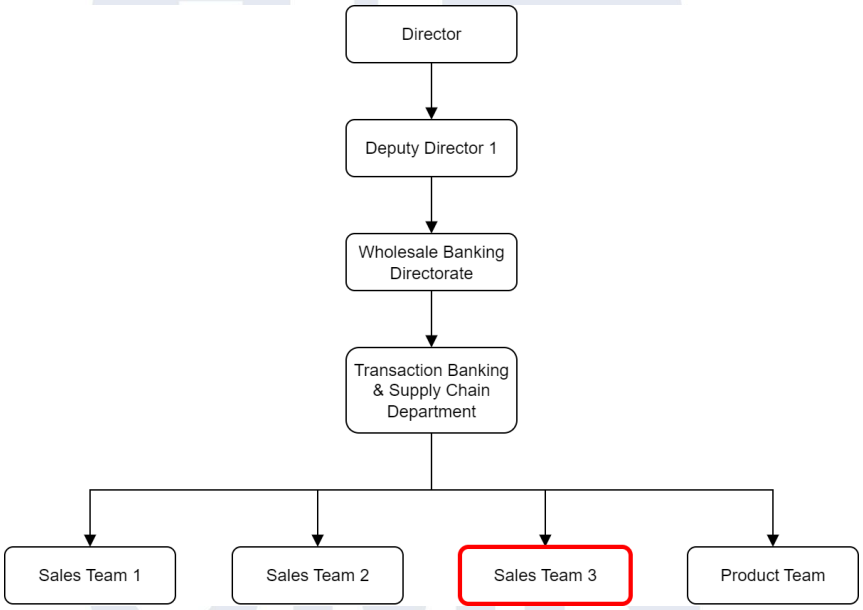
BAB III
PELAKSANAAN KERJA

3.1 Kedudukan dan Koordinasi

3.1.1 Kedudukan

Gambar 3.1 menunjukkan bagan departemen *Transaction Banking & Supply Chain* (TBSC). Departemen TBSC terletak di bawah direktorat *Wholesale Banking* (WB) pada struktur organisasi. Departemen ini dibagi menjadi 4 tim yaitu 3 tim untuk kegiatan *sales* dan 1 tim untuk mengelola produk terkait *transaction banking & supply chain*.

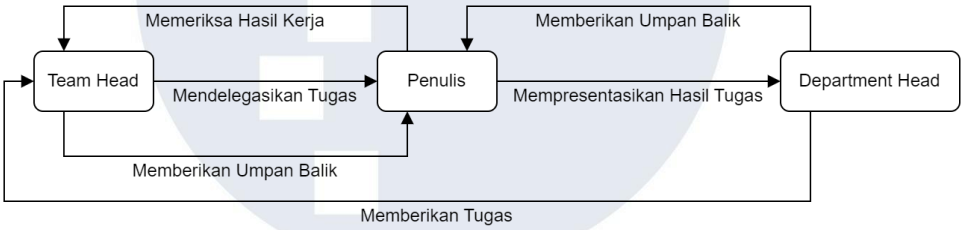


Gambar 3.1 Bagan Departemen *Transaction Banking & Supply Chain* (TBSC)

Selama program magang, mahasiswa diletakkan pada tim *sales* 3 yang antara lain memiliki tanggung jawab untuk melakukan kegiatan *sales* pada nasabah berjenis *non-bank financial institution* dan sekuritas. Tim *sales* 3 juga memiliki tanggung jawab melaksanakan proyek terkait data yang akan digunakan untuk membantu mengukur kinerja departemen TBSC dan mendukung inisiatif berbasis data untuk TBSC.

3.1.2 Koordinasi

Gambar 3.2 menunjukkan bagan alur koordinasi yang diikuti selama kerja praktik magang ini. Tugas diterima dari *department head* oleh *team head*, kemudian tugas didelegasikan kepada mahasiswa untuk dikerjakan. Ketika tugas sudah selesai dikerjakan, maka hasil kerja ditunjukkan kepada *team head* untuk diperiksa dan diberikan umpan balik, kemudian mahasiswa merevisi hasil kerja agar sesuai umpan balik tersebut. Jika hasil kerja sudah memuaskan, maka mahasiswa bersama dengan *team head* mempresentasikan hasil tersebut kepada *department head*. *Department head* kemudian memberikan umpan balik dan mahasiswa merevisi lagi agar hasil kerja sesuai sesuai dengan umpan balik tersebut.



Gambar 3.2 Bagan Alur Koordinasi

3.2 Tugas yang Dilakukan

Tabel 3.1 menunjukkan detail proyek atau pekerjaan yang dilakukan selama kerja praktik magang ini.

Tabel 3.1 Detail Pekerjaan yang Dilakukan

No.	Periode	Proyek	Keterangan
1	25 Agustus 2025 – 12 September 2025	*Data Scraping* Laporan Keuangan	Menggunakan Python dengan *libraries* seperti Selenium dan Pandas untuk navigasi dan *scraping* data dari situs web IDX secara otomatis dan menyusun data finansial perusahaan.
2	25 Agustus 2025 – 24 Oktober 2025	*Dashboard* Segmentasi Nasabah Berdasarkan Jumlah Transaksi	Menggunakan Python untuk memproses data transaksi nasabah dan melakukan segmentasi berdasarkan jumlah transaksi dan Microsoft Power BI untuk membangun *dashboard* monitoring segmentasi nasabah.
3	15 September 2025 – 30 September 2025	*Dashboard* Ekosistem Nasabah	Menggunakan Python untuk menambahkan hubungan *anchor-spoke* antar nasabah dan Microsoft Excel untuk membangun *dashboard* monitoring ekosistem.

4	1 Oktober 2025 – 17 Oktober 2025	Otomatisasi Pengisian Formulir	Menggunakan Python dengan *libraries* seperti Threading dan Pywin32 untuk menambahkan konsep *multithreading* dan interaksi otomatis dengan file formulir. Pyinstaller untuk konversi *script* menjadi file *executable*.
5	20 Oktober 2025 – 28 November 2025	Pengembangan *Dashboard* Kinerja TBSC	Menggunakan Python untuk mengolah data limit fasilitas nasabah dan Microsoft Power BI untuk menambahkan *dashboard* monitoring limit. Beralih ke *engine* python-calamine dan jenis file Parquet untuk meningkatkan efisiensi pengolahan dan penyimpanan data.
6	27 Oktober 2025 – 30 Oktober 2025	Proyek *Geomapping* Nasabah	Menggunakan Microsoft Excel dan Google Maps untuk mengkompilasi koordinat numerik lokasi entitas, lalu menggunakan Microsoft Power BI untuk membangun *dashboard* visualisasi perbandingan lokasi.

3.3 Uraian Pelaksanaan Kerja

3.3.1 Proses Pelaksanaan

3.3.1.1 Proyek *Data Scraping* Laporan Keuangan

Dalam industri perbankan, akses terhadap informasi yang terbaru dan akurat menjadi kemampuan yang sangat penting. Tim *TB Sales* perlu mengevaluasi laporan keuangan calon nasabah dan nasabah yang sudah ada untuk mendapatkan pengetahuan mengenai stabilitas finansial perusahaan dan tren perubahan keadaan finansial, serta likuiditas perusahaan. Pengetahuan ini memungkinkan tim *TB Sales* untuk menawarkan produk dan layanan *transaction banking* yang lebih sesuai dengan kebutuhan calon nasabah. Bagi nasabah yang sudah ada, pengetahuan ini juga memungkinkan *TB Sales* untuk menemukan kesempatan *cross-selling* baru untuk memaksimalkan pendapatan bank. Data keuangan perusahaan ini masih dikumpulkan secara manual, sehingga menghabiskan banyak waktu untuk mencari dan mengunduh data yang tepat, serta mengkompilasi data tersebut dalam satu tempat yang mudah dibaca.

Salah satu cara untuk mendapatkan laporan keuangan bagi perusahaan terbuka adalah melalui situs web Indonesia Stock Exchange (IDX). Namun, situs web IDX tidak memiliki fitur untuk mengunduh laporan keuangan secara *bulk*, sehingga perlu diunduh satu demi satu. Proses ini memakan banyak waktu, mengingat bahwa ada sekian banyak data perusahaan terbuka di situs web IDX dan laporan keuangan dibagi per kuartal dan tahun. Proyek ini bertujuan untuk mengotomatisasi proses pengunduhan laporan keuangan IDX dan pengekstraksian data dari laporan keuangan menjadi bentuk yang mudah dan cepat untuk dibaca manusia, yaitu dalam bentuk satu file Microsoft Excel yang mengandung semua data keuangan perusahaan terbuka. Gambar 3.3 menunjukkan alur pengerjaan proyek ini.

Gambar 3.3 Alur Pengerjaan Proyek 3.3.1.1

Untuk melaksanakan proyek ini, digunakan perangkat lunak sebagai berikut:

1. Microsoft Edge sebagai *web browser* untuk mengunjungi dan berinteraksi dengan situs web IDX.
2. Python dengan *library* utama Selenium untuk mengotomatisasi interaksi *web browser* dan proses *scraping* data.
3. Microsoft Edge WebDriver yang menjadi perantara Microsoft Edge dengan Python untuk memungkinkan kendali otomatis melalui Selenium.

```
import os, re, random, math
import pandas as pd
import pymupdf as pmp
from tqdm.notebook import tqdm

from fake_useragent import UserAgent
from selenium import webdriver
from selenium.webdriver.edge.options import Options
from selenium.webdriver.edge.service import Service
from selenium.webdriver.support.ui import Select
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
```

Gambar 3.4 *Library* yang diimpor untuk proyek 3.3.1.1

Gambar 3.4 menunjukkan semua *library* yang diimpor untuk memfasilitasi *scraping* data yang akan dilakukan. Beberapa *library* utama yang dipakai adalah pandas, pymupdf, os, dan Selenium.

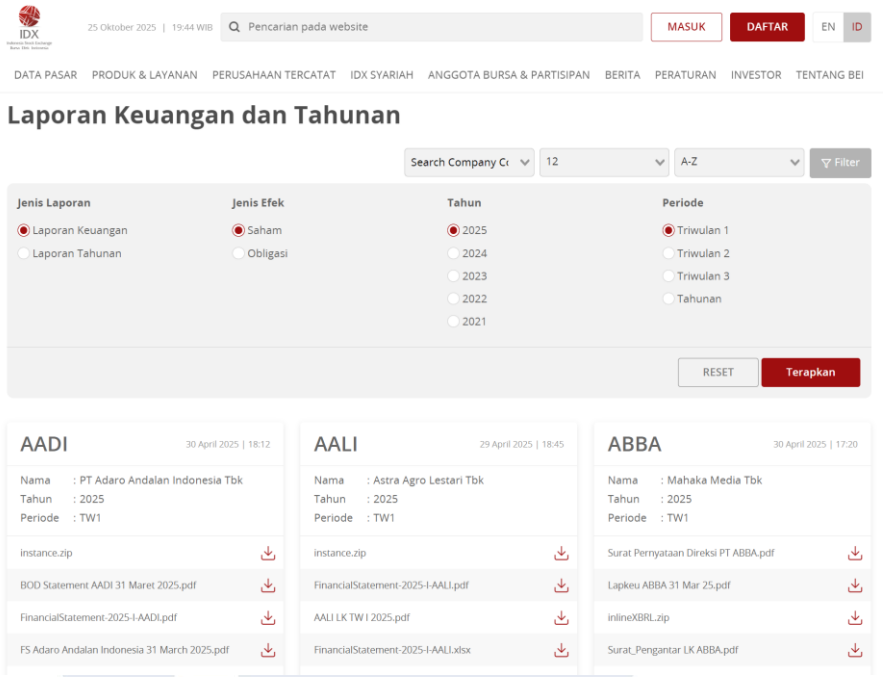
```
UA = UserAgent(browsers=['Edge','Chrome','Firefox','Opera','Safari','Google']),
               os=['Windows','Mac OS X','Linux','Ubuntu','Chrome OS'],
               platforms='desktop').random

service = Service("msedgedriver.exe")
options = Options()
options.add_argument("--headless")
options.add_argument("--disable-gpu")
options.add_argument(f'--user-agent={UA}')

prefs = {
    "download.default_directory": os.path.abspath("PDFs"),
    "download.prompt_for_download": False,
    "profile.default_content_setting_values.automatic_downloads": 1
}
options.add_experimental_option("prefs", prefs)
```

Gambar 3.5 Kode pengaturan *webdriver*

Gambar 3.5 menunjukkan kode pengaturan *webdriver*. Kode ini menetapkan *user agent* untuk *web browser*, yaitu identitas yang dilaporkan oleh *browser* saat mengunjungi sebuah situs web. Beberapa *user agent* digunakan untuk menyamarkan *web browser* sebagai pengunjung yang berbeda agar menghindari pendeteksian dan pemblokiran robot. Kode ini juga menetapkan beberapa argumen untuk *web browser* seperti menyalakan mode *headless* dan mematikan penggunaan GPU untuk mengurangi penggunaan sumber daya sistem ketika pengunduhan file sedang berjalan. Terkait dengan pengunduhan file, kode ini juga menetapkan folder yang menjadi lokasi penyimpanan semua unduhan file.



Gambar 3.6 Situs web laporan keuangan dan tahunan IDX

Untuk melakukan pengunduhan file secara otomatis dari situs web IDX, maka perlu diketahui terlebih dahulu bentuk situs web IDX. Gambar 3.6 menunjukkan tampilan situs web laporan keuangan dan tahunan IDX. Beberapa elemen yang penting diperhatikan di sini adalah kotak filter yang memungkinkan pengguna untuk memilih tahun dan periode laporan keuangan yang ingin dilihat. Adapun data setiap perusahaan terbuka yang terdaftar di IDX disajikan dalam kotak-kotak tersendiri. Daftar perusahaan dibagi menjadi beberapa halaman karena banyak perusahaan terbuka yang terdaftar di IDX, sehingga pada saat mengunduh file laporan keuangan maka perlu menavigasi antar halaman. File laporan keuangan tersedia dalam dua bentuk yaitu file Microsoft Excel dan file PDF. Format yang terstandar adalah file PDF sehingga lebih praktis untuk kebutuhan *scraping* data, sehingga pada saat proses pengunduhan juga harus memperhatikan ekstensi file agar file yang diunduh berbentuk PDF, bukan Microsoft Excel.

```python
# Year 0 is newest year, year 4 is oldest year
# Period 0 is Q1, period 3 is Q4

arr_year = ["year0", "year1", "year2", "year3", "year4"]
arr_period = ["period0", "period1", "period2", "period3"]
```

Gambar 3.7 Kode variabel pilihan tahun dan periode

Situs web IDX menyediakan data perusahaan terbuka dari 5 tahun terakhir dan setiap tahun dibagi menjadi per kuartal atau triwulan. Gambar 3.7 menunjukkan kode yang menetapkan variabel tahun dan periode dari file laporan keuangan yang ingin diunduh. Tahun dinamakan “year0” sampai dengan “year4”, di mana angka paling kecil adalah tahun paling baru. Periode dinamakan “period0” sampai dengan “period3”, di mana angka paling kecil adalah kuartal 1 dan angka paling besar adalah kuartal 4.

```python
driver = webdriver.Edge(service=service, options=options)
wait = WebDriverWait(driver, 5)

driver.get("https://www.idx.co.id/id/perusahaan-tercatat/laporan-keuangan-dan-tahunan")
time.sleep(1.5)

for year in arr_year:
    for period in arr_period:
        try:
            yearradio = wait.until(EC.presence_of_element_located((By.ID, year)))
            driver.execute_script("arguments[0].click();", yearradio)
            time.sleep(0.5)

            periodradio = wait.until(EC.presence_of_element_located((By.ID, period)))
            driver.execute_script("arguments[0].click();", periodradio)
            time.sleep(0.5)

            applybutton = wait.until(EC.element_to_be_clickable((By.XPATH, "//button[contains(text(), 'Terapkan')]")))
            driver.execute_script("arguments[0].click();", applybutton)
            time.sleep(1.5)

            try:
                pageselection = driver.find_element(By.XPATH, "//ul[@class='pagination']//select")
                pages = pageselection.find_elements(By.TAG_NAME, "option")
                totalpages = len(pages)
            except:
                totalpages = 1

            for page in range(1, totalpages + 1):
                if page > 1:
                    pageselection = driver.find_element(By.XPATH, "//ul[@class='pagination']//select")
                    driver.execute_script("arguments[0].value = arguments[1]; arguments[0].dispatchEvent(new Event('chan
                    time.sleep(0.5)

                pdflinks = driver.find_elements(By.XPATH, "//a[contains(@href, '.pdf') and contains(@href, 'FinancialSta

                for link in pdflinks:
                    file_url = link.get_attribute("href")
                    driver.execute_script("arguments[0].click();", link)
                    print(f"Downloading {file_url}")
                    time.sleep(0.5)

            filterbutton = wait.until(EC.element_to_be_clickable((By.XPATH, "//button[contains(text(), 'Filter')]")))
            filterbutton.click()
            time.sleep(0.5)

        except Exception as e:
            print(f"Error for {year} and {period}: {e}")

driver.quit()
```

Gambar 3.8 Kode pengunduhan file laporan keuangan

Gambar 3.8 menunjukkan kode pengunduhan file laporan keuangan menggunakan *webdriver*. Pertama, *browser* mengunjungi pranala situs web laporan keuangan dan tahunan IDX. Lalu, untuk

setiap tahun dan setiap periode yang ditentukan dalam variabel pada Gambar 3.7, akan dipilih tombol filter yang sesuai untuk mendapatkan laporan keuangan periode dan tahun itu. Jumlah halaman kemudian diperiksa dan disimpan untuk melakukan *looping*. Setiap *loop* mencari file laporan keuangan untuk setiap perusahaan yang terdaftar dengan mencari teks “FinancialStatement” dan berakhir dengan ekstensi file “.pdf”. Setiap file yang ditemukan akan diunduh berurutan dengan interval antar file sebesar 0.5 detik untuk memastikan semua file diunduh dengan benar. Ketika semua file laporan keuangan dalam satu halaman sudah diunduh, maka akan lanjut ke halaman berikutnya sampai halaman terakhir. Ketika mencapai halaman terakhir, maka filter akan diubah lagi ke periode dan tahun selanjutnya yang ditentukan. Jika semua periode dan tahun telah dilewatkan, maka *webdriver* akan ditutup.

[4220000] Statement of financial position presented using order of liquidity - Financial and Sharia Industry

Laporan posisi keuangan			Statement of financial position
	31 December 2024	31 December 2023	
Laporan posisi keuangan			**Statement of financial position**
Aset			**Assets**
Kas	1,379,647	1,428,683	Cash
Dana yang dibatasi penggunaannya			Restricted funds
Giro pada Bank Indonesia	9,443,461	9,276,598	Current accounts with Bank Indonesia
Giro pada bank lain			**Current accounts with other banks**
Giro pada bank lain pihak ketiga	509,209	500,326	Current accounts with other banks third parties
Giro pada bank lain pihak berelasi	525,792	462,061	Current accounts with other banks related parties
Cadangan kerugian penurunan nilai pada giro pada bank lain	(37)	(158)	Allowance for impairment losses for current accounts with other bank
Penempatan pada Bank Indonesia dan bank lain			**Placements with Bank Indonesia and other banks**
Penempatan pada Bank Indonesia dan bank lain pihak ketiga	12,694,705	8,961,654	Placements with Bank Indonesia and other banks third parties
Penempatan pada Bank Indonesia dan bank lain pihak berelasi			Placements with Bank Indonesia and other banks related parties
Cadangan kerugian penurunan nilai pada penempatan pada bank lain	(582)	(0)	Allowance for impairment losses for placements with other banks
Piutang asuransi			**Insurance receivables**
Piutang asuransi pihak ketiga			Insurance receivables third parties
Piutang asuransi pihak berelasi			Insurance receivables related parties
Cadangan kerugian penurunan nilai pada piutang asuransi			Allowance for impairment losses for insurance receivables
Biaya akuisisi tangguhan			Deferred acquisition costs
Deposito pada lembaga kliring dan penjaminan			Deposits to clearing and settlement guarantee institution
Efek-efek yang diperdagangkan			**Marketable securities**
Efek-efek yang diperdagangkan pihak ketiga	27,981,308	17,038,501	Marketable securities third parties
Efek-efek yang diperdagangkan pihak berelasi		0	Marketable securities related parties
Cadangan kerugian penurunan nilai pada	(593)	(799)	Allowance for impairment losses for marketable

Gambar 3.9 Sampel konten laporan keuangan, bagian aset

Gambar 3.9 menunjukkan sampel konten dari salah satu laporan keuangan yang diunduh. Dapat dilihat bahwa laporan keuangan ini menggunakan dua bahasa yaitu Bahasa Indonesia (konten terletak di sisi kiri halaman) dan Bahasa Inggris (konten terletak di sisi kanan halaman). Data terbaru yang sesuai periode dan tahun yang dilaporkan terletak di sisi Bahasa Indonesia, sedangkan data periode sebelumnya dipaparkan pada sisi Bahasa Inggris. Secara keseluruhan, data disajikan dalam bentuk semacam tabel di mana satu

kolom berisi metrik yang diukur dan satu kolom lagi berisi angka atau informasi yang sesuai untuk metrik tersebut.

```
FIELDS = {
    "Entity Name": r"Nama entitas\s+(.*)",
    "Entity Code": r"Kode entitas\s+([A-Z]{4})",
    "Sector": r"\bSektor\s+([A-Z]\.?\s+[\s\S]*?)(?=\s+Sector\b)",
    "Subsector": r"\bSubsektor\s+([A-Z]\d{1,2}\.\s+[\s\S]*?)(?=\s+Subsector\b)",
    "Industry": r"\bIndustri\s+(?!utama)([A-Z]\d{1,2}\.\s+[\s\S]*?)(?=\s+Industry\b)",
    "Subindustry": r"\bSubindustri\s+([A-Z]\d{3}\.\s+[\s\S]*?)(?=\s+Subindustry\b)",
    "Assets": r"\bJumlah[\s\n]+aset[\s\n]+([\(\)]?[\d,]+[\(\)]?)",
    "Equities": r"\bJumlah[\s\n]+ekuitas[\s\n]+([\(\)]?[\d,]+[\(\)]?)",
    "Other Payables Related Party": r"\bUtang[\s\n]+lainnya[\s\n]+pihak[\s\n]+berelasi[\s\n]+([\(\)]?[\d,]+[\(\)]?)",
    "Sales and Revenue": r"\bPenjualan[\s\n]+dan[\s\n]+pendapatan[\s\n]+usaha[\s\n]+([\(\)]?[\d,]+[\(\)]?)",
    "Rounding": r"Pembulatan\s+yang\s+digunakan\s+dalam\s+penyajian\s+jumlah\s+dalam\s+laporan\s+keuangan\s*:?[\s\n]+([
}
```

Gambar 3.10 Kode penentuan data yang di-*scrape* dan polanya

Usai mengunduh semua file laporan keuangan yang diinginkan, maka langkah selanjutnya adalah melakukan *scraping* data dari konten setiap laporan tersebut. Gambar 3.10 menunjukkan kode penentuan data yang ingin di-*scrape* dan pola dari data tersebut berdasarkan observasi pada Gambar 3.9. Pola data dibantu dengan *regular expression* untuk memastikan data diangkat dari konten laporan dengan benar. Untuk kebutuhan proyek ini, data yang diambil melibatkan informasi dasar perusahaan seperti kode entitas serta sektor dan industri di mana perusahaan tersebut beroperasi. Data numerik yang diambil adalah jumlah aset, ekuitas, utang lainnya kepada pihak berelasi, penjualan dan pendapatan usaha, dan pembulatan angka yang digunakan dalam laporan.

```
ROUNDING_MAP = {
    "Satuan Penuh / Full Amount": 1,
    "Ribuan / In Thousand": 1000,
    "Jutaan / In Million": 1000000,
    "Miliaran / In Billion": 1000000000
}
```

Gambar 3.11 Kode *mapping* pembulatan angka

Untuk memuat angka yang lebih besar, laporan keuangan IDX terkadang dibulatkan dalam ribuan, jutaan, atau miliaran, sehingga pembulatan ini perlu diketahui untuk mendapatkan angka penuh. Gambar 3.11 menunjukkan kode *mapping* pembulatan angka yang mengasosiasi teks pembulatan dengan angka yang sesuai.

```python
def extract_text_from_pdf(file_path):
    text = ""
    with pmp.open(file_path) as doc:
        for page in doc:
            text += page.get_text() + "\n"
    return text
```

Gambar 3.12 Kode fungsi *scraping* teks dari file PDF

Gambar 3.12 menunjukkan kode fungsi untuk *scraping* seluruh teks dari file laporan keuangan. Kode ini menggunakan *library* PyMuPDF untuk membaca file PDF.

```python
def extract_fields_from_text(text):
    result = {}
    for field, pattern in FIELDS.items():
        match = re.search(pattern, text, re.IGNORECASE)
        if match:
            val = match.group(1).strip()

            # Collapse new lines and multiple spaces into single space
            if field in ["Sector", "Subsector", "Industry", "Subindustry"] and isinstance(val, str):
                val = re.sub(r"\s+", " ", val).strip()

            # Handle numeric values
            if any(c.isdigit() for c in val):
                val = val.replace(",", "")  # Remove comma separators

                # Convert negative numbers with ()surround to -prefix
                if val.startswith("(") and val.endswith(")"):
                    val = "-" + val[1:-1]

                # Make sure dashes are normal dashes
                val = val.strip().replace("–", "-").replace("−", "-")

                try:
                    val = int(val)
                    # Force COGS to be negative if it's not
                    if field == "Cost of Sales and Revenue" and val > 0:
                        val = -val
                except ValueError:
                    pass

            result[field] = val
        else:
            result[field] = None
    return result
```

Gambar 3.13 Kode fungsi *scraping* pola data yang ditentukan dari file PDF

Gambar 3.13 menunjukkan kode fungsi untuk *scraping* berdasarkan pola data yang telah ditentukan pada Gambar 3.10. Selain menemukan pola data yang ditentukan, kode ini juga berfungsi untuk membersihkan data seperti menghapus spasi yang duplikat dan separator koma dari angka. Pada laporan keuangan, angka yang negatif direpresentasikan dengan sebuah pasangan simbol dalam kurung dan kode ini menggantinya menjadi satu simbol minus di depan agar Python dapat memahami polaritas angkanya. Khusus untuk angka beban pokok penjualan dan pendapatan usaha yang selalu negatif karena merupakan biaya bagi perusahaan, maka kode ini memastikan angka tersebut selalu dibaca dengan polaritas negatif.

```python
def extract_year_quarter_from_filename(filename):
    match = re.match(r"FinancialStatement-(\d{4})-([A-Z]+)-([A-Z]{4})\.pdf", filename, re.IGNORECASE)
    if match:
        year = match.group(1)
        quarter_raw = match.group(2).upper()
        quarter_map = {
            "I": "Q1",
            "II": "Q2",
            "III": "Q3",
            "IV": "Q4",
            "TAHUNAN": "Q4"
        }
        quarter = quarter_map.get(quarter_raw, quarter_raw)
        return year, quarter
    return None, None
```

Gambar 3.14 Kode fungsi pengambilan tahun dan periode dari nama file

Gambar 3.14 menunjukkan kode fungsi untuk mengambil tahun dan periode laporan keuangan dari nama file. Nama setiap file sudah standar sehingga dapat digunakan *regular expression* untuk mengekstraksi tahun dan periode secara cepat.

```python
def process_pdf(file_path):
    text = extract_text_from_pdf(file_path)
    return extract_fields_from_text(text)
```

Gambar 3.15 Kode fungsi pemrosesan file PDF

Gambar 3.15 menunjukkan kode fungsi untuk memproses satu file PDF yang menggunakan dua fungsi yang telah dibuat sebelumnya yaitu extract_text_from_pdf() dan extract_fields_from_text().

```python
pdf_files = os.listdir("PDFs")
industry_records = []
long_records = []

for filename in tqdm(pdf_files, desc="Scraping PDF files", unit=" PDF", ncols=900):
    if filename.lower().endswith(".pdf"):
        full_path = os.path.join("PDFs", filename)
        year, quarter = extract_year_quarter_from_filename(filename)
        data = process_pdf(full_path)

        rounding_str = str(data.get("Rounding", "")).strip()
        multiplier = ROUNDING_MAP.get(rounding_str, 1)

        # Add to industry info
        industry_records.append({
            "Entity Code": data.get("Entity Code"),
            "Industry": data.get("Industry"),
            "Subindustry": data.get("Subindustry"),
            "Sector": data.get("Sector"),
            "Subsector": data.get("Subsector"),
            "Year": year
        })

        # Add metric rows
        for key, value in data.items():
            if key in ["Entity Code", "Entity Name", "Industry", "Subindustry", "Sector", "Subsector", "Rounding"]:
                continue

            if isinstance(value, int):
                value *= multiplier

            long_records.append({
                "Entity Code": data.get("Entity Code"),
                "Entity Name": data.get("Entity Name"),
                "Year": year,
                "Quarter": quarter,
                "Metric": key,
                "Value": value,
                "Source File": filename
            })

# Keep latest industry/subindustry per entity
df_industry = pd.DataFrame(industry_records)
df_industry = df_industry.sort_values(["Entity Code", "Year"], ascending=[True, False])
df_industry = df_industry.drop_duplicates(subset=["Entity Code"], keep="first")[["Entity Code", "Sector", "Subsector", "

# Merge into main long-form metrics dataset
df_metrics = pd.DataFrame(long_records)
df_final = df_metrics.merge(df_industry, on="Entity Code", how="left")
```

Gambar 3.16 Kode eksekusi *scraping* data dari file PDF dan pembuatan *dataframe*

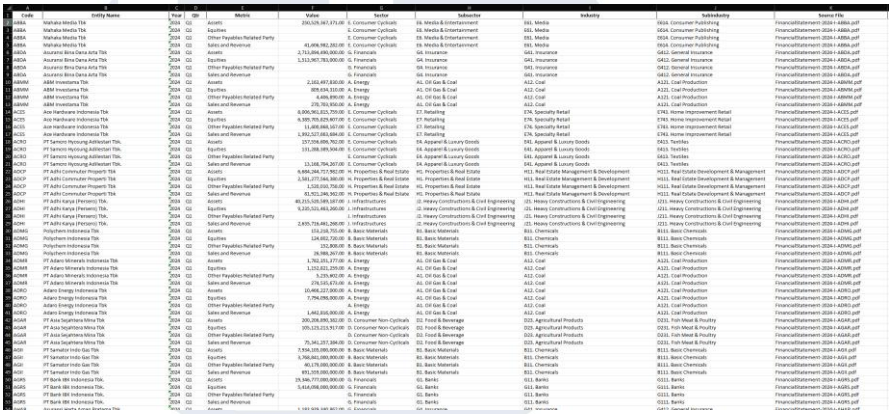
Gambar 3.16 menunjukkan kode pengkeksekusian *scraping* data dari semua file PDF yang telah diunduh dari situs web IDX. Untuk setiap file PDF, hal pertama yang dilakukan adalah mengekstraksi tahun dan periode dari file tersebut. Data kemudian diekstraksi dari konten laporan menggunakan fungsi process_pdf(). Pembulatan angka kemudian diperiksa dan angka yang sesuai diambil dari *mapping* yang sudah dibuat. Data dasar perusahaan seperti kode entitas dan industri disimpan dalam bentuk *dictionary* dan ditambahkan pada *array* industry_records, sedangkan data yang dapat berubah seperti angka disimpan dalam bentuk *dictionary* dan ditambahkan pada *array* long_records. Jika data yang diambil berbentuk angka, maka akan dikalikan dengan angka pembulatan untuk mendapatkan angka aslinya. Jika semua file PDF sudah diproses, maka akan dibangun *dataframe* yang berisi data perusahaan yang unik (tanpa data duplikat). Data lainnya kemudian ditambahkan

menggunakan operasi pd.DataFrame.merge() untuk menghasilkan *dataframe* final.

```
df_final.to_excel("IDX Scrape.xlsx", index=False)
```

Gambar 3.17 Kode output *dataframe* hasil *scraping* data ke file Microsoft Excel

Setelah *dataframe* final telah terbentuk, maka hasilnya dapat dioutput. Gambar 3.17 menunjukkan kode output *dataframe* final ke file Microsoft Excel.



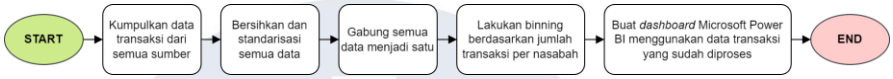
Gambar 3.18 Sampel file Microsoft Excel yang dihasilkan dari *scraping*

Gambar 3.18 menunjukkan sampel dari file Microsoft Excel yang dihasilkan dari output Python setelah melakukan *scraping* pada semua file PDF laporan keuangan yang diunduh dari situs web IDX. Sesuai dengan tujuan proyek, file ini mengandung data keuangan dari nasabah perusahaan terbuka dalam satu tempat yang mudah untuk diakses dan dibaca.

3.3.1.2 Proyek *Dashboard* Segmentasi Nasabah Berdasarkan Jumlah Transaksi

Kantor pusat SMBC telah mengidentifikasi pola di mana nasabah dengan jumlah transaksi tertentu per bulan cenderung menghasilkan lebih banyak keuntungan bagi bank daripada nasabah yang di bawah jumlah transaksi tersebut per bulan. SMBCI ditugaskan untuk memeriksa jika adanya pola yang sama dengan nasabah yang beroperasi di Indonesia.

Untuk mendukung inisiatif ini, proyek ini bertujuan untuk melakukan segmentasi nasabah berdasarkan jumlah transaksi yang mereka lakukan per bulan dan mengungkapkan pola dan tren yang muncul dari segmentasi tersebut melalui sebuah *dashboard*. Gambar 3.19 menunjukkan alur pengerjaan proyek ini.



Gambar 3.19 Alur pengerjaan proyek 3.3.1.2

```
import pandas as pd
pd.set_option('future.no_silent_downcasting', True)

import numpy as np
import re, os, calendar
from datetime import datetime
from rapidfuzz import process, fuzz
```

Gambar 3.20 *Library* yang dipakai untuk proyek 3.3.1.2

Gambar 3.20 menunjukkan *library* yang diimpor untuk melaksanakan proyek ini seperti pandas, numpy, dan rapidfuzz, serta re untuk *regular expression*.

```
def process_domesticin(file):
    print(f"Processing {file}")
    df = pd.read_excel(file, dtype={"Cust_CIF":str}, engine="calamine") # Read file

    # Strip column names and column values
    df.columns = df.columns.str.strip()
    df[df.select_dtypes(include="object").columns] = df.select_dtypes(include="object").apply(lambda col: col.str.strip(

    # Drop columns
    df.drop(["Payer_Name", "XBank_Code","XBank_Name","XBank_Country"], axis=1, inplace=True)

    # Rename columns
    df.rename(columns={"DomCode_In":"Code",
                       "FX_Currency":"Ccy",
                       "Cust_CIF":"Cust. No"},
              inplace=True)

    # Convert datatypes and change date to period
    df["Cust. No"] = df["Cust. No"].astype(str)
    df["Date"] = pd.to_datetime(df["Date"])
    df["Date"] = df["Date"] + pd.offsets.MonthEnd(0)

    # Add missing/descriptor columns
    df["Type"] = "Domestic"
    df["Direction"] = "In"
    df["Source"] = "SMAR&TS"

    return df

df_domesticin = pd.concat([process_domesticin(file) for file in files_domesticin], ignore_index=True)
df_domesticin.head(3)
```

Gambar 3.21 Kode fungsi pemrosesan data transaksi domestik

Gambar 3.21 menunjukkan kode fungsi yang dibuat untuk memproses data transaksi domestik, yaitu transaksi yang dilakukan melalui SKN atau RTGS, dalam platform *internet banking* SMAR&TS. Hal pertama yang dilakukan adalah membaca file

berbentuk Microsoft Excel dan dipastikan bahwa kolom nomor nasabah dibaca sebagai *string* karena merupakan *identifier* nasabah. Nama dari setiap kolom kemudian dibersihkan dengan fungsi strip() untuk memastikan tidak ada spasi atau karakter lain yang tersembunyi. Strip() juga dilakukan kepada semua nilai dari semua kolom yang berisi objek *string*. Ada beberapa kolom yang dihapus karena tidak relevan dengan analisis yang ingin dilakukan, dan ada juga beberapa kolom yang di-*rename* untuk standarisasi. Semua tanggal dalam data dibuat menjadi akhir bulan karena analisis dilakukan pada tingkat per bulan. Untuk memudahkan identifikasi jenis transaksi, maka ditambahkan kolom "Type" yang diisi dengan nilai "Domestic", kolom "Direction" yang diisi dengan nilai "In" atau "Out" berdasarkan arah transaksi, dan kolom "Source" yang diisi dengan nilai "SMAR&TS". Setelah fungsi dibuat, maka dijalankan menggunakan *for loop* dalam sebuah *array* untuk memproses semua file transaksi domestik dan digabungkan dalam satu *dataframe*.

```python
def process_swiftin(file):
    print(f"Processing {file}")
    df = pd.read_excel(file, dtype={"CustomerCode":str}, engine="calamine") # Read file

    # Strip column names and column values
    df.columns = df.columns.str.strip()
    df[df.select_dtypes(include="object").columns] = df.select_dtypes(include="object").apply(lambda col: col.str.strip(

    # Drop columns
    df.drop(["CompanyName", "BeneficiaryName","OriginalBeneficiaryAccountNumber","DebitAccountNumber","DebitCurrency","D

    # Rename columns
    df.rename(columns={"SwiftInCode":"Code",
                       "ValueDate":"Date",
                       "CreditCurrency":"Ccy",
                       "CreditAmount":"Amount",
                       "CustomerCode":"Cust. No"},
              inplace=True)

    # Convert datatypes and change date to period
    df["Cust. No"] = df["Cust. No"].astype(str)
    df["Date"] = pd.to_datetime(df["Date"])
    df["Date"] = df["Date"] + pd.offsets.MonthEnd(0)

    # Add missing/descriptor columns
    df["Type"] = "SWIFT"
    df["Direction"] = "In"
    df["Source"] = "SMAR&TS"

    return df

df_swiftin = pd.concat([process_swiftin(file) for file in files_swiftin], ignore_index=True)
df_swiftin.head(3)
```

Gambar 3.22 Kode fungsi pemrosesan data transaksi internasional

Gambar 3.22 menunjukkan kode fungsi yang dibuat untuk memproses data transaksi internasional, yaitu transaksi yang dilakukan melalui SWIFT, dalam platform *internet banking* SMAR&TS. Proses yang dilakukan secara umum sama dengan kode

pemrosesan data transaksi domestik pada Gambar 3.21, namun nama kolom berbeda dan kolom yang perlu di-*rename* juga berbeda. Penambahan kolom dilakukan namun dengan nilai yang berbeda. Kolom “Type” diisi dengan nilai “SWIFT”, kolom “Direction” diisi dengan nilai “In” atau “Out” berdasarkan arah transaksi, dan kolom “Source” diisi dengan nilai “SMAR&TS”. Setelah fungsi dibuat, maka dijalankan menggunakan *for loop* dalam sebuah *array* untuk memproses semua file transaksi internasional dan digabungkan dalam satu *dataframe*.

```python
def process_internal(file):
    print(f"Processing {file}")
    df = pd.read_excel(file, dtype={"CustomerCIF":str}, engine="calamine") # Read file

    # Strip column names and column values
    df.columns = df.columns.str.strip()
    df[df.select_dtypes(include="object").columns] = df.select_dtypes(include="object").apply(lambda col: col.str.strip(

    # Drop columns
    df.drop(["payeename", "AO_Code", "Dept_Id", "AO_Name", "payeebankname", "payeebankcode", "payeebankcountrycode"],
            axis=1, inplace=True)

    # Rename columns
    df.rename(columns={"InternalCode":"Code",
                       "valuedate":"Date",
                       "remitcurrency":"Ccy",
                       "remitamount":"Amount",
                       "CustomerCIF":"Cust. No",
                       "payername":"Cust. Name"},
              inplace=True)

    # Convert datatypes and change date to period
    df["Cust. No"] = df["Cust. No"].astype(str)
    df["Date"] = pd.to_datetime(df["Date"])
    df["Date"] = df["Date"] + pd.offsets.MonthEnd(0)

    # Add missing/descriptor columns
    df["Type"] = "Internal"
    df["Direction"] = "Internal"
    df["Source"] = "SMAR&TS"

    return df

df_internal = pd.concat([process_internal(file) for file in files_internal], ignore_index=True)
df_internal.head(3)
```

Gambar 3.23 Kode fungsi pemrosesan data transaksi internal

Gambar 3.23 menunjukkan kode fungsi untuk memproses data transaksi internal, yaitu transaksi yang dilakukan di dalam layanan *internet banking* SMAR&TS dan tidak meninggalkan ekosistem tersebut. Proses yang dilakukan secara umum sama dengan kode pemrosesan data transaksi domestik pada Gambar 3.21, namun nama kolom berbeda dan kolom yang perlu di-*rename* juga berbeda. Penambahan kolom dilakukan namun dengan nilai yang berbeda. Kolom “Type” diisi dengan nilai “Internal”, kolom “Direction” diisi dengan nilai “Internal”, dan kolom “Source” diisi dengan nilai “SMAR&TS”. Setelah fungsi dibuat, maka dijalankan menggunakan

for loop dalam sebuah *array* untuk memproses semua file transaksi internal dan digabungkan dalam satu *dataframe*.

```python
def process_aksesbisnis(file):
    print(f"Processing {file}")
    df = pd.read_excel(file, dtype={"CIF":str}, engine="calamine") # Read file

    # Strip column names and column values
    df.columns = df.columns.str.strip()
    df[df.select_dtypes(include="object").columns] = df.select_dtypes(include="object").apply(lambda col: col.str.strip(

    # Drop columns
    df.drop(["Company Code","Payment Ref#","Payment Subject","Created Date","Updated Date","Source Account",
             "Beneficiary Account","Message","Additional Message","Charge Status","Transfer Type",
             "Status"], axis=1, inplace=True)

    # Rename columns
    df.rename(columns={"CIF":"Cust. No",
                       "Company Name":"Cust. Name",
                       "Payment Date":"Date",
                       "Payment Type":"Type",
                       "Currency / Amount":"Amount",
                       "Transaction Ref#":"Code"},
              inplace=True)

    # Convert datatypes and change date to period
    df["Cust. No"] = df["Cust. No"].astype(str)
    df["Date"] = pd.to_datetime(df["Date"])
    df["Date"] = df["Date"] + pd.offsets.MonthEnd(0)

    # Add missing/descriptor columns
    df["Type"] = df["Type"].replace({"BTPN":"Internal","Payroll":"Internal","Multi Debit":"Domestic","Multi Credit":"Dom
    df["Direction"] = "Out"
    df["Source"] = "AksesBisnis"
    df["Ccy"] = "IDR"

    return df

df_aksesbisnis = pd.concat([process_aksesbisnis(file) for file in files_aksesbisnis], ignore_index=True)
df_aksesbisnis.head(3)
```

Gambar 3.24 Kode fungsi pemrosesan data transaksi platform AksesBisnis

Gambar 3.24 menunjukkan kode fungsi untuk memproses data transaksi yang dilakukan melalui platform AksesBisnis, yaitu sebuah platform *internet banking* yang juga dimiliki oleh SMBCI (sebelumnya BTPN). Proses yang dilakukan secara umum sama dengan kode pemrosesan data transaksi domestik pada Gambar 3.21, namun nama kolom berbeda dan kolom yang perlu di-*rename* juga berbeda. Penambahan kolom dilakukan namun dengan nilai yang berbeda. Kolom "Type" diisi beberapa nilai seperti "Internal" dan "Domestic" berdasarkan jenis transaksi, kolom "Direction" diisi dengan nilai "Out" karena semua transaksi dalam data ini berupa transaksi keluar, dan kolom "Source" diisi dengan nilai "AksesBisnis". Setelah fungsi dibuat, maka dijalankan menggunakan *for loop* dalam sebuah *array* untuk memproses semua file transaksi AksesBisnis dan digabungkan dalam satu *dataframe*.

```python
def process_pto(file):
    print(f"Processing {file}")
    df=pd.read_excel(file, skiprows=7, engine="calamine") # Read file

    # Strip column names and column values
    df.columns=df.columns.str.strip()
    df[df.select_dtypes(include="object").columns]=df.select_dtypes(include="object").apply(lambda col: col.str.strip())

    # Fill down status
    df["Status"]=df["Status"].ffill()

    # Filter only approved & not cancelled transactions
    df=df[df["Status"] == "Approved"]
    df=df[df["Is Cancel"] == False]

    # Drop columns
    df.drop(["No","Receive Date","Debit Account Number","Envelope Code","Status","Is Cancel","Transfer Type"], axis=1, i
    df.dropna(axis=1, inplace=True)

    # Rename columns
    df.rename(columns={"Company Name":"Cust. Name",
                       "Domestic Out Code":"Code",
                       "Value Date":"Date",
                       "Remit Currency":"Ccy",
                       "Payment Amount":"Amount"},
              inplace=True)

    # Convert datatypes and change date to period
    df["Date"] = pd.to_datetime(df["Date"])
    df["Date"] = df["Date"] + pd.offsets.MonthEnd(0)

    # Add missing/descriptor columns
    df["Type"] = "Domestic"
    df["Direction"] = "Out"
    df["Source"] = "Manual"

    return df

df_pto = pd.concat([process_pto(file) for file in files_pto], ignore_index=True)
df_pto.head(3)
```

Gambar 3.25 Kode fungsi pemrosesan data transaksi manual

Gambar 3.25 menunjukkan kode fungsi yang dibuat untuk memproses data transaksi manual, yaitu transaksi yang dilakukan melalui salah satu kantor cabang SMBCI sesuai permintaan nasabah. Proses yang dilakukan secara umum sama dengan kode pemrosesan data transaksi domestik pada Gambar 3.21, namun nama kolom berbeda dan kolom yang perlu di-*rename* juga berbeda. Data ini mengandung detail status setiap transaksi dan jika transaksi dibatalkan, sehingga data perlu difilter agar hanya diambil transaksi yang sudah berjalan dan tidak dibatalkan. Penambahan kolom dilakukan namun dengan nilai yang berbeda. Kolom “Type” diisi dengan nilai “Domestic” karena semua transaksi manual bersifat domestik, kolom “Direction” diisi dengan nilai “Out” karena semua transaksi dalam data ini berupa transaksi keluar, dan kolom “Source” diisi dengan nilai “Manual”. Setelah fungsi dibuat, maka dijalankan menggunakan *for loop* dalam sebuah *array* untuk memproses semua file transaksi manual dan digabungkan dalam satu *dataframe*.

```python
# Concatenate all dataframes
df_combined = pd.concat([df_domesticin, df_domesticout,
                         df_swiftin, df_swiftout,
                         df_internal,
                         df_aksesbisnis,
                         df_pto], ignore_index=True)

# Replace nan string values with actual NaN
df_combined["Cust. No"] = df_combined["Cust. No"].replace(["nan", "<NA>"], pd.NA)

# Sort values
df_combined.sort_values(by=["Source", "Type", "Direction", "Date"], ignore_index=True, inplace=True)

notify("Source data has been preprocessed and combined!")
print(f"Length of combined dataframe is {len(df_combined)}")
display(df_combined.head(3))
```

Gambar 3.26 Kode penggabungan semua data transaksi

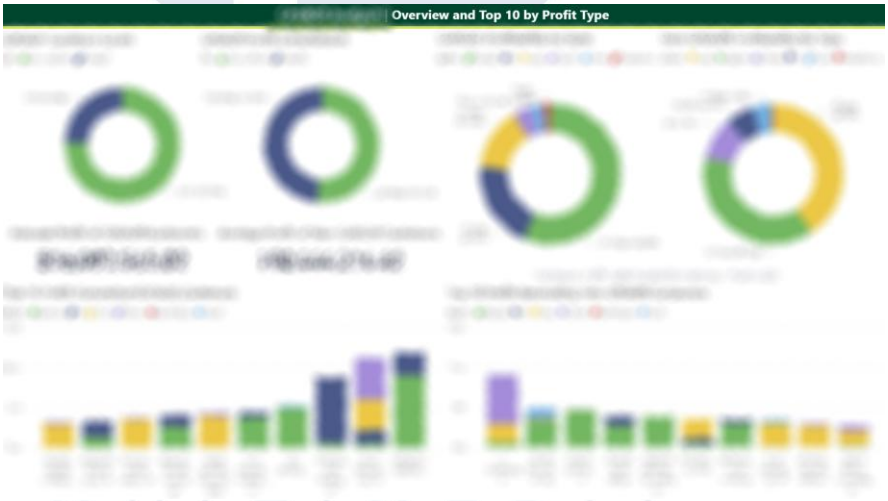
Gambar 3.26 menunjukkan kode untuk menggabungkan semua data transaksi yang telah diproses menggunakan fungsi pd.concat(). Kode ini juga mensortir baris data berdasarkan sumber, jenis, arah, dan tanggal transaksi.

Gambar 3.27 Kode *binning* nasabah berdasarkan jumlah transaksi

Gambar 3.27 menunjukkan kode untuk *binning* setiap nasabah berdasarkan rata-rata jumlah transaksi yang mereka lakukan per bulan. *Binning* ini dilakukan dengan pertama memecahkan data transaksi menjadi 3 *dataframe* berdasarkan 3 jenis *identifier* nasabah dalam sistem SMBCI, yaitu nama nasabah, nomor nasabah dalam sistem 1, dan nomor nasabah dalam sistem 2. Setiap *dataframe* dilakukan *group by* berdasarkan *identifier* yang ada dan berdasarkan tanggal, kemudian digunakan fungsi transform() dengan parameter

“mean” untuk mendapatkan rata-rata jumlah transaksi di semua bulan untuk setiap *identifier*. Jumlah transaksi ini kemudian dapat dilakukan *binning* menggunakan fungsi pd.cut() untuk mengklasifikasi nasabah berdasarkan rata-rata jumlah transaksi mereka.

Setelah *binning* nasabah selesai dilakukan, maka informasi segmentasi ini dapat dikembalikan menjadi satu *dataframe* yang mengandung semua nasabah. Untuk melakukan hal ini, dilakukan operasi merge() untuk menambahkan kolom segmen nasabah ke masing-masing *dataframe* dan terakhir dilakukan operasi pd.concat() untuk menyatukan 3 *dataframe* terpisah kembali menjadi 1 *dataframe*. Hasil akhirnya adalah data nasabah yang dibagi menjadi 2 segmen: nasabah yang melakukan jumlah transaksi tertentu setiap bulannya, dan nasabah yang di bawah minimum jumlah transaksi per bulan yang ditentukan.



Gambar 3.28 *Dashboard* Segmentasi Nasabah

Gambar 3.28 menunjukkan *dashboard* segmentasi nasabah yang dibuat menggunakan Microsoft Power BI setelah memproses data transaksi. *Dashboard* ini terdiri dari bagian sebagai berikut:

1. Proporsi Segmen Nasabah

   Bagian ini terdiri dari 2 visualisasi *pie chart*. *Pie chart* pertama menunjukkan proporsi dari setiap segmen

nasabah. *Pie chart* kedua menunjukkan proporsi profitabilitas dari setiap segmen nasabah. Bagian ini memungkinkan pembuktian bahwa nasabah yang jatuh ke dalam segmen yang memenuhi jumlah transaksi tertentu setiap bulannya relatif lebih sedikit, namun kontribusinya terhadap *profit* bank lebih besar.

2. Rata-Rata Profitabilitas Segmen Nasabah

   Bagian ini terdiri dari 2 visualisasi *scorecard*. *Scorecard* pertama menunjukkan rata-rata profitabilitas dari semua nasabah yang jatuh ke dalam segmen tidak memenuhi jumlah transaksi tertentu per bulan. *Scorecard* kedua menunjukkan rata-rata profitabilitas dari semua nasabah yang jatuh ke dalam segmen yang melakukan jumlah transaksi tertentu per bulan. Bagian ini menunjukkan bahwa rata-rata profitabilitas segmen nasabah kedua per bulannya relatif lebih tinggi daripada segmen nasabah pertama.

3. Detail Proporsi Jenis Profitabilitas Nasabah

   Bagian ini terdiri dari 2 visualisasi *pie chart*. *Pie chart* pertama menunjukkan perincian dari profitabilitas segmen nasabah pertama. *Pie chart* kedua menunjukkan perincian dari profitabilitas segmen nasabah kedua. Perincian ini mendetailkan sumber profitabilitas, contohnya jumlah *profit* yang datang dari deposit, *loan*, *trade*, dan sebagainya, sehingga dapat terlihat bagaimana kedua segmen berbeda dalam jenis *profit* yang paling banyak dihasilkan.

4. Nasabah Terbesar per Segmen

   Bagian ini terdiri dari 2 visualisasi *bar chart*. *Bar chart* pertama menunjukkan 10 nasabah yang menghasilkan paling banyak *profit* dalam segmen pertama. *Bar chart*

kedua menunjukkan 10 nasabah yang menghasilkan paling banyak *profit* dalam segmen kedua. Kedua *bar chart* ini dirincikan lagi menggunakan jenis *profit* sehingga terlihat susunan *profit* per nasabah. Bagian ini memungkinkan peninjauan siapa saja nasabah terbesar dalam setiap segmen.

3.3.1.3 Proyek *Dashboard* Ekosistem Nasabah

Dalam *transaction banking*, nasabah sering kali memiliki sejenis hubungan dengan nasabah lain. Hubungan ini terdiri dari sebuah nasabah *anchor* yang merupakan perusahaan utama, dan sebuah nasabah *spoke* yang merupakan perusahaan yang mendukung aktivitas bisnis nasabah *anchor*. Contohnya, jika ada nasabah yang menjadi *supplier* atau *distributor* bagi nasabah lain, maka nasabah tersebut dianggap sebagai *spoke* dan nasabah yang menerima disebut sebagai *anchor*. Beberapa hubungan dengan nasabah *anchor* yang sama membentuk sebuah ekosistem.

Visibilitas kinerja finansial pada tingkat ekosistem memungkinkan tim *TB Sales* untuk memahami kondisi, profitabilitas, dan potensi pertumbuhan seluruh jaringan bisnis dalam satu ekosistem. Terlebihnya, dibandingkan dengan analisis yang hanya berfokus pada nasabah individual, pendekatan ini memungkinkan tim *TB Sales* untuk melihat kontribusi setiap nasabah terhadap profitabilitas ekosistem mereka sehingga mendapatkan gambaran yang lebih komprehensif. Dengan demikian, tim *TB Sales* dapat mengidentifikasi ekosistem yang paling menguntungkan dan menemukan peluang *cross-selling* baru serta merumuskan strategi pengembangan hubungan bisnis yang lebih efektif.

Sejauh ini belum ada cara yang mudah bagi tim *TB Sales* untuk melihat keadaan nasabah pada tingkat ekosistem, sehingga menjadi penghambat peningkatan profitabilitas bank. Proyek ini bertujuan

untuk membuat *dashboard* interaktif yang mengatasi keterbatasan ini dengan menunjukkan informasi profitabilitas nasabah pada tingkat ekosistem, sehingga tim *TB Sales* memiliki dukungan data yang semakin komprehensif untuk menawarkan produk dan layanan kepada nasabah. Gambar 3.29 menunjukkan alur pengerjaan untuk proyek ini.

Gambar 3.29 Alur pengerjaan proyek 3.3.1.3

```
import pandas as pd
import numpy as np
from rapidfuzz import process, fuzz
```

Gambar 3.30 *Library* yang dipakai untuk proyek 3.3.1.3

Gambar 3.30 menunjukkan semua *library* yang diimpor untuk melaksanakan proyek ini, yaitu pandas, numpy, dan rapidfuzz.

```
print("Reading profit data...")
df_profit = pd.read_parquet("Processed Profit Data.parquet")
df_profit["Date"] = df_profit["Date"] + pd.offsets.MonthEnd(0)

print("Reading balance data...")
df_balance = pd.read_parquet("Processed Balance Data.parquet")
df_balance["Date"] = df_balance["Date"] + pd.offsets.MonthEnd(0)
df_balance.loc[:, "Metric"] = df_balance["Product"].astype(str) + " " + df_balance["Metric"].astype(str)
df_balance.drop("Value", axis=1, inplace=True)
df_balance.rename({"Local Value" : "Value"}, axis=1, inplace=True)

df_combined = pd.concat([df_profit, df_balance], ignore_index=True)
print("Profit & balance data read and concatenated!")
```

Gambar 3.31 Kode membaca dan menggabungkan data *balance* dan *profit*

Gambar 3.31 menunjukkan kode untuk membaca dan menggabungkan data *balance* dan data *profit* nasabah. Untuk data *balance*, dilakukan beberapa langkah praproses yaitu memastikan semua tanggal adalah akhir bulan, menggabungkan kolom produk dengan kolom metrik, dan menghapus nilai dalam mata uang asing untuk digantikan dengan nilai dalam rupiah. Untuk data *profit*, langkah praproses yang dilakukan adalah memastikan semua tanggal adalah akhir bulan.

```
print("Reading ecosystem data...")
df_ecosystem = pd.read_excel("Ecosystem Data.xlsx", sheet_name="Merge1")

df_ecosystem.drop                                          axis=1, inplace=True)

df_ecosystem.rename
                    "Debtor" : "Cust. Name"},
                   axis=1, inplace=True)

df_ecosystem["Cust. Name"] = df_ecosystem["Cust. Name"].str.upper()
df_ecosystem["Strategic Partner"] = df_ecosystem["Strategic Partner"].str.upper()
df_ecosystem["Downline"] = df_ecosystem["Downline"].str.title()
df_ecosystem["Role"] = df_ecosystem["Role"].str.title()
print("Ecosystem data read!")
```

Gambar 3.32 Kode membaca dan praproses data ekosistem nasabah

Gambar 3.32 menunjukkan kode untuk membaca dan praproses data ekosistem nasabah. *Dataframe* ini menjadi referensi untuk hubungan satu nasabah dengan nasabah lain dan peran yang dimainkan setiap nasabah. Praproses yang dilakukan antara lain memberikan nama baru untuk beberapa kolom dan memastikan nama nasabah dalam huruf besar semua.

```
combined_unique_names = df_combined["Cust. Name"].unique()
```

Gambar 3.33 Kode mendapatkan nama nasabah unik

Gambar 3.33 menunjukkan kode untuk mendapatkan nama nasabah unik dari data *balance* dan *profit* yang sudah digabungkan. Daftar nama ini akan dipakai untuk *fuzzy-matching* selanjutnya.

```
print("Fuzzy-matching unique strategic partner names...")
ecosystem_sp_names = df_ecosystem["Strategic Partner"].unique()

mapping_sp = {}
for name in combined_unique_names:
    match, score, _ = process.extractOne(
        name, ecosystem_sp_names, scorer=fuzz.token_sort_ratio
    )
    if score >= 92.5:
        mapping_sp[name] = match

print("Mapping to all strategic partner names...")
df_combined["Matched SP Name"] = df_combined["Cust. Name"].map(mapping_sp)
print("Done!")
```

Gambar 3.34 Kode *fuzzy-matching* nama *anchor* dengan data *balance* dan *profit* nasabah

Gambar 3.34 menunjukkan kode untuk melakukan *fuzzy-matching* nama *anchor* pada data ekosistem nasabah dengan daftar nama nasabah unik pada Gambar 3.33. Jika ada *match* yang ditemukan, maka *match* tersebut akan ditambahkan ke sebuah *mapping*. *Mapping* ini kemudian diterapkan pada kolom "Matched SP

Name" untuk menemukan nasabah yang berupa *anchor* di data *balance* dan *profit*.

```
print("Fuzzy-matching unique customer names...")
ecosystem_unique_names = df_ecosystem["Cust. Name"].unique()

mapping = {}
for name in ecosystem_unique_names:
    match, score, _ = process.extractOne(
        name, combined_unique_names, scorer=fuzz.token_sort_ratio
    )
    if score >= 92.5:
        mapping[name] = match

print("Mapping to all customer names...")
df_combined["Matched Name"] = df_combined["Cust. Name"].map(mapping)
print("Done!")
```

Gambar 3.35 Kode *fuzzy-matching* nama *spoke* dengan data *balance* dan *profit* nasabah

Gambar 3.35 menunjukkan kode untuk melakukan *fuzzy-matching* nama *spoke* pada data ekosistem nasabah dengan daftar nama nasabah unik pada Gambar 3.33. Jika ada *match* yang ditemukan, maka *match* tersebut akan ditambahkan ke sebuah *mapping*. *Mapping* ini kemudian diterapkan pada kolom "Matched Name" untuk menemukan nasabah yang berupa *spoke* di data *balance* dan *profit.*

```
print("Dropping profit & balance rows that are irrelevant...")
df_matched_sp = df_combined.dropna(subset="Matched SP Name")
df_matched_name = df_combined.dropna(subset="Matched Name")

df_result = pd.concat([df_matched_sp, df_matched_name], ignore_index=True)

df_result.drop("Matched SP Name", axis=1, inplace=True)
print("Done!")
```

Gambar 3.36 Kode *filtering* data *balance* dan *profit* nasabah

Setelah menemukan nasabah yang berupa *anchor* dan *spoke*, maka data *balance* dan *profit* dapat difilter agar tersisa hanya nasabah yang berhubungan. Gambar 3.36 menunjukkan kode untuk melakukan filter ini dengan mengambil baris-baris data di mana kolom "Matched SP Name" atau "Matched Name" tidak kosong.

```
df_result_labels = df_result[["Cust. Name", "Dept",
                              "RM Name", "TL Name", "TH Name", "TB Sales", "TB Sales Lead", "TB Sales Head",
                              "Ccy", "Matched Name"]].drop_duplicates(subset="Cust. Name")
```

Gambar 3.37 Kode pembuatan *dataframe* label nasabah

Gambar 3.37 menunjukkan kode untuk membuat *dataframe* yang berisi label-label nasabah seperti nama, departemen internal bank yang bertanggung jawab, dan sebagainya. *Dataframe* ini perlu dibuat untuk mempertahankan label ketika dilakukan pivot tabel selanjutnya.

```
df_result_pivot = pd.pivot_table(df_result,
                                 index=["Cust. No", "Cust. Name", "Date"],
                                 columns="Metric",
                                 values="Value")
df_result_pivot.columns.name = None
df_result_pivot.reset_index(inplace=True)
```

Gambar 3.38 Kode pivot untuk membuat *dataframe* utama

Gambar 3.38 menunjukkan kode untuk mempivot *dataframe balance* dan *profit* yang sudah digabungkan sebelumnya. *Dataframe* ini akan menjadi *dataframe* utama yang mengandung semua informasi yang dibutuhkan. Pivot dilakukan dengan indeks nomor nasabah, nama nasabah, dan tanggal. Nilai kolom yang dilebarkan adalah kolom metrik agar terlihat nilai dari setiap metrik.

```
df_result_pivot = df_result_pivot.merge(df_result_labels, on="Cust. Name", how="left")
```

Gambar 3.39 Kode penambahan label kembali ke *dataframe* utama

Gambar 3.39 menunjukkan kode untuk menambahkan label-label nasabah kembali ke *dataframe* utama, sehingga *dataframe* utama sekarang memiliki informasi tambahan terkait nasabah.

```
df_result_pivot = df_result_pivot.merge(df_ecosystem.drop(df_ecosystem.columns[0], axis=1), on="Cust. Name", how="left")
```

Gambar 3.40 Kode penambahan detail ekosistem ke *dataframe* utama

Gambar 3.40 menunjukkan kode untuk menambahkan detail ekosistem ke *dataframe* utama, sehingga *dataframe* utama sekarang memiliki informasi hubungan ekosistem bagi setiap nasabah.

```
df_result_pivot.loc[df_result_pivot["Role"].isna(), "Role"] = "Anchor"
df_result_pivot.loc[df_result_pivot["Role"] == "Anchor", "Strategic Partner"] = df_result_pivot["Cust. Name"]

df_result_pivot["Matched SP Name"] = df_result_pivot["Strategic Partner"].map(mapping_sp)

df_result_pivot.loc[(df_result_pivot["Role"] == "Anchor") & (df_result_pivot["Matched SP Name"].notna()), "Strategic Par

df_result_pivot.drop("Matched SP Name", axis=1, inplace=True)
```

Gambar 3.41 Kode penanganan nasabah *anchor*

Gambar 3.41 menunjukkan kode untuk menangani nasabah yang berupa *anchor* pada *dataframe* utama. Antara lain, kode ini mengisi kolom “Role” yang masih kosong dengan nilai ”Anchor” karena sudah dipastikan bahwa semua baris kosong pasti berupa *anchor*. Kolom “Strategic Partner” juga diisi dengan nama *anchor* itu sendiri agar muncul saat difilter pada *dashboard*.

```
df_result_pivot["DEPOSIT PROFIT"] = (df_result_pivot["CA NII"] + df_result_pivot["CA LIQ. ADJ"]) + (df_result_pivot["TD
df_result_pivot["TRADE PROFIT"] = df_result_pivot["IMPORT FEES"] + df_result_pivot["EXPORT FEES"] + df_result_pivot["GUA
df_result_pivot["LOAN PROFIT"] = df_result_pivot["LOAN NII"] + df_result_pivot["LOAN LIQ. ADJ"] + df_result_pivot["JIBOR
df_result_pivot["FX PROFIT"] = df_result_pivot["FX"] + df_result_pivot["DERIVATIVES"]
```

Gambar 3.42 Kode menambahkan angka *profit* agregat

Angka *profit* dibagi-bagi menjadi beberapa metrik. Untuk memudahkan pembacaan angka tersebut dan mempercepat pemahaman, maka dapat dibuat angka agregat. Gambar 3.42 menunjukkan kode untuk menambahkan angka *profit* agregat pada *dataframe* utama. *Profit* agregat yang dibuat adalah:

1. Deposit yang terdiri dari NII dan *liquid adjustment* untuk *current account*, *savings account*, dan *time deposit*, serta *special rate adjustment*.
2. *Trade* yang terdiri dari biaya impor, ekspor, *guarantee*, dan lainnya.
3. *Loan* yang terdiri dari NII dan *liquid adjustment* untuk pinjaman serta JIBOR *adjustment*.
4. FX yang terdiri dari devisa dan derivatif.

```
unique_pairs = df_ecosystem[["Cust. No", "Cust. Name", "Strategic Partner", "Scheme", "Downline", "Role"]].drop_duplicat
unique_dates = df_result_pivot["Date"].unique()

df_multiindex = pd.MultiIndex.from_product([unique_pairs.index, unique_dates], names=["Pair Index", "Date"]).to_frame(in

df_multiindex = df_multiindex.merge(unique_pairs.reset_index(drop=True),
                                    left_on="Pair Index", right_index=True).drop(columns="Pair Index")
```

Gambar 3.43 Kode membuat *dataframe* multiindex

Tidak semua nasabah yang ada dalam data ekosistem terdata di data *balance* dan *profit* sehingga nasabah-nasabah tersebut sejauh ini tidak muncul pada *dataframe* utama. Gambar 3.43 menunjukkan kode untuk membuat *dataframe* multiindex yang bertujuan untuk memastikan semua nasabah muncul pada *dataframe* utama.

Dataframe multiindex ini memasangkan setiap nama nasabah unik dengan semua tanggal yang terdata di *dataframe* utama agar muncul untuk setiap tanggal.

```
df_absent = df_multiindex.merge(df_result_pivot[["Cust. Name", "Matched Name"]], left_on="Cust. Name", right_on="Matched
df_absent.drop("Cust. Name_y", axis=1, inplace=True)
df_absent.rename({"Cust. Name_x" : "Cust. Name"}, axis=1, inplace=True)
```

Gambar 3.44 Kode penggabungan *dataframe* multiindex dengan *dataframe* utama

Gambar 3.44 menunjukkan kode untuk menggabungkan *dataframe* multiindex dengan *dataframe* utama dan koreksi nama nasabah.

```
df_absent = df_absent[df_absent["_merge"] == "left_only"].copy()
df_absent.drop("_merge", axis=1, inplace=True)
```

Gambar 3.45 Kode untuk mendapatkan nasabah yang tidak memiliki data *profit*

Gambar 3.45 menunjukkan kode untuk mendapatkan nasabah yang tidak memiliki data *profit* dengan mengambil baris-baris data di mana kolom "_merge" bernilai "left_only". Nilai ini mengindikasikan bahwa data ini hanya muncul pada data ekosistem, sehingga baris data tersebut adalah yang harus ditambahkan ke *dataframe* utama.

```
df_final = pd.concat([df_result_pivot, df_absent], ignore_index=True)

df_final = df_final[["Strategic Partner", "Role", "Scheme", "Downline",
                     "Date", "Cust. No", "Cust. Name",
                     "RM Name", "TB Sales Head", "TB Sales Lead", "TB Sales",
                     "CA AVR", "CA END", "TD AVR", "TD END", "LOAN AVR", "LOAN END", "DEPOSIT PROFIT", "TRADE PROFIT", "

df_final["Date"] = df_final["Date"].dt.date
```

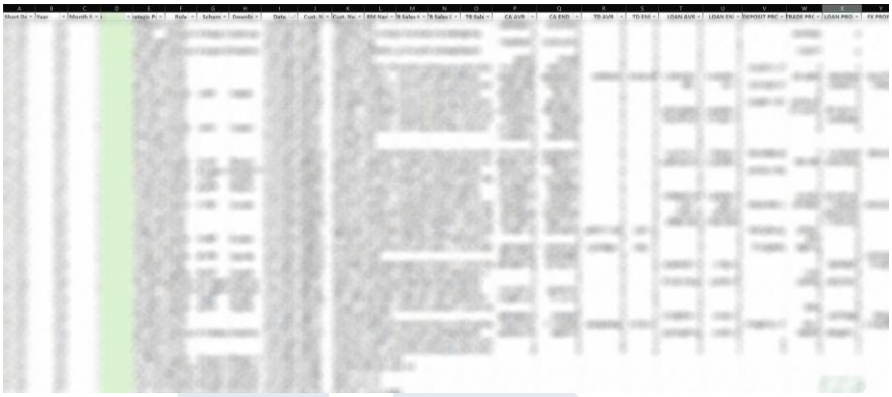
Gambar 3.46 Kode membuat *dataframe* final

Gambar 3.46 menunjukkan kode untuk membuat *dataframe* final. Kode ini terdiri dari beberapa langkah yaitu menggabungkan *dataframe* utama dengan *dataframe* yang berisi nasabah tanpa data *balance* atau *profit*, kemudian mengatur kembali urutan kolom. Terakhir adalah membersihkan kolom tanggal dengan menghapus konotasi jam dan menit.

```
df_final.to_excel("Profit & Balance with Ecosystem Data.xlsx", index=False)
```

Gambar 3.47 Kode mengoutput *dataframe* final

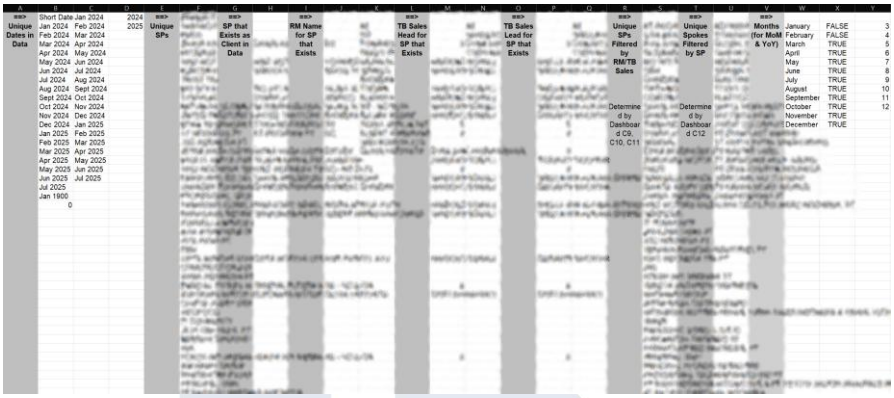
Gambar 3.47 menunjukkan kode untuk mengoutput *dataframe* final menjadi file Microsoft Excel.



Gambar 3.48 Sampel data yang sudah diproses untuk *dashboard* ekosistem

Gambar 3.48 menunjukkan sampel data berbentuk Microsoft Excel yang merupakan hasil output *dataframe* final dari Gambar 3.47. Untuk memfasilitasi pembuatan *dashboard*, maka data digeser ke kanan beberapa kolom, kemudian ditambahkan tiga kolom baru yang berisi formula Excel. Nilai dari ketiga kolom ini akan dipakai pada *helper sheet*. Tiga kolom tersebut adalah:

1. Kolom A (*Short Date*)
   Menggunakan formula TEXT(I, “mmm yyyy”) untuk mendapatkan bulan dan tahun dalam bentuk teks dari kolom I yaitu kolom “Date”.
2. Kolom B (*Year*)
   Menggunakan formula YEAR(I) untuk mendapatkan tahun dari kolom I yaitu kolom “Date”.
3. Kolom C (*Month Num*)
   Menggunakan formula MONTH(I) untuk mendapatkan bulan dalam bentuk angka dari kolom I yaitu kolom “Date”.



Gambar 3.49 *Helper sheet* untuk memfasilitasi *dashboard* ekosistem

Gambar 3.49 menunjukkan *helper sheet* yang dibuat untuk memfasilitasi fungsionalitas *dashboard*. Kolom dalam *helper sheet* ini dibagi sebagai berikut:

1. Kolom B, C, D (Bulan dan Tahun)

   Kolom B berisi daftar nilai unik dari kolom "Date" pada data terproses menggunakan fungsi UNIQUE(). Kolom C membersihkan daftar nilai unik tersebut dengan menghapus nilai seperti "Short Date", "Jan 1900", dan "0" menggunakan fungsi FILTER(). Kolom D berisi daftar tahun yang ada pada data terproses menggunakan fungsi FILTER() dan UNIQUE().
2. Kolom F (*Anchor* Unik)

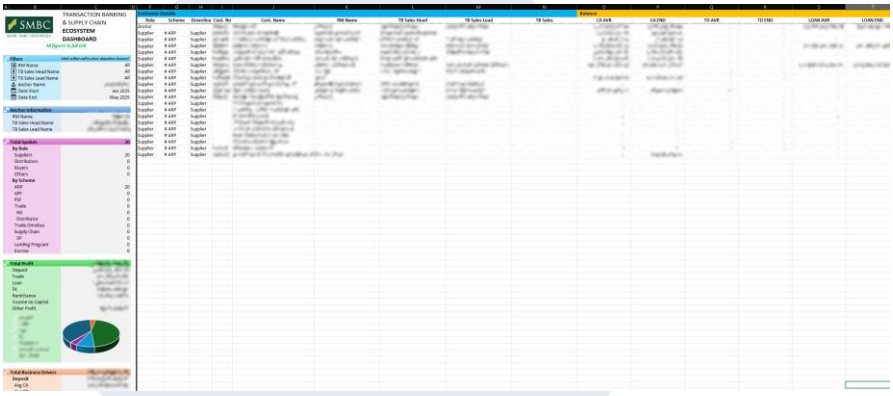
   Kolom F berisi daftar nilai unik dari kolom "Strategic Partner" (*anchor*) pada data terproses menggunakan fungsi UNIQUE().
3. Kolom H (*Anchor* yang Ada Sebagai Nasabah)

   Kolom H berisi daftar nama *anchor* yang ada di data terproses sebagai nasabah SMBCI menggunakan fungsi IF() dan COUNTIF(). Jika COUNTIF() menghasilkan angka yang lebih dari 0, maka *anchor* tersebut ada.
4. Kolom J, K (*RM Anchor*)

Kolom J berisi nama *relationship manager* (RM) dari setiap *anchor* yang didaftarkan pada kolom F. Kolom K membersihkan kolom J dengan menghapus nilai "0" menggunakan FILTER(), mendapatkan nilai unik dengan UNIQUE(), dan menambahkan nilai "All" di paling atas menggunakan VSTACK().

5. Kolom M, N (*TB Sales Head Anchor*)
   Kolom J berisi nama *TB Sales Head* dari setiap *anchor* yang didaftarkan pada kolom F. Kolom K membersihkan kolom J dengan menghapus nilai "0" menggunakan FILTER(), mendapatkan nilai unik dengan UNIQUE(), dan menambahkan nilai "All" di paling atas menggunakan VSTACK().
6. Kolom P, Q (*TB Sales Lead Anchor*)
   Kolom J berisi nama *TB Sales Lead* dari setiap *anchor* yang didaftarkan pada kolom F. Kolom K membersihkan kolom J dengan menghapus nilai "0" menggunakan FILTER(), mendapatkan nilai unik dengan UNIQUE(), dan menambahkan nilai "All" di paling atas menggunakan VSTACK().
7. Kolom S (*Anchor* Unik Berdasarkan Filter *TB Sales*)
   Kolom S berisi daftar nama *anchor* unik yang berada di bawah naungan nama *TB Sales* yang diseleksi pada filter. Selain menggunakan fungsi FILTER() dan UNIQUE(), daftar ini juga menggunakan fungsi SORT() untuk menyortir nama dari A ke Z.
8. Kolom U (*Spoke* Unik Berdasarkan Filter *Anchor*)
   Kolom U berisi daftar nama *spoke* unik yang berada di bawah naungan *anchor* yang diseleksi pada filter. Daftar ini menggunakan fungsi UNIQUE() dan FILTER().
9. Kolom W, X, Y (Seleksi Bulan MoM & YoY)

Kolom W berisi daftar nama bulan dari Januari sampai dengan Desember. Kolom X berisi nilai TRUE/FALSE dari kotak centang setiap bulan pada *sheet dashboard* MoM & YoY. Kolom Y mendaftarkan angka bulan yang memiliki nilai TRUE menggunakan fungsi FILTER() dan SEQUENCE().



Gambar 3.50 Tampilan *dashboard* YTD

Gambar 3.50 menunjukkan tampilan *dashboard year-to-date*. *Dashboard* ini dibagi menjadi dua sisi. Sisi kanan menunjukkan detail dari semua nasabah yang berada di dalam satu ekosistem. Detail yang ditunjukkan dari setiap nasabah adalah:

1. Detail Nasabah

   Kelompok kolom ini menunjukkan peran nasabah dalam ekosistem mereka dan produk/layanan TB yang dipakai. Kelompok kolom ini juga menunjukkan data utama nasabah seperti nomor unik nasabah, nama nasabah, dan RM serta *TB Sales* yang bertanggung jawab atas nasabah tersebut.

2. Data *Balance*

   Kelompok kolom ini menunjukkan data *balance* nasabah yaitu angka *average balance* dan *ending balance* dari akun *current account* (CA), *time deposit* (TD), dan *loan*.

3. Data *Profit* Majemuk

   Kelompok kolom ini menunjukkan data *profit* majemuk nasabah yaitu *profit* dari deposit, perdagangan, *loan*, dan devisa.

4. Data *Profit* Granular

   Kelompok kolom ini menunjukkan data *profit* granular nasabah termasuk *net interest income* dari akun *current account* (CA), *time deposit* (TD), dan *loan*. Ada juga *profit* berbasis *fee* seperti biaya dokumen impor, ekspor, dan *bank guarantee*.

5. Total *Profit*

   Kelompok kolom ini menggunakan fungsi SUM() untuk menghitung jumlah *profit* yang dihasilkan nasabah menggunakan angka dari data *profit* granular.

Sisi kiri menunjukkan semua filter interaktif yang bisa diubah oleh pengguna *dashboard* bersama dengan data yang bersifat ringkasan. Filter interaktif yang tersedia adalah:

1. *RM Name*

   Filter ini merupakan *dropdown list* yang dapat digunakan untuk memfilter nama *anchor* yang muncul pada filter *anchor name* berdasarkan nama RM yang dipilih.

2. *TB Sales Name*

   Filter ini merupakan tiga *dropdown list* yang dapat digunakan untuk memfilter nama *anchor* yang muncul pada filter *anchor name* berdasarkan nama *TB Sales*, *TB Sales Lead*, dan/atau *TB Sales Head* yang dipilih.

3. *Anchor Name*

   Filter ini merupakan *dropdown list* yang dapat digunakan untuk memfilter nasabah yang ditunjukkan

pada bagian kanan *dashboard* berdasarkan nama *anchor* yang dipilih.

4. *Date Start & Date End*

   Filter ini merupakan *dropdown list* yang dapat digunakan untuk memfilter angka yang ditunjukkan pada bagian kanan *dashboard* berdasarkan tanggal awal dan akhir yang dipilih. Contohnya, jika tanggal awal yang dipilih adalah Januari 2025 dan tanggal akhir yang dipilih adalah Maret 2025, maka angka *profit* yang akan ditunjukkan adalah jumlah *profit* yang dihasilkan selama Januari 2025 sampai dengan Maret 2025.

Ada juga data ringkasan yang tersedia yaitu:

1. *Total Spokes by Role*

   Ringkasan ini menunjukkan jumlah nasabah *spoke* berdasarkan peran mereka dalam ekosistem yaitu *supplier*, *distributor*, *buyers*, dan *others*.

2. *Total Spokes by Scheme*

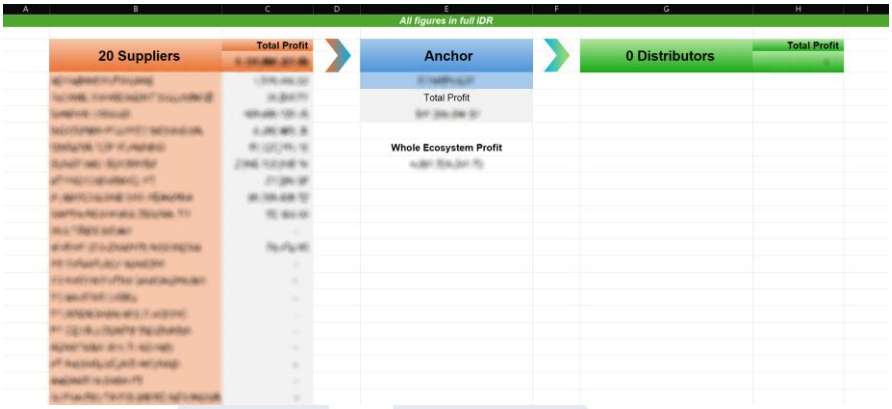
   Ringkasan ini menunjukkan jumlah nasabah *spoke* berdasarkan produk/layanan TB yang mereka pakai seperti *accounts receivable purchase* (ARP) dan *accounts payable financing* (APF).

3. *Total Profit*

   Ringkasan ini menunjukkan jumlah *profit* yang dihasilkan dari satu ekosistem dan memberikan rincian seperti berapa banyak *profit* yang dihasilkan datang dari deposit.

4. *Total Business Drivers*

   Ringkasan ini menunjukkan jumlah *business driver* dari satu ekosistem dan memberikan rincian seperti rata-rata *balance* pada *current account* untuk ekosistem tersebut.



Gambar 3.51 Tampilan visualisasi ekosistem berbasis *dashboard* YTD

Gambar 3.51 menunjukkan tampilan visualisasi ekosistem nasabah. Data yang ditampilkan pada visualisasi ini berdasarkan data yang ditunjukkan pada *dashboard* YTD. Bagian kiri menunjukkan jumlah *supplier* dan daftar nama semua *supplier* untuk ekosistem tersebut dan jumlah *profit* mereka. Bagian tengah menunjukkan nama *anchor* dan jumlah *profit* dari *anchor* tersebut, serta jumlah *profit* dari seluruh ekosistem. Bagian kanan menunjukkan jumlah *distributor* dan daftar nama semua *distributor* untuk ekosistem tersebut dan jumlah *profit* mereka.

Gambar 3.52 Tampilan *dashboard* MoM & YoY

Gambar 3.52 menunjukkan tampilan *dashboard month-over-month* (MoM) dan *year-over-year* (YoY). Tampilan *dashboard* ini secara keseluruhan mirip dengan *dashboard* YTD pada Gambar 3.50. Perbedaannya adalah dalam filter interaktif yang disediakan, yaitu:

1. *Year A & Year B*

Filter ini merupakan *dropdown list* yang dapat digunakan untuk memfilter angka yang ditunjukkan pada bagian kanan *dashboard* berdasarkan tahun awal dan tahun akhir yang dipilih.

2. *Months*

   Filter ini terdiri dari 12 objek *checkbox* untuk merepresentasikan setiap bulan. Filter ini dapat digunakan untuk memfilter angka yang ditunjukkan pada bagian kanan *dashboard* berdasarkan bulan-bulan yang dicentang.

3.3.1.4 Proyek Otomatisasi Pengisian Formulir

Salah satu layanan yang ditawarkan oleh SMBCI kepada nasabah korporat adalah layanan *internet banking* SMAR&TS. Proses *onboarding* nasabah baru ke SMAR&TS melibatkan pengisian banyak formulir sehingga memakan banyak waktu dan menimbulkan risiko *human error*. Hal ini mengharuskan adanya proses yang lebih sederhana bagi nasabah dan tim *TB Sales* agar pendaftaran nasabah baru ke SMAR&TS dapat dilakukan dengan lebih cepat dan efisien.

Proyek ini bertujuan untuk mengembangkan sistem automasi pengisian formulir pendaftaran SMAR&TS. Nasabah hanya perlu mengisi satu formulir sederhana yang berisi data yang dibutuhkan oleh bank untuk mendaftarkan mereka. Setelah formulir ini diberikan ke bank, maka tim *TB Sales* tinggal memasukkan formulir ini ke dalam program yang akan menghasilkan formulir-formulir resmi pendaftaran SMAR&TS berdasarkan data yang diinput oleh nasabah. Proses ini mengeliminasi kebutuhan untuk menginput data yang sama ke dalam formulir yang berbeda-beda sehingga sangat mengurangi beban kerja manual dan mengurangi risiko *human error* karena penginputan data dilakukan secara otomatis. Proyek ini sudah pernah dikerjakan sebelumnya, namun ada kebutuhan untuk merevisi

kembali program yang dibuat agar lebih cepat dan mudah untuk digunakan oleh tim TB, serta mendukung formulir pendaftaran SMAR&TS versi terbaru.

```
#%% Import Libraries
from datetime import datetime
from pathlib import Path
import re, os, docx, win32com.client, pythoncom, shutil, threading, gc
from pypdf import PdfReader, PdfWriter

from tkinter import *
from tkinter.ttk import *
from tkinter import filedialog, messagebox, Checkbutton
from ctypes import windll
windll.shcore.SetProcessDpiAwareness(2)
```

Gambar 3.53 *Library* yang diimpor untuk proyek 3.3.1.4

Gambar 3.53 menunjukkan semua *library* yang diimpor untuk memfasilitasi pembuatan program otomatisasi pengisian formulir. Beberapa *library* penting yang dipakai antara lain adalah tkinter untuk pembuatan *graphical user interface* (GUI) yang akan ditunjukkan kepada pengguna program, pywin32 untuk berinteraksi langsung dengan program Microsoft Word dan Microsoft Excel melalui antarmuka *component object model* (COM) Windows untuk memodifikasi file DOCX dan XLSX, serta pypdf untuk memodifikasi file PDF.

Country Supplement
Applicable to: Indonesia

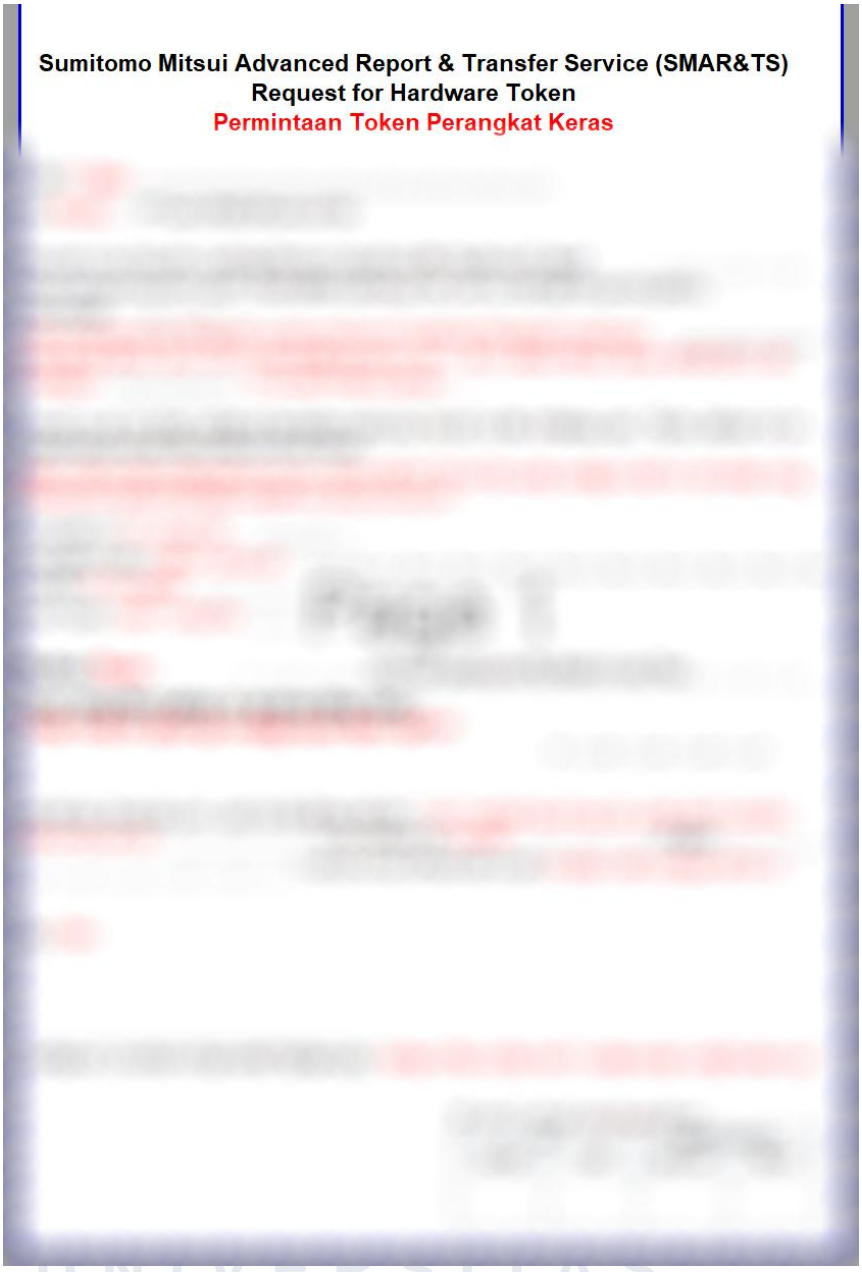
Country Supplement
Berlaku untuk: Indonesia

Gambar 3.54 Sampel formulir persetujuan negara spesifik

```
1081  def fill_countrysupplement(word):
1082      try:
1083          doc = word.Documents.Open(file_countrysupplement)
1084          doc.SaveAs(os.path.join(os.getcwd(), newfoldername, newname["countrysupplement"]))
1085
1086          # Day
1087          doc.FormFields(1).Result = str(date_day)
1088          doc.FormFields(4).Result = str(date_day)
1089
1090          # Month & Year
1091          doc.FormFields(2).Result = f"{date_english_month_name} {str(date_year)}"
1092          doc.FormFields(5).Result = f"{date_indonesian_month_name} {str(date_year)}"
1093
1094          # Full company name
1095          doc.FormFields(3).Result = _1basicinformation["fullcompanyname"]
1096          doc.FormFields(6).Result = _1basicinformation["fullcompanyname"]
1097          doc.FormFields(7).Result = _1basicinformation["fullcompanyname"]
1098
1099          doc.Close(True)
1100      except Exception as e:
1101          print(f"Error @ Country Supplement: {e}")
1102          messagebox.showerror("Error @ Country Supplement", f"{e}\n\nThe program will now exit.")
1103          word.Quit()
1104          root.destroy()
1105          root.quit()
```

Gambar 3.55 Fungsi untuk pengisian formulir persetujuan negara spesifik

Gambar 3.54 menunjukkan sampel isi formulir persetujuan negara spesifik. Formulir ini merupakan tambahan dari persetujuan pendaftaran SMAR&TS yang spesifik untuk satu negara yaitu Indonesia. Formulir ini adalah dokumen baru, sehingga perlu dibuat kode baru agar program pengisian formulir dapat mengisi formulir ini secara otomatis. Gambar 3.55 menunjukkan kode yang dibuat untuk mengisi formulir persetujuan negara spesifik. Formulir ini merupakan file Microsoft Word sehingga digunakan antarmuka COM untuk berinteraksi dengan file tersebut. Semua bagian formulir yang perlu diisi sudah berbentuk objek *formfield*, sehingga referensi ke *formfield* spesifik dapat dilakukan melalui angka indeksnya. Data yang dimasukkan ke dalam formulir ini antara lain tanggal dan nama perusahaan nasabah. Ketika semua *formfield* sudah diisi, maka akan disimpan sebagai file Microsoft Word dengan nama baru sesuai Gambar 3.59.



Gambar 3.56 Sampel formulir permintaan *hard token*

```
def fill_hardtoken(excel):
    generated_files = []

    folder_temp = Path.home() / "SMARTSAutomationTemp"
    folder_temp.mkdir(exist_ok=True)
    for usernum in range(0, len(_4users)):
        if _4users[usernum]["hardtoken"] == "Y":
            workbook = excel.Workbooks.Open(file_hardtoken)
            form = workbook.Sheets("C10")

            rng = form.Range("M6"); rng.Value = f"{date_day} {date_english_month_name} {date_year}"; del rng

            rng = form.Range("BF10"); rng.Value = f"{date_day} {date_english_month_name} {date_year}"; del rng
            rng = form.Range("L12"); rng.Value = f"{date_day} {date_english_month_name} {date_year}"; del rng

            rng = form.Range("BI13"); rng.Value = f"{date_day} {date_indonesian_month_name} {date_year}"; del rng
            rng = form.Range("J15"); rng.Value = f"{date_day} {date_indonesian_month_name} {date_year}"; del rng

            [illegible] del rng

            rng = form.Range("AB22"); rng.Value = _1basicinformation["fullcompanyname"][:50]; del rng

            [illegible] del rng
            [illegible] del rng

            [illegible]
            del form
            workbook.Close(SaveChanges=False)
            del workbook
            gc.collect()

            [illegible]

    for temp_file, userid in generated_files:
        [illegible]
        shutil.move(temp_file, os.path.join(os.getcwd(), newfoldername, filename))

    if folder_temp.exists():
        shutil.rmtree(folder_temp)
```

Gambar 3.57 Fungsi untuk pengisian formulir permintaan *hard token*

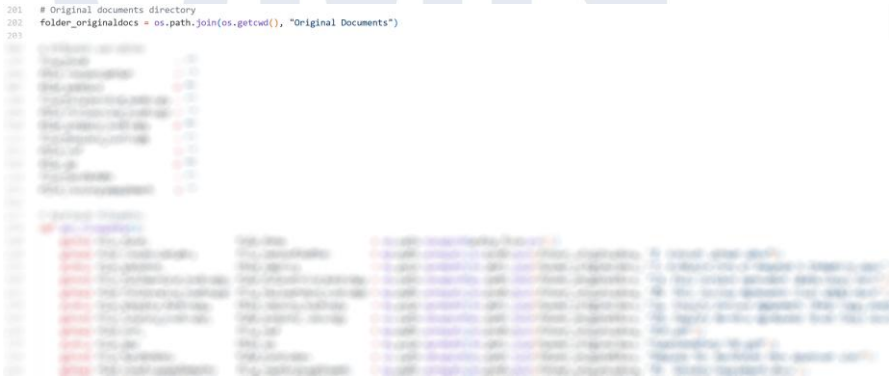
Gambar 3.56 menunjukkan sampel isi formulir permintaan *hard token* yang berbentuk file Microsoft Excel. Sebagai standar keamanan bertransaksi, sistem SMAR&TS akan menanyakan kode autentikasi sekali pakai kepada pengguna setiap kali pengguna membuat transaksi. Kode autentikasi sekali pakai ini dapat dihasilkan oleh sebuah aplikasi di telepon genggam (*soft token*) atau melalui sebuah perangkat keras khusus untuk menghasilkan kode tersebut (*hard token*). Jika nasabah memilih untuk menggunakan *hard token*, maka formulir ini perlu diisi. Gambar 3.57 menunjukkan kode yang dibuat untuk mengisi formulir permintaan *hard token* secara otomatis. Kode ini hanya berjalan jika ada *user* yang meminta *hard token* pada formulir pendaftaran awal. Hal pertama yang dilakukan adalah membuat folder sementara untuk menampung file formulir yang sudah terisi. Untuk setiap *user* yang meminta *hard token*, akan dibuka file *template* formulir dan semua *cell* diisi dengan data yang sesuai, kemudian disimpan ke dalam folder sementara. Ketika semua formulir permintaan *hard token* telah dihasilkan, maka semua file akan

dipindahkan ke folder output sebenarnya dan folder sementara akan dihapus. Secara teknis, kode ini perlu melibatkan banyak *garbage collection* dan perpindahan file formulir untuk menghindari proses program Microsoft Excel yang tetap terbuka setelah sudah tidak dibutuhkan lagi.

```
if _4hardtoken == True and checkbtn_hardtoken["value"].get() == 1:
    update_ui(label_status.config, text="Hard token request detected, generating the form(s)...");
    try:
        pythoncom.CoInitialize()
        excel = win32com.client.DispatchEx("Excel.Application")
        excel.Visible = False
        update_progressbar(90); fill_hardtoken(excel)
    except Exception as e:
        print(f"Error @ Hard Token Form: {e}")
        messagebox.showerror("Error at Hard Token Form", e)
    finally:
        excel.Quit()
        win32com.client.gencache.Rebuild()
        pythoncom.CoUninitialize()
```

Gambar 3.58 Kode eksekusi fungsi pengisian formulir permintaan *hard token*

Gambar 3.58 menunjukkan kode untuk mengeksekusi fungsi pengisian formulir permintaan *hard token* yang ditunjukkan pada Gambar 3.57. Fungsi hanya dijalankan jika pengguna program menyatakan bahwa mereka ingin programnya untuk menghasilkan file formulir permintaan *hard token* dan jika memang ada *user* yang terdata meminta *hard token* pada formulir pendaftaran awal. *Library* yang dipakai untuk membantu fungsi ini berinteraksi dengan program Microsoft Excel secara langsung adalah pythoncom dan win32com.



Gambar 3.59 Kode pencarian file formulir *template*

Gambar 3.59 menunjukkan kode untuk mencari semua file formulir yang berupa *template* kosong dan siap untuk diisi dengan

data. Kode ini mengasumsikan bahwa ada folder bernama “Original Documents” di tempat yang sama dengan file *executable* (EXE) program. Dalam folder tersebut, setiap formulir akan dipasangkan dengan satu variabel.

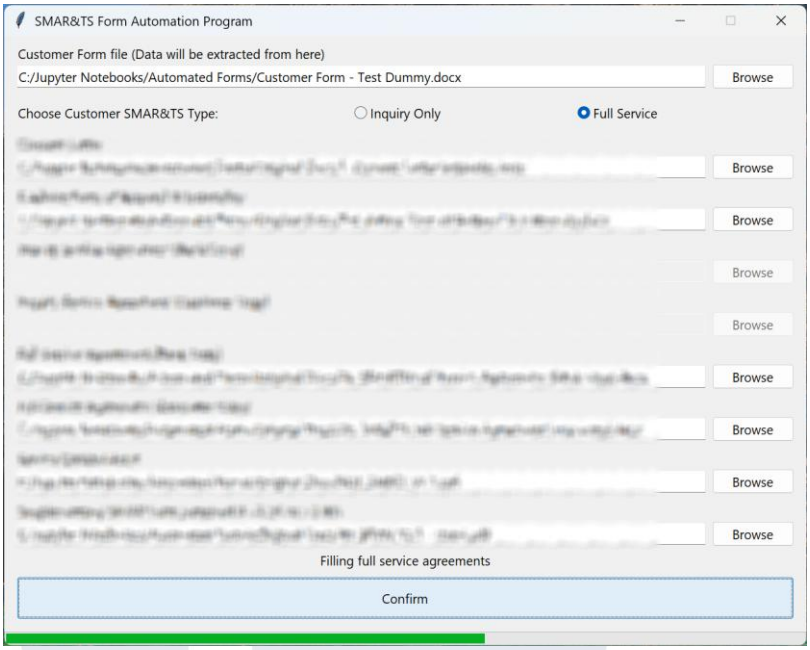
Gambar 3.60 Kode nama baru file formulir yang sudah terisi

Gambar 3.60 menunjukkan kode untuk membuat nama baru bagi semua file formulir yang sudah terisi. Ada variabel “newfoldername” yang dibuat, yaitu nama folder yang akan dibuat untuk menampung semua file formulir yang sudah terisi. Ada juga variabel “newname” yang merupakan sebuah *dictionary* dengan beberapa pasangan *key* dan *value*. Variabel ini berisi semua nama baru untuk setiap jenis formulir.

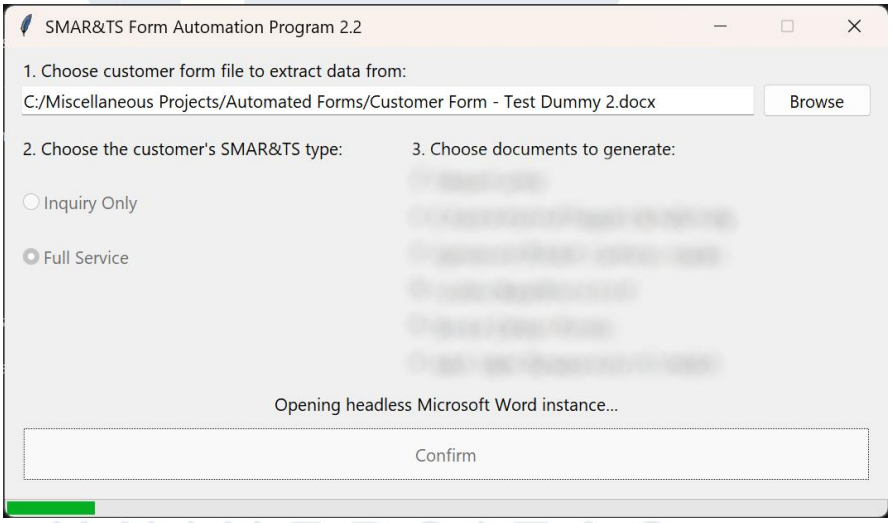
```
1220  def start_main():
1221      threading.Thread(target=main, daemon=True).start()
```

Gambar 3.61 Kode memulai fungsi pengisian formulir pada *thread* baru

Gambar 3.61 menunjukkan kode untuk memulai fungsi pengisian formulir pada *thread* baru. Sebagai bagian dari usaha untuk mempercepat program, semua fungsi pengisian formulir dipindahkan ke *thread* baru agar lepas dari *loop* GUI program. Hasilnya adalah GUI program yang lebih responsif (tidak *hang*) sehingga pengguna program dapat melihat kemajuan proses pengisian formulir secara lebih *real-time*.



Gambar 3.62 Tampilan *user interface* lama



Gambar 3.63 Tampilan *user interface* baru

Gambar 3.62 menunjukkan tampilan *user interface* lama yang dibuat untuk program otomatisasi pengisian formulir. *User interface* ini dibangun dengan *library* tkinter dan terdiri dari beberapa elemen. *User interface* ini memungkinkan pengguna untuk mengarahkan program ke file formulir pendaftaran awal yang sudah diisi nasabah. Namun, *user interface* ini kurang efisien karena juga membutuhkan pengguna untuk mengarahkan program ke setiap file

formulir lainnya, sehingga memakan banyak waktu bagi pengguna untuk memilih file satu demi satu. *User interface* yang lama ini juga dijalankan pada *thread* yang sama dengan fungsi main(), sehingga menjadi *not responding* saat fungsi main() sedang berjalan.

Gambar 3.63 menunjukkan tampilan *user interface* baru yang dibuat untuk program otomatisasi pengisian formulir. Pada bagian atas, ada sebuah tombol "Browse" untuk mengarahkan program ke file formulir pendaftaran awal yang sudah diisi nasabah. Ketika file dipilih, maka *path* ke file tersebut akan dimasukkan ke *widget entry* di sebelah kiri tombol tersebut. Pengguna kemudian dapat memilih jenis SMAR&TS yang diinginkan oleh nasabah (antara *inquiry only* atau *full service*) cukup dengan *widget radio button*. Tergantung jenis SMAR&TS yang dipilih, *widget checkbox* di sisi kanan GUI akan berubah. *Checkbox* tersebut dapat langsung diubah oleh pengguna program untuk memilih formulir yang ingin dihasilkan oleh program. Setelah pengguna selesai menentukan kebutuhannya, maka dapat diklik tombol "Confirm". Ini akan memulai kode pada Gambar 3.61 dan kemajuan proses pengisian formulir dapat dipantau oleh pengguna secara *real-time* melalui teks dan *progress bar* pada bagian bawah GUI.

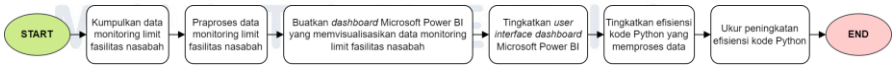
Dengan *user interface* baru ini, pengguna tidak perlu lagi memilih file formulir yang ingin diisi satu demi satu karena sudah diubah dengan *widget* checkbox sederhana dan program sudah mengetahui lokasi file formulir yang sesuai. *User interface* baru ini juga sudah dipisahkan dari *thread* fungsi main(), sehingga *user interface* tetap responsif dan menunjukkan kemajuan pengisian formulir secara *real-time*.

3.3.1.5 Proyek Pengembangan *Dashboard* Kinerja TBSC

Monitoring kinerja merupakan kapabilitas yang sangat dibutuhkan dalam perbankan agar diketahui keadaan dan

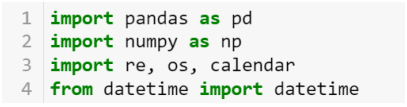
pertumbuhan bisnis, profitabilitas nasabah, dan efektivitas operasional. Departemen TBSC tidak terkecuali dalam hal ini. Kapabilitas monitoring kinerja sebelumnya sudah dikembangkan menggunakan Python untuk memproses dan mengkompilasi data, serta menggunakan Microsoft Power BI untuk menyajikan data dalam bentuk *dashboard* agar cepat dan mudah dipahami. Namun, data yang ditunjukkan pada *dashboard* belum mencakup semua informasi yang dibutuhkan oleh tim TB, terutama data monitoring limit fasilitas nasabah. *Dashboard* yang dibuat juga masih sangat sederhana dengan elemen interaktif yang sedikit sehingga sulit bagi pengguna *dashboard* untuk menavigasi *dashboard* dan menggali informasi lebih dalam mengenai data yang ada. Terakhir, jumlah data yang perlu diproses dan dikonsolidasi oleh Python sangat besar sehingga dibutuhkan cara pemrosesan yang lebih efisien.

Proyek ini bertujuan untuk mengatasi kekurangan yang masih dimiliki *dashboard* kinerja TBSC dengan menambahkan data monitoring limit fasilitas nasabah agar tim TB dapat melihat penggunaan limit fasilitas nasabah dengan cepat dan mudah, meningkatkan *user interface* agar memudahkan navigasi *dashboard* dan memungkinkan penggalian informasi yang lebih dalam, serta meningkatkan efisiensi pemrosesan data *dashboard* agar data dapat diproses setiap bulannya dengan lebih cepat. Gambar 3.64 menunjukkan alur pengerjaan proyek ini.



Gambar 3.64 Alur pengerjaan proyek 3.3.1.5

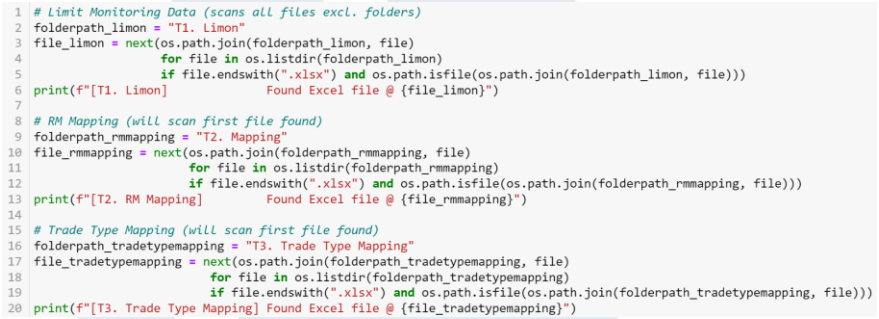
Fase pertama dari pengembangan yang dilakukan adalah menambahkan data monitoring limit fasilitas nasabah yang melibatkan pemrosesan data dan pembuatan halaman *dashboard* baru.



```
import pandas as pd
import numpy as np
import re, os, calendar
from datetime import datetime
```

Gambar 3.65 *Library* yang diimpor untuk pemrosesan data monitoring limit fasilitas nasabah

Gambar 3.65 menunjukkan semua *library* yang diimpor untuk melakukan pemrosesan data monitoring limit fasilitas nasabah, termasuk pandas, numpy, dan re (untuk *regular expression*).



```
# Limit Monitoring Data (scans all files excl. folders)
folderpath_limon = "T1. Limon"
file_limon = next(os.path.join(folderpath_limon, file)
                  for file in os.listdir(folderpath_limon)
                  if file.endswith(".xlsx") and os.path.isfile(os.path.join(folderpath_limon, file)))
print(f"[T1. Limon]              Found Excel file @ {file_limon}")

# RM Mapping (will scan first file found)
folderpath_rmmapping = "T2. Mapping"
file_rmmapping = next(os.path.join(folderpath_rmmapping, file)
                      for file in os.listdir(folderpath_rmmapping)
                      if file.endswith(".xlsx") and os.path.isfile(os.path.join(folderpath_rmmapping, file)))
print(f"[T2. RM Mapping]         Found Excel file @ {file_rmmapping}")

# Trade Type Mapping (will scan first file found)
folderpath_tradetypemapping = "T3. Trade Type Mapping"
file_tradetypemapping = next(os.path.join(folderpath_tradetypemapping, file)
                             for file in os.listdir(folderpath_tradetypemapping)
                             if file.endswith(".xlsx") and os.path.isfile(os.path.join(folderpath_tradetypemapping, file)))
print(f"[T3. Trade Type Mapping] Found Excel file @ {file_tradetypemapping}")
```

Gambar 3.66 Kode membaca data terkait monitoring limit fasilitas nasabah

Gambar 3.66 menunjukkan kode untuk membaca semua data terkait monitoring limit fasilitas nasabah yaitu data limit sendiri, data *mapping* nama nasabah ke RM, dan data *mapping* jenis *trade*. Semua data yang dibaca dalam bentuk file Microsoft Excel.

```
# Read Excel file
df_limon = pd.read_excel(file_limon, engine="calamine")

# Strip column names
df_limon.columns = df_limon.columns.str.strip()

# Rename some columns
df_limon.rename(columns={"CIF / Customer Name" : "Cust. Name",
                         "CIF Number" : "Cust. No"},
          inplace=True)

# Drop some columns
df_limon.drop
              axis=1, inplace=True)

# Capitalize object columns' values
for col in df_limon.select_dtypes(include="object"):
    df_limon[col] = df_limon[col].str.upper()

# Strip string values
df_limon[df_limon.select_dtypes(include="object").columns] = df_limon.select_dtypes(include="object").apply(lambda col:
```

Gambar 3.67 Kode praproses data limit nasabah

Gambar 3.67 menunjukkan kode untuk praproses data limit nasabah. Hal pertama yang dilakukan adalah membaca file Excel untuk menyimpan kontennya sebagai *dataframe* Pandas. Nama setiap kolom kemudian dilakukan *stripping* untuk memastikan tidak ada karakter tersembunyi. Kolom nama nasabah dan nomor nasabah

diubah nama agar menjadi nama standarnya, lalu beberapa kolom dihapus karena tidak relevan dengan kebutuhan analisis yang dilakukan. Semua nilai yang berupa *string* diubah menjadi huruf besar, kemudian dilakukan *stripping* juga untuk memastikan penghapusan semua karakter tersembunyi jika ada.

```python
# Read Excel file
df_rmmapping = pd.read_excel(file_rmmapping, engine="calamine")

# Strip column names
df_rmmapping.columns = df_rmmapping.columns.str.strip()

# Rename some columns
df_rmmapping.rename(columns={"CIF" : "Cust. No",
                             "AO_NAME" : "RM Name"},
                             inplace=True)

# Rearrange column order
df_rmmapping = df_rmmapping[["Cust. No", "RM Name"]]

# Capitalize all object columns' values
for col in df_rmmapping.select_dtypes(include="object"):
    df_rmmapping[col] = df_rmmapping[col].str.upper()
```

Gambar 3.68 Kode praproses data *mapping* RM

```python
# Read Excel file
df_trademapping = pd.read_excel(file_tradetypemapping, engine="calamine")

# Strip column names
df_trademapping.columns = df_trademapping.columns.str.strip()

# Capitalize all object columns' values
for col in df_trademapping.select_dtypes(include="object"):
    df_trademapping[col] = df_trademapping[col].str.upper()

# Strip string values
df_trademapping[df_trademapping.select_dtypes(include="object").columns] = df_trademapping.select_dtypes(include="object'
```

Gambar 3.69 Kode praproses data *mapping trade*

Gambar 3.68 menunjukkan kode untuk praproses data *mapping* nama *relationship manager* ke nama nasabah, sedangkan Gambar 3.69 menunjukkan kode praproses data *mapping* jenis *trade*. Langkah-langkah praproses yang dilakukan mirip dengan praproses data limit pada Gambar 3.67 dengan tujuan untuk menstandarkan bentuk data.

```python
# Combine all dataframes
df_combined = df_limon.copy().merge(df_rmmapping, on="Cust. No", how="left") \
              .merge(df_trademapping, on="Facility Type", how="left")
```

Gambar 3.70 Kode penggabungan data terkait monitoring limit fasilitas nasabah

Setelah semua data yang dibutuhkan telah dibersihkan, maka dapat digabungkan menjadi satu. Gambar 3.70 menunjukkan kode untuk melakukan penggabungan semua data menggunakan fungsi pd.DataFrame.merge(). Penggabungan pertama yang dilakukan

adalah antara data limit dengan data *mapping* RM berdasarkan nomor nasabah. Penggabungan kedua yang dilakukan adalah antara data limit dengan data *mapping* jenis *trade* berdasarkan jenis fasilitas.

```
# Trade Type per CIF
dict_cif_tradetypes = df_combined.groupby("Cust. No")["Trade Type"].unique().to_dict()
```

Gambar 3.71 Kode pembuatan *dictionary* jenis *trade* berdasarkan nomor nasabah

Gambar 3.71 menunjukkan kode pembuatan *dictionary* untuk jenis *trade* berdasarkan nomor nasabah. *Dictionary* ini digunakan di kode pada Gambar 3.72.

```
# Get Trade Type
for cif, tradetypes in dict_cif_tradetypes.items():
    if "TRADE ASSET" in tradetypes:
        df_combined.loc[df_combined["Cust. No"] == cif, "Trade Type"] = "Trade Asset"
    elif "TRADE CONTINGENT" in tradetypes:
        df_combined.loc[df_combined["Cust. No"] == cif, "Trade Type"] = "Trade Contingent"
    else:
        df_combined.loc[df_combined["Cust. No"] == cif, "Trade Type"] = "Non-Trade"
```

Gambar 3.72 Kode menambahkan jenis *trade* per nasabah

Gambar 3.72 menunjukkan kode untuk menambahkan kolom jenis *trade* per nasabah berdasarkan *dictionary* yang telah dibuat. Kolom ini akan memungkinkan filter data berdasarkan jenis *trade* pada *dashboard*.

```
# Multiply limit values with exchange rate to get IDR values
df_combined["IDR Limit"] = df_combined["Limit / Facility Amount"] * df_combined["Exchange Rate H-1"]
df_combined["IDR OS H-1"] = df_combined["Outstanding Amount H-1"] * df_combined["Exchange Rate H-1"]
df_combined["IDR Unused H-1"] = df_combined["IDR Limit"] - df_combined["IDR OS H-1"]

# Create limit usage percentage column
df_combined["Limit Usage"] = df_combined["Outstanding Amount H-1"] / df_combined["Limit / Facility Amount"]
```

Gambar 3.73 Kode penambahan kolom

Gambar 3.73 menunjukkan kode penambahan beberapa kolom yaitu limit fasilitas dalam mata uang rupiah, jumlah *outstanding* dalam mata uang rupiah, dan jumlah sisa limit fasilitas yang tidak digunakan dalam mata uang rupiah. Ketiga kolom ini dikalkulasi menggunakan kolom *exchange rate*. Ada juga kolom penggunaan limit yang ditambahkan untuk menunjukkan penggunaan limit dalam bentuk persentase dari 0-100% (di mana 0% berarti fasilitas sama sekali tidak digunakan dan 100% berarti fasilitas sepenuhnya digunakan).

```python
# Rename columns
df_combined.rename(columns={"Omnibus Group"           : "Omni Group",
                            "Limit Number"            : "Limit No",
                            "Limit / Facility Amount" : "Limit",
                            "Exchange Rate H-1"       : "FX Rate H-1",
                            "Outstanding Amount H-1"  : "OS H-1",
                            "Unused Amount H-1"       : "Unused H-1",
                            "Limit Usage"             : "Usage %"},
                            inplace=True)
```

Gambar 3.74 Kode penamaan ulang kolom

Gambar 3.74 menunjukkan kode untuk melakukan penamaan ulang beberapa kolom agar lebih singkat.

Gambar 3.75 Kode penggantian nilai kelompok omnibus

Limit fasilitas nasabah tidak selalu berdiri sendiri. Terkadang, ada limit fasilitas yang merupakan pecahan dari sebuah limit induk. Limit fasilitas yang bersifat seperti ini disebut sebagai limit omnibus. Ada juga limit fasilitas yang dibagi antara beberapa nasabah atau entitas, disebut sebagai limit JBGL (*joint borrower's group limit*). Jika sebuah limit fasilitas adalah limit omnibus atau JBGL, maka akan diindikasikan melalui kolom "Omni Group". Contohnya, nilai "1" berarti limit fasilitas tersebut merupakan bagian dari limit omnibus pertama, dan nilai "2" berarti limit fasilitas tersebut merupakan bagian dari limit omnibus kedua. Gambar 3.75 menunjukkan penggantian nilai kelompok omnibus ini agar nantinya dapat ditunjukkan sebagai satu kelompok pada *dashboard*.

```python
df_combined["Limit No"] = df_combined["Limit No"].fillna("PARENT")

df_combined['Limit No'] = df_combined['Limit No'].apply(
    lambda x: f'PARENT - {x}' if pd.notna(x) and re.match(r'^\d{6}$', str(x)) else x
)
```

Gambar 3.76 Kode penandaan limit induk

Gambar 3.76 menunjukkan kode untuk menandakan limit yang berupa induk dari limit lain, jika limit tersebut merupakan sebuah limit omnibus atau JBGL. Penandaan dilakukan dengan menambahkan *string* "PARENT" pada awal nomor limit.

```
df_combined = df_combined[["Cust. No", "Cust. Name",
                           "Omni Group", "Limit No", "Facility Type", "CCY",
                           "Limit", "FX Rate H-1", "OS H-1", "Unused H-1",
                           "RM Name", "Trade Type", "IDR Limit",
                           "IDR OS H-1", "IDR Unused H-1", "Usage %"]]
```

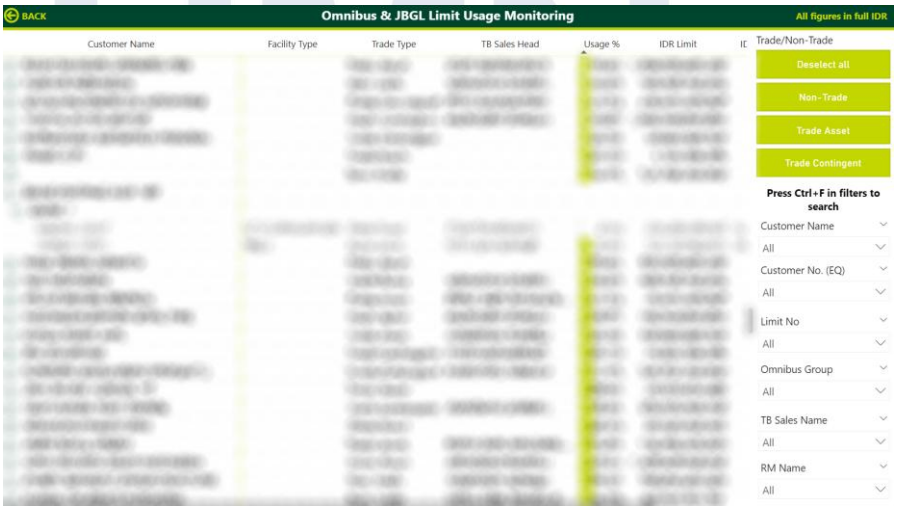
Gambar 3.77 Kode pengambilan dan urutan kolom

Gambar 3.77 menunjukkan kode untuk mengambil kolom yang sesuai dan mengubah urutan kolom agar lebih mudah untuk dibaca.

```
# Output final Parquet file
print("Outputting Parquet file...")
df_combined.to_parquet("Processed Limon Data.parquet",
                       index=False)
print("All done!")
```

Gambar 3.78 Kode output data monitoring limit fasilitas nasabah yang telah diproses

Gambar 3.78 menunjukkan kode untuk mengoutput data monitoring limit fasilitas nasabah yang telah diproses. Output yang dihasilkan dalam bentuk Parquet.



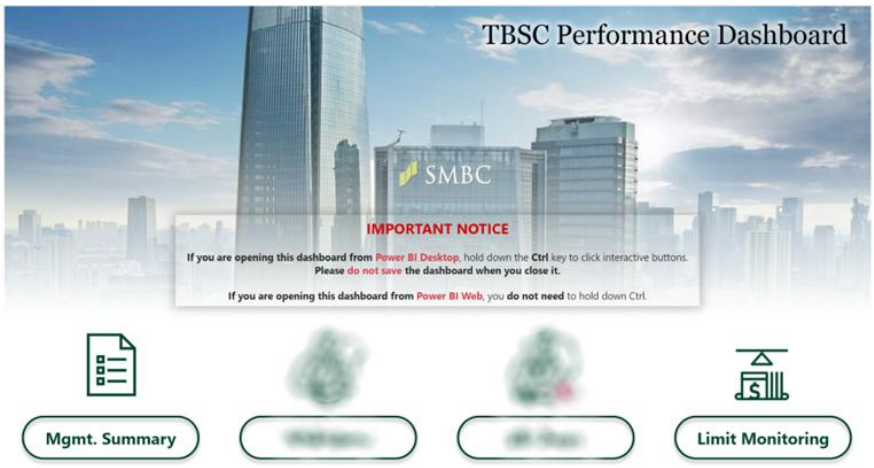
Gambar 3.79 *Dashboard* monitoring limit fasilitas nasabah

Gambar 3.79 menunjukkan *dashboard* yang dibuat untuk menyajikan data monitoring limit fasilitas nasabah. Visualisasi utama yang digunakan adalah matriks dengan data sebagai berikut:

1. Baris: Nama Nasabah, Kelompok Omni, Nomor Limit
2. Kolom: Tidak ada
3. Nilai: Jenis Fasilitas, Jenis *Trade*, Nama *TB Sales*, Persentase Penggunaan Fasilitas, Jumlah Limit dalam IDR, Jumlah Limit yang Tidak Digunakan dalam IDR

Dengan visualisasi matriks ini, semua limit dari satu nasabah muncul di dalam kelompok nama nasabah tersebut. Setiap nama nasabah dapat digali lebih dalam untuk melihat limit individual. Limit yang merupakan bagian dari satu limit omnibus atau JBGL induk juga dikelompokkan. Untuk setiap limit, dapat dilihat jenis fasilitasnya dan jenis *trade*nya, persentase limit yang terpakai, jumlah limit dalam IDR, dan jumlah limit yang tidak terpakai dalam IDR. Nama *TB Sales* dari setiap nasabah juga terlihat. Ada beberapa filter yang diletakkan di sebelah kanan *dashboard* untuk memungkinkan akses lebih cepat terhadap informasi yang dibutuhkan pengguna, yaitu jenis *trade*, nama nasabah, nomor nasabah, nomor limit, kelompok omnibus, nama *TB Sales*, dan nama RM.

Fase kedua dari pengembangan yang dilakukan adalah meningkatkan *user interface* agar memudahkan navigasi *dashboard* dan memungkinkan penggalian informasi yang lebih dalam. Navigasi *dashboard* ditingkatkan dengan membuat halaman awal dan halaman menu untuk mempercepat akses ke halaman-halaman lainnya.



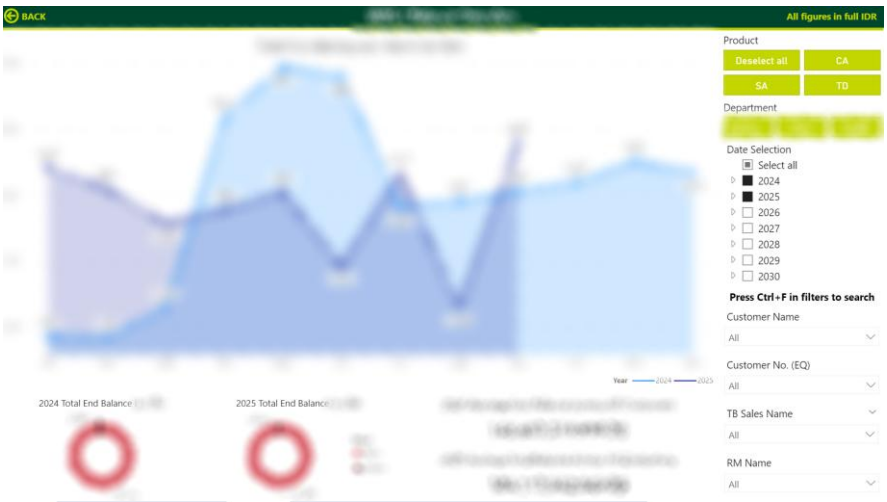
Gambar 3.80 Halaman Awal *Dashboard*

Gambar 3.80 menunjukkan halaman awal *dashboard* yang dibuat untuk mempermudah navigasi *dashboard*. Dari halaman ini, pengguna dapat memilih halaman selanjutnya yang ingin dilihat. Tombol "Management Summary" dan "Limit Monitoring" akan membawa pengguna ke *dashboard* yang sesuai, sedangkan dua tombol lainnya akan membawa pengguna ke menu yang lebih dalam untuk setiap departemen.

Gambar 3.81 Halaman Menu Dalam

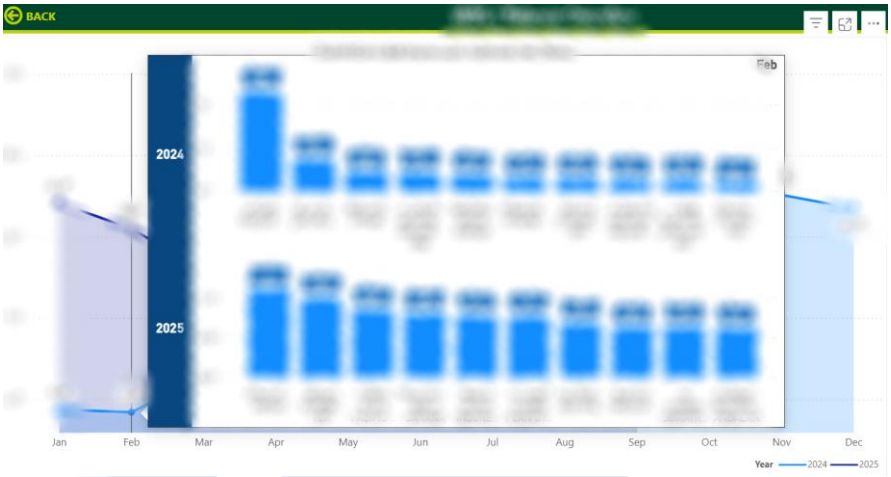
Gambar 3.81 menunjukkan halaman menu dalam yang muncul jika pengguna menekan salah satu tombol menu pada halaman

awal. Halaman menu dalam ini berisi beberapa tombol untuk membawa pengguna ke *dashboard* yang diinginkan.



Gambar 3.82 Halaman *Balance Overview*

Gambar 3.82 menunjukkan halaman *balance overview*. Gambar ini merupakan sampel dari desain ulang yang dilakukan untuk memudahkan pengguna dalam menggali data lebih dalam. Hal ini dicapai melalui penambahan elemen filter pada sisi sebelah kanan halaman yang memungkinkan filter data menurut produk, departemen, tanggal, nama nasabah, nomor nasabah, nama *TB Sales*, dan nama RM tertentu. Ada juga penambahan elemen interaktif lainnya berupa sebuah tombol “Back” untuk membawa pengguna kembali ke halaman menu secara instan. *Dashboard* selain *balance overview* telah diperbarui juga dengan elemen-elemen serupa.



Gambar 3.83 *Tooltip* 10 Nasabah Terbesar

Gambar 3.83 menunjukkan *tooltip* 10 nasabah terbesar yang muncul ketika pengguna meletakkan *pointer* mereka di atas grafik garis jumlah *balance* nasabah. *Tooltip* ini menunjukkan 10 nasabah terbesar untuk 2 tahun terakhir yang diseleksi pada filter tanggal sehingga memungkinkan pengguna untuk mengetahui pergerakan *balance* nasabah terbesar.

Fase ketiga dari pengembangan yang dilakukan adalah meningkatkan efisiensi pemrosesan data agar menjadi lebih cepat sehingga mengurangi waktu yang dibutuhkan untuk memperbarui data *dashboard* setiap bulannya. Hal ini dilakukan melalui dua perubahan yaitu pengalihan dari menggunakan *engine* openpyxl ke *engine* python-calamine untuk membaca file mentah yang berbentuk Microsoft Excel, serta pengalihan dari menggunakan file Microsoft Excel ke file Parquet untuk data output.

Gambar 3.84 Perbandingan pemrosesan data mentah Microsoft Excel dengan *engine* openpyxl vs. python-calamine

Gambar 3.84 menunjukkan perbandingan pemrosesan data mentah Microsoft Excel menggunakan *engine* openpyxl dan python-calamine. Dengan *engine* openpyxl, waktu yang dibutuhkan untuk selesai memproses semua data mentah adalah 12.63 detik, sedangkan dengan *engine* python-calamine waktu yang dibutuhkan hanya 7.55 detik. Perbandingan ini menunjukkan *engine* python-calamine memotong waktu yang dibutuhkan untuk memproses semua data mentah sebesar 1.67x. Peningkatan ini akan sangat terasa dengan semakin besarnya data mentah setiap bulannya.

```
print("Outputting Microsoft Excel file...")
start = time.perf_counter()
df_combined.to_excel("Processed Profit Data (Benchmarking).xlsx", index=False)
end = time.perf_counter()
print("All done!")
print(f"Time taken: {end - start:.2f} seconds")
print(f"File size: {os.path.getsize('Processed Profit Data (Benchmarking).xlsx') / 1048576:.2f} MB")
```

```
Outputting Microsoft Excel file...
All done!
Time taken: 92.59 seconds
File size: 19.50 MB
```

```
print("Outputting Parquet file...")
start = time.perf_counter()
df_combined.to_parquet("Processed Profit Data (Benchmarking).parquet", index=False)
end = time.perf_counter()
print("All done!")
print(f"Time taken: {end - start:.2f} seconds")
print(f"File size: {os.path.getsize('Processed Profit Data (Benchmarking).parquet') / 1048576:.2f} MB")
```

```
Outputting Parquet file...
All done!
Time taken: 1.20 seconds
File size: 1.19 MB
```

Gambar 3.85 Perbandingan proses output file Microsoft Excel vs. Parquet

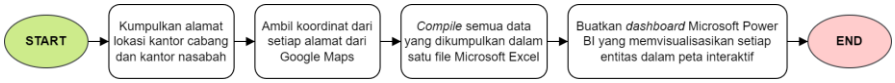
Gambar 3.85 menunjukkan perbandingan proses output file Microsoft Excel dan Parquet. Jika data yang sudah terproses dioutput sebagai file Microsoft Excel, maka waktu yang dibutuhkan untuk menyelesaikan output adalah 92.59 detik, sedangkan jika dioutput

sebagai file Parquet maka waktu yang dibutuhkan adalah 1.20 detik. Output dengan file Parquet menunjukkan peningkatan kecepatan sebesar 77x. Selain waktu yang dibutuhkan, ukuran file yang dihasilkan juga turun drastis dengan Parquet menjadi 1.19 MB dari 19.50 MB dengan Microsoft Excel sehingga ukuran file Parquet 16x lebih kecil daripada file Microsoft Excel.

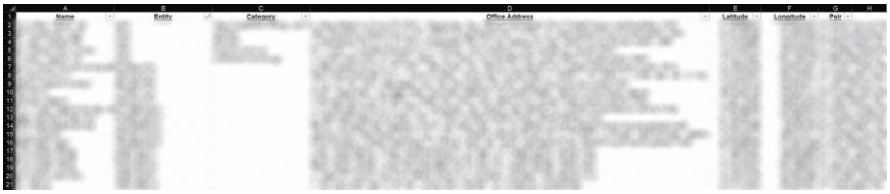
3.3.1.6 Proyek *Geomapping* Nasabah

SMBCI memiliki jaringan kantor cabang yang tersebar di Indonesia untuk memfasilitasi kenyamanan dan kemudahan bertransaksi nasabahnya. Namun, pembukaan kantor cabang di lokasi tertentu perlu dipikirkan secara matang, tidak hanya dari lokasinya tetapi juga kedekatan lokasi tersebut dengan nasabah-nasabah SMBCI. Investasi pembukaan kantor cabang di lokasi yang strategis dapat membantu meningkatkan profitabilitas bank dan sekaligus meningkatkan kualitas pengalaman nasabah dalam bertransaksi bersama SMBCI.

Proyek ini bertujuan untuk mendata lokasi beberapa nasabah prioritas SMBCI dan memvisualisasikan setiap lokasi dalam sebuah peta grafik untuk mendukung bank dalam membuat keputusan untuk membuka kantor cabang bank baru. Peta ini juga perlu dilengkapi dengan lokasi beberapa entitas lain, yakni kantor cabang SMBCI terdekat yang sudah berdiri saat ini, kantor cabang bank kompetitor terdekat, serta mesin ATM bank kompetitor terdekat, agar menjadi poin pertimbangan tambahan dalam diskusi untuk menilai potensi dan risiko pembukaan cabang baru. Gambar 3.86 menunjukkan alur pengerjaan proyek ini.



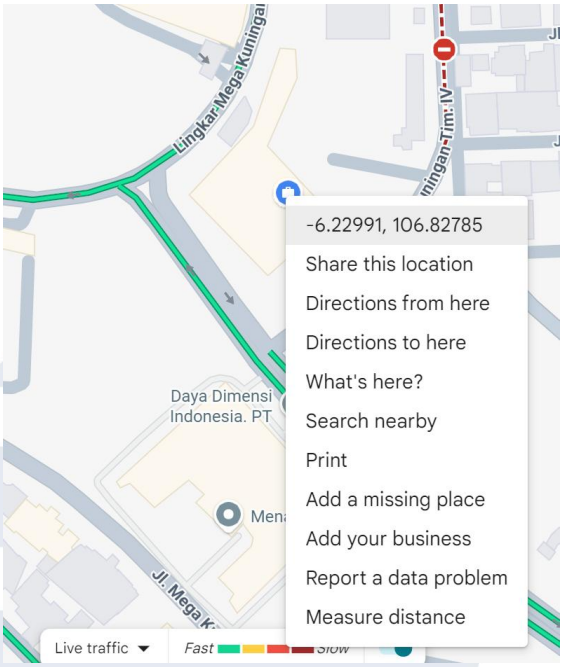
Gambar 3.86 Alur pengerjaan proyek 3.3.1.6



Gambar 3.87 File Microsoft Excel yang Dibuat untuk *Geomapping* Nasabah

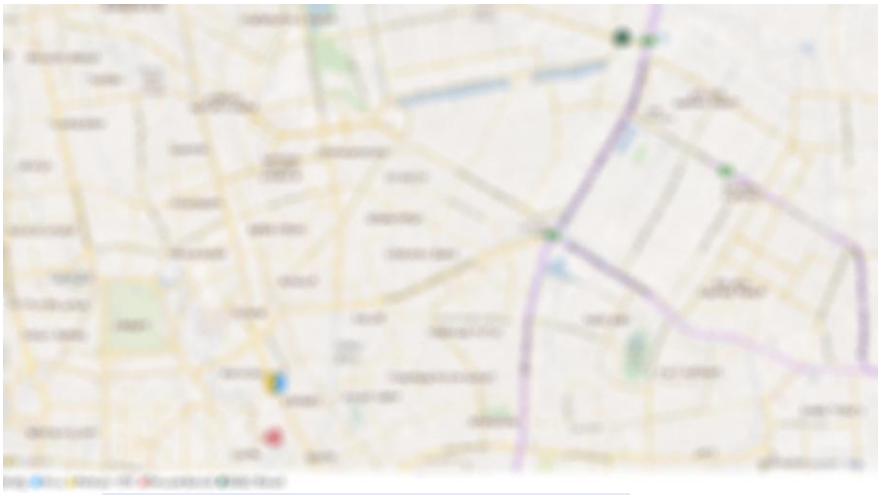
Gambar 3.87 menunjukkan file Microsoft Excel yang dibuat untuk *geomapping* lokasi nasabah. *Sheet* ini terdiri dari kolom:

1. Nama entitas: Nama nasabah, perusahaan atau kantor yang terdata.
2. Jenis entitas: Apakah entitas tersebut merupakan nasabah SMBCI, cabang SMBCI terdekat, cabang bank kompetitor terdekat, atau ATM bank kompetitor terdekat.
3. Kategori: Diisi dengan nama industri jika entitas berjenis nasabah. Untuk jenis entitas lain tidak diisi.
4. Alamat: Alamat gedung atau kantor entitas.
5. Angka *latitude* dan *longitude*: Lokasi gedung atau kantor entitas secara angka garis bujur dan lintang. Kedua angka tersebut didapatkan dari Google Maps.
6. Pasangan: Hubungan entitas dengan nasabah.



Gambar 3.88 Koordinat dari Google Maps

Walaupun visualisasi peta di Microsoft Power BI dapat menggunakan alamat untuk mendapatkan lokasi suatu entitas, koordinat masih menjadi pilihan yang paling akurat karena berupa angka absolut dan bukan alamat yang dapat memiliki ambiguitas karena ada similaritas dengan alamat lain. Gambar 3.88 menunjukkan koordinat *latitude* dan *longitude* yang dapat diambil dari Google Maps (koordinat Menara SMBC ditunjukkan pada gambar) dengan mengklik kanan pada peta di posisi yang ingin diketahui. Koordinat yang didapatkan dimasukkan ke dalam file Microsoft Excel pada Gambar 3.87 di kolom “Latitude” dan “Longitude”.



Gambar 3.89 Peta *Geomapping* Nasabah

File Microsoft Excel yang sudah selesai diisi kemudian dimasukkan ke dalam Microsoft Power BI. Gambar 3.89 menunjukkan peta *geomapping* yang dibuat untuk menunjukkan lokasi kantor pusat nasabah dengan lokasi lainnya yaitu kantor cabang SMBCI terdekat, kantor cabang bank kompetitor terdekat, dan ATM bank kompetitor terdekat. Setiap entitas dalam peta diberikan warna unik untuk memudahkan pemahaman peta.

Visualisasi *geomapping* yang dihasilkan memberikan gambaran menyeluruh mengenai kondisi geografis yang relevan bagi keputusan pembukaan kantor cabang baru. Dengan menggabungkan data lokasi nasabah prioritas, posisi kantor cabang SMBCI yang sudah ada saat ini, serta kompetitor di area sekitar, analisis dapat dilakukan secara lebih komprehensif. Peta ini membantu mengidentifikasi area dengan potensi aktivitas bisnis yang tinggi, menilai tingkat persaingan, serta mengukur jarak layanan terhadap nasabah prioritas.

3.3.2 Kendala yang Ditemukan

Beberapa kendala ditemukan selama praktik kerja magang ini yaitu sebagai berikut:

1. Mahasiswa yang melaksanakan praktik kerja magang di SMBCI dianggap sebagai pihak eksternal sehingga tidak bisa mengakses

data yang tersimpan di *network-attached storage* (NAS) dalam jaringan internal SMBCI. Kebijakan ini diimplementasi dengan tujuan untuk mencegah pembocoran data dan menjaga keamanan informasi, terlebihnya karena industri perbankan adalah industri dengan banyak data sensitif sehingga memiliki regulasi ketat. Namun, pembatasan ini berdampak pada pengerjaan proyek magang karena mahasiswa tidak bisa mengakses data yang dibutuhkan secara langsung.

2. Proses adaptasi terhadap berbagai macam struktur data dan sistem internal yang digunakan oleh SMBCI menjadi salah satu tantangan. Setiap sistem memiliki istilah berbeda dan struktur data yang unik. Hal ini memperlambat pengerjaan proyek magang karena diperlukan waktu tambahan untuk memahami bagaimana data terstruktur agar dapat distandarisasi untuk digunakan dalam proyek.

3.3.3 Solusi atas Kendala yang Ditemukan

1. Mahasiswa perlu menanyakan anggota tim yang merupakan karyawan tetap yang memiliki akses ke NAS SMBCI untuk membantu mengambil data yang dibutuhkan dan mentransfer data tersebut ke laptop kantor mahasiswa.
2. Untuk mengatasi tantangan dalam memahami struktur data dan sistem internal, ditinjau contoh penggunaan data pada proyek-proyek sebelumnya, serta dilakukan diskusi dengan anggota tim yang lebih berpengalaman untuk mendapatkan penjelasan yang lebih jelas. Mahasiswa juga membuat catatan internal mengenai definisi data dan hubungan antar-kolom sehingga proses pengolahan data selanjutnya dapat dilakukan lebih cepat dan lebih akurat.