BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Selama pelaksanaan magang di PT Entrefine Data Indonesia, posisi yang dijalankan adalah *Data Engineer Intern* dengan fokus utama pada pengelolaan data yang diterima dari klien. *Intern Data Engineer* berada di bawah pengawasan langsung oleh supervisor *Developer IT* dan bertanggung jawab untuk memastikan bahwa setiap proses pengolahan data berjalan sesuai standar dan kebutuhan klien, strukturnya seperti yang sudah ditunjukkan pada Gambar 2.2. Hal yang dilakukan adalah mengerjakan alur dari cara penginputan data mentah klien, memasukkan data klien ke *data warehourse* Google Bigquery, pemrosesan data dari data mentah dan ditransformasi menggunakan DBT *(*Data Build Tool) sehingga menjadi data yang terstruktur dan siap digunakan untuk di *consume* BI *tools*.

Proses pembelajaran di lingkungan kerja dilakukan melalui pendekatan praktik langsung di lapangan, di mana pemahaman alur kerja dilakukan melalui sesi *briefing* dan bimbingan teknis singkat. Apabila terdapat kendala selama proses pengerjaan, supervisor turut memberikan pendampingan dan solusi agar kegiatan pengolahan data tetap berjalan efektif dan sesuai dengan target yang telah ditetapkan. Pelaporan aktivitas kerja dilaksanakan secara sistematis melalui platform Trello yang berfungsi sebagai media pencatatan tugas harian bagi seluruh anggota tim. Setiap hasil pekerjaan yang telah diselesaikan akan diperiksa oleh *Project Manager* untuk memastikan kesesuaian dengan kebutuhan proyek. Selain itu, dilakukan diskusi rutin pada awal dan akhir hari kerja untuk membahas perkembangan tugas, hambatan yang dihadapi, serta rencana pekerjaan berikutnya. Proses pelaporan ini tidak memerlukan presentasi formal, melainkan dilakukan secara interaktif melalui pembaruan status tugas dan komunikasi singkat.

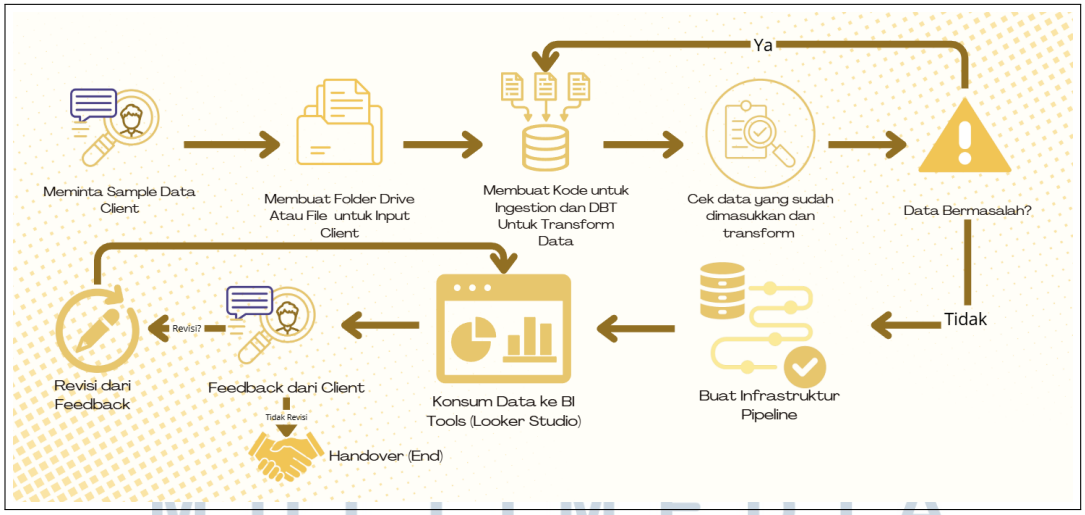
3.2 Tugas yang Dilakukan

Selama masa pelaksanaan magang di PT Entrefine Data Indonesia, tugas utama yang dilaksanakan berfokus pada kegiatan pengelolaan dan pemrosesan data klien. Peran sebagai *Data Engineer Intern* mencakup beberapa tahapan pekerjaan mulai dari proses penerimaan data mentah, pembersihan data, transformasi, hingga penyimpanan data ke dalam *data warehouse* berbasis *Google BigQuery*. Data yang

diterima dari klien umumnya masih dalam bentuk mentah dan tidak terstruktur, sehingga diperlukan proses penyesuaian format serta validasi agar dapat digunakan pada tahap analisis selanjutnya.

Proses pengolahan data dilakukan dengan membangun dan mengimplementasikan *data pipeline* otomatis menggunakan layanan Google Cloud Platform (GCP). *Pipeline* ini dirancang untuk mengatur alur proses mulai dari ekstraksi data, transformasi, hingga pemuatan (ETL *process*). Dalam pelaksanaannya, Data Build Tool (DBT) digunakan untuk melakukan transformasi data di dalam BigQuery, sehingga data mentah dapat diolah menjadi tabel-tabel terstruktur sesuai kebutuhan proyek.

Untuk mendukung otomatisasi, proses orkestrasi *pipeline* memanfaatkan Google Workflow sebagai pengendali utama yang mengatur urutan eksekusi setiap komponen *pipeline*. Eksekusi tugas-tugas pemrosesan dilakukan melalui Google Cloud Run Job, yang menjalankan skrip transformasi atau pemrosesan data dalam lingkungan terisolasi dan efisien. Selain itu, Cloud Scheduler digunakan sebagai pemicu otomatis untuk menjalankan Google Workflow sesuai dengan jadwal yang telah ditentukan, sehingga proses pengolahan data dapat berjalan secara terjadwal tanpa intervensi manual. Untuk lebih ringkasnya pekerjaan yang dilakukan dapat dilihat pada Gambar 3.1



Gambar 3.1. Alur Pekerjaan

Selain bertanggung jawab pada proses teknis, kegiatan magang juga melibatkan koordinasi harian melalui platform Trello yang digunakan untuk mencatat *daily task* dan memantau perkembangan pekerjaan. Setiap tugas yang telah diselesaikan diverifikasi oleh *Supervisor Developer IT* dan *Project Manager*

untuk evaluasi akhir. Diskusi harian dilakukan pada awal dan akhir jam kerja untuk membahas progres, hambatan teknis, serta langkah-langkah perbaikan apabila ditemukan kendala dalam proses pengolahan data. Dengan sistem kerja tersebut, setiap tahapan pengerjaan dapat terpantau dengan baik dan hasil akhir yang dihasilkan sesuai dengan standar kualitas yang diterapkan perusahaan.

3.3 Uraian Pelaksanaan Magang

Kegiatan magang dilaksanakan dengan fokus pada pengelolaan data, pengembangan sistem, serta penerapan teknologi berbasis *cloud*. Setiap minggu memiliki tujuan dan tanggung jawab yang berbeda, menyesuaikan dengan kebutuhan proyek yang sedang berjalan. Secara umum, kegiatan meliputi proses pembelajaran penggunaan berbagai *tools*, keterlibatan dalam penyelesaian masalah teknis, hingga pengembangan dan otomatisasi sistem data. Rincian kegiatan mingguan selama masa magang disajikan pada Tabel 3.1.

Tabel 3.1. Uraian kegiatan setiap minggu selama pelaksanaan kerja magang

Minggu Ke-	Pekerjaan yang Dilakukan
1	Mempelajari dan memahami berbagai *tools* yang digunakan dalam *project*, seperti Google Cloud Platform (BigQuery, Cloud Run Job, Google Workflow, dan Scheduler), DBT, Docker, serta beberapa teknologi pendukung lainnya.
2	Mulai melakukan praktik langsung (*hands-on*) dengan membantu menyelesaikan permasalahan sederhana, seperti mengganti *data source* pada Looker serta menyalin dan menyesuaikan kode fungsi yang telah ada untuk digunakan pada *brand* lain dalam *project* yang sama.
3	Melanjutkan kegiatan pada minggu sebelumnya, yaitu membantu penyelesaian permasalahan teknis kecil serta memahami alur kerja *project* secara lebih mendalam.
4	Berpartisipasi dalam pembuatan *project* dashboard dari awal untuk salah satu *client*, mencakup tahap perencanaan, implementasi, dan integrasi data.
Bersambung ke halaman berikutnya	

Tabel 3.1 Uraian kegiatan setiap minggu selama pelaksanaan kerja magang (Lanjutan)

Minggu Ke-	Pekerjaan yang Dilakukan
5	Melakukan perbaikan (*fixing*) terhadap *project* yang sedang berjalan serta membantu pada *project-project* baru yang mulai dikembangkan.
6	Melanjutkan kegiatan perbaikan dan pengembangan *project* yang telah dilakukan pada minggu sebelumnya.
7	Melanjutkan kegiatan minggu sebelumnya, dengan fokus pada penyempurnaan dan stabilisasi sistem yang sedang dikerjakan.
8	Mendapat kepercayaan untuk menangani satu *project client* baru secara mandiri, meliputi pembuatan struktur folder data untuk penyimpanan data klien, melakukan *setup* awal *project*, serta mulai mengerjakan proses integrasi data.
9	Mengembangkan kode Python untuk proses *ingest* data mentah (*raw data*) ke dalam BigQuery sebagai bagian dari *pipeline* data.
10	Membuat dan melakukan *setup project* DBT (Data Build Tool) untuk pengelolaan transformasi data yang lebih terstruktur.
11	Mengembangkan *jobs* untuk menjalankan berbagai tugas otomatis, seperti proses *ingest data* dari Google Spreadsheets dan Google Drive, serta menjalankan *workflow* DBT.
12	Membuat GitHub Action untuk otomatisasi *deployment* kode ke *cloud*, serta melakukan *setup workflow* dan Scheduler guna men-*trigger* pekerjaan (*jobs*) dan *workflow* pada waktu tertentu.
13	Memantau dan melakukan *maintenance* dari *pipeline* yang telah dibuat.
14 – 17	Melakukan *maintenance* pada *project* Skin Game, Cartiera, dan Sukses Group.
18	Melakukan *kickoff project* Social Barn serta mempelajari data yang diberikan oleh pihak Social Barn.
19	Memulai *setup project* dan konfigurasi untuk *project* Social Barn.
	Bersambung ke halaman berikutnya

Tabel 3.1 Uraian kegiatan setiap minggu selama pelaksanaan kerja magang (Lanjutan)

Minggu Ke-	Pekerjaan yang Dilakukan
20 – 23	Membuat dashboard serta melakukan *adjustment* pada *project* Social Barn.

3.3.1 User Requirement

User requirement digunakan sebagai panduan utama dalam pengembangan sistem *pipeline data* Cartiera agar sesuai dengan kebutuhan pengguna bisnis.

1. Data laporan harus selalu ter*update* secara berkala sesuai jadwal yang disepakati, sehingga pengguna dapat mengambil keputusan berdasarkan informasi terbaru.
2. Pengguna dapat melihat informasi yang sudah digabung dari berbagai sumber (misalnya penjualan online, stok gudang, dan hasil stock opname) dalam satu tampilan yang konsisten.
3. Informasi yang ditampilkan harus akurat dan konsisten, sehingga angka di laporan tidak membingungkan dan dapat dipercaya untuk pengambilan keputusan.
4. Data tidak boleh terduplikasi ketika proses *ingestion data*

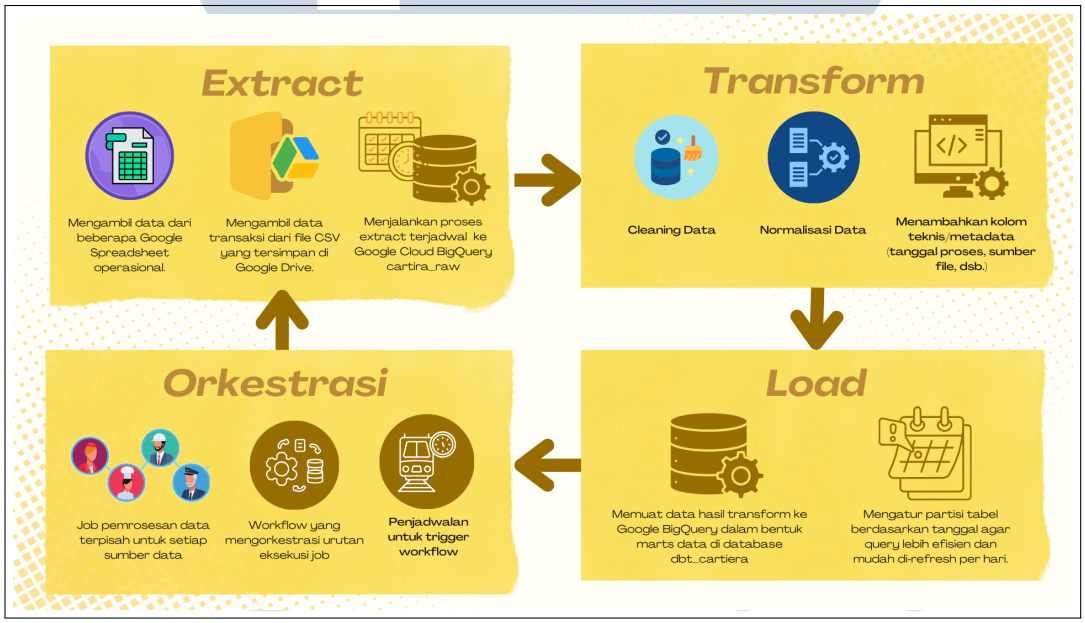
3.3.2 Perancangan Pipeline Otomatis Cartiera

Entrefine sebagai vendor penyedia layanan pengolahan data menjalin kerja sama dengan Cartiera, sebuah merek fashion aksesori pria yang dikenal dengan gaya simple dan elegan. Produk yang dipasarkan mencakup dompet, tas, dan berbagai aksesori lainnya [9]. Kerja sama ini berfokus pada penguatan sistem informasi internal yang mendukung proses operasional dan analisis penjualan.

Proyek Cartiera resmi dimulai pada 2 September 2025 dengan cakupan utama berupa pengembangan dashboard tren penjualan serta sistem pengelolaan input untuk stok dan pencatatan retur. Ruang lingkup pengelolaan stok mencakup proses mulai dari input rencana barang masuk, barang masuk, barang keluar, barang *defect*, hasil perbaikan gudang *reject*, hingga monitoring keseluruhan pergerakan

barang. Sistem ini dirancang untuk meningkatkan visibilitas dan akurasi data inventori pada brand Cartiera.

Berdasarkan hasil diskusi bersama pihak Cartiera, diputuskan bahwa proses input UPFOS dan *stock opname* dilakukan melalui file berformat .csv yang disimpan pada Google Drive. Untuk mendukung alur kerja yang lebih terstruktur dan *cost efficient*, terstandarisasi, dan otomatis, direkomendasikan penggunaan lingkungan Google Cloud Platform. Rangkaian layanan yang digunakan meliputi Google Cloud Run Jobs yang menjalankan kode Python untuk proses ingestion data melalui container berbasis Docker, Google Workflows sebagai orkestrator antarproses, serta Google Cloud Scheduler sebagai pemicu otomatisasi dalam menjalankan workflow yang telah ditetapkan. Struktur alur proses secara umum digambarkan melalui tahapan *extract*, *transform*, *load*, serta orkestrasi *data pipeline* untuk dikonsumsi Looker Studio seperti yang terlihat pada Gambar 3.2.



Gambar 3.2. Alur input Cartiera

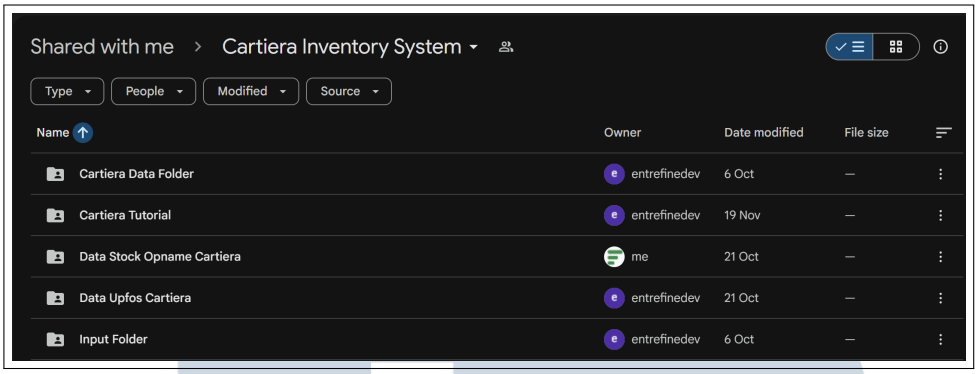
Konsep *Extract, Transform, Load* (ETL) merupakan pendekatan umum yang digunakan dalam pengelolaan data untuk mengintegrasikan data dari berbagai sumber ke dalam satu sistem terpusat [10]. Tahap *extract* berfokus pada proses pengambilan data mentah dari beragam sistem sumber, baik yang bersifat terstruktur maupun semi-terstruktur. Selanjutnya, tahap *transform* bertujuan untuk mengolah data hasil ekstraksi melalui proses pembersihan, penyesuaian format, serta penyesuaian skema agar data sesuai dengan kebutuhan bisnis dan analisis. Tahap terakhir, yaitu *load*, merupakan proses pemuatan data yang telah

ditransformasi ke dalam sistem penyimpanan akhir, seperti *data warehouse* atau *data mart*, sehingga data tersebut siap digunakan oleh sistem analitik dan *Business Intelligence*.

Pengerjaan *pipeline* input untuk Cartiera terbagi ke dalam tiga tahapan utama dengan penambahan proses orkestrasi, yaitu proses ekstraksi (*Extract*), transformasi (*Transform*), dan pemuatan (*Load*) dan juga proses orkestrasi ketiga tahapan sebelumnya. Pada tahap ekstraksi, data mentah diambil dari berbagai sistem sumber, seperti Google Drive dan Google Spreadsheet. Data yang telah diekstraksi kemudian diproses pada tahap transformasi, di mana data dibersihkan dan disesuaikan yang merepresentasikan kebutuhan data operasional Cartiera. Selanjutnya, data hasil transformasi tersebut dimuat ke dalam penyimpanan akhir sebagai bagian dari proses dalam bentuk *marts-marts* yang merupakan bagian dari tahapan *load*. Data yang telah siap selanjutnya dapat dikonsumsi oleh sistem *Business Intelligence* (*BI*), seperti Looker Studio, untuk keperluan analisis dan pelaporan yang dilakukan pada tahap terpisah. Setelah semuanya sudah berjalan, selanjutnya adalah mengorkestrasi semua tahapan sebelumnya untuk berjalan secara otomatis. Keempat tahapan tersebut sudah digambarkan pada Gambar 3.2 dan akan dijelaskan lebih lanjut pada subbab berikut agar alur pemrosesan data dapat dipahami secara lebih rinci.

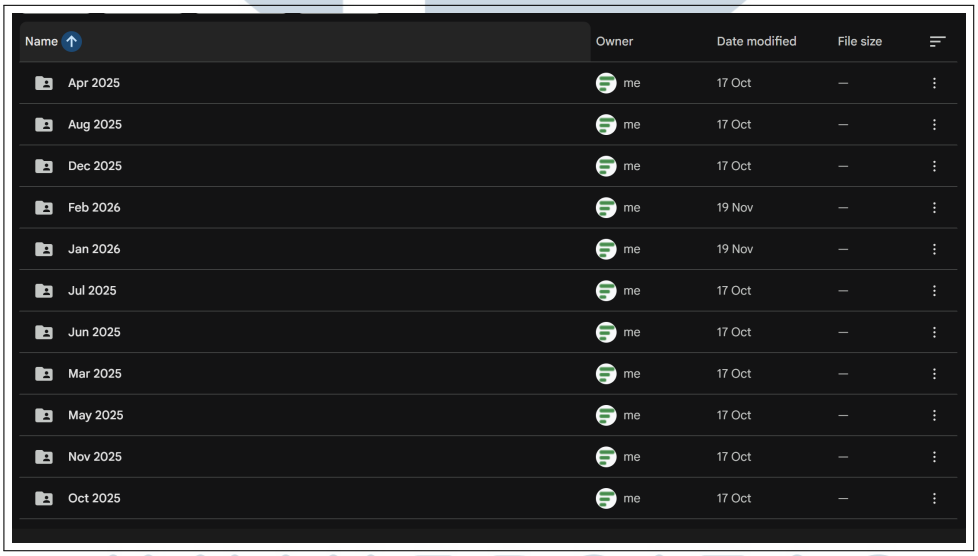
A Extract

Pada tahap ini, pihak Cartiera melakukan proses input data operasional dengan mengunggah berkas berformat .csv ke Google Drive untuk UPFOS dan *stock opname* serta memasukkan data langsung ke Google Spreadsheet untuk input retur, *complain*, dan sebagainya yang telah disediakan oleh Entrefine. Seluruh berkas dan Spreadsheet tersebut dikelola secara terpusat dalam satu folder master bernama "*Cartiera Inventory System*", sehingga memudahkan proses pengorganisasian dan pemantauan data sebelum memasuki tahap berikutnya dalam *pipeline*. Struktur penyimpanan ini dapat dilihat pada gambar 3.3.



Gambar 3.3. Google Drive Master

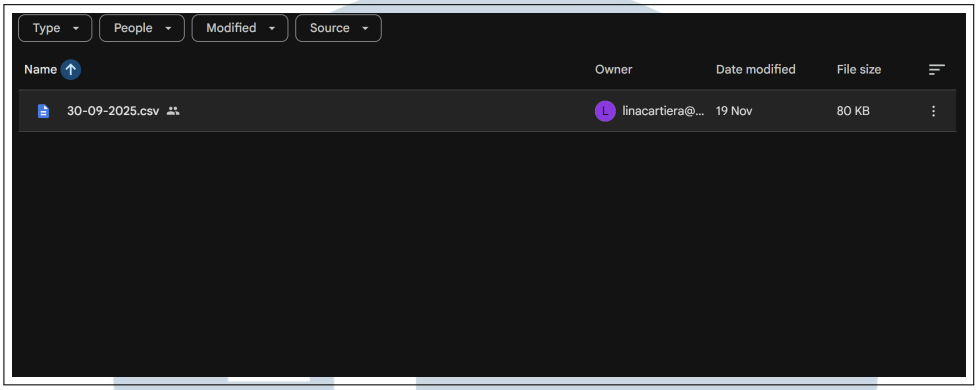
Untuk mengunggah file data UPFOS dan *stock opname*, pengguna terlebih dahulu membuka folder yang sesuai dengan kategori data pada folder master. Setiap jenis data telah disediakan struktur penyimpanan berbasis waktu dengan format folder MMM/YYYY (misalnya Sep 2025), sehingga proses pengarsipan menjadi lebih teratur dan dapat dideteksi oleh kode Python nantinya. Tampilan struktur folder tersebut ditunjukkan pada gambar 3.4.



Gambar 3.4. Contoh Folder Bulan Untuk *Stock Opname* dan UPFOS

Setelah memilih folder berdasarkan bulan yang relevan, pengguna dapat mengunggah file data dengan mengikuti format penamaan yang telah distandarkan, yaitu DD-MM-YYYY*.csv (misalnya 30-09-2025.csv). Penerapan format penamaan ini tidak hanya memastikan keteraturan dan mempermudah identifikasi file berdasarkan tanggal input, tetapi juga menjadi bagian penting dalam proses *Extract, Transform, Load* (ETL) yang telah dibangun. Format nama file tersebut dimanfaatkan dalam *pipeline* otomatis untuk memfilter, membaca, dan

mengekstraksi data sesuai tanggal. Contoh format nama file tersebut dapat dilihat pada gambar 3.5.



Gambar 3.5. Contoh Memasukkan Data UPFOS dan *Stock Opname*

Setelah data diunggah oleh pihak Cartiera ke Google Drive dengan struktur dan format penamaan yang telah distandarisasi, tahap selanjutnya adalah proses *ingestion* data ke dalam sistem penyimpanan terpusat berbasis BigQuery. Proses ini dijalankan menggunakan serangkaian skrip Python yang berfungsi untuk membaca file dari Google Drive, melakukan pembersihan data (*data cleaning*), transformasi format, serta memuat hasilnya ke BigQuery dalam bentuk tabel terpartisi.

Langkah pertama dalam *pipeline* adalah inisialisasi koneksi menggunakan *Application Default Credentials (ADC)*, yang memberikan akses hanya-baca ke Google Drive dan akses penuh ke layanan BigQuery. Skrip kemudian membangun dua klien terpisah, yaitu satu klien untuk berkomunikasi dengan Google Drive API dan satu klien untuk menjalankan operasi pada BigQuery. Pada tahap ini, konfigurasi seperti lokasi dataset, ID folder pada Google Drive, serta jumlah hari *lookback* didefinisikan agar proses *ingest* dapat berjalan secara otomatis tanpa konfigurasi ulang. Potongan kode dapat terlihat pada kode 3.1.

```
from google.auth import default
from google.cloud import bigquery
from googleapiclient.discovery import build

# Load credentials
credentials, project = default(
    scopes=[
        'https://www.googleapis.com/auth/drive.readonly',
        'https://www.googleapis.com/auth/cloud-platform'
    ]
)

```

```
# BigQuery & Drive client
bq_client = bigquery.Client(credentials=credentials, project=
    project)
drive_service = build('drive', 'v3', credentials=credentials)
```

Kode 3.1: Inisiasi Kredensial dan Membuat klien BigQuery dan Google Drive

Pipeline kemudian memulai proses dengan mengidentifikasi seluruh folder bulanan yang ada di dalam direktori *root* SO pada Google Drive. Struktur folder bulan, misalnya "Oct 2025" atau "Nov 2025", dipindai menggunakan pencocokan pola (*regex*) untuk memastikan hanya folder dengan format waktu yang valid yang diproses. Dari setiap folder bulan tersebut, skrip akan mencari file-file .csv yang namanya mengikuti format tanggal DD-MM-YYYY, sehingga hanya file hasil ekspor UPFOS yang memenuhi standar penamaan yang akan diproses. Untuk mendapatkan nama bulan folder dapat menggunakan kode yang ditunjukkan pada kode 3.2.

```
def get_month_folders(parent_folder_id: str):
    """
    Get all month folders (format: Oct 2025, Nov 2025, etc.) from
    parent folder.
    Returns list of folder IDs and names.
    """
    query = (
        f"'{parent_folder_id}' in parents and "
        "mimeType='application/vnd.google-apps.folder' and trashed
    =false"
    )

    try:
        results = drive_service.files().list(
            q=query,
            fields='files(id, name)',
            orderBy='name desc'
        ).execute()
    except Exception:
        return []

    folders = results.get('files', [])

    # Filter folders that match month-year pattern (e.g., "Oct
    2025", "Nov 2025")
    month_folders = []
    month_pattern = re.compile(
```

```
        r'^(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)\s+\d{4}$',
        re.IGNORECASE
    )

    for folder in folders:
        if month_pattern.match(folder['name']):
            month_folders.append(folder)

    return month_folders
```

Kode 3.2: Mendapatkan Folder Nama Bulan

Setelah file ditemukan, proses berlanjut pada pembacaan isi file. Karena file CSV dari sistem operasional sering memiliki inkonsistensi seperti perbedaan *separator* atau *encoding*, fungsi *read csv from drive* dirancang untuk mencoba beberapa strategi *parsing* yang berbeda. Proses ini mencoba kombinasi koma atau titik koma sebagai pemisah kolom serta beberapa *encoding* umum hingga ditemukan konfigurasi yang dapat dibaca dengan benar. Untuk menjaga konsistensi struktur, skrip memastikan bahwa hanya 28 kolom pertama yang digunakan, karena struktur data SO Cartiera secara operasional memang terbatas hingga kolom AB. Fungsi *read csv from drive* dapat dilihat pada kode 3.3.

```
def read_csv_from_drive(file_id: str, file_name: str):
    """
    Read CSV file from Google Drive with fallback strategies for encodings and separators.
    Always limits to max 28 columns (A to AB).
    """
    request = drive_service.files().get_media(fileId=file_id)
    file_content = io.BytesIO()
    downloader = MediaIoBaseDownload(file_content, request)

    done = False
    while not done:
        _, done = downloader.next_chunk()

    file_content.seek(0)

    strategies = [
        {'sep': ',', 'encoding': 'utf-8'},
        {'sep': ';', 'encoding': 'utf-8'},
        {'sep': ',', 'encoding': 'ISO-8859-1'},
        {'sep': ';', 'encoding': 'ISO-8859-1'},
```

```
        {'sep': ',', 'encoding': 'windows-1252'},
        {'sep': ';', 'encoding': 'windows-1252'},
    ]

    for strategy in strategies:
        try:
            file_content.seek(0)
            df = pd.read_csv(
                file_content,
                sep=strategy['sep'],
                encoding=strategy['encoding']
            )

            if len(df.columns) >= 3:
                df = df.iloc[:, :28]   # limit to first 28 columns
                return df

        except Exception:
            continue

    return None
```

Kode 3.3: Membaca CSV dari Google Drive

B Transform

Pada tahap *transform* dalam proses ETL, sebelum data mentah dimuat ke dalam *data warehouse* Google BigQuery, dilakukan serangkaian transformasi untuk menyiapkan data agar siap dianalisis. Tujuan utama dari proses *transform* ini adalah memastikan data telah bersih, terstruktur, dan relevan sesuai dengan kebutuhan analisis, serta disimpan terlebih dahulu ke dalam *data staging*.

Data mentah masuk ke tahap pembersihan dan transformasi. Pada tahap ini, nama kolom diseragamkan melalui proses *renaming*, mulai dari menghilangkan karakter khusus hingga mengonversi format nama agar seluruh kolom menggunakan huruf kecil dan garis bawah. Selain itu, nilai pada kolom teks dibersihkan dari karakter yang tidak diperlukan, dan tipe data pada kolom tanggal dikonversi ke tipe waktu yang sesuai. Jika data tidak memiliki kolom tanggal bawaan, skrip secara otomatis mengekstraknya dari nama file untuk memastikan setiap baris data dapat dipetakan ke partisi tanggal yang tepat di BigQuery. Kode dari tahap ini dapat dilihat pada kode 3.4.

```python
def clean_dataframe(df: pd.DataFrame, file_name: str):
    df = df.rename(columns=COLUMN_MAPPING_LANGUAGE_RENAME)
    df.columns = [str(col).strip('=".,') for col in df.columns]
    df.columns = [str(col).lower().replace(' ', '_') for col in df.columns]
    df = df.rename(columns=COLUMN_MAPPING_RENAME)

    for col in df.columns:
        if df[col].dtype == 'object':
            df[col] = df[col].apply(lambda x: str(x).strip('=".,') if pd.notna(x) else x)
            df[col] = df[col].apply(lambda x: x.strip() if isinstance(x, str) else x)

    if 'creation_time' in df.columns:
        df['creation_time'] = pd.to_datetime(df['creation_time'], errors='coerce')
        df['tanggal'] = df['creation_time'].dt.date
        df['tanggal'] = pd.to_datetime(df['tanggal'])
    else:
        date_string = file_name[:10]
        try:
            df['tanggal'] = pd.to_datetime(date_string, format='%d-%m-%Y')
        except:
            df['tanggal'] = pd.NaT

    if 'stocktaking_time' in df.columns:
        df['stocktaking_time'] = pd.to_datetime(df['stocktaking_time'], errors='coerce')

    return df
```

Kode 3.4: Pembersihan *Header* dan *DataFrame*

Proses *transform* kemudian dilanjutkan dengan validasi tipe data, di mana kolom numerik seperti kuantitas stok, jumlah inventori, dan semua kolom yang bertipe data angka dipastikan tersimpan dalam format angka, sedangkan kolom yang bersifat identitas tetap disimpan sebagai *string* agar format aslinya tidak berubah. Langkah ini penting untuk menjaga akurasi perhitungan serta konsistensi data saat proses analisis dilakukan. Setelah seluruh proses pembersihan dan penyesuaian data selesai, skrip menambahkan *metadata* tambahan berupa nama file sumber, folder asal, serta *timestamp ingestion*. Penambahan informasi ini bertujuan

untuk menjaga keterlacakan data dan memudahkan proses audit di kemudian hari. Keseluruhan proses *transform* ini diimplementasikan dalam kode yang ditunjukkan pada kode 3.5.

```
def numeric_transformation(df: pd.DataFrame):
    qty_columns = ['stocktaking_qty', 'inventory_qty', 'difference']
    for col in qty_columns:
        if col in df.columns:
            df[col] = pd.to_numeric(df[col], errors='coerce')

    id_columns = ['stocktake_order_no', 'product_code', 'product_specification_code']
    for col in id_columns:
        if col in df.columns:
            df[col] = df[col].astype(str)

    return df

def process_so_data(days_string: str = None):
    all_dataframes = []
    month_folders = get_month_folders(SO_FOLDER_ID)

    if not month_folders:
        return pd.DataFrame()

    for folder in month_folders:
        folder_id = folder['id']
        folder_name = folder['name']
        csv_files = get_csv_files_from_folder(folder_id, days_string)

        if not csv_files:
            continue

        for file in csv_files:
            file_id = file['id']
            file_name = file['name']

            try:
                df = read_csv_from_drive(file_id, file_name)

                if df is None or df.empty:
```

```
                    continue

                df = clean_dataframe(df, file_name)
                df = numeric_transformation(df)
                df['source_file'] = file_name
                df['source_folder'] = folder_name
                df['ingestion_timestamp'] = datetime.now()
                all_dataframes.append(df)

            except:
                continue

    if all_dataframes:
        final_df = pd.concat(all_dataframes, ignore_index=True)
        return final_df
    else:
        return pd.DataFrame()
```

Kode 3.5: Proses SO DataFrame

Setelah semua file dalam periode waktu tertentu selesai diolah, seluruh DataFrame yang berhasil dibaca digabungkan menjadi satu kumpulan data final. *Pipeline* kemudian memverifikasi keberadaan tabel target di BigQuery. Jika tabel belum tersedia, sistem akan membuat tabel baru dengan skema yang telah ditentukan dan partisi berdasarkan kolom tanggal. Sebelum data ditulis ke BigQuery, *pipeline* juga menghapus terlebih dahulu partisi yang memiliki tanggal sama, sehingga data yang diunggah selalu dalam keadaan bersih dan tidak terjadi duplikasi. Pembuatan partisi dan penghapusan partisi terlihat pada kode 3.6

```
def create_partitioned_table_if_not_exists(table_name: str):
    project_id = bq_client.project
    table_id = f"{project_id}.{DATASET_ID}.{table_name}"

    try:
        bq_client.get_table(table_id)
        return
    except NotFound:
        schema = [
            bigquery.SchemaField("creator", "STRING"),
            bigquery.SchemaField("creation_time", "TIMESTAMP"),
            bigquery.SchemaField("operator", "STRING"),
            bigquery.SchemaField("stocktaking_time", "TIMESTAMP"),
            bigquery.SchemaField("stocktake_order_no", "STRING"),
            bigquery.SchemaField("stocktake_type", "STRING"),
```

```
            bigquery.SchemaField("warehouse", "STRING"),
            bigquery.SchemaField("products_name", "STRING"),
            bigquery.SchemaField("product_code", "STRING"),
            bigquery.SchemaField("sku_name", "STRING"),
            bigquery.SchemaField("product_specification_code", "
    STRING"),
            bigquery.SchemaField("stocktaking_qty", "FLOAT"),
            bigquery.SchemaField("inventory_qty", "FLOAT"),
            bigquery.SchemaField("difference", "FLOAT"),
            bigquery.SchemaField("notes", "STRING"),
            bigquery.SchemaField("tanggal", "DATE"),
        ]

        table = bigquery.Table(table_id, schema=schema)
        table.time_partitioning = bigquery.TimePartitioning(
            type_=bigquery.TimePartitioningType.DAY,
            field="tanggal",
        )

        table = bq_client.create_table(table)


def delete_partition_data(table_name: str, dates: list):
    project_id = bq_client.project
    table_id = f"{project_id}.{DATASET_ID}.{table_name}"

    for date in dates:
        date_str = date.strftime('%Y-%m-%d')
        query = f"""
        DELETE FROM `{table_id}`
        WHERE tanggal = DATE '{date_str}'
        """

        try:
            query_job = bq_client.query(query)
            query_job.result()
        except:
            pass
```

Kode 3.6: Membuat Tabel Partisi dan Menghapus Partisi

Tahap terakhir dari transformasi ini adalah memuat data ke BigQuery menggunakan mode *append* setelah proses pembersihan partisi. Seluruh baris dengan tanggal yang valid akan dimasukkan kembali ke tabel terpartisi, dan jika

seluruh proses berjalan tanpa kendala, sistem akan menampilkan ringkasan jumlah baris yang berhasil diunggah serta partisi yang diperbarui. Dengan pendekatan ini, tahan terhadap format data yang tidak konsisten, dan memastikan bahwa data stok opname Cartiera tersimpan dalam struktur yang rapi, mudah dicari, dan siap digunakan untuk analisis lebih lanjut 3.7.

```
def upload_to_bigquery(df: pd.DataFrame, table_name: str):
    if df.empty:
        return

    project_id = bq_client.project
    table_id = f"{project_id}.{DATASET_ID}.{table_name}"

    try:
        bq_client.get_dataset(f"{project_id}.{DATASET_ID}")
    except NotFound:
        dataset = bigquery.Dataset(f"{project_id}.{DATASET_ID}")
        dataset.location = "asia-southeast2"
        bq_client.create_dataset(dataset, exists_ok=True)

    create_partitioned_table_if_not_exists(table_name)

    valid_dates = df[df['tanggal'].notna()]['tanggal'].dt.date.
    unique()
    unique_dates = sorted(valid_dates)

    if len(unique_dates) == 0:
        return

    delete_partition_data(table_name, unique_dates)

    try:
        df = df[df['tanggal'].notna()].copy()

        if df.empty:
            return

        metadata_columns = ['source_file', 'source_folder', '
    ingestion_timestamp', 'tanggal_file']
        df = df.drop(columns=[col for col in metadata_columns if
    col in df.columns], errors='ignore')

        df = df.replace([pd.NA, pd.NaT], None)
```

```
35
36          pandas_gbq.to_gbq(
37              df,
38              table_id,
39              project_id=project_id,
40              if_exists='append',
41              location='asia-southeast2',
42              credentials=credentials
43          )
44
45      except Exception as e:
46          raise e
```

Kode 3.7: Mengunggah *Dataframe* ke Google BigQuery

C Load

Data yang sudah masuk ke Bigquery disusun dan dimuat menjadi lima *data marts*, yaitu *marts complain*, *marts estimation in*, *marts inventory trend*, *marts so data*, dan *marts UPFOS*. *Marts complain* berfungsi untuk menyajikan data keluhan pelanggan yang telah diseleksi dari data mentah dengan hanya mengambil kolom-kolom yang relevan tanpa melibatkan proses perhitungan tambahan. Transformasi yang dilakukan pada *marts complain* berfokus pada penyederhanaan struktur data serta penyesuaian penamaan kolom agar lebih konsisten, mudah dipahami, dan siap digunakan dalam proses analisis lanjutan. Implementasi marts complain direalisasikan menggunakan perintah SQL sebagaimana ditunjukkan pada Kode 3.8

```
1  SELECT
2      tanggal_pengajuan_komplain AS Tanggal,
3      member_id,
4      member_name,
5      sku_code AS Product_Code,
6      product_quantity AS Qty_Komplain,
7      no_handphone AS nomor_hp,
8      nomor_pesanan AS order_number,
9      kategori_keterangan AS Alasan_Complain,
10     logistics_name_code AS Ekspedisi,
11     marketplace AS  Marketplace,
12     kategori_masalah_otomatis AS Problem_WH,
13     solusi AS Solusi,
14     penyelesaian_status AS Penyelesaian_Status,
15     product_name AS Product,
```

```
16 FROM {{ source('cartiera_raw', 'input_complain') }}
```

Kode 3.8: Kode SQL *marts complain*

Marts estimation in digunakan untuk menyajikan estimasi barang masuk (*estimated inbound*) berdasarkan data perencanaan dan realisasi *inbound* pada periode berjalan. Data pada mart ini diolah dengan memanfaatkan *window function* untuk menghitung total rencana *inbound* per SKU dalam satu bulan serta akumulasi *inbound* yang telah terealisasi hingga tanggal tertentu. Melalui proses ini, *marts estimation in* mampu memberikan informasi mengenai estimasi *inbound* pada hari berjalan, total estimasi *inbound* bulanan, serta sisa *inbound* yang diperkirakan akan datang. Hasil transformasi ini bertujuan untuk mendukung analisis perencanaan stok dan monitoring realisasi *inbound* secara lebih efisien. Implementasi *marts estimation in* ini dapat dilihat pada kode 3.9

```
1  with base_data as (
2      select
3          tanggal,
4          SKU,
5          planned_in,
6          Inbound,
7          sum(planned_in) over (partition by SKU, date_trunc(tanggal
       , month)) as total_planned_month,
8          sum(Inbound) over (
9              partition by SKU, date_trunc(tanggal, month)
10             order by tanggal
11             rows between unbounded preceding and current row
12         ) as cumulative_inbound
13     from {{ ref('marts_inventory_trend') }}
14     where tanggal >= DATE_TRUNC(CURRENT_DATE(), MONTH)
15 )
16
17 select
18     tanggal,
19     SKU as sku,
20     total_planned_month as est_in_this_month,
21     planned_in as est_in,
22     total_planned_month - cumulative_inbound as est_in_akan_datang
23 from base_data
24 order by tanggal, sku
```

Kode 3.9: Kode SQL *marts est in*

Marts so data digunakan untuk menyajikan data hasil *stock opname*

dengan membandingkan jumlah persediaan yang tercatat pada sistem dengan hasil perhitungan fisik di gudang. Data pada mart ini diperoleh dari sumber data mentah tanpa melibatkan proses perhitungan tambahan, dengan fokus pada pemilihan kolom-kolom yang relevan seperti tanggal pelaksanaan stock opname, identitas produk, nilai persediaan pada sistem, nilai hasil stock opname, serta selisih di antara keduanya. Penyajian data dalam marts so data bertujuan untuk mendukung proses evaluasi akurasi pencatatan stok, identifikasi selisih persediaan, serta sebagai dasar analisis pengendalian inventori. Kode SQL dari marts so data seperti ditunjukkan pada kode 3.10

```
SELECT
    tanggal,
    product_specification_code,
    sku_name,
    inventory_qty as Nilai_Sistem,
    stocktaking_qty as Nilai_Stock_Opname,
    difference as selisih
FROM {{ source('cartiera_raw', 'so_inventory') }}
```

Kode 3.10: Kode SQL *marts so data*

Marts inventory trend digunakan untuk menyajikan pergerakan persediaan barang secara harian berdasarkan berbagai sumber data yang berkaitan dengan aktivitas gudang. *Mart* ini mengintegrasikan data barang masuk, barang keluar, barang cacat (*defect*), hasil *stock opname*, estimasi *inbound*, transaksi UPFOS, serta hasil perbaikan barang *reject* ke dalam satu struktur data yang terpadu. Proses transformasi dilakukan dengan membangun date spine dan daftar SKU agar data persediaan tersedia secara konsisten untuk setiap tanggal dan setiap produk, meskipun tidak terdapat transaksi pada hari tertentu.

Selanjutnya, perhitungan stok dilakukan dengan menjadikan hasil stock opname terakhir sebagai titik awal (anchor point), kemudian menambahkan mutasi stok harian secara kumulatif untuk memperoleh nilai stok awal, stok akhir, serta sisa persediaan. Selain itu, marts inventory trend juga menyediakan metrik analitis seperti selisih antara stok sistem dan hasil stock opname, rata-rata pergerakan barang keluar dalam periode tertentu, *moving average*, *deviasi standar*, serta akumulasi barang keluar bulanan. Penyajian data dalam mart ini bertujuan untuk mendukung analisis tren persediaan, pemantauan stabilitas stok, serta pengambilan keputusan terkait perencanaan dan pengendalian inventori secara lebih akurat. Kode lengkapnya dapat terlihat pada kode 3.11

```
WITH input_barang_masuk AS (
```

```
    SELECT
      tanggal_masuk,
      product_name,
      sku_code,
      total_qty,
      qty_masuk,
      qty_reject
    FROM {{ source('cartiera_raw', 'input_barang_masuk') }}
),

input_barang_keluar AS (
    SELECT
      tanggal_keluar,
      product_name,
      sku_code,
      qty_keluar
    FROM {{ source('cartiera_raw', 'input_barang_keluar') }}
),

input_barang_defect AS (
    SELECT
      tanggal_keluar,
      product_name,
      sku_code,
      qty_defect
    FROM {{ source('cartiera_raw', 'input_barang_defect') }}
),

data_est_in AS (
    SELECT
      tanggal,
      sku,
      product_name,
      qty_planned,
      actual_delivery,
      qty_in_estimation
    FROM {{ source('cartiera_raw', 'data_est_in') }}
),

data_inventory_so AS (
    SELECT
      tanggal,
      product_specification_code AS sku,
```

```
45        sku_name AS product_name,
46        stocktaking_qty AS hasilstockopname
47    FROM {{ source('cartiera_raw', 'so_inventory') }}
48),
49
50upfos_inventory AS (
51    SELECT
52      DATE(tanggal) AS tanggal,
53       product_specification_code AS sku_code,
54      SUM(product_qty) AS qty_upfos
55    FROM {{ source('cartiera_raw', 'upfos_inventory') }}
56    GROUP BY DATE(tanggal), product_specification_code
57),
58
59input_perbaikan_reject AS (
60    SELECT
61        tanggal_masuk,
62        product_name,
63        sku_code,
64        qty_masuk
65    FROM {{ source('cartiera_raw', '
    input_hasil_perbaikan_gudang_reject') }}
66),
67
68-- Get unique SKU list from all data
69sku_list AS (
70    SELECT DISTINCT
71        sku AS sku_code,
72        product_name AS nama_produk
73    FROM data_inventory_so
74),
75
76-- Generate date spine for the next 5 years from today
77tanggal_5_tahun AS (
78    SELECT DISTINCT
79        DATE(tanggal) AS tanggal
80    FROM UNNEST(
81        GENERATE_DATE_ARRAY(
82            DATE('2025-10-01'),
83            DATE_ADD(DATE('2025-10-01'), INTERVAL 5 YEAR),
84            INTERVAL 1 DAY
85        )
86    ) AS tanggal
```

```
),

gabungan_data_harian AS (
    SELECT
        t.tanggal,
        s.sku_code,
        s.nama_produk,
        so.hasilstockopname AS hasilstockopname,
        COALESCE(est.qty_planned, 0) AS planned_in,
        COALESCE(masuk.qty_masuk, 0) AS stock_in,
        COALESCE(masuk.qty_reject, 0) AS stock_in_rejected,
        COALESCE(keluar.qty_keluar, 0) AS stock_out,
        COALESCE(defect.qty_defect, 0) AS stock_defect,
        COALESCE(upfos.qty_upfos, 0) AS stock_out_upfos,
        COALESCE(perbaikan.qty_masuk, 0) AS qty_retur
    FROM tanggal_5_tahun t
    CROSS JOIN sku_list s
    LEFT JOIN data_inventory_so so ON DATE(so.tanggal) = t.tanggal
     AND so.sku = s.sku_code
    LEFT JOIN input_barang_masuk masuk ON DATE(masuk.tanggal_masuk
    ) = t.tanggal AND masuk.sku_code = s.sku_code
    LEFT JOIN input_barang_keluar keluar ON DATE(keluar.
    tanggal_keluar) = t.tanggal AND keluar.sku_code = s.sku_code
    LEFT JOIN input_barang_defect defect ON DATE(defect.
    tanggal_keluar) = t.tanggal AND defect.sku_code = s.sku_code
    LEFT JOIN data_est_in est ON DATE(est.tanggal) = t.tanggal AND
     est.sku = s.sku_code
    LEFT JOIN upfos_inventory upfos ON upfos.tanggal = t.tanggal
    AND upfos.sku_code = s.sku_code
    LEFT JOIN input_perbaikan_reject perbaikan ON DATE(perbaikan.
    tanggal_masuk) = t.tanggal AND perbaikan.sku_code = s.sku_code
),

-- Ambil hasilstockopname dari hari sebelumnya untuk digunakan
    sebagai basis
with_lag_opname AS (
    SELECT
        tanggal,
        sku_code,
        nama_produk,
        hasilstockopname,
        planned_in,
        stock_in,
```

```
122         stock_in_rejected,
123         stock_out,
124         stock_defect,
125         stock_out_upfos,
126         qty_retur,
127         -- Ambil hasilstockopname dari hari sebelumnya sebagai anchor point
128         LAG(hasilstockopname) OVER (PARTITION BY sku_code ORDER BY tanggal) AS hasilstockopname_kemarin
129     FROM gabungan_data_harian
130 ),
131
132 -- Identifikasi tanggal terakhir ada stock opname untuk setiap hari
133 with_opname_group AS (
134     SELECT
135         tanggal,
136         sku_code,
137         nama_produk,
138         hasilstockopname,
139         hasilstockopname_kemarin,
140         planned_in,
141         stock_in,
142         stock_in_rejected,
143         stock_out,
144         stock_defect,
145         stock_out_upfos,
146         qty_retur,
147         -- Buat group ID berdasarkan tanggal terakhir ada opname (dari hari kemarin)
148         SUM(CASE WHEN hasilstockopname_kemarin IS NOT NULL THEN 1 ELSE 0 END)
149             OVER (PARTITION BY sku_code ORDER BY tanggal) AS opname_group_id
150     FROM with_lag_opname
151 ),
152
153 -- Hitung stock base (dari opname terakhir) dan mutasi kumulatif per group
154 with_cumulative AS (
155     SELECT
156         tanggal,
157         sku_code,
```

```
158         nama_produk,
159         hasilstockopname,
160         planned_in,
161         stock_in,
162         stock_in_rejected,
163         stock_out,
164         stock_defect,
165         stock_out_upfos,
166         qty_retur,
167         opname_group_id,
168         -- Stock base dari stock opname terakhir (dari hari
    kemarin)
169         FIRST_VALUE(COALESCE(CAST(hasilstockopname_kemarin AS
    INT64), 0))
170             OVER (PARTITION BY sku_code, opname_group_id ORDER BY
    tanggal ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
    FOLLOWING) AS stock_base,
171         -- Mutasi kumulatif dari awal group sampai hari ini (
    termasuk UPFOS sebagai stock_out dan qty_retur sebagai stock_in
    )
172         SUM(stock_in + qty_retur - stock_out - stock_defect -
    stock_out_upfos)
173             OVER (PARTITION BY sku_code, opname_group_id ORDER BY
    tanggal ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS
    mutasi_kumulatif,
174         -- Mutasi kumulatif sampai kemarin untuk jadi stock awal
    hari ini
175         COALESCE(
176             SUM(stock_in + qty_retur - stock_out - stock_defect -
    stock_out_upfos)
177                 OVER (PARTITION BY sku_code, opname_group_id ORDER
     BY tanggal ROWS BETWEEN UNBOUNDED PRECEDING AND 1 PRECEDING),
178             0
179         ) AS mutasi_kumulatif_kemarin
180     FROM with_opname_group
181 )
182
183 SELECT
184     tanggal,
185     sku_code as SKU,
186     nama_produk as Product,
187     -- Stock awal: stock base + mutasi sampai kemarin
188     sum(stock_base + mutasi_kumulatif_kemarin) AS stock_awal,
```

```
189     sum(planned_in) as planned_in,
190     sum(stock_in) as Inbound,
191     sum(stock_in_rejected) as stock_in_rejected,
192     sum(stock_out + stock_out_upfos) as Outbound,
193     sum(stock_defect) as out_reject,
194     -- Stock akhir: stock base + mutasi sampai hari ini (sudah
    include qty_retur)
195     sum(qty_retur) AS in_retur,
196     sum(stock_base + mutasi_kumulatif) AS Sisa,
197     -- Hasil stock opname dari data_inventory_so
198     max(CAST(hasilstockopname AS INT64)) AS hasilstockopname,
199     -- Selisih: hasilstockopname - stock_akhir
200     max(CASE
201         WHEN hasilstockopname IS NOT NULL
202         THEN CAST(hasilstockopname AS INT64) - (stock_base +
    mutasi_kumulatif)
203         ELSE NULL
204     END) AS selisih,
205
206     avg(sum(stock_out + stock_out_upfos)) OVER (
207         PARTITION BY sku_code
208         ORDER BY tanggal
209         ROWS BETWEEN 59 PRECEDING AND 1 PRECEDING
210     ) AS avg_outbound_60d,
211
212     avg(sum(stock_out + stock_out_upfos)) OVER (
213         PARTITION BY sku_code
214         ORDER BY tanggal
215         ROWS BETWEEN 89 PRECEDING AND CURRENT ROW
216     ) AS moving_avg_90d,
217
218     STDDEV_POP(avg(stock_out + stock_out_upfos)) OVER (
219         PARTITION BY sku_code
220         ORDER BY tanggal
221         ROWS BETWEEN 59 PRECEDING AND 1 PRECEDING
222     ) AS std_dev,
223
224     sum(sum(stock_out + stock_out_upfos)) OVER (
225         PARTITION BY sku_code, date_trunc(tanggal, month)
226         ORDER BY tanggal
227         ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
228     ) as outbound_mtd,
229
```

```
    lag(sum(stock_base + mutasi_kumulatif), 1) OVER (
        PARTITION BY sku_code
        ORDER BY tanggal
    ) AS sisa_yesteday

FROM with_cumulative
GROUP BY tanggal, sku_code, nama_produk
ORDER BY tanggal, sku_code
```

Kode 3.11: Kode SQL *marts inventory trend*

Marts yang terakhir adalah *marts* UPFOS digunakan untuk menyajikan data transaksi penjualan online dari sistem UPFOS yang telah distandarisasi dan siap digunakan untuk analisis. *Mart* ini menyediakan informasi utama terkait transaksi, pelanggan, produk, kanal penjualan, serta pengiriman. Selain itu, ditambahkan atribut urutan transaksi pelanggan untuk mendukung analisis pola dan frekuensi pembelian. Tujuan utama dari *marts* UPFOS adalah mendukung analisis performa penjualan *online* dan perilaku pelanggan secara terstruktur. Secara lengkap kode SQL dapat terlihat pada kode 3.12

```
WITH upfos_raw AS (
  SELECT
    platform AS marketplace,
    dispatch_order_no AS order_number,
    platform_order_id AS order_id,
    product_specification_code AS product_code,
    product_qty AS qty,
    member_name AS `name`,
    recipient_mobile_number AS nomor_hp,
    recipient_address AS `address`,
    source AS item_bundling,
    tanggal,
    "Online" AS category,
    recipient_name AS customer_id,
    logistics_vendor AS ekspedisi,
    sku_name AS product_name,
    NULL AS bundling_name,
    FALSE AS is_bundling,
    SPLIT(region, '/')[SAFE_OFFSET(0)] AS province,
    SPLIT(region, '/')[SAFE_OFFSET(1)] AS city,
    "Utama" AS jenis_sku,
    logistics_tracking_no AS no_resi,
    NULL as sub_category_code,
    NULL sub_category
```

```
  FROM {{ source('cartiera_raw', 'upfos_inventory') }}
)

SELECT
  *,
  -- Transaction sequence per member (customer)
  DENSE_RANK() OVER (PARTITION BY customer_id ORDER BY tanggal) AS
     transaction_sequence,
  DENSE_RANK() OVER (PARTITION BY customer_id, sub_category ORDER
    BY tanggal) AS product_sequence,
  LAG(tanggal) OVER (PARTITION BY customer_id ORDER BY tanggal) AS
     customer_previous_transaction_date,
  LAG(tanggal) OVER (PARTITION BY customer_id, sub_category ORDER
    BY tanggal) AS customer_subcategory_previous_transaction_date,
  LEAD(tanggal) OVER (PARTITION BY customer_id ORDER BY tanggal)
    AS customer_next_transaction_date,
  LEAD(tanggal) OVER (PARTITION BY customer_id, sub_category ORDER
     BY tanggal) AS customer_subcategory_next_transaction_date
FROM upfos_raw
```

Kode 3.12: Kode SQL *marts UPFOS*

D Orkestrasi Pipeline Data Otomatis

Seluruh komponen *pipeline* data Cartiera dikemas ke dalam *container* Docker agar proses eksekusi dapat berjalan secara konsisten dan terisolasi pada lingkungan *cloud*. Pendekatan ini memungkinkan proses *ingestion data* dijalankan sebagai unit mandiri tanpa ketergantungan terhadap lingkungan lokal tempat pengembangan. Implementasi *container* untuk proses ingestion *Stock Opname* direalisasikan melalui Dockerfile sebagaimana ditunjukkan pada Kode 3.13.

```
FROM python:3.10-slim

WORKDIR /app

# Copy requirements and install dependencies
COPY so_ingestion/requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

# Copy common config module (from parent context)
COPY common ./common

# Copy ingestion script
```

```
COPY so_ingestion/ingestion.py ./

CMD ["python", "ingestion.py"]
```

Kode 3.13: Docker SO Ingestion

Selain kode Python ingestion, seluruh file Data Build Tool juga dikemas ke dalam *container* Docker untuk memastikan konsistensi eksekusi di lingkungan *cloud*. Proses transformasi menggunakan DBT dijalankan sebagai *job* terpisah agar tidak bergantung langsung pada proses ingestion data dan dapat dieksekusi secara independen. Implementasi *container* DBT ditunjukkan pada Dockerfile yang dirujuk pada Kode 3.14.

```
FROM python:3.10-slim

WORKDIR /app

# Install dbt-bigquery
RUN pip install --no-cache-dir \
  dbt-bigquery==1.10.1 \
  google-cloud-bigquery==3.38.0

# Copy dbt project
COPY . /app/

# Set profiles directory
ENV DBT_PROFILES_DIR=/app

# Run dbt with full refresh to rebuild tables
CMD ["dbt", "run", "--profiles-dir", "/app", "--project-dir", "/
    app", "--full-refresh"]
```

Kode 3.14: Docker SO Ingestion

Setelah seluruh proses containerisasi selesai, tahap selanjutnya adalah deployment otomatis ke lingkungan Google Cloud menggunakan mekanisme *Continuous Integration* dan *Continuous Deployment* (CI/CD). Proses CI/CD diimplementasikan menggunakan GitHub Actions yang bertugas untuk membangun Docker image, mengunggah image ke Artifact Registry, serta membuat atau memperbarui Cloud Run Job yang digunakan untuk menjalankan proses ingestion dan transformasi data. Selain itu, GitHub Actions juga digunakan untuk melakukan deployment Google Cloud Workflows dan konfigurasi Cloud Scheduler sebagai bagian dari satu alur deployment terintegrasi, sebagaimana ditunjukkan pada Kode 3.15.

```yaml
name: CI/CD Pipeline

on:
  push:
    branches:
      - main
  workflow_dispatch: # Allow manual trigger

env:
  PROJECT_ID: your-gcp-project-id
  REGION: asia-southeast2
  SERVICE_ACCOUNT: your-service-account@your-project.iam.gserviceaccount.com
  ARTIFACT_REGISTRY_REPO: cloud-run-jobs

  # SO Ingestion Image
  SO_IMAGE: asia-southeast2-docker.pkg.dev/your-gcp-project-id/cloud-run-jobs/cartierra-so-ingestion:latest

  # SO Ingestion Job
  SO_JOB: cartierra-so-ingestion

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Authenticate to Google Cloud
        uses: google-github-actions/auth@v2
        with:
          credentials_json: ${{ secrets.GCP_SA_KEY }}

      - name: Set up Google Cloud SDK
        uses: google-github-actions/setup-gcloud@v2
        with:
          project_id: ${{ env.PROJECT_ID }}

      - name: Configure Docker for Artifact Registry
        run: gcloud auth configure-docker asia-southeast2-docker.pkg.dev
```

```
41
42      - name: Enable required APIs
43        run: |
44          gcloud services enable \
45            cloudbuild.googleapis.com \
46            run.googleapis.com \
47            artifactregistry.googleapis.com \
48            cloudscheduler.googleapis.com \
49            workflows.googleapis.com \
50            workflowexecutions.googleapis.com \
51            logging.googleapis.com \
52            bigquery.googleapis.com
53
54      - name: Create Artifact Registry repository (if not exists)
55        run: |
56          if ! gcloud artifacts repositories describe ${{ env.
    ARTIFACT_REGISTRY_REPO }} \
57              --location=${{ env.REGION }} &>/dev/null; then
58            gcloud artifacts repositories create ${{ env.
    ARTIFACT_REGISTRY_REPO }} \
59              --repository-format=docker \
60              --location=${{ env.REGION }} \
61              --description="Docker repository for Cloud Run Jobs"
62          fi
63
64      # ==================== SO Ingestion ====================
65      - name: Build and push SO Ingestion image
66        run: |
67          cd cloud_run_job
68          gcloud builds submit \
69            --config so_ingestion/cloudbuild.yaml \
70            --substitutions=_IMAGE_URL=${{ env.SO_IMAGE }} \
71            --timeout=20m \
72            --machine-type=e2-highcpu-8
73
74      - name: Deploy SO Ingestion Job
75        run: |
76          if gcloud run jobs describe ${{ env.SO_JOB }} --region=$
    {{ env.REGION }} &>/dev/null; then
77            gcloud run jobs update ${{ env.SO_JOB }} \
78              --image=${{ env.SO_IMAGE }} \
79              --region=${{ env.REGION }} \
80              --memory=2Gi \
```

```
              --cpu=1 \
              --max-retries=2 \
              --task-timeout=3600s \
              --service-account=${{ env.SERVICE_ACCOUNT }} \
              --set-env-vars=LOOKBACK_DAYS=7
          else
            gcloud run jobs create ${{ env.SO_JOB }} \
              --image=${{ env.SO_IMAGE }} \
              --region=${{ env.REGION }} \
              --memory=2Gi \
              --cpu=1 \
              --max-retries=2 \
              --task-timeout=3600s \
              --service-account=${{ env.SERVICE_ACCOUNT }} \
              --set-env-vars=LOOKBACK_DAYS=7
          fi

      # ==================== Deployment Summary
  ====================
      - name: Deployment Summary
        run: |
          echo "================================================"
          echo "    SO INGESTION DEPLOYMENT COMPLETED!"
          echo "================================================"
          echo ""
          echo "       Cloud Run Job:"
          echo "      ${{ env.SO_JOB }}"
          echo ""
          echo "       Console Link:"
          echo "  - Jobs: https://console.cloud.google.com/run/
  jobs?project=${{ env.PROJECT_ID }}"
          echo "================================================"
```

Kode 3.15: Kode Github Action

Melalui pendekatan ini, GitHub Actions berperan sebagai pengelola siklus hidup infrastruktur *pipeline* data, sedangkan eksekusi *pipeline* secara harian dijalankan oleh Google Cloud Workflows yang dipicu oleh Cloud Scheduler. Workflow bertanggung jawab mengatur urutan eksekusi Cloud Run Job, dimulai dari proses ingestion data hingga transformasi menggunakan DBT, sehingga dependensi antarproses dapat terjaga dengan baik. Dengan arsitektur tersebut, *pipeline data* Cartiera dapat berjalan secara otomatis, terjadwal, dan konsisten tanpa memerlukan intervensi manual.

E BI Dashboard

Sebagai keluaran akhir dari proses *pipeline data*, data yang telah melalui tahap *ingestion*, transformasi, dan pemuatan ke dalam *data warehouse* Google BigQuery kemudian disajikan dalam bentuk *Business Intelligence (BI) dashboard*. Dashboard ini menjadi representasi visual dari hasil pengolahan data yang telah terstandarisasi dan terintegrasi, sehingga dapat dimanfaatkan secara langsung oleh pengguna.

BI Dashboard dibangun menggunakan Looker Studio dan terhubung langsung dengan tabel *data marts* hasil pemrosesan DBT. Dengan integrasi tersebut, dashboard mampu menampilkan data yang konsisten dan diperbarui secara berkala sesuai dengan alur *pipeline data* yang telah dirancang. Hal ini memastikan bahwa informasi yang disajikan bersumber dari data yang telah melalui proses validasi dan transformasi yang sistematis.

Tampilan dashboard, sebagaimana ditunjukkan pada Gambar 3.6, merepresentasikan hasil akhir dari keseluruhan proses pengolahan data dalam satu antarmuka visual. Penyajian ini memungkinkan pengguna untuk mengakses dan memantau informasi hasil olahan data tanpa perlu berinteraksi langsung dengan sistem basis data atau melakukan pengolahan data secara manual.

Gambar 3.6. Tampilan BI Dashboard Operasional Cartiera

Secara keseluruhan, BI Dashboard berperan sebagai lapisan akhir yang menjembatani proses pengolahan data teknis dengan kebutuhan analisis dan pengambilan keputusan. Dengan dukungan *pipeline data* yang terotomatisasi, dashboard memastikan hasil pengolahan data dapat diakses secara efektif, konsisten, dan berkelanjutan.

3.4 Kendala dan Solusi yang Ditemukan

3.4.1 Kendala yang Dihadapi

Selama proses kegiatan magang berlangsung, ditemukan beberapa kendala sebagai berikut:

1. Proses penyesuaian terhadap lingkungan kerja dari sistem yang telah berjalan
2. Keterbatasan dokumentasi pada beberapa bagian sistem yang sudah dibuat
3. Penyesuaian terhadap pengelolaan waktu dan pembagian prioritas pekerjaan menjadi tantangan

3.4.2 Solusi atas Kendala yang Ditemukan

Dari berbagai kendala yang ditemukan selama pelaksanaan kegiatan magang, diperoleh solusi sebagai berikut:

1. Dilakukan proses pembelajaran bertahap melalui penelusuran alur sistem yang telah berjalan
2. Keterbatasan dokumentasi diatasi dengan melakukan diskusi dan koordinasi secara langsung dengan Kak Fattah selaku supervisor
3. Pengelolaan waktu dan prioritas pekerjaan ditingkatkan dengan menyusun jadwal kerja yang lebih terstruktur, lalu untuk setiap task yang dikerjakan ditulis dan disusun secara prioritas melalui Trello

Secara keseluruhan, solusi yang diterapkan memungkinkan kendala yang muncul dapat diatasi secara bertahap.