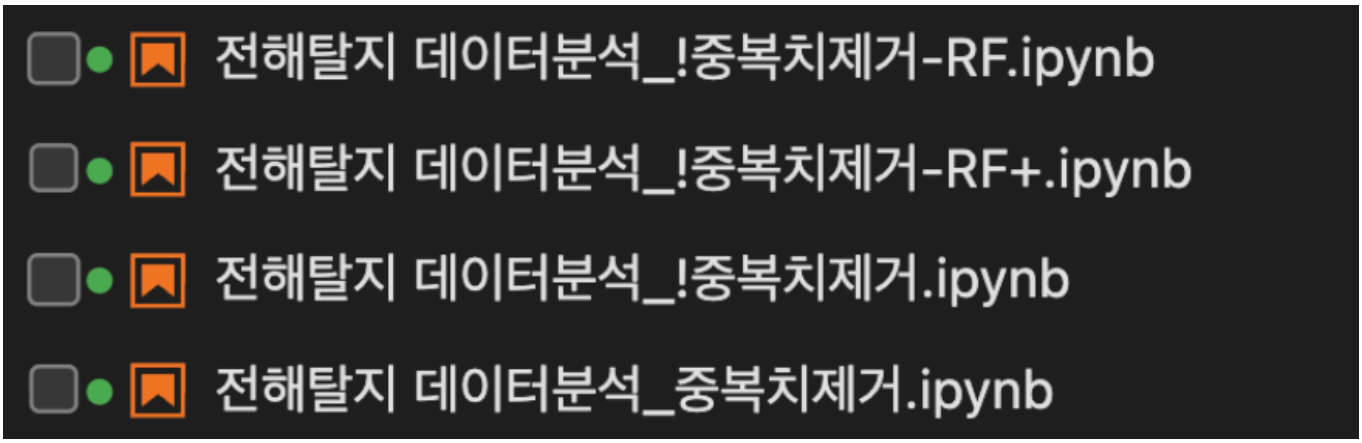
데이터분석 정리

기존 데이터와 다른 점

기존 데이터는 Lot, Time, pH, Temp, Current 총 4개의 컬럼을 가진 센서 데이터 파일과 Error Lot List.csv 파일을 가짐.
본 최종 데이터는 온도, 습도, 압력, 불량 여부로 총 4개의 컬럼을 가짐

코드 파일

`전해탈지_데이터분석*` 코드 파일 - 진행방향은 다 비슷



순서대로

```
중복치 제거 하지 않고 RF모델 사용
중복치 제거 하지 않고 RF 모델 사용, 디벨롭 코드 일부 추가
중복치 제거 하지 않고 RF, XGBoost모델 사용
중복치 제거함, RF, XGBoost 사용
```

데이터 분석 실행 흐름

라이브러리 설치 및 임포트

```
# 필수 라이브러리 설치 (이미 설치되어 있으면 자동으로 건너뜀)
!pip install pandas numpy scikit-learn autokeras seaborn tensorflow
matplotlib pydot graphviz

# 기본 라이브러리
import os
import pandas as pd
import numpy as np
import datetime
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from math import sqrt

# 머신러닝 관련
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    mean_squared_error, confusion_matrix, accuracy_score, precision_score,
    recall_score, f1_score, roc_curve, classification_report
)

# 딥러닝 및 AutoML 관련
import tensorflow as tf
from tensorflow.keras.utils import plot_model
import autokeras as ak

# ✅ 필수 패키지 설치 여부 및 버전 확인 코드
import importlib

# 확인할 패키지 리스트
packages = [
    "pandas",
    "numpy",
    "sklearn",
    "matplotlib",
    "seaborn",
    "xgboost"
]

print("📦 패키지 설치 확인 결과\n")

for pkg in packages:
    try:
        module = importlib.import_module(pkg)
        version = getattr(module, "__version__", "버전 정보 없음")
        print(f"✅ {pkg} 설치됨 (버전: {version})")
    except ModuleNotFoundError:
        print(f"❌ {pkg} 미설치")
```

데이터 전처리

- 파일 불러오기

```
from pathlib import Path
import re

# 1) 경로 설정
ROOT = Path.cwd()
DATA_DIR = ROOT / "data_kmap"
if not DATA_DIR.exists():
    raise FileNotFoundError(f"[ERROR] 데이터 폴더가 없습니다: {DATA_DIR}")

# 2) CSV 파일 수집 (파일명 기준 정렬)
csv_paths = sorted(DATA_DIR.glob("*.csv"), key=lambda p: p.name.lower())

# 3) Error 파일과 센서 파일 분리
error_paths = [p for p in csv_paths if re.search(r'error', p.name,
flags=re.IGNORECASE)]
sensor_paths = [p for p in csv_paths if p not in error_paths]

# 단일 에러 파일이 필요하다면 첫 번째만 선택
error_csv = error_paths[0] if error_paths else None
sensor_csvs = [p.name for p in sensor_paths]   # 파일명만 필요하면 name 사용

# 4) 확인 출력
print("에러 파일:", error_csv.name if error_csv else None)
print("센서 파일 개수:", len(sensor_csvs))
print(sensor_csvs[:10])  # 미리보기(최대 10개)
```

- 파일 병합 및 컬럼 명 정규화

```
# =====  CSV 안전 읽기 =====
def read_csv_safely(path):
    encodings = ["utf-8-sig", "utf-8", "cp949", "euc-kr"]
    for enc in encodings:
        try:
            return pd.read_csv(path, encoding=enc, low_memory=False)
        except Exception:
            continue
    # 위 인코딩으로도 안 되면 최후 시도
    return pd.read_csv(path, low_memory=False)


# =====  컬럼명 정규화 =====
def normalize_columns(df):
    df = df.copy()
    df.columns = (
```

```
        pd.Series(df.columns)
        .astype(str).str.strip()
        .str.replace(r"\s+", "_", regex=True)
        .str.replace(r"[^\w]+", "_", regex=True)
        .str.lower()
    )
    return df


# =====  단일 파일 두 개 확인 (컬럼명 temp/humid/press로 변경 포함) =====
dfs_for_merge = []
for idx, path in enumerate(sensor_paths[:2], start=1):
    df = read_csv_safely(path)
    df = normalize_columns(df)

    # 온도/습도/압력 컬럼명을 temp, humid, press로 통일
    df = df.rename(columns={
        "온도": "temp",
        "temp": "temp",
        "습도": "humid",
        "압력": "press",
        "불량여부": "error"
    })

    print(f"\n[단일 파일 확인 {idx}] {path.name}")
    print("행 수:", len(df))
    print("열 수:", df.shape[1])
    print(df.tail(3))   # 끝 3행 미리보기

    dfs_for_merge.append(df)


# =====  모든 센서 CSV의 요약 (파일 수, 총 행 수) =====
per_file = []
for p in sensor_paths:
    dfi = read_csv_safely(p)
    per_file.append({"file": p.name, "rows": len(dfi), "cols":
dfi.shape[1]})

total_rows = sum(x["rows"] for x in per_file)

print("\n[전체 센서 CSV 요약]")
print("파일 수:", len(per_file))
print("총 행 수 합계:", total_rows)
```

```
# =====  단일 파일 두 개를 결합해 새로운 데이터프레임 생성 =====
combined_df = pd.concat(dfs_for_merge, ignore_index=True)

print("\n[두 단일 파일 병합 결과]")
print("새 데이터프레임 행 수:", len(combined_df))
print("새 데이터프레임 열 수:", combined_df.shape[1])
print(combined_df.head(5))
```

- 결측치 제거, 이상치 제거 (중복치는 제거하지 않음)

```
import pandas as pd
import numpy as np
from scipy import stats

# ========================================
#  원본 보존 & 사본 생성
# ========================================
# 원본 데이터: combined_df (10970행)
print(f"원본 데이터 행 수: {len(combined_df)}")

# 분석용 사본 생성
base_df = combined_df.copy()
print(f"사본(base_df) 생성 완료. 행 수: {len(base_df)}, 열 수:
{base_df.shape[1]}\n")

# ========================================
# 결측치 제거 또는 대체
# ========================================
# 컬럼별 결측치 개수 확인
print("📋 [결측치 개수 확인]")
print(base_df.isna().sum())

# 결측치가 많지 않다면 dropna()로 제거,
# 많다면 평균값/중앙값으로 대체할 수도 있음
missing_before = base_df.isnull().sum().sum()

# 예시 1: 결측치가 매우 적을 때 – 결측 행 제거
df_no_missing = base_df.dropna().reset_index(drop=True)

# 예시 2: (선택) 결측치를 평균으로 대체하고 싶을 때는 아래 코드 사용
# df_no_missing = base_df.fillna(base_df.mean(numeric_only=True))

missing_after = df_no_missing.isnull().sum().sum()
print(f"\n결측치 제거 전: {missing_before}, 제거 후: {missing_after}")
print(f"결측치 제거 후 행 수: {len(df_no_missing)}\n")
```

```
# ========================================
#  Z-score 기반 이상치 제거
# ========================================
def remove_outliers_zscore(df, cols, threshold=3.0):
    """
    지정된 컬럼들에 대해 Z-score 기반 이상치 제거
    |z| > threshold 인 행 제거
    """
    df = df.copy()
    z_scores = np.abs(stats.zscore(df[cols], nan_policy='omit'))
    mask = (z_scores < threshold).all(axis=1)
    clean_df = df[mask].reset_index(drop=True)
    return clean_df, mask

# 이상치 제거 대상 수치 컬럼 지정
num_cols = ["temp", "humid", "press"]

# Z-score 기반 이상치 제거 실행
df_clean2, mask = remove_outliers_zscore(df_no_missing, num_cols,
threshold=3.0)

print("📊 [Z-score 이상치 제거 결과]")
print(f"제거 기준: |z| > 3.0")
print(f"이상치 제거 전 행 수: {len(df_no_missing)}")
print(f"이상치 제거 후 행 수: {len(df_clean2)}")
print(f"총 제거된 행 수: {len(df_no_missing) - len(df_clean2)}")
print(f"제거 비율: {(1 - len(df_clean2)/len(df_no_missing))*100:.2f}%\n")

# ========================================
# 최종 데이터 정보 확인
# ========================================
print("✅ [최종 정제 데이터 정보]")
print(df_clean2.info())
print("\n결측치 개수:")
print(df_clean2.isna().sum())
print("\n통계 요약:")
print(df_clean2[num_cols].describe().round(3))

# 최종 정제 데이터 저장(선택)
# df_clean.to_csv("cleaned_manufacturing_data.csv", index=False)
```

- 데이터 품질 계산

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# ========================================
# CQI / UQI / VQI / DQI 계산 함수
# ========================================
def compute_quality_indices(df, cols, valid_ranges=None):
    """
    df : 평가할 데이터프레임 (df_clean)
    cols : 평가할 컬럼 리스트 (예: ["temp", "humid", "press"])
    valid_ranges : {컬럼명: (min, max)} 없으면 IQR 기반 범위로 자동 설정
    """
    results = []

    for col in cols:
        series = df[col]
        total_count = len(series)
        missing_count = series.isnull().sum()
        unique_count = series.nunique(dropna=True)

        # (1) Completeness Quality Index (CQI)
        cqi = (1 - missing_count / total_count) * 100

        # (2) Uniqueness Quality Index (UQI)
        uqi = (unique_count / total_count) * 100

        # (3) Validity Quality Index (VQI)
        if valid_ranges and col in valid_ranges:
            vmin, vmax = valid_ranges[col]
        else:
            q1, q3 = series.quantile([0.25, 0.75])
            iqr = q3 - q1
            vmin, vmax = q1 - 1.5 * iqr, q3 + 1.5 * iqr

        valid_count = series.between(vmin, vmax).sum()
        vqi = (valid_count / total_count) * 100

        # (4) Data Quality Index (DQI) = 세 지수 평균
        dqi = (cqi + uqi + vqi) / 3

        results.append({
            "column": col,
            "CQI": round(cqi, 2),
            "UQI": round(uqi, 2),
            "VQI": round(vqi, 2),
```

```
            "DQI": round(dqi, 2)
        })

    return pd.DataFrame(results)


# ----------------------------------------
#  df_clean 기준 품질 지수 계산
# ----------------------------------------
num_cols = ["temp", "humid", "press"]

quality_df = compute_quality_indices(df_clean2, num_cols)

print("📊 [정제 데이터(df_clean2) 품질 지수 (CQI / UQI / VQI / DQI)]")
print(quality_df)

overall_dqi = quality_df["DQI"].mean()
print(f"\n➡ 정제 데이터 평균 데이터 품질 지수 (DQI): {overall_dqi:.2f} / 100")
```

- 전처리 후 데이터 정보 요약 출력 및 시각화 출력

```
# ----------------------------------
#  최종 정제 데이터 정보 요약
# ----------------------------------
print("✅ [최종 정제 데이터(df_clean2) 기본 정보]")
print("shape:", df_clean2.shape)
print("\ncolumns:", list(df_clean2.columns))
print("\ninfo:")
print(df_clean2.info())
print("\nNA count:")
print(df_clean2.isna().sum())

# 수치 컬럼만 따로
df_num = df_clean2[num_cols]

print("\n📈 [수치 컬럼 기술 통계량]")
print(df_num.describe().round(3))

# ----------------------------------
#  히스토그램 (분포 확인)
# ----------------------------------
df_num.hist(bins=30, figsize=(10, 6))
plt.suptitle("Distribution of temp / humid / press (after cleaning)",
fontsize=13)
plt.tight_layout(rect=[0, 0, 1, 0.95])
```

```
plt.show()

# ----------------------------------
#  상관관계 히트맵
# ----------------------------------
corr = df_num.corr()
plt.figure(figsize=(4, 3))
sns.heatmap(corr, annot=True, fmt=".2f", cmap="RdYlBu_r", square=True)
plt.title("Correlation (temp - humid - press)")
plt.tight_layout()
plt.show()

# ----------------------------------
#  박스플롯 (이상치 제거 후 분포 확인)
# ----------------------------------
plt.figure(figsize=(8, 5))
sns.boxplot(data=df_num, orient="h")
plt.title("Boxplot of temp / humid / press (after cleaning)")
plt.xlabel("value")
plt.tight_layout()
plt.show()

# ----------------------------------
#  불량 여부 분포 (라벨 분포 확인)
# ----------------------------------
plt.figure(figsize=(4, 3))
df_clean2["error"].value_counts().sort_index().plot(kind="bar")
plt.xticks([0, 1], ["normal(0)", "error(1)"], rotation=0)
plt.title("Label Distribution (error)")
plt.ylabel("count")
plt.tight_layout()
plt.show()
```

머신러닝

- 훈련 데이터, 테스트 데이터 분리

```
# ========================================
# 1️⃣ 데이터 준비
# ========================================
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, roc_auc_score,
classification_report
```

```
)
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
import pandas as pd

# 입력(X), 타깃(y) 분리
X = df_clean2[["temp", "humid", "press"]]
y = df_clean2["error"]

# 학습용/테스트용 데이터 분할 (8:2)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)

print(f"훈련 데이터: {X_train.shape}, 테스트 데이터: {X_test.shape}")


# ========================================
# 2️⃣ 데이터 스케일링 (표준화)
# ========================================
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

- 모델 학습 - 두 가지 모델 사용 RF, XGBoost

```
# ========================================
# 3️⃣ 모델 학습: RandomForest vs XGBoost
# ========================================
rf_model = RandomForestClassifier(n_estimators=200, random_state=42)
xgb_model = xgb.XGBClassifier(
    n_estimators=200,
    learning_rate=0.05,
    max_depth=5,
    subsample=0.8,
    colsample_bytree=0.8,
    eval_metric="logloss",
    random_state=42
)

rf_model.fit(X_train_scaled, y_train)
xgb_model.fit(X_train_scaled, y_train)


# ========================================
```

```
# 4️⃣ 예측 및 성능 평가
# ========================================
rf_pred = rf_model.predict(X_test_scaled)
xgb_pred = xgb_model.predict(X_test_scaled)

models = {"RandomForest": rf_pred, "XGBoost": xgb_pred}

for name, pred in models.items():
    print(f"\n📊 [{name} 성능 요약]")
    print(f"Accuracy : {accuracy_score(y_test, pred):.4f}")
    print(f"Precision: {precision_score(y_test, pred):.4f}")
    print(f"Recall   : {recall_score(y_test, pred):.4f}")
    print(f"F1-score : {f1_score(y_test, pred):.4f}")
    print(f"AUC      : {roc_auc_score(y_test, pred):.4f}")
    print("\nDetailed report:\n", classification_report(y_test, pred,
digits=4))
```

- 성능 평가 - 변수 중요도

```
import matplotlib.pyplot as plt
import seaborn as sns

# 랜덤포레스트 변수 중요도
rf_importance = pd.DataFrame({
    "Feature": X.columns,
    "Importance": rf_model.feature_importances_
}).sort_values("Importance", ascending=False)

plt.figure(figsize=(5,3))
sns.barplot(data=rf_importance, x="Importance", y="Feature")
plt.title("Feature Importance - RandomForest")
plt.tight_layout()
plt.show()

# XGBoost 변수 중요도
xgb_importance = pd.DataFrame({
    "Feature": X.columns,
    "Importance": xgb_model.feature_importances_
}).sort_values("Importance", ascending=False)

plt.figure(figsize=(5,3))
sns.barplot(data=xgb_importance, x="Importance", y="Feature")
plt.title("Feature Importance - XGBoost")
plt.tight_layout()
plt.show()
```

- 성능 평가 - Roc Curve, Precision-Recall Curve

```
import matplotlib.pyplot as plt
from sklearn.metrics import (
    roc_curve, auc,
    precision_recall_curve, average_precision_score
)

# ----------------------------------------
# 1️⃣ 예측 확률 추출 (양성 클래스=1에 대한 확률)
# ----------------------------------------
rf_probs = rf_model.predict_proba(X_test_scaled)[:, 1]
xgb_probs = xgb_model.predict_proba(X_test_scaled)[:, 1]

# ----------------------------------------
# 2️⃣ ROC Curve 계산
# FPR(위양성률), TPR(재현율) 계산
# ----------------------------------------
rf_fpr, rf_tpr, _ = roc_curve(y_test, rf_probs)
xgb_fpr, xgb_tpr, _ = roc_curve(y_test, xgb_probs)
rf_auc = auc(rf_fpr, rf_tpr)
xgb_auc = auc(xgb_fpr, xgb_tpr)

# ----------------------------------------
# 3️⃣ Precision-Recall Curve 계산
# ----------------------------------------
rf_prec, rf_rec, _ = precision_recall_curve(y_test, rf_probs)
xgb_prec, xgb_rec, _ = precision_recall_curve(y_test, xgb_probs)
rf_ap = average_precision_score(y_test, rf_probs)
xgb_ap = average_precision_score(y_test, xgb_probs)

# ----------------------------------------
# 4️⃣ 그래프 시각화 (2개 서브플롯)
# ----------------------------------------
plt.figure(figsize=(12, 5))

# ROC Curve
plt.subplot(1, 2, 1)
plt.plot(rf_fpr, rf_tpr, color="blue", lw=2,
         label=f"RandomForest (AUC = {rf_auc:.4f})")
plt.plot(xgb_fpr, xgb_tpr, color="orange", lw=2,
         label=f"XGBoost (AUC = {xgb_auc:.4f})")
plt.plot([0, 1], [0, 1], color="gray", linestyle="--")
plt.title("ROC Curve", fontsize=13)
plt.xlabel("False Positive Rate (1 - Specificity)")
plt.ylabel("True Positive Rate (Recall)")
```

```
plt.legend(loc="lower right")
plt.grid(alpha=0.3)

# Precision-Recall Curve
plt.subplot(1, 2, 2)
plt.plot(rf_rec, rf_prec, color="blue", lw=2,
         label=f"RandomForest (AP = {rf_ap:.4f})")
plt.plot(xgb_rec, xgb_prec, color="orange", lw=2,
         label=f"XGBoost (AP = {xgb_ap:.4f})")
plt.title("Precision-Recall Curve", fontsize=13)
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.legend(loc="lower left")
plt.grid(alpha=0.3)

plt.tight_layout()
plt.show()
```

최종 결과물 파일 형태로 저장

```
# === 최종 결과물 저장 스크립트 ===
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import joblib

from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    roc_auc_score, average_precision_score, balanced_accuracy_score,
    classification_report, confusion_matrix, ConfusionMatrixDisplay,
    roc_curve, precision_recall_curve, auc
)

# 0️⃣ 아티팩트 폴더 생성
ART_DIR = "artifacts_dup_data"
os.makedirs(ART_DIR, exist_ok=True)

def save_model_artifacts(model, model_name, X_test, y_test, feature_names):
    """하나의 모델에 대해 지표/결과/이미지/모델파일을 모두 저장"""
    model_tag = model_name.lower().replace("classifier", "").replace(" ",
"")

    # ===== 1) 예측 및 점수 계산 =====
    y_true = y_test.values if isinstance(y_test, pd.Series) else
```

```
np.asarray(y_test)
    y_pred = model.predict(X_test)
    # 양성 클래스(1)에 대한 확률 또는 점수
    if hasattr(model, "predict_proba"):
        y_score = model.predict_proba(X_test)[:, 1]
    else:
        # decision_function만 있는 경우 대비
        y_score = model.decision_function(X_test)

    # 혼동행렬
    cm = confusion_matrix(y_true, y_pred)
    tn, fp, fn, tp = cm.ravel()

    # 주요 지표 계산
    metrics = {
        "model":              model_name,
        "accuracy":           accuracy_score(y_true, y_pred),
        "precision":          precision_score(y_true, y_pred),
        "recall":             recall_score(y_true, y_pred),
        "f1_score":           f1_score(y_true, y_pred),
        "roc_auc":            roc_auc_score(y_true, y_score),
        "avg_precision_AP":   average_precision_score(y_true, y_score),  #
PR AUC
        "balanced_accuracy":  balanced_accuracy_score(y_true, y_pred),
        "specificity":        tn / (tn + fp + 1e-12),  # 정상 중 정상으로 맞춘 비
율
        "n_test_samples":     len(y_true),
        "n_positive":         int(y_true.sum()),
        "n_negative":         int((y_true == 0).sum()),
    }

    # ===== 2) metrics 저장 =====
    metrics_path = os.path.join(ART_DIR, f"metrics_{model_tag}.csv")
    pd.DataFrame(list(metrics.items()), columns=["metric",
"value"]).to_csv(metrics_path, index=False)
    print(f"[저장 완료] 평가 지표 → {metrics_path}")

    # classification_report 저장
    report_txt = classification_report(y_true, y_pred, digits=4)
    report_path = os.path.join(ART_DIR,
f"classification_report_{model_tag}.txt")
    with open(report_path, "w", encoding="utf-8") as f:
        f.write(f"Model: {model_name}\n\n")
        f.write(report_txt)
    print(f"[저장 완료] 분류 리포트 → {report_path}")
```

```python
# ===== 3) 테스트 예측 결과 저장 =====
pred_df = pd.DataFrame({
    "y_true":  y_true,
    "y_pred":  y_pred,
    "y_score": y_score,
})
pred_path = os.path.join(ART_DIR, f"test_predictions_{model_tag}.csv")
pred_df.to_csv(pred_path, index=False)
print(f"[저장 완료] 예측 결과 → {pred_path}")

# ===== 4) 혼동행렬 이미지 & CSV =====
# 이미지
disp = ConfusionMatrixDisplay.from_predictions(y_true, y_pred)
plt.title(f"Confusion Matrix ({model_name})")
cm_img_path = os.path.join(ART_DIR, f"confusion_matrix_{model_tag}.png")
plt.savefig(cm_img_path, bbox_inches="tight")
plt.close()
print(f"[저장 완료] 혼동행렬 이미지 → {cm_img_path}")

# 수치 CSV
cm_df = pd.DataFrame(cm,
                     index=["true_0", "true_1"],
                     columns=["pred_0", "pred_1"])
cm_csv_path = os.path.join(ART_DIR, f"confusion_matrix_{model_tag}.csv")
cm_df.to_csv(cm_csv_path)
print(f"[저장 완료] 혼동행렬 값 → {cm_csv_path}")

# ===== 5) ROC Curve 이미지 =====
fpr, tpr, _ = roc_curve(y_true, y_score)
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(5, 4))
plt.plot(fpr, tpr, label=f"AUC={roc_auc:.3f}", color="darkorange")
plt.plot([0, 1], [0, 1], "k--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title(f"ROC Curve ({model_name})")
plt.legend()
plt.grid(True)
roc_path = os.path.join(ART_DIR, f"roc_curve_{model_tag}.png")
plt.savefig(roc_path, bbox_inches="tight")
plt.close()
print(f"[저장 완료] ROC Curve 이미지 → {roc_path}")

# ===== 6) Precision–Recall Curve 이미지 =====
prec, rec, _ = precision_recall_curve(y_true, y_score)
ap = average_precision_score(y_true, y_score)
```

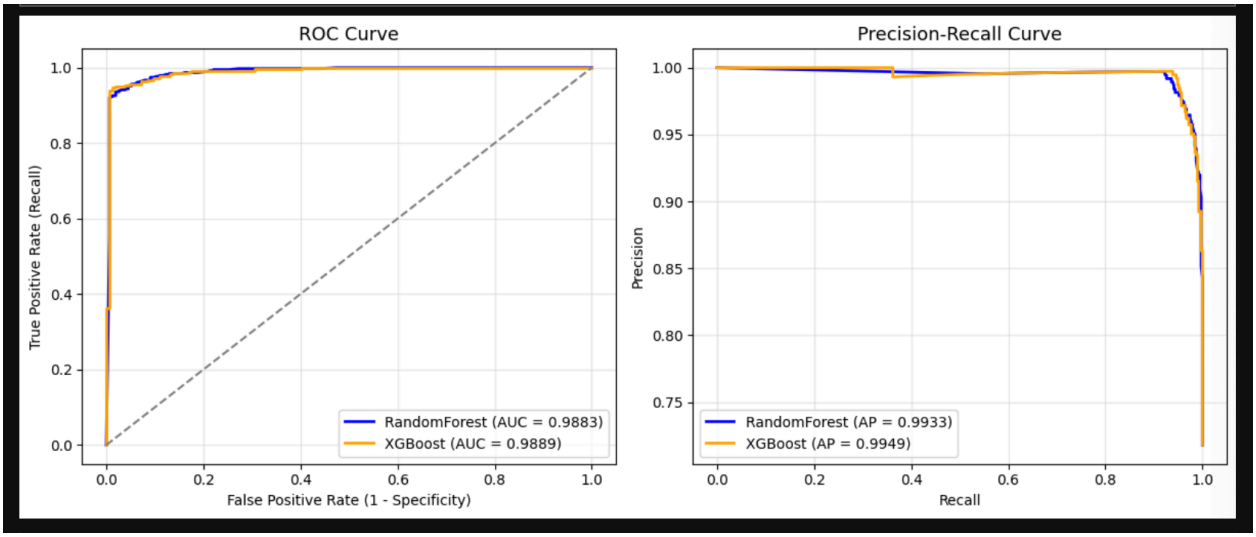
```
    plt.figure(figsize=(5, 4))
    plt.plot(rec, prec, color="blue", label=f"AP={ap:.3f}")
    plt.xlabel("Recall")
    plt.ylabel("Precision")
    plt.title(f"Precision-Recall Curve ({model_name})")
    plt.legend()
    plt.grid(True)
    pr_path = os.path.join(ART_DIR, f"pr_curve_{model_tag}.png")
    plt.savefig(pr_path, bbox_inches="tight")
    plt.close()
    print(f"[저장 완료] PR Curve 이미지 → {pr_path}")

    # ===== 7) Feature Importance (트리 계열만) =====
    if hasattr(model, "feature_importances_"):
        fi = pd.Series(model.feature_importances_,
index=feature_names).sort_values(ascending=False)
        fi_path = os.path.join(ART_DIR,
f"feature_importance_{model_tag}.csv")
        fi.to_csv(fi_path, header=["importance"])
        print(f"[저장 완료] Feature Importance → {fi_path}")

    # ===== 8) 모델 파일(joblib) =====
    model_path = os.path.join(ART_DIR, f"model_{model_tag}.joblib")
    joblib.dump(model, model_path)
    print(f"[저장 완료] 모델 파일 → {model_path}")
    print("-" * 60)


# 🔁 실제 저장 실행 (RandomForest & XGBoost)
feature_names = X.columns  # df_clean2에서 사용한 입력 컬럼들

save_model_artifacts(rf_model, "RandomForestClassifier", X_test_scaled,
y_test, feature_names)
save_model_artifacts(xgb_model, "XGBClassifier",         X_test_scaled,
y_test, feature_names)

print(f"\n🎉 모든 결과 저장 완료 → {os.path.abspath(ART_DIR)}")
```

진행 과정 이슈

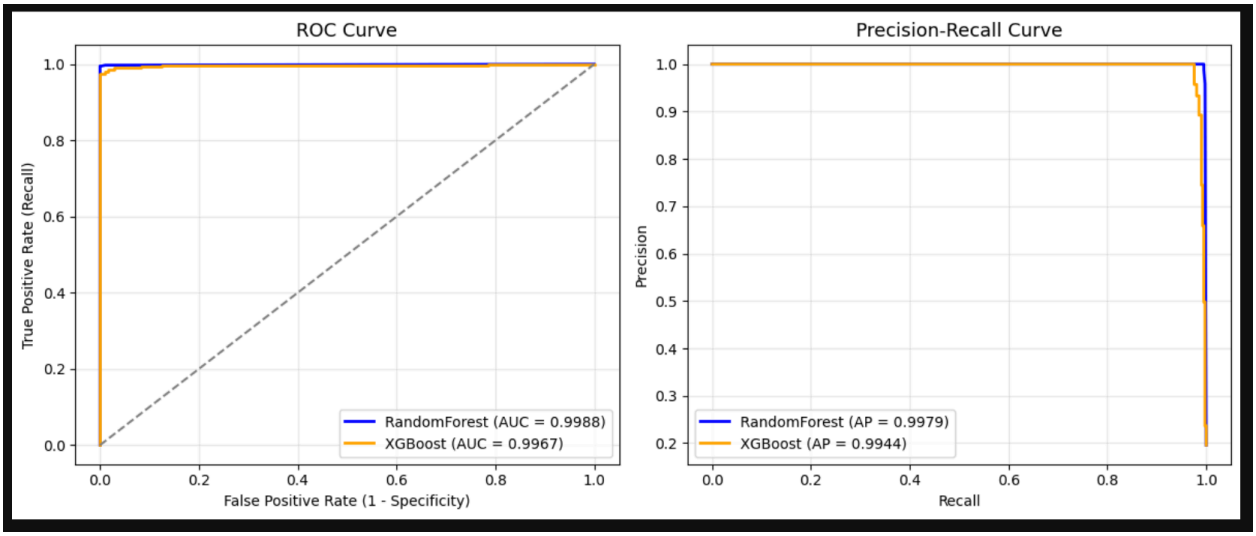
1. RF 모델 vs XGBoost 모델
    1. 두 개 다 수행 후, Random Forest 모델의 성능이 좀 더 우수하여 RF 모델을 사용하기로 함
2. 데이터 전처리 과정에서 중복치를 제거할지 말지
    - 중복치를 제거하면 10813개의 데이터 중 8000여개가 제거되는 문제 발생, 그러면 DQI가 99.97로 데이터의 품질은 좋게 나오나, 과적합 발생 가능성이 매우 높음

```
♻️ [중복 제거 결과]
중복 제거 전 행 수: 10813
중복 제거 후 행 수: 2756
제거된 중복 행 수: 8057
제거 비율: 74.51%
```

- 그래서, 중복치를 제거하지 않고 많은 데이터를 사용하여 품질 모델 예측 결과의 정확성을 높이기로 결정
  - 중복치를 제거하지 않은 데이터로 모델 학습을 하였을 때 성능이 훨씬 좋은 것을 확인

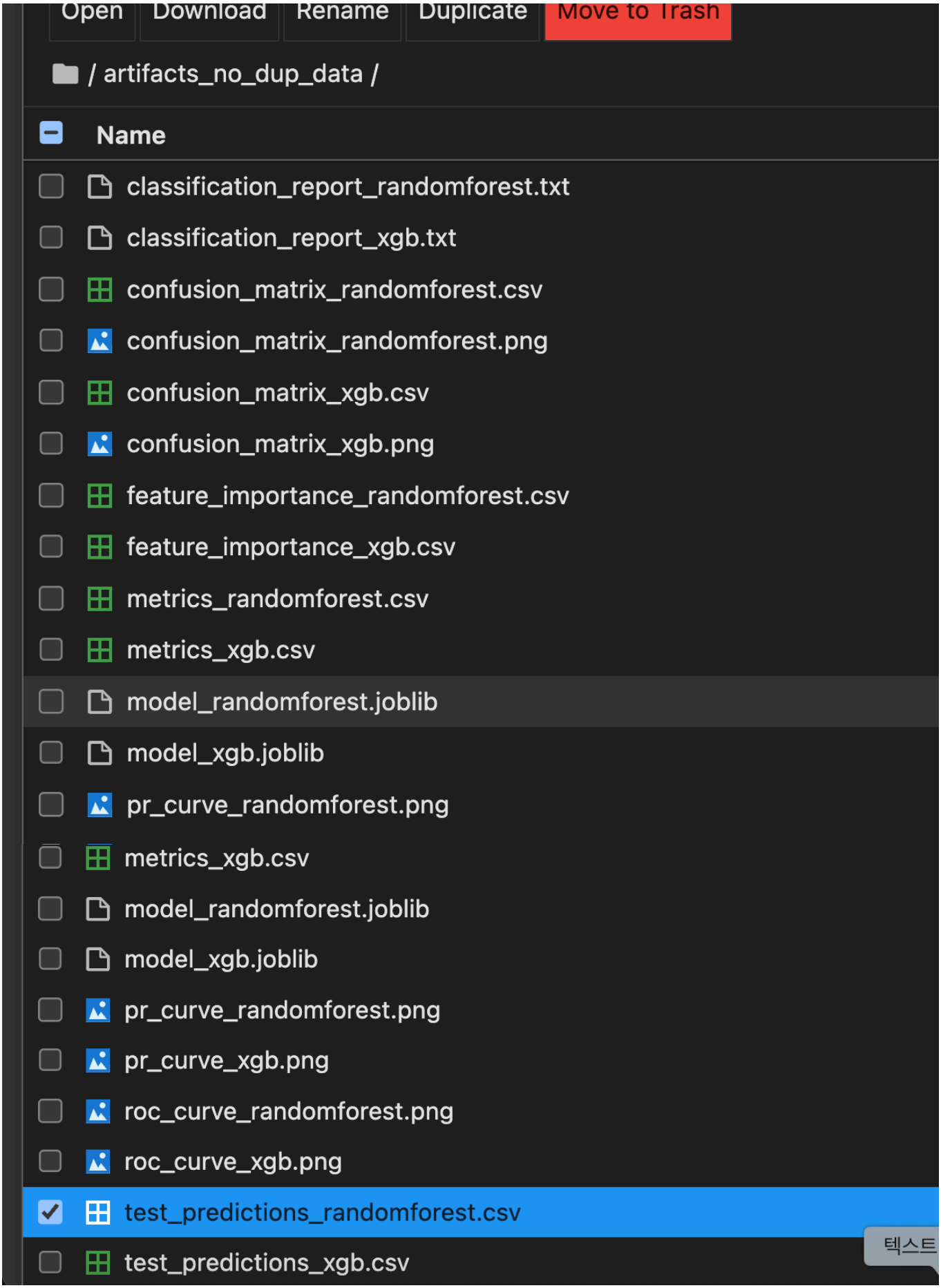


- 중복치 제거 o



- 중복치 제거 x

저장된 결과물


Open
Download
Rename
Duplicate
Move to Trash
/ artifacts_no_dup_data /
Name
classification_report_randomforest.txt
classification_report_xgb.txt
confusion_matrix_randomforest.csv
confusion_matrix_randomforest.png
confusion_matrix_xgb.csv
confusion_matrix_xgb.png
feature_importance_randomforest.csv
feature_importance_xgb.csv
metrics_randomforest.csv
metrics_xgb.csv
model_randomforest.joblib
model_xgb.joblib
pr_curve_randomforest.png
metrics_xgb.csv
model_randomforest.joblib
model_xgb.joblib
pr_curve_randomforest.png
pr_curve_xgb.png
roc_curve_randomforest.png
roc_curve_xgb.png
test_predictions_randomforest.csv
test_predictions_xgb.csv
텍스트

jupyter classification_report_randomforest.txt Last Checkpoint: yesterday

File Edit View Settings Help

```
Model: RandomForestClassifier

              precision    recall  f1-score   support

           0     0.9108    0.9286    0.9196       154
           1     0.9717    0.9643    0.9680       392

    accuracy                         0.9542       546
   macro avg     0.9413    0.9464    0.9438       546
weighted avg     0.9545    0.9542    0.9543       546
```

jupyter confusion_matrix_randomforest.csv Last Checkpoint: yesterday

File Edit View Settings Help

Delimiter: ,

		pred_0	pred_1
1	true_0	143	11
2	true_1	14	378

jupyter feature_importance_randomforest.csv Last Checkpoint: yesterday

File Edit View Settings Help

Delimiter: ,

		importance
1	temp	0.4462733427139276
2	press	0.35355270868271094
3	humid	0.2001739486033614

jupyter metrics_randomforest.csv Last Checkpoint: yesterday

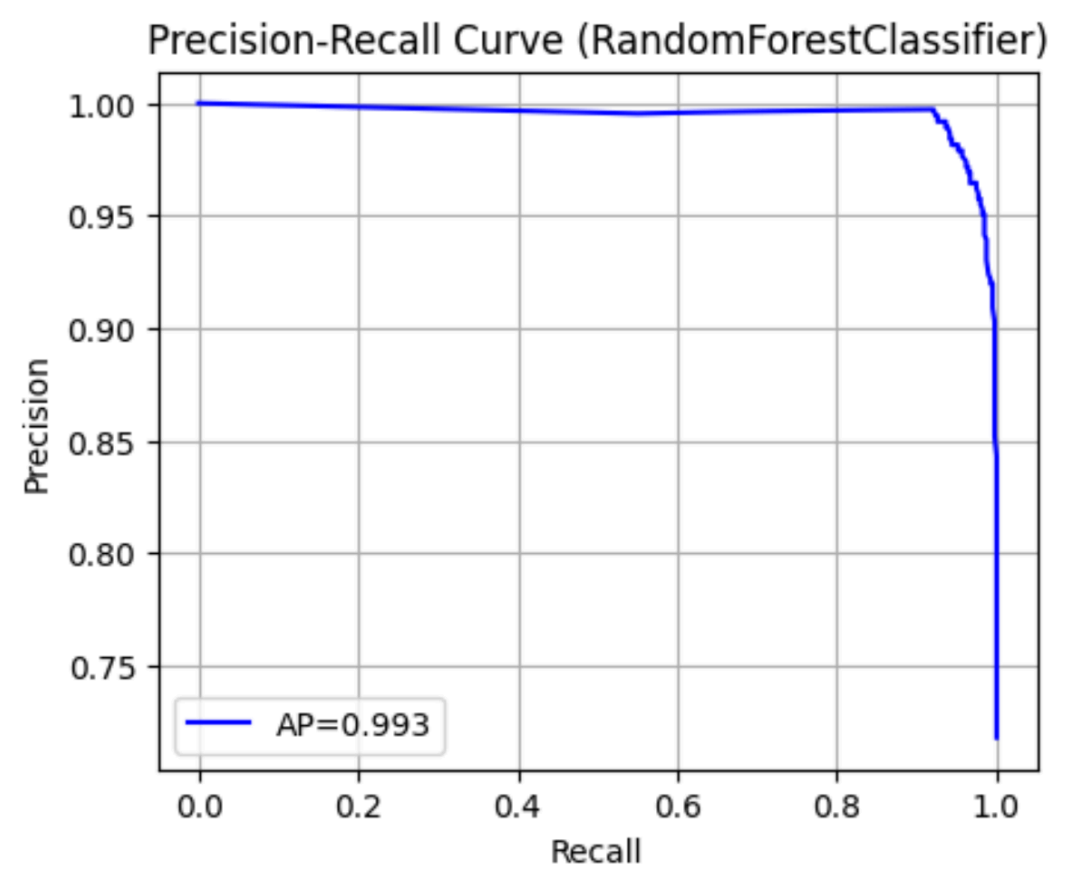
File Edit View Settings Help

Delimiter: ,

	metric	value
1	model	RandomForestClassifier
2	accuracy	0.9542124542124543
3	precision	0.9717223650385605
4	recall	0.9642857142857143
5	f1_score	0.967989756722151
6	roc_auc	0.9882802146832759
7	avg_precision_AP	0.99332403107193
8	balanced_accuracy	0.9464285714285714
9	specificity	0.9285714285714226
10	n_test_samples	546
11	n_positive	392
12	n_negative	154

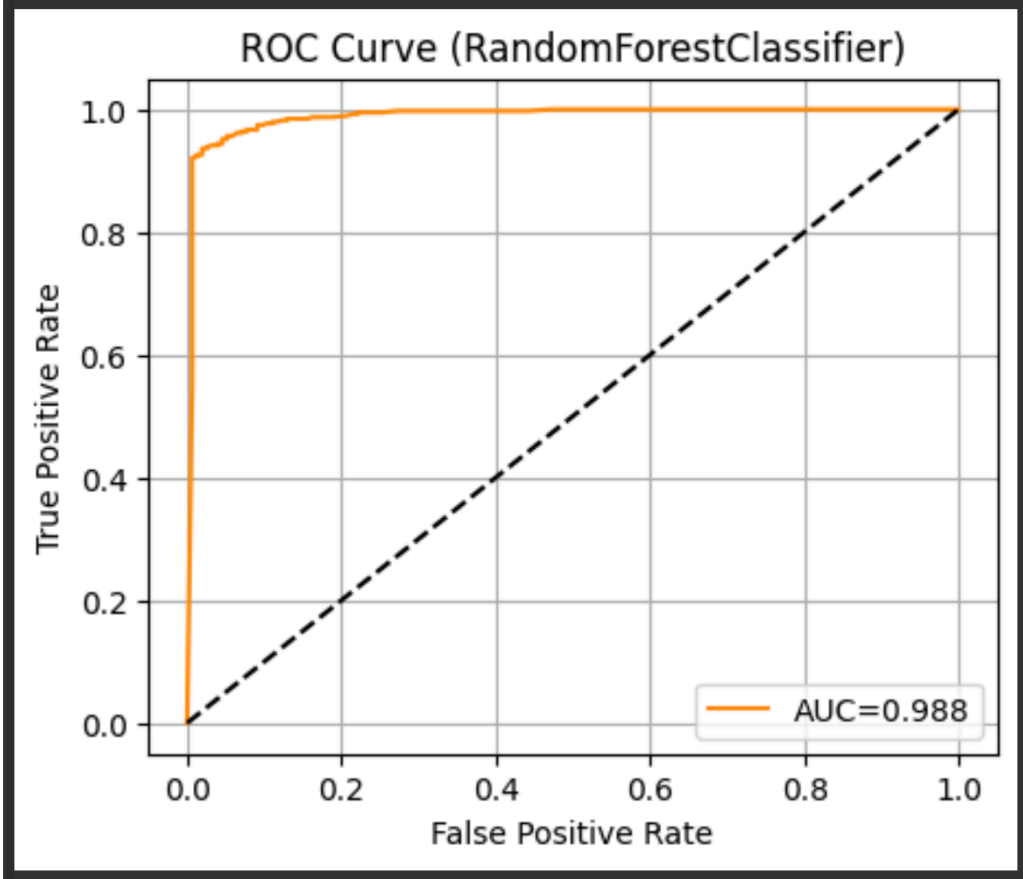
jupyter pr_curve_randomforest.png Last Checkpoint: yesterday

File Edit View Settings Help



jupyter roc_curve_randomforest.png Last Checkpoint: yesterday

File Edit View Settings Help



jupyter test_predictions_randomforest.csv Last Checkpoint: yesterday

File Edit View Settings Help

Delimiter: ,

	y_true	y_pred	y_score
1	1	1	0.99
2	1	1	0.985
3	1	1	0.99
4	0	0	0.06
5	1	1	0.985
6	0	0	0.0
7	1	1	0.99
8	1	1	0.965
9	1	1	1.0
10	1	1	1.0
11	1	1	0.82
12	1	1	1.0
13	0	0	0.0
14	1	1	1.0
15	0	0	0.04

디벨롭 방향 - RF모델 사용, 통계적 권장 을 통한 정상 데이터 임계값 확보

[16]:
```
temp_range  = (df_clean2["temp"].quantile(0.05),  df_clean2["temp"].quantile(0.95))
humid_range = (df_clean2["humid"].quantile(0.05), df_clean2["humid"].quantile(0.95))
press_range = (df_clean2["press"].quantile(0.05), df_clean2["press"].quantile(0.95))

safe_summary, safe_points = estimate_safe_region(
    rf_model, scaler,
    temp_range, humid_range, press_range,
    prob_threshold=0.05  # "불량 확률 5% 미만"을 안전으로 정의
)

print_safe_summary(safe_summary, prob_threshold=0.05)
```

```
📌 [모델 기준 안전 운전 범위 추정]
   (불량 확률 < 5.0% 인 영역 기준)

 - 온도 temp   : 43.93  ~  54.79
 - 습도 humid  : 25.31  ~  36.48
 - 압력 press  : 65.10  ~  82.71

 - 안전 영역 내 격자점 수 : 647 개
```

[11]:
```python
# 정제 데이터 중 정상인 것만 사용
normal = df_clean2[df_clean2["error"] == 0]

SAFE_RANGES = {
    "temp":  (normal["temp"].quantile(0.10),  normal["temp"].quantile(0.90)),
    "humid": (normal["humid"].quantile(0.10), normal["humid"].quantile(0.90)),
    "press": (normal["press"].quantile(0.10), normal["press"].quantile(0.90)),
}

print("[통계적 권장 운전 범위]")
for k, (lo, hi) in SAFE_RANGES.items():
    print(f"{k}: {lo:.3f} ~ {hi:.3f}")
```

```
[통계적 권장 운전 범위]
temp: 46.021 ~ 53.798
humid: 26.346 ~ 33.534
press: 68.860 ~ 81.536
```