What This Algorithm Does:

Step	Logic
Smooths the data	Removes sensor noise to detect true trend
Detects a stable window	Finds a low-slope, low-noise period — treats it as "normal" behavior
Compares slope	Measures how fast values are rising now, compared to baseline
Self-learns thresholds	No hardcoded values; learns from past system behavior
Classifies state	Normal, Warning, or Critical based on trend steepnes

STEP-BY-STEP EXPLANATION

Step 1: Generate or Input the Sensor Data

python
CopyEdit
```python
sensor_data = [...100 readings...]
```

You can use real sensor readings, or generate synthetic ones like in this code:

- First 70 readings → **flat/stable**
- Last 30 readings → **slowly rising**

📦 This simulates a real-world case where your system heats up or degrades over time.

Step 2: Smooth the Sensor Data

python
CopyEdit
```python
smoothed = savgol_filter(sensor_data, window_length=11, polyorder=2)
```

Real sensors are noisy, so we **smooth the curve** to focus on the **actual trend**, not random jumps.

🎯 Smoothing makes slope detection more accurate.

Step 3: Detect a Stable "Baseline" Window Automatically

python
CopyEdit
```python
baseline_start, baseline_end = find_stable_baseline(smoothed)
```

Here, the algorithm **scans the smoothed data** to find a window that is:

- Flat (slope ≈ 0)
- Low in noise (low variance)

This window is assumed to be when the system was **behaving normally**.

💡 This is your **reference point** for comparison.

Step 4: Define the Current Window (Recent Data)

python
CopyEdit
```python
current_window = smoothed[-24:]  # last 2 hours if 5 min interval
```

We look at the last 24 readings (2 hours) to see **what's happening now**.

Step 5: Calculate Slope for Both Windows

python
CopyEdit
```python
baseline_slope = (baseline_window[-1] - baseline_window[0]) /
window_size
current_slope = (current_window[-1] - current_window[0]) / window_size
```

- The **baseline slope** shows how fast the system was changing when it was healthy.
- The **current slope** shows how fast it's changing now.

📈 This tells us how much the system behavior has shifted.

Step 6: Define Dynamic Thresholds

python
CopyEdit

```
warning_threshold = 1.5 × baseline_slope
critical_threshold = 2.5 × baseline_slope
```

The algorithm **learns** what's "too fast" by multiplying the baseline slope.

✅ No hardcoded values — this adapts to each system's own behavior.

Step 7: Compare and Classify the System

python
CopyEdit

```
if current_slope > critical_threshold:
    final_state = "Critical"
elif current_slope > warning_threshold:
    final_state = "Warning"
else:
    final_state = "Normal"
```

- If the system is heating up **a lot faster** than it used to → 🔥 Critical
- If it's warming **a bit faster** than before → ⚠️ Warning
- If it's behaving similarly → ✅ Normal

Imagine this:

You're observing a **machine's temperature sensor**.

You're not just asking:

> ❌ "Is the temperature high?"

You're asking:

> ✅ "Is the machine heating up **faster** than it usually does when it's okay?"

That's the **core intuition**

> **You're comparing how fast values are changing now to how fast they used to change when things were normal.**

It's like:

- **You know how calmly your system normally behaves (baseline slope).**
- **You notice that it's now speeding up (current slope).**
- **If the current behavior is too different — you raise an alert.**

💬 Summary: What Your Algorithm Is Thinking

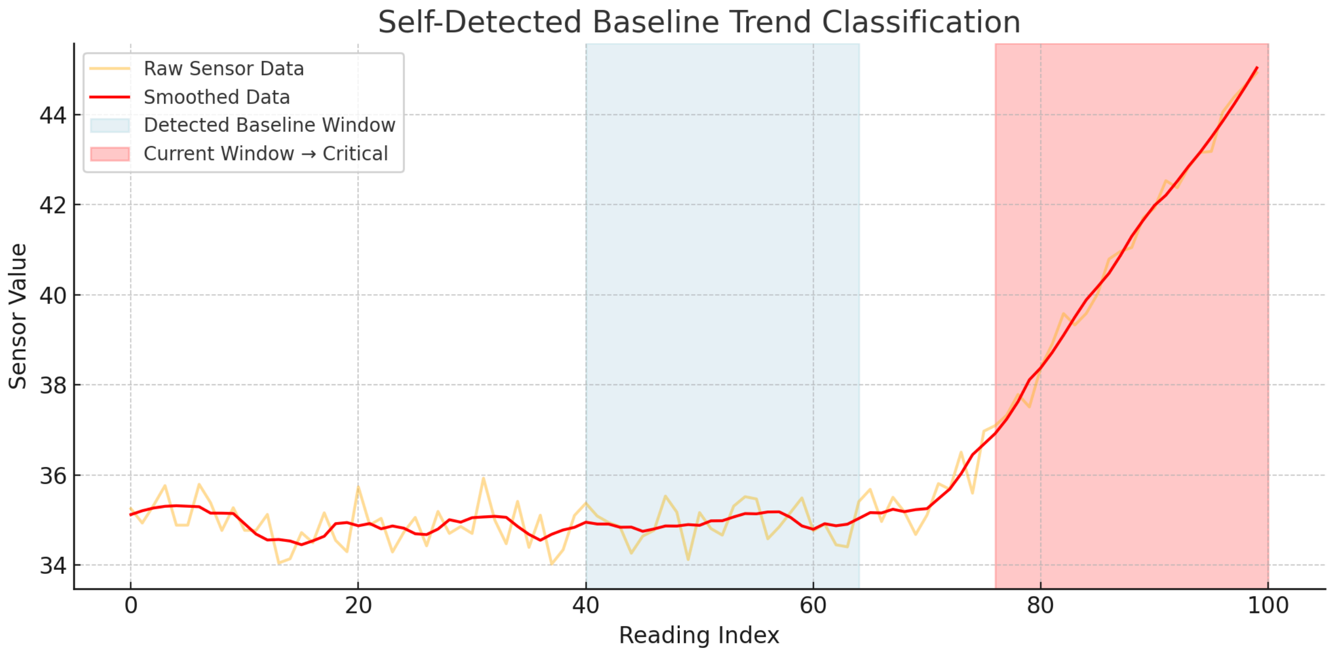
> **"I remember how the system behaves when it's calm.**
> **Now I see it's speeding up more than usual.**
> **This could be trouble — better raise a warning or alert!"**

That's intelligent behavior — not just reacting to fixed values.

logic

```
START
  |
  v
[1] Get sensor data (e.g., 100 readings)
  |
  v
[2] Smooth the data using Savitzky-Golay filter
    - Removes noise
```

```
    - Reveals the true trend
  |
  v
[3] Find a stable "baseline window"
    - Slide a fixed-size window over data
    - In each window:
        • Calculate slope = (last - first) / window_size
        • Calculate variance of the values
    - Select window with:
        • Slope ≈ 0 (almost flat)
        • Lowest variance (least noisy)
  |
  v
[4] Extract the "current window" (latest N readings)
    - This is what we want to classify
  |
  v
[5] Compute slope in both windows:
    - Baseline Slope = (last - first) / size
    - Current Slope = (last - first) / size
  |
  v
[6] Compute thresholds from baseline:
    - Warning Threshold  = 1.5 × baseline_slope
    - Critical Threshold = 2.5 × baseline_slope
  |
  v
[7] Compare current slope:
    IF current_slope > critical_threshold:
        → System State = "Critical"
    ELSE IF current_slope > warning_threshold:
        → System State = "Warning"
    ELSE:
        → System State = "Normal"
  |
  v
[8] Output:
    - Print/return system state
    - Optional: Plot smoothed curve
      with baseline and current window highlights
  |
  v
END
```


Self-Detected Baseline Trend Classification
Raw Sensor Data
Smoothed Data
Detected Baseline Window
Current Window → Critical
Sensor Value
Reading Index
34
36
38
40
42
44
0
20
40
60
80
100