

Agentic AI를 활용한 지역아동센터 수요 예측 서비스

프로젝트 수행 결과서



https://github.com/tpwnsdl123-bit/child_project

김세준

26.01.05

목차

01 프로젝트 기획서

- 기획의도
- 개발목표

02 사용 기술 목록/경험

03 개발 스케줄

04 요구사항 정의서 및 분석서

05 화면 설계서

06 UML(Usecase/Sequence)

07 주요 서비스 기능 및 소스 코드

- 1차 프로젝트
- 2차 프로젝트

08 머신러닝 결과 보고서

09 시연

10 향후 개발 계획

11 프로젝트 수행 소감

01 기획서 – 기획의도(1/2)

여성농업인신문 종합 농촌여성 농민단체 소식

Since2005

홈 > 농촌여성

아이돌봄서비스 수요 느는데 인력은 태부족

A 김수현 기자 | © 입력 2025.11.07 14:01 | 댓글 0

전국 아이돌봄센터 232개소,
돌보미수 2만9천여명
지난해 13만9천명 신청
평균 대기 기간 32.8일

사회 교육

초등 학부모 50% "수업 전후 돌봄 필요"...4년새 20%p 급증

2023년 범정부 온종일 돌봄 수요조사

김민제 기자

수정 2023-03-06 13:35 등록 2023-03-06 13:35

가족 형태 변화 + 맞벌이 증가로
방과후 돌봄 수요 확대

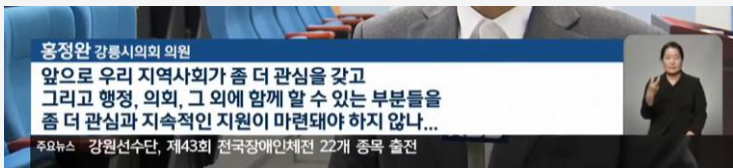
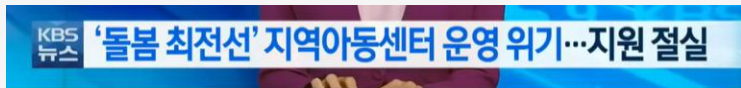


가정에서 아이 돌봄 감당하기 어려워
야간, 방과후 돌봄 공백 발생



“지역아동센터 이용자 수요를 예측할 필요”

01 기획서 – 기획의도(2/2)



돌봄 수요는 전국적으로 증가하지만
지원 여건은 지역마다 제각각



어느 지역은 수요 대비
예산·시설·인력이 부족



“자치구별 이용자 수를 예측해
정책 의사결정을 지원 ”

01 기획서 - 개발 목표

지역아동센터 수요 예측 모델을 활용한 대시보드 시각화 시스템

Input (한부모 가정 수, 인구수, 기초생활 수급자 수 등)
target (지역아동센터 이용자 수)

머신러닝 기반(Multiple Linear Regression, Ensemble 기법
XGBoost 등)으로 다양한 데이터 학습

KOSIS 국가통계포털, 서울열린데이터광장 데이터 사용

교차검증 기반 하이퍼파라미터 튜닝으로 최적 모델 선정
예) n_estimators, max_depth, learning_rate,
subsample, colsample_bytree

LLaMA3 베이스 모델을 활용한 인스트럭션 파인튜닝 시스템

(Instruction Tuning : Alpaca 데이터셋 기반)

Base Model 로드: Hugging Face에서 LLaMA3 베이스 모델/
토큰라이저 로드, Alpaca Clean 데이터셋 기반 학습 데이터
사용

파인튜닝 수행: LoRA/QLoRA 기반 파라미터 효율 학습으로
도메인 지시문 응답 성능 강화

체크포인트 관리: 학습 단계별 체크포인트 저장 및 검증을
통해 최적 모델 선택

LLaMA3를 활용한 RAG 기반 지역아동센터 Q&A 시스템

Generative AI(보고서 생성, 정책제안, 텍스트 요약)
RAG (문서 기반 Q&A)

RAG 파이프라인 구축: 정책·운영지침 PDF → 청크 분할 →
임베딩 → 벡터DB 저장 → 유사도 검색 후 근거 문서와 함께
응답 생성

근거 기반 응답 품질 강화: 프롬프트 템플릿/출력 형식 고정,
인용(근거) 포함, 환각 최소화를 위한 답변 가이드 적용

서빙/연동 구조: RunPod(FastAPI) 모델 추론 서버 ↔ Flask
UI 연동으로 실시간 질의응답 및 결과 출력

02 사용 기술 목록/경험

사용 기술 목록

Language



[Python]
[3.11]



[HTML5]



[JavaScript]



[CSS3]

Library



[Numpy]
[1.26.4]



[Pandas]
[2.3.3]



[Scikit-learn]

AI Framework



[Hugging Face]



LangChain

[LangChain]

Infra



[runpod]



[docker]

Database



[SQL Developer]



[Oracle]
[11g]



[ChromaDB]

FrameWork



[Flask]
[3.0.3]

02 사용 기술 목록/경험 (1/3)

Fine Tuning

Neural Network

- Meta Llama-3-8B-Base
- Transformer 기반 Causal Language Model

Adaptation Techniques

- LoRA Dropout(0.0) 적용
- PEFT/LoRA: Rank 16, Alpha 16 가중치 튜닝

Training & Checkpoint Strategy

- Batch: per_device_train_batch_size=2, gradient_accumulation_steps=8
- Steps: max_steps=300, warmup_steps=50
- 검증/저장: eval_steps=50, save_steps=100 (strategy='steps')
- Seed=42, report_to='none'

Optimization

- Optimizer: adamw_8bit, learning_rate=1e-4, weight_decay=0.01
- Scheduler: constant_with_warmup
- gradient_checkpointing=True (메모리 절감)

RAG

Knowledge Ingestion

- 복지/정책 PDF 문서 수집 및 폴더 단위 로딩(DirectoryLoader, PyPDFLoader)
- RecursiveCharacterTextSplitter 기반 청크 분할(문맥 보존)

Embedding & Vector

- jhgan/ko-sroberta-multitask 임베딩 적용
- Chroma VectorStore 구축 및 persist_directory로 영속화
- 기존 DB 존재 시 재인덱싱 없이 로드(초기 로딩 시간 절감)

Retrieval & Prompting

- 사용자 질문 → 관련 문서 Top-k 검색 후 컨텍스트 주입
- 규칙/수치 기반 답변 형식(3줄 불릿 등) 프롬프트 템플릿 적용
- 할루시네이션 완화(근거 없는 단정/절대값 금지 규칙 적용)

Intergration

- Flask서비스(GenAIService)에서 RAG 모듈 지연 로딩/캐싱
- LLM 응답 + 근거 컨텍스트를 UI에 함께 출력하도록 연동

Machine Learning

Supervised Learning

- Multiple Linear Regression / Random Forest / XGBoost 모델 비교·적용

Feature Engineering & Preprocessing

- 자치구 One-Hot Encoding + 결측치/이상치 처리, 학습 컬럼 스키마 고정

Model Interpretation

- RandomForest Feature Importance로 주요 영향 변수 분석

Model Evaluation

- MAE, RMSE 기반 모델 성능 비교 및 검증
- 모델별 결과 비교 후 성능/안정성 기준으로 모델 선정

Hyperparameter Tuning

- RandomizedSearchCV(n_iter=30, cv=3, scoring=R²)로 XGBoost 튜닝
- 2차 Local Search + log1p(target) 적용 후 expm1로 원복 평가

02 사용 기술 목록/경험 (2/3)

Flask / Backend

Architecture

- Blueprint 기반 모듈 분리 및 라우팅 설계
- ServiceLayer(GenAIService, RagService, predict등)로 로직 분리

API & Integration

- Flask JSON API 설계(요청/응답 스키마) 및 예외 처리
- RunPod(FastAPI) 추론 서버 연동: /generate, /switch_model 호출

GenAI Features

- 보고서/요약/규칙 기반 응답 등 프롬프트 템플릿 적용 및 결과 후처리
- 입력 파싱 → 필요한 데이터 조회/가공 → LLM 컨텍스트 구성 흐름 구현

Deployment

- 환경변수(.env) 기반 설정 분리 (RUNPOD_API_URL 등) 및 배포 환경 구성

Database

관계형 데이터베이스 개념

- Oracle과 DBMS 기본 이해
- 테이블, 기본키/외래키, 무결성 제약 조건
- 정규화(Normalization)를 통한 중복 최소화

Oracle 활용

- SELECT, JOIN, GROUP BY 등 작성
- 조건문/필터링을 통한 데이터 추출 및 집계

테이블 관리

- 머신러닝 학습용 테이블 생성 및 분할

Python 연동

- cx_Oracle을 활용한 DB ↔ Python 연계
- 학습데이터 추출 및 모델 입력용 데이터 프레임 변환

HTML / CSS / JS

UI composition

- Jinja2 템플릿 기반 화면 구성 및 공통 레이아웃(상단 메뉴/네비) 적용
- 생성형 AI 탭 포함 다중 기능 페이지 UI 설계

Async Communication

- Fetch API로 Flask 엔드포인트 호출 (JSON 송수신)
- 로딩 상태/에러 처리 등 사용자 인터랙션 흐름 구현

Visualization

- Chart.js로 예측/통계 데이터 시각화 (추세 그래프, 비교차트)
- 자치구/연도 조건 변경 시 그래프 동적 업데이트

02 사용 기술 목록/경험 (3/3)

Git

- GitHub 기반 브랜치 전략으로 기능 단위 개발/통합 (feature → main)
- PR/merge 중심으로 변경 이력 관리 및 팀 협업 흐름 운영
- 기능 단위 커밋 메시지 규칙 적용 (원인/변경/영향이 보이게)
- README/문서 정리로 실행 방법 및 결과 공유

Docker

- docker-compose로 Flask 앱 + Oracle DB 개발 환경 구성/실행
- 서비스 간 네트워크/포트 구성으로 로컬 통합 테스트 가능
- 실행 환경을 컨테이너로 표준화하여 "동일 환경 재현" 확보
- 환경 변수(.env) 기반 설정 분리로 로컬/배포 환경 대응

Cloud / RunPod

- RunPod GPU 인스턴스에서 Llama 3 실행 환경 구성 (CUDA/라이브러리 세팅)
- 추론 서버(FastAPI)로 모델 서빙 엔드포인트 제공
- 체크포인트/LoRA 어댑터 등 모델 산출물 버전 관리 및 교체 테스트
- 학습(파인튜닝)과 추론 서빙을 분리 운영하여 비용/안정성 고려

03 개발 스케줄 (1/2)

| 담당자 | 색상 |
|-----|----|
| 김진혁 | |
| 김세준 | |
| 공동 | |

| 구분 | 담당자 | 파트 | 날짜 | | 기간 | 12월 | | | | | 1월 | |
|---------|-----|--------------------------|------------|------------|----|-----|-----|-----|-----|-----|-----|-----|
| | | | | | | 1주차 | 2주차 | 3주차 | 4주차 | 5주차 | 1주차 | 2주차 |
| 프로젝트 기획 | 공동 | 아이디어 구상 | 2025-10-27 | 2025-10-27 | 1 | | | | | | | |
| | | 화면 목록 구성 | 2025-10-28 | 2025-10-28 | 1 | | | | | | | |
| | | Information Architecture | 2025-10-29 | 2025-10-30 | 2 | | | | | | | |
| 데이터 단계 | 공동 | 데이터 수집 (공공데이터) | 2025-10-31 | 2025-11-04 | 5 | | | | | | | |
| | | EDA(통계 시각화, 관계분석) | 2025-11-06 | 2025-11-07 | 2 | | | | | | | |
| | | 모델 피쳐 확정 | 2025-11-08 | 2025-11-08 | 1 | | | | | | | |
| | 김진혁 | 지원센터 이용자수 데이터 정제 | 2025-11-05 | 2025-11-05 | 1 | | | | | | | |
| | 김세준 | 변수 데이터 정제 | 2025-11-05 | 2025-11-05 | 1 | | | | | | | |
| 모델링 | 김진혁 | 베이스라인 모델 구축 | 2025-11-10 | 2025-11-11 | 2 | | | | | | | |
| | 김세준 | 트리 기반 모델 개발 | 2025-11-11 | 2025-11-12 | 2 | | | | | | | |
| | 공동 | 모델 성능 검증 및 비교 | 2025-11-13 | 2025-11-13 | 1 | | | | | | | |
| | | 파이프라인 구성 | 2025-11-14 | 2025-11-14 | 1 | | | | | | | |
| | | 트리 기반 모델 최종 선정 | 2025-11-17 | 2025-11-17 | 1 | | | | | | | |
| 시스템 설계 | 김세준 | Flask 구조 설계 | 2025-11-18 | 2025-11-19 | 2 | | | | | | | |
| | 김진혁 | DB 테이블 설계 | 2025-11-18 | 2025-11-19 | 2 | | | | | | | |
| | 공동 | API 설계 | 2025-11-19 | 2025-11-20 | 2 | | | | | | | |
| UI/UX | 공동 | 화면설계서 초안 작성 | 2025-11-20 | 2025-11-21 | 2 | | | | | | | |
| | | UI 시안 | 2025-11-24 | 2025-11-24 | 1 | | | | | | | |
| | 김세준 | 전체 화면 설계서 | 2025-11-25 | 2025-11-25 | 2 | | | | | | | |
| | 김진혁 | 플로우 다이어그램 | 2025-11-25 | 2025-11-26 | 2 | | | | | | | |
| 백엔드 | 김세준 | 통계 대시보드 API 개발 | 2025-11-27 | 2025-11-28 | 2 | | | | | | | |
| | | Q&A API | 2025-11-27 | 2025-11-28 | 2 | | | | | | | |
| | 김진혁 | 계정 로그인 / 회원가입 API | 2025-12-01 | 2025-12-01 | 1 | | | | | | | |
| | | 지역아동센터 수요 예측 API 개발 | 2025-12-01 | 2025-12-01 | 1 | | | | | | | |

03 개발 스케줄 (2/2)

| 담당자 | 색상 |
|-----|----|
| 김진혁 | |
| 김세준 | |
| 공동 | |

| 구분 | 담당자 | 파트 | 날짜 | | 기간 | 12월 | | | | | 1월 | |
|-------------|-----|--------------------------|------------|------------|----|-----|-----|-----|-----|-----|-----|-----|
| | | | | | | 1주차 | 2주차 | 3주차 | 4주차 | 5주차 | 1주차 | 2주차 |
| 백엔드 | 김세준 | 통계 대시보드 API 개발 | 2025-11-27 | 2025-11-28 | 2 | | | | | | | |
| | | Q&A API | 2025-11-27 | 2025-11-28 | 2 | | | | | | | |
| | 김진혁 | 계정 로그인 / 회원가입 API | 2025-12-01 | 2025-12-01 | 1 | | | | | | | |
| | | 지역아동센터 수요 예측 API 개발 | 2025-12-01 | 2025-12-01 | 1 | | | | | | | |
| 프론트엔드 | 김세준 | 메인 페이지 UI | 2025-12-02 | 2025-12-05 | 4 | | | | | | | |
| | | Q&A 페이지 | 2025-12-02 | 2025-12-03 | 2 | | | | | | | |
| | 김진혁 | 대시보드 UI | 2025-12-02 | 2025-12-03 | 2 | | | | | | | |
| | | 예측 결과 페이지 UI | 2025-12-04 | 2025-12-05 | 2 | | | | | | | |
| | | UI소개 페이지 UI | 2025-12-04 | 2025-12-05 | 2 | | | | | | | |
| 로컬 서비스 테스트 | 공동 | 프론트-백엔드 연동 및 최종 테스트 (로컬) | 2025-12-08 | 2025-12-09 | 2 | | | | | | | |
| 모델 탐색 및 선정 | 김진혁 | ai 서비스 기획 | 2025-12-10 | 2025-12-10 | 1 | | | | | | | |
| | | 기본 학습 실험(gpt2, bert 등) | 2025-12-10 | 2025-12-11 | 2 | | | | | | | |
| | | 베이스 모델 선정 및 UI prototype | 2025-12-11 | 2025-12-12 | 2 | | | | | | | |
| 파인튜닝 | 김진혁 | Llama3-8b 모델 파인튜닝 | 2025-12-15 | 2025-12-19 | 5 | | | | | | | |
| | | 모델 평가 및 최적화 (경량화) | 2025-12-15 | 2025-12-19 | 5 | | | | | | | |
| | | LLM 연동 API 개발 | 2025-12-17 | 2025-12-19 | 3 | | | | | | | |
| 서비스 통합 및 배포 | 김진혁 | 생성형 ai UI 개발 | 2025-12-22 | 2025-12-23 | 2 | | | | | | | |
| | | 배포 시스템 구축 | 2025-12-24 | 2025-12-26 | 3 | | | | | | | |
| | | 최종 성능 테스트 | 2025-12-29 | 2025-12-30 | 2 | | | | | | | |
| | | RunPod Serverless 배포 | 2025-12-29 | 2025-12-30 | 2 | | | | | | | |

04 요구사항 분석서 / 정의서

요구사항 분석서(1/2)

1. 메인 페이지 기능

1.1 메뉴 이동 기능

- 상단 메뉴 클릭 시 각 페이지로 이동할 수 있어야 한다.
- 상단 메뉴는 모든 페이지에서 동일한 레이아웃으로 고정되어야 한다.
- 로그인 상태일 경우 "로그아웃" 버튼이 표시되고 클릭 시 로그아웃 처리 후 메인 화면으로 이동해야 한다.

1.2 캐러셀 기능

- 메인 페이지에는 자동 재생되는 캐러셀이 있어야 한다.
- 캐러셀은 약 5 초 간격으로 자동 전환되어야 한다.
- 좌우 네비게이션 버튼 또는 인디케이터 클릭을 통해 수동 제어가 가능해야 한다.

1.3 스크롤 스냅 기능

- 메인 화면 스크롤 시 주요 섹션 단위로 화면이 자연스럽게 정렬 되도록 스크롤 스냅이 적용되어야 한다.
- 사용자 스크롤을 내리거나 올릴 때, 각 섹션이 부드럽게 전환되며 화면이 고정되어야 한다.

1.4 주요 서비스 이동 버튼 기능

- 메인 페이지에 서비스 이동 버튼(통계 대시보드, 머신러닝, QnA)이 제공되어야 한다.
- 버튼 클릭 시 해당 기능 페이지로 이동해야 한다.

2. 서비스 소개 기능

2.1 서비스 목적 소개 기능

- 서비스 목적이 첫 화면에서 명확히 전달되어야 한다.
- 예측 기반 서비스의 필요성과 사회적 활용성을 문구 및 시각 요소로 설명해야 한다.

2.2 데이터 기반 설명 기능

- 서비스가 실제 데이터 기반으로 동작한다는 안내가 포함되어야 한다.

3. 통계 대시보드 기능

3.1 연도·지역 필터 기능

- 사용자는 연도 범위 및 자치구를 선택하여 데이터를 조회할 수 있어야 한다.
- 필터 변경 시 새로고침 없이 동적으로 값이 갱신되어야 한다.

3.2 표 및 지표 표시 기능

- 이용자 수, 시설 수, 증감률 등 핵심 수치가 카드형 UI로 표시되어야 한다.
- 조회된 데이터는 표 및 그래프 형태로 시각화되어야 한다.

3.3 데이터 예외 처리 기능

- 조회 범위 내 데이터가 없을 경우 "데이터가 없습니다." 메시지를 표시해야 한다.

04 요구사항 분석서 / 정의서

요구사항 분석서(2/2)

4. 머신러닝 화면 기능

4.1 연도 및 자치구 선택 기능

- 사용자가 예측 값 조회를 위해 자치구 및 연도를 선택할 수 있어야 한다.
- 선택된 값은 즉시 화면에 반영되어야 한다.

4.2 예측 그래프 표시 기능

- 2015~2022 년 실제 데이터와 2023~2030 년 예측 데이터를 동일 그래프에서 비교하여 표시해야 한다.
- 그래프는 마우스 오버 시 수치가 강조 표시되어야 한다.
- 예측 그래프 밑에는 모델 성능 표시가 있어야 한다.

4.3 비교 분석 기능

- 서울 평균과 선택 구의 예측 값 비교 기능이 제공되어야 한다.
- 전년 대비 증감 수치 및 변화율(%) 정보가 포함되어야 한다.

5. 지도 연동 기능

5.1 SVG 지도 표시 기능

- 서울시 자치구 기반 SVG 지도 UI 가 제공되어야 한다.
- 자치구 영역은 Hover 및 선택 시 색상 변화가 적용되어야 한다.

5.2 선택 연동 기능

- 지도 선택, 드롭다운 선택 등 UI 요소 간 동기화가 이루어져야 한다.
- 선택된 구의 예측 분석 데이터가 우측 UI 에 업데이트되어야 한다.

6. QnA 게시판 기능

6.1 목록 조회 기능

- 사용자는 게시글 목록을 확인할 수 있어야 한다.
- 목록에는 제목, 작성자, 작성일, 댓글 수가 포함되어야 한다.

6.2 글 작성·수정·삭제 기능

- 로그인 사용자는 질문을 작성할 수 있어야 한다.
- 게시글 수정 및 삭제는 작성자 본인만 가능해야 한다.

6.3 댓글 기능

- 게시글에는 댓글 작성 기능이 제공되어야 한다.

7. 사용자 인증 기능

7.1 로그인 기능

- 사용자는 ID(또는 이메일)과 비밀번호로 로그인할 수 있어야 한다.
- 로그인 실패 시 오류 메시지가 표시되어야 한다.

7.2 회원가입 기능

- 사용자 정보 입력 후 회원가입이 가능해야 한다.
- 비밀번호는 암호화(HASH) 처리되어 저장되어야 한다.

7.3 비밀번호·아이디 찾기 기능

- 이메일 또는 인증 방식으로 정보 찾기 기능이 제공되어야 한다.

04 요구사항 정의서/ 분석서

요구사항 정의서(1/2)

Agentic AI를 활용한 지역아동센터 수요 예측 서비스

요구사항 명세서

| RQ-ID | 화면 명 | 요구사항 명 | 요구사항 상세 |
|------------|---------|----------------|---|
| RQ-ID-0001 | 메인 페이지 | 로고 | 웹 페이지 왼쪽 상단 클릭 시, 첫 화면으로 이동할 수 있는 로고를 삽입 |
| RQ-ID-0002 | 메인 페이지 | 상단 네비게이션 바 | 네비게이션 바 각 영역에 머신러닝/통계 대시보드/로그인/회원가입으로 구분되어 있고 클릭 시 해당 페이지로 이동 |
| RQ-ID-0003 | 메인 페이지 | 웹페이지 하단 | 웹 페이지 하단에 위치하며 서비스 이용 약관/개인정보 처리방침/정보 등을 포함 |
| RQ-ID-0004 | 메인 페이지 | 이미지 슬라이드(캐러셀) | 자동으로 전환하는 이미지를 페이지 중앙에 설정 |
| RQ-ID-0005 | 메인 페이지 | 스크롤 스냅 | 메인 페이지 내 스크롤 시 섹션 단위로 자동 정렬되며 스크롤 이동이 부드럽게 처리 |
| RQ-ID-0101 | 서비스 소개 | 메인 서비스 이동 버튼 | 주요 서비스 페이지로 빠르게 이동할 수 있는 버튼 제공, 클릭 시 해당 서비스 페이지로 이동 |
| RQ-ID-0102 | 서비스 소개 | 데이터 기반 안내 | 서비스가 실제 통계 기반 머신러닝 모델로 동작함을 사용자에게 안내 |
| RQ-ID-0201 | 통계 대시보드 | 연도/지역 필터 | 사용자는 연도 범위 및 자치구 선택을 통해 데이터를 조회 |
| RQ-ID-0202 | 통계 대시보드 | 자치구 검색 자동완성 기능 | 자치구 이름 일부 입력 시 일치/유사 자치구를 자동완성 목록으로 제공 |
| RQ-ID-0203 | 통계 대시보드 | 동적 데이터 갱신 | 필터 변경 시 새로고침 없이 값이 즉시 갱신 |
| RQ-ID-0204 | 통계 대시보드 | 카드형 지표 | 이용자 수·시설 수 지표가 카드 UI로 구성 |
| RQ-ID-0205 | 통계 대시보드 | 연도 범위 선택 제약 기능 | 종료 연도는 시작 연도와 같거나 이후 연도만 선택 가능 |
| RQ-ID-0206 | 통계 대시보드 | 데이터 예외 처리 | 선택된 기간에 데이터가 없을 경우 '데이터가 없습니다.' 메시지가 표시 |
| RQ-ID-0301 | 예측 화면 | 연도 및 자치구 선택 | 사용자가 예측 조회를 위해 자치구 및 연도를 선택 |
| RQ-ID-0302 | 예측 화면 | 예측 그래프 | 2015~2022 실제값과 2023~2030 예측값이 동일 그래프에서 비교 표시 |
| RQ-ID-0303 | 예측 화면 | 성능 정보 제공 | 예측 결과 하단에 모델 성능 정보 표시 |
| RQ-ID-0304 | 예측 화면 | 비교 분석 기능 | 서울 평균 대비 선택 지역의 예측 추세를 비교할 수 있는 기능이 포함 |
| RQ-ID-0401 | 지도 UI | SVG 지도 표시 | 서울시 기반 SVG 지도 UI가 화면에 표시 |
| RQ-ID-0402 | 지도 UI | 선택 연동 기능 | 특정 자치구 클릭 시 예측 데이터 및 그래프가 동기화 |
| RQ-ID-0501 | Q&A | 목록 조회 | 게시글 목록 조회 가능 |

04 요구사항 정의서/ 분석서

요구사항 정의서(2/2)

Agentic AI를 활용한 지역아동센터 수요 예측 서비스

요구사항 명세서

| RQ-ID | 화면 명 | 요구사항 명 | 요구사항 상세 |
|------------|-------|-----------------|--|
| RQ-ID-0502 | Q&A | 글 작성 | Q&A 게시판 목록 화면에서는 각 게시글의 번호, 제목, 작성일시가 표 형태로 함께 표시 |
| RQ-ID-0504 | Q&A | 덧글 가능 | 게시글에는 덧글 작성 및 표시 기능이 제공 |
| RQ-ID-0601 | 로그인 | 로그인 기능 | 사용자 아이디 + 비밀번호로 로그인 |
| RQ-ID-0602 | 로그인 | 오류 처리 | 로그인 실패 시 오류 메시지가 표시 |
| RQ-ID-0603 | 회원가입 | 회원 등록 | 사용자 정보 입력 후 회원가입 |
| RQ-ID-0604 | 회원가입 | 회원가입 입력값 유효성 검사 | 이메일 형식이 올바르지 않거나 비밀번호/비밀번호 확인이 일치하지 않을 경우 각각 오류 메시지를 표시 |
| RQ-ID-0605 | 회원가입 | 암호화 저장 | 비밀번호는 암호화(HASH) 처리되어 저장 |
| RQ-ID-0606 | 로그아웃 | 로그아웃 알림 메시지 | 사용자가 로그아웃할 경우, "로그아웃 되었습니다."와 같은 안내 내용을 팝업으로 표시 |
| RQ-ID-0607 | 인증 | 아이디/비밀번호 찾기 | 이메일 인증 기반으로 계정 정보 찾기 |
| RQ-ID-0701 | 생성형AI | AI 페이지 UI 구성 | 생성형 AI 전용 대시보드 및 채팅/리포트 출력 영역 구성 |
| RQ-ID-0702 | 생성형AI | 자치구별 분석 리포트 생성 | DB 예측 데이터를 컨텍스트로 활용하여 3줄 요약(요약, 요인, 데이터) 형태의 분석 결과 제공/제정 정보 찾기 |
| RQ-ID-0703 | 생성형AI | 정책 아이디어 제안 | 파인튜닝된 모델을 통해 자치구 상황에 맞는 3가지 정책 아이디어 리스트 도출 |
| RQ-ID-0704 | 생성형AI | 전문 Q&A | RAG를 참조하여 아동 복지 정책 관련 신뢰도 높은 질의응답 기능 제공 |
| RQ-ID-0705 | 생성형AI | 텍스트 요약 | 사용자가 입력한 복지 정책 원문이나 긴 텍스트를 경량 모델(KoBART)을 활용하여 핵심 내용 요약 제공 |
| RQ-ID-0706 | 생성형AI | 인프라 연동(FastAPI) | Flask 백엔드와 외부 FastAPI(RunPod) 추론 서버 간 비동기 API 통신 구현 |
| RQ-ID-0707 | 생성형AI | 모델 버전 비교 모드 | Base, Middle, Final 등 파인튜닝 단계별 모델 답변 비교 기능 |
| RQ-ID-0708 | 생성형AI | 추론 성능 로깅 | AI 답변 생성 시간 및 추론 성공 여부를 DB(Chat Log)에 기록 |


05 화면 설계서

메인화면(1/5)

| 화면 코드 | MA-01 | 화면 경로 | Main | 페이지 명 | 메인 페이지 (캐러셀) | 페이지 | 2 |
|-------|-------|-------|------|-------|--------------|-------------|---|
| | | | | | | Description | |
| | | | | | | 1 | <ul style="list-style-type: none"> 캐러셀 좌우 화살표 클릭 시 이전·다음 슬라이드로 이동 |
| | | | | | | 2 | <ul style="list-style-type: none"> 하단 인디케이터 하단 점(●) 클릭 시 해당 슬라이드로 이동 |
| | | | | | | 3 | <ul style="list-style-type: none"> Snap Scroll 마우스 휠 또는 스크롤 시 다음 섹션으로 자동 이동 |
| | | | | | | 4 | <ul style="list-style-type: none"> 같은 페이지 내 '서비스 소개(스냅 스크롤-1)' 섹션으로 자동 스크롤 |

05 화면 설계서

머신러닝 페이지(1/2)

| 화면 코드 | ML-02 | 화면 경로 | Main - ML | 페이지 명 | 머신러닝 | 페이지 | 8 |
|--|-------|-------|-----------|-------|------|-------------|--|
| | | | | | | Description | |
|  <p>1. Year and District selection</p> <p>2. Map of Seoul districts</p> <p>3. Line chart showing user trends</p> <p>4. Summary table of indicators</p> | | | | | | 1 | <ul style="list-style-type: none"> 원하는 연도·자치구 선택 후 조회 시 지도·그래프·표가 동시에 갱신 페이지 최초 진입 시 기본값 : 최신 연도, 서울시 전체 |
| | | | | | | 2 | <ul style="list-style-type: none"> 지도에서 자치구 클릭 시 조회 (선택 구 색상 강조) |
| | | | | | | 3 | <ul style="list-style-type: none"> 조회 조건의 실제·예측 이용자 수 그래프 전년·서울 평균 대비 증감 표시 마우스 오버 시 연도별 실제/예측 값과 증감률 툴팁 표시 |
| | | | | | | 4 | <ul style="list-style-type: none"> 선택 연도·자치구의 예측 이용자 수와 주요 지표 표 |

05 화면 설계서

생성형 AI 페이지(1/5)

| 화면 코드 | GEN-01 | 화면 경로 | Main - GEN | 페이지 명 | GEN | 페이지 | 10 |
|--|--------|-------|------------|-------|-----|-------------|--|
| | | | | | | Description | |
|  | | | | | | 1 | <ul style="list-style-type: none"> LoRA 체크포인트 선택 영역을 표시한다 Base/cp100/cp200/최종 모델 버전을 선택할 수 있다. 모델 선택 시 서버에 모델 교체 요청을 전송하고, 이후 생성 결과에 반영한다 |
| | | | | | | 2 | <ul style="list-style-type: none"> 생성형 AI 기능 메뉴를 표시한다. 메뉴 클릭 시 해당 기능 화면으로 이동한다. |
| | | | | | | 3 | <ul style="list-style-type: none"> 보고서 생성에 필요한 자치구/기간 입력 폼을 표시한다. 사용자가 값을 입력하면 입력값을 기준으로 보고서 생성 요청을 준비한다. |
| | | | | | | 3 | <ul style="list-style-type: none"> 버튼 클릭 시 입력값을 전송하여 요약 보고서 생성 요청을 수행하고 결과를 화면에 출력한다 |

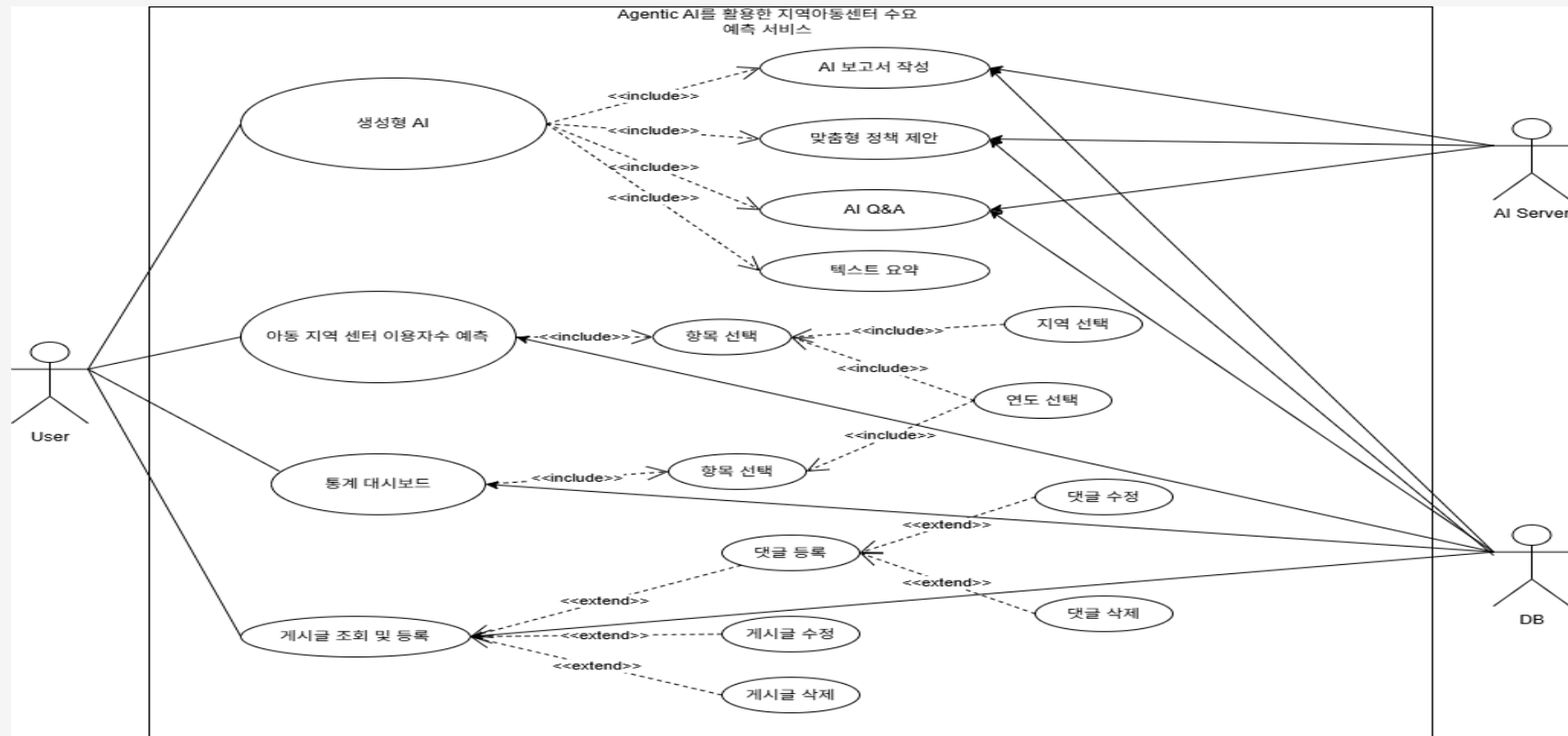
05 화면 설계서

생성형 AI 페이지(5/5)

| 화면 코드 | GEN-05 | 화면 경로 | Main – GEN | 페이지 명 | GEN | 페이지 | 14 |
|--|--------|-------|------------|-------|-----|-------------|--|
| | | | | | | Description | |
|  | | | | | | 1 | <ul style="list-style-type: none"> 생성형 AI 기능의 설정 모달을 표시한다. 모달 내에서 추론/학습 관련 옵션을 확인, 조정할 수 있다. |
| | | | | | | 2 | <ul style="list-style-type: none"> Temperature, Max Token 입력값을 표시한다. 사용자가 값을 변경하면 생성 결과의 창의성과, 출력길이에 반영된다. |
| | | | | | | 3 | <ul style="list-style-type: none"> LoRA 학습에 사용된 Training Arguments(학습 파라미터)를 표시한다. 사용자가 값을 변경하면 생성 결과의 창의성과, 출력길이에 반영된다. |
| | | | | | | 4 | <ul style="list-style-type: none"> 닫기 / 설정 적용 버튼을 표시한다. 설정 적용 클릭 시 변경된 값을 서버로 전송하여 설정을 저장하고, 이후 생성 요청에 적용한다. |

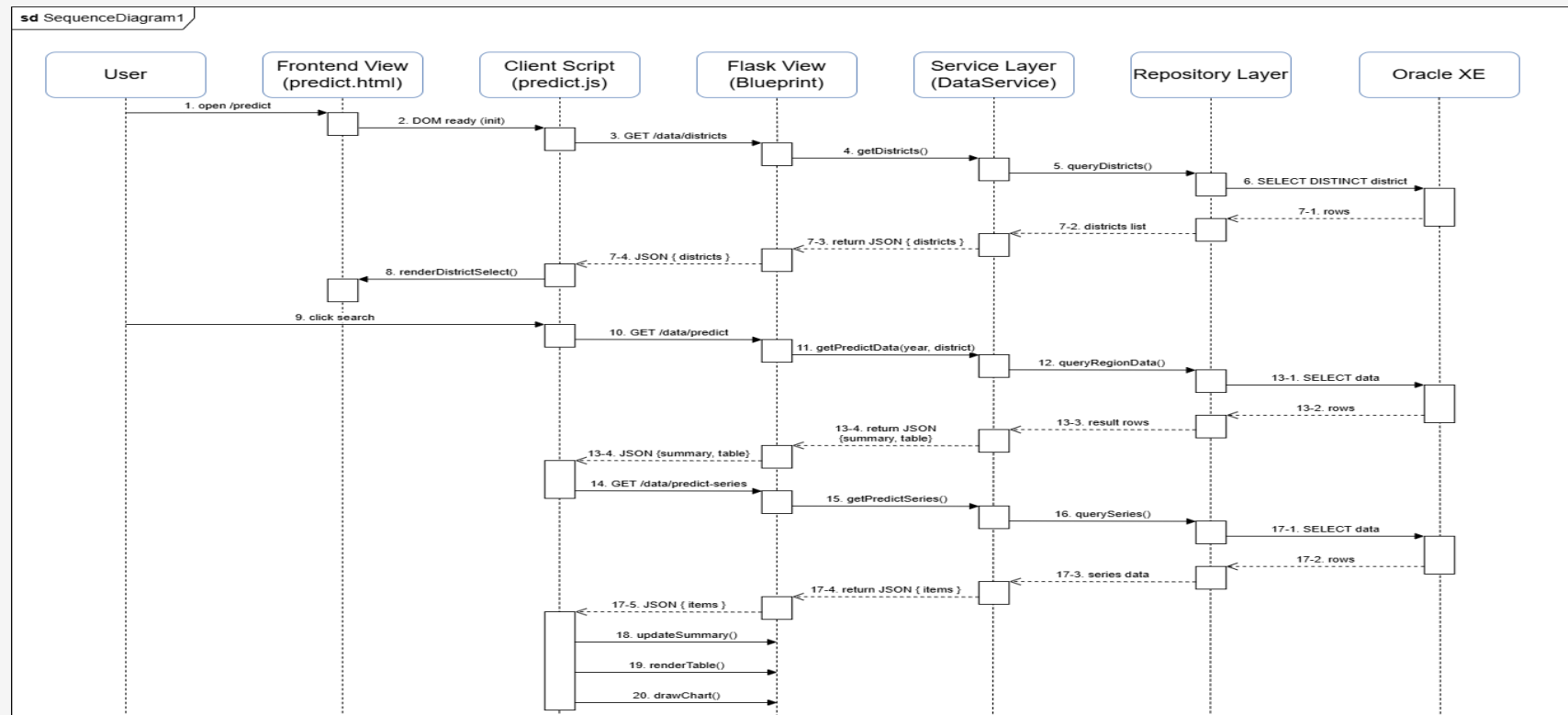
06 UML

Usecase Diagram



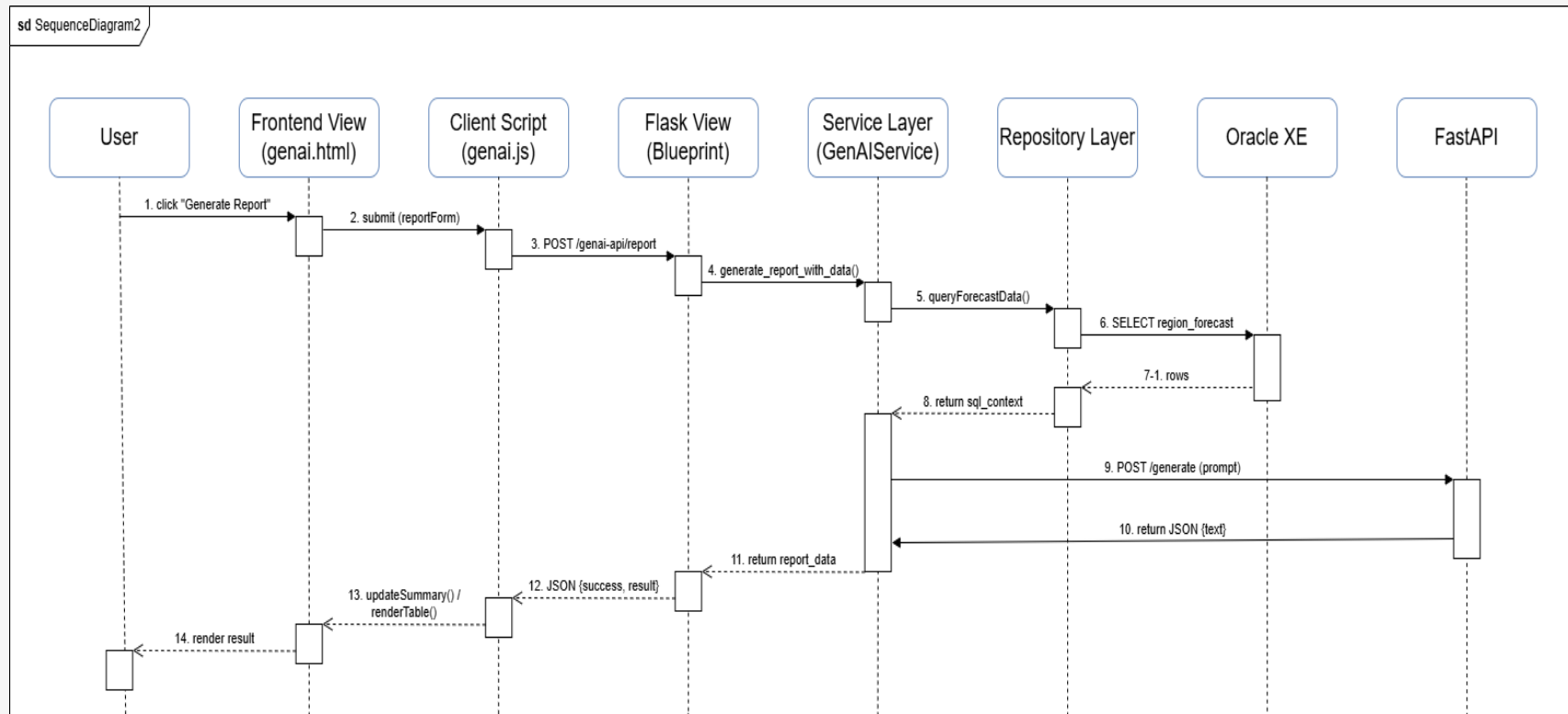
06 UML

Sequence Diagram (1/2)



06 UML

Sequence Diagram (2/2)



07 주요 서비스 기능 및 코드 (1/6)

```
# district 원핫 인코딩 [train.py]
df = pd.get_dummies(df, columns=["district"], drop_first=False)

district_oh_cols = [c for c in df.columns if c.startswith("district_")]
```

- district 범주형 변수를 pd.get_dummies()로 원-핫 인코딩해 district_로 시작하는 더미 컬럼들을 생성하고 목록(district_oh_cols)으로 저장한다.

```
param_grid_local = { [train.py]
    "max_depth": [4, 5, 6],
    "learning_rate": [0.03, 0.05, 0.07],
    "n_estimators": [600, 700, 800],
    "subsample": [0.5, 0.6, 0.7],
    "colsample_bytree": [0.5, 0.6, 0.7],
    "gamma": [0.1, 0.3, 0.5],
    "reg_lambda": [1.0, 1.5, 2.0],
    "reg_alpha": [0, 0.1, 0.3]
}

xgb_model_local = XGBRegressor(
    random_state=42,
    tree_method="hist"
)
```

- XGBRegressor를 tree_method="hist"로 빠르게 학습시키면서, 깊이/학습률/트리수/샘플링/정규화 등 주요 하이퍼파라미터의 탐색 범위(param_grid_local)를 정의한 코드다.

```
# Feature 설정 [train.py]
base_features = [
    "year",
    "single_parent",
    "basic_beneficiaries",
    "multicultural_hh",
    "academy_cnt",
    "grdp",
    "population"
]

features = base_features + district_oh_cols
target = "child_user"

# Train/Test Split
train = df[df["year"] <= 2020]
test = df[df["year"] >= 2021]

X_train = train[features]
y_train = train[target]

X_test = test[features]
y_test = test[target]

y_train_log=np.log1p(y_train)
```

- 학습 피쳐(features)와 타겟(child_user)을 정의한 뒤, 2020년까지 train / 2021년 이후 test로 연도 기준 분리하고 X/y를 각각 생성한다.
- 또한 y_train에 log1p를 적용해 스케일을 완화하고 학습 안정성을 높인다.

07 주요 서비스 기능 및 코드 (2/6)

```
[train.py]
search_local = RandomizedSearchCV(
    estimator=xgb_model_local,
    param_distributions=param_grid_local,
    n_iter=30,
    scoring="r2",
    cv=3,
    verbose=2,
    n_jobs=-1,
    random_state=42
)

search_local.fit(X_train, y_train_log)

best_xgb_local = XGBRegressor(
    **search_local.best_params_,
    random_state=42
)
best_xgb_local.fit(X_train, y_train_log)

best_xgb_local.district_ohe_cols = district_ohe_cols
best_xgb_local.base_features = base_features

pred_local_log = best_xgb_local.predict(X_test)
pred_local = np.expm1(pred_local_log)
```

- RandomizedSearchCV로 XGBoost 하이퍼파라미터를 30회 랜덤 탐색($n_iter=30$) 하고, 3-fold CV($cv=3$) 에서 R^2 ($scoring="r2"$) 기준으로 최적 조합을 찾는다
- 찾은 best_params_로 best_xgb_local 모델을 다시 생성해 전체 학습 데이터(X_train , y_train_log) 로 재학습한다.
- 모델 객체에 district_ohe_cols, base_features를 속성으로 저장해 추론 시 컬럼 스키마를 고정하고, 예측은 log 스케일로 나온 값을 expm1로 원래 스케일로 복원해 pred_local을 만든다.

```
class DataService:
    def get_predict_series(self, district: str) -> dict:
        else:
            actual_rows = self.region_repo.get_total_series_actual()
            for r in actual_rows:
                if r.child_user is None:
                    continue
                items.append({
                    "year": int(r.year),
                    "child_user": int(r.child_user),
                    "is_pred": False,
                })

            pred_rows = self.region_repo.get_total_series_forecast()
            for r in pred_rows:
                if r.child_user is None:
                    continue
                items.append({
                    "year": int(r.year),
                    "child_user": int(r.child_user),
                    "is_pred": True,
                })

            items.sort(key=lambda x: x["year"]) # 마지막으로 순서 점검 year기준으로 정렬
```

- DB에서 실제값 시계열(get_total_series_actual) 과 예측값 시계열(get_total_series_forecast) 을 각각 조회해, year / child_user / is_pred 형태의 리스트로 합친다.
- child_user가 없는 행은 건너뛰고(continue), 실제 데이터는 is_pred=False, 예측 데이터는 is_pred=True로 구분해 프론트(차트)에서 구간을 나눠 그릴 수 있게 한다.
- 마지막에 year 기준으로 정렬해 시간 순서가 보장된 시계열 데이터를 반환한다.

07 주요 서비스 기능 및 코드 (3/6)

```
# Quantization Setup
quantization_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_compute_dtype=torch.bfloat16,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type='nf4'
)
model_id = "meta-llama/Meta-Llama-3-8B"
model = AutoModelForCausalLM.from_pretrained(
    model_id,
    quantization_config=quantization_config,
    device_map={"": 0}
)
```

[train.py]

- Llama-3 8B 모델을 4-bit(NF4) 양자화 + double quant 설정으로 로드해 GPU 메모리 사용량을 크게 줄인 구성이다.
- 연산은 bfloat16으로 수행해 안정성을 유지하고, device_map={"":0}로 GPU 0번에 모델을 배치한다.

```
# LoRA Setup
peft_config = LoraConfig(
    lora_alpha=16,
    lora_dropout=0,
    r=16,
    bias="none",
    task_type="CAUSAL_LM",
    target_modules=["q_proj", "v_proj", "k_proj", "o_proj", "gate_proj", "up_proj", "down_proj"]
)
```

- LoRA(PEFT)로 Llama-3를 전체 재학습 없이 미세조정하기 위해 r=16, alpha=16, dropout=0 설정을 적용하고 bias는 학습하지 않는다.
- q/k/v/o_proj와 gate/up/down/proj 등 Attention-FFN 핵심 projection 레이어만 타깃으로 어댑터를 붙여 파라미터 효율을 높인다

```
# Training Arguments (300 steps)
output_dir = "/workspace/finetune/outputs"
training_arguments = TrainingArguments(
    output_dir=output_dir,
    report_to="none",
    per_device_train_batch_size=2,
    gradient_accumulation_steps=8,
    warmup_steps=50,
    max_steps=300,
    eval_steps=50,
    save_steps=100,
    evaluation_strategy='steps',
    save_strategy='steps',
    learning_rate=1e-4,
    logging_steps=1,
    optim="adamw_8bit",
    weight_decay=0.01,
    lr_scheduler_type="constant_with_warmup",
    seed=42,
    gradient_checkpointing=True,
    gradient_checkpointing_kwargs={'use_reentrant': True}
)
```

- per_device_train_batch_size=2 : GPU 메모리 한계로 마이크로배치 최소화
- gradient_accumulation_steps=8 : 유효 배치=2×8=16으로 안정성 확보
- max_steps=300 : 짧은 실험 주기로 빠른 검증/비교
- warmup_steps=50 : 초반 발산 방지(학습률 점진 상승)
- learning_rate=1e-4 : LoRA 튜닝에서 안정적인 기본 학습률
- optim=adamw_8bit : 옵티마이저 메모리 절감(저사양 GPU 대응)
- weight_decay=0.01 : 과적합 완화(일반화 보조)
- lr_scheduler_type=constant_with_warmup : warmup 이후 일정 LR로 간단/안정
- gradient_checkpointing=True : activation 메모리 절감(속도 ↓, 메모리 ↑)
- eval_steps=50 / save_steps=100 : 50 step마다 성능 점검, 100 step마다 저장
- seed=42 / report_to='none' : 재현성 확보 / 불필요 로깅 제거

07 주요 서비스 기능 및 코드 (4/6)

1.2 구성 요소 역할

genai_views.py (Flask Blueprint): /genai-api/* 엔드포인트

(report/policy/qa/summarize/switch-model/config)를 제공하며, 요청값 검증 후 서비스 레이어 호출 결과를 JSON으로 반환한다.

genai_service.py (서비스 레이어): 보고서/정책/QA/요약 기능의 핵심 로직을 담당한다. DB 컨텍스트 구성, 프롬프트 생성, RunPod 추론 서버 호출, 응답 가공을 수행한다.

rag_service.py (RAG): jsonl 규칙/수치 데이터를 Document로 변환하여 Chroma에 적재/로드하고, 질문 유형(doc_type)에 따라 필터링된 유사도 검색으로 근거 컨텍스트를 생성한다.

server.py (RunPod FastAPI): Llama-3 8B 4bit 로딩 및 LoRA 체크포인트 로딩을 수행한다. /switch_model로 활성 모델을 교체하고 /generate로 추론 결과를 제공한다.

genai.js (프론트 연동): 모델 비교 라디오 선택 시 Flask의 /genai-api/switch-model을 호출한다. 보고서/정책/QA/요약 요청을 fetch로 전송하고 로딩 스피너/버튼 비활성화 등 UX 처리를 수행한다.

```
@app.on_event("startup")
async def load_initial_model():
    global tokenizer, base_model, current_ft_model
    print("Loading Base Model and Default Adapter (300 steps)...")

    bnb_config = BitsAndBytesConfig(
        load_in_4bit=True,
        bnb_4bit_compute_dtype=torch.bfloat16,
        bnb_4bit_use_double_quant=True,
        bnb_4bit_quant_type="nf4",
    )

    tokenizer = AutoTokenizer.from_pretrained(BASE_MODEL_ID)
    tokenizer.pad_token = tokenizer.eos_token

    base_model = AutoModelForCausalLM.from_pretrained(
        BASE_MODEL_ID,
        quantization_config=bnb_config,
        device_map={"": 0},
    )

    current_ft_model = PeftModel.from_pretrained(base_model, os.path.join(OUTPUT_DIR, "checkpoint-300"))
    print("Default Server Ready (Final Model).")
```

[server.py]

- FastAPI 서버가 시작될 때(startup) 토크나이저를 초기화하고, Llama-3 Base 모델을 4-bit(NF4) 양자화 설정으로 로드해 GPU0에 올린다.
- pad_token을 eos_token으로 맞춰 패딩 토큰 문제를 방지하고, 입력 처리 일관성을 확보한다.
- 마지막으로 checkpoint-300의 LoRA 어댑터(PEFT)를 base 모델에 결합해 기본 추론 모델(current_ft_model)로 설정한 뒤 준비 완료 로그를 출력한다.

07 주요 서비스 기능 및 코드 (5/6)

```
@app.post("/generate")
def generate(req: GenerateRequest):
    global current_ft_model, tokenizer

    device = next(current_ft_model.parameters()).device
    print("model device:", device, "cuda avail:", torch.cuda.is_available())
    print("req.max_new_tokens:", req.max_new_tokens, "req.temperature:", req.temperature)

    current_ft_model.eval()

    prompt = (
        f"Below is an instruction that describes a task. "
        f"Write a response that appropriately completes the request.\n\n"
        f"### Instruction:\n{req.instruction}\n\n"
        f"### Input:\n{req.input}\n\n"
        f"### Response:\n"
    )

    inputs = tokenizer(prompt, return_tensors="pt")
    print("input(before):", inputs["input_ids"].device, "len:", inputs["input_ids"].shape[-1])

    inputs = {k: v.to(device) for k, v in inputs.items()}
    print("input(after):", inputs["input_ids"].device)

    if device.type == "cuda":
        torch.cuda.synchronize()
        t0 = time.time()

    with torch.inference_mode():
        output_tokens = current_ft_model.generate(
            **inputs,
            max_new_tokens=req.max_new_tokens,
            temperature=req.temperature,
            do_sample=(req.temperature > 0),
            repetition_penalty=1.2,
            pad_token_id=tokenizer.eos_token_id,
            eos_token_id=tokenizer.eos_token_id
        )

    if device.type == "cuda":
        torch.cuda.synchronize()
        t1 = time.time()
        new_tokens = output_tokens.shape[-1] - inputs["input_ids"].shape[-1]
        print(f"[perf] gen_time={t1-t0:.2f}s new_tokens={new_tokens} tok/s={new_tokens/((t1-t0+1e-6):.2f}")

    text = tokenizer.decode(
        output_tokens[0][inputs["input_ids"].shape[-1]:],
        skip_special_tokens=True
    ).strip()

    return {"text": text}
```

[server.py]

- 요청으로 받은 instruction/input을 프롬프트로 합친 뒤 토큰나이징해서 모델 디바이스 (GPU/CPU)로 올리고, generate()로 지정된 max_new_tokens/temperature 설정에 따라 응답 토큰을 생성한다.
- 생성 시간-토큰 처리량을 로그로 찍고, 입력 길이 이후의 생성 부분만 디코딩해 최종 텍스트를 JSON({"text": ...})으로 반환한다.

```
class RagService:
    @staticmethod
    def _route_doc_type(question: str) -> Optional[str]:
        q = (question or "").lower()
        if any(k in q for k in ["인건비", "급여", "호봉", "수당", "보수", "연봉", "돈", "월급"]):
            return "salary"
        if any(k in q for k in ["지원", "보조금", "운영비", "배치기준", "정원", "시설장", "생활복지사"]):
            return "support"
        if any(k in q for k in ["법", "조문", "시행령", "시행규칙", "아동복지법"]):
            return "law"
        return None

    def get_relevant_context(self, question: str) -> str:
        if not self.vector_db:
            return "참조할 수 있는 운영 지침 데이터가 없습니다."

        doc_type = self._route_doc_type(question)

        # MMR 대신 속도가 빠른 similarity 검색 사용
        search_kwargs: Dict[str, Any] = {"k": 3} # 검색 결과 개수를 3개로 최적화

        if doc_type:
            search_kwargs["filter"] = {"doc_type": doc_type}

        # search_type을 similarity로 변경하여 연산 속도 향상
        retriever = self.vector_db.as_retriever(
            search_type="similarity",
            search_kwargs=search_kwargs
        )

        # 랭체인 invoke 메서드 사용
        docs = retriever.invoke(question)
```

[rag_service.py]

- 질문 문자열에 포함된 키워드를 기준으로 문서 타입을 급여(salary) / 지원(support) / 법령(law) 으로 분류해, 검색 범위를 좁히는 라우팅 로직이다.
- 벡터DB가 없으면 안내 문구를 반환하고, 있으면 k=3으로 Top-3 유사도(similarity) 검색을 수행한다(MMR 대신 속도 우선).
- 분류된 doc_type이 있으면 해당 타입으로 filter를 걸어 관련 문서만 가져오고, retriever.invoke(question)으로 컨텍스트 문서를 추출한다.

07 주요 서비스 기능 및 코드 (6/6)

```
# 모델 변경
@bp.route("/switch-model", methods=["POST"])
def switch_model():
    data = request.get_json()
    try:
        # 런포드 서버의 /switch_model 엔드포인트 호출
        runpod_url = os.getenv("RUNPOD_API_URL").replace("/generate", "/switch_model")
        res = requests.post(runpod_url, json=data, timeout=60)
        return jsonify({"success": True, "result": res.json()})
    except Exception as e:
        return jsonify({"success": False, "error": str(e)}), 500
```

[genai_views.py]

- flask의 /switch-model POST API로, 클라이언트가 보낸 JSON을 받아 RunPod 추론 서버의 /switch_model 엔드포인트로 그대로 전달한다.
- RUNPOD_API_URL의 /generate를 /switch_model로 바꿔 호출하고, 응답 JSON을 success/result 형태로 반환한다(타임아웃 60초).
- 호출 실패 시 예외를 잡아 success:false와 에러 메시지를 500으로 반환한다.

```
class GenAIService:
    def generate_report_with_data(self, user_prompt: str, **kwargs) -> str:
        start_all = time.time()
        print(f"\n[로그 1] 함수 진입 완료: {time.time() - start_all:.4f}s") #

        meta = self._extract_query_meta(user_prompt)
        meta.district = kwargs.get('district', meta.district)
        meta.end_year = kwargs.get('end_year', meta.end_year)
        meta.start_year = kwargs.get('start_year', 2023)
        print(f"[로그 2] 메타데이터 추출 완료: {time.time() - start_all:.4f}s") #

        print(f"[로그 3] DB 조회 시작..." #
        sql_context = self._build_forecast_context(meta)
        print(f"[로그 4] DB 조회 및 가공 완료: {time.time() - start_all:.4f}s") #

        instruction = (
            "너는 서울시 아동복지 정책 전문가다. 반드시 한국어로 답해.\n"
            "아래 형식을 정확히 지켜. (약 3줄만, 추가 문장/설명 금지)\n"
            "- 요약: 한 문장\n"
            "- 가능 요인: 한 문장\n"
            "- 추가 데이터: 한 문장\n"
        )
        input_text = f"지역:{meta.district}\n기간:{meta.start_year}-{meta.end_year}\n데이터:\n{sql_context}"

        print(f"[로그 5] 런포드 요청 직전: {time.time() - start_all:.4f}s") #

        raw_response = self._call_llama3(
            instruction,
            input_text,
            max_tokens=128,
            model_version=kwargs.get('model_version', 'final')
        )
```

[genai_service.py]

- 사용자의 프롬프트를 self._extract_query_meta()로 파싱해 자치구/연도 범위 같은 메타정보를 추출하고, 단계별 처리 시간을 로그로 기록한다.
- 추출된 조건을 기반으로 DB 조회를 수행해 sql_context(예측/통계 등 필요한 데이터 컨텍스트)를 만들고, 이를 LLM 입력에 포함시킨다.
- instruction에 출력 형식(예: 3줄 불릿, 단정 금지, 추가 설명 금지 등)을 강하게 지정해 답변 스타일을 통제한다.
- 최종적으로 self._call_llama3()를 호출해 DB 컨텍스트 + 사용자 질의를 합친 요청으로 Llama3 응답을 생성하고 반환한다.

08 머신러닝 결과보고서 (1/6)

서론

1. 분석 배경 및 기획 의도

1.1 분석 배경

서울시는 지역아동센터를 통해 방과 후 돌봄, 학습 지원, 정서 안정 프로그램 등을 제공하고 있다. 특히 한부모 가정, 기초생활수급 가구, 다문화 가정 비율이 높은 자치구일수록 지역아동센터 수요가 높게 나타나는 경향이 있다.

그러나 자치구별 인구 구조, 가구 특성, 소득 수준, 학원 시설 분포는 지속적으로 변하고 있으며, 이에 따라 수요 격차도 확대되고 있다. 그럼에도 시설 확충과 예산 편성은 직전 연도 이용 실적이나 현재 센터 수 등 제한된 기준에 의존하는 경우가 많아, “어디에 얼마나 수요가 늘어날지”를 선제적으로 반영하기 어렵다.

또한 자치구 간 여건 차이가 큰 환경에서는 과거 실적만으로 향후 수요 변화를 충분히 설명하기 어렵고, 현재 이용자 수가 많은 지역 중심으로 자원을 배분할 경우 수요 증가가 빠른 취약 지역을 놓칠 위험이 존재한다.

1.2 기획 의도

본 분석은 서울시 25개 자치구의 과거 데이터를 기반으로 지역아동센터 이용자 수 변화를 분석하고, 2023~2030년 자치구별 수요를 예측하는 것을 목표로 한다. 한부모 가정 수, 기초생활수급 가구 수, 다문화 가구 수, 인구, 학원 수, GRDP 등 여러 지표를 함께 고려해 자치구별 수요 변화 속도를 수치로 제시하고, 예산 편성, 센터 신·증설, 인력 배치 등 정책 의사결정에 참고 가능한 근거를 제공하고자 한다.

2. 분석 목표

2.1 사용데이터 및 분석 범위

KOSIS(국가통계포털)와 서울열린데이터광장에서 제공하는 통계를 수집하여, 2015~2022년 서울시 25개 자치구 데이터를 구축하였다. 연도·자치구별 로 지역아동센터 이용자 수(child_user), 한부모 가정 수(single_parent), 기

초생활수급 가구 수(basic_beneficiaries), 다문화 가구 수(multicultural_hh), 사설 학원 수(academy_cnt), GRDP(1인당 지역총소득), 인구(population)를 정리하고 master_2015_2022.csv로 병합하였다.

2.2 예측 모델 구성 및 단계적 예측 수행

변수 분포와 상관관계를 확인하는 EDA를 수행한 뒤 결측치·이상치를 점검하고, 최종 변수를 확정하였다. 이후 연도 기준으로 학습(2015~2020)과 테스트(2021~2022) 구간을 분리하여 선형 회귀와 XGBoost를 비교·평가하고 최종 모델을 선정하였다. 마지막으로 2015~2022년 변화 추세를 바탕으로 2023~2030년 설명변수를 추정해 입력하고, 자치구별 예측 결과를 연도별로 정리하여 수요 증가 지역을 해석 가능한 형태로 제시하였다.

2.3 모델링 및 변수 선정

자치구는 범주형 변수이므로 원-핫 인코딩을 적용해 district_강남구와 같은 더미 변수를 생성하였다. 변수 선정 기준은 수요와의 직접 연관성, 2015~2022년 기간 동안 변동성이 있어 학습에 기여하는지 여부이며, 해석이 명확한 변수 중심으로 최종 피처를 구성하였다.

2.4 모델 성능 평가 지표

모델 성능 평가는 결정계수(R^2)와 평균제곱근오차(RMSE)를 사용하였다. R^2 는 설명력을, RMSE는 “평균적으로 몇 명 정도 오차가 나는지”를 보여준다. 2015~2020년 학습, 2021~2022년 테스트 조건에서 최종 XGBoost 모델은 테스트 구간 R^2 가 약 0.84 수준으로 나타났으며, RMSE는 다중 선형 회귀 모델과 유사한 수준이었다.

2.5 결과 활용 및 기대 효과

본 분석은 자치구별 지역아동센터 수요를 연도별로 정량화하고 향후 변화를 미리 점검할 수 있도록 정리한 것이다. 이를 통해 수요 증가가 예상되는 자치구에 예산·시설을 우선 배치하거나, 현재 수요는 낮지만 향후 증가 가능성이 큰 지역을 조기에 파악하는 데 활용할 수 있다.

08 머신러닝 결과보고서 (2/6)

본론

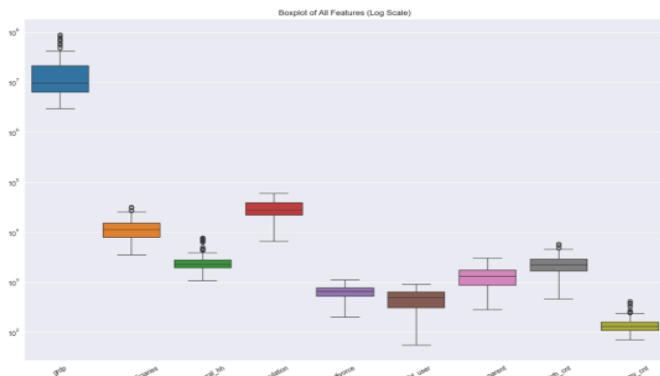
1. EDA

1.1 데이터 개요

```
import pandas as pd
df_master = pd.read_csv(r"C:\flaskteam_projects\flask_basic\data\master_2015_2022.csv")
df_master.head()
```

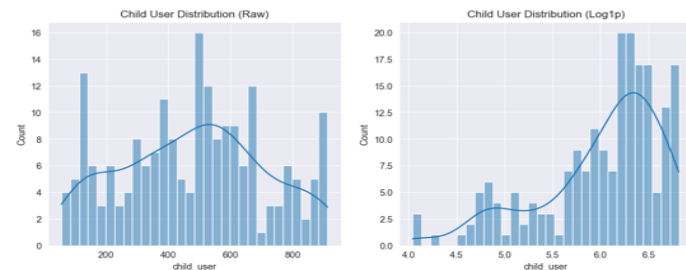
| | district | year | gdp | basic_beneficiaries | multicultural_jh | population | divorce | child_facility | child_user | single_parent | birth_cnt | academy_cnt |
|---|----------|------|----------|---------------------|------------------|------------|---------|----------------|------------|---------------|-----------|-------------|
| 0 | 동대문구 | 2015 | 26762944 | 2540 | 1221 | 11009 | 476 | 12 | 205 | 400 | 906 | 112.1 |
| 1 | 종로구 | 2015 | 24594009 | 2916 | 1150 | 2078 | 267 | 5 | 169 | 417 | 950 | 72.7 |
| 2 | 용인시 | 2015 | 12991160 | 3639 | 2110 | 17997 | 424 | 6 | 162 | 128 | 1967 | 79.2 |
| 3 | 성동구 | 2015 | 10729559 | 7125 | 1995 | 21972 | 563 | 12 | 344 | 1011 | 2753 | 107.8 |
| 4 | 광진구 | 2015 | 6343138 | 7232 | 2585 | 27104 | 651 | 17 | 495 | 1511 | 2951 | 123.6 |

master_2015_2022.csv 는 2015 년부터 2022 년까지 서울시 25 개 자치구의 연도별 통계를 한 번에 다룰 수 있도록 정리한 데이터셋이다. 각 행은 연도 × 자치구 조합을 의미하며, 지역아동센터 이용자 수, 한부모 가정 수, 기초생활수급 가구 수, 다문화 가구 수, 학원 수, GRDP, 인구, 자치구명이 포함되어 있다.



박스플롯(Log 스케일)은 변수별 값의 대략적인 규모 차이와 분포를 보여준다. GRDP와 인구가 절대적인 규모가 가장 크고, 학원 수·지역아동센터 이용자 수는 상대적으로 작은 범위에서 분포한다. 동시에 몇몇 자치구에서 상·하위 극단값이 존재해, 자치구 간 격차가 크다는 것을 확인할 수 있다.

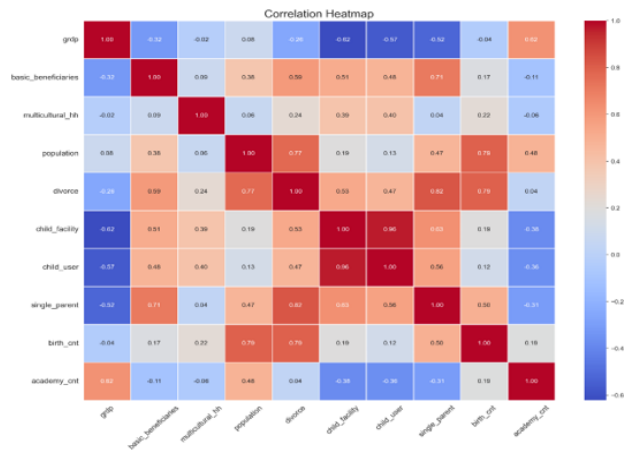
1.2 변수 분포 및 상관관계 분석



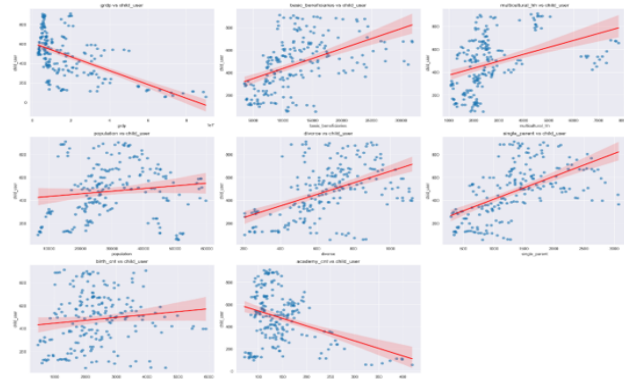
타겟 분포 비교 그래프는 지역아동센터 이용자 수 분포를 나타낸다. 왼쪽의 원본 분포를 보면 오른쪽 꼬리가 긴 형태로, 일부 자치구에서 이용자 수가 많이 몰려 있는 것을 확인할 수 있다. 오른쪽의 log1p 변환 분포는 보다 대칭적인 형태를 보여주며, 이후 회귀·부스팅 모델에서 타겟을 안정적으로 학습시키기 위해 로그 변환을 사용하는 근거가 된다.

박스플롯을 통해 다른 변수들의 스케일과 분포도 함께 확인하였다. 기초생활수급 가구 수, 다문화 가구 수, 한부모 가정 수는 자치구에 따라 편차가 크고, 출생아 수와 인구는 전반적으로 감소 추세에 있으나 여전히 자치구 간 수준 차이가 크다.

08 머신러닝 결과보고서 (3/6)



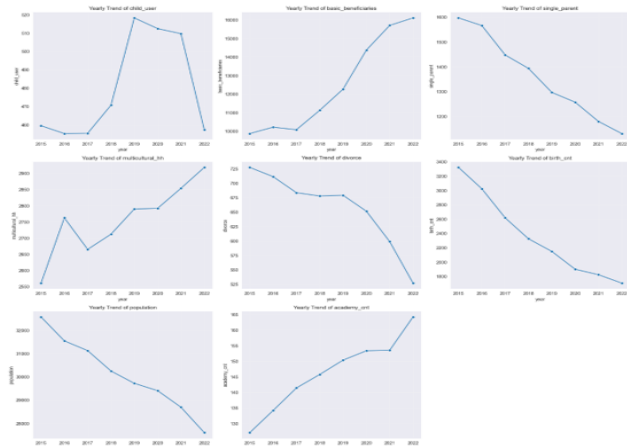
상관관계 히트맵을 보면, 지역아동센터 이용자 수(child_user)는 지역아동센터 수(child_facility, 약0.96), 이혼 건수(divorce), 한부모 가정수(single_parent), 기초생활수급 가구수(basic_beneficiaries), 다문화 가구 수(multicultural_hh)와 양의 상관을 보인다. 즉, 취약 가구 관련 지표와 이혼, 센터 수가 많을수록 실제 이용자 수도 함께 늘어나는 경향이 있다. 반대로 GRDP와 학원 수(academy_cnt)는 지역아동센터 이용자 수와 음의 상관을 보여, 소득 수준이 높고 사교육 인프라가 발달한 자치구일수록 지역아동센터 이용 의존도는 상대적으로 낮을 가능성을 시사한다.



Scatterplot 분석 그래프는 주요 변수와 지역아동센터 이용자 수 간의 개별 관계를 직관적으로 보여준다. 기초생활수급 가구 수, 다문화 가구 수, 한부모 가정 수, 이혼 건수와 child_user는 우상향 패턴을 보여, 해당 값이 많을수록 이용자 수가 증가하는 경향이 뚜렷하다. 인구와 출생아 수는 완만한 우상향 관계를 보여, 인구 규모가 클수록 기본적인 수요 수준이 함께 높아지는 모습을 확인할 수 있다. GRDP와 학원 수는 child_user와 대체로 우하향 관계를 보여, 경제력이 높고 학원이 많은 지역일수록 지역아동센터 이용자 수는 상대적으로 적은 편이다. 이러한 분포·상관 분석 결과를 종합하면, 지역아동센터 수요는 단순히 인구 규모뿐 아니라 취약 가구 비율, 이혼, 지역의 경제력·사교육 인프라 등 여러 요인의 영향을 동시에 받는다는 점을 확인할 수 있다.

08 머신러닝 결과보고서 (4/6)

1.3 연도별 데이터 분석



연도별 추세(trend) 분석 그래프는 2015~2022 년 주요 변수의 변화를 보여준다. 지역아동센터 이용자 수는 2018 년까지 큰 변화가 없었다가 2019 년 일시 증가 후 2020 년 이후 소폭 감소하는 흐름을 보인다. 기초생활수급 가구 수는 꾸준히 증가하고, 다문화 가구 수도 완만한 증가 추세다. 반면 한부모 가정 수, 이혼 건수, 출생아 수, 전체 인구는 전반적으로 감소하며, 학원 수는 연도별로 조금씩 증가한다.

즉, 아동 수와 인구는 줄어드는 반면 기초생활수급·다문화 가구와 학원 수는 증가하고 있다. 이는 “전체 아동은 감소하지만 돌봄 지원이 필요한 가구 비율은 높아지는 상황”을 시사하며, 수요 예측 시 단순 인구 감소만으로 판단하기 어렵다는 점을 보여준다.

2. 데이터 전처리 과정

2.1 결측치 및 이상치 확인

데이터 전처리 단계에서는 먼저 연도별·자치구별로 값이 비어 있는 경우가 있는지 확인하고, 극단적으로 크거나 작은 값이 실제 통계와 맞지 않는지 점검하였다. 공식 통계에서 가져온 데이터인 만큼 전반적으로 결측치는 많지 않았지만, 일부 연·구 조합에서 값이 누락되거나 갑자기 튀는 구간이 있어 원자료를 다시 확인하였다. 명백한 입력 오류라고 보기 어려운 경우에는 그대로 유지하고 모델이 학습 과정에서 반영하도록 하였고, 한두 개 연도만 비어 있는 경우에는 인접 연도의 값과 추세를 참고해 보간하는 방식으로 처리하였다.

2.2 변수 선택 기준 및 최종 피쳐

```
# 사용할 최종 변수들
features = [
    "year",
    "single_parent",
    "basic_beneficiaries",
    "multicultural_hh",
    "academy_cnt",
    "grdp",
    "population"
]

target = "child_user"

df = df_master.copy()
X = df[features]
y = df[target]

train = df[df["year"] <= 2020]
test = df[df["year"] >= 2021]

X_train = train[features]
y_train = train[target]

X_test = test[features]
y_test = test[target]

X_train.shape

(150, 7)
```

전처리 이후 실제 모델에 입력할 변수를 정리하였다. 지역아동센터 이용자 수를 타겟으로 두고, 한부모 가정 수, 기초생활수급 가구 수, 다문화 가구 수, 학원 수, GRDP, 인구, 연도를 설명 변수로 사용하였다. 자치구명은 범주형 변수이므로 원-핫 인코딩을 사용해 각 자치구를 나타내는 더미 변수로 변환하였다. 값이 거의 변하지 않거나 다른 변수와 완전히 겹치는 변수는 제외하고, 해석과 활용이 쉬운 변수 위주로 피쳐를 구성하였다. 최종적으로는 “연도 + 자치구 + 주요 지표”라는 단순한 구조를 유지하면서도, 자치구별·시간별 차이를 충분히 반영할 수 있도록 설계하였다.

08 머신러닝 결과보고서 (5/6)

3. 모델 구축 및 선택과정

3.1 다중 선형 회귀(Multiple Linear Regression) 1차 시도

```
lr = LinearRegression()
lr.fit(X_train, y_train)

pred_lr = lr.predict(X_test)

print("MAE :", mean_absolute_error(y_test, pred_lr))
print("RMSE:", np.sqrt(mean_squared_error(y_test, pred_lr)))
print("R² :", r2_score(y_test, pred_lr))
```

MAE : 126.65530475907494
RMSE : 158.52671662810272
R² : 0.5417029992670379

기본 성능 확인을 위해 다중 선형 회귀를 적용했으며, 타깃은 지역아동센터 이용자 수(child_user)로 설정했다. 한부모 가정 수, 기초생활수급 가구 수, 다문화 가구 수, 학원 수, GRDP, 인구, 연도 등을 변수로 사용했고 원-핫 인코딩과 로그 변환은 적용하지 않았다.

2015~2020년 학습, 2021~2022년 테스트 결과 R²는 약 0.56으로, 전체 흐름은 반영했지만 자치구-연도별 차이를 충분히 설명하기에는 한계가 있었다.

3.2 XGBoost 모델 2차 시도

```
xgb = XGBRegressor(
    n_estimators=600,          # 트리 개수
    max_depth=4,              # 트리 깊이
    learning_rate=0.05,        # 학습률
    subsample=0.9,            # 샘플링 비율
    colsample_bytree=0.8,      # feature 샘플링 비율
    random_state=42
)

xgb.fit(X_train, y_train)
pred_xgb = xgb.predict(X_test)

print("MAE :", mean_absolute_error(y_test, pred_xgb))
print("RMSE:", np.sqrt(mean_squared_error(y_test, pred_xgb)))
print("R² :", r2_score(y_test, pred_xgb))
```

MAE : 80.66059875488281
RMSE : 110.0002648392113
R² : 0.7460734844207764

1차 시도의 한계를 보완하기 위해 트리 기반 부스팅 모델인 XGBoost를 동일한 변수 구성에 적용하였다. XGBoost는 비선형 관계와 변수 간 상호작용을 함께 학습할 수 있어, 단순 선형 구조보다 유연한 예측이 가능하다. 동일한 학습-테스트 분할을 사용했을 때 테스트 R²는 0.74으로 상승하여, 다중 선형 회귀보다 의미 있게 높은 성능을 보였다. 이 단계에서 XGBoost가 자치구별 이용자 수 패턴을 더 잘 설명할 수 있다는 점을 확인했고, 이후 모델 고도화의 기본 틀로 XGBoost를 선택하였다.

3.3 하이퍼파라미터 튜닝 및 로그 변환 적용

```
param_dist = {
    "n_estimators": [200, 300, 500, 700],
    "learning_rate": [0.01, 0.03, 0.05, 0.1],
    "max_depth": [3, 4, 5, 6],
    "subsample": [0.6, 0.7, 0.8, 1.0],
    "colsample_bytree": [0.6, 0.8, 1.0],
    "gamma": [0, 0.1, 0.2, 0.3],
    "reg_alpha": [0, 0.01, 0.1, 0.5],
    "reg_lambda": [1, 1.5, 2]
}

# Randomized Search
search = RandomizedSearchCV(
    estimator=xgb,
    param_distributions=param_dist,
    n_iter=10, # 100 AIC
    scoring='r2',
    cv=5,
    random_state=42,
    n_jobs=-1,
    verbose=1
)

search.fit(X_train, y_train)
best_xgb = search.best_estimator_
print("Best Parameters:", search.best_params_)
```

```
best_xgb = search.best_estimator_
print("Best Parameters:", search.best_params_)

# Evaluate tuned model
pred_xgb = best_xgb.predict(X_test)

print("MAE :", mean_absolute_error(y_test, pred_xgb))
print("RMSE:", np.sqrt(mean_squared_error(y_test, pred_xgb)))
print("R² :", r2_score(y_test, pred_xgb))
```

Fitting 3 folds for each of 30 candidates, totalling 90 fits
Best Parameters: {'subsample': 0.7, 'reg_lambda': 2, 'reg_alpha': 0.1, 'n_estimators': 500, 'max_depth': 3,
MAE : 71.44576263427734
RMSE : 98.47595479837083
R² : 0.8231510519981384

08 머신러닝 결과보고서 (6/6)

```
'learning_rate': 0.1, 'gamma': 0, 'colsample_bytree': 0.6)
```

XGBoost의 성능을 안정적으로 끌어올리기 위해 하이퍼파라미터 튜닝과 데이터 변환을 추가로 진행하였다. 우선 자치구명에 대해 원-핫 인코딩을 적용해 `district_○○구` 형태의 더미 변수를 추가하고, 학습률(learning rate), 트리 개수(n_estimators), 최대 깊이(max_depth), 각 단계의 샘플링 비율 등을 여러 조합으로 실험하였다. 또한 타깃 변수 `child_user`의 오른쪽 꼬리가 긴 분포를 완화하기 위해 `log1p` 변환을 적용한 뒤 모델을 학습하고, 예측 결과는 `expm1`으로 되돌렸다. 이러한 변환과 튜닝을 반영한 최종 XGBoost 모델의 테스트 R^2 는 약 0.84 수준까지 향상되었으며, 자치구별 예측값도 전반적으로 안정적인 패턴을 보였다.

```
pred_log_oh = best_xgb_oh.predict(X_test_oh)
pred_oh = np.expm1(pred_log_oh)
```

```
print("==== XGBoost (district OHE + log1p) =====")
print("MAE :", mean_absolute_error(y_test_oh, pred_oh))
print("RMSE :", np.sqrt(mean_squared_error(y_test_oh, pred_oh)))
print("R² :", r2_score(y_test_oh, pred_oh))
```

```
X_train_oh: (150, 32)
X_test_oh : (50, 32)
feature 개수: 32
Fitting 3 folds for each of 30 candidates, totalling 90 fits
Best Params (OHE + log1p): {'subsample': 0.6, 'reg_lambda': 2.0, 'reg_alpha': 0, 'n_estimators': 800, 'max_depth': 6,
===== XGBoost (district OHE + log1p) =====
MAE : 70.68213653564453
RMSE: 92.53254811652762
R² : 0.8438539585004883
```

3.4 동일 조건에서의 선형회귀 재평가

```
# district 원핫 인코딩 적용한 데이터 준비
df_oh = pd.get_dummies(df_master.copy(), columns=["district"], drop_first=False)

# district_로 시작하는 컬럼들만 추출
district_oh_cols = [c for c in df_oh.columns if c.startswith("district_")]
```

```
print("==== Linear Regression (with district One-Hot) =====")
print("MAE :", mean_absolute_error(y_test_oh, pred_lr_oh))
print("RMSE :", np.sqrt(mean_squared_error(y_test_oh, pred_lr_oh)))
print("R² :", r2_score(y_test_oh, pred_lr_oh))
```

```
X_train_oh: (150, 32)
X_test_oh : (50, 32)
==== Linear Regression (with district One-Hot) =====
MAE : 75.83299385825849
RMSE : 91.62394720498811
R² : 0.846905326084218
```

XGBoost에서 좋은 결과를 얻은 후, “같은 조건을 선형 회귀에도 적용하면 어떨지” 확인하기 위해 다중 선형 회귀 모델을 다시 평가했다.

먼저 XGBoost와 동일하게 타깃 로그 변환과 자치구 원-핫 인코딩을 모두 적용해 학습한 결과, 테스트 R^2 는 약 0.84 수준으로 개선되었지만 미세하게 XGBoost보다는 낮게 나타났다.

이후 타깃 로그 변환은 제거하고, 자치구 원-핫 인코딩만 적용한 선형 회귀 모델을 다시 학습했다. 이 경우 테스트 R^2 가 약 0.84 수준으로 올라가 XGBoost와 거의 비슷한 수치를 기록했다. 지표만 보면 “다중 선형 회귀 + 원-핫 인코딩” 모델도 충분히 사용할 수 있는 후보가 되었다.

3.5 두 모델의 예측값 비교 및 최종 모델 채택

선형 최종 모델 선택은 단순 R^2 크기보다 예측값의 안정성과 현실성을 기준으로 결정하였다. XGBoost와 다중 선형 회귀+원-핫 모델로 미래 연도(예: 2030년) 예측값을 비교한 결과, 선형 회귀 모델에서 일부 자치구의 예측값이 과도하게 증가하는 문제가 확인되었다..

반면 XGBoost는 동일한 미래 피처를 사용했을 때도 증가/감소 방향성을 유지하면서, 예측값이 과거 관측 범위를 크게 벗어나지 않는 수준에서 비교적 완만하게 변했다. 정책 보고서 활용 관점에서는 지표상의 미세한 R^2 차이보다 특정 자치구만 비정상적으로 튀지 않는 현실적 예측이 더 중요하다고 판단하여, 최종적으로 XGBoost를 최종 예측 모델로 채택하고 2023~2030년 수요 예측에도 동일 모델 결과를 사용하였다.

09 시연

프로젝트 시연

<https://drive.google.com/file/d/1DzBx8PQqxOeYAgGnsNaFRTk2noertKaA/view>

10 향후 개발 계획

1. 지능형 자율 워크플로우 및 도구 표준화 (LangGraph & MCP)

자율 사고: **LangGraph** 기반의 '판단-실행-검토' 루프를 통해 **AI가 스스로 DB/RAG 도구를 선택**하고
답변을 **자가 수정(Self-Correction)**

표준 연동: **MCP(Model Context Protocol)** 서버 도입으로 **데이터 조회 로직을 표준화**하여
모델(라마3, GPT 등)에 관계없이 즉시 도구 활용 가능

2. 하이브리드 안정성 및 추론 품질 최적화 (Fail-safe & Optimization)

무중단 운영: RunPod(파인튜닝 모델) 장애 시 OpenAI API(GPT-4o)로 즉시 전환되는 **실시간 스위칭(Fail-safe)** 체계 구축

품질 고도화: 기능별(보고서, Q&A 등) **추론 파라미터 최적화** 및 정밀한 도구 호출을 위한 인스트럭션 가이드라인 확정

11 프로젝트 수행 소감

소통의 중요성

초반엔 역할만 나누고 진행해서, 마지막 통합 단계에서 **소통 부족**이 한꺼번에 터지며 **오류/충돌**이 많았습니다.
주제 변경 이후엔 **설계·기준을 먼저 맞추고** 진행하니, 개발 **속도와 효율**이 확실히 좋아졌습니다.

기능 개발 과정과 어려움과 배움

머신러닝·LLM 흐름을 완전히 **이해하지 못한 채** 시작해 **시행착오**와 **시간 낭비**가 컸습니다.
직접 부딪히며 학습하는 과정에서 개념들이 연결되기 시작했고, 제 **약점(설계, 흐름 이해 부족)**이 **명확**해졌습니다.
다음 프로젝트부터는 **설계를 먼저 확정**하고, 같은 실수를 하지 않고 완성도를 높이는게 첫 번째 목표입니다.