Analytics & Alert Logic

1. Main Objective of Analytics & Alert Logi c

The **main objective** of Analytics & Alert Logic is to:

> **Convert raw data into meaningful insights and timely alerts to enable fast, accurate decisions.**

In simple words:

- **Analytics** → Understand *what is happening and why*
- **Alert Logic** → Notify *when immediate action is required*

Key objectives:

- Detect abnormal conditions early (faults, overheating, failures)
- Reduce manual monitoring
- Enable proactive and predictive actions
- Improve system safety, reliability, and performance
- Support real-time decision making (Edge or Cloud)

---

2. Advantages of Analytics & Alert Logic

1. Early Fault Detection

- Identifies issues **before failure happens**
- Example: Battery temperature rising beyond safe limit

2. Real-Time Response

- Alerts trigger **immediate action**

- Useful in safety-critical systems (vehicles, IoT, industrial systems)

3. Reduced Downtime

- Prevents sudden breakdowns
- Enables planned maintenance instead of emergency repair

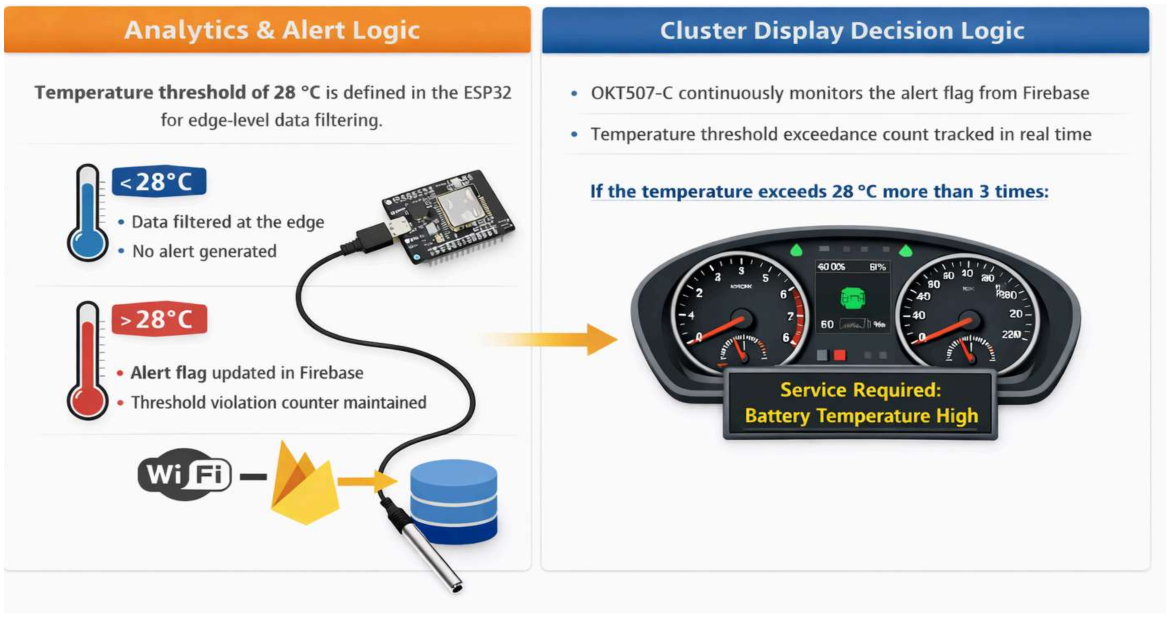
4. Improved Safety

- Alerts protect users, equipment, and environment
- Example: Over-voltage, over-temperature alerts

5. Better Decision Making

- Data-driven insights instead of assumptions
- Helps engineers, operators, and management

This is work flow


Analytics & Alert Logic
Temperature threshold of 28 °C is defined in the ESP32 for edge-level data filtering.
<28°C
Data filtered at the edge
No alert generated
>28°C
Alert flag updated in Firebase
Threshold violation counter maintained
WiFi
Cluster Display Decision Logic
OKT507-C continuously monitors the alert flag from Firebase
Temperature threshold exceedance count tracked in real time
If the temperature exceeds 28 °C more than 3 times:
Service Required:
Battery Temperature High

Step 1: Sensor Data Collection
First, a **temperature sensor connected to the ESP32** continuously measures the battery or ambient temperature.
These readings are generated in **real time** as raw sensor data.

Step 2: Threshold Defined at the Edge
Inside the ESP32, a **temperature threshold of 28 °C** is already defined.
This is important because it means the **decision logic runs locally on the device**, instead of sending all data to the cloud for processing.

Step 3: Data Filtering
Now, the ESP32 performs **analytics** by filtering the data:

- **If the temperature is below 28 °C**, the data is simply ignored.
  No alert is generated and no data is sent to the cloud.
  This helps reduce **network usage and cloud cost**.

- **If the temperature exceeds 28 °C**, the edge device detects a **threshold violation**.
  It generates an **alert flag**, increments a **violation counter**, and prepares only this important information for transmission.

So instead of sending continuous raw data, only **meaningful events** are selected.

Step 4: Data Sent to Cloud (Firebase)
Using **Wi-Fi**, the ESP32 sends only the alert-related data to the **Firebase Realtime Database**.
The cloud stores:

- Alert flag
- Current temperature
- Threshold exceed count

This ensures the cloud receives **filtered and relevant data only**.

Step 5: Continuous Monitoring by Cluster
The **instrument cluster (OKT507-C)** continuously monitors the alert flag and exceeds count from Firebase in real time.

Step 6: Decision Logic at Cluster Side
To avoid false alarms, the cluster applies its own logic:
If the temperature **exceeds 28 °C more than three times**, only then a service warning is triggered.
This prevents alerts caused by short or temporary temperature spikes.

Step 7: Driver Alert Display
Once the condition is met, the cluster displays the message:
“Service Required: Battery Temperature High.”
This informs the driver at the **right time**, helping prevent battery damage and improving vehicle safety.

Arduino Code

```
#include <Arduino.h>
#include <WiFi.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <Firebase_ESP_Client.h>

#include "addons/TokenHelper.h"
#include "addons/RTDBHelper.h"

/* ================== WiFi Credentials ================== */
#define WIFI_SSID       "iSprout-NRE"
#define WIFI_PASSWORD   "Isprout@n-202$"

/* ================== Firebase Credentials ================== */
#define API_KEY         "AIzaSyCR_FwvqqMGctW9i6MNn4ZUAGcuIjxPAqQ"
#define DATABASE_URL
"https://edge-data-filtering-default-rtdb.asia-southeast1.firebasedatabase.app/"
#define USER_EMAIL      "tejaswini.kopperla@votarytech.com"
#define USER_PASSWORD   "Votarytech@2025"

/* ================== Temperature Sensor ================== */
#define ONE_WIRE_BUS     15
#define TEMP_THRESHOLD  28.0

/* ================== Objects ================== */
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

FirebaseData fbdo;
```

```
FirebaseAuth auth;
FirebaseConfig config;

/* ================== Timing ================== */
unsigned long lastSend = 0;
const unsigned long interval = 5000;

/* ================== Analytics Tracking ================== */
int highTempCount = 0;
bool wasAboveThreshold = false;

/* ================== SETUP ================== */
void setup() {
 Serial.begin(115200);
 delay(1000);

 /* Temperature Sensor */
 sensors.begin();
 Serial.println("DS18B20 Initialized");

 /* WiFi */
 WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
 Serial.print("Connecting to WiFi");
 while (WiFi.status() != WL_CONNECTED) {
   delay(500);
   Serial.print(".");
 }
 Serial.println("\nWiFi Connected");
 Serial.print("IP Address: ");
 Serial.println(WiFi.localIP());

 /* Firebase */
 config.api_key = API_KEY;
 config.database_url = DATABASE_URL;

 auth.user.email = USER_EMAIL;
 auth.user.password = USER_PASSWORD;

 Firebase.begin(&config, &auth);
 Firebase.reconnectWiFi(true);

 Serial.println("Firebase Connected");
}

/* ================== LOOP ================== */
void loop() {
```

```
if (millis() - lastSend >= interval) {
  lastSend = millis();

  /* ===== Read Temperature ===== */
  sensors.requestTemperatures();
  float temperatureC = sensors.getTempCByIndex(0);

  if (temperatureC == DEVICE_DISCONNECTED_C) {
    Serial.println(" DS18B20 not detected");
    return;
  }

  Serial.print("Temperature: ");
  Serial.print(temperatureC);
  Serial.println(" °C");

  /* ===== EDGE ANALYTICS ===== */
  bool isAboveThreshold = (temperatureC > TEMP_THRESHOLD);

  // Rising-edge detection (Normal → High)
  if (isAboveThreshold && !wasAboveThreshold) {
    highTempCount++;
    Serial.print("⚠ High Temp Event Count: ");
    Serial.println(highTempCount);
  }

  wasAboveThreshold = isAboveThreshold;

  /* ===== Firebase Analytics ===== */
  Firebase.RTDB.setFloat(&fbdo, "/analytics/temperature/value", temperatureC);
  Firebase.RTDB.setInt(&fbdo, "/analytics/temperature/event_count", highTempCount);
  Firebase.RTDB.setBool(&fbdo, "/alerts/high_temp/status", isAboveThreshold);

  /* ===== Alert Messages ===== */
  if (isAboveThreshold) {

    Firebase.RTDB.setFloat(&fbdo, "/alerts/high_temp/value", temperatureC);

    if (highTempCount >= 3) {
      Firebase.RTDB.setString(
        &fbdo,
        "/alerts/high_temp/msg",
        "Service Required: Battery Temperature High"
      );
    } else {
      Firebase.RTDB.setString(
        &fbdo,
        "/alerts/high_temp/msg",
        "Temperature crossed 28C"
      );
    }

  } else {
    Firebase.RTDB.setString(
      &fbdo,
      "/alerts/high_temp/msg",
      "Temperature Normal"
    );
  }
 }
}
```

Conclusion: Analytics & Alert Logic

Analytics & Alert Logic plays a **critical role in turning raw sensor data into actionable insights and timely alerts**, enabling faster and smarter decision-making. By processing data at the edge and applying intelligent thresholds, it **reduces unnecessary data transfer, avoids false alarms, and ensures only meaningful events are communicated**.

The workflow—from **sensor data collection → edge threshold checking → filtered cloud transmission → cluster-side decision → driver alert**—demonstrates a **real-time, efficient, and scalable system** that improves safety, reliability, and performance.

Implementing this logic using platforms like **ESP32 with Firebase** shows how **IoT edge devices can handle analytics locally**, while the cloud supports monitoring and alert management. Overall, Analytics & Alert Logic **enables proactive maintenance, prevents failures, and enhances user safety**, making it an indispensable component in modern connected systems such as vehicles and industrial IoT.