BAB III
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Selama program magang di PT Menara Indonesia, *jobdesk* magang yang diberikan peneliti adalah sebagai *data scientist intern* yang berada pada divisi *Data Science & AI Programer* di dalam departemen *Tech*. Selama kegiatan magang berlangsung, Bapak M. Andrianto berperan sebagai mentor atau atasan langsung. Selain itu, terdapat pengawas project *Data Science* yaitu Muhammad Andrianto Abdillah selaku supervisor, dan Untuk pelaporan kinerja dan output kerja saat magang akan di pantau oleh *Project Manager Officer* dengan Rolan.

Divisi *Data Science* di PT Menara Indonesia memiliki tanggung jawab utama dalam mengembangkan model AI untuk mendukung berbagai inisiatif strategis perusahaan, termasuk proyek *credit assesment*. Salah satu langkah penting yang dilakukan adalah *data preprocessing*, yaitu proses pembersihan dan pengolahan data agar siap digunakan untuk analisis lebih lanjut. Dengan memastikan kualitas data yang tinggi, divisi ini dapat menghasilkan model klasifikasi yang akurat, yang pada gilirannya membantu meminimalkan risiko kesalahan dalam pengambilan keputusan terkait kredit.

Divisi ini juga fokus pada pengembangan model klasifikasi dan algoritma khusus yang dirancang untuk proyek *credit assesment*. Model-model ini memungkinkan evaluasi risiko kredit dengan lebih mendalam, membantu klien perusahaan mengambil keputusan berbasis data dalam hal pemberian kredit. Setelah model dikembangkan, tim mengintegrasikannya ke dalam *server* perusahaan menggunakan *Django REST Framework* dan melakukan *deployment* dengan Streamlit, sehingga memungkinkan akses *real-time* oleh berbagai pengguna yang membutuhkan informasi tersebut.

Selain itu, divisi ini bertanggung jawab dalam pembuatan *dashboard* interaktif menggunakan alat analitik seperti Tableau dan Power BI, yang memfasilitasi visualisasi data secara intuitif untuk berbagai pemangku kepentingan.

Di samping pengembangan model klasifikasi, divisi ini juga tengah membangun *AI Chatbot* RAG (*Retrieval-Augmented Generation*) dengan Python, yang dirancang untuk memberikan respons otomatis berdasarkan data internal. Untuk memastikan pengelolaan data yang optimal.

3.2 Tugas dan Uraian Kerja Magang

PT. Menara Indonesia memiliki fokus utama kegiatan magang adalah pada pengembangan dan pemrograman model analitik untuk klasifikasi risiko kredit berdasarkan karakteristik pribadi pelanggan. Model ini dirancang menggunakan data yang relevan untuk membangun prototipe dan mengembangkan kode yang mendukung evaluasi kredit. Table 3.1 menunjukan Aktivitas yang dilakukan oleh mahasiswa dalam program ini mencakup berbagai tugas dan proyek yang dirancang untuk mengembangkan keterampilan teknis dan profesional, sekaligus memberikan pengalaman dunia kerja yang nyata.

Tabel 2. 1 Tabel Realisasi Kerja Magang

NO	Judul Aktivitas	Mulai	Selesai
1.	**Mempelajari SOP Divisi Data Science**		
	*Mematuhi persyaratan Standard Operating Procedur*e (SOP) untuk Divisi *Data Scientist*.	04/08/23	16/08/23
2.	**Melakukan *data preprocessing*, dan *data exploration* pada dataset nasabah**		
	Melakukan *Feature selection*, dan *remove duplicate column*	19/08/23	20/08/23
	Melakukan Pengecekan *Missing value*, *unique Value* pada dataset nasabah	21/08/24	21/08/24
	Implementasi *MinMax Scaler* & *Encoding*	26/08/24	27/08/24
	Melakukan eliminasi *outliers* menggunakan IQR Method	28/08/24	29/08/24
3.	**Melakukan *Cluster*ing dataset Nasabah**		

NO	Judul Aktivitas	Mulai	Selesai
	Melakukan *Cluster*ing menggunakan Algoritma K-Means	02/09/24	03/09/24
	Evaluasi model *cluster* menggunakan *Elbow Method*	04/09/24	05/09/24
	Mendefinisikan label *clustering*	06/09/24	06/09/24
4.	**Melakukan perancangan klasifikasi *credit assesment***		
	Screening Model Dengan Implementasi *Library Pycaret*	09/09/24	10/09/24
	Membangun model klasifikasi dengan algoritma KNN	11/09/24	12/09/24
	Membangun model klasifikasi dengan algoritma *Decision Tree*	13/09/24	13/09/24
5.	**Melakukan perancangan model *Retrieval-Augmented Generation*(RAG) dengan Ollama**		
	Membangun *script input pdf text splitting* untuk pdf	16/09/24	17/09/24
	Membangun *Vector database* untuk pdf	18/09/24	18/09/24
	Membangun model RAG dengan model Ollama	19/09/24	20/09/24
6.	**Melakukan Deployment model RAG dengan streamlit**		
	Membangun function *extract_model_names()*	23/09/24	24/09/24
	Membangun function *create_avector_db()*	25/09/24	26/09/24
	Membangun function *process_question()*	27/09/24	30/09/24
	Membangun function *extract_all_pages_as_ images()*	01/10/24	02/10/24
	Membangun function *delete_vector_db()*	03/10/24	04/10/24
7.	**Melakukan perancangan *Model* RAG dengan Gpt 4.o**		
	Membangun *text splitting* untuk pdf	07/10/24	08/10/24
	Membangun *Vector Database*	09/10/24	10/10/24
	Membangun model RAG dengan model Chat gpt 4.o	10/10/24	14/10/24
8.	**Melakukan perancangan API model *credit assesment* menggunakan Django Rest API**		

NO	Judul Aktivitas	Mulai	Selesai
	Membangun *pipeline preprocessing*	15/10/24	16/10/24
	Membangun *Serializer*	17/10/24	19/10/24
	Membangun *Request Handler API*	18/10/24	23/10/24
9.	**Melakukan Perancangan Website klasifikasi nasabah credit assesment menggunakan Streamlit**		
	Membangun *Fungsi get_prediction()*	24/10/24	28/10/24
	Membangun *Fungsi predict_from_file()*	29/10/24	31/10/24
	Membangun Tampilan Input Manual	01/11/24	05/11/24
	Membangun tampilan halam Unggah *File*	06/11/24	08/11/24
10.	**Melakukan perancangan *Dashboard Character credit assesment* menggunakan Power BI**		
	Membangun Tampilan *Dashboard* Situasi Lingkungan Nasabah	11/11/23	15/11/23
	Membangun Tampilan *Dashboard Financial* Nasabah	18/11/23	25/11/23
	Membangun Tampilan *Dashboard* Kesehatan Nasabah	25/11/24	4/11/24

3.2.1 Mempelajari SOP Divisi Data Science

mahasiswa di PT Menara Indonesia diwajibkan untuk mempelajari dan memahami Standar Operasional Prosedur (SOP) yang telah ditetapkan sebagai langkah awal sebelum melanjutkan ke proyek perusahaan. SOP ini mencakup berbagai aspek penting dalam *Data Science*, mulai dari penulisan kode yang mengikuti standar PEP8 hingga penggunaan library yang relevan, seperti *Faker* untuk menghasilkan *dummy data* dan Scrapy untuk *web scraping*. Memahami prinsip-prinsip dasar ini sangat krusial untuk memastikan bahwa mahasiswa tidak hanya dapat berkontribusi secara efektif dalam tim, tetapi juga menjamin kualitas dan konsistensi dalam pengolahan data yang digunakan dalam proyek.

Selain itu, mahasiswa perlu menguasai metode transformasi data menggunakan *pipeline*, serta berbagai *library* yang mendukung seperti *LazyPredict* dan PyCaret. Pengetahuan tentang cara menyimpan model menggunakan joblib atau pickle juga sangat penting untuk efisiensi pengelolaan model yang telah dibangun. Selanjutnya, mahasiswa diharapkan dapat mengintegrasikan model yang telah dibuat ke dalam bentuk API menggunakan Django REST, serta menyajikannya secara visual dengan Streamlit. Dengan mengikuti SOP yang telah ditetapkan, mahasiswa akan lebih siap menghadapi tantangan dalam proyek yang akan datang, serta dapat memberikan kontribusi yang lebih bernilai bagi perusahaan.

Sebelum memasuki fase pemodelan, mahasiswa diharapkan untuk melakukan *Explanatory data analysis (EDA*) dan memeriksa asumsi-asumsi yang berkaitan dengan validitas dan keandalan hasil analisis regresi. Proses ini mencakup pemeriksaan asumsi normalitas, linearitas, independensi, homoskedastisitas, dan nonmultikolineritas. Selain itu, pemahaman tentang metode penentuan signifikansi variabel, seperti VIF, *ablation study*, dan *feature importance*, sangat penting untuk mengevaluasi relevansi setiap variabel dalam model. Dengan demikian, melalui pemahaman yang mendalam terhadap teori dan penerapan praktis, mahasiswa tidak hanya akan memperkuat kemampuan analisis data, tetapi juga mendukung pengambilan keputusan yang lebih baik dalam konteks bisnis kredit di PT Menara Indonesia.

3.2.2 Melakukan data preprocessing, dan data exploration pada dataset nasabah

Tahapan berikut merupakan tahapan setelah mengetahui masalah perancangan sistem credit assesment berbasis data mining, tahapan awal pengerjaan proyek credit assessment telah dilalui sebelumnya yaitu memahami kebutuhan dan tujuan bisnis melalui tahapan *Business Understanding* dalam kerangka kerja CRISP-DM (*Cross-Industry Standard Process for Data Mining*) [8]. Tahap ini dapat dilihat pada gambar 3.1, *Business Understanding merupakan tahapan yang* sangat penting untuk mengidentifikasi tujuan dan permasalahan yang ingin

diselesaikan dengan data. Dalam konteks pengembangan sistem credit assessment, tujuan dari pembuatan model adalah mengembangkan model klasifikasi yang dapat mengidentifikasi tingkat risiko kredit nasabah, yang akan membantu lembaga keuangan dalam pengambilan keputusan yang lebih akurat dan cepat.

Setelah tujuan bisnis dipahami, tahapan selanjutnya adalah *Data Understanding* yang merupakan bagian dari sub bab ini, Pengumpulan data dan eksplorasi awal untuk memahami karakteristik dataset yang ada. Di sini, dilakukan analisis awal terhadap data pada seperti riwayat kredit, dan faktor-faktor lain yang dapat mempengaruhi penilaian risiko kredit, serta untuk mengidentifikasi masalah potensial dalam data seperti nilai yang hilang atau distribusi yang tidak merata.

Setelah data dianalisis, tahapan yang dilakukan mahasiswa magang adalah *Data Preparation*, yang mencakup pembersihan data (*data cleaning*), penanganan nilai yang hilang, transformasi variabel kategori menjadi numerik, normalisasi data, dan penghapusan outlier. Tahap ini bertujuan untuk memastikan data siap digunakan dalam proses Modeling selanjutnya.

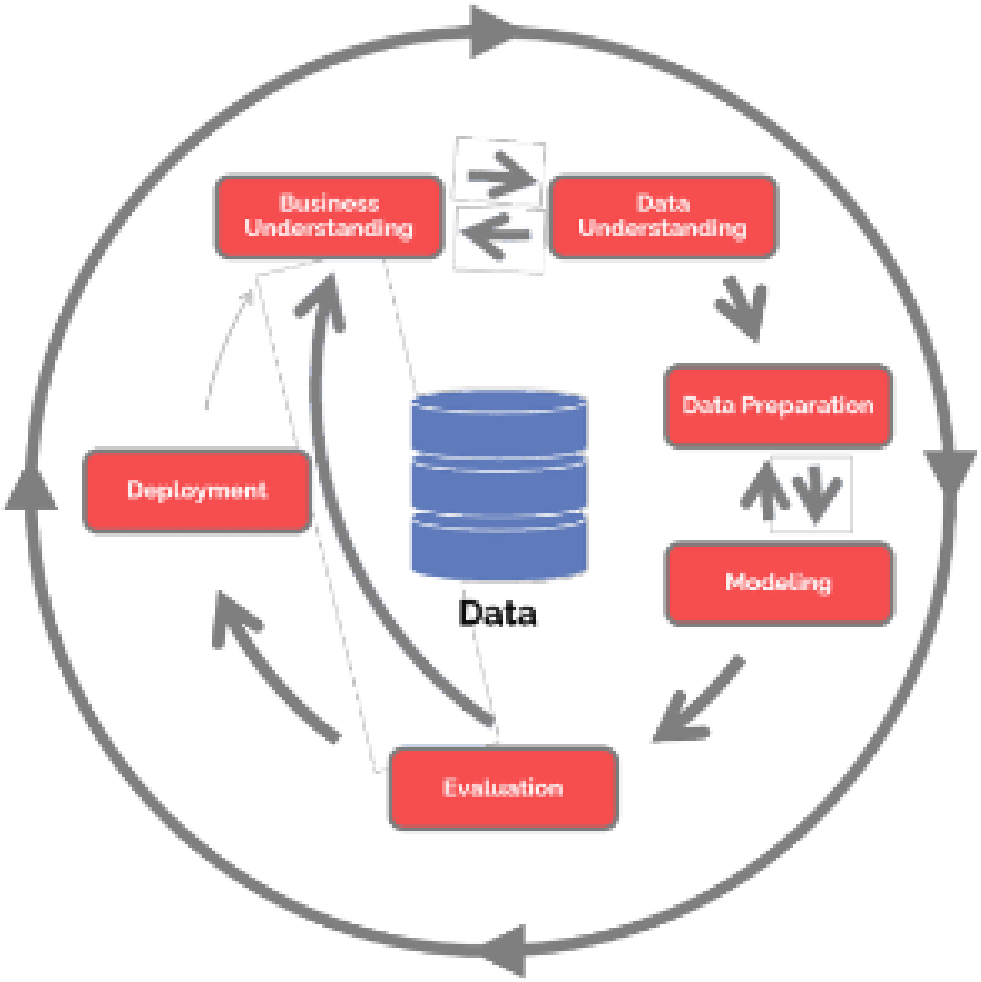
Pada Tahap *Modeling* mahasiswa magang melakukan implementasi algoritma *machine learning* untuk membangun model klasifikasi. Misalnya, dalam *credit assesment,* digunakan *K-Nearest Neighbours* dan *decision trees* dalam pemodelan. Tahap ini juga mencakup penyetelan parameter model dan pengujian untuk menemukan konfigurasi terbaik yang memberikan hasil yang optimal [9].

Selanjutnya adalah *Evaluation,* tahap di mana model yang telah dibangun dievaluasi untuk mengukur kinerjanya. Mahasiswa magang menerapkan evaluasi dengan menggunakan berbagai metrik seperti akurasi, *precision, recall*, F1 Score, dan *confusion matrix* untuk memastikan bahwa model memenuhi kebutuhan bisnis dan dapat digunakan untuk mengambil keputusan yang tepat [10].

Tahap terakhir yang dilakukan oleh mahasiswa magang adalah Deployment, yaitu proses mengimplementasikan model yang telah melalui tahap evaluasi ke dalam lingkungan produksi [10]. Pada tahap ini, mahasiswa magang bertugas mengintegrasikan model *credit assessment* ke dalam *API* menggunakan *Django*

Rest Framework. Setelah model berhasil diintegrasikan ke dalam *API*, langkah selanjutnya adalah menghubungkan *API* tersebut dengan *website* yang dikembangkan menggunakan Streamlit.

Secara keseluruhan, mahasiswa magang menerapkan seluruh proses CRISP-DM pada perancangan klasifikasi website *credit assesment*, mulai dari *business understanding*, *data understanding*, *data preparation*, *modeling*, *evaluation*, hingga *deployment*, untuk memastikan pengembangan model dilakukan secara sistematis dan relevan dengan tujuan bisnis.



Gambar 3. 1 CRISP-DM

Sumber : Analisis Sentimen Kepuasan Publik Terhadap Masa Kepemimpinan Shin Tae Yong Menggunakan Algoritma Naïve Bayes, 2025 [10].

Dalam proyek ini, tahapan *Data Preparation* dan *Data Exploration* sangat krusial untuk memastikan keberhasilan tahapan berikutnya. *Data Preprocessing* mencakup pembersihan data, seperti menangani nilai yang hilang, mengatasi inkonsistensi, dan mengonversi data ke format yang sesuai. Normalisasi juga dilakukan untuk memastikan skala antar atribut seragam, mencegah bias pada algoritma. *Data Exploration* bertujuan untuk memahami pola dan hubungan antar variabel dalam dataset nasabah. Analisis ini meliputi visualisasi distribusi skor kredit, korelasi antar variabel, dan identifikasi kelompok nasabah dengan karakteristik tertentu. Hasil eksplorasi ini membantu dalam memilih variabel relevan dan memberikan wawasan untuk meningkatkan pemahaman bisnis secara keseluruhan. Berikut merupakan tahapan dari Data Preprocessing, dan Data Exploration:

1. Melakukan *Feature selection*, dan *remove duplicate column*

Tahapan awal proses ini dimulai dengan melakukan import data dataset nasabah. Langkah ini bertujuan untuk memuat data nasabah ke dalam sistem, yang nantinya akan digunakan sebagai basis pengembangan model *machine learning*. Setelah data berhasil diimpor, langkah berikutnya adalah *feature selection*, yaitu proses pemilihan subset dari fitur (variabel atau kolom) yang paling relevan dan signifikan untuk mempengaruhi hasil klasifikasi. Proses ini penting dilakukan karena tidak semua fitur dalam dataset memiliki kontribusi yang besar terhadap performa model. Dengan memilih hanya fitur yang relevan, proses pelatihan model menjadi lebih efisien, mengurangi risiko *overfitting*, dan meningkatkan interpretabilitas model.

Tahap selanjutnya adalah *eliminate duplicate data*, yaitu menghapus data yang berulang (*redundant*) di dalam dataset. Proses ini bertujuan untuk memastikan bahwa dataset hanya berisi entri unik, sehingga analisis dan pengembangan model dilakukan berdasarkan informasi yang valid dan tidak terduplikasi. Data duplikat dapat menyebabkan bias dalam analisis serta

memengaruhi kualitas klasifikasi model, sehingga eliminasi langkah ini menjadi salah satu bagian penting dalam *preprocessing data*.

Hasil dari proses *feature selection* dan *eliminate duplicate data* ditampilkan pada Gambar 3.2 dan Gambar 3.3, yang menunjukkan *dataset* yang telah diperbaiki dan dipersiapkan untuk tahap pengembangan model selanjutnya. Melalui langkah-langkah ini, data yang digunakan menjadi lebih bersih, relevan, dan siap untuk diolah lebih lanjut, memberikan fondasi yang kuat untuk membangun model klasifikasi yang akurat.

```
df_character_personal = df[['Penghasilan_setelah_dipotong_pajak','total_asset','total_hutang','huru_hara',
                            'kriminalitas','narkoba','terkena_kriminalitas_sebagai_korban',
                            'Tujuan_Peminjaman','status_agunan_terburuk','situasi_tempat_tinggal.1',
                             'Jumlah_Hari_Tunggakan','potensi_daerah']]
```

Gambar 3. 2 Proses Feature Selection & Eliminate Duplicate Data

```
print(f"Jumlah baris sebelum penghapusan duplikat: {len(df)}")

# Menghapus data duplikat
df_character_personal = df_character_personal.drop_duplicates()

# Menampilkan jumlah baris setelah penghapusan data duplikat
print(f"Jumlah baris setelah penghapusan duplikat: {len(df_character_personal)}")
```

```
[79] ✓ 0.5s

Jumlah baris sebelum penghapusan duplikat: 499997
Jumlah baris setelah penghapusan duplikat: 499997
```

Gambar 3. 3 Output Duplicate Data

2. Melakukan Pengecekan *Missing value, unique Value* pada dataset nasabah

Setelah dilakukan *feature selection* dan eliminasi data duplikat, tahapan selanjutnya adalah mengecek *unique value* pada *dataset*, di mana proses pengecekan tersebut ditunjukkan pada Gambar 3.4. Langkah ini penting untuk memahami karakteristik dan distribusi data, serta mengidentifikasi kategori-kategori dalam kolom-kolom kategorikal yang dapat memengaruhi analisis atau model yang akan dibangun. Proses ini juga berfungsi untuk mendeteksi anomali atau kesalahan dalam data, seperti entri duplikat, nilai yang tidak valid, atau nilai

yang tidak sesuai dengan domain tertentu, sehingga dapat memastikan kualitas dan integritas dataset sebelum tahap analisis lebih lanjut.

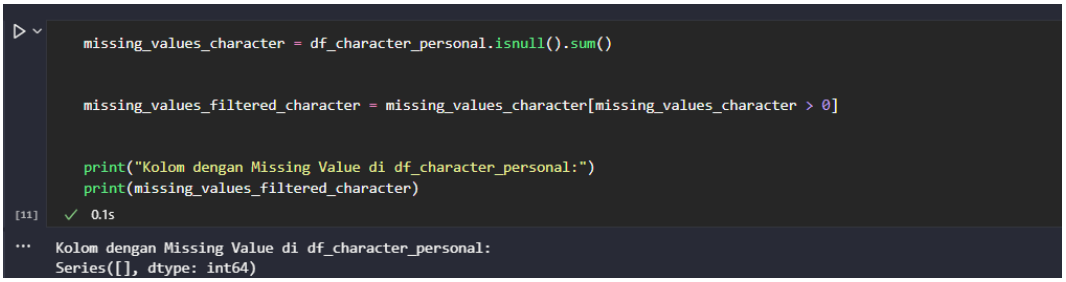
Dengan mengetahui nilai unik, analisis dapat lebih efektif mempersiapkan data untuk analisis lebih lanjut, termasuk visualisasi dan pemodelan, yang pada gilirannya mendukung pengambilan keputusan yang lebih tepat dalam konteks bisnis. Dapat disimpulkan dari proses tersebut seluruh *value feature categorical tidak* ada entri duplikat atau nilai yang tidak valid, sehingga tidak diperlukan penanganan khusus untuk langkah ini, sehingga data dapat langsung digunakan dalam tahap analisis lebih lanjut, termasuk visualisasi dan pemodelan.

```
column Penghasilan_setelah_dipotong_pajak:
 Index([       0.0,  3924013.95,  3064807.35,   5900493.7,   5044919.9,
       22818265.65,  2557229.95,  3822377.25,   9347063.3,   4188485.4,
       ...
        6131566.95,  3537012.45,  3381695.05,  7364860.75,  46948920.2,
         3377489.4,  4004737.35,   8161588.7,  20368955.7,  8311060.75],
      dtype='float64', name='Penghasilan_setelah_dipotong_pajak', length=450865)

column huru_hara:
 Index(['Rendah', 'Tinggi', 'Sedang'], dtype='object', name='huru_hara')

column kriminalitas:
 Index(['Rendah', 'Sedang', 'Tinggi'], dtype='object', name='kriminalitas')

column narkoba:
 Index(['Tidak', 'Iya'], dtype='object', name='narkoba')

column total_hutang:
 Index([ 5753063.0, 28444219.0, 12837800.0, 11646741.0,  6085790.0, 21048126.0,
         4297616.0, 20112334.0, 20277292.0, 22632350.0,
       ...
       22275338.0,  8495492.0, 17525583.0, 18999214.0, 28961727.0,  6524783.0,
        7471226.0, 29769339.0, 29948901.0, 21750985.0],
      dtype='float64', name='total_hutang', length=495895)

column Tujuan_Peminjaman:
 Index(['Modal Kerja', 'Ekspansi Bisnis', 'Inovasi dan R&D',
       'Pengembangan Teknologi', 'Pembelian Aset Tetap', 'Cadangan Likuiditas',
       'Penyelamatan Bisnis', 'Konsolidasi Utang', 'Akuisisi atau Merger',
       'Modal Ventura'],
```

Gambar 3. 4 Proses Pengecekan Unique Value

tahapan selanjutnya akan dilakukan *handling missing data*, proses ini bertujuan untuk menangani data yang hilang atau tidak lengkap dalam *dataset* agar analisis dan *model machine learning* menjadi lebih akurat dan andal. Tercantum

hasil dari proses pengecekan *missing value* pada Gambar 3.5 dapat disimpulkan bahwa seluruh *feature tidak* terdapat baris atau entri yang kosong, sehingga tidak diperlukan penanganan khusus untuk proses tersebut.



Gambar 3. 5 Proses Pengecekan Missing Value

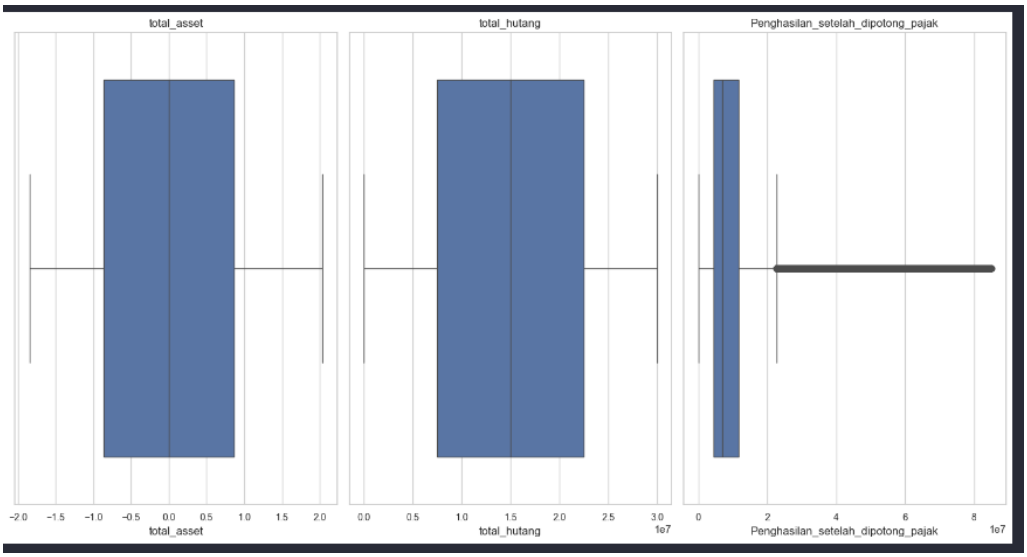
3. Implementasi *MinMax Scaler & Encoding*

Tahap berikutnya adalah melakukan *MinMax Scaler & Encoding*, tujuan dilakukan *Minmax Scaler* adalah untuk menormalkan fitur numerik ke dalam rentang tertentu, umumnya antara 0 dan 1. Normalisasi ini penting untuk mengurangi bias yang mungkin muncul akibat perbedaan skala antar fitur, sehingga setiap fitur dapat memberikan kontribusi yang setara dalam proses pembelajaran model. Tujuan *encoding* dalam konteks *machine learning* adalah untuk mengubah variabel kategorikal menjadi format numerik yang dapat diproses oleh algoritma pembelajaran mesin. proses implementasi *MinMax Scaler & Encoding* ditunjukkan pada Gambar 3.6

Gambar 3. 6 Implementasi MinMax Scaler & Encoding

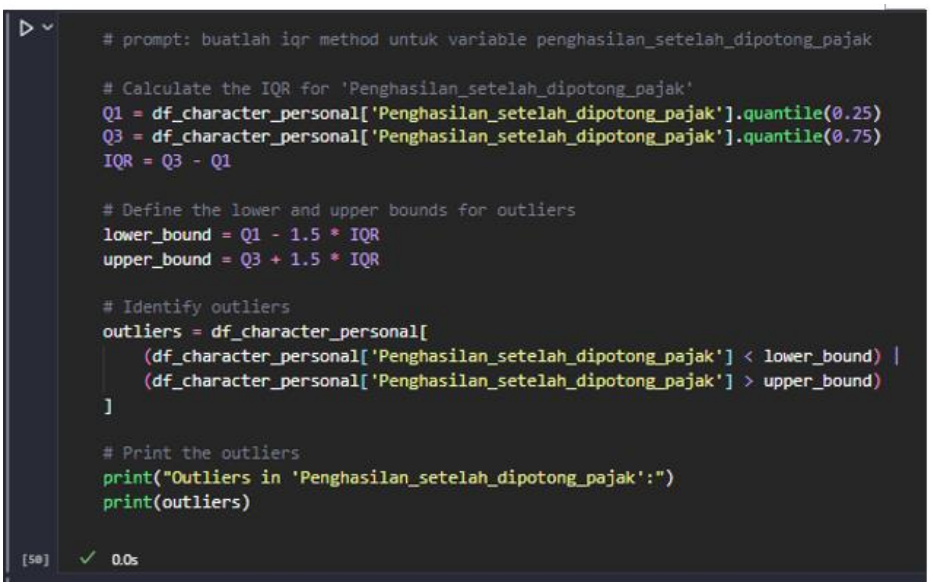
4. Implementasi IQR Method

Tahap berikutnya adalah mengeliminasi *outliers*. *Outliers* adalah data yang memiliki nilai ekstrem, berbeda signifikan dari sebagian besar data dalam suatu dataset. Nilai-nilai ini bisa muncul akibat kesalahan pengukuran, entri data yang tidak akurat, atau karakteristik data yang unik namun jarang terjadi. Salah satu metode yang umum digunakan untuk mendeteksi dan menangani *outliers* adalah *Interquartile Range (IQR).* IQR mengukur rentang tengah dari data, sehingga memudahkan identifikasi titik data yang berada jauh di luar pola distribusi utama, membantu menjaga kualitas dan keandalan analisis data secara keseluruhan. Untuk mendeteksi *features* yang memiliki *outliers* dalam suatu dataset, akan dilakukan visualisasi data melalui boxplot pada seluruh *Features*, yang memungkinkan pengamatan pada seluruh *features* bertipe data kontinu. Dengan visualisasi ini, distribusi data dari setiap *features* bertipe data kontinu dapat diamati, di mana titik-titik pada *boxplot* menunjukkan keberadaan nilai *outliers* yang signifikan, yang berada di luar jangkauan distribusi utama. Penggunaan *boxplot* sangat membantu dalam identifikasi awal *outliers*, karena secara visual menampilkan batas atas dan bawah data. Sebagai tambahan, proses pengecekan *outliers* menggunakan boxplot ditunjukkan pada Gambar 3.7



Gambar 3.7 Proses Pengecekan Outliers Menggunakan Boxplot

Dari visualisasi Gambar 3.7 menunjukkan adanya individu atau entitas dengan penghasilan yang jauh lebih tinggi dibandingkan dengan kebanyakan data, yang bisa disebabkan oleh berbagai faktor, termasuk kesalahan pengukuran atau kondisi khusus yang tidak biasa. yaitu Penghasilan_setelah_dipotong_pajak, Maka dari itu akan dilakukan proses mengeliminasi *outliers* menggunakan IQR method ditunjukkan pada Gambar 3.8




```
# prompt: buatlah iqr method untuk variable penghasilan_setelah_dipotong_pajak

# Calculate the IQR for 'Penghasilan_setelah_dipotong_pajak'
Q1 = df_character_personal['Penghasilan_setelah_dipotong_pajak'].quantile(0.25)
Q3 = df_character_personal['Penghasilan_setelah_dipotong_pajak'].quantile(0.75)
IQR = Q3 - Q1

# Define the lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify outliers
outliers = df_character_personal[
    (df_character_personal['Penghasilan_setelah_dipotong_pajak'] < lower_bound) |
    (df_character_personal['Penghasilan_setelah_dipotong_pajak'] > upper_bound)
]

# Print the outliers
print("Outliers in 'Penghasilan_setelah_dipotong_pajak':")
print(outliers)
```

Gambar 3. 8 Proses Pengecekan Outliers Menggunakan Boxplot

Dalam kode ini, metode *Interquartile Range* (IQR) digunakan untuk mendeteksi outlier pada variabel Penghasilan_setelah_dipotong_pajak dari dataset df_character_personal. Tahapan Pertama, kuartil pertama (Q1) dan kuartil ketiga (Q3) dihitung untuk menentukan rentang IQR, yang merupakan selisih antara Q3 dan Q1. Kemudian, batas bawah dan batas atas untuk *outlier* ditentukan dengan mengurangkan 1.5 kali IQR dari Q1 dan menambahkan 1.5 kali IQR ke Q3, masing masing. Setelah itu, entri dalam dataset yang nilai Penghasilan setelah dipotong pajak berada di luar batas-batas tersebut diidentifikasi sebagai *outlier*. Terakhir, kode mencetak entri yang terdeteksi sebagai *outlier* untuk memberikan informasi lebih lanjut tentang data yang mungkin perlu dianalisis lebih dalam atau ditangani dalam proses pembersihan data.

3.2.3 Melakukan *Cluster*ing Dataset Nasabah

Setelah melakukan *preprocessing data* dan memastikan tidak ada *missing valu*e, langkah berikutnya adalah *cluster*ing, yaitu teknik analisis data untuk mengelompokkan data berdasarkan kemiripan atau karakteristik tertentu. *Cluster*ing membantu menemukan pola atau segmentasi yang berguna dalam pengambilan keputusan atau analisis lebih lanjut. Dengan menggunakan *cluster*ing, kita dapat mengidentifikasi kelompok nasabah berdasarkan tingkat risiko kredit dari risiko tinggi, rendah, hingga aman dalam data yang dapat diinterpretasikan. Pendekatan ini sangat berguna untuk keperluan seperti segmentasi risiko nasabah atau pengelompokan berdasarkan karakteristik perilaku, yang dapat mendukung penilaian *credit assesment* berbasis karakter. Tahapan yang dilakukan *Clustering* Dataset Nasabah adalah sebagai berikut:

1. Melakukan *Cluster*ing menggunakan Algoritma *K-Means*

Pada Gambar 3.9, dilakukan implementasi algoritma *K-Means* dari pustaka sklearn untuk melakukan proses *cluster*ing pada dataset yang tersimpan dalam variabel df_character_personal. Dengan menetapkan parameter n_*cluster*s=3, yang bertujuan untuk membagi data menjadi tiga kelompok. Parameter n_init='auto' mengindikasikan bahwa jumlah inisialisasi *centroid* akan ditentukan secara otomatis oleh algoritma, sedangkan random_state=0 memastikan hasil *cluster*ing akan tetap konsisten di setiap eksekusi karena posisi awal centroid akan selalu diinisialisasi dengan cara yang sama. *Setelah* model K-Means dilatih menggunakan metode fit_predict() , hasil *cluster*ing disimpan dalam variabel labels, yang berisi label kelompok yang ditugaskan kepada setiap entitas dataset

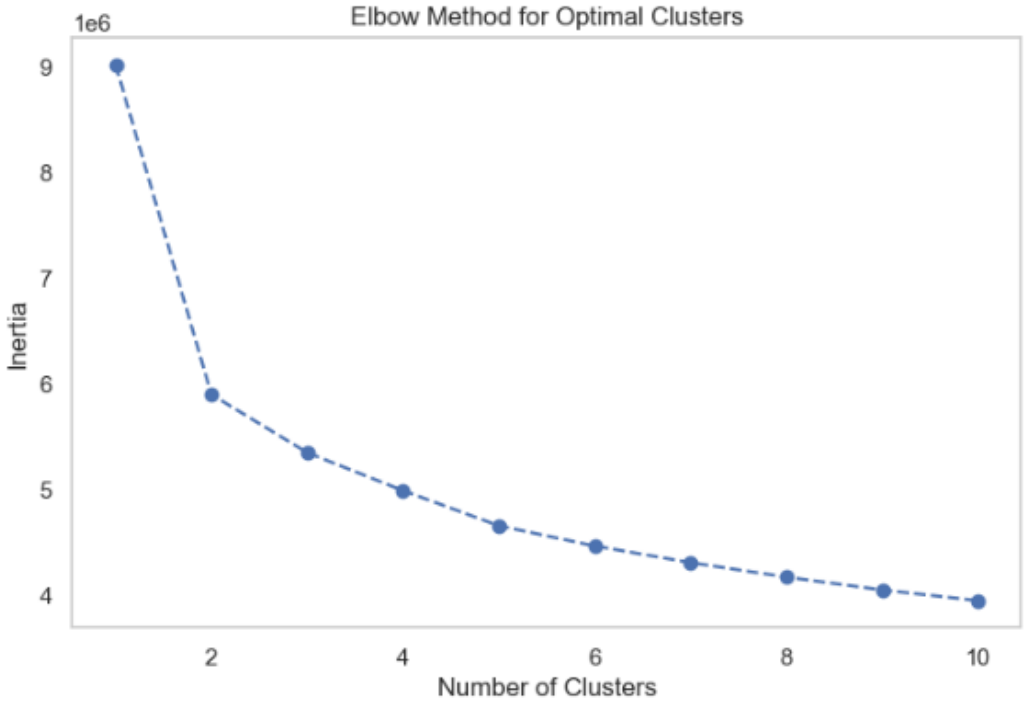
```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, n_init='auto', random_state=0)
labels = kmeans.fit_predict(df_character_personal)
```

Gambar 3. 9 Proses Pengecekan Outliers Menggunakan Boxplot

2. Evaluasi model *cluster* menggunakan *Elbow Method*

Pada Gambar 3.10, diterapkan *Elbow Method* yang bertujuan untuk menentukan jumlah *cluster* optimal dalam analisis *clustering*, khususnya dengan menggunakan algoritma *K-Means*. Metode ini dilakukan dengan menghitung nilai

inertia, yaitu total jarak dari setiap titik data ke centroid terdekatnya, untuk berbagai jumlah *cluster* yang berbeda. Inertia umumnya menurun seiring dengan bertambahnya jumlah *cluster*, karena setiap data semakin dekat dengan centroid kelompoknya. Namun, setelah jumlah *cluster* tertentu, penurunan inertia menjadi tidak signifikan, yang menghasilkan pola "siku" atau "*elbow*" pada grafik. Titik siku ini dianggap sebagai jumlah *cluster* optimal, karena menandakan titik keseimbangan antara kompleksitas model dengan kemampuan model dalam menjelaskan variabilitas data.



Gambar 3. 10 Visualisasi Elbow Method

Dalam kasus ini, peneliti memilih empat *cluster* dikarnakan penurunan inertia sudah tidak signifikan dan membentuk pola *elbow*. Selain itu, kebutuhan model adalah untuk melakukan klasifikasi tingkat risiko yang terklasifikasi dalam tiga kategori, yaitu "risiko_rendah," "risiko_sedang," dan "risiko_tinggi," sehingga pemilihan tiga *cluster* cukup memadai untuk memenuhi tujuan klasifikasi risiko ini. Selain alasan penurunan inertia yang digunakan untuk membentuk pola *elbow*, hasil perhitungan modus untuk setiap fitur dengan menggunakan 3 *cluster* tidak menunjukkan pola yang konsisten atau jelas, sehingga tidak memungkinkan untuk

melakukan pendefinisian kategori risiko berdasarkan pola tersebut, karena karakteristik masing-masing *cluster* tidak dapat diidentifikasi secara tegas.

3. Evaluasi model *cluster* menggunakan *Shillouete Score*

Gambar 3.11, dan Gambar 3.12, dilakukan perhitungan modus (nilai yang paling sering muncul) untuk beberapa fitur dalam dataset yang telah dikelompokkan menggunakan algoritma *K-Means*. Tujuan dari analisis ini adalah untuk mengidentifikasi karakteristik yang dominan pada masing-masing *cluster*. Dengan mengetahui pola yang ada dalam setiap *cluster*, kita dapat mempermudah dalam mendefinisikan kategori risiko seperti "risiko_rendah," "risiko_sedang," dan "risiko_tinggi". Pendekatan ini memberikan wawasan yang lebih jelas mengenai distribusi data di setiap *cluster* dan mendukung pengambilan keputusan yang lebih tepat dalam mengklasifikasikan tingkat risiko.

```
Modus untuk Cluster 0:
kriminalitas: 1
narkoba: 0
terkena_kriminalitas_sebagai_korban: 0
Tujuan_Peminjaman: 5
huru_hara: 2
situasi_tempat_tinggal_1: 1
Jumlah_Hari_Tunggakan: 2
potensi_daerah: 3

Modus untuk Cluster 1:
kriminalitas: 0
narkoba: 0
terkena_kriminalitas_sebagai_korban: 1
Tujuan_Peminjaman: 2
huru_hara: 0
situasi_tempat_tinggal_1: 3
Jumlah_Hari_Tunggakan: 3
potensi_daerah: 1

Modus untuk Cluster 2:
kriminalitas: 0
narkoba: 0
terkena_kriminalitas_sebagai_korban: 0
Tujuan_Peminjaman: 7
huru_hara: 0
situasi_tempat_tinggal_1: 3
Jumlah_Hari_Tunggakan: 0
potensi_daerah: 1
```

Gambar 3. 11 Output Perhitungan Modus Cluster

```
Modus untuk Cluster 3:
kriminalitas: 2
narkoba: 1
terkena_kriminalitas_sebagai_korban: 0
Tujuan_Peminjaman: 3
huru_hara: 0
situasi_tempat_tinggal_1: 1
Jumlah_Hari_Tunggakan: 0
potensi_daerah: 0
```

Gambar 3. 12 Output Perhitungan Modus Cluster 3

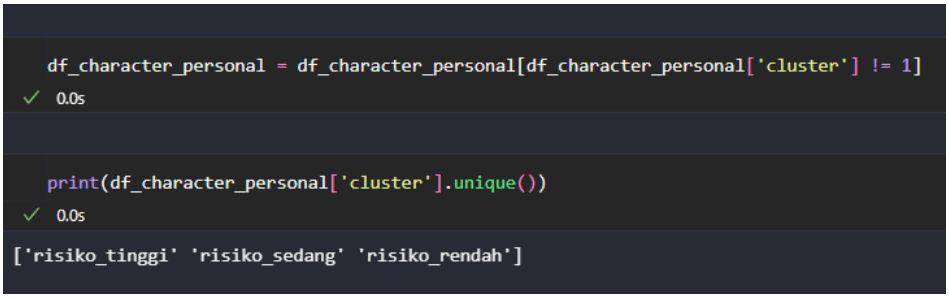
Gambar 3.13, dilakukan pendefinisian label *Clustering* bedasarkan karakter dari tiap *cluster*, untuk *cluster* 2 didefinisikan risiko_rendah dikarnakan lingkungan tempat tinggal nasabah memiliki tingkat kriminalitas yang rendah, dan tidak pernah mengkonsumsi narkoba, dan situasi dari lingkunganya memiliki tingkat huru_hara yang rendah(kondusif), dan cenderung tidak melakukan tunggakan. Sedangkan *cluster* 0 di definisikan menjadi risiko_sedang dikarnakan lingkungan tempat tinggal nasabah memiliki tingkat kriminalitas yang sedang, dan tidak pernah mengkonsumsi narkoba, dan situasi dari lingkunganya memiliki tingkat huru_hara yang tinggi(sering terjadi keributan), dan cenderung melakukan tunggakan 30-60 hari. *Cluster* 3 di definisikan menjadi risiko_tinggi dikarnakan lingkungan tempat tinggal nasabah memiliki tingkat kriminalitas yang tinggi, dan pernah mengkonsumsi narkoba, dan situasi dari lingkunganya memiliki tingkat huru_hara yang tinggi(sering terjadi keributan).

```python
df_character_personal['cluster'] = df_character_personal['cluster'].replace({
    2: 'risiko_rendah',
    3: 'risiko_tinggi',
    0: 'risiko_sedang'
})
```

Gambar 3. 13 Pendefinisian Label Clustering

Gambar 3.14 menggambarkan proses penghapusan *cluster* yang dianggap tidak relevan. *Cluster* tersebut diidentifikasi sebagai entitas yang tidak menunjukkan pola signifikan terkait tingkat risiko kredit, sehingga tidak memberikan kontribusi yang berarti terhadap proses *Credit Assestment* berdasarkan karakteristik pelanggan. Dengan mengecualikan *cluster* yang kurang informatif,

model klasifikasi dapat diarahkan untuk lebih fokus pada data yang relevan, meningkatkan efisiensi analisis dan akurasi pengambilan keputusan berbasis data.



Gambar 3. 14 Pendefinisian Label Clustering

3.2.4 Melakukan perancangan klasifikasi *credit assesment*

Setelah melakukan *cluster*ing, maka dari itu data telah siap untuk dilakukan klasifikasi, Tahapan yang dilakukan *preprocessing* dan eksplorasi data adalah sebagai berikut:

1. Screening Model Dengan Implementasi *Library Pycaret*

tahapan utama pada tahap ini adalah melakukan proses *train test splitting* pada Gambar 3.15. *train test splitting* adalah proses pembagian dataset menjadi 2 subset yaitu *train* dan *test* dengan perbandingan 8:2, dan menggunakan parameter random_state hal tersebut bertujuan untuk membangun hasil pembagian data tetap konsisten untuk setiap melakukan eksekusi program, sedangkan Tujuanya dari proses *train test split* adalah untuk melatih model pada 80% total keseluruhan data yang ditempati pada *training* data dan menguji kinerjanya pada data yang belum pernah dilihat sebelumnya pada *test* data.

Setelah itu akan dilakukan. Setelah itu dilakukan eksekusi fungsi setup dari *PyCaret* untuk mempersiapkan *pipeline* klasifikasi. Data latih (X_train) dan *label target* (y_train) digabungkan menjadi satu *DataFrame* untuk mematuhi format input yang diperlukan, dengan kolom *cluster* ditetapkan sebagai target klasifikasi. *Parameter* session_id=42 digunakan untuk memastikan semua proses pengacakan dalam *pipeline*, seperti pembagian data atau pemilihan model, menghasilkan hasil yang konsisten. Setelah inisialisasi ini, *PyCaret* secara otomatis melakukan

praproses data dan mempersiapkan model untuk pelatihan serta melakukan pemodelan secara otomatis beberapa algoritma, dan melakukan evaluasi.

```
from sklearn.model_selection import train_test_split
from pycaret.classification import *

X_train, X_test, y_train, y_test = train_test_split(
    df_character_personal.drop('cluster', axis=1),
    df_character_personal['cluster'],
    test_size=0.2,
    random_state=42
)

# Setup
clf = setup(data=pd.concat([X_train, y_train], axis=1), target='cluster', session_id=42)
```

Gambar 3. 15 Train Test Split

Setelah model dilatih secara otomatis menggunakan PyCaret, langkah berikutnya adalah melakukan proses *screening model*. Proses ini bertujuan untuk menyeleksi model terbaik dengan menentukan 15 model unggulan yang dipilih berdasarkan hasil perbandingan performa. Seleksi dilakukan dengan mempertimbangkan metrik evaluasi, seperti akurasi, presisi, *recall*, *F1-score*, atau metrik lainnya, tergantung pada jenis masalah yang ditangani, apakah itu klasifikasi atau regresi.

Untuk memastikan bahwa model diuji secara menyeluruh pada berbagai subset data, validasi dilakukan menggunakan teknik *cross-validation* dengan 5 lipatan (*5-fold cross-validation*). Teknik ini membagi data menjadi lima subset yang berbeda, di mana setiap subset secara bergiliran berfungsi sebagai data uji, sementara subset lainnya digunakan untuk melatih model. Proses ini membantu mengurangi risiko overfitting dan memberikan evaluasi yang lebih akurat terhadap performa model pada data yang belum pernah dilihat sebelumnya.

Proses seleksi model dieksekusi dengan memanfaatkan fungsi *compare_models* yang ditampilkan pada Gambar 3.14. Fungsi ini secara otomatis membandingkan performa berbagai algoritma yang tersedia di PyCaret dan mengurutkannya berdasarkan metrik tertentu. Hasil dari fungsi ini memberikan wawasan yang berharga untuk memilih model yang paling sesuai dengan karakteristik dataset dan

tujuan analisis, sehingga memungkinkan langkah selanjutnya, seperti *tuning hyperparameter* atau *deployment*, dapat dilakukan dengan lebih terarah dan efisien.

```
# Bandingkan semua model
best_models = compare_models(n_select=15, fold=5)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lr	Logistic Regression	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.8480
nb	Naive Bayes	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.3680
dt	Decision Tree Classifier	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.3940
ridge	Ridge Classifier	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.3160
rf	Random Forest Classifier	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	2.9520
ada	Ada Boost Classifier	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	2.4580
gbc	Gradient Boosting Classifier	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	16.4720
lda	Linear Discriminant Analysis	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.3520
et	Extra Trees Classifier	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	2.2180
xgboost	Extreme Gradient Boosting	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.4180
lightgbm	Light Gradient Boosting Machine	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.6440
knn	K Neighbors Classifier	0.9988	1.0000	0.9988	0.9988	0.9988	0.9982	0.9982	8.2360
svm	SVM - Linear Kernel	0.9696	0.0000	0.9696	0.9727	0.9698	0.9543	0.9556	0.6300
qda	Quadratic Discriminant Analysis	0.5885	0.0000	0.5885	0.6406	0.4795	0.3715	0.4978	0.3640
dummy	Dummy Classifier	0.3635	0.5000	0.3635	0.1321	0.1938	0.0000	0.0000	0.3280

Gambar 3. 16 Evaluasi Model Pycaret

Gambar 3.15, menampilkan evaluasi yang telah dilakukan terhadap beberapa model, Peneliti memutuskan untuk memilih *K Neighbors Classifier* sebagai model yang lebih optimal. Meskipun ada beberapa model lain dengan performa serupa, termasuk *Decision Tree Classifier* yang menunjukkan hasil sempurna pada metrik evaluasi seperti akurasi, *AUC, recall, precision, F1 score, kappa, dan MCC* dengan nilai 1.0000, Peneliti menyadari bahwa model *Decision Tree* berpotensi mengalami *overfitting*.

Model *Decision Tree* cenderung sangat sensitif terhadap kompleksitas data dan rentan terhadap *overfitting*, terutama jika pohon keputusan terlalu dalam atau terlalu rumit. Meskipun waktu eksekusinya sangat cepat, yakni hanya 0.394 detik, hasil yang sangat tinggi pada data pelatihan sering kali menjadi indikasi bahwa model ini menghafal pola-pola tertentu dari data pelatihan tanpa mampu menggeneralisasi dengan baik pada data yang belum terlihat sebelumnya.

Di sisi lain, *K Neighbors Classifier* meskipun sedikit lebih rendah dalam hal akurasi dibandingkan dengan *Decision Tree* (0.9988 vs. 1.0000), menunjukkan performa yang sangat baik pada semua metrik lainnya, dengan nilai *AUC, recall, precision, F1 score, kappa*, dan *MCC* yang sangat mendekati 1.0000.

Ini menunjukkan bahwa *K Neighbors Classifier* mampu memberikan klasifikasi yang sangat akurat dan stabil, dengan sedikit risiko *overfitting*, karena metode ini cenderung lebih *robust* terhadap *noise* dan lebih mengandalkan hubungan jarak antara data, tanpa perlu mempelajari hubungan yang sangat kompleks seperti pada pohon keputusan.

Dengan mempertimbangkan hasil evaluasi ini, Peneliti memilih *K Neighbors Classifier* karena kemampuannya untuk memberikan hasil yang konsisten dan akurat tanpa terlalu terikat pada data pelatihan tertentu. Model ini juga lebih mudah untuk dioptimalkan dan lebih baik dalam melakukan generalisasi pada data yang tidak terlihat sebelumnya.

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       monotonic_cst=None, random_state=42, splitter='best')
```

Gambar 3. 17 Parameter Decision Tree Classifier

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=-1, n_neighbors=5, p=2,
                     weights='uniform')
```

Gambar 3. 18 Parameter Decision Tree Classifier

Selanjutnya, setelah menentukan model terbaik, langkah berikutnya adalah memeriksa parameter yang digunakan dalam pemodelan di *PyCaret,* parameter yang digunakan pycaret untuk *Decision Tree Classifier* dapat dilihat pada gambar 3.17, dan *K Nearest Nightbour Classifier* dapat dilihat pada gambar 3.16. Parameter ini akan digunakan untuk pembuatan ulang model menggunakan *library scikit-learn* (sklearn), dengan tujuan untuk melatih model pada dataset yang lebih lengkap dan memastikan performa yang lebih stabil serta dapat diandalkan. Proses ini memungkinkan evaluasi model secara lebih mendalam, seperti mengukur akurasi, presisi, *recall, F1-score*, dan metrik relevan lainnya, serta melakukan evaluasi lebih lanjut menggunakan *confusion matrix* untuk memperoleh wawasan yang lebih detail mengenai kinerja model dalam klasifikasi kelas.

2. Membangun model klasifikasi dengan *K-Nearest Neighbour*

Setelah melakukan pemodelan klasifikasi *K-Nearest Neighbour* (KNN) dengan parameter yang sama seperti yang digunakan oleh PyCaret namun dilakukan secara manual menggunakan *library* scikit-learn pada Gambar 3.19, ditemukan adanya perbedaan hasil yang lebih baik sebesar 0.0018 dengan akurasi test data sebesar 0.997 , 0.997 untuk *precision*, 0.997 recall, 0.997 *f1-score*, dan akurasi training sebesar 0.999 dengan menggunakan *library* scikit-learn.

Meskipun terdapat perbedaan tersebut, nilai ini tergolong sangat kecil atau minor, sehingga kemungkinan besar tidak akan memberikan dampak signifikan pada interpretasi model atau kinerjanya dalam aplikasi nyata. Perbedaan semacam ini sering kali disebabkan oleh faktor teknis, seperti cara implementasi algoritma, presisi perhitungan, atau pengaturan awal yang digunakan oleh kedua pendekatan.

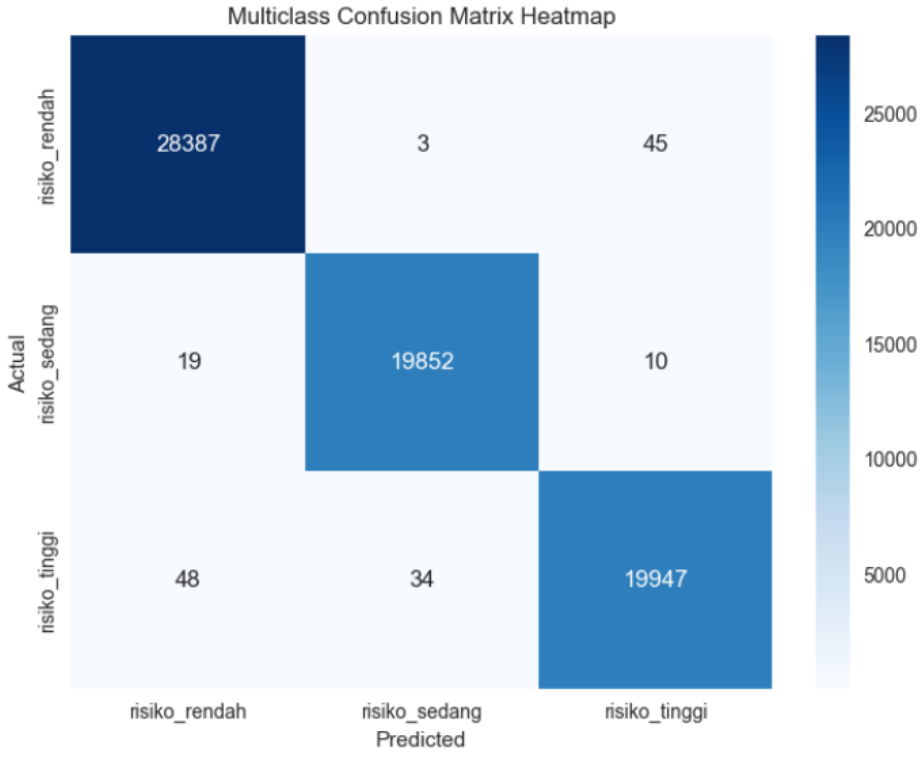
```
Test Accuracy: 0.997673567927427
Overall Precision: 0.9976733539966269
Precision per kelas: [0.99764532 0.99813968 0.99725027]
Overall Recall: 0.997673567927427
Recall per kelas: [0.99831194 0.99854132 0.99590594]
Overall F1-score: 0.9976732700349674
F1-score per kelas: [0.99797852 0.99834046 0.99657765]
==============================
Training Accuracy: 0.9991184399679566
Overall Precision (Train): 0.9991184712897021
Precision per kelas (Train): [0.99893167 0.99941179 0.99908995]
Overall Recall (Train): 0.9991184399679566
Recall per kelas (Train): [0.9993552  0.99937426 0.99852978]
Overall F1-score (Train): 0.9991184138807925
F1-score per kelas (Train): [0.99914339 0.99939303 0.99880978]
```

Gambar 3. 19 Hasil Pembuatan KNN Manual

Hasil pada Gambar 3.20 memberikan gambaran yang jelas tentang performa model dalam klasifikasi kategori risiko. Dari total 28.879 data berlabel risiko_rendah, hanya 48 data yang salah diklasifikasi, dengan rincian 45 data diklasifikasi sebagai risiko_tinggi dan 3 data sebagai risiko_sedang. Kesalahan pada kategori ini dapat dikatakan sangat kecil, menunjukkan kemampuan model yang sangat baik dalam mengenali data dengan risiko rendah.

Untuk data berlabel risiko_sedang yang berjumlah 19881, terdapat 29 data yang salah diklasifikasi, 10 data diklasifikasi sebagai risiko_tinggi dan 19 data sebagai risiko_rendah. Hal ini mencerminkan bahwa model cukup handal dalam membedakan data risiko_sedang dari kategori lainnya.

Sementara itu, dari 20029 data dengan label risiko_tinggi, terdapat 48 data yang salah diklasifikasi ke kategori risiko_rendah, dan 34 data yang salah di klasifikasi pada risiko_sedang. Jumlah ini relatif lebih kecil dibandingkan keseluruhan data pada kategori tersebut, yang menunjukkan bahwa model juga mampu klasifikasi data dengan risiko tinggi secara akurat, meskipun terdapat sedikit kecenderungan untuk salah mengklasifikasikan data ini ke kategori risiko_rendah.



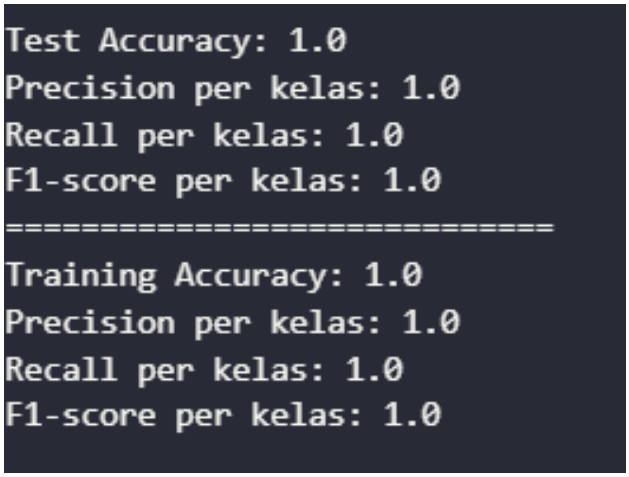
Gambar 3. 20 Confusin Matrix KNN

Dari hasil keseluruhan, dapat disimpulkan bahwa model *K-Nearest Neighbours* memiliki tingkat akurasi yang tinggi, terutama pada kategori risiko_rendah. Meskipun jumlah kesalahan klasifikasi terbesar secara absolut

terjadi pada data dengan label risiko_rendah, persentase kesalahan tersebut sangat kecil dibandingkan dengan total data dalam kategori tersebut, menunjukkan keunggulan model dalam menangani kategori risiko yang dominan. Selain itu, performa model dalam klasifikasi kategori risiko_sedang dan risiko_tinggi juga cukup baik, dengan potensi perbaikan lebih lanjut pada kesalahan yang cenderung mengarah ke kategori risiko_rendah. Secara keseluruhan, hasil ini mencerminkan keandalan model dalam memberikan klasifikasi yang akurat, yang menjadi dasar penting untuk implementasi lebih lanjut dalam proses pengambilan keputusan berbasis data.

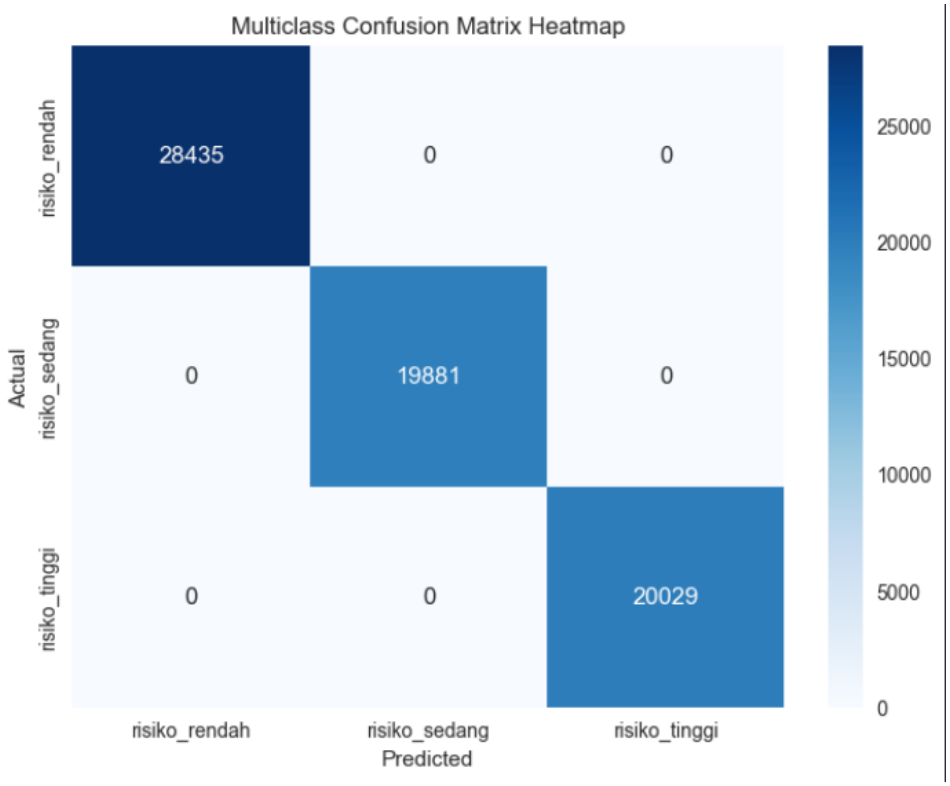
3. Membangun model klasifikasi dengan *DecisionTreeClassifier*

Pemodelan klasifikasi menggunakan *DecisionTreeClassifier* dari scikit-learn dengan parameter yang sama seperti yang digunakan oleh PyCaret pada Gambar 3.21 menghasilkan akurasi sempurna dengan nilai 1.0, serta metrik lainnya seperti *precision, recall*, dan *f1-score* juga mencapai nilai maksimal, dan akurasi training mencapai 1.0., di mana model terlalu sesuai dengan data pelatihan hingga kehilangan kemampuan untuk menggeneralisasi pada data baru. Indikasi overfitting semakin kuat karena semua metrik evaluasi lainnya, seperti *precision, recall, f1-score* juga menunjukkan nilai sempurna, yang jarang terjadi pada model yang benar-benar robust. Setelah melakukan evaluasi lebih lanjut, diputuskan untuk mencoba algoritma alternatif, yaitu *K-Nearest Neighbors (KNN).*



Gambar 3. 21 Hasil Pembuatan DecisionTreeClassifier Manual

Hasil *confusion matrix DecisionTreeClassifier* pada Gambar 3.22 menunjukkan bahwa model tidak memiliki kesalahan klasifikasi sama sekali, di mana semua data berhasil diklasifikasikan dengan benar ke dalam kategori yang sesuai.



Gambar 3. 22 Confusion Matrix DecisionTreeClassifier

3.2.5 Melakukan perancangan model Retrieval-Augmented Generation(RAG) dengan Ollama

Pembuatan model *Retrieval-Augmented Generation*(RAG) dengan Ollama pada PT. Menara Indonesia bertujuan untuk mempermudah karyawan dalam mengakses informasi penting secara cepat dan efisien dari berbagai dokumen perusahaan, seperti SOP, berkas, atau *file* lainnya. Model RAG menggabungkan kemampuan *retrieval* dan *generation* untuk mencari informasi relevan dari dokumen yang disimpan, Setelah itu model akan menyajikannya dalam format respons yang mudah dipahami. Dengan integrasi Ollama sebagai *platform* AI, proses ini menjadi lebih efisien karena model dapat menangani bahasa alami, memungkinkan karyawan cukup mengetikkan pertanyaan untuk mendapatkan jawaban langsung tanpa harus membaca keseluruhan dokumen, sehingga

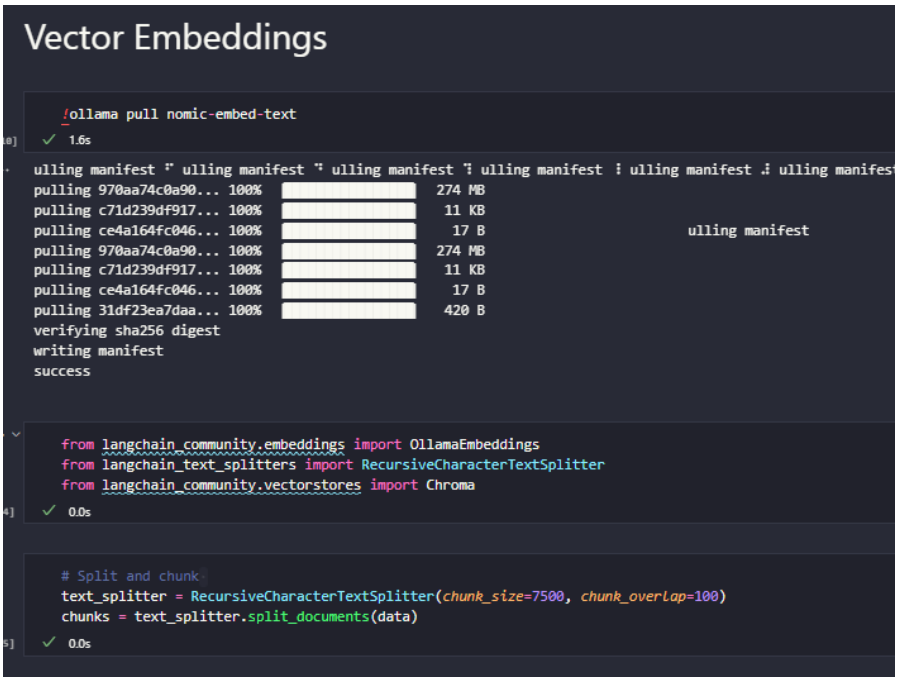
meningkatkan produktivitas kerja. Berikut merupakan Tahapan Dalam Melakukan pembuatan model *Retrical-Augmented Generation*(RAG) dengan Ollama. Berikut merupakan Tahapan yang dilakukan dalam pembuatan model *Retrieval-Augmented Generation*(RAG) dengan Ollama :

1. Membangun *script input pdf* dan *text splitting* untuk pdf

Pada Gambar 3.23, peneliti menggunakan modul UnstructuredPDFLoader dari pustaka langchain_community.document_loaders untuk membaca *file PDF* yang tersedia secara lokal. *File* PDF yang ditentukan melalui variabel *local_path*akan diproses oleh *UnstructuredPDFLoader*, yang berfungsi mengekstrak teks dari dokumen PDF ke dalam format data terstruktur. Proses pemuatan konten dilakukan menggunakan metode loader.load(), yang menyimpan hasil ekstraksi ke dalam variabel data. Setiap elemen dalam data merepresentasikan bagian tertentu dari dokumen, seperti halaman. Untuk menampilkan isi halaman pertama, kode mengakses atribut data[0].page_content, yang berisi teks dari halaman tersebut. Jika variabel *local_path*tidak memiliki isi atau *file* tidak ditemukan, program akan menampilkan pesan "*Upload a PDF file*".

Gambar 3. 23 Script Input PDF IPYNB

Selanjutnya, setelah melakukan *input data* berupa *file PDF*, dilakukan proses pemecahan teks seperti yang terlihat pada Gambar 3.24. Pada tahap pertama, teks yang ada diubah menjadi format yang lebih mudah dipahami oleh komputer, yakni dalam bentuk vektor, menggunakan model *embedding nomic-embed-text*. Model ini bertugas mengonversi teks menjadi representasi numerik sehingga bisa diproses lebih lanjut dalam aplikasi seperti pencarian atau analisis. Setelah itu, teks yang telah diubah menjadi vektor dibagi menjadi beberapa potongan kecil (chunk) dengan ukuran yang telah ditentukan (chunk_size) untuk mempermudah pemrosesan lebih lanjut,Setiap potongan teks ini memiliki panjang sekitar 7500 karakter, dengan sedikit tumpang tindih sebanyak 100 karakter antar potongan, yang bertujuan untuk menjaga kontinuitas atau konteks antar bagian teks. Langkah ini penting agar informasi yang ada pada teks tidak terputus begitu saja.

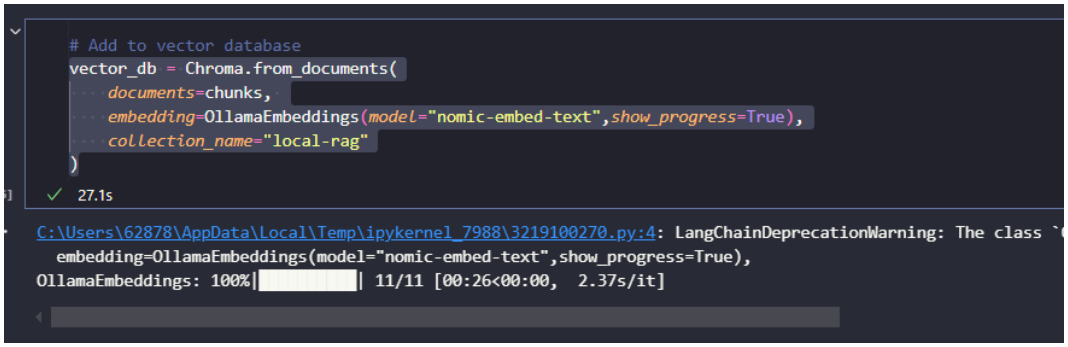


Gambar 3. 24 Text Splitting IPYNB

2. Membangun *Vector database*

Setelah merubah teks pdf menjadi data vektor dan telah dibagi menjadi potongan kecil, maka diperlukan pembuatan *vector database* untuk menyimpan data vektor menggunakan chroma. Pada Gambar 3.25, basis data vektor dibuat

menggunakan Chroma untuk menyimpan representasi vektor dari teks yang telah dibagi menjadi potongan-potongan kecil (*chunks*). Fungsi Chroma.from_documents() digunakan untuk mengonversi teks yang ada dalam dokumen menjadi vektor numerik dengan bantuan model embedding nomic-embed-text dari *OllamaEmbeddings*. Model ini mengubah teks menjadi representasi vektor yang memudahkan analisis dan pencarian berbasis kesamaan isi teks. Proses *embedding* ini juga menyertakan parameter show_progress=True, yang memungkinkan pemantauan perkembangan selama proses berlangsung. Hasil *embedding* tersebut disimpan dalam sebuah koleksi vektor yang dinamakan "local-rag", yang memungkinkan akses cepat dan efisien untuk pencarian atau pemrosesan lebih lanjut berdasarkan kemiripan vektor, mendukung aplikasi yang membutuhkan analisis teks dalam skala besar.



Gambar 3. 25 ChromaDB IPYNB

3. Membangun *Model RAG* dengan Ollama

Setelah data vektor dimasukkan ke dalam database, pada Gambar 3.26, peneliti memanfaatkan pustaka *LangChain* untuk mengintegrasikan model bahasa lokal "Mistral" dengan teknik pemrosesan berbasis *chunk*. Pendekatan ini dirancang untuk memanfaatkan berbagai alat, seperti *ChatPromptTemplate* untuk membangun templat pertanyaan yang terarah dan *StrOutputParser* untuk mengolah hasil keluaran model agar lebih mudah dipahami. Model "Mistral" diakses melalui ChatOllama, sementara komponen tambahan seperti *RunnablePassthrough* dan *MultiQueryRetriever* mendukung pengolahan data dan pengambilan informasi yang relevan dari berbagai sumber. Dengan integrasi ini, sistem mampu

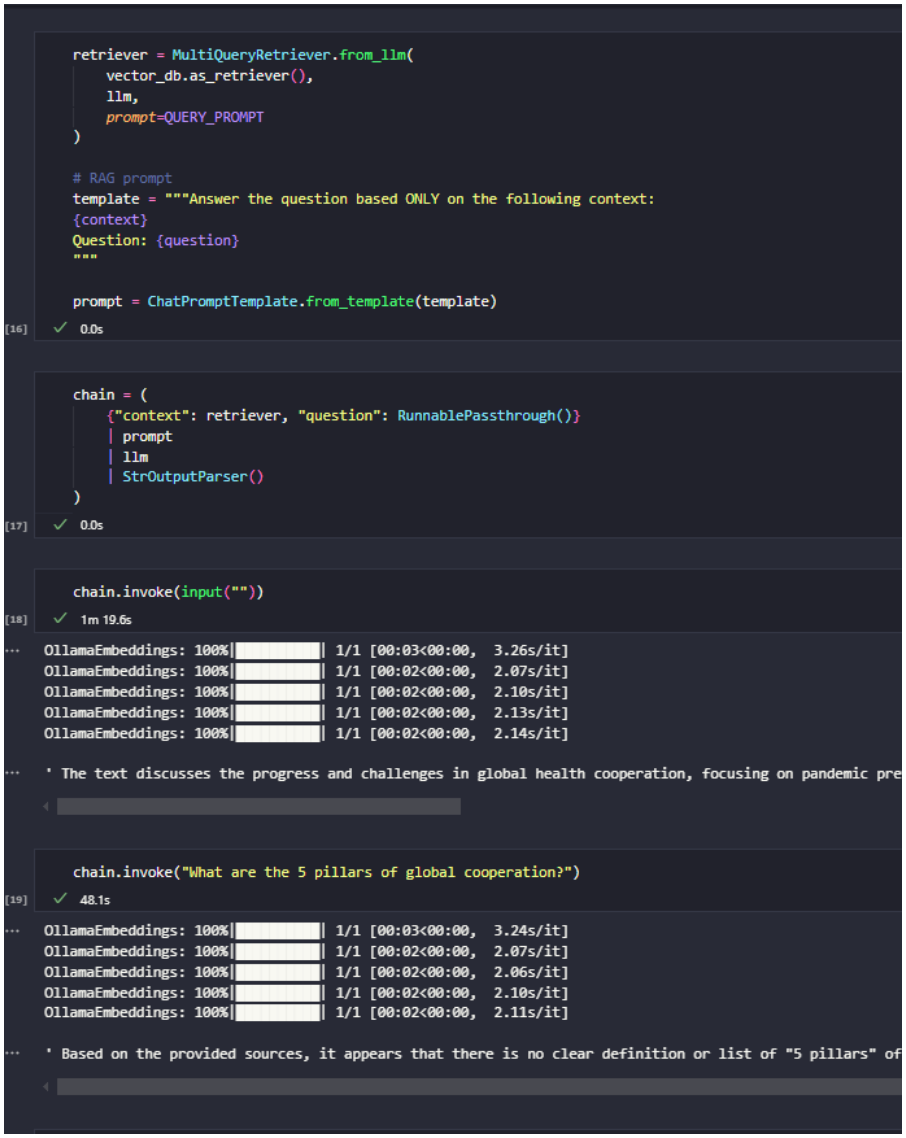
menganalisis data secara lebih efektif dan memberikan solusi pencarian informasi yang cerdas dan terstruktur.

Gambar 3. 26 Ollama Model Minstral IPYNB

Setelah berhasil mengintegrasikan model dengan database vektor, tahap selanjutnya adalah melakukan pengujian untuk menguji performa model *Retrieval-Augmented Generation*(RAG). Pengujian ini dilakukan dengan mengunggah *file* PDF ke dalam sistem dan memberikan *prompt* atau pertanyaan yang berkaitan dengan isi dokumen tersebut. Tujuan utama dari pengujian ini adalah untuk mengevaluasi kemampuan model dalam memahami, menganalisis, dan memberikan jawaban yang relevan berdasarkan informasi yang terkandung dalam dokumen.

Hasil pengujian yang ditampilkan pada Gambar 3.27 menunjukkan bahwa model mampu memberikan keluaran dengan tingkat kemiripan yang tinggi dengan pdf. Jawaban yang dihasilkan tidak hanya konsisten tetapi juga sesuai dengan isi dokumen PDF yang diuji, menandakan integrasi yang sukses antara model dan database vektor. Hal ini membuktikan bahwa sistem dapat secara efektif memanfaatkan data vektor untuk mendukung analisis berbasis dokumen, sekaligus memberikan informasi yang tepat dan kontekstual sesuai dengan pertanyaan yang diajukan.

Keakuratan hasil ini juga menunjukkan kemampuan model dalam memahami hubungan antara data vektor yang tersimpan di database dengan konteks informasi yang terdapat dalam dokumen. Dengan kemampuan tersebut, sistem ini menjadi alat yang andal dan efisien untuk pengolahan dokumen digital, analisis data, serta penelusuran informasi berbasis dokumen.



Gambar 3. 27 Pengujian Ollama Model IPYNB

3.2.6 Melakukan *Deployment model* RAG dengan streamlit

Setelah berhasil menyelesaikan pengembangan prototipe sistem *Retrieval-Augmented Generation*(RAG) dengan memanfaatkan model Ollama, tahap selanjutnya adalah melanjutkan ke proses implementasi dalam bentuk aplikasi berbasis web. Proses ini direncanakan dengan menggunakan Streamlit sebagai kerangka kerja utama untuk membangun aplikasi.

Streamlit dipilih karena kemampuannya yang unggul dalam menyediakan alat untuk membangun antarmuka pengguna (UI) yang sederhana namun

fungsional. *Framework* ini memungkinkan integrasi langsung dengan *model machine learning*, menjadikannya ideal untuk membangun aplikasi yang responsif dan intuitif. Dengan pendekatan ini, aplikasi berbasis RAG yang mengadopsi model Ollama diharapkan dapat memberikan kemudahan akses bagi pengguna, memungkinkan untuk memanfaatkan teknologi RAG tanpa memerlukan pengetahuan teknis yang mendalam.Tahap pembuatan website ini akan mencakup beberapa langkah penting, antara lain:

1. Membangun function *extract_model_names()*

Gambar 3.28 menunjukkan fungsi *extract_model_names()* yang digunakan untuk mengumpulkan dan mengembalikan daftar model yang tersedia pada Ollama dalam bentuk tuple. Fungsi ini pertama-tama mendefinisikan data model yang disimulasikan dalam bentuk *list of dictionaries*, di mana setiap *dictionary* berisi informasi mengenai model dengan kunci "model" yang mencantumkan nama model beserta versinya (contohnya "llama2:*latest*"). Selanjutnya, fungsi ini mengekstrak nama model dari setiap *dictionary* menggunakan pemahaman list dan mengonversinya menjadi tuple. Tuple yang dihasilkan kemudian dikembalikan sebagai *output* fungsi. Dengan demikian, fungsi ini berperan penting dalam mengidentifikasi dan memperoleh daftar model yang tersedia untuk digunakan dalam aplikasi Ollama, yang dapat mempermudah proses pemilihan model yang sesuai untuk kebutuhan selanjutnya.

```python
def _extract_model_names() -> Tuple[str, ...]:

    models_info = [
        {"model": "llama2:latest"},
        {"model": "llama3.2:latest"},
        {"model": "mistral:latest"}
    ]

    # Extract model names
    model_names = tuple(model["model"] for model in models_info)
    return model_names
```

Gambar 3. 28 Function extract_model_names

2. Membangun function *create_avector_db()*

Gambar 3.28 menunjukkan Fungsi *create_vector_db* dirancang untuk membangun basis data vektor dari *file* PDF yang diunggah oleh pengguna. Fungsi

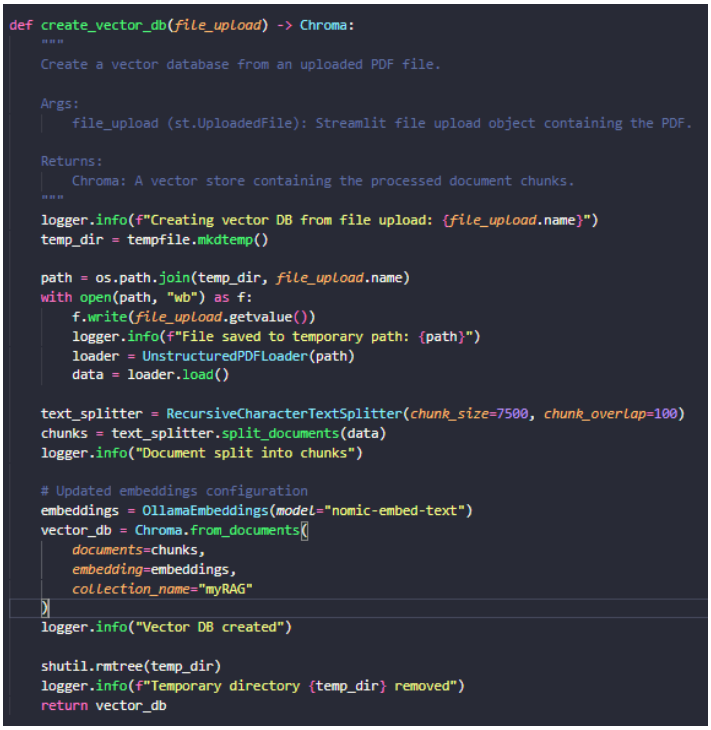
ini dimulai dengan menerima objek *file* yang diunggah melalui Streamlit dan menyimpan *file* tersebut di direktori sementara.

Setelah *file* disalin ke direktori tersebut, fungsi ini memuat data dari PDF menggunakan *UnstructuredPDFLoader*, yang bertugas mengonversi *file* PDF menjadi format yang dapat diproses lebih lanjut. Kemudian, dokumen yang telah dimuat dibagi menjadi potongan-potongan teks menggunakan *RecursiveCharacterTextSplitter*, dengan panjang potongan yang ditentukan dan overlap untuk menjaga konsistensi informasi antar potongan. Potongan-potongan teks ini selanjutnya digunakan untuk membentuk basis data vektor.

Embedding yang dihasilkan dari model OllamaEmbeddings dengan model "nomic-embed-text" digunakan untuk mentransformasi potongan-potongan teks menjadi vektor-vektor numerik. Proses ini bertujuan untuk merepresentasikan teks dalam bentuk yang dapat diproses oleh algoritma komputer, sehingga informasi yang terkandung dalam teks dapat dianalisis dan dimanipulasi secara lebih efisien. Model "nomic-embed-text" dirancang untuk menghasilkan embedding berkualitas tinggi, yang dapat mencerminkan konteks semantik dari teks secara akurat, sehingga sangat berguna dalam berbagai aplikasi berbasis kecerdasan buatan.

Vektor-vektor yang dihasilkan dari *embedding* tersebut kemudian disimpan dalam basis data vektor menggunakan Chroma. Chroma bertugas mengelola vektor-vektor ini dengan mengorganisasikannya dalam koleksi yang diberi nama "myRAG." Dengan menggunakan basis data vektor, proses pencarian informasi dari teks dapat dilakukan secara lebih cepat dan efisien melalui metode pencarian berbasis kesamaan (*similarity search*). Hal ini memungkinkan pengguna untuk mendapatkan informasi yang relevan dari dokumen dengan waktu respon yang lebih singkat, menjadikan teknologi ini sangat bermanfaat dalam berbagai skenario, seperti pembuatan chatbot, analisis data teks, atau implementasi sistem penjawab pertanyaan. Setelah proses pembuatan basis data vektor selesai, direktori sementara dihapus untuk membersihkan sumber daya yang digunakan. Fungsi ini memberikan kemudahan dalam mengubah *file* PDF menjadi vektor-vektor yang dapat digunakan

untuk aplikasi pencarian atau pemrosesan teks berbasis AI, seperti dalam sistem *Retrieval-Augmented Generation*(RAG).



```python
def create_vector_db(file_upload) -> Chroma:
    """
    Create a vector database from an uploaded PDF file.

    Args:
        file_upload (st.UploadedFile): Streamlit file upload object containing the PDF.

    Returns:
        Chroma: A vector store containing the processed document chunks.
    """
    logger.info(f"Creating vector DB from file upload: {file_upload.name}")
    temp_dir = tempfile.mkdtemp()

    path = os.path.join(temp_dir, file_upload.name)
    with open(path, "wb") as f:
        f.write(file_upload.getvalue())
        logger.info(f"File saved to temporary path: {path}")
        loader = UnstructuredPDFLoader(path)
        data = loader.load()

    text_splitter = RecursiveCharacterTextSplitter(chunk_size=7500, chunk_overlap=100)
    chunks = text_splitter.split_documents(data)
    logger.info("Document split into chunks")

    # Updated embeddings configuration
    embeddings = OllamaEmbeddings(model="nomic-embed-text")
    vector_db = Chroma.from_documents(
        documents=chunks,
        embedding=embeddings,
        collection_name="myRAG"
    )
    logger.info("Vector DB created")

    shutil.rmtree(temp_dir)
    logger.info(f"Temporary directory {temp_dir} removed")
    return vector_db
```

Gambar 3. 29 Function create_vector_db

3. Membangun function *process_question()*

Gambar 3.29 menunjukkan Fungsi *process_question()* dirancang untuk memproses pertanyaan pengguna dengan memanfaatkan basis data vektor dan model bahasa yang dipilih. Proses dimulai dengan mencatat pertanyaan yang diajukan serta model yang digunakan. Fungsi ini kemudian menginisialisasi model bahasa (LLM) dan menggunakan *template prompt* untuk menghasilkan dua versi alternatif dari pertanyaan pengguna. Strategi ini bertujuan untuk meningkatkan kualitas hasil pencarian dari basis data vektor dengan mempertimbangkan berbagai perspektif terhadap pertanyaan yang sama. Dengan menggunakan fitur *MultiQueryRetriever*, dokumen relevan diambil dari basis data vektor berdasarkan hasil transformasi pertanyaan.

Setelah dokumen relevan berhasil diambil, fungsi ini menggunakan *template prompt* tambahan untuk memberikan jawaban terhadap pertanyaan

pengguna berdasarkan konteks yang dihasilkan oleh dokumen tersebut. Proses ini melibatkan pembentukan *pipeline* yang terdiri atas beberapa langkah, termasuk ekstraksi konteks, formulasi jawaban melalui LLM, dan pengolahan keluaran menjadi teks. Pendekatan ini memastikan bahwa jawaban diberikan secara eksklusif berdasarkan informasi yang tersedia di basis data, sehingga meningkatkan relevansi dan akurasi jawaban. Fungsi ini sangat cocok untuk digunakan dalam aplikasi pencarian berbasis konteks dan sistem tanya jawab berbasis dokumen.

```python
def process_question(question: str, vector_db: Chroma, selected_model: str) -> str:
    logger.info(f"Processing question: {question} using model: {selected_model}")

    # Initialize LLM
    llm = ChatOllama(model=selected_model)

    # Query prompt template
    QUERY_PROMPT = PromptTemplate(
        input_variables=["question"],
        template="""You are an AI language model assistant. Your task is to generate 2
        different versions of the given user question to retrieve relevant documents from
        a vector database. By generating multiple perspectives on the user question, your
        goal is to help the user overcome some of the limitations of the distance-based
        similarity search. Provide these alternative questions separated by newlines.
        Original question: {question}""",
    )

    # Set up retriever
    retriever = MultiQueryRetriever.from_llm(
        vector_db.as_retriever(),
        llm,
        prompt=QUERY_PROMPT
    )

    # RAG prompt template
    template = """Answer the question based ONLY on the following context:
    {context}
    Question: {question}
    """

    prompt = ChatPromptTemplate.from_template(template)

    # Create chain
    chain = (
        {"context": retriever, "question": RunnablePassthrough()}
        | prompt
        | llm
        | StrOutputParser()
    )

    response = chain.invoke(question)
    logger.info("Question processed and response generated")
    return response
```

Gambar 3. 30 Pengujian Function process_question()

4. Membangun function *extract_all_pages_as_images()*

Gambar 3.30 menunjukkan Fungsi *extract_all_pages_as_images()* dirancang untuk mengekstraksi seluruh halaman dari sebuah *file* PDF dan

mengubahnya menjadi objek gambar. Fungsi ini memanfaatkan objek unggahan *file* dari Streamlit sebagai masukan, di mana *file* PDF yang diunggah diproses menggunakan pustaka *pdfplumber*. Setiap halaman dalam PDF diubah menjadi gambar menggunakan metode *to_image()* yang tersedia dalam pustaka tersebut, dan hasilnya disimpan dalam bentuk daftar yang berisi objek gambar dari setiap halaman PDF. Fungsi ini juga dilengkapi dengan mekanisme pencatatan (*logging*) untuk memantau proses ekstraksi.

Pendekatan ini memungkinkan representasi visual dari dokumen PDF untuk berbagai keperluan analisis, seperti ekstraksi data berbasis gambar atau pembuatan antarmuka yang menampilkan dokumen secara visual. Dengan memanfaatkan kemampuan pustaka Streamlit untuk mengunggah *file*, fungsi ini dapat dengan mudah diintegrasikan ke dalam aplikasi berbasis web yang berfokus pada pengolahan dokumen. Selain itu, penggunaan pustaka *pdfplumber* memastikan hasil konversi yang akurat dan kompatibel dengan berbagai format PDF. Hal ini menjadikan fungsi ini relevan untuk digunakan dalam penelitian yang membutuhkan manipulasi dan visualisasi dokumen berbasis gambar.

```python
@st.cache_data
def extract_all_pages_as_images(file_upload) -> List[Any]:
    """
    Extract all pages from a PDF file as images.

    Args:
        file_upload (st.UploadedFile): Streamlit file upload object containing the PDF.

    Returns:
        List[Any]: A list of image objects representing each page of the PDF.
    """
    logger.info(f"Extracting all pages as images from file: {file_upload.name}")
    pdf_pages = []
    with pdfplumber.open(file_upload) as pdf:
        pdf_pages = [page.to_image().original for page in pdf.pages]
    logger.info("PDF pages extracted as images")
    return pdf_pages
```

Gambar 3. 31 Function extract_all_pages_as_images()

5. Membangun function *delete_vector_db*

Gambar 3.31 menunjukkan Fungsi delete_vector_db digunakan untuk menghapus V*ector database* beserta semua data vektor sementara yang terkait di

dalam *session state*. Fungsi ini memiliki peran untuk menerima parameter *vector_db* yang bersifat opsional dan akan memproses penghapusan hanya jika parameter tersebut tidak bernilai None. Proses ini mencakup pemanggilan metode *delete_collection()* untuk menghapus koleksi data dalam basis data vektor dan menghapus elemen terkait dari *session state* Streamlit, seperti *pdf_pages*, *file_upload*, dan vector_db.

Jika penghapusan berhasil, fungsi akan memberikan pesan sukses kepada pengguna melalui antarmuka Streamlit, diikuti dengan pemanggilan st.rerun() untuk memuat ulang aplikasi agar *session state* yang diperbarui diterapkan. Sebaliknya, jika basis data vektor tidak ditemukan (bernilai *None*), fungsi akan memberikan pesan kesalahan kepada pengguna dan mencatat peringatan dalam *log*. Fungsi ini sangat berguna dalam aplikasi berbasis Streamlit yang menggunakan basis data vektor untuk memproses dokumen, memungkinkan pengelolaan data sementara yang efisien dan memastikan kebersihan memori saat data tidak lagi diperlukan.

```python
def delete_vector_db(vector_db: Optional[Chroma]) -> None:
    """
    Delete the vector database and clear related session state.

    Args:
        vector_db (Optional[Chroma]): The vector database to be deleted.
    """
    logger.info("Deleting vector DB")
    if vector_db is not None:
        vector_db.delete_collection()
        st.session_state.pop("pdf_pages", None)
        st.session_state.pop("file_upload", None)
        st.session_state.pop("vector_db", None)
        st.success("Collection and temporary files deleted successfully.")
        logger.info("Vector DB and related session state cleared")
        st.rerun()
    else:
        st.error("No vector database found to delete.")
        logger.warning("Attempted to delete vector DB, but none was found")
```

Gambar 3. 32 Function extract_all_pages_as_images()

3.2.7 Melakukan Perancangan Model RAG dengan Gpt 4.o

Pembuatan model RAG GPT bertujuan untuk melakukan Perbandingan antara GPT dan Ollama agar dapat melihat keunggulan masing-masing dalam penerapan model RAG. Penilaian dilakukan berdasarkan kecepatan pencarian, akurasi jawaban, dan kemampuan memahami dokumen yang rumit. Hasil dari perbandingan ini akan membantu perusahaan menentukan platform mana yang paling cocok untuk kebutuhan para karyawan dalam menyediakan akses informasi yang mudah dan efisien di tempat kerja. Berikut adalah tahapan dalam membangun model *Retrieval-Augmented Generation*(RAG) menggunakan GPT. Berikut merupakan tahapan dari pembuatan model RAG menggunakan GPT :

1. Membangun Data Loader

Gambar 3.33 menunjukkan kode untuk memuat dan membaca dokumen PDF lokal dengan bantuan pustaka pemrosesan dokumen. *File* PDF yang akan diproses ditentukan melalui variabel *local_path*, yang mengacu pada *file* "WEF_The_Global_Cooperation_Barometer_2024.pdf" yang tersimpan di folder lokal pengguna. Jika *file* ditemukan (variabel *local_path* tidak kosong), kode ini akan menggunakan kelas *UnstructuredPDFLoader* untuk membaca dokumen dan mengekstrak data di dalamnya. Hasil ekstraksi ini kemudian dapat digunakan untuk analisis atau proses lebih lanjut. Namun, jika *file* tidak ditemukan atau belum diunggah, kode akan menampilkan pesan kepada pengguna menggunakan perintah print.

Pendekatan ini dirancang untuk mempermudah proses pengolahan dokumen dalam berbagai aplikasi berbasis data, seperti analisis teks dokumen, ekstraksi informasi, atau pembuatan model *Retrieval-Augmented Generation*(RAG). Dengan menggunakan pustaka *UnstructuredPDFLoader*, dokumen PDF dapat diproses secara otomatis, memungkinkan pengambilan data penting dari *file* tanpa memerlukan intervensi manual yang memakan waktu.

Proses ini sangat berguna dalam situasi di mana dokumen yang dianalisis memiliki jumlah halaman yang banyak atau mengandung informasi yang kompleks.Hasil ekstraksi yang dihasilkan oleh pustaka ini dapat dimanfaatkan untuk berbagai keperluan, seperti analisis data, pembuatan ringkasan dokumen,

atau integrasi ke dalam model kecerdasan buatan yang membutuhkan masukan berbasis teks.

```
from langchain_community.document_loaders import UnstructuredPDFLoader
from langchain_community.document_loaders import OnlinePDFLoader
local_path = "WEF_The_Global_Cooperation_Barometer_2024.pdf"

# Local PDF file uploads
if local_path:
  loader = UnstructuredPDFLoader(file_path=local_path)
  data = loader.load()
else:
  print("Upload a PDF file")
```

Gambar 3. 33 Data Loader GPT IPYNB

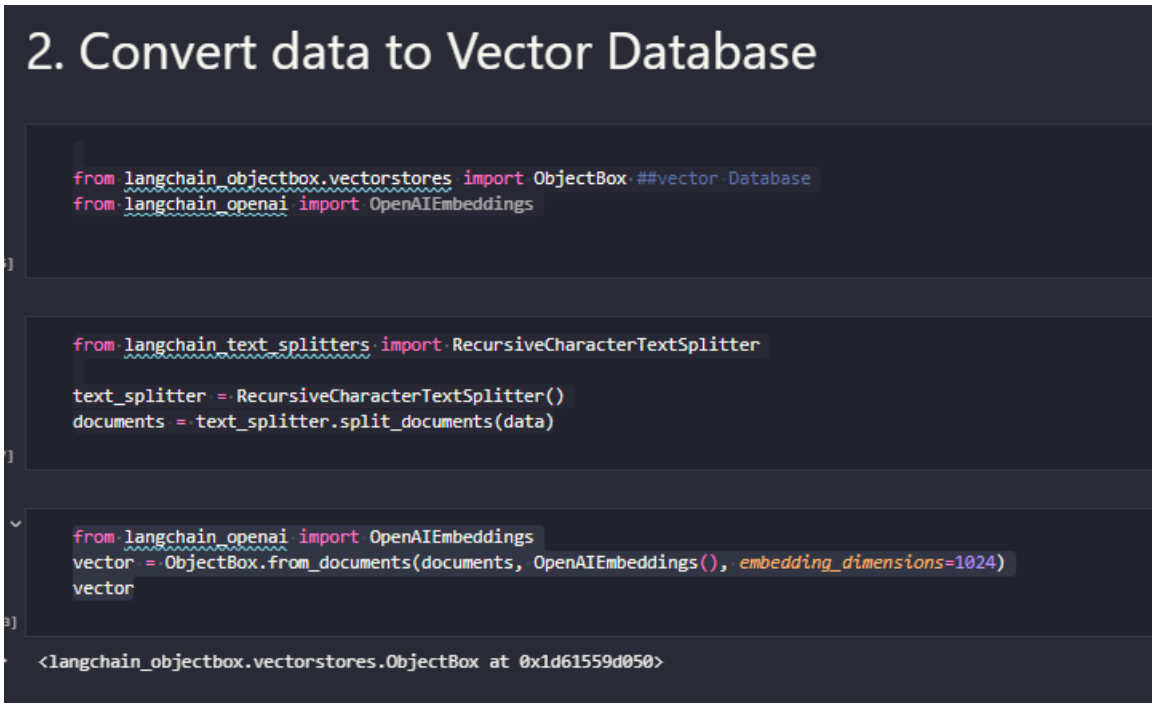
2. Membangun Vector Database

Gambar 3.34 menunjukkan proses pembuatan dan pengelolaan database vektor menggunakan pustaka *ObjectBox* yang dikombinasikan dengan *embedding* dari OpenAI. Langkah pertama adalah memecah dokumen teks menjadi bagian-bagian kecil menggunakan *RecursiveCharacterTextSplitter*. Metode ini membantu dalam mempersiapkan data untuk diproses lebih lanjut dengan membagi dokumen menjadi potongan-potongan teks yang lebih terstruktur dan mudah diolah. Setelah teks dipisahkan, setiap potongan dokumen diubah menjadi representasi vektor menggunakan *OpenAIEmbeddings*, dengan dimensi *embedding* yang disesuaikan (1024 dimensi).

Representasi vektor ini kemudian disimpan dalam database vektor *ObjectBox*. Database ini memungkinkan penyimpanan dan pengambilan dokumen berbasis vektor dengan efisien, yang sangat berguna dalam aplikasi seperti pencarian informasi atau pengolahan bahasa alami.

Dengan pendekatan ini, informasi dalam dokumen dapat diakses melalui teknik pencarian berbasis vektor, sehingga mendukung analisis data yang lebih cepat dan akurat. Penggunaan database vektor juga mempermudah integrasi ke

dalam model pembelajaran mesin atau aplikasi AI lainnya yang memerlukan data dalam bentuk *embedding*.



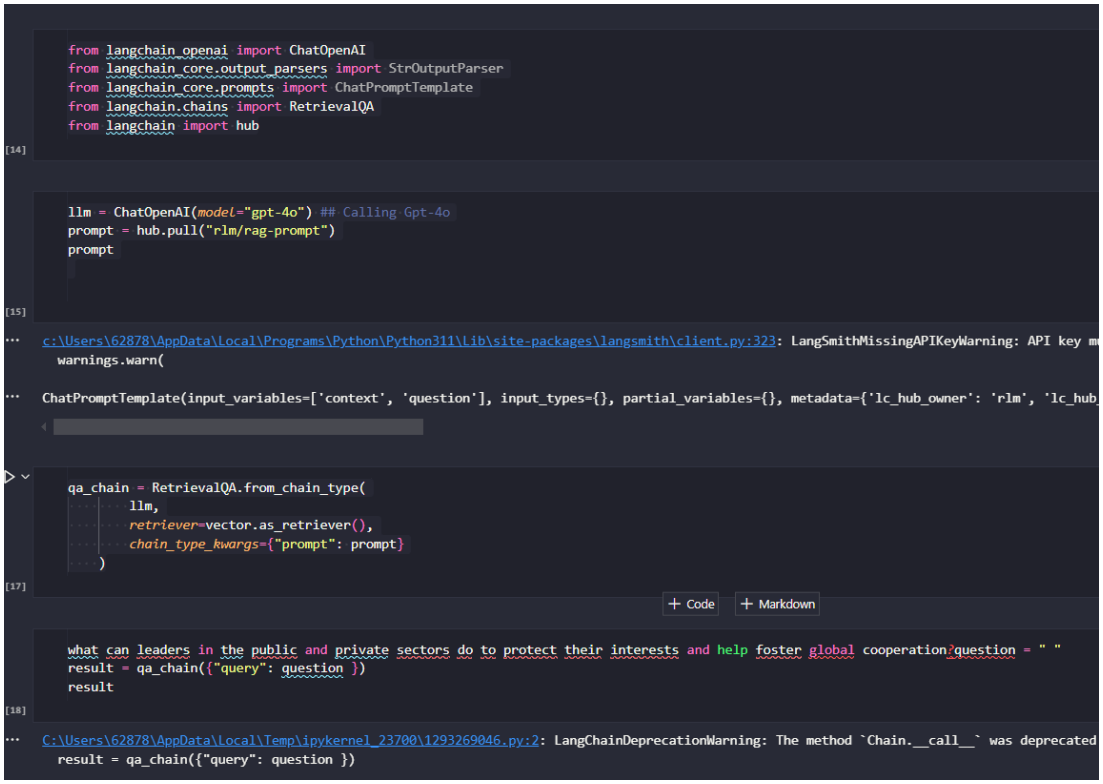
Gambar 3. 34 Vector Database GPT IPYNB

3. Membangun *RAG Pipeline GPT*

Gambar 3.35 menunjukkan proses implementasi model *Retrieval-Augmented Generation*(RAG) menggunakan *LangChain* dengan model GPT-4o sebagai inti pemrosesan bahasa. Langkah pertama adalah menginisialisasi model ChatOpenAI dengan menggunakan GPT-4o dan mengunduh *prompt* RAG dari repositori yang tersedia menggunakan fungsi hub.pull. *Prompt* ini dirancang untuk membantu model memberikan jawaban yang relevan berdasarkan konteks yang diperoleh dari database vektor.

Model RAG dibuat dengan menggunakan kelas *RetrievalQA*, yang menggabungkan kemampuan *retriever* (pengambil dokumen relevan) dari database vektor ObjectBox dengan model GPT-4o untuk menghasilkan jawaban. Dengan memberikan pertanyaan pada *pipeline* RAG, model mencari dokumen relevan

melalui *retriever*, kemudian menghasilkan respons berbasis teks yang sesuai dengan konteks yang ditemukan. Hasil akhir diproses dan ditampilkan dalam format yang terstruktur menggunakan print. Pendekatan ini sangat bermanfaat dalam mendukung pengambilan keputusan berbasis data di berbagai sektor, karena memungkinkan pengguna untuk mendapatkan informasi yang terfokus dan relevan dari sumber data yang besar dan kompleks.

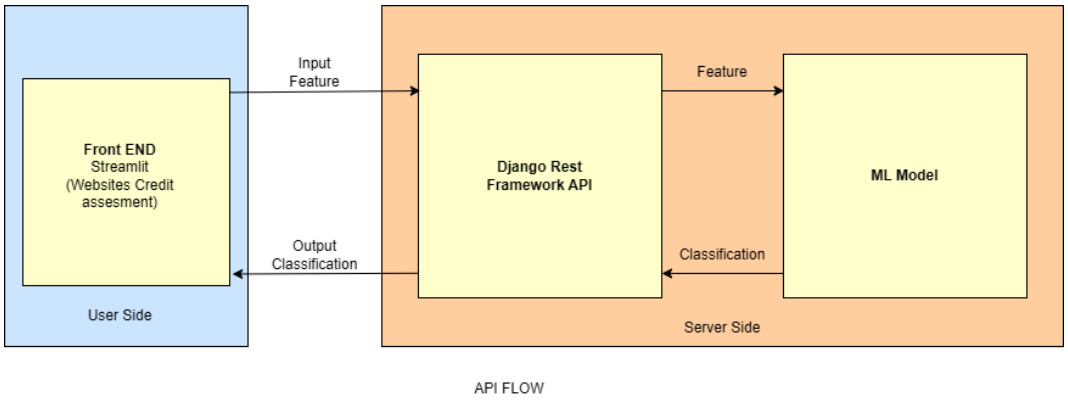


Gambar 3. 35 RAG Pipeline GPT

3.2.8 Melakukan perancangan API model *credit assesment* menggunakan Django Rest API

Pembuatan API untuk model *credit assessment* menggunakan *Django REST Framework* (DRF) dengan tujuan mempermudah integrasi model ke aplikasi Streamlit, dan juga platform lainya. Dengan menjadikan model sebagai API, klasifikasi risiko kredit dapat diakses secara *real-time* melalui antarmuka yang interaktif di Streamlit.

Flow API klasifikasi *credit assessment* ini dimulai dari *front-end* berbasis *Streamlit* yang berfungsi sebagai antarmuka pengguna untuk memasukkan data fitur yang diperlukan. Data tersebut kemudian dikirim ke *Django Rest Framework API* di sisi server, yang bertugas sebagai penghubung antara *front-end* dan model pembelajaran mesin (*ML Model*). API memproses data yang diterima dan meneruskannya ke *ML Model*, yang kemudian melakukan klasifikasi berdasarkan data fitur tersebut. Hasil klasifikasi dari *ML Model* dikembalikan ke API, lalu diteruskan kembali ke *front-end* untuk ditampilkan kepada pengguna. Alur ini memastikan komunikasi yang terintegrasi antara komponen *user side* dan *server side*.



Gambar 3. 36 API Diagram Flow

1. Membangun pipeline preprocessing

Gambar 3.36 berfungsi untuk memproses dataset besar dalam format CSV secara efisien. Dataset dibaca secara bertahap menggunakan *chunks* melalui fungsi pandas.read_csv agar penggunaan memori lebih efisien, terutama jika *dataset* memiliki ukuran besar.

Setelah seluruh potongan data dibaca, data tersebut digabungkan menjadi satu *DataFrame* untuk diproses lebih lanjut. Pendekatan ini membantu menghindari masalah *out-of-memory error* yang sering terjadi saat memproses data besar secara langsung.

```python
# Load dataset secara bertahap dengan chunksize
chunks = pd.read_csv('C:/Users/62878/OneDrive/Documents/magang/last_master.csv', chunksize=10000, low_memory=False)

# Gabungkan semua chunks menjadi satu DataFrame
df = pd.concat(chunks, ignore_index=True)
```

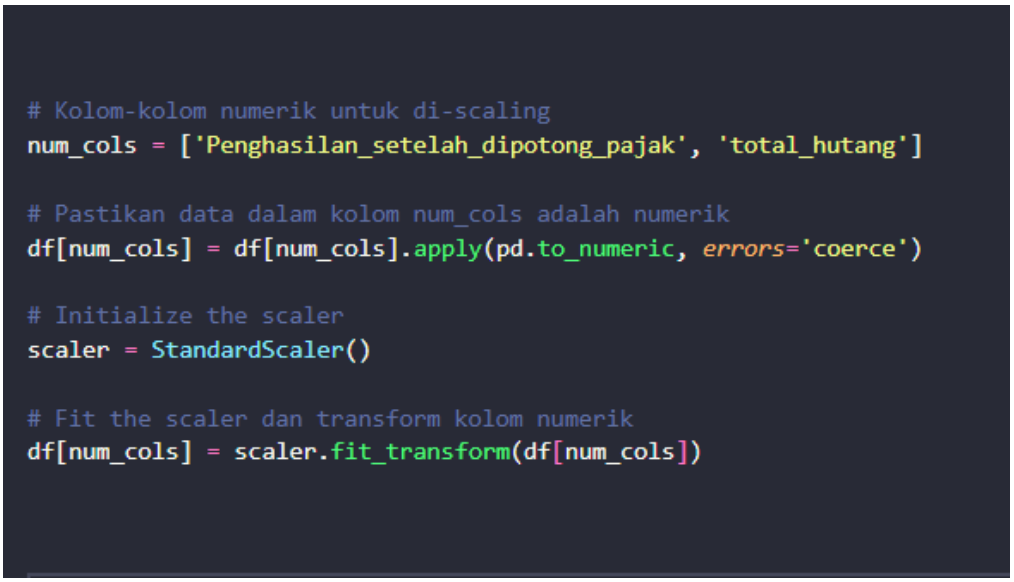
Gambar 3. 37 Input Dataset Preprocessing

Gambar 3.37 menunjukkan langkah selanjutnya dalam memproses dataset, yaitu melakukan *encoding* pada kolom-kolom yang berisi data kategorikal, seperti tingkat kriminalitas, tujuan peminjaman, dan situasi tempat tinggal. Data kategorikal ini tidak dapat langsung digunakan dalam model pembelajaran mesin, sehingga perlu diubah menjadi format numerik. Untuk setiap kolom, kategori-kategori spesifik didefinisikan sebelumnya dalam sebuah *dictionary* bernama *categories*. *Dictionary* ini menentukan urutan kategori yang akan digunakan sebagai dasar dalam proses konversi. Teknik *ordinal encoding* diterapkan untuk mengubah kategori-kategori tersebut menjadi nilai numerik berdasarkan urutan yang telah ditentukan. Proses ini memberikan nilai unik pada setiap kategori dalam kolom, yang mencerminkan tingkat prioritas atau skala relatif antar kategori. Dengan pendekatan ini, model dapat memahami hubungan hierarkis antar kategori, yang membantu meningkatkan akurasi klasifikasi. *Encoding* juga menangani kategori yang tidak dikenal dengan mengubahnya menjadi nilai default tertentu, sehingga menjaga integritas data selama proses analisis.

```python
# Kolom-kolom untuk ordinal encoding
ordinal_columns = ['kriminalitas', 'narkoba', 'terkena_kriminalitas_sebagai_korban',
                   'Tujuan_Peminjaman','huru_hara','situasi_tempat_tinggal.1',
                   'Jumlah_Hari_Tunggakan','situasi_tempat_tinggal','potensi_daerah']

# Mapping untuk ordinal encoding
categories = {
    'kriminalitas': ['Rendah', 'Sedang', 'Tinggi'],
    'narkoba': ['Tidak', 'Iya'],
    'terkena_kriminalitas_sebagai_korban': ['Iya', 'Tidak'],
    'Tujuan_Peminjaman': ['Modal Kerja', 'Ekspansi Bisnis', 'Inovasi dan R&D',
                          'Pengembangan Teknologi', 'Pembelian Aset Tetap', 'Cadangan Likuiditas',
                          'Penyelamatan Bisnis', 'Konsolidasi Utang', 'Akuisisi atau Merger',
                          'Modal Ventura'],
    'huru_hara': ['Rendah', 'Tinggi', 'Sedang'],
    'situasi_tempat_tinggal.1': ['rawan bencana', 'kriminalitas tinggi', 'padat penduduk', 'area kumuh'],
    'Jumlah_Hari_Tunggakan': ['1-29(hari)', '30-90(hari)', '30-60(hari)', '>60(hari)', '>90(hari)'],
    'situasi_tempat_tinggal': ['Kriminalitas', 'BPR', 'Slum', 'Padat'],
    'potensi_daerah': ['Rendah', 'Sedang', 'Tinggi', 'Tidak Ada'],
}

# Remove duplicates from each category list
for col in categories:
    categories[col] = list(dict.fromkeys(categories[col]))

# Buat ordinal encoder dengan mapping categories dan menangani unknown values
ordinal_encoder = OrdinalEncoder(
    categories=[categories[col] for col in ordinal_columns],
    handle_unknown='use_encoded_value',
    unknown_value=-1
)
```

Gambar 3. 38 Encoding API

Gambar 3.38 menunjukan proses fitur numerik seperti Penghasilan_setelah_dipotong_pajak dan total_hutang juga diproses menggunakan *StandardScaler*. Teknik ini mengubah data numerik menjadi bentuk terstandardisasi, sehingga nilai-nilai dalam kolom tersebut memiliki rata-rata 0 dan standar deviasi 1. Standardisasi ini penting untuk memastikan bahwa data numerik dalam berbagai skala dapat digunakan secara efektif dalam model pembelajaran mesin tanpa mempengaruhi performa model.



```
# Kolom-kolom numerik untuk di-scaling
num_cols = ['Penghasilan_setelah_dipotong_pajak', 'total_hutang']

# Pastikan data dalam kolom num_cols adalah numerik
df[num_cols] = df[num_cols].apply(pd.to_numeric, errors='coerce')

# Initialize the scaler
scaler = StandardScaler()

# Fit the scaler dan transform kolom numerik
df[num_cols] = scaler.fit_transform(df[num_cols])
```

Gambar 3. 39 Standard Scaler API

Setelah data selesai ditransformasi, seperti yang ditunjukkan pada Gambar 3.39, langkah berikutnya adalah menggabungkan seluruh proses transformasi menggunakan objek *ColumnTransformer*. Objek ini memungkinkan penggabungan berbagai jenis transformasi dalam satu langkah yang efisien.

Dalam kasus ini, *ColumnTransformer* menggabungkan dua proses utama, yaitu *ordinal encoding* untuk data kategorikal dan *scaling* untuk data numerik. *Ordinal encoding* mengubah data kategorikal menjadi format numerik sesuai dengan urutan kategori yang telah ditentukan, sedangkan *scaling* mengubah nilai-nilai numerik menjadi bentuk terstandarisasi. Dengan menggunakan metode ini, semua data diproses dengan metode yang sesuai, tanpa perlu melakukan transformasi terpisah untuk setiap jenis data.

Penggunaan *ColumnTransformer* berfungsi untuk melakukan menyederhanakan proses transformasi. Semua langkah-langkah transformasi dapat dilakukan secara bersamaan dalam satu objek, yang diterapkan pada seluruh dataset sekaligus. Ini memastikan bahwa setiap kolom diproses dengan benar sesuai dengan jenis datanya, sehingga tidak ada data yang terlewat atau salah diproses. Selain itu, penggunaan *ColumnTransformer* juga mempermudah pengelolaan kode, karena seluruh proses transformasi dapat dilakukan dalam satu langkah terorganisir yang lebih mudah dipahami dan dipelihara.

Sebelum proses penggabungan dilakukan, kolom-kolom tertentu, seperti *cluster* dan *cluster*_label, dihapus dari dataset., yaitu situasi di mana informasi yang tidak relevan atau tidak seharusnya digunakan dalam model tetap diproses dan dapat mempengaruhi hasil analisis atau klasifikasi.

Dengan menghapus kolom-kolom yang tidak diperlukan, dataset menjadi lebih bersih dan relevan, serta hasil transformasi menjadi lebih akurat. Data yang telah dibersihkan dan ditransformasi dengan benar siap untuk tahap analisis atau pemodelan lebih lanjut, sehingga model yang dibangun dapat memberikan hasil yang lebih baik dan lebih dapat diandalkan.

```
# Column transformer untuk menggabungkan kedua jenis encoding
preprocessor = ColumnTransformer(
    transformers=[
        ('ordinal', ordinal_encoder, ordinal_columns),
        ('scaler', scaler, num_cols)
    ],
    remainder='passthrough'  # Menambahkan kolom lainnya ke hasil akhir
)

# Exclude 'cluster' and 'cluster_label' columns if they exist in the dataset
if 'cluster' in df.columns:
    df = df.drop(columns=['cluster'])
if 'cluster_label' in df.columns:
    df = df.drop(columns=['cluster_label'])

# Latih preprocessor pada dataset
preprocessor.fit(df)
```

Gambar 3. 40 Column Transformer API

Setelah seluruh proses selesai, objek *preprocessor* yang berisi semua transformasi ini disimpan dalam *file* .pkl menggunakan *library pickle*. Penyimpanan ini mempermudah penggunaan kembali *preprocessor* untuk klasifikasi atau aplikasi lain tanpa perlu melatih ulang transformasi pada data baru. *File* ini dapat diintegrasikan ke dalam sistem berbasis *Django REST*, sehingga mempermudah pengolahan data yang konsisten di berbagai *platform*.

Gambar 3. 41 Column Transformer API

2. Membangun *Serializers*

Gambar 3.41 mendefinisikan *serializer* yang digunakan dengan *Django REST Framework* (DRF) untuk memproses data input yang diperlukan oleh model klasifikasi. *Serializer* ini berfungsi untuk mengonversi data dari format yang mudah dipahami oleh pengguna, seperti JSON, menjadi format yang dapat diproses oleh aplikasi Python. Begitu pula sebaliknya, *serializer* akan mengonversi data yang dihasilkan oleh model atau aplikasi kembali menjadi format yang dapat dikirim ke *frontend*. Dengan cara ini, *serializer* berperan penting dalam menjaga komunikasi yang lancar antara *frontend* dan *backend*, memastikan bahwa data yang diterima atau dikirim sesuai dengan format yang diinginkan dan dapat diproses dengan benar oleh aplikasi.

Pada kelas *PredictSerializer*, terdapat beberapa *field input* yang telah ditentukan untuk model klasifikasi kredit, yang mencakup dua jenis data utama: numerik dan kategorikal. Untuk data numerik, seperti Penghasilan_setelah_dipotong_pajak, total_hutang, dan total_asset, digunakan FloatField, yang memungkinkan aplikasi untuk menerima nilai numerik dalam bentuk desimal. Data numerik ini penting untuk model klasifikasi karena sering

kali mencerminkan aspek penting dalam analisis kredit, seperti kemampuan membayar dan total kewajiban finansial.

Sementara itu, untuk data kategorikal, seperti huru_hara, kriminalitas, dan Tujuan_Peminjaman, digunakan *CharField*, yang memungkinkan input dalam bentuk teks. Data kategorikal ini biasanya berisi informasi yang harus diproses lebih lanjut, seperti melalui teknik *encoding*, agar model dapat menginterpretasikannya dalam bentuk numerik yang dapat dianalisis.

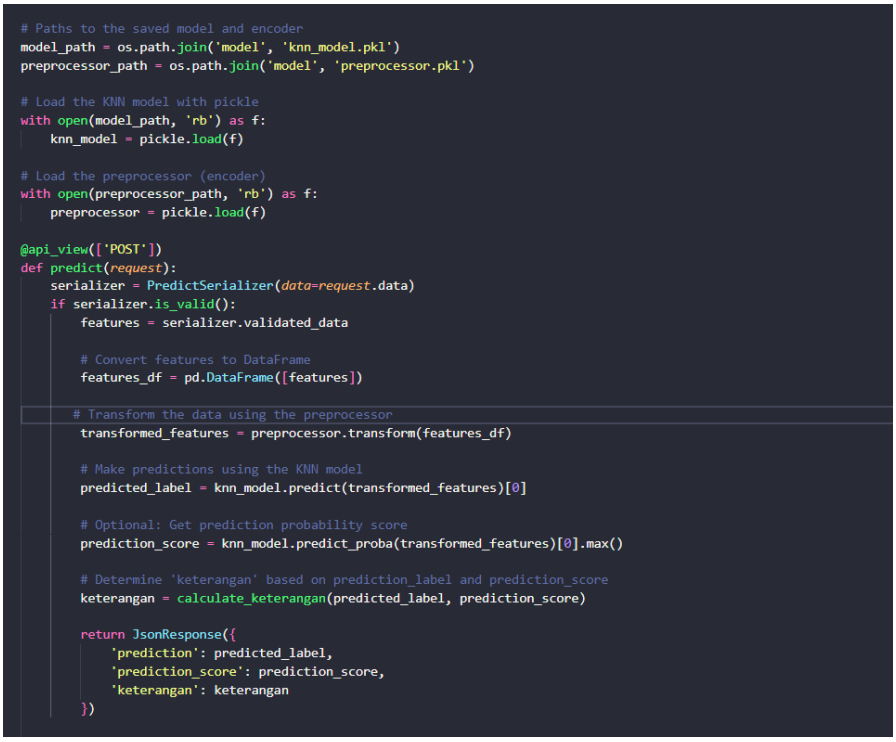
```
from rest_framework import serializers

class PredictSerializer(serializers.Serializer):
    Penghasilan_setelah_dipotong_pajak = serializers.FloatField()
    total_hutang = serializers.FloatField()
    total_asset = serializers.FloatField()
    status_agunan_terburuk = serializers.FloatField()

    huru_hara = serializers.CharField()
    kriminalitas = serializers.CharField()
    Tujuan_Peminjaman = serializers.CharField()
    situasi_tempat_tinggal_1 = serializers.CharField()
    Jumlah_Hari_Tunggakan  = serializers.CharField()
    potensi_daerah = serializers.CharField()
    narkoba = serializers.CharField()
    terkena_kriminalitas_sebagai_korban = serializers.CharField()
```

Gambar 3. 42 Serializer API

3. Membangun *Request Handler* API

Pada Gambar 3.42, fungsi predict digunakan untuk menangani permintaan POST pada API yang menerima data untuk melakukan klasifikasi menggunakan model KNN (*K-Nearest Neighbors*) yang telah dilatih sebelumnya. Ketika permintaan diterima, data input dari pengguna akan diproses menggunakan serializer (*PredictSerializer*). *Serializer* ini memastikan bahwa data yang diterima dalam format JSON valid dan dapat diproses lebih lanjut. Jika data valid, informasi fitur yang dibutuhkan untuk klasifikasi diekstraksi dan dimasukkan ke dalam sebuah *DataFrame*. Data ini kemudian diproses lebih lanjut menggunakan

preprocessor yang telah dilatih, yang sebelumnya telah disimpan dalam bentuk *file* pickle.



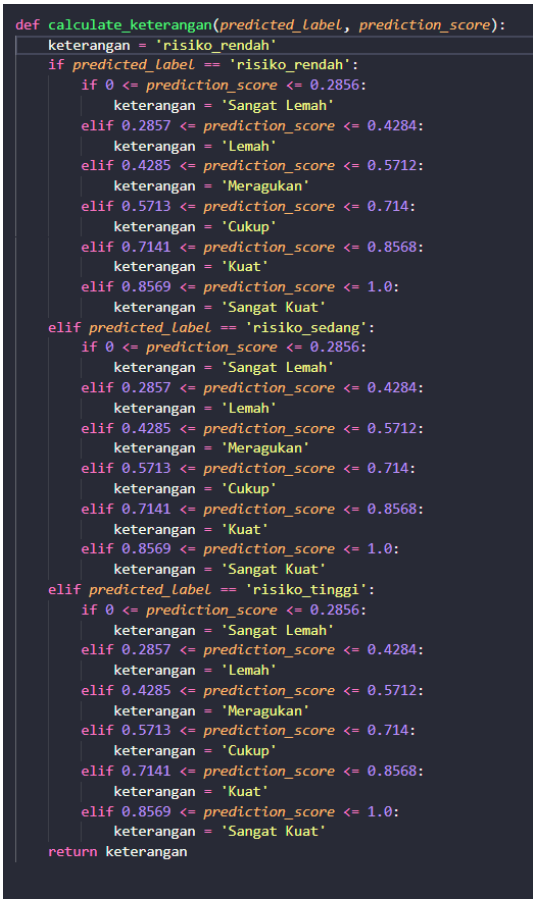
```python
# Paths to the saved model and encoder
model_path = os.path.join('model', 'knn_model.pkl')
preprocessor_path = os.path.join('model', 'preprocessor.pkl')

# Load the KNN model with pickle
with open(model_path, 'rb') as f:
    knn_model = pickle.load(f)

# Load the preprocessor (encoder)
with open(preprocessor_path, 'rb') as f:
    preprocessor = pickle.load(f)

@api_view(['POST'])
def predict(request):
    serializer = PredictSerializer(data=request.data)
    if serializer.is_valid():
        features = serializer.validated_data

        # Convert features to DataFrame
        features_df = pd.DataFrame([features])

        # Transform the data using the preprocessor
        transformed_features = preprocessor.transform(features_df)

        # Make predictions using the KNN model
        predicted_label = knn_model.predict(transformed_features)[0]

        # Optional: Get prediction probability score
        prediction_score = knn_model.predict_proba(transformed_features)[0].max()

        # Determine 'keterangan' based on prediction_label and prediction_score
        keterangan = calculate_keterangan(predicted_label, prediction_score)

        return JsonResponse({
            'prediction': predicted_label,
            'prediction_score': prediction_score,
            'keterangan': keterangan
        })
```

Gambar 3. 43 PredictSerializer Function API

Selanjutnya, setelah data melalui transformasi oleh preprocessor, model KNN yang telah dimuat dari *file* pickle digunakan untuk melakukan klasifikasi berdasarkan data yang telah diproses. Klasifikasi ini dilakukan dengan memanggil metode predict pada model KNN, yang menghasilkan label klasifikasi yang menunjukkan risiko kredit (misalnya, risiko rendah, sedang, atau tinggi).

Pada Gambar 3.43, fungsi calculate_keterangan dipanggil untuk menentukan kategori risiko yang lebih terperinci berdasarkan nilai klasifikasi dan skor probabilitas. Fungsi ini akan memberikan keterangan berupa tingkat kekuatan risiko, seperti "Sangat Lemah", "Lemah", "Cukup", "Kuat", atau "Sangat Kuat", berdasarkan rentang nilai yang telah ditentukan. Keterangan ini memberikan penjelasan lebih lanjut kepada pengguna mengenai tingkat risiko yang ditilai oleh model. Terakhir, hasil klasifikasi, skor probabilitas, dan keterangan dikirim kembali

ke pengguna dalam bentuk JSON melalui *JsonResponse*. Jika data yang diterima tidak valid, API akan mengembalikan pesan kesalahan dengan detail dari error *serializer*, memberikan umpan balik yang jelas kepada pengguna mengenai masalah dengan input yang diberikan. Dengan pendekatan ini, API yang dibangun tidak hanya memberikan klasifikasi yang berguna, tetapi juga memastikan interaksi yang mudah dan informatif dengan pengguna.



```
def calculate_keterangan(predicted_label, prediction_score):
    keterangan = 'risiko_rendah'
    if predicted_label == 'risiko_rendah':
        if 0 <= prediction_score <= 0.2856:
            keterangan = 'Sangat Lemah'
        elif 0.2857 <= prediction_score <= 0.4284:
            keterangan = 'Lemah'
        elif 0.4285 <= prediction_score <= 0.5712:
            keterangan = 'Meragukan'
        elif 0.5713 <= prediction_score <= 0.714:
            keterangan = 'Cukup'
        elif 0.7141 <= prediction_score <= 0.8568:
            keterangan = 'Kuat'
        elif 0.8569 <= prediction_score <= 1.0:
            keterangan = 'Sangat Kuat'
    elif predicted_label == 'risiko_sedang':
        if 0 <= prediction_score <= 0.2856:
            keterangan = 'Sangat Lemah'
        elif 0.2857 <= prediction_score <= 0.4284:
            keterangan = 'Lemah'
        elif 0.4285 <= prediction_score <= 0.5712:
            keterangan = 'Meragukan'
        elif 0.5713 <= prediction_score <= 0.714:
            keterangan = 'Cukup'
        elif 0.7141 <= prediction_score <= 0.8568:
            keterangan = 'Kuat'
        elif 0.8569 <= prediction_score <= 1.0:
            keterangan = 'Sangat Kuat'
    elif predicted_label == 'risiko_tinggi':
        if 0 <= prediction_score <= 0.2856:
            keterangan = 'Sangat Lemah'
        elif 0.2857 <= prediction_score <= 0.4284:
            keterangan = 'Lemah'
        elif 0.4285 <= prediction_score <= 0.5712:
            keterangan = 'Meragukan'
        elif 0.5713 <= prediction_score <= 0.714:
            keterangan = 'Cukup'
        elif 0.7141 <= prediction_score <= 0.8568:
            keterangan = 'Kuat'
        elif 0.8569 <= prediction_score <= 1.0:
            keterangan = 'Sangat Kuat'
    return keterangan
```

Gambar 3. 44 Calculate Keterangan Function API

3.2.9 Melakukan Perancangan Website klasifikasi nasabah credit assesment menggunakan Streamlit

Setelah API selesai dibuat, antarmuka pengguna dibangun menggunakan Streamlit sebagai aplikasi web yang interaktif dan sederhana. Dalam Streamlit, pengguna dapat mengisi form input data kredit, lalu aplikasi akan mengirim data tersebut ke API Django REST Framework.

Hasil klasifikasi dari API kemudian ditampilkan secara real-time di halaman Streamlit. Kombinasi *Django REST Framework* untuk pengelolaan *backend* dan API serta Streamlit untuk *frontend* memungkinkan aplikasi ini memberikan pengalaman pengguna yang mudah, interaktif, dan cepat dalam memperoleh hasil *credit assesment*. Tahap pembuatan website ini akan mencakup beberapa langkah penting, antara lain:

1. Membangun Fungsi get_prediction()

Gambar 3.44 merupakan fungsi get_prediction() yang memiliki tugas untuk mengirimkan data input dalam format JSON ke API menggunakan metode *POST*, sehingga memungkinkan komunikasi antara aplikasi *frontend* dan *backend*. Input berupa fitur-fitur yang telah diisi pengguna dikemas dalam format JSON, lalu dikirim ke URL *endpoint* API yang telah didefinisikan sebelumnya.

```python
# URL endpoint API Django
API_URL = 'http://localhost:8000/api/predict/'

def get_prediction(features):
    """Kirim permintaan POST ke endpoint API dan kembalikan hasilnya."""
    response = requests.post(API_URL, json=features)
    if response.status_code == 200:
        return response.json()
    else:
        st.error("Terjadi kesalahan saat mengambil prediksi.")
        return None
```

Gambar 3. 45 Function get_prediction

Jika permintaan berhasil (status kode 200), fungsi akan mengembalikan hasil klasifikasi yang diterima dari API dalam format JSON, yang kemudian dapat digunakan untuk ditampilkan kepada pengguna. jika terjadi kesalahan, seperti respons gagal dari server, koneksi yang terputus, atau status kode HTTP yang tidak sesuai, fungsi akan menampilkan pesan *error* di Streamlit untuk memberi tahu pengguna bahwa klasifikasi tidak dapat diambil. Dengan cara ini, fungsi tidak hanya bertindak sebagai penghubung antara *frontend* dan *backend* tetapi juga memastikan pengguna mendapatkan umpan balik yang jelas dalam situasi kegagalan.

2. Membangun predict_from_*file*()

Gambar 3.45 merupakan Fungsi predict_from_*file*() dirancang untuk menangani *file* unggahan yang berisi data calon peminjam. Fungsi ini dimulai dengan memeriksa ekstensi *file* yang diunggah pengguna untuk memastikan apakah formatnya didukung, yaitu CSV atau Excel (.xlsx). Berdasarkan jenis *file*, fungsi menggunakan library Pandas untuk membaca data dan mengubahnya menjadi sebuah *DataFrame*, yang memungkinkan akses baris per baris untuk setiap data calon peminjam. Jika *file* memiliki format yang tidak didukung, seperti TXT atau lainnya, fungsi akan menghentikan proses dan memberikan pesan kesalahan melalui antarmuka Streamlit. Setelah *file* berhasil dibaca, fungsi memproses setiap baris data dalam *DataFrame* dengan mengonversinya menjadi *dictionary* menggunakan metode *to_dict().*

```
def predict_from_file(uploaded_file):
    """Baca file dan kirimkan data untuk prediksi."""
    # Cek ekstensi file
    file_extension = uploaded_file.name.split('.')[-1]

    # Baca file sesuai dengan ekstensi
    if file_extension == 'xlsx':
        df = pd.read_excel(uploaded_file)
    elif file_extension == 'csv':
        df = pd.read_csv(uploaded_file)
    else:
        st.error("Format file tidak didukung. Harap unggah file Excel (.xlsx) atau CSV (.csv).")
        return None

    predictions = []

    for _, row in df.iterrows():
        features = row.to_dict()
        result = get_prediction(features)
        if result:
            predictions.append(result)

    return predictions
```

Gambar 3. 46 Function predict_from_file

Setiap *dictionary* yang dihasilkan berisi fitur-fitur calon peminjam, seperti tingkat kriminalitas, total aset, atau riwayat penggunaan narkoba. *Dictionary* ini kemudian dikirim ke fungsi get_prediction() untuk diproses melalui API. Hasil klasifikasi dari setiap baris data dikumpulkan dalam sebuah list bernama predictions. Setelah semua baris diproses, fungsi mengembalikan list tersebut, yang nantinya dapat digunakan untuk menampilkan klasifikasi masing-masing calon peminjam di antarmuka Streamlit. Pendekatan ini memastikan fungsi dapat menangani data dalam jumlah besar dengan efisien.

3. Membangun Tampilan Input Manual

Gambar 3.46 merupakan kode yang berfungsi membangun antarmuka pengguna dengan Streamlit yang memungkinkan pengguna untuk memasukkan data calon peminjam secara manual. Di bagian sidebar, pengguna dapat memilih antara dua opsi: "Input Manual" dan "Unggah *File*".

```
def main():
    st.title("Prediksi Kelayakan karakter nasabah")

    # Navigasi halaman
    page = st.sidebar.selectbox("Pilih Halaman", ["Input Manual", "Unggah File"])

    if page == "Input Manual":
        st.header("Input Manual")

        # Input dari pengguna
        kriminalitas = st.selectbox('Tingkat Kriminalitas pada Lingkungan calon peminjam', ['Rendah', 'Sedang', 'Tinggi'])
        narkoba = st.selectbox('Riwayat Penggunaan Narkoba Calon Peminjam', ['Tidak', 'Iya'])
        terkena_kriminalitas_sebagai_korban = st.selectbox('Pernah terkena kriminalitas sebagai korban', ['Iya', 'Tidak'])
        Tujuan_Peminjaman = st.selectbox('Tujuan Peminjaman', ['Modal Kerja', 'Ekspansi Bisnis', 'Inovasi dan R&D',
                        'Pengembangan Teknologi', 'Pembelian Aset Tetap', 'Cadangan Likuiditas',
                        'Penyelamatan Bisnis', 'Konsolidasi Utang', 'Akuisisi atau Merger',
                        'Modal Ventura'])
        huru_hara = st.selectbox('Tingkat Huru hara pada Lingkungan calon peminjam', ['Rendah', 'Tinggi', 'Sedang'])
        situasi_tempat_tinggal_1 = st.selectbox('Situasi Tempat Tinggal calon peminjam', ['rawan bencana', 'kriminalitas tinggi', 'padat penduduk', 'area kumuh'])
        Jumlah_Hari_Tunggakan = st.selectbox('Histori hari Tunggakan kredit calon peminjam', ['1-29(hari)', '30-90(hari)', '30-60(hari)', '>60(hari)', '>90(hari)'])
        potensi_daerah = st.selectbox('Potensi daerah', ['Rendah', 'Sedang', 'Tinggi', 'Tidak Ada'])
        total_asset = st.number_input('Total asset calon peminjam', min_value=0)
        total_hutang = st.number_input('Total hutang calon peminjam', min_value=0)
        Penghasilan_setelah_dipotong_pajak = st.number_input('Penghasilan bersih calon peminjam', min_value=0)
        status_agunan_terburuk = st.number_input('status agunan terburuk', min_value=0)

        features = {
            'kriminalitas': kriminalitas,
            'narkoba': narkoba,
            'terkena_kriminalitas_sebagai_korban': terkena_kriminalitas_sebagai_korban,
            'Tujuan_Peminjaman': Tujuan_Peminjaman,
            'huru_hara': huru_hara,
            'situasi_tempat_tinggal_1': situasi_tempat_tinggal_1,
            'Jumlah_Hari_Tunggakan': Jumlah_Hari_Tunggakan,
            'potensi_daerah': potensi_daerah,
            'total_asset': total_asset,
            'total_hutang': total_hutang,
            'Penghasilan_setelah_dipotong_pajak': Penghasilan_setelah_dipotong_pajak,
            'status_agunan_terburuk': status_agunan_terburuk,
        }

        if st.button('Dapatkan Prediksi'):
            result = get_prediction(features)
            if result:
                st.write(f"**Prediksi**: {result.get('prediction', 'Tidak Diketahui')}")
                st.write(f"**Skor Prediksi**: {result.get('prediction_score', 'Tidak Diketahui')}")
                st.write(f"**Keterangan**: {result.get('keterangan', 'Tidak Diketahui')}")
```

Jika pengguna memilih "*Input Manual*", halaman akan menampilkan berbagai input untuk mengumpulkan data, seperti tingkat kriminalitas, riwayat penggunaan narkoba, tujuan peminjaman, dan faktor lainnya. Beberapa input menggunakan *selectbox* untuk pilihan kategori, sedangkan number_input digunakan untuk angka, seperti total aset atau hutang. Semua input ini disusun dalam bentuk *dictionary* bernama *features* yang berisi data yang dimasukkan pengguna.

Setelah pengguna mengisi data dan menekan tombol Dapatkan Klasifikasi, fungsi get_prediction(features) akan dipanggil untuk mengirimkan data ke API yang telah dibangun sebelumnya. API ini kemudian memberikan hasil klasifikasi, yang bisa mencakup kategori skor kredit dan penjelasan lainnya. Hasil klasifikasi tersebut akan ditampilkan langsung di halaman web, memungkinkan pengguna

untuk melihat apakah calon peminjam memiliki skor kredit yang baik, rata-rata, atau buruk. Fitur ini memberikan pengalaman interaktif dan langsung untuk memeriksa kelayakan kredit calon peminjam berdasarkan input yang diberikan.

4. Membangun tampilan halaman Unggah *File*

Gambar 3.46 menunjukkan tampilan program pada halaman "Unggah *File*", yang akan muncul ketika pengguna memilih opsi "Unggah *File*" melalui sidebar. Halaman ini dirancang untuk memudahkan pengguna dalam mengunggah *file* berformat CSV atau Excel (XLSX) langsung dari perangkat pribadi menggunakan komponen *file*_uploader. Setelah *file* berhasil dipilih dan diunggah, sistem akan memanggil fungsi predict_from_*file*(uploaded_*file*) untuk memproses data yang terdapat dalam *file* tersebut. Fungsi ini berperan penting dalam membaca setiap baris data dan melakukan klasifikasi secara otomatis, sehingga pengguna tidak perlu lagi memasukkan data satu per satu secara manual. Fitur ini sangat membantu bagi pengguna yang ingin memproses data dalam jumlah besar dengan lebih cepat dan efisien.

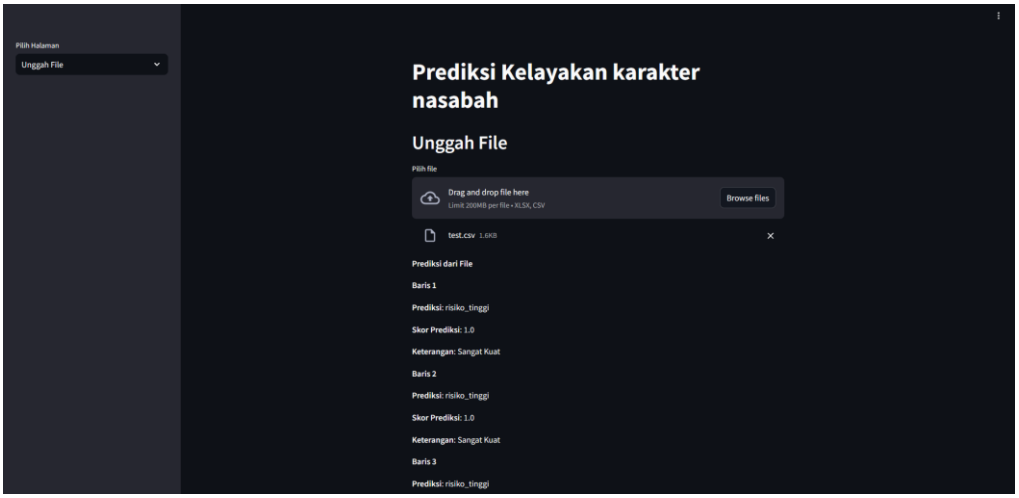
```
elif page == "Unggah File":
    st.header("Unggah File")
    uploaded_file = st.file_uploader("Pilih file", type=["xlsx", "csv"])

    if uploaded_file:
        predictions = predict_from_file(uploaded_file)

        if predictions:
            st.write("**Prediksi dari File**")
            for i, result in enumerate(predictions):
                st.write(f"**Baris {i+1}**")
                st.write(f"**Prediksi**: {result.get('prediction', 'Tidak Diketahui')}")
                st.write(f"**Skor Prediksi**: {result.get('prediction_score', 'Tidak Diketahui')}")
                st.write(f"**Keterangan**: {result.get('keterangan', 'Tidak Diketahui')}")
        else:
            st.error("Tidak ada prediksi yang dihasilkan.")
```

Gambar 3. 47 Code Halaman Unggah File

Jika proses klasifikasi berhasil, hasilnya akan ditampilkan dalam *page* unggah *file*. Setiap baris data akan diberikan informasi lengkap, meliputi klasifikasi hasil, skor klasifikasi, serta keterangan tambahan. Selain itu, hasil klasifikasi juga diberi nomor urut untuk memudahkan identifikasi setiap entri dalam *file*.

Apabila tidak ada hasil klasifikasi yang dapat diproses, program akan menampilkan pesan kesalahan melalui st.error, memberikan *feedback* yang jelas dan informatif kepada pengguna. Dengan fitur "Unggah *File*" ini, aplikasi menjadi lebih fleksibel dan mampu menangani volume data yang besar secara efisien,

menjadikannya solusi ideal bagi lembaga atau organisasi yang membutuhkan evaluasi kredit secara massal.



Gambar 3. 48 UI Halaman Unggah File

3.2.10 Melakukan perancangan Dashboard *Character credit assesment* menggunakan Power BI

Dashboard *Character Credit Assessment* dibuat menggunakan Power BI sebagai alat bantu untuk mengevaluasi data nasabah berdasarkan karakteristik penting yang mempengaruhi kelayakan kredit. *Dashboard* ini memiliki tujuan utama untuk memberikan gambaran komprehensif mengenai situasi lingkungan nasabah, kondisi finansial, serta riwayat kesehatan atau penyakit nasabah yang dapat berdampak signifikan terhadap kemampuan nasabah dalam memenuhi kewajiban kredit.

Dengan memanfaatkan fitur visualisasi dan analisis data yang interaktif dari Power BI, pihak evaluasi dapat dengan cepat mengidentifikasi pola, tren, dan potensi risiko dalam data yang dikumpulkan. Analisis berbasis karakter ini tidak hanya berfokus pada aspek finansial tetapi juga mencakup faktor non-finansial yang sering kali menjadi indikator penting dalam menentukan perilaku pembayaran

kredit nasabah. Oleh karena itu, penyusunan dashboard ini diharapkan dapat menjadi solusi efektif bagi lembaga keuangan dalam melakukan penilaian kredit yang lebih objektif, transparan, dan akurat, sehingga mendukung pengambilan keputusan yang lebih baik dan minim risiko.

Pemilihan Power BI dibandingkan Tableau pada perancangan *dashboard* didasarkan pada beberapa pertimbangan utama. Power BI menawarkan integrasi yang lebih baik dengan ekosistem Microsoft, seperti Excel, SharePoint, dan Azure, sehingga mempermudah alur kerja bagi organisasi yang sudah menggunakan layanan Microsoft. Selain itu, Power BI memiliki biaya lisensi yang lebih terjangkau dibandingkan Tableau, menjadikannya pilihan yang lebih ekonomis untuk organisasi kecil hingga menengah. Antarmuka Power BI yang intuitif dan kemudahan dalam membuat laporan interaktif juga menjadi nilai tambah, terutama bagi pengguna yang kurang berpengalaman dalam visualisasi data. Power BI menyediakan fitur *drag-and-drop* yang sederhana serta mendukung berbagai koneksi data, mulai dari basis data lokal hingga layanan *cloud*. Selain itu, kemampuan Power BI untuk melakukan pembaruan otomatis pada dasbor dan laporan memberikan fleksibilitas yang tinggi dalam pemantauan data secara real-time. Kombinasi antara kemudahan penggunaan, biaya yang efisien, dan kompatibilitas yang luas menjadikan Power BI solusi visualisasi data yang ideal bagi banyak organisasi. Berikut merupakan Tahap pembuatan *dashboard* ini akan mencakup beberapa langkah penting, antara lain:

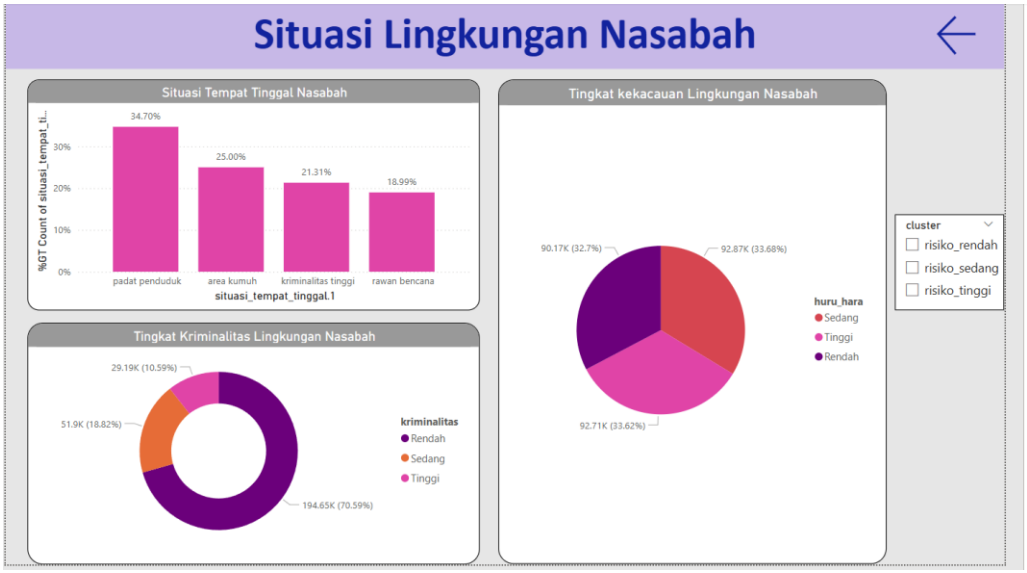
1. Membangun tampilan *Dashboard* Situasi Lingkungan Nasabah

Grafik pada Gambar 3.48 menampilkan informasi mengenai Situasi Lingkungan Nasabah, yang dipecah menjadi tiga bagian utama untuk memberikan wawasan mendalam terkait faktor lingkungan yang dapat mempengaruhi kelayakan kredit nasabah. Grafik batang di bagian kiri atas menunjukkan distribusi Situasi Tempat Tinggal Nasabah. Di sini, "padat penduduk" mendominasi dengan persentase sebesar 34,70%, diikuti oleh "area kumuh" sebesar 25,00%, "kriminalitas tinggi" sebesar 21,31%, dan "rawan bencana" sebesar 18,99%. Hal ini

mengindikasikan bahwa sebagian besar nasabah tinggal di area yang padat penduduk, yang dapat berpotensi mempengaruhi stabilitas ekonomi dan sosial.

Bagian kanan atas berfokus pada Tingkat Kekacauan Lingkungan Nasabah, yang ditampilkan dalam bentuk diagram lingkaran. Diagram ini memecah data menjadi tiga kategori berdasarkan tingkat "huru-hara": rendah, sedang, dan tinggi. Tingkat "sedang" memiliki porsi yang cukup besar dengan nilai 33,68% (92.87K), sedangkan kategori "tinggi" hampir mendekati dengan 33,62% (92.71K). Sementara itu, tingkat "rendah" menyumbang 32,7% (90.17K). Distribusi yang merata ini menunjukkan bahwa lingkungan nasabah cukup bervariasi, dengan banyak nasabah berada di area yang memiliki risiko kekacauan sedang hingga tinggi.

Bagian terakhir di sisi kiri bawah menampilkan Tingkat Kriminalitas Lingkungan Nasabah dalam bentuk diagram donat. Di sini, kategori rendah memiliki proporsi terbesar dengan 70,59% (194.65K), diikuti oleh tingkat sedang sebesar 18,82% (51.9K), dan tingkat tinggi sebesar 10,59% (29.19K). Grafik ini memperlihatkan bahwa mayoritas nasabah tinggal di lingkungan dengan tingkat kriminalitas rendah, namun terdapat sebagian kecil yang tinggal di area dengan risiko kriminalitas tinggi. Dengan demikian, dashboard ini membantu pihak penilai kredit memahami kondisi lingkungan nasabah secara keseluruhan, yang meliputi situasi tempat tinggal, tingkat kekacauan, dan tingkat kriminalitas, sehingga dapat menjadi faktor pertimbangan penting dalam menilai kelayakan kredit nasabah.

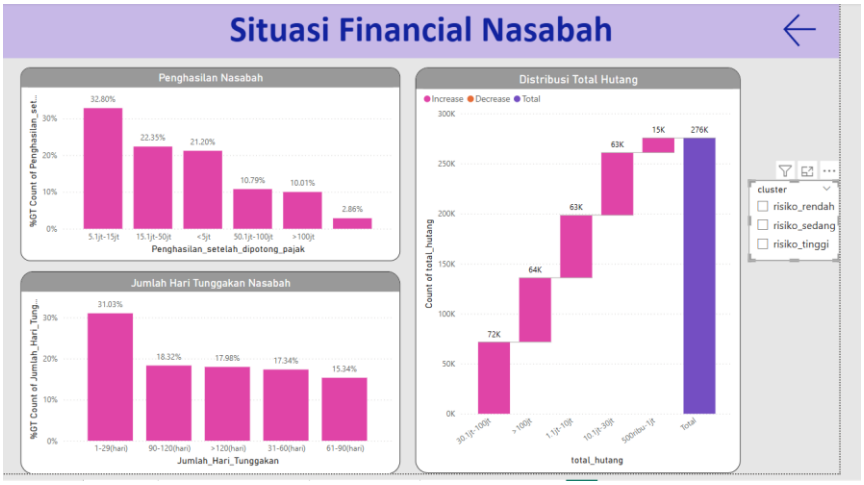


Gambar 3. 49 Dashboard Situasi Lingkungan Nasabah

2. Membangun tampilan *Dashboard* Situasi Financial Nasabah

Grafik pada Gambar 3.49 menunjukkan situasi finansial nasabah dengan tiga visualisasi utama yaitu Penghasilan Nasabah, Jumlah Hari Tunggakan Nasabah, dan Distribusi Total Hutang. Pada bagian Penghasilan Nasabah, terlihat bahwa mayoritas nasabah (32,80%) memiliki penghasilan bersih setelah pajak di rentang 5,1 juta - 15 juta rupiah, diikuti oleh rentang 15,1 juta - 50 juta rupiah sebesar 22,35%. Namun, terdapat kelompok kecil dengan penghasilan sangat rendah di bawah 5 juta rupiah (21,20%) dan di atas 100 juta rupiah hanya 2,86%. Hal ini mengindikasikan bahwa sebagian besar nasabah berada dalam kelompok penghasilan menengah. Untuk Jumlah Hari Tunggakan Nasabah, sebagian besar nasabah memiliki keterlambatan pembayaran antara 1-29 hari (31,03%). Kelompok keterlambatan yang lebih signifikan adalah nasabah dengan tunggakan lebih dari 120 hari (17,98%) dan 90-120 hari (18,32%). Kondisi ini menunjukkan adanya risiko potensi kredit macet pada nasabah yang mengalami keterlambatan lama, meskipun sebagian besar keterlambatan masih dalam periode yang lebih singkat dan dapat dikategorikan sebagai risiko sedang.Terakhir, pada Distribusi Total Hutang, terlihat kontribusi jumlah hutang terbesar berasal dari rentang 500 ribu - 1

juta rupiah (276K), disusul oleh kelompok 10 juta - 30 juta rupiah (63K). Namun, hutang dalam rentang 30 juta - 100 juta rupiah terlihat memiliki nilai cukup rendah (72K).



Gambar 3. 50 Dashboard Situasi Financial Nasabah

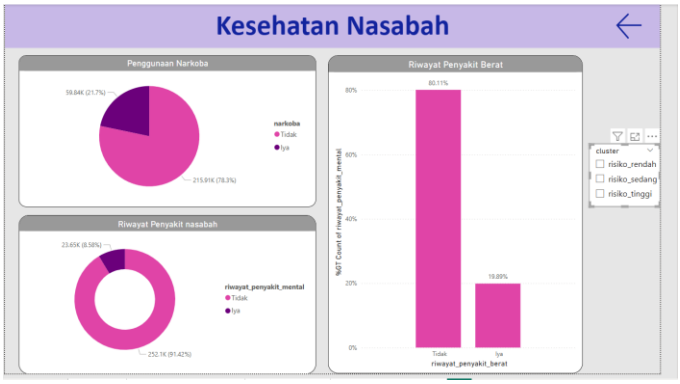
3. Membangun tampilan *Dashboard* Situasi Kesehatan Nasabah

Grafik pada Gambar 3.50 menunjukkan visualisasi Kesehatan Nasabah dengan tiga visualisasi utama. Pada bagian Penggunaan Narkoba, sebanyak 78,3% atau 215,91 ribu nasabah tidak menggunakan narkoba, sementara 21,7% atau 59,84 ribu nasabah tercatat sebagai pengguna narkoba. Meskipun persentase pengguna narkoba lebih kecil, angka tersebut tetap signifikan dan memerlukan perhatian, terutama untuk mitigasi risiko kredit terkait faktor kesehatan.

Selanjutnya, pada bagian Riwayat Penyakit Mental, mayoritas nasabah, yaitu 91,42% atau 252,1 ribu nasabah, tidak memiliki riwayat penyakit mental, sedangkan 8,58% atau 23,65 ribu nasabah tercatat memiliki riwayat penyakit mental. Angka ini menunjukkan bahwa meskipun proporsi nasabah dengan riwayat penyakit mental kecil, nasabah tersebut tetap perlu mendapatkan perhatian khusus untuk memastikan kondisi kesehatan mental tidak memengaruhi kemampuan pembayaran kredit.

Pada bagian Riwayat Penyakit Berat, sebanyak 80,11% nasabah tidak memiliki riwayat penyakit berat, sedangkan 19,89% nasabah atau sekitar seperlima dari total populasi memiliki riwayat penyakit berat. Hal ini menunjukkan bahwa

penyakit berat masih menjadi faktor signifikan yang dapat memengaruhi stabilitas finansial nasabah. Dengan demikian, nasabah dengan riwayat penyakit berat atau penyakit mental memerlukan evaluasi risiko yang lebih mendalam guna memastikan profil risiko kesehatan nasabah tidak berdampak negatif terhadap pengelolaan kredit.



Gambar 3. 51 Dashboard Situasi Kesehatan Nasabah

3.3 Kendala yang Ditemukan

Selama menjalani program magang sebagai *Data Scientist* di PT. Menara Indonesia, saya menghadapi beberapa tantangan sebagai seorang pemula yang baru terjun langsung ke dunia kerja. Tantangan-tantangan tersebut meliputi:

1. Keterbatasan pengalaman dalam membangun model berbasis data dalam lingkungan kerja profesional.
2. Kurangnya deskripsi yang jelas terhadap setiap variabel dalam dataset *credit assessment* menjadi hambatan utama yang mengurangi efektivitas pengembangan model.
3. Keterbatasan komunikasi selama magang terjadi karena sistem kerja hybrid

3.4 Solusi atas Kendala yang Ditemukan

Berikut merupakan solusi atas kendala yang telah dihadapi pada pelaksanaan program kerja magang menjadi *Data Scientist* di PT. Menara Indonesia

1. Mencari referensi pembelajaran dari sumber eksternal seperti dari jurnal, *search engine*, dan youtube.

2. Tim dari divisi *Data Scientist* akan mengonfirmasi data yang belum diberikan atau yang perlu diperbarui kepada mentor, lalu atau divisi lain yang bertanggung jawab atas data tersebut.
3. Menjadwalkan diskusi rutin melalui komunikasi online menggunakan platform seperti Zoom dan Discord, serta terus menjaga komunikasi melalui grup WhatsApp.