CSCI E-103 2주차: 데이터 모델링과 실습 *

Gemini (학습 노트 정리)

October 26, 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 모델링**을 다룹니다. 데이터를 비즈니스 요구에 맞게 구성하는 3단계(개념적, 논리적, 물리적)를 학습합니다. **OLTP** (운영)와 **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 모델**을 훈련시키는 전 과정을 살펴봅니다.

*본 문서는 Harvard Extension CSCI E-103 강의 자료(강사: Anindita Mahapatra, Eric Gieseke)를 기반으로, 초심자의 이해를 돕기 위해 내용을 재구성하고 보충 설명, 예시, 비유를 추가한 학습 노트입니다.

Contents

Part I

개요 및 복습

1 지난 강의 핵심 복습

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

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 상태인지**를 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 과정**입니다.

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

데이터 모델링은 왜 중요할까요?

- **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 기술**에 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$_ADDRESS_JOIN$'

□ **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):**

```sql
CREATE TABLE T_CUSTOMER (
  CUST_ID SERIAL PRIMARY KEY,
  CUST_NAME VARCHAR(100) NOT NULL,
  EMAIL VARCHAR(255) UNIQUE
);
CREATE TABLE T_BOOK (
  BOOK_ID SERIAL PRIMARY KEY,
  TITLE VARCHAR(500) NOT NULL,
  AUTHOR VARCHAR(200)
);
-- 고객과책의다대다관계를위한조인테이블
CREATE TABLE T_ORDER_ITEMS (
  ORDER_ID INT NOT NULL, -- (T_ORDER 테이블을참조 )
  BOOK_ID INT REFERENCES T_BOOK(BOOK_ID),
  QUANTITY INT DEFAULT 1,
  PRIMARY KEY (ORDER_ID, BOOK_ID)
);
```

Listing 1: 물리적 모델 예시 (SQL)

Part III

데이터베이스 유형별 모델링

4 OLTP vs OLAP: 은행원과 분석가

데이터베이스 시스템은 크게 두 가지 목적, 즉 **운영**과 **분석**으로 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 분석가가 거대한 역사적 데이터를 대상으로 복잡하고 긴 쿼리(대부분 읽기)를 수행합니다.
    - **쿼리 속도**가 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).

5 분석형 모델링 (Dimensional Modeling)

OLAP 시스템(DWH)에서는 분석 속도를 높이기 위해 **차원 모델링**을 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.

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)

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

```
{
  "post_id": "p123",
  "title": "My First Post",
  "content": "Hello world!",
  "comments": [
    { "user": "alice", "text": "Great post!" },
    { "user": "bob", "text": "Welcome." }
  ]
}
```

Listing 2: 임베디드 모델 예시

장점: 게시글과 모든 댓글을 **한 번의 쿼리**로 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.

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

Listing 3: 참조 모델 예시 (Posts)

```
(*@\textit{// Comments Collection}@*)
{ "comment_id": "c1", "post_id_ref": "p123", "user": "alice", "text":
    "Great post!" },
{ "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 DWH**를 **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 (킴볼) 모델: 바텀업 (Bottom-up)**
  - **비유:** "레고 블록(부서)부터 조립하기"
  - **접근법:** 비즈니스 요구사항에 맞춰 각 부서별 **데이터 마트**를 **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 위 레고 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.
    - **타입 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 타입, 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 쓰는 데 유리합니다.
- **열(Column) 기반 저장 (예: Parquet, ORC):**
  - **비유:** 책의 '목차'만 쭉 모으고, '본문 1장'만 쭉 모으고, '색인'만 쭉 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 타입의 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 성능**에 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 **신뢰성**을 더한 오픈소스 스토리지 프로토콜입니다.

기본적으로 데이터는 **Parquet** 포맷으로 저장하지만, 여기에 **트랜잭션 로그(Transaction Log)** 디렉토리(‘$_{d}elta_{l}og/$‘) .

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 띄지 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 여부**는 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 워커**에게 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, **Parquet**처럼 파일 내부적으로 분할(블록) 처리가 되는 포맷을 사용해야 합니다.

11 메타데이터 (Metadata)

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

비유: 책의 내용물(소설)이 **데이터**라면, 책의 표지, 목차, 저자, 출판일, ISBN 코드가 **메타데이터**입니다.

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, 검색할 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 섞여있지 않은가?)
- **내용 발견 (Content discovery):** 데이터 품질을 세부적으로 봅니다. (예: Null 값이 몇 %인가? 유효한 범위(Range)를 벗어난 값은 없는가? - '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 예측**하는 **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) 환경**에서는 **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 **분산**되어 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 피해야 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' 셀 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: 'sqft$_living$'( )
  - 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 데이터**를 사용하여 '.fit()' 메서드를 호출해 모델을 훈련시킵니다.

14.6 6. 모델 평가 및 예측

- **모델 평가 (Evaluate):** 훈련된 모델을 **테스트용 데이터**에 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 데이터**를 입력하여 **예측 가격**을 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 의미**로 사용됩니다. 'CREATE DATABASE my$_d$b' '$CREATESCHEMAmy_db$' .
- **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 저장**하면 됩니다. (혼동을 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 생성**합니다. (예: 'assignment$_0$1$_v$ol')3. , ( ) .(: '$people_data$', '$names_data$')4.'$Uploadtothisvolume$' '$/Volumes/main/lab01/assignment_01_vol/people_data/$') .6.'

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)))

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