BAB III

PELAKSANAAN KERJA

3.1 Kedudukan dan Koordinasi

Setiap posisi memiliki kedudukan yang jelas serta pola koordinasi yang terstruktur agar pekerjaan dapat berjalan efektif dalam pelaksanaan program magang di Kawan Lama Group. Kedudukan menggambarkan peran dan posisi intern dalam struktur organisasi perusahaan, sementara koordinasi menjelaskan hubungan kerja sama dengan tim atau divisi lain yang terkait. Pemahaman yang baik mengenai kedudukan dan koordinasi membantu dalam penyesuaian diri dengan budaya kerja perusahaan serta memberikan kontribusi yang relevan terhadap proyek yang sedang berjalan. Berikut penjelasan rinci mengenai kedudukan dan koordinasi posisi *Data Engineer Associate Intern* dalam lingkungan kerja Kawan Lama Group.

3.1.1 Kedudukan

Posisi *Data Engineer Associate Intern* merupakan bagian dari divisi *Data Engineer* di Kawan Lama Group, yang memiliki peran utama dalam membantu membangun arsitektur data yang terstruktur, agar data siap digunakan oleh unit-unit bisnis. Peran ini sangat penting karena hampir seluruh proses analisis dan pengambilan keputusan strategis perusahaan bergantung pada ketersediaan data yang akurat, terstruktur, dan mudah diakses. Pada periode magang, posisi *Data Engineer Associate Intern* diisi oleh dua orang yang memiliki lingkup pekerjaan serupa, meskipun detail tugas yang diberikan dapat berbeda sesuai dengan kebutuhan tim. Secara umum, tanggung jawab posisi magang ini tidak dibedakan secara signifikan dengan anggota divisi *Data Engineer* lainnya, sehingga kesempatan untuk belajar dan berkontribusi terbuka sangat luas. Setiap tugas yang diberikan berasal dari arahan tim *Data Analytics*, yang berfungsi sebagai pengguna utama data sekaligus pihak yang menentukan prioritas pekerjaan. Dalam pelaksanaannya, posisi magang berada di bawah naungan tim *Data Engineer*,

tetapi dapat pula ditempatkan secara fleksibel pada lingkup pekerjaan yang lebih spesifik, misalnya di tim AI/ML, atau *Analytics Engineer*, tergantung dari kebutuhan proyek yang sedang berjalan. Pola koordinasi yang fleksibel ini mencerminkan budaya kerja Kawan Lama Group yang kolaboratif dan adaptif, sehingga tidak hanya memahami peran teknis, tetapi juga mendapatkan gambaran menyeluruh mengenai integrasi kerja antar divisi dalam mendukung tujuan perusahaan.

3.1.2 Koordinasi

Tim *Data Analytics* mengadopsi metode kerja berbasis *sprint* dengan siklus mingguan yang dimulai setiap hari Selasa dan berakhir pada hari Senin minggu berikutnya. Pada metode ini, setiap tugas telah direncanakan dengan matang dan target pencapaian ditetapkan secara jelas untuk setiap periode *sprint*. Penerapan metode *sprint* ini bertujuan untuk meningkatkan produktivitas tim sekaligus memastikan bahwa setiap proyek dapat diselesaikan tepat waktu sesuai dengan kebutuhan bisnis perusahaan. Seluruh aktivitas *sprint* dikelola melalui platform Jira, sebuah perangkat lunak manajemen proyek yang dikembangkan oleh Atlassian. Jira berfungsi sebagai alat untuk melacak progres pekerjaan, mengelola *backlog* tugas, serta mendokumentasikan seluruh aktivitas tim. Melalui Jira, setiap anggota tim dapat memantau status pekerjaan secara *real-time*, mengidentifikasi hambatan yang muncul, dan berkolaborasi dengan anggota tim lainnya secara lebih efektif.

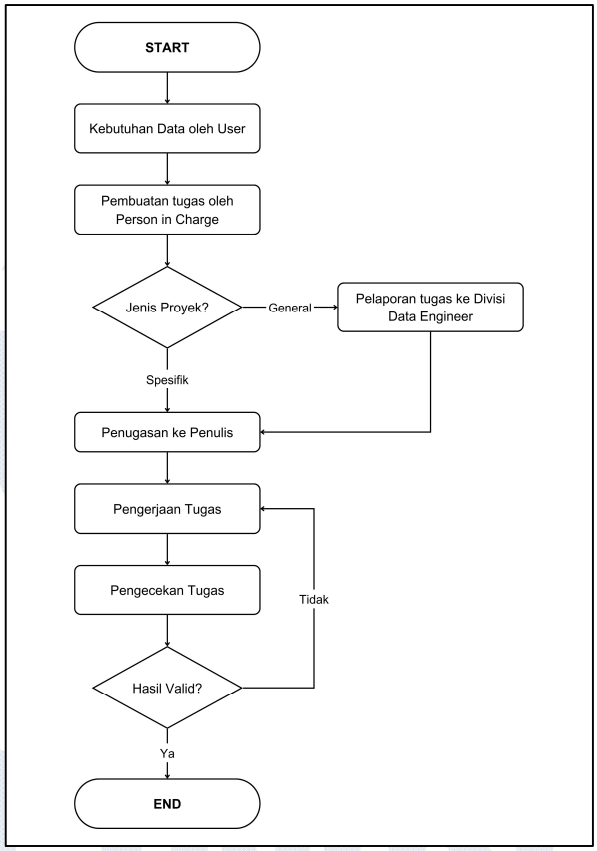
Untuk menjaga koordinasi dan komunikasi yang optimal, tim mengadakan *Daily Stand Up* (DSU) setiap hari pada pukul 09.00 WIB selama periode *sprint* berlangsung. Pertemuan harian ini memberikan kesempatan bagi setiap anggota tim untuk menyampaikan *update* mengenai kemajuan pekerjaan yang sedang ditangani, berbagi informasi tentang kendala atau hambatan teknis yang dihadapi, serta memaparkan rencana kerja untuk hari tersebut. Format pertemuan yang singkat namun rutin ini dirancang agar tim dapat segera mengidentifikasi masalah dan mencari solusi secara kolektif

tanpa mengganggu alur kerja harian. Seluruh informasi yang dibahas dalam DSU kemudian didokumentasikan dalam Jira untuk memastikan adanya jejak rekam yang dapat dirujuk kembali apabila diperlukan.

Setiap hari Senin, tim *Data Analytics* mengadakan sesi *backlog grooming* yang melibatkan seluruh anggota dari berbagai divisi. Sesi ini berfungsi sebagai forum perencanaan bersama untuk meninjau tugas-tugas yang akan dilakukan pada *sprint* selanjutnya. Selain itu, tim juga membahas dan menentukan prioritas serta tingkat urgensi dari tugas-tugas yang masih tertunda di *backlog* Jira, sehingga setiap anggota memiliki pemahaman yang sama mengenai apa yang harus dikerjakan. Setelah sesi *backlog grooming* selesai, setiap divisi melanjutkan ke sesi *sprint planning* internal di hari yang sama. Pada sesi ini, setiap divisi memetakan tugas-tugas untuk minggu berikutnya secara mendetail, menentukan siapa yang bertanggung jawab atas masing-masing tugas, dan menetapkan perkiraan waktu penyelesaiannya. Proses perencanaan ini memastikan bahwa seluruh pekerjaan dapat berjalan dengan terstruktur dan selaras dengan target bisnis yang telah ditetapkan, serta efisien dalam pelaksanaannya, sehingga tim dapat memaksimalkan hasil kerja dalam setiap siklus *sprint*.

Tim *Data Analytics* dalam menjalankan tugasnya selalu melakukan koordinasi yang sistematis lintas divisi untuk memastikan setiap permintaan dapat diproses secara efisien serta mengeluarkan hasil yang akurat dan sesuai dengan kebutuhan bisnis. Alur koordinasi ini divisualisasikan pada Gambar 3.1 yang menunjukkan diagram proses kerja tim *Data Analytics* dalam memenuhi kebutuhan *user*. Proses dimulai ketika *user* mengajukan permintaan atau menyampaikan kebutuhannya kepada *Person in Charge* (PIC). Penentuan PIC sangat bergantung pada jenis proyek yang sedang dijalankan. Jika proyek yang dikerjakan bersifat spesifik dengan tujuan yang jelas dan memiliki tim khusus, maka PIC umumnya adalah *project lead* dari tim tersebut. Sebaliknya, apabila tugas yang diajukan hanya bersifat minor, tidak memerlukan perubahan besar, atau termasuk dalam pekerjaan rutin,

maka proyek tersebut dikategorikan sebagai proyek general. Untuk proyek *general* atau umum, PIC biasanya berasal dari tim *Data Platform* yang memang berperan sebagai penghubung utama dengan *user*.



Gambar 3.1 Alur Koordinasi Pekerjaan Tim *Data Analytics* Kawan Lama Group

Setelah menerima kebutuhan dari *user*, PIC kemudian mencatat dan mendistribusikan pekerjaan ke dalam sistem manajemen tugas seperti Jira. Pada proyek spesifik, *project lead* dapat langsung menugaskan pekerjaan tersebut kepada penulis selaku *intern* untuk dikerjakan. Namun, pada proyek general, alur penugasan sedikit berbeda. PIC akan terlebih dahulu membawa tugas tersebut ke sesi *backlog grooming*, di mana kebutuhan tersebut didiskusikan bersama tim terkait. Setelah disepakati, tugas akan diteruskan kepada divisi yang relevan, kemudian karyawan di divisi tersebut akan secara resmi menugaskannya kepada penulis. Kemudian penulis mengerjakan tugas sesuai instruksi dan kebutuhan yang telah diberikan. Setelah pekerjaan

selesai, hasilnya akan diperiksa oleh PIC sebagai bentuk *quality control*. Apabila hasil sudah sesuai standar dan memenuhi kebutuhan, maka tugas dianggap selesai. Namun, jika ditemukan ketidaksesuaian atau masih terdapat hal yang perlu diperbaiki, maka PIC akan mengembalikan tugas untuk dilakukan revisi. Proses ini terus berlangsung hingga hasil akhir benar-benar valid dan siap digunakan oleh *user*. Alur koordinasi yang terstruktur ini memungkinkan setiap permintaan dapat ditangani dengan baik, sekaligus menjaga kualitas hasil agar tetap konsisten dan sesuai dengan standar perusahaan.

3.2 Tugas yang Dilakukan

Tugas magang yang dilaksanakan berlandaskan pada kebutuhan nyata bisnis maupun kebutuhan internal divisi untuk melakukan optimalisasi terhadap sistem yang telah berjalan. Selama periode magang, terdapat dua proyek utama yang dijalankan guna menjawab tantangan tersebut. Proyek pertama berfokus pada pengembangan dan pengoptimalan AI Generatif yang digunakan sebagai katalog produk yang menampilkan informasi produk secara lebih lengkap dan relevan. Proyek ini sebagian besar melibatkan kolaborasi dengan divisi *Data Engineer*, khususnya tim AI/ML yang memiliki keahlian dalam merancang dan menyesuaikan model berbasis kecerdasan buatan. Meskipun demikian, proses optimasi AI katalog produk dikerjakan secara mandiri sebagai bagian dari tanggung jawab individu dalam proyek ini. Sementara itu, proyek kedua diarahkan pada pengembangan model *Machine Learning* (ML) untuk melakukan prediksi permintaan barang untuk perencanaan inventaris, yang berfungsi mendukung efektivitas distribusi dan ketersediaan barang di suatu unit bisnis. Proyek ini dikerjakan oleh tim khusus beranggotakan empat orang, terdiri dari tiga *Data Engineer* dan seorang anggota *Data Platform*, yang secara bersama-sama menyusun rancangan teknis sekaligus mengimplementasikan solusi. Kontribusi yang dilakukan pada proyek ini difokuskan pada pembuatan *data mart* dan pembuatan model prediksi permintaan, sehingga hasil prediksi dapat dimanfaatkan secara langsung oleh user melalui *pipeline* data yang terintegrasi.

Kedua proyek tersebut dikomunikasikan secara intensif dengan *Person in Charge* (PIC) masing-masing, baik untuk membahas kebutuhan teknis, menyelaraskan prioritas bisnis, serta memastikan bahwa hasil kerja memenuhi standar kualitas yang diharapkan. Tugas-tugas yang berasal dari proyek tersebut kemudian dimasukkan ke dalam *sprint* mingguan tim *Data Analytics*, di mana setiap PIC akan mengatur dan membagikan pekerjaan sesuai dengan estimasi waktu pengerjaan yang telah direncanakan. Setiap tahapan proyek memiliki jalur koordinasi yang jelas, sehingga progres dapat dipantau secara berkala dan risiko keterlambatan dapat diminimalisasi. Sebagai bentuk dokumentasi, rincian detail pekerjaan yang telah dilaksanakan sepanjang masa magang disajikan pada Tabel 3.1. Tabel ini merepresentasikan realisasi tugas yang telah diselesaikan, sekaligus menunjukkan keterkaitan setiap pekerjaan dengan lini masa proyek yang sebelumnya telah digambarkan pada Tabel 1.1, sehingga dapat memberikan gambaran mengenai kontribusi yang dihasilkan selama periode magang.

Tabel 3.1 Tabel Detail Pekerjaan yang Dilakukan

No.	Aktivitas	Periode ke-	Tanggal Mulai	Tanggal Selesai
1	**Mengelola dan menganalisis data untuk mendukung *data-driven decision-making***	2 Juli	14 Juli 2025	19 September 2025
1.1	Menganalisis kebutuhan bisnis dan sumber data	2 Juli	14 Juli 2025	4 September 2025
1.2	Mengumpulkan data dari sumber	4 Juli	28 Juli 2025	15 September 2025
1.3	Membuat *data mart*	4 Juli	4 Agustus 2025	19 September 2025
2	**Mengoptimasi AI generatif untuk meningkatkan akses informasi dan efisiensi manajemen produk**	3 Juli	21 Juli 2025	26 Agustus 2025
2.1	Mempelajari konsep AI generatif menggunakan LangChain dan LangGraph	3 Juli	21 Juli 2025	25 Juli 2025
2.2	Melakukan proses *embedding* teks dan pemuatan ke dalam basis data vektor	2 Agustus	11 Agustus 2025	14 Agustus 2025

No.	Aktivitas	Periode ke-	Tanggal Mulai	Tanggal Selesai
2.3	Melakukan *fine-tuning* dan pengujian *Large Language Model*	3 Agustus	15 Agustus 2025	25 Agustus 2025
3	**Membangun model prediktif menggunakan *Machine Learning* agar hasil lebih mudah diakses dan diimplementasikan secara efisien**	4 September	22 September 2025	3 November 2025
3.1	Mengeksplorasi jenis model prediksi BigQuery ML	4 September	21 September 2025	23 September 2025
3.2	Menganalisis data histori untuk membuat data pelatihan	4 September	24 September 2025	26 September 2025
3.3	Membuat dan melatih model prediksi	1 Oktober	29 September 2025	14 Oktober 2025
3.4	Menguji dan membandingkan hasil evaluasi kinerja model	3 Oktober	15 Oktober 2025	17 Oktober 2025
3.5	Mengimplementasikan model prediksi dan integrasi *output*	4 Oktober	22 Oktober 2025	31 Oktober 2025

Berdasarkan Tabel 3.1, setiap aktivitas yang dijalankan selama periode magang memiliki alur yang sistematis, mulai dari perencanaan, eksekusi, hingga penyelesaian. Setiap tugas dicatat dengan jelas melalui pembagian waktu berdasarkan minggu, disertai tanggal mulai dan selesai untuk memastikan keteraturan serta keterlacakan progres. Detail pekerjaan tersebut membantu dalam memantau ketepatan jadwal, serta memberikan gambaran konkret mengenai beban kerja yang diselesaikan dalam setiap fase proyek. Pencatatan yang rinci memudahkan tim dalam melakukan evaluasi, baik terhadap efisiensi pengerjaan maupun terhadap kualitas hasil yang dicapai, sehingga memudahkan proses refleksi dan pengembangan strategi kerja untuk proyek serupa di masa mendatang.

3.3 Uraian Pelaksanaan Kerja

Pelaksanaan kerja magang pada divisi *Data Engineer* di Kawan Lama Group berjalan dalam suasana kerja yang dinamis, di mana setiap aktivitas selalu terkait erat dengan kebutuhan operasional maupun pengembangan sistem yang sedang berlangsung. Pekerjaan tidak hanya terbatas pada penyelesaian tugas teknis, tetapi

juga melibatkan pemahaman konteks bisnis yang lebih luas agar setiap solusi yang dihasilkan benar-benar relevan. Proses pengerjaan dimulai dari diskusi singkat mengenai kebutuhan proyek, dilanjutkan dengan pembagian peran, serta eksekusi yang terintegrasi dalam *sprint* mingguan. Dalam praktiknya, setiap tahapan tidak berdiri sendiri, melainkan saling melengkapi, mulai dari proses ekstraksi dan transformasi data, pembuatan *pipeline* yang stabil, hingga validasi hasil untuk memastikan kualitas data yang diolah dapat mendukung pengambilan keputusan. Untuk mendukung seluruh proses tersebut, perusahaan menyediakan berbagai perangkat dan platform yang terintegrasi, sehingga pekerjaan dapat dilakukan dengan lebih terstruktur dan efisien. *Tools* yang digunakan selama pelaksanaan magang di antaranya:

1. Apache Airflow

   Apache Airflow berfungsi sebagai *tools* orkestrasi *pipeline* data utama yang digunakan oleh divisi *Data Engineer* Kawan Lama Group. Sebagai *workflow orchestrator*, Airflow menjalankan serangkaian proses otomatis dalam pengelolaan data, mulai dari tahap ekstraksi hingga penyimpanan ke *data lake* atau *data warehouse*. Kemampuan penjadwalan yang dimiliki memungkinkan setiap *pipeline* dijalankan secara otomatis berdasarkan interval waktu yang telah ditentukan, baik harian, mingguan, bulanan, sehingga pembaruan data berlangsung konsisten tanpa memerlukan intervensi manual. Setiap *pipeline* data dalam Airflow didefinisikan melalui *Directed Acyclic Graph* (DAG), sebuah struktur yang menggambarkan urutan serta ketergantungan antar *task* dalam proses otomasi. DAG memastikan alur eksekusi berjalan tanpa membentuk siklus, sehingga setiap *task* dieksekusi berdasarkan urutan dan dependensi yang telah ditentukan.

2. Visual Studio Code

   Visual Studio Code (VS Code) adalah *integrated development environment* (IDE) yang banyak digunakan dalam pengembangan perangkat lunak maupun analisis data. VS Code dikembangkan oleh Microsoft dengan fitur utama berupa editor yang tetap memiliki kemampuan lengkap untuk mendukung

berbagai bahasa pemrograman. Dalam pelaksanaan magang, VS Code berperan sebagai alat utama dalam menulis, menguji, sekaligus memperbaiki skrip yang digunakan untuk keperluan *pipeline* data, baik untuk proses ETL. Keunggulan VS Code terletak pada ketersediaan ekstensi yang sangat beragam, mulai dari dukungan Python, SQL, YAML, hingga integrasi langsung dengan Git. Hal ini memudahkan proses kolaborasi serta memastikan kualitas kode tetap terjaga.

3. GitLab

   GitLab adalah platform *version control system* sekaligus collaboration *tool* berbasis Git yang mendukung pengelolaan repositori kode secara terpusat. GitLab tidak hanya digunakan untuk menyimpan kode, tetapi juga untuk mengatur alur kerja pengembangan melalui *branching strategy*, *merge request*, serta integrasi dengan *pipeline* CI/CD. Dengan GitLab, setiap perubahan kode dapat terdokumentasi secara rapi, sehingga memungkinkan proses kolaborasi yang transparan dan terkendali. Dalam praktik magang, GitLab digunakan untuk menyimpan seluruh repositori proyek tim *Data Analytics*. Hal ini memastikan setiap anggota tim dapat mengakses, meninjau, dan memberikan masukan terhadap kode yang sedang dikembangkan. GitLab juga memfasilitasi proses peninjauan kode yang penting untuk menjaga standar kualitas sebelum kode diimplementasikan ke sistem utama.

4. Google BigQuery

   Google BigQuery merupakan layanan *cloud data warehouse* yang dikembangkan oleh Google Cloud Platform (GCP), yang dirancang khusus untuk menangani analisis data dalam skala besar secara cepat dan efisien. BigQuery bekerja dengan konsep *serverless*, sehingga pengguna tidak perlu melakukan pengaturan infrastruktur secara manual. BigQuery di Kawan Lama Group berperan sebagai pusat penyimpanan sekaligus pengolahan data dari berbagai sumber, baik data penjualan, inventaris, hingga data operasional lainnya. BigQuery memungkinkan integrasi langsung dengan *pipeline* ETL yang dibangun oleh tim *Data Engineer*, sehingga data yang sudah masuk ke

data warehouse dapat langsung digunakan untuk berbagai keperluan analisis maupun model *machine learning*. Salah satu keunggulannya adalah kemampuan menjalankan *query* SQL dengan performa tinggi, bahkan pada dataset berukuran *terabyte* hingga *petabyte*, tanpa memerlukan waktu pemrosesan yang lama. BigQuery juga dilengkapi fitur *partitioning* dan *clustering* yang memungkinkan data dikelompokkan dan diakses lebih efisien, sehingga biaya penggunaan dapat dioptimalkan. Selain itu, BigQuery mendukung integrasi dengan berbagai *tools* analitik dan visualisasi seperti Looker Studio sehingga hasil analisis dapat dipresentasikan dengan lebih jelas untuk mendukung pengambilan keputusan.

5. Jupyter Notebook

   Jupyter Notebook adalah salah satu platform interaktif yang populer digunakan dalam analisis data, pengembangan model *machine learning*, maupun eksplorasi kode. Notebook ini berbasis web dan memungkinkan pengguna menulis kode, menjalankan perintah, serta menambahkan catatan atau visualisasi data dalam satu dokumen yang terintegrasi. Keunggulan Jupyter Notebook terletak pada fleksibilitasnya dalam mendukung berbagai bahasa pemrograman, terutama Python yang menjadi bahasa utama dalam pengolahan data. Selain itu, notebook ini juga memudahkan dokumentasi proses kerja karena setiap langkah kode dapat langsung dijelaskan dengan *markdown*, sehingga alur pengerjaan menjadi lebih mudah dipahami oleh anggota tim lainnya.

6. Google Colab

   Google Colaboratory atau yang lebih dikenal dengan Google Colab adalah platform berbasis *cloud* yang dikembangkan oleh Google, yang memungkinkan pengguna menjalankan kode Python secara langsung tanpa perlu melakukan instalasi di perangkat lokal. Google Colab terintegrasi dengan Google Drive, sehingga memudahkan kolaborasi antar pengguna dalam mengakses maupun membagikan *notebook*. Keunggulan utama Google Colab adalah tersedianya akses gratis terhadap GPU dan TPU, yang sangat

membantu dalam mempercepat proses pelatihan model *machine learning* yang membutuhkan komputasi tinggi. Dalam konteks magang, Colab juga memudahkan pengujian model dengan cepat karena lingkungannya sudah dilengkapi banyak pustaka populer seperti TensorFlow, PyTorch, dan scikit-learn. Selain itu, Colab mendukung kerja tim dengan fitur berbagi yang mirip Google Docs, sehingga beberapa anggota dapat meninjau dan menjalankan *notebook* yang sama secara bersamaan.

7. Chainlit

   Chainlit merupakan kerangka kerja *open-source* yang digunakan untuk membangun dan menguji aplikasi berbasis *Large Language Model* (LLM). Kerangka kerja ini menyediakan antarmuka interaktif yang memungkinkan pengembang melakukan uji coba model bahasa melalui tampilan web tanpa perlu membuat sistem tambahan di sisi *front-end*. Penggunaan Chainlit membantu proses pengembangan menjadi lebih efisien karena hasil keluaran model dapat langsung ditampilkan dan dievaluasi melalui halaman antarmuka. Chainlit juga memiliki kompatibilitas yang baik dengan framework AI populer seperti LangChain, sehingga banyak digunakan dalam proses riset dan pengembangan aplikasi berbasis kecerdasan buatan yang berorientasi pada interaksi pengguna.

Seluruh *tools* yang digunakan selama pelaksanaan magang memiliki peran penting dalam mendukung efisiensi dan ketepatan pekerjaan. Setiap perangkat memiliki fungsi yang saling melengkapi, mulai dari tahap pengembangan kode, pengelolaan repositori, pemrosesan data, hingga pengujian model. Kombinasi berbagai *tools* tersebut membantu proses kerja berjalan lebih terstruktur dan mudah dipantau, baik dari sisi teknis maupun kolaboratif. Pemanfaatan lingkungan kerja yang berbasis *cloud* dan *open-source juga* memberikan fleksibilitas tinggi, memungkinkan integrasi lintas platform serta meminimalkan kendala teknis selama implementasi proyek. Keseluruhan dukungan teknologi ini berperan besar dalam memastikan hasil kerja mencapai standar kualitas yang diharapkan serta dapat diterapkan secara berkelanjutan di lingkungan kerja profesional.

3.3.1 Proses Pelaksanaan

Proses pelaksanaan kerja magang mencakup serangkaian tahapan yang saling berkaitan dan berfokus pada pencapaian tujuan proyek yang telah direncanakan sejak awal. Setiap aktivitas dijalankan secara sistematis, dimulai dari analisis kebutuhan bisnis hingga implementasi teknis dan evaluasi hasil. Pelaksanaan tugas dilakukan melalui koordinasi rutin bersama pembimbing lapangan maupun tim terkait, guna memastikan setiap keputusan dan langkah teknis tetap selaras dengan kebutuhan perusahaan serta tujuan proyek. Dalam prosesnya, kegiatan magang tidak hanya berorientasi pada penyelesaian teknis semata, tetapi juga menekankan pemahaman terhadap konteks bisnis, alur data, serta bagaimana hasil kerja dapat memberikan nilai tambah terhadap efisiensi sistem yang sudah ada. Bagian berikut menjelaskan secara rinci tahapan pekerjaan yang dilakukan selama pelaksanaan magang.

3.3.1.1 Menganalisis kebutuhan bisnis dan sumber data

Seluruh proyek yang dijalankan berawal dari proses identifikasi kebutuhan bisnis yang dilakukan melalui diskusi dan koordinasi bersama tim proyek serta pihak *user* atau *stakeholder* yang berkepentingan. Tahap ini menjadi dasar dalam menentukan arah pengembangan sistem dan memastikan solusi yang dihasilkan sesuai dengan kebutuhan operasional maupun tujuan strategis perusahaan. Proses analisis dilakukan melalui beberapa kali pertemuan, di mana setiap anggota tim berperan dalam menggali kebutuhan pengguna, memahami permasalahan yang dihadapi, serta menetapkan ruang lingkup pekerjaan yang realistis untuk diselesaikan selama magang.

Pada proyek katalog produk, analisis kebutuhan difokuskan pada bagaimana sistem AI dapat membantu pengguna memperoleh informasi produk secara cepat dan ringkas. Berdasarkan hasil diskusi, user menginginkan sistem yang mampu menampilkan daftar produk disertai informasi penting yang relevan, seperti deskripsi, harga, dan ketersediaan, dalam format poin yang mudah dipahami. Tujuannya

adalah agar sistem mampu memberikan jawaban yang efisien dan informatif ketika pengguna mengajukan pertanyaan mengenai rekomendasi produk tertentu. Dalam tahap ini, tim juga meninjau kembali struktur data produk yang tersedia di sistem internal untuk memastikan bahwa semua informasi yang dibutuhkan sudah dapat diakses dan diolah oleh model AI. Setelah kebutuhan dan ruang lingkup dipastikan, hasil analisis tersebut menjadi dasar penyusunan rancangan teknis serta perancangan alur data yang digunakan pada tahap pengembangan berikutnya. Daftar informasi yang menjadi komponen utama sistem AI dirangkum dalam Tabel 3.2.

Tabel 3.2 Tabel Daftar Informasi Produk yang Dibutuhkan

No.	Informasi	Deskripsi
1	SKU	Kode produk
2	Nama produk	Nama produk
3	Harga	Harga produk
4	Status	Status produk (aktif atau *discontinue*)
5	Deskripsi	Deskripsi broduk
6	Spesifikasi	Spesifikasi produk
7	Keunggulan	Keunggulan produk
8	Garansi	Jenis garansi dan lamanya
9	Jumlah terjual	Jumlah produk terjual dalam 3 dan 6 bulan terakhir
10	Sisa stok	Sisa stok produk di setiap gudang
11	Produk alternatif	Alternatif produk (apabila status produk *discontinue*)

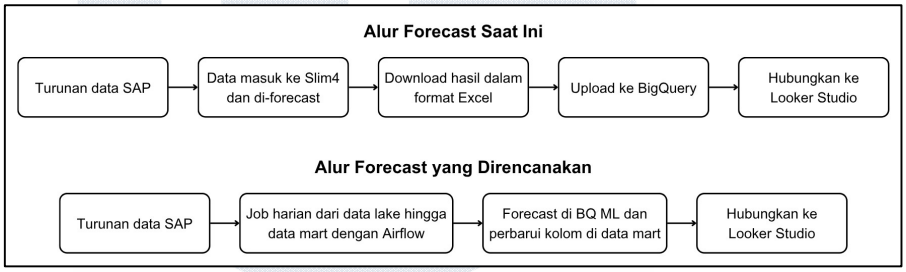
Informasi produk yang harus ditampilkan pada sistem AI pada Tabel 3.2 bersumber dari platform internal milik Kawan Lama Group yang menjadi pusat aktivitas operasional harian *user*, seperti pengelolaan katalog, pembaruan harga, hingga pemantauan status produk. Sementara itu, data yang berkaitan dengan volume penjualan serta jumlah stok diperoleh dari *data warehouse* yang tersimpan di BigQuery. *Data warehouse* tersebut berfungsi sebagai penyimpanan terpusat yang menampung informasi historis maupun *real-time* dari

berbagai sumber data perusahaan. Hasil pengambilan data dari platform internal ke BigQuery akan berpengaruh terhadap pengembangan sistem AI, karena memungkinkan model untuk mengakses data yang konsisten, terstruktur, dan siap diolah menjadi keluaran yang bermanfaat bagi pengguna.

Pada proyek perencanaan inventaris, tujuan utama yang ingin dicapai adalah membangun sistem baru yang dapat menghasilkan data pengganti dari *software* vendor agar proses peramalan stok dapat dijalankan langsung di BigQuery. Permintaan ini muncul karena sistem lama dianggap tidak efisien, terutama dari sisi waktu dan biaya operasional. Proses prediksi melalui *software* tersebut membutuhkan langkah manual yang panjang, mulai dari pengambilan data, pengunggahan *file*, hingga penyesuaian kembali di sistem analitik. Selain itu, keterbatasan integrasi antarplatform membuat hasil peramalan sering kali terlambat masuk ke laporan bisnis, sehingga memengaruhi kecepatan pengambilan keputusan.

Gambar 3.2 memperlihatkan perbandingan antara alur prediksi yang berjalan saat ini dan alur yang diusulkan. Pada alur saat ini, data diambil dari SAP, lalu dikirim ke *software* vendor untuk dihitung proyeksi kebutuhan stok selama satu bulan ke depan. Hasil perhitungan tersebut kemudian diunduh dalam bentuk Excel dan diunggah secara manual ke BigQuery menjadi tabel, di mana tabel tersebut kemudian dihubungkan ke Looker Studio untuk menampilkan laporan. Proses ini memakan waktu cukup lama karena melibatkan banyak perpindahan data dan masih bergantung pada campur tangan pengguna. Kapasitas BigQuery yang digunakan juga terbatas karena masih memakai akun versi gratis, sehingga data yang bisa diproses tidak terlalu besar. Sistem yang baru dirancang agar seluruh proses bisa berjalan otomatis dan terpusat di satu ekosistem. Data dari SAP diturunkan ke *data lake*, kemudian diteruskan hingga

ke data mart menggunakan penjadwalan harian yang diatur melalui Airflow. Kemudian tahap prediksi dilakukan langsung menggunakan BigQuery ML, yang hasilnya disimpan pada tabel di BigQuery dan diperbarui setiap bulan untuk memproyeksikan kebutuhan stok bulan berikutnya. Hasil prediksi pada tabel tersebut dimasukkan ke dalam tabel *data mart* dalam bentuk kolom, dan *data mart* tersebut akan langsung terhubung ke Looker Studio agar laporan dapat diperbarui secara *real-time* tanpa perlu unggahan manual. Pola kerja ini membuat proses prediksi menjadi lebih singkat, konsisten, dan mudah diaudit ketika terjadi perubahan data.



Gambar 3.2 Alur Prediksi Permintaan Barang untuk Perencanaan Inventaris

Dari alur baru yang direncanakan, langkah awal yang perlu dilakukan adalah melakukan asesmen terhadap struktur *data mart* yang akan digunakan sebagai dasar proses prediksi. Tujuannya untuk memastikan seluruh data yang dibutuhkan *user* benar-benar tersedia dan memiliki keterhubungan logis antara satu tabel dengan lainnya. Asesmen ini mencakup identifikasi sumber data dari sistem SAP, *data warehouse* di BigQuery, serta tabel-tabel pendukung yang belum memiliki jadwal *ingestion* harian. Hasil asesmen tersebut kemudian dirangkum dalam Tabel 3.3 yang menjabarkan daftar kolom apa saja yang dibutuhkan dalam *data mart*.

Tabel 3.3 Tabel Daftar Informasi Produk yang Dibutuhkan

No.	Nama Kolom	Deskripsi
1	Code	Nomor produk
2	Article Description	Deskripsi produk

No.	Nama Kolom	Deskripsi
3	Sales Price	Harga jual produk
4	Warehouse Code	Kode toko dan gudang
5	Warehouse Description	Deskripsi toko dan warehouse
6	Insurance Inventory	Minimum parameter di tiap toko atau gudang
7	Max	Maximum parameter di tiap toko atau gudang
8	Stock on Hand	Jumlah stock yang ada di toko atau gudang
9	Sales Last 12 Months	Penjumlahan sales 12 bulan ke belakang
10	Department Code	Kode departemen produk
11	Department Description	Deskripsi nama departemen produk
12	Merchandise Class	Kode kategori produk dalam departemen
13	Merchandise Class Description	Deskripsi kategori article dalam departemen
14	Listed Status	Status produk terdaftar (di-*maintain*) atau tidak di toko
15	Actual Buying Price	Harga beli produk
16	CBM	Satuan volume produk *(Cubic Meter)*
17	Island	Pulau dimana toko berada
18	Area	Kota atau wilayah toko
19	Size	Ukuran toko berdasarkan luasan toko
20	ZMUOM	Kelipatan pengiriman barang per produk
21	Import or Lokal	Kode regular vendor dari suatu produk
22	Average Demand	Prediksi rata-rata penjualan per produk dan toko
23	Total Demand Article	Penjumlahan dari kolom Average Demand per produk

Sebagian besar data sudah tersedia dalam *data warehouse* yang aktif digunakan oleh tim *Data Analytics* di BigQuery. Namun, terdapat beberapa kolom yang belum terintegrasi, seperti “ZMUOM”,

yang bersumber dari tabel khusus di SAP dengan nama yang sama. Tabel ini menyimpan informasi satuan ukuran barang *(Unit of Measurement)* yang menjadi acuan dalam proses transaksi dan perhitungan stok. Karena tabel ini belum diturunkan ke *data lake*, tim perlu menyiapkan *job* harian agar data tersebut dapat ter-*ingest* secara otomatis setiap hari. Selain "ZMUOM", dua kolom lain yaitu "Average Demand" dan "Total Demand Article" belum tersedia dalam data mentah karena berasal dari hasil prediksi yang dijalankan melalui model di BigQuery ML. Kedua kolom ini akan terbentuk setelah model melakukan prediksi terhadap permintaan barang, lalu hasilnya disimpan kembali ke *data mart* sebagai referensi analitik untuk periode berikutnya.

Dalam proses pengembangan model prediksi perencanaan inventaris, *user* menetapkan sejumlah kriteria yang harus dipenuhi agar hasil peramalan yang dihasilkan tetap akurat dan relevan terhadap kondisi bisnis. Kriteria ini berfungsi sebagai filter awal dalam pemilihan data yang akan digunakan oleh model prediksi, sehingga prediksi tidak dilakukan pada data yang tidak representatif atau berpotensi menghasilkan bias. Aturan tersebut juga membantu sistem untuk berfokus pada produk yang benar-benar aktif dan memiliki pola permintaan yang dapat dipelajari oleh model. Adapun kriteria yang ditetapkan oleh *user* meliputi:

1. Produk yang diprediksi hanya produk yang masih aktif dan bersifat *replenishment*, yakni produk yang perlu diisi ulang secara rutin dan teratur.
2. Prediksi hanya diterapkan pada produk yang memiliki penjualan lebih dari 0. Jika pada bulan tertentu suatu produk memiliki jumlah penjualan bernilai 0, maka bulan tersebut tidak akan disertakan dalam proses prediksi.

3. Produk yang tergolong baru dan hanya memiliki data historis selama satu bulan tetap dimasukkan dalam proses peramalan agar model dapat mulai membangun pola awal dari data tersebut.

Kriteria tersebut membantu menjaga efisiensi proses komputasi sekaligus meningkatkan ketepatan model dalam memperkirakan permintaan produk. Pendekatan ini memastikan hasil prediksi tidak hanya relevan dari sisi teknis, tetapi juga mencerminkan kondisi bisnis riil yang dihadapi perusahaan. Seluruh proses analisis kebutuhan bisnis dan sumber data menjadi tahap fundamental dalam pelaksanaan proyek peramalan inventaris dan pengembangan sistem AI katalog produk. Tahap ini memastikan bahwa setiap data yang digunakan telah melalui proses verifikasi, baik dari sisi ketersediaan maupun kesesuaiannya terhadap tujuan proyek. Hasil analisis kemudian dijadikan dasar dalam penyusunan *data mart* yang akan digunakan pada tahap pemodelan berikutnya.

3.3.1.2 Mengumpulkan data dari sumber

Tahap pengumpulan data merupakan langkah setelah proses analisis kebutuhan bisnis selesai dilakukan. Aktivitas ini berfokus pada pengambilan data dari berbagai sumber yang telah diidentifikasi sebelumnya agar dapat digunakan dalam proses pengolahan dan analisis. Kualitas data yang dikumpulkan sangat berpengaruh terhadap hasil akhir proyek, sehingga setiap data harus dipastikan relevan, lengkap, serta memiliki format yang sesuai dengan kebutuhan sistem. Proses pengumpulan ini juga menjadi dasar bagi tahapan transformasi dan integrasi data ke dalam *data warehouse*, sehingga struktur dan hubungan antarvariabel dapat terbentuk secara konsisten.

Pada proyek pengembangan sistem katalog produk, sumber data utama berasal dari platform internal Kawan Lama Group yang menyimpan informasi produk di MongoDB Compass. Basis data ini

bersifat *unstructured*, yaitu tidak memiliki skema atau format tetap sebagaimana tabel pada basis data relasional. Data tersimpan dalam bentuk *document-based*, sehingga setiap entri dapat memiliki atribut berbeda, baik dari sisi jumlah kolom, tipe data, maupun struktur hierarkinya. Situasi tersebut menimbulkan tantangan dalam proses analisis dan integrasi, karena sistem analitik modern seperti *data warehouse* memerlukan format yang konsisten agar dapat diproses secara efisien. Untuk mengatasi hal tersebut, dilakukan proses mengekstrak data, transformasi, dan pemuatan data menggunakan Jupyter Notebook sebagai lingkungan kerja utama.

```
client = MongoClient(MONGO_URI)
db = client[MONGO_DB]
collection = db[MONGO_COLLECTION]
```

```
from pymongo import MongoClient

MONGO_URI = os.getenv('MONGO_URI', 'mongodb://username:pass@host:port')
MONGO_DB = 'db'
MONGO_COLLECTION = 'product_final'

client = MongoClient(MONGO_URI)
db = client[MONGO_DB]
collection = db[MONGO_COLLECTION]
```

```
# get data with condition
dataset = list(collection.find(
        {
            "brand": "MM01"
            , "websiteKey": "website_key"
        }
    )
```

Gambar 3.3 Proses Pengambilan Data dari MongoDB

```
# Define BigQuery schema with STRUCT (RECORD) and nested fields
def get_bq_schema():
    return [
        SchemaField('sku', 'STRING', mode='REQUIRED'),
        SchemaField('name', 'STRING', mode='NULLABLE'),
        SchemaField('short_description', 'STRING', mode='NULLABLE'),
        SchemaField('bom', 'RECORD', mode='NULLABLE', fields = [
            SchemaField('sku', 'STRING', mode='NULLABLE'),
            SchemaField('bill_of_material', 'STRING', mode='NULLABLE'),
            SchemaField('unit_of_measure', 'STRING', mode='NULLABLE'),
            SchemaField('unit_article', 'RECORD', mode='REPEATED', fields = [
                SchemaField('component', 'STRING', mode='NULLABLE'),
                SchemaField('qty', 'INTEGER', mode='NULLABLE'),
                SchemaField('uom', 'STRING', mode='NULLABLE'),
            ]),
            SchemaField('unit_price', 'RECORD', mode='REPEATED', fields = [
                SchemaField('component', 'STRING', mode="NULLABLE"),
                SchemaField('component_level', 'STRING', mode="NULLABLE"),
                SchemaField('rate', 'FLOAT', mode="NULLABLE"),
            ]),
        ]),
```

Gambar 3.4 Pembentukan Struktur Skema Data untuk BigQuery

Proses diawali dengan pengambilan data dari MongoDB untuk kemudian diolah lebih lanjut. Langkah awal dilakukan pada Gambar 3.3 dengan menghubungkan program ke basis data dan koleksi yang diinginkan menggunakan alamat server yang telah ditentukan. Setelah koneksi terbentuk, sistem mengambil data dengan kriteria tertentu, misalnya hanya memilih dokumen dengan merek dan sumber situs tertentu. Data yang berhasil diambil kemudian dikonversi menjadi bentuk daftar agar lebih mudah diolah di tahap berikutnya.

Proses selanjutnya adalah pembentukan struktur data yang diperlukan agar informasi dari MongoDB dapat dimasukkan ke data lake di BigQuery, yang memiliki sifat terstruktur. Karena data dari MongoDB bersifat tidak terstruktur, maka perlu dibuat panduan atau skema yang menjelaskan bentuk data, nama kolom, dan jenis nilai yang dapat disimpan di masing-masing kolom tersebut. Pada Gambar 3.4, fungsi get_bq_schema() berfungsi mengembalikan daftar SchemaField yang menggambarkan kolom, tipe data, dan mode (REQUIRED atau NULLABLE). Dalam struktur ini juga diatur bagaimana data bersarang *(nested)*, misalnya ketika satu produk memiliki daftar komponen atau harga yang berbeda. Tahapan ini penting untuk menjaga konsistensi dan keterbacaan data saat dipindahkan ke lingkungan analisis yang membutuhkan format yang lebih terstruktur. Dengan adanya skema ini, data yang awalnya acak dari MongoDB dapat disesuaikan agar kompatibel dengan tabel BigQuery.

```
# Export to jsonl

import jsonlines
list_schema = get_bq_schema()
list_schema_name = [x.name for x in list_schema]

import datetime
dataset_filtered = []

for data in dataset:
    temp = {k: v.isoformat() if isinstance(v, datetime.datetime) else v for k, v in data.items() if k in list_schema_name}
    if 'bom' in temp and isinstance(temp['bom'], dict):
        temp['bom'] = {k: v.isoformat() if isinstance(v, datetime.datetime) else v for k, v in temp['bom'].items() if k != '_id'}

    dataset_filtered.append(temp)
```

Gambar 3.5 Transformasi dan Penyesuaian Format Dataset

```
import jsonlines
# write jsonl file (can be find in the same folder as the ipynb file)
with jsonlines.open('dataset_filtered.jsonl', 'w') as writer:
    writer.write_all(dataset_filtered)
```

```
job_config = bigquery.LoadJobConfig(
    schema=get_bq_schema(),
    source_format=bigquery.SourceFormat.NEWLINE_DELIMITED_JSON,
    ignore_unknown_values=True,
    # write_disposition=bigquery.WriteDisposition.WRITE_TRUNCATE
)
```

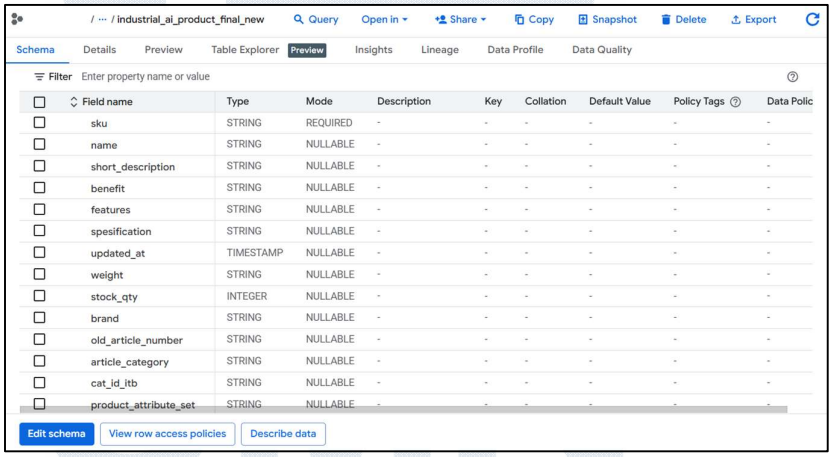
```
# Load data from GCS
client = bigquery.Client(project=BQ_PROJECT)
gcs_uri = 'gs://bucket/dataset_filtered.jsonl'
table_id = f"{BQ_PROJECT}.{BQ_DATASET}.{BQ_TABLE}"
job = client.load_table_from_uri(gcs_uri, table_id, job_config=job_config)
job.result()
```

Gambar 3.6 Penyimpanan dan Pemuatan Data Terstruktur ke BigQuery

Gambar 3.5 menunjukkan proses transformasi data agar sesuai dengan struktur yang telah ditentukan sebelumnya. Data mentah hasil pengambilan dari MongoDB diperiksa dan disesuaikan dengan format yang lebih seragam dengan memanggil fungsi get_bq_schema() untuk mendapatkan skema yang telah didefinisikan sebelumnya. Selanjutnya, dilakukan iterasi terhadap setiap dokumen dalam dataset, di mana setiap elemen data diproses untuk memastikan nilai bertipe datetime dikonversi menjadi format ISO standar menggunakan isoformat(). Jika terdapat atribut bersarang seperti bom, maka struktur di dalamnya juga ditransformasi dengan cara yang sama agar tetap konsisten dengan format BigQuery. Setiap data yang telah difilter kemudian ditambahkan ke dalam list dataset_filtered untuk tahap ekspor berikutnya. Proses ini memastikan bahwa data yang awalnya tidak terstruktur dari MongoDB diubah menjadi format terstruktur dan kompatibel untuk analisis di platform data warehouse seperti BigQuery.

Tahap terakhir adalah penyimpanan dan pemuatan data hasil transformasi ke BigQuery. Pada Gambar 3.6, data yang telah disusun rapi disimpan terlebih dahulu dalam *file* berformat JSON Lines, yang cocok untuk menyimpan data dalam jumlah besar karena setiap barisnya mewakili satu entri. Setelah *file* disimpan ke Google Cloud Storage, tahap berikutnya adalah mendefinisikan konfigurasi

pemuatan menggunakan bigquery.LoadJobConfig, yang menentukan skema data, format sumber *file*, serta opsi untuk mengabaikan *field* yang tidak dikenal agar proses impor tetap berjalan lancar. Kemudian, *file* JSON dimuat ke dalam tabel di BigQuery agar dapat digunakan untuk analisis. Selama proses pemuatan, sistem akan mengikuti struktur data yang telah ditentukan sebelumnya agar formatnya tetap konsisten. Langkah ini memastikan seluruh data dari MongoDB telah berubah menjadi bentuk yang terstruktur, mudah diakses, dan siap digunakan untuk kebutuhan analisis atau pelaporan di platform *data warehouse*. Hasil pengumpulan data dari MongoDB terdapat pada Gambar 3.7, di mana data telah melewati proses *Extract*, *Transform*, dan *Load* (ETL) yang sudah berbentuk tabel dengan skema.



Gambar 3.7 Tabel Hasil Proses ETL Sumber Data Katalog Produk

Selain proses ETL yang bersifat khusus seperti pada pengolahan data dari MongoDB, terdapat pula proses ETL yang bersifat rutin dan menjadi bagian dari *job* harian tim *Data Engineer*. Proses ini digunakan untuk menjaga agar data operasional dari berbagai sistem sumber selalu mutakhir di *data lake*, *data warehouse* maupun *data mart*. Berbeda dengan pengolahan data yang dilakukan secara manual atau insidental, *job* harian berjalan otomatis sesuai jadwal yang telah ditentukan melalui orkestrasi sistem Apache

Airflow. Tujuannya adalah memastikan sinkronisasi data dari sistem transaksional ke *data lake* berlangsung stabil, terukur, dan efisien. Mekanisme ini juga meminimalkan intervensi manual, sekaligus memastikan seluruh data yang digunakan untuk analisis maupun model prediksi selalu berada dalam kondisi terbaru. Salah satu contoh implementasi proses ETL harian ini terlihat pada proyek perencanaan inventaris yang menggunakan sumber data dari SAP.

Proses pengumpulan data dari sumber SAP untuk proyek perencanaan inventaris dilakukan untuk tabel ZMUOM yang berisi informasi terkait *Unit of Measurement* (UoM). Tabel ini memiliki peran penting karena berfungsi sebagai acuan konversi satuan produk di seluruh rantai pasok, misalnya antara satuan pembelian, penyimpanan, dan penjualan. Namun, pada tahap awal pengembangan sistem, data dari tabel ini belum tersedia di *data lake* dan belum memiliki proses otomatis untuk diperbarui setiap hari. Oleh karena itu, langkah awal yang perlu dilakukan adalah menginisiasi konfigurasi untuk membangun *job* harian yang dapat menurunkan dan memproses data ZMUOM ke *data lake* di BigQuery.

	A	B
1	header_uom	dtype
2	article_no	str
3	article_description	str
4	article_hierarchy_code	str
5	article_hierarchy_description	str
6	main_ah_assignment	str
7	uom	str
8	denominator	float64
9	numerator	float64
10	based_uom	str
11	order_uom	str
12	delivery_issue_uom	str
13	sales_uom	str
14	gtin	str

Gambar 3.8 *File* Metadata Tabel ZMUOM

Proses inisiasi konfigurasi dilakukan melalui repositori Airflow dengan menyiapkan sebuah *file* Excel yang berisi struktur

tabel ZMUOM, mencakup nama kolom, tipe data, dan keterangan tambahan yang diperlukan untuk proses pemetaan, yang ditunjukkan pada Gambar 3.8. *File* ini bertujuan agar sistem *pipeline* dapat membaca struktur tabel secara dinamis tanpa harus menulis ulang definisi skema di kode program. Setelah *file* konfigurasi selesai disusun, selanjutnya dilakukan penambahan informasi tabel ke dalam *file* konfigurasi YAML, yaki pendefinisian beberapa parameter penting seperti nama tabel, *prefix* nama *file* hasil ekstraksi, primary key, dan status partisi (apakah tabel tersebut menggunakan partisi atau tidak). *File* YAML ini berperan sebagai acuan bagi Airflow untuk mengetahui bagaimana data harus diproses di setiap tahap *pipeline*.

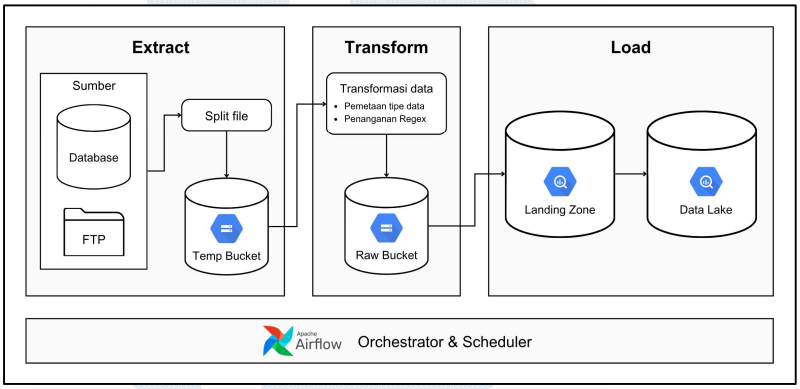
```yaml
daily:
  sap:
    product_management:
      article:
        tables:
          - UOM:
              partition: true
              lz_config: {}
              dl_config:
                primary_key: [article_no|trim, uom]
                trim_column:
                  leading_zero: [article_no]
```

Gambar 3.9 *File* Konfigurasi Tabel ZMUOM

Penambahan konfigurasi sistem pada Gambar 3.9 akan mengenali bahwa tabel ZMUOM termasuk dalam skop eksekusi rutin yang dijalankan oleh Airflow *scheduler*. Proses konfigurasi ini tidak terlalu kompleks, karena tim *Data Engineer* Kawan Lama Group telah membangun arsitektur *pipeline* dan backend yang stabil serta terdokumentasi dengan baik. Infrastruktur tersebut membuat proses integrasi data baru hanya memerlukan penyesuaian kecil pada level konfigurasi, tanpa perlu mengubah alur utama *pipeline*.

Gambar 3.10 memperlihatkan alur bagaimana data dari SAP ditransfer dan diproses hingga akhirnya tersimpan di data lake BigQuery. Berdasarkan ilustrasi tersebut, alur dimulai dari proses ekstraksi data dari sumber, tepatnya melalui *File Transfer Protocol* (FTP), di mana sistem SAP mengirimkan *file* hasil ekstraksi ke lokasi

yang telah ditentukan. Setelah *file* diterima, Airflow menjalankan *task* untuk melakukan ekstraksi dan pembagian *(split) file* menjadi ukuran lebih kecil agar proses pemrosesan dan pemuatan data ke sistem berikutnya dapat berjalan lebih efisien dan tidak membebani sumber daya komputasi. *File* hasil pemisahan kemudian disimpan sementara di Google Cloud Storage (GCS), yang berfungsi sebagai area transit sebelum data diproses lebih lanjut.



Gambar 3.10 Penyimpanan dan Pemuatan Data Terstruktur ke BigQuery

Tahap berikutnya adalah proses transformasi. Pada tahap ini, sistem membaca *file* Excel yang telah dibuat sebelumnya untuk melakukan pemetaan antara kolom sumber dan kolom tujuan, serta menyesuaikan tipe data agar konsisten dengan struktur di *data lake*. Selain penyesuaian tipe data, dilakukan pula proses pembersihan data seperti penanganan karakter khusus atau *regular expression* agar format data menjadi bersih dan terstandardisasi. Setelah transformasi selesai, hasilnya kembali disimpan ke GCS, kali ini dalam format yang telah siap untuk dimuat ke BigQuery.

Proses berikutnya adalah pemuatan data ke tahap *landing zone* di BigQuery. *Landing zone* berfungsi sebagai area sementara untuk menampung data hasil ekstraksi yang belum digabungkan dengan data sebelumnya. Sistem ini dirancang hanya untuk menyimpan perubahan delta atau hasil ekstraksi terbaru, sehingga jika dilakukan ekstraksi

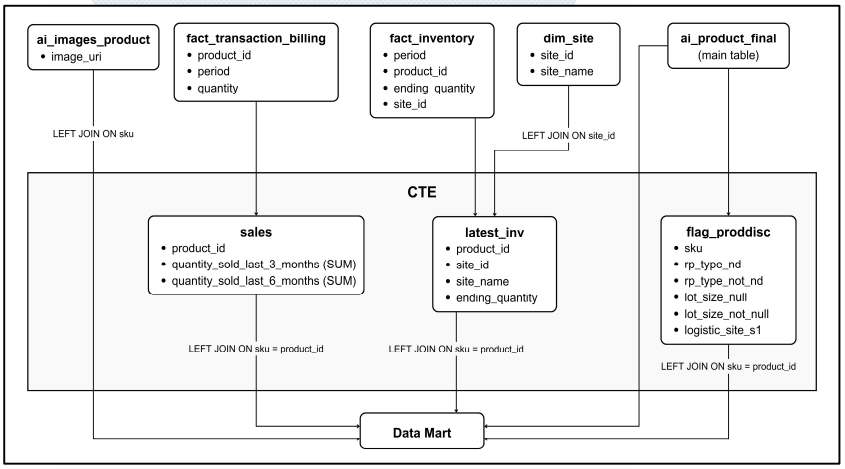
ulang, data sebelumnya akan otomatis tergantikan. Setelah berada di *landing zone*, data kemudian dimuat ke tahap *data lake*, yaitu tempat penyimpanan utama yang menyimpan seluruh riwayat data dan siap digunakan untuk analisis maupun kebutuhan lanjutan. Proses pemuatan ini menggunakan metode *update-insert* atau *merge*, di mana sistem membandingkan data baru dengan data yang sudah ada berdasarkan kolom kunci. Jika ditemukan baris data dengan kunci yang sama, maka nilai tersebut akan diperbarui. Namun, jika tidak ditemukan kecocokan, maka data akan ditambahkan sebagai baris baru. Informasi mengenai kolom kunci dan konfigurasi tabel ini diambil dari *file* YAML yang telah dibuat sebelumnya, sehingga seluruh proses berjalan secara otomatis dan konsisten.

Pengumpulan data pada kedua proyek ini memiliki pendekatan yang berbeda sesuai dengan kebutuhan masing-masing. Pada proyek katalog produk, tantangan utama terletak pada pengelolaan data yang tidak terstruktur. Data tersebut perlu ditata ulang dan disesuaikan formatnya supaya bisa diproses dengan baik. Hal ini memerlukan seleksi atribut yang tepat dan penyesuaian struktur data untuk keperluan analisis. Di sisi lain, proyek perencanaan inventaris lebih mengutamakan otomatisasi dalam pengambilan data dari sistem sumber. *Job* harian harus berjalan stabil karena data ini dipakai langsung untuk menghitung kebutuhan permintaan barang. Kedua proses ini saling mendukung untuk membangun fondasi data yang solid, di mana setiap data yang masuk sudah melewati tahap validasi dan transformasi sesuai kebutuhan perusahaan.

3.3.1.3 Membuat *data mart*

Data yang telah berhasil diturunkan dari sumber umumnya masih bersifat mentah dan tersebar di berbagai tabel, sehingga belum siap digunakan secara langsung untuk analisis maupun pengembangan model. Hal ini disebabkan oleh tidak semua kolom memiliki nilai

yang relevan dengan kebutuhan proyek, serta hubungan antar tabel sering kali tidak memberikan konteks yang cukup untuk menghasilkan informasi yang bermakna. Untuk menjembatani hal tersebut, diperlukan pembangunan *data mart* sebagai tempat penyimpanan yang menyatukan data dari berbagai sumber ke dalam struktur yang lebih terfokus pada kebutuhan analisis tertentu. *Data mart* pada dasarnya merupakan *subset* dari *data warehouse* yang dirancang agar lebih ringkas, mudah diakses, dan efisien digunakan oleh tim analitik dalam menjawab kebutuhan bisnis spesifik. Melalui pendekatan ini, proses pengambilan keputusan menjadi lebih cepat karena data yang tersaji sudah terfilter, terintegrasi, dan memiliki skema yang konsisten dengan konteks permasalahan yang sedang dikerjakan.



Gambar 3.11 Diagram Visualisasi *Query Data Mart* Katalog Produk

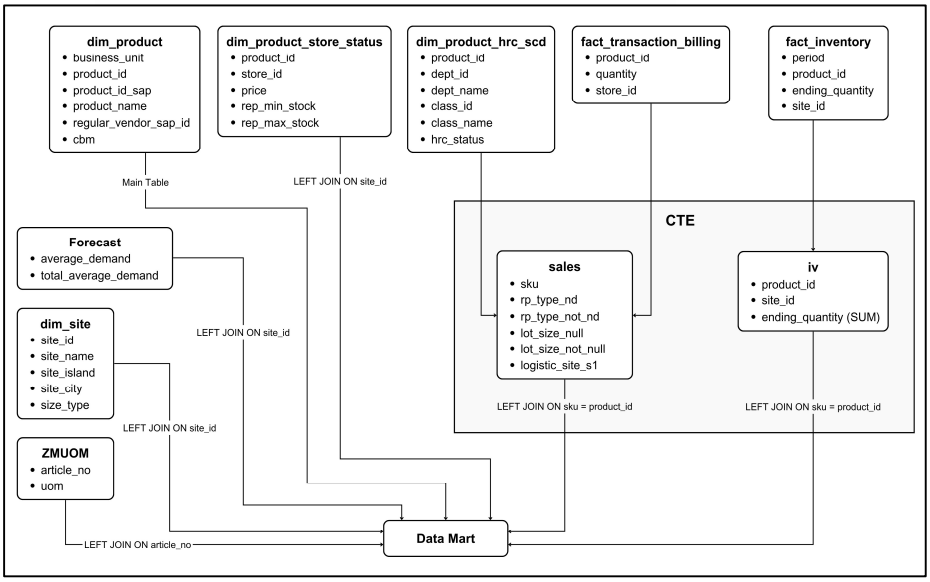
Gambar 3.11 menunjukkan diagram visualisasi *query* yang dirancang untuk membuat *data mart* yang menyatukan seluruh informasi penting tentang produk industri, mulai dari riwayat penjualan, tingkat stok saat ini, hingga status produk apakah masih dijual secara aktif atau sudah dihentikan. *Query* dimulai dengan menarik data penjualan dari tabel fakta yang menyimpan informasi transaksi (fact_transaction_billing), yang dibuat dalam *Common*

Table Expression (CTE) agar lebih terstruktur dan mudah dibaca. Untuk setiap produk, query menghitung total unit yang terjual dalam periode 3 bulan terakhir dan 6 bulan terakhir. Data ini sangat berguna karena memberikan gambaran jelas mengenai produk mana yang populer di kalangan pelanggan, yang nantinya akan digunakan sebagai faktor sortir dalam AI katalog produk. Selanjutnya, terdapat CTE "latest_inv" untuk menganalisis tingkat persediaan dengan menelusuri catatan inventaris untuk menemukan jumlah stok terkini dari setiap produk di setiap gudang atau lokasi yang tersimpan dalam tabel fact_inventory. CTE ini melacak secara detail di mana setiap produk berada dan berapa banyak stok yang tersedia di masing-masing tempat. Pada CTE terakhir, yakni "flag_proddisc", dilakukan proses identifikasi untuk menandai apakah suatu produk telah dihentikan atau tidak. Proses ini memerlukan analisis terhadap berbagai informasi yang saling berkaitan dengan memeriksa kode status penjualan, jenis pengisian ulang, ukuran lot, serta konfigurasi logistik. Sebuah produk dapat dikategorikan sebagai produk yang dihentikan jika memiliki kode status penjualan tertentu dan pengaturan inventaris tertentu, atau jika tidak memiliki status penjualan namun memiliki konfigurasi gudang spesifik.

```
SELECT
  ai.* EXCEPT(site_listed, stock_qty, warranty),
  s.qty_sold_last_3_months,
  s.qty_sold_last_6_months,
  inv.stock_site,
  -- parse/translate warranty code to separate columns (in struct)
  STRUCT(
    CASE
      WHEN SPLIT(TRIM(ai.warranty), '-')[SAFE_OFFSET(1)] IS NULL THEN NULL
      WHEN SUBSTR(SPLIT(TRIM(ai.warranty), '-')[SAFE_OFFSET(1)], 4) = '000' THEN NULL
      ELSE CAST(REPLACE(SUBSTR(SPLIT(TRIM(ai.warranty), '-')[SAFE_OFFSET(1)], 4), 'M', '') AS INT64) || ' Month'
    END AS warranty_unit,
    CASE
      WHEN SPLIT(TRIM(ai.warranty), '-')[SAFE_OFFSET(2)] IS NULL THEN NULL
      WHEN SUBSTR(SPLIT(TRIM(ai.warranty), '-')[SAFE_OFFSET(2)], 4) = '000' THEN NULL
      ELSE CAST(REPLACE(SUBSTR(SPLIT(TRIM(ai.warranty), '-')[SAFE_OFFSET(2)], 4), 'M', '') AS INT64) || ' Month'
    END AS warranty_parts,
    CASE
      WHEN SPLIT(TRIM(ai.warranty), '-')[SAFE_OFFSET(3)] IS NULL THEN NULL
      WHEN SUBSTR(SPLIT(TRIM(ai.warranty), '-')[SAFE_OFFSET(3)], 4) = '000' THEN NULL
      ELSE CAST(REPLACE(SUBSTR(SPLIT(TRIM(ai.warranty), '-')[SAFE_OFFSET(3)], 4), 'M', '') AS INT64) || ' Month'
    END AS warranty_service
  ) AS warranty,
  CASE
    WHEN ai.client_sales_block is null and pd.sales_status_z1 = TRUE and pd.rp_type_nd = TRUE and pd.lot_size_null = TRUE and pd.logistic_sit
    OR ai.client_sales_block is null and pd.sales_status_z1 = TRUE and pd.rp_type_not_nd = TRUE and pd.lot_size_not_null = TRUE and pd.logist
= TRUE
    OR ai.client_sales_block = 'Z1' and pd.sales_status_null_or_empty = TRUE and pd.rp_type_nd = TRUE and pd.lot_size_null = TRUE and pd.
logistic_site_s001 = TRUE
    THEN 'yes' else 'no'
  END AS is_discontinue
FROM `project.dl.industrial_ai_product_final_new` ai
LEFT JOIN sales s ON ai.sku = s.product_id
LEFT JOIN agg_stock inv ON ai.sku = inv.product_id
LEFT JOIN flag_proddisc pd ON ai.sku = pd.sku
```

Gambar 3.12 *Query Data Mart* Katalog Produk

Hasil akhir dari *query* ini menggabungkan seluruh elemen tersebut, yakni informasi dasar produk, angka penjualan untuk berbagai periode waktu, detail lokasi stok yang terperinci, informasi garansi, status penghentian produk, hingga gambar produk. Informasi garansi pada bagian ini diuraikan dan dipecah menjadi tiga komponen, yaitu garansi untuk unit produk itu sendiri, garansi untuk suku cadang, dan garansi untuk layanan. Produk-produk sebelumnya memiliki kode garansi yang pada awalnya berupa kode, sekarang telah dikonversi ke dalam format yang lebih mudah dibaca seperti "6 Bulan" atau "12 Bulan", sehingga pengguna dapat dengan cepat memahami jenis cakupan garansi yang ditawarkan oleh masing-masing produk. *Data mart* ini memberikan rangkuman yang dapat digunakan oleh pengguna bisnis untuk membuat keputusan strategis terkait perencanaan inventaris, pengadaan barang, dan manajemen siklus hidup produk. Gambar 3.12 menunjukkan sepotong *query* untuk membuat *data mart* katalog produk, yang disimpan pada sebuah *view* agar data yang disimpan selalu yang terbaru ketika dijalankan.



Gambar 3.13 Diagram Visualisasi *Query Data Mart* Perencanaan Inventaris

Pembuatan tabel *data mart* selanjutnya adalah untuk keperluan perencanaan inventaris, di mana tabel ini berfokus pada pembuatan laporan master produk yang terperinci, dengan menggabungkan informasi produk, jumlah penjualan, tingkat inventaris, dan hasil prediksi dalam satu tabel yang terkonsolidasi. Visualisasi *query data mart* ini ditunjukkan pada Gambar 3.13, di mana terdapat CTE pertama, yaitu "sales", untuk mendapatkan data penjualan dengan menelusuri data 12 bulan terakhir dan menjumlahkan total unit setiap produk yang terjual di masing-masing lokasi toko. Proses ini juga mengecualikan lokasi yang merupakan gudang, menyaring kelas produk dan departemen spesifik yang tidak perlu disertakan, serta menghapus transaksi yang bersifat *buy back*, atau transaksi yang bukan penjualan reguler. Adanya filter tersebut berfungsi untuk memastikan bahwa angka penjualan yang dihasilkan mencerminkan permintaan pelanggan aktual, bukan pergerakan internal atau situasi khusus lainnya. Kemudian terdapat CTE lain bernama "iv", yang berisikan perhitungan inventaris dengan melihat stok penutupan kemarin untuk menunjukkan dengan tepat berapa banyak unit setiap produk yang saat ini tersedia di setiap gudang. Data ini bertujuan untuk memberikan gambaran akurat tentang posisi inventaris terkini perusahaan.

Dari kedua CTE tersebut, dibuat *query* utama kemudian mengintegrasikan seluruh komponen tersebut dengan tabel dim_product sebagai tabel utama. Pada *query* ini juga dilakukan penggabungan dari tabel-tabel dimensi lain untuk membuat kolom sesuai dengan kebutuhan bisnis. Untuk setiap produk di setiap gudang, laporan menampilkan kode dan deskripsi produk, harga jual saat ini, detail gudang termasuk lokasi seperti pulau dan kota, tingkat stok minimum dan maksimum yang telah ditetapkan untuk keperluan perencanaan inventaris, stok aktual saat ini, serta total penjualan selama setahun terakhir. *Query* juga memperkaya data ini dengan

informasi klasifikasi, seperti departemen dan kelas barang dagangan yang dimiliki produk tersebut, yang sangat membantu dalam pengorganisasian dan analisis data. Terdapat juga status apakah produk "Terdaftar" atau "Tidak Terdaftar" untuk dijual, yang merupakan informasi penting untuk memahami produk mana yang secara aktif tersedia bagi pelanggan. Detail tambahan yang disertakan mencakup harga pembelian aktual yang dibayarkan perusahaan untuk produk tersebut, ukuran fisik produk dalam meter kubik (CBM) yang sangat penting untuk perencanaan ruang gudang, unit pengukuran, serta informasi apakah produk tersebut bersumber dari lokal atau impor. Kode vendor dianalisis untuk secara otomatis mengkategorikan produk dibeli secara lokal, diimpor, atau dari sumber lain.

```
SELECT
  p.product_id_sap AS code,
  p.product_name AS article_description,
  dps.price AS sales_price,
  dps.store_id AS warehouse_code,
  ds.site_name AS warehouse_description,
  dps.rep_min_stock AS insurance_inventory,
  dps.rep_max_stock AS max,
  iv.stock_on_hand,
  s.sales_last_12_months,
  dphs.dept_id AS department_code,
  dphs.dept_name AS department_description,
  dphs.class_id AS merchandise_class,
  dphs.class_name AS merchandise_class_description,
  CASE WHEN product_status_id not in (       ) OR product_status_id is null THEN 'Listed' ELSE 'Not Listed' END AS listed_status,
  map.moving_price as actual_buying_price,
  p.cbm,
  ds.site_island AS island,
  ds.site_city AS city,
  CASE WHEN ds.size_type = 'nan' THEN NULL ELSE ds.size_type END AS size,
  zm.uom,
  CASE WHEN p.regular_vendor_sap_id like       THEN
    WHEN p.regular_vendor_sap_id like       or p.regular_vendor_sap_id like         THEN
    WHEN p.regular_vendor_sap_id like         THEN
    WHEN p.regular_vendor_sap_id like         THEN
    ELSE 'Other' END AS import_or_local
FROM product p
LEFT JOIN project.klg_dwh.dim_product_store_status dps on p.product_id_sap = dps.product_id and dps.business_unit = 'BU'
LEFT JOIN project.klg_dwh.dim_site ds on dps.store_id = ds.site_id and ds.business_unit = 'BU'
LEFT JOIN (
  select site_id, product_id, sum(ending_qty) AS stock_on_hand from project.klg_dwh.bu_fact_inventory
  where period = DATE_SUB(CURRENT_DATE(), INTERVAL 1 DAY)
  group by 1,2
) iv on p.product_id_sap = iv.product_id and iv.site_id = ds.site_id
LEFT JOIN sales s on p.product_id_sap = s.product_id and ds.site_id = s.store_id
LEFT JOIN project.klg_dwh.dim_product_hrc_scd dphs on p.product_id_sap = dphs.product_id_sap and hrc_status = 'active' and dphs.bus
LEFT JOIN (
  select * from project.dl.article_uom
  qualify row_number() over (partition by article_no order by nullif(delivery_issue_uom,'nan') desc, nullif(order_uom,'nan') desc,
) zm on ltrim(zm.article_no,'0') = p.product_id_sap and coalesce(nullif(delivery_issue_uom,'nan'), order_uom) = 'X'
LEFT JOIN project.dl.bu_unitprice_map map on p.product_id_sap = map.article_code and ds.site_id = map.valuation_area_site
```

Gambar 3.14 *Query Data Mart* Katalog Produk

Diagram visualisasi *data mart* pada Gambar 3.13 direalisasikan menjadi *query* pada Gambar 3.14. Hasil akhir disaring untuk hanya memasukkan produk yang memiliki tingkat stok minimum yang sudah ditentukan dan mengecualikan kode gudang tertentu, sehingga menghasilkan kumpulan data yang terfokus dan

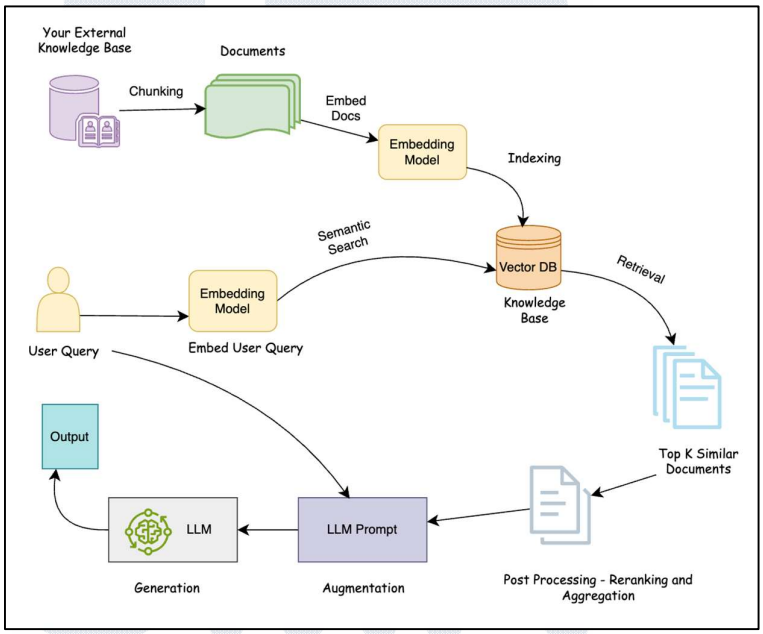
siap digunakan untuk perencanaan serta analisis inventaris. Laporan ini pada dasarnya menyediakan semua informasi yang dibutuhkan oleh *merchandiser* dan manajer inventaris dalam satu tempat untuk membuat keputusan terkait pemesanan, tingkat stok, dan keragaman produk di setiap lokasi operasional.

Penerapan *data mart* dalam kedua proyek memiliki keunggulannya masing-masing. Pada proyek katalog produk, *data mart* membantu melakukan penyusunan struktur informasi yang memuat atribut produk yang relevan dengan sistem rekomendasi berbasis AI, tanpa harus menelusuri ulang data mentah dari MongoDB setiap kali dibutuhkan. Sementara itu, dalam proyek perencanaan inventaris, *data mart* berperan sebagai fondasi untuk menyimpan hasil prediksi serta metrik penjualan dan stok secara terpusat. Hal ini tidak hanya mempercepat proses analisis, tetapi juga memastikan konsistensi antar laporan yang dihasilkan dari berbagai sistem. Dengan kata lain, *data mart* berfungsi sebagai penghubung antara data mentah dan sistem analitik, menjadikan alur kerja tim data lebih efisien, terukur, dan berorientasi pada kebutuhan bisnis yang nyata.

3.3.1.4 Mempelajari konsep AI generatif menggunakan LangChain dan LangGraph

Optimasi *Artificial Intelligence* (AI) katalog produk perlu adanya pemahaman konsep mengenai AI generatif. Perkembangan AI generatif telah mengalami peningkatan signifikan sejak munculnya *Large Language Model* (LLM) yang mampu memproses dan menghasilkan bahasa alami secara kontekstual. LLM seperti ChatGPT, LLaMA, atau Gemini dikembangkan untuk memahami hubungan semantik antar kata dan menyusun kalimat yang menyerupai gaya berbahasa manusia. Model tersebut dilatih pada jumlah data teks yang sangat besar sehingga mampu menjawab pertanyaan, merangkum dokumen, hingga menghasilkan konten baru

berdasarkan perintah atau *prompt* yang diberikan. Namun, meskipun memiliki kemampuan bahasa yang luas, LLM memiliki keterbatasan dalam mengakses informasi terkini atau spesifik, karena model hanya dapat mengandalkan data yang digunakan saat pelatihan. Kondisi tersebut menjadi alasan utama dikembangkannya pendekatan *Retrieval-Augmented Generation* (RAG), yang bertujuan meningkatkan akurasi dan relevansi hasil keluaran model melalui integrasi antara kemampuan generatif LLM dan sistem pencarian berbasis data eksternal.



Gambar 3.15 Diagram Alur Sistem RAG
Sumber: [19]

Sistem RAG bekerja melalui dua tahap utama, yakni tahap persiapan data dan tahap pemrosesan *query* pengguna, yang digambarkan pada Gambar 3.15. Pada tahap persiapan data, sistem dimulai dengan mengambil informasi dari basis pengetahuan eksternal. Dokumen-dokumen tersebut kemudian dipecah menjadi bagian-bagian yang lebih kecil melalui proses *chunking*. Setelah itu, potongan-potongan dokumen ini diubah menjadi representasi vektor menggunakan model *embedding*. Hasil *embedding* kemudian

disimpan dan diindeks dalam basis data vektor yang berfungsi sebagai basis pengetahuan sistem. Ketika pengguna mengajukan pertanyaan, *query* tersebut diproses secara paralel dalam dua jalur. Pertama, *query* pengguna diubah menjadi vektor menggunakan model *embedding* yang sama. Vektor *query* tersebut kemudian digunakan untuk melakukan pencarian semantik di basis data vektor guna menemukan dokumen-dokumen yang paling relevan. Sistem akan mengambil beberapa dokumen teratas yang memiliki kesamaan paling tinggi dengan *query*. Dokumen-dokumen hasil pencarian kemudian melalui tahap *post-processing*, yaitu *reranking* dan agregasi, untuk memastikan relevansi dan kualitasnya. Hasil ini digabungkan dengan *query* pengguna asli untuk membentuk *prompt* yang lengkap dan kontekstual. Pada tahap akhir, *prompt* yang telah diperkaya dengan konteks relevan dikirim ke LLM. LLM kemudian menghasilkan *output* berupa jawaban yang akurat berdasarkan kombinasi antara pemahaman bahasanya dan informasi faktual dari dokumen-dokumen yang diambil. *Output* inilah yang akhirnya diberikan kepada pengguna sebagai respons atas pertanyaan mereka. Dalam proyek katalog produk, penerapan RAG berfungsi agar sistem AI dapat memberikan rekomendasi atau informasi produk yang akurat berdasarkan data internal perusahaan, bukan hanya dari hasil prediksi model bahasa umum.

Salah satu kerangka kerja yang banyak digunakan untuk membangun sistem RAG adalah LangChain. Kerangka kerja ini dirancang untuk mempermudah penggabungan antara LLM dan sumber data eksternal, sehingga pengembang dapat membuat aplikasi berbasis AI generatif yang memiliki konteks domain tertentu. Meskipun LangChain menjadi salah satu kerangka kerja yang populer untuk membangun sistem berbasis LLM, terdapat sejumlah keterbatasan yang sering muncul dalam penerapannya. Salah satu kelemahan utamanya adalah sifat alur kerja yang berjalan secara lurus

dan satu arah. Setiap langkah atau *chain* pada LangChain dijalankan secara berurutan tanpa kemampuan untuk menyesuaikan diri secara dinamis terhadap hasil keluaran sebelumnya. Artinya ketika sistem menghadapi kondisi yang memerlukan evaluasi ulang, percabangan keputusan, atau proses interaksi dua arah antara model dan data, LangChain tidak mampu menanganinya secara efisien tanpa penulisan logika tambahan yang rumit. Selain itu, LangChain belum memiliki mekanisme bawaan untuk mempertahankan konteks proses dalam skala besar, sehingga setiap interaksi sering kali berdiri sendiri tanpa keterkaitan yang kuat antar langkah pelaksanaan. Keterbatasan tersebut menimbulkan kebutuhan akan kerangka kerja yang lebih lentur dan adaptif dalam menangani interaksi kompleks antar komponen dalam sistem AI generatif.

Keterbatasan LangChain diatasi dengan pengembangan kerangka kerja LangGraph, yaitu kerangka kerja berbasis pengelolaan *state* graf yang dirancang untuk membangun alur kerja LLM yang bersifat dinamis dan dua arah. Berbeda dengan LangChain yang menjalankan *chain* secara berurutan, setiap langkah atau komponen dalam sistem LangGraph berinteraksi melalui hubungan antar *nodes* dan *edges*, mirip dengan alur graf. Kerangka kerja ini mendukung model untuk melakukan percabangan logika, pengulangan umpan balik, dan penyesuaian hasil berdasarkan kondisi yang muncul selama proses berlangsung. Struktur utamanya disebut StateGraph, yang berfungsi sebagai peta alur kerja di mana setiap *node* mewakili unit tugas seperti pelaksanaan prompt atau pengambilan data, sedangkan edge menunjukkan arah aliran informasi antar *node*.

Konsep *state* dalam LangGraph menjadi elemen inti yang menyimpan dan memperbarui konteks selama proses berjalan. Fungsi *state* adalah menjadikan sistem dapat melacak apa yang sudah dilakukan, hasil yang diperoleh, dan langkah berikutnya yang harus

diambil tanpa kehilangan konteks. Selain itu, LangGraph mendukung penggunaan *tools*, yaitu komponen eksternal atau fungsi tambahan yang dapat dipanggil oleh sistem untuk melakukan tindakan spesifik, seperti mengambil data dari API, menjalankan perhitungan, atau memproses *file*. Pemanfaatan *tools* diaplikasikan ke sebuah metode utama LangGraph yang terstruktur dalam pengambilan keputusan yang disebut sebagai *Reasoning and Action* (ReAct). ReAct merupakan sebuah metode yang menggabungkan kemampuan penalaran logis dengan tindakan langsung. Metode ini membuat sistem dapat berpikir terlebih dahulu sebelum mengambil keputusan, lalu mengeksekusi tindakan berdasarkan hasil pemikiran tersebut sehingga menjadikannya lebih efisien dalam menangani permasalahan kompleks yang memerlukan interaksi berulang antara pemahaman dan eksekusi.

Keunggulan utama LangGraph terletak pada kemampuannya dalam menciptakan alur kerja yang lebih adaptif, kontekstual, dan efisien untuk sistem berbasis LLM. Struktur graf yang digunakan memungkinkan pengembang membangun sistem yang tidak hanya dapat bereaksi terhadap input, tetapi juga mampu memproses skenario yang bercabang. Selain itu, penggabungan antara pengelolaan *state* dan pendekatan ReAct memberikan kestabilan proses tanpa kehilangan konteks, bahkan saat sistem menghadapi alur yang rumit. Fleksibilitas ini menjadikan LangGraph sebagai solusi yang lebih ideal untuk pengembangan sistem AI generatif berskala besar dan dinamis. Tabel 3.4 memperlihatkan perbandingan antara LangChain dan LangGraph berdasarkan karakteristik utama keduanya.

Tabel 3.4 Tabel Perbedaan Kerangka Kerja LangChain dan LangGraph

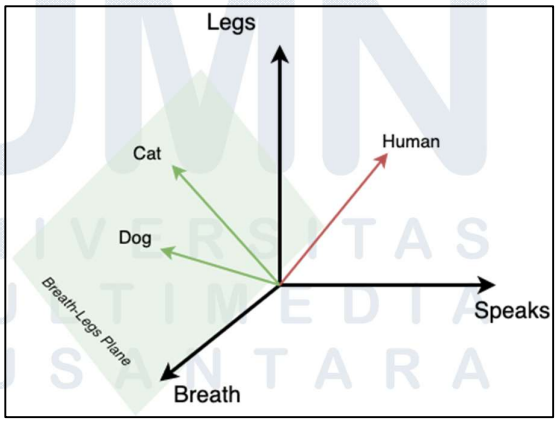
Aspek	LangChain	LangGraph
Struktur eksekusi	Linear dan satu arah	Bersifat dinamis dan dua arah

Aspek	LangChain	LangGraph
Pengelolaan konteks (State)	Terbatas, hanya per eksekusi	Tersimpan dan diperbarui sepanjang proses berjalan
Pola kerja	Sekuensial, bergantung pada langkah sebelumnya	Non-sekuensial, setiap node dapat berinteraksi secara fleksibel
Penanganan logika kompleks	Memerlukan logika tambahan manual	Didukung secara bawaan melalui StateGraph dan *edges*
Adaptivitas	Rendah, sulit menyesuaikan hasil selama proses	Tinggi, dapat beradaptasi terhadap kondisi proses
Penggunaan *tools* eksternal	Terbatas pada konfigurasi manual	Terintegrasi langsung dalam alur graf

LangGraph memiliki perbedaan peningkatan yang signifikan dibandingkan LangChain, terutama dalam hal fleksibilitas alur kerja dan pengelolaan konteks. Pendekatan berbasis graf membuat setiap komponen dalam sistem berinteraksi secara dinamis, sehingga proses pengambilan keputusan dapat menyesuaikan dengan kondisi yang terjadi selama eksekusi. Integrasi konsep *state* juga memberikan stabilitas yang lebih tinggi, karena sistem mampu mempertahankan konteks di setiap tahap tanpa kehilangan jejak data atau hasil sebelumnya. Selain itu, dukungan terhadap pendekatan ReAct membuat LangGraph lebih cerdas dalam menyeimbangkan antara proses berpikir dan bertindak, menjadikannya efektif untuk kasus penggunaan yang membutuhkan reasoning yang berulang dan tindakan adaptif. Keunggulan-keunggulan tersebut menjadikan LangGraph sebagai penyempurna dari LangChain dan fondasi bagi pengembangan sistem AI generatif yang lebih modular, interaktif, dan efisien.

3.3.1.5 Melakukan proses *embedding* teks dan pemuatan ke dalam basis data vektor

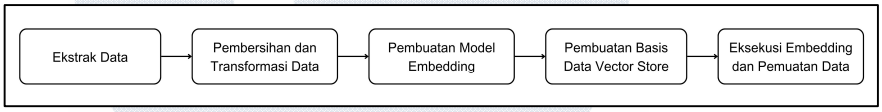
Pembuatan sistem berbasis *Large Language Model* (LLM) tidak bisa langsung menggunakan data berupa tabel terstruktur seperti di BigQuery. Data tersebut harus diubah menjadi bentuk yang dapat dipahami oleh model secara semantik, karena LLM tidak bekerja dengan mencocokkan kata secara harfiah, melainkan dengan memahami makna dan hubungan antar kata. Oleh karena itu, data perlu melalui proses *embedding*, yaitu mengubah teks menjadi representasi numerik dalam bentuk vektor. Vektor yang dihasilkan adalah hasil pemetaan makna dari teks ke dalam ruang multidimensi, di mana setiap dimensi merepresentasikan aspek tertentu dari arti kata atau kalimat. Sebagai contoh, pada Gambar 3.16, kata "anjing" dan "kucing" akan memiliki posisi vektor yang berdekatan karena keduanya punya konteks serupa sebagai mahluk hidup berkaki empat dan tidak bisa berbicara, sementara kata "manusia" posisinya akan jauh karena maknanya berbeda karena ia merupakan mahluk hidup berkaki dua yang bisa berbicara.



Gambar 3.16 Gambaran Pencarian Kemiripan Kata dengan *Embedding* Teks
Sumber: [20]

Embedding bertujuan agar komputer atau sistem memahami hubungan semantik antar kata atau kalimat dengan cara yang lebih alami. Setiap teks yang di-*embed* akan punya nilai koordinat dalam

ruang vektor. Kedekatan antar teks bisa diukur menggunakan jarak atau *similarity score*, di mana semakin kecil jarak antar dua vektor, semakin mirip konteks atau makna dari teks yang dibandingkan. Ketika pengguna memberikan pertanyaan, LLM tidak hanya mencari kata yang sama persis di basis data, namun akan mencari teks dengan vektor yang paling mendekati vektor pertanyaan tersebut sehingga hasil pencarian menjadi lebih kontekstual dan relevan. Hasil *embedding* teks nantinya disimpan dalam basis data vektor, sehingga sistem bisa melakukan pencarian semantik secara efisien dan menghasilkan jawaban yang selaras dengan konteks pertanyaan pengguna.



Gambar 3.17 Alur Pengerjaan *Embedding* Teks

Alur *embedding* teks untuk data katalog produk ditunjukkan pada Gambar 3.17. Data yang digunakan dalam proses *embedding* diambil dari data mart yang sebelumnya telah dibangun di BigQuery dalam bentuk view, yang kemudian diekstraksi ke dalam *dataframe* di lingkungan Python agar dapat diolah secara fleksibel. Tahapan berikutnya adalah melakukan transformasi kolom, yaitu menyesuaikan format data agar sesuai dengan standar Python serta merapikan isi data seperti menghapus nilai kosong, mengonversi tipe data, dan memastikan konsistensi teks yang akan di-*embed*. Hal ini dilakukan karena model *embedding* sangat sensitif terhadap format dan kebersihan data, di mana sedikit ketidakteraturan akan membuat proses *embedding* menjadi gagal atau hasil representasi vektornya kurang akurat.

Setelah data siap, model *embedding* dibangun untuk mengubah setiap entri teks menjadi vektor numerik yang merepresentasikan makna semantiknya di ruang multidimensi. Proses

ini memastikan setiap teks memiliki posisi unik berdasarkan konteks dan hubungan antar kata di dalamnya. Setelah *embedding* selesai, hasil vektor tersebut tidak langsung disimpan sebagai *file* biasa, melainkan dimasukkan ke dalam *vector store*, yaitu basis data khusus yang dirancang untuk menyimpan dan mengelola vektor. Basis data vektor ini menjadi fondasi utama dalam sistem *Retrieval-Augmented Generation* (RAG) yang digunakan dalam proyek ini.

```
query = """
SELECT
  CONCAT(
    COALESCE(a.name,'no product name'),
    ' ',
    COALESCE(a.short_description,'no description'),
    ' ',
    COALESCE(a.benefit, 'no specification'),
    ' ',
    COALESCE(a.features, 'no specification'),
    ' ',
    COALESCE(a.spesification, 'no specification'),
    ' ',
    COALESCE(a.range, 'no range')
  ) AS description,
  a.* EXCEPT(updated_at, created_at)
FROM `dataset.vw_product_catalogue` a
"""

query_job = client.query(query)
df = query_job.to_dataframe()
df.head()
```

Gambar 3.18 Potongan Kode untuk Ekstrak Data dari BigQuery

Proses *embedding* dilakukan menggunakan platform Google Colab karena menyediakan lingkungan komputasi yang fleksibel dan mendukung berbagai *library* untuk kecerdasan buatan. Proses ekstraksi data pada Gambar 3.18 dilakukan dengan menggunakan query untuk mengeluarkan data dari BigQuery. Data setiap produk digabungkan menjadi satu deskripsi utuh yang berisi nama, deskripsi singkat, manfaat, fitur, spesifikasi, dan jangkauan produk. Jika ada bagian informasi yang kosong, sistem otomatis mengisinya dengan keterangan pengganti seperti “tidak ada deskripsi” atau “tidak ada spesifikasi”. Tujuan langkah ini adalah memastikan setiap produk memiliki teks lengkap yang bisa dipahami oleh model bahasa saat proses *embedding* nanti. Setelah data dibersihkan dan digabungkan,

hasilnya diubah menjadi bentuk *dataframe* agar mudah diolah pada tahap berikutnya.

Setelah data berhasil diekstrak, data tersebut perlu ditransformasi dengan melakukan penyesuaian struktur data agar lebih seragam sebelum dimasukkan ke dalam basis data *vector* dengan mengubah kolom-kolom yang berisi kumpulan data berbentuk *nested* diubah menjadi *list* supaya bisa dibaca oleh sistem. Selain itu, data tanggal juga dikonversi ke format teks agar tidak menimbulkan error saat digunakan di proses selanjutnya. Tahap ini memastikan semua informasi produk sudah rapi, memiliki format yang konsisten, dan siap diproses menjadi representasi vektor yang bisa digunakan untuk pencarian atau analisis berbasis makna. Kode untuk proses transformasi data ditunjukkan pada Gambar 3.19.

```
df['prices'] = df['prices'].apply(lambda x: x.tolist())
df['stock_site'] = df['stock_site'].apply(lambda x: x.tolist())
df['alternative_article'] = df['alternative_article'].apply(lambda x: x.tolist())

def convert_datetimes(price_list):
    from datetime import datetime
    for item in price_list:
        if isinstance(item.get('valid_from'), datetime):
            item['valid_from'] = item['valid_from'].isoformat()
        if isinstance(item.get('valid_to'), datetime):
            item['valid_to'] = item['valid_to'].isoformat()
    return price_list

df['prices'] = df['prices'].apply(convert_datetimes)
```

Gambar 3.19 Potongan Kode untuk Transformasi Data

```
# replace NaN data
import numpy as np
df = df.replace({np.nan: None})
```

```
# replace NaN data with None for nested lists
def replace_nan_with_none(obj):
    import math
    if isinstance(obj, dict):
        return {k: replace_nan_with_none(v) for k, v in obj.items()}
    elif isinstance(obj, list):
        return [replace_nan_with_none(item) for item in obj]
    elif isinstance(obj, float) and math.isnan(obj):
        return None
    else:
        return obj

df = df.applymap(replace_nan_with_none)
```

Gambar 3.20 Potongan Kode untuk Pembersihan Data

Selain transformasi data, dilakukan juga proses pembersihan data sebelum digunakan untuk *embedding* dengan mengganti nilai kosong atau tidak terdefinisi agar tidak mengganggu saat data diolah oleh model bahasa, seperti pada Gambar 3.20. Pada tahap awal, sistem mengganti seluruh nilai kosong sederhana dengan nilai pengganti agar tidak ada data yang bernilai NaN. Namun, karena beberapa kolom berisi struktur data yang lebih kompleks seperti *list* dan *dict*, dibuatlah fungsi tambahan untuk memeriksa setiap elemen di dalamnya secara mendalam. Fungsi ini memastikan bahwa bahkan nilai kosong yang tersembunyi di dalam struktur bersarang pun akan diganti dengan nilai yang aman, sehingga seluruh data memiliki format yang konsisten dan bebas dari elemen tak dikenal yang bisa menyebabkan eror saat diproses lebih lanjut.

```python
from google.cloud.sql.connector import Connector
import sqlalchemy

# initialize Connector object
connector = Connector()

# function to return the database connection object
def getconn():
    conn = connector.connect(
        INSTANCE_CONNECTION_NAME,
        "pg8000",
        user=DB_USER,
        password=DB_PASS,
        db=DB_NAME
    )
    return conn

# create connection pool with 'creator' argument to connection object function
connection = sqlalchemy.create_engine(
    "postgresql+pg8000://",
    creator=getconn,
)
```

Gambar 3.21 Potongan Kode untuk Pengaturan Koneksi Basis Data Cloud SQL

Gambar 3.21 menunjukkan kode untuk mengatur koneksi antara sistem AI dan basis data yang berjalan di layanan Cloud SQL milik Google. Koneksi tersebut dilakukan agar sistem dapat terhubung ke basis data secara aman dan efisien tanpa harus mengatur koneksi manual yang rumit. Setelah koneksi berhasil dibuat, sistem dapat menyimpan, membaca, atau memproses data di dalam basis data

tersebut. Basis data ini digunakan untuk menyimpan hasil konversi teks menjadi vektor yang dihasilkan oleh model *embedding* pada Gambar 3.22. Model *embedding* dibuat menggunakan model yang berasal dari layanan Vertex AI di Google Cloud, yakni text-multilingual-embedding-002 yang memiliki keunggulan dalam penanganan teks multibahasa, termasuk bahasa Indonesia, sehingga bisa digunakan untuk berbagai kebutuhan global. Hasil pengaturan koneksi basis data PostgreSQL dan pembuatan model dihubungkan dengan melakukan *embedding* dari model kecerdasan buatan dengan menggunakan ekstensi PGVector.

```python
from langchain_google_vertexai import VertexAIEmbeddings
from langchain_postgres import PGVector

embeddings = VertexAIEmbeddings(
    model_name="text-multilingual-embedding-002",
    project=PROJECT_ID,
    location=LOCATION
)

vector_store = PGVector(
    embeddings=embeddings,
    collection_name=collection_name,
    connection=connection,
    use_jsonb=True,
)
```

Gambar 3.22 Potongan Kode untuk Pengaturan Model *Embedding*

```python
import time

# create metadata
list_metadata = df_metadata.to_dict(orient="records")
list_text = df_text.tolist()

i = 0
while i < len(list_text):
    if i + 1000 < len(list_text):
        print(f"Embedding index {i} to {i + 999}")
        vector_store.add_texts(list_text[i:i+1000], metadatas=list_metadata[i:i+1000])
        i += 1000
    else:
        print(f"Embedding index {i} to last")
        vector_store.add_texts(list_text[i:], metadatas=list_metadata[i:])
        i += 1000

    time.sleep(10)
```

Gambar 3.23 Potongan Kode untuk Pemuatan Data *Embedding* ke Basis Data

Langkah selanjutnya adalah pemuatan data hasil *embedding* teks ke dalam basis data vektor. Sebelum memasukkan data ke dalam

basis data, terdapat tahap pembentukan data pendukung yang disebut metadata menggunakan kode pada Gambar 3.23. Metadata ini berisi informasi penting mengenai setiap entri teks, yakni identitas produk, kategori, dan atribut lain yang berguna saat pencarian dilakukan pada tahap pengambilan data pada RAG. Data tersebut diubah menjadi bentuk daftar kamus agar mudah dikenali dan dipasangkan dengan teks deskripsi produk yang sudah disiapkan sebelumnya. Kemudian, metadata juga diubah menjadi bentuk list agar bisa langsung digunakan dalam proses *embedding*. Langkah ini memastikan setiap teks memiliki konteks informatif melalui metadata, sehingga hasil pencarian semantik nantinya bisa lebih akurat dan relevan. Setelah metadata dibuat, data teks akan dikirim ke dalam basis data vektor secara bertahap. Proses ini dilakukan dengan cara memecah data menjadi beberapa bagian kecil agar sistem tidak terbebani saat memproses dalam jumlah besar. Setiap kelompok teks diubah menjadi vektor menggunakan model yang sudah disiapkan sebelumnya, lalu disimpan ke dalam basis data beserta metadatanya. Setelah setiap bagian selesai, program menunggu beberapa detik sebelum melanjutkan ke batch berikutnya, sebagai cara menghindari beban berlebih pada server atau koneksi. Ketika proses telah selesai, hasil dari data *embedding* yang telah tersimpan pada basis data vektor dapat dilihat melalui Google Cloud SQL, dengan bentuk struktur data seperti pada Tabel 3.5. Semua data yang berhasil diubah menjadi vektor dan tersimpan dengan aman di dalam basis data siap digunakan untuk pencarian produk pada AI katalog produk.

Tabel 3.5 Tabel Contoh Data *Embedding* pada Google Cloud SQL

id	collection_id	embedding	document	cmetadata
1	abcde123	[-0.0320,-0.0572,-0.0294,0.0868, …]	Micrometer …	{"sku": "7", "name": "Micrometer", "brand": "A1", …}

id	collection_id	embedding	document	cmetadata
2	defgh456	[-0.0029,-0.0904,-0.0253,0.0678, …]	Snap Gauge …	{"sku": "5", "name": "Snap Gauge", "brand": "B2", …}
3	ijklm789	[-0.0105,-0.0635, -0.0405,0.0741, …]	Caliper Dial 505 …	{"sku": "6", "name": "Caliper Dial 505", "brand": "C3", …}
4	nopqr098	[-0.7743,-0.2290, -0.0025,0.0732, …]	Caliper Vernier …	{"sku": "8", "name": "Caliper Vernier", "brand": "D4", …}
5	stuvw765	[-0.0330,-0.0591,-0.0225,0.0843, …]	Height Gauge …	{"sku": "1", "name": "Height Gauge", "brand": "E5", …}
6	xyzab432	[-0.0320,-0.0394,-0.0537,0.0063, …]	Micrometer …	{"sku": "4", "name": "Micrometer", "brand": "F6", …}
7	cdefg010	[-0.0854,-0.0361,-0.0199,0.0372, …]	Caliper Dial 98 …	{"sku": "2", "name": "Caliper Dial 98", "brand": "G7", …}
8	hijkl353	[-0.0367,-0.0372,-0.0218,0.0749, …]	Magnetic Stand …	{"sku": "4", "name": "Magnetic Stand", "brand": "H8", …}

3.3.1.6 Melakukan *fine-tuning* dan pengujian *Large Language Model*

Pembuatan *Large Language Model* (LLM) untuk sistem AI katalog produk dilakukan menggunakan Chainlit sebagai platform utama dalam membangun dan menguji hasil model. Kerangka kerja ini dipilih karena mampu menghubungkan model bahasa dengan alur interaktif secara langsung, sehingga proses pengujian hasil model dapat dilakukan secara *real time*. Terdapat dua komponen utama yang perlu dioptimasi, yaitu *file* aplikasi Chainlit dan *prompt* untuk SystemMessage. *File* aplikasi berfungsi sebagai dasar utama yang mengatur bagaimana model berinteraksi dengan pengguna, termasuk penerimaan input, pemanggilan fungsi model, hingga penampilan respons di antarmuka. Sementara itu, *prompt* SystemMessage

digunakan untuk mengarahkan perilaku model agar menghasilkan jawaban yang konsisten dengan apa yang dibutuhkan *user*.

```
# Init VertexAI
vertexai.init(project=PROJECT_ID, location=LLM_LOCATION)
memory = MemorySaver()

# Init Data Persistence
storage_client = GCSStorageClient(bucket=PERSISTENCE_BUCKET)
cl_data._data_layer = CloudSQLSQLAlchemyDataLayer(
    instance_connection_name=f"{PROJECT_ID}:{PROJECT_LOCATION}:{VECTORSTORE_DB_INSTANCE}",
    db_name=VECTORSTORE_DB_NAME,
    db_user=VECTORSTORE_DB_USER,
    db_pass=VECTORSTORE_DB_PASS,
    storage_provider=storage_client
)

# Init Connection to Vector Store
def getconn():
    connector = Connector()
    conn = connector.connect(
        VECTORSTORE_INSTANCE_CONNECTION_NAME,
        "pg8000",
        user=VECTORSTORE_DB_USER,
        password=VECTORSTORE_DB_PASS,
        db=VECTORSTORE_DB_NAME
    )
    return conn

vectorstore_connection = sqlalchemy.create_engine(
    "postgresql+pg8000://", creator=getconn,
)
```

Gambar 3.24 Potongan Kode Inisialisasi Aplikasi Chainlit

Proses diawali dengan menginisialisasi koneksi dan penyimpanan data pada sistem Vertex AI pada Gambar 3.24 dan pendefinisian *memory saver* untuk membantu sistem menyimpan sementara hasil interaksi atau proses yang berjalan. Terdapat juga penambahan *data persistence* menggunakan Google Cloud Storage sebagai media penyimpanan permanen dan koneksi ke basis data menggunakan Google Cloud SQL dengan SQLAlchemy *Data Layer*. Tujuannya adalah agar data yang dikelola, seperti hasil pemrosesan atau *embedding* vektor, bisa tersimpan dan diakses secara aman di basis data *cloud*. Terakhir, fungsi getconn() digunakan untuk membuat koneksi langsung ke basis data vektor yang telah dibuat sebelumnya dengan memanfaatkan Cloud SQL Connector, dan hasilnya dijadikan sebagai *engine* SQLAlchemy agar aplikasi dapat berinteraksi dengan basis data secara efisien.

Setelah koneksi dan penyimpanan data berhasil diinisialisasi, langkah berikutnya adalah menyiapkan model utama yang digunakan

dalam aplikasi. Pada Gambar 3.25, sistem memulai LLM dengan memanggil fungsi init_chat_model, yang mengambil model dari layanan Vertex AI. Model ini menjadi inti dari proses percakapan atau pemrosesan bahasa alami yang dilakukan aplikasaplikasi. Kemudian, dilakukan inisialisasi model *embedding* dan basis data vektor dengan dengan VertexAIEmbeddings dan PGVector agar dapat digunakan saat melakukan pencarian produk dari pertanyaan pengguna. Setelah model utama dan basis data siap digunakan, tahap berikutnya adalah mendefinisikan alat bantu atau *tools* yang akan memperluas kemampuan sistem dalam menjawab pertanyaan pengguna, yakni find_product_details, sebuah fungsi yang berfungsi untuk mencari detail produk dari katalog yang telah disimpan di basis data vektor. Fungsi yang tertera pada Gambar 3.26 dirancang agar sistem dapat memahami dan merespons permintaan pengguna yang berkaitan dengan produk.

```
# Init LLM, Embedding Model, and Vector Store
llm = init_chat_model(LLM_MODEL, model_provider="google_vertexai")
embeddings = VertexAIEmbeddings(
    model_name=EMBEDDING_MODEL,
    project=PROJECT_ID,
    location=PROJECT_LOCATION
)
vector_store = PGVector(
    embeddings=embeddings,
    collection_name=VECTORSTORE_COLLECTION_NAME,
    connection=vectorstore_connection,
    use_jsonb=True,
)
```

Gambar 3.25 Potongan Kode Pembuatan Model *Embedding*

Proses pencarian dilakukan menggunakan metode *similarity search* pada basis data vektor, yang mencari hingga 20 entri paling mirip dengan teks yang dimasukkan oleh pengguna. Berbeda dengan pencarian kata kunci biasa, metode ini mencari berdasarkan kesamaan makna, sehingga hasil yang diperoleh lebih kontekstual dan akurat. Hasil pencarian kemudian diproses ulang melalui sebuah perulangan untuk membersihkan serta melengkapi informasi yang akan dikembalikan. Dalam proses ini, setiap item hasil pencarian diambil metadata-nya, seperti ID produk (SKU), nama, deskripsi, harga,

gambar, dan jumlah stok. Selain itu, sistem juga secara otomatis menambahkan tautan *e-commerce* berdasarkan SKU, agar pengguna bisa langsung diarahkan ke halaman produk terkait di situs Kawan Lama. Setelah seluruh hasil dibersihkan dan diperkaya dengan tautan, data dikembalikan dalam bentuk daftar *(list)* berisi detail produk yang sudah siap ditampilkan atau diolah lebih lanjut oleh sistem. Terakhir, fungsi find_product_details ini dimasukkan ke dalam daftar *tools* agar dapat dipanggil oleh model utama saat dibutuhkan dalam percakapan.

```python
# Init tools
@tool
def find_product_details(
        product_to_search: Annotated[
            str, "Product to be searched from the user question. Can be a specific product name or a broad product t
):
    """Tool to find product details from the product catalogue.
    The information that are covered by this tool are product name, product ID, product price, product description,

    result = vector_store.similarity_search(
        product_to_search,
        k=20,
    )

    print(result)

    cleaned_result = []
    for item in result:
        content = item.metadata

        # URL
        sku = content['sku']
        url = f"https://www.website.com/product/detail/{sku}"
        content['ecommerce_url'] = url

        cleaned_result.append(content)

    result = cleaned_result
    return result

tools = [find_product_details]
```

Gambar 3.26 Potongan Kode Definisi *Tools* find_product_details

```python
# Node and Graph builder
sys_msg = SystemMessage(content=KAPIBARA_SYSTEM)
llm_with_tools = llm.bind_tools(tools)

# Node
def assistant(state: MessagesState):
    return {"messages": [llm_with_tools.invoke([sys_msg] + state["messages"])]}

# Graph
builder = StateGraph(MessagesState)

# Define nodes: these do the work
builder.add_node("assistant", assistant)
builder.add_node("tools", ToolNode(tools))

# Define edges: these determine how the control flow moves
builder.add_edge(START, "assistant")
builder.add_conditional_edges(
    "assistant",
    # If the latest message (result) from assistant is a tool call -> tools_condition routes to tools
    # If the latest message (result) from assistant is a not a tool call -> tools_condition routes to END
    tools_condition,
)
builder.add_edge("tools", "assistant")
graph = builder.compile(checkpointer=memory)
```

Gambar 3.27 Potongan Kode Pembuatan *Node* dan *Graph*

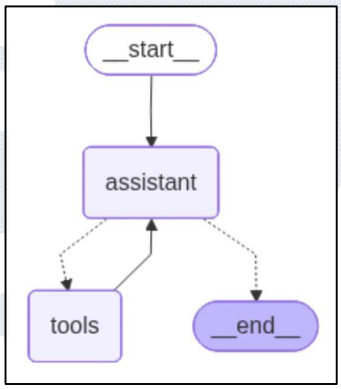
Langkah berikutnya adalah membangun *node* dan *graph* yang akan mengatur alur percakapan antara model, alat bantu, dan pengguna, menggunakan kode pada Gambar 3.27. Bagian ini dimulai dengan mendefinisikan SystemMessage yang berisi instruksi utama untuk model, diambil dari *file prompt* pada Gambar 3.28. Pesan sistem ini berperan sebagai panduan perilaku model agar tetap konsisten dengan tujuan aplikasi, seperti menjaga gaya komunikasi, memahami konteks percakapan, dan memutuskan kapan harus memanggil alat tertentu. Selanjutnya, model LLM dihubungkan dengan alat bantu melalui llm.bind_tools(tools), sebagai fungsi tambahan model seperti find_product_details saat dibutuhkan dalam percakapan.

```
KAPIBARA_SYSTEM = """You are a Mitutoyo product specialist and salesman for precision measuring tools
and system instruments (e.g., CMMs, form and optical measuring instruments).
Your goal is to give accurate, detailed, and relevant answers about Mitutoyo products.

### Rules & Behavior
- You can access the `find_product_details` tool to retrieve product data.
- Always use this tool first when searching for product information.
- Do **not** use external knowledge or the internet. Only use data returned by `find_product_details`.
- Always reply in the **same language as the user** (Bahasa Indonesia or English).
- If user input is unclear (e.g., missing measurement range or reading type), ask for clarification.
- If no specific product is requested, recommend **10 products by default** (including discontinued ones).
- If the user requests a specific number of products, follow that number.
- Sort results by **quantity sold (last 6 months, then last 3 months)** in descending order.
- Prioritize product names that are most similar to the user's query.
- You may call `find_product_details` multiple times if needed.

### Output Format (for each product)
1. **Product Name**
2. **Product ID**
3. **Price** - Normal and Discounted (in IDR)
4. **Status** - Active or Discontinued - If discontinued, include the **alternative name & ID**
5. **Description **
6. **Specifications ** - accuracy, material, resolution, output type (digital/analog), IP rating
7. **Benefits **
8. **After-sales Service ** - e.g., warranty info
9. ** Indent Time **
10. **Quantity Sold** - last 6 months & last 3 months
11. **Stock Details** - list by site (site name, ID, quantity), sorted by highest stock first
12. **E-commerce URL**
13. **Image URL** - Markdown format: `! [](image_url)'

If a detail is missing, write "Not available."

### Final Output
Your final answer MUST contains the summary list of the products recommendation.
"""
```

Gambar 3.28 *Prompt* SystemMessage untuk AI Katalog Produk

Setelah pembangunan *node* dan *graph*, kedua komponen tersebut didefinisikan sebagai struktur utama yang mengatur bagaimana sistem berpikir dan berinteraksi. Fungsi assistant(state)

menjadi *node* inti yang menangani proses komunikasi antara pengguna dan model. Setiap kali pengguna mengirim pesan, fungsi ini akan memanggil model yang sudah terhubung dengan alat bantu, kemudian menghasilkan balasan berdasarkan konteks yang ada di pesan. Setelah itu, dibentuklah *graph* menggunakan StateGraph(MessagesState), yang berfungsi seperti peta alur kerja percakapan. Dalam *graph* ini, setiap *node* mewakili satu langkah logika, yakni "assistant" untuk pemrosesan pesan dan "tools" untuk menjalankan fungsi tambahan seperti pencarian produk.



Gambar 3.29 Ilustrasi Arsitektur *Agent* dengan StateGraph

Selanjutnya, ditentukan hubungan antar *node* melalui *edges*, yaitu jalur yang menentukan arah alur proses. Ilustrasi dari StateGraph yang telah dibuat dapat dilihat pada Gambar 3.29. Alur dimulai dari START menuju assistant, kemudian sistem akan memeriksa kondisi tertentu menggunakan tools_condition. Jika hasil dari model memerlukan penggunaan alat bantu, alur akan diarahkan ke *node tools*, jika tidak, maka proses akan langsung berakhir. Setelah alat dijalankan, alur kembali ke *assistant* untuk menghasilkan respons akhir yang sudah diperkaya dengan hasil pencarian. Graf ini dapat disebut sebagai arsitektur *agent* dari AI katalog produk.

Kode pada Gambar 3.30 berfungsi untuk mengatur komunikasi langsung antara pengguna dan sistem di dalam aplikasi

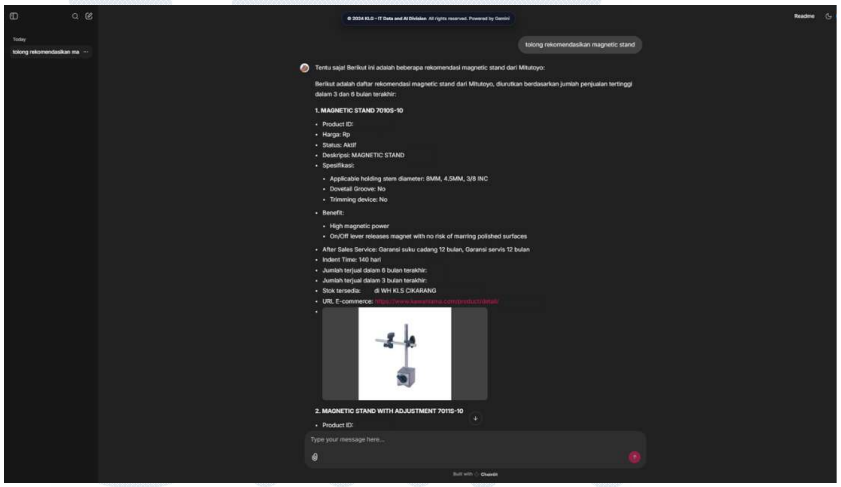
Chainlit. Saat pengguna mengirim pesan, fungsi main(msg) dijalankan untuk meneruskan pesan tersebut ke alur percakapan yang telah disiapkan. Pada tahap awal, sistem membuat pengaturan sesi agar setiap pengguna memiliki percakapan yang tersimpan dan tidak saling bertumpuk. Selain itu, sistem juga menyiapkan mekanisme pencatat proses agar setiap langkah model dapat dipantau selama percakapan berlangsung. Kemudian, sistem akan menyiapkan wadah untuk menampilkan hasil jawaban dari model. Melalui proses yang berjalan secara bertahap, pesan dari pengguna diproses, lalu model memberikan tanggapan sedikit demi sedikit secara langsung, sehingga terasa seperti percakapan alami. Begitu seluruh jawaban selesai diproses, sistem mengirimkan hasil akhirnya ke layar pengguna. Pengalaman berinteraksi akan terasa lebih responsif dan dinamis bagi pengguna melalui fungsi ini, karena sistem mampu menampilkan hasil secara *real-time* sambil tetap menjaga alur percakapan berjalan dengan lancar.

```python
@cl.on_message
async def main(msg: cl.Message):
    config = {"configurable": {"thread_id": cl.context.session.id}}
    cb = cl.LangchainCallbackHandler()
    final_answer = cl.Message(content="")

    for msg in graph.stream(
        {"messages": [HumanMessage(content=msg.content)]},
        stream_mode="messages",
        # config=RunnableConfig(callbacks=[cb], **config)
        config=config
        ):
        if (
                msg.content
                and not isinstance(msg, HumanMessage)
                and not isinstance(msg, ToolMessage)
        ):
            await final_answer.stream_token(msg.content)

    await final_answer.send()
```

Gambar 3.30 Potongan Kode Penanganan Pesan Pengguna pada Aplikasi Chainlit

Setelah proses *fine-tuning* selesai, tahap berikutnya adalah melakukan pengujian hasil *output* LLM untuk memastikan model mampu merespons pertanyaan pengguna sesuai dengan konteks dan

format yang diharapkan. Pengujian dilakukan melalui platform Chainlit yang dapat diakses menggunakan tautan internal perusahaan. Pada tahap ini, model diuji dengan berbagai jenis pertanyaan seputar produk, mulai dari permintaan rekomendasi hingga pencarian informasi detail. Hasil uji menunjukkan bahwa ketika pengguna memberikan pertanyaan, sistem dapat menampilkan hasil berupa rangkuman produk yang disusun dalam bentuk poin dan diurutkan berdasarkan tingkat penjualan tertinggi, sebagaimana ditunjukkan pada Gambar 3.31. Pengujian ini menjadi langkah awal untuk menilai kemampuan model dalam memberikan rekomendasi berbasis data penjualan aktual sebelum dilakukan pengembangan yang lebih kompleks.



Gambar 3.31 Hasil Pengujian Respons Model AI Katalog Produk

Hasil *chatbot* berbasis AI memberikan peningkatan aksesibilitas dan keterbacaan informasi produk bagi pengguna internal. Informasi yang sebelumnya bersifat teknis dan tersebar dalam skema data yang kompleks dapat disajikan ulang dalam bentuk rangkuman yang mudah dipahami melalui interaksi berbasis pertanyaan. Pendekatan ini mengurangi ketergantungan terhadap proses eksplorasi data secara manual maupun bantuan teknis dari tim lain. Implementasi ini juga membuka peluang pemanfaatan lebih

lanjut sebagai fondasi pengembangan fitur analitik dan rekomendasi produk yang lebih kaya di masa mendatang, seiring dengan peningkatan kualitas data dan perluasan cakupan informasi yang digunakan oleh model.

3.3.1.7 Mengeksplorasi jenis model prediksi BigQuery ML

Pengembangan model prediksi dapat dilakukan menggunakan berbagai metode, mulai dari pemodelan berbasis Python hingga proses otomatis yang tersedia di platform seperti BigQuery. Pada tahap awal eksplorasi untuk membuat model prediksi permintaan barang, beberapa jenis model deret waktu telah dicoba menggunakan Google Colab untuk menangani karakteristik data yang berbeda, seperti pola permintaan reguler hingga permintaan sporadis. Namun, setelah beberapa kali percobaan menggunakan sampel data, pendekatan tersebut tidak memadai untuk skala data yang sangat besar karena lamanya waktu eksekusi dan kapasitas RAM tidak mampu memuat data penuh yang mencapai puluhan juta baris. Kondisi tersebut membuat proses pengembangan model kurang efisien dan tidak memungkinkan untuk diterapkan sebagai *pipeline* yang berjalan secara rutin. Karena itu, diputuskan untuk beralih ke BigQuery sebagai platform utama dalam membangun model prediksi.

Pembuatan model prediksi permintaan barang dilakukan menggunakan BigQuery ML karena platform ini menawarkan kemudahan dalam proses pengembangan tanpa perlu melakukan konfigurasi lingkungan pemodelan secara manual. Pendekatan ini dinilai efisien karena seluruh proses, mulai dari pelatihan model hingga prediksi, dapat dilakukan langsung melalui perintah SQL. Selain itu, BigQuery ML memiliki keunggulan dalam integrasinya dengan *pipeline* data yang sudah dibangun oleh tim *Data Engineer*, sehingga proses *deployment* model ke tahap produksi dapat berjalan lebih sederhana tanpa perlu membangun infrastruktur tambahan.

Dalam BigQuery ML, terdapat dua pendekatan utama untuk menghasilkan model prediksi, yaitu menggunakan fungsi AI.FORECAST() dan ML.FORECAST(). Fungsi AI.FORECAST() merupakan metode berbasis *auto* ML yang secara otomatis memilih algoritma dan parameter terbaik untuk melakukan peramalan deret waktu tanpa memerlukan pengaturan teknis yang rumit. Sementara itu, ML.FORECAST() memiliki fleksibilitas lebih besar karena penentuan parameter model dapat dilakukan secara manual, sehingga cocok untuk kasus yang membutuhkan kontrol lebih mendetail terhadap proses pelatihan dan evaluasi model. Kedua metode ini memberikan hasil yang dapat langsung digunakan untuk menganalisis tren kebutuhan inventaris di masa mendatang, dengan keunggulan efisiensi eksekusi dan kemudahan integrasi ke sistem yang sudah berjalan di lingkungan BigQuery.

Fungsi AI.FORECAST() di BigQuery ML menggunakan model bawaan bernama TimesFM 2.0 *(Time-series Foundation Model)*, yang dirancang untuk menangani prediksi deret waktu secara otomatis. Model ini merupakan hasil pengembangan dari pendekatan *foundation model* milik Google yang telah dilatih menggunakan miliaran titik data deret waktu dari berbagai domain, sehingga mampu mengenali pola musiman, tren, dan fluktuasi secara umum tanpa memerlukan proses pelatihan manual dari pengguna. Penggunaan model ini hanya perlu memanggil fungsi AI.FORECAST() dan memberikan data historis dan nilai yang ingin diprediksi, lalu sistem akan melakukan proses analisis pola dan menghasilkan proyeksi nilai di periode mendatang. Model ini secara otomatis mengatur parameter seperti deteksi musiman dan penyesuaian tren berdasarkan karakteristik data yang dimasukkan. Keunggulan utama dari pendekatan ini terletak pada kepraktisan dan kecepatan penggunaannya, karena tidak diperlukan proses pelatihan eksplisit maupun konfigurasi model yang kompleks. Selain itu, hasil prediksi

yang dihasilkan cukup stabil dan akurat untuk kebutuhan analisis umum atau perencanaan jangka pendek. Namun, kapasitas data historis yang dapat dibaca oleh AI.FORECAST() hanya terbatas pada 512 titik waktu, yang membuatnya kurang cocok untuk dataset dengan rentang waktu yang panjang atau frekuensi tinggi, seperti data transaksi harian selama beberapa tahun. Selain itu, karena modelnya sudah ditentukan oleh sistem, pengguna tidak memiliki keleluasaan untuk menyesuaikan parameter, mengubah arsitektur model, atau menambahkan fitur tambahan seperti variabel eksternal. AI.FORECAST() cocok digunakan pada skenario di mana kemudahan dan kecepatan lebih diutamakan daripada kendali penuh terhadap proses pemodelan.

Berbeda dengan AI.FORECAST(), fungsi ML.FORECAST() di BigQuery ML difokuskan untuk membuat dan melatih model prediksi deret waktu secara mandiri seperti menentukan struktur model dan parameter pelatihan yang ingin dilibatkan dalam perhitungan prediksi. Prosesnya dimulai dengan membuat model pada konsol BigQuery, di mana pengguna dapat memilih algoritma yang sesuai, kemudian melakukan pelatihan menggunakan data historis yang relevan. Setelah model terbentuk, fungsi ML.FORECAST() dijalankan untuk menghasilkan prediksi nilai di masa depan berdasarkan pola yang telah dipelajari. Dalam ML.FORECAST(), terdapat dua model yang disediakan, yakni ARIMA_PLUS dan ARIMA_PLUS_XREG. Model ARIMA_PLUS digunakan ketika prediksi hanya didasarkan pada data historis tanpa mempertimbangkan faktor tambahan, sedangkan model ARIMA_PLUS_XREG memiliki tambahan faktor regresor eksternal seperti variabel bisnis, promosi, atau faktor musiman lainnya, sehingga hasilnya lebih kontekstual dan akurat terhadap kondisi nyata. Kelebihan utama dari ML.FORECAST() terletak pada kontrol penuh terhadap model dan kemampuannya untuk melakukan analisis

diagnostik mendalam, sehingga cocok untuk kebutuhan bisnis yang menuntut transparansi hasil dan penyesuaian parameter yang presisi. Namun, fleksibilitas ini juga menuntut pemahaman teknis yang lebih baik karena pengguna harus memastikan data bersih, parameter model optimal, serta frekuensi pembaruan model seimbang antara performa dan efisiensi.

Tabel 3.6 Tabel Perbandingan Fungsi Prediksi pada BigQuery ML

Aspek	AI.FORECAST()	ML.FORECAST()
Proses Pelatihan	Tidak perlu membuat atau melatih model	Perlu membuat dan melatih model
Faktor Eksternal	Tidak dapat menambahkan faktor eksternal	Dapat menambahkan faktor eksternal (model ARIMA_PLUS_XREG)
Kemudahan Penggunaan	Sangat mudah	Lebih kompleks
Kontrol terhadap Model	Terbatas	Tidak terbatas
Kebutuhan Data Historis	Maksimal 512 titik waktu	Tidak memiliki batasan ketat
Frekuensi Pembaruan	Otomatis berdasarkan data terkini	Dapat diatur manual sesuai jadwal pelatihan
Kelebihan Utama	Cepat dan praktis untuk prediksi umum	Lebih akurat dan fleksibel untuk kebutuhan bisnis spesifik
Kekurangan Utama	Kurang transparan dan sulit dikustomisasi	Membutuhkan waktu dan pemahaman teknis
Kesesuaian Penggunaan	Cocok untuk analisis cepat dan eksploratif	Cocok untuk prediksi operasional dan berbasis faktor bisnis

Untuk memahami perbedaan antara kedua fungsi tersebut, Tabel 3.6 menampilkan perbandingan antara AI.FORECAST() dan ML.FORECAST() berdasarkan karakteristik, proses, serta fleksibilitas penggunaannya. Berdasarkan hasil perbandingan tersebut, fungsi yang lebih sesuai untuk kebutuhan prediksi

perencanaan inventaris adalah ML.FORECAST(), karena memberikan kendali penuh terhadap proses pelatihan dan penyesuaian model. Oleh karena itu tim akan membangun model ARIMA_PLUS dan ARIMA_PLUS_XREG, untuk kemudian dibandingkan dari sisi performa, efisiensi, serta ketepatan hasilnya. Evaluasi ini akan menjadi dasar dalam menentukan model mana yang paling optimal digunakan untuk menghasilkan prediksi inventaris yang lebih akurat dan dapat diandalkan.

3.3.1.8 Menganalisis data histori untuk membuat data pelatihan

Proses menghasilkan model prediksi yang akurat tidak hanya bergantung pada pemilihan jenis model, tetapi juga pada kualitas serta struktur data historis yang digunakan sebagai dasar pelatihan. Data yang tidak lengkap, tidak konsisten, atau tidak mencerminkan pola sebenarnya akan membuat model sulit mempelajari hubungan antarvariabel secara tepat, sehingga hasil prediksinya menjadi kurang reliabel. Karena itu, tahap analisis data histori perlu dilakukan sebelum masuk ke proses pemodelan.

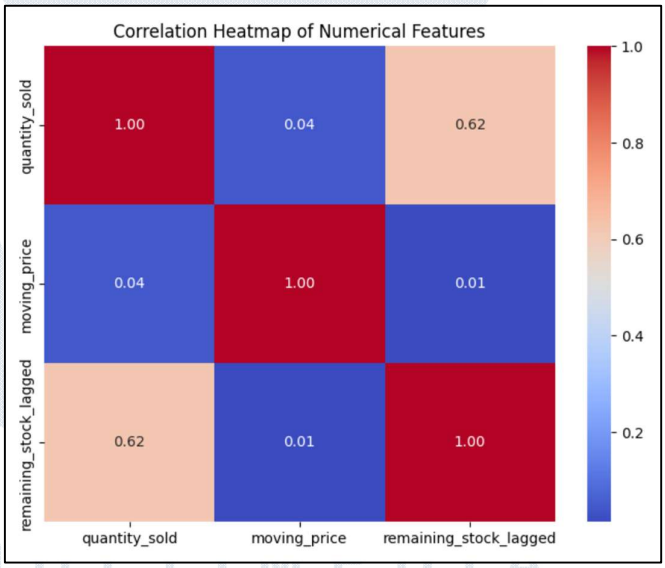
Tahap awal yang perlu dilakukan adalah memastikan bahwa variabel-variabel yang akan dimasukkan ke dalam model sudah ditentukan dan disiapkan dengan benar. Dalam kasus model ARIMA_PLUS_XREG, terdapat dua komponen utama yang harus disusun, yaitu variabel target yang menjadi sasaran prediksi, serta variabel regresor eksternal yang berfungsi menjelaskan faktor-faktor yang dapat memengaruhi nilai target tersebut. Pemilihan kedua jenis variabel ini menjadi langkah yang penting karena akan menentukan sejauh mana model mampu memahami pola historis dan hubungan antarvariabel sebelum masuk ke tahap pemodelan yang lebih teknis.

Pemilihan variabel prediksi dimulai dari tujuan utama analisis, yaitu membangun model untuk memperkirakan permintaan suatu barang di tingkat toko. Variabel yang digunakan sebagai target adalah

jumlah produk yang terjual pada masing-masing toko, yang direpresentasikan melalui kolom quantity_sold. Setelah target ditetapkan, langkah berikutnya adalah menentukan variabel regresor eksternal yang berpotensi mempengaruhi pola penjualan. Proses penentuan regresor dilakukan dengan menelusuri dua sumber data utama, yaitu dim_product_store_status dan fact_inventory, karena keduanya menyimpan konteks operasional yang berkaitan dengan harga dan pergerakan stok barang. Tabel dim_product_store_status menyediakan gambaran mengenai karakteristik produk pada level toko. Dari tabel ini dipilih kolom moving_price, yang merepresentasikan nilai penilaian persediaan yang terus diperbarui seiring perubahan biaya perolehan barang. Informasi tersebut penting untuk model prediksi, sebab fluktuasi harga dapat memengaruhi keputusan pembelian pelanggan serta ritme keluar masuknya stok. Sementara itu, tabel fact_inventory digunakan untuk memahami kondisi stok barang berdasarkan periode waktu. Dari tabel ini variabel didapatkan dengan menghitung sisa stok yang diposisikan satu bulan sebelumnya, yang diberi nama remaining_stock_lagged, yaitu. Penggunaan nilai *"lag"* memberikan perspektif yang lebih realistis dibanding persediaan saat ini, karena stok pada periode lalu merupakan kondisi yang benar-benar tersedia ketika proses penjualan di periode berjalan terjadi. Informasi tersebut sangat membantu model dalam menangkap dinamika permintaan yang dipengaruhi oleh ketersediaan barang, sekaligus menghindari masalah *data leakage*, yaitu kondisi ketika model "melihat" informasi dari masa depan saat proses pelatihan, yang dapat menimbulkan prediksi tidak valid di implementasi sebenarnya.

Setelah menentukan variabel prediksi dan kandidat regresor eksternal, langkah berikutnya adalah menentukan variabel regresor eksternal dengan melihat hubungan masing-masing kandidat variabel terhadap quantity_sold. Proses ini dilakukan menggunakan visualisasi

korelasi *(heatmap)* untuk memahami seberapa kuat sebuah variabel memengaruhi variabel target. Hasil pada Gambar 3.32 menunjukkan bahwa remaining_stock_lagged memiliki korelasi yang cukup kuat dengan quantity_sold, yaitu sebesar 0.62, sehingga dapat dianggap relevan sebagai faktor yang memengaruhi volume penjualan, sedangkan variabel moving_price hanya memiliki korelasi sekitar 0.04, angka yang sangat rendah dan menunjukkan bahwa perubahan pada nilai penilaian persediaan hampir tidak berpengaruh terhadap jumlah penjualan. Berdasarkan temuan tersebut, remaining_stock_lagged dipilih sebagai variabel eksternal yang digunakan dalam data pelatihan model prediksi karena memiliki hubungan yang lebih jelas dan bermakna terhadap pergerakan permintaan.



Gambar 3.32 Hasil Korelasi *Heatmap* Kandidat Variabel Model Prediksi

Secara bersamaan dengan penentuan variabel regresor eksternal, dilakukan proses pembersihan *outlier* pada variabel prediksi untuk memastikan distribusi nilai penjualan berada dalam rentang yang wajar dan dapat dipelajari oleh model. Nilai penjualan perlu dipastikan lebih dari 0 karena terdapat kasus tertentu di mana

data tercatat bernilai negatif akibat transaksi retur yang tercatat sebagai pengurangan penjualan. Selain itu, diperlukan pembersihan terhadap nilai penjualan yang melampaui batas atas atau *threshold* tertentu, sebab lonjakan penjualan yang terlalu ekstrim merupakan kejadian yang tidak bersifat reguler dan tidak merepresentasikan pola permintaan yang stabil. Penentuan *threshold* dilakukan dengan menerapkan metode *3-sigma rule*, sebuah metode statistik yang umum digunakan dalam analisis deret waktu dan peramalan karena mampu mengidentifikasi nilai yang berada jauh di luar variabilitas alami data [21].

Penentuan *threshold* dihitung menggunakan Persamaan (3.3), dengan mengombinasikan nilai rata-rata populasi (dihitung hanya dari nilai penjualan di atas 0) dan nilai tiga kali standar deviasi. Dua komponen tersebut diperoleh melalui proses perhitungan menggunakan Persamaan (3.1) dan (3.2). Hasil perhitungan menghasilkan batas atas yang berfungsi sebagai penanda nilai penjualan yang masih berada dalam pola permintaan normal. Seluruh data yang berada di atas batas tersebut dikeluarkan dari proses pelatihan model agar distribusi variabel prediksi tidak terdistorsi oleh lonjakan yang tidak mewakili kondisi operasional yang sebenarnya.

$$\mu = \frac{\sum_{i=1}^{N} X_i}{N} \tag{3.1}$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N} (X_i - \mu)}{N}} \tag{3.2}$$

$$threshold = \mu + 3\sigma \tag{3.3}$$

Rumus 3.1 Perhitungan *Threshold* dengan *3-sigma Rule*
Sumber: [22]

Hasil data yang sudah dibersihkan kemudian digunakan untuk analisis pola historis permintaan demi memahami bentuk perilaku penjualan dari setiap produk. Klasifikasi didasarkan pada variasi

jumlah permintaan dan variasi waktu permintaan pada barang tersebut. Variasi jumlah permintaan dianalisis untuk melihat seberapa besar fluktuasi kuantitas penjualan dari waktu ke waktu, sedangkan variasi waktu permintaan dievaluasi untuk mengetahui seberapa sering suatu produk mengalami penjualan dalam periode tertentu. Terdapat empat kategori pola permintaan yang digunakan dalam industri, yaitu [23]:

1. *Smooth*

   Pola ini menggambarkan kondisi permintaan yang relatif stabil sepanjang waktu. Produk dengan pola ini memiliki fluktuasi yang kecil dan aliran penjualan yang cenderung dapat diprediksi dengan mudah. Penjualan terjadi secara konsisten tanpa lonjakan atau penurunan yang ekstrem.

2. *Intermittent*

   Pola *intermittent* muncul ketika penjualan tidak terjadi secara konsisten. Pada pola ini, sering kali terdapat banyak periode di mana tidak ada transaksi sama sekali, namun ketika permintaan muncul, jumlahnya tidak terlalu besar atau ekstrem. Produk dengan pola ini memiliki frekuensi penjualan yang rendah namun masih dalam jumlah yang wajar ketika terjadi transaksi.

3. *Erratic*

   Pola ini memiliki frekuensi permintaan yang lebih sering dibandingkan pola intermittent, namun tingkat fluktuasinya jauh lebih tinggi. Volume penjualan pada pola ini sangat bervariasi dari satu periode ke periode lainnya, sehingga sulit untuk memprediksi berapa banyak unit yang akan terjual meskipun permintaan muncul secara lebih rutin. Ketidakpastian ini membuat pola *erratic* menjadi salah satu yang paling menantang untuk diramalkan.

4. *Lumpy*

   Pola *lumpy* merupakan kombinasi dari rendahnya frekuensi permintaan dan tingginya variasi jumlah ketika permintaan tersebut muncul. Produk dengan pola ini memiliki perilaku penjualan yang sangat acak dan tidak beraturan. Dalam beberapa periode mungkin tidak ada permintaan sama sekali, namun ketika permintaan tiba, jumlahnya bisa sangat besar atau sangat kecil tanpa pola yang jelas. Hal ini membuat *lumpy* menjadi pola yang paling kompleks dan membutuhkan teknik peramalan yang lebih canggih.

Setiap kategori memiliki karakteristik yang berbeda dan memerlukan pendekatan prediksi yang disesuaikan. Pengklasifikasian pola permintaan ini dilakukan untuk memastikan bahwa setiap tipe produk mendapatkan pendekatan peramalan yang paling sesuai dengan karakteristiknya. Satu metode tunggal tidak mampu menangani seluruh variasi perilaku permintaan yang muncul pada berbagai produk. Oleh karena itu, dengan memahami pola historis masing-masing produk, model prediksi dapat disesuaikan agar menghasilkan akurasi yang lebih tinggi dan relevan dengan kondisi aktual di lapangan.

$$ADI = \frac{juml \quad total\ periode\ waktu}{juml \quad periode\ dengan\ permintaan\ tidak\ nol} \tag{3.4}$$

Rumus 3.2 Perhitungan *Average Demand Interval*
Sumber: [24]

$$CV^2 = \left(\frac{\sigma}{\mu}\right)^2 \tag{3.5}$$

Rumus 3.3 Perhitungan *Coefficient of Variation*^{2}
Sumber: [24]

Keempat kategori permintaan barang dapat diklasifikasikan dengan menghitung dua metrik utama, yaitu *Average Demand Interval* (ADI) dan *Coefficient of Variation*^{2} (CV^2). ADI

menggambarkan seberapa sering permintaan muncul dalam suatu periode historis, di mana metrik ini membantu memahami apakah sebuah produk memiliki permintaan yang stabil atau justru jarang terjadi. Perolehan nilai ADI didapat dengan Persamaan (3.3), yang secara konsep menghitung rata-rata jarak antar kemunculan permintaan tidak nol sepanjang lini masa penjualan. Semakin besar nilai ADI, semakin jarang permintaan muncul, sehingga pola penjualannya cenderung tidak teratur. Sebaliknya, nilai ADI yang kecil menunjukkan bahwa permintaan relatif sering terjadi dan alurnya lebih stabil. CV^2 menggambarkan tingkat variabilitas permintaan dari waktu ke waktu. Metrik ini dihitung dengan menggunakan Persamaan (3.4), yang membandingkan besarnya variasi atau penyimpangan data (σ) terhadap nilai rata-ratanya (μ). Dalam perhitungannya, hanya titik data dengan permintaan tidak nol yang digunakan agar varians yang diperoleh benar-benar mencerminkan variasi pada saat permintaan memang terjadi. Nilai CV^2 yang tinggi menandakan adanya fluktuasi besar pada jumlah permintaan, sedangkan nilai CV^2 yang rendah menunjukkan pola yang lebih seragam. Kedua metrik tersebut digunakan sebagai standar klasifikasi yang menyatukan informasi frekuensi permintaan dan tingkat ketidakstabilannya. Standar ini tercantum pada Tabel 3.7, yang berfungsi sebagai acuan untuk menentukan apakah suatu produk termasuk kategori *smooth*, *intermittent*, *erratic*, atau *lumpy*.

Tabel 3.7 Tabel Kriteria Kategori Permintaan

Kategori	**ADI**	**CV^2**
Smooth	< 1.32	< 0.49
Intermittent	>= 1.32	< 0.49
Erratic	< 1.32	>= 0.49
Lumpy	>= 1.32	>= 0.49

Setiap kategori memerlukan standar penanganan yang berbeda untuk memastikan metode peramalan yang digunakan dapat

menyesuaikan karakteristik datanya. Penentuan metode prediksi dilakukan melalui diskusi bersama tim dan pengguna, dengan mempertimbangkan pola historis serta tingkat variasi pada masing-masing kategori. Kombinasi kategori *smooth* dan *erratic* dinilai memiliki pola yang relatif lebih terbaca, sehingga tetap menggunakan metode peramalan berbasis BigQuery ML seperti rencana awal. Metode ini dianggap memadai karena kedua kategori tersebut memiliki karakteristik pergerakan data yang masih dapat ditangkap oleh model deret waktu tradisional, terutama ketika variasi jumlah permintaan tidak terlalu ekstrem atau tidak terjadi jeda permintaan yang terlalu panjang.

Sebaliknya, kategori *intermittent* dan *lumpy* menunjukkan perilaku yang jauh lebih tidak beraturan akibat rendahnya frekuensi permintaan serta tingginya tingkat variasi penjualan ketika permintaan muncul. Pola tersebut menyebabkan metode peramalan berbasis model deret waktu menjadi kurang efektif, karena model sering gagal mengidentifikasi struktur pola yang dapat dipelajari secara konsisten. Solusi yang dipilih untuk menangani kedua kategori ini adalah penggunaan metode *trimmean*. *Trimmean* merupakan teknik statistik yang menghitung nilai rata-rata dengan menghilangkan masing-masing 10 persen nilai ekstrim yang berada pada bagian atas dan bawah distribusi. Metode ini memberikan nilai representatif yang lebih stabil dibandingkan rata-rata biasa, terutama untuk data dengan lonjakan permintaan yang tidak berulang serta interval penjualan yang tidak menentu. Hasil perhitungan *trimmean* kemudian digunakan sebagai nilai prediksi untuk seluruh kombinasi produk dan toko yang termasuk dalam kategori *intermittent* dan *lumpy*. Seluruh proses analisis ini menjadi tahap akhir dalam penyusunan data pelatihan, sekaligus memastikan bahwa setiap pola permintaan memperoleh pendekatan prediksi yang paling sesuai dengan karakteristiknya.

3.3.1.9 Membuat dan melatih model prediksi

Pembuatan model prediksi dilakukan menggunakan fungsi ML.FORECAST() pada BigQuery, yang telah terbukti bahwa pendekatan ini lebih sesuai untuk kebutuhan perencanaan inventaris berdasarkan hasil eksplorasi sebelumnya. Model yang dikembangkan adalah ARIMA_PLUS_XREG, sementara model pembanding ARIMA_PLUS dikerjakan oleh anggota tim lainnya. Model ARIMA_PLUS_XREG dipilih karena memiliki kemampuan untuk melibatkan variabel regresor eksternal yang relevan dengan konteks bisnis, sehingga hasil prediksi lebih mencerminkan kondisi aktual.

Tabel 3.8 Tabel Contoh Data Pelatihan Model ARIMA_PLUS_XREG

period	store	article	quantity sold	remaining stock lagged	outlier	demand category
2023-01-01	S1	122	10	12	true	Lumpy
2023-01-01	S2	286	70	75	false	Smooth
2023-02-01	S1	122	2	15	false	Lumpy
2023-02-01	S2	286	180	100	true	Smooth
2023-03-01	S1	122	3	4	false	Lumpy
2023-03-01	S2	286	60	120	false	Smooth
2023-04-01	S1	122	5	6	false	Lumpy
2023-04-01	S2	286	75	80	false	Smooth
2023-05-01	S1	122	1	2	false	Lumpy
2023-05-01	S2	286	84	90	false	Smooth

Data yang digunakan sebagai bahan pelatihan berasal dari kombinasi informasi penjualan dan stok di setiap toko untuk setiap produk, dengan rentang waktu dua tahun, mulai dari tahun 2023 hingga 2025. Rentang waktu ini dianggap representatif karena mampu menangkap pola musiman dan fluktuasi permintaan yang bervariasi antarperiode. Frekuensi data ditetapkan per bulan, bukan per hari, untuk menekan biaya komputasi dan menjaga efisiensi pemrosesan, mengingat ukuran data transaksi harian bisa sangat besar. Struktur data pelatihan pada Tabel 3.8 terdiri atas 7 kolom dan data riil sebanyak 76 juta baris. Kolom store dan article menyimpan informasi

mengenai ID toko dan produk pada periode tersebut, kemudian kolom outlier menandakan apakah data tersebut melebihi nilai *threshold* untuk kombinasi produk dan toko tersebut atau tidak. Kolom demand_category menunjukkan tipe permintaan dari data terkait, sesuai dengan hasil analisis pelatihan data sebelumnya. Model dilatih berdasarkan data tersebut menggunakan perintah SQL di BigQuery, di mana quantity_sold dijadikan variabel target dan remaining_stock_lagged sebagai regresor eksternal sesuai dengan analisis data histori sebelumnya. Setiap kombinasi produk dan toko akan membentuk satu model prediksi yang berdiri sendiri untuk menangkap pola unik pada masing-masing entitas.

```
CREATE OR REPLACE MODEL
`project.dataset.forecast_model_arima_plus_xreg`
OPTIONS (
    MODEL_TYPE = 'ARIMA_PLUS_XREG',
    TIME_SERIES_TIMESTAMP_COL = 'period',
    TIME_SERIES_DATA_COL = 'quantity_sold',
    TIME_SERIES_ID_COL = ['Store', 'Article'],
    AUTO_ARIMA = TRUE,
    DATA_FREQUENCY = 'MONTHLY',
    HOLIDAY_REGION = 'ID',
    CLEAN_SPIKES_AND_DIPS = TRUE
) AS
select period,Store,Article,quantity_sold,remaining_stock_lagged from `project.dataset.training_data_xreg`
where period <= '2025-07-31'
```

Gambar 3.33 *Query* Pembuatan Model ARIMA_PLUS_XREG pada BigQuery ML

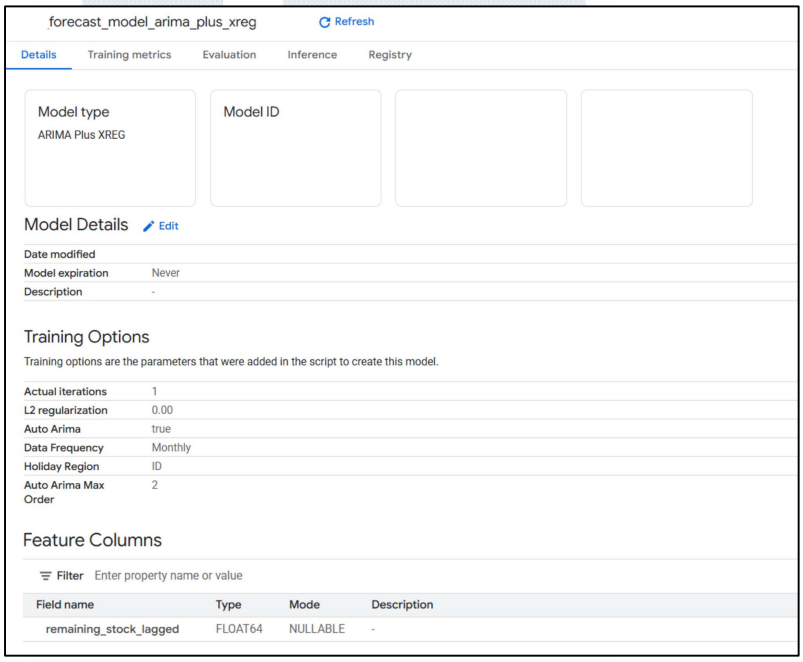
Gambar 3.33 menunjukan merupakan *query* untuk membuat atau model *machine learning* berbasis ARIMA_PLUS_XREG, yang digunakan untuk melakukan peramalan deret waktu dengan mempertimbangkan faktor eksternal. Pengaturan untuk menentukan cara model dibangun dan dilatih dapat ditentukan pada bagian OPTIONS. Pemilihan model ARIMA_PLUS_XREG dapat didefinisikan pada parameter MODEL_TYPE. Selanjutnya, TIME_SERIES_TIMESTAMP_COL bertujuan menentukan kolom yang berperan sebagai penanda waktu pada data. Kolom ini wajib berisi nilai tanggal atau waktu yang berurutan, karena model akan menggunakan urutan waktu ini untuk mempelajari pola historis. Kemudian parameter TIME_SERIES_DATA_COL menunjukkan kolom yang menjadi target prediksi, yaitu jumlah unit terjual

(quantity_sold) yang akan diramalkan pada periode mendatang. Sedangkan TIME_SERIES_ID_COL berfungsi sebagai pengenal unik yang membedakan setiap seri waktu, yakni kolom Store dan Article. Dengan parameter ini, BigQuery akan membangun model terpisah untuk setiap kombinasi toko dan produk, sehingga hasil peramalan lebih spesifik.

Parameter AUTO_ARIMA yang diatur TRUE membuat sistem secara otomatis mencari kombinasi parameter terbaik untuk model ARIMA. Proses otomatis ini menghemat waktu pengguna karena tidak perlu melakukan pencarian parameter secara manual. Selanjutnya, DATA_FREQUENCY diatur sebagai data bulanan, di mana setiap nilai period merepresentasikan satu bulan, sesuai dengan data pelatihan yang telah dibuat. Hal ini bertujuan agar model memahami interval waktu antarobservasi secara tepat. Terdapat juga opsi HOLIDAY_REGION, di mana opsi ini menginstruksikan BigQuery ML untuk memasukkan informasi hari libur nasional Indonesia dengan kode 'ID' ke dalam proses peramalan, sehingga model dapat mengenali dampak potensi kenaikan atau penurunan penjualan yang terjadi akibat hari libur tertentu, seperti Lebaran atau Natal. Selain itu, parameter CLEAN_SPIKES_AND_DIPS berfungsi untuk menentukan apakah model ingin diatur atau tidak untuk mendeteksi dan menyesuaikan data yang mengandung anomali ekstrem, seperti lonjakan atau penurunan tajam yang tidak wajar dalam periode tertentu. Hal ini membantu menjaga kestabilan model dan mencegah hasil prediksi terpengaruh oleh nilai-nilai ekstrem yang bersifat *outlier*. Proses pembersihan ini tidak menghapus data, tetapi melakukan penyesuaian agar model tidak terlalu sensitif terhadap perubahan mendadak yang tidak merepresentasikan tren sebenarnya.

Setelah penentuan opsi parameter, pembuatan model diakhiri dengan *query* SELECT yang mendefinisikan data pelatihan yang akan

digunakan untuk membangun model. Query tersebut mengambil data pelatihan, dengan kolom remaining_stock_lagged. Data yang diambil pada tabel data pelatihan difilter dengan mengambil data di bawah tanggal 31 Juli 2025, sebab tanggal setelah itu akan digunakan sebagai data pengujian, sehingga model tidak "melihat" data masa depan yang seharusnya diprediksi nantinya. Setelah *query* pembuatan model selesai disusun, *query* tersebut dieksekusi dan berjalan selama kurang lebih 40 menit hingga model selesai dibuat. Hasil dari eksekusi *query* pembuatan model dapat dilihat pada Gambar 3.34.



Gambar 3.34 Hasil Pembuatan Model ARIMA_PLUS_XREG

3.3.1.10 Menguji dan membandingkan hasil evaluasi kinerja model

Setelah proses pelatihan model selesai, dilakukan pengujian dan evaluasi kinerja model menggunakan fungsi ML.EVALUATE di BigQuery. Evaluasi dilakukan untuk menilai seberapa baik model mampu melakukan prediksi serta seberapa baik model menyesuaikan diri terhadap pola data historis. Dalam tahap ini, terdapat dua kategori utama yang digunakan, yaitu akurasi model dan kesesuaian model.

Untuk kategori akurasi, metrik yang digunakan adalah *Mean Absolute Percentage Error* (MAPE) dan *Root Mean Square Error* (RMSE), di mana MAPE mengukur tingkat kesalahan prediksi dalam persentase terhadap nilai aktual, sedangkan RMSE menunjukkan seberapa besar rata-rata deviasi hasil prediksi terhadap nilai sebenarnya dalam satuan asli data. Kedua metrik ini diperoleh melalui proses *backtesting* dengan menggunakan data pengujian yang diambil dari sebagian data pelatihan pada rentang waktu Agustus hingga Oktober 2025, agar dapat menggambarkan performa model terhadap data terbaru. Pada Gambar 3.35, nilai MAPE dan RMSE dari setiap kombinasi model dan dataset akan dihitung, lalu dirata-rata untuk memperoleh hasil evaluasi keseluruhan pada kategori akurasi.

```sql
-- Model Accuracy
SELECT
    AVG(mean_absolute_percentage_error) AS average_mape,
    AVG(root_mean_squared_error) AS average_rmse
FROM
    ML.EVALUATE(
        MODEL `project.dataset.forecast_model_arima_plus_xreg`,
        (
            SELECT period, Store, Article, quantity_sold, remaining_stock_lagged
            FROM `project.dataset.training_data_xreg`
            WHERE period BETWEEN '2025-08-01' AND '2025-10-31' -- Test window (3 months)
        )
    );

-- Model Fit/Statistical Quality
SELECT
    AVG(log_likelihood) AS avg_log_likelihood,
    AVG(AIC) AS avg_AIC,
    AVG(variance) AS avg_variance,
    COUNTIF(AIC IS NOT NULL AND NOT (IS_INF(AIC) OR IS_NAN(AIC))) AS successful_fit_count
FROM
    ML.EVALUATE(MODEL `project.dataset.forecast_model_arima_plus_xreg`)
WHERE -- Filter null/infinity/nan
    AIC IS NOT NULL
    AND NOT (IS_INF(AIC) OR IS_NAN(AIC))
    AND NOT (IS_INF(log_likelihood) OR IS_NAN(log_likelihood))
    AND NOT (IS_INF(variance) OR IS_NAN(variance))
```

Gambar 3.35 *Query* Pengujian dan Evaluasi Model ARIMA_PLUS_XREG

Sementara itu, kategori kesesuaian model dievaluasi dengan metrik statistik seperti *log-likelihood*, *Akaike Information Criterion* (AIC), dan varians. Ketiga metrik ini sudah disediakan secara otomatis oleh BigQuery ML setelah model dijalankan melalui fungsi ML.EVALUATE, sehingga tidak memerlukan perhitungan manual. *Log-likelihood* menggambarkan seberapa baik model menjelaskan data aktual, AIC menilai keseimbangan antara kompleksitas dan

kecocokan model, sedangkan varians memberikan gambaran tentang stabilitas hasil prediksi terhadap variasi data. Untuk memastikan hanya model yang terlatih dengan baik yang dihitung dalam evaluasi, dilakukan proses filtrasi terhadap data hasil evaluasi, dengan hanya menyertakan baris yang memiliki nilai *log-likelihood*, AIC, dan varians yang valid (tidak bernilai *null*, tidak terhingga, maupun NaN). Dari hasil ini, dapat diketahui berapa banyak kombinasi data yang berhasil di-*fit* dengan baik, sekaligus menjadi indikator seberapa stabil performa model terhadap berbagai variasi data produk dan toko. Hasil evaluasi dari kedua kategori tersebut kemudian dibandingkan antara model ARIMA_PLUS dan ARIMA_PLUS_XREG untuk menentukan model yang paling akurat dan efisien digunakan dalam proyek perencanaan inventaris perusahaan.

Tabel 3.9 Tabel Perbandingan Evaluasi Model BigQuery ML

Metrik	**ARIMA_PLUS_XREG**	**ARIMA_PLUS**
MAPE	9.63%	9.82%
RMSE	14.02	4.27
Log-likelihood	-7.16	10.74
AIC	79.76	13.99
Varians	33.22	0.17
Persentase validasi *fitting* model	87.1%	81.5%
Estimasi biaya eksekusi per bulan	347.22 GB	111.52 GB

Berdasarkan hasil perbandingan pada Tabel 3.9, dapat disimpulkan bahwa model ARIMA_PLUS menunjukkan performa yang lebih baik dibandingkan ARIMA_PLUS_XREG pada sebagian besar aspek evaluasi. Pada metrik MAPE, nilai kedua model memiliki selisih yang sangat sedikit, yakni sebesar 0.19%, dengan model ARIMA_PLUS_XREG sebagai model yang lebih unggul. Namun, untuk metrik RMSE, ARIMA_PLUS memiliki nilai sebesar 4.27, yang jauh lebih rendah dibandingkan model ARIMA_PLUS_XREG

yang mencapai 14.02. Dari hasil evaluasi kategori akurasi model, model ARIMA_PLUS menunjukkan nilai MAPE yang lebih tinggi tetapi RMSE yang lebih rendah, yang artinya model ini membuat kesalahan absolut yang lebih kecil secara rata-rata, namun kesalahan tersebut lebih besar relatif terhadap nilai aktual. Sebaliknya, model ARIMA_PLUS_XREG menampilkan MAPE yang lebih rendah dengan RMSE yang lebih tinggi, yang menunjukkan bahwa meskipun kadang membuat kesalahan individu yang lebih besar, kesalahan tersebut lebih kecil relatif terhadap apa yang diprediksi. Hal ini dipengaruhi oleh adanya regresor eksternal dalam model ARIMA_PLUS_XREG yang dapat menangkap pola penting yang tidak tertangkap model ARIMA_PLUS, sehingga membantu model melacak nilai aktual dengan lebih baik secara proporsional.

Dari sisi kualitas model, ARIMA_PLUS lebih stabil dengan nilai *log-likelihood* (10.74) dan AIC (13.99) yang jauh lebih baik dibandingkan ARIMA_PLUS_XREG yang memiliki *log-likelihood* negatif (-7.16) dan AIC tinggi (79.76). Nilai *log-likelihood* yang lebih tinggi serta AIC yang lebih rendah menunjukkan bahwa model ARIMA_PLUS mampu menjelaskan pola data historis dengan efisiensi yang lebih baik tanpa menimbulkan kompleksitas berlebihan. Selain itu, varians model ARIMA_PLUS yang sangat kecil (0.17) dibandingkan dengan ARIMA_PLUS_XREG (33.22) menunjukkan tingkat kestabilan prediksi yang jauh lebih tinggi dan tidak terlalu sensitif terhadap fluktuasi data. Sebaliknya, dari sisi validasi *fitting* model, ARIMA_PLUS_XREG menunjukkan tingkat keberhasilan yang lebih tinggi, yakni sebesar 87.1%, dibandingkan ARIMA_PLUS yang hanya mencapai 81.5%. Hal ini berarti model ARIMA_PLUS_XREG mampu menyesuaikan diri dengan data historis lebih baik pada sebagian besar kombinasi produk dan toko, yang dipengaruhi oleh adanya faktor regresor eksternal yang membantu model memahami konteks stok dan penjualan secara lebih

menyeluruh. Namun, tingginya rasio keberhasilan *fitting* tidak selalu berarti performa prediktif yang baik, terlebih ketika *output* model menunjukkan banyak nilai prediksi yang mendekati nol. Selain performa teknis, faktor efisiensi biaya juga penting untuk dipertimbangkan di BigQuery. ARIMA_PLUS menghabiskan estimasi eksekusi sebesar 111.52 GB per bulan, yang jauh lebih ringan dibandingkan ARIMA_PLUS_XREG yang mencapai 347.22 GB. Hal ini disebabkan karena adanya penambahan faktor regresor eksternal pada data pelatihan ARIMA_PLUS_XREG, yang mengakibatkan perbedaan selisih yang cukup besar pada total eksekusi.

Hasil evaluasi dari kategori akurasi dan kualitas model, hasil menunjukkan bahwa regresor eksternal pada ARIMA_PLUS_XREG mengalami *overfitting*, di mana model lebih bersifat menghafalkan pola prediksi dibandingkan memahaminya, sehingga ketika terdapat data baru, performa model tersebut akan menurun. Meskipun ARIMA_PLUS_XREG memiliki potensi untuk memasukkan faktor bisnis tambahan melalui regresor eksternal, hasil pengujian menunjukkan bahwa model dasar ARIMA_PLUS lebih stabil, efisien, dan akurat untuk kasus prediksi penjualan bulanan dengan pola yang relatif sederhana. Kesimpulannya, model ARIMA_PLUS menjadi pilihan yang lebih tepat untuk digunakan dalam tahap awal implementasi sistem peramalan inventaris.

3.3.1.11 Mengimplementasikan model prediksi dan integrasi *output*

Model ARIMA_PLUS yang telah dikembangkan diimplementasikan secara terjadwal untuk memperbarui hasil prediksi setiap satu bulan sekali, di mana model dijalankan mengikuti jadwal yang sudah ditentukan dan menghasilkan *output* prediksi untuk setiap kombinasi toko dan produk. Hasil prediksi tersebut ditampilkan pada Tabel 3.10 yang memuat contoh nilai kebutuhan barang untuk periode

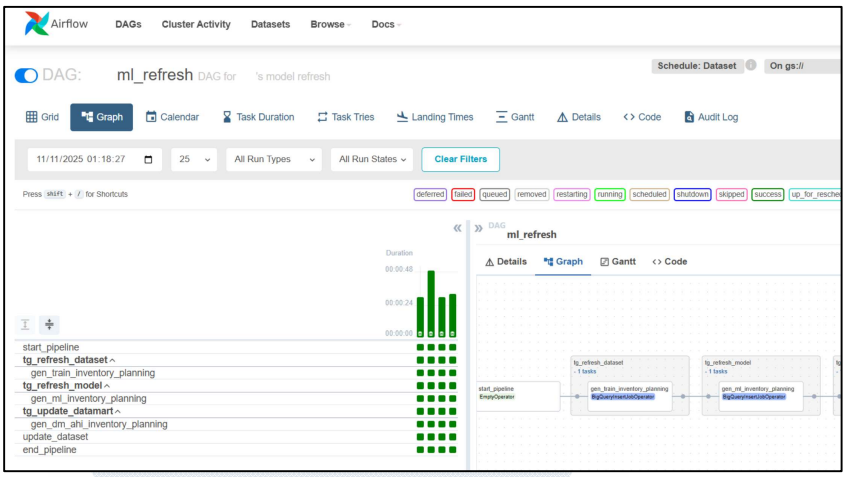
November 2025. *Output* ini menjadi dasar bagi estimasi permintaan setiap bulan dan akan digunakan oleh tim terkait untuk perencanaan stok. Prediksi tersebut dimasukkan ke dalam kolom average_demand dan total_average_demand pada *data mart* yang diperbarui secara harian. Tujuannya adalah agar data permintaan tetap selalu *up-to-date* dan dapat langsung dimanfaatkan oleh *user* tanpa perlu menjalankan proses secara manual.

Tabel 3.10 Tabel Hasil Prediksi Permintaan Barang Bulan November 2025

store	article	forecast_period	forecast_value
S1	122	2025-11-01	10
S1	286	2025-11-01	5
S1	347	2025-11-01	45
S2	122	2025-11-01	13
S2	286	2025-11-01	48
S2	347	2025-11-01	35
S3	756	2025-11-01	7
S3	640	2025-11-01	15
S4	756	2025-11-01	6
S4	640	2025-11-01	23

Proses pembaruan model dan *data mart* diotomatisasi melalui *Directed Acyclic Graph* (DAG) Apache Airflow pada Gambar 3.36, yang menjadi orkestrator utama untuk seluruh *pipeline* data di tim *Data Engineer*. Dalam DAG tersebut terdapat beberapa *task* yang disusun berurutan, meliputi pembuatan data pelatihan berdasarkan periode dua tahun terakhir hingga bulan berjalan, pembuatan ulang model, eksekusi model prediksi, dan pembaruan *data mart*. Karena proses DAG dijalankan setiap hari, maka konfigurasi dibuat agar task pelatihan data dan pembuatan model hanya dieksekusi setiap tanggal 1 setiap bulannya, sedangkan *task* pembaruan *data mart* tetap berjalan harian mengikuti jadwal normal setelah DAG dependensi seperti DAG *data warehouse* selesai dijalankan. Integrasi model ke dalam *pipeline* berjalan otomatis, efisien, dan memastikan seluruh hasil

prediksi selalu sinkron dengan data terbaru tanpa intervensi manual dari tim. *Data mart* yang telah dieksekusi secara otomatis kemudian disambungkan ke *dashboard* pada Looker Studio dan dapat digunakan oleh *user*.



Gambar 3.36 DAG Harian untuk Pembaruan Data Perencanaan Inventaris

Implementasi model prediksi ini memberikan dampak langsung terhadap proses operasional dan analitik perencanaan inventaris di perusahaan. Adanya estimasi permintaan yang diperbarui secara rutin dan terintegrasi ke dalam *data mart* membuat tim bisnis tidak lagi bergantung pada perhitungan manual atau asumsi subjektif dalam menentukan kebutuhan stok. Proses pengambilan keputusan menjadi lebih cepat dan berbasis data, terutama dalam mengantisipasi fluktuasi permintaan antar toko dan produk. Selain itu, otomasi melalui Airflow mengurangi risiko kesalahan akibat intervensi manual serta menurunkan beban kerja operasional tim, karena seluruh alur prediksi hingga penyajian data berjalan konsisten dan terstandarisasi. Ketersediaan hasil prediksi pada *dashboard* juga memperluas pemanfaatan model, karena wawasan permintaan dapat diakses oleh berbagai pihak terkait sebagai dasar perencanaan pengadaan, distribusi, dan evaluasi performa penjualan.

3.3.2 Kendala yang Ditemukan

Selama menjalani kegiatan kerja magang, terdapat beberapa kendala teknis dan operasional yang cukup menantang dalam proses optimasi sistem katalog produk serta implementasi model prediksi perencanaan inventaris. Berbagai hambatan ini memerlukan identifikasi yang cermat agar tidak mengganggu kelancaran pelaksanaan proyek. Kendala-kendala tersebut muncul baik dari sisi keterbatasan platform yang digunakan, kapasitas komputasi, hingga kompleksitas struktur data yang harus diolah, di antaranya:

1. Lingkungan *default* Google Colab tidak memiliki kapasitas yang cukup untuk melakukan proses *embedding* dalam jumlah besar, karena keterbatasan memori dan waktu eksekusi yang sering menyebabkan proses terhenti di tengah jalan.
2. Biaya eksekusi BigQuery yang tinggi saat melakukan pelatihan dan implementasi model prediksi, terutama apabila dijalankan dengan frekuensi yang terlalu sering.
3. Kesulitan dalam mengidentifikasi atribut produk yang berstatus *discontinue* atau tidak, disebabkan oleh struktur data sumber yang tidak seragam dan jumlah kolom yang sangat banyak sehingga mempersulit proses filterisasi atribut yang relevan bagi AI katalog produk.

3.3.3 Solusi atas Kendala yang Ditemukan

Setiap kendala yang dialami selama pelaksanaan magang dianalisis untuk menemukan solusi yang efisien tanpa mengganggu jalannya proyek utama. Kendala yang muncul diatasi melalui pendekatan kolaboratif, yakni dengan komunikasi intensif bersama tim teknis dan rekan tim. Beberapa solusi yang diterapkan antara lain sebagai berikut:

1. Mengubah tipe *hardware accelerator* pada Google Colab menjadi versi v5e-1 CPU, yang dirancang untuk mendukung pemrosesan berbasis AI dan memiliki kapasitas *runtime* yang lebih besar. Selain itu, proses *embedding* dilakukan secara bertahap agar dapat

menghindari kegagalan eksekusi akibat beban komputasi yang terlalu tinggi ketika dilakukan sekaligus.

2. Melakukan diskusi dengan pihak user untuk meninjau ulang frekuensi pelaksanaan prediksi. Berdasarkan hasil kesepakatan, proses prediksi dilakukan setiap satu bulan sekali guna menjaga efisiensi biaya dan tetap memenuhi kebutuhan analisis yang relevan.
3. Mengadakan pertemuan dengan tim *development* yang bertanggung jawab terhadap sumber data, untuk memperoleh daftar kolom serta kondisi logis yang digunakan dalam menentukan status produk *discontinue* atau tidak sehingga membantu mempercepat proses integrasi data.