IoT Terintegrasi dengan Machine Learning & Data Analytics

Dari Sensor ke Keputusan Cerdas dengan Python dan Open Source

Onno W. Purbo
onno@indo.net.id
onno@itts.ac.id
Twitter onnowpurbo

Institut Teknologi Tangerang Selatan (ITTS)
OnnoCenter
2026

Hak Cipta

Karya ini dilisensikan di bawah lisensi **Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0).** Artinya, siapapun bebas untuk menggunakan, menyalin, membagikan, dan mengadaptasi materi ini, dengan syarat:

- **Memberikan atribusi yang sesuai** (menyebut sumber asli),
- **Menyertakan lisensi yang sama** jika dimodifikasi atau dikembangkan lebih lanjut.

Lihat detail lisensi di sini https://creativecommons.org/licenses/by-nd/4.0/legalcode.id

Disclaimer

Buku ini disusun untuk tujuan edukasi, pembelajaran, dan pengembangan kapasitas pembaca dalam bidang *Internet of Things*, Python, *Machine Learning*, dan *Data Analytics*. Seluruh materi, contoh, ilustrasi, kode, konfigurasi, dan studi kasus disajikan sebagai referensi praktis yang perlu diuji, disesuaikan, dan divalidasi kembali sebelum diterapkan pada lingkungan nyata, terutama pada sistem produksi, industri, keselamatan, atau layanan publik. Penulis tidak bertanggung jawab atas kerusakan, kehilangan data, gangguan operasional, maupun dampak lain yang timbul dari penggunaan langsung materi dalam buku ini tanpa pengujian, pengamanan, dan penyesuaian yang memadai. Pembaca tetap bertanggung jawab untuk memperhatikan aspek keamanan, privasi, regulasi, etika, dan keandalan sistem dalam setiap implementasi.

Kata Pengantar

Alhamdulillah, puji syukur ke hadirat Allah SWT, karena atas rahmat-Nya buku ini dapat disusun dan dibagikan kepada pembaca. Buku ini lahir dari semangat bahwa **IoT, Python, Machine Learning, dan Data Analytics** bukan teknologi yang jauh dari kita, tetapi ilmu yang bisa dipelajari, dicoba, dan dimanfaatkan untuk memecahkan masalah nyata di sekitar kita.

Terima kasih kepada teman-teman komunitas, mahasiswa, dosen, guru, praktisi, dan semua pegiat *open source* yang terus menjaga semangat belajar dan berbagi ilmu di Indonesia. Dari rasa ingin tahu, keberanian mencoba, dan kebiasaan berbagi, sering lahir inovasi yang sederhana tetapi berdampak besar.

Buku ini disusun dengan pendekatan yang praktis dan rinci, khususnya dalam penggunaan Python untuk IoT, agar pembaca tidak hanya memahami konsep, tetapi juga mampu membangun solusi nyata berbasis data. Harapannya, buku ini dapat menjadi teman belajar yang membumi, menyenangkan, dan mendorong lahirnya lebih banyak karya teknologi anak bangsa.

Semoga buku ini bermanfaat, menginspirasi, dan menjadi bagian kecil dari ikhtiar bersama untuk **memajukan Indonesia melalui penguasaan teknologi yang terbuka, cerdas, dan mandiri**.

Jakarta, Maret 2026
Onno W. Purbo

Daftar Isi

Hak Cipta **2**
Disclaimer 2
Kata Pengantar **3**
Daftar Isi **4**
Bagian I — Fondasi Praktis IoT Analytics **35**
Bab 1. IoT, Data, dan Machine Learning: Gambaran Besar 35
1.2 Apa itu IoT? 35
1.3 Apa itu IIoT? 36
1.4 Apa itu data-driven decision making? 37
1.5 Alur Ujung ke Ujung: Sensor → Gateway → Broker → Storage → Analytics → Model → Aksi 38
1.5.1 Sensor 38
1.5.2 Gateway 38
1.5.3 Broker 39
1.5.4 Storage 39
1.5.5 Analytics 39
1.5.6 Model 40
1.5.7 Aksi 40
1.6 Mengapa Machine Learning di IoT Berbeda? 40
1.6.1 Data bersifat streaming, bukan diam 41
1.6.2 Data sensor sering berisik (noise) 41
1.6.3 Perangkat sering terbatas 41
1.6.4 Inferensi sering harus cepat 42
1.6.5 Konteks sangat penting 42
1.7 Open-Source Stack untuk Buku Ini 42
Stack yang digunakan 43
Kenapa stack ini kuat untuk belajar? 43
1.8 Peta Belajar Bab 1 untuk Pembaca 44
1.9 Langkah Praktis yang Harus Dilakukan Pembaca Setelah Mempelajari Bab Ini. 44
Tahap 1 — Pahami alur sistem 44
Tahap 2 — Tentukan contoh kasus kecil 44
Tahap 3 — Siapkan lingkungan open source 45
Tahap 4 — Simulasikan data pertama 45
Tahap 5 — Melatih cara berpikir berbasis keputusan 45
1.10 Penutup Bab 46
Hands-on 1 - Menyiapkan Lingkungan Kerja Python untuk IoT Analytics di Ubuntu 26.04. 47
1. Tujuan Hands-on 47
2. Contoh use case nyata yang relevan 47
Contoh 1 — Monitoring suhu ruang server kampus 47
Contoh 2 — Analitik konsumsi energi laboratorium 48
Contoh 3 — Prototipe predictive maintenance sederhana 48
3. Prinsip kerja yang dipakai di buku ini 48

4. Langkah instalasi di Ubuntu 26.04 48
4.1 Update sistem terlebih dahulu 48
4.2 Pastikan Python dan tool dasar tersedia 48
4.3 Buat folder kerja utama 49
4.4 Buat virtual environment 49
4.5 Upgrade pip dan pasang pustaka inti 49
4.6 Simpan daftar paket 50
4.7 Jalankan JupyterLab 50
5. Struktur folder proyek yang disarankan 51
6. Uji lingkungan dengan skrip pertama 51
7. Notebook pertama untuk uji kerja 52
8. Langkah kerja ringkas yang harus dilakukan pembaca 52
9. Masalah umum dan solusinya 53
10. Penutup Hands-on 1 53
Hands-on 2 - Membaca Data Sensor Dummy CSV dan Membuat Visualisasi Pertama. 54
1. Tujuan Hands-on 54
2. Contoh nyata yang bisa diimplementasikan 54
Contoh 1 — Smart Room Monitoring 54
Contoh 2 — Smart Greenhouse / urban farming 54
Contoh 3 — Energi dan peralatan listrik 54
3. Dataset dummy yang akan dipakai 55
4. Buat file CSV dummy 55
5. Membaca CSV dengan Python 56
6. Mengubah timestamp menjadi format waktu 57
7. Menjadikan timestamp sebagai index 57
8. Visualisasi pertama dengan matplotlib 58
9. Visualisasi multi-variabel 59
10. Visualisasi relasi antar-variabel 60
11. Membuat ringkasan otomatis 61
12. Versi notebook yang cocok untuk mahasiswa 61
13. Langkah-langkah kerja yang harus dilakukan pembaca 62
Tahap 1 — Siapkan file data 62
Tahap 2 — Baca data dengan pandas 62
Tahap 3 — Ubah kolom waktu 62
Tahap 4 — Jadikan data sebagai time series 62
Tahap 5 — Buat grafik pertama 62
Tahap 6 — Tambahkan variabel lain 62
Tahap 7 — Tarik insight awal 62
14. Insight nyata yang bisa ditindaklanjuti 63
15. Variasi latihan lanjutan yang bisa dicoba 63
16. Penutup Hands-on 2 64
Bab 2. Arsitektur IoT Open Source untuk Praktikum 65
2.1 Mengapa arsitektur IoT perlu dipahami sejak awal? 65
2.2 Komponen inti: sensor, mikrokontroler, gateway, broker, database, dashboard .. 65

2.2.1 Sensor..... 65
2.2.2 Mikrokontroler..... 66
2.2.3 Gateway..... 66
2.2.4 Broker..... 66
2.2.5 Database..... 67
2.2.6 Dashboard..... 67
2.3 Protokol yang cukup untuk buku: MQTT, HTTP REST, serial..... 68
MQTT..... 68
HTTP REST..... 68
Serial..... 68
2.4 Peran edge, fog, dan cloud secara praktis..... 69
Edge..... 69
Fog..... 69
Cloud..... 70
2.5 Arsitektur minimalis berbasis open source..... 70
Versi 1 — Paling sederhana..... 70
Versi 2 — Praktikum kampus/lab..... 71
Versi 3 — Menuju sistem cerdas..... 71
2.6 Langkah berpikir yang harus dilakukan pembaca..... 71
2.7 Rekomendasi praktis untuk praktikum Ubuntu..... 72
2.8 Penutup bab..... 72
Hands-on 3 - Mengirim Data Sensor ke MQTT..... 73
1. Tujuan Hands-on..... 73
2. Contoh implementasi nyata yang relevan..... 73
Contoh 1 — Monitoring suhu ruang server..... 73
Contoh 2 — Greenhouse / urban farming..... 73
Contoh 3 — Monitoring konsumsi energi laboratorium..... 73
3. Arsitektur yang dipakai..... 74
4. Instalasi Mosquitto di Ubuntu 26.04..... 74
5. Siapkan pustaka Python..... 74
6. Struktur folder yang disarankan..... 75
7. Topik MQTT yang akan dipakai..... 75
8. Format data yang dikirim..... 75
9. Source code utama: publisher / simulator sensor Python..... 76
10. Variasi realistis: publisher untuk sensor greenhouse..... 77
11. Langkah kerja yang harus dilakukan pembaca..... 78
12. Tantangan kecil yang menantang..... 78
13. Penutup Hands-on 3..... 78
Hands-on 4 - Subscribe Data MQTT dengan Python..... 79
1. Tujuan Hands-on..... 79
2. Mengapa subscriber Python penting?..... 79
3. Source code utama: subscriber Python sederhana..... 80
4. Subscriber wildcard untuk banyak perangkat..... 81
5. Validasi data di subscriber..... 82

6. Contoh implementasi real yang bisa dipakai....83
Kasus ruang kelas....83
Kasus bengkel mesin....83
Kasus smart water....83
7. Langkah kerja yang harus dilakukan pembaca....83
8. Penutup Hands-on 4....83
Hands-on 5 - Simpan Data ke CSV / SQLite / InfluxDB....84
1. Tujuan Hands-on....84
2. Kapan memakai CSV, SQLite, dan InfluxDB?....84
CSV....84
SQLite....84
InfluxDB....84
3. Simpan data MQTT ke CSV....85
4. Simpan data MQTT ke SQLite....86
5. Simpan data MQTT ke InfluxDB....88
5.1 Menjalankan InfluxDB dengan Docker di Ubuntu 26.04....88
5.2 Pasang pustaka Python untuk InfluxDB....89
5.3 Simpan data ke InfluxDB dengan Python....89
6. Contoh implementasi real untuk penyimpanan....91
Ruang server....91
Greenhouse....91
Energi gedung....91
7. Pola satu subscriber untuk banyak tujuan....91
8. Versi gabungan: simpan ke CSV + SQLite sekaligus....91
9. Langkah kerja yang harus dilakukan pembaca....93
10. Tantangan latihan lanjutan....94
11. Insight praktis untuk mahasiswa dan praktisi....94
12. Penutup Hands-on 5....94
Bab 3. Memahami Data Sensor IoT....95
3.1 Mengapa memahami data sensor lebih penting daripada langsung memilih model?....95
3.2 Apa yang dimaksud dengan data sensor IoT?....95
3.3 Jenis data IoT yang paling relevan untuk buku ini....96
3.3.1 Data suhu....96
3.3.2 Data kelembaban....96
3.3.3 Data arus atau daya listrik....97
3.3.4 Data getaran....97
3.3.5 Data GPS atau lokasi....97
3.3.6 Data konsumsi energi....98
3.4 Karakteristik data time series pada IoT....99
Ciri utama time series IoT....99
Mengapa urutan waktu penting?....99
Time series IoT sering multivariat....99
Time series IoT sering tidak stasioner....100

3.5 Masalah umum pada data sensor IoT....100
3.5.1 Missing value atau data hilang....100
3.5.2 Outlier atau nilai menyimpang....101
3.5.3 Drift atau perubahan pola dari waktu ke waktu....102
3.5.4 Interval waktu yang tidak konsisten....103
3.5.5 Label yang jarang atau tidak seimbang....104
3.6 Hubungan antara jenis data dan masalah kualitas....105
3.7 Contoh-contoh real yang bisa diimplementasikan....106
3.8 Langkah kerja yang harus dilakukan pembaca saat menerima data sensor....107
3.9 Sikap profesional terhadap data sensor....108
3.10 Penutup bab....108
Hands-on 6 - Eksplorasi Dataset Sensor dengan pandas....109
1. Tujuan Hands-on....109
2. Contoh implementasi nyata yang relevan....109
3. Dataset yang akan dipakai....110
4. Persiapan lingkungan Ubuntu 26.04....111
5. Source code utama: eksplorasi dataset dengan pandas....111
6. Konversi timestamp dan cek rentang waktu....112
7. Menjadikan data sebagai time series....112
8. Eksplorasi statistik yang lebih berguna....113
9. Eksplorasi korelasi awal....114
10. Visualisasi eksploratif pertama....115
11. Langkah kerja yang harus dilakukan pembaca....116
12. Penutup Hands-on 6....116
Hands-on 7 - Membersihkan Timestamp, Resampling, dan Imputasi Sederhana....117
1. Tujuan Hands-on....117
2. Contoh nyata yang bisa diimplementasikan....117
3. Dataset kotor untuk latihan....118
4. Source code utama: membersihkan waktu....119
5. Set index waktu dan cek celah data....120
6. Resampling ke interval 10 menit....120
7. Imputasi sederhana untuk nilai hilang....121
7.1 Forward fill....121
7.2 Interpolasi linier....121
8. Simpan hasil data bersih....122
9. Versi lengkap skrip....122
10. Langkah kerja yang harus dilakukan pembaca....123
11. Penutup Hands-on 7....123
Hands-on 8 - Deteksi Anomali Dasar dengan Aturan Statistik....124
1. Tujuan Hands-on....124
2. Contoh nyata yang bisa diimplementasikan....124
3. Dataset untuk latihan anomali....125
4. Metode 1 — Aturan domain sederhana....126
5. Metode 2 — Z-score....127

6. Metode 3 — IQR..... 128
7. Visualisasi anomali.....129
8. Versi gabungan: satu skrip lengkap..... 130
9. Cara membaca hasil..... 131
10. Langkah kerja yang harus dilakukan pembaca.....131
11. Contoh interpretasi real..... 132
12. Penutup Hands-on 8..... 132
Bab 4. Membangun Dummy Data dan Synthetic Data untuk IoT.....133
4.1 Mengapa dummy data penting untuk pembelajaran, pengujian pipeline, dan prototyping?.....133
4.2 Perbedaan: dummy data, simulated data, synthetic data, dan augmented data..... 134
4.2.1 Dummy data..... 134
4.2.2 Simulated data.....134
4.2.3 Synthetic data.....135
4.2.4 Augmented data..... 135
4.3 Mengapa Bab 4 sangat penting dalam alur buku ini?..... 136
4.4 Sumber pembentukan data sintetik.....136
4.4.1 Aturan fisik sederhana.....136
4.4.2 Distribusi statistik.....137
4.4.3 Pola periodik.....137
4.4.4 Skenario alarm atau fault.....137
4.5 Membangun data IoT sintetis dengan Python.....138
4.5.1 Contoh 1 — Suhu dan kelembaban berbasis pola harian.....138
4.5.2 Contoh 2 — Konsumsi energi dengan pola jam sibuk.....139
4.5.3 Contoh 3 — Getaran mesin normal vs fault..... 141
4.5.4 Menambahkan noise pada data sensor..... 142
4.5.5 Menambahkan missing value, drift, dan outlier buatan..... 142
4.5.6 Menambahkan label sintetis.....143
Contoh 1 — Label nyaman / tidak nyaman..... 144
Contoh 2 — Label normal / warning / fault..... 144
Contoh 3 — Label hemat / boros.....145
4.6 Contoh-contoh real yang bisa diimplementasikan.....145
4.6.1 Smart room monitoring.....145
4.6.2 Smart energy analytics.....146
4.6.3 Predictive maintenance sederhana..... 146
4.6.4 Greenhouse atau pertanian presisi..... 146
4.7 Validasi sederhana: kapan data sintetik “cukup realistis”?.....147
4.8 Risiko dan batasan: data sintetik tidak boleh dianggap pengganti penuh data lapangan.....148
4.9 Langkah kerja yang harus dilakukan pembaca.....149
Tahap 1 — Pilih use case.....149
Tahap 2 — Tentukan variabel inti..... 149
Tahap 3 — Tentukan pola dasar.....149
Tahap 4 — Tambahkan noise dan variasi.....149

Tahap 5 — Tambahkan masalah realistis....149
Tahap 6 — Tambahkan label sintetis....149
Tahap 7 — Validasi cepat....150
Tahap 8 — Gunakan di bab berikutnya....150
4.10 Penutup bab....150
Hands-on 9 - Generate Data Suhu–kelembaban Ruangan 30 Hari....151
1. Tujuan hands-on....151
2. Contoh implementasi nyata....151
3. Persiapan Ubuntu 26.04....151
4. Source code utama....152
5. Visualisasi cepat....153
6. Hal yang harus diamati pembaca....153
Hands-on 10 - Generate Data Konsumsi Energi Berbasis Jam Kerja....154
1. Tujuan hands-on....154
2. Contoh implementasi nyata....154
3. Logika sintetis yang dipakai....154
4. Source code utama....155
5. Tambahkan label hemat / boros....156
6. Insight yang bisa diambil....156
Hands-on 11 - Generate Data Getaran Mesin dengan Fault Injection....157
1. Tujuan hands-on....157
2. Contoh implementasi nyata....157
3. Ide fault injection....157
4. Source code utama....158
5. Visualisasi fault....159
6. Apa yang harus diamati....159
Hands-on 12 - Membuat Dataset Sintetis untuk Klasifikasi dan Anomaly Detection....160
1. Tujuan hands-on....160
2. Skenario yang akan dibuat....160
3. Source code utama....161
4. Mengapa dataset ini berguna?....162
5. Cek keseimbangan label....163
Hands-on 13 - Ekspor Dataset ke CSV dan Publish ke MQTT Simulator....164
1. Tujuan hands-on....164
2. Persiapan Ubuntu 26.04....164
3. Uji broker MQTT....164
4. Source code utama: publish CSV ke MQTT....165
5. Subscriber Python untuk memverifikasi....166
6. Variasi realistis yang bisa dicoba....166
7. Versi publish multi-topic....167
8. Mengapa langkah ini sangat penting?....167
Ringkasan langkah kerja yang harus dilakukan pembaca....168
Tahap 1 — Bangun data sintetik....168
Tahap 2 — Tambahkan label....168

Tahap 3 — Validasi cepat....168
Tahap 4 — Simpan ke CSV....168
Tahap 5 — Publish ke MQTT....168
Tahap 6 — Verifikasi dengan subscriber....168
Tahap 7 — Siapkan untuk bab berikutnya....169
Penutup....169
Bagian II — Python Data Analytics untuk IoT (Bab 5–7)....170
Bab 5. Feature Engineering untuk Data Sensor....170
5.1 Mengapa feature engineering penting dalam IoT?....170
5.2 Apa yang dimaksud feature engineering?....170
5.3 Rolling window, lag features, moving average, difference....171
5.3.1 Rolling window....171
5.3.2 Lag features....171
5.3.3 Moving average....171
5.3.4 Difference....172
5.4 Statistik per jendela: mean, std, min, max....172
Mean....172
Standard deviation....172
Minimum dan maksimum....172
5.5 Encoding waktu: jam, hari, pola musiman....173
Jam....173
Hari....173
Pola musiman....173
5.6 Sensor fusion sederhana....174
5.7 PCA....174
5.8 Contoh-contoh real yang bisa diimplementasikan....175
Smart room monitoring....175
Smart energy analytics....175
Predictive maintenance sederhana....176
5.9 Langkah kerja yang harus dilakukan pembaca....177
Tahap 1 — Pastikan data sudah bersih....177
Tahap 2 — Tentukan tujuan....177
Tahap 3 — Buat fitur waktu....177
Tahap 4 — Buat fitur historis....177
Tahap 5 — Gabungkan beberapa sensor....177
Tahap 6 — Ringkas bila perlu....177
Tahap 7 — Validasi sederhana....178
5.10 Sikap profesional dalam feature engineering....178
5.11 Penutup bab....178
Hands-on 14 - Membuat Fitur Time-Series dari Data Suhu/kelembaban....179
1. Tujuan hands-on....179
2. Contoh implementasi nyata....179
3. Dataset yang dipakai....180
4. Persiapan Ubuntu 26.04....180

5. Source code utama: membangun fitur time-series....180
6. Penjelasan fitur yang dibuat....182
Lag features....182
Difference....182
Moving average....182
Rolling statistics....182
Encoding waktu....182
Sensor fusion sederhana....182
7. Visualisasi fitur....183
8. Hasil yang harus diamati....183
Hands-on 15 - Fitur Getaran Sederhana untuk Mendeteksi Kondisi Mesin....184
1. Tujuan hands-on....184
2. Contoh implementasi nyata....184
3. Dataset yang dipakai....184
4. Fitur sederhana yang akan dibuat....185
5. Source code utama....185
6. Mengapa fitur-fitur ini berguna?....187
Mean....187
Standard deviation....187
RMS....187
Peak-to-peak dan range....187
Kurtosis....187
7. Visualisasi fitur....187
8. Hal yang harus diamati....188
Hands-on 16 - PCA untuk Merangkum Beberapa Sensor....189
1. Tujuan hands-on....189
2. Contoh implementasi nyata....189
3. Dataset yang dipakai....189
4. Persiapan....189
5. Source code utama....190
6. Mengapa perlu standardisasi?....191
7. Visualisasi PCA 2 dimensi....191
8. Interpretasi hasil PCA....192
9. Contoh perluasan realistis....192
Smart room....192
Smart energy....192
Multi-sensor fusion....192
Langkah kerja yang harus dilakukan pembaca....193
Tahap 1 — Pastikan data bersih....193
Tahap 2 — Buat fitur time-series....193
Tahap 3 — Buat fitur sinyal getaran....193
Tahap 4 — Gabungkan dan pilih fitur....193
Tahap 5 — Standarisasi....193
Tahap 6 — Jalankan PCA....193

Tahap 7 — Evaluasi hasil..... 193
Penutup..... 194
Bab 6. Visualisasi, Dashboard, dan Analitik Real-Time..... 195
6.1 Mengapa visualisasi sangat penting dalam IoT?..... 195
6.2 Posisi Bab 6 dalam alur buku..... 195
6.3 Visualisasi eksploratif dengan matplotlib / plotly ringan..... 196
6.3.1 Mengapa memilih matplotlib dan plotly ringan?..... 196
6.3.2 Fungsi visualisasi eksploratif..... 196
6.3.3 Bentuk visualisasi yang paling relevan untuk IoT..... 196
6.3.4 Prinsip visualisasi yang baik..... 197
6.4 Analitik deskriptif dan diagnostik untuk IoT..... 197
6.4.1 Analitik deskriptif..... 197
6.4.2 Analitik diagnostik..... 197
6.4.3 Perbedaan praktis bagi pembaca..... 198
6.5 Konsep streaming vs batch..... 198
6.5.1 Mengapa pembaca IoT harus memahami keduanya?..... 198
6.5.2 Batch processing..... 198
6.5.3 Streaming processing..... 199
6.5.4 Kapan memilih batch, kapan memilih streaming?..... 199
6.5.5 Dalam sistem nyata, keduanya dipakai bersama..... 200
6.6 Dashboard open source dengan Grafana..... 200
6.6.1 Mengapa dashboard dibutuhkan?..... 200
6.6.2 Mengapa Grafana?..... 200
6.6.3 Fungsi dashboard dalam IoT..... 201
6.6.4 Contoh panel dashboard yang relevan..... 201
6.6.5 Prinsip desain dashboard yang baik..... 201
6.6.6 Grafana dalam arsitektur buku ini..... 202
6.7 Contoh-contoh nyata yang bisa diimplementasikan..... 202
Smart room monitoring..... 202
Smart energy analytics..... 202
Predictive maintenance sederhana..... 203
6.8 Langkah kerja yang harus dilakukan pembaca..... 203
Tahap 1 — Pilih data yang akan divisualisasikan..... 203
Tahap 2 — Mulai dari visualisasi eksploratif sederhana..... 203
Tahap 3 — Tarik insight deskriptif..... 203
Tahap 4 — Lakukan analitik diagnostik ringan..... 204
Tahap 5 — Bedakan kebutuhan batch dan streaming..... 204
Tahap 6 — Susun kebutuhan dashboard..... 204
Tahap 7 — Integrasikan dengan Grafana..... 204
6.9 Sikap profesional dalam visualisasi dan dashboard..... 204
6.10 Penutup bab..... 205
Bab 7. Membangun Dataset Siap Machine Learning..... 206
7.1 Mengapa “dataset siap ML” berbeda dari sekadar data bersih?..... 206
7.2 Posisi Bab 7 dalam alur buku..... 206

7.3 Menentukan target..... 207
7.3.1 Klasifikasi status..... 207
7.3.2 Regresi prediksi nilai sensor..... 208
7.3.3 Deteksi anomali..... 209
7.4 Memilih target yang tepat..... 210
7.5 Split data time-aware untuk IoT..... 210
7.6 Validasi yang benar untuk time series..... 211
7.7 Metrik evaluasi..... 212
7.7.1 MAE / RMSE untuk regresi..... 212
MAE..... 212
RMSE..... 213
7.7.2 Accuracy / Precision / Recall / F1 untuk klasifikasi..... 213
Accuracy..... 213
Precision..... 214
Recall..... 214
F1 score..... 214
7.7.3 Confusion matrix..... 214
7.8 Menggabungkan data nyata dan data sintetik..... 215
7.9 Kapan synthetic data membantu?..... 216
1. Data nyata terlalu sedikit..... 216
2. Label kejadian penting terlalu jarang..... 216
3. Ingin menguji pipeline..... 216
4. Ingin membangun baseline cepat..... 216
5. Ingin menguji skenario yang jarang terjadi..... 217
7.10 Kapan synthetic data bisa menyesatkan?..... 217
1. Jika data sintetik terlalu “rapi”..... 217
2. Jika aturan sintetik terlalu sederhana..... 217
3. Jika proporsi sintetik terlalu dominan..... 217
4. Jika distribusi sintetik tidak cocok dengan lapangan..... 217
5. Jika data sintetik dipakai untuk evaluasi akhir..... 218
7.11 Strategi praktis yang disarankan..... 218
Gunakan data nyata sebagai referensi distribusi..... 218
Pisahkan peran data..... 218
Dokumentasikan sumber label..... 218
Uji model pada data nyata terbaru..... 218
7.12 Contoh-contoh nyata yang bisa diimplementasikan..... 218
Smart room monitoring..... 218
Smart energy analytics..... 219
Predictive maintenance sederhana..... 219
7.13 Langkah kerja yang harus dilakukan pembaca..... 220
Tahap 1 — Tentukan tujuan model..... 220
Tahap 2 — Tentukan target secara eksplisit..... 220
Tahap 3 — Pastikan fitur dan target selaras waktu..... 220
Tahap 4 — Lakukan split berbasis waktu..... 220

Tahap 5 — Pilih metrik yang tepat....220
Tahap 6 — Evaluasi risiko ketidakseimbangan label....221
Tahap 7 — Pertimbangkan data sintetik secara hati-hati....221
Tahap 8 — Simpan versi dataset final....221
7.14 Sikap profesional dalam membangun dataset ML untuk IoT....221
7.15 Penutup bab....222
Hands-on 20 - Menyiapkan Dataset Klasifikasi Normal vs Abnormal....223
1. Tujuan hands-on....223
2. Contoh implementasi nyata....223
Smart room monitoring....223
Smart energy....223
Predictive maintenance sederhana....224
3. Dataset yang akan dipakai....224
4. Persiapan Ubuntu 26.04....224
5. Strategi yang dipakai....225
6. Source code utama....225
7. Hal yang harus diperiksa pembaca....227
8. Insight penting....227
Hands-on 21 - Menyiapkan Dataset Regresi Prediksi Konsumsi Energi....228
1. Tujuan hands-on....228
2. Contoh implementasi nyata....228
Smart energy gedung....228
Laboratorium komputer....228
Rumah pintar / UMKM....229
3. Dataset yang akan dipakai....229
4. Strategi....229
5. Source code utama....229
6. Hal penting yang harus dipahami....231
Mengapa target digeser ke atas dengan shift(-1)?....231
Mengapa tidak boleh sembarang shuffle?....231
Mengapa ada lag dan rolling?....231
Hands-on 22 - Evaluasi Baseline Model pada Data Nyata vs Sintetik....232
1. Tujuan hands-on....232
2. Strategi baseline yang dipakai....232
3. Evaluasi baseline klasifikasi....233
4. Evaluasi baseline regresi....234
5. Membandingkan data nyata vs sintetik....235
Tujuan....235
Skenario praktis....235
Pendekatan....236
Pola A — Train sintetik, test sintetik....236
Pola B — Train sintetik + nyata, test nyata....236
Pola C — Train nyata, test nyata....236
6. Source code contoh perbandingan sederhana....236

7. Kapan synthetic data membantu, dan kapan menyesatkan?.....237
8. Langkah kerja yang harus dilakukan pembaca.....238
Tahap 1 — Siapkan dataset klasifikasi.....238
Tahap 2 — Siapkan dataset regresi.....238
Tahap 3 — Latih baseline sederhana.....238
Tahap 4 — Evaluasi metrik dengan benar.....238
Tahap 5 — Bandingkan sintetik dan nyata.....238
Tahap 6 — Simpan hasil.....238
9. Penutup.....238
Bagian III — Machine Learning Hands-on untuk IoT (Bab 8–11).....239
Bab 8. Supervised Learning untuk IoT dengan scikit-learn.....239
8.1 Mengapa supervised learning penting untuk IoT?.....239
8.2 Mengapa scikit-learn?.....240
8.3 Apa yang dimaksud supervised learning?.....240
8.4 Regresi: Linear Regression.....241
8.4.1 Mengapa mulai dari Linear Regression?.....241
8.4.2 Intuisi sederhana.....242
8.4.3 Kelebihan Linear Regression.....242
8.4.4 Keterbatasan Linear Regression.....242
8.4.5 Contoh implementasi nyata.....242
8.4.6 Kapan Linear Regression cocok?.....243
8.5 Regresi: Random Forest Regressor.....243
8.5.1 Mengapa Random Forest Regressor penting?.....243
8.5.2 Intuisi sederhana.....243
8.5.3 Kelebihan Random Forest Regressor.....244
8.5.4 Keterbatasan.....244
8.5.5 Contoh implementasi nyata.....244
8.5.6 Kapan Random Forest Regressor cocok?.....244
8.6 Klasifikasi: Logistic Regression.....245
8.6.1 Mengapa Logistic Regression penting?.....245
8.6.2 Intuisi sederhana.....245
8.6.3 Kelebihan Logistic Regression.....245
8.6.4 Keterbatasan.....245
8.6.5 Contoh implementasi nyata.....246
8.6.6 Kapan Logistic Regression cocok?.....246
8.7 Klasifikasi: Random Forest Classifier.....246
8.7.1 Mengapa Random Forest Classifier sangat relevan untuk IoT?.....246
8.7.2 Intuisi sederhana.....246
8.7.3 Kelebihan Random Forest Classifier.....247
8.7.4 Keterbatasan.....247
8.7.5 Contoh implementasi nyata.....247
8.7.6 Kapan Random Forest Classifier cocok?.....247
8.8 Gradient Boosting / XGBoost opsional singkat.....248
8.8.1 Mengapa dibahas singkat?.....248

8.8.2 Intuisi sederhana boosting....248
8.8.3 Kapan Gradient Boosting menarik?....248
8.8.4 Kapan tidak perlu memaksakan?....248
8.8.5 Catatan tentang XGBoost....248
8.9 Kapan memilih model sederhana dibanding model kompleks?....249
8.10 Contoh-contoh nyata yang bisa diimplementasikan....250
8.11 Langkah kerja yang harus dilakukan pembaca....251
Tahap 1 — Pastikan dataset sudah siap....251
Tahap 2 — Pilih jenis masalah....251
Tahap 3 — Mulai dari model paling sederhana....251
Tahap 4 — Tingkatkan bila perlu....251
Tahap 5 — Evaluasi dengan metrik yang sesuai....251
Tahap 6 — Bandingkan hasil dan interpretasi....251
Tahap 7 — Simpan model terbaik sebagai baseline operasional....252
8.12 Sikap profesional dalam supervised learning untuk IoT....252
8.13 Penutup bab....252
Hands-on 23 - Prediksi Suhu / Energi dengan Regresi....253
1. Tujuan hands-on....253
2. Contoh implementasi nyata....253
Prediksi suhu ruang....253
Prediksi energi....254
3. Dataset yang digunakan....254
4. Persiapan Ubuntu 26.04....254
5. Baseline pertama: Linear Regression....254
6. Model kedua: Random Forest Regressor....256
7. Visualisasi prediksi regresi....257
8. Insight praktis regresi....258
Hands-on 24 - Klasifikasi Status Perangkat (normal, warning, fault)....259
1. Tujuan hands-on....259
2. Contoh implementasi nyata....259
Smart room monitoring....259
Predictive maintenance....259
Smart energy....260
3. Dataset yang digunakan....260
4. Menyiapkan dataset multi-kelas....260
5. Logistic Regression untuk multi-kelas....262
6. Random Forest Classifier untuk multi-kelas....263
Hands-on 25 - Perbandingan 3 Model dengan Metrik dan Visualisasi Hasil....265
1. Tujuan hands-on....265
2. Model yang dibandingkan....265
3. Source code perbandingan model....266
4. Visualisasi hasil perbandingan....267
5. Variasi perbandingan untuk regresi....268
6. Cara membaca hasil perbandingan....269

Jika Logistic Regression menang....269
Jika Random Forest menang.... 269
Jika Gradient Boosting menang tipis....269
7. Kapan memilih model sederhana dibanding model kompleks?.... 270
8. Langkah kerja yang harus dilakukan pembaca....270
9. Penutup....271
Bab 9. Unsupervised Learning dan Anomaly Detection....272
9.1 Mengapa unsupervised learning penting untuk IoT?.... 272
9.2 Posisi Bab 9 dalam alur buku....273
9.3 Apa itu unsupervised learning?....273
9.4 Mengapa anomaly detection sangat relevan untuk IoT?....274
9.5 K-Means....274
9.5.1 Apa itu K-Means?....274
9.5.2 Intuisi sederhana....275
9.5.3 Kelebihan K-Means....275
9.5.4 Keterbatasan K-Means....275
9.5.5 Contoh implementasi nyata....275
9.5.6 Kapan K-Means cocok?.... 276
9.6 DBSCAN.... 276
9.6.1 Apa itu DBSCAN?.... 276
9.6.2 Intuisi sederhana.... 276
9.6.3 Kelebihan DBSCAN....276
9.6.4 Keterbatasan DBSCAN.... 277
9.6.5 Contoh implementasi nyata....277
9.6.6 Kapan DBSCAN cocok?....277
9.7 K-Means vs DBSCAN.... 277
9.8 Isolation Forest....278
9.8.1 Apa itu Isolation Forest?....278
9.8.2 Intuisi sederhana.... 278
9.8.3 Kelebihan Isolation Forest....278
9.8.4 Keterbatasan.... 278
9.8.5 Contoh implementasi nyata....278
9.8.6 Kapan Isolation Forest cocok?.... 279
9.9 One-Class SVM secara ringkas.... 279
9.9.1 Apa itu One-Class SVM?....279
9.9.2 Intuisi sederhana.... 279
9.9.3 Kelebihan....279
9.9.4 Keterbatasan.... 279
9.9.5 Kapan dipakai?....280
9.10 Threshold vs model ML....280
9.10.1 Pendekatan threshold....280
9.10.2 Pendekatan model ML....281
9.10.3 Kapan threshold cukup?....281
9.10.4 Kapan model ML lebih tepat?....282

9.10.5 Pendekatan terbaik di dunia nyata..... 282
9.11 Contoh-contoh nyata yang bisa diimplementasikan..... 282
Smart room monitoring..... 282
Smart energy analytics..... 282
Predictive maintenance sederhana..... 283
Keamanan / intrusion detection IoT..... 283
9.12 Langkah kerja yang harus dilakukan pembaca..... 283
Tahap 1 — Tentukan tujuan..... 283
Tahap 2 — Siapkan fitur yang masuk akal..... 283
Tahap 3 — Pilih metode awal..... 283
Tahap 4 — Validasi dengan visualisasi..... 283
Tahap 5 — Bandingkan dengan domain knowledge..... 284
Tahap 6 — Putuskan strategi operasional..... 284
9.13 Sikap profesional dalam unsupervised learning dan anomaly detection..... 284
9.14 Penutup bab..... 284
Hands-on 21 - Clustering Pola Ruangan dari Data Suhu–kelembaban..... 285
1. Tujuan hands-on..... 285
2. Contoh implementasi nyata..... 285
Smart room monitoring..... 285
Greenhouse kecil..... 285
Laboratorium komputer..... 285
3. Dataset yang dipakai..... 286
4. Persiapan Ubuntu 26.04..... 286
5. Mengapa perlu scaling?..... 286
6. Source code utama: K-Means untuk pola ruangan..... 286
7. Cara membaca hasil K-Means..... 287
8. Source code utama: DBSCAN untuk pola ruangan dan noise..... 288
9. Cara membaca hasil DBSCAN..... 289
10. Insight praktis Hands-on 21..... 289
Hands-on 22 - Deteksi Anomali Arus Listrik dengan Isolation Forest..... 290
1. Tujuan hands-on..... 290
2. Contoh implementasi nyata..... 290
Smart energy analytics..... 290
Laboratorium kampus..... 290
Bengkel / workshop..... 290
3. Dataset yang dipakai..... 290
Script generate dataset..... 291
4. Source code utama: Isolation Forest..... 292
5. Evaluasi sederhana terhadap ground truth sintetis..... 293
Hands-on 23 - Membandingkan Threshold Statis vs Model Anomaly Detection..... 294
1. Tujuan hands-on..... 294
2. Strategi perbandingan..... 294
3. Source code threshold statis..... 295
4. Source code perbandingan threshold vs Isolation Forest..... 296

5. Visualisasi hasil perbandingan..... 297
6. Cara membaca hasil..... 298
Jika threshold menang..... 298
Jika Isolation Forest menang..... 298
Jika keduanya punya kekuatan berbeda..... 298
7. Contoh implementasi nyata..... 298
Ruang server..... 298
Energi gedung..... 298
Mesin..... 298
8. Langkah kerja yang harus dilakukan pembaca..... 299
9. Penutup..... 300
Bab 10. Time-Series Forecasting untuk Sensor IoT..... 300
10.1 Mengapa forecasting penting dalam IoT?..... 301
10.2 Posisi Bab 10 dalam alur buku..... 301
10.3 Apa itu time-series forecasting?..... 302
10.4 Dasar berpikir forecasting untuk sensor IoT..... 302
10.5 Baseline: last value, moving average..... 303
10.5.1 Mengapa baseline wajib?..... 303
10.5.2 Last value..... 303
10.5.3 Moving average..... 304
10.6 Linear regression berbasis lag..... 305
10.6.1 Inti pendekatan..... 305
10.6.2 Mengapa cocok untuk IoT?..... 305
10.6.3 Kelebihan..... 305
10.6.4 Keterbatasan..... 305
10.6.5 Contoh implementasi nyata..... 305
10.7 Random Forest untuk time-series tabular..... 306
10.7.1 Mengapa Random Forest relevan?..... 306
10.7.2 Mengapa disebut “time-series tabular”?..... 306
10.7.3 Kelebihan..... 306
10.7.4 Keterbatasan..... 306
10.7.5 Kapan cocok?..... 306
10.8 LSTM sederhana dengan Keras..... 307
10.8.1 Apa itu LSTM?..... 307
10.8.2 Mengapa LSTM menarik untuk IoT?..... 307
10.8.3 Namun: kenapa “LSTM sederhana”?..... 307
10.8.4 Kelebihan LSTM..... 307
10.8.5 Keterbatasan LSTM..... 307
10.9 Kapan tidak perlu memakai LSTM..... 308
1. Data sedikit..... 308
2. Pola cukup sederhana..... 308
3. Fitur tabular sudah kuat..... 308
4. Sumber daya terbatas..... 308
5. Interpretasi penting..... 308

6. Kecepatan eksperimen lebih penting..... 309
10.10 Kapan LSTM layak dicoba?..... 309
10.11 Contoh-contoh nyata yang bisa diimplementasikan..... 310
Forecasting suhu ruang..... 310
Forecasting konsumsi energi..... 310
Forecasting kelembaban tanah..... 310
10.12 Langkah kerja yang harus dilakukan pembaca..... 311
Tahap 1 — Tentukan target forecasting..... 311
Tahap 2 — Bangun baseline..... 311
Tahap 3 — Bangun model klasik..... 311
Tahap 4 — Evaluasi dengan metrik yang tepat..... 311
Tahap 5 — Baru coba LSTM mini..... 311
Tahap 6 — Bandingkan hasil secara jujur..... 311
10.13 Sikap profesional dalam forecasting IoT..... 312
10.14 Penutup bab..... 312
Hands-on 24 - Forecasting Suhu Ruang..... 313
1. Tujuan hands-on..... 313
2. Dataset yang digunakan..... 313
3. Strategi forecasting yang dipakai..... 314
4. Persiapan Ubuntu 26.04..... 314
5. Menyiapkan dataset forecasting suhu..... 314
6. Baseline forecasting suhu: last value dan moving average..... 316
7. Linear Regression berbasis lag untuk suhu..... 317
8. Visualisasi hasil forecasting suhu..... 318
Hands-on 25 - Forecasting Konsumsi Energi Harian..... 319
1. Tujuan hands-on..... 319
2. Dataset yang digunakan..... 319
3. Baseline model: Linear Regression..... 320
4. Model kedua: Random Forest Regressor..... 320
5. Visualisasi prediksi energi..... 322
Hands-on 26 - Membandingkan Baseline vs Random Forest vs LSTM Mini..... 323
1. Tujuan hands-on..... 323
2. Dataset yang digunakan..... 323
3. Membandingkan baseline dan Random Forest..... 324
4. Persiapan TensorFlow/Keras..... 325
5. Menyiapkan sequence untuk LSTM mini..... 325
6. LSTM mini dengan Keras..... 326
7. Perbandingan akhir: baseline vs RF vs LSTM..... 328
8. Visualisasi perbandingan model..... 329
9. Cara membaca hasil..... 330
Jika last value menang..... 330
Jika moving average menang..... 330
Jika Random Forest menang..... 330
Jika LSTM menang tipis..... 330

Jika LSTM tidak menang.....330
10. Kapan tidak perlu memakai LSTM?.....331
11. Langkah kerja yang harus dilakukan pembaca.....331
12. Penutup.....331
Bab 11. Edge AI dan TinyML yang Realistis.....332
11.1 Mengapa Edge AI penting untuk IoT?.....332
11.2 Posisi Bab 11 dalam alur buku.....333
11.3 Inferensi di edge.....333
11.3.1 Apa itu inferensi?.....333
11.3.2 Apa itu inferensi di edge?.....333
11.3.3 Mengapa inferensi di edge menarik?.....334
11.3.4 Contoh implementasi nyata.....334
11.4 Pelatihan di laptop/cloud vs inferensi di perangkat.....335
11.4.1 Mengapa training dan inferensi sebaiknya dibedakan?.....335
11.4.2 Pelatihan di laptop atau cloud.....335
11.4.3 Inferensi di perangkat.....335
11.4.4 Pola kerja yang sehat.....336
11.4.5 Mengapa pola ini realistis?.....336
11.5 Konversi model ringan.....337
11.5.1 Mengapa model perlu dikonversi?.....337
11.5.2 Bentuk penyederhanaan model.....337
11.5.3 Strategi praktis yang disarankan.....337
11.5.4 Prinsip penting.....337
11.6 ESP32 / Raspberry Pi sebagai target praktis.....338
11.6.1 Mengapa dua platform ini dipilih?.....338
11.6.2 ESP32.....338
11.6.3 Raspberry Pi.....338
11.6.4 Rekomendasi praktis untuk pembaca.....339
11.7 TensorFlow Lite secara praktis.....340
11.7.1 Mengapa TensorFlow Lite?.....340
11.7.2 Apa itu TensorFlow Lite?.....340
11.7.3 Kapan TensorFlow Lite cocok?.....340
11.7.4 Kapan tidak perlu dipaksakan?.....340
11.8 Batasan memori, latensi, dan konsumsi daya.....341
11.8.1 Memori.....341
11.8.2 Latensi.....341
11.8.3 Konsumsi daya.....341
11.8.4 Pelajaran praktis.....341
11.9 Contoh-contoh nyata yang bisa diimplementasikan.....342
Smart room monitoring di Raspberry Pi.....342
Predictive maintenance ringan di gateway.....342
TinyML sederhana di ESP32.....342
Greenhouse edge inference.....342
11.10 Strategi implementasi yang realistis untuk pembaca.....343

Tahap 1 — Latih model di laptop Ubuntu..... 343
Tahap 2 — Simulasikan inferensi edge di laptop.....343
Tahap 3 — Pindahkan inferensi ke Raspberry Pi.....343
Tahap 4 — Eksplorasi TinyML ringan ke ESP32.....343
11.11 Langkah kerja yang harus dilakukan pembaca.....344
Tahap 1 — Tentukan model mana yang layak di deploy..... 344
Tahap 2 — Bedakan training dan inference..... 344
Tahap 3 — Pilih target perangkat..... 344
Tahap 4 — Siapkan format deployable.....344
Tahap 5 — Uji inferensi lokal.....344
Tahap 6 — Integrasikan ke pipeline IoT..... 344
Tahap 7 — Tinjau trade-off..... 345
11.12 Sikap profesional dalam Edge AI dan TinyML.....345
11.13 Penutup bab..... 345
Hands-on 27 - Ekspor Model Sederhana ke Format Ringan..... 347
1. Tujuan hands-on.....347
2. Contoh implementasi nyata.....347
3. Strategi yang dipakai.....347
4. Persiapan Ubuntu 26.04.....348
5. Ekspor model scikit-learn ringan..... 348
6. Ekspor model Keras ke TensorFlow Lite.....349
7. Apa yang harus dipahami pembaca?.....350
Hands-on 28 - Simulasi Inferensi Lokal di Edge..... 351
1. Tujuan hands-on.....351
2. Skenario yang akan dipakai..... 351
3. Source code inferensi lokal untuk model scikit-learn..... 352
4. Apa yang dipelajari?.....352
5. Simulasi inferensi lokal untuk TensorFlow Lite.....353
6. Hal penting yang harus dicatat.....354
7. Mengapa ini disebut “simulasi edge”?.....354
Hands-on 29 - Mini Proyek Klasifikasi Sederhana di Raspberry Pi / Emulator..... 355
1. Tujuan hands-on.....355
2. Target perangkat..... 355
Raspberry Pi.....355
Emulator..... 355
3. Struktur mini proyek..... 356
4. Source code mini proyek tanpa MQTT.....356
5. Versi mini proyek dengan MQTT opsional..... 357
6. Apa yang membuat proyek ini “realistis”?..... 359
7. Contoh pengembangan lanjutan..... 359
8. Langkah kerja yang harus dilakukan pembaca.....359
9. Penutup..... 360
Bagian IV — Integrasi End-to-End Open Source (Bab 12–13)..... 361
Bab 12. Pipeline IoT-ML End-to-End.....361

12.1 Mengapa pipeline end-to-end penting?.....361
12.2 Posisi Bab 12 dalam alur buku.....362
12.3 Arsitektur proyek dari sensor ke prediksi..... 362
12.3.1 Arsitektur minimal yang disarankan.....362
12.3.2 Peran setiap komponen.....363
12.3.3 Mengapa arsitektur ini kuat?..... 364
12.4 Ingest data dari MQTT ke Python..... 364
12.4.1 Mengapa MQTT → Python menjadi tulang punggung pipeline?.....364
12.4.2 Alur ingest yang sehat.....364
12.4.3 Validasi dasar saat ingest.....364
12.4.4 Struktur payload yang disarankan.....365
12.5 Penyimpanan data..... 365
12.5.1 Mengapa storage tidak boleh dianggap sekadar arsip?.....365
12.5.2 Penyimpanan yang realistis untuk buku ini..... 366
12.5.3 Data apa saja yang sebaiknya disimpan?.....366
12.5.4 Praktik minimum yang disarankan.....366
12.6 Batch retraining sederhana..... 367
12.6.1 Mengapa retraining perlu dibahas?.....367
12.6.2 Apa itu batch retraining sederhana?.....367
12.6.3 Kapan re-training dilakukan?.....367
12.6.4 Mengapa disebut “batch”?.....367
12.6.5 Strategi yang disarankan.....367
12.7 Inferensi otomatis.....368
12.7.1 Apa maksud inferensi otomatis?..... 368
12.7.2 Alur inferensi otomatis yang sehat..... 368
12.7.3 Hal yang perlu diperhatikan.....368
12.7.4 Inferensi otomatis untuk use case buku ini.....368
12.8 Dashboard / alert.....369
12.8.1 Mengapa hasil model harus diteruskan ke dashboard?.....369
12.8.2 Dashboard yang disarankan.....369
12.8.3 Alert sederhana yang realistis..... 369
12.8.4 Kapan alert dipicu?.....370
12.8.5 Prinsip penting.....370
12.9 Docker opsional.....370
12.9.1 Mengapa Docker dibahas sebagai opsional?.....370
12.9.2 Kapan Docker sangat membantu?.....370
12.9.3 Kapan belum perlu?..... 370
12.9.4 Sikap yang sehat.....371
12.10 Contoh arsitektur end-to-end yang bisa diimplementasikan..... 371
12.10.1 Smart Room Monitoring.....371
12.10.2 Predictive Maintenance Sederhana.....371
12.10.3 Smart Energy Analytics..... 371
12.11 Struktur folder proyek yang disarankan.....372
12.12 Langkah kerja yang harus dilakukan pembaca.....372

Tahap 1 — Tentukan use case.... 372
Tahap 2 — Tentukan arsitektur minimal.... 373
Tahap 3 — Implementasikan ingest dari MQTT.... 373
Tahap 4 — Tambahkan preprocessing.... 373
Tahap 5 — Jalankan inferensi otomatis.... 373
Tahap 6 — Simpan hasil.... 373
Tahap 7 — Hubungkan ke dashboard / alert.... 373
Tahap 8 — Tambahkan batch retraining sederhana.... 374
Tahap 9 — Bungkus dengan Docker bila perlu.... 374
12.13 Sikap profesional dalam membangun pipeline end-to-end.... 374
12.14 Penutup bab.... 374
Hands-on 30 - Pipeline Lengkap: Sensor/Simulator → MQTT → Python → Preprocessing → Prediksi → Storage → Dashboard.... 375
1. Tujuan hands-on.... 375
2. Skenario use case yang dipakai.... 375
3. Arsitektur minimal yang dipakai.... 376
4. Persiapan layanan di Ubuntu 26.04.... 376
4.1 Pasang Mosquitto.... 376
4.2 Jalankan InfluxDB dengan Docker.... 376
4.3 Jalankan Grafana.... 377
4.4 Pasang library Python.... 377
5. Menyiapkan model yang akan dipakai.... 377
6. Simulator sensor: kirim data ke MQTT.... 378
7. Subscriber Python: ingest, preprocessing, prediksi, storage.... 379
8. Verifikasi ingest MQTT.... 381
9. Menyusun dashboard Grafana.... 382
10. Apa yang dipelajari pembaca dari Hands-on 30?.... 382
Hands-on 31 - Membuat Alert Sederhana jika Skor Anomali Tinggi.... 384
1. Tujuan hands-on.... 384
2. Dua cara sederhana membuat alert.... 384
Pendekatan A — alert berbasis status model.... 384
Pendekatan B — alert berbasis skor / threshold.... 384
3. Menambahkan publisher alert ke pipeline.... 384
4. Verifikasi alert di MQTT.... 387
5. Menampilkan alert di Grafana.... 387
6. Versi sederhana berbasis skor anomali.... 388
7. Kapan alert sederhana sudah cukup?.... 388
8. Kapan alert perlu dikembangkan?.... 389
9. Langkah kerja yang harus dilakukan pembaca.... 389
10. Penutup.... 389
Bab 13. Keamanan, Kualitas Model, dan Operasional.... 391
13.1 Mengapa bab ini penting?.... 391
13.2 Risiko data palsu, data hilang, data rusak.... 391
13.2.1 Mengapa kualitas data adalah garis pertahanan pertama?.... 391

13.2.2 Data palsu..... 391
13.2.3 Data hilang..... 392
13.2.4 Data rusak..... 392
13.2.5 Langkah minimum yang wajib dilakukan..... 392
13.2.6 Contoh validasi sederhana..... 393
13.3 Dasar keamanan MQTT..... 393
13.3.1 Mengapa MQTT security perlu dibahas?..... 393
13.3.2 Tiga dasar keamanan MQTT yang realistis untuk buku ini..... 393
13.3.3 Praktik minimum di level topik..... 394
13.3.4 Praktik operasional yang sehat..... 394
13.3.5 Batas pembahasan yang realistis..... 394
13.4 Drift data dan penurunan performa model..... 394
13.4.1 Apa itu drift?..... 394
13.4.2 Bentuk drift yang umum..... 395
13.4.3 Mengapa drift berbahaya?..... 395
13.4.4 Tanda-tanda drift operasional..... 395
13.4.5 Langkah minimum untuk memonitor drift..... 395
13.5 Logging, monitoring, dan versi model sederhana..... 396
13.5.1 Mengapa logging wajib?..... 396
13.5.2 Apa yang minimal harus dicatat?..... 396
13.5.3 Monitoring minimum yang realistis..... 396
13.5.4 Versi model sederhana..... 396
13.5.5 Kenapa versi model penting?..... 397
13.5.6 Praktik minimum yang sangat disarankan..... 397
13.6 Explain Ability ringan..... 397
13.6.1 Mengapa explainability perlu dibahas?..... 397
13.6.2 Explain Ability yang realistis untuk buku ini..... 397
13.6.3 Contoh explain ability ringan..... 398
13.6.4 Mengapa model pohon/random forest cocok?..... 398
13.6.5 Kapan explain ability cukup dianggap “baik”?..... 398
13.7 Synthetic data untuk stress testing pipeline..... 398
13.7.1 Mengapa synthetic data relevan lagi di bab operasional?..... 398
13.7.2 Jenis stress test yang realistis..... 399
13.7.3 Tujuan stress testing..... 399
13.7.4 Praktik minimum yang sangat bagus untuk pembaca..... 399
13.7.5 Kenapa ini penting?..... 399
13.8 Contoh-contoh nyata yang bisa diimplementasikan..... 400
Smart room monitoring..... 400
Predictive maintenance sederhana..... 400
Smart energy analytics..... 400
13.9 Langkah kerja yang harus dilakukan pembaca..... 400
Tahap 1 — Lindungi data masuk..... 400
Tahap 2 — Amankan MQTT minimum..... 400
Tahap 3 — Pantau drift..... 401

Tahap 4 — Bangun logging dan monitoring..... 401
Tahap 5 — Versikan model.....401
Tahap 6 — Tambahkan explain ability ringan.....401
Tahap 7 — Lakukan stress test..... 401
Tahap 8 — Evaluasi kesiapan operasional.....401
13.10 Sikap profesional dalam keamanan, kualitas model, dan operasional.....402
13.11 Penutup bab..... 402
Hands-on 32 - Memonitor Perubahan Distribusi Data.....403
1. Tujuan hands-on.....403
2. Contoh implementasi nyata.....403
Smart room monitoring.....403
Smart energy analytics.....403
Predictive maintenance sederhana..... 403
3. Prinsip monitoring drift yang realistis.....404
4. Dataset yang dipakai.....404
5. Persiapan Ubuntu 26.04.....404
6. Source code utama: monitoring distribusi data..... 405
7. Cara membaca hasil..... 406
8. Menambahkan indikator drift sederhana.....406
9. Insight penting Hands-on 32..... 406
Hands-on 33 - Feature Importance untuk Menjelaskan Prediksi..... 408
1. Tujuan hands-on.....408
2. Mengapa feature importance penting?.....408
3. Dataset dan model yang dipakai.....408
4. Source code utama..... 409
5. Cara membaca hasil..... 410
6. Explainability ringan untuk satu prediksi..... 410
7. Mengapa ini cukup untuk tahap ini?.....410
8. Insight penting Hands-on 33.....411
Hands-on 34 - Checklist Operasional Sistem IoT-ML..... 412
1. Tujuan hands-on.....412
2. Mengapa checklist operasional penting?..... 412
3. Format checklist yang disarankan.....412
4. Template checklist operasional..... 413
5. Cara memakai checklist ini.....414
6. Versi checklist yang lebih kuantitatif.....414
7. Insight penting Hands-on 34..... 415
Penutup..... 415
Bagian V — Studi Kasus Praktis (Bab 14–15)..... 416
Bab 14. Studi Kasus 1–3: Dari Dataset ke Solusi..... 416
14.1 Mengapa studi kasus penting di bagian akhir buku?..... 416
14.2 Pola yang dipakai untuk setiap studi kasus.....416
Studi Kasus A — Smart Room Monitoring..... 418
14.A.1 Masalah.....418

14.A.2 Arsitektur....418
14.A.3 Dataset....419
Opsi 1 — Data sensor nyata....419
Opsi 2 — Dummy data / synthetic data....419
14.A.4 Synthetic data fallback....419
14.A.5 Preprocessing....420
Langkah 1 — Validasi payload....420
Langkah 2 — Parsing waktu....420
Langkah 3 — Buat fitur ringan....420
Langkah 4 — Buat label....420
Langkah 5 — Split data time-aware....420
14.A.6 Model....421
Model baseline....421
Model utama....421
Target....421
14.A.7 Evaluasi....421
14.A.8 Dashboard real-time....421
14.A.9 Deployment mini....422
Studi Kasus B — Predictive Maintenance Sederhana....423
14.B.1 Masalah....423
14.B.2 Arsitektur....423
14.B.3 Dataset....424
Opsi 1 — Data nyata....424
Opsi 2 — Dataset sintetis....424
14.B.4 Synthetic data fallback....424
14.B.5 Preprocessing....424
Langkah 1 — Windowing....424
Langkah 2 — Ekstraksi fitur....424
Langkah 3 — Validasi sinyal....425
Langkah 4 — Label atau skor....425
14.B.6 Model....425
Lapis 1 — anomaly detection....425
Lapis 2 — klasifikasi fault dasar....425
14.B.7 Evaluasi....426
14.B.8 Interpretasi hasil....426
14.B.9 Deployment mini....426
Studi Kasus C — Smart Energy Analytics....428
14.C.1 Masalah....428
14.C.2 Arsitektur....428
14.C.3 Dataset....429
Opsi 1 — Data nyata....429
Opsi 2 — Dataset sintetis....429
14.C.4 Synthetic data fallback....429
14.C.5 Preprocessing....429

Langkah 1 — Parsing waktu....429
Langkah 2 — Buat fitur waktu.... 429
Langkah 3 — Buat fitur historis.... 430
Langkah 4 — Tentukan target.... 430
14.C.6 Model.... 430
Forecasting beban....430
Alert beban tidak normal....430
14.C.7 Evaluasi.... 431
14.C.8 Alert beban tidak normal....431
14.C.9 Deployment mini.... 431
● gedung kecil.... 432
14.10 Benang merah tiga studi kasus.... 432
Smart Room Monitoring....432
Predictive Maintenance Sederhana....432
Smart Energy Analytics.... 432
14.11 Langkah kerja yang harus dilakukan pembaca.... 433
Tahap 1 — Definisikan masalah....433
Tahap 2 — Susun arsitektur minimal....433
Tahap 3 — Tentukan dataset....433
Tahap 4 — Lakukan preprocessing....433
Tahap 5 — Pilih model sesuai masalah....433
Tahap 6 — Evaluasi....433
Tahap 7 — Buat deployment mini....434
14.12 Sikap profesional dalam menyusun studi kasus.... 434
14.13 Penutup bab....434
Hands-on 35 - Studi Kasus A — Smart Room Monitoring lengkap....435
1. Masalah....435
2. Arsitektur.... 435
3. Dataset....435
Opsi data nyata.... 435
Opsi synthetic fallback....436
4. Preprocessing.... 437
Langkah 1 — validasi payload....437
Langkah 2 — parsing waktu....437
Langkah 3 — buat fitur.... 437
Langkah 4 — buat label....437
5. Model....438
6. Evaluasi....438
7. Dashboard real-time....438
8. Deployment mini....439
8.1 Simulator publisher MQTT....439
8.2 Subscriber + klasifikasi....440
Hands-on 36 - Studi Kasus B — Predictive Maintenance Sederhana lengkap....441
1. Masalah....441

2. Arsitektur 441
3. Dataset 442
4. Preprocessing 442
Langkah 1 — windowing 442
Langkah 2 — ekstraksi fitur 442
Langkah 3 — validasi 442
5. Model 443
Lapis 1 — anomaly detection 443
Lapis 2 — klasifikasi fault dasar 443
6. Evaluasi 443
7. Interpretasi hasil 443
8. Deployment mini 444
8.1 Generator dataset sintetis getaran + arus 444
8.2 Ekstraksi fitur per jendela 445
8.3 Anomaly detection + klasifikasi 446
Hands-on 37 - Studi Kasus C — Smart Energy Analytics lengkap 447
1. Masalah 447
2. Arsitektur 447
3. Dataset 447
Opsi data nyata 447
Opsi synthetic fallback 447
4. Preprocessing 448
Langkah 1 — parsing waktu 448
Langkah 2 — buat fitur waktu 448
Langkah 3 — buat fitur historis 448
Langkah 4 — target forecasting 448
5. Model 448
Forecasting 448
Alert beban tidak normal 448
6. Evaluasi 449
7. Deployment mini 449
7.1 Generator energi sintetis 449
7.2 Forecasting + alert sederhana 450
7.3 Dashboard mini 451
Penutup 451
Bab 15. Proyek Akhir Buku: Sistem IoT-ML Open Source Lengkap 453
15.1 Tujuan proyek akhir 453
15.2 Mengapa proyek ini penting? 453
15.3 Masalah yang ingin diselesaikan 454
15.4 Komponen utama proyek 454
15.4.1 Data sensor nyata / simulator / synthetic generator 454
15.4.2 Broker MQTT 454
15.4.3 Skrip subscriber Python 455
15.4.4 Preprocessing otomatis 455

15.4.5 Model anomaly detection....455
15.4.6 Dashboard Grafana....456
15.4.7 Notifikasi sederhana....456
15.5 Arsitektur proyek akhir....456
15.6 Struktur folder proyek....458
15.7 Dataset dan sumber data....459
15.7.1 Sensor nyata....459
15.7.2 Simulator....459
15.7.3 Synthetic generator....459
15.8 Preprocessing otomatis....460
15.9 Model anomaly detection....460
Mengapa threshold tetap dipakai?....461
Mengapa Isolation Forest?....461
Strategi hybrid yang disarankan....461
15.10 Dashboard Grafana....461
15.11 Notifikasi sederhana....462
Opsi notifikasi yang realistis....462
Contoh aturan notifikasi....462
15.12 Cara pengujian....463
15.12.1 Uji aliran dasar....463
15.12.2 Uji preprocessing....463
15.12.3 Uji model....463
15.12.4 Uji dashboard....463
15.12.5 Uji skenario anomali....463
15.12.6 Uji stabilitas ringan....463
15.13 Cara pengembangan lanjut....464
15.13.1 Multi-device....464
15.13.2 Multi-sensor fusion....464
15.13.3 Retraining berkala....464
15.13.4 Edge deployment....464
15.13.5 Alert yang lebih matang....464
15.13.6 Keamanan lebih baik....465
15.13.7 Docker untuk reproducibility....465
15.14 Alur implementasi proyek akhir yang disarankan....465
Tahap 1 — mulai dari simulator....465
Tahap 2 — hidupkan MQTT dan subscriber....465
Tahap 3 — tambahkan preprocessing otomatis....465
Tahap 4 — tambahkan model anomaly detection....465
Tahap 5 — simpan hasil ke storage....465
Tahap 6 — buat dashboard Grafana....465
Tahap 7 — tambahkan notifikasi sederhana....466
Tahap 8 — uji kondisi buruk....466
Tahap 9 — rapikan struktur proyek....466
15.15 Contoh keluaran proyek akhir yang baik....466

15.16 Sikap profesional dalam proyek akhir..... 466
15.17 Penutup bab..... 466
Hands-on 38 - Membangun Proyek Akhir End-to-End..... 468
1. Tujuan hands-on..... 468
2. Skenario proyek yang dipakai..... 468
3. Arsitektur sistem..... 469
Peran tiap komponen..... 469
4. Struktur folder proyek..... 469
5. Persiapan environment Ubuntu 26.04..... 470
5.1 Pasang Mosquitto..... 470
5.2 Jalankan InfluxDB..... 470
5.3 Jalankan Grafana..... 470
5.4 Pasang library Python..... 470
6. Menyiapkan data: simulator / synthetic generator..... 471
7. Menyiapkan model anomaly detection..... 472
8. Subscriber Python + preprocessing otomatis + inferensi + storage..... 473
9. Dashboard Grafana..... 476
10. Verifikasi sistem end-to-end..... 476
Hands-on 39 - Menambahkan Notifikasi Sederhana..... 477
1. Tujuan hands-on..... 477
2. Strategi notifikasi yang dipakai..... 477
3. Modifikasi subscriber agar mengirim alert..... 477
4. Verifikasi alert di MQTT..... 479
5. Tampilkan alert di Grafana..... 479
6. Insight Hands-on 39..... 480
Hands-on 40 - Uji Beban serta Skenario Pengembangan Lanjut..... 481
1. Tujuan hands-on..... 481
2. Jenis pengujian yang realistis..... 481
3. Uji volume: percepat laju publisher..... 482
4. Uji payload rusak..... 483
5. Uji drift buatan..... 484
6. Checklist hasil uji beban..... 485
7. Skenario pengembangan lanjut..... 485
7.1 Multi-device..... 485
7.2 Multi-sensor fusion..... 485
7.3 Edge deployment..... 485
7.4 Retraining berkala..... 485
7.5 Security hardening..... 486
7.6 Docker dan reproducibility..... 486
7.7 Explainability ringan..... 486
8. Bentuk hasil akhir capstone yang ideal..... 486
9. Penutup..... 487
Bab Penutup..... 488
Epilog. Roadmap Belajar dan Pengembangan Lanjut..... 488

Jalur belajar 1 — Python IoT Developer..... 488
Jalur belajar 2 — Data Analyst IoT.....489
Jalur belajar 3 — ML Engineer untuk edge/IIoT.....490
Kapan pindah ke Kafka atau Spark?.....491
Kapan masuk ke MLOps yang lebih serius?.....491
Kapan lanjut ke TinyML yang lebih serius?.....492
Kapan federated learning mulai relevan?.....493
Komunitas open source.....493
Dataset publik untuk latihan..... 494
Strategi belajar setelah buku ini selesai.....494
Penutup epilog..... 495
Appendix A. Instalasi dan Setup Open Source Stack.....496
Python..... 496
Jupyter.....497
Mosquitto.....497
Node-RED..... 498
InfluxDB / SQLite.....498
Grafana.....499
1. Menjalankan Grafana..... 499
2. Login pertama kali..... 500
3. Ganti password awal..... 500
4. Tambahkan data source..... 500
5. Hubungkan ke InfluxDB.....500
6. Buat panel pertama..... 501
7. Buat panel historis..... 501
8. Buat panel status atau alert sederhana.....502
Penutup Appendix A.....502
Appendix B. Dataset IoT Publik untuk Latihan.....503
Dataset lingkungan.....503
Dataset energi..... 503
Dataset mesin / predictive maintenance.....504
Dataset wearable.....504
Cara memakai Appendix B.....504
Appendix C. Template Kode Python.....505
C.1 Template koneksi MQTT.....505
Contoh publisher MQTT sederhana..... 505
Contoh subscriber MQTT sederhana..... 506
C.2 Template loader CSV.....506
Contoh loader CSV dasar.....507
C.3 Template preprocessing.....507
Contoh fungsi preprocessing singkat..... 507
C.4 Template training model.....508
Contoh training model regresi..... 508
Contoh training model klasifikasi.....509

Contoh deteksi anomali....509
C.5 Template evaluasi....509
Untuk klasifikasi....509
Contoh evaluasi klasifikasi....509
Untuk regresi / forecasting....510
Contoh evaluasi regresi....510
C.6 Template inferensi....510
Contoh inferensi dari satu record....511
Contoh inferensi dari payload JSON....511
Contoh simpan hasil inferensi....511
Penutup Appendix C....511
Appendix D. Template Generator Synthetic Data....513
Generator suhu dan kelembaban....513
Generator energi....514
Generator getaran mesin....515
Injector noise, outlier, missing value, drift....516
Penekanan....517
Appendix E. Glosarium IoT, ML, dan Analytics....518
Appendix F. Referensi dan Proyek Open Source Lanjutan....531
F.1 Prinsip menggunakan referensi lanjutan....531
F.2 Referensi teknis utama....531
F.3 Kategori proyek open source lanjutan....532
F.4 Jalur pengembangan tematik....532
F.5 Strategi memilih proyek lanjutan....533
F.6 Komunitas open source yang layak diikuti....533
F.7 Penutup Appendix F....533
Referensi....535
Tentang Penulis....536

Bagian I — Fondasi Praktis IoT Analytics

Bab 1. IoT, Data, dan Machine Learning: Gambaran Besar

Dunia komputasi modern tidak lagi hanya berbicara tentang aplikasi di layar komputer atau ponsel. Hari ini, banyak keputusan penting justru muncul dari **data yang ditangkap oleh benda-benda fisik**: sensor suhu di ruang server, meter energi di gedung, sensor getaran di mesin industri, sensor kelembaban di lahan pertanian, kamera di jalan raya, hingga *wearable device* di tubuh manusia. Semua perangkat ini menghasilkan aliran data yang terus bergerak. Ketika data tersebut dikumpulkan, dikirim, disimpan, dianalisis, lalu dipakai untuk mengambil tindakan, di situlah kita masuk ke dunia **Internet of Things** atau **IoT**. Konteks ini sejalan dengan pembahasan referensi IoT yang menempatkan IoT sebagai pertemuan dunia fisik dengan perangkat komputasi, perangkat lunak, konektivitas, penyimpanan data, analitik, dan pengguna manusia yang mengkonsumsi hasilnya.

Namun IoT tanpa analitik hanyalah sekumpulan perangkat yang “ramai mengirim data.” Nilai sebenarnya baru muncul saat data tersebut diproses menjadi **informasi**, lalu ditingkatkan lagi menjadi **keputusan**. Di sinilah *machine learning* berperan. Dengan *machine learning*, sistem tidak hanya menampilkan angka, tetapi juga dapat **mengenali pola**, **memprediksi kejadian**, **mendeteksi anomali**, dan **memberi rekomendasi tindakan**. Literatur IoT analytics juga menegaskan bahwa analitik IoT banyak bertumpu pada tiga wilayah besar: **pemodelan kinerja, statistik, dan machine learning**, karena sistem IoT harus dinilai dan dipahami baik dari sisi perangkat, jaringan, komputasi, maupun kualitas keputusan yang dihasilkan.

Buku ini karena itu mengambil posisi yang sangat penting: bukan sekadar membahas IoT sebagai jaringan perangkat, dan bukan pula membahas *machine learning* sebagai model matematis yang berdiri sendiri. Buku ini membahas **bagaimana data dari dunia nyata mengalir dari sensor sampai menjadi aksi yang berguna**, dengan fondasi perangkat lunak **open source** agar dapat dipelajari, diulang, dimodifikasi, dan diterapkan secara mandiri.

1.2 Apa itu IoT?

> Secara sederhana, **IoT adalah ekosistem perangkat fisik yang mampu mengindra, berkomunikasi, dan bertindak melalui jaringan**. Perangkat ini dapat berupa sensor, aktuator, mikrokontroler, *gateway*, komputer mini, mesin industri, kendaraan, alat medis, atau benda sehari-hari yang diberi kemampuan digital. Sensor membaca kondisi fisik seperti suhu, cahaya, tekanan, getaran, kelembaban, posisi, arus listrik, dan sebagainya. Data hasil penginderaan kemudian dikirim ke sistem lain untuk diolah lebih lanjut. Setelah dianalisis, sistem dapat memutuskan apakah perlu memberi notifikasi, menyalakan aktuator, mengubah setelan operasi, atau menyimpan hasil untuk evaluasi jangka panjang.
>
> Dari sisi arsitektur, handbook IoT menjelaskan bahwa arsitektur IoT hadir sebagai “lem” yang menyatukan berbagai komponen seperti teknologi sensor, pengumpulan data, konektivitas, penyimpanan data, analitik, visualisasi, logika bisnis, dan

pengalaman pengguna. Artinya, IoT bukan hanya urusan sensor dan jaringan, tetapi juga **bagaimana semua komponen bekerja sebagai satu sistem yang bernilai bagi manusia**.

Agar lebih mudah dipahami, bayangkan sebuah **ruang kelas cerdas**. Di dalamnya ada sensor suhu, kelembaban, kualitas udara, dan konsumsi listrik. Setiap beberapa detik, sensor mengirim data ke server lokal. Data ini disimpan dan divisualisasikan dalam dashboard. Sebuah model sederhana lalu menganalisis apakah ruangan terlalu panas, terlalu lembab, atau boros energi. Jika kondisi tidak ideal, sistem dapat mengirim peringatan atau bahkan mengaktifkan perangkat tertentu. Seluruh alur itu adalah IoT.

Ciri utama sistem IoT

- **Menghubungkan dunia fisik dengan dunia digital**
- **Menghasilkan data secara terus-menerus**
- **Membutuhkan konektivitas dan interoperabilitas**
- **Bernilai ketika data berubah menjadi aksi**
- **Sering bekerja dalam kondisi nyata yang tidak sempurna**

1.3 Apa itu IIoT?

Industrial Internet of Things atau **IIoT** adalah penerapan konsep IoT di lingkungan industri, manufaktur, energi, utilitas, logistik, dan infrastruktur kritis. Jika IoT umum sering dipakai untuk rumah pintar, kota pintar, atau perangkat konsumen, maka IIoT beroperasi di lingkungan yang jauh lebih menuntut: mesin harus tetap berjalan, data harus akurat, dan keterlambatan keputusan bisa menimbulkan kerugian besar. Handbook IoT secara khusus membedakan IIoT sebagai domain yang memiliki aplikasi, standar konektivitas, dan arsitektur yang lebih berorientasi pada keandalan serta operasi industri.

Perbedaan paling terasa antara IoT umum dan IIoT terletak pada **konsekuensi kegagalan**. Bila sensor suhu di rumah salah baca, mungkin dampaknya hanya membuat AC tidak nyaman. Tetapi bila sensor getaran di pabrik gagal mendeteksi kerusakan awal pada motor, kerusakan bisa menjalar ke mesin lain, menghentikan produksi, bahkan membahayakan keselamatan kerja. Karena itu IIoT biasanya lebih menekankan:

- **keandalan**
- **ketahanan**
- **akurasi**
- **integrasi dengan sistem operasi industri**
- **respon cepat**
- **keamanan yang lebih ketat**

Dalam bahasa sederhana, **IoT sering fokus pada kenyamanan dan efisiensi**, sedangkan **IIoT fokus pada kontinuitas operasi, keselamatan, dan optimasi proses**.

1.4 Apa itu *data-driven decision making*?

Istilah **data-driven decision making** berarti **pengambilan keputusan berbasis data, bukan sekadar intuisi atau kebiasaan**. Dalam konteks IoT, data dihasilkan oleh sensor dan sistem lapangan, lalu diproses menjadi informasi yang bisa dipakai untuk memutuskan sesuatu secara lebih objektif.

Contohnya:

- Di gedung pintar, keputusan menyalakan pendingin ruangan tidak lagi berdasarkan “rasanya panas,” tetapi berdasarkan data suhu, kelembaban, jumlah orang di ruangan, dan pola penggunaan listrik.
- Di industri, keputusan penjadwalan perawatan mesin tidak lagi menunggu mesin rusak, tetapi berdasarkan data getaran, temperatur motor, arus listrik, dan prediksi *machine learning*.
- Di pertanian, keputusan irigasi tidak lagi hanya mengikuti jadwal tetap, tetapi berdasarkan kelembaban tanah, cuaca, dan pola historis.

Buku *Big Data Analytics for Internet of Things* bahkan menempatkan masalah inti IoT bukan hanya soal data, tetapi soal **jembatan antara data dan keputusan**. Ini penting: data yang banyak tidak otomatis menghasilkan nilai; yang dibutuhkan adalah kemampuan mengubah data menjadi tindakan yang tepat.

Tingkatan kematangan keputusan berbasis data

- **Deskriptif**: Apa yang sedang terjadi?
- **Diagnostik**: Mengapa itu terjadi?
- **Prediktif**: Apa yang kemungkinan akan terjadi?
- **Preskriptif**: Apa tindakan terbaik yang sebaiknya dilakukan?

Pada tahap awal, sistem IoT mungkin hanya menampilkan suhu ruangan. Itu baru deskriptif. Ketika sistem mulai memberi tahu bahwa lonjakan suhu disebabkan ventilasi buruk, itu diagnostik. Saat sistem mampu memprediksi suhu ruangan akan melebihi ambang batas dalam 20 menit, itu prediktif. Ketika sistem menyarankan menyalakan kipas, membuka ventilasi, atau menggeser jadwal peralatan tertentu, itu sudah masuk preskriptif.

1.5 Alur Ujung ke Ujung: Sensor → Gateway → Broker → Storage → Analytics → Model → Aksi

Salah satu hal paling penting yang harus dipahami pembaca sejak awal adalah bahwa sistem IoT yang cerdas bukan sekadar “sensor kirim data ke cloud.” Sistem yang baik memiliki **rantai kerja ujung ke ujung**.

Mari kita bahas satu per satu.

1.5.1 Sensor

Sensor adalah titik awal. Sensor mengubah fenomena fisik menjadi sinyal digital yang dapat dibaca komputer. Sensor suhu membaca temperatur, sensor kelembaban membaca kadar uap air, sensor arus membaca beban listrik, dan sensor getaran membaca pola mekanik mesin.

Tanpa sensor, tidak ada data lapangan. Tetapi perlu diingat: sensor bukan sumber kebenaran mutlak. Sensor bisa:

- salah baca,
- rusak,
- terlambat,
- *drift*,
- menghasilkan *noise*,
- atau mengirim data dengan interval yang tidak stabil.

Karena itu, data IoT sejak awal harus dianggap sebagai data yang hidup di dunia nyata, bukan data laboratorium yang selalu bersih.

1.5.2 Gateway

Gateway adalah penghubung antara perangkat lapangan dan sistem pemrosesan yang lebih tinggi. Tidak semua sensor langsung berbicara ke server atau jaringan IP. Dalam banyak sistem, sensor kecil mengirim data ke perangkat perantara seperti ESP32, Raspberry Pi, atau komputer mini lain yang bertugas:

- mengumpulkan data dari beberapa sensor,
- menerjemahkan protokol,
- menyaring data awal,
- memberi *timestamp*,
- dan meneruskan data ke jaringan.

Pada model referensi IoT yang dibahas Perros, lapisan bawah menangani perangkat IoT dan controller, lalu data diteruskan melalui lapisan jaringan, *fog computing*, penyimpanan data, abstraksi data, aplikasi, hingga kolaborasi/proses bisnis. Ini membantu kita melihat bahwa *gateway* dan komputasi dekat perangkat bukan tambahan mewah, tetapi bagian wajar dari sistem IoT modern.

1.5.3 Broker

Dalam sistem berbasis *publish-subscribe*, khususnya MQTT, **broker** berfungsi sebagai “kantor pos data.” Sensor atau *gateway* tidak harus tahu siapa yang akan membaca datanya. Mereka cukup mengirim (*publish*) ke topik tertentu, misalnya:

- `ruang101/suhu`
- `mesinA/getaran`
- `gedung1/energi`

Pihak lain yang butuh data tersebut tinggal berlangganan (*subscribe*) ke topik yang sama. Dengan model ini, sistem menjadi lebih fleksibel, modular, dan mudah diperluas. Untuk buku berbasis open source, penggunaan **Mosquitto MQTT** sangat tepat karena ringan, matang, luas dipakai, dan mudah dipasang di Linux maupun Raspberry Pi.

1.5.4 Storage

Data yang masuk perlu disimpan agar bisa dianalisis ulang, divisualisasikan, dipakai untuk pelatihan model, dan diaudit jika terjadi masalah. Pada tahap awal belajar, penyimpanan bisa sederhana:

- CSV
- SQLite
- PostgreSQL
- InfluxDB untuk *time series*

Dalam model referensi IoT, lapisan penyimpanan data berfungsi menampung berbagai format data dari banyak sumber. Data dapat dikonsolidasikan di satu tempat atau tersebar di beberapa lokasi, lalu diakses oleh aplikasi di lapisan lebih tinggi.

1.5.5 Analytics

Setelah data tersimpan, tahap berikutnya adalah **analytics**. Pada tahap ini kita mulai bertanya:

- Bagaimana pola suhu harian?
- Jam berapa konsumsi energi tertinggi?
- Kapan mesin mulai menunjukkan gejala aneh?
- Sensor mana yang paling sering gagal?
- Apakah ada korelasi antara kelembaban dan konsumsi listrik?

Di sini peran Python menjadi sangat kuat. Dengan **pandas**, **NumPy**, dan **matplotlib**, pembaca dapat membersihkan data, melakukan agregasi, menghitung statistik, dan membuat visualisasi sederhana namun bermakna.

1.5.6 Model

Jika analitik membantu kita memahami masa lalu dan kondisi sekarang, maka **model *machine learning*** membantu kita melihat pola lebih dalam atau memprediksi masa depan. Model dapat dipakai untuk:

- klasifikasi kondisi normal vs tidak normal,
- prediksi nilai sensor berikutnya,
- deteksi anomali,
- perkiraan konsumsi energi,
- prediksi potensi kerusakan.

Pada banyak kasus IoT, model sederhana justru sangat berguna. Tidak semua masalah butuh *deep learning*. Sering kali *random forest*, *logistic regression*, *isolation forest*, atau model berbasis *lag feature* sudah cukup kuat, cepat, dan jauh lebih mudah dijelaskan ke pengguna lapangan.

1.5.7 Aksi

Tahap terakhir adalah **aksi**. Ini yang membedakan IoT dari sekadar sistem monitoring. Aksi bisa berupa:

- mengirim notifikasi Telegram/email,
- menyalakan kipas,
- mematikan pompa,
- mengubah ambang alarm,
- membuat tiket maintenance,
- atau menampilkan peringatan di dashboard.

Pada titik inilah data benar-benar berubah menjadi nilai.

1.6 Mengapa Machine Learning di IoT Berbeda?

Secara teori, *machine learning* adalah teknik belajar dari data. Tetapi saat diterapkan pada IoT, ada tantangan tambahan yang membuat pendekatannya berbeda dari ML biasa pada data statis. Ada empat kata kunci:

streaming, noise, keterbatasan device, dan kebutuhan inferensi cepat.

1.6.1 Data bersifat *streaming*, bukan diam

Banyak dataset ML klasik berbentuk tabel statis: kumpulkan data, bersihkan, latih model, selesai. Di IoT, data sering datang **terus-menerus**, detik demi detik, menit demi menit. Ini berarti sistem harus siap menangani:

- aliran data berkelanjutan,
- data datang tidak selalu teratur,
- kebutuhan pengolahan *near-real-time*,
- perubahan pola dari waktu ke waktu.

Buku Perros menegaskan bahwa pada aplikasi IoT *real-time*, waktu yang dibutuhkan sistem untuk menerima data sensor, memprosesnya, lalu menentukan respons bisa menjadi sangat kritis. Untuk aplikasi non-*real-time*, keterlambatan beberapa saat mungkin masih bisa ditoleransi; untuk aplikasi *real-time*, tidak.

1.6.2 Data sensor sering berisik (*noise*)

Data IoT jarang sempurna. Sensor murah bisa menghasilkan pembacaan yang meloncat. Koneksi bisa putus. Nilai bisa hilang. Perangkat bisa panas dan mulai *drift*. Karena itu, sebelum bicara model yang canggih, pembaca harus belajar menghargai pentingnya:

- pembersihan data,
- *resampling*,
- imputasi,
- *smoothing*,
- dan validasi kualitas data.

Inilah sebabnya bab-bab awal buku Anda memang lebih menekankan **data acquisition, cleaning, visualization**, lalu baru masuk ke *machine learning*.

1.6.3 Perangkat sering terbatas

Tidak semua inferensi dilakukan di server besar. Pada banyak skenario, keputusan harus diambil dekat sumber data: di Raspberry Pi, *gateway*, atau bahkan mikrokontroler. Ini berarti model harus mempertimbangkan:

- memori terbatas,
- CPU kecil,
- daya listrik kecil,
- koneksi tidak selalu stabil,
- dan kebutuhan model yang ringan.

Referensi juga menempatkan *fog computing*, server lapisan atas, dan perangkat di lapisan bawah sebagai bagian dari evaluasi performa IoT. Artinya, desain model tidak boleh dilepaskan dari kemampuan komputasi tempat model itu dijalankan.

1.6.4 Inferensi sering harus cepat

Dalam prediksi perawatan mesin, keterlambatan beberapa detik mungkin masih aman. Dalam deteksi bahaya pada sistem kritis, keterlambatan bisa menjadi masalah besar. Karena itu di IoT, sering kali kita lebih menghargai model yang:

- cukup akurat,
- ringan,
- stabil,
- cepat,
- dan mudah dijelaskan,

dibanding model yang sangat rumit tetapi lambat dan sulit dipelihara.

1.6.5 Konteks sangat penting

Angka “85” tidak berarti apa-apa jika kita tidak tahu itu suhu, kelembaban, tekanan, atau arus. Bahkan jika itu suhu, kita masih perlu tahu satuannya, lokasi sensor, waktu pencatatan, kondisi lingkungan, dan hubungan dengan sensor lain. ML di IoT hampir selalu lebih kuat bila diperkaya dengan konteks:

- *timestamp*
- lokasi
- identitas perangkat
- mode operasi
- status lingkungan
- histori pembacaan

Jadi, **ML di IoT bukan sekadar memilih algoritma**, melainkan merancang alur data yang benar sejak awal.

1.7 Open-Source Stack untuk Buku Ini

Buku ini secara sadar dibangun di atas **perangkat lunak open source**, bukan perangkat proprietary. Pilihan ini penting bukan hanya karena lebih hemat biaya, tetapi juga karena:

- pembaca dapat belajar tanpa terkunci lisensi,
- sistem dapat dipasang ulang di laboratorium kampus, komunitas, sekolah, atau industri kecil,
- kode dapat dilihat, dimodifikasi, dan dikembangkan,
- dan ekosistem open source mendorong pembelajaran yang lebih mandiri.

Handbook IoT bahkan menyediakan pembahasan khusus tentang **IoT open-source systems**, manfaat penggunaan open source, serta pentingnya memilih proyek open source yang tepat untuk pekerjaan yang tepat.

Stack yang digunakan

Berikut stack inti:

- **Python**
  Bahasa utama untuk analitik, *machine learning*, scripting, dan integrasi data.
- **JupyterLab**
  Tempat terbaik untuk eksperimen interaktif, dokumentasi, dan pembelajaran bertahap.
- **pandas & NumPy**
  Untuk manipulasi data tabular, *time series*, statistik dasar, dan transformasi numerik.
- **matplotlib**
  Untuk visualisasi dasar yang ringan dan kuat.
- **scikit-learn**
  Untuk model klasifikasi, regresi, *clustering*, dan deteksi anomali.
- **TensorFlow/Keras ringan atau PyTorch ringan**
  Untuk eksplorasi model *deep learning* sederhana bila diperlukan.
- **Mosquitto MQTT**
  Broker ringan untuk komunikasi *publish-subscribe*.
- **Node-RED**
  Untuk orkestrasi alur data dan integrasi cepat antar komponen.
- **SQLite / PostgreSQL / InfluxDB**
  Untuk penyimpanan data, dari sederhana sampai *time-series database*.
- **Grafana**
  Untuk dashboard monitoring.
- **ESP32 / Raspberry Pi**
  Untuk perangkat lapangan dan *gateway* praktis.
- **Linux / Docker opsional**
  Untuk lingkungan yang rapi, stabil, dan mudah direplikasi.

Kenapa stack ini kuat untuk belajar?

Stack ini kuat karena memberi jembatan yang lengkap:

- **murah dan mudah diakses**
- **cukup ringan untuk eksperimen**
- **cukup kuat untuk proyek nyata**
- **didukung komunitas luas**
- **relevan untuk dunia akademik maupun praktik**

Dengan stack ini, pembaca bisa memulai dari laptop biasa, lalu bertahap naik ke eksperimen nyata dengan broker MQTT, dashboard, model ML, sampai *edge inference*.

1.8 Peta Belajar Bab 1 untuk Pembaca

Agar pembaca tidak merasa bab pertama hanya teori, penting untuk menanamkan pola pikir bahwa bab ini adalah **peta jalan**. Setelah membaca bab ini, pembaca diharapkan sudah mengerti empat hal besar:

- **IoT adalah sistem ujung ke ujung**, bukan hanya sensor.
- **Data adalah pusat nilai**, tetapi data mentah belum cukup.
- ***Machine learning* di IoT harus memperhatikan realitas lapangan**, bukan hanya akurasi model.
- **Open source memungkinkan pembaca belajar, bereksperimen, dan membangun sistem secara mandiri.**

Dengan fondasi ini, pembaca akan lebih siap memasuki bab berikutnya tentang arsitektur praktikum, data sensor, *feature engineering*, *dashboard*, *machine learning*, dan integrasi end-to-end yang memang menjadi struktur utama buku Anda.

1.9 Langkah Praktis yang Harus Dilakukan Pembaca Setelah Mempelajari Bab Ini

Supaya bab ini terasa hidup dan tidak berhenti pada pemahaman konsep, berikut tahapan belajar yang sebaiknya dilakukan pembaca.

Tahap 1 — Pahami alur sistem

Gambarlah satu skema sederhana:

> **sensor → gateway → MQTT broker → penyimpanan → analitik Python → model → dashboard / aksi**

Tujuannya bukan agar gambarnya indah, tetapi agar pembaca benar-benar memahami posisi setiap komponen.

Tahap 2 — Tentukan contoh kasus kecil

Pilih satu kasus sederhana, misalnya:

- suhu ruangan,
- kelembaban tanah,
- konsumsi energi harian,
- atau getaran mesin mini.

Jangan langsung memilih sistem yang terlalu rumit. Fokus pada satu aliran data terlebih dahulu.

Tahap 3 — Siapkan lingkungan open source

Pasang:

- Python
- JupyterLab
- pandas
- matplotlib
- scikit-learn
- Mosquitto MQTT

Bila ingin lebih lanjut, tambahkan Node-RED dan Grafana.

Tahap 4 — Simulasikan data pertama

Sebelum memakai sensor sungguhan, buat data *dummy* sederhana:

- `timestamp`
- `device_id`
- `temperature`
- `humidity`

Simpan ke CSV lalu baca dengan Python. Ini akan mempermudah transisi ke bab berikutnya.

Tahap 5 — Melatih cara berpikir berbasis keputusan

Setiap kali melihat data, biasakan bertanya:

- Apa yang sedang terjadi?
- Apakah ini normal?
- Apakah ada pola berulang?
- Apa keputusan yang bisa diambil dari data ini?

Pertanyaan-pertanyaan itu akan membentuk kebiasaan **data-driven thinking** yang menjadi inti dari buku ini.

1.10 Penutup Bab

Bab ini adalah pintu masuk ke dunia yang sangat menarik: dunia ketika **benda fisik mulai berbicara lewat data**, lalu komputer belajar membaca pola dari suara data itu, dan akhirnya membantu manusia mengambil keputusan yang lebih cerdas. IoT memberi kita mata dan telinga digital. Data analytics memberi kita cara memahami apa yang tertangkap. *Machine learning* memberi kita kemampuan untuk melihat pola yang lebih dalam. Dan open source memberi kita kebebasan untuk membangun semuanya secara terbuka, hemat, kolaboratif, dan berkelanjutan. Fokus seperti ini juga konsisten dengan arah revisi buku Anda yang menempatkan praktik Python, *data acquisition*, pembersihan data, visualisasi, ML techniques, serta konteks cloud-edge-fog sebagai inti buku, bukan sekadar teknologi yang bombastis.

Hands-on 1 - Menyiapkan Lingkungan Kerja Python untuk IoT Analytics di Ubuntu 26.04

1. Tujuan Hands-on

Pada latihan ini, pembaca akan menyiapkan **lingkungan kerja Python yang rapi, stabil, dan siap dipakai** untuk seluruh isi buku. Setelah selesai, pembaca diharapkan sudah memiliki:

- **Python 3** yang berjalan baik di Ubuntu 26.04
- **virtual environment** khusus proyek IoT
- paket inti untuk analitik data dan visualisasi
- struktur folder proyek yang bersih
- *notebook* atau skrip Python pertama yang siap dipakai untuk eksperimen data sensor

Tujuan utama latihan ini bukan sekadar “berhasil install,” tetapi membangun kebiasaan kerja profesional sejak awal. Dalam proyek IoT dan *machine learning*, banyak masalah justru muncul karena lingkungan kerja berantakan, versi paket tidak konsisten, atau file proyek tercampur tanpa struktur yang jelas.

2. Contoh use case nyata yang relevan

Sebelum masuk ke terminal, mari pahami mengapa setup ini penting lewat beberapa contoh nyata.

Contoh 1 — Monitoring suhu ruang server kampus

Sebuah kampus ingin memonitor suhu ruang server dengan ESP32 dan sensor suhu. Data dikirim tiap 10 detik, disimpan ke file, lalu divisualisasikan untuk melihat apakah pendingin ruangan bekerja normal. Untuk kasus seperti ini, pembaca perlu:

- Python untuk membaca data
- pandas untuk membersihkan dan mengolah
- matplotlib untuk membuat grafik
- JupyterLab agar eksperimen mudah dilakukan

Contoh 2 — Analitik konsumsi energi laboratorium

Sebuah laboratorium ingin mengetahui jam-jam puncak pemakaian listrik. Data energi direkam per menit, lalu dianalisis harian dan mingguan. Di sini pembaca memerlukan:

- struktur proyek yang rapi
- paket numerik
- *time series processing*
- visualisasi awal sebelum nanti dibawa ke model prediksi

Contoh 3 — Prototipe predictive maintenance sederhana

Sebuah bengkel teknik ingin mempelajari data getaran motor secara sederhana sebelum masuk ke *anomaly detection*. Langkah pertama tetap sama: siapkan lingkungan Python yang stabil dan bisa digunakan berulang kali.

Dengan kata lain, **setup yang baik adalah fondasi semua eksperimen berikutnya**.

3. Prinsip kerja yang dipakai di buku ini

Untuk seluruh buku, kita akan memakai prinsip berikut:

- **Linux Ubuntu 26.04** sebagai sistem operasi utama
- **Python + virtual environment** untuk isolasi dependensi
- **JupyterLab** untuk eksperimen interaktif
- **VS Code atau editor lain** hanya sebagai pelengkap, bukan kewajiban
- semua alat berbasis **open source**
- proyek disusun dengan struktur folder yang mudah dipahami mahasiswa maupun praktisi

4. Langkah instalasi di Ubuntu 26.04

4.1 Update sistem terlebih dahulu

Buka terminal, lalu jalankan:

```
sudo apt update
sudo apt upgrade -y
```

Langkah ini penting agar indeks paket dan dependensi sistem berada dalam kondisi terbaru.

4.2 Pastikan Python dan tool dasar tersedia

```
python3 --version
pip3 --version
```

Bila `pip3` belum tersedia, pasang dengan:

```
sudo apt install -y python3-pip python3-venv
```

Penjelasan:

- `python3-pip` dipakai untuk memasang pustaka Python
- `python3-venv` dipakai untuk membuat *virtual environment*

4.3 Buat folder kerja utama

Misalnya kita akan menaruh semua proyek buku di folder `iot-ml-book` pada home directory:

```
mkdir -p ~/iot-ml-book
cd ~/iot-ml-book
```

Cek isi folder:

```
pwd
ls -la
```

4.4 Buat *virtual environment*

```
python3 -m venv .venv
```

Aktifkan:

```
source .venv/bin/activate
```

Kalau berhasil, prompt terminal biasanya berubah, misalnya menjadi:

```
(.venv) user@ubuntu:~/iot-ml-book$
```

Kenapa ini penting? Karena lingkungan ini akan mengisolasi paket proyek dari paket sistem. Jadi instalasi untuk buku ini tidak “mengotori” Python global Ubuntu.

4.5 Upgrade pip dan pasang pustaka inti

```
pip install --upgrade pip
pip install jupyterlab pandas numpy matplotlib scipy
scikit-learn
```

Untuk praktik awal Bab 1, paket di atas sudah cukup.

Fungsi masing-masing

- **jupyterlab**: lingkungan kerja interaktif
- **pandas**: membaca dan mengolah CSV/tabular data
- **numpy**: operasi numerik
- **matplotlib**: visualisasi dasar

- **scipy**: fungsi ilmiah pendukung
- **scikit-learn**: dipakai di bab ML berikutnya

4.6 Simpan daftar paket

Agar proyek mudah direplikasi di komputer lain:

```
pip freeze > requirements.txt
```

Cek hasil:

```
cat requirements.txt
```

Ini kebiasaan penting dalam proyek profesional.

4.7 Jalankan JupyterLab

```
jupyter lab
```

Biasanya terminal akan menampilkan URL lokal. Buka URL tersebut di browser. Dari sini pembaca bisa membuat:

- *notebook* baru
- file Python
- folder data
- dokumentasi eksperimen

5. Struktur folder proyek yang disarankan

Sejak awal, biasakan struktur proyek seperti ini:

```
iot-ml-book/
├── .venv/
├── data/
│   ├── raw/
│   └── processed/
├── notebooks/
├── scripts/
├── outputs/
│   ├── figures/
│   └── reports/
├── requirements.txt
└── README.md
```

Buat foldernya:

```
mkdir -p data/raw data/processed notebooks scripts
outputs/figures outputs/reports
touch README.md
```

Arti masing-masing folder

- `data/raw/` → data mentah asli
- `data/processed/` → data hasil pembersihan
- `notebooks/` → eksperimen Jupyter
- `scripts/` → skrip Python yang siap dijalankan ulang
- `outputs/figures/` → hasil grafik
- `outputs/reports/` → laporan, ringkasan, atau ekspor hasil

Struktur seperti ini akan sangat membantu saat proyek berkembang menjadi pipeline MQTT, dashboard, dan model ML.

6. Uji lingkungan dengan skrip pertama

Buat file `scripts/test_env.py`:

```
import sys
import pandas as pd
import numpy as np
import matplotlib

print("Python version :", sys.version)
print("pandas version :", pd.__version__)
print("numpy version  :", np.__version__)
print("matplotlib     :", matplotlib.__version__)
```

```
print("\nLingkungan Python IoT Analytics siap
digunakan.")
```

Jalankan:

```
python scripts/test_env.py
```

Contoh output:

```
Python version : 3.13.x ...
pandas version : 2.x.x
numpy version  : 2.x.x
matplotlib     : 3.x.x
```

Lingkungan Python IoT Analytics siap digunakan.

7. Notebook pertama untuk uji kerja

Buat file `notebooks/00_uji_lingkungan.ipynb`, lalu isi salah satu sel dengan:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

print("Notebook aktif dan siap dipakai.")
```

Kalau sel berjalan tanpa error, berarti lingkungan kerja siap.

8. Langkah kerja ringkas yang harus dilakukan pembaca

Tahap	Kegiatan
1	**Update Ubuntu** dan pasang **tool Python dasar**.
2	Buat **folder proyek** dan **virtual environment**.
3	Pasang **pustaka inti**: *JupyterLab*, *pandas*, *NumPy*, *matplotlib*, *scikit-learn*.
4	Buat **struktur folder** yang rapi.
5	Uji lingkungan dengan **skrip** dan **notebook** sederhana.

9. Masalah umum dan solusinya

Masalah	Solusi
`python3 -m venv .venv` gagal	Coba pasang ulang: `sudo apt install -y python3-venv`
pip tidak ditemukan	Pasang: `sudo apt install -y python3-pip`
Jupyter tidak bisa dibuka	Pastikan Anda masih berada dalam virtual environment: `source .venv/bin/activate` lalu jalankan `jupyter lab`
Izin file kacau karena pernah pakai sudo pip	Hindari sudo pip install. Gunakan selalu virtual environment (venv).

10. Penutup Hands-on 1

Setelah latihan ini, pembaca sudah memiliki **fondasi lingkungan kerja Python yang siap untuk analitik IoT**. Ini tampak sederhana, tetapi inilah langkah yang sering menentukan apakah proyek selanjutnya berjalan mulus atau justru penuh masalah. Dalam konteks IoT analytics, data akan terus bertambah, jenis eksperimen akan makin banyak, dan integrasi antar-komponen akan makin kompleks. Karena itu, **lingkungan kerja yang bersih dan reproduktif adalah investasi penting sejak hari pertama**. Kebutuhan ini juga sejalan dengan sifat data IoT yang heterogen, berubah cepat, dan memerlukan visualisasi serta pemrosesan yang konsisten.

Hands-on 2 - Membaca Data Sensor Dummy CSV dan Membuat Visualisasi Pertama

1. Tujuan Hands-on

Latihan ini bertujuan agar pembaca:

- memahami bentuk dasar data sensor IoT
- mampu membaca file CSV dengan Python
- mengenali *timestamp*, *device_id*, dan variabel sensor
- membuat visualisasi awal
- mulai berpikir secara **data-driven**

Hands-on ini sangat penting karena sebelum masuk ke MQTT, broker, *streaming*, atau model *machine learning*, pembaca harus nyaman terlebih dahulu dengan **data sensor sederhana**.

2. Contoh nyata yang bisa diimplementasikan

Agar latihan terasa hidup, berikut tiga contoh implementasi nyata yang sangat dekat dengan dunia kampus, lab, industri kecil, dan komunitas.

Contoh 1 — Smart Room Monitoring

Sensor suhu dan kelembaban ditempatkan di ruang kelas, laboratorium, atau ruang server. Tujuannya:

- memantau kenyamanan
- mendeteksi ruangan terlalu panas
- melihat pengaruh jumlah orang terhadap suhu

Contoh 2 — Smart Greenhouse / urban farming

Sensor suhu, kelembaban udara, dan kelembaban tanah dipakai untuk memantau kondisi tanaman. Tahap awal bisa dimulai dari file CSV, sebelum dihubungkan ke sensor sungguhan.

Contoh 3 — Energi dan peralatan listrik

Data konsumsi listrik atau arus perangkat dikumpulkan berkala. Dengan CSV sederhana, pembaca bisa mulai memahami pola beban harian.

Referensi juga menunjukkan banyak contoh domain seperti **smart water**, **monitoring aliran**, **monitoring kesehatan**, dan **real-time visualization**, yang semuanya berangkat dari alur yang sama: data sensor → preprocess → modeling → visualization.

3. Dataset dummy yang akan dipakai

Kita akan memakai data *dummy* untuk **monitoring ruangan**. Ini realistis karena sangat mudah dibayangkan dan relevan untuk banyak pembaca.

Kolom yang digunakan

- `timestamp` → waktu pembacaan
- `device_id` → identitas sensor
- `temperature_c` → suhu dalam Celsius
- `humidity_pct` → kelembaban relatif
- `co2_ppm` → kadar CO_2 sederhana
- `power_watt` → konsumsi daya perangkat

4. Buat file CSV dummy

Simpan file berikut sebagai:

```
data/raw/room_sensor_dummy.csv
```

Isi file:

```
timestamp,device_id,temperature_c,humidity_pct,co2_ppm,power_watt
2026-01-01 08:00:00,room01,24.5,58,520,110
2026-01-01 08:10:00,room01,24.7,57,525,112
2026-01-01 08:20:00,room01,25.1,56,540,115
2026-01-01 08:30:00,room01,25.6,55,560,118
2026-01-01 08:40:00,room01,26.0,54,590,121
2026-01-01 08:50:00,room01,26.4,53,610,124
2026-01-01 09:00:00,room01,26.9,52,640,128
2026-01-01 09:10:00,room01,27.3,51,670,132
2026-01-01 09:20:00,room01,27.8,50,700,136
2026-01-01 09:30:00,room01,28.2,49,730,141
2026-01-01 09:40:00,room01,28.5,48,760,145
2026-01-01 09:50:00,room01,28.7,47,780,148
2026-01-01 10:00:00,room01,28.9,46,800,150
2026-01-01 10:10:00,room01,29.1,46,815,151
2026-01-01 10:20:00,room01,29.0,47,810,149
2026-01-01 10:30:00,room01,28.8,48,790,147
2026-01-01 10:40:00,room01,28.4,49,760,143
2026-01-01 10:50:00,room01,28.0,50,730,139
2026-01-01 11:00:00,room01,27.6,51,700,134
2026-01-01 11:10:00,room01,27.1,52,670,129
```

Data ini mensimulasikan ruangan yang mulai ramai pada pagi hari:

- suhu naik
- kelembaban turun
- CO_2 naik
- daya listrik naik karena perangkat atau pendingin bekerja lebih keras

Ini adalah contoh kecil dari **hubungan antar-variabel sensor** yang sering muncul di lingkungan nyata.

5. Membaca CSV dengan Python

Buat skrip `scripts/read_dummy_csv.py`:

```
from pathlib import Path
import pandas as pd

# lokasi file
data_path = Path("data/raw/room_sensor_dummy.csv")

# baca CSV
df = pd.read_csv(data_path)

# tampilkan 5 baris pertama
print("=== Lima baris pertama ===")
print(df.head())

print("\n=== Info dataset ===")
print(df.info())

print("\n=== Statistik numerik ===")
print(df.describe())
```

Jalankan:

```
python scripts/read_dummy_csv.py
```

Apa yang dipelajari di sini?

- cara membaca file CSV
- mengenali tipe data
- melihat bentuk dataset
- mendapatkan ringkasan statistik awal

6. Mengubah `timestamp` menjadi format waktu

Dalam data IoT, waktu adalah elemen inti. Jangan membiarkan `timestamp` tetap dianggap string biasa.

Perbarui skrip:

```
from pathlib import Path
import pandas as pd

data_path = Path("data/raw/room_sensor_dummy.csv")
df = pd.read_csv(data_path)

# ubah kolom waktu
df["timestamp"] = pd.to_datetime(df["timestamp"])

print(df.dtypes)
print("\nRentang waktu data:")
print(df["timestamp"].min(), "sampai", df["timestamp"].max())
```

Mengapa ini penting?

Karena begitu `timestamp` menjadi format waktu yang benar, kita bisa:

- memfilter data berdasarkan jam
- membuat grafik time series
- melakukan *resampling*
- mencari pola harian dan mingguan

7. Menjadikan `timestamp` sebagai index

Langkah ini sangat umum dalam analitik IoT.

```
from pathlib import Path
import pandas as pd

data_path = Path("data/raw/room_sensor_dummy.csv")
df = pd.read_csv(data_path)

df["timestamp"] = pd.to_datetime(df["timestamp"])
df = df.set_index("timestamp")

print(df.head())
```

Sekarang dataset siap diperlakukan sebagai **time series** sederhana.

8. Visualisasi pertama dengan matplotlib

Buat file `scripts/plot_dummy_sensor.py`:

```
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt

# baca data
data_path = Path("data/raw/room_sensor_dummy.csv")
df = pd.read_csv(data_path)

# ubah timestamp
df["timestamp"] = pd.to_datetime(df["timestamp"])
df = df.set_index("timestamp")

# buat folder output jika belum ada
output_dir = Path("outputs/figures")
output_dir.mkdir(parents=True, exist_ok=True)

# plot suhu
plt.figure(figsize=(10, 5))
plt.plot(df.index, df["temperature_c"], marker="o")
plt.title("Perubahan Suhu Ruangan")
plt.xlabel("Waktu")
plt.ylabel("Suhu (C)")
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig(output_dir / "plot_suhu_ruangan.png")
plt.show()
```

Jalankan:

```
python scripts/plot_dummy_sensor.py
```

Hasilnya adalah grafik pertama pembaca. Ini sangat penting secara psikologis: dari file CSV biasa, kini data sudah berubah menjadi informasi visual.

9. Visualisasi multi-variabel

Sekarang kita tingkatkan sedikit. Tambahkan suhu, kelembaban, dan CO_2.

```
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt

data_path = Path("data/raw/room_sensor_dummy.csv")
df = pd.read_csv(data_path)

df["timestamp"] = pd.to_datetime(df["timestamp"])
df = df.set_index("timestamp")

output_dir = Path("outputs/figures")
output_dir.mkdir(parents=True, exist_ok=True)

# suhu
plt.figure(figsize=(10, 5))
plt.plot(df.index, df["temperature_c"], marker="o")
plt.title("Suhu Ruangan")
plt.xlabel("Waktu")
plt.ylabel("Suhu (C)")
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig(output_dir / "temperature.png")
plt.show()

# kelembaban
plt.figure(figsize=(10, 5))
plt.plot(df.index, df["humidity_pct"], marker="o")
plt.title("Kelembaban Ruangan")
plt.xlabel("Waktu")
plt.ylabel("kelembaban (%)")
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig(output_dir / "humidity.png")
plt.show()

# CO2
plt.figure(figsize=(10, 5))
plt.plot(df.index, df["co2_ppm"], marker="o")
plt.title("Kadar CO2 Ruangan")
plt.xlabel("Waktu")
plt.ylabel("CO2 (ppm)")
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig(output_dir / "co2.png")
plt.show()
```

Interpretasi awal

Dari grafik tersebut, pembaca bisa melihat:

- suhu cenderung naik saat ruangan aktif
- kelembaban cenderung turun saat suhu naik
- CO_2 meningkat ketika aktivitas manusia meningkat

Ini contoh yang sangat realistis untuk **monitoring ruang kelas, ruang kerja, laboratorium, atau ruang server**.

10. Visualisasi relasi antar-variabel

Tidak semua analitik IoT harus langsung kompleks. Kadang relasi sederhana antar-variabel sudah sangat bermanfaat.

Buat plot suhu vs CO_2:

```
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt

data_path = Path("data/raw/room_sensor_dummy.csv")
df = pd.read_csv(data_path)

output_dir = Path("outputs/figures")
output_dir.mkdir(parents=True, exist_ok=True)

plt.figure(figsize=(7, 5))
plt.scatter(df["temperature_c"], df["co2_ppm"])
plt.title("Relasi Suhu dan CO2")
plt.xlabel("Suhu (C)")
plt.ylabel("CO2 (ppm)")
plt.tight_layout()
plt.savefig(output_dir / "temperature_vs_co2.png")
plt.show()
```

Mengapa ini berguna?

Karena dalam banyak kasus IoT, pembaca ingin menjawab pertanyaan seperti:

- apakah kenaikan suhu berkaitan dengan kepadatan ruangan?
- apakah ventilasi yang buruk ikut meningkatkan CO_2?
- apakah beban listrik naik saat suhu meningkat?

Inilah awal dari **berpikir analitik**, sebelum masuk ke model ML.

11. Membuat ringkasan otomatis

Tambahkan skrip sederhana untuk membuat insight awal.

```
from pathlib import Path
import pandas as pd

data_path = Path("data/raw/room_sensor_dummy.csv")
df = pd.read_csv(data_path)

df["timestamp"] = pd.to_datetime(df["timestamp"])
df = df.set_index("timestamp")

print("=== Ringkasan Analitik Awal ===")
print(f"Suhu minimum   : {df['temperature_c'].min():.1f} C")
print(f"Suhu maksimum  : {df['temperature_c'].max():.1f} C")
print(f"Rata-rata suhu : {df['temperature_c'].mean():.2f} C")

print(f"\nCO2 minimum    : {df['co2_ppm'].min()} ppm")
print(f"CO2 maksimum   : {df['co2_ppm'].max()} ppm")
print(f"Rata-rata CO2  : {df['co2_ppm'].mean():.2f} ppm")

print(f"\nDaya minimum   : {df['power_watt'].min()} watt")
print(f"Daya maksimum  : {df['power_watt'].max()} watt")
print(f"Rata-rata daya : {df['power_watt'].mean():.2f} watt")
```

Output seperti ini membantu pembaca mulai memahami bagaimana data sensor diterjemahkan menjadi informasi operasional.

12. Versi notebook yang cocok untuk mahasiswa

Di JupyterLab, pembaca bisa memecah latihan menjadi sel-sel kecil:

Sel 1

```
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path
```

Sel 2

```
df = pd.read_csv("data/raw/room_sensor_dummy.csv")
df.head()
```

Sel 3

```
df["timestamp"] = pd.to_datetime(df["timestamp"])
df = df.set_index("timestamp")
df.info()
```

Sel 4

```
df.describe()
```

Sel 5

```
plt.figure(figsize=(10, 5))
plt.plot(df.index, df["temperature_c"], marker="o")
plt.title("Perubahan Suhu Ruangan")
plt.xlabel("Waktu")
plt.ylabel("Suhu (C)")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Dengan model seperti ini, mahasiswa bisa belajar langkah demi langkah tanpa takut gagal.

13. Langkah-langkah kerja yang harus dilakukan pembaca

Tahap 1 — Siapkan file data

Buat file CSV *dummy* di folder `data/raw/`.

Tahap 2 — Baca data dengan pandas

Pastikan file bisa dibuka dan tampil 5 baris pertama.

Tahap 3 — Ubah kolom waktu

Konversi `timestamp` menjadi `datetime`.

Tahap 4 — Jadikan data sebagai time series

Set `timestamp` sebagai index.

Tahap 5 — Buat grafik pertama

Visualisasikan suhu ruangan.

Tahap 6 — Tambahkan variabel lain

Plot kelembaban, CO_2, dan daya.

Tahap 7 — Tarik insight awal

Apa pola yang terlihat? Apa keputusan yang mungkin diambil?

14. Insight nyata yang bisa ditindaklanjuti

Dari dataset sederhana tadi, sistem nyata dapat berkembang menjadi:

- **Alarm ventilasi** bila CO_2 naik terlalu cepat
- **Peringatan suhu ruang server** bila suhu melewati ambang batas
- **Optimasi energi** bila daya meningkat tanpa alasan jelas
- **Prediksi kenyamanan ruang** dengan ML sederhana pada bab berikutnya

Ini sejalan dengan berbagai use case yang sering muncul dalam literatur IoT analytics, misalnya monitoring air, kesehatan, energi, dan sistem real-time, yang semuanya mengandalkan visualisasi dan analitik awal sebelum melangkah ke model prediksi.

15. Variasi latihan lanjutan yang bisa dicoba

Agar pembaca tertantang namun tetap fun, berikan tantangan kecil berikut:

- ganti `room01` menjadi beberapa ruangan: `room01`, `room02`, `lab01`
- tambahkan data hilang
- tambahkan nilai anomali ekstrem
- buat grafik daya listrik
- hitung korelasi antar-variabel
- cari waktu suhu tertinggi
- simpan hasil pembersihan ke `data/processed/`

Contoh menyimpan data hasil olahan:

```
from pathlib import Path
import pandas as pd

data_path = Path("data/raw/room_sensor_dummy.csv")
output_path =
Path("data/processed/room_sensor_clean.csv")

df = pd.read_csv(data_path)
df["timestamp"] = pd.to_datetime(df["timestamp"])

output_path.parent.mkdir(parents=True, exist_ok=True)
df.to_csv(output_path, index=False)

print(f"Data bersih disimpan ke: {output_path}")
```

16. Penutup Hands-on 2

Latihan ini tampak sederhana, tetapi sesungguhnya sangat penting. Di sinilah pembaca mulai belajar bahwa **data sensor bukan hanya angka**, melainkan bahan mentah untuk memahami kondisi nyata. Dengan CSV sederhana, pembaca sudah bisa:

- membaca data sensor,
- mengenali dimensi waktu,
- membuat grafik,
- dan menarik insight awal.

Kebiasaan kecil ini akan menjadi fondasi untuk semua tahap berikutnya: MQTT, streaming, penyimpanan, feature engineering, anomaly detection, forecasting, sampai dashboard real-time. Dalam konteks IoT analytics, **kemampuan membaca dan memvisualisasikan data awal dengan benar sering jauh lebih berharga daripada langsung mengejar model yang rumit**. Hal ini juga sejalan dengan pandangan bahwa banyak pengguna justru paling diuntungkan oleh solusi sensor analytics yang sederhana, jelas, dan berguna di titik penggunaan lokal dalam atau mendekati waktu nyata.

Bab 2. Arsitektur IoT Open Source untuk Praktikum

2.1 Mengapa arsitektur IoT perlu dipahami sejak awal?

Banyak pemula melihat IoT hanya sebagai “sensor yang kirim data.” Padahal, pada sistem nyata, data harus melewati beberapa lapisan: perangkat pembaca, penghubung jaringan, mekanisme pengiriman pesan, penyimpanan, analitik, lalu tampilan atau aksi. Karena itu, Bab 2 sangat penting: pembaca perlu memahami **siapa melakukan apa** dalam satu sistem IoT, agar saat nanti masuk ke *hands-on* MQTT, Python subscriber, database, dan dashboard, semua komponen terasa masuk akal, bukan sekadar susunan alat yang berdiri sendiri. Ini konsisten dengan arah buku yang ingin membawa pembaca cepat dari **sensor → data → model → inferensi → dashboard**.

Literatur yang Anda lampirkan juga menunjukkan bahwa arsitektur IoT umumnya dibangun secara berlapis. Ada model **tiga lapis** seperti **perception–network–application**, ada juga model yang menambahkan **middleware**, **access gateway**, atau **edge technology** untuk menjelaskan bagaimana data bergerak dari perangkat fisik sampai layanan aplikasi.

2.2 Komponen inti: sensor, mikrokontroler, gateway, broker, database, dashboard

Untuk kebutuhan buku ini, kita tidak perlu memulai dari arsitektur paling rumit. Yang kita butuhkan adalah **arsitektur minimal tetapi lengkap**, sehingga pembaca bisa membangun sistem ujung-ke-ujung dengan biaya masuk yang rendah dan logika yang mudah dipahami.

2.2.1 Sensor

Sensor adalah komponen yang membaca kondisi fisik dunia nyata. Sensor dapat mengukur suhu, , cahaya, getaran, arus listrik, tekanan, posisi, kadar gas, atau variabel lain. Dalam arsitektur IoT, sensor berada di lapisan paling bawah, yaitu lapisan yang berinteraksi langsung dengan lingkungan. Literatur menyebut lapisan ini sebagai *perception layer*, tempat data dikumpulkan dari dunia fisik.

Masalah	**Solusi**
`python3 -m venv .venv` gagal	Coba pasang ulang: `sudo apt install -y python3-venv`
pip tidak ditemukan	Pasang: `sudo apt install -y python3-pip`
Jupyter tidak bisa dibuka	Pastikan Anda masih berada dalam virtual environment: `source .venv/bin/activate` lalu jalankan `jupyter lab`
Izin file kacau karena pernah pakai sudo pip	Hindari sudo pip install. Gunakan selalu virtual environment (venv).

Contoh implementasi nyata:

- sensor suhu dan kelembaban di ruang server
- sensor kelembaban tanah di kebun atau greenhouse
- sensor arus listrik di laboratorium
- sensor getaran pada motor kecil untuk simulasi *predictive maintenance*

2.2.2 Mikrokontroler

Mikrokontroler adalah otak kecil di sisi perangkat. Ia membaca sensor, memformat data, lalu mengirimkannya ke jaringan. Pada praktikum, pilihan paling masuk akal adalah **ESP32**, karena murah, banyak komunitas, mendukung Wi-Fi, dan cukup kuat untuk eksperimen awal. Dalam konteks arsitektur, perangkat semacam ini bisa dipandang sebagai pengendali lokal yang menjembatani sensor dengan sistem yang lebih besar. Literatur Industry 4.0 juga menyebut *embedded system* sebagai bagian penting yang dapat memuat **microchip, gateway, dan sensor/mote** untuk mendukung komunikasi ujung-ke-ujung dan interoperabilitas.

2.2.3 Gateway

Gateway adalah penghubung antara perangkat lapangan dengan sistem jaringan yang lebih tinggi. Tidak semua sensor perlu bicara langsung ke server. Dalam banyak desain, gateway bertugas:

- mengumpulkan data dari beberapa perangkat,
- menerjemahkan protokol,
- memberi *timestamp*,
- melakukan penyaringan awal,
- dan meneruskan data ke broker atau database.

Dalam banyak referensi IoT, **gateway** muncul sebagai bagian penting di lapisan jaringan atau lapisan akses, terutama ketika data harus berpindah dari sensor lokal ke Internet, cloud, atau sistem analitik.

Pada sistem praktikum, gateway bisa berupa:

- **Raspberry Pi**
- mini PC Ubuntu
- laptop Ubuntu
- atau bahkan **ESP32** itu sendiri untuk skenario sederhana

2.2.4 Broker

Untuk buku ini, komponen sentral komunikasi adalah **broker MQTT**. MQTT relevan untuk IoT karena **ringan, sederhana, terbuka, dan dirancang untuk perangkat dengan sumber daya terbatas serta jaringan minimal**. MQTT memakai model *publish/subscribe*: perangkat mengirim pesan ke topik tertentu, lalu broker

mendistribusikannya ke subscriber yang tepat. Dalam model ini, broker berperan sebagai koordinator komunikasi antara publisher dan subscriber.

Ini sangat cocok untuk praktikum karena:

- sensor tidak perlu tahu siapa yang membaca data,
- dashboard bisa ditambahkan tanpa mengubah sensor,
- Python subscriber bisa dipasang belakangan,
- sistem mudah diperluas.

2.2.5 Database

Setelah data masuk, data perlu disimpan. Untuk pembelajaran, kita tidak perlu langsung memakai *distributed data platform*. Pilihan yang cukup untuk buku ini:

- **CSV** untuk latihan awal,
- **SQLite** untuk penyimpanan ringan dan lokal,
- **InfluxDB** untuk *time-series database* yang lebih cocok untuk data sensor.

Peran database adalah:

- menyimpan histori,
- mendukung visualisasi,
- menjadi bahan analitik Python,
- menjadi dataset pelatihan model.

2.2.6 Dashboard

Dashboard adalah wajah dari sistem. Inilah titik di mana manusia melihat hasil kerja IoT. Untuk open source, pilihan paling praktis adalah **Grafana**. Dengan dashboard, pembaca bisa melihat:

- suhu naik turun,
- sensor mana yang offline,
- kapan daya listrik melonjak,
- kapan anomali muncul.

Dashboard penting karena IoT tidak cukup hanya “mengumpulkan data.” Nilai sistem muncul ketika data dapat dibaca, dipahami, dan ditindaklanjuti.

2.3 Protokol yang cukup untuk buku: MQTT, HTTP REST, serial

Buku ini sengaja membatasi protokol agar pembaca tidak tenggelam dalam terlalu banyak teknologi sekaligus. Untuk kebutuhan praktikum, **tiga protokol sudah cukup**.

MQTT

Ini protokol utama untuk komunikasi IoT di buku ini. MQTT unggul untuk skenario:

- data berkala dari sensor,
- perangkat ringan,
- komunikasi banyak-ke-banyak,
- sistem yang modular.

Contoh penggunaan:

- ESP32 mengirim suhu ke topik `kampus/lab1/suhu`
- Python subscriber membaca topik itu
- Node-RED ikut berlangganan
- Grafana menampilkan data yang sudah disimpan

HTTP REST

HTTP REST berguna saat kita ingin:

- membuat API sederhana,
- mengambil data dari web service,
- mengirim konfigurasi,
- atau mengintegrasikan Node-RED / Python dengan aplikasi lain.

Dari referensi handbook, **REST** dan **RESTful** jelas termasuk komponen penting dalam ekosistem layanan IoT dan data.

Contoh penggunaan nyata:

- Python mengirim hasil prediksi ke API lokal
- Node-RED menerima data lewat HTTP endpoint
- dashboard atau aplikasi web meminta data history dari server

Serial

Komunikasi **serial** tetap penting untuk praktikum karena sering menjadi jalur pertama antara sensor/mikrokontroler dan komputer. Saat belum memakai Wi-Fi atau saat ingin debugging, serial sangat membantu.

Contoh penggunaan:

- ESP32 mengirim data CSV lewat USB serial
- Python membaca data serial

- data lalu diteruskan ke MQTT atau disimpan ke SQLite

Dengan kata lain:

- **serial** cocok untuk tahap awal dan debugging,
- **MQTT** cocok untuk alur IoT utama,
- **HTTP REST** cocok untuk integrasi aplikasi.

2.4 Peran *edge*, *fog*, dan *cloud* secara praktis

Dalam banyak diskusi, istilah *edge*, *fog*, dan *cloud* terdengar abstrak. Untuk buku ini, kita sederhanakan secara praktis.

Edge

Edge berarti pemrosesan dilakukan dekat sumber data. Literatur menekankan bahwa edge memproses data **di tempat data dihasilkan**, agar hasil bisa lebih cepat dan lebih relevan untuk aplikasi real-time.

Contoh:

- ESP32 menyaring data sebelum dikirim
- Raspberry Pi menghitung rata-rata 1 menit
- gateway lokal mendeteksi anomali awal

Kelebihan:

- latensi rendah
- hemat bandwidth
- tetap jalan walau Internet buruk

Fog

Fog berada di tengah antara perangkat lapangan dan cloud. Literatur menggambarkan fog sebagai arsitektur komputasi terdistribusi yang menempatkan komputasi, storage, dan layanan **di antara sumber data dan cloud**.

Contoh:

- Raspberry Pi lab menerima data dari banyak ESP32
- Pi menjalankan Mosquitto, Node-RED, dan SQLite
- sebagian data penting saja diteruskan ke server pusat

Fog cocok untuk:

- kampus,
- pabrik kecil,
- gedung,
- komunitas,

- sekolah yang ingin server lokal dulu

Cloud

Cloud dipakai untuk penyimpanan besar, analitik berat, integrasi banyak lokasi, atau akses dari luar. Tetapi untuk buku ini, cloud bukan pusat pembelajaran awal. Kita cukup pahami bahwa cloud berguna saat:

- skala data membesar,
- banyak gateway perlu dikonsolidasikan,
- model ML dilatih terpusat,
- dashboard harus diakses dari banyak lokasi.

Arsitektur IoT–Fog–Cloud dalam referensi menunjukkan bahwa gateway berperan memutuskan beban mana yang diproses lebih dekat dan mana yang diteruskan ke lapisan lebih tinggi, terutama karena perangkat IoT memiliki keterbatasan memori dan baterai.

2.5 Arsitektur minimalis berbasis open source

Untuk buku ini, arsitektur minimalis yang paling cocok adalah:

> **Sensor / simulator → ESP32 / Python generator → Mosquitto MQTT → Python subscriber / Node-RED → SQLite atau InfluxDB → Grafana**

Struktur ini kuat karena:

- mudah dipasang di Ubuntu,
- tidak bergantung layanan proprietary,
- bisa dijalankan lokal,
- cukup realistis untuk dunia nyata,
- mudah diperluas ke ML pada bab berikutnya.

Versi 1 — Paling sederhana

- sensor disimulasikan dengan Python
- Python publish ke MQTT
- Python lain subscribe
- hasil simpan ke CSV/SQLite

Cocok untuk:

- mahasiswa baru
- kelas awal
- demo cepat

Versi 2 — Praktikum kampus/lab

- ESP32 membaca sensor
- kirim ke Mosquitto di laptop Ubuntu / Raspberry Pi
- Node-RED memproses alur
- simpan ke InfluxDB
- tampilkan di Grafana

Cocok untuk:

- laboratorium
- tugas akhir awal
- workshop

Versi 3 — Menuju sistem cerdas

- data sensor masuk ke broker
- Python subscriber melakukan preprocessing
- model ML memberi skor anomali / prediksi
- hasil prediksi dikirim kembali ke database / dashboard
- notifikasi sederhana dibuat jika ambang terlampaui

Ini sudah sejalan dengan arah buku Anda yang nantinya masuk ke **pipeline IoT-ML end-to-end**.

2.6 Langkah berpikir yang harus dilakukan pembaca

Saat merancang arsitektur praktikum, biasakan pembaca bertanya:

- **apa yang dibaca sensor?**
- **siapa yang mengirim data?**
- **ke mana data pertama kali masuk?**
- **di mana data disimpan?**
- **siapa yang menganalisis?**
- **siapa yang melihat hasil?**
- **apa aksi yang akan terjadi jika kondisi tertentu muncul?**

Pertanyaan ini membentuk pola pikir sistem, bukan hanya pola pikir perangkat.

2.7 Rekomendasi praktis untuk praktikum Ubuntu

Untuk bab ini, lingkungan yang paling aman dan sederhana adalah:

- **Ubuntu** sebagai OS utama
- **Mosquitto** sebagai broker MQTT
- **Python** untuk subscriber dan analitik
- **Node-RED** untuk orkestrasi alur
- **SQLite** untuk awal, **InfluxDB** untuk *time series*
- **Grafana** untuk dashboard
- **ESP32** atau **simulator Python** sebagai penghasil data

2.8 Penutup bab

Bab 2 menanamkan satu ide besar: sistem IoT yang baik bukan kumpulan alat, melainkan arsitektur kerja yang utuh. Sensor membaca dunia fisik, mikrokontroler mengemas data, gateway menjembatani koneksi, broker mengalirkan pesan, database menyimpan histori, dashboard memvisualisasikan kondisi, dan lapisan analitik memberi makna pada data. Dengan memahami struktur ini sejak awal, pembaca akan jauh lebih siap masuk ke *hands-on* berikutnya: **mengirim data ke MQTT, membaca data dengan Python, menyimpan ke database, lalu menampilkan hasil di dashboard**. Itulah jalur praktis dari **perangkat → data → keputusan cerdas** yang menjadi jiwa utama buku ini.

Hands-on 3 - Mengirim Data Sensor ke MQTT

1. Tujuan Hands-on

Pada latihan ini, pembaca akan belajar mengirim data sensor ke broker MQTT menggunakan **Python** di **Ubuntu 26.04**. Setelah selesai, pembaca diharapkan mampu:

- memasang dan menjalankan **Mosquitto MQTT broker**
- memahami konsep **topic**, **publisher**, **subscriber**, dan **QoS**
- membuat **simulator sensor** sederhana dengan Python
- mengirim data sensor dalam format **JSON**
- menguji bahwa pesan benar-benar masuk ke broker

Hands-on ini penting karena di sinilah sistem mulai berubah dari sekadar file CSV lokal menjadi **aliran data IoT yang hidup**.

2. Contoh implementasi nyata yang relevan

Sebelum menulis kode, mari lihat tiga contoh implementasi yang sangat realistis.

Contoh 1 — Monitoring suhu ruang server

ESP32 atau simulator Python membaca suhu dan kelembaban lalu mengirimkannya setiap 10 detik ke broker MQTT di mini PC Ubuntu. Data tersebut kemudian dibaca oleh Python subscriber untuk alarm suhu berlebih.

Contoh 2 — Greenhouse / urban farming

Sensor kelembaban tanah, suhu udara, dan intensitas cahaya dikirim ke MQTT. Sistem lokal lalu menentukan apakah pompa irigasi perlu dinyalakan.

Contoh 3 — Monitoring konsumsi energi laboratorium

Perangkat pengukur arus atau daya listrik mengirimkan pembacaan berkala ke broker. Data disimpan untuk analitik beban puncak dan efisiensi penggunaan listrik.

Literatur juga menunjukkan contoh domain seperti **precision agriculture**, **smart water metering**, dan **monitoring lingkungan**, yang seluruhnya bergantung pada aliran data sensor menuju pemrosesan dan pengambilan keputusan.

3. Arsitektur yang dipakai

Untuk latihan ini, kita gunakan arsitektur berikut:

> **Python sensor simulator → Mosquitto MQTT broker → subscriber lain / database / dashboard**

Ini konsisten dengan arsitektur hub-and-spoke MQTT yang berfokus pada **telemetry**, yaitu pengumpulan data dari banyak perangkat menuju infrastruktur TI melalui satu *data concentrator server* atau broker.

4. Instalasi Mosquitto di Ubuntu 26.04

Pasang broker dan klien uji:

```
sudo apt update
sudo apt install -y mosquitto mosquitto-clients
```

Aktifkan agar berjalan otomatis:

```
sudo systemctl enable mosquitto
sudo systemctl start mosquitto
sudo systemctl status mosquitto
```

Cek apakah broker aktif di port default **1883**:

```
ss -ltnp | grep 1883
```

Uji cepat dengan terminal

Buka terminal pertama:

```
mosquitto_sub -h localhost -t "kampus/lab1/sensor/#" -v
```

Buka terminal kedua:

```
mosquitto_pub -h localhost -t "kampus/lab1/sensor/suhu"
-m "27.5"
```

Jika berhasil, terminal pertama akan menampilkan pesan yang dipublikasikan. Ini membuktikan bahwa broker sudah berjalan.

5. Siapkan pustaka Python

Aktifkan *virtual environment* proyek, lalu pasang pustaka MQTT:

```
source .venv/bin/activate
pip install paho-mqtt
pip freeze > requirements.txt
```

6. Struktur folder yang disarankan

Tambahkan folder berikut:

```
mkdir -p scripts/mqtt data/raw outputs/logs
```

7. Topik MQTT yang akan dipakai

Agar rapi, gunakan pola topik seperti ini:

- `kampus/lab1/room01/telemetry`
- `kampus/lab1/room01/status`
- `kampus/lab1/room01/alert`

Untuk latihan awal, kita fokus pada:

- **telemetry** → data sensor utama
- **status** → status perangkat
- **alert** → peringatan jika ada kondisi tidak normal

8. Format data yang dikirim

Kita kirim data dalam format **JSON** agar mudah dibaca Python dan mudah diperluas.

Contoh payload:

```
{
  "device_id": "room01",
  "temperature_c": 27.8,
  "humidity_pct": 54.1,
  "co2_ppm": 640,
  "power_watt": 132,
  "timestamp": "2026-01-01T09:20:00"
}
```

9. Source code utama: publisher / simulator sensor Python

Buat file `scripts/mqtt/publish_sensor.py`:

```
import json
import random
import time
from datetime import datetime

import paho.mqtt.client as mqtt

BROKER_HOST = "localhost"
BROKER_PORT = 1883
TOPIC = "kampus/lab1/room01/telemetry"
CLIENT_ID = "publisher-room01"

client = mqtt.Client(client_id=CLIENT_ID)

def connect():
    client.connect(BROKER_HOST, BROKER_PORT, keepalive=60)

def generate_sensor_data():
    temperature = round(random.uniform(24.0, 30.0), 2)
    humidity = round(random.uniform(45.0, 65.0), 2)
    co2 = random.randint(450, 900)
    power = round(random.uniform(100.0, 160.0), 2)

    payload = {
        "device_id": "room01",
        "temperature_c": temperature,
        "humidity_pct": humidity,
        "co2_ppm": co2,
        "power_watt": power,
        "timestamp": datetime.now().isoformat(timespec="seconds")
    }
    return payload

def main():
    connect()
    print("Terhubung ke broker MQTT.")

    while True:
        payload = generate_sensor_data()
        payload_json = json.dumps(payload)

        result = client.publish(TOPIC, payload_json, qos=1)
        status = result.rc

        if status == 0:
            print(f"Berhasil publish ke {TOPIC}: {payload_json}")
        else:
            print(f"Gagal publish, kode status: {status}")

        time.sleep(5)

if __name__ == "__main__":
    main()
```

Jalankan:

```
python scripts/mqtt/publish_sensor.py
```

Apa yang dilakukan kode ini?

- membuat klien MQTT
- terhubung ke broker lokal
- membangkitkan data sensor sintetis
- mengubah data menjadi JSON
- mengirim data ke topik tertentu setiap 5 detik

10. Variasi realistis: publisher untuk sensor greenhouse

Buat file `scripts/mqtt/publish_greenhouse.py`:

```
import json
import random
import time
from datetime import datetime

import paho.mqtt.client as mqtt

BROKER_HOST = "localhost"
BROKER_PORT = 1883
TOPIC = "kampus/greenhouse/zone1/telemetry"
CLIENT_ID = "publisher-greenhouse-zone1"

client = mqtt.Client(client_id=CLIENT_ID)

def generate_data():
    return {
        "device_id": "greenhouse-zone1",
        "air_temp_c": round(random.uniform(25.0, 35.0), 2),
        "soil_moisture_pct": round(random.uniform(30.0, 80.0), 2),
        "light_lux": random.randint(8000, 30000),
        "timestamp": datetime.now().isoformat(timespec="seconds")
    }

client.connect(BROKER_HOST, BROKER_PORT, keepalive=60)

while True:
    payload = generate_data()
    payload_json = json.dumps(payload)
    client.publish(TOPIC, payload_json, qos=1)
    print(f"[GREENHOUSE] {payload_json}")
    time.sleep(10)
```

Ini relevan untuk kasus pertanian presisi, di mana sensor tanah dan cuaca lokal dipakai untuk optimasi irigasi dan pemantauan kondisi tanaman.

11. Langkah kerja yang harus dilakukan pembaca

Tahap	Kegiatan
1	Pasang dan aktifkan Mosquitto di Ubuntu.
2	Uji broker dengan mosquitto_pub dan mosquitto_sub.
3	Pasang paho-mqtt di virtual environment.
4	Buat publisher Python sederhana.
5	Kirim data JSON secara periodik.
6	Amati pesan yang masuk dengan tool terminal atau subscriber Python.

12. Tantangan kecil yang menantang

Agar latihan terasa fun tetapi tetap menantang, pembaca bisa mencoba:

- ganti `localhost` dengan IP Raspberry Pi atau server Ubuntu lain
- ubah topik menjadi beberapa ruangan: `room01`, `room02`, `room03`
- kirim data dengan interval berbeda
- tambahkan field `battery_pct`
- kirim status perangkat ke topik `status`
- buat satu publisher untuk suhu, satu lagi untuk energi

13. Penutup Hands-on 3

Pada titik ini, pembaca sudah berhasil membangun **jalur telemetry IoT yang sesungguhnya**. Data tidak lagi diam di CSV, melainkan mengalir melalui broker yang bisa dilihat, diuji, dan diintegrasikan dengan komponen lain. Inilah fondasi penting sebelum masuk ke subscriber Python dan penyimpanan data. Kekuatan pendekatan MQTT terletak pada kesederhanaannya: **ringan, terbuka, mudah diimplementasikan, dan cocok untuk perangkat terbatas**, sehingga sangat pas untuk praktikum berbasis open source.

Hands-on 4 - Subscribe Data MQTT dengan Python

1. Tujuan Hands-on

Pada latihan ini, pembaca akan membuat **subscriber Python** untuk menerima data dari broker MQTT, menampilkan payload, memeriksa isi JSON, dan menyiapkan data agar siap disimpan ke file atau database.

Setelah selesai, pembaca diharapkan mampu:

- terhubung ke broker sebagai subscriber
- berlangganan ke satu atau beberapa topik
- menerima pesan real-time
- mengubah payload JSON menjadi objek Python
- melakukan validasi awal data

2. Mengapa subscriber Python penting?

Dalam sistem IoT open source, subscriber Python sering menjadi jembatan menuju:

- pembersihan data
- penyimpanan ke database
- *feature engineering*
- deteksi anomali
- prediksi ML
- notifikasi otomatis

3. Source code utama: subscriber Python sederhana

Buat file `scripts/mqtt/subscribe_sensor.py`:

```
import json

import paho.mqtt.client as mqtt

BROKER_HOST = "localhost"
BROKER_PORT = 1883
TOPIC = "kampus/lab1/room01/telemetry"
CLIENT_ID = "subscriber-room01"

def on_connect(client, userdata, flags, reason_code, properties=None):
    print("Terhubung ke broker MQTT.")
    client.subscribe(TOPIC, qos=1)
    print(f"Subscribe ke topik: {TOPIC}")

def on_message(client, userdata, msg):
    print("\n=== Pesan baru diterima ===")
    print(f"Topik   : {msg.topic}")
    print(f"QoS     : {msg.qos}")
    print(f"Payload : {msg.payload.decode()}")

    try:
        data = json.loads(msg.payload.decode())
        print("Payload JSON valid.")
        print(f"Device ID     : {data.get('device_id')}")
        print(f"Temperature C : {data.get('temperature_c')}")
        print(f"Humidity %    : {data.get('humidity_pct')}")
        print(f"CO2 ppm       : {data.get('co2_ppm')}")
        print(f"Power watt    : {data.get('power_watt')}")
        print(f"Timestamp     : {data.get('timestamp')}")
    except json.JSONDecodeError:
        print("Payload bukan JSON yang valid.")

def main():
    client = mqtt.Client(client_id=CLIENT_ID)
    client.on_connect = on_connect
    client.on_message = on_message

    client.connect(BROKER_HOST, BROKER_PORT, keepalive=60)
    client.loop_forever()

if __name__ == "__main__":
    main()
```

Jalankan:

```
python scripts/mqtt/subscribe_sensor.py
```

Biarkan publisher dari Hands-on 3 berjalan. Subscriber akan menerima pesan secara real-time.

4. Subscriber wildcard untuk banyak perangkat

Pada implementasi nyata, kita sering ingin membaca banyak perangkat sekaligus. MQTT mendukung *wildcard*.

Buat file `scripts/mqtt/subscribe_all_rooms.py`:

```
import json

import paho.mqtt.client as mqtt

BROKER_HOST = "localhost"
BROKER_PORT = 1883
TOPIC = "kampus/lab1/+/telemetry"
CLIENT_ID = "subscriber-all-lab1"

def on_connect(client, userdata, flags, reason_code,
properties=None):
    print("Terhubung ke broker.")
    client.subscribe(TOPIC, qos=1)
    print(f"Subscribe wildcard ke: {TOPIC}")

def on_message(client, userdata, msg):
    try:
        payload = msg.payload.decode()
        data = json.loads(payload)
        print(f"[{msg.topic}] {data}")
    except Exception as exc:
        print(f"Gagal memproses pesan: {exc}")

client = mqtt.Client(client_id=CLIENT_ID)
client.on_connect = on_connect
client.on_message = on_message

client.connect(BROKER_HOST, BROKER_PORT, keepalive=60)
client.loop_forever()
```

Topik `kampus/lab1/+/telemetry` akan membaca:

- `kampus/lab1/room01/telemetry`
- `kampus/lab1/room02/telemetry`
- `kampus/lab1/room03/telemetry`

Ini sangat berguna untuk **multi-ruang**, **multi-mesin**, atau **multi-zona greenhouse**.

5. Validasi data di subscriber

Sebelum data disimpan, biasakan pembaca melakukan validasi sederhana. Ini penting karena data IoT sering heterogen, berubah cepat, dan rentan masalah kualitas. Referensi juga menekankan tantangan **heterogeneity**, **correctness of data**, dan visualisasi pada data IoT.

Contoh subscriber dengan validasi nilai:

```
import json

import paho.mqtt.client as mqtt

BROKER_HOST = "localhost"
BROKER_PORT = 1883
TOPIC = "kampus/lab1/room01/telemetry"

def is_valid(data: dict) -> bool:
    temp = data.get("temperature_c")
    hum = data.get("humidity_pct")
    co2 = data.get("co2_ppm")

    if temp is None or hum is None or co2 is None:
        return False

    if not (0 <= temp <= 60):
        return False
    if not (0 <= hum <= 100):
        return False
    if not (300 <= co2 <= 5000):
        return False

    return True

def on_connect(client, userdata, flags, reason_code, properties=None):
    client.subscribe(TOPIC, qos=1)

def on_message(client, userdata, msg):
    try:
        data = json.loads(msg.payload.decode())
        if is_valid(data):
            print(f"Data valid: {data}")
        else:
            print(f"Data tidak valid: {data}")
    except Exception as exc:
        print(f"Error parsing payload: {exc}")

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect(BROKER_HOST, BROKER_PORT, 60)
client.loop_forever()
```

6. Contoh implementasi real yang bisa dipakai

Kasus ruang kelas

Subscriber Python menerima data dari beberapa ruangan, lalu menentukan ruangan mana yang terlalu panas untuk kuliah sore.

Kasus bengkel mesin

Subscriber menerima getaran mesin dari beberapa motor. Jika salah satu topik menunjukkan lonjakan RMS getaran, sistem memberi label *warning*.

Kasus smart water

Subscriber membaca data debit air dan status katup dari beberapa jalur distribusi. Jika laju aliran melebihi ambang, sistem menandai kemungkinan kebocoran atau konsumsi abnormal. Literatur tentang smart water juga menunjukkan alur **sensor** → **preprocess** → **modelling** → **storage** → **visualization** sebagai kerangka yang relevan.

7. Langkah kerja yang harus dilakukan pembaca

Tahap	Kegiatan
1	Jalankan **broker**.
2	Jalankan **publisher**.
3	Jalankan **subscriber** sederhana.
4	Ubah **subscriber** agar memakai **wildcard**.
5	Tambahkan **validasi payload**.
6	Siapkan **subscriber** untuk tahap **penyimpanan data**.

8. Penutup Hands-on 4

Dengan subscriber Python, pembaca sekarang sudah memiliki **mata digital** yang membaca arus telemetry MQTT secara real-time. Langkah ini sangat penting karena pada banyak sistem IoT, Python-lah yang menjadi jembatan dari broker menuju analitik, database, dan model ML. Di titik ini, pembaca telah menyelesaikan dua sisi utama komunikasi MQTT: **publish** dan **subscribe**. Selanjutnya, data itu akan diberi “rumah” di file dan database agar bisa dianalisis, divisualisasikan, dan dipakai melatih model.

Hands-on 5 - Simpan Data ke CSV / SQLite / InfluxDB

1. Tujuan Hands-on

Pada latihan ini, pembaca akan menyimpan data dari subscriber MQTT ke tiga target berbeda:

- **CSV** untuk latihan paling sederhana
- **SQLite** untuk penyimpanan lokal yang ringan
- **InfluxDB** untuk *time-series storage* yang lebih cocok untuk data sensor

Dengan latihan ini, pembaca akan memahami bahwa penyimpanan bukan sekadar arsip, tetapi **fondasi untuk analitik, visualisasi, dashboard, dan machine learning**. Pada arsitektur IoT, data memang perlu diproses, disimpan, lalu dimanfaatkan oleh aplikasi di lapisan yang lebih tinggi.

2. Kapan memakai CSV, SQLite, dan InfluxDB?

CSV

Paling cocok untuk:

- latihan awal
- ekspor cepat
- debugging
- dataset kecil

SQLite

Paling cocok untuk:

- aplikasi lokal
- server mini / laptop Ubuntu
- riwayat data menengah
- analitik sederhana dengan SQL

InfluxDB

Paling cocok untuk:

- data sensor *time series*
- integrasi dengan Grafana
- query berbasis waktu
- dashboard real-time

3. Simpan data MQTT ke CSV

Buat file `scripts/mqtt/save_to_csv.py`:

```
import csv
import json
from pathlib import Path

import paho.mqtt.client as mqtt

BROKER_HOST = "localhost"
BROKER_PORT = 1883
TOPIC = "kampus/lab1/room01/telemetry"

OUTPUT_FILE = Path("data/raw/mqtt_room01.csv")
OUTPUT_FILE.parent.mkdir(parents=True, exist_ok=True)

HEADER = [
    "device_id",
    "temperature_c",
    "humidity_pct",
    "co2_ppm",
    "power_watt",
    "timestamp"
]

if not OUTPUT_FILE.exists():
    with open(OUTPUT_FILE, "w", newline="", encoding="utf-8") as
f:
        writer = csv.writer(f)
        writer.writerow(HEADER)

def on_connect(client, userdata, flags, reason_code,
properties=None):
    print("Terhubung ke broker.")
    client.subscribe(TOPIC, qos=1)

def on_message(client, userdata, msg):
    try:
        data = json.loads(msg.payload.decode())

        row = [
            data.get("device_id"),
            data.get("temperature_c"),
            data.get("humidity_pct"),
            data.get("co2_ppm"),
            data.get("power_watt"),
            data.get("timestamp"),
        ]

        with open(OUTPUT_FILE, "a", newline="",
encoding="utf-8") as f:
            writer = csv.writer(f)
```

```
                writer.writerow(row)

            print(f"Data disimpan ke CSV: {row}")

        except Exception as exc:
            print(f"Gagal menyimpan ke CSV: {exc}")

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect(BROKER_HOST, BROKER_PORT, 60)
client.loop_forever()
```

Jalankan:

```
python scripts/mqtt/save_to_csv.py
```

Hasil

Setiap pesan dari MQTT akan ditambahkan ke file:
`data/raw/mqtt_room01.csv`

4. Simpan data MQTT ke SQLite

SQLite sangat cocok untuk Ubuntu lokal karena tidak perlu server database terpisah.

Buat file `scripts/mqtt/save_to_sqlite.py`:

```
import json
import sqlite3
from pathlib import Path

import paho.mqtt.client as mqtt

BROKER_HOST = "localhost"
BROKER_PORT = 1883
TOPIC = "kampus/lab1/room01/telemetry"

DB_PATH = Path("data/raw/iot_sensor.db")
DB_PATH.parent.mkdir(parents=True, exist_ok=True)

conn = sqlite3.connect(DB_PATH)
cursor = conn.cursor()

cursor.execute("""
CREATE TABLE IF NOT EXISTS room_telemetry (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    device_id TEXT,
    temperature_c REAL,
    humidity_pct REAL,
    co2_ppm INTEGER,
```

```
    power_watt REAL,
    timestamp TEXT
)
""")
conn.commit()

def on_connect(client, userdata, flags, reason_code,
properties=None):
    print("Terhubung ke broker.")
    client.subscribe(TOPIC, qos=1)

def on_message(client, userdata, msg):
    try:
        data = json.loads(msg.payload.decode())

        cursor.execute("""
        INSERT INTO room_telemetry (
            device_id, temperature_c, humidity_pct, co2_ppm,
power_watt, timestamp
        ) VALUES (?, ?, ?, ?, ?, ?)
        """, (
            data.get("device_id"),
            data.get("temperature_c"),
            data.get("humidity_pct"),
            data.get("co2_ppm"),
            data.get("power_watt"),
            data.get("timestamp")
        ))
        conn.commit()

        print(f"Data disimpan ke SQLite: {data}")

    except Exception as exc:
        print(f"Gagal menyimpan ke SQLite: {exc}")

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect(BROKER_HOST, BROKER_PORT, 60)

try:
    client.loop_forever()
finally:
    conn.close()
```

Cek isi database

Pasang `sqlite3` bila belum ada:

```
sudo apt install -y sqlite3
```

Lalu cek:

```
sqlite3 data/raw/iot_sensor.db
```

Di prompt SQLite:

```
.tables
SELECT * FROM room_telemetry LIMIT 5;
SELECT AVG(temperature_c) FROM room_telemetry;
SELECT MAX(co2_ppm) FROM room_telemetry;
.quit
```

Mengapa SQLite penting?

SQLite memberi pembaca pengalaman nyata memakai query SQL untuk data sensor, tetapi tanpa kompleksitas server DB besar.

5. Simpan data MQTT ke InfluxDB

Untuk data sensor berbasis waktu, InfluxDB lebih natural karena dirancang untuk *time series*. Ini juga sangat cocok untuk integrasi dashboard.

5.1 Menjalankan InfluxDB dengan Docker di Ubuntu 26.04

Jika Docker belum terpasang, pasang dulu sesuai preferensi Anda. Untuk buku ini, pendekatan cepat yang praktis adalah Docker.

Jalankan:

```
docker run -d \
  --name influxdb2 \
  -p 8086:8086 \
  -v ~/iot-ml-book/influxdb-data:/var/lib/influxdb2 \
  influxdb:2
```

Cek:

```
docker ps
```

Buka antarmuka web lokal pada port **8086**. Saat pertama kali membuka, InfluxDB akan meminta:

- username
- password
- organization
- bucket
- token

Contoh konfigurasi awal:

- **username**: `admin`
- **organization**: `iotlab`
- **bucket**: `sensor_data`

Catat **token** yang dibuat karena akan dipakai oleh Python client.

5.2 Pasang pustaka Python untuk InfluxDB

```
source .venv/bin/activate
pip install influxdb-client
pip freeze > requirements.txt
```

5.3 Simpan data ke InfluxDB dengan Python

Buat file `scripts/mqtt/save_to_influxdb.py`:

import json

```
import paho.mqtt.client as mqtt
from influxdb_client import InfluxDBClient, Point,
WritePrecision
from influxdb_client.client.write_api import SYNCHRONOUS

BROKER_HOST = "localhost"
BROKER_PORT = 1883
TOPIC = "kampus/lab1/room01/telemetry"

INFLUX_URL = "http://localhost:8086"
INFLUX_TOKEN = "GANTI_DENGAN_TOKEN_ANDA"
INFLUX_ORG = "iotlab"
INFLUX_BUCKET = "sensor_data"

influx_client = InfluxDBClient(
    url=INFLUX_URL,
    token=INFLUX_TOKEN,
    org=INFLUX_ORG
)
write_api = influx_client.write_api(write_options=SYNCHRONOUS)
```

```
def on_connect(client, userdata, flags, reason_code,
properties=None):
    print("Terhubung ke broker MQTT.")
    client.subscribe(TOPIC, qos=1)

def on_message(client, userdata, msg):
    try:
        data = json.loads(msg.payload.decode())

        point = (
            Point("room_telemetry")
            .tag("device_id", data.get("device_id", "unknown"))
            .field("temperature_c",
float(data.get("temperature_c", 0)))
            .field("humidity_pct",
float(data.get("humidity_pct", 0)))
            .field("co2_ppm", int(data.get("co2_ppm", 0)))
            .field("power_watt", float(data.get("power_watt",
0)))
            .time(data.get("timestamp"), WritePrecision.S)
        )

        write_api.write(bucket=INFLUX_BUCKET, org=INFLUX_ORG,
record=point)
        print(f"Data disimpan ke InfluxDB: {data}")

    except Exception as exc:
        print(f"Gagal menyimpan ke InfluxDB: {exc}")

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect(BROKER_HOST, BROKER_PORT, 60)

try:
    client.loop_forever()
finally:
    influx_client.close()
```

6. Contoh implementasi real untuk penyimpanan

Ruang server

- MQTT menerima suhu dan kelembaban
- SQLite dipakai untuk pencatatan lokal
- InfluxDB dipakai untuk dashboard real-time
- Python ML nanti membaca histori suhu untuk mendeteksi anomali

Greenhouse

- data kelembaban tanah disimpan ke InfluxDB
- Grafana menampilkan tren per jam
- Python subscriber memutuskan kapan pompa menyala

Energi gedung

- data daya listrik dari beberapa ruangan masuk via MQTT
- SQLite dipakai untuk rekap harian
- InfluxDB dipakai untuk tampilan real-time beban gedung

7. Pola satu subscriber untuk banyak tujuan

Di sistem nyata, Anda tidak harus memilih hanya satu target. Anda bisa membuat satu subscriber yang:

- menyimpan ke CSV untuk backup mentah
- menyimpan ke SQLite untuk query cepat
- menyimpan ke InfluxDB untuk dashboard

Ini mendukung praktik **data acquisition**, **data storage**, **visualization**, dan **preprocessing** yang memang menjadi inti IoT analytics.

8. Versi gabungan: simpan ke CSV + SQLite sekaligus

Buat file `scripts/mqtt/save_to_csv_sqlite.py`:

```
import csv
import json
import sqlite3
from pathlib import Path

import paho.mqtt.client as mqtt

BROKER_HOST = "localhost"
BROKER_PORT = 1883
TOPIC = "kampus/lab1/room01/telemetry"

CSV_PATH = Path("data/raw/mqtt_room01_combined.csv")
```

```
DB_PATH = Path("data/raw/iot_sensor_combined.db")

CSV_PATH.parent.mkdir(parents=True, exist_ok=True)

if not CSV_PATH.exists():
    with open(CSV_PATH, "w", newline="", encoding="utf-8") as f:
        writer = csv.writer(f)
        writer.writerow([
            "device_id", "temperature_c", "humidity_pct",
            "co2_ppm", "power_watt", "timestamp"
        ])

conn = sqlite3.connect(DB_PATH)
cursor = conn.cursor()

cursor.execute("""
CREATE TABLE IF NOT EXISTS room_telemetry (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    device_id TEXT,
    temperature_c REAL,
    humidity_pct REAL,
    co2_ppm INTEGER,
    power_watt REAL,
    timestamp TEXT
)
""")
conn.commit()

def on_connect(client, userdata, flags, reason_code,
properties=None):
    client.subscribe(TOPIC, qos=1)

def on_message(client, userdata, msg):
    try:
        data = json.loads(msg.payload.decode())

        row = [
            data.get("device_id"),
            data.get("temperature_c"),
            data.get("humidity_pct"),
            data.get("co2_ppm"),
            data.get("power_watt"),
            data.get("timestamp"),
        ]

        with open(CSV_PATH, "a", newline="", encoding="utf-8")
as f:
            writer = csv.writer(f)
            writer.writerow(row)

        cursor.execute("""
        INSERT INTO room_telemetry (
```

```
            device_id, temperature_c, humidity_pct, co2_ppm,
power_watt, timestamp
        ) VALUES (?, ?, ?, ?, ?, ?)
        """, tuple(row))
        conn.commit()

        print(f"Disimpan ke CSV dan SQLite: {row}")

    except Exception as exc:
        print(f"Error: {exc}")

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect(BROKER_HOST, BROKER_PORT, 60)

try:
    client.loop_forever()
finally:
    conn.close()
```

9. Langkah kerja yang harus dilakukan pembaca

Tahap 1 — Jalankan broker

Pastikan Mosquitto aktif.

Tahap 2 — Jalankan publisher

Biarkan simulator sensor mengirim data terus-menerus.

Tahap 3 — Pilih target penyimpanan

Mulai dari CSV, lalu naik ke SQLite, lalu InfluxDB.

Tahap 4 — Jalankan subscriber penyimpanan

Pilih skrip sesuai target.

Tahap 5 — Verifikasi hasil

- buka CSV
- query SQLite
- cek bucket InfluxDB

Tahap 6 — Bandingkan kelebihan

Rasakan perbedaan kenyamanan masing-masing pendekatan.

10. Tantangan latihan lanjutan

Agar lebih menantang dan inspiratif, pembaca bisa mencoba:

- menyimpan data dari **beberapa topic** ke satu tabel
- menambahkan kolom `location`
- membuat tabel SQLite per jenis sensor
- memisahkan bucket InfluxDB untuk `room`, `energy`, `machine`
- menambahkan data kualitas udara lain seperti PM2.5
- menambahkan *alert topic* jika suhu > 30°C
- membuat skrip pembersihan data mentah menjadi `data/processed/`

11. Insight praktis untuk mahasiswa dan praktisi

Dari tiga media penyimpanan ini, pembaca akan belajar satu hal penting: **arsitektur data harus proporsional dengan kebutuhan**.

- Untuk belajar cepat, **CSV** sudah cukup.
- Untuk aplikasi lokal yang rapi, **SQLite** sangat berguna.
- Untuk *dashboard* dan *time-series analytics*, **InfluxDB** jauh lebih nyaman.

Tidak semua proyek harus langsung “besar”. Justru pendekatan bertahap seperti ini lebih cocok untuk laboratorium kampus, bengkel teknik, komunitas maker, dan implementasi awal di lapangan.

12. Penutup Hands-on 5

Dengan menyelesaikan latihan ini, pembaca telah membangun satu fondasi IoT yang utuh: **publisher mengirim data ke MQTT, subscriber Python menerima data, lalu data disimpan untuk analitik dan visualisasi**. Ini adalah bentuk paling nyata dari alur **sensor → broker → storage**, yang nanti akan diperluas menjadi **analytics → model → aksi** pada bab-bab berikutnya. Pendekatan ini juga sepenuhnya selaras dengan semangat buku: **praktis, open source, mudah direplikasi di Ubuntu, dan cukup kuat untuk berkembang menjadi pipeline IoT-ML lengkap**.

Pada tahap berikutnya, materi ini sudah sangat siap untuk dilanjutkan ke **Bab 3**, khususnya:

- eksplorasi dataset sensor dengan pandas,
- pembersihan timestamp,
- resampling,
- imputasi,
- dan deteksi anomali dasar.

Bab 3. Memahami Data Sensor IoT

3.1 Mengapa memahami data sensor lebih penting daripada langsung memilih model?

Banyak pemula ingin segera mencoba *machine learning*: melatih model, menghitung akurasi, lalu menampilkan hasil. Padahal, pada sistem IoT nyata, persoalan terbesar justru sering muncul **sebelum model dibangun**, yaitu pada **bentuk, kualitas, ritme, dan makna data sensor**. Data IoT datang dari perangkat yang beragam, dengan kecepatan berbeda, format berbeda, kualitas berbeda, dan konteks operasional yang tidak selalu rapi. Literatur yang Anda lampirkan menegaskan bahwa data IoT memiliki karakteristik khas seperti **volume, velocity, variety, veracity, variability, dan visualization**, serta mengandung informasi **spasial** dan **temporal** yang membuatnya berbeda dari data tabel biasa.

Karena itu, Bab 3 harus membentuk satu kebiasaan intelektual yang kuat pada pembaca: **jangan percaya buta pada data sensor**. Sebelum berpikir tentang model, pembaca harus bertanya: data ini datang dari sensor apa, seberapa sering dikirim, apakah waktunya rapi, apakah nilainya masuk akal, apakah ada data yang hilang, apakah pola datanya berubah, dan apakah label target tersedia dalam jumlah yang cukup. Pola pikir seperti ini akan sangat membantu saat pembaca nanti masuk ke bab tentang *anomaly detection*, *forecasting*, dan *pipeline IoT-ML* yang memang menjadi lanjutan struktur buku ini.

3.2 Apa yang dimaksud dengan data sensor IoT?

Secara sederhana, **data sensor IoT adalah representasi digital dari kondisi fisik atau kejadian di dunia nyata**. Sensor membaca besaran tertentu—misalnya suhu, kelembaban, posisi, arus, atau getaran—lalu mengubahnya menjadi nilai yang bisa dikirim, disimpan, dan dianalisis. Pada ekosistem IoT, sensor menjadi titik awal seluruh rantai nilai: data dikumpulkan dari lingkungan, dikirim ke platform IoT, diproses, dianalisis, lalu dipakai untuk mengirim perintah atau tindakan kembali ke sistem. Literatur juga menegaskan bahwa perangkat dan sensor merupakan komponen inti IoT, dan contoh sensor umum mencakup suhu, kelembaban, tekanan, cahaya, kedekatan, RFID, serta sensor bergerak seperti GPS pada ponsel atau kendaraan.

Namun yang sering dilupakan adalah: **data sensor tidak pernah berdiri sendiri**. Nilai “27.4” baru punya arti jika kita tahu bahwa itu adalah suhu dalam derajat Celsius, diambil dari sensor mana, pada jam berapa, di lokasi mana, dalam kondisi apa, dan apakah nilai itu normal untuk konteks tersebut. Itulah sebabnya analitik IoT selalu membutuhkan kombinasi antara **nilai numerik**, **waktu**, **identitas perangkat**, dan sering kali juga **lokasi** serta **kondisi operasional**. Literatur tentang IoT analytics juga memberi contoh bahwa sebuah stasiun cuaca biasanya menghasilkan beberapa *time series* sekaligus—suhu, kelembaban, tekanan, kecepatan angin, arah angin, dan curah hujan—yang sering kali lebih tepat dianalisis sebagai **sistem multivariat**, bukan variabel yang berdiri sendiri.

3.3 Jenis data IoT yang paling relevan untuk buku ini

Untuk menjaga buku tetap praktis dan fokus pada implementasi open source, kita akan memusatkan pembahasan pada beberapa jenis data sensor yang sangat umum dan realistis untuk kampus, laboratorium, industri kecil, komunitas *maker*, dan proyek riset mahasiswa.

3.3.1 Data suhu

Data suhu termasuk jenis data IoT yang paling mudah dipahami dan paling sering dipakai. Sensor suhu muncul di ruang server, ruang kelas, gudang, kulkas farmasi, greenhouse, ruang produksi, dan bahkan perangkat rumah tangga. Data suhu umumnya berbentuk numerik dan dicatat berkala, misalnya setiap 5 detik, 1 menit, atau 10 menit. Karena sederhana, data suhu sangat cocok untuk:

- latihan *time series* awal,
- deteksi ambang batas,
- visualisasi real-time,
- *forecasting* sederhana,
- dan *anomaly detection* dasar.

Contoh implementasi nyata:

- memantau suhu ruang server kampus,
- mengawasi suhu ruang laboratorium komputer,
- memonitor suhu lemari pendingin obat atau vaksin,
- memantau suhu greenhouse untuk sayuran hidroponik.

Literatur juga memberi contoh sistem *smart building* dengan sensor DHT11 yang mengukur suhu dan kelembaban di beberapa ruangan, lalu menyalakan indikator atau perangkat lain jika nilai melewati ambang tertentu.

3.3.2 Data kelembaban

kelembaban sering berpasangan dengan suhu. Pada bangunan, kelembaban mempengaruhi kenyamanan dan risiko jamur. Pada pertanian, kelembaban udara dan tanah sangat penting untuk pertumbuhan tanaman. Pada beberapa sistem industri, kelembaban mempengaruhi kualitas proses dan penyimpanan material.

Contoh implementasi nyata:

- kenyamanan ruang kuliah,
- ruang arsip,
- greenhouse,
- ruang penyimpanan bahan,
- pengendalian lingkungan pada budidaya jamur.

kelembaban biasanya lebih menarik bila dianalisis bersama suhu, karena banyak fenomena nyata muncul dari hubungan keduanya, bukan dari satu variabel saja. Ini

sejalan dengan literatur yang menekankan pentingnya properti temporal dan *visualization* serta hubungan antar data pada sistem IoT.

3.3.3 Data arus atau daya listrik

Arus listrik, tegangan, dan konsumsi energi adalah tipe data yang sangat relevan untuk gedung, laboratorium, peralatan industri, rumah pintar, dan sistem efisiensi energi. Data ini sangat bermanfaat untuk:

- memonitor beban,
- mendeteksi konsumsi tidak normal,
- mengenali jam puncak,
- mengidentifikasi pemborosan,
- dan membangun *smart energy analytics*.

Contoh implementasi nyata:

- pemantauan beban listrik ruang komputer,
- pengukuran daya AC atau pompa,
- pemantauan konsumsi energi gedung,
- evaluasi efisiensi alat praktikum.

Smart Energy Analytics sebagai salah satu studi kasus utama, sehingga data energi memang sangat layak dikenalkan sejak Bab 3 sebagai tipe data inti.

3.3.4 Data getaran

Data getaran sering dipakai pada *predictive maintenance*. Mesin yang masih normal biasanya memiliki pola getaran tertentu; ketika bantalan aus, poros tidak seimbang, atau ada gangguan mekanik, pola tersebut bisa berubah. Data getaran lebih menantang dibanding suhu karena:

- frekuensi sampelnya sering lebih tinggi,
- polanya lebih kompleks,
- perlu ekstraksi fitur,
- dan anomali sering tidak terlihat langsung pada nilai mentah.

Contoh implementasi nyata:

- motor listrik kecil di laboratorium,
- kipas industri,
- pompa air,
- simulasi mesin produksi.

Karena itu data getaran sangat cocok untuk menjembatani pembaca dari analitik sederhana ke dunia *feature engineering* dan *anomaly detection* pada bab berikutnya.

3.3.5 Data GPS atau lokasi

Data GPS menambahkan dimensi **spasial** pada IoT. Berbeda dari suhu atau arus yang relatif tetap pada titik tertentu, GPS berkaitan dengan **posisi bergerak**: lintang, bujur, kecepatan, arah, dan waktu. Literatur menjelaskan bahwa data IoT memiliki sifat spasial dan temporal; perangkat bisa bersifat tetap atau bergerak, serta bisa memiliki posisi absolut atau relatif. Ini penting karena analitik GPS tidak hanya menanyakan “berapa nilainya,” tetapi juga “di mana” dan “bergerak ke mana.”

Contoh implementasi nyata:

- pelacakan kendaraan kampus,
- logistik kecil,
- monitoring kendaraan pengangkut hasil panen,
- pemantauan rute drone atau robot bergerak.

Data GPS biasanya lebih menantang karena:

- update rate bisa berubah,
- sinyal bisa hilang,
- lokasi dapat loncat tidak realistis,
- dan butuh pembersihan sebelum dipakai.

3.3.6 Data konsumsi energi

Walaupun terkait dengan arus dan daya, **konsumsi energi** biasanya dianalisis dalam cakupan waktu yang lebih panjang: per menit, per jam, per hari, atau per minggu. Tipe data ini sangat penting untuk:

- audit energi,
- analitik gedung pintar,
- evaluasi efisiensi operasional,
- prediksi beban,
- dan keputusan manajemen sumber daya.

Contoh implementasi nyata:

- pemakaian listrik laboratorium per hari,
- konsumsi energi ruang kelas,
- penggunaan pompa irigasi,
- evaluasi pola pemakaian listrik asrama.

Data energi hampir selalu merupakan *time series* yang kuat, sering memiliki pola musiman harian atau mingguan, dan sangat cocok untuk mengajarkan konsep tren, periodisitas, dan *forecasting*.

3.4 Karakteristik data *time series* pada IoT

Salah satu karakter terpenting data sensor IoT adalah: **sebagian besar data itu adalah *time series***. Artinya, data diukur berurutan sepanjang waktu, dan urutannya **tidak boleh diabaikan**. Dalam literatur IoT analytics, *time series* dijelaskan sebagai urutan data yang diukur pada titik waktu berturut-turut, sering kali berjarak sama, seperti suhu yang dicatat setiap menit. Banyak data yang dilaporkan sensor adalah *time series*, sehingga teknik analitik dan *forecasting* IoT harus menghormati aspek temporal ini.

Ciri utama *time series* IoT

- **Ada urutan waktu**
- **Nilai sekarang sering dipengaruhi nilai sebelumnya**
- **Perubahan kecil bisa bermakna**
- **Data bisa periodik atau tidak periodik**
- **Sering mengandung tren, musiman, dan noise**

Literatur juga menegaskan bahwa data IoT dapat diperbarui secara **periodik** maupun **non-periodik**, dan frekuensi pembaruan perlu diperhatikan karena berpengaruh pada cara penyimpanan, *streaming*, dan *partitioning* data.

Mengapa urutan waktu penting?

Pada data tabel biasa, kita sering boleh mengacak baris selama nilainya tetap sama. Pada *time series*, itu berbahaya. Misalnya:

- suhu jam 08.00 lalu 08.10 lalu 08.20 menggambarkan perubahan fisik nyata;
- jika urutan rusak, kita bisa salah membaca tren;
- jika *timestamp* kacau, model prediksi akan belajar hal yang salah.

Karena itu, sebelum melakukan analitik apapun, pembaca harus selalu memeriksa:

- apakah `timestamp` valid,
- apakah urutan waktu benar,
- apakah ada duplikasi,
- apakah jarak antarwaktu konsisten,
- apakah ada loncatan yang tidak masuk akal.

Time series IoT sering multivariat

Banyak sistem IoT tidak hanya menghasilkan satu variabel, tetapi beberapa variabel sekaligus. Contoh klasik adalah stasiun cuaca dengan suhu, kelembaban, tekanan, dan curah hujan; atau ruang pintar dengan suhu, CO_2, kelembaban, dan konsumsi daya. Literatur Perros menegaskan bahwa bila beberapa *time series* saling mempengaruhi, lebih tepat memandangnya sebagai **variabel multivariat** daripada sekumpulan variabel terpisah. Ini sangat penting untuk IoT karena lingkungan nyata hampir selalu saling terkait.

Time series IoT sering tidak stasioner

Literatur juga menjelaskan bahwa *time series* bisa **stasioner** atau **tidak stasioner**. *Time series* yang tidak stasioner dapat menunjukkan:

1. **tren** atau *drift* rata-rata,
2. **variansi yang berubah**,
3. **musiman** atau *seasonality*.

Dalam praktik IoT, data non-stasioner justru sangat umum:

- suhu ruang berubah antara pagi dan siang,
- beban listrik berbeda antara hari kerja dan akhir pekan,
- pola mesin berubah setelah perawatan,
- kelembaban tanah berbeda antara musim hujan dan kemarau.

Karena itu, pembaca tidak boleh mengharapkan data sensor selalu tenang, stabil, dan “cantik”. Justru dinamika itulah yang membuat IoT menarik.

3.5 Masalah umum pada data sensor IoT

Setelah memahami jenis dan sifat temporal data, kita masuk ke masalah yang paling sering ditemui. Bagian ini sangat penting karena kualitas analitik dan *machine learning* sangat ditentukan oleh bagaimana kita menangani masalah-masalah berikut.

3.5.1 *Missing value* atau data hilang

Missing value adalah kondisi ketika data yang seharusnya ada ternyata tidak tersedia. Ini sangat umum pada IoT. Penyebabnya bisa bermacam-macam:

- sensor mati,
- baterai habis,
- koneksi jaringan terputus,
- gateway restart,
- broker sibuk,
- kesalahan parsing,
- atau perangkat sengaja hanya mengirim saat terjadi perubahan.

Literatur yang Anda lampirkan secara eksplisit menyebut **missing values** sebagai penghambat kualitas data IoT, khususnya pada pemantauan kesehatan jarak jauh, sehingga dibutuhkan pendekatan imputasi untuk memperbaikinya.

Contoh nyata

- Sensor suhu ruang server tidak mengirim data selama 15 menit karena Wi-Fi putus.
- Sensor kelembaban tanah mengirim setiap 10 menit, tetapi pukul 13.00–14.00 kosong.
- GPS kendaraan kehilangan sinyal di area tertutup.
- Sensor daya gagal merekam saat listrik padam lalu sistem reboot.

Dampak *missing value*

- grafik jadi patah,
- rata-rata menjadi bias,
- model prediksi salah belajar,
- deteksi anomali bisa salah menandai,
- dashboard terlihat “normal” padahal datanya tidak lengkap.

Hal yang harus dilakukan pembaca

- hitung berapa banyak data yang hilang,
- cari pola kehilangannya,
- bedakan hilang acak vs hilang sistematis,
- pilih strategi penanganan yang sederhana dulu:
    - hapus jika sangat sedikit,
    - isi dengan *forward fill*,
    - isi dengan rata-rata/median,
    - atau tandai sebagai “hilang” untuk analisis khusus.

3.5.2 *Outlier* atau nilai menyimpang

Outlier adalah nilai yang jauh berbeda dari pola normal data. Dalam IoT, *outlier* tidak selalu berarti kesalahan. Kadang ia menunjukkan:

- sensor rusak,
- noise sesaat,
- atau justru kejadian penting yang harus diperhatikan.

Contoh:

- suhu ruang tiba-tiba tercatat **85°C** padahal biasanya 24–29°C,
- GPS melompat dari Jakarta ke Surabaya dalam 1 menit,
- sensor arus tiba-tiba nol saat mesin sebenarnya tetap menyala,
- getaran mesin melonjak sesaat karena benturan.

Kenapa *outlier* penting?

Karena *outlier* bisa berarti dua hal:

1. **error data**, atau
2. **event penting**.

Dalam konteks IoT dan IIoT, *outlier* sering menjadi pintu masuk menuju *anomaly detection*, yang memang sangat relevan untuk sistem pemantauan mesin, energi, keamanan, dan kesehatan.

Langkah yang harus dilakukan

- visualisasikan data dulu,
- tetapkan batas wajar berbasis domain,
- bandingkan dengan sensor lain bila ada,
- cek apakah *outlier* tunggal atau berulang,
- jangan langsung buang semua *outlier*.

Untuk pembelajaran awal, metode sederhana seperti:

- ambang minimum-maksimum,
- Z-score,
- IQR,
  sudah sangat cukup sebelum masuk ke model yang lebih canggih.

3.5.3 *Drift* atau perubahan pola dari waktu ke waktu

Drift adalah salah satu tantangan paling penting dalam IoT, tetapi sering diabaikan pemula. Secara sederhana, *drift* berarti pola data berubah secara bertahap atau permanen. Literatur time series menjelaskan bahwa *drift* dapat muncul sebagai **pergeseran rata-rata** atau tren, dan data yang tidak stasioner memang bisa menunjukkan perubahan mean, varians, atau seasonality sepanjang waktu.

Bentuk *drift* yang umum

- Sensor mulai bias karena penuaan perangkat.
- Lingkungan berubah, misalnya ruangan direnovasi atau ventilasi dibuka.
- Pola beban energi berubah setelah jadwal kelas berganti.
- Mesin mengalami keausan bertahap.
- Pengguna mengubah perilaku penggunaan perangkat.

Contoh nyata

- Suhu rata-rata ruang server naik perlahan setiap minggu karena filter AC kotor.
- Konsumsi daya laboratorium naik sejak ditambah perangkat baru.
- Sensor pH atau kelembaban tanah mulai bias setelah beberapa bulan.
- GPS kendaraan menunjukkan pola rute berbeda setelah perubahan operasional.

Mengapa *drift* penting?

Karena model ML yang dilatih dari data lama bisa menjadi tidak akurat saat pola berubah. **Drift detection** dan **penurunan performa model** sebagai bagian penting pada bab operasional sistem IoT-ML.

Langkah praktis untuk pembaca

- bandingkan statistik mingguan/bulanan,
- plot rata-rata bergerak,
- cek apakah distribusi data bergeser,
- jangan menganggap model berlaku selamanya,
- siapkan proses evaluasi dan *retraining* di masa depan.

3.5.4 Interval waktu yang tidak konsisten

Dalam dunia ideal, sensor mungkin mengirim data tepat setiap 10 detik. Dalam dunia nyata, itu sering tidak terjadi. Ada jeda, duplikasi, keterlambatan, dan pengiriman tidak beraturan. Literatur menekankan bahwa data IoT dapat diperbarui dengan frekuensi berbeda-beda, baik periodik maupun non-periodik, dan bahwa kecepatan pengumpulan data dapat berbeda antar perangkat atau antar insiden.

Contoh masalah interval

- seharusnya data tiap 1 menit, tetapi kadang 58 detik, kadang 70 detik,
- dua data masuk pada *timestamp* yang sama,
- ada celah 20 menit,
- sensor hanya mengirim saat nilai berubah,
- gateway mengirim batch data terlambat setelah koneksi pulih.

Mengapa ini berbahaya?

- agregasi per jam bisa salah,
- visualisasi menjadi membingungkan,
- *rolling window* jadi tidak akurat,
- model *forecasting* menjadi lemah,
- fitur berbasis *lag* bisa menipu.

Langkah yang harus dilakukan

- urutkan data berdasarkan waktu,
- cek selisih antar-*timestamp*,
- identifikasi duplikasi,
- lakukan *resampling* ke interval target,
- isi celah dengan strategi yang sesuai.

hands-on Bab 3 secara eksplisit memasukkan **membersihkan timestamp, resampling, dan imputasi sederhana** sebagai latihan inti.

3.5.5 Label yang jarang atau tidak seimbang

Pada banyak tugas ML IoT, label “kejadian penting” justru sangat sedikit. Misalnya:

- kondisi normal mesin ada 99%,
- kondisi rusak hanya 1%,
- ruangan nyaman hampir sepanjang hari,
- alarm kebakaran sangat jarang,
- pencurian air, lonjakan daya, atau kebocoran pipa hanya sesekali terjadi.

Inilah yang disebut **label yang jarang** atau **data tidak seimbang**. Masalah ini sangat nyata di IoT karena dunia fisik normalnya memang lebih sering stabil daripada bermasalah.

Dampaknya

- model klasifikasi bisa terlihat akurat padahal hanya selalu menebak “normal”,
- *recall* untuk kejadian penting menjadi sangat buruk,
- evaluasi dengan akurasi saja menjadi menyesatkan.

Contoh nyata

- Dari 100.000 pembacaan sensor getaran, hanya 300 yang terkait kondisi *fault*.
- Dari jutaan data energi, hanya sedikit yang menunjukkan anomali konsumsi.
- Dari data kesehatan *wearable*, kejadian abnormal relatif sangat sedikit.

Langkah yang harus dilakukan

- jangan hanya lihat *accuracy*,
- gunakan *precision*, *recall*, dan F1,
- pertimbangkan *anomaly detection* bila label sangat terbatas,
- buat strategi pengambilan sampel yang masuk akal,
- dan pahami bahwa data lapangan yang “sedikit kasus buruk” bukan masalah kecil, tetapi kondisi khas IoT.

Pendekatan berbasis *threshold*, *Isolation Forest*, dan *One-Class SVM* sebagai topik yang sangat relevan untuk data sensor dan IIoT.

3.6 Hubungan antara jenis data dan masalah kualitas

Supaya pembaca tidak melihat masalah-masalah tadi sebagai daftar terpisah, penting untuk memahami bahwa **setiap tipe data punya pola masalahnya sendiri**.

Suhu dan kelembaban

- rentan *missing value* karena koneksi atau sensor murah,
- mudah punya *outlier* ekstrim,
- sering menunjukkan *drift* musiman,
- relatif mudah di-*resample*.

Arus dan energi

- bisa melonjak tajam saat beban berubah,
- *outlier* bisa berarti event penting,
- pola harian/mingguan kuat,
- label anomali sering sedikit.

Getaran

- data sering padat dan lebih bising,
- *outlier* bisa campuran antara noise dan indikasi *fault*,
- interval pencatatan penting sekali,
- perlu fitur tambahan.

GPS

- rentan *missing value* di area sinyal lemah,
- lonjakan lokasi palsu sangat umum,
- interval tidak konsisten,
- konteks spasial wajib diperhatikan.

Literatur juga menegaskan bahwa data IoT harus dipahami berdasarkan properti **spasial**, **temporal**, dan **sensing**, termasuk apakah perangkat bersifat tetap atau bergerak, bagaimana frekuensi pembaruan berlangsung, dan apakah format data antar perangkat berbeda.

3.7 Contoh-contoh real yang bisa diimplementasikan

Agar bab ini terasa hidup dan inspiratif, berikut beberapa contoh implementasi yang sangat cocok untuk mahasiswa dan praktisi.

Smart room monitoring

Sensor suhu, kelembaban, dan CO_2 dipasang di ruang kelas atau lab. Tantangannya:

- ada *missing value* saat Wi-Fi putus,
- ada *outlier* saat sensor restart,
- ada *drift* antara pagi dan siang,
- interval pengiriman kadang berubah.

Kasus ini sangat mudah dipakai di kampus karena murah, konkret, dan dekat dengan kehidupan sehari-hari. Sistem serupa juga disebutkan dalam studi *smart building* dengan suhu dan kelembaban berbasis sensor.

Smart energy analytics

Sensor daya dipasang pada perangkat laboratorium atau gedung kecil. Tantangannya:

- beban harian berubah,
- *outlier* dapat menandai alat rusak atau pemakaian tak wajar,
- label pemborosan energi biasanya jarang,
- data sangat cocok untuk visualisasi dan *forecasting*.

Predictive maintenance sederhana

Sensor getaran atau arus dipasang pada motor kecil. Tantangannya:

- sinyal mentah berisik,
- *outlier* tidak selalu salah,
- *drift* bisa muncul karena keausan,
- data normal jauh lebih banyak daripada data rusak.

Precision agriculture

Sensor suhu udara, kelembaban tanah, dan cahaya dipasang di greenhouse. Literatur yang Anda lampirkan menunjukkan bahwa pendekatan *data-driven* pada pertanian presisi sangat penting untuk memahami kondisi budidaya dan menghasilkan peringatan dini atau kontrol otomatis yang lebih baik.

3.8 Langkah kerja yang harus dilakukan pembaca saat menerima data sensor

Agar bab ini benar-benar praktis, pembaca perlu dibiasakan pada alur kerja sederhana namun disiplin.

Tahap 1 — Kenali sensor dan konteksnya

Sebelum membuka Python, jawab dulu:

- sensor apa?
- satuannya apa?
- lokasi di mana?
- frekuensi kirim berapa?
- normalnya seperti apa?

Tahap 2 — Cek struktur data

Pastikan ada kolom minimal seperti:

- `timestamp`
- `device_id`
- nilai sensor
- lokasi atau metadata lain bila ada

Tahap 3 — Cek kualitas dasar

Lihat:

- adakah nilai kosong?
- adakah duplikasi?
- adakah nilai yang jelas tidak masuk akal?
- apakah interval waktunya rapi?

Tahap 4 — Visualisasikan

Buat grafik sederhana. Sering kali masalah data justru paling cepat terlihat dari grafik.

Tahap 5 — Identifikasi pola waktu

Tanyakan:

- apakah ada tren?
- apakah ada musiman?
- apakah ada lonjakan abnormal?
- apakah ada jeda pengiriman?

Tahap 6 — Tandai masalah utama

Jangan langsung memperbaiki semuanya. Tandai dulu mana yang paling kritis:

- *missing value*,
- *outlier*,
- *drift*,
- interval kacau,
- label langka.

Tahap 7 — Baru lakukan pembersihan

Setelah itu barulah lakukan:

- konversi `timestamp`,
- *sorting*,
- *resampling*,
- imputasi,
- filter awal.

3.9 Sikap profesional terhadap data sensor

Satu pelajaran penting dari bab ini adalah: **data sensor harus diperlakukan dengan rendah hati**. Jangan terlalu cepat menyimpulkan hanya karena angkanya banyak. Data yang besar belum tentu rapi. Data yang real-time belum tentu benar. Data dari sensor canggih pun bisa bias, hilang, atau berubah pola. Literatur juga menekankan bahwa IoT users tidak tertarik pada data mentah; yang dibutuhkan adalah terjemahan data menjadi bentuk yang produktif dan berguna bagi pengambilan keputusan.

Karena itu, seorang praktisi IoT yang baik bukan hanya pandai memasang sensor atau melatih model, tetapi juga teliti membaca kualitas data. Inilah pembeda antara proyek demo dan sistem yang benar-benar bermanfaat.

3.10 Penutup bab

Bab 3 adalah titik ketika pembaca mulai benar-benar "bertemu" dengan data lapangan. Di sini pembaca belajar bahwa data IoT tidak hanya beragam—suhu, kelembaban, arus, getaran, GPS, energi—tetapi juga hidup dalam waktu sebagai *time series*, sering multivariat, dan penuh tantangan nyata seperti *missing value*, *outlier*, *drift*, interval waktu yang tidak konsisten, serta label yang jarang. Semua ini bukan hambatan untuk ditakuti, melainkan bahan pembelajaran yang justru membuat analitik IoT menjadi menarik, menantang, dan bermakna. Fondasi ini akan sangat membantu saat pembaca masuk ke latihan berikutnya: **eksplorasi dataset sensor dengan pandas**, **membersihkan timestamp**, **resampling**, **imputasi**, dan **deteksi anomali dasar**—persis seperti jalur yang sudah dirancang.

Hands-on 6 - Eksplorasi Dataset Sensor dengan pandas

1. Tujuan Hands-on

Pada latihan ini, pembaca akan belajar membaca, memeriksa, dan memahami dataset sensor IoT menggunakan **Python** dan **pandas** di **Ubuntu 26.04**. Setelah selesai, pembaca diharapkan mampu:

- membaca file CSV sensor ke dalam DataFrame,
- memeriksa struktur kolom dan tipe data,
- memahami rentang waktu data,
- melihat statistik dasar,
- menemukan indikasi awal masalah kualitas data,
- dan menarik insight awal yang berguna untuk monitoring, dashboard, atau langkah pembersihan data berikutnya.

Latihan ini penting karena dalam praktik IoT, **sebelum melakukan ML**, kita harus bisa **membaca bahasa data**. Buku *An Introduction to IoT Analytics* juga menempatkan *time series forecasting* dan identifikasi *seasonality* sebagai bagian penting dalam analitik IoT, sementara *Big Data Analytics for Internet of Things* menekankan pentingnya menjembatani **data** menjadi **keputusan**.

2. Contoh implementasi nyata yang relevan

Agar latihan terasa hidup, kita gunakan sudut pandang implementasi yang benar-benar bisa dilakukan di kampus, laboratorium, UMKM, atau industri kecil.

Contoh 1 — Monitoring ruang server kampus

Sensor suhu, kelembaban, dan daya listrik dikirim tiap menit. Sebelum membuat alarm, kita harus tahu:

- kapan suhu paling tinggi,
- apakah ada data yang hilang,
- apakah daya naik saat suhu naik,
- apakah ada pola harian.

Contoh 2 — Smart classroom atau laboratorium

Sensor CO_2, suhu, dan kelembaban dipasang di ruangan belajar. Dari eksplorasi data, kita bisa mulai menjawab:

- apakah ruangan makin panas saat jam kuliah padat,
- apakah ventilasi cukup baik,
- kapan ruangan terasa paling tidak nyaman.

Contoh 3 — Monitoring beban energi bengkel atau lab

Data arus dan daya direkam per menit. Dari eksplorasi awal, kita bisa melihat:

- jam puncak beban,
- lonjakan konsumsi,
- potensi pemborosan,
- dan kapan alat menyala tetapi seharusnya tidak digunakan.

3. Dataset yang akan dipakai

Kita akan memakai dataset sintetis yang realistis untuk **monitoring ruangan**. Ini cocok untuk latihan awal karena:

- mudah dibayangkan,
- dekat dengan kebutuhan nyata,
- multivariat,
- dan cukup kaya untuk eksplorasi maupun *cleaning*.

Kolom dataset

- `timestamp`
- `device_id`
- `temperature_c`
- `humidity_pct`
- `co2_ppm`
- `power_watt`

Simpan file sebagai:

```
data/raw/room_sensor_week.csv
```

Contoh isi file:

```
timestamp,device_id,temperature_c,humidity_pct,co2_ppm,power_watt
2026-01-01 08:00:00,room01,24.5,58,520,110
2026-01-01 08:10:00,room01,24.7,57,525,112
2026-01-01 08:20:00,room01,25.1,56,540,115
2026-01-01 08:30:00,room01,25.8,55,560,118
2026-01-01 08:40:00,room01,26.2,54,590,121
2026-01-01 08:50:00,room01,26.8,53,620,126
2026-01-01 09:00:00,room01,27.1,52,650,130
2026-01-01 09:10:00,room01,27.6,51,680,135
2026-01-01 09:20:00,room01,28.1,50,710,139
2026-01-01 09:30:00,room01,28.4,49,740,143
2026-01-01 09:40:00,room01,28.8,48,770,147
2026-01-01 09:50:00,room01,29.0,47,790,150
2026-01-01 10:00:00,room01,29.2,46,810,152
2026-01-01 10:10:00,room01,29.0,47,800,150
2026-01-01 10:20:00,room01,28.7,48,780,146
2026-01-01 10:30:00,room01,28.3,49,750,142
2026-01-01 10:40:00,room01,27.9,50,720,138
```

```
2026-01-01 10:50:00,room01,27.4,51,690,134
2026-01-01 11:00:00,room01,27.0,52,660,130
2026-01-01 11:10:00,room01,26.7,53,635,126
```

4. Persiapan lingkungan Ubuntu 26.04

Pastikan *virtual environment* aktif:

```
source .venv/bin/activate
```

Pastikan pustaka tersedia:

```
pip install pandas matplotlib numpy
```

Buat folder kerja:

```
mkdir -p data/raw data/processed scripts outputs/figures
```

5. Source code utama: eksplorasi dataset dengan pandas

Buat file:

```
scripts/explore_sensor_dataset.py
```

Isi:

```
from pathlib import Path
import pandas as pd

DATA_PATH = Path("data/raw/room_sensor_week.csv")

df = pd.read_csv(DATA_PATH)

print("=== 5 baris pertama ===")
print(df.head())

print("\n=== Ukuran data ===")
print(df.shape)

print("\n=== Nama kolom ===")
print(df.columns.tolist())

print("\n=== Info dataset ===")
print(df.info())

print("\n=== Statistik numerik ===")
print(df.describe(include="all"))
Jalankan:
python scripts/explore_sensor_dataset.py
```

Apa yang dipelajari?

- ukuran dataset,
- kolom yang tersedia,
- tipe data,
- ringkasan statistik awal,
- dan apakah ada indikasi kolom waktu yang masih berupa string.

6. Konversi `timestamp` dan cek rentang waktu

Tambahkan eksplorasi waktu:

```
from pathlib import Path
import pandas as pd

DATA_PATH = Path("data/raw/room_sensor_week.csv")

df = pd.read_csv(DATA_PATH)
df["timestamp"] = pd.to_datetime(df["timestamp"])

print("=== Rentang waktu ===")
print("Mulai :", df["timestamp"].min())
print("Akhir :", df["timestamp"].max())

print("\n=== Jumlah device ===")
print(df["device_id"].nunique())
print(df["device_id"].unique())
```

Mengubah kolom waktu menjadi `datetime` sangat penting karena data sensor IoT umumnya bersifat *time series*, dan analisis waktu akan sulit dilakukan jika `timestamp` masih berupa teks biasa. Literatur Perros juga menempatkan *time series* dan identifikasi pola seperti *seasonality* sebagai aspek dasar analitik IoT.

7. Menjadikan data sebagai *time series*

```
from pathlib import Path
import pandas as pd

DATA_PATH = Path("data/raw/room_sensor_week.csv")

df = pd.read_csv(DATA_PATH)
df["timestamp"] = pd.to_datetime(df["timestamp"])
df = df.sort_values("timestamp")
df = df.set_index("timestamp")

print(df.head())
print("\nFrekuensi data belum dipaksa, tapi index waktu sudah
siap.")
```

8. Eksplorasi statistik yang lebih berguna

Buat ringkasan operasional:

```
from pathlib import Path
import pandas as pd

DATA_PATH = Path("data/raw/room_sensor_week.csv")

df = pd.read_csv(DATA_PATH)
df["timestamp"] = pd.to_datetime(df["timestamp"])
df = df.set_index("timestamp")

summary = {
    "temperature_min": df["temperature_c"].min(),
    "temperature_max": df["temperature_c"].max(),
    "temperature_mean": df["temperature_c"].mean(),
    "humidity_min": df["humidity_pct"].min(),
    "humidity_max": df["humidity_pct"].max(),
    "co2_max": df["co2_ppm"].max(),
    "power_mean": df["power_watt"].mean(),
}

print("=== Ringkasan operasional ===")
for k, v in summary.items():
    print(f"{k}: {v}")
```

Contoh insight nyata

- suhu maksimum membantu menentukan apakah alarm suhu perlu dipasang,
- CO_2 maksimum membantu mengevaluasi ventilasi,
- daya rata-rata membantu membaca pola beban,
- kelembaban minimum membantu melihat kondisi ruangan terlalu kering atau tidak.

9. Eksplorasi korelasi awal

Dalam banyak sistem IoT, hubungan antar variabel lebih menarik daripada satu variabel tunggal. Buku Perros juga mencontohkan bahwa data sensor seperti stasiun cuaca sering sebaiknya diperlakukan sebagai **sistem multivariat**, bukan hanya variabel tunggal yang berdiri sendiri.

```
from pathlib import Path
import pandas as pd

DATA_PATH = Path("data/raw/room_sensor_week.csv")

df = pd.read_csv(DATA_PATH)
df["timestamp"] = pd.to_datetime(df["timestamp"])

corr = df[["temperature_c", "humidity_pct", "co2_ppm",
"power_watt"]].corr()

print("=== Korelasi antar variabel ===")
print(corr)
```

Interpretasi yang bisa dilakukan

- suhu naik bersama CO_2 bisa menandakan ruangan makin ramai,
- suhu naik bersama daya bisa menandakan pendingin bekerja lebih keras,
- kelembaban turun saat suhu naik adalah pola yang realistis di banyak ruangan.

10. Visualisasi eksploratif pertama

Buat file:

`scripts/plot_exploration.py`

Isi:

```
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt

DATA_PATH = Path("data/raw/room_sensor_week.csv")
OUTPUT_DIR = Path("outputs/figures")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

df = pd.read_csv(DATA_PATH)
df["timestamp"] = pd.to_datetime(df["timestamp"])
df = df.set_index("timestamp")

# Plot suhu
plt.figure(figsize=(10, 5))
plt.plot(df.index, df["temperature_c"], marker="o")
plt.title("Eksplorasi Awal - Suhu Ruangan")
plt.xlabel("Waktu")
plt.ylabel("Suhu (C)")
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig(OUTPUT_DIR / "explore_temperature.png")
plt.show()

# Plot CO2
plt.figure(figsize=(10, 5))
plt.plot(df.index, df["co2_ppm"], marker="o")
plt.title("Eksplorasi Awal - CO2 Ruangan")
plt.xlabel("Waktu")
plt.ylabel("CO2 (ppm)")
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig(OUTPUT_DIR / "explore_co2.png")
plt.show()
```

11. Langkah kerja yang harus dilakukan pembaca

Tahap	Kegiatan
1	Baca file CSV.
2	Periksa kolom, ukuran, dan tipe data.
3	Ubah timestamp ke format waktu.
4	Urutkan berdasarkan waktu.
5	Hitung statistik dasar.
6	Cek korelasi awal.
7	Buat grafik eksploratif.
8	Tuliskan insight operasional sederhana.

12. Penutup Hands-on 6

Dengan latihan ini, pembaca sudah mulai melihat bahwa **eksplorasi data sensor bukan sekadar membuka CSV**, tetapi proses membaca perilaku sistem fisik dari data. Inilah jembatan awal menuju pembersihan data, *resampling*, imputasi, dan nanti *anomaly detection*. Semangat ini sejalan dengan struktur buku Anda yang memang memposisikan eksplorasi data sebagai fondasi sebelum teknik yang lebih lanjut.

Hands-on 7 - Membersihkan Timestamp, Resampling, dan Imputasi Sederhana

1. Tujuan Hands-on

Pada latihan ini, pembaca akan:

- membersihkan kolom waktu,
- menghapus duplikasi waktu,
- menangani urutan waktu yang berantakan,
- melakukan *resampling* ke interval target,
- dan mengisi data hilang dengan imputasi sederhana.

Ini sangat penting karena data IoT nyata sering datang **tidak rapi**, dengan interval tidak konsisten dan nilai hilang.

Di sisi teori, *An Introduction to IoT Analytics* membahas bahwa *time series* bisa bersifat non-stasioner, memiliki tren dan *seasonality*, sehingga penanganan waktu yang benar menjadi hal mendasar sebelum model apapun diterapkan.

2. Contoh nyata yang bisa diimplementasikan

Ruang server

Sensor seharusnya mengirim tiap 10 menit, tetapi kadang:

- terlambat,
- ganda,
- atau hilang karena koneksi putus.

Greenhouse

Sensor tanah kadang mengirim lebih cepat setelah penyiraman, lalu lambat saat baterai lemah.

Monitoring kendaraan

GPS kadang hilang saat melewati area tertutup, lalu muncul lagi dengan lompatan waktu.

Semua kasus ini butuh pembersihan waktu dan *resampling* agar data siap dianalisis.

3. Dataset kotor untuk latihan

Simpan file ini sebagai:

```
data/raw/room_sensor_dirty.csv
```

Isi:

```
timestamp,device_id,temperature_c,humidity_pct,co2_ppm,p
ower_watt
2026-01-01 08:00:00,room01,24.5,58,520,110
2026-01-01 08:10:00,room01,24.7,57,525,112
2026-01-01 08:25:00,room01,25.1,56,540,115
2026-01-01 08:30:00,room01,25.8,55,560,118
2026-01-01 08:30:00,room01,25.9,55,562,119
2026-01-01 08:55:00,room01,26.8,53,620,126
2026-01-01 09:00:00,room01,27.1,52,650,130
2026-01-01 09:20:00,room01,28.1,50,710,139
2026-01-01 09:30:00,room01,,49,740,143
2026-01-01 09:40:00,room01,28.8,,770,147
2026-01-01 09:50:00,room01,29.0,47,,150
2026-01-01 10:00:00,room01,29.2,46,810,
bad_timestamp,room01,29.0,47,800,150
2026-01-01 10:20:00,room01,28.7,48,780,146
2026-01-01 10:50:00,room01,27.4,51,690,134
```

Dataset ini sengaja memiliki:

- *timestamp* rusak,
- duplikasi waktu,
- interval tidak konsisten,
- beberapa nilai kosong.

4. Source code utama: membersihkan waktu

Buat file:

```
scripts/clean_timestamp_resample.py
```

Isi:

```
from pathlib import Path
import pandas as pd

DATA_PATH = Path("data/raw/room_sensor_dirty.csv")
OUTPUT_PATH = Path("data/processed/room_sensor_clean.csv")
OUTPUT_PATH.parent.mkdir(parents=True, exist_ok=True)

df = pd.read_csv(DATA_PATH)

print("=== Data awal ===")
print(df.head())
print(f"Jumlah baris awal: {len(df)}")

# Parse timestamp, ubah error menjadi NaT
df["timestamp"] = pd.to_datetime(df["timestamp"],
errors="coerce")

# Buang baris dengan timestamp invalid
df = df.dropna(subset=["timestamp"])

# Urutkan berdasarkan waktu
df = df.sort_values("timestamp")

# Hilangkan duplikasi timestamp, ambil record terakhir
df = df.drop_duplicates(subset=["timestamp"], keep="last")

print("\n=== Setelah membersihkan timestamp ===")
print(df.head())
print(f"Jumlah baris setelah clean timestamp: {len(df)}")
```

Yang dilakukan kode

- `errors="coerce"` mengubah waktu rusak menjadi `NaT`,
- `dropna(subset=["timestamp"])` membuang baris waktu invalid,
- `sort_values()` merapikan urutan waktu,
- `drop_duplicates()` menghapus duplikasi waktu.

5. Set index waktu dan cek celah data

Tambahkan:

```
df = df.set_index("timestamp")

print("\n=== Selisih antar timestamp ===")
print(df.index.to_series().diff())
```

Ini membantu pembaca melihat apakah sensor benar-benar konsisten mengirim data.

6. Resampling ke interval 10 menit

Dalam praktik IoT, kita sering ingin mengubah data “berantakan” menjadi interval tetap agar lebih mudah dianalisis. Literatur Perros juga menunjukkan pentingnya pengolahan *time series* untuk mendukung *forecasting*, termasuk pengenalan pola musiman dan transformasi data.

Tambahkan:

```
# Resampling per 10 menit
df_resampled = df.resample("10min").mean(numeric_only=True)

# Untuk kolom kategori seperti device_id, isi dengan forward
fill
df_resampled["device_id"] = "room01"

print("\n=== Data setelah resampling 10 menit ===")
print(df_resampled.head(12))
```

Apa itu resampling?

Resampling berarti memetakan data ke interval waktu baru yang konsisten, misalnya:

- 10 menit,
- 1 menit,
- 1 jam,
- atau 1 hari.

7. Imputasi sederhana untuk nilai hilang

Sekarang isi nilai kosong secara bertahap.

7.1 Forward fill

```
df_ffill = df_resampled.copy()
df_ffill[["temperature_c", "humidity_pct", "co2_ppm",
"power_watt"]] = (
    df_ffill[["temperature_c", "humidity_pct", "co2_ppm",
"power_watt"]].ffill()
)

print("\n=== Setelah forward fill ===")
print(df_ffill.head(12))
```

7.2 Interpolasi linier

```
df_interp = df_resampled.copy()
df_interp[["temperature_c", "humidity_pct", "co2_ppm",
"power_watt"]] = (
    df_interp[["temperature_c", "humidity_pct", "co2_ppm",
"power_watt"]].interpolate(method="linear")
)

print("\n=== Setelah interpolasi linier ===")
print(df_interp.head(12))
```

Kapan pakai apa?

- **Forward fill** cocok bila nilai tidak berubah drastis dan sensor memang bersifat stabil.
- **Interpolasi linier** cocok bila perubahan di antara dua titik masuk akal diasumsikan mulus.

8. Simpan hasil data bersih

Tambahkan:

```
df_interp.to_csv(OUTPUT_PATH)
print(f"\nData bersih disimpan ke: {OUTPUT_PATH}")
```

9. Versi lengkap skrip

Berikut versi lengkapnya:

```
from pathlib import Path
import pandas as pd

DATA_PATH = Path("data/raw/room_sensor_dirty.csv")
OUTPUT_PATH = Path("data/processed/room_sensor_clean.csv")
OUTPUT_PATH.parent.mkdir(parents=True, exist_ok=True)

df = pd.read_csv(DATA_PATH)

# 1. Parse timestamp
df["timestamp"] = pd.to_datetime(df["timestamp"],
errors="coerce")

# 2. Hapus timestamp invalid
df = df.dropna(subset=["timestamp"])

# 3. Urutkan waktu
df = df.sort_values("timestamp")

# 4. Hapus duplikasi waktu
df = df.drop_duplicates(subset=["timestamp"], keep="last")

# 5. Set index waktu
df = df.set_index("timestamp")

# 6. Resample ke 10 menit
df_resampled = df.resample("10min").mean(numeric_only=True)
df_resampled["device_id"] = "room01"

# 7. Imputasi sederhana
numeric_cols = ["temperature_c", "humidity_pct", "co2_ppm",
"power_watt"]
df_resampled[numeric_cols] =
df_resampled[numeric_cols].interpolate(method="linear")

# 8. Simpan
df_resampled.to_csv(OUTPUT_PATH)

print("=== Data hasil cleaning ===")
print(df_resampled.head(15))
print(f"\nDisimpan ke: {OUTPUT_PATH}")
```

10. Langkah kerja yang harus dilakukan pembaca

Tahap	Kegiatan
1	Baca file kotor.
2	Parse timestamp.
3	Buang waktu invalid.
4	Urutkan data dan hapus duplikasi.
5	Set index waktu.
6	Lakukan resample ke interval target.
7	Lakukan imputasi sederhana.
8	Simpan hasil ke folder processed.

11. Penutup Hands-on 7

Dengan latihan ini, pembaca belajar bahwa **data sensor yang baik hampir selalu dibuat, bukan ditemukan dalam keadaan sempurna**. Membersihkan waktu, menyamakan interval, dan mengisi nilai hilang adalah pekerjaan inti seorang praktisi IoT analytics. Tanpa ini, grafik bisa menipu, statistik bisa bias, dan model bisa belajar pola yang salah. Itulah mengapa *time series handling* sangat penting dalam analitik IoT.

Hands-on 8 - Deteksi Anomali Dasar dengan Aturan Statistik

1. Tujuan Hands-on

Pada latihan ini, pembaca akan mendeteksi anomali sederhana pada data sensor menggunakan aturan statistik dasar, tanpa model ML kompleks. Setelah selesai, pembaca diharapkan mampu:

- memahami anomali sebagai penyimpangan dari pola normal,
- menggunakan ambang domain,
- menggunakan **Z-score**,
- menggunakan **IQR**,
- menandai data mencurigakan,
- dan membuat visualisasi titik anomali.

Ini penting karena pada banyak sistem IoT, anomali awal justru dapat dideteksi dengan metode ringan sebelum memakai model seperti Isolation Forest atau One-Class SVM yang nanti dibahas di bab-bab berikutnya.

2. Contoh nyata yang bisa diimplementasikan

Ruang server

Anomali suhu bisa berarti AC gagal, pintu server room terbuka terlalu lama, atau sensor bermasalah.

Energi gedung

Anomali daya bisa menunjukkan alat rusak, pemakaian di luar jam operasional, atau beban tidak wajar.

Greenhouse

Anomali kelembaban tanah bisa menandakan pompa macet atau sensor lepas.

Mesin kecil

Lonjakan getaran bisa menjadi *warning* awal sebelum kerusakan serius.

Di dunia IoT analytics, penggunaan analitik untuk membedakan pola normal dan tidak normal adalah salah satu langkah penting sebelum keputusan operasional diambil.

3. Dataset untuk latihan anomali

Simpan file berikut sebagai:

```
data/raw/room_sensor_anomaly.csv
```

Isi:

```
timestamp,device_id,temperature_c,humidity_pct,co2_ppm,power_watt
2026-01-01 08:00:00,room01,24.5,58,520,110
2026-01-01 08:10:00,room01,24.7,57,525,112
2026-01-01 08:20:00,room01,25.1,56,540,115
2026-01-01 08:30:00,room01,25.8,55,560,118
2026-01-01 08:40:00,room01,26.2,54,590,121
2026-01-01 08:50:00,room01,26.8,53,620,126
2026-01-01 09:00:00,room01,27.1,52,650,130
2026-01-01 09:10:00,room01,27.6,51,680,135
2026-01-01 09:20:00,room01,28.1,50,710,139
2026-01-01 09:30:00,room01,45.0,49,1500,300
2026-01-01 09:40:00,room01,28.8,48,770,147
2026-01-01 09:50:00,room01,29.0,47,790,150
2026-01-01 10:00:00,room01,29.2,46,810,152
2026-01-01 10:10:00,room01,29.0,47,800,150
2026-01-01 10:20:00,room01,28.7,48,780,146
2026-01-01 10:30:00,room01,28.3,49,750,142
```

Baris pukul **09:30** sengaja dibuat abnormal:

- suhu melonjak,
- CO_2 melonjak,
- daya melonjak.

4. Metode 1 — Aturan domain sederhana

Buat file:

```
scripts/detect_anomaly_rules.py
```

Isi:

```
from pathlib import Path
import pandas as pd

DATA_PATH = Path("data/raw/room_sensor_anomaly.csv")

df = pd.read_csv(DATA_PATH)
df["timestamp"] = pd.to_datetime(df["timestamp"])

# Aturan domain
df["anomaly_rule"] = (
    (df["temperature_c"] > 35) |
    (df["co2_ppm"] > 1200) |
    (df["power_watt"] > 250)
)

print(df[["timestamp", "temperature_c", "co2_ppm", "power_watt",
"anomaly_rule"]])
```

Mengapa aturan domain penting?

Karena di banyak implementasi nyata, alarm pertama justru dibuat dari batas wajar operasional:

- suhu > 35°C,
- CO_2 > 1200 ppm,
- daya > 250 watt.

Metode ini:

- sederhana,
- cepat,
- mudah dijelaskan ke teknisi,
- dan sangat cocok untuk tahap awal.

5. Metode 2 — Z-score

Z-score mengukur seberapa jauh suatu nilai dari rata-rata dalam satuan deviasi standar.

Tambahkan atau buat file baru:

```
from pathlib import Path
import pandas as pd
import numpy as np

DATA_PATH = Path("data/raw/room_sensor_anomaly.csv")

df = pd.read_csv(DATA_PATH)
df["timestamp"] = pd.to_datetime(df["timestamp"])

def zscore(series: pd.Series) -> pd.Series:
    return (series - series.mean()) / series.std(ddof=0)

df["z_temp"] = zscore(df["temperature_c"])
df["z_co2"] = zscore(df["co2_ppm"])
df["z_power"] = zscore(df["power_watt"])

df["anomaly_zscore"] = (
    (df["z_temp"].abs() > 3) |
    (df["z_co2"].abs() > 3) |
    (df["z_power"].abs() > 3)
)

print(df[[
    "timestamp", "temperature_c", "z_temp",
    "co2_ppm", "z_co2",
    "power_watt", "z_power",
    "anomaly_zscore"
]])
```

Kapan metode ini cocok?

- ketika distribusi data relatif stabil,
- saat ingin mendeteksi lonjakan yang benar-benar jauh dari pola umum,
- untuk latihan awal statistik.

6. Metode 3 — IQR

Metode **IQR** lebih tahan terhadap nilai ekstrim daripada rata-rata biasa.

from pathlib import Path

```
import pandas as pd

DATA_PATH = Path("data/raw/room_sensor_anomaly.csv")

df = pd.read_csv(DATA_PATH)
df["timestamp"] = pd.to_datetime(df["timestamp"])

def iqr_bounds(series: pd.Series):
    q1 = series.quantile(0.25)
    q3 = series.quantile(0.75)
    iqr = q3 - q1
    lower = q1 - 1.5 * iqr
    upper = q3 + 1.5 * iqr
    return lower, upper

temp_lower, temp_upper = iqr_bounds(df["temperature_c"])
co2_lower, co2_upper = iqr_bounds(df["co2_ppm"])
power_lower, power_upper = iqr_bounds(df["power_watt"])

df["anomaly_iqr"] = (
    (df["temperature_c"] < temp_lower) | (df["temperature_c"] >
temp_upper) |
    (df["co2_ppm"] < co2_lower) | (df["co2_ppm"] > co2_upper) |
    (df["power_watt"] < power_lower) | (df["power_watt"] >
power_upper)
)

print(df[[
    "timestamp", "temperature_c", "co2_ppm", "power_watt",
"anomaly_iqr"
]])
```

7. Visualisasi anomali

Buat file:

```
scripts/plot_anomaly.py
```

Isi:

```
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt

DATA_PATH = Path("data/raw/room_sensor_anomaly.csv")
OUTPUT_DIR = Path("outputs/figures")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

df = pd.read_csv(DATA_PATH)
df["timestamp"] = pd.to_datetime(df["timestamp"])

# Aturan domain
df["anomaly_rule"] = (
    (df["temperature_c"] > 35) |
    (df["co2_ppm"] > 1200) |
    (df["power_watt"] > 250)
)

plt.figure(figsize=(10, 5))
plt.plot(df["timestamp"], df["temperature_c"], marker="o",
label="Temperature")
plt.scatter(
    df.loc[df["anomaly_rule"], "timestamp"],
    df.loc[df["anomaly_rule"], "temperature_c"],
    marker="x",
    s=100,
    label="Anomaly"
)
plt.title("Deteksi Anomali Suhu dengan Aturan Domain")
plt.xlabel("Waktu")
plt.ylabel("Suhu (C)")
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.savefig(OUTPUT_DIR / "anomaly_temperature_rule.png")
plt.show()
```

8. Versi gabungan: satu skrip lengkap

Berikut skrip yang menggabungkan tiga metode:

```
from pathlib import Path
import pandas as pd

DATA_PATH = Path("data/raw/room_sensor_anomaly.csv")

df = pd.read_csv(DATA_PATH)
df["timestamp"] = pd.to_datetime(df["timestamp"])

# Rule-based
df["anomaly_rule"] = (
    (df["temperature_c"] > 35) |
    (df["co2_ppm"] > 1200) |
    (df["power_watt"] > 250)
)

# Z-score
def zscore(series):
    return (series - series.mean()) / series.std(ddof=0)

df["z_temp"] = zscore(df["temperature_c"])
df["z_co2"] = zscore(df["co2_ppm"])
df["z_power"] = zscore(df["power_watt"])

df["anomaly_zscore"] = (
    (df["z_temp"].abs() > 3) |
    (df["z_co2"].abs() > 3) |
    (df["z_power"].abs() > 3)
)

# IQR
def iqr_bounds(series):
    q1 = series.quantile(0.25)
    q3 = series.quantile(0.75)
    iqr = q3 - q1
    return q1 - 1.5 * iqr, q3 + 1.5 * iqr

temp_lower, temp_upper = iqr_bounds(df["temperature_c"])
co2_lower, co2_upper = iqr_bounds(df["co2_ppm"])
power_lower, power_upper = iqr_bounds(df["power_watt"])

df["anomaly_iqr"] = (
    (df["temperature_c"] < temp_lower) | (df["temperature_c"] > temp_upper)
|
    (df["co2_ppm"] < co2_lower) | (df["co2_ppm"] > co2_upper) |
    (df["power_watt"] < power_lower) | (df["power_watt"] > power_upper)
)

print(df[[
    "timestamp",
    "temperature_c", "co2_ppm", "power_watt",
    "anomaly_rule", "anomaly_zscore", "anomaly_iqr"
]])
```

9. Cara membaca hasil

Jika satu titik terdeteksi oleh:

- **aturan domain** → berarti melanggar ambang operasional,
- **Z-score** → berarti jauh dari distribusi normal,
- **IQR** → berarti di luar rentang data mayoritas.

Praktik terbaik

Titik yang muncul di ketiga metode sekaligus layak diperiksa lebih serius.

10. Langkah kerja yang harus dilakukan pembaca

Tahap	Aktivitas	Tujuan
1	Baca dataset.	Memuat data agar siap diperiksa, dibersihkan, dan dianalisis lebih lanjut.
2	Tentukan ambang domain.	Menetapkan batas normal berdasarkan pengetahuan proses, konteks operasional, atau aturan bisnis.
3	Tandai anomali berbasis aturan.	Mengidentifikasi data yang melampaui ambang domain sebagai indikasi awal anomali.
4	Hitung Z-score.	Mengukur seberapa jauh suatu nilai menyimpang dari rata-rata dalam satuan simpangan baku.
5	Hitung batas IQR.	Menentukan outlier berdasarkan sebaran data menggunakan kuartil dan *interquartile range*.
6	Bandingkan hasil ketiga metode.	Melihat konsistensi dan perbedaan temuan antara aturan domain, Z-score, dan IQR.
7	Buat visualisasi.	Menampilkan pola, penyimpangan, dan titik anomali agar lebih mudah dipahami secara visual.
8	Tulis interpretasi operasional.	Menjelaskan makna anomali bagi proses nyata serta kemungkinan tindak lanjut yang perlu dilakukan.

11. Contoh interpretasi real

Ruang server

Jika suhu, CO_2, dan daya naik bersamaan, kemungkinan:

- ruangan ramai,
- pendingin bermasalah,
- ventilasi tidak cukup.

Greenhouse

Jika kelembaban tanah turun tajam tetapi suhu tetap, kemungkinan:

- pompa tidak bekerja,
- sensor tanah bermasalah,
- atau ada perubahan kondisi media tanam.

Energi gedung

Jika daya melonjak di luar jam kerja, kemungkinan:

- alat lupa dimatikan,
- ada beban tidak sah,
- atau proses tertentu berjalan di luar jadwal.

12. Penutup Hands-on 8

Latihan ini menunjukkan bahwa **deteksi anomali tidak harus langsung rumit**. Dengan aturan domain, Z-score, dan IQR, pembaca sudah bisa membuat sistem peringatan awal yang sangat berguna. Ini adalah fondasi yang kuat sebelum melangkah ke *anomaly detection* berbasis ML pada bab-bab berikutnya. Dalam semangat buku ini, pembaca diajak membangun solusi yang **ringan, terbuka, mudah dijelaskan, dan bisa langsung dipakai di Ubuntu 26.04** untuk kasus nyata di kampus, laboratorium, komunitas, atau industri kecil.

Bab 4. Membangun Dummy Data dan Synthetic Data untuk IoT

4.1 Mengapa dummy data penting untuk pembelajaran, pengujian pipeline, dan prototyping?

Dalam proyek IoT nyata, kita sering menghadapi situasi yang tidak ideal. Sensor belum datang. Sensor datang tetapi belum stabil. Perangkat masih sering *restart*. Jaringan Wi-Fi belum rapi. Data yang terkumpul masih terlalu sedikit. Bahkan ketika semua alat sudah terpasang, kadang data lapangan belum cukup kaya untuk melatih model atau menguji semua cabang logika sistem. Di sinilah **dummy data** dan **synthetic data** menjadi sangat penting.

Bagi mahasiswa, data sintetik memungkinkan pembelajaran dimulai **hari ini**, tanpa menunggu laboratorium sempurna. Bagi praktisi, data sintetik membantu menguji apakah alur

sensor → broker → storage → analytics → model → dashboard

benar-benar berjalan dengan baik. Bagi pengembang sistem, data sintetik sangat berguna untuk *stress testing*, simulasi *fault*, dan validasi awal desain sebelum diterapkan di lapangan. Hal ini sejalan dengan arah buku Anda yang sangat menekankan sistem **open source**, eksperimen Python yang **repeatable**, dan jalur praktis dari **sensor ke keputusan cerdas**.

Secara praktis, data sintetik sangat membantu dalam beberapa kondisi berikut:

- **Pembelajaran awal**
  Saat pembaca baru mulai belajar pandas, visualisasi, MQTT, atau *machine learning*, data sintetik jauh lebih aman karena strukturnya bisa dikendalikan.
- **Pengujian pipeline**
  Kita dapat mengirim ribuan baris data sintetis ke MQTT, menyimpan ke SQLite atau InfluxDB, lalu memeriksa apakah pipeline berjalan stabil.
- **Pengujian alarm dan anomali**
  Kita bisa sengaja menambahkan lonjakan suhu, CO_2, daya, atau getaran untuk melihat apakah sistem alarm bekerja.
- **Pengujian model ML**
  Untuk banyak use case IoT, label kejadian abnormal sangat jarang. Dengan data sintetik, kita bisa menambahkan contoh *warning* atau *fault* agar eksperimen awal lebih seimbang.
- **Prototyping cepat**
  Sebelum sensor sungguhan dipasang, dashboard, API, *notebook*, dan model awal sudah bisa dibuat.

Dengan kata lain, **data sintetik bukan pengganti sensor**, tetapi **percepatan belajar dan percepatan rekayasa sistem**.

4.2 Perbedaan: *dummy data*, *simulated data*, *synthetic data*, dan *augmented data*

Empat istilah ini sering dipakai bercampur, padahal sebenarnya ada perbedaan yang penting. Untuk buku ini, perbedaannya perlu dibuat sederhana tetapi tegas.

4.2.1 *Dummy data*

Dummy data adalah data buatan yang dibuat terutama untuk **mengisi struktur** atau **mendemonstrasikan alur dasar**. Biasanya *dummy data* belum terlalu peduli pada realisme tinggi. Tujuannya adalah agar file CSV, tabel database, atau payload MQTT memiliki bentuk yang bisa langsung diuji.

Contoh:

- `temperature_c = 25.0`, `humidity_pct = 60`, `timestamp = 2026-01-01 08:00:00`
- 20 baris data ruangan untuk latihan `pandas.read_csv()`
- payload MQTT contoh untuk uji subscriber

Dummy data cocok untuk:

- belajar format file,
- uji tampilan dashboard,
- uji parsing JSON,
- uji alur dasar publish-subscribe.

4.2.2 Simulated data

Simulated data adalah data buatan yang dibangun dari **aturan proses** atau **mekanisme sederhana yang meniru dunia nyata**. Tingkat realisme lebih tinggi daripada *dummy data*. Biasanya ada asumsi fisik, pola perilaku, atau hubungan sebab-akibat di dalamnya.

Contoh:

- suhu ruangan naik saat jam kerja dan turun malam hari,
- konsumsi energi melonjak saat jam sibuk,
- kelembaban tanah turun perlahan jika tidak disiram,
- getaran motor meningkat ketika fault diinjeksikan.

Simulated data cocok untuk:

- prototipe kontrol,
- simulasi monitoring,
- pengujian pola waktu,
- eksperimen *what-if*.

4.2.3 *Synthetic data*

Synthetic data adalah data buatan yang secara statistik atau struktural dirancang agar **mirip dengan data nyata**, walaupun tidak berasal langsung dari pengukuran lapangan. Fokusnya bukan hanya bentuk file, tetapi juga distribusi, korelasi, tren, musiman, dan noise yang “cukup realistis” untuk analitik atau pelatihan model sederhana.

Contoh:

- dataset suhu-kelembaban mingguan dengan pola harian, *noise*, *missing value*, dan *drift* kecil,
- dataset konsumsi energi gedung dengan pola jam sibuk dan akhir pekan,
- dataset getaran mesin dengan distribusi normal untuk kondisi sehat dan distribusi lain untuk kondisi rusak.

Synthetic data cocok untuk:

- eksperimen ML,
- evaluasi preprocessing,
- validasi statistik awal,
- benchmarking pipeline.

4.2.4 *Augmented data*

Augmented data adalah data yang berasal dari data yang sudah ada, lalu **diperluas atau dimodifikasi** untuk menambah variasi. Pada data sensor, *augmentation* bisa dilakukan dengan:

- menambahkan noise kecil,
- menggeser waktu,
- menskalakan amplitudo,
- menambah variasi musiman,
- atau menyuntikkan anomali.

Contoh:

- data suhu asli ditambah variasi noise ±0.3°C,
- sinyal getaran asli digeser dan diberi perubahan amplitudo,
- data energi asli dibuat versi “hari sibuk” dan “hari sepi”.

Untuk konteks buku ini, pendekatan yang paling penting adalah **dummy data** → **simulated data** → **synthetic data**, sementara *augmented data* dapat dikenalkan sebagai langkah tambahan ketika pembaca sudah memiliki data awal.

4.3 Mengapa Bab 4 sangat penting dalam alur buku ini?

Pola pikir dalam buku ini bergerak dari:

1. pemahaman arsitektur dan data sensor,
2. eksplorasi dan pembersihan data,
3. lalu masuk ke *feature engineering* dan *machine learning*.

Tanpa Bab 4, kita bisa merasa terlalu bergantung pada:

- dataset publik,
- sensor fisik,
- atau data yang belum tentu tersedia.

Dengan Bab 4, kita memperoleh **kemerdekaan eksperimen**. Kita bisa:

- membangun data sendiri,
- mengontrol skenario normal dan abnormal,
- menguji pipeline secara berulang,
- dan menyiapkan fondasi untuk studi kasus seperti **Smart Room Monitoring**, **Predictive Maintenance Sederhana**, dan **Smart Energy Analytics** yang memang menjadi inti studi kasus buku ini.

4.4 Sumber pembentukan data sintetik

Agar data sintetik berguna, pembaca perlu tahu dari mana pola itu dibangun. Untuk buku ini, ada empat sumber pembentukan data sintetik yang paling praktis.

4.4.1 Aturan fisik sederhana

Sumber pertama adalah **aturan fisik sederhana**. Kita tidak perlu simulasi ilmiah yang rumit. Cukup gunakan logika yang masuk akal.

Contoh:

- suhu ruangan cenderung naik saat siang dan turun malam,
- kelembaban relatif sering turun saat suhu naik,
- konsumsi listrik meningkat saat banyak perangkat menyala,
- getaran mesin sehat relatif stabil, sedangkan mesin fault menunjukkan amplitudo lebih besar.

Kelebihan pendekatan ini adalah:

- mudah dipahami,
- mudah dijelaskan,
- dan sangat cocok untuk mahasiswa.

4.4.2 Distribusi statistik

Sumber kedua adalah **distribusi statistik**. Nilai sensor bisa dibangkitkan dari distribusi tertentu:

- suhu dari distribusi normal dengan rata-rata tertentu,
- konsumsi energi dari distribusi yang bergeser tergantung jam,
- getaran dari distribusi normal untuk kondisi sehat dan distribusi lain untuk kondisi fault.

Pendekatan ini sangat cocok saat kita ingin membuat data yang:

- punya variasi,
- tidak terlalu deterministik,
- dan lebih realistis daripada angka konstan.

4.4.3 Pola periodik

Banyak data IoT memiliki pola harian, mingguan, atau musiman. Literatur juga menekankan bahwa data sensor IoT umumnya berbentuk *time series*, dan *seasonality* merupakan salah satu pola penting yang perlu dikenali.

Contoh:

- suhu ruangan naik mulai pagi, puncak di siang, turun sore,
- energi gedung tinggi pada jam kerja, rendah malam,
- kelembaban tanah turun perlahan setelah penyiraman,
- suara kota meningkat pada jam sibuk.

4.4.4 Skenario alarm atau *fault*

Sumber keempat adalah **skenario alarm/fault**. Ini sangat penting untuk IoT karena banyak sistem dipakai justru untuk mendeteksi kejadian langka:

- suhu terlalu tinggi,
- CO_2 melewati ambang,
- daya melonjak tidak wajar,
- getaran mesin meningkat,
- nilai sensor hilang tiba-tiba.

Tanpa menyuntikkan skenario seperti ini, banyak pipeline terlihat “berhasil” padahal belum pernah diuji pada kondisi yang benar-benar bermasalah.

4.5 Membangun data IoT sintetis dengan Python

Seluruh contoh di bawah memakai **Python** pada **Ubuntu 26.04**, dengan pustaka **open source**:

- `pandas`
- `numpy`
- `matplotlib`

Aktifkan *virtual environment*:

```
source .venv/bin/activate
pip install pandas numpy matplotlib
```

Buat struktur folder:

```
mkdir -p data/raw data/synthetic scripts outputs/figures
```

4.5.1 Contoh 1 — Suhu dan kelembaban berbasis pola harian

Kasus ini sangat realistis untuk:

- ruang kelas,
- ruang server,
- laboratorium,
- greenhouse ringan.

Logika yang dipakai

- suhu mengikuti pola harian,
- kelembaban cenderung berlawanan arah dengan suhu,
- ada sedikit noise,
- data dikirim tiap 10 menit.

Buat file `scripts/generate_room_synthetic.py`:

```
from pathlib import Path
import numpy as np
import pandas as pd

OUTPUT_PATH = Path("data/synthetic/room_daily_synthetic.csv")
OUTPUT_PATH.parent.mkdir(parents=True, exist_ok=True)

# Waktu 1 hari, interval 10 menit
timestamps = pd.date_range(
    start="2026-01-01 00:00:00",
    end="2026-01-01 23:50:00",
    freq="10min"
)

hours = timestamps.hour + timestamps.minute / 60.0

# Pola suhu harian: rendah malam, naik siang
temperature = 24 + 4 * np.sin((hours - 6) / 24 * 2 * np.pi)
temperature += np.random.normal(0, 0.3, len(timestamps))

# kelembaban berlawanan dengan suhu
humidity = 70 - 10 * np.sin((hours - 6) / 24 * 2 * np.pi)
humidity += np.random.normal(0, 1.5, len(timestamps))

df = pd.DataFrame({
    "timestamp": timestamps,
    "device_id": "room01",
    "temperature_c": np.round(temperature, 2),
    "humidity_pct": np.round(humidity, 2)
})

df.to_csv(OUTPUT_PATH, index=False)
print(f"Data sintetik disimpan ke: {OUTPUT_PATH}")
print(df.head())
```

Mengapa contoh ini bagus?

Karena pembaca langsung melihat bahwa data sintetik tidak harus acak total. Ia bisa punya **pola harian** yang sangat masuk akal untuk sistem IoT.

4.5.2 Contoh 2 — Konsumsi energi dengan pola jam sibuk

Kasus ini cocok untuk:

- laboratorium,
- ruang komputer,
- gedung kecil,
- bengkel.

Logika yang dipakai

- beban rendah di malam hari,
- beban tinggi saat jam kerja,
- puncak lebih tinggi saat siang,
- akhir hari kembali turun.

Buat file `scripts/generate_energy_synthetic.py`:

```
from pathlib import Path
import numpy as np
import pandas as pd

OUTPUT_PATH = Path("data/synthetic/energy_daily_synthetic.csv")
OUTPUT_PATH.parent.mkdir(parents=True, exist_ok=True)

timestamps = pd.date_range(
    start="2026-01-01 00:00:00",
    end="2026-01-01 23:50:00",
    freq="10min"
)

power = []
for ts in timestamps:
    hour = ts.hour

    if 0 <= hour < 6:
        base = 120
    elif 6 <= hour < 8:
        base = 180
    elif 8 <= hour < 12:
        base = 350
    elif 12 <= hour < 14:
        base = 450
    elif 14 <= hour < 18:
        base = 320
    else:
        base = 170

    value = base + np.random.normal(0, 15)
    power.append(value)

df = pd.DataFrame({
    "timestamp": timestamps,
    "device_id": "lab-energy-01",
    "power_watt": np.round(power, 2)
})

df.to_csv(OUTPUT_PATH, index=False)
print(f"Data energi sintetik disimpan ke: {OUTPUT_PATH}")
print(df.head())
```

Implementasi nyata

Dataset ini bisa dipakai untuk:

- dashboard energi harian,
- pengujian alarm beban puncak,
- latihan *forecasting* sederhana,
- deteksi “pemakaian di luar jam kerja”.

4.5.3 Contoh 3 — Getaran mesin normal vs *fault*

Kasus ini sangat relevan untuk **predictive maintenance sederhana**, yang memang menjadi salah satu studi kasus utama.

Logika yang dipakai

- kondisi normal: amplitudo kecil, noise rendah,
- kondisi fault: amplitudo lebih besar, variasi lebih kasar.

Buat file `scripts/generate_vibration_synthetic.py`:

```
from pathlib import Path
import numpy as np
import pandas as pd

OUTPUT_PATH = Path("data/synthetic/vibration_machine_synthetic.csv")
OUTPUT_PATH.parent.mkdir(parents=True, exist_ok=True)

n = 1000
t = np.arange(n)

# Sinyal normal
normal_signal = 0.8 * np.sin(2 * np.pi * 0.03 * t) +
np.random.normal(0, 0.1, n)

# Sinyal fault
fault_signal = 1.6 * np.sin(2 * np.pi * 0.03 * t) +
np.random.normal(0, 0.25, n)

df_normal = pd.DataFrame({
    "sample_id": range(n),
    "machine_id": "motor01",
    "vibration": np.round(normal_signal, 4),
    "label": "normal"
})

df_fault = pd.DataFrame({
    "sample_id": range(n, 2*n),
    "machine_id": "motor01",
    "vibration": np.round(fault_signal, 4),
    "label": "fault"
})

df = pd.concat([df_normal, df_fault], ignore_index=True)
df.to_csv(OUTPUT_PATH, index=False)
```

```
print(f"Data getaran sintetik disimpan ke: {OUTPUT_PATH}")
print(df.head())
print(df.tail())
```

Mengapa penting?

Karena data *fault* nyata sering jarang. Dengan data sintetik ini, pembaca dapat:

- menguji fitur statistik,
- mencoba klasifikasi awal,
- membangun *threshold*,
- dan memahami perbedaan sinyal sehat vs bermasalah.

4.5.4 Menambahkan *noise* pada data sensor

Literatur menegaskan bahwa **veracity** data IoT tidak selalu tinggi; data IoT dapat mengandung **noise** dan **error**.

Contoh sederhana menambahkan noise:

```
import numpy as np

clean_temp = np.array([25.0, 25.2, 25.4, 25.3, 25.5])
noisy_temp = clean_temp + np.random.normal(0, 0.2, len(clean_temp))

print("Data asli :", clean_temp)
print("Data noisy:", np.round(noisy_temp, 2))
```

Manfaat noise buatan

- menguji robust tidaknya algoritma,
- membuat data terlihat lebih realistis,
- menghindari dataset yang “terlalu sempurna”.

4.5.5 Menambahkan *missing value*, *drift*, dan *outlier* buatan

Bab sebelumnya sudah menekankan bahwa data IoT nyata sering menghadapi:

- *missing value*,
- *drift*,
- *outlier*,
- dan interval tidak konsisten.

Contoh menambahkan kerusakan buatan pada data suhu

Buat file `scripts/inject_data_issues.py`:

```
from pathlib import Path
import numpy as np
import pandas as pd

INPUT_PATH = Path("data/synthetic/room_daily_synthetic.csv")
OUTPUT_PATH =
Path("data/synthetic/room_daily_synthetic_corrupted.csv")

df = pd.read_csv(INPUT_PATH)

# 1. Missing value acak
missing_idx = np.random.choice(df.index, size=8, replace=False)
df.loc[missing_idx, "temperature_c"] = np.nan

# 2. Drift bertahap pada suhu
df["temperature_c"] = df["temperature_c"] + np.linspace(0, 1.2,
len(df))

# 3. Outlier buatan
outlier_idx = np.random.choice(df.index, size=3, replace=False)
df.loc[outlier_idx, "temperature_c"] = df.loc[outlier_idx,
"temperature_c"] + 8

df.to_csv(OUTPUT_PATH, index=False)

print(f"Data dengan missing value, drift, dan outlier disimpan ke:
{OUTPUT_PATH}")
print("Index missing:", missing_idx)
print("Index outlier:", outlier_idx)
```

Kegunaan praktis

Dataset semacam ini sangat bagus untuk:

- latihan *data cleaning*,
- deteksi anomali,
- uji *resampling* dan imputasi,
- pengujian pipeline operasional.

4.5.6 Menambahkan label sintetis

Salah satu kekuatan data sintetik adalah kita bisa menambahkan label dengan cara yang terkontrol. Ini sangat penting karena banyak use case IoT memiliki label yang langka.

Label yang relevan untuk buku ini

- **normal / warning / fault**
- **nyaman / tidak nyaman**
- **hemat / boros**

Contoh 1 — Label nyaman / tidak nyaman

Untuk sistem ruangan, kita bisa membuat aturan sederhana:

- nyaman jika suhu 24–28°C dan kelembaban 45–65%
- selain itu: tidak nyaman

```
from pathlib import Path
import pandas as pd

INPUT_PATH = Path("data/synthetic/room_daily_synthetic.csv")
OUTPUT_PATH = Path("data/synthetic/room_daily_labeled.csv")

df = pd.read_csv(INPUT_PATH)

df["comfort_label"] = df.apply(
    lambda row: "nyaman"
    if (24 <= row["temperature_c"] <= 28 and 45 <= row["humidity_pct"]
<= 65)
    else "tidak_nyaman",
    axis=1
)

df.to_csv(OUTPUT_PATH, index=False)
print(df.head())
```

Contoh 2 — Label normal / warning / fault

Untuk energi atau suhu, gunakan tiga tingkat:

from pathlib import Path
import pandas as pd

```
INPUT_PATH = Path("data/synthetic/energy_daily_synthetic.csv")
OUTPUT_PATH = Path("data/synthetic/energy_daily_labeled.csv")

df = pd.read_csv(INPUT_PATH)

def label_energy(power):
    if power < 250:
        return "normal"
    elif power < 400:
        return "warning"
    return "fault"

df["status_label"] = df["power_watt"].apply(label_energy)

df.to_csv(OUTPUT_PATH, index=False)
print(df.head())
```

Contoh 3 — Label hemat / boros

Untuk konsumsi energi harian:

from pathlib import Path
import pandas as pd

```
INPUT_PATH = Path("data/synthetic/energy_daily_synthetic.csv")
OUTPUT_PATH = Path("data/synthetic/energy_daily_efficiency.csv")

df = pd.read_csv(INPUT_PATH)

mean_power = df["power_watt"].mean()

df["efficiency_label"] = df["power_watt"].apply(
    lambda x: "hemat" if x <= mean_power else "boros"
)

df.to_csv(OUTPUT_PATH, index=False)
print(df.head())
```

Catatan penting

Label sintetis ini **bukan kebenaran absolut**, tetapi **alat belajar**. Ia membantu pembaca memahami hubungan antara aturan domain, data sensor, dan target klasifikasi.

4.6 Contoh-contoh real yang bisa diimplementasikan

Agar bab ini terasa sangat hidup, berikut beberapa contoh implementasi nyata yang bisa langsung dipakai dalam praktikum.

4.6.1 Smart room monitoring

Gunakan data sintetik suhu, kelembaban, dan CO_2 untuk:

- menguji dashboard Grafana,
- melatih klasifikasi `nyaman / tidak nyaman`,
- menguji alarm ventilasi,
- mempelajari pola harian ruang kelas atau ruang server.

Literatur juga menunjukkan bahwa di lingkungan indoor, sensor yang paling sering dipakai adalah **temperature**, **humidity**, dan **CO_2**, sehingga contoh ini sangat realistis.

4.6.2 Smart energy analytics

Gunakan data energi sintetis untuk:

- melihat pola jam sibuk,
- membangun alarm `warning / fault`,
- menguji pipeline MQTT → database → dashboard,
- melatih *forecasting* sederhana.

4.6.3 Predictive maintenance sederhana

Gunakan data getaran sintetis untuk:

- membedakan kondisi normal vs fault,
- menghitung fitur statistik,
- menguji model klasifikasi ringan,
- mendemonstrasikan *anomaly detection*.

4.6.4 Greenhouse atau pertanian presisi

Gunakan data suhu, kelembaban udara, kelembaban tanah, dan cahaya sintetis untuk:

- simulasi irigasi,
- pengujian aturan `normal / warning`,
- pengujian dashboard lokal.

Referensi tentang pertanian presisi dalam materi Anda juga menunjukkan bahwa sensor kelembaban tanah dan sistem irigasi berbasis IoT merupakan use case yang sangat relevan untuk pembelajaran dan eksperimen.

4.7 Validasi sederhana: kapan data sintetik “cukup realistis”?

Ini pertanyaan penting. Data sintetik tidak harus sempurna seperti data lapangan, tetapi ia harus **cukup realistis** untuk tujuan yang dimaksud.

Data sintetik bisa dianggap cukup realistis bila:

- **struktur kolomnya masuk akal**
  Ada `timestamp`, `device_id`, nilai sensor, dan label jika perlu.
- **rentang nilainya wajar**
  Suhu ruang tidak tiba-tiba -50°C atau 200°C kecuali memang untuk uji ekstrem.
- **pola waktunya masuk akal**
  Ada ritme harian, tren kecil, atau pola beban yang sesuai use case.
- **ada variasi**
  Data tidak terlalu datar.
- **korelasi sederhana sesuai domain**
  Misalnya suhu naik, kelembaban turun; aktivitas naik, CO_2 naik; jam sibuk, daya naik.
- **mengandung gangguan realistis**
  Sejumlah noise, beberapa *missing value*, atau anomali yang disengaja.
- **bisa mendukung tujuan eksperimen**
  Misalnya cukup baik untuk uji dashboard, uji broker, atau latihan klasifikasi awal.

Cara validasi sederhana dengan Python

```
from pathlib import Path
import pandas as pd

DATA_PATH = Path("data/synthetic/room_daily_synthetic.csv")
df = pd.read_csv(DATA_PATH)

print("=== Statistik dasar ===")
print(df.describe())

print("\n=== Korelasi ===")
print(df[["temperature_c", "humidity_pct"]].corr())

print("\n=== Cek nilai ekstrem ===")
print("Suhu min:", df["temperature_c"].min())
print("Suhu max:", df["temperature_c"].max())
print("Humidity min:", df["humidity_pct"].min())
print("Humidity max:", df["humidity_pct"].max())
```

Jika statistiknya masuk akal dan grafiknya terlihat seperti perilaku sistem nyata, maka untuk kebutuhan pembelajaran awal data tersebut sudah cukup baik.

4.8 Risiko dan batasan: data sintetik tidak boleh dianggap pengganti penuh data lapangan

Ini bagian yang sangat penting. **Data sintetik tidak boleh dianggap sebagai pengganti penuh data lapangan.**

Mengapa?

Karena dunia nyata selalu lebih rumit daripada yang kita simulasikan.

Beberapa keterbatasannya:

- sensor nyata punya bias dan karakter fisik yang tidak selalu sederhana,
- jaringan dan perangkat bisa gagal dengan pola yang tidak terduga,
- perilaku manusia dan lingkungan sering tidak linier,
- distribusi data lapangan bisa berubah seiring waktu,
- *fault* nyata sering lebih kompleks daripada injeksi anomali sederhana.

Literatur juga mengingatkan bahwa data IoT memiliki sifat **vagueness**, **incompleteness**, **different speeds**, dan **low-value density**, yang tidak selalu mudah ditiru sempurna oleh data sintetik.

Maka, gunakan data sintetik untuk:

- belajar,
- menguji pipeline,
- menguji dashboard,
- membuat prototipe,
- menguji logika,
- membangun intuisi,
- membuat model baseline.

Tetapi tetap butuh data lapangan untuk:

- validasi akhir,
- kalibrasi model,
- evaluasi performa nyata,
- tuning ambang operasional,
- dan pengambilan keputusan produksi.

Dengan kata lain:
data sintetik mempercepat langkah awal, tetapi data lapangan tetap menjadi ujian sebenarnya.

4.9 Langkah kerja yang harus dilakukan pembaca

Agar bab ini benar-benar operasional, pembaca dapat mengikuti tahapan berikut.

Tahap 1 — Pilih use case

Pilih salah satu:

- smart room,
- energi,
- getaran mesin,
- greenhouse.

Tahap 2 — Tentukan variabel inti

Contoh:

- suhu, kelembaban
- daya listrik
- getaran
- kelembaban tanah

Tahap 3 — Tentukan pola dasar

Apakah ada:

- pola harian,
- jam sibuk,
- perilaku periodik,
- hubungan antar variabel?

Tahap 4 — Tambahkan noise dan variasi

Jangan membuat data terlalu sempurna.

Tahap 5 — Tambahkan masalah realistis

Sisipkan:

- *missing value*
- *outlier*
- *drift*
- label langka

Tahap 6 — Tambahkan label sintetis

Gunakan aturan domain sederhana.

Tahap 7 — Validasi cepat

Lihat:

- statistik dasar,
- grafik,
- korelasi,
- rentang nilai.

Tahap 8 — Gunakan di bab berikutnya

Pakai dataset ini untuk:

- *feature engineering*,
- visualisasi,
- *machine learning*,
- dan pengujian pipeline.

4.10 Penutup bab

Bab 4 memberi pembaca kemampuan yang sangat strategis: kemampuan untuk tidak bergantung penuh pada data yang "kebetulan tersedia." Dengan *dummy data*, *simulated data*, *synthetic data*, dan *augmented data*, pembaca dapat membangun laboratorium digital kecil yang dapat dipakai untuk belajar, menguji pipeline, membuat prototipe, dan menyiapkan eksperimen *machine learning* dengan cara yang terukur dan menyenangkan. Ini sangat selaras dengan arah buku Anda yang menekankan praktik Python, eksperimen open source, dan sistem yang *repeatable* dari

sensor → data → model → inferensi → dashboard.

Pada saat yang sama, bab ini juga menanamkan kedewasaan metodologis: **data sintetik sangat berguna, tetapi tidak boleh disalahpahami sebagai pengganti penuh realitas lapangan**. Justru nilai terbesarnya adalah sebagai jembatan: dari nol menjadi siap menguji, dari ide menjadi prototipe, dan dari prototipe menjadi sistem yang nantinya diuji dengan data sensor nyata.

Hands-on 9 - Generate Data Suhu–kelembaban Ruangan 30 Hari

1. Tujuan hands-on

Pada latihan ini, pembaca akan membuat **dataset sintetis suhu dan kelembaban ruangan selama 30 hari**. Dataset ini dirancang agar:

- punya **pola harian** yang masuk akal,
- bisa dipakai untuk **visualisasi**, **resampling**, **forecasting**, dan **klasifikasi kenyamanan**,
- serta cukup realistis untuk dipakai menguji dashboard, pipeline MQTT, dan model ML awal.

Ini penting karena data sensor indoor sering berbentuk **multivariat time series** dan biasanya menunjukkan pola harian, musiman kecil, serta variasi akibat aktivitas manusia dan perangkat pendingin. Pola seperti ini sangat cocok untuk latihan analitik IoT.

2. Contoh implementasi nyata

Contoh use case yang sangat realistis:

- **ruang server kampus** yang dipantau 24 jam,
- **ruang kelas** dengan pola panas saat jam kuliah,
- **laboratorium komputer** yang lebih hangat saat praktikum,
- **ruang kerja UMKM** yang ingin memantau kenyamanan dan beban pendingin.

3. Persiapan Ubuntu 26.04

Aktifkan lingkungan Python:

```
source .venv/bin/activate
pip install pandas numpy matplotlib
mkdir -p data/synthetic scripts outputs/figures
```

4. Source code utama

Buat file `scripts/generate_room_30days.py`:

```
from pathlib import Path
import numpy as np
import pandas as pd

OUTPUT_PATH = Path("data/synthetic/room_30days.csv")
OUTPUT_PATH.parent.mkdir(parents=True, exist_ok=True)

# Reproducible
rng = np.random.default_rng(42)

# 30 hari, interval 10 menit
timestamps = pd.date_range(
    start="2026-01-01 00:00:00",
    end="2026-01-30 23:50:00",
    freq="10min"
)

hours = timestamps.hour + timestamps.minute / 60.0
day_of_week = timestamps.dayofweek  # 0=Senin ... 6=Minggu

# Pola suhu harian
daily_temp_cycle = 3.5 * np.sin((hours - 6) / 24 * 2 * np.pi)

# Sedikit beda antara hari kerja dan akhir pekan
workday_boost = np.where(day_of_week < 5, 0.8, 0.2)

# Drift kecil antar-hari, misalnya efek cuaca/lingkungan
slow_drift = np.linspace(0, 1.0, len(timestamps))

# Noise
temp_noise = rng.normal(0, 0.35, len(timestamps))

temperature = 24.0 + daily_temp_cycle + workday_boost + slow_drift +
temp_noise

# kelembaban cenderung berlawanan arah dengan suhu
humidity_noise = rng.normal(0, 1.8, len(timestamps))
humidity = 68.0 - 0.9 * daily_temp_cycle - 0.3 * workday_boost - 0.2 *
slow_drift + humidity_noise

# Batasi rentang agar realistis
humidity = np.clip(humidity, 35, 85)

df = pd.DataFrame({
    "timestamp": timestamps,
    "device_id": "room01",
    "temperature_c": np.round(temperature, 2),
    "humidity_pct": np.round(humidity, 2)
})

df.to_csv(OUTPUT_PATH, index=False)

print(f"Dataset disimpan ke: {OUTPUT_PATH}")
print(df.head())
```

```
print(df.tail())
print(df.describe())
```

Jalankan:

```
python scripts/generate_room_30days.py
```

5. Visualisasi cepat

Buat file `scripts/plot_room_30days.py`:

```
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt

DATA_PATH = Path("data/synthetic/room_30days.csv")
OUTPUT_DIR = Path("outputs/figures")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

df = pd.read_csv(DATA_PATH)
df["timestamp"] = pd.to_datetime(df["timestamp"])
df = df.set_index("timestamp")

# Plot 3 hari pertama agar mudah dibaca
sample = df.loc["2026-01-01":"2026-01-03 23:59:59"]

plt.figure(figsize=(12, 5))
plt.plot(sample.index, sample["temperature_c"], label="Temperature")
plt.plot(sample.index, sample["humidity_pct"], label="Humidity")
plt.title("Data Sintetik Suhu-kelembaban (3 Hari Pertama)")
plt.xlabel("Waktu")
plt.ylabel("Nilai")
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.savefig(OUTPUT_DIR / "room_30days_sample.png")
plt.show()
```

6. Hal yang harus diamati pembaca

Perhatikan apakah:

- suhu lebih tinggi pada siang hari,
- akhir pekan sedikit berbeda,
- kelembaban cenderung turun saat suhu naik,
- noise kecil membuat data tidak “terlalu sempurna”.

Hands-on 10 - Generate Data Konsumsi Energi Berbasis Jam Kerja

1. Tujuan hands-on

Latihan ini membuat **dataset konsumsi energi** yang mengikuti **jam kerja**, **jam sibuk**, dan **akhir pekan**. Dataset ini sangat cocok untuk:

- analitik energi gedung,
- simulasi dashboard,
- prediksi beban,
- klasifikasi `hemat / boros`,
- dan pengujian alarm beban puncak.

Data energi seperti ini sangat relevan karena **Smart Energy Analytics** sebagai salah satu studi kasus utama.

2. Contoh implementasi nyata

Use case yang bisa langsung diimplementasikan:

- ruang komputer kampus,
- laboratorium jaringan,
- kantor kecil,
- bengkel teknik,
- server mini komunitas.

3. Logika sintetis yang dipakai

Kita buat aturan sederhana:

- malam: beban rendah,
- pagi sebelum kerja: naik,
- jam kerja: tinggi,
- jam makan siang: sangat tinggi,
- sore: menurun,
- akhir pekan: lebih rendah.

4. Source code utama

Buat file `scripts/generate_energy_workhours.py`:

```
from pathlib import Path
import numpy as np
import pandas as pd

OUTPUT_PATH = Path("data/synthetic/energy_30days.csv")
OUTPUT_PATH.parent.mkdir(parents=True, exist_ok=True)

rng = np.random.default_rng(123)

timestamps = pd.date_range(
    start="2026-01-01 00:00:00",
    end="2026-01-30 23:50:00",
    freq="10min"
)

records = []

for ts in timestamps:
    hour = ts.hour
    minute = ts.minute
    dow = ts.dayofweek  # 0-4 kerja, 5-6 akhir pekan

    is_weekend = dow >= 5

    if is_weekend:
        if 0 <= hour < 7:
            base = 90
        elif 7 <= hour < 12:
            base = 130
        elif 12 <= hour < 18:
            base = 160
        else:
            base = 110
    else:
        if 0 <= hour < 6:
            base = 110
        elif 6 <= hour < 8:
            base = 170
        elif 8 <= hour < 12:
            base = 320
        elif 12 <= hour < 14:
            base = 430
        elif 14 <= hour < 18:
            base = 300
        elif 18 <= hour < 21:
            base = 180
        else:
            base = 130

    # fluktuasi kecil dalam satu jam
    sub_cycle = 8 * np.sin((minute / 60.0) * 2 * np.pi)

    # drift kecil selama 30 hari, misalnya pemakaian naik sedikit
    day_idx = (ts.date() - timestamps[0].date()).days
```

```
        drift = day_idx * 0.8

        noise = rng.normal(0, 12)

        power_watt = base + sub_cycle + drift + noise

        records.append({
            "timestamp": ts,
            "device_id": "energy-lab01",
            "power_watt": round(power_watt, 2)
        })

df = pd.DataFrame(records)
df.to_csv(OUTPUT_PATH, index=False)

print(f"Dataset energi disimpan ke: {OUTPUT_PATH}")
print(df.head())
print(df.describe())
```

Jalankan:

```
python scripts/generate_energy_workhours.py
```

5. Tambahkan label hemat / boros

Buat file `scripts/label_energy_efficiency.py`:

```
from pathlib import Path
import pandas as pd

INPUT_PATH = Path("data/synthetic/energy_30days.csv")
OUTPUT_PATH = Path("data/synthetic/energy_30days_labeled.csv")

df = pd.read_csv(INPUT_PATH)

mean_power = df["power_watt"].mean()

df["efficiency_label"] = df["power_watt"].apply(
    lambda x: "hemat" if x <= mean_power else "boros"
)

df.to_csv(OUTPUT_PATH, index=False)

print(f"Label hemat/boros disimpan ke: {OUTPUT_PATH}")
print(df.head())
```

6. Insight yang bisa diambil

Pembaca bisa langsung mencoba:

- jam berapa daya paling tinggi,
- apakah akhir pekan benar lebih rendah,
- bagaimana jika ambang `warning` dipasang di 350 watt,
- bagaimana jika label `fault` dipasang di atas 450 watt.

Hands-on 11 - Generate Data Getaran Mesin dengan Fault Injection

1. Tujuan hands-on

Pada latihan ini, pembaca membuat **data getaran sintetis** untuk dua kondisi:

- **normal**,
- **fault**.

Selain itu, kita tambahkan **fault injection** langsung ke sinyal sehingga pembaca bisa belajar:

- bagaimana pola fault muncul,
- bagaimana membedakan sinyal sehat dan tidak sehat,
- dan bagaimana menyiapkan data awal untuk *predictive maintenance*.

Use case seperti ini sangat sesuai dengan penekanan literatur IoT analytics pada **early warning indicators** untuk mesin atau komponen yang mulai gagal.

2. Contoh implementasi nyata

Use case yang sangat implementatif:

- motor kecil di laboratorium,
- kipas industri,
- pompa air,
- mesin putar sederhana untuk praktikum.

3. Ide fault injection

Kita akan membuat:

- sinyal normal dengan amplitudo stabil,
- sinyal fault dengan amplitudo naik,
- spike sesaat,
- noise lebih besar,
- dan *imbalance* sederhana.

4. Source code utama

Buat file `scripts/generate_vibration_fault.py`:

```
from pathlib import Path
import numpy as np
import pandas as pd

OUTPUT_PATH = Path("data/synthetic/vibration_fault_injection.csv")
OUTPUT_PATH.parent.mkdir(parents=True, exist_ok=True)

rng = np.random.default_rng(7)

n = 3000
t = np.arange(n)

# Sinyal dasar mesin normal
base_freq = 0.03
normal_signal = 0.8 * np.sin(2 * np.pi * base_freq * t) +
rng.normal(0, 0.08, n)

# Buat salinan untuk fault
fault_signal = normal_signal.copy()

# Fault 1: amplitudo meningkat
fault_signal[1000:2000] += 0.9 * np.sin(2 * np.pi * base_freq *
t[1000:2000])

# Fault 2: noise bertambah
fault_signal[1500:2300] += rng.normal(0, 0.18, 800)

# Fault 3: spike sesaat
spike_idx = rng.choice(np.arange(1800, 2500), size=15, replace=False)
fault_signal[spike_idx] += rng.uniform(1.0, 2.0, len(spike_idx))

df = pd.DataFrame({
    "sample_id": np.arange(n),
    "machine_id": "motor01",
    "vibration": np.round(fault_signal, 5)
})

# Label sintetis
df["status_label"] = "normal"
df.loc[1000:2499, "status_label"] = "fault"

df.to_csv(OUTPUT_PATH, index=False)

print(f"Dataset getaran disimpan ke: {OUTPUT_PATH}")
print(df.head())
print(df.iloc[995:1005])
print(df.iloc[1795:1805])
```

Jalankan:

```
python scripts/generate_vibration_fault.py
```

5. Visualisasi fault

Buat file `scripts/plot_vibration_fault.py`:

```
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt

DATA_PATH = Path("data/synthetic/vibration_fault_injection.csv")
OUTPUT_DIR = Path("outputs/figures")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

df = pd.read_csv(DATA_PATH)

plt.figure(figsize=(12, 5))
plt.plot(df["sample_id"], df["vibration"], linewidth=1)
plt.title("Sinyal Getaran dengan Fault Injection")
plt.xlabel("Sample ID")
plt.ylabel("Vibration")
plt.tight_layout()
plt.savefig(OUTPUT_DIR / "vibration_fault_injection.png")
plt.show()
```

6. Apa yang harus diamati

Perhatikan:

- sinyal awal relatif halus,
- mulai pertengahan sinyal amplitudo membesar,
- noise bertambah,
- muncul spike yang tidak biasa.

Ini sangat cocok sebagai dasar bab fitur getaran dan *anomaly detection*.

Hands-on 12 - Membuat Dataset Sintetis untuk Klasifikasi dan Anomaly Detection

1. Tujuan hands-on

Pada latihan ini, pembaca akan menggabungkan beberapa prinsip sebelumnya menjadi dataset yang:

- bisa dipakai untuk **klasifikasi**,
- bisa dipakai untuk **anomaly detection**,
- memiliki label sintetis yang jelas,
- dan tetap cukup realistis.

Literatur IoT analytics menekankan bahwa banyak sistem IoT memerlukan gabungan teknik statistik, *machine learning*, dan simulasi, serta bahwa *anomaly detection* adalah salah satu area yang sangat penting pada data sensor.

2. Skenario yang akan dibuat

Kita buat dataset **smart room monitoring** dengan variabel:

- `temperature_c`
- `humidity_pct`
- `co2_ppm`
- `power_watt`

Lalu kita tambahkan label:

- `normal`
- `warning`
- `fault`

Aturan sintetis:

- `normal`: semua nilai dalam rentang sehat
- `warning`: salah satu mendekati batas
- `fault`: ada lonjakan besar atau kombinasi buruk

3. Source code utama

Buat file `scripts/generate_room_classification_dataset.py`:

```
from pathlib import Path
import numpy as np
import pandas as pd

OUTPUT_PATH = Path("data/synthetic/room_classification_dataset.csv")
OUTPUT_PATH.parent.mkdir(parents=True, exist_ok=True)

rng = np.random.default_rng(2026)

timestamps = pd.date_range(
    start="2026-01-01 00:00:00",
    end="2026-01-15 23:50:00",
    freq="10min"
)

hours = timestamps.hour + timestamps.minute / 60.0
dow = timestamps.dayofweek

daily_temp = 24 + 3.5 * np.sin((hours - 6) / 24 * 2 * np.pi)
workday_effect = np.where(dow < 5, 1.0, 0.3)

temperature = daily_temp + workday_effect + rng.normal(0, 0.4,
len(timestamps))
humidity = 67 - 0.8 * daily_temp + rng.normal(0, 1.8, len(timestamps))
co2 = 500 + (temperature - 24) * 70 + rng.normal(0, 30,
len(timestamps))
power = 120 + (temperature - 24) * 12 + workday_effect * 18 +
rng.normal(0, 8, len(timestamps))

df = pd.DataFrame({
    "timestamp": timestamps,
    "device_id": "room01",
    "temperature_c": np.round(temperature, 2),
    "humidity_pct": np.round(humidity, 2),
    "co2_ppm": np.round(co2, 0),
    "power_watt": np.round(power, 2)
})

# Injeksi warning
warning_idx = rng.choice(df.index, size=80, replace=False)
df.loc[warning_idx, "temperature_c"] += rng.uniform(2.0, 4.0,
len(warning_idx))
df.loc[warning_idx, "co2_ppm"] += rng.uniform(120, 280,
len(warning_idx))
df.loc[warning_idx, "power_watt"] += rng.uniform(20, 50,
len(warning_idx))

# Injeksi fault
remaining_idx = list(set(df.index) - set(warning_idx))
fault_idx = rng.choice(remaining_idx, size=30, replace=False)
df.loc[fault_idx, "temperature_c"] += rng.uniform(6.0, 10.0,
len(fault_idx))
df.loc[fault_idx, "co2_ppm"] += rng.uniform(500, 1000, len(fault_idx))
```

```
df.loc[fault_idx, "power_watt"] += rng.uniform(80, 180,
len(fault_idx))

# Label
df["status_label"] = "normal"
df.loc[warning_idx, "status_label"] = "warning"
df.loc[fault_idx, "status_label"] = "fault"

# Kolom biner untuk anomaly detection
df["is_anomaly"] = df["status_label"].apply(lambda x: 0 if x ==
"normal" else 1)

# Pembulatan akhir
df["co2_ppm"] = df["co2_ppm"].clip(lower=350).round().astype(int)

df.to_csv(OUTPUT_PATH, index=False)

print(f"Dataset klasifikasi/anomali disimpan ke: {OUTPUT_PATH}")
print(df["status_label"].value_counts())
print(df.head())
```

Jalankan:

```
python scripts/generate_room_classification_dataset.py
```

4. Mengapa dataset ini berguna?

Dataset ini bisa langsung dipakai untuk:

- klasifikasi 3 kelas: `normal`, `warning`, `fault`
- *binary anomaly detection*: `is_anomaly`
- visualisasi sebaran kelas
- uji *imbalance*
- latihan model dasar seperti Logistic Regression, Random Forest, dan Isolation Forest di bab berikutnya.

5. Cek keseimbangan label

Buat file `scripts/check_label_distribution.py`:

```
from pathlib import Path
import pandas as pd

DATA_PATH = Path("data/synthetic/room_classification_dataset.csv")
df = pd.read_csv(DATA_PATH)

print("Distribusi status_label:")
print(df["status_label"].value_counts())

print("\nDistribusi is_anomaly:")
print(df["is_anomaly"].value_counts())
```

Ini penting karena di IoT, kejadian abnormal memang sering lebih sedikit daripada kondisi normal. Itu sebabnya label sintetis sangat membantu untuk latihan awal.

Hands-on 13 - Ekspor Dataset ke CSV dan Publish ke MQTT Simulator

1. Tujuan hands-on

Pada latihan ini, pembaca akan:

- membaca dataset sintetis dari CSV,
- mengubah setiap baris menjadi payload JSON,
- lalu mengirimkannya ke broker MQTT sebagai **simulator sensor**.

Ini sangat penting karena di sinilah data sintetis benar-benar dipakai untuk **menggerakkan pipeline IoT**, bukan hanya disimpan sebagai file statis. Pendekatan ini sangat cocok dengan arsitektur buku Anda yang menekankan **MQTT Mosquitto**, **subscriber Python**, **storage**, dan **dashboard**.

Selain itu, *Internet of Things and Data Analytics Handbook* menjelaskan bahwa untuk banyak use case IoT, pola **publish-subscribe** sangat cocok sebagai pelengkap REST karena lebih sesuai dengan aplikasi yang *event-driven*.

2. Persiapan Ubuntu 26.04

Pastikan Mosquitto aktif:

```
sudo apt update
sudo apt install -y mosquitto mosquitto-clients
sudo systemctl enable mosquitto
sudo systemctl start mosquitto
sudo systemctl status mosquitto
```

Pasang pustaka MQTT Python:

```
source .venv/bin/activate
pip install paho-mqtt pandas
```

3. Uji broker MQTT

Terminal 1:

```
mosquitto_sub -h localhost -t "iotlab/room01/telemetry" -v
```

Terminal 2:

```
mosquitto_pub -h localhost -t "iotlab/room01/telemetry" -m '{"test":
"ok"}'
```

4. Source code utama: publish CSV ke MQTT

Buat file `scripts/publish_csv_to_mqtt.py`:

```
from pathlib import Path
import json
import time
import pandas as pd
import paho.mqtt.client as mqtt

BROKER_HOST = "localhost"
BROKER_PORT = 1883
TOPIC = "iotlab/room01/telemetry"
CLIENT_ID = "csv-publisher-room01"

DATA_PATH = Path("data/synthetic/room_classification_dataset.csv")

PUBLISH_INTERVAL_SECONDS = 0.5  # percepat simulasi

def main():
    df = pd.read_csv(DATA_PATH)

    client = mqtt.Client(client_id=CLIENT_ID)
    client.connect(BROKER_HOST, BROKER_PORT, keepalive=60)

    print(f"Mengirim data dari {DATA_PATH} ke topic {TOPIC}")

    for _, row in df.iterrows():
        payload = {
            "timestamp": row["timestamp"],
            "device_id": row["device_id"],
            "temperature_c": float(row["temperature_c"]),
            "humidity_pct": float(row["humidity_pct"]),
            "co2_ppm": int(row["co2_ppm"]),
            "power_watt": float(row["power_watt"]),
            "status_label": row["status_label"],
            "is_anomaly": int(row["is_anomaly"])
        }

        payload_json = json.dumps(payload)
        result = client.publish(TOPIC, payload_json, qos=1)

        if result.rc == 0:
            print(f"Published: {payload_json}")
        else:
            print(f"Gagal publish: {payload_json}")

        time.sleep(PUBLISH_INTERVAL_SECONDS)

    client.disconnect()
    print("Selesai mengirim semua data.")

if __name__ == "__main__":
    main()
```

Jalankan:

```
python scripts/publish_csv_to_mqtt.py
```

5. Subscriber Python untuk memverifikasi

Buat file `scripts/subscribe_verify_mqtt.py`:

```
import json
import paho.mqtt.client as mqtt

BROKER_HOST = "localhost"
BROKER_PORT = 1883
TOPIC = "iotlab/room01/telemetry"

def on_connect(client, userdata, flags, reason_code, properties=None):
    print("Terhubung ke broker.")
    client.subscribe(TOPIC, qos=1)
    print(f"Subscribe ke: {TOPIC}")

def on_message(client, userdata, msg):
    try:
        payload = json.loads(msg.payload.decode())
        print(
            f"[{payload['timestamp']}] "
            f"T={payload['temperature_c']}C, "
            f"H={payload['humidity_pct']}%, "
            f"CO2={payload['co2_ppm']}, "
            f"P={payload['power_watt']}W, "
            f"label={payload['status_label']}, "
            f"anomaly={payload['is_anomaly']}"
        )
    except Exception as exc:
        print(f"Gagal parse payload: {exc}")

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect(BROKER_HOST, BROKER_PORT, keepalive=60)
client.loop_forever()
```

6. Variasi realistis yang bisa dicoba

Agar lebih menantang, pembaca bisa mencoba:

- publish hanya data `warning` dan `fault` ke topik khusus `iotlab/room01/alert`
- publish multi-device seperti `room01`, `room02`, `lab01`
- memperlambat atau mempercepat simulasi
- menambahkan `battery_pct`, `location`, atau `sensor_status`
- mengirim data energi atau getaran ke topik lain

7. Versi publish multi-topic

Contoh pengiriman berdasarkan label:

```
from pathlib import Path
import json
import time
import pandas as pd
import paho.mqtt.client as mqtt

BROKER_HOST = "localhost"
BROKER_PORT = 1883

DATA_PATH = Path("data/synthetic/room_classification_dataset.csv")

client = mqtt.Client(client_id="multi-topic-publisher")
client.connect(BROKER_HOST, BROKER_PORT, keepalive=60)

df = pd.read_csv(DATA_PATH)

for _, row in df.iterrows():
    payload = row.to_dict()
    payload_json = json.dumps(payload)

    telemetry_topic = "iotlab/room01/telemetry"
    client.publish(telemetry_topic, payload_json, qos=1)

    if row["status_label"] in ["warning", "fault"]:
        alert_topic = "iotlab/room01/alert"
        client.publish(alert_topic, payload_json, qos=1)

    print(payload_json)
    time.sleep(0.3)

client.disconnect()
```

8. Mengapa langkah ini sangat penting?

Karena pada titik ini pembaca sudah bisa:

- membuat data sintetis,
- menyimpannya ke CSV,
- mengubahnya menjadi aliran telemetry,
- dan memakainya untuk menguji sistem end-to-end.

Ini sangat dekat dengan praktik nyata di edge/gateway Linux, yang menurut referensi memang cocok dipakai untuk pemrosesan lokal, translasi protokol, dan orkestrasi data sebelum ke sistem analitik yang lebih tinggi.

Ringkasan langkah kerja yang harus dilakukan pembaca

Tahap 1 — Bangun data sintetik

Mulai dari:

- suhu–kelembaban 30 hari,
- energi berbasis jam kerja,
- getaran mesin dengan *fault injection*.

Tahap 2 — Tambahkan label

Buat:

- `normal / warning / fault`
- `hemat / boros`
- `is_anomaly`

Tahap 3 — Validasi cepat

Periksa:

- statistik dasar,
- rentang nilai,
- grafik,
- distribusi label.

Tahap 4 — Simpan ke CSV

Jadikan dataset mudah dipakai ulang.

Tahap 5 — Publish ke MQTT

Gunakan CSV sebagai sumber *simulator sensor*.

Tahap 6 — Verifikasi dengan subscriber

Pastikan payload diterima dengan benar.

Tahap 7 — Siapkan untuk bab berikutnya

Dataset yang sama bisa dipakai untuk:

- *feature engineering*,
- dashboard,
- klasifikasi,
- *anomaly detection*,
- dan *forecasting*.

Penutup

Serangkaian Hands-on 9–13 ini memberi pembaca satu kemampuan yang sangat kuat: **membangun laboratorium IoT virtual sendiri**. Dengan Python di Ubuntu 26.04, pembaca tidak perlu menunggu semua sensor fisik siap untuk mulai belajar serius. Mereka bisa membuat pola harian ruangan, beban energi, dan getaran mesin; menyuntikkan *fault*; memberi label sintetis; lalu mengalirkannya ke MQTT untuk menguji pipeline seolah-olah data itu datang dari perangkat nyata. Ini sangat selaras dengan arah buku yang **open source**, **praktis**, dan berfokus pada eksperimen yang **repeatable** dari **sensor → data → model → inferensi → dashboard**.

Bagian II — Python Data Analytics untuk IoT (Bab 5–7)

Bab 5. Feature Engineering untuk Data Sensor

5.1 Mengapa *feature engineering* penting dalam IoT?

Banyak pemula mengira bahwa model yang hebat akan otomatis menghasilkan prediksi yang baik, padahal pada banyak proyek IoT hasil paling besar justru datang dari **fitur yang baik**. Sensor biasanya hanya mengirim nilai mentah: suhu, kelembaban, daya, getaran, posisi, atau arus. Nilai itu penting, tetapi sering belum cukup. Kita sering lebih tertarik pada pertanyaan seperti: **apakah suhu sedang naik cepat?**, **apakah konsumsi energi lebih tinggi dibanding 1 jam lalu?**, **berapa rata-rata getaran dalam 10 detik terakhir?**, atau **apakah pola beberapa sensor bersama-sama menunjukkan gejala gangguan?**. Pertanyaan-pertanyaan seperti ini tidak dijawab langsung oleh data mentah; ia dijawab oleh fitur hasil rekayasa.

Dalam konteks IoT, *feature engineering* menjadi semakin penting karena data lapangan bersifat **berurutan**, **berisik**, **berubah terhadap waktu**, dan sering berasal dari **lebih dari satu sensor**. Literatur juga menekankan bahwa data IoT yang heterogen perlu melalui **cleansing, ordering, preprocessing**, dan dalam banyak situasi hasil terbaik diperoleh ketika informasi dari banyak sensor atau banyak atribut digabungkan secara cerdas. Inilah alasan mengapa Bab 5 berada tepat setelah pembahasan pembersihan data dan data sintetis: pembaca perlu belajar mengubah data yang “sekadar ada” menjadi data yang “punya makna prediktif”.

5.2 Apa yang dimaksud *feature engineering*?

Secara sederhana, **feature engineering** adalah proses membuat variabel baru dari data yang sudah ada agar pola penting menjadi lebih mudah dibaca oleh manusia maupun algoritma. Pada data sensor, fitur baru bisa berasal dari:

- hubungan waktu,
- hubungan antar sensor,
- ringkasan statistik,
- konteks operasional,
- atau transformasi numerik tertentu.

Contoh sederhana:

- dari suhu mentah menjadi **rata-rata suhu 1 jam terakhir**,
- dari daya mentah menjadi **selisih daya terhadap 10 menit sebelumnya**,
- dari banyak sensor menjadi **fitur gabungan**,
- dari *timestamp* menjadi **jam**, **hari**, atau **status hari kerja/akhir pekan**.

Prinsip ini sangat cocok dengan semangat buku Anda: **praktis, ringan, dan terbuka**, sehingga pembaca tidak langsung tenggelam ke model rumit tetapi membangun intuisi yang kuat dari data.

5.3 Rolling window, lag features, moving average, difference

Bagian ini adalah inti paling praktis dari *feature engineering* untuk data sensor.

5.3.1 Rolling window

Rolling window adalah teknik menghitung ringkasan data pada jendela waktu tertentu yang bergerak dari satu titik ke titik berikutnya. Misalnya:

- rata-rata suhu 3 sampel terakhir,
- standar deviasi daya 6 sampel terakhir,
- maksimum getaran 20 sampel terakhir.

Teknik ini sangat penting karena banyak gejala di IoT tidak terlihat dari satu titik data, tetapi terlihat dari **perilaku beberapa titik terakhir**. Pada mesin, lonjakan sesaat mungkin tidak cukup berarti, tetapi **kenaikan variabilitas dalam beberapa jendela** bisa menjadi sinyal awal masalah. Pada ruangan, suhu saat ini mungkin masih normal, tetapi **rata-rata 1 jam terakhir** bisa menunjukkan ruangan makin panas. Pemikiran seperti ini konsisten dengan pembahasan Perros tentang **moving average** sebagai alat untuk membaca tren pada *time series*.

5.3.2 Lag features

Lag feature adalah nilai masa lalu yang dibawa ke baris saat ini. Misalnya:

- `temperature_lag_1` = suhu 10 menit lalu,
- `temperature_lag_6` = suhu 1 jam lalu,
- `power_lag_144` = daya 1 hari lalu jika interval 10 menit.

Lag sangat penting karena data IoT hampir selalu memiliki **ketergantungan waktu**. Nilai sekarang sering dipengaruhi nilai sebelumnya. Dalam *forecasting*, *anomaly detection*, dan klasifikasi berbasis waktu, lag sering menjadi fitur paling berguna. Ini juga berkaitan dengan literatur *time series* yang menekankan peran **autokorelasi**, **ACF**, dan urutan waktu dalam memahami data.

5.3.3 Moving average

Moving average adalah rata-rata bergerak dari sejumlah observasi terakhir. Secara praktis, *moving average* membantu:

- menghaluskan noise,
- memperjelas tren,
- mengurangi efek lonjakan kecil yang tidak penting,
- dan menghasilkan fitur baseline lokal.

Dalam literatur Perros, *simple moving average* digunakan untuk membaca tren *time series* dan memisahkan komponen tren dari variasi lain. Untuk pembaca buku ini, konsep tersebut sangat berguna secara praktis: *moving average* bisa langsung dipakai sebagai fitur atau sebagai alat eksplorasi untuk melihat apakah pola sensor sedang naik, turun, atau stabil.

5.3.4 Difference

Difference atau differencing adalah selisih antara nilai sekarang dan nilai sebelumnya. Misalnya:

- `temp_diff_1 = suhu sekarang - suhu 10 menit lalu`
- `power_diff_6 = daya sekarang - daya 1 jam lalu`

Fitur difference berguna untuk menangkap **kecepatan perubahan**, bukan hanya nilai absolut. Suhu 29°C mungkin masih normal untuk ruang tertentu, tetapi jika 1 jam lalu hanya 24°C, maka laju kenaikannya bisa menjadi sinyal masalah. Literatur Perros juga menjelaskan bahwa **differencing** adalah teknik penting untuk mengurangi efek tren dan membantu analisis *time series* yang tidak stasioner. Dalam konteks buku ini, differencing bukan hanya alat statistik, tetapi juga fitur yang sangat berguna untuk model.

5.4 Statistik per jendela: mean, std, min, max

Pada banyak sistem IoT, fitur yang sangat efektif justru datang dari ringkasan statistik yang sederhana.

Mean

Mean per jendela membantu menangkap level rata-rata lokal. Misalnya:

- rata-rata daya 30 menit terakhir,
- rata-rata suhu 1 jam terakhir,
- rata-rata getaran 100 sampel terakhir.

Standard deviation

Standar deviasi per jendela sangat penting untuk mendeteksi ketidakstabilan. Pada mesin, kenaikan standar deviasi getaran bisa menandakan ketidaknormalan walaupun rata-ratanya belum berubah banyak. Pada ruangan, variasi CO_2 yang membesar dapat menunjukkan perubahan okupansi atau ventilasi yang tidak stabil.

Minimum dan maksimum

Nilai minimum dan maksimum per jendela membantu menangkap rentang operasi lokal. Pada energi, *max* per jendela dapat mewakili *peak load* kecil. Pada suhu, *min* dan *max* membantu melihat apakah ruangan terlalu berfluktuasi.

Pendekatan statistik sederhana seperti ini juga sejalan dengan gagasan di materi IoT analytics bahwa tahap analitik sering melibatkan **filtering, enrichment, transformation**, dan perhitungan yang memperkaya data mentah sebelum diteruskan ke penyimpanan, visualisasi, atau model.

5.5 Encoding waktu: jam, hari, pola musiman

Salah satu kekuatan besar data IoT adalah ia selalu punya konteks waktu. Karena itu, **waktu sendiri harus diperlakukan sebagai fitur**.

Jam

Jam sangat penting karena banyak sistem mengikuti ritme harian:

- ruangan lebih panas di siang hari,
- energi lebih tinggi saat jam kerja,
- lalu lintas lebih padat pada pagi dan sore,
- getaran mesin lebih aktif saat produksi berlangsung.

Hari

Hari dalam minggu juga sering relevan:

- hari kerja dan akhir pekan berbeda,
- Senin bisa berbeda dengan Jumat,
- pola gedung kampus bisa berbeda saat libur.

Pola musiman

Literatur Perros menjelaskan bahwa **seasonality** dapat dikenali melalui pola berulang pada *time series*, dan menjadi salah satu komponen utama data non-stasioner. Dalam buku ini, pola musiman tidak harus langsung berarti musim tahunan; pola **harian** dan **mingguan** sudah sangat penting untuk IoT. Dengan demikian, fitur seperti:

- `hour`,
- `day_of_week`,
- `is_weekend`,
- `hour_sin`, `hour_cos`,
  dapat membuat model lebih peka terhadap ritme alami sistem.

5.6 Sensor fusion sederhana

Sensor fusion berarti menggabungkan informasi dari beberapa sensor untuk memperoleh gambaran yang lebih baik daripada melihat masing-masing sensor secara terpisah. Literatur *Big Data Analytics for Internet of Things* menjelaskan bahwa **data fusion** bertujuan mengoptimalkan pemanfaatan data dari banyak sensor agar diperoleh inferensi yang tidak mungkin atau tidak kuat bila hanya memakai satu sensor. Literatur tersebut juga membedakan beberapa level data fusion, termasuk **feature-level fusion**, yang sangat relevan untuk Bab 5 ini.

Dalam konteks buku ini, sensor fusion tidak perlu dimulai dari algoritma canggih. Kita bisa memulai dari bentuk yang sangat praktis:

- menggabungkan suhu dan kelembaban untuk klasifikasi kenyamanan,
- menggabungkan suhu, CO_2, dan daya untuk mendeteksi kondisi ruangan,
- menggabungkan arus dan getaran untuk tanda awal gangguan mesin,
- menggabungkan GPS dan kecepatan untuk memahami perilaku kendaraan.

Contoh sederhana:

- suhu tinggi + kelembaban rendah → ruangan terasa kering dan panas,
- suhu tinggi + CO_2 tinggi → ventilasi dan okupansi mungkin bermasalah,
- getaran naik + arus naik → motor mungkin bekerja lebih berat dari normal.

Sensor fusion penting karena data IoT sangat heterogen. Dengan menggabungkan fitur dari beberapa sumber, pembaca dapat membangun model yang lebih informatif dan lebih mendekati cara manusia menilai kondisi lapangan.

5.7 PCA

Principal Component Analysis (PCA) adalah teknik reduksi dimensi yang mengubah sejumlah variabel yang mungkin saling berkorelasi menjadi sejumlah komponen baru yang lebih ringkas. Dalam praktik IoT, PCA berguna ketika:

- sensor banyak,
- beberapa variabel saling berkorelasi,
- kita ingin merangkum sistem ke dalam 2–3 dimensi utama,
- atau ingin mengurangi *noise* dan redundansi sebelum model.

Untuk pembaca buku ini, PCA tidak perlu diposisikan sebagai konsep matematis yang menakutkan. Cara paling mudah memahaminya adalah:

- bila ada banyak sensor yang “bercerita hal yang mirip”,
- PCA membantu menemukan arah variasi utama,
- sehingga kita bisa memadatkan informasi tanpa menyimpan semua kompleksitas asli.

Ini sangat cocok untuk kasus:

- suhu, kelembaban, CO_2, dan daya di ruangan,
- banyak kanal getaran pada mesin,
- beberapa sensor lingkungan dalam satu lokasi.

PCA juga sangat relevan setelah sensor fusion sederhana, karena setelah banyak fitur dibuat, pembaca mungkin ingin merangkum fitur-fitur itu menjadi representasi yang lebih padat dan lebih nyaman divisualisasikan.

5.8 Contoh-contoh real yang bisa diimplementasikan

Smart room monitoring

Data:

- suhu,
- kelembaban,
- CO_2,
- daya.

Fitur yang bisa dibuat:

- `temp_lag_1`, `temp_lag_6`
- `temp_ma_6`
- `co2_diff_1`
- `power_mean_1h`
- `hour`, `is_weekend`
- fitur gabungan sederhana untuk kenyamanan

Use case:

- klasifikasi `nyaman / tidak nyaman`
- deteksi ruangan mulai terlalu padat
- *alert* ventilasi

Smart Room Monitoring sebagai studi kasus praktis yang memakai sensor suhu, kelembaban, dan CO_2.

Smart energy analytics

Data:

- `power_watt`,
- `current_amp`,
- `voltage`,
- waktu.

Fitur:

- `power_lag_1`
- `power_ma_6`
- `power_diff_1`
- `hour`, `day_of_week`
- rata-rata beban per jendela

Use case:

- prediksi beban,
- deteksi konsumsi tidak normal,
- klasifikasi `hemat / boros`.

Ini konsisten dengan studi kasus **Smart Energy Analytics**.

Predictive maintenance sederhana

Data:

- getaran,
- arus motor,
- suhu motor.

Fitur:

- mean/std/max getaran per jendela,
- perubahan arus,
- rolling RMS sederhana,
- fitur gabungan getaran + arus.

Use case:

- klasifikasi `normal / warning / fault`
- *anomaly detection* dasar
- interpretasi kondisi mesin.

Ini selaras dengan studi kasus **Predictive Maintenance Sederhana**.

5.9 Langkah kerja yang harus dilakukan pembaca

Agar bab ini benar-benar operasional, pembaca dapat mengikuti alur berikut:

Tahap 1 — Pastikan data sudah bersih

Sebelum *feature engineering*, data harus:

- punya `timestamp` valid,
- terurut,
- interval cukup konsisten,
- *missing value* sudah ditangani minimal.

Tahap 2 — Tentukan tujuan

Tanyakan:

- apakah ingin klasifikasi?
- ingin *forecasting*?
- ingin deteksi anomali?
- ingin visualisasi pola?

Tujuan akan menentukan fitur mana yang relevan.

Tahap 3 — Buat fitur waktu

Tambahkan:

- jam,
- hari,
- akhir pekan / hari kerja,
- siklus harian sederhana.

Tahap 4 — Buat fitur historis

Tambahkan:

- lag,
- moving average,
- difference,
- rolling statistics.

Tahap 5 — Gabungkan beberapa sensor

Lihat apakah kombinasi beberapa sensor memberi informasi tambahan.

Tahap 6 — Ringkas bila perlu

Jika fitur banyak atau saling berkorelasi, gunakan PCA.

Tahap 7 — Validasi sederhana

Periksa:

- adakah fitur yang semuanya kosong?
- adakah fitur yang sangat redundan?
- adakah fitur yang jelas masuk akal terhadap domain?

Langkah-langkah ini konsisten dengan literatur yang menekankan bahwa pada data IoT heterogen, proses **cleansing, ordering, preprocessing, feature selection, dan fusion** sangat penting agar pelatihan dan pengujian model tidak menghasilkan prediksi palsu.

5.10 Sikap profesional dalam *feature engineering*

Satu hal penting yang perlu ditanamkan: **fitur yang baik bukan fitur yang banyak, tetapi fitur yang bermakna**. Pembaca tidak perlu membuat ratusan kolom hanya karena bisa. Yang lebih penting adalah:

- fitur sesuai konteks fisik,
- mudah dijelaskan,
- punya hubungan yang masuk akal dengan target,
- dan tidak membuat sistem terlalu berat.

Dalam dunia IoT, kesederhanaan sangat berharga karena nanti fitur itu bisa dipakai di laptop, server kecil, Raspberry Pi, atau *edge gateway* dengan sumber daya terbatas. Itulah mengapa Bab 5 harus dipahami sebagai seni memilih informasi paling relevan dari data, bukan sekadar menambah kolom sebanyak-banyaknya. Arah ini juga selaras dengan keseluruhan struktur buku yang memprioritaskan praktik yang ringan dan *repeatable* ketimbang kerumitan yang belum perlu.

5.11 Penutup bab

Bab 5 adalah titik ketika data sensor mulai “berubah bentuk” menjadi sesuatu yang jauh lebih cerdas. Dari suhu mentah menjadi tren lokal, dari daya mentah menjadi perubahan beban, dari *timestamp* menjadi pola harian, dari banyak sensor menjadi informasi gabungan, dan dari banyak fitur menjadi representasi yang lebih ringkas lewat PCA. Inilah tahap yang sangat menentukan kualitas analitik IoT, karena model yang baik hampir selalu lahir dari representasi data yang baik. Dengan fondasi ini, pembaca akan jauh lebih siap masuk ke bagian *hands-on* Bab 5, seperti membuat fitur *time-series* dari suhu-kelembaban, fitur getaran sederhana, dan PCA untuk merangkum beberapa sensor. Semua ini sangat selaras dengan arah buku Anda: **open source**, **praktis**, **mudah direplikasi**, dan kuat sebagai jembatan dari **sensor → fitur → model → keputusan**.

Hands-on 14 - Membuat Fitur *Time-Series* dari Data Suhu/kelembaban

1. Tujuan hands-on

Pada latihan ini, pembaca akan mengubah data suhu dan kelembaban menjadi fitur *time-series* yang siap dipakai untuk:

- analitik lanjutan,
- klasifikasi kondisi ruangan,
- prediksi jangka pendek,
- dan deteksi anomali dasar.

Setelah selesai, pembaca diharapkan mampu membuat:

- **lag features**,
- **moving average**,
- **difference**,
- **rolling statistics**,
- dan **fitur waktu** seperti jam, hari, serta penanda akhir pekan.

Latihan ini penting karena pada banyak kasus IoT, nilai sensor saat ini tidak cukup berdiri sendiri. Yang jauh lebih informatif justru adalah **hubungan nilai sekarang dengan masa lalu**, **rata-rata jendela lokal**, dan **konteks waktunya**. Hal ini konsisten dengan pembahasan *time series forecasting* pada Perros, yang menekankan pentingnya **lag**, **moving average**, **differencing**, dan **seasonality**.

2. Contoh implementasi nyata

Contoh use case yang sangat realistis:

- **ruang server** yang perlu mendeteksi kenaikan suhu bertahap,
- **ruang kelas** yang ingin mengukur kenyamanan,
- **laboratorium komputer** yang ingin mengetahui pola panas selama jam praktikum,
- **greenhouse kecil** yang ingin memonitor pola suhu-kelembaban harian.

3. Dataset yang dipakai

Kita pakai dataset sintetis dari bab sebelumnya, misalnya:

```
data/synthetic/room_30days.csv
```

Kolom:

- `timestamp`
- `device_id`
- `temperature_c`
- `humidity_pct`

4. Persiapan Ubuntu 26.04

Aktifkan *virtual environment*:

```
source .venv/bin/activate
pip install pandas numpy matplotlib
mkdir -p scripts data/processed outputs/figures
```

5. Source code utama: membangun fitur *time-series*

Buat file `scripts/feature_room_timeseries.py`:

```
from pathlib import Path
import numpy as np
import pandas as pd

INPUT_PATH = Path("data/synthetic/room_30days.csv")
OUTPUT_PATH = Path("data/processed/room_30days_features.csv")
OUTPUT_PATH.parent.mkdir(parents=True, exist_ok=True)

# 1. Baca data
df = pd.read_csv(INPUT_PATH)

# 2. Konversi timestamp dan urutkan
df["timestamp"] = pd.to_datetime(df["timestamp"])
df = df.sort_values("timestamp").set_index("timestamp")

# 3. Fitur waktu dasar
df["hour"] = df.index.hour
df["day_of_week"] = df.index.dayofweek
df["is_weekend"] = (df["day_of_week"] >= 5).astype(int)

# 4. Encoding siklik untuk jam
df["hour_sin"] = np.sin(2 * np.pi * df["hour"] / 24)
df["hour_cos"] = np.cos(2 * np.pi * df["hour"] / 24)

# 5. Lag features
df["temp_lag_1"] = df["temperature_c"].shift(1)
df["temp_lag_6"] = df["temperature_c"].shift(6)     # 1 jam jika
interval 10 menit
```

```
df["hum_lag_1"] = df["humidity_pct"].shift(1)
df["hum_lag_6"] = df["humidity_pct"].shift(6)

# 6. Difference
df["temp_diff_1"] = df["temperature_c"].diff(1)
df["temp_diff_6"] = df["temperature_c"].diff(6)
df["hum_diff_1"] = df["humidity_pct"].diff(1)

# 7. Moving average / rolling mean
df["temp_ma_3"] = df["temperature_c"].rolling(window=3).mean()
df["temp_ma_6"] = df["temperature_c"].rolling(window=6).mean()
df["hum_ma_3"] = df["humidity_pct"].rolling(window=3).mean()

# 8. Rolling statistics
df["temp_roll_std_6"] = df["temperature_c"].rolling(window=6).std()
df["temp_roll_min_6"] = df["temperature_c"].rolling(window=6).min()
df["temp_roll_max_6"] = df["temperature_c"].rolling(window=6).max()

df["hum_roll_std_6"] = df["humidity_pct"].rolling(window=6).std()
df["hum_roll_min_6"] = df["humidity_pct"].rolling(window=6).min()
df["hum_roll_max_6"] = df["humidity_pct"].rolling(window=6).max()

# 9. Sensor fusion sederhana
df["comfort_index_simple"] = (
    0.6 * df["temperature_c"] + 0.4 * (100 - df["humidity_pct"])
)

# 10. Hapus baris awal yang masih NaN karena lag/rolling
df_clean = df.dropna().copy()

# 11. Simpan
df_clean.to_csv(OUTPUT_PATH)

print(f"Fitur time-series disimpan ke: {OUTPUT_PATH}")
print(df_clean.head())
print("\nKolom fitur:")
print(df_clean.columns.tolist())
```

Jalankan:

```
python scripts/feature_room_timeseries.py
```

6. Penjelasan fitur yang dibuat

Lag features

`temp_lag_1` dan `temp_lag_6` membantu model memahami suhu masa lalu. Ini berguna karena data sensor sangat sering punya ketergantungan waktu. Literatur *time series* memang menekankan bahwa hubungan antar-lag adalah inti pembacaan pola temporal.

Difference

`temp_diff_1` dan `temp_diff_6` menunjukkan laju perubahan suhu. Jika suhu naik cepat, itu bisa menjadi sinyal awal masalah.

Moving average

`temp_ma_3` dan `temp_ma_6` membantu menghaluskan *noise* dan memberi gambaran tren lokal. Ini sangat sesuai dengan konsep *moving average* pada analitik *time series*.

Rolling statistics

`std`, `min`, `max` per jendela membantu menangkap kestabilan dan rentang perilaku lokal sensor.

Encoding waktu

`hour`, `day_of_week`, `is_weekend`, `hour_sin`, dan `hour_cos` membantu model memahami pola harian dan mingguan. *Seasonality* seperti ini sangat umum pada data IoT.

Sensor fusion sederhana

`comfort_index_simple` bukan indeks standar baku, tetapi contoh sederhana bagaimana dua sensor bisa digabungkan pada level fitur. Ini sejalan dengan gagasan **feature-level fusion** pada IoT.

7. Visualisasi fitur

Buat file `scripts/plot_room_features.py`:

```
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt

INPUT_PATH = Path("data/processed/room_30days_features.csv")
OUTPUT_DIR = Path("outputs/figures")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

df = pd.read_csv(INPUT_PATH)
df["timestamp"] = pd.to_datetime(df["timestamp"])
df = df.set_index("timestamp")

sample = df.loc["2026-01-01":"2026-01-03 23:59:59"]

plt.figure(figsize=(12, 5))
plt.plot(sample.index, sample["temperature_c"], label="Temp Raw")
plt.plot(sample.index, sample["temp_ma_6"], label="Temp MA 6")
plt.title("Suhu Mentah vs Moving Average")
plt.xlabel("Waktu")
plt.ylabel("Suhu (C)")
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.savefig(OUTPUT_DIR / "room_temp_ma6.png")
plt.show()
```

8. Hasil yang harus diamati

Pembaca perlu melihat:

- bagaimana *moving average* menghaluskan data,
- bagaimana *difference* menangkap kenaikan atau penurunan cepat,
- bagaimana fitur waktu memberi konteks pola harian,
- bagaimana fitur gabungan bisa lebih bermakna daripada satu sensor tunggal.

Hands-on 15 - Fitur Getaran Sederhana untuk Mendeteksi Kondisi Mesin

1. Tujuan hands-on

Pada latihan ini, pembaca akan membuat fitur sederhana dari sinyal getaran mesin untuk mendukung:

- klasifikasi `normal / fault`,
- deteksi anomali dasar,
- dan pemahaman awal *predictive maintenance*.

Ini sangat penting karena pada data getaran, **nilai mentah per sampel** sering kurang informatif dibanding **fitur statistik per jendela**. Literatur feature engineering juga menekankan bahwa tahap penting dalam ML adalah **feature development, transformation, extraction, dan selection**, dan bahwa PCA maupun teknik ekstraksi fitur lain dapat membantu mereduksi kompleksitas data mentah.

2. Contoh implementasi nyata

Use case realistis:

- motor kecil di lab teknik,
- pompa air kampus,
- kipas industri,
- blower atau mesin putar ringan pada workshop.

3. Dataset yang dipakai

Gunakan dataset sintetis dari bab sebelumnya:

```
data/synthetic/vibration_fault_injection.csv
```

Kolom:

- `sample_id`
- `machine_id`
- `vibration`
- `status_label`

4. Fitur sederhana yang akan dibuat

Untuk tahap awal, kita fokus pada fitur yang mudah dijelaskan:

- **mean**
- **std**
- **min**
- **max**
- **range**
- **RMS**
- **peak-to-peak**
- **kurtosis sederhana**

Pendekatan ini sangat sesuai dengan pemilihan statistik per jendela dan fitur ringan, bukan langsung model rumit.

5. Source code utama

Buat file `scripts/feature_vibration_machine.py`:

```
from pathlib import Path
import numpy as np
import pandas as pd

INPUT_PATH = Path("data/synthetic/vibration_fault_injection.csv")
OUTPUT_PATH = Path("data/processed/vibration_window_features.csv")
OUTPUT_PATH.parent.mkdir(parents=True, exist_ok=True)

df = pd.read_csv(INPUT_PATH)

window_size = 50
step_size = 25

records = []

def compute_features(signal: np.ndarray) -> dict:
    mean_val = np.mean(signal)
    std_val = np.std(signal)
    min_val = np.min(signal)
    max_val = np.max(signal)
    range_val = max_val - min_val
    rms_val = np.sqrt(np.mean(signal ** 2))
    ptp_val = np.ptp(signal)  # peak to peak

    # Kurtosis sederhana tanpa scipy
    centered = signal - mean_val
    if std_val == 0:
        kurtosis_val = 0.0
    else:
        kurtosis_val = np.mean((centered / std_val) ** 4)

    return {
        "mean": mean_val,
        "std": std_val,
```

```
            "min": min_val,
            "max": max_val,
            "range": range_val,
            "rms": rms_val,
            "peak_to_peak": ptp_val,
            "kurtosis": kurtosis_val,
        }

    for start in range(0, len(df) - window_size + 1, step_size):
        end = start + window_size
        window = df.iloc[start:end]

        signal = window["vibration"].values
        features = compute_features(signal)

        # Label mayoritas dalam jendela
        label = window["status_label"].mode()[0]

        row = {
            "window_start": int(window["sample_id"].iloc[0]),
            "window_end": int(window["sample_id"].iloc[-1]),
            "machine_id": window["machine_id"].iloc[0],
            "status_label": label,
            **features
        }
        records.append(row)

    features_df = pd.DataFrame(records)
    features_df.to_csv(OUTPUT_PATH, index=False)

    print(f"Fitur getaran disimpan ke: {OUTPUT_PATH}")
    print(features_df.head())
    print("\nDistribusi label:")
    print(features_df["status_label"].value_counts())
```

Jalankan:

```
python scripts/feature_vibration_machine.py
```

6. Mengapa fitur-fitur ini berguna?

Mean

Berguna untuk melihat apakah sinyal cenderung bergeser dari baseline.

Standard deviation

Sangat berguna untuk melihat kestabilan. Pada fault, variasi sinyal sering membesar.

RMS

RMS sering dipakai dalam pembacaan sinyal karena menangkap energi rata-rata sinyal.

Peak-to-peak dan range

Membantu melihat seberapa ekstrem ayunan sinyal dalam satu jendela.

Kurtosis

Membantu menangkap bentuk distribusi sinyal. Bila ada banyak spike, kurtosis cenderung meningkat.

7. Visualisasi fitur

Buat file `scripts/plot_vibration_features.py`:

```
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt

INPUT_PATH = Path("data/processed/vibration_window_features.csv")
OUTPUT_DIR = Path("outputs/figures")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

df = pd.read_csv(INPUT_PATH)

plt.figure(figsize=(12, 5))
plt.plot(df["window_start"], df["rms"], label="RMS")
plt.plot(df["window_start"], df["std"], label="STD")
plt.title("Fitur RMS dan STD per Jendela")
plt.xlabel("Window Start")
plt.ylabel("Nilai")
plt.legend()
plt.tight_layout()
plt.savefig(OUTPUT_DIR / "vibration_features_rms_std.png")
plt.show()
```

8. Hal yang harus diamati

Perhatikan apakah saat fault mulai terjadi:

- RMS naik,
- STD naik,
- range membesar,
- peak-to-peak lebih ekstrem,
- kurtosis berubah.

Ini adalah langkah yang sangat realistis menuju *predictive maintenance* ringan.

Hands-on 16 - PCA untuk Merangkum Beberapa Sensor

1. Tujuan hands-on

Pada latihan ini, pembaca akan memakai **PCA (Principal Component Analysis)** untuk merangkum beberapa fitur sensor menjadi **2–3 komponen utama**. Ini sangat berguna ketika:

- jumlah fitur mulai banyak,
- beberapa fitur saling berkorelasi,
- pembaca ingin membuat visualisasi 2 dimensi,
- atau ingin mereduksi dimensi sebelum modelling.

Literatur juga menegaskan bahwa **feature extraction** bertujuan menurunkan dimensi data mentah agar lebih mudah dimodelkan, dan PCA termasuk teknik ekstraksi fitur yang umum.

2. Contoh implementasi nyata

Contoh yang realistis:

- ruangan dengan sensor suhu, kelembaban, CO_2, daya, serta fitur turunannya,
- mesin dengan beberapa sensor getaran dan arus,
- greenhouse dengan suhu, kelembaban, cahaya, dan kelembaban tanah.

3. Dataset yang dipakai

Kita gunakan dataset fitur dari Hands-on 14:

```
data/processed/room_30days_features.csv
```

Karena dataset ini sudah memiliki:

- variabel mentah,
- fitur waktu,
- lag,
- moving average,
- rolling statistics,
- dan sensor fusion sederhana.

4. Persiapan

Pastikan scikit-learn tersedia:

```
source .venv/bin/activate
pip install scikit-learn pandas matplotlib
```

5. Source code utama

Buat file `scripts/pca_room_features.py`:

```
from pathlib import Path
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

INPUT_PATH = Path("data/processed/room_30days_features.csv")
OUTPUT_PATH = Path("data/processed/room_30days_pca.csv")
OUTPUT_PATH.parent.mkdir(parents=True, exist_ok=True)

df = pd.read_csv(INPUT_PATH)

# Pilih fitur numerik yang relevan
feature_cols = [
    "temperature_c",
    "humidity_pct",
    "temp_lag_1",
    "temp_lag_6",
    "hum_lag_1",
    "hum_lag_6",
    "temp_diff_1",
    "temp_diff_6",
    "hum_diff_1",
    "temp_ma_3",
    "temp_ma_6",
    "hum_ma_3",
    "temp_roll_std_6",
    "temp_roll_min_6",
    "temp_roll_max_6",
    "hum_roll_std_6",
    "hum_roll_min_6",
    "hum_roll_max_6",
    "comfort_index_simple",
    "hour_sin",
    "hour_cos",
    "is_weekend"
]

X = df[feature_cols].copy()

# Standarisasi
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# PCA jadi 3 komponen
pca = PCA(n_components=3)
X_pca = pca.fit_transform(X_scaled)

pca_df = pd.DataFrame(X_pca, columns=["pc1", "pc2", "pc3"])
result_df = pd.concat([df[["timestamp", "device_id"]], pca_df],
axis=1)

result_df.to_csv(OUTPUT_PATH, index=False)

print(f"Hasil PCA disimpan ke: {OUTPUT_PATH}")
```

```
print(result_df.head())

print("\nExplained variance ratio:")
for i, ratio in enumerate(pca.explained_variance_ratio_, start=1):
    print(f"PC{i}: {ratio:.4f}")

print(f"\nTotal varian 3 komponen:
{pca.explained_variance_ratio_.sum():.4f}")
```

Jalankan:

```
python scripts/pca_room_features.py
```

6. Mengapa perlu standardisasi?

PCA sensitif terhadap skala. Misalnya:

- `temperature_c` mungkin berada di kisaran 20–30,
- `humidity_pct` di 40–80,
- `comfort_index_simple` bisa lebih besar,
- `hour_sin` dan `hour_cos` di -1 sampai 1.

Karena itu, fitur harus dinormalisasi atau distandarisasi lebih dulu. Literatur feature engineering juga menyinggung bahwa transformasi seperti scaling atau normalization membantu performa model dan membuat data lebih konsisten.

7. Visualisasi PCA 2 dimensi

Buat file `scripts/plot_pca_room.py`:

```
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt

INPUT_PATH = Path("data/processed/room_30days_pca.csv")
OUTPUT_DIR = Path("outputs/figures")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

df = pd.read_csv(INPUT_PATH)

plt.figure(figsize=(8, 6))
plt.scatter(df["pc1"], df["pc2"], s=10)
plt.title("Visualisasi PCA: PC1 vs PC2")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.tight_layout()
plt.savefig(OUTPUT_DIR / "room_pca_scatter.png")
plt.show()
```

8. Interpretasi hasil PCA

Yang perlu dilihat:

- apakah 2–3 komponen pertama sudah menjelaskan porsi besar variasi data,
- apakah data yang awalnya banyak kolom kini bisa dipadatkan,
- apakah pola harian atau cluster kasar mulai tampak di ruang PCA.

PCA tidak langsung memberi makna fisik yang sesederhana suhu atau kelembaban, tetapi ia sangat berguna untuk:

- merangkum informasi,
- mengurangi redundansi,
- membantu visualisasi,
- dan menyiapkan model yang lebih ringan.

9. Contoh perluasan realistis

Smart room

PCA dapat merangkum banyak fitur suhu-kelembaban menjadi 2–3 komponen untuk:

- dashboard analitik,
- deteksi pola harian,
- klasifikasi kondisi ruangan.

Smart energy

PCA dapat dipakai pada fitur beban listrik dan waktunya untuk melihat pola pemakaian.

Multi-sensor fusion

Jika banyak sensor dipasang dalam satu ruang, PCA dapat membantu melihat apakah sebagian besar variasi bisa dijelaskan hanya oleh beberapa komponen utama. Ini sejalan dengan kebutuhan IoT terhadap **feature-level fusion** dan pemilihan fitur yang paling relevan.

Langkah kerja yang harus dilakukan pembaca

Tahap 1 — Pastikan data bersih

Gunakan data yang sudah:

- punya timestamp valid,
- tidak kacau urutannya,
- dan sudah ditangani missing value dasarnya.

Tahap 2 — Buat fitur time-series

Tambahkan:

- lag,
- moving average,
- difference,
- rolling statistics,
- encoding waktu.

Tahap 3 — Buat fitur sinyal getaran

Hitung:

- mean,
- std,
- RMS,
- peak-to-peak,
- kurtosis,
- dan label per jendela.

Tahap 4 — Gabungkan dan pilih fitur

Tentukan fitur mana yang masuk akal untuk analitik atau model.

Tahap 5 — Standarisasi

Sebelum PCA, lakukan scaling.

Tahap 6 — Jalankan PCA

Mulai dari 2–3 komponen.

Tahap 7 — Evaluasi hasil

Lihat *explained variance ratio* dan visualisasi.

Penutup

Hands-on 14–16 ini adalah titik ketika pembaca mulai benar-benar merasakan kekuatan **feature engineering**. Data sensor yang sebelumnya hanya deretan angka sekarang berubah menjadi:

- representasi historis melalui **lag**,
- tren lokal melalui **moving average**,
- sinyal perubahan melalui **difference**,
- kestabilan lokal melalui **rolling statistics**,
- pemahaman lintas sensor melalui **sensor fusion**,
- dan ringkasan padat melalui **PCA**.

Bab 6. Visualisasi, Dashboard, dan Analitik Real-Time

6.1 Mengapa visualisasi sangat penting dalam IoT?

Pada banyak proyek IoT, orang sering terlalu cepat ingin masuk ke *machine learning*. Padahal, pada tahap awal implementasi, manfaat terbesar sering justru datang dari **visualisasi yang benar**. Ketika data sensor divisualisasikan dengan baik, pembaca, operator, teknisi, dan manajer dapat langsung melihat banyak hal yang sebelumnya tersembunyi dalam tabel angka: suhu yang naik bertahap, CO_2 yang melonjak saat ruangan ramai, konsumsi energi yang memuncak pada jam tertentu, *missing value*, *drift*, atau perilaku perangkat yang tidak lazim. Dalam proses analitik, langkah **"explore data and plot the data to see anomalies or patterns"** bahkan disebut secara eksplisit sebagai bagian penting sebelum model dibangun dan dioperasionalkan.

Dalam konteks IoT, visualisasi bukan sekadar hiasan. Visualisasi adalah alat untuk:

- **melihat apa yang sedang terjadi**
- **membandingkan beberapa sensor sekaligus**
- **mengenali pola harian atau mingguan**
- **menemukan masalah kualitas data**
- **menjelaskan kondisi sistem kepada pengguna nonteknis**
- **membantu pengambilan keputusan cepat**

Itulah sebabnya ditempatkan visualisasi sebelum dashboard dan real-time pipeline. Alurnya logis: pembaca harus bisa membaca data terlebih dahulu, baru kemudian membangun sistem operasional yang lebih hidup.

6.2 Posisi Bab 6 dalam alur buku

Dalam struktur buku ini, Bab 6 datang setelah pembaca mempelajari:

- dasar IoT dan arsitektur praktikum,
- pembacaan dan pembersihan data sensor,
- pembuatan data sintetis,
- serta *feature engineering* seperti *lag*, *rolling window*, *moving average*, dan PCA.

Artinya, pada tahap ini pembaca **sudah memiliki data yang layak dibaca** dan **sudah mulai memiliki fitur yang informatif**. Maka langkah berikutnya adalah menjawab pertanyaan yang lebih operasional:

- Apa pola utama data sensor?
- Variabel mana yang saling bergerak bersama?
- Kapan sistem tampak normal dan kapan tidak?
- Apa perbedaan melihat data historis dan data real-time?
- Bagaimana menampilkan informasi ini dalam dashboard yang berguna?

Dengan demikian, Bab 6 adalah bab yang mengubah pembaca dari sekadar "pengolah data" menjadi **pengamat sistem**.

6.3 Visualisasi eksploratif dengan matplotlib / plotly ringan

6.3.1 Mengapa memilih matplotlib dan plotly ringan?

Untuk buku ini, pendekatan visualisasi dibuat **ringan, terbuka, dan mudah direplikasi di Ubuntu 26.04**. Karena itu, pilihan paling tepat adalah:

- **matplotlib** untuk grafik dasar yang stabil, ringan, dan mudah dipahami
- **plotly ringan** untuk eksplorasi interaktif bila dibutuhkan
- keduanya berjalan baik di Python dan tidak memerlukan perangkat lunak proprietary

Pilihan ini konsisten dengan arah keseluruhan buku yang mengutamakan **open source stack**, eksperimen Python, dan implementasi yang dapat dijalankan di laptop, mini PC, Raspberry Pi, atau server Ubuntu kecil.

6.3.2 Fungsi visualisasi eksploratif

Visualisasi eksploratif dipakai untuk menjawab pertanyaan awal sebelum masuk ke dashboard operasional. Fungsi utamanya antara lain:

- memeriksa **tren waktu**
- membandingkan **beberapa sensor**
- menemukan **nilai ekstrim**
- melihat **hubungan antar variabel**
- mengidentifikasi **pola musiman**
- memverifikasi apakah hasil *feature engineering* masuk akal

Literatur juga menekankan bahwa data analytics tools harus mampu **mengkomunikasikan insight melalui data visualization**, terutama untuk menjelaskan hasil yang rumit kepada pengguna nonteknis. Dalam IoT, hal ini sangat penting karena dashboard dan visualisasi sering digunakan bukan hanya oleh data scientist, tetapi juga oleh teknisi lapangan, operator gedung, atau pengelola fasilitas.

6.3.3 Bentuk visualisasi yang paling relevan untuk IoT

Untuk mahasiswa dan praktisi, jenis visualisasi yang paling relevan biasanya bukan yang paling rumit, tetapi yang paling berguna secara operasional.

Grafik yang sangat penting untuk IoT:

- **line chart** untuk melihat *time series*
- **multi-line chart** untuk membandingkan beberapa sensor
- **scatter plot** untuk analitik diagnostik ringan
- **histogram** untuk melihat distribusi
- **box plot** untuk mendeteksi *outlier*

- **overlay dengan moving average** untuk memisahkan tren dan noise
- **heatmap korelasi** untuk membaca hubungan antara fitur

6.3.4 Prinsip visualisasi yang baik

Agar visualisasi benar-benar membantu, pembaca perlu memegang beberapa prinsip:

- **jangan menumpuk terlalu banyak variabel dalam satu grafik**
- **gunakan label sumbu dan satuan yang jelas**
- **pilih rentang waktu yang masuk akal**
- **pisahkan grafik eksploratif dan grafik operasional**
- **buat visualisasi untuk menjawab pertanyaan, bukan sekadar karena datanya ada**
- **jangan mengaburkan kondisi abnormal dengan skala yang buruk**

6.4 Analitik deskriptif dan diagnostik untuk IoT

Salah satu kekuatan besar bab ini adalah mengajarkan pembaca membedakan dua jenis analitik yang sering dipakai dalam sistem IoT: **deskriptif** dan **diagnostik**.

6.4.1 Analitik deskriptif

Analitik deskriptif menjawab pertanyaan: **apa yang terjadi?** Literatur menjelaskan bahwa *descriptive analytics* adalah bentuk paling alami dari analitik karena membantu memberi pemahaman awal tentang **bentuk** dan **pola** data. Hasilnya biasanya disajikan dalam bentuk visualisasi, grafik, statistik ringkas, dan segmentasi awal.

Dalam konteks IoT, analitik deskriptif meliputi:

- suhu rata-rata ruangan per jam
- maksimum CO_2 per hari
- total energi per hari atau per minggu
- rata-rata getaran per jendela waktu
- jumlah alarm dalam 24 jam terakhir
- pola suhu pada hari kerja vs akhir pekan

Analitik deskriptif sangat penting karena:

- memberi gambaran umum sistem
- membantu menemukan baseline
- mempermudah identifikasi perubahan pola
- menjadi langkah awal sebelum diagnostik dan prediksi

6.4.2 Analitik diagnostik

Analitik diagnostik menjawab pertanyaan: **mengapa itu terjadi?** Literatur menjelaskan bahwa *diagnostic analytics* berfokus pada **correlations**, *data discovery*,

dan *data mining* untuk melihat faktor-faktor yang mungkin berkaitan dengan suatu kejadian atau perilaku. Namun, penting juga diingat bahwa **korelasi tidak sama dengan kausalitas**.

Dalam sistem IoT, analitik diagnostik bisa berarti:

- melihat apakah suhu naik bersamaan dengan CO_2
- memeriksa apakah beban listrik meningkat saat ruangan lebih panas
- membandingkan getaran mesin dengan arus motor
- menganalisis apakah alarm lebih sering muncul pada jam atau hari tertentu

6.4.3 Perbedaan praktis bagi pembaca

Agar mudah dipahami:

- **deskriptif** = *apa yang terjadi*
- **diagnostik** = *apa yang kemungkinan berkaitan dengan itu*

Contoh:

- "CO_2 meningkat tajam pada pukul 10.00–12.00" → deskriptif
- "CO_2 meningkat bersamaan dengan suhu dan konsumsi daya, kemungkinan karena ruangan penuh dan ventilasi kurang baik" → diagnostik

Inilah mengapa visualisasi multi-variabel sangat penting. Ia menjadi jembatan dari deskriptif ke diagnostik.

6.5 Konsep streaming vs batch

6.5.1 Mengapa pembaca IoT harus memahami keduanya?

Sistem IoT hampir selalu berhadapan dengan dua dunia sekaligus:

- dunia **historis**, tempat data dikumpulkan dan dianalisis secara periodik
- dunia **real-time**, tempat data datang terus-menerus dan harus segera direspons

Karena itu, pembaca perlu memahami perbedaan antara **batch** dan **streaming**.

6.5.2 Batch processing

Batch processing berarti data dikumpulkan dulu, lalu diproses dalam satu kelompok. Pendekatan ini cocok untuk:

- laporan harian
- pembuatan dataset pelatihan
- rekap mingguan
- analisis historis
- eksplorasi dan *feature engineering*

- evaluasi model

Contoh di IoT:

- menghitung rata-rata suhu per hari
- menghitung total konsumsi energi mingguan
- membuat dataset 30 hari untuk klasifikasi atau *forecasting*
- membersihkan data sebelum dipakai untuk melatih model

Batch sangat berguna ketika:

- respons tidak perlu instan
- data perlu diproses lebih mendalam
- analisis fokus pada pola historis

6.5.3 Streaming processing

Streaming processing berarti data diproses ketika data datang, atau sangat dekat dengan waktu kedatangannya. Ini cocok untuk:

- monitoring operasional
- dashboard langsung
- notifikasi cepat
- alarm otomatis
- deteksi anomali real-time ringan
- *edge analytics*

Contoh:

- setiap pesan MQTT langsung ditampilkan di dashboard
- suhu yang melewati ambang langsung mengirim alert
- subscriber Python langsung menghitung skor anomali saat data tiba
- *edge node* memfilter dan menandai data tertentu sebelum diteruskan ke sistem lain

IoT berbeda karena ada **data streaming**, keterbatasan perangkat, dan kebutuhan respons cepat.

6.5.4 Kapan memilih batch, kapan memilih streaming?

Secara praktis:

Gunakan batch jika:

- ingin analisis historis
- sedang menyiapkan dataset untuk ML
- ingin membuat laporan periodik
- respons beberapa menit atau jam masih dapat diterima

Gunakan streaming jika:

- ingin monitoring langsung
- perlu respons cepat
- ingin notifikasi segera
- ingin dashboard yang terus hidup

6.5.5 Dalam sistem nyata, keduanya dipakai bersama

Sistem IoT yang matang biasanya memakai kombinasi keduanya:

- **streaming** untuk tampilan dan respons cepat
- **batch** untuk evaluasi historis, perbaikan model, dan pelaporan

Referensi juga menunjukkan bahwa platform analitik IoT modern sering mendukung **batch analytics** dan **stream analytics** secara bersamaan, karena keduanya melayani kebutuhan yang berbeda namun saling melengkapi.

6.6 Dashboard open source dengan Grafana

6.6.1 Mengapa dashboard dibutuhkan?

Setelah data dibersihkan, divisualisasikan, dan dipahami secara eksploratif, langkah berikutnya adalah membuat **dashboard operasional**. Dashboard dibutuhkan karena:

- operator tidak selalu membuka notebook Python
- data harus bisa dilihat cepat
- keputusan sering perlu diambil dalam hitungan detik atau menit
- histori dan kondisi saat ini harus bisa dibandingkan dengan mudah

Dalam banyak sistem IoT, dashboard adalah titik temu antara:

- data sensor
- analitik
- monitoring operasional
- tindakan manusia

6.6.2 Mengapa Grafana?

Untuk buku ini, **Grafana** adalah pilihan yang sangat tepat karena:

- bersifat **open source**
- sangat kuat untuk **time series**
- cocok dipadukan dengan **InfluxDB**
- mudah digunakan untuk panel suhu, kelembaban, CO_2, daya, dan alarm
- ringan untuk praktikum dan cukup kuat untuk sistem nyata

Grafana dalam stack open source utama, baik pada arsitektur praktikum maupun proyek akhir.

6.6.3 Fungsi dashboard dalam IoT

Dashboard dalam konteks IoT tidak hanya menampilkan grafik, tetapi membantu pengguna:

- melihat nilai saat ini
- membandingkan histori dan kondisi terbaru
- mengidentifikasi perangkat yang bermasalah
- membaca alarm atau *warning*
- memantau pola operasional
- mendukung diskusi teknis dan manajerial

6.6.4 Contoh panel dashboard yang relevan

Untuk kasus **smart room monitoring**, panel yang berguna antara lain:

- suhu saat ini
- kelembaban saat ini
- CO_2 saat ini
- daya listrik saat ini
- grafik 24 jam terakhir
- indikator ruangan nyaman / tidak nyaman
- jumlah alarm hari ini

Untuk **smart energy analytics**:

- daya real-time
- total energi harian
- beban puncak hari ini
- perbandingan hari kerja vs akhir pekan
- beban di luar jam operasional

Untuk **predictive maintenance sederhana**:

- RMS getaran
- standar deviasi getaran per jendela
- arus motor
- indikator normal / warning / fault
- histori alarm

6.6.5 Prinsip desain dashboard yang baik

Agar dashboard berguna dan tidak sekadar cantik, pegang prinsip berikut:

- **utamakan informasi yang bisa ditindaklanjuti**
- **gunakan warna dan threshold secara konsisten**
- **pisahkan panel operasional dan panel analitik**
- **jangan memaksa terlalu banyak panel dalam satu layar**
- **pastikan label, satuan, dan rentang waktu jelas**

- **fokus pada pertanyaan pengguna, bukan hanya struktur database**

6.6.6 Grafana dalam arsitektur buku ini

Dalam alur buku Anda, Grafana berada setelah:

- data dikirim melalui MQTT
- subscriber Python membaca data
- data disimpan di SQLite atau InfluxDB
- analitik atau model mulai menambahkan nilai

Dengan kata lain, Grafana adalah lapisan presentasi yang membuat seluruh kerja backend terlihat nyata. Ini sangat cocok dengan tujuan buku: **dari sensor ke keputusan cerdas**.

6.7 Contoh-contoh nyata yang bisa diimplementasikan

Smart room monitoring

Sistem:

- sensor suhu, kelembaban, CO_2
- data masuk ke MQTT
- subscriber Python membersihkan data
- InfluxDB menyimpan data
- Grafana menampilkan dashboard real-time

Nilai praktis:

- ruang server bisa dipantau
- ruang kelas bisa dievaluasi kenyamanannya
- ventilasi bisa dianalisis secara visual

Smart Room Monitoring sebagai studi kasus penting dengan **dashboard real-time**.

Smart energy analytics

Sistem:

- sensor daya listrik
- data dikumpulkan dan disimpan
- dashboard menampilkan pola beban
- alarm muncul jika daya terlalu tinggi

Nilai praktis:

- identifikasi pemborosan
- pemantauan beban puncak
- evaluasi pemakaian di luar jam operasional

Predictive maintenance sederhana

Sistem:

- data getaran dan arus motor
- fitur dihitung per jendela
- dashboard menampilkan RMS, STD, dan status mesin
- *warning* ditampilkan sebelum terjadi kerusakan lebih besar

Nilai praktis:

- memudahkan maintenance berbasis kondisi
- membantu operator membaca gejala awal
- menjadi jembatan menuju *anomaly detection* berbasis ML

6.8 Langkah kerja yang harus dilakukan pembaca

Agar bab ini benar-benar operasional, pembaca dapat mengikuti tahapan berikut:

Tahap 1 — Pilih data yang akan divisualisasikan

Gunakan data yang sudah cukup bersih dan memiliki:

- `timestamp`
- satu atau lebih variabel sensor
- metadata seperlunya

Tahap 2 — Mulai dari visualisasi eksploratif sederhana

Buat:

- line chart untuk setiap sensor
- plot gabungan multivariabel
- scatter plot untuk hubungan antar variabel

Tahap 3 — Tarik insight deskriptif

Tanyakan:

- apa yang terlihat?
- kapan pola berubah?
- berapa rentang normalnya?

Tahap 4 — Lakukan analitik diagnostik ringan

Cari:

- korelasi antar sensor
- pola jam tertentu
- perubahan pada hari kerja vs akhir pekan
- hubungan antara alarm dan perilaku sensor lain

Tahap 5 — Bedakan kebutuhan batch dan streaming

Putuskan:

- mana yang cukup diproses harian
- mana yang harus ditampilkan langsung
- mana yang perlu alarm cepat

Tahap 6 — Susun kebutuhan dashboard

Tentukan:

- siapa pengguna dashboard
- indikator apa yang harus selalu terlihat
- nilai apa yang perlu diberi ambang warna
- grafik apa yang perlu histori

Tahap 7 — Integrasikan dengan Grafana

Gunakan:

- InfluxDB atau sumber data lain yang sesuai
- panel sederhana terlebih dahulu
- lalu tambahkan threshold, alert, dan filter waktu

6.9 Sikap profesional dalam visualisasi dan dashboard

Ada satu pelajaran penting yang perlu ditekankan: **dashboard yang baik bukan dashboard yang paling ramai, tetapi yang paling berguna**. Dalam praktik IoT, informasi harus disajikan sedemikian rupa sehingga:

- operator cepat paham
- teknisi cepat bertindak
- analis bisa mengecek histori
- pimpinan bisa melihat indikator utama tanpa tenggelam dalam detail

Visualisasi juga harus dijaga agar tidak menyesatkan. Skala, warna, dan rentang waktu yang buruk bisa membuat kondisi tampak normal padahal sebenarnya bermasalah. Karena itu, desain visual harus diperlakukan sebagai bagian dari rekayasa sistem, bukan sekadar lapisan kosmetik.

6.10 Penutup bab

Bab 6 menegaskan bahwa data IoT yang baik harus bisa dilihat, dibaca, dan dipahami dengan cepat. Visualisasi eksploratif dengan **matplotlib** atau **plotly ringan** membantu pembaca membangun pemahaman awal tentang pola data. Dari situ, analitik **deskriptif** menjawab *apa yang terjadi*, sedangkan analitik **diagnostik** membantu memahami *apa yang kemungkinan berkaitan dengan itu*. Di atas fondasi tersebut, pembaca kemudian dibawa untuk memahami perbedaan antara **batch** dan **streaming**, dua cara berpikir yang sangat penting dalam sistem IoT modern. Akhirnya, semua itu bermuara pada **dashboard open source dengan Grafana**, yang berfungsi sebagai jendela operasional dari seluruh sistem IoT analytics. Arah ini sepenuhnya selaras dengan bab ini sebagai jembatan dari *feature engineering* menuju sistem monitoring dan analitik real-time yang lebih matang.

Bab 7. Membangun Dataset Siap Machine Learning

7.1 Mengapa “dataset siap ML” berbeda dari sekadar data bersih?

Banyak pemula merasa bahwa setelah data dibersihkan dari *missing value*, *outlier*, dan *timestamp* yang kacau, maka data itu otomatis siap untuk *machine learning*. Padahal, pada praktik IoT, **data bersih belum tentu dataset siap ML**. Dataset siap ML harus memenuhi syarat tambahan: targetnya jelas, fitur dan labelnya selaras, pembagian train-test masuk akal, validasinya tidak bocor, dan metrik evaluasinya sesuai dengan jenis masalah yang ingin diselesaikan.

Dalam sistem IoT, kesalahan pada tahap ini bisa sangat mahal. Model bisa terlihat akurat di laboratorium, tetapi gagal total di lapangan hanya karena:

- target dirumuskan dengan buruk,
- data dibagi sembarangan tanpa memperhatikan waktu,
- validasi dilakukan secara tidak realistis,
- atau data sintetik mendominasi pola sehingga model salah belajar.

Karena itu, Bab 7 sangat penting. Inilah tahap ketika data tidak lagi hanya “bisa dibaca”, tetapi sudah disusun agar dapat **mengajarkan sesuatu yang benar kepada model**.

7.2 Posisi Bab 7 dalam alur buku

Sebelum sampai di sini, pembaca dalam alur buku ini sudah belajar:

- memahami jenis data sensor IoT dan masalah kualitas data,
- membangun *dummy data* dan *synthetic data*,
- membuat fitur *time-series*,
- memvisualisasikan data multivariabel,
- dan membuat pipeline kecil dari MQTT ke database dan dashboard.

Maka, Bab 7 adalah langkah alami berikutnya. Setelah data:

- sudah cukup bersih,
- punya fitur bermakna,
- dan bisa divisualisasikan,

pembaca sekarang perlu menjawab:

- target ML-nya apa?
- bagaimana membagi data supaya realistis?
- bagaimana mengukur apakah model benar-benar bagus?
- kapan data sintetik membantu?
- dan kapan ia justru menyesatkan?

7.3 Menentukan target

Langkah pertama dalam membangun dataset siap ML adalah **menentukan target**. Ini terlihat sederhana, tetapi justru sangat menentukan arah seluruh proyek. Secara umum, untuk konteks buku ini, ada tiga target utama yang sangat relevan untuk IoT:

- **klasifikasi status**
- **regresi prediksi nilai sensor**
- **deteksi anomali**

Referensi Perros juga menjelaskan bahwa label atau *target variable* adalah inti dari proses klasifikasi dan regresi; pada pohon keputusan misalnya, label bisa bersifat **nominal** untuk klasifikasi atau **kontinu** untuk regresi.

7.3.1 Klasifikasi status

Klasifikasi status berarti kita ingin model menebak kategori tertentu dari data sensor. Label target biasanya berupa kelas seperti:

- `normal`
- `warning`
- `fault`

atau:

- `nyaman`
- `tidak_nyaman`

atau:

- `hemat`
- `boros`

Klasifikasi sangat cocok untuk IoT karena banyak keputusan operasional berbentuk kategori. Di lapangan, teknisi sering tidak bertanya “berapa kemungkinan suhu akan berubah 0,7 derajat”, tetapi bertanya “apakah ruangan ini aman atau perlu perhatian?”, “apakah mesin ini normal atau bermasalah?”, “apakah konsumsi energi masih wajar atau sudah boros?”

Contoh implementasi nyata

Smart room monitoring

- fitur: suhu, kelembaban, CO_2, daya, *moving average*, *lag*
- target: `nyaman` vs `tidak_nyaman`

Predictive maintenance sederhana

- fitur: RMS getaran, STD getaran, arus, suhu motor
- target: `normal`, `warning`, `fault`

Smart energy analytics

- fitur: daya, jam, hari, rata-rata beban lokal
- target: `hemat` vs `boros`

Kapan target klasifikasi tepat?

Gunakan klasifikasi jika:

- hasil akhirnya memang kategori,
- keputusan operasional berbasis status,
- pengguna sistem lebih mudah memahami label daripada angka prediksi,
- atau data historis sudah punya label/aturan untuk membuat label.

7.3.2 Regresi prediksi nilai sensor

Regresi berarti kita ingin memprediksi **nilai numerik kontinu**. Dalam konteks IoT, target regresi biasanya berupa:

- suhu 10 menit ke depan,
- konsumsi daya 1 jam ke depan,
- kelembaban tanah berikutnya,
- atau nilai sensor lain di masa depan.

Referensi juga menjelaskan bahwa regresi sangat banyak dipakai pada analitik IoT untuk **predictive modeling**, **time series modeling**, dan **controlling operations**, termasuk contoh pada prediksi energi rumah pintar dan aplikasi IoT lain.

Contoh implementasi nyata

Smart room

- target: `temperature_next_1step`

Smart energy

- target: `power_watt_next_6step`
- artinya memprediksi daya 1 jam ke depan jika interval 10 menit

Greenhouse

- target: `soil_moisture_next_3step`

Kapan target regresi tepat?

Gunakan regresi jika:

- tujuan utama adalah memprediksi angka,
- keputusan bergantung pada besaran kuantitatif,
- atau Anda ingin membangun dasar *forecasting*.

7.3.3 Deteksi anomali

Deteksi anomali adalah target yang sangat khas di IoT. Tujuannya bukan selalu memprediksi kategori detail, melainkan mengenali apakah suatu kondisi **normal** atau **menyimpang**. Ini sangat relevan karena di banyak sistem IoT, kejadian abnormal justru jarang terjadi.

Referensi Mohanty menjelaskan bahwa pada *unsupervised learning*, salah satu jenis masalah yang sangat cocok adalah **identifying abnormalities in data**, misalnya untuk mendeteksi intrusi atau perilaku yang tidak biasa. Referensi Saleem juga menyebut penggunaan ML untuk **attack and anomaly detection in IoT sensors**.

Bentuk target anomali

- `is_anomaly = 0/1`
- `normal / anomaly`
- skor anomali buatan dari aturan domain

Contoh implementasi nyata

Ruang server

- suhu dan CO_2 yang melonjak bersamaan → anomali

Energi gedung

- beban listrik tinggi di luar jam operasional → anomali

Mesin

- RMS getaran dan arus motor meningkat tajam → anomali

Kapan target anomali tepat?

Gunakan pendekatan ini jika:

- label kelas rinci sangat sedikit,
- tujuan utamanya mendeteksi penyimpangan,
- atau sistem perlu *alert* awal sebelum klasifikasi lebih rinci dibuat.

7.4 Memilih target yang tepat

Sebelum membangun dataset, pembaca harus menjawab pertanyaan berikut:

- **Apakah saya ingin status?** → klasifikasi
- **Apakah saya ingin angka masa depan?** → regresi
- **Apakah saya ingin mendeteksi penyimpangan?** → deteksi anomali

Jangan mencampur semua tujuan sekaligus pada tahap awal. Lebih baik satu target yang jelas daripada tiga target yang ambigu.

7.5 Split data *time-aware* untuk IoT

Ini salah satu bagian paling penting dalam seluruh bab.

Pada banyak tutorial ML umum, data sering dibagi secara acak menjadi:

- train
- validation
- test

Namun untuk **data IoT yang berbentuk *time series***, pembagian acak sering **salah secara metodologis**. Mengapa? Karena data IoT memiliki urutan waktu. Jika model dilatih menggunakan data dari masa depan lalu diuji pada data dari masa lalu, hasil evaluasi akan terlalu optimistis dan tidak realistis.

Referensi Perros menegaskan bahwa banyak data sensor adalah *time series* yang diukur pada titik waktu berturut-turut, sehingga urutan waktu adalah bagian dari struktur data itu sendiri.

Prinsip dasar split *time-aware*

Data harus dibagi berdasarkan urutan waktu, misalnya:

- **70% awal** untuk training
- **15% berikutnya** untuk validation
- **15% terakhir** untuk testing

Dengan begitu:

- model belajar dari masa lalu,
- divalidasi pada masa yang lebih baru,
- dan diuji pada masa yang lebih paling baru.

Kenapa ini penting?

Karena di dunia nyata, model IoT hampir selalu digunakan seperti ini:

- model dilatih dari histori,
- lalu dipakai untuk memprediksi atau mengklasifikasi data berikutnya.

Contoh pembagian

Misalnya data 30 hari:

- hari 1–21 → training
- hari 22–26 → validation
- hari 27–30 → testing

Ini jauh lebih realistis daripada mengacak semua baris.

7.6 Validasi yang benar untuk *time series*

Selain train-test split, pembaca juga perlu memahami validasi yang benar.

Referensi Mohanty menjelaskan pentingnya membedakan:

- **training set**
- **validation set**
- **test set**
- serta **holdout set** yang tidak pernah dilihat model selama pelatihan.

Dalam konteks *time series* IoT, validasi yang baik harus menjaga urutan waktu. Artinya:

- jangan sampai data masa depan dipakai untuk menyetel model yang diuji pada masa lalu,
- jangan lakukan *leakage* dari fitur masa depan ke fitur masa kini,
- jangan menghitung statistik global seluruh dataset lalu memakainya sebelum split.

Bentuk validasi yang tepat

Untuk buku ini, bentuk yang paling mudah dijelaskan adalah:

Holdout berbasis waktu

- train pada bagian awal,
- validasi pada bagian tengah,
- test pada bagian akhir.

Expanding window sederhana

Model dilatih pada jendela waktu yang makin besar:

- train 1–10, validasi 11–12
- train 1–12, validasi 13–14
- train 1–14, validasi 15–16

Pendekatan ini lebih realistis untuk sistem IoT yang terus bertambah data dari waktu ke waktu.

Hal yang harus dihindari

- *random shuffle split* untuk *forecasting*
- normalisasi memakai seluruh data sebelum split
- membuat fitur target masa depan lalu lupa menggeser waktu dengan benar
- mencampur data train dan test dalam *rolling statistic* yang bocor

7.7 Metrik evaluasi

Setelah dataset siap dan split dilakukan dengan benar, pertanyaan berikutnya adalah: **bagaimana menilai model?**

Jawabannya tergantung target yang dipilih.

7.7.1 MAE / RMSE untuk regresi

Untuk prediksi numerik, dua metrik yang paling penting dalam buku ini adalah:

- **MAE** (*Mean Absolute Error*)
- **RMSE** (*Root Mean Squared Error*)

Referensi Perros membahas pentingnya mengecek akurasi *forecasting* melalui residual, yaitu selisih antara nilai aktual dan nilai hasil prediksi, lalu memeriksa apakah residual cukup kecil dan tidak menunjukkan pola bermasalah.

MAE

MAE mengukur rata-rata besar kesalahan absolut.

Maknanya sederhana:

- jika MAE = 0.8 untuk suhu,
- berarti rata-rata kesalahan model sekitar 0.8°C.

MAE sangat enak dijelaskan ke pengguna non teknis.

RMSE

RMSE juga mengukur kesalahan, tetapi lebih menghukum error besar.

Gunakan RMSE jika:

- lonjakan error besar ingin dianggap lebih serius,
- atau ingin membedakan model yang sama-sama punya MAE mirip tetapi satu model kadang salah sangat jauh.

Interpretasi praktis

Untuk prediksi suhu:

- MAE kecil → model stabil
- RMSE jauh lebih besar dari MAE → ada beberapa prediksi yang sangat meleset

7.7.2 Accuracy / Precision / Recall / F1 untuk klasifikasi

Untuk klasifikasi, referensi yang Anda lampirkan jelas menggunakan metrik:

- **accuracy**
- **precision**
- **recall**
- **F1 score**

Accuracy

Accuracy adalah proporsi prediksi yang benar dari seluruh prediksi.

Namun, di IoT accuracy **sering menipu**, terutama bila data tidak seimbang.

Contoh:

- 95% data normal
- 5% data fault

Model yang selalu menebak “normal” akan mendapat accuracy 95%, tetapi sebenarnya gagal total untuk mendeteksi fault.

Precision

Precision menjawab:
dari semua yang diprediksi positif/anomali/fault, berapa yang benar-benar positif?

Penting bila *false positive* mahal.

Contoh:

- alarm fault palsu terlalu sering akan membuat teknisi jenuh.

Recall

Recall menjawab:
dari semua kejadian positif/anomali/fault yang benar-benar ada, berapa yang berhasil ditangkap?

Penting bila *false negative* berbahaya.

Contoh:

- melewatkan fault mesin bisa lebih berbahaya daripada alarm palsu.

F1 score

F1 menggabungkan precision dan recall menjadi satu angka.

Sangat berguna ketika:

- kelas tidak seimbang,
- dan kita ingin kompromi antara terlalu banyak alarm palsu dan terlalu banyak kasus yang lolos.

7.7.3 Confusion matrix

Confusion matrix adalah alat yang sangat penting untuk klasifikasi, karena ia menunjukkan:

- berapa banyak prediksi benar,
- berapa banyak *false positive*,
- berapa banyak *false negative*,
- dan bagaimana model salah mengklasifikasi tiap kelas.

Referensi Obaidat dan Sharma sama-sama menegaskan peran **confusion matrix** dalam mengevaluasi performa model klasifikasi dan melihat jumlah *false positives* maupun hasil klasifikasi multi-kelas.

Mengapa confusion matrix sangat cocok untuk IoT?

Karena dalam banyak sistem operasional, kita ingin tahu jenis kesalahan model:

- apakah model sering menganggap `warning` sebagai `normal`?
- apakah `fault` sering lolos?
- apakah model terlalu mudah memberi alarm?

Bagi teknisi, informasi seperti ini sering lebih berguna daripada satu angka accuracy tunggal.

7.8 Menggabungkan data nyata dan data sintetik

Ini bagian yang sangat penting dan sangat praktis.

Pada banyak proyek IoT, data nyata:

- sedikit,
- kotor,
- tidak seimbang,
- atau belum mencakup kasus *fault* yang cukup.

Sementara itu, data sintetik:

- mudah dibuat,
- bisa diberi label,
- bisa disuntik anomali,
- dan sangat berguna untuk prototipe.

Karena itu, menggabungkan keduanya sering masuk akal.

Referensi Mohanty menekankan bahwa data IoT yang heterogen perlu melalui **cleansing, ordering, preprocessing**, dan bahwa penggabungan data dari berbagai sumber harus dilakukan hati-hati agar tidak menimbulkan prediksi palsu.

Strategi penggabungan yang masuk akal

- gunakan **data nyata sebagai inti utama**
- gunakan **data sintetik untuk menambah variasi**
- gunakan **data sintetik untuk menambah kelas langka**
- gunakan **data sintetik untuk menguji robustitas pipeline**

Contoh implementasi

Predictive maintenance

- data nyata normal: banyak
- data nyata fault: sangat sedikit

- data sintetik fault: ditambahkan agar model belajar pola gangguan awal

Smart room

- data nyata suhu-kelembaban: ada
- data sintetik CO_2 atau skenario ventilasi buruk: ditambahkan untuk simulasi kondisi tertentu

Energi

- data nyata beban normal: ada
- data sintetik lonjakan daya: ditambahkan untuk uji alarm dan klasifikasi `warning / fault`

7.9 Kapan synthetic data membantu?

Synthetic data membantu ketika:

1. Data nyata terlalu sedikit

Misalnya:

- baru 5 hari pengukuran,
- belum cukup untuk melatih model awal.

2. Label kejadian penting terlalu jarang

Misalnya:

- data fault mesin hampir tidak ada,
- kejadian alarm sangat sedikit.

3. Ingin menguji pipeline

Data sintetik bagus untuk:

- uji MQTT,
- uji subscriber,
- uji database,
- uji dashboard,
- uji alert.

4. Ingin membangun baseline cepat

Sebelum sistem lapangan matang, model awal sudah bisa dibangun.

5. Ingin menguji skenario yang jarang terjadi

Misalnya:

- suhu sangat tinggi,
- CO_2 melonjak,
- beban daya ekstrem,
- getaran abnormal.

Dalam konteks pembelajaran dan prototipe, synthetic data bisa sangat mempercepat kemajuan.

7.10 Kapan synthetic data bisa menyesatkan?

Ini bagian yang harus ditekankan kuat. **Synthetic data sangat berguna, tetapi bisa berbahaya bila diperlakukan terlalu naif.**

1. Jika data sintetik terlalu “rapi”

Dunia nyata penuh:

- noise,
- *missing value*,
- drift,
- perangkat rusak,
- interval tak konsisten.

Jika data sintetik terlalu sempurna, model akan belajar dunia yang terlalu ideal.

2. Jika aturan sintetik terlalu sederhana

Misalnya fault selalu berarti suhu > 35°C. Model bisa jadi hanya hafal ambang buatan itu, bukan memahami pola sebenarnya.

3. Jika proporsi sintetik terlalu dominan

Jika 80% dataset adalah sintetik dan hanya 20% nyata, model bisa lebih “setia” pada pola sintetik daripada kenyataan lapangan.

4. Jika distribusi sintetik tidak cocok dengan lapangan

Misalnya:

- data sintetik suhu berkisar 24–30°C,
- tetapi lapangan nyata sering 22–34°C.

Maka model akan bias.

5. Jika data sintetik dipakai untuk evaluasi akhir

Data sintetik bagus untuk pelatihan awal atau validasi internal, tetapi **uji akhir** tetap sebaiknya pada data nyata yang tidak pernah dilihat model.

7.11 Strategi praktis yang disarankan

Agar penggunaan synthetic data tetap aman, berikut strategi yang disarankan:

Gunakan data nyata sebagai referensi distribusi

Buat synthetic data yang:

- mengikuti rentang wajar,
- pola waktu wajar,
- noise yang masuk akal.

Pisahkan peran data

- data sintetik untuk eksplorasi, pelatihan awal, dan balancing
- data nyata untuk validasi akhir dan pengujian lapangan

Dokumentasikan sumber label

Jelaskan apakah label berasal dari:

- observasi nyata,
- aturan domain,
- atau injeksi sintetik

Uji model pada data nyata terbaru

Ini sangat penting agar model benar-benar relevan terhadap kondisi lapangan.

7.12 Contoh-contoh nyata yang bisa diimplementasikan

Smart room monitoring

Target klasifikasi

- `nyaman` vs `tidak_nyaman`

Fitur

- suhu
- kelembaban
- CO_2
- daya

- lag dan moving average

Split

- minggu 1–3 train
- minggu 4 validation
- minggu 5 test

Metrik

- accuracy
- recall untuk `tidak_nyaman`
- confusion matrix

Smart energy analytics

Target regresi

- `power_watt_next_1step`

Fitur

- daya saat ini
- lag daya
- moving average daya
- jam
- hari kerja / akhir pekan

Split

- berbasis waktu

Metrik

- MAE
- RMSE

Predictive maintenance sederhana

Target

- `normal / warning / fault`
  atau
- `is_anomaly`

Fitur

- RMS getaran
- STD getaran
- arus motor
- suhu motor
- rolling max

Metrik

- precision
- recall
- F1
- confusion matrix

Penggabungan data

- data nyata normal
- data sintetik fault untuk penyeimbang awal

7.13 Langkah kerja yang harus dilakukan pembaca

Supaya bab ini benar-benar operasional, pembaca dapat mengikuti langkah berikut.

Tahap 1 — Tentukan tujuan model

Pilih satu:

- klasifikasi
- regresi
- deteksi anomali

Tahap 2 — Tentukan target secara eksplisit

Buat kolom target yang jelas dan tidak ambigu.

Tahap 3 — Pastikan fitur dan target selaras waktu

Jangan sampai fitur masa depan bocor ke target masa kini.

Tahap 4 — Lakukan split berbasis waktu

Pisahkan train, validation, dan test dengan urutan waktu yang realistis.

Tahap 5 — Pilih metrik yang tepat

- regresi → MAE, RMSE
- klasifikasi → accuracy, precision, recall, F1
- lihat confusion matrix untuk klasifikasi

Tahap 6 — Evaluasi risiko ketidakseimbangan label

Cek apakah kelas langka terlalu sedikit.

Tahap 7 — Pertimbangkan data sintetik secara hati-hati

Gunakan untuk membantu, bukan untuk menggantikan kenyataan lapangan.

Tahap 8 — Simpan versi dataset final

Pisahkan:

- data mentah
- data fitur
- train set
- validation set
- test set

Ini juga sejalan dengan praktik *data lake* atau *data zone* yang memisahkan raw, refined, dan production-ready data agar pengguna tahu tahap pemrosesan tiap dataset.

7.14 Sikap profesional dalam membangun dataset ML untuk IoT

Ada satu pelajaran penting yang perlu ditanamkan: **model yang baik dimulai dari dataset yang jujur**. Jujur berarti:

- targetnya sesuai kenyataan,
- split-nya realistis,
- validasinya tidak bocor,
- metriknya sesuai tujuan,
- dan penggunaan data sintetik tidak menipu diri sendiri.

Dalam proyek IoT, sering kali godaan terbesarnya adalah membuat model terlihat bagus di atas kertas. Tetapi sistem yang benar-benar berguna adalah sistem yang tetap masuk akal ketika diuji pada data yang lebih baru, lebih berisik, dan lebih sulit. Karena itu, Bab 7 bukan hanya soal teknik, tetapi juga soal disiplin metodologis.

7.15 Penutup bab

Bab 7 adalah titik ketika data sensor mulai benar-benar disiapkan untuk mengajar model secara benar. Di sini pembaca belajar bahwa membangun dataset siap *machine learning* bukan hanya urusan “membersihkan data”, tetapi juga **memilih target yang tepat**, **membagi data dengan cara yang menghormati waktu**, **memvalidasi model tanpa kebocoran**, **memilih metrik yang sesuai**, dan **menggunakan data sintetik secara bijaksana**.

Hands-on 20 - Menyiapkan Dataset Klasifikasi Normal vs Abnormal

1. Tujuan hands-on

Pada latihan ini, pembaca akan menyiapkan dataset klasifikasi sederhana untuk masalah **normal vs abnormal** pada konteks IoT. Setelah selesai, pembaca diharapkan mampu:

- memilih target klasifikasi yang jelas,
- menyiapkan fitur dari data sensor,
- membuat label `normal` dan `abnormal`,
- melakukan **split *time-aware***,
- dan menyimpan dataset train/validation/test secara rapi.

Latihan ini sangat penting karena di banyak use case IoT, keputusan awal yang dibutuhkan bukan prediksi yang rumit, melainkan **apakah kondisi masih normal atau perlu perhatian**. Referensi juga menegaskan bahwa *classification* adalah salah satu fondasi ML untuk IoT analytics, dan label target bisa berupa kelas nominal seperti dua kelas atau multi-kelas.

2. Contoh implementasi nyata

Smart room monitoring

Fitur:

- suhu
- kelembaban
- CO_2
- daya

Target:

- `normal` bila kondisi masih wajar
- `abnormal` bila salah satu indikator melewati pola aman

Smart energy

Fitur:

- daya
- jam
- rata-rata beban lokal
- perubahan daya

Target:

- `normal` vs `abnormal` untuk beban tidak wajar

Predictive maintenance sederhana

Fitur:

- RMS getaran
- standar deviasi getaran
- arus
- suhu motor

Target:

- `normal` vs `abnormal`

3. Dataset yang akan dipakai

Kita gunakan dataset sintetis hasil bab sebelumnya, misalnya:

```
data/synthetic/room_classification_dataset.csv
```

Kolom umumnya:

- `timestamp`
- `device_id`
- `temperature_c`
- `humidity_pct`
- `co2_ppm`
- `power_watt`
- `status_label`
- `is_anomaly`

Bila Anda sudah punya dataset fitur, itu lebih baik. Jika belum, kita akan membuat beberapa fitur ringan di dalam skrip.

4. Persiapan Ubuntu 26.04

Aktifkan lingkungan Python:

```
source .venv/bin/activate
pip install pandas numpy scikit-learn matplotlib
mkdir -p scripts data/processed
```

5. Strategi yang dipakai

Kita akan:

- membaca dataset,
- mengonversi `timestamp`,
- membuat fitur sederhana,
- menentukan target biner `normal` vs `abnormal`,
- membagi data berdasarkan urutan waktu,
- menyimpan hasilnya ke file terpisah.

Kenapa target biner?

Karena untuk baseline awal, klasifikasi dua kelas jauh lebih mudah dipahami dan sangat relevan untuk IoT operasional.

Kenapa split berbasis waktu?

Karena data IoT adalah *time series*; model harus belajar dari masa lalu dan diuji pada masa yang lebih baru. Ini lebih realistis dibanding *shuffle* acak.

6. Source code utama

Buat file `scripts/prepare_classification_dataset.py`:

```
from pathlib import Path
import pandas as pd
import numpy as np

INPUT_PATH = Path("data/synthetic/room_classification_dataset.csv")
OUTPUT_DIR = Path("data/processed/classification_room")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

# 1. Baca data
df = pd.read_csv(INPUT_PATH)

# 2. Parse timestamp dan urutkan
df["timestamp"] = pd.to_datetime(df["timestamp"])
df = df.sort_values("timestamp").reset_index(drop=True)

# 3. Buat target biner
# Jika sudah ada is_anomaly, gunakan itu.
if "is_anomaly" in df.columns:
    df["target_abnormal"] = df["is_anomaly"].astype(int)
else:
    # fallback sederhana jika belum ada label
    df["target_abnormal"] = (
        (df["temperature_c"] > 30) |
        (df["co2_ppm"] > 1000) |
        (df["power_watt"] > 220)
    ).astype(int)

# 4. Fitur waktu sederhana
```

```
df["hour"] = df["timestamp"].dt.hour
df["day_of_week"] = df["timestamp"].dt.dayofweek
df["is_weekend"] = (df["day_of_week"] >= 5).astype(int)

# 5. Fitur lag dan rolling sederhana
df["temp_lag_1"] = df["temperature_c"].shift(1)
df["hum_lag_1"] = df["humidity_pct"].shift(1)
df["co2_lag_1"] = df["co2_ppm"].shift(1)
df["power_lag_1"] = df["power_watt"].shift(1)

df["temp_ma_3"] = df["temperature_c"].rolling(3).mean()
df["co2_ma_3"] = df["co2_ppm"].rolling(3).mean()
df["power_ma_3"] = df["power_watt"].rolling(3).mean()

df["temp_diff_1"] = df["temperature_c"].diff(1)
df["co2_diff_1"] = df["co2_ppm"].diff(1)
df["power_diff_1"] = df["power_watt"].diff(1)

# 6. Hapus baris awal yang NaN karena lag/rolling
df = df.dropna().reset_index(drop=True)

# 7. Pilih kolom fitur
feature_cols = [
    "temperature_c", "humidity_pct", "co2_ppm", "power_watt",
    "hour", "day_of_week", "is_weekend",
    "temp_lag_1", "hum_lag_1", "co2_lag_1", "power_lag_1",
    "temp_ma_3", "co2_ma_3", "power_ma_3",
    "temp_diff_1", "co2_diff_1", "power_diff_1"
]

target_col = "target_abnormal"

# 8. Split time-aware: 70% train, 15% val, 15% test
n = len(df)
train_end = int(n * 0.70)
val_end = int(n * 0.85)

train_df = df.iloc[:train_end].copy()
val_df = df.iloc[train_end:val_end].copy()
test_df = df.iloc[val_end:].copy()

# 9. Simpan semua
df.to_csv(OUTPUT_DIR / "full_dataset.csv", index=False)
train_df.to_csv(OUTPUT_DIR / "train.csv", index=False)
val_df.to_csv(OUTPUT_DIR / "validation.csv", index=False)
test_df.to_csv(OUTPUT_DIR / "test.csv", index=False)

# 10. Simpan X/y terpisah bila perlu
train_df[feature_cols].to_csv(OUTPUT_DIR / "X_train.csv", index=False)
train_df[[target_col]].to_csv(OUTPUT_DIR / "y_train.csv", index=False)

val_df[feature_cols].to_csv(OUTPUT_DIR / "X_val.csv", index=False)
val_df[[target_col]].to_csv(OUTPUT_DIR / "y_val.csv", index=False)

test_df[feature_cols].to_csv(OUTPUT_DIR / "X_test.csv", index=False)
test_df[[target_col]].to_csv(OUTPUT_DIR / "y_test.csv", index=False)

print("=== Ringkasan Dataset Klasifikasi ===")
print(f"Total data      : {len(df)}")
```

```
print(f"Train           : {len(train_df)}")
print(f"Validation      : {len(val_df)}")
print(f"Test            : {len(test_df)}")
print("\nDistribusi target keseluruhan:")
print(df[target_col].value_counts())
print("\nDistribusi target train:")
print(train_df[target_col].value_counts())
print("\nFile disimpan di:", OUTPUT_DIR)
```

Jalankan:

```
python scripts/prepare_classification_dataset.py
```

7. Hal yang harus diperiksa pembaca

Setelah skrip dijalankan, periksa:

- apakah jumlah data train/validation/test masuk akal,
- apakah target `abnormal` terlalu sedikit atau cukup,
- apakah fitur yang dibuat tidak mengandung kebocoran masa depan,
- apakah semua file berhasil disimpan.

Cek cepat dengan Python

```
import pandas as pd

train = pd.read_csv("data/processed/classification_room/train.csv")
print(train.head())
print(train["target_abnormal"].value_counts(normalize=True))
```

8. Insight penting

Latihan ini mengajarkan bahwa dataset klasifikasi yang baik bukan hanya soal punya label, tetapi juga:

- targetnya jelas,
- fiturnya relevan,
- urutan waktunya dihormati,
- dan distribusi labelnya dipahami.

Hands-on 21 - Menyiapkan Dataset Regresi Prediksi Konsumsi Energi

1. Tujuan hands-on

Pada latihan ini, pembaca akan membuat dataset regresi untuk memprediksi **konsumsi energi satu langkah ke depan**. Setelah selesai, pembaca diharapkan mampu:

- menentukan target regresi yang benar,
- menggeser target ke masa depan,
- membuat fitur berbasis histori,
- melakukan split *time-aware*,
- dan menghasilkan dataset siap model.

Regresi sangat relevan untuk IoT karena banyak aplikasi membutuhkan **prediksi nilai sensor**, misalnya energi, suhu, atau beban. Referensi juga menegaskan bahwa regresi banyak dipakai untuk **predictive modeling**, **time series modeling**, dan **energy usage prediction** pada aplikasi IoT.

2. Contoh implementasi nyata

Smart energy gedung

Prediksi:

- daya 10 menit berikutnya
- daya 1 jam berikutnya

Laboratorium komputer

Prediksi:

- beban listrik mendatang saat jam praktikum

Rumah pintar / UMKM

Prediksi:

- konsumsi listrik jangka pendek untuk optimasi operasional

3. Dataset yang akan dipakai

Gunakan dataset energi sintetis dari bab sebelumnya, misalnya:

```
data/synthetic/energy_30days.csv
```

Kolom:

- `timestamp`
- `device_id`
- `power_watt`

4. Strategi

Kita akan memprediksi:

- `power_watt_next_1step`

Artinya:

- target pada baris sekarang adalah nilai `power_watt` pada langkah waktu berikutnya.

Jika interval data 10 menit, maka target adalah daya **10 menit ke depan**.

5. Source code utama

Buat file `scripts/prepare_regression_energy_dataset.py`:

```
from pathlib import Path
import pandas as pd
import numpy as np

INPUT_PATH = Path("data/synthetic/energy_30days.csv")
OUTPUT_DIR = Path("data/processed/regression_energy")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

# 1. Baca data
df = pd.read_csv(INPUT_PATH)

# 2. Parse waktu dan urutkan
df["timestamp"] = pd.to_datetime(df["timestamp"])
df = df.sort_values("timestamp").reset_index(drop=True)

# 3. Target satu langkah ke depan
```

```
df["target_power_next_1step"] = df["power_watt"].shift(-1)

# 4. Fitur waktu
df["hour"] = df["timestamp"].dt.hour
df["day_of_week"] = df["timestamp"].dt.dayofweek
df["is_weekend"] = (df["day_of_week"] >= 5).astype(int)

# Encoding siklik jam
df["hour_sin"] = np.sin(2 * np.pi * df["hour"] / 24)
df["hour_cos"] = np.cos(2 * np.pi * df["hour"] / 24)

# 5. Fitur lag
df["power_lag_1"] = df["power_watt"].shift(1)
df["power_lag_2"] = df["power_watt"].shift(2)
df["power_lag_6"] = df["power_watt"].shift(6)   # 1 jam jika 10 menit

# 6. Fitur rolling
df["power_ma_3"] = df["power_watt"].rolling(3).mean()
df["power_ma_6"] = df["power_watt"].rolling(6).mean()
df["power_std_6"] = df["power_watt"].rolling(6).std()

# 7. Difference
df["power_diff_1"] = df["power_watt"].diff(1)
df["power_diff_6"] = df["power_watt"].diff(6)

# 8. Hapus NaN
df = df.dropna().reset_index(drop=True)

# 9. Pilih fitur
feature_cols = [
    "power_watt",
    "hour", "day_of_week", "is_weekend",
    "hour_sin", "hour_cos",
    "power_lag_1", "power_lag_2", "power_lag_6",
    "power_ma_3", "power_ma_6", "power_std_6",
    "power_diff_1", "power_diff_6"
]

target_col = "target_power_next_1step"

# 10. Split time-aware
n = len(df)
train_end = int(n * 0.70)
val_end = int(n * 0.85)

train_df = df.iloc[:train_end].copy()
val_df = df.iloc[train_end:val_end].copy()
test_df = df.iloc[val_end:].copy()

# 11. Simpan
df.to_csv(OUTPUT_DIR / "full_dataset.csv", index=False)
train_df.to_csv(OUTPUT_DIR / "train.csv", index=False)
val_df.to_csv(OUTPUT_DIR / "validation.csv", index=False)
test_df.to_csv(OUTPUT_DIR / "test.csv", index=False)

train_df[feature_cols].to_csv(OUTPUT_DIR / "X_train.csv", index=False)
train_df[[target_col]].to_csv(OUTPUT_DIR / "y_train.csv", index=False)

val_df[feature_cols].to_csv(OUTPUT_DIR / "X_val.csv", index=False)
```

```
val_df[[target_col]].to_csv(OUTPUT_DIR / "y_val.csv", index=False)

test_df[feature_cols].to_csv(OUTPUT_DIR / "X_test.csv", index=False)
test_df[[target_col]].to_csv(OUTPUT_DIR / "y_test.csv", index=False)

print("=== Ringkasan Dataset Regresi ===")
print(f"Total data : {len(df)}")
print(f"Train      : {len(train_df)}")
print(f"Validation : {len(val_df)}")
print(f"Test       : {len(test_df)}")
print("\nPreview:")
print(df[["timestamp", "power_watt",
"target_power_next_1step"]].head(10))
print("\nFile disimpan di:", OUTPUT_DIR)
```

Jalankan:

```
python scripts/prepare_regression_energy_dataset.py
```

6. Hal penting yang harus dipahami

Mengapa target digeser ke atas dengan shift(-1)?

Karena kita ingin nilai masa depan menjadi label target untuk fitur saat ini.

Mengapa tidak boleh sembarang *shuffle*?

Karena ini *time series*. Model harus diuji pada data yang benar-benar lebih baru.

Mengapa ada lag dan rolling?

Karena prediksi energi jangka pendek sangat dipengaruhi histori beberapa langkah sebelumnya. Ini juga sejalan dengan literatur *time series* dan prediksi energi IoT.

Hands-on 22 - Evaluasi Baseline Model pada Data Nyata vs Sintetik

1. Tujuan hands-on

Pada latihan ini, pembaca akan membuat **baseline model** sederhana lalu membandingkan hasil evaluasi pada:

- **data sintetik**
- **data nyata**
- atau kombinasi keduanya jika tersedia

Tujuan utamanya bukan mencari model paling hebat, tetapi memahami:

- bagaimana evaluasi dilakukan,
- bagaimana metrik dibaca,
- dan bagaimana performa bisa berubah tergantung kualitas data.

Ini sangat penting karena referensi menekankan bahwa model harus diuji pada data yang **belum pernah dilihat** dan bahwa evaluasi model adalah bagian utama dari strategi AI/ML.

2. Strategi baseline yang dipakai

Kita gunakan dua baseline sederhana:

Baseline klasifikasi

- **Logistic Regression**
  untuk kasus `normal` vs `abnormal`

Baseline regresi

- **Linear Regression**
  untuk prediksi `power_watt_next_1step`

Kenapa model sederhana?

Pembaca sebaiknya memahami dulu model ringan sebelum melompat ke model yang lebih kompleks.

3. Evaluasi baseline klasifikasi

Buat file `scripts/evaluate_baseline_classification.py`:

```
from pathlib import Path
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score,
    f1_score, confusion_matrix, classification_report
)
DATA_DIR = Path("data/processed/classification_room")

X_train = pd.read_csv(DATA_DIR / "X_train.csv")
y_train = pd.read_csv(DATA_DIR / "y_train.csv").squeeze()

X_val = pd.read_csv(DATA_DIR / "X_val.csv")
y_val = pd.read_csv(DATA_DIR / "y_val.csv").squeeze()

X_test = pd.read_csv(DATA_DIR / "X_test.csv")
y_test = pd.read_csv(DATA_DIR / "y_test.csv").squeeze()

# Model baseline
model = LogisticRegression(max_iter=1000)

# Training
model.fit(X_train, y_train)

# Validasi
y_val_pred = model.predict(X_val)

print("=== HASIL VALIDASI KLASIFIKASI ===")
print("Accuracy :", accuracy_score(y_val, y_val_pred))
print("Precision:", precision_score(y_val, y_val_pred,
zero_division=0))
print("Recall   :", recall_score(y_val, y_val_pred, zero_division=0))
print("F1 Score :", f1_score(y_val, y_val_pred, zero_division=0))
print("\nConfusion Matrix:")
print(confusion_matrix(y_val, y_val_pred))
print("\nClassification Report:")
print(classification_report(y_val, y_val_pred, zero_division=0))

# Test
y_test_pred = model.predict(X_test)

print("\n=== HASIL TEST KLASIFIKASI ===")
print("Accuracy :", accuracy_score(y_test, y_test_pred))
print("Precision:", precision_score(y_test, y_test_pred,
zero_division=0))
print("Recall   :", recall_score(y_test, y_test_pred,
zero_division=0))
print("F1 Score :", f1_score(y_test, y_test_pred, zero_division=0))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_test_pred))
```

Jalankan:

```
python scripts/evaluate_baseline_classification.py
```

Apa yang harus dibaca?

- **accuracy**: bagus untuk gambaran umum, tetapi jangan dipakai sendirian
- **precision**: penting kalau alarm palsu mahal
- **recall**: penting kalau kegagalan menangkap anomali berbahaya
- **F1**: bagus untuk kompromi
- **confusion matrix**: sangat penting untuk melihat tipe kesalahan model

4. Evaluasi baseline regresi

Buat file `scripts/evaluate_baseline_regression.py`:

```
from pathlib import Path
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error

DATA_DIR = Path("data/processed/regression_energy")

X_train = pd.read_csv(DATA_DIR / "X_train.csv")
y_train = pd.read_csv(DATA_DIR / "y_train.csv").squeeze()

X_val = pd.read_csv(DATA_DIR / "X_val.csv")
y_val = pd.read_csv(DATA_DIR / "y_val.csv").squeeze()

X_test = pd.read_csv(DATA_DIR / "X_test.csv")
y_test = pd.read_csv(DATA_DIR / "y_test.csv").squeeze()

# Model baseline
model = LinearRegression()

# Training
model.fit(X_train, y_train)

# Validasi
y_val_pred = model.predict(X_val)
mae_val = mean_absolute_error(y_val, y_val_pred)
rmse_val = np.sqrt(mean_squared_error(y_val, y_val_pred))

print("=== HASIL VALIDASI REGRESI ===")
print("MAE  :", mae_val)
print("RMSE :", rmse_val)

# Test
y_test_pred = model.predict(X_test)
mae_test = mean_absolute_error(y_test, y_test_pred)
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

print("\n=== HASIL TEST REGRESI ===")
print("MAE  :", mae_test)
print("RMSE :", rmse_test)

# Simpan hasil prediksi jika ingin diperiksa
result_df = pd.DataFrame({
    "actual": y_test,
```

```
    "predicted": y_test_pred
})
result_df.to_csv(DATA_DIR / "test_predictions.csv", index=False)
print("\nPrediksi test disimpan ke:", DATA_DIR /
"test_predictions.csv")
```

Jalankan:

```
python scripts/evaluate_baseline_regression.py
```

Bagaimana membaca MAE dan RMSE?

- **MAE** = rata-rata kesalahan absolut
- **RMSE** = menghukum kesalahan besar lebih keras

Untuk prediksi energi:

- MAE kecil berarti prediksi cukup dekat secara rata-rata
- RMSE yang jauh lebih besar dari MAE berarti ada beberapa prediksi yang sangat meleset

5. Membandingkan data nyata vs sintetik

Ini bagian yang sangat menarik dan sangat penting.

Tujuan

Melihat apakah model:

- terlalu bagus pada data sintetik,
- tetapi melemah pada data nyata,
- atau cukup stabil di keduanya.

Skenario praktis

Misalnya:

- dataset sintetik untuk ruang/energi dibuat sangat rapi,
- data nyata dari MQTT/InfluxDB/CSV lapangan lebih berisik.

Pendekatan

Ada tiga pola evaluasi yang bisa dipakai:

Pola A — Train sintetik, test sintetik

Ini cocok untuk baseline awal dan debugging.

Pola B — Train sintetik + nyata, test nyata

Ini lebih realistis untuk melihat apakah data sintetik membantu.

Pola C — Train nyata, test nyata

Ini adalah evaluasi paling dekat dengan kondisi lapangan.

6. Source code contoh perbandingan sederhana

Buat file `scripts/compare_synthetic_vs_real.py`:

```
from pathlib import Path
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Contoh:
# synthetic_train.csv dan real_test.csv harus punya kolom fitur/target
yang sama
SYN_PATH = Path("data/processed/regression_energy/train.csv")
REAL_PATH = Path("data/processed/regression_energy/test.csv")

df_syn = pd.read_csv(SYN_PATH)
df_real = pd.read_csv(REAL_PATH)

feature_cols = [
    "power_watt",
    "hour", "day_of_week", "is_weekend",
    "hour_sin", "hour_cos",
    "power_lag_1", "power_lag_2", "power_lag_6",
    "power_ma_3", "power_ma_6", "power_std_6",
    "power_diff_1", "power_diff_6"
]

target_col = "target_power_next_1step"

X_syn = df_syn[feature_cols]
y_syn = df_syn[target_col]

X_real = df_real[feature_cols]
y_real = df_real[target_col]

model = LinearRegression()
model.fit(X_syn, y_syn)

y_real_pred = model.predict(X_real)
```

```
mae_real = mean_absolute_error(y_real, y_real_pred)
rmse_real = np.sqrt(mean_squared_error(y_real, y_real_pred))

print("=== Train pada data sintetik, test pada data nyata/test holdout
===")
print("MAE  :", mae_real)
print("RMSE :", rmse_real)
```

Interpretasi

Jika performa:

- bagus di sintetik tetapi buruk di nyata → synthetic data terlalu ideal
- cukup stabil di nyata → synthetic data membantu
- buruk di semua kondisi → fitur atau target perlu diperbaiki

7. Kapan synthetic data membantu, dan kapan menyesatkan?

Synthetic data membantu jika:

- data nyata masih sedikit,
- label abnormal jarang,
- ingin menguji pipeline dan baseline,
- ingin menambah variasi skenario.

Synthetic data menyesatkan jika:

- distribusinya terlalu rapi,
- fault terlalu sederhana,
- proporsinya terlalu dominan,
- dipakai sebagai satu-satunya dasar evaluasi akhir.

Referensi juga mengingatkan bahwa pada data IoT heterogen, proses *preprocessing*, *cleansing*, dan *fusion* harus hati-hati karena kombinasi data yang salah dapat menghasilkan prediksi palsu.

8. Langkah kerja yang harus dilakukan pembaca

Tahap 1 — Siapkan dataset klasifikasi

Bangun target `normal` vs `abnormal`, buat fitur, lalu split berbasis waktu.

Tahap 2 — Siapkan dataset regresi

Bangun target `power_watt_next_1step`, buat lag dan rolling, lalu split berbasis waktu.

Tahap 3 — Latih baseline sederhana

Gunakan:

- Logistic Regression untuk klasifikasi
- Linear Regression untuk regresi

Tahap 4 — Evaluasi metrik dengan benar

- klasifikasi → accuracy, precision, recall, F1, confusion matrix
- regresi → MAE, RMSE

Tahap 5 — Bandingkan sintetik dan nyata

Lihat apakah performa konsisten atau menurun tajam.

Tahap 6 — Simpan hasil

Simpan:

- train/validation/test
- file prediksi
- ringkasan metrik
- confusion matrix atau residual sederhana

9. Penutup

Hands-on 20–22 ini adalah tahap ketika pembaca mulai benar-benar **mengubah data sensor menjadi dataset yang siap melatih model dengan jujur**. Pada Hands-on 20, pembaca belajar menyiapkan klasifikasi `normal vs abnormal`. Pada Hands-on 21, pembaca belajar membangun dataset regresi untuk prediksi konsumsi energi. Pada Hands-on 22, pembaca melihat bagaimana baseline model dievaluasi dengan metrik yang tepat, lalu mulai memahami perbedaan perilaku model pada **data sintetik** dan **data nyata**. Inilah fondasi yang sangat kuat sebelum melangkah ke bab model supervised berikutnya.

Bagian III — Machine Learning Hands-on untuk IoT (Bab 8–11)

Bab 8. Supervised Learning untuk IoT dengan scikit-learn

8.1 Mengapa supervised learning penting untuk IoT?

Setelah pembaca belajar:

- memahami data sensor,
- membersihkan *time series*,
- membangun data sintetik,
- membuat fitur,
- memvisualisasikan data,
- dan menyiapkan dataset siap *machine learning*,

langkah berikutnya sangat alami: **melatih model** yang benar-benar dapat belajar dari data. Pada tahap inilah *supervised learning* menjadi sangat penting. Dalam *supervised learning*, model tidak hanya melihat data sensor mentah, tetapi juga belajar dari **contoh-contoh yang sudah memiliki jawaban**. Misalnya:

- suhu dan kelembaban tertentu diberi label `nyaman` atau `tidak_nyaman`,
- pola getaran tertentu diberi label `normal` atau `fault`,
- histori daya dipasangkan dengan target daya 10 menit ke depan.

Referensi Mohanty menegaskan bahwa **supervised learning** adalah pembelajaran dari data yang sudah “labeled”, yaitu data yang hasil atau kelasnya sudah diketahui sebelumnya. Ini sangat cocok untuk banyak skenario IoT, karena kita sering memang ingin model belajar dari histori status perangkat, histori energi, histori alarm, atau histori perilaku sensor yang telah diberi interpretasi domain.

Dalam bahasa yang sederhana, **supervised learning membantu sensor “bercerita lebih jauh” daripada sekadar menampilkan angka**. Ia dapat menjawab:

- apakah ruangan ini normal atau perlu perhatian?
- berapa kemungkinan konsumsi energi 10 menit ke depan?
- apakah perangkat ini sedang masuk fase warning?
- apakah kombinasi beberapa sensor mengarah ke fault?

8.2 Mengapa scikit-learn?

Buku ini sengaja memilih **scikit-learn** sebagai fondasi untuk supervised learning karena:

- **open source**
- sangat matang dan stabil
- ringan untuk laptop Ubuntu, mini PC, atau server kecil
- sangat cocok untuk pembelajaran bertahap
- dokumentasinya baik
- integrasinya kuat dengan **pandas** dan **NumPy**
- cukup untuk banyak use case IoT nyata sebelum masuk ke *deep learning*

Referensi juga mencantumkan **scikit-learn** sebagai salah satu platform open source penting dalam ekosistem machine learning.

Pilihan ini sangat tepat untuk target pembaca mahasiswa, praktisi, engineer, dan pengembang yang ingin cepat produktif dengan pendekatan yang **mudah direplikasi**, **hemat sumber daya**, dan **jelas secara metodologis**. Dibanding langsung memakai model yang terlalu berat, scikit-learn memberi jalur belajar yang bersih:

- mudah memahami masukan dan keluaran model,
- mudah mengaitkan model dengan metrik,
- mudah membandingkan model sederhana dan model ensemble,
- dan sangat cocok untuk data tabular hasil *feature engineering* pada bab-bab sebelumnya.

8.3 Apa yang dimaksud supervised learning?

Secara sederhana, **supervised learning** adalah teknik *machine learning* di mana model belajar dari pasangan:

- **fitur** (*input*)
- **target** (*output*)

Jika targetnya berupa angka kontinu, kita masuk ke **regresi**.
Jika targetnya berupa kelas atau label, kita masuk ke **klasifikasi**.

Referensi juga menegaskan bahwa sebagian besar masalah supervised learning dapat dipandang sebagai:

- **classification**
- **regression**

dan bahwa pemilihan model sangat bergantung pada sifat target yang ingin diprediksi.

Contoh dalam IoT

Regresi

- memprediksi `power_watt_next_1step`
- memprediksi suhu 10 menit ke depan
- memprediksi kelembaban tanah berikutnya

Klasifikasi

- `normal` vs `abnormal`
- `nyaman` vs `tidak_nyaman`
- `normal` / `warning` / `fault`

Dengan demikian, Bab 8 bukan tentang teori abstrak, tetapi tentang **memilih model yang cocok untuk pertanyaan operasional yang nyata**.

8.4 Regresi: Linear Regression

8.4.1 Mengapa mulai dari Linear Regression?

Linear Regression adalah model regresi yang paling sederhana dan paling penting untuk dipahami. Referensi menegaskan bahwa pengembangan model berbasis regresi relatif **straightforward**, cepat, dan sederhana ketika data tidak terlalu rumit, serta sangat luas dipakai pada analitik IoT untuk:

- *predictive modeling*
- *time-series modeling*
- pengendalian operasi
- prediksi penggunaan energi
- imputasi data sensor
- analitik keamanan dan anomali tertentu

Karena itulah, Linear Regression adalah titik awal yang sangat baik untuk pembaca. Model ini membantu memahami:

- hubungan fitur dan target,
- arah pengaruh fitur,
- konsep residual,
- dan baseline yang mudah dijelaskan.

8.4.2 Intuisi sederhana

Linear Regression mencoba membangun hubungan linear antara fitur dan target. Misalnya:

- daya sekarang,
- daya 10 menit lalu,
- rata-rata daya 1 jam terakhir,
- jam,
- hari kerja atau akhir pekan,

dipakai untuk memprediksi **daya langkah berikutnya**.

Jika model cukup baik, pembaca bisa menjelaskan:

- fitur mana yang paling berpengaruh,
- apakah histori lokal cukup untuk prediksi,
- dan seberapa besar rata-rata kesalahan model.

8.4.3 Kelebihan Linear Regression

- sangat mudah dipahami
- cepat dilatih
- ringan dijalankan
- hasilnya mudah dijelaskan
- sangat bagus sebagai baseline awal

8.4.4 Keterbatasan Linear Regression

Referensi Saleem juga menegaskan bahwa kelemahan utamanya adalah ia hanya cocok untuk hubungan yang relatif **linear**, dan asumsi tertentu tentang residual bisa menjadi batasannya.

Dalam praktik IoT, Linear Regression bisa lemah jika:

- hubungan antara fitur dan target sangat non-linear,
- ada interaksi kompleks antara sensor,
- atau data berubah sangat tajam dan tidak cukup ditangkap oleh pola linear.

8.4.5 Contoh implementasi nyata

- prediksi konsumsi energi gedung kecil
- prediksi suhu ruang sederhana
- prediksi beban listrik laboratorium
- baseline awal sebelum mencoba model ensemble

8.4.6 Kapan Linear Regression cocok?

Gunakan Linear Regression jika:

- Anda butuh baseline yang cepat
- fitur masih sedikit dan cukup jelas
- interpretabilitas penting
- Anda ingin membandingkan model sederhana vs model kompleks

8.5 Regresi: Random Forest Regressor

8.5.1 Mengapa Random Forest Regressor penting?

Jika Linear Regression adalah baseline paling mudah, maka **Random Forest Regressor** adalah langkah berikutnya yang sangat kuat. Referensi menjelaskan bahwa **random forest** adalah metode ensemble yang membangun banyak pohon keputusan, lalu:

- untuk **regresi**, hasil akhirnya diambil dari **rata-rata prediksi**
- untuk **klasifikasi**, hasil akhirnya diambil dari **kelas mayoritas**

Ini sangat berguna untuk IoT karena data sensor:

- sering non-linear,
- sering punya interaksi antar fitur,
- kadang tidak nyaman ditangani model linear sederhana.

8.5.2 Intuisi sederhana

Alih-alih membuat satu rumus garis lurus, Random Forest membangun banyak pohon keputusan dari sampel data dan subset fitur yang berbeda. Setiap pohon memberi prediksi, lalu semua prediksi digabungkan.

Pendekatan ini membuat model:

- lebih fleksibel,
- lebih kuat menangkap hubungan non-linear,
- dan sering memberi hasil bagus tanpa perlu banyak rekayasa matematis.

8.5.3 Kelebihan Random Forest Regressor

- kuat untuk data non-linear
- bekerja baik pada fitur tabular
- tidak terlalu sensitif terhadap skala fitur
- relatif tahan terhadap gangguan kecil pada data
- sering bagus untuk baseline yang lebih kuat daripada regresi linear

Referensi Perros juga menekankan bahwa random forests bekerja baik pada data berdimensi tinggi dan juga cukup toleran terhadap *missing values* dibanding banyak model lain.

8.5.4 Keterbatasan

- interpretasi lebih sulit daripada Linear Regression
- ukuran model lebih besar
- prediksi bisa lebih lambat
- tidak sejelas model linear jika tujuan utama adalah menjelaskan hubungan kausal sederhana

8.5.5 Contoh implementasi nyata

- prediksi konsumsi energi dengan pola jam kerja yang non-linear
- prediksi suhu ruang dengan pengaruh beberapa variabel
- prediksi kelembaban tanah dari kombinasi waktu, suhu, dan histori sebelumnya

8.5.6 Kapan Random Forest Regressor cocok?

Gunakan ketika:

- hubungan tidak terlalu linear
- ada cukup fitur hasil *feature engineering*
- Anda ingin performa kuat tanpa langsung masuk ke *deep learning*

8.6 Klasifikasi: Logistic Regression

8.6.1 Mengapa Logistic Regression penting?

Logistic Regression adalah model klasifikasi yang sangat penting untuk dipahami. Referensi Mohanty menjelaskan bahwa Logistic Regression adalah pendekatan statistik yang kuat untuk memodelkan **hasil biner** atau kategorikal dengan satu atau lebih variabel deskriptif, memakai fungsi logistik untuk menghubungkan fitur dan probabilitas kelas.

Dalam konteks IoT, model ini sangat cocok untuk masalah seperti:

- `normal` vs `abnormal`
- `attack` vs `non_attack`
- `nyaman` vs `tidak_nyaman`

Referensi Obaidat juga menunjukkan bahwa Logistic Regression tetap relevan dan efektif dalam tugas klasifikasi IoT seperti deteksi intrusi, meskipun kadang dikalahkan oleh model ensemble pada data yang lebih kompleks.

8.6.2 Intuisi sederhana

Logistic Regression tidak memprediksi angka kontinu seperti Linear Regression, tetapi memprediksi **probabilitas sebuah kelas**. Misalnya:

- probabilitas ruangan berada pada status abnormal,
- probabilitas mesin sedang fault,
- probabilitas trafik sensor termasuk serangan.

Lalu probabilitas itu diubah menjadi keputusan kelas.

8.6.3 Kelebihan Logistic Regression

- mudah dipahami
- cepat dilatih
- sangat cocok untuk baseline klasifikasi
- hasil probabilitas cukup berguna untuk thresholding
- lebih interpretatif daripada banyak model non-linear

8.6.4 Keterbatasan

- kurang fleksibel untuk hubungan non-linear yang kompleks
- sensitif jika pola kelas sangat rumit
- kadang kalah akurasi dibanding ensemble seperti Random Forest atau boosting

8.6.5 Contoh implementasi nyata

- klasifikasi `normal` vs `abnormal` pada smart room
- klasifikasi `serangan` vs `bukan serangan` pada data keamanan IoT
- klasifikasi `fault` awal vs `normal`

8.6.6 Kapan Logistic Regression cocok?

Gunakan ketika:

- ingin baseline klasifikasi yang cepat
- data relatif tabular dan tidak terlalu kompleks
- interpretasi probabilitas penting
- ingin model yang ringan untuk eksperimen awal

8.7 Klasifikasi: Random Forest Classifier

8.7.1 Mengapa Random Forest Classifier sangat relevan untuk IoT?

Untuk klasifikasi IoT, **Random Forest Classifier** adalah salah satu pilihan yang sangat kuat. Referensi menjelaskan bahwa Random Forest sangat robust, akurat, dan memiliki kemampuan baik dalam menangani masalah klasifikasi, termasuk data yang tidak beraturan dan bahkan kondisi ketidakseimbangan kelas tertentu.

Pada konteks IoT, ini sangat penting karena data lapangan sering:

- tidak terlalu rapi,
- punya interaksi fitur kompleks,
- mengandung kelas minoritas seperti `fault`,
- dan memerlukan model yang cukup kuat tanpa banyak penyesuaian rumit.

8.7.2 Intuisi sederhana

Setiap pohon dalam random forest membuat keputusan berdasarkan cabang-cabang fitur. Hutan pohon ini lalu memilih kelas final berdasarkan **majority vote**. Karena keputusan berasal dari banyak pohon, model cenderung:

- lebih stabil,
- tidak terlalu bergantung pada satu pola tunggal,
- dan lebih kuat terhadap variasi data.

8.7.3 Kelebihan Random Forest Classifier

- kuat untuk hubungan non-linear
- bekerja baik untuk data tabular IoT
- dapat menangani banyak fitur
- sering memberi performa tinggi pada klasifikasi
- cocok untuk baseline yang kuat pada `normal / warning / fault`

Referensi Obaidat bahkan menunjukkan bahwa Random Forest dapat mencapai performa sangat tinggi pada tugas klasifikasi tertentu dan efektif untuk mengatasi data yang tidak seimbang dalam beberapa konteks.

8.7.4 Keterbatasan

- interpretasi tidak sesederhana Logistic Regression
- ukuran model bisa lebih besar
- tidak selalu ideal untuk deployment sangat kecil jika sumber daya terbatas
- bisa *overfit* jika parameter tidak dijaga, walau umumnya lebih stabil daripada satu decision tree

8.7.5 Contoh implementasi nyata

- klasifikasi `normal / warning / fault` pada smart room
- klasifikasi status mesin dari fitur getaran
- klasifikasi trafik sensor aman vs mencurigakan
- klasifikasi kondisi energi normal vs boros

8.7.6 Kapan Random Forest Classifier cocok?

Gunakan ketika:

- pola klasifikasi cukup kompleks
- fitur banyak dan beragam
- baseline model linear terasa kurang kuat
- Anda ingin model yang relatif tangguh tanpa setup terlalu rumit

8.8 Gradient Boosting / XGBoost opsional singkat

8.8.1 Mengapa dibahas singkat?

Gradient Boosting / XGBoost dibahas **opsional singkat**, bukan sebagai fokus utama. Ini keputusan yang bagus. Alasannya:

- pembaca perlu menguasai fondasi dulu,
- tidak semua buku IoT perlu langsung masuk ke banyak model ensemble tingkat lanjut,
- dan untuk target halaman yang efisien, lebih baik mendalami model inti yang paling sering dipakai.

Di buku ini, Gradient Boosting / XGBoost hanya sebagai pengayaan, bukan utama.

8.8.2 Intuisi sederhana boosting

Referensi Perros menjelaskan bahwa **boosting** membangun model secara bertahap, di mana model-model kecil atau lemah dibangun berulang kali untuk memperbaiki kesalahan dari iterasi sebelumnya. Ini berbeda dari bagging/random forest yang membangun banyak model paralel dari sampel berbeda.

8.8.3 Kapan Gradient Boosting menarik?

- saat ingin performa lebih tinggi pada data tabular
- saat hubungan antar fitur cukup kompleks
- saat pembaca sudah menguasai baseline lebih sederhana

8.8.4 Kapan tidak perlu memaksakan?

- saat pembaca masih belajar dasar supervised learning
- saat interpretasi sederhana lebih penting
- saat sistem harus tetap ringan
- saat baseline seperti Random Forest sudah cukup

8.8.5 Catatan tentang XGBoost

XGBoost sangat populer untuk data tabular, tetapi untuk konteks buku ini:

- cukup disebut sebagai model ensemble boosting yang kuat,
- tidak perlu menjadi pusat pembahasan,
- dan boleh diposisikan sebagai langkah lanjutan setelah pembaca nyaman dengan scikit-learn inti.

8.9 Kapan memilih model sederhana dibanding model kompleks?

Ini salah satu bagian terpenting dalam bab ini.

Banyak pembelajar baru tergoda mengejar model yang lebih “canggih”. Padahal dalam praktik IoT, **model sederhana seringkali lebih berharga** jika:

- lebih mudah dijelaskan,
- lebih ringan dijalankan,
- lebih cepat dilatih,
- lebih mudah di-debug,
- dan cukup akurat untuk kebutuhan operasional.

Pilih model sederhana jika:

- tujuan utama adalah baseline awal
- data masih sedikit
- interpretasi penting
- sistem akan dijalankan pada perangkat terbatas
- pengguna lapangan butuh penjelasan mudah

Contoh:

- Linear Regression untuk prediksi energi awal
- Logistic Regression untuk abnormal vs normal
- model linear sebagai pembanding resmi sebelum model ensemble

Pilih model lebih kompleks jika:

- baseline sederhana terlalu lemah
- pola data jelas non-linear
- ada cukup fitur dan data
- performa tambahan memang bernilai bagi operasi

Contoh:

- Random Forest saat fitur interaktif dan non-linear
- boosting saat baseline ensemble masih ingin ditingkatkan

Prinsip penting

Jangan memilih model karena terdengar keren. Pilih model karena cocok dengan masalah, data, dan kebutuhan implementasi.

8.10 Contoh-contoh nyata yang bisa diimplementasikan

Smart room monitoring

Regresi

- target: suhu atau CO_2 langkah berikutnya
- model: Linear Regression, Random Forest Regressor

Klasifikasi

- target: `normal / warning / fault`
- model: Logistic Regression, Random Forest Classifier

Smart energy analytics

Regresi

- target: daya 10 menit atau 1 jam ke depan
- model: Linear Regression sebagai baseline, Random Forest Regressor untuk pola lebih kompleks

Klasifikasi

- target: `hemat / boros`
- model: Logistic Regression, Random Forest Classifier

Referensi juga menyebut penggunaan regresi untuk prediksi energi dan kontrol aplikasi IoT berbasis rumah pintar.

Predictive maintenance sederhana

Klasifikasi

- target: `normal`, `warning`, `fault`
- fitur: RMS, STD, peak-to-peak, arus, suhu

Model:

- Logistic Regression untuk baseline
- Random Forest Classifier untuk menangkap pola lebih kompleks

Keamanan atau integritas komunikasi IoT

Referensi Obaidat mencontohkan penggunaan Random Forest dan Logistic Regression dalam klasifikasi data sensor menjadi malicious dan non-malicious pada lingkungan smart home.

8.11 Langkah kerja yang harus dilakukan pembaca

Agar bab ini benar-benar operasional, pembaca dapat mengikuti urutan berpikir berikut.

Tahap 1 — Pastikan dataset sudah siap

Gunakan dataset yang:

- sudah punya target jelas,
- sudah melalui split *time-aware*,
- dan punya fitur yang relevan.

Tahap 2 — Pilih jenis masalah

Tentukan:

- regresi atau klasifikasi

Tahap 3 — Mulai dari model paling sederhana

Untuk baseline:

- regresi → Linear Regression
- klasifikasi → Logistic Regression

Tahap 4 — Tingkatkan bila perlu

Jika baseline kurang kuat:

- coba Random Forest Regressor
- coba Random Forest Classifier

Tahap 5 — Evaluasi dengan metrik yang sesuai

- regresi → MAE, RMSE
- klasifikasi → accuracy, precision, recall, F1, confusion matrix

Tahap 6 — Bandingkan hasil dan interpretasi

Tanyakan:

- apakah model kompleks benar-benar jauh lebih baik?
- apakah peningkatan performa sepadan dengan biaya interpretasi dan kompleksitas?

Tahap 7 — Simpan model terbaik sebagai baseline operasional

Model terbaik belum tentu paling canggih, tetapi yang paling seimbang antara:

- performa
- interpretabilitas
- kemudahan implementasi

8.12 Sikap profesional dalam supervised learning untuk IoT

Ada satu pelajaran penting yang harus ditanamkan kuat: **supervised learning bukan lomba memilih algoritma, melainkan proses membangun keputusan yang dapat dipercaya dari data sensor**.

Model yang baik untuk IoT harus:

- cukup akurat
- stabil pada data baru
- tidak mudah tertipu data sintetik yang terlalu ideal
- bisa dievaluasi dengan jujur
- dan, bila mungkin, bisa dijelaskan ke pengguna lapangan

Dalam banyak proyek nyata, model yang paling berguna justru bukan model paling rumit, tetapi model yang:

- mudah dipelihara,
- ringan dijalankan,
- cocok dengan perangkat dan data yang tersedia,
- dan benar-benar membantu keputusan operasional.

8.13 Penutup bab

Bab 8 adalah titik ketika pembaca mulai benar-benar memakai machine learning terawasi untuk masalah IoT yang nyata. Di sini pembaca belajar bahwa:

- **Linear Regression** memberi baseline yang jernih dan mudah dijelaskan untuk prediksi numerik,
- **Random Forest Regressor** memberi kekuatan tambahan untuk pola yang lebih non-linear,
- **Logistic Regression** memberi dasar klasifikasi yang kuat dan interpretable,
- **Random Forest Classifier** memberi performa yang sering lebih tangguh untuk klasifikasi IoT,
- dan **Gradient Boosting / XGBoost** cukup dikenalkan sebagai langkah lanjutan, bukan kewajiban awal.

Hands-on 23 - Prediksi Suhu / Energi dengan Regresi

1. Tujuan hands-on

Pada latihan ini, pembaca akan membangun model regresi sederhana untuk memprediksi nilai sensor berikutnya, khususnya:

- **prediksi suhu**
- atau **prediksi konsumsi energi**

Setelah selesai, pembaca diharapkan mampu:

- membaca dataset regresi yang sudah disiapkan,
- melatih **Linear Regression** sebagai baseline,
- melatih **Random Forest Regressor** sebagai model non-linear,
- mengevaluasi dengan **MAE** dan **RMSE**,
- serta memvisualisasikan prediksi vs nilai aktual.

Ini sangat penting karena banyak aplikasi IoT memang membutuhkan **predictive modeling** dan **time-series modeling**, misalnya prediksi energi rumah pintar, prediksi beban perangkat, atau prediksi nilai sensor berikutnya. Referensi Saleem secara eksplisit menempatkan regresi sebagai alat penting untuk **predictive modeling**, **time series modeling**, dan **energy usage prediction** pada IoT.

2. Contoh implementasi nyata

Prediksi suhu ruang

Use case:

- ruang server kampus
- laboratorium komputer
- ruang kelas
- greenhouse kecil

Pertanyaan:

- berapa suhu 10 menit berikutnya?
- apakah suhu akan naik terus?
- apakah perlu pendinginan atau ventilasi tambahan?

Use case:

- laboratorium
- ruang komputer
- kantor kecil
- smart home / UMKM

Pertanyaan:

- berapa konsumsi daya 10 menit berikutnya?
- apakah beban akan naik pada jam sibuk?
- apakah ada indikasi pemborosan mendatang?

3. Dataset yang digunakan

Untuk latihan ini, kita gunakan dataset regresi yang disiapkan sebelumnya, misalnya:

```
data/processed/regression_energy/
├── X_train.csv
├── y_train.csv
├── X_val.csv
├── y_val.csv
├── X_test.csv
└── y_test.csv
```

Struktur ini sejalan dengan bab sebelumnya tentang **split *time-aware***, di mana data dibagi menjadi training, validation, dan testing secara berurutan, bukan diacak. Pendekatan ini penting karena data IoT umumnya berbentuk *time series*.

4. Persiapan Ubuntu 26.04

Aktifkan lingkungan Python:

```
source .venv/bin/activate
pip install pandas numpy matplotlib scikit-learn
mkdir -p scripts outputs/figures outputs/reports
```

5. Baseline pertama: Linear Regression

Buat file `scripts/regression_linear_energy.py`:

```
from pathlib import Path
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error

DATA_DIR = Path("data/processed/regression_energy")
```

```
X_train = pd.read_csv(DATA_DIR / "X_train.csv")
y_train = pd.read_csv(DATA_DIR / "y_train.csv").squeeze()

X_val = pd.read_csv(DATA_DIR / "X_val.csv")
y_val = pd.read_csv(DATA_DIR / "y_val.csv").squeeze()

X_test = pd.read_csv(DATA_DIR / "X_test.csv")
y_test = pd.read_csv(DATA_DIR / "y_test.csv").squeeze()

model = LinearRegression()
model.fit(X_train, y_train)

# Validasi
y_val_pred = model.predict(X_val)
mae_val = mean_absolute_error(y_val, y_val_pred)
rmse_val = np.sqrt(mean_squared_error(y_val, y_val_pred))

# Test
y_test_pred = model.predict(X_test)
mae_test = mean_absolute_error(y_test, y_test_pred)
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

print("=== LINEAR REGRESSION ===")
print("Validation MAE  :", mae_val)
print("Validation RMSE :", rmse_val)
print("Test MAE        :", mae_test)
print("Test RMSE       :", rmse_test)

result_df = pd.DataFrame({
    "actual": y_test,
    "predicted_linear": y_test_pred
})
result_df.to_csv(DATA_DIR / "pred_linear_test.csv", index=False)
print("Prediksi test disimpan ke:", DATA_DIR / "pred_linear_test.csv")
```

Jalankan:

```
python scripts/regression_linear_energy.py
```

Apa yang dipelajari?

- **Linear Regression** memberi baseline yang cepat,
- MAE membantu memahami rata-rata besar kesalahan,
- RMSE membantu melihat apakah ada error besar yang sesekali sangat meleset.

Referensi juga menekankan bahwa regresi linear mudah dibangun, cepat, dan cocok untuk data yang relatif sederhana, tetapi terbatas pada hubungan linear.

6. Model kedua: Random Forest Regressor

Buat file `scripts/regression_rf_energy.py`:

```
from pathlib import Path
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error

DATA_DIR = Path("data/processed/regression_energy")

X_train = pd.read_csv(DATA_DIR / "X_train.csv")
y_train = pd.read_csv(DATA_DIR / "y_train.csv").squeeze()

X_val = pd.read_csv(DATA_DIR / "X_val.csv")
y_val = pd.read_csv(DATA_DIR / "y_val.csv").squeeze()

X_test = pd.read_csv(DATA_DIR / "X_test.csv")
y_test = pd.read_csv(DATA_DIR / "y_test.csv").squeeze()

model = RandomForestRegressor(
    n_estimators=200,
    max_depth=10,
    random_state=42,
    n_jobs=-1
)

model.fit(X_train, y_train)

# Validasi
y_val_pred = model.predict(X_val)
mae_val = mean_absolute_error(y_val, y_val_pred)
rmse_val = np.sqrt(mean_squared_error(y_val, y_val_pred))

# Test
y_test_pred = model.predict(X_test)
mae_test = mean_absolute_error(y_test, y_test_pred)
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

print("=== RANDOM FOREST REGRESSOR ===")
print("Validation MAE  :", mae_val)
print("Validation RMSE :", rmse_val)
print("Test MAE        :", mae_test)
print("Test RMSE       :", rmse_test)

result_df = pd.DataFrame({
    "actual": y_test,
    "predicted_rf": y_test_pred
})
result_df.to_csv(DATA_DIR / "pred_rf_test.csv", index=False)
print("Prediksi test disimpan ke:", DATA_DIR / "pred_rf_test.csv")
```

Jalankan:

```
python scripts/regression_rf_energy.py
```

Mengapa Random Forest Regressor?

Karena data IoT sering tidak sepenuhnya linear. Random Forest bisa menangkap hubungan non-linear dan interaksi antar fitur dengan lebih baik. Referensi Saleem menjelaskan bahwa random forest bekerja dengan membangun banyak pohon, lalu untuk regresi hasil akhirnya adalah **rata-rata prediksi** dari seluruh pohon.

7. Visualisasi prediksi regresi

Buat file `scripts/plot_regression_predictions.py`:

```
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt

DATA_DIR = Path("data/processed/regression_energy")
OUTPUT_DIR = Path("outputs/figures")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

df_linear = pd.read_csv(DATA_DIR / "pred_linear_test.csv")
df_rf = pd.read_csv(DATA_DIR / "pred_rf_test.csv")

# Gabungkan
plot_df = pd.DataFrame({
    "actual": df_linear["actual"],
    "pred_linear": df_linear["predicted_linear"],
    "pred_rf": df_rf["predicted_rf"]
})

sample = plot_df.iloc[:120].copy()  # ambil subset agar grafik nyaman
dibaca

plt.figure(figsize=(12, 5))
plt.plot(sample.index, sample["actual"], label="Actual")
plt.plot(sample.index, sample["pred_linear"], label="Linear
Regression")
plt.plot(sample.index, sample["pred_rf"], label="Random Forest
Regressor")
plt.title("Prediksi Energi: Aktual vs Model")
plt.xlabel("Index waktu test")
plt.ylabel("Power Watt")
plt.legend()
plt.tight_layout()
plt.savefig(OUTPUT_DIR / "regression_predictions_compare.png")
plt.show()
```

Hal yang harus diamati

- apakah model mengikuti pola aktual,
- apakah prediksi terlalu datar,
- apakah model non-linear menangkap puncak beban lebih baik,
- apakah error besar muncul di jam tertentu.

8. Insight praktis regresi

Jika **Linear Regression** sudah cukup baik:

- hubungan data mungkin cukup linear,
- baseline sederhana mungkin sudah cukup untuk implementasi awal.

Jika **Random Forest Regressor** jauh lebih baik:

- data mengandung pola non-linear,
- *feature engineering* Anda cukup kaya untuk dimanfaatkan model ensemble.

Hands-on 24 - Klasifikasi Status Perangkat (normal, warning, fault)

1. Tujuan hands-on

Pada latihan ini, pembaca akan membangun model klasifikasi untuk memprediksi status perangkat:

- `normal`
- `warning`
- `fault`

Setelah selesai, pembaca diharapkan mampu:

- membaca dataset klasifikasi multi-kelas,
- melatih **Logistic Regression**,
- melatih **Random Forest Classifier**,
- mengukur **accuracy**, **precision**, **recall**, **F1**, dan **confusion matrix**,
- dan menafsirkan kesalahan klasifikasi.

Ini sangat relevan untuk IoT karena banyak keputusan operasional memang berbentuk status diskret seperti ini. Referensi juga menunjukkan bahwa **Random Forest Classifier** dan **Logistic Regression** banyak dipakai pada sistem IoT dan evaluasinya lazim memakai **confusion matrix** dan akurasi.

2. Contoh implementasi nyata

Smart room monitoring

Status:

- `normal`
- `warning` bila suhu/CO_2 mulai mendekati ambang
- `fault` bila kondisi buruk atau sistem gagal ventilasi

Predictive maintenance

Status:

- `normal`
- `warning` bila getaran mulai naik
- `fault` bila pola mesin jelas abnormal

Status:

- `normal`
- `warning` saat beban tinggi
- `fault` saat lonjakan ekstrem atau kondisi tidak wajar

3. Dataset yang digunakan

Gunakan dataset klasifikasi multi-kelas, misalnya:

```
data/synthetic/room_classification_dataset.csv
```

dengan kolom:

- `timestamp`
- `temperature_c`
- `humidity_pct`
- `co2_ppm`
- `power_watt`
- `status_label`

Jika sebelumnya Anda baru membuat target biner `normal vs abnormal`, untuk latihan ini kita gunakan label tiga kelas.

4. Menyiapkan dataset multi-kelas

Buat file `scripts/prepare_multiclass_status_dataset.py`:

```
from pathlib import Path
import pandas as pd
import numpy as np

INPUT_PATH = Path("data/synthetic/room_classification_dataset.csv")
OUTPUT_DIR = Path("data/processed/classification_status")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

df = pd.read_csv(INPUT_PATH)
df["timestamp"] = pd.to_datetime(df["timestamp"])
df = df.sort_values("timestamp").reset_index(drop=True)

# Pastikan label ada
if "status_label" not in df.columns:
    def make_label(row):
        if row["temperature_c"] > 33 or row["co2_ppm"] > 1400 or
row["power_watt"] > 280:
            return "fault"
        elif row["temperature_c"] > 29 or row["co2_ppm"] > 900 or
row["power_watt"] > 200:
            return "warning"
```

```
        return "normal"

    df["status_label"] = df.apply(make_label, axis=1)

# Fitur waktu
df["hour"] = df["timestamp"].dt.hour
df["day_of_week"] = df["timestamp"].dt.dayofweek
df["is_weekend"] = (df["day_of_week"] >= 5).astype(int)

# Lag dan rolling sederhana
df["temp_lag_1"] = df["temperature_c"].shift(1)
df["co2_lag_1"] = df["co2_ppm"].shift(1)
df["power_lag_1"] = df["power_watt"].shift(1)

df["temp_ma_3"] = df["temperature_c"].rolling(3).mean()
df["co2_ma_3"] = df["co2_ppm"].rolling(3).mean()
df["power_ma_3"] = df["power_watt"].rolling(3).mean()

df = df.dropna().reset_index(drop=True)

feature_cols = [
    "temperature_c", "humidity_pct", "co2_ppm", "power_watt",
    "hour", "day_of_week", "is_weekend",
    "temp_lag_1", "co2_lag_1", "power_lag_1",
    "temp_ma_3", "co2_ma_3", "power_ma_3"
]

n = len(df)
train_end = int(n * 0.70)
val_end = int(n * 0.85)

train_df = df.iloc[:train_end].copy()
val_df = df.iloc[train_end:val_end].copy()
test_df = df.iloc[val_end:].copy()

train_df.to_csv(OUTPUT_DIR / "train.csv", index=False)
val_df.to_csv(OUTPUT_DIR / "validation.csv", index=False)
test_df.to_csv(OUTPUT_DIR / "test.csv", index=False)

train_df[feature_cols].to_csv(OUTPUT_DIR / "X_train.csv", index=False)
train_df[["status_label"]].to_csv(OUTPUT_DIR / "y_train.csv",
index=False)

val_df[feature_cols].to_csv(OUTPUT_DIR / "X_val.csv", index=False)
val_df[["status_label"]].to_csv(OUTPUT_DIR / "y_val.csv", index=False)

test_df[feature_cols].to_csv(OUTPUT_DIR / "X_test.csv", index=False)
test_df[["status_label"]].to_csv(OUTPUT_DIR / "y_test.csv",
index=False)

print("Distribusi label:")
print(df["status_label"].value_counts())
print("File disimpan di:", OUTPUT_DIR)
```

Jalankan:

```
python scripts/prepare_multiclass_status_dataset.py
```

5. Logistic Regression untuk multi-kelas

Buat file `scripts/classification_logreg_status.py`:

```
from pathlib import Path
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

DATA_DIR = Path("data/processed/classification_status")

X_train = pd.read_csv(DATA_DIR / "X_train.csv")
y_train = pd.read_csv(DATA_DIR / "y_train.csv").squeeze()

X_val = pd.read_csv(DATA_DIR / "X_val.csv")
y_val = pd.read_csv(DATA_DIR / "y_val.csv").squeeze()

X_test = pd.read_csv(DATA_DIR / "X_test.csv")
y_test = pd.read_csv(DATA_DIR / "y_test.csv").squeeze()

model = LogisticRegression(max_iter=2000)
model.fit(X_train, y_train)

# Validasi
y_val_pred = model.predict(X_val)
print("=== LOGISTIC REGRESSION - VALIDATION ===")
print("Accuracy:", accuracy_score(y_val, y_val_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_val, y_val_pred, labels=["normal", "warning",
"fault"]))
print("\nClassification Report:")
print(classification_report(y_val, y_val_pred, zero_division=0))

# Test
y_test_pred = model.predict(X_test)
print("\n=== LOGISTIC REGRESSION - TEST ===")
print("Accuracy:", accuracy_score(y_test, y_test_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_test_pred, labels=["normal",
"warning", "fault"]))
print("\nClassification Report:")
print(classification_report(y_test, y_test_pred, zero_division=0))
```

6. Random Forest Classifier untuk multi-kelas

Buat file `scripts/classification_rf_status.py`:

```
from pathlib import Path
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

DATA_DIR = Path("data/processed/classification_status")

X_train = pd.read_csv(DATA_DIR / "X_train.csv")
y_train = pd.read_csv(DATA_DIR / "y_train.csv").squeeze()

X_val = pd.read_csv(DATA_DIR / "X_val.csv")
y_val = pd.read_csv(DATA_DIR / "y_val.csv").squeeze()

X_test = pd.read_csv(DATA_DIR / "X_test.csv")
y_test = pd.read_csv(DATA_DIR / "y_test.csv").squeeze()

model = RandomForestClassifier(
    n_estimators=200,
    max_depth=10,
    random_state=42,
    n_jobs=-1
)

model.fit(X_train, y_train)

# Validasi
y_val_pred = model.predict(X_val)
print("=== RANDOM FOREST CLASSIFIER - VALIDATION ===")
print("Accuracy:", accuracy_score(y_val, y_val_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_val, y_val_pred, labels=["normal", "warning",
"fault"]))
print("\nClassification Report:")
print(classification_report(y_val, y_val_pred, zero_division=0))

# Test
y_test_pred = model.predict(X_test)
print("\n=== RANDOM FOREST CLASSIFIER - TEST ===")
print("Accuracy:", accuracy_score(y_test, y_test_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_test_pred, labels=["normal",
"warning", "fault"]))
print("\nClassification Report:")
print(classification_report(y_test, y_test_pred, zero_division=0))
```

Apa yang harus dilihat?

- apakah `warning` sering salah dikira `normal`,
- apakah `fault` tertangkap baik,
- apakah model terlalu percaya diri pada kelas mayoritas.

Confusion matrix sangat penting untuk ini. Referensi juga menekankan bahwa confusion matrix membantu melihat jumlah kesalahan dan *false positives* pada klasifikasi IoT.

Hands-on 25 - Perbandingan 3 Model dengan Metrik dan Visualisasi Hasil

1. Tujuan hands-on

Pada latihan ini, pembaca akan:

- membandingkan **3 model** secara lebih sistematis,
- menyusun tabel metrik,
- membuat visualisasi sederhana hasil perbandingan,
- dan belajar memilih model berdasarkan **keseimbangan antara performa, interpretabilitas, dan kompleksitas**.

Sangat sering dalam praktik, model tidak dipilih hanya karena akurasi tertinggi. Kita juga perlu mempertimbangkan:

- stabilitas,
- kecepatan,
- kemudahan penjelasan,
- dan kecocokan dengan perangkat atau pipeline IoT.

2. Model yang dibandingkan

Untuk klasifikasi status perangkat, kita bandingkan:

- **Logistic Regression**
- **Random Forest Classifier**
- **Gradient Boosting Classifier**

Catatan:

- Anda bisa mengganti Gradient Boosting dengan XGBoost jika lingkungan Anda sudah siap,
- tetapi untuk buku ini, **Gradient Boosting dari scikit-learn** lebih praktis dan tetap open source,
- serta sejalan dengan penempatan boosting sebagai pengayaan singkat.

Referensi Perros menjelaskan bahwa boosting membangun model bertahap untuk memperbaiki kesalahan model sebelumnya, berbeda dengan random forest yang mengandalkan banyak pohon secara paralel.

3. Source code perbandingan model

Buat file `scripts/compare_three_models_status.py`:

```
from pathlib import Path
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.metrics import accuracy_score,
precision_recall_fscore_support

DATA_DIR = Path("data/processed/classification_status")
OUTPUT_DIR = Path("outputs/reports")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

X_train = pd.read_csv(DATA_DIR / "X_train.csv")
y_train = pd.read_csv(DATA_DIR / "y_train.csv").squeeze()

X_test = pd.read_csv(DATA_DIR / "X_test.csv")
y_test = pd.read_csv(DATA_DIR / "y_test.csv").squeeze()

models = {
    "LogisticRegression": LogisticRegression(max_iter=2000),
    "RandomForest": RandomForestClassifier(
        n_estimators=200, max_depth=10, random_state=42, n_jobs=-1
    ),
    "GradientBoosting": GradientBoostingClassifier(random_state=42)
}

results = []

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    precision, recall, f1, _ = precision_recall_fscore_support(
        y_test, y_pred, average="weighted", zero_division=0
    )

    results.append({
        "model": name,
        "accuracy": acc,
        "precision_weighted": precision,
        "recall_weighted": recall,
        "f1_weighted": f1
    })

results_df = pd.DataFrame(results)
results_df = results_df.sort_values("f1_weighted", ascending=False)

print("=== HASIL PERBANDINGAN MODEL ===")
print(results_df)

results_df.to_csv(OUTPUT_DIR / "model_comparison_status.csv",
index=False)
print("\nDisimpan ke:", OUTPUT_DIR / "model_comparison_status.csv")
```

Jalankan:

```
python scripts/compare_three_models_status.py
```

4. Visualisasi hasil perbandingan

Buat file `scripts/plot_model_comparison.py`:

```
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt

INPUT_PATH = Path("outputs/reports/model_comparison_status.csv")
OUTPUT_DIR = Path("outputs/figures")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

df = pd.read_csv(INPUT_PATH)

plt.figure(figsize=(10, 5))
plt.bar(df["model"], df["accuracy"])
plt.title("Perbandingan Accuracy Model")
plt.xlabel("Model")
plt.ylabel("Accuracy")
plt.tight_layout()
plt.savefig(OUTPUT_DIR / "compare_accuracy_models.png")
plt.show()

plt.figure(figsize=(10, 5))
plt.bar(df["model"], df["f1_weighted"])
plt.title("Perbandingan F1 Weighted Model")
plt.xlabel("Model")
plt.ylabel("F1 Weighted")
plt.tight_layout()
plt.savefig(OUTPUT_DIR / "compare_f1_models.png")
plt.show()
```

Mengapa F1 weighted penting?

Karena pada data IoT multi-kelas, kelas bisa tidak seimbang. Accuracy saja kadang terlalu optimistis. Referensi evaluasi klasifikasi juga menekankan pentingnya melihat precision, recall, dan F1, bukan hanya akurasi tunggal.

5. Variasi perbandingan untuk regresi

Jika Anda ingin membandingkan 3 model regresi, gunakan:

- Linear Regression
- Random Forest Regressor
- Gradient Boosting Regressor

Buat file `scripts/compare_three_regressors.py`:

```
from pathlib import Path
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error

DATA_DIR = Path("data/processed/regression_energy")
OUTPUT_DIR = Path("outputs/reports")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

X_train = pd.read_csv(DATA_DIR / "X_train.csv")
y_train = pd.read_csv(DATA_DIR / "y_train.csv").squeeze()

X_test = pd.read_csv(DATA_DIR / "X_test.csv")
y_test = pd.read_csv(DATA_DIR / "y_test.csv").squeeze()

models = {
    "LinearRegression": LinearRegression(),
    "RandomForestRegressor": RandomForestRegressor(
        n_estimators=200, max_depth=10, random_state=42, n_jobs=-1
    ),
    "GradientBoostingRegressor":
GradientBoostingRegressor(random_state=42)
}

results = []

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))

    results.append({
        "model": name,
        "mae": mae,
        "rmse": rmse
    })

results_df = pd.DataFrame(results).sort_values("rmse")
print(results_df)

results_df.to_csv(OUTPUT_DIR / "model_comparison_regression.csv",
index=False)
```

```
print("Disimpan ke:", OUTPUT_DIR / "model_comparison_regression.csv")
```

6. Cara membaca hasil perbandingan

Jika Logistic Regression menang

Kemungkinan:

- pemisahan kelas cukup sederhana,
- hubungan antar fitur cukup linear,
- model ringan sudah cukup baik.

Jika Random Forest menang

Kemungkinan:

- ada pola non-linear,
- interaksi fitur lebih kompleks,
- data tabular IoT Anda cocok untuk ensemble berbasis pohon.

Jika Gradient Boosting menang tipis

Kemungkinan:

- data cukup kompleks,
- boosting membantu memperbaiki kelemahan model sebelumnya,
- tetapi Anda tetap harus menilai apakah peningkatan itu sepadan dengan kompleksitas tambahan.

7. Kapan memilih model sederhana dibanding model kompleks?

Ini bagian yang sangat penting untuk ditekankan ke pembaca.

Pilih model sederhana jika:

- performa cukup baik,
- interpretabilitas penting,
- sistem harus ringan,
- implementasi di edge/gateway kecil,
- tim operasional butuh model yang mudah dijelaskan.

Pilih model lebih kompleks jika:

- baseline terlalu lemah,
- pola jelas non-linear,
- data cukup kaya,
- peningkatan metrik memang signifikan dan berguna secara operasional.

Referensi juga menunjukkan bahwa random forest dan ensemble bisa sangat kuat, tetapi tidak selalu harus menjadi pilihan pertama. Baseline yang sederhana sering lebih baik sebagai titik awal yang jujur.

8. Langkah kerja yang harus dilakukan pembaca

Tahap 1

Siapkan dataset regresi dan klasifikasi yang sudah *time-aware*.

Tahap 2

Latih model baseline sederhana:

- Linear Regression
- Logistic Regression

Tahap 3

Latih model ensemble:

- Random Forest Regressor
- Random Forest Classifier
- Gradient Boosting

Tahap 4

Hitung metrik:

- regresi → MAE, RMSE
- klasifikasi → accuracy, precision, recall, F1, confusion matrix

Tahap 5

Visualisasikan:

- aktual vs prediksi untuk regresi
- bar chart perbandingan model untuk klasifikasi/regresi

Tahap 6

Tafsirkan hasil bukan hanya dari skor tertinggi, tetapi juga dari:

- kestabilan,
- kemudahan implementasi,
- kemudahan penjelasan,
- dan kebutuhan lapangan.

9. Penutup

Hands-on 23–25 adalah tahap ketika pembaca benar-benar mulai **menghubungkan dataset IoT dengan model supervised yang nyata**. Pada Hands-on 23, pembaca belajar memprediksi suhu atau energi dengan regresi. Pada Hands-on 24, pembaca belajar mengklasifikasikan status perangkat ke dalam `normal`, `warning`, dan `fault`. Pada Hands-on 25, pembaca belajar bahwa memilih model terbaik bukan sekadar melihat satu angka, tetapi membandingkan beberapa model dengan metrik yang tepat dan visualisasi hasil yang jelas.

Bab 9. Unsupervised Learning dan Anomaly Detection

9.1 Mengapa unsupervised learning penting untuk IoT?

Dalam dunia IoT, salah satu masalah terbesar adalah **kurangnya label**. Banyak sensor terus menghasilkan data, tetapi tidak semua data punya penjelasan manusia yang rapi seperti:

- ini normal,
- ini warning,
- ini fault,
- ini serangan,
- ini kebocoran,
- atau ini perilaku tidak wajar.

Pada praktik lapangan, justru yang paling sering tersedia adalah **data mentah yang belum diberi label**. Sensor suhu terus mengirim angka. Sensor getaran terus memproduksi sinyal. Meter energi terus menghasilkan histori daya. Namun belum tentu ada teknisi yang sempat memberi penanda pada setiap kejadian. Di sinilah **unsupervised learning** menjadi sangat penting.

Literatur Perros menjelaskan bahwa pada data tak berlabel, algoritma unsupervised dapat membantu **mengelompokkan** data ke dalam cluster, yang pada praktiknya sering menjadi langkah awal untuk memahami struktur data sebelum klasifikasi atau sistem keputusan yang lebih formal dibangun. Literatur Saleem juga menekankan bahwa unsupervised ML tidak memakai label target sebelumnya, dan salah satu prosedur yang paling luas dipakai adalah **clustering**.

Untuk IoT, manfaatnya sangat nyata:

- menemukan pola operasi normal tanpa label manual,
- memisahkan perilaku ruangan siang vs malam,
- mengenali mode operasi mesin,
- mencari titik-titik yang tampak menyimpang,
- membangun baseline untuk alarm atau deteksi intrusi,
- dan menyiapkan data tak berlabel agar kelak bisa dipakai oleh metode supervised.

Dengan kata lain, **unsupervised learning sangat berguna ketika data banyak tetapi label sedikit**.

9.2 Posisi Bab 9 dalam alur buku

Sebelum sampai pada bab ini, pembaca dalam alur buku sudah belajar:

- memahami data sensor dan masalah kualitas data,
- membangun *dummy* dan *synthetic data*,
- membuat fitur *time-series*,
- menyiapkan dataset siap ML,
- serta melatih model supervised untuk regresi dan klasifikasi.

Setelah fondasi itu kuat, kini pembaca diajak menjawab pertanyaan yang berbeda:

- Bagaimana jika data belum punya label?
- Bagaimana jika saya hanya punya pola sensor, tetapi belum tahu kelompok operasinya?
- Bagaimana jika saya ingin mendeteksi penyimpangan tanpa menunggu dataset fault lengkap?
- Kapan cukup memakai **threshold statis**, dan kapan perlu model yang lebih pintar?

Itulah inti Bab 9.

9.3 Apa itu unsupervised learning?

Secara sederhana, **unsupervised learning** adalah pembelajaran mesin yang bekerja pada data **tanpa label target**. Algoritma mencoba memahami **struktur**, **kemiripan**, atau **penyimpangan** dalam data tanpa diberi tahu lebih dulu mana kelas yang benar.

Referensi Saleem menjelaskan bahwa unsupervised ML tidak memiliki label atau informasi target sebelumnya, dan pembelajaran dilakukan dengan mengamati bentuk serta struktur data. Referensi Perros juga menegaskan bahwa clustering adalah contoh utama unsupervised learning, yaitu mengelompokkan data yang sebelumnya belum dikelompokkan ke dalam cluster-cluster data yang mirip.

Dalam konteks IoT, unsupervised learning biasanya berguna untuk dua hal besar:

- **clustering** → menemukan pola operasi atau kelompok kondisi
- **anomaly detection** → menemukan data yang menyimpang dari pola umum

9.4 Mengapa anomaly detection sangat relevan untuk IoT?

Jika kita melihat sistem IoT nyata, banyak kejadian penting justru berupa **anomali**:

- suhu ruang server tiba-tiba melonjak,
- arus listrik naik tidak biasa,
- mesin mulai bergetar lebih kasar,
- trafik sensor tampak tidak normal,
- konsumsi energi melonjak di luar jam kerja,
- atau data sensor tampak “aneh” dibanding pola biasanya.

Referensi Obaidat menjelaskan bahwa *anomaly-based intrusion discovery* adalah masalah mengenali **infrequent patterns** yang tidak sesuai dengan trafik atau perilaku normal, dan bahwa pola-pola semacam ini disebut juga **anomalies, outliers, peculiarities**, dan seterusnya. Meskipun contoh utamanya banyak dari keamanan jaringan, logikanya sangat cocok untuk IoT umum: **cari perilaku yang tidak cocok dengan baseline**.

Karena itu, anomaly detection menjadi sangat penting pada:

- **predictive maintenance**
- **smart energy analytics**
- **smart room monitoring**
- **monitoring keamanan dan intrusi**
- **monitoring lingkungan**
- **edge analytics**

9.5 K-Means

9.5.1 Apa itu K-Means?

K-Means adalah algoritma clustering yang membagi data tak berlabel menjadi **K kelompok** berdasarkan kemiripan, biasanya diukur dengan jarak. Referensi Saleem menjelaskan bahwa tujuan K-Means adalah mempartisi dataset tak berlabel ke dalam **K cluster** dengan meminimalkan jarak antara titik data dan pusat cluster terdekatnya. Algoritma ini bekerja secara iteratif: memilih pusat cluster, mengalokasikan data ke pusat terdekat, lalu memperbarui pusat cluster sampai stabil.

9.5.2 Intuisi sederhana

Bayangkan Anda punya data ruangan dengan dua variabel:

- suhu
- kelembaban

Tanpa label apa pun, K-Means bisa membantu menemukan, misalnya:

- cluster kondisi **sejuk**
- cluster kondisi **normal**
- cluster kondisi **panas-kering**

Model tidak tahu arti cluster itu sejak awal. Tugas model hanya mengelompokkan titik-titik yang mirip.

9.5.3 Kelebihan K-Means

- sederhana
- cepat
- mudah diimplementasikan
- skalabel untuk data cukup besar
- cocok untuk eksplorasi awal

Referensi Saleem menyebut K-Means sebagai algoritma yang **scalable**, **efficient**, **highly flexible**, dan cepat konvergen, serta dipakai di banyak use case IoT seperti smart city, smart home, stream data analysis, hingga analisis konsumsi energi.

9.5.4 Keterbatasan K-Means

- harus menentukan **K** di awal
- sensitif terhadap *outlier*
- cenderung kurang cocok untuk cluster berbentuk tidak konveks
- hasil bisa berbeda tergantung inisialisasi awal

Referensi Saleem juga menyebut bahwa K-Means tidak cocok untuk cluster dengan bentuk non-convex dan sensitif terhadap outlier.

9.5.5 Contoh implementasi nyata

- mengelompokkan pola operasi ruangan dari suhu-kelembaban
- mengelompokkan pola beban energi per jam
- mengelompokkan mode kerja mesin dari fitur getaran
- menemukan pola harian penggunaan ruangan

9.5.6 Kapan K-Means cocok?

Gunakan K-Means jika:

- ingin eksplorasi pola awal
- jumlah cluster kira-kira bisa diperkirakan
- data cukup bersih
- dan bentuk cluster diperkirakan cukup sederhana

9.6 DBSCAN

9.6.1 Apa itu DBSCAN?

DBSCAN adalah algoritma clustering berbasis **kepadatan** (*density-based*). Berbeda dari K-Means yang butuh jumlah K di awal, DBSCAN mencari **daerah padat** sebagai cluster dan memperlakukan titik yang berada di daerah jarang sebagai **noise** atau data luar biasa.

Referensi Perros menegaskan bahwa salah satu keunggulan utama DBSCAN adalah **jumlah cluster tidak perlu ditentukan di awal**, berbeda dengan K-Means. Referensi Saleem juga menjelaskan bahwa DBSCAN memandang cluster sebagai daerah yang padat dan dipisahkan oleh daerah dengan kepadatan rendah atau *noisy regions*.

9.6.2 Intuisi sederhana

Jika K-Means membagi data berdasarkan pusat cluster, DBSCAN justru bertanya:

- di mana titik-titik banyak berkumpul?
- di mana titik-titik menyendiri?
- mana yang cukup padat untuk dianggap cluster?

Ini sangat menarik untuk IoT karena pada data sensor:

- kadang ada mode operasi padat,
- kadang ada data menyimpang yang muncul sendiri,
- dan bentuk cluster bisa tidak bulat.

9.6.3 Kelebihan DBSCAN

- tidak perlu menentukan jumlah cluster di awal
- lebih tahan terhadap outlier
- bisa menemukan cluster berbentuk acak / non-linear
- cocok untuk mendeteksi noise

Referensi Perros menyebut DBSCAN robust terhadap outlier dan dapat menemukan cluster yang tidak terpisah secara linear. Referensi Saleem juga menyebut DBSCAN cepat, cukup kuat pada dataset besar, dan robust terhadap outlier.

9.6.4 Keterbatasan DBSCAN

- pemilihan parameter ε dan MinPts tidak selalu mudah
- kesulitan jika kepadatan cluster sangat berbeda-beda
- hasil dipengaruhi metrik jarak

Referensi Perros dan Saleem sama-sama menekankan tantangan utama DBSCAN: pemilihan parameter dan kesulitan saat data punya kepadatan yang sangat bervariasi.

9.6.5 Contoh implementasi nyata

- mendeteksi pola ruangan yang umum vs kondisi yang terpencil
- mengenali mode beban energi yang padat vs titik-titik aneh
- memisahkan cluster getaran mesin normal dan titik noise
- mendeteksi lokasi GPS yang menyendiri dibanding rute umum

9.6.6 Kapan DBSCAN cocok?

Gunakan DBSCAN jika:

- tidak tahu jumlah cluster
- menduga ada *noise* atau outlier
- bentuk cluster kemungkinan tidak sederhana
- dan ingin algoritma yang sekaligus bisa memberi sinyal data “aneh”

9.7 K-Means vs DBSCAN

Ini perbandingan praktis yang penting untuk pembaca.

K-Means lebih cocok jika:

- ingin cluster cepat dan sederhana
- tahu kira-kira jumlah cluster
- data relatif bersih
- fokus utama pada segmentasi pola operasi

DBSCAN lebih cocok jika:

- jumlah cluster tidak diketahui
- ada kemungkinan banyak outlier
- cluster mungkin bentuknya tidak beraturan
- ingin sekaligus mendeteksi noise

Untuk pembelajaran IoT, keduanya bagus:

- **K-Means** untuk memahami pola operasi
- **DBSCAN** untuk memahami cluster dan noise dalam kondisi lebih realistis

9.8 Isolation Forest

9.8.1 Apa itu Isolation Forest?

Isolation Forest adalah algoritma anomaly detection yang bekerja dengan ide sederhana namun cerdas: **anomali cenderung lebih mudah "diisolasi" daripada data normal**. Jika data normal berkumpul di area padat, maka titik-titik aneh biasanya bisa dipisahkan dengan lebih cepat melalui pemotongan acak pada fitur.

Walaupun referensi terlampir lebih banyak menekankan anomaly detection secara umum daripada membahas Isolation Forest panjang lebar, posisi algoritma ini sangat tepat karena:

- ringan,
- cocok untuk data sensor tabular,
- tidak butuh label,
- dan efektif sebagai jembatan dari pendekatan statistik sederhana ke model anomaly detection yang lebih sistematis.

9.8.2 Intuisi sederhana

Bayangkan Anda punya data arus listrik:

- sebagian besar berada di rentang 4–8 ampere
- sesekali muncul nilai 16–20 ampere

Nilai 16–20 ampere akan lebih mudah terpisah dari kumpulan normal dibanding titik-titik yang rapat di tengah distribusi. Isolation Forest memanfaatkan ide itu.

9.8.3 Kelebihan Isolation Forest

- tidak perlu label
- cocok untuk anomaly detection tabular
- relatif cepat
- cukup baik untuk dataset sedang hingga besar
- sangat praktis untuk baseline anomaly detection pada IoT

9.8.4 Keterbatasan

- hasil masih bergantung pada kualitas fitur
- perlu penentuan *contamination* atau proporsi anomali kira-kira
- tidak selalu paling interpretatif untuk pengguna awam

9.8.5 Contoh implementasi nyata

- deteksi anomali arus listrik
- deteksi perilaku energi yang tidak biasa
- deteksi pola ruangan yang jauh dari baseline
- deteksi perubahan fitur getaran yang menyimpang

9.8.6 Kapan Isolation Forest cocok?

Gunakan jika:

- target utama adalah mencari penyimpangan
- label anomali terbatas atau tidak ada
- fitur sudah cukup baik
- data berbentuk tabular hasil *feature engineering*

9.9 One-Class SVM secara ringkas

9.9.1 Apa itu One-Class SVM?

One-Class SVM adalah pendekatan yang mencoba mempelajari **batas wilayah normal** dari data, lalu menandai titik di luar wilayah itu sebagai anomali. Jika supervised SVM memisahkan dua kelas, One-Class SVM fokus membangun deskripsi untuk **satu kelas**, yaitu kelas normal.

Walaupun referensi terlampir lebih banyak membahas SVM dalam konteks supervised classification, posisi One-Class SVM di bab ini tepat karena ia merupakan salah satu algoritma klasik yang sering dipakai untuk anomaly detection pada data tak berlabel atau semi-tak-berlabel. Referensi Perros juga menempatkan SVM sebagai teknik penting dalam ekosistem pembelajaran mesin untuk analitik IoT.

9.9.2 Intuisi sederhana

Anda punya banyak contoh data normal:

- suhu, kelembaban, daya, CO_2 pada kondisi ruangan biasa

One-Class SVM mencoba membangun “kulit pembungkus” untuk area normal itu. Titik baru yang jatuh jauh di luar pembungkus dianggap anomali.

9.9.3 Kelebihan

- cocok jika data normal banyak tetapi data anomali hampir tidak ada
- konsepnya kuat untuk “belajar normalitas”
- bisa menangkap batas non-linear dengan kernel yang tepat

9.9.4 Keterbatasan

- sensitif terhadap skala fitur
- butuh tuning parameter
- bisa lebih berat dan lebih rewel daripada Isolation Forest
- lebih sulit dijelaskan ke pemula

9.9.5 Kapan dipakai?

Untuk buku ini, One-Class SVM sebaiknya dikenalkan **singkat** sebagai:

- opsi klasik anomaly detection,
- berguna ketika hanya kelas normal yang tersedia,
- tetapi tidak harus menjadi fokus utama jika pembaca masih berada di tahap dasar.

9.10 Threshold vs model ML

Ini salah satu bagian terpenting dalam bab ini.

Di lapangan, anomaly detection tidak selalu harus dimulai dari model ML. Sering kali sistem operasional awal justru memakai **threshold statis**.

9.10.1 Pendekatan threshold

Pendekatan threshold berarti kita menentukan batas berbasis domain, misalnya:

- suhu > 30°C → warning
- suhu > 35°C → fault
- CO_2 > 1000 ppm → ventilasi perlu perhatian
- daya > 450 watt → alarm
- RMS getaran > ambang tertentu → warning

Kelebihan threshold

- sangat mudah dipahami
- cepat diimplementasikan
- sangat transparan
- ringan dijalankan
- bagus untuk baseline awal

Kekurangan threshold

- kaku
- sulit menangkap pola multivariat
- tidak adaptif terhadap drift
- bisa menghasilkan banyak alarm palsu jika kondisi sistem berubah
- sulit saat pola “aneh” tidak hanya ditentukan oleh satu nilai besar

9.10.2 Pendekatan model ML

Pendekatan model ML mencoba belajar dari pola data normal atau struktur cluster, lalu mendeteksi titik yang menyimpang.

Contohnya:

- K-Means untuk melihat titik jauh dari pusat cluster
- DBSCAN untuk memberi label noise
- Isolation Forest untuk mengisolasi anomali
- One-Class SVM untuk mempelajari wilayah normal

Kelebihan model ML

- lebih adaptif
- bisa menangkap pola multivariat
- dapat mengenali penyimpangan yang tidak terlihat dari satu threshold
- lebih cocok untuk sistem yang kompleks

Kekurangan model ML

- lebih sulit dijelaskan
- perlu preprocessing dan fitur yang baik
- perlu validasi
- lebih rentan disalahgunakan jika data tidak jujur atau tidak representatif

9.10.3 Kapan threshold cukup?

Gunakan threshold jika:

- sistem masih awal
- variabel pentingnya sedikit
- domain knowledge sangat kuat
- kebutuhan alarm harus sangat transparan
- sumber daya komputasi terbatas

Contoh:

- suhu ruang server
- daya beban maksimum
- kelembaban tanah minimum
- arus lebih

9.10.4 Kapan model ML lebih tepat?

Gunakan model ML jika:

- banyak variabel saling berinteraksi
- anomali tidak bisa dijelaskan hanya oleh satu angka
- pola berubah terhadap waktu
- threshold menghasilkan terlalu banyak alarm palsu
- sistem butuh baseline yang lebih dinamis

Contoh:

- kombinasi suhu + CO_2 + daya
- getaran + arus + suhu motor
- pola trafik sensor atau jaringan
- perilaku energi dengan banyak mode operasi

9.10.5 Pendekatan terbaik di dunia nyata

Untuk sistem IoT nyata, strategi yang paling masuk akal sering bukan memilih salah satu secara mutlak, tetapi **menggabungkan threshold dan model ML**.

Misalnya:

- threshold untuk alarm cepat yang sangat jelas
- model ML untuk mendeteksi pola aneh yang lebih halus
- dashboard menampilkan keduanya
- teknisi memakai dua sumber informasi itu bersama

9.11 Contoh-contoh nyata yang bisa diimplementasikan

Smart room monitoring

- K-Means untuk mengelompokkan mode ruangan
- DBSCAN untuk melihat kondisi noise atau cluster tak biasa
- Isolation Forest untuk mendeteksi pola ruangan yang menyimpang
- threshold untuk suhu/CO_2 ekstrem

Smart energy analytics

- K-Means untuk pola beban harian
- Isolation Forest untuk beban tidak wajar
- threshold untuk daya puncak
- One-Class SVM bila hanya data normal yang cukup banyak

Predictive maintenance sederhana

- clustering untuk mode operasi mesin
- Isolation Forest untuk fitur getaran abnormal
- threshold untuk RMS/getaran puncak
- DBSCAN untuk membedakan cluster normal dan titik noise

Keamanan / intrusion detection IoT

Referensi Obaidat menekankan bahwa anomaly detection sangat relevan untuk mengenali pola trafik yang tidak cocok dengan perilaku reguler, dan bahwa ML/DL telah memainkan peran besar dalam anomaly detection untuk jaringan IoT.

9.12 Langkah kerja yang harus dilakukan pembaca

Agar bab ini benar-benar operasional, pembaca dapat mengikuti alur berikut.

Tahap 1 — Tentukan tujuan

Apakah ingin:

- segmentasi pola operasi?
- menemukan mode ruangan?
- mendeteksi anomali?
- atau membandingkan threshold dengan model?

Tahap 2 — Siapkan fitur yang masuk akal

Gunakan:

- fitur mentah yang relevan
- atau fitur hasil *feature engineering* seperti lag, moving average, rolling std, RMS, dan sebagainya

Tahap 3 — Pilih metode awal

- K-Means jika ingin cluster sederhana
- DBSCAN jika ingin cluster + noise
- Isolation Forest jika fokus anomaly detection
- One-Class SVM jika ingin model berbasis “normal class”

Tahap 4 — Validasi dengan visualisasi

Plot data:

- scatter plot
- warna berdasarkan cluster/anomali
- grafik waktu untuk titik yang dicurigai

Tahap 5 — Bandingkan dengan domain knowledge

Periksa:

- apakah cluster masuk akal?
- apakah anomali benar-benar terlihat aneh?
- apakah threshold domain lebih mudah dipakai?

Tahap 6 — Putuskan strategi operasional

- threshold saja,
- model saja,
- atau kombinasi keduanya.

9.13 Sikap profesional dalam unsupervised learning dan anomaly detection

Ada satu pelajaran penting yang harus ditanamkan: **unsupervised learning tidak "menemukan kebenaran mutlak", tetapi membantu kita melihat struktur dan penyimpangan yang layak diperiksa**.

Artinya:

- cluster belum tentu otomatis berarti kelas bisnis tertentu,
- anomali belum tentu selalu fault,
- noise belum tentu selalu kesalahan,
- dan hasil model tetap harus ditafsirkan bersama konteks domain.

Di sistem IoT, hasil unsupervised learning paling berguna ketika diperlakukan sebagai:

- alat bantu eksplorasi,
- alat bantu monitoring,
- dan alat bantu prioritisasi pemeriksaan.

9.14 Penutup bab

Bab 9 membawa pembaca ke tahap yang sangat penting dalam IoT analytics: belajar dari data yang belum diberi label. Di sini pembaca memahami bahwa:

- **K-Means** berguna untuk menemukan pola operasi yang sederhana dan cepat,
- **DBSCAN** kuat untuk cluster yang tidak rapi dan lebih tahan terhadap *outlier*,
- **Isolation Forest** sangat praktis untuk anomaly detection pada data sensor,
- **One-Class SVM** memberi alternatif klasik untuk mempelajari batas perilaku normal,
- dan **threshold vs model ML** bukan pertarungan mutlak, melainkan pilihan desain yang harus disesuaikan dengan kebutuhan sistem.

Hands-on 21 - Clustering Pola Ruangan dari Data Suhu–kelembaban

1. Tujuan hands-on

Pada latihan ini, pembaca akan menggunakan **K-Means** dan **DBSCAN** untuk menemukan pola operasi ruangan dari data:

- suhu
- kelembaban

Tanpa label apa pun, model akan membantu menjawab:

- apakah ruangan punya beberapa mode operasi?
- apakah ada kondisi sejuk, normal, dan panas-kering?
- apakah ada titik-titik yang tampak menyendiri atau *noise*?

Ini sangat cocok untuk IoT karena data ruangan sering berulang dalam pola harian, dan tidak semua kondisi punya label manual. Referensi juga memberi contoh bahwa unsupervised learning dapat mengelompokkan data suhu tak berlabel menjadi beberapa cluster yang mewakili lokasi atau pola berbeda.

2. Contoh implementasi nyata

Smart room monitoring

Gunakan suhu dan kelembaban untuk menemukan:

- cluster ruangan nyaman
- cluster ruangan panas
- cluster ruangan kering
- kondisi aneh di luar pola umum

Greenhouse kecil

Gunakan suhu dan kelembaban untuk melihat:

- fase pagi lembap
- siang panas
- malam lebih stabil

Laboratorium komputer

Gunakan data lingkungan untuk membedakan:

- kondisi normal
- jam praktikum
- kondisi beban tinggi

3. Dataset yang dipakai

Gunakan dataset sintetis 30 hari dari bab sebelumnya:

```
data/synthetic/room_30days.csv
```

Kolom minimal:

- `timestamp`
- `device_id`
- `temperature_c`
- `humidity_pct`

4. Persiapan Ubuntu 26.04

Aktifkan lingkungan Python:

```
source .venv/bin/activate
pip install pandas numpy matplotlib scikit-learn
mkdir -p scripts outputs/figures outputs/reports
```

5. Mengapa perlu scaling?

K-Means dan DBSCAN berbasis jarak. Karena itu, fitur dengan skala lebih besar bisa mendominasi hasil. Walaupun pada kasus suhu dan kelembaban skalanya masih relatif dekat, praktik yang baik tetap melakukan **standardization** agar jarak lebih adil. Referensi clustering juga menegaskan bahwa pengukuran **distance** adalah jantung dari proses clustering.

6. Source code utama: K-Means untuk pola ruangan

Buat file `scripts/cluster_room_kmeans.py`:

```
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score

INPUT_PATH = Path("data/synthetic/room_30days.csv")
OUTPUT_DIR = Path("outputs")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

# 1. Baca data
df = pd.read_csv(INPUT_PATH)
df["timestamp"] = pd.to_datetime(df["timestamp"])

# 2. Pilih fitur
X = df[["temperature_c", "humidity_pct"]].copy()
```

```
# 3. Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. Coba beberapa nilai K
results = []
for k in range(2, 6):
    model = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = model.fit_predict(X_scaled)
    score = silhouette_score(X_scaled, labels)
    results.append((k, score))

print("=== Silhouette Score untuk beberapa K ===")
for k, score in results:
    print(f"K={k}, silhouette={score:.4f}")

# 5. Pilih K, misalnya 3
best_k = 3
kmeans = KMeans(n_clusters=best_k, random_state=42, n_init=10)
df["cluster_kmeans"] = kmeans.fit_predict(X_scaled)

print("\nJumlah data per cluster:")
print(df["cluster_kmeans"].value_counts())

# 6. Simpan hasil
df.to_csv("outputs/reports/room_kmeans_cluster.csv", index=False)

# 7. Visualisasi
plt.figure(figsize=(8, 6))
plt.scatter(
    df["temperature_c"],
    df["humidity_pct"],
    c=df["cluster_kmeans"],
    s=10
)
plt.title("K-Means Clustering: Suhu vs kelembaban")
plt.xlabel("Temperature (C)")
plt.ylabel("Humidity (%)")
plt.tight_layout()
plt.savefig("outputs/figures/room_kmeans_scatter.png")
plt.show()
```

Jalankan:

```
python scripts/cluster_room_kmeans.py
```

7. Cara membaca hasil K-Means

Perhatikan:

- apakah cluster membentuk kelompok kondisi yang masuk akal,
- apakah ada cluster sejuk-lembap, hangat-normal, dan panas-kering,
- apakah jumlah K yang dipilih memberi pemisahan yang cukup jelas.

Referensi Perros menyarankan penggunaan **elbow method** dan **silhouette score** saat mencoba beberapa nilai K untuk mencari jumlah cluster yang masuk akal.

8. Source code utama: DBSCAN untuk pola ruangan dan noise

Buat file `scripts/cluster_room_dbscan.py`:

```
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

INPUT_PATH = Path("data/synthetic/room_30days.csv")

# 1. Baca data
df = pd.read_csv(INPUT_PATH)
df["timestamp"] = pd.to_datetime(df["timestamp"])

# 2. Pilih fitur
X = df[["temperature_c", "humidity_pct"]].copy()

# 3. Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. DBSCAN
dbscan = DBSCAN(eps=0.35, min_samples=10)
df["cluster_dbscan"] = dbscan.fit_predict(X_scaled)

print("=== Hasil DBSCAN ===")
print(df["cluster_dbscan"].value_counts().sort_index())

# 5. Simpan hasil
df.to_csv("outputs/reports/room_dbscan_cluster.csv", index=False)

# 6. Visualisasi
plt.figure(figsize=(8, 6))
plt.scatter(
    df["temperature_c"],
    df["humidity_pct"],
    c=df["cluster_dbscan"],
    s=10
)
plt.title("DBSCAN: Suhu vs kelembaban")
plt.xlabel("Temperature (C)")
plt.ylabel("Humidity (%)")
plt.tight_layout()
plt.savefig("outputs/figures/room_dbscan_scatter.png")
plt.show()
```

Jalankan:

```
python scripts/cluster_room_dbscan.py
```

9. Cara membaca hasil DBSCAN

Pada DBSCAN:

- label `-1` biasanya berarti **noise**
- label `0, 1, 2, ...` berarti cluster

Perhatikan:

- apakah ada titik-titik yang dianggap noise,
- apakah noise itu memang masuk akal secara domain,
- apakah DBSCAN menangkap bentuk cluster yang lebih alami daripada K-Means.

DBSCAN sangat berguna untuk IoT ketika data mengandung *outlier* atau bentuk cluster tidak terlalu bulat. Referensi juga menegaskan bahwa DBSCAN unggul untuk **random-shaped clusters** dan lebih robust terhadap *outlier*.

10. Insight praktis Hands-on 21

Dari latihan ini, pembaca belajar bahwa:

- **K-Means** bagus untuk segmentasi cepat,
- **DBSCAN** bagus untuk cluster + noise,
- dan cluster hasil unsupervised harus tetap ditafsirkan dengan **domain knowledge**, bukan dianggap label bisnis otomatis.

Hands-on 22 - Deteksi Anomali Arus Listrik dengan Isolation Forest

1. Tujuan hands-on

Pada latihan ini, pembaca akan menggunakan **Isolation Forest** untuk mendeteksi anomali pada data arus listrik atau daya listrik. Setelah selesai, pembaca diharapkan mampu:

- membangun baseline perilaku normal,
- mendeteksi titik-titik menyimpang,
- memahami konsep *contamination*,
- dan memvisualisasikan hasil anomaly detection.

Latihan ini sangat relevan karena pada banyak sistem IoT, anomaly detection lebih realistis daripada klasifikasi penuh, terutama saat label *fault* sangat sedikit. Referensi juga menekankan bahwa unsupervised learning cocok untuk **identifying abnormalities in data**, termasuk mendeteksi akses server yang tidak biasa atau pola jaringan yang aneh. Logika yang sama sangat cocok untuk sensor listrik.

2. Contoh implementasi nyata

Smart energy analytics

- mendeteksi beban tidak wajar
- melihat penggunaan listrik di luar jam kerja
- mendeteksi lonjakan mendadak

Laboratorium kampus

- arus alat praktikum tiba-tiba melonjak
- konsumsi perangkat meningkat tidak normal

Bengkel / workshop

- motor atau pompa menarik arus di luar baseline

3. Dataset yang dipakai

Kita buat dataset sintetis kecil dengan arus listrik dan beberapa anomali buatan. Simpan sebagai:

```
data/synthetic/current_sensor_anomaly.csv
```

Script generate dataset

Buat file `scripts/generate_current_anomaly_dataset.py`:

```
from pathlib import Path
import numpy as np
import pandas as pd

OUTPUT_PATH = Path("data/synthetic/current_sensor_anomaly.csv")
OUTPUT_PATH.parent.mkdir(parents=True, exist_ok=True)

rng = np.random.default_rng(42)

timestamps = pd.date_range(
    start="2026-01-01 00:00:00",
    periods=500,
    freq="10min"
)

current_amp = rng.normal(loc=6.0, scale=0.5, size=len(timestamps))
voltage = rng.normal(loc=220.0, scale=3.0, size=len(timestamps))
power_watt = current_amp * voltage

df = pd.DataFrame({
    "timestamp": timestamps,
    "device_id": "meter01",
    "current_amp": np.round(current_amp, 3),
    "voltage": np.round(voltage, 2),
    "power_watt": np.round(power_watt, 2)
})

# Inject anomali
anomaly_idx = rng.choice(df.index, size=15, replace=False)
df.loc[anomaly_idx, "current_amp"] += rng.uniform(4, 8,
len(anomaly_idx))
df.loc[anomaly_idx, "power_watt"] = (
    df.loc[anomaly_idx, "current_amp"] * df.loc[anomaly_idx,
"voltage"]
)
df["ground_truth_anomaly"] = 0
df.loc[anomaly_idx, "ground_truth_anomaly"] = 1

df.to_csv(OUTPUT_PATH, index=False)
print("Dataset arus listrik disimpan ke:", OUTPUT_PATH)
print(df.head())
print("Jumlah anomali buatan:", df["ground_truth_anomaly"].sum())
```

Jalankan:

```
python scripts/generate_current_anomaly_dataset.py
```

4. Source code utama: Isolation Forest

Buat file `scripts/isolation_forest_current.py`:

```
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler

INPUT_PATH = Path("data/synthetic/current_sensor_anomaly.csv")

df = pd.read_csv(INPUT_PATH)
df["timestamp"] = pd.to_datetime(df["timestamp"])

# Pilih fitur
feature_cols = ["current_amp", "voltage", "power_watt"]
X = df[feature_cols].copy()

# Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Isolation Forest
model = IsolationForest(
    n_estimators=200,
    contamination=0.03,
    random_state=42
)

df["iforest_pred"] = model.fit_predict(X_scaled)
# sklearn: -1 anomali, 1 normal
df["iforest_anomaly"] = (df["iforest_pred"] == -1).astype(int)

print("=== Hasil Isolation Forest ===")
print(df["iforest_anomaly"].value_counts())

# Simpan
df.to_csv("outputs/reports/current_iforest_results.csv", index=False)

# Visualisasi
plt.figure(figsize=(12, 5))
plt.plot(df["timestamp"], df["current_amp"], label="Current (A)")
plt.scatter(
    df.loc[df["iforest_anomaly"] == 1, "timestamp"],
    df.loc[df["iforest_anomaly"] == 1, "current_amp"],
    marker="x",
    s=80,
    label="Anomaly"
)
plt.title("Isolation Forest - Deteksi Anomali Arus Listrik")
plt.xlabel("Waktu")
plt.ylabel("Current (A)")
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.savefig("outputs/figures/current_iforest_anomaly.png")
plt.show()
```

Jalankan:

```
python scripts/isolation_forest_current.py
```

5. Evaluasi sederhana terhadap ground truth sintetis

Walaupun Isolation Forest bersifat unsupervised, pada dataset sintetis kita bisa membandingkan hasilnya dengan anomali buatan.

Buat file `scripts/evaluate_iforest_current.py`:

```
from pathlib import Path
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix

INPUT_PATH = Path("outputs/reports/current_iforest_results.csv")

df = pd.read_csv(INPUT_PATH)

y_true = df["ground_truth_anomaly"]
y_pred = df["iforest_anomaly"]

print("=== Evaluasi Isolation Forest vs Ground Truth Sintetis ===")
print("Confusion Matrix:")
print(confusion_matrix(y_true, y_pred))
print("\nClassification Report:")
print(classification_report(y_true, y_pred, zero_division=0))
```

Jalankan:

```
python scripts/evaluate_iforest_current.py
```

Catatan penting

Evaluasi seperti ini hanya mungkin karena kita sengaja membuat *ground truth* sintetis. Di dunia nyata, anomaly detection sering tidak punya label sempurna. Maka hasil model harus dilihat juga melalui visualisasi, domain knowledge, dan inspeksi operasional.

Hands-on 23 - Membandingkan Threshold Statis vs Model Anomaly Detection

1. Tujuan hands-on

Pada latihan ini, pembaca akan membandingkan dua pendekatan yang sangat umum di IoT:

- **threshold statis**
- **model anomaly detection** berbasis **Isolation Forest**

Tujuannya bukan memilih “pemenang mutlak”, tetapi memahami:

- kapan threshold sudah cukup,
- kapan model ML memberi nilai tambah,
- dan bagaimana keduanya bisa dipakai bersama.

Ini sangat penting karena dalam sistem IoT nyata, threshold sering menjadi baseline pertama, sedangkan model ML ditambahkan ketika pola semakin kompleks. Referensi anomaly detection IoT juga menekankan bahwa yang dicari adalah **infrequent patterns** yang tidak sesuai dengan perilaku reguler, dan hal ini bisa didekati dengan aturan sederhana maupun model ML.

2. Strategi perbandingan

Kita gunakan dataset arus listrik yang sama:

- threshold: anomali jika `current_amp > batas`
- ML: anomali jika Isolation Forest menandai titik sebagai `-1`

Lalu kita bandingkan terhadap `ground_truth_anomaly`.

3. Source code threshold statis

Buat file `scripts/current_threshold_detection.py`:

```
from pathlib import Path
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix

INPUT_PATH = Path("data/synthetic/current_sensor_anomaly.csv")

df = pd.read_csv(INPUT_PATH)

# Threshold statis sederhana
CURRENT_THRESHOLD = 10.0
df["threshold_anomaly"] = (df["current_amp"] >
CURRENT_THRESHOLD).astype(int)

print("=== Threshold Detection ===")
print(df["threshold_anomaly"].value_counts())

y_true = df["ground_truth_anomaly"]
y_pred = df["threshold_anomaly"]

print("\nConfusion Matrix:")
print(confusion_matrix(y_true, y_pred))
print("\nClassification Report:")
print(classification_report(y_true, y_pred, zero_division=0))

df.to_csv("outputs/reports/current_threshold_results.csv",
index=False)
```

Jalankan:

```
python scripts/current_threshold_detection.py
```

4. Source code perbandingan threshold vs Isolation Forest

Buat file `scripts/compare_threshold_vs_iforest.py`:

```
from pathlib import Path
import pandas as pd
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score

THRESH_PATH = Path("outputs/reports/current_threshold_results.csv")
IFOREST_PATH = Path("outputs/reports/current_iforest_results.csv")

df_thr = pd.read_csv(THRESH_PATH)
df_if = pd.read_csv(IFOREST_PATH)

y_true = df_thr["ground_truth_anomaly"]

results = []

# Threshold
y_thr = df_thr["threshold_anomaly"]
results.append({
    "method": "Threshold",
    "accuracy": accuracy_score(y_true, y_thr),
    "precision": precision_score(y_true, y_thr, zero_division=0),
    "recall": recall_score(y_true, y_thr, zero_division=0),
    "f1": f1_score(y_true, y_thr, zero_division=0)
})

# Isolation Forest
y_if = df_if["iforest_anomaly"]
results.append({
    "method": "IsolationForest",
    "accuracy": accuracy_score(y_true, y_if),
    "precision": precision_score(y_true, y_if, zero_division=0),
    "recall": recall_score(y_true, y_if, zero_division=0),
    "f1": f1_score(y_true, y_if, zero_division=0)
})

results_df = pd.DataFrame(results)
print("=== Perbandingan Threshold vs Isolation Forest ===")
print(results_df)

results_df.to_csv("outputs/reports/threshold_vs_iforest.csv",
index=False)
```

Jalankan:

```
python scripts/compare_threshold_vs_iforest.py
```

5. Visualisasi hasil perbandingan

Buat file `scripts/plot_threshold_vs_iforest.py`:

```
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt

INPUT_PATH = Path("outputs/reports/threshold_vs_iforest.csv")
OUTPUT_DIR = Path("outputs/figures")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

df = pd.read_csv(INPUT_PATH)

plt.figure(figsize=(8, 5))
plt.bar(df["method"], df["f1"])
plt.title("Perbandingan F1: Threshold vs Isolation Forest")
plt.xlabel("Metode")
plt.ylabel("F1 Score")
plt.tight_layout()
plt.savefig(OUTPUT_DIR / "threshold_vs_iforest_f1.png")
plt.show()

plt.figure(figsize=(8, 5))
plt.bar(df["method"], df["recall"])
plt.title("Perbandingan Recall: Threshold vs Isolation Forest")
plt.xlabel("Metode")
plt.ylabel("Recall")
plt.tight_layout()
plt.savefig(OUTPUT_DIR / "threshold_vs_iforest_recall.png")
plt.show()
```

6. Cara membaca hasil

Jika threshold menang

Kemungkinan:

- anomali memang sangat jelas,
- domain knowledge cukup kuat,
- dan sistem sederhana sudah cukup.

Jika Isolation Forest menang

Kemungkinan:

- anomali tidak selalu ditentukan oleh satu ambang tunggal,
- kombinasi fitur memberi informasi tambahan,
- atau pola abnormal lebih halus.

Jika keduanya punya kekuatan berbeda

Ini sangat umum di IoT.

Contoh:

- threshold punya **precision** tinggi tetapi **recall** rendah
- Isolation Forest punya **recall** lebih baik tetapi lebih banyak alarm palsu

Dalam kondisi seperti ini, kombinasi keduanya sering paling masuk akal:

- threshold untuk alarm cepat yang sangat jelas,
- model ML untuk pemeriksaan pola halus.

7. Contoh implementasi nyata

Ruang server

- threshold suhu > 35°C untuk alarm keras
- model anomaly detection untuk kombinasi suhu + CO_2 + daya

Energi gedung

- threshold daya > 500 watt
- model anomaly detection untuk pola pemakaian tak wajar di luar jam kerja

Mesin

- threshold RMS getaran
- model ML untuk pola multivariat dari getaran + arus + suhu

Pendekatan seperti ini sangat realistis, karena dunia nyata sering membutuhkan **aturan domain yang transparan** sekaligus **analitik yang lebih adaptif**.

8. Langkah kerja yang harus dilakukan pembaca

Tahap 1

Siapkan data sensor yang cukup bersih.

Tahap 2

Mulai dari pendekatan paling sederhana:

- threshold statis

Tahap 3

Bangun model anomaly detection:

- Isolation Forest

Tahap 4

Bandingkan hasilnya terhadap:

- ground truth sintetis
- atau verifikasi visual/domain knowledge bila data nyata

Tahap 5

Lihat metrik:

- precision
- recall
- F1
- confusion matrix

Tahap 6

Putuskan strategi operasional:

- threshold saja
- model saja
- atau kombinasi keduanya

9. Penutup

Hands-on 21–23 ini membawa pembaca ke inti praktis **unsupervised learning dan anomaly detection untuk IoT**. Pada Hands-on 21, pembaca belajar memakai **K-Means** dan **DBSCAN** untuk menemukan pola operasi ruangan dari suhu dan kelembaban. Pada Hands-on 22, pembaca belajar memakai **Isolation Forest** untuk mendeteksi anomali arus listrik tanpa harus bergantung pada label penuh. Pada Hands-on 23, pembaca melihat bahwa di dunia nyata, keputusan terbaik sering bukan memilih antara **threshold** atau **model ML** secara mutlak, tetapi memahami kelebihan keduanya dan memakainya secara strategis.

Bab 10. Time-Series Forecasting untuk Sensor IoT

10.1 Mengapa forecasting penting dalam IoT?

Banyak sistem IoT tidak cukup hanya **mendeteksi kondisi saat ini**. Dalam praktik, kita sering ingin tahu **apa yang kemungkinan akan terjadi berikutnya**. Itulah inti dari *time-series forecasting*. Pada sistem sensor, pertanyaan semacam ini sangat umum:

- berapa suhu ruangan 10 menit ke depan?
- apakah beban listrik akan naik pada jam sibuk?
- apakah kelembaban tanah akan turun di bawah ambang aman?
- apakah getaran mesin cenderung meningkat menjelang fault?
- apakah pola konsumsi energi hari ini mirip dengan kemarin?

Literatur Perros menegaskan bahwa forecasting digunakan untuk menentukan **future values of a time series**, dan ini sangat penting di IoT karena data yang dilaporkan sensor sering berupa *time series*. Referensi yang sama juga membahas *stationary* dan *non-stationary time series*, *seasonality*, dan berbagai model peramalan, yang semuanya memperkuat bahwa forecasting bukan tambahan kecil, melainkan kemampuan inti dalam analitik IoT.

Dalam konteks operasional, forecasting memberi nilai nyata karena memungkinkan sistem:

- **mengantisipasi** masalah sebelum benar-benar terjadi,
- **mengoptimalkan** sumber daya,
- **menjadwalkan** tindakan lebih awal,
- dan **mengurangi risiko** keputusan yang terlambat.

10.2 Posisi Bab 10 dalam alur buku

Sebelum masuk ke forecasting, pembaca dalam alur buku ini sudah belajar:

- membersihkan data *time series*,
- membangun fitur seperti *lag*, *rolling window*, *moving average*, dan *difference*,
- menyiapkan dataset siap ML,
- memakai model supervised untuk regresi dan klasifikasi,
- serta memahami anomaly detection.

Karena itu, Bab 10 adalah tahap yang sangat alami. Jika pada bab-bab sebelumnya model sering dipakai untuk:

- memprediksi status,
- mendeteksi abnormalitas,
- atau memetakan pola operasi,

maka pada bab ini model dipakai untuk **memprediksi nilai masa depan** dari sensor.

10.3 Apa itu time-series forecasting?

Secara sederhana, **time-series forecasting** adalah upaya memprediksi nilai masa depan berdasarkan **pengamatan masa lalu yang berurutan terhadap waktu**. Berbeda dari regresi tabular biasa yang kadang boleh mengabaikan urutan, forecasting sangat bergantung pada **konteks waktu**, **lag**, **tren**, dan **musiman**.

Literatur Perros menyebutkan bahwa *time series* adalah urutan data yang diamati sepanjang waktu, dan berbagai teknik forecasting dipakai untuk memprediksi nilai berikutnya dari urutan itu. Referensi tersebut juga menjelaskan pentingnya memahami apakah suatu *time series* stasioner atau tidak, apakah ada tren, dan apakah ada seasonality.

Dalam IoT, forecasting umumnya dipakai untuk:

- suhu ruang
- konsumsi energi
- kelembaban tanah
- tekanan
- arus listrik
- getaran agregat
- dan berbagai metrik sensor lainnya

10.4 Dasar berpikir forecasting untuk sensor IoT

Sebelum masuk ke algoritma, pembaca perlu memahami bahwa data sensor IoT biasanya memiliki beberapa sifat berikut:

- **urutan waktu penting**
- **nilai sekarang dipengaruhi nilai sebelumnya**
- **sering ada pola harian atau mingguan**
- **bisa ada noise**
- **bisa ada drift**
- **kadang ada lonjakan mendadak**
- **tidak selalu stasioner**

Itulah sebabnya forecasting pada IoT tidak cukup hanya “memasang model”, tetapi perlu:

- memilih horizon prediksi yang masuk akal,
- memilih fitur historis yang relevan,
- menjaga split data tetap *time-aware*,
- dan mengevaluasi error secara jujur.

10.5 Baseline: last value, moving average

10.5.1 Mengapa baseline wajib?

Salah satu kesalahan umum dalam pembelajaran forecasting adalah terlalu cepat masuk ke model kompleks tanpa pernah membandingkan dengan baseline sederhana. Padahal, dalam banyak kasus sensor, **baseline yang sangat sederhana sudah cukup kuat**. Literatur Perros juga menempatkan **simple average model**, **weighted average**, dan **moving average or smoothing models** sebagai fondasi penting dalam forecasting.

Dalam buku ini, dua baseline utama yang sangat penting adalah:

- **last value**
- **moving average**

10.5.2 Last value

Last value berarti prediksi untuk langkah berikutnya sama dengan nilai terakhir yang diamati.

Contoh:

- jika suhu sekarang 28.4°C, maka prediksi suhu 10 menit berikutnya juga 28.4°C
- jika daya sekarang 310 watt, maka prediksi berikutnya 310 watt

Kelebihan

- sangat sederhana
- sangat cepat
- sulit dikalahkan pada data yang berubah lambat
- bagus sebagai baseline jujur

Kekurangan

- tidak menangkap tren
- tidak menangkap pola jam kerja atau musiman
- gagal jika sistem berubah cepat

10.5.3 Moving average

Moving average menggunakan rata-rata beberapa nilai terakhir sebagai prediksi berikutnya. Literatur Perros membahas *moving average* dan *smoothing models* sebagai teknik penting untuk memperhalus *time series* dan mendukung forecasting.

Contoh:

- prediksi suhu berikutnya = rata-rata 3 pengamatan terakhir
- prediksi daya berikutnya = rata-rata 6 pengamatan terakhir

Kelebihan

- lebih stabil daripada last value
- mengurangi efek noise kecil
- bagus untuk data yang bergerak perlahan

Kekurangan

- sering tertinggal saat tren berubah cepat
- puncak dan lembah tajam bisa terlalu dihaluskan

Kapan baseline ini berguna?

Sangat berguna untuk:

- baseline awal
- pembanding sebelum model ML
- sistem ringan di edge
- validasi apakah model kompleks benar-benar memberi nilai tambah

10.6 Linear regression berbasis lag

10.6.1 Inti pendekatan

Langkah berikutnya setelah baseline adalah memakai **Linear Regression berbasis fitur lag**. Konsepnya sederhana:

- ambil beberapa nilai masa lalu,
- buat menjadi fitur,
- lalu latih model untuk memprediksi nilai masa depan.

Referensi Perros menjelaskan **multivariable linear regression** sebagai teknik yang memakai beberapa variabel input untuk memperkirakan satu variabel target. Dalam konteks forecasting IoT, fitur input itu bisa berupa:

- nilai sekarang,
- lag-1,
- lag-2,
- lag-6,
- moving average,
- perbedaan,
- dan fitur waktu seperti jam/hari.

10.6.2 Mengapa cocok untuk IoT?

Karena pada data sensor tabular hasil *feature engineering*, banyak pola bisa dinyatakan cukup baik lewat kombinasi linear dari:

- histori lokal,
- rata-rata jendela,
- perbedaan terhadap nilai sebelumnya,
- dan konteks waktu.

10.6.3 Kelebihan

- mudah dipahami
- ringan
- cepat dilatih
- sangat cocok untuk baseline forecasting berbasis fitur
- mudah dijalankan di laptop dan server kecil

10.6.4 Keterbatasan

- hubungan linear sering terlalu sederhana
- kurang kuat untuk pola non-linear
- kurang bagus bila interaksi fitur sangat kompleks

10.6.5 Contoh implementasi nyata

- prediksi suhu ruang 10 menit ke depan
- prediksi beban energi 1 langkah ke depan
- prediksi kelembaban berikutnya dari histori lokal

10.7 Random Forest untuk time-series tabular

10.7.1 Mengapa Random Forest relevan?

Bila Linear Regression adalah baseline yang jernih, maka **Random Forest** adalah langkah berikutnya saat pola mulai lebih kompleks. Referensi Saleem menjelaskan bahwa *random forest* adalah metode ensemble yang membangun banyak pohon, lalu untuk regresi menghasilkan prediksi akhir dari **rata-rata semua pohon**. Referensi Mohanty juga menunjukkan bahwa **RF** dipakai sebagai salah satu metode prediksi bersama metode lain seperti ANN dan LSTM.

10.7.2 Mengapa disebut “time-series tabular”?

Karena model ini tidak langsung memproses urutan mentah seperti LSTM. Sebaliknya, kita mengubah *time series* menjadi dataset tabular melalui:

- lag features
- moving average
- rolling statistics
- fitur waktu

Lalu Random Forest belajar dari tabel itu.

10.7.3 Kelebihan

- menangkap hubungan non-linear
- kuat pada data tabular
- cukup stabil
- tidak terlalu sensitif terhadap scaling
- sering lebih kuat daripada model linear saat fitur banyak

10.7.4 Keterbatasan

- interpretasi lebih sulit
- ukuran model lebih besar
- tidak secara alami “urutan-sequence aware”, semua harus dibantu lewat fitur

10.7.5 Kapan cocok?

Gunakan ketika:

- sudah punya fitur lag yang baik
- hubungan antar fitur cukup kompleks
- baseline linear terasa terlalu lemah

- ingin performa lebih tinggi tanpa masuk ke deep learning

10.8 LSTM sederhana dengan Keras

10.8.1 Apa itu LSTM?

LSTM (Long Short-Term Memory) adalah salah satu jenis *recurrent neural network* yang dirancang untuk mempelajari dependensi jangka pendek maupun jangka panjang dalam urutan data. Referensi Mohanty menunjukkan bahwa LSTM dipakai untuk memprediksi harga penutupan saham pada hari berikutnya sebagai contoh forecasting berbasis data historis. Walaupun contoh domainnya bukan IoT sensor langsung, prinsipnya sangat relevan: LSTM dipakai ketika data memang berupa **urutan** dan kita ingin model belajar langsung dari pola temporal.

10.8.2 Mengapa LSTM menarik untuk IoT?

Karena data sensor memang berbentuk urutan waktu, dan secara teori LSTM dapat:

- menangkap pola temporal langsung,
- belajar dari sekuens nilai mentah,
- dan tidak selalu membutuhkan *feature engineering* tabular sebanyak model klasik.

10.8.3 Namun: kenapa “LSTM sederhana”?

posisi LSTM di buku ini adalah:

- pengenalan deep learning ringan,
- pembanding terhadap baseline dan model klasik,
- bukan titik awal utama.

10.8.4 Kelebihan LSTM

- cocok untuk urutan data
- bisa menangkap pola temporal yang tidak mudah ditangkap model linear
- berguna jika ada dependensi lebih panjang

10.8.5 Keterbatasan LSTM

- lebih berat
- butuh lebih banyak data
- tuning lebih rumit
- lebih sulit dijelaskan
- tidak selalu menang dari model klasik pada data tabular kecil/menengah

10.9 Kapan tidak perlu memakai LSTM

Ini salah satu bagian paling penting dalam seluruh bab.

Banyak pembelajar baru menganggap bahwa karena data sensor adalah *time series*, maka otomatis harus memakai LSTM. Itu tidak benar. Dalam banyak kasus IoT praktis, **LSTM justru tidak perlu**.

Tidak perlu memakai LSTM jika:

1. Data sedikit

Jika dataset hanya beberapa hari atau beberapa ratus observasi, model klasik sering lebih stabil.

2. Pola cukup sederhana

Jika last value, moving average, atau regresi berbasis lag sudah memberi hasil baik, LSTM mungkin hanya menambah kompleksitas.

3. Fitur tabular sudah kuat

Jika Anda sudah punya:

- lag,
- moving average,
- rolling stats,
- fitur waktu,
  maka Random Forest atau model boosting sering sangat kompetitif.

4. Sumber daya terbatas

Jika target implementasi adalah:

- edge gateway
- mini PC
- laptop operasional
- Raspberry Pi
  maka model klasik sering lebih praktis.

5. Interpretasi penting

Jika pengguna butuh model yang mudah dijelaskan, LSTM bukan pilihan terbaik.

6. Kecepatan eksperimen lebih penting

Untuk pendidikan, prototyping, dan baseline, model klasik jauh lebih cepat dicoba berulang.

Disini didorong prinsip:

> **jangan langsung memakai model besar jika model ringan sudah cukup**.

10.10 Kapan LSTM layak dicoba?

Walaupun tidak selalu perlu, LSTM tetap layak dicoba jika:

- data cukup panjang
- pola temporal kompleks
- ada dependensi yang tidak mudah ditangkap fitur manual
- baseline klasik mentok
- dan Anda memang ingin mengeksplorasi deep learning ringan

Dengan kata lain, LSTM adalah **opsi lanjutan yang masuk akal**, bukan kewajiban default.

10.11 Contoh-contoh nyata yang bisa diimplementasikan

Forecasting suhu ruang

Gunakan:

- baseline last value
- moving average
- linear regression berbasis lag
- random forest
- LSTM sederhana

Tujuan:

- prediksi suhu 10 menit ke depan
- evaluasi apakah tren panas ruangan bisa diantisipasi

Forecasting konsumsi energi

Gunakan:

- last value
- moving average
- linear regression dengan lag dan fitur waktu
- random forest regressor
- LSTM mini

Tujuan:

- prediksi daya langkah berikutnya
- prediksi pola beban harian

Forecasting kelembaban tanah

Gunakan histori:

- soil moisture
- suhu
- jam
- hari
  untuk memprediksi kebutuhan irigasi mendatang

10.12 Langkah kerja yang harus dilakukan pembaca

Agar bab ini benar-benar operasional, pembaca dapat mengikuti tahapan berikut.

Tahap 1 — Tentukan target forecasting

Misalnya:

- `temperature_next_1step`
- `power_watt_next_1step`
- `power_watt_next_6step`

Tahap 2 — Bangun baseline

Coba dulu:

- last value
- moving average

Tahap 3 — Bangun model klasik

Gunakan:

- Linear Regression berbasis lag
- Random Forest untuk dataset tabular hasil *feature engineering*

Tahap 4 — Evaluasi dengan metrik yang tepat

Pakai:

- MAE
- RMSE

Referensi forecasting juga menekankan pentingnya **forecast accuracy** dan analisis residual.

Tahap 5 — Baru coba LSTM mini

Setelah baseline dan model klasik jelas, barulah LSTM dicoba sebagai pembanding.

Tahap 6 — Bandingkan hasil secara jujur

Tanyakan:

- apakah LSTM benar-benar lebih baik?
- apakah peningkatan sepadan dengan biaya komputasi dan kompleksitas?
- apakah model klasik sebenarnya sudah cukup?

10.13 Sikap profesional dalam forecasting IoT

Satu pelajaran penting yang harus ditanamkan adalah: **forecasting yang baik bukan tentang memakai model yang paling mewah, tetapi model yang paling masuk akal untuk data dan tujuan sistem**.

Dalam praktik IoT:

- baseline sederhana sering sangat kuat,
- model klasik sering cukup untuk data tabular hasil *feature engineering*,
- dan LSTM sebaiknya dipakai ketika memang ada alasan kuat, bukan hanya karena terdengar modern.

Model yang baik adalah model yang:

- akurat secukupnya
- stabil pada data baru
- masuk akal untuk perangkat target
- mudah dirawat
- dan benar-benar membantu keputusan operasional

10.14 Penutup bab

Bab 10 membawa pembaca ke inti peramalan sensor IoT: belajar melihat beberapa langkah ke depan, bukan hanya memahami saat ini. Di sini pembaca belajar bahwa:

- **last value** dan **moving average** adalah baseline yang wajib dihormati,
- **Linear Regression berbasis lag** memberi baseline yang kuat dan jernih,
- **Random Forest untuk time-series tabular** sangat berguna ketika pola mulai non-linear,
- **LSTM sederhana** bisa menjadi pengayaan penting untuk urutan data,
- tetapi **LSTM tidak selalu perlu**, dan sering kali model klasik justru lebih praktis.

Hands-on 24 - Forecasting Suhu Ruang

1. Tujuan hands-on

Pada latihan ini, pembaca akan membangun sistem *forecasting* sederhana untuk **memprediksi suhu ruang langkah berikutnya**, misalnya **10 menit ke depan** jika data dicatat tiap 10 menit. Setelah selesai, pembaca diharapkan mampu:

- menyiapkan dataset suhu ruang untuk forecasting,
- membuat baseline **last value** dan **moving average**,
- melatih **Linear Regression berbasis lag**,
- mengevaluasi hasil dengan **MAE** dan **RMSE**,
- serta memvisualisasikan hasil prediksi.

Latihan ini sangat realistis untuk kasus:

- ruang server,
- laboratorium komputer,
- ruang kelas,
- ruang kerja,
- greenhouse kecil.

Forecasting semacam ini penting karena nilai utama IoT tidak hanya pada mengetahui kondisi saat ini, tetapi juga **mengantisipasi kondisi berikutnya** agar tindakan bisa dilakukan lebih awal.

2. Dataset yang digunakan

Gunakan dataset sintetis suhu–kelembaban 30 hari dari bab sebelumnya, misalnya:

```
data/synthetic/room_30days.csv
```

Kolom minimal:

- `timestamp`
- `device_id`
- `temperature_c`
- `humidity_pct`

3. Strategi forecasting yang dipakai

Kita akan membandingkan tiga pendekatan sederhana:

- **Baseline 1: last value**
- **Baseline 2: moving average**
- **Model klasik: Linear Regression berbasis lag**

Ini penting karena baseline yang sederhana sering kali sudah sangat kuat. Jika model ML tidak bisa mengalahkan baseline secara konsisten, maka mungkin model itu belum memberi nilai tambah yang nyata. Pendekatan ini juga sangat sejalan dengan prinsip forecasting yang sehat dalam literatur.

4. Persiapan Ubuntu 26.04

Aktifkan lingkungan Python:

```
source .venv/bin/activate
pip install pandas numpy matplotlib scikit-learn
mkdir -p scripts data/processed/forecast_room outputs/figures
outputs/reports
```

5. Menyiapkan dataset forecasting suhu

Buat file `scripts/prepare_room_forecast_dataset.py`:

```
from pathlib import Path
import numpy as np
import pandas as pd

INPUT_PATH = Path("data/synthetic/room_30days.csv")
OUTPUT_DIR = Path("data/processed/forecast_room")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

df = pd.read_csv(INPUT_PATH)
df["timestamp"] = pd.to_datetime(df["timestamp"])
df = df.sort_values("timestamp").reset_index(drop=True)

# Target 1 langkah ke depan (10 menit ke depan jika interval 10 menit)
df["target_temp_next_1step"] = df["temperature_c"].shift(-1)

# Fitur waktu
df["hour"] = df["timestamp"].dt.hour
df["day_of_week"] = df["timestamp"].dt.dayofweek
df["is_weekend"] = (df["day_of_week"] >= 5).astype(int)

# Encoding siklik jam
df["hour_sin"] = np.sin(2 * np.pi * df["hour"] / 24)
df["hour_cos"] = np.cos(2 * np.pi * df["hour"] / 24)

# Lag features
df["temp_lag_1"] = df["temperature_c"].shift(1)
df["temp_lag_2"] = df["temperature_c"].shift(2)
```

```
df["temp_lag_6"] = df["temperature_c"].shift(6)  # 1 jam
df["hum_lag_1"] = df["humidity_pct"].shift(1)

# Rolling features
df["temp_ma_3"] = df["temperature_c"].rolling(3).mean()
df["temp_ma_6"] = df["temperature_c"].rolling(6).mean()
df["temp_std_6"] = df["temperature_c"].rolling(6).std()

# Difference
df["temp_diff_1"] = df["temperature_c"].diff(1)
df["temp_diff_6"] = df["temperature_c"].diff(6)

# Hapus NaN
df = df.dropna().reset_index(drop=True)

feature_cols = [
    "temperature_c", "humidity_pct",
    "hour", "day_of_week", "is_weekend",
    "hour_sin", "hour_cos",
    "temp_lag_1", "temp_lag_2", "temp_lag_6",
    "hum_lag_1",
    "temp_ma_3", "temp_ma_6", "temp_std_6",
    "temp_diff_1", "temp_diff_6"
]

target_col = "target_temp_next_1step"

# Split time-aware
n = len(df)
train_end = int(n * 0.70)
val_end = int(n * 0.85)

train_df = df.iloc[:train_end].copy()
val_df = df.iloc[train_end:val_end].copy()
test_df = df.iloc[val_end:].copy()

train_df.to_csv(OUTPUT_DIR / "train.csv", index=False)
val_df.to_csv(OUTPUT_DIR / "validation.csv", index=False)
test_df.to_csv(OUTPUT_DIR / "test.csv", index=False)

train_df[feature_cols].to_csv(OUTPUT_DIR / "X_train.csv", index=False)
train_df[[target_col]].to_csv(OUTPUT_DIR / "y_train.csv", index=False)

val_df[feature_cols].to_csv(OUTPUT_DIR / "X_val.csv", index=False)
val_df[[target_col]].to_csv(OUTPUT_DIR / "y_val.csv", index=False)

test_df[feature_cols].to_csv(OUTPUT_DIR / "X_test.csv", index=False)
test_df[[target_col]].to_csv(OUTPUT_DIR / "y_test.csv", index=False)

print("Dataset forecasting suhu siap digunakan.")
print(f"Train: {len(train_df)}, Validation: {len(val_df)}, Test:
{len(test_df)}")
print("File disimpan di:", OUTPUT_DIR)
```

Jalankan:

```
python scripts/prepare_room_forecast_dataset.py
```

6. Baseline forecasting suhu: last value dan moving average

Buat file `scripts/baseline_room_forecast.py`:

```
from pathlib import Path
import pandas as pd
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error

DATA_DIR = Path("data/processed/forecast_room")

test_df = pd.read_csv(DATA_DIR / "test.csv")

y_true = test_df["target_temp_next_1step"]

# Baseline 1: last value
y_pred_last = test_df["temperature_c"]

# Baseline 2: moving average 3 langkah terakhir
y_pred_ma = test_df["temp_ma_3"]

def evaluate(name, y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    print(f"=== {name} ===")
    print("MAE :", mae)
    print("RMSE:", rmse)
    print()

evaluate("Baseline Last Value", y_true, y_pred_last)
evaluate("Baseline Moving Average", y_true, y_pred_ma)

result_df = pd.DataFrame({
    "actual": y_true,
    "pred_last": y_pred_last,
    "pred_ma": y_pred_ma
})
result_df.to_csv(DATA_DIR / "baseline_predictions.csv", index=False)
print("Hasil baseline disimpan ke:", DATA_DIR /
"baseline_predictions.csv")
```

Jalankan:

```
python scripts/baseline_room_forecast.py
```

Mengapa dua baseline ini penting?

Karena:

- **last value** sering sangat kuat jika perubahan antar-langkah kecil,
- **moving average** membantu menghaluskan *noise*,
- dan keduanya menjadi patokan jujur sebelum model ML dipakai.

7. Linear Regression berbasis lag untuk suhu

Buat file `scripts/linear_regression_room_forecast.py`:

```
from pathlib import Path
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error

DATA_DIR = Path("data/processed/forecast_room")

X_train = pd.read_csv(DATA_DIR / "X_train.csv")
y_train = pd.read_csv(DATA_DIR / "y_train.csv").squeeze()

X_val = pd.read_csv(DATA_DIR / "X_val.csv")
y_val = pd.read_csv(DATA_DIR / "y_val.csv").squeeze()

X_test = pd.read_csv(DATA_DIR / "X_test.csv")
y_test = pd.read_csv(DATA_DIR / "y_test.csv").squeeze()

model = LinearRegression()
model.fit(X_train, y_train)

y_val_pred = model.predict(X_val)
y_test_pred = model.predict(X_test)

def evaluate(name, y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    print(f"=== {name} ===")
    print("MAE :", mae)
    print("RMSE:", rmse)
    print()

evaluate("Linear Regression - Validation", y_val, y_val_pred)
evaluate("Linear Regression - Test", y_test, y_test_pred)

result_df = pd.DataFrame({
    "actual": y_test,
    "pred_linear": y_test_pred
})
result_df.to_csv(DATA_DIR / "linear_predictions.csv", index=False)
print("Prediksi Linear Regression disimpan ke:", DATA_DIR /
"linear_predictions.csv")
```

Jalankan:

```
python scripts/linear_regression_room_forecast.py
```

Apa yang harus diperhatikan?

- apakah MAE lebih kecil dari baseline,
- apakah model cukup stabil,
- apakah prediksi mengikuti pola suhu aktual.

8. Visualisasi hasil forecasting suhu

Buat file `scripts/plot_room_forecast_comparison.py`:

```
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt

DATA_DIR = Path("data/processed/forecast_room")
OUTPUT_DIR = Path("outputs/figures")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

base_df = pd.read_csv(DATA_DIR / "baseline_predictions.csv")
lin_df = pd.read_csv(DATA_DIR / "linear_predictions.csv")

plot_df = pd.DataFrame({
    "actual": lin_df["actual"],
    "pred_last": base_df["pred_last"],
    "pred_ma": base_df["pred_ma"],
    "pred_linear": lin_df["pred_linear"]
})

sample = plot_df.iloc[:120].copy()

plt.figure(figsize=(12, 5))
plt.plot(sample.index, sample["actual"], label="Actual")
plt.plot(sample.index, sample["pred_last"], label="Last Value")
plt.plot(sample.index, sample["pred_ma"], label="Moving Average")
plt.plot(sample.index, sample["pred_linear"], label="Linear
Regression")
plt.title("Forecasting Suhu Ruang")
plt.xlabel("Index waktu test")
plt.ylabel("Temperature (C)")
plt.legend()
plt.tight_layout()
plt.savefig(OUTPUT_DIR / "room_forecast_compare.png")
plt.show()
```

Hands-on 25 - Forecasting Konsumsi Energi Harian

1. Tujuan hands-on

Pada latihan ini, pembaca akan membangun forecasting untuk **konsumsi energi** dengan horizon lebih operasional, misalnya:

- prediksi **1 langkah ke depan**
- atau analitik untuk pola **harian**

Secara praktis, latihan ini memakai dataset energi hasil bab sebelumnya dan melatih:

- **Linear Regression**
- **Random Forest Regressor**

Forecasting energi sangat relevan untuk IoT karena berkaitan langsung dengan:

- optimasi beban,
- penghematan energi,
- prediksi jam sibuk,
- dan kesiapan infrastruktur. Literatur juga secara eksplisit mengaitkan regresi dan forecasting dengan **energy usage prediction**.

2. Dataset yang digunakan

Gunakan dataset yang sudah dipersiapkan di bab sebelumnya:

```
data/processed/regression_energy/
├── X_train.csv
├── y_train.csv
├── X_val.csv
├── y_val.csv
├── X_test.csv
└── y_test.csv
```

3. Baseline model: Linear Regression

Buat file `scripts/forecast_energy_linear.py`:

```
from pathlib import Path
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error

DATA_DIR = Path("data/processed/regression_energy")

X_train = pd.read_csv(DATA_DIR / "X_train.csv")
y_train = pd.read_csv(DATA_DIR / "y_train.csv").squeeze()

X_val = pd.read_csv(DATA_DIR / "X_val.csv")
y_val = pd.read_csv(DATA_DIR / "y_val.csv").squeeze()

X_test = pd.read_csv(DATA_DIR / "X_test.csv")
y_test = pd.read_csv(DATA_DIR / "y_test.csv").squeeze()

model = LinearRegression()
model.fit(X_train, y_train)

y_val_pred = model.predict(X_val)
y_test_pred = model.predict(X_test)

def evaluate(name, y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    print(f"=== {name} ===")
    print("MAE :", mae)
    print("RMSE:", rmse)
    print()

evaluate("Linear Regression - Validation", y_val, y_val_pred)
evaluate("Linear Regression - Test", y_test, y_test_pred)

result_df = pd.DataFrame({
    "actual": y_test,
    "pred_linear": y_test_pred
})
result_df.to_csv(DATA_DIR / "energy_linear_predictions.csv",
index=False)
print("Hasil Linear Regression disimpan.")
```

4. Model kedua: Random Forest Regressor

Buat file `scripts/forecast_energy_rf.py`:

```
from pathlib import Path
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error

DATA_DIR = Path("data/processed/regression_energy")
```

```
X_train = pd.read_csv(DATA_DIR / "X_train.csv")
y_train = pd.read_csv(DATA_DIR / "y_train.csv").squeeze()

X_val = pd.read_csv(DATA_DIR / "X_val.csv")
y_val = pd.read_csv(DATA_DIR / "y_val.csv").squeeze()

X_test = pd.read_csv(DATA_DIR / "X_test.csv")
y_test = pd.read_csv(DATA_DIR / "y_test.csv").squeeze()

model = RandomForestRegressor(
    n_estimators=200,
    max_depth=10,
    random_state=42,
    n_jobs=-1
)
model.fit(X_train, y_train)

y_val_pred = model.predict(X_val)
y_test_pred = model.predict(X_test)

def evaluate(name, y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    print(f"=== {name} ===")
    print("MAE :", mae)
    print("RMSE:", rmse)
    print()

evaluate("Random Forest - Validation", y_val, y_val_pred)
evaluate("Random Forest - Test", y_test, y_test_pred)

result_df = pd.DataFrame({
    "actual": y_test,
    "pred_rf": y_test_pred
})
result_df.to_csv(DATA_DIR / "energy_rf_predictions.csv", index=False)
print("Hasil Random Forest disimpan.")
```

Mengapa Random Forest relevan?

Karena pola energi sering tidak sepenuhnya linear. Ada interaksi:

- jam,
- hari kerja,
- beban lokal,
- rata-rata 1 jam terakhir,
- lonjakan periodik.
  Model ensemble sering lebih baik menangkap pola seperti ini.

5. Visualisasi prediksi energi

Buat file `scripts/plot_energy_forecast.py`:

```
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt

DATA_DIR = Path("data/processed/regression_energy")
OUTPUT_DIR = Path("outputs/figures")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

lin_df = pd.read_csv(DATA_DIR / "energy_linear_predictions.csv")
rf_df = pd.read_csv(DATA_DIR / "energy_rf_predictions.csv")

plot_df = pd.DataFrame({
    "actual": lin_df["actual"],
    "pred_linear": lin_df["pred_linear"],
    "pred_rf": rf_df["pred_rf"]
})

sample = plot_df.iloc[:150].copy()

plt.figure(figsize=(12, 5))
plt.plot(sample.index, sample["actual"], label="Actual")
plt.plot(sample.index, sample["pred_linear"], label="Linear
Regression")
plt.plot(sample.index, sample["pred_rf"], label="Random Forest")
plt.title("Forecasting Konsumsi Energi")
plt.xlabel("Index waktu test")
plt.ylabel("Power Watt")
plt.legend()
plt.tight_layout()
plt.savefig(OUTPUT_DIR / "energy_forecast_compare.png")
plt.show()
```

Hal yang harus diperiksa

- apakah model mengikuti puncak dan lembah,
- apakah prediksi terlalu halus,
- apakah Random Forest menangkap pola jam sibuk lebih baik.

Hands-on 26 - Membandingkan Baseline vs Random Forest vs LSTM Mini

1. Tujuan hands-on

Pada latihan ini, pembaca akan membandingkan tiga pendekatan forecasting:

- **baseline sederhana**
- **Random Forest**
- **LSTM mini**

Tujuannya bukan membuktikan bahwa LSTM selalu terbaik, tetapi justru menunjukkan secara jujur:

- kapan baseline sudah cukup,
- kapan model klasik unggul,
- kapan LSTM memberi nilai tambah,
- dan kapan LSTM justru tidak perlu.

2. Dataset yang digunakan

Agar lebih mudah, kita gunakan dataset suhu ruang:

```
data/processed/forecast_room/
```

Untuk LSTM, kita akan membuat *sequence* sederhana dari suhu mentah.

3. Membandingkan baseline dan Random Forest

Sebelum masuk ke LSTM, kita buat ringkasan baseline vs Random Forest pada dataset suhu. Random Forest dipakai karena menjadi model klasik utama untuk *time-series tabular*.

Buat file `scripts/forecast_room_rf.py`:

```
from pathlib import Path
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error

DATA_DIR = Path("data/processed/forecast_room")

X_train = pd.read_csv(DATA_DIR / "X_train.csv")
y_train = pd.read_csv(DATA_DIR / "y_train.csv").squeeze()

X_test = pd.read_csv(DATA_DIR / "X_test.csv")
y_test = pd.read_csv(DATA_DIR / "y_test.csv").squeeze()

model = RandomForestRegressor(
    n_estimators=200,
    max_depth=10,
    random_state=42,
    n_jobs=-1
)
model.fit(X_train, y_train)

y_test_pred = model.predict(X_test)

mae = mean_absolute_error(y_test, y_test_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))

print("=== RANDOM FOREST ROOM FORECAST ===")
print("MAE :", mae)
print("RMSE:", rmse)

result_df = pd.DataFrame({
    "actual": y_test,
    "pred_rf": y_test_pred
})
result_df.to_csv(DATA_DIR / "rf_predictions.csv", index=False)
print("Hasil Random Forest suhu disimpan.")
```

4. Persiapan TensorFlow/Keras

Pasang TensorFlow:

```
source .venv/bin/activate
pip install tensorflow
```

Catatan praktis:

- pada beberapa sistem Ubuntu, install TensorFlow bisa lebih berat daripada scikit-learn,
- ini salah satu alasan mengapa LSTM tidak selalu menjadi pilihan awal.

5. Menyiapkan sequence untuk LSTM mini

Buat file `scripts/prepare_lstm_room_sequences.py`:

```
from pathlib import Path
import numpy as np
import pandas as pd

INPUT_PATH = Path("data/synthetic/room_30days.csv")
OUTPUT_DIR = Path("data/processed/lstm_room")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

df = pd.read_csv(INPUT_PATH)
df["timestamp"] = pd.to_datetime(df["timestamp"])
df = df.sort_values("timestamp").reset_index(drop=True)

series = df["temperature_c"].values.astype("float32")

window_size = 12  # 12 langkah = 2 jam jika interval 10 menit

X = []
y = []

for i in range(window_size, len(series)):
    X.append(series[i-window_size:i])
    y.append(series[i])

X = np.array(X)
y = np.array(y)

# reshape untuk LSTM: [samples, timesteps, features]
X = X.reshape((X.shape[0], X.shape[1], 1))

# split time-aware
n = len(X)
train_end = int(n * 0.70)
val_end = int(n * 0.85)

X_train, X_val, X_test = X[:train_end], X[train_end:val_end],
X[val_end:]
y_train, y_val, y_test = y[:train_end], y[train_end:val_end],
y[val_end:]
```

```
np.save(OUTPUT_DIR / "X_train.npy", X_train)
np.save(OUTPUT_DIR / "X_val.npy", X_val)
np.save(OUTPUT_DIR / "X_test.npy", X_test)
np.save(OUTPUT_DIR / "y_train.npy", y_train)
np.save(OUTPUT_DIR / "y_val.npy", y_val)
np.save(OUTPUT_DIR / "y_test.npy", y_test)

print("Sequence LSTM disimpan di:", OUTPUT_DIR)
print("Shape X_train:", X_train.shape)
```

6. LSTM mini dengan Keras

Buat file `scripts/lstm_room_forecast.py`:

```
from pathlib import Path
import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import mean_absolute_error, mean_squared_error

DATA_DIR = Path("data/processed/lstm_room")

X_train = np.load(DATA_DIR / "X_train.npy")
X_val = np.load(DATA_DIR / "X_val.npy")
X_test = np.load(DATA_DIR / "X_test.npy")

y_train = np.load(DATA_DIR / "y_train.npy")
y_val = np.load(DATA_DIR / "y_val.npy")
y_test = np.load(DATA_DIR / "y_test.npy")

model = Sequential([
    LSTM(32, input_shape=(X_train.shape[1], X_train.shape[2])),
    Dense(1)
])

model.compile(optimizer="adam", loss="mse")

early_stop = EarlyStopping(
    monitor="val_loss",
    patience=5,
    restore_best_weights=True
)

history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=30,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)

y_test_pred = model.predict(X_test).flatten()

mae = mean_absolute_error(y_test, y_test_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))
```

```
print("=== LSTM MINI ===")
print("MAE :", mae)
print("RMSE:", rmse)

result_df = pd.DataFrame({
    "actual": y_test,
    "pred_lstm": y_test_pred
})
result_df.to_csv(DATA_DIR / "lstm_predictions.csv", index=False)
print("Prediksi LSTM disimpan.")
```

Mengapa ini disebut LSTM mini?

Karena arsitekturnya sengaja dibuat sangat sederhana:

- satu layer LSTM
- satu output Dense
- tanpa tuning rumit
- cukup untuk pembelajaran dan pembanding

Ini sejalan untuk lebih memilih **satu model deep learning yang benar-benar dijelaskan** daripada banyak model yang hanya disentuh permukaan.

7. Perbandingan akhir: baseline vs RF vs LSTM

Buat file `scripts/compare_room_forecast_models.py`:

```
from pathlib import Path
import pandas as pd
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error

BASE_DIR = Path("data/processed/forecast_room")
LSTM_DIR = Path("data/processed/lstm_room")
OUTPUT_DIR = Path("outputs/reports")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

base_df = pd.read_csv(BASE_DIR / "baseline_predictions.csv")
rf_df = pd.read_csv(BASE_DIR / "rf_predictions.csv")
lstm_df = pd.read_csv(LSTM_DIR / "lstm_predictions.csv")

# Ambil panjang minimum agar bisa dibandingkan
n = min(len(base_df), len(rf_df), len(lstm_df))

actual = base_df["actual"].iloc[:n].reset_index(drop=True)
pred_last = base_df["pred_last"].iloc[:n].reset_index(drop=True)
pred_ma = base_df["pred_ma"].iloc[:n].reset_index(drop=True)
pred_rf = rf_df["pred_rf"].iloc[:n].reset_index(drop=True)
pred_lstm = lstm_df["pred_lstm"].iloc[:n].reset_index(drop=True)

def metrics(y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    return mae, rmse

results = []

for name, pred in [
    ("LastValue", pred_last),
    ("MovingAverage", pred_ma),
    ("RandomForest", pred_rf),
    ("LSTMmini", pred_lstm),
]:
    mae, rmse = metrics(actual, pred)
    results.append({
        "model": name,
        "mae": mae,
        "rmse": rmse
    })

results_df = pd.DataFrame(results).sort_values("rmse")
print(results_df)
results_df.to_csv(OUTPUT_DIR / "room_forecast_model_comparison.csv",
index=False)
print("Hasil perbandingan disimpan ke:", OUTPUT_DIR /
"room_forecast_model_comparison.csv")
```

8. Visualisasi perbandingan model

Buat file `scripts/plot_room_forecast_model_comparison.py`:

```
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt

INPUT_PATH =
Path("outputs/reports/room_forecast_model_comparison.csv")
OUTPUT_DIR = Path("outputs/figures")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

df = pd.read_csv(INPUT_PATH)

plt.figure(figsize=(10, 5))
plt.bar(df["model"], df["mae"])
plt.title("Perbandingan MAE Forecasting Suhu")
plt.xlabel("Model")
plt.ylabel("MAE")
plt.tight_layout()
plt.savefig(OUTPUT_DIR / "room_compare_mae.png")
plt.show()

plt.figure(figsize=(10, 5))
plt.bar(df["model"], df["rmse"])
plt.title("Perbandingan RMSE Forecasting Suhu")
plt.xlabel("Model")
plt.ylabel("RMSE")
plt.tight_layout()
plt.savefig(OUTPUT_DIR / "room_compare_rmse.png")
plt.show()
```

9. Cara membaca hasil

Jika last value menang

Kemungkinan:

- perubahan suhu antar-langkah sangat kecil,
- sistem cukup stabil,
- model kompleks belum memberi nilai tambah nyata.

Jika moving average menang

Kemungkinan:

- data cukup berisik,
- smoothing ringan sudah membantu.

Jika Random Forest menang

Kemungkinan:

- fitur lag dan waktu cukup kaya,
- hubungan non-linear penting,
- model tabular klasik sudah sangat kuat.

Jika LSTM menang tipis

Kemungkinan:

- ada pola temporal yang lebih panjang,
- tetapi Anda tetap harus menilai apakah kenaikan performa sepadan dengan biaya komputasi.

Jika LSTM tidak menang

Itu bukan kegagalan. Justru itu pelajaran penting: **tidak semua forecasting IoT perlu deep learning**.

10. Kapan tidak perlu memakai LSTM?

Berdasarkan praktik di bab ini, pembaca akan melihat sendiri bahwa LSTM tidak selalu perlu jika:

- data tidak terlalu banyak,
- pola cukup sederhana,
- baseline sudah kuat,
- fitur lag dan rolling sudah informatif,
- sistem perlu ringan,
- dan interpretabilitas lebih penting.

Ini sesuai dengan arah bab yang menekankan bahwa model klasik sering sudah cukup kuat untuk forecasting sensor IoT kecil hingga menengah.

11. Langkah kerja yang harus dilakukan pembaca

Tahap	Uraian
1	Siapkan dataset forecasting suhu dan energi berbasis waktu.
2	Bangun baseline: last value dan moving average.
3	Bangun model klasik: Linear Regression dan Random Forest Regressor.
4	Siapkan sequence untuk LSTM mini.
5	Latih LSTM sederhana dengan Keras.
6	Bandingkan semua model dengan MAE, RMSE, grafik prediksi vs aktual, dan bar chart perbandingan metrik.
7	Tafsirkan hasil secara jujur: mana yang benar-benar lebih baik, mana yang paling praktis, dan mana yang paling cocok untuk implementasi IoT.

12. Penutup

Hands-on 24–26 ini adalah bagian yang sangat penting karena disinilah pembaca benar-benar melihat bahwa **forecasting IoT bukan soal langsung memakai model yang paling canggih, tetapi soal membangun pembanding yang jujur dan memilih model yang paling masuk akal**. Pada Hands-on 24, pembaca belajar forecasting suhu ruang dengan baseline dan regresi berbasis lag. Pada Hands-on 25, pembaca belajar forecasting konsumsi energi dengan model klasik yang sangat relevan untuk IoT. Pada Hands-on 26, pembaca membandingkan **baseline vs Random Forest vs LSTM mini**, lalu melihat sendiri bahwa **LSTM bisa berguna, tetapi tidak selalu perlu**.

Bab 11. Edge AI dan TinyML yang Realistis

11.1 Mengapa Edge AI penting untuk IoT?

Selama bab-bab sebelumnya, pembaca sudah belajar membangun alur:

- sensor atau simulator
- broker MQTT
- penyimpanan
- preprocessing
- model
- dashboard

Namun dalam sistem IoT nyata, ada satu pertanyaan yang sangat penting:
di mana model dijalankan?

Apakah model dijalankan:

- di laptop?
- di server kampus?
- di cloud?
- di gateway lokal?
- atau langsung di perangkat seperti ESP32 atau Raspberry Pi?

Pertanyaan ini tidak hanya soal teknis, tetapi soal **latensi**, **biaya bandwidth**, **privasi**, **keandalan saat internet buruk**, dan **konsumsi daya**. Referensi edge computing menegaskan bahwa ketika komputasi dipindahkan lebih dekat ke pengguna atau sumber data, penundaan transmisi WAN dapat dikurangi, analisis real-time menjadi lebih mungkin, dan beban jaringan inti ikut berkurang.

Dalam konteks IoT, Edge AI menjadi sangat menarik karena:

- sensor menghasilkan data terus-menerus
- tidak semua data perlu dikirim utuh ke cloud
- beberapa keputusan harus dibuat cepat
- beberapa sistem tetap harus berjalan walau koneksi internet tidak stabil
- dan ada kasus di mana data sensitif lebih aman diproses dekat sumbernya

Dengan kata lain, **Edge AI adalah cara membuat sistem IoT lebih responsif, lebih hemat, dan lebih mandiri**.

11.2 Posisi Bab 11 dalam alur buku

Dalam alur buku ini, pembaca sudah sampai pada tahap di mana:

- dataset sensor sudah dipahami
- preprocessing sudah dilakukan
- fitur sudah dibangun
- model supervised, anomaly detection, dan forecasting sudah dicoba
- visualisasi dan dashboard sudah dibuat

Maka pertanyaan berikutnya sangat logis:

- bagaimana model ini dijalankan dekat perangkat?
- bagaimana membedakan tahap **training** dan **inference**?
- bagaimana mengubah model menjadi format yang lebih ringan?
- perangkat apa yang realistis untuk target praktikum?
- bagaimana menjalankan TensorFlow Lite secara praktis?

Itulah ruang lingkup Bab 11.

11.3 Inferensi di edge

11.3.1 Apa itu inferensi?

Inferensi adalah proses ketika model yang sudah dilatih menerima data baru lalu menghasilkan keluaran, misalnya:

- prediksi suhu berikutnya
- status `normal / warning / fault`
- skor anomali
- hasil klasifikasi sederhana

Banyak orang baru belajar ML fokus pada training, padahal pada sistem IoT nyata, yang jauh lebih sering terjadi adalah **inferensi**. Sensor terus mengirim data, dan sistem harus terus meresponsnya.

11.3.2 Apa itu inferensi di edge?

Inferensi di edge berarti model dijalankan **dekat sumber data**, misalnya pada:

- gateway lokal
- mini PC
- Raspberry Pi
- perangkat edge industrial
- atau mikrokontroler tertentu bila model sangat ringan

Bukan berarti semua komputasi harus selalu dilakukan di edge. Maksudnya adalah bagian yang perlu cepat, lokal, dan responsif didekatkan ke perangkat.

Referensi edge computing menjelaskan bahwa edge menyediakan **communication, storage, and computing resources at the edge of the network** dan sangat cocok untuk layanan dengan kebutuhan **real-time** tinggi. Dalam skenario kampus, taman industri, atau lingkungan lokal, perangkat dengan kemampuan komputasi dan storage dapat dipakai sebagai edge device. Bahkan router rumah atau gateway lokal pun bisa berperan sebagai edge node.

11.3.3 Mengapa inferensi di edge menarik?

Keuntungan inferensi di edge:

- **latensi lebih rendah**
  keputusan bisa dibuat lebih cepat tanpa menunggu cloud
- **lebih hemat bandwidth**
  tidak semua data mentah perlu dikirim
- **lebih tahan terhadap internet yang putus-putus**
- **lebih baik untuk privasi** pada beberapa use case
- **lebih realistis untuk sistem lokal**, kampus, bengkel, greenhouse, atau sekolah

Referensi juga menegaskan bahwa pemrosesan di edge dapat mengurangi beban transmisi dan mempercepat layanan dengan kebutuhan real-time.

11.3.4 Contoh implementasi nyata

Smart room monitoring

- sensor suhu, kelembaban, CO_2 mengirim data
- Raspberry Pi lokal menghitung status ruangan
- hanya hasil ringkas atau alarm yang dikirim ke dashboard pusat

Predictive maintenance sederhana

- data getaran dan arus motor dibaca lokal
- gateway menghitung fitur atau skor anomali
- hanya event penting yang diteruskan

Smart greenhouse

- sensor tanah, suhu, kelembaban dianalisis di edge
- keputusan penyiraman dasar dilakukan lokal
- cloud dipakai untuk histori dan analitik lanjut

11.4 Pelatihan di laptop/cloud vs inferensi di perangkat

Ini salah satu konsep terpenting dalam bab ini.

11.4.1 Mengapa training dan inferensi sebaiknya dibedakan?

Dalam praktik IoT modern, sangat umum bahwa:

- **training** dilakukan pada perangkat yang lebih kuat
- **inferensi** dilakukan di perangkat yang lebih dekat dengan data

Referensi edge AI yang dilampirkan menunjukkan pola ini dengan sangat jelas. Pada contoh UAV inspection, **AI training services** diletakkan di cloud data center yang punya komputasi besar, sedangkan analisis lokal didekatkan ke edge/MEC platform untuk mengurangi delay dan beban jaringan. Referensi sistem kesehatan edge-based juga memperlihatkan pembagian tiga lapis: akuisisi data, edge computing, dan visualisasi.

11.4.2 Pelatihan di laptop atau cloud

Training biasanya lebih cocok dilakukan di:

- laptop pengembang
- workstation lab
- server kampus
- cloud
- atau mesin lokal yang lebih kuat

Karena training membutuhkan:

- dataset lebih besar
- eksperimen berulang
- validasi
- tuning parameter
- monitoring metrik
- dan kadang memori/CPU/GPU lebih besar

11.4.3 Inferensi di perangkat

Setelah model selesai dilatih dan dievaluasi, model dapat diekspor ke format yang lebih ringan, lalu dijalankan di:

- Raspberry Pi
- mini PC Ubuntu
- gateway lokal
- atau mikrokontroler tertentu jika model cukup kecil

11.4.4 Pola kerja yang sehat

Pola yang paling masuk akal:

Tahap 1 — Training

- dilakukan di laptop Ubuntu / server / cloud
- gunakan dataset lengkap
- bandingkan beberapa model
- pilih model terbaik

Tahap 2 — Konversi

- ekspor model ke format lebih ringan
- sederhanakan bila perlu

Tahap 3 — Deployment inferensi

- jalankan model pada Raspberry Pi / gateway / perangkat praktis
- kirim hasil ke MQTT / dashboard / notifikasi

11.4.5 Mengapa pola ini realistis?

Karena:

- training tidak perlu dilakukan berulang-ulang di perangkat kecil
- edge device bisa difokuskan untuk pekerjaan inferensi
- sistem jadi lebih stabil dan mudah dipelihara

11.5 Konversi model ringan

11.5.1 Mengapa model perlu dikonversi?

Model yang berjalan nyaman di laptop belum tentu cocok langsung untuk edge device. Masalah yang umum muncul:

- ukuran file model terlalu besar
- RAM perangkat terlalu kecil
- latensi inferensi terlalu tinggi
- CPU edge tidak cukup kuat untuk model besar
- konsumsi daya meningkat

Karena itu, model sering perlu diubah menjadi format yang:

- lebih kecil
- lebih ringan
- lebih mudah dipanggil di perangkat
- lebih efisien untuk inferensi

Inilah yang dimaksud dengan **konversi model ringan**.

11.5.2 Bentuk penyederhanaan model

Untuk konteks buku ini, pendekatan yang realistis adalah:

- memilih model yang memang ringan dari awal
- mengurangi kompleksitas model
- mengekspor model ke format deployable
- memakai format inferensi yang lebih efisien seperti **TensorFlow Lite**

11.5.3 Strategi praktis yang disarankan

Untuk model tabular sederhana:

- pertimbangkan tetap memakai model klasik seperti logistic regression atau random forest kecil di Raspberry Pi
- gunakan Python langsung bila perangkat cukup kuat

Untuk model neural network ringan:

- latih di laptop
- konversi ke TensorFlow Lite
- jalankan di perangkat edge

11.5.4 Prinsip penting

Konversi model bukan tujuan akhir. Tujuan akhirnya adalah inferensi yang cukup cepat, cukup akurat, dan cukup stabil pada perangkat target.

11.6 ESP32 / Raspberry Pi sebagai target praktis

11.6.1 Mengapa dua platform ini dipilih?

Ini pilihan yang sangat baik karena keduanya mewakili dua dunia yang berbeda tetapi saling melengkapi.

11.6.2 ESP32

ESP32 cocok untuk:

- proyek murah
- akuisisi sensor
- konektivitas Wi-Fi
- inferensi yang sangat ringan
- TinyML sederhana

Kelebihan:

- murah
- populer
- banyak komunitas
- bagus untuk eksperimen mikrokontroler

Keterbatasan:

- RAM kecil
- storage kecil
- model harus sangat ringan
- debugging dan deployment lebih menantang daripada Raspberry Pi

11.6.3 Raspberry Pi

Raspberry Pi sangat cocok sebagai target praktis utama untuk buku ini.

Kelebihan:

- menjalankan Linux
- lebih mudah dipakai dengan Python
- lebih mudah integrasi dengan MQTT, SQLite, InfluxDB, Grafana, dan skrip lain
- lebih cocok untuk inferensi edge yang realistis
- sangat bagus sebagai jembatan antara laptop dan embedded device

Referensi edge computing juga menegaskan bahwa edge device dapat berupa perangkat dengan kemampuan komputasi dan storage lokal yang menjadi titik perantara antara sensor dan cloud. Dalam konteks buku praktis, Raspberry Pi sangat sesuai dengan peran ini.

11.6.4 Rekomendasi praktis untuk pembaca

Untuk buku ini, urutan belajar yang paling sehat adalah:

Langkah 1 — Jalankan model di laptop Ubuntu
Pastikan model dan pipeline benar.

Langkah 2 — Jalankan inferensi di Raspberry Pi
Ini target edge paling realistis untuk mayoritas pembaca.

Langkah 3 — Baru kenalkan konsep TinyML / ESP32
Gunakan model yang jauh lebih kecil dan contoh yang sederhana.

Dengan cara ini, pembaca tidak terlalu cepat tenggelam pada keterbatasan hardware sebelum memahami alur utamanya.

11.7 TensorFlow Lite secara praktis

11.7.1 Mengapa TensorFlow Lite?

Untuk konteks buku ini, **TensorFlow Lite** sangat masuk akal sebagai jembatan dari model Keras/TensorFlow di laptop ke model ringan di edge. Disini **TensorFlow Lite secara praktis** adalah fokus utama, bukan TinyML riset yang terlalu dalam.

TensorFlow sendiri juga disebut dalam referensi sebagai salah satu platform open source yang luas dipakai, mudah di deploy di berbagai platform, dan cocok untuk membangun neural network maupun model komputasional lain.

11.7.2 Apa itu TensorFlow Lite?

Secara praktis, TensorFlow Lite adalah format dan runtime yang memungkinkan model TensorFlow/Keras:

- dikonversi
- diperkecil
- dan dijalankan lebih efisien untuk inferensi

Untuk buku ini, pembaca tidak perlu dibebani terlalu dalam dengan seluruh ekosistem TensorFlow Lite Micro. Cukup pahami alur utamanya:

> **latih model → simpan model → konversi ke `.tflite` → jalankan inferensi di edge**

11.7.3 Kapan TensorFlow Lite cocok?

Gunakan jika:

- model dibuat dengan TensorFlow/Keras
- ingin inferensi lebih ringan
- ingin menjalankan model di Raspberry Pi atau perangkat edge lain
- ingin mulai masuk ke TinyML secara bertahap

11.7.4 Kapan tidak perlu dipaksakan?

Tidak perlu memaksakan TensorFlow Lite jika:

- model klasik Python di Raspberry Pi sudah cukup
- sistem tidak terlalu terbatas
- fokus pembelajaran masih di pipeline dan logika inferensi, bukan optimisasi deployment

Ini penting karena buku ini menekankan **realisme praktis**, bukan memaksa semua proyek harus menjadi TinyML murni.

11.8 Batasan memori, latensi, dan konsumsi daya

11.8.1 Memori

Perangkat edge kecil tidak punya RAM seperti laptop. Ini mempengaruhi:

- ukuran model
- ukuran batch inferensi
- jumlah fitur
- runtime yang bisa dipasang

11.8.2 Latensi

Pada IoT, inferensi yang akurat tetapi terlalu lambat bisa jadi kurang berguna. Beberapa sistem butuh respons:

- cepat
- lokal
- konsisten

Referensi edge computing menegaskan bahwa pengurangan delay adalah salah satu alasan utama komputasi dipindahkan ke edge.

11.8.3 Konsumsi daya

Ini sangat penting untuk perangkat kecil atau sistem yang tidak selalu mendapat suplai daya besar. Model yang terlalu berat bisa:

- memperbesar beban CPU
- meningkatkan suhu perangkat
- mempercepat pengosongan baterai
- membuat sistem tidak stabil

11.8.4 Pelajaran praktis

Model terbaik di edge bukan selalu model paling akurat, tetapi model yang paling seimbang antara akurasi, ukuran, kecepatan, dan kestabilan.

11.9 Contoh-contoh nyata yang bisa diimplementasikan

Smart room monitoring di Raspberry Pi

- data dari sensor atau MQTT masuk ke Pi
- model ringan memprediksi `nyaman / tidak_nyaman`
- hasil dikirim ke dashboard lokal

Predictive maintenance ringan di gateway

- Raspberry Pi menerima fitur getaran
- model anomaly detection memberi skor
- hanya alarm atau ringkasan yang diteruskan

TinyML sederhana di ESP32

- model klasifikasi kecil diekspor ke format ringan
- inferensi dilakukan langsung pada perangkat
- hasilnya hanya berupa label sederhana

Greenhouse edge inference

- model prediksi kelembaban tanah dijalankan di edge
- penyiraman awal bisa dipicu lokal
- cloud dipakai untuk histori dan pelatihan model berkala

11.10 Strategi implementasi yang realistis untuk pembaca

Agar pembaca tidak terjebak ke implementasi yang terlalu sulit di awal, tahapan yang disarankan untuk bab ini adalah:

Tahap 1 — Latih model di laptop Ubuntu

Gunakan:

- scikit-learn atau TensorFlow/Keras
- dataset lokal
- evaluasi metrik dengan tenang

Tahap 2 — Simulasikan inferensi edge di laptop

Jalankan model seperti seolah-olah perangkat hanya menerima satu data baru setiap saat.

Tahap 3 — Pindahkan inferensi ke Raspberry Pi

Gunakan Python atau TensorFlow Lite sesuai kebutuhan.

Tahap 4 — Eksplorasi TinyML ringan ke ESP32

Hanya setelah konsep inferensi edge benar-benar dipahami.

11.11 Langkah kerja yang harus dilakukan pembaca

Agar bab ini benar-benar operasional, pembaca dapat mengikuti tahapan berikut:

Tahap 1 — Tentukan model mana yang layak di deploy

Pilih model yang:

- cukup akurat
- cukup ringan
- stabil

Tahap 2 — Bedakan training dan inference

Putuskan:

- training di laptop/cloud
- inference di edge

Tahap 3 — Pilih target perangkat

Mulai dari:

- Raspberry Pi - lalu, jika ingin,
- ESP32

Tahap 4 — Siapkan format deployable

Gunakan:

- model Python langsung untuk Pi
- TensorFlow Lite untuk model neural network ringan

Tahap 5 — Uji inferensi lokal

Ukur:

- waktu inferensi
- penggunaan memori
- kestabilan hasil

Tahap 6 — Integrasikan ke pipeline IoT

Hubungkan inferensi dengan:

- MQTT
- storage
- dashboard
- alert

Tahap 7 — Tinjau trade-off

Bandingkan:

- akurasi
- latensi
- ukuran model
- kemudahan deployment

11.12 Sikap profesional dalam Edge AI dan TinyML

Ada satu pelajaran penting yang perlu ditekankan: **Edge AI bukan tentang memindahkan semua hal ke perangkat kecil, tetapi tentang menempatkan komputasi di lokasi yang paling masuk akal.**

Artinya:

- training tidak harus dipaksa di edge
- model besar tidak harus selalu dijalankan lokal
- tidak semua proyek butuh TinyML
- tidak semua perangkat cocok untuk model yang sama

Yang paling penting adalah membuat sistem yang:

- responsif
- cukup akurat
- cukup ringan
- stabil
- dan realistis untuk dipelihara

11.13 Penutup bab

Bab 11 mengajarkan pembaca bahwa kecerdasan pada sistem IoT tidak selalu harus tinggal di cloud. Di sini pembaca memahami bahwa:

- **inferensi di edge** memberi latensi lebih rendah dan sistem yang lebih mandiri,
- **training** dan **inferensi** adalah dua tahap yang sebaiknya dibedakan,
- **konversi model ringan** penting agar model benar-benar bisa dijalankan pada perangkat target,
- **ESP32** dan **Raspberry Pi** mewakili dua tingkat realisme deployment yang sangat berguna untuk praktikum,
- dan **TensorFlow Lite** adalah jembatan praktis dari model neural network di laptop menuju inferensi yang lebih ringan di edge.

Hands-on 27 - Ekspor Model Sederhana ke Format Ringan

1. Tujuan hands-on

Pada latihan ini, pembaca akan:

- melatih model sederhana di laptop Ubuntu,
- mengekspor model ke format yang lebih ringan atau lebih mudah dideploy,
- dan menyiapkan model agar siap dipakai pada inferensi lokal.

Inti latihan ini adalah memahami bahwa **training** dan **deployment** adalah dua tahap yang berbeda. Model yang nyaman dilatih di laptop belum tentu langsung ideal untuk edge. Karena itu, perlu ada proses pengemasan ulang agar model lebih praktis dipakai di perangkat target. **Konversi model ringan** dan **TensorFlow Lite secara praktis** adalah fokus utama bab ini.

2. Contoh implementasi nyata

Contoh yang realistis untuk tahap ini:

- model klasifikasi `normal / warning / fault` dari data ruangan,
- model prediksi sederhana suhu atau energi,
- model anomaly score ringan untuk edge gateway.

Use case seperti ini cocok karena:

- ukuran model relatif kecil,
- input biasanya berbentuk tabular,
- dan inferensi bisa dilakukan cepat di edge.

3. Strategi yang dipakai

Pada hands-on ini kita akan menunjukkan dua jalur yang realistis:

Jalur A — model scikit-learn

- latih model ringan
- simpan dengan `joblib`
- deploy di Raspberry Pi dengan Python

Jalur B — model Keras ringan

- latih model kecil
- konversi ke **TensorFlow Lite**
- jalankan inferensi memakai interpreter TFLite

Pendekatan ini sangat realistis untuk buku praktis, karena tidak semua kasus perlu memaksa model deep learning. Referensi juga menempatkan **scikit-learn**,

TensorFlow, dan **Keras** sebagai ekosistem open source yang relevan untuk pembelajaran ML.

4. Persiapan Ubuntu 26.04

Aktifkan lingkungan Python:

```
source .venv/bin/activate
pip install pandas numpy scikit-learn joblib tensorflow
mkdir -p models scripts data/processed
```

5. Ekspor model scikit-learn ringan

Kita pakai contoh model klasifikasi ruangan.

Buat file `scripts/export_sklearn_model.py`:

```
from pathlib import Path
import pandas as pd
import joblib
from sklearn.ensemble import RandomForestClassifier

DATA_DIR = Path("data/processed/classification_status")
MODEL_DIR = Path("models")
MODEL_DIR.mkdir(parents=True, exist_ok=True)

X_train = pd.read_csv(DATA_DIR / "X_train.csv")
y_train = pd.read_csv(DATA_DIR / "y_train.csv").squeeze()

model = RandomForestClassifier(
    n_estimators=100,
    max_depth=8,
    random_state=42,
    n_jobs=-1
)

model.fit(X_train, y_train)

model_path = MODEL_DIR / "room_status_rf.joblib"
joblib.dump(model, model_path)

print("Model scikit-learn berhasil disimpan ke:", model_path)
```

Jalankan:

```
python scripts/export_sklearn_model.py
```

Mengapa ini sudah “ringan”?

Untuk Raspberry Pi atau mini PC Linux, model scikit-learn kecil seperti Random Forest yang dibatasi kedalamannya sering sudah cukup praktis. Tidak semua deployment edge harus langsung memakai TensorFlow Lite. Ini sesuai dengan semangat bab: realistis, bukan memaksa semua hal menjadi TinyML.

6. Ekspor model Keras ke TensorFlow Lite

Sekarang contoh jalur neural network kecil.

Buat file `scripts/export_keras_to_tflite.py`:

```
from pathlib import Path
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

DATA_DIR = Path("data/processed/classification_room")
MODEL_DIR = Path("models")
MODEL_DIR.mkdir(parents=True, exist_ok=True)

X_train = pd.read_csv(DATA_DIR /
"X_train.csv").values.astype("float32")
y_train = pd.read_csv(DATA_DIR /
"y_train.csv").squeeze().values.astype("float32")

model = Sequential([
    Dense(16, activation="relu", input_shape=(X_train.shape[1],)),
    Dense(8, activation="relu"),
    Dense(1, activation="sigmoid")
])

model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["accuracy"])
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)

keras_path = MODEL_DIR / "room_binary.keras"
model.save(keras_path)

converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

tflite_path = MODEL_DIR / "room_binary.tflite"
with open(tflite_path, "wb") as f:
    f.write(tflite_model)

print("Model Keras disimpan ke:", keras_path)
print("Model TFLite disimpan ke:", tflite_path)
print("Ukuran file TFLite:", tflite_path.stat().st_size, "bytes")
```

Jalankan:

```
python scripts/export_keras_to_tflite.py
```

7. Apa yang harus dipahami pembaca?

Pada tahap ini pembaca belajar bahwa:

- **model training** bisa tetap dilakukan di laptop,
- model lalu disimpan ke format yang cocok untuk deployment,
- **joblib** cukup realistis untuk model klasik di Raspberry Pi,
- **TensorFlow Lite** adalah jalur praktis untuk model neural network ringan.

Ini konsisten dengan arsitektur edge modern, di mana edge dipakai untuk inferensi dan cloud/laptop dipakai untuk training dan eksperimen.

Hands-on 28 - Simulasi Inferensi Lokal di Edge

1. Tujuan hands-on

Pada latihan ini, pembaca akan mensimulasikan inferensi lokal seperti seolah-olah model berjalan di perangkat edge. Artinya:

- data sensor masuk satu per satu,
- model membaca data baru,
- model memberi prediksi lokal,
- hasil bisa dikirim ke terminal, file, MQTT, atau dashboard.

Ini sangat penting karena dalam sistem IoT nyata, yang paling sering terjadi bukan training, melainkan **inferensi berulang dan cepat**. Referensi edge computing menegaskan bahwa pemrosesan di edge ditujukan untuk layanan dengan kebutuhan **real-time** dan latency rendah.

2. Skenario yang akan dipakai

Kita akan mensimulasikan:

- data ruangan masuk satu demi satu,
- model klasifikasi ringan memprediksi `normal` atau `abnormal`,
- inferensi dilakukan lokal,
- hasil dicetak ke terminal.

3. Source code inferensi lokal untuk model scikit-learn

Buat file `scripts/simulate_edge_inference_sklearn.py`:

```
from pathlib import Path
import time
import pandas as pd
import joblib

MODEL_PATH = Path("models/room_status_rf.joblib")
DATA_PATH = Path("data/processed/classification_status/test.csv")

model = joblib.load(MODEL_PATH)
df = pd.read_csv(DATA_PATH)

feature_cols = [
    "temperature_c", "humidity_pct", "co2_ppm", "power_watt",
    "hour", "day_of_week", "is_weekend",
    "temp_lag_1", "co2_lag_1", "power_lag_1",
    "temp_ma_3", "co2_ma_3", "power_ma_3"
]

print("=== Simulasi inferensi lokal di edge ===")

for i, row in df.iterrows():
    x = row[feature_cols].to_frame().T
    pred = model.predict(x)[0]

    print(
        f"[{row['timestamp']}] "
        f"T={row['temperature_c']:.2f}C, "
        f"H={row['humidity_pct']:.2f}%, "
        f"CO2={row['co2_ppm']}, "
        f"P={row['power_watt']:.2f}W "
        f"-> status_pred={pred}"
    )

    time.sleep(0.2)  # simulasi aliran data
```

Jalankan:

```
python scripts/simulate_edge_inference_sklearn.py
```

4. Apa yang dipelajari?

Pembaca melihat bahwa inferensi edge pada dasarnya adalah:

- model dimuat sekali,
- data baru masuk,
- model memproses satu sampel,
- hasil keluar cepat.

Inilah pola yang sangat umum di gateway edge.

5. Simulasi inferensi lokal untuk TensorFlow Lite

Buat file `scripts/simulate_edge_inference_tflite.py`:

```
from pathlib import Path
import time
import numpy as np
import pandas as pd
import tensorflow as tf

MODEL_PATH = Path("models/room_binary.tflite")
DATA_PATH = Path("data/processed/classification_room/test.csv")

df = pd.read_csv(DATA_PATH)

feature_cols = [col for col in df.columns if col not in ["timestamp",
"device_id", "target_abnormal"]]

interpreter = tf.lite.Interpreter(model_path=str(MODEL_PATH))
interpreter.allocate_tensors()

input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

print("=== Simulasi inferensi TFLite lokal ===")

for _, row in df.iterrows():
    x = row[feature_cols].values.astype("float32").reshape(1, -1)

    interpreter.set_tensor(input_details[0]["index"], x)
    interpreter.invoke()
    output = interpreter.get_tensor(output_details[0]["index"])[0][0]

    pred_label = 1 if output >= 0.5 else 0

    print(
        f"[{row['timestamp']}] "
        f"score={output:.4f} -> pred_abnormal={pred_label}"
    )

    time.sleep(0.2)
```

Jalankan:

```
python scripts/simulate_edge_inference_tflite.py
```

6. Hal penting yang harus dicatat

Dari latihan ini pembaca akan melihat perbedaan penting:

scikit-learn

- lebih mudah untuk tabular model
- sangat cocok di Raspberry Pi
- workflow lebih sederhana

TensorFlow Lite

- lebih cocok untuk model Keras yang ingin diringankan
- jalur deployment lebih mirip TinyML / mobile / embedded inference
- setup sedikit lebih teknis

7. Mengapa ini disebut “simulasi edge”?

Karena walau masih berjalan di laptop Ubuntu, pola inferensinya sudah meniru edge:

- input datang satu per satu
- model tidak dilatih ulang
- prediksi dihasilkan lokal
- respons cepat

Ini adalah langkah yang sangat sehat sebelum benar-benar memindahkan inferensi ke Raspberry Pi.

Hands-on 29 - Mini Proyek Klasifikasi Sederhana di Raspberry Pi / Emulator

1. Tujuan hands-on

Pada latihan ini, pembaca akan menyusun **mini proyek end-to-end** yang meniru deployment nyata di Raspberry Pi atau emulator Linux sejenis. Tujuannya:

- membaca data sensor atau simulasi sensor,
- menjalankan model lokal,
- menghasilkan status klasifikasi,
- dan, bila diinginkan, meneruskan hasil ke MQTT atau log file.

Latihan ini sangat penting karena menyatukan semua konsep:

- training di laptop
- ekspor model
- inferensi lokal
- integrasi edge ke pipeline IoT

Ini juga sangat sejalan dengan arsitektur edge yang dijelaskan dalam referensi, di mana edge node:

- menerima data dari device,
- menjalankan komputasi lokal,
- lalu mengirim hasil atau ringkasan ke lapisan lain.

2. Target perangkat

Raspberry Pi

Untuk buku ini, Raspberry Pi adalah target paling realistis karena:

- menjalankan Linux
- bisa memakai Python
- mudah integrasi dengan MQTT dan file lokal
- cocok sebagai edge gateway praktis

Emulator

Jika pembaca belum punya Raspberry Pi, maka:

- laptop Ubuntu
- VM Linux
- mini PC
- atau container lokal
  dapat dipakai sebagai **emulator logika edge**

Pendekatan ini sangat realistis dan sesuai semangat buku yang fokus pada praktik cepat, bukan menunggu perangkat ideal.

3. Struktur mini proyek

Kita buat alur sederhana:

simulator sensor → model lokal → hasil status → log / MQTT

4. Source code mini proyek tanpa MQTT

Buat file `scripts/edge_room_classifier_project.py`:

```
from pathlib import Path
import time
import random
import joblib
import pandas as pd

MODEL_PATH = Path("models/room_status_rf.joblib")
OUTPUT_LOG = Path("outputs/reports/edge_room_predictions.csv")
OUTPUT_LOG.parent.mkdir(parents=True, exist_ok=True)

model = joblib.load(MODEL_PATH)

records = []

def build_feature_row(temp, hum, co2, power, hour, dow):
    # contoh fitur sederhana; pada implementasi nyata
    # beberapa lag/rolling sebaiknya dipertahankan dari buffer lokal
    row = {
        "temperature_c": temp,
        "humidity_pct": hum,
        "co2_ppm": co2,
        "power_watt": power,
        "hour": hour,
        "day_of_week": dow,
        "is_weekend": int(dow >= 5),
        "temp_lag_1": temp,
        "co2_lag_1": co2,
        "power_lag_1": power,
        "temp_ma_3": temp,
        "co2_ma_3": co2,
        "power_ma_3": power
    }
    return pd.DataFrame([row])

print("=== Mini proyek edge room classifier dimulai ===")

for step in range(30):
    temp = round(random.uniform(24.0, 33.0), 2)
    hum = round(random.uniform(40.0, 70.0), 2)
    co2 = random.randint(450, 1500)
    power = round(random.uniform(100.0, 300.0), 2)

    hour = (8 + step // 3) % 24
```

```
    dow = 2  # misal Rabu

    X = build_feature_row(temp, hum, co2, power, hour, dow)
    pred = model.predict(X)[0]

    print(
        f"step={step} | T={temp} H={hum} CO2={co2} P={power} ->
status={pred}"
    )

    records.append({
        "step": step,
        "temperature_c": temp,
        "humidity_pct": hum,
        "co2_ppm": co2,
        "power_watt": power,
        "predicted_status": pred
    })

    time.sleep(0.3)

pd.DataFrame(records).to_csv(OUTPUT_LOG, index=False)
print("Log hasil inferensi disimpan ke:", OUTPUT_LOG)
```

Jalankan:

```
python scripts/edge_room_classifier_project.py
```

5. Versi mini proyek dengan MQTT opsional

Jika pembaca ingin alur yang lebih “IoT banget”, kita tambahkan MQTT.

Buat file `scripts/edge_room_classifier_mqtt.py`:

```
from pathlib import Path
import json
import time
import random
import joblib
import pandas as pd
import paho.mqtt.client as mqtt

MODEL_PATH = Path("models/room_status_rf.joblib")
TOPIC = "iotlab/room01/prediction"

model = joblib.load(MODEL_PATH)

client = mqtt.Client(client_id="edge-room-classifier")
client.connect("localhost", 1883, 60)

def build_feature_row(temp, hum, co2, power, hour, dow):
    row = {
        "temperature_c": temp,
        "humidity_pct": hum,
        "co2_ppm": co2,
        "power_watt": power,
        "hour": hour,
```

```
            "day_of_week": dow,
            "is_weekend": int(dow >= 5),
            "temp_lag_1": temp,
            "co2_lag_1": co2,
            "power_lag_1": power,
            "temp_ma_3": temp,
            "co2_ma_3": co2,
            "power_ma_3": power
        }
        return pd.DataFrame([row])

    for step in range(20):
        temp = round(random.uniform(24.0, 33.0), 2)
        hum = round(random.uniform(40.0, 70.0), 2)
        co2 = random.randint(450, 1500)
        power = round(random.uniform(100.0, 300.0), 2)

        hour = (8 + step // 3) % 24
        dow = 2

        X = build_feature_row(temp, hum, co2, power, hour, dow)
        pred = model.predict(X)[0]

        payload = {
            "step": step,
            "temperature_c": temp,
            "humidity_pct": hum,
            "co2_ppm": co2,
            "power_watt": power,
            "predicted_status": pred
        }

        client.publish(TOPIC, json.dumps(payload), qos=1)
        print("Published:", payload)

        time.sleep(0.5)

    client.disconnect()
```

Uji subscriber cepat

```
mosquitto_sub -h localhost -t "iotlab/room01/prediction" -v
```

Dengan ini, mini proyek edge sudah benar-benar menyentuh pola operasional IoT:

- akuisisi data
- inferensi lokal
- pengiriman hasil

6. Apa yang membuat proyek ini “realistis”?

Karena alurnya sama dengan banyak sistem edge nyata:

- model dilatih di tempat lain
- edge hanya menerima model final
- edge menerima data sensor
- edge membuat keputusan lokal
- hasil disimpan atau dikirim

Ini konsisten dengan pembagian peran antara cloud/laptop dan edge yang dijelaskan dalam referensi edge computing.

7. Contoh pengembangan lanjutan

Dari mini proyek ini, pembaca bisa berkembang ke:

- threshold + model hybrid
- logging ke SQLite lokal
- kirim hasil ke dashboard Grafana
- retraining periodik di laptop lalu redeploy ke edge
- integrasi Raspberry Pi dengan sensor nyata
- percobaan TinyML yang jauh lebih kecil di ESP32

8. Langkah kerja yang harus dilakukan pembaca

Tahap 1

Latih model di laptop Ubuntu.

Tahap 2

Ekspor model:

- `joblib` untuk model klasik
- `.tflite` untuk model Keras ringan

Tahap 3

Uji inferensi lokal satu per satu.

Tahap 4

Simulasikan aliran data seolah-olah dari sensor.

Tahap 5

Jalankan mini proyek di Raspberry Pi atau emulator Linux.

Tahap 6

Tambahkan output:

- terminal
- CSV
- MQTT
- atau dashboard

Tahap 7

Evaluasi apakah model:

- cukup cepat
- cukup stabil
- cukup ringan
- dan layak untuk edge

9. Penutup

Hands-on 27–29 ini membawa pembaca dari konsep **Edge AI** menuju implementasi yang benar-benar terasa nyata. Pada Hands-on 27, pembaca belajar mengekspor model ke format yang lebih praktis untuk deployment. Pada Hands-on 28, pembaca mensimulasikan inferensi lokal seperti yang akan terjadi di edge. Pada Hands-on 29, pembaca menyusun mini proyek klasifikasi sederhana yang bisa dijalankan di **Raspberry Pi** atau **emulator Linux**, dan bahkan bisa dihubungkan ke **MQTT**. Inilah bentuk yang sangat konkret dari filosofi buku ini: **dari sensor ke keputusan cerdas**, tetapi dilakukan dengan cara yang **realistis, bertahap, dan open source**.
Pendekatan ini juga sangat konsisten dengan referensi edge computing yang menekankan komputasi dekat sumber data, latensi rendah, pengurangan beban jaringan, dan pembagian kerja yang sehat antara **device–edge–cloud**.

Bagian IV — Integrasi End-to-End Open Source (Bab 12–13)

Bab 12. Pipeline IoT-ML End-to-End

12.1 Mengapa pipeline end-to-end penting?

Banyak pemula belajar IoT, data analytics, dan machine learning secara terpisah:

- sensor berdiri sendiri,
- MQTT berdiri sendiri,
- preprocessing di notebook terpisah,
- model dilatih terpisah,
- dashboard dibuat terpisah.

Masalahnya, dunia nyata tidak berjalan dalam potongan-potongan seperti itu. Di lapangan, kita butuh **pipeline end-to-end** yang benar-benar menyambungkan semuanya:

- data masuk,
- dibaca,
- dibersihkan,
- diproses,
- diberi prediksi,
- disimpan,
- dan hasilnya bisa dilihat atau memicu aksi.

Itulah mengapa bab ini sangat penting. Ini adalah bab yang membuat pembaca berhenti melihat IoT-ML sebagai kumpulan teknik lepas, lalu mulai melihatnya sebagai **sistem kerja yang utuh**.

Secara praktis, pipeline end-to-end membantu pembaca menjawab pertanyaan berikut:

- bagaimana data sensor masuk ke sistem?
- siapa yang membersihkan dan memvalidasi data?
- kapan model dijalankan?
- ke mana hasil prediksi disimpan?
- bagaimana dashboard membaca hasil?
- kapan model diperbarui?
- bagaimana sistem tetap sederhana tetapi berguna?

12.2 Posisi Bab 12 dalam alur buku

Sebelum sampai ke bab ini, pembaca sudah mempelajari:

- arsitektur IoT open source
- MQTT dan subscriber Python
- penyimpanan ke CSV/SQLite/InfluxDB
- pembersihan dan eksplorasi data
- *feature engineering*
- visualisasi dan dashboard
- supervised learning
- anomaly detection
- forecasting
- inferensi di edge dan model ringan.

Maka Bab 12 adalah tahap integrasi. Di sinilah semua komponen itu disatukan menjadi satu proyek operasional yang bisa dijalankan berulang.

12.3 Arsitektur proyek dari sensor ke prediksi

Untuk buku ini, arsitektur yang paling sehat adalah arsitektur yang:

- cukup realistis untuk dunia nyata,
- cukup sederhana untuk dipahami mahasiswa dan praktisi,
- sepenuhnya open source,
- dan bisa dijalankan di Ubuntu 26.04, mini PC, atau Raspberry Pi.

12.3.1 Arsitektur minimal yang disarankan

Arsitektur proyek end-to-end yang sangat cocok untuk buku ini adalah:

> **sensor / simulator → MQTT broker → Python subscriber → preprocessing → model inferensi → storage → dashboard / alert**

Hands-on 30 menyebut pipeline lengkap:

- sensor/simulator
- MQTT
- subscriber Python
- preprocessing
- prediksi
- penyimpanan hasil
- dashboard.

12.3.2 Peran setiap komponen

Sensor / simulator
Menghasilkan data mentah. Dalam buku ini bisa berupa:

- sensor nyata,
- ESP32,
- simulator Python,
- atau CSV publisher.

MQTT broker
Menjadi titik distribusi pesan. MQTT sangat cocok di sini karena bersifat *publish/subscribe*, ringan, dan dirancang untuk device dengan keterbatasan resource. Broker memisahkan pengirim data dan penerima data sehingga pipeline lebih modular.

Python subscriber
Menjadi jantung pipeline praktis. Ia:

- menerima pesan dari broker,
- melakukan parsing JSON,
- validasi dasar,
- preprocessing,
- memanggil model,
- dan menulis hasil ke storage.

Storage
Tempat menyimpan:

- data mentah,
- data hasil preprocessing,
- skor model,
- hasil inferensi,
- histori alarm.

Dashboard / alert
Lapisan yang membuat sistem berguna secara operasional:

- visualisasi data mentah,
- visualisasi hasil model,
- notifikasi bila skor anomali tinggi,
- rekap operasional.

12.3.3 Mengapa arsitektur ini kuat?

Karena:

- mudah diuji per komponen,
- mudah diperluas,
- dapat dipakai untuk smart room, predictive maintenance, dan energy analytics,
- dan sangat cocok untuk pembelajaran bertahap.

Referensi edge computing juga mendukung pola seperti ini: data dikumpulkan di device layer, diproses dekat edge jika perlu, lalu diteruskan atau diringkas ke layer lain untuk visualisasi, histori, atau pengambilan keputusan.

12.4 Ingest data dari MQTT ke Python

12.4.1 Mengapa MQTT → Python menjadi tulang punggung pipeline?

Karena untuk buku praktis ini, Python adalah alat yang paling fleksibel untuk:

- membaca data sensor,
- membersihkan data,
- memanggil model,
- menyimpan ke database,
- mengirim alert,
- dan menjadi jembatan antara IoT dan ML.

Di sisi lain, MQTT memberi cara yang sangat ringan dan modular untuk mengalirkan data sensor. Referensi menegaskan bahwa MQTT:

- memakai model *publish/subscribe*,
- cocok untuk device dengan keterbatasan resource,
- ringan dan mudah diimplementasikan,
- dan broker bertugas mengorkestrasi distribusi pesan antar-klien.

12.4.2 Alur ingest yang sehat

Urutan yang disarankan:

1. subscriber Python terhubung ke broker
2. pesan sensor diterima dari topik tertentu
3. payload JSON diparsing
4. validasi dasar dilakukan
5. data dibentuk menjadi record standar
6. record diteruskan ke preprocessing dan model

12.4.3 Validasi dasar saat ingest

Jangan langsung percaya semua payload. Lakukan minimal:

- cek `timestamp`
- cek `device_id`
- cek field sensor penting
- cek tipe data
- cek rentang wajar

Ini penting karena heterogenitas dan kualitas data memang menjadi tantangan nyata pada data IoT. Referensi Mohanty menegaskan bahwa data IoT yang heterogen perlu melalui **cleansing, ordering, and preprocessing**, karena kombinasi data yang tidak tertata bisa menghasilkan prediksi palsu.

12.4.4 Struktur payload yang disarankan

Contoh payload yang cukup bersih:

```
{
  "timestamp": "2026-01-01T10:10:00",
  "device_id": "room01",
  "temperature_c": 28.6,
  "humidity_pct": 54.1,
  "co2_ppm": 820,
  "power_watt": 145.3
}
```

Struktur yang rapi seperti ini akan sangat memudahkan seluruh pipeline berikutnya.

12.5 Penyimpanan data

12.5.1 Mengapa storage tidak boleh dianggap sekadar arsip?

Dalam pipeline IoT-ML, storage punya beberapa fungsi sekaligus:

- menyimpan histori mentah,
- menyimpan hasil preprocessing,
- menyimpan hasil inferensi,
- menjadi sumber dashboard,
- menjadi bahan retraining,
- dan menjadi jejak audit sederhana.

Karena itu, storage sebaiknya dipikirkan sebagai bagian aktif dari sistem.

12.5.2 Penyimpanan yang realistis untuk buku ini

Berdasarkan stack open source yang sudah dipakai di bab-bab sebelumnya, pilihan yang paling realistis adalah:

- **CSV** untuk debugging atau log sederhana
- **SQLite** untuk sistem lokal ringan
- **InfluxDB** untuk *time series* dan dashboard Grafana

InfluxDB / SQLite ditetapkan sebagai storage praktis sejak bab arsitektur awal.

12.5.3 Data apa saja yang sebaiknya disimpan?

Agar pipeline sehat, idealnya ada minimal tiga lapisan data:

1. Data mentah

- payload asli dari sensor
- timestamp asli
- status parsing

2. Data olahan

- hasil cleaning ringan
- fitur yang dipakai model
- *rolling stats*, lag, atau skor lain bila perlu

3. Hasil inferensi

- label prediksi
- skor anomali
- confidence sederhana
- timestamp inferensi
- versi model

12.5.4 Praktik minimum yang disarankan

Bila ingin tetap sederhana:

- simpan data mentah dan hasil inferensi saja di SQLite atau InfluxDB
- simpan log debug ke CSV
- jangan terlalu dini membuat storage terlalu kompleks

12.6 Batch retraining sederhana

12.6.1 Mengapa retraining perlu dibahas?

Banyak sistem demo berhenti setelah model pertama berhasil dijalankan. Padahal dalam sistem nyata:

- distribusi data bisa berubah,
- perangkat bisa menua,
- kebiasaan penggunaan berubah,
- ruangan berganti pola,
- mesin makin aus,
- beban energi berubah.

Maka model yang awalnya bagus bisa menurun performanya. **Drift data dan penurunan performa model** juga ditetapkan sebagai topik penting di bab operasional berikutnya.

12.6.2 Apa itu batch retraining sederhana?

Untuk konteks buku ini, **batch retraining sederhana** berarti:

- kumpulkan data baru selama periode tertentu,
- gabungkan dengan data historis yang relevan,
- latih ulang model secara berkala,
- simpan model versi baru,
- lalu gunakan model baru itu untuk inferensi berikutnya.

12.6.3 Kapan re-training dilakukan?

Contoh ritme yang realistis:

- harian untuk eksperimen cepat
- mingguan untuk smart room
- mingguan / bulanan untuk energi
- tergantung volume data dan perubahan pola

12.6.4 Mengapa disebut “batch”?

Karena pembaruan model tidak dilakukan setiap kali satu data masuk. Data dikumpulkan dulu, lalu dilatih ulang sekaligus. Ini jauh lebih realistis untuk buku praktis dibanding *online learning* penuh yang lebih kompleks.

12.6.5 Strategi yang disarankan

Untuk buku ini, strategi sederhana yang masuk akal:

- simpan data hasil inferensi ke storage
- setiap periode tertentu, jalankan skrip retraining

- evaluasi model baru pada holdout terbaru
- jika layak, simpan sebagai model aktif

Dengan ini, pembaca belajar bahwa pipeline end-to-end tidak berhenti pada inferensi, tetapi juga punya siklus penyegaran model.

12.7 Inferensi otomatis

12.7.1 Apa maksud inferensi otomatis?

Inferensi otomatis berarti begitu data sensor masuk, sistem:

- memproses data,
- membentuk fitur,
- memanggil model,
- menghasilkan prediksi,
- lalu menyimpan atau meneruskan hasil
  tanpa campur tangan manual.

Inilah titik di mana pipeline benar-benar terasa “hidup”.

12.7.2 Alur inferensi otomatis yang sehat

1. data baru diterima dari MQTT
2. payload divalidasi
3. fitur dibentuk
4. model aktif dimuat atau sudah berada di memori
5. prediksi dilakukan
6. hasil ditulis ke storage
7. hasil dapat dikirim ke dashboard/alert

12.7.3 Hal yang perlu diperhatikan

Agar inferensi otomatis stabil:

- model jangan dibuka-tutup terus setiap record jika tidak perlu
- fitur harus konsisten dengan saat training
- penamaan kolom harus stabil
- kalau ada data kurang, tentukan strategi fallback
- log error harus disimpan

12.7.4 Inferensi otomatis untuk use case buku ini

Smart room

- prediksi `nyaman / tidak_nyaman`
- atau skor anomali ruangan

Predictive maintenance

- skor anomali getaran
- atau klasifikasi `normal / warning / fault`

Smart energy

- prediksi beban berikutnya
- atau alert beban tidak normal

12.8 Dashboard / alert

12.8.1 Mengapa hasil model harus diteruskan ke dashboard?

Karena model tanpa output yang dapat dibaca operator akan berhenti menjadi eksperimen laboratorium. Agar berguna, hasil inferensi harus:

- terlihat,
- dapat ditinjau,
- bisa memicu aksi,
- dan tercatat.

Itulah mengapa dashboard dan alert adalah bagian inti dari pipeline end-to-end.

12.8.2 Dashboard yang disarankan

Dengan stack buku ini, **Grafana** tetap menjadi pilihan utama untuk dashboard. Data dari InfluxDB atau storage lain dapat dipakai untuk menampilkan:

- data sensor mentah
- hasil klasifikasi
- skor anomali
- histori prediksi
- kondisi terakhir perangkat

12.8.3 Alert sederhana yang realistis

Untuk buku praktis, alert tidak perlu terlalu rumit. Cukup misalnya:

- kirim log ke topik MQTT `alert`
- tulis record khusus ke tabel alert
- tampilkan panel merah di Grafana
- cetak warning di terminal
- kirim notifikasi sederhana ke Telegram/WA jika pembaca ingin mengembangkan lebih lanjut

12.8.4 Kapan alert dipicu?

Contoh:

- skor anomali > ambang
- status prediksi = `fault`
- suhu > batas aman
- prediksi energi > threshold operasional
- inferensi gagal berulang kali

12.8.5 Prinsip penting

Alert harus berguna, bukan berisik.
Kalau terlalu sering salah, operator akan berhenti percaya.

12.9 Docker opsional

12.9.1 Mengapa Docker dibahas sebagai opsional?

Karena Docker sangat berguna untuk:

- memudahkan setup ulang
- menjaga versi layanan konsisten
- memindahkan sistem antar mesin
- membuat eksperimen lebih repeatable

Namun, untuk pembaca pemula, Docker tidak harus menjadi syarat wajib sejak awal. Docker ditempatkan sebagai **opsional untuk reproducibility**, bukan utama.

12.9.2 Kapan Docker sangat membantu?

Gunakan Docker jika:

- ingin menjalankan Mosquitto, InfluxDB, dan Grafana cepat
- ingin setup yang mudah diulang
- ingin berbagi proyek ke teman atau mahasiswa lain
- ingin menghindari konflik instalasi lokal

12.9.3 Kapan belum perlu?

Belum perlu jika:

- pembaca masih fokus memahami logika pipeline
- semua komponen sudah jalan stabil secara lokal
- targetnya pembelajaran Python dan MQTT dulu

12.9.4 Sikap yang sehat

Pahami alur sistem dulu, baru bungkus dengan Docker.
Jangan sampai container membuat pembaca hafal `docker compose up` tetapi belum paham data mengalir ke mana.

12.10 Contoh arsitektur end-to-end yang bisa diimplementasikan

12.10.1 Smart Room Monitoring

Alur:

- simulator atau sensor mengirim suhu, kelembaban, CO_2
- Mosquitto menerima data
- subscriber Python melakukan preprocessing
- model klasifikasi atau anomaly detection dijalankan
- hasil disimpan ke InfluxDB
- Grafana menampilkan ruangan normal/warning/fault
- jika perlu, alert dikirim ke topik khusus

12.10.2 Predictive Maintenance Sederhana

Alur:

- data arus/getaran masuk
- Python membentuk fitur
- model anomaly detection memberi skor
- hasil disimpan
- dashboard menampilkan histori anomali
- alert muncul jika skor tinggi

12.10.3 Smart Energy Analytics

Alur:

- data daya/arus dikirim periodik
- subscriber Python menulis data ke storage
- model forecasting memprediksi beban berikutnya
- dashboard menampilkan aktual vs prediksi
- alert muncul jika beban tidak normal

12.11 Struktur folder proyek yang disarankan

Agar pembaca bisa cepat produktif, berikut struktur folder yang sangat masuk akal:

```
iot-ml-project/
├── data/
│   ├── raw/
│   ├── processed/
│   └── models/
├── scripts/
│   ├── publisher/
│   ├── subscriber/
│   ├── preprocessing/
│   ├── inference/
│   └── retraining/
├── dashboards/
├── logs/
├── docker/
├── requirements.txt
└── README.md
```

Kenapa struktur ini baik?

Karena:

- data mentah dan olahan dipisah
- model tidak bercampur dengan skrip
- komponen pipeline jelas
- memudahkan pembaca berkembang ke proyek akhir buku

12.12 Langkah kerja yang harus dilakukan pembaca

Agar bab ini benar-benar operasional, pembaca dapat mengikuti tahapan berikut.

Tahap 1 — Tentukan use case

Pilih salah satu:

- smart room
- predictive maintenance
- smart energy

Tahap 2 — Tentukan arsitektur minimal

Gunakan:

- sensor/simulator
- MQTT broker
- Python subscriber
- storage
- dashboard

Tahap 3 — Implementasikan ingest dari MQTT

Pastikan data:

- terbaca
- tervalidasi
- konsisten formatnya

Tahap 4 — Tambahkan preprocessing

Lakukan:

- parsing
- pemilihan fitur
- imputasi ringan bila perlu
- normalisasi jika model butuh

Tahap 5 — Jalankan inferensi otomatis

Gunakan model yang sudah dipilih dari bab sebelumnya.

Tahap 6 — Simpan hasil

Tulis:

- data mentah
- hasil inferensi
- log error

Tahap 7 — Hubungkan ke dashboard / alert

Buat panel untuk:

- data sensor
- skor model
- status prediksi
- alarm

Tahap 8 — Tambahkan batch retraining sederhana

Jadwalkan re-training berkala dari data historis baru.

Tahap 9 — Bungkus dengan Docker bila perlu

Lakukan ini setelah logika sistem benar-benar dipahami.

12.13 Sikap profesional dalam membangun pipeline end-to-end

Ada satu pelajaran penting yang perlu ditekankan: **pipeline yang baik bukan yang paling rumit, tetapi yang paling stabil dan dapat dipelihara**.

Dalam praktik IoT-ML:

- sistem yang terlalu banyak komponen bisa rapuh,
- model yang bagus tetapi pipeline rapuh tetap tidak berguna,
- dashboard yang indah tetapi data tidak valid akan menyesatkan,
- alert yang terlalu sering salah akan diabaikan.

Karena itu, untuk tahap ini:

- lebih baik pipeline kecil tetapi utuh,
- daripada banyak komponen tetapi tidak selesai terhubung.

12.14 Penutup bab

Bab 12 adalah titik ketika seluruh isi buku mulai menyatu menjadi sistem operasional yang benar-benar hidup. Di sini pembaca belajar bahwa pipeline IoT-ML end-to-end berarti menyusun alur lengkap dari **sensor ke prediksi**: arsitektur proyek yang jelas, ingest dari MQTT ke Python, penyimpanan yang rapi, inferensi otomatis, dashboard/alert yang berguna, dan retraining sederhana agar model tidak membeku di masa lalu. Docker kemudian hadir sebagai opsi untuk membuat semuanya lebih mudah diulang, tetapi bukan pengganti pemahaman sistem.

Hands-on 30 - Pipeline Lengkap: Sensor/Simulator → MQTT → Python → Preprocessing → Prediksi → Storage → Dashboard

1. Tujuan hands-on

Pada latihan ini, pembaca akan membangun satu **pipeline IoT-ML lengkap** yang benar-benar bekerja dari ujung ke ujung. Setelah selesai, pembaca diharapkan mampu:

- mensimulasikan sensor yang mengirim data,
- menerima data melalui MQTT,
- melakukan *preprocessing* ringan di Python,
- menjalankan model untuk inferensi otomatis,
- menyimpan data mentah dan hasil prediksi,
- dan menampilkan hasil pada dashboard.

Ini adalah latihan paling penting di bagian integrasi, karena disinilah semua komponen yang telah dipelajari sebelumnya benar-benar menyatu menjadi satu sistem kerja.

2. Skenario use case yang dipakai

Agar konkret dan realistis, kita gunakan **Smart Room Monitoring** sebagai studi kasus mini.

Data yang dikirim sensor/simulator

- `temperature_c`
- `humidity_pct`
- `co2_ppm`
- `power_watt`

Target inferensi

- status ruangan:
  - `normal`
  - `warning`
  - `fault`

Alur sistem

1. simulator mengirim data ke **Mosquitto MQTT**
2. subscriber Python membaca pesan
3. subscriber melakukan validasi dan *preprocessing*
4. model klasifikasi dijalankan
5. hasil disimpan ke **InfluxDB**

6. Grafana membaca hasil untuk dashboard

3. Arsitektur minimal yang dipakai

Arsitektur yang akan kita bangun:

> **simulator sensor → Mosquitto MQTT → Python subscriber → preprocessing → model → InfluxDB → Grafana**

Kenapa ini kuat?

- cukup realistis untuk dunia nyata,
- cukup ringan untuk laptop Ubuntu atau Raspberry Pi,
- seluruh stack *open source*,
- dan sangat mudah dikembangkan.

4. Persiapan layanan di Ubuntu 26.04

4.1 Pasang Mosquitto

```
sudo apt update
sudo apt install -y mosquitto mosquitto-clients
sudo systemctl enable mosquitto
sudo systemctl start mosquitto
sudo systemctl status mosquitto
```

4.2 Jalankan InfluxDB dengan Docker

```
docker run -d \
  --name influxdb2 \
  -p 8086:8086 \
  -v ~/iot-ml-book/influxdb-data:/var/lib/influxdb2 \
  influxdb:2
```

Lalu buka:

```
http://localhost:8086
```

Buat:

- organization: `iotlab`
- bucket: `sensor_data`
- simpan token

4.3 Jalankan Grafana

```
docker run -d \
  --name grafana \
  -p 3000:3000 \
  grafana/grafana-oss
```

Buka:

```
http://localhost:3000
```

4.4 Pasang library Python

```
source .venv/bin/activate
pip install pandas paho-mqtt influxdb-client scikit-learn joblib
mkdir -p scripts data/raw data/processed models outputs/reports
```

5. Menyiapkan model yang akan dipakai

Untuk hands-on ini, kita asumsikan model klasifikasi ruangan sudah ada, misalnya model scikit-learn dari bab sebelumnya:

```
models/room_status_rf.joblib
```

Model ini memprediksi status:

- `normal`
- `warning`
- `fault`

Bila model belum tersedia, gunakan model yang sudah dilatih dari dataset `classification_status`.

6. Simulator sensor: kirim data ke MQTT

Buat file `scripts/publisher_room_simulator.py`:

```
import json
import random
import time
from datetime import datetime
import paho.mqtt.client as mqtt

BROKER_HOST = "localhost"
BROKER_PORT = 1883
TOPIC = "iotlab/room01/telemetry"

client = mqtt.Client(client_id="publisher-room-sim")
client.connect(BROKER_HOST, BROKER_PORT, keepalive=60)

while True:
    payload = {
        "timestamp": datetime.now().isoformat(timespec="seconds"),
        "device_id": "room01",
        "temperature_c": round(random.uniform(24.0, 33.5), 2),
        "humidity_pct": round(random.uniform(40.0, 70.0), 2),
        "co2_ppm": random.randint(450, 1500),
        "power_watt": round(random.uniform(90.0, 300.0), 2)
    }

    payload_json = json.dumps(payload)
    result = client.publish(TOPIC, payload_json, qos=1)

    if result.rc == 0:
        print("Published:", payload_json)
    else:
        print("Publish gagal.")

    time.sleep(2)
```

Jalankan:

```
python scripts/publisher_room_simulator.py
```

7. Subscriber Python: ingest, preprocessing, prediksi, storage

Ini adalah jantung pipeline.

Buat file `scripts/pipeline_room_subscriber.py`:

```
import json
from collections import deque
from pathlib import Path

import joblib
import pandas as pd
import paho.mqtt.client as mqtt
from influxdb_client import InfluxDBClient, Point, WritePrecision
from influxdb_client.client.write_api import SYNCHRONOUS

# MQTT
BROKER_HOST = "localhost"
BROKER_PORT = 1883
TOPIC = "iotlab/room01/telemetry"

# Model
MODEL_PATH = Path("models/room_status_rf.joblib")

# InfluxDB
INFLUX_URL = "http://localhost:8086"
INFLUX_TOKEN = "GANTI_DENGAN_TOKEN_ANDA"
INFLUX_ORG = "iotlab"
INFLUX_BUCKET = "sensor_data"

# Buffer sederhana untuk lag / moving average
temp_buffer = deque(maxlen=3)
co2_buffer = deque(maxlen=3)
power_buffer = deque(maxlen=3)

model = joblib.load(MODEL_PATH)

influx_client = InfluxDBClient(url=INFLUX_URL, token=INFLUX_TOKEN,
org=INFLUX_ORG)
write_api = influx_client.write_api(write_options=SYNCHRONOUS)

def safe_mean(values):
    return sum(values) / len(values) if len(values) > 0 else None

def build_features(data):
    temp = float(data["temperature_c"])
    hum = float(data["humidity_pct"])
    co2 = float(data["co2_ppm"])
    power = float(data["power_watt"])

    # update buffer
    temp_buffer.append(temp)
    co2_buffer.append(co2)
    power_buffer.append(power)

    feature_row = {
        "temperature_c": temp,
        "humidity_pct": hum,
```

```
        "co2_ppm": co2,
        "power_watt": power,
        "hour": pd.to_datetime(data["timestamp"]).hour,
        "day_of_week": pd.to_datetime(data["timestamp"]).dayofweek,
        "is_weekend": int(pd.to_datetime(data["timestamp"]).dayofweek
>= 5),
        "temp_lag_1": temp_buffer[-2] if len(temp_buffer) >= 2 else
temp,
        "co2_lag_1": co2_buffer[-2] if len(co2_buffer) >= 2 else co2,
        "power_lag_1": power_buffer[-2] if len(power_buffer) >= 2 else
power,
        "temp_ma_3": safe_mean(temp_buffer),
        "co2_ma_3": safe_mean(co2_buffer),
        "power_ma_3": safe_mean(power_buffer),
    }
    return pd.DataFrame([feature_row])

def on_connect(client, userdata, flags, reason_code, properties=None):
    print("Subscriber terhubung ke broker.")
    client.subscribe(TOPIC, qos=1)
    print(f"Subscribe ke: {TOPIC}")

def on_message(client, userdata, msg):
    try:
        payload = json.loads(msg.payload.decode())

        # Validasi dasar
        required_fields = [
            "timestamp", "device_id", "temperature_c",
            "humidity_pct", "co2_ppm", "power_watt"
        ]
        for field in required_fields:
            if field not in payload:
                print(f"Field hilang: {field}")
                return

        # Preprocessing + fitur
        X = build_features(payload)

        # Prediksi
        pred_status = model.predict(X)[0]

        print(
            f"[{payload['timestamp']}] "
            f"T={payload['temperature_c']} H={payload['humidity_pct']}
"
            f"CO2={payload['co2_ppm']} P={payload['power_watt']} "
            f"-> status={pred_status}"
        )

        # Simpan ke InfluxDB
        point = (
            Point("room_pipeline")
            .tag("device_id", payload["device_id"])
            .field("temperature_c", float(payload["temperature_c"]))
            .field("humidity_pct", float(payload["humidity_pct"]))
            .field("co2_ppm", float(payload["co2_ppm"]))
            .field("power_watt", float(payload["power_watt"]))
```

```
            .field("pred_status_code", 0 if pred_status == "normal"
else (1 if pred_status == "warning" else 2))
            .tag("pred_status", str(pred_status))
            .time(payload["timestamp"], WritePrecision.S)
        )

        write_api.write(bucket=INFLUX_BUCKET, org=INFLUX_ORG,
record=point)

    except Exception as exc:
        print("Error processing message:", exc)

client = mqtt.Client(client_id="subscriber-room-pipeline")
client.on_connect = on_connect
client.on_message = on_message

client.connect(BROKER_HOST, BROKER_PORT, keepalive=60)
client.loop_forever()
```

Jalankan:

```
python scripts/pipeline_room_subscriber.py
```

8. Verifikasi ingest MQTT

Di terminal lain, jalankan:

```
mosquitto_sub -h localhost -t "iotlab/room01/telemetry" -v
```

Ini membantu memastikan aliran data dari simulator ke broker memang berjalan.

9. Menyusun dashboard Grafana

Tambahkan **InfluxDB** sebagai data source di Grafana:

- URL: `http://host.docker.internal:8086` atau `http://localhost:8086`
- Organization: `iotlab`
- Bucket: `sensor_data`
- Token: token Anda

Panel yang disarankan

Panel 1 — suhu real-time

- measurement: `room_pipeline`
- field: `temperature_c`

Panel 2 — kelembaban

- field: `humidity_pct`

Panel 3 — CO_2

- field: `co2_ppm`

Panel 4 — daya

- field: `power_watt`

Panel 5 — status prediksi

- tag: `pred_status`
- atau field numerik `pred_status_code`

Panel 6 — histori status

- tampilkan perubahan `normal → warning → fault`

Warna ambang yang disarankan

- `normal` → hijau
- `warning` → kuning
- `fault` → merah

10. Apa yang dipelajari pembaca dari Hands-on 30?

Hands-on ini mengajarkan bahwa:

- MQTT bukan hanya untuk kirim data, tetapi menjadi pintu masuk ke pipeline analitik,
- Python subscriber bisa menjadi pusat logika sistem,
- model bisa dipanggil otomatis tanpa notebook,
- hasil inferensi bisa disimpan dan langsung divisualisasikan,
- dan keseluruhan sistem dapat dijalankan secara lokal dengan stack open source.

Hands-on 31 - Membuat Alert Sederhana jika Skor Anomali Tinggi

1. Tujuan hands-on

Pada latihan ini, pembaca akan menambahkan **alert sederhana** ke pipeline end-to-end. Setelah selesai, pembaca diharapkan mampu:

- mendefinisikan skor atau kondisi alarm,
- memeriksa hasil prediksi atau skor anomali,
- menerbitkan alert ke topik MQTT lain,
- menyimpan alert ke storage,
- dan menampilkan alert pada dashboard.

Ini sangat penting karena sistem IoT-ML yang baik tidak berhenti pada prediksi; ia juga harus **mendorong tindakan** saat kondisi tertentu terdeteksi.

2. Dua cara sederhana membuat alert

Untuk tahap awal, ada dua pendekatan yang sangat realistis:

Pendekatan A — alert berbasis status model

Jika model memprediksi:

- `warning`
- atau `fault`
  maka alert diterbitkan.

Pendekatan B — alert berbasis skor / threshold

Jika skor anomali lebih tinggi dari ambang tertentu, alert diterbitkan.

Untuk buku ini, kita mulai dari pendekatan yang paling mudah dijelaskan:

- **status prediksi**
- dan **threshold sederhana**

3. Menambahkan publisher alert ke pipeline

Kita modifikasi subscriber agar:

- mengirim alert ke topik `iotlab/room01/alert`
- bila status `fault`
- atau jika kombinasi sensor menunjukkan kondisi berat

Buat file `scripts/pipeline_room_with_alert.py`:

```
import json
from collections import deque
from pathlib import Path

import joblib
import pandas as pd
import paho.mqtt.client as mqtt
from influxdb_client import InfluxDBClient, Point, WritePrecision
from influxdb_client.client.write_api import SYNCHRONOUS

BROKER_HOST = "localhost"
BROKER_PORT = 1883
TOPIC_INPUT = "iotlab/room01/telemetry"
TOPIC_ALERT = "iotlab/room01/alert"

MODEL_PATH = Path("models/room_status_rf.joblib")

INFLUX_URL = "http://localhost:8086"
INFLUX_TOKEN = "GANTI_DENGAN_TOKEN_ANDA"
INFLUX_ORG = "iotlab"
INFLUX_BUCKET = "sensor_data"

temp_buffer = deque(maxlen=3)
co2_buffer = deque(maxlen=3)
power_buffer = deque(maxlen=3)

model = joblib.load(MODEL_PATH)

influx_client = InfluxDBClient(url=INFLUX_URL, token=INFLUX_TOKEN,
org=INFLUX_ORG)
write_api = influx_client.write_api(write_options=SYNCHRONOUS)

def safe_mean(values):
    return sum(values) / len(values) if len(values) > 0 else None

def build_features(data):
    temp = float(data["temperature_c"])
    hum = float(data["humidity_pct"])
    co2 = float(data["co2_ppm"])
    power = float(data["power_watt"])

    temp_buffer.append(temp)
    co2_buffer.append(co2)
    power_buffer.append(power)

    ts = pd.to_datetime(data["timestamp"])

    return pd.DataFrame([{
        "temperature_c": temp,
        "humidity_pct": hum,
        "co2_ppm": co2,
        "power_watt": power,
        "hour": ts.hour,
        "day_of_week": ts.dayofweek,
        "is_weekend": int(ts.dayofweek >= 5),
        "temp_lag_1": temp_buffer[-2] if len(temp_buffer) >= 2 else
temp,
        "co2_lag_1": co2_buffer[-2] if len(co2_buffer) >= 2 else co2,
```

```
        "power_lag_1": power_buffer[-2] if len(power_buffer) >= 2 else
power,
        "temp_ma_3": safe_mean(temp_buffer),
        "co2_ma_3": safe_mean(co2_buffer),
        "power_ma_3": safe_mean(power_buffer),
    }])

def should_alert(payload, pred_status):
    # Aturan sederhana yang mudah dipahami
    if pred_status == "fault":
        return True, "Model memprediksi FAULT"

    if float(payload["temperature_c"]) > 32.0 and
float(payload["co2_ppm"]) > 1200:
        return True, "Suhu dan CO2 sama-sama tinggi"

    return False, ""

def on_connect(client, userdata, flags, reason_code, properties=None):
    print("Subscriber+Alert terhubung ke broker.")
    client.subscribe(TOPIC_INPUT, qos=1)
    print(f"Subscribe ke: {TOPIC_INPUT}")

def on_message(client, userdata, msg):
    try:
        payload = json.loads(msg.payload.decode())
        X = build_features(payload)
        pred_status = model.predict(X)[0]

        print(
            f"[{payload['timestamp']}] "
            f"-> status={pred_status}"
        )

        # Simpan inferensi biasa
        point = (
            Point("room_pipeline")
            .tag("device_id", payload["device_id"])
            .field("temperature_c", float(payload["temperature_c"]))
            .field("humidity_pct", float(payload["humidity_pct"]))
            .field("co2_ppm", float(payload["co2_ppm"]))
            .field("power_watt", float(payload["power_watt"]))
            .field("pred_status_code", 0 if pred_status == "normal"
else (1 if pred_status == "warning" else 2))
            .tag("pred_status", str(pred_status))
            .time(payload["timestamp"], WritePrecision.S)
        )
        write_api.write(bucket=INFLUX_BUCKET, org=INFLUX_ORG,
record=point)

        # Alert sederhana
        is_alert, alert_reason = should_alert(payload, pred_status)

        if is_alert:
            alert_payload = {
                "timestamp": payload["timestamp"],
                "device_id": payload["device_id"],
                "pred_status": pred_status,
                "alert_reason": alert_reason,
```

```
                    "temperature_c": payload["temperature_c"],
                    "humidity_pct": payload["humidity_pct"],
                    "co2_ppm": payload["co2_ppm"],
                    "power_watt": payload["power_watt"]
                }

                client.publish(TOPIC_ALERT, json.dumps(alert_payload),
qos=1)
                print("ALERT:", alert_payload)

                alert_point = (
                    Point("room_alerts")
                    .tag("device_id", payload["device_id"])
                    .tag("pred_status", str(pred_status))
                    .tag("alert_reason", alert_reason)
                    .field("temperature_c",
float(payload["temperature_c"]))
                    .field("humidity_pct", float(payload["humidity_pct"]))
                    .field("co2_ppm", float(payload["co2_ppm"]))
                    .field("power_watt", float(payload["power_watt"]))
                    .field("alert_flag", 1)
                    .time(payload["timestamp"], WritePrecision.S)
                )
                write_api.write(bucket=INFLUX_BUCKET, org=INFLUX_ORG,
record=alert_point)

            except Exception as exc:
                print("Error:", exc)

client = mqtt.Client(client_id="subscriber-room-alert")
client.on_connect = on_connect
client.on_message = on_message

client.connect(BROKER_HOST, BROKER_PORT, keepalive=60)
client.loop_forever()
```

Jalankan:

```
python scripts/pipeline_room_with_alert.py
```

4. Verifikasi alert di MQTT

Buka terminal baru:

```
mosquitto_sub -h localhost -t "iotlab/room01/alert" -v
```

Jika kondisi terpenuhi, akan terlihat payload alert muncul.

5. Menampilkan alert di Grafana

Tambahkan panel baru dengan measurement:

- `room_alerts`

Panel yang disarankan

- jumlah alert per jam
- daftar alert terbaru
- panel status dengan warna merah saat ada alert terbaru
- panel alasan alert (`alert_reason`)

Dashboard yang ideal

- panel sensor utama di atas
- panel status model di tengah
- panel alert di bawah atau di sisi kanan

Dengan ini, operator tidak hanya melihat data, tetapi juga melihat **kejadian yang perlu ditindaklanjuti**.

6. Versi sederhana berbasis skor anomali

Jika pembaca ingin memakai skor numerik, logikanya bisa dibuat sangat mudah.

Contoh:

```
anomaly_score = 0.0

if float(payload["temperature_c"]) > 31:
    anomaly_score += 0.4
if float(payload["co2_ppm"]) > 1100:
    anomaly_score += 0.4
if float(payload["power_watt"]) > 220:
    anomaly_score += 0.2

if anomaly_score >= 0.7:
    # kirim alert
    pass
```

Ini belum model anomaly detection penuh, tetapi cukup bagus untuk:

- tahap awal
- sistem ringan
- kombinasi threshold + skor sederhana

7. Kapan alert sederhana sudah cukup?

Alert sederhana sudah sangat berguna jika:

- sistem masih awal
- domain knowledge cukup kuat
- tim operasional butuh transparansi
- belum ada cukup data untuk model anomali yang lebih kompleks

Contoh:

- suhu > batas
- status model = `fault`
- kombinasi suhu + CO_2 tinggi
- daya di luar jam kerja

8. Kapan alert perlu dikembangkan?

Pengembangan lebih lanjut layak dilakukan jika:

- alarm palsu terlalu banyak
- pola masalah tidak cukup ditangkap threshold
- sistem punya banyak variabel
- baseline operasional sudah stabil
- data historis sudah cukup kaya

Pada tahap itu, pembaca bisa beralih ke:

- Isolation Forest
- One-Class SVM
- threshold dinamis
- atau model anomaly detection lain

9. Langkah kerja yang harus dilakukan pembaca

Hands-on 30

1. jalankan Mosquitto, InfluxDB, dan Grafana
2. jalankan simulator sensor
3. jalankan subscriber Python
4. validasi data masuk ke broker
5. pastikan preprocessing dan prediksi berjalan
6. periksa data dan hasil inferensi di InfluxDB
7. bangun dashboard Grafana

Hands-on 31

1. tambahkan logika alert ke subscriber
2. tentukan aturan alert sederhana
3. kirim alert ke topik MQTT khusus
4. simpan alert ke measurement khusus
5. tampilkan alert di dashboard
6. uji dengan nilai sensor ekstrem atau simulasi fault

10. Penutup

Hands-on 30 dan 31 adalah bentuk paling konkret dari seluruh perjalanan buku ini. Pada Hands-on 30, pembaca membangun **pipeline lengkap** dari **sensor/simulator → MQTT → Python → preprocessing → prediksi → storage → dashboard**. Pada

Hands-on 31, pembaca menambahkan **alert sederhana** sehingga sistem tidak hanya memprediksi, tetapi juga mulai **bertindak**. Inilah inti dari sistem IoT-ML yang berguna: bukan sekadar mengumpulkan data atau melatih model, tetapi membuat alur yang bisa bekerja terus-menerus dan memberi manfaat operasional nyata.

Bab 13. Keamanan, Kualitas Model, dan Operasional

13.1 Mengapa bab ini penting?

Setelah pembaca belajar:

- mengirim data sensor,
- membersihkan data,
- membuat fitur,
- melatih model,
- menjalankan inferensi,
- dan membangun pipeline end-to-end,

langkah berikutnya adalah memastikan sistem itu **tetap waras, tetap aman, dan tetap berguna** saat dijalankan terus-menerus. Dalam arsitektur IoT modern, ini berarti kita tidak cukup hanya berpikir tentang model, tetapi juga tentang **data quality**, **security**, **monitoring**, **versioning**, dan **operational trust**. Referensi tentang edge/cloud IoT juga menekankan bahwa sistem nyata harus menangani layanan real-time, interoperabilitas, dan pengelolaan data/komputasi secara berkelanjutan, bukan hanya eksperimen sesaat.

Dengan kata lain, kalau Bab 1–12 mengajarkan **cara membangun**, maka bab ini mengajarkan **cara menjaga sistem tetap layak dipakai**.

13.2 Risiko data palsu, data hilang, data rusak

13.2.1 Mengapa kualitas data adalah garis pertahanan pertama?

Model machine learning tidak akan pernah lebih baik daripada data yang masuk ke dalamnya. Pada sistem IoT, data berasal dari sensor, gateway, jaringan, broker, dan parser aplikasi. Artinya, titik kegagalan sangat banyak. Referensi Mohanty menegaskan bahwa data IoT bersifat **heterogen** dan perlu melalui **cleansing, ordering, dan preprocessing**, karena kombinasi data dari berbagai sumber yang tidak tertata bisa menghasilkan **false prediction**.

13.2.2 Data palsu

Data palsu bisa muncul karena beberapa hal:

- perangkat mengirim payload hasil manipulasi,
- sensor dibajak atau dipalsukan,
- sistem simulator tidak dibedakan dari sensor nyata,
- nilai dibuat-buat oleh bug aplikasi,
- atau *false data injection* pada rantai pengukuran.

Referensi Obaidat menyinggung secara eksplisit bahwa pada sistem pengukuran cerdas, **false data injection** dapat menurunkan reliabilitas sistem dan perlu ditangani dengan mekanisme deteksi tambahan. Walaupun contoh referensinya

berasal dari konteks power system, logikanya sangat relevan untuk IoT umum: data palsu bisa membuat analitik dan kontrol menjadi menyesatkan.

13.2.3 Data hilang

Data hilang adalah masalah sangat umum di IoT. Penyebabnya bisa berupa:

- Wi-Fi putus,
- broker sibuk,
- perangkat reboot,
- baterai habis,
- gateway restart,
- atau subscriber gagal memproses pesan.

Dampaknya:

- *rolling feature* rusak,
- dashboard menampilkan tren yang patah,
- model menerima input tidak lengkap,
- inferensi menjadi bias,
- dan alert bisa gagal muncul.

13.2.4 Data rusak

Data rusak mencakup:

- payload JSON tidak valid,
- field penting hilang,
- tipe data salah,
- *timestamp* rusak,
- nilai sensor di luar rentang wajar,
- satuan tidak konsisten.

Contoh:

- `temperature_c = "abc"`
- `co2_ppm = -40`
- `timestamp = "bad_time"`

13.2.5 Langkah minimum yang wajib dilakukan

Untuk buku ini, praktik minimum yang sangat disarankan:

- validasi field wajib,
- cek tipe data,
- cek rentang nilai wajar,
- tandai record yang gagal validasi,
- simpan log error,
- jangan langsung memasukkan record rusak ke model.

13.2.6 Contoh validasi sederhana

Sebelum model dipanggil:

- suhu harus numerik dan masuk akal,
- kelembaban 0–100%,
- CO_2 tidak negatif,
- `device_id` harus ada,
- timestamp bisa di-parse.

Ini sederhana, tetapi sangat menentukan kualitas sistem.

13.3 Dasar keamanan MQTT

13.3.1 Mengapa MQTT security perlu dibahas?

Karena dalam arsitektur buku ini, MQTT adalah tulang punggung aliran data. Jika broker atau topik MQTT tidak dijaga sama sekali, siapa pun yang punya akses jaringan bisa:

- mengirim data palsu,
- membaca telemetry,
- membanjiri topik,
- atau memicu alert palsu.

Referensi Obaidat menjelaskan bahwa **MQTT** adalah protokol *publish/subscribe* yang ringan, memakai TCP, dan untuk keamanan dapat menggunakan **SSL/TLS**. Referensi itu juga menegaskan bahwa broker bertanggung jawab mengorkestrasi komunikasi antara publisher dan subscriber, sehingga broker menjadi titik yang sangat penting untuk dijaga.

13.3.2 Tiga dasar keamanan MQTT yang realistis untuk buku ini

Untuk buku praktis, disarankan tiga lapisan minimum:

1. Autentikasi
Jangan biarkan broker terbuka tanpa identitas. Minimal gunakan:

- username/password,
- atau kredensial khusus untuk publisher dan subscriber.

2. Segmentasi sederhana
Pisahkan topik dan, bila mungkin, pisahkan jaringan atau VLAN sederhana untuk device IoT.

3. TLS bila memungkinkan
Jika deployment sudah melampaui lab kecil, gunakan enkripsi transport agar data tidak lewat polos begitu saja di jaringan.

13.3.3 Praktik minimum di level topik

Minimal lakukan:

- topik telemetry berbeda dari topik alert,
- topik training/debug jangan terbuka untuk semua klien,
- hak publish dan subscribe dibedakan bila memungkinkan.

Contoh:

- `iotlab/room01/telemetry`
- `iotlab/room01/alert`
- `iotlab/admin/retrain`

13.3.4 Praktik operasional yang sehat

- jangan pakai broker terbuka di jaringan umum,
- jangan hardcode token dan password di banyak file,
- bedakan lingkungan `dev`, `test`, dan `prod`,
- simpan konfigurasi sensitif di file `.env` atau secret manager sederhana jika memungkinkan.

13.3.5 Batas pembahasan yang realistis

Bab ini tidak perlu tenggelam ke keamanan enterprise yang terlalu luas. Untuk target buku ini, yang penting adalah pembaca paham bahwa:

- MQTT **harus** dijaga,
- autentikasi dan segmentasi sederhana sangat penting,
- dan broker adalah komponen operasional yang tidak boleh diperlakukan sembarangan.

13.4 Drift data dan penurunan performa model

13.4.1 Apa itu drift?

Drift berarti pola data berubah dari waktu ke waktu. Ini bisa terjadi karena:

- sensor menua,
- ruangan berubah penggunaan,
- mesin makin aus,
- jadwal operasional berubah,
- pola energi berubah,
- atau lingkungan fisik berubah.

Di level operasional model, drift berarti model yang dulunya bagus bisa **diam-diam** menjadi semakin buruk.

13.4.2 Bentuk drift yang umum

Data drift
Distribusi input berubah. Contoh:

- suhu rata-rata kini lebih tinggi,
- CO_2 lebih sering tinggi karena ruang dipakai lebih padat,
- pola beban energi bergeser karena jam kerja berubah.

Concept drift
Hubungan antara fitur dan target berubah. Contoh:

- suhu yang dulu dianggap normal kini justru terkait keluhan karena ventilasi berubah,
- pola getaran yang dulu aman kini menandakan masalah karena mesin sudah menua.

13.4.3 Mengapa drift berbahaya?

Karena model sering gagal **pelan-pelan**, bukan tiba-tiba. Sistem tampak masih hidup, dashboard masih bergerak, prediksi masih keluar, tetapi kualitasnya menurun.

13.4.4 Tanda-tanda drift operasional

- distribusi fitur input bergeser,
- rasio kelas prediksi berubah drastis,
- error forecasting meningkat,
- alarm makin sering terlalu banyak atau terlalu sedikit,
- teknisi mulai tidak percaya hasil model.

13.4.5 Langkah minimum untuk memonitor drift

Untuk buku ini, pendekatan minimum yang realistis:

- pantau statistik ringkas fitur utama,
- bandingkan rata-rata/median mingguan dengan baseline,
- pantau distribusi label prediksi,
- simpan error aktual vs prediksi untuk use case yang punya ground truth,
- jadwalkan batch retraining sederhana bila perubahan cukup besar.

Referensi Mohanty juga menekankan bahwa data heterogen dari berbagai sumber harus diperlakukan hati-hati agar prediksi tidak menjadi salah; ini secara praktis memperkuat alasan mengapa monitoring distribusi input perlu dilakukan secara rutin.

13.5 Logging, monitoring, dan versi model sederhana

13.5.1 Mengapa logging wajib?

Tanpa **logging**, sistem IoT-ML akan sulit didiagnosis saat ada masalah. Kita tidak tahu:

- payload mana yang gagal,
- model versi berapa yang aktif,
- kapan inferensi mulai error,
- apakah retraining terakhir berhasil,
- atau kapan alert palsu mulai meningkat.

13.5.2 Apa yang minimal harus dicatat?

Log ingest

- timestamp masuk
- device_id
- apakah payload valid atau tidak

Log inferensi

- timestamp inferensi
- model aktif
- fitur minimum / input utama
- output prediksi
- skor atau status

Log error

- parsing error
- koneksi MQTT gagal
- tulis database gagal
- model file tidak ditemukan
- inferensi exception

13.5.3 Monitoring minimum yang realistis

Monitoring sederhana yang sangat berguna:

- jumlah pesan masuk per jam,
- jumlah record valid vs invalid,
- jumlah prediksi per kelas,
- jumlah alert,
- latensi inferensi,
- kegagalan koneksi ke broker atau database.

13.5.4 Versi model sederhana

Pembaca tidak perlu langsung masuk ke MLOps enterprise. Cukup mulai dari praktik berikut:

- simpan model dengan nama versi, misalnya:
  - `room_model_v1.joblib`
  - `room_model_v2.joblib`
- simpan metadata:
  - tanggal training
  - dataset yang dipakai
  - metrik validasi
- tulis versi model ke log inferensi

13.5.5 Kenapa versi model penting?

Karena tanpa versi, saat performa turun kita tidak tahu:

- apakah masalahnya karena drift,
- karena model baru lebih buruk,
- atau karena pipeline input berubah.

13.5.6 Praktik minimum yang sangat disarankan

Simpan file seperti:

```
models/
├── room_model_v1.joblib
├── room_model_v2.joblib
└── metadata_room_model_v2.json
```

Ini sederhana, tetapi sangat berguna.

13.6 Explain Ability ringan

13.6.1 Mengapa explainability perlu dibahas?

Karena banyak pengguna operasional tidak puas hanya dengan output:

- “status = fault”
- “skor anomali = 0.91”

Mereka ingin tahu:
kenapa?

Explain Ability ringan untuk model pohon keputusan / random forest. Ini sangat tepat, karena model berbasis pohon relatif lebih mudah dijelaskan dibanding model yang sangat kompleks.

13.6.2 Explain Ability yang realistis untuk buku ini

Untuk buku praktis, explain ability tidak perlu langsung masuk ke SHAP/LIME yang berat. Mulailah dari:

- **feature importance**
- **aturan threshold yang memicu alert**
- **perbandingan fitur saat ini dengan baseline**
- **narasi sederhana untuk operator**

13.6.3 Contoh explain ability ringan

Contoh keluaran yang bagus:

- "Status `warning` karena CO_2 tinggi dan suhu di atas rata-rata 1 jam terakhir."
- "Anomali kemungkinan dipicu oleh kenaikan arus dan daya secara bersamaan."
- "Prediksi `fault` didominasi oleh fitur RMS getaran dan power_ma_3."

13.6.4 Mengapa model pohon/random forest cocok?

Karena pada model ini, feature importance lebih mudah diambil dan dijelaskan. Itu sebabnya dipilih explain ability ringan untuk model jenis ini, bukan untuk model deep learning yang lebih sulit diterjemahkan.

13.6.5 Kapan explain ability cukup dianggap "baik"?

Untuk tahap ini, explain ability sudah cukup baik jika:

- membantu teknisi memahami faktor dominan,
- membantu membedakan alarm yang masuk akal dari alarm aneh,
- dan membantu tim operasional membangun kepercayaan pada model.

13.7 Synthetic data untuk stress testing pipeline

13.7.1 Mengapa synthetic data relevan lagi di bab operasional?

Karena *synthetic data* bukan hanya untuk belajar model. Ia juga sangat berguna untuk **stress testing pipeline**:

- menguji beban broker,
- menguji subscriber saat data buruk datang,
- menguji dashboard saat lonjakan alarm terjadi,
- menguji apakah model tetap stabil saat input aneh masuk.

Synthetic data untuk stress testing pipeline sebagai bagian bab ini. Ini sangat kuat karena menunjukkan bahwa synthetic data dipakai bukan hanya sebagai data latihan, tetapi juga sebagai alat *quality assurance* sistem.

13.7.2 Jenis stress test yang realistis

Stress test volume

- kirim data jauh lebih cepat dari biasanya

Stress test kualitas

- kirim payload dengan field hilang
- kirim tipe data salah
- kirim nilai ekstrem

Stress test pola

- injeksi drift buatan
- injeksi lonjakan fault beruntun
- injeksi kombinasi anomali

13.7.3 Tujuan stress testing

- melihat apakah pipeline tetap jalan,
- melihat apakah parsing rusak,
- melihat apakah storage tahan,
- melihat apakah alert terlalu berisik,
- melihat apakah dashboard tetap berguna,
- melihat apakah model mulai memberi output aneh.

13.7.4 Praktik minimum yang sangat bagus untuk pembaca

Sebelum deployment mini:

- jalankan simulator normal,
- jalankan simulator anomali,
- jalankan simulator payload rusak,
- pantau log, dashboard, dan alert.

13.7.5 Kenapa ini penting?

Karena sistem yang terlihat baik saat data normal masuk belum tentu tahan saat kondisi lapangan buruk. Dan dunia nyata hampir selalu memberi kondisi buruk.

13.8 Contoh-contoh nyata yang bisa diimplementasikan

Smart room monitoring

- data palsu: CO_2 palsu tinggi terus-menerus
- data hilang: sensor tidak kirim 20 menit
- drift: suhu rata-rata ruangan naik bertahap
- logging: catat prediksi `normal/warning/fault`
- explainability: tampilkan fitur dominan pada prediksi warning
- stress test: kirim payload ekstrem selama 1 jam simulasi

Predictive maintenance sederhana

- data rusak: getaran hilang sebagian
- data palsu: arus melonjak palsu
- drift: mesin makin bising perlahan
- monitoring: pantau skor anomali rata-rata
- versi model: ganti model v1 ke v2 setelah retraining
- explain ability: fitur RMS dan STD sebagai penjelas utama

Smart energy analytics

- data hilang: meter terlambat kirim
- drift: pola beban bergeser karena jam operasional berubah
- monitoring: error forecasting mingguan
- explain ability: fitur jam, lag daya, dan rata-rata lokal
- stress test: simulasi lonjakan daya tidak normal

13.9 Langkah kerja yang harus dilakukan pembaca

Agar bab ini benar-benar operasional, pembaca dapat mengikuti urutan berikut.

Tahap 1 — Lindungi data masuk

Lakukan:

- validasi payload
- cek rentang nilai
- cek field wajib
- pisahkan data valid dan invalid

Tahap 2 — Amankan MQTT minimum

Terapkan:

- autentikasi
- segmentasi topik
- bila memungkinkan TLS

Tahap 3 — Pantau drift

Bandingkan:

- distribusi fitur sekarang vs baseline
- rasio kelas prediksi
- error model dari waktu ke waktu

Tahap 4 — Bangun logging dan monitoring

Catat:

- ingest
- inferensi
- error
- alert
- versi model

Tahap 5 — Versikan model

Simpan model dan metadata secara rapi.

Tahap 6 — Tambahkan explain ability ringan

Minimal tampilkan feature importance atau faktor dominan.

Tahap 7 — Lakukan stress test

Gunakan synthetic data untuk:

- volume tinggi
- payload rusak
- drift buatan
- anomali beruntun

Tahap 8 — Evaluasi kesiapan operasional

Tanyakan:

- apakah sistem tetap jalan?
- apakah alert tetap berguna?
- apakah hasil model masih bisa dipercaya?

13.10 Sikap profesional dalam keamanan, kualitas model, dan operasional

Ada satu pelajaran penting yang perlu ditekankan: **sistem IoT-ML yang baik bukan hanya sistem yang akurat, tetapi sistem yang tahan gangguan, mudah diawasi, dan tetap bisa dipercaya saat kondisi memburuk**.

Artinya:

- keamanan dasar harus dipikirkan sejak awal,
- kualitas data harus dipantau terus,
- drift tidak boleh diabaikan,
- model harus diberi versi,
- hasil model harus cukup bisa dijelaskan,
- dan pipeline harus diuji saat kondisi buruk, bukan hanya saat normal.

Itulah pembeda antara demo laboratorium dan sistem yang benar-benar siap dipakai.

13.11 Penutup bab

Bab 13 adalah bab yang mengubah proyek IoT-ML dari "bisa jalan" menjadi "layak dipakai." Di sini pembaca memahami bahwa ancaman utama sistem bukan hanya kurangnya algoritma canggih, tetapi juga **data palsu, data hilang, data rusak, broker yang tidak aman, drift yang diam-diam menggerus performa, logging yang tidak ada, versi model yang berantakan, hasil model yang sulit dijelaskan, dan pipeline yang tidak pernah diuji di bawah tekanan**. Karena itu, topik-topik seperti **dasar keamanan MQTT, monitoring drift, logging, versi model sederhana, explain ability ringan, dan synthetic data untuk stress testing** adalah fondasi operasional minimum yang wajib.

Hands-on 32 - Memonitor Perubahan Distribusi Data

1. Tujuan hands-on

Pada latihan ini, pembaca akan membangun pemantauan sederhana untuk melihat apakah distribusi data sensor **bergeser** dari baseline. Setelah selesai, pembaca diharapkan mampu:

- membaca data historis baseline dan data baru,
- membandingkan statistik ringkas antar-periode,
- melihat perubahan distribusi fitur utama,
- membuat indikator drift sederhana,
- dan menyimpan hasil monitoring untuk operasional.

Latihan ini sangat penting karena **konsep drift data dan penurunan performa model** sebagai praktik minimum yang wajib.

2. Contoh implementasi nyata

Smart room monitoring

- suhu rata-rata ruangan perlahan naik karena ventilasi berubah
- CO_2 jadi lebih tinggi karena ruangan lebih sering penuh
- model lama mulai terlalu sering memberi `warning`

Smart energy analytics

- pola beban berubah karena jam operasional bergeser
- puncak daya tidak lagi muncul pada jam yang sama
- model forecasting mulai meleset

Predictive maintenance sederhana

- getaran rata-rata naik perlahan karena mesin menua
- baseline “normal” lama tidak lagi representatif

3. Prinsip monitoring drift yang realistis

Untuk buku ini, kita **tidak perlu** langsung memakai paket drift detection yang berat. Pendekatan yang sangat realistis untuk mahasiswa dan praktisi adalah:

- bandingkan **mean**
- bandingkan **std**
- bandingkan **median**
- bandingkan **quantile**
- bandingkan **histogram**
- lalu beri ambang sederhana untuk menandai perubahan mencolok

Ini cukup kuat untuk tahap awal, dan jauh lebih mudah dipahami.

4. Dataset yang dipakai

Gunakan dua potongan data:

- **baseline**: data lama yang dianggap sehat
- **current**: data baru yang ingin dibandingkan

Contoh:

```
data/synthetic/room_30days.csv
```

Kita akan pakai:

- 14 hari pertama sebagai baseline
- 7 hari terakhir sebagai current

5. Persiapan Ubuntu 26.04

```
source .venv/bin/activate
pip install pandas numpy matplotlib
mkdir -p scripts outputs/figures outputs/reports
```

6. Source code utama: monitoring distribusi data

Buat file `scripts/monitor_data_distribution.py`:

```
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt

INPUT_PATH = Path("data/synthetic/room_30days.csv")
OUTPUT_DIR = Path("outputs")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

df = pd.read_csv(INPUT_PATH)
df["timestamp"] = pd.to_datetime(df["timestamp"])
df = df.sort_values("timestamp")

# Baseline: 14 hari pertama
baseline_end = df["timestamp"].min() + pd.Timedelta(days=14)
baseline_df = df[df["timestamp"] < baseline_end].copy()

# Current: 7 hari terakhir
current_start = df["timestamp"].max() - pd.Timedelta(days=7)
current_df = df[df["timestamp"] >= current_start].copy()

features = ["temperature_c", "humidity_pct"]

results = []

for col in features:
    baseline_mean = baseline_df[col].mean()
    current_mean = current_df[col].mean()

    baseline_std = baseline_df[col].std()
    current_std = current_df[col].std()

    baseline_median = baseline_df[col].median()
    current_median = current_df[col].median()

    mean_shift = current_mean - baseline_mean
    std_shift = current_std - baseline_std

    results.append({
        "feature": col,
        "baseline_mean": baseline_mean,
        "current_mean": current_mean,
        "mean_shift": mean_shift,
        "baseline_std": baseline_std,
        "current_std": current_std,
        "std_shift": std_shift,
        "baseline_median": baseline_median,
        "current_median": current_median
    })

results_df = pd.DataFrame(results)
results_df.to_csv("outputs/reports/data_distribution_monitor.csv",
index=False)

print("=== Ringkasan perubahan distribusi ===")
```

```
print(results_df)

# Visualisasi histogram
fig_dir = Path("outputs/figures")
fig_dir.mkdir(parents=True, exist_ok=True)

for col in features:
    plt.figure(figsize=(8, 5))
    plt.hist(baseline_df[col], bins=30, alpha=0.6, label="Baseline")
    plt.hist(current_df[col], bins=30, alpha=0.6, label="Current")
    plt.title(f"Perbandingan Distribusi {col}")
    plt.xlabel(col)
    plt.ylabel("Frekuensi")
    plt.legend()
    plt.tight_layout()
    plt.savefig(fig_dir / f"dist_compare_{col}.png")
    plt.show()
```

Jalankan:

```
python scripts/monitor_data_distribution.py
```

7. Cara membaca hasil

Perhatikan:

- apakah `current_mean` bergeser jauh dari `baseline_mean`
- apakah variasi (`std`) membesar
- apakah histogram bergeser ke kanan atau kiri
- apakah bentuk distribusi menjadi berbeda

Contoh interpretasi

- `temperature_c` mean naik +1.5°C → ruangan cenderung lebih panas dari baseline
- `humidity_pct` std membesar → kelembaban lebih tidak stabil
- distribusi CO_2 bergeser ke kanan → okupansi atau ventilasi berubah

8. Menambahkan indikator drift sederhana

Bila ingin sangat praktis, gunakan aturan sederhana seperti:

```
if abs(mean_shift) > 1.0:
    print("WARNING: mean feature bergeser signifikan")
```

Untuk buku ini, pendekatan sederhana seperti ini sudah sangat berguna sebagai alarm awal drift operasional.

9. Insight penting Hands-on 32

Latihan ini mengajarkan bahwa:

- drift bisa dideteksi **tanpa** framework rumit,
- monitoring distribusi fitur utama adalah langkah minimum yang realistis,
- dan perubahan distribusi input sering menjadi tanda awal penurunan performa model.

Hands-on 33 - Feature Importance untuk Menjelaskan Prediksi

1. Tujuan hands-on

Pada latihan ini, pembaca akan mengambil **feature importance** dari model pohon keputusan / random forest untuk menjelaskan prediksi secara ringan. Setelah selesai, pembaca diharapkan mampu:

- memuat model Random Forest yang sudah dilatih,
- mengekstrak nilai feature importance,
- memvisualisasikan fitur paling berpengaruh,
- dan menulis narasi sederhana untuk menjelaskan hasil model.

Ini sangat selaras dengan **explain ability ringan untuk model pohon keputusan / random forest** sebagai bagian penting bab operasional.

2. Mengapa feature importance penting?

Dalam praktik operasional, tim tidak selalu puas hanya dengan:

- `status = fault`
- `prediksi = warning`

Mereka ingin tahu:

- fitur apa yang paling mempengaruhi hasil
- apakah CO_2 lebih penting dari suhu
- apakah daya atau suhu yang dominan
- apakah model masuk akal secara domain

Feature importance memberi jalan tengah yang praktis: cukup ringan untuk pembelajaran, tetapi tetap sangat berguna.

3. Dataset dan model yang dipakai

Gunakan model Random Forest dari bab sebelumnya, misalnya:

```
models/room_status_rf.joblib
```

dan fitur dari dataset klasifikasi status:

```
data/processed/classification_status/X_train.csv
```

4. Source code utama

Buat file `scripts/feature_importance_room_model.py`:

```
from pathlib import Path
import joblib
import pandas as pd
import matplotlib.pyplot as plt

MODEL_PATH = Path("models/room_status_rf.joblib")
X_TRAIN_PATH =
Path("data/processed/classification_status/X_train.csv")
OUTPUT_DIR = Path("outputs")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

model = joblib.load(MODEL_PATH)
X_train = pd.read_csv(X_TRAIN_PATH)

importances = model.feature_importances_

importance_df = pd.DataFrame({
    "feature": X_train.columns,
    "importance": importances
}).sort_values("importance", ascending=False)

importance_df.to_csv("outputs/reports/feature_importance_room_model.cs
v", index=False)

print("=== Feature Importance ===")
print(importance_df)

# Visualisasi top 10
top_n = 10
plot_df = importance_df.head(top_n).sort_values("importance",
ascending=True)

plt.figure(figsize=(8, 6))
plt.barh(plot_df["feature"], plot_df["importance"])
plt.title("Top Feature Importance - Random Forest Room Status")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.tight_layout()
plt.savefig("outputs/figures/feature_importance_room_model.png")
plt.show()
```

Jalankan:

```
python scripts/feature_importance_room_model.py
```

5. Cara membaca hasil

Misalnya hasil teratas adalah:

- `co2_ppm`
- `temperature_c`
- `power_ma_3`
- `temp_ma_3`

Maka interpretasinya bisa ditulis seperti ini:

> Prediksi status ruangan paling banyak dipengaruhi oleh **CO_2**, **suhu**, dan **rata-rata daya lokal**, sehingga model tidak hanya melihat satu sensor tunggal, tetapi kombinasi kondisi lingkungan dan beban.

6. Explainability ringan untuk satu prediksi

Untuk buku ini, kita juga bisa menambahkan narasi sederhana berbasis fitur dominan dan nilai saat ini.

Contoh:

```
top_features = importance_df.head(3)["feature"].tolist()
print("Tiga fitur paling berpengaruh:", top_features)
```

Lalu narasi operasional:

- jika `co2_ppm` dan `temperature_c` dominan, maka perubahan ventilasi dan okupansi kemungkinan sangat penting
- jika `power_ma_3` dominan, maka beban listrik lokal mempengaruhi status
- jika `temp_lag_1` dominan, maka histori jangka pendek suhu sangat penting

7. Mengapa ini cukup untuk tahap ini?

Karena target bab ini adalah **explainability ringan**, bukan interpretabilitas penuh berbasis SHAP/LIME. Feature importance sudah:

- cukup mudah dipahami,
- cukup kuat untuk membangun kepercayaan,
- dan sangat cocok untuk model pohon/random forest.

8. Insight penting Hands-on 33

Latihan ini mengajarkan bahwa:

- explain ability tidak harus rumit,
- model yang lebih “kuat” tetap bisa dijelaskan secara ringan,
- dan feature importance sangat membantu jembatan antara data scientist dan operator.

Hands-on 34 - Checklist Operasional Sistem IoT-ML

1. Tujuan hands-on

Pada latihan ini, pembaca akan menyusun **checklist operasional** untuk memastikan sistem IoT-ML siap dipakai, tidak hanya “bisa jalan sekali”. Setelah selesai, pembaca diharapkan mampu:

- memeriksa kesiapan data,
- memeriksa keamanan dasar,
- memeriksa kesiapan model,
- memeriksa logging dan monitoring,
- memeriksa alert dan dashboard,
- dan menilai apakah sistem cukup stabil untuk dijalankan.

2. Mengapa checklist operasional penting?

Karena banyak sistem gagal bukan karena algoritma salah, tetapi karena hal-hal sederhana:

- broker tidak dijaga
- payload tidak divalidasi
- model versi aktif tidak jelas
- dashboard hidup tetapi model diam
- inferensi jalan tetapi error tidak tercatat
- alert terlalu banyak atau tidak pernah muncul

Checklist membantu mengubah sistem dari:

- **demo**
  menjadi
- **sistem yang layak dipakai**

3. Format checklist yang disarankan

Buat file markdown atau CSV sederhana agar tim bisa memakainya berulang.

Contoh file:

```
outputs/reports/iot_ml_operational_checklist.md
```

4. Template checklist operasional

Buat file `scripts/generate_operational_checklist.py`:

```
from pathlib import Path

OUTPUT_PATH = Path("outputs/reports/iot_ml_operational_checklist.md")
OUTPUT_PATH.parent.mkdir(parents=True, exist_ok=True)

content = """
# Checklist Operasional Sistem IoT-ML

## A. Data & Sensor
- [ ] Semua field wajib tersedia pada payload (`timestamp`,
`device_id`, sensor utama)
- [ ] Tipe data numerik sudah diverifikasi
- [ ] Rentang nilai sensor masuk akal
- [ ] Timestamp valid dan konsisten
- [ ] Data hilang dicatat
- [ ] Data rusak/log parsing error dicatat

## B. MQTT & Jaringan
- [ ] Broker MQTT berjalan stabil
- [ ] Topik telemetry dan alert dipisah
- [ ] Autentikasi minimum diterapkan
- [ ] Hak publish/subscribe jelas
- [ ] Pesan uji berhasil masuk dan diterima subscriber

## C. Preprocessing
- [ ] Semua fitur yang dibutuhkan model tersedia
- [ ] Lag / rolling feature punya fallback bila data awal belum cukup
- [ ] Data invalid tidak langsung masuk ke model
- [ ] Pipeline preprocessing konsisten dengan saat training

## D. Model
- [ ] File model aktif tersedia
- [ ] Versi model tercatat
- [ ] Metadata model tersedia (tanggal training, metrik)
- [ ] Feature importance / penjelasan ringan tersedia
- [ ] Model diuji pada data terbaru

## E. Inferensi
- [ ] Inferensi berjalan otomatis
- [ ] Error inferensi tercatat
- [ ] Latensi inferensi masih wajar
- [ ] Output model tersimpan

## F. Storage
- [ ] Data mentah tersimpan
- [ ] Hasil inferensi tersimpan
- [ ] Alert tersimpan
- [ ] Historis dapat dibaca ulang untuk retraining

## G. Dashboard & Alert
- [ ] Dashboard menampilkan sensor utama
- [ ] Dashboard menampilkan hasil model
- [ ] Alert terlihat jelas
- [ ] Alert tidak terlalu sering salah
```

```
- [ ] Operator paham makna tiap panel utama

## H. Monitoring Drift & Retraining
- [ ] Distribusi fitur utama dimonitor
- [ ] Error model dipantau
- [ ] Drift dicatat
- [ ] Jadwal retraining sederhana tersedia

## I. Stress Test
- [ ] Pipeline pernah diuji dengan payload rusak
- [ ] Pipeline pernah diuji dengan anomali sintetis
- [ ] Pipeline pernah diuji dengan volume data lebih tinggi
- [ ] Sistem tetap stabil saat uji beban ringan

## Kesimpulan
- [ ] Sistem siap untuk demo internal
- [ ] Sistem siap untuk pilot terbatas
- [ ] Sistem butuh perbaikan sebelum dipakai
"""

OUTPUT_PATH.write_text(content, encoding="utf-8")
print("Checklist disimpan ke:", OUTPUT_PATH)
```

Jalankan:

```
python scripts/generate_operational_checklist.py
```

5. Cara memakai checklist ini

Checklist ini bisa dipakai:

- sebelum demo
- sebelum deployment mini
- setelah retraining model
- setelah perubahan arsitektur
- saat audit internal sederhana

6. Versi checklist yang lebih kuantitatif

Kalau ingin sedikit lebih formal, tambahkan skor:

- 0 = belum ada
- 1 = ada tetapi belum stabil
- 2 = sudah ada dan stabil

Contoh:

```
Autentikasi MQTT: 0 / 1 / 2
Logging inferensi: 0 / 1 / 2
Dashboard status model: 0 / 1 / 2
```

Dengan ini, pembaca bisa menilai kesiapan sistem secara lebih objektif.

7. Insight penting Hands-on 34

Latihan ini mengajarkan bahwa:

- sistem IoT-ML yang baik harus bisa diperiksa secara sistematis,
- operasi yang stabil membutuhkan disiplin, bukan hanya coding,
- dan checklist sederhana sering menjadi pembeda antara proyek mahasiswa yang rapi dan proyek yang cepat rapuh.

Penutup

Hands-on 32–34 adalah bagian yang membuat bab operasional ini benar-benar hidup. Pada **Hands-on 32**, pembaca belajar memonitor perubahan distribusi data sebagai sinyal awal drift. Pada **Hands-on 33**, pembaca belajar memakai **feature importance** untuk menjelaskan prediksi secara ringan, sesuai ruh explain ability untuk model pohon/random forest. Pada **Hands-on 34**, pembaca merangkum semuanya dalam **checklist operasional sistem IoT-ML**, sehingga sistem tidak hanya akurat, tetapi juga lebih siap dipakai, diawasi, dan dipelihara.

Bagian V — Studi Kasus Praktis (Bab 14–15)

Bab 14. Studi Kasus 1–3: Dari Dataset ke Solusi

14.1 Mengapa studi kasus penting di bagian akhir buku?

Setelah pembaca mempelajari fondasi IoT, data sensor, *feature engineering*, supervised learning, anomaly detection, forecasting, edge inference, pipeline end-to-end, serta keamanan dan operasional, ada satu kebutuhan yang sangat penting: **melihat semua potongan itu bekerja bersama dalam konteks masalah nyata**. Itulah peran studi kasus.

Studi kasus mengubah pembelajaran dari:

- “saya tahu cara memakai pandas”
- “saya tahu cara melatih model”
- “saya tahu cara subscribe MQTT”

menjadi:

- “saya bisa membangun solusi kecil yang masuk akal untuk masalah nyata.”

Bab ini ditetapkan sebagai **“Dari Dataset ke Solusi”**, bukan sekadar kumpulan eksperimen terpisah. Itu berarti fokusnya bukan lagi pada algoritma sebagai tujuan, melainkan pada **alur pemecahan masalah** yang utuh.

14.2 Pola yang dipakai untuk setiap studi kasus

Agar konsisten dan mudah diikuti mahasiswa maupun praktisi, tiap studi kasus di bab ini memakai struktur yang sama:

- **masalah**
- **arsitektur**
- **dataset**
- **synthetic data fallback**
- **preprocessing**
- **model**
- **evaluasi**
    - **deployment mini**

Mengapa pola ini efektif?

Karena pembaca tidak hanya belajar “model apa yang dipakai”, tetapi juga:

- dari mana data datang,
- apa fallback jika data belum ada,
- bagaimana data dibersihkan,

- bagaimana solusi diuji,
- dan bagaimana solusi dijalankan secara mini di lapangan.

Dengan pola ini, pembaca bisa meniru struktur yang sama untuk kasus lain di luar buku.

Studi Kasus A — Smart Room Monitoring

14.A.1 Masalah

Banyak ruang kerja, ruang kelas, laboratorium, dan ruang server mengalami masalah yang tampak sederhana tetapi sebenarnya penting:

- ruangan terlalu panas,
- kelembaban tidak stabil,
- CO_2 meningkat saat ruangan penuh,
- pendinginan bekerja lebih keras,
- dan pengguna merasa tidak nyaman meskipun sistem tampak “normal”.

Referensi Perros secara eksplisit menempatkan **smart home** dan **occupancy-aware control** sebagai domain IoT penting, termasuk penggunaan **CO_2 sensors** untuk mendeteksi keberadaan orang dan mengelola kondisi lingkungan. Referensi lain juga menempatkan **smart building** sebagai area penting untuk **environment health and safety, security, operating efficiency**, dan pengelolaan energi.

Rumusan masalah

Bagaimana membangun sistem pemantauan ruangan yang:

- membaca suhu, kelembaban, dan CO_2,
- mengklasifikasikan kondisi ruangan menjadi **nyaman** atau **tidak nyaman**,
- dan menampilkan hasilnya secara real-time?

14.A.2 Arsitektur

Arsitektur minimal yang realistis:

> **sensor / simulator → MQTT → Python subscriber → preprocessing → model klasifikasi → storage → Grafana dashboard**

Komponen:

- **sensor nyata** atau **dummy data**
- **Mosquitto MQTT**
- **Python** untuk parsing dan inferensi
- **SQLite / InfluxDB**
- **Grafana**

Arsitektur ini sangat konsisten dengan arsitektur open source minimal yang Anda tetapkan sejak awal buku: ESP32/simulator, Mosquitto, Python, InfluxDB/SQLite, dan Grafana.

14.A.3 Dataset

Opsi 1 — Data sensor nyata

Sensor yang dipakai:

- suhu
- kelembaban
- CO_2

Data bisa berasal dari:

- ESP32 dengan sensor lingkungan
- Raspberry Pi + sensor
- simulator publisher MQTT

Opsi 2 — Dummy data / synthetic data

Bila sensor nyata belum tersedia, gunakan dataset sintetis ruangan seperti yang sudah dibangun pada bab-bab sebelumnya:

- pola suhu harian
- kelembaban yang cenderung berlawanan arah dengan suhu
- CO_2 naik saat aktivitas meningkat
- noise kecil dan beberapa kondisi warning/fault buatan

14.A.4 Synthetic data fallback

Fallback sintetis sangat penting karena:

- tidak semua pembaca punya sensor CO_2,
- tidak semua lab siap dengan perangkat nyata,
- dan pembelajaran harus bisa dimulai tanpa menunggu hardware lengkap.

Dataset fallback yang baik minimal memiliki:

- `timestamp`
- `device_id`
- `temperature_c`
- `humidity_pct`
- `co2_ppm`
- label `nyaman / tidak_nyaman`

Kapan fallback sintetis dipakai?

- saat awal pembelajaran
- saat pengujian dashboard
- saat menguji MQTT → model → dashboard
- saat ingin menyuntikkan kondisi ekstrem yang jarang

14.A.5 Preprocessing

Langkah preprocessing yang disarankan:

Langkah 1 — Validasi payload

Periksa:

- field wajib ada
- numerik valid
- rentang wajar

Langkah 2 — Parsing waktu

- ubah `timestamp` ke datetime
- urutkan data

Langkah 3 — Buat fitur ringan

Misalnya:

- `hour`
- `is_weekend`
- `temp_lag_1`
- `co2_lag_1`
- `temp_ma_3`
- `co2_ma_3`

Langkah 4 — Buat label

Contoh aturan:

- `nyaman` jika suhu 24–28°C dan kelembaban 45–65% dan CO_2 < 1000
- selain itu → `tidak_nyaman`

Langkah 5 — Split data time-aware

Pisahkan:

- train
- validation
- test

Ini sangat penting karena sistem ruangan adalah *time series*.

14.A.6 Model

Model yang paling realistis untuk studi kasus ini:

Model baseline

- **Logistic Regression**

Model utama

- **Random Forest Classifier**

  Kenapa?

  - data tabular
  - fitur mudah dibangun
  - output kategorikal
  - feature importance mudah dijelaskan

Target

- `nyaman`
- `tidak_nyaman`

Model ini sesuai dengan semangat buku: ringan, *open source*, cukup kuat, dan mudah dijelaskan.

14.A.7 Evaluasi

Metrik yang disarankan:

- accuracy
- precision
- recall
- F1
- confusion matrix

Fokus interpretasi

- apakah `tidak_nyaman` sering lolos sebagai `nyaman`?
- apakah alarm terlalu sensitif?
- fitur apa yang paling dominan?

14.A.8 Dashboard real-time

Dashboard Grafana minimal berisi:

- suhu real-time
- kelembaban real-time

- CO_2 real-time
- status ruangan
- histori status
- panel warning/alert

14.A.9 Deployment mini

Deployment mini yang realistis:

- simulator atau ESP32 mengirim data ke MQTT
- Python subscriber di laptop atau Raspberry Pi membaca data
- model memprediksi status
- hasil ditulis ke InfluxDB
- Grafana menampilkan kondisi ruangan

Nilai praktis

Pembaca bisa menggunakannya untuk:

- ruang kelas
- ruang kerja
- lab komputer
- ruang server kecil

Studi Kasus B — Predictive Maintenance Sederhana

14.B.1 Masalah

Dalam banyak sistem mekanik atau elektro-mekanik, kerusakan jarang muncul secara tiba-tiba tanpa tanda. Biasanya ada gejala awal:

- getaran meningkat,
- arus motor berubah,
- suhu motor naik,
- suara makin kasar,
- pola operasi berubah.

Referensi Perros menempatkan **asset maintenance: predictive maintenance to maximize reliability** sebagai salah satu aplikasi utama IoT pada manufaktur. Referensi *Internet of Things and Data Analytics Handbook* juga menempatkan **predictive maintenance** dan **equipment optimization** sebagai fungsi inti di smart manufacturing dan industrial IoT.

Rumusan masalah

Bagaimana membuat sistem sederhana yang:

- membaca data getaran dan/atau arus motor,
- mendeteksi kondisi abnormal,
- memberi klasifikasi fault dasar,
- dan membantu interpretasi untuk maintenance?

14.B.2 Arsitektur

Arsitektur minimal:

> **sensor getaran / arus → MQTT / CSV → Python preprocessing → anomaly detection + klasifikasi dasar → storage → dashboard / log maintenance**

Komponen:

- data getaran atau arus
- Python subscriber atau loader CSV
- model anomaly detection
- model klasifikasi fault dasar
- dashboard/terminal/log

14.B.3 Dataset

Opsi 1 — Data nyata

- sensor getaran dari motor/pompa/kipas
- arus listrik motor
- suhu motor jika tersedia

Opsi 2 — Dataset sintetis

Bila alat belum ada:

- sinyal getaran sehat vs fault
- spike sesaat
- amplitudo meningkat
- noise lebih besar
- arus yang mulai naik saat fault

Data getaran / arus motor digunakan untuk studi kasus ini.

14.B.4 Synthetic data fallback

Fallback sintetis untuk predictive maintenance sangat penting karena:

- data fault nyata sering sangat sedikit,
- mesin rusak tidak mudah “disengaja” hanya untuk latihan,
- banyak pembaca belum punya sensor industri nyata.

Synthetic data fallback sebaiknya mencakup:

- sinyal normal
- sinyal warning
- sinyal fault
- injeksi spike
- injeksi drift kecil
- label `normal / warning / fault`

14.B.5 Preprocessing

Langkah preprocessing yang disarankan:

Langkah 1 — Windowing

Bagi sinyal menjadi jendela kecil.

Langkah 2 — Ekstraksi fitur

Fitur sederhana yang sangat berguna:

- mean

- std
- min
- max
- RMS
- peak-to-peak
- kurtosis sederhana
- arus rata-rata
- arus maksimum

Langkah 3 — Validasi sinyal

Periksa:

- missing value
- jendela kosong
- saturasi nilai
- outlier sangat ekstrem

Langkah 4 — Label atau skor

Bila ada label:

- gunakan klasifikasi

Bila tidak ada label cukup:

- mulai dari anomaly detection

14.B.6 Model

Pendekatan yang sangat sehat untuk studi kasus ini adalah **dua lapis**:

Lapis 1 — anomaly detection

Gunakan:

- threshold RMS sederhana
- Isolation Forest

Lapis 2 — klasifikasi fault dasar

Jika label tersedia, gunakan:

- Logistic Regression
- Random Forest Classifier

Kenapa kombinasi ini bagus?

Karena di lapangan:

- data abnormal biasanya lebih mudah dicari dulu,
- lalu fault dasar bisa diklasifikasi bila data dan label bertambah.

Anomaly detection + klasifikasi fault dasar digunakan dalam studi kasus ini.

14.B.7 Evaluasi

Untuk anomaly detection:

- visualisasi sinyal
- confusion matrix bila ground truth sintetis tersedia
- precision / recall terhadap anomali sintetis

Untuk klasifikasi:

- accuracy
- recall untuk `fault`
- confusion matrix

Fokus penting

Pada maintenance, **recall fault** sering lebih penting daripada accuracy semata.

14.B.8 Interpretasi hasil

Interpretasi sederhana yang baik:

- “RMS getaran meningkat stabil selama 2 jam terakhir.”
- “Fault dasar dipicu terutama oleh kenaikan RMS dan peak-to-peak.”
- “Arus dan getaran naik bersamaan, indikasi beban mekanik abnormal.”
- “Status warning lebih sering muncul di jam operasi penuh.”

Explain Ability ringan yang cocok

- feature importance
- grafik fitur per jendela
- tabel status dan waktu kejadian

14.B.9 Deployment mini

Deployment mini yang realistis:

- data dari CSV / simulator / sensor sederhana
- Python membaca data secara batch atau semi-real-time
- menghitung fitur jendela
- memberi skor anomali
- menulis hasil ke log/InfluxDB
- dashboard menampilkan histori warning dan fault

Nilai praktis

Pembaca bisa mencoba ini pada:

- kipas
- motor kecil
- pompa air
- blower
- alat lab

Studi Kasus C — Smart Energy Analytics

14.C.1 Masalah

Salah satu domain IoT yang paling jelas manfaatnya adalah **energi**. Referensi menempatkan **smart grid**, **energy management**, dan **real-time energy optimization** sebagai area utama IoT. Dalam konteks bangunan, laboratorium, atau fasilitas kecil, pertanyaan utamanya sering berupa:

- kapan beban listrik meningkat?
- berapa prediksi beban berikutnya?
- apakah ada konsumsi tidak normal?
- apakah ada pemborosan di luar jam kerja?

Rumusan masalah

Bagaimana membangun sistem yang:

- membaca data konsumsi energi,
- memprediksi beban,
- dan memberi alert saat pola beban tampak tidak normal?

14.C.2 Arsitektur

Arsitektur minimal:

> **meter energi / simulator → MQTT / CSV → Python preprocessing → forecasting + anomaly/threshold alert → storage → dashboard**

Komponen:

- data daya / arus / tegangan
- preprocessing waktu
- model forecasting
- rule/alert beban tidak normal
- dashboard energi

14.C.3 Dataset

Opsi 1 — Data nyata

- smart meter
- meter daya lokal
- logger energi

Opsi 2 — Dataset sintetis

Bila belum ada perangkat:

- data energi 30 hari
- pola jam kerja
- jam sibuk
- akhir pekan lebih rendah
- beberapa lonjakan abnormal

14.C.4 Synthetic data fallback

Fallback sintetis sangat membantu karena:

- smart meter nyata belum selalu tersedia,
- pola jam kerja bisa disimulasikan dengan cukup realistis,
- lonjakan abnormal bisa disuntik untuk menguji alert.

Dataset fallback sebaiknya berisi:

- `timestamp`
- `power_watt`
- `current_amp` opsional
- `voltage` opsional
- label atau penanda anomali sintetis jika ingin evaluasi mudah

14.C.5 Preprocessing

Langkah preprocessing yang disarankan:

Langkah 1 — Parsing waktu

- urutkan timestamp
- cek interval

Langkah 2 — Buat fitur waktu

- `hour`
- `day_of_week`
- `is_weekend`

- `hour_sin`, `hour_cos`

Langkah 3 — Buat fitur historis

- `power_lag_1`
- `power_lag_6`
- `power_ma_3`
- `power_ma_6`
- `power_std_6`
- `power_diff_1`

Langkah 4 — Tentukan target

Untuk forecasting:

- `power_watt_next_1step`

Untuk alert:

- threshold atau skor anomali

14.C.6 Model

Pendekatan utama untuk studi kasus ini:

Forecasting beban

- baseline: last value
- baseline: moving average
- Linear Regression berbasis lag
- Random Forest Regressor

Alert beban tidak normal

- threshold statis
- Isolation Forest opsional
- rule berbasis forecast error bila ingin lanjut

Kenapa pendekatan ini bagus?
Karena energi sangat cocok untuk kombinasi:

- prediksi jangka pendek
- rule operasional yang transparan

14.C.7 Evaluasi

Untuk forecasting:

- MAE
- RMSE
- plot aktual vs prediksi

Untuk alert:

- precision / recall bila ada anomali sintetis
- atau evaluasi operasional berbasis visual dan rule

Hal penting yang perlu dilihat

- apakah model menangkap jam sibuk?
- apakah akhir pekan terlihat berbeda?
- apakah alert terlalu sering palsu?
- apakah model masih stabil saat distribusi beban bergeser?

14.C.8 Alert beban tidak normal

Strategi paling realistis:

- **threshold** untuk alarm jelas, misalnya `power_watt > batas`
- **forecast-aware alert** jika nilai aktual jauh di atas prediksi
- **hybrid**: threshold + deviasi dari baseline

Contoh narasi alert

- “Beban aktual 35% di atas prediksi 10 menit terakhir.”
- “Lonjakan daya muncul di luar jam operasional.”
- “Pola beban tidak sesuai baseline hari kerja.”

14.C.9 Deployment mini

Deployment mini yang realistis:

- simulator energi mengirim data ke MQTT
- Python subscriber membentuk fitur
- model forecasting memprediksi beban berikutnya
- hasil disimpan ke InfluxDB
- Grafana menampilkan:
    - aktual
    - prediksi
    - alert

Nilai praktis

Studi kasus ini sangat cocok untuk:

- lab komputer
- ruang server
- gedung kecil
- UMKM kecil
- mini smart building

14.10 Benang merah tiga studi kasus

Ketiga studi kasus ini sengaja dipilih karena mewakili tiga pola pemecahan masalah yang sangat penting dalam IoT analytics:

Smart Room Monitoring

Fokus pada:

- **klasifikasi kondisi**
- **dashboard real-time**
- **konteks lingkungan dan kenyamanan**

Predictive Maintenance Sederhana

Fokus pada:

- **anomaly detection**
- **fault dasar**
- **interpretasi untuk maintenance**

Smart Energy Analytics

Fokus pada:

- **forecasting**
- **operasional energi**
- **alert beban tidak normal**

Dengan demikian, pembaca melihat bahwa IoT-ML tidak hanya satu gaya solusi. Kadang masalah diselesaikan dengan:

- klasifikasi,
- kadang dengan anomaly detection,
- kadang dengan forecasting,
- tetapi semuanya tetap memakai pola rekayasa sistem yang sama.

14.11 Langkah kerja yang harus dilakukan pembaca

Agar ketiga studi kasus ini benar-benar operasional, pembaca dapat mengikuti pola berikut untuk masing-masing kasus.

Tahap 1 — Definisikan masalah

Tentukan:

- apa yang ingin diprediksi
- apa yang ingin dideteksi
- siapa pengguna hasilnya

Tahap 2 — Susun arsitektur minimal

Pilih:

- sensor nyata atau simulator
- MQTT / CSV
- Python
- storage
- dashboard

Tahap 3 — Tentukan dataset

Gunakan:

- data nyata jika ada
- synthetic fallback bila belum ada

Tahap 4 — Lakukan preprocessing

Buat:

- validasi
- parsing waktu
- fitur penting

Tahap 5 — Pilih model sesuai masalah

- klasifikasi untuk room monitoring
- anomaly detection + fault dasar untuk maintenance
- forecasting untuk energi

Tahap 6 — Evaluasi

Gunakan metrik yang sesuai dan visualisasi yang jujur.

Tahap 7 — Buat deployment mini

Pastikan solusi benar-benar bisa dijalankan, bukan hanya ada di notebook.

14.12 Sikap profesional dalam menyusun studi kasus

Ada satu pelajaran penting yang perlu ditekankan: **studi kasus yang baik bukan yang paling besar, tetapi yang paling utuh**.

Utuh berarti:

- masalahnya jelas,
- data dan fallback-nya ada,
- preprocessing-nya masuk akal,
- modelnya cocok,
- evaluasinya jujur,
- dan deployment mininya benar-benar bisa dijalankan.

Itulah yang membuat studi kasus berguna untuk pembaca dan bisa diulang kembali.

14.13 Penutup bab

Bab 14 adalah titik ketika seluruh isi buku berubah menjadi solusi yang benar-benar terasa nyata. Melalui **Studi Kasus A — Smart Room Monitoring**, pembaca melihat bagaimana data suhu, kelembaban, dan CO_2 dapat diubah menjadi klasifikasi kenyamanan dan dashboard real-time. Melalui **Studi Kasus B — Predictive Maintenance Sederhana**, pembaca memahami bagaimana data getaran dan arus motor dapat dipakai untuk anomaly detection, klasifikasi fault dasar, dan interpretasi maintenance. Melalui **Studi Kasus C — Smart Energy Analytics**, pembaca melihat bagaimana data konsumsi energi dapat dipakai untuk forecasting beban dan alert operasional. Ketiganya disusun dengan pola yang sama—**masalah, arsitektur, dataset, synthetic fallback, preprocessing, model, evaluasi, deployment mini**—sehingga pembaca bukan hanya melihat contoh, tetapi juga belajar kerangka berpikir yang bisa diulang untuk masalah lain.

Hands-on 35 - Studi Kasus A — Smart Room Monitoring lengkap

1. Masalah

Tujuan studi kasus ini adalah membangun sistem yang dapat:

- membaca **suhu**, **kelembaban**, dan **CO_2**
- mengklasifikasikan kondisi ruangan menjadi **nyaman** atau **tidak nyaman**
- menampilkan hasil pada **dashboard real-time**

Kasus ini sangat realistis untuk:

- ruang kelas
- laboratorium komputer
- ruang kerja
- ruang server kecil
- greenhouse ringan

Referensi juga menunjukkan bahwa pada domain **smart home** dan **smart building**, sensor lingkungan seperti suhu, kelembaban, dan CO_2 sangat relevan untuk **occupancy-aware control**, efisiensi operasional, serta kesehatan dan keselamatan lingkungan.

2. Arsitektur

Arsitektur minimal yang dipakai:

> **sensor / simulator → MQTT → Python subscriber → preprocessing → model klasifikasi → InfluxDB → Grafana**

Komponen:

- **simulator Python** atau sensor nyata
- **Mosquitto MQTT**
- **Python subscriber** untuk parsing, preprocessing, inferensi
- **InfluxDB** untuk penyimpanan *time series*
- **Grafana** untuk dashboard

Arsitektur ini konsisten dengan stack open source buku ini: ESP32/simulator, Mosquitto, Python, InfluxDB/SQLite, dan Grafana.

3. Dataset

Opsi data nyata

Payload minimal:

- `timestamp`
- `device_id`
- `temperature_c`
- `humidity_pct`
- `co2_ppm`

Opsi synthetic fallback

Jika sensor belum ada, gunakan data sintetis ruangan dengan pola:

- suhu harian
- kelembaban berlawanan arah ringan dengan suhu
- CO_2 meningkat pada jam sibuk
- beberapa kondisi ekstrem untuk label `tidak_nyaman`

Fallback sintetis ini penting karena studi kasus harus tetap bisa dijalankan walau perangkat belum lengkap, sesuai arah buku yang menekankan praktik yang *repeatable*.

4. Preprocessing

Langkah preprocessing yang dipakai:

Langkah 1 — validasi payload

Periksa:

- field wajib ada
- nilai numerik valid
- rentang wajar

Contoh rentang wajar:

- suhu: 10–45°C
- kelembaban: 0–100%
- CO_2: > 0

Langkah 2 — parsing waktu

- ubah `timestamp` ke datetime
- urutkan data

Langkah 3 — buat fitur

Fitur minimal:

- `temperature_c`
- `humidity_pct`
- `co2_ppm`
- `hour`
- `is_weekend`
- `temp_lag_1`
- `co2_lag_1`
- `temp_ma_3`
- `co2_ma_3`

Langkah 4 — buat label

Aturan sederhana:

- `nyaman` jika suhu 24–28°C, kelembaban 45–65%, CO_2 < 1000
- selain itu: `tidak_nyaman`

5. Model

Model yang paling realistis:

- **Logistic Regression** sebagai baseline
- **Random Forest Classifier** sebagai model utama

Kenapa Random Forest?

- cocok untuk data tabular
- cukup kuat untuk hubungan non-linear ringan
- feature importance mudah dijelaskan

6. Evaluasi

Metrik yang dipakai:

- accuracy
- precision
- recall
- F1
- confusion matrix

Fokus utama:

- apakah `tidak_nyaman` sering lolos sebagai `nyaman`
- apakah model terlalu sensitif memberi `tidak_nyaman`

7. Dashboard real-time

Panel Grafana yang disarankan:

- suhu real-time
- kelembaban real-time
- CO_2 real-time
- status ruangan (`nyaman/tidak_nyaman`)
- histori status
- panel alert sederhana

8. Deployment mini

8.1 Simulator publisher MQTT

Buat file `scripts/case_a_room_publisher.py`:

```
import json
import random
import time
from datetime import datetime
import paho.mqtt.client as mqtt

BROKER = "localhost"
TOPIC = "casea/room01/telemetry"

client = mqtt.Client(client_id="casea-room-pub")
client.connect(BROKER, 1883, 60)

while True:
    payload = {
        "timestamp":
datetime.now().isoformat(timespec="seconds"),
        "device_id": "room01",
        "temperature_c": round(random.uniform(23.0, 33.0), 2),
        "humidity_pct": round(random.uniform(40.0, 72.0), 2),
        "co2_ppm": random.randint(450, 1500)
    }
    client.publish(TOPIC, json.dumps(payload), qos=1)
    print("Published:", payload)
    time.sleep(2)
```

8.2 Subscriber + klasifikasi

Buat file `scripts/case_a_room_subscriber.py`:

```
import json
from collections import deque
from pathlib import Path

import joblib
import pandas as pd
import paho.mqtt.client as mqtt
from influxdb_client import InfluxDBClient, Point, WritePrecision
from influxdb_client.client.write_api import SYNCHRONOUS

BROKER = "localhost"
TOPIC = "casea/room01/telemetry"

MODEL_PATH = Path("models/case_a_room_rf.joblib")

INFLUX_URL = "http://localhost:8086"
INFLUX_TOKEN = "GANTI_DENGAN_TOKEN_ANDA"
INFLUX_ORG = "iotlab"
INFLUX_BUCKET = "sensor_data"

temp_buffer = deque(maxlen=3)
co2_buffer = deque(maxlen=3)

model = joblib.load(MODEL_PATH)

client_db = InfluxDBClient(url=INFLUX_URL, token=INFLUX_TOKEN,
org=INFLUX_ORG)
write_api = client_db.write_api(write_options=SYNCHRONOUS)

def safe_mean(values):
    return sum(values) / len(values) if len(values) > 0 else None

def build_features(payload):
    ts = pd.to_datetime(payload["timestamp"])
    temp = float(payload["temperature_c"])
    hum = float(payload["humidity_pct"])
    co2 = float(payload["co2_ppm"])

    temp_buffer.append(temp)
    co2_buffer.append(co2)

    return pd.DataFrame([{
        "temperature_c": temp,
        "humidity_pct": hum,
        "co2_ppm": co2,
        "hour": ts.hour,
        "is_weekend": int(ts.dayofweek >= 5),
        "temp_lag_1": temp_buffer[-2] if len(temp_buffer) >= 2 else
temp,
        "co2_lag_1": co2_buffer[-2] if len(co2_buffer) >= 2 else co2,
        "temp_ma_3": safe_mean(temp_buffer),
        "co2_ma_3": safe_mean(co2_buffer),
    }])
```

```
def on_connect(client, userdata, flags, reason_code, properties=None):
    client.subscribe(TOPIC, qos=1)
    print("Subscribe ke", TOPIC)

def on_message(client, userdata, msg):
    payload = json.loads(msg.payload.decode())
    X = build_features(payload)
    pred = model.predict(X)[0]

    print(f"[{payload['timestamp']}] -> status={pred}")

    point = (
        Point("case_a_room")
        .tag("device_id", payload["device_id"])
        .tag("pred_status", str(pred))
        .field("temperature_c", float(payload["temperature_c"]))
        .field("humidity_pct", float(payload["humidity_pct"]))
        .field("co2_ppm", float(payload["co2_ppm"]))
        .time(payload["timestamp"], WritePrecision.S)
    )
    write_api.write(bucket=INFLUX_BUCKET, org=INFLUX_ORG,
record=point)

client = mqtt.Client(client_id="casea-room-sub")
client.on_connect = on_connect
client.on_message = on_message
client.connect(BROKER, 1883, 60)
client.loop_forever()
```

Nilai praktis

Studi kasus ini cocok sebagai proyek mini untuk **smart classroom**, **lab monitoring**, dan **ruang server ringan**.

Hands-on 36 - Studi Kasus B — Predictive Maintenance Sederhana lengkap

1. Masalah

Tujuan studi kasus ini adalah membangun sistem sederhana yang:

- membaca **data getaran** dan/atau **arus motor**
- mendeteksi **anomali**
- memberi **klasifikasi fault dasar**
- membantu **interpretasi hasil** untuk maintenance

Referensi menempatkan **predictive maintenance** sebagai salah satu aplikasi paling penting pada smart manufacturing dan asset management.

2. Arsitektur

Arsitektur minimal:

CSV / simulator getaran-arus → Python preprocessing → anomaly detection + fault classifier → storage → dashboard / log

Jika ingin versi IoT penuh:

sensor → MQTT → subscriber Python → fitur jendela → model → storage → dashboard

3. Dataset

Data yang dipakai

- `sample_id`
- `vibration`
- `current_amp`
- `status_label` opsional

Synthetic fallback

Karena data fault nyata jarang, fallback sintetis sangat penting:

- normal: amplitudo stabil
- warning: amplitudo naik perlahan
- fault: amplitudo naik + spike + arus meningkat

4. Preprocessing

Langkah 1 — windowing

Bagi sinyal menjadi jendela kecil, misalnya 50 sampel.

Langkah 2 — ekstraksi fitur

Fitur yang dipakai:

- mean
- std
- RMS
- peak-to-peak
- kurtosis sederhana
- current_mean
- current_max

Langkah 3 — validasi

- tidak ada NaN
- tidak ada jendela kosong
- nilai tidak saturasi ekstrem tanpa alasan

5. Model

Pendekatan dua lapis:

Lapis 1 — anomaly detection

- threshold RMS sederhana
- **Isolation Forest**

Lapis 2 — klasifikasi fault dasar

- **Random Forest Classifier**
- kelas: `normal / warning / fault`

6. Evaluasi

Untuk anomaly detection:

- confusion matrix bila ground truth sintetis ada
- precision / recall terhadap anomali

Untuk klasifikasi:

- accuracy
- recall untuk `fault`
- confusion matrix

7. Interpretasi hasil

Interpretasi yang baik harus menjawab:

- apakah RMS naik?
- apakah current_mean ikut naik?
- apakah spike makin sering?
- fitur mana yang paling berpengaruh?

8. Deployment mini

8.1 Generator dataset sintetis getaran + arus

Buat file `scripts/case_b_generate_motor_data.py`:

```
from pathlib import Path
import numpy as np
import pandas as pd

OUTPUT = Path("data/synthetic/case_b_motor.csv")
OUTPUT.parent.mkdir(parents=True, exist_ok=True)

rng = np.random.default_rng(42)
n = 3000
t = np.arange(n)

vibration = 0.8 * np.sin(2 * np.pi * 0.03 * t) + rng.normal(0,
0.08, n)
current = rng.normal(5.0, 0.2, n)

status = np.array(["normal"] * n, dtype=object)

status[1000:1800] = "warning"
vibration[1000:1800] += 0.4 * np.sin(2 * np.pi * 0.03 *
t[1000:1800])
current[1000:1800] += rng.normal(0.6, 0.1, 800)

status[1800:2600] = "fault"
vibration[1800:2600] += 1.0 * np.sin(2 * np.pi * 0.03 *
t[1800:2600])
current[1800:2600] += rng.normal(1.5, 0.2, 800)

df = pd.DataFrame({
    "sample_id": np.arange(n),
    "vibration": np.round(vibration, 5),
    "current_amp": np.round(current, 3),
    "status_label": status
})

df.to_csv(OUTPUT, index=False)
print("Disimpan ke:", OUTPUT)
```

8.2 Ekstraksi fitur per jendela

Buat file `scripts/case_b_feature_window.py`:

```
from pathlib import Path
import numpy as np
import pandas as pd

INPUT = Path("data/synthetic/case_b_motor.csv")
OUTPUT = Path("data/processed/case_b_motor_features.csv")
OUTPUT.parent.mkdir(parents=True, exist_ok=True)

df = pd.read_csv(INPUT)

window_size = 50
step = 25
rows = []

for start in range(0, len(df) - window_size + 1, step):
    end = start + window_size
    w = df.iloc[start:end]

    vib = w["vibration"].values
    cur = w["current_amp"].values

    row = {
        "window_start": int(w["sample_id"].iloc[0]),
        "window_end": int(w["sample_id"].iloc[-1]),
        "vib_mean": np.mean(vib),
        "vib_std": np.std(vib),
        "vib_rms": np.sqrt(np.mean(vib ** 2)),
        "vib_ptp": np.ptp(vib),
        "cur_mean": np.mean(cur),
        "cur_max": np.max(cur),
        "status_label": w["status_label"].mode()[0]
    }
    rows.append(row)

feat_df = pd.DataFrame(rows)
feat_df.to_csv(OUTPUT, index=False)
print("Disimpan ke:", OUTPUT)
```

8.3 Anomaly detection + klasifikasi

Buat file `scripts/case_b_detect_and_classify.py`:

```
from pathlib import Path
import pandas as pd
from sklearn.ensemble import IsolationForest,
RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,
confusion_matrix

INPUT = Path("data/processed/case_b_motor_features.csv")
df = pd.read_csv(INPUT)

feature_cols = ["vib_mean", "vib_std", "vib_rms", "vib_ptp",
"cur_mean", "cur_max"]
X = df[feature_cols]
y = df["status_label"]

# Isolation Forest
iso = IsolationForest(n_estimators=200, contamination=0.15,
random_state=42)
df["anomaly_pred"] = (iso.fit_predict(X) == -1).astype(int)

print("=== Isolation Forest ===")
print(df["anomaly_pred"].value_counts())

# Klasifikasi fault dasar
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, shuffle=False
)

clf = RandomForestClassifier(n_estimators=200, max_depth=8,
random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print("\n=== Fault Classification ===")
print(confusion_matrix(y_test, y_pred, labels=["normal",
"warning", "fault"]))
print(classification_report(y_test, y_pred, zero_division=0))
```

Nilai praktis

Studi kasus ini cocok untuk **motor kecil**, **pompa**, **blower**, dan **alat lab**.

Hands-on 37 - Studi Kasus C — Smart Energy Analytics lengkap

1. Masalah

Tujuan studi kasus ini adalah membangun sistem yang:

- membaca **data konsumsi energi**
- melakukan **forecasting beban**
- memberi **alert beban tidak normal**

Referensi menempatkan **smart grid**, **smart metering**, dan **real-time energy optimization** sebagai domain IoT yang sangat penting.

2. Arsitektur

Arsitektur minimal:

meter energi / simulator → MQTT / CSV → Python preprocessing → forecasting → alert → InfluxDB → Grafana

3. Dataset

Opsi data nyata

- smart meter
- logger energi lokal
- pembacaan daya/arus/tegangan

Opsi synthetic fallback

- pola beban jam kerja
- beban malam rendah
- akhir pekan lebih rendah
- lonjakan abnormal disuntik

4. Preprocessing

Langkah 1 — parsing waktu

- urutkan timestamp
- cek interval konsisten

Langkah 2 — buat fitur waktu

- `hour`
- `day_of_week`
- `is_weekend`
- `hour_sin`
- `hour_cos`

Langkah 3 — buat fitur historis

- `power_lag_1`
- `power_lag_6`
- `power_ma_3`
- `power_ma_6`
- `power_std_6`

Langkah 4 — target forecasting

- `power_watt_next_1step`

5. Model

Forecasting

- baseline: **last value**
- baseline: **moving average**
- **Linear Regression**
- **Random Forest Regressor**

Alert beban tidak normal

- threshold statis
- deviasi dari prediksi
- kombinasi keduanya

6. Evaluasi

Forecasting:

- MAE
- RMSE
- plot aktual vs prediksi

Alert:

- precision / recall bila ada anomali sintetis
- atau verifikasi visual operasional

7. Deployment mini

7.1 Generator energi sintetis

Buat file `scripts/case_c_generate_energy.py`:

```
from pathlib import Path
import numpy as np
import pandas as pd

OUTPUT = Path("data/synthetic/case_c_energy.csv")
OUTPUT.parent.mkdir(parents=True, exist_ok=True)

rng = np.random.default_rng(123)
timestamps = pd.date_range(
    start="2026-01-01 00:00:00",
    end="2026-01-30 23:50:00",
    freq="10min"
)

records = []
for ts in timestamps:
    hour = ts.hour
    dow = ts.dayofweek
    is_weekend = dow >= 5

    if is_weekend:
        base = 110 if hour < 7 else 150 if hour < 18 else 120
    else:
        if hour < 6:
            base = 110
        elif hour < 8:
            base = 180
        elif hour < 12:
            base = 320
        elif hour < 14:
            base = 430
        elif hour < 18:
            base = 300
        else:
            base = 160
```

```
        power = base + rng.normal(0, 12)
        records.append({
            "timestamp": ts,
            "device_id": "energy01",
            "power_watt": round(power, 2)
        })

df = pd.DataFrame(records)

# inject beberapa anomali
anomaly_idx = rng.choice(df.index, size=20, replace=False)
df["is_anomaly"] = 0
df.loc[anomaly_idx, "power_watt"] += rng.uniform(120, 220,
len(anomaly_idx))
df.loc[anomaly_idx, "is_anomaly"] = 1

df.to_csv(OUTPUT, index=False)
print("Disimpan ke:", OUTPUT)
```

7.2 Forecasting + alert sederhana

Buat file `scripts/case_c_energy_forecast_alert.py`:

```
from pathlib import Path
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error

INPUT = Path("data/synthetic/case_c_energy.csv")
df = pd.read_csv(INPUT)
df["timestamp"] = pd.to_datetime(df["timestamp"])
df = df.sort_values("timestamp").reset_index(drop=True)

df["target_next"] = df["power_watt"].shift(-1)
df["hour"] = df["timestamp"].dt.hour
df["day_of_week"] = df["timestamp"].dt.dayofweek
df["is_weekend"] = (df["day_of_week"] >= 5).astype(int)

df["power_lag_1"] = df["power_watt"].shift(1)
df["power_lag_6"] = df["power_watt"].shift(6)
df["power_ma_3"] = df["power_watt"].rolling(3).mean()
df["power_ma_6"] = df["power_watt"].rolling(6).mean()
df["power_std_6"] = df["power_watt"].rolling(6).std()

df = df.dropna().reset_index(drop=True)

feature_cols = [
    "power_watt", "hour", "day_of_week", "is_weekend",
    "power_lag_1", "power_lag_6", "power_ma_3", "power_ma_6",
"power_std_6"
]

n = len(df)
train_end = int(n * 0.70)
X_train = df.iloc[:train_end][feature_cols]
y_train = df.iloc[:train_end]["target_next"]
X_test = df.iloc[train_end:][feature_cols]
```

```
y_test = df.iloc[train_end:]["target_next"]

model = RandomForestRegressor(n_estimators=200, max_depth=10,
random_state=42, n_jobs=-1)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print("MAE :", mae)
print("RMSE:", rmse)

result = df.iloc[train_end:].copy()
result["pred_next"] = y_pred
result["forecast_error_abs"] = (result["target_next"] -
result["pred_next"]).abs()

# Alert beban tak normal
result["alert_abnormal_load"] = (
    (result["power_watt"] > 500) |
    (result["forecast_error_abs"] > 80)
).astype(int)

print(result[["timestamp", "power_watt", "pred_next",
"forecast_error_abs", "alert_abnormal_load"]].head(20))
result.to_csv("outputs/reports/case_c_energy_results.csv",
index=False)
```

7.3 Dashboard mini

Grafana panel yang disarankan:

- power aktual
- prediksi beban
- forecast error
- jumlah alert beban tidak normal

Nilai praktis

Studi kasus ini cocok untuk:

- lab komputer
- ruang server
- gedung kecil
- UMKM
- mini smart building

Penutup

Hands-on 35–37 adalah puncak aplikatif dari seluruh buku. Pada **Hands-on 35**, pembaca membangun **Smart Room Monitoring** lengkap dari data suhu,

kelembaban, CO_2 menuju klasifikasi kenyamanan dan dashboard real-time. Pada **Hands-on 36**, pembaca membangun **Predictive Maintenance Sederhana** dari data getaran/arus menuju anomaly detection, fault dasar, dan interpretasi yang berguna untuk maintenance. Pada **Hands-on 37**, pembaca membangun **Smart Energy Analytics** dari data konsumsi energi menuju forecasting beban dan alert beban tidak normal.

Bab 15. Proyek Akhir Buku: Sistem IoT-ML Open Source Lengkap

15.1 Tujuan proyek akhir

Proyek akhir ini dirancang sebagai **capstone project** yang merangkum seluruh isi buku menjadi satu sistem yang utuh, sederhana, dan realistis. Tujuan akhirnya adalah membangun sistem yang dapat:

- menerima data sensor atau simulasi sensor,
- mengalirkan data melalui **MQTT**,
- memproses data secara otomatis di **Python**,
- menjalankan **model anomaly detection**,
- menyimpan data dan hasil inferensi,
- menampilkan hasil ke **dashboard Grafana**,
- dan mengirim **notifikasi sederhana** bila kondisi dianggap tidak normal.

Dengan kata lain, ini bukan lagi latihan lepas, tetapi **sistem IoT-ML mini yang hidup**.

15.2 Mengapa proyek ini penting?

Setelah pembaca menyelesaikan bab-bab sebelumnya, mereka sudah punya hampir semua blok bangunan:

- pengiriman data sensor,
- MQTT,
- penyimpanan,
- preprocessing,
- supervised learning,
- anomaly detection,
- forecasting,
- edge inference,
- pipeline end-to-end,
- keamanan dan operasional,
- serta studi kasus praktis.

Namun, kemampuan sesungguhnya baru terasa ketika semua blok itu dirangkai menjadi satu sistem yang:

- punya data masuk,
- punya inferensi otomatis,
- punya histori,
- punya tampilan,
- dan punya alarm.

Itulah fungsi proyek akhir ini.

15.3 Masalah yang ingin diselesaikan

Rumusan masalah proyek akhir:

> Bagaimana membangun sistem **monitoring dan deteksi anomali perangkat IoT** berbasis **Python, MQTT, dan dashboard open source** yang ringan, dapat diulang, mudah diuji, dan bisa dikembangkan lebih lanjut?

Masalah ini sangat realistis untuk banyak skenario:

- ruang kelas / lab,
- ruang server,
- pompa / motor kecil,
- smart room,
- smart energy node,
- greenhouse,
- perangkat edge lokal.

Karena itu, proyek akhir ini sengaja dibuat **cukup generik** agar bisa dipakai sebagai template untuk banyak domain.

15.4 Komponen utama proyek

Proyek akhir ini terdiri dari komponen berikut.

15.4.1 Data sensor nyata / simulator / synthetic generator

Sistem harus bisa berjalan meskipun pembaca:

- belum punya sensor nyata,
- baru punya simulator,
- atau ingin menguji kondisi ekstrem.

Karena itu, sumber data dibuat fleksibel:

- **sensor nyata**
- **simulator Python**
- **synthetic generator**

Pendekatan ini juga konsisten dengan pendekatan buku sejak awal, yang selalu menyediakan *fallback* sintetis agar pipeline tetap bisa dibangun dan diuji tanpa tergantung hardware.

15.4.2 Broker MQTT

MQTT broker menjadi tulang punggung distribusi data. Dalam konteks buku ini, pilihan paling realistis adalah **Mosquitto**.

MQTT cocok karena:

- ringan,
- terbuka,
- mudah diimplementasikan,
- memakai pola *publish/subscribe*,
- dan dirancang untuk perangkat dengan keterbatasan sumber daya dan bandwidth. Broker bertugas mengorkestrasi komunikasi antar-klien, sehingga publisher dan subscriber tidak perlu saling mengenal langsung.

15.4.3 Skrip subscriber Python

Skrip ini adalah **jantung sistem**. Ia bertugas untuk:

- menerima payload dari MQTT,
- memvalidasi isi data,
- melakukan preprocessing ringan,
- membentuk fitur,
- memanggil model,
- menyimpan hasil,
- dan bila perlu memicu notifikasi.

Pilihan Python sangat sesuai dengan arah buku yang memprioritaskan praktik cepat, fleksibel, dan mudah direplikasi.

15.4.4 Preprocessing otomatis

Preprocessing otomatis berarti sistem tidak menunggu notebook dibuka setiap saat. Ketika data masuk, sistem langsung:

- memeriksa field wajib,
- memeriksa tipe dan rentang nilai,
- membentuk fitur dasar,
- menangani kondisi awal seperti lag/rolling yang belum penuh,
- lalu meneruskan data ke model.

Hal ini konsisten bahwa data IoT harus melalui **cleaning, preprocessing, feature engineering, dan validasi kualitas data** agar hasil model tidak menyesatkan.

15.4.5 Model anomaly detection

Karena proyek akhir ini fokus pada **deteksi anomali perangkat IoT**, maka model yang paling realistis adalah model **anomaly detection** yang ringan dan mudah dipakai. Untuk buku ini, pendekatan yang sangat cocok adalah:

- **threshold sederhana** sebagai baseline,
- **Isolation Forest** sebagai model utama,
- dengan kemungkinan pengembangan ke model lain bila diperlukan.

Pilihan ini konsisten dengan bagian buku yang sebelumnya membahas bahwa anomaly detection sangat relevan saat label terbatas, dan bahwa pendekatan hybrid antara **threshold** dan **model ML** sering lebih masuk akal dalam sistem nyata.

15.4.6 Dashboard Grafana

Dashboard berfungsi sebagai:

- jendela operasional sistem,
- tampilan histori sensor,
- tampilan skor atau status anomali,
- dan tempat operator membaca kondisi perangkat.

Grafana sangat cocok karena:

- open source,
- kuat untuk *time series*,
- mudah diintegrasikan dengan InfluxDB,
- dan sudah menjadi dashboard standar praktis dalam buku ini.

15.4.7 Notifikasi sederhana

Notifikasi tidak perlu rumit. Untuk proyek akhir ini, notifikasi sederhana bisa berupa:

- publish ke topik MQTT `alert`,
- log khusus ke database,
- panel merah di Grafana,
- atau output terminal yang jelas.

Fokusnya adalah **aksi minimum yang berguna**, bukan sistem notifikasi enterprise.

15.5 Arsitektur proyek akhir

Arsitektur paling sehat untuk proyek akhir ini adalah:

> **sensor / simulator / synthetic generator → MQTT broker → Python subscriber → preprocessing otomatis → model anomaly detection → storage → dashboard / notifikasi**

Arsitektur ini sangat konsisten penjelasan dalam buku dan dengan pola **device–edge–cloud** yang dijelaskan di referensi: data dikumpulkan dari device, diproses dekat edge saat perlu, lalu disimpan dan divisualisasikan pada lapisan lain untuk analitik dan aksi operasional.

Peran tiap komponen

Publisher

- mengirim data sensor atau simulasi

Broker

- mendistribusikan data ke subscriber yang tepat

Subscriber Python

- memproses, memprediksi, dan memutuskan

Storage

- menyimpan histori mentah dan hasil inferensi

Grafana

- menampilkan data dan status

Alert channel

- memunculkan kejadian penting yang perlu perhatian

15.6 Struktur folder proyek

Agar proyek mudah dirawat, berikut struktur folder yang disarankan:

```
iot-ml-capstone/
├── data/
│   ├── raw/
│   ├── processed/
│   └── models/
├── scripts/
│   ├── publishers/
│   ├── subscribers/
│   ├── preprocessing/
│   ├── inference/
│   └── utils/
├── dashboards/
├── outputs/
│   ├── reports/
│   └── figures/
├── docker/
├── requirements.txt
├── .env.example
└── README.md
```

Penjelasan singkat

data/raw/
Untuk menyimpan dump payload mentah, CSV uji, atau arsip data sebelum diproses.

data/processed/
Untuk dataset hasil pembersihan, feature set, atau agregasi tertentu.

data/models/
Untuk menyimpan model aktif dan versi model.

scripts/publishers/
Untuk simulator atau publisher sensor.

scripts/subscribers/
Untuk subscriber MQTT dan logika pipeline utama.

scripts/preprocessing/
Untuk fungsi validasi, cleaning, dan feature builder.

scripts/inference/
Untuk model loader, scorer, dan rule alert.

dashboards/
Untuk ekspor JSON dashboard Grafana atau catatan query.

docker/
Untuk `docker-compose.yml` atau file container opsional.

Struktur seperti ini membantu pembaca membedakan:

- data,
- logika pipeline,
- model,
- dan artefak visualisasi.

15.7 Dataset dan sumber data

Proyek akhir ini sengaja dirancang fleksibel. Pembaca bisa memakai salah satu dari tiga sumber data.

15.7.1 Sensor nyata

Contoh:

- suhu
- kelembaban
- CO_2
- arus
- getaran
- daya listrik

15.7.2 Simulator

Publisher Python yang mengirim payload realistis berkala ke MQTT.

15.7.3 Synthetic generator

Generator yang bisa:

- meniru pola normal,
- menyuntik noise,
- menyuntik drift,
- menyuntik anomali,
- dan dipakai untuk stress testing.

Pendekatan ini sangat berguna karena proyek akhir harus tetap dapat dijalankan walau perangkat keras belum lengkap. Itu juga sejalan pola yang digunakan dalam buku, yang menggunakan komponen utama **data sensor nyata / simulator / synthetic generator**.

15.8 Preprocessing otomatis

Agar sistem benar-benar operasional, preprocessing harus terjadi secara otomatis saat data masuk.

Tahap preprocessing minimum

1. Validasi field wajib

- `timestamp`
- `device_id`
- fitur sensor utama

2. Validasi tipe

- suhu, kelembaban, arus, daya harus numerik

3. Validasi rentang

- nilai negatif atau sangat ekstrim perlu ditandai

4. Parsing waktu

- ubah ke format datetime

5. Bentuk fitur

- lag sederhana
- moving average
- rolling mean
- indikator jam/hari bila perlu

6. Fallback

- jika buffer belum penuh, pakai nilai saat ini atau strategi lain yang konsisten

Kenapa preprocessing otomatis penting?

Karena tanpa itu:

- model menerima data mentah yang belum siap,
- inferensi jadi rapuh,
- dan sistem akan sering gagal saat data lapangan “tidak cantik”.

Referensi juga menekankan bahwa heterogenitas data IoT menuntut **cleansing, ordering, and preprocessing** agar prediksi tidak menjadi salah.

15.9 Model anomaly detection

Untuk proyek akhir ini, model yang paling realistis adalah **Isolation Forest** sebagai model utama, dengan **threshold sederhana** sebagai baseline operasional.

Mengapa threshold tetap dipakai?

Karena:

- transparan,
- mudah dipahami,
- dan bagus untuk alarm cepat.

Mengapa Isolation Forest?

Karena:

- tidak membutuhkan label lengkap,
- cocok untuk data tabular hasil preprocessing,
- ringan,
- dan sangat praktis untuk sistem IoT yang fokus pada deteksi penyimpangan.

Strategi hybrid yang disarankan

Pendekatan terbaik untuk proyek akhir:

- **threshold** untuk alarm yang sangat jelas
- **Isolation Forest** untuk pola yang lebih halus
- keduanya ditampilkan pada dashboard

Ini sejalan dengan pendekatan buku sebelumnya bahwa **threshold vs model ML** tidak harus dipertentangkan, tetapi bisa dipakai bersama.

15.10 Dashboard Grafana

Dashboard proyek akhir minimal sebaiknya memuat:

- sensor utama real-time
- histori sensor utama
- skor atau status anomali
- jumlah alert
- panel status perangkat
- daftar alert terbaru

Panel yang disarankan

Panel 1 — data sensor utama

- suhu / arus / daya / getaran

Panel 2 — skor anomali

- field numerik hasil model atau label

Panel 3 — status perangkat

- `normal`
- `warning`
- `anomaly`

Panel 4 — alert terbaru

- tabel alert paling baru

Panel 5 — histori distribusi

- untuk melihat drift atau perubahan pola

Grafana sangat cocok untuk ini karena sejak awal buku memang diarahkan menjadi dashboard utama untuk *time series* open source.

15.11 Notifikasi sederhana

Untuk tahap buku, notifikasi cukup dibuat sederhana dan jelas.

Opsi notifikasi yang realistis

- publish ke topik MQTT `alert`
- log ke measurement khusus
- panel merah di Grafana
- terminal message
- file log CSV

Contoh aturan notifikasi

- skor anomali > ambang
- suhu dan CO_2 sama-sama tinggi
- arus melonjak di luar baseline
- sensor berhenti kirim data terlalu lama

Prinsip utamanya:
notifikasi harus cukup berguna, bukan terlalu berisik.

15.12 Cara pengujian

15.12.1 Uji aliran dasar

Pastikan:

- publisher bisa kirim ke broker
- subscriber bisa menerima
- payload bisa diparse
- data masuk ke storage

15.12.2 Uji preprocessing

Coba:

- payload normal
- payload field hilang
- payload tipe salah
- payload nilai ekstrim

15.12.3 Uji model

Periksa:

- model bisa dimuat
- inferensi keluar konsisten
- skor anomali masuk akal

15.12.4 Uji dashboard

Periksa:

- data sensor muncul
- histori bergerak
- alert muncul saat dipicu

15.12.5 Uji skenario anomali

Gunakan synthetic generator untuk:

- spike mendadak
- drift perlahan
- nilai sensor palsu
- burst pesan cepat

15.12.6 Uji stabilitas ringan

Biarkan sistem berjalan beberapa waktu, lalu cek:

- error log
- jumlah pesan diterima
- jumlah pesan gagal
- apakah dashboard tetap hidup

15.13 Cara pengembangan lanjut

Berikut arah pengembangan yang sangat masuk akal setelah proyek akhir ini berhasil:

15.13.1 Multi-device

Dari satu perangkat menjadi:

- `room01`, `room02`, `lab01`
- atau banyak node sensor

15.13.2 Multi-sensor fusion

Gabungkan lebih banyak sensor:

- suhu
- kelembaban
- CO_2
- arus
- daya
- getaran

15.13.3 Retraining berkala

Tambahkan script:

- retraining mingguan
- evaluasi drift
- versioning model

15.13.4 Edge deployment

Pindahkan subscriber + inferensi ke:

- Raspberry Pi
- mini PC
- gateway lokal

15.13.5 Alert yang lebih matang

Tambahkan:

- threshold dinamis
- integrasi Telegram / email / WhatsApp

- severity level

15.13.6 Keamanan lebih baik

Tambahkan:

- autentikasi MQTT lebih ketat
- TLS
- segmentasi topik dan hak akses
- manajemen secret yang lebih rapi

15.13.7 Docker untuk reproducibility

Setelah sistem dipahami, bungkus dengan Docker agar:

- setup mudah diulang
- mudah dipindah mesin
- mudah diajarkan ke kelas lain

Docker sebagai opsi, bukan syarat wajib.

15.14 Alur implementasi proyek akhir yang disarankan

Agar pembaca tidak tersesat, berikut alur paling sehat:

Tahap 1 — mulai dari simulator

Gunakan publisher sintetis agar pipeline cepat hidup.

Tahap 2 — hidupkan MQTT dan subscriber

Pastikan aliran data berjalan stabil.

Tahap 3 — tambahkan preprocessing otomatis

Jangan langsung memanggil model sebelum validasi dan fitur dasar siap.

Tahap 4 — tambahkan model anomaly detection

Mulai dari threshold, lalu tambah Isolation Forest.

Tahap 5 — simpan hasil ke storage

Gunakan InfluxDB atau SQLite.

Tahap 6 — buat dashboard Grafana

Tampilkan sensor, skor, dan alert.

Tahap 7 — tambahkan notifikasi sederhana

Gunakan topik MQTT `alert` atau panel khusus.

Tahap 8 — uji kondisi buruk

Coba synthetic drift, spike, dan payload rusak.

Tahap 9 — rapikan struktur proyek

Simpan model, log, dashboard, dan skrip dengan rapi.

15.15 Contoh keluaran proyek akhir yang baik

Sebuah proyek akhir yang berhasil seharusnya menghasilkan:

- publisher/simulator berjalan stabil
- subscriber menerima dan memproses data otomatis
- model anomaly detection aktif
- histori data tersimpan
- dashboard real-time hidup
- alert muncul saat kondisi abnormal
- proyek bisa diulang di mesin lain dengan setup yang jelas

15.16 Sikap profesional dalam proyek akhir

Ada satu pelajaran penting yang ingin ditekankan di bab penutup ini:

> **proyek akhir yang baik bukan proyek yang paling banyak komponennya, tetapi proyek yang paling utuh, stabil, dan bisa dijelaskan.**

Artinya:

- lebih baik satu pipeline kecil tetapi lengkap,
- daripada banyak ide besar tetapi tidak benar-benar terhubung.

Di dunia nyata, sistem yang berguna adalah sistem yang:

- datanya valid,
- modelnya masuk akal,
- dashboard-nya terbaca,
- alert-nya berguna,
- dan mudah dirawat.

15.17 Penutup bab

Bab 15 adalah puncak dari seluruh perjalanan buku ini. Di sini pembaca tidak lagi hanya belajar tentang sensor, data, model, atau dashboard secara terpisah, tetapi benar-benar menyusun sebuah **Sistem IoT-ML Open Source Lengkap** untuk **monitoring dan deteksi anomali perangkat IoT**. Komponen yang ditetapkan—**data sensor nyata/simulator/synthetic generator, broker MQTT, skrip subscriber Python, preprocessing otomatis, model anomaly detection, dashboard Grafana, notifikasi sederhana, struktur folder proyek, cara pengujian, dan cara pengembangan lanjut**—membentuk sebuah capstone project yang sangat realistis, repeatable, dan kuat sebagai hasil akhir buku.

Hands-on 38 - Membangun Proyek Akhir End-to-End

1. Tujuan hands-on

Pada latihan ini, pembaca akan membangun **proyek akhir end-to-end** yang utuh, mulai dari:

- data sensor atau simulator,
- pengiriman ke MQTT,
- subscriber Python,
- preprocessing otomatis,
- inferensi anomaly detection,
- penyimpanan hasil,
- hingga dashboard Grafana.

Setelah selesai, pembaca diharapkan mampu menghasilkan sistem mini yang:

- benar-benar hidup,
- dapat dijalankan berulang,
- dan bisa dikembangkan menjadi proyek lapangan yang lebih serius.

Ini adalah praktik paling penting dalam seluruh buku, karena di sinilah semua blok pengetahuan dari bab-bab sebelumnya benar-benar menyatu. Bab ini ditujukan untuk pembaca yang ingin **hasil nyata, bukan hanya potongan-potongan teori**.

2. Skenario proyek yang dipakai

Agar tetap fleksibel, proyek akhir ini dirancang sebagai **monitoring dan deteksi anomali perangkat IoT**. Data yang dipakai dapat berupa:

- suhu, kelembaban, CO_2,
- arus listrik,
- daya listrik,
- getaran sederhana,
- atau kombinasi sensor yang tersedia.

Untuk menjaga latihan tetap praktis, kita gunakan payload generik berikut:

```
{
  "timestamp": "2026-01-01T10:10:00",
  "device_id": "iotnode01",
  "temperature_c": 28.6,
  "humidity_pct": 54.1,
  "power_watt": 145.3,
  "current_amp": 5.8
}
```

Ini cukup fleksibel untuk:

- smart room monitoring,
- smart energy node,
- node peralatan kecil,
- atau gateway edge kampus/lab.

3. Arsitektur sistem

Arsitektur yang dibangun:

> **sensor / simulator / synthetic generator → MQTT broker → Python subscriber → preprocessing otomatis → model anomaly detection → storage → dashboard Grafana**

Peran tiap komponen

- **Publisher**: mengirim telemetry sensor
- **Broker MQTT**: mendistribusikan pesan
- **Subscriber Python**: validasi, preprocessing, inferensi
- **Storage**: menyimpan data dan hasil model
- **Grafana**: menampilkan histori dan status
- **Alert**: memberi notifikasi saat kondisi abnormal

4. Struktur folder proyek

Buat struktur proyek berikut:

```
iot-ml-capstone/
├── data/
│   ├── raw/
│   ├── processed/
│   └── models/
├── scripts/
│   ├── publishers/
│   ├── subscribers/
│   ├── preprocessing/
│   ├── inference/
│   └── utils/
├── dashboards/
├── outputs/
│   ├── reports/
│   └── figures/
├── docker/
├── requirements.txt
└── README.md
```

Struktur seperti ini konsisten dengan kebutuhan proyek akhir yang rapi, mudah diuji, dan mudah dikembangkan.

5. Persiapan environment Ubuntu 26.04

5.1 Pasang Mosquitto

```
sudo apt update
sudo apt install -y mosquitto mosquitto-clients
sudo systemctl enable mosquitto
sudo systemctl start mosquitto
```

5.2 Jalankan InfluxDB

```
docker run -d \
  --name influxdb2 \
  -p 8086:8086 \
  -v ~/iot-ml-capstone/influxdb-data:/var/lib/influxdb2 \
  influxdb:2
```

5.3 Jalankan Grafana

```
docker run -d \
  --name grafana \
  -p 3000:3000 \
  grafana/grafana-oss
```

5.4 Pasang library Python

```
source .venv/bin/activate
pip install pandas numpy paho-mqtt influxdb-client scikit-learn joblib
matplotlib
```

6. Menyiapkan data: simulator / synthetic generator

Buat file `scripts/publishers/capstone_publisher.py`:

```
import json
import random
import time
from datetime import datetime
import paho.mqtt.client as mqtt

BROKER = "localhost"
PORT = 1883
TOPIC = "capstone/iotnode01/telemetry"

client = mqtt.Client(client_id="capstone-publisher")
client.connect(BROKER, PORT, 60)

while True:
    payload = {
        "timestamp": datetime.now().isoformat(timespec="seconds"),
        "device_id": "iotnode01",
        "temperature_c": round(random.uniform(24.0, 34.0), 2),
        "humidity_pct": round(random.uniform(40.0, 75.0), 2),
        "power_watt": round(random.uniform(80.0, 260.0), 2),
        "current_amp": round(random.uniform(3.5, 9.5), 2)
    }

    # injeksi anomali kecil sesekali
    if random.random() < 0.08:
        payload["power_watt"] += random.uniform(80, 180)
        payload["current_amp"] += random.uniform(2, 5)

    client.publish(TOPIC, json.dumps(payload), qos=1)
    print("Published:", payload)
    time.sleep(2)
```

Jalankan:

```
python scripts/publishers/capstone_publisher.py
```

Mengapa synthetic generator penting?

Karena proyek akhir memang menghendaki sistem tetap bisa berjalan walau pembaca belum memiliki sensor nyata, dan synthetic generator juga sangat berguna untuk pengujian dan stress test.

7. Menyiapkan model anomaly detection

Untuk capstone ini, kita gunakan **Isolation Forest** sebagai model utama, dengan **threshold sederhana** sebagai baseline operasional. Ini sangat sejalan dengan arah buku sebelumnya yang menempatkan anomaly detection sebagai pilihan realistis ketika label lengkap sulit didapat.

Buat file `scripts/inference/train_capstone_iforest.py`:

```
from pathlib import Path
import numpy as np
import pandas as pd
import joblib
from sklearn.ensemble import IsolationForest

MODEL_DIR = Path("data/models")
MODEL_DIR.mkdir(parents=True, exist_ok=True)

# synthetic training baseline
rng = np.random.default_rng(42)
n = 2000
df = pd.DataFrame({
    "temperature_c": rng.normal(27.0, 1.5, n),
    "humidity_pct": rng.normal(55.0, 6.0, n),
    "power_watt": rng.normal(140.0, 20.0, n),
    "current_amp": rng.normal(5.5, 0.8, n),
})

feature_cols = ["temperature_c", "humidity_pct", "power_watt",
"current_amp"]

model = IsolationForest(
    n_estimators=200,
    contamination=0.05,
    random_state=42
)
model.fit(df[feature_cols])

joblib.dump(model, MODEL_DIR / "capstone_iforest.joblib")
print("Model disimpan ke:", MODEL_DIR /
"capstone_iforest.joblib")
```

Jalankan:

```
python scripts/inference/train_capstone_iforest.py
```

8. Subscriber Python + preprocessing otomatis + inferensi + storage

Buat file `scripts/subscribers/capstone_subscriber.py`:

```
import json
from pathlib import Path
import joblib
import pandas as pd
import paho.mqtt.client as mqtt
from influxdb_client import InfluxDBClient, Point,
WritePrecision
from influxdb_client.client.write_api import SYNCHRONOUS

BROKER = "localhost"
PORT = 1883
TOPIC = "capstone/iotnode01/telemetry"

MODEL_PATH = Path("data/models/capstone_iforest.joblib")

INFLUX_URL = "http://localhost:8086"
INFLUX_TOKEN = "GANTI_DENGAN_TOKEN_ANDA"
INFLUX_ORG = "iotlab"
INFLUX_BUCKET = "sensor_data"

feature_cols = ["temperature_c", "humidity_pct", "power_watt",
"current_amp"]

model = joblib.load(MODEL_PATH)

influx_client = InfluxDBClient(url=INFLUX_URL,
token=INFLUX_TOKEN, org=INFLUX_ORG)
write_api = influx_client.write_api(write_options=SYNCHRONOUS)

def validate_payload(payload):
    required = ["timestamp", "device_id", "temperature_c",
"humidity_pct", "power_watt", "current_amp"]
    for field in required:
        if field not in payload:
            raise ValueError(f"Field hilang: {field}")

    # validasi ringan
    float(payload["temperature_c"])
    float(payload["humidity_pct"])
    float(payload["power_watt"])
    float(payload["current_amp"])

def build_features(payload):
    row = pd.DataFrame([{
        "temperature_c": float(payload["temperature_c"]),
        "humidity_pct": float(payload["humidity_pct"]),
        "power_watt": float(payload["power_watt"]),
        "current_amp": float(payload["current_amp"]),
    }])
```

```
    return row

def on_connect(client, userdata, flags, reason_code,
properties=None):
    print("Connected to broker.")
    client.subscribe(TOPIC, qos=1)
    print("Subscribed to", TOPIC)

def on_message(client, userdata, msg):
    try:
        payload = json.loads(msg.payload.decode())
        validate_payload(payload)
        X = build_features(payload)

        # threshold baseline
        threshold_alert = int(
            X["power_watt"].iloc[0] > 280 or
            X["current_amp"].iloc[0] > 10.0
        )

        # Isolation Forest
        pred = model.predict(X[feature_cols])[0]  # 1 normal, -1
anomaly
        anomaly_flag = 1 if pred == -1 else 0

        print(
            f"[{payload['timestamp']}] "
            f"T={payload['temperature_c']} "
            f"H={payload['humidity_pct']} "
            f"P={payload['power_watt']} "
            f"I={payload['current_amp']} "
            f"-> anomaly={anomaly_flag},
threshold={threshold_alert}"
        )

        point = (
            Point("capstone_device_monitoring")
            .tag("device_id", payload["device_id"])
            .field("temperature_c",
float(payload["temperature_c"]))
            .field("humidity_pct",
float(payload["humidity_pct"]))
            .field("power_watt", float(payload["power_watt"]))
            .field("current_amp", float(payload["current_amp"]))
            .field("iforest_anomaly", anomaly_flag)
            .field("threshold_alert", threshold_alert)
            .time(payload["timestamp"], WritePrecision.S)
        )
        write_api.write(bucket=INFLUX_BUCKET, org=INFLUX_ORG,
record=point)

    except Exception as exc:
        print("Error:", exc)
```

```
client = mqtt.Client(client_id="capstone-subscriber")
client.on_connect = on_connect
client.on_message = on_message
client.connect(BROKER, PORT, 60)
client.loop_forever()
```

Jalankan:

```
python scripts/subscribers/capstone_subscriber.py
```

Apa yang dilakukan skrip ini?

- menerima data dari MQTT,
- memvalidasi payload,
- melakukan preprocessing otomatis ringan,
- menjalankan **threshold** dan **Isolation Forest**,
- menyimpan hasil ke InfluxDB.

Ini sangat sesuai dengan komponen proyek akhir: **subscriber Python, preprocessing otomatis, model anomaly detection, dashboard, dan notifikasi sederhana**.

9. Dashboard Grafana

Tambahkan **InfluxDB** sebagai data source di Grafana. Setelah itu buat dashboard dengan panel berikut:

Panel minimum

- **Temperature**
- **Humidity**
- **Power**
- **Current**
- **iforest_anomaly**
- **threshold_alert**

Panel yang sangat disarankan

- histori sensor 1 jam / 24 jam
- status anomali real-time
- jumlah alert per jam
- tabel event terbaru

Grafana memang sangat cocok sebagai dashboard open source utama untuk data *time series* dalam alur buku ini.

10. Verifikasi sistem end-to-end

Periksa hal berikut:

- publisher mengirim payload,
- broker menerima,
- subscriber mencetak prediksi,
- InfluxDB menerima data,
- Grafana menampilkan histori dan status.

Jika semuanya berjalan, maka capstone end-to-end dasar sudah hidup.

Hands-on 39 - Menambahkan Notifikasi Sederhana

1. Tujuan hands-on

Pada latihan ini, pembaca akan menambahkan **notifikasi sederhana** ke proyek akhir, sehingga sistem tidak hanya mencatat anomali, tetapi juga **mengabarkan** bahwa ada kondisi yang perlu perhatian.

2. Strategi notifikasi yang dipakai

Untuk tahap buku, strategi notifikasi tidak perlu rumit. Kita gunakan dua cara sederhana:

- publish ke topik MQTT `capstone/iotnode01/alert`
- simpan alert ke measurement InfluxDB khusus

Dengan pendekatan ini, pembaca bisa:

- melihat alert di terminal,
- subscribe alert dengan `mosquitto_sub`,
- menampilkan alert di Grafana,
- dan nanti mengembangkannya ke Telegram/email/WA bila mau.

3. Modifikasi subscriber agar mengirim alert

Buat file `scripts/subscribers/capstone_subscriber_with_alert.py`:

```
import json
from pathlib import Path
import joblib
import pandas as pd
import paho.mqtt.client as mqtt
from influxdb_client import InfluxDBClient, Point, WritePrecision
from influxdb_client.client.write_api import SYNCHRONOUS

BROKER = "localhost"
PORT = 1883
TOPIC_INPUT = "capstone/iotnode01/telemetry"
TOPIC_ALERT = "capstone/iotnode01/alert"

MODEL_PATH = Path("data/models/capstone_iforest.joblib")

INFLUX_URL = "http://localhost:8086"
INFLUX_TOKEN = "GANTI_DENGAN_TOKEN_ANDA"
INFLUX_ORG = "iotlab"
INFLUX_BUCKET = "sensor_data"

feature_cols = ["temperature_c", "humidity_pct", "power_watt",
"current_amp"]

model = joblib.load(MODEL_PATH)
```

```
influx_client = InfluxDBClient(url=INFLUX_URL, token=INFLUX_TOKEN,
org=INFLUX_ORG)
write_api = influx_client.write_api(write_options=SYNCHRONOUS)

def validate_payload(payload):
    required = ["timestamp", "device_id", "temperature_c",
"humidity_pct", "power_watt", "current_amp"]
    for field in required:
        if field not in payload:
            raise ValueError(f"Field hilang: {field}")

def build_features(payload):
    return pd.DataFrame([{
        "temperature_c": float(payload["temperature_c"]),
        "humidity_pct": float(payload["humidity_pct"]),
        "power_watt": float(payload["power_watt"]),
        "current_amp": float(payload["current_amp"]),
    }])

def on_connect(client, userdata, flags, reason_code, properties=None):
    client.subscribe(TOPIC_INPUT, qos=1)
    print("Subscribed to", TOPIC_INPUT)

def on_message(client, userdata, msg):
    try:
        payload = json.loads(msg.payload.decode())
        validate_payload(payload)
        X = build_features(payload)

        threshold_alert = int(
            X["power_watt"].iloc[0] > 280 or
            X["current_amp"].iloc[0] > 10.0
        )

        pred = model.predict(X[feature_cols])[0]
        anomaly_flag = 1 if pred == -1 else 0

        is_alert = threshold_alert == 1 or anomaly_flag == 1
        reason = []
        if threshold_alert:
            reason.append("threshold")
        if anomaly_flag:
            reason.append("iforest")
        reason_text = ",".join(reason) if reason else "none"

        # simpan data monitoring biasa
        point = (
            Point("capstone_device_monitoring")
            .tag("device_id", payload["device_id"])
            .field("temperature_c", float(payload["temperature_c"]))
            .field("humidity_pct", float(payload["humidity_pct"]))
            .field("power_watt", float(payload["power_watt"]))
            .field("current_amp", float(payload["current_amp"]))
            .field("iforest_anomaly", anomaly_flag)
            .field("threshold_alert", threshold_alert)
            .time(payload["timestamp"], WritePrecision.S)
        )
        write_api.write(bucket=INFLUX_BUCKET, org=INFLUX_ORG,
record=point)
```

```
            if is_alert:
                alert_payload = {
                    "timestamp": payload["timestamp"],
                    "device_id": payload["device_id"],
                    "temperature_c": payload["temperature_c"],
                    "humidity_pct": payload["humidity_pct"],
                    "power_watt": payload["power_watt"],
                    "current_amp": payload["current_amp"],
                    "reason": reason_text
                }

                client.publish(TOPIC_ALERT, json.dumps(alert_payload),
qos=1)
                print("ALERT:", alert_payload)

                alert_point = (
                    Point("capstone_alerts")
                    .tag("device_id", payload["device_id"])
                    .tag("reason", reason_text)
                    .field("temperature_c",
float(payload["temperature_c"]))
                    .field("humidity_pct", float(payload["humidity_pct"]))
                    .field("power_watt", float(payload["power_watt"]))
                    .field("current_amp", float(payload["current_amp"]))
                    .field("alert_flag", 1)
                    .time(payload["timestamp"], WritePrecision.S)
                )
                write_api.write(bucket=INFLUX_BUCKET, org=INFLUX_ORG,
record=alert_point)

        except Exception as exc:
            print("Error:", exc)

client = mqtt.Client(client_id="capstone-subscriber-alert")
client.on_connect = on_connect
client.on_message = on_message
client.connect(BROKER, PORT, 60)
client.loop_forever()
```

Jalankan:

```
python scripts/subscribers/capstone_subscriber_with_alert.py
```

4. Verifikasi alert di MQTT

Buka terminal baru:

```
mosquitto_sub -h localhost -t "capstone/iotnode01/alert" -v
```

Jika kondisi abnormal terjadi, Anda akan melihat payload alert keluar.

5. Tampilkan alert di Grafana

Tambahkan panel dari measurement:

- `capstone_alerts`

Panel yang bagus

- jumlah alert per jam
- tabel alert terbaru
- panel status merah bila ada alert baru
- filter berdasarkan `reason`

Tujuannya

Agar operator tidak hanya melihat data mentah, tetapi langsung melihat event penting yang perlu tindakan.

6. Insight Hands-on 39

Latihan ini mengajarkan bahwa:

- notifikasi sederhana sangat cukup untuk tahap awal,
- kombinasi **threshold + model** sering lebih berguna daripada salah satu saja,
- dan alert sebaiknya ditujukan untuk aksi operasional, bukan sekadar log tambahan.

Hands-on 40 - Uji Beban serta Skenario Pengembangan Lanjut

1. Tujuan hands-on

Pada latihan ini, pembaca akan:

- melakukan **uji beban ringan**,
- menguji pipeline dengan skenario data buruk,
- dan merancang **pengembangan lanjut** untuk capstone.

Ini sangat penting karena sistem yang tampak baik saat data normal masuk belum tentu tahan saat:

- data datang lebih cepat,
- payload rusak,
- anomali datang beruntun,
- atau distribusi data mulai bergeser.

2. Jenis pengujian yang realistis

Untuk buku ini, uji beban tidak perlu dibuat seperti *benchmark enterprise*. Cukup fokus pada empat hal:

- **uji volume**
- **uji payload rusak**
- **uji anomali beruntun**
- **uji drift buatan**

Mengapa empat ini?

Karena empat kondisi ini paling sering membuat pipeline lapangan mulai terlihat rapuh.

3. Uji volume: percepat laju publisher

Buat file `scripts/publishers/capstone_stress_publisher.py`:

```
import json
import random
import time
from datetime import datetime
import paho.mqtt.client as mqtt

BROKER = "localhost"
PORT = 1883
TOPIC = "capstone/iotnode01/telemetry"

client = mqtt.Client(client_id="capstone-stress-publisher")
client.connect(BROKER, PORT, 60)

for i in range(1000):
    payload = {
        "timestamp":
datetime.now().isoformat(timespec="seconds"),
        "device_id": "iotnode01",
        "temperature_c": round(random.uniform(24.0, 34.0), 2),
        "humidity_pct": round(random.uniform(40.0, 75.0), 2),
        "power_watt": round(random.uniform(80.0, 260.0), 2),
        "current_amp": round(random.uniform(3.5, 9.5), 2)
    }

    if random.random() < 0.12:
        payload["power_watt"] += random.uniform(100, 220)
        payload["current_amp"] += random.uniform(2, 6)

    client.publish(TOPIC, json.dumps(payload), qos=0)

    if i % 100 == 0:
        print(f"Published {i} messages")

    time.sleep(0.05)
```

Apa yang harus diamati?

- apakah subscriber tetap stabil,
- apakah Grafana tetap menampilkan data,
- apakah InfluxDB tetap menerima data,
- apakah alert menjadi terlalu banyak.

4. Uji payload rusak

Buat file `scripts/publishers/capstone_bad_payload_publisher.py`:

```
import json
import random
import time
from datetime import datetime
import paho.mqtt.client as mqtt

BROKER = "localhost"
TOPIC = "capstone/iotnode01/telemetry"

client = mqtt.Client(client_id="capstone-bad-publisher")
client.connect(BROKER, 1883, 60)

samples = [
    {"timestamp": datetime.now().isoformat(timespec="seconds"),
"device_id": "iotnode01"},
    {"timestamp": "bad_time", "device_id": "iotnode01",
"temperature_c": "abc", "humidity_pct": 50, "power_watt": 100,
"current_amp": 5},
    {"timestamp": datetime.now().isoformat(timespec="seconds"),
"device_id": "iotnode01", "temperature_c": -99, "humidity_pct":
999, "power_watt": 10000, "current_amp": -3},
]

for payload in samples:
    client.publish(TOPIC, json.dumps(payload), qos=1)
    print("Published bad payload:", payload)
    time.sleep(1)
```

Apa yang harus diamati?

- apakah subscriber menolak payload rusak,
- apakah error tercatat,
- apakah sistem tetap jalan setelah menerima data rusak.

Ini sangat sesuai dengan topik bab operasional sebelumnya tentang **data palsu, data hilang, data rusak** dan perlunya validasi input.

5. Uji drift buatan

Buat file `scripts/publishers/capstone_drift_publisher.py`:

```
import json
import random
import time
from datetime import datetime
import paho.mqtt.client as mqtt

BROKER = "localhost"
TOPIC = "capstone/iotnode01/telemetry"

client = mqtt.Client(client_id="capstone-drift-publisher")
client.connect(BROKER, 1883, 60)

base_temp = 26.0
base_power = 130.0

for i in range(500):
    # drift perlahan
    temp = base_temp + (i * 0.01) + random.uniform(-0.5, 0.5)
    power = base_power + (i * 0.08) + random.uniform(-10, 10)

    payload = {
        "timestamp":
datetime.now().isoformat(timespec="seconds"),
        "device_id": "iotnode01",
        "temperature_c": round(temp, 2),
        "humidity_pct": round(random.uniform(45.0, 65.0), 2),
        "power_watt": round(power, 2),
        "current_amp": round(random.uniform(4.0, 7.5), 2)
    }

    client.publish(TOPIC, json.dumps(payload), qos=1)
    print("Published drift payload:", payload)
    time.sleep(0.1)
```

Apa yang harus diamati?

- apakah distribusi data bergeser di dashboard,
- apakah model mulai lebih sering memberi anomaly,
- apakah sistem butuh retraining.

Ini berkaitan langsung dengan kebutuhan monitoring drift dan penurunan performa model

6. Checklist hasil uji beban

Setelah pengujian, pembaca sebaiknya menjawab:

- apakah broker tetap stabil?
- apakah subscriber tetap hidup?
- apakah error handling bekerja?
- apakah storage masih menerima data?
- apakah dashboard tetap bisa dipakai?
- apakah alert masih berguna atau jadi terlalu berisik?
- apakah ada tanda drift yang perlu ditangani?

7. Skenario pengembangan lanjut

Setelah capstone dasar berjalan, arah pengembangan lanjut yang sangat masuk akal adalah:

7.1 Multi-device

Dari satu node menjadi banyak node:

- `iotnode01`
- `iotnode02`
- `lab01`
- `room01`

7.2 Multi-sensor fusion

Tambah:

- CO_2
- vibration
- voltage
- GPS
- soil moisture

7.3 Edge deployment

Pindahkan subscriber + inferensi ke:

- Raspberry Pi
- mini PC edge
- gateway lokal

Ini sejalan dengan pembahasan edge AI dan inferensi lokal pada bab sebelumnya.

7.4 Retraining berkala

Tambahkan:

- batch retraining mingguan
- versioning model
- evaluasi drift berkala

7.5 Security hardening

Tambahkan:

- autentikasi MQTT
- pemisahan topik
- TLS bila perlu

7.6 Docker dan reproducibility

Setelah sistem dipahami, bungkus:

- Mosquitto
- InfluxDB
- Grafana
- subscriber
  ke dalam `docker-compose.yml`

7.7 Explainability ringan

Tambahkan:

- feature importance
- narasi penyebab alert
- log “kenapa alert dipicu”

8. Bentuk hasil akhir capstone yang ideal

Jika Hands-on 38–40 berjalan baik, maka hasil akhir proyek seharusnya memiliki:

- simulator atau sensor nyata yang mengirim data,
- broker MQTT yang stabil,
- subscriber Python dengan preprocessing otomatis,
- model anomaly detection yang bekerja,
- histori data di storage,
- dashboard Grafana yang hidup,
- alert sederhana,
- hasil uji beban dasar,
- dan daftar arah pengembangan lanjut.

Itulah bentuk **Sistem IoT-ML Open Source Lengkap** yang realistis, bukan hanya demo sekali jalan.

9. Penutup

Hands-on 38–40 adalah penutup yang sangat kuat untuk seluruh buku ini. Pada **Hands-on 38**, pembaca membangun **proyek akhir end-to-end** yang benar-benar menyatukan data, MQTT, preprocessing, model anomaly detection, storage, dan dashboard. Pada **Hands-on 39**, pembaca menambahkan **notifikasi sederhana** agar sistem mulai bergerak dari sekadar observasi menuju tindakan. Pada **Hands-on 40**, pembaca menguji sistem dengan **beban lebih tinggi, payload rusak, drift buatan, dan skenario pengembangan lanjut**, sehingga proyek akhir tidak hanya “berjalan”, tetapi mulai terlihat layak sebagai sistem operasional kecil.

Bab Penutup

Epilog. Roadmap Belajar dan Pengembangan Lanjut

Buku ini dimulai dari pertanyaan yang sederhana tetapi sangat penting: bagaimana membawa pembaca dari data sensor mentah menuju **keputusan cerdas** dengan pendekatan yang **praktis**, **open source**, dan **mudah direplikasi**. Di sepanjang perjalanan, pembaca telah belajar memahami karakter data IoT, membangun *pipeline* dari MQTT ke Python, membersihkan dan membentuk fitur, melatih model, membuat dashboard, mendeteksi anomali, melakukan *forecasting*, mencoba inferensi di edge, hingga menyusun proyek akhir yang utuh. Semua itu dirancang agar pembaca tidak berhenti di teori, tetapi benar-benar bisa membangun sistem kecil yang hidup. Hal ini sepenuhnya sejalan dengan ruh buku yang memang memprioritaskan praktik yang dapat dijalankan dibanding spekulasi teknologi yang terlalu jauh.

Namun, **menyelesaikan buku ini bukan garis akhir**. Justru di sinilah perjalanan yang lebih menarik dimulai. Setelah memahami fondasi, pembaca kini berada pada posisi yang sangat baik untuk memilih jalur pengembangan yang paling sesuai dengan minat, kebutuhan kerja, atau konteks riset. Ada pembaca yang akan lebih menikmati menyusun perangkat, broker, dan *pipeline* Python. Ada yang lebih tertarik menggali data, membuat visualisasi, dan membangun *insight*. Ada juga yang ingin melangkah lebih jauh ke arah model yang lebih kuat, *deployment* di edge, atau sistem industri yang lebih besar. Karena itu, epilog ini akan memetakan tiga jalur belajar utama, menjelaskan kapan saatnya naik kelas ke teknologi yang lebih kompleks, lalu menutup dengan ekosistem komunitas dan dataset publik yang bisa dipakai untuk terus berlatih. Semua elemen ini memang telah direncanakan sebagai isi utama epilog.

Jalur belajar 1 — *Python IoT Developer*

Jalur pertama sangat cocok untuk pembaca yang paling menikmati sisi membangun sistem. Bila Anda senang saat melihat sensor berhasil mengirim data, MQTT broker hidup, subscriber Python bekerja, dashboard menampilkan grafik, dan notifikasi muncul saat kondisi tidak normal, maka kemungkinan besar Anda berada di jalur **Python IoT developer**. Jalur ini adalah jalur bagi orang yang ingin menjadi penghubung antara dunia perangkat, jaringan ringan, *data pipeline*, dan aplikasi operasional.

Seorang *Python IoT developer* idealnya kuat di beberapa fondasi inti. Pertama, ia harus sangat nyaman dengan **Python** untuk hal-hal praktis: membaca CSV/JSON, membuat *script*, membangun fungsi *preprocessing*, menangani error, menulis ke database, dan berkomunikasi lewat MQTT atau HTTP. Kedua, ia perlu memahami arsitektur IoT yang cukup untuk bekerja dengan tenang: sensor, mikrokontroler, gateway, broker, penyimpanan, dan dashboard. Ketiga, ia perlu disiplin pada struktur proyek, logging, validasi data, dan *deployment* sederhana. Ini sangat sejalan dengan fokus utama yang sejak awal memang menempatkan Python, MQTT, dashboard

open source, dan *pipeline end-to-end* sebagai prioritas paling relevan untuk *hands-on*.

Langkah belajar yang disarankan untuk jalur ini adalah sebagai berikut. Mulailah dengan memperkuat Python sampai Anda benar-benar lancar membaca dan menulis *script* tanpa bergantung penuh pada notebook. Setelah itu, kuasai MQTT, baik sebagai publisher maupun subscriber, lalu biasakan menyimpan data ke SQLite atau InfluxDB. Berikutnya, pelajari cara merapikan proyek: memisahkan folder `scripts`, `data`, `models`, `outputs`, dan membuat `README` yang jelas. Bila fondasi itu sudah stabil, lanjutkan dengan integrasi ke **Grafana**, lalu latih diri Anda untuk membuat sistem yang tetap hidup walau input tidak selalu rapi. Setelah itu, Anda bisa mengarah ke Raspberry Pi atau mini PC sebagai target edge yang lebih realistis. Ini adalah jalur yang sangat kuat bagi pembaca yang ingin cepat produktif di sekolah, laboratorium, kampus, UMKM, atau proyek integrasi sistem kecil-menengah.

Yang perlu dipahami, jalur ini bukan jalur “teknisi kabel” semata dan juga bukan jalur “data scientist” murni. Ia berada di tengah: cukup paham perangkat, cukup paham data, cukup paham *deployment*, dan cukup paham operasional. Di Indonesia, profil seperti ini sangat dibutuhkan, karena banyak organisasi belum membutuhkan sistem AI raksasa, tetapi sangat membutuhkan orang yang mampu membuat **sistem IoT yang benar-benar bekerja**.

Jalur belajar 2 — *Data Analyst IoT*

Jalur kedua cocok untuk pembaca yang paling menikmati saat data mulai “berbicara”. Bila Anda tertarik mencari pola, membuat visualisasi, membandingkan distribusi, membaca *time series*, menjelaskan hasil model, dan menyusun *insight* yang berguna untuk keputusan, maka jalur **data analyst IoT** adalah arah yang sangat masuk akal.

Pada jalur ini, fokus utama bukan pada sensor sebagai perangkat, melainkan pada **makna dari data sensor**. Anda akan lebih banyak bekerja pada pertanyaan seperti: kapan suhu ruangan mulai tidak nyaman, pola beban energi seperti apa yang muncul tiap hari kerja, bagaimana membedakan perilaku normal dan anomali, apakah distribusi data bergeser, dan bagaimana menyampaikan temuan itu ke operator atau pimpinan secara jelas. Jalur ini sangat sejalan dengan isi buku yang banyak menekankan **data cleaning**, **visualization**, **descriptive/diagnostic analytics**, **feature engineering**, **forecasting**, dan **explainability ringan**.

Untuk berkembang di jalur ini, pembaca perlu memperdalam **pandas**, **NumPy**, dan **matplotlib/plotly**. Setelah itu, latih kemampuan membangun metrik yang berguna: rata-rata per jam, variasi lokal, *rolling statistics*, rasio anomali, *forecast error*, dan sejenisnya. Berikutnya, kuasai cara membaca model secara operasional: confusion matrix, MAE/RMSE, precision/recall, feature importance, serta monitoring drift. Lalu, biasakan menyusun dashboard yang tidak hanya “cantik”, tetapi **mendorong keputusan**. Data analyst IoT yang baik harus mampu mengubah *time series* menjadi jawaban terhadap pertanyaan operasional, bukan sekadar membuat grafik panjang.

Yang menarik, jalur ini juga bisa menjadi jembatan menuju dua arah lain. Dari sini, seseorang bisa bergerak ke arah **product analytics** untuk sistem IoT, ke arah **operation intelligence** untuk gedung/energi/maintenance, atau naik ke tingkat yang lebih teknis menuju *machine learning engineer*. Jadi, bila Anda suka data tetapi belum yakin ingin mendalami model yang rumit, jalur ini adalah titik yang sangat sehat.

Jalur belajar 3 — *ML Engineer* untuk edge/IIoT

Jalur ketiga cocok untuk pembaca yang tertarik pada inti model dan ingin membawa kecerdasan sistem lebih jauh, terutama pada konteks **edge** dan **Industrial IoT (IIoT)**. Bila Anda tertarik membandingkan model, mengevaluasi performa, memikirkan drift, menjalankan inferensi di Raspberry Pi, mengonversi model ke TensorFlow Lite, atau menyusun *pipeline* yang lebih matang, maka jalur **ML engineer untuk edge/IIoT** adalah langkah lanjut yang sangat natural.

Jalur ini membutuhkan fondasi yang lebih menyeluruh. Anda tetap perlu paham Python, data, dan *pipeline*, tetapi juga harus lebih dalam pada:

- desain dataset,
- *feature engineering* yang konsisten,
- evaluasi model yang jujur,
- konsep drift,
- versi model,
- *deployment* di edge,
- serta trade-off antara akurasi, ukuran model, latensi, dan kestabilan.

Ini sangat sejalan dengan bab-bab pertengahan hingga akhir buku yang menekankan **supervised learning**, **anomaly detection**, **forecasting**, **edge inference**, **model ringan**, dan **operasional sistem IoT-ML**. Bahkan secara eksplisit menempatkan **ML engineer untuk edge/IIoT** sebagai salah satu jalur belajar utama di epilog.

Bila ingin berkembang ke arah ini, disarankan urutan belajar yang disiplin. Pertama, kuasai model klasik di scikit-learn sampai Anda benar-benar paham kapan memakai Logistic Regression, Random Forest, Isolation Forest, dan kapan baseline sederhana sudah cukup. Kedua, kuasai *forecasting time series* berbasis lag dan model tabular sebelum tergoda langsung ke model yang lebih besar. Ketiga, pelajari deployment model ke edge: bagaimana mengekspor model, bagaimana menguji inferensi lokal, bagaimana mengukur latensi, dan bagaimana menyiapkan fallback saat input buruk. Keempat, biasakan memikirkan sistem bukan hanya saat training, tetapi juga saat **operasional**: logging, monitoring, drift, alert, dan versi model. Bila fondasi ini kuat, Anda akan jauh lebih siap masuk ke dunia IIoT yang lebih kompleks.

Jalur ini sangat menarik untuk pembaca yang ingin membangun solusi untuk:

- predictive maintenance,
- smart energy node,

- soft sensor,
- monitoring industri,
- anomaly detection lokal,
- atau edge analytics pada fasilitas kampus, pabrik kecil, dan sistem utilitas.

Kapan pindah ke Kafka atau Spark?

Salah satu pertanyaan yang sangat penting setelah menyelesaikan buku ini adalah: kapan saatnya meninggalkan stack ringan seperti MQTT + Python subscriber + SQLite/InfluxDB, lalu pindah ke sistem yang lebih besar seperti **Kafka** atau **Spark**?

Jawaban praktisnya adalah: **jangan pindah terlalu cepat**. Kafka, Spark, dan stack enterprise besar **tidak dijadikan fokus utama** untuk target buku ini, karena ruang utama sebaiknya dipakai untuk praktek Python yang benar-benar bisa dijalankan pembaca. Itu keputusan yang sangat bijak.

Anda baru perlu mempertimbangkan **Kafka** ketika sistem mulai mengalami karakter berikut:

- volume data jauh lebih tinggi,
- banyak sumber data masuk secara paralel,
- Anda butuh *event streaming* yang lebih tahan gangguan dan lebih skalabel,
- ada banyak konsumen data yang berbeda,
- atau Anda mulai butuh pemisahan yang lebih kuat antara ingest, pemrosesan, dan konsumsi data.

Referensi yang Anda lampirkan juga menunjukkan bahwa **Apache Kafka** sering dipakai sebagai *streaming backbone* untuk *soft sensor*, streaming analytics, dan sistem data industri skala lebih besar. Namun, itu biasanya dilakukan setelah fondasi *feature extraction, model training, dan inference* sudah jelas. Artinya, Kafka masuk akal ketika masalah Anda bukan lagi “bagaimana membuat pipeline hidup”, melainkan “bagaimana membuat pipeline yang hidup ini bisa melayani banyak aliran data secara lebih andal dan besar.”

Adapun **Spark** baru relevan ketika Anda mulai menghadapi salah satu dari kondisi berikut:

- data historis sangat besar,
- analisis batch menjadi terlalu berat untuk pandas biasa,
- ada kebutuhan agregasi besar dan berulang,
- atau Anda ingin menggabungkan *streaming* dan batch secara lebih formal.

Jadi, tanda Anda siap pindah ke Kafka/Spark bukan karena “ingin terlihat modern”, melainkan karena stack ringan Anda mulai **terbukti menjadi bottleneck**.

Kapan masuk ke MLOps yang lebih serius?

Banyak pembaca akan mendengar istilah *MLOps* dan tergoda merasa belum lengkap bila belum memakai *model registry*, *experiment tracking*, *automated deployment*, *CI/CD*, dan sebagainya. Untuk konteks buku ini, Anda **tidak perlu terburu-buru**.

Anda baru perlu masuk ke **MLOps yang lebih serius** ketika:

- Anda punya lebih dari satu model aktif,
- model diperbarui secara berkala,
- ada kebutuhan audit yang lebih formal,
- lebih dari satu orang mengelola model,
- atau sistem sudah dipakai untuk keputusan yang lebih penting dan berulang.

Sebelum itu, praktik minimum yang sudah Anda pelajari sebenarnya sudah sangat bagus:

- simpan model dengan versi,
- simpan metadata training,
- catat metrik validasi,
- tulis log inferensi,
- pantau drift sederhana,
- dan dokumentasikan model aktif.

Logging, monitoring, dan versi model sederhana adalah praktik minimum wajib, bukan langsung MLOps enterprise. Itu adalah langkah yang sangat sehat.

Begitu sistem tumbuh, barulah MLOps menjadi masuk akal. Tetapi selama proyek Anda masih berada pada tahap laboratorium, kampus, prototipe, atau implementasi kecil, *good engineering discipline* sering jauh lebih penting daripada tumpukan alat MLOps yang terlalu berat.

Kapan lanjut ke TinyML yang lebih serius?

Buku ini sudah memperkenalkan **Edge AI** dan **TensorFlow Lite secara praktis**, serta menempatkan **ESP32 / Raspberry Pi** sebagai target praktis. Itu adalah fondasi yang tepat.

Anda baru perlu masuk ke **TinyML lanjutan** bila:

- model benar-benar harus jalan di mikrokontroler,
- latensi harus sangat kecil,
- konektivitas minim atau tidak stabil,
- perangkat harus hemat daya,
- atau Anda ingin sistem benar-benar mandiri di level perangkat.

Namun, sebelum masuk ke TinyML yang lebih dalam, sangat disarankan pembaca memastikan tiga hal sudah kuat:

- inferensi lokal di Raspberry Pi atau edge Linux sudah dikuasai,

- model ringan dan *pipeline* inferensi sudah stabil,
- dan trade-off antara akurasi, ukuran model, serta latensi sudah dipahami.

Jangan melompat ke TinyML hanya karena istilahnya menarik. TinyML lanjutan masuk akal ketika **batas perangkat menjadi faktor utama desain**, bukan sekadar rasa ingin mencoba.

Kapan federated learning mulai relevan?

Federated learning adalah topik yang menarik, tetapi untuk banyak pembaca buku ini, ia belum perlu menjadi fokus awal. Anda baru perlu serius mempelajari federated learning ketika sistem Anda mulai memiliki karakter seperti:

- banyak perangkat atau node yang masing-masing menyimpan data lokal,
- data tidak boleh atau tidak mudah dikumpulkan ke satu tempat,
- privasi menjadi isu penting,
- atau Anda ingin memperbarui model dari banyak lokasi tanpa memindahkan data mentah secara penuh.

Untuk sebagian besar pembaca yang baru menyelesaikan buku ini, tantangan yang lebih mendesak biasanya masih berada pada:

- kualitas data,
- *pipeline* stabil,
- model yang benar,
- deployment yang masuk akal,
- dan monitoring operasional.

Jadi, federated learning adalah **langkah lanjut**, bukan langkah wajib berikutnya.

Komunitas open source

Salah satu kekuatan terbesar dari jalur belajar ini adalah Anda **tidak harus belajar sendirian**. Ekosistem IoT dan machine learning open source sangat kaya, dan justru akan menjadi jauh lebih kuat bila pembaca aktif berinteraksi dengan komunitas.

Komunitas yang sangat layak diikuti adalah komunitas yang terkait dengan stack utama buku ini:

- komunitas **Python**
- komunitas **pandas**, **NumPy**, **matplotlib**, dan **scikit-learn**
- komunitas **TensorFlow** atau **Keras**
- komunitas **Mosquitto MQTT**
- komunitas **Node-RED**
- komunitas **InfluxDB**
- komunitas **Grafana**
- komunitas **Home Assistant** untuk inspirasi integrasi *smart environment*
- komunitas **Raspberry Pi** untuk edge deployment

Yang perlu dipahami, komunitas open source bukan hanya tempat bertanya saat error. Ia juga tempat untuk:

- melihat contoh proyek nyata,
- belajar gaya dokumentasi yang baik,
- memahami *best practice*,
- membaca diskusi tentang trade-off teknis,
- dan menemukan inspirasi lanjutan.

Buku ini dengan mengarahkan anda ke **komunitas open source dan proyek lanjutan**, sehingga pembaca memang didorong untuk terus berkembang di ekosistem terbuka, bukan bergantung pada satu vendor atau alat tertutup.

Dataset publik untuk latihan

Epilog ini juga idealnya menutup dengan pesan yang sangat praktis: **jangan berhenti berlatih hanya karena proyek akhir buku selesai**. Salah satu cara terbaik untuk terus berkembang adalah mengerjakan ulang *pipeline* dan model dengan **dataset publik**.

Seperti disarankan di **Appendix B. Dataset IoT Publik untuk Latihan** dan menyebut kategori seperti:

- lingkungan,
- energi,
- mesin / predictive maintenance,
- wearable.

Itu adalah arah yang sangat tepat, karena empat kategori ini memang mewakili spektrum aplikasi yang kaya:

- **lingkungan** untuk suhu, kelembaban, CO_2, kualitas udara, cuaca mikro
- **energi** untuk smart meter, beban listrik, smart building, efisiensi
- **mesin / predictive maintenance** untuk getaran, arus, fault, anomaly
- **wearable** untuk data tubuh, aktivitas, kesehatan, atau gerak

Cara terbaik memakai dataset publik bukan sekadar “menghasilkan skor model tinggi”, tetapi menggunakan dataset itu untuk melatih ulang seluruh kebiasaan kerja yang sudah Anda bangun dari buku ini: validasi data, preprocessing, *feature engineering*, baseline, evaluasi, visualisasi, *deployment* mini, hingga monitoring drift sederhana.

Strategi belajar setelah buku ini selesai

Setelah menutup buku ini, disarankan pembaca tidak langsung melompat ke terlalu banyak topik sekaligus. Pilih satu jalur utama, lalu latih diri Anda dengan ritme yang stabil.

Bila Anda ingin menjadi **Python IoT developer**, ambil satu proyek nyata kecil dan fokus sampai selesai. Misalnya, bangun sistem monitoring ruangan atau energi yang benar-benar stabil dan bisa dijalankan terus. Pastikan Anda nyaman dengan struktur proyek, MQTT, storage, dashboard, dan *deployment* sederhana.

Bila Anda ingin menjadi **data analyst IoT**, ulangi beberapa studi kasus dengan dataset publik yang berbeda. Latih kemampuan membaca pola, membuat *insight*, membandingkan distribusi, menjelaskan drift, dan menulis interpretasi hasil model yang mudah dimengerti.

Bila Anda ingin bergerak ke **ML engineer edge/IIoT**, ambil satu kasus dan dalami. Misalnya predictive maintenance sederhana. Latih evaluasi model, versi model, inferensi di edge, dan *pipeline* retraining sederhana. Setelah itu baru naik ke topik yang lebih besar seperti Kafka, TinyML lanjutan, atau MLOps yang lebih formal.

Kuncinya adalah **kedalaman lebih penting daripada banyaknya topik**. Oleh karenanya struktur buku dibuat agar buku lebih fokus, lebih mudah diikuti, lebih kuat secara konsep, dan tidak tenggelam dalam terlalu banyak topik tingkat riset. Prinsip itu sebaiknya tetap dibawa pembaca setelah buku selesai.

Penutup epilog

Akhir dari buku ini seharusnya menjadi awal dari perjalanan baru. Bila pada awal buku pembaca mungkin hanya melihat IoT sebagai kumpulan sensor dan machine learning sebagai kumpulan algoritma, maka pada akhir buku ini pembaca seharusnya sudah melihat keduanya sebagai bagian dari satu sistem yang utuh: **data dikumpulkan, dipahami, dibersihkan, dimodelkan, dijalankan, diawasi, dan dipakai untuk mengambil keputusan**. Itulah inti dari transformasi yang ingin dibangun buku ini. Dan karena itu, epilog ini tidak menutup perjalanan, tetapi justru membuka tiga jalan yang sangat nyata: menjadi **Python IoT developer** yang kuat di *pipeline* dan integrasi, menjadi **data analyst IoT** yang mampu mengubah data sensor menjadi *insight* operasional, atau menjadi **ML engineer untuk edge/IIoT** yang mampu membawa model dari laptop ke sistem nyata. Setelah fondasi ini kuat, barulah topik seperti **Kafka/Spark**, **MLOps yang lebih serius**, **TinyML lanjutan**, dan **federated learning** menjadi masuk akal sebagai langkah berikutnya, bukan sekadar nama besar yang dikejar terlalu cepat.

Appendix A. Instalasi dan Setup Open Source Stack

Appendix A sebaiknya menjadi lampiran yang paling sering dibuka pembaca pada tahap awal. Fungsinya adalah memastikan semua alat utama yang dipakai sepanjang buku benar-benar dapat dipasang, dijalankan, dan diuji secara bertahap. Buku ini memang menempatkan **Python, Jupyter, Mosquitto, Node-RED, InfluxDB/SQLite, dan Grafana** sebagai isi utama appendix ini, dan pilihan tersebut sangat tepat karena seluruh bab praktik bergantung pada stack tersebut. Selain itu, buku ini juga secara eksplisit mengadopsi arsitektur open source minimal untuk praktikum: **ESP32/simulator sensor, Mosquitto MQTT, Node-RED, InfluxDB/SQLite, dan Grafana**.

Python

Python adalah fondasi utama buku ini. Hampir semua praktik—mulai dari membaca CSV, membersihkan *time series*, membuat fitur, melatih model, mengirim data MQTT, hingga inferensi di edge—dilakukan dengan Python. Karena itu, appendix ini perlu memandu pembaca dari nol secara rapi.

Langkah yang disarankan:

- pasang Python versi stabil yang modern
- cek versi Python dari terminal
- buat folder proyek utama
- buat *virtual environment* `.venv`
- aktifkan *virtual environment*
- instal library inti buku
- uji impor library dasar

Alur yang sehat untuk pembaca:

1. Pasang Python.
2. Buka terminal dan jalankan pengecekan versi.
3. Buat folder proyek, misalnya `iot-ml-book/`.
4. Masuk ke folder proyek lalu buat `.venv`.
5. Aktifkan `.venv`.
6. Instal library inti seperti `pandas`, `numpy`, `matplotlib`, `scikit-learn`, `paho-mqtt`, `influxdb-client`, dan bila perlu `tensorflow`.
7. Buat skrip kecil untuk menguji apakah impor library berhasil.

Yang sangat penting untuk ditekankan adalah **biasakan bekerja di dalam *virtual environment***. Ini akan mengurangi konflik dependensi, membuat proyek lebih rapi, dan memudahkan reproduksi latihan di komputer lain. Karena buku ini memang menargetkan pembaca yang ingin cepat produktif secara praktis, kebiasaan kecil seperti ini justru sangat menentukan kelancaran seluruh *hands-on*.

Jupyter

Jupyter sangat berguna untuk eksperimen cepat, eksplorasi dataset, visualisasi awal, dan *notebook* pengujian model. Walaupun banyak skrip akhir dalam buku ini ditulis sebagai file `.py`, Jupyter tetap penting sebagai ruang eksplorasi.

Tahap yang perlu dilakukan:

- instal `jupyterlab`
- jalankan Jupyter dari folder proyek
- buat notebook pertama
- uji pembacaan CSV sederhana
- uji visualisasi pertama dengan `matplotlib`

Appendix ini sebaiknya menekankan bahwa **Jupyter cocok untuk eksplorasi**, sedangkan **skrip `.py` cocok untuk operasional**. Kebiasaan membedakan keduanya akan membantu pembaca saat nanti masuk ke pipeline otomatis, inferensi berulang, dan deployment mini.

Mosquitto

Mosquitto adalah broker MQTT utama yang dipakai sepanjang buku. Appendix ini perlu menjelaskan instalasi, cara menyalakan layanan, dan cara menguji broker dengan publisher/subscriber sederhana. Ini penting karena banyak pembaca baru akan benar-benar memahami IoT ketika mereka melihat data mengalir melalui topik MQTT, bukan hanya saat membaca definisi protokol.

Langkah yang sebaiknya dijelaskan:

- instal paket Mosquitto
- aktifkan service
- cek status service
- uji publish pesan sederhana
- uji subscribe dari terminal
- uji publish dari Python

Referensi menempatkan **MQTT Mosquitto** sebagai komponen inti arsitektur praktikum. Hands-on awal juga langsung membawa pembaca ke pengiriman data sensor ke MQTT dan *subscribe* dengan Python.

Node-RED

Node-RED sangat cocok untuk pembaca yang ingin melihat alur data secara visual. Walaupun buku ini tetap menjadikan Python sebagai jantung pemrosesan dan machine learning, Node-RED dapat menjadi alat bantu yang sangat nyaman untuk:

- memonitor data MQTT
- membuat dashboard ringan
- membuat alur otomasi cepat
- menghubungkan broker ke layanan lain

Tahap yang disarankan:

- pasang Node-RED
- jalankan Node-RED
- buka antarmuka web lokal
- buat alur sederhana: MQTT input → debug
- buat alur kedua: MQTT input → dashboard ringan

Appendix ini sebaiknya tidak membuat Node-RED menjadi fokus utama, tetapi memposisikannya sebagai alat visual yang sangat membantu. Ini sejalan dengan ruh buku yang mencantumkan Node-RED sebagai bagian dari stack open source minimal, tetapi tetap menjaga Python sebagai alat utama praktik ML.

InfluxDB / SQLite

Bagian ini sangat penting karena banyak pembaca bingung kapan memakai database *time series* dan kapan cukup memakai database ringan lokal. Appendix A perlu membantu pembaca membedakan keduanya secara praktis.

SQLite cocok untuk:

- penyimpanan lokal ringan
- prototipe kecil
- log inferensi
- eksperimen offline
- sistem yang ingin sangat sederhana

InfluxDB cocok untuk:

- data *time series*
- data sensor yang terus mengalir
- integrasi dengan Grafana
- histori pengukuran real-time
- alert dan panel operasional

Tahap yang perlu dijelaskan:

- instal SQLite atau gunakan bawaan Python
- uji menulis satu tabel sederhana
- jalankan InfluxDB, misalnya lewat Docker
- buat organization, bucket, dan token
- uji menulis satu record dari Python
- uji membaca data dari antarmuka web atau Grafana

Buku ini memang menempatkan **InfluxDB / SQLite** sebagai storage utama dalam stack praktikum. Karena itu, appendix ini harus memberi pembaca pedoman keputusan yang sederhana: **mulailah dari SQLite bila ingin sangat ringan, gunakan InfluxDB bila ingin dashboard *time series* yang hidup**.

Grafana

Grafana adalah **dashboard utama** dalam buku ini. Grafana dipakai untuk menampilkan data *real-time*, membaca data historis, dan membantu pemantauan operasional. Karena banyak praktikum berujung pada dashboard, lampiran ini harus **praktis** dan **langsung bisa dicoba**.

Lampiran Grafana sebaiknya memandu pembaca untuk:

- menjalankan Grafana,
- login pertama kali,
- mengganti password awal,
- menambahkan *data source*,
- menghubungkan ke **InfluxDB**,
- membuat panel pertama,
- membuat panel historis untuk suhu atau daya,
- membuat panel status atau *alert* sederhana.

Secara arsitektural, buku ini menempatkan Grafana sebagai **lapisan visualisasi utama** pada *open source stack* praktikum.

Langkah praktik

1. Menjalankan Grafana

Jika memakai layanan Linux:

```
sudo systemctl enable grafana-server
sudo systemctl start grafana-server
sudo systemctl status grafana-server
```

Cek akses:

```
curl -I http://localhost:3000
ss -tulpn | grep 3000
```

Jika memakai Docker:

```
docker run -d --name grafana -p 3000:3000
grafana/grafana:latest
docker ps
docker logs grafana
```

2. Login pertama kali

Gunakan akun awal:

- **username**: `admin`
- **password**: `admin`

Setelah masuk, pembaca perlu langsung mengganti password bawaan agar instalasi lebih aman.

3. Ganti password awal

Dapat dilakukan lewat antarmuka web. Jika perlu dari terminal:

```
sudo grafana-cli admin reset-admin-password
PasswordBaruYangKuat
```

Jika memakai Docker:

```
docker exec -it grafana grafana-cli admin
reset-admin-password PasswordBaruYangKuat
```

4. Tambahkan *data source*

Grafana memerlukan *data source* untuk membaca data. Dalam buku ini, *data source* utama adalah **InfluxDB** karena cocok untuk data *time-series* seperti suhu, kelembaban, arus, tegangan, dan daya.

Cek layanan InfluxDB:

```
curl http://localhost:8086/health
```

Jika menjalankan InfluxDB dengan Docker:

```
docker run -d --name influxdb -p 8086:8086 influxdb:2
docker logs influxdb
```

5. Hubungkan ke InfluxDB

Parameter yang biasanya diperlukan:

- **URL**: `http://localhost:8086`
- **Organization**

- **Bucket**
- **Token**

Setelah koneksi berhasil, Grafana siap membaca data sensor dan data historis dari InfluxDB.

6. Buat panel pertama

Panel pertama sebaiknya sederhana, misalnya:

- nilai suhu saat ini,
- daya saat ini,
- status perangkat.

Jenis panel yang cocok:

- **Stat**
- **Gauge**
- **Time series**

Contoh kirim data uji ke InfluxDB:

```
curl -X POST
"http://localhost:8086/api/v2/write?org=lab-iot&bucket=sensor&p
recision=s" \
  --header "Authorization: Token TOKEN_ANDA" \
  --data-raw "suhu,device=node1 value=29.5"
```

7. Buat panel historis

Panel historis dipakai untuk melihat perubahan data dari waktu ke waktu, misalnya:

- suhu harian,
- konsumsi daya,
- lonjakan beban,
- pola anomali.

Contoh kirim data berulang:

```
for i in {1..10}; do
  nilai=$((20 + RANDOM % 10))
  curl -X POST
"http://localhost:8086/api/v2/write?org=lab-iot&bucket=sensor&p
recision=s" \
    --header "Authorization: Token TOKEN_ANDA" \
    --data-raw "suhu,device=node1 value=${nilai}"
  sleep 5
done
```

8. Buat panel status atau *alert* sederhana

Panel status membantu melihat kondisi sistem secara cepat, misalnya:

- **aman**,
- **waspada**,
- **kritis**.

Contoh aturan sederhana:

- suhu < 30°C = aman
- suhu 30–35°C = waspada
- suhu > 35°C = kritis

Alert sederhana dapat dibuat untuk kondisi seperti:

- suhu melewati ambang,
- daya terlalu tinggi,
- sensor tidak mengirim data.

Penutup Appendix A

Bagian penutup Appendix A sebaiknya memuat **struktur folder proyek standar**. Ini akan membantu pembaca sejak awal untuk membedakan antara data mentah, data olahan, model, skrip, dan output.

Appendix B. Dataset IoT Publik untuk Latihan

Appendix B sebaiknya menjadi peta latihan lanjutan bagi pembaca. Fungsinya bukan sekadar memberi daftar nama dataset, tetapi membantu pembaca memahami **dataset jenis apa cocok untuk latihan yang mana**. Dataset publik untuk IoT terutama dalam kategori **lingkungan, energi, mesin/predictive maintenance, dan wearable**, dan empat kategori ini sangat tepat karena mewakili spektrum aplikasi utama IoT analytics.

Dataset lingkungan

Kategori lingkungan paling cocok untuk latihan:

- monitoring suhu dan kelembaban
- kualitas udara
- CO_2
- tekanan
- pencahayaan
- cuaca mikro

Dataset lingkungan sangat bagus untuk:

- pembersihan *time series*
- *resampling*
- imputasi *missing value*
- visualisasi multi-variabel
- klasifikasi kondisi nyaman
- *forecasting* suhu/kelembaban

Kategori ini juga sangat selaras dengan studi kasus **Smart Room Monitoring** dan dengan pembahasan data suhu, kelembaban, serta CO_2.

Dataset energi

Kategori energi sangat cocok untuk:

- smart meter
- smart building
- beban listrik
- arus dan daya
- analitik jam sibuk
- *forecasting* konsumsi energi

Dataset energi berguna untuk:

- *feature engineering* berbasis waktu
- prediksi beban
- deteksi beban tidak normal
- dashboard operasional
- analitik akhir pekan vs hari kerja

Dataset mesin / predictive maintenance

Kategori ini sangat kuat untuk:

- anomaly detection
- ekstraksi fitur sinyal
- *windowing*
- fault classification
- deteksi gejala awal kerusakan

Dataset jenis ini biasanya memiliki:

- getaran
- arus motor
- suhu mesin
- label normal/warning/fault, atau
- hanya data normal dengan sedikit kejadian anomali

Kategori ini selaras dengan aplikasi **predictive maintenance** yang di buku ini ditempatkan sebagai salah satu studi kasus utama dan salah satu domain aplikasi penting IoT.

Dataset wearable

Kategori wearable sangat baik untuk latihan:

- data fisiologis
- aktivitas tubuh
- sinyal gerak
- monitoring kesehatan ringan
- klasifikasi aktivitas
- anomaly detection pada sinyal tubuh

Referensi juga menyinggung peran *wearable devices* dan smart healthcare dalam konteks IoT modern.

Cara memakai Appendix B

Agar appendix ini benar-benar berguna, setiap kategori sebaiknya diakhiri dengan saran singkat seperti:

- **Latihan yang cocok**
- **Bab yang relevan untuk diulang**
- **Model yang cocok dicoba**
- **Jenis dashboard yang cocok dibuat**

Dengan begitu, pembaca tidak hanya melihat kategori data, tetapi juga langsung tahu **apa yang bisa dikerjakan** dengan data tersebut.

Appendix C. Template Kode Python

Appendix C berfungsi sebagai **tool kit praktis** bagi pembaca. Lampiran ini sebaiknya berisi kumpulan potongan kode yang paling sering dipakai berulang dalam seluruh buku, disusun singkat, jelas, dan mudah di-*copy-paste* untuk dimodifikasi. Fokus utamanya sangat tepat: **koneksi MQTT, loader CSV, preprocessing, training model, evaluasi, dan inferensi**, karena hampir semua eksperimen dan studi kasus dalam buku ini bergerak pada pola teknis tersebut.

Lampiran ini bukan tempat untuk teori panjang. Tujuannya adalah memberi pembaca **template siap pakai** agar mereka dapat lebih cepat membangun *pipeline* dari data mentah sampai prediksi atau *alert* operasional.

C.1 Template koneksi MQTT

Bagian ini sebaiknya menyediakan dua pola dasar:

- **publisher sederhana**
- **subscriber sederhana**

Elemen yang wajib ada:

- koneksi ke broker,
- nama topik,
- fungsi `on_connect`,
- fungsi `on_message`,
- *publish* payload JSON,
- *decode* payload JSON.

Template ini penting agar pembaca bisa cepat:

- mengirim data telemetri,
- menerima pesan atau *alert*,
- menghubungkan sensor ke *pipeline* Python.

Contoh publisher MQTT sederhana

```
import json
import time
import paho.mqtt.client as mqtt

BROKER = "localhost"
PORT = 1883
TOPIC = "iot/sensor/suhu"

client = mqtt.Client()

client.connect(BROKER, PORT, 60)

payload = {
```

```
    "device_id": "node-01",
    "temperature": 29.5,
    "humidity": 70
}

client.publish(TOPIC, json.dumps(payload))
print("Data terkirim:", payload)

client.disconnect()
```

Contoh subscriber MQTT sederhana

```
import json
import paho.mqtt.client as mqtt

BROKER = "localhost"
PORT = 1883
TOPIC = "iot/sensor/suhu"

def on_connect(client, userdata, flags, rc):
    print("Terhubung ke broker, rc =", rc)
    client.subscribe(TOPIC)

def on_message(client, userdata, msg):
    payload = json.loads(msg.payload.decode())
    print("Pesan masuk:", payload)

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect(BROKER, PORT, 60)
client.loop_forever()
```

C.2 Template loader CSV

Banyak pembaca akan memulai dari **file CSV** sebelum masuk ke data *streaming*. Karena itu, lampiran ini perlu memberi pola yang sangat jelas dan mudah diulang.

Template minimal yang sebaiknya tersedia:

- membaca file dengan `pandas`,
- *parse timestamp*,
- mengurutkan data,
- memeriksa *missing value*,
- membuat ringkasan statistik,
- menyimpan file hasil pembersihan.

Template ini akan berguna hampir di semua topik, terutama:

- eksplorasi data,
- pembersihan data,

- *feature engineering*,
- evaluasi model.

Contoh loader CSV dasar

```
import pandas as pd

df = pd.read_csv("data_sensor.csv")

df["timestamp"] = pd.to_datetime(df["timestamp"], errors="coerce")
df = df.dropna(subset=["timestamp"])
df = df.sort_values("timestamp")

print(df.isna().sum())
print(df.describe(include="all"))

df.to_csv("data_sensor_bersih.csv", index=False)
```

C.3 Template preprocessing

Bagian *preprocessing* perlu berisi kumpulan fungsi pendek yang bisa dipakai ulang di banyak bab. Fokusnya adalah membersihkan, merapikan, dan membentuk data agar siap dianalisis atau dimasukkan ke model.

Blok yang sebaiknya ada:

- validasi kolom,
- konversi tipe numerik,
- *resampling*,
- imputasi sederhana,
- *lag feature*,
- *moving average*,
- *rolling standard deviation*,
- fitur waktu seperti `hour`, `day_of_week`, dan `is_weekend`.

Ini penting karena data IoT biasanya **heterogen**, tidak selalu rapi, dan mudah menghasilkan prediksi keliru bila tidak melalui tahap validasi, pengurutan, agregasi, dan pembersihan terlebih dahulu.

Contoh fungsi preprocessing singkat

```
import pandas as pd

def preprocess_iot(df):
    required_cols = ["timestamp", "value"]
    for col in required_cols:
        if col not in df.columns:
            raise ValueError(f"Kolom {col} tidak ditemukan")

    df["timestamp"] = pd.to_datetime(df["timestamp"], errors="coerce")
    df["value"] = pd.to_numeric(df["value"], errors="coerce")
```

```
    df = df.dropna(subset=["timestamp"])
    df = df.sort_values("timestamp")
    df = df.set_index("timestamp")

    df["value"] = df["value"].interpolate()
    df["lag_1"] = df["value"].shift(1)
    df["ma_3"] = df["value"].rolling(3).mean()
    df["std_3"] = df["value"].rolling(3).std()

    df["hour"] = df.index.hour
    df["day_of_week"] = df.index.dayofweek
    df["is_weekend"] = df["day_of_week"].isin([5, 6]).astype(int)

    return df.dropna()
```

C.4 Template training model

Bagian ini harus langsung **siap pakai**. Hindari template yang terlalu abstrak. Pembaca perlu melihat pola inti yang selalu berulang:

- memuat `X_train` dan `y_train`,
- memilih model,
- menjalankan `fit()`,
- menyimpan model dengan `joblib`,
- menyimpan metadata sederhana.

Model contoh yang cocok untuk dimasukkan:

- `LinearRegression`
- `RandomForestClassifier`
- `RandomForestRegressor`
- `IsolationForest`

Model-model ini relevan karena mencakup:

- *supervised classification*,
- *supervised regression*,
- deteksi anomali,
- dasar *forecasting* atau prediksi numerik.

Contoh training model regresi

```
import joblib
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)

joblib.dump(model, "model_regresi.pkl")
```

```
metadata = {
    "model_name": "RandomForestRegressor",
    "features": list(X_train.columns)
}
print(metadata)
```

Contoh training model klasifikasi

```
import joblib
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

joblib.dump(model, "model_klasifikasi.pkl")
```

Contoh deteksi anomali

```
from sklearn.ensemble import IsolationForest

model = IsolationForest(random_state=42)
model.fit(X_train)

anomali = model.predict(X_train)   # -1 anomali, 1 normal
```

C.5 Template evaluasi

Bagian evaluasi sebaiknya dibagi tegas menjadi dua kelompok: **klasifikasi** dan **regresi/forecasting**. Ini membantu pembaca memilih metrik yang benar sesuai jenis masalah.

Untuk klasifikasi

Metrik yang sebaiknya ditampilkan:

- accuracy,
- precision,
- recall,
- F1-score,
- confusion matrix.

Contoh evaluasi klasifikasi

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix

y_pred = model.predict(X_test)

print("Accuracy :", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred,
average="weighted"))
print("Recall   :", recall_score(y_test, y_pred, average="weighted"))
print("F1-score :", f1_score(y_test, y_pred, average="weighted"))
```

```
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred))
```

Untuk regresi / forecasting

Metrik yang sebaiknya ditampilkan:

- MAE,
- RMSE,
- plot aktual vs prediksi.

Contoh evaluasi regresi

```
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

y_pred = model.predict(X_test)

print("MAE :", mean_absolute_error(y_test, y_pred))
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))

plt.plot(y_test.values[:50], label="Aktual")
plt.plot(y_pred[:50], label="Prediksi")
plt.legend()
plt.show()
```

C.6 Template inferensi

Bagian inferensi harus menjadi jembatan antara **model** dan **sistem operasional**. Di sinilah pembaca mulai melihat bagaimana model dipakai dalam alur nyata, misalnya menerima *payload* dari sensor, membentuk satu record, memanggil model, lalu mengembalikan label atau skor.

Pola yang sebaiknya ada:

- `predict_from_payload()`,
- `predict_one_record()`,
- membentuk `DataFrame` satu baris,
- memanggil model,
- mengembalikan label atau skor,
- menyimpan hasil inferensi.

Bagian ini sangat penting untuk:

- *pipeline* end-to-end,
- sistem *alert*,
- integrasi MQTT ke model,
- proyek akhir.

Contoh inferensi dari satu record

```
import pandas as pd
import joblib

model = joblib.load("model_regresi.pkl")

def predict_one_record(record: dict):
    df = pd.DataFrame([record])
    pred = model.predict(df)[0]
    return pred

data_baru = {
    "temperature": 30.2,
    "humidity": 68,
    "pressure": 1008
}

hasil = predict_one_record(data_baru)
print("Hasil prediksi:", hasil)
```

Contoh inferensi dari payload JSON

```
import json
import pandas as pd
import joblib

model = joblib.load("model_klasifikasi.pkl")

def predict_from_payload(payload_json: str):
    payload = json.loads(payload_json)
    df = pd.DataFrame([payload])
    pred = model.predict(df)[0]
    return pred
```

Contoh simpan hasil inferensi

```
hasil = {
    "device_id": "node-01",
    "prediction": "normal"
}

pd.DataFrame([hasil]).to_csv("hasil_inferensi.csv", mode="a",
header=False, index=False)
```

Penutup Appendix C

Secara keseluruhan, **Appendix C** perlu menjadi **tool box teknis** yang dapat langsung digunakan ulang oleh pembaca. Nilai utamanya bukan pada kompleksitas kode, tetapi pada **keterkaitannya** dalam banyak konteks: mulai dari membaca file, membersihkan data, membangun model, mengevaluasi hasil, hingga menjalankan inferensi pada sistem operasional.

Dengan lampiran seperti ini, pembaca tidak perlu menulis ulang pola kode dasar dari nol. Mereka cukup mengambil template yang relevan, menyesuaikan nama kolom,

model, atau topik MQTT, lalu melanjutkan ke eksperimen dan implementasi yang lebih bermakna.

Appendix D. Template Generator Synthetic Data

Appendix ini penting karena **data sintetis adalah alat kerja**, bukan pelengkap. Dalam praktik, pembaca sering belum punya data lapangan yang cukup baik. Karena itu, generator sintetis dibutuhkan untuk:

- **belajar lebih cepat,**
- **menguji pipeline,**
- **menyisipkan anomali,**
- **mensimulasikan drift,**
- **melakukan stress testing.**

Pesan utamanya jelas: **jangan menunggu data sempurna untuk memulai menguji sistem**. Jika model, dashboard, atau pipeline gagal pada data sintetis yang terkontrol, besar kemungkinan ia juga rapuh di kondisi nyata.

Generator suhu dan kelembaban

Generator ini sebaiknya meniru pola dasar lingkungan:

- suhu naik siang, turun malam,
- kelembaban bergerak sedikit berlawanan,
- ada noise kecil,
- timestamp teratur,
- opsional label **nyaman / tidak nyaman**.

Cocok untuk:

- *smart room monitoring*,
- klasifikasi kondisi ruangan,
- dashboard suhu dan kelembaban.

```
import numpy as np, pandas as pd

t = pd.date_range("2026-01-01", periods=24*3, freq="H")
h = t.hour.to_numpy()

temp = 26 + 4*np.sin(2*np.pi*(h-6)/24) + np.random.normal(0,
0.3, len(t))
humid = 75 - 6*np.sin(2*np.pi*(h-6)/24) + np.random.normal(0,
1.0, len(t))

df = pd.DataFrame({"timestamp": t, "temperature": temp,
"humidity": humid})
df["label"] = np.where((df["temperature"] < 30) &
(df["humidity"] < 75), "nyaman", "tidak_nyaman")
df.to_csv("synthetic_room.csv", index=False)
```

Generator energi

Generator energi harus meniru **ritme beban**, bukan sekadar angka acak.

Pola minimum:

- jam kerja lebih tinggi,
- akhir pekan lebih rendah,
- malam lebih rendah,
- lonjakan abnormal opsional.

Cocok untuk:

- *forecasting* beban,
- analitik energi,
- alert konsumsi tidak normal.

```
import numpy as np, pandas as pd

t = pd.date_range("2026-01-01", periods=24*7, freq="H")
hour = t.hour.to_numpy()
day = t.dayofweek.to_numpy()

base = np.where((hour >= 8) & (hour <= 17), 800, 300)
weekend = np.where(day >= 5, 0.7, 1.0)
power = base * weekend + np.random.normal(0, 25, len(t))
power[50:53] += 400  # lonjakan

df = pd.DataFrame({"timestamp": t, "power_watt": power})
df.to_csv("synthetic_energy.csv", index=False)
```

Generator getaran mesin

Bagian ini penting untuk **predictive maintenance**. Yang perlu ditiru bukan hanya sinyal normal, tetapi juga transisi ke kondisi buruk.

Elemen yang perlu ada:

- sinyal normal,
- warning,
- fault,
- amplitudo meningkat,
- spike sesaat,
- noise lebih besar.

Cocok untuk:

- ekstraksi fitur jendela,
- anomaly detection,
- klasifikasi fault dasar.

```
import numpy as np, pandas as pd

x = np.arange(1000)
normal = 0.5*np.sin(2*np.pi*x/50) + np.random.normal(0, 0.05,
len(x))
fault = 1.2*np.sin(2*np.pi*x/50) + np.random.normal(0, 0.15,
len(x))
fault[200] += 2.0
fault[700] += 1.5

df = pd.DataFrame({"t": x, "normal": normal, "fault": fault})
df.to_csv("synthetic_vibration.csv", index=False)
```

Injector noise, outlier, missing value, drift

Inilah bagian paling penting. Data sintetis baru berguna penuh jika pembaca bisa **merusaknya secara terkontrol**.

Fungsinya:

- **noise** untuk uji robustitas,
- **outlier** untuk uji threshold dan anomaly detection,
- **missing value** untuk uji imputasi dan fallback,
- **drift** untuk uji perubahan distribusi dan turunnya performa model.

Urutan yang harus diajarkan:

- buat data normal dulu,
- baru injeksi gangguan,
- bandingkan sebelum dan sesudah,
- simpan versi normal dan versi stres.

```
import pandas as pd, numpy as np

def inject_noise(s, scale=0.5):
    return s + np.random.normal(0, scale, len(s))

def inject_missing(s, idx):
    s = s.copy()
    s.iloc[idx] = np.nan
    return s

def inject_drift(s, start, step=0.02):
    s = s.copy()
    for i in range(start, len(s)):
        s.iloc[i] += (i - start) * step
    return s

Contoh pakai:
df = pd.read_csv("synthetic_room.csv")
df["temp_noisy"] = inject_noise(df["temperature"], 0.3)
df["temp_missing"] = inject_missing(df["temperature"], [5, 6,
7])
df["temp_drift"] = inject_drift(df["temperature"], 30, 0.03)
df.to_csv("synthetic_room_stress.csv", index=False)
```

Penekanan

Appendix ini harus menanamkan satu prinsip penting:

- **data sintetis membantu membangun sistem,**
- **data stres membantu menguji ketahanannya.**

Dengan demikian, Appendix D tidak hanya membantu pembaca membuat data, tetapi juga **menguji apakah pipeline, model, dan dashboard tetap bekerja saat kondisi lapangan memburuk**.

Appendix E. Glosarium IoT, ML, dan Analytics

Istilah	Definisi Singkat	Konteks dalam Buku
Accuracy	Persentase prediksi yang benar dari seluruh prediksi.	Dipakai untuk mengevaluasi model klasifikasi seperti **normal / warning / fault** atau **nyaman / tidak nyaman**.
Actuator	Komponen yang melakukan aksi fisik berdasarkan perintah sistem, misalnya menyalakan kipas, pompa, atau relay.	Dalam arsitektur IoT, actuator adalah pasangan alami dari sensor pada sistem otomatis.
Alert	Peringatan yang dikirim sistem ketika kondisi tertentu dianggap penting atau tidak normal.	Dalam buku ini, alert dapat berasal dari **threshold**, **skor anomali**, atau **status prediksi model**.
Anomaly Detection	Teknik untuk mendeteksi data, pola, atau perilaku yang menyimpang dari kondisi normal.	Dipakai pada kasus seperti **arus tidak normal**, **getaran mesin aneh**, atau **ruangan dengan pola sensor yang menyimpang**.
API	*Application Programming Interface*, antarmuka yang memungkinkan dua sistem saling berkomunikasi.	Bisa dipakai untuk menghubungkan dashboard, database, model, atau sistem notifikasi.
ARIMA	Model statistik klasik untuk *time-series forecasting* berbasis tren, autoregresi, dan moving average.	Disebut sebagai jalur lanjutan, tetapi bukan fokus utama buku ini.
Autoencoder	Model jaringan saraf yang belajar merekonstruksi data input dan sering dipakai untuk *unsupervised anomaly detection*.	Cocok untuk pembaca lanjut yang ingin mendeteksi anomali dari pola rekonstruksi.
Automasi	Proses menjalankan tugas secara otomatis tanpa intervensi manual terus-menerus.	Seluruh pipeline IoT-ML dalam buku ini diarahkan menuju automasi: ingest, preprocessing, inferensi, dan alert.
Bandwidth	Kapasitas jalur komunikasi untuk mengirim data dalam periode tertentu.	Menjadi pertimbangan penting dalam IoT, terutama saat memilih antara kirim data mentah ke cloud atau inferensi di edge.

Baseline	Metode pembanding paling sederhana yang digunakan sebelum memakai model yang lebih kompleks.	Contohnya **last value**, **moving average**, atau **threshold statis**.
Batch Retraining	Pelatihan ulang model secara berkala menggunakan kumpulan data baru yang dikumpulkan selama periode tertentu.	Dipakai untuk menjaga model tetap relevan saat terjadi **drift**.
Big Data	Kumpulan data besar dan kompleks yang memiliki karakter seperti volume, velocity, variety, veracity, dan value.	Data IoT sering dikaitkan dengan *big data* karena sensor dapat menghasilkan aliran data terus-menerus.
Broker	Komponen perantara dalam arsitektur *publish/subscribe* yang menerima dan mendistribusikan pesan.	Dalam buku ini, **Mosquitto** berperan sebagai broker MQTT.
Buffer	Penyimpanan sementara untuk beberapa data terakhir yang sering dipakai dalam pembentukan fitur seperti *lag* dan *moving average*.	Sangat berguna dalam inferensi real-time dan preprocessing otomatis.
Classification	Tugas machine learning untuk memprediksi kategori atau kelas tertentu.	Contoh: **nyaman / tidak nyaman**, **normal / warning / fault**.
Cloud	Infrastruktur komputasi dan penyimpanan yang berjalan di pusat data jarak jauh dan diakses melalui jaringan.	Dalam konteks buku, cloud cocok untuk pelatihan model besar, penyimpanan skala besar, atau integrasi multi-lokasi.
CO_2	Karbon dioksida, salah satu variabel lingkungan yang sering dipakai sebagai indikator kualitas udara atau okupansi ruangan.	Pada studi kasus **Smart Room Monitoring**, CO_2 menjadi fitur penting untuk klasifikasi kenyamanan.
Confusion Matrix	Tabel evaluasi untuk melihat jumlah prediksi benar dan salah pada model klasifikasi.	Dipakai untuk memahami apakah model sering salah mengklasifikasi `warning` sebagai `normal`, atau `fault` lolos terdeteksi.
Contamination	Perkiraan proporsi anomali dalam dataset yang dipakai oleh beberapa model anomaly detection seperti **Isolation Forest**.	Membantu model menentukan seberapa banyak data yang dianggap menyimpang.

CSV	*Comma-Separated Values*, format file teks sederhana untuk menyimpan data tabular.	Sangat sering dipakai di buku ini untuk latihan awal, debugging, dan pertukaran data antar-tahap.
Dashboard	Tampilan visual yang menyajikan data, status, grafik, dan indikator utama secara ringkas.	Dalam buku ini, **Grafana** menjadi dashboard utama untuk data sensor, prediksi, dan alert.
Data Drift	Perubahan distribusi data input dari waktu ke waktu dibandingkan baseline awal.	Perlu dimonitor karena dapat menyebabkan penurunan kualitas model.
Data Lake	Penyimpanan data besar yang menampung data mentah, semi-terstruktur, dan terstruktur dalam jumlah besar.	Disebut sebagai konsep lanjutan ketika sistem mulai tumbuh lebih besar dari prototipe lokal.
Data Logger	Perangkat atau sistem yang merekam data sensor secara periodik.	Berguna untuk membangun dataset historis sebelum masuk ke analitik dan machine learning.
Data Quality	Tingkat kualitas data dilihat dari kelengkapan, konsistensi, validitas, dan reliabilitasnya.	Kualitas data sangat menentukan mutu hasil model IoT-ML.
Dataset	Kumpulan data yang disusun untuk analisis, pelatihan model, atau evaluasi.	Buku ini memakai dataset nyata maupun sintetis untuk semua studi kasus.
DBSCAN	Algoritma clustering berbasis kepadatan yang dapat menemukan cluster dan noise tanpa harus menentukan jumlah cluster di awal.	Cocok untuk menemukan pola operasi dan titik yang menyendiri pada data sensor.
Deployment	Proses menempatkan model atau sistem agar benar-benar dijalankan di lingkungan target.	Dalam buku ini, deployment mini bisa dilakukan di laptop, Raspberry Pi, atau *edge gateway*.
Descriptive Analytics	Analitik yang berfokus menjelaskan apa yang terjadi berdasarkan data historis.	Contoh: melihat suhu rata-rata ruangan, pola konsumsi energi, atau jumlah alert per hari.
Diagnostic Analytics	Analitik yang berfokus menjelaskan mengapa sesuatu terjadi.	Dipakai saat mencari hubungan antara suhu, CO_2, daya, atau fitur lain yang menjelaskan kondisi sistem.

Digital Twin	Representasi digital dari sistem fisik yang dapat dipakai untuk simulasi, monitoring, dan optimasi.	Disebut sebagai arah pengembangan lanjutan setelah pembaca menguasai fondasi IoT-ML.
Docker	Platform untuk membungkus aplikasi dan layanan ke dalam container agar mudah dipindah dan dijalankan ulang.	Dalam buku ini, Docker bersifat opsional untuk **reproducibility** dan mempermudah setup Mosquitto, InfluxDB, dan Grafana.
Drift	Perubahan bertahap pada data atau hubungan data-target yang membuat model lama makin tidak relevan.	Menjadi topik penting dalam operasional sistem IoT-ML.
Edge AI	Pendekatan menjalankan inferensi model di dekat sumber data, misalnya di gateway, mini PC, atau Raspberry Pi.	Berguna untuk menurunkan latensi, mengurangi beban jaringan, dan meningkatkan respons lokal.
Edge Device	Perangkat komputasi yang berada dekat sumber data dan dapat menjalankan tugas pengumpulan data, preprocessing, atau inferensi.	Contoh dalam buku: **Raspberry Pi**, mini PC, atau gateway lokal.
Elbow Method	Teknik untuk membantu memilih jumlah cluster yang masuk akal pada K-Means dengan melihat perubahan *inertia*.	Digunakan saat pembaca mencoba K-Means pada data ruangan atau energi.
Ensemble Learning	Pendekatan menggabungkan banyak model untuk meningkatkan performa dan kestabilan prediksi.	**Random Forest** dan **Gradient Boosting** adalah contoh model ensemble yang dibahas dalam buku ini.
ESP32	Mikrokontroler populer dengan Wi-Fi/Bluetooth yang sering dipakai untuk proyek IoT murah dan fleksibel.	Dalam buku ini, ESP32 menjadi salah satu target praktis untuk akuisisi data atau eksperimen edge/TinyML ringan.
Evaluation	Proses menilai kualitas model menggunakan metrik tertentu.	Evaluasi adalah tahap penting setelah training model, baik untuk klasifikasi, regresi, anomaly detection, maupun forecasting.

Explain Ability	Kemampuan untuk menjelaskan mengapa model menghasilkan prediksi tertentu.	Dalam buku ini, explain ability ringan dicapai misalnya dengan **feature importance**.
Feature	Variabel input yang dipakai model untuk belajar atau melakukan prediksi.	Contoh: suhu, kelembaban, CO_2, lag daya, rolling mean, RMS getaran.
Feature Engineering	Proses membuat atau memilih fitur yang lebih informatif dari data mentah.	Topik ini sangat penting di buku, misalnya saat membuat **lag**, **moving average**, dan statistik jendela.
Feature Importance	Ukuran yang menunjukkan seberapa besar pengaruh suatu fitur terhadap keputusan model.	Dipakai untuk explain ability ringan pada model seperti Random Forest.
Federated Learning	Pendekatan pelatihan model lintas banyak perangkat/lokasi tanpa mengumpulkan seluruh data mentah ke satu server pusat.	Disebut sebagai topik lanjutan setelah pembaca menguasai fondasi pipeline dan deployment.
F1 Score	Metrik klasifikasi yang menggabungkan **precision** dan **recall** dalam satu angka.	Sangat berguna ketika kelas tidak seimbang, misalnya anomali jauh lebih sedikit daripada kondisi normal.
False Negative	Kasus ketika sistem gagal mendeteksi kondisi positif yang sebenarnya ada.	Dalam anomaly detection atau fault detection, false negative bisa sangat berbahaya karena masalah nyata lolos.
False Positive	Kasus ketika sistem menandai kondisi positif padahal sebenarnya normal.	Terlalu banyak false positive dapat membuat operator kehilangan kepercayaan pada alert.
Fault	Kondisi gangguan atau kerusakan pada sistem/perangkat.	Dalam buku ini sering dipakai sebagai salah satu label pada klasifikasi kondisi perangkat.
Fog Computing	Lapisan komputasi antara perangkat dan cloud yang membantu pemrosesan lebih dekat ke sumber data.	Disebut untuk membantu pembaca memahami peran edge, fog, dan cloud secara praktis.
Forecasting	Proses memprediksi nilai masa depan berdasarkan data historis berurutan terhadap waktu.	Dipakai pada studi kasus suhu ruang, energi, dan *time-series forecasting*.

Gateway	Perangkat perantara yang menghubungkan sensor atau node lapangan ke jaringan, broker, atau sistem analitik.	Dalam arsitektur buku, gateway bisa berupa Raspberry Pi, mini PC, atau sistem lokal lain.
Grafana	Platform dashboard open source untuk visualisasi data *time series* dan observability.	Menjadi dashboard utama di banyak hands-on, studi kasus, dan proyek akhir.
Gradient Boosting	Metode ensemble yang membangun model secara bertahap untuk memperbaiki kesalahan model sebelumnya.	Dibahas sebagai opsi lanjutan pada supervised learning tabular.
Ground Truth	Label atau kondisi sebenarnya yang dipakai sebagai acuan evaluasi model.	Pada data sintetis, ground truth sering dibuat manual untuk membantu evaluasi anomaly detection atau klasifikasi.
Holdout Set	Bagian data yang disisihkan dan tidak digunakan saat pelatihan untuk menguji performa model secara jujur.	Sangat penting dalam evaluasi model agar hasil tidak terlalu optimistis.
HTTP REST	Gaya komunikasi antar sistem melalui permintaan HTTP berbasis sumber daya dan *endpoint*.	Dalam buku ini, REST disebut sebagai salah satu protokol praktis selain MQTT dan serial.
Hub-and-Spoke	Arsitektur komunikasi di mana banyak perangkat mengirim data ke satu pusat distribusi atau koordinasi.	MQTT broker sering berperan sebagai pusat hub-and-spoke dalam sistem IoT praktis.
Humidity	kelembaban udara, biasanya dinyatakan dalam persen.	Menjadi variabel penting pada smart room monitoring dan generator data lingkungan.
Imputasi	Proses mengisi nilai yang hilang pada dataset dengan pendekatan tertentu.	Dipakai saat data sensor memiliki *missing value* dan pembaca perlu menjaga pipeline tetap berjalan.
Inference	Proses menjalankan model yang sudah dilatih untuk memprediksi data baru.	Dalam sistem IoT nyata, inferensi jauh lebih sering terjadi daripada training.
InfluxDB	Database *time series* open source yang sangat cocok untuk data sensor dan integrasi dengan Grafana.	Dipakai dalam banyak hands-on untuk menyimpan telemetry dan hasil inferensi.

Isolation Forest	Algoritma anomaly detection yang mendeteksi anomali berdasarkan kemudahan suatu titik data untuk "diisolasi".	Menjadi model anomaly detection utama yang praktis dalam banyak bagian buku.
IIoT	*Industrial Internet of Things*, penerapan IoT pada lingkungan industri, pabrik, utilitas, dan operasi teknis lain.	Banyak studi kasus maintenance dan energi di buku ini berada dekat dengan konteks IIoT.
IoE	*Internet of Everything*, konsep yang lebih luas dari IoT karena mencakup orang, proses, data, dan benda.	Disebut di bagian awal untuk membedakan IoT, IIoT, dan IoE.
IoT	*Internet of Things*, jaringan perangkat fisik yang dapat mengumpulkan, bertukar, dan memanfaatkan data melalui konektivitas digital.	Menjadi fondasi utama seluruh buku ini.
Jupyter	Lingkungan notebook interaktif yang sangat cocok untuk eksplorasi data, visualisasi, dan eksperimen awal.	Dipakai dalam buku ini untuk eksplorasi dan pembelajaran, sebelum logika dipindah ke skrip operasional.
K-Means	Algoritma clustering yang membagi data menjadi K kelompok berdasarkan kemiripan jarak.	Digunakan untuk menemukan pola operasi atau segmentasi kondisi sensor.
Kafka	Platform *distributed event streaming* yang dipakai untuk aliran data skala lebih besar dan lebih kompleks.	Disebut sebagai langkah lanjut saat pipeline sederhana sudah tidak cukup.
Keras	API jaringan saraf tingkat tinggi yang memudahkan pembangunan model deep learning di atas TensorFlow.	Dipakai pada bagian LSTM sederhana dan konversi ke TensorFlow Lite.
KPI	*Key Performance Indicator*, metrik utama untuk memantau kinerja sistem atau proses.	Dalam konteks IoT-ML, KPI bisa berupa jumlah alert, MAE, latensi inferensi, atau persentase payload valid.
Lag Feature	Fitur yang dibentuk dari nilai masa lalu suatu variabel pada beberapa langkah sebelumnya.	Sangat penting pada forecasting, anomaly detection tabular, dan machine learning berbasis *time series*.

Latency	Waktu tunda antara kejadian, pengiriman data, pemrosesan, dan keluarnya hasil.	Menjadi alasan penting mengapa inferensi di edge sering dipilih.
Linear Regression	Model regresi sederhana yang memprediksi target numerik berdasarkan hubungan linear dengan fitur input.	Menjadi baseline penting untuk prediksi suhu, energi, dan forecasting tabular.
LSTM	*Long Short-Term Memory*, jenis jaringan saraf berulang yang dirancang untuk mempelajari dependensi dalam urutan data.	Dibahas sebagai model deep learning ringan untuk forecasting *time series*.
MAE	*Mean Absolute Error*, rata-rata kesalahan absolut antara nilai aktual dan nilai prediksi.	Dipakai untuk mengevaluasi model regresi dan forecasting.
Machine Learning	Cabang kecerdasan buatan yang memungkinkan sistem belajar dari data untuk melakukan prediksi, klasifikasi, atau deteksi pola.	Menjadi inti pembahasan buku pada bagian analitik dan model.
Missing Value	Nilai yang hilang pada dataset karena sensor gagal kirim, logger bermasalah, atau proses lain.	Menjadi salah satu masalah klasik data sensor yang harus ditangani melalui validasi atau imputasi.
Model Drift	Penurunan performa model dari waktu ke waktu akibat perubahan data atau hubungan data-target.	Menjadi topik operasional penting saat sistem dipakai terus-menerus.
Model Registry	Sistem untuk menyimpan, melacak, dan mengelola versi model secara formal.	Disebut sebagai langkah lanjut saat pembaca masuk ke MLOps yang lebih serius.
Monitoring	Aktivitas memantau data, performa sistem, status model, dan alert secara berkelanjutan.	Menjadi bagian inti dari pipeline end-to-end dan operasional proyek akhir.
Mosquitto	Broker MQTT open source yang ringan dan populer.	Menjadi broker utama yang dipakai sepanjang buku untuk semua latihan MQTT.
Moving Average	Rata-rata bergerak dari beberapa data terakhir yang dipakai untuk smoothing atau baseline prediksi.	Dipakai dalam forecasting, feature engineering, dan analisis tren lokal.

MQTT	Protokol pesan ringan berbasis *publish/subscribe* yang sangat cocok untuk IoT.	Menjadi jalur utama pengiriman telemetry sensor ke subscriber Python.
Node-RED	Alat pemrograman visual berbasis alur yang memudahkan integrasi IoT dan otomasi ringan.	Dipakai sebagai komponen opsional untuk integrasi cepat dan eksperimen visual.
Noise	Variasi acak pada data yang tidak mencerminkan pola utama sistem.	Perlu dipahami karena data sensor nyata hampir selalu mengandung noise.
Normalization	Proses mengubah skala data agar lebih seragam.	Berguna pada beberapa model machine learning dan pada proses *preprocessing*.
One-Class SVM	Model anomaly detection yang mempelajari batas wilayah data normal dan menandai titik di luar wilayah itu sebagai anomali.	Dibahas secara ringkas sebagai alternatif klasik anomaly detection.
Open Source	Perangkat lunak dengan kode sumber terbuka yang dapat dipelajari, digunakan, dimodifikasi, dan dikembangkan bersama komunitas.	Seluruh buku ini berfokus pada stack open source untuk alasan biaya, fleksibilitas, dan kedaulatan teknologi.
Outlier	Titik data yang jauh berbeda dari mayoritas data lain.	Outlier dapat menjadi gejala anomali, kesalahan sensor, atau kejadian langka yang penting.
Overfitting	Kondisi ketika model terlalu menyesuaikan diri pada data latihan sehingga performanya buruk pada data baru.	Perlu dihindari melalui validasi yang benar dan evaluasi yang jujur.
Payload	Isi pesan atau data yang dikirim dalam komunikasi digital, misalnya melalui MQTT atau HTTP.	Dalam buku ini, payload biasanya berisi JSON telemetry sensor.
PCA	*Principal Component Analysis*, teknik reduksi dimensi yang merangkum beberapa fitur menjadi komponen yang lebih sedikit.	Dipakai untuk merangkum beberapa sensor atau eksplorasi struktur data.
Pipeline	Rangkaian tahapan terhubung dari data masuk sampai hasil akhir keluar.	Misalnya: sensor → MQTT → Python → preprocessing → model → storage → dashboard.

Power Watt	Besaran daya listrik yang biasa dipakai sebagai variabel utama dalam smart energy analytics.	Menjadi salah satu fitur penting dalam studi kasus energi dan capstone project.
Precision	Proporsi prediksi positif yang benar dari seluruh prediksi positif.	Penting bila *false positive* mahal, misalnya alert palsu terlalu sering.
Predictive Analytics	Analitik yang berfokus pada prediksi kejadian atau nilai yang mungkin terjadi di masa depan.	Meliputi forecasting, klasifikasi risiko, dan prediksi status perangkat.
Predictive Maintenance	Pendekatan maintenance yang memanfaatkan data untuk mendeteksi gejala awal kerusakan sebelum kegagalan besar terjadi.	Menjadi salah satu studi kasus utama buku ini.
Preprocessing	Tahap membersihkan, memvalidasi, menyusun, dan menyiapkan data sebelum dipakai model.	Menjadi fondasi semua bab analitik dan machine learning.
Prescriptive Analytics	Analitik yang tidak hanya memprediksi, tetapi juga membantu menentukan tindakan yang sebaiknya diambil.	Disebut sebagai level analitik lebih lanjut setelah descriptive, diagnostic, dan predictive.
Publisher	Klien yang mengirim pesan ke topik tertentu dalam arsitektur MQTT.	Dalam buku ini, publisher dapat berupa sensor nyata, simulator Python, atau synthetic generator.
Python	Bahasa pemrograman utama yang dipakai dalam buku ini untuk data analytics, machine learning, MQTT, dan otomasi pipeline.	Menjadi alat inti dari hampir seluruh hands-on.
Random Forest	Model ensemble berbasis banyak pohon keputusan yang dipakai untuk klasifikasi maupun regresi.	Menjadi model penting untuk supervised learning dan explainability ringan.
Raspberry Pi	Komputer mini berbasis Linux yang sering dipakai sebagai edge device atau gateway IoT.	Menjadi target praktis untuk deployment mini dan inferensi lokal.
Recall	Proporsi kasus positif yang berhasil dideteksi dari seluruh kasus positif yang sebenarnya ada.	Sangat penting untuk anomaly detection dan fault detection agar masalah tidak lolos.
Regression	Tugas machine learning untuk memprediksi nilai numerik kontinu.	Dipakai untuk prediksi suhu, energi, atau variabel sensor lain.

Resampling	Proses mengubah frekuensi waktu data *time series*, misalnya dari 1 menit menjadi 10 menit.	Dipakai untuk merapikan interval data sensor agar konsisten.
RMSE	*Root Mean Squared Error*, akar dari rata-rata kuadrat kesalahan prediksi.	Dipakai untuk mengevaluasi model regresi dan forecasting, terutama jika ingin menghukum kesalahan besar lebih kuat.
Rolling Window	Jendela bergerak yang dipakai untuk menghitung statistik lokal seperti mean, std, min, dan max.	Sangat penting dalam *feature engineering* data sensor.
RMS	*Root Mean Square*, ukuran energi atau besarnya sinyal yang sering dipakai pada getaran dan predictive maintenance.	Menjadi fitur penting pada studi kasus maintenance.
Scaler / Scaling	Proses mengubah skala fitur agar lebih seimbang, misalnya dengan *standardization*.	Sering dipakai pada clustering, One-Class SVM, dan model tertentu.
scikit-learn	Library machine learning Python yang ringan, matang, dan sangat cocok untuk data tabular dan pembelajaran terstruktur.	Menjadi platform utama supervised learning, anomaly detection, dan evaluasi model dalam buku ini.
Sensor	Komponen yang mengukur besaran fisik atau lingkungan, seperti suhu, kelembaban, arus, atau getaran.	Menjadi titik awal seluruh pipeline IoT dalam buku ini.
Serial	Komunikasi data secara berurutan antarperangkat melalui jalur serial seperti UART/USB.	Disebut sebagai salah satu jalur komunikasi praktis selain MQTT dan HTTP REST.
Silhouette Score	Metrik untuk menilai seberapa baik hasil clustering berdasarkan kedekatan antar-anggota cluster dan jarak ke cluster lain.	Digunakan saat mengevaluasi K-Means atau hasil segmentasi sederhana.
SQLite	Database lokal ringan yang sangat cocok untuk prototipe, logging, dan penyimpanan sederhana.	Dipakai di buku ini sebagai alternatif storage lokal yang sangat praktis.
Standard Deviation (Std)	Ukuran seberapa menyebar data terhadap rata-ratanya.	Sering dipakai sebagai fitur rolling dan sebagai indikator variasi data sensor.

Storage	Lapisan penyimpanan data dalam arsitektur sistem.	Dalam buku ini, storage dapat berupa CSV, SQLite, atau InfluxDB.
Streaming	Alur data yang terus mengalir secara real-time atau mendekati real-time.	Data sensor MQTT adalah contoh streaming dalam praktik buku ini.
Subscriber	Klien yang menerima pesan dari topik tertentu dalam arsitektur MQTT.	Subscriber Python sering menjadi pusat logika pipeline IoT-ML dalam buku ini.
Supervised Learning	Pembelajaran mesin berbasis data yang sudah memiliki label atau target.	Digunakan untuk klasifikasi kondisi dan regresi prediksi nilai sensor.
Synthetic Data	Data buatan yang dibuat untuk meniru pola data nyata.	Dipakai untuk pembelajaran, fallback, pengujian pipeline, dan stress testing.
Telemetry	Data pengukuran yang dikirim dari perangkat lapangan ke sistem pengumpulan atau analitik.	Dalam buku ini, telemetry biasanya dikirim melalui MQTT dalam format JSON.
TensorFlow	Framework open source untuk machine learning dan deep learning.	Dipakai dalam buku untuk LSTM sederhana, TensorFlow Lite, dan eksperimen model neural network.
TensorFlow Lite	Format dan runtime ringan untuk menjalankan model TensorFlow/Keras secara lebih efisien di perangkat edge.	Menjadi jembatan praktis dari model laptop ke inferensi edge ringan.
Threshold	Batas nilai tertentu yang dipakai sebagai aturan sederhana untuk memicu status atau alert.	Menjadi baseline penting sebelum memakai model anomaly detection yang lebih kompleks.
Time Series	Data yang tersusun berurutan menurut waktu.	Sebagian besar data sensor dalam buku ini adalah time series.
TinyML	Penerapan machine learning pada perangkat sangat kecil dan sangat terbatas sumber dayanya, seperti mikrokontroler.	Dibahas sebagai arah lanjut setelah pembaca memahami edge AI dan model ringan.

Training	Proses melatih model menggunakan dataset agar model belajar pola atau hubungan tertentu.	Berbeda dari inferensi, training biasanya dilakukan di laptop, server, atau cloud.
Validation Set	Bagian data yang dipakai untuk mengevaluasi model selama pengembangan sebelum pengujian akhir.	Penting untuk memilih model dan parameter tanpa “mencuri” data test.
Virtual Environment	Lingkungan Python terisolasi untuk mengelola dependensi proyek secara rapi.	Sangat dianjurkan untuk semua praktik di buku ini.
Visualization	Penyajian data atau hasil model dalam bentuk visual seperti grafik, histogram, atau dashboard.	Menjadi bagian inti dari eksplorasi data, evaluasi model, dan operasional sistem.
Wearable	Perangkat yang dikenakan pada tubuh dan mengumpulkan data sensor, misalnya gerak, detak, atau aktivitas.	Menjadi salah satu kategori dataset publik untuk latihan lanjutan.
Windowing	Proses membagi data sinyal atau *time series* menjadi potongan-potongan jendela untuk ekstraksi fitur atau pelatihan model.	Sangat penting untuk predictive maintenance dan data sinyal.
XGBoost	Metode boosting yang sangat populer untuk data tabular dan sering memberi performa tinggi.	Disebut sebagai opsi singkat/lanjutan setelah pembaca menguasai model utama seperti Random Forest.

Appendix F. Referensi dan Proyek Open Source Lanjutan

Appendix ini berfungsi sebagai **peta belajar setelah buku selesai**. Jika bab utama membawa pembaca dari sensor, data, model, inferensi, hingga dashboard, maka bagian ini menjawab pertanyaan paling penting: **setelah ini, langkah berikutnya apa?**

Fokus appendix ini bukan mendorong pembaca langsung masuk ke sistem yang lebih rumit. Yang lebih penting adalah **memperkuat fondasi**, **memperluas proyek secara bertahap**, dan mulai mengenal ekosistem *open source* dengan arah yang jelas. Karena itu, appendix ini harus menjadi **panduan navigasi**, bukan sekadar daftar referensi.

Secara isi, bagian ini sebaiknya dibagi menjadi tiga kelompok utama:

- **referensi teknis inti**, untuk memperdalam stack yang dipakai di buku,
- **kategori proyek lanjutan**, untuk memilih arah pengembangan nyata,
- **jalur belajar tematik**, untuk membantu pembaca menentukan fokus berikutnya.

F.1 Prinsip menggunakan referensi lanjutan

Prinsip utamanya sederhana: **jangan pelajari semuanya sekaligus**. Cara terbaik memakai referensi lanjutan adalah mengikuti kebutuhan nyata.

- Jika sedang membangun *pipeline* data, fokus pada **Python, pandas, MQTT, dan database**.
- Jika sedang mengembangkan model, fokus pada **scikit-learn, TensorFlow/Keras, evaluasi, dan interpretasi model**.
- Jika sedang membangun dashboard operasional, fokus pada **InfluxDB, Grafana, logging, dan alert**.
- Jika ingin masuk ke edge deployment, fokus pada **Raspberry Pi, TensorFlow Lite, dan inferensi lokal**.
- Jika fondasi sudah matang dan sistem mulai membesar, barulah melihat **Kafka, Spark, Docker, dan MLOps**.

Intinya, **belajar berbasis kebutuhan jauh lebih efektif daripada belajar karena semua terlihat menarik**.

F.2 Referensi teknis utama

Referensi utama yang layak dijadikan pegangan adalah yang langsung terkait dengan stack buku ini.

- **Python, pandas, NumPy, matplotlib** untuk membaca data, membersihkan data, membangun fitur, dan visualisasi awal.
- **scikit-learn** untuk klasifikasi, regresi, deteksi anomali, evaluasi, dan *pipeline* dasar.
- **TensorFlow, Keras, dan TensorFlow Lite** untuk *deep learning* ringan, LSTM sederhana, dan inferensi di edge.
- **MQTT dan Mosquitto** untuk komunikasi data antar perangkat, topik, QoS, autentikasi, dan *broker operation*.

- **Node-RED** untuk integrasi visual, automasi cepat, dan *flow* berbasis aturan.
- **InfluxDB, SQLite, dan Grafana** untuk penyimpanan, history *time series*, dan dashboard operasional.
- **Raspberry Pi** untuk eksperimen edge, layanan lokal, inferensi ringan, dan integrasi sensor.

Yang perlu ditekankan: **dalam alat yang memang dipakai**, bukan semua alat yang tersedia.

F.3 Kategori proyek open source lanjutan

Appendix ini sebaiknya menawarkan arah proyek yang realistis dan relevan.

- **Smart room monitoring**
  Cocok untuk monitoring multi-ruangan, kenyamanan, ventilasi, dan dashboard gedung kecil.
- **Smart energy dan smart building**
  Cocok untuk *smart meter*, prediksi beban, efisiensi energi, dan deteksi konsumsi tidak normal.
- **Predictive maintenance dan industrial monitoring**
  Cocok untuk getaran, arus motor, deteksi anomali, klasifikasi gangguan, dan histori warning/fault.
- **Edge AI dan inferensi lokal**
  Cocok untuk deployment model ringan di Raspberry Pi, gateway cerdas, dan alert lokal tanpa bergantung pada cloud.
- **Synthetic telemetry dan simulator sensor**
  Cocok untuk pengujian *pipeline*, *stress testing*, bahan ajar, dan demonstrasi sistem saat hardware terbatas.

Yang penting, proyek dipilih bukan karena terlihat canggih, tetapi karena **bisa diselesaikan, diuji, dan memberi hasil nyata**.

F.4 Jalur pengembangan tematik

Setelah proyek akhir selesai, pembaca perlu memilih arah pendalaman.

- Bila tertarik pada **dashboard dan monitoring**, fokuskan pada Grafana, InfluxDB, alert, histori, dan metrik operasional.
- Bila tertarik pada **analitik data**, fokuskan pada pandas lanjutan, analisis *time series*, visualisasi, drift, dan interpretasi fitur.
- Bila tertarik pada **machine learning dan deployment**, fokuskan pada scikit-learn, TensorFlow/Keras, TensorFlow Lite, inferensi edge, dan versioning model.
- Bila ingin masuk ke **sistem streaming yang lebih besar**, fokuskan pada Kafka, Spark, arsitektur *event-driven*, dan MLOps.
- Bila tertarik pada **TinyML**, mulai dari inferensi ringan dan edge deployment, lalu bergerak ke perangkat yang lebih terbatas sumber dayanya.

Pesan pentingnya: **naik level hanya setelah fondasi Python, MQTT, dan pipeline ringan benar-benar kuat**.

F.5 Strategi memilih proyek lanjutan

Appendix ini perlu menutup bagian proyek dengan panduan yang tegas.

- Pilih **smart room** bila ingin hasil cepat dan mudah divisualisasikan.
- Pilih **energi** bila tertarik pada efisiensi, operasi gedung, dan forecasting.
- Pilih **predictive maintenance** bila tertarik pada anomaly detection dan sistem industri.
- Pilih **edge AI** bila ingin membawa model keluar dari laptop ke perangkat nyata.
- Pilih **synthetic generator** bila ingin membangun lab latihan, bahan ajar, atau platform uji.

Prinsip yang harus ditekankan adalah: **satu proyek kecil yang utuh lebih bernilai daripada banyak proyek setengah jadi**.

F.6 Komunitas open source yang layak diikuti

Dokumentasi resmi penting, tetapi belajar akan jauh lebih cepat jika pembaca juga mengikuti komunitasnya.

Komunitas yang relevan meliputi:

- Python,
- pandas dan scikit-learn,
- TensorFlow/Keras,
- Mosquitto MQTT,
- Node-RED,
- InfluxDB,
- Grafana,
- Raspberry Pi,
- komunitas otomasi rumah dan *smart environment* berbasis *open source*.

Mengikuti komunitas membantu pembaca melihat contoh nyata, mempelajari masalah lapangan, menemukan *best practice*, dan memahami alat yang terus berkembang.

F.7 Penutup Appendix F

Appendix ini adalah **jembatan dari buku ke tahap belajar berikutnya**. Isinya harus membantu pembaca memilih referensi yang tepat, menentukan proyek yang relevan, dan memperdalam stack *open source* secara bertahap. Tujuan akhirnya bukan membuat pembaca tahu lebih banyak alat, tetapi membuat mereka **lebih mampu membangun sistem yang benar-benar berjalan, berguna, dan bisa dikembangkan lebih lanjut**.

Kalau Anda mau, saya bisa lanjut ubah ini menjadi versi **lebih singkat lagi, siap-tempel ke naskah buku**, dengan gaya yang seragam seperti appendix lain.

Referensi

Das, H., Dey, N., & Balas, V. E. (Eds.). (2020). *Real-time data analytics for large scale sensor data* (Vol. 6). Academic Press.

Geng, H. (Ed.). (2017). *Internet of things and data analytics handbook*. John Wiley & Sons.

Mohanty, S. N., Chatterjee, J. M., Mangla, M., Satpathy, S., & Potluri, S. (Eds.). (2021). *Machine learning approach for cloud data analytics in IoT*. John Wiley & Sons.

Obaidat, M. S., Nayak, P., & Ray, N. K. (Eds.). (2024). *Intelligent computing on IoT 2.0, AI, and block chain technology*. CRC Press.

Pandey, A. C., Verma, A., Rathor, V. P. S., Singh, M., & Singh, A. K. (Eds.). (2023). *Intelligent analytics for Industry 4.0 applications*. CRC Press.

Perros, H. G. (2021). *An introduction to IoT analytics*. CRC Press.

Saleem, T. J., & Chishti, M. A. (Eds.). (2021). *Big data analytics for Internet of Things*. John Wiley & Sons.

Sharma, R., Jeon, G., & Zhang, Y. (Eds.). (2023). *Data analytics for Internet of Things infrastructure*. Springer.

Tentang Penulis



Onno W. Purbo, saat ini bertugas sebagai rektor di Institut Teknologi Tangerang Selatan (ITTS). Onno memperoleh gelar Ph.D bidang Electrical Engineering dari University of Waterloo, Canada, adalah seorang copyleft, educator dan ICT evangelist. Dia sudah mempublikasikan 50++ buku, termasuk free ICT ebook untuk sekolah tahun 2008. Beberapa buku terakhirnya adalah "Internet-TCP/IP: Konsep Dan Implementasi", 2018; "Sistem Operasi, Konsep Dan Membuat Linux OpenWRT Dan ROM Android", 2019; "IPv6 Untuk Mendukung Operasi Jaringan Dan Domain Name System", 2019; "Kubernetes untuk Pemula", 2024 dan "Membuat Operator Seluler 5G Sendiri", 2024.

Dia memimpin sambungan pertama Internet di Institut Teknologi Bandung, tahun 1993-2000, dan menggunakannya untuk membuat jaringan Internet pendidikan yang pertama di Indonesia. Dia membebaskan frekuensi WiFi, memperkenalkan RT/RW-net, antenna Wajanbolic dan jaringan selular OpenBTS dan private 5G sendiri. Dia memimpin jaringan telepon pertama di atas Internet, VoIP Merdeka, yang kemudian hari dikenal sebagai VoIP Rakyat berbasis SIP dan menggunakan kode area +62520 dan +62521. Dia saat ini aktif memperkenalkan e-Learning, dan menjalankan server e-Learning di http://lms.onnocenter.or.id/moodle/ 66,000++ siswa / mahasiswa dan https://opencourse.itts.ac.id 39.000+ mahasiswa secara gratis. Email: onno@indo.net.id