

PROPOSAL FINAL PROJECT
DATA & AI ANALYTICS GRADXPRT BITHEALTH
STUDY CASE DISTRIBUTION



Disusun Oleh:

Maulana Muhamad Priadhi	(BTP23120610)
Mangara Haposan Immanuel S.	(BTP24010818)
Santriana Pratama	(BTP24010821)
Yohana Tambunan	(BTP24010823)
Taufiqurrahman	(BTP24010824)

Green Office Park 9, 3rd Floor, Jl. BSD Green Office Park Jl. BSD Grand Boulevard, Sampora,
Kec. Cisauk, Kabupaten Tangerang, Banten 15345

DAFTAR ISI

DAFTAR ISI.....	2
BAB I PENDAHULUAN.....	4
I.1. Latar Belakang.....	4
I.2. Rumusan Masalah.....	5
I.3. Tujuan.....	5
BAB II METODOLOGI PENELITIAN.....	7
II.1. Dataset.....	7
II.2. Diagram Alir Project.....	10
II.3. Load Raw Data.....	11
II.4. Data Transformation.....	16
II.5. Data Normalization.....	29
II.6. Setting up Airflow Pipeline.....	58
II.7. Label Encoding.....	68
II.8. Correlation Analysis.....	68
II.9. Variance Inflation Factor (VIF).....	69
II.10. Random Forest Regressor.....	70
II.11. Cross Validation.....	70
II.12. MAE.....	71
II.13. RMSE.....	71
II.14. R2-score.....	71
II.15. Hyperparameter Tuning.....	72
II.16. Data Visualization.....	72
BAB III HASIL DAN PEMBAHASAN.....	77
III.1. Hasil ETL Pipeline Airflow.....	77
1.1. Tabel Admission.....	77
1.2. Tabel Stock_Obat.....	77
1.3. Tabel Reviews.....	77
1.4. Tabel Payment.....	78
1.5. Tabel Doctor.....	78
1.6. Tabel Lab.....	78
1.7. Tabel Surgery.....	78
1.8. Tabel Branch.....	79
1.9. Tabel Hospital Care.....	79
1.10. Tabel Patient.....	79
1.11. Tabel Room Type.....	80
1.12. Tabel Drugs.....	80
1.13. Tabel Drugs Type.....	80
1.14. Tabel Logs.....	81

III.2 Analisis Dashboard.....	81
III.3. Analisis Script Python (Machine Learning).....	86
3.1. Import Library.....	86
3.2. Data Loading.....	87
3.3. Feature Engineering.....	87
3.3.1 Data preprocessing.....	87
3.3.2 Encoding.....	89
3.3.3 Scaling.....	89
3.3.4 Feature Selection.....	92
3.3.5 Data Splitting.....	94
3.4. Model Training.....	94
3.5. Model Evaluation.....	95
3.6. Model Tuning.....	98
3.7. Model Saving.....	100
BAB IV KESIMPULAN.....	101
BAB V DAFTAR PUSTAKA.....	103
LAMPIRAN.....	104

BAB I

PENDAHULUAN

I.1. Latar Belakang

Transformasi digital telah berperan penting dalam berbagai industri, tidak terkecuali industri kesehatan. Banyak rumah sakit berupaya melakukan transformasi digital dengan optimalisasi teknologi, dengan harapan data-data yang ada di rumah sakit dapat terekam dengan baik sekaligus meningkatkan layanan yang berkualitas bagi pasien. Data-data digital di rumah sakit didapatkan dari perangkat yang terkoneksi satu sama lain antar bagian dalam sebuah rumah sakit. Kumpulan data tersebut diperoleh dari perjalanan paling awal seorang pasien di rumah sakit, dimulai dari administrasi, hasil pemeriksaan, monitoring dalam rawat inap maupun rawat jalan, hingga ke proses paling akhir yaitu pembayaran dan pasien kembali pulih.

Pemanfaatan data atau biasa disebut big data memiliki potensi besar dalam bisnis, di mana semua data yang ada terhubung satu sama lain dan dapat memberikan insight serta manfaat seperti health tracking, menganalisa dan memprediksi, menyediakan layanan yang lebih sesuai dengan setiap pasien, diagnosa yang lebih efektif, kemajuan dalam telemedicine, serta masih banyak manfaat lainnya. Big data di rumah sakit terdiri dari koordinasi pelayanan kesehatan, pemberdayaan pasien, obat yang disesuaikan dengan kebutuhan pasien, dan reformasi pembayaran. Implementasi pemanfaatan big data ini menghasilkan kemudahan baik bagi operasional rumah sakit maupun pengalaman pasien dalam rumah sakit.

Data kesehatan sangat dicari seperti ladang emas yang baru, di mana data-data yang ada dan telah diolah dengan baik (data ingestion, data cleansing, data mining, reporting & visualization dari data mentah) dapat menjadi informasi yang digunakan oleh pengguna informasi dalam pengambilan keputusan melalui proses digitalisasi data. Hal ini terwujud dengan adanya data warehouse, yaitu jenis sistem manajemen data yang dirancang untuk mengaktifkan dan mendukung aktivitas business intelligence (BI), terutama analitik. Manfaat yang didapatkan dalam menggunakan data warehouse adalah integrasi seluruh data yang ada dan juga melakukan queries serta analisis yang seringkali berisi data historis dalam jumlah besar. Selain itu, pemusatan data ini dapat memberikan akses ke berbagai departemen di sebuah rumah sakit dengan mudah sehingga rumah sakit memiliki satu sumber terpercaya dan dapat memaksimalkan value dari data tersebut.

Digitalisasi ini mendukung tercapainya efisiensi pelayanan yang tidak perlu berulang-ulang dan meminimalisir risiko terhadap pasien. Pada akhirnya, jika diterapkan prinsip dasar teknologi kesehatan pada setiap fasilitas kesehatan seperti rumah sakit, hal ini akan bermanfaat tidak hanya bagi masyarakat yang menerima layanan, tetapi juga bagi bagian dari rumah sakit itu sendiri yang memberikan pelayanan maupun regulasi secara umum. Pengolahan data yang baik berguna untuk memaksimalkan sistem analytics dan maturity suatu model bisnis yang dimiliki, dalam hal ini yaitu rumah sakit. Contoh penerapan analytics di rumah sakit adalah menambahkan akses pasien dengan adanya telemedicine, memperbaiki di level prosedur dan diagnostik serta layanan kritis di rumah sakit, sampai melakukan prediksi baik pasien masuk ataupun keluar, dan hal lain yang mempengaruhi sistem keputusan. Melalui hal ini diharapkan dapat tercapainya sistem informasi digital yang cerdas dan terpadu (Smart Hospital).

Apabila data yang ada sudah siap serta memiliki dashboard yang baik, maka suatu rumah sakit akan siap memasuki era transformasi digital dan penerapan teknologi dengan baik. Dalam pelayanan kesehatan, terdapat beberapa cabang dari sebuah rumah sakit. Setiap cabang rumah sakit memiliki tantangan sendiri dalam menyediakan layanan kesehatan terbaik bagi masyarakat. Setiap cabang memiliki inventory stok obat dan permintaan dokter untuk spesialis penyakit tertentu. Sering terjadi permintaan dokter untuk spesialis penyakit tertentu yang tidak seimbang, hal ini tentunya berpengaruh pada jumlah kebutuhan obat yang dikonsumsi pasien dan stok obat di setiap cabang rumah sakit yang berbeda. Oleh karena itu, perlu dilakukan optimalisasi pendistribusian obat dan dokter serta memprediksi obat bulan berikutnya untuk menghindari kelebihan dan kekurangan persediaan obat di masing-masing cabang rumah sakit.

I.2. Rumusan Masalah

Bagaimana cara mengoptimalkan pendistribusian obat dan dokter spesialis di beberapa cabang rumah sakit, untuk memastikan keseimbangan antara permintaan dan ketersediaan obat serta kebutuhan dokter spesialis penyakit tertentu setiap bulan, guna menghindari kelebihan dan kekurangan persediaan obat di masing-masing cabang rumah sakit?

I.3. Tujuan

1. Mengetahui persediaan obat di masing masing cabang Rumah Sakit Mulia

2. Mengetahui selisih antara ketersediaan dan kebutuhan obat di cabang rumah sakit RSMA, RSMD, RSM
3. Mengetahui jumlah kebutuhan dokter di setiap rumah sakit
4. Mengetahui implementasi *Machine Learning* untuk memprediksi kuantitas obat di cabang rumah sakit RSMA, RSMD, RSMS pada bulan acuan.

BAB II METODOLOGI PENELITIAN

II.1. Dataset

Pada project digunakan dua buah dataset yaitu *Hospital Data* dan *Drug Data*. Kedua dataset masing - masing memiliki keunikan dan karakteristik yang berbeda. Rincian untuk masing - masing dataset dijelaskan sebagai berikut.

1. Hospital Data

Dataset ini berisi 9474 admission. Deskripsi untuk dataset hospital data dijelaskan pada Tabel 2.1.

Tabel 2.1. Deskripsi dataset hospital data

Column	Description
ID	ID penjualan (Invoice)
Date IN	Tanggal pasien masuk rumah sakit
Date OUT	Tanggal pasien keluar rumah sakit
Branch	Cabang rumah sakit:
	- RSMA : Rumah Sakit Mulia Anggrek
	- RSMD : Rumah Sakit Mulia Duri
	- RSMS : Rumah Sakit Mulia Simatupang
Name	Nama pasien
Age	Usia pasien
Gender	Gender pasien
Hospital Care	Tipe perawatan pasien
Room	Tipe kamar:

	- VIP : Rp. 300.000/malam
	- Kelas 1 : Rp. 250.000/malam
	- Kelas 2 : Rp. 200.000/malam
	- Kelas 3 : Rp. 150.000/malam
Doctor	Tipe dokter:
	- Bedah : Rp. 300.000/kunjungan
	- Gigi : Rp. 300.000/kunjungan
	- Kandungan : Rp. 300.000/kunjungan
	- Penyakit dalam : Rp. 300.000/kunjungan
	- Umum : Rp. 200.000/kunjungan
Surgery	Tipe tindakan/operasi:
	- Kecil : Rp. 4.000.000
	- Besar : Rp. 8.000.000
	- Kusus : Rp. 15.000.000
Lab	Tipe uji laboratorium:
	- Hematologi : Rp. 90.000
	- Kimia Darah : Rp. 195.000
	- Rontgen : Rp. 150.000
	- Serologi : Rp. 200.000
	- Urinalisa : Rp. 80.000

Drug Types	Tipe obat:
	- Antibiotik : Rp. 75.000
	- Pereda nyeri : Rp. 50.000
	- Umum : Rp. 40.000
	- Vitamin : Rp. 110.000
Drug Brands	Merek-merek obat berdasarkan tipenya
Drug Quantity	Jumlah strip/botol yang perlu dikonsumsi oleh pasien selama masa penyembuhan
Food	Harga makanan untuk pasien rawat inap setiap hari (harga sudah termasuk sarapan, makan siang, dan malam)
Admin	Biaya administrasi rumah sakit (fix price/pasien)
COGS	Cost of Good Sold Rumah sakit untuk setiap pasien
Payment	Tipe pembayaran pasien
Review	Review dari pasien

2. Drug Data

Dataset ini berisi 575 admission. Deskripsi untuk dataset hospital data dijelaskan pada Tabel 2.2.

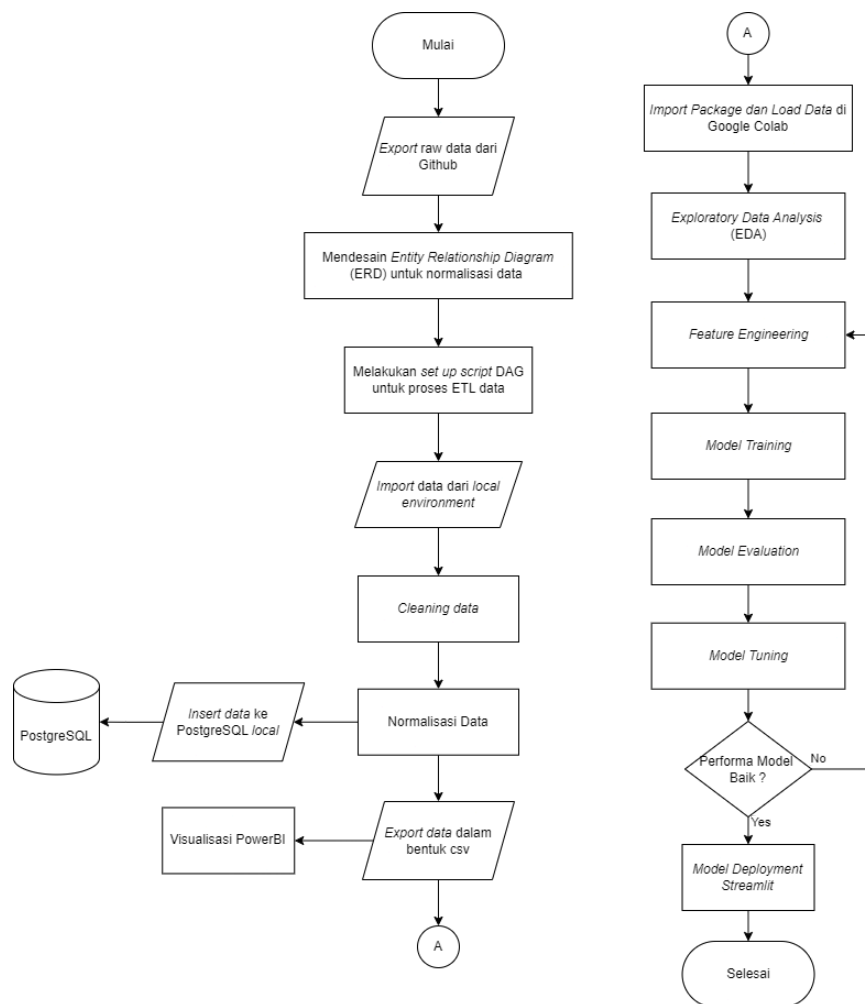
Tabel 2.2. Deskripsi dataset drug data

Column	Description
Date	Tanggal obat masuk rumah sakit
Drugs	Merek-merek obat berdasarkan tipenya
Qty	Jumlah strip/botol yang masuk rumah sakit

Branch	Cabang rumah sakit:
	- RSMA : Rumah Sakit Mulia Anggrek
	- RSMD : Rumah Sakit Mulia Duri
	- RSMS : Rumah Sakit Mulia Simatupang

II.2. Diagram Alir Project

Desain dari diagram alir atau *flowchart* mencakup seluruh proses utama termasuk ekstraksi data, transformasi data, hingga data *diconsume* untuk keperluan *machine learning modelling* dan pembentukan visualisasi untuk *analytics*. Diagram alir *project* digambarkan oleh Gambar 2.1.



Gambar 2.1. Diagram alir *project*

II.3. Load Raw Data

Proses *load* data dilakukan secara otomatis dengan menggunakan *scheduler airflow*. Proses dimulai dengan melakukan setup koneksi ke *local database*, pada kasus ini digunakan postgresql sebagai *relational database management system* (RDBMS) sebagai berikut.

a. Setup Package dan Koneksi ke PostgreSQL

```
from airflow import DAG
from datetime import timedelta, datetime
import psycopg2 as db
import pandas as pd
import os
import csv
from airflow.operators.python_operator import PythonOperator

# setup configuration to postgresql database
conn_string = "dbname='final_project' host='host.docker.internal'
user='postgres' password='2023'"
```

b. Kemudian dilanjutkan dengan membuat tabel utama, yaitu tabel Admission dan Stock_Obat.

Tabel Admission

```
# function to create table admission
def create_table_admission():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to create table admission
        sql_adm="""
            create table if not exists admission (
                id int,
                date_in date,
                date_out date,
                branch varchar(10),
                fullname varchar(100),
                age varchar(50),
```

```

        gender varchar(50),
        hospital_care varchar(50),
        room varchar(10),
        doctor varchar(50),
        surgery varchar(10),
        lab varchar(50),
        drug_types varchar(50),
        drug_brands varchar(50),
        drug_qty varchar(10),
        food varchar(50),
        admin_cost varchar(50),
        cogs varchar(50),
        payment varchar(50),
        review varchar(50),
        call_date timestamp,
        primary key(id)
    );
"""

# execute the sql command
cur.execute(sql_adm)

# Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message

# commit the task to database
conn.commit()

# close the connection to database
conn.close()

```

Tabel Stock_Obat

```

# function to create table stock obat
def create_table_stock_obat():
    # connect to database using psycopg2.connect

```

```

conn = db.connect(conn_string)
# create cur object to execute SQL commands
cur = conn.cursor()
try:
    # sql command to create table stock obat
    sql_stock="""
        create table if not exists drug_stock (
            id serial primary key,
            date_buy date not null,
            drugs varchar(50) not null,
            quantity varchar(50) not null,
            branch varchar(10) not null,
            call_date timestamp
        );
    """
    # execute the sql command
    cur.execute(sql_stock)

# Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message
# commit the task to database
conn.commit()
# close the connection to database
conn.close()

```

- c. Selanjutnya dilakukan proses *load data* dari *local memory* (csv) ke RDBMS dengan menggunakan *syntax insert into* ke dalam tabel Admission dan Stock_Obat.

Load data ke tabel Admission

```

# function to insert data to table admission
def insert_data_admission():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)

```

```

# create cur object to execute SQL commands
cur = conn.cursor()
try:
    # path of csv file
    csv_file_adm = os.path.join(os.path.dirname(__file__),
'Hospital_data.csv')
    # read the csv file into dataframe
    df_adm = pd.read_csv(csv_file_adm, delimiter=';')
    # iterate over each row in dataframe
    for index, row in df_adm.iterrows():
        # get current timestamp
        call_date = datetime.now()
        # sql command to insert data into admission table
        insert_sql_adm = "INSERT INTO admission (id, date_in,
date_out, branch, fullname, age, gender, hospital_care, room,
doctor, surgery, lab, drug_types, drug_brands, drug_qty, food,
admin_cost, cogs, payment, review, call_date) VALUES (%s, %s, %s,
%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
%s);"

        # define the value to be inserted into the table
        values_adm = (row['ID'], row['Date IN'], row['Date
OUT'], row['Branch'], row['Name'], row['Age'], row['Gender'],
row['Hospital Care'], row['Room'], row['Doctor'], row['Surgery'],
row['Lab'], row['Drug Types'], row['Drug Brands'], row['Drug Qty'],
row['Food'], row['Admin'], row['COGS'], row['Payment'],
row['Review'], call_date)

        # execute the sql command with its value
        cur.execute(insert_sql_adm, values_adm)
    # Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message
# commit the task to database
conn.commit()

```

```
# close the connection to database
conn.close()
```

Load data ke tabel Stock_Obat

```
# function to insert data to table stock obat
def insert_data_stock_obat():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # path of csv file
        csv_file_stock = os.path.join(os.path.dirname(__file__),
        'Drugs_data1.csv')
        # read the csv file into dataframe
        df_stock = pd.read_csv(csv_file_stock, delimiter=';')
        # iterate over each row in dataframe
        for index, row in df_stock.iterrows():
            # get current timestamp
            call_date = datetime.now()
            # sql command to insert data into drug_stock table
            insert_sql_stock = "INSERT INTO drug_stock (date_buy,
            drugs, quantity, branch, call_date) VALUES (%s, %s, %s, %s, %s);"
            # define the value to be inserted into the table
            values_stock = (row['Date'], row['Drugs'], row['Qty'],
            row['Branch'], call_date)
            # execute the sql command with its value
            cur.execute(insert_sql_stock, values_stock)
            # Log the error to the logs table
    except Exception as e:
        # convert any error message and save it in str format
        error_message = str(e)
        # return error message
        return error_message
    # commit the task to database
    conn.commit()
```

```
# close the connection to database
conn.close()
```

II.4. Data Transformation

Proses transformasi data terdapat pada satu *DAG code* yang sama dengan proses *load* dan normalisasi data yang terdiri atas beberapa tahapan transformasi sebagai berikut.

a. Cleaning Data

Fungsi “data_cleaning” memiliki beberapa *task* yang tergabung dalam satu fungsi ini yang dapat dijabarkan sebagai berikut:

1. Dimulai dengan proses pengambilan data dari tabel di database dengan perintah SQL `SELECT * FROM admission;` dieksekusi menggunakan metode `execute` dari objek cursor `cur`, yang akan mengembalikan semua baris dari tabel `admission`. Setelah perintah dieksekusi, nama-nama kolom dari hasil query diperoleh melalui atribut `description` dari objek cursor, yang berisi metadata tentang kolom-kolom dalam hasil query. Nama-nama kolom ini diekstraksi menggunakan list comprehension `[desc[0] for desc in cur.description]`, di mana `desc[0]` adalah nama kolom. Selanjutnya, semua baris hasil query diambil menggunakan metode `fetchall()` dari objek cursor, dan data tersebut kemudian digunakan untuk membuat sebuah `DataFrame` `pandas`. `DataFrame` ini dibuat dengan memanfaatkan nama-nama kolom yang telah diperoleh sebelumnya sebagai label kolom, sehingga memungkinkan manipulasi dan analisis data yang lebih mudah di dalam `pandas`.

```
# function to clean the data
def data_cleaning():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # Execute a query to select all data from the 'admission'
        table
        cur.execute("SELECT * FROM admission;")
        # Get the column names from the query result
        columns = [desc[0] for desc in cur.description]
        # Fetch all rows and create a pandas DataFrame with the data
```



```
df = pd.DataFrame(cur.fetchall(), columns=columns)
```

2. Memeriksa apakah terdapat nilai yang hilang dalam DataFrame `df` menggunakan metode `isnull().values.any()`, yang mengembalikan `True` jika ada setidaknya satu nilai yang hilang. Jika ada nilai yang hilang, kode tersebut menggunakan metode `fillna()` untuk mengganti semua nilai yang hilang dengan tanda '-' dan mengatur parameter `inplace=True` untuk memastikan perubahan dilakukan langsung pada Data Frame asli tanpa membuat salinan baru. Jika tidak ditemukan nilai yang hilang, program akan mencetak pesan "Data doesnt have missing value" untuk memberitahu bahwa data tidak memiliki nilai yang hilang. Setelah itu, terlepas dari apakah ada nilai yang hilang atau tidak, kode menggunakan metode `replace()` untuk mengganti semua kemunculan tanda '-' dalam DataFrame dengan '0', sekali lagi dengan `inplace=True` untuk melakukan perubahan langsung pada DataFrame asli. Proses ini memastikan bahwa semua nilai yang hilang diganti dengan '0', baik nilai tersebut awalnya *Null* maupun telah diisi dengan tanda '-'.

```
# Check missing value
if df.isnull().values.any():
    # fill missing values with '-'
    df.fillna(value='-', inplace=True)
else:
    # print message if there are no missing value
    print("Data doesnt have missing value")

# replace all '-' with 0
df.replace('-', '0', inplace=True)
```

3. Selanjutnya mengganti value "Kusus" menjadi bentuk kata bakunya yaitu "Khusus" pada kolom *surgery* dengan menggunakan perintah `replace()`.

```
# replace "kusus" with "khusus"
df['surgery'] = df['surgery'].replace('Kusus', 'Khusus')
```

4. Selanjutnya dilakukan *cleaning data* pada tiga kolom dalam DataFrame `df`, yaitu 'food', 'admin_cost', dan 'cogs'. Setiap kolom tersebut awalnya berisi nilai-nilai yang termasuk simbol mata uang 'Rp.' dan tanda koma ',' yang umum digunakan dalam format angka di

beberapa negara, termasuk Indonesia. Untuk setiap kolom, metode `str.replace('Rp.', '')` digunakan untuk menghapus semua kemunculan simbol 'Rp.' dari nilai-nilai dalam kolom tersebut. Setelah itu, metode `str.replace(',', '')` digunakan untuk menghapus semua tanda koma ',', sehingga nilai-nilai dalam kolom tersebut menjadi angka murni tanpa simbol mata uang atau tanda pemisah ribuan. Dengan demikian, kolom-kolom tersebut sekarang berisi nilai-nilai numerik yang dapat lebih mudah diproses lebih lanjut atau dianalisis.

```
# clean the unnecessary from currency value in food, admin_cost,
and cogs column

df['food'] = df['food'].str.replace('Rp.',
''.str.replace(',', ''))
df['admin_cost'] = df['admin_cost'].str.replace('Rp.',
''.str.replace(',', ''))
df['cogs'] = df['cogs'].str.replace('Rp.',
''.str.replace(',', ''))
```

5. Mengganti *value* “laki - laki” dengan “pria” dan *value* “perempuan” dengan “wanita” dengan menggunakan fungsi *lambda* dimana setiap *value* akan diubah menjadi bentuk *lower case* dengan perintah *lower()* kemudian setiap *value* akan digantikan sesuai dengan *value* yang sudah didefinisikan.

```
# replace 'pria' with 'laki - laki' in the 'Gender' column
df['gender'] = df['gender'].apply(lambda x: 'laki - laki' if
x.lower() == 'pria' else x)

# replace 'wanita' with 'perempuan' in the 'Gender' column
df['gender'] = df['gender'].apply(lambda x: 'perempuan' if
x.lower() == 'wanita' else x)
```

6. Melakukan *sorting data* berdasarkan tanggal kedatangan pasien dengan menggunakan perintah *sort_values()*.

```
# sort date-in ascending
df = df.sort_values('date_in')
```

7. Membuat *empty list* untuk menghimpun nama kolom yang nantinya akan digunakan untuk meng-*update* setiap *value* pada kolom di DataFrame. Selanjutnya dilakukan pengecekan

untuk nilai duplikat dengan perintah *duplicated()* yang mengembalikan True untuk setiap baris yang duplikat dan jika terdapat baris yang duplikat, baris pertama dari setiap duplikat dihapus menggunakan perintah *drop_duplicates()* dengan parameter *inplace=True* untuk melakukan perubahan langsung pada DataFrame. Setelah DataFrame df dibersihkan dari duplikat, kita perlu memastikan bahwa tabel admission di database juga mencerminkan perubahan yang sama. Untuk ini, kita melakukan penghapusan baris duplikat langsung di dalam tabel database menggunakan query SQL DELETE. Dengan begitu, dibentuk kode dengan klausa WHERE menggunakan kolom - kolom yang telah didefinisikan dalam *columns_to_update*. Klausa WHERE ini digunakan untuk menentukan kondisi unik dari setiap baris yang akan dihapus. Query SQL DELETE kemudian dibentuk untuk menghapus baris-baris duplikat dari tabel admission, menyisakan hanya satu baris unik untuk setiap set kolom yang telah didefinisikan. Query DELETE ini dieksekusi menggunakan objek cursor *cur*.

```
# define the set of columns to update
columns_to_update = ['date_in', 'date_out', 'branch',
                    'fullname', 'age', 'gender',
                    'hospital_care', 'room', 'doctor',
                    'surgery', 'lab', 'drug_types',
                    'drug_brands', 'drug_qty', 'food',
                    'admin_cost', 'cogs', 'payment',
                    'review']

# check duplicate value
duplicate_rows = df.duplicated()

# Drop the first duplicate row if any
if duplicate_rows.any():
    df.drop_duplicates(inplace=True)

# Construct the set of columns for the WHERE clause
where_clause = " AND ".join([f"{column} = %s" for column
                              in columns_to_update])
```

```

                                where_values = [row[column] for column in
columns_to_update]

```

```

# Construct the DELETE query
delete_sql_adm = f"""
    DELETE FROM admission
    WHERE id NOT IN (
        SELECT MIN(id)
        FROM admission
        GROUP BY {'', '}.join(columns_to_update)}
    );
"""
cur.execute(delete_sql_adm)

```

8. Setelah menghapus baris-baris duplikat, kode melanjutkan untuk memperbaiki setiap baris dalam DataFrame df. Untuk setiap baris, klausa SET dibentuk menggunakan kolom-kolom yang telah didefinisikan dalam columns_to_update, yang menentukan nilai-nilai baru yang akan diupdate dalam tabel admission. Query SQL UPDATE kemudian dibentuk untuk memperbaiki baris-baris dalam tabel admission berdasarkan nilai-nilai dalam DataFrame. Nilai-nilai dari setiap baris, termasuk ID baris, diambil dan digunakan sebagai parameter untuk query UPDATE. Query ini kemudian dieksekusi untuk setiap baris dalam DataFrame menggunakan objek cursor cur, memastikan bahwa tabel admission diperbarui sesuai dengan data yang telah dibersihkan dan disiapkan dalam DataFrame df.

```

for index, row in df.iterrows():
    # Construct the SET clause using the columns_to_update
set
    set_clause = ", ".join([f"{column} = %s" for column in
columns_to_update])
    # Construct the update query
update_sql_adm = f"""
        UPDATE admission
        SET {set_clause}
        WHERE id = %s;
    """

```

```

# Get the values from the row, including the id
values = [row[column] for column in columns_to_update]
values.append(row['id'])

# Execute the update query
cur.execute(update_sql_adm, values)

```

9. Terakhir, fungsi ini memiliki *logic* untuk mengetahui kondisi *error* dan setiap *error* yang terjadi pada fungsi ini akan disimpan dalam *variable error_message*.

```

# log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message

# commit the task to database
conn.commit()

# close the connection to database
conn.close()

```

b. Convert Data Type

Selanjutnya, ketika data berhasil dilakukan *cleaning* maka dilakukan pengonversian tipe data dengan menggunakan fungsi *convert_datatype()*. Fungsi ini mendefinisikan daftar kolom yang akan diubah tipe datanya menjadi *integer* dalam *variable columns_to_convert*, yang mencakup kolom 'age', 'drug_qty', 'food', 'admin_cost', dan 'cogs'. Setiap kolom dalam *columns_to_convert*, fungsi ini menjalankan query SQL ALTER TABLE untuk mengubah tipe data kolom tersebut menjadi *integer*. Query ALTER TABLE menggunakan sintaks ALTER COLUMN {column} TYPE int USING {column}::integer, yang menginstruksikan PostgreSQL untuk mengubah tipe data kolom menjadi integer dengan menggunakan operator ::integer untuk melakukan konversi eksplisit.

```

# function to convert data type
def convert_datatype():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands

```

```

cur = conn.cursor()
try:
    # define the columns to convert
    columns_to_convert = ['age', 'drug_qty', 'food',
'admin_cost', 'cogs']

    # convert columns to integers
    for column in columns_to_convert:
        cur.execute(f"ALTER TABLE admission ALTER COLUMN {column}
TYPE int USING {column}::integer")

```

Terakhir, fungsi ini memiliki *logic* untuk mengetahui kondisi *error* dan setiap *error* yang terjadi pada fungsi ini akan disimpan dalam *variable error_message*.

```

# log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message

# commit the task to database
conn.commit()

# close the connection to database
conn.close()

```

c. Delete Column

Proses penghapusan kolom dilakukan setelah proses normalisasi data. Hal ini bertujuan untuk menghapus kolom - kolom pada tabel Admission dan tabel Stock_Obat yang tidak dibutuhkan kembali. Proses ini dilakukan dengan menggunakan *code* sebagai berikut.

```

# function to delete column
def delete_columns():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:

```

```

        # Delete the age, gender, drug_types, and food columns
        cur.execute("ALTER TABLE admission DROP COLUMN age, DROP
COLUMN gender, DROP COLUMN drug_types, DROP COLUMN food;")

# Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message

# commit the task to database
conn.commit()

# close the connection to database
conn.close()

```

Fungsi *delete_columns()* memanfaatkan *alter table admission DROP COLUMN age, DROP COLUMN gender, DROP COLUMN drug_types, DROP COLUMN food;* untuk menghapus kolom - kolom yang sudah digantikan dengan *foreign key* dari masing - masing tabel. Selanjutnya apabila proses penghapusan kolom mengalami *error* maka akan disimpan dalam variabel untuk disimpan dalam tabel *logs*.

a. Exporting Normalized Table

Selanjutnya data yang berhasil ditransformasi dan dilakukan *normalization* dilakukan *export* menjadi format *Comma-Separated Value* atau CSV. Fungsi *export_tables_to_csv()* dimulai dengan mendefinisikan daftar tabel yang akan diekspor dalam variabel *tables* yang mencakup tabel *admission, drug_stock, doctor, lab, drugs, branch, surgery, patient, room_type, dan hosp_care*. Kemudian *output directory* untuk menyimpan file ditentukan juga dengan menggunakan syntax *os.path.dirname(os.path.abspath(__file__))*.

```

# function to export normalized table into csv files
def export_tables_to_csv():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()

```

```

try :
    # define what table want to export
    tables = ['admission', 'drug_stock', 'doctor', 'lab',
              'drugs', 'branch', 'surgery', 'patient', 'room_type', 'hosp_care']
    # Set the output directory to be the same as the DAG file
    directory
    output_dir = os.path.dirname(os.path.abspath(__file__))

```

Selanjutnya, fungsi ini melakukan iterasi melalui setiap tabel dalam daftar tables. Untuk setiap tabel, fungsi ini menjalankan query *SQL SELECT * FROM {table};* untuk memilih semua data dari tabel tersebut, dan hasil query disimpan dalam variabel rows.

```

# loop through each table
for table in tables:
    # execute a query to select all data from each table
    cur.execute(f"SELECT * FROM {table};")
    # fetch all rows in every table
    rows = cur.fetchall()

```

Kemudian, fungsi ini membuka file CSV baru untuk menulis data, dengan memastikan bahwa file tersebut dipisahkan dengan *newline*. Ini dilakukan menggunakan pernyataan *with open(os.path.join(output_dir, f'{table}.csv'), 'w', newline='') as csvfile:* yang secara otomatis menangani penutupan file setelah operasi penulisan selesai. Objek penulis CSV (*csvwriter*) dibuat menggunakan *csv.writer(csvfile)*. Dan terakhir, fungsi ini menulis baris header ke file CSV menggunakan nama kolom dari deskripsi cursor (*cur.description*). Nama-nama kolom diambil dengan *csvwriter.writerow([desc[0] for desc in cur.description])*. Akhirnya, fungsi ini menulis baris-baris data ke file CSV menggunakan variabel *rows*, yang berisi data hasil query, dengan memanggil *csvwriter.writerows(rows)*.

```

# open a new CSV file for writing, specifying the file path and
ensuring it is newline-separated
with open(os.path.join(output_dir, f'{table}.csv'), 'w',
newline='') as csvfile:
    # create a csv writer object
    csvwriter = csv.writer(csvfile)
    # Write the header row to the CSV file, using the
column names from the cursor description

```



```

        csvwriter.writerow([desc[0] for desc in
cur.description])

        # Write the data rows to the CSV file, using the
'rows' variable which contains the data
        csvwriter.writerows(rows)

# log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message

# commit the task to database
conn.commit()

# close the connection to database
conn.close()

```

b. Exporting Merged Table

Selain meng-*export* tabel yang sudah dinormalisasi, dilakukan juga untuk table yang sudah *dimerge* untuk keperluan analisis tim Data Scientist dengan menggunakan fungsi *export_merged_table_to_csv()*. Fungsi ini bertujuan untuk menggabungkan data dari beberapa tabel dalam database dan mengekspor hasilnya ke dalam file CSV. Dimulai dengan menentukan direktori output di mana file CSV akan disimpan, yaitu di direktori yang sama dengan file DAG. Kemudian menentukan query SQL *sql_merged* yang menggabungkan data dari beberapa tabel (*admission*, *branch*, *patient*, *hosp_care*, *room_type*, *doctor*, *surgery*, *lab*, dan *drugs*) menggunakan *LEFT JOIN* dan menjalankan query tersebut dan mengambil semua hasilnya menggunakan *cur.fetchall()*. Dan terakhir, fungsi ini menulis baris header ke file CSV menggunakan nama kolom dari deskripsi cursor (*cur.description*). Nama-nama kolom diambil dengan *csvwriter.writerow([desc[0] for desc in cur.description])*. Akhirnya, fungsi ini menulis baris-baris data ke file CSV menggunakan variabel *rows*, yang berisi data hasil query, dengan memanggil *csvwriter.writerows(rows)*.

```

# function to export merged table into csv files
def export_merged_table_to_csv():
    # connect to database using psycopg2.connect

```

```

conn = db.connect(conn_string)
# create cur object to execute SQL commands
cur = conn.cursor()
try:
    # Set the output directory to be the same as the DAG file
directory
    output_dir = os.path.dirname(os.path.abspath(__file__))
    # execute a query to select all data from each table
    # sql command to export the merged table
    sql_merged = """
        SELECT
            a.id,
            a.date_in,
            a.date_out,
            b.branch_name AS branch,
            p.patient_name AS patient_name,
            p.age AS age,
            p.gender AS gender,
            hc.hospital_care AS hospital_care,
            r.room_type AS room_type,
            d.name AS doctor,
            s.surgery_type AS surgery_type,
            l.lab_name AS lab_name,
            dr.drug_type AS drug_type,
            dr.drug_name AS drug_brands,
            a.drug_qty,
            r.food_price AS food,
            a.admin_cost,
            a.cogs,
            a.payment,
            a.review
        FROM
            admission a
        LEFT JOIN
            branch b ON a.id_branch::integer = b.id
        LEFT JOIN

```

```

        patient p ON a.id_patient::integer = p.id
LEFT JOIN
        hosp_care hc ON a.id_hospital_care::integer =
hc.id

LEFT JOIN
        room_type r ON a.id_room::integer = r.id
LEFT JOIN
        doctor d ON a.id_doctor::integer = d.id
LEFT JOIN
        surgery s ON a.id_surgery::integer = s.id
LEFT JOIN
        lab l ON a.id_lab::integer = l.id
LEFT JOIN
        drugs dr ON a.id_drug::integer = dr.id;
"""

# execute sql command
cur.execute(sql_merged)
# fetch all rows in every table
rows = cur.fetchall()

# open a new CSV file for writing
csv_file_path = os.path.join(output_dir, 'merged_table.csv')
with open(csv_file_path, 'w', newline='') as csvfile:
    # create a csv writer object
    csvwriter = csv.writer(csvfile)
    # write the header row
    csvwriter.writerow([desc[0] for desc in cur.description])
    # write the data rows
    csvwriter.writerows(rows)

```

c. *Insert Table Logs*

Dan proses terakhir yaitu menyimpan semua setiap variabel *error message* dari masing - masing fungsi DAG ke dalam tabel *logs* dengan menggunakan fungsi *insert_table_logs*. Fungsi ini dimulai dengan menjalankan setiap fungsi dalam daftar ini dan menyimpan hasilnya ke dalam *error_messages*. Jika suatu fungsi mengalami kesalahan, pesan kesalahan akan dikembalikan

sebagai string; jika tidak, akan mengembalikan *None*. Kemudian melakukan pengecekan untuk setiap fungsi DAG dan mengecek jika ada kesalahan (*err_msg != None*), memasukkan pesan kesalahan ke dalam tabel *logs* bersama dengan *call_date* dan jika tidak ada kesalahan, memasukkan pesan sukses ("Success") ke dalam tabel *logs* bersama dengan *call_date*.

```
# function to insert every logs into table logs
def insert_table_logs():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    # list of every function that has error check condition
    error_messages = [create_table_logs(), create_table_stock_obat(),
create_table_admission(),                insert_data_stock_obat(),
insert_data_admission(),    data_cleaning(),    create_table_doctor(),
convert_datatype(),        create_table_lab(),        create_table_drug(),
create_table_branch(),                create_table_surgery(),
create_table_patient(), create_table_type(), create_table_hoscare(),
insert_table_doctor(),    insert_table_lab(),    insert_table_drug(),
insert_table_branch(),                insert_table_surgery(),
insert_table_patient(),

                insert_table_room_type(),
insert_table_hoscare(),    update_doctor_id(),    alter_doctor_id(),
update_lab_id(),    alter_lab_id(),    update_drug_id(),    alter_drug_id(),
update_branch_id(),        alter_branch_id(),        update_surg_id(),
alter_surg_id(),        update_patient_id(),        alter_pat_id(),
update_room_id(), alter_room_id(), update_hoscare(), alter_hoscare(),
update_drug_stock(),        alter_drug_stock(),        update_branch(),
alter_branch(),        delete_columns(),        export_tables_to_csv(),
export_merged_table_to_csv()]
    # loops through every element
    for err_msg in error_messages:
        # get current datetime
        call_date = datetime.now()
        # if there is any error
        if err_msg != None:
```

```

        # define the value (call_date and error message) to be
        inserted into the table logs
        values_logs = (call_date, err_msg)
        # sql command to insert data into logs table
        insert_sql_logs = "INSERT INTO logs (call_date,
error_message) VALUES (%s, %s);"
        # execute sql command and its value
        cur.execute(insert_sql_logs, values_logs)
    else:
        # Define the success message
        success_message = "Success"
        # Define the value (call_date and success message) to be
        inserted into the logs table
        values_logs = (call_date, success_message)
        # SQL command to insert data into the logs table
        insert_sql_logs = "INSERT INTO logs (call_date,
error_message) VALUES (%s, %s);"
        # Execute the SQL command and its value
        cur.execute(insert_sql_logs, values_logs)
    # commit the task to database
    conn.commit()
    # close the connection to database
    conn.close()

```

II.5. Data Normalization

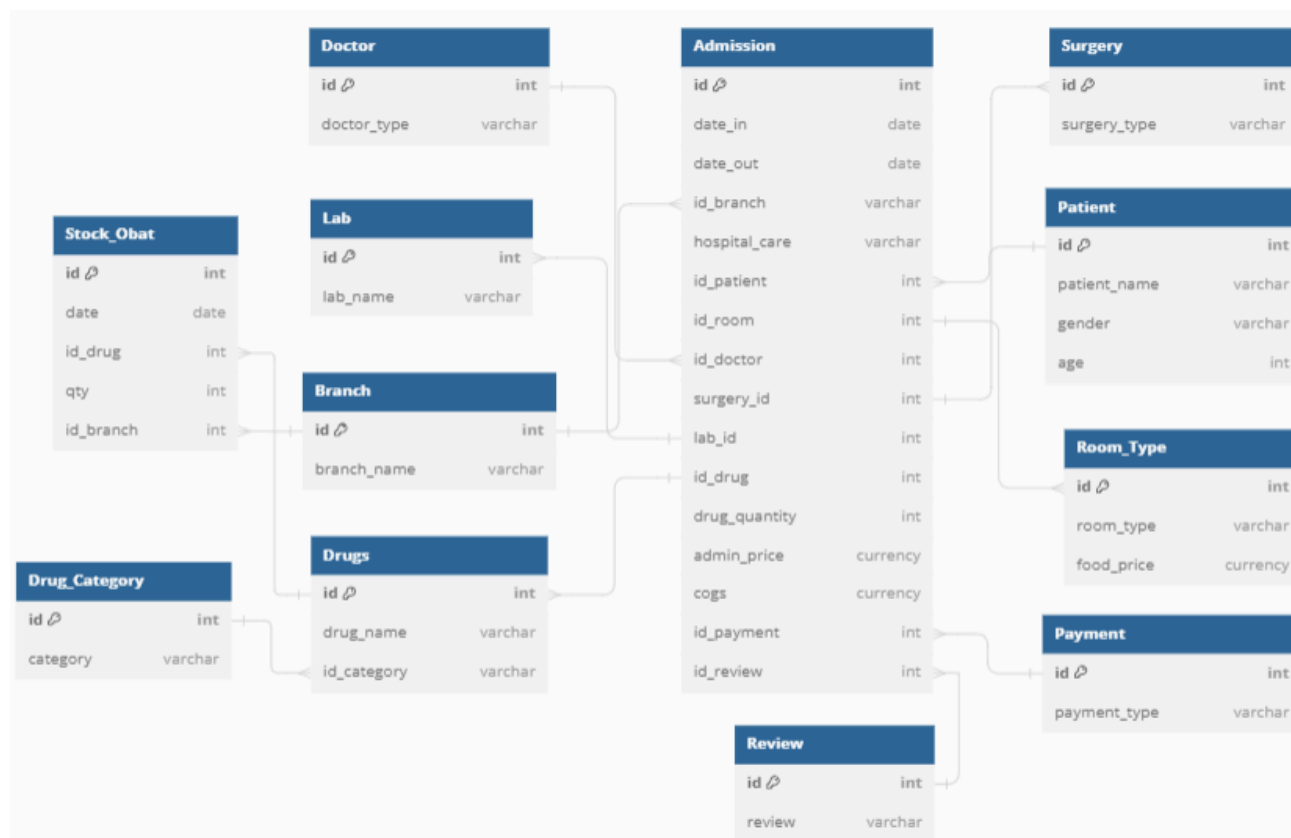
Data normalization atau normalisasi data adalah suatu proses pengorganisasian tabel *relational database* untuk meminimalkan redundansi. Normalisasi biasanya melibatkan pembagian tabel besar menjadi tabel yang lebih kecil (dan tidak terlalu berlebihan) dan mendefinisikan hubungan di antara tabel tersebut. Tujuannya adalah untuk mengisolasi data sehingga penambahan, penghapusan, dan modifikasi suatu skema dapat dilakukan hanya dalam satu tabel dan kemudian disebarkan ke seluruh *database* menggunakan hubungan yang ditentukan. Pada tahap normalisasi data ini, digunakan pendekatan normalisasi data 3NF. Table yang dikatakan 3NF apabila memenuhi kondisi berikut: [1]. M. Ebrahim, "Project-Database Normalization," 2014. [Online]. Available: 10.13140/RG.2.2.19486.18243.

1. Atomisitas, dimana setiap sel dalam tabel harus berisi satu nilai (atomisitas).
2. Tidak ada ketergantungan parsial, dimana setiap atribut bukan *key* harus bergantung pada seluruh *primary key*.
3. Tidak ada ketergantungan transitif, dimana jika suatu atribut bukan kunci bergantung pada atribut bukan kunci lainnya, maka tabel tersebut tidak ada dalam 3NF.

Proses normalisasi data pada *project* dilakukan dengan beberapa tahapan sebagai berikut:

1. Mendesain *Entity Relationship Diagram* (ERD)

Dataset pada github dilakukan normalisasi dengan memecah dari satu dataset menjadi beberapa tabel atau entitas. Admission sebagai entitas utama berisikan *primary key* yaitu *id* yang merupakan *admission id* dan terdapat beberapa *foreign key* yang merupakan *id* dari entitas - entitas lainnya. Selain itu, Stock_Obat sebagai entitas utama kedua yang berisikan *primary key* yaitu *id* dan terdapat beberapa *foreign key* yang merupakan *id* dari entitas - entitas lainnya. Setiap entitas, memiliki hubungan *one-to-many* dengan entitas utama (Admission dan Stock_Obat). Skema detail dari ERD yang digunakan ditampilkan sebagai berikut.



Gambar 2.1 *Entity relationship diagram*

2. Implementasi ERD

Proses implementasi ERD dimulai dengan membuat tabel - tabel master untuk menyimpan nilai - nilai unik dari atribut - atribut yang berbeda yang akan menjadi acuan untuk menormalisasi tabel utama yaitu tabel Admission dan Stock_Obat.

Tabel Doctor

```
# function to create table doctor
def create_table_doctor():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to create table doctor
        sql_doc = """
            create table doctor (
                id serial primary key,
                name varchar(50) not null
            );
        """
        # execute the sql command
        cur.execute(sql_doc)
        # Log the error to the logs table
    except Exception as e:
        # convert any error message and save it in str format
        error_message = str(e)
        # return error message
        return error_message
    # commit the task to database
    conn.commit()
    # close the connection to database
    conn.close()

# function to insert data into table doctor
def insert_table_doctor():
    # connect to database using psycopg2.connect
```

```

conn = db.connect(conn_string)
# create cur object to execute SQL commands
cur = conn.cursor()
try:
    # sql command to insert into table doctor
    sql_ins_doc = """
        insert into doctor (name)
        select distinct doctor
        from admission;
    """

    # execute sql command
    cur.execute(sql_ins_doc)
# Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message

# commit the task to database
conn.commit()

# close the connection to database
conn.close()

```

Tabel Lab

```

# function to create table lab
def create_table_lab():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to create table lab
        sql_lab = """
            create table lab(
                id serial primary key,
                lab_name varchar(50) not null
            )
        """

```



```

        );

        """

        # execute sql command
        cur.execute(sql_lab)
    # Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message

# commit the task to database
conn.commit()

# close the connection to database
conn.close()

# function to insert data into table lab
def insert_table_lab():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to insert into table lab
        sql_ins_lab = """
            insert into lab (lab_name)
            select distinct lab
            from admission
            where lab != '0';
        """

        # execute sql command
        cur.execute(sql_ins_lab)
    # Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message

```

```

        return error_message
    # commit the task to database
    conn.commit()
    # close the connection to database
    conn.close()

```

Tabel Drug

```

# function to create table drug
def create_table_drug():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to create table drugs
        sql_drug = """
            create table drugs(
                id serial primary key,
                drug_name varchar(50) not null,
                drug_type varchar(50) not null
            );
        """
        # execute sql command
        cur.execute(sql_drug)
        # Log the error to the logs table
    except Exception as e:
        # convert any error message and save it in str format
        error_message = str(e)
        # return error message
        return error_message
    # commit the task to database
    conn.commit()
    # close the connection to database
    conn.close()

# function to insert data into table drugs

```

```

def insert_table_drug():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to insert data into table drugs
        sql_ins_drug = """
            insert into drugs (drug_name, drug_type)
            select distinct drug_brands, drug_types
            from admission;
        """
        # execute sql command
        cur.execute(sql_ins_drug)
        # Log the error to the logs table
    except Exception as e:
        # convert any error message and save it in str format
        error_message = str(e)
        # return error message
        return error_message
    # commit the task to database
    conn.commit()
    # close the connection to database
    conn.close()

```

Tabel Branch

```

# function to create table branch
def create_table_branch():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to create table branch
        sql_branch = """
            create table branch (

```

```

        id serial primary key,
        branch_name varchar(10) not null
    );
    """

    # execute sql command
    cur.execute(sql_branch)
    # Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message

# commit the task to database
conn.commit()

# close the connection to database
conn.close()

# function to insert data into table branch
def insert_table_branch():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to insert data into table branch
        sql_ins_branch = """
            insert into branch (branch_name)
            select distinct branch
            from admission;
        """

        # execute sql command
        cur.execute(sql_ins_branch)
    # Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)

```

```

        # return error message
        return error_message
    # commit the task to database
    conn.commit()
    # close the connection to database
    conn.close()

```

Tabel Surgery

```

# function to create table surgery
def create_table_surgery():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to create table surgery
        sql_surg = """
            create table surgery (
                id serial primary key,
                surgery_type varchar(50) not null
            );
        """
        # execute sql command
        cur.execute(sql_surg)
        # Log the error to the logs table
    except Exception as e:
        # convert any error message and save it in str format
        error_message = str(e)
        # return error message
        return error_message
    # commit the task to database
    conn.commit()
    # close the connection to database
    conn.close()

# function to insert data into table surgery

```

```

def insert_table_surgery():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to insert data into table surgery
        sql_ins_surg = """
            insert into surgery (surgery_type)
            select distinct surgery
            from admission
            where surgery != '0';
        """
        # execute sql command
        cur.execute(sql_ins_surg)
        # Log the error to the logs table
    except Exception as e:
        # convert any error message and save it in str format
        error_message = str(e)
        # return error message
        return error_message
    # commit the task to database
    conn.commit()
    # close the connection to database
    conn.close()

```

Tabel Patient

```

# function to create table patient
def create_table_patient():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to create table patient
        sql_pat = """

```

```

        create table patient (
            id serial primary key,
            patient_name varchar(50) not null,
            gender varchar(50) not null,
            age int not null
        );
    """

    # execute sql command
    cur.execute(sql_pat)
# Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message

# commit the task to database
conn.commit()

# close the connection to database
conn.close()

# function to insert data into table patient
def insert_table_patient():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to insert data into table patient
        sql_ins_patient = """
            insert into patient (patient_name, gender, age)
            select distinct fullname, gender, age
            from admission;
        """

        # execute sql command
        cur.execute(sql_ins_patient)
    # Log the error to the logs table

```

```

except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message
# commit the task to database
conn.commit()
# close the connection to database
conn.close()

```

Tabel Room Type

```

# function to create table type
def create_table_type():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to create table room type
        sql_type = """
            create table room_type (
                id serial primary key,
                room_type varchar(50) not null,
                food_price int not null
            );
        """
        # execute sql command
        cur.execute(sql_type)
    # Log the error to the logs table
    except Exception as e:
        # convert any error message and save it in str format
        error_message = str(e)
        # return error message
        return error_message
    # commit the task to database
    conn.commit()

```



```

        # close the connection to database
        conn.close()
# function to insert data into table room type
def insert_table_room_type():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to insert data into table room type
        sql_ins_rt = """
            insert into room_type (room_type, food_price)
            select distinct room, food
            from admission
            where room != '0';
        """
        # execute sql command
        cur.execute(sql_ins_rt)
    # Log the error to the logs table
    except Exception as e:
        # convert any error message and save it in str format
        error_message = str(e)
        # return error message
        return error_message
    # commit the task to database
    conn.commit()
    # close the connection to database
    conn.close()

```

Tabel Hospital Care

```

# function to create table hosp_care
def create_table_hoscare():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()

```

```

try:
    # sql command to create table hosp_care
    sql_hosc = """
        create table hosp_care (
            id serial primary key,
            hospital_care varchar(50) not null
        );
    """

    # execute sql command
    cur.execute(sql_hosc)
    # Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message

# commit the task to database
conn.commit()

# close the connection to database
conn.close()

# function to insert data into table hosp_care
def insert_table_hoscare():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to insert data into table hosp_care
        sql_ins_hosc = """
            insert into hosp_care (hospital_care)
            select distinct hospital_care
            from admission;
        """

        # execute sql command
        cur.execute(sql_ins_hosc)

```

```

# Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message
# commit the task to database
conn.commit()
# close the connection to database
conn.close()

```

Setelah tabel - tabel dibuat dimana data untuk tabel tersebut diambil dari tabel Admission dan Stock_Obat berdasarkan *unique value* dari kolom yang didefinisikan sebagai tabel. Selanjutnya kolom pada tabel Admission dan Stock_Obat tersebut perlu dilakukan *update value* menjadi *id* dari tabel yang sesuai dengan kolom tersebut serta dilakukan penamaan ulang untuk kolom - kolom tersebut.

```

# function to update doctor name column to doctor id in admission
table
def update_doctor_id():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to update admission table
        sql_upt_doc = """
            update admission as a
            set doctor = d.id
            from doctor as d
            where a.doctor = d.name;
        """
        # execute sql command
        cur.execute(sql_upt_doc)

# Log the error to the logs table

```

```

except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message
# commit the task to database
conn.commit()
# close the connection to database
conn.close()

# function to change doctor name column into id_doctor
def alter_doctor_id():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to change column name
        sql_alt_doc = """
            alter table admission
            rename column doctor to id_doctor;
        """
        # execute sql command
        cur.execute(sql_alt_doc)

    # Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message
# commit the task to database
conn.commit()
# close the connection to database
conn.close()

```

```

# function to update lab name column to lab id in admission table
def update_lab_id():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to update admission table
        sql_upt_lab = """
            update admission as a
            set lab = l.id
            from lab as l
            where a.lab = l.lab_name;
        """

        # execute sql command
        cur.execute(sql_upt_lab)

        # Log the error to the logs table
    except Exception as e:
        # convert any error message and save it in str format
        error_message = str(e)
        # return error message
        return error_message

    # commit the task to database
    conn.commit()

    # close the connection to database
    conn.close()

# function to change lab name column into id_lab
def alter_lab_id():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands

```

```

cur = conn.cursor()
try:
    # sql command to change column name
    sql_alt_lab = """
        alter table admission
        rename column lab to id_lab;
    """
    # execute sql command
    cur.execute(sql_alt_lab)

    # Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message

# commit the task to database
conn.commit()

# close the connection to database
conn.close()

# function to update drug brands column to drug id in admission
table
def update_drug_id():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to update admission table
        sql_upt_drg = """
            update admission as a
            set drug_brands = g.id
            from drugs as g
            where a.drug_brands = g.drug_name and a.drug_types =
g.drug_type;

```

```

        """

        # execute sql command
        cur.execute(sql_upt_drg)

# Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message

# commit the task to database
conn.commit()

# close the connection to database
conn.close()

# function to change drug brands column into id_drug
def alter_drug_id():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to change column name
        sql_alt_drg = """
            alter table admission
            rename column drug_brands to id_drug;
        """

        # execute sql command
        cur.execute(sql_alt_drg)

# Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message

```

```

# commit the task to database
conn.commit()

# close the connection to database
conn.close()

# function to update branch name column to branch id in admission
table
def update_branch_id():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to update admission table
        sql_upt_brch = """
            update admission as a
            set branch = b.id
            from branch as b
            where a.branch = b.branch_name;
        """

        # execute sql command
        cur.execute(sql_upt_brch)

        # Log the error to the logs table
    except Exception as e:
        # convert any error message and save it in str format
        error_message = str(e)
        # return error message
        return error_message

    # commit the task to database
    conn.commit()

    # close the connection to database
    conn.close()

# function to change branch name column into id_branch
def alter_branch_id():

```



```

# connect to database using psycopg2.connect
conn = db.connect(conn_string)
# create cur object to execute SQL commands
cur = conn.cursor()
try:
    # sql command to change column name
    sql_alt_branch = """
        alter table admission
        rename column branch to id_branch;
    """

    # execute sql command
    cur.execute(sql_alt_branch)

# Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message

# commit the task to database
conn.commit()

# close the connection to database
conn.close()

# function to update surgery type column to surgery id in admission
table
def update_surg_id():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to update admission table
        sql_upt_surg = """
            update admission as a
            set surgery = s.id

```

```

        from surgery as s
        where a.surgery = s.surgery_type;
    """

    # execute sql command
    cur.execute(sql_upt_surg)

# Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message

# commit the task to database
conn.commit()

# close the connection to database
conn.close()

# function to change surgery column into id_surgery
def alter_surg_id():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to change column name
        sql_alt_surg = """
            alter table admission
            rename column surgery to id_surgery;
        """

        # execute sql command
        cur.execute(sql_alt_surg)

# Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)

```

```

        # return error message
        return error_message
    # commit the task to database
    conn.commit()
    # close the connection to database
    conn.close()

# function to update patient fullname column to patient id in
admission table
def update_patient_id():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to update admission table
        sql_upt_pat = """
            update admission as a
            set fullname = p.id
            from patient as p
            where a.fullname = p.patient_name and a.gender =
p.gender and a.age = p.age;
        """
        # execute sql command
        cur.execute(sql_upt_pat)

    # Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message
# commit the task to database
conn.commit()
# close the connection to database
conn.close()

```

```

# function to change fullname column into id_patient
def alter_pat_id():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to change column name
        sql_alt_pat = """
            alter table admission
            rename column fullname to id_patient;
        """
        # execute sql command
        cur.execute(sql_alt_pat)

        # Log the error to the logs table
    except Exception as e:
        # convert any error message and save it in str format
        error_message = str(e)
        # return error message
        return error_message
    # commit the task to database
    conn.commit()
    # close the connection to database
    conn.close()

# function to update room column to room id in admission table
def update_room_id():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to update admission table
        sql_upt_room = """
            update admission as a

```

```

        set room = r.id
        from room_type as r
            where a.room = r.room_type and a.food =
r.food_price;
        """

        # execute sql command
        cur.execute(sql_upt_room)

# Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message

# commit the task to database
conn.commit()

# close the connection to database
conn.close()

# function to change room column into id_room
def alter_room_id():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to change column name
        sql_alt_room = """
            alter table admission
            rename column room to id_room;
        """

        # execute sql command
        cur.execute(sql_alt_room)

# Log the error to the logs table
except Exception as e:

```

```

        # convert any error message and save it in str format
        error_message = str(e)
        # return error message
        return error_message
    # commit the task to database
    conn.commit()
    # close the connection to database
    conn.close()

# function to update hospital care column to hc id in admission
table
def update_hoscare():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to update admission table
        sql_upt_hosc = """
            update admission as a
            set hospital_care = hc.id
            from hosp_care as hc
            where a.hospital_care = hc.hospital_care;
        """
        # execute sql command
        cur.execute(sql_upt_hosc)

    # Log the error to the logs table
    except Exception as e:
        # convert any error message and save it in str format
        error_message = str(e)
        # return error message
        return error_message
    # commit the task to database
    conn.commit()
    # close the connection to database

```

```

conn.close()

# function to change room column into id_room
def alter_hoscare():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to update admission table
        sql_alt_room = """
            alter table admission
            rename column hospital_care to id_hospital_care;
        """
        # execute sql command
        cur.execute(sql_alt_room)

    # Log the error to the logs table
    except Exception as e:
        # convert any error message and save it in str format
        error_message = str(e)
        # return error message
        return error_message
    # commit the task to database
    conn.commit()
    # close the connection to database
    conn.close()

# function to update drugs column to drug id in drug_stock table
def update_drug_stock():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to update drug_stock table

```

```

sql_upt_dr = """
    update drug_stock as ds
    set drugs = d.id
    from drugs as d
    where ds.drugs = d.drug_name;
"""

# execute sql command
cur.execute(sql_upt_dr)

# Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message

# commit the task to database
conn.commit()

# close the connection to database
conn.close()

# function to change drugs column into id_drug
def alter_drug_stock():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to change column name
        sql_alt_dr = """
            alter table drug_stock
            rename column drugs to id_drug;
        """

        # execute sql command
        cur.execute(sql_alt_dr)

# Log the error to the logs table

```



```

except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message

# commit the task to database
conn.commit()

# close the connection to database
conn.close()

# function to update branch column to branch id in drug_stock table
def update_branch():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to update drug_stock table
        sql_upt_branch = """
            update drug_stock as ds
            set branch = b.id
            from branch as b
            where ds.branch = b.branch_name;
        """
        # excute sql command
        cur.execute(sql_upt_branch)

    # Log the error to the logs table
except Exception as e:
    # convert any error message and save it in str format
    error_message = str(e)
    # return error message
    return error_message

# commit the task to database
conn.commit()

# close the connection to database

```

```

conn.close()

# function to change branch column into id_branch
def alter_branch():
    # connect to database using psycopg2.connect
    conn = db.connect(conn_string)
    # create cur object to execute SQL commands
    cur = conn.cursor()
    try:
        # sql command to change column name
        sql_alt_bh = """
            alter table drug_stock
            rename column branch to id_branch;
        """
        # execute sql command
        cur.execute(sql_alt_bh)

    # Log the error to the logs table
    except Exception as e:
        # convert any error message and save it in str format
        error_message = str(e)
        # return error message
        return error_message
    # commit the task to database
    conn.commit()
    # close the connection to database
    conn.close()

```

II.6. *Setting up Airflow Pipeline*

Airflow pipeline dapat dijalankan dengan cara melakukan *set up* untuk beberapa parameter. Dimulai dengan melakukan *set up* untuk *default arguments* yang digunakan untuk mendefinisikan argumen *default* yang akan diterapkan pada semua operator di dalam DAG kecuali jika operator tersebut memiliki argumen yang bertentangan. Parameter *retries* merupakan argumen yang menentukan berapa kali *airflow* akan mencoba menjalankan *task* jika terjadi

kegagalan. Sedangkan parameter *retry_delay* adalah argumen yang menentukan berapa lama *airflow* akan menunggu sebelum mencoba kembali menjalankan *task* yang gagal.

```
default_args = {
    'owner': 'ucup',
    'retries': 5,
    'retry_delay': timedelta(minutes=5)
}
```

Selanjutnya, untuk memanggil fungsi DAG beserta argumen *default* yang telah dilakukan *set up*, digunakan blok code *with DAG(...) as dag:*. Dengan *with* yang digunakan untuk membuat konteks DAG. Semua operator yang didefinisikan dalam blok *with* ini akan menjadi bagian dari DAG yang didefinisikan. Kemudian parameter *default_args* merujuk pada argumen *default* yang sudah di-*set up* sebelumnya. Parameter *dag_id* merupakan *identifier* unik untuk DAG untuk menghindari konflik di *airflow environment*. Parameter *start_date* digunakan untuk menentukan tanggal dan waktu mulai dari DAG. Dan terakhir, parameter *schedule_interval* digunakan untuk menentukan interval penjadwalan untuk DAG ini menggunakan ekspresi *cron*. Dalam contoh ini, DAG dijadwalkan untuk berjalan setiap hari pada tengah malam dengan ekspresi *cron 0 0 * * **. Dengan rincian sebagai berikut.

- 0 pertama merupakan menit (00 menit)
- 0 kedua merupakan jam (00 jam)
- * pertama memiliki arti setiap hari dalam sebulan
- * kedua memiliki arti setiap bulan
- * ketiga memiliki arti setiap hari dalam seminggu

```
with DAG(
    default_args=default_args, # Default arguments for the DAG
    dag_id='dag_with_logs_11', # Unique identifier for the DAG
    start_date=datetime(2024, 6, 4), # Start date of the DAG
    schedule_interval='0 0 * * *' # Cron expression for scheduling the
    DAG
) as dag:
```

Selanjutnya diperlukan pendefinisian *task* pada *airflow* untuk masing - masing fungsi DAG yang telah didefinisikan dengan menggunakan *PythonOperator* untuk mengeksekusi fungsi Python sebagai bagian dari alur kerja (workflow) di DAG. Parameter *task_id* merupakan

identifier unik untuk tugas di dalam DAG. *python_callable* adalah parameter yang menerima fungsi Python yang akan dieksekusi oleh *task*. Untuk detail masing - masing *task* dijabarkan pada *code* berikut.

```
# Task to create the 'stock_obat' table in PostgreSQL
create_stock_obat = PythonOperator(
    task_id='create_postgres_stock_obat', # Unique identifier for the
task
    python_callable=create_table_stock_obat # Python function to be
executed
)
# Task to insert data into the 'stock_obat' table
insert_stock_obat = PythonOperator(
    task_id='insert_into_table_stock_obat',
    python_callable=insert_data_stock_obat,
    provide_context=True # Provide Airflow context to the function
)
# Task to create the 'admission' table in PostgreSQL
create_admission = PythonOperator(
    task_id='create_postgres_admission',
    python_callable=create_table_admission
)
# Task to insert data into the 'admission' table
insert_admission = PythonOperator(
    task_id='insert_into_table_admission',
    python_callable=insert_data_admission,
    provide_context=True
)
# Task to create the 'logs' table in PostgreSQL
create_logs = PythonOperator(
    task_id='create_table_logs',
    python_callable=create_table_logs,
    provide_context=True
)
# Task to clean the data
cleaning_data = PythonOperator(
    task_id='cleaning_data',
```

```

        python_callable=data_cleaning,
        provide_context=True
    )
    # Task to create the 'doctor' table in PostgreSQL
    create_doctor = PythonOperator(
        task_id='create_table_doctor',
        python_callable=create_table_doctor,
        provide_context=True
    )
    # Task to create the 'lab' table in PostgreSQL
    create_lab = PythonOperator(
        task_id='create_table_lab',
        python_callable=create_table_lab,
        provide_context=True
    )
    # Task to create the 'drug' table in PostgreSQL
    create_drug = PythonOperator(
        task_id='create_table_drug',
        python_callable=create_table_drug,
        provide_context=True
    )
    # Task to create the 'branch' table in PostgreSQL
    create_branch = PythonOperator(
        task_id='create_table_branch',
        python_callable=create_table_branch,
        provide_context=True
    )
    # Task to create the 'surgery' table in PostgreSQL
    create_surgery = PythonOperator(
        task_id='create_table_surgery',
        python_callable=create_table_surgery,
        provide_context=True
    )
    # Task to create the 'patient' table in PostgreSQL
    create_patient = PythonOperator(
        task_id='create_table_patient',

```

```

        python_callable=create_table_patient,
        provide_context=True
    )
# Task to create the 'type' table in PostgreSQL
create_type = PythonOperator(
    task_id='create_table_type',
    python_callable=create_table_type,
    provide_context=True
)
# Task to create the 'hoscare' table in PostgreSQL
create_hoscare = PythonOperator(
    task_id='create_table_hoscare',
    python_callable=create_table_hoscare,
    provide_context=True
)
# Task to insert data into the 'doctor' table
insert_doctor = PythonOperator(
    task_id='insert_table_doctor',
    python_callable=insert_table_doctor,
    provide_context=True
)
# Task to insert data into the 'lab' table
insert_lab = PythonOperator(
    task_id='insert_table_lab',
    python_callable=insert_table_lab,
    provide_context=True
)
# Task to insert data into the 'drug' table
insert_drug = PythonOperator(
    task_id='insert_table_drug',
    python_callable=insert_table_drug,
    provide_context=True
)
# Task to insert data into the 'branch' table
insert_branch = PythonOperator(
    task_id='insert_table_branch',

```

```

        python_callable=insert_table_branch,
        provide_context=True
    )
    # Task to insert data into the 'surgery' table
    insert_surgery = PythonOperator(
        task_id='insert_table_surgery',
        python_callable=insert_table_surgery,
        provide_context=True
    )
    # Task to insert data into the 'patient' table
    insert_patient = PythonOperator(
        task_id='insert_table_patient',
        python_callable=insert_table_patient,
        provide_context=True
    )
    # Task to insert data into the 'room_type' table
    insert_room_type = PythonOperator(
        task_id='insert_table_room_type',
        python_callable=insert_table_room_type,
        provide_context=True
    )
    # Task to insert data into the 'hoscare' table
    insert_hoscare = PythonOperator(
        task_id='insert_table_hoscare',
        python_callable=insert_table_hoscare,
        provide_context=True
    )
    # Task to convert data types
    convert_dtype = PythonOperator(
        task_id='convert_datatype',
        python_callable=convert_datatype,
        provide_context=True
    )
    # Task to update the 'doctor' table
    update_doc = PythonOperator(
        task_id='update_doctor_id',

```

```

        python_callable=update_doctor_id,
        provide_context=True
    )
    # Task to alter the 'doctor' table
    alter_doc = PythonOperator(
        task_id='alter_doctor_id',
        python_callable=alter_doctor_id,
        provide_context=True
    )
    # Task to update the 'lab' table
    update_lab = PythonOperator(
        task_id='update_lab_id',
        python_callable=update_lab_id,
        provide_context=True
    )
    # Task to alter the 'lab' table
    alter_lab = PythonOperator(
        task_id='alter_lab_id',
        python_callable=alter_lab_id,
        provide_context=True
    )
    # Task to update the 'drug' table
    update_drug = PythonOperator(
        task_id='update_drug_id',
        python_callable=update_drug_id,
        provide_context=True
    )
    # Task to alter the 'drug' table
    alter_drug = PythonOperator(
        task_id='alter_drug_id',
        python_callable=alter_drug_id,
        provide_context=True
    )
    # Task to update the 'branch' table
    update_brch_id = PythonOperator(
        task_id='update_branch_id',

```



```

        python_callable=update_branch_id,
        provide_context=True
    )
    # Task to alter the 'branch' table
    alter_brch_id = PythonOperator(
        task_id='alter_branch_id',
        python_callable=alter_branch_id,
        provide_context=True
    )
    # Task to update the 'surgery' table
    update_surgery = PythonOperator(
        task_id='update_surg_id',
        python_callable=update_surg_id,
        provide_context=True
    )
    # Task to alter the 'surgery' table
    alter_surgery = PythonOperator(
        task_id='alter_surg_id',
        python_callable=alter_surg_id,
        provide_context=True
    )
    # Task to update the 'patient' table
    update_pat = PythonOperator(
        task_id='update_patient_id',
        python_callable=update_patient_id,
        provide_context=True
    )
    # Task to alter the 'patient' table
    alter_pat = PythonOperator(
        task_id='alter_pat_id',
        python_callable=alter_pat_id,
        provide_context=True
    )
    # Task to update the 'room' table
    update_room = PythonOperator(
        task_id='update_room_id',

```

```

        python_callable=update_room_id,
        provide_context=True
    )
    # Task to alter the 'room' table
    alter_room = PythonOperator(
        task_id='alter_room_id',
        python_callable=alter_room_id,
        provide_context=True
    )
    # Task to update the 'hospital care' table
    update_hos_care = PythonOperator(
        task_id='update_hoscare',
        python_callable=update_hoscare,
        provide_context=True
    )
    # Task to alter the 'hospital care' table
    alter_hos_care = PythonOperator(
        task_id='alter_hoscare',
        python_callable=alter_hoscare,
        provide_context=True
    )
    # Task to update the 'drug stock' table
    update_drugstock = PythonOperator(
        task_id='update_drug_stock',
        python_callable=update_drug_stock,
        provide_context=True
    )
    # Task to alter the 'drug stock' table
    alter_drugstock = PythonOperator(
        task_id='alter_drug_stock',
        python_callable=alter_drug_stock,
        provide_context=True
    )
    # Task to update the 'branch' table
    update_brch = PythonOperator(
        task_id='update_branch',

```

```

        python_callable=update_branch,
        provide_context=True
    )
    # Task to alter the 'branch' table
    alter_brch = PythonOperator(
        task_id='alter_branch',
        python_callable=alter_branch,
        provide_context=True
    )
    # Task to delete specific columns
    delete_col = PythonOperator(
        task_id='delete_columns',
        python_callable=delete_columns,
        provide_context=True
    )
    # Task to export tables to CSV files
    export_table = PythonOperator(
        task_id='export_tables_to_csv',
        python_callable=export_tables_to_csv,
        provide_context=True
    )
    # Task to export merged table to a CSV file
    export_merged_table = PythonOperator(
        task_id='export_merged_table-to_csv',
        python_callable=export_merged_table_to_csv,
        provide_context=True
    )
    # Task to insert logs into the 'logs' table
    insert_logs = PythonOperator(
        task_id='insert_table_logs',
        python_callable=insert_table_logs,
        provide_context=True
    )

```

Terakhir, diperlukan pendefinisian untuk *task dependencies* antara masing - masing *task* dalam DAG dengan cara menentukan urutan eksekusi *task - task* dalam alur kerja proses DAG. Dalam membuat alur kerja dari proses DAG, diperlukan operator (>>) untuk menentukan urutan

eksekusi tugas dalam DAG. Tugas di sebelah kiri operator >> harus selesai dieksekusi sebelum tugas di sebelah kanan dapat dimulai. Alur kerja proses DAG digambarkan sebagai berikut.

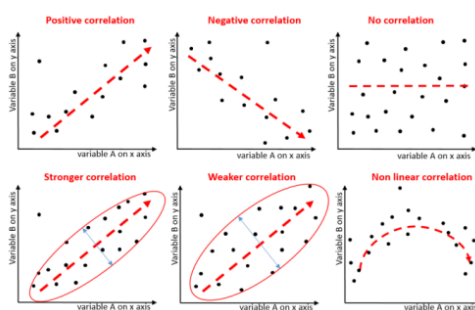
```
# Define the task dependencies
create_stock_obat >> create_admission >> create_logs >> create_doctor
>> create_lab >> create_drug >> create_branch >> create_surgery >>
create_patient >> create_type >> create_hoscare >> insert_stock_obat >>
insert_admission >> cleaning_data >> convert_dtype >> insert_doctor >>
insert_lab >> insert_drug >> insert_branch >> insert_surgery >>
insert_patient >> insert_hoscare >> insert_room_type >> update_doc >>
alter_doc >> update_lab >> alter_lab >> update_drug >> alter_drug >>
update_brch_id >> alter_brch_id >> update_surgery >> alter_surgery >>
update_pat >> alter_pat >> update_room >> alter_room >> update_hos_care >>
alter_hos_care >> update_drugstock >> alter_drugstock >> update_brch >>
alter_brch >> delete_col >> export_table >> export_merged_table >>
insert_logs
```

II.7. Label Encoding

Label Encoding adalah teknik pra pemrosesan data yang digunakan untuk mengubah nilai-nilai kategori pada data menjadi format numerik yang dapat digunakan oleh algoritma pembelajaran mesin. Kelebihan mencakup Prosesnya cepat dan mudah diimplementasikan dan dapat mengubah data kategori menjadi numerik mengurangi kompleksitas data [1].

II.8. Correlation Analysis

Correlation analysis atau analisis korelasi adalah ukuran statistik yang banyak digunakan di mana berbagai penelitian telah secara efisien mengidentifikasi hubungan kolinear yang relevan di antara berbagai atribut kumpulan data [2].



Gambar 2.1. Korelasi berdasarkan arah, bentuk, dan kekuatan [2].

Korelasi menentukan bagaimana suatu variabel bergerak atau berubah hubungannya dengan variabel lainnya. Hal ini memberikan gambaran tentang derajat hubungan kedua variabel. Korelasi adalah ukuran analisis bivariat yang menggambarkan hubungan antara berbagai variabel. Di sebagian besar bisnis, mengekspresikan satu subjek dalam kaitannya dengan hubungannya dengan subjek lain akan berguna.

II.9. Variance Inflation Factor (VIF)

VIF adalah alat untuk mengukur dan mengukur seberapa besar varians meningkat. VIF biasanya dihitung oleh perangkat lunak sebagai bagian dari analisis regresi dan akan muncul di kolom VIF sebagai bagian dari keluaran. Untuk menafsirkan nilai VIF digunakan aturan berikut pada tabel di bawah ini:

Tabel 1. Interpretasi nilai VIF

Nilai VIF	Keterangan
$VIF = 1$	Tidak berkorelasi
$1 < VIF \leq 5$	Berkorelasi sedang
$VIF > 5$	Berkorelasi kuat

Selain itu, VIF dapat juga menunjukkan apakah prediktornya berkorelasi, square root dari VIF menunjukkan seberapa besar standar errornya, misalnya jika VIF 9 berarti standar error koefisien prediktor tersebut adalah 3 kali lebih besar jika prediktor tersebut tidak berkorelasi dengan prediktor lainnya. VIF dapat dihitung dengan menggunakan rumus:

$$VIF = 1 \div (1 - R_i^2)$$

VIF ini dapat dihitung untuk setiap prediktor dalam model, dan caranya adalah dengan melakukan regresi variabel dengan asumsi itu adalah variabel ke-i terhadap semua prediktor lainnya. Dengan memperoleh R_i^2 yang dapat digunakan untuk mencari VIF, hal yang sama dapat diterapkan pada semua prediktor lainnya.

Dengan begitu, berdasarkan Tabel-1 terlihat bahwa nilai VIF untuk x_1 adalah 1 baik untuk model sederhana maupun model ganda, sama untuk x_2 tidak berubah dan menjadi 1, hal

ini disebabkan oleh a korelasi yang sangat rendah untuk kumpulan data pertama, sedangkan untuk kumpulan data kedua, VIF untuk kedua variabel diubah dari 1 untuk model sederhana menjadi 113,67 untuk model ganda. Dalam kasus terakhir kita tidak dapat memulai regresi kecuali masalah ini diselesaikan [3].

II.10. Random Forest Regressor

Random Forest Regressor adalah model *machine learning* yang termasuk dalam *ensemble learning*, yang berarti model ini menggabungkan prediksi dari beberapa model lain (dalam hal ini, pohon keputusan) untuk membuat prediksi yang lebih baik. Dalam hal regresi, model ini menggunakan kumpulan pohon keputusan untuk membuat prediksi kontinu. Setiap pohon keputusan dalam model ini dibangun secara independen menggunakan subset acak dari data pelatihan, dan prediksi dari semua pohon keputusan digunakan untuk menghasilkan prediksi akhir melalui proses penggabungan atau perataan. Hal ini membuat Random Forest Regressor memiliki kemampuan untuk menangani data dengan fitur yang kompleks dan jumlah dimensi yang tinggi, serta memiliki keunggulan dalam mengurangi overfitting dibandingkan dengan pohon keputusan tunggal [4].

II.11. Cross Validation

Cross validation dalam machine learning adalah sebuah teknik evaluasi model yang digunakan untuk mengukur seberapa baik model dapat menggeneralisasi dari data pelatihan ke data baru yang belum pernah dilihat sebelumnya. Teknik ini membagi data menjadi subset yang saling tumpang tindih dan menggunakan data tersebut secara bergantian sebagai data pelatihan dan pengujian. Hal ini memastikan bahwa setiap titik data digunakan baik untuk pelatihan maupun pengujian, sehingga memberikan estimasi kinerja model yang lebih konsisten dan dapat diandalkan [5].

II.12. MAE

Mean Absolute Error (MAE) adalah menunjukkan rata – rata kesalahan (Error) absolut antara hasil peramalan/prediksi dengan nilai aktual [6]. MAE memberikan gambaran seberapa besar kesalahan rata-rata yang dibuat oleh model tanpa memperhatikan arah kesalahan (positif

atau negatif). MAE mudah diinterpretasikan karena menggunakan satuan yang sama dengan data asli.

$$MAE = \frac{1}{n} \sum_{t=1}^n |f_t - y_t|$$

Keterangan :

f_t : Nilai hasil perkiraan (prediksi) ke- t ($t=1, \dots, n$)

y_t : Nilai sebenarnya (aktual) ke- t ($t=1, \dots, n$)

n : Banyaknya data yang diuji

II.13. RMSE

Root Mean Squared Error (RMSE) adalah nilai rata - rata dari jumlah kuadrat kesalahan atau metrik evaluasi yang mengukur rata-rata kuadrat kesalahan antara prediksi model dengan nilai aktual, kemudian diakarkan. Nilai hasil RMSE akan baik hasil perkiraan (prediksi) apabila nilai RMSE semakin rendah [6]. RMSE memberikan penalti lebih besar untuk kesalahan yang lebih besar, sehingga lebih sensitif terhadap outlier dibandingkan MAE

$$RMSE = \frac{\sum_{t=1}^n (Y'_t - Y_t)^2}{n}$$

Keterangan:

Y'_t : Nilai data sebenarnya (aktual) ke- t ($t=1, \dots, n$)

Y_t : Nilai sebenarnya (aktual) ke- t ($t=1, \dots, n$)

n : Banyaknya data yang diuji

II.14. *R2-score*

R-squared (R^2), suatu nilai yang menampilkan seberapa besar variabel independen mempengaruhi variabel dependen [6]. Nilainya berkisar antara 0 dan 1, dengan 1 menunjukkan bahwa model menjelaskan semua variansi dalam data, dan 0 menunjukkan bahwa model tidak menjelaskan variansi sama sekali.

$$R^2 = 1 - \frac{\sum_{t=1}^n (y_t - \hat{y})^2}{\sum_{t=1}^n (y_t - \bar{y})^2}$$

Keterangan:

y_t : Data yang diuji respon ke - t ($t=1, \dots, n$)

\hat{y} : Ramalan respon ke- t ($t=1, \dots, n$)

\bar{y} : Rata-rata, n :banyaknya data yang diuji

II.15. *Hyperparameter Tuning*

Hyperparameter tuning adalah proses untuk menemukan kombinasi optimal dari hyperparameter yang digunakan oleh model pembelajaran mesin untuk memaksimalkan performanya. Hyperparameter sendiri adalah parameter yang nilainya ditetapkan sebelum proses pelatihan dimulai, dan tidak diupdate selama pelatihan [7].

II.16. *Data Visualization*

1. Menentukan visualisasi yang ingin ditampilkan berdasarkan data yang dimiliki, informasi yang ingin disampaikan dan *metrics* yang digunakan
2. Data yang sudah dinormalisasi adalah dalam format.csv
3. Data di import ke Power BI sesuai dengan tabel yang sudah dinormalisasi yaitu tabel *Admission, Branch, Doctor, Drug_Stock, Drugs, Hosp_care, Lab, Patient, Room_Type* dan *Surgery*
4. Mengecek data yang sudah diimport sudah sesuai dan bersih
5. Data yang di *import* kemudian di *load* di Power BI
6. Menghubungkan *relationship* antara satu tabel dengan tabel lainnya sesuai dengan diagram ERD yang telah dibuat.
7. Menambahkan *measurement* untuk *metrics* yang dicari menggunakan fitur DAX di Power BI
8. Menambahkan tabel baru *Date* dengan menggunakan fitur measurement di DAX.
 - Tabel Date

Tabel Date merupakan tabel yang digunakan untuk menggabungkan 3 rentang waktu yaitu *date_in*, *date_out* dan *date_buy* supaya periode waktu yang digunakan selaras.

- Rentang waktu *date_in* dan *date_out* digunakan dalam tabel *admission* untuk mengetahui rentang waktu pemakaian obat
- Rentang waktu *date_buy* digunakan dalam tabel *drug_stock* untuk mengetahui rentang waktu rumah sakit membeli obat sebagai stok obat


```

Date =
VAR MinDate = MINX(
    UNION(
        SELECTCOLUMNS('admission', "Date", 'admission'[date_in]),
        SELECTCOLUMNS('admission', "Date", 'admission'[date_out]),
        SELECTCOLUMNS('drug_stock', "Date", 'drug_stock'[date_buy])
    ),
    [Date]
)
VAR MaxDate = MAXX(
    UNION(
        SELECTCOLUMNS('admission', "Date", 'admission'[date_in]),
        SELECTCOLUMNS('admission', "Date", 'admission'[date_out]),
        SELECTCOLUMNS('drug_stock', "Date", 'drug_stock'[date_buy])
    ),
    [Date]
)
RETURN
ADDCOLUMNS(
    CALENDAR(MinDate, MaxDate),
    "Year", YEAR([Date]),
    "Quarter", "Q" & FORMAT(QUARTER([Date]), "0"),
    "Month", FORMAT([Date], "MMMM"),
    "Month Number", MONTH([Date]),
    "Weekday", FORMAT([Date], "dddd"),
    "Day", DAY([Date])
)

```

MinDate :

Variabel Min Date menggunakan fungsi MIN X untuk menghitung tanggal terawal dari hasil penggabungan tiga kolom tanggal berbeda. Fungsi UNION digunakan untuk menggabungkan tiga set data yang berbeda menjadi satu tabel, di mana setiap set data terdiri dari satu kolom tanggal yang dinamakan "Date". Fungsi SELECT COLUMNS digunakan untuk memilih kolom yang relevan dari setiap tabel.

MaxDate:

Variabel MaxDate serupa dengan MinDate, tetapi menggunakan fungsi MAX X untuk menghitung tanggal terakhir dari hasil penggabungan tersebut.

RETURN:

Bagian RETURN ini mengembalikan tabel baru yang dibuat oleh fungsi CALENDER yang menghasilkan tabel tanggal mulai dari MinDate hingga MaxDate.

ADD COLOUMNS:

Fungsi ADD COLOUMNS kemudian digunakan untuk menambahkan kolom tambahan ke tabel tersebut:

- "Year": Tahun dari tanggal.
- "Quarter": Kuartal dari tanggal dalam format "Qn".
- "Month": Nama bulan dari tanggal.
- "Month Number": Nomor bulan dari tanggal.
- "Weekday": Nama hari dalam seminggu dari tanggal.
- "Day": Hari dari tanggal.

DAX ini membuat sebuah tabel kalender yang mencakup seluruh tanggal dari tanggal terawal hingga tanggal terakhir yang ditemukan dalam kolom date_in, date_out, dan date_buy dari tabel admission dan drug_stock, serta menambahkan informasi tahun, kuartal, bulan, nomor bulan, hari dalam seminggu, dan hari dari setiap tanggal dalam rentang tersebut.

- **Total_Patients**

Fungsi ini digunakan untuk menghitung jumlah pasien unik berdasarkan id_patient di tabel admission

```
Total_Patients = DISTINCTCOUNT(admission[id_patient])
```

DISTINCT COUNT:

Fungsi ini untuk menghitung jumlah nilai unik atau berbeda dari kolom id_patient. Apabila ada nilai yang muncul lebih dari sekali dalam kolom id_patient, fungsi ini hanya akan menghitungnya satu kali.

Admission[id_patient]:

Ini merujuk pada kolom id_patient dalam tabel admission. Id_patient menjadi sebuah kolom yang berisi ID unik untuk setiap pasien.

- **Total_Stock_Drug**

Total_Stock_Drug digunakan untuk menghitung total stok obat-obatan berdasarkan data yang terdapat dalam tabel drug_stock.

```
Total_Stock_Drug = SUM('drug_stock'[Stock Drugs])
```

SUM: Fungsi untuk menjumlahkan nilai dari kolom Stock Drugs untuk seluruh baris dalam tabel drug_stock.

'drug_stock': Tabel yang berisi data terkait persediaan obat-obatan yang tersedia.

'drug_stock'[Stock Drugs]: Kolom dalam tabel drug_stock yang berisi data tentang jumlah stok obat-obatan.

- **Total_Usage_Drug**

Total_Usage_Drug digunakan untuk menghitung total penggunaan obat-obatan berdasarkan data yang terdapat dalam tabel admission. Fungsi ini memanfaatkan fungsi SUM X, yang bertujuan untuk menjumlahkan nilai dari kolom Usage Drugs untuk setiap baris dalam tabel admission.

```
Total_Usage_Drug =  
  
SUMX (  
  
    'admission',  
  
    'admission'[Usage Drugs]
```

)

SUM X: Fungsi SUM X akan menjumlahkan nilai dari kolom Usage Drugs untuk setiap baris dalam tabel admission.

'admission': Tabel yang berisi data terkait penerimaan pasien yang di dalamnya terdapat kolom Usage Drugs.

'admission'[Usage Drugs]: Kolom dalam tabel admission yang berisi data tentang jumlah penggunaan obat-obatan.

- **GAP_Drugs**

GAP_Drugs digunakan untuk menghitung selisih antara total stok obat-obatan dan total penggunaan obat-obatan. Fungsi ini untuk mengetahui ketersediaan obat-obatan dan obat jumlah yang telah digunakan

$$\text{GAP_Drugs} = [\text{Total_Stock_Drug}] - [\text{Total_Usage_Drug}]$$

[Total_Stock_Drug]: Mengacu pada total stok obat-obatan yang tersedia, yang dihitung menggunakan rumus yang telah diperoleh sebelumnya menunjukkan jumlah keseluruhan obat-obatan yang ada dalam inventory

[Total_Usage_Drug]: Mengacu pada total penggunaan obat-obatan, yang dihitung menggunakan rumus yang telah diperoleh sebelumnya . Ini menunjukkan jumlah keseluruhan obat-obatan yang telah digunakan berdasarkan data penerimaan pasien.

Selisih: Rumus ini menghitung selisih antara total stok dan total penggunaan untuk menentukan jumlah obat-obatan yang tersisa setelah memperhitungkan penggunaan.

BAB III HASIL DAN PEMBAHASAN

III.1. Hasil ETL Pipeline Airflow

Hasil dari setiap tahap proses ETL merupakan sebuah tabel yang telah memenuhi standar normalisasi 3NF dengan cara melibatkan proses - proses pada *task* DAG. Berikut merupakan hasil dari proses ETL tersebut.

1.1. Tabel Admission

	id [PK] integer	date_in date	date_out date	id_branch character var	id_patient character var	id_hospital_c character var	id_room character var	id_doctor character var	id_surgery character var	id_lab character var	id_drug character var	drug_qty integer	admin_cost integer	cogs integer	id_payment character var	id_review character var	call_date timestamp wi
1	18402	2020-05-13	2020-05-14	1	4242	2	2	1	1	4	9	3	50000	4356897	1	4	2024-06-0...
2	24391	2022-06-26	2022-06-26	1	7287	1	0	5	0	3	10	3	50000	1119919	1	3	2024-06-0...
3	55287	2020-10-04	2020-10-08	1	6279	2	2	2	1	5	10	1	50000	7696150	2	2	2024-06-0...
4	86202	2021-04-28	2021-04-28	2	6595	1	0	1	0	4	10	5	50000	2121559	1	3	2024-06-0...
5	80995	2021-10-25	2021-10-26	1	5967	2	3	5	0	4	10	2	50000	3273172	1	1	2024-06-0...
6	75430	2023-06-19	2023-06-20	1	5824	2	3	2	2	5	10	5	50000	14556442	2	2	2024-06-0...
7	25634	2020-03-18	2020-03-21	1	5211	2	1	4	1	5	10	5	50000	5294587	2	4	2024-06-0...
8	11856	2021-08-23	2021-08-23	1	1019	1	0	2	0	4	10	5	50000	4051191	1	3	2024-06-0...
9	22565	2021-05-11	2021-05-13	1	7524	2	3	3	2	4	10	1	50000	16589927	2	4	2024-06-0...
10	54695	2021-12-24	2021-12-27	1	7864	2	3	2	1	5	6	3	50000	6469716	2	4	2024-06-0...



1.2. Tabel Stock_Obat

	id [PK] integer	date_buy date	id_drug character varying (50)	quantity character varying (50)	id_branch character varying (10)	call_date timestamp without time zone
1	6	2020-01-07	4	40	1	2024-06-04 03:53:09.811737
2	8	2020-01-13	2	62	1	2024-06-04 03:53:09.815519
3	9	2020-01-17	1	48	1	2024-06-04 03:53:09.817779
4	10	2020-01-20	2	26	1	2024-06-04 03:53:09.819571
5	18	2020-02-13	3	28	1	2024-06-04 03:53:09.834109
6	19	2020-02-13	7	47	1	2024-06-04 03:53:09.836293
7	22	2020-02-20	12	65	1	2024-06-04 03:53:09.841244
8	23	2020-02-23	11	70	1	2024-06-04 03:53:09.842972
9	24	2020-02-24	11	55	1	2024-06-04 03:53:09.84443
10	26	2020-03-05	1	50	1	2024-06-04 03:53:09.847286



1.3. Tabel Reviews

	id [PK] integer	review character varying (50)
1	1	Sangat Tidak Puas
2	2	Tidak Puas
3	3	Sangat Puas
4	4	Puas
5	5	Netral



1.4. Tabel Payment

	id [PK] integer 	payment_type character varying (50) 
1	1	Pribadi
2	2	Asuransi



1.5. Tabel Doctor

	id [PK] integer 	name character varying (50) 
1	1	Kandungan
2	2	Bedah
3	3	Penyakit Dalam
4	4	Gigi
5	5	Umum

1.6. Tabel Lab

	id [PK] integer 	lab_name character varying (50) 
1	1	Hematologi
2	2	Urinalisa
3	3	Kimia Darah
4	4	Serologi
5	5	Rontgen

1.7. Tabel Surgery

	id [PK] integer 	surgery_type character varying (50) 
1	1	Kecil
2	2	Khusus
3	3	Besar

1.8. Tabel Branch

	id [PK] integer	branch_name character varying (10)
1	1	RSMA
2	2	RSMS
3	3	RSMD

1.9. Tabel Hospital Care

	id [PK] integer	hospital_care character varying (50)
1	1	Rawat Jalan
2	2	Rawat Inap

1.10. Tabel Patient

	id [PK] integer	patient_name character varying (50)	gender character varying (50)	age integer
1	1	Budi Sitompul, M.Ak	Laki-laki	57
2	2	Zelda Padmasari	Laki-laki	17
3	3	Ratna Uwais, S.Kom	Perempuan	42
4	4	Iriana Kurniawan, S.Ked	Perempuan	38
5	5	Ade Winarsih	Perempuan	30
6	6	Darijan Nasyiah	Perempuan	77
7	7	R.A. Dian Nuraini, S.Sos	Laki-laki	42
8	8	Drs. Michelle Sinaga, M.Farm	Laki-laki	59
9	9	Drs. Aris Hakim, S.Psi	Perempuan	75
10	10	Kayla Simbolon	Perempuan	27

1.11. Tabel Room Type

	id [PK] integer	room_type character varying (50)	food_price integer
1	1	Kelas 3	50000
2	2	Kelas 1	110000
3	3	VIP	150000
4	4	Kelas 2	80000

1.12. Tabel Drugs

	id [PK] integer	drug_name character varying (50)	id_category character varying (50)
1	12	Naproxen	1
2	2	Tramadol	1
3	1	Diclofenac	1
4	8	Holland & Barrett	2
5	7	Enervon-C	2
6	6	Blackmores	2
7	10	Azithromycin	3
8	9	Amoxicillin	3
9	4	Ciprofloxacin	3
10	11	Panadol	4
11	5	Calpol	4
12	3	Paramex	4

1.13. Tabel Drugs Type

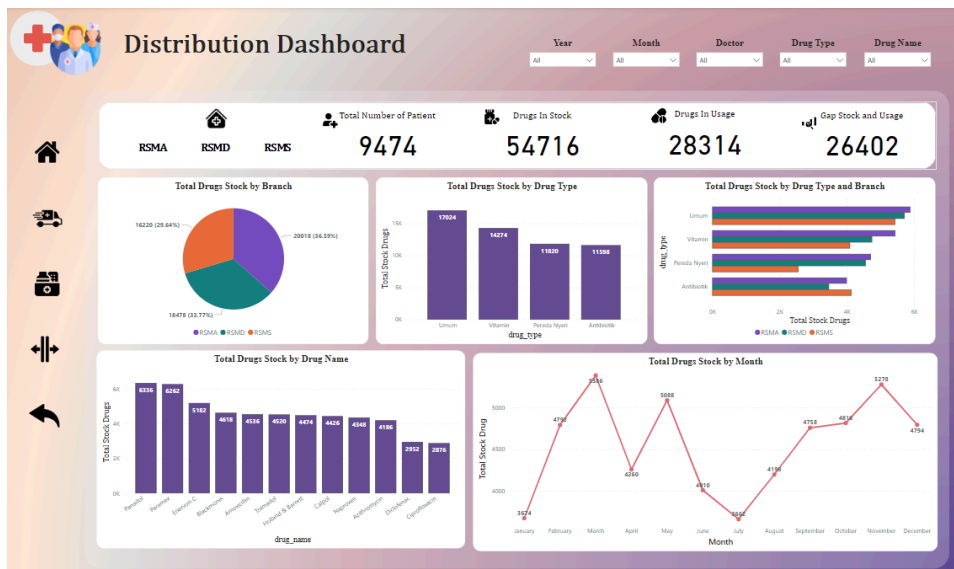
	id [PK] integer	category character varying (50)
1	1	Pereda Nyeri
2	2	Vitamin
3	3	Antibiotik
4	4	Umum

1.14. Tabel Logs

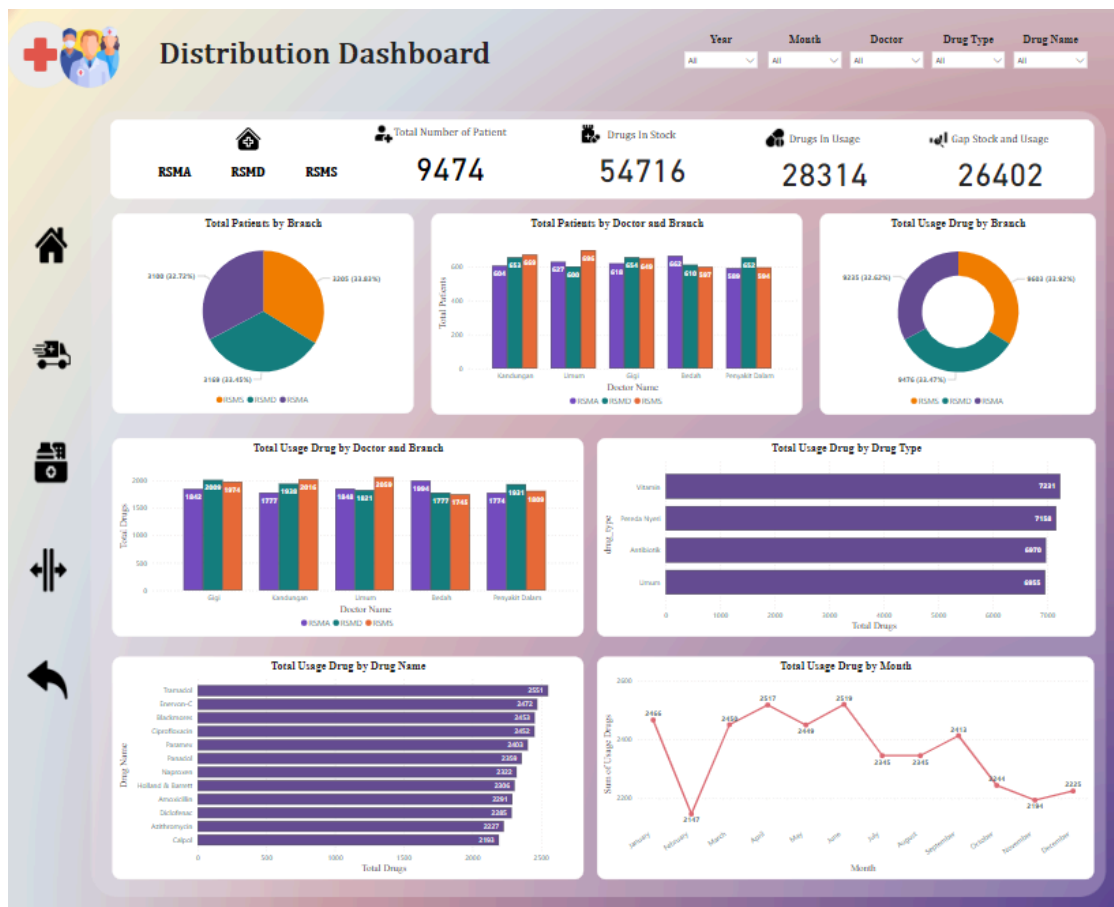
	id [PK] integer	call_date timestamp without time zone	error_message text
1	1	2024-06-04 03:54:29.006046	Success
2	2	2024-06-04 03:54:29.022587	Success
3	3	2024-06-04 03:54:29.023631	Success
4	4	2024-06-04 03:54:29.024442	column "drugs" of relation "drug_stock" does not exist
5	5	2024-06-04 03:54:29.025245	column "branch" of relation "admission" does not exist
6	6	2024-06-04 03:54:29.026075	'surgery'
7	7	2024-06-04 03:54:29.026882	relation "doctor" already exists
8	8	2024-06-04 03:54:29.027545	column "age" does not exist
9	9	2024-06-04 03:54:29.028354	relation "lab" already exists

III.2 Analisis Dashboard

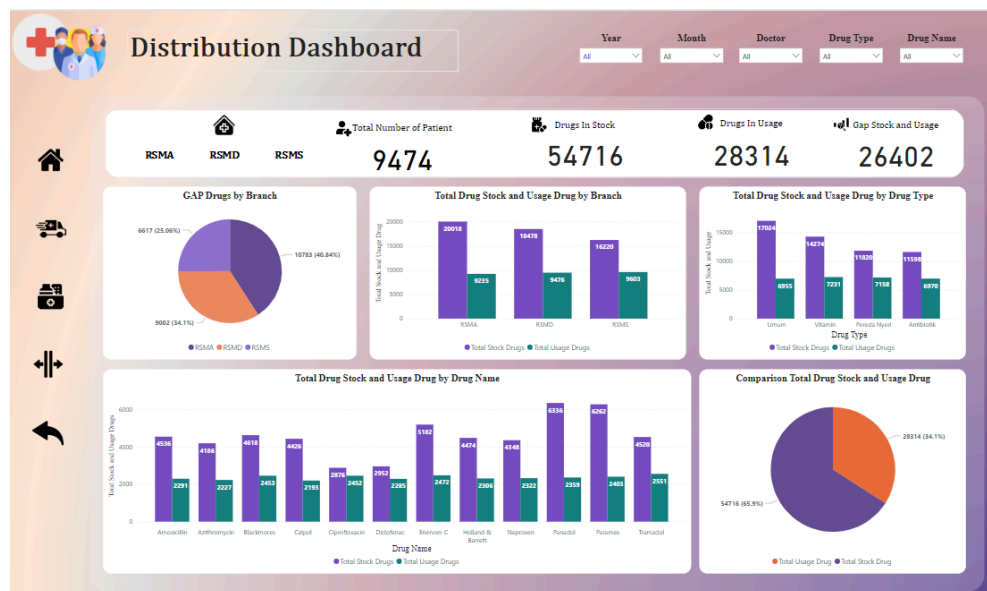
Visualisasi data dilakukan dengan menggunakan dashboard pada aplikasi Power BI. Dashboard kami terdiri dari 3 halaman yaitu *Drug Stock*, *Drug Usage*, dan *Gap Total Drug Stock and Drug Usage*. Halaman *Drug Stock* menampilkan informasi terkait dengan persediaan obat, lalu halaman *Usage Drug* menampilkan informasi terkait penggunaan obat, dan halaman *Gap Total Drug Stock and Drug Usage* menampilkan informasi terkait perbandingan persediaan dan penggunaan obat. Informasi yang ditampilkan pada tiap halaman mencakup total data dari ketiga cabang rumah sakit dan data untuk masing-masing cabang rumah sakit tersebut.



Gambar 3.1. Dashboard Drug Stock



Gambar 3.2. Dashboard Drug Usage



Gambar 3.3. Gap Total Stock Drug and Usage Drug

Hasil analisis dari *dashboard* adalah sebagai berikut:

1. Pada tahun 2020-2024 Rumah Sakit Mulia yang terbagi di tiga cabang yaitu RSMA, RSMD, dan RSMS memiliki persediaan obat sebanyak 54716 obat. Selama periode tersebut cabang RSMA memiliki persediaan sebanyak 20018 obat, cabang RSMD memiliki 18478 persediaan obat, dan RSMS memiliki 16220 persediaan obat. Dari total 54716 persediaan obat yang ada, total obat yang digunakan adalah sebanyak 28314 obat. Dengan rincian dari cabang RSMA sebanyak 9235 obat, cabang RSMD sebanyak 9476 obat, dan RSMS sebanyak 9603 obat.
2. Berdasarkan tipe obat yang ada, obat dengan tipe Umum adalah obat dengan persediaan terbanyak, sedangkan obat dengan tipe Antibiotik adalah obat dengan persediaan paling sedikit. Jika dilihat berdasarkan nama obat yang ada, obat dengan persediaan terbanyak adalah Panadol yang termasuk obat berkategori Umum. Untuk obat dengan persediaan paling sedikit adalah Ciprofloxacin yang termasuk obat berkategori Antibiotik.
3. Jika dilihat pada masing-masing cabang, tipe obat dengan persediaan paling banyak juga merupakan obat tipe Umum, dan tipe obat dengan persediaan paling sedikit merupakan tipe Antibiotik.
4. Jumlah persediaan obat terbanyak tiap bulannya terjadi pada bulan Maret 2023 dengan persediaan sebanyak 1976 obat
5. Jumlah persediaan obat dari ketiga cabang rumah sakit paling banyak terjadi pada tahun 2023 dengan jumlah 16578 obat, sedangkan jumlah persediaan obat paling sedikit terjadi pada tahun 2022 dengan jumlah 12158 obat. Namun penggunaan obat paling banyak justru tidak terjadi pada tahun 2023 yang mana hanya 6715 obat yang digunakan, penggunaan obat paling banyak terjadi pada tahun 2020 yaitu sebanyak 7582 obat dari persediaan sebanyak 12544 obat.
6. Dari jumlah persediaan dan penggunaan obat pada periode tahun 2020-2024, dapat dilihat bahwa total obat yang digunakan oleh rumah sakit kurang lebih hanya setengah dari persediaan yang ada, yaitu 28314 obat dari 54716 persediaan obat.
7. Pada cabang RSMA persediaan obat paling banyak terjadi pada tahun 2023 dengan jumlah 7274 obat, sedangkan jumlah persediaan obat paling sedikit terjadi pada tahun 2022 dengan jumlah 3704 obat.

8. Pada cabang RSMD persediaan obat paling banyak terjadi pada tahun 2023 dengan jumlah 4964 obat, sedangkan jumlah persediaan obat paling sedikit terjadi pada tahun 2021 dengan jumlah 4494 obat.
9. Pada cabang RSMS persediaan obat paling banyak terjadi pada tahun 2021 dengan jumlah 4786 obat, sedangkan jumlah persediaan obat paling sedikit terjadi pada tahun 2020 dengan jumlah 3160 obat.
10. Namun penggunaan obat paling banyak di masing-masing cabang rumah sakit justru juga tidak terjadi pada tahun yang paling banyak memiliki persediaan obat. Hal ini memperlihatkan bahwa adanya kesalahan dalam memprediksi jumlah kebutuhan obat sehingga banyak obat yang tidak terpakai.
11. Berdasarkan tren persediaan obat tiap bulannya, jumlah persediaan obat tertinggi terjadi pada bulan Mei dengan jumlah obat 5088 dan November dengan jumlah obat 5278. Persediaan obat terendah pada bulan Januari dengan jumlah obat 3674 dan bulan Juli dengan jumlah obat 3662.
12. Terdapat peningkatan signifikan persediaan obat dari Januari ke Maret diikuti penurunan pada bulan April. Tren ini berulang dengan persediaan obat kembali meningkat di bulan Mei namun menurun pada bulan Juli.
13. Berdasarkan tren penggunaan obat, jumlah penggunaan obat tertinggi terjadi pada bulan Juni dengan jumlah obat 2519 dan bulan April dengan jumlah obat 2517. Namun pada bulan Februari jumlah penggunaan obat sebanyak 2147 dan bulan November sebanyak 2194.
14. Penggunaan obat relatif stabil pada bulan Januari sampai Juni dengan sedikit peningkatan pada bulan April dan Juni. Namun bulan Juli sampai Desember, penggunaan obat cenderung menurun secara bertahap dan penurunan mencapai titik terendah pada bulan November dan sedikit meningkat di bulan Desember.
15. Persediaan obat tertinggi terjadi pada bulan Mei sebanyak 5088 dan bulan November sebanyak 5278, namun penggunaan obat tertinggi pada bulan April dan Juni, menunjukkan kemungkinan melebihi kebutuhan aktual
16. Jumlah pasien dari ketiga cabang Rumah Sakit Mulia adalah sebanyak 9474 pasien. Dengan rincian RSMA sebanyak 3100 pasien dengan persentase 32,72%, RSMD 3169 pasien dengan persentase 33,45 dan RSMS sebanyak 3205 pasien dengan persentase 33,83%. Dari ketiga

persentase cabang rumah sakit menunjukkan perbedaan persentase yang tidak terlalu signifikan. Namun persentase jumlah pasien tertinggi terdapat pada cabang RSMS.

17. Berdasarkan persentase jumlah pasien berdasarkan dokter yang dikunjungi di setiap cabang, bisa dilihat kebutuhan dokter yang paling dibutuhkan. Terlihat bahwa terdapat lima spesialis dokter yaitu dokter kandungan, umum, bedah, umum dan penyakit dalam. Dari lima spesialis dokter tersebut, terlihat cabang rumah sakit RSMS memiliki kebutuhan dokter tertinggi di tiga spesialis dokter yaitu dokter umum, kandungan dan gigi dengan jumlah kebutuhan dokter tertinggi di cabang ini adalah dokter umum dengan jumlah pasien sebanyak 696 pasien
18. Cabang rumah sakit RSMD memiliki kebutuhan dokter tertinggi di dua spesialis dokter yaitu dokter gigi dan penyakit dalam dengan jumlah kebutuhan dokter tertinggi di cabang ini adalah dokter penyakit dalam dengan jumlah pasien sebanyak 652 pasien.
19. Berdasarkan informasi jumlah pasien dengan kebutuhan masing masing dokter, dapat dilihat juga kebutuhan pasien tersebut terhadap penggunaan obat di masing masing cabang. Dari visualisasi penggunaan obat tiap cabang menunjukkan jumlah penggunaan obat di cabang RSMS tertinggi dengan jumlah penggunaan obat 9603 atau 33,92% kemudian cabang RSMD dengan jumlah penggunaan obat 9476 atau 33,47% dan terendah adalah cabang RSMA dengan jumlah penggunaan obat 9235 atau 33,62%.
20. Informasi ini selaras dengan jumlah pasien di tiap cabang tersebut. Informasi menunjukkan bahwa semakin banyak pasien pada rumah sakit tersebut maka semakin banyak juga penggunaan obat pada rumah sakit.
21. Berdasarkan informasi penggunaan obat di tiap cabang rumah sakit, dapat dilihat bahwa jumlah penggunaan obat dengan kebutuhan masing masing dokter. Dari lima spesialis dokter tersebut, cabang rumah sakit RSMS memiliki kebutuhan dokter tertinggi di dua spesialis dokter yaitu dokter umum dan kandungan namun kebutuhan dokter tertinggi pada cabang ini adalah dokter umum dengan jumlah penggunaan obat 2059
22. Cabang rumah sakit RSMD memiliki kebutuhan dokter tertinggi di dua spesialis dokter yaitu dokter gigi dan penyakit dalam dengan jumlah kebutuhan dokter tertinggi di cabang ini adalah dokter gigi dengan jumlah pasien sebanyak 2009
23. Cabang rumah sakit RSMA memiliki kebutuhan dokter tertinggi pada spesialis dokter yaitu dokter bedah dengan jumlah pasien sebanyak 1194.

24. Hasil distribusi dokter pada penggunaan obat menunjukkan informasi yang sama berdasarkan jumlah pasien.
25. Berdasarkan tipe obat yang ada pada tiga cabang rumah sakit, obat dengan tipe Vitamin adalah obat dengan penggunaan obat terbanyak, sedangkan obat dengan tipe Umum adalah obat dengan penggunaan paling sedikit. Jika dilihat berdasarkan nama obat yang ada, obat dengan penggunaan terbanyak adalah Tramol dan obat dengan persediaan paling sedikit adalah Calpol yang termasuk obat berkategori Umum.
26. Jika dilihat pada masing-masing cabang, tipe obat dengan persediaan paling banyak juga merupakan obat tipe Umum, dan tipe obat dengan persediaan paling sedikit merupakan tipe Antibiotik.
27. Berdasarkan informasi persediaan obat dan penggunaan obat dapat dilihat gap antara persediaan dan penggunaan obat tersebut. Visualisasi menunjukkan bahwa ada perbedaan yang signifikan antara persediaan obat dan penggunaan obat.
28. Analisis Gap dapat dilihat di tiap cabang rumah sakit dengan jumlah gap tertinggi terdapat pada cabang RSMA dengan persentase sebanyak 40,84% kemudian cabang RSMD dengan persentase sebanyak 34,41% dan yang terendah pada cabang RSMS dengan persentase sebanyak 25,06%
29. Dari hasil keseluruhan persentase gap untuk persediaan obat sebanyak 28314 atau 34,41% dan persentase gap penggunaan obat adalah 54716 atau 65,9%. Terdapat perbedaan yang signifikan antara persediaan obat dan penggunaan obat. Hal ini menyebabkan adanya penimbunan obat disebabkan persediaan obat yang tidak sesuai dengan kebutuhan aktual rumah sakit. Oleh karena itu, perlu dilakukan monitor persediaan obat dan keselarasan antara penggunaan obat setiap bulannya dengan melakukan prediksi penggunaan obat ke bulan berikutnya berdasarkan data historical jumlah penggunaan obat.

III.3. Analisis Script Python (Machine Learning)

3.1. Import Library

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import joblib
```

3.2. Data Loading

Load data Rumah Sakit untuk melihat stok penjualan

```
hd = pd.read_csv('/content/Hospital_data.csv')
```

Load data Obat untuk melihat stok pembelian

```
dd = pd.read_csv('/content/Drugs_data.csv')
```

3.3. Feature Engineering

3.3.1 Data preprocessing

Ambil kolom yang relevan dari Data Rumah Sakit agar sama dengan kolom dari Data Obat

```
hd1 = hd[['Date OUT', 'Drug Brands', 'Drug Qty', 'Branch']]
```

Menyesuaikan penamaan kolom untuk seluruh tabel

```
hd1.rename(columns={'Date OUT': 'Date', 'Drug Qty': 'OUT Qty'}, inplace=True)
```

```
dd.rename(columns={'Drugs': 'Drug Brands', 'Qty': 'IN Qty'}, inplace=True)
```

Menggabungkan seluruh tabel

```
mdf = pd.merge(dd, hd1, on=['Date', 'Drug Brands', 'Branch'], how='outer')
```

Menangani nilai yang hilang

```
mdf.fillna(0, inplace=True)
```

Penanganan nilai yang hilang dilakukan dengan angka 0 saja, hal ini dikarenakan beberapa kombinasi kolom Tanggal, Merk Obat, dan Cabang tidak mempunyai nilai data stok pembelian/penjualan.

Tambahkan kolom untuk melihat nilai selisih antara stok pembelian dan stok penjualan

```
mdf['Adjusted Qty'] = mdf['IN Qty'] - mdf['OUT Qty']
```

Ubah tipe data pada kolom 'Date' menjadi datetime dan urutkan dari yang paling awal

```
mdf['Date'] = pd.to_datetime(mdf['Date'])
```

```
data = mdf.sort_values(by='Date')
```

Kelompokkan kolom 'Date' berdasarkan bulan

```
mmdf = data.groupby([data['Date'].dt.to_period('M'), 'Drug Brands', 'Branch']).agg({'IN Qty':  
'sum', 'OUT Qty': 'sum', 'Adjusted Qty': 'sum'}).reset_index()
```

Tambahkan kolom baru berdasarkan bulan dan tahun dari kolom 'Date'

```
mmdf['year'] = mmdf.Date.dt.year
```

```
mmdf['month'] = mmdf.Date.dt.month
```

Duplikat data untuk menghapus kolom 'Date'

```
smmdf = mmdf.copy()
```

```
smmdf.drop('Date', axis=1, inplace=True)
```

Data duplikat dibuat untuk memfasilitasi pemisahan data menjadi set pelatihan dan pengujian. Pembagian ini memungkinkan kami mengisolasi fitur untuk melatih model dan mencadangkan fitur tambahan untuk mengevaluasi performanya. Alasan pemisahan ini adalah selama pelatihan model, transformasi data tertentu, seperti konversi kolom tanggal dan waktu, diterapkan. Akibatnya, saat mengevaluasi model, perbandingan langsung dengan data waktu dan tanggal menjadi tidak praktis karena transformasi yang diterapkan selama pelatihan.

Hitung rata-rata kuantitas stok penjualan bulanan untuk setiap kombinasi Merek Obat, Cabang, dan Bulan

```
smmdf['monthly_avg'] = smmdf.groupby(['Drug Brands', 'Branch', 'month'])['OUT Qty'].transform('mean')
```

3.3.2 Encoding

Buat encoding untuk kolom kategorikal, yaitu kolom 'Drug Brands' dan 'Branch'

```
label_encoder_drugs = LabelEncoder()
label_encoder_branch = LabelEncoder()

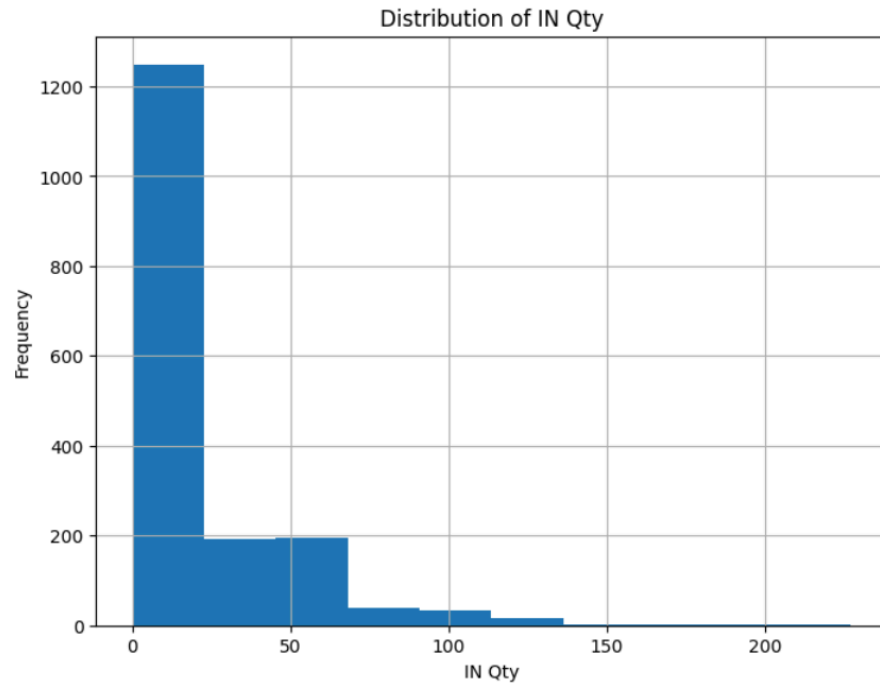
smmdf['Drug Brands'] = label_encoder_drugs.fit_transform(smmdf['Drug Brands'])
smmdf['Branch'] = label_encoder_branch.fit_transform(smmdf['Branch'])
```

3.3.3 Scaling

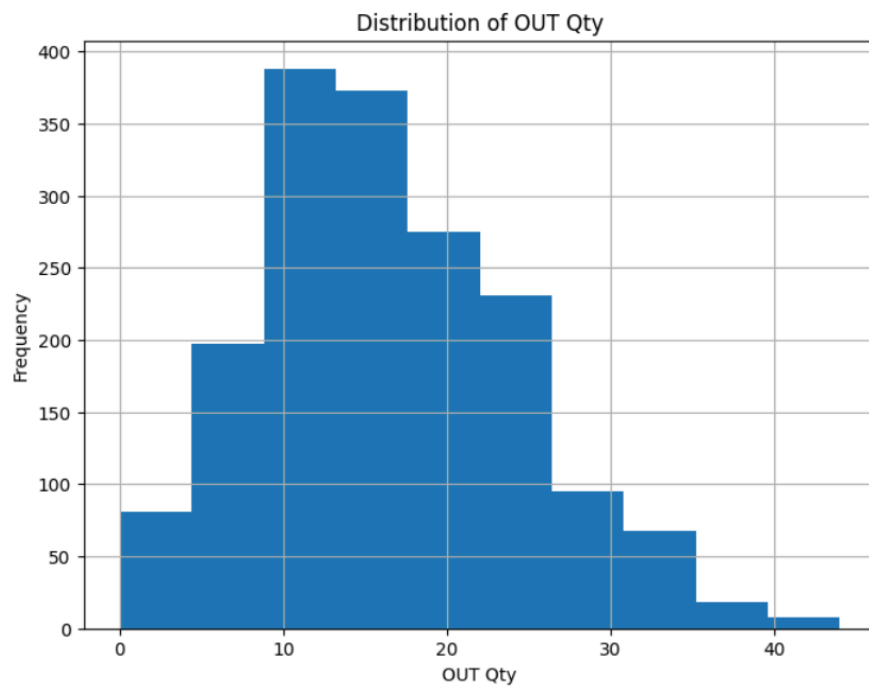
Visualisasikan distribusi data di setiap kolom numerik

```
columns_to_exclude = ['Drug Brands', 'Branch', 'year', 'month']

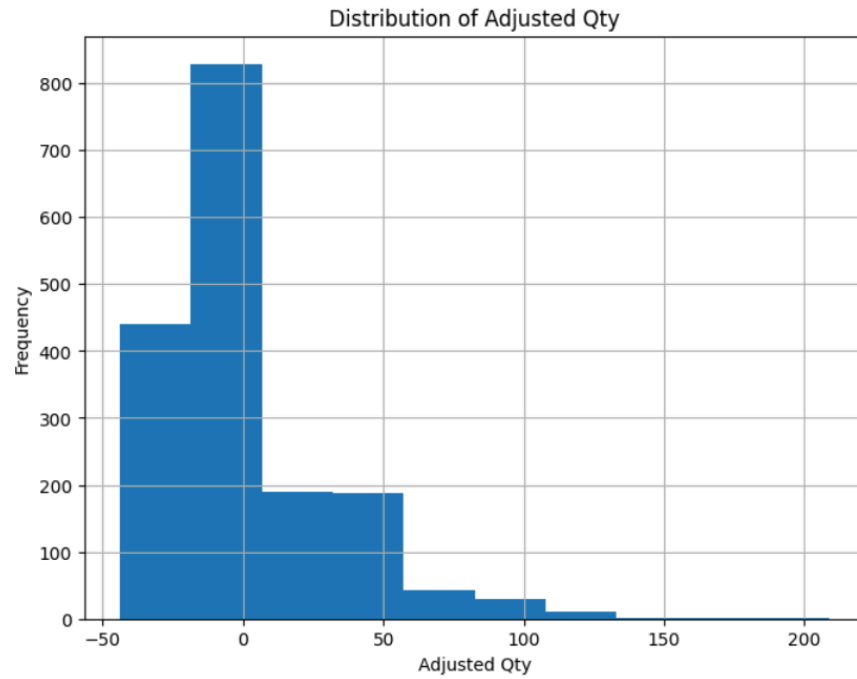
for column in smmdf.columns:
    if column not in columns_to_exclude and smmdf[column].dtype in ['int64', 'float64']:
        plt.figure(figsize=(8, 6))
        smmdf[column].hist()
        plt.title(f'Distribution of {column}')
        plt.xlabel(column)
        plt.ylabel('Frequency')
        plt.show()
```



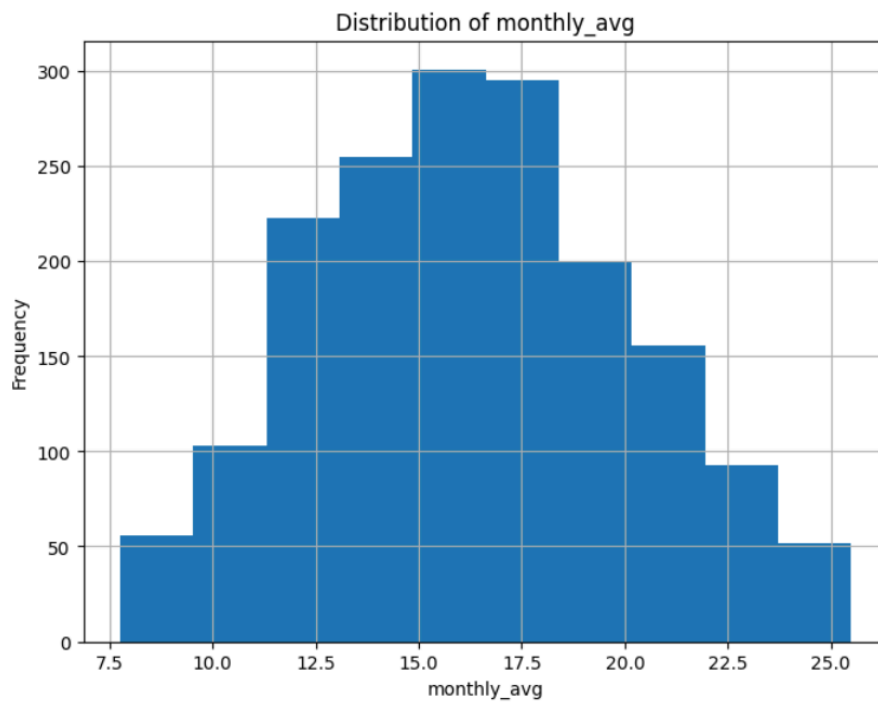
Gambar 3.1. Distribusi IN Qty



Gambar 3.2. Distribusi OUT Qty



Gambar 3.3. Distribusi Adjusted Qty



Gambar 3.4. Distribusi Monthly Average

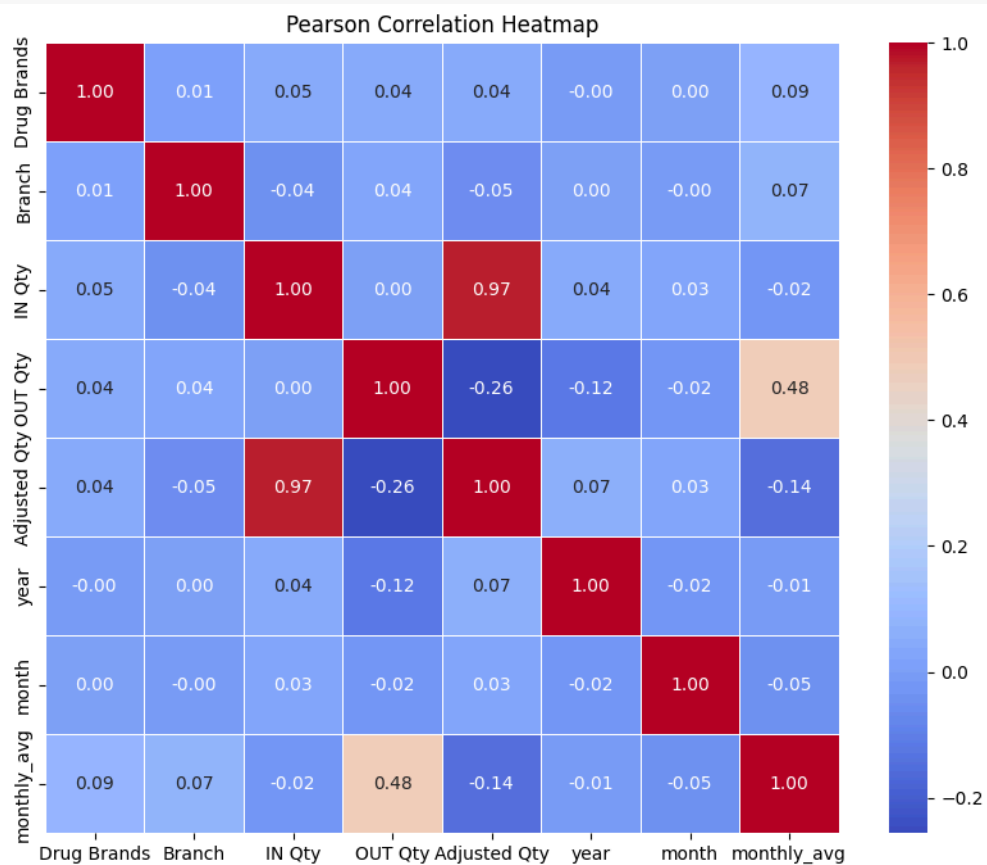
Scaling tidak dilakukan karena nilai datanya tidak besar.

3.3.4 Feature Selection

Visualisasikan korelasi antar kolom

```
df_encoded = pd.get_dummies(smmddf)
corr = df_encoded.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Pearson Correlation Heatmap')
plt.show()
```



Gambar 3.5. Heatmap Correlation

Periksa multikolinearitas di seluruh kolom

```
def calc_vif(nilai):
```

```

vif = pd.DataFrame()
vif["variables"] = nilai.columns
vif["VIF"] = [variance_inflation_factor(nilai.values, i) for i in range(nilai.shape[1])]
vif = vif.sort_values(by=["VIF"],ascending=False)
return(vif)

```

```
calc_vif(smmddf)
```

Tabel 3.1. Nilai VIF antar Variable

	variables	VIF
2	IN Qty	inf
3	OUT Qty	inf
4	Adjusted Qty	inf
5	year	26.229161
7	monthly_avg	25.707747
6	month	4.471970
0	Drug Brands	3.579131
1	Branch	2.514886

Catatan : VIF (*Variance Inflation Factor*) digunakan dalam analisis regresi untuk mengevaluasi multikolinearitas, yaitu ketika dua atau lebih variabel independen dalam model regresi mempunyai hubungan yang kuat satu sama lain.

Hapus kolom yang tidak relevan

```

for df in [smmddf]:
    df.drop(['IN Qty',
            'year'],
            axis=1,
            inplace=True)

```

Berdasarkan nilai korelasi dan nilai VIF, kolom yang tidak digunakan adalah kolom IN Qty dan kolom year karena mempunyai nilai korelasi yang mendekati 0 dan mempunyai nilai VIF yang dapat dikatakan tinggi.

3.3.5 Data Splitting

Train Test Split untuk fitur tambahan yang mengevaluasi kinerjanya

```
X_train, X_testt, y_train, y_testt = train_test_split(mmdf.drop('OUT Qty',axis=1),mmdf.pop('OUT Qty'), random_state=123, test_size=0.2)
X_testt['Date'] = X_testt['Date'].dt.to_timestamp().dt.to_period('M').astype(str)
```

Train Test Split untuk melatih model

```
X_train, X_test, y_train, y_test = train_test_split(smmdf.drop('OUT Qty',axis=1),smmdf.pop('OUT Qty'), random_state=123, test_size=0.2)
X_traino = X_train.apply(pd.to_numeric, errors='coerce')
X_testo = X_test.apply(pd.to_numeric, errors='coerce')
y_traino = y_train.apply(pd.to_numeric, errors='coerce')
y_testo = y_test.apply(pd.to_numeric, errors='coerce')
```

3.4. Model Training

Inisialisasi dan latih empat model regresi yang berbeda:

Model Linear Regression

```
model_lr = LinearRegression()
model_lr.fit(X_traino, y_traino)
```

Model Random Forest Regressor

```
model_rfr = RandomForestRegressor()
model_rfr.fit(X_traino, y_traino)
```

Model Support Vector Regressor

```
model_svr = SVR()
model_svr.fit(X_traino, y_traino)
```

Model XGBoost Regressor

```
model_xgb = XGBRegressor()
model_xgb.fit(X_traino, y_traino)
```

3.5. Model Evaluation

Periksa nilai cross val untuk semua model

```
models = {
    'Linear Regression': model_lr,
    'Random Forest Regressor': model_rfr,
    'SVR': model_svr,
    'XGBRegressor': model_xgb
}

datacv = {'Model': [], 'Mean Score': [], 'Std Score': []}

for name, model in models.items():
    scores = cross_val_score(model, X_traino, y_traino, cv=10)
    datacv['Model'].append(name)
    datacv['Mean Score'].append(scores.mean())
    datacv['Std Score'].append(scores.std())

pd.DataFrame(datacv)
```

	Model	Mean Score	Std Score
0	Linear Regression	0.236381	0.063976
1	Random Forest Regressor	0.698302	0.096544
2	SVR	0.602801	0.104527
3	XGBRegressor	0.636141	0.129198

Periksa nilai evaluasi metrik untuk semua model

```
results = []
```

```
def evaluate_model(model, X_train, y_train, X_test, y_test):  
    train_metrics = {}  
    test_metrics = {}  
  
    y_train_pred = model.predict(X_train)  
    train_metrics['MAE'] = mean_absolute_error(y_train, y_train_pred)  
    train_metrics['RMSE'] = mean_squared_error(y_train, y_train_pred, squared=False)  
    train_metrics['R2-Score'] = r2_score(y_train, y_train_pred)  
  
    y_test_pred = model.predict(X_test)  
    test_metrics['MAE'] = mean_absolute_error(y_test, y_test_pred)  
    test_metrics['RMSE'] = mean_squared_error(y_test, y_test_pred, squared=False)  
    test_metrics['R2-Score'] = r2_score(y_test, y_test_pred)  
  
    return train_metrics, test_metrics  
  
for name, model in models.items():  
    train_metrics, test_metrics = evaluate_model(model, X_traino, y_traino, X_testo, y_testo)  
    results.append({  
        'Model': name,  
        'Train MAE': train_metrics['MAE'],  
        'Train RMSE': train_metrics['RMSE'],  
        'Train R2-Score': train_metrics['R2-Score'],  
        'Test MAE': test_metrics['MAE'],  
        'Test RMSE': test_metrics['RMSE'],  
        'Test R2-Score': test_metrics['R2-Score']  
    })  
  
pd.DataFrame(results)
```


	Model	Train MAE	Train RMSE	Train R2-Score	Test MAE	Test RMSE	Test R2-Score
0	Linear Regression	5.405825	6.774449	0.253698	5.049443	6.433510	0.334894
1	Random Forest Regressor	0.651228	1.593329	0.958716	1.768504	4.008307	0.741823
2	SVR	2.264850	4.850034	0.617478	2.287509	4.598663	0.660173
3	XGBRegressor	0.141300	0.317165	0.998364	2.138281	4.544230	0.668170

Dari hasil Cross Val dan Evaluation Metrics, model Random Forest Regressor (RFR) dipilih karena mempunyai nilai Mean Score yang tinggi, nilai Std Score yang rendah, nilai MAE/RMSE yang rendah, nilai R2-Score yang tinggi, dan kesenjangan yang cukup rendah dalam nilai prediksi antara pelatihan dan pengujian.

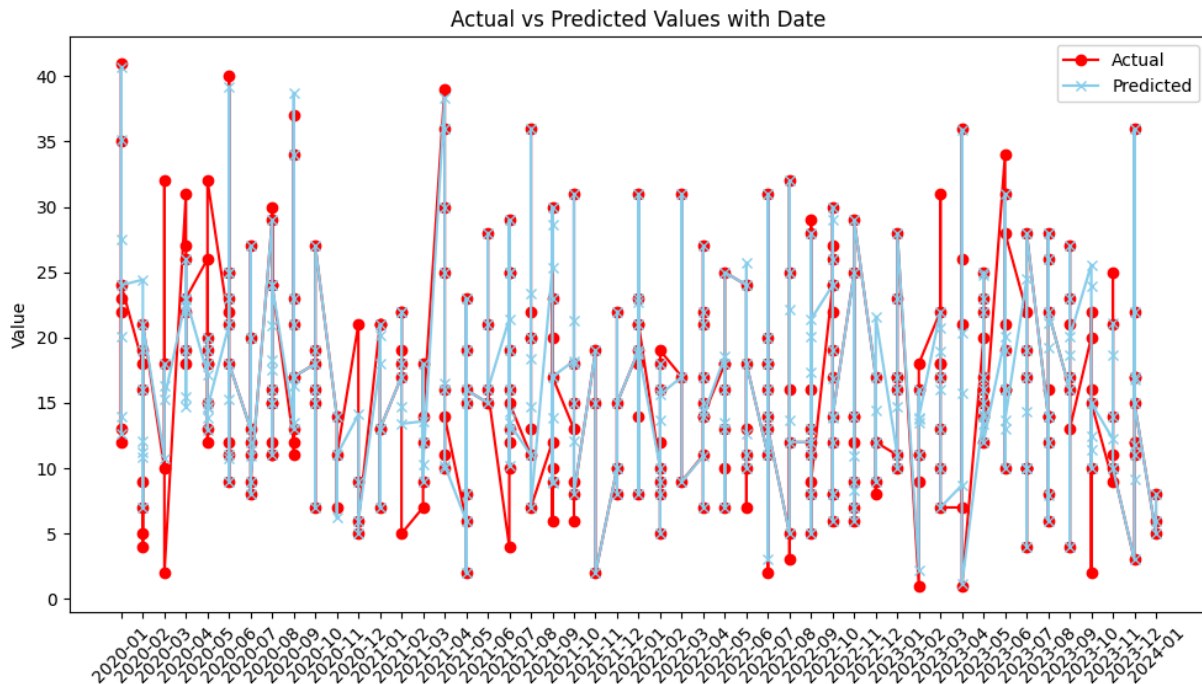
Menampilkan grafik hasil perbandingan antara nilai sebenarnya dengan nilai prediksi

```

results = pd.DataFrame({'Date': X_testt['Date'], 'Actual': y_testo, 'Predicted':
model_rfr.predict(X_testo)})
results.sort_values(by='Date', inplace=True)

plt.figure(figsize=(10, 6))
plt.plot(results['Date'], results['Actual'], label='Actual', marker='o', color='red')
plt.plot(results['Date'], results['Predicted'], label='Predicted', marker='x', color='skyblue')
plt.title('Actual vs Predicted Values with Date')
plt.xlabel('Date')
plt.ylabel('Value')
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.show()

```



3.6. Model Tuning

Tuning dilakukan dengan harapan dapat meningkatkan performa model

Lakukan penyetelan hyperparameter

```
param_grid = {
    'n_estimators': [i for i in range(100, 500, 100)],
    'max_depth': [i for i in range(10, 50, 10)],
    'min_samples_split': [i for i in range(10, 50, 10)],
    'min_samples_leaf': [i for i in range(1, 10, 2)]
}

grid_search = GridSearchCV(estimator=model_rfr, param_grid=param_grid,
                           cv=5, scoring='neg_mean_squared_error', n_jobs=-1)

grid_search.fit(X_traino, y_traino)

best_params = grid_search.best_params_
print("Best Parameters:", best_params)
```

```
best_rfr = grid_search.best_estimator_
```

Best Parameters: {'max_depth': 40, 'min_samples_leaf': 3, 'min_samples_split': 40, 'n_estimators': 100}

Periksa perbandingan nilai evaluasi metrik antara sebelum dan sesudah penyetelan model

```
metrics = {  
    "Train Non Tuning": {  
        "MAE": mean_absolute_error(y_traino, model_rfr.predict(X_traino)),  
        "RMSE": mean_squared_error(y_traino, model_rfr.predict(X_traino), squared=False),  
        "R2-Score": r2_score(y_traino, model_rfr.predict(X_traino))  
    },  
    "Train Tuning": {  
        "MAE": mean_absolute_error(y_traino, best_rfr.predict(X_traino)),  
        "RMSE": mean_squared_error(y_traino, best_rfr.predict(X_traino), squared=False),  
        "R2-Score": r2_score(y_traino, best_rfr.predict(X_traino))  
    },  
    "Test Non Tuning": {  
        "MAE": mean_absolute_error(y_testo, model_rfr.predict(X_testo)),  
        "RMSE": mean_squared_error(y_testo, model_rfr.predict(X_testo), squared=False),  
        "R2-Score": r2_score(y_testo, model_rfr.predict(X_testo))  
    },  
    "Test Tuning": {  
        "MAE": mean_absolute_error(y_testo, best_rfr.predict(X_testo)),  
        "RMSE": mean_squared_error(y_testo, best_rfr.predict(X_testo), squared=False),  
        "R2-Score": r2_score(y_testo, best_rfr.predict(X_testo))  
    }  
}  
  
t_results = pd.DataFrame(metrics)
```

t_results

	Train Non Tuning	Train Tuning	Test Non Tuning	Test Tuning
MAE	0.651228	1.586724	1.768504	1.777221
RMSE	1.593329	3.466933	4.008307	3.686037
R2-Score	0.958716	0.804540	0.741823	0.781669

Setelah menyempurnakan model, kami mengamati bahwa matrik evaluasi untuk data pelatihan pada awalnya memburuk. Namun, perbedaan antara prediksi pada data pelatihan dan pengujian berkurang secara signifikan. Hal ini menunjukkan bahwa model Random Forest Regressor (RFR) awal sudah overfit, namun penyetelannya menghasilkan kecocokan yang lebih baik.

Karena pengamatan ini, kami memilih model Random Forest Regressor (RFR) yang telah disesuaikan untuk diterapkan.

3.7. Model Saving

```
joblib.dump(label_encoder_drugs, 'led.pkl')  
joblib.dump(label_encoder_branch, 'leb.pkl')  
joblib.dump(best_rfr, 'rfr.pkl')
```

BAB IV

KESIMPULAN

Penelitian terkait analisis dan prediksi stok obat pada seluruh cabang rumah sakit telah berhasil dilakukan. Bersamaan hasil penelitian tersebut, kesimpulan yang dapat ditarik diantaranya sebagai berikut.

1. Pada tahun 2020-2024 Rumah Sakit Mulia yang terbagi di tiga cabang yaitu RSMA, RSMD, dan RSMS memiliki persediaan obat sebanyak 54716 obat. Selama periode tersebut cabang RSMA memiliki persediaan obat paling banyak yaitu 20018 obat, diikuti cabang RSMD yaitu 18478 persediaan obat, dan yang paling sedikit adalah cabang RSMS yaitu 16220 persediaan obat.
2. Dari hasil keseluruhan, terdapat perbedaan yang signifikan antara persediaan obat dan penggunaan obat. Hal ini menyebabkan adanya penimbunan obat disebabkan persediaan obat yang tidak sesuai dengan kebutuhan aktual rumah sakit. Persentase gap persediaan obat sebanyak 28314 atau 34,41% dan persentase gap penggunaan obat adalah 54716 atau 65,9%. Oleh karena itu, perlu dilakukan monitor persediaan obat dan keselarasan antara penggunaan obat setiap bulannya dengan melakukan prediksi penggunaan obat ke bulan berikutnya berdasarkan data historical jumlah penggunaan obat.
3. Kebutuhan dokter di setiap rumah sakit bervariasi berdasarkan spesialisasi dan jumlah pasien yang dilayani. Di cabang rumah sakit RSMS, terdapat kebutuhan tertinggi untuk dokter kandungan dengan penggunaan obat sebanyak 2016, diikuti oleh dokter umum. Cabang RSMD menunjukkan kebutuhan tertinggi untuk dokter penyakit dalam dengan jumlah pasien sebanyak 2009, serta dokter gigi. Sedangkan di cabang RSMA, kebutuhan tertinggi terdapat pada dokter bedah dengan jumlah pasien sebanyak 1194. Distribusi dokter berdasarkan penggunaan obat sejalan dengan jumlah pasien yang dilayani, menunjukkan kebutuhan spesifik masing-masing cabang rumah sakit terhadap dokter spesialis tertentu. Kesimpulan ini mengindikasikan bahwa alokasi dokter spesialis perlu disesuaikan dengan kebutuhan yang teridentifikasi di setiap cabang untuk memastikan pelayanan kesehatan yang optimal.
4. Hasil *modelling* menunjukkan bahwa algoritma *Random Forest Regressor* mampu melakukan prediksi kuantitas obat di suatu cabang rumah sakit. Dengan nilai *train*

R2-score sebesar 0.80 dan *test R2-score* sebesar 0.78. Selain itu, nilai metrik *error train* dengan menggunakan *MAE* dan *RMSE* berturut - turut sebesar 1.58 dan 3.46 sedangkan untuk nilai *error test* dengan menggunakan *MAE* dan *RMSE* berturut - turut sebesar 1.77 dan 3.68. Dengan begitu, algoritma *Random Forest Regressor* terbukti efektif dan memiliki tingkat keakuratan yang tinggi dalam memprediksi kuantitas obat di suatu cabang rumah sakit.

BAB V DAFTAR PUSTAKA

- [1]. Yadav, Dinesh. "Towards Data Science - Categorical encoding using Label-Encoding and One-Hot-Encoder"
- [2]. H. Smith, J. Doe, and A. Brown, "Correlation Analysis to Identify the Effective Data in Machine Learning: Prediction of Depressive Disorder and Emotion States," *IEEE Transactions on Computational Social Systems*, vol. 7, no. 2, pp. 215-224, April 2021.
- [3]. J. I. Daoud, "Multicollinearity and Regression Analysis," *Journal of Engineering and Science*, vol. 5, no. 3, pp. 112-119, Sept. 2020.
- [4]. Graw, J. H., Wood, W. T., & Phrampus, B. J. (2021). Predicting global marine sediment density using the random forest regressor machine learning algorithm. *Journal of Geophysical Research: Solid Earth*, 126(1), e2020JB020135
- [5]. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer.
- [6]. Nurani, Alfida Tegar, Adi Setiawan, and Bambang Susanto. "Perbandingan Kinerja Regresi Decision Tree dan Regresi Linear Berganda untuk Prediksi BMI pada Dataset Asthma." *Jurnal Sains dan Edukasi Sains* 6.1 (2023): 34-43.
- [7]. Anggrawan, Anthony, Hairani Hairani, and Nurul Azmi. "Prediksi Penjualan Produk Unilever Menggunakan Metode Regresi Linear." *Jurnal Bumigora Information Technology (BITe)* 4.2 (2022): 123-132.

LAMPIRAN

Link Repository Project :

<https://github.com/Bithealth-x-Hacktiv8/final-project-team3-group-3/tree/main>

Link Dashboard Drive :

https://drive.google.com/file/d/1WO5dw69ngz-3gsvgvXQV-JQsAdbth5Lj/view?usp=drive_link

Link Dashboard Power BI :

<https://app.powerbi.com/reportEmbed?reportId=0d10d86c-9358-4478-96d4-e6226e1ec226&auth=true&ctid=42d20f55-f787-4741-ace9-b0153477840c>

Link Deployment :

<https://bithealth-data-group-3.streamlit.app/>