# Threshold 자동계산 코드추가

## 목표

- 현재 모든 모델이 학습 후 deafault 값(1)으로 설정됨
- 학습 완료 후 자동으로 Threshold 를 구하는 기능을 추가하고자 함

## 기존 방식 확인, 변경 및 추가사항

- 기존 방식, 변경사항

  Model_training (api_server/apis/multiproc/proc/proc_annoy.py)

  influx 에서 데이터 조회 -> 전체 데이터 concat 후 scaler fit -> 각 pkl 파일 scaler 적용
  -> 조정된 데이터로 모델 학습 -> 모델 및 스케일러 저장

  influx 에서 데이터 조회 -> train_test 데이터 split -> train 데이터로만 scaler fit ->
  -> train, test 데이터 scaler 적용 -> train 데이터로 모델 학습 -> 모델 및 스케일러 저장
  -> 학습된 모델로 test 진행 -> thrshold 계산 -> threshold csv 파일로 저장

- 추가사항

  threshold 계산함수가 들어있는 .py 생성 (src₩tag_setting₩threshold_calculator.py)

## 변경 코드 확인

1. 데이터 조회 후 train, test 데이터 split(test 구간을 전체 데이터 앞 5 일로 설정)

```
# 3. Split train/test (앞 5일을 테스트 데이터로)
split_point = df_concat.index.min() + pd.Timedelta(days=5)
test_df = df_concat[df_concat.index < split_point]
train_df = df_concat[df_concat.index >= split_point]
```

2. train 데이터로만 scaler fit 한 후 scaler 적용

```
# 4. Fit scaler only on train data
model.get_scaler(
    data_input=train_df, data_output=train_df, scaler_type=request.model_params.scaler_type, convert_range=(0, 1)
)

train_scaled = pd.DataFrame(model.scaler.transform(train_df), columns=train_df.columns, index=train_df.index)
test_scaled = pd.DataFrame(model.scaler.transform(test_df), columns=test_df.columns, index=test_df.index)
```

3. 학습 완료 후 test 진행(기존 test 에서 진행했던 방식 사용)

```
# 6. Predict on test data
logger.info(f"Start model test (internal after training) | modeltype={modeltype}, modelname={model.modelname}")

if len(test_scaled) < model.window_size:
    raise ValueError(f"Test data size ({len(test_scaled)}) is smaller than window_size ({model.window_size}). Cannot

test_vectors = model._vectorize(test_scaled)
preds = []
for _, test_row in enumerate(test_vectors):
    pred = model.predict(test_row)
    preds.append(pred)
preds = np.array(preds)

# inverse transform
inputs_inversed = test_df.values[model.window_size - 1:]
preds_inversed = model.scaler.inverse_transform(preds)
logger.info(f"Completed model test (internal after training) | modeltype={modeltype}, modelname={model.modelname}")

# threshold 계산
threshold_df = calculate_thresholds(
    inputs=inputs_inversed,
    preds=preds_inversed,
    tagnames=model.tagnames,
    top_percent=0.95
)

save_path = os.path.join('src','tag_setting', 'settings',f"{modeltype}_{model.modelname}_thresholds.csv")
threshold_df.to_csv(save_path, index=False)
logger.info(f"Saved threshold settings CSV at {save_path}")
```

4. 사용 계산식(threshold 계산식 4 종류)

```
# 1. 잔차평균 + 6 * 잔차표준편차
mean_plus_6std = np.nanmean(tag_residual_abs) + 6 * np.nanstd(tag_residual_abs)

# 2. 전체잔차의 min값
residual_min = np.nanmin(tag_residual_abs)

# 3. 전체잔차의 max값
residual_max = np.nanmax(tag_residual_abs)

# 4. raw_data와의 이동평균 잔차값
moving_avg_window = 10  # 기본 10개 윈도우
raw_series = inputs[:, idx]
raw_moving_avg = pd.Series(raw_series).rolling(window=moving_avg_window, min_periods=1, center=True).mean().to_numpy()
moving_avg_residual = np.nanmean(np.abs(raw_series - raw_moving_avg))

# 5. 잔차 상위 top_percent
sorted_residuals = np.sort(tag_residual_abs)
cutoff_idx = int(len(sorted_residuals) * top_percent)
top_percent_threshold = sorted_residuals[cutoff_idx - 1]  # 0.95 위치 값
```

5. csv 파일 결과 확인

training-server₩src₩tag_setting₩settings 폴더 안에 modeltype_modelname 으로 저장됨

## 추후 계획

현재 ANNOY 모델에만 적용된 상태임으로 다른 모델에도 적용시킬 예정