BAB III

PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Sisi magang sebagai intern berada pada divisi Business Intelligence and Analytics Platform Operation Management. Perusahaan menempatkan intern berdasarkan latar belakang keilmuan dan keterampilan teknis yang dimiliki masing-masing individu. Dalam hal ini, karena memiliki ketertarikan dan dasar keahlian di bidang data dan analitik, intern menjalankan peran yang berfokus pada pengolahan data dan penyusunan visualisasi informasi.

Selama masa magang, posisi intern diarahkan untuk mempelajari dan mengimplementasikan alat bantu kerja yang lazim digunakan di industri, khususnya Apache Airflow untuk mengotomasi proses data pipeline, serta Power BI untuk pengembangan dashboard visualisasi berbasis data dummy. Proyek dummy ini digunakan sebagai simulasi dari alur kerja nyata dalam tim data, dengan tujuan untuk membangun pemahaman mendalam mengenai proses kerja teknis yang berjalan di divisi.

Fokus utama kegiatan magang terletak pada pemahaman alur pengelolaan data di tim BI and Analytics Platform Operation Management, mulai dari tahap ekstraksi data, pembersihan, transformasi, validasi, penyimpanan ke dalam database, hingga penyajian data dalam bentuk visual. Intern menjalankan eksperimen melalui tahapan-tahapan tersebut, sekaligus mengembangkan dokumentasi teknis yang menggambarkan proses dan logika dari pipeline yang dibangun.

Proses pembelajaran dan pengembangan dilakukan secara bertahap dan terstruktur, dengan bimbingan langsung dari mentor divisi. Mentor tidak hanya memberikan pengarahan teknis mengenai penggunaan *software* yang relevan, tetapi juga menjelaskan konteks penerapan teknologi tersebut terhadap kebutuhan bisnis perusahaan. Intern juga berkesempatan untuk mengikuti sesi evaluasi dan review hasil kerja bersama mentor, di mana proses ini digunakan untuk

mengidentifikasi aspek yang perlu disempurnakan, serta mendiskusikan pengembangan lanjutan dari solusi yang telah dibangun.

Alur kerja intern bersifat fleksibel namun tetap terintegrasi dengan ritme proyek yang sedang berjalan di divisi. Koordinasi dilakukan melalui komunikasi langsung dengan mentor. Intern dapat menyesuaikan diri dengan kebutuhan proyek yang bersifat dinamis, misalnya ketika terdapat permintaan untuk penyesuaian pada *pipeline* atau *dashboard*. Proses ini juga membantu mengembangkan kemampuan adaptasi dan kolaborasi dalam lingkungan kerja yang nyata.

3.2 Tugas dan Uraian Kerja Magang

IT intern pada perusahaan Telkomsel memiliki berbagai macam tugas yang disesuaikan dengan proyek yang sedang berjalan serta kemampuan masing-masing *Intern*. Selama pelaksanaan magang, intern ditempatkan pada tim yang menangani proses pengelolaan dan visualisasi data, dengan fokus utama pada penggunaan Apache Airflow sebagai tools orkestrasi pipeline ETL, serta Power BI sebagai platform untuk pembuatan dashboard visualisasi data.

Dalam mendukung proses pembelajaran dan pengembangan keterampilan teknis, intern diberikan tanggung jawab untuk mengerjakan sebuah proyek dummy yang dirancang menyerupai alur kerja data yang digunakan di lingkungan kerja sebenarnya. Proyek ini mencakup pembuatan pipeline ETL menggunakan Apache Airflow untuk membaca, membersihkan, mentransformasi, dan memuat data ke dalam database PostgreSQL, kemudian menyajikan hasilnya dalam bentuk dashboard interaktif melalui Power BI.

Pipeline ETL dibangun untuk menjalankan proses pengolahan data secara otomatis dan terjadwal, sementara dashboard yang dibuat berfungsi untuk menampilkan informasi kinerja bisnis, tren historis, serta insight analitis berdasarkan dataset yang digunakan. Proyek ini bertujuan untuk memberikan pemahaman menyeluruh mengenai workflow yang ada di tim data Telkomsel,

sekaligus melatih intern dalam penerapan teknologi data pipeline dan visualisasi secara praktikal.

Tabel 3.1 Alur Kerja Magang

No	Pekerjaan yang dilakukan	Waktu Pengerjaan	Hasil
1	Pengenalan kerja, dan tools	Februari 03 - 10	Memahami alur kerja dan tools seperti *airflow* dan nifi
2	Instalasi software (*airflow* & nifi)	Februari 10 - 21	Memahami penggunaan software
3	Mempelajari konsep dasar Apache *Airflow* dan Apache NiFi (DAG, task, operator, scheduler)	Februari 21 – Maret 14	Memahami alur kerja dan arsitektur *pipeline* menggunakan *Airflow* dan tidak memilih nifi untuk membikin proses ETL.
4	Visualisasi *pipeline* ke Power BI	Maret 31 – April 13	Dashboard awal berhasil dibuat dengan data hasil ETL *pipeline*
5	Presentasi awal kepada mentor untuk menjelaskan alur *pipeline* dan tampilan visualisasi	April 16	Feedback diterima terkait efisiensi query dan desain *pipeline*
6	Melakukan perbaikan *pipeline* dan visualisasi berdasarkan masukan dari mentor	April 16 – May 2	*Pipeline* lebih optimal, dokumentasi dan visualisasi diperbarui
7	Finalisasi dokumentasi dan evaluasi keseluruhan hasil kerja magang	May 02 - 28	Dokumentasi lengkap (teknis dan fungsional) disusun sebagai laporan akhir

Pada tabel 3.1, pelaksanaan program magang di PT Telekomunikasi Seluler (Telkomsel), pada tim yang menangani pengelolaan pipeline data dan pembuatan visualisasi bisnis. *Intern* bertugas mengerjakan sebuah proyek dummy guna memahami alur kerja tim data, serta mendalami penerapan teknologi data pipeline dan visualisasi yang digunakan di lingkungan perusahaan.

Kegiatan magang difokuskan pada penggunaan Apache Airflow untuk membangun proses ETL *(Extract, Transform, Load)* dan *Power BI* untuk menyajikan hasil data dalam bentuk dashboard interaktif. Selain itu, juga dikenalkan tools pendukung lain seperti Apache NiFi, meskipun tidak digunakan secara langsung dalam implementasi pipeline.

Seluruh proses kerja dilakukan secara bertahap, dimulai dari pengenalan sistem dan tools, instalasi software, pemahaman konsep dasar, hingga pengembangan pipeline dan visualisasi. Setiap tahapan didampingi oleh supervisi dari mentor untuk memastikan pemahaman dan pelaksanaan sesuai dengan standar yang berlaku di lingkungan kerja profesional.

Alur pengerjaan proyek yang dikerjakan selama magang mengikuti tahapan *data pipeline* secara lengkap, dimulai dari pengambilan data mentah hingga penyajian dalam bentuk *visualisasi*. Proyek ini mensimulasikan proses kerja tim *Business Intelligence* dalam mengelola dan memproses data secara otomatis, sistematis, dan terstruktur.

Langkah awal dimulai dengan mengambil data mentah dari sumber yang telah disiapkan dalam format *CSV*. Data tersebut kemudian diproses melalui tahap *extract*, di mana sistem membaca isi data dan menyiapkannya untuk tahapan berikutnya. Setelah data berhasil di-*extract*, proses dilanjutkan ke tahap *data cleaning*. Pada tahap ini, *pipeline* akan menghapus *missing values*, *duplikat*, serta mendeteksi dan menangani data yang tidak valid atau tidak sesuai format.

Setelah proses pembersihan selesai, *pipeline* memasuki tahap *transformation*, yaitu tahapan yang mengolah data agar sesuai dengan kebutuhan *analitik*. Transformasi ini mencakup pengubahan format data, perhitungan nilai

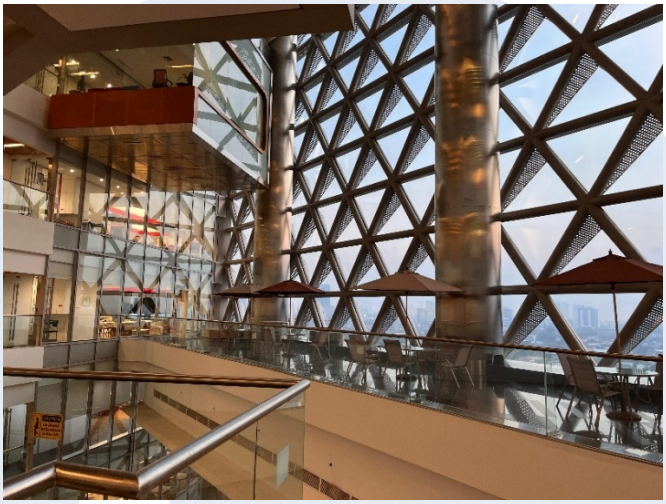
baru, serta proses *aggregation* seperti menghitung total penjualan, rata-rata pendapatan, atau rasio antar variabel. Tahapan ini bertujuan untuk menyusun data dalam bentuk yang lebih ringkas, terstruktur, dan siap digunakan dalam analisis lanjutan.

Selanjutnya, *pipeline* menjalankan proses *load*, yaitu memuat data yang telah dibersihkan dan ditransformasi ke dalam sistem penyimpanan ke dalam *database PostgreSQL*. Proses ini memastikan bahwa data dapat diakses secara efisien oleh sistem pelaporan dan *visualisasi*.

Tahapan akhir berupa *visualization* dilakukan dengan menggunakan *Power BI*. *Dashboard interaktif* disusun untuk menampilkan *insight* penting, seperti tren penjualan, distribusi kategori produk, serta perbandingan performa antar segmen. Tujuan dari *visualization* ini adalah untuk memudahkan pengguna dalam memahami informasi secara cepat dan mengambil keputusan berdasarkan data yang tersedia.

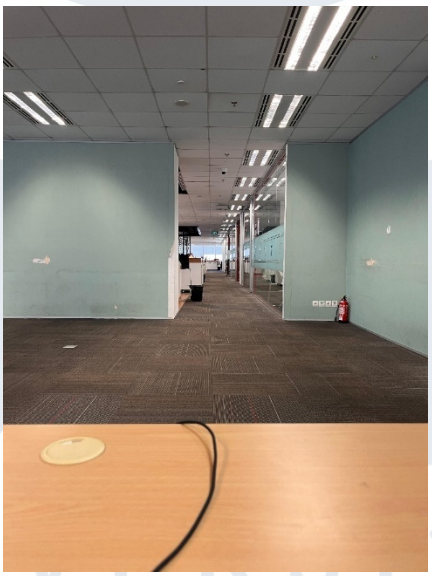
Melalui proyek ini, seluruh proses pengolahan data berjalan secara otomatis menggunakan *Apache Airflow* sebagai alat orkestrasi *pipeline* yang mengatur jadwal, alur kerja, dan ketergantungan antar *task*. Proyek ini memberikan pemahaman menyeluruh mengenai bagaimana *dataset* diproses secara *end-to-end*, mulai dari kondisi mentah hingga siap digunakan untuk mendukung kebutuhan *business analysis* di lingkungan kerja profesional.

3.2.1 Pengenalan Kerja



Gambar 3.1 Suasana Kantor

Pada gambar 3.1, menunjukkan suasana area kerja di kantor Telkomsel, tepatnya di Gedung Telkomsel Smart Office (TSO), Jakarta di lantai 15.



Gambar 3. 2 Tempat Kerja Kantor

Pada gambar 3.2 adalah lingkungan kerja dimana *intern* langsung mengikuti sesi pengenalan lingkungan kerja dan memahami struktur tim

Business Intelligence and Analytics Platform Operation Management. Dalam sesi ini, tim menjelaskan secara menyeluruh bagaimana alur data dijalankan dalam skala perusahaan, mulai dari tahap pengumpulan data mentah, proses pembersihan dan transformasi, hingga penyajian dalam bentuk visualisasi yang dapat digunakan untuk mendukung pengambilan keputusan bisnis. Penjelasan ini sekaligus memberikan gambaran tentang pentingnya peran *data pipeline* yang terstruktur dalam mendukung efisiensi operasional perusahaan.

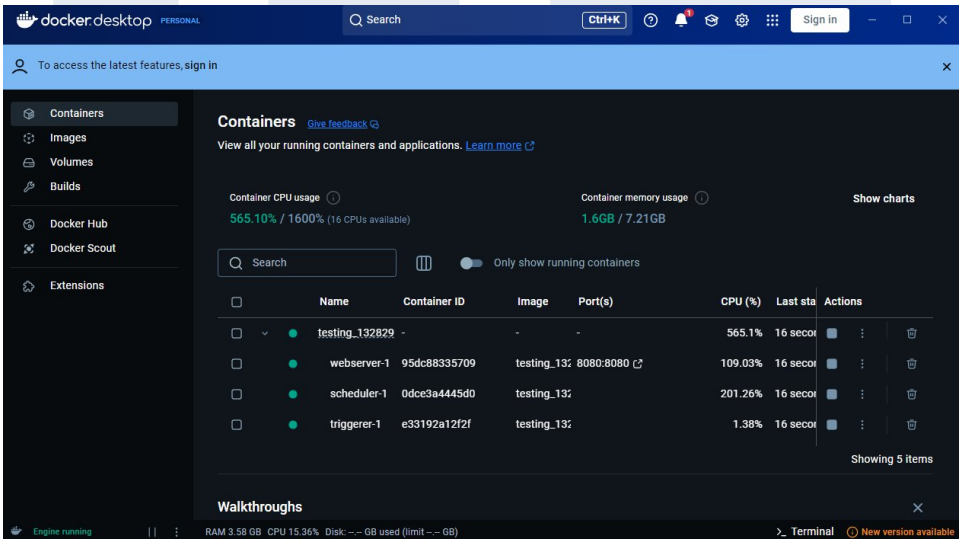
Tim juga memperkenalkan dua platform utama yang biasa digunakan dalam proses orkestrasi data, yaitu *Apache NiFi* dan *Apache Airflow*. Keduanya memiliki fungsi untuk mengatur alur proses *ETL (Extract, Transform, Load)*, namun dengan pendekatan dan arsitektur yang berbeda. Setelah melalui proses eksplorasi dan perbandingan fitur, tim memilih *Apache Airflow* sebagai *tools* utama dalam *proyek dummy* karena kemampuannya dalam menangani *workflow* yang kompleks, terjadwal, dan mudah dipantau melalui antarmuka visual.

Intern kemudian mulai menyusun *pipeline ETL* menggunakan *Airflow*, dengan merancang *DAG (Directed Acyclic Graph)* yang terdiri atas beberapa *task* terpisah, seperti *ingest* data dari sumber file, proses pembersihan data, transformasi logika bisnis, validasi integritas data, serta pemuatan ke dalam *database PostgreSQL*. Setiap *task* disusun secara modular agar memudahkan pengelolaan, *debugging*, dan pengembangan lebih lanjut. Hasil akhir dari *pipeline* ini digunakan sebagai sumber data untuk *dashboard* visual yang dikembangkan melalui *Power BI*.

Visualisasi yang dibangun bertujuan untuk menampilkan *insight* secara jelas dan interaktif, seperti tren penjualan, distribusi kategori produk, dan performa berdasarkan wilayah. Proses ini memperlihatkan bagaimana data mentah yang awalnya tidak terstruktur dapat diubah menjadi informasi bernilai guna, yang dapat digunakan oleh tim bisnis dan manajemen untuk mengambil keputusan secara cepat dan tepat.

Melalui tahapan awal ini, intern memahami bahwa pengolahan data berskala besar tidak hanya membutuhkan pemahaman teknis terhadap *tools*, tetapi juga keterampilan dalam merancang alur kerja yang efisien dan mudah dioperasikan oleh tim lintas fungsi. Pengalaman ini menjadi fondasi penting dalam menjalankan tugas-tugas selanjutnya selama program magang berlangsung.

3.2.2 Instalasi *Software*



Gambar 3.3 Instalasi Docker Dekstop

Pada gambar 3.3 adalah tahap instalasi dan konfigurasi awal dalam proyek magang, penggunaan *Docker* menjadi bagian penting dalam membangun lingkungan kerja yang stabil dan terstandarisasi. Untuk menjalankan *Apache Airflow* secara efisien tanpa harus melakukan instalasi dependensi secara manual di sistem lokal, proses instalasi dilakukan menggunakan *Docker* dan *Docker Compose*.

Docker adalah platform *open-source* yang memungkinkan pengguna untuk membuat, mengemas, dan menjalankan aplikasi di dalam *container*. *Container* tersebut bekerja dalam lingkungan terisolasi yang sudah mencakup seluruh dependensi dan konfigurasi yang dibutuhkan

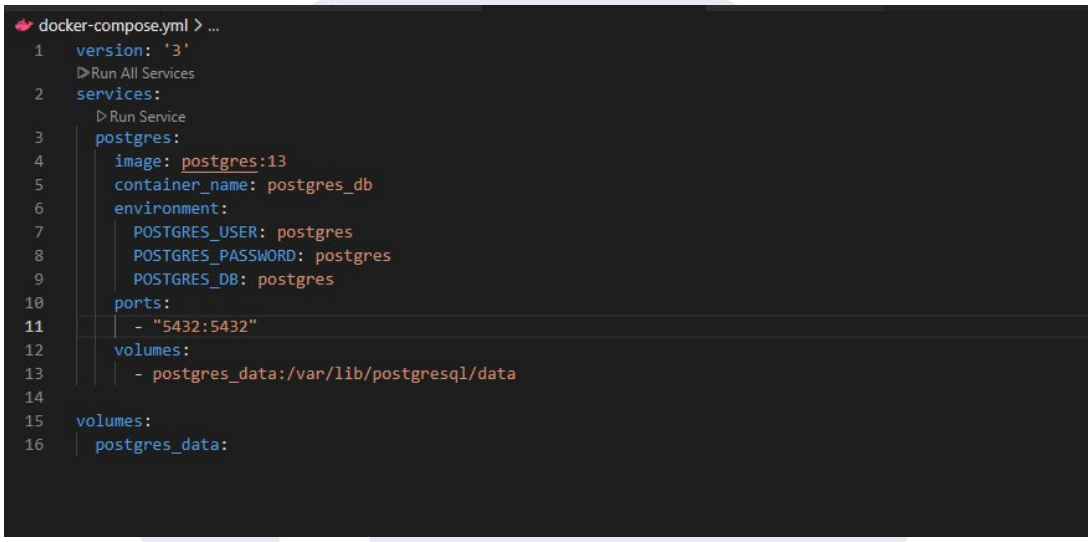
aplikasi. Dengan pendekatan ini, sistem menjadi lebih modular, portabel, dan konsisten lintas perangkat dan sistem operasi.

Dalam konteks instalasi Airflow, setiap komponen utama seperti *scheduler*, *webserver*, *triggerer*, dan *worker* dijalankan dalam *container* terpisah yang saling terhubung melalui jaringan virtual yang diatur oleh *Docker Compose*. File konfigurasi *docker-compose.yaml* digunakan untuk mengatur jalannya seluruh layanan Airflow secara bersamaan. Proses ini memungkinkan seluruh sistem Airflow berfungsi sepenuhnya tanpa perlu menginstal satu per satu di dalam *host machine*, sehingga menghemat waktu setup dan menghindari konflik lingkungan.

Selama proses instalasi, dilakukan beberapa penyesuaian tambahan, seperti menentukan lokasi volume permanen untuk menyimpan metadata DAG dan log eksekusi, menyesuaikan timezone sistem, serta mengatur retry policy untuk task Airflow agar lebih fleksibel dalam proses pengujian. Intern juga melakukan eksplorasi terhadap antarmuka Airflow, seperti tampilan DAGs list, graph view, task tree, serta pemantauan logs untuk setiap task yang berjalan. Pengalaman ini menjadi kunci dalam memahami bagaimana sistem data orchestration bekerja secara modular dan terisolasi, namun tetap saling terhubung secara fungsional.

Penggunaan Docker dalam proyek magang memberikan pemahaman mendalam tentang bagaimana sistem layanan berskala besar dikembangkan dan dijalankan dalam lingkungan profesional. Tahapan ini tidak hanya memperkenalkan konsep containerization, tetapi juga memperkuat keterampilan dalam mengelola sistem multi-layanan yang saling terintegrasi. Pengalaman ini menjadi pondasi teknis yang penting sebelum mulai membangun data pipeline dengan Apache Airflow, dan mencerminkan praktik standar industri dalam pengelolaan infrastruktur data modern.

3.2.3 Docker Connection ke Visual Studio Code



Gambar 3.4 *Set Up Docker* ke *Airflow*

Pada gambar 3.4 adalah *docker-compose.yml* tersebut digunakan untuk menjalankan layanan *database* PostgreSQL di dalam *container* dengan bantuan *Docker Compose*. Dalam konfigurasi ini, layanan bernama postgres menjalankan *image* resmi dari **PostgreSQL versi 13**, dan menggunakan nama *container* postgres_db sebagai identitas unik dalam jaringan *container* yang dibangun. Penulisan konfigurasi ini bertujuan untuk memastikan bahwa layanan *database* dapat berjalan secara otomatis, stabil, dan mudah diakses oleh layanan lain dalam ekosistem pengembangan.

Pada bagian *environment*, konfigurasi menetapkan bahwa sistem akan secara otomatis membuat *database* bernama postgres, dengan pengguna (*user*) dan kata sandi (*password*) yang juga menggunakan nilai postgres. Pengaturan ini mempermudah proses autentikasi saat menghubungkan *pipeline* atau aplikasi lain ke *database*, terutama saat berada dalam tahap pengembangan atau pengujian.

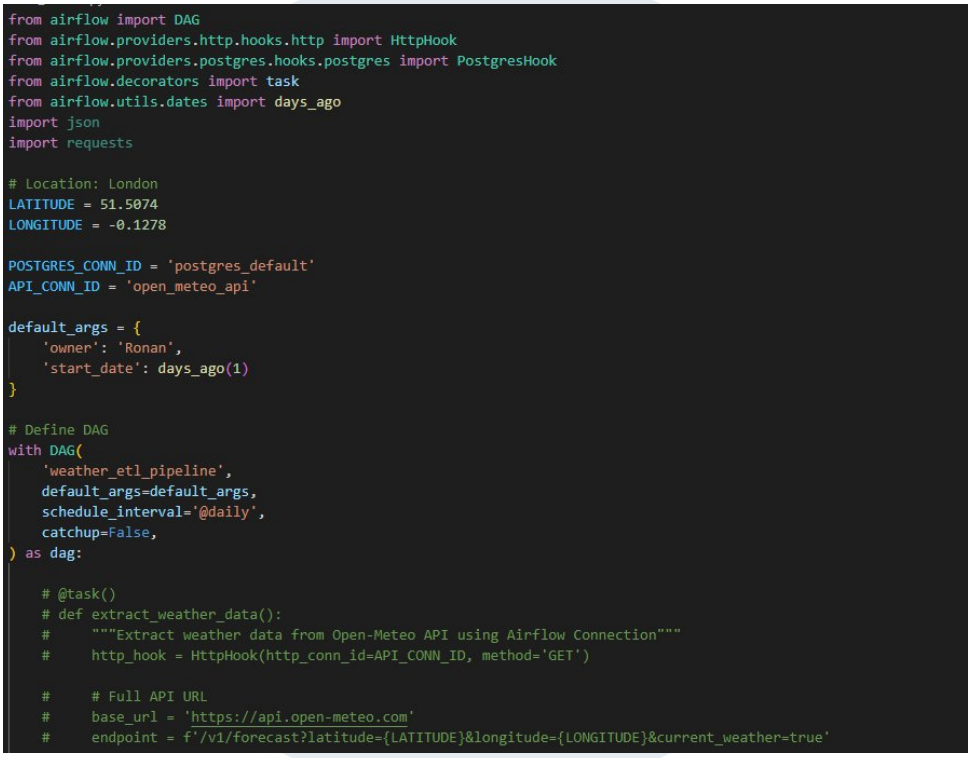
Port 5432 yang merupakan port standar untuk *PostgreSQL*—diekspose dari dalam *container* ke *port* 5432 pada *host machine*. Dengan konfigurasi ini, sistem atau aplikasi yang berjalan di luar *container* tetap dapat mengakses

database yang berjalan di dalam *container* seolah-olah berada di dalam jaringan lokal. Hal ini sangat berguna saat melakukan pengujian konektivitas dari *tool* analisis data seperti DBeaver, pgAdmin, maupun dalam pembuatan *data pipeline*.

Agar data yang disimpan tidak hilang ketika *container* dimatikan, konfigurasi *volume* postgres_data disertakan. Volume ini mengikat direktori penyimpanan data di dalam *container* ke direktori tertentu di luar *container* (pada host), sehingga seluruh data tetap tersimpan secara persisten meskipun terjadi restart atau penghapusan *container*. Pendekatan ini memberikan keunggulan dalam hal *data durability*, yang sangat krusial dalam implementasi sistem yang menangani *data pipeline* dan proses analitik.

Struktur konfigurasi seperti ini sangat umum digunakan dalam arsitektur aplikasi modern yang membutuhkan *backend database*, dan juga menjadi bagian penting dalam implementasi *workflow orchestration* menggunakan *Apache Airflow*. Dalam konteks magang, layanan PostgreSQL ini menjadi elemen kunci dalam menyimpan hasil transformasi data yang dilakukan dalam *pipeline ETL*, sebelum akhirnya divisualisasikan melalui *dashboard* menggunakan *Power BI*. Dengan kombinasi *Docker Compose* dan PostgreSQL, seluruh sistem pengolahan data dapat dibangun dengan cepat, efisien, dan siap dijalankan di berbagai lingkungan kerja tanpa harus melakukan konfigurasi ulang secara manual.

3.2.4 Mempelajari Konsep *Airflow*



```python
from airflow import DAG
from airflow.providers.http.hooks.http import HttpHook
from airflow.providers.postgres.hooks.postgres import PostgresHook
from airflow.decorators import task
from airflow.utils.dates import days_ago
import json
import requests

# Location: London
LATITUDE = 51.5074
LONGITUDE = -0.1278

POSTGRES_CONN_ID = 'postgres_default'
API_CONN_ID = 'open_meteo_api'

default_args = {
    'owner': 'Ronan',
    'start_date': days_ago(1)
}

# Define DAG
with DAG(
    'weather_etl_pipeline',
    default_args=default_args,
    schedule_interval='@daily',
    catchup=False,
) as dag:

    # @task()
    # def extract_weather_data():
    #     """Extract weather data from Open-Meteo API using Airflow Connection"""
    #     http_hook = HttpHook(http_conn_id=API_CONN_ID, method='GET')

    #     # Full API URL
    #     base_url = 'https://api.open-meteo.com'
    #     endpoint = f'/v1/forecast?latitude={LATITUDE}&longitude={LONGITUDE}&current_weather=true'
```

Gambar 3. 5 Konsep *Airflow* 1

Pada gambar 3.5, *Airflow* untuk mengambil data cuaca dari API Open-Meteo dan menyimpannya ke dalam database PostgreSQL. *Pipeline* didefinisikan dalam sebuah DAG bernama weather_etl_*pipeline* yang dijalankan setiap hari (@daily) dan tidak menjalankan tugas yang terlewat (catchup=False).

```python
@task()
def extract_weather_data():
    """Extract weather data from Open-Meteo API using Airflow Connection"""

    http_hook = HttpHook(http_conn_id=API_CONN_ID, method='GET')

    # Only use the endpoint (not base_url)
    endpoint = f'/v1/forecast?latitude={LATITUDE}&longitude={LONGITUDE}&current_weather=true'

    # Make request
    response = http_hook.run(endpoint)

    if response.status_code == 200:
        return response.json()
    else:
        raise Exception(f"Failed to fetch weather data: {response.status_code}")

@task()
def transform_weather_data(weather_data):
    """Transform the extracted weather data"""
    current_weather = weather_data['current_weather']
    transformed_data = {
        'latitude': LATITUDE,
        'longitude': LONGITUDE,
        'temperature': current_weather['temperature'],
        'windspeed': current_weather['windspeed'],
        'winddirection': current_weather['winddirection'],
        'weathercode': current_weather['weathercode']
    }
    return transformed_data
```

Gambar 3.6 Konsep *Airflow* 2

Pada gambar 3.6 Tahap extract dilakukan oleh fungsi extract_weather_data, yang menggunakan HttpHook dari *Airflow* untuk mengirim permintaan HTTP GET ke endpoint API Open-Meteo dengan parameter lintang dan bujur kota London. Jika respons berhasil, data JSON dikembalikan. Selanjutnya, fungsi transform_weather_data mengambil informasi cuaca saat ini dari hasil extract, lalu menyusunnya dalam format yang lebih ringkas dan terstruktur.

```python
@task()
def load_weather_data(transformed_data):
    """Load transformed data into PostgreSQL."""
    pg_hook = PostgresHook(postgres_conn_id=POSTGRES_CONN_ID)
    conn = pg_hook.get_conn()
    cursor = conn.cursor()

    # Create table if it doesn't exist
    cursor.execute("""
    CREATE TABLE IF NOT EXISTS weather_data (
        latitude FLOAT,
        longitude FLOAT,
        temperature FLOAT,
        windspeed FLOAT,
        winddirection FLOAT,
        weathercode INT,
        timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    );
    """)

    # Insert transformed data into the table
    cursor.execute("""
    INSERT INTO weather_data (latitude, longitude, temperature, windspeed, winddirection, weathercode)
    VALUES (%s, %s, %s, %s, %s, %s)
    """, (
        transformed_data['latitude'],
        transformed_data['longitude'],
        transformed_data['temperature'],
        transformed_data['windspeed'],
        transformed_data['winddirection'],
        transformed_data['weathercode']
    ))

    conn.commit()
    cursor.close()
```

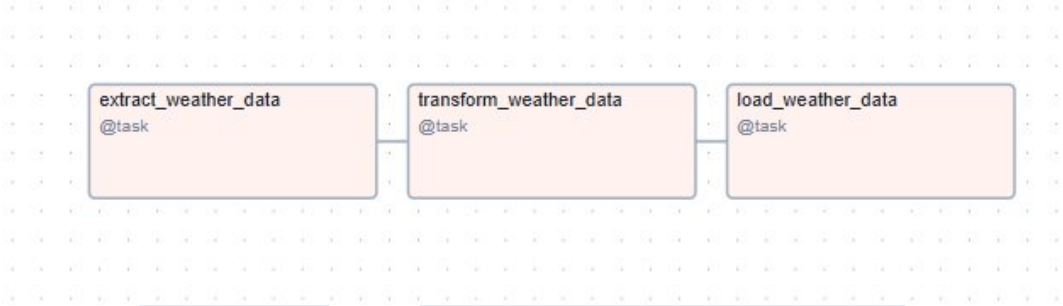
Gambar 3.7 Konsep *Airflow* 3

Pada gambar 3.7, fungsi load_weather_data menyimpan data hasil transformasi ke dalam tabel PostgreSQL bernama weather_data menggunakan PostgresHook. Jika tabel belum ada, akan dibuat terlebih dahulu, kemudian data dimasukkan melalui query SQL.

```python
# DAG pipeline
weather_data = extract_weather_data()
transformed_data = transform_weather_data(weather_data)
load_weather_data(transformed_data)
```

Gambar 3.8 Membuat *Pipeline ETL API Mateo*

Pada gambar 3.8, *Pipeline* ini menunjukkan bagaimana *Airflow* dapat digunakan untuk mengotomatisasi alur kerja pengambilan dan penyimpanan data dari sumber eksternal ke sistem database.



Gambar 3.9 Hasil *coding DAG ETL Weather*

Pada gambar 3.9, menunjukkan visualisasi dari sebuah DAG (Directed Acyclic Graph) pada Apache *Airflow* dengan nama weather_etl_*pipeline*. DAG ini menjalankan proses ETL (Extract, Transform, Load) untuk data cuaca dari API Open-Meteo, yang terdiri dari tiga tahapan utama dan divisualisasikan dalam tiga node terhubung secara berurutan.

Node pertama, extract_weather_data, bertugas untuk mengambil data cuaca berdasarkan koordinat geografis kota London menggunakan API open-meteo.com. Task ini menggunakan HttpHook untuk melakukan permintaan HTTP dan mengembalikan data dalam format JSON.

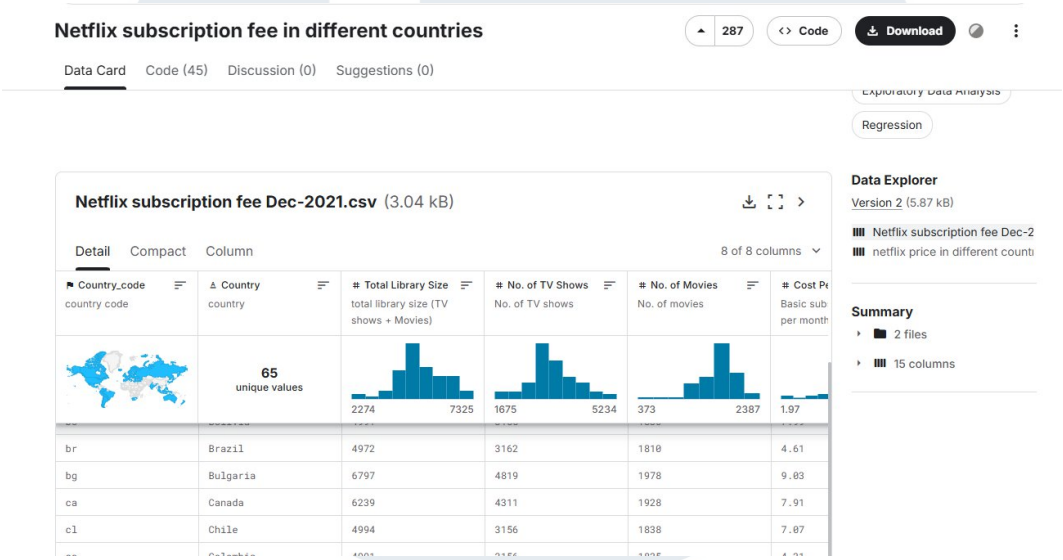
Node kedua, transform_weather_data, menjalankan proses transformasi data dari hasil ekstraksi. Data cuaca mentah disaring menjadi elemen-elemen penting seperti suhu (temperature), kecepatan angin (windspeed), arah angin (winddirection), dan kode cuaca (weathercode), yang dikemas dalam struktur dictionary Python.

Node terakhir, load_weather_data, bertanggung jawab untuk memuat data ke dalam basis data PostgreSQL. Task ini menggunakan PostgresHook untuk terkoneksi ke database, memastikan tabel weather_data tersedia atau membuatnya jika belum ada, lalu menyimpan data yang telah ditransformasi ke dalam tabel tersebut.

DAG ini dijadwalkan untuk dieksekusi setiap hari (@daily) dan tidak menjalankan catch-up task untuk tanggal sebelumnya (catchup=False). Visualisasi

DAG memperlihatkan alur ETL secara jelas dan terstruktur, mencerminkan praktik yang efisien dalam *pipeline* data modern.

3.2.5 *Project Dummy (Data Subscription)*



Gambar 3.10 Mengambil Data *Netflix Subscription*

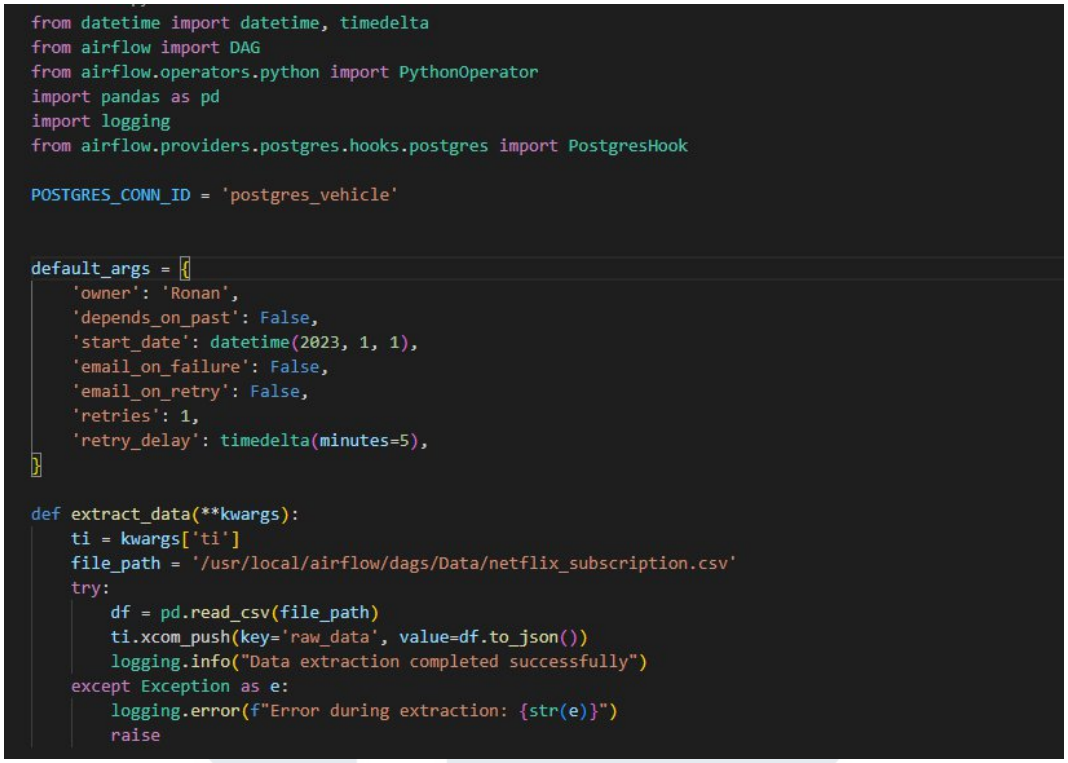
Pada gambar 3.10, dataset "*Netflix Subscription*" yang tersedia di *Kaggle* merupakan kumpulan data yang menyajikan informasi harga langganan *Netflix* di berbagai negara beserta jumlah konten yang tersedia di tiap negara tersebut. Dataset ini mencakup beberapa kolom penting seperti nama negara, mata uang lokal, harga langganan untuk tiga jenis paket (Basic, Standard, dan Premium), harga dalam USD, serta jumlah total konten, jumlah TV show, dan jumlah film. Selain itu, terdapat juga informasi seperti rata-rata biaya per konten atau yang dikenal dengan cost per title.

Dataset ini sangat cocok digunakan dalam workflow Extract, Transform, Load (ETL) menggunakan Apache *Airflow* karena struktur datanya rapi, relevan secara analitis, dan mudah untuk dilakukan pembersihan dan transformasi. Dalam tahap data cleaning, dataset ini dapat dibersihkan dari nilai kosong (missing values), data duplikat, serta standarisasi format numerik, misalnya menghapus simbol mata

uang pada nilai harga sebelum dikonversi ke tipe data float. Validasi juga diperlukan untuk memastikan bahwa tidak ada nilai tidak logis seperti harga langganan yang bernilai negatif.

Pada tahap transformasi data, berbagai teknik dapat diterapkan untuk memperkaya insight dari data mentah. Contohnya, menghitung rata-rata harga langganan dari ketiga paket untuk tiap negara, menghitung rasio jumlah TV Show terhadap jumlah film, dan mengelompokkan negara berdasarkan kategori harga langganan seperti Low Cost, Medium Cost, dan High Cost. Transformasi ini juga dapat mencakup penambahan kolom baru seperti Average Subscription Cost, TV_to_Movie_Ratio, atau Price Tier yang sangat bermanfaat dalam proses analisis lebih lanjut.

Setelah melalui proses cleaning dan transformasi, data ini kemudian dimuat (load) ke dalam sistem database seperti PostgreSQL untuk disimpan secara terstruktur. Selanjutnya, data dapat divisualisasikan menggunakan Power BI dalam bentuk dashboard yang interaktif dan informatif, seperti perbandingan harga antar negara, jumlah konten per wilayah, dan rasio biaya terhadap jumlah konten. Dengan fleksibilitas dan kekayaan metrik yang tersedia dalam dataset ini, penggunaan dalam *pipeline* Apache *Airflow* memungkinkan proses otomasi pengolahan data yang efisien, terstruktur, dan siap digunakan untuk analisis bisnis skala global.



```python
from datetime import datetime, timedelta
from airflow import DAG
from airflow.operators.python import PythonOperator
import pandas as pd
import logging
from airflow.providers.postgres.hooks.postgres import PostgresHook

POSTGRES_CONN_ID = 'postgres_vehicle'


default_args = {
    'owner': 'Ronan',
    'depends_on_past': False,
    'start_date': datetime(2023, 1, 1),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

def extract_data(**kwargs):
    ti = kwargs['ti']
    file_path = '/usr/local/airflow/dags/Data/netflix_subscription.csv'
    try:
        df = pd.read_csv(file_path)
        ti.xcom_push(key='raw_data', value=df.to_json())
        logging.info("Data extraction completed successfully")
    except Exception as e:
        logging.error(f"Error during extraction: {str(e)}")
        raise
```

Gambar 3.11 Membuat Coding ETL Netflix Subscription 1

Pada gambar 3.11, merupakan sebuah *pipeline* ETL (*Extract, Transform, Load*) yang dikembangkan menggunakan *Apache Airflow*, dengan tujuan utama untuk memproses dan menganalisis data langganan layanan streaming *Netflix* yang tersimpan dalam format CSV. Data tersebut dibaca menggunakan library pandas, lalu disimpan sementara dalam sistem Airflow melalui fitur yang disebut *XCom*, agar bisa digunakan oleh proses selanjutnya dalam pipeline. Task ini bertanggung jawab untuk membaca file Netflix_subscription.csv menggunakan pustaka Pandas, kemudian menyimpan data mentah tersebut ke dalam XCom, yaitu fitur Airflow untuk membagikan data antar-task. Dengan mekanisme ini, data dapat diteruskan dengan efisien ke task berikutnya tanpa perlu menulis ulang ke penyimpanan sementara.

```python
def clean_data(**kwargs):
    ti = kwargs['ti']
    raw_data = ti.xcom_pull(task_ids='extract_data', key='raw_data')
    df = pd.read_json(raw_data)

    df.dropna(inplace=True)
    df.drop_duplicates(inplace=True)

    ti.xcom_push(key='validated_data', value=df.to_json())
    logging.info("Data cleaned successfully")


def transform_data(**kwargs):
    ti = kwargs['ti']
    validated_data = ti.xcom_pull(task_ids='validate_data', key='validated_data')
    df = pd.read_json(validated_data)

    try:
        price_columns = ['Cost Per Month - Basic ($)', 'Cost Per Month - Standard ($)', 'Cost Per Month - Premium ($)']
        df['Average Cost ($)'] = df[price_columns].mean(axis=1).round(2)

        df['TV_to_Movie_Ratio'] = (df['No. of TV Shows'] / df['No. of Movies']).round(2)

        def categorize_price_tier(row):
            if row['Average Cost ($)'] < 5:
                return 'Low Cost'
            elif row['Average Cost ($)'] < 10:
                return 'Medium Cost'
            else:
                return 'High Cost'

        df['Price_Tier'] = df.apply(categorize_price_tier, axis=1)
```

Gambar 3.12 Membuat Coding ETL Netflix Subscription 2

Pada gambar 3.12, Sebelumnya, proses validasi data dilakukan melalui pengecekan nilai kosong dan entri yang tidak valid, namun kini telah disederhanakan dengan langsung menghapus baris yang mengandung nilai NaN (kosong) dan data duplikat. Proses pembersihan ini penting untuk memastikan kualitas data sebelum dilakukan analisis lebih lanjut.

```python
        df['Price_Tier'] = df.apply(categorize_price_tier, axis=1)

        ti.xcom_push(key='transformed_data', value=df.to_json())
        logging.info("Data transformation completed successfully")
    except Exception as e:
        logging.error(f"Error during transformation: {str(e)}")
        raise
```

Gambar 3.13 Membuat Coding ETL Netflix Subscription 3

Pada gambar 3.13, dilakukan beberapa proses transformasi data yang bersifat analitis, antara lain:

- Menghitung rata-rata biaya langganan untuk tiga jenis paket: basic, standard, dan premium.

- Menghitung rasio jumlah TV shows terhadap jumlah film sebagai indikator keberagaman konten per negara.
- Mengklasifikasikan tiap negara ke dalam kategori biaya: Low Cost, Medium Cost, dan High Cost, berdasarkan harga rata-rata paket langganannya.

```
def load_results(**kwargs):
    ti = kwargs['ti']
    transformed_data = ti.xcom_pull(task_ids='transform_data', key='transformed_data')
    df = pd.read_json(transformed_data)

    pg_hook = PostgresHook(postgres_conn_id=POSTGRES_CONN_ID)
    conn = pg_hook.get_conn()
    cursor = conn.cursor()

    cursor.execute("DROP TABLE IF EXISTS netflix_subscriptions")

    cursor.execute("""
        CREATE TABLE netflix_subscriptions (
            country_code TEXT PRIMARY KEY,
            country TEXT,
            total_library_size INTEGER,
            no_tv_shows INTEGER,
            no_movies INTEGER,
            cost_basic DECIMAL(10,2),
            cost_standard DECIMAL(10,2),
            cost_premium DECIMAL(10,2),
            average_cost DECIMAL(10,2),
            tv_to_movie_ratio DECIMAL(10,2),
            price_tier TEXT
        )
    """)
```

Gambar 3.14 Membuat Coding ETL Netflix Subscription 4

Pada gambar 3.14, *task* ini bertugas untuk membuat ulang (drop dan create) tabel di dalam database PostgreSQL. Data hasil transformasi kemudian dimasukkan (insert) ke tabel tersebut agar dapat digunakan untuk pelaporan, visualisasi, atau analisis lebih lanjut.

```python
df = df.rename(columns={
    'Country_code': 'country_code',
    'Country': 'country',
    'Total Library Size': 'total_library_size',
    'No. of TV Shows': 'no_tv_shows',
    'No. of Movies': 'no_movies',
    'Cost Per Month - Basic ($)': 'cost_basic',
    'Cost Per Month - Standard ($)': 'cost_standard',
    'Cost Per Month - Premium ($)': 'cost_premium',
    'Average Cost ($)': 'average_cost',
    'TV_to_Movie_Ratio': 'tv_to_movie_ratio',
    'Price_Tier': 'price_tier'
})

data_to_insert = [
    (
        row['country_code'],
        row['country'],
        row['total_library_size'],
        row['no_tv_shows'],
        row['no_movies'],
        row['cost_basic'],
        row['cost_standard'],
        row['cost_premium'],
        row['average_cost'],
        row['tv_to_movie_ratio'],
        row['price_tier']
    )
    for _, row in df.iterrows()
]
```

Gambar 3.15 Membuat Coding ETL Netflix Subscription 5

Pada Gambar 3.15, dilakukan proses *rename* atau penggantian nama terhadap beberapa kolom pada data agar lebih mudah dibaca dan dipahami. Penggantian nama ini bertujuan untuk meningkatkan keterbacaan dan konsistensi penamaan kolom, terutama jika data tersebut akan dianalisis lebih lanjut atau digunakan dalam visualisasi. Selain itu, pada tahap ini juga dilakukan proses insert row, yaitu menambahkan baris data baru ke dalam tabel. Penambahan data ini dapat digunakan untuk melengkapi informasi, menguji alur pemrosesan, atau menambahkan nilai default yang diperlukan sebelum masuk ke tahap transformasi atau analisis berikutnya. Dengan langkah ini, data menjadi lebih siap dan terstruktur untuk digunakan dalam proses selanjutnya.

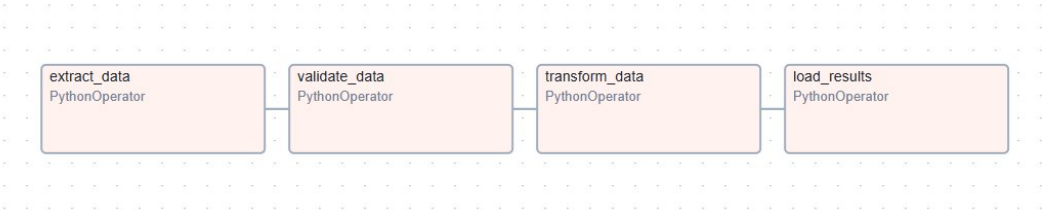
```
    cursor.executemany(
        """
        INSERT INTO netflix_subscriptions
        (country_code, country, total_library_size, no_tv_shows, no_movies,
         cost_basic, cost_standard, cost_premium, average_cost,
         tv_to_movie_ratio, price_tier)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
        """,
        data_to_insert
    )

    conn.commit()
    cursor.close()
    conn.close()
    logging.info("Data successfully loaded into PostgreSQL")

with DAG(
    'netflix_subscription_analysis_v4',
    default_args=default_args,
    description='Pipeline for analyzing Netflix subscription fees across countries',
    schedule_interval='@monthly',
    catchup=False,
) as dag:

    extract_task = PythonOperator(
        task_id='extract_data',
        python_callable=extract_data,
        provide_context=True,
    )
```

Gambar 3.16 Membuat Coding ETL Netflix Subscription 6

Pada gambar 3.16, kode ini merupakan bagian akhir dari pipeline ETL di Airflow. Data yang sudah diproses akan dimasukkan ke dalam database PostgreSQL melalui perintah executemany(). Setelah data berhasil dimasukkan, koneksi ditutup dan log sukses dicatat. Dengan alur ini, data langganan Netflix dari berbagai negara dapat diotomatisasi proses ekstraksi dan penyimpanannya ke database untuk dianalisis lebih lanjut.

Gambar 3.17 Membuat Coding ETL Netflix Subscription 7

Pada gambar 3.17, merupakan definisi pipeline ETL di Airflow yang terdiri dari empat tahap: extract, clean, transform, dan load. Masing-masing tahap dijalankan sebagai task menggunakan PythonOperator, dan dieksekusi secara berurutan. Task extract_task mengambil data, clean_task membersihkannya, transform_task memprosesnya, dan load_task menyimpan hasil akhirnya ke database. Alur ini membuat proses pengolahan data menjadi otomatis dan terstruktur.



Gambar 3.18 Hasil Pengerjaan DAG *Netflix Subscription*

Pada gambar 3.18 menunjukkan visualisasi DAG (Directed Acyclic Graph) pada Apache *Airflow* dengan nama *Netflix*_subscription_analysis_v4. DAG ini terdiri dari empat tahapan yang menggambarkan proses ETL (Extract, Transform, Load) dengan tambahan proses validasi data. Setiap tahapan direpresentasikan oleh satu task yang dijalankan secara berurutan dari kiri ke kanan menggunakan PythonOperator.

Task pertama adalah extract_data, yang bertugas mengekstrak data dari file CSV berisi informasi biaya langganan *Netflix* per negara. File ini dibaca menggunakan pandas dan hasilnya dikirim antar-task melalui XCom.

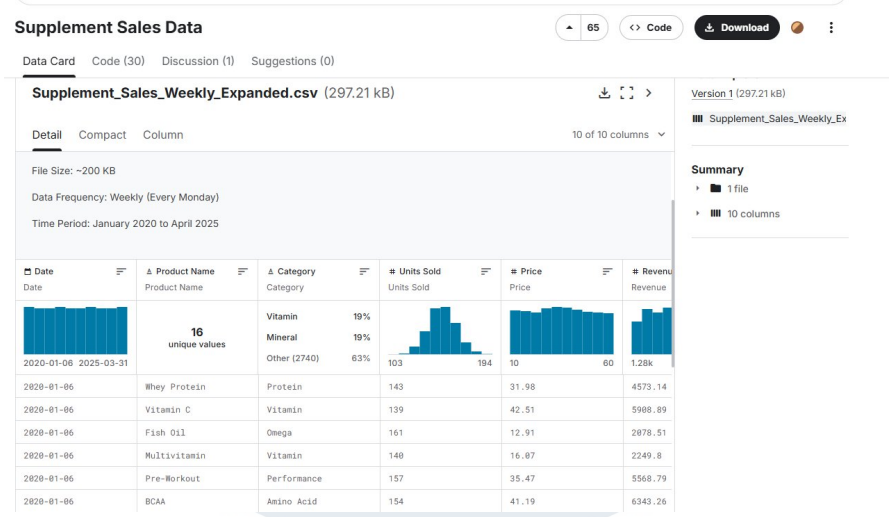
Task kedua, validate_data, merupakan proses validasi atau pembersihan data. Pada tahapan ini, data akan dibersihkan dari nilai kosong dan data duplikat menggunakan dropna() dan drop_duplicates(), kemudian disimpan kembali ke XCom untuk diproses lebih lanjut.

Task ketiga adalah transform_data, yang menjalankan proses transformasi data. Beberapa kolom harga digabungkan untuk menghasilkan rata-rata biaya langganan (Average Cost ($)), rasio antara jumlah TV Shows dan Movies dihitung (TV_to_Movie_Ratio), serta setiap negara diklasifikasikan ke dalam tiga tier harga: Low, Medium, dan High berdasarkan nilai rata-rata biaya.

Task terakhir, load_results, bertanggung jawab untuk memuat data yang telah diproses ke dalam basis data PostgreSQL. Tabel *Netflix*_subscriptions akan dibuat ulang jika sudah ada, dan data hasil transformasi akan dimasukkan ke dalam tabel tersebut melalui koneksi yang disediakan oleh PostgresHook.

DAG ini dijalankan dengan jadwal bulanan (@monthly) dan tidak melakukan catch-up untuk eksekusi masa lalu. *Pipeline* ini mendukung proses analisis biaya langganan *Netflix* antar negara secara otomatis, terstruktur, dan dapat diandalkan dalam sistem produksi.

3.2.6 *Data Supplement Sales*



Gambar 3.19 Mengambil *Data Supplement Sales*

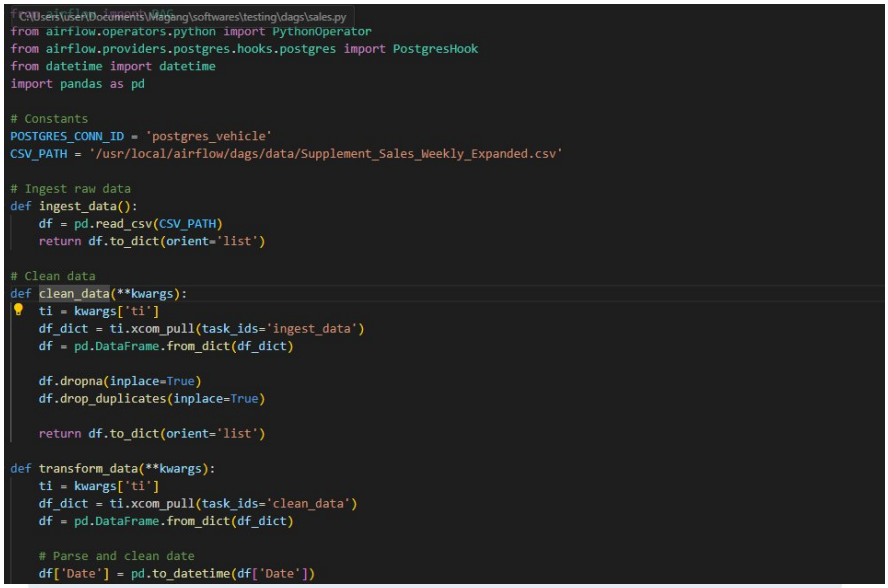
Pada gambar 3.19, dataset "*Supplement Sales Data*" yang tersedia di Kaggle merupakan dataset yang menyajikan informasi penjualan produk suplemen dari waktu ke waktu. Dataset ini mencakup berbagai fitur penting seperti tanggal transaksi (Date), nama produk (Product Name), kategori produk (Category), jumlah unit yang terjual (Units Sold), harga per unit (Unit Price), wilayah penjualan (Region), dan total pendapatan (Revenue). Dengan struktur yang terorganisir, dataset ini menggambarkan tren penjualan dan performa produk secara 37ndicato, yang sangat bermanfaat dalam analisis bisnis dan perencanaan strategi pemasaran.

Dataset ini sangat sesuai untuk digunakan dalam workflow ETL (Extract, Transform, Load) dengan Apache *Airflow*, terutama untuk simulasi otomatisasi *pipeline* penjualan. Dalam tahap data cleaning, 37ndicat awal yang dilakukan adalah membersihkan nilai kosong, menghapus data duplikat, dan memastikan format tanggal telah sesuai agar dapat diolah lebih lanjut. Selain itu, perlu dilakukan

validasi data numerik seperti memastikan nilai Units Sold dan Unit Price tidak 38 ndicato atau nol, untuk menjaga integritas data sebelum digunakan dalam analisis.

Proses transformasi data melibatkan pengolahan data mentah menjadi informasi yang lebih bermakna. Salah satu transformasi penting dalam dataset ini adalah pembuatan kolom baru 38ndicat Total Revenue, yaitu hasil perkalian antara Units Sold dan Unit Price, yang kemudian digunakan untuk menganalisis pendapatan per produk maupun per kategori. Selain itu, data juga dapat dikelompokkan berdasarkan waktu (mingguan, bulanan) untuk melihat pola musiman atau tren pertumbuhan penjualan. Transformasi lain yang bermanfaat termasuk pengelompokan produk dengan performa tinggi, analisis berdasarkan wilayah, serta kalkulasi metrik bisnis seperti rata-rata pendapatan per unit atau kontribusi kategori terhadap total penjualan.

Setelah dibersihkan dan ditransformasi, data akan dimuat ke dalam database PostgreSQL sebagai tahap akhir dari *pipeline*. Dari sana, data siap divisualisasikan dalam dashboard Power BI untuk menampilkan 38ndicator kinerja utama (KPI), grafik tren pendapatan, distribusi penjualan berdasarkan kategori, hingga perbandingan produk. Dengan mengotomatisasi proses ini menggunakan Apache *Airflow*, *pipeline* ETL menjadi lebih efisien, terjadwal, dan mudah dipantau, sehingga sangat mendukung kebutuhan analitik bisnis berbasis data secara real time dan akurat.



Gambar 3.20 Membuat Coding Supplement Sales 1

Pada gambar 3.20, merupakan definisi dari sebuah DAG (Directed Acyclic Graph) pada Apache Airflow. DAG ini dirancang untuk menjalankan proses ETL (*Extract, Transform, Load)* terhadap data penjualan suplemen dan menyimpannya ke dalam database PostgreSQL.

Task ini bertugas membaca data dari sumber CSV menggunakan pustaka Pandas, lalu mengonversinya ke dalam format dictionary untuk dikirimkan melalui XCom ke task selanjutnya. Proses ini memungkinkan pertukaran data antar-task tanpa perlu menyimpan file secara fisik.

Task ini melakukan proses pembersihan data, yaitu menghapus baris kosong dan menghilangkan data yang bersifat duplikat. Pembersihan ini penting untuk menjaga kualitas dan akurasi data sebelum dianalisis lebih lanjut.

```python
def transform_data(**kwargs):
    ti = kwargs['ti']
    df_dict = ti.xcom_pull(task_ids='clean_data')
    df = pd.DataFrame.from_dict(df_dict)

    # Parse and clean date
    df['Date'] = pd.to_datetime(df['Date'])

    # Rename columns
    df.rename(columns={
        'Product Name': 'Product',
        'Price': 'Unit Price'
    }, inplace=True)

    # Compute Total Revenue
    df['Total Revenue'] = df['Units Sold'] * df['Unit Price']

    # Select and order final columns
    df = df[['Date', 'Product', 'Category', 'Units Sold', 'Unit Price', 'Total Revenue']]

    # Convert datetime to ISO string to make it JSON serializable
    df['Date'] = df['Date'].dt.strftime('%Y-%m-%d')

    # Push to XCom as dictionary with only JSON-serializable types
    ti.xcom_push(key='average_units_sold', value=df['Units Sold'].mean())

    return df.to_dict(orient='list')
```

Gambar 3.21 Membuat Coding Supplement Sales 2

Pada gambar 3.21, dilakukan berbagai transformasi data, antara lain:

- Mengonversi format tanggal agar sesuai dengan standar analisis.
- Mengganti nama-nama kolom agar lebih deskriptif dan mudah dibaca.
- Menghitung nilai Total Revenue sebagai hasil perkalian antara harga per unit dan jumlah unit terjual.
- Menyusun ulang urutan kolom agar lebih rapi.
- Menghitung rata-rata penjualan, yang hasilnya dikirim ke XCom sebagai referensi untuk analisis tambahan.

```python
# Validate data
def validate_data(**kwargs):
    ti = kwargs['ti']
    df_dict = ti.xcom_pull(task_ids='transform_data')
    df = pd.DataFrame.from_dict(df_dict)

    assert not df.empty, "DataFrame is empty!"
    assert df['Units Sold'].min() >= 0, "Negative units sold found!"

# Store data in PostgreSQL
def store_data(**kwargs):
    ti = kwargs['ti']
    df_dict = ti.xcom_pull(task_ids='transform_data')
    df = pd.DataFrame.from_dict(df_dict)

    pg_hook = PostgresHook(postgres_conn_id=POSTGRES_CONN_ID)
    conn = pg_hook.get_conn()
    cursor = conn.cursor()

    cursor.execute("DROP TABLE IF EXISTS supplement_sales")

    create_table_query = """
        CREATE TABLE supplement_sales (
            "Date" DATE,
            "Product" TEXT,
            "Category" TEXT,
            "Units_Sold" INTEGER,
            "Unit_Price" FLOAT,
            "Total_Revenue" FLOAT
        )
    """
    cursor.execute(create_table_query)
```

Gambar 3.22 Membuat Coding Supplement Sales 3

Pada Gambar 3.22, task ini bertanggung jawab untuk melakukan proses validasi data sebelum data tersebut disimpan ke dalam database. Validasi ini dilakukan untuk memastikan bahwa data yang akan dimuat telah memenuhi standar kualitas yang diperlukan. Langkah pertama dalam proses ini adalah memverifikasi bahwa dataset tidak kosong, sehingga dapat dipastikan bahwa proses sebelumnya seperti ekstraksi, pembersihan, dan transformasi telah berhasil menghasilkan data yang dapat digunakan.

Selain itu, task ini juga memeriksa nilai pada kolom *Units Sold* untuk memastikan tidak terdapat nilai negatif yang dapat menyebabkan ketidaksesuaian atau kesalahan dalam analisis dan pelaporan. Setelah semua data dinyatakan valid, task ini akan membuat ulang (recreate) tabel supplement_sales di PostgreSQL.

Pembuatan ulang tabel ini biasanya dilakukan untuk memastikan bahwa struktur tabel sesuai dengan data yang baru, sekaligus menghapus data lama jika ada. Setelah itu, seluruh data yang telah melewati proses validasi akan disimpan ke dalam tabel tersebut, sehingga database berisi data yang bersih, akurat, dan siap digunakan untuk keperluan analisis lanjutan seperti peramalan penjualan atau pelaporan kinerja produk.

```
    # Prepare and insert data
    data_to_insert = list(df.itertuples(index=False, name=None))
    insert_query = """
        INSERT INTO supplement_sales
        ("Date", "Product", "Category", "Units_Sold", "Unit_Price", "Total_Revenue")
        VALUES (%s, %s, %s, %s, %s, %s)
    """
    cursor.executemany(insert_query, data_to_insert)
    conn.commit()

    cursor.close()
    conn.close()

# Default DAG args
default_args = {
    'owner': 'Ronan',
    'start_date': datetime(2023, 10, 1),
    'retries': 1,
}

# DAG definition
dag = DAG(
    'supplement_sales_pipeline_v4',
    default_args=default_args,
    description='ETL pipeline for supplement sales data (forecast-ready)',
    schedule_interval='@daily',
    catchup=False,
)
```

Gambar 3.23 Membuat Coding Supplement Sales 4

Pada Gambar 3.23, kode tersebut digunakan untuk memasukkan data penjualan suplemen ke dalam database PostgreSQL serta mendefinisikan pipeline ETL menggunakan Apache Airflow. Proses dimulai dengan menyiapkan data dari sebuah DataFrame, di mana setiap baris data dikonversi menjadi format tuple agar dapat dimasukkan ke dalam tabel menggunakan perintah SQL INSERT. Query tersebut menyisipkan data ke dalam tabel supplement_sales, mencakup kolom-kolom seperti tanggal, produk, kategori, jumlah unit terjual, harga satuan, dan total pendapatan. Setelah data berhasil dimasukkan, koneksi ke database ditutup untuk menjaga efisiensi. Pipeline ini dirancang untuk memastikan data penjualan selalu diperbarui secara rutin, sehingga dapat digunakan untuk analisis, pelaporan, maupun prediksi penjualan di masa mendatang.

```
# Tasks
ingest_task = PythonOperator(
    task_id='ingest_data',
    python_callable=ingest_data,
    dag=dag,
)

clean_task = PythonOperator(
    task_id='clean_data',
    python_callable=clean_data,
    provide_context=True,
    dag=dag,
)

transform_task = PythonOperator(
    task_id='transform_data',
    python_callable=transform_data,
    provide_context=True,
    dag=dag,
)

validate_task = PythonOperator(
    task_id='validate_data',
    python_callable=validate_data,
    provide_context=True,
    dag=dag,
)

store_task = PythonOperator(
    task_id='store_data',
    python_callable=store_data,
    provide_context=True,
    dag=dag,
)
```

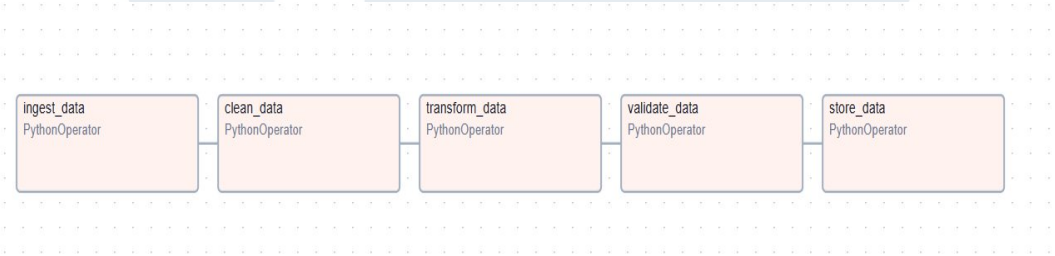
Gambar 3.24 Membuat Coding Supplement Sales 5

Pada gambar 3.24, Kode tersebut merupakan bagian dari definisi task dalam pipeline ETL menggunakan Apache Airflow. Setiap task menjalankan fungsi Python untuk melakukan proses bertahap, mulai dari mengambil data (ingest), membersihkan, mentransformasi, memvalidasi, hingga menyimpan data. Seluruh task ini terhubung ke dalam satu DAG dan disusun untuk berjalan secara otomatis dan berurutan sesuai alur pemrosesan data.

```
)

# Task pipeline
ingest_task >> clean_task >> transform_task >> validate_task >> store_task
```

Gambar 3.25 Membuat Coding Supplement Sales 6

Pada gambar 3.25, Seluruh proses ETL ini dirancang secara berurutan dan otomatis, sehingga data penjualan dapat diproses secara konsisten setiap hari. *Pipeline* ini mendukung kebutuhan pelaporan dan analisis lanjutan yang berbasis data, serta memastikan bahwa informasi bisnis yang dihasilkan bersifat akurat, bersih, dan siap digunakan oleh tim terkait.



Gambar 3.26 Hasil Coding DAG Supplement Sales

Pada gambar 3.26, memperlihatkan urutan task dalam sebuah DAG Apache *Airflow* bernama supplement_sales_*pipeline*_v4 yang menjalankan proses ETL lengkap untuk data penjualan suplemen. DAG ini terdiri dari lima tahapan yang dijalankan secara berurutan, masing-masing diwakili oleh PythonOperator, dan dirancang untuk mengolah data dari file CSV hingga dimasukkan ke dalam database PostgreSQL.

Task pertama adalah ingest_data, yang bertugas membaca data mentah dari file CSV Supplement_Sales_Weekly_Expanded.csv. Data dibaca dengan pandas dan dikonversi ke format dictionary agar bisa disimpan sementara melalui XCom.

Task kedua, clean_data, menjalankan proses pembersihan data. Tahapan ini menghapus nilai yang kosong (NaN) dan data duplikat menggunakan fungsi

dropna() dan drop_duplicates() untuk memastikan data bersih sebelum diproses lebih lanjut.

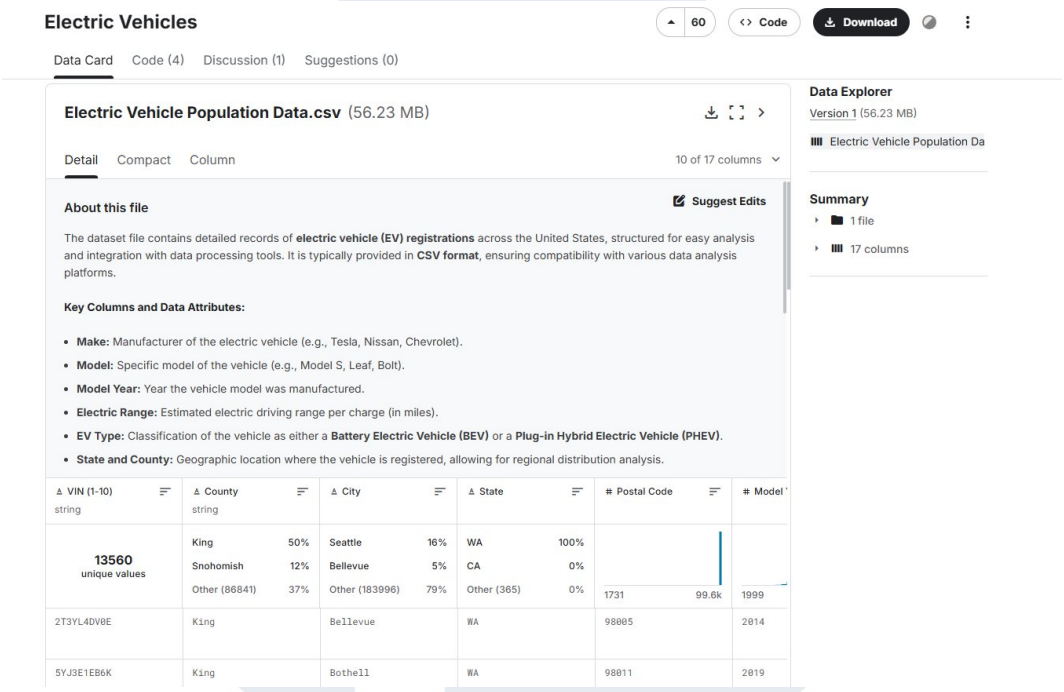
Task ketiga, transform_data, melakukan proses transformasi. Kolom tanggal (Date) diubah ke format datetime, beberapa kolom diganti namanya (seperti Product Name menjadi Product), dan kolom baru Total Revenue dihitung dari hasil perkalian Units Sold dan Unit Price. Selain itu, nilai rata-rata unit terjual juga dihitung dan disimpan melalui XCom.

Task keempat, validate_data, berfungsi untuk memastikan kualitas data. Task ini memverifikasi bahwa dataset tidak kosong dan tidak ada nilai negatif pada kolom Units Sold.

Task terakhir adalah store_data, yang menyimpan data hasil transformasi ke dalam tabel PostgreSQL bernama supplement_sales. Jika tabel sudah ada, maka akan dihapus dan dibuat ulang, lalu data disisipkan dengan metode batch menggunakan executemany.

DAG ini dijadwalkan berjalan setiap hari (@daily) dan tidak melakukan catch-up pada eksekusi yang terlewat (catchup=False). *Pipeline* ini dirancang untuk mengotomatisasi proses data penjualan secara bersih, terstruktur, dan siap digunakan untuk analisis lanjutan seperti forecasting

3.2.7 *Data Electric Vehicle*



Gambar 3.27 Mengambil *Data Electric Vehicles*

Pada gambar 3.27 dataset "*Electric Vehicle Specifications*" merupakan 46 eputusa data yang menyediakan informasi teknis dari berbagai model kendaraan 46eputus (Electric Vehicle/EV) yang tersedia secara global. Dataset ini mencakup sejumlah kolom penting seperti Brand, Model, Vehicle Class, Battery Capacity (kWh), Charging Time, Electric Range (km), Top Speed, dan Acceleration. Informasi ini sangat relevan dalam analisis performa kendaraan 46eputus serta untuk memahami kemajuan teknologi di sektor otomotif berbasis energi terbarukan.

Dataset ini sangat cocok digunakan dalam workflow ETL (Extract, Transform, Load) menggunakan Apache *Airflow*, terutama untuk simulasi pengolahan data teknis dengan tujuan analisis performa dan efisiensi kendaraan. Dalam tahap data cleaning, dilakukan pembersihan data dari nilai kosong (missing values), duplikat, serta pengecekan logis terhadap nilai-nilai seperti Electric Range dan Battery Capacity, yang tidak boleh bernilai nol. Format data numerik juga perlu distandarkan, terutama jika terdapat satuan atau format yang tidak konsisten antar baris.

Tahap transformasi data dalam *pipeline* ini sangat penting untuk menghasilkan insight yang berguna. Transformasi yang dilakukan antara lain: menghitung performa seperti efisiensi jarak tempuh per kapasitas baterai (*Electric Range per Battery Capacity*), menandai kendaraan yang memiliki range di atas rata-rata, serta menambahkan kolom kategori efisiensi seperti *"Low", "Medium",* dan *"High*". Data juga bisa dikelompokkan berdasarkan Brand atau Vehicle Class untuk mengetahui produsen mana yang paling efisien dalam teknologi baterai atau memiliki performa tertinggi dari sisi kecepatan atau daya jangkau.

Setelah proses cleaning dan transformasi selesai, data dimuat ke dalam sistem database seperti PostgreSQL. Data tersebut kemudian dapat digunakan untuk visualisasi melalui Power BI dalam bentuk dashboard interaktif yang menyajikan perbandingan performa antar merek kendaraan, efisiensi energi, distribusi kecepatan maksimum, serta jarak tempuh. *Pipeline* ini, jika dijalankan secara otomatis menggunakan *Airflow*, memungkinkan pemrosesan data teknis kendaraan secara rutin dan real time, yang bermanfaat dalam analisis pasar otomotif, studi keberlanjutan energi, atau pengambilan konsumen.

```python
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime
from airflow.providers.http.hooks.http import HttpHook
from airflow.providers.postgres.hooks.postgres import PostgresHook
import pandas as pd
import os

POSTGRES_CONN_ID = 'postgres_vehicle'

def ingest_data():
    print("path:", os.getcwd())
    df = pd.read_csv('/usr/local/airflow/dags/Data/electric_car.csv')
    return df

def clean_data(**kwargs):
    ti = kwargs['ti']
    df = ti.xcom_pull(task_ids='ingest_data')
    df.dropna(inplace=True)
    df.drop_duplicates(inplace=True)
    return df

def transform_data(**kwargs):
    ti = kwargs['ti']
    df = ti.xcom_pull(task_ids='clean_data')

    df['Range_Above_Avg'] = df['Electric Range'] > df['Electric Range'].mean()
    df['Has_Zero_Range'] = df['Electric Range'] == 0

    ti.xcom_push(key='average_electric_range', value=df['Electric Range'].mean())

    return df.to_dict()
```

Gambar 3.28 Membuat Coding Pipeline Electric Vehicles 1

Pada gambar 3.28, pipeline terdiri dari lima tahapan utama yang dieksekusi secara berurutan, yaitu: i*ngest_data, clean_data, transform_data, validate_data*, dan *store_data*. Masing-masing tahapan memiliki peran khusus dalam memastikan integritas dan kelayakan data yang akan digunakan untuk analisis.

Tahapan *ingest data* bertugas membaca data kendaraan listrik dari file CSV dan mengubahnya ke dalam format dictionary menggunakan pustaka Pandas. Hasil ekstraksi kemudian dikirim ke task selanjutnya melalui mekanisme XCom, yang memungkinkan pertukaran data antar-task tanpa perlu penyimpanan file sementara.

Di tahap *clean data*, data dibersihkan dari baris-baris yang mengandung nilai kosong (NaN) dan duplikat. Proses pembersihan ini penting untuk menjaga kualitas data sebelum dilakukan transformasi.

Tahapan *transformasi data*, termasuk penambahan dua kolom analitis:

- *Range_Above_Avg*: indikator apakah jarak tempuh kendaraan berada di atas rata-rata.
- *Has_Zero_Range*: penanda apakah kendaraan memiliki jarak tempuh nol (potensi data outlier atau tidak valid).

Selain itu, nilai rata-rata jarak tempuh juga dihitung dan disimpan ke dalam XCom sebagai referensi analitik tambahan.

```python
def validate_data(**kwargs):
    ti = kwargs['ti']
    df = ti.xcom_pull(task_ids='clean_data')
    assert not df.empty, "DataFrame is empty!"
    assert df['Electric Range'].min() >= 0, "Negative range values found!"

def store_data(**kwargs):
    ti = kwargs['ti']
    df_dict = ti.xcom_pull(task_ids='transform_data')

    df = pd.DataFrame.from_dict(df_dict)

    pg_hook = PostgresHook(postgres_conn_id=POSTGRES_CONN_ID)
    conn = pg_hook.get_conn()
    cursor = conn.cursor()

    cursor.execute("DROP TABLE IF EXISTS electric_vehicles")

    cursor.execute("""
        CREATE TABLE electric_vehicles (
            "Make" TEXT,
            "Model" TEXT,
            "Electric_Range" INTEGER,
            "Range_Above_Avg" BOOLEAN,
            "Has_Zero_Range" BOOLEAN
        )
    """)

    data_to_insert = [
        (
            row['Make'],
            row['Model'],
            row['Electric Range'],
            row['Range_Above_Avg'],
            row['Has_Zero_Range']
        )
```

Gambar 3.29 Membuat Coding Pipeline Electric Vehicles 2

Pada gambar 3.29, dilakukan verifikasi terhadap data untuk memastikan bahwa dataset tidak kosong dan tidak mengandung nilai negatif pada kolom jarak tempuh, yang dapat menandakan kesalahan input atau data tidak valid. Jika data lolos validasi, task ini akan menghapus (drop) dan membuat ulang tabel electric_vehicles di dalam database PostgreSQL. Seluruh data bersih dan tertransformasi kemudian disimpan ke dalam tabel tersebut.

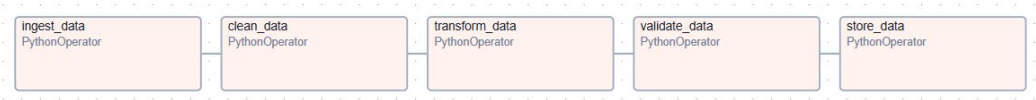
Gambar 3.30 Membuat Coding Pipeline Electric Vehicles 3

Pada gambar 3.30, merupakan implementasi *pipeline* ETL *(Extract, Transform, Load)* menggunakan Apache Airflow yang dirancang untuk memproses data kendaraan listrik. *Pipeline* ini didefinisikan dalam sebuah DAG (*Directed Acyclic Graph*) bernama electric_vehicle_*pipeline* dan dijadwalkan untuk berjalan setiap hari *(@daily*), guna memastikan data selalu diperbarui secara berkala.

```
clean_task = PythonOperator(
    task_id='clean_data',
    python_callable=clean_data,
    dag=dag,
)

transform_task = PythonOperator(
    task_id='transform_data',
    python_callable=transform_data,
    dag=dag,
)

validate_task = PythonOperator(
    task_id='validate_data',
    python_callable=validate_data,
    dag=dag,
)

store_task = PythonOperator(
    task_id='store_data',
    python_callable=store_data,
    dag=dag,
)

ingest_task >> clean_task >> transform_task >> validate_task >> store_task
```

Gambar 3.31 Membuat Coding Pipeline Electric Vehicles 4

Pada gambar 3.31, dengan alur yang otomatis dan terjadwal, *pipeline* ini memungkinkan proses yang efisien dalam membersihkan, menganalisis, dan menyimpan data kendaraan listrik ke dalam sistem basis data. Hasil akhir dari *pipeline* dapat digunakan untuk keperluan pelaporan, analisis performa kendaraan, ataupun studi tren adopsi kendaraan listrik berdasarkan spesifikasi dan jarak tempuh.



Gambar 3.32 Hasil *Coding DAG Electric Vehicles*

Gambar 3.32 merupakan DAG (*Directed Acyclic Graph*) *electric vehicle pipeline* merupakan alur kerja otomatisasi ETL (Extract, Transform, Load) yang dibuat menggunakan Apache *Airflow* untuk memproses data kendaraan listrik.

Pipeline ini terdiri dari lima tahap utama yang saling terhubung dan dijalankan secara berurutan setiap hari. Tahap pertama, ingest_data, bertugas membaca data mentah dari file CSV yang berisi informasi kendaraan listrik. Data tersebut kemudian dibersihkan pada tahap clean_data dengan menghapus nilai kosong dan duplikat agar lebih akurat dan siap diproses lebih lanjut.

Selanjutnya, pada tahap transform_data, data ditambahkan kolom baru seperti Range_Above_Avg untuk menandai kendaraan dengan jarak tempuh di atas rata-rata, serta Has_Zero_Range untuk mengidentifikasi kendaraan yang tidak memiliki jarak tempuh. Tahap keempat adalah validate_data, di mana dilakukan pemeriksaan bahwa data tidak kosong dan semua nilai jarak listrik bernilai positif. Terakhir, data yang telah divalidasi dimasukkan ke dalam database PostgreSQL melalui tahap store_data. Tabel akan dihapus dan dibuat ulang setiap kali *pipeline* dijalankan untuk memastikan data yang tersimpan selalu terbaru.

Secara keseluruhan, *pipeline* ini membantu memastikan bahwa data kendaraan listrik dikelola dengan baik, siap dianalisis, dan disimpan secara terstruktur dalam sistem basis data.

3.2.8 *PowerBI Visualization*

3.2.8.1 Dashboard *Netflix* Subscription

Gambar 3.33 Membuat Dashboard *Netflix Subscription*

Pada gambar 3.33 ini dirancang untuk memberikan gambaran menyeluruh mengenai biaya langganan *Netflix* dan jumlah konten yang tersedia di berbagai

negara. Tujuan utama dari dashboard ini adalah untuk menyajikan data dalam bentuk visual yang informatif, interaktif, dan mudah dipahami, sehingga dapat mendukung proses analisis dan pengambilan keputusan secara efektif. Dashboard terdiri dari beberapa komponen visual utama yang saling melengkapi dalam menyampaikan informasi, yaitu:

A. *Indicators*

Komponen ini digunakan untuk menampilkan total jumlah film dan acara TV secara global. Angka besar diletakkan di bagian atas dashboard untuk langsung menarik perhatian pengguna dan memberikan gambaran umum mengenai skala data yang sedang dianalisis.

B. *BarChart*

Visualisasi ini digunakan untuk menampilkan rata-rata biaya langganan per negara. Grafik batang horizontal memudahkan pengguna dalam membandingkan harga antarnegara secara langsung dan cepat, serta membantu mengidentifikasi negara dengan harga langganan tertinggi maupun terendah.

C. *WorldMap*

Peta dunia digunakan untuk menunjukkan distribusi geografis dari harga langganan *Netflix*. Dengan visualisasi ini, pengguna dapat dengan mudah melihat pola penyebaran harga berdasarkan lokasi negara, yang sangat berguna dalam melakukan analisis spasial.

D. *Table*

Tabel ini menyajikan informasi numerik secara lengkap dan terstruktur, seperti harga langganan per paket, jumlah film dan acara TV di setiap negara, serta klasifikasi kategori harga. Tabel berfungsi sebagai pelengkap visualisasi lainnya, sekaligus memberikan fleksibilitas bagi pengguna yang ingin melihat data secara lebih mendalam.

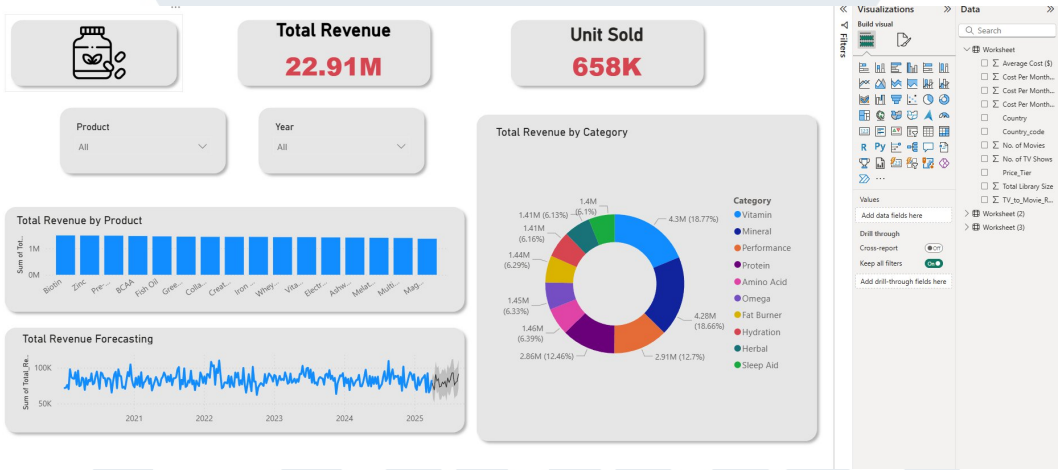
E. Filter Interaktif

Filter berdasarkan kategori harga (*Low, Medium, High Cost*) ditambahkan untuk memberikan fleksibilitas dalam eksplorasi data. Fitur ini

memungkinkan pengguna menyesuaikan tampilan dashboard sesuai dengan kebutuhan analisis tertentu, misalnya hanya menampilkan negara-negara dengan harga langganan rendah.

Seluruh komponen visualisasi tersebut dirancang untuk mendukung pemahaman data secara menyeluruh, cepat, dan akurat. Kombinasi antara grafik, peta, tabel, dan filter interaktif memberikan pengalaman eksplorasi data yang intuitif dan mendalam. Dashboard ini berfungsi sebagai alat bantu penting dalam menyampaikan insight terkait strategi harga dan distribusi konten *Netflix* di berbagai negara, baik untuk tujuan riset pasar, analisis performa, maupun perencanaan bisnis.

3.2.8.2 *Dashboard Supplement Sales Subscription*



Gambar 3.34 Membuat Dashboard Supplement Sales

Pada gambar 3.34 , secara umum memberikan gambaran mengenai performa penjualan produk suplemen, dengan fokus utama pada total pendapatan (revenue) dan jumlah unit yang terjual. Visualisasi yang ditampilkan bertujuan untuk menyampaikan informasi bisnis secara ringkas, jelas, dan interaktif agar memudahkan proses analisis serta pengambilan keputusan. Berikut adalah komponen-komponen visual utama yang digunakan dalam dashboard ini:

A. Indikator Angka Besar (Big Number Indicators)
Di bagian atas dashboard ditampilkan dua indikator utama, yaitu:

a. Total Revenue sebesar 22,91 juta

b. Total Units Sold sebanyak 658 ribu unit
Visualisasi ini dipilih karena mampu menyampaikan ringkasan performa bisnis secara cepat dan langsung menarik perhatian pengguna terhadap pencapaian utama.

B. *BarChart*

Visualisasi ini menampilkan pendapatan dari masing-masing produk dalam bentuk grafik batang vertikal. Tujuannya adalah untuk memudahkan perbandingan antarproduk dan mengidentifikasi produk-produk dengan kontribusi pendapatan tertinggi secara visual dan intuitif.

C. *LineChart*

Grafik garis ini menampilkan tren pendapatan dari waktu ke waktu, serta dilengkapi dengan prediksi (forecasting) untuk periode mendatang. Visualisasi ini penting untuk memantau pertumbuhan penjualan dan membantu dalam proyeksi performa bisnis ke depan.

D. *DonutChart*

Di bagian kanan dashboard, ditampilkan donut chart yang menggambarkan distribusi pendapatan berdasarkan kategori produk, seperti Vitamin, Mineral, Protein, dan lain-lain. Donut chart dipilih karena efektif dalam menunjukkan proporsi kontribusi tiap kategori terhadap total pendapatan, dengan cara yang mudah dipahami secara visual.

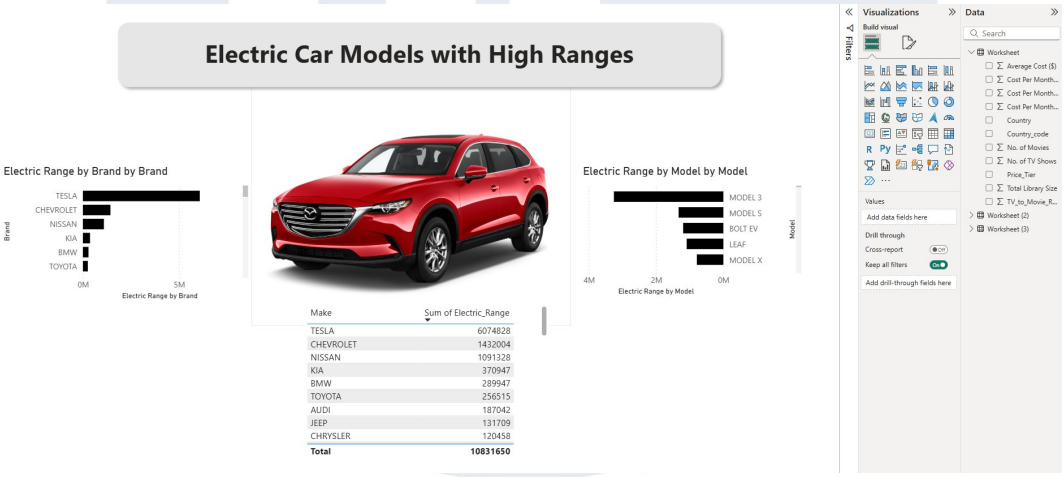
E. Filter Interaktif

Filter disediakan agar pengguna dapat melakukan eksplorasi data secara lebih mendalam. Pengguna bisa memilih produk tertentu atau tahun tertentu untuk memfokuskan analisis, misalnya melihat performa satu kategori produk di tahun tertentu.

Seluruh komponen visualisasi dalam dashboard ini dirancang untuk memberikan kombinasi antara informasi ringkasan, perbandingan performa

antarproduk, analisis tren waktu, dan distribusi pendapatan antar kategori. Penyusunan dashboard yang interaktif dan intuitif ini memungkinkan pengguna untuk memperoleh insight yang cepat dan relevan, serta mendukung pengambilan keputusan berbasis data dalam konteks bisnis penjualan suplemen.

3.2.8.3 *Dashboard Electric Vehicle Ranges*



Gambar 3.35 Membuat Dashboard Electric Vehicles

Pada gambar 3.35, menyajikan visualisasi mengenai model mobil listrik dengan jarak tempuh (electric range) yang tinggi. Fokus utama dari dashboard ini tercermin dalam judul "*Electric Car Models with High Ranges*", yang secara jelas menunjukkan topik analisis. Desain visual di bagian tengah dashboard juga menyertakan gambar mobil listrik sebagai elemen pendukung konteks, guna memperkuat keterkaitan antara visualisasi dan tema. Dashboard terdiri dari beberapa elemen visual utama yang saling melengkapi, yaitu:

A. *BarChart*

Visualisasi ini ditampilkan di sisi kiri dashboard dan menampilkan total jarak tempuh kendaraan listrik berdasarkan merek mobil. Tujuan utama grafik ini adalah untuk memfasilitasi perbandingan performa antarbrand. Dari visualisasi ini terlihat bahwa Tesla memiliki kontribusi jarak tempuh tertinggi secara signifikan dibanding merek lainnya.

Di sisi kanan dashboard ditampilkan grafik batang serupa yang menyoroti model-model kendaraan listrik dengan jarak tempuh terpanjang. Model seperti Model 3, Model S, dan Bolt EV menempati posisi teratas. Visualisasi ini sangat berguna untuk mengidentifikasi model-model mobil paling efisien dari segi daya jangkau.

B. *Data Table*

Di bagian bawah dashboard terdapat tabel yang merangkum total jarak tempuh per merek secara numerik. Tabel ini memperkuat informasi yang disajikan dalam grafik batang, serta menyediakan angka absolut yang dapat digunakan sebagai referensi yang lebih presisi dalam proses analisis atau perbandingan.

Pemilihan visualisasi berupa bar chart dan data tabel bertujuan untuk memberikan pemahaman yang cepat dan jelas mengenai performa berbagai merek dan model mobil listrik.

3.2.9 Presentasi Project

Gambar 3.36 Presentasi Final Project

Pada gambar 3.36 adalah tahap presentasi, intern menampilkan keseluruhan progres dari proyek yang telah dikerjakan selama masa magang. Presentasi berlangsung di hadapan mentor dan beberapa anggota tim dari divisi *BI and Analytics Platform Operation Management*. Sesi ini bertujuan untuk memberikan gambaran menyeluruh mengenai hasil kerja, mulai dari tahap perencanaan, pembangunan pipeline ETL menggunakan Apache *Airflow*, hingga pembuatan *dashboard* visualisasi data dengan *Power BI*.

Selama presentasi, penjelasan disampaikan secara general dan non-teknis agar seluruh peserta, termasuk pihak yang tidak memiliki latar belakang teknis, tetap dapat memahami alur kerja serta manfaat dari proyek yang dikembangkan. Komunikasi dilakukan dengan pendekatan yang *user-friendly*, dengan fokus utama pada penyampaian nilai bisnis dari data pipeline yang dibangun. Penjelasan mencakup bagaimana data mentah melalui proses otomatisasi: dibersihkan, ditransformasikan, dan dimuat ke dalam database sebelum divisualisasikan untuk kebutuhan analitik.

Untuk membantu pemahaman audiens, digunakan elemen visual seperti diagram alur, ilustrasi proses, dan tangkapan layar dashboard. Penjelasan juga mencakup beberapa tantangan teknis yang muncul selama proses pengembangan, seperti permasalahan pada XCom *serialization* dan upaya optimasi performa *query* dalam visualisasi data.

Melalui sesi ini, intern menerima berbagai masukan konstruktif dari mentor dan tim. Beberapa masukan tersebut mencakup peningkatan efisiensi *workflow*, konsistensi penamaan tabel, serta perbaikan desain visualisasi agar lebih berorientasi pada pengguna. Setelah sesi presentasi, intern segera melakukan penyempurnaan terhadap proyek yang telah dibangun. Perbaikan mencakup refactor struktur DAG pada *Airflow* dan penyesuaian desain *dashboard* agar lebih intuitif serta mudah digunakan. Revisi ini memastikan bahwa solusi yang dikembangkan tidak hanya stabil secara teknis, tetapi juga relevan dan layak diterapkan dalam konteks lingkungan kerja profesional.

3.2.10 Finalisasi Proyek dan Evaluasi Hasil

Setelah menyelesaikan seluruh tahapan pengembangan pipeline dan visualisasi, kegiatan magang berlanjut ke tahap finalisasi proyek dan evaluasi hasil kerja. Tahap ini berfokus pada penyempurnaan alur teknis yang telah dibangun, serta validasi output data dan visualisasi berdasarkan masukan yang diberikan selama proses bimbingan.

Finalisasi pipeline dilakukan dengan meninjau kembali struktur DAG pada Apache Airflow, memastikan setiap task berjalan sesuai urutan dan saling terhubung dengan benar. Intern memverifikasi bahwa proses extract, clean, transform, validate, dan load tidak menghasilkan error serta menyimpan data ke dalam database PostgreSQL dengan hasil yang bersih dan sesuai format. Selain itu,

pengaturan XCom untuk pertukaran data antar-task juga diperiksa kembali agar tidak terjadi konflik saat pipeline dijalankan.

Pada sisi visualisasi, dashboard yang dibangun menggunakan Power BI mengalami beberapa penyesuaian akhir berdasarkan masukan mentor. Penyesuaian tersebut mencakup perbaikan tampilan visual agar lebih mudah dibaca, penyederhanaan grafik, penambahan filter interaktif, dan pengelompokan data berdasarkan kategori yang lebih relevan. Tujuannya adalah memastikan informasi tersampaikan secara jelas dan dashboard dapat digunakan dengan mudah oleh pihak non-teknis.

Intern juga melakukan pengujian ulang terhadap alur pipeline dan hasil visualisasi untuk memastikan stabilitas sistem dan akurasi data. Semua elemen diperiksa untuk meminimalkan kesalahan teknis sebelum proyek dipresentasikan sebagai hasil akhir.

Secara keseluruhan, proyek yang telah diselesaikan berhasil menunjukkan bagaimana data pipeline dapat dijalankan secara otomatis dan efisien menggunakan Apache Airflow, serta bagaimana hasil pengolahan data tersebut divisualisasikan secara interaktif melalui Power BI. Evaluasi dari mentor menunjukkan bahwa proyek sudah mencerminkan alur kerja yang relevan dengan praktik profesional di tim Business Intelligence, serta memberikan kontribusi positif terhadap pemahaman alur teknis dan proses analisis data di Telkomsel.

3.3 Kendala yang Ditemukan

Selama menjalani proses magang di PT Telekomunikasi Selular (Telkomsel), intern menghadapi beberapa kendala baik dari sisi teknis maupun non-teknis. Beberapa kendala tersebut antara lain:

- Belum memiliki pengalaman dengan tools seperti *Apache Airflow, Docker,* dan *Apache NiFi*, sehingga memerlukan waktu lebih lama untuk memahami

alur kerja, struktur file, serta cara konfigurasi masing-masing *tool*. Proses pembelajaran menjadi lebih menantang karena belum pernah menggunakan tools tersebut sebelumnya dalam konteks nyata.

- Tidak memperoleh pelatihan formal dari pihak perusahaan, sehingga harus mencari referensi dan metode kerja secara mandiri. *Intern* memanfaatkan dokumentasi resmi dan sumber eksternal untuk menggali informasi teknis serta langkah-langkah implementasi.
- Menghadapi kesulitan saat menyelesaikan *error* atau kendala teknis, terutama ketika dokumentasi tidak secara langsung menjelaskan solusi. Proses troubleshooting menjadi lebih kompleks karena perlu memahami konteks error serta mencoba berbagai pendekatan penyelesaian secara mandiri.
- Menghadapi keterbatasan waktu untuk mendalami tools baru secara menyeluruh, karena tetap harus menyelesaikan tugas proyek dalam jangka waktu tertentu. Hal ini menuntut kemampuan mengatur waktu dan prioritas agar proses belajar dan penyelesaian tugas dapat berjalan seimbang.
- Selama menjalankan proyek menghadapi tidak disediakan data atau data *dummy* yang dari pihak perusahaan. Akibatnya, harus mencari dan menggunakan data open source dari internet, sehingga proyek yang dikerjakan tidak memiliki korelasi langsung dengan data operasional Telkomsel.

3.4 Solusi atas Kendala yang Ditemukan

Untuk mengatasi kendala-kendala tersebut, intern menerapkan beberapa solusi secara aktif dan terstruktur. Beberapa solusi yang dilakukan antara lain:

- Melatih kebiasaan melakukan pembelajaran mandiri (self-learning) dengan cara membaca dokumentasi resmi, mengikuti video tutorial, serta mencari jawaban di forum diskusi seperti *Stack Overflow, GitHub*, dan komunitas teknis lainnya. Dengan pendekatan ini, intern dapat mempercepat proses pemahaman terhadap tools baru yang digunakan.

- Mencatat seluruh langkah konfigurasi dan solusi dari permasalahan yang pernah ditemui, sehingga dapat membangun dokumentasi pribadi yang berguna sebagai referensi ketika menghadapi kendala serupa. Langkah ini juga memudahkan proses pelacakan kesalahan dan mempercepat *debugging*.
- Mengembangkan kemampuan troubleshooting dengan membaca dan menganalisis pesan error secara cermat, kemudian mencoba berbagai pendekatan penyelesaian melalui metode *trial and error*. Pendekatan ini membantu meningkatkan kemampuan berpikir logis dan analitis dalam menyelesaikan masalah teknis.
- Mengelola waktu secara efektif dan menyusun prioritas kerja berdasarkan urgensi dan kompleksitas tugas, agar proses belajar *tools* baru tetap berjalan tanpa mengganggu penyelesaian proyek utama. *Intern* menggunakan jadwal harian untuk membagi waktu antara eksplorasi tools dan pengerjaan tugas, sehingga semua target dapat tercapai dengan *optimal*.