머신러닝 제조분야 적용 연구 중간 발표 대본

오프닝

안녕하십니까, 5조 삼성팀의 정유현입니다.

오늘 '머신러닝 제조분야 적용 연구 - 데이터 분석 및 시각화'에 대해 발표하겠습니다.

본 연구는 (주)초록에이아이와 함께 진행하였습니다.

1.발표 목차(슬라이드2)

먼저 발표의 전체 목차를 말씀드리겠습니다.

첫 번째로 팀원을 간략히 소개한 뒤, 프로젝트의 분석 및 설계 내용을 설명드리겠습니다.

이후 시연 영상과 진행 상황을 공유하고, 프로젝트 진행 중 발생한 이슈와 그 해결 방안을 말씀드리겠습니다.

마지막으로 향후 일정 계획을 안내드리며 발표를 마무리하겠습니다.

2. 팀원 소개(슬라이드3)

팀원 소개는 계획발표 때 자세히 했으니 간략하게 하고 넘어가겠습니다.

저희 팀원은 저와 곽민서, 이미진님 총 3명으로 구성되어있습니다.

3. 프로젝트 개요 (슬라이드 4)

본격적으로 발표에 들어가기 앞서, 프로젝트 분석 및 설계 내용에 대해 말씀드리겠습니다.

본 프로젝트의 핵심은 소행 배경 및 목표 설정입니다.

실제 데이터 기반 분석

저희희는 KAMP 등록 데이터를 활용하여 실제 제조 현장의 데이터를 분석했습니다. 모든 데이터는 정리 과정을 거쳐 불량 데이터를 수집하고, 불량 원인을 추론하여 불량률 축소를 목적으로 했습니다.

왼쪽에 보시는 것처럼 실제 데이터 분석 사례를 보면, 2021년 9월 8일부터 27일까지의 날짜별 데이터가 수집되었고, 온도, pH, 전류 등 다양한 센서 값들이 기록되어 있습니다.

오른쪽 그래프는 이상치 점수 분포(Anomaly Score Distribution)를 보여주는데, 대부분의 데이터가 정상 범위에 분포하지만 일부 이상치가 감지되는 것을 확인할 수 있습니다.

---

4. USE CASE 다이어그램 (슬라이드 5)

다음은 시스템의 전체적인 흐름을 보여주는 USE CASE 다이어그램입니다.

크게 두 명의 액터가 있습니다.

- 데이터 분석가는 왼쪽에 위치하며
- 품질 관리자는 오른쪽에 위치합니다

주요 프로세스의 경우,

1. 데이터 분석가는 데이터 수집 및 관리를 시작으로
2. 데이터 전처리 및 **EDA**(탐색적 데이터 분석)를 수행합니다
3. 이후 모델 학습 및 튜닝, 모델 평가 및 검증 단계를 거칩니다
4. 마지막으로 시각화 및 리포팅을 통해 결과를 도출합니다

품질 관리자는 이렇게 도출된 품질 예측 수행 결과를 받아 불량 환경 분석 및 관리를 수행하게 됩니다.

---

5. CLASS 다이어그램 - 전체 구조 (슬라이드 6)

이제 시스템의 클래스 구조를 살펴보겠습니다.

화면에 보시는 것처럼 복잡해 보이지만, 크게 세 가지 영역으로 구분됩니다.

빨간색 박스 (왼쪽)

데이터 처리 및 전처리 관련 클래스들이 모여있습니다.

파란색 박스 (하단)

머신러닝 모델 학습 및 평가 관련 클래스들입니다.

상단 빨간색 박스

실제 데이터를 담는 데이터 클래스들이 위치합니다.

각 클래스들은 상속 관계와 연관 관계로 연결되어 있으며, 이를 통해 체계적인 데이터 처리와 모델링이 가능합니다.

---

6. CLASS 다이어그램 - 데이터 처리 (슬라이드 7)

좀 더 자세히 데이터 처리 부분을 보겠습니다.

먼저, **DataLoader** 클래스입니다.
이 클래스는 데이터를 불러오는 역할을 담당합니다.
루트 디렉터리와 파일 목록을 관리하며, `read_csv()`와 `filter_csv_files()` 같은 메서드를 통해 여러 CSV 파일을 한 번에 읽고 필요한 데이터만 추출할 수 있도록 설계했습니다.

다음은 **DataPreprocessor** 클래스입니다.
불러온 데이터를 본격적으로 분석하기 전에 전처리하는 단계입니다.
결측치를 제거하거나(`remove_nulls()`), 불필요한 컬럼을 제거(`drop_columns()`)하고,
데이터 인덱스를 지정하거나 품질을 점검(`check_data_quality()`)하는 기능을 포함하고 있습니다.

DataQualityChecker 클래스는 데이터 품질을 다섯 가지 지표로 평가합니다.
완전성, 고유성, 유효성, 일관성, 정확성, 그리고 무결성입니다.
각각의 점수를 계산하는 메서드(`calculate_completeness()`, `calculate_accuracy()` 등)가 포함되어 있고, 최종적으로 `get_quality_report()`를 통해 데이터 품질 보고서를 제공합니다.

그 다음은 **DataExplorer** 클래스입니다.
이 부분은 탐색적 데이터 분석, 즉 EDA를 수행하는 단계입니다.
데이터의 형태와 컬럼 정보를 확인하거나(`get_info()`),
결측치 개수를 확인하고(`count_nulls()`), 히스토그램, 상관관계, 히트맵을 시각화(`plot_histogram()`, `plot_heatmap()`)하여 데이터의 전반적인 특성을 한눈에 파악할 수 있습니다.

마지막으로 **RegressionModel** 클래스입니다.
이 클래스는 AutoML 기반의 `StructuredDataRegressor`를 활용해 회귀 모델을 학습하고 예측하는 역할을 합니다.
`fit()` 메서드로 모델을 학습시키고, `predict()`로 예측을 수행하며, 학습된 모델은 `export_model()`로 저장할 수 있습니다.
또한 `summary()`를 통해 모델의 구조와 성능 요약을 확인할 수 있습니다.

전체적으로 보면,
데이터 로딩 → 전처리 → 품질 검증 → 탐색 분석 → 모델 학습
이라는 하향식 데이터 파이프라인 구조를 기반으로 설계되어 있습니다.
이런 모듈화 구조를 통해 각 단계별 유지보수와 기능 확장이 용이하며,
데이터 품질에서부터 예측 모델까지 일관된 흐름으로 관리할 수 있습니다.

---

7. CLASS 다이어그램 - 모델링 (슬라이드 8)

마지막으로 프로젝트의 모델 학습 및 평가 단계의 클래스 다이어그램입니다.
이 구조는 데이터 전처리 이후, 실제로 모델을 학습하고 성능을 평가하며 시각화하는 과정을 체계적으로 관리하기 위한 설계입니다.

가장 중심에 있는 클래스는 **ModelTrainer**입니다.
이 클래스는 학습의 전체 과정을 총괄하는 핵심 클래스입니다.
`train_data`와 `test_data`를 관리하며, 모델 생성(`create_model()`), 데이터 분할(`prepare_train_test_split()`), 학습(`fit()`), 저장(`export_model()`), 불러오기(`load_model()`) 기능을 수행합니다.
즉, 모델 학습의 시작부터 종료까지의 전체 흐름을 담당한다고 볼 수 있습니다.

다음으로 **RegressionModel** 클래스입니다.
이 클래스는 AutoML 기반의 `StructuredDataRegressor`를 활용해
회귀 모델을 생성하고 학습(`fit()`), 예측(`predict()`), 모델 요약(`summary()`) 기능을 제공합니다.
`ModelTrainer`가 학습 전체를 제어한다면, `RegressionModel`은 그 중에서도 실제 학습 로직을 수행하는 엔진 역할을 합니다.

학습된 모델의 성능은 **ModelEvaluator** 클래스가 담당합니다.
여기서는 예측값(`y_pred`)과 실제값(`y_test`)을 비교하여
MSE, 정확도, 정밀도, 재현율, F1-score 등을 계산하고(`calculate_mse()`, `calculate_accuracy()` 등), 혼동 행렬(`plot_confusion_matrix()`)과 ROC 곡선(`plot_roc_curve()`) 시각화를 제공합니다.
또한 `classification_report()`를 통해 종합적인 모델 성능 리포트를 생성합니다.
즉, 모델이 얼마나 잘 작동하는지를 정량적으로 평가하는 단계입니다.

다음은 **SensorDataset** 클래스입니다.
이 클래스는 실제 제조 공정 데이터를 관리합니다.
각 데이터는 pH, 온도(Temp), 전류(Current) 등의 공정 변수로 구성되어 있으며,
`validate_ph_range()`나 `validate_temp_range()` 등의 메서드를 통해
센서 값이 정상 범위에 있는지를 검증합니다.

즉, 모델 입력 데이터의 신뢰성을 보장하는 역할을 합니다.

ProcessDataset 클래스는 공정 단위 데이터를 관리합니다.
LOT 단위로 데이터를 추출하거나(`extract_lot_lists()`), 날짜별로
정리(`extract_date_lists()`)하여 분석이나 모델 학습 시 특정 기간 또는 공정 단위로
데이터를 활용할 수 있게 합니다.
`load_error_list()` 기능을 통해 불량이나 이상 데이터도 함께 관리할 수 있습니다.

마지막으로 **DataVisualizer** 클래스입니다.
이 클래스는 학습된 결과를 시각화하는 역할을 담당합니다.
3D 산점도(`plot_3d_scatter()`), 히스토그램(`plot_histogram()`), 상관관계
히트맵(`plot_correlation_heatmap()`), 그리고 모델 구조
시각화(`plot_model_structure()`) 등을 통해 모델의 학습 결과와 데이터 패턴을
직관적으로 확인할 수 있도록 합니다.
즉, 분석 결과를 한눈에 볼 수 있는 시각적 인터페이스입니다.

요약하자면, 이 다이어그램은 데이터셋 검증 → 모델 학습 → 성능 평가 → 결과 시각화로
이어지는 **AI** 모델 학습 파이프라인의 후반부 구조를 보여줍니다.
각 단계가 독립적인 클래스로 분리되어 있기 때문에 유지보수와 확장이 용이하고, 실제
산업 데이터에도 유연하게 적용할 수 있습니다.

---

8. 시연동영상**(**슬라이드 **9)**

이제부터는 실제 프로젝트의 실행 과정을 보여드리겠습니다.
전체 실행 순서는 총 **13**단계로 구성되어 있습니다.

0단계**,** 런타임 초기화입니다**.**
먼저 런타임 환경을 초기화해 이전 실행 기록이나 캐시를 모두 비워줍니다.
이 과정을 통해 패키지 충돌이나 데이터 잔여값 없이 깨끗한 환경에서 시작할 수 있습니다.

1단계**,** 필수 패키지 설치입니다**.**
프로젝트 실행에 필요한 라이브러리를 자동으로 설치합니다.
여기에는 pandas, numpy, matplotlib 같은 데이터 분석용 패키지와
autokeras, scikit-learn 등의 머신러닝 패키지가 포함됩니다.

원래는 AutoML 기능을 활용하기 위해 `tensorflow`와 `autokeras`를 함께 설치하려고 했지만, 환경에서 **TensorFlow 2.12** 이상 버전과 **Python 3.10** 조합에서 호환성 오류가 발생했습니다. (`ModuleNotFoundError: keras.saving.hdf5_format` 등의 오류)

따라서 두 패키지는 설치 과정에서 **try-except** 블록으로 예외 처리했습니다

이렇게 해서, AutoKeras 관련 코드는 건너뛰고 대신 **scikit-learn** 기반의 수동 모델링 로직으로 전환했습니다. 이 방식으로 환경 충돌을 피하면서도 동일한 분석 파이프라인을 유지할 수 있었습니다.

2단계**,** 라이브러리 임포트 및 환경 설정입니다**.**
설치된 패키지를 불러오고, 시각화 스타일이나 워닝 메시지 제거 등
분석 환경을 깔끔하게 세팅합니다.
또한 matplotlib과 seaborn을 사용해 결과를 보기 좋게 출력하도록 설정합니다

3단계**,** 데이터 불러오기입니다**.**
이 단계에서는 실제 공정 데이터를 로드합니다.
프로젝트에서는 KAMP 제조 AI 데이터셋을 사용했으며,
(pH, 온도, 전류 등 센서 데이터와 품질 관련 Lot 데이터가 각각 CSV 형식으로 저장되어 있습니다.

4단계**,** 파일 스캔 및 리스트업입니다**.**
DataLoader 클래스를 활용해 폴더 내 CSV 파일들을 자동으로 탐색하고,

올바른 데이터 파일 리스트를 생성합니다.
이 과정을 통해 잘못된 파일이나 누락된 파일을 사전에 걸러냅니다.

5단계**,** 안전한 **CSV** 로더와 날짜·시간 파서입니다**.**
CSV 파일을 pandas로 불러오되, 인코딩 에러를 자동으로 처리하고 날짜(date)와 시간(time) 컬럼을 결합해 `datetime` 객체로 변환합니다. 이를 통해 시계열 정렬 및 시점 단위 분석이 가능해집니다.

6단계**,** 센서 **CSV** 읽기 및 **Timestamp** 생성입니다**.**
이제 센서 데이터를 읽고, 'date'와 'time' 컬럼을 합쳐 하나의 `Timestamp` 컬럼을 생성합니다.
이를 통해 공정 데이터가 시간 순서대로 정렬되어 이상치 탐지나 품질 분석에 활용할 수 있게 됩니다.

7단계**, EDA(**탐색적 데이터 분석**)**입니다**.**
이제 7단계, **EDA**, 즉 탐색적 데이터 분석 결과를 설명드리겠습니다.
이 단계에서는 센서별 데이터 분포와 변수 간 상관관계를 시각적으로 확인했습니다.

먼저 위쪽의 히스토그램 세 개를 보시면,
왼쪽부터 **pH**, 온도**(Temp)**, 전류(Current)의 분포를 나타냅니다.

먼저 **pH**입니다**.**

대부분의 값이 9.8에서 10.8 사이에 고르게 분포해 있습니다.

11 이상으로 넘어가는 구간에서는 데이터가 급격히 줄어드는데요,

이 구간은 센서 보정 시점이거나, 비정상 반응이 발생했을 가능성이 있습니다.

즉, pH는 전반적으로 안정적이지만, 소수의 이상값이 존재합니다.

다음은 온도**(Temp)**입니다**.**

온도는 43도에서 47도 사이에 주로 분포해 있고,

50도 이상에서는 거의 값이 없습니다.

이건 과열 구간이 일시적으로 발생했거나 센서 노이즈일 가능성이 있습니다.

그래서 전반적으로는 온도 제어가 잘 유지되고 있다고 볼 수 있습니다.

마지막으로 전류**(Current)**입니다**.**

전류는 7에서 8.5암페어 구간에 가장 많이 분포합니다.

6암페어 이하나 9암페어 이상에서는 거의 측정값이 없는데요,

이건 전류 제어가 비교적 안정적이었다는 의미입니다.

이제 아래쪽의 히트맵을 보겠습니다.

pH, 온도, 전류 간의 상관관계**(Correlation)** 를 계산한 결과,

모든 조합에서 상관계수가 0.02 수준으로 거의 0에 가깝습니다.

즉, 세 변수는 서로 독립적으로 움직이고 있다는 것을 의미합니다.

온도가 올라간다고 해서 pH가 바뀌거나,

전류가 변한다고 해서 온도가 따라 움직이지는 않는다는 거죠.

요약하자면,
전체적으로 센서 데이터는 안정적으로 수집되고 있고**,**
세 변수 간 상관은 약하지만,
각 변수 내부에서 일부 구간에 이상값이 존재한다는 점을 확인했습니다.

특히 pH 11 이상, 온도 50도 이상, 전류 9암페어 이상에서는
센서 이상이나 공정 불안정이 발생했을 가능성이 있어,
이 구간의 데이터를 이후 단계에서 불량 **Lot(QC=1)** 과 비교해
실제 품질 이상과 연관이 있는지 검증할 예정입니다.

8단계**,** 에러 **Lot**을 불러와 품질 라벨**(QC)**을 생성합니다**.**
불량 데이터 목록을 로드하고, 각 Lot에 품질 라벨(QC)을 부여합니다.
이때 0은 정상, 1은 불량으로 구분되어, 지도학습 시 목표변수(y)로 사용됩니다.

9단계**,** 학습 및 평가 데이터셋 구성입니다**.**
데이터를 학습(train)과 평가(test) 세트로 분리합니다. QC 라벨이 있는 경우 분류 모델 학습에 사용하고, 라벨이 없는 경우에는 비지도 기반의 이상치 탐지 모델에 활용됩니다.
이 과정에서 `train_test_split()`을 통해 8:2 비율로 데이터를 나누고, 입력 변수(X)와 목표 변수(y)를 정의합니다.

10단계**,** 분류 모델 학습과 비지도 이상치 탐지 전환입니다**.**
이제 **10**단계**,** 분류 모델 학습과 비지도 이상치 탐지 전환 단계입니다.
이 부분이 이번 프로젝트의 핵심입니다.

먼저, 저희는 **AutoKeras** 대신 **scikit-learn** 기반의 모델을 직접 구현했습니다.
데이터에 품질 라벨(QC)이 존재하는 경우에는
RandomForestClassifier나 **LogisticRegression**을 사용해
품질 이상 여부를 예측하는 지도학습**(supervised learning)** 을 수행했습니다.

하지만 QC 라벨이 없는 구간이 많았기 때문에,
이 경우에는 자동으로 비지도학습**(unsupervised learning)** 으로 전환되었습니다.
비지도 모델로는 **IsolationForest**와 **One-Class SVM**을 사용하여
센서 데이터 패턴을 기반으로 이상치를 탐지했습니다.

즉, 데이터 라벨 유무에 따라 모델이 자동으로 전환되는 구조입니다.
라벨이 있으면 분류(Classification),
라벨이 없으면 이상 탐지(Anomaly Detection)로 바뀌도록 설계했습니다.

화면에 보이는 그래프는
비지도학습 모델의 결과를 시각화한 **"Anomaly Score Distribution"** 입니다.
가로축은 이상 점수, 세로축은 해당 점수 구간의 데이터 개수입니다.

보시면 대부분의 데이터가 0.45에서 0.55 사이에 몰려 있고,
0.6 이상 구간부터는 데이터가 급격히 줄어듭니다.
이 구간이 바로 비정상 또는 잠재적 불량 가능성이 높은 센서 패턴입니다.

즉, 모델이 전체 데이터 중에서
이상 징후가 있는 구간을 자동으로 분리해낸 것을 확인할 수 있습니다.

정리하자면,

이 단계에서는 데이터의 라벨 존재 여부에 따라 지도·비지도 모델을 자동 선택하고,

그 결과로 이상 점수 분포를 통해 공정 내 이상 구간을 시각적으로 확인했습니다.

11단계**,** 예측 및 평가 지표 계산입니다**.**
학습된 모델로 예측을 수행하고,
정확도, 정밀도, 재현율, F1-score 같은 핵심 평가 지표를 계산합니다.
비지도 모델의 경우에는 이상치 탐지율을 기반으로 성능을 평가했습니다.

12단계**, ROC Curve** 시각화입니다**.**
다음은 **12**단계**, ROC Curve** 시각화입니다.
이 단계는 **QC** 라벨이 존재하는 지도학습**(supervised learning)** 의 경우에만 수행됩니다.
ROC Curve는 모델의 분류 성능을 시각적으로 표현해,
정상과 이상을 얼마나 잘 구분하는지를 한눈에 보여주는 그래프입니다.

다만 저희 프로젝트의 경우,
QC 라벨이 불완전하거나 누락된 구간이 많아서
비지도학습 기반의 이상치 탐지**(Unsupervised Anomaly Detection)** 방식으로
진행했습니다.
따라서 ROC Curve 시각화 단계는 이번 분석에서는 생략했습니다**.**

13단계**,** 결과 및 아티팩트 저장입니다**.**
마지막으로 모델, 시각화 이미지, 로그, 분석 결과 파일 등을
자동으로 `/results/` 폴더에 저장합니다.
이렇게 생성된 아티팩트는 이후 재학습이나 리포트 생성 시 재활용할 수 있습니다.

이상으로, 데이터 로딩부터 모델 평가까지 이어지는
전체 실행 파이프라인 시연 과정을 설명드렸습니다.
이 구조를 통해 데이터 분석과 모델링을 완전 자동화할 수 있었으며,
실제 산업 데이터에서도 안정적으로 동작하는 시스템을 구현할 수 있었습니다.

---

8. 진행상황 **(**슬라이드 **10)**

다음은 진행 현황에 대해 말씀드리겠습니다.

현재 저희 **5**팀은 데이터 분석 단계를 완료했지만,

논문 작성과 웹 시각화**(**바이브 코딩**)** 부분은 아직 진행 중입니다.

따라서 전체 진행률을 약 **40%** 수준으로 산정했습니다.

9. 진행상황 **(**슬라이드 **11)**

우선 Docs 파트입니다.

10. 진행상황 **(**슬라이드 **12, 13)**

지금까지 저희는 **9**월 **8**일**, 15**일**, 17**일**, 10**월 **1**일**, 10**월 **10**일

총 **5**회의 팀 회의를 진행했습니다.

11. 진행상황 **(**슬라이드 **14)**

또한 **9**월 **18**일과 **9**월 **29**일, 총 **2**회의 멘토 미팅을 진행했습니다.
멘토님께 피드백을 받고, 데이터 처리 방식과 프로젝트 구조를 개선했습니다.

12. 진행상황 **(**슬라이드 **15, 16,17)**

회의 이후에는 이렇게 보이는 것처럼
회의록과 관련 문서**,** 코드 파일을 모두 **GitHub**와 **Notion**에 공유했습니다.
이를 통해 팀원 간의 협업 효율을 높이고, 자료를 체계적으로 관리하고 있습니다.

13. 진행상황 **(**슬라이드 **18, 19, 20, 21, 22, 23, 24)**

마지막으로 **Data Science** 파트입니다.
해당 부분의 코드는 앞서 시연 영상 파트에서 이미 설명드렸기 때문에,
여기서는 작성한 **README** 문서만 간략히 공유드리겠습니다.
README에는 데이터 분석 과정, 모델 구조, 결과 시각화 요약이 포함되어 있습니다.

—

먼저, 프로젝트 개요입니다.
저희는 **KEMP ABH Sensor** 데이터를 이용해 품질 이상을 탐지하고 진단하는 실습을 진행했습니다.
센서 로그와 Error Lot 리스트를 함께 활용해,
QC 라벨이 존재하면 지도학습**(XGBoost** 분류**)**,
라벨이 없으면 비지도학습**(IsolationForest)** 으로 자동 전환되도록 설계했습니다.
모든 실습은 **Google Colab** 환경에서 바로 실행 가능하게 구성했습니다.

—

다음은 실행 환경 및 패키지 구성입니다.

Colab 환경에서 pandas, numpy, scikit-learn, matplotlib, seaborn, xgboost를 설치해 사용했습니다.

AutoKeras는 TensorFlow 버전 충돌로 제외하고, 대신 XGBoost를 메인 모델로 설정했습니다.

—

파이프라인은 총 **7**단계로 구성되어 있습니다**.**
1️⃣CSV 로딩과 정규화,
2️⃣Timestamp 생성,
3️⃣피처 표준화,
4️⃣라벨 생성(QC),
5️⃣모델 학습(자동 전환),
6️⃣EDA,
7️⃣결과 저장으로 이루어집니다.

—

모델구성의 경우, 라벨이 정상/불량 모두 존재하면 XGBClassifier를,
한 클래스만 존재하면 IsolationForest를 선택해
데이터 상태에 맞는 경로로 자동 학습이 진행됩니다.

—

그래프 해석 부분에서는 히스토그램으로 변수별 분포를, 히트맵으로 변수 간 상관관계를 확인했습니다. 또한 비지도 모델의 결과로 생성된 **Anomaly Score** 분포 그래프를 통해 정상 구간과 이상 구간을 직관적으로 구분할 수 있었습니다.

—

마지막으로, 산출물**(Artifacts)** 입니다.
모델 학습 결과와 그래프, 예측 CSV 파일이
`/artifacts` 폴더에 자동으로 저장됩니다.
지도학습일 경우에는 Confusion Matrix, ROC Curve, Feature Importance를 포함하고,
비지도학습일 경우에는 이상치 분포와 상위 이상치 리스트를 저장합니다.

14. 이슈상황 및 해결방안 **(**슬라이드 **25)**

다음으로 이슈사항과 해결방안에 대해 설명드리겠습니다.

이슈는 크게 총 3가지가 있었습니다.

15. 이슈상황 및 해결방안 **(**슬라이드 **26, 27)**

저희 팀이 겪었던 첫 번째 이슈는 시간 부족으로 인한 환경 제약이었습니다.
초기에는 익숙한 **Google Colab** 환경에서 먼저 분석을 진행했습니다.

Colab은 클라우드 기반이기 때문에 별도의 설치 과정이 필요 없고, GPU 지원 덕분에 빠르게 모델을 실험할 수 있었습니다. 무엇보다 팀원 모두가 익숙한 환경이라 협업 효율이 높았습니다.

이후에 배부해주신 GPU자원을 활용하기 위해 Colab에서 정리된 코드를 바탕으로 Jupyter 환경에서도 동일한 분석을 재현했지만, 모델 평가 및 일부 시각화 파트가 아직 미완성이어서 중간 발표에서는 Colab 버전을 중심으로 결과를 공유했습니다.

16. 이슈상황 및 해결방안 **(**슬라이드 **28, 29)**

두 번째 이슈는 **AutoKeras**의 호환성 문제였습니다.

저희가 사용한 Google Colab 환경에는 **TensorFlow 2.19** 버전이 기본 탑재되어 있었는데,
이 버전에서는 AutoKeras가 사용하는 내부 모듈 구조가 달라져
설치 후에도 실행 시 오류가 발생했습니다.
쉽게 말하면, AutoKeras가 Colab의 최신 TensorFlow 환경과 호환되지 않았던 겁니다.

처음에는 `pip install autokeras` 명령으로 설치를 시도했지만,
`keras.saving.hdf5_format` 관련 에러가 지속적으로 발생했습니다.
그래서 코드 상단에서 예외 처리**(try-except)** 로 감싸서,
AutoKeras가 정상적으로 import되지 않을 경우에는
자동으로 **scikit-learn** 또는 **XGBoost** 기반 모델로 전환되도록 설정했습니다.

실제로 여기 보시는 코드에서는
AutoKeras 설치를 시도한 뒤, 실패하면 메시지를 출력하고
`AUTOKERAS_AVAILABLE = False`로 설정하여
대체 경로로 넘어가도록 구현했습니다.

결과적으로,
Colab에서는 AutoKeras를 사용할 수 없었지만,
동일한 기능을 **scikit-learn**과 **XGBoost**로 완전히 대체했습니다.
따라서 모델 학습과 예측 과정에는 전혀 차질이 없었고,
자동 모델 선택 구조 역시 그대로 유지할 수 있었습니다.

17. 이슈상황 및 해결방안 **(**슬라이드 **30, 31)**

세 번째 이슈는 **QC** 라벨 불균형 문제였습니다.

저희가 받은 센서 CSV 데이터에는 품질 라벨**(QC)** 이
아예 없거나, 한쪽 클래스만 존재하는 경우가 있었습니다.
예를 들어, 전부 정상(1)로만 표시되어 있거나,
불량(0) 데이터가 전혀 없는 형태였죠.

이런 경우에는 정답 데이터가 없기 때문에
지도학습 모델의 정확도나 **F1-score** 같은 성능 지표를 계산할 수 없습니다**.**
즉, 분류 모델을 학습시키더라도 평가 자체가 불가능했습니다.

그래서 저희는 접근 방식을 바꿔,
비지도학습 기반의 이상치 탐지 모델**(IsolationForest)** 만을 수행했습니다.
이 모델은 라벨 없이도 데이터의 패턴을 학습하고,
다른 데이터와 동떨어진 이상치를 자동으로 찾아냅니다.

이 방식을 통해 라벨이 불완전한 상황에서도 이상 구간을 식별할 수 있었고,
이상 점수 분포(Anomaly Score Distribution)를 활용해
센서 이상 패턴을 시각적으로 확인할 수 있었습니다.

18. 향후일정 **(**슬라이드 **32)**

마지막으로 향후 일정에 대해 말씀드리겠습니다.

앞으로는 주피터 환경에서 데이터 분석을 완성하고,
바이브 코딩**(Vibe Coding)** 을 활용해 웹 시각화 페이지를 구현할 예정입니다.
이후 분석 결과를 정리하여 최종 논문을 작성하고 발표까지 진행하는 것을 목표로 하고 있습니다.

19. 감사 **(**슬라이드 **33)**

이상으로 중간발표를 마치도록하겠습니다.

감사합니다.