

December 10, 2025

■ 강의명: CSCI E-103: 재현 가능한 머신러닝

■ 주차: Lecture 01

■ 교수명: Anindita Mahapatra

Eric Gieseke

■ 목적: Lecture 01의 핵심 개념 학습

▣ 핵심 요약

본 문서는 1주차 '데이터 엔지니어링 입문' 강의의 핵심 내용을 정리합니다. 데이터 엔지니어링의 정의와 필요성부터 시작하여, 빅데이터를 특징짓는 5V, 다양한 데이터 직군(페르소나), 그리고 데이터 처리 시스템(OLTP vs OLAP, Hadoop vs Spark)의 변천사를 다룹니다. 또한, Databricks 플랫폼을 사용한 첫 번째 실습(Lab 0)의 상세한 설정 및 실행 가이드를 포함합니다.

Contents

1 과정 소개 및 물류 (Course Logistics)	2
1.1 과목 개요	2
1.2 학습 플랫폼 및 자료	2
1.3 수업 구성 및 평가	2
2 핵심 용어 정리 (Key Terminology)	3
3 왜 데이터 엔지니어링인가? (Motivation)	5
3.1 데이터, 그 이상의 가치	5
3.2 데이터 엔지니어링의 핵심 역할	5
4 데이터 엔지니어와 페르소나 (DE Data Personas)	6
4.1 데이터 엔지니어의 위치	6
4.2 데이터 페르소나 (Data Personas)	6
5 핵심 개념 1: 빅데이터의 5가지 특징 (The 5 V's)	7
6 핵심 개념 2: 데이터 처리 방식 비교	8
6.1 OLTP vs. OLAP	8
6.2 SQL (ACID) vs. NoSQL (BASE)	8

6.3 CAP 이론 (CAP Theorem)	8
7 데이터 플랫폼의 진화 (Evolution)	10
7.1 1세대: 데이터 웨어하우스 (Data Warehouse, 1980s)	10
7.2 2세대: 하둡과 데이터 레이크 (Hadoop & Data Lake, 2010s)	10
7.3 3세대: 스파크 (Spark, 2012)	10
7.4 4세대: 레이크하우스 (Lakehouse, 2020s)	11
7.5 클라우드 플랫폼 모델 (IaaS, PaaS, SaaS)	11
8 실습 0: Databricks 시작하기 (Lab 0 Guide)	12
8.1 목표	12
8.2 단계 1: 실습 환경 설정 (Lab Setup)	12
8.3 단계 2: 컴퓨터 연결 (Connect Compute)	12
8.4 단계 3: 노트북 실행 및 매직 커맨드	12
8.5 단계 4: Spark DataFrame 기초 (Python)	13
8.6 단계 5: 데이터 탐색 (SQL)	13
9 1주차 학습 체크리스트 (Checklist)	14

1 과정 소개 및 물류 (Course Logistics)

1.1 과목 개요

- **과목명:** CSCI E-103: 비즈니스 과제 해결을 위한 분석 데이터 엔지니어링
- **1주차 주제:** 데이터 엔지니어링 입문
- **핵심 목표:** 원시 데이터(Raw Data)를 비즈니스 가치가 있는 통찰(Valuable Insights)로 바꾸는 전 과정을 이해하고, 최신 도구를 사용하여 데이터 파이프라인을 구축하는 기술을 학습합니다.

1.2 학습 플랫폼 및 자료

- **Canvas:** 강의 슬라이드, 과제, 강의 계획서 등 모든 공식 자료가 게시됩니다.
- **Slack:** 실시간 소통, 질의응답, 토론을 위한 주 채널입니다.
- **교재:**
 - 주교재: *Simplifying Data Engineering and Analytics with Delta* (본 과정 강사가 저술)
 - 부교재: *Spark - The Definitive Guide*
 - 참고: 두 교재 모두 하버드 도서관을 통해 무료로 전자책을 이용할 수 있습니다.
- **실습 플랫폼:** Databricks Platform (무료 버전)을 사용합니다.

1.3 수업 구성 및 평가

- **수업 구성:** 전반부는 이론 강의, 후반부는 실습(Lab)으로 구성됩니다.
- **섹션 (Section):** 목요일 저녁(ET 기준)에 진행되며, 과제 도움말 및 복습 세션으로 운영됩니다. (녹화본 제공)
- **오피스 아워:** Calendly 링크를 통해 교수 및 조교진(TA)과 1:1 면담을 예약할 수 있습니다.
- **평가 비율:**
 - 4개 과제 (Assignments): 40%
 - 2개 케이스 스터디 (Case Studies): 20%
 - 2개 퀴즈 (Quizzes): 8%
 - 기말 발표 (Final Presentations): 30% (그룹 프로젝트)
 - 참여도 (Participation): 2% (Slack 활동, 출석 등)
- **지각 정책:** 과제 제출 지각 시 하루당 2점 감점되며, 최대 10점까지 감점됩니다.

2 핵심 용어 정리 (Key Terminology)

본 강의에서 자주 사용되는 핵심 용어들을 미리 정리합니다.

Table 1: 데이터 엔지니어링 핵심 용어

용어	원어 (English)	쉬운 설명 (비유)
데이터 엔지니어링	Data Engineering	'데이터 정유 공장'을 짓는 일. 원유(원시 데이터)를 받아 정제(처리)하여 휘발유(분석용 데이터)로 만듭니다.
ETL	Extract, Transform, Load	'데이터 이사'. 여러 곳(Extract)에서 데이터를 모아, 쓸모 있게 다듬고(Transform), 새 집(Load)에 저장합니다.
OLTP	Online Transactional Processing	'가게 계산대(POS)'. 실시간으로 발생하는 개별 거래(결제, 등록)를 빠르게 처리하는 시스템입니다.
OLAP	Online Analytical Processing	'본사 분석실'. 가게 계산대에서 쌓인 과거 매출 전체를 모아 분기별/지역별로 분석하는 시스템입니다.
ACID	Atomicity, Consistency, Isolation, Durability	'은행 송금' 원칙. 거래는 100% 성공하거나 100% 실패(롤백)해야 하며(원자성), 데이터는 항상 정확해야 합니다(일관성). (SQL DB)
BASE	Basically Available, Soft State, Eventual Consistency	'소셜 미디어 좋아요' 원칙. 잠시 불일치하더라도(최종 일관성), 서비스는 절대 멈추면 안 됩니다(가용성). (NoSQL DB)
CAP 이론	CAP Theorem	(일관성, 가용성, 분할 감내) 중 2가지만 선택 가능. 분산 시스템의 근본적인 트레이드 오프입니다.
데이터 웨어하우스	Data Warehouse (DW)	'잘 정리된 창고'. 주로 정형 데이터(SQL)를 BI 리포팅 목적으로 저장합니다.
데이터 레이크	Data Lake	'거대한 호수'. 모든 형태(정형, 비정형)의 데이터를 원시 상태 그대로 저장합니다. (Hadoop 기반)
레이크하우스	Lakehouse	DW(안정성, ACID)와 Data Lake(유연성)의 장점을 결합한 현대적 아키텍처입니다. (Databricks, Delta Lake)
하둡 (Hadoop)	Hadoop	2세대 빅데이터 플랫폼. 저렴한 하드웨어 여러 대로 데이터를 분산 저장(HDFS) 및 처리(MapReduce) 합니다.
스파크 (Spark)	Spark	3세대 빅데이터 플랫폼. '인메모리 (In-Memory)' 처리로 Hadoop보다 100배 빠릅니다. (Databricks의 엔진)
데이터 페르소나	Data Persona	데이터 관련 직군. (데이터 엔지니어, 데이터 과학자, BI 분석가, MLOps 엔지니어 등)
메달리온 아키텍처	Medallion Architecture	데이터 정제 단계를 나누는 방식. Bronze (원시) → Silver (정제) → Gold (집계/분석용).

3 왜 데이터 엔지니어링인가? (Motivation)

3.1 데이터, 그 이상의 가치

우리는 빅데이터(Big Data) + 클라우드(Cloud) + 인공지능(AI)이 융합되는 거대한 혁신의 교차점에서 있습니다. 과거 ”석유”가 산업 시대를 이끌었다면, 오늘날 ”데이터는 새로운 석유”로 불리며 디지털 시대를 주도합니다.

- 데이터는 머신러닝(ML)과 AI 모델을 학습시키는 핵심 **연료**입니다.
- 모든 산업(금융, 헬스케어, 제조, 엔터테인먼트 등)이 데이터를 활용해 경쟁 우위를 확보하려 하며, 스스로를 ’디지털 기업’으로 재정의하고 있습니다.
- 단순히 데이터를 수집하는 것을 넘어, 데이터를 ’활용 가능하게’ 만드는 능력이 기업의 생존을 좌우합니다.

3.2 데이터 엔지니어링의 핵심 역할

데이터 엔지니어링의 본질은 ”원시 데이터(Raw Data)를 가치 있는 통찰(Valuable Insights)로 바꾸는 것”입니다.

비즈니스는 데이터를 원하지 않습니다. 비즈니스는 ’통찰’을 원합니다. 하지만 이 통찰을 얻기까지의 과정은 복잡하고 어렵습니다. 데이터 엔지니어는 이 지저분하고 복잡한 과정을 책임지는 전문가입니다.

비즈니스의 가장 큰 요구사항은 ”통찰의 속도 (Speed to Insights)”입니다. 즉, 데이터가 발생하는 순간부터 분석가가 의미 있는 결론을 내리기까지 걸리는 시간을 최소화하는 것이 데이터 엔지니어링의 핵심 목표입니다.

□ 예제:

데이터의 정제 비유 (탄소 → 다이아몬드)

1. 원시 데이터 (Raw): 쓸모없는 탄소 덩어리와 같습니다. (e.g., 앱 로그 파일)
2. 정렬된 데이터 (Sorted): 데이터를 분류합니다. (e.g., 날짜/사용자별로그 정렬)
3. 정리된 데이터 (Arranged): 필요한 정보만 추출하고 정제합니다. (e.g., 오류 로그 제거, 사용자 ID 매핑)
4. 시각화된 데이터 (Visualized): 차트나 그래프로 표현합니다. (e.g., 시간대별 접속자 수 그래프)
5. 스토리 (Story): 비즈니스 의미를 도출합니다. (e.g., ”새벽 3시에 접속 오류가 급증하며, 이는 특정 지역 사용자에게 집중됨”)

4 데이터 엔지니어와 페르소나 (DE Data Personas)

4.1 데이터 엔지니어의 위치

데이터 엔지니어는 종종 소프트웨어 엔지니어(Software Engineer)와 데이터 과학자(Data Scientist)의 교집합에 위치한다고 말합니다.

- **소프트웨어 엔지니어링 역량:** 안정적인 시스템 설계, DevOps, 분산 컴퓨팅, 서비스 개발.
- **데이터 과학 역량:** 데이터 모델링, 머신러닝 알고리즘 이해, 비즈니스 인텔리전스(BI).
- **데이터 엔지니어 고유 역량:** 대규모 데이터 파이프라인(ETL) 구축, Spark/Kafka 등 분산 처리 도구 활용.

4.2 데이터 페르소나 (Data Personas)

데이터를 다루는 조직에는 다양한 역할(페르소나)이 존재하며, 기말 그룹 프로젝트에서 이 역할들을 맡아 수행하게 됩니다.

- **데이터 엔지니어 (Data Engineer):** 파이프라인을 구축하고 유지보수합니다. 데이터 수집, 변환, 적재(ETL)를 책임집니다.
- **BI 분석가 (BI Analyst):** SQL 기반 리포팅에 집중합니다. 정제된 데이터를 사용하여 대시보드를 만듭니다.
- **데이터 과학자 (Data Scientist):** 탐색적 데이터 분석(EDA) 및 ML 모델링을 수행합니다. 통계 지식을 바탕으로 미래를 예측합니다.
- **MLOps 엔지니어 (MLOps Engineer):** 인프라 자동화를 담당합니다. ML 모델이 안정적으로 배포되고 모니터링되도록 관리합니다.
- **데이터 리더 (Data Leader):** 최고 데이터 책임자(CDO) 등. 데이터 전략과 거버넌스를 총괄합니다.

title=현실의 머신러닝: 빙산의 일각 흔히 AI/ML이라고 하면 '모델 코드' 자체에만 집중하지만, 이는 전체 시스템의 극히 일부에 불과합니다.

실제 ML 시스템의 90% 이상은 데이터 엔지니어링 영역입니다.

모델 코드를 둘러싼 데이터 수집, 데이터 검증, 피처 추출, 리소스 관리, 서빙 인프라, 모니터링 등의 복잡한 인프라가 훨씬 더 많은 노력을 필요로 합니다.

5 핵심 개념 1: 빅데이터의 5가지 특징 (The 5 V's)

빅데이터는 단순히 '많은 데이터'가 아닙니다. 다음 5가지 특징(5V)으로 정의됩니다.

1. **Volume (규모):** 데이터의 물리적인 크기 (e.g., Terabytes, Petabytes). 한 대의 컴퓨터에 저장할 수 없는 규모.
2. **Velocity (속도):** 데이터가 생성되고 처리되는 속도.
 - **배치 (Batch):** 데이터를 모았다가 주기적으로 한꺼번에 처리 (e.g., 일일 정산).
 - **스트리밍 (Streaming):** 데이터가 발생하는 즉시 실시간으로 처리 (e.g., 주가 변동, 사기 탐지).
3. **Variety (다양성):** 데이터의 형태.
 - **정형 (Structured):** 스키마(구조)가 명확함 (e.g., SQL 데이터베이스의 행과 열).
 - **반정형 (Semi-Structured):** 스키마가 데이터 내에 포함됨 (e.g., JSON, XML).
 - **비정형 (Unstructured):** 스키마가 없음 (e.g., 이미지, 오디오, 비디오, 텍스트 문서). 현대 기업 데이터의 80-90%를 차지하며, AI/ML의 핵심 재료입니다.
4. **Veracity (정확성):** 데이터의 품질과 신뢰도. "쓰레기가 들어가면 쓰레기가 나온다 (GIGO)" 원칙. 데이터의 노이즈, 누락, 편향 등을 관리해야 합니다.
5. **Value (가치):** 데이터에서 추출할 수 있는 비즈니스 임팩트. 5V 중 가장 중요합니다.

title=데이터 온도 (Data Temperature) 데이터의 활용 빈도와 속도에 따라 '온도'를 구분합니다.

- **Hot Data:** 실시간으로 자주 접근하고 즉시 처리해야 하는 데이터 (e.g., 현재 접속자 수).
- **Cold Data:** 가끔 접근하며 배치 처리되는 데이터 (e.g., 1년 전 로그).

6 핵심 개념 2: 데이터 처리 방식 비교

6.1 OLTP vs. OLAP

데이터베이스 시스템은 목적에 따라 크게 OLTP와 OLAP로 나뉩니다. 이는 데이터 엔지니어링에서 가장 기본이 되는 구분입니다.

□ 예제:

비유: 마트(Mart)

- **OLTP (Online Transactional Processing):** 마트의 '계산대(POS)'.
- **OLAP (Online Analytical Processing):** 마트 '본사의 분석실'.

Table 2: *OLTP vs. OLAP* 비교

특징	OLTP (운영 시스템)	OLAP (분석 시스템)
목적	실시간 거래(트랜잭션) 처리	비즈니스 분석 및 의사결정
비유	(마트) 계산대	(마트) 본사 분석실
데이터	현재, 실시간, 원본 데이터	과거, 집계된 데이터
주요 작업	빠른 읽기/쓰기/수정 (Insert, Update)	복잡하고 무거운 조회 (Select)
사용자	현장 직원, 고객	분석가, 경영진
예시	ATM 출금, 쇼핑몰 장바구니, 좌석 예약	분기별 매출 보고서, 사용자 이탈률 분석

데이터 파이프라인(ETL)은 OLTP 시스템(운영 DB)에서 데이터를 추출(Extract) 하여, 분석하기 좋은 형태로 변환(Transform)한 뒤, OLAP 시스템(데이터 웨어하우스)에 적재(Load)하는 과정을 의미합니다.

6.2 SQL (ACID) vs. NoSQL (BASE)

데이터를 저장하는 방식은 데이터의 일관성 요구 수준에 따라 나뉩니다.

- **SQL (관계형, ACID):** ACID 속성을 보장합니다. (원자성, 일관성, 고립성, 지속성).
 - **비유 (은행 송금):** 100만 원 송금 시, 내 계좌-100만 원과 상대 계좌+100만 원은 절대적으로 일관성을 유지해야 합니다. 둘 중 하나만 성공하면 재앙입니다.
- **NoSQL (비관계형, BASE):** BASE 속성을 따릅니다. (기본적 가용성, 소프트 상태, 최종적 일관성).
 - **비유 (소셜 미디어 '좋아요'):** '좋아요' 수가 1초 동안 친구에게 100개로 보이고 나에게 101개로 보여도 치명적이지 않습니다. 대신, 서비스가 절대 다운되지 않는 것(가용성)이 더 중요합니다.

6.3 CAP 이론 (CAP Theorem)

분산 데이터 시스템은 다음 세 가지 속성 중 최대 두 가지만 동시에 만족시킬 수 있습니다.

- Consistency (일관성): 모든 노드가 동시에 같은 데이터를 보여줌.
- Availability (가용성): 모든 요청에 항상 응답함 (에러 X).
- Partition Tolerance (분할 감내): 노드 간 통신이 끊겨도(네트워크 장애) 시스템이 동작함.

현대 분산 시스템은 네트워크 장애(P)를 기본 전제로 합니다. 따라서 CP 또는 AP 시스템을 선택하게 됩니다.

- **CP 시스템 (일관성 중요):** 전통적인 RDBMS.
- **AP 시스템 (가용성 중요):** NoSQL (e.g., Cassandra, DynamoDB).

7 데이터 플랫폼의 진화 (Evolution)

7.1 1세대: 데이터 웨어하우스 (Data Warehouse, 1980s)

- 목적: 비즈니스 인텔리전스(BI), 리포팅.
- 데이터: 정형 데이터(SQL)만 처리 가능.
- 한계: 고가의 전용 하드웨어 필요, 비정형 데이터(이미지, 텍스트) 처리 불가, 확장성 부족.

7.2 2세대: 하둡과 데이터 레이크 (Hadoop & Data Lake, 2010s)

- **Hadoop (하둡):** 저렴한 범용 하드웨어(Commodity Hardware) 여러 대를 묶어 대용량 데이터를 처리하는 오픈소스 프레임워크.
- **데이터 레이크 (Data Lake):** 모든 형태(정형, 비정형)의 데이터를 원시 상태 그대로 저장하는 거대 저장소.
- 핵심 구성:
 - **HDFS (Hadoop Distributed File System):** 분산 저장 시스템. (데이터를 3중 복제하여 안정성 확보)
 - **YARN (Yet Another Resource Negotiator):** 클러스터 자원 관리.
 - **MapReduce (맵리듀스):** 분산 처리 엔진.

매우 중요:

Hadoop(MapReduce) 이 느린 이유

MapReduce는 'Map(분할 처리)' 단계와 'Reduce(결과 취합)' 단계로 나뉩니다. 이때 각 단계가 끝날 때마다 결과를 HDFS(디스크)에 썼다가 다시 읽어옵니다. 디스크 I/O는 RAM 접근보다 수천 배 느리기 때문에 전체 처리 속도가 매우 느렸습니다.

7.3 3세대: 스파크 (Spark, 2012)

- **Spark (스파크):** Hadoop의 단점을 보완한 통합 분석 엔진. Hadoop MapReduce보다 최대 100배 빠릅니다.
- 핵심 추상화: RDD (탄력적 분산 데이터셋) → **DataFrame/DataSet** (최신 고수준 API. SQL처럼 사용 가능).

▣ 핵심 요약

Spark가 압도적으로 빠른 이유: 인메모리(In-Memory) 처리

Spark는 MapReduce처럼 매번 디스크에 쓰지 않고, 가능한 한 모든 데이터를 **RAM(메모리)**에 올려놓고 처리합니다.

또한 **DAG(방향성 비순환 그래프)**라는 실행 계획을 세우고, 실제 'Action' (e.g., 'count()', 'collect()'처럼 결과를 봐야 하는 명령)이 호출될 때까지 변환(Transformation)을 지연시켰다가 한꺼번에 처리합니다 (Lazy Evaluation).

7.4 4세대: 레이크하우스 (Lakehouse, 2020s)

- **Data Lake** (유연성, 저비용, 비정형 처리)와 **Data Warehouse** (안정성, ACID 트랜잭션, 거버넌스)의 장점을 결합한 현대적 아키텍처입니다.
- Databricks의 **Delta Lake**가 대표적인 레이크하우스 기술입니다. (Parquet 파일 형식 + 트랜잭션 로그)

7.5 클라우드 플랫폼 모델 (IaaS, PaaS, SaaS)

□ 예제:

비유: 피자(Pizza) 만들기

- **IaaS (Infrastructure as a Service)**: 인프라만 제공. (오븐, 밀가루, 소스 재료를 사서 처음부터 끝까지 직접 요리). 예: AWS EC2, GCP Compute Engine.
- **PaaS (Platform as a Service)**: 플랫폼 제공. (배달된 피자 도우와 소스 위에 토핑만 내가 올림). 예: Heroku, AWS Elastic Beanstalk.
- **SaaS (Software as a Service)**: 완성된 소프트웨어 제공. (완성된 피자를 주문해서 먹기). 예: Gmail, Databricks.

8 실습 0: Databricks 시작하기 (Lab 0 Guide)

8.1 목표

Databricks 무료 버전을 사용하여 클러스터에 연결하고, 노트북을 실행하며, Spark DataFrame 및 SQL을 사용한 기초적인 데이터 조작을 실습합니다.

매우 중요:

계정 확인: "Free Edition" vs. "Trial"

- 본 과정은 "Databricks Free Edition" (무료 버전)을 사용합니다.
- "Trial" (체험판) 계정은 14일 후 만료되며, 유료 플랜 기능이 포함되어 있어 실습 환경이 다를 수 있습니다.
- 로그인 시, 화면 좌측 상단에 "Free Edition" 로고가 있는지 반드시 확인하세요.
- Trial 계정으로 진행 시, 실습 노트북(파일)이 정상적으로 로드되지 않는 오류가 발생할 수 있습니다.

8.2 단계 1: 실습 환경 설정 (Lab Setup)

1. Databricks Free Edition에 로그인합니다.
2. 왼쪽 탐색 메뉴에서 **Workspace** (작업 공간)를 클릭합니다.
3. **Home** 또는 자신의 사용자 이름으로 된 폴더를 찾습니다.
4. 해당 폴더 이름 옆의 화살표 (또는 점 3개)를 클릭하고 **Create → Folder**를 선택하여 'labs'와 같은 새 폴더를 생성합니다.
5. 방금 생성한 'labs' 폴더로 이동합니다.
6. 'labs' 폴더의 빈 공간에서 마우스 우클릭 (또는 폴더 이름 옆 메뉴) 후 **Import** (가져오기)를 선택합니다.
7. 강의 자료로 제공된 실습용 .zip 파일 (e.g., 'Lab0.zip')을 대화 상자에 드래그 앤 드롭합니다. Databricks가 자동으로 압축을 해제하고 노트북 파일들을 폴더에 생성합니다.

8.3 단계 2: 컴퓨터 연결 (Connect Compute)

1. Import된 노트북 파일 중 **00_Initialize** (또는 첫 번째) 노트북을 클릭하여 엽니다.
2. 노트북 우측 상단에 있는 **Connect** (연결) 버튼을 클릭합니다.
3. 드롭다운 메뉴에서 **Serverless** (또는 사용 가능한 기본 클러스터)를 선택합니다.
4. 잠시 기다리면 클러스터가 준비되고, 버튼이 녹색(Running)으로 바뀝니다.

8.4 단계 3: 노트북 실행 및 매직 커맨드

매직 커맨드(Magic Commands)는 노트북 셀의 기본 언어를 임시로 변경하는 명령어입니다.

- **%python**: (기본값) 파이썬 코드를 실행합니다.
- **%sql**: SQL 쿼리를 실행합니다.
- **%md**: 마크다운(Markdown) 텍스트를 렌더링합니다.
- **%sh**: 리눅스 쉘(Shell) 명령어를 실행합니다.
- **%fs**: Databricks File System(DBFS) 명령어를 실행합니다 (e.g., **%fs ls /**).

00_Initialize 노트북의 첫 번째 셀(아마도 **%sql** 셀)을 실행하여 실습에 필요한 Catalog(최상위 데이터

컨테이너)와 Schema(데이터베이스)를 생성합니다. (단축키: Shift + Enter)

8.5 단계 4: Spark DataFrame 기초 (Python)

Spark의 핵심 데이터 구조인 DataFrame을 다뤄봅니다.

```

1 # 예 Databricks 내장된 예제 데이터셋 경로
2 filePath = "/databricks-datasets/bikeSharing/data-001/day.csv"
3
4 # 를 Spark 사용해 CSV 파일을으로 DataFrame 읽기
5 # .option("header", "true"): 첫번째 줄을 헤더 컬럼명 ()로 사용
6 # .option("inferSchema", "true"): 데이터 탑입을 자동으로 추론
7 df = spark.read.format("csv") \
8     .option("header", "true") \
9     .option("inferSchema", "true") \
10    .load(filePath)
11
12 # Databricks 전용 함수 'display'로 결과를 표로 시각화
13 display(df)

```

Listing 1: Python으로 CSV 파일 읽고 DataFrame 생성하기

8.6 단계 5: 데이터 탐색 (SQL)

%sql 매직 커맨드를 사용하여 SQL로 데이터를 직접 조회할 수 있습니다.

```

1 -- 실습용으로 생성한 카탈로그 및 스키마 사용 예시 ()  

2 USE CATALOG csci_e103;  

3 USE SCHEMA lab01;  

4  

5 -- 으로 DataFrame 생성한 테이블 조회  

6 -- 앞 ( 단계에서 df.write.saveAsTable("bike_data") 를 실행했다고 가정 )  

7 SELECT * FROM bike_data WHERE season = '1' LIMIT 10;

```

Listing 2: SQL로 테이블 조회하기

title=Delta Lake: Databricks의 기본 형식 Databricks에서 CREATE TABLE이나 saveAsTable을 사용하면 기본적으로 **Delta Lake** 형식으로 저장됩니다.

Delta Lake는 Parquet 파일 포맷에 ACID 트랜잭션 로그를 추가한 것입니다. 이를 통해 데이터 레이크에서도 안정적인 데이터 변경(Update, Delete, Merge)이 가능해져 '레이크하우스' 아키텍처를 구현할 수 있습니다.

9 1주차 학습 체크리스트 (Checklist)

- Databricks **Free Edition** 계정을 생성했는가? (Trial 계정이 아닌지 재확인)
- 실습 0 (Lab 0) .zip 파일을 Workspace에 성공적으로 Import 했는가?
- Serverless 컴퓨터에 연결하고 00_Initialize 노트북의 첫 번째 셀을 실행했는가?
- 데이터 엔지니어링의 정의를 ”원시 데이터를 가치 있는 통찰로 바꾸는 것”이라고 설명할 수 있는가?
- OLTP와 OLAP의 차이점을 비유(e.g., 가계 예산대 vs. 본사 분석실)를 들어 설명할 수 있는가?
- Hadoop(MapReduce)과 Spark의 속도 차이의 핵심 원인이 디스크 I/O vs. 인메모리 처리임을 이해 했는가?
- 빅데이터의 5V (Volume, Velocity, Variety, Veracity, Value)가 무엇인지 아는가?
- 데이터 페르소나(DE, DS, BI, MLOps)의 역할 차이를 구분할 수 있는가?

December 10, 2025

- 강의명: CSCI E-103: 재현 가능한 머신러닝
- 주차: Lecture 02
- 교수명: Anindita Mahapatra
Eric Gieseke
- 목적: Lecture 02의 핵심 개념 학습

▣ 핵심 요약

본 문서는 데이터 엔지니어링의 핵심인 데이터 모델링을 다룹니다. 데이터를 비즈니스 요구에 맞게 구성하는 3단계(개념적, 논리적, 물리적)를 학습합니다. OLTP(운영)과 OLAP(분석) 시스템의 차이를 이해하고, Inmon, Kimball 등 전통적인 데이터 웨어하우스(DWH) 모델링 기법과 현대적인 메달리온 아키텍처(Bronze, Silver, Gold)를 비교합니다. Parquet, Delta Lake 등 주요 데이터 포맷과 압축 방식의 중요성을 배웁니다. 마지막으로, 주택 가격 예측 실습(Lab-01)을 통해 Databricks 환경에서 데이터를 준비하고 선형 회귀 모델을 훈련시키는 전 과정을 살펴봅니다.

Contents

I 개요 및 복습	2
1 지난 강의 핵심 복습	2
II 데이터 모델링 (Data Modeling)	3
2 데이터 모델링이란?	3
3 데이터 모델링의 3단계	3
3.1 1. 개념적 (Semantic) 모델	3
3.2 2. 논리적 (Logical) 모델	3
3.3 3. 물리적 (Physical) 모델	3
III 데이터베이스 유형별 모델링	5
4 OLTP vs OLAP: 은행원과 분석가	5

5	분석형 모델링 (Dimensional Modeling)	6
5.1	스타 스키마 (Star Schema)	6
5.2	눈송이 스키마 (Snowflake Schema)	6
6	NoSQL 데이터 모델링	7
6.1	Denormalization: 비정규화	7
6.2	임베디드(Embedded) vs. 참조(Referenced) 모델	7
 IV 데이터 웨어하우스(DWH) 아키텍처		 9
7	주요 DWH 모델링 기법	9
8	현대적 아키텍처: 레이크하우스와 메달리온	9
8.1	Conformed Data: 레고(Lego) 비유	9
8.2	메달리온 아키텍처 (Bronze, Silver, Gold)	10
 V 데이터 저장 및 관리		 11
9	데이터 포맷 (Data Formats)	11
9.1	텍스트 vs 바이너리	11
9.2	행(Row) 기반 vs 열(Column) 기반	11
9.3	델타 레이크 (Delta Lake)	12
10	데이터 압축 (Data Compression)	12
11	메타데이터 (Metadata)	13
 VI 데이터 품질과 분석		 14
12	데이터 프로파일링 (Data Profiling)	14
12.1	데이터 시각화 (Data Visualization)	14
 VII 실습 (Lab-01): 주택 가격 예측		 15
13	실습 목표 및 환경	15
14	실습 절차 (Scikit-learn 기준)	15
14.1	1. 환경 설정 (Setup)	15
14.2	2. 데이터 로드 및 변환	16
14.3	3. 데이터 탐색 및 시각화	16
14.4	4. 데이터 준비 (Feature Engineering)	16
14.5	5. 모델 훈련	16
14.6	6. 모델 평가 및 예측	17
14.7	7. 결과 분석	17
 VIII 부록: 과제 및 Q&A		 18
15	과제(Assignment-1) 관련 Q&A	18
16	핵심 용어 정리	18

17 빠르게 훑어보기 (1-Page Summary) 18

Part I

개요 및 복습

1 지난 강의 핵심 복습

데이터 엔지니어링의 기본 용어들을 다시 확인합니다. 이 개념들은 2주차 학습의 기초가 됩니다.

Table 1: 1주차 핵심 용어 복습

용어 (영문)	핵심 설명
ACID	원자성(Atomicity), 일관성(Consistency), 고립성(Isolation), 지속성(Durability). (예: 관계형 데이터베이스, 은행 거래)
BASE	Basically Available, Soft State, Eventual Consistency. (예: NoSQL, 소셜 미디어 피드)
CAP 정리	분산 시스템은 일관성(C), 가용성(A), 분할 용인성(P) 중 최대 2가지만 가질 수 있다는 이론입니다.
IaaS, PaaS, SaaS	클라우드 서비스 모델(Infrastructure, Platform, Software as a Service).
레이크하우스 (Lakehouse)	데이터 레이크(유연성)와 데이터 웨어하우스(성능/관리)의 장점을 결합한 현대적 아키텍처입니다.
DAG	Directed Acyclic Graph(방향성 비순환 그래프). 데이터 파이프라인의 작업 흐름을 정의하는 데 사용됩니다.
데이터의 3대 유형	정형(Structured), 반정형(Semi-structured, 예: JSON, XML), 비정형(Unstructured, 예: 텍스트, 이미지)
ETL	Extract(추출), Transform(변환), Load(적재). 데이터를 한 시스템에서 다른 시스템으로 옮기는 과정입니다.
Spark 아키텍처	작업을 총괄하는 드라이버(Driver)와 실제 작업을 수행하는 여러 워커 노드(Worker Nodes)로 구성됩니다.
Spark의 장점	Polyglot(다양한 언어 지원: Python, SQL, Scala 등) / 빠른 속도(Hadoop과 달리 디스크가 아닌 메모리 사용)
빅데이터의 5V	Volume(규모), Velocity(속도), Variety(다양성), Veracity(정확성), Value(가치).

주의사항

가장 큰 도전 과제: 데이터 품질(Data Quality)과 부실성(Staleness)

빅데이터 시대의 가장 큰 기술적 문제는 데이터의 양이나 속도보다, 데이터가 얼마나 정확하고 최신 상태인지를 보장하는 것입니다. ”쓰레기(Garbage In)가 들어가면 쓰레기(Garbage Out)가 나온다”는 말처럼, 모델링과 파이프라인의 첫 번째 목표는 데이터 품질을 확보하는 것입니다.

Part II

데이터 모델링 (Data Modeling)

2 데이터 모델링이란?

데이터 모델링(Data Modeling)이란 비즈니스 프로세스의 요구에 맞게 데이터를 조직화하고 구성하는 설계 과정입니다.

간단히 말해, 현실 세계의 복잡한 정보를 컴퓨터가 이해하고 효율적으로 저장/검색할 수 있도록 '청사진'을 그리는 작업입니다.

데이터 모델링은 왜 중요 할까요?

- **일관성 및 품질 향상:** 데이터의 이름, 규칙, 형식을 통일하여 데이터 품질을 높입니다.
- **효율적인 저장 및 검색:** 데이터를 어떻게 저장할지 최적의 방식을 설계하여 성능을 향상시킵니다.
- **커뮤니케이션 도구:** 비즈니스 담당자와 개발자 간의 공통된 어휘(common vocabulary) 역할을 하여 오해를 줄입니다.
- **오류 조기 발견:** 설계 단계에서 불일치나 오류를 발견하면, 나중에 시스템을 수정하는 것보다 훨씬 적은 비용으로 해결할 수 있습니다.

3 데이터 모델링의 3단계

데이터 모델링은 추상적인 수준에서 구체적인 수준으로 3단계에 걸쳐 진행됩니다.

3.1 1. 개념적 (Semantic) 모델

- **목적:** 비즈니스의 핵심 개념과 규칙을 정의합니다. (예: "고객은 제품을 구매한다.")
- **사용자:** 비즈니스 이해관계자, 현업 담당자.
- **특징:** 기술적인 세부 사항은 완전히 배제하고, 현실 세계의 엔티티(Entity)와 그들 간의 관계(Relationship)에만 집중합니다.

3.2 2. 논리적 (Logical) 모델

- **목적:** 개념적 모델을 바탕으로 데이터의 구조, 속성(Attribute), 관계를 상세하게 정의합니다.
- **사용자:** 개발자, 데이터 아키텍트, 비즈니스 분석가(BA).
- **특징:** 특정 데이터베이스 기술(예: MySQL, MongoDB)에는 종속되지 않습니다. 데이터의 타입(문자열, 숫자 등), 키(Key) 등을 정의합니다.

3.3 3. 물리적 (Physical) 모델

- **목적:** 논리적 모델을 특정 데이터베이스 기술에 맞게 변환합니다.
- **사용자:** 데이터베이스 관리자(DBA), 데이터 엔지니어.
- **특징:** 실제 데이터베이스에 구현할 수 있도록 모든 세부 사항을 정의합니다.

- 포함 내용:

- 실제 테이블 및 컬럼 이름 (예: ‘CUSTOMER_NAME‘ -> ‘CUST_NM‘(VARCHAR(100)))
- 데이터 타입 (예: ‘Integer‘ -> ‘INT(11)‘)
- 제약 조건 (Constraints), 인덱스(Index), 파티셔닝(Partitioning)
- (예시) 고객과 주소가 N:M(다대다) 관계일 경우, 이를 해소하기 위한 ‘CUSTOMER_ADRESS_JOIN‘

□ 예제:

모델링 3단계 예시: 온라인 서점

- 1. 개념적 모델: ”고객(Customer)이 책(Book)을 주문(Order)한다.”
- 2. 논리적 모델:
 - 고객(Customer) 엔티티: 고객ID (PK), 이름, 이메일
 - 책(Book) 엔티티: 책ID (PK), 제목, 저자
 - 주문(Order) 엔티티: 주문ID (PK), 주문날짜, 고객ID (FK), 책ID (FK)
 - (여기서는 한 주문에 여러 책을 담는 N:M 관계를 단순화함)
- 3. 물리적 모델 (PostgreSQL 기준):

```

1 CREATE TABLE T_CUSTOMER (
2   CUST_ID SERIAL PRIMARY KEY,
3   CUST_NAME VARCHAR(100) NOT NULL,
4   EMAIL VARCHAR(255) UNIQUE
5 );
6 CREATE TABLE T_BOOK (
7   BOOK_ID SERIAL PRIMARY KEY,
8   TITLE VARCHAR(500) NOT NULL,
9   AUTHOR VARCHAR(200)
10);
11 -- 고객과 책의 다대다 관계를 위한 조인 테이블
12 CREATE TABLE T_ORDER_ITEMS (
13   ORDER_ID INT NOT NULL, -- (T_ORDER 테이블을 참조)
14   BOOK_ID INT REFERENCES T_BOOK(BOOK_ID),
15   QUANTITY INT DEFAULT 1,
16   PRIMARY KEY (ORDER_ID, BOOK_ID)
17);

```

Listing 1: 물리적 모델 예시 (SQL)

Part III

데이터베이스 유형별 모델링

4 OLTP vs OLAP: 은행원과 분석가

데이터베이스 시스템은 크게 두 가지 목적, 즉 운영과 분석으로 나뉩니다. 이 목적에 따라 모델링 방식이 완전히 달라집니다.

- **OLTP (Online Transaction Processing):**

- **비유:** 은행 창구 직원의 컴퓨터.
- **목적:** 실시간 운영 및 거래 처리 (예: 계좌 이체, 재고 관리, 회원 가입).
- **특징:**
 - * 수많은 사용자가 동시에 짧고 간단한 작업(읽기/쓰기/수정)을 수행합니다.
 - * 데이터 무결성(정확성)이 매우 중요합니다. (ACID 준수)
 - * 정규화(Normalization)를 통해 데이터 중복을 최소화합니다. (ERD 모델 사용)
- **주요 기술:** 관계형 데이터베이스 (예: MySQL, PostgreSQL, Oracle).

- **OLAP (Online Analytical Processing):**

- **비유:** 은행 본사의 분기별 실적 보고서.
- **목적:** 의사 결정을 위한 데이터 분석 (예: ”지난 1년간 지역별/연령대별 대출 실적은?”)
- **특징:**
 - * 소수의 분석자가 거대한 역사적 데이터를 대상으로 복잡하고 긴 쿼리(대부분 읽기)를 수행합니다.
 - * 쿼리 속도가 매우 중요합니다.
 - * **비정규화(Denormalization)**를 통해 데이터를 중복 저장하더라도 조인(Join)을 줄여 분석 속도를 높입니다. (Dimensional 모델 사용)
- **주요 기술:** 데이터 웨어하우스 (예: Snowflake, BigQuery, Redshift).

Table 2: OLTP vs OLAP 시스템 비교

특징	OLTP (운영 시스템)	OLAP (분석 시스템)
주요 기능	일상적인 운영 (Day-to-day operation)	의사 결정 지원 (Decision support)
설계 목표	애플리케이션 지향 (Application oriented)	주제 지향 (Subject oriented, 예: '매출', '고객')
데이터	현재, 최신 데이터 (Current, up-to-date)	역사적, 요약된 데이터 (Historical, summarized)
사용 패턴	짧고 간단한 트랜잭션 (Read/Write)	복잡한 쿼리 (Lots of scans, Read-only)
데이터 크기	기가바이트 (GB)	테라바이트 페타바이트 (TB, PB)
성능 척도	트랜잭션 처리량 (Transaction throughput)	쿼리 응답 속도 (Query response time)
모델링	ER 모델 (정규화)	차원 모델 (비정규화)

5 분석형 모델링 (Dimensional Modeling)

OLAP 시스템(DWH)에서는 분석 속도를 높이기 위해 차원 모델링을 사용합니다. 이는 '사실(Fact)'과 '차원(Dimension)'으로 데이터를 구분하는 방식입니다.

- **사실 테이블 (Fact Table):**
 - 무엇을 측정할 것인가? (예: 매출액, 판매 수량, 클릭 횟수)
 - 숫자(Numeric)와 측정값(Measure)으로 구성됩니다.
 - 차원 테이블의 키(FK)들을 포함합니다. (예: '고객ID', '제품ID', '날짜ID')
 - 매우 거대하고(Deep) 좁습니다(Narrow).
- **차원 테이블 (Dimension Table):**
 - 어떻게 분석할 것인가? (예: 고객 정보, 제품 상세, 날짜)
 - '누가', '언제', '어디서', '무엇을'에 해당하는 맥락(Context) 정보입니다.
 - 텍스트(Descriptive) 속성으로 구성됩니다. (예: '고객명', '제품 카테고리', '도시명', '요일')
 - 비교적 작고(Shallow) 넓습니다(Wide).

5.1 스타 스키마 (Star Schema)

- **정의:** 하나의 사실 테이블(Fact Table)이 중앙에 있고, 여러 차원 테이블(Dimension Table)이 그 주위를 둘러싼 구조.
- **비유:** 별(Star) 모양. 중앙에 몸통(Fact)이 있고, 팔다리(Dimension)가 뻗어 나간 형태.
- **특징:**
 - 차원 테이블이 정규화되어 있지 않습니다. (예: '고객' 차원에 '도시', '국가' 정보가 모두 포함됨)
 - **장점:** 구조가 단순하고, 사실과 차원 간의 조인이 한 번(Single Join)만 필요하여 쿼리 성능이 매우 빠릅니다.
 - **단점:** 데이터 중복성이 높습니다. (예: '서울'이라는 도시 이름이 고객 차원에 수천 번 중복 저장됨)

5.2 눈송이 스키마 (Snowflake Schema)

- **정의:** 스타 스키마에서 차원 테이블을 정규화(Normalize)한 구조.
- **비유:** 눈송이(Snowflake) 모양. 별의 팔다리(Dimension)에서 또 다른 가지(Sub-dimension)가 뻗어 나간 형태.
- **특징:**
 - (예시) '고객' 차원에서 '도시' 정보를 분리하고, '도시' 차원에서 다시 '국가' 정보를 분리합니다.
 - **장점:** 데이터 중복성이 낮아 저장 공간을 효율적으로 사용하고, 데이터 관리가 용이합니다.
 - **단점:** 구조가 복잡해지고, 원하는 데이터를 얻기 위해 여러 번의 조인(Multiple Joins)이 필요하여 쿼리 성능이 저하될 수 있습니다.

Table 3: 스타 스키마 vs 눈송이 스키마

특징	스타 스키마 (Star Schema)	눈송이 스키마 (Snowflake Schema)
차원 테이블	비정규화 (Denormalized)	정규화 (Normalized)
데이터 중복성	높음	낮음
필요한 조인	적음 (주로 1회)	많음 (여러 단계)
쿼리 성능	빠름	상대적으로 느림
데이터 구조	단순함	복잡함
큐브 처리 속도	빠름	느릴 수 있음

6 NoSQL 데이터 모델링

NoSQL (Not Only SQL) 데이터베이스는 관계형 모델의 한계를 극복하기 위해 등장했습니다. (예: Key-Value, Document, Graph DB)

주의사항

질문의 방향이 다르다: ”답” vs ”질문”

- 관계형 모델링 (SQL): ”내가 가진 데이터(구조)로 어떤 답을 할 수 있는가?” (데이터 구조 중심)
- NoSQL 모델링: ”비즈니스가 원하는 어떤 질문에 답해야 하는가?” (애플리케이션의 쿼리 패턴 중심)

NoSQL은 스키마가 유연(Schema-free) 하지만, 모델링이 필요 없다는 뜻이 절대 아닙니다. 오히려 애플리케이션이 어떻게 데이터를 읽을지(Access Pattern)를 예측하여 읽기 성능에 최적화된 모델을 설계하는 것이 훨씬 더 중요합니다.

6.1 Denormalization: 비정규화

NoSQL 모델링의 핵심은 비정규화(Denormalization)입니다.

- 이유: NoSQL 데이터베이스는 분산 환경에서 대용량 데이터를 다루므로, 여러 테이블을 조인(Join)하는 작업이 매우 비싸거나 아예 지원하지 않는 경우가 많습니다.
- 전략: 데이터를 중복 저장하더라도, 쿼리 한 번에 필요한 모든 정보를 가져올 수 있도록 데이터를 구성합니다. (읽기 성능 극대화)

6.2 임베디드(Embedded) vs. 참조(Referenced) 모델

비정규화의 대표적인 예시입니다. (예: Document DB인 MongoDB)

예제:

상황: 블로그 게시글(Post)과 댓글(Comments)

- 임베디드 (Embedded) 모델 (비정규화) 게시글 도큐먼트 안에 댓글 배열을 포함시킵니다.

```

1 {
2   "post_id": "p123",
3   "title": "My First Post",
4   "content": "Hello world!",

```

```

5   "comments": [
6     { "user": "alice", "text": "Great post!" },
7     { "user": "bob", "text": "Welcome." }
8   ]
9 }
```

Listing 2: 임베디드 모델 예시

장점: 게시글과 모든 댓글을 한 번의 쿼리로 가져올 수 있어 매우 빠릅니다. **단점:** 댓글이 수만 개가 되면 도큐먼트 크기가 너무 커질 수 있습니다.

2. 참조 (Referenced) 모델 (정규화와 유사) 게시글과 댓글을 별도의 컬렉션(테이블)으로 분리하고, ID로 참조합니다.

```

1 (*@\textit{ // Posts Collection }@*)
2 {
3   "post_id": "p123",
4   "title": "My First Post",
5   "content": "Hello world!"
6 }
```

Listing 3: 참조 모델 예시 (Posts)

```

1 (*@\textit{ // Comments Collection }@*)
2 { "comment_id": "c1", "post_id_ref": "p123", "user": "alice", "text": "Great post!" },
3 { "comment_id": "c2", "post_id_ref": "p123", "user": "bob", "text": "Welcome." }
```

Listing 4: 참조 모델 예시 (Comments)

장점: 댓글이 아무리 많아져도 확장성이 좋습니다. **단점:** 게시글과 댓글을 함께 보려면 두 번의 쿼리 (또는 ‘lookup’) .

Part IV

데이터 웨어하우스(DWH) 아키텍처

7 주요 DWH 모델링 기법

데이터 웨어하우스를 구축하는 방식에는 크게 세 가지 접근법이 있습니다.

- 1. Inmon (이몬) 모델: 탑다운 (Top-down)
 - **비유:** "큰 그림(전사)부터 그리기"
 - **접근법:** 먼저 전사적인 중앙 DWH를 3정규형(3NF)으로 완벽하게 구축합니다. (Single Source of Truth)
 - 각 부서(재무, 마케팅)가 필요한 데이터는 이 중앙 DWH에서 추출하여 별도의 데이터 마트(Data Mart)를 만듭니다.
 - **장점:** 데이터의 일관성과 무결성이 높습니다.
 - **단점:** 초기 구축 시간이 매우 오래 걸리고, 변화에 대응하기 어렵습니다.
- 2. Kimball (김볼) 모델: 바텀업 (Bottom-up)
 - **비유:** "레고 블록(부서)부터 조립하기"
 - **접근법:** 비즈니스 요구사항에 맞춰 각 부서별 데이터 마트를 스타 스키마(비정규화)로 빠르게 구축합니다.
 - 전사 DWH는 이 데이터 마트들의 집합으로 구성됩니다.
 - **장점:** 특정 비즈니스 문제에 빠르게 대응할 수 있고, 초기 구현 속도가 빠릅니다.
 - **단점:** 데이터 마트 간의 일관성이 깨지기 쉽습니다. (Single Source of Truth가 아님)
- 3. Data Vault (데이터 볼트): 하이브리드
 - **비유:** "변경 이력을 추적하는 금고"
 - **접근법:** Inmon과 Kimball의 장점을 결합하여, 변경에 유연하게 대응하도록 설계되었습니다.
 - **핵심 구성 요소:**
 - * **Hubs:** 비즈니스의 핵심 키 (예: '고객ID')
 - * **Links:** Hubs 간의 관계 (예: '고객'과 '제품'의 관계)
 - * **Satellites:** Hubs나 Links의 상세 속성 및 변경 이력 (예: 고객의 주소, 이름 변경 내역)
 - **장점:** 데이터 소스 추가/변경이 용이하고, 모든 변경 이력을 추적/감사할 수 있습니다.
 - **단점:** 모델이 복잡해질 수 있습니다.

8 현대적 아키텍처: 레이크하우스와 메달리온

8.1 Conformed Data: 레고(Lego) 비유

데이터가 여러 소스에서 들어올 때는 제멋대로인 '듀플로' 블록과 같습니다. 데이터 엔지니어링의 목표는 이 블록들을 정제하여, 누구나 쉽게 조립할 수 있는 표준 '레고' 블록처럼 만드는 것입니다.

- 데이터 정합(Data Conformance):

- 정렬(Sorted): 데이터를 특정 순서로 맞춥니다.
- 정리(Arranged) 및 그룹화(Grouped): 관련 데이터를 묶습니다. (예: 노란색 블록끼리)
- 일관성(Consistent): 표준을 맞춥니다. (예: 'M', 'F' / 'Male', 'Female' / '0', '1' → 모두 'M', 'F'로 통일)
- 이렇게 정제된 데이터를 **Conformed Data**라 부르며, 비즈니스 사용자들이 믿고 사용할 수 있는 데이터가 됩니다.

8.2 메달리온 아키텍처 (Bronze, Silver, Gold)

현대의 데이터 레이크하우스는 데이터를 품질 수준에 따라 3개의 영역으로 나누어 관리합니다. 이는 위례고 비유를 시스템으로 구현한 것입니다.

- **Bronze (Raw Ingestion)**
 - 목적: 원천(Source) 데이터를 변경 없이 그대로 수집 (Raw data).
 - 특징: 최소한의 변환(예: 타임스탬프 추가)만 수행. 원본을 보존하여 나중에 재처리할 수 있습니다.
 - 상태: 레고 비유에서 '정렬(Sorted)' 단계와 유사.
- **Silver (Filtered, Cleaned, Augmented)**
 - 목적: Bronze 데이터를 가져와 정제, 클렌징, 변환, 통합 (Conformed data).
 - 특징: 데이터 일관성을 맞추고(예: 성별 코드 통일), 여러 소스의 데이터를 조인하여 보강(Augmented)합니다.
 - 상태: 레고 비유에서 '정리 및 그룹화(Arranged)' 단계. 비즈니스 분석가들이 사용하기 시작하는 신뢰할 수 있는 데이터.
- **Gold (Business-level Aggregates)**
 - 목적: 비즈니스 요구에 맞춘 집계(Aggregate) 데이터.
 - 특징: 특정 보고서, 대시보드, AI 모델 훈련에 최적화된 최종 데이터. (예: '일별 매출 요약', '고객별 구매 패턴')
 - 상태: 레고 비유에서 '시각화(Presented Visually)' 또는 완성된 '레고 집'.

Part V

데이터 저장 및 관리

9 데이터 포맷 (Data Formats)

데이터를 어떤 형식으로 파일에 저장하느냐는 성능에 막대한 영향을 줍니다.

9.1 텍스트 vs 바이너리

- **텍스트 포맷 (예: CSV, JSON):**
 - 장점: 사람이 읽고 수정하기 쉽습니다 (Human-readable).
 - 단점:
 - * 공간 낭비: ‘12345’ (숫자)를 “”12345”” (문자열)로 저장해 공간을 많이 차지합니다.
 - * 탑입 없음: ‘123’이 숫자인지 문자인지 알 수 없어 매번 파싱(Parsing)해야 합니다.
 - * 압축 비효율: 반복되는 값에 대한 네이티브 압축이 없습니다.
 - * (JSON의 경우) 모든 행마다 스키마(Key)가 반복되어 매우 비효율적입니다.
- **바이너리 포맷 (예: Parquet, Avro, ORC):**
 - 장점: 기계가 읽기 최적화되어 있습니다 (Machine-readable).
 - 특징: 데이터 탑입, 스키마 정보, 압축, 인덱싱을 파일 내에 포함하여 공간 효율성과 처리 속도가 매우 뛰어납니다.
 - 단점: 사람이 메모장으로 열어서 읽을 수 없습니다.

9.2 행(Row) 기반 vs 열(Column) 기반

데이터를 디스크에 저장하는 방식의 차이입니다.

- **행(Row) 기반 저장 (예: CSV, JSON, Avro):**
 - 비유: 책을 1페이지, 2페이지, 3페이지... 순서대로 저장.
 - 방식: 데이터 한 줄(Row)의 모든 컬럼을 디스크에 순서대로 저장합니다. ‘[Row1: (colA, colB, colC)] [Row2: (colA, colB, colC)] ...’
 - 적합한 용도: OLTP. (예: ”고객 ID 123의 모든 정보 수정”) → 특정 행 전체를 빠르게 읽고 쓰는데 유리합니다.
- **열(Column) 기반 저장 (예: Parquet, ORC):**
 - 비유: 책의 ’목차’만 쪽 모으고, ’본문 1장’만 쪽 모으고, ’색인’만 쪽 모아서 저장.
 - 방식: 모든 행의 특정 컬럼(Column) 데이터를 디스크에 순서대로 저장합니다. ‘[ColA: (row1, row2, ...)] [ColB: (row1, row2, ...)] ...’
 - 적합한 용도: OLAP. (예: ”모든 고객의 평균 나이 계산”) → ‘이름’, ‘주소’ 등 불필요한 컬럼은 읽지 않고, ‘나이’ 컬럼 데이터만 읽어와서 성능이 압도적으로 좋습니다.
 - 장점:
 - * 높은 압축률: 같은 탑입의 데이터(예: 숫자만, 문자열만)가 모여있어 압축 효율이 극대화됩니다.
 - * 분석 성능: 분석 쿼리에 필요한 컬럼만 읽을 수 있습니다(I/O 감소).

Table 4: 주요 데이터 포맷 비교

속성	CSV	JSON	Parquet (파케이)	Avro (아브로)
저장 방식	행(Row)	행(Row)	열(Column)	행(Row)
가독성	좋음	좋음	나쁨	나쁨
압축 가능	✓	✓	✓✓✓ (매우 좋음)	✓
분할 가능 (Splittable)	✓*	✓*	✓	✓
복잡한 구조	X	✓	✓	✓
스키마 진화	X	X	✓	✓✓✓ (매우 좋음)

- **Parquet (파케이):** 열 기반 저장 방식으로, 분석(OLAP) 및 쿼리 성능에 최적화되어 있습니다. Spark의 기본 포맷입니다.
- **Avro (아브로):** 행 기반 저장 방식으로, 스키마를 JSON으로 별도 정의하며 스키마 변경(진화)에 매우 유연합니다. (예: Kafka 데이터 직렬화)

9.3 델타 레이크 (Delta Lake)

델타 레이크(Delta Lake)는 데이터 레이크(Data Lake)에 신뢰성을 더한 오픈소스 스토리지 프로토콜입니다.

기본적으로 데이터는 Parquet 포맷으로 저장하지만, 여기에 트랜잭션 로그(Transaction Log) 디렉토리(`deltaлог/`) .

이 트랜잭션 로그 덕분에 Parquet 파일만 있을 때(데이터 레이크)는 불가능했던 다음과 같은 기능들이 가능해집니다.

- **ACID 트랜잭션:** 빅데이터 환경에서도 데이터베이스처럼 원자성, 일관성, 고립성, 지속성을 보장합니다. (예: 작업 실패 시 롤백)
- **스키마 진화 (Schema Evolution):** 테이블 스키마가 변경되어도 기존 데이터를 문제없이 처리합니다.
- **Time Travel:** 데이터의 모든 변경 이력이 로그에 남아있어, 특정 시점의 데이터로 되돌아갈 수 있습니다.
- **UPSERT (Merge):** 업데이트(Update)와 삽입(Insert)을 한 번의 작업으로 처리할 수 있습니다.
- **배치/스트리밍 통합:** 동일한 테이블에 배치 작업과 스트리밍 작업을 동시에 수행할 수 있습니다.

10 데이터 압축 (Data Compression)

- **목적:** 저장 공간(Disk)을 절약하고, 디스크에서 데이터를 읽어오는 시간(I/O)을 줄여 성능을 향상시킵니다.
- **Lossy vs Lossless:**
 - **Lossy (손실 압축):** 이미지, 비디오에 사용. 사람 눈에 잘 띠지 않는 미세한 디테일을 제거하여 파일 크기를 크게 줄입니다. (데이터베이스에는 절대 사용 불가)
 - **Lossless (무손실 압축):** 데이터베이스, 텍스트에 사용. 원본 데이터를 100% 복원할 수 있습니다.

주의사항

가장 중요한 속성: 분할 가능성 (Splittable)

분산 처리 시스템(Spark, Hadoop)에서 파일 압축 시 분할 가능 여부는 성능에 치명적입니다.

- **분할 불가능 (Non-splittable) (예: Gzip, Snappy):** 100GB짜리 ‘data.gz’ 파일이 1개 있다면, Spark는 이 파일을 100개의 조각으로 나눌 수 없습니다. 오직 1개의 워커(코어)만이 파일 전체를 순차적으로 읽어야 합니다. 이는 분산 시스템의 장점을 완전히 무효화하는 최악의 병목 현상입니다.
- **분할 가능 (Splittable) (예: Bzip2):** 100GB짜리 ‘data.bz2’ 파일이 있다면, Spark는 이 파일을 100개의 조각으로 나눠 100개의 워커에게 동시에 전송할 수 있습니다. 모든 워커가 병렬로 압축을 풀고 작업을 처리하므로 매우 빠릅니다.

결론: Spark에서 대용량 파일을 다룰 때는 압축 속도가 조금 느리더라도 분할 가능한 압축 방식(Bzip2 등)을 사용하거나, Parquet처럼 파일 내부적으로 분할(블록) 처리가 되는 포맷을 사용해야 합니다.

Table 5: 주요 압축 코덱(Codec) 비교

코덱	확장자	분할 가능 (Splittable)?	압축률 (정도)	압축/해제 속도
Gzip	.gz	아니요 (No)	중간	중간
Bzip2	.bz2	예 (Yes)	높음 (좋음)	느림
Snappy	.snappy	아니요 (No)	낮음 (나쁨)	매우 빠름
LZO	.lzo	아니요 (인덱싱 시 가능)	낮음 (나쁨)	빠름

11 메타데이터 (Metadata)

메타데이터(Metadata)란 ”데이터에 대한 데이터(Data about Data)”입니다.

비유: 책의 내용물(소설)이 데이터라면, 책의 표지, 목차, 저자, 출판일, ISBN 코드가 메타데이터입니다.

데이터 자체는 거대한 빙산의 일각일 뿐이며, 그 데이터를 쓸모 있게 만드는 것은 수면 아래의 거대한 메타데이터입니다.

메타데이터는 데이터의 ’이력서’이며, 다음과 같은 질문에 답합니다.

- **Who?** 누가 이 데이터를 생성했는가? 소유자는 누구인가?
- **What?** 이 데이터의 비즈니스 정의는 무엇인가? (예: ’Active User’ = 24시간 내 접속자)
- **Where?** 이 데이터는 어디에 저장되어 있으며, 어디에서 왔는가?
- **Why?** 이 데이터를 왜 저장하는가?
- **When?** 언제 생성/업데이트되었는가? 언제 폐기해야 하는가?
- **How?** 데이터 포맷은 무엇인가?

데이터 카탈로그 (Data Catalog): 이러한 메타데이터를 체계적으로 수집, 관리, 검색 할 수 있도록 도와주는 시스템입니다. (예: Databricks Unity Catalog) 사용자는 카탈로그를 통해 원하는 데이터를 쉽게 찾고, 그 의미와 품질, 출처(Lineage)를 신뢰할 수 있게 됩니다.

Part VI

데이터 품질과 분석

12 데이터 프로파일링 (Data Profiling)

데이터 프로파일링(Data Profiling)은 원본 데이터를 검토하고 분석하여, 데이터의 구조, 내용, 품질, 관계를 파악하는 프로세스입니다.

데이터를 사용하기 전에 ”데이터가 쓸만한지” 건강검진을 하는 것과 같습니다. (예: Spark에서 ‘df.describe()’ 또는 ‘df.summary()’ 명령)

주요 기법:

- **구조 발견 (Structure discovery):** 데이터가 일관되고 올바른 형식인지 확인합니다. (예: 날짜 컬럼에 ‘ABC’ 같은 문자가 섞여 있지 않은가?)
- **내용 발견 (Content discovery):** 데이터 품질을 세부적으로 봅니다. (예: Null 값이 몇 %인가? 유효한 범위(Range)를 벗어난 값은 있는가? - ‘나이’ 컬럼에 ‘200’이 있는 경우)
- **관계 발견 (Relationship discovery):** 데이터셋 간의 연결 고리를 찾습니다. (예: ‘USER’ 테이블의 ‘user_id’ ‘PURCHASE’ ‘user_id’ ?)
- **데이터 상관관계 (Data correlation):** 두 변수가 얼마나 강하게 연관되어 있는지 통계적으로 분석합니다. (Univariate/Multivariate 분석)

12.1 데이터 시각화 (Data Visualization)

프로파일링 결과를 눈으로 쉽게 이해하기 위해 시각화를 사용합니다.

- **히스토그램 (Histogram):** 데이터의 분포를 볼 때 사용합니다. (예: 연령대별 고객 분포)
- **파이 차트 (Pie Chart):** 전체 대비 각 부분의 비율을 볼 때 사용합니다. (예: 플랫폼별(Web/iOS/Android) 매출 비중)
- **산점도 (Scatter Plot):** 두 변수 간의 상관관계를 볼 때 사용합니다. (예: 주택 크기(X축)와 주택 가격(Y축)의 관계)

Part VII

실습 (Lab-01): 주택 가격 예측

13 실습 목표 및 환경

- 목표:** 주택 관련 여러 피처(특성) 데이터를 사용하여 주택 가격을 예측하는 선형 회귀(Linear Regression) 모델을 만듭니다.
- 환경:** Databricks 무료 버전 (Free Edition)을 사용합니다.
- 주요 도구:** Spark DataFrame, Pandas DataFrame, **Scikit-learn**.

주의사항

Spark ML vs. Scikit-learn: 왜 Scikit-learn을 사용하는가?

- Spark ML (MLlib):** Spark의 네이티브 분산 머신러닝 라이브러리입니다. 대용량 데이터를 여러 워커 노드에 분산시켜 훈련합니다.
- Scikit-learn (sklearn):** Python의 대표적인 머신러닝 라이브러리. 단일 노드(Single-node)에서만 동작합니다. (데이터가 드라이버 노드의 메모리에 모두 올라가야 함)
- 이유:** 현재 Databricks 무료 버전의 서버리스(Serverless) 환경에서는 Spark ML(분산 훈련) 라이브러리를 지원하지 않습니다. (2025년 9월 기준)
- 결론:** 어쩔 수 없이 단일 노드 라이브러리인 **Scikit-learn**을 사용합니다. 이를 위해 Spark DataFrame을 Pandas DataFrame으로 변환하는 과정이 필요합니다.

주의사항

병목 지점: Spark DataFrame → Pandas DataFrame

- Spark DataFrame:** 데이터가 여러 워커 노드에 분산되어 저장됩니다.
- Pandas DataFrame:** 데이터가 드라이버 노드(하나의 머신)의 메모리에 모두 존재해야 합니다. Spark에서 ‘.toPandas()’ 명령을 실행하면, Spark 드라이버는 모든 워커 노드에 흩어져 있던 데이터 조각들을 수집(Collect)하여 하나의 머신 메모리로 가져옵니다.
데이터가 수백 GB라면, 드라이버 노드의 메모리가 터져버릴(OOM, Out-Of-Memory) 것입니다. 이는 분산 처리에서 가장 피해해야 할 안티 패턴(Anti-pattern)이며 극심한 병목을 유발합니다.
(이번 실습은 데이터가 작기 때문에 교육용으로 사용합니다.)

14 실습 절차 (Scikit-learn 기준)

14.1 1. 환경 설정 (Setup)

- Databricks 카탈로그 탐색기에서 실습용 스키마(Schema) (데이터베이스)를 생성합니다. (예: ‘lab01’)
- 데이터를 저장할 볼륨(Volume) (폴더)을 생성합니다. (예: ‘input’, ‘output’)

14.2 2. 데이터 로드 및 변환

- **데이터 다운로드:** ‘wget’ 셸 명령어로 공개된 GitHub에서 주택 가격 CSV 파일을 다운로드하여 ‘input’ 볼륨에 저장합니다.
- **데이터 읽기:** Spark를 사용해 CSV 파일을 **Spark DataFrame**으로 읽어옵니다.
- **데이터 보강 (Augment):**
 - 기존 ‘date’ 컬럼(Unix timestamp)을 변환하여 읽기 쉬운 ‘date’ 컬럼을 새로 추가합니다.
 - ‘zipcode’ 컬럼을 ‘string’ 타입으로 변환합니다.
- **Delta 테이블로 저장:** 변환된 Spark DataFrame을 **Delta Lake** 포맷으로 저장합니다. (예: ‘table_{housing}’)

14.3 3. 데이터 탐색 및 시각화

- **데이터 탐색:** ‘DESCRIBE TABLE’이나 ‘SELECT * FROM ...’ 같은 SQL 쿼리를 실행하여 데이터를 탐색합니다.
- **상관관계 시각화:**
 - ”주택 가격은 크기와 관련이 있을까?” 가설을 검증하기 위해 산점도(Scatter Plot)를 생성합니다.
 - X축: ‘sqftliving’()
 - Y축: ‘price’ (가격)
 - → 면적이 클수록 가격이 높아지는 우상향 경향(양의 상관관계)을 시각적으로 확인합니다.

14.4 4. 데이터 준비 (Feature Engineering)

- **Spark → Pandas 변환:** ‘table_{housing}’ **SparkDataFrame** , ‘.toPandas()’ **Pandas DataFrame**
- **인코딩 (Encoding):**
 - 머신러닝 모델은 ’숫자’만 이해할 수 있습니다. ‘zipcode’ 같은 문자열(범주형) 데이터를 숫자로 변환해야 합니다.
 - ‘scikit-learn’의 ‘LabelEncoder’를 사용하여 ‘zipcode’ 컬럼을 숫자형으로 인코딩합니다. (예: ’98101’ → 0, ’98102’ → 1)
- **피처 어셈블 (Feature Assemble):** 모델에 입력으로 사용할 모든 피처(컬럼)들을 하나의 벡터(Vector)로 묶습니다.

14.5 5. 모델 훈련

- **데이터 분할:** 준비된 데이터를 훈련용(Training set) (예: 80%)과 테스트용(Test set) (예: 20%)으로 무작위 분할합니다.
- (팁) ‘randomSplit’ 사용 시 ‘seed’ 값을 고정하면, 매번 실행해도 동일하게 분할되어 실험 결과를 재현 (Reproducible) 할 수 있습니다.
- **모델 훈련:** ‘scikit-learn’의 ‘LinearRegression’ 모델을 초기화하고, 훈련용 데이터를 사용하여 ‘.fit()’ 메서드를 호출해 모델을 훈련시킵니다.

14.6 6. 모델 평가 및 예측

- 모델 평가 (Evaluate): 훈련된 모델을 테스트용 데이터에 적용하여 성능을 평가합니다.
- 주요 지표:
 - RMSE (Root Mean Square Error): 예측값과 실제값의 차이(오차). 낮을수록 좋습니다.
 - R2 (R-squared): 모델이 데이터의 분산을 얼마나 잘 설명하는지 (0~1 사이). 1에 가까울수록 좋습니다.
- 예측 (Predict): 훈련된 모델의 ‘.predict()’ 메서드에 테스트용 데이터를 입력하여 예측 가격을 얻습니다.
- 결과 저장: 예측 결과를 포함한 테스트 데이터셋을 다시 Spark DataFrame으로 변환한 뒤, 새로운 Delta 테이블(예: ‘table_predictions’) .

14.7 7. 결과 분석

모델이 얼마나 잘 예측했는지 시각적으로 분석합니다.

- 잔차(Residual Error) 계산: (실제 가격 - 예측 가격)의 절대값을 계산합니다.
- 히스토그램 분석: 잔차(오차) 값의 분포를 히스토그램으로 그립니다. (대부분의 오차가 0 근처에 몰려있으면 좋습니다.)
- 파이 차트 분석:
 - 예측값의 오차가 1xRMSE 범위 내에 있는지, 2xRMSE 범위 내에 있는지 등을 계산합니다.
 - 결과: 약 80%의 예측이 1xRMSE 범위 내에, 96%의 예측이 2xRMSE 범위 내에 포함되는 것을 확인
→ 이 데이터셋에서는 꽤 괜찮은 모델 성능임을 의미합니다.

Part VIII

부록: 과제 및 Q&A

15 과제(Assignment-1) 관련 Q&A

수업 중 나온 과제 관련 질의응답을 정리합니다.

- **Q: Databricks의 AI Assistant를 사용해도 되나요?**
- **A:** 네, 사용해도 됩니다. 하지만 AI가 생성한 코드를 이해하지 못하면 학습에 도움이 되지 않습니다. AI의 도움을 받더라도, 직접 코드를 타이핑하며 '손 근육 기억(muscle memory)'을 익히고 ABC(기초)를 배우는 것이 중요합니다.
- **Q: 노트북에서 'database'를 생성하라고 하는데, Databricks UI에는 'schema'라고 나옵니다. 다른 것인가요?**
- **A:** Databricks SQL 환경에서는 'SCHEMA'와 'DATABASE'가 동일한 의미로 사용됩니다. 'CREATE DATABASE mydb' 'CREATESCHEMA mydb'
- **Q: 과제 2.9에서 CSV 파일로 저장하라고 하는데, 나중에 Delta 테이블로 또 저장합니다. 둘 다 해야 하나요?**
- **A:** 아니요. CSV 저장 파트는 이전 버전의 과제 내용입니다. CSV로 저장하는 부분은 무시하고, Delta 테이블로만 저장하면 됩니다. (흔동을 피하기 위해 과제 노트북 파일을 다시 다운로드하여 확인하는 것을 권장합니다.)
- **Q: 과제용 데이터를 어떻게 업로드하나요?**
- **A:** 1. Databricks의 'Catalog' 탭에서 스키마(데이터베이스)를 선택합니다. 2. 볼륨(Volume)을 생성합니다. (예: 'assignment01vol') 3. , () . (: 'peopledata', 'namesdata') 4. 'Uploadtothisvolume' '/Volumes/main/lab01/assignment01vol/peopledata/' . 6. 'PC

16 핵심 용어 정리

17 빠르게 훑어보기 (1-Page Summary)

모델링 3단계

1. 개념적 (Why?): 비즈니스와 대화 (예: 고객이 제품을 산다)
2. 논리적 (What?): 개발용 설계 (예: Customer(ID, Name), Product(ID, Price))
3. 물리적 (How?): 실제 DB 구현 (예: T_CUSTOMER (CUST_ID INT, C_NAME VARCHAR(100)))

Table 6: 2주차 핵심 용어

용어	영문	쉬운 설명
데이터 모델링	Data Modeling	데이터를 효율적으로 저장/사용하기 위한 청사진(설계도)을 그리는 작업.
개념적 모델	Conceptual Model	비즈니스 관점에서 데이터의 핵심 개념과 관계를 정의. (기술 배제)
논리적 모델	Logical Model	데이터의 속성, 타입, 관계를 상세히 정의. (특정 DB 기술과 무관)
물리적 모델	Physical Model	특정 DB 기술(예: MySQL)에 맞게 실제 구현을 위한 상세 설계.
OLTP	Online Transaction Processing	실시간 거래 처리 시스템. (예: 은행 창구, 쇼핑몰 주문) (정규화)
OLAP	Online Analytical Processing	데이터 분석 및 리포팅 시스템. (예: 분기별 실적 보고서) (비정규화)
차원 모델링	Dimensional Modeling	OLAP을 위한 설계 방식. '사실(Fact)'과 '차원(Dimension)'으로 구분.
스타 스키마	Star Schema	1개의 Fact 테이블과 N개의 비정규화된 Dimension 테이블로 구성. (빠름)
눈송이 스키마	Snowflake Schema	스타 스키마에서 Dimension 테이블을 정규화한 구조. (복잡함)
Inmon (이몬)	Inmon Model	탑다운(Top-down) DWH 구축 방식. (중앙 3NF DWH → Data Marts)
Kimball (김볼)	Kimball Model	바텀업(Bottom-up) DWH 구축 방식. (Data Marts → DWH)
메달리온 아키텍처	Medallion Architecture	데이터를 Bronze(Raw), Silver(Cleaned), Gold(Aggregated) 3단계로 정제.
행 기반 저장	Row-based Store	데이터 한 줄(Row)을 순서대로 저장. (OLTP 유리, 예: Avro, CSV)
열 기반 저장	Column-based Store	데이터 한 열(Column)을 순서대로 저장. (OLAP 유리, 예: Parquet)
Parquet (파케이)	Parquet	Spark의 기본 저장 포맷. 열 기반 저장 방식으로 분석 성능이 뛰어남.
Delta Lake (델타)	Delta Lake	Parquet + 트랜잭션 로그. 데이터 레이크에 ACID, Time Travel 등 신뢰성을 부여.
분할 가능 압축	Splittable Compression	분산 시스템에서 병렬 처리가 가능한 압축 방식. (예: Bzip2) (매우 중요)
데이터 프로파일링	Data Profiling	데이터의 품질과 구조를 파악하기 위한 '건강검진'.

OLTP vs OLAP

OLTP (은행원):

- 목적: 실시간 거래 (빠른 R/W)
- 모델: 정규화 (ERD)
- 데이터: 최신

OLAP (분석가):

- 목적: 분석/보고 (빠른 Read)
- 모델: 비정규화 (Star/Snowflake)
- 데이터: 역사적

Star vs Snowflake

Star (별):

- 특징: 비정규화된 Dimension.
- 장점: 빠름 (조인 1번)
- 단점: 데이터 중복

Snowflake (눈송이):

- 특징: 정규화된 Dimension.
- 장점: 중복 적음
- 단점: 느림 (조인 여러 번)

Row vs Column (저장 방식)

Row (행 기반, 예: Avro):

- 저장: '(R1:C1,C2,C3), (R2:C1,C2,C3)'
- 특징: 한 줄(Row) 전체를 읽고 쓸 때 빠름 (OLTP 적합)

Column (열 기반, 예: Parquet):

- 저장: '(C1:R1,R2), (C2:R1,R2), (C3:R1,R2)'
- 특징: 특정 열(Column)만 읽을 때 빠름 (OLAP 적합)

Bronze → Silver → Gold

- **Bronze:** 원본 데이터 그대로 (Raw)
- **Silver:** 정제/클렌징된 데이터 (Cleaned) → 분석가들이 신뢰하고 사용
- **Gold:** 비즈니스 목적에 맞게 집계된 데이터 (Aggregated) → 리포트/대시보드용

December 10, 2025

■ 강의명: CSCI E-103: 재현 가능한 머신러닝

■ 주차: Lecture 03

■ 교수명: Anindita Mahapatra

Eric Gieseke

■ 목적: Lecture 03의 핵심 개념 학습

Contents

1 개요 (Overview)	2
2 지난 강의 핵심 복습	3
3 데이터 파이프라인 (Data Pipelines)	4
3.1 데이터 파이프라인이란?	4
3.2 ETL 파이프라인: 특별한 목적	4
3.3 파이프라인 구축 프로세스 및 자동화	4
3.4 장점과 도전과제	5
4 파이프라인 유형 비교	6
5 스트리밍 (Streaming)	7
5.1 왜 스트리밍이 필요한가? (Why Streaming?)	7
5.2 스트리밍 핵심 용어 정리	7
5.3 스트리밍 사용 사례 스펙트럼	8
6 람다(Lambda) vs. 카파(Kappa) 아키텍처	9
6.1 람다 아키텍처 (Lambda Architecture) - 구세대	9
6.2 카파 아키텍처 (Kappa Architecture) - 신세대	9
6.3 FAQ: 카파 아키텍처에서 배치는 어떻게 처리하나요?	10
7 스트리밍 처리 프레임워크	11
8 스트리밍 아키텍처의 트레이드오프	12
8.1 균형잡기: 반복적 프로세스	12
8.2 실전 시나리오와 디버깅	12
8.3 스트리밍 모범 사례 (Best Practices)	13

9	실습: 네트워크 로그 분석 (Lab 02: Network Analysis)	14
9.1	실습 목표 및 준비	14
9.2	1단계: 원시 데이터 읽기 (Read Raw Data)	14
9.3	2단계: 데이터 파싱 (Parse Data)	14
9.4	3단계: 타임스탬프 변환 (UDF 사용)	15
9.5	4단계: 데이터 보강 (Join Dimension)	15
9.6	5단계: 필터링 및 저장 (Save to Delta)	16
9.7	6단계: 확인 (Verify)	17
10	빠른 훑어보기 (1-Page Summary)	18

1 개요 (Overview)

이 문서는 데이터 엔지니어링의 핵심 구성 요소인 데이터 파이프라인, 특히 ETL(추출, 변환, 적재) 프로세스에 대해 다룹니다.

데이터 처리의 두 가지 주요 패러다임인 **배치(Batch) 처리**와 **스트리밍(Streaming) 처리**의 개념을 비교하고, 이 두 가지를 통합하려는 시도에서 발전한 **람다(Lambda) 아키텍처**와 **카파(Kappa) 아키텍처**의 차이점을 중점적으로 학습합니다.

이 문서를 통해 데이터를 효율적으로 수집하고 처리하는 방법론을 이해하고, 다양한 비즈니스 요구사항에 맞는 적절한 데이터 아키텍처를 선택하는 기준을 확립하는 것을 목표로 합니다. 또한 Spark Structured Streaming을 사용한 실습 예제를 포함하여 이론적 개념이 실제 코드에서 어떻게 구현되는지 살펴봅니다.

2 지난 강의 핵심 복습

데이터 파이프라인과 스트리밍을 배우기에 앞서, 데이터 모델링 및 저장소에 대한 이전 강의의 핵심 용어들을 복습합니다.

Table 1: 데이터 모델링 및 저장소 핵심 용어 복습

개념 (Concept)	핵심 설명 (Explanation)	예시 (Example)
3NF (정규화)	데이터 중복을 최소화하고 데이터 무결성을 보장하기 위한 관계형 데이터 베이스 모델링 기법입니다.	고객 테이블, 주문 테이블, 상품 테이블을 분리하여 관리합니다.
비정규화 (Denormalization)	데이터 중복을 감수하는 대신, 빠른 조회 속도를 얻기 위해 테이블을 통합하거나 중복 필드를 추가하는 기법입니다. (주로 NoSQL에서 사용)	고객 테이블에 '최근 주문일'을 중복 저장하여 조인(Join)을 피합니다.
ETL / ELT	ETL (Extract, Transform, Load): 추출 → 변환 → 적재. 전통적인 방식. ELT (Extract, Load, Transform): 추출 → 적재 → 변환. 데이터 레이크ハウス에서 주로 사용.	ETL: CRM 데이터를 정제하여 DW에 저장. ELT: 원본 로그를 S3에 먼저 올리고 Spark로 변환.
Star / Snowflake Schema	데이터 웨어하우스(DW)의 대표적 Star (성형): 하나의 팩트(Fact) 테이블이 여러 차원(Dimension) 테이블에 직접 연결됩니다. (비정규화) Snowflake (눈송이): 차원 테이블이 추가로 정규화되어 여러 단계로 나뉩니다.	Star: '판매' 팩트가 '날짜', '고객', '상품' 차원에 바로 연결됨. Snowflake: '상품' 차원이 '카테고리' 차원으로 추가 분리됨. 인스키마 디자인입니다.
키-값 스토어	고유한 키(Key)와 값(Value)의 쌍으로 만 데이터를 저장하는 가장 단순한 NoSQL 모델입니다.	Amazon S3, Redis
도큐먼트 스토어	JSON이나 XML 같은 유연한 문서 (Document) 형식으로 데이터를 저장합니다. 스키마가 유연합니다.	MongoDB, CouchDB
컬럼형 스토어	데이터를 행(Row) 단위가 아닌 열(Column) 단위로 저장하여, 분석 쿼리(OLAP)에 매우 높은 성능을 보입니다.	Cassandra, DynamoDB
바이너리 포맷	텍스트(CSV, JSON)보다 압축률이 높고 성능이 뛰어난 이진 파일 형식입니다. Parquet: 컬럼(열) 기반 저장 방식으로 분석에 최적화되어 있습니다.	Avro, Parquet
델타 레이크 (Delta Lake)	데이터 레이크(S3 등) 위에 ACID 트랜잭션, 스키마 진화, 탑업 트래블(버전 관리) 기능을 제공하는 오픈소스 스토리지 포맷입니다.	Parquet 파일과 _delta_log(트랜잭션 로그)로 구성됩니다.
메타데이터 (Metadata)	데이터에 대한 데이터로, 데이터의 구조, 유형, 이력 등 맥락을 설명하는 정보입니다.	테이블 스키마, 델타 레이크의 트랜잭션 로그.

3 데이터 파이프라인 (Data Pipelines)

3.1 데이터 파이프라인이란?

데이터 파이프라인(Data Pipeline)은 데이터를 한 지점(소스)에서 다른 지점(대상)으로 이동시키는 전체 프로세스를 의미합니다. 이는 데이터 이동을 위한 일반적인 ”배관(Plumbing)” 작업에 비유할 수 있습니다.

데이터 파이프라인은 단순히 데이터를 A에서 B로 복사하는 간단한 작업일 수도 있고, 여러 소스에서 데이터를 수집하고, 복잡하게 변환하며, 여러 대상에 저장하는 정교한 프로세스일 수도 있습니다.

3.2 ETL 파이프라인: 특별한 목적

ETL(Extract, Transform, Load) 파이프라인은 데이터 파이프라인의 특수한 유형입니다. 이는 원시(Raw) 데이터를 수집하여 비즈니스 분석, 애플리케이션, 머신러닝(ML) 시스템에서 즉시 사용할 수 있는 ’준비된 데이터’로 만드는 데 특화되어 있습니다.

▣ 핵심 요약

데이터 파이프라인 흐름 예시 데이터 소스 → 수집 (Ingestion) → Raw/Landing (원본 저장) → 변환 (ELT/ETL) → Curated (정제됨) → 데이터 목적지

데이터 파이프라인 vs. ETL 파이프라인

데이터 파이프라인 (Data Pipeline)

ETL 파이프라인 (ETL Pipeline)

ETL 파이프라인은 데이터 분석을 목적으로 하는, 데이터 파이프라인의 전문화된 하위 집합입니다.

3.3 파이프라인 구축 프로세스 및 자동화

데이터 파이프라인은 일회성 작업이 아니라 지속적으로 관리되어야 하는 소프트웨어 애플리케이션과 같습니다.

- **구축 프로세스:** 설계 → 구현 → 테스트 → 배포 → 모니터링
- **변경 관리:** 비즈니스 요구사항이나 데이터 소스가 변경되면 파이프라인을 수정하고 다시 배포해야 합니다.
- **자동화 (Automation):** 파이프라인의 핵심 가치입니다. 수동 개입 없이 자동으로 실행되어야 합니다.

파이프라인이 실행되는 시점을 결정하는 것을 **트리거(Trigger)**라고 하며, 주요 유형은 다음과 같습니다.

- **파일 도착 (File Arrival):** 특정 위치에 새 파일이 도착하면 실행됩니다.
- **스케줄 (Schedule):** 매일 오전 6시, 매주 월요일 등 정해진 일정(Cron)에 따라 실행됩니다.
- **수동 (Manual):** 사용자가 직접 실행을 명령합니다.

3.4 장점과 도전과제

▣ 핵심 요약

파이프라인의 장점 (Pros)

- **셀프 서비스 및 IT 의존도 감소:** 현업 사용자가 IT 팀의 개입 없이도 자동화된 채널을 통해 필요한 데이터에 접근할 수 있습니다.
- **클라우드 자원 활용:** 데이터 양에 따라 동적으로 컴퓨팅 자원을 확장(Elasticity)하여 비용 효율적으로 처리할 수 있습니다.
- **실시간 의사결정 지원:** 실시간 분석 및 애플리케이션을 지원하여 비즈니스의 빠른 의사결정을 돋憬니다.

주의사항

파이프라인의 도전과제 (Cons)

- **데이터 드리프트 (Data Drift):** 가장 큰 문제입니다. 원본 데이터의 스키마 (열 추가/삭제/변경)나 품질 (값의 형식 변경)이 예고 없이 변경되면 파이프라인이 실패할 수 있습니다.
- **인프라 종속성:** 파이프라인은 특정 기술, 인프라, 프로세스에 강하게 결합되어 있어, 해당 환경이 변경되면 파이프라인도 영향을 받습니다.

4 파이프라인 유형 비교

데이터 파이프라인은 처리 방식, 목적, 사용 도구에 따라 다양하게 분류할 수 있습니다.

Table 2: 주요 데이터 파이프라인 유형 비교

파이프라인 유형	처리 스타일	주요 도구	대표 사용 사례
배치 (Batch)	대량의 데이터를 주기적으로 처리 (시간별, 일별, 주별)	Apache Spark, AWS Glue, Talend	일일 리포트 생성, 금여 정산, 월간 빌링
스트리밍 (Real-Time)	이벤트 중심의 연속적인 데이터 처리	Apache Kafka, Flink, AWS Kinesis, Databricks Streaming	사기 탐지, 실시간 대시보드, IoT 센서 데이터
ETL	추출 → 변환 → 적재. (주로 배치 방식)	Informatica, Talend, Matillion, dbt	CRM/ERP 데이터를 정제하여 데이터 웨어하우스로 로드
ELT	추출 → 적재 → 변환. (배치 또는 실시간)	dbt, Snowflake, Databricks	원본 로그를 DW에 먼저 적재 후 나중에 변환
데이터 복제	데이터를 거의 실시간 또는 스케줄에 따라 동기화	Fivetran, Airbyte, AWS DMS	운영 DB(OLTP)를 분석용 DB(OLAP)로 복제
ML 파이프라인	모델 훈련 또는 배포를 위한 데이터 처리 (배치/실시간)	MLflow, Kubeflow, Vertex AI	고객 이탈 예측 모델 훈련 및 서빙
워크플로우 / 오케스트레이션	파이프라인 간의 의존성을 관리하는 '메타 파이프라인'	Apache Airflow, Prefect, Dagster	ETL 작업 실행 → 완료 시 모델 훈련 → 완료 시 대시보드 업데이트

5 스트리밍 (Streaming)

5.1 왜 스트리밍이 필요한가? (Why Streaming?)

”It's all about SPEED.” (핵심은 속도입니다.)

스트리밍의 목적은 이벤트 스트림(연속적인 데이터 흐름)을 더 빠르게 분석 가능한 데이터로 전환하여 인사이트를 얻는 것입니다. 이는 비즈니스 중심의 가치(예: 사기 탐지)를 IT 중심의 코딩 프로세스보다 우선시하는 현대 데이터 플랫폼의 핵심입니다.

모든 작업에 실시간(초 단위) 처리가 필요한 것은 아닙니다. 하지만 비즈니스 요구사항에 따라 ”배치(일 단위)”에서 ”실시간(초 단위)”까지 다양한 속도 스펙트럼이 존재하며, 스트리밍 기술은 이 스펙트럼 전반을 다룰 수 있습니다.

5.2 스트리밍 핵심 용어 정리

스트리밍 파이프라인을 구성하는 핵심 개념들은 다음과 같습니다.

Table 3: 스트리밍 핵심 용어

용어 (Term)	설명 (Description)
소스 (Source) & 싱크 (Sink)	모든 파이프라인은 데이터를 읽어오는 소스(예: Kafka)와 데이터를 저장하는 싱크(예: Delta Table)로 구성됩니다.
파일 기반 vs. 이벤트 기반	파일 기반: 데이터가 디스크(파일)에 저장된 후 처리됩니다. (상대적 느림) 이벤트 기반: 데이터가 메모리(이벤트)에서 디스크를 거치지 않고 바로 처리됩니다. (매우 빠름, 예: Kafka)
마이크로배치 vs. 연속	마이크로배치 (Micro-batch): 데이터를 아주 작은 묶음(예: 1초)으로 나누어 배치처럼 처리합니다. (Spark Structured Streaming의 기본값) 연속 (Continuous): 이벤트가 도착하는 즉시 처리합니다. (Flink의 방식)
처리 트리거 (Trigger)	마이크로배치가 실행되는 간격을 의미합니다. (예: 30초마다)
출력 모드 (Output Modes)	싱크에 데이터를 쓰는 방식입니다. (Append: 추가, Overwrite: 덮어쓰기, Update: 갱신)
체크포인트 (Checkpoint)	[매우 중요] 파이프라인이 어디까지 처리했는지 기록하는 ”게임 저장 지점”입니다. 장애 발생 시, 체크포인트부터 복구하여 데이터 유실이나 중복 없이 처리를 재개할 수 있습니다.
윈도우 (Window)	집계(Aggregation)를 위한 시간 간격입니다. (예: ”지난 5분간”的 평균 클릭 수)
워터마크 (Watermark)	[매우 중요] ”지연 도착 데이터”를 얼마나 기다려줄지 정의하는 허용 시간입니다. (예: ”버스가 10분 더 기다려줌”) 이는 집계 연산 시 무한정 상태(State)를 저장하는 것을 방지하여 메모리 초과(OOM)를 막는 핵심 기능입니다.

5.3 스트리밍 사용 사례 스펙트럼

모든 비즈니스가 초(second) 단위의 응답을 요구하지는 않습니다. 요구사항에 맞춰 적절한 속도를 선택해야 합니다.

Table 4: 데이터 처리 속도별 사용 사례

시간 단위 (Latency)	유형 (Type)	사용 사례 (Use Cases)
일 (Day) - 시간 (Hours)	배치 (Batch)	ETL, 미터링 및 빌링(정산), 임시 분석, BI 리포트,
분 (Minutes)	준 실시간 (Near Real-Time)	모바일 및 IoT 데이터 수집, 서비스 모니터링 및 알림, 로그 수집, 클릭스트림 분석
초 (Sec) - 밀리초 (Sub-Sec)	실시간 (Real-Time)	사기 탐지 (Fraud Detection), 금융 자산 거래, 멀티플레이어 게임, ML 추론 (Inference)

6 람다(Lambda) vs. 카파(Kappa) 아키텍처

주의사항

[용어 혼동 주의] AWS Lambda ≠ 람다 아키텍처 여기서 다루는 **람다(Lambda) 아키텍처**는 데이터 처리 파이프라인을 구성하는 업계 표준 아키텍처 패턴(Pattern)입니다.

이는 AWS의 서비스(Serverless) 컴퓨팅 서비스인 **"AWS Lambda"**와는 전혀 다른 개념**입니다. 두 용어가 우연히 이름이 같을 뿐이므로 혼동하지 않아야 합니다.

6.1 람다 아키텍처 (Lambda Architecture) - 구세대

람다 아키텍처는 스트리밍 기술이 성숙하지 않았던 시기에 배치의 신뢰성과 스트리밍의 속도라는 두 마리 토끼를 잡기 위해 고안된 방식입니다.

▣ 핵심 요약

람다 아키텍처 흐름

- 데이터가 들어오면 두 개의 파이프라인으로 동시에 복제되어 흐릅니다.
- 1. 배치 레이어 (Batch Layer): (느리지만 정확함)
 - 모든 원본 데이터를 저장하고 주기적으로 배치 처리하여 '정확한 과거 데이터 뷰(Batch View)'를 생성합니다.
- 2. 스트리밍 레이어 (Streaming Layer): (빠르지만 근사치일 수 있음)
 - 실시간 데이터를 빠르게 처리하여 '최신 데이터 뷰(Real-time View)'를 생성합니다.
- 3. 서빙 레이어 (Serving Layer):
 - 사용자가 쿼리를 보내면, 'Batch View'와 'Real-time View'의 결과를 조합(Reconciliation)하여 최종 결과를 반환합니다.
- 장점: 속도와 신뢰성의 균형을 맞출 수 있습니다.
- 단점 (치명적):
 - 코드 중복: 동일한 비즈니스 로직을 배치용과 스트리밍용으로 두 번 개발해야 합니다.
 - 복잡성 및 비용: 두 개의 파이프라인을 유지보수해야 하므로 매우 복잡하고 비용이 많이 듭니다.
 - 결과 일치 문제 (Reconciliation Headache): 배치 결과와 스트리밍 결과가 정확히 일치하도록 보장하는 것이 매우 어렵습니다.

6.2 카파 아키텍처 (Kappa Architecture) - 신세대

카파 아키텍처는 람다 아키텍처의 복잡성을 해결하기 위해 등장했으며, 현대 스트리밍 프레임워크가 성숙하면서 표준으로 자리 잡았습니다.

카파 아키텍처의 핵심 아이디어: "모든 것을 스트림으로 취급한다."

▣ 핵심 요약

카파 아키텍처 흐름

- 데이터 파이프라인이 오직 하나(스트리밍 레이어)만 존재합니다.
- 1. 스트리밍 레이어 (Streaming Layer):
 - 모든 데이터(실시간이든 배치든)를 스트림으로 간주하고 처리합니다.
- 2. 서빙 레이어 (Serving Layer):
 - 스트리밍 레이어에서 처리된 결과를 받아 사용자에게 제공합니다.

- 장점:

- 단순함: 단 하나의 코드베이스와 파이프라인만 관리하면 됩니다.
 - 확장성: 수평 확장이 용이합니다.
 - 결과 일치성: 결과 일치 문제(Reconciliation)가 원천적으로 발생하지 않습니다.

6.3 FAQ: 카파 아키텍처에서 배치는 어떻게 처리하나요?

▣ 핵심 요약

Q: 카파 아키텍처는 스트리밍 전용인가요? 일(Day) 단위 배치 작업은 어떻게 하나요? A: 카파 아키텍처에서도 배치 작업을 완벽하게 수행할 수 있습니다.

배치(Batch)는 그저 ”처리 간격(Trigger Interval)이 매우 긴 스트림”일 뿐입니다.

예를 들어, Spark에서 `spark.read` (배치 API) 대신 `spark.readStream` (스트리밍 API)을 사용하되, 트리거 옵션을 `.trigger(once=True)` 또는 `.trigger(processingTime='24 hours')`로 설정하면 됩니다.

이렇게 하면 디자인은 스트리밍이지만(`readStream`), 동작은 배치처럼(하루 한 번 실행) 됩니다.

이점: 스트리밍 API를 사용하면 체크포인트(Checkpoint) 기능이 활성화됩니다. 플랫폼이 알아서 ”어떤 파일을 처리했고, 어떤 파일을 처리하지 않았는지” 상태를 관리해 줍니다. 따라서 개발자가 수동으로 파일 처리 여부를 추적하는 로직을 만들 필요가 없어집니다.

7 스트리밍 처리 프레임워크

다양한 스트리밍 처리 프레임워크가 있으며, 각기 다른 장단점을 가집니다.

Table 5: 주요 스트리밍 처리 프레임워크 비교

프레임워크	처리 모델	지연 시간(Latency)	상태 관리	내결합성	비고 / 최고 장점
Kafka Streams	이벤트 단위(Event-at-a-time)	매우 낮음(<10ms)	RocksDB(로컬)	Kafka 로그	Kafka와 완벽 통합, 경량 라이브러리
Apache Flink	진짜 스트리밍(True streaming)	매우 낮음(ms)	고급(대용량 상태)	초저지연, 복잡한 상태 처리(CEP)에 최강	
Spark Structured Streaming	マイクロ배치(기본값)	높음(100ms 초)	체크포인트	강력함(Exactly-once)	배치+스트리밍 통합 API, Spark 생태계
Apache Storm	이벤트 단위(Event-at-a-time)	매우 낮음(ms)	제한적	기본(Acks)	Legacy, 현재는 거의 사용 안 함
Apache Samza	이벤트 단위(Event-at-a-time)	낮음	RocksDB	Kafka/YARN	LinkedIn에서 개발, 현재는 Niche
KSQLDB	Kafka 기반 연속 SQL	낮음	내부 관리	Kafka 상속	Kafka 데이터를 SQL로 쉽게 스트리밍
Amazon Kinesis	マイクロ배치	중간	관리형	관리형	AWS 네이티브 앱, IoT
Google Dataflow (Beam)	통합 배치 + 스트리밍	낮음 중간	고급(Stateful)	강력함	GCP 네이티브, Apache Beam 기반(이식성)
Azure Stream Analytics	SQL 기반 스트리밍	중간	제한적	관리형	Azure 네이티브, 로우코드(Low-code)

▣ 핵심 요약

프레임워크 선택 가이드 (Rule of Thumb)

- 초저지연 및 복잡한 이벤트 처리가 필요하다면? → **Apache Flink**
- 배치와 스트리밍을 통합하고 Spark/Lakehouse 생태계를 활용한다면? → **Spark Structured Streaming**
- 이미 **Kafka**를 사용 중이고 간단한 처리가 필요하다면? → **Kafka Streams** 또는 **KSQLDB**
- 특정 클라우드(AWS, GCP, Azure)에 종속적이어야 된다면? → Kinesis, Dataflow, Azure Stream Analytics

8 스트리밍 아키텍처의 트레이드오프

8.1 균형잡기: 반복적 프로세스

스트리밍 파이프라인 설계는 ”한 번에 완벽하게”가 아니라, 요구사항과 비용 사이의 균형을 맞추는 반복적인 프로세스입니다.

1. 목표(Goals) 이해: 비즈니스 요구사항(SLA)을 명확히 합니다. (얼마나 빨라야 하는가? 데이터 정확도는?)
2. 전략(Strategy) 정의: 목표 달성을 위한 기술적 접근법을 정의합니다. (Flink? Spark? Watermark 10초? 10분?)
3. 자원(Resources) 정의: 전략에 필요한 자원을 산출합니다. (VM 스펙, 클러스터 규모)
4. 비용(Cost) 계산: 해당 자원의 비용을 계산합니다.
5. 트레이드오프(Tradeoffs) 재검토: 비용이 너무 높다면 목표를 조정하거나(예: 실시간 → 준실시간), 전략을 변경합니다. (1번으로 복귀)

결국 스트리밍 설계는 성능(Performance), 품질(Quality), 확장성(Scalability)이라는 목표와 컴퓨팅(Compute), 스토리지(Storage), 개발/운영 노력(Effort)이라는 자원 간의 줄다리기입니다.

8.2 실전 시나리오와 디버깅

스트리밍 파이프라인 운영 시 흔히 겪는 문제와 해결책입니다.

주의사항

시나리오 1: 비용 문제 (네트워크 위협 탐지)

- 증상: VM(컴퓨팅) 비용(30%)보다 스토리지 비용(70%)이 비정상적으로 높게 나옴.
- 원인 1 (작은 파일 문제): 스트리밍 볼륨이 너무 낮아(Low frequency), 아주 작은 파일(Small files)이 대량으로 생성됨. 클라우드 스토리지(S3, ADLS)는 파일 개수가 많으면 API 호출 비용(LIST, GET 등)이 급증합니다.
- 원인 2 (스토리지 티어): 데이터를 'Cool' 스토리지(저장 비용은 싸지만 접근 비용은 비쌈)에 보관하고, 이 데이터를 자주 접근하는 파이프라인을 실행함. (활성 데이터는 'Hot' 티어에 있어야 함)
- 해결: 처리 트리거 간격을 늘려 파일 크기를 키우고, 활성 데이터는 Hot/Warm 스토리지에 보관.

시나리오 2: 멀티 테넌시 (Multi-Tenancy)

- 목표: 여러 고객사(Tenant)의 데이터를 격리하여 처리.
- 잘못된 전략: 고객사별로 별도의 클러스터와 잡(Job)을 생성함.
- 문제: 고객사가 100개면 클러스터 100개가 필요. 비용이 폭증하고 리소스 활용률이 낮으며 운영이 복잡해짐.
- 권장 전략: 하나의 대형 클러스터에서 모든 고객 데이터를 처리하되, 데이터를 저장할 때 ‘partitionBy("client_id")’ . (DB) (View) .

주의사항

시나리오 3: 중복 제거(Deduplication)와 OOM (메모리 초과)

- 증상: `stream_df.dropDuplicates("user_id")` , Out-Of-Memory(OOM) 오류 .
- 원인 : dropDuplicates 는 중복을 확인하기 위해 지금까지 본 모든 user_id (State) . (Statestore) .
- 해결 (워터마크 사용): 워터마크(Watermark)를 사용하여 상태를 제한해야 합니다.

```

1 # Watermark: "ET" 컬럼기준분 10 지연까지허용
2 # 분이10 지난데이터의상태 (state)는 정리(clean up)됨
3 uniqueVisitors = stream_df \
4     .withWatermark("ET", "10 minutes") \
5     .dropDuplicates("ET", "uid")

```

- 설명: 워터마크는 "10분 늦게 오는 데이터까지만 중복을 체크하고, 그보다 더 늦으면 무시하겠다"는 의미입니다. 이를 통해 Spark는 10분이 지난 상태(State) 정보를 메모리에서 삭제할 수 있게 되어 OOM을 방지합니다.
- 대안 (더 안정적): foreachBatch와 MERGE INTO (Delta Lake 기능)를 사용하면 속도는 조금 느릴 수 있으나, 상태 관리가 필요 없어 더 안정적으로 중복 제거(Idempotent writes)가 가능합니다.

8.3 스트리밍 모범 사례 (Best Practices)

- SLA 및 요구사항 이해: 가장 중요합니다. TPS(초당 트랜잭션 수), E2E 지연 시간 요구사항을 명확히 합니다.
- 항상 체크포인트(Checkpoint) 사용: 내결함성(Fault Tolerance)과 복구를 위해 필수입니다.
- 처리 트리거 간격(Processing Trigger Interval) 조절: 비용 최적화를 위해 사용합니다. (예: 1분마다, 1시간마다). 실시간이 필요 없다면 24/7 실행할 필요가 없습니다.
- 쓰기 최적화(Optimized Writes): 작은 파일 문제를 피하기 위해 delta.autoOptimize.optimizeWrite = true 같은 옵션을 사용하여 파일 압축(Compaction)을 수행합니다.
- 용량 계획(Capacity Planning): Mux-Demux 아키텍처(모든 데이터를 Bronze에 모으고, 이후에 분리) 등을 고려하여 클러스터 용량을 계획합니다.

9 실습: 네트워크 로그 분석 (Lab 02: Network Analysis)

이 실습에서는 원시(Raw) Apache 네트워크 로그 파일을 Spark Structured Streaming을 사용하여 파싱, 정제, 보강(Enrich)하고 Delta Lake 테이블로 저장하는 전 과정을 다룹니다.

9.1 실습 목표 및 준비

- 목표:** 비정형(Unstructured) 텍스트 로그를 정형(Structured) 데이터로 변환하고, 외부 메타데이터와 조인하여 분석 가능한 델타 테이블을 생성합니다.
- 준비:** 실습을 위한 카탈로그, 스키마, 볼륨을 생성합니다.

1. 준비: 스키마 및 볼륨 생성 (Idempotent)

```

1 -- 이명령은여러번실행해도안전합니다며등성      ()
2 CREATE CATALOG IF NOT EXISTS csci_e103;
3 USE CATALOG csci_e103;
4
5 CREATE SCHEMA IF NOT EXISTS lab02;
6 USE SCHEMA lab02;
7
8 CREATE VOLUME IF NOT EXISTS lab02_input;
9 CREATE VOLUME IF NOT EXISTS lab02_output;
```

9.2 1단계: 원시 데이터 읽기 (Read Raw Data)

먼저, 텍스트 파일을 그대로 읽어옵니다.

2. 원시 로그 파일 읽기

```

1 # databricks-에 datasets 있는샘플로그경로
2 log_files_path = "/databricks-datasets/structured-streaming/events/"
3
4 # 텍스트파일로읽기
5 raw_log_files_df = spark.read.text(log_files_path)
6
7 # raw_log_files_df.printSchema()
8 # root
9 #   |-- value: string (nullable = true)
```

결과: DataFrame은 value라는 단 하나의 문자열(String) 컬럼을 가지며, 각 행에는 로그 한 줄이 통째로 들어가 있습니다.

9.3 2단계: 데이터 파싱 (Parse Data)

정규식(Regular Expression)을 사용하여 비정형 문자열을 여러 개의 구조화된 컬럼으로 분리합니다.

3. 정규식을 사용한 로그 파싱

```

1 from pyspark.sql.functions import regexp_extract
2
```

```

3 # Apache 로그형식을파싱하기위한정규식
4 log_pattern = r'(\S+) (\S+) (\S+) \[(.*?)\] "(\S+) (\S+) (\S+)" (\d{3}) (\d+) '
5
6 # 를 regexp_extract 사용하여각그룹을컬럼으로추출
7 parsed_df = raw_log_files_df.select(
8     regexp_extract('value', log_pattern, 1).alias('ip'),
9     regexp_extract('value', log_pattern, 4).alias('timestamp'),
10    regexp_extract('value', log_pattern, 5).alias('method'),
11    regexp_extract('value', log_pattern, 6).alias('endpoint'),
12    regexp_extract('value', log_pattern, 8).alias('response_code'),
13    regexp_extract('value', log_pattern, 9).alias('content_size')
14 )

```

9.4 3단계: 타임스탬프 변환 (UDF 사용)

주의사항

문제 발생: 비표준 타임스탬프 파싱된 timestamp 컬럼 (예: 28/Sep/2025:10:00:00 +0000)은 표준 SQL 형식이 아닙니다.

CAST(timestamp AS TIMESTAMP)를 실행하면 변환에 실패하여 NULL을 반환합니다.

▣ 핵심 요약

해결: UDF (User-Defined Function) 사용 표준 함수로 처리할 수 없는 복잡한 로직을 위해 사용자 정의 함수(UDF)를 만듭니다.

1. 먼저 Python 함수를 정의합니다.
2. 이 함수를 Spark SQL에서 사용 할 수 있도록 **SQL UDF로 등록합니다.**

4. UDF 정의 및 등록

```

1 import datetime
2
3 # 1. 비표준형식을파싱하는 Python 함수정의
4 def parse_datestr(datestr):
5     dt = datetime.datetime.strptime(datestr, '%d/%b/%Y:%H:%M:%S %z')
6     return dt.timestamp() # Unix timestamp (float)로 반환
7
8 # 2. Python 함수를 SQL로 UDF 등록
9 # 이름 : "parse_state", 반환타입 : "float"
10 spark.udf.register("parse_state", parse_datestr, "float")

```

이제 parse_state라는 함수를 SQL 쿼리 내에서 자유롭게 사용할 수 있습니다.

9.5 4단계: 데이터 보강 (Join Dimension)

로그 데이터의 response_code (예: 200, 404)는 숫자일 뿐 의미를 알 수 없습니다. 이 코드의 의미(예: "OK", "Not Found")가 담긴 외부 '차원 테이블(Dimension Table)'과 조인하여 데이터를 보강(Enrich)합니다.

5. 차원 테이블(JSON) 로드 및 조인

```

1 # 1. HTTP 상태코드메타데이터 (JSON 파일) 로드
2 status_codes_path = "..." # (JSON 파일경로 )
3 status_codes_df = spark.read.json(status_codes_path, multiLine=True)
4 # status_codes_df.printSchema()
5 #   -- code: string
6 #   -- description: string
7 #   -- ...
8
9 # 2. 를 UDF 사용하여로그데이터정리
10 clean_logs_df = spark.sql("""
11     SELECT
12         ip,
13         CAST(parse_state(timestamp) AS TIMESTAMP) AS event_time,
14         method,
15         endpoint,
16         response_code,
17         CAST(content_size AS int)
18     FROM log_data_view -- (를parsed_df 미리 Temp 로View 생성)
19 """)

20
21 # 3. 로그데이터와상태코드조인
22 network_df = clean_logs_df.join(
23     status_codes_df,
24     #
25     # [!!! 중요] 데이터타입일치시키기
26     #
27     # clean_logs_df.response_code (int)
28     # status_codes_df.code (string)
29     # -> 이대로조인하면실패하므로 , 타입을맞춰야함 .
30     #
31     clean_logs_df.response_code.cast("string") == status_codes_df.code,
32     "left_outer"
33 )

```

주의사항

[실무 핵심] 조인(Join) 전 데이터 타입 확인 조인가 실패하는 가장 흔한 원인은 키(Key)의 데이터 타입 불일치입니다. (예: Integer vs. String)

위 예제에서 로그의 `response_code`는 `int` 타입(예: 200)이었지만, 참조용 JSON 파일의 `code`는 `string` 타입(예: "200")이었습니다.

`.cast("string")`을 사용하여 타입을 강제로 일치시켜야만 조인이 성공합니다.

9.6 5단계: 필터링 및 저장 (Save to Delta)

최종적으로 정제되고 보강된 데이터를 분석용 델타 테이블로 저장합니다.

6. 최종 데이터 필터링 및 Delta Lake로 저장

```
1 # 1. 원하는 조건으로 데이터 필터링 예      (: 성공한 대용량 요청 )
2 final_df = network_df.filter(
3     (network_df.response_code == 200) &
4     (network_df.content_size > 100)
5 )
6
7 # 2. Delta Lake 테이블로 저장되어쓰기 (모드)
8 final_df.write \
9     .format("delta") \
10    .mode("overwrite") \
11    .saveAsTable("network_filter")
```

9.7 6단계: 확인 (Verify)

저장이 완료되면, Databricks의 'Catalog Explorer'에서 lab02 스키마 아래에 network_filter 테이블이 생성된 것을 확인할 수 있으며, SQL로 즉시 조회할 수 있습니다.

```
1 SELECT * FROM csci_e103.lab02.network_filter LIMIT 10;
2
3 SELECT description, COUNT(*)
4 FROM csci_e103.lab02.network_filter
5 GROUP BY description;
```

Listing 1: 최종 테이블 조회

10 빠른 훑어보기 (1-Page Summary)

▣ 핵심 요약

배치(Batch) vs. 스트리밍(Streaming)

- 배치 (Batch): ”대량의 빨래를 모아서” 한 번에 처리.
- 처리 단위: 대용량 데이터 묶음 (Bulk).
- 실행 시점: 주기적 (Scheduled) (예: 매 일 밤).
- 주요 용도: 일일 리포트, 굽여 정산.
- 스트리밍 (Streaming): ”물이 흐르는 즉시” 처리.
- 처리 단위: 개별 이벤트 (Event-driven).
- 실행 시점: 연속적 (Continuous) 또는 마이크로배치 (예: 1초마다).
- 주요 용도: 사기 탐지, 실시간 대시보드.

▣ 핵심 요약

람다(Lambda) vs. 카파(Kappa) 아키텍처

- 람다 (Lambda): 파이프라인 2개 (배치 + 스트리밍).
- 장점: 과거 기술로 속도와 신뢰성 동시 추구.
- 단점: 결과 일치(Reconciliation) 문제 발생. 코드 중복 및 복잡성.
- 카파 (Kappa): 파이프라인 1개 (스트리밍).
- 핵심: ”배치도 결국 트리거가 긴 스트림이다.” (`spark.readStream`)
- 장점: 단순함 (단일 코드베이스), 결과 일치 문제 원천 차단.

주의사항

스트리밍 핵심 개념 Top 2

- 체크포인트 (Checkpoint): ”게임 저장 지점”.
- 역할: 장애 발생 시 어디부터 다시 시작할지 알려줌. 데이터 유실 및 중복 방지 (Exactly-once).
- 스트리밍의 필수 요소.
- 워터마크 (Watermark): ”지연 데이터 허용 시간” (예: 10분).
- 역할: 집계 연산 시 상태(State)가 무한정 커지는 것을 방지함. 10분이 지난 상태 정보는 메모리에서 삭제하여 OOM(메모리 초과) 오류를 막음.

실습(Lab) 핵심 요약

- UDF (User-Defined Function): CAST 등 표준 함수로 변환할 수 없는 비표준 형식(예: 타임스탬프)을 처리하기 위해 사용자 정의 함수를 만들어 사용.
- Join 타입 매칭: 조인(Join)을 수행할 때는 두 DataFrame의 키(Key) 컬럼 데이터 타입이 일치하는지 반드시 확인해야 함. (예: int 200 vs. string "200" → `cast("string")` 필요)

December 10, 2025

■ 강의명: CSCI E-103: 재현 가능한 머신러닝

■ 주차: Lecture 04

■ 교수명: Anindita Mahapatra

Eric Gieseke

■ 목적: Lecture 04의 핵심 개념 학습

Contents

1 개요 (Overview)	2
2 지난 강의 복습 (Review of Lecture 3)	2
3 데이터 엔지니어링 디자인 패턴 (Design Patterns)	4
3.1 디자인 패턴이란? (What is a Design Pattern?)	4
3.2 빅데이터 디자인 패턴 분류	4
3.3 주요 아키텍처 및 스토리지 패턴	5
3.4 기타 주요 처리 및 서비스 패턴	6
4 데이터 컴플라이언스 (Data Compliance)	7
4.1 주요 규제: GDPR과 CCPA	7
4.2 데이터 엔지니어의 과제: 핵심 권리	7
4.3 기술적 해결 방안	7
5 Spark 핵심 개념: 실행 구조와 파티션	8
5.1 Spark 실행 아키텍처 (Jobs → Stages → Tasks)	8
5.2 테이블 파티션 vs. Spark 파티션 (매우 중요)	8
5.3 Z-Ordering (Z-순서 지정)	9
6 Spark 조인 전략 (Spark Join Strategies)	10
7 데이터 조작 (CRUD) 및 스키마 관리	11
7.1 CRUD (Create, Read, Update, Delete)	11
7.2 Upsert와 Merge (중요)	11
7.3 스키마 관리 (Schema Management)	11
8 변경 데이터 관리 (CDC & SCD)	13

8.1	CDC (Change Data Capture, 변경 데이터 캡처)	13
8.2	SCD (Slowly Changing Dimensions, 느리게 변하는 차원)	13
8.3	SCD Type 1 및 Type 2 상세 예시	14
8.3.1	SCD Type 1: Overwrite (덮어쓰기)	14
8.3.2	SCD Type 2: New Row (새 행 추가)	14
9	고급 기능 및 UDF (Advanced Features & UDFs)	16
9.1	Delta Lake 관리 기능	16
9.2	UDF (사용자 정의 함수)	16
10	실습: Autoloader를 이용한 데이터 로딩	18
10.1	Autoloader 란?	18
10.2	Autoloader 핵심 옵션 및 코드	18
10.3	불량 데이터 처리 (Handling Bad Data)	18
10.4	스키마 힌트 (Schema Hints)	19
10.5	스트리밍 중단 (Stopping Streams)	19
11	용어 정리 (Glossary)	20
12	FAQ 및 과제 Q&A (FAQ and Homework Q&A)	21
13	한눈에 보기 (Quick Look Summary)	22

1 개요 (Overview)

이번 4강에서는 데이터 엔지니어링의 핵심 작업인 '데이터 변환'에 대해 배웁니다.

데이터 파이프라인을 설계할 때 자주 사용되는 검증된 해결책인 **디자인 패턴**을 학습합니다. 또한, 데이터를 다룰 때 반드시 지켜야 할 법적 규제(GDPR, CCPA)인 **컴플라이언스**의 중요성을 이해합니다.

Spark의 내부 동작 원리인 **파티션, Z-Ordering, 조인 전략**을 살펴보고, **CRUD, CDC, SCD**와 같은 데이터 변경 및 이력 관리 기법을 배웁니다. 마지막으로, **Autoloader**를 사용해 클라우드 스토리지의 파일을 충분 처리하는 실습을 진행합니다.

▣ 핵심 요약

이번 세션의 핵심 목표

- 데이터 아키텍처의 주요 디자인 패턴(Lambda, Kappa, Data Mesh 등)을 이해합니다.
- GDPR과 같은 데이터 규제가 엔지니어링에 미치는 영향을 파악합니다.
- Spark의 성능 최적화 핵심인 파티션(테이블 vs Spark)과 조인 전략을 구분합니다.
- CDC(변경 데이터 캡처)와 SCD(느리게 변하는 차원)의 차이와 구현(Type 1, 2)을 이해합니다.
- Autoloader를 사용하여 스키마 진화 및 불량 데이터를 처리하는 방법을 실습합니다.

2 지난 강의 복습 (Review of Lecture 3)

데이터 변환을 배우기에 앞서, 지난 강의에서 다룬 스트리밍과 데이터 아키텍처의 핵심 용어들을 복습합니다.

- 스트리밍 유형 (2가지):**
 - 파일 기반 (File-based):** 파일이 도착하는 것을 이벤트로 감지하여 처리합니다.
 - 이벤트 기반 (Event-based):** Kafka, Pulsar 등 메시징 큐를 통해 개별 이벤트(메시지)가 발생하는 즉시 처리합니다.
- 체크포인팅 (Checkpointing):** 스트리밍 작업이 실패했을 때를 대비하여, '어디까지 처리했는지' 마지막으로 성공한 위치(오프셋)를 저장하는 것입니다. 작업이 재시작되면 이 저장된 지점부터 다시 시작하여 데이터 유실이나 중복 처리를 방지합니다.
- 워터마킹 (Watermarking):** '얼마나 늦게 도착하는 데이터까지 허용할 것인가'를 정의하는 기준입니다. 예를 들어, 워터마크를 10분으로 설정하면, 특정 윈도우(예: 10:00 10:10)가 닫힌 후 10분 (즉, 10:20)이 지나면, 10:00 10:10에 속하는 데이터가 늦게 도착하더라도 무시(삭제)합니다. 집계의 정확성과 속도 사이의 트레이드오프입니다.
- 스트리밍 출력 모드 (Streaming Output Modes):**
 - Append (추가):** 처리된 새로운 행만 출력합니다. (워터마크 없이는 집계에 사용 불가)
 - Update (업데이트):** 변경된 행만 출력합니다.
 - Complete (전체):** 전체 결과 테이블을 매번 다시 계산하여 출력합니다.
- 처리 트리거 (Processing Trigger):** 마이크로배치(Micro-batch) 스트리밍에서 '얼마나 자주' 배치를 실행할지 결정하는 간격입니다. (예: 1초마다)
- Lambda vs. Kappa 아키텍처:**

- **Lambda (람다):** 속도가 중요한 실시간 처리(Speed Layer)와 정확도가 중요한 배치 처리(Batch Layer)를 두 개의 별도 파이프라인으로 운영한 뒤, 서빙 레이어에서 합치는 구조입니다. (복잡하지만 안정적)
- **Kappa (카파):** 하나의 스트리밍 파이프라인으로 모든 것을 처리합니다. 배치가 필요하면 스트림을 처음부터 다시 읽어옵니다. (단순하지만 스트리밍 시스템에 대한 의존도가 높음)
- **데이터 레이크하우스 (Lakehouse):** 데이터 웨어하우스(정형 데이터, BI)의 장점과 데이터 레이크(원시 데이터, ML)의 장점을 결합한 현대적인 아키텍처입니다.
- **관리형 vs. 외부 테이블 (Managed vs. External):**
 - **관리형 (Managed):** Spark가 데이터와 메타데이터(스키마 정보)를 모두 관리합니다. ‘DROP TABLE’ 시 데이터와 메타데이터가 모두 삭제됩니다.
 - **외부 (External):** Spark는 메타데이터만 관리하고, 실제 데이터는 외부(예: S3)에 있습니다. ‘DROP TABLE’ 시 메타데이터만 삭제되고 원본 데이터는 유지됩니다.
- **메타데이터 (Metadata):** 데이터에 대한 데이터(스키마, 소유자, 위치 등)입니다. 메타데이터가 잘 관리되면 데이터 검색 가능성(Discoverability)과 데이터 거버넌스(Governance, 통제 및 관리)가 향상됩니다.
- **구체화된 뷰 (Materialized View):** 일반적인 뷰(View)는 쿼리 시점에 계산되는 ‘가상 테이블’이지만, 구체화된 뷰는 쿼리 결과를 ‘미리 계산하여 물리적으로 저장’해 둔 테이블입니다.
 - **장점:** 쿼리 속도가 매우 빠릅니다.
 - **단점:** 원본 데이터가 변경되어도 즉시 반영되지 않아 데이터가 오래될(stale) 수 있으며, 별도의 저장 공간이 필요합니다. (주기적인 새로고침(Refresh) 필요)

3 데이터 엔지니어링 디자인 패턴 (Design Patterns)

3.1 디자인 패턴이란? (What is a Design Pattern?)

디자인 패턴은 소프트웨어 엔지니어링이나 데이터 엔지니어링에서 **자주 발생하는 공통적인 문제에 대한 검증된 해결책(템플릿)**입니다.

□ 예제: title

요리를 할 때 '재료를 기름에 볶는다'(볶음), '물에 오래 끓인다'(찜) 같은 검증된 조리법이 있습니다. 데이터 엔지니어링에서도 '대용량 데이터를 실시간과 배치로 동시에 처리하는 문제'에 대해 '람다 아키텍처'라는 검증된 레시피(패턴)가 있습니다.

디자인 패턴은 다음과 같은 이점을 제공합니다.

- 좋은 설계 원칙 증진: 추상화(Abstractation), 관심사 분리(Separation of Concerns), 문제 분할(Divide and Conquer) 등 좋은 설계 원칙을 자연스럽게 따르도록 유도합니다.
- 공통 어휘 제공: 팀원 간에 "이 부분은 람다 패턴을 적용하죠"라고 말하면, 복잡한 설계를 매번 설명할 필요 없이 효율적으로 소통할 수 있습니다.
- 재사용성 및 안정성: 이미 많은 사람이 사용하고 검증한 해결책이므로, 처음부터 설계를 고민하는 것보다 안정적이고 효율적입니다.

주의: 맹목적인 적용 금지

디자인 패턴은 만병통치약이 아닙니다. 내가 해결하려는 문제의 맥락(데이터 크기, 속도, 비용 등)에 맞는지 신중하게 검토하고 적용해야 합니다.

3.2 빅데이터 디자인 패턴 분류

빅데이터 환경(대용량, 고속도, 다양성)에서는 전통적인 소프트웨어 디자인 패턴(GoF 패턴)과는 다른, 데이터 처리에 특화된 패턴들이 사용됩니다.

데이터 파이프라인의 각 단계별로 적용 할 수 있는 주요 디자인 패턴은 다음과 같습니다.

단계	주요 패턴	설명 (해결하려는 문제)
모델링 (Modeling)	Star/Snowflake Schema Vault Modeling	분석(BI)에 최적화된 데이터 웨어하우스 구조(사실 테이블 + 차원 테이블) 데이터의 모든 변경 이력을 원본 그대로 저장하는 모델링 기법
수집 (Ingestion)	Connectors Lambda / Kappa Compute/Storage Separation	다양한 데이터 소스(DB, API, 파일)에 일관된 방식으로 연결 실시간 및 배치 데이터를 처리하는 아키텍처(앞서 복습) 계산 자원과 저장 자원을 분리하여 각각 독립적으로 확장(비용 효율화)
변환 (Transformation)	Schema on Read / Write ACID Transactions Multi-Hop Pipeline	스키마(데이터 구조)를 언제 정의 할지 결정 대용량 데이터 환경에서 원자성, 일관성, 고립성, 지속성을 보장(예: Delta Lake) 데이터를 여러 단계(Bronze → Silver → Gold)로 정제 (Medallion 아키텍처)
저장 (Persist)	Columnar Storage Denormalized Tables	데이터를 행(Row) 단위가 아닌 열(Column) 단위로 저장(예: Parquet). 분석 쿼리 속도 향상. 정규화를 낮추고(중복 허용) 테이블을 넓게 만들어, 비싼 조인(Join) 연산을 피함.
분석 (Analytics)	In-Stream Analytics	데이터가 저장되기 전, 스트리밍 중에 실시간으로 분석(예: 실시간 사기 탐지)

데이터 파이프라인 단계별 디자인 패턴

3.3 주요 아키텍처 및 스토리지 패턴

- **이벤트 기반 아키텍처 (EDA, Event-Driven Architecture):** 시스템의 모든 동작이 '이벤트'(상태 변경 또는 데이터 입력)에 대한 반응으로 설계된 구조입니다.
 - 구성: 이벤트 생산자(Producer) → 이벤트 스트리밍(예: Kafka) → 이벤트 소비자(Consumer)
 - 용도: 마이크로서비스, 실시간 전자상거래 주문 처리 등
- **CQRS (Command Query Responsibility Segregation):** '관심사 분리' 원칙의 극단적인 예로, 시스템의 명령(Write/Update)과 조회(Read/Query) 책임을 완전히 분리하는 패턴입니다.
 - 목적: 쓰기 작업(Command)에 최적화된 모델과 읽기 작업(Query)에 최적화된 모델을 따로 설계하여, 각각 독립적으로 확장하고 성능을 극대화합니다.
 - 예시: 쓰기는 빠르지만 조회가 느린 DB에 데이터를 쓰고, 이 데이터를 읽기에 최적화된 다른 DB(예: 검색 엔진)로 복제하여 사용자가 조회하게 만듭니다.
- **폴리글랏 퍼시스턴스 (Polyglot Persistence):** 'Polyglot'은 '여러 언어를 사용하는' 이란 뜻입니다. 이 패턴은 "망치(하나의 DB)를 들고 모든 것을 뜯으로 보지 말라"는 철학입니다.
 - 목적: 단 하나의 데이터베이스 기술로 모든 문제를 해결하려 하지 않고, 각 기능에 가장 적합한 DB

를 여러 개 조합하여 사용하는 것입니다.

- 예시: 트랜잭션 데이터는 RDBMS(관계형 DB)에, 비정형 문서는 Document DB(예: MongoDB)에, 친구 관계망은 Graph DB(예: Neo4j)에 저장합니다.
- 데이터 레이크 패턴 (Data Lake Pattern): 모든 데이터를(정형, 비정형) 원본(Raw) 형식 그대로 저장하는 거대한 저장소입니다.
 - 목적: ”일단 저장하고, 나중에 어떻게 쓸지 결정하자”는 접근입니다. 데이터가 어떻게 사용될지 모르는 미래의 분석이나 ML을 위해 원본을 보존합니다.
 - 특징: 스키마 온 리드(Schema on Read) 방식을 사용합니다. (데이터를 읽는 시점에 스키마 정의)

3.4 기타 주요 처리 및 서비스 패턴

- CAP 정리 (CAP Theorem): 분산 시스템(여러 대의 컴퓨터가 협력하는 시스템)은 세 가지 속성(일관성, 가용성, 분할 내성) 중 최대 두 가지만 동시에 만족시킬 수 있다는 이론입니다.
 - Consistency (일관성): 모든 노드가 동시에 같은 데이터를 보여줌.
 - Availability (가용성): 모든 요청에 대해 (오래된 데이터일지라도) 항상 응답함.
 - Partition Tolerance (분할 내성): 노드 간 통신이 끊겨도 시스템이 계속 동작함.
- 빅데이터 시스템은 대부분 P(분할 내성)를 기본으로 선택하고, C(일관성)와 A(가용성) 사이에서 타협합니다.
 - CP 시스템 (일관성 우선): RDBMS, HBase. 데이터가 틀릴 바에는 차라리 응답하지 않습니다.
 - AP 시스템 (가용성 우선): DynamoDB, Cassandra. 데이터가 조금 틀리더라도(최종적 일관성) 항상 응답합니다.
- 데이터 메시 (Data Mesh): 최신 아키텍처 패턴으로, 중앙화된 데이터 팀(병목 현상) 대신, 각 도메인(현업 부서) 팀이 자신의 데이터를 ‘제품(Data as a Product)’처럼 직접 소유하고 관리/제공하는 분산형 아키텍처입니다.
- 멀티 흡 (Multi-Hop) / 메달리온 (Medallion) 아키텍처: 데이터를 여러 단계에 걸쳐 점진적으로 정제하는 가장 일반적인 패턴입니다. ‘관심사 분리’와 ‘문제 분할’ 원칙을 따릅니다.
 - Bronze (Landing): 원본(Raw) 데이터. 최소한의 변환만 거쳐 저장. (데이터 엔지니어용)
 - Silver (Refined): 비즈니스 로직 적용. 정제되고(Cleaned), 필터링되고, 보강된 데이터. (데이터 사이언티스트용)
 - Gold (Aggregated): 최종 집계 데이터. BI 대시보드나 리포트를 위한 ‘사용 준비 완료’ 상태. (비즈니스 분석가용)

이 구조의 장점은 파이프라인이 모듈화되어 디버깅이 쉽고, 데이터 품질은 Gold로 갈수록 높아지지만, 데이터 가용성(속도)은 Bronze가 가장 빠르다는 것입니다.
- 데이터 이동 최소화 (Minimize Data Movement): 대용량 데이터를 네트워크로 복사하는 것은 비용과 시간이 많이 듭니다. 이를 피하기 위한 패턴입니다.
 - Time Travel (시간 여행): 데이터의 변경 이력을 버전별로 관리하여, 데이터를 복사하지 않고도 과거 특정 시점의 스냅샷을 조회할 수 있게 합니다. (예: Delta Lake)
 - Zero Copy Clone (제로 카피 복제): 실제 데이터를 복사하지 않고, 메타데이터만 복제하여 새로운 테이블을 즉시 생성합니다. 스토리지 비용 없이 테스트 환경을 만들 수 있습니다.

4 데이터 컴플라이언스 (Data Compliance)

데이터 엔지니어는 데이터를 기술적으로 처리할 뿐만 아니라, 법적/윤리적 책임을 가지고 데이터를 관리해야 합니다.

4.1 주요 규제: GDPR과 CCPA

- **GDPR (General Data Protection Regulation):** 유럽 연합(EU)의 일반 데이터 보호 규정입니다. EU 시민의 개인정보를 처리하는 전 세계 모든 기업에 적용됩니다.
- **CCPA (California Consumer Privacy Act):** 미국 캘리포니아주의 소비자 개인정보 보호법입니다.

이러한 규제들은 조직이 개인 데이터를 합법적이고 엄격한 조건 하에 수집/관리하도록 강제하며, 데이터 소유자(소비자)의 권리를 존중할 의무를 부여합니다. 위반 시 막대한 과징금이 부과될 수 있습니다.

4.2 데이터 엔지니어의 과제: 핵심 권리

데이터 엔지니어링 관점에서 컴플라이언스는 다음 두 가지 핵심 권리를 기술적으로 구현해야 하는 과제를 의미합니다.

1. **삭제권 (Right of Erasure) / 잊힐 권리 (Right to be Forgotten):** 소비자가 자신의 개인정보 삭제를 요청할 경우, 시스템에서 해당 데이터를 완전히 삭제해야 합니다.

기술적 난제

데이터 레이크에 저장된 수백 TB의 데이터 속에서 특정 사용자 1명의 데이터를 찾아내어 정확하게 삭제하는 것은 매우 어렵습니다. 특히 Parquet 같은 불변(immutable) 파일 포맷에서는 특정 행만 삭제하기가 거의 불가능합니다. (Delta Lake 같은 기술이 이 문제를 해결해줍니다.)

2. **이동권 (Right of Portability):** 소비자가 자신의 데이터를 (경쟁사 등) 다른 서비스로 이전할 수 있도록, 식별 가능하고 재사용 가능한 형태로 추출하여 제공해야 합니다.

4.3 기술적 해결 방안

- **세분화된 업데이트/삭제 (Fine-grained Updates/Deletes):** Delta Lake, Hudi, Iceberg와 같은 최신 데이터 레이크 포맷은 대용량 데이터셋에 대해 특정 행을 업데이트하거나 삭제하는 기능을 지원하여 GDPR의 삭제권 준수를 돋습니다.
- **가명화 (Pseudonymization):** 개인정보(식별자)를 외부에서 식별 불가능한 키(가명, 토큰)로 대체(토크화)하는 기법입니다.
 - **작동 방식:** 원본 [사용자 ID ↔ 토큰] 매핑 테이블을 안전한 곳에 별도로 보관하고, 모든 분석 시스템에서는 '토큰'만 사용합니다.
 - **삭제 요청 시:** 매핑 테이블에서 해당 사용자의 [ID ↔ 토큰] 연결 고리만 파괴하면, 시스템에 데이터가 남아있더라도 더 이상 해당 사용자와 연결(추적) 할 수 없게 됩니다.

5 Spark 핵심 개념: 실행 구조와 파티션

5.1 Spark 실행 아키텍처 (Jobs → Stages → Tasks)

우리가 작성한 Spark 코드(예: PySpark, SQL)는 클러스터에서 복잡한 계층 구조로 나뉘어 실행됩니다.

Spark 작업 실행 계층

1. **Job (작업):** Spark 드라이버(Driver) 프로그램에 제출된 '액션(Action)'(예: 'save()', 'collect()') 하나당 하나의 Job이 생성됩니다.
2. **Stage (스테이지):** 하나의 Job은 셔플(Shuffle)이 발생하는 지점을 기준으로 여러 개의 Stage로 나뉩니다.
 - 셔플(Shuffle)이란? 'groupBy', 'join' 등 데이터 재분배가 필요한 작업 시, 여러 작업자(Worker) 노드 간에 데이터가 네트워크를 통해 교환되는 비싼 작업입니다.
 - Stage는 셔플 없이 한 번에 실행될 수 있는 작업들의 묶음입니다.
3. **Task (태스크):** Stage 내에서 실행되는 가장 작은 작업 단위입니다.
 - 하나의 Task는 하나의 파티션(데이터 조각)에 대해 동일한 연산을 수행합니다.
 - Task는 실제 작업자인 Executor의 코어에서 병렬로 실행됩니다.

요약: Job (액션 기준) → Stages (셔플 기준) → Tasks (파티션 기준)

5.2 테이블 파티션 vs. Spark 파티션 (매우 중요)

초심자가 가장 혼동하기 쉬운 개념으로, 두 '파티션'은 목적과 작동 방식이 완전히 다릅니다.

▣ 핵심 비유: 도서관

- **테이블 파티션(물리적 책장):** 도서관에서 책을 '국가별'(예: 한국, 미국)로 다른 책장에 나눠 끊는 것입니다. 디스크 I/O를 줄이는 것이 목적입니다.
- **Spark 파티션(논리적 작업자):** 도서관의 모든 책을 정리하기 위해 '몇 명의 사서'에게 작업을 나눠줄지 정하는 것입니다. CPU 병렬성을 높이는 것이 목적입니다.

아래 표는 두 파티션의 차이점을 자세히 비교합니다.

특징	테이블 파티셔닝(Table Partitioning)	Spark 파티셔닝(Spark Partitioning)
개념/범위	(DB/테이블 수준) 대용량 테이블을 특정 컬럼 값(예: 날짜, 국가)에 따라 더 작은 조각(파일 디렉토리)으로 분할.	(Spark 처리 수준) 데이터를 클러스터 노드 간에 어떻게 분산시켜 병렬 처리할지 정하는 논리적 단위.
목적	데이터 스캔 최소화(Partition Pruning) 쿼리 성능 향상(디스크 I/O 감소)	병렬 처리(Parallelism) 최적화 리소스 활용률, 내 결함성(Fault Tolerance) 향상
관리 주체	사용자가 명시적으로 정의 (예: 'CREATE TABLE ... PARTITIONED BY (date)')	Spark가 동적으로 결정 (입력 파일 크기, 클러스터 설정, 셔플 등에 따라)
최적화	파티션 프루닝(Partition Pruning): 'WHERE date='2025-01-01'' 쿼리 시, 다른 날짜의 파티션(디렉토리)은 스캔조차 안 함.	셔플(Shuffle) 최소화: 'join', 'groupBy' 시 데이터 재분배(셔플) 방식에 영향을 주어 네트워크 오버헤드를 줄임.

이를 파티션과 Spark 파티션 비교

테이블 파티셔닝 모범 사례:

- ‘WHERE’ 절에서 자주 사용되는 컬럼을 선택합니다. (예: 날짜, 국가, 지역)
- 카디널리티(Cardinality, 고유값의 개수)가 너무 높은 컬럼(예: 사용자 ID)은 피해야 합니다. 파티션 이 수백만 개로 잘게 쪼개져 오히려 성능이 저하됩니다.
- 각 파티션의 데이터 크기는 최소 1GB 이상이 되도록 하는 것이 좋습니다. (작은 파일 문제 방지)
- (Databricks 기준) 최근에는 데이터가 1TB 미만이면 굳이 파티셔닝하지 않는 것을 권장합니다. 내부 최적화 엔진(Delta Lake)이 더 잘 처리할 수 있습니다.

5.3 Z-Ordering (Z-순서 지정)

Z-Ordering은 테이블 파티셔닝을 보완하는 Delta Lake의 데이터 레이아웃 최적화 기법입니다.

- **목적:** 파티션 키로 사용하지 않았지만 자주 함께 조회되는 여러 컬럼의 데이터들을 물리적으로 가깝게 모아(데이터 건너뛰기, Data Skipping) 쿼리 속도를 향상시킵니다.
- **작동 방식:** 다차원(여러 컬럼)의 값들을 Z-커브라는 알고리즘을 사용해 1차원의 값으로 변환한 뒤, 이 값을 기준으로 데이터를 정렬합니다.
- **예시:** ‘(IP 주소, 포트)’처럼 자주 같이 필터링되지만 파티션 키로는 부적절한 컬럼들에 ‘ZORDER BY (ipaddress, port)’, .

```

1 -- 파일들을 압축 (Bin-packing)하고
2 -- eventType 컬럼을 기준으로 데이터를 재정렬 (Z-Ordering)
3 OPTIMIZE events
4 ZORDER BY (eventType)

```

Listing 1: OPTIMIZE와 Z-Ordering 사용 예시

6 Spark 조인 전략 (Spark Join Strategies)

Spark SQL은 두 테이블을 조인할 때, 데이터의 크기, 분포, 힌트 등을 고려하여 가장 효율적인 조인 전략을 자동으로 선택합니다. 어떤 전략이 있는지 아는 것은 성능 튜닝에 매우 중요합니다.

전략	설명	요구 조건	특징 (장/단점)
Broadcast Hash Join (BHJ)	하나의 작은 테이블을 모든 Executor에 복제(Broadcast)하여 메모리에 올린 뒤, 큰 테이블의 파티션과 해시 조인.	한쪽 테이블이 매우 작아야 함 (기본 10MB, 설정 가능)	가장 빠름. 셔플(Shuffle) 없음. 정렬(Sort) 없음.
Shuffle Hash Join (SHJ)	두 테이블을 조인 키 기준으로 셔플(재분배)한 뒤, 각 Executor에서 더 작은 쪽의 해시 테이블을 만들며 조인.	한쪽 테이블이 (전체는 아니어도) 각 파티션이 메모리에 들어갈 만큼 작아야 함.	셔플 발생. (단점) 정렬(Sort) 없음. (장점) 데이터가 심하게 쏠리면(Skew) OOM(메모리 부족) 발생 가능.
Sort-Merge Join (SMJ)	두 테이블을 조인 키 기준으로 셔플(재분배)하고, 각 파티션 내에서 정렬(Sort)한 뒤, 병합(Merge)하며 조인.	조인 키가 정렬 가능해야 함.	가장 안정적. (기본 전략) 셔플과 정렬 모두 필요. (단점) 어떤 크기의 데이터도 처리 가능.
Cartesian Join (Cross Join)	조인 조건이 없거나 비효율적일 때 가능한 모든 행의 조합($N \times M$)을 생성.	(없음)	최악의 조인. (피해야 함) 거대한 셔플 발생.

의 주요 조인 전략 비교

□ Spark 조인 힌트

Spark 옵티마이저가 잘못된 전략을 선택할 경우, SQL 쿼리 내에 힌트('/*+ BROADCAST(*small_table*) */').

7 데이터 조작(CRUD) 및 스키마 관리

7.1 CRUD (Create, Read, Update, Delete)

빅데이터 환경, 특히 Delta Lake에서는 전통적인 RDBMS와 유사하게 CRUD 작업을 수행할 수 있습니다.

- **Create (생성):**
 - **Append (추가):** 기존 데이터에 새 데이터를 추가합니다. (기본 모드)
 - **Overwrite (덮어쓰기):** 기존 데이터를 모두 삭제하고 새 데이터로 교체합니다. ('.mode("overwrite")')
- **Read (읽기):** ‘SELECT’, ‘.read()’. 파티션, 키, Z-Ordering을 활용하여 효율적으로 읽습니다.
- **Update (수정):** ‘UPDATE ... SET ... WHERE ...’. 특정 조건을 만족하는 행의 값을 수정합니다. (Delta Lake 지원)
- **Delete (삭제):** ‘DELETE FROM ... WHERE ...’. 특정 조건을 만족하는 행을 삭제합니다. (Delta Lake 지원)

7.2 Upsert와 Merge (중요)

Upsert는 ‘Update + Insert’의 합성어로, 데이터 엔지니어링에서 매우 흔하게 발생하는 작업입니다.

□ 예제: title

매일 밤 소스 시스템에서 최신 고객 목록을 가져옵니다.

- 이 고객이 타겟 테이블에 없으면 → **INSERT** (신규 고객)
- 이 고객이 타겟 테이블에 있으면 → **UPDATE** (기존 고객 정보 변경)

이 Upsert 로직은 Delta Lake의 ‘MERGE’ 명령어를 통해 하나의 원자적(atomic) 명령으로 간단하게 처리할 수 있습니다.

```

1 MERGE INTO target_table AS t
2 USING source_updates AS s
3 ON t.id = s.id
4 WHEN MATCHED THEN
5   -- 조건이맞으면기존 ( 고객 ) 업데이트
6   UPDATE SET t.name = s.name, t.address = s.address
7 WHEN NOT MATCHED THEN
8   -- 조건이안맞으면신규 ( 고객 ) 삽입
9   INSERT (id, name, address) VALUES (s.id, s.name, s.address)

```

Listing 2: MERGE INTO를 사용한 Upsert 구현

7.3 스키마 관리 (Schema Management)

데이터 파이프라인 운영 시 스키마(데이터 구조) 변경은 피할 수 없는 문제입니다. Spark는 3가지 스키마 관리 전략을 제공합니다.

1. **스키마 추론 (Schema Inference):** Spark가 데이터(예: JSON, CSV)를 샘플링하여 스키마를 자동으로 추측합니다. 간편하지만, 데이터가 크거나 형식이 불일치하면(예: ‘id’가 어려 땐 숫자, 어려 땐

문자열) 부정확하거나 성능이 저하될 수 있습니다.

2. **스키마 진화 (Schema Evolution):** 기존 스키마에 새로운 컬럼이 추가되는 것을 허용하는 옵션입니다. (예: ‘df.write.option(“mergeSchema”, “true”) ...’)
 - **장점:** 파이프라인 중단 없이 소스의 스키마 변경(컬럼 추가)에 유연하게 대응할 수 있습니다.
 - **주의:** 데이터 타입이 호환되지 않게 변경(예: String → Int)되는 것은 막지 못할 수 있습니다.
3. **스키마 강제 (Schema Enforcement):** 기본 동작입니다. 저장하려는 데이터의 스키마가 타겟 테이블의 스키마와 일치하지 않으면 에러를 발생시키고 작업을 중단시킵니다.
 - **장점:** 데이터 품질을 엄격하게 관리할 수 있습니다. 예기치 않은 데이터(예: ‘age’ 컬럼에 문자열)가 유입되는 것을 막아줍니다.

□ 권장 파이프라인 전략 (Multi-Hop)

- **Bronze (수집 단계):** 스키마 진화 (Schema Evolution) 허용. (소스에서 어떤 변경이 생길지 모르므로, 일단 모든 데이터를 받아들입니다.)
- **Silver/Gold (정제/집계 단계):** 스키마 강제 (Schema Enforcement) 적용. (데이터 컨트랙트가 확정된 단계이므로, 약속된 스키마 외의 데이터는 거부하여 품질을 보장합니다.)

8 변경 데이터 관리 (CDC & SCD)

데이터는 정지해 있지 않고 끊임없이 변경됩니다. 이러한 변경 사항을 추적하고 관리하는 기법은 매우 중요합니다.

8.1 CDC (Change Data Capture, 변경 데이터 캡처)

CDC는 소스(Source) 시스템에서 발생한 변경 사항(Insert, Update, Delete)을 식별하고 '캡처'하는 기술입니다.

- 목적:** 전체 테이블을 매번 복사(Full Load)하는 대신, 변경된 내용만(Incremental Load) 가져와 타겟 시스템에 효율적으로 반영하기 위함입니다.
- 예시:** 데이터베이스의 트랜잭션 로그를 읽어 'A 고객의 주소가 변경됨', 'B 고객이 삭제됨' 같은 이벤트 로그를 생성합니다.

Delta Lake의 **Change Data Feed (CDF)** 기능은 Delta 테이블 자체에 대한 변경 내역(pre-image, post-image)을 로그로 제공하여, Delta 테이블을 CDC의 소스로 활용할 수 있게 해줍니다.

8.2 SCD (Slowly Changing Dimensions, 느리게 변하는 차원)

SCD는 CDC로 캡처된 변경 사항을 타겟(Target) 시스템(주로 데이터 웨어하우스의 차원 테이블)에 어떻게 반영할지 결정하는 정책입니다.

'느리게 변하는 차원'(예: 고객, 제품, 매장)은 트랜잭션(사실) 데이터만큼 자주 변하지는 않지만, 변경 시 그 이력을 어떻게 관리할지가 중요합니다.

- Type 0 (Fixed, 고정):** 변경을 허용하지 않습니다. (예: 입사일)
- Type 1 (Overwrite, 덮어쓰기):** 이력을 관리하지 않습니다. 변경 사항이 발생하면 기존 데이터를 그냥 덮어씁니다.
- Type 2 (New Row, 새 행 추가):** 모든 변경 이력을 관리합니다. 변경 사항이 발생하면 기존 행은 '만료' 처리하고, 새로운 행을 추가하여 이력을 쌓습니다.
- Type 3 (New Column, 새 컬럼 추가):** 이전 값을 저장하기 위해 'previous_address' 같은 새 컬럼을 추가합니다. (이력이 몇 개 없을 때만 사용, 비추천)
- Type 4 (History Table, 이력 테이블):** 현재 값을 원본 테이블에 유지하고, 모든 변경 이력은 별도의 이력 테이블에 저장합니다.
- Type 6 (Hybrid):** Type 1, 2, 3을 조합한 하이브리드 방식입니다.

핵심: Type 1 vs Type 2

실무에서 가장 많이 사용되는 것은 Type 1과 Type 2입니다.

- Type 1:** 이력이 필요 없을 때. (예: 잘못된 스펠링 수정)
- Type 2:** 모든 변경 이력 추적이 필요할 때. (예: 고객 주소 변경에 따른 매출 지역 분석)

8.3 SCD Type 1 및 Type 2 상세 예시

8.3.1 SCD Type 1: Overwrite (덮어쓰기)

정책: 이력 없음. 변경 시 원본 데이터를 덮어씁니다.

cust_id	name
---------	------

초기 원본 테이블 (Target Dimension Table):

1	A
2	B

변경 데이터 발생 (Incoming Data):

- ID 1: 이름 'A' → 'AA'로 변경 (Update)
- ID 2: 삭제 (Delete)
- ID 3: 신규 추가 (Insert)

cust_id	name	action
---------	------	--------

SCD Type 1 적용 후 최종 테이블:

1	AA
3	C

Type 1 결과 (ID 2는 삭제되고 ID 1은 덮어쓰기됨) 단점: ID 1의 이름이 'A'였다는 사실과 ID 2가 존재했다는 이력이 사라집니다.

8.3.2 SCD Type 2: New Row (새 행 추가)

정책: 모든 이력 보관. 변경 시 새 행을 추가하고, 이력 관리를 위한 추가 컬럼(시작일, 종료일, 현재 여부 플래그)을 사용합니다.

	cust_id	name	age	s_date
초기 원본 테이블 (Target Dimension Table):	1	A	21	2021-09-29
	2	B	31	2021-09-29

동일한 변경 데이터 발생 (2021-10-02):

- ID 1: 이름 'A' → 'AA'로 변경 (Update)
- ID 2: 삭제 (Delete)
- ID 3: 신규 추가 (Insert)

	cust_id	name	age	s_date	e_date
SCD Type 2 적용 후 최종 테이블:	1	A	21	2021-09-29	2021-10-02
	1	AA	21	2021-10-02	9999-12-31
	gray!10 2	B	31	2021-09-29	2021-10-02
	3	C	16	2021-10-02	9999-12-31

Type 2 결과 (모든 이력이 보존됨) 장점: 2021년 9월 30일 시점에서는 ID 1의 이름이 'A'였음을 정확히 추적할 수 있습니다. 'WHERE current = 'T''로 항상 최신 데이터만 조회할 수도 있습니다.

9 고급 기능 및 UDF (Advanced Features & UDFs)

9.1 Delta Lake 관리 기능

Delta Lake는 데이터 레이크의 안정성을 높이기 위해 다음과 같은 관리 명령어를 제공합니다.

- **OPTIMIZE (최적화):** 작은 파일들을 더 큰 파일로 병합(Bin-packing)하여 쿼리 성능을 향상시킵니다. (작은 파일 문제 해결) ‘ZORDER BY’와 함께 사용하여 데이터 레이아웃을 최적화할 수 있습니다.
- **RESTORE (복원):** 실수로 데이터를 잘못 덮어썼을 때, Delta Lake의 트랜잭션 로그(버전 관리)를 사용하여 테이블을 특정 버전이나 타임스탬프로 되돌립니다. (Time Travel)

```
1 RESTORE TABLE my_table TO TIMESTAMP AS OF '2025-10-25 00:00:00'
```

Listing 3: 테이블을 어제 시간으로 복원

- **VACUUM (진공청소):** Time Travel을 위해 유지되던 오래된 버전의 데이터 파일 중, 설정된 보존 기간(기본 7일)이 지난 파일들을 실제로 삭제하여 스토리지 비용을 절감합니다.

VACUUM 주의사항

보존 기간(0일)을 너무 짧게 설정하고 ‘VACUUM’을 실행하면, Time Travel로 복구할 수 있는 데이터가 사라지므로 주의해야 합니다.

- **CLONE (복제):** 테이블을 복제합니다.
 - **Shallow Clone (얕은 복제):** 메타데이터만 복사합니다. (Zero Copy Clone) 원본 데이터 파일을 참조하므로 즉시 생성되며 비용이 거의 들지 않습니다. (테스트용)
 - **Deep Clone (깊은 복제):** 메타데이터와 데이터 파일 전체를 복사합니다. (백업, 마이그레이션용)

9.2 UDF (사용자 정의 함수)

Spark SQL이 제공하는 내장 함수 외에, 사용자가 직접 정의한 복잡한 로직을 실행해야 할 때 UDF를 사용합니다.

- **UDF (User Defined Function):** 가장 일반적인 형태. 행(Row) 단위로 작동합니다. (1:1 행 입력 → 1:1 값 출력)
- **Pandas UDF (Vectorized UDF):** 성능이 훨씬 뛰어난 UDF입니다. 데이터를 행 단위가 아닌, Apache Arrow를 통해 Pandas Series/DataFrame ‘뭉치’(Vector) 단위로 전달받아 처리합니다.
- **UDAF (User Defined Aggregate Function):** 사용자 정의 ‘집계’ 함수. 여러 행을 입력받아 단일 값을 출력합니다. (N:1) (예: ‘SUM’, ‘COUNT’)
- **UDTF (User Defined Table Function):** 사용자 정의 ‘변환’ 함수. 하나의 행을 입력받아 여러 행(테이블)을 출력할 수 있습니다. (1:N) (예: ‘EXPLODE’)

□ UDF vs Pandas UDF: 왜 Pandas UDF가 빠를까?

Spark의 핵심은 JVM(Java/Scala) 위에서 실행되고, PySpark는 Python 인터프리터에서 실행됩니다.

- **일반 Python UDF:** 데이터를 처리할 때마다 JVM ↔ Python 간에 비싼 데이터 직렬화(Serialization)/역직렬화(Deserialization)가 발생합니다. (마치 한 줄 한 줄 통역하는 것과 같습니다.)

- **Pandas UDF:** Apache Arrow라는 메모리 형식을 사용하여 JVM과 Python 간에 데이터를 '뭉치'로 복사 없이 공유합니다. 직렬화 오버헤드가 거의 없어 성능이 매우 뛰어납니다. (마치 책 한 권을 통째로 넘겨주는 것과 같습니다.)

결론: Python UDF를 사용해야 한다면, 성능을 위해 항상 Pandas UDF를 우선적으로 고려해야 합니다.

```
1 from pyspark.sql.functions import pandas_udf, PandasUDFType
2
3 # Pandas UDF 정의데코레이터 ( 사용 )
4 @pandas_udf('double', PandasUDFType.SCALAR)
5 def pandas_plus_one(v):
6     # 입력은 v Pandas Series 객체
7     return v + 1
8
9 df = spark.range(10)
10 df.withColumn('id_transformed', pandas_plus_one("id")).show()
```

Listing 4: Pandas UDF (Vectorized UDF) 예시

10 실습: Autoloader를 이용한 데이터 로딩

10.1 Autoloader란?

Autoloader는 클라우드 스토리지(S3, ADLS 등)에 새로운 파일이 도착했을 때, 이를 자동으로 감지하여 증분(incrementally) 처리해주는 Spark의 스트리밍 소스입니다.

- **작동 방식:** 파일 기반 스트리밍입니다. 별도 설정 없이 파일 도착을 효율적으로 감지합니다.
- **주요 기능:** 스키마 추론(Inference) 및 진화(Evolution)를 강력하게 지원하며, 불량 데이터 처리가 용이합니다.
- **사용법:** ‘.readStream.format("cloudFiles")’를 사용합니다.

10.2 Autoloader 핵심 옵션 및 코드

```

1 (spark.readStream
2   .format("cloudFiles") # Autoloader 사용
3   .option("cloudFiles.format", "json") # 원본파일포맷
4   .option("cloudFiles.schemaLocation", "/path/to/schema/location") # 스키마저장위치
5   .load("/path/to/source/data")
6 .writeStream
7   .option("checkpointLocation", "/path/to/checkpoint/location") # 체크포인트
8   .option("mergeSchema", "true") # 스키마진화허용
9   .table("bronze_table")
10 )

```

Listing 5: Autoloader 기본 사용법

- ‘cloudFiles.format’: 원본 파일 형식(JSON, CSV, Parquet 등)을 지정합니다.
- ‘cloudFiles.schemaLocation’: Autoloader가 추론하거나 변경한 스키마 정보를 저장할 위치입니다. 이 위치를 기준으로 스키마 버전이 관리됩니다.
- ‘mergeSchema’ (Write 옵션): 스키마 진화를 허용합니다. 소스 파일에 새 컬럼이 추가되면, 파이프라인 중단 없이 타겟 테이블(Bronze)의 스키마도 변경됩니다.

10.3 불량 데이터 처리 (Handling Bad Data)

데이터 로딩 시, 스키마가 맞지 않는(예: ‘int’ 컬럼에 문자열) 불량 데이터가 들어오면 파이프라인이 실패할 수 있습니다. Autoloader는 이를 우아하게 처리하는 ‘Rescue’ 기능을 제공합니다.

- ‘rescueddata’ : Autoloader , ‘rescueddata’ . (Parsing) () , ‘NULL’
- ‘cloudFiles.schemaEvolutionMode = ”rescue”’ (옵션): 스키마 힌트와 일치하지 않는 데이터 탑입을 발견했을 때, 해당 데이터를 ‘rescueddata’ .

□ 예제: title

1. 파이프라인은 ‘rescueddata’ .2. ‘WHERE rescueddata IS NOT NULL’
‘CAST’), (Repair) .
- 3.

10.4 스키마 힌트 (Schema Hints)

스키마 추론은 완벽하지 않습니다. (예: ‘id’가 ‘1’만 있으면 ‘int’로 추론하지만, 사실은 ‘string’일 수 있음)
스키마 힌트는 Autoloader에게 특정 컬럼의 데이터 타입을 강제하도록 ‘힌트’를 주는 기능입니다.

```
1 .option("cloudFiles.schemaHints", "timestamp long, signal_strength float")
```

Listing 6: 스키마 힌트 적용

위 예시에서 ‘timestamp’가 문자열로 추론되더라도 ‘long’ 타입으로, ‘signalStrength’ ‘string’ ‘float’ .

10.5 스트리밍 중단 (Stopping Streams)

Jupyter(Databricks) 노트북에서 스트리밍 작업을 시작하면, 명시적으로 중단하지 않는 한 계속 실행됩니다. 모든 활성 스트림을 중단하려면 다음 코드를 사용합니다.

```
1 for s in spark.streams.active:  
2     s.stop()
```

Listing 7: 모든 활성 스트림 중단

11 용어 정리 (Glossary)

용어	원어	쉬운 설명
디자인 패턴	Design Pattern	자주 발생하는 문제에 대한 검증된 해결책(설계도).
람다 아키텍처	Lambda Architecture	배치(Batch)와 스피드(Speed) 레이어를 분리하여 운영하는 아키텍처.
카파 아키텍처	Kappa Architecture	모든 것을 스트리밍으로 처리하는 단일 파이프라인 아키텍처.
CQRS	Command Query Responsibility Segregation	쓰기(Command)와 읽기(Query)의 책임을 완전히 분리하는 패턴.
폴리글로트 퍼시스턴스	Polyglot Persistence	하나의 DB가 아닌, 여러 종류의 DB를 목적에 맞게 조합하여 사용.
컴플라이언스	Compliance	법규(예: GDPR)를 준수하는 것.
삭제권	Right of Erasure	사용자가 본인의 데이터 삭제를 요청할 수 있는 권리.
테이블 파티션	Table Partition	디스크 I/O 감소를 위해 데이터를 물리적 디렉토리로 나누는 것. (책장 정리)
Spark 파티션	Spark Partition	CPU 병렬 처리를 위해 데이터를 논리적 작업 단위로 나누는 것. (작업자 분배)
Z-Ordering	Z-Ordering	여러 컬럼을 기준으로 데이터를 물리적으로 가깝게 정렬하는 최적화 기법.
브로드캐스트 조인	Broadcast Hash Join	작은 테이블을 모든 노드에 복제(방송)하여 셜플 없이 수행하는 가장 빠른 조인.
셔플	Shuffle	'join'이나 'groupBy' 시, 노드 간 데이터 재분배가 일어나는 비싼 네트워크 작업.
CRUD	Create, Read, Update, Delete	데이터의 기본 연산(생성, 읽기, 수정, 삭제).
Upsert	Update + Insert	데이터가 있으면 수정(Update), 없으면 삽입(Insert)하는 작업. 'MERGE'로 구현.
스키마 진화	Schema Evolution	기존 스키마에 새로운 컬럼 추가를 협용하는 것. ('mergeSchema=true')
CDC	Change Data Capture	소스 시스템의 변경(Insert, Update, Delete) 내역을 캡처하는 기술.
SCD	Slowly Changing Dimensions	변경 내역(CDC)을 타겟 차원 테이블에 어떻게 반영할지 정하는 정책(이력 관리).
SCD Type 1	Overwrite	SCD 정책 중 하나. 이력 없이 기존 데이터를 덮어씀.
SCD Type 2	New Row	SCD 정책 중 하나. 변경 시 새 행을 추가하여 모든 이력을 보관함.
Pandas UDF	Pandas UDF (Vectorized)	Python ↔ JVM 간 직렬화 없이 Arrow를 통해 데이터를 '뭉치'로 처리하는 고성능 UDF.
Autoloader	Autoloader	클라우드 스토리지의 신규 파일을 자동으로 감지하여 종분 로딩하는 Spark 기능.
_rescued_data	_rescued_data	Autoloader가 파싱에 실패한 불량 데이터를 격리하는 컬럼.

강 핵심 용어 정리

12 FAQ 및 과제 Q&A (FAQ and Homework Q&A)

Q: 왜 굳이 Delta Lake 같은 복잡한 걸 쓰나요? 그냥 RDBMS(관계형 DB)를 쓰면 안 되나요?

A: 스케일(Scale) 때문입니다. RDBMS는 수십 TB(테라바이트) 수준의 데이터를 처리하고 저장하는 데는 비용과 성능의 한계가 명확합니다. Delta Lake와 같은 레이크하우스 기술은 저렴한 클라우드 스토리지(S3 등)를 기반으로 하면서도, RDBMS의 장점인 ACID 트랜잭션, 업데이트/삭제(CRUD), 안정적인 스키마 관리를 페타바이트(PB)급 대용량 데이터에서도 가능하게 해줍니다.

Q: 테이블 파티션과 Z-Ordering은 언제 같이 쓰나요?

A: 두 기술은 상호 보완적입니다.

- **1순위 (테이블 파티션):** 데이터 조회 시 항상 사용되는 필터 (예: ‘날짜’, ‘국가’). 카디널리티가 낮은(수십 수백 개) 컬럼에 적용하여 스캔할 데이터의 총량을 대폭 줄입니다.
- **2순위 (Z-Ordering):** 파티션으로 거론 데이터 안에서 추가로 자주 사용되는 필터 (예: ‘IP 주소’, ‘사용자ID’). 카디널리티가 높은(수백만 개 이상) 컬럼에 적용하여 파티션 내부의 파일들을 효율적으로 건너뛰게(Skip) 만듭니다.

(예: ‘PARTITIONED BY (date)’, ‘ZORDER BY (user_id)’)

과제 Q&A: ERD 다이어그램 작성 시 모든 테이블에 Primary Key(PK)가 있어야 하나요?

A: 아니요, 필수는 아닙니다. 물론 대부분의 차원 테이블(Dimension)에는 고유한 식별자(PK)가 있습니다. 하지만 트랜잭션 데이터를 담는 사실 테이블(Fact)이나, 두 테이블을 연결하는 브릿지 테이블(예: ‘partsupp’ 테이블)은 두 개 이상의 키를 조합해야만(Composite Key) 고유성이 보장되거나, 혹은 PK가 아예 정의되지 않을 수도 있습니다. TPC-H 데이터셋의 ‘lineitem’ 테이블도 ‘(l_orderkey, l_linenumber)’ .

과제 Q&A: ERD 다이어그램 툴은 아무거나 사용해도 되나요?

A: 네, 그렇습니다. ‘dbdiagram.io’와 같이 무료로 사용할 수 있고, 결과물을 스크린샷으로 찍거나 공개 링크로 공유하여 과제에 첨부할 수 있다면 어떤 툴을 사용해도 좋습니다.

13 한눈에 보기 (Quick Look Summary)

디자인 패턴 (Design Patterns)

- **Lambda**: 배치(Batch) + 스트리밍(Speed) 2개 파이프라인.
- **Kappa**: 스트리밍(Streaming) 1개 파이프라인.
- **Multi-Hop (Medallion)**: 데이터 정제 단계 (Bronze → Silver → Gold).
- **CQRS**: 쓰기(Write)와 읽기(Read)를 분리.
- **Polyglot Persistence**: 여러 종류의 DB를 조합하여 사용.

Spark 파티션 (매우 중요)

- **Table Partition (테이블 파티션)**: 물리적 저장. 디스크 I/O 감소 목적. (예: ‘PARTITIONED BY (date)’)
- **Spark Partition (스파크 파티션)**: 논리적 분배. CPU 병렬 처리 목적. (셔플 단위)

Spark 조인 전략 (Join Strategies)

- **Broadcast Hash Join (빠름)**: 작은 테이블 → 모든 노드에 복사. (셔플 없음)
- **Sort-Merge Join (안정적)**: 큰 테이블 → 셔플 + 정렬 후 병합. (기본값)

데이터 변경 관리 (CDC vs SCD)

- **CDC (변경 데이터 캡처)**: 소스(Source)의 변경 사항(Insert/Update/Delete)을 감지.
- **SCD (느리게 변하는 차원)**: 타겟(Target)에 변경 이력을 어떻게 반영할지에 대한 정책.
- **SCD Type 1 (덮어쓰기)**: 이력 없음. ‘UPDATE’
- **SCD Type 2 (새 행 추가)**: 모든 이력 보관. ‘INSERT’ + (기존 행 만료 처리)

Delta Lake & Autoloader

- **MERGE**: Upsert (Update + Insert)를 원자적으로 실행.
- **OPTIMIZE**: 작은 파일을 병합. (Z-Ordering과 사용)
- **Pandas UDF**: Python UDF 성능 최적화 (Arrow 사용, 직렬화↓)
- **Autoloader (‘cloudFiles’)**: 신규 파일 충분 로딩.
- ‘*r*escuedata’ :

December 10, 2025

■ 강의명: CSCI E-103: 재현 가능한 머신러닝

■ 주차: Lecture 05

■ 교수명: Anindita Mahapatra

Eric Gieseke

■ 목적: Lecture 05의 핵심 개념 학습

Contents

1 개요: 데이터 레이크 강의 핵심 요약	2
2 핵심 용어 정리	3
3 데이터 저장소의 진화: 파일로에서 레이크하우스까지	4
3.1 데이터 파일로 (Data Silos): 고립된 섬	4
3.2 데이터 레이크 (Data Lake): 모든 것을 담는 호수	4
3.3 데이터 스윔프 (Data Swamp): 관리되지 않는 늪	5
3.4 데이터 웨어하우스 (Data Warehouse): 정제된 백화점	5
3.5 비교: 데이터 레이크 (DL) vs. 데이터 웨어하우스 (DW)	5
3.6 데이터 레이크하우스 (Data Lakehouse): 두 세계의 통합	6
4 델타 레이크: 강력한 레이크하우스의 기반 (Delta Lake)	7
4.1 델타 레이크란? (포맷이 아닌 프로토콜)	7
4.2 델타 레이크의 핵심 기능 (The Guarantees)	7
5 데이터 아키텍처 및 원칙	8
5.1 메달리온 아키텍처 (Medallion Architecture)	8
5.2 레이크하우스 6대 가이드 원칙	8
5.3 데이터 메시 vs. 데이터 패브릭	8
6 데이터 파이프라인 구축 및 운영	10
6.1 데이터 수집 도구 (Consolidation Tools)	10
6.2 작업 오케스트레이션 (Job Orchestration)	10
6.3 작업 성능 문제 해결 (Debugging Job Slowness)	10
6.4 모니터링 (Monitoring)	11

7 [실습] 델타 레이크 핵심 기능 (Lab: Delta Lake Features)	12
7.1 Parquet에서 Delta로 변환	12
7.2 동시 작업 (Streaming + Batch)	12
7.3 DML (DELETE, UPDATE)	12
7.4 MERGE (Upsert)	13
7.5 스키마 진화 (Schema Evolution)	13
7.6 타임 트래블 (Time Travel)	13
8 [실습] Databricks 워크플로우 (Lab: Workflows)	15
9 빠르게 훑어보기 (1페이지 요약)	16

1 개요: 데이터 레이크 강의 핵심 요약

▣ 핵심 요약

이 문서는 5주차 '데이터 레이크' 강의의 핵심 내용을 정리합니다.

데이터가 고립되는 **'데이터 사일로'** 문제를 해결하기 위해 등장한 **'데이터 레이크'**의 개념을 배웁니다. 하지만 데이터 레이크가 적절한 관리 없이 방치되면 **'데이터 스웜프(늪)'** 가 되는 위험성을 인지합니다.

이 문제를 해결하고 데이터 웨어하우스의 장점(신뢰성, 성능)을 결합한 **'데이터 레이크하우스'** 가 왜 필요한지, 그리고 그 기반 기술인 **'델타 레이크(Delta Lake)'** 가 어떻게 동작하는지(ACID, 타임 트래블) 학습합니다.

마지막으로, 실제 데이터 파이프라인을 구축하는 **'메달리온 아키텍처'** (Bronze/Silver/Gold) 와 파이프라인의 성능을 저하하는 주요 원인들(Spill, Shuffle, Skew)을 이해합니다.

이 문서는 강의 내용과 실습 코드를 통합하여, 데이터 엔지니어링을 처음 접하는 사람도 쉽게 이해할 수 있도록 구성되었습니다.

2 핵심 용어 정리

데이터 엔지니어링 분야는 유사하지만 미묘하게 다른 용어들이 많아 혼란스러울 수 있습니다. 각 개념의 핵심 차이를 표로 정리했습니다.

용어 (원어)	핵심 설명 (비유)	주요 특징	관계
데이터 사일로 (Data Silo)	부서별로 따로 쓰는 '데이터 섬'. (예: 재무 팀의 Excel, 마케팅 팀의 DB)	- 데이터 고립 및 중복 - 전사적 분석 불가	데이터 레이크가 해결하려는 문제
데이터 레이크 (Data Lake)	모든 데이터를 원본(Raw) 그대로 저장하는 거대한 '저장용 호수'.	- 모든 데이터 유형 저장 - 스키마 온 리드 (Schema on Read) - 저렴한 저장 비용	사일로를 해결. 스웜프가 될 수 있음.
데이터 스웜프 (Data Swamp)	무엇이 어디 있는지 모르는 '데이터 늪'.	- 거버넌스 부재 - 메타데이터 관리 실패 - 데이터 신뢰도 하락	데이터 레이크가 실패한 상태
데이터 웨어하우스 (Data Warehouse)	정제되고 구조화된 데이터만 보관하는 '제품 창고' 또는 '백화점'.	- 정형 데이터만 저장 - 스키마 온 라이트 (Schema on Write) - BI 및 리포트용, 고성능	레이크와 상호 보완적. (비유: 폐라리)
데이터 레이크하우스 (Data Lakehouse)	'호수 (Lake)'의 유연함과 '창고 (Warehouse)'의 신뢰성을 합친 '하이브리드'.	- 데이터 레이크 + 데이터 웨어하우스 - 단일 플랫폼에서 BI와 AI 모두 지원	레이크의 진화형. 델타 레이크가 핵심 기반.
델타 레이크 (Delta Lake)	데이터 레이크를 '신뢰할 수 있게' 만드는 '관리 프로토콜'.	- Parquet 기반 + 트랜잭션 로그 - ACID 트랜잭션, 타임 트래블 지원	레이크하우스를 구현하는 핵심 기술
메달리온 아키텍처 (Medallion Arch.)	데이터를 '동/은/금메달'처럼 3단계로 정제하는 파이프라인 구조.	- Bronze: 원본 (Raw) - Silver: 정제/통합 (Cleaned) - Gold: 집계/활용 (Aggregated)	레이크하우스에서 데이터를 관리하는 방법론
DAG (Directed Acyclic Graph)	'방향이 있고 순환하지 않는 그래프'. 작업의 의존 관계를 표현.	- (A → B), (A → C) → D - 작업 병렬성/의존성 정의	스파크(Spark) 및 워크플로우의 작동 원리
데이터 메시 (Data Mesh)	데이터 관리를 중앙팀이 아닌 '각 현업 부서(도메인)'가 맡는 조직 문화.	- 분산 소유권 (Domain Ownership) - 데이터=제품 (Data as a Product)	레이크하우스와 함께 적용 가능한 조직/문화 전략
데이터 패브릭 (Data Fabric)	여러 곳에 흩어진 데이터를 '하나의 천'처럼 연결하는 기술 아키텍처.	- 하이브리드/멀티 클라우드 통합 - 메타데이터 기반 자동화	데이터 메시보다 기술/아키텍처에 집중화

3 데이터 저장소의 진화: 사일로에서 레이크하우스까지

데이터를 저장하고 활용하는 방식은 비즈니스 요구사항에 따라 진화해왔습니다.

3.1 데이터 사일로 (Data Silos): 고립된 섬

데이터 사일로는 데이터가 조직 내의 여러 부서나 시스템에 고립되어 저장된 상태를 말합니다.

□ 예제:

비유: 고립된 섬

A 회사의 재무팀은 재무 데이터(매출)를 자신들의 Excel 파일에, 마케팅팀은 고객 데이터(클릭 기록)를 별도의 마케팅 솔루션에 저장합니다. 두 팀의 데이터가 연결되지 않아 “어떤 마케팅 활동이 실제 매출로 이어졌는지” 분석하는 것이 거의 불가능합니다. 각 부서가 자신만의 ‘데이터 섬’에 갇혀 있는 것입니다.

데이터 사일로의 주요 문제점:

- 구조적 문제 (Structural): 시스템 자체가 데이터 공유를 고려하지 않고 설계되었습니다.
- 정치적 문제 (Political): 부서가 데이터를 ‘소유물’로 여기고 공유를 거부합니다.
- 성장 문제 (Growth): 조직이 성장하며 도입한 새 시스템이 기존 시스템과 호환되지 않습니다.
- 벤더 종속 (Vendor Lock-in): 특정 솔루션 (SaaS 등)이 데이터 외부 반출을 어렵게 만듭니다.

3.2 데이터 레이크 (Data Lake): 모든 것을 담는 호수

데이터 사일로 문제를 해결하기 위해 등장한 것이 데이터 레이크 (Data Lake)입니다. “모든 종류의 데이터를(정형, 반정형, 비정형) 원본(Raw) 형태 그대로 한곳에 저장하자”는 개념입니다.

- 장점:** 저렴한 비용으로 모든 데이터를 한곳에 모아 사일로를 제거합니다. 데이터 과학자와 ML 엔지니어는 원본 데이터에 자유롭게 접근하여 새로운 가치를 탐색할 수 있습니다.
- 핵심 특징:** 스키마 온 리드 (Schema on Read)

개념 비교: 스키마 온 리드 vs. 스키마 온 라이트

스키마 온 리드 (Schema on Read)	스키마 온 라이트 (Schema on Write)
(데이터 레이크 방식)	(전통적 데이터베이스/웨어하우스 방식)
1. 저장 (Write): 일단 그냥 저장한다. (스키마 없음)	1. 저장 (Write): 엄격한 스키마(테이블 구조) 검사. (불일치 시 저장 실패)
2. 읽기 (Read): 읽는 시점에 스키마(구조)를 정의하여 해석한다.	2. 읽기 (Read): 이미 정의된 스키마대로 빠르게 읽는다.
장점: 빠른 수집, 유연함 (모든 데이터 수용)	장점: 데이터 품질 보장, 빠른 읽기 속도
단점: 읽기 복잡, 데이터 품질 보장 어려움	단점: 수집이 까다로움, 비정형 데이터 저장 불가

3.3 데이터 스웜프 (Data Swamp): 관리되지 않는 늪

데이터 레이크는 강력하지만, 치명적인 함정이 있습니다. '스키마 온 리드'의 유연함은 곧 '관리의 부재'를 의미할 수 있습니다.

데이터가 계속 쌓이지만,

- 어떤 데이터인지 설명하는 메타데이터(Metadata)가 없거나,
- 데이터의 품질을 관리하는 거버넌스(Governance)가 없다면,

데이터 레이크는 아무도 무엇이 어디 있는지 알 수 없는 데이터 스웜프(Data Swamp, 늪)가 되어 버립니다.

□ 예제:

비유: 창고 vs. 폐기물 처리장

잘 관리된 데이터 레이크는 모든 물건에 '라벨'이 붙어 있고 '위치'가 기록된 거대한 창고(아마존 물류센터)와 같습니다. 하지만 데이터 스웜프는 온갖 종류의 쓰레기를 한곳에 버리기만 한 폐기물 처리장과 같습니다. 무언가 유용한 것이 있을지도 모르지만, 찾을 수가 없습니다.

주의사항

진자의 비유 (The Pendulum)

데이터 관리는 양극단을 오가는 진자와 같습니다.

- 데이터 사일로 (Silos): 너무 엄격하게 분리되어 활용이 불가능.
- 데이터 스웜프 (Swamps): 너무 자유롭게 방치되어 활용이 불가능.

우리의 목표는 이 진자의 중간 지점, 즉 관리되는 데이터 레이크(Governed Data Lake)를 찾는 것입니다.

3.4 데이터 웨어하우스 (Data Warehouse): 정제된 백화점

데이터 레이크와 자주 비교되는 개념이 데이터 웨어하우스(Data Warehouse, DW)입니다. DW는 비즈니스 인텔리전스(BI) 및 리포팅을 위해 '정제되고', '구조화된' 데이터만 저장하는 시스템입니다.

- 핵심 특징: 스키마 온 라이트 (Schema on Write)
- 주요 사용자: 비즈니스 분석가 (Business Analysts)
- 장점: 데이터가 신뢰할 수 있고, 쿼리(조회) 속도가 매우 빠릅니다.
- 단점: 비싸고, 비정형 데이터를 처리할 수 없으며, 데이터 과학(ML) 활용에 제한적입니다.

3.5 비교: 데이터 레이크 (DL) vs. 데이터 웨어하우스 (DW)

□ 예제:

비유: 폐라리 vs. 트랙터 트레일러

- 데이터 웨어하우스 (폐라리): 매우 빠르고 멋지지만(고성능 쿼리), 2명밖에 못 타고(정형 데이터), 짐을 실을 수 없습니다(유연성 낮음).
- 데이터 레이크 (트랙터 트레일러): 폐라리 만큼 빠르진 않지만, 모든 종류의 짐(모든 데이터)을

원하는 만큼(대용량) 실어 나를 수 있습니다.

특징	데이터 레이크 (Data Lake)	데이터 웨어하우스 (Data Warehouse)
데이터 종류	모든 데이터 (정형, 반정형, 비정형)	정형 데이터 (Structured)
데이터 상태	원본 (Raw), 정제된 데이터 모두	정제되고 변환된 데이터 (Processed)
스키마	Schema on Read (읽을 때 정의)	Schema on Write (저장할 때 정의)
프로세스	ELT (Extract, Load → Transform)	ETL (Extract, Transform → Load)
주요 사용자	데이터 과학자, ML 엔지니어	비즈니스 분석가, SQL 사용자
주요 용도	AI / ML 모델링, 데이터 탐색	BI 리포트, 대시보드
비용	저렴 (스토리지/컴퓨트 분리)	고비용 (스토리지/컴퓨트 결합)
형식	오픈 포맷 (Open Formats)	독점 포맷 (Proprietary)

3.6 데이터 레이크하우스 (Data Lakehouse): 두 세계의 통합

과거에는 BI를 위해 DW를, AI/ML을 위해 DL을 따로 구축했습니다. 하지만 이로 인해 데이터가 다시 사일로화되고(DW/DL 사일로), 비용이 이중으로 드는 문제가 발생했습니다.

데이터 레이크하우스(Data Lakehouse)는 이 두 시스템을 하나로 통합하려는 시도입니다. 즉, 데이터 레이크의 저렴한 비용과 유연성 위에 데이터 웨어하우스의 신뢰성, 거버넌스, 성능을 구현한 것입니다.

이를 가능하게 하는 핵심 기술이 바로 델타 레이크(Delta Lake)입니다.

4 델타 레이크: 강력한 레이크하우스의 기반 (Delta Lake)

4.1 델타 레이크란? (포맷이 아닌 프로토콜)

주의사항

오해 바로 잡기: 델타 레이크는 파일 포맷이 아닙니다.

델타 레이크(Delta Lake)는 프로토콜(Protocol) 또는 관리 계층입니다. 여러분이 저장하는 실제 데이터 파일은 여전히 효율적인 Parquet 포맷입니다.

델타 레이크는 이 Parquet 파일들 위에 `_delta_log`라는 트랜잭션 로그 폴더를 추가하여, Parquet 파일만으로는 불가능했던 강력한 기능들을 제공합니다.

전통적인 데이터 레이크(Hadoop, S3)의 파일(Parquet, CSV)은 한번 쓰면 수정이 불가능(Immutable)한 경우가 많아 다음과 같은 치명적인 문제들이 있었습니다.

- 데이터 일부만 수정(UPDATE)하거나 삭제(DELETE)하기가 극도로 어렵습니다.
- 작업이 중간에 실패하면 데이터가 오염됩니다(예: 중복 데이터).
- 여러 사용자가 동시에 데이터를 읽고 쓸 때 일관성이 깨집니다.

델타 레이크는 이 모든 문제를 해결하여 데이터 레이크를 신뢰할 수 있는 저장소로 만듭니다. (대안 기술: Apache Iceberg, Apache Hudi)

4.2 델타 레이크의 핵심 기능 (The Guarantees)

- ACID 트랜잭션 (ACID Transactions):** 전통적인 데이터베이스처럼 원자성(Atomicity), 일관성(Consistency), 고립성(Isolation), 지속성(Durability)을 보장합니다. 작업이 중간에 실패해도 데이터가 오염되지 않으며, 여러 사용자가 동시에 접근해도 데이터가 깨지지 않습니다.
- 스키마 관리 (Schema Evolution & Enforcement):** 데이터를 쓸 때 스키마가 일치하는지 검사(Enforcement)할 수 있으며, 의도적으로 스키마(컬럼)를 변경(Evolution)하는 것도 허용합니다.
- 타임 트래블 (Time Travel):** 모든 변경 이력(`_delta_log`)을 저장하므로, “어제 날짜의 데이터” 또는 “버전 5의 데이터”처럼 과거의 특정 시점으로 데이터를 되돌려 조회할 수 있습니다. (예: ‘VERSION AS OF 5’)
- DML 지원 (UPDATE, DELETE, MERGE):** 데이터 레이크의 파일에 직접 SQL의 ‘UPDATE’, ‘DELETE’, ‘MERGE’ (Upsert) 명령을 실행할 수 있습니다. (예: GDPR로 인한 사용자 데이터 삭제)
- 성능 최적화 (Optimization):** 데이터 스키핑(Data Skipping), 통계정보 기반 파일 필터링, Z-Ordering(다차원 정렬) 등을 통해 쿼리 성능을 향상시킵니다.

5 데이터 아키텍처 및 원칙

5.1 메달리온 아키텍처 (Medallion Architecture)

레이크하우스에서 데이터를 효과적으로 정제하고 관리하기 위해 가장 널리 쓰이는 패턴이 메달리온 아키텍처입니다. 데이터의 품질 수준에 따라 3개의 레이어(레이블)로 구분합니다.

□ 예제:

비유: 광물 제련소

- Bronze (동메달):** 광산에서 막 캐낸 '원본 광석' (Raw Data). 어떤 가공도 하지 않고 그대로 쌓아둡니다.
- Silver (은메달):** 원본 광석에서 불순물을 제거하고(Null, 중복 제거) 필요한 것끼리 합친 '제련된 금속' (Cleaned, Filtered).
- Gold (금메달):** 제련된 금속을 특정 목적에 맞게 가공한 '최종 완제품' (Aggregated, Business-ready).

레이어	Bronze	Silver	Gold
데이터 상태	원본 (Raw)	정제됨 (Cleaned)	집계됨 (Aggregated)
주요 작업	- 모든 소스 데이터 수집- (예: JSON, CSV, Parquet)	- Null/중복 데이터 제거- 데이터 유형(Type) 동일- 여러 소스 조인(Join)- (이때부터 Delta 포맷)	- BI/리포트를 위한 집계- (예: 부서별 월간 매출)- AI/ML용 피쳐(Feature) 생성
데이터 구조	소스 시스템과 동일	3정규화(3NF) 또는 유사	비정규화(Denormalized), 스타 스키마
주요 사용자	데이터 엔지니어	데이터 엔지니어, 데이터 과학자	비즈니스 분석가, 데이터 과학자
업데이트 빈도	실시간 또는 배치	배치 (Bronze → Silver)	배치 (Silver → Gold)

5.2 레이크하우스 6대 가이드 원칙

성공적인 레이크하우스를 구축하기 위한 6가지 핵심 원칙입니다.

- 데이터를 제품(Data-as-Products)으로 취급하라:** (메달리온 아키텍처) 데이터를 단순히 쌓아두는 것이 아니라, 신뢰할 수 있는 '제품'으로 가공하여 제공해야 합니다.
- 데이터 파일로를 제거하고 데이터 이동을 최소화하라:** 데이터를 한곳(레이크)에 보관하고 여러 시스템이 직접 접근하게 하여, 데이터를 복제/이동하면서 발생하는 비효율과 데이터 불일치(Stale data) 문제를 막아야 합니다.
- 셀프 서비스(Self-Service) 경험으로 가치 창출을 민주화하라:** 적절한 거버넌스 하에 현업 사용자들이 직접 데이터에 접근하고 분석할 수 있는 환경을 제공해야 합니다.
- 전사적인 데이터 거버넌스 전략을 채택하라:** 데이터 접근 제어(Access Control), 품질 관리, 카탈로그(Unity Catalog) 등 중앙화된 거버넌스 전략이 필수입니다.
- 개방형 인터페이스와 포맷(Open Formats)을 사용하라:** 특정 벤더에 종속되지 않는 Parquet, Delta Lake 같은 오픈 포맷을 사용하여 유연성을 확보해야 합니다.
- 확장성, 성능, 비용을 고려하여 구축하라:** 데이터와 사용자가 증가할 것을 대비하여 확장 가능하게 설계하고, 성능과 비용 간의 균형을 맞춰야 합니다.

5.3 데이터 메시 vs. 데이터 패브릭

레이크하우스와 함께 자주 언급되는 두 가지 상위 개념입니다.

주의사항

핵심 차이: 조직 vs. 기술

- 데이터 메시 (Data Mesh):** 조직 문화에 가깝습니다. ”중앙 데이터팀이 모든 것을 관리하는 것은 비효율적이니, 데이터 관리를 각 협업 부서(도메인)에 맡기자”는 분산형 조직/프로세스 철학입니다.
- 데이터 패브릭 (Data Fabric):** 기술 아키텍처에 가깝습니다. ”회사의 데이터가 여러 클라우드와 온프레미스(사내 서버)에 흩어져 있으니, 이를 기술적으로 연결하여 마치 하나의 거대한 ‘천 (Fabric)’처럼 보이게 만들자”는 통합 아키텍처입니다.

관점	데이터 메시 (Data Mesh)	데이터 패브릭 (Data Fabric)
핵심 아이디어	분산화 (Decentralization)	통합 및 연결 (Integration)
주요 초점	조직, 사람, 프로세스	기술, 아키텍처, 자동화
데이터 소유권	각 도메인 팀 (협업 부서)	중앙팀 (IT) 또는 위임
거버넌스	연합 거버넌스 (Federated)	중앙 집중식 거버넌스 (Centralized)
데이터 취급	데이터 = 제품 (Data as a Product)	데이터 = 접근 가능한 자산
적용 대상	조직의 복잡성이 높은 대기업	기술/환경의 복잡성이 높은 기업 (하이브리드/멀티 클라우드)

6 데이터 파이프라인 구축 및 운영

6.1 데이터 수집 도구 (Consolidation Tools)

데이터 레이크로 데이터를 가져오는(수집/적재) 방법들입니다.

- **AutoLoader (오토로더):** 지속적, 증분(Incremental) 수집에 사용됩니다. 클라우드 스토리지(S3, ADLS)의 특정 폴더를 모니터링하다가 새로운 파일이 도착하면 자동으로 감지하여 레이크로 적재합니다. 스트리밍 방식이며 ‘cloudFiles’ 포맷을 사용합니다.
- **COPY INTO (카피 인트):** 대용량 일괄(Bulk Batch) 수집에 사용되는 SQL 명령어입니다. 이미 적재된 파일은 건너뛰는 멱등성(Idempotent)을 보장하여, 명령을 여러 번 실행해도 데이터 중복이 발생하지 않습니다.
- **DLT (Delta Live Tables):** 차세대 ETL 도구입니다. 선언적(Declarative)으로 파이프라인을 정의하면, 복잡한 처리와 데이터 품질 검사(Quality Constraints)를 자동화해줍니다.
- **Local File Upload (로컬 파일 업로드):** Databricks UI를 통해 로컬 PC의 작은 파일**^(예: 테스트용 CSV)을 업로드하는 기능입니다. (최대 10개 파일, 총 2GB 제한)

6.2 작업 오케스트레이션 (Job Orchestration)

단일 작업이 아닌, 여러 작업(Task)이 연결된 복잡한 파이프라인을 워크플로우(Workflow)라고 부릅니다. 이 워크플로우를 관리하고 스케줄링하는 것을 작업 오케스트레이션(Job Orchestration)이라고 합니다.

- **의존성 (Dependencies):** 작업 간의 의존 관계를 DAG (Directed Acyclic Graph)로 정의합니다.
 - **병렬(Parallel) 수행:** 서로 의존성이 없는 작업들 (예: ‘고객 데이터 수집’과 ‘주문 데이터 수집’)
 - **순차(Sequential) 수행:** 실행 작업이 끝나야만 시작할 수 있는 작업 (예: ‘고객 클릭 수집’ → ‘클릭 세션화’)
- **주요 기능:**
 - **스케줄링 (Scheduling):** 특정 시간(예: 매일 오전 3시)에 실행되도록 예약(Cron 표현식 사용).
 - **트리거 (Triggers):** 특정 이벤트(예: 새 파일 도착)에 의해 실행.
 - **재시도/타임아웃 (Retry/Timeout):** 작업 실패 시 자동 재시도, 또는 일정 시간 초과 시 강제 종료.
 - **복구 및 재실행 (Repair and Run):** 파이프라인의 10개 작업 중 8번째 작업만 실패했을 때, 17번 작업을 재사용하고 8번부터 다시 실행하여 시간과 비용을 절약합니다.

6.3 작업 성능 문제 해결 (Debugging Job Slowness)

Spark 작업이 유난히 느리게 실행될 때 확인해야 할 4가지 주요 원인 (The 4 S's)입니다.

□ 예제:

비유로 이해하는 Spark 성능 저하 원인

- **스필 (Spill):** (증상) 메모리(RAM) 가 부족하여 데이터를 디스크(Disk)에 임시로 썼다가 다시 읽어오는 현상. (비유) 요리하는데 조리대(RAM) 가 너무 좁아서 재료를 바닥(Disk)에 내려놓았다가 다시 집어 드는 것. 매우 느리고 비효율적입니다. (해결) 더 많은 메모리를 가진 노드를 사용

하거나, T-shirt 사이즈(Serverless)를 늘립니다.

- 2. 셔플 (Shuffle): (증상) 정렬(Sort)이나 조인(Join) 시 여러 노드(컴퓨터) 간에 대규모 데이터 교환이 발생하는 현상. (비유) 팀 프로젝트를 하는데, 필요한 자료가 여러 팀원에게 흩어져 있어 (Nodes) 서로 카톡으로 자료를 주고받느라(Network) 시간을 다 보내는 것. (해결) 데이터 파티셔닝 전략을 최적화하여 노드 간 데이터 이동을 최소화합니다.
- 3. 스큐 / 스트래글러 (Skew / Stragglers): (증상) 데이터가 불균등하게 분배(Skew)되어 특정 노드(Straggler) 하나에만 일이 몰리는 현상. (비유) 팀 프로젝트 5명 중 1명(Straggler)에게만 모든 일이 몰리고, 나머지 4명은 일찍 끝나서 놀고 있는 것. 전체 프로젝트는 그 1명이 끝나야 끝납니다. (해결) 파티션 키를 변경하여 데이터가 고르게 분배되도록 합니다. (예: '국가' 대신 '국가+도시')
- 4. 작은 파일 문제 (Small Files): (증상) 수백만 개의 아주 작은 파일들을 처리할 때, 실제 데이터 처리 시간보다 파일을 '열고/닫는' I/O 오버헤드가 더 커지는 현상. (비유) 책 1권(1GB)을 읽는데, 1권짜리 책이 아니라 1페이지짜리 100만 개의 파일로 쪼개져 있어서, 파일을 100만 번 '열고-읽고-닫는' 것. (해결) 'OPTIMIZE' 명령 등으로 작은 파일들을 적절한 크기의 큰 파일로 병합(Compaction)합니다.

6.4 모니터링 (Monitoring)

- 전통적 방식 (Ganglia Metrics): 클러스터의 CPU, 메모리, 네트워크 사용량을 시각적으로 모니터링합니다. (예: Spill이 발생하면 메모리 사용량이 100%에 육박하고 디스크 I/O가 급증합니다.)
- 서비스 (Serverless): Ganglia 같은 세부적인 인프라 모니터링 대신, T-shirt Sizing (예: Small, Medium, Large)을 통해 워크로드에 맞는 리소스를 선택합니다. 관리는 단순화되지만, 세밀한 메모리 튜닝은 어려울 수 있습니다.

7 [실습] 델타 레이크 핵심 기능 (Lab: Delta Lake Features)

7.1 Parquet에서 Delta로 변환

기존 Parquet 데이터를 Delta Lake 형식으로 변환하는 것은 매우 간단합니다. ‘CREATE TABLE ... USING delta AS ...’ 구문을 사용합니다.

```

1 -- Lending Club 데이터셋(Parquet)을 읽어와
2 -- 'loans_by_state_delta'라는 'Delta Lake' 테이블로 생성
3 CREATE TABLE IF NOT EXISTS loans_by_state_delta
4 USING delta
5 LOCATION '/path/to/delta/table'
6 AS SELECT * FROM parquet_table_view;

```

Listing 1: Parquet 파일을 Delta Lake 테이블로 변환

테이블이 Delta 형식인지 확인하려면 ‘DESCRIBE DETAIL’을 사용합니다.

```
1 DESCRIBE DETAIL loans_by_state_delta;
```

Listing 2: 테이블 상세 정보 확인 (Delta 확인)

7.2 동시 작업 (Streaming + Batch)

Delta Lake의 ACID 트랜잭션 덕분에, 한쪽에서는 데이터를 스트리밍으로 읽는(‘readStream’) 동시에 다른 쪽에서는 배치(Batch)로 데이터를 삽입(‘INSERT’) 할 수 있습니다.

```

1 # streaming_query = (
2 #     spark.readStream
3 #         .format("delta")
4 #         .load("/path/to/delta/table")
5 #         .writeStream
6 #         ...
7 #         .start()
8 # )

```

Listing 3: Delta 테이블에서 스트리밍 읽기 시작 (가상 코드)

```

1 -- Iowa (IA) 주에 건의 450 대출데이터를 번 6 반복삽입
2 INSERT INTO loans_by_state_delta VALUES ('IA', 450);
3 INSERT INTO loans_by_state_delta VALUES ('IA', 450);
4 ... (6 times)

```

Listing 4: 동시에 배치 데이터 삽입 (Iowa 데이터 추가)

스트리밍 쿼리는 중단 없이 이 새로운 삽입을 감지하여 처리합니다.

7.3 DML (DELETE, UPDATE)

전통적인 Parquet 파일에서는 불가능했던 행(Row) 단위 데이터 삭제가 가능합니다.

```

1 -- Iowa (IA) 주의 모든 데이터를 삭제
2 DELETE FROM loans_by_state_delta WHERE state = 'IA';

```

Listing 5: Delta 테이블에서 특정 데이터 삭제

이 명령은 일반 Parquet 테이블에서는 실패하지만, Delta Lake에서는 성공합니다.

7.4 MERGE (Upsert)

‘MERGE’는 ‘Upsert’ (Update + Insert) 작업을 원자적으로 수행합니다. 새로운 데이터(Source)를 기존 테이블(Target)과 비교하여,

- 조건이 일치하면 (‘WHEN MATCHED’) → **UPDATE**
- 조건이 일치하지 않으면 (‘WHEN NOT MATCHED’) → **INSERT**

```

1 -- 'new_updates_table' (Source)의 데이터를
2 -- 'loans_by_state_delta' (Target)에 병합
3 MERGE INTO loans_by_state_delta AS target
4 USING new_updates_table AS source
5 ON target.state = source.state -- state 컬럼을 기준으로 비교
6
7 WHEN MATCHED THEN
8   -- 주(state)가 이미 존재하면 count 업데이트
9   UPDATE SET target.count = source.new_count
10
11 WHEN NOT MATCHED THEN
12   -- 주(state)가 존재하지 않으면 새로 삽입
13   INSERT (state, count) VALUES (source.state, source.new_count);

```

Listing 6: MERGE를 사용한 Upsert 작업

7.5 스키마 진화 (Schema Evolution)

기본적으로 Delta Lake는 기존 테이블 스키마와 다른 데이터가 들어오면 작업을 실패시킵니다 (Schema Enforcement).

주의사항

상황: 기존 테이블은 ‘(state, count)’ 2개 컬럼인데, ‘(state, count, amount)’ 3개 컬럼의 새 데이터를 추가(append) 하려고 합니다.

기본 동작 (실패): ‘spark.write.format(“delta”).mode(“append”).save(...)' → 오류 발생! (Schema mismatch)

해결 (스키마 진화 옵션): 데이터프레임 쓰기 옵션에 ‘option(“mergeSchema”, “true”)’를 추가하면, Delta Lake가 기존 스키마에 ‘amount’ 컬럼을 자동으로 추가하여 작업을 성공시킵니다.

7.6 타임 트래블 (Time Travel)

Delta Lake는 모든 변경 이력(로그)을 보관합니다. ‘DESCRIBE HISTORY’ 명령으로 이력을 볼 수 있습니다.

```
1 DESCRIBE HISTORY loans_by_state_delta;
```

Listing 7: 테이블 변경 이력 조회

이 버전을 사용하여 과거 데이터를 조회할 수 있습니다.

```
1 -- 가장처음버전 ( 0 )의 데이터조회 ( Iowa 데이터가없던시점 )
2 SELECT * FROM loans_by_state_delta VERSION AS OF 0;
3
4 -- 가 MERGE 완료된시점버전 ( 9 )의 데이터조회
5 SELECT * FROM loans_by_state_delta VERSION AS OF 9;
```

Listing 8: 특정 버전(Version)으로 과거 데이터 조회

버전 번호 대신 타임스탬프(시간)를 사용할 수도 있습니다.

```
1 -- 특정날짜시간 / 기준의데이터조회
2 SELECT * FROM loans_by_state_delta
3 TIMESTAMP AS OF '2025-10-26 03:00:00';
```

Listing 9: 특정 시간(Timestamp)으로 과거 데이터 조회

8 [실습] Databricks 워크플로우 (Lab: Workflows)

Databricks Jobs UI를 사용하여 여러 노트북을 연결하는 파이프라인(워크플로우)을 만드는 방법입니다.

1. **작업(Job) 생성:** Databricks UI의 'Jobs' 탭에서 새 작업을 생성합니다.
2. **태스크(Task) 추가:** 작업 내부에 여러 태스크를 추가합니다. 각 태스크는 노트북, SQL 스크립트, Python 파일 등이 될 수 있습니다.
3. **의존성(Dependency) 설정:** 태스크 간의 실행 순서를 정의합니다.
 - Task B가 Task A에 의존하도록 설정하면, Task A가 성공적으로 완료되어야만 Task B가 실행됩니다.
 - 의존성을 설정하지 않으면 두 태스크는 병렬로 실행됩니다.
4. **파라미터(Parameter) 전달:** 워크플로우 실행 시 동적으로 값을 전달할 수 있습니다.
 - **작업 파라미터 (Job Parameters):** 워크플로우 전체에서 사용되는 전역 변수.
 - **태스크 파라미터 (Task Parameters):** 특정 태스크에만 전달되는 지역 변수.

노트북 내에서 이 파라미터를 받으려면 'dbutils.widgets.get()'을 사용합니다.

```

1 # 'param1'이라는 이름으로 전달된 태스크 파라미터 값을 가져옴
2 param_value = dbutils.widgets.get("param1")
3
4 # 'job_param1'이라는 이름으로 전달된 작업 파라미터 값을 가져옴
5 job_param_value = dbutils.widgets.get("job_param1")
6
7 print(f"태스크 파라미터 값 : {param_value}")
8 print(f"작업 파라미터 값 : {job_param_value}")

```

Listing 10: 노트북에서 워크플로우 파라미터 받기

5. **스케줄링:** 'Cron' 구문을 사용하여 "매주 월요일 오전 9시"처럼 주기적으로 작업을 실행하도록 예약할 수 있습니다.

9 빠르게 훑어보기 (1페이지 요약)

데이터 저장소의 진화

1. 데이터 사일로 (Silo):

고립된 '데이터 섬'. 부서 간 데이터 공유 불가. (문제) ↓

2. 데이터 레이크 (Lake):

모든 데이터를 원본(Raw) 그대로 한곳에 저장하는 '호수'. (Schema on Read) (해결책) ↓

3. 데이터 스윔프 (Swamp):

관리가 안 되어 아무도 못 쓰는 '데이터 늪'. (거버넌스 실패) (위험)

웨어하우스 vs. 레이크하우스

4. 데이터 웨어하우스 (Warehouse):

정제된 데이터만 저장하는 '백화점'. (Schema on Write) BI에 빠르지만 비싸고 비유연. ↓

5. 데이터 레이크하우스 (Lakehouse):

레이크(유연성/저비용)와 웨어하우스(신뢰성/성능)의 장점을 합친 통합 플랫폼.

레이크하우스 핵심 기술 및 방법론

Delta Lake (델타 레이크):

Parquet 파일 위에서 ACID 트랜잭션, 타임 트래블, MERGE/UPDATE/DELETE를 가능하게 하는 '프로토콜'. 레이크를 신뢰할 수 있게 만듦.

Medallion Architecture (메달리온 아키텍처):

데이터를 3단계로 정제하는 파이프라인.

- **Bronze:** 원본 (Raw)
- **Silver:** 정제 (Cleaned)
- **Gold:** 집계 (Aggregated)

파이프라인 성능 저하 4대 원인

- **Spill (스필):** 메모리 부족 → 디스크 사용 (느림)
- **Shuffle (셔플):** 노드 간 불필요한 데이터 교환 (네트워크 병목)
- **Skew (스큐):** 데이터 불균형 → 특정 노드만 바쁨 (작업 지연)
- **Small Files (작은 파일):** 너무 많은 작은 파일 → I/O 오버헤드

December 10, 2025

■ 강의명: CSCI E-103: 재현 가능한 머신러닝

■ 주차: Lecture 06

■ 교수명: Anindita Mahapatra

Eric Gieseke

■ 목적: Lecture 06의 핵심 개념 학습

Contents

1	강의 개요	2
1.1	이 강의의 주요 질문 (Key Questions)	2
2	핵심 용어 정리	3
3	핵심 개념 1: 데이터 저장소의 진화	4
3.1	데이터 웨어하우스 (Data Warehouse) vs. 데이터 레이크 (Data Lake)	4
3.2	아키텍처의 한계와 레이크하우스 (Lakehouse)의 등장	4
3.3	데이터 저장소 기술의 역사적 흐름	5
4	핵심 개념 2: 비즈니스 인텔리전스 (BI)란?	6
4.1	BI의 정의와 목적	6
4.2	BI vs. BA: 무엇이 다른가?	6
4.3	BI 프로세스 (5단계)	7
5	핵심 개념 3: BI 페르소나와 데이터 모델링	8
5.1	BI 분석가 (BI Analyst) 페르소나	8
5.2	BI를 위한 데이터 모델링 (Data Modeling for BI)	8
6	Databricks를 활용한 BI 및 데이터 웨어하우징	9
6.1	Databricks 데이터 인텔리전스 플랫폼	9
6.2	Databricks SQL (DBSQL)의 주요 BI 기능	9
6.3	SQL의 AI 기능 (AI Functions in SQL)	10
6.3.1	1. ai_query(): 외부 LLM 호출	10
6.3.2	2. 내장 AI 함수 (Built-in AI Functions)	10
7	실습: Lakeview 대시보드와 Genie	12

7.1	Lakeview 대시보드 (Lakeview Dashboards)	12
7.2	게시 (Publish) 및 공유 (Share)	12
7.3	Databricks Genie: 대화형 AI 분석	12
8	학습 점검 체크리스트	14
9	FAQ (자주 묻는 질문)	14
10	빠르게 훑어보기 (1-Page Summary)	16

1 강의 개요

본 문서는 비즈니스 인텔리전스(BI) 분석 및 데이터 시각화의 핵심 개념을 다룹니다. 데이터 웨어하우스와 데이터 레이크의 전통적인 차이점에서 시작하여, 두 아키텍처의 장점을 결합한 레이크하우스(Lakehouse)의 필요성과 구조를 중점적으로 설명합니다.

또한 BI와 비즈니스 분석(BA)의 차이점을 명확히 하고, BI 분석가(BI Analyst)라는 핵심 페르소나와 이들의 주요 기술(SQL) 및 데이터 모델링(Star Schema 등) 방법을 배웁니다.

마지막으로 Databricks 플랫폼을 활용한 최신 BI 기능들, 특히 **Databricks SQL**, **Lakeview 대시보드**, 그리고 **Genie**와 같은 자연어 기반 AI 분석 도구의 작동 원리와 활용법을 실습 예제와 함께 살펴봅니다.

1.1 이 강의의 주요 질문 (Key Questions)

이 문서를 학습한 후, 다음 질문들에 답할 수 있어야 합니다.

- 비즈니스 인텔리전스(BI)란 무엇이며, 비즈니스 분석(BA) 및 AI와 어떻게 다른가?
- 데이터 웨어하우스(Data Warehouse)와 데이터 레이크(Data Lake)의 근본적인 차이점은 무엇인가?
- 왜 레이크하우스(Lakehouse)라는 새로운 아키텍처가 필요하게 되었는가?
- BI 데이터를 소비하는 주요 사용자 페르소나는 누구이며, 그들의 핵심 기술은 무엇인가? (힌트: SQL)
- BI 성능을 측정하는 핵심 성과지표(KPI)인 동시성(Concurrency)과 지연 시간(Latency)은 무엇을 의미하는가?
- Databricks SQL 환경에서 AI 함수 (예: ai_query)를 어떻게 활용할 수 있는가?
- Genie가 일반 챗봇(ChatGPT 등)과 달리 ”환각(Hallucination)”을 일으키지 않는 이유는 무엇인가?

2 핵심 용어 정리

본격적인 학습에 앞서, 이번 강의에서 자주 등장하는 핵심 용어들을 정리합니다.

Table 1: BI 분석 및 데이터 저장소 핵심 용어

용어 (Term)	쉬운 설명	원어 (English)	비고
BI	기업이 더 나은 의사결정을 하도록 데이터를 수집, 분석, 시각화하는 기술. (과거 현재 ”무엇”이 일어났는지)	Business Intelligence	리포트, 대시보드
BA	과거 데이터를 사용해 현재를 설명하고 미래를 예측하는 분석. (“왜” 일어났는지, ”무엇”이 일어날지)	Business Analytics	통계, 예측 모델링
데이터 웨어하우스	정형화된(구조화된) 데이터만 저장하는, 빠르고 비싼 ‘데이터 도서관’. 스키마가 미리 정해짐.	Data Warehouse (DW)	Schema-on-Write
데이터 레이크	모든 종류(정형, 비정형)의 데이터를 원본 그대로 저장하는, 저렴하고 거대한 ‘데이터 차고’.	Data Lake (DL)	Schema-on-Read
레이크하우스	데이터 레이크의 저렴한 저장소 위에 데이터 웨어하우스의 성능/안정성(ACID)을 구현한 통합 아키텍처.	Lakehouse	DW + DL
메달리온 아키텍처	데이터를 3단계(Bronze: 원본, Silver: 정제, Gold: 집계/BI-용)로 나누어 관리하는 파이프라인 구조.	Medallion Architecture	Bronze, Silver, Gold
데이터 사일로	데이터가 부서별, 시스템별로 고립되어 연결되지 않은 상태.	Data Silo	
데이터 스왑프	데이터 레이크가 거버넌스(관리) 없이 방치되어 쓸모없게 된 상태. ’데이터 늪’.	Data Swamp	
데이터 연합	데이터를 물리적으로 이동(ETL)하지 않고, 원격 소스에서 직접 쿼리(읽기)하는 기술.	Data Federation	소유권(Ownership)이 없음.
뷰 (View)	쿼리 자체를 가상 테이블처럼 저장한 것. 호출 시점에 쿼리가 실행됨. (물리적 저장 X)	View	
구체화된 뷰	쿼리 결과를 물리적으로 미리 계산하여 저장해 둔 테이블. (성능 향상 목적)	Materialized View (MV)	
동시성	시스템이 동시에 몇 개의 쿼리(작업)를 처리할 수 있는지 나타내는 KPI.	Concurrency	
지연 시간	쿼리를 요청한 시점부터 결과가 반환될 때까지 걸리는 시간(딜레이).	Latency	

3 핵심 개념 1: 데이터 저장소의 진화

BI를 이해하기 위해서는 먼저 BI가 사용하는 데이터가 어디에, 어떻게 저장되는지 알아야 합니다. 데이터 저장소는 크게 '데이터 웨어하우스'와 '데이터 레이크'로 나뉩니다.

3.1 데이터 웨어하우스 (Data Warehouse) vs. 데이터 레이크 (Data Lake)

두 개념은 데이터를 저장하는 목적과 방식에서 근본적인 차이가 있습니다.

비유로 이해하기: 도서관 vs. 차고

- 데이터 웨어하우스(DW)는 '잘 정리된 도서관'입니다. 도서관에는 오직 '책'(정형 데이터)만 받습니다. 책을 받으면 사서가 즉시 분류하고(Schema-on-Write), 정해진 책장(테이블)에 꽂습니다. 덕분에 특정 책을 찾을 때(BI 쿼리) 매우 빠르고 정확합니다. 하지만 비디오페이지나 사진(비정형 데이터)은 보관할 수 없고, 도서관을 짓고 유지하는데 비용이 많이 듭니다.
- 데이터 레이크(DL)는 '모든 것을 쌓아두는 차고'입니다. 차고에는 책, 사진, 비디오, 고장 난 자전거 등 모든 것(정형/비정형 데이터)을 원본 그대로 던져 넣을 수 있습니다. 저장 공간은 매우 저렴합니다. 나중에 무언가 필요할 때(Schema-on-Read) 차고를 뒤져서 찾아야 하므로 시간이 오래 걸립니다. 관리를 안 하면 쓰레기장, 즉 데이터 스웜프(Data Swamp)가 되기 쉽습니다.

다음은 데이터 레이크와 웨어하우스의 기술적 장단점을 비교한 표입니다.

Table 2: 데이터 레이크와 데이터 웨어하우스 비교

2*측면 (Dimension)	데이터 레이크 (Data Lake)		데이터 웨어하우스 (Data Warehouse)	
	장점 (Pro)	단점 (Con)	장점 (Pro)	단점 (Con)
스토리지	모든 파일 타입 지원 (Open-format)	데이터 품질이 낮을 수 있음 파일 수준의 접근 제어	신뢰성 높은 데이터 세분화된 접근 제어	주로 정형 데이터만 지원 특정 벤더 종속적 포맷
컴퓨팅	매우 경제적임 (스토리지/컴퓨트 분리)	운영 복잡성이 높음	사용하기 쉬움 높은 동시성, 낮은 지연시간	확장 시 비용이 많이 듦 (스토리지/컴퓨트 결합)
소비	풍부한 도구 생태계 (ML, AI, DS)	BI 사용 사례에 최적화 안됨	SQL에 최적화 (BI)	ML, 스트리밍 사용 제한

3.2 아키텍처의 한계와 레이크하우스 (Lakehouse)의 등장

전통적으로 기업들은 두 시스템을 함께 사용하려 했습니다.

- 1세대 (DW Only): 정형 데이터만 ETL을 통해 DW에 적재. BI, 리포트에만 사용.
- 2세대 (Two-tier: Lake + DW): 모든 데이터를 DL에 저장. 이 중 필요한 정형 데이터만 다시 ETL을 통해 DW로 복사하여 BI에 사용. ML/DS는 DL을 사용.

2세대 (Two-tier) 아키텍처의 문제점

모든 데이터를 DL에 저장하고 BI용 데이터만 DW로 복사하는 2세대 방식은 여러 문제를 야기했습니다.

- **데이터 중복:** 똑같은 데이터가 Lake와 Warehouse에 이중으로 저장되어 비용이 낭비됩니다.
- **복잡성 증가:** Lake와 DW라는 두 개의 시스템을 별도로 관리하고 동기화해야 합니다.
- **데이터 최신성 문제:** Lake의 원본 데이터가 DW로 복사(ETL)되는 데 시간이 걸려, BI 사용자는 항상 최신 데이터를 보지 못할 수 있습니다.

이러한 문제를 해결하기 위해 레이크하우스(Lakehouse) 아키텍처가 등장했습니다.

레이크하우스(Lakehouse)란? 데이터 레이크의 저렴하고 유연한 개방형 스토리지(예: S3, ADLS) 위에, 데이터 웨어하우스의 핵심 기능(ACID 트랜잭션, 데이터 거버넌스, 빠른 쿼리 성능)을 제공하는 단일 통합 아키텍처입니다. (예: Databricks Delta Lake)

레이크하우스의 장점:

- 단일 시스템 (Simplicity): 데이터 복제나 별도 시스템 관리가 필요 없습니다.
- 모든 워크로드 지원: BI, 리포팅, 데이터 사이언스(DS), 머신러닝(ML) 등 모든 작업을 동일한 단일 데이터 소스에서 직접 수행할 수 있습니다.
- 비용 효율성: 웨어하우스의 성능을 레이크의 저렴한 비용으로 달성합니다.
- 최신성: 데이터가 한 곳에만 있으므로(Single Source of Truth), BI 사용자도 항상 최신 데이터를 쿼리 할 수 있습니다.

3.3 데이터 저장소 기술의 역사적 흐름

현재의 레이크하우스 개념은 다음과 같은 기술적 진화를 거쳐 등장했습니다.

1. 스프레드시트 (Spreadsheets): 가장 원시적인 데이터 저장소 (예: CSV)
2. 데이터 웨어하우스 (Data Warehouse):
 - Bill Inmon (인몬): ER 모델 기반, 정규화(3NF)된 중앙 DW를 구축. (데이터 일관성 중시)
 - Ralph Kimball (김볼): 비즈니스 사용자에 초점, 비정규화된 차원 모델(Star/Snowflake Schema)을 제안. (BI 쿼리 속도 중시)
3. MPP (대규모 병렬 처리): Teradata, Greenplum 등. 데이터와 컴퓨팅을 여러 노드에 분산. 테라바이트(TB)급 처리가 가능해졌으나 매우 고가였습니다.
4. NoSQL / BigTable: Google이 시작. 페타바이트(PB)급 대용량 테이블 데이터를 처리.
5. Hadoop / 데이터 레이크: Doug Cutting. 저렴한 하드웨어(범용 서버)를 수평적으로 확장. 스토리지와 컴퓨팅을 분리하는 개념을 도입하며 '데이터 레이크' 시대를 열었습니다.
6. 데이터 메시 / 패브릭 (Data Mesh / Fabric): 데이터를 '제품'으로 취급하는 분산형 아키텍처 (Mesh)와 데이터 위치를 추상화하는 기술(Fabric)이 등장.
7. 레이크하우스 (Lakehouse): 데이터 레이크 위에 DW 기능을 결합한 현재의 아키텍처.

4 핵심 개념 2: 비즈니스 인텔리전스 (BI) 란?

4.1 BI의 정의와 목적

비유: “오즈의 마법사”의 수정 구슬

BI는 기업의 경영진에게 마치 ‘수정 구슬’과 같습니다. 비즈니스는 데이터를 통해 현재 무슨 일이 일어나고 있는지 명확히 보고(Insight), 특히 미래에 어떤 일이 일어날지 예측(Foresight)하여 경쟁 우위(Competitive Advantage)를 점하고 싶어 합니다.

비즈니스 인텔리전스 (BI) 란, 기업의 데이터를 수집, 통합, 분석, 시각화하여 더 나은 비즈니스 의사결정을 지원하는 모든 기술, 애플리케이션, 프로세스를 의미합니다.

BI의 핵심 철학

”데이터(Data)는 분석(Aalytics)을 위해 필요한 것이다.
정보(Information)는 비즈니스(Business)를 위해 필요한 것이다.”

BI는 원본 데이터(Data)를 가공하여 비즈니스에 유용한 ‘정보(Information)’로 만드는 과정입니다.

BI의 주요 구성 요소:

- **데이터 분석 (Data Analysis):** 데이터 탐색 및 쿼리
- **시각적 분석 (Visual Analytics):** 차트, 그래프, 대시보드
- **고급 분석 (Advanced Analytics):** 예측, 통계 (BA 영역과 겹침)
- **데이터 거버넌스 (Data Governance):** 데이터 품질, 보안, 접근 제어
- **전략 문서화 (Strategy Documentation):** 비즈니스 미션과 전략

4.2 BI vs. BA: 무엇이 다른가?

BI와 BA(Business Analytics)는 자주 혼용되지만, 초점과 질문이 다릅니다.

Table 3: Business Intelligence (BI) vs. Business Analytics (BA)

비즈니스 인텔리전스 (BI)	비즈니스 분석 (BA)
과거와 현재의 데이터를 사용합니다.	과거 데이터를 사용합니다.
”무엇”이, ”어떻게” 일어났는지에 집중합니다. (Descriptive)	”왜” 일어났는지 설명하고, ”무엇이 일어날지” 예측합니다. (Explanatory, Predictive)
주요 질문 예시: <ul style="list-style-type: none"> • ”지난 분기 매출은 얼마인가?” (What) • ”가장 많이 팔린 제품은?” (Who) • ”언제 가장 많이 팔렸나?” (When) 	주요 질문 예시: <ul style="list-style-type: none"> • ”왜 그 제품이 많이 팔렸나?” (Why) • ”이 추세가 계속될까?” (Will it happen again?) • ”가격을 10% 올리면 어떻게 될까?” (What if?)
주요 기술: 리포팅, 대시보드, OLAP, Ad-hoc 쿼리	주요 기술: 통계 분석, 데이터 마이닝, 예측 모델링, A/B 테스트

최근 BI의 트렌드는 단순한 ‘서술적(Descriptive)’ 분석(과거 리포팅)에서 ‘처방적(Prescriptive)’ 분석

(미래에 무엇을 해야 하는지 제안)으로 나아가고 있습니다.

4.3 BI 프로세스 (5단계)

BI는 다음과 같은 5단계를 거쳐 비즈니스 가치를 창출합니다.

1. 데이터 수집 (Collect): 여러 소스 시스템(CRM, ERP 등)의 데이터를 데이터 웨어하우스(또는 레이크하우스)로 통합(ETL)합니다.
2. 데이터 조직 (Organize): 수집된 데이터를 분석하기 좋은 모델(예: OLAP 큐브, Star Schema)로 구성합니다.
3. 데이터 분석 (Analyze): BI 분석가나 사용자가 SQL을 사용해 데이터를 쿼리합니다.
4. 데이터 시각화 (Visualize): 쿼리 결과를 차트, 대시보드, 리포트 등 이해하기 쉬운 형태로 만듭니다.
5. 의사 결정 (Decide): 경영진과 실무자가 이 시각화된 '정보'를 보고 전략적 의사결정을 내립니다. (예: 어떤 신제품을 개발할지, 어떤 시장에 진출할지)

5 핵심 개념 3: BI 페르소나와 데이터 모델링

5.1 BI 분석가 (BI Analyst) 페르소나

BI 워크플로우에는 여러 역할(데이터 엔지니어, 데이터 과학자 등)이 있지만, BI의 핵심 소비자는 **BI 분석가**입니다.

- **주요 기술:** SQL BI 분석가의 주무기는 **SQL**입니다. 이들은 SQL을 사용해 데이터를 탐색하고, 비즈니스 질문에 답하며, 대시보드에 필요한 데이터를 추출합니다.
- **소비 데이터:** 정제된 데이터 (**Curated Data**) BI 분석가는 원본(Bronze) 데이터가 아닌, 데이터 엔지니어가 1차 정제(Silver)하고 비즈니스 용도에 맞게 집계/가공한(Gold) 데이터를 주로 사용합니다.
- **역할:** 분석 엔지니어링 (**Analytics Engineering**) 최근에는 BI 분석가가 SQL을 사용해 데이터를 모델링하고 골드 테이블을 직접 쿼리하는 역할까지 맡기도 하며, 이를 '분석 엔지니어링'이라고 부릅니다.

5.2 BI를 위한 데이터 모델링 (Data Modeling for BI)

BI 쿼리는 매우 빨라야 하므로(Low Latency), 데이터를 BI에 최적화된 구조로 모델링해야 합니다. 이때 가장 널리 쓰이는 방식이 차원 모델링(Dimensional Modeling), 특히 스타 스키마(Star Schema)입니다.

- **스타 스키마 (Star Schema):** 이름처럼 '별' 모양의 구조입니다.
 - **팩트 테이블 (Fact Table):** 중앙에 위치. 비즈니스 이벤트의 측정값(숫자 데이터)을 담습니다. (예: ‘sales_amount’, ‘quantity_{sold}’)
 - **디멘션 테이블 (Dimension Tables):** 팩트 테이블을 둘러싼 '별'의 꼭짓점. 이벤트가 일어난 맥락 (Context)을 설명합니다. (예: dim_customer, dim_product, dim_time)
- **스노우플레이크 스키마 (Snowflake Schema):** 스타 스키마의 변형. 디멘션 테이블이 추가로 정규화되어 또 다른 테이블에 연결된 구조. (예: ‘dim_product‘ ‘dim_category‘)

스타 스키마 예시: 온라인 상점 매출

- Fact_Sales (팩트 테이블): {date_key, product_key, customer_key, sales_amount, quantity}
- Dim_Time (시간 차원): {date_key, date, month, year, quarter, day_of_week}
- Dim_Product (제품 차원): {product_key, product_name, category, brand}
- Dim_Customer (고객 차원): {customer_key, customer_name, city, country}

"2025년 1분기, 서울에 거주하는 고객들의 카테고리별 매출액은?" 같은 BI 쿼리를 매우 빠르고 단순한 Join으로 처리할 수 있습니다.

Data Vault 모델

또 다른 모델로 **Data Vault**가 있습니다. 이는 허브(Hubs: 핵심 비즈니스 키), 링크(Links: 관계), 새틀라이트(Satellites: 설명 속성)로 구성되며, 변화에 유연하게 대응할 수 있습니다.

하지만 Data Vault가 Silver 레이어에서 데이터를 유연하게 통합하는 데 쓰이더라도, 최종적으로 BI 사용자가 쿼리하는 Gold 레이어는 여전히 Star Schema 같은 차원 모델로 변환되는 경우가 많습니다.

6 Databricks를 활용한 BI 및 데이터 웨어하우징

Databricks는 '레이크하우스' 아키텍처를 기반으로 BI 및 데이터 웨어하우징 기능을 제공합니다.

6.1 Databricks 데이터 인텔리전스 플랫폼

Databricks 플랫폼은 데이터 흐름에 따라 여러 구성요소로 나뉩니다.

Source (소스): 모든 종류의 데이터 (정형, 비정형, 스트리밍)

Ingest (수집): 데이터를 레이크하우스로 가져옵니다.

- **ETL:** 전통적인 방식. 데이터를 복사하여 레이크하우스가 '소유'합니다.
- **Data Federation (데이터 연합):** 데이터를 복사하지 않고, 외부 시스템(예: Oracle, Redshift)에 읽기 전용(read-only) 쿼리를 날려 가상으로 데이터를 가져옵니다. 소유권(Ownership)이 없는 것이 ETL과의 핵심 차이입니다.

Transform (변환): Medallion Architecture (Bronze → Silver → Gold)에 따라 데이터를 정제하고 가공합니다.

Query and Process (쿼리/처리): • **Databricks SQL (DBSQL):** BI 및 데이터 웨어하우징 워크로드를 위한 SQL 엔진입니다. ANSI SQL 표준을 따릅니다.

- **Data Science & ML:** 데이터 과학 및 머신러닝 워크로드.

Governance (거버넌스): Unity Catalog가 모든 데이터 자산(테이블, 파일, 모델)의 접근 제어, 데이터 계보(Lineage), 감사를 중앙에서 관리합니다.

Core Engine (엔진): Photon (Spark을 C++로 재작성한 차세대 벡터화 엔진)이 쿼리 성능을 높여줍니다.

Serve/Analysis (제공/분석): Lakeview Dashboards (내장 대시보드), BI Tools (Tableau, PowerBI 연동), Lakehouse Apps 등을 통해 최종 사용자에게 데이터를 제공합니다.

ETL vs. Federation 선택 기준

Federation은 데이터를 이동할 필요가 없어 편리해 보이지만 만능이 아닙니다.

- **Federation이 적합할 때:** 외부 시스템의 데이터 용량이 작거나(Modest data), 단순 참조/조회용(Lookup) 일 때 사용합니다.
- **ETL이 적합할 때:** 대용량 데이터를 처리해야 하거나, 낮은 지연 시간(Low Latency)의 빠른 쿼리 성능이 반드시 필요할 때는 ETL을 통해 데이터를 레이크하우스로 물리적으로 가져와야 합니다.

6.2 Databricks SQL (DBSQL)의 주요 BI 기능

DBSQL은 BI 분석가를 위해 다음과 같은 강력한 기능들을 제공합니다.

- **서버리스 컴퓨팅 (Serverless Compute):** 쿼리 요청 시 즉시 컴퓨트 자원을 할당받습니다. 기존 방식처럼 VM(클러스터)이 켜지는 데 3~6분씩 기다릴 필요가 없습니다.
- **스트리밍 테이블 (Streaming Tables):** 복잡한 코드 없이, SQL만으로 스트리밍 데이터 소스(예: Kafka, 클라우드 파일)를 읽어 자동으로 업데이트되는 테이블을 정의할 수 있습니다.
- **구체화된 뷰 (Materialized Views - MV):** 복잡하고 오래 걸리는 쿼리 결과를 미리 계산하여 물리적 테이블로 저장합니다. BI 대시보드가 이 MV를 조회하면 매우 빠른 응답을 얻을 수 있습니다. 데이터

원본이 변경되면 MV는 증분(incrementally) 업데이트됩니다.

- **지오스페이셜 지원 (Geospatial Support):** 위치/자리 정보(예: H3)를 처리하는 함수를 SQL에서 바로 지원합니다.

6.3 SQL의 AI 기능 (AI Functions in SQL)

Databricks SQL은 LLM(거대 언어 모델)을 SQL 쿼리 내에서 직접 호출하는 혁신적인 기능을 제공합니다.

6.3.1 1. ai_query(): 외부 LLM 호출

ai_query() 함수를 사용하면 SQL 문 내에서 OpenAI의 GPT나 Anthropic의 Claude 같은 외부 모델을 직접 호출하여 데이터를 보강(enrich)할 수 있습니다.

```

1  -- 'my-openai-chat'이라는 모델엔드포인트를 호출
2  SELECT
3    sku_id,
4    product_name,
5    ai_query(
6      "my-openai-chat",   -- 미리등록한모델엔드포인트
7      -- 프롬프트: 제품이름을포함하여단어 30 흥보문구생성
8      "You are a marketing expert. Generate a promotional text
9      in 30 words for product: " || product_name
10     ) AS promotional_text
11 FROM
12   retail_products;

```

Listing 1: ai_query()를 사용한 제품 홍보 문구 생성 예시

6.3.2 2. 내장 AI 함수 (Built-in AI Functions)

자주 사용되는 AI 작업을 위해 미리 구축된 모델을 제공합니다. 외부 모델을 설정할 필요 없이 즉시 사용 가능합니다.

```

1  -- 감성분석 (Sentiment Analysis)
2  -- 'positive', 'negative', 'neutral' 반환
3  SELECT ai_analyze_sentiment('I am happy');
4  -- 결과: positive
5
6  -- 텍스트분류 (Classification)
7  -- 주어진레이블중하나로분류
8  SELECT ai_classify('My password is leaked.', ARRAY('urgent', 'not urgent'));
9  -- 결과: urgent
10
11 -- 정보추출 (Extraction)
12 -- 텍스트에서원하는정보이름 (, 이메일등) 추출
13 SELECT ai_extract('John Doe lives in New York', ARRAY('person', 'location'));
14 -- 결과: {"person": "John Doe", "location": "New York"}
15
16 -- 문법교정 (Grammar Fix)

```

```
17 SELECT ai_fix_grammar('This sentence have some mistake');  
18 -- 결과: 'This sentence has some mistakes'  
19  
20 -- 민감정보마스킹 (Masking)  
21 SELECT ai_mask('My email is john.doe@example.com', ARRAY('email'));  
22 -- 결과: 'My email is [MASKED]'
```

Listing 2: 내장 AI 함수 사용 예시

7 실습: Lakeview 대시보드와 Genie

Databricks는 내장 대시보드 도구인 **Lakeview**와 대화형 AI 분석 도구인 **Genie**를 제공합니다.

7.1 Lakeview 대시보드 (Lakeview Dashboards)

- 정의:** Databricks 플랫폼 내에서 직접 데이터를 시각화하고 대시보드를 구축하는 도구입니다.
- 구성:** '데이터' 탭에서 대시보드에 사용할 데이터를 정의하고, 캔버스에 '시각화 위젯', '텍스트 상자', '필터' 등을 추가하여 대시보드를 만듭니다.
- 자연어 생성:** 차트를 만들 때, 내장된 Assistant에게 자연어로 요청할 수 있습니다. (예: "show me total revenue by zip code")

Lakeview vs. PowerBI/Tableau: 언제 무엇을 쓸까?

- Lakeview (Databricks 내장):** 추가 라이선스 비용이 없습니다. 데이터 엔지니어, 분석가가 빠르게 데이터를 확인하고 공유하는 '운영용' 대시보드에 적합합니다.
- PowerBI / Tableau (외부 BI 도구):** 경영진(CEO 등)에게 보고하기 위한 매우 정교하고 복잡한 (예: 3D 효과, 특정 브랜드 색상) '전략적' 대시보드에 적합합니다.

7.2 게시 (Publish) 및 공유 (Share)

- 공유 (Share):** 조직 내부의 다른 Databricks 사용자에게 대시보드 접근 권한(보기/편집)을 부여합니다.
- 게시 (Publish):** Databricks 계정이 없는 외부 사용자에게 대시보드를 공유하는 기능입니다.
 - '자격 증명 포함(Embed credentials)' 옵션으로 게시합니다.
 - 비용 주의:** 게시된 대시보드의 쿼리가 캐시(Cache)되지 않은 상태에서 외부 사용자가 대시보드를 조회하면, 대시보드를 게시한 사람(Publisher)의 계정에서 컴퓨터 비용이 발생합니다.

7.3 Databricks Genie: 대화형 AI 분석

Genie는 게시된 대시보드에서 사용할 수 있는 대화형 AI 비서입니다.

- 사용 시나리오:** BI 분석가가 만든 대시보드를 현업 사용자(예: 마케팅 매니저)가 보다가 추가적인 궁금증이 생겼습니다. (예: "이 데이터에서 20대 남성 고객만 보면 어떨까?") 과거에는 이 요청을 BI팀에 보내고 답변까지 몇 주를 기다려야 했지만, 이제는 Genie에게 자연어로 직접 질문하여 즉시 답을 얻을 수 있습니다.
- 작동 방식 (예시):**
 - 사용자 질문: "평균 여행 시간은 얼마야?"
 - Genie 답변: "평균 13.7분입니다."
- 투명성 (Transparency):** Genie는 이 답을 얻기 위해 실행한 SQL 쿼리를 함께 보여줍니다. (예: `SELECT avg(dropoff_time - pickup_time) ...`)
- 환각(Hallucination)이 없는 이유:** Genie는 ChatGPT 같은 범용 챗봇이 아닙니다. Genie의 답변 범위는 대시보드를 생성할 때 사용된 특정 테이블의 메타데이터(컬럼명, 타입 등)로 엄격하게 제한됩니다. 만약 사용자가 "날씨가 어때?"처럼 데이터와 무관한 질문을 하면, Genie는 "죄송합니다. 저는 해당

데이터에 대해서만 답변할 수 있습니다”라고 답하며 환각을 일으키지 않습니다.

8 학습 점검 체크리스트

이 강의의 핵심 내용을 잘 이해했는지 다음 항목들로 점검해 보세요.

데이터 웨어하우스(DW)와 데이터 레이크(DL)의 4가지 주요 차이점(데이터, 스키마, 비용, 용도)을 설명할 수 있는가?

”레이크하우스” 아키텍처가 왜 등장했으며, 기존 2-tier (Lake + DW) 아키텍처의 어떤 문제를 해결하는가? (힌트: 데이터 중복, 복잡성)

BI와 BA의 차이점을 ”질문”과 ”목적” 관점에서 비교 설명할 수 있는가?

BI 분석가 페르소나의 핵심 기술은 무엇인가? (정답: SQL)

데이터 모델링에서 킴볼(Kimball)의 ”스타 스키마”가 무엇인지 팩트/디멘션 테이블로 설명할 수 있는가?

”데이터 연합(Data Federation)”과 ”ETL”的 가장 큰 차이점은 무엇인가? (정답: 데이터 소유권/이동 여부)

”뷰(View)”와 ”구체화된 뷰(Materialized View)”의 차이점(저장 공간, 성능)을 설명할 수 있는가?

Databricks SQL의 ai_query() 와 ai_analyze_sentiment() 의 차이점과 용도를 아는가?

Lakeview 대시보드를 PowerBI 대신 사용하는 이유는 무엇인가? (운영용, 라이선스 비용 없음)

Databricks Genie가 챗GPT와 달리 ”환각(Hallucination)”을 일으키지 않는 이유는 무엇인가? (정답: 지정된 테이블 메타데이터만 참조)

외부 사용자와 대시보드를 ”게시(Publish)” 기능으로 공유할 때 발생할 수 있는 비용 문제는 무엇인가?

9 FAQ (자주 묻는 질문)

Q: 데이터 레이크가 있는데 왜 굳이 데이터 웨어하우스가 필요한가요?

A: 데이터 레이크는 모든 데이터를 *저장*하는 데는 뛰어나지만, 정제되지 않아 *분석*하기에는 느리고 복잡합니다. 데이터 웨어하우스는 BI 리포팅 및 대시보드처럼 *매우 빠른 응답 속도*(Low Latency)와 *높은 동시성*(High Concurrency)이 필요한 BI 워크로드에 특화되어 있습니다. 레이크하우스는 이 두 장점을 결합하려는 시도입니다.

Q: 레이크하우스는 그냥 마케팅 용어 아닌가요? 데이터 레이크와 뭐가 다른가요?

A: 레이크하우스는 데이터 레이크의 저렴한 저장소(S3, ADLS 등) 위에 ACID 트랜잭션, 데이터 버전 관리, 인덱싱 등 웨어하우스의 핵심 기능을 제공하는 *기술적 아키텍처*(예: Delta Lake)입니다. 덕분에 별도의 DW 없이 레이크에서 직접 BI와 ML을 모두 수행할 수 있습니다.

Q: ’데이터 연합(Federation)’은 항상 ETL보다 좋은 것 아닌가요?

A: 아닙니다. 연합은 데이터를 복제/이동하지 않아 편리하지만, 실시간으로 원격 시스템에 쿼리를 날립니다. 이는 *소량의 데이터*나 *참조용 데이터*(lookup)에는 좋지만, 대용량 데이터를 조인하거나 빠른 성능이 필요할 때는 매우 비효율적입니다. 이 경우 ETL을 통해 데이터를 레이크하우스로 가져오는 것이 성능상 유리합니다.

Q: Genie가 SQL을 생성해준다면, 이제 SQL을 배울 필요가 없나요?

A: 아닙니다. Genie는 *보조* 도구입니다. Genie가 생성한 SQL이 100% 정확하지 않을 수 있으며, 복잡한 비즈니스 로직을 구현하려면 여전히 SQL 지식이 필수입니다. Genie가 생성한 SQL을 *검증하고

수정*할 수 있어야 합니다.

10 빠르게 훑어보기 (1-Page Summary)

저장소 비교: 웨어하우스 vs. 레이크

- **웨어하우스(DW):** 깐깐한 도서관 (정형 데이터, Schema-on-Write, BI/SQL 최적화, 고비용, 고성능)
- **레이크(Lake):** 막 쌓는 차고 (모든 데이터, Schema-on-Read, ML/DS 최적화, 저비용, 저성능)

레이크하우스 (Lakehouse): 두 세계의 통합

레이크의 저렴한 스토리지 + 웨어하우스의 성능/안정성/거버넌스

- 단일 시스템에서 BI와 ML/DS 워크로드를 모두 지원.
- 데이터 중복 및 ETL 파이프라인 복잡성 해소.

질문의 차이: BI vs. BA

- **BI (Business Intelligence):** ”무엇이 일어났나?” (과거/현재 리포팅, 대시보드)
- **BA (Business Analytics):** ”왜 일어났나? 무엇이 일어날까?” (통계, 예측 모델링)

BI 분석가와 데이터 모델링

- **BI 페르소나:** 핵심 기술은 SQL.
- **데이터 모델링:** Star Schema (중앙의 팩트 테이블 + 주변의 디멘션 테이블) 가 BI 쿼리 성능에 가장 효율적임. (킴볼 방식)

Databricks SQL 핵심 기능

- **서비스 컴퓨트:** 클러스터 부팅 대기 시간 없음 (Instant Compute).
- **Materialized Views (MV):** 복잡한 쿼리 결과를 미리 계산/저장하여 대시보드 속도 향상.
- **Streaming Tables:** SQL만으로 스트리밍 데이터 처리.

SQL + AI: 지능형 쿼리

- **ai_query():** SQL 내에서 외부 LLM (GPT 등) 호출.
- **ai_analyze_sentiment(), ai_classify()...:** 내장 AI 함수로 감성분석, 분류 등을 SQL로 즉시 수행.

대시보드와 AI 비서: Lakeview Genie

- **Lakeview:** Databricks 내장 대시보드 (운영용, 라이선스 무료).
- **Genie:** 게시된 대시보드에서 사용하는 대화형 AI. 자연어 질문을 SQL로 변환.
- **Genie의 특징:** 환각(Hallucination) 없음. 지정된 테이블의 메타데이터만 참조.

December 10, 2025

■ 강의명: CSCI E-103: 재현 가능한 머신러닝

■ 주차: Lecture 07

■ 교수명: Anindita Mahapatra

Eric Gieseke

■ 목적: Lecture 07의 핵심 개념 학습

Contents

1	핵심 용어 정리	2
2	2부: 데이터 파이프라인 운영의 개념	4
2.1	왜 ”운영(Operationalizing)”이 필요한가?	4
2.2	요구사항 정의: 모든 설계의 시작	4
2.3	데이터 파이프라인의 6가지 핵심 원칙	5
2.3.1	1. 데이터 큐레이션 및 신뢰할 수 있는 ”데이터 제품” 제공	5
2.3.2	2. 데이터 파일로 제거 및 이동 최소화	5
2.3.3	3. 셀프 서비스 경험으로 가치 창출 민주화	5
2.3.4	4. 전사적 데이터 거버넌스 전략 채택	5
2.3.5	5. 개방형 인터페이스 및 포맷 사용	6
2.3.6	6. 확장성 및 비용/성능 최적화	6
3	2부: 데이터 품질 및 검증	7
3.1	데이터 품질이란? (Garbage In, Garbage Out)	7
3.2	손상된 레코드 처리 (Handling Corrupt Records)	7
3.3	결측치 및 중복 데이터 처리	7
3.4	DLT를 활용한 자동 데이터 품질 관리 (Expectations)	8
4	3부: Databricks Lakeflow 및 DLT 실습	9
4.1	명령형 vs 선언형 파이프라인	9
4.2	DLT의 핵심: CDC (Change Data Capture) 자동화	9
5	4부: 실습 오류 및 해결 (Checkpoint Q&A)	11
5.1	사례: 스트리밍 체크포인트 오류	11

5.2 해결 방안	11
6 5부: 요약 및 점검	13
6.1 운영 파이프라인 체크리스트	13
6.2 FAQ (자주 묻는 질문)	13
6.3 빠르게 훑어보기 (1페이지 요약)	14
A 부록: 과제 및 공지사항	15

▣ 핵심 요약

이 문서는 데이터 파이프라인을 개념 증명(PoC) 단계에서 실제 운영(Production) 환경으로 전환하는데 필요한 핵심 원칙과 기술을 요약합니다.

데이터 과학자가 만든 노트북(프로토타입)을 비즈니스에서 신뢰하고 사용할 수 있는 자동화된 시스템으로 만드는 과정을 다룹니다.

이 과정에는 명확한 요구사항 정의(기능적/비기능적), 데이터 품질 보장, 거버넌스 수립, 그리고 확장성 확보가 필수적입니다.

Databricks의 **Lakeflow** 및 **DLT(Delta Live Tables)**와 같은 도구가 이 복잡한 과정을 어떻게 '선언형'으로 단순화하는지 살펴봅니다.

1 핵심 용어 정리

데이터 파이프라인 운영을 이해하기 위해 다음 용어들을 먼저 숙지해야 합니다.

주요 용어 해설표

용어	원어	한 줄 요약
운영(화)	Operationalizing	프로토타입을 실제 서비스
개념 증명	PoC (Proof of Concept)	아이디어가 기술적으로 실현 가능
기능적 요구사항	Functional Req.	시스템이 '무엇을' 해야 하는지
비기능적 요구사항	Non-Functional Req.	시스템이 '어떻게' 동작해야 하는지
Medallion Arch.	데이터를 Bronze → Silver → Gold 3단계로 정제하는 구조.	원석(Bronze)을 다듬어 보석(Silver)과 보석(Gold)로 만드는 과정
데이터 거버넌스	Data Governance	데이터의 품질, 보안, 접근성 관리
데이터 계보	Data Lineage	데이터가 어디서 와서 어떻게 처리되는지
ETL / ELT	-	데이터 처리 순서. (Extract, Transform, Load)
Lakeflow	Lakeflow	Databricks의 데이터 파이프라인
DLT	Delta Live Tables	선언형 ETL 프레임워크. 노출된 데이터를 실시간으로 업데이트
선언형 파이프라인	Declarative Pipeline	'어떻게'가 아닌 '무엇을' 정의
오토로더	Autoloader	클라우드 스토리지에 새 파일을 자동으로 업로드
CDC	Change Data Capture	원본 데이터의 변경(Insert, Update, Delete) 감지
확장성	Scalability	시스템이 증가하는 작업량에 대처
탄력성	Elasticity	작업량 변화에 따라 자원을 조절
체크포인트	Checkpoint	스트리밍 작업이 어디까지

2 1부: 데이터 파이프라인 운영의 개념

2.1 왜 ”운영(Operationalizing)”이 필요한가?

데이터 과학자가 만든 프로토타입(PoC) 노트북은 특정 문제를 해결할 수 있음을 보여주지만, 비즈니스에서 매일 신뢰하며 사용하기에는 부족합니다.

□ 예제: title

- **PoC (노트북):** 세프가 주방에서 실험적으로 만든 ‘신메뉴 레시피’와 같습니다. 맛은 있지만, 한 번에 1인분만 만들 수 있고, 세프의 컨디션에 따라 품질이 달라질 수 있습니다.
- **운영 (Production):** 이 레시피를 ‘대규모 식품 공장’으로 가져와, 하루에 수만 개씩 일정한 품질로 자동 생산하는 시스템을 구축하는 것입니다.

운영 파이프라인은 신뢰성(Reliability), 자동화(Automation), 확장성(Scalability), 모니터링(Monitoring)을 갖춰야 합니다.

2.2 요구사항 정의: 모든 설계의 시작

설계를 시작하기 전에 비즈니스(고객)가 ’무엇을’ 원하고 ’어떻게’ 작동하길 기대하는지 명확히 해야 합니다. 이는 두 가지로 나뉩니다.

1. 기능적 요구사항 (Functional Requirements) - ”무엇을?”

- 시스템이 사용자에게 제공해야 하는 구체적인 기능을 정의합니다.
- 비즈니스 규칙 (예: 특정 조건에서 거래 취소)
 - 인증 및 권한 수준 (예: 관리자만 데이터 삭제 가능)
 - 외부 인터페이스 (예: 여러 데이터 소스 연결)
 - 리포팅 요구사항 (예: 일일 매출 요약 생성)
 - 감사 추적 (예: 누가 데이터를 수정했는지 기록)

2. 비기능적 요구사항 (Non-Functional Requirements) - ”어떻게?”

시스템의 품질, 성능, 제약 조건을 정의합니다. 기능적 요구사항만큼, 때로는 그 이상으로 중요합니다.

- **성능 (Performance):** 응답 시간(Latency), 처리량(Throughput) (예: 쿼리 응답은 5초 이내)
- **확장성 (Scalability):** 향후 데이터/사용자 증가에 대한 대비 (예: 10배 많은 데이터 처리 가능)
- **가용성 (Availability):** 시스템이 중단 없이 작동하는 시간의 비율
- **보안 (Security):** 데이터 접근 제어 및 규정 준수

비기능적 요구사항의 함정

비기능적 요구사항은 종종 간과되지만, 프로젝트 성패를 좌우합니다.

- **과잉 설계 (Over-design):** 비즈니스는 하루 지연된 데이터를 받아도 되는데, 엔지니어가 1초 미만 실시간 시스템을 구축하면 막대한 비용이 낭비됩니다.

- **미달 설계 (Under-design):** 1초 미만 응답을 기대하는 고객에게 10초 걸리는 시스템을 제공하면, 그 시스템은 비즈니스 가치를 잃게 됩니다.
예를 들어, 'Five Nines' (99.999%) 가용성은 1년 중 단 5분의 장애만 허용하며, 이는 엄청난 구축 비용을 요구합니다.

2.3 데이터 파이프라인의 6가지 핵심 원칙

효율적이고 견고한 데이터 레이크하우스(Data Lakehouse)를 구축하기 위한 6가지 지침입니다.

2.3.1 1. 데이터 큐레이션 및 신뢰할 수 있는 "데이터 제품" 제공

데이터를 목적에 맞게 정제하고 가공하여, 사용자가 신뢰하고 소비할 수 있는 '제품' 형태로 제공해야 합니다.

메달리온 아키텍처 (Medallion Architecture)

데이터를 품질 수준에 따라 3단계로 관리하는 표준 방식입니다.

- **Bronze (Raw Layer):** 원본 소스 데이터를 변경 없이 그대로 저장합니다. (원석)
- **Silver (Curated Layer):** Bronze 데이터를 가져와 정제(Cleansing), 필터링, 보강(Enriching)합니다. (정제된 은)
- **Gold (Final Layer):** Silver 데이터를 비즈니스 목적(예: BI, 리포팅, ML)에 맞게 집계(Aggregation)하고 요약합니다. (최종 제품, 금)

Bronze → Silver → Gold로 갈수록 데이터의 품질과 신뢰도는 높아지지만, 가공 비용(Cost)도 증가합니다.

2.3.2 2. 데이터 사일로 제거 및 이동 최소화

데이터를 불필요하게 복제하고 이동시키면 비용, 지연 시간(Latency), 품질 문제가 발생하며, 부서 간 데이터가 고립되는 '사일로(Silo)'가 생깁니다.

- **해결책:** 데이터 복제 대신 얇은 복제(Shallow Clone, 메타데이터만 복사), 뷰(Views), 델타 타임 트래블(Delta Time Travel) 등을 활용하여 데이터 중복을 최소화합니다.

2.3.3 3. 셀프 서비스 경험으로 가치 창출 민주화

데이터 팀뿐만 아니라 현업의 비즈니스 사용자들도 필요한 데이터를 직접 탐색하고 활용할 수 있어야 합니다. (데이터 민주화)

- **필수 조건:** 사용자가 데이터를 함부로 변경하거나 삭제하지 못하도록 적절한 "가드레일(Guardrails)", 즉 데이터 거버넌스(보안 및 접근 제어)가 반드시 전제되어야 합니다.

2.3.4 4. 전사적 데이터 거버넌스 전략 채택

데이터 거버넌스는 단순한 보안이 아니라, 데이터의 수명 주기 전반을 관리하는 시스템입니다.

- **데이터 품질 (Quality):** 데이터가 정확하고 일관되도록 제약 조건(Constraints)을 적용합니다.
- **데이터 카탈로그 및 계보 (Catalog & Lineage):** 데이터의 의미(메타데이터)를 정의하고, 데이터의

출처와 변환 이력(Lineage)을 추적 가능하게 합니다.

- 접근 제어 (Access Control): 누가(Who) 어떤(What) 데이터에 접근할 수 있는지 관리합니다. (예: PII-개인식별정보 마스킹, 행/열 단위 접근 제한)

2.3.5 5. 개방형 인터페이스 및 포맷 사용

특정 벤더(Vendor)에 종속되는 독점적(Proprietary) 포맷 대신, 델타(Delta), 파케이(Parquet) 같은 개방형 포맷(Open Format)을 사용해야 합니다.

- 이유: 1. 벤더 종속 탈피 (No Lock-in): 다른 플랫폼으로 자유롭게 이전할 수 있습니다. 2. 상호 운용성 (Interoperability): 다양한 3rd-party 도구와 쉽게 연동됩니다. 3. 비용 절감 (Lower Cost): 값비싼 독점 플랫폼 라이선스를 피할 수 있습니다.

2.3.6 6. 확장성 및 비용/성능 최적화

대규모 데이터를 효율적으로 처리하려면 확장성과 비용 효율성을 모두 고려해야 합니다.

핵심: 스토리지와 컴퓨터의 분리 (Decoupling)

가장 중요한 아키텍처 원칙 중 하나입니다.

- 전통적 시스템 (결합): 데이터를 저장하는 디스크(스토리지)와 데이터를 처리하는 CPU(컴퓨트)가 한 서버에 묶여 있습니다. 데이터만 10배 늘어도, 비싼 CPU까지 10배 증설해야 했습니다. (예: ElasticSearch)
- 현대적 레이크하우스 (분리): 데이터는 저렴한 클라우드 스토리지(S3, ADLS 등)에 무한히 저장하고, 데이터 처리가 필요할 때만 컴퓨터 클러스터를 빌려 씁니다.

□ 예제: title

- 스토리지 (Storage): 물건을 보관하는 '창고'. 창고 크기는 저렴하게 무한히 늘릴 수 있습니다.
- 컴퓨트 (Compute): 물건을 가공하는 '공장 기계'. 비싸지만, 물건을 가공할 때만 기계를 켜서 사용료를 냅니다.

이 둘을 분리하면, 창고가 아무리 커져도 공장 운영비는 필요한 만큼만 지불하게 되어 매우 효율적입니다.

3 2부: 데이터 품질 및 검증

3.1 데이터 품질이란? (Garbage In, Garbage Out)

“쓰레기가 들어가면, 쓰레기가 나온다 (GIGO)” 데이터 파이프라인의 핵심은 데이터 품질을 보장하는 것입니다. 부정확한 데이터를 기반으로 한 분석이나 AI 모델은 잘못된 비즈니스 결정으로 이어집니다.

- 정확성 (Accuracy): 데이터가 실제 값과 일치하는가?
- 일관성 (Consistency): 데이터가 시스템 내에서 모순 없이 일관되는가? (예: 'NY' 와 'New York'의 혼용)
- 완전성 (Completeness): 필수 데이터가 누락되지 않았는가?
- 적시성 (Timeliness): 데이터가 필요한 시점에 제공되는가? (신선도)
- 무결성 (Integrity): 데이터 간의 관계(예: FK)가 올바른가?

3.2 손상된 레코드 처리 (Handling Corrupt Records)

소스 데이터는 스키마 불일치, 형식 오류, 누락 값 등 다양한 이유로 ‘손상될’ 수 있습니다. Spark은 이러한 데이터를 처리하는 3가지 모드(ParseMode)를 제공합니다.

Spark의 3가지 오류 처리 모드

1. PERMISSIVE (기본값)

- 동작: 오류가 발생한 레코드를 _corrupt_record라는 별도 컬럼에 저장하고, 파싱 가능한 컬럼은 null로 채웁니다. 작업은 중단되지 않습니다.
- 용도: 파이프라인 중단 없이 모든 데이터를 일단 수집한 후, 나중에 손상된 데이터를 분석하거나 수정할 때 유용합니다.

2. DROPMALFORMED

- 동작: 손상된 레코드를 즉시 삭제(무시)합니다.
- 용도: 품질이 중요하지 않거나, 일부 데이터가 손실되어도 무방한 로그 분석 등에 사용됩니다.

3. FAILFAST

- 동작: 손상된 레코드를 만나는 즉시 작업을 중단시키고 예외(Exception)를 발생시킵니다.
- 용도: 데이터 품질이 매우 엄격하게 요구되는 금융 거래 데이터 등, 단 하나의 오류도 허용되지 않는 경우에 사용됩니다.

또한, option("badRecordsPath", "경로")를 지정하여 오류 레코드의 원본 파일을 별도 위치에 기록할 수 있습니다.

3.3 결측치 및 중복 데이터 처리

- 중복 데이터: dropDuplicates(["id", "color"])와 같이 고유 키를 기준으로 중복 행을 제거합니다.
- 결측치 (Missing Values):
 - 행 삭제 (Drop): dropna()를 사용하여 결측치가 있는 행을 무시합니다. (정보 손실 위험)
 - 대체값 (Placeholder): 스키마를 위반하지 않도록 -1이나 'N/A' 같은 값으로 채웁니다.
 - 기본 대체 (Basic Imputing): na.fill()을 사용하여 전체 평균값이나 중앙값으로 채웁니다.

- 고급 대체 (Advanced Imputing): ML 모델(예: 회귀)을 사용하여 결측치를 예측합니다.

3.4 DLT를 활용한 자동 데이터 품질 관리 (Expectations)

Delta Live Tables (DLT)는 데이터 품질 규칙을 파이프라인 정의에 직접 '기대(Expectations)'로 선언할 수 있게 해줍니다.

DLT Expectations: 데이터 품질 선언

품질 규칙(제약 조건)을 정의하고, 위반 시(On Violation) 어떻게 처리할지 지정합니다.

- EXPECT ... ON VIOLATION FAIL UPDATE:** 규칙 위반 시 파이프라인을 중단합니다. (Spark의 FAILFAST와 유사)
- EXPECT ... ON VIOLATION DROP ROW:** 규칙 위반 시 해당 행을 삭제합니다. (Spark의 DROPMALFORMED와 유사)
- EXPECT ... (처리 지정 없음):** 규칙 위반 시 데이터는 통과시키되, 위반 내역을 메트릭으로 기록하여 모니터링할 수 있게 합니다.

```

1 import dlt
2 from pyspark.sql.functions import col
3
4 # @dlt.expect_or_drop: 규칙위반시행삭제
5 @dlt.expect_or_drop("valid_age", "age > 0 AND age < 120")
6
7 # @dlt.expect_or_fail: 규칙위반시파이프라인중단
8 @dlt.expect_or_fail("valid_email", "email IS NOT NULL")
9
10 # @dlt.expect: 위반시기록만함
11 @dlt.expect("reasonable_score", "score >= 50")
12
13 @dlt.table(
14     comment="데이터 품질제약조건이 적용된      Silver 테이블"
15 )
16 def users_silver():
17     return (
18         dlt.read_stream("users_bronze")
19             .select("id", "age", "email", "score")
20     )

```

Listing 1: DLT Python에서 Expectations를 선언하는 예시

데이터 격리 (Quarantining)

규칙을 위반한(실패한) 데이터를 DROP하거나 FAIL시키는 대신, 별도의 '격리(Quarantine) 테이블'로 라우팅하는 패턴도 많이 사용됩니다.

이를 통해 메인 파이프라인은 계속 실행하면서, 실패한 데이터는 나중에 수동으로 검토하고 수정하여 다시 파이프라인에 주입(re-inject)할 수 있습니다.

4 3부: Databricks Lakeflow 및 DLT 실습

Lakeflow는 데이터 수집(Ingest), 변환(Transform), 오케스트레이션(Orchestrate)을 통합 관리하는 Databricks의 플랫폼입니다. DLT는 Lakeflow의 핵심 기능으로, '선언형' 파이프라인을 구축합니다.

4.1 명령형 vs 선언형 파이프라인

전통적인 파이프라인 코딩과 DLT의 차이점입니다.

□ 예제: title

- **명령형 (Imperative - 전통 방식):** 셰프에게 ”1. 소고기 200g을 굽고, 2. 야채를 찢고, 3. 빵을 데우고, 4....” 라며 ‘모든 절차(How)’를 직접 지시합니다. (개발자가 스케일링, 오류 처리, 리트라이, 상태 관리 코드를 모두 작성)
- **선언형 (Declarative - DLT 방식):** 메뉴판을 보고 ”스테이크 주세요”라고 ‘원하는 결과(What)’만 주문합니다. 주방(DLT 엔진)이 최적의 레시피로 요리(파이프라인 실행), 품질 검사, 서빙(데이터 저장)을 알아서 처리합니다.

DLT를 사용하면 개발자는 비즈니스 로직(변환)에만 집중하고, 복잡한 인프라 관리(오류 처리, 확장, 최적화, 모니터링)는 DLT 엔진에 맡길 수 있습니다.

4.2 DLT의 핵심: CDC (Change Data Capture) 자동화

DLT의 가장 강력한 기능 중 하나는 원본 데이터의 변경(Insert, Update, Delete)을 자동으로 감지하고 Silver 테이블에 반영하는 `dlt.apply_changes()`입니다.

이 기능을 사용하면 복잡한 Merge/Upsert 로직을 직접 코딩할 필요가 없습니다.

```

1 import dlt
2 from pyspark.sql.functions import col, expr
3
4 # 소스데이터경로예    (: S3, ADLS)
5 SOURCE_PATH = "/path/to/source-data/"
6
7 # 1. Bronze: 로Autoloader 원시데이터스트리밍      (JSON, CSV 등)
8 @dlt.table(
9     comment="로Autoloader 원시트랜잭션수집"
10 )
11 def bronze_transactions():
12     return (
13         spark.readStream
14             .format("cloudFiles") # Autoloader 사용
15             .option("cloudFiles.format", "json")
16             .load(SOURCE_PATH)
17     )
18
19 # 2. View: Bronze 데이터를에 Silver 적용하기전정제
20 # 뷰는 ( 디스크에저장되지않고인메모리에서사용됨 )
21 @dlt.view(

```

```

22     comment="Bronze 데이터정제및시퀀스컬럼추가      "
23 )
24 def clean_transactions():
25     return (
26         dlt.read_stream("bronze_transactions")
27         .selectExpr(
28             "transaction_id",
29             "CAST(amount AS DOUBLE)",
30             "operation_type", # 'INSERT', 'UPDATE', 'DELETE'
31             "CAST(update_timestamp AS LONG) as sequence" # 변경순서
32         )
33     )
34
35 # 3. Silver: CDC (Change Data Capture) 적용
36 @dlt.table(
37     comment="정제된 트랜잭션데이터를로 CDC에 Silver 적용"
38 )
39 def silver_transactions():
40     dlt.apply_changes(
41         target = "silver_transactions", # 최종대상테이블
42         source = dlt.read_stream("clean_transactions"), # 소스정제된 (뷰)
43         keys = ["transaction_id"], # 레코드의 고유키 (PK)
44         sequence_by = col("sequence"), # 변경순서식별컬럼
45         apply_as_deletes = expr("operation_type = 'DELETE'") # 삭제조건
46         # If operation_type 'DELETE'인 경우, 해당키를 가진 레코드를 삭제
47         # INSERT,는 UPDATE 자동으로 처리됨 (SCD Type 1)
48     )

```

Listing 2: DLT를 사용한 Bronze → Silver CDC 파이프라인 예시

dlt.apply_changes() 상세

- **target:** 데이터가 최종적으로 저장될 테이블 이름.
- **source:** 변경 사항이 포함된 스트리밍 데이터프레임.
- **keys:** 레코드를 고유하게 식별하는 키(PK) 리스트.
- **sequence_by:** 변경 순서를 나타내는 컬럼 (예: 타임스탬프). 동일한 키의 레코드가 여러 개일 경우, 이 값이 가장 높은(최신) 레코드를 적용합니다.
- **apply_as_deletes:** 이 조건이 참(True)이 되는 레코드는 'DELETE'로 처리됩니다.

5 4부: 실습 오류 및 해결 (Checkpoint Q&A)

과제 수행 중 흔히 발생하는 ‘checkpointLocation’ 오류는 스트리밍 파이프라인의 상태 관리와 밀접한 관련이 있습니다.

5.1 사례: 스트리밍 체크포인트 오류

Q: 증상 (Symptom)

과제 3번에서 `spark.readStream.format("delta").load(...).writeStream.option("checkpointLocation", ...).start()` 코드를 실행할 때, 체크포인트 위치(`checkpointLocation`) 관련 오류가 계속 발생합니다. 경로를 바꿔봐도 해결되지 않습니다.

원인 분석 (Cause Analysis)

스트리밍 쿼리는 ‘체크포인트(Checkpoint)’에 자신이 어디까지 데이터를 처리했는지 ‘상태(State)’를 기록합니다. 이 오류는 주로 두 가지 이유로 발생합니다.

- 체크포인트 충돌:** 이전에 실행했던 스트리밍 쿼리의 체크포인트가 남아있는 상태에서, 파이프라인 코드(로직)를 수정하거나 소스 데이터를 삭제한 후 다시 실행하면, 이전 상태와 현재 상태가 충돌하여 오류가 발생합니다.
- 잘못된 경로 사용:** (Databricks 환경) `dbfs:/...` 같은 경로 대신, Unity Catalog (UC)에서 관리하는 `/Volumes/...` 경로를 사용해야 할 수 있습니다.

5.2 해결 방안

▣ 예제: title

스트리밍 쿼리를 새로 시작하기 전에, 이전 체크포인트 디렉토리를 수동으로 삭제하여 상태를 초기화합니다.

```

1 # 체크포인트 경로 정의
2 checkpoint_path = "/path/to/my/checkpoint"
3
4 # (1) 스트리밍 쿼리 중지 이미 ( 실행 중이라면 )
5 # for s in spark.streams.active:
6 #     s.stop()
7
8 # (2) 체크포인트 디렉토리 강제 삭제
9 # 주의 (: 실제 운영 환경에서는 복구 불가능하므로 신중해야 함)
10 dbutils.fs.rm(checkpoint_path, recurse=True)
11
12 # (3) 스트리밍 쿼리 다시 시작
13 (
14     spark.readStream
15         .format("delta")
16         .load(...)
17         .writeStream
18         .option("checkpointLocation", checkpoint_path)

```

```
19     .table(...)  
20 )
```

Listing 3: 스트리밍 시작 전 체크포인트 강제 삭제

해결책 2: DLT/Lakeflow 사용 (자동 방식)

근본적인 해결책: DLT가 상태 관리를 자동화합니다.

이러한 수동 체크포인트 관리는 번거롭고 오류를 유발하기 쉽습니다.

DLT/Lakeflow는 파이프라인의 상태를 엔진이 직접 관리합니다. 만약 파이프라인을 처음부터 다시 실행하고 싶다면, DLT 실행 시 ”**Full Refresh (전체 새로 고침)**” 옵션을 선택하면 됩니다.

”Full Refresh”는 DLT 엔진이 자동으로 모든 이전 상태와 체크포인트를 삭제하고, 소스 데이터를 처음부터 다시 처리하도록 지시합니다. (수동 dbutils.fs.rm 블필요)

6 5부: 요약 및 점검

6.1 운영 파이프라인 체크리스트

- ✓ **요구사항:** 기능적 요구사항(What)과 비기능적 요구사항(How - 성능, 가용성, 비용)이 명확하게 정의되었는가?
- ✓ **아키텍처:** Medallion 아키텍처(Bronze/Silver/Gold)가 설계되었는가?
- ✓ **품질:** 데이터 품질 규칙(Expectations)이 정의되었고, 위반 시 처리(Fail/Drop/Log) 전략이 수립되었는가?
- ✓ **오류 처리:** 손상된 레코드(Corrupt records) 및 파이프라인 실패 시 복구/재시도(Retry) 전략이 있는가?
- ✓ **거버넌스:** 데이터 접근 제어, PII 마스킹, 데이터 계보(Lineage)가 고려되었는가?
- ✓ **최적화:** 스토리지와 컴퓨터가 분리되었는가? 워크로드에 맞게 탄력적(Elastic)으로 자원이 조절되는가?
- ✓ **자동화:** 수동 개입 없이 파이프라인이 스케줄링(Orchestration)되고 자동 실행되는가?
- ✓ **모니터링:** 파이프라인 상태, 성능, 데이터 품질 메트릭이 모니터링되고 알림이 설정되었는가?

6.2 FAQ (자주 묻는 질문)

Q: ETL과 ELT 중 무엇을 사용해야 하나요?

A: 상황에 따라 다릅니다. 원본 데이터가 이미 잘 구조화되어 있고(예: RDBMS) 스키마가 명확하면 전통적인 **ETL**(추출 → 변환 → 적재)이 효율적일 수 있습니다.

하지만 대부분의 현대 데이터(비정형, 반정형)는 스키마가 불명확하므로, 일단 원본(Bronze)을 적재(L)한 후 Lakehouse의 강력한 컴퓨팅 파워를 이용해 변환(T)하는 **ELT** 방식이 더 유연하고 권장됩니다.

Q: DLT는 SQL과 Python 중 무엇으로 작성해야 하나요?

A: 둘 다 가능하며, 동일한 파이프라인 내에서 혼용할 수도 있습니다. 간단한 변환이나 SQL에 익숙한 분석가들은 SQL을, 복잡한 로직, ML 전처리, Python 라이브러리 활용이 필요하면 Python을 사용합니다.

Q: 파이프라인은 항상 처음부터 다시 실행해야 하나요?

A: 아닙니다. 그럴 필요가 없습니다. DLT와 같은 현대적 파이프라인은 증분 처리(Incremental Processing)를 기본으로 합니다. 즉, 마지막 실행 이후 새롭게 추가되거나 변경된 데이터만 처리하여 비용과 시간을 획기적으로 줄입니다.

”Full Refresh“는 스키마가 크게 변경되거나 로직을 근본적으로 수정했을 때만 예외적으로 사용합니다.

Q: 확장성(Scalability)과 탄력성(Elasticity)의 차이는 무엇인가요?

A:

- **확장성(Scalability):** 시스템의 최대 용량을 늘릴 수 있는 능력입니다. (예: 100 명용 → 1000 명용 시스템으로 업그레이드)
- **탄력성(Elasticity):** 현재 수요에 맞춰 자원을 동적으로 늘리거나 줄이는 능력입니다. (예: 낮에 1000 명이 쓰면 1000 명용, 밤에 10 명만 쓰면 10 명용으로 자동 축소)

탄력성은 클라우드 환경에서 비용을 최적화하는 핵심 개념입니다. Databricks의 Serverless 기능이 대표

적인 예입니다.

6.3 빠르게 훑어보기 (1페이지 요약)

PoC (프로토타입) vs Production (운영)

PoC: 수동 실행, 불안정, 품질 보장 X, 1회성 분석 (예: 노트북)

↓ Operationalizing ↓

Production: 자동화, 신뢰성, 품질 보장(Quality), 확장성, 모니터링 (예: DLT)

요구사항 (Requirements)

- 기능적 (Functional): 무엇을? (기능)
- 비기능적 (Non-Func.): 어떻게? (성능, 품질, 비용) ← 놓치기 쉬움!

Medallion 아키텍처 (품질 향상)

Bronze (원시) → Silver (정제/검증) → Gold (집계/최종 제품)

6대 핵심 원칙 (Guiding Principles)

1. 데이터 제품 (Data-as-Products)
2. 사일로 제거 (No Silos)
3. 셀프 서비스 (Self-Service) + 거버넌스
4. 전사적 거버넌스 (Governance)
5. 개방형 포맷 (Open Formats)
6. 확장 및 최적화 (Scale & Cost)

DLT (Delta Live Tables) 핵심 기능

- 선언형 (Declarative): "What"만 정의 (엔진이 "How" 처리)
- CDC 자동화: `dlt.apply_changes()`
- 품질 자동화: `@dlt.expect_...` (Expectations)
- 상태 관리: 체크포인트, Full Refresh 자동화

확장성 (Scaling)

- 수평적 확장 (Scale Out): 서버 대수 추가 (권장)
- 수직적 확장 (Scale Up): 서버 사양 업그레이드
- 핵심: 스토리지(창고)와 컴퓨트(공장)는 반드시 분리!

A 부록: 과제 및 공지사항

다음은 강의 초반에 공지된 행정 사항 요약입니다.

- **Assignment 1 (Spark):** 채점이 완료되었으며 성적이 배포되었습니다. 성적을 받지 못한 경우 교수진에게 연락 바랍니다.
- **Assignment 2 (ETL/EDA):** 채점이 진행 중이며, 곧 완료될 예정입니다.
- **Assignment 3 (Batch & Streaming):** 과제가 공개되었으며, 현재 진행 중이어야 합니다.
- **Quiz 1:** 퀴즈가 이번 주에 공개되었습니다. (시간 제한이 있는 퀴즈가 아니라 며칠 또는 1주일 정도의 기간을 두고 푸는 방식입니다.)
- **Case Study 1 (Data Architectures):**
 - 이번 주 말(목요일 또는 금요일)에 공개될 예정입니다.
 - 팀 프로젝트로 진행되며, 팀은 Canvas의 그룹 도구를 통해 랜덤으로 배정됩니다.
 - 과제 완료까지는 최소 2주의 시간이 주어질 것입니다.
 - 과제 수행을 위한 템플릿(Deck)이 제공될 예정입니다.
 - (참고: 여러 번의 케이스 스터디가 있을 예정이며, 이는 다른 학생들과 네트워킹할 기회입니다.)
- **질문:** 질문은 **Slack**을 이용하는 것이 가장 좋습니다. Slack을 통해 질문과 답변을 공유하면 모든 학생이 도움을 받을 수 있습니다. 개인적인 질문은 교수 또는 TA에게 직접 연락 가능합니다.

December 10, 2025

■ 강의명: CSCI E-103: 재현 가능한 머신러닝

■ 주차: Lecture 08

■ 교수명: Anindita Mahapatra

Eric Gieseke

■ 목적: Lecture 08의 핵심 개념 학습

Contents

1 개요: 이 노트의 핵심	2
2 주요 용어 정리	3
3 문제: 왜 ML 재현성은 어려운가?	5
3.1 숨겨진 기술 부채 (Hidden Technical Debt)	5
3.2 단편화된 생태계 (Fragmented Ecosystem)	5
3.3 모델은 '살아있는 자산'이다	5
4 ML 수명 주기와 다양한 역할	7
4.1 ML 수명 주기 (Lifecycle)	7
4.2 프로젝트의 세 가지 핵심 역할	7
5 핵심 개념 1: 피처 엔지니어링과 피처 스토어	9
5.1 피처 엔지니어링 (Feature Engineering)	9
5.2 피처 스토어 (Feature Store)	9
6 핵심 개념 2: ML 파이프라인 (Transformers vs. Estimators)	11
7 핵심 개념 3: 모델 드리프트 (Model Drift)	12
7.1 모델은 왜 성능이 저하되는가?	12
7.2 드리프트의 4가지 유형	12
8 해결책: MLOps와 MLflow	13
8.1 MLflow: MLOps를 위한 만능 도구	13
8.1.1 1. MLflow Tracking (트래킹)	13
8.1.2 2. MLflow Projects (프로젝트)	13
8.1.3 3. MLflow Models (모델)	14

8.1.4 4. MLflow Model Registry (모델 레지스트리)	14
9 실습: MLflow 사용하기 (주요 코드 해설)	15
9.1 실습 1: XGBoost + Optuna (하이퍼파라미터 튜닝)	15
9.2 실습 2: Scikit-learn + Hyperopt (중요 수정 사항)	16
10 학습 체크리스트	18
11 한 페이지 요약 (Quick Look)	19

1 개요: 이 노트의 핵심

▣ 핵심 요약

이 문서는 머신러닝(ML) 프로젝트의 '재현성(Reproducibility)'을 달성하는 방법을 다룹니다.

재현성이란, 동일한 데이터와 코드로 언제든 동일한 모델과 결과를 만들 수 있음을 의미합니다. 하지만 ML 프로젝트는 코드 외에도 데이터, 환경, 파라미터 등 수많은 변수로 인해 재현이 극히 어렵습니다. ('숨겨진 기술 부채' 문제)

이 노트는 재현성을 가로막는 도전 과제(모델 드리프트, 단편화된 도구)를 살펴보고, 이에 대한 해결책으로 MLOps(엠엘옵스) 문화와 MLflow(엠엘플로우)라는 강력한 도구를 소개합니다.

특히 MLflow의 4가지 핵심 구성요소(Tracking, Projects, Models, Registry)가 어떻게 실험 관리, 환경 패키징, 모델 배포, 버전 관리를 자동화하여 이 모든 문제를 해결하는지 상세히 해설합니다.

▣ 예제: title

이 노트를 가장 효과적으로 학습하기 위한 추천 순서입니다.

1. 용어 정리(2장): 먼저 핵심 용어들의 '쉬운 정의'를 훑어봅니다.
2. 문제 인식(3, 4장): ML 재현성이 '왜' 어렵고 '왜' 중요한지(숨겨진 기술 부채, 드리프트) 공감합니다.
3. 핵심 개념(5, 6, 7장): 문제 해결에 필요한 피처 스토어, ML 파이프라인 등의 개념을 이해합니다.
4. 솔루션(8, 9장): '어떻게' 해결하는지 MLflow의 4가지 기능을 중심으로 학습하고, 실습 코드를 살펴봅니다.
5. 복습(10, 11장): 체크리스트와 한 페이지 요약으로 스스로 점검합니다.

2 주요 용어 정리

본격적인 학습에 앞서, 이 문서에서 반복적으로 사용되는 핵심 용어들을 쉬운 설명과 함께 정리합니다.

Table 1: 재현 가능한 ML 핵심 용어

용어	쉬운 설명 (직관적 이해)	원어 (English)	비고
재 현 성 (Reproducibility)	”요리 레시피(코드, 데이터, 환경)만 있으면 언제 누가 따라 해도 똑같은 맛(결과)의 요리(모델)가 나오는 상태.”	Reproducibility	ML에서는 극히 어려움.
숨겨진 기술 부채 (Hidden Technical Debt)	”ML 시스템은 ‘ML 코드’라는 작은 얼음덩이 밑에 ‘데이터 관리, 모니터링, 인프라’라는 거대한 빙산이 숨어 있다는 개념.”	Hidden Technical Debt	구글 논문에서 유래.
데 이 터 엔지니어 (Data Engineer)	”데이터라는 ‘원자재’를 수집, 정제, 저장하고 ‘공급망(파이프라인)’을 구축/관리하는 인프라 설계자.”	Data Engineer	(DE)
데 이 터 사이언티스트 (Data Scientist)	”정제된 ‘원자재(데이터)’를 분석하고 ‘피처’를 추출하여 ‘제품(모델)’을 개발하고 통찰력을 도출하는 연금술사.”	Data Scientist	(DS)
ML 엔지니어 (ML Engineer)	”개발된 ‘제품(모델)’을 ‘대량 생산(배포)’하고, ‘품질 관리(모니터링)’ 하며, ‘자동화 공정(MLOps)’을 구축하는 전문가.”	ML Engineer	(MLE)
MLOps (엠엘옵스)	”DevOps(개발+운영)의 ML 버전. 모델 개발(DS)과 배포/운영(MLE)을 자동화된 파이프라인으로 통합하는 문화이자 기술.”	ML Operations	DataOps + ModelOps
피처 스토어 (Feature Store)	”모델 훈련과 실시간 추론에 사용될 ‘피처(특성)’들을 중앙에서 관리, 저장, 공유하는 전용 데이터베이스.”	Feature Store	재사용성, 일관성 확보.
모 델 드 리프트 (Model Drift)	”시간이 흘러 현실 세계의 데이터 패턴이 변하면서, 과거 데이터로 학습한 모델의 성능이 저하되는 현상.”	Model Drift	모델은 ‘살아있는 자산’.

Table 1 – 이어서

용어	쉬운 설명 (직관적 이해)	원어 (English)	비고
----	----------------	--------------	----

MLflow (엠 엘 플로우)	”단편화된 ML 작업을 하나로 통합 관리해주는 'ML 프로젝트 만능 도구 (Swiss Army Knife)'. 오픈소스 프레임워크.”	MLflow	재현성의 핵심 솔루션.
MLflow Tracking	”모든 ML 실험의 '실험 노트'. 사용한 파라미터, 성능(지표), 산출물(모델 파일, 그래프)을 자동으로 기록.”	MLflow Tracking	”내가 뭘 했는지” 추적.
MLflow Projects	”실험 환경(코드 + 라이브러리)을 '밀키트'처럼 통째로 패키징하여, 어디서든 동일한 환경을 재현하게 함.”	MLflow Projects	‘requirements.txt’ 포함.
MLflow Models	”훈련된 모델을 '표준 규격'으로 저장. '파이썬 함수', '스파크 UDF' 등 다양한 '맛(Flavor)'으로 서빙 가능.”	MLflow Models	배포 유연성.
MLflow Model Registry	”완성된 모델을 등록하고 '버전 관리'하는 '모델의 Git 저장소'. 'Staging', 'Production' 단계를 관리.”	MLflow Model Registry	모델 거버넌스.
하이퍼파라미터 튜닝	”모델이 '학습'하는 값(가중치)이 아닌, 개발자가 '설정'하는 값(e.g., 학습률, 트리 깊이)의 최적 조합을 찾는 과정.”	Hyperparameter Tuning	‘Optuna’, ‘Hyperopt’ 사용.
A/B 테스팅 (A/B Testing)	”기존 모델(챔피언)과 새 모델(도전자)에 실제 트래픽을 나눠 보내어, 어떤 모델이 더 나은 성과를 내는지 비교 검증.”	A/B Testing	e.g., 80% vs 20%
Medallion 아키텍처	”데이터를 'Bronze(원본)' → 'Silver(정제)' → 'Gold(집계)' 3단계로 정제하며 가치를 높이는 데이터 관리 패턴.”	Medallion Architecture	Silver는 ML용, Gold는 BI용.

Table 1 – 이어서

용어	쉬운 설명 (직관적 이해)	원어 (English)	비고
----	----------------	--------------	----

3 문제: 왜 ML 재현성은 어려운가?

3.1 숨겨진 기술 부채 (Hidden Technical Debt)

초심자들은 흔히 'ML = 모델링 코드'라고 생각하지만, 이는 큰 오해입니다. 유명한 "숨겨진 기술 부채" 개념에 따르면, 실제 ML 시스템에서 순수 ML 코드가 차지하는 비중은 매우 작습니다.

오해: ML은 코드다 vs. 진실: ML은 시스템이다

잘못된 직관: "모델 성능을 높이려면 복잡한 알고리즘(ML 코드)을 짜는 것이 전부다."

올바른 이해: "실제 ML 시스템의 90% 이상은 ML 코드가 아니라, 그 코드를 둘러싼 방대한 인프라다." 이 인프라(빙산의 수면 아래)를 관리하지 못하면 시스템 전체가 무너집니다.

ML 코드를 둘러싼 인프라 (빙산의 아랫부분):

- **Data Collection:** 데이터 수집
- **Data Verification:** 데이터 검증 (e.g., 데이터 품질)
- **Configuration:** 수많은 설정 값 (e.g., 어떤 피처를 쓸지, 파라미터는 뭔지)
- **Feature Extraction:** 피처 추출 및 공학
- **Infrastructure:** 자원 관리 (e.g., CPU, GPU, 분산 처리)
- **Monitoring:** 모델 성능 및 데이터 드리프트 감시
- **Serving:** 모델을 실제 서비스로 배포

이것이 재현성과 무슨 관계인가? 모델을 재현하려면, 그 작은 'ML 코드' 뿐만 아니라 이 모든 '주변 인프라'의 상태(설정 값, 데이터 버전, 라이브러리 버전 등)를 정확히 동일하게 복원해야 합니다. 이는 사실상 불가능에 가깝습니다.

3.2 단편화된 생태계 (Fragmented Ecosystem)

ML 프로젝트는 하나의 도구로 끝나지 않습니다. 데이터 전처리, 훈련, 튜닝, 배포에 사용되는 언어와 프레임워크가 모두 다릅니다.

- **언어:** Python, R, SQL, Spark
- **프레임워크:** Scikit-learn, PyTorch, TensorFlow, XGBoost
- **배포 환경:** Docker, Kubernetes, Batch, Streaming

이처럼 파편화된 도구들을 수동으로 연결하다 보면, "내 노트북에서는 잘 됐는데, 동료 컴퓨터나 서버에서는 왜 안 되지?"라는 '의존성 지옥(Dependency Hell)'에 빠지게 됩니다. 각 도구의 버전이 조금만 달라져도 결과가 바뀌거나 오류가 발생합니다.

3.3 모델은 '살아있는 자산'이다

모델은 한 번 만들고 끝나는 '조각상'이 아닙니다. 시간이 지남에 따라 성능이 변하는 '살아있는 자산'이며, 지속적인 '건강검진(모니터링)'과 'TLC(Tender Loving Care, 애정 어린 돌봄)'가 필요합니다.

□ 예제: title

모델을 훈련시키는 것은 '운전면허 시험(과거 데이터)'에 합격한 것과 같습니다. 하지만 면허를 땠다고 해서 실제 '도로(현실 세계)'에서 완벽하게 운전할 수 있는 것은 아닙니다.

시간이 지나면 '새로운 교통 법규(패턴 변화)'가 생기고, '못 보던 형태의 교차로(신규 데이터)'가 등장합니다. 과거의 지식(훈련된 모델)만으로는 사고(잘못된 예측)를 낼 수 있습니다. 이것이 바로 모델 드리프트입니다.

결국, 6개월 전의 '최고 성능 모델'을 오늘날 똑같이 재현하는 것 자체가 무의미 할 수 있습니다. 중요한 것은 "6개월 전에 '어떻게' 그 모델을 만들었는지" 그 과정을 정확히 추적(Tracking)하고, "오늘의 데이터로 '빠르게' 재훈련"하여 새 모델을 배포할 수 있는 자동화된 파이프라인입니다.

이것이 바로 MLOps와 MLflow가 해결하려는 핵심 문제입니다.

4 ML 수명 주기와 다양한 역할

ML 프로젝트는 여러 전문가의 협업으로 이루어집니다. 각자의 역할과 전체 흐름을 이해하는 것이 중요합니다.

4.1 ML 수명 주기 (Lifecycle)

모든 ML 프로젝트는 다음과 같은 단계를 반복하는 순환 고리(Loop)입니다.

▣ 핵심 정보

ML 프로젝트의 6단계 수명 주기

순방향 프로세스:

1. 원본 데이터 (Raw Data) →
2. ETL (정제) →
3. 피처화 (Featurize) →
4. 훈련 (Train) →
5. 서빙/추론 (Serve) →
6. 모니터링 (Monitor)

피드백 루프: 모니터링 결과 성능 저하 감지 시 → 1단계로 돌아가 재훈련

1. 데이터 준비 (Raw Data → ETL): 원본 데이터를 수집하고 정제합니다.
2. 피처 엔지니어링 (Featurize): 정제된 데이터로 모델이 학습할 '특성(Feature)'을 만듭니다.
3. 모델 훈련 (Train): 피처를 입력 받아 모델을 학습시킵니다. (전체 과정 중 아주 작은 부분!)
4. 모델 서빙 (Serve/Inference): 훈련된 모델을 배포하여, 새로운 데이터에 대한 '예측(추론)'을 제공합니다. (Batch, 실시간, HTTP 등)
5. 모니터링 (Monitor): 모델의 성능이 잘 유지되는지, 데이터가 변하지는 않았는지(Drift) 감시합니다.
6. 피드백 루프 (Feedback): 모니터링 결과, 성능이 저하되면 1, 2, 3단계로 돌아가 새 데이터로 모델을 재훈련합니다.

4.2 프로젝트의 세 가지 핵심 역할

이 수명 주기는 보통 세 가지 주요 역할의 협업으로 완성됩니다.

Table 2: 데이터 프로젝트의 3가지 핵심 역할

역할	데이터 엔지니어 (DE)	데이터 사이언티스트 (DS)	ML 엔지니어 (MLE)
핵심 임무	데이터 인프라 구축	비즈니스 통찰력 도출	모델의 안정적인 배포/운영
주요 작업	- 데이터 수집 (Ingest) - 데이터 저장 (Store) - 데이터 파이프라인 (ETL) - 데이터 정제 (Curate)	- 피처 추출 (Extract Features) - 모델 코딩 (Code) - 모델 훈련 (Train) - 모델 검증 (Validate)	- 모델 평가 (Evaluate) - 모델 패키징 (Package) - 모델 배포 (Deploy / Serve) - MLOps 파이프라인 구축
비유	원자재 공급 및 공장 설계	시제품(Prototype) 개발	대량 생산 라인 구축/운영

협업의 어려움 (Silo 문제)

전통적으로 이 세 역할은 분리되어(Silo) 일했습니다.

- DS가 노트북(Jupyter)에서 모델을 개발 → MLE에게 코드를 전달

- MLE는 그 코드를 서버 환경(e.g., Docker)에서 다시 작성

- 이 과정에서 버전이 꼬이고, 환경이 달라 결과가 재현되지 않으며, 배포에 수개월이 소요됩니다.

MLOps와 **MLflow**는 이 장벽을 허물고, DS가 개발한 모델을 '그대로', '빠르게' 배포할 수 있도록 돕습니다.

5 핵심 개념 1: 피처 엔지니어링과 피처 스토어

5.1 피처 엔지니어링 (Feature Engineering)

- **한 줄 요약:** 데이터에 대한 도메인 지식을 사용해, ML 알고리즘이 더 잘 학습할 수 있도록 데이터를 '가공'하는 과정입니다.
- **직관적 비유:** ”날고기(Raw Data)를 그대로 먹을 순 없습니다. 요리사(DS)가 먹기 좋게 손질하고, 양념하고(Feature Engineering), ’스테이크(Feature)’로 만드는 것과 같습니다.”
- **기술적 설명:** 원본 데이터 컬럼을 그대로 사용하기보다, 특정 의미를 갖도록 집계(Aggregate)하거나 변환하는 작업을 말합니다.

□ 예제: title

원본 데이터 (Raw Data):

- 고객 ID, 상품 명, 구매 일시, 구매 액

가공된 피처 (Features):

- ‘avg_{transaction}_amount‘ : ' '()
- ‘purchase_frequency₇days‘ : ' 7 '()
- ‘weekday_orweekend‘ : ' 'vs' ' ()
- ‘weather_on_purchase‘ : '()

모델은 '구매 액' 자체보다 '평균 거래액'이나 '최근 구매 빈도' 같은 가공된 피처로부터 훨씬 더 많은 정보를 학습할 수 있습니다.

Andrew Ng의 통찰

”응용 머신러닝은 기본적으로 피처 엔지니어링이다.”

(Applied machine learning is basically feature engineering.)

이는 모델의 성능을 결정하는 가장 중요한 작업이 복잡한 알고리즘을 선택하는 것이 아니라, 데이터를 얼마나 의미 있는 피처로 잘 만드느냐에 달려 있음을 의미합니다.

5.2 피처 스토어 (Feature Store)

피처 엔지니어링은 '고객 7일간 평균 구매액'처럼 컴퓨팅 비용이 매우 비싼 작업을 포함합니다. 또한, 여기서 심각한 재현성 문제가 발생합니다.

치명적 문제: 훈련-서빙 스蹊 (Training-Serving Skew)

증상: ”모델을 훈련시킬 땐(Offline) 성능이 99%였는데, 실제 서비스(Online)에 배포하니 60%로 떨어졌다!”

원인:

- 훈련 시: ‘SELECT AVG(purchase) ...‘ (SQL로 느긋하게 배치 계산)
- 서빙 시: ‘feature = (sum_val + new_val)/(n + 1)‘(Python)

두 계산 로직이 미묘하게 다르거나(e.g., 반올림, 누락 값 처리), 데이터의 시점이 달라 동일한 피처가 아니게 됩니다. 이 미세한 차이가 모델의 성능을 급격히 저하시킵니다.

피처 스토어(Feature Store)는 이 문제를 해결합니다.

피처 스토어의 2가지 핵심 역할

피처 스토어는 피처를 위한 중앙 저장소(Repository)입니다.

1. 일관성 보장 (Consistency): 피처를 단 한 번만 계산하여 저장소에 저장합니다. 모델 훈련 시와 모델 서빙 시에 정확히 동일한 피처를 가져다 쓰게 하여 '훈련-서빙 스큐'를 원천 봉쇄합니다.
2. 재사용성 및 효율성 (Reusability & Efficiency): A 팀이 비싼 돈 들여 계산한 '유저 프로필 피처'를 B 팀, C 팀도 즉시 가져다 쓸 수 있습니다. 중복 계산을 방지하고, 모든 팀이 검증된 피처를 공유하게 됩니다.

6 핵심 개념 2: ML 파이프라인 (Transformers vs. Estimators)

ML 작업을 코드로 구현할 때, 특히 Spark ML 같은 프레임워크에서는 '파이프라인' 개념을 사용합니다. 이때 초심자들이 가장 혼동하는 두 가지 구성요소가 **Transformer**와 **Estimator**입니다.

□ 예제: title

데이터(재료)가 컨베이어 벨트를 따라 이동하며 요리(모델)가 완성되는 과정을 상상해 보세요.

- **Transformer (변환기)**: 재료를 '손질' 하는 기계. (e.g., 'StringIndexer' = "양파 껍질 까기", 'StandardScaler' = "재료 무게 맞추기")
- **Estimator (추정기)**: 재료를 '학습' 하여 '레시피(모델)'를 만드는 기계. (e.g., 'LinearRegression' = "재료 조합을 학습해 최고의 스테이크 레시피 만들기")

이 둘의 가장 결정적인 차이는 '학습(Learning)'의 유무입니다.

Table 3: Transformer vs. Estimator 비교

특징	Transformer (변환기)
목적	데이터 변환 (Preprocessing)
핵심 메서드	.transform()
입력 → 출력	DataFrame → DataFrame
설명	데이터를 입력받아 규칙에 따라 변환된 데이터를 반환합니다.
예시	- 'StringIndexer' (문자 → 숫자) - 'OneHotEncoder' (숫자 → 벡터) - 'StandardScaler' (정규화) - 'VectorAssembler'

□ 핵심 요약

중요한 점은, **Estimator**의 `.fit()` 메서드를 실행하면 그 결과물로 **Model**이 나온다는 것입니다. 그리고 이 **Model**은 그 자체로 **Transformer**입니다. (새로운 데이터에 대해 `.transform()` 또는 `.predict()`를 수행할 수 있으므로)
즉, **Estimator**는 '모델을 만드는 객체'이고, **Transformer**는 '데이터를 변환하는 객체'입니다.

7 핵심 개념 3: 모델 드리프트 (Model Drift)

7.1 모델은 왜 성능이 저하되는가?

모델 드리프트(Model Drift)는 ”배포된 모델의 성능이 시간이 지남에 따라 저하되는 현상”을 말합니다. 모델은 ’오래된(stale)’ 데이터로 학습했기 때문에, ’현재(fresh)’의 데이터 패턴을 따라가지 못하게 됩니다.

우리의 목표는 모델 성능이 완전히 나빠지기(벼랑 끝) 전에, 성능 저하의 ’징후(inflexion)’를 미리 감지하고 모델을 ’재훈련(refresh)’하는 것입니다.

7.2 드리프트의 4가지 유형

드리프트가 발생했을 때, ”왜?” 성능이 떨어졌는지 원인을 파악하는 것이 중요합니다.

Table 4: 모델 드리프트의 4가지 유형과 대응

드리프트 유형	설명 (무엇이 변했는가?)	대응 전략
Feature Drift (피처 드리프트)	입력 데이터(X)의 분포가 변했습니다. 예: 새로운 카테고리의 상품이 등장하여 ’카테고리’ 피처의 분포가 바뀜.	피처 생성 프로세스 점검, 새로운 데이터로 재훈련
Label Drift (레이블 드리프트)	정답 데이터(Y)의 분포가 변했습니다. 예: 갑자기 ’사기 거래(정답)’의 비율이 급증함.	레이블 생성 프로세스 점검, 재훈련
Prediction Drift (예측 드리프트)	모델의 예측(Y-hat) 분포가 변했습니다. 예: 모델이 갑자기 모든 거래를 ’정상’으로만 예측하기 시작함.	모델 훈련 과정 점검, 비즈니스 영향도 평가
Concept Drift (개념 드리프트)	X와 Y의 ’관계’ 자체가 변했습니다. (가장 심각) 예: COVID-19 발생. 이전에는 ’주말’과 ’매장 방문’이 강한 양의 관계였으나, 팬데믹 이후 그 관계가 완전히 깨짐.	단순 재훈련으로 해결 불가. 새로운 피처 엔지니어링, 근본적인 모델 재설계 필요.

이러한 드리프트를 감지하기 위해 지속적인 모니터링이 필수적이며, 드리프트가 감지되면 자동으로 재훈련 파이프라인이 실행되도록 하는 것이 MLOps의 핵심 목표입니다.

8 해결책: MLOps와 MLflow

지금까지 살펴본 모든 문제(숨겨진 기술 부채, 단편화된 생태계, 수동 작업, 모델 드리프트)를 해결하기 위한 문화적, 기술적 접근 방식이 바로 **MLOps**입니다.

- **DevOps** = Development (개발) + Operations (운영) → 코드 빌드/배포 자동화
- **DataOps** = Data (데이터) + Operations (운영) → 데이터 파이프라인 자동화
- **MLOps** = Model (모델) + Operations (운영) → 모델 훈련/배포/모니터링 자동화

MLOps는 이 모든 것을 통합하여, 데이터 수집부터 모델 배포 및 모니터링까지의 전체 수명 주기를 자동화하는 것을 목표로 합니다.

8.1 MLflow: MLOps를 위한 만능 도구

MLflow는 MLOps를 구현하기 위한 가장 인기 있는 오픈소스 프레임워크입니다. 단편화된 ML 수명 주기를 관리하기 위한 4가지 강력한 구성요소를 제공합니다.

MLflow의 4가지 핵심 구성요소

1. **MLflow Tracking** (실험실 노트)
2. **MLflow Projects** (환경 밀키트)
3. **MLflow Models** (표준 규격 모델)
4. **MLflow Model Registry** (모델 Git 저장소)

8.1.1 1. MLflow Tracking (트래킹)

- **문제점:** 데이터 사이언티스트가 수백 번의 실험을 하며 파라미터를 바꿔봅니다. ”어제 돌렸던 그 모델이 성능이 제일 좋았는데... 파라미터가 뭐였더라?” → 기억 상실
- **솔루션:** 모든 실험을 ’실험 노트’에 자동으로 기록합니다.
- **무엇을 기록하는가?**
 - **Parameters** (입력): 실험에 사용한 하이퍼파라미터 (e.g., ‘learning_rate = 0.1’)
 - **Metrics** (출력): 실험 결과(성능 지표) (e.g., ‘RMSE=0.85’)
 - **Artifacts** (산출물): 모델 파일 그 자체, 성능 그래프(PNG), 피쳐 중요도 등
 - **Source** (출처): 이 실험을 실행한 소스 코드(e.g., 노트북 버전)
- **핵심 API:**
 - ‘mlflow.start_run()’ :
 - ‘mlflow.log_param()’ :
 - ‘mlflow.log_metric()’ :
 - ‘mlflow.log_artifact()’ : ()
 - **mlflow.autolog()**: (강력 추천) 이 함수 하나만 호출하면, Scikit-learn, TensorFlow 등이 자동으로 위 모든 것을 기록해줍니다.

8.1.2 2. MLflow Projects (프로젝트)

- **문제점:** ”제 노트북에선 잘 됐는데, 서버에선 라이브러리 버전이 안 맞아서 오류 나요.” → 환경 의존성

- **솔루션:** 코드를 실행하는 데 필요한 '환경'까지 통째로 패키징합니다.
- **어떻게?:** 프로젝트 폴더 안에 MLproject라는 명세 파일을 둡니다.
 - 이 파일은 "이 코드를 실행하려면 'python train.py' 명령을 써야 하고, 'requirements.txt' (또는 'conda.yaml') 파일에 명시된 정확한 라이브러리 버전들이 필요하다"라고 정의합니다.
- **결과:** 'mlflow run <프로젝트 경로>' 명령 하나로, MLflow가 자동으로 격리된 환경(e.g., conda)을 만들고 그 안에서 코드를 실행하여 완벽한 재현성을 보장합니다.

8.1.3 3. MLflow Models (모델)

- **문제점:** "DS가 Scikit-learn으로 만든 모델을 MLE가 Spark 환경에서 서빙하려니 호환이 안 됩니다." → 배포 비호환성
- **솔루션:** 모델을 '표준화된 포맷'으로 저장합니다.
- **어떻게?:** MLflow는 모델을 저장할 때, 다양한 '맛(Flavor)'으로 저장합니다.
 - 'pythonfunction'(pyfunc) : 가장 중요한 맛. Python (wrapping) .
 - 'sklearn' : Scikit-learn 네이티브 포맷
 - 'spark' : Spark UDF로 바로 로드할 수 있는 포맷
- **결과:** 모델을 'pyfunc' 맛으로 저장하면, 배포팀은 이 모델이 원래 Scikit-learn인지 PyTorch인지 신경 쓸 필요 없이, 그냥 표준 Python 함수를 호출하듯 모델을 서빙할 수 있습니다.

8.1.4 4. MLflow Model Registry (모델 레지스트리)

- **문제점:** "어떤 모델이 최신 버전이죠? 이 모델은 테스트는 끝난 건가요? 지금 서비스 중인 모델은 뭐죠?" → 모델 관리/거버넌스 부재
- **솔루션:** 모델을 위한 중앙 'Git 저장소' 역할을 합니다.
- **어떻게?:** Tracking Server에 기록된 수백 개의 모델 중, "최고의 모델"을 선택하여 Registry에 등록(Register)합니다.
- **핵심 기능: Stages (단계 관리)**
 - **Staging:** "이 모델을 'Staging' 환경에 배포해서 A/B 테스트를 시작하세요."
 - **Production:** "테스트 결과가 좋으니, 이 모델을 'Production' (실제 서비스)으로 승격시키세요." (e.g., v3 모델)
 - **Archived:** "이 모델(e.g., v2)은 이제 사용하지 않으니 'Archived' (보관) 상태로 변경하세요."
- **결과:** 모델의 전체 수명 주기(개발 → 테스트 → 배포 → 폐기)를 체계적으로 추적하고 관리할 수 있습니다.

9 실습: MLflow 사용하기 (주요 코드 해설)

강의 자료의 실습은 MLflow의 핵심 기능을 실제로 사용하는 방법을 보여줍니다. 여기서는 두 가지 주요 실습(XGBoost, Scikit-learn)의 핵심 코드와 중요 수정 사항을 해설합니다.

9.1 실습 1: XGBoost + Optuna (하이퍼파라미터 튜닝)

이 실습은 하이퍼파라미터 튜닝 라이브러리인 Optuna와 MLflow를 중첩(Nested) 실행하는 방법을 보여줍니다.

```

1 import mlflow
2 import optuna
3 from sklearn.metrics import mean_squared_error
4
5 # 1. 가 MLflow Unity Catalog (UC)를 레지스트리로 사용하도록 설정
6 mlflow.set_registry_uri("databricks-uc")
7
8 # 2. 가 Optuna 최적화할 목적 '함수' 정의
9 def objective(trial):
10     # 3. 각을 trial 중첩된 '(Nested) Run'으로 '기록'
11     # 이것이 핵심입니다 !
12     with mlflow.start_run(nested=True):
13         # 가 Optuna 제안하는 하이퍼파라미터
14         params = {
15             "max_depth": trial.suggest_int("max_depth", 3, 10),
16             "n_estimators": trial.suggest_int("n_estimators", 50, 500),
17             "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.3)
18         }
19
20         # MLflow Tracking: 파라미터 기록
21         mlflow.log_params(params)
22
23         # 모델 훈련 (XGBoost)
24         model = XGBRegressor(**params)
25         model.fit(X_train, y_train)
26
27         # 예측 및 평가
28         preds = model.predict(X_val)
29         rmse = mean_squared_error(y_val, preds, squared=False)
30
31         # MLflow Tracking: 성능 지표 (Metric) 기록
32         mlflow.log_metric("rmse", rmse)
33
34         return rmse # 는 Optuna 이 값을 최소화하려고 시도
35
36 # 4. 부모 Run: 전체 튜닝 '작업을' 기록
37 with mlflow.start_run(run_name="XGBoost Tuning"):
38     # 5. Optuna Study 생성 및 최적화 실행
39     # 주의 (: n_trials 은 =50 시간이 오래 걸리므로 20 줄여서 테스트 )
40     study = optuna.create_study(direction="minimize")

```

```

41     study.optimize(objective, n_trials=20) # 50 -> 20
42
43     # 6. 가장 성능이 좋았던 Trial (Run)의 모델을 찾음
44     best_run = mlflow.search_runs(
45         experiment_ids=study.study_name,
46         order_by=["metrics.rmse ASC"],
47         max_results=1
48     ).iloc[0]
49
50     # 7. 최고의 모델을 Model에 Registry 등록
51     best_model_uri = f"runs:{best_run.run_id}/model"
52     mlflow.register_model(best_model_uri, "xgboost_best_model")

```

Listing 1: Optuna와 MLflow 중첩 실행

이 코드의 의미

- 전체 튜닝 작업(e.g., 20회)이 하나의 '부모 Run'으로 기록됩니다.
- 튜닝의 각 시도(Trial)가 '자식 Run'으로 기록되어, 어떤 파라미터가 어떤 성능을 냈는지 모두 추적할 수 있습니다.
- 튜닝이 끝나면, MLflow의 `search_runs` API를 사용해 가장 좋았던(RMSE가 가장 낮은) 자식 Run 을 찾습니다.
- 이 최고의 모델을 찾아 **Model Registry**에 등록하여 'Staging' 또는 'Production' 단계로 넘길 준비를 합니다.

9.2 실습 2: Scikit-learn + Hyperopt (중요 수정 사항)

이 실습은 또 다른 튜닝 라이브러리인 Hyperopt와 `autolog()` 기능을 사용합니다. 특히, 강의 환경(서버리스)에서 발생하는 문제를 해결하기 위한 중요 수정 사항이 포함됩니다.

실습 환경 중요 수정 사항

강의 자료의 원본 코드는 두 가지 수정이 필요합니다.

1. **카탈로그명 변경:** Unity Catalog 사용 시, `main` 카탈로그가 아닌 본인 소유의 카탈로그 (e.g., `cscie103_catalog`)를 사용해야 합니다.
2. **SparkTrials → Trials:** Hyperopt의 `SparkTrials`는 분산 튜닝을 지원하지만, 일부 서비스 환경에서는 작동하지 않을 수 있습니다. 이 경우, 단일 노드에서 실행되는 기본 `Trials`로 변경해야 합니다.

```

1 import mlflow
2 from hyperopt import fmin, tpe, hp, Trials # SparkTrials 대신 Trials
3 from sklearn.ensemble import GradientBoostingClassifier
4
5 # --- 1. 중요수정사항카탈로그 (설정) ---
6 CATALOG_NAME = "cscie103_catalog"
7 SCHEMA_NAME = "default"
8 mlflow.set_registry_uri("databricks-uc")
9 # 모델저장시 경로를 사용 : f'{CATALOG_NAME}.{SCHEMA_NAME}.my_model'

```

```

10
11 # --- 2. MLflow autolog() 활성화 ---
12 # 이한줄이 log_param, log_metric, 을 log_model 자동으로수행 !
13 mlflow.autolog(log_models=True)
14
15 # 3. Hyperopt 탐색공간 (Search Space) 정의
16 search_space = {
17     'n_estimators': hp.quniform('n_estimators', 50, 500, 10),
18     'learning_rate': hp.loguniform('learning_rate', -3, 0),
19     'max_depth': hp.quniform('max_depth', 2, 5, 1)
20 }
21
22 # 4. Hyperopt 목적함수정의
23 def train_model(params):
24     # autolog()가 켜져있으므로 , start_run()만 호출하면됨
25     with mlflow.start_run(nested=True):
26         model = GradientBoostingClassifier(**params)
27         model.fit(X_train, y_train)
28
29     # ... 평가(로직) ...
30     accuracy = model.score(X_val, y_val)
31
32     # autolog()가 지표를자동으로기록하지만 ,
33     # 가hyperopt 반환받을값을명시적으로지정
34     return {'loss': -accuracy, 'status': 'ok'}
35
36 # --- 5. 중요수정사항 (Trials 객체) ---
37 # SparkTrials 대신로컬 Trials 사용
38 # spark_trials = SparkTrials(parallelism=4) # <- 원본오류 ( 가능 )
39 local_trials = Trials() # <- 수정본안정적 ()
40
41 # 6. Hyperopt 실행 (fmin)
42 best_params = fmin(
43     fn=train_model,
44     space=search_space,
45     algo=tpe.suggest,
46     max_evals=10, # 테스트를위해회만 10 실행
47     trials=local_trials # 수정된 trials 객체전달
48 )
49
50 # 7. autolog() 덕분에모든이 Run 기록되었음
51 # search_runs()를 사용해최고의모델을찾아에 Registry 등록
52 # 이후 ( 과정은실습과 1 유사 )

```

Listing 2: Hyperopt와 MLflow autolog() 사용 (수정 사항 적용)

10 학습 체크리스트

이 노트를 통해 다음 질문들에 답할 수 있는지 스스로 점검해 보세요.

ML의 '재현성'이 왜 중요한지, 그리고 왜 어려운지(e.g., 숨겨진 기술 부채) 설명할 수 있나요?

데이터 엔지니어(DE), 데이터 사이언티스트(DS), ML 엔지니어(MLE)의 역할을 구분하고, 이들의 협업이 왜 중요한지 이해했나요?

'피처 스토어(Feature Store)'가 무엇이며, 이것이 '훈련-서빙 스케ュ(Training-Serving Skew)' 문제를 어떻게 해결하는지 설명할 수 있나요?

ML 파이프라인에서 Transformer와 Estimator의 근본적인 차이(transform vs. fit)를 설명 할 수 있나요?

'모델 드리프트(Model Drift)'가 무엇이며, 4가지 유형(Feature, Label, Prediction, Concept)을 구분할 수 있나요?

MLflow의 4가지 구성요소 각각의 목적을 명확히 설명 할 수 있나요?

Tracking: "모든 실험을 기록하는 실험 노트"

Projects: "환경(라이브러리)까지 패키징하는 밀키트"

Models: "다양한 맛(Flavor)으로 배포하는 표준 규격"

Registry: "모델 버전을 관리(Staging/Prod) 하는 Git 저장소"

MLflow의 autolog() 기능이 개발자에게 어떤 편리함을 주는지 이해했나요?

Model Registry의 'Stage'(e.g., Staging, Production) 개념이 A/B 테스트 및 모델 배포 관리에 왜 유용한지 이해했나요?

11 한 페이지 요약 (Quick Look)

문제: ML의 재현성 위기

1. 숨겨진 기술 부채: ML 코드는 병산의 일각. 데이터 관리, 인프라, 모니터링 등 수면 아래의 거대한 인프라 관리가 더 복잡하고 중요함.
2. 단편화된 생태계: Python, R, Spark, TensorFlow 등 수많은 도구가 단편화되어 “내 컴퓨터에선 됐는데...”라는 ‘의존성 지옥’ 발생.
3. 모델 드리프트: 모델은 ‘살아있는 자산’. 배포 후에도 현실 데이터가 변하면서(Drift) 성능이 계속 저하되므로 지속적인 관리가 필요함.

솔루션: MLOps 문화와 MLflow 도구

MLOps (문화): 데이터, 모델, 운영을 하나로 통합. DataOps (데이터 파이프라인) + ModelOps (모델 훈련/배포) + DevOps (인프라/운영) → 모델 개발부터 배포, 모니터링까지 전 과정을 자동화

MLflow (도구): MLOps를 구현하기 위한 오픈소스 프레임워크. 단편화된 ML 수명 주기를 4가지 구성요소로 통합 관리.

MLflow Component 1: Tracking (실험 노트)

목적: ”내가 뭘 했는지 잊어버리는” 문제 해결.

기능: 모든 실험의 파라미터, 성능 지표, 모델/그래프(Artifacts), 소스 코드를 자동으로 기록. `mlflow.autolog()`로 간편화.

MLflow Component 2: Projects (환경 밀키트)

목적: ”내 PC에선 됐는데...” (의존성 지옥) 문제 해결.

기능: 코드 + `requirements.txt` 등 환경 정보를 통째로 패키징. 어디서든 `mlflow run`으로 동일 환경, 동일 결과 재현.

MLflow Component 3: Models (표준 규격)

목적: ”훈련과 배포 환경이 다른” (호환성) 문제 해결.

기능: 모델을 다양한 ‘맛(Flavor)’ (e.g., `pyfunc`, `spark`)으로 저장. 훈련(Python)과 배포(Spark/Docker)가 달라도 유연하게 서빙.

MLflow Component 4: Model Registry (모델 저장소)

목적: ”지금 서비스 중인 모델이 뭐죠?” (버전 관리) 문제 해결.

기능: 검증된 모델을 ‘등록’하고 ‘버전’ 관리. Stage (‘Staging’ → ‘Production’ → ‘Archived’)를 통해 A/B 테스트 및 릴리즈를 통제.

December 10, 2025

■ 강의명: CSCI E-103: 재현 가능한 머신러닝

■ 주차: Lecture 09

■ 교수명: Anindita Mahapatra

Eric Gieseke

■ 목적: Lecture 09의 핵심 개념 학습

▣ 핵심 요약

이 문서는 머신러닝(ML) 모델을 개발하고 운영하는 전 과정을 다루는 **MLOps (Machine Learning Operations)**의 핵심 개념을 정리합니다. MLOps는 단순히 모델을 만드는 것을 넘어, 신뢰할 수 있고 반복 가능한 방식으로 모델을 배포하고 모니터링하는 프로세스입니다.

이 노트는 MLOps에 참여하는 **다양한 역할**(데이터 엔지니어, 과학자, ML 엔지니어 등)을 정의하고, 고객 이탈 방지 사례를 통해 **3대 핵심 파이프라인(학습, 추론, 모니터링)**을 살펴봅니다. 또한, 모델의 성능을 결정하는 **데이터 중심 AI**의 중요성과 실습을 통한 E2E 파이프라인 구축 과정을 설명합니다.

Contents

1	공지사항 및 주요 일정	2
2	지난 강의 핵심 복습	3
3	핵심 개념: MLOps란 무엇인가?	4
3.1	MLOps의 정의와 필요성	4
3.2	비즈니스 가치 (예: 고객 이탈 방지)	4
4	MLOps의 핵심 구성원 (The "People")	5
4.1	비즈니스 이해관계자 (Business Stakeholder)	5
4.2	데이터 엔지니어 (Data Engineer)	5
4.3	데이터 사이언티스트 (Data Scientist)	5
4.4	머신러닝 엔지니어 (ML Engineer)	6
4.5	데이터 거버넌스 책임자 (Data Governance Officer)	6
5	MLOps의 3대 파이프라인 (The 3 Pipelines)	8

5.1 1. 모델 학습 파이프라인 (Model Training Pipeline)	8
5.2 2. 모델 추론 파이프라인 (Model Inferencing Pipeline)	8
5.3 3. 모델 모니터링 파이프라인 (Model Monitoring Pipeline)	8
6 모델 배포 전략 (ML Deployment Patterns)	10
6.1 1. 모델 배포 (Deploy Models)	10
6.2 2. 코드 배포 (Deploy Code)	10
6.3 코드 배포의 장단점 (Benefits of Deploy Code)	10
7 데이터 중심 AI (Data-Centric AI)	12
7.1 빅데이터 vs. 좋은 데이터 (Big Data vs. Good Data)	12
7.2 왜 데이터 중심 AI가 중요한가?	12
8 MLOps 성숙도 모델 (MLOps Progression)	13
9 모델 튜닝 및 선택 (Model Tuning and Selection)	14
9.1 하이퍼파라미터 튜닝 (Hyperparameter Tuning)	14
9.2 K-겹 교차 검증 (K-Fold Cross Validation)	14
10 실습: E2E MLOps 파이프라인 (Lab Walkthrough)	15
10.1 실습 목표 및 도구	15
10.2 1단계: 데이터 준비 및 피처 엔지니어링	15
10.3 2단계: 모델 학습 및 로깅 (MLflow Tracking)	15
10.4 3단계: 모델 레지스트리 등록 (MLflow Registry)	16
10.5 4단계: 챌린저 모델 검증 (Champion vs. Challenger)	16
10.6 5단계: 배치 추론 (Batch Inference)	16
11 핵심 용어 정리	17
12 빠르게 훑어보기 (1-Page Summary)	18

1 공지사항 및 주요 일정

- **과제 4 (Assignment 4):** 곧 공개될 예정이며, MLflow를 기반으로 합니다. 목요일 세션에서 리뷰할 예정이며, 제출 기한은 약 3주입니다.
- **케이스 스터디 2 (Case Study 2):** 그룹 프로젝트로 진행되며, 추후 공개될 예정입니다. 제출 기한은 약 3주입니다.
 - 케이스 스터디 1과는 다른 새로운 그룹이 배정될 것입니다.
 - 이는 가능한 한 많은 동료 학생들과 교류하고 네트워크를 구축할 기회를 제공하기 위함입니다.
- **기말 프로젝트 (Final Project):** 그룹 프로젝트로 진행되며, 역시 새로운 그룹이 배정됩니다.
- **퀴즈 2 (Quiz 2):** 11월 말경에 공개될 예정입니다.
- **성적 공지:** 퀴즈 1 성적은 마감일로부터 약 2주 이내에 공개될 예정입니다.

2 지난 강의 핵심 복습

MLOps를 본격적으로 배우기 전, 지난 강의에서 다룬 핵심 용어들을 복습합니다. 이 용어들은 MLOps의 기반이 됩니다.

Table 1: *MLOps* 관련 핵심 용어 복습

용어	설명 (초심자 가이드)
모델 드리프트 (Model Drift)	시간이 지남에 따라 모델의 성능이 저하되는 현상입니다. (예: 계절이 바뀌어 작년의 판매 예측 모델이 안 맞게 됨)
MLflow	모델 개발의 전 과정을 관리(추적, 등록, 배포)하는 오픈소스 프레임워크입니다. 모델의 '이력서'를 관리해 준다고 생각할 수 있습니다.
MLflow Tracking	모델 학습 시 사용된 모든 정보(파라미터, 성능 지표, 코드, 데이터 버전 등)를 기록하는 기능입니다.
MLflow Registry	완성된 모델을 등록하고 버전을 관리하는 '모델 카탈로그'입니다. (예: '개발 중', '스테이징', '운영 배포' 상태 관리)
피처 엔지니어링 (Feature Eng.)	원본 데이터(Raw Data)를 모델이 더 잘 학습할 수 있는 유의미한 '특성(Feature)'으로 가공하는 과정입니다. (예: '생년월일'을 '나이'로 변경)
모델 서빙 유형 (Model Serving)	모델을 배포하여 예측값을 제공하는 방식입니다. (1) 배치: 한 번에 대량 처리. (2) 스트리밍: 실시간 데이터 흐름 처리. (3) 실시간: 즉각적인 요청/응답.
과적합 (Overfitting)	모델이 학습 데이터에만 너무 치중하여, 새로운(실제) 데이터에 대한 예측 성능이 떨어지는 현상입니다. (예: 족보만 외워서 응용 문제를 못 푸는 학생)
피처 스토어 (Feature Store)	자주 사용되는 피처(특성)들을 저장하고 재사용할 수 있게 만든 중앙 저장소입니다.
AutoML	모델 생성 과정을 자동화하는 기술입니다. 데이터만 입력하면 AI가 알아서 최적의 모델을 탐색하고 생성해 줍니다.

3 핵심 개념: MLOps란 무엇인가?

3.1 MLOps의 정의와 필요성

전통적인 소프트웨어 개발은 'DevOps(개발+운영)'를 통해 안정적인 배포와 운영을 자동화했습니다. 하지만 머신러닝 시스템은 기존 소프트웨어와 근본적인 차이가 있습니다.

- 전통적 소프트웨어 = 코드 (Code)
- AI 시스템 = 코드 (Model/Algorithm) + 데이터 (Data)

AI 시스템은 코드뿐만 아니라 데이터의 변화에도 민감하게 반응합니다. 따라서 모델을 한 번 만들고 끝나는 것이 아니라, 지속적으로 관리하고 업데이트해야 합니다.

MLOps (Machine Learning Operations)는 머신러닝(ML), 개발(Dev), 운영(Ops)의 합성어입니다. 이는 머신러닝 모델의 개발, 배포, 운영 전 과정을 체계적이고, 신뢰할 수 있으며, 반복 가능하게 만드는 문화와 기술적 관행을 의미합니다.

▣ 핵심 요약

MLOps의 목표: 기존 DevOps의 원칙(CI/CD, 자동화, 모니터링)을 머신러닝 파이프라인에 적용하여, 모델이 실제 비즈니스 환경에서 안정적이고 지속적으로 가치를 창출하도록 보장하는 것입니다.

3.2 비즈니스 가치 (예: 고객 이탈 방지)

모든 ML 프로젝트는 비즈니스 문제 해결에서 시작됩니다. MLOps가 어떻게 가치를 창출하는지 '고객 이탈(Churn) 방지' 사례를 통해 알아보겠습니다.

- **상황:** 한 마케팅 분석 팀이 고객의 인구통계 및 과거 서비스 데이터를 대시보드로 보고 있습니다.
- **기존 방식 (BI):** "지난 달에 어떤 고객들이 이탈했는가?" (과거 분석)
- **비즈니스 요구 (ML):** "앞으로 어떤 고객이 이탈할 것 같은가?" (미래 예측)

왜 예측이 중요한가? 신규 고객을 유치하는 비용보다 기존 고객을 유지하는 비용이 훨씬 저렴합니다. 고객이 떠나기 *전에* 예측할 수 있다면, 할인 쿠폰이나 인센티브를 제공하는 등 선제적인 조치를 취해 고객 이탈을 막고 수익을 보존할 수 있습니다.

MLOps는 이 '이탈 예측 모델'을 개발하고, 실제 운영 환경에 배포하며, 모델 성능이 떨어지지 않도록 지속적으로 모니터링하는 전 과정을 관리합니다.

4 MLOps의 핵심 구성원 (The "People")

성공적인 MLOps는 다양한 역할의 전문가들이 협업하는 '프로세스'입니다. 각 구성원의 역할을 이해하는 것이 중요합니다.

4.1 비즈니스 이해관계자 (Business Stakeholder)

- **역할:** ML 솔루션의 비즈니스 가치를 책임집니다.
- **핵심 임무:**
 - **문제 정의:** 모호한 비즈니스 목표를 명확한 데이터 문제로 번역합니다.
* (예: "고객 이탈"이란 정확히 무엇인가? → "90일 내 재구매가 없거나, 1개월 내 구독을 취소한 고객")
 - **KPI 설정:** 모델의 성공을 측정할 핵심 성과 지표(KPI)를 결정합니다.
* (예: 이탈률 감소, 유지율 증가, 개입(할인) 대비 투자 수익률(ROI))
 - **결과 검증:** 배포된 모델이 실제로 비즈니스 목표 달성을 기여하는지(예: 이탈률 감소) 확인하고 분석합니다.

4.2 데이터 엔지니어 (Data Engineer)

- **역할:** 데이터 파이프라인을 구축합니다.
- **핵심 임무:**
 - **데이터 수집:** 다양한 소스(CRM, 웹 로그, 결제 시스템 등)에서 데이터를 수집합니다.
 - **데이터 처리 (ETL/ELT):** 데이터를 추출(Extract), 변환(Transform), 적재(Load)하여 분석 가능한 형태로 가공합니다.
 - **품질 보증:** 누락된 값, 중복 데이터, 불일치 등을 처리하여 데이터의 신뢰성을 보장합니다.
 - **인프라 관리:** 데이터가 저장되고 접근될 수 있는 확장 가능한 인프라(예: AWS, Azure)를 관리합니다.
- **최종 산출물:** 분석 및 모델링에 즉시 사용 가능한 "신뢰할 수 있는 프로덕션 등급 데이터셋"

4.3 데이터 사이언티스트 (Data Scientist)

- **역할:** 데이터를 예측 가능한 '인사이트'로 변환합니다.
- **핵심 임무:**
 - **탐색적 데이터 분석 (EDA):** 데이터를 탐색하며 패턴을 파악합니다.
 - **피처 엔지니어링:** 이탈에 영향을 미치는 변수(피처)를 식별하고 생성합니다. (예: 사용 빈도, 마지막 구매일, 불만 접수 횟수)
 - **모델 선택 및 학습:** 다양한 알고리즘(예: 로지스틱 회귀, 랜덤 포레스트, XGBoost)을 실험하고 최적의 모델을 학습시킵니다.
 - **성능 평가:** 정확도, 정밀도 등 적절한 지표로 모델 성능을 검증합니다.
 - **결과 해석:** 어떤 요인이 이탈에 가장 큰 영향을 미치는지 해석하고 비즈니스팀과 협업합니다.

4.4 머신러닝 엔지니어 (ML Engineer)

- **역할:** 데이터 사이언티스트가 만든 모델(프로토타입)을 실제 '제품(Production)'으로 만들고 유지보수합니다.
- **핵심 임무:**
 - **모델 배포:** 학습된 모델을 API 엔드포인트나 배치 스코어링 파이프라인으로 배포합니다.
 - **CI/CD 구축:** 모델의 재학습, 테스트, 버전 관리, 배포 과정을 자동화합니다.
 - **모니터링:** 모델 성능(정확도), 데이터 드리프트, 지연 시간(latency), 편향성(bias) 등을 지속적으로 추적합니다.
 - **확장성 보장:** 트래픽이 증가해도 안정적으로 예측 서비스를 제공할 수 있도록 인프라를 최적화합니다.

4.5 데이터 거버넌스 책임자 (Data Governance Officer)

- **역할:** 데이터 거버넌스 및 규정 준수(Compliance)를 책임집니다.
- **핵심 임무:**
 - 전체 파이프라인에 걸쳐 데이터 보안, 개인정보 보호(의명화), 접근 제어가 올바르게 적용되는지 감독합니다.

▣ 핵심 요약

역할별 ML 파이프라인 참여도 요약

프로세스의 각 단계는 여러 역할의 협업으로 이루어집니다.

- 1. 데이터 준비 (Data Preparation):
 - 주도: 데이터 엔지니어
 - 감독: 데이터 거버넌스
- 2. 탐색적 데이터 분석 (EDA):
 - 주도: 데이터 사이언티스트
 - 감독: 데이터 거버넌스
- 3. 피처 엔지니어링 (Feature Engineering):
 - 주도: 데이터 사이언티스트
 - 감독: 데이터 거버넌스
- 4. 모델 학습 (Model Training):
 - 주도: 데이터 사이언티스트
 - 감독: 데이터 거버넌스
- 5. 모델 검증 (Model Validation):
 - 협업: 데이터 사이언티스트, ML 엔지니어, 비즈니스 이해관계자
 - 감독: 데이터 거버넌스
- 6. 배포 (Deployment):
 - 주도: ML 엔지니어
 - 협업: 비즈니스 이해관계자
 - 감독: 데이터 거버넌스

- 7. 모니터링 (Monitoring):
 - 주도: ML 엔지니어
 - 협업: 비즈니스 이해관계자
 - 감독: 데이터 거버넌스

5 MLOps의 3대 파이프라인 (The 3 Pipelines)

MLOps는 크게 세 가지의 상호 연결된 파이프라인으로 구성됩니다.

5.1 1. 모델 학습 파이프라인 (Model Training Pipeline)

- 목표: 원본 데이터(Raw Data)로부터 잘 일반화된(과적합되지 않은) 모델 아티팩트(파일)를 생성합니다.
- 주요 단계:
 1. 데이터 준비: 데이터를 학습(Train) / 검증(Validation) / 테스트(Test) 세트로 분할합니다.
 2. 피처 엔지니어링: 원본 변수를 모델에 유의미한 입력값으로 변환합니다. (피처 스토어에서 가져오거나 새로 생성)
 3. 모델 학습: 적절한 알고리즘을 선택하고 학습 데이터로 모델을 훈련시킵니다.
 4. 파라미터 튜닝: 모델의 정확도를 높이고 과적합을 줄이기 위해 하이퍼파라미터를 최적화합니다.
 5. 평가: 검증 세트를 사용해 정확도, 정밀도(Precision), 재현율(Recall), F1 점수 등 다양한 지표로 모델 성능을 평가합니다.
 6. 아티팩트 저장: 성능이 좋은 모델을 재사용 및 배포할 수 있도록 파일(아티팩트)로 저장하고 MLflow 같은 도구에 기록합니다.

5.2 2. 모델 추론 파이프라인 (Model Inferencing Pipeline)

- 목표: 학습된 모델을 사용하여 새로운 데이터에 대한 예측(결정)을 생성하고, 비즈니스 의사결정을 지원합니다.
- 주요 배포 방식:
 - 배치 추론 (Batch Inferencing):
 - * 작동 방식: 주기적으로(예: 매일 밤) 대량의 데이터를 한 번에 처리합니다.
 - * 예시: "다음 주에 이탈할 가능성이 있는 고객" 목록을 미리 생성하여 마케팅 팀에 전달합니다.
 - 실시간 추론 (Real-time Inferencing):
 - * 작동 방식: 사용자의 요청이 올 때마다 즉각적으로 예측을 반환합니다. (보통 API 형태)
 - * 예시: 고객이 웹사이트에서 '구독 취소' 버튼 근처에 마우스를 가져갈 때, 실시간으로 이탈 점수를 계산하여 할인 쿠폰 팝업을 띄웁니다.
- 주요 고려사항: 성능(지연 시간), 처리량(Throughput), 확장성, 보안.

5.3 3. 모델 모니터링 파이프라인 (Model Monitoring Pipeline)

- 목표: 운영 환경(Production)에서 모델의 동작과 성능을 지속적으로 추적하여 문제(성능 저하)를 감지합니다.
- 주요 모니터링 대상(드리프트):
 - 데이터 드리프트 (Data Drift):
 - * 정의: 모델에 입력되는 데이터의 통계적 분포가 시간이 지남에 따라 변하는 현상입니다.
 - * 예시: 새로운 마케팅 캠페인으로 인해 이전과 다른 연령대의 신규 고객이 대거 유입되는 경우.
 - 개념 드리프트 (Concept Drift):

- * 정의: 입력 데이터와 목표 변수(예측 대상) 간의 관계 자체가 변하는 현상입니다. 데이터 분포는 그대로일 수 있습니다.
- * 예시: (1) 새로운 경쟁사 출시로 인해 고객이 '가격'보다 '서비스 품질'을 이탈의 주요 원인으로 삼기 시작하는 경우. (2) COVID-19로 인해 사람들의 소비 패턴이 근본적으로 바뀐 경우.
- 모델 품질 저하: 실제 결과(Ground Truth)와 모델의 예측값을 비교하여 정확도, F1 점수 등이 설정한 임계값 이하로 떨어지는지 확인합니다.
- 조치: 드리프트나 성능 저하가 감지되면, 경고(Alert)를 보내거나 자동으로 모델 재학습(Retrain) 파이프라인을 실행합니다.

□ 예제:

Q: 데이터 드리프트와 모델 드리프트(성능 저하)는 어떻게 자동으로 감지하나요?

A: 주로 자동화된 통계적 품질 검사를 통해 감지합니다.

- **데이터 드리프트 감지:** 모델을 학습시킬 때 사용했던 데이터(베이스라인)의 통계적 특성(평균, 분산, 분포 등)을 저장해둡니다. 그리고 실제 운영 환경에 새로 들어오는 데이터의 통계적 특성을 이 베이스라인과 주기적으로 비교합니다. 두 분포 간의 차이가 특정 임계값을 넘으면 드리프트로 간주합니다.
- **모델 드리프트 (성능 저하) 감지:** 모델의 예측 정확도를 모니터링하는 것이 가장 확실한 방법입니다. (예: 모델이 "이탈 안 함"으로 예측했는데, 실제로는 이탈한 고객의 비율이 점점 증가하는지 추적)
- **개념 드리프트 감지:** 가장 감지하기 어렵습니다. 데이터 드리프트나 모델 성능 저하가 뚜렷하지 않은데도 비즈니스 KPI(예: 할인 쿠폰 성공률)가 하락한다면, 비즈니스 맥락 자체가 변했을(개념 드리프트) 가능성을 의심하고 수동으로 분석해야 할 수 있습니다.

6 모델 배포 전략 (ML Deployment Patterns)

모델을 개발(dev) 환경에서 스테이징(staging), 운영(prod) 환경으로 승격시킬 때 두 가지 주요 전략이 있습니다.

6.1 1. 모델 배포 (Deploy Models)

- 방식:** 개발(dev) 환경에서 모델을 학습시키고, 생성된 모델 아티팩트(파일) 자체를 스테이징, 운영 환경으로 복사하여 배포합니다.
- 흐름:** [Dev에서 학습] → [모델 파일 생성] → [파일을 Staging으로 복사] → [파일을 Prod로 복사]

6.2 2. 코드 배포 (Deploy Code)

- 방식:** 개발(dev) 환경에서 모델 학습 코드를 개발하고, 이 코드를 스테이징, 운영 환경으로 복사(배포) 합니다. 그 다음, 각 환경에서 해당 환경의 데이터로 모델을 다시 학습시킵니다.
- 흐름:** [Dev에서 코드 개발] → [코드를 Staging으로 복사] → [Staging 데이터로 재학습] → [코드를 Prod로 복사] → [Prod 데이터로 재학습]

6.3 코드 배포의 장단점 (Benefits of Deploy Code)

'코드 배포' 방식은 더 복잡해 보이지만, MLOps 관점에서 많은 이점을 제공합니다.

Table 2: '코드 배포' 전략의 주요 장점

장점	설명
데이터 접근 제어	가장 큰 장점. 민감한 운영(Prod) 데이터를 비-운영(Dev, Staging) 환경으로 가져올 필요가 없습니다. 각 환경은 자신의 데이터에만 접근하여 모델을 학습합니다.
재현성 높은 모델	엔지니어링 팀이 각 환경(Dev, Staging, Prod)의 학습 환경(라이브러리, 의존성 등)을 완벽하게 제어할 수 있어 모델의 재현성을 단순화하는 데 도움이 됩니다.
대규모 프로젝트 지원	이 패턴은 데이터 사이언스팀이 모듈화된 코드를 작성하고 반복적인 테스트를 수행하도록 강제합니다. 이는 대규모 프로젝트에서 협업과 개발을 용이하게 합니다.
실제 데이터 문제 처리	개발 환경의 샘플 데이터에서는 발생하지 않던 실제 운영 데이터의 편향(skew)이나 대용량으로 인한 문제를 운영 환경에서 직접 학습하며 처리할 수 있습니다. (예: Staging에서는 잘 동작했으나 Prod의 대용량 데이터 패턴으로 인해 실패하는 문제 방지)

주의사항

'코드 배포'의 단점

- 더 많은 컴퓨팅 비용: 각 환경에서 모델을 재학습해야 하므로 컴퓨팅 자원이 더 많이 소모됩니다.
- 인프라 요구사항: 일회성 모델이라도 단위 테스트, 통합 테스트를 위한 CI/CD 인프라가 필요합니다.
- 요구 역량: 데이터 사이언티스트가 단순히 모델 파일만 전달하는 것이 아니라, 엔지니어링 팀이 운영할 수 있는 모듈화된 코드를 작성하고 핸드오프(handoff)해야 합니다.

7 데이터 중심 AI (Data-Centric AI)

모델의 성능을 높이는 방법은 크게 두 가지입니다.

1. 모델 중심 AI (Model-Centric AI): ”더 좋은 모델(코드/알고리즘)을 만들어서 성능을 높일 수 없을까?”
2. 데이터 중심 AI (Data-Centric AI): ”데이터(입력 또는 레이블)를 체계적으로 개선해서 성능을 높일 수 없을까?”

과거에는 모델 중심 접근법이 주를 이루었지만, 최근 MLOps의 핵심 철학은 데이터 중심 AI입니다.

7.1 빅데이터 vs. 좋은 데이터 (Big Data vs. Good Data)

무조건 데이터가 많다고(Big Data) 좋은 모델이 나오는 것은 아닙니다. 품질이 낮은 대량의 데이터는 오히려 모델 성능에 해가 될 수 있습니다. 중요한 것은 ”좋은 데이터(Good Data)”입니다.

좋은 데이터란?

- 명확한 레이블 (Unambiguous labels): 정답(레이블)이 일관되고 명확해야 합니다.
- 중요 사용 사례 포괄 (Good cover): 예측해야 할 중요한 케이스들을 충분히 포함해야 합니다. (예: 모든 주(State)를 예측해야 하는데, 매사추세츠주 데이터만 사용하면 안 됨)
- 시기적절한 피드백 (Timely feedback): 운영 환경에서 발생한 최신 데이터를 빠르게 피드백 받을 수 있어야 합니다.
- 적절한 크기 (Sized appropriately): 너무 적어도 안 되지만, 불필요하게 많을 필요도 없습니다.

7.2 왜 데이터 중심 AI가 중요한가?

AI에게 데이터는 ’음식’과 같습니다.

□ 예제:

비유: 데이터는 요리의 재료다

최고의 셰프(모델/알고리즘)라 할지라도, 상한 재료(나쁜 데이터)로는 맛있는 요리(좋은 예측)를 만들 수 없습니다.

- 모델 중심 접근: 이미 좋은 재료가 있을 때, 레시피를 미세 조정하는 것 (예: 15분 끓일 것을 16분 끓이기). → 성능 향상에 한계가 있음.
- 데이터 중심 접근: 레시피는 그대로 두고, 더 신선하고 고품질의 재료(데이터)를 공수해 오는 것. → 요리(모델)의 품질을 극적으로 향상시킬 수 있음.

실제 산업 사례(강철 결함 감지 등)에서도, 모델 코드를 수정하는 것(Model-centric)보다 데이터 품질을 개선하는 것(Data-centric)이 **월등히 높은(예: +16.9%) 성능 향상**을 보였습니다. MLOps는 이처럼 고품질의 데이터를 지속적으로 확보하고 개선하는 프로세스를 자동화하는 데 중점을 둡니다.

8 MLOps 성숙도 모델 (MLOps Progression)

모든 조직이 처음부터 완벽한 MLOps를 갖추는 것은 아닙니다. MLOps는 점진적으로 발전하는 '성숙도' 단계를 가집니다.

- **Level 1: Beginner (초급)**
 - 익숙한 도구를 사용하여 생산성을 높이고, 더 복잡한 작업을 계획하는 단계.
 - (예: 데이터 사이언티스트가 수동으로 모델을 학습하고 파일을 전달함)
- **Level 2: Intermediate (중급)**
 - 데이터 및 워크로드를 확장(Scale) 합니다.
 - 자동화와 재현성이 중점을 둡니다.
 - 흩어져 있던 데이터 팀을 통합하고 협업을 개선합니다.
- **Level 3: Advanced (고급)**
 - 더 빠른 생산 주기: CI/CD가 완전히 자동화되어 한 달에 한 번이 아닌, 하루에도 여러 번 배포가 가능합니다.
 - E2E 자동화 및 재현성: 여러 워크로드에 걸쳐 엔드투엔드(End-to-End) 자동화가 이루어집니다.
 - 회복 탄력성: 넷플릭스(Netflix)의 '카오스 몽키(Chaos Monkey)'처럼 의도적으로 프로덕션에 이슈를 발생시켜 시스템이 얼마나 빨리 스스로 복구하는지 테스트합니다.
 - 롤백(Rollback) 메커니즘: 배포에 문제가 생겼을 때 즉시 이전 버전으로 되돌리는 기능이 완비되어 있습니다.

9 모델 튜닝 및 선택 (Model Tuning and Selection)

모델 학습 파이프라인에서 '좋은 모델'을 만들기 위한 핵심 기술입니다.

9.1 하이퍼파라미터 튜닝 (Hyperparameter Tuning)

- 목표:** 모델이 스스로 학습하지 않는 값, 즉 개발자가 직접 설정해야 하는 '하이퍼파라미터'(예: 트리의 깊이, 트리의 개수)를 최적화하여 과적합/과소적합을 방지하고 모델 성능을 극대화합니다.
- 주요 방식:**
 - 수동 (Grid Search):**
 - * **작동 방식:** 개발자가 지정한 하이퍼파라미터 값의 모든 조합을 시도합니다. (예: [깊이]: 3, 5], [트리 개수: 100, 200] → (3,100), (3,200), (5,100), (5,200) 총 4번 학습)
 - * **특징:** 중첩된 for문과 같아서 파라미터가 많아지면 매우 비효율적이고 비쌉니다.
 - 자동 (Automated):**
 - * **작동 방식:** Optuna, Hyperopt 같은 라이브러리를 사용합니다.
 - * **종류:**
 - **Random Search:** 무작위로 조합을 탐색 (Grid Search보다 효율적일 때가 많음)
 - **Bayesian Search:** 이전 시도 결과를 바탕으로 다음 시도에 더 성능이 좋을 만한 영역을 지능적으로 탐색합니다.
 - **Genetic Search:** 좋은 모델과 나쁜 모델의 파라미터를 유전 알고리즘처럼 조합하여 더 나은 모델을 탐색합니다. (딥러닝에서 유용)

9.2 K-겹 교차 검증 (K-Fold Cross Validation)

- 목표:** 모델을 평가할 때 '테스트 세트'를 사용하면, 모델이 해당 테스트 세트에만 과적합될 위험이 있습니다. (일종의 '치팅') 이를 방지하기 위해 학습 데이터 내부에서만 모델을 검증하는 기법입니다.
- 작동 방식 (예: K=3인 경우):**
 1. 학습 데이터(Train + Validation)를 3개의 '폴드(Fold)'로 나눕니다. [Fold 1], [Fold 2], [Fold 3]
 2. **Pass 1:** [Fold 1, 2]로 학습 → [Fold 3]로 검증
 3. **Pass 2:** [Fold 1, 3]로 학습 → [Fold 2]로 검증
 4. **Pass 3:** [Fold 2, 3]로 학습 → [Fold 1]로 검증
 5. 3개 검증 점수의 평균을 내어 해당 하이퍼파라미터의 최종 성능으로 삼습니다.
- 장점:** 학습 데이터를 최대한 활용하면서도, 테스트 세트(Holdout set)를 오염시키지 않고 모델의 일반화 성능을 안정적으로 평가할 수 있습니다.

10 실습: E2E MLOps 파이프라인 (Lab Walkthrough)

Databricks 환경에서 고객 이탈 예측 모델을 E2E(End-to-End) MLOps 파이프라인으로 구축하는 과정을 요약합니다.

10.1 실습 목표 및 도구

- 목표:** 고객 이탈(Churn)을 예측하는 LightGBM 모델을 MLOps 수명 주기에 맞춰 개발, 등록, 검증, 배포합니다.
- 주요 도구:**
 - Delta Lake:** 데이터의 저장, 버전 관리, 트랜잭션 보장
 - MLflow:** 모델 수명 주기 관리 (Tracking, Registry)
 - Unity Catalog (UC):** 중앙화된 데이터 및 모델 거버넌스 (카탈로그, 스키마, 모델 관리)
 - LightGBM:** 머신러닝 알고리즘

10.2 1단계: 데이터 준비 및 피처 엔지니어링

- 원본 데이터(mllops_churn_bronze_customer)를 읽어옵니다.
- EDA를 수행합니다. (예: Pandas API on Spark를 사용해 인터넷 서비스 유형별 고객 수 계산)
- 데이터를 정제하고 새로운 피처를 생성합니다.
 - (예: '온라인 보안', '온라인 백업' 등 부가 서비스 가입 여부를 합산하여 num_optional_services 피처 생성)
- 데이터를 학습(Train) 및 추론(Inference) 용으로 분할하여 Delta Table(mllops_churn_training)로 저장합니다.

10.3 2단계: 모델 학습 및 로깅 (MLflow Tracking)

- MLflow Experiment(실험실)를 설정합니다.
- 데이터 라인지(Lineage) 로깅 (핵심):**
 - 모델을 학습시키기 전에, 어떤 데이터 테이블의 어떤 버전을 사용했는지 MLflow에 명시적으로 기록합니다.
 - `mlflow.data.load_delta(...)` → `mlflow.log_input(dataset, ...)`
 - 이유:** 모델의 재현성을 위해 ”어떤 코드가 어떤 데이터로 만들었는지” 추적하는 것은 필수입니다. (이때 데이터 복사본이 아닌 메타데이터만 저장됩니다.)
- 데이터 전처리 파이프라인을 정의합니다. (Boolean, Numerical, Categorical 피처별 변환기)
- LightGBM 분류기(Classifier)를 포함한 전체 학습 파이프라인을 정의합니다.
- `mlflow.autolog()`를 활성화하여 파라미터, 지표, 아티팩트를 자동으로 로깅합니다.
- 모델을 학습(`.fit()`)시키고 MLflow에 모델을 로깅(`mlflow.log_model()`)합니다.
- 결과:** MLflow 실행(Run) 페이지에 모든 정보(파라미터, F1 점수 등)와 아티팩트(ROC 커브, 혼동 행렬, `requirements.txt`, 모델 파일 등)가 기록됩니다.

10.4 3단계: 모델 레지스트리 등록 (MLflow Registry)

1. 2단계에서 실행된 수많은 실험(Run) 중에서 최고의 모델(예: F1 점수가 가장 높은 모델)을 검색합니다.
2. `mlflow.register_model()`을 사용하여 이 모델을 Unity Catalog 내의 모델 레지스트리(예: `cscie103_catalog`)에 등록합니다.
3. 등록된 모델은 ‘version 1’을 받습니다.
4. 모델 리니지 확인: UC의 모델 레지스트리 UI에서, 이 모델 버전(v1)이 어떤 데이터셋과 어떤 노트북(코드)을 통해 생성되었는지 업스트림(Upstream) 리니지를 시각적으로 확인할 수 있습니다.
5. 모델과 버전에 설명(Description)과 태그(Tag)를 추가합니다. (예: ‘F1score = 0.85’)
5. 이 신규 등록 모델에 ‘Challenger’(도전자)라는 별칭(Alias)을 설정합니다.

10.5 4단계: 챌린저 모델 검증 (Champion vs. Challenger)

1. 자동화된 품질 검사: ‘Challenger’ 별칭으로 모델을 로드한 뒤, 자동화된 테스트를 수행합니다. (예: 모델 설명이 비어있지 않은가? F1 점수가 최소 기준치(0.8) 이상인가?)
2. 챔피언-챌린저 대결:
 - 현재 운영 환경에 배포된 모델(별칭: ‘Champion’)을 로드합니다. (try-except 구문 사용. 챔피언이 없으면(첫 배포) Challenger가 자동 승리)
 - ‘Challenger’ 모델의 F1 점수와 ‘Champion’ 모델의 F1 점수를 비교합니다.
3. 승격 (Promotion):
 - 만약 ‘Challenger’가 더 우수하다면, ‘Challenger’ 모델의 별칭을 ‘Champion’으로 설정(승격)합니다.
 - (기존 챔피언은 ‘Archived’ 상태가 되거나 별칭이 제거됩니다.)

10.6 5단계: 배치 추론 (Batch Inference)

1. ‘Champion’ 별칭으로 현재 운영 모델을 로드합니다.
2. 1단계에서 분리해둔 새로운 데이터(추론용 데이터)를 로드합니다.
3. `mlflow.pyfunc.spark_udf`를 사용하여 Spark 데이터프레임에 챔피언 모델을 적용합니다.
4. 결과: 원본 데이터에 ‘predictions’ 열(예: 1(이탈), 0(유지))이 추가된 최종 결과 테이블을 얻습니다. 이 테이블은 마케팅 팀의 대시보드나 액션 시스템으로 전달됩니다.

11 핵심 용어 정리

이 강의에서 다룬 MLOps의 핵심 용어를 표로 정리합니다.

Table 3: *MLOps* 핵심 용어집

용어	설명 (초심자 가이드)
MLOps	머신러닝(ML), 개발(Dev), 운영(Ops)의 합성어. ML 시스템을 안정적이고 반복적으로 개발, 배포, 운영하기 위한 문화 및 기술.
데이터 드리프트 (Data Drift)	모델에 입력되는 실제 데이터의 통계적 분포가 학습 당시의 데이터 분포와 달라지는 현상. (예: 신규 고객 연령층 변화)
개념 드리프트 (Concept Drift)	데이터와 정답 간의 '관계' 자체가 변하는 현상. (예: 가격 때문에 이탈하던 고객들이 이제는 서비스 품질 때문에 이탈)
CI/CD	지속적 통합(Continuous Integration) / 지속적 배포(Continuous Deployment). 코드 변경사항을 자동으로 테스트하고 운영 환경에 배포하는 자동화 프로세스.
모델 아티팩트 (Model Artifact)	모델 학습의 결과물. 모델 가중치가 저장된 파일(예: pickle 파일), conda.yaml, requirements.txt 등을 포함하는 패키지.
모델 레지스트리 (Model Registry)	학습된 모델 아티팩트를 저장, 버전 관리, 상태(Staging, Prod) 추적하는 중앙 카탈로그. (예: MLflow Registry)
리니지 (Lineage)	'혈통' 또는 '계보'. 데이터나 모델이 어떤 원본(데이터, 코드)으로부터 어떻게 변환되어 생성되었는지 그 이력을 추적하는 것.
하이퍼파라미터 (Hyperparameter)	모델이 학습 과정에서 스스로 찾는 값(Parameter)이 아니라, 개발자가 사전에 직접 설정해야 하는 값. (예: 학습률, 트리 깊이)
Grid Search	하이퍼파라미터 튜닝 기법 중 하나. 개발자가 지정한 모든 파라미터 조합을 시도하는 '격자 탐색' 방식.
K-겹 교차 검증 (K-Fold CV)	학습 데이터를 K개로 나누어, K-1개로 학습하고 1개로 검증하는 과정을 K번 반복하여 모델 성능을 평균내는 검증 기법.
챔피언/챌린저 (Champion/Challenger)	모델 배포 전략. 현재 운영 중인 최고 성능 모델(Champion)과 새로 개발된 모델(Challenger)의 성능을 비교하여, 챌린저가 더 우수할 경우 챔피언을 교체하는 방식.
데이터 중심 AI (Data-Centric AI)	모델 알고리즘 개선보다 고품질의 데이터를 확보하고 개선하는 데 집중하여 AI 성능을 높이려는 접근 방식.

12 빠르게 훑어보기 (1-Page Summary)

□ MLOps 핵심 요약 카드

1. MLOps란?

ML(머신러닝) + Dev(개발) + Ops(운영)의 합성어.

AI 시스템(Code + Data)을 신뢰할 수 있고 반복 가능하게 개발, 배포, 모니터링하는 전 과정.

2. MLOps 핵심 4대 역할

- 데이터 엔지니어(DE): 데이터 파이프라인 구축 (신뢰 가능한 데이터 제공)
- 데이터 사이언티스트(DS): 모델 개발 (데이터 → 인사이트)
- ML 엔지니어(MLE): 모델 배포 및 운영 (프로토타입 → 프로덕션)
- 비즈니스 이해관계자: 문제 정의 및 KPI 설정

3. 3대 핵심 파이프라인

- 학습(Training): 데이터 → 모델 아티팩트 생성
- 추론(Inferencing): 모델 → 예측값 생성 (Batch vs. Real-time)
- 모니터링(Monitoring): 모델 성능 추적 및 드리프트 감지

4. 드리프트(Drift) 3종 세트

- 데이터 드리프트: 입력 데이터의 분포 변경
- 개념 드리프트: 데이터-정답 간의 '관계' 변경
- 모델 드리프트: 위 둘로 인한 모델 성능 저하

5. 데이터 중심 AI (Data-Centric AI)

AI 성능 향상은 '모델 코드' 개선보다 '데이터 품질' 개선이 더 효과적이다. (데이터는 AI의 '음식 재료'와 같다)

→ MLOps는 고품질 데이터를 지속 공급/관리하는 프로세스.

6. 배포 전략 (Code vs. Model)

- 모델 배포: 학습된 '모델 파일'을 Prod 환경에 복사.
- 코드 배포 (권장): '학습 코드'를 Prod 환경에 복사 후, Prod 데이터로 '재학습'. (데이터 접근 제어 및 재현성에 유리)

December 10, 2025

■ 강의명: CSCI E-103: 재현 가능한 머신러닝

■ 주차: Lecture 10

■ 교수명: Anindita Mahapatra

Eric Gieseke

■ 목적: Lecture 10의 핵심 개념 학습

▣ 핵심 요약

이 문서는 10번째 강의인 '데이터 및 머신러닝 문제 다루기'의 핵심 내용을 요약 및 보완하여 정리한 노트입니다. 데이터 문제가 모델 문제로 이어지는 과정을 이해하고, 특히 모델 편향(Model Bias), 데이터 불균형(Class Imbalance) 문제를 중점적으로 다룹니다.

또한, AutoML의 개념과 'Glassbox ML' 접근법을 살펴보고, 마지막으로 이 모든 개념이 통합된 실제 대규모 실시간 사기 탐지(Fraud Detection) 시스템 아키텍처 사례를 심층적으로 분석합니다.

Contents

1	개요: 데이터에서 모델까지	2
2	주요 용어 정리	3
3	AutoML: 모델 개발의 자동화	5
3.1	Blackbox vs Glassbox	5
3.2	AutoML 지원 모델 유형	5
3.3	AutoML 활용 시나리오 (데모 요약)	6
4	머신러닝의 오류: 편향과 분산	7
4.1	편향 (Bias) = 과소적합 (Underfitting)	7
4.2	분산 (Variance) = 과대적합 (Overfitting)	7
4.3	편향-분산 트레이드오프 (Bias-Variance Trade-off)	7
5	모델 편향(Model Bias) 심층 분석	8
5.1	모델 편향은 왜 발생하는가?	8
5.1.1	1. 인간의 인지 편향 (Human Factor)	8
5.1.2	2. 낮은 품질의 훈련 데이터 (Poor Quality Data)	8

5.2 모델 편향을 줄이는 방법 (슬라이드 7)	8
6 데이터 불균형 문제 (Class Imbalance)	10
6.1 불균형 데이터의 진짜 문제: '정확도'의 함정	10
6.2 해결책 1: 평가 지표 변경 (Accuracy 대신 F1 Score)	10
6.3 해결책 2: 리샘플링 (Resampling)	10
6.4 해결책 3: SMOTE (합성 소수 오버샘플링 기법)	10
7 실전 사례 연구: 실시간 사기 탐지 시스템	12
7.1 비즈니스 요구사항	12
7.2 피처(Feature) 정의	12
7.3 시스템 아키텍처: 람다 아키텍처 (Lambda Architecture)	12
7.4 핵심 기술 1: 메타데이터 기반 코드 생성 (Slide 19)	13
7.5 핵심 기술 2: 순환 버퍼 (Circular Buffer) (Slide 20)	13
7.6 핵심 기술 3: 서빙 레이어 (Cassandra) (Slide 22)	13
7.7 종합 아키텍처 (Slide 21)	14
8 학습 점검을 위한 체크리스트	15
9 FAQ (자주 묻는 질문)	16
10 빠르게 훑어보기 (1-Page 요약)	17

1 개요: 데이터에서 모델까지

이번 강의의 핵심 주제는 ”데이터 문제가 곧 모델 문제”라는 것입니다. 머신러닝 모델의 성능은 결국 우리가 제공하는 데이터의 품질에 달려 있습니다.

강의 핵심 요약

- **모델 오류의 이해:** 모델의 총 오류는 편향(Bias), 분산(Variance), 그리고 줄일 수 없는 오류(Irreducible Error)로 구성됩니다. 이 중 편향과 분산은 트레이드오프 관계에 있습니다.
- **모델 편향 문제:** 모델이 특정 집단에게 불공정한 예측을 하는 현상입니다. 이는 주로 훈련 데이터의 부족(Under-representation)이나 인간의 인지 편향이 데이터에 반영되어 발생합니다.
- **데이터 불균형 문제:** 사기 탐지나 질병 진단처럼, 한 클래스가 극도로 드문(예: 99% vs 1%) 상황입니다. 이때 ’정확도(Accuracy)’는 무의미하며, F1-Score, 재현율(Recall) 등 다른 평가지표와 SMOTE 같은 리샘플링 기법이 필요합니다.
- **AutoML과 Glassbox:** AutoML은 모델 개발을 자동화하지만, 내부를 볼 수 없는 ’Blackbox’는 한계가 있습니다. Databricks 등이 지향하는 ’Glassbox ML’은 EDA, 하이퍼파라미터 튜닝 등 모든 과정을 노트북으로 투명하게 제공합니다.
- **실전 사례 (사기 탐지):** 15ms의 응답 속도와 초당 5만 건의 트래픽을 처리하는 실시간 사기 탐지 시스템을 구축하기 위해, **람다(Lambda) 아키텍처**, **순환 버퍼(Circular Buffer)**, **Cassandra** 저장소 등 고성능 엔지니어링 기술이 어떻게 적용되었는지 살펴봅니다.

2 주요 용어 정리

강의에서 다룬 핵심 용어들을 미리 정리합니다.

용어	설명 (초심자용 풀이)	원어 / 관련 용어
AutoML	사용자가 데이터를 입력하면, 모델 선택, 피처 엔지니어링, 하이퍼파라미터 튜닝 등을 자동으로 수행하여 최적의 모델을 제안하는 기술입니다.	Automated ML
Glassbox ML	AutoML의 한 종류로, 모델이 만들어지는 전 과정을 투명하게 공개하는 접근법입니다. (예: Databricks AutoML) 내부를 볼 수 없는 'Blackbox'와 대비됩니다.	Glassbox ML vs Blackbox ML
편향 (Bias)	모델이 실제 데이터의 복잡한 관계를 제대로 파악하지 못해, 예측값이 정답과 체계적으로 벗어나는 경향입니다. 과소적합(Underfitting)의 주요 원인입니다.	Bias
분산 (Variance)	모델이 훈련 데이터의 노이즈까지 너무 복잡하게 학습하여, 새로운 데이터(테스트 데이터)에 대한 예측이 불안정하게 흔들리는 정도입니다. 과대적합(Overfitting)의 주요 원인입니다.	Variance
편향-분산 트레이드오프	편향을 낮추려 하면(모델이 복잡해지면) 분산이 높아지고, 분산을 낮추려 하면(모델이 단순해지면) 편향이 높아지는, 두 오류가 반비례하는 관계입니다.	Bias-Variance Trade-off
데이터 불균형	훈련 데이터에서 클래스(정답) 간의 비율이 크게 차이 나는 상황입니다. (예: 정상 99%, 사기 1%)	Class Imbalance / Imbalanced Data
정밀도 (Precision)	모델이 "Yes"라고 예측한 것 중, 실제로 "Yes"인 것의 비율입니다. ($TP / (TP+FP)$) "스팸"이라고 예측한 메일 중 실제 스팸 비율.	Precision
재현율 (Recall)	실제 "Yes"인 것들 중, 모델이 "Yes"라고 예측해낸 비율입니다. ($TP / (TP+FN)$) 실제 "암 환자" 중 몇 명을 "암"이라고 맞혔는가. (민감도, Sensitivity)	Recall / Sensitivity / True Positive Rate
F1 Score	정밀도와 재현율의 조화 평균입니다. 불균형 데이터에서 정확도(Accuracy) 대신 사용하는 핵심 평가지표입니다.	F1 Score
리샘플링 (Resampling)	불균형 데이터를 처리하기 위해 데이터셋을 조작하는 기법입니다. 소수 클래스를 늘리거나(Oversampling) 다수 클래스를 줄입니다(Undersampling).	Resampling

용어	설명 (초심자용 풀이)	원어 / 관련 용어
SMOTE	대표적인 오버샘플링 기법. 소수 클래스 데이터를 단순히 복제하는 것이 아니라, 기존 데이터들 사이에 '합성(Synthetic)' 데이터를 생성하여 추가합니다.	Synthetic Minority Over-sampling Technique
PII	개인 식별 정보. 이름, 전화번호, 주민등록번호, 주소 등 개인을 특정할 수 있는 민감 정보입니다. 모델 학습 전에 반드시 제거하거나 익명화해야 합니다.	Personally Identifiable Information
람다 아키텍처	대용량 데이터를 처리하기 위한 하이브리드 아키텍처. 모든 데이터를 처리하는 배치(Batch) 레이어와 실시간 데이터를 처리하는 스피드(Speed) 레이어로 구성됩니다.	Lambda Architecture
CEP	실시간 스트리밍 환경에서 발생하는 이벤트들의 패턴을 즉각적으로 감지하고 대응하는 기술입니다. (예: 1초 이내 5회 이상 결제 시도 감지)	Complex Event Processing
순환 버퍼	고정된 크기의 메모리 공간을 순환하면서 사용하는 데이터 구조. 사기 탐지 시스템에서는 '최근 7일간의 거래 합계' 등을 매우 빠르고 효율적으로 계산하기 위해 사용되었습니다.	Circular Buffer / Ring Buffer
Cassandra	대용량의 비정형 데이터를 처리하기 위한 NoSQL 데이터베이스. 특히 쓰기(Write) 성능이 매우 빠르며, 사기 탐지 시스템에서 피쳐 스토어(Feature Store)로 사용되었습니다.	Apache Cassandra

3 AutoML: 모델 개발의 자동화

머신러닝 프로젝트의 성공은 좋은 모델을 선택하고, 데이터를 적절히 전처리하며, 최적의 하이퍼파라미터를 찾는 데 달려 있습니다. 이 과정은 시간과 전문 지식이 많이 필요합니다.

AutoML(Automated Machine Learning)은 이 복잡한 과정을 자동화하는 기술입니다.

3.1 Blackbox vs Glassbox

- **Blackbox AutoML (블랙박스):**
 - 데이터만 넣으면 클릭 몇 번으로 최적의 모델을 '결과물'로만 제공합니다.
 - (예: Data Robot)
 - 단점: 왜 이 모델이 선택되었는지, 내부적으로 어떤 피처가 중요했는지 알기 어렵습니다. 기업 환경에서 문제가 발생했을 때 "후드 아래(under the hood)"를 볼 수 없어 대응이 불가능합니다.
- **Glassbox AutoML (글래스박스):**
 - Databricks 등이 추구하는 접근법입니다.
 - 모델 생성에 사용된 모든 과정을 투명하게 노트북(Notebook)으로 제공합니다.
 - 제공되는 산출물:
 1. 데이터 탐색(EDA) 노트북
 2. 하이퍼파라미터 튜닝 과정이 담긴 노트북
 3. 최종 선정된 '베스트 모델' 노트북 (예: LightGBM)
 4. 모델의 예측 근거를 설명하는 SHAP 등 해석 가능성 자료
 - 장점: MLflow와 통합되어 모든 실험이 추적되며, '시민 데이터 과학자(Citizen Data Scientist)'가 생성한 모델을 전문가가 이어받아 수정하고 배포할 수 있습니다.

3.2 AutoML 지원 모델 유형

AutoML은 다양한 유형의 문제에 대해 여러 모델을 동시에 병렬로 실행하고 비교합니다. 다음 표는 Databricks AutoML에서 지원하는 모델의 예시입니다.

Table 2: AutoML 지원 모델 예시 (분류, 회귀, 예측)

분류 (Classification)	회귀 (Regression)	예측 (Forecasting)	예측 (서비스)
결정 트리 (Decision trees)	결정 트리 (Decision trees)	Prophet	Prophet
랜덤 포레스트 (Random forests)	랜덤 포레스트 (Random forests)	Auto-ARIMA	Auto-ARIMA
로지스틱 회귀 (Logistic regression)	선형 회귀 (Linear regression)		DeepAR
XGBoost	XGBoost		
LightGBM	LightGBM		

참고: 분류는 '고객이 이탈할까?(Yes/No)', '암인가?(Yes/No)'처럼 정해진 범주로 나누는 문제입니다. 회귀는 '내일의 금값은?', '부동산 가격은?'처럼 연속된 숫자 값을 예측하는 문제입니다. 예측은 시간 순서가 있는 데이터(시계열)를 기반으로 미래를 예측하는 것입니다. (예: Prophet, ARIMA)

3.3 AutoML 활용 시나리오 (데모 요약)

강의에서는 고객 이탈(Churn) 데이터셋을 사용한 AutoML 데모를 시연했습니다.

1. **실험 설정:** 'churn' 데이터를 선택하고, 예측 할 컬럼('Churn' 여부)을 지정합니다. 평가 지표로 'F1 Score'를 선택하고, 최대 15분의 타임아웃을 설정합니다.
2. **데이터 탐색 (EDA):** AutoML이 자동으로 생성한 '데이터 탐색 노트북'을 확인합니다. 각 변수의 분포, 결측치, 고유값 등을 시각화하여 보여주므로, 사용자가 직접 코드를 짤 필요가 없습니다.
3. **모델 확인:** 15분 후, 여러 모델(XGBoost, LightGBM, 로지스틱 회귀 등)이 F1 Score 기준으로 정렬됩니다. 이 중 'LightGBM'이 베스트 모델로 선정되었습니다.
4. **코드 검토 (Glassbox):** '베스트 모델 노트북'을 열면, 데이터 전처리(Boolean, Numeric, Categorical 컬럼 분리), 하이퍼파라미터 튜닝(Hyperopt 사용), 파이프라인 구성 등 모든 소스 코드가 공개됩니다. 사용자는 이 코드를 그대로 사용하거나 수정할 수 있습니다.
5. **모델 해석:** SHAP 라이브러리를 사용해 어떤 피처(Feature)가 이탈 예측에 중요하게 작용했는지 (Feature Importance) 확인할 수 있습니다.
6. **모델 등록:** 클릭 한 번으로 이 베스트 모델을 MLflow 모델 레지스트리에 등록하여, 추후 배치 또는 실시간 추론에 사용할 수 있습니다.

4 머신러닝의 오류: 편향과 분산

머신러닝 모델이 완벽한 예측을 하지 못하는 이유는 '오류(Error)' 때문입니다. 모델의 총 예측 오류는 세 가지 구성 요소로 나뉩니다.

모델의 총 오류 (Prediction Error) 총 오류 = 편향(Bias) 오류 + 분산(Variance) 오류 + 줄일 수 없는 오류 (Irreducible Error)

- 줄일 수 없는 오류 (Irreducible Error): 데이터 자체에 포함된 무작위성(노이즈)으로, 모델이 제어할 수 없는 한계입니다.

우리가 집중해야 할 것은 편향과 분산입니다.

4.1 편향 (Bias) = 과소적합 (Underfitting)

- 정의: 모델이 너무 단순해서, 데이터에 내재된 실제 관계(신호)를 제대로 잡아내지 못할 때 발생하는 오류입니다.
- 비유: 복잡한 곡선 형태의 데이터를 단순한 '직선'으로만 예측하려는 것과 같습니다.
- 결과: 훈련(Train) 데이터와 테스트(Test) 데이터 모두에서 성능이 낮게 나옵니다.
- 특징: 높은 편향 (High Bias), 낮은 분산 (Low Variance)을 보입니다. (모델이 단순해서 예측이 안정적이지만, 정답과는 거리가 멎)
- 슬라이드 예시: 그래프 A (데이터 분포를 제대로 따르지 못하는 단순한 선형 모델)

4.2 분산 (Variance) = 과대적합 (Overfitting)

- 정의: 모델이 너무 복잡해서, 데이터의 실제 신호(signal)뿐만 아니라 노이즈(noise)까지 과도하게 학습했을 때 발생하는 오류입니다.
- 비유: 훈련 데이터의 모든 점을 통과하기 위해 극도로 구불구불한 선을 그리는 것과 같습니다.
- 결과: 훈련(Train) 데이터에서는 성능이 매우 높지만, 새로운 데이터(Test)에서는 성능이 급격히 떨어집니다. 모델이 '일반화'에 실패한 것입니다.
- 특징: 낮은 편향 (Low Bias), 높은 분산 (High Variance)을 보입니다. (훈련 데이터에 너무 맞춰져 있어, 데이터가 조금만 바뀌어도 예측이 크게 흔들림)
- 슬라이드 예시: 그래프 D (훈련 데이터 포인트를 모두 연결하는 복잡하고 구불구불한 모델)

4.3 편향-분산 트레이드오프 (Bias-Variance Trade-off)

가장 중요한 개념: 트레이드오프 편향과 분산은 트레이드오프(Trade-off) 관계에 있습니다.

- 모델을 단순하게 만들면 (예: 덜 훈련시킴) → 분산은 낮아지지만, 편향이 높아집니다. (과소적합)
- 모델을 복잡하게 만들면 (예: 너무 많이 훈련시킴) → 편향은 낮아지지만, 분산이 높아집니다. (과대적합)

좋은 머신러닝 모델이란, 이 두 오류가 적절히 균형을 이루는 '스위트 스팟(Sweet Spot)'을 찾는 것입니다.

5 모델 편향(Model Bias) 심층 분석

모델이 특정 집단에게만 불리한 예측을 하거나, 사회적 편견을 그대로 학습하는 문제를 '모델 편향'이라고 합니다. 이는 앞서 설명한 통계적 편향(과소적합)과는 다른, '공정성(Fairness)'의 문제입니다.

5.1 모델 편향은 왜 발생하는가?

5.1.1 1. 인간의 인지 편향 (Human Factor)

모델은 데이터를 통해 학습합니다. 만약 데이터 자체가 인간의 편견을 담고 있다면, 모델은 그 편견을 그대로, 심지어 증폭시켜 학습합니다.

- 사례 1: 안면 인식 기술의 정확도 차별 (슬라이드 6 그래프)
 - **현상:** Microsoft, IBM 등 여러 기업의 안면 인식 기술이 '밝은 피부의 남성'에게는 90% 이상의 높은 정확도를 보인 반면, '어두운 피부의 여성'에게는 6070%대의 현저히 낮은 정확도를 보였습니다.
 - **원인:** 이는 알고리즘 자체의 문제가 아니라, 훈련 데이터의 편향성 때문입니다. 모델을 훈련시킬 때 사용한 데이터셋에 '어두운 피부의 여성'의 안면 데이터가 절대적으로 부족(Under-represented) 했기 때문입니다. 모델은 충분히 배우지 못한 것에 대해 나쁜 성능을 보일 수밖에 없습니다.
- 사례 2: 검색 엔진의 편견
 - **현상:** 과거 일부 검색 엔진에서 'evil(악)'을 검색하면 특정 인종의 이미지가, 'good(선)'을 검색하면 백인 아기 이미지가 주로 노출되었습니다.
 - **원인:** 이는 인터넷상의 데이터(텍스트, 이미지 태그)에 존재하는 인간의 사회적, 문화적 편견을 모델이 그대로 학습했기 때문입니다.
- 사례 3: Microsoft 'Tay' 챗봇
 - **현상:** 사용자와의 대화를 통해 학습하도록 설계된 챗봇 'Tay' 가, 일부 악의적인 사용자들로부터 인종차별적, 혐오적 발언("홀로코스트는 조작된 것이다")을 학습하여 공개적으로 해당 발언을 하기 시작했습니다.
 - **원인:** 나쁜 품질의 입력 데이터를 필터링 없이 실시간으로 학습한 결과입니다.

5.1.2 2. 낮은 품질의 훈련 데이터 (Poor Quality Data)

데이터의 양이 많더라도 품질이 낮으면 편향이 발생할 수 있습니다.

- 데이터 해상도 문제: 슬라이드 6의 '개(Volpino Italiano) vs 북극 여우' 사진처럼, 이미지가 너무 저해 상도이거나 품질이 나쁘면, 인간도 구분하기 어렵습니다. 모델도 마찬가지입니다.
- 데이터 라벨링 오류: (지도 학습의 경우) 데이터를 분류하는 '라벨' 자체가 잘못되었을 수 있습니다. 인간 라밸러가 편견을 가지고 '악당' 캐릭터 이미지를 '어두운' 색상과 연관지어 라밸링했다면, 모델은 그 '잘못된' 관계를 학습합니다.

5.2 모델 편향을 줄이는 방법 (슬라이드 7)

편향 오류 줄이기

- 모델 변경: 데이터의 특성(예: 비선형 관계)에 맞지 않는 단순한 모델(예: 선형 회귀)을 사용하고 있다면, 더 복잡하거나 적절한 모델(예: 트리 기반 알고리즘)로 변경합니다.

- **데이터의 대표성 확보 (가장 중요):** 훈련 데이터가 모든 그룹(인종, 성별, 연령 등)을公正하게 대표하고 있는지 확인하고, 부족한 그룹의 데이터를 추가로 수집해야 합니다.
- **가중치 또는 페널티 모델 사용:** 데이터 불균형이 심한 경우(다음 섹션 참고), 소수 그룹에 더 높은 가중치(weight)를 부여하는 모델을 사용합니다.
- **앙상블 학습 (Ensemble Learning):** (편향보다는 분산을 줄이는 데 더 효과적이지만) 여러 모델(약한 학습기, 강한 학습기)을 결합하여 전반적인 모델의 견고함과 정확도를 높입니다.
- **더 많은 데이터로 훈련 (분산 줄이기):** 데이터가 많아지면 데이터 대 노이즈 비율이 증가하여, 모델이 특정 노이즈에 과대적합(Overfitting)되는 것을 방지하고 일반화 성능(분산 감소)을 높이는 데 도움이 됩니다.

6 데이터 불균형 문제 (Class Imbalance)

모델 편향의 특수한 경우로, 훈련 데이터의 클래스(정답) 분포가 한쪽으로 크게 치우친 상황을 말합니다.

- 예시: 신용카드 사기 탐지 (정상 99.9%, 사기 0.1%), 암 진단 (정상 98%, 암 2%)
- 산업 분야: 사기 탐지(Fraud), 클레임(Claim), 이상 징후(Anomaly), 침입 탐지(Intrusion) 등

6.1 불균형 데이터의 진짜 문제: '정확도'의 함정

대부분의 머신러닝 알고리즘은 **정확도(Accuracy)** (전체 중 몇 개를 맞혔는가)를 최대화하도록 설계되었습니다.

정확도(Accuracy)의 함정 사기 거래 비율이 0.1%인 데이터셋이 있다고 가정해 봅시다. 만약 어떤 모델이 모든 거래에 대해 "정상(Not Fraud)"이라고만 예측한다면?

이 모델의 정확도는 **99.9%**입니다. 정확도 지표상으로는 완벽해 보이지만, 우리가 잡고 싶은 '사기 거래'는 단 한 건도 잡아내지 못하는, 완전히 쓸모없는 모델입니다.

6.2 해결책 1: 평가 지표 변경 (Accuracy 대신 F1 Score)

불균형 데이터에서는 정확도 대신, 정밀도(Precision), 재현율(Recall), 그리고 이 둘을 조합한 **F1 Score**를 사용해야 합니다.

정밀도(Precision) vs 재현율(Recall)

- 정밀도 (Precision): 모델이 "사기"라고 예측한 것 중에, 실제 "사기"인 비율. ($TP / (TP+FP)$)
 - 중요한 경우: 스팸 메일 분류 (정상 메일을 스팸으로 잘못 분류하면 안 됨). 모델의 예측을 신뢰해야 할 때.
- 재현율 (Recall): 실제 "사기"인 것 중에, 모델이 "사기"라고 맞힌 비율. ($TP / (TP+FN)$)
 - 중요한 경우: 암 진단, 사기 탐지 (실제 암 환자나 사기 거래를 '정상'으로 놓치면 치명적임).
- F1 Score: 정밀도와 재현율의 조화 평균. $2 \times \frac{Precision \times Recall}{Precision + Recall}$
 - 두 지표가 모두 중요할 때 사용하는, 불균형 데이터의 핵심 평가지표입니다.

6.3 해결책 2: 리샘플링 (Resampling)

데이터의 분포를 인위적으로 조절하여 균형을 맞추는 기법입니다.

- 과소추출 (Undersampling): 다수 클래스(예: 정상 거래) 데이터를 대거 삭제하여 소수 클래스(사기 거래)와 비율을 맞춥니다.
 - 단점: 유용한 정보를 잃어버릴 위험이 큽니다.
- 과대추출 (Oversampling): 소수 클래스 데이터를 복제하여 수를 늘립니다.
 - 단점: 단순히 복제만 하면 모델이 특정 샘플에 과대적합(Overfitting)될 수 있습니다.

6.4 해결책 3: SMOTE (합성 소수 오버샘플링 기법)

단순 과대추출(Oversampling)의 과대적합 문제를 해결하기 위해 등장한 기법입니다.

SMOTE (Synthetic Minority Oversampling Technique) 작동 방식 SMOTE는 데이터를 '복제'하는 것 아니라 '생성'합니다. (슬라이드 12 그림 참고)

1. 소수 클래스(예: 검은색 원)에서 임의의 샘플(x_1)을 선택합니다.
2. 그 샘플(x_1)과 가장 가까운 이웃 샘플들(x_2, x_3 등)을 찾습니다.
3. 두 샘플을 잇는 직선 상에 무작위로 점(a, b, e)을 찍어, 이를 '새로운 합성 데이터'로 간주합니다.

이 방식을 통해 모델은 소수 클래스의 '특징 공간'을 더 넓고 견고하게 학습할 수 있습니다.

리샘플링 시 절대 주의사항 데이터 리샘플링(SMOTE 등)은 데이터를 훈련(Train) / 테스트(Test) 세트로 분리한 후, 훈련(Train) 세트에만 적용해야 합니다.

만약 전체 데이터에 SMOTE를 적용한 후 분리하면, 원본과 합성 데이터가 훈련/테스트 셋에 섞여 들어가기 때문에, 테스트 성능이 비정상적으로 높게 측정되는 '데이터 누수(Data Leakage)'가 발생합니다.

7 실전 사례 연구: 실시간 사기 탐지 시스템

강의 후반부는 이 모든 개념이 적용된, Eric Gieseke가 직접 참여했던 대규모 신용카드 사기 탐지 시스템 (Fraud Detection System) 구축 사례입니다.

7.1 비즈니스 요구사항

이 시스템의 성공 여부를 결정짓는 핵심 요구사항은 다음과 같습니다.

- **높은 신뢰도:** 실제 사기(Fraud)는 반드시 잡아내고(High Recall), 정상 거래를 사기로 오인(False Positive)해서는 안 됩니다. (정상 거래를 막으면 고객이 카드를 잘라버릴 수 있음)
- **초저지연 응답 속도 (Latency): → 15 밀리초 (ms)**
 - 고객이 카드를 긁고 POS 기기에서 응답을 받기까지의 총 시간(약 1~2초) 중, 사기 탐지 시스템에 할당된 시간입니다. 15ms는 일반적인 디스크(HDD)에서 데이터를 읽는 시간보다도 짧은, 극도로 도전적인 목표입니다.
- **높은 처리량 (Throughput): → 초당 50,000 건 (TPS)**
- **유연성:** 데이터 사이언티스트가 새로운 피처(Feature)나 룰(Rule)을 쉽게 추가하고 배포할 수 있어야 합니다.

7.2 피처(Feature) 정의

사기를 탐지하기 위해 원본 거래(Transaction) 데이터로부터 다양한 '파생 변수(피처)'를 계산합니다.

- **거래(Transaction) 기반 피처:**
 - 고객의 집 주소로부터의 거리 (예: 평소엔 서울인데 갑자기 이스탄불에서 결제)
 - 고객의 평균/최대/최소 거래 금액과의 차이 (예: 평소 1만원 쓰는데 갑자기 2,000만원 결제)
 - 당일 거래 횟수 (예: 평소 하루 2건인데 1시간 내 100건 결제)
- **차원(Dimension) 기반 피처:**
 - 판매자(Merchant)의 평균 거래 금액
 - 소비자(Consumer)의 거래 빈도

7.3 시스템 아키텍처: 람다 아키텍처 (Lambda Architecture)

15ms 응답 속도와 대용량 배치를 모두 만족시키기 위해, 람다(Lambda) 아키텍처를 채택했습니다. (슬라이드 19, 21 참고)

람다(Lambda) 아키텍처 대용량 데이터 파이프라인의 표준 중 하나로, 전체 시스템을 3개의 레이어로 분리합니다.

- **배치 레이어 (Batch Layer) (Hadoop/Spark)**
 - 모든 원본 데이터(Immutable)를 저장하고, 주기적으로(예: 매일 밤) 전체 데이터를 계산하여 '마스터 데이터셋'(피처 값)을 만듭니다. (느리지만 정확함)
 - 역할: 새로운 피처를 추가할 때, 과거 1년치 데이터에 대해 이 피처 값을 계산(Backfill)합니다.
- **스피드 레이어 (Speed Layer) (CEP / Realtime)**
 - 실시간으로 들어오는 데이터(이벤트)를 즉시 처리합니다. (빠르지만 근사치일 수 있음)

- 역할: '지금' 들어온 거래가 사기인지 아닌지 15ms 안에 판단합니다.
- 서빙 레이어 (Serving Layer) (Cassandra / Feature Store)
 - 배치 레이어와 스피드 레이어의 결과를 결합하여 저장하고, 실시간 쿼리에 응답합니다.

7.4 핵심 기술 1: 메타데이터 기반 코드 생성 (Slide 19)

- 문제: 수백 개의 피처를 배치(SQL)와 스피드(EPL) 환경을 위해 두 번씩 수동으로 코딩해야 했습니다. 이는 오류가 발생하기 쉽고 유지보수가 어렵습니다.
- 해결: 피처를 '코드'로 정의하지 않고, '메타데이터(Metadata)'로 정의했습니다.
 - (예) 메타데이터: ' $\text{Feature}_A = \text{AVG}(\text{TransactionAmount}, 7\text{days})$ '
- 코드 생성기(Code Generator)가 이 메타데이터를 읽어, 배치용 SQL 코드와 스트리밍용 EPL 코드를 자동으로 생성했습니다.
- 효과: 피처 관리가 용이해지고, 인간의 실수(Human Error)가 줄어들었습니다.

7.5 핵심 기술 2: 순환 버퍼 (Circular Buffer) (Slide 20)

- 문제: "특정 고객의 최근 7일간 평균 거래액"을 15ms 안에 계산해야 합니다. 수백만 명의 고객에 대해 최근 7일치 거래 내역 전체를 DB에서 읽어오는 것은 불가능합니다.
- 해결: 순환 버퍼(Circular Buffer)라는 독자적인(Proprietary) 데이터 구조를 고안했습니다.
- 작동 원리:
 - 고객별로 7일치(7칸) 버퍼를 만듭니다.
 - 각 칸(날짜)에 모든 거래 내역을 저장하는 대신, 오직 '거래 횟수(Count)'와 '거래 총액(Sum)' 두 값만 저장합니다.
 - (예: [월: (3 건, 30\$), 화: (5 건, 50\$), ..., 일: (5 건, 250\$)])
 - 새 날(월요일)이 되면, 8일 전의 데이터(지난주 월요일)를 버리고 새 데이터를 덮어씁니다.
- 효과: 이 컴팩트한 구조 덕분에, (Count의 합) / (Sum의 합) 연산만으로 7일간의 평균, 합계, 최대/최소값 등을 메모리상에서 매우 빠르게 계산할 수 있었습니다.

7.6 핵심 기술 3: 서빙 레이어 (Cassandra) (Slide 22)

- 문제: 순환 버퍼 외에도, 고객/판매자/터미널별 피처를 저장하고 15ms 내에 '읽어야' 할 저장소가 필요했습니다.
- 해결: Cassandra (NoSQL) 데이터베이스를 피처 스토어(Feature Store)로 채택했습니다.
- 이유: Cassandra는 쓰기(Write)도 빠르지만, 이 시스템에서는 극도로 빠른 읽기(Read) 속도 (약 2.3ms 달성) 때문에 선택되었습니다.
- 데이터 모델링 (Columnar):
 - RDBMS처럼 정규화된 테이블(고객 테이블, 판매자 테이블...)을 사용하지 않았습니다.
 - 단 2개의 거대한 테이블: 이벤트 테이블(Fact)과 차원 테이블(Dimension)을 사용했습니다.
 - Cassandra의 '컬럼(Column)' 기반 특성을 활용하여, 새로운 피처나 차원이 생길 때마다 단순히 새 컬럼을 추가하는 유연한 스키마(Flexible Schema)를 구현했습니다.

7.7 종합 아키텍처 (Slide 21)

Streaming Fraud Detection 시스템 흐름

1. (분석가) 사기 분석가가 GUI/CLI를 통해 '메타데이터 서비스'에 새로운 피처나 룰을 정의합니다. (예: 'Feature_B')
1. (배치) '배치 프로세싱(Spark)'이 이 메타데이터를 기반으로 코드를 생성, 과거 1년치 데이터에 대해 'Feature_B' ' (Cassandra)' .
1. (스트리밍) '스트림 분석(Stream Analytics)'이 이 메타데이터를 기반으로 실시간 코드를 업데이트합니다.
2. (고객) 고객이 '결제 서비스(Payment Service)'를 통해 카드 거래를 요청합니다.
3. (탐지) '사기 탐지 서비스(Fraud Detection Service)'가 거래 정보를 '스트림 분석'에 전달합니다.
4. (판단 15ms) '스트림 분석'은 15ms 이내에 다음을 수행합니다.
 - (a) 실시간 피처 계산 (순환 버퍼 등)
 - (b) 과거 피처 조회 (Cassandra에서 'Feature_B')
 - (c) ML 모델 및 룰 실행
 - (d) '사기/정상' 응답(Response) 반환
5. (승인) '사기 탐지 서비스'가 '정상' 응답을 '결제 서비스'에 전달하고, 고객의 거래가 승인됩니다.

8 학습 점검을 위한 체크리스트

이번 강의 내용을 스스로 점검해 봅시다.

핵심 개념 체크리스트

모델 오류의 세 가지 구성요소(편향, 분산, 출일 수 없는 오류)를 설명할 수 있는가?

편향-분산 트레이드오프(Trade-off)가 무엇인지, 왜 중요한지 이해하는가?

과소적합(Underfitting)과 과대적합(Overfitting)이 각각 편향, 분산과 어떻게 연결되는지 아는가?

안면 인식 사례에서 '어두운 피부의 여성'에 대한 정확도가 낮았던 근본적인 원인을 설명할 수 있는가?

데이터 불균형 문제에서 '정확도(Accuracy)'를 평가지표로 쓰면 안 되는 이유를 설명할 수 있는가?

'정밀도(Precision)'와 '재현율(Recall)'의 차이를 암 진단(Cancer) 예시로 설명할 수 있는가?

SMOTE가 무작위 오버샘플링(Random Oversampling)과 어떻게 다른지 설명할 수 있는가?

람다(Lambda) 아키텍처가 왜 배치(Batch) 레이어와 스피드(Speed) 레이어를 둘 다 사용하는지 아는가?

사기 탐지 시스템이 15ms의 응답 속도를 맞추기 위해 사용한 핵심 기술 2가지(순환 버퍼, Cassandra)가 어떤 문제를 해결했는지 아는가?

9 FAQ (자주 묻는 질문)

초심자가 혼동할 수 있는 내용들을 Q&A 형식으로 정리합니다.

Q: AutoML을 쓰면 데이터 사이언티스트가 필요 없나요? A: 아닙니다. AutoML은 '시민 데이터 과학자'가 좋은 출발점 (Baseline)을 얻거나, 데이터의 유용성을 빠르게 검증하는 데 매우 유용한 도구입니다.

하지만 'Blackbox' AutoML은 내부 수정이 어렵고, 'Glassbox' AutoML이라 할지라도, 결국 비즈니스 문제를 정의하고, 어떤 데이터를 사용할지 결정하며, 모델의 결과를 해석하여 비즈니스에 적용하는 것은 여전히 데이터 사이언티스트와 도메인 전문가의 핵심 역량입니다.

Q: 모델 편향은 알고리즘만 수정하면 해결되나요? A: 아닙니다. 편향의 근본 원인은 알고리즘 자체가 아니라 '데이터'에 있는 경우가 압도적으로 많습니다. (예: 특정 그룹 데이터 부족, 데이터 라벨링의 편견)

알고리즘 수정(예: 가중치 부여)도 한 가지 방법이지만, 가장 중요한 해결책은 훈련 데이터 자체가 현실 세계를 공정하고 다양하게 대표할 수 있도록 데이터 수집 및 정제 과정에 투자하는 것입니다.

Q: 15ms (0.015초) 안에 어떻게 그 많은 계산을 하나요? A: '모든' 계산을 15ms 안에 하는 것이 아닙니다. 핵심은 '미리 계산(Pre-computation)'과 '빠른 조회(Fast Retrieval)'입니다.

- (1) 미리 계산: '고객의 1년치 평균 거래액'처럼 오래 걸리는 피처는 배치 레이어 (Spark)가 매일 밤 미리 계산해서 Cassandra에 저장해둡니다.
- (2) 빠른 조회: 실시간 요청이 오면, 이 미리 계산된 값을 Cassandra에서 2.3ms 만에 조회합니다.
- (3) 실시간 계산: '최근 10분간 거래 횟수'처럼 실시간성이 강한 피처만 순환 버퍼 (Circular Buffer) 같은 메모리 기반 구조를 이용해 즉석에서 계산합니다.

결국 15ms는 이 '조회 + 최소한의 실시간 계산 + 모델 추론'을 합친 시간입니다.

10 빠르게 훑어보기 (1-Page 요약)

강의 10 핵심 요약 카드

오류 1: 편향(Bias) vs 분산(Variance)

편향(과소적합): 모델이 너무 단순함. (훈련/테스트 모두 실패)
분산(과대적합): 모델이 너무 복잡함. (훈련만 성공, 테스트 실패)
Trade-off: 둘은 반비례 관계. 둘의 균형점을 찾는 것이 목표.

오류 2: 모델 편향(Model Bias)

원인 1: 인간의 편견: 데이터 자체가 편견을 가짐 (예: 검색 결과)
원인 2: 데이터 부족: 특정 그룹(예: 어두운 피부 여성) 데이터가 부족(Under-represented)하여 모델이 제대로 학습하지 못함. **해결:** 데이터의 대표성과 다양성 확보가 최우선.

오류 3: 데이터 불균형(Imbalance)

문제: 사기(1%) vs 정상(99%). '정확도(Accuracy)'는 99%짜리 가짜 모델을 만들.
해결 1 (평가): F1 Score (정밀도+재현율) 사용. (특히 재현율이 중요!) **해결 2 (데이터):** SMOTE 기법으로 소수 클래스의 '합성' 데이터를 생성하여 밸런스 맞추기. (단, Train 셋에만 적용!)

실전: 사기 탐지 시스템 (15ms 응답)

아키텍처: 람다(Lambda) (배치 + 스피드) **핵심 1 (계산):** 순환 버퍼(Circular Buffer) 사용. ('Count'와 'Sum'만 저장하여 메모리에서 초고속 접근) **핵심 2 (저장):** Cassandra (NoSQL) 사용. (2.3ms의 초고속 읽기 속도로 피쳐 조회) **핵심 3 (유지보수):** 코드 생성기 (메타데이터로 피쳐 정의)

■ 강의명: CSCI E-103: 재현 가능한 머신러닝

■ 주차: Lecture 11

■ 교수명: Anindita Mahapatra

Eric Gieseke

■ 목적: Lecture 11의 핵심 개념 학습

Contents

1 개요: AI의 진화와 LLM 개발의 현재	2
2 필수 용어 정리	2
3 핵심 개념: AI의 진화와 도입 전략	3
3.1 1. AI의 포함 관계 (Venn Diagram)	3
3.2 2. LLM 성숙도 모델 (Maturity Curve)	3
4 기술 심화: RAG (검색 증강 생성)	4
4.1 1. 왜 RAG가 필요한가?	4
4.2 2. RAG의 작동 원리 (Workflow)	4
5 실무: Databricks 도구를 활용한 구현	4
5.1 1. Genie (지니)	4
5.2 2. Agent Bricks (에이전트 브릭스)	5
5.3 3. AI Functions (SQL에서 AI 쓰기)	6
5.4 4. LLMOps (LLM 운영)	6
6 운영과 윤리: 리스크와 대응	6
7 학습 체크리스트 및 공지사항	6
7.1 주요 일정 (강의 공지)	6
7.2 FAQ: 자주 묻는 질문	7
8 빠르게 훑어보기: 1페이지 핵심 요약	8

1 개요: AI의 진화와 LLM 개발의 현재

이 문서는 대규모 언어 모델(LLM)과 에이전트(Agent)를 활용하여 실제 비즈니스 가치를 창출하는 방법을 다룹니다. 단순히 ”신기한 기술”을 넘어, 데이터를 통합하고 배포하며 운영하는 전체 수명주기 (LLMOps)를 이해하는 것이 목표입니다.

▣ 핵심 정보

학습 목표

- 이론:** AI/ML의 진화 과정과 생성형 AI(GenAI)의 차별점 이해
- 전략:** 조직의 성숙도에 따른 LLM 도입 단계 (Prompting → RAG → Fine-tuning)
- 실무:** Databricks의 도구(Genie, Agent Bricks, Unity Catalog)를 활용한 에이전트 구축
- 운영:** LLMOps, 환각(Hallucination) 관리, 윤리적 AI 사용

2 필수 용어 정리

생성형 AI 분야는 새로운 용어가 많습니다. 아래 표는 이 문서를 이해하기 위한 핵심 단어장입니다.

용어 (약어)	원어	쉬운 설명 (직관적 비유)
LLM	Large Language Model	인터넷의 모든 텍스트를 읽고 언어 패턴을 익힌 ‘초거대 독서광 AI’
GenAI	Generative AI	데이터를 분류하는 것을 넘어, 새로운 텍스트/이미지를 ‘창조’하는 AI
Hallucination	Hallucination	AI가 사실이 아닌 내용을 마치 사실인 것처럼 ‘뻔뻔하게 거짓말’하는 현상
RAG	Retrieval Augmented Generation	AI에게 ‘오픈북 테스트’를 치르게 하듯, 참고 자료를 검색해서 보여주고 답하게 하는 기술
Grounding	Grounding	AI의 답변을 현실 세계의 사실이나 데이터에 ‘단단히 묶어두는’ 과정
Fine-Tuning	Fine-Tuning	일반적인 지식을 가진 AI에게 특정 분야(예: 법률, 의학)를 ‘심화 과외’ 시키는 것
Embeddings	Embeddings	텍스트의 의미를 컴퓨터가 이해할 수 있는 ‘숫자 좌표(벡터)’로 변환하는 기술
Vector DB	Vector Database	의미(Embedding)를 저장하여, 키워드가 달라도 ‘문맥상 유사한’ 자료를 찾아주는 검색 엔진
Chain of Thought	Chain of Thought	AI에게 “답만 말하지 말고 ‘풀이 과정’을 단계별로 써라”고 시키는 기법

Table 1: LLM 및 생성형 AI 핵심 용어 요약

3 핵심 개념: AI의 진화와 도입 전략

3.1 1. AI의 포함 관계 (Venn Diagram)

AI 기술은 갑자기 튀어나온 것이 아니라, 수십 년간 발전해 온 기술의 집약체입니다.

- **AI (Artificial Intelligence):** 인간의 지능을 모방하는 모든 시스템 (가장 큰 범주)
- **ML (Machine Learning):** 데이터를 통해 학습하는 AI의 하위 분야
- **Deep Learning (Neural Networks):** 인간의 뇌 신경망을 모방한 복잡한 ML
- **GenAI (Generative AI):** 기존 데이터를 분석하는 것을 넘어, 새로운 데이터를 생성하는 딥러닝의 최신 분야
 - **LLM:** 텍스트 생성 특화 (예: GPT, Gemini, Claude)
 - **GAN:** 이미지/비디오 생성 특화 (예: Deepfake, StyleGAN)

3.2 2. LLM 성숙도 모델 (Maturity Curve)

기업이나 개인이 LLM을 도입할 때, 무조건 처음부터 자체 모델을 만드는 것은 비효율적입니다. 비용과 복잡도를 고려하여 단계별로 접근해야 합니다.

LLM 도입의 4단계 (쉬움 → 어려움)

1. Prompt Engineering (프롬프트 엔지니어링)

- **설명:** AI에게 일을 시키는 명령어를 정교하게 다듬는 단계.
- **비용/난이도:** 매우 낮음 / 데이터 필요 없음.
- **한계:** AI가 학습하지 않은 최신 정보나 사내 비공개 정보는 모름.

2. RAG (검색 증강 생성) - *가장 권장되는 단계

- **설명:** 사내 문서나 데이터베이스를 검색(Retrieval)하여 그 내용을 AI에게 참고자료로 주고 답변(Generation)하게 함.
- **장점:** 최신 정보 반영 가능, 할루시네이션 감소, 가성비 최고.
- **비유:** 시험 보는 학생(AI)에게 교과서(사내 데이터)를 펼쳐놓고 답을 쓰게 하는 것.

3. Fine-Tuning (파인 투닝)

- **설명:** 기존 모델에 우리만의 데이터로 추가 학습을 시키는 것.
- **용도:** 특수한 말투, 전문 용어, 특정 형식의 답변이 필요할 때.
- **비용:** 데이터 준비와 학습에 상당한 비용 발생.

4. Pre-Training (사전 학습)

- **설명:** 처음부터 모델을 바닥부터 만드는 것 (예: 자체 GPT 만들기).
- **현실:** 구글, 메타급 기업이 아니면 비용(수십 수백 억 원) 문제로 거의 불가능.

4 기술 심화: RAG (검색 증강 생성)

LLM의 가장 큰 약점인 **”최신 정보 부재”**와 **”할루시네이션(거짓말)”**을 해결하는 가장 현실적인 기술입니다.

4.1 1. 왜 RAG가 필요한가?

LLM은 학습이 끝난 시점(예: 2023년) 이후의 정보는 모릅니다. 또한, 우리 회사의 비공개 문서는 당연히 본 적이 없습니다. 이를 해결하기 위해 모델을 재학습시키는 것은 너무 비쌉니다. RAG는 **”외부 지식 검색”**을 통해 이 문제를 해결합니다.

4.2 2. RAG의 작동 원리 (Workflow)

RAG 시스템은 크게 데이터 준비(Ingestion)와 검색 및 답변(Retrieval & Generation) 두 단계로 나뉩니다.

- 단계 1: 데이터 준비 (사전 작업)

1. 문서를 잘게 쪼갭니다 (Chunking). (예: 800~1200자 단위)
2. 쪼갠 문서를 AI가 이해하는 숫자 배열(Vector/Embedding)로 변환합니다.
3. 이 숫자를 **Vector DB**에 저장합니다. (유사도 검색을 위해)

- 단계 2: 실제 질문 시 (실시간)

1. 사용자가 질문을 합니다. (“우리 회사의 재택근무 규정이 뭐야?”)
2. 질문을 숫자(Vector)로 변환합니다.
3. Vector DB에서 질문과 숫자가 가장 비슷한(의미가 유사한) 문서 조각을 찾아냅니다.
4. 찾아낸 문서 조각을 **프롬프트**에 붙여서 **LLM**에게 보냅니다.
5. LLM은 ”아래 문서를 참고해서 답변해”라는 지시를 받고 정확한 답을 생성합니다.

초심자가 자주 하는 오해 Q. RAG를 쓰면 LLM이 학습을 하나요?

A. 아닙니다! RAG는 LLM에게 잠시 참고자료를 보여주는 것뿐입니다. LLM의 뇌(모델 파라미터) 자체는 변하지 않습니다. 마치 시험 때 오픈북을 한다고 해서 학생의 지능이 영구적으로 변하는 것은 아닌 것과 같습니다.

5 실무: Databricks 도구를 활용한 구현

Databricks 플랫폼은 LLM 애플리케이션 개발을 위한 통합 도구를 제공합니다.

5.1 1. Genie (지니)

- 정의: 정형 데이터(SQL 테이블)와 대화하는 에이전트.
- 특징: 텍스트로 질문하면 자동으로 SQL 쿼리를 생성하여 답을 줍니다.
- 장점: 데이터의 메타데이터(스키마)만 보고 SQL을 짜기 때문에, 일반 LLM보다 할루시네이션이 적고 정확도가 높습니다.
- 사용 예: ”지난달 매출이 가장 높은 제품 5개 보여줘” → SELECT name, sales FROM ... 자동 실행

5.2 2. Agent Bricks (에이전트 브릭스)

- 정의: 코딩 없이 클릭 몇 번으로 RAG 에이전트를 만드는 'AutoML' 같은 도구.
- 기능: PDF 파일이 있는 경로만 지정하면, 자동으로 벡터 DB를 구축하고 챗봇을 생성해줍니다.
- 활용: 고객 응대 챗봇, 사내 규정 검색기 등을 10분 만에 프로토타이핑 가능.

5.3 3. AI Functions (SQL에서 AI 쓰기)

복잡한 파이썬 코드 없이, SQL 쿼리 안에서 바로 LLM 기능을 호출할 수 있습니다.

```

1 -- 고객리뷰의 감정을 분석하고 요약하는      SQL 쿼리
2 SELECT
3     review_text,
4     ai_analyze_sentiment(review_text) AS sentiment, -- 긍정부정/ 분석
5     ai_summarize(review_text, 10) AS summary          -- 단어10 요약
6 FROM customer_reviews
7 WHERE date > '2025-01-01';

```

Listing 1: AI Functions 활용 예시

5.4 4. LLMOps (LLM 운영)

기존 MLOps(머신러닝 운영)에 **"언어적 특성"**이 추가된 개념입니다.

- **모델 평가:** 답변이 정확한지, 독성이 있는지 평가 (LLM을 심판으로 사용하기도 함).
- **피드백 루프 (RLHF):** 사용자의 "좋아요/싫어요" 버튼 클릭을 모아 모델을 개선.
- **보안:** 프롬프트 인젝션(해킹) 방지, 개인정보 유출 방지.

6 운영과 윤리: 리스크와 대응

LLM은 강력하지만 위험 요소도 큽니다. 이를 관리하는 것이 엔지니어의 핵심 역량입니다.

리스크 유형	설명	대응 방안
환각 (Hallucination)	없는 사실을 지어냄	RAG 사용, Temperature(창의성 수치)를 0으로 설정
편향 (Bias)	특정 인종/성별에 차별적 발언	학습 데이터의 다양성 확보, 필터링 가드레일 설치
보안 위협	프롬프트 인젝션 (AI 속이기)	입력값 검증, 관리자 권한 분리
개인정보 유출	학습된 민감 정보 노출	학습 데이터 정제(Masking), 사내 전용 모델 사용

Table 2: LLM 주요 리스크 및 대응 방안

▣ 핵심 정보

ESG와 AI 최근 기업은 **ESG(환경, 사회, 지배구조)** 관점에서 AI를 평가합니다. AI 모델 학습에 막대한 전기가 소모(환경)되거나, AI가 차별적 발언(사회)을 하는 것은 기업 가치에 치명적입니다. 따라서 "윤리적 AI"는 선택이 아닌 필수입니다.

7 학습 체크리스트 및 공지사항

7.1 주요 일정 (강의 공지)

- **성적 공개:** Quiz-1, Assignment-3 채점 완료.
- **과제:** Use Case-1, Assignment-4 곧 채점 예정.
- **최종 프로젝트:**

- 11월 18일: 프로젝트 주제 및 세부 내용 공개.
- 12월 16일: 최종 발표 (팀원 전원 참석 권장).
- **기타:** 다음 주는 추수감사절(Thanksgiving)로 휴강. 12월 9일 산업계 전문가 초청 강연.

7.2 FAQ: 자주 묻는 질문

Q. RAG를 쓸 때, 문서가 업데이트되면 어떻게 하나요?

A. Vector DB를 업데이트해야 합니다. 문서를 통째로 교체하거나 변경된 부분만 다시 임베딩하여 덮어 써야 합니다. 다만, 부분 업데이트 시 버전 관리가 복잡해질 수 있습니다.

Q. LLM이 항상 최신 정보를 알 수 있나요?

A. 순수 LLM은 불가능합니다. RAG나 웹 검색 엔진트를 붙여야만 실시간 정보를 반영할 수 있습니다.

Q. 프롬프트 엔지니어링만으로 충분한가요?

A. 간단한 업무는 충분합니다. 하지만 전문적인 지식이 필요하거나 사내 데이터를 써야 한다면 RAG가 필수적입니다.

8 빠르게 훑어보기: 1페이지 핵심 요약

▣ 핵심 요약

1. **AI의 흐름:** Rule-based → ML → Deep Learning → **GenAI(생성형 AI)**.
2. **LLM 활용 전략:** 무작정 모델을 만드는(Pre-train) 것이 아니라, **RAG(검색 증강)**를 통해 사내 데이터를 연결하는 것이 가장 효율적임.
3. **RAG 핵심:** 문서 쪼개기(Chunking) → 숫자 변환(Embedding) → Vector DB 저장 → 유사도 검색 → 답변 생성.
4. **Databricks 도구:**
 - **Genie:** 정형 데이터(SQL) 담당, 정확도 높음.
 - **Agent Bricks:** 비정형 데이터(PDF 등) 담당, 빠른 구축.
 - **AI Gateway:** 모델 보안 및 관리의 관문.
5. **주의사항:** 할루시네이션(거짓말) 방지, 데이터 보안, 윤리적 책임이 기술 구현만큼 중요함.
6. **미래:** 단순 챗봇을 넘어, 스스로 도구를 선택하고 행동하는 **에이전트(Agent)** 시대로 진입 중.

부록: 초심자를 위한 Agent 구축 단계 (Pseudo-Flow)

Databricks에서 RAG 에이전트를 만드는 사고의 흐름입니다.

1. **데이터 준비:** PDF 파일들을 Unity Catalog Volume에 업로드.
2. **인덱싱 (Vector Search):** ”이 파일들을 읽어서 검색 가능하게 만들어줘” (Agent Bricks 활용 시 클릭 몇 번으로 완료).
3. **에이전트 생성:** ”이 인덱스를 참고하는 챗봇을 만들어.”
4. **테스트 (Playground):** 채팅창에서 질문해보고 답변 확인.
 - 답변이 이상하다? → 프롬프트 수정 (“전문가처럼 답변해”, “모르면 모른다고 해”).
 - 근거가 없다? → 데이터(PDF) 보강.
5. **배포:** Review App을 통해 사용자들에게 링크 공유 및 피드백 수집.

■ 강의명: CSCI E-103: 재현 가능한 머신러닝

■ 주차: Lecture 12

■ 교수명: Anindita Mahapatra

Eric Gieseke

■ 목적: Lecture 12의 핵심 개념 학습

Contents

1 개요 (Overview)	2
2 용어 정리 (Terminology)	3
3 핵심 개념과 원리	4
3.1 1. 정보 거버넌스 vs 데이터 거버넌스 vs AI 거버넌스	4
3.2 2. 성숙도 모델 (Maturity Models)	4
3.3 3. 거버넌스와 민첩성의 균형 (Trade-off)	4
4 기술적 구현: Databricks Unity Catalog (UC)	5
4.1 1. 계층 구조 (Hierarchy)	5
4.2 2. 관리형 테이블 vs 외부 테이블 (Managed vs External)	5
4.3 3. Lakehouse Federation (연합)	5
4.4 4. Delta Sharing (공유)	5
5 고급 기능: 보안과 자동화 (Security & Automation)	6
5.1 1. 데이터 분류 (Data Classification)	6
5.2 2. 속성 기반 접근 제어 (ABAC)	6
5.3 3. 데이터 품질 모니터링 (Data Quality Monitoring)	6
5.4 4. 리니지 (Lineage)	6
6 거버넌스 운영 모델 (Operational Model)	7
7 자주 묻는 질문 (FAQ)	7
8 마무리 요약 및 체크리스트	8

1 개요 (Overview)

이 문서는 **Data & AI Governance(데이터 및 AI 거버넌스)**의 핵심 이론과 이를 구현하는 기술 (Databricks Unity Catalog 등)을 다룹니다.

▣ 핵심 요약

핵심 요약

- **목표:** 데이터의 가치를 최대화하면서 동시에 보안 및 규제 리스크를 최소화하는 것.
- **변화:** 과거에는 데이터만 관리했지만, 이제는 AI 모델과 그 결과물까지 관리 범위가 확장됨.
- **도구:** Databricks의 **Unity Catalog**를 중심으로 중앙화된 접근 제어, 감사(Audit), 혈통(Lineage) 추적을 구현.
- **핵심 기능:** 데이터를 자동으로 분류(Tagging)하고, 속성 기반 접근 제어(ABAC)를 통해 민감 정보를 마스킹하며, 외부 데이터베이스까지 통합 관리(Federation) 함.

왜 이 내용을 배워야 하나요?

데이터는 '새로운 원유'라고 불릴 만큼 가치가 큽니다. 하지만 관리가 안 된 데이터는 유출 사고 시 기업에 막대한 벌금과 신뢰 하락을 초래합니다. 데이터를 안전하게 지키면서도 필요할 때 바로 꺼내 쓸 수 있게 만드는 '규칙'과 '시스템'을 배우는 것이 이 강의의 목표입니다.

2 용어 정리 (Terminology)

초심자가 혼동하기 쉬운 핵심 용어를 정리했습니다.

Table 1: 거버넌스 핵심 용어 비교표

gray!20 용어 (한글/영문)	쉬운 설명 (비유)	기술적 의미
거버넌스 (Governance)	교통 법규: 사고 안 나게 정한 규칙	정책, 표준, 절차를 수립하여 조직을 통제하는 프레임워크
관리 (Management)	운전 행위: 법규를 지키며 실제 운전함	거버넌스에서 정한 규칙을 매일 실행하는 운영 활동
메타스토어 (Metastore)	도서관 목록 카드: 책 위치 정보 저장소	데이터의 구조(스키마), 위치, 권한 정보를 저장하는 최상위 컨테이너
Unity Catalog (UC)	통합 신분증: 어디서든 통하는 ID 카드	Databricks에서 파일, 테이블, 모델 등 모든 자산을 통합 관리하는 계층
ABAC	꼬리표 검사: ”빨간 딱지 붙은 건 못 봐”	속성(Attribute) 기반 접근 제어. 태그(Tag)를 통해 권한을自動화함
리니지 (Lineage)	족보/가계도: 이 데이터의 조상은 누구?	데이터가 어디서 생성되어 어떻게 변환되고 어디로 흘러갔는지 추적
Federation	대사관: 남의 땅에 있지만 우리 법 적용	외부 DB(Snowflake 등)의 데이터를 복사하지 않고 연결하여 조회/관리

3 핵심 개념과 원리

3.1 1. 정보 거버넌스 vs 데이터 거버넌스 vs AI 거버넌스

거버넌스는 범위에 따라 크게 세 가지 층위로 나뉩니다. 가장 큰 우산이 '정보 거버넌스'입니다.

정보 거버넌스 (Information Governance)

조직의 모든 정보(종이 서류, 디지털 파일, 지식 등)를 관리하는 가장 큰 개념입니다.

- 예: "퇴근할 때 책상 위 기밀 서류 치우기", "노트북 잡금 화면 설정하기"

데이터 거버넌스 (Data Governance)

디지털 데이터의 품질, 보안, 수명 주기를 관리합니다.

- 목표: 데이터의 정확성, 일관성, 신뢰성 확보.
- 예: "고객 테이블에 접근할 수 있는 사람은 누구인가?", "이 데이터는 암호화되었는가?"

AI 거버넌스 (AI Governance)

AI 모델의 개발, 배포, 윤리적 사용을 관리합니다. 데이터 거버넌스 없이는 불가능합니다.

- 목표: 편향성(Bias) 방지, 설명 가능성(Explainability), 모델의 투명성.
- 예: "이 AI 모델이 특정 인종에게 불리한 대출 심사를 하지는 않는가?", "학습 데이터에 저작권 위반 데이터가 포함되었는가?"

주의: AI 거버넌스는 데이터 거버넌스 위에 쌓입니다 "걷기도 전에 뛸 수 없다"는 말처럼, 데이터가 엉망인 상태(데이터 거버넌스 부재)에서는 안전하고 공정한 AI(AI 거버넌스)를 만들 수 없습니다.

3.2 2. 성숙도 모델 (Maturity Models)

조직이 거버넌스를 얼마나 잘하고 있는지 5단계로 평가합니다. 단계를 건너뛰는 것(Skip)은 보통 실패합니다. 차근차근 밟아 올라가야 합니다.

- 초기/인지 (Initial/Aware): 규칙 없음. 스타트업 초기 단계. 각자 알아서 함.
- 반응형 (Managed/Reactive): 문제가 터지면 수습함. 부분적인 규칙 존재.
- 정의됨 (Defined/Proactive): 전사적인 표준과 프레임워크가 잡힘. (대부분의 기업 목표)
- 정량화 (Quantified): 거버넌스 성과를 수치로 측정 가능함.
- 최적화 (Optimized): AI가 자동으로 위반 사항을 감지하고 조치함. 지속적 개선.

3.3 3. 거버넌스와 민첩성의 균형 (Trade-off)

모든 데이터를 금고에 가두면 안전하지만 아무도 일을 못 합니다(민첩성 저하). 반대로 다 풀어주면 빠르지만 위험합니다.

- 고위험 데이터 (개인정보, 금융정보): 엄격한 통제 (Strict Governance). 접근 절차가 까다로움.
- 저위험 데이터 (공개 데이터, 실험용 데이터): 느슨한 통제 (Permissive Governance). 혁신을 위해 빠르게 접근 허용.

4 기술적 구현: Databricks Unity Catalog (UC)

이론을 실제 시스템으로 구현하는 도구입니다. Databricks는 **Unity Catalog**라는 단일 계층을 통해 모든 데이터와 AI 자산을 관리합니다.

4.1 1. 계층 구조 (Hierarchy)

UC는 3단계 구조로 데이터를 정리합니다. (파일 시스템의 폴더 구조와 비슷합니다.)

Metastore $\xrightarrow{\text{포함}}$ Catalog $\xrightarrow{\text{포함}}$ Schema (Database) $\xrightarrow{\text{포함}}$ Table / Volume / Model

- **Metastore:** 최상위 컨테이너. 보통 리전(Region) 당 하나.
- **Catalog:** 데이터 자산의 가장 큰 그룹. (예: ‘prod’, ‘dev’, ‘hrdata’)
- **Schema:** 테이블과 뷰를 담는 논리적 그룹.
- **Table/Volume:** 실제 데이터. (Table은 정형 데이터, Volume은 비정형 파일)

4.2 2. 관리형 테이블 vs 외부 테이블 (Managed vs External)

- **Managed Table:** Databricks가 데이터 파일의 위치와 수명 주기를 직접 관리합니다. 테이블을 지우면 실제 파일도 지워집니다. (가장 추천됨)
- **External Table:** 데이터 파일은 내 클라우드 스토리지(S3 등)에 있고, Databricks는 그 위치만 참조합니다. 테이블을 지워도 원본 파일은 남습니다.

4.3 3. Lakehouse Federation (연합)

외부 데이터베이스(MySQL, Snowflake, Postgres 등)를 데이터를 복사해오지 않고(No Copy), 마치 로컬 테이블처럼 조회하는 기능입니다.

□ 예제:

비유: 대사관 Databricks 안에 있는 'Snowflake Catalog'는 대사관과 같습니다. 실제 영토(데이터)는 Snowflake에 있지만, Databricks의 법(거버넌스 규칙)을 적용하여 조회할 수 있습니다. 쿼리를 날리면 Databricks가 처리하는 게 아니라 원본 DB(Snowflake)로 쿼리를 보내서 결과만 받습니다 (Pushdown).

4.4 4. Delta Sharing (공유)

서로 다른 플랫폼 간에 데이터를 안전하게 공유하는 개방형 프로토콜입니다.

- 데이터를 복제해서 이메일로 보내거나 FTP로 전송할 필요가 없습니다.
- 받는 사람이 Databricks를 안 써도 됩니다(Pandas, Tableau, PowerBI 등으로 직접 접속 가능).

5 고급 기능: 보안과 자동화 (Security & Automation)

5.1 1. 데이터 분류 (Data Classification)

시스템이 자동으로 데이터를 스캔하여 이메일, 신용카드 번호 같은 민감 정보(PII)를 찾아냅니다. 찾으면 자동으로 태그(Tag)를 붙입니다.

5.2 2. 속성 기반 접근 제어 (ABAC)

과거에는 ”철수에게 A 테이블 권한 주기” 식(RBAC)이었다면, 이제는 ”‘기밀’ 태그가 붙은 건 관리자만 보기” 식으로 규칙을 만듭니다.

- **Row Level Filtering:** 특정 조건을 만족하는 행(Row)만 보여줌. (예: 내 부서 데이터만 보기)
- **Column Level Masking:** 특정 열(Column)의 데이터를 가림. (예: 주민번호 뒷자리 별표 처리)

```

1 -- 1. 마스킹함수정의관리자가      ( 아니면별표로표시 )
2 CREATE FUNCTION ssn_mask(ssn STRING)
3 RETURN IF(
4     is_account_group_member('admin'), -- 관리자그룹인지확인
5     ssn,                            -- 관리자면원본노출
6     '*****'                         -- 아니면별표표시
7 );
8
9 -- 2. 테이블의특정컬럼에마스킹함수적용
10 ALTER TABLE users
11 ALTER COLUMN social_security_number
12 SET MASK ssn_mask;

```

Listing 1: SQL을 이용한 마스킹 정책 생성 예시

5.3 3. 데이터 품질 모니터링 (Data Quality Monitoring)

데이터가 썩지 않았는지 감시합니다.

- **Freshness (최신성):** 데이터가 제때 들어왔는가?
- **Completeness (완전성):** 데이터 양이 갑자기 줄지 않았는가? (어제는 1000행이었는데 오늘은 0행?)
- 별도 설정 없이 버튼 클릭 한 번으로 자동 감시가 가능합니다.

5.4 4. 리니지 (Lineage)

데이터의 가계도입니다. ”이 차트의 숫자가 왜 이상하지?”라는 질문이 나왔을 때, 그 데이터가 어떤 원천 테이블에서 와서 어떤 변환 과정을 거쳤는지 시각적으로 보여줍니다. (Source → Transform → Dashboard)

6 거버넌스 운영 모델 (Operational Model)

조직의 크기와 문화에 따라 거버넌스를 누가 주도할지 결정해야 합니다. 정답은 없으며, 조직 상황에 맞춰 선택합니다.

Table 2: 거버넌스 운영 모델 비교

gray!20 모델	특징	장점	단점/적합 조직
중앙 집중형	중앙 팀이 모든 규칙 결정	일관성 높음, 보안 강력	느림, 병목 현상 발생 (규제 산업)
분산형	각 부서가 알아서 관리	혁신 속도 빠름, 유연함	표준 없음, 중복 발생 (스타트업)
연합형 (Federated)	중앙 가이드 + 부서 실행	자율성과 통제의 균형	조율이 어려울 수 있음 (대기업)
하이브리드	핵심 데이터는 중앙, 나머지는 분산	중요 정보 보호 + 업무 효율	구조가 복잡함

7 자주 묻는 질문 (FAQ)

Q. 클라우드(AWS/Azure)의 IAM 역할만 쓰면 안 되나요? 왜 Unity Catalog가 필요한가요?

A. 클라우드 IAM은 '파일'이나 '폴더' 단위의 접근은 제어할 수 있지만, '테이블의 특정 컬럼'이나 '행' 단위의 정교한 제어는 어렵습니다. UC는 데이터 내용에 기반한 세밀한(Fine-grained) 제어를 가능하게 합니다.

Q. 데이터 거버넌스를 도입하면 업무 속도가 느려지지 않나요?

A. 초기에는 규칙 설정 때문에 느려 보일 수 있습니다. 하지만 장기적으로는 '데이터를 찾는 시간', '데이터 품질을 의심하고 검증하는 시간', '보안 사고 수습 시간'을 줄여주어 전체적인 속도는 빨라집니다.

Q. ABAC과 RBAC의 차이는 무엇인가요?

A. **RBAC(Role-Based)**은 "팀장님은 다 볼 수 있어"처럼 역할에 권한을 줍니다. **ABAC(Attribute-Based)**은 "누구든 '기밀' 태그가 붙은 건 못 봐"처럼 데이터의 속성(태그)에 따라 동적으로 권한을 줍니다. ABAC이 더 유연하고 확장성이 좋습니다.

8 마무리 요약 및 체크리스트

이 문서를 통해 학습한 내용을 점검해 보세요.

학습 체크리스트

- 정보, 데이터, AI 거버넌스의 포함 관계를 이해했는가?
- 성숙도 모델 5단계를 순서대로 나열할 수 있는가?
- Unity Catalog의 3계층 구조(Metastore-Catalog-Schema)를 그릴 수 있는가?
- ABAC의 개념과 왜 태깅(Tagging)이 중요한지 설명할 수 있는가?
- Lakehouse Federation이 데이터를 복사하지 않고(No Copy) 조회한다는 의미를 아는가?
- 데이터 품질 모니터링의 두 가지 핵심 지표(Freshness, Completeness)를 이해했는가?

▣ 핵심 요약

- 1페이지 요약 1. 거버넌스는 전략이다: 단순한 제약이 아니라, 데이터를 자산으로 만들기 위한 필수 전략입니다.
2. 통합 관리: Unity Catalog를 통해 파일, 테이블, 모델을 한 곳에서 관리합니다.
3. 자동화: AI를 활용해 민감 정보를 자동 분류하고, 품질을 모니터링합니다.
4. 연결성: Federation과 Sharing을 통해 데이터 파일로(고립)를 제거하고 외부와 연결합니다.
5. 균형: 보안(Security)과 민첩성(Agility) 사이에서 조직에 맞는 적절한 운영 모델을 선택해야 합니다.

■ 강의명: CSCI E-103: 재현 가능한 머신러닝

■ 주차: Lecture 13

■ 교수명: Anindita Mahapatra

Eric Gieseke

■ 목적: Lecture 13의 핵심 개념 학습

Contents

1	기말 과제 (Final Project) 가이드	3
1.1	발표 및 제출 요구사항	3
1.2	팀 구성 및 역할 (R&R)	3
2	데이터 프로젝트의 개발 수명 주기 (SDLC)	4
2.1	환경별 특징 및 전략	4
2.2	서비스 수준 계약 (SLA) 및 지표	4
3	인프라 자동화(IaC)와 CI/CD	5
3.1	IaC: 코드로 인프라 관리하기	5
3.1.1	프로비저닝 vs 구성 관리	5
3.2	CI/CD (지속적 통합/지속적 배포)	5
3.3	데이터 메쉬(Data Mesh)와 워크스페이스	5
4	관측성(Observability)과 데이터 품질 관리	6
4.1	비용 및 사용량 모니터링 (System Tables)	6
4.2	데이터 품질 모니터링	6
5	Databricks Asset Bundles (DAB)	7
5.1	왜 DAB를 쓰는가?	7
5.2	DAB의 핵심 구성 요소	7
5.3	DAB 워크플로우 예시 (CLI)	7

Lecture 13: 데이터 이니셔티브 가치 극대화와 지속적 개선

(CI/CD, IaC, Observability, and Final Project)

개요 (Overview)

이번 강의는 데이터 파이프라인을 구축한 이후, 이를 **어떻게 안정적으로 운영하고 자동화하며 품질을 유지할 것인가**에 초점을 맞춥니다. 단순히 ”돌아가는 코드”를 만드는 것을 넘어, 기업 환경에서 실제 가치를 창출하기 위한 엔지니어링 관행(DevOps)을 데이터 영역(DataOps/MLOps)에 적용하는 방법을 배웁니다.

▣ 핵심 요약

- **기말 과제(Final Project):** 팀 구성 및 역할, 발표 자료(20장 내외) 작성 요령 안내.
- **SDLC (소프트웨어 개발 수명 주기):** 개발(Dev) → 스테이징(Stage) → 운영(Prod) 단계별 전략.
- **인프라 자동화 (IaC):** 테라폼(Terraform) 등을 활용해 인프라를 코드로 관리하는 법.
- **CI/CD 관측성(Observability):** 자동화된 테스트/배포 파이프라인과 데이터 품질/비용 모니터링.
- **Databricks Asset Bundles (DAB):** 프로젝트 자산을 패키징하여 배포하는 최신 표준.

1 기말 과제 (Final Project) 가이드

학기말 프로젝트는 "Lakehouse 아키텍처 구축"을 목표로 하며, 팀 단위로 진행됩니다. 실제 협업의 데이터 프로젝트 제안 및 구축 과정을 모사합니다.

1.1 발표 및 제출 요구사항

- 분량:** 슬라이드 최대 20장.
- 시간:** 발표 15분 + 질의응답(Q&A) 5분.
- 일정:** 마지막 강의(12/15 예정) 시간에 팀별 발표 진행 (타임존 이슈가 있는 팀 우선 배정 가능).
- 참여:** 모든 팀원이 발표에 참여해야 함 (슬라이드 넘기는 역할 포함).

1.2 팀 구성 및 역할 (R&R)

팀당 2명씩 짝을 이루어 아래의 핵심 역할을 분담하고 발표에 포함해야 합니다.

Table 1: 기말 프로젝트 팀원별 역할 정의

역할 (Role)	담당 업무 (Responsibilities)	발표 포인트
데이터 아키텍트	전체 설계 및 아키텍처	확장성, 품질, 성능, 신뢰성, 거버넌스 전략 설명
데이터 엔지니어	데이터 파이프라인 구축	데이터 수집(Ingestion), 변환, 조인, 집계 과정 시연
ML 전문가	모델 개발 및 관리	모델 훈련, 추론(Inference), 모델 수명 주기 관리
데이터 분석가	BI 대시보드 및 인사이트	쿼리 작성, 대시보드 시각화, 알림 설정, 비즈니스 인사이트 도출

[프로젝트 필수 포함 사항] 발표에는 반드시 **"문제 정의 (Problem Statement)"**, **"팀 소개 (역할별)"**, **"아키텍처 리뷰"**, **"인사이트 요약"**, 그리고 **"향후 개선 계획 (Next Steps)"** 이 포함되어야 합니다. 팀 이름(가상의 컨설팅 회사명 등)을 정하는 것도 잊지 마세요.

2 데이터 프로젝트의 개발 수명 주기 (SDLC)

데이터 파이프라인을 만들 때, 로컬에서 대충 코드를 짜고 바로 운영 서버에 올리는 것은 매우 위험합니다. 안정적인 서비스를 위해 3단계 환경을 거칩니다.

[비유: 요리의 3단계]

- **Dev (개발):** 집 부엌에서 혼자 새로운 레시피를 실험해보는 단계. 재료를 조금만 씀.
- **Stage (스테이징):** 친구들을 불러 시식회를 하는 단계. 실제 손님상과 비슷하게 차려놓고 문제 없는지 확인.
- **Prod (운영):** 실제 레스토랑에서 손님에게 돈 받고 파는 단계. 실수가 용납되지 않음.

2.1 환경별 특징 및 전략

Table 2: SDLC 3단계 상세 비교

구분	1. 개발 (Development)	2. 스테이징 (Staging)	3. 운영 (Production)
목표	비즈니스 목적 구현 및 단위 테스트	운영 환경 모사 및 통합 테스트	실제 서비스 제공 및 안정성 유지
데이터	가짜 데이터(Canned data) 또는 소량 샘플	운영 데이터와 유사한 대용량 데이터	실제 실시간/배치 데이터 전체
리소스	단일 노드 클러스터, Spot 인스턴스(저비용)	성능 테스트를 위한 확장된 리소스	고가용성 정책, 안정적인 인스턴스
권한	개발자 개인 계정(User) 사용	서비스 주체(Service Principal) 도입 시작	**오직 서비스 주체(Service Principal)만 사용**
상태	상호작용(Interactive) 중심	자동화(Automated)로 전환	**완전 자동화(Fully Automated)**

2.2 서비스 수준 계약 (SLA) 및 지표

운영 단계에서는 시스템이 얼마나 잘 돌아가는지 약속(SLA)하고 측정해야 합니다.

- **가용성(Availability):** 시스템이 정상 작동하는 시간 비율. (예: 99.999% = 연간 다운타임 5분 미만)
- **재해 복구(Disaster Recovery):**
 - **RTO (Recovery Time Objective):** 사고 후 복구까지 걸리는 시간(얼마나 빨리 고치나?)
 - **RPO (Recovery Point Objective):** 데이터 백업 주기 (과거 데이터를 어디까지 잃어도 되나?)

3 인프라 자동화(IaC)와 CI/CD

수백 개의 워크스페이스와 클러스터를 사람이 일일이 클릭해서 만들면 실수하기 쉽고 관리가 불가능합니다. 이를 코드로 관리하는 것이 **IaC(Infrastructure as Code)**입니다.

3.1 IaC: 코드로 인프라 관리하기

[비유: 건물 설계도] 웹 콘솔에서 클릭으로서 서버를 만드는 건 '손으로 흙을 빚어 집을 짓는 것'과 같습니다. 똑같은 집을 또 지으려면 처음부터 다시 빚어야 합니다. **IaC**는 '3D 프린터 설계도'를 만드는 것입니다. 설계도(코드)만 있으면 버튼 하나로 똑같은 집을 100채든 1,000채든 완벽하게 지을 수 있습니다.

3.1.1 프로비저닝 vs 구성 관리

- **프로비저닝 (Provisioning):** 인프라 자체를 생성 (예: Terraform, CloudFormation).
 - 특징: 불변(Immutable). 변경이 필요하면 기존 것을 부수고 새로 짓는 방식을 선호.
 - *Databricks* 활용: 워크스페이스 생성, 네트워크 설정 등 큰 틀을 잡을 때 주로 사용.
- **구성 관리 (Config Management):** 이미 있는 인프라 내부의 소프트웨어를 관리 (예: Ansible, Puppet).
 - 특징: 가변(Mutable). 기존 서버에 들어가서 라이브러리 버전을 업데이트하는 방식.

3.2 CI/CD (지속적 통합/지속적 배포)

코드 변경 사항이 자동으로 테스트되고 배포되는 파이프라인입니다.

1. **CI (통합):** 개발자가 Git에 코드를 'commit'하면 자동으로 빌드하고 유닛 테스트를 수행.
2. **CD (배포):** 테스트를 통과하면 스테이징/운영 환경으로 자동 배포(Deploy).
3. **이점:** 배포 주기가 빨라지고(분기별 → 매일), 버그를 조기에 발견.

3.3 데이터 메쉬(Data Mesh)와 워크스페이스

조직이 커지면 중앙 팀이 모든 데이터를 관리할 수 없습니다. **데이터 메쉬**는 데이터를 '제품(Product)'으로 취급하고, 도메인(마케팅, 영업 등) 별로 소유권을 분산시킵니다.

- **워크스페이스 분리:** 각 도메인/프로젝트 별로 별도 워크스페이스를 할당하여 격리(Isolation) 확보.
- **Unity Catalog:** 분산된 도메인 데이터를 하나로 묶어주는 중앙 거버넌스 역할.

4 관측성(Observability)과 데이터 품질 관리

시스템이 ”잘 돌아가는지” 확인하는 것을 넘어, ”무엇이 잘못되었고 왜 그런지”를 파악하는 능력입니다.

4.1 비용 및 사용량 모니터링 (System Tables)

Databricks의 ‘system’ 카탈로그를 통해 모든 과금 및 사용 정보를 SQL로 조회할 수 있습니다.

- **Cost Management:** 누가(어떤 태그가) 돈을 얼마나 썼는지 확인 → 예산 초과 방지.
- **Audit Logs:** 누가 언제 어떤 데이터에 접근했는지 감사 추적.
- **Cluster Usage:** 클러스터가 놀고 있는데 켜져 있는지(유휴 상태) 확인.

4.2 데이터 품질 모니터링

데이터 팀은 보통 ”데이터를 제때 주는 것(Delivery SLA)”에 집중하지만, 사용자는 ”데이터가 정확한 것(Quality)”을 기대합니다. 이를 자동화해야 합니다.

Table 3: 데이터 품질 모니터링 도구 비교

기능	설명	특징
Anomaly Detection	AI가 과거 패턴을 학습하여 이상 징후 감지.	설정이 매우 쉽음(버튼 클릭). 신선도(Freshness)와 완전성(Completeness) 체크.
Lakehouse Monitoring	테이블의 통계적 프로파일(분포, Null 비율 등) 및 드리프트(변화) 감지.	대시보드 자동 생성. 데이터 내용의 변화(Drift)를 상세히 추적 가능.
DQX (Data Quality X)	코드로 정의하는 데이터 품질 프레임워크.	파이프라인 코드 내에서 ‘age > 18’ 같은 구체적 규칙(Rule) 적용 가능.

[비유: 건강 검진]

- **Anomaly Detection:** 스마트워치가 ”평소보다 심박수가 높아요”라고 알려주는 것 (자동, 패턴 기반).
- **Lakehouse Monitoring:** 정기 건강검진 결과표(혈압, 콜레스테롤 수치 등)를 받아보는 것 (상세 통계).
- **DQX:** 의사가 ”술은 주 1회 이하로”라고 처방을 내리고 지키는지 체크하는 것 (명시적 규칙).

5 Databricks Asset Bundles (DAB)

DAB는 데이터 프로젝트(코드, 잡 설정, 파이프라인 등)를 **하나의 패키지로 묶어** 개발, 스테이징, 운영 환경으로 손쉽게 배포하는 최신 도구입니다.

5.1 왜 DAB를 쓰는가?

기존에는 API를 일일이 호출하거나 수동으로 옮겨야 했지만, DAB를 쓰면 ‘bundle deploy‘ 명령어 하나로 프로젝트 전체를 다른 환경에 똑같이 복제할 수 있습니다.

5.2 DAB의 핵심 구성 요소

- **YAML 설정 파일 ('databricks.yml')**: 프로젝트의 모든 설정(어떤 잡을 실행할지, 어떤 클러스터를 쓸지)이 정의된 설계도.
- **Target (타겟)**: 배포할 목적지 (Dev, Stage, Prod). 타겟별로 설정을 다르게(Override) 할 수 있음.
- **CLI 명령어**: ‘databricks bundle validate’, ‘databricks bundle deploy’, ‘databricks bundle run’.

5.3 DAB 워크플로우 예시 (CLI)

```

1 # 1. 번들유효성검사      (YAML 문법등확인 )
2 databricks bundle validate
3
4 # 2. 개발(dev) 환경에배포
5 databricks bundle deploy -t development
6
7 # 3. 개발환경에서잡실행테스트      ()
8 databricks bundle run -t development <job_key>
9
10 # 4. 테스트(통과후) 운영(prod) 환경에배포
11 databricks bundle deploy -t production

```

Listing 1: DAB를 활용한 배포 및 실행 절차

부록: 핵심 용어 정리 (Glossary)

용어	원어	шу운 설명
서비스 주체	Service Principal	사람이 아닌 '로봇 ID'. 자동화 작업이나 파이프라인 실행 시 사용하는 계정.
스팟 인스턴스	Spot Instance	클라우드 남는 자원을 싸게 쓰는 것. 회수될 수 있어 중요 작업엔 비추천.
데이터 메쉬	Data Mesh	데이터를 중앙에서 독점하지 않고, 각 부서(도메인)가 직접 관리/제공하는 방식.
드리프트	Drift	데이터의 통계적 속성이 과거와 달라지는 현상 (예: 갑자기 Null이 늘어남).
테라폼	Terraform	인프라 생성 도구. 코드로 서버, 네트워크 등을 정의하고 생성함.
단위 테스트	Unit Test	코드의 가장 작은 단위(함수 등)가 의도대로 작동하는지 검사하는 것.
통합 테스트	Integration Test	여러 모듈을 합쳤을 때 서로 잘 연결되어 작동하는지 검사하는 것.

부록: 학습 및 프로젝트 체크리스트

기말 프로젝트 및 시험 대비 점검

- 프로젝트 준비:** 팀원 컨택 및 역할 분담(아키텍트, 엔지니어, ML, 분석가) 완료했는가?
- 발표 자료:** 문제 정의, 아키텍처, 파이프라인 시연, 인사이트, 향후 계획이 모두 포함되었는가?
- 개념 이해:** SDLC 3단계(Dev/Stage/Prod)의 데이터/리소스/권한 차이를 설명할 수 있는가?
- 자동화:** IaC(Terraform) 와 DAB(Asset Bundle)의 차이(인프라 vs 프로젝트)를 구분할 수 있는가?
- 모니터링:** System Table로 비용을 추적하고, Lakehouse Monitoring으로 데이터 품질을 감시하는 방법을 아는가?
- CI/CD:** 코드가 변경되었을 때 배포까지의 자동화 흐름을 그릴 수 있는가?

■ 강의명: CSCI E-103: 재현 가능한 머신러닝

■ 주차: Lecture 14

■ 교수명: Anindita Mahapatra

Eric Gieseke

■ 목적: Lecture 14의 핵심 개념 학습

■ 강의명: Big Data Analysis & Engineering

■ 주차: Day 14 - Databricks Roadmap & Review

■ 교수명: Prof. & Teaching Staff

■ 목적: Databricks의 최신 기능(Lakebase, AI/BI, Agent) 이해 및 Quiz 2, Assignment 3의 주요 개념(Spark 최적화, Streaming) 심층 리뷰

Contents

1 Databricks Data Intelligence Platform (Roadmap)	2
1.1 주요 구성 요소 요약	2
1.2 Lakebase (Managed Postgres)	2
1.3 AI/BI: Genie & Research Mode	2
1.4 Apps & Model Serving	2
2 Quiz 2 Review: Spark & Delta Lake Concepts	3
2.1 Spark Optimization & Troubleshooting	3
2.2 Delta Lake Internals	3
2.3 Data Frame Operations	3
3 Assignment 3 Review: Streaming & Architecture	4
3.1 Kafka vs. Kinesis	4
3.2 Slowly Changing Dimensions (SCD)	4
3.3 Spark Streaming Logic	4
4 Next Steps & Action Items	4

1 Databricks Data Intelligence Platform (Roadmap)

이번 강의에서는 Databricks 로드맵에 포함된 최신 기능들을 다룹니다. 이 기능들은 주로 **Data Intelligence Capabilities**를 강화하여 플랫폼을 더 스마트하게 만들고 데이터 담당자의 업무를 줄이는 데 초점을 맞춥니다.

1.1 주요 구성 요소 요약

Category	Feature	Description
Data Warehousing	Lakebase (Postgres)	Lakehouse 환경 내에서 OLTP 트랜잭션 처리를 지원하는 관리형 Postgres.
AI & Agents	Agent Bricks	AutoML처럼 데이터셋만 지정하면 에이전트를 자동 생성해주는 프레임워크.
	Genie	정형화된 "What" 질문에 대한 답변을 제공하는 BI 도구.
	Research Mode	가설 기반의 "Why" 질문에 대해 Chain of Thought를 통해 심층 답변 제공.
Governance	ABAC	속성 기반 접근 제어(Attribute-Based Access Control). PII 태그 등에 따라 자동 권한 부여.
Apps	Lakehouse Apps	데이터가 있는 곳에서 바로 실행되는 Python(Streamlit 등) 기반 앱. 보안 및 확장성 보장.

Table 1: *Databricks* 주요 신규 기능 요약

1.2 Lakebase (Managed Postgres)

기존의 Databricks는 OLAP(분석용)에 최적화되어 있었으나, **Lakebase**의 도입으로 OLTP(트랜잭션용) 워크로드까지 커버하게 되었습니다.

- **구조적 특징:** Compute와 Storage가 분리되어 있습니다.
- **장점:**
 - 사용량이 늘어나면 Compute가 확장(Scale-up)되고, 사용하지 않으면 0으로 축소(Scale-to-Zero)되어 비용 효율적입니다.
 - Production과 Development 브랜치를 기본 제공하여, 운영 환경에 영향을 주지 않고 테스트가 가능합니다.
- **통합:** Unity Catalog와 연동되며, Lakehouse와 Lakebase 간 양방향 데이터 동기화(Reverse Sync)가 가능합니다.

1.3 AI/BI: Genie & Research Mode

- **Genie:** 정형 데이터에 대한 사실적 질문(Fact-based questions)을 SQL로 변환하여 답변합니다.
- **Research Mode:** "왜(Why)"에 대한 질문을 처리합니다. 단순 검색이 아니라 가설을 세우고 여러 경로를 탐색(Reasoning)하여 답변을 도출합니다. (예: "관세 인상이 포트폴리오에 미칠 영향은?")

1.4 Apps & Model Serving

- **Apps:** Streamlit, Flask 등으로 만든 앱을 데이터가 저장된 플랫폼 위에서 직접 구동합니다. 데이터 이동 없이 보안이 유지된 상태로 대시보드 이상의 상호작용(Write-back 등)이 가능합니다.
- **Model Serving:** 최신 LLM(Llama, Gemini, Claude 등)을 즉시 사용할 수 있도록 제공하며, MCP(Model Context Protocol) 서버를 통해 에이전트가 외부 도구나 데이터에 접근하도록 지원합니다.

2 Quiz 2 Review: Spark & Delta Lake Concepts

퀴즈에서 많이 혼동했던 Spark와 Delta Lake의 핵심 개념을 정리합니다.

2.1 Spark Optimization & Troubleshooting

1. **Spark 최적화 4대 요소:** Data Skew, Shuffle, Spill, Small Files.
2. **Broadcast Variables:**
 - **Immutable (불변)**: 생성 후 변경할 수 없습니다.
 - **Local to Workers**: 클러스터 간 공유되는 것이 아니라, 각 워커 노드에 복사본이 저장됩니다.
3. **Repartition vs Coalesce:**
 - 파티션을 늘릴 때 (예: 12 → 24): ‘repartition()’ 사용 (Shuffle 발생).
 - 파티션을 줄일 때: ‘coalesce()’ 사용 (Shuffle 최소화).
4. **Action 함수 주의사항:**
 - ‘collect()’: 모든 데이터를 드라이버로 가져오므로 OOM(Out of Memory) 위험이 큼. 디버깅용으로만 사용 권장.
 - ‘take(n)’: 필요한 n 개 행만 가져오므로 대용량 데이터에서 안전함.

2.2 Delta Lake Internals

- **Transaction Log:** ‘*delta_log*’ .
- **구조:** 각 트랜잭션은 **개별 JSON 파일**로 기록됩니다. (하나의 JSON이 아님). 10번째 트랜잭션마다 Checkpoint(Parquet) 파일이 생성됩니다.
- **Time Travel:** 별도의 복사본을 만드는 것이 아니라, 트랜잭션 로그를 통해 과거 시점의 데이터 상태를 조회합니다.

2.3 Data Frame Operations

- **Explode:** DataFrame의 메서드가 아니라 ‘select’ 문 내에서 사용해야 합니다. (예: ‘df.select(explode(col))’)
- **Drop Duplicates:** ‘df.dropDuplicates(['col1', 'col2'])’ 형태로 리스트를 전달해야 합니다.

3 Assignment 3 Review: Streaming & Architecture

3.1 Kafka vs. Kinesis

Feature	AWS Kinesis	Apache Kafka
관리 주체	AWS 완전 관리형 (Managed)	Confluent (상용) 또는 Open Source (직접 관리)
확장 단위	Shard (샤드 수에 따라 비용/성능 결정)	Partition (파티션 단위로 병렬 처리)
데이터 분배	Partition Key 사용	Round-robin 또는 Key-Value 기반 Partitioning

Table 2: *Kinesis*와 *Kafka* 비교

3.2 Slowly Changing Dimensions (SCD)

- **Type 1:** 과거 데이터를 덮어씀(Overwrite). 이력 관리 안 됨.
- **Type 2:** 새로운 행을 추가하고 유효 기간(Start/End Date)이나 플래그(Current)를 두어 이력을 관리함.
- **Delta Merge:** ‘MERGE INTO’ 구문을 사용하여 변경된 컬럼(예: City)만 감지하고 업데이트/삽입로직을 구현합니다.

3.3 Spark Streaming Logic

- **Trigger 옵션:**
 - ‘trigger(once=True)’: **Deprecated**.
 - ‘trigger(availableNow=True)’: 권장됨. 데이터를 마이크로 배치로 나누어 처리하므로 클러스터 과부하를 방지합니다.
- **Checkpointing:** 스트리밍의 핵심. 시스템 중단 후 재시작 시 중단된 지점부터 정확히 다시 처리(Exactly-once) 할 수 있게 해줍니다.
- **Image Processing:**
 - Spark Serverless 환경에서는 ‘display()’로 이미지를 직접 렌더링하지 못할 수 있습니다.
 - ‘pillow’ 라이브러리 등을 사용하여 바이너리 데이터를 이미지로 변환하거나 처리(Invert 등)하는 UDF를 작성해야 합니다.

4 Next Steps & Action Items

- **Assignment 4:** 다음 수업 전까지 최종 과제(Assignment 4)를 확인하고 준비할 것.
- **Guest Lecture:** 다음 수업에는 업계 전문가 초청 강연이 예정되어 있음.