query": query,
        "display": display,
        "sort": "sim"
    }
    response = requests.get(url, headers=headers, params=params)
    if response.status_code == 200:
        return response.json()["items"]
    else:
        raise Exception(f"Error {response.status_code}: {response.text}")

```

- apptemp.py

```

import streamlit as st
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI
from langchain_community.embeddings import OpenAIEmbeddings
from langchain_community.vectorstores import Chroma
import openai
from fetch_news import fetch_naver_news
import os
from dotenv import load_dotenv
from datetime import datetime
from bs4 import BeautifulSoup
import re
import spacy
from spacy.matcher import PhraseMatcher

# 환경 변수 로드
load_dotenv()

# API 키 읽어오기
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
if OPENAI_API_KEY:
    print("API 키 로드 성공")
else:
    print("API 키 로드 실패")

# OpenAI 모델 설정
embedding_model = OpenAIEmbeddings(model="text-embedding-ada-002", api_key=OPENAI_API_KEY)
llm = ChatOpenAI(model="gpt-4o-mini", api_key=OPENAI_API_KEY) # gpt-4 모델 사용

# Chroma에 persist_directory 경로 지정
persist_directory = "./chroma_db"

# Streamlit 앱 제목
st.title("금융 뉴스 요약 챗봇 🗨️")
st.write("챗봇에 질문을 입력하면 최신 금융 뉴스를 요약하여 답변해드립니다!")

# 세션 상태 초기화
if "messages" not in st.session_state:
    st.session_state["messages"] = []

# 최초 접속 시 인사말 추가

```

```

if len(st.session_state.messages) == 0:
    st.session_state["messages"].append({"role": "assistant", "content": "안녕하세요! 질문을 입력해 주세요."})

# 이전 메시지 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

# 사용자 입력 받기
user_input = st.chat_input("질문을 입력해주세요 😊(ex. 2025년 1월 13일의 금융 뉴스를 알려줄래?, 최신 금융 동향을 알려줘, ...)")

# spaCy 모델 로드 (한국어)
nlp = spacy.load("ko_core_news_sm")

# 특정 키워드 리스트
predefined_keywords = [
    'Tesla', 'Bitcoin', 'Stock', '경제', '금융', '주식', '테슬라', '비트코인',
    '금리', '채권', '주식시장', '증권', '주식거래', 'ETF', '포트폴리오', '펀드',
    '주식투자', '가상화폐', '크립토', '블록체인', '상장', '코스피', '코스닥',
    '상장폐지', '상장주식', '배당', '주식배당', '시가총액', '이자율', '자산',
    '자산운용', '리스크관리', '채권시장', '헤지펀드', '투자전략', '경제지표',
    '물가', '소비자물가', '환율', '금융위기', '금융정책', '금융시스템', '국채',
    '지수', '상승률', '하락률', '증시', '매수', '매도', '자산관리', '고배당주',
    '금융상품', '부동산', '모기지', '대출', '채권금리', '금융기관', '거래소',
    '리보금리', '금융규제', 'FOMC', 'IMF', 'OECD', 'GDP', '실업률', '인플레이션',
    '유동성', '마진', '헤지', '옵션', '선물', '주식분석', '기업분석', '매출', '순이익',
    '영업이익', '비즈니스모델', '가치투자', '성장투자', '워런버핏', '테크주', '그로스',
    '인덱스펀드', '투자자', '주요지표', '저금리', '금융사기', '핀테크', '모바일뱅킹',
    '디지털자산', '핀테크기업', '금융기술', '블록체인기술', '디지털화폐', '리브라',
    '스테이블코인', '대체투자', '경기지표', '증권사', '금융컨설팅', '고정금리',
    '변동금리', '국제금융', '금융분석', '경제위기', '경제성장', '고용지표', '상장기업',
    '투자기회', '정책금리', '기준금리', '금융거래', '가치주', '성장주', '신용카드',
    '해외주식', '자본시장', '중앙은행', '금융업계', '회계기준', '기업회계', '회계사'
]

# 사용자 입력에서 키워드를 추출하는 함수
def extract_keyword(text):
    """사용자 입력에서 키워드를 추출합니다."""
    doc = nlp(text)

    # PhraseMatcher를 사용하여 미리 정의된 키워드를 문장에서 찾기
    matcher = PhraseMatcher(nlp.vocab)
    patterns = [nlp.make_doc(keyword) for keyword in predefined_keywords]
    matcher.add("PredefinedKeywords", patterns)

    matches = matcher(doc)
    matched_keywords = [doc[start:end].text for _, start, end in matches]

    # 키워드가 있으면 추출하고, 없으면 None 반환
    if matched_keywords:
        return matched_keywords[0] # 첫 번째 키워드 반환
    else:
        return None

# 날짜 추출 함수
def extract_date(text):
    """입력된 텍스트에서 날짜를 추출합니다. (YYYY년 MM월 DD일 형식)"""
    match = re.search(r"(\d{4})년 (\d{1,2})월 (\d{1,2})일", text)
    if match:
        return datetime.strptime(match.group(), "%Y년 %m월 %d일")
    return None

# HTML 태그를 제거하는 함수
def clean_html(text):
    """HTML 태그를 제거합니다."""
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

# 기사 내용을 청크로 나누는 함수
def chunk_text(text, chunk_size=1000):
    """기사 내용을 1000자 이하로 청크로 나눕니다."""
    chunks = [text[i:i + chunk_size] for i in range(0, len(text), chunk_size)]
    return chunks

```

```

if user_input:
    st.session_state.messages.append({"role": "user", "content": user_input})

    # 날짜와 키워드 추출
    date = extract_date(user_input)
    keyword = extract_keyword(user_input)

    # 기본 쿼리 생성
    query = user_input if not keyword else keyword

    # 날짜가 있으면 쿼리에 포함
    if date:
        query = f"{query} {date.strftime('%Y-%m-%d')}}" # 날짜를 쿼리에 포함

    with st.spinner("네이버에서 뉴스 가져오는 중..."):
        try:
            # 뉴스 검색
            news_items = fetch_naver_news(query, display=20)

            # 날짜 필터링 (사용자가 입력한 날짜에 해당하는 기사만 선택)
            filtered_news = []
            if date:
                for item in news_items:
                    pub_date = datetime.strptime(item["pubDate"], "%a, %d %b %Y %H:%M:%S %z")
                    if pub_date.date() == date.date():
                        filtered_news.append(item)
            else:
                filtered_news = news_items

            if not filtered_news:
                st.warning("지정된 날짜나 쿼리에 맞는 뉴스가 없습니다.")
            else:
                # 뉴스 내용 전처리 및 벡터화
                documents = []
                summaries = [] # 요약 리스트 저장
                sources = [] # 출처 리스트 저장
                titles = [] # 제목 리스트 저장

                for item in filtered_news:
                    clean_article = clean_html(item["description"]) # HTML 태그 제거 후 내용 정리
                    chunks = chunk_text(clean_article)
                    for chunk in chunks:
                        documents.append({"content": chunk, "source": item["originallink"]})
                    summaries.append(clean_html(item["description"])) # HTML 태그 제거 후 요약 추가
                    sources.append(item["originallink"]) # 출처 추가
                    titles.append(clean_html(item["title"])) # 제목 추가

                texts = [doc["content"] for doc in documents]
                metadatas = [{"source": doc["source"]} for doc in documents]

                # Chroma 벡터스토어 생성, persist_directory 경로 설정
                vectorstore = Chroma.from_texts(
                    texts,
                    embedding_model,
                    persist_directory=persist_directory,
                    metadatas=metadatas
                )
                retriever = vectorstore.as_retriever()

                # QA 체인 설정
                qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever, return_source_documents=True)

                # 쿼리 처리
                result = qa_chain(query)
                answer = result["result"]

                # 챗봇 형식의 응답 출력
                st.session_state.messages.append({"role": "assistant", "content": answer})

                # 제목, 요약, 출처 출력

```



```

        for title, summary, source in zip(titles, summaries, sources):
            st.session_state.messages.append({"role": "assistant", "content": f"**제목**: {title}"})
            st.session_state.messages.append({"role": "assistant", "content": f"**요약**: {summary}"})
            st.session_state.messages.append({"role": "assistant", "content": f"**출처**: [원문 보기]({source})"})

    except Exception as e:
        st.error(f"뉴스를 가져오는 중 오류가 발생했습니다: {e}")

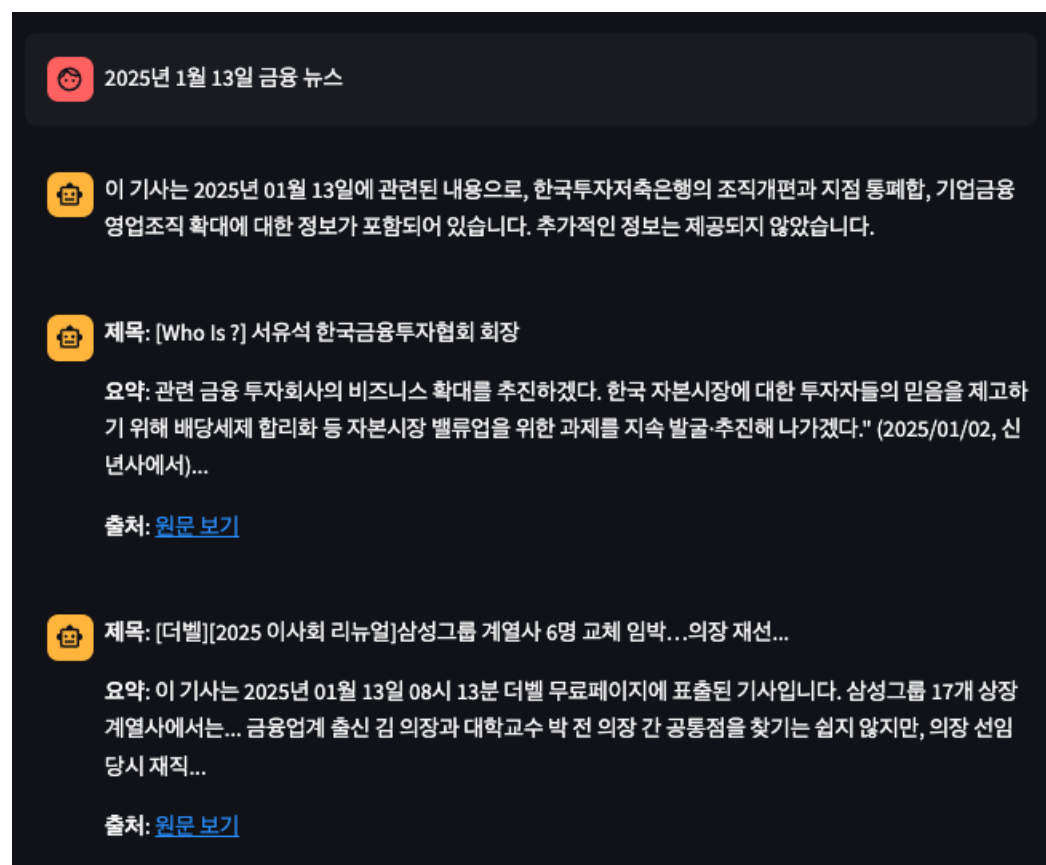
# 대화 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

```

- 가시성은 조금 떨어지지만, 비교적 이전 글들도 날짜에 맞게 잘 가져옴!
 - 요약 내용도 잘되는 것은 아니지만 어느 정도 잘 나옴.

<https://drive.google.com/file/d/1jl4Vne1ljD0ip6HEhk5TGxSEjeDCs5UT/view?usp=sharing>

전체 요약, 제목-요약-출처로 묶어 표기



- apptemp_markdown.py

```

import streamlit as st
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI
from langchain_community.embeddings import OpenAIEmbeddings
from langchain_community.vectorstores import Chroma
import openai
from fetch_news import fetch_naver_news
import os
from dotenv import load_dotenv
from datetime import datetime
from bs4 import BeautifulSoup
import re
import spacy
from spacy.matcher import PhraseMatcher

# 환경 변수 로드
load_dotenv()

# API 키 읽어오기
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
if OPENAI_API_KEY:
    print("API 키 로드 성공")

```

```

else:
    print("API 키 로드 실패")

# OpenAI 모델 설정
embedding_model = OpenAIEmbeddings(model="text-embedding-ada-002", api_key=OPENAI_API_KEY)
llm = ChatOpenAI(model="gpt-4o-mini", api_key=OPENAI_API_KEY) # gpt-4 모델 사용

# Chroma에 persist_directory 경로 지정
persist_directory = "./chroma_db"

# Streamlit 앱 제목
st.title("금융 뉴스 요약 챗봇 🗨️")
st.write("챗봇에 질문을 입력하면 최신 금융 뉴스를 요약하여 답변해드립니다!")

# 세션 상태 초기화
if "messages" not in st.session_state:
    st.session_state["messages"] = []

# 최초 접속 시 인사말 추가
if len(st.session_state.messages) == 0:
    st.session_state["messages"].append({"role": "assistant", "content": "안녕하세요! 질문을 입력해 주세요."})

# 이전 메시지 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

# 사용자 입력 받기
user_input = st.chat_input("질문을 입력해주세요😊(ex.2025년 1월 13일의 금융 뉴스를 알려줄래?, 최신 금융 동향을 알려줘, ...)")

# spaCy 모델 로드 (한국어)
nlp = spacy.load("ko_core_news_sm")

# 특정 키워드 리스트
predefined_keywords = [
    'Tesla', 'Bitcoin', 'Stock', '경제', '금융', '주식', '테슬라', '비트코인',
    '금리', '채권', '주식시장', '증권', '주식거래', 'ETF', '포트폴리오', '펀드',
    '주식투자', '가상화폐', '크립토', '블록체인', '상장', '코스피', '코스닥',
    '상장폐지', '상장주식', '배당', '주식배당', '시가총액', '이자율', '자산',
    '자산운용', '리스크관리', '채권시장', '헤지펀드', '투자전략', '경제지표',
    '물가', '소비자물가', '환율', '금융위기', '금융정책', '금융시스템', '국채',
    '지수', '상승률', '하락률', '증시', '매수', '매도', '자산관리', '고배당주',
    '금융상품', '부동산', '모기지', '대출', '채권금리', '금융기관', '거래소',
    '리보금리', '금융규제', 'FOMC', 'IMF', 'OECD', 'GDP', '실업률', '인플레이션',
    '유동성', '마진', '헤지', '옵션', '선물', '주식분석', '기업분석', '매출', '순이익',
    '영업이익', '비즈니스모델', '가치투자', '성장투자', '워런버핏', '테크주', '그로쓰',
    '인덱스펀드', '투자자', '주요지표', '저금리', '금융사기', '핀테크', '모바일뱅킹',
    '디지털자산', '핀테크기업', '금융기술', '블록체인기술', '디지털화폐', '리브라',
    '스테이블코인', '대체투자', '경기지표', '증권사', '금융컨설팅', '고정금리',
    '변동금리', '국제금융', '금융분석', '경제위기', '경제성장', '고용지표', '상장기업',
    '투자기회', '정책금리', '기준금리', '금융거래', '가치주', '성장주', '신용카드',
    '해외주식', '자본시장', '중앙은행', '금융업계', '회계기준', '기업회계', '회계사'
]

# 사용자 입력에서 키워드를 추출하는 함수
def extract_keyword(text):
    """사용자 입력에서 키워드를 추출합니다."""
    doc = nlp(text)

    # PhraseMatcher를 사용하여 미리 정의된 키워드를 문장에서 찾기
    matcher = PhraseMatcher(nlp.vocab)
    patterns = [nlp.make_doc(keyword) for keyword in predefined_keywords]
    matcher.add("PredefinedKeywords", patterns)

    matches = matcher(doc)
    matched_keywords = [doc[start:end].text for _, start, end in matches]

    # 키워드가 있으면 추출하고, 없으면 None 반환
    if matched_keywords:
        return matched_keywords[0] # 첫 번째 키워드 반환
    else:
        return None

```

```

# 날짜 추출 함수
def extract_date(text):
    """입력된 텍스트에서 날짜를 추출합니다. (YYYY년 MM월 DD일 형식)"""
    match = re.search(r"(\d{4})년 (\d{1,2})월 (\d{1,2})일", text)
    if match:
        return datetime.strptime(match.group(), "%Y년 %m월 %d일")
    return None

# HTML 태그를 제거하는 함수
def clean_html(text):
    """HTML 태그를 제거합니다."""
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

# 기사 내용을 청크로 나누는 함수
def chunk_text(text, chunk_size=1000):
    """기사 내용을 1000자 이하로 청크로 나눕니다."""
    chunks = [text[i:i + chunk_size] for i in range(0, len(text), chunk_size)]
    return chunks

if user_input:
    st.session_state.messages.append({"role": "user", "content": user_input})

    # 날짜와 키워드 추출
    date = extract_date(user_input)
    keyword = extract_keyword(user_input)

    # 기본 쿼리 생성
    query = user_input if not keyword else keyword

    # 날짜가 있으면 쿼리에 포함
    if date:
        query = f"{query} {date.strftime('%Y-%m-%d')}}" # 날짜를 쿼리에 포함

    with st.spinner("네이버에서 뉴스 가져오는 중..."):
        try:
            # 뉴스 검색
            news_items = fetch_naver_news(query, display=20)

            # 날짜 필터링 (사용자가 입력한 날짜에 해당하는 기사만 선택)
            filtered_news = []
            if date:
                for item in news_items:
                    pub_date = datetime.strptime(item["pubDate"], "%a, %d %b %Y %H:%M:%S %z")
                    if pub_date.date() == date.date():
                        filtered_news.append(item)
            else:
                filtered_news = news_items

            if not filtered_news:
                st.warning("지정된 날짜나 쿼리에 맞는 뉴스가 없습니다.")
            else:
                # 뉴스 내용 전처리 및 벡터화
                documents = []
                summaries = [] # 요약 리스트 저장
                sources = [] # 출처 리스트 저장
                titles = [] # 제목 리스트 저장

                for item in filtered_news:
                    clean_article = clean_html(item["description"]) # HTML 태그 제거 후 내용 정리
                    chunks = chunk_text(clean_article)
                    for chunk in chunks:
                        documents.append({"content": chunk, "source": item["originallink"]})
                    summaries.append(clean_html(item["description"])) # HTML 태그 제거 후 요약 추가
                    sources.append(item["originallink"]) # 출처 추가
                    titles.append(clean_html(item["title"])) # 제목 추가

                texts = [doc["content"] for doc in documents]
                metadatas = [{"source": doc["source"]} for doc in documents]

                # Chroma 벡터스토어 생성, persist_directory 경로 설정

```

```

        vectorstore = Chroma.from_texts(
            texts,
            embedding_model,
            persist_directory=persist_directory,
            metadatas=metadatas
        )
        retriever = vectorstore.as_retriever()

        # QA 체인 설정
        qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever, return_source_documents=True)

        # 쿼리 처리
        result = qa_chain(query)
        answer = result["result"]

        # 챗봇 형식의 응답 출력
        st.session_state.messages.append({"role": "assistant", "content": answer})

        # 제목, 요약, 출처 출력
        # for title, summary, source in zip(titles, summaries, sources):
        #     st.session_state.messages.append({"role": "assistant", "content": f"**제목**: {title}"})
        #     st.session_state.messages.append({"role": "assistant", "content": f"**요약**: {summary}"})
        #     st.session_state.messages.append({"role": "assistant", "content": f"**출처**: [원문 보기]({source})"})

        for title, summary, source in zip(titles, summaries, sources):
            combined_message = f"**제목**: {title}\n\n**요약**: {summary}\n\n**출처**: [원문 보기]({source})"
            st.session_state.messages.append({"role": "assistant", "content": combined_message})

    except Exception as e:
        st.error(f"뉴스를 가져오는 중 오류가 발생했습니다: {e}")

# 대화 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

```

→ 더 정리

- apptemp_summaryfortitle.py
 - 요약 정보를 1개의 정보당 1개의 답변으로 개선

```

import streamlit as st
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI
from langchain_community.embeddings import OpenAIEmbeddings
from langchain_community.vectorstores import Chroma
import openai
from fetch_news import fetch_naver_news
import os
from dotenv import load_dotenv
from datetime import datetime
from bs4 import BeautifulSoup
import re
import spacy
from spacy.matcher import PhraseMatcher

# 환경 변수 로드
load_dotenv()

# API 키 읽어오기
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
if OPENAI_API_KEY:
    print("API 키 로드 성공")
else:
    print("API 키 로드 실패")

# OpenAI 모델 설정
embedding_model = OpenAIEmbeddings(model="text-embedding-ada-002", api_key=OPENAI_API_KEY)
llm = ChatOpenAI(model="gpt-4o-mini", api_key=OPENAI_API_KEY) # gpt-4 모델 사용

# Chroma에 persist_directory 경로 지정
persist_directory = "./chroma_db"

```

```

# Streamlit 앱 제목
st.title("금융 뉴스 요약 챗봇 💬")
st.write("챗봇에 질문을 입력하면 최신 금융 뉴스를 요약하여 답변해드립니다!")

# 세션 상태 초기화
if "messages" not in st.session_state:
    st.session_state["messages"] = []

# 최초 접속 시 인사말 추가
if len(st.session_state.messages) == 0:
    st.session_state["messages"].append({"role": "assistant", "content": "안녕하세요! 질문을 입력해 주세요."})

# 이전 메시지 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

# 사용자 입력 받기
user_input = st.chat_input("질문을 입력해주세요😊(ex.2025년 1월 13일의 금융 뉴스를 알려줄래?, 최신 금융 동향을 알려줘, ...)")

# spaCy 모델 로드 (한국어)
nlp = spacy.load("ko_core_news_sm")

# 특정 키워드 리스트
predefined_keywords = [
    'Tesla', 'Bitcoin', 'Stock', '경제', '금융', '주식', '테슬라', '비트코인',
    '금리', '채권', '주식시장', '증권', '주식거래', 'ETF', '포트폴리오', '펀드',
    '주식투자', '가상화폐', '크립토', '블록체인', '상장', '코스피', '코스닥',
    '상장폐지', '상장주식', '배당', '주식배당', '시가총액', '이자율', '자산',
    '자산운용', '리스크관리', '채권시장', '헤지펀드', '투자전략', '경제지표',
    '물가', '소비자물가', '환율', '금융위기', '금융정책', '금융시스템', '국채',
    '지수', '상승률', '하락률', '증시', '매수', '매도', '자산관리', '고배당주',
    '금융상품', '부동산', '모기지', '대출', '채권금리', '금융기관', '거래소',
    '리보금리', '금융규제', 'FOMC', 'IMF', 'OECD', 'GDP', '실업률', '인플레이션',
    '유동성', '마진', '헤지', '옵션', '선물', '주식분석', '기업분석', '매출', '순이익',
    '영업이익', '비즈니스모델', '가치투자', '성장투자', '워런버핏', '테크주', '그로스',
    '인덱스펀드', '투자자', '주요지표', '저금리', '금융사기', '핀테크', '모바일뱅킹',
    '디지털자산', '핀테크기업', '금융기술', '블록체인기술', '디지털화폐', '리브라',
    '스테이블코인', '대체투자', '경기지표', '증권사', '금융컨설팅', '고정금리',
    '변동금리', '국제금융', '금융분석', '경제위기', '경제성장', '고용지표', '상장기업',
    '투자기회', '정책금리', '기준금리', '금융거래', '가치주', '성장주', '신용카드',
    '해외주식', '자본시장', '중앙은행', '금융업계', '회계기준', '기업회계', '회계사'
]

# 사용자 입력에서 키워드를 추출하는 함수
def extract_keyword(text):
    """사용자 입력에서 키워드를 추출합니다."""
    doc = nlp(text)

    # PhraseMatcher를 사용하여 미리 정의된 키워드를 문장에서 찾기
    matcher = PhraseMatcher(nlp.vocab)
    patterns = [nlp.make_doc(keyword) for keyword in predefined_keywords]
    matcher.add("PredefinedKeywords", patterns)

    matches = matcher(doc)
    matched_keywords = [doc[start:end].text for _, start, end in matches]

    # 키워드가 있으면 추출하고, 없으면 None 반환
    if matched_keywords:
        return matched_keywords[0] # 첫 번째 키워드 반환
    else:
        return None

# 날짜 추출 함수
def extract_date(text):
    """입력된 텍스트에서 날짜를 추출합니다. (YYYY년 MM월 DD일 형식)"""
    match = re.search(r"(\d{4})년 (\d{1,2})월 (\d{1,2})일", text)
    if match:
        return datetime.strptime(match.group(), "%Y년 %m월 %d일")
    return None

# HTML 태그를 제거하는 함수

```

```

def clean_html(text):
    """HTML 태그를 제거합니다."""
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

# 기사 내용을 청크로 나누는 함수
def chunk_text(text, chunk_size=1000):
    """기사 내용을 1000자 이하로 청크로 나눕니다."""
    chunks = [text[i:i + chunk_size] for i in range(0, len(text), chunk_size)]
    return chunks

if user_input:
    st.session_state.messages.append({"role": "user", "content": user_input})

    # 날짜와 키워드 추출
    date = extract_date(user_input)
    keyword = extract_keyword(user_input)

    # 기본 쿼리 생성
    query = user_input if not keyword else keyword

    # 날짜가 있으면 쿼리에 포함
    if date:
        query = f"{query} {date.strftime('%Y-%m-%d')}" # 날짜를 쿼리에 포함

    with st.spinner("네이버에서 뉴스 가져오는 중..."):
        try:
            # 뉴스 검색
            news_items = fetch_naver_news(query, display=20)

            # 날짜 필터링 (사용자가 입력한 날짜에 해당하는 기사만 선택)
            filtered_news = []
            if date:
                for item in news_items:
                    pub_date = datetime.strptime(item["pubDate"], "%a, %d %b %Y %H:%M:%S %z")
                    if pub_date.date() == date.date():
                        filtered_news.append(item)
            else:
                filtered_news = news_items

            if not filtered_news:
                st.warning("지정된 날짜나 쿼리에 맞는 뉴스가 없습니다.")
            else:
                # 뉴스 내용 전처리 및 벡터화
                documents = []
                summaries = [] # 요약 리스트 저장
                sources = [] # 출처 리스트 저장
                titles = [] # 제목 리스트 저장

                for item in filtered_news:
                    clean_article = clean_html(item["description"]) # HTML 태그 제거 후 내용 정리
                    chunks = chunk_text(clean_article)
                    for chunk in chunks:
                        documents.append({"content": chunk, "source": item["originallink"]})
                    summaries.append(clean_html(item["description"])) # HTML 태그 제거 후 요약 추가
                    sources.append(item["originallink"]) # 출처 추가
                    titles.append(clean_html(item["title"])) # 제목 추가

                texts = [doc["content"] for doc in documents]
                metadatas = [{"source": doc["source"]} for doc in documents]

                # Chroma 벡터스토어 생성, persist_directory 경로 설정
                vectorstore = Chroma.from_texts(
                    texts,
                    embedding_model,
                    persist_directory=persist_directory,
                    metadatas=metadatas
                )
                retriever = vectorstore.as_retriever()

                # QA 체인 설정

```

```

        qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever, return_source_documents=True)

        # 쿼리 처리
        result = qa_chain(query)
        answer = result["result"]

        # 챗봇 형식의 응답 출력
        st.session_state.messages.append({"role": "assistant", "content": answer})

        # 제목, 요약, 출처 출력
        for title, summary, source in zip(titles, summaries, sources):
            combined_message = f"{title}\n\n**요약**: {summary}\n\n**출처**: [원문 보기]({source})"
            st.session_state.messages.append({"role": "assistant", "content": combined_message})

    except Exception as e:
        st.error(f"뉴스를 가져오는 중 오류가 발생했습니다: {e}")

# 대화 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

```

https://drive.google.com/file/d/1bg1RFIK_Kvt2EK18zDTIGx2PEi1GK0qF/view?usp=sharing

결론

- 최종 코드 리팩토링
 - apptemp_summaryfortitle.py

```

import streamlit as st
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI
from langchain_community.embeddings import OpenAIEmbeddings
from langchain_community.vectorstores import Chroma
import openai
from fetch_news import fetch_naver_news
import os
from dotenv import load_dotenv
from datetime import datetime
from bs4 import BeautifulSoup
import re
import spacy
from spacy.matcher import PhraseMatcher

# 환경 변수 로드
load_dotenv()

# API 키 읽어오기
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
if OPENAI_API_KEY:
    print("API 키 로드 성공")
else:
    print("API 키 로드 실패")

# OpenAI 모델 설정
embedding_model = OpenAIEmbeddings(model="text-embedding-ada-002", api_key=OPENAI_API_KEY)
llm = ChatOpenAI(model="gpt-4o-mini", api_key=OPENAI_API_KEY) # gpt-4 모델 사용

# Chroma에 persist_directory 경로 지정
persist_directory = "./chroma_db"

# Streamlit 앱 제목
st.title("금융 뉴스 요약 챗봇 🗨️")
st.write("챗봇에 질문을 입력하면 최신 금융 뉴스를 요약하여 답변해드립니다!")

# 세션 상태 초기화

```



```

if "messages" not in st.session_state:
    st.session_state["messages"] = []

# 최초 접속 시 인사말 추가
if len(st.session_state.messages) == 0:
    st.session_state["messages"].append({"role": "assistant", "content": "안녕하세요! 질문을 입력해 주세요."})

# 이전 메시지 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

# 사용자 입력 받기
user_input = st.chat_input("질문을 입력해주세요😊(ex.2025년 1월 13일의 금융 뉴스를 알려줄래?, 최신 금융 동향을 알려줘, ...)")

# spaCy 모델 로드 (한국어)
nlp = spacy.load("ko_core_news_sm")

# 특정 키워드 리스트
predefined_keywords = [
    'Tesla', 'Bitcoin', 'Stock', '경제', '금융', '주식', '테슬라', '비트코인',
    '금리', '채권', '주식시장', '증권', '주식거래', 'ETF', '포트폴리오', '펀드',
    '주식투자', '가상화폐', '크립토', '블록체인', '상장', '코스피', '코스닥',
    '상장폐지', '상장주식', '배당', '주식배당', '시가총액', '이자율', '자산',
    '자산운용', '리스크관리', '채권시장', '헤지펀드', '투자전략', '경제지표',
    '물가', '소비자물가', '환율', '금융위기', '금융정책', '금융시스템', '국채',
    '지수', '상승률', '하락률', '증시', '매수', '매도', '자산관리', '고배당주',
    '금융상품', '부동산', '모기지', '대출', '채권금리', '금융기관', '거래소',
    '리보금리', '금융규제', 'FOMC', 'IMF', 'OECD', 'GDP', '실업률', '인플레이션',
    '유동성', '마진', '헤지', '옵션', '선물', '주식분석', '기업분석', '매출', '순이익',
    '영업이익', '비즈니스모델', '가치투자', '성장투자', '워런버핏', '테크주', '그로스',
    '인덱스펀드', '투자자', '주요지표', '저금리', '금융사기', '핀테크', '모바일뱅킹',
    '디지털자산', '핀테크기업', '금융기술', '블록체인기술', '디지털화폐', '리브라',
    '스테이블코인', '대체투자', '경기지표', '증권사', '금융컨설팅', '고정금리',
    '변동금리', '국제금융', '금융분석', '경제위기', '경제성장', '고용지표', '상장기업',
    '투자기회', '정책금리', '기준금리', '금융거래', '가치주', '성장주', '신용카드',
    '해외주식', '자본시장', '중앙은행', '금융업계', '회계기준', '기업회계', '회계사'
]

# 사용자 입력에서 키워드를 추출하는 함수
def extract_keyword(text):
    """사용자 입력에서 키워드를 추출합니다."""
    doc = nlp(text)

    # PhraseMatcher를 사용하여 미리 정의된 키워드를 문장에서 찾기
    matcher = PhraseMatcher(nlp.vocab)
    patterns = [nlp.make_doc(keyword) for keyword in predefined_keywords]
    matcher.add("PredefinedKeywords", patterns)

    matches = matcher(doc)
    matched_keywords = [doc[start:end].text for _, start, end in matches]

    # 키워드가 있으면 추출하고, 없으면 None 반환
    if matched_keywords:
        return matched_keywords[0] # 첫 번째 키워드 반환
    else:
        return None

# 날짜 추출 함수
def extract_date(text):
    """입력된 텍스트에서 날짜를 추출합니다. (YYYY년 MM월 DD일 형식)"""
    match = re.search(r"(\d{4})년 (\d{1,2})월 (\d{1,2})일", text)
    if match:
        return datetime.strptime(match.group(), "%Y년 %m월 %d일")
    return None

# HTML 태그를 제거하는 함수
def clean_html(text):
    """HTML 태그를 제거합니다."""
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

# 기사 내용을 청크로 나누는 함수

```

```

def chunk_text(text, chunk_size=1000):
    """기사 내용을 1000자 이하로 청크로 나눕니다."""
    chunks = [text[i:i + chunk_size] for i in range(0, len(text), chunk_size)]
    return chunks

if user_input:
    st.session_state.messages.append({"role": "user", "content": user_input})

    # 날짜와 키워드 추출
    date = extract_date(user_input)
    keyword = extract_keyword(user_input)

    # 기본 쿼리 생성
    query = user_input if not keyword else keyword

    # 날짜가 있으면 쿼리에 포함
    if date:
        query = f"{query} {date.strftime('%Y-%m-%d')}" # 날짜를 쿼리에 포함

    with st.spinner("네이버에서 뉴스 가져오는 중..."):
        try:
            # 뉴스 검색
            news_items = fetch_naver_news(query, display=20)

            # 날짜 필터링 (사용자가 입력한 날짜에 해당하는 기사만 선택)
            filtered_news = []
            if date:
                for item in news_items:
                    pub_date = datetime.strptime(item["pubDate"], "%a, %d %b %Y %H:%M:%S %z")
                    if pub_date.date() == date.date():
                        filtered_news.append(item)
            else:
                filtered_news = news_items

            if not filtered_news:
                st.warning("지정된 날짜나 쿼리에 맞는 뉴스가 없습니다.")
            else:
                # 뉴스 내용 전처리 및 벡터화
                documents = []
                summaries = [] # 요약 리스트 저장
                sources = [] # 출처 리스트 저장
                titles = [] # 제목 리스트 저장

                for item in filtered_news:
                    clean_article = clean_html(item["description"]) # HTML 태그 제거 후 내용 정리
                    chunks = chunk_text(clean_article)
                    for chunk in chunks:
                        documents.append({"content": chunk, "source": item["originallink"]})
                    summaries.append(clean_html(item["description"])) # HTML 태그 제거 후 요약 추가
                    sources.append(item["originallink"]) # 출처 추가
                    titles.append(clean_html(item["title"])) # 제목 추가

                texts = [doc["content"] for doc in documents]
                metadatas = [{"source": doc["source"]} for doc in documents]

                # Chroma 벡터스토어 생성, persist_directory 경로 설정
                vectorstore = Chroma.from_texts(
                    texts,
                    embedding_model,
                    persist_directory=persist_directory,
                    metadatas=metadatas
                )
                retriever = vectorstore.as_retriever()

                # QA 체인 설정
                qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever, return_source_documents=True)

                # 쿼리 처리
                result = qa_chain(query)
                answer = result["result"]

```

```

        # 챗봇 형식의 응답 출력
        st.session_state.messages.append({"role": "assistant", "content": answer})

    # 제목, 요약, 출처 출력
    for title, summary, source in zip(titles, summaries, sources):
        combined_message = f"{title}\n\n**요약**: {summary}\n\n**출처**: [원문 보기]({source})"
        st.session_state.messages.append({"role": "assistant", "content": combined_message})

    except Exception as e:
        st.error(f"뉴스를 가져오는 중 오류가 발생했습니다: {e}")

# 대화 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

```

- 날짜까지만 적용한 버전
- app_titleSummaryV2.py
 - 제목에 불필요한 특수문자 제거
 - 실제 기사 날짜 및 시간 추출하여 추가

```

# app_titleSummaryV2.py

import streamlit as st
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI
from langchain_community.embeddings import OpenAIEmbeddings
from langchain_community.vectorstores import Chroma
import openai
from fetch_news import fetch_naver_news
import os
from dotenv import load_dotenv
from datetime import datetime
from bs4 import BeautifulSoup
import re
import spacy
from spacy.matcher import PhraseMatcher

# 환경 변수 로드
load_dotenv()

# API 키 읽어오기
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
if OPENAI_API_KEY:
    print("API 키 로드 성공")
else:
    print("API 키 로드 실패")

# OpenAI 모델 설정
embedding_model = OpenAIEmbeddings(model="text-embedding-ada-002", api_key=OPENAI_API_KEY)
llm = ChatOpenAI(model="gpt-4o-mini", api_key=OPENAI_API_KEY) # gpt-4 모델 사용

# Chroma에 persist_directory 경로 지정
persist_directory = "./chroma_db"

# Streamlit 앱 제목
st.title("금융 뉴스 요약 챗봇 🗨️")
st.write("챗봇에 질문을 입력하면 최신 금융 뉴스를 요약하여 답변해드립니다!")

# 세션 상태 초기화
if "messages" not in st.session_state:
    st.session_state["messages"] = []

# 최초 접속 시 인사말 추가
if len(st.session_state.messages) == 0:
    st.session_state["messages"].append({"role": "assistant", "content": "안녕하세요! 질문을 입력해 주세요."})

# 이전 메시지 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

```

```

# 사용자 입력 받기
user_input = st.chat_input("질문을 입력해주세요 😊(ex. 2025년 1월 13일의 금융 뉴스를 알려줄래?, 최신 금융 동향을 알려줘, ...)")

# spaCy 모델 로드 (한국어)
nlp = spacy.load("ko_core_news_sm")

# 특정 키워드 리스트
predefined_keywords = [
    'Tesla', 'Bitcoin', 'Stock', '경제', '금융', '주식', '테슬라', '비트코인',
    '금리', '채권', '주식시장', '증권', '주식거래', 'ETF', '포트폴리오', '펀드',
    '주식투자', '가상화폐', '크립토', '블록체인', '상장', '코스피', '코스닥',
    '상장폐지', '상장주식', '배당', '주식배당', '시가총액', '이자율', '자산',
    '자산운용', '리스크관리', '채권시장', '헤지펀드', '투자전략', '경제지표',
    '물가', '소비자물가', '환율', '금융위기', '금융정책', '금융시스템', '국채',
    '지수', '상승률', '하락률', '증시', '매수', '매도', '자산관리', '고배당주',
    '금융상품', '부동산', '모기지', '대출', '채권금리', '금융기관', '거래소',
    '리보금리', '금융규제', 'FOMC', 'IMF', 'OECD', 'GDP', '실업률', '인플레이션',
    '유동성', '마진', '헤지', '옵션', '선물', '주식분석', '기업분석', '매출', '순이익',
    '영업이익', '비즈니스모델', '가치투자', '성장투자', '워런버핏', '테크주', '그로스',
    '인덱스펀드', '투자자', '주요지표', '저금리', '금융사기', '핀테크', '모바일뱅킹',
    '디지털자산', '핀테크기업', '금융기술', '블록체인기술', '디지털화폐', '리브라',
    '스테이블코인', '대체투자', '경기지표', '증권사', '금융컨설팅', '고정금리',
    '변동금리', '국제금융', '금융분석', '경제위기', '경제성장', '고용지표', '상장기업',
    '투자기회', '정책금리', '기준금리', '금융거래', '가치주', '성장주', '신용카드',
    '해외주식', '자본시장', '중앙은행', '금융업계', '회계기준', '기업회계', '회계사'
]

# 사용자 입력에서 키워드를 추출하는 함수
def extract_keyword(text):
    """사용자 입력에서 키워드를 추출합니다."""
    doc = nlp(text)

    # PhraseMatcher를 사용하여 미리 정의된 키워드를 문장에서 찾기
    matcher = PhraseMatcher(nlp.vocab)
    patterns = [nlp.make_doc(keyword) for keyword in predefined_keywords]
    matcher.add("PredefinedKeywords", patterns)

    matches = matcher(doc)
    matched_keywords = [doc[start:end].text for _, start, end in matches]

    # 키워드가 있으면 추출하고, 없으면 None 반환
    if matched_keywords:
        return matched_keywords[0] # 첫 번째 키워드 반환
    else:
        return None

# 날짜 추출 함수
def extract_date(text):
    """입력된 텍스트에서 날짜를 추출합니다. (YYYY년 MM월 DD일 형식)"""
    match = re.search(r"(\d{4})년 (\d{1,2})월 (\d{1,2})일", text)
    if match:
        return datetime.strptime(match.group(), "%Y년 %m월 %d일")
    return None

# HTML 태그를 제거하는 함수
def clean_html(text):
    """HTML 태그를 제거합니다."""
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

# 기사 내용을 청크로 나누는 함수
def chunk_text(text, chunk_size=1000):
    """기사 내용을 1000자 이하로 청크로 나눕니다."""
    chunks = [text[i:i + chunk_size] for i in range(0, len(text), chunk_size)]
    return chunks

if user_input:
    st.session_state.messages.append({"role": "user", "content": user_input})

# 날짜와 키워드 추출
date = extract_date(user_input)

```

```

keyword = extract_keyword(user_input)

# 기본 쿼리 생성
query = user_input if not keyword else keyword

# 날짜가 있으면 쿼리에 포함
if date:
    query = f"{query} {date.strftime('%Y-%m-%d')}}" # 날짜를 쿼리에 포함

with st.spinner("네이버에서 뉴스 가져오는 중..."):
    try:
        # 뉴스 검색
        news_items = fetch_naver_news(query, display=20)

        # 날짜 필터링 (사용자가 입력한 날짜에 해당하는 기사만 선택)
        filtered_news = []
        if date:
            for item in news_items:
                pub_date = datetime.strptime(item["pubDate"], "%a, %d %b %Y %H:%M:%S %z")
                if pub_date.date() == date.date():
                    filtered_news.append(item)
        else:
            filtered_news = news_items

        if not filtered_news:
            st.warning("지정된 날짜나 쿼리에 맞는 뉴스가 없습니다.")
        else:
            # 뉴스 내용 전처리 및 벡터화
            documents = []
            summaries = [] # 요약 리스트 저장
            sources = [] # 출처 리스트 저장
            titles = [] # 제목 리스트 저장
            dates = [] # 날짜 리스트 저장

            for item in filtered_news:
                clean_article = clean_html(item["description"]) # HTML 태그 제거 후 내용 정리
                raw_title = clean_html(item["title"])

                clean_title = re.sub(r"\[.*?\]", "", raw_title).strip()

                chunks = chunk_text(clean_article)
                for chunk in chunks:
                    documents.append({"content": chunk, "source": item["originallink"]})
                    summaries.append(clean_html(item["description"])) # HTML 태그 제거 후 요약 추가
                    sources.append(item["originallink"]) # 출처 추가
                    titles.append(clean_title) # 제목 추가

                pub_date = datetime.strptime(item["pubDate"], "%a, %d %b %Y %H:%M:%S %z") # 날짜 추출
                dates.append(pub_date.strftime('%Y-%m-%d %H:%M:%S')) # 날짜와 시간 추가

            texts = [doc["content"] for doc in documents]
            metadatas = [{"source": doc["source"]} for doc in documents]

            # Chroma 벡터스토어 생성, persist_directory 경로 설정
            vectorstore = Chroma.from_texts(
                texts,
                embedding_model,
                persist_directory=persist_directory,
                metadatas=metadatas
            )
            retriever = vectorstore.as_retriever()

            # QA 체인 설정
            qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever, return_source_documents=True)

            # 쿼리 처리
            result = qa_chain(query)
            answer = result["result"]

```

```

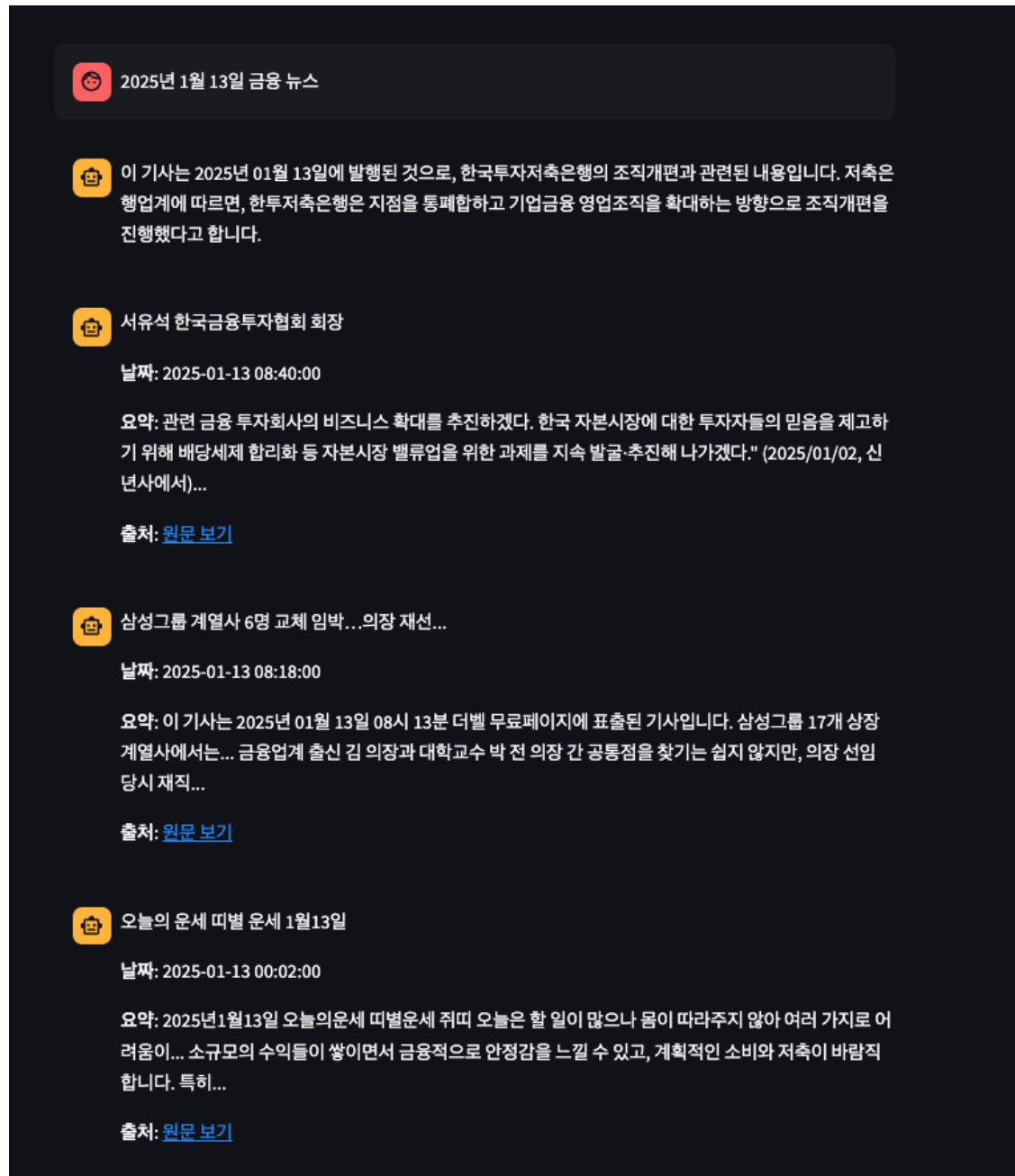
# 챗봇 형식의 응답 출력
st.session_state.messages.append({"role": "assistant", "content": answer})

# 제목, 날짜, 요약, 출처 출력
for title, summary, source, date in zip(titles, summaries, sources, dates):
    combined_message = f"{title}\n\n**날짜**: {date}\n\n**요약**: {summary}\n\n**출처**: [원문 보기]({source})"
    st.session_state.messages.append({"role": "assistant", "content": combined_message})

except Exception as e:
    st.error(f"뉴스를 가져오는 중 오류가 발생했습니다: {e}")

# 대화 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

```



[app_titleSummaryV2.pdf](#)

- app_titleSummaryV3.py
 - 출력 형식에 HTML 적용
 - 사용자 입력 시 “2025년 1월 13일”, “25년 1월 13일”, “1월 13일”, “13일”을 동일 날짜로 처리할 수 있도록 유연성 부여
 - 날짜 출력 시 시간까지 함께 나오도록 하나, 기사마다 시간 표기가 조금씩 달라 이부분도 유연성 부여
 - 📅: 시간대가 잘못 표기되는 문제가 있음 → 날짜 파싱함수 적용했으나 결과 동일..

```

# app_titleSummaryV3.py

import streamlit as st
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI
from langchain_community.embeddings import OpenAIEmbeddings
from langchain_community.vectorstores import Chroma

```

```

import openai
from fetch_news import fetch_naver_news
import os
from dotenv import load_dotenv
from datetime import datetime
from bs4 import BeautifulSoup
import re
import spacy
from spacy.matcher import PhraseMatcher

# 환경 변수 로드
load_dotenv()

# API 키 읽어오기
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
if OPENAI_API_KEY:
    print("API 키 로드 성공")
else:
    print("API 키 로드 실패")

# OpenAI 모델 설정
embedding_model = OpenAIEmbeddings(model="text-embedding-ada-002", api_key=OPENAI_API_KEY)
llm = ChatOpenAI(model="gpt-4o-mini", api_key=OPENAI_API_KEY) # gpt-4 모델 사용

# Chroma에 persist_directory 경로 지정
persist_directory = "./chroma_db"

# Streamlit 앱 제목
st.title("금융 뉴스 요약 챗봇 🗨️")
st.write("챗봇에 질문을 입력하면 최신 금융 뉴스를 요약하여 답변해드립니다!")

# 세션 상태 초기화
if "messages" not in st.session_state:
    st.session_state["messages"] = []

# 최초 접속 시 인사말 추가
if len(st.session_state.messages) == 0:
    st.session_state["messages"].append({"role": "assistant", "content": "안녕하세요! 질문을 입력해 주세요."})

# 이전 메시지 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

# 사용자 입력 받기
user_input = st.chat_input("질문을 입력해주세요 😊(ex. 2025년 1월 13일의 금융 뉴스를 알려줄래?, 최신 금융 동향을 알려줘, ...)")

# spaCy 모델 로드 (한국어)
nlp = spacy.load("ko_core_news_sm")

# 특정 키워드 리스트
predefined_keywords = [
    'Tesla', 'Bitcoin', 'Stock', '경제', '금융', '주식', '테슬라', '비트코인',
    '금리', '채권', '주식시장', '증권', '주식거래', 'ETF', '포트폴리오', '펀드',
    '주식투자', '가상화폐', '크립토', '블록체인', '상장', '코스피', '코스닥',
    '상장폐지', '상장주식', '배당', '주식배당', '시가총액', '이자율', '자산',
    '자산운용', '리스크관리', '채권시장', '헤지펀드', '투자전략', '경제지표',
    '물가', '소비자물가', '환율', '금융위기', '금융정책', '금융시스템', '국채',
    '지수', '상승률', '하락률', '증시', '매수', '매도', '자산관리', '고배당주',
    '금융상품', '부동산', '모기지', '대출', '채권금리', '금융기관', '거래소',
    '리보금리', '금융규제', 'FOMC', 'IMF', 'OECD', 'GDP', '실업률', '인플레이션',
    '유동성', '마진', '헤지', '옵션', '선물', '주식분석', '기업분석', '매출', '순이익',
    '영업이익', '비즈니스모델', '가치투자', '성장투자', '워런버핏', '테크주', '그로스',
    '인덱스펀드', '투자자', '주요지표', '저금리', '금융사기', '핀테크', '모바일뱅킹',
    '디지털자산', '핀테크기업', '금융기술', '블록체인기술', '디지털화폐', '리브라',
    '스테이블코인', '대체투자', '경기지표', '증권사', '금융컨설팅', '고정금리',
    '변동금리', '국제금융', '금융분석', '경제위기', '경제성장', '고용지표', '상장기업',
    '투자기회', '정책금리', '기준금리', '금융거래', '가치주', '성장주', '신용카드',
    '해외주식', '자본시장', '중앙은행', '금융업계', '회계기준', '기업회계', '회계사'
]

# 사용자 입력에서 키워드를 추출하는 함수
def extract_keyword(text):

```



```

"""사용자 입력에서 키워드를 추출합니다."""
doc = nlp(text)

# PhraseMatcher를 사용하여 미리 정의된 키워드를 문장에서 찾기
matcher = PhraseMatcher(nlp.vocab)
patterns = [nlp.make_doc(keyword) for keyword in predefined_keywords]
matcher.add("PredefinedKeywords", patterns)

matches = matcher(doc)
matched_keywords = [doc[start:end].text for _, start, end in matches]

# 키워드가 있으면 추출하고, 없으면 None 반환
if matched_keywords:
    return matched_keywords[0] # 첫 번째 키워드 반환
else:
    return None

# 날짜 추출 함수
def extract_date(text):
    """
    입력된 텍스트에서 날짜를 추출합니다.
    - "YYYY년 MM월 DD일", "MM월 DD일", "DD일" 등의 다양한 형식 지원
    - 형식이 없으면 None 반환
    """
    today = datetime.today()

    # 정규 표현식 패턴
    patterns = [
        (r"(\d{4})년 (\d{1,2})월 (\d{1,2})일", "%Y년 %m월 %d일"), # YYYY년 MM월 DD일
        (r"(\d{1,2})월 (\d{1,2})일", "%m월 %d일"), # MM월 DD일
        (r"(\d{1,2})일", "%d일") # DD일
    ]

    for pattern, date_format in patterns:
        match = re.search(pattern, text)
        if match:
            try:
                if date_format == "%d일": # 날짜만 주어진 경우
                    day = int(match.group(1))
                    return today.replace(day=day) # 이번 달의 해당 날짜 반환
                elif date_format == "%m월 %d일": # 월과 날짜만 주어진 경우
                    month, day = map(int, match.groups())
                    return today.replace(month=month, day=day)
                else: # YYYY년 MM월 DD일 형식
                    return datetime.strptime(match.group(), date_format)
            except ValueError:
                # 유효하지 않은 날짜일 경우 무시
                continue

    return None

# 날짜 파싱 함수 - 정규 표현식 사용
def parse_date(date_string):
    """
    다양한 날짜 형식을 유연하게 처리하는 함수.
    - date_string: 날짜 문자열
    """
    date_string = date_string.strip()

    # 날짜 형식에 대응하는 정규 표현식
    date_regex = r'(\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2})|' \
        r'(\d{2} \w{3} \d{4} \d{2}:\d{2}:\d{2})|' \
        r'(\d{4}/\d{2}/\d{2} \d{2}:\d{2})|' \
        r'(\d{2}-\d{2}-\d{4} \d{2}:\d{2})|' \
        r'(\w{3} \d{2}, \d{4} \d{2}:\d{2} \w{2})'

    # 정규 표현식으로 날짜 찾기
    match = re.search(date_regex, date_string)
    if match:
        date_str = match.group(0) # 추출된 날짜 문자열

```

```

# 각 날짜 형식에 맞는 파싱
try:
    if '-' in date_str and ':' in date_str:
        # 예: "2025-01-13 12:34:56"
        return datetime.strptime(date_str, "%Y-%m-%d %H:%M:%S")
    elif '/' in date_str:
        # 예: "2025/01/13 12:34"
        return datetime.strptime(date_str, "%Y/%m/%d %H:%M")
    elif '-' in date_str:
        # 예: "13-01-2025 12:34"
        return datetime.strptime(date_str, "%d-%m-%Y %H:%M")
    elif ',' in date_str:
        # 예: "Jan 13, 2025 12:34 PM"
        return datetime.strptime(date_str, "%b %d, %Y %I:%M %p")
    else:
        # 예: "13 Jan 2025 12:34:56"
        return datetime.strptime(date_str, "%d %b %Y %H:%M:%S")
except ValueError:
    print(f"날짜 파싱 실패: {date_str}")
    return None # 날짜 파싱 실패 시 None 반환
else:
    return None # 정규 표현식에 맞지 않으면 None 반환

# HTML 태그를 제거하는 함수
def clean_html(text):
    """HTML 태그를 제거합니다."""
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

# 기사 내용을 청크로 나누는 함수
def chunk_text(text, chunk_size=1000):
    """기사 내용을 1000자 이하로 청크로 나눕니다."""
    chunks = [text[i:i + chunk_size] for i in range(0, len(text), chunk_size)]
    return chunks

if user_input:
    st.session_state.messages.append({"role": "user", "content": user_input})

# 날짜와 키워드 추출
date = extract_date(user_input)
keyword = extract_keyword(user_input)

# 기본 쿼리 생성
query = user_input if not keyword else keyword

# 날짜가 있으면 쿼리에 포함
if date:
    query = f"{query} {date.strftime('%Y-%m-%d')}}" # 날짜를 쿼리에 포함

with st.spinner("네이버에서 뉴스 가져오는 중..."):
    try:
        # 뉴스 검색
        news_items = fetch_naver_news(query, display=20)

        # 날짜 필터링 (사용자가 입력한 날짜에 해당하는 기사만 선택)
        filtered_news = []
        if date:
            for item in news_items:
                pub_date = datetime.strptime(item["pubDate"], "%a, %d %b %Y %H:%M:%S %z")
                if pub_date.date() == date.date():
                    filtered_news.append(item)
        else:
            filtered_news = news_items

        if not filtered_news:
            st.warning("지정된 날짜나 쿼리에 맞는 뉴스가 없습니다.")
        else:
            # 뉴스 내용 전처리 및 벡터화
            documents = []
            summaries = [] # 요약 리스트 저장

```

```

sources = [] # 출처 리스트 저장
titles = [] # 제목 리스트 저장
dates = [] # 날짜 리스트 저장

for item in filtered_news:
    clean_article = clean_html(item["description"]) # HTML 태그 제거 후 내용 정리
    raw_title = clean_html(item["title"])

    clean_title = re.sub(r"\.[*?]", "", raw_title).strip()

    chunks = chunk_text(clean_article)
    for chunk in chunks:
        documents.append({"content": chunk, "source": item["originallink"]})
    summaries.append(clean_html(item["description"])) # HTML 태그 제거 후 요약 추가
    sources.append(item["originallink"]) # 출처 추가
    titles.append(clean_title) # 제목 추가

    pub_date = parse_date(item["pubDate"]) # 날짜 추출
    if pub_date:
        dates.append(pub_date.strftime('%Y-%m-%d %H:%M')) # 날짜와 시간 추가
    else:
        dates.append("Invalid Date") # 파싱 실패 시 "Invalid Date" 추가

texts = [doc["content"] for doc in documents]
metadatas = [{"source": doc["source"]} for doc in documents]

# Chroma 벡터스토어 생성, persist_directory 경로 설정
vectorstore = Chroma.from_texts(
    texts,
    embedding_model,
    persist_directory=persist_directory,
    metadatas=metadatas
)
retriever = vectorstore.as_retriever()

# QA 체인 설정
qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever, return_source_documents=True)

# 쿼리 처리
result = qa_chain(query)
answer = result["result"]

# 챗봇 형식의 응답 출력
st.session_state.messages.append({"role": "assistant", "content": answer})

# 제목, 날짜, 요약, 출처 출력
for title, summary, source, date in zip(titles, summaries, sources, dates):
    combined_message = f"""
<div style="background-color:#f9f9f9; padding:15px; border-radius:10px; margin-bottom:20px; border:1px solid #ccc">
    <h3 style="color:#333; font-weight:bold; margin-bottom:10px;">{title}</h3>
    <p style="font-size:14px; color:#888; margin-bottom:10px;"><strong>날짜:</strong> {date}</p>
    <p style="font-size:16px; color:#444; margin-bottom:15px;">{summary}</p>
    <a href="{source}" target="_blank" style="display:inline-block; background-color:#007BFF; color:white; padding:2px 10px; text-decoration:none;">출처</a>
</div>
"""
    st.session_state.messages.append({"role": "assistant", "content": combined_message})

except Exception as e:
    st.error(f"뉴스를 가져오는 중 오류가 발생했습니다: {e}")

# 대화 출력
for msg in st.session_state.messages:
    if msg['role'] == 'assistant':
        st.markdown(msg['content'], unsafe_allow_html=True)
    else:
        st.chat_message(msg['role']).write(msg['content'])

```

- 2차 수정

- 날짜 형식을 "2025-01-13 오후 12시"와 같이 시간대까지만 표기할 수 있도록 변경
- 기사 내용이 없는 경우 "기사가 없습니다"를 출력하도록 수정

```
import streamlit as st
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI
from langchain_community.embeddings import OpenAIEmbeddings
from langchain_community.vectorstores import Chroma
import openai
from fetch_news import fetch_naver_news
import os
from dotenv import load_dotenv
from datetime import datetime
from bs4 import BeautifulSoup
import re
import spacy
from spacy.matcher import PhraseMatcher

load_dotenv()

OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
if OPENAI_API_KEY:
    print("API 키 로드 성공")
else:
    print("API 키 로드 실패")

embedding_model = OpenAIEmbeddings(model="text-embedding-ada-002", api_key=OPENAI_API_KEY)
llm = ChatOpenAI(model="gpt-4o-mini", api_key=OPENAI_API_KEY)

persist_directory = "./chroma_db"

# Streamlit
st.title("금융 뉴스 요약 챗봇 😊")
st.write("챗봇에 질문을 입력하면 최신 금융 뉴스를 요약하여 답변해드립니다!")

# 세션 상태 초기화
if "messages" not in st.session_state:
    st.session_state["messages"] = []

# 최초 접속 시 인사말
if len(st.session_state.messages) == 0:
    st.session_state["messages"].append({"role": "assistant", "content": "안녕하세요! 질문을 입력해 주세요."})

# 이전 메시지 출력
for msg in st.session_state.messages:
    st.chat_message(msg['role']).write(msg['content'])

# 사용자 입력
user_input = st.chat_input("질문을 입력해주세요 😊(ex. 2025년 1월 13일의 금융 뉴스를 알려줄래?, 최신 금융 동향을 알려줘, ...)")

# spaCy 모델 로드 (한국어)
nlp = spacy.load("ko_core_news_sm")

# 특정 키워드 리스트(spaCy 모델에서 키워드를 뽑지 못할 경우 사용)
predefined_keywords = [
    'Tesla', 'Bitcoin', 'Stock', '경제', '금융', '주식', '테슬라', '비트코인',
    '금리', '채권', '주식시장', '증권', '주식거래', 'ETF', '포트폴리오', '펀드',
    '주식투자', '가상화폐', '크립토', '블록체인', '상장', '코스피', '코스닥',
    '상장폐지', '상장주식', '배당', '주식배당', '시가총액', '이자율', '자산',
    '자산운용', '리스크관리', '채권시장', '헤지펀드', '투자전략', '경제지표',
    '물가', '소비자물가', '환율', '금융위기', '금융정책', '금융시스템', '국채',
    '지수', '상승률', '하락률', '증시', '매수', '매도', '자산관리', '고배당주',
    '금융상품', '부동산', '모기지', '대출', '채권금리', '금융기관', '거래소',
    '리보금리', '금융규제', 'FOMC', 'IMF', 'OECD', 'GDP', '실업률', '인플레이션',
    '유동성', '마진', '헤지', '옵션', '선물', '주식분석', '기업분석', '매출', '순이익',
    '영업이익', '비즈니스모델', '가치투자', '성장투자', '워런버핏', '테크주', '그로스',
    '인덱스펀드', '투자자', '주요지표', '저금리', '금융사기', '핀테크', '모바일뱅킹',
    '디지털자산', '핀테크기업', '금융기술', '블록체인기술', '디지털화폐', '리브라',
    '스테이블코인', '대체투자', '경기지표', '증권사', '금융컨설팅', '고정금리',
    '변동금리', '국제금융', '금융분석', '경제위기', '경제성장', '고용지표', '상장기업',
    '투자기회', '정책금리', '기준금리', '금융거래', '가치주', '성장주', '신용카드',
    '해외주식', '자본시장', '중앙은행', '금융업계', '회계기준', '기업회계', '회계사'
```

```

]

# 사용자 입력에서 키워드를 추출하는 함수
def extract_keyword(text):
    """사용자 입력에서 키워드를 추출합니다."""
    doc = nlp(text)

    # PhraseMatcher를 사용하여 미리 정의된 키워드를 문장에서 찾기
    matcher = PhraseMatcher(nlp.vocab)
    patterns = [nlp.make_doc(keyword) for keyword in predefined_keywords]
    matcher.add("PredefinedKeywords", patterns)

    matches = matcher(doc)
    matched_keywords = [doc[start:end].text for _, start, end in matches]

    # 키워드가 있으면 추출하고, 없으면 None 반환
    if matched_keywords:
        return matched_keywords[0] # 첫번째 키워드만 반환
    else:
        return None

# 날짜 추출 함수
def extract_date(text):
    """
    입력된 텍스트에서 날짜를 추출합니다.
    - "YYYY년 MM월 DD일", "MM월 DD일", "DD일" 등의 다양한 형식 지원
    - 형식이 없으면 None 반환
    """
    today = datetime.today()

    # 정규 표현식 패턴
    patterns = [
        (r"(\d{4})년 (\d{1,2})월 (\d{1,2})일", "%Y년 %m월 %d일"), # YYYY년 MM월 DD일
        (r"(\d{1,2})월 (\d{1,2})일", "%m월 %d일"), # MM월 DD일
        (r"(\d{1,2})일", "%d일") # DD일
    ]

    for pattern, date_format in patterns:
        match = re.search(pattern, text)
        if match:
            try:
                if date_format == "%d일": # 날짜만 주어진 경우
                    day = int(match.group(1))
                    return today.replace(day=day) # 이번 달의 해당 날짜 반환
                elif date_format == "%m월 %d일": # 월과 날짜만 주어진 경우
                    month, day = map(int, match.groups())
                    return today.replace(month=month, day=day)
                else: # YYYY년 MM월 DD일 형식
                    return datetime.strptime(match.group(), date_format)
            except ValueError:
                # 유효하지 않은 날짜일 경우 무시
                continue

    return None

# HTML 태그를 제거하는 함수
def clean_html(text):
    """HTML 태그를 제거합니다."""
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

# 기사 내용을 청크로 나누는 함수
def chunk_text(text, chunk_size=1000):
    """기사 내용을 1000자 이하로 청크로 나눕니다."""
    chunks = [text[i:i + chunk_size] for i in range(0, len(text), chunk_size)]
    return chunks

```

```

if user_input:
    st.session_state.messages.append({"role": "user", "content": user_input})

    # 날짜와 키워드 추출
    date = extract_date(user_input)
    keyword = extract_keyword(user_input)

    # 기본 쿼리
    query = user_input if not keyword else keyword

    # 날짜가 있으면 쿼리에 포함
    if date:
        query = f"{query} {date.strftime('%Y-%m-%d')}}" # 날짜를 쿼리에 포함

    with st.spinner("네이버에서 뉴스 가져오는 중..."):
        try:
            # 뉴스 검색
            news_items = fetch_naver_news(query, display=20)

            # 날짜 필터링 (사용자가 입력한 날짜에 해당하는 기사만 선택)
            filtered_news = []
            if date:
                for item in news_items:
                    pub_date = datetime.strptime(item["pubDate"], "%a, %d %b %Y %H:%M:%S %z")
                    if pub_date.date() == date.date():
                        filtered_news.append(item)
            else:
                filtered_news = news_items

            if not filtered_news:
                # st.warning("지정된 날짜나 쿼리에 맞는 뉴스가 없습니다.")
                st.session_state.messages.append({"role": "assistant", "content": "기사가 없습니다😞"})
            else:
                # 뉴스 내용 전처리 및 벡터화
                documents = []
                summaries = [] # 요약 리스트
                sources = [] # 출처 리스트
                titles = [] # 제목 리스트
                dates = [] # 날짜 리스트

                for item in filtered_news:
                    # HTML 태그 제거
                    clean_article = clean_html(item["description"])

                    raw_title = clean_html(item["title"])
                    clean_title = re.sub(r"\[.*?\]", "", raw_title).strip()

                    chunks = chunk_text(clean_article)
                    for chunk in chunks:
                        documents.append({"content": chunk, "source": item["originallink"]})

                    # 요약
                    summaries.append(clean_html(item["description"]))

                    # 출처
                    sources.append(item["originallink"])

                    # 제목
                    titles.append(clean_title)

                    # 날짜
                    pub_date = datetime.strptime(item["pubDate"], "%a, %d %b %Y %H:%M:%S %z")
                    formatted_date = pub_date.strftime('%Y-%m-%d %p %I시')
                    formatted_date = formatted_date.replace('AM', '오전').replace('PM', '오후')
                    dates.append(formatted_date)

                texts = [doc["content"] for doc in documents]
                metadatas = [{"source": doc["source"]} for doc in documents]

                # Chroma 벡터스토어 생성, persist_directory 경로 설정
                vectorstore = Chroma.from_texts(
                    texts,

```

```

        embedding_model,
        persist_directory=persist_directory,
        metadatas=metadatas
    )
    retriever = vectorstore.as_retriever()

# QA 체인 설정
qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever, return_source_documents=True)

# 쿼리 처리
result = qa_chain(query)
answer = result["result"]

# 챗봇 형식의 응답 출력
st.session_state.messages.append({"role": "assistant", "content": answer})

# 제목, 날짜, 요약, 출처 출력
for title, summary, source, date in zip(titles, summaries, sources, dates):
    combined_message = f"""
    <div style="background-color:#FFFFFF; padding:15px; border-radius:10px; margin-bottom:20px; border:1
    <h5 style="color:#333; font-weight:bold; margin-bottom:10px; white-space: normal; word-wrap: bre
    <p style="font-size:14px; color:#888; margin-bottom:10px;"><strong>날짜:</strong> {date}</p>
    <p style="font-size:16px; color:#444; margin-bottom:15px;">{summary}</p>
    <div style="text-align:right;">
        <a href="{source}" target="_blank" style="display:inline-block; background-color:#FFBF00; co
    </div>
    </div>
    """

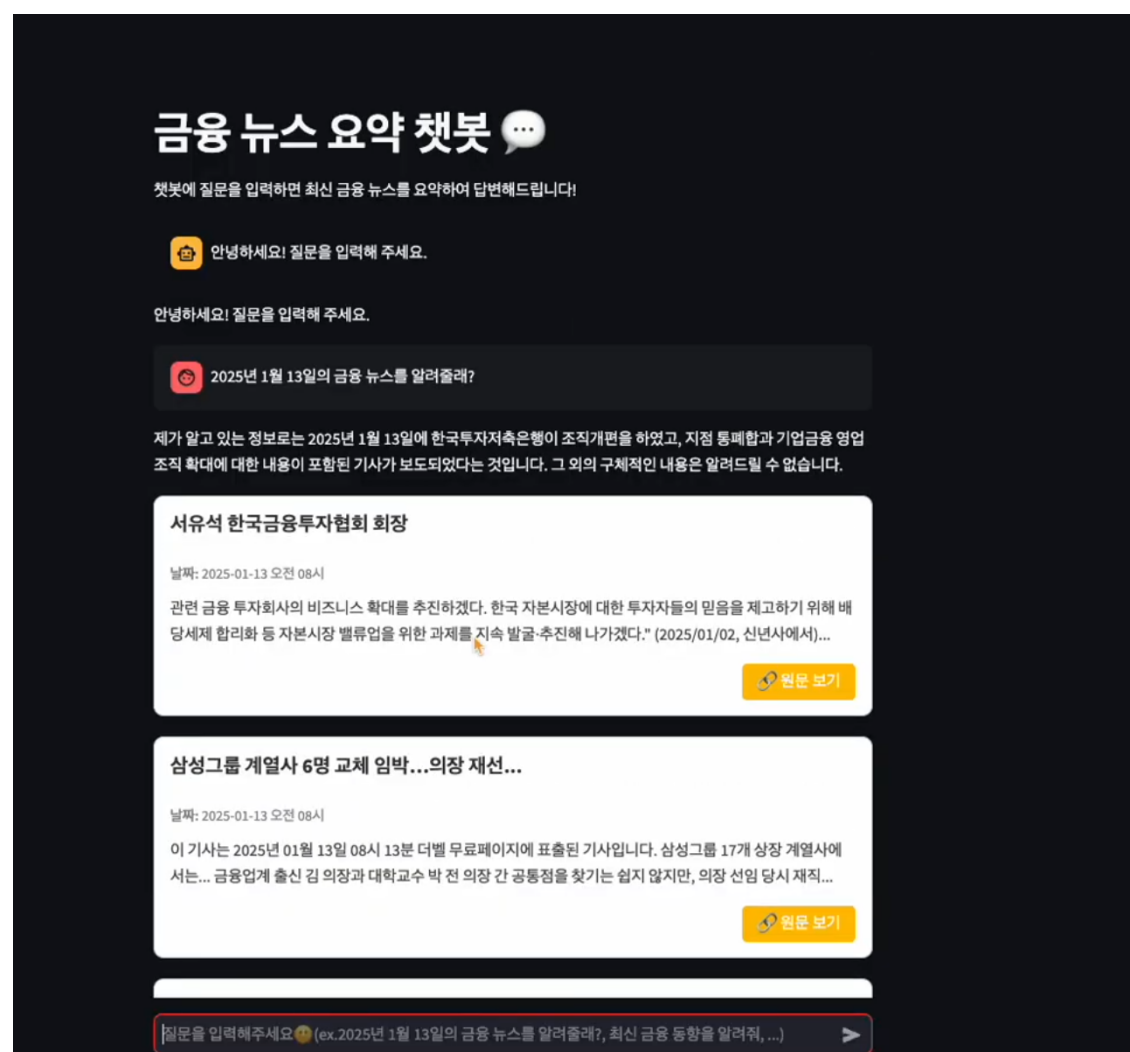
    st.session_state.messages.append({"role": "assistant", "content": combined_message})

except Exception as e:
    st.error(f"뉴스를 가져오는 중 오류가 발생했습니다: {e}")

# 전체 요약 출력
for msg in st.session_state.messages:
    if msg['role'] == 'assistant':
        st.markdown(msg['content'], unsafe_allow_html=True)
    else:
        st.chat_message(msg['role']).write(msg['content'])

```

0. 질문과 답변 형태



1. 날짜와 키워드 입력

<https://drive.google.com/file/d/1719QaBcQp-PKGfXXS4yVnlvrcikehDDg/view?usp=sharing>

2. 키워드만 입력

https://drive.google.com/file/d/1uKuRu1TQ8Bgl-EEChVPAXg_gqzuHzQuB/view?usp=sharing

3. 다른 날짜 뉴스 가져오기

https://drive.google.com/file/d/1gYtB40Mdd62TqjyAlK8R7eYN_zm8azDr/view?usp=sharing

4. 키워드 다양하게 적용하기

<https://drive.google.com/file/d/1kCmWT4C4i0leiPWBMMaa3CnDc7zjo05/view?usp=sharing>

5. 동일 질문 Ver.ChatGPT

https://drive.google.com/file/d/1_sXyNakBfDuoQw31y968rcqggwasz5G_/view?usp=sharing