UJIAN AKHIR SEMESTER

KECERDASAN BISNIS

[DOC_TYPE : TECHNICAL REPORT]

IMPLEMENTASI DATA LAKE

" Implementasi Data Lake Rekomendasi Waktu Terbaik Buat Ngecas HP"

Optimasi Keputusan Sehari-hari Menggunakan Prescriptive Analytics

RINGKASAN TEKNIS:

implementasi Sistem Data Lake berbasis MinIO ini diorkestrasi oleh Apache Airflow menggunakan skema Medallion untuk mengintegrasikan data SQL, API, dan CSV. Melalui proses ELT, sistem menghasilkan analisis preskriptif berupa rekomendasi pengisian daya yang dipersonalisasi dan disajikan secara responsif melalui dasbor Streamlit.

TIM ENGINEERING

Mario Franca Wijaya
NIM: 2210817310009

Dhea Aprilinda Utami
NIM: 2210817220019

PROGRAM STUDI TEKNOLOGI INFORMASI
SEMESTER GANJIL 2025

HALAMAN PERSEMBAHAN

Puji syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa, karena atas rahmat dan karunia-Nya, penulis dapat menyelesaikan laporan tugas Ujian Akhir Semester (UAS) mata kuliah kecerdasan bisnis yang berjudul "Implementasi Data Lake Rekomendasi Waktu Terbaik Buat Ngecas HP ".

Laporan ini disusun sebagai bentuk implementasi pemahaman penulis terhadap konsep Business Intelligence, mulai dari pengelolaan data mentah hingga transformasi data menjadi informasi yang memiliki nilai guna bagi pengambilan keputusan bisnis. Dalam proyek ini, penulis merancang solusi berbasis Data Lake untuk mengatasi tantangan kompleksitas data telemetri baterai guna memperpanjang usia pakai perangkat melalui analisis deskriptif hingga preskriptif.

Penulis menyadari bahwa keberhasilan penyusunan laporan ini tidak lepas dari dukungan berbagai pihak. Oleh karena itu, penulis ingin menyampaikan terima kasih kepada:

1. Bapak Irham Maulani Abdul Gani, S.Kom., M.Kom., selaku dosen mata kuliah Business Intelligence yang telah memberikan bimbingan, ilmu, dan arahan selama semester ini.
2. Rekan-rekan mahasiswa yang telah bertukar pikiran dan memberikan motivasi dalam proses pengerjaan proyek ini.
3. Seluruh pihak yang secara langsung maupun tidak langsung telah membantu kelancaran penyusunan laporan ini.

Penulis menyadari bahwa laporan ini masih jauh dari sempurna. Oleh karena itu, penulis mengharapkan kritik dan saran yang membangun dari pembaca demi perbaikan di masa mendatang. Akhir kata, semoga laporan ini dapat memberikan manfaat serta wawasan tambahan bagi pembaca, khususnya dalam bidang Business Intelligence dan arsitektur data modern.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1. Latar belakang

Di era transformasi digital saat ini, ketergantungan masyarakat terhadap perangkat seluler telah mencapai titik puncaknya, namun inovasi pada teknologi penyimpanan daya atau baterai belum mampu mengimbangi laju konsumsi energi dari aplikasi modern yang semakin kompleks. Permasalahan utama yang dihadapi oleh hampir seluruh pengguna *smartphone* adalah degradasi kapasitas baterai *Lithium-ion* yang berlangsung secara progresif. Berdasarkan fakta teknis, kesehatan baterai (*Battery Health*) sangat dipengaruhi oleh variabel eksternal seperti suhu operasional, tegangan listrik, dan pola siklus pengisian daya. Pengisian daya hingga kapasitas penuh (100%) secara terus-menerus atau membiarkan daya merosot hingga di bawah 20% menciptakan tekanan kimiawi yang mempercepat pembentukan kristal pada elektroda baterai, yang secara permanen mengurangi efisiensi penyimpanan energinya[1].

Kesehatan baterai (battery health) sangat dipengaruhi oleh berbagai faktor eksternal, seperti suhu operasional perangkat, tegangan listrik saat pengisian daya, serta pola siklus pengisian yang diterapkan oleh pengguna. Kebiasaan mengisi daya hingga 100% secara terus-menerus atau membiarkan baterai turun hingga di bawah 20% dapat menimbulkan tekanan kimiawi pada sel baterai[2]. Tekanan ini mempercepat pembentukan kristal pada elektroda yang berdampak pada penurunan kapasitas penyimpanan energi secara permanen, sehingga umur pakai baterai menjadi lebih singkat.

Permasalahan tersebut diperparah oleh kompleksitas data telemetri yang dihasilkan oleh berbagai sensor pada perangkat seluler, mulai dari arus listrik (mA), tegangan (V), hingga suhu dalam derajat Celsius[3]. Data dengan volume besar dan kecepatan tinggi ini tidak dapat dikelola secara optimal hanya dengan basis data tradisional. Oleh karena itu, penerapan arsitektur Data Lake menjadi solusi yang relevan. Pendekatan ini mengombinasikan fleksibilitas dan skalabilitas Data Lake sehingga memungkinkan pemrosesan data mentah secara real-time untuk analisis lanjutan[4][5].

Pengembangan sistem ini tidak hanya berfokus pada pelaporan data historis, tetapi juga diarahkan pada penerapan Prescriptive Analytics. Melalui pendekatan ini, sistem mampu memberikan rekomendasi tindakan yang spesifik dan personal, seperti menentukan waktu terbaik untuk mengisi daya agar baterai lebih awet berdasarkan pola penggunaan masing-masing individu. Setiap pengguna memiliki karakteristik beban kerja perangkat yang berbeda, sehingga solusi yang bersifat umum tidak lagi efektif untuk menjaga kesehatan baterai dalam jangka Panjang[6][7]. proyek ini dirancang untuk menunjukkan bagaimana teknologi kecerdasan bisnis dapat diterapkan pada permasalahan mikro yang memiliki dampak ekonomi signifikan bagi pengguna, yaitu memperpanjang usia pakai perangkat keras. Dengan mengintegrasikan arsitektur Data Lake yang modern, data sensor yang terfragmentasi dapat diolah menjadi informasi bernilai dan selanjutnya dikonversi menjadi kebijakan tindakan yang preskriptif.

1.2. Permasalahan yang dihadapi

Beberapa permasalahan utama yang menjadi latar belakang perancangan sistem ini adalah:

1. Pengguna smartphone sering mengalami penurunan kesehatan baterai (Battery Health) secara prematur akibat pola pengisian daya yang kurang tepat. Faktor utama meliputi siklus pengisian yang tidak teratur (sering membiarkan daya di bawah 20% atau di atas 80%) serta paparan panas berlebih saat proses pengisian berlangsung.
2. Informasi yang dibutuhkan untuk memberikan saran pengisian daya yang akurat tersebar di berbagai tempat, mulai dari log operasional basis data (PostgreSQL), berkas spesifikasi perangkat (CSV), hingga kondisi lingkungan (API Cuaca). Tanpa arsitektur Data Lake, sulit untuk menggabungkan data yang berbeda format ini secara cepat untuk menghasilkan analisis yang komprehensif.
3. Solusi manajemen daya yang tersedia saat ini umumnya bersifat statis dan tidak mempertimbangkan karakteristik unik setiap perangkat maupun pola aktivitas individu.

1.3. Tujuan penelitian

Tujuan dari penelitian dan implementasi sistem ini adalah:

1. Mengembangkan proses Extract, Load, Transform (ELT) yang terotomatisasi menggunakan Apache Airflow untuk mengintegrasikan data dari berbagai sumber heterogen ke dalam MinIO dengan skema Medallion Architecture (Bronze, Silver, dan Gold).
2. Menciptakan logika Recommendation Engine di dalam lapisan Gold Layer yang mampu melakukan kalkulasi Smart Estimation (perpaduan laju pengisian teoretis dan historis) guna memberikan saran tindakan nyata bagi pengguna berdasarkan profil penggunaan dan suhu lingkungan.
3. Membangun dasbor visualisasi interaktif menggunakan Streamlit yang terhubung langsung ke Data Lake untuk menyajikan tren kesehatan baterai, analisis penggunaan harian, serta estimasi waktu pengisian daya yang akurat sebagai sistem pendukung keputusan bagi pengguna..

BAB II
PERANCANGAN ARSITEKTUR

2.1. Urgensi Rancangan

Alasan mendasar mengapa arsitektur Data Lake diperlukan untuk solusi optimasi pengisian daya baterai didasari oleh permasalahan mengenai degradasi sel baterai lithium-ion yang dipercepat oleh faktor stres termal dan pola pengisian daya yang tidak teratur. Pengguna seringkali melakukan pengisian daya tanpa mempertimbangkan kondisi suhu lingkungan atau pola penggunaan harian, sehingga perangkat berisiko mengalami panas berlebih yang merusak kesehatan baterai jangka panjang. Tanpa adanya sistem yang mampu mengintegrasikan data internal perangkat dengan variabel eksternal seperti kondisi cuaca real-time, manajemen daya hanya bersifat reaktif dan kurang efektif dalam memperpanjang usia pakai (longevity) perangkat.

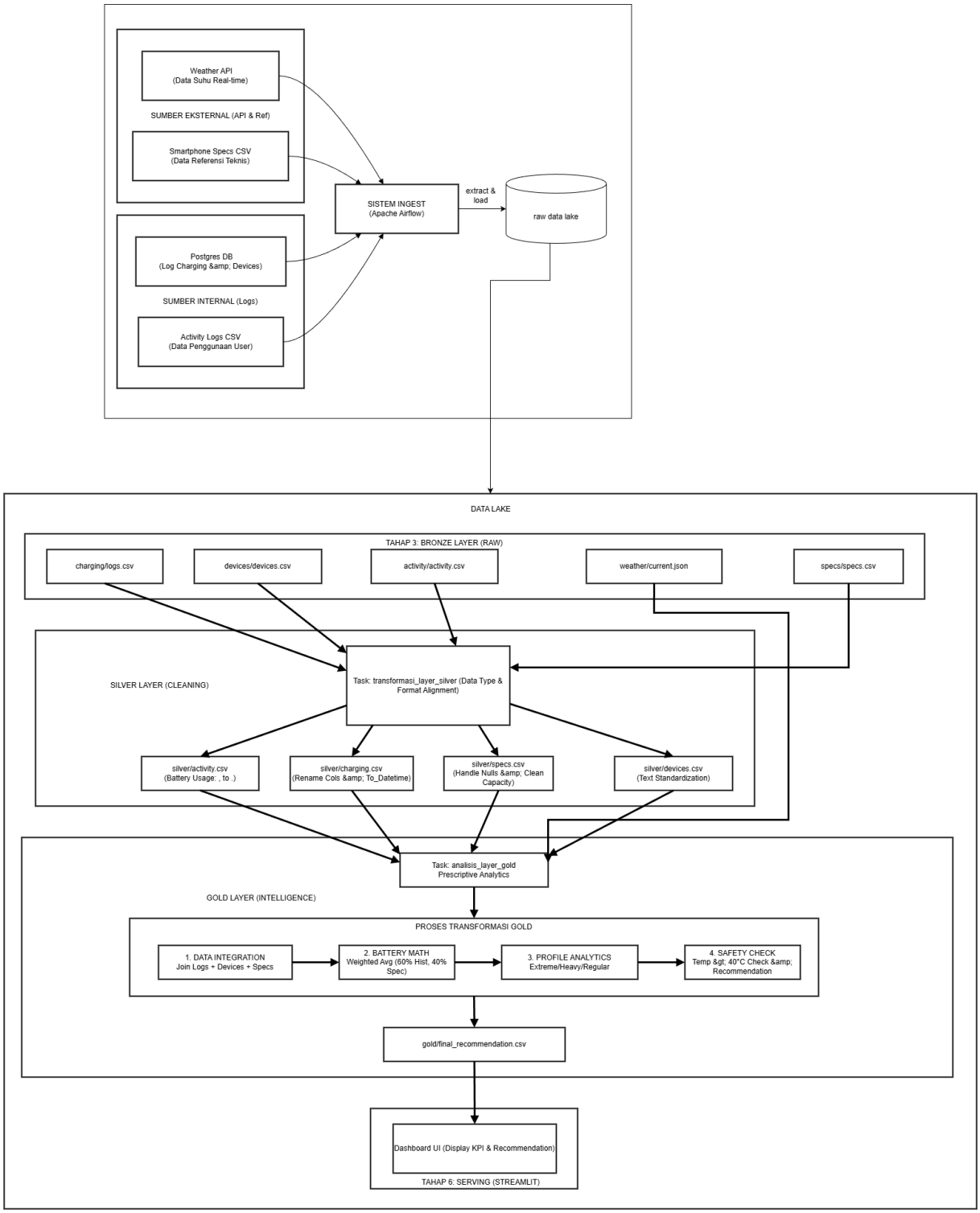
Alasan utama penggunaan arsitektur Data Lake dalam sistem ini adalah kemampuannya untuk mengintegrasikan data heterogen dari berbagai sumber seperti log operasional SQL, berkas CSV spesifikasi smartphone, dan API cuaca eksternal ke dalam satu ekosistem analitik yang terpusat. Arsitektur ini dipilih karena fleksibilitasnya dalam menyimpan data mentah sebagai objek immutable (tidak berubah) di Bronze Layer (MinIO). Hal ini menjamin data asli tetap tersedia untuk audit atau pemrosesan ulang jika di masa depan terdapat perubahan algoritma rekomendasi. Selain itu, Data Lake memfasilitasi penggunaan komputasi Python (Pandas) yang sangat fleksibel untuk mengeksekusi logika prescriptive analytics yang kompleks seperti penggabungan laju pengisian historis dan teoretis sebelum data disajikan.

Pemilihan arsitektur Data Lake dibandingkan Data Lakehouse didasari oleh efisiensi sumber daya dan karakteristik data proyek yang bersifat *append-only*. Berbeda dengan *Data Lakehouse* yang memerlukan manajemen metadata dan lapisan transaksi ACID (seperti Delta Lake atau Iceberg) yang kompleks, Data Lake memberikan solusi yang lebih ringan (*lightweight*) dan hemat biaya untuk mengelola log pengisian daya. Mengingat data log baterai tidak memerlukan perubahan (update/delete) yang intens, penggunaan format terbuka seperti CSV dan JSON langsung di atas MinIO memungkinkan integrasi yang lebih cepat dan simpel dengan pustaka Pandas dan Streamlit tanpa beban komputasi tambahan untuk pemeliharaan *transaction logs*.

Penerapan skema Medallion Architecture (Bronze, Silver, Gold) di dalam Data Lake bertujuan untuk menjaga integritas data di setiap tahap pemrosesan secara terstruktur di dalam Object Storage. Dengan memisahkan data mentah di staging area (Bronze & Silver) dari data yang sudah matang di Gold Layer, sistem dapat membentuk Analytical Wide Table yang sangat optimal. Data final dalam bentuk CSV terdenormalisasi ini memastikan bahwa dashboard Streamlit dapat memanggil informasi secara instan melalui protokol S3 tanpa perlu melakukan proses join atau kalkulasi berat lagi di sisi aplikasi.

Tabel 1. Analisis Urgensi

Aspek Komparasi	Kondisi Konvensional (OS Default)	Kondisi Menggunakan Data Lake
Integrasi Data	Log internal tidak mempertimbangkan faktor eksternal seperti suhu cuaca.	Terpusat (Single Source of Truth). Data SQL, CSV, dan API diintegrasikan dalam arsitektur Medallion.
Dasar Analisis	Estimasi waktu pengisian hanya berdasarkan sisa persentase baterai sesaat.	Menggunakan *Battery Math* yang menggabungkan data historis dan spesifikasi teknis.
Keamanan Perangkat	Risiko *overheating* tinggi karena tidak ada sensor suhu lingkungan yang terhubung ke logika pengisian.	Sistem melakukan *Safety Check* otomatis terhadap ambang batas suhu (Health Threshold).
Sifat Analisis	Hanya menampilkan statistik penggunaan masa lalu.	Memberikan rekomendasi waktu terbaik dan durasi pengisian yang optimal.



Gambar 1. Skema arsitektur

2.2. Dataset yang Dibutuhkan dan Dikumpulkan

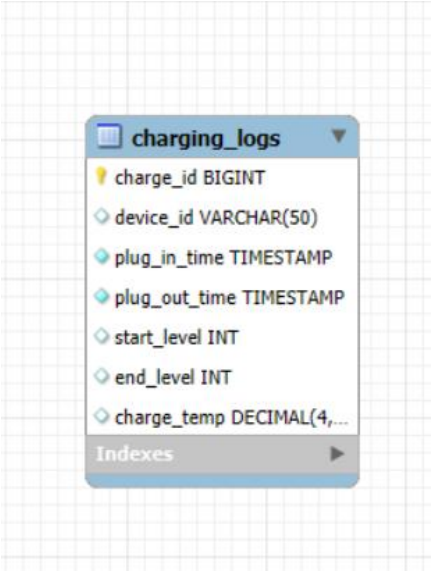
Tahap perancangan ini melibatkan identifikasi sumber data mentah (*raw data*) yang beragam untuk memberikan pandangan komprehensif terhadap kesehatan baterai. Data yang dikumpulkan dibagi menjadi empat kategori utama:

2.2.1. Histori Pengisian Daya Perangkat (*Charging History*)

Dataset ini berfungsi untuk memantau perilaku teknis saat perangkat sedang diisi dayanya. Data ini mencatat efisiensi pengisian (dari level awal ke akhir), durasi pengisian melalui penanda waktu (timestamp), serta faktor keamanan berupa suhu perangkat saat dialiri listrik. Ini adalah kunci untuk mendeteksi apakah kebiasaan mengisi daya (seperti overnight charging) berdampak pada suhu baterai.

Tabel 2. Histori Pengisian Daya Perangkat

Nama Kolom	**Tipe Data**	**Keterangan**
charge_id	SERIAL	Bertindak sebagai **Primary Key**. Tipe SERIAL berarti angka akan bertambah secara otomatis (auto-increment) setiap ada data baru yang masuk.
Device_id	VARCHAR(50)	Menyimpan ID unik perangkat (maksimal 50 karakter). Kolom ini digunakan untuk mengidentifikasi HP mana yang melakukan pengisian daya.
Plug_in_time	TIMESTAMP	Mencatat tanggal dan waktu yang sangat detail saat kabel *charger* mulai dicolokkan.
Plug_out_time	TIMESTAMP	Mencatat tanggal dan waktu yang detail saat kabel *charger* dicabut dari perangkat.
Start_level	INT	Menyimpan angka bulat (integer) yang menunjukkan persentase baterai saat pengisian dimulai.
End_level	INT	Menyimpan angka bulat yang menunjukkan persentase baterai saat pengisian selesai atau dicabut.
Charge_temp	DECIMAL(4,1)	Menyimpan data suhu dengan presisi 13 ecimal. Angka (4,1) berarti total ada 4 digit, dengan 1 digit di belakang koma (contoh: 30,5).



Gambar 2. Histori Pengisian Daya Perangkat

2.2.2. Sheet Log Pengguna (*User Log*)

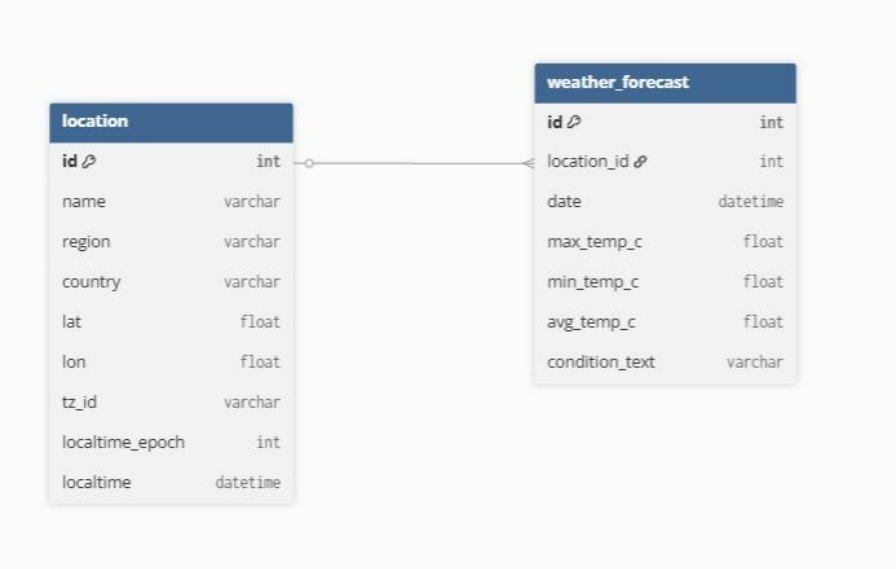
Dataset ini berfungsi untuk melihat perilaku konsumsi daya oleh pengguna. Melalui log ini, kita dapat melihat aplikasi apa saja yang paling banyak menguras baterai dan berapa lama layar aktif (screen time). Data ini membantu memetakan pola penggunaan harian dan mengidentifikasi kategori aplikasi (misalnya Social Media vs Messaging) yang paling boros energi.

Tabel 3. Sheet Log Pengguna

Nama Atribut	Keterangan	Contoh Data
log_id	ID unik untuk setiap baris data (primary key). Digunakan untuk mengidentifikasi setiap entri log secara spesifik.	1, 2, 3...
device_id	Kode identifikasi untuk perangkat HP yang digunakan. Memungkinkan kita melacak aktivitas di perangkat yang berbeda.	D001, D003
activity_date	Tanggal ketika aktivitas penggunaan aplikasi tersebut tercatat.	16/12/2025
application_name	Nama aplikasi yang dijalankan oleh pengguna pada perangkat tersebut.	TikTok, WhatsApp, Safari
screen_time_minutes	Durasi waktu (dalam satuan **menit**) selama layar HP aktif menampilkan aplikasi tersebut.	138, 60, 20
battery_usage_percent	Persentase daya baterai yang dikonsumsi oleh aplikasi tersebut selama waktu penggunaan yang tercatat.	24,4; 18,0; 5,7
activity_category	Pengelompokan jenis aplikasi berdasarkan fungsinya.	Social Media, Messaging, Browser

2.2.3. Weather API

dataset dari Weather API berfungsi untuk memberikan konteks kondisi lingkungan eksternal. Data cuaca (seperti suhu lokasi) digunakan untuk menganalisis apakah suhu lingkungan sekitar berpengaruh terhadap peningkatan suhu baterai baik saat digunakan maupun saat diisi daya.



Gambar 3. Diagram *Weather API*

2.2.4. Sheet Scraping Data Spesifikasi Perangkat

Dataset ini menyediakan data dasar (baseline) dari perangkat tersebut. Berisi spesifikasi teknis pabrikan seperti kapasitas baterai (mAh), kecepatan pengisian maksimal (Watt), dan jenis prosesor. Data ini sangat penting sebagai pembanding: misalnya, apakah penurunan baterai yang terjadi masih wajar sesuai kapasitas aslinya (mAh) atau sudah menunjukkan tanda-tanda degradasi.

Tabel 4. Sheet Scraping Data Spesifikasi Perangkat

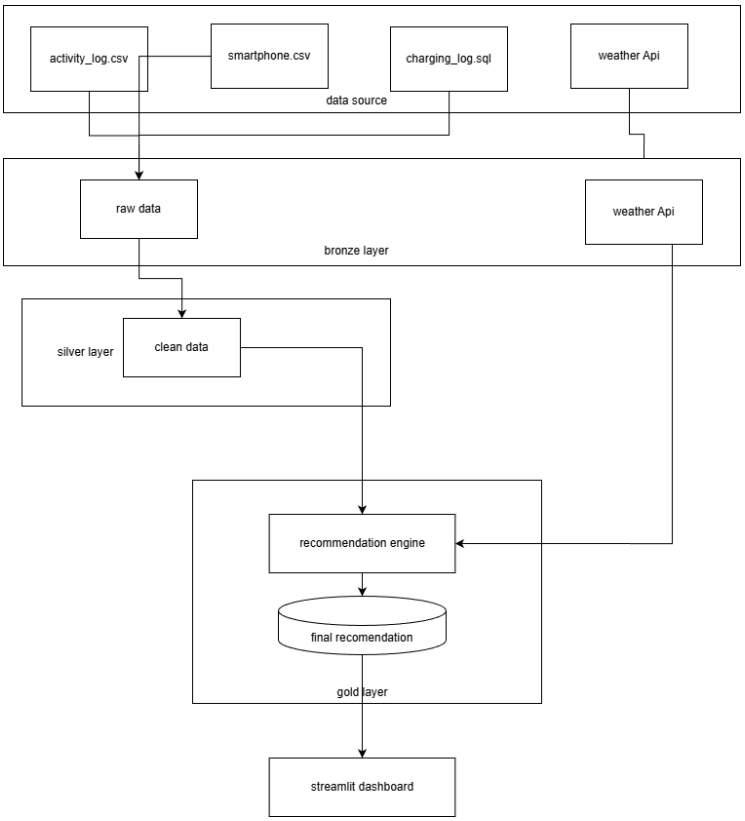
Nama Atribut	Keterangan	Contoh Data
brand_name	Nama merek atau produsen smartphone.	apple, xiaomi, samsung
model	Nama model spesifik dari smartphone tersebut.	Apple iPhone 11, Xiaomi Redmi Note 12
price	Harga perangkat (biasanya dalam mata uang asal data, misal: Rupee atau unit lainnya).	38999, 109900

Nama Atribut	Keterangan	Contoh Data
avg_rating	Skor penilaian rata-rata (skala 1-10) untuk perangkat tersebut.	7.3, 8.1
5G_or_not	Indikator dukungan jaringan 5G (1 = Ya, 0 = Tidak).	1, 0
processor_brand	Merek prosesor (chipset) yang digunakan.	bionic, snapdragon, dimensity
num_cores	Jumlah inti prosesor (CPU cores).	6, 8
processor_speed	Kecepatan prosesor dalam satuan GHz.	2.65, 3.1
battery_capacity	Kapasitas baterai dalam satuan mAh.	3110, 5000
fast_charging_available	Indikator apakah mendukung pengisian daya cepat (1 = Ya, 0 = Tidak).	1, 0
fast_charging	Kecepatan pengisian daya dalam satuan Watt (jika tersedia).	18, 67
ram_capacity	Kapasitas RAM dalam satuan GB.	4, 6, 8
internal_memory	Kapasitas penyimpanan internal dalam satuan GB.	64, 128
screen_size	Ukuran diagonal layar dalam satuan inci.	6.1, 6.67
refresh_rate	Kecepatan penyegaran layar dalam satuan Hz.	60, 90, 120
num_rear_cameras	Jumlah modul kamera yang ada di bagian belakang.	2, 3, 4
os	Sistem operasi yang digunakan.	ios, android
primary_camera_rear	Resolusi kamera belakang utama dalam satuan Megapixel (MP).	12, 50, 108

Nama Atribut	Keterangan	Contoh Data
primary_camera_front	Resolusi kamera depan (selfie) dalam satuan Megapixel (MP).	12, 16, 32
extended_memory_available	Indikator apakah tersedia slot memori tambahan/MicroSD (1 = Ya, 0 = Tidak).	0, 1
resolution_height	Jumlah piksel vertikal pada layar.	1792, 2400
resolution_width	Jumlah piksel horizontal pada layar.	828, 1080

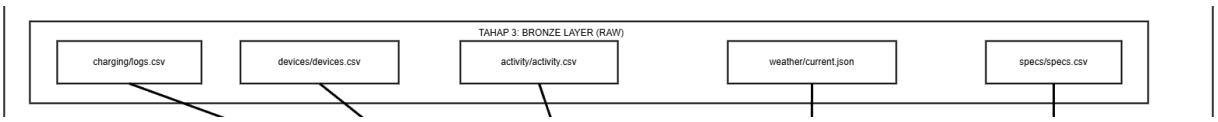
2.3. Desain Data Storage

Sistem ini mengadopsi Arsitektur Medallion untuk mengelola siklus hidup data dari mentah hingga menjadi informasi siap pakai. Arsitektur ini memungkinkan pelacakan data yang lebih baik (data lineage) dan memastikan kualitas data yang tinggi di setiap tahap.



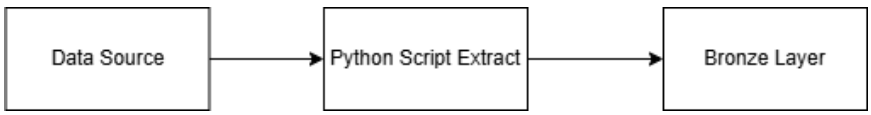
Gambar 4. Diagram design data storage mengunakan arsitektur medalion

2.3.1. Bronze Layer



Gambar 5. Diagram bronze layer

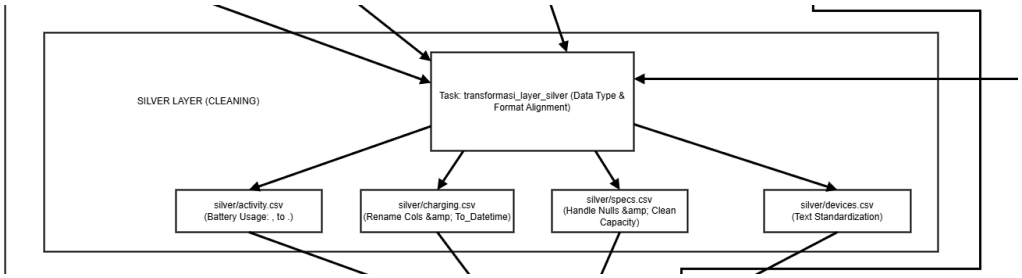
Lapisan ini berfungsi sebagai zona pendaratan pertama bagi semua data yang berasal dari sumber eksternal. Data disimpan dalam format aslinya tanpa modifikasi apa pun untuk menjaga integritas data historis.



Gambar 6. Alur data ke bronze layer

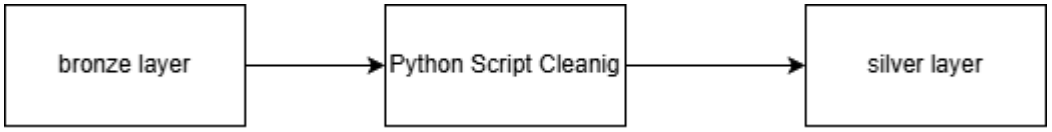
1. Penyimpanan menggunakan MiniO (Object Storage yang kompatibel dengan S3).
2. Isi data mencakup
    - `bronze/activity_log/activity.csv` menyimpan data mentah penggunaan aplikasi.
    - `bronze/charging_log/log.csv` menyimpan data mentah histori pengisisan handphone.
    - `Bronze/device/devices.csv` menyimpan data Log spesifikasi handphone pengguna.
    - `Bronze/smartphone_specs/specs.csv` menyimpan data spesifikasi handphone global yang didapatkan dari data scraping.
    - `Bronze/weather/current_weather.json` menyimpan data respons mentah dari WeatherAPI.
3. Karakteristik data meliputi data tidak terstruktur atau semi-terstruktur, mencakup duplikasi dan format yang tidak konsisten.

2.3.2. Silver Layer



Gambar 7. Diagram silver layer

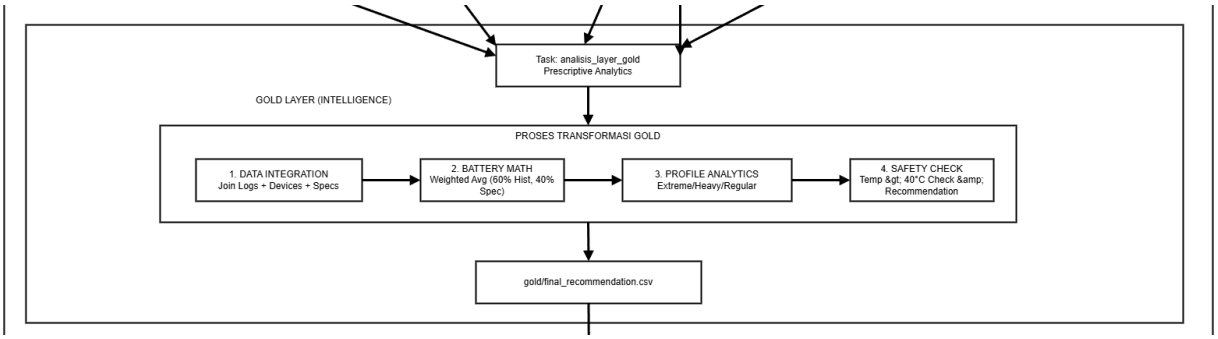
Data dari Bronze Layer dibersihkan, divalidasi, dan ditransformasi ke dalam format tabel relasional. Proses ini memastikan data siap untuk digabungkan namun belum sepenuhnya dioptimalkan untuk analitik akhir.



Gambar 8. Alur data ke silver layer

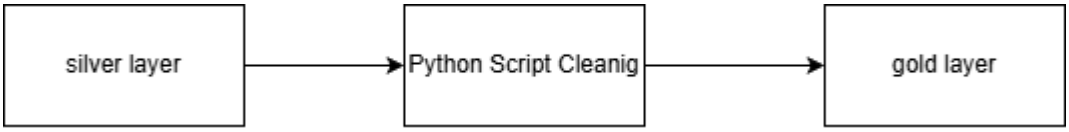
1. Penyimpanan menggunkan MiniO pada silver layer.
2. Proses ELT
    - Mengonversi format tanggal (contohnya DD/MM/YYYY ke ISO Standard).
    - Pembersihan angka (mengubah koma , menjadi titik . untuk tipe data *decimal*).
    - Normalisasi teks (contohnya penyeragaman nama brand perangkat).

2.3.3. Gold Layer



Gambar 9. Diagram gold layer

Lapisan ini merupakan tingkatan tertinggi dalam arsitektur Data Lake proyek ini, di mana data telah diolah menjadi informasi matang yang siap dikonsumsi. Data pada lapisan ini tidak lagi bersifat teknis, melainkan sudah berorientasi pada nilai bisnis dan pengambilan keputusan.



Gambar 10. Alur data ke gold layer

1. Penyimpanan menggunkan MiniO pada gold layer.
2. Isi Data meliputi final_recommendations (hasil akhir dari *Recommendation Engine*).
3. Karakteristik: Data sangat terstruktur, bersih, dan berorientasi pada kebutuhan bisnis (rekomendasi pengisian daya).

2.4. Arsitektur Data Lake

Sistem ini mengadopsi arsitektur Data Lake. Dengan menggunakan MiniO sebagai penyimpanan objek dan PostgreSQL sebagai mesin analitik, arsitektur ini memungkinkan data mengalir secara otomatis dari format mentah hingga menjadi informasi siap pakai yang diorkestrasi sepenuhnya oleh Apache Airflow.

2.4.1. Layer Ingestion

Lapisan ini merupakan pintu masuk utama bagi seluruh data dari berbagai format. Kami menggunakan Apache Airflow sebagai mesin orkestrasi untuk menarik data secara otomatis melalui beberapa protokol. Data aktivitas aplikasi ditarik dari format .csv (seperti yang terlihat pada file log_aktivitas_baterai_hp.csv), sementara log teknis pengisian daya diekstraksi dari basis data PostgreSQL menggunakan *query* SQL terencana. Selain itu, faktor eksternal berupa suhu lingkungan diambil secara *real-time* dari WeatherAPI. Seluruh data ini kemudian disimpan dalam kondisi aslinya (*raw data*) di dalam *Bronze Bucket* pada MinIO.

2.4.2. Layer Distillation

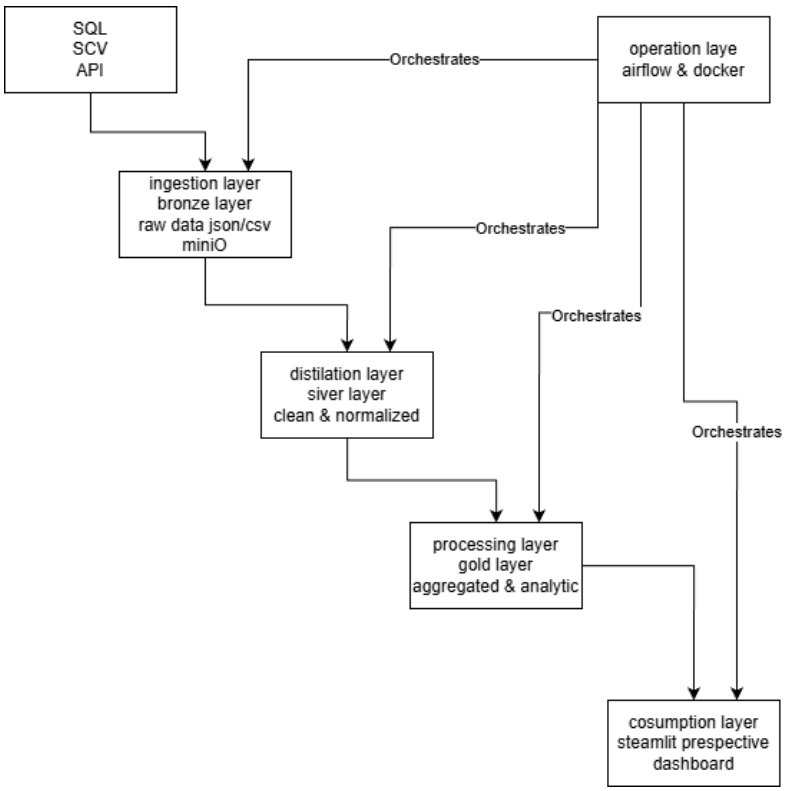
Pada *Layer Distillation* atau zona Silver, data mentah dari Bronze Layer mengalami purifikasi dan standarisasi. Melalui tugas transformasi_layer_silver, dilakukan proses *cleaning* seperti penyelarasan tipe data, penanganan nilai kosong (*null values*), serta normalisasi format penulisan (misalnya mengubah pemisah koma menjadi titik pada data persentase baterai). Hasilnya adalah data yang bersih dan terstruktur dalam format berkas yang lebih efisien, siap untuk masuk ke tahap analisis mendalam.

2.4.3. Layer Processing

Layer Processing atau zona Gold adalah pusat kecerdasan sistem di mana logika bisnis diterapkan. Di sini, sistem menjalankan tugas analisis_layer_gold yang mencakup perhitungan Battery Math menggabungkan bobot laju pengisian historis dan spesifikasi teknis perangkat. Selain itu, dilakukan Safety Check otomatis yang membandingkan suhu lingkungan dengan ambang batas kesehatan baterai. Hasil akhirnya adalah berkas final_recommendation.csv yang bersifat preskriptif, berisi panduan konkret bagi pengguna

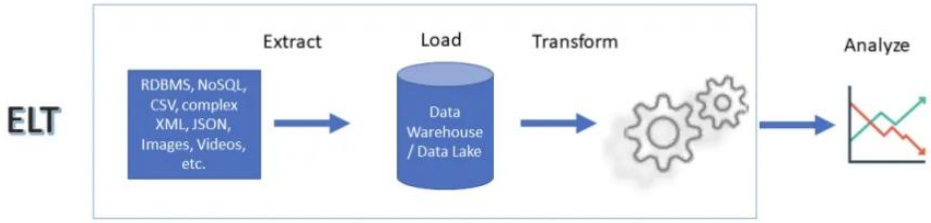
2.4.4. Layer Consumption

Lapisan terakhir ini adalah wajah dari seluruh proses arsitektur data. Melalui antarmuka Streamlit yang ke dalam app.py, *Layer* Consumption menyajikan informasi secara visual dan preskriptif kepada pengguna. Insight yang dihasilkan bersifat kontekstual; misalnya, jika suhu di Banjarmasin terdeteksi tinggi melalui *Ingestion Layer*, maka pada Consumption *Layer* akan muncul peringatan otomatis untuk menghindari pengisian daya cepat. Di sinilah nilai dari proyek ini terwujud bukan lagi sekadar tumpukan angka, melainkan panduan nyata bagi pengguna untuk memperpanjang usia pakai baterai mereka berdasarkan pola penggunaan yang tercatat.



Gambar 11. Diagram Data Lake

2.5. Proses Loading Data ELT (Extract, Load, Transform)



Gambar 12. Skema ELT

Dalam mengelola siklus hidup data pada proyek ini, kami menerapkan paradigma ELT. Berbeda dengan ETL konvensional yang melakukan transformasi sebelum data disimpan, pendekatan ELT memungkinkan untuk memuat data mentah ke dalam Data Lake terlebih dahulu. Hal ini memberikan fleksibilitas tinggi jika di masa mendatang diperlukan logika bisnis baru, untuk dapat memproses ulang data asli yang tersimpan di Bronze Layer tanpa harus melakukan ekstraksi ulang dari sumber primer.

2.5.1. Ekstraksi dan Pemuatan Mentah (Raw Loading)

Tahap pertama adalah ekstraksi paralel yang dikendalikan oleh Apache Airflow. Dengan mendefinisikan beberapa operator Python dalam data_lake_dag.py untuk menarik data dari tiga titik ujung (*endpoints*) yang berbeda:

- Data Relasional dengan menggunakan pustaka psycopg2, sistem melakukan *query* ke basis data PostgreSQL untuk mengambil log pengisian daya.

- Data Eksternal (API) dengan mengambil kondisi cuaca terkini dari WeatherAPI untuk mendapatkan variabel suhu luar ruangan di Banjarmasin.
- Data Flat Files dengan mengunggah file spesifikasi ponsel dan log aktivitas aplikasi dari format CSV.

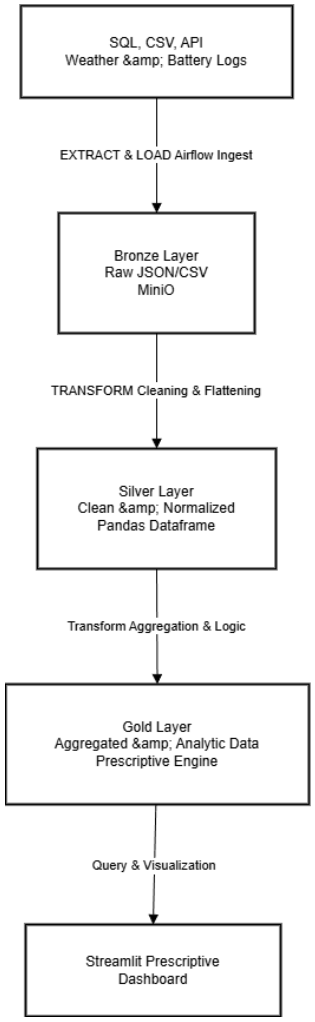
Data yang berhasil diekstraksi langsung dimuat ke dalam *Bronze Bucket* di MinIO menggunakan *library* boto3. Pada tahap ini, tidak ada modifikasi yang dilakukan pada isi data (as-is), sehingga integritas data dari sumber tetap terjaga sepenuhnya.

2.5.2. Transformasi dalam Data Lake

Setelah seluruh data tersedia di storage, proses transformasi dilakukan secara internal di dalam lingkungan Data Lake. Proses ini dibagi menjadi dua fase utama yaitu :

1. Fase Silver (Pembersihan)
   Pada fase silver layer skrip Python melakukan pembersihan pada dataset aktivitas. yaitu dengan melakukan pemisahan desimal pada kolom penggunaan baterai agar konsisten secara matematis. Selain itu juga, melakukan operasi join antara data log pengisian dengan tabel devices untuk menyematkan identitas perangkat yang lengkap. Hasil dari fase ini disimpan kembali ke dalam Silver Bucket.
2. Fase Gold (Agregasi & Logika Preskriptif)
   Pada fase merupakan hasil dari menghitung rata-rata kecepatan pengisian daya untuk menghasilkan estimasi waktu "20% ke 80%". Di sini juga diterapkan logika kondisional: jika suhu lingkungan (dari API) dan suhu pengisian (dari SQL) berada di atas ambang batas tertentu, sistem akan menghasilkan teks rekomendasi "Hindari Fast Charging" dalam file final_recommendation.csv.

2.5.3. Alur ELT



Gambar 13. Diagram alur ELT

Penjelasan singkat diagram:

1. *Data Sources* -> Bronze
   Data mentah yang berasal dari log aktivitas (.csv), spesifikasi smartphone, data log pengisian dari PostgreSQL, serta data cuaca Banjarmasin dari API diekstraksi menggunakan Apache Airflow. Data ini dimuat langsung ke MinIO Bronze Layer dalam format aslinya (JSON/CSV) tanpa transformasi untuk menjamin integritas data historis dan ketersediaan data "as-is".
2. *Bronze -> Silver* (*Transform*)
   Skrip Python (Pandas) melakukan pembersihan data (data cleansing). Proses utama meliputi konversi pemisah desimal pada kolom penggunaan baterai (koma ke titik), standarisasi format tanggal, serta penanganan nilai kosong. Data kemudian diintegrasikan melalui operasi join antara log penggunaan dan metadata perangkat untuk memperkaya konteks informasi.
3. *Silver -> Gold* (*Transform*)

Data yang telah bersih dikonsolidasi menjadi dataset siap olah. Kemudian melakukan agregasi untuk menghitung metrik penggunaan harian dan tren konsumsi daya. Di lapisan ini, data cuaca Banjarmasin disinkronkan dengan waktu pengisian daya untuk membentuk dataset multidimensi yang menghubungkan variabel lingkungan dengan performa teknis perangkat.

4. *Gold* → Streamlit
   Data warehouse pada PostgreSQL berfungsi sebagai serving layer yang menyediakan kueri cepat dan stabil bagi dashboard Streamlit untuk menyajikan visualisasi tren penggunaan baterai dan saran aksi nyata bagi pengguna.

2.6. Perancangan Proses ELT (Extract -Load-Transform) pada Layer Analitik

Sistem ini menerapkan pendekatan ELT (Extract-Load-Transform) pada layer analitik untuk memastikan data yang masuk ke dalam database sudah dalam kondisi bersih dan siap pakai. Pada tahap akhir siklus ELT, fokus bergeser dari integritas data menuju penciptaan nilai melalui analisis preskriptif. Layer analitik (Gold Layer) dirancang untuk mentransformasi data terstruktur di tahap Silver menjadi informasi strategis. tidak hanya menyajikan statistik deskriptif, tetapi juga menerapkan logika kondisional yang mempertimbangkan variabel teknis dan lingkungan secara simultan.

2.6.1. Extract (E) & Initial Load (L) to Bronze

Untuk menjaga performa, sistem tidak membebani database operasional atau API pihak ketiga dengan transformasi berat di awal. Sebaliknya, Apache Airflow bertindak sebagai *orchestration pipeline* yang menjalankan dua langkah utama:

1. Extract
   Melalui PythonOperator, Airflow menarik data mentah berupa berkas CSV (log aktivitas aplikasi dan spesifikasi smartphone), data log pengisian dari PostgreSQL, serta data cuaca real-time dari WeatherAPI dalam format JSON.
2. Load (Bronze)
   Data tersebut langsung disimpan sebagai objek permanen (*immutable*) di dalam MinIO Bronze Bucket. Tahap ini menjamin bahwa data asli tetap tersedia jika sewaktu-waktu diperlukan pemrosesan ulang (audit data) tanpa harus membebani sumber data asli (PostgreSQL/API).

2.6.2. Transform (T) via Procedural & Logic Modeling

Transformasi data menjadi inti dari *pipeline* ini, di mana data mentah diolah menggunakan Python (Pandas) di dalam *worker* Airflow sebelum akhirnya siap disajikan. Proses ini mencakup dua mekanisme transformasi utama:

1. Integrasi Data (Bronze to Silver) data mentah dari *Data Lake* dibaca oleh Pandas untuk dibersihkan. Dengam melakukan normalisasi format, seperti menyeragamkan pemisah desimal pada kolom penggunaan baterai dan melakukan *mapping* antara device_id pada log dengan spesifikasi teknis di berkas `smartphones.csv`. Hasilnya adalah dataset terintegrasi yang disimpan pada Silver Layer.
2. Analisis Preskriptif (Silver to Gold) dengan menerapkan logika bisnis untuk menghasilkan rekomendasi aksi nyata.
   - Kalkulasi Durasi sistem menghitung rata-rata kecepatan pengisian daya untuk menentukan estimasi durasi pengisian 20% ke 80%.

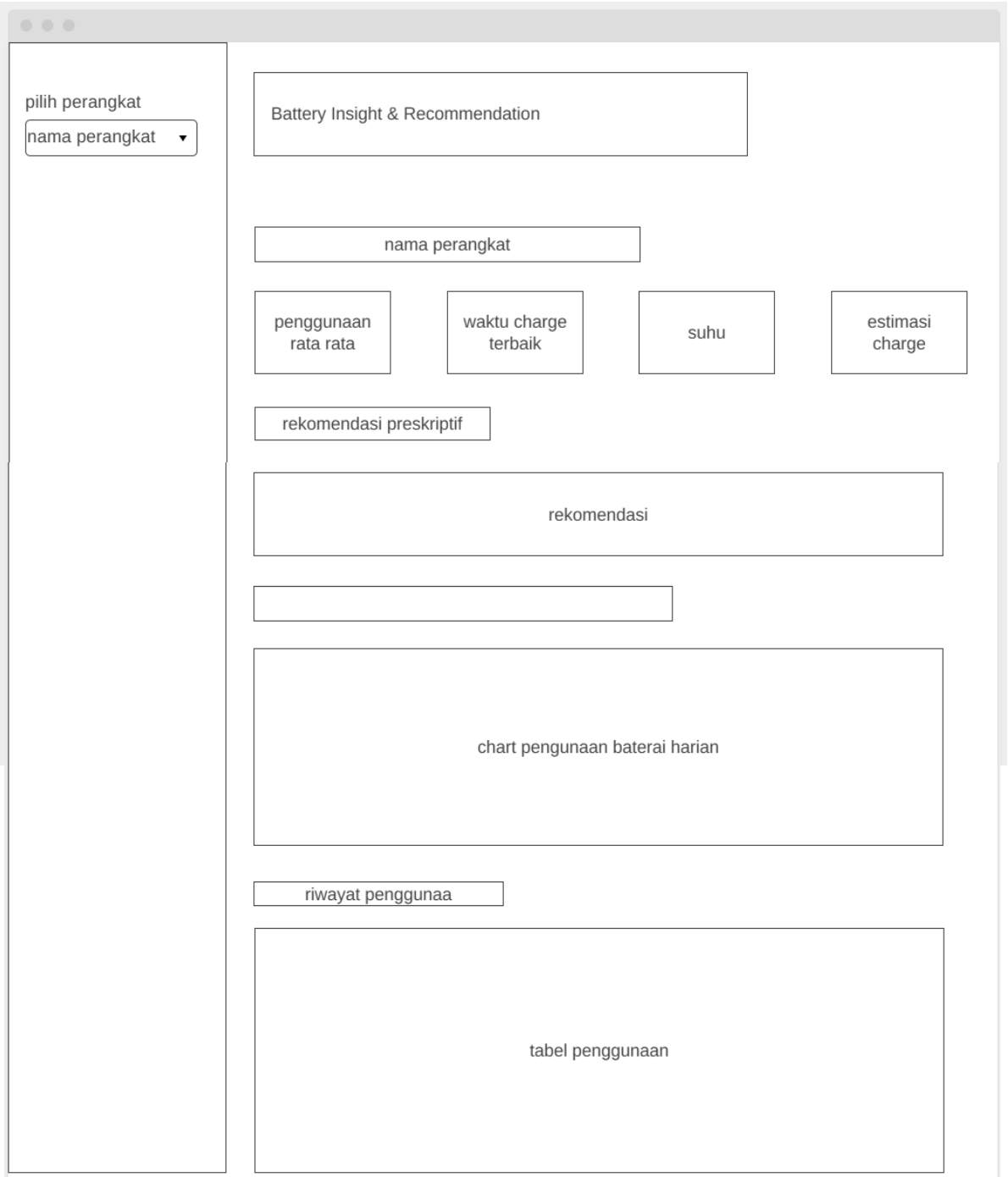
- Logika Kondisional dengan memadukan data suhu pengisian (charge_temp) dengan suhu luar ruangan di Banjarmasin. Jika akumulasi panas melampaui ambang batas, sistem akan menyematkan pesan peringatan kesehatan baterai.
- Hasil akhirnya dimuat ke dalam berkas `final_recommendation.csv` di Gold Layer, yang bertindak sebagai satu-satunya sumber data (*serving layer*) bagi dasbor Streamlit untuk menjamin kecepatan kueri.

2.7. Perancangan Front-End (Visualisasi atau OLAP)

Antarmuka pengguna dirancang menggunakan framework Streamlit dengan fokus utama pada Actionability. Dashboard ini tidak hanya berfungsi sebagai visualisasi data pasif, melainkan sebagai Decision Support System (DSS) bagi pengguna dalam memantau kesehatan baterai dan mengambil langkah preventif terhadap degradasi sel daya. Front-end aplikasi dibangun menggunakan pendekatan berbasis web interaktif yang terhubung langsung ke MinIO Gold Layer. Dengan memanfaatkan hasil pengolahan data terstruktur, dashboard memungkinkan penyajian metrik yang kompleks menjadi informasi yang intuitif.

2.7.1. Desain Layout Dashboard

Desain antarmuka pengguna (*User Interface*) pada sistem *Battery Insight & Recommendation* dikembangkan menggunakan kerangka kerja (*framework*) Streamlit dengan pendekatan *responsive web design*. Layout ini dirancang secara modular untuk memastikan informasi diagnostik dan preskriptif dapat disampaikan secara hierarkis dan efisien kepada pengguna. Struktur desain dibagi menjadi empat zona fungsional utama sebagai berikut:



Gambar 14. Design wireframe dashboard

1. Panel Kontrol Navigasi (Sidebar)
   Zona ini berfungsi sebagai pengendali input utama bagi pengguna. Implementasi desain menempatkan komponen *selectbox* di sisi kiri layar untuk memfasilitasi pemilihan perangkat berdasarkan identitas unik (*device_id*) dan tipe model. Hal ini bertujuan untuk memisahkan logika pemilihan data dengan area visualisasi utama, sesuai dengan prinsip *separation of concerns* pada desain antarmuka.
2. Header dan Mekanisme Interaksi Data
   Bagian atas panel utama memuat judul aplikasi dan deskripsi singkat operasional. Fitur krusial pada zona ini adalah tombol interaktif "Refresh & Update Data" yang terintegrasi dengan *Apache Airflow* melalui *REST API*. Tombol ini berfungsi untuk memicu proses ELT (*Extract, Load, Transform*) pada *Gold Layer* di *Minio Object Storage*, memberikan fleksibilitas bagi pengguna untuk memperbarui data secara *real-time* sebelum memulai analisis.
3. Panel Indikator Kinerja Utama (Metric Card Display) Penyajian data kuantitatif menggunakan tata letak empat kolom (*four-column grid*) untuk menampilkan metrik utama secara ringkas (*at-a-glance*). Komponen ini mencakup:
   - Rata-rata Penggunaan Representasi efisiensi baterai harian.
   - Waktu Charge Terbaik Hasil optimasi algoritma untuk jadwal pengisian daya.
   - Kondisi Lingkungan Integrasi suhu *outdoor* berdasarkan lokasi spesifik (API Banjarmasin).
   - Estimasi Durasi Proyeksi waktu pengisian pada rentang kritis 20-80%.
4. Visualisasi Analitik dan Rekomendasi Preskriptif Bagian akhir dashboard difokuskan pada interpretasi data mendalam dan saran tindakan:
   - Modul Preskriptif Menggunakan elemen *info box* untuk menyajikan teks rekomendasi otomatis yang dihasilkan oleh model analitik terkait tindakan preventif perawatan baterai.
   - Analisis Tren Visualisasi menggunakan *bar chart* untuk memetakan fluktuasi penggunaan energi harian dalam rentang waktu mingguan.
   - Detail Transaksional Tabel data historis disediakan di bagian paling bawah untuk transparansi data, memungkinkan pengguna melakukan verifikasi rincian penggunaan pada tanggal tertentu.

2.7.2. Fitur Analisis Preskriptif (Rekomendasi)

1. Mekanisme Estimasi Durasi Charging (*Charging Time Estimator*)
   Sistem menghilangkan ketidakpastian pengguna mengenai berapa lama mereka harus meninggalkan perangkat saat diisi daya. Berbeda dengan estimasi statis pabrikan, sistem kami menghitung durasi berdasarkan performa nyata perangkat yang tercatat di tabel charging_logs.
   - Target pengisian ditetapkan pada level optimal 80% untuk menjaga kesehatan jangka panjang sel Lithium-ion.
   - Kalkulasi

$$Estimasi\ Waktu = \frac{(80\% - Level\ saat\ ini)}{Rata - Rata\ Laju\ pengisian\ (per\ menit)}$$

- Jika perangkat terdeteksi mendukung *fast charging* berdasarkan data smartphones.csv, sistem akan menyesuaikan koefisien efisiensi untuk memberikan estimasi yang lebih presisi.

2. Deteksi Pola Kebiasaan (*Behavioral Pattern Recognition*)

   Sistem melakukan audit terhadap log aktivitas aplikasi (`log_aktivitas_baterai_hp.csv`) untuk mengidentifikasi kapan "Jam Kritis" pengguna terjadi. Hal ini memungkinkan sistem untuk memberikan saran pengisian daya sebelum baterai benar-benar habis di tengah aktivitas penting.

   - Mengidentifikasi jam-jam dengan *screen-on-time* tertinggi (misalnya, penggunaan intensif TikTok atau Games).
   - Menentukan waktu di mana level baterai diprediksi akan menyentuh angka 20% berdasarkan laju degradasi rata-rata harian.
   - Dashboard akan memunculkan pesan "Saran Pengisian Daya" 1 hingga 2 jam sebelum pengguna memasuki *Critical Zone*, memastikan kontinuitas produktivitas.

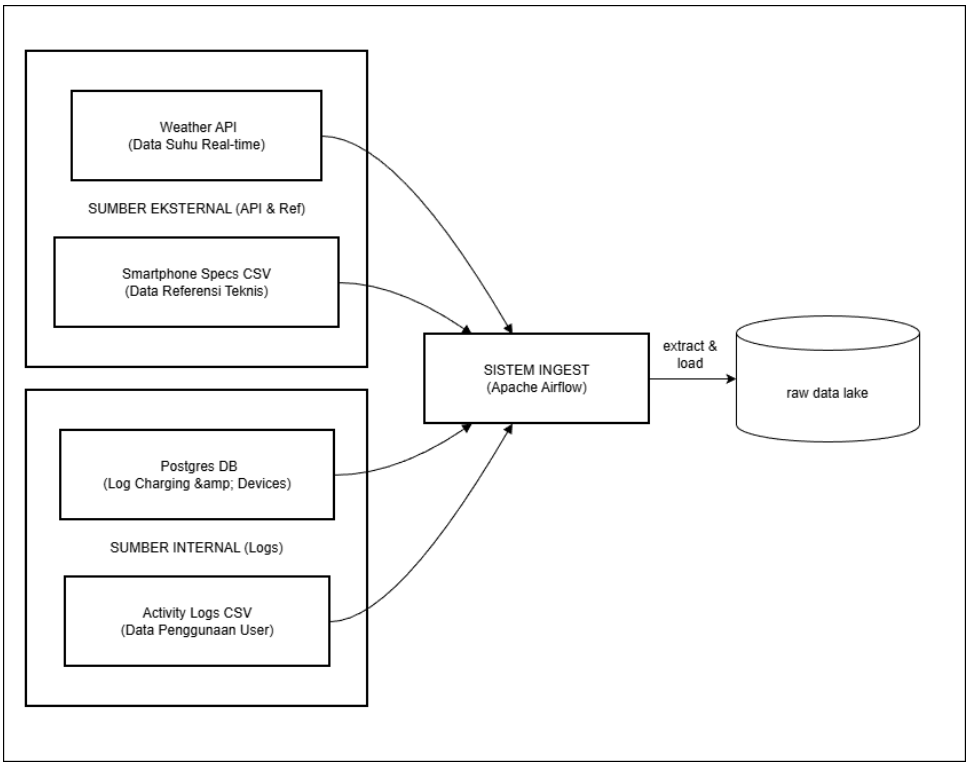
3. Pengayaan Data Kontekstual (*External Weather Context*)

   inovasi utama dalam proyek ini adalah pengintegrasian suhu eksternal Banjarmasin dari WeatherAPI ke dalam variabel keputusan. Panas berlebih adalah musuh utama baterai, sehingga suhu lingkungan menjadi faktor penentu apakah *Fast Charging* aman digunakan.

   - Fungsi elt_weather_api pada Airflow menyuplai data suhu terkini yang kemudian diproses dengan ambang batas termal sebagai berikut:
     - Suhu < 28°C (Status: Aman) Lingkungan optimal, pengisian daya cepat sangat disarankan untuk efisiensi waktu.
     - Suhu 28°C - 33°C (Status: Hati-hati) Suhu moderat, sistem menyarankan pemantauan suhu perangkat selama proses pengisian.
     - Suhu > 33°C (Status: Tunda/Bahaya) Risiko *overheating* tinggi. Sistem secara preskriptif menyarankan pengguna untuk menunda pengisian daya atau menghindari penggunaan aplikasi berat saat ponsel tersambung ke pengisi daya.

BAB III
IMPLEMENTASI PERANCANGAN

3.1. Implementasi Akusisi Data



Gambar 15. Alur Sumber Data

Implementasi system menggunakan akusisi data (data ingestion) dari empat sumber yang digunakan. Dengan meggunakan struktur data lake. Proses ini melibatkan data Excel, SQL, dan API. Akusisi data menggunakan skrip dari python (battery_papeline.py) yang mengekstraksi dari 4 sumber berikut:

1. Sql dabatabase (data charging.db) ntuk menyimpan data histori pengechasan.
2. Excel Spreadsheet (log_aktivitas_baterai_hp.csv) untuk menyimpan data loq harian aktifitas penggunaan device.
3. Open WeatherAPI untuk mengakusisi data kondisi lingkungan.
4. Scraping data Excel Spreadsheet (smartphones.csv) digunakan untuk data spesifikasi handphone yang digunakan.

3.2. Implementasi Data Log Harian Aktifitas Penggunaan Device

Data Log Harian Aktifitas Penggunaan Device disimpan dala format .csv digunakan untuk input data secra manual, Data ini merepresentasikan konsumsi baterai berdasarkan aktivitas aplikasi harian.

1. Sumber Data: File data_source/log_aktivitas_baterai_hp.csv.
2. Mekanisme Pemrosesan: Sistem diorkestrasi oleh Apache Airflow menggunakan pustaka Pandas untuk membaca berkas CSV tersebut. Implementasi mencakup tahap pra-pemrosesan (*pre-processing*) sesuai alur *Data Lake*, yaitu:

- Standardisasi format numerik pada kolom battery_usage_percent (mengubah desimal koma menjadi titik).
- Konversi tipe data kolom tanggal (activity_date) menjadi objek datetime yang valid.
- Penyimpanan: Data yang telah dibersihkan disimpan ke tabel staging.stg_activity untuk diproses lebih lanjut pada lapisan analitik.

3.3. Implementasi Data Spesifikasi Perangkat (Scraped Device Data)

Data Spesifikasi Perangkat (Scraped Device Data) Implementasi ini berfokus pada integrasi data spesifikasi teknis smartphone yang diperoleh dari hasil web scraping eksternal. Data ini krusial untuk menentukan kapasitas baterai dan kemampuan fast charging.

1. Sumber Data: File data_source/smartphones.csv.
2. Alur Integrasi: Data spesifikasi dimuat ke dalam sistem melalui fungsi `elt_csv_specs`. Tanpa memerlukan transformasi kompleks pada tahap awal, data mentah dari file CSV (smartphones.csv) langsung disimpan ke Bronze Layer pada Data Lake (MinIO) dengan path bronze/smartphone_specs/specs.csv. Data ini nantinya dibersihkan di Silver Layer (silver/specs/clean_specs.csv) dan digunakan sebagai referensi spesifikasi baterai pada analisis di Gold Layer.

3.4. Implementasi Akuisisi Data Cuaca (Weather API Integration)

Data Cuaca (Weather API Integration) mendukung analisis preskriptif berbasis suhu lingkungan, sistem mengimplementasikan koneksi Application Programming Interface (API) ke layanan pihak ketiga, WeatherAPI.

1. Mekanisme Koneksi: Diimplementasikan dalam DAG Airflow menggunakan pustaka REQUEST untuk mengirim permintaan HTTP GET ke endpoint API yang tertera di API.txt.
2. Parameterisasi: Sistem mengambil data cuaca saat ini(current) untuk kota parameter (contoh: Banjarmasin).

```
elt_weather_api():
with open('/opt/airflow/API.txt', 'r') as f: url = f.read().strip()
respon = requests.get(url).json()
get_minio_client().put_object(Bucket=BUCKET_NAME, Key="bronze/weather/current_weather.json", Body=json.dumps(respon))
```

Gambar 16. Extrak data dari API WeatherAPI

3.5. Pembuatan Infrastruktur Penyimpanan Data (Data Storage)

Implementasi penyimpanan data pada sistem ini menggunakan MinIO Object Storage yang dikonfigurasi sebagai layanan container di dalam ekosistem Docker. MinIO dipilih sebagai solusi Data Lake karena kompatibilitasnya dengan protokol Amazon S3, kinerja tinggi, dan kemampuan skalabilitas untuk menangani data terstruktur maupun semi-terstruktur.

Mekanisme penyimpanan diorganisir menggunakan satu bucket utama bernama datalake yang dibagi secara logis menjadi tiga lapisan (Medallion bernama datalake yang dibagi secara logis menjadi tiga lapisan (Medallion Architecture) untuk mendukung alur kerja ELT (Extract, Load, Transform):

1. Bronze Layer (Penyimpanan Data Mentah)

   Lapisan ini berfungsi sebagai zona pendaratan (landing zone) untuk data asli yang diekstrak langsung dari sumber tanpa modifikasi.

   1. Format Data: CSV dan JSON.
   2. Path Penyimpanan:
      - bronze/weather/current_weather.json:
        Menyimpan respon JSON mentah dari API cuaca.
      - bronze/charging_logs/logs.csv: Salinan mentah tabel log pengisian daya dari database PostgreSQL.
      - bronze/devices/devices.csv: Salinan mentah data perangkat pengguna.
      - bronze/activity_logs/activity.csv & bronze/smartphone_specs/specs.csv: Data log aktivitas dan spesifikasi HP dari file CSV eksternal.
2. Silver Layer (Penyimpanan Data Bersih)

   Lapisan ini menyimpan data yang telah melalui proses validasi, pembersihan, dan standarisasi tipe data (seperti konversi timestamp dan normalisasi format angka).

   1. Format Data: Cleaned CSV.
   2. Path Penyimpanan:
      - `silver/activity/clean_activity.csv`
      - `silver/charging/clean_charging.csv`
      - `silver/specs/clean_specs.csv`
      - `silver/devices/clean_devices.csv`
3. Gold Layer (Penyimpanan Data Analitik)

   Lapisan akhir yang menyimpan hasil agregasi dan perhitungan logika bisnis yang siap digunakan untuk pelaporan atau konsumsi pengguna akhir.

   1. Format Data: Final CSV.
   2. Path Penyimpanan:
      - gold/recommendations/final_recommendation.csv: Menyimpan hasil akhir rekomendasi pengisian daya yang menggabungkan data historis, spesifikasi perangkat, dan kondisi cuaca real-time.

3.6. Implementasi Proses Pemuatan Data (ELT Pipeline Orchestration)

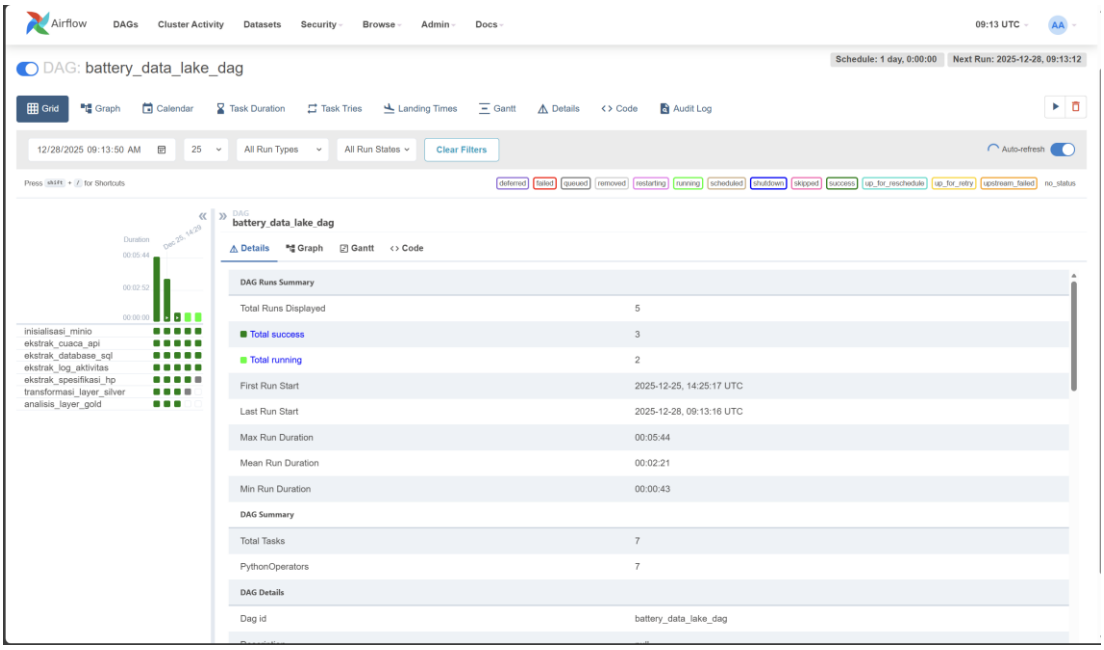
proses orkestrasi data dilakukan menggunakan Apache Airflow untuk mengelola aliran data (pipeline) dari berbagai sumber ke dalam Data Lake (MinIO). Pipeline ini dirancang dengan pendekatan ELT (Extract, Load, Transform) yang dibagi menjadi tiga lapisan data: Bronze (Raw), Silver (Cleaned), dan Gold (Analytic/Recommendation).

3.6.1. Arsitektur Pipeline

Pipeline diatur dalam sebuah DAG (Directed Acyclic Graph) bernama battery_data_lake_dag yang dijadwalkan berjalan setiap hari (daily). Alur kerja pipeline adalah sebagai berikut:

1. Inisialisasi: Memastikan bucket penyimpanan siap.
2. Ekstraksi (Parallel): Mengambil data dari API, Database SQL, dan File CSV secara bersamaan.
3. Transformasi (Silver): Membersihkan dan menstandarisasi data mentah.

4. Analisis (Gold): Menggabungkan data, melakukan perhitungan metrik, dan menghasilkan rekomendasi pengisian daya.



Gambar 17. Alur pipeline pada apache airflow

3.6.2. Komponen Task DAG

Berikut adalah rincian tugas (task) yang diimplementasikan dalam DAG Airflow:

1. inisialisasi_minio:
   Memastikan koneksi ke MinIO Server.:
   1. Memastikan koneksi ke MinIO Server.

```
MINIO_ENDPOINT = "http://minio:9000"
MINIO_ACCESS_KEY = "minioadmin"
MINIO_SECRET_KEY = "minioadmin"
BUCKET_NAME = "datalake"
```

Gambar 18. Source code Mengkoneksikan MinIO server

   2. Membuat bucket bernama datalake jika belum tersedia untuk menampung seluruh data.

```
def init_minio():
    try: get_minio_client().create_bucket(Bucket=BUCKET_NAME)
    except: pass
```

Gambar 19. Source code Membuat bucket di MinIO

2. Layer Bronze (Ekstraksi Data):
   Tugas-tugas ini berjalan secara paralel setelah inisialisasi:
   1. ekstrak_cuaca_api: Mengambil data cuaca real-time dari API eksternal (URL dibaca dari API.txt) dan menyimpannya sebagai file JSON (bronze/weather/current_weather.json).

```
def elt_weather_api():
```

```
    with open('/opt/airflow/API.txt', 'r') as f: url =
f.read().strip()
    respon = requests.get(url).json()
    get_minio_client().put_object(Bucket=BUCKET_NAME,
Key="bronze/weather/current_weather.json",
Body=json.dumps(respon))
```

Gambar 20. Source code Ekstraksi data Weather API

2. ekstrak_database_sql; Terhubung ke PostgreSQL (battery_analytics_db), melakukan query tabel charging_logs dan devices, lalu menyimpannya sebagai CSV.

```
def elt_sql_data():
    # Connect to the real database "battery_analytics_db"
    koneksi = psycopg2.connect(
        host="postgres",
        database="battery_analytics_db",
        user="airflow",
        password="airflow"
    )
    kursor = koneksi.cursor()

    # Extract Charging Logs
    kursor.execute("SELECT * FROM charging_logs")
    data_pengisian = kursor.fetchall()
    df_pengisian_mentah = pd.DataFrame(data_pengisian,
columns=['charge_id', 'device_id', 'plug_in_time',
'plug_out_time', 'start_level', 'end_level', 'charge_temp'])
    penyangga_logs = io.StringIO()
    df_pengisian_mentah.to_csv(penyangga_logs, index=False)
    get_minio_client().put_object(Bucket=BUCKET_NAME,
Key="bronze/charging_logs/logs.csv",
Body=penyangga_logs.getvalue())

    # Extract Devices
    kursor.execute("SELECT * FROM devices")
    data_perangkat = kursor.fetchall()
    df_perangkat_mentah = pd.DataFrame(data_perangkat,
columns=['device_id', 'device_type', 'brand_name', 'model'])
    penyangga_dev = io.StringIO()
    df_perangkat_mentah.to_csv(penyangga_dev, index=False)
    get_minio_client().put_object(Bucket=BUCKET_NAME,
Key="bronze/devices/devices.csv", Body=penyangga_dev.getvalue())

    koneksi.close()
```

Gambar 21. Source code Ektraksi data sql dari local

3. ekstrak_log_aktivitas: Membaca file lokal log_aktivitas_baterai_hp.csv dan memuatnya ke MinIO.

```
def elt_csv_activity():
    df =
pd.read_csv('/opt/airflow/data_source/log_aktivitas_baterai_hp.cs
v', sep=';')
    penyangga = io.StringIO()
    df.to_csv(penyangga, index=False)
    get_minio_client().put_object(Bucket=BUCKET_NAME,
Key="bronze/activity_logs/activity.csv",
Body=penyangga.getvalue())
```

Gambar 22. Source code Ekstraksi data log aktivitas.csv

4. ekstrak_spesifikasi_hp: Membaca data spesifikasi smartphone dari smartphones.csv ke MinIO.

```
def elt_csv_specs():
    df = pd.read_csv('/opt/airflow/data_source/smartphones.csv',
sep=';')
    penyangga = io.StringIO()
    df.to_csv(penyangga, index=False)
    get_minio_client().put_object(Bucket=BUCKET_NAME,
Key="bronze/smartphone_specs/specs.csv",
Body=penyangga.getvalue())
```

Gambar 23. Source code Ekstraksi data spesifikasi smartphone

3. Layer Silver (Pembersihan & Transformasi)

```
def etl_silver_layer():
    s3 = get_minio_client()

    # Activity
    df_aktivitas =
pd.read_csv(io.BytesIO(s3.get_object(Bucket=BUCKET_NAME,
Key="bronze/activity_logs/activity.csv")['Body'].read()))
    if df_aktivitas['battery_usage_percent'].dtype == 'object':
df_aktivitas['battery_usage_percent'] =
df_aktivitas['battery_usage_percent'].str.replace(',',
'.').astype(float)
    df_aktivitas['activity_date'] =
pd.to_datetime(df_aktivitas['activity_date'], dayfirst=True)

    # Charging Logs (From DB CSV dump)
    df_cas =
pd.read_csv(io.BytesIO(s3.get_object(Bucket=BUCKET_NAME,
Key="bronze/charging_logs/logs.csv")['Body'].read()))
    df_cas['plug_in'] = pd.to_datetime(df_cas['plug_in_time'])
    df_cas['plug_out'] = pd.to_datetime(df_cas['plug_out_time'])
```

```
    df_cas = df_cas.rename(columns={'start_level': 'start_lvl',
'end_level': 'end_lvl', 'charge_temp': 'temp'})

    # Devices (From DB CSV dump)
    df_perangkat =
pd.read_csv(io.BytesIO(s3.get_object(Bucket=BUCKET_NAME,
Key="bronze/devices/devices.csv")['Body'].read()))

    # Specs
    df_spek =
pd.read_csv(io.BytesIO(s3.get_object(Bucket=BUCKET_NAME,
Key="bronze/smartphone_specs/specs.csv")['Body'].read()))

    # Save Cleaned
    for nama, data in [('activity', df_aktivitas), ('charging',
df_cas), ('specs', df_spek), ('devices', df_perangkat)]:
        penyangga = io.StringIO(); data.to_csv(penyangga,
index=False)
        s3.put_object(Bucket=BUCKET_NAME,
Key=f"silver/{nama}/clean_{nama}.csv", Body=penyangga.getvalue())
```

Gambar 24. Source code pada silver layer

transformasi_layer_silver:

1. Membaca data mentah dari folder bronze/ di MinIO.
2. Melakukan pembersihan data: mengubah format desimal (koma ke titik), konversi tipe data tanggal (datetime), dan normalisasi nama kolom.
3. Menyimpan hasil bersih ke folder silver/ (contoh: silver/activity/clean_activity.csv).

4. Layer Gold (Analisis & Rekomendasi):

```
def etl_gold_analysis():
    s3 = get_minio_client()
    df_aktivitas =
pd.read_csv(io.BytesIO(s3.get_object(Bucket=BUCKET_NAME,
Key="silver/activity/clean_activity.csv")['Body'].read()))
    df_cas =
pd.read_csv(io.BytesIO(s3.get_object(Bucket=BUCKET_NAME,
Key="silver/charging/clean_charging.csv")['Body'].read()))
    df_spek =
pd.read_csv(io.BytesIO(s3.get_object(Bucket=BUCKET_NAME,
Key="silver/specs/clean_specs.csv")['Body'].read()))
    df_perangkat =
pd.read_csv(io.BytesIO(s3.get_object(Bucket=BUCKET_NAME,
Key="silver/devices/clean_devices.csv")['Body'].read()))

    try:
        data_cuaca = json.loads(s3.get_object(Bucket=BUCKET_NAME,
Key="bronze/weather/current_weather.json")['Body'].read().decode('ut
f-8'))
```

```
        suhu_saat_ini, lokasi = data_cuaca['current']['temp_c'],
data_cuaca['location']['name']
    except: suhu_saat_ini, lokasi = 30.0, "Unknown"

    # Map Device ID to Model Name using the devices table
    peta_perangkat = pd.Series(df_perangkat.model.values,
index=df_perangkat.device_id).to_dict()

    # Historical Rate
    df_cas['durasi'] = (pd.to_datetime(df_cas['plug_out']) -
pd.to_datetime(df_cas['plug_in'])).dt.total_seconds() / 60
    df_cas = df_cas[df_cas['durasi'] > 0]
    df_cas['laju'] = (df_cas['end_lvl'] - df_cas['start_lvl']) /
df_cas['durasi']
    rata_rata_laju =
df_cas.groupby('device_id')['laju'].mean().to_dict()

    # Historical Charging Hour Pattern (Mode)
    df_cas['jam_cas'] = pd.to_datetime(df_cas['plug_in']).dt.hour
    pola_jam_cas = df_cas.groupby('device_id')['jam_cas'].agg(lambda
x: x.mode()[0] if not x.mode().empty else None).to_dict()

    data_harian = df_aktivitas.groupby(['device_id',
'activity_date'])['battery_usage_percent'].sum().reset_index()
    hasil_akhir = []
    for id_perangkat in data_harian['device_id'].unique():
        harian_perangkat = data_harian[data_harian['device_id'] ==
id_perangkat]
        rata_rata_penggunaan =
harian_perangkat['battery_usage_percent'].mean()
        model = peta_perangkat.get(id_perangkat, "Unknown")

        # Spec Lookup
        cocok =
df_spek[df_spek['model'].str.contains(model.split('(')[0].strip(),
case=False, na=False)]
        kapasitas, watt = 4000, 15
        if not cocok.empty:
            kapasitas = cocok.iloc[0].get('battery_capacity', 4000)
            if pd.isna(kapasitas): kapasitas = 4000
            watt = cocok.iloc[0].get('fast_charging', 15)
            if pd.isna(watt) or watt == 0: watt = 20 if "iPhone" in
model else 15

        # Smart Estimation
        menit_teoretis = (0.6 * kapasitas * 3.7 / 1000) / (watt *
0.85) * 60
        laju_historis = rata_rata_laju.get(id_perangkat, 1.0)
```

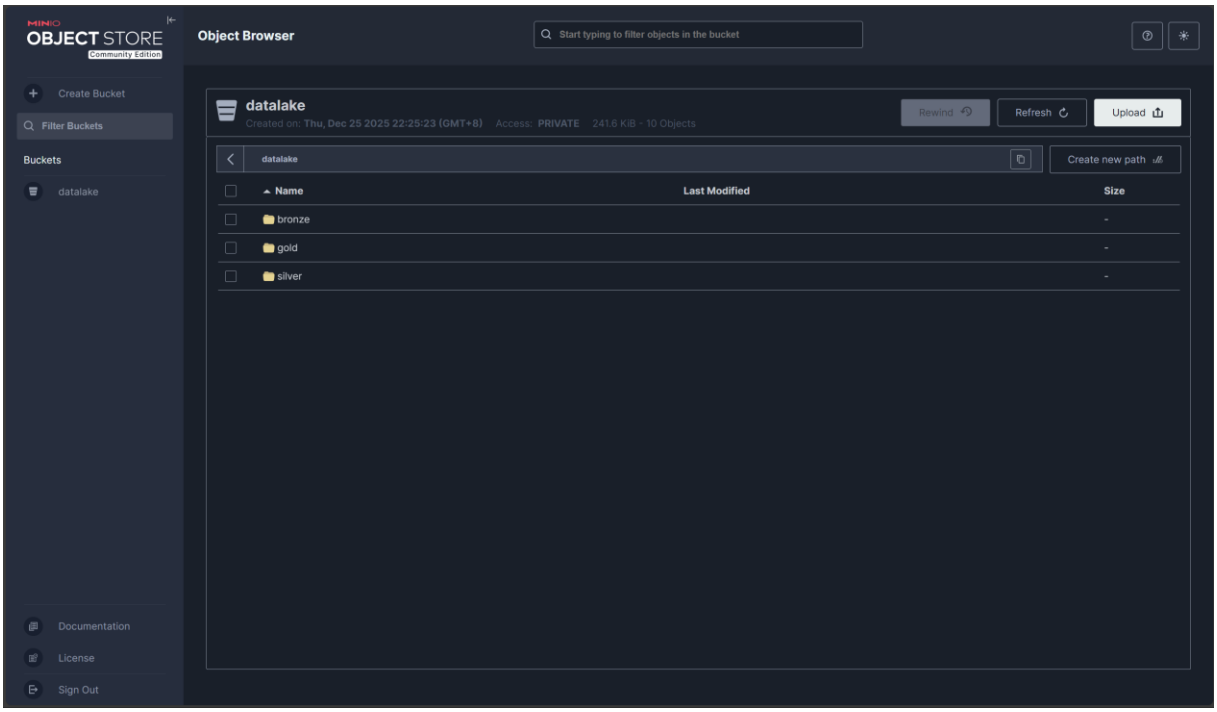
```
        if pd.isna(laju_historis) or laju_historis <= 0:
laju_historis = 1.0
        menit_historis = 60 / laju_historis
        estimasi_menit = int((menit_teoretis * 0.4) +
(menit_historis * 0.6))

        persen_cas_dibutuhkan = 60 # Target 20% to 80%
        siklus_dibutuhkan = rata_rata_penggunaan /
persen_cas_dibutuhkan

        batas_aman_suhu = 40.0
        apakah_aman = suhu_saat_ini < batas_aman_suhu

        # Get Historical Charging Hour Preference
        jam_kebiasaan = pola_jam_cas.get(id_perangkat)
        if jam_kebiasaan is None: jam_kebiasaan = 22 # Default if no
data
        jam_kebiasaan = int(jam_kebiasaan)

        # Build recommendation based on calculated metrics
        if siklus_dibutuhkan > 2.0:
            profil = "Extreme User"
            waktu_terbaik = f"{jam_kebiasaan:02d}:00,
{(jam_kebiasaan+8)%24:02d}:00 & {(jam_kebiasaan+16)%24:02d}:00"
            saran = f"Dengan rata-rata pengurasan baterai sebesar
{rata_rata_penggunaan:.1f}% per hari (>120%), perangkat Anda
dikategorikan sangat intensif. Disarankan melakukan 3 kali pengisian
daya sehari (pagi, siang, dan malam) guna menjaga kesehatan baterai
di rentang 20-80%."
        elif siklus_dibutuhkan > 1.0:
            profil = "Heavy User"
            waktu_terbaik = f"{jam_kebiasaan:02d}:00 &
{(jam_kebiasaan+12)%24:02d}:00"
            saran = f"Penggunaan harian Anda mencapai
{rata_rata_penggunaan:.1f}% (60%-120%). Disarankan pengisian daya
dua kali sehari (pagi dan sore) untuk menjaga kesehatan sel baterai
dalam jangka panjang."
        else:
            profil = "Regular User"
            waktu_terbaik = f"{jam_kebiasaan:02d}:00"
            saran = f"Konsumsi daya harian Anda stabil di angka
{rata_rata_penggunaan:.1f}% (<60%). Cukup lakukan satu kali
pengisian daya harian."

        saran_cuaca = f"Kondisi Banjarmasin saat ini
({suhu_saat_ini:.1f}°C) {'mendukung' if apakah_aman else 'berisiko
untuk'} pengisian daya cepat. Ambang batas panas (overheat) berada
pada suhu lingkungan di atas {batas_aman_suhu}°C."
```

```
        rekomendasi_lengkap = f"{saran} {saran_cuaca}"

        for baris in harian_perangkat.itertuples():
            hasil_akhir.append({
                "device_id": id_perangkat, "model": model, "date":
baris.activity_date,
                "daily_usage": baris.battery_usage_percent,
"avg_weekly_usage": rata_rata_penggunaan,
                "best_time_to_charge": waktu_terbaik, "temp_status":
f"Aman ({suhu_saat_ini+5:.1f}°C)",
                "recommendation": f"[{profil}]
{rekomendasi_lengkap}",
                "charge_estimation_20_80": f"{estimasi_menit}
Menit",
                "outdoor_temp": f"{suhu_saat_ini:.1f}°C",
"location": lokasi
            })

    df_gold = pd.DataFrame(hasil_akhir)
    penyangga_gold = io.StringIO(); df_gold.to_csv(penyangga_gold,
index=False)
    s3.put_object(Bucket=BUCKET_NAME,
Key="gold/recommendations/final_recommendation.csv",
Body=penyangga_gold.getvalue())
```

Gambar 25. Source code pada gold layer

analisis_layer_gold:

1. Menggabungkan data aktivitas, log pengisian daya, spesifikasi HP, dan cuaca.
2. Menghitung Siklus Penggunaan Harian untuk menentukan profil pengguna (Extreme/Heavy/Regular).
3. Menghitung Estimasi Waktu Cas (20-80%) berdasarkan spesifikasi watt dan riwayat laju pengisian.
4. Menentukan waktu terbaik mengisi daya berdasarkan pola kebiasaan (mode) jam cas pengguna.
5. Hasil akhir berupa tabel rekomendasi disimpan di gold/recommendations/final_recommendation.csv.



Gambar 26 Struktur Bucket Minio.

3.7. Implementasi Transformasi Analitik (Analytic Layer)

Transformasi analitik dilakukan pada Gold Layer bertujuan untuk menghasilkan rekomendasi pengisian daya yang personal untuk setiap perangkat. Proses ini dijalankan oleh fungsi elt_gold_analysis dalam DAG Airflow, yang mengintegrasikan data historis, spesifikasi teknis perangkat, dan kondisi lingkungan.

3.7.1. Integrasi Data

Proses ini menggabungkan lima sumber data utama dari Silver Layer dan Bronze Layer:

1. Aktivitas Baterai: Untuk menghitung rata-rata konsumsi daya harian (% per hari).
2. Log Pengisian Daya: Untuk mempelajari kecepatan pengisian aktual dan kebiasaan waktu (charging habit) pengguna.
3. Spesifikasi Perangkat: Untuk mendapatkan kapasitas baterai (mAh) dan kemampuan charging (Watt).
4. Data Perangkat: Untuk memetakan device_id ke nama model smartphone.
5. Data Cuaca: Untuk memberikan konteks keamanan suhu saat pengisian daya.

3.7.2. Logika Pembuatan Rekomendasi

Sistem menerapkan algoritma berbasis aturan (rule-based) dan statistik deskriptif untuk menghasilkan tiga komponen utama rekomendasi:

1. Profiling Pengguna (User Behavior Class)
   Sistem mengkategorikan pengguna berdasarkan Siklus Baterai Harian.
   1. Extreme User (> 2.0 siklus/hari): Pengguna sangat intensif, disarankan mengisi daya 3 kali sehari.

2. Heavy User (> 1.0 siklus/hari): Pengguna berat, disarankan mengisi daya 2 kali sehari.
3. Regular User (< 1.0 siklus/hari): Pengguna normal, cukup 1 kali sehari.

2. Estimasi Durasi Pengisian Cerdas (Smart Estimation)
   Sistem menghitung estimasi waktu yang dibutuhkan untuk mengisi daya dari 20% ke 80% menggunakan pendekatan Hybrid Calculation dengan bobot:
   1. 40% Teoritis: Berdasarkan rumus fisika (Kapasitas mAh / Watt Charger) dengan efisiensi 85%
   2. 60% Historis: Berdasarkan rata-rata kecepatan pengisian (% per menit) yang pernah tercatat dari perangkat tersebut.

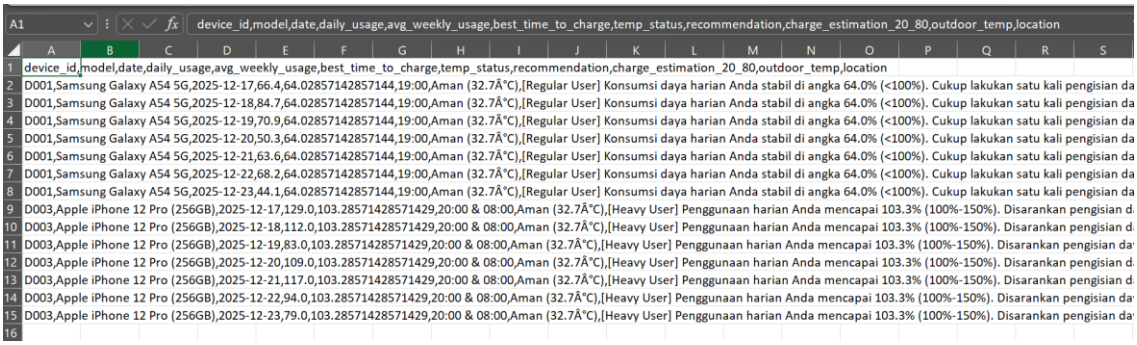
   Rumus implementasi dalam kode: estimasi_menit = int((menit_teoretis * 0.4) + (menit_historis * 0.6))

3. Penentuan Waktu Terbaik (Best Time Optimization)
   Waktu terbaik ditentukan dengan mencari Modus (Mode) dari jam mulai pengisian daya (plug_in_time) pengguna di masa lalu. Sistem menyesuaikan saran waktu agar sesuai dengan kebiasaan alami pengguna, namun dengan frekuensi yang disesuaikan berdasarkan profil pengguna (misal: Heavy User disarankan di jam biasa + interval 12 jam).

3.7.3. Keluaran Data (Output)

Hasil analisis disimpan kembali ke Data Lake (MinIO) pada path gold/recommendations/final_recommendation.csv. Data ini siap dikonsumsi oleh aplikasi dashboard (Streamlit).

Contoh struktur data hasil transformasi:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	device_id,model,date,daily_usage,avg_weekly_usage,best_time_to_charge,temp_status,recommendation,charge_estimation_20_80,outdoor_temp,location																		
2	D001,Samsung Galaxy A54 5G,2025-12-17,66.4,64.02857142857144,19:00,Aman (32.7Â°C),[Regular User] Konsumsi daya harian Anda stabil di angka 64.0% (<100%). Cukup lakukan satu kali pengisian da																		
3	D001,Samsung Galaxy A54 5G,2025-12-18,84.7,64.02857142857144,19:00,Aman (32.7Â°C),[Regular User] Konsumsi daya harian Anda stabil di angka 64.0% (<100%). Cukup lakukan satu kali pengisian da																		
4	D001,Samsung Galaxy A54 5G,2025-12-19,70.9,64.02857142857144,19:00,Aman (32.7Â°C),[Regular User] Konsumsi daya harian Anda stabil di angka 64.0% (<100%). Cukup lakukan satu kali pengisian da																		
5	D001,Samsung Galaxy A54 5G,2025-12-20,50.3,64.02857142857144,19:00,Aman (32.7Â°C),[Regular User] Konsumsi daya harian Anda stabil di angka 64.0% (<100%). Cukup lakukan satu kali pengisian da																		
6	D001,Samsung Galaxy A54 5G,2025-12-21,63.6,64.02857142857144,19:00,Aman (32.7Â°C),[Regular User] Konsumsi daya harian Anda stabil di angka 64.0% (<100%). Cukup lakukan satu kali pengisian da																		
7	D001,Samsung Galaxy A54 5G,2025-12-22,68.2,64.02857142857144,19:00,Aman (32.7Â°C),[Regular User] Konsumsi daya harian Anda stabil di angka 64.0% (<100%). Cukup lakukan satu kali pengisian da																		
8	D001,Samsung Galaxy A54 5G,2025-12-23,44.1,64.02857142857144,19:00,Aman (32.7Â°C),[Regular User] Konsumsi daya harian Anda stabil di angka 64.0% (<100%). Cukup lakukan satu kali pengisian da																		
9	D003,Apple iPhone 12 Pro (256GB),2025-12-17,129.0,103.28571428571429,20:00 & 08:00,Aman (32.7Â°C),[Heavy User] Penggunaan harian Anda mencapai 103.3% (100%-150%). Disarankan pengisian d																		
10	D003,Apple iPhone 12 Pro (256GB),2025-12-18,112.0,103.28571428571429,20:00 & 08:00,Aman (32.7Â°C),[Heavy User] Penggunaan harian Anda mencapai 103.3% (100%-150%). Disarankan pengisian d																		
11	D003,Apple iPhone 12 Pro (256GB),2025-12-19,83.0,103.28571428571429,20:00 & 08:00,Aman (32.7Â°C),[Heavy User] Penggunaan harian Anda mencapai 103.3% (100%-150%). Disarankan pengisian da																		
12	D003,Apple iPhone 12 Pro (256GB),2025-12-20,109.0,103.28571428571429,20:00 & 08:00,Aman (32.7Â°C),[Heavy User] Penggunaan harian Anda mencapai 103.3% (100%-150%). Disarankan pengisian d																		
13	D003,Apple iPhone 12 Pro (256GB),2025-12-21,117.0,103.28571428571429,20:00 & 08:00,Aman (32.7Â°C),[Heavy User] Penggunaan harian Anda mencapai 103.3% (100%-150%). Disarankan pengisian d																		
14	D003,Apple iPhone 12 Pro (256GB),2025-12-22,94.0,103.28571428571429,20:00 & 08:00,Aman (32.7Â°C),[Heavy User] Penggunaan harian Anda mencapai 103.3% (100%-150%). Disarankan pengisian da																		
15	D003,Apple iPhone 12 Pro (256GB),2025-12-23,79.0,103.28571428571429,20:00 & 08:00,Aman (32.7Â°C),[Heavy User] Penggunaan harian Anda mencapai 103.3% (100%-150%). Disarankan pengisian da																		
16																			

Gambar 27. Contoh output

3.8. Implementasi Visualisasi Front-End

Tahap akhir dari pengembangan sistem ini adalah penyajian data (data presentation) melalui antarmuka pengguna (UI). Visualisasi front-end dirancang untuk mengubah data mentah dari Data Warehouse menjadi informasi yang mudah dipahami oleh pengguna smartphone melalui dashboard interaktif.

3.8.1. Teknologi yang Digunakan

Antarmuka pengguna dibangun menggunakan Streamlit, sebuah kerangka kerja berbasis Python yang efisien untuk membuat aplikasi web data. Beberapa pustaka pendukung yang digunakan meliputi:

1. Bahasa Pemprograman: Python dan Pandas
2. Streamlit: Untuk struktur web, input kontrol (selectbox, button), dan tata letak (layout).
3. Plotly (Express & Graph Objects): Untuk menghasilkan grafik tren penggunaan baterai dan ramalan suhu yang interaktif.
4. SQLAlchemy: Untuk melakukan koneksi langsung dan pengambilan data dari database PostgreSQL (battery_db).
5. Pandas: Untuk manipulasi data sebelum ditampilkan ke dalam grafik.

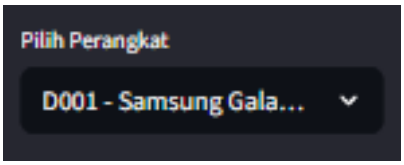
3.8.2. Arsitektur Antarmuka (Dashboard Layout)

Dashboard dirancang dengan prinsip Material Design yang bersih dan informatif, terbagi menjadi beberapa bagian utama:

1. Header & Metrik Cuaca: Bagian atas menampilkan judul sistem dan metrik kondisi cuaca real-time (suhu saat ini, kondisi langit, dan suhu yang dirasakan) di lokasi pengguna dan menampilkan semua informasi yang di perlukan.

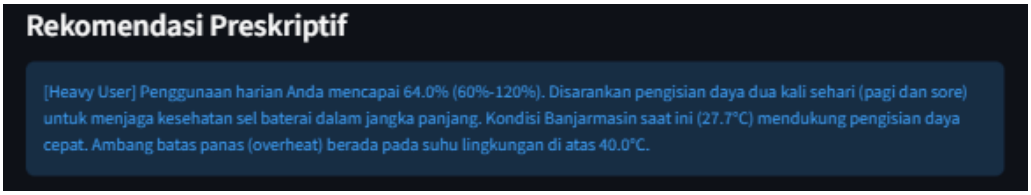
Gambar 28. Header & Metrik Cuaca

2. Kontrol Input: Area dropdown untuk memilih perangkat smartphone tertentu yang ingin dianalisis.



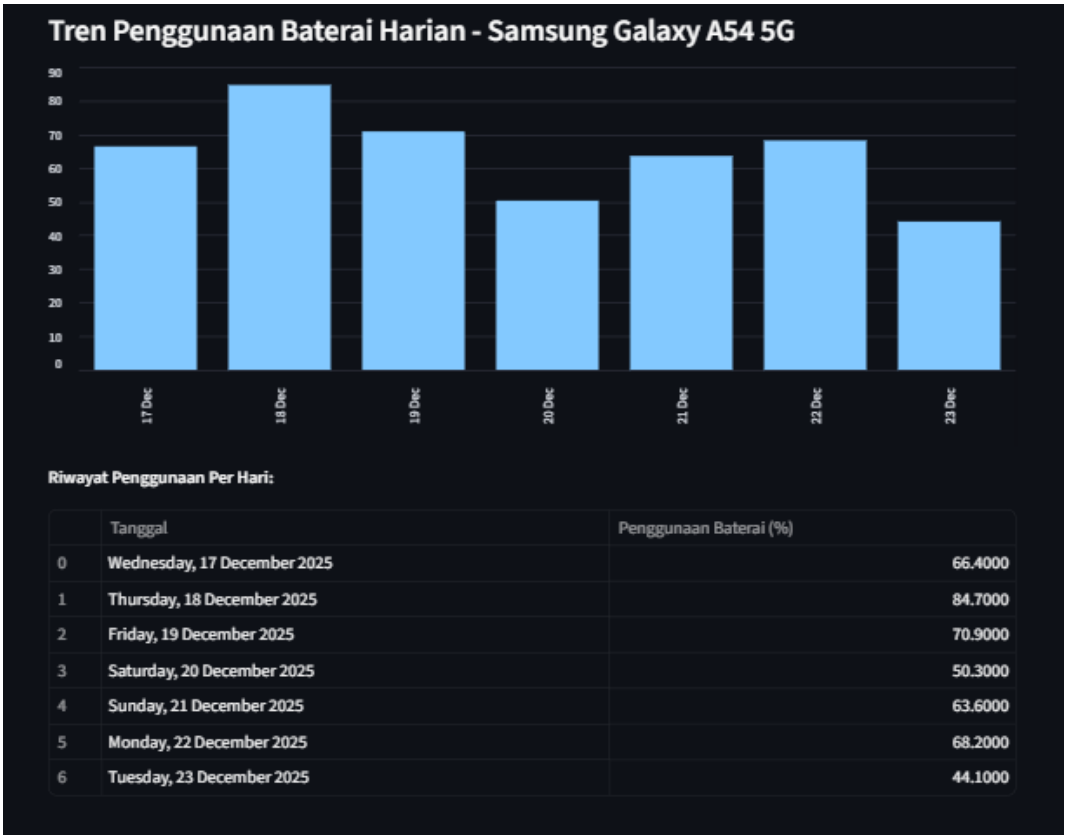
Gambar 29. Kontrol Input

3. Kartu Rekomendasi: yang menampilkan penentuan user termasuk dalam reguler/heavy/extreme yang akan diberikan rekomendasi berapa kali untuk charge dan pada waktu kapan saja, berdasarkan suhu ruangan sekarang.



Gambar 30. Kartu Rekomendasi

4. Panel Visualisasi Analitik: Bagian bawah yang terdiri dari kolom grafik besar untuk menampilkan tren data historis dan sebuah tabel memperlihatkan riaway pengguna.



Gambar 31. Panel Visualisasi Analitik

3.8.3. Fitur Interaktivitas

Untuk meningkatkan pengalaman pengguna (User Experience), dashboard dilengkapi dengan fitur:

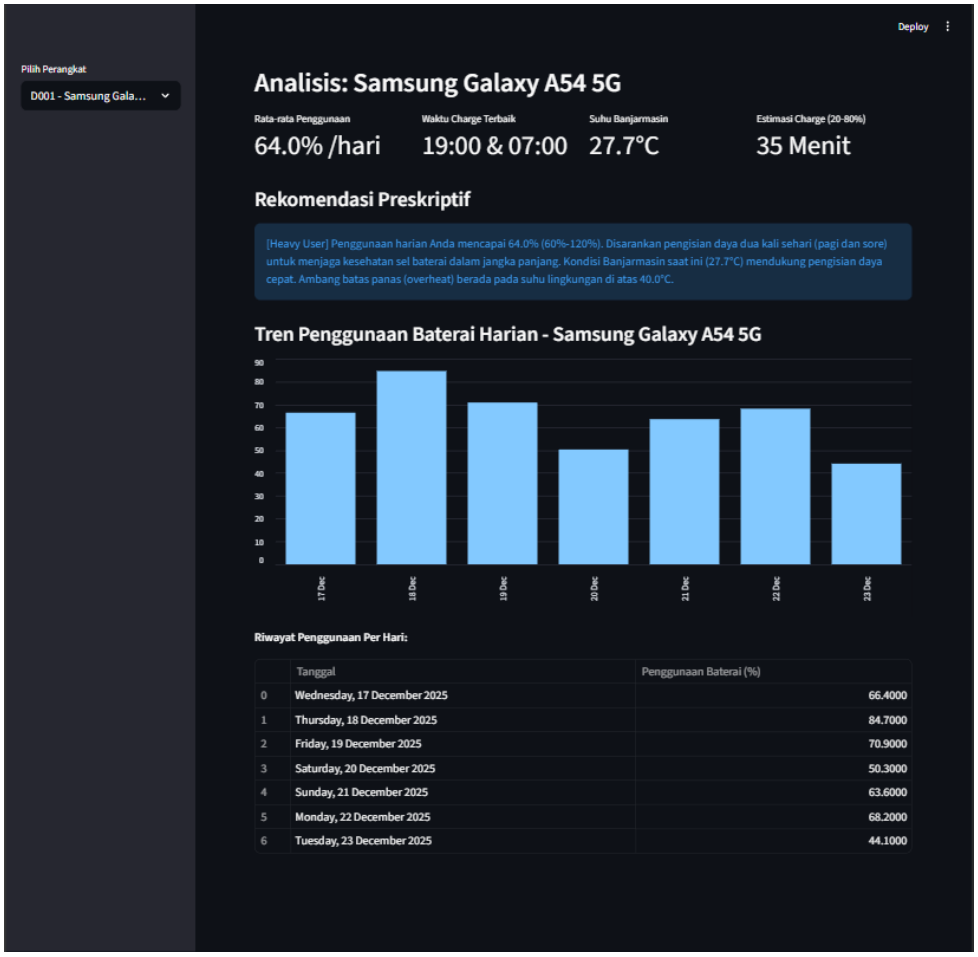
1. Dropdown Seleksi Perangkat: Pengguna dapat beralih antar perangkat (misalnya dari Samsung ke iPhone) untuk melihat rekomendasi yang dipersonalisasi sesuai spesifikasi masing-masing HP.
2. Tombol Perbarui Data: Fitur untuk membersihkan cache dan memuat ulang data terbaru dari database secara manual.
3. Grafik Interaktif Plotly: Grafik yang mendukung fitur hover (menampilkan detail data saat kursor diarahkan), zoom, serta kemampuan untuk menyembunyikan atau menampilkan seri data tertentu pada legenda.

3.8.4. Hasil Visualisasi

Implementasi visualisasi pada dashboard "Battery Insight" menyajikan antarmuka yang terintegrasi untuk memudahkan pengguna dalam memantau kesehatan baterai secara personal. Pada bagian utama, sistem menyediakan panel navigasi di sisi kiri yang memungkinkan pemilihan perangkat berdasarkan ID dan model smartphone secara spesifik, di mana setiap perubahan pilihan akan memperbarui seluruh data visual di halaman tersebut secara otomatis. Pengguna dapat melihat ringkasan performa perangkat

melalui empat metrik utama yang ditampilkan secara horizontal di bagian atas, mencakup rata-rata konsumsi baterai harian, rekomendasi waktu pengisian daya terbaik berdasarkan kebiasaan pengguna, suhu lingkungan real-time dari lokasi pengguna yang ditarik melalui API, serta estimasi durasi pengisian daya yang dibutuhkan untuk mencapai rentang ideal 20% hingga 80%.

Inti dari hasil visualisasi ini terletak pada panel rekomendasi preskriptif yang menyajikan narasi cerdas mengenai profil penggunaan perangkat, apakah termasuk dalam kategori regular, heavy, atau extreme user. Narasi tersebut tidak hanya memberikan instruksi konkret mengenai frekuensi pengisian daya yang disarankan, tetapi juga menyertakan saran mitigasi risiko berdasarkan ambang batas panas lingkungan guna menjaga umur panjang sel baterai. Selain itu, dashboard menyajikan tren penggunaan baterai harian dalam bentuk grafik batang (bar chart) yang informatif untuk membantu pengguna melihat fluktuasi konsumsi daya dari waktu ke waktu, disertai dengan tabel riwayat penggunaan yang merinci data setiap tanggalnya secara transparan. Untuk memastikan data yang ditampilkan selalu mutakhir, tersedia tombol interaktif di bagian atas yang terhubung langsung dengan Apache Airflow untuk memicu ulang seluruh proses ELT dan analisis data dari Gold Layer secara real-time langsung dari antarmuka web.



Gambar 32. Dashboard secara keseluruhan

3.9. Output Scheduler

Untuk menjamin keberlanjutan data (data freshness) dan reliabilitas sistem tanpa intervensi manual yang berulang, sistem ini menerapkan mekanisme orkestrasi otomatis menggunakan Apache Airflow Scheduler. Scheduler ini berfungsi sebagai jantung operasional yang memantau waktu dan memicu eksekusi Directed Acyclic Graph (DAG) battery_data_lake_dag secara periodik. Implementasi ini krusial untuk memastikan bahwa transformasi data dari lapisan Bronze (data mentah), Silver (pembersihan), hingga Gold (rekomendasi) selalu diperbarui secara konsisten setiap harinya.

3.9.1. Konfigurasi DAG Scheduler

Penjadwalan didefinisikan secara deklaratif menggunakan bahasa Python di dalam objek DAG Airflow. Berdasarkan konfigurasi yang diterapkan pada file dags/data_lake_dag.py, parameter penjadwalan diatur sebagai berikut:

```
18  default_args = {
19      'owner': 'airflow',
20      'depends_on_past': False,
21      'start_date': datetime(2025, 1, 1),
22      'retries': 1,
23      'retry_delay': timedelta(minutes=5),
24  }
```

Gambar 33. Konfigurasi DAG Scheduler

Konfigurasi diatas memiliki arti sebagai berikut:

- schedule_interval=timedelta(days=1): Instruksi ini menetapkan frekuensi eksekusi menjadi Harian (Daily). Airflow akan membuat satu DAG Run baru setiap kali satu periode 24 jam selesai.
- start_date=datetime(2025, 1, 1): Menandakan tanggal efektif dimulainya penjadwalan.
- catchup=False: Mencegah sistem menjalankan backfill (pemrosesan data masa lalu secara massal) saat scheduler baru dinyalakan, sehingga sistem hanya fokus pada interval waktu terkini.

Pemilihan siklus eksekusi harian ini didasarkan pada strategi "Full-Day Insight" untuk menjamin akurasi analisis perilaku pengguna. Secara teknis, sistem perlu menghitung akumulasi penggunaan baterai dan pola waktu pengecasan dalam satu siklus 24 jam penuh agar klasifikasi profil pengguna (seperti Extreme, Heavy, atau Regular User) menjadi valid. Dengan menjalankan pipeline tepat setelah hari berakhir, sistem memastikan seluruh data aktivitas dari pagi hingga malam telah terekam sepenuhnya tanpa ada data yang tertinggal. Selain itu, jadwal ini menjamin ketersediaanrekomendasi yang sudah diperbarui di pagi hari, sehingga pengguna mendapatkan informasi strategi pengecasan yang relevan dan siap pakai tepat saat mereka memulai aktivitas harian.

3.10. Implementasi Analisis Preskriptif (Actionable Insights)

Analisis preskriptif merupakan fitur puncak dari sistem rekomendasi ini, yang bertujuan untuk mengubah data analitik menjadi saran tindakan konkret (actionable insights) yang dapat langsung diterapkan oleh pengguna. Berbeda dengan analisis deskriptif yang hanya menjelaskan "apa yang terjadi", analisis preskriptif menjawab "apa yang harus dilakukan" untuk mencapai tujuan optimalisasi kesehatan baterai. Implementasi ini dijalankan pada Gold Layer melalui logika pemrograman di Apache Airflow dan disajikan kepada pengguna melalui dashboard.

3.10.1. Konsep Analisis Preskriptif

Analisis preskriptif dalam sistem ini dirancang untuk menjawab pertanyaan "Apa yang harus dilakukan?" dengan memberikan rekomendasi tindakan yang spesifik dan dapat ditindaklanjuti (actionable insights) guna mengoptimalkan kesehatan baterai. Berbeda dengan analisis deskriptif yang hanya memberikan laporan historis, analisis preskriptif menggunakan aturan logika (rule-based logic) yang menggabungkan berbagai parameter data untuk menghasilkan saran personal.

Proses pembentukan rekomendasi melibatkan integrasi tiga dimensi data utama:

1. Profil Intensitas Penggunaan :
   Sistem menghitung rata-rata pengurasan baterai harian (daily drain rate) pengguna. Berdasarkan ambang batas tertentu, pengguna dikategorikan ke dalam tiga profil:
   1. Regular User (< 60% per hari): Pengguna dengan intensitas normal. Sistem merekomendasikan satu siklus pengisian daya per hari pada jam kebiasaan pengguna.
   2. Heavy User (60% - 120% per hari): Pengguna dengan intensitas tinggi. Sistem merekomendasikan dua kali pengisian daya (misal: pagi dan sore) untuk menghindari pengurasan baterai hingga 0% (deep discharge).
   3. Extreme User (> 120% per hari): Pengguna sangat aktif. Sistem merekomendasikan tiga kali pengisian daya parsial (20-80%) untuk menjaga tegangan sel baterai tetap stabil.
2. Konteks Lingkungan Eksternal (Suhu):
   Memanfaatkan data real-time dari OpenWeatherMap API, sistem mengevaluasi keamanan lingkungan saat pengisian daya.
3. Estimasi & Kebiasaan Pengisian Daya:
   1. Waktu Terbaik (Best Time to Charge): Dihitung berdasarkan modus (nilai yang paling sering muncul) dari jam kebiasaan pengguna melakukan pengisian daya (historical charging logs). Ini memastikan rekomendasi sesuai dengan ritme hidup pengguna
   2. Estimasi Durasi (20-80%): Sistem menghitung estimasi waktu yang dibutuhkan untuk mengisi daya di rentang sehat (20% hingga 80%) dengan menggabungkan spesifikasi teoritis perangkat (Kapasitas mAh & Watt Charger) dan laju pengisian historis aktual pengguna.

3.10.2. Logika Penentuan Rekomendasi (Decision Logic)

Sistem menghasilkan rekomendasi preskriptif melalui serangkaian evaluasi logis yang memproses data teragregasi dari Silver Layer. Logika penentuan rekomendasi berfokus pada keseimbangan antara kebiasaan pengguna, spesifikasi teknis perangkat, dan faktor eksternal. Berikut adalah rincian algoritma yang diterapkan:

1. Penentuan Waktu Charge Terbaik
   Waktu terbaik untuk pengisian daya tidak ditentukan secara statis, melainkan dipelajari dari pola historis pengguna (metode statistik modus).
   - Input: Data historis plug_in_time (waktu mulai cas) dari charging_logs.
   - Logika:
     1. Ekstrak jam (0-23) dari setiap sesi pengisian daya.
     2. Hitung jam yang paling sering muncul (modus) untuk setiap device_id.
     3. Jika data historis tidak tersedia, sistem menggunakan default fallback jam 22:00 (malam hari).
   - Output: Jam referensi (jam_kebiasaan) yang menjadi basis rekomendasi jadwal harian.
2. Kategorisasi Profil Pengguna & Penjadwalan
   Sistem mengklasifikasikan pengguna berdasarkan total konsumsi baterai harian rata-rata.
   - Rumus: Siklus Dibutuhkan = Rata-rata Penggunaan Harian (%) / 60% (Angka 60% merepresentasikan siklus pengisian sehat ideal, yaitu dari 20% ke 80%)
   - Aturan Keputusan:
     1. IF Siklus Dibutuhkan > 2.0 (Extreme User):
        - Kondisi: Penggunaan > 120% per hari.
        - Rekomendasi: 3x Pengisian (Pagi, Siang, Malam).
        - Jadwal: jam_kebiasaan, jam_kebiasaan + 8 jam, jam_kebiasaan + 16 jam.
     2. ELSE IF Siklus Dibutuhkan > 1.0 (Heavy User):
        - Kondisi: Penggunaan 60% - 120% per hari.
        - Rekomendasi: 2x Pengisian.
        - Jadwal: jam_kebiasaan & jam_kebiasaan + 12 jam (siklus 12 jam).
     3. ELSE (Regular User):
        - Kondisi: Penggunaan < 60% per hari.
        - Rekomendasi: 1x Pengisian.
        - Jadwal: Sesuai jam_kebiasaan tunggal.
3. Evaluasi Keamanan Termal (Suhu)
   Faktor keamanan diperhitungkan dengan membandingkan data cuaca real-time dengan ambang batas keselamatan baterai Lithium-ion.
   - Input: Suhu lingkungan saat ini (Current Temperature) dari OpenWeatherMap API.
   - Logika:
     1. Batas Aman ditetapkan pada 40°C
     2. IF Suhu Saat Ini < Batas Aman: Status "Mendukung".
     3. ELSE: Status "Berisiko" (potensi overheat).

- Output: Pesan peringatan atau validasi keamanan yang disisipkan ke dalam teks rekomendasi.

4. Estimasi Durasi Pengisian Cerdas
   Sistem memprediksi berapa lama waktu yang dibutuhkan untuk mengisi daya dari 20% hingga 80% dengan pendekatan hybrid.
   - Model Teoretis: Berdasarkan spesifikasi fisik (Kapasitas Baterai / Arus Charger).

$$Waktu\ Teoretis = \frac{Kapasitas(mAh) \times 0.60}{Watt\ Charger\ \times Efisiensi\ (0.85)} \times 60$$

   - Model Empiris yaitu berdasarkan kecepatan pengisian historis rata-rata (% per menit) yang tercatat di database.
   - Rumus Akhir (Weighted Average)

$$Estimasi\ Akhir = (0.4\ \times Waktu\ Teoretis) + (0.6\ \times Waktu\ Historis)$$

   Bobot lebih besar diberikan pada data historis karena mencerminkan kondisi nyata (kualitas kabel, kesehatan baterai aktual)

3.10.3. Dampak Analisis Preskriptif (Value Delivery)

Dengan adanya implementasi analisis preskriptif ini, pengguna mendapatkan tiga manfaat nyata:

1. Perpanjangan Umur Baterai (*Battery Health Longevity*)
   Dengan merekomendasikan pola pengisian daya "20-80%" yang disesuaikan dengan intensitas penggunaan harian, sistem secara aktif mencegah pengguna melakukan siklus deep discharge (0%) atau overcharging (100% terus-menerus). Strategi ini secara ilmiah terbukti mengurangi stres kimiawi pada sel baterai Lithium-ion, sehingga memperlambat degradasi kapasitas baterai dalam jangka panjang.
2. Peningkatan Keselamatan & Pencegahan Risiko
   Integrasi data suhu lingkungan eksternal memberikan lapisan perlindungan tambahan. Dengan memberikan peringatan dini saat suhu lingkungan berisiko tinggi (>40°C), sistem membantu pengguna menghindari kejadian thermal runaway atau overheat saat pengisian daya cepat (fast charging), yang tidak hanya merusak komponen HP tetapi juga berpotensi membahayakan keselamatan pengguna.
3. Efisiensi Waktu & Kenyamanan Pengguna
   Sistem menghilangkan ketidakpastian pengguna tentang "kapan harus mengisi daya" dengan memberikan jadwal spesifik yang dipersonalisasi berdasarkan kebiasaan historis mereka. Estimasi durasi pengisian yang akurat (pendekatan hybrid teoretis & historis) memungkinkan pengguna merencanakan aktivitas mereka dengan lebih baik, memastikan perangkat selalu siap digunakan saat dibutuhkan tanpa harus terpaku menunggu proses charging yang tidak terprediksi.

BAB IV
DOKUMENTASI DAN IMPLEMENTASI SISTEM

4.1. Instruksi Menjalankan Sistem

Sistem ini dikemas menggunakan Docker untuk menjamin portabilitas dan kemudahan instalasi. Seluruh layanan(Database, Airflow, Streamlit, Minio, dan Postgres) didefinisikan dalam satu konfigurasi orkestrasi. Berikut adalah langkah-langkah untuk menjalankan sistem dari awal hingga siap digunakan.(dashboard rekomendasi).

4.1.1. Prasyarat Sistem

Sebelum melakukan instalasi dan menjalankan sistem, pastikan lingkungan pengembangan atau server memenuhi persyaratan minimum berikut:

1. Perangkat Lunak (Software Requirements)
    1. Sistem Operasi: Windows 10/11 (dengan WSL2), Linux (Ubuntu 20.04+ direkomendasikan), atau macOS.
    2. Docker & Docker Compose: Versi terbaru (Docker Desktop atau Docker Engine). Seluruh layanan (Airflow, MinIO, PostgreSQL, Streamlit) dikelola menggunakan kontainerisasi.
    3. Python 3.8 - 3.10: Diperlukan untuk pengembangan lokal, meskipun eksekusi utama berada di dalam kontainer Docker.
    4. Koneksi Internet: Diperlukan untuk proses pulling images dari Docker Hub, instalasi library melalui pip, serta pengambilan data suhu secara real-time dari OpenWeatherMap API.
2. Perangkat Keras (Hardware Requirements)
    1. Processor: Minimum 2 Core (4 Core direkomendasikan).
    2. RAM: Minimum 4 GB (8 GB direkomendasikan agar performa Airflow Scheduler dan Webserver stabil).
    3. Penyimpanan: Minimum 10 GB ruang kosong untuk penyimpanan images Docker, database metadata, serta penyimpanan objek di MinIO (Bronze/Silver/Gold layers).
3. Akses & Kredensial (Access Requirements)
    1. API Key: Kredensial aktif dari OpenWeatherMap API untuk fitur ekstraksi data cuaca.
    2. Port Availability: Pastikan port-port berikut tidak sedang digunakan oleh aplikasi lain:
        - 8080 (Airflow Webserver)
        - 9000 & 9001 (MinIO Console & API)
        - 8501 (Streamlit Dashboard)
        - 5432 (PostgreSQL)
4. Pengetahuan Teknis (Technical Skillsets)
    1. Data Lake: Struktur penyimpanan data (Object Storage).
    ~~1.~~2. ELT: Alur pemrosesan data.
    ~~2.~~3. Orkestrasi: Penggunaan Apache Airflow untuk penjadwalan tugas.

4.1.2. Langkah-Langkah Instalasi & Eksekusi

Sistem ini dikemas menggunakan kontainer Docker untuk memastikan kemudahan deployment dan isolasi lingkungan yang konsisten. Berikut adalah panduan langkah demi langkah untuk menjalankan sistem secara lokal:

A. Persiapan Lingkungan
   Pastikan Docker Desktop sudah berjalan dan Anda memiliki akses internet stabil untuk proses image pulling.
B. Menjalankan Layanan (Deployment)
   1. Eksekusi Docker Compose:
      Jalankan perintah berikut di terminal (pastikan Anda berada di direktori root proyek yang berisi docker-compose.yml): docker-compose up -d –build
   2. Tunggu beberapa saat (sekitar 2-5 menit tergantung kecepatan internet dan performa PC) hingga semua container berstatus healthy atau running. Docker akan menjalankan layanan berikut secara paralel:
      - PostgreSQL: Database backend untuk Airflow.
      - MinIO: Layanan Object Storage
      - Airflow Init: Inisialisasi database dan pembuatan user default.
      - Airflow Webserver & Scheduler: Antarmuka & orkestrator workflow.
      - Streamlit: Aplikasi dashboard pengguna.
C. Akses & Operasional Sistem
   1. Mengakses Apache Airflow (Orchestrator): http://localhost:8080
      1. Login dengan kredensial default:
         - Username: admin
         - Password: admin
      2. Cari DAG bernama battery_data_lake_dag di halaman utama
      3. Aktifkan DAG dengan menggeser toggle di sebelah kiri nama DAG ke posisi ON.
      4. (Opsional) Klik tombol "Trigger DAG" (ikon play) untuk menjalankan pipeline secara manual pertama kali.
   2. Mengakses MinIO Console (Data Lake Storage):
      1. Buka browser dan akses: http://localhost:9001
      2. Login dengan kredensial:
         - Username: minioadmin
         - Password: minioadmin
      3. Anda dapat memantau pembentukan folder bronze/, silver/, dan gold/ di dalam bucket datalake seiring berjalannya pipeline Airflow.
   3. Mengakses Dashboard Pengguna (Streamlit):
      1. Buka browser dan akses: http://localhost:8501
      2. Dashboard akan menampilkan antarmuka "Battery Insight & Recommendation".
      3. Jika data belum muncul, klik tombol " Refresh & Update Data" pada dashboard untuk memicu pipeline Airflow secara otomatis dan mengambil data terbaru dari Gold Layer.

4.1.3. Mengakses Antarmuka Pengguna

Setelah sistem berjalan, Anda dapat mengakses antarmuka melalui peramban web (web browser):

1. Akses Dashboard Monitoring (Streamlit)

- URL: http://localhost:8501

- Ini adalah tampilan utama untuk pengguna akhir. Di sini Anda dapat melihat rekomendasi pengisian daya, grafik tren baterai, dan prediksi cuaca.

2 Akses Manajemen Pipeline (Apache Airflow)

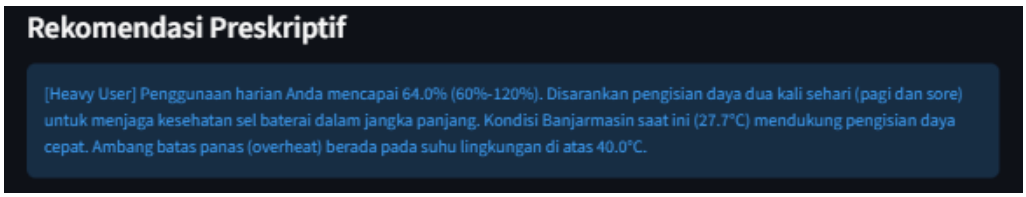
- URL: http://localhost:8085
- Username: admin
  Password: admin
- Gunakan antarmuka ini untuk memantau status ELT, melihat log error, atau memicu pipeline secara manual.

4.2. Instruksi Navigasi Zona Atas

Gambar 34. Instruksi Navigasi Zona Atas

Setelah data dimuat, sistem akan menyajikan antarmuka. Pengguna dapat melihat suhu real-time dan analisis beberapa indikator.

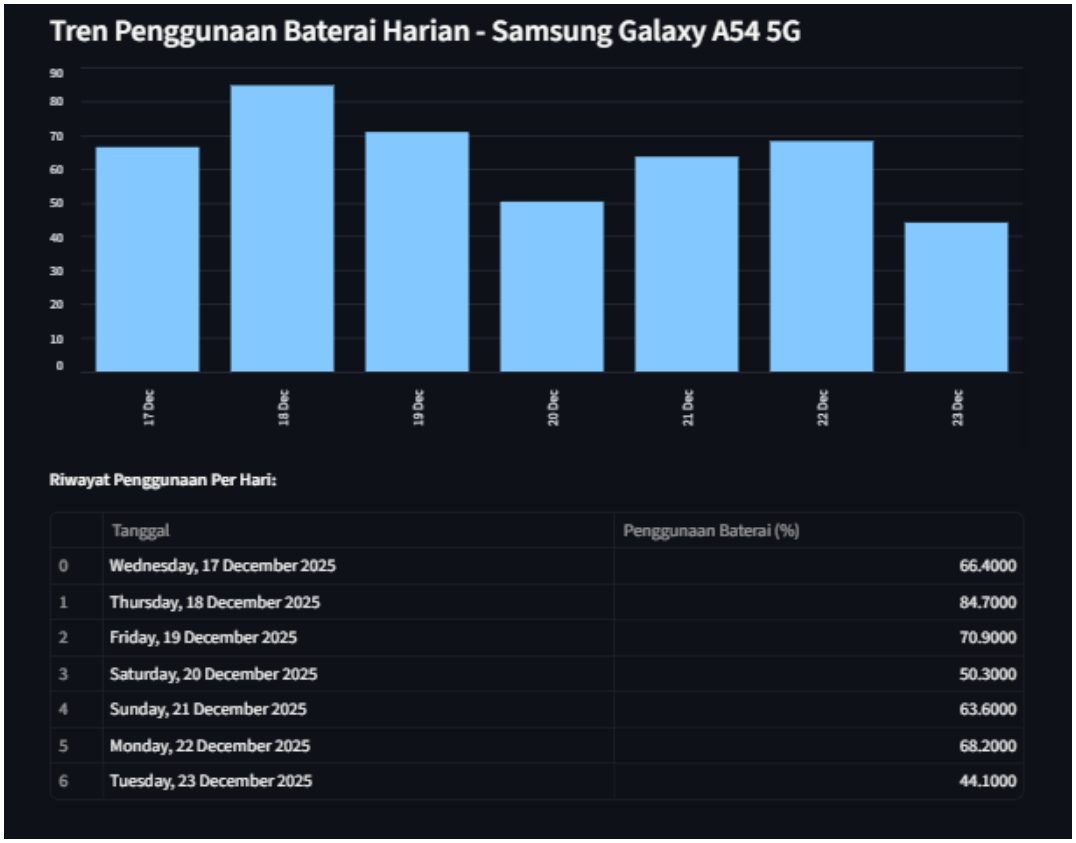
4.3. Instruksi Navigasi Zona Tengah



Gambar 35. Instruksi Navigasi Zona Tengah

Ini adalah kolom untuk hasil rekomendasi yang didapat dari hasil analisis.

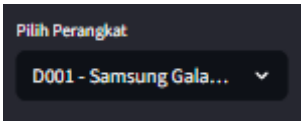
4.4. Instruksi Navigasi Zona Bawah



Gambar 36. Instruksi Navigasi Zona Bawah

Ini adalah bagian Tren pengguna harian dan riwayat penggunaan.

4.5. Intruksi Sidebar



Gambar 37. Intruksi Sidebar

Ini adalah bagian sidebar pengguna dapat memilih device yang telah di input.

BAB V
kesimpulan

5.1. Kesimpulan

Berdasarkan implementasi sistem monitoring dan rekomendasi pengisian daya baterai cerdas yang telah dilakukan, dapat ditarik beberapa kesimpulan utama sebagai berikut:

1. Implementasi MinIO sebagai pusat penyimpanan objek dengan skema Medallion Architecture (Bronze, Silver, Gold) berhasil menyatukan data heterogen (SQL, CSV, dan API) ke dalam satu ekosistem yang terorganisir dan mudah dikelola.
2. Penggunaan Apache Airflow memastikan seluruh alur data, mulai dari ekstraksi hingga transformasi menjadi Analytical Wide Table, berjalan secara otomatis dan terjadwal.
3. Visualisasi Data yang Interaktif Melalui dashboard Streamlit, data teknis yang kompleks diubah menjadi informasi yang mudah dipahami. Fitur unggulan seperti estimasi durasi pengisian (dari 20% ke 80%) dan memberikan wawasan praktis bagi pengguna dalam manajemen daya perangkat.

5.2. Insight dari Implementasi Permasalahan

Dari proses implementasi solusi teknis terhadap permasalahan manajemen baterai, diperoleh beberapa insight mendalam:

1. data cuaca eksternal dan spesifikasi teknis perangkat memiliki pengaruh signifikan terhadap laju pengisian daya. Dengan memusatkan data tersebut di Implementasi ini menunjukkan bahwa profil penggunaan (Regular, Heavy, Extreme) jauh lebih efektif dalam memberikan saran tindakan daripada instruksi umum. Sebagai contoh, pengguna dengan kategori Extreme mendapatkan jadwal pengisian daya tiga kali sehari untuk menjaga rentang 20-80%, yang secara langsung memitigasi stres termal pada sel baterai.
2. Otomatisasi Mengurangi Latensi Informasi: Penggunaan orkestrator Apache Airflow yang berjalan secara harian memastikan pengolahan data dilakukan tanpa intervensi manual. Otomatisasi pipa data menjadi syarat mutlak agar rekomendasi yang dihasilkan tetap aktual dan relevan terhadap perubahan cuaca maupun pola pemakaian.
3. Pemisahan layer penyimpanan membuktikan bahwa pemrosesan data di dalam Data Lake menjadi lebih aman. Data mentah di Bronze Layer bertindak sebagai "asuransi data" yang memungkinkan tim pengembang untuk menguji ulang algoritma rekomendasi baru tanpa harus mengganggu operasional database produksi PostgreSQL.

DAFTAR PUSTAKA

[1] R. Fahrizal dan D. S. Rahayu, "Analisis Pengaruh Suhu Operasional Terhadap Degradasi Kapasitas Baterai Lithium-Ion pada Perangkat Seluler," *Jurnal Teknik Elektro dan Komputer (JITEK)*, vol. 11, no. 2, pp. 85-92, 2022.

[2] M. S. Noor dan A. T. Pratama, "Studi Pola Siklus Pengisian Daya dan Dampaknya terhadap Pembentukan Kristal Elektroda Baterai Smartphone," *Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK)*, vol. 9, no. 4, pp. 745-754, 2022.

[3]Baskoro, Farid, et al. "MONITORING ARUS, TEGANGAN, DAN SUHU PADA PROTOTYPE THERMOELECTRIC GENERATOR BERBASIS IoT: MONITORING ARUS, TEGANGAN, DAN SUHU PADA PROTOTYPE THERMOELECTRIC GENERATOR BERBASIS IoT." *Jurnal Teknik Elektro* 10.1 (2021): 137-144.

[4] F. H. Saputra dan I. G. N. Mantra, "Perancangan Arsitektur Data Lakehouse untuk Manajemen Data Telemetri Sensor Real-Time," *Jurnal Nasional Teknologi dan Sistem Informasi*, vol. 8, no. 1, pp. 10-18, 2022.

[5] H. Kurniawan dan A. S. Wiguna, "Integrasi Data Lake dan Data Warehouse dengan Konsistensi Transaksi ACID pada Lingkungan Big Data," *Jurnal Informatika dan Sistem Informasi*, vol. 5, no. 2, pp. 112-120, 2021.

[6] R. Wijaya dan S. Hartati, "Penerapan Analitik Preskriptif pada Sistem Rekomendasi Pengisian Daya Baterai Berbasis Karakteristik Pengguna," *Jurnal Sistem Informasi Bisnis (JSINBIS)*, vol. 12, no. 1, pp. 45-53, 2022.

[7] T. Maryanto dan B. Sudarsono, "Transformasi Data Sensor Menjadi Kebijakan Preskriptif untuk Memperpanjang Usia Pakai Perangkat Keras melalui Business Intelligence," *Jurnal Riset Teknologi Informasi dan Komputer*, vol. 15, no. 3, pp. 201-210, 2023.

LAMPIRAN

Data Source

log_aktivitas_baterai_hp.csv

```
log_id;device_id;activity_date;application_name;screen_time_minutes;battery_usage_percent;activity_category
1;D001;23/12/2025;Instagram;91;15,6;Social Media
2;D001;23/12/2025;TikTok;59;12,9;Social Media
3;D001;23/12/2025;WhatsApp;20;3;Messaging
4;D001;23/12/2025;YouTube Music;26;2,6;Music
5;D001;23/12/2025;Layanan Google Play;0;2,4;System
6;D001;23/12/2025;UI sistem;0;1,7;System
7;D001;23/12/2025;Galaxy Store;0;1;System
8;D001;23/12/2025;Google Play Store;0;1;System
9;D001;23/12/2025;Shopee;3;0,9;E-Commerce
10;D001;23/12/2025;KCMTix;11;0,7;Entertainment
11;D001;23/12/2025;Maps;3;0,7;Navigation
12;D001;23/12/2025;Grab;1;0,7;Transportation
13;D001;23/12/2025;Chrome;2;0,3;Browser
14;D001;23/12/2025;GoPay;2;0,3;Finance
15;D001;23/12/2025;Livin' by Mandiri;2;0,3;Finance
16;D003;23/12/2025;TikTok;229;51;Social Media
17;D003;23/12/2025;Safari;47;14;Browser
18;D003;23/12/2025;WhatsApp;38;9;Messaging
19;D003;23/12/2025;Preview;10;3;Tools
20;D003;23/12/2025;Shopee;2;1;E-Commerce
21;D003;23/12/2025;YouTube;2;1;Entertainment
22;D001;22/12/2025; TikTok;138;24,4; Social Media
23;D001;22/12/2025; Instagram;137;19,2; Social Media
24;D001;22/12/2025; Shopee;19;6,5; E-Commerce
25;D001;22/12/2025; WhatsApp;19;5,7; Messaging
26;D001;22/12/2025; Layanan Google Play;0;2,1; System
27;D001;22/12/2025; Traveloka;17;2; Travel
28;D001;22/12/2025; Chrome;8;1,6; Browser
29;D001;22/12/2025; UI sistem;0;1,5; System
30;D001;22/12/2025; Beranda One UI;13;1,3; System
31;D001;22/12/2025; Google;3;0,6; Utility
32;D001;22/12/2025; Flightradar24;2;0,6; Utility
33;D001;22/12/2025; YouTube;1;0,5; Streaming
34;D001;22/12/2025; Google Play Store;0;0,3; System
35;D001;22/12/2025; Keyboard Samsung;0;0,3; System
36;D001;22/12/2025; Kalkulator;4;0,2; Utility
37;D001;22/12/2025; KCMTix;3;0,2; Entertainment
```

38;D001;22/12/2025; Kamera;1;0,2; Camera
39;D001;22/12/2025; Livin' by Mandiri;1;0,2; Finance
40;D001;22/12/2025; Maps;1;0,2; Navigation
41;D001;22/12/2025; Telegram;1;0,2; Messaging
42;D001;22/12/2025; Galeri;1;0,1; Utility
43;D001;22/12/2025; Facebook;0;0,1; Social Media
44;D001;22/12/2025; Gmail;0;0,1; Email
45;D001;22/12/2025; Gojek;0;0,1; Transportation
46;D001;17/12/2025;TikTok;217;34,4;Entertainment
47;D001;17/12/2025;Instagram;110;20,5;Social Media
48;D001;17/12/2025;WhatsApp;15;3,7;Communication
49;D001;17/12/2025;Layanan Google Play;0,5;2,3;System
50;D001;17/12/2025;CamScanner;10;1,4;Productivity
51;D001;17/12/2025;MitraDarat;4;1,2;Travel
52;D001;17/12/2025;YouTube Music;0,5;1,1;Entertainment
53;D001;17/12/2025;UI Sistem;0,5;1;System
54;D001;17/12/2025;Shopee;2;0,8;Shopping
55;D001;18/12/2025;TikTok;253;37,4;Entertainment
56;D001;18/12/2025;Instagram;206;33,7;Social Media
57;D001;18/12/2025;Shopee;25;5;Shopping
58;D001;18/12/2025;Layanan Google Play;0,5;2,1;System
59;D001;18/12/2025;WhatsApp;15;1,6;Communication
60;D001;18/12/2025;UI Sistem;0,5;1,6;System
61;D001;18/12/2025;Chrome;3;1,4;Browser
62;D001;18/12/2025;Beranda One UI;6;0,9;System
63;D001;18/12/2025;BitLife;5;0,5;Games
64;D001;18/12/2025;ChatGPT;5;0,5;Productivity
65;D001;19/12/2025;TikTok;301;38,2;Entertainment
66;D001;19/12/2025;Instagram;111;18,9;Social Media
67;D001;19/12/2025;Shopee;23;6,8;Shopping
68;D001;19/12/2025;Layanan Google Play;0,5;2;System
69;D001;19/12/2025;ZALORA;10;1,4;Shopping
70;D001;19/12/2025;UI Sistem;0,5;1,1;System
71;D001;19/12/2025;Beranda One UI;5;1;System
72;D001;19/12/2025;WhatsApp;5;1;Communication
73;D001;19/12/2025;Samsung Internet;4;0,5;Browser
74;D001;20/12/2025;TikTok;159;18,4;Entertainment
75;D001;20/12/2025;Instagram;96;15,1;Social Media
76;D001;20/12/2025;Shopee;23;5;Shopping
77;D001;20/12/2025;Drive;2;4,1;Productivity
78;D001;20/12/2025;Layanan Google Play;0,5;2,3;System
79;D001;20/12/2025;WhatsApp;11;1,5;Communication
80;D001;20/12/2025;UI Sistem;0,5;1,2;System
81;D001;20/12/2025;Subway Surf;7;1;Games
82;D001;20/12/2025;Galaxy Buds;0,5;0,9;Tools
83;D001;20/12/2025;YouTube Music;4;0,8;Entertainment
84;D001;21/12/2025;TikTok;186;28,3;Entertainment

85;D001;21/12/2025;Instagram;63;13,8;Social Media
86;D001;21/12/2025;WhatsApp;34;6,6;Communication
87;D001;21/12/2025;BitLife;28;4,5;Games
88;D001;21/12/2025;Layanan Google Play;0,5;2,7;System
89;D001;21/12/2025;Shopee;5;2,2;Shopping
90;D001;21/12/2025;Eraspace;12;1,9;Shopping
91;D001;21/12/2025;Chrome;1;1,4;Browser
92;D001;21/12/2025;UI Sistem;0,5;1,2;System
93;D001;21/12/2025;Panggilan Suara (WA);0,5;1;Communication
94;D003;22/12/2025;TikTok;184;48;Entertainment
95;D003;22/12/2025;Safari;91;24;Browser
96;D003;22/12/2025;WhatsApp;142;18;Communication
97;D003;22/12/2025;Preview;4;2;Tools
98;D003;22/12/2025;Youtube;4;1;Entertainment
99;D003;22/12/2025;Instagram;1;1;Social Media
100;D003;21/12/2025;Safari;169;51;Browser
101;D003;21/12/2025;TikTok;114;31;Entertainment
102;D003;21/12/2025;WhatsApp;151;26;Communication
103;D003;21/12/2025;Shopee;25;7;E-Commerce
104;D003;21/12/2025;Discord;3;1;Communication
105;D003;21/12/2025;Youtube;3;1;Entertainment
106;D003;20/12/2025;TikTok;144;40;Entertainment
107;D003;20/12/2025;Safari;136;34;Browser
108;D003;20/12/2025;WhatsApp;105;18;Communication
109;D003;20/12/2025;Clash of Clans;45;10;Games
110;D003;20/12/2025;Shopee;8;3;E-Commerce
111;D003;20/12/2025;Youtube;11;3;Entertainment
112;D003;20/12/2025;Voice Memos;7;1;Tools
113;D003;19/12/2025;TikTok;159;38;Entertainment
114;D003;19/12/2025;Safari;89;23;Browser
115;D003;19/12/2025;WhatsApp;81;13;Communication
116;D003;19/12/2025;Clash of Clans;21;4;Games
117;D003;19/12/2025;Youtube;16;4;Entertainment
118;D003;19/12/2025;Shopee;1;1;E-Commerce
119;D003;18/12/2025;TikTok;154;44;Entertainment
120;D003;18/12/2025;Safari;45;19;Browser
121;D003;18/12/2025;WhatsApp;61;18;Communication
122;D003;18/12/2025;Clash of Clans;47;17;Games
123;D003;18/12/2025;Gojek;22;7;Travel
124;D003;18/12/2025;Shopee;4;4;E-Commerce
125;D003;18/12/2025;App Store;1;1;System
126;D003;18/12/2025;Gmail;4;1;Tools
127;D003;18/12/2025;Photos;1;1;Tools
128;D003;17/12/2025;TikTok;209;48;Entertainment
129;D003;17/12/2025;WhatsApp;271;40;Communication
130;D003;17/12/2025;Safari;133;30;Browser
131;D003;17/12/2025;Shopee;14;4;E-Commerce

132;D003;17/12/2025;Clash of Clans;7;3;Games
133;D003;17/12/2025;Voice Memos;1;1;Tools
134;D003;17/12/2025;Photos;1;1;Tools
135;D003;17/12/2025;Livin' by Mandiri;4;1;Tools
136;D003;17/12/2025;Instagram;2;1;Entertainment

smartphones.csv

brand_name;model;price;avg_rating;5G_or_not;processor_brand;num_cores;processor_speed;battery_capacity;fast_charging_available;fast_charging;ram_capacity;internal_memory;screen_size;refresh_rate;num_rear_cameras;os;primary_camera_rear;primary_camera_front;extended_memory_available;resolution_height;resolution_width
apple;Apple iPhone 11;38999;7.3;0;bionic;6;2.65;3110;0;;4;64;6.1;60;2;ios;12;12;0;1792;828
apple;Apple iPhone 11 (128GB);46999;7.5;0;bionic;6;2.65;3110;0;;4;128;6.1;60;2;ios;12;12;0;1792;828
apple;Apple iPhone 11 Pro Max;109900;7.7;0;bionic;6;2.65;3500;1;18;4;64;6.5;60;3;ios;12;12;0;2688;1242
apple;Apple iPhone 12;51999;7.4;1;bionic;6;3.1;;0;;4;64;6.1;60;2;ios;12;12;0;2532;1170
apple;Apple iPhone 12 (128GB);55999;7.5;1;bionic;6;3.1;;0;;4;128;6.1;60;2;ios;12;12;0;2532;1170
apple;Apple iPhone 12 (256GB);67999;7.6;1;bionic;6;3.1;;0;;4;256;6.1;60;2;ios;12;12;0;2532;1170
apple;Apple iPhone 12 Mini;40999;7.4;1;bionic;6;3.1;;0;;4;64;5.4;60;2;ios;12;12;0;2340;1080
apple;Apple iPhone 12 Mini (128GB);45999;7.5;1;bionic;6;3.1;;0;;4;128;5.4;60;2;ios;12;12;0;2340;1080
apple;Apple iPhone 12 Mini (256GB);55999;7.5;1;bionic;6;3.1;;0;;4;256;5.4;60;2;ios;12;12;0;2340;1080
apple;Apple iPhone 12 Pro (256GB);119900;8;1;bionic;6;3.1;;0;;6;256;6.1;60;3;ios;12;12;0;2532;1170
apple;Apple iPhone 12 Pro (512GB);139900;8;1;bionic;6;3.1;;0;;6;512;6.1;60;3;ios;12;12;0;2532;1170
apple;Apple iPhone 13;62999;7.9;1;bionic;6;3.22;3240;1;;4;128;6.1;60;2;ios;12;12;0;2532;1170
apple;Apple iPhone 13 (256GB);72999;7.9;1;bionic;6;3.22;3240;1;;4;256;6.1;60;2;ios;12;12;0;2532;1170
apple;Apple iPhone 13 (512GB);91999;8;1;bionic;6;3.22;3240;1;;4;512;6.1;60;2;ios;12;12;0;2532;1170
apple;Apple iPhone 13 Mini;64900;7.9;1;bionic;6;3.22;2438;1;;4;128;5.4;60;2;ios;12;12;0;2340;1080
apple;Apple iPhone 13 Pro;119900;8.3;1;bionic;6;3.22;3095;1;;6;128;6.1;120;3;ios;12;12;0;2532;1170

apple;Apple iPhone 13 Pro (1TB);147900;8.4;1;bionic;6;3.22;3095;1;;6;1024;6.1;120;3;ios;12;12;0;2532;1170
apple;Apple iPhone 13 Pro (256GB);129900;8.3;1;bionic;6;3.22;3095;1;;6;256;6.1;120;3;ios;12;12;0;2532;1170
apple;Apple iPhone 13 Pro Max;129900;8.4;1;bionic;6;3.22;4352;1;;6;128;6.7;120;3;ios;12;12;0;2778;1284
apple;Apple iPhone 13 Pro Max (1TB);179900;8.6;1;bionic;6;3.22;4352;1;;6;1024;6.7;120;3;ios;12;12;0;2778;1284
apple;Apple iPhone 13 Pro Max (256GB);139900;8.4;1;bionic;6;3.22;4352;1;;6;256;6.7;120;3;ios;12;12;0;2778;1284
apple;Apple iPhone 14;65999;8.1;1;bionic;6;3.22;3279;1;;6;128;6.1;60;2;ios;12;12;0;2532;1170
apple;Apple iPhone 14 (256GB);75999;8.2;1;bionic;6;3.22;3279;1;;6;256;6.1;60;2;ios;12;12;0;2532;1170
apple;Apple iPhone 14 (512GB);95999;8.2;1;bionic;6;3.22;3279;1;;6;512;6.1;60;2;ios;12;12;0;2532;1170
apple;Apple iPhone 14 Mini;69990;7;0;bionic;;;3500;1;;6;128;5.42;60;2;ios;12;12;0;2340;1080
apple;Apple iPhone 14 Plus;74999;8.2;1;bionic;6;3.22;4325;1;;6;128;6.7;60;2;ios;12;12;0;2778;1284
apple;Apple iPhone 14 Plus (256GB);84999;8.3;1;bionic;6;3.22;4325;1;;6;256;6.7;60;2;ios;12;12;0;2778;1284
apple;Apple iPhone 14 Plus (512GB);104999;8.3;1;bionic;6;3.22;4325;1;;6;512;6.7;60;2;ios;12;12;0;2778;1284
apple;Apple iPhone 14 Pro;119990;7.5;1;bionic;6;;3200;1;;6;128;6.1;120;3;ios;48;12;0;2556;1179
apple;Apple iPhone 14 Pro (1TB);172999;7.7;1;bionic;6;;3200;1;;6;1024;6.1;120;3;ios;48;12;0;2556;1179
apple;Apple iPhone 14 Pro (256GB);129990;7.6;1;bionic;6;;3200;1;;6;256;6.1;120;3;ios;48;12;0;2556;1179
apple;Apple iPhone 14 Pro Max;129990;7.6;1;bionic;6;;4323;1;;6;128;6.7;120;3;ios;48;12;0;2796;1290
apple;Apple iPhone 14 Pro Max (1TB);182999;7.8;1;bionic;6;;4323;1;;6;1024;6.7;120;3;ios;48;12;0;2796;1290
apple;Apple iPhone 14 Pro Max (256GB);139990;7.7;1;bionic;6;;4323;1;;6;256;6.7;120;3;ios;48;12;0;2796;1290
apple;Apple iPhone 14 Pro Max (512GB);169900;7.8;1;bionic;6;;4323;1;;6;512;6.7;120;3;ios;48;12;0;2796;1290
apple;Apple iPhone 15;82990;7.2;0;bionic;;;3285;1;;6;128;6.06;60;2;ios;50;13;0;2532;1170
apple;Apple iPhone 15 Plus;84990;7.5;1;bionic;6;;4532;1;;8;128;6.71;120;2;ios;12;12;0;2778;1284
……

DAGS

Data_lake_dag.py

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime, timedelta
import pandas as pd
import requests
import boto3
import os
import io
import json
import psycopg2

# Configuration
MINIO_ENDPOINT = "http://minio:9000"
MINIO_ACCESS_KEY = "minioadmin"
MINIO_SECRET_KEY = "minioadmin"
BUCKET_NAME = "datalake"

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2025, 1, 1),
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

def get_minio_client():
      return  boto3.client("s3",  endpoint_url=MINIO_ENDPOINT,
aws_access_key_id=MINIO_ACCESS_KEY,
aws_secret_access_key=MINIO_SECRET_KEY)

def init_minio():
    try: get_minio_client().create_bucket(Bucket=BUCKET_NAME)
    except: pass

# --- BRONZE LAYER ---
def elt_weather_api():
      with  open('/opt/airflow/API.txt',  'r')  as  f:  url  =
f.read().strip()
    respon = requests.get(url).json()
              get_minio_client().put_object(Bucket=BUCKET_NAME,
Key="bronze/weather/current_weather.json", Body=json.dumps(respon))

def elt_sql_data():
    # Connect to the real database "battery_analytics_db"
    koneksi = psycopg2.connect(
```

```
            host="postgres",
            database="battery_analytics_db",
            user="airflow",
            password="airflow"
        )
        kursor = koneksi.cursor()

        # Extract Charging Logs
        kursor.execute("SELECT * FROM charging_logs")
        data_pengisian = kursor.fetchall()
            df_pengisian_mentah   =   pd.DataFrame(data_pengisian,
columns=['charge_id', 'device_id', 'plug_in_time', 'plug_out_time',
'start_level', 'end_level', 'charge_temp'])
        penyangga_logs = io.StringIO()
        df_pengisian_mentah.to_csv(penyangga_logs, index=False)
                get_minio_client().put_object(Bucket=BUCKET_NAME,
Key="bronze/charging_logs/logs.csv", Body=penyangga_logs.getvalue())

        # Extract Devices
        kursor.execute("SELECT * FROM devices")
        data_perangkat = kursor.fetchall()
            df_perangkat_mentah   =   pd.DataFrame(data_perangkat,
columns=['device_id', 'device_type', 'brand_name', 'model'])
        penyangga_dev = io.StringIO()
        df_perangkat_mentah.to_csv(penyangga_dev, index=False)
                get_minio_client().put_object(Bucket=BUCKET_NAME,
Key="bronze/devices/devices.csv", Body=penyangga_dev.getvalue())

        koneksi.close()

    def elt_csv_activity():
                                            df                  =
pd.read_csv('/opt/airflow/data_source/log_aktivitas_baterai_hp.csv',
sep=';')
        penyangga = io.StringIO()
        df.to_csv(penyangga, index=False)
                get_minio_client().put_object(Bucket=BUCKET_NAME,
Key="bronze/activity_logs/activity.csv", Body=penyangga.getvalue())

    def elt_csv_specs():
        df = pd.read_csv('/opt/airflow/data_source/smartphones.csv',
sep=';')
        penyangga = io.StringIO()
        df.to_csv(penyangga, index=False)
                get_minio_client().put_object(Bucket=BUCKET_NAME,
Key="bronze/smartphone_specs/specs.csv", Body=penyangga.getvalue())

    # --- SILVER LAYER ---
```

```
    def etl_silver_layer():
        s3 = get_minio_client()

        # Activity
                                        df_aktivitas                =
pd.read_csv(io.BytesIO(s3.get_object(Bucket=BUCKET_NAME,
Key="bronze/activity_logs/activity.csv")['Body'].read()))
        if df_aktivitas['battery_usage_percent'].dtype == 'object':
df_aktivitas['battery_usage_percent']                               =
df_aktivitas['battery_usage_percent'].str.replace(',',
'.').astype(float)
                              df_aktivitas['activity_date']         =
pd.to_datetime(df_aktivitas['activity_date'], dayfirst=True)

        # Charging Logs (From DB CSV dump)
                                            df_cas                  =
pd.read_csv(io.BytesIO(s3.get_object(Bucket=BUCKET_NAME,
Key="bronze/charging_logs/logs.csv")['Body'].read()))
        df_cas['plug_in'] = pd.to_datetime(df_cas['plug_in_time'])
        df_cas['plug_out'] = pd.to_datetime(df_cas['plug_out_time'])
        df_cas = df_cas.rename(columns={'start_level': 'start_lvl',
'end_level': 'end_lvl', 'charge_temp': 'temp'})

        # Devices (From DB CSV dump)
                                        df_perangkat                =
pd.read_csv(io.BytesIO(s3.get_object(Bucket=BUCKET_NAME,
Key="bronze/devices/devices.csv")['Body'].read()))

        # Specs
                                          df_spek                   =
pd.read_csv(io.BytesIO(s3.get_object(Bucket=BUCKET_NAME,
Key="bronze/smartphone_specs/specs.csv")['Body'].read()))

        # Save Cleaned
        for nama, data in [('activity', df_aktivitas), ('charging',
df_cas), ('specs', df_spek), ('devices', df_perangkat)]:
                penyangga  =  io.StringIO();  data.to_csv(penyangga,
index=False)
                                    s3.put_object(Bucket=BUCKET_NAME,
Key=f"silver/{nama}/clean_{nama}.csv", Body=penyangga.getvalue())

    # --- GOLD LAYER ---
    def etl_gold_analysis():
        s3 = get_minio_client()
                                        df_aktivitas                =
pd.read_csv(io.BytesIO(s3.get_object(Bucket=BUCKET_NAME,
Key="silver/activity/clean_activity.csv")['Body'].read()))
```

```
                                                df_cas                  =
pd.read_csv(io.BytesIO(s3.get_object(Bucket=BUCKET_NAME,
Key="silver/charging/clean_charging.csv")['Body'].read()))
                                                df_spek                 =
pd.read_csv(io.BytesIO(s3.get_object(Bucket=BUCKET_NAME,
Key="silver/specs/clean_specs.csv")['Body'].read()))
                                            df_perangkat                =
pd.read_csv(io.BytesIO(s3.get_object(Bucket=BUCKET_NAME,
Key="silver/devices/clean_devices.csv")['Body'].read()))

        try:
                                                    data_cuaca          =
json.loads(s3.get_object(Bucket=BUCKET_NAME,
Key="bronze/weather/current_weather.json")['Body'].read().decode('ut
f-8'))
            suhu_saat_ini, lokasi = data_cuaca['current']['temp_c'],
data_cuaca['location']['name']
        except: suhu_saat_ini, lokasi = 30.0, "Unknown"

        # Map Device ID to Model Name using the devices table
           peta_perangkat   =   pd.Series(df_perangkat.model.values,
index=df_perangkat.device_id).to_dict()

        # Historical Rate
          df_cas['durasi']  =  (pd.to_datetime(df_cas['plug_out'])  -
pd.to_datetime(df_cas['plug_in'])).dt.total_seconds() / 60
        df_cas = df_cas[df_cas['durasi'] > 0]
         df_cas['laju'] = (df_cas['end_lvl'] - df_cas['start_lvl'])
/ df_cas['durasi']
                                        rata_rata_laju                  =
df_cas.groupby('device_id')['laju'].mean().to_dict()

        # Historical Charging Hour Pattern (Mode)
                                      df_cas['jam_cas']                 =
pd.to_datetime(df_cas['plug_in']).dt.hour
                                         pola_jam_cas                   =
df_cas.groupby('device_id')['jam_cas'].agg(lambda  x:  x.mode()[0]  if
not x.mode().empty else None).to_dict()

              data_harian    =    df_aktivitas.groupby(['device_id',
'activity_date'])['battery_usage_percent'].sum().reset_index()
        hasil_akhir = []
        for id_perangkat in data_harian['device_id'].unique():
            harian_perangkat = data_harian[data_harian['device_id']
== id_perangkat]
                                         rata_rata_penggunaan           =
harian_perangkat['battery_usage_percent'].mean()
            model = peta_perangkat.get(id_perangkat, "Unknown")
```

```
            # Spec Lookup
                                                    cocok           =
df_spek[df_spek['model'].str.contains(model.split('(')[0].strip(),
case=False, na=False)]
            kapasitas, watt = 4000, 15
            if not cocok.empty:
                  kapasitas = cocok.iloc[0].get('battery_capacity',
4000)
                if pd.isna(kapasitas): kapasitas = 4000
                watt = cocok.iloc[0].get('fast_charging', 15)
                if pd.isna(watt) or watt == 0: watt = 20 if "iPhone"
in model else 15

            # Smart Estimation
            menit_teoretis = (0.6 * kapasitas * 3.7 / 1000) / (watt
* 0.85) * 60
            laju_historis = rata_rata_laju.get(id_perangkat, 1.0)
               if pd.isna(laju_historis) or laju_historis <= 0:
laju_historis = 1.0
            menit_historis = 60 / laju_historis
                estimasi_menit = int((menit_teoretis * 0.4) +
(menit_historis * 0.6))

            persen_cas_dibutuhkan = 60 # Target 20% to 80%
                    siklus_dibutuhkan = rata_rata_penggunaan /
persen_cas_dibutuhkan

            batas_aman_suhu = 40.0
            apakah_aman = suhu_saat_ini < batas_aman_suhu

            # Get Historical Charging Hour Preference
            jam_kebiasaan = pola_jam_cas.get(id_perangkat)
             if jam_kebiasaan is None: jam_kebiasaan = 22 # Default
if no data
            jam_kebiasaan = int(jam_kebiasaan)

            # Build recommendation based on calculated metrics
            if siklus_dibutuhkan > 2.0:
                profil = "Extreme User"
                       waktu_terbaik = f"{jam_kebiasaan:02d}:00,
{(jam_kebiasaan+8)%24:02d}:00 & {(jam_kebiasaan+16)%24:02d}:00"
                saran = f"Dengan rata-rata pengurasan baterai sebesar
{rata_rata_penggunaan:.1f}% per hari (>120%), perangkat Anda
dikategorikan sangat intensif. Disarankan melakukan 3 kali pengisian
daya sehari (pagi, siang, dan malam) guna menjaga kesehatan baterai
di rentang 20-80%."
            elif siklus_dibutuhkan > 1.0:
```

```
                profil = "Heavy User"
                      waktu_terbaik  =  f"{jam_kebiasaan:02d}:00  &
{(jam_kebiasaan+12)%24:02d}:00"
                     saran  =  f"Penggunaan  harian  Anda  mencapai
{rata_rata_penggunaan:.1f}% (60%-120%). Disarankan pengisian daya dua
kali sehari (pagi dan sore) untuk menjaga kesehatan sel baterai dalam
jangka panjang."
            else:
                profil = "Regular User"
                waktu_terbaik = f"{jam_kebiasaan:02d}:00"
                saran = f"Konsumsi daya harian Anda stabil di angka
{rata_rata_penggunaan:.1f}% (<60%). Cukup lakukan satu kali pengisian
daya harian."

                 saran_cuaca  =  f"Kondisi  Banjarmasin  saat  ini
({suhu_saat_ini:.1f}°C)  {'mendukung' if apakah_aman else 'berisiko
untuk'} pengisian daya cepat. Ambang batas panas (overheat) berada
pada suhu lingkungan di atas {batas_aman_suhu}°C."

            rekomendasi_lengkap = f"{saran} {saran_cuaca}"

            for baris in harian_perangkat.itertuples():
                hasil_akhir.append({
                  "device_id": id_perangkat, "model": model, "date":
baris.activity_date,
                     "daily_usage":  baris.battery_usage_percent,
"avg_weekly_usage": rata_rata_penggunaan,
                               "best_time_to_charge":  waktu_terbaik,
"temp_status": f"Aman ({suhu_saat_ini+5:.1f}°C)",
                                    "recommendation":   f"[{profil}]
{rekomendasi_lengkap}",
                     "charge_estimation_20_80": f"{estimasi_menit}
Menit",
                        "outdoor_temp":  f"{suhu_saat_ini:.1f}°C",
"location": lokasi
                })

         df_gold = pd.DataFrame(hasil_akhir)
                     penyangga_gold        =        io.StringIO();
df_gold.to_csv(penyangga_gold, index=False)
                                  s3.put_object(Bucket=BUCKET_NAME,
Key="gold/recommendations/final_recommendation.csv",
Body=penyangga_gold.getvalue())

     with   DAG('battery_data_lake_dag',   default_args=default_args,
schedule_interval=timedelta(days=1), catchup=False) as dag:
```

```
                                    task_inisialisasi               =
PythonOperator(task_id='inisialisasi_minio',
python_callable=init_minio)
                                   task_ekstrak_cuaca               =
PythonOperator(task_id='ekstrak_cuaca_api',
python_callable=elt_weather_api)
                                     task_ekstrak_sql               =
PythonOperator(task_id='ekstrak_database_sql',
python_callable=elt_sql_data)
                                 task_ekstrak_aktivitas             =
PythonOperator(task_id='ekstrak_log_aktivitas',
python_callable=elt_csv_activity)
                                   task_ekstrak_specs               =
PythonOperator(task_id='ekstrak_spesifikasi_hp',
python_callable=elt_csv_specs)
                                task_transformasi_silver            =
PythonOperator(task_id='transformasi_layer_silver',
python_callable=etl_silver_layer)
                                   task_analisis_gold               =
PythonOperator(task_id='analisis_layer_gold',
python_callable=etl_gold_analysis)

         task_inisialisasi >> [task_ekstrak_cuaca, task_ekstrak_sql,
task_ekstrak_aktivitas,            task_ekstrak_specs]             >>
task_transformasi_silver >> task_analisis_gold
```

SQL

data_charging.sql

```
      CREATE TABLE IF NOT EXISTS devices (
          device_id VARCHAR(50) PRIMARY KEY,
          device_type VARCHAR(30) NOT NULL,
          brand_name VARCHAR(50) NOT NULL,
          model VARCHAR(100) NOT NULL
      );

      CREATE TABLE IF NOT EXISTS charging_logs (
          charge_id SERIAL PRIMARY KEY,
          device_id VARCHAR(50) NOT NULL,
          plug_in_time TIMESTAMP NOT NULL,
          plug_out_time TIMESTAMP NOT NULL,
          start_level INT NOT NULL,
          end_level INT NOT NULL,
          charge_temp DECIMAL(4,1) NOT NULL,
          CONSTRAINT fk_device
              FOREIGN KEY (device_id)
              REFERENCES devices(device_id)
              ON UPDATE CASCADE
              ON DELETE CASCADE
      );

      -- Clear existing data
      DELETE FROM charging_logs;
      DELETE FROM devices;

      INSERT INTO devices (device_id, device_type, brand_name,
model)
      VALUES
      ('D001', 'Smartphone', 'Samsung', 'Samsung Galaxy A54 5G'),
      ('D003', 'Smartphone', 'Apple', 'Apple iPhone 12 Pro
(256GB)');

      -- Data Dummy REALISTIC Fast Charging
      -- Samsung A54 (5000mAh): Agak lama karena baterai besar
      -- Durasi: ~45 menit untuk naik ~70%
      INSERT INTO charging_logs (device_id, plug_in_time,
plug_out_time, start_level, end_level, charge_temp)
      VALUES
      ('D001', '2025-12-17 19:00:00', '2025-12-17 19:45:00', 18,
92, 36.5),
      ('D001', '2025-12-18 19:15:00', '2025-12-18 20:00:00', 20,
90, 36.7),
```

```
     ('D001', '2025-12-19 18:50:00', '2025-12-19 19:35:00', 22,
92, 36.6),
     ('D001', '2025-12-20 19:05:00', '2025-12-20 19:50:00', 23,
94, 36.8),
     ('D001', '2025-12-21 19:30:00', '2025-12-21 20:15:00', 24,
95, 37.0),
     ('D001', '2025-12-22 19:10:00', '2025-12-22 19:55:00', 25,
96, 36.9),
     ('D001', '2025-12-23 19:00:00', '2025-12-23 19:45:00', 27,
98, 37.1);

     -- iPhone 12 Pro (2815mAh): Cepat penuh karena baterai
kecil
     -- Durasi: ~30 menit untuk naik ~60-70%
     INSERT INTO charging_logs (device_id, plug_in_time,
plug_out_time, start_level, end_level, charge_temp)
     VALUES
     ('D003', '2025-12-17 20:00:00', '2025-12-17 20:30:00', 30,
90, 35.8),
     ('D003', '2025-12-18 20:10:00', '2025-12-18 20:40:00', 32,
91, 35.9),
     ('D003', '2025-12-19 20:30:00', '2025-12-19 21:00:00', 33,
93, 35.7),
     ('D003', '2025-12-20 20:05:00', '2025-12-20 20:35:00', 34,
94, 35.8),
     ('D003', '2025-12-21 20:45:00', '2025-12-21 21:15:00', 35,
95, 35.9),
     ('D003', '2025-12-22 20:15:00', '2025-12-22 20:45:00', 37,
97, 36.1),
     ('D003', '2025-12-23 20:00:00', '2025-12-23 20:30:00', 38,
98, 36.2);
```

init.sql

```
-- Buat database untuk Airflow
-- CREATE DATABASE airflow; -- Already created by POSTGRES_DB
-- Buat Database tambahan (Jika script docker-compose env tidak jalan di OS tertentu)
CREATE DATABASE battery_db;

-- Pindah ke database battery_db
\c battery_db;

-- Buat Skema
CREATE SCHEMA staging;
CREATE SCHEMA datawarehouse;

-- Tabel awal untuk charging logs
CREATE TABLE staging.stg_charging_logs (
    charge_id SERIAL PRIMARY KEY,
    device_id VARCHAR(50),
    plug_in_time TIMESTAMP,
    plug_out_time TIMESTAMP,
    start_level INT,
    end_level INT,
    charge_temp DECIMAL(4,1)
);

-- Tabel sumber untuk charging logs di public schema
CREATE TABLE public.charging_logs (
    charge_id SERIAL PRIMARY KEY,
    device_id VARCHAR(50),
    plug_in_time TIMESTAMP,
    plug_out_time TIMESTAMP,
    start_level INT,
    end_level INT,
    charge_temp DECIMAL(4,1)
);
```

Streamlit

App.py

```
import streamlit as st
import pandas as pd
import boto3
import io
import os
import requests
from requests.auth import HTTPBasicAuth

# Configuration
MINIO_ENDPOINT          =          os.getenv("MINIO_ENDPOINT",
"http://localhost:9000")
MINIO_ACCESS_KEY = os.getenv("MINIO_ACCESS_KEY", "minioadmin")
MINIO_SECRET_KEY = os.getenv("MINIO_SECRET_KEY", "minioadmin")
BUCKET_NAME = os.getenv("BUCKET_NAME", "datalake")
AIRFLOW_API_URL = os.getenv("AIRFLOW_API_URL", "http://airflow-
webserver:8080/api/v1")
AIRFLOW_USER = "admin"
AIRFLOW_PASS = "admin"

st.set_page_config(page_title="Battery   Insight   Dashboard",
layout="wide")

st.title("Battery Insight & Recommendation")
st.markdown("Dashboard analisis penggunaan baterai mingguan dan
rekomendasi preskriptif (Data Source: Minio Gold Layer).")

def get_minio_client():
    return boto3.client(
        "s3",
        endpoint_url=MINIO_ENDPOINT,
        aws_access_key_id=MINIO_ACCESS_KEY,
        aws_secret_access_key=MINIO_SECRET_KEY,
    )



@st.cache_data(ttl=60)
def load_data(refresh_key=0):
    s3 = get_minio_client()
    try:
        obj         =         s3.get_object(Bucket=BUCKET_NAME,
Key="gold/recommendations/final_recommendation.csv")
        df = pd.read_csv(io.BytesIO(obj['Body'].read()))
        # Check if it's the correct version
        if 'outdoor_temp' not in df.columns:
```

```
            return "OLD_VERSION"
        return df
    except Exception as e:
        return None

if 'refresh_count' not in st.session_state:
    st.session_state.refresh_count = 0

df_all = load_data(st.session_state.refresh_count)


if df_all is None:
    st.error("Data Gold belum tersedia di Minio. Silakan klik
Refresh.")
elif isinstance(df_all, str) and df_all == "OLD_VERSION":
    st.warning("Data di Minio masih versi lama. Sedang menunggu
Airflow selesai memproses...")
    st.info("Mohon tunggu 10-20 detik lalu klik Refresh lagi.")
else:
    # Sidebar Selection
    device_options            =            df_all[['device_id',
'model']].drop_duplicates()
    device_labels = {row['device_id']: f"{row['device_id']} -
{row['model']}" for _, row in device_options.iterrows()}

    selected_device_id = st.sidebar.selectbox("Pilih Perangkat",
options=list(device_labels.keys()),    format_func=lambda    x:
device_labels[x], key="device_selector")

    df         =         df_all[df_all['device_id']         ==
selected_device_id].copy()
    if 'date' in df.columns:
        df['date'] = pd.to_datetime(df['date'])
        df = df.sort_values('date')

    model_name = df['model'].iloc[0] if 'model' in df.columns
else "Unknown Model"
    st.header(f"Analisis: {model_name}")

    col1, col2, col3, col4 = st.columns(4)

    avg_usage      =      df['avg_weekly_usage'].iloc[0]      if
'avg_weekly_usage' in df.columns else 0
    best_time     =     df['best_time_to_charge'].iloc[0]     if
'best_time_to_charge' in df.columns else "N/A"
    temp_status = df['temp_status'].iloc[0] if 'temp_status' in
df.columns else "N/A"
```

```
    est_charge   =   df['charge_estimation_20_80'].iloc[0]   if
'charge_estimation_20_80' in df.columns else "N/A"

    col1.metric("Rata-rata   Penggunaan",   f"{avg_usage:.1f}%
/hari")
    col2.metric("Waktu Charge Terbaik", best_time)

    # Menampilkan Suhu Banjarmasin di metrik utama (Murni dari
API)
    outdoor_temp = df['outdoor_temp'].iloc[0] if 'outdoor_temp'
in df.columns else "N/A"
    location   =   df['location'].iloc[0]   if   'location'   in
df.columns else "Banjarmasin"
    col3.metric(f"Suhu {location}", outdoor_temp)

    col4.metric("Estimasi Charge (20-80%)", est_charge)

    st.subheader("Rekomendasi Preskriptif")
    rec_text = df['recommendation'].iloc[0] if 'recommendation'
in df.columns else "N/A"
    st.info(rec_text)

    st.subheader(f"Tren    Penggunaan    Baterai    Harian    -
{model_name}")
    if 'date' in df.columns and 'daily_usage' in df.columns:
        chart_df = df.copy()
        chart_df['date_label']                                  =
chart_df['date'].dt.strftime('%d %b')

st.bar_chart(chart_df.set_index('date_label')['daily_usage'])

        st.write("**Riwayat Penggunaan Per Hari:**")
        summary_df = df[['date', 'daily_usage']].copy()
        summary_df['date']                                      =
summary_df['date'].dt.strftime('%A, %d %B %Y')
        summary_df.columns  =  ['Tanggal',  'Penggunaan  Baterai
(%)']
        st.table(summary_df)
```

docker-compose.yml

```
version: '3.8'
services:
  postgres:
    image: postgres:13
    environment:
      - POSTGRES_USER=airflow
```

```
      - POSTGRES_PASSWORD=airflow
      - POSTGRES_DB=airflow
    ports:
      - "5433:5432"
    volumes:
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U airflow"]
      interval: 5s
      timeout: 5s
      retries: 5

  minio:
    image: minio/minio
    environment:
      - MINIO_ROOT_USER=minioadmin
      - MINIO_ROOT_PASSWORD=minioadmin
    ports:
      - "9000:9000"
      - "9001:9001"
    command: server /data --console-address ":9001"
    healthcheck:
                      test:      ["CMD",      "curl",      "-f",
"http://localhost:9000/minio/health/live"]
      interval: 5s
      timeout: 5s
      retries: 5

  airflow-webserver:
    image: apache/airflow:2.7.1
    depends_on:
      postgres:
        condition: service_healthy
      airflow-init:
        condition: service_completed_successfully
    environment:
      - LOAD_EX=n
      - EXECUTOR=LocalExecutor
                                                                -
AIRFLOW__DATABASE__SQL_ALCHEMY_CONN=postgresql+psycopg2://airfl
ow:airflow@postgres/airflow
                                                                -
AIRFLOW__CORE__FERNET_KEY=fb391195ca1e472d4c72d4c72d4c72d4c72d4
c72d4c=
      - AIRFLOW__CORE__LOAD_EXAMPLES=False
        -  _PIP_ADDITIONAL_DEPENDENCIES=boto3  pandas  requests
apache-airflow-providers-postgres
    restart: always
```

```
    ports:
      - "8080:8080"
    volumes:
      - ./dags:/opt/airflow/dags
      - ./logs:/opt/airflow/logs
      - ./plugins:/opt/airflow/plugins
      - ./data_source:/opt/airflow/data_source
      - ./data_charging.sql:/opt/airflow/data_charging.sql
      - ./API.txt:/opt/airflow/API.txt
    command: webserver

  airflow-scheduler:
    image: apache/airflow:2.7.1
    depends_on:
      postgres:
        condition: service_healthy
      airflow-init:
        condition: service_completed_successfully
    environment:
      - LOAD_EX=n
      - EXECUTOR=LocalExecutor
                                                                    -
AIRFLOW__DATABASE__SQL_ALCHEMY_CONN=postgresql+psycopg2://airfl
ow:airflow@postgres/airflow
                                                                    -
AIRFLOW__CORE__FERNET_KEY=fb391195ca1e472d4c72d4c72d4c72d4c72d4
c72d4c=
      - AIRFLOW__CORE__LOAD_EXAMPLES=False
      - _PIP_ADDITIONAL_DEPENDENCIES=boto3 pandas requests
    restart: always
    volumes:
      - ./dags:/opt/airflow/dags
      - ./logs:/opt/airflow/logs
      - ./plugins:/opt/airflow/plugins
      - ./data_source:/opt/airflow/data_source
      - ./data_charging.sql:/opt/airflow/data_charging.sql
      - ./API.txt:/opt/airflow/API.txt
    command: scheduler

  airflow-init:
    image: apache/airflow:2.7.1
    depends_on:
      postgres:
        condition: service_healthy
    environment:
                                                                    -
AIRFLOW__DATABASE__SQL_ALCHEMY_CONN=postgresql+psycopg2://airfl
ow:airflow@postgres/airflow
```

```
    command: >
      bash -c "airflow db init &&
      airflow users create --username admin --firstname admin --lastname admin --role Admin --email admin@example.com --password admin"

  streamlit:
    build:
      context: .
      dockerfile: Dockerfile.streamlit
    depends_on:
      minio:
        condition: service_healthy
    volumes:
      - ./app.py:/app/app.py
    ports:
      - "8501:8501"
    environment:
      - MINIO_ENDPOINT=http://minio:9000
      - MINIO_ACCESS_KEY=minioadmin
      - MINIO_SECRET_KEY=minioadmin
      - BUCKET_NAME=datalake
      - AIRFLOW_API_URL=http://airflow-webserver:8080/api/v1
```

Docker.airflow

```
ARG AIRFLOW_VERSION=2.7.1
ARG PYTHON_VERSION=3.9
FROM apache/airflow:${AIRFLOW_VERSION}-python${PYTHON_VERSION}

# Install requests library and other DAG dependencies
RUN pip install requests pandas numpy psycopg2-binary boto3
```

LINK GitHub
https://github.com/dheaaprilinda01/uas-kecerdasan-bisnis.git