BAB IV

HASIL PENELITIAN DAN PEMBAHASAN

4.1. *Emphatize*

Pada tahapan ini, penulis melakukan serangkaian aktivitas untuk mengungkap pemahaman komprehensif tentang pengalaman dan persepsi pengguna terhadap *dashboard* pemantauan harga menggunakan *Superset* yang dikoneksikan langsung ke *database* SP2KP. Tahap awal dilaksanakan melalui observasi langsung terhadap penggunaan *dashboard* dalam berbagai skenario operasional, guna mengidentifikasi kendala dan hambatan yang dialami pengguna. Observasi ini difokuskan untuk memperoleh insight mengenai mekanisme operasional *dashboard* dan hubungannya yang langsung terkoneksi dengan *database live* . Berikut kuesioner yang disampaikan terkait *dashboard Superset* yang masih menggunakan *database production* (*live*) sebagai sumber data utama.

Tabel IV. 1. Kuesioner SUS *database production (live)*

No	Pernyataan
1	Saya merasa nyaman menggunakan *dashboard* ini.
2	Saya merasa *dashboard* ini rumit untuk digunakan.
3	*Dashboard* ini mudah digunakan.
4	Saya membutuhkan bantuan teknis untuk bisa menggunakan *dashboard* ini.
5	Fitur-fitur dalam *dashboard* ini berjalan sesuai harapan.
6	Saya merasa banyak hal yang membingungkan saat menggunakan *dashboard* ini.
7	*Dashboard* ini sangat fungsional dan cepat.
8	Saya merasa *dashboard* ini tidak konsisten dan sulit diprediksi.

9	Saya merasa percaya diri saat menggunakan *dashboard* ini.
10	Saya perlu mempelajari banyak hal sebelum bisa menggunakan *dashboard* ini.

Setelah data kuesioner terkumpul dari responden, langkah selanjutnya adalah mengolah data tersebut menggunakan rumus *System Usability Scale* (SUS). Pertama, skor untuk pernyataan bernomor ganjil dikurangi 1, sedangkan pernyataan bernomor genap dikurangi 5. Selanjutnya, total skor dari pernyataan-pernyataan dihitung dan dikalikan dengan 2,5 untuk mendapatkan skor SUS akhir. Tahap ini dilakukan secara individual untuk setiap responden, sehingga setiap orang memiliki skor SUS yang merefleksikan penilaian dari mereka terhadap kegunaan *dashboard*. Melalui metode ini, peneliti dapat memahami tingkat kegunaan *dashboard* yang terhubung *database production (live)* berdasarkan perspektif pengguna yang telah memberikan respon/tanggapan.

Tabel IV. 2. Hasil perhitungan dengan metode SUS

Respondent ID	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	SUS Score
Orang 1	5	5	5	5	5	5	5	5	5	5	50
Orang 2	4	2	4	2	3	2	4	2	4	2	72.5
Orang 3	2	2	4	4	4	4	4	3	3	3	52.5
Orang 4	3	4	2	4	2	4	2	5	3	4	27.5
Orang 5	3	3	3	1	3	3	2	1	3	3	57.5
Orang 6	3	4	3	3	2	3	2	3	2	2	42.5
Orang 7	3	3	3	3	3	3	3	3	3	3	50
Orang 8	3	4	2	4	3	3	3	3	3	4	40
Orang 9	5	4	4	2	5	1	4	1	4	3	77.5
Orang 10	2	4	2	4	3	4	2	3	3	4	32.5
Rata-Rata											**50.25**

Dengan jumlah skor SUS 50,25 *dashboard* yang diuji termasuk dalam kategori *marginal low,* kurang memuaskan namun masih dapat diterima, tetapi memiliki masalah kegunaan yang cukup nyata, sehingga pengguna mungkin merasa kurang puas

atau mengalami kesulitan dalam beberapa fitur. Masih terdapat ruang untuk perbaikan pada beberapa aspek untuk dapat lebih meningkatkan kenyamanan penggunaan dan efisiensi.

Selanjutnya, penulis melaksanakan wawancara dengan responden analis data untuk mengeksplorasi kebutuhan spesifik, ekspektasi, serta kesulitan dalam pengalaman mengolah data. Proses pengumpulan data ini menghasilkan beberapa temuan kritis yang memerlukan perbaikan diantaranya ketergantungan pada *query* langsung ke *database production* yang menyebabkan penurunan performa saat analis menjalankan *query* kompleks pada *dashboard Superset* dan keterbatasan data historis akibat tidak adanya data *warehouse* untuk menyimpan *time-series* data harga secara terstruktur. Temuan ini menjadi dasar perancangan solusi ETL dengan *Apache Airflow* untuk memisahkan beban analitik dari *database* operasional.

4.2. *Define*

Tantangan utama dalam sistem pemantauan harga saat ini terletak pada arsitektur data dan proses analitik, terutama karena struktur data yang tidak terpusat sehingga menyulitkan pembuatan laporan. *Dashboard* saat ini bergantung pada *query* langsung ke DB *production*, yang menyebabkan keterlambatan (*latency*) dan beban berlebih pada sistem. Untuk mengatasi hal tersebut, penelitian ini mengusulkan pembangunan data *warehouse* terpisah dengan model terstruktur yang dirancang khusus untuk kebutuhan analisis pemantauan harga. Selain itu, diimplementasikan pula *pipeline* ETL menggunakan *Apache Airflow* untuk mengotomatiskan proses pengumpulan data serta pembaruan data secara terjadwal ke dalam data *warehouse*, sehingga sistem pelaporan menjadi lebih cepat, stabil, dan tidak mengganggu kinerja sistem produksi.

4.3. *Ideate*

Berdasarkan analisis kebutuhan dari tahap *Empathize* dan *Define*, penulis mengidentifikasi bahwa stakeholder pemantauan harga membutuhkan:

1. Ketersediaan data terpusat untuk analisis harga, menggantikan ketergantungan pada *query* langsung ke *database production*.
2. Pembaruan data yang konsisten dan terjadwal guna meminimalkan *latency* dalam pelaporan.
3. Fleksibilitas dalam pelacakan histori harga (contoh: perbandingan harga mingguan/bulanan).

Konsep solusi yang diusulkan dalam penelitian ini adalah membangun ETL *pipeline* menggunakan *Apache Airflow* yang mencakup proses ekstraksi data harga, dilanjutkan dengan tahap transformasi yang melibatkan pembersihan data (*data cleansing*), agregasi data, serta pelacakan perubahan data historis (*transform)*. Data yang telah diproses kemudian dimuat (*load*) ke dalam data *warehouse* terpisah, misalnya menggunakan *PostgreSQL*. Selanjutnya, *dashboard* visualisasi menggunakan *Apache Superset* akan terintegrasi langsung dengan data *warehouse* tersebut, sehingga dapat menghasilkan performa *query* yang lebih cepat dan stabil tanpa membebani sistem operasional utama.

4.4. *Prototype*

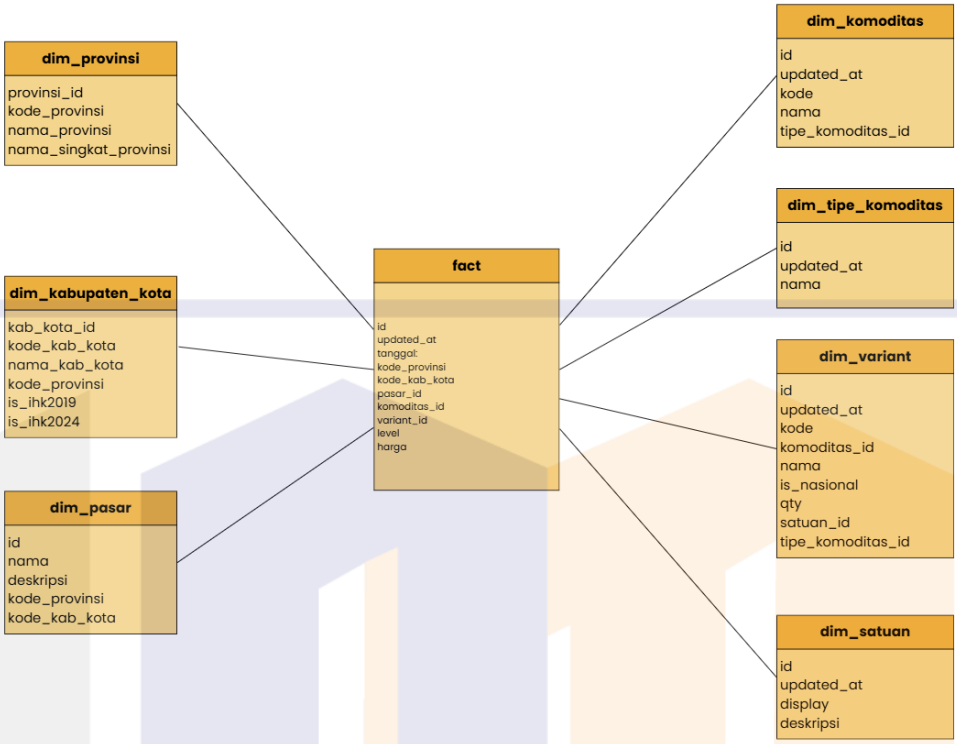
Setelah merumuskan solusi berdasarkan pendekatan *design thinking*, penulis menyusun *prototype* untuk menjelaskan gambaran alur kerja sistem ETL yang akan diterapkan pada proses pemantauan harga. *Prototype* ini tidak berfokus pada antarmuka pengguna (*user interface*), melainkan pada perancangan arsitektur sistem

dan alur data yang mendasari implementasi ETL dan pemisahan antara *database production* dan data *warehouse*.

Peneliti melakukan perancangan prototipe sistem data *warehouse* sebagai bagian dari solusi arsitektur data yang terstruktur untuk mendukung analisis harga kebutuhan pokok. *Prototype* ini mencakup perancangan skema data *warehouse*, implementasi *workflow* ETL menggunakan *Apache Airflow*, bahasa pemrograman *python* dan integrasi dengan *dashboard* visualisasi menggunakan *Apache Superset*.

4.4.1 Perancangan model skema bintang (*star schema*)

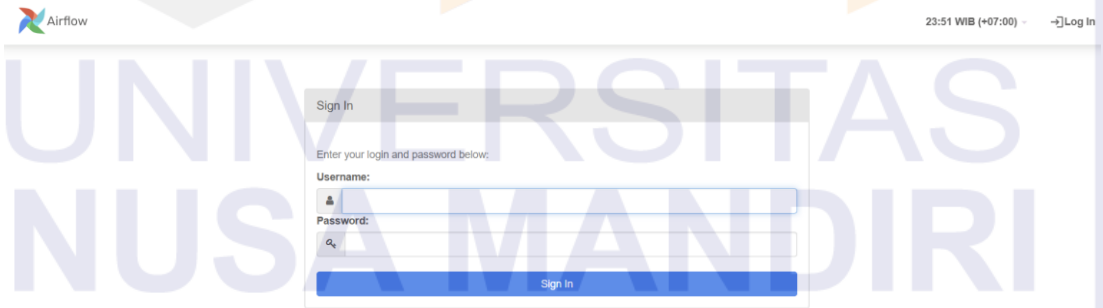
Langkah awal dalam proses perancangan adalah membuat model skema bintang (*star schema*), yang terdiri dari satu tabel fakta dan beberapa tabel dimensi. Tabel fakta menyimpan data harga komoditas dari berbagai wilayah dan waktu, sedangkan tabel dimensi meliputi dimensi waktu, komoditas, lokasi (provinsi/kabupaten), dan jenis pasar. Pemilihan *star schema* dilakukan karena struktur ini mendukung *query* analitik secara efisien dan memudahkan agregasi data.



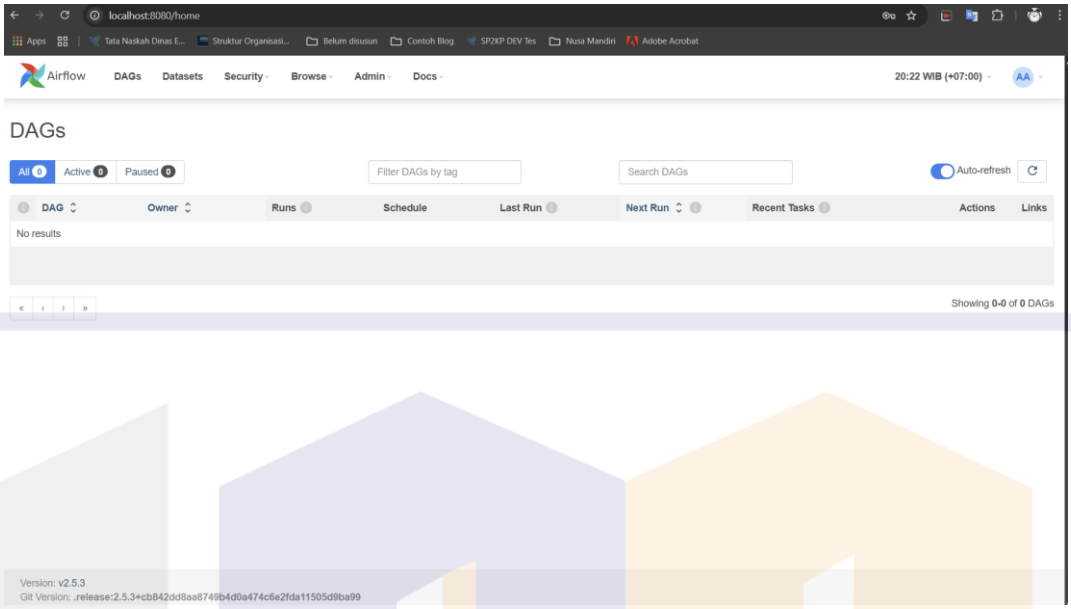
Gambar IV. 1. Model *star schema* data *warehouse*

4.4.2. *Pipeline* dengan *Apache Airflow*

Setelah struktur data dirancang dengan menggunakan pendekatan *star schema*, tahap selanjutnya adalah membangun *pipeline* ETL (*Extract, Transform, Load*) dengan memanfaatkan *Apache Airflow*.



Gambar IV. 2. Proses login *Airflow*



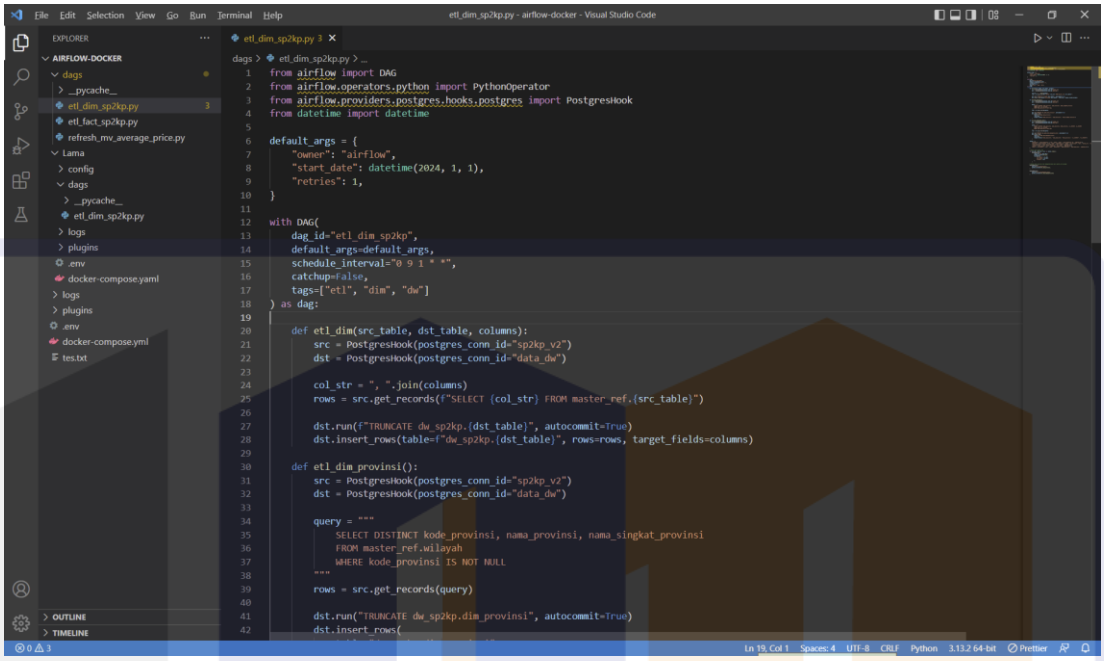
Gambar IV. 3. Tampilan awal *Airflow*

Gambar IV. 4. Pembuatan koneksi *Airflow*

4.4.3. Pemrograman DAG dengan *Python*

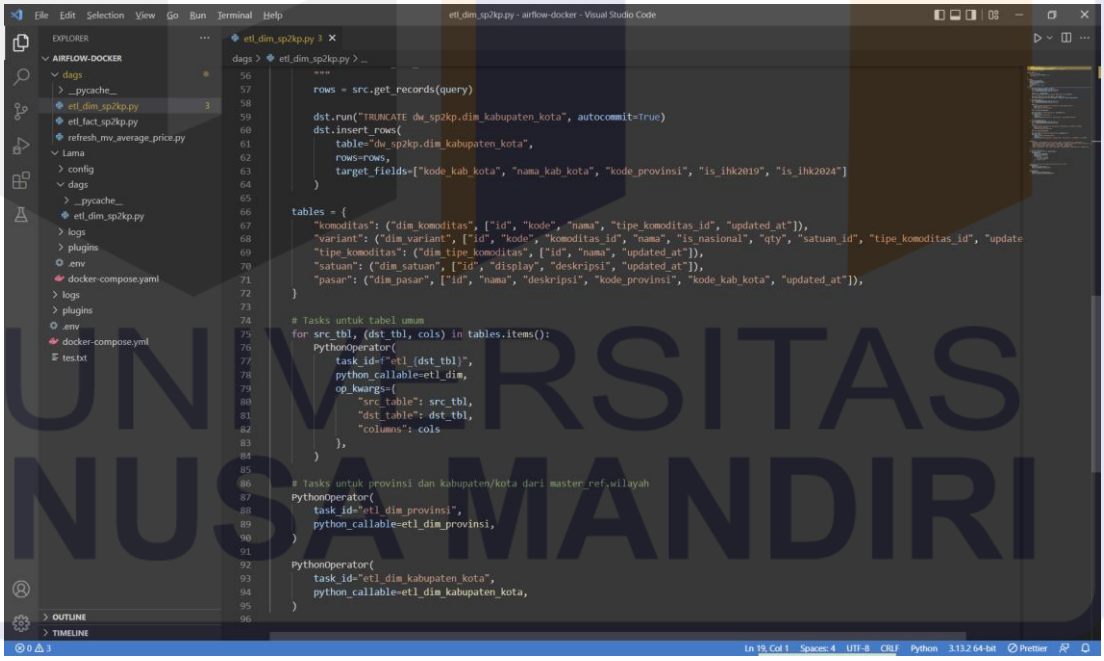
Dengan menggunakan pemrograman *Python*, disusun script pemrograman untuk proses *Directed Acyclic Graph* (DAG). Ada 3 program DAG yang disusun, yaitu:

a. etl_dim_sp2kp.py



```python
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.providers.postgres.hooks.postgres import PostgresHook
from datetime import datetime

default_args = {
    "owner": "airflow",
    "start_date": datetime(2024, 1, 1),
    "retries": 1,
}

with DAG(
    dag_id="etl_dim_sp2kp",
    default_args=default_args,
    schedule_interval="0 9 1 * *",
    catchup=False,
    tags=["etl", "dim", "dw"]
) as dag:

    def etl_dim(src_table, dst_table, columns):
        src = PostgresHook(postgres_conn_id="sp2kp_v2")
        dst = PostgresHook(postgres_conn_id="data_dw")

        col_str = ", ".join(columns)
        rows = src.get_records(f"SELECT {col_str} FROM master_ref.{src_table}")

        dst.run(f"TRUNCATE dw_sp2kp.{dst_table}", autocommit=True)
        dst.insert_rows(table=f"dw_sp2kp.{dst_table}", rows=rows, target_fields=columns)

    def etl_dim_provinsi():
        src = PostgresHook(postgres_conn_id="sp2kp_v2")
        dst = PostgresHook(postgres_conn_id="data_dw")

        query = """
            SELECT DISTINCT kode_provinsi, nama_provinsi, nama_singkat_provinsi
            FROM master_ref.wilayah
            WHERE kode_provinsi IS NOT NULL
        """
        rows = src.get_records(query)

        dst.run("TRUNCATE dw_sp2kp.dim_provinsi", autocommit=True)
        dst.insert_rows(
```

Gambar IV. 5. ETL tabel *master* (dim) *extract* dari *database production*

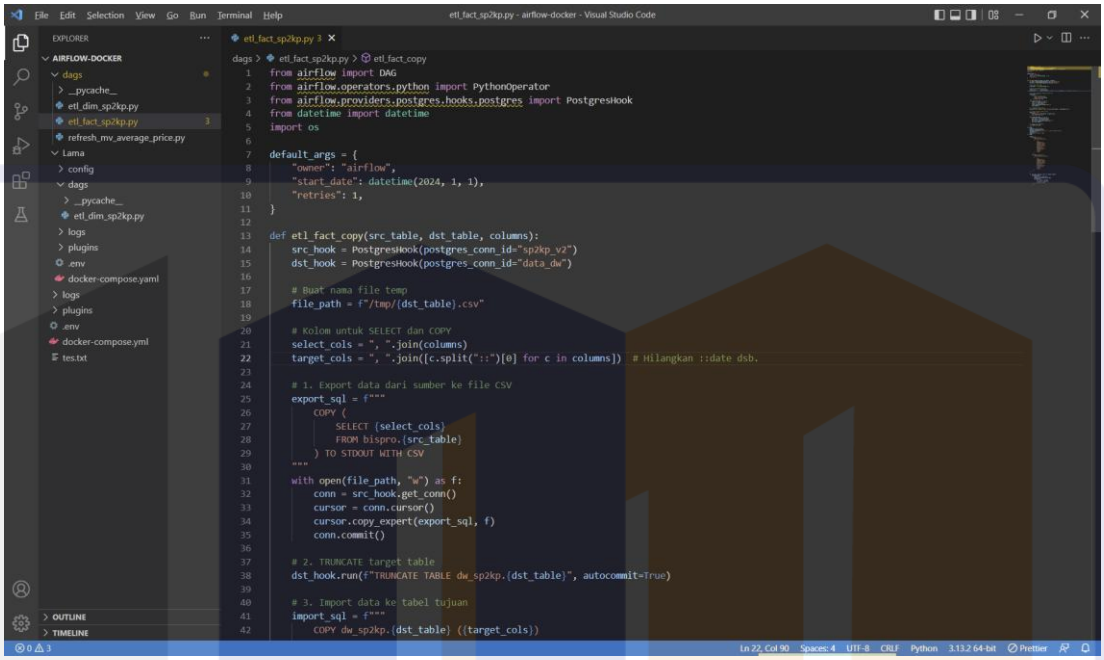


```python
        """
        rows = src.get_records(query)

        dst.run("TRUNCATE dw_sp2kp.dim_kabupaten_kota", autocommit=True)
        dst.insert_rows(
            table="dw_sp2kp.dim_kabupaten_kota",
            rows=rows,
            target_fields=["kode_kab_kota", "nama_kab_kota", "kode_provinsi", "is_ihk2019", "is_ihk2024"]
        )

    tables = {
        "komoditas": ("dim_komoditas", ["id", "kode", "nama", "tipe_komoditas_id", "updated_at"]),
        "variant": ("dim_variant", ["id", "kode", "komoditas_id", "nama", "is_nasional", "qty", "satuan_id", "tipe_komoditas_id", "update
        "tipe_komoditas": ("dim_tipe_komoditas", ["id", "nama", "updated_at"]),
        "satuan": ("dim_satuan", ["id", "display", "deskripsi", "updated_at"]),
        "pasar": ("dim_pasar", ["id", "nama", "deskripsi", "kode_provinsi", "kode_kab_kota", "updated_at"]),
    }

    # Tasks untuk tabel umum
    for src_tbl, (dst_tbl, cols) in tables.items():
        PythonOperator(
            task_id=f"etl_{dst_tbl}",
            python_callable=etl_dim,
            op_kwargs={
                "src_table": src_tbl,
                "dst_table": dst_tbl,
                "columns": cols
            },
        )

    # Tasks untuk provinsi dan kabupaten/kota dari master_ref.wilayah
    PythonOperator(
        task_id="etl_dim_provinsi",
        python_callable=etl_dim_provinsi,
    )

    PythonOperator(
        task_id="etl_dim_kabupaten_kota",
        python_callable=etl_dim_kabupaten_kota,
    )
```

Gambar IV. 6. ETL tabel *master* (dim) *transform* dan *load* ke data *warehouse*

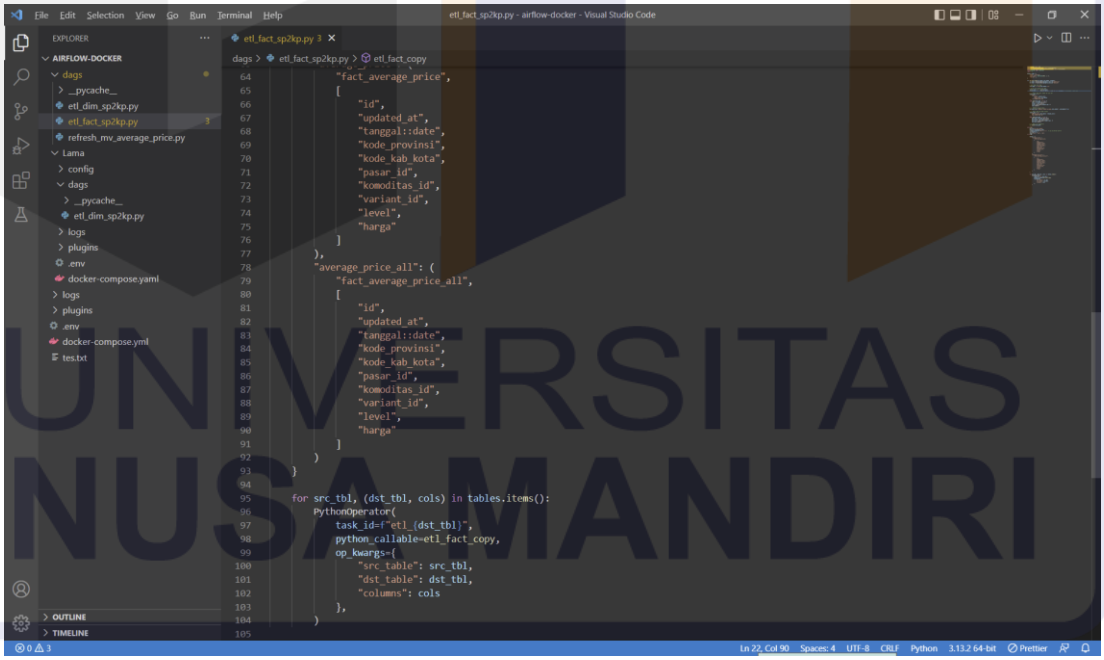
Program ini bertujuan untuk mengotomatisasi alur pemindahan data master (*dimension*) dari sistem operasional ke dalam data *warehouse*.

b. etl_fact_sp2kp.py



```python
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.providers.postgres.hooks.postgres import PostgresHook
from datetime import datetime
import os

default_args = {
    "owner": "airflow",
    "start_date": datetime(2024, 1, 1),
    "retries": 1,
}

def etl_fact_copy(src_table, dst_table, columns):
    src_hook = PostgresHook(postgres_conn_id="sp2kp_v2")
    dst_hook = PostgresHook(postgres_conn_id="data_dw")

    # Buat nama file temp
    file_path = f"/tmp/{dst_table}.csv"

    # Kolom untuk SELECT dan COPY
    select_cols = ", ".join(columns)
    target_cols = ", ".join([c.split("::")[0] for c in columns])  # Hilangkan ::date dsb.

    # 1. Export data dari sumber ke file CSV
    export_sql = f"""
        COPY (
            SELECT {select_cols}
            FROM bispro.{src_table}
        ) TO STDOUT WITH CSV
    """
    with open(file_path, "w") as f:
        conn = src_hook.get_conn()
        cursor = conn.cursor()
        cursor.copy_expert(export_sql, f)
        conn.commit()

    # 2. TRUNCATE target table
    dst_hook.run(f"TRUNCATE TABLE dw_sp2kp.{dst_table}", autocommit=True)

    # 3. Import data ke tabel tujuan
    import_sql = f"""
        COPY dw_sp2kp.{dst_table} ({target_cols})
```

Gambar IV. 7. ETL tabel *fact extract* dari *database production*

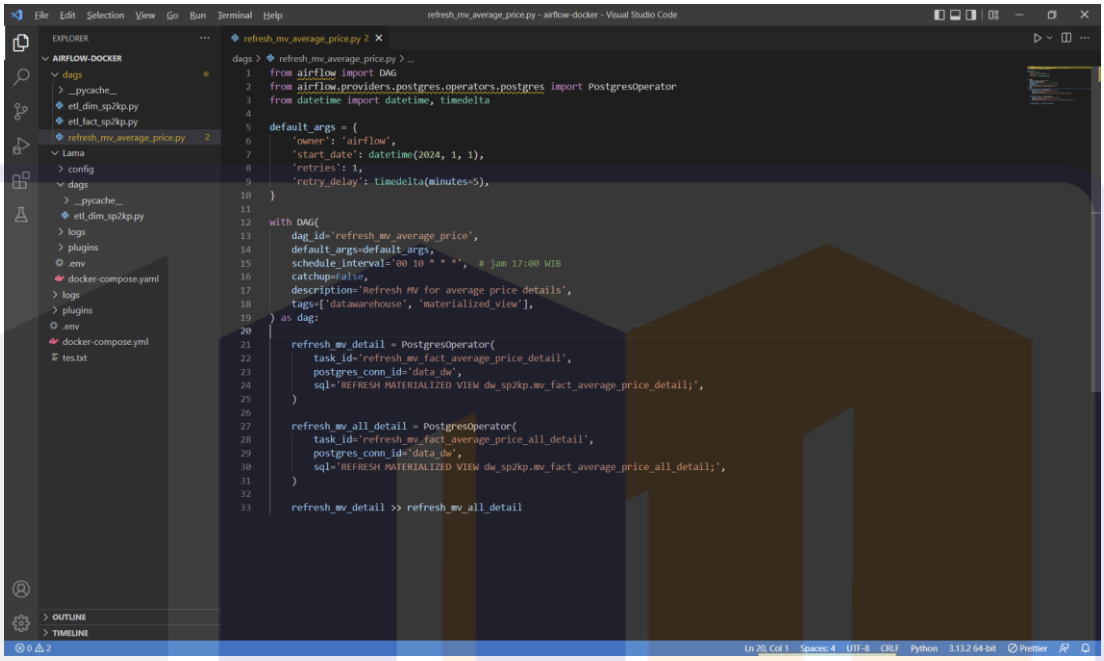


```python
            "fact_average_price",
            [
                "id",
                "updated_at",
                "tanggal::date",
                "kode_provinsi",
                "kode_kab_kota",
                "pasar_id",
                "komoditas_id",
                "variant_id",
                "level",
                "harga"
            ]
        ),
        "average_price_all": (
            "fact_average_price_all",
            [
                "id",
                "updated_at",
                "tanggal::date",
                "kode_provinsi",
                "kode_kab_kota",
                "pasar_id",
                "komoditas_id",
                "variant_id",
                "level",
                "harga"
            ]
        )
    }

    for src_tbl, (dst_tbl, cols) in tables.items():
        PythonOperator(
            task_id=f"etl_{dst_tbl}",
            python_callable=etl_fact_copy,
            op_kwargs={
                "src_table": src_tbl,
                "dst_table": dst_tbl,
                "columns": cols
            },
        )
```

Gambar IV. 8. ETL tabel *fact transform* dan *load* ke data *warehouse*

Program ini bertujuan untuk mengotomatisasi alur pemindahan data transaksi (*fact*) dari sistem operasional ke dalam data *warehouse*.

c. refresh_mv_average_price.py

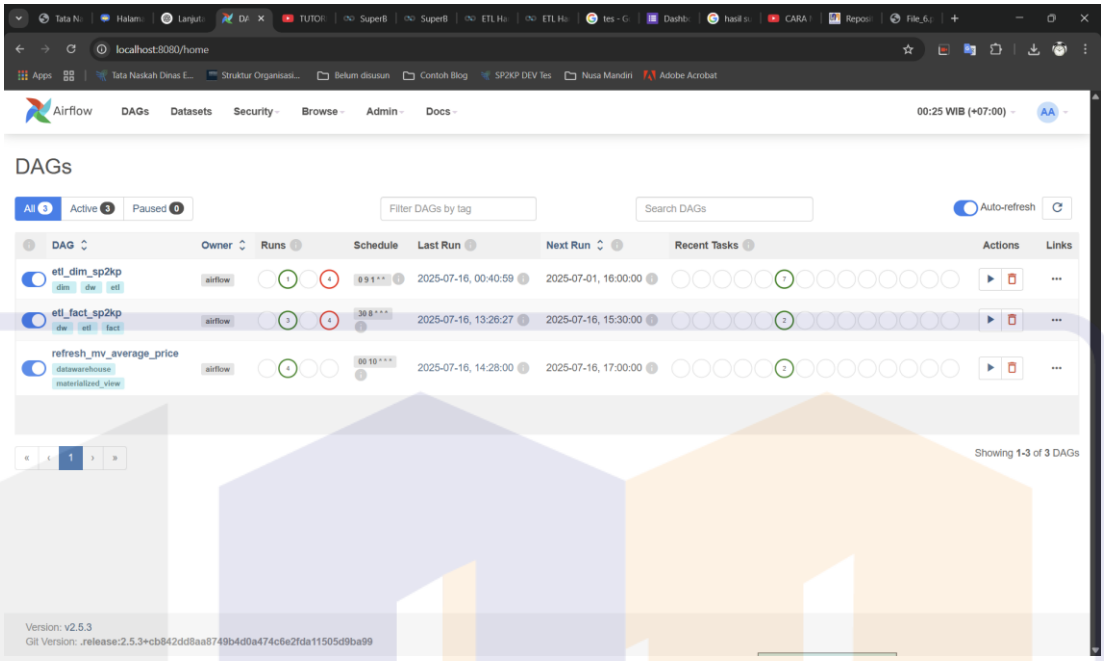


Gambar IV. 9. Program *refresh materialized view*

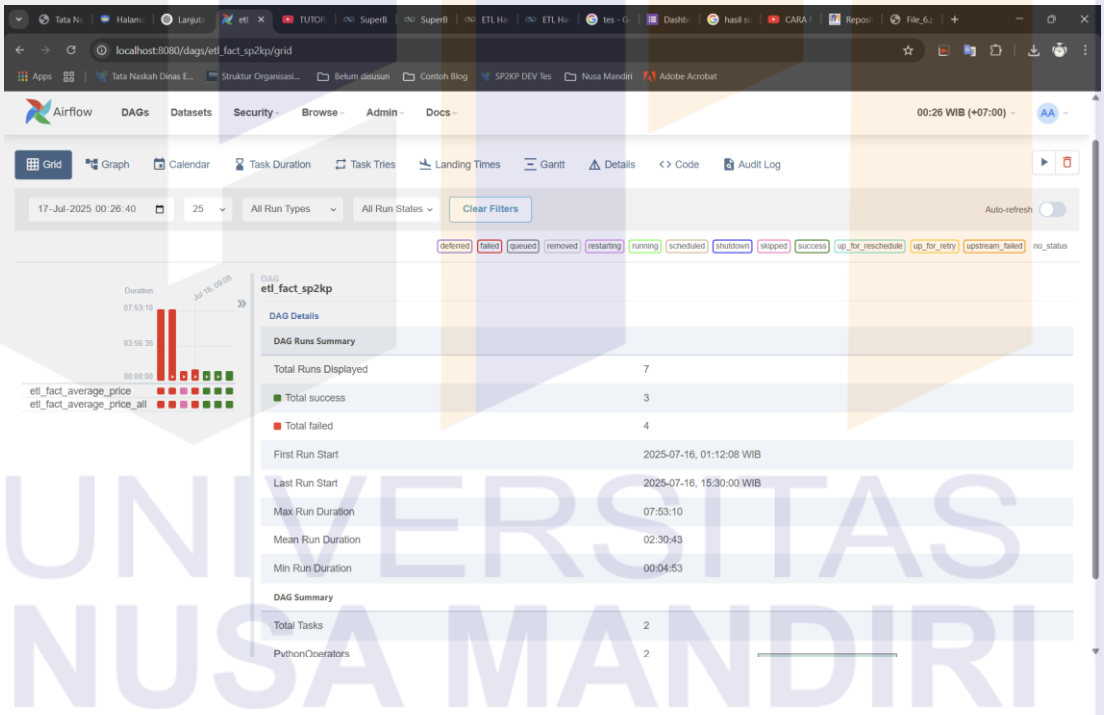
Data dari tabel dim dan *fact* akan diambil, digabungkan, dan disimpan dalam sebuah *materialized view* untuk mengoptimalkan kinerja sistem. Pendekatan ini dipilih guna mengurangi beban komputasi saat menjalankan *query* secara langsung ke tabel-tabel dasar. Dengan menyimpan hasil penggabungan tersebut dalam bentuk *materialized view*, proses pembacaan data menjadi lebih cepat dan efisien

4.4.4. Menjalankan DAG dari *Airflow*

Program yang sudah dibuat menggunakan *python*, dimasukkan ke dalam folder *dags* pada *Airflow*. Selanjutnya program DAG sudah dapat diaktifkan pada *Airflow*, dan bisa di *trigger* untuk berjalan manual di awal, untuk kemudian otomatis melaksanakan proses DAG sesuai jadwal yang ditetapkan.



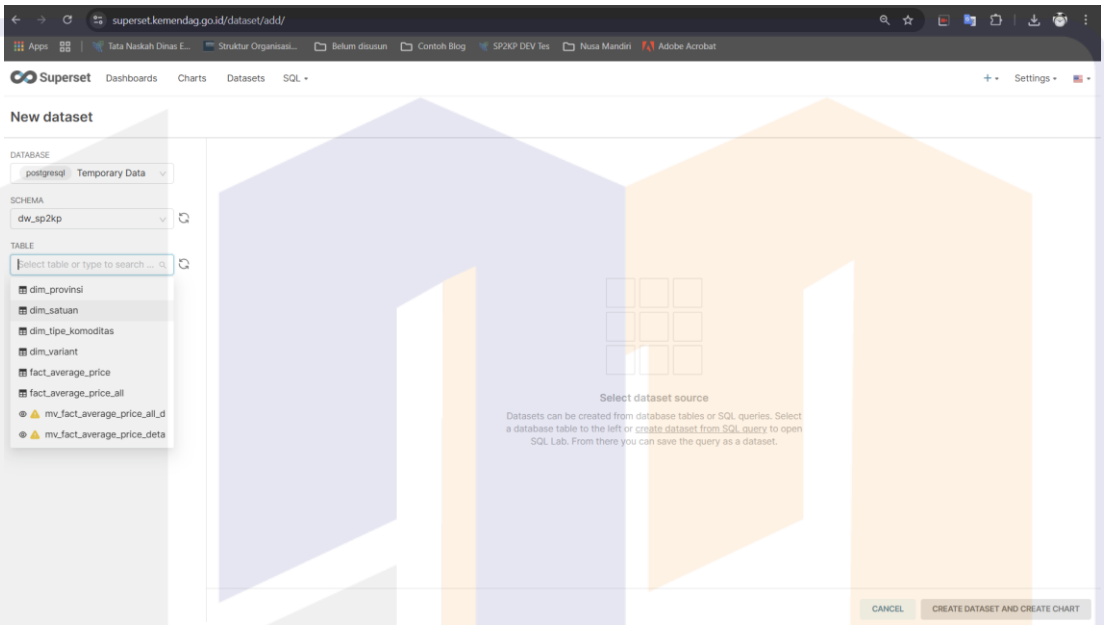
Gambar IV. 10. DAG sudah terbaca pada *Airflow*



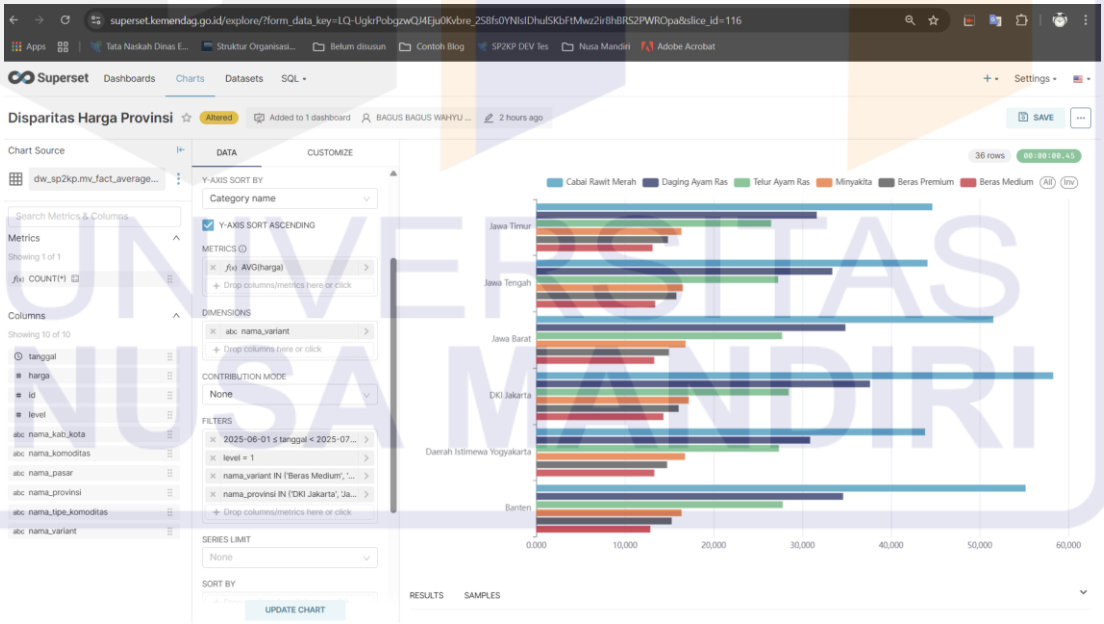
Gambar IV. 11. Proses kerja DAG pada *Airflow*

4.4.5. Integrasi *Apache Superset* ke data *warehouse*

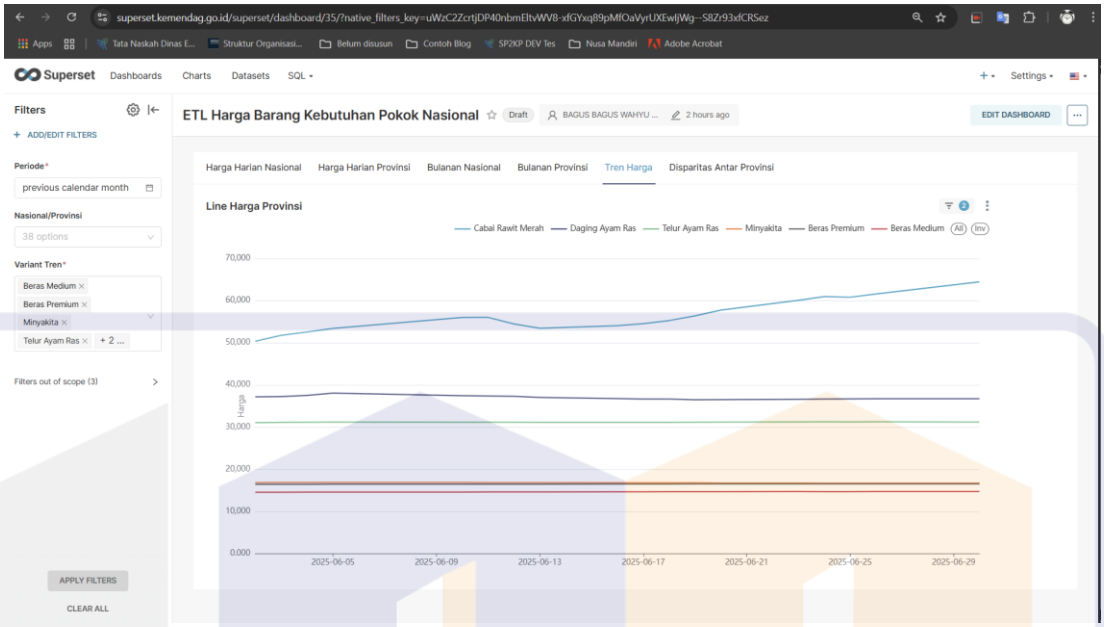
Tabel *fact* dan dim yang sudah dimasukkan dalam data *warehouse*, diintegrasikan dengan *Superset* sehingga kinerja *dashboard* yang dibuat bisa berjalan optimal dan tidak mengganggu sistem operasional utama.



Gambar IV. 12. Pembuatan *dataset Superset* dari data *warehouse*



Gambar IV. 13. Pembuatan *chart* dengan *dataset materialized view* data *warehouse*



Gambar IV. 14. Pembuatan *dashboard* menggunakan data *warehouse*

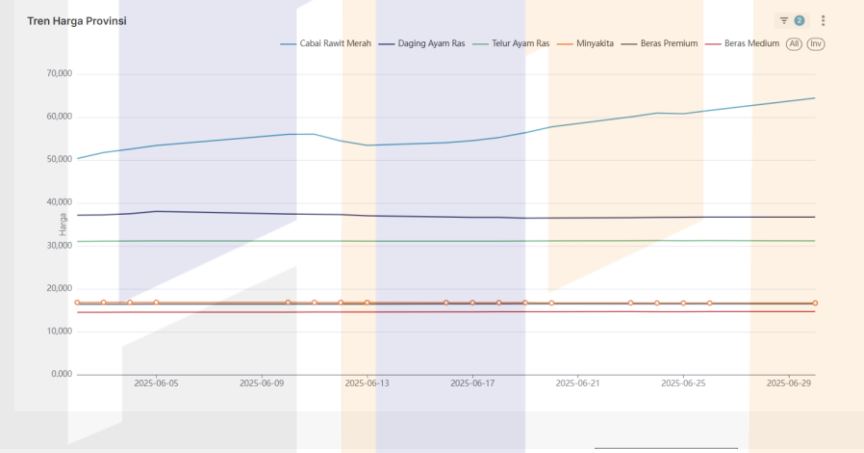
Tahapan-tahapan ini bertujuan untuk mengotomatisasi alur pemindahan data dari sistem operasional ke dalam data *warehouse*. Proses ETL akan memastikan bahwa data yang dimuat ke dalam tabel *fact* dan dimensi selalu mutakhir, terintegrasi, dan siap digunakan untuk keperluan analisis lebih lanjut melalui *dashboard* visualisasi.

Ekstraksi dilakukan dengan mengambil data dari *database production* yang menyimpan harga komoditas harian. Proses transformasi meliputi pembersihan data, normalisasi satuan harga, dan penerapan metode untuk melacak histori perubahan. Selanjutnya, data dimuat (*load*) secara berkala ke dalam data *warehouse PostgreSQL*.

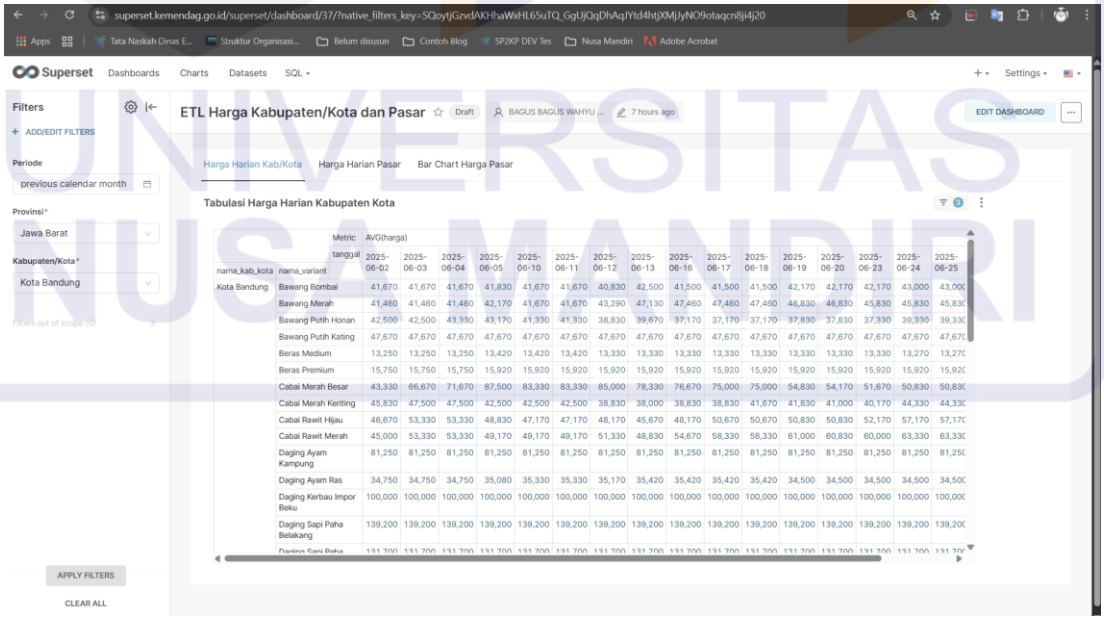
Workflow ETL dirancang dan dijadwalkan secara otomatis menggunakan *Apache Airflow*. *Airflow* digunakan karena mendukung orkestrasi pipeline secara modular dan memiliki kemampuan visualisasi dependensi antar proses. Peneliti membuat program DAG (Directed Acyclic Graph) menggunakan bahasa pemrograman *python*. Skrip program disusun untuk mendefinisikan tahapan ETL dari

ekstraksi hingga pemuatan data ke *warehouse*, yang dijalankan secara otomatis pada waktu tertentu.

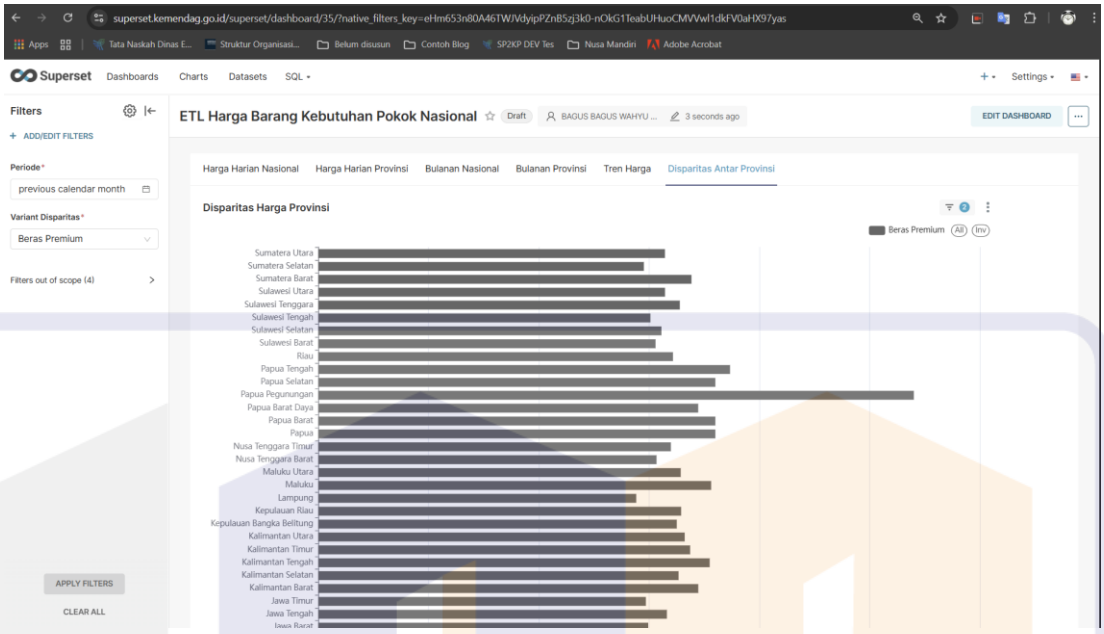
Prototipe ini juga mencakup implementasi visualisasi data menggunakan *Apache Superset*. *Superset* dikonfigurasi untuk membaca data dari *warehouse*, bukan dari *database production*. Hal ini mengurangi beban pada sistem utama dan meningkatkan performa *dashboard*. Beberapa contoh *dashboard* yang dibangun antara lain tabulasi harga harian, visualisasi tren harga, dan perbandingan harga antar wilayah komoditas tertentu.



Gambar IV. 15. Visualisasi tren harga



Gambar IV. 16. Tabulasi harga harian



Gambar IV. 17. Perbandingan harga antar Provinsi

Dengan adanya prototipe ini, proses analisis data menjadi lebih sistematis, efisien, dan mudah digunakan oleh pengguna akhir. Prototipe yang dikembangkan juga menjadi bukti awal bahwa pendekatan ETL dan data *warehouse* mampu meningkatkan kualitas sistem pemantauan harga yang andal dan berkelanjutan.

4.5. *Test*

Peneliti melakukan evaluasi hasil implementasi data *warehouse* pada sistem pemantauan harga barang kebutuhan pokok di Kementerian Perdagangan, melalui penyebaran kuesioner secara daring menggunakan *Google Form*. Pengujian ini bertujuan untuk mengetahui tingkat *usability* serta persepsi pengguna terhadap sistem setelah penerapan proses ETL dengan *Airflow* dan *Python*.

Dalam tahap pengujian ini, peneliti menggunakan metode *System Usability Scale* (SUS). Metode kuesioner ini terdiri dari 10 (sepuluh) pernyataan. Responden diminta untuk memberikan penilaian terhadap pernyataan dengan menggunakan skala 1 sampai 5, dimana angka 1 yaitu sangat tidak setuju dan angka 5 yaitu sangat setuju.

Melalui pengukuran skala ini, diharapkan bisa memperoleh data mengenai tingkat kegunaan dan kenyamanan *dashboard* berdasarkan pengalaman dari pengguna.

Tabel IV. 3. Kuesioner setelah implementasi *system usability scale*

No	Pernyataan
1	Saya merasa nyaman menggunakan *dashboard* ini.
2	Saya merasa *dashboard* ini rumit untuk digunakan.
3	*Dashboard* ini mudah digunakan.
4	Saya membutuhkan bantuan teknis untuk bisa menggunakan *dashboard* ini.
5	Fitur-fitur dalam *dashboard* ini berjalan sesuai harapan.
6	Saya merasa banyak hal yang membingungkan saat menggunakan *dashboard* ini.
7	*Dashboard* ini sangat fungsional dan cepat.
8	Saya merasa *dashboard* ini tidak konsisten dan sulit diprediksi.
9	Saya merasa percaya diri saat menggunakan *dashboard* ini.
10	Saya perlu mempelajari banyak hal sebelum bisa menggunakan *dashboard* ini.

Setelah kuesioner dikumpulkan, data diolah menggunakan rumus *System Usability Scale* (SUS). Skor untuk pernyataan ganjil dikurangi 1, sedangkan pernyataan genap dikurangi dari 5. Total skor kemudian dikalikan 2,5 untuk menghasilkan skor SUS akhir. Proses ini dilakukan per responden untuk merefleksikan penilaian individu terhadap kegunaan *dashboard*.

Tabel IV. 4. Hasil perhitungan SUS setelah implementasi

Respondent ID	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	SUS Score
Orang 1	5	1	5	3	4	2	5	2	5	2	85
Orang 2	5	1	5	1	5	1	5	1	5	1	100
Orang 3	5	1	5	2	4	1	5	1	4	3	87.5
Orang 4	5	3	5	4	5	2	5	1	5	3	80
Orang 5	4	2	4	2	4	2	4	2	4	2	75
Orang 6	5	5	5	5	5	5	5	5	5	5	50
Orang 7	3	3	3	1	5	3	4	2	4	3	67.5
Orang 8	5	1	4	2	4	2	5	2	4	3	80
Orang 9	4	2	4	3	4	2	5	1	5	3	77.5
Orang 10	4	1	4	3	5	2	5	2	4	3	77.5
Rata-Rata											**78**

Berdasarkan hasil kuesioner, Implementasi ETL data *warehouse* pada sistem pemantauan harga berada dalam kategori *acceptable*, yang berarti implementasi ETL data *warehouse* ini diterima dengan baik oleh para pengguna. Skor ini menunjukkan bahwa ETL data *warehouse* memiliki tingkat kegunaan yang tinggi, mampu memberikan pengalaman pengguna yang positif dan memadai, meskipun tetap ada ruang untuk perbaikan di beberapa aspek guna lebih meningkatkan efisiensi dan kenyamanan.

Selain menggunakan SUS, peneliti juga menambahkan delapan pernyataan tambahan untuk menggali persepsi pengguna terhadap aspek kinerja dan manfaat sistem SP2KP hasil implementasi data warehouse. Pernyataan ini mencakup aspek seperti kecepatan akses, stabilitas sistem, kelengkapan data, dan kemudahan analisis yang tidak secara langsung diukur oleh SUS. Responden menilai setiap pernyataan menggunakan skala Likert 1–5 (1 = sangat tidak setuju, 5 = sangat setuju).

Tabel IV.5. Pertanyaan tambahan spesifik terhadap sistem

No	Pernyataan
1	Setelah implementasi data warehouse, kecepatan akses data di dashboard pemantauan harga menjadi lebih baik.
2	Data di dashboard pemantauan harga selalu tersedia dan jarang mengalami kegagalan pemuatan.
3	*Data yang ditampilkan di dashboard pemantauan harga sudah mencakup indikator harga yang dibutuhkan.*
4	Data yang disajikan pada dashboard akurat dan konsisten dengan sumber data aslinya.
5	Jadwal pembaruan data (setiap hari pukul 15.00) sudah sesuai dengan kebutuhan analisis.
6	Tampilan dashboard memudahkan dalam memahami tren harga dan kebutuhan pokok.
7	*Sistem berjalan dengan stabil tanpa sering mengalami error atau gangguan koneksi database.*
8	Sistem hasil implementasi data warehouse membantu mempercepat pekerjaan dalam analisis dan pelaporan.

Tabel IV. 6. Hasil perhitungan kuesioner tambahan

Respondent ID	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Rata-rata
Orang 1	4	4	5	5	5	5	5	5	4.75
Orang 2	5	5	4	5	4	5	5	5	4.75
Orang 3	4	5	4	5	5	4	4	4	4.38
Orang 4	4	4	5	4	4	3	4	4	4.00
Orang 5	5	5	5	5	5	5	5	5	5.00
Orang 6	3	3	4	4	4	3	3	4	3.50
Orang 7	5	4	5	5	4	5	5	5	4.75
Orang 8	5	5	4	4	5	5	4	5	4.63
Orang 9	5	5	5	5	5	5	5	5	5.00
Orang 10	5	4	3	4	4	4	4	4	4.00
Rata-Rata Keseluruhan									**4.48**

Berdasarkan hasil tersebut, nilai rata-rata keseluruhan sebesar 4,48 menunjukkan bahwa pengguna memberikan penilaian yang sangat positif terhadap sistem SP2KP setelah implementasi data warehouse. Responden merasakan peningkatan pada aspek kecepatan akses, stabilitas, dan akurasi data, serta menyatakan bahwa dashboard baru lebih membantu dalam analisis dan pelaporan.

Hasil ini sejalan dengan skor SUS yang tinggi (78), memperkuat kesimpulan bahwa implementasi ETL data warehouse menggunakan *Airflow* dan *Python* berhasil meningkatkan *usability* dan kinerja sistem dashboard pemantauan harga secara signifikan.

4.6. Analisis Hasil

Dalam klasifikasi *Grade* SUS, skor 78 masuk dalam kategori *Grade* B (rentang 70–84), yang mengindikasikan bahwa ETL data *warehouse* ini memiliki kualitas baik dalam hal kemudahan penggunaan dan efektivitas. Mayoritas pengguna merasa bahwa data *warehouse* ini dapat digunakan dengan lancar serta telah memenuhi sebagian besar kebutuhan mereka dalam memantau data harga. Adapun kata sifat yang tepat untuk menggambarkan sistem dengan skor ini adalah "*good*" (baik). Hal ini

mencerminkan bahwa meskipun implementasi ini belum sepenuhnya sempurna, namun telah berhasil memberikan pengalaman yang positif dan layak digunakan dalam konteks operasional sehari-hari.

Pertanyaan tambahan menunjukkan adanya peningkatan nyata dalam kinerja sistem, dengan rata-rata skor sebesar 4,48 dari 5. Hasil ini menunjukkan bahwa sebagian besar responden merasakan perbedaan signifikan setelah implementasi data warehouse, di mana sistem menjadi lebih cepat menampilkan data, pembaruan informasi lebih stabil, dan tampilan dashboard lebih mudah dipahami. Temuan ini memperkuat hasil skor SUS bahwa sistem hasil implementasi tidak hanya mudah digunakan, tetapi juga memberikan manfaat nyata dalam efisiensi, kecepatan, dan keandalan data yang ditampilkan.

Waktu respon *dashboard* setelah menggunakan data *warehouse* terbukti jauh lebih cepat dan efisien. Rata-rata, *dashboard* dapat ditampilkan dalam waktu kurang dari 2 detik, berbeda jauh dibandingkan saat masih menggunakan *database production*, di mana waktu tampil *dashboard* bisa mencapai 10 hingga 20 detik.

Selain meningkatkan kecepatan akses, *workflow* ETL juga berperan penting dalam menjaga performa sistem tetap ringan. Dengan menghindari *query* langsung ke *database* operasional, sistem tidak lagi terbebani oleh permintaan data yang intens sepanjang waktu. Proses pembaruan data dilakukan secara otomatis, tepat waktu, dan akurat, serta dijadwalkan pada periode *low traffic*, yaitu saat aktivitas input data harga sedang minim.

Dengan demikian, kontributor yang melakukan input harga tidak lagi mengalami gangguan seperti sistem lambat atau lemot akibat proses pembacaan data oleh *dashboard Superset*. Hal ini menciptakan lingkungan kerja yang lebih stabil dan efisien, baik untuk penginput data maupun pengguna *dashboard* analitik.